Resultssets in resultsframes in Stata 16–plus

Roger B. Newson
roger.newson@kcl.ac.uk
http://www.rogernewsonresources.org.uk

Cancer Prevention Group, King's College London

Presented at the 2022 UK Stata Conference, London,
8–9 September, 2022
Downloadable from the conference website at
http://ideas.repec.org/s/boc/usug22.html

Recap on resultssets

- A **resultsset**[1] is a Stata dataset created as output by a Stata program.
- It can be listed and/or saved in a disk file and/or written over an existing dataset in memory.
- And, in Stata 16 and higher, it can be written to a newly–created **data frame**, co–existing in memory with other data frames.

Recap on resultssets

- A **resultsset**[1] is a Stata dataset created as output by a Stata program.
- It can be listed and/or saved in a disk file and/or written over an existing dataset in memory.
- And, in Stata 16 and higher, it can be written to a newly–created **data frame**, co–existing in memory with other data frames.

Recap on resultssets

- A **resultsset**[1] is a Stata dataset created as output by a Stata program.
- It can be listed and/or saved in a disk file and/or written over an existing dataset in memory.
- And, in Stata 16 and higher, it can be written to a newly–created **data frame**, co–existing in memory with other data frames.

Recap on resultssets

- A **resultsset**[1] is a Stata dataset created as output by a Stata program.
- It can be listed and/or saved in a disk file and/or written over an existing dataset in memory.
- And, in Stata 16 and higher, it can be written to a newly–created **data frame**, co–existing in memory with other data frames.

Recap on resultsset–generating programs

These ado–files on SSC create resultssets with one observation per *thing* and data on *attributes_of_things*.

Module:	*Creates a resultsset with 1 observation per:*	*And data on:*
`parmest`	parameter	parameter attributes
`parmby`	by–group per parameter	parameter attributes
`metaparm`	meta–parameter	parameter attributes
`xcollapse`	by–group	by–group statistics
`xcontract`	value combination	frequencies and percents
`descsave`	variable	variable attributes
`xdir`	file	file attributes
`xframedir`	frame	frame attributes
`xsvmat`	matrix row	row attributes and values

All these resultssets can be listed and/or overwritten over the current dataset and/or saved to a disk file and/or saved to a newly created **resultsframe**. The resultssets can then be used as input to create **resultsplots**, or **resultstables** in a breathtaking variety of formats (eg HTML, Markdown, TeX, RTF, or even `.docx`). To find out more about these modules, use `findit` in Stata.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*`framename`*`, [replace change])`.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*`framename`*`, [replace change])`.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*`framename`*`, [replace change])`.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*`framename`*`, [replace change])`.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*`framename`*`, [replace change])`.

So what do resultsframes add?

- In Stata Versions 1 to 15, to alternate between datasets in memory, users had to save them to disk and read them in again, usually using `preserve` and `restore`.
- In Stata Versions 16 or higher, multiple datasets can live in multiple data frames, which can co–exist in the memory at the same time.
- Resultsframes are among the most useful data frames, because a gigabyte–sized Big Dataset can produce multiple resultssets containing only kilobytes.
- Thanks to resultsframes, we can modify and/or append and/or merge these resultssets, and then plot them and/or tabulate them and/or save them to disk, and then return to the original dataset in the default frame.
- To do this, resultsset–generating programs now have the option `frame(`*framename*`, [replace change])`.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

So what can we do with multiple resultsframes?

- Multiple resultsframes are frequently **appended** or **merged**.
- The SSC packages `frameappend` and `xframeappend` append single or multiple data frames, respectively, to the current data frame.
- The SSC package `addinby` was written as a wrapper for `merge m:1`, to merge new variables into a dataset from a second dataset, using a **foreign key** of variables.
- `addinby` now has a second module `fraddinby`, which merges new variables into the current data *frame* from a second data *frame*, again using a foreign key.
- We will demonstrate the use of `xframeappend` and `fraddinby` with multiple resultsframes.
- And, instead of a gigabyte-sized dataset, we will use the SSC package `xauto` to generate an extended version of the `auto` data.

Example 1: Using `xframeappend` to append multiple resultsframes

In the `xauto` data, we start by creating an empty frame `frankie`, and then loop over 4 variables, whose means we want to estimate. For each variable, we use `regress` to compute confidence intervals. and use `parmest` with the `ylabel` option to create a resultsframe `frieda`, with 1 observation containing a confidence interval for the mean. We then use `xframeappend` to append the frame `frieda` to the frame `frankie`. The code to do this is as follows:

```
frame create frankie;
foreach Y of var tons npm trunk price {;
  regress ‘Y’, vce(robust);
  parmest, ylabel format(estimate min* max* %8.2f)
    frame(frieda, replace);
  frame frankie: xframeappend frieda, drop;
};
```

This code produces output for 4 regressions (which we have omitted), and also a resultsframe `frankie`, with 1 observation per variable, and data on the variable’s mean and its confidence limits.

Listing the appended resultsframe `frankie`

We then list the most important variables in the frame `frankie`, including the variable `ylabel` containing the variable label of the *Y*–variable, and the estimates and their confidence limits:

```
. frame frankie: list ylabel estimate min* max*, clean noobs;

                           ylabel   estimate     min95     max95
                Weight (US tons)       1.51      1.42      1.60
  Fuel consumption (nipperkins/mile)  12.85     12.09     13.61
              Trunk space (cu. ft.)   13.76     12.77     14.75
                            Price   6165.26   5481.91   6848.60
```

This resultsframe can then be used to produce a resultstable in any one of a variety of formats, typically using. . .

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- ...the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

From resultsframe to resultstable using `listtab`

- …the SSC package `listtab`[2], which inputs a dataset and outputs a data table (to the log or to a file).
- This table can be in any one of a variety of **row styles**, identified by the string options `begin()`, `end()`, and `delimiter()`.
- Combinations of these string options are specified by the row style option `rstyle()`.
- Row style values include `html` and `markdown` for HTML tables, `tabular`, `halign` and `settabs` for TEX tables, and even `tabdelim` for tab–delimited tables, which can be pasted into a Microsoft Excel worksheet.
- There are also options like `headlines()`, `footlines()`, `headchars()`, and `footchars()` to specify header and footer rows for the tables.
- *So*, `listtab` outputs most generic table formats known to science, and others yet to be invented.

Converting the resultsframe **frankie** to a resultstable

In the frame `frankie`, we use `listtab`, with the option `rstyle(tabular)`, to output the estimates and confidence limits to the log in an alien–looking format, which LaTeX users will recognise as a LaTeX `tabular` environment:

```
. frame frankie: listtab ylabel estimate min* max*, rstyle(tabular) type
>   head(
>     "\begin{tabular}{rrrr}"
>     "\textit{Variable}&\textit{Mean}&\textit{(95\%}&\textit{CI)}\\"
>   )
>   foot("\end{tabular}");
\begin{tabular}{rrrr}
\textit{Variable}&\textit{Mean}&\textit{(95\%}&\textit{CI)}\\
Weight (US tons)&1.51&1.42&1.60\\
Fuel consumption (nipperkins/mile)&12.85&12.09&13.61\\
Trunk space (cu. ft.)&13.76&12.77&14.75\\
Price&6165.26&5481.91&6848.60\\
\end{tabular}
```

This output can be copied and pasted from the log file into a LaTeX document . . .

Table of means of car model attributes with confidence limits

. . . which, in this case, was the LaTeX document converted to this Beamer presentation, where we see the following resultstable:

Variable	*Mean*	*(95%*	*CI)*
Weight (US tons)	1.51	1.42	1.60
Fuel consumption (nipperkins/mile)	12.85	12.09	13.61
Trunk space (cu. ft.)	13.76	12.77	14.75
Price	6165.26	5481.91	6848.60

We could have used `listtab`, with different `rstyle()`, `head()`, and `foot()` options, to create tables in plain TeX, HTML, Markdown, or RTF. Or even tab–delimited tables, which can be copied and pasted into Microsoft Excel (for people who like that kind of thing). Or, instead of using `listtab`, we could have used the SSC package `docxtab` with `putdocx` to make a `.docx` table.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Example 2: A tale of 3 resultsframes `pframe`, `dframe`, and `fframe`

- In the `xauto` data, we create a variable `tradebloc`, identifying the 1970s trade bloc (USA, Japan, or EEC/EFTA) of the firm that makes the 1970s car model.
- We want to create a resultsset with 1 observation per trade bloc, and data on the mean fuel consumption of models from firms from that trade bloc.
- This resultsset will live in a data frame `pframe`, created by `parmest` after `regress`.
- The variable `tradebloc` will be regenerated in this resultsset, using the SSC packages `fvregen`[3] and `invdesc`[4] with a `descsave` resultsframe `dframe`.
- Finally, we will add frequencies of the 3 trade blocs, by merging in a `xcontract` resultsframe `fframe`.
- The resultsframe `pframe` will then be used to create a resultsplot of confidence intervals.

Creating the frequency resultsframe `fframe`

After creating the factor variable `tradebloc`, we use the SSC package `xcontract` to create a resultsset, with one observation per trade bloc and data on frequencies:

```
. xcontract tradebloc, list(, abbr(32))
>   frame(fframe, replace);

     +-------------------------------+
     | tradebloc   _freq   _percent |
     |-------------------------------|
  1. |       USA      52      70.27 |
  2. |     Japan      11      14.86 |
  3. |  EEC/EFTA      11      14.86 |
     +-------------------------------+
```

This is listed and saved to the resultsframe `fframe`.

Creating the descriptive resultsframe `dframe`

We then use the SSC package `descsave` to create a descriptive resultsset, with one observation for each of a list of one variable `tradebloc` and data on its attributes:

```
. descsave tradebloc, list(, abbr(32)) frame(dframe, replace);

     +----------------------------------------------------------------------------------+
     | order   name        type   isnumeric   format   vallab      varlab               |
     |----------------------------------------------------------------------------------|
  1. |     1   tradebloc   byte           1   %8.0g    tradebloc   Trading bloc of firm |
     +----------------------------------------------------------------------------------+
```

This is listed and saved to the resultsframe `dframe`.

Creating the parameter resultsframe `pframe`

We fit an *equal–variance* regression model for fuel consumption (in nipperkins per mile) with respect to the factor `tradebloc`, using the command

```
. regress npm ibn.tradebloc, noconst;
```

This generates the usual output (not shown). We then use the SSC package `parmest` to create a resultsset with one observation per parameter:

```
. parmest, format(estimate min* max* %8.2f)
>   list(parm estimate min* max*, abbr(32))
>   frame(pframe, replace);

     +--------------------------------------+
     |        parm   estimate   min95   max95 |
     |--------------------------------------|
  1. | 1.tradebloc      13.61   12.77   14.46 |
  2. | 2.tradebloc      10.29    8.45   12.14 |
  3. | 3.tradebloc      11.79    9.94   13.63 |
     +--------------------------------------+
```

This is listed and saved to the resultsframe `pframe`. Note that the parameters are group means (in nipperkins per mile).

Regenerating the factor variable `tradebloc`

In the resultsframe `pframe`, we regenerate the factor variable `tradebloc` from the parameter names, using the SSC package `fvregen`[3]:

```
. frame pframe {;
.   fvregen;
Factor variables generated:
tradebloc
.   describe tradebloc, full;

Variable      Storage   Display    Value
    name         type    format    label      Variable label
---------------------------------------------------------------------------------------------
tradebloc       byte    %12.0g
.   list parm tradebloc estimate min* max*, abbr(32);

     +----------------------------------------------------+
     |        parm   tradebloc   estimate   min95   max95 |
     |----------------------------------------------------|
  1. | 1.tradebloc           1      13.61   12.77   14.46 |
  2. | 2.tradebloc           2      10.29    8.45   12.14 |
  3. | 3.tradebloc           3      11.79    9.94   13.63 |
     +----------------------------------------------------+
.   };
```

The variable `tradebloc` in the resultsset has the correct values (extracted from the parameter–name variable `parm`), but *not* the variable and value labels that it had in the input dataset.

Regenerating attributes for the factor variable `tradebloc`

In the resultsframe `pframe`, we regenerate the attributes of the factor variable `tradebloc` from the parameter names, using the SSC package `invdesc`[4] to input these attributes from the `descsave` resultsframe `dframe` that we made earlier:

```
. frame pframe {;
.   invdesc, dframe(dframe) lframe(dframe);
.   describe tradebloc, full;

Variable      Storage   Display    Value
    name         type    format    label      Variable label
----------------------------------------------------------------------------------------------
tradebloc       byte    %8.0g      tradebloc
                                              Trading bloc of firm
.   list parm tradebloc estimate min* max*, abbr(32);

     +---------------------------------------------------+
     |        parm   tradebloc   estimate   min95   max95 |
     |---------------------------------------------------|
  1. | 1.tradebloc         USA      13.61   12.77   14.46 |
  2. | 2.tradebloc       Japan      10.29    8.45   12.14 |
  3. | 3.tradebloc    EEC/EFTA      11.79    9.94   13.63 |
     +---------------------------------------------------+
.   };
```

The variable `tradebloc` in the resultsset now has the variable and value labels that it had in the input dataset.

Merging in the frequency variables from `fframe`

We then use the module `fraddinby` of the SSC package `addinby` to merge in the frequencies and percents of cars from each trade bloc from the `xcontract` resultsframe `fframe` that we made earlier:

```
. frame pframe {;
.   fraddinby tradebloc, frame(fframe);
.   list parm tradebloc _freq _percent estimate min* max*, abbr(32);

     +------------------------------------------------------------------------+
     |        parm   tradebloc   _freq   _percent   estimate   min95   max95 |
     |------------------------------------------------------------------------|
  1. | 1.tradebloc         USA      52      70.27      13.61   12.77   14.46 |
  2. | 2.tradebloc       Japan      11      14.86      10.29    8.45   12.14 |
  3. | 3.tradebloc    EEC/EFTA      11      14.86      11.79    9.94   13.63 |
     +------------------------------------------------------------------------+
.     };
```

The variable `tradebloc` now has frequencies and percentages. We can now make a resultsplot.

Preparing for the resultsplot

- We would like to plot the confidence intervals for the trade bloc mean fuel consumption against the trade bloc factor.
- And we would like to label each trade bloc with its frequency (in parenthesis).
- For doing this, it is very useful to be able to convert from factors to string variables (and *vice versa*) at will.
- Fortunately, we have 2 SSC packages `sencode` and `sdecode`[5], which are "super" versions of `encode` and `decode`, respectively.

Preparing for the resultsplot

- We would like to plot the confidence intervals for the trade bloc mean fuel consumption against the trade bloc factor.
- And we would like to label each trade bloc with its frequency (in parenthesis).
- For doing this, it is very useful to be able to convert from factors to string variables (and *vice versa*) at will.
- Fortunately, we have 2 SSC packages `sencode` and `sdecode`[5], which are "super" versions of `encode` and `decode`, respectively.

Preparing for the resultsplot

- We would like to plot the confidence intervals for the trade bloc mean fuel consumption against the trade bloc factor.
- And we would like to label each trade bloc with its frequency (in parenthesis).
- For doing this, it is very useful to be able to convert from factors to string variables (and *vice versa*) at will.
- Fortunately, we have 2 SSC packages `sencode` and `sdecode`[5], which are "super" versions of `encode` and `decode`, respectively.

Preparing for the resultsplot

- We would like to plot the confidence intervals for the trade bloc mean fuel consumption against the trade bloc factor.
- And we would like to label each trade bloc with its frequency (in parenthesis).
- For doing this, it is very useful to be able to convert from factors to string variables (and *vice versa*) at will.
- Fortunately, we have 2 SSC packages `sencode` and `sdecode`[5], which are "super" versions of `encode` and `decode`, respectively.

Preparing for the resultsplot

- We would like to plot the confidence intervals for the trade bloc mean fuel consumption against the trade bloc factor.
- And we would like to label each trade bloc with its frequency (in parenthesis).
- For doing this, it is very useful to be able to convert from factors to string variables (and *vice versa*) at will.
- Fortunately, we have 2 SSC packages `sencode` and `sdecode`[5], which are "super" versions of `encode` and `decode`, respectively.

Creating a new version of the trade bloc variable

We use the SSC package `sdecode` to decode `tradebloc` to `tradebloc2`, add frequencies in parentheses to `tradebloc2`, and then use `sencode` to encode `tradebloc2`, in the order specified by `tradebloc`:

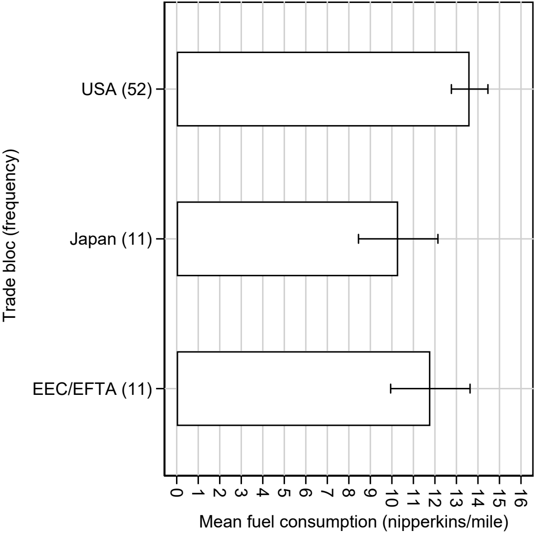
```
. frame pframe {;
.   sdecode tradebloc, gene(tradebloc2);
.   replace tradebloc2=tradebloc2+" ("+string(_freq)+")";
variable tradebloc2 was str8 now str13
(3 real changes made)
.   sencode tradebloc2, replace gsort(tradebloc);
.   lab var tradebloc2 "Trade bloc (frequency)";
.   list tradebloc2 estimate min* max*, abbr(32);

     +-------------------------------------------+
     |    tradebloc2   estimate   min95   max95 |
     |-------------------------------------------|
  1. |      USA (52)      13.61   12.77   14.46 |
  2. |    Japan (11)      10.29    8.45   12.14 |
  3. | EEC/EFTA (11)      11.79    9.94   13.63 |
     +-------------------------------------------+
. };
```

The variable `tradebloc2` contains trade blocs with their frequencies. We can now make a resultsplot.

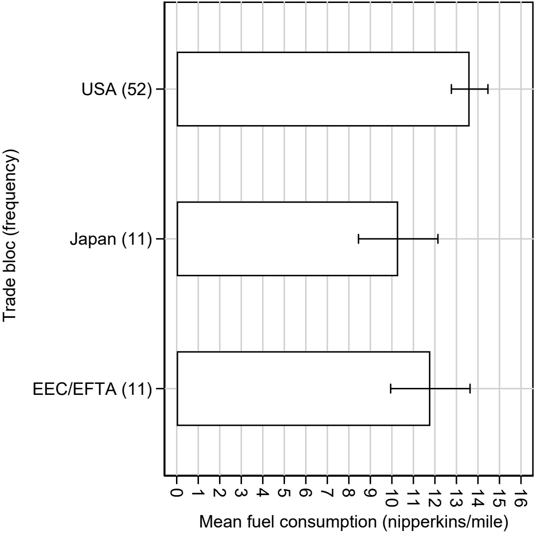
Resultsplot of fuel consumption by trade bloc

- This plot was made using the SCC package `eclplot`[6].
- The vertical axis gives the trade blocs, with their frequencies.
- The horizontal axis gives the mean fuel consumption for models from each trade bloc, with confidence limits.
- Note that resultsframes (unlike tables) can be plotted!



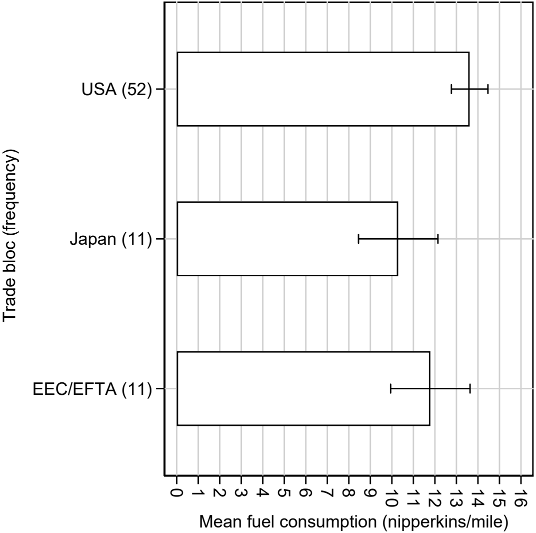
Resultsplot of fuel consumption by trade bloc

- This plot was made using the SCC package `eclplot`[6].
- The vertical axis gives the trade blocs, with their frequencies.
- The horizontal axis gives the mean fuel consumption for models from each trade bloc, with confidence limits.
- Note that resultsframes (unlike tables) can be plotted!



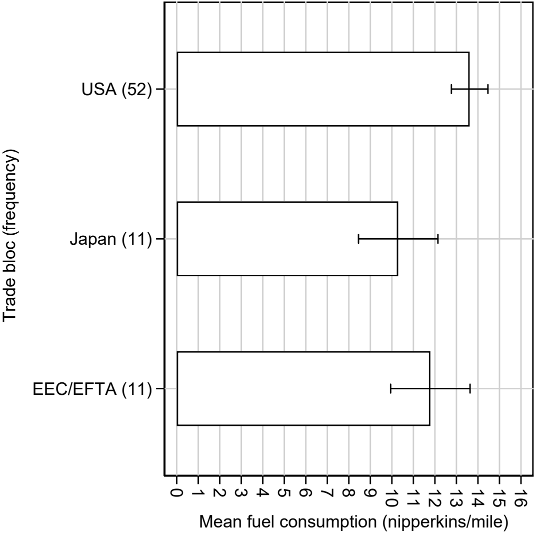
Resultsplot of fuel consumption by trade bloc

- This plot was made using the SCC package `eclplot`[6].
- The vertical axis gives the trade blocs, with their frequencies.
- The horizontal axis gives the mean fuel consumption for models from each trade bloc, with confidence limits.
- Note that resultsframes (unlike tables) can be plotted!



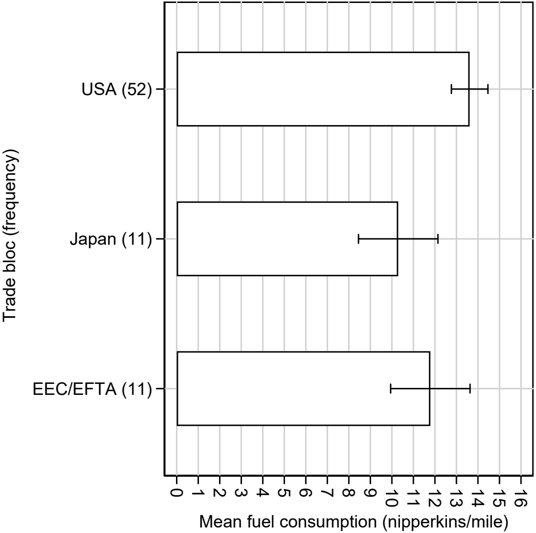
Resultsplot of fuel consumption by trade bloc

- This plot was made using the SCC package `eclplot`[6].
- The vertical axis gives the trade blocs, with their frequencies.
- The horizontal axis gives the mean fuel consumption for models from each trade bloc, with confidence limits.
- Note that resultsframes (unlike tables) can be plotted!



Resultsplot of fuel consumption by trade bloc

- This plot was made using the SCC package `eclplot`[6].
- The vertical axis gives the trade blocs, with their frequencies.
- The horizontal axis gives the mean fuel consumption for models from each trade bloc, with confidence limits.
- Note that resultsframes (unlike tables) can be plotted!



References

[1] Newson, R. From datasets to resultssets in Stata. Presented at the 10th UK Stata User Meeting, 28–29 June, 2004. Downloadable from the conference website at *http://ideas.repec.org/p/boc/usug04/16.html*

[2] 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*

[3] Newson, R. B. Post–`parmest` peripherals: `fvregen`, `invcise`, and `qqvalue`. Presented at the 16th UK Stata User Meeting, 9–10 September, 2010. Downloadable from the conference website at *http://ideas.repec.org/p/boc/usug10/01.html*

[4] Newson, R. B. From datasets to metadatasets in Stata. Presented at the 2020 London Stata Conference, 10–11 September, 2020. Downloadable from the conference website at *http://ideas.repec.org/p/boc/usug20/01.html*

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

[6] Newson, R. Generalized confidence interval plots using commands or dialogs. Presented at the 11th UK Stata User Meeting, 17–18 May, 2005. Downloadable from the conference website at *https://ideas.repec.org/p/boc/usug05/01.html*

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