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/

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

This presentation, including the handout and examples, can be downloaded from the conference website at
http://www.stata.com/support/meeting/10uk/

What are resultssets?

What are resultssets?

- A **resultsset** is a Stata dataset created as output by a Stata program.

What are resultssets?

- A **resultsset** is a Stata dataset created as output by a Stata program.
- The term was coined by Nick Cox, who has used it a few times on Statalist.

What are resultssets?

- A **resultsset** is a Stata dataset created as output by a Stata program.
- The term was coined by Nick Cox, who has used it a few times on Statalist.
- SAS users should note that Stata datasets do the job of SAS data sets, and Stata resultssets do the job of SAS output data sets.

What are resultssets?

- A **resultsset** is a Stata dataset created as output by a Stata program.
- The term was coined by Nick Cox, who has used it a few times on Statalist.
- SAS users should note that Stata datasets do the job of SAS data sets, and Stata resultssets do the job of SAS output data sets.
- Stata resultssets may be saved to disk files, and/or written to the memory (overwriting any existing data), and/or simply listed to the Stata log and/or Results window.

Why resultssets?

Why resultssets?

- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.

Why resultssets?

- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.
- The `auto` dataset has one observation per car model, and data on car attributes. The `voter` dataset has one observation per presidential candidate per income bracket, and data on numbers of votes.

Why resultssets?

- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.
- The `auto` dataset has one observation per car model, and data on car attributes. The `voter` dataset has one observation per presidential candidate per income bracket, and data on numbers of votes.
- Data analysts typically start with datasets with one observation per "experimental unit", and measurements on these units.

Why resultssets?

- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.
- The `auto` dataset has one observation per car model, and data on car attributes. The `voter` dataset has one observation per presidential candidate per income bracket, and data on numbers of votes.
- Data analysts typically start with datasets with one observation per "experimental unit", and measurements on these units.
- *However*, they are usually paid to produce presentation-ready plots and/or publication-ready tables.

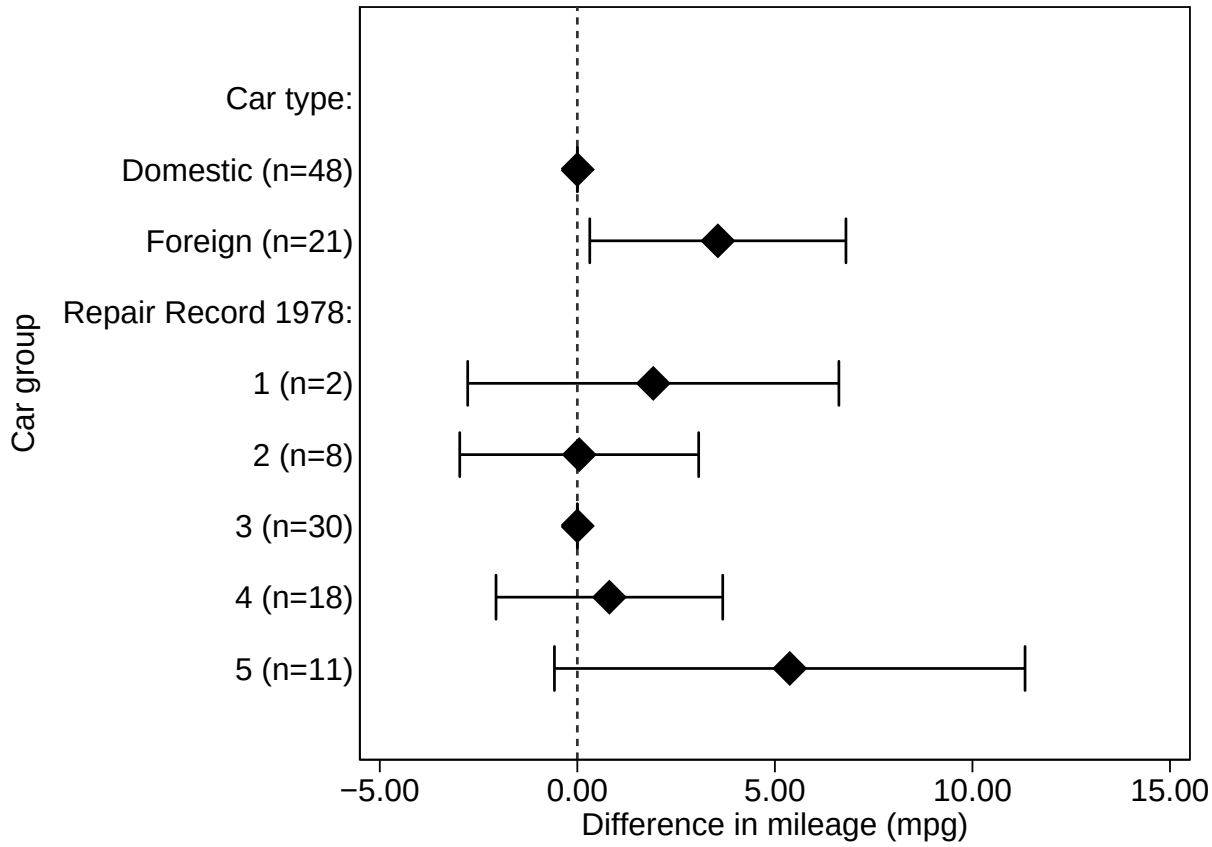
Why resultssets?

- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.
- The `auto` dataset has one observation per car model, and data on car attributes. The `voter` dataset has one observation per presidential candidate per income bracket, and data on numbers of votes.
- Data analysts typically start with datasets with one observation per "experimental unit", and measurements on these units.
- *However*, they are usually paid to produce presentation-ready plots and/or publication-ready tables.
- To do this, they require datasets with one observation per table row, or per axis label, and usually produce them manually using spreadsheets.

Why resultssets?

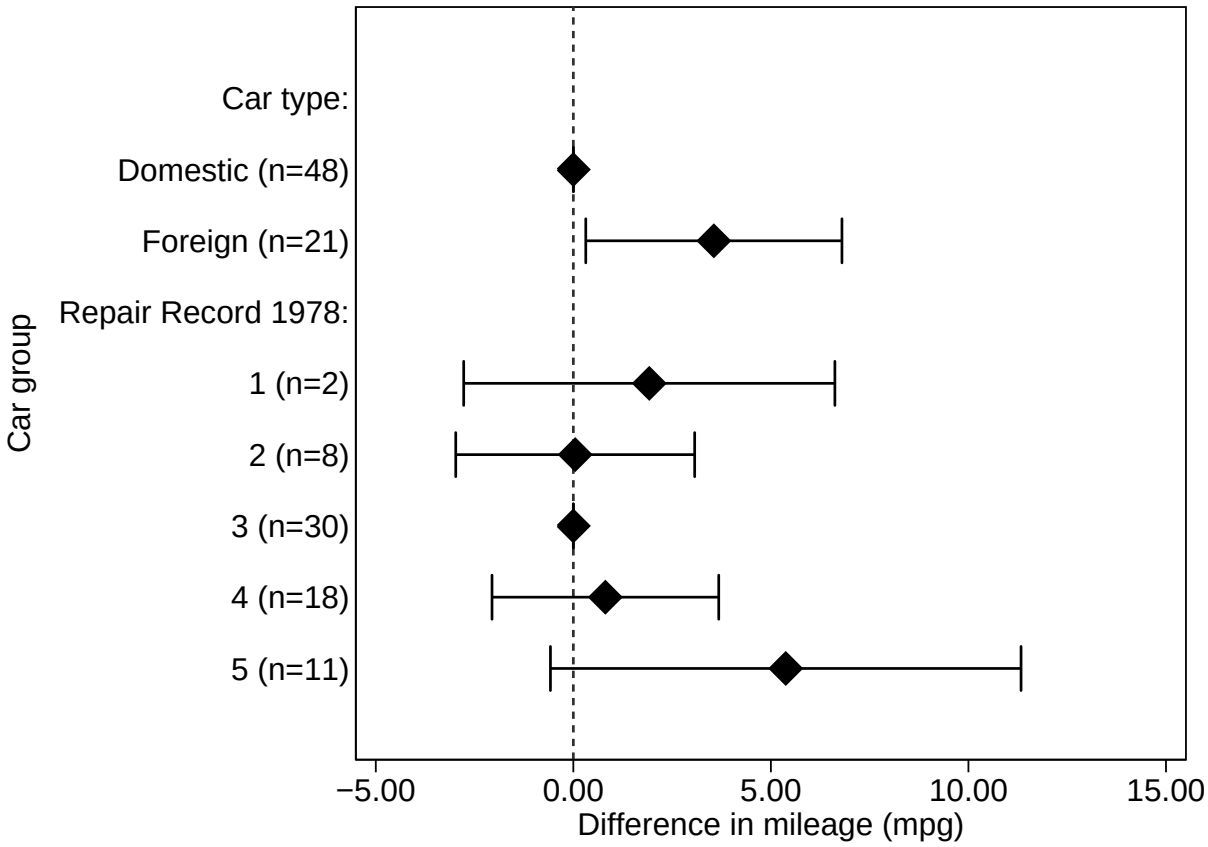
- A dataset (Stata or otherwise) should have one observation per *thing*, and data on *attributes_of_things*.
- The `auto` dataset has one observation per car model, and data on car attributes. The `voter` dataset has one observation per presidential candidate per income bracket, and data on numbers of votes.
- Data analysts typically start with datasets with one observation per "experimental unit", and measurements on these units.
- *However*, they are usually paid to produce presentation-ready plots and/or publication-ready tables.
- To do this, they require datasets with one observation per table row, or per axis label, and usually produce them manually using spreadsheets.
- We will demonstrate alternative ways of producing such datasets in Stata.

Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 3)



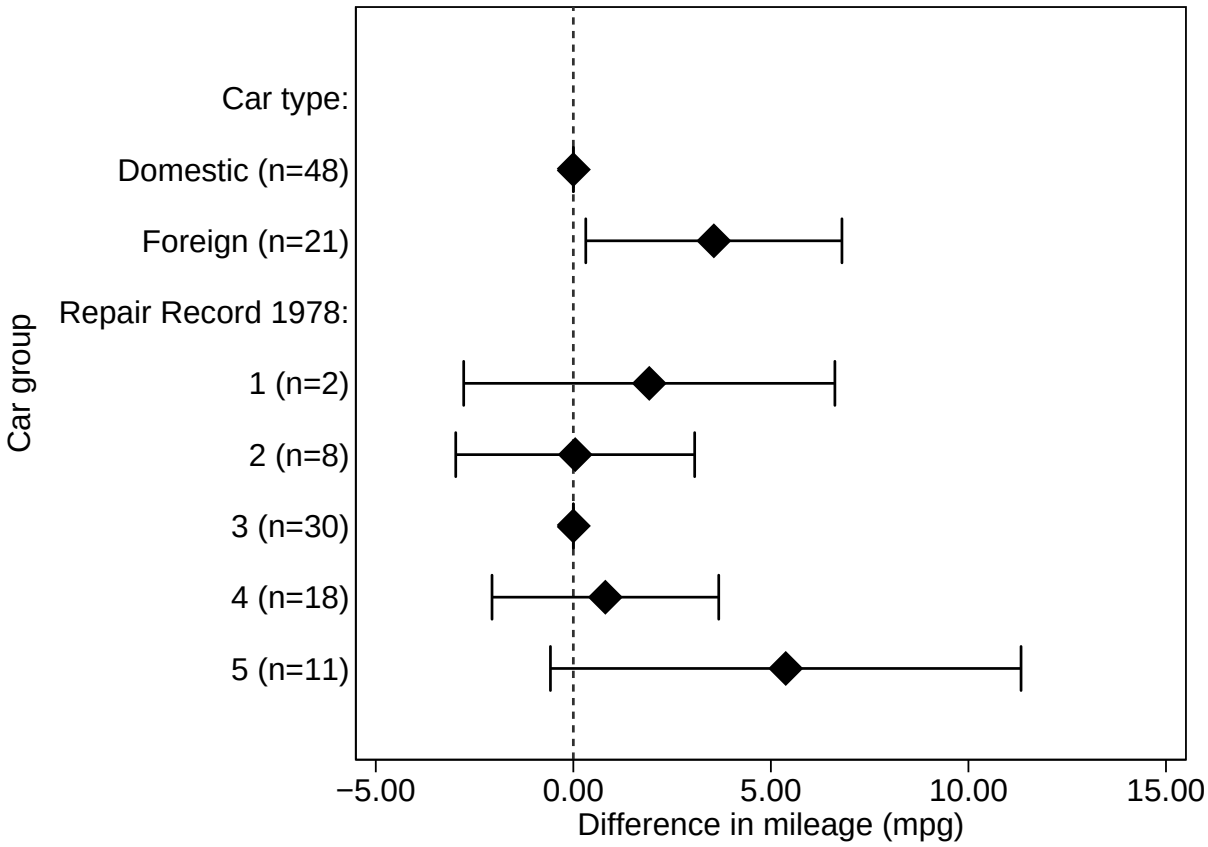
Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 3)

- This confidence interval plot was created using the `eclplot` package.



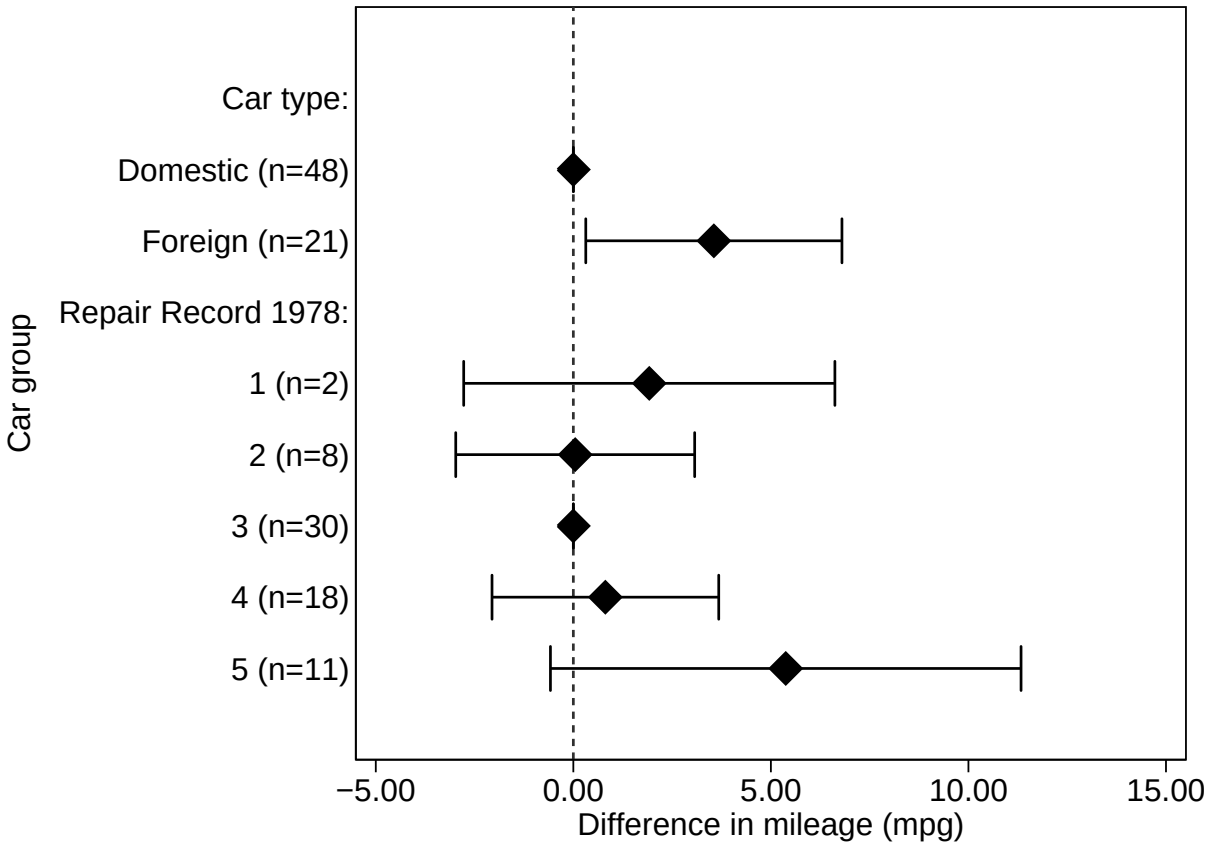
Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 3)

- This confidence interval plot was created using the `eclplot` package.
- `eclplot` inputs a dataset with 1 observation per parameter, and data on estimates, confidence limits, and one more variable for the other axis.



Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 3)

- This confidence interval plot was created using the `eclplot` package.
- `eclplot` inputs a dataset with 1 observation per parameter, and data on estimates, confidence limits, and one more variable for the other axis.
- It *cannot* create this plot directly using the original `auto` data.



Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 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

Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 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

This table was produced using `listtex`, which produces TEX, HTML and Microsoft Word tables, and inputs a dataset with one observation per table row.

Resultsset-generating programs downloadable from SSC

	The resultsset contains:	
Program	*one observation per...*	*and data on...*
`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 resultsset-generating programs include `collapse`, `contract`, `statsby`, `bootstrap`, `simulate`, and the utility `post`.

`example1.do`: Demonstration of the resultsset-generating programs `descsave`, `parmby`, `xcollapse` and `xcontract`

`example1.do`: Demonstration of the resultsset-generating programs `descsave`, `parmby`, `xcollapse` and `xcontract`

- This example is one of a set, downloadable from the conference website at *http://www.stata.com/support/meeting/10uk/* together with the overheads and the handout.

`example1.do`: Demonstration of the resultsset-generating programs `descsave`, `parmby`, `xcollapse` and `xcontract`

- This example is one of a set, downloadable from the conference website at *http://www.stata.com/support/meeting/10uk/* together with the overheads and the handout.
- Each program inputs the `auto` dataset and writes a resultsset to the memory.

`example1.do`: Demonstration of the resultsset-generating programs `descsave`, `parmby`, `xcollapse` and `xcontract`

- This example is one of a set, downloadable from the conference website at *http://www.stata.com/support/meeting/10uk/* together with the overheads and the handout.
- Each program inputs the `auto` dataset and writes a resultsset to the memory.
- We then `describe` the variables, and `list` the observations.

Common resultsset-destination options for resultsset-generating programs

Option	*Function*
`list()`	List the resultsset to the log or Results window
`saving()`	Save the resultsset to a disk file
`norestore` (or `replace`)	Write the resultsset to memory
`fast`	Fast version of `norestore` for programmers
`flist()`	Global macro accumulating `saving()` filenames

These options are not mutually exclusive.

Resultssets are frequently concatenated

Resultssets are frequently concatenated

- The `dsconcat` package inputs a list of dataset files, and concatenates these datasets (or a subset of each dataset) into the memory.

Resultssets are frequently concatenated

- The `dsconcat` package inputs a list of dataset files, and concatenates these datasets (or a subset of each dataset) into the memory.
- This is very useful if we have multiple `parmby` resultssets for multiple models, and want to plot or tabulate the "interesting" parameters from all the models, discarding the "nuisance" parameters.

Resultssets are frequently concatenated

- The `dsconcat` package inputs a list of dataset files, and concatenates these datasets (or a subset of each dataset) into the memory.
- This is very useful if we have multiple `parmby` resultssets for multiple models, and want to plot or tabulate the "interesting" parameters from all the models, discarding the "nuisance" parameters.
- This ability is an advantage of the resultsset method over more "instant" complementary methods, using `estimates table`, `outreg` or `reformat`.

Resultssets are frequently concatenated

- The `dsconcat` package inputs a list of dataset files, and concatenates these datasets (or a subset of each dataset) into the memory.
- This is very useful if we have multiple `parmby` resultssets for multiple models, and want to plot or tabulate the "interesting" parameters from all the models, discarding the "nuisance" parameters.
- This ability is an advantage of the resultsset method over more "instant" complementary methods, using `estimates table`, `outreg` or `reformat`.
- These complementary methods produce tables (but *not* plots) of results from a *single* model.

Common resultsset-modifying options for resultsset-generating programs

Option	*Function*
`rename()`	Give variables in resultsset nondefault 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 `idnum()` and `idstr()` options are useful if multiple resultssets are concatenated. The `by()` option causes the resultsset to be "concatenated at birth", with a "resultssubset" for each by-group.

Resultssets frequently live in `tempfiles`

Resultssets frequently live in `tempfiles`

- Temporary files are created using the `tempfile` command or the `tempfile` extended macro function.

Resultssets frequently live in `tempfiles`

- Temporary files are created using the `tempfile` command or the `tempfile` extended macro function.
- In the Stata world (in contrast to SAS), temporary files are viewed as a specialist subject, of interest to a subset of those programmers who use macros.

Resultssets frequently live in `tempfiles`

- Temporary files are created using the `tempfile` command or the `tempfile` extended macro function.
- In the Stata world (in contrast to SAS), temporary files are viewed as a specialist subject, of interest to a subset of those programmers who use macros.
- Stata has the advantage over SAS of processing a whole dataset in memory.

Resultssets frequently live in `tempfiles`

- Temporary files are created using the `tempfile` command or the `tempfile` extended macro function.
- In the Stata world (in contrast to SAS), temporary files are viewed as a specialist subject, of interest to a subset of those programmers who use macros.
- Stata has the advantage over SAS of processing a whole dataset in memory.
- *However*, it can only do this with one dataset at a time.

Resultssets frequently live in `tempfiles`

- Temporary files are created using the `tempfile` command or the `tempfile` extended macro function.
- In the Stata world (in contrast to SAS), temporary files are viewed as a specialist subject, of interest to a subset of those programmers who use macros.
- Stata has the advantage over SAS of processing a whole dataset in memory.
- *However*, it can only do this with one dataset at a time.
- *Therefore*, Stata users who concatenate resultssets should know at least enough about macros to produce `tempfiles`.

`example2.do`: Concatenating and merging resultssets in `tempfiles`

`example2.do`: Concatenating and merging resultssets in `tempfiles`

- We will create 10 temporary files, with names in macros `tf0` to `tf9`.

`example2.do`: Concatenating and merging resultssets in `tempfiles`

- We will create 10 temporary files, with names in macros `tf0` to `tf9`.
- The file '`tf0`' is a `descsave` resultsset, with information on 9 quantitative variables in the `auto` data.

`example2.do`: Concatenating and merging resultssets in `tempfiles`

- We will create 10 temporary files, with names in macros `tf0` to `tf9`.
- The file '`tf0`' is a `descsave` resultsset, with information on 9 quantitative variables in the `auto` data.
- The files '`tf1`' to '`tf9`' are `xcollapse` resultssets, one for each of the 9 quantitative variables, containing medians of those variables for each car type (US and non-US cars).

`example2.do`: Concatenating and merging resultssets in `tempfiles`

- We will create 10 temporary files, with names in macros `tf0` to `tf9`.
- The file ‘`tf0`’ is a `descsave` resultsset, with information on 9 quantitative variables in the `auto` data.
- The files ‘`tf1`’ to ‘`tf9`’ are `xcollapse` resultssets, one for each of the 9 quantitative variables, containing medians of those variables for each car type (US and non-US cars).
- We concatenate the files ‘`tf1`’ to ‘`tf9`’ into the memory to form a long resultsset, with one observation per variable per car type.

`example2.do`: Concatenating and merging resultssets in `tempfiles`

- We will create 10 temporary files, with names in macros `tf0` to `tf9`.
- The file ‘`tf0`’ is a `descsave` resultsset, with information on 9 quantitative variables in the `auto` data.
- The files ‘`tf1`’ to ‘`tf9`’ are `xcollapse` resultssets, one for each of the 9 quantitative variables, containing medians of those variables for each car type (US and non-US cars).
- We concatenate the files ‘`tf1`’ to ‘`tf9`’ into the memory to form a long resultsset, with one observation per variable per car type.
- We then `merge` in the `descsave` resultsset in ‘`tf0`’, adding variable labels.

String-numeric conversion in resultssets

String-numeric conversion in resultssets

- *Most* Stata graphics programs will not plot string variables until they are converted to labelled numeric.

String-numeric conversion in resultssets

- *Most* Stata graphics programs will not plot string variables until they are converted to labelled numeric.
- *On the other hand*, in tables, we cannot add commas and parentheses to confidence limits, or `"%"` to percentages, without converting them to string.

String-numeric conversion in resultssets

- *Most* Stata graphics programs will not plot string variables until they are converted to labelled numeric.
- *On the other hand*, in tables, we cannot add commas and parentheses to confidence limits, or `"%"` to percentages, without converting them to string.
- *Therefore*, resultsset processing is much easier if we can convert between string and numeric with minimum effort.

String-numeric conversion in resultssets

- *Most* Stata graphics programs will not plot string variables until they are converted to labelled numeric.
- *On the other hand*, in tables, we cannot add commas and parentheses to confidence limits, or `"%"` to percentages, without converting them to string.
- *Therefore*, resultsset processing is much easier if we can convert between string and numeric with minimum effort.
- Official Stata has 4 programs for this, `encode`, `decode`, `tostring` and `destring`, which all have limitations (especially `encode`).

String-numeric conversion in resultssets

- *Most* Stata graphics programs will not plot string variables until they are converted to labelled numeric.
- *On the other hand*, in tables, we cannot add commas and parentheses to confidence limits, or `"%"` to percentages, without converting them to string.
- *Therefore*, resultsset processing is much easier if we can convert between string and numeric with minimum effort.
- Official Stata has 4 programs for this, `encode`, `decode`, `tostring` and `destring`, which all have limitations (especially `encode`).
- I therefore developed the packages `sencode` and `sdecode` for string-numeric conversion, and `factext` and `factmerg` for string-factor conversion.

String-numeric conversion using `sencode` and `sdecode`

String-numeric conversion using `sencode` and `sdecode`

- `sencode` and `sdecode` convert a single input variable to a single output variable, which may be `generate`d as a new variable, or `replace` the input variable.

String-numeric conversion using `sencode` and `sdecode`

- `sencode` and `sdecode` convert a single input variable to a single output variable, which may be `generate`d as a new variable, or `replace` the input variable.
- `sencode` encodes a string variable to a labelled numeric variable in a non-alphabetic order, defaulting to order of appearance and reset by the `gsort()` option.

String-numeric conversion using `sencode` and `sdecode`

- `sencode` and `sdecode` convert a single input variable to a single output variable, which may be `generate`d as a new variable, or `replace` the input variable.
- `sencode` encodes a string variable to a labelled numeric variable in a non-alphabetic order, defaulting to order of appearance and reset by the `gsort()` option.
- The `manyto1` option allows multiple numeric codes to share one string label.

String-numeric conversion using `sencode` and `sdecode`

- `sencode` and `sdecode` convert a single input variable to a single output variable, which may be `generate`d as a new variable, or `replace` the input variable.
- `sencode` encodes a string variable to a labelled numeric variable in a non-alphabetic order, defaulting to order of appearance and reset by the `gsort()` option.
- The `manyto1` option allows multiple numeric codes to share one string label.
- `sdecode` decodes a numeric variable to string, using value labels if possible and display formats otherwise.

`example3.do`: Statistical methods

`example3.do`: Statistical methods

- In the `auto` data, we will use the 9 quantitative variables of `example2.do`.

`example3.do`: Statistical methods

- In the `auto` data, we will use the 9 quantitative variables of `example2.do`.
- We compare their performance as diagnostic predictors of the "condition" of non-US origin, using Somers' D.

`example3.do`: Statistical methods

- In the `auto` data, we will use the 9 quantitative variables of `example2.do`.
- We compare their performance as diagnostic predictors of the "condition" of non-US origin, using Somers' D.
- Somers' D is related to the area A under the sensitivity-specificity (or ROC) curve by the formula $D = 2A - 1$.

`example3.do`: Statistical methods

- In the `auto` data, we will use the 9 quantitative variables of `example2.do`.
- We compare their performance as diagnostic predictors of the "condition" of non-US origin, using Somers' D.
- Somers' D is related to the area A under the sensitivity-specificity (or ROC) curve by the formula $D = 2A - 1$.
- The two performance measures are therefore equivalent, but Somers' D is positive for positive predictors, negative for negative predictors, and zero for non-predictors.

`example3.do`: Creating a plot and table using `sencode` and `sdecode`

`example3.do`: Creating a plot and table using `sencode` and `sdecode`

- We use `somersd` to estimate the Somers' D parameters, and save them in a `parmby` resultsset.

`example3.do`: Creating a plot and table using `sencode` and `sdecode`

- We use `somersd` to estimate the Somers' D parameters, and save them in a `parmby` resultsset.
- This resultsset contains a variable `label`, containing the variable label of the predictor.

`example3.do`: Creating a plot and table using `sencode` and `sdecode`

- We use `somersd` to estimate the Somers' D parameters, and save them in a `parmby` resultsset.
- This resultsset contains a variable `label`, containing the variable label of the predictor.
- Using `sencode`, we convert `label` to numeric, and produce a confidence interval plot.

`example3.do`: Creating a plot and table using `sencode` and `sdecode`

- We use `somersd` to estimate the Somers' D parameters, and save them in a `parmby` resultsset.
- This resultsset contains a variable `label`, containing the variable label of the predictor.
- Using `sencode`, we convert `label` to numeric, and produce a confidence interval plot.
- We then use `sdecode` to convert the confidence limits to string, and produce a confidence interval table.

String-factor conversion using `factext` and `descsave`

String-factor conversion using `factext` and `descsave`

- Many predictor variables are dummy variables for categorical factors, produced mainly by `xi`, with variable labels of the form
  "*factor_name*==*value*"
  which are stored in the `label` variable of `parmby` resultssets.

String-factor conversion using `factext` and `descsave`

- Many predictor variables are dummy variables for categorical factors, produced mainly by `xi`, with variable labels of the form
  "*factor_name*==*value*"
  which are stored in the `label` variable of `parmby` resultssets.
- `factext` inputs the `label` variable and generates a list of factors, with names from the left-hand side and values from the right-hand side.

String-factor conversion using `factext` and `descsave`

- Many predictor variables are dummy variables for categorical factors, produced mainly by `xi`, with variable labels of the form
  "*factor_name*==*value*"
  which are stored in the `label` variable of `parmby` resultssets.
- `factext` inputs the `label` variable and generates a list of factors, with names from the left-hand side and values from the right-hand side.
- `descsave` helps by writing a do-file to reconstruct the storage types, display formats, value labels, variable labels and characteristics of the variables described.

String-factor conversion using `factext` and `descsave`

- Many predictor variables are dummy variables for categorical factors, produced mainly by `xi`, with variable labels of the form
  "*factor_name*==*value*"
  which are stored in the `label` variable of `parmby` resultssets.
- `factext` inputs the `label` variable and generates a list of factors, with names from the left-hand side and values from the right-hand side.
- `descsave` helps by writing a do-file to reconstruct the storage types, display formats, value labels, variable labels and characteristics of the variables described.
- `factext` runs this do-file to reconstruct the factors, which are then ready for use in confidence interval plots.

Factor-string conversion using `factmerg`

Factor-string conversion using `factmerg`

- Resultssets often contain multiple factors, each missing in most observations.

Factor-string conversion using `factmerg`

- Resultssets often contain multiple factors, each missing in most observations.
- A multi-factor table, by contrast, usually has a *single* left column of row labels, containing information on *all* the factors.

Factor-string conversion using `factmerg`

- Resultssets often contain multiple factors, each missing in most observations.
- A multi-factor table, by contrast, usually has a *single* left column of row labels, containing information on *all* the factors.
- `factmerg` inputs a list of factors, and generates up to 3 string variables, containing the factors' names, variable labels and string values.

Factor-string conversion using `factmerg`

- Resultssets often contain multiple factors, each missing in most observations.
- A multi-factor table, by contrast, usually has a *single* left column of row labels, containing information on *all* the factors.
- `factmerg` inputs a list of factors, and generates up to 3 string variables, containing the factors' names, variable labels and string values.
- These string variables can in turn be used to create row label variables for multi-factor tables. (Or for multi-factor plots.)

`example4.do`: Confidence interval plots using `factext` and `factmerg`

`example4.do`: Confidence interval plots using `factext` and `factmerg`

- In the `auto` data, we use `descsave` to write a do-file to reconstruct the two factors `foreign` and `rep78`.

`example4.do`: Confidence interval plots using `factext` and `factmerg`

- In the `auto` data, we use `descsave` to write a do-file to reconstruct the two factors `foreign` and `rep78`.
- We then fit a regression model predicting `mpg` from these two factors, storing the results in a `parmby` resultsset.

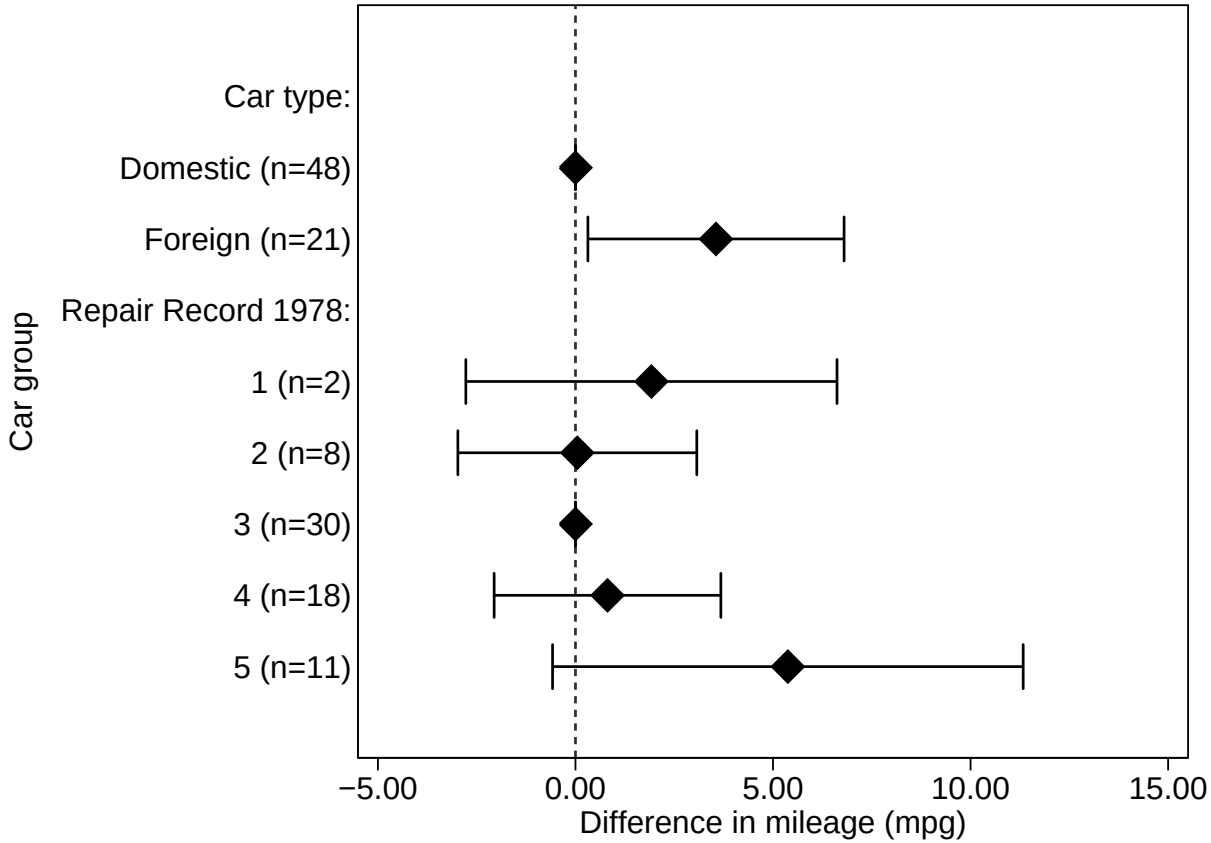
`example4.do`: Confidence interval plots using `factext` and `factmerg`

- In the `auto` data, we use `descsave` to write a do-file to reconstruct the two factors `foreign` and `rep78`.
- We then fit a regression model predicting `mpg` from these two factors, storing the results in a `parmby` resultsset.
- Using `factext`, we reconstruct these two factors in the resultsset, and draw a single-factor confidence interval plot.

`example4.do`: Confidence interval plots using `factext` and `factmerg`

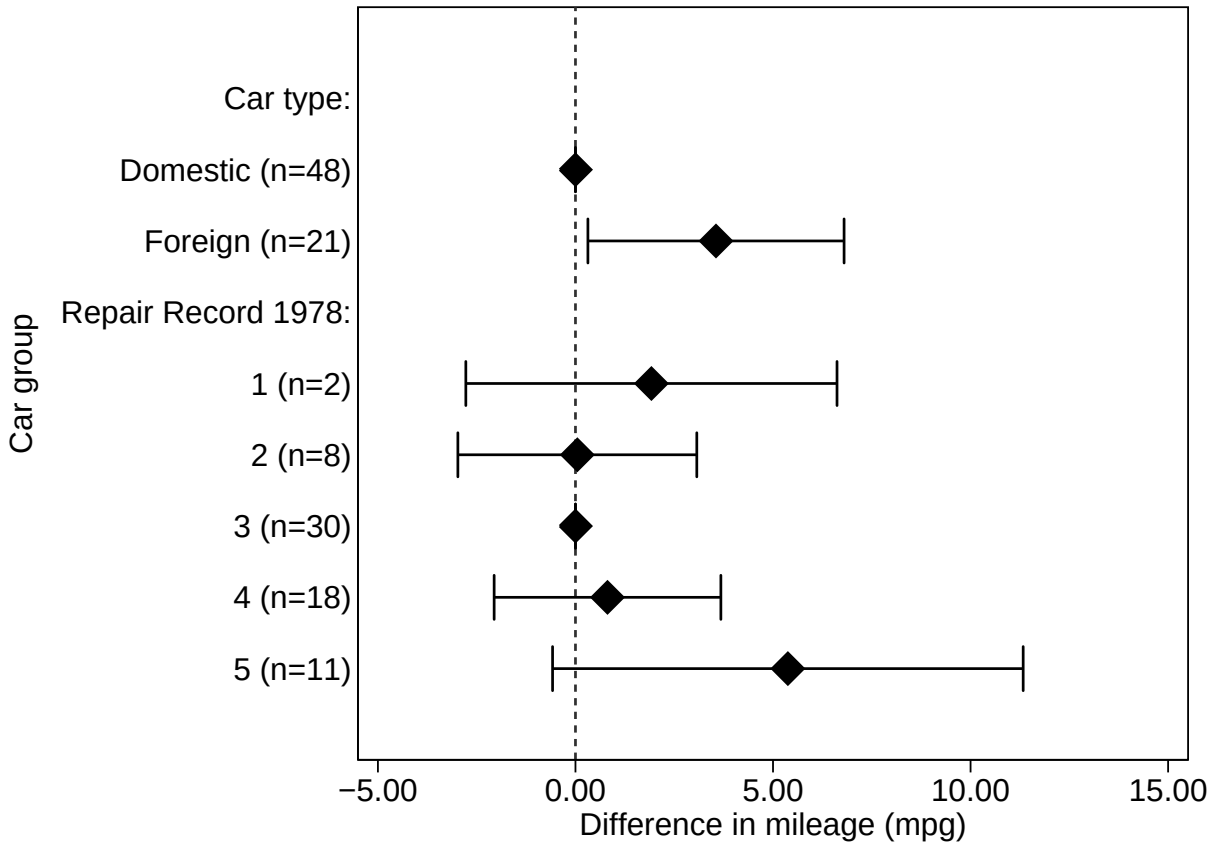
- In the `auto` data, we use `descsave` to write a do-file to reconstruct the two factors `foreign` and `rep78`.
- We then fit a regression model predicting `mpg` from these two factors, storing the results in a `parmby` resultsset.
- Using `factext`, we reconstruct these two factors in the resultsset, and draw a single-factor confidence interval plot.
- Using `factmerg`, we merge the two factors together, and draw a multi-factor confidence interval plot.

`example4.do` does not create this confidence interval plot



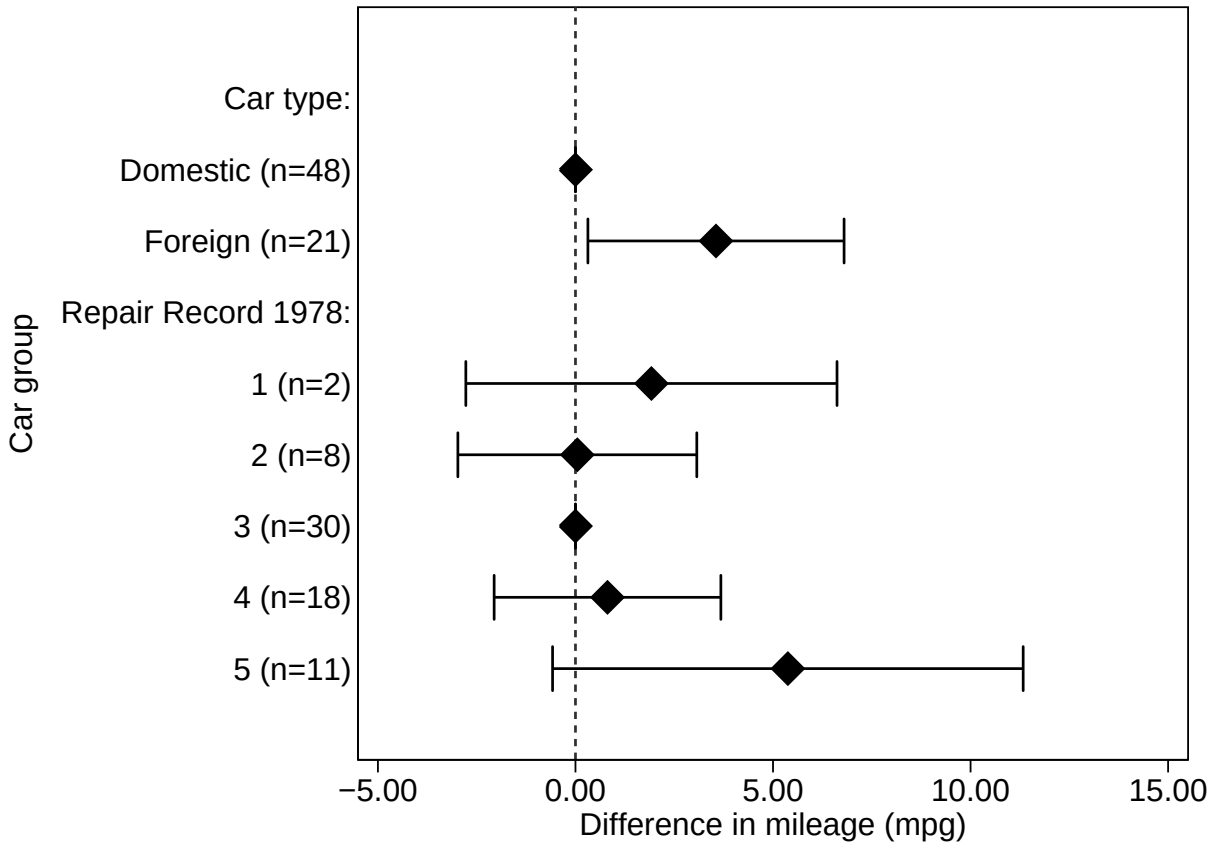
`example4.do` does not create this confidence interval plot

- This plot has group frequencies as well as group names.



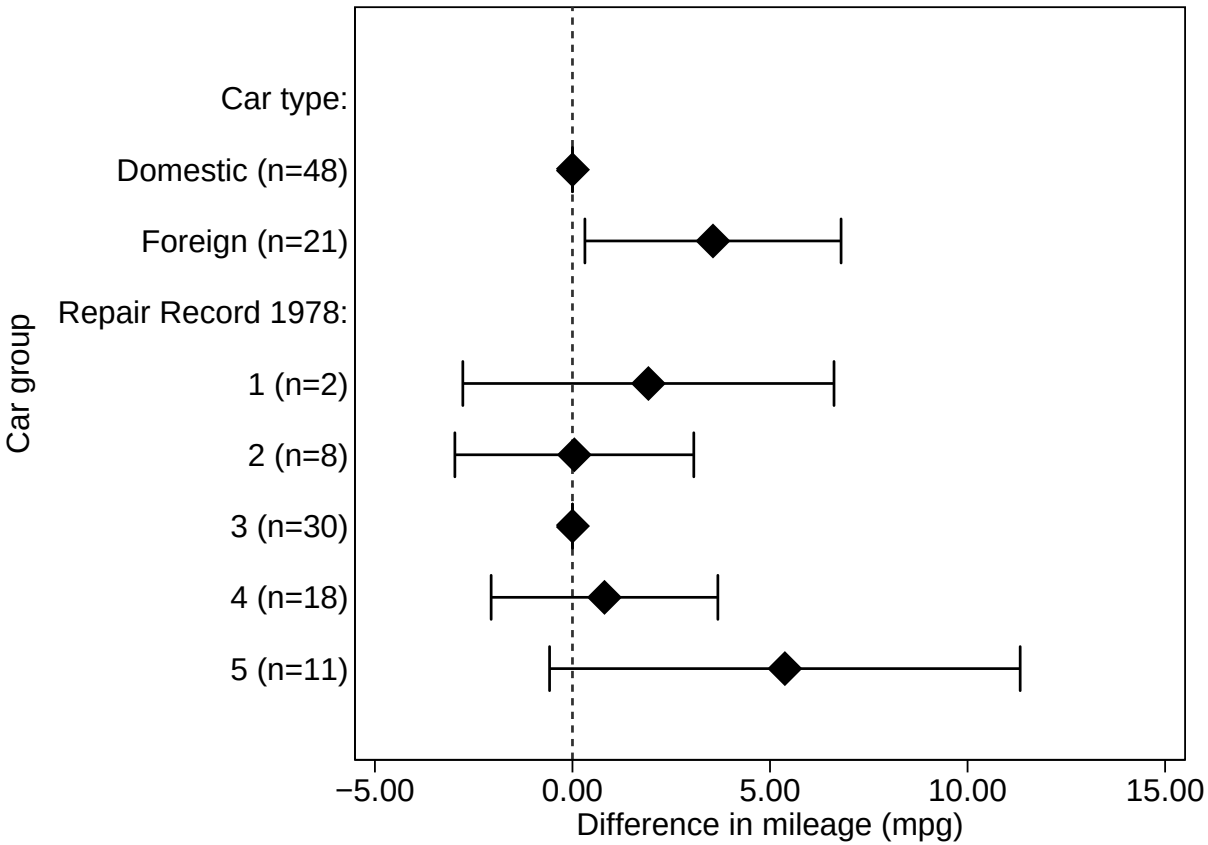
`example4.do` does not create this confidence interval plot

- This plot has group frequencies as well as group names.
- It also has reference groups (US-made and medium-reliability cars).



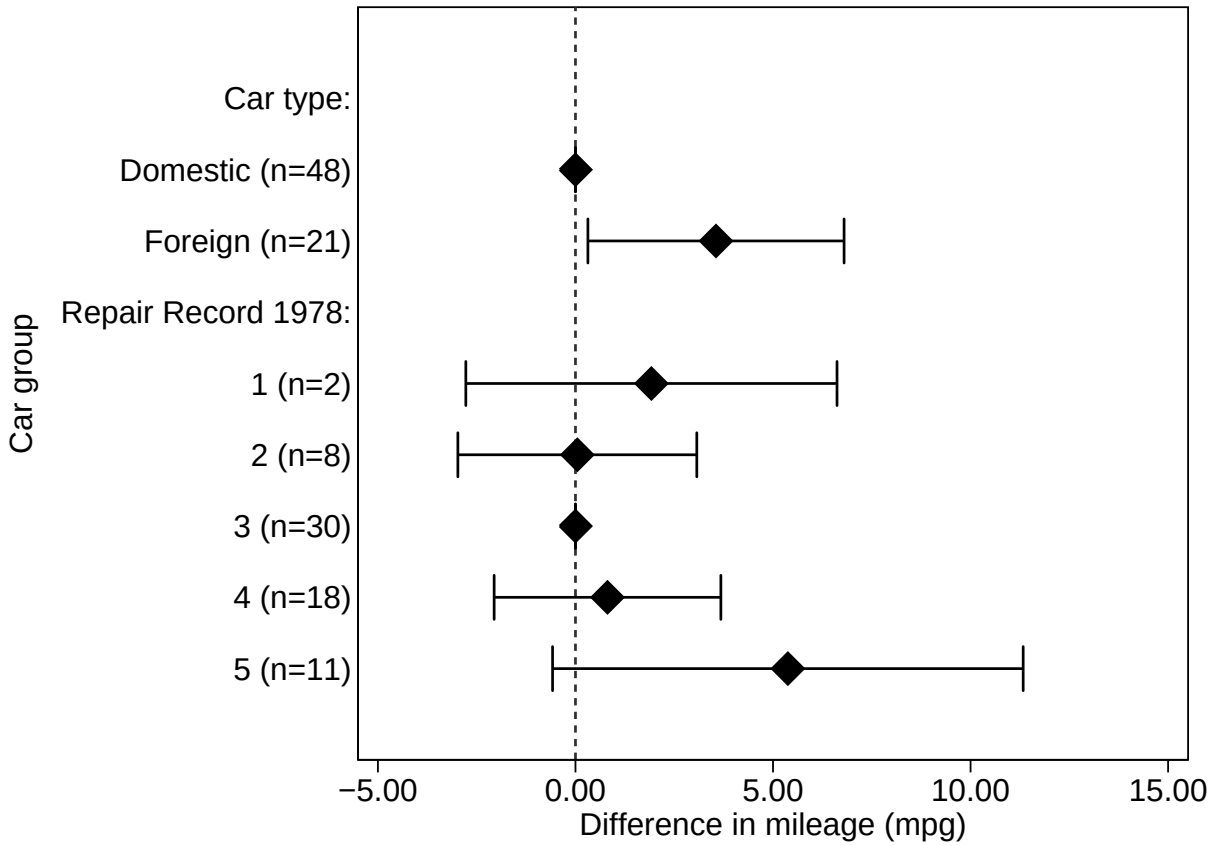
`example4.do` does not create this confidence interval plot

- This plot has group frequencies as well as group names.
- It also has reference groups (US-made and medium-reliability cars).
- It also has the factor names in gap rows, instead of repeating them.



`example4.do` does not create this confidence interval plot

- This plot has group frequencies as well as group names.
- It also has reference groups (US-made and medium-reliability cars).
- It also has the factor names in gap rows, instead of repeating them.
- The first two improvements are enabled by `xcontract`, and the third by `ingap`.



`example5.do`: Creating the plot and table that we saw earlier

`example5.do`: Creating the plot and table that we saw earlier

- Our final example is an improved version of `example4.do`.

`example5.do`: Creating the plot and table that we saw earlier

- Our final example is an improved version of `example4.do`.
- Before fitting the regression model, we create two `xcontract` resultssets, containing frequencies for the factors `foreign` and `rep78`.

`example5.do`: Creating the plot and table that we saw earlier

- Our final example is an improved version of `example4.do`.
- Before fitting the regression model, we create two `xcontract` resultssets, containing frequencies for the factors `foreign` and `rep78`.
- After fitting the regression model, we merge these two `xcontract` resultssets into the `parmby` resultsset.

`example5.do`: Creating the plot and table that we saw earlier

- Our final example is an improved version of `example4.do`.
- Before fitting the regression model, we create two `xcontract` resultssets, containing frequencies for the factors `foreign` and `rep78`.
- After fitting the regression model, we merge these two `xcontract` resultssets into the `parmby` resultsset.
- We generate the row label variable `cargp` (car group), this time using `ingap` to insert gap rows.

Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 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

Differences in mileage in the `auto` data (compared with US cars with a medium repair record of 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

`example5.do` may seem a complicated way to create a small plot and table. *However*, economies of scale become important with large plots and tables, multiple plots and tables, or multiple versions of the same plots and tables.

From datasets to resultssets in Stata

In this survey, we have learned about:

From datasets to resultssets in Stata

In this survey, we have learned about:

- Programs to generate resultssets.

From datasets to resultssets in Stata

In this survey, we have learned about:

- Programs to generate resultssets.
- Programs to subset and concatenate resultssets (often from `tempfiles`).

From datasets to resultssets in Stata

In this survey, we have learned about:

- Programs to generate resultssets.
- Programs to subset and concatenate resultssets (often from `tempfiles`).
- Programs for string-numeric and string-factor conversion in resultssets.

From datasets to resultssets in Stata

In this survey, we have learned about:

- Programs to generate resultssets.
- Programs to subset and concatenate resultssets (often from `tempfiles`).
- Programs for string-numeric and string-factor conversion in resultssets.
- Programs to output resultssets to presentation-ready plots and publication-ready tables.

From datasets to resultssets in Stata

In this survey, we have learned about:

- Programs to generate resultssets.
- Programs to subset and concatenate resultssets (often from `tempfiles`).
- Programs for string-numeric and string-factor conversion in resultssets.
- Programs to output resultssets to presentation-ready plots and publication-ready tables.

All these programs are downloadable from SSC. The overheads, handout and example do-files can be downloaded from the conference website at
http://www.stata.com/support/meeting/10uk/