Resultssets to resultstables revisited

Roger B. Newson
r.newson@qmul.ac.uk
http://www.rogernewsonresources.org.uk

Cancer Prevention Group, Wolfson Institute of Population Health, Queen Mary University of London

Presented at the 2025 UK Stata Conference, London,
11–12 September, 2025
Downloadable from the conference website at
https://econpapers.repec.org/paper/boc/lsug25/

Resultssets revisited

- A **resultsset** is a Stata dataset created as output by a Stata program.
- They are nowadays created in **resultsframes**[1], but can also be listed, written to a file, or overwritten over the input dataset.
- Resultsset–generating SSC packages include `parmest`, `xcollapse`, `xcontract`, `descsave`, `xdir`, `xframedir`, and `xsvmat`.
- And, like other Stata datasets, resultssets can be input into “SQL–like” operations, using `append`, `merge`, `joinby`, and `cross` in official Stata, or the SSC packages `keyby`, `addinby`, `expgen`, and `xframeappend`, to output **secondary resultssets**.

Resultssets revisited

- A **resultsset** is a Stata dataset created as output by a Stata program.
- They are nowadays created in **resultsframes**[1], but can also be listed, written to a file, or overwritten over the input dataset.
- Resultsset–generating SSC packages include `parmest`, `xcollapse`, `xcontract`, `descsave`, `xdir`, `xframedir`, and `xsvmat`.
- And, like other Stata datasets, resultssets can be input into "SQL–like" operations, using `append`, `merge`, `joinby`, and `cross` in official Stata, or the SSC packages `keyby`, `addinby`, `expgen`, and `xframeappend`, to output **secondary resultssets**.

Resultssets revisited

- A **resultsset** is a Stata dataset created as output by a Stata program.
- They are nowadays created in **resultsframes**[1], but can also be listed, written to a file, or overwritten over the input dataset.
- Resultsset–generating SSC packages include `parmest`, `xcollapse`, `xcontract`, `descsave`, `xdir`, `xframedir`, and `xsvmat`.
- And, like other Stata datasets, resultssets can be input into "SQL–like" operations, using `append`, `merge`, `joinby`, and `cross` in official Stata, or the SSC packages `keyby`, `addinby`, `expgen`, and `xframeappend`, to output **secondary resultssets**.

Resultssets revisited

- A **resultsset** is a Stata dataset created as output by a Stata program.
- They are nowadays created in **resultsframes**[1], but can also be listed, written to a file, or overwritten over the input dataset.
- Resultsset–generating SSC packages include `parmest`, `xcollapse`, `xcontract`, `descsave`, `xdir`, `xframedir`, and `xsvmat`.
- And, like other Stata datasets, resultssets can be input into "SQL–like" operations, using `append`, `merge`, `joinby`, and `cross` in official Stata, or the SSC packages `keyby`, `addinby`, `expgen`, and `xframeappend`, to output **secondary resultssets**.

Resultssets revisited

- A **resultsset** is a Stata dataset created as output by a Stata program.
- They are nowadays created in **resultsframes**[1], but can also be listed, written to a file, or overwritten over the input dataset.
- Resultsset–generating SSC packages include `parmest`, `xcollapse`, `xcontract`, `descsave`, `xdir`, `xframedir`, and `xsvmat`.
- And, like other Stata datasets, resultssets can be input into "SQL–like" operations, using `append`, `merge`, `joinby`, and `cross` in official Stata, or the SSC packages `keyby`, `addinby`, `expgen`, and `xframeappend`, to output **secondary resultssets**.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or **resultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or **resultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or r**esultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or r**esultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or r**esultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

The resultsset–central dogma: datasets make resultssets make resultsplots and/or resultstables[2]

- Statisticians make their living producing **resultsplots** and/or r**esultstables**.
- And a string variable needs to be encoded to numeric in order to be plotted.
- And a numeric variable needs to be decoded to string in order to be tabulated.
- Resultssets (unlike Stata tables and graphs) are therefore a sensible **common currency** for results, as their variables can be used equally to make resultsplots and/or resultstables. encoding and/or decoding when necessary.
- SSC packages used include `sencode`[3], `factext`, and `fvregen` for encoding, and `sdecode` and its family of dependents `bmjcip`, `factmerg`, `ingap`, and `insingap` for decoding.

Example in `example1.do`: Statistics for quantitative variables by US origin in the `xauto` data

- The SSC package `xauto` creates an extended version of the `auto` data supplied with official Stata.
- We will use it to generate a secondary `xcollapse` resultsset, containing statistics on the list of 10 quantitative variables `price npm rep78 trunk headroom tons length turn displacement gear_ratio`, broken down by origin of car model (US or non–US).
- We then convert the resultsset to a multi–page resultstable in a `.docx` document `example1.docx`.

Example in `example1.do`: Statistics for quantitative variables by US origin in the `xauto` data

- The SSC package `xauto` creates an extended version of the `auto` data supplied with official Stata.
- We will use it to generate a secondary `xcollapse` resultsset, containing statistics on the list of 10 quantitative variables `price npm rep78 trunk headroom tons length turn displacement gear_ratio`, broken down by origin of car model (US or non–US).
- We then convert the resultsset to a multi–page resultstable in a `.docx` document `example1.docx`.

Example in `example1.do`: Statistics for quantitative variables by US origin in the `xauto` data

- The SSC package `xauto` creates an extended version of the `auto` data supplied with official Stata.
- We will use it to generate a secondary `xcollapse` resultsset, containing statistics on the list of 10 quantitative variables `price npm rep78 trunk headroom tons length turn displacement gear_ratio`, broken down by origin of car model (US or non–US).
- We then convert the resultsset to a multi–page resultstable in a `.docx` document `example1.docx`.

Example in **`example1.do`**: Statistics for quantitative variables by US origin in the **`xauto`** data

- The SSC package `xauto` creates an extended version of the `auto` data supplied with official Stata.
- We will use it to generate a secondary `xcollapse` resultsset, containing statistics on the list of 10 quantitative variables `price npm rep78 trunk headroom tons length turn displacement gear_ratio`, broken down by origin of car model (US or non–US).
- We then convert the resultsset to a multi–page resultstable in a `.docx` document `example1.docx`.

The secondary resultsset to be converted

This was created by `xframeappend`ing 10 `xcollapse` resultsframes, one for each quantitative variable. We then `sencode`d the string ID variable `idstr` to create the variable `quanvar`:

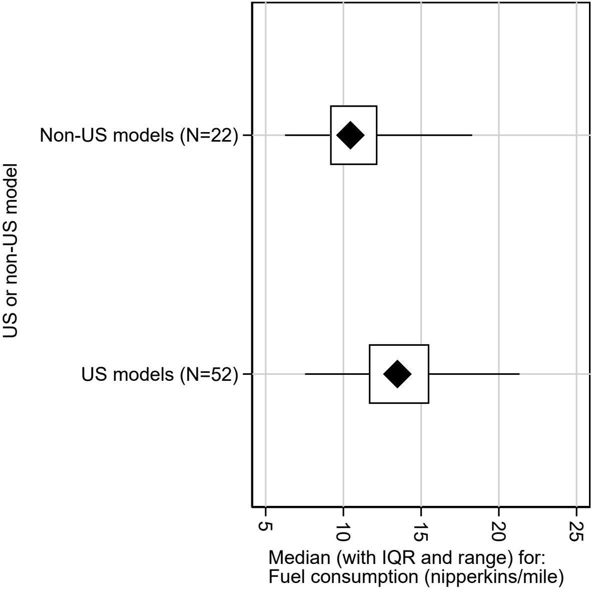
```
. desc, fu;

Contains data
 Observations:            20
    Variables:            10
---------------------------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
---------------------------------------------------------------------------------------------------
quanvar         byte    %-34.0g    quanvar    Quantitative variable
us              byte    %-8.0g     us         US or non-US model
N               byte    %8.0g                 (count) X
mean            float   %8.2f                 (mean) X
sd              float   %8.2f                 (sd) X
p0              float   %8.2f                 (min) X
p25             float   %8.2f                 (p 25) X
p50             float   %8.2f                 (p 50) X
p75             float   %8.2f                 (p 75) X
p100            float   %8.2f                 (max) X
---------------------------------------------------------------------------------------------------
Sorted by: quanvar  us
     Note: Dataset has changed since last saved.
```

We see that the dataset has 1 observation per quantitative variable per car model origin group (non–US or US), and data on statistics.

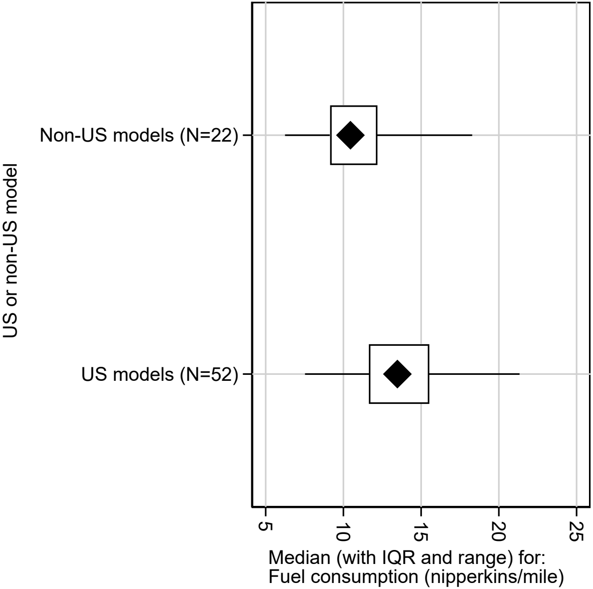
A resultsplot from our resultsset

- This plot was produced from our resultsset, using the SSC packages `sdecode`, `sencode`, and `eclplot`.
- And there are many other things we can do with resultssets!
- *However*, today we concentrate on multi–page tables in `.docx` documents, which clinical trial committees like.



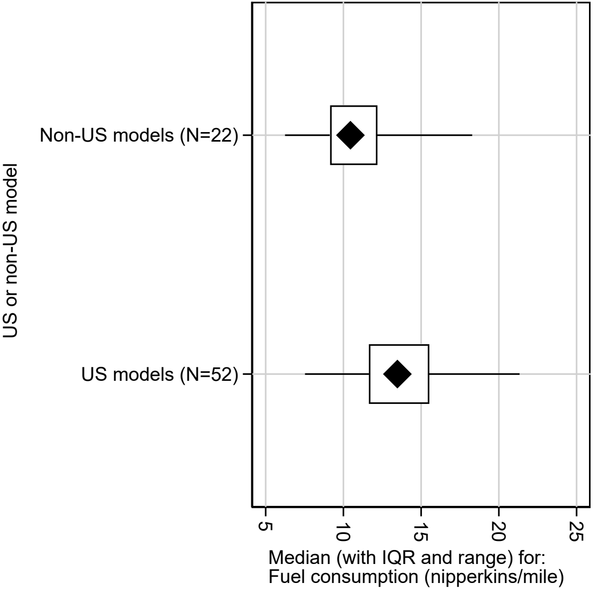
A resultsplot from our resultsset

- This plot was produced from our resultsset, using the SSC packages `sdecode`, `sencode`, and `eclplot`.
- And there are many other things we can do with resultssets!
- *However*, today we concentrate on multi–page tables in `.docx` documents, which clinical trial committees like.



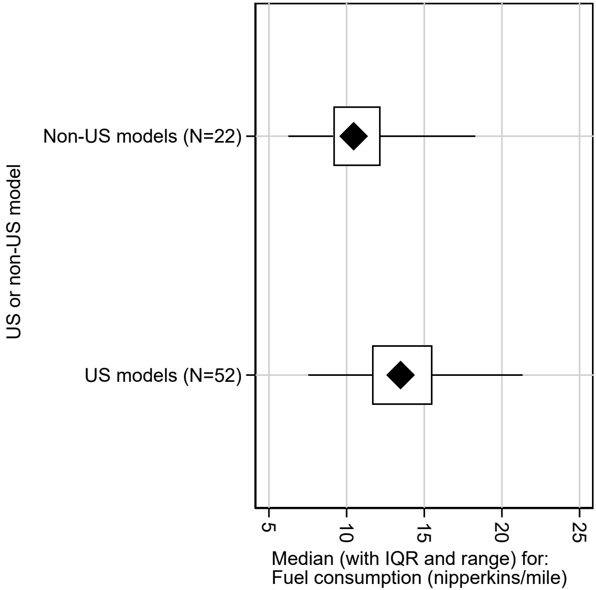
A resultsplot from our resultsset

- This plot was produced from our resultsset, using the SSC packages `sdecode`, `sencode`, and `eclplot`.
- And there are many other things we can do with resultssets!
- *However*, today we concentrate on multi–page tables in `.docx` documents, which clinical trial committees like.



A resultsplot from our resultsset

- This plot was produced from our resultsset, using the SSC packages `sdecode`, `sencode`, and `eclplot`.
- And there are many other things we can do with resultssets!
- *However*, today we concentrate on multi–page tables in `.docx` documents, which clinical trial committees like.



Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Resultssets to resultstables: decoding, listing and other steps

- Converting resultssets to resultstables has previously been discussed in Newson (2012)[4] and Newson (2023)[5].
- The process *usually* starts with decoding, using the `sdecode` family of SSC packages.
- And it *always* ends with listing, using the SSC packages `docxtab` (for tables in `.docx` documents) or `listtab` (for tables in Markdown, HTML, LaTeX, plain TeX, or `.rtf` documents).
- *However*, there may be other steps between decoding and listing, involving `reshape`ing (`long` or `wide`), appending, merging, characterizing (to define table–column headers), inserting gap observations, and/or grouping rows into pages in multi–page tables.
- These steps convert a resultsset (with a primary key and 1 observation per result) to a dataset ready for listing (with a primary key and 1 observation per table row).

Steps in converting a resultsset to a resultstable

These 11 steps are given in the order in which they *usually* happen. There are SSC modules for each step.

Step type	*SSC modules used*	*Importance*
Decode non–key variables to table cells	`sdecode` and dependents	Semi–compulsory
Reshape to long	`xrelong`	Optional
Append extra table rows	`xframeappend`, `factmerg`	Optional
Characterize table columns	`chardef`, `xrewide`	Optional
Reshape to wide	`xrewide`	Optional
Merge in extra table columns	`addinby`, `fraddinby`	Optional
Decode key variables to table row label	`sdecode` and dependents	Semi–compulsory
Characterize table row label	`chardef`	Optional
Insert gap observations	`insingap`, `ingap`	Optional
Group observations into pages	`ltop`	Optional
List table	`listtab`, `docxtab`	Compulsory

The “Compulsory” step (listing) is always necessary. The 2 “Semi–compulsory” steps (decoding) are *nearly* always necessary. The “Optional” steps are *frequently* absent (because, fortunately, *most* tables are simple). To find out more about the SSC modules, use `findit` in Stata.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **“vector–statistic”**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **"vector–statistic"**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **"vector–statistic"**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **“vector–statistic”**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **“vector–statistic”**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

Example: Decode and reshape to long

- We start making our resultstable by decoding our statistics variables.
- This is done using the `msdecode` module of the `sdecode` package, which can input multiple numeric statistics variables to output a string variable displaying a decoded **“vector–statistic”**, like a variable range in parentheses.
- This creates new string variables `stat1`, `stat2`, `stat3`, and `stat4`, displaying, respectively, the sample number, the mean (with SD), the median (with IQR), and the range.
- We then use the module `xrelong`, an extension of `reshape long`, which creates a long version of our resultsset, with an extra labelled key variable `statseq` and a single displayed statistic value variable `stat`.
- This gives us a dataset with 1 observation per quantitative variable per car–origin group per displayed statistic, and data on the values of those statistics.

The code for decoding and reshaping to long

The code to do this was as follows:

```
msdecode N, gene(stat1);
msdecode mean sd, delim(" (") suff(")") gene(stat2);
msdecode p50 p25 p75, delim(" (" ", ") suff(")")
  gene(stat3);
msdecode p0 p100, pref("(") delim(", ") suff(")")
  gene(stat4);
lab def statseq 1 "N" 2 "Mean (SD)" 3 "Median (IQR)"
  4 "Range";
drop N mean sd p*;
xrelong stat, i(quanvar us) j(statseq) jlabel(statseq);
jformat statseq stat;
lab var statseq "Statistic sequence";
lab var stat "Statistic value";
desc, fu;
```

We start by using `msdecode` to decode our 8 numeric statistics to 4 string variables, drop the numeric variables, and use `xrelong`, with the option `jlabel(statseq)`, to reshape the dataset to long (with labelled *j*–values). The SSC package `jformat` left–justifies the new variables.

The resultssetset decoded and reshaped to long

We `list`ed the new long dataset:

```
. by quanvar: list us statseq stat, abbr(32) sepby(quanvar us);

--------------------------------------------------------------------------------------------
-> quanvar = Price

     +------------------------------------------------------+
     | us       statseq         stat                        |
     |------------------------------------------------------|
  1. | Non-US   N               22                          |
  2. | Non-US   Mean (SD)       6384.68 (2621.92)           |
  3. | Non-US   Median (IQR)    5759.00 (4499.00, 7140.00)  |
  4. | Non-US   Range           (3748.00, 12990.00)         |
     |------------------------------------------------------|
  5. | US       N               52                          |
  6. | US       Mean (SD)       6072.42 (3097.10)           |
  7. | US       Median (IQR)    4782.50 (4184.00, 6234.00)  |
  8. | US       Range           (3291.00, 15906.00)         |
     +------------------------------------------------------+

--------------------------------------------------------------------------------------------
-> quanvar = Fuel consumption (nipperkins/mile)

     +------------------------------------------------+
     | us       statseq         stat                  |
     |------------------------------------------------|
  1. | Non-US   N               22                    |
  2. | Non-US   Mean (SD)       11.04 (2.93)          |
  3. | Non-US   Median (IQR)    10.45 (9.14, 12.19)   |
  4. | Non-US   Range           (6.24, 18.29)         |
```

The long format allows dissimilar vector–statistics to be stacked.

Example: Reshaping to wide and adding gap rows

- We continued by using `xrewide` (an extension of `reshape wide`), with the options `i(quanvar statseq) j(us) cjlabel(varname)`, to create a dataset with 1 observation per quantitative variable per output vector–statistic, and data on that statistic in non–US and US models (side by side).
- We then created a string row label variable `rowlabel` by `sdecode`ing `statseq`.
- We then inserted gap observations using `insingap`, adding a gap observation at the start of each quantitative variable.
- This creates a dataset with 5 observations per quantitative variable, the first a gap observation and the other 4 containing data on the 4 vector–statistics in non–US and US models.

Example: Reshaping to wide and adding gap rows

- We continued by using `xrewide` (an extension of `reshape wide`), with the options `i(quanvar statseq) j(us) cjlabel(varname)`, to create a dataset with 1 observation per quantitative variable per output vector–statistic, and data on that statistic in non–US and US models (side by side).
- We then created a string row label variable `rowlabel` by `sdecode`ing `statseq`.
- We then inserted gap observations using `insingap`, adding a gap observation at the start of each quantitative variable.
- This creates a dataset with 5 observations per quantitative variable, the first a gap observation and the other 4 containing data on the 4 vector–statistics in non–US and US models.

Example: Reshaping to wide and adding gap rows

- We continued by using `xrewide` (an extension of `reshape wide`), with the options `i(quanvar statseq) j(us) cjlabel(varname)`, to create a dataset with 1 observation per quantitative variable per output vector–statistic, and data on that statistic in non–US and US models (side by side).
- We then created a string row label variable `rowlabel` by `sdecode`ing `statseq`.
- We then inserted gap observations using `insingap`, adding a gap observation at the start of each quantitative variable.
- This creates a dataset with 5 observations per quantitative variable, the first a gap observation and the other 4 containing data on the 4 vector–statistics in non–US and US models.

Example: Reshaping to wide and adding gap rows

- We continued by using `xrewide` (an extension of `reshape wide`), with the options `i(quanvar statseq) j(us) cjlabel(varname)`, to create a dataset with 1 observation per quantitative variable per output vector–statistic, and data on that statistic in non–US and US models (side by side).
- We then created a string row label variable `rowlabel` by `sdecode`ing `statseq`.
- We then inserted gap observations using `insingap`, adding a gap observation at the start of each quantitative variable.
- This creates a dataset with 5 observations per quantitative variable, the first a gap observation and the other 4 containing data on the 4 vector–statistics in non–US and US models.

Example: Reshaping to wide and adding gap rows

- We continued by using `xrewide` (an extension of `reshape wide`), with the options `i(quanvar statseq) j(us) cjlabel(varname)`, to create a dataset with 1 observation per quantitative variable per output vector–statistic, and data on that statistic in non–US and US models (side by side).
- We then created a string row label variable `rowlabel` by `sdecode`ing `statseq`.
- We then inserted gap observations using `insingap`, adding a gap observation at the start of each quantitative variable.
- This creates a dataset with 5 observations per quantitative variable, the first a gap observation and the other 4 containing data on the 4 vector–statistics in non–US and US models.

The dataset reshaped to wide with added gap rows

The new dataset, when `list`ed, started like this:

```
. list rowlabel stat0 stat1, abbr(32) subvar sepby(quanvar) clean noobs;

                  Quantitative variable   Non-US                       US
                                 Price:
                                      N   22                           52
                              Mean (SD)   6384.68 (2621.92)            6072.42 (3097.10)
                           Median (IQR)   5759.00 (4499.00, 7140.00)   4782.50 (4184.00, 6234
                                  Range   (3748.00, 12990.00)          (3291.00, 15906.00)
    Fuel consumption (nipperkins/mile):
                                      N   22                           52
                              Mean (SD)   11.04 (2.93)                 13.61 (3.13)
                           Median (IQR)   10.45 (9.14, 12.19)          13.47 (11.64, 15.53)
                                  Range   (6.24, 18.29)                (7.53, 21.33)
                    Repair record 1978:
                                      N   21                           48
                              Mean (SD)   4.29 (0.72)                  3.02 (0.84)
                           Median (IQR)   4.00 (4.00, 5.00)            3.00 (3.00, 3.00)
                                  Range   (3.00, 5.00)                 (1.00, 5.00)
                 Trunk space (cu. ft.):
                                      N   22                           52
                              Mean (SD)   11.41 (3.22)                 14.75 (4.31)
                           Median (IQR)   11.00 (9.00, 14.00)          16.00 (11.00, 17.00)
                                  Range   (5.00, 16.00)                (7.00, 23.00)
```

This looks *a bit* more like a resultstable! *However*. . .

Grouping table rows into pages using `ltop`

- … there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` ("lines to pages") creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Grouping table rows into pages using `ltop`

- ... there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` ("lines to pages") creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Grouping table rows into pages using `ltop`

- . . . there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` (“lines to pages”) creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Grouping table rows into pages using `ltop`

- ... there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` ("lines to pages") creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Grouping table rows into pages using `ltop`

- . . . there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` (“lines to pages”) creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Grouping table rows into pages using `ltop`

- . . . there are 5 observations (including gap observations) for each of 10 quantitative variables. These 50 rows might be too many for one page of our A4 `.docx` output!
- Fortunately, the SSC package `ltop` (“lines to pages”) creates a page sequence variable, grouping table rows into pages.
- `ltop` has an option `maxlperp(#)`, specifying the maximum lines per page.
- It has an option `iby(`*`varlist`*`)`, specifying internal by–groups that must not be split between pages.
- And it can have a `weight` expression, specifying that some table rows might be taller than others.

Example: Grouping rows into pages

We use `ltop` to create a new page sequence variable `pageseq`, with maximum lines per page set by `maxlperp(40)`, inner by–groups corresponding to values of `quanvar`, and weights equal to `gapobs+1`, where `gapobs` is a binary indicator that an observation is a gap row. We then use `xcontract` to display numbers of rows on each page:

```
. ltop pageseq [weight=gapobs+1], iby(quanvar)
>   maxlperp(40);
(frequency weights assumed)

. xcontract pageseq, list(, abbr(32));


     +-----------------------------+
     | pageseq   _freq   _percent |
     |-----------------------------|
  1. |       1      30      60.00 |
  2. |       2      20      40.00 |
     +-----------------------------+
```

We see that 30 table rows are on Page 1 and that 20 are on Page 2. Note that the weights allow a gap row to be twice as tall as other rows.

Making the final .docx document

- We now have a dataset with 1 observation per table row, with the rows grouped into pages.
- *So*, we can now write a document–generating section to write that dataset to a document `example1.docx`, looping over pages and creating a multi–page "Table XYZ".
- We can now have a look at our new document.

Making the final `.docx` document

- We now have a dataset with 1 observation per table row, with the rows grouped into pages.
- *So*, we can now write a document–generating section to write that dataset to a document `example1.docx`, looping over pages and creating a multi–page “Table XYZ”.
- We can now have a look at our new document.

Making the final `.docx` document

- We now have a dataset with 1 observation per table row, with the rows grouped into pages.
- *So*, we can now write a document–generating section to write that dataset to a document `example1.docx`, looping over pages and creating a multi–page “Table XYZ”.
- We can now have a look at our new document.

Making the final `.docx` document

- We now have a dataset with 1 observation per table row, with the rows grouped into pages.
- *So*, we can now write a document–generating section to write that dataset to a document `example1.docx`, looping over pages and creating a multi–page “Table XYZ”.
- We can now have a look at our new document.

References

[1] Newson, R. B. Resultssets in resultsframes in Stata 16–plus. Presented at the 2022 London Stata Conference, 8–9 September, 2022. . Downloadable from *http://ideas.repec.org/p/boc/lsug22/01.html*

[2] Newson, R. Resultssets, resultsspreadsheets, and resultsplots in Stata. Presented at the 2006 German Stata User Meeting, 31 March, 2006. . Downloadable from *http://ideas.repec.org/p/boc/dsug06/01.html*

[3] Newson, R. B. Creating factor variables in resultssets and other datasets. Presented at the 19th UK Stata User Meeting, 12–13 September, 2013. Downloadable fron *https://ideas.repec.org/p/boc/usug13/01.html*

[4] Newson, R. B. 2012. From resultssets to resultstables in Stata. *The Stata Journal* **12(2)**: 191–213. Downloadable from *https://journals.sagepub.com/doi/pdf/10.1177/1536867X1201200203*

[5] Newson, R. B. Customized Markdown and `.docx` tables using `listtab` and `docxtab`. Presented at the 2023 London Stata Conference, 7–8 September, 2023. Downloadable from *https://econpapers.repec.org/paper/boclsug23/01.htm*

The presentation, and the example do–file, can be downloaded from the conference website. The packages can be downloaded from SSC.