b
UNIVERSITÄT
BERN

Faculty of Business, Economics and Social Sciences

Department of Social Sciences

University of Bern Social Sciences Working Paper No. 1

Plotting regression coefficients and other estimates in Stata

Ben Jann

Current version: January 31, 2014
First version: August 25, 2013

http://ideas.repec.org/p/bss/wpaper/1.html
http://econpapers.repec.org/paper/bsswpaper/1.htm

University of Bern
Department of Social Sciences
Fabrikstrasse 8
CH-3012 Bern

Tel. +41 (0)31 631 48 11
Fax +41 (0)31 631 48 17
info@sowi.unibe.ch
www.sowi.unibe.ch

Plotting regression coefficients and other estimates in Stata

Ben Jann
Institute of Sociology
University of Bern
`ben.jann@soz.unibe.ch`

January 31, 2014

Abstract

Graphical presentation of regression results has become increasingly popular in the scientific literature, as graphs are much easier to read than tables in many cases. In Stata such plots can be produced by the `marginsplot` command ([R] **marginsplot**). However, while `marginsplot` is very versatile and flexible, it has two major limitations: it can only process results left behind by `margins` ([R] **margins**) and it can only handle one set of results at the time. In this article I introduce a new command called `coefplot` that overcomes these limitations. It plots results from any estimation command and combines results from several models into a single graph. The default behavior of `coefplot` is to plot markers for coefficients and horizontal spikes for confidence intervals. However, `coefplot` can also produce various other types of graphs. The capabilities of `coefplot` are illustrated in this article using a series of examples.

Keywords: `coefplot`, `marginsplot`, `margins`, regression plot, coefficients plot

Contents

1 Introduction

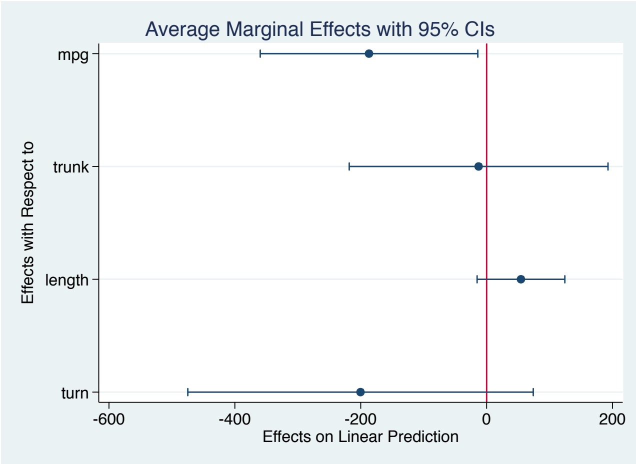
Tabulating regression coefficients has long been the preferred way of communicating results from statistical models. However, researchers now more and more employ graphs to present regression results. This has several reasons. On the one hand, interpretation of regression tables can be very challenging, especially if there are interaction effects, categorical variables, or nonlinear functional forms. Moreover, in nonlinear models, the original regression coefficients are often not the primary interest of researchers. For example, in logistic regression the raw coefficients represent effects on log odds. However, most people would be more comfortable with effects expressed on the probability scale. Since probability effects are not constant in such a model, it can be helpful, for example, to plot effect functions. On the other hand, and more fundamentally, it has been recognized that presentation of results in form of graphs can me much more effective than tabulation.

While graphics have always been very present in science, one type of plot has become increasingly popular. In such a plot, regression coefficients or other statistics of interest are displayed as markers accompanied by spikes indicating confidence intervals. See Kastellec and Leoni (2007) for some examples. Creating such graphs in Stata is tedious (although see Newson, 2003). The coefficients and variances have to be gathered from the `e()`-returns, confidence intervals have to be computed, and the results have to be appropriately stored as variables in the data set. Then a suitable variable for the category axis has to be generated and coefficient labels have to be defined. Finally, a complicated graph command has to be issued to plot the coefficients and confidence intervals. With the introduction of `marginsplot` ([R] **marginsplot**) in Stata 12 this task has been greatly simplified. It is now possible to plot coefficients and confidence intervals with just a few lines of code. For example, consider the following linear regression model ([R] **regress**):

```
. sysuse auto, clear
(1978 Automobile Data)

. regress price mpg trunk length turn

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  4,    69) =    5.79
       Model |   159570047     4  39892511.8           Prob > F      =  0.0004
    Residual |   475495349    69  6891236.94           R-squared     =  0.2513
-------------+------------------------------           Adj R-squared =  0.2079
       Total |   635065396    73  8699525.97           Root MSE      =  2625.1

------------------------------------------------------------------------------
       price |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         mpg |  -186.8417   88.17601    -2.12   0.038     -362.748   -10.93533
       trunk |  -12.72642   104.8785    -0.12   0.904    -221.9534    196.5005
      length |   54.55294   35.56248     1.53   0.130    -16.39227    125.4981
        turn |  -200.3248   140.0166    -1.43   0.157    -479.6502    79.00066
       _cons |   8009.893   6205.538     1.29   0.201    -4369.817     20389.6
------------------------------------------------------------------------------
```

To plot the regression coefficients (which, in this case, are equal to the average marginals effects), we could type:

```
. margins, dydx(*)
  (output omitted)
. marginsplot, horizontal xline(0)
>     yscale(reverse) recast(scatter)
  (output omitted)
```



`marginsplot` is a very versatile command that can do much more than what is shown above, especially when plotting predictive margins, the area of application `marginsplot` was primarily designed for. However, two main drawbacks prevent `marginsplot` from being easily employed as a general tool for plotting coefficients or other estimation results. First, `marginsplot` can only process results left behind by `margins` ([R] **margins**). Second, `marginsplot` can only deal with one set of results at the time (i.e. the results from one call of `margins`).

I therefore wrote a command that can be applied to any estimation results and can combine results from several estimation sets into one graph. The new `coefplot` command can be seen as a graphical equivalent to popular tabulation programs such as `outreg` (Gallup, 2012) or `estout` (Jann, 2007).

To install the `coefplot` package, type

```
. ssc install coefplot, replace
```

Stata 11 or newer is required. After installation, type

```
. help coefplot
```

to view the help file.

2 Syntax

The basic syntax of `coefplot` is:

`coefplot` *subgraph* [`||` *subgraph* ...] [, *globalopts*]

where *subgraph* is defined as

`(`*plot*`)` [`(`*plot*`)` ...] [, *subgropts*]

and *plot* is either `_skip` (to skip a plot) or

model [`\` *model* ...][, *plotopts*]

and *model* is either

name [, *modelopts*]

where *name* is the name of a stored model (see [R] **estimates**; type `.` or leave blank to refer to the active model) or matrix(*mspec*) [, *modelopts*]

to plot results from a matrix (see [P] **matrix**) where *mspec* may be:

name	to use the first row of matrix *name*
name[#,.]	to use row # of matrix *name*; may also type *name*[#,] or *name*[#]
name[.,#]	to use column # of matrix *name*; may also type *name*[,#]

Parentheses around *plot* can be omitted if *plot* does not contain spaces.

coefplot has four levels of options: (1) *modelopts* are options that apply to a single model (or matrix). They specify the information to be collected and displayed. (2) *plotopts* are options that apply to a single plot, possibly containing results from multiple models. They affect the rendition of markers and confidence intervals and provide a label for the plot. (3) *subgropts* are options that apply to a single subgraph, possibly containing multiple plots. (4) *globalopts* are options that apply to the overall graph. The options levels are nested in the sense that upper level options include all lower level options. That is, *globalopts* includes *subgropts*, *plotopts*, and *modelopts*; *subgropts* includes *plotopts* and *modelopts*; *plotopts* includes *modelopts*. If lower level options are specified at an upper level, they serve as defaults for all included lower levels elements. These defaults, however, are overwritten by options specified at a lower level.

The following sections give a brief overview of the available options. For a more detailed description see the see the online help.

2.1 Model options

Main

omitted includes omitted coefficients.

baselevels includes base levels of factor variables.

b(*mspec*) specifies the source to be plotted; default is to plot e(b); *mspec* may be:

name	to use the first row of e(*name*)
name[#,.]	to use row # of e(*name*); may also type *name*[#,] or *name*[#]
name[.,#]	to use column # of e(*name*); may also type *name*[,#]

at[(*mspec*)] retrieves plot positions from e(at) or as specified by *mspec*, where *mspec* is:

name	to use the first row of e(*name*)
name[#,.]	to use row # of e(*name*); may also type *name*[#,] or *name*[#]
name[.,#]	to use column # of e(*name*); may also type *name*[,#]
#	to use the #th at-dimension (for results from margins only)
matrix(*mspec*)	to read from a matrix instead of e()
_coef	to use coefficient names as plot positions
_eq	to use equation names as plot positions

keep(*coeflist*) keeps specified coefficients, where *coeflist* is a space-separated list of elements such as:

coef	coefficient *coef*
eq:	all coefficients from equation *eq*
eq:*coef*	coefficient *coef* from equation *eq*

eq and *coef* may contain * (any string) and ? (any nonzero character) wildcards.

drop(*coeflist*) drops specified coefficients, where *coeflist* is as above for keep().

rename(*matchlist*) renames coefficients, where *matchlist* is

coeflist = *newname* [*coeflist* = *newname* ...]

and *coeflist* is as above for keep(), except that wildcards are only allowed in *eq* and *coef* can be specified as *prefix** to replace a prefix or **suffix* to replace a suffix.

eqrename(*matchlist*) renames equations, where *matchlist* is *eqlist* = *newname* [*eqlist* = *newname* ...]

and *eqlist* is a space separated list of equation names; type equation names as *prefix*`*` to replace a prefix or `*`*suffix* to replace a suffix.

`asequation(`*string*`)` sets the equation for all coefficients to *string*.

`eform`[`(`*coeflist*`)`] plots exponentiated coefficients, where *coeflist* is as above for `keep()`.

`rescale(`*spec*`)` rescales coefficients; *spec* is either # or

coeflist = # [*coeflist* = # ...]

where *coeflist* is as above for `keep()`.

`swapnames` swaps coefficient names and equation names.

Confidence intervals

`noci` omits confidence intervals.

`levels(`*numlist*`)` sets level(s) for conficence intervals; default is `level(95)` or as set by `set level` (see [R] **level**).

`ci(`*spec*`)` provides confidence intervals, where *spec* is

cispec [*cispec* ...]

and *cispec* is either *name* to use rows 1 and 2 of `e(`*name*`)` or `(`*mspec* *mspec*`)` (identifying lower and upper bounds, respectively) with *mspec* as above for `b()`. *cispec* may also be # for a specific confidence level as in `levels()` above.

`v(`*name*`)` retrieves variances from the diagonal of `e(`*name*`)`; default is to use `e(V)`.

`se(`*mspec*`)` provides standard errors, where *mspec* is as above for `b()`.

`df(`*spec*`)` provides degrees of freedom, where *spec* is either # or *mspec* as above for `b()`.

`citype(logit|normal)` determines the method to compute confidence intervals; default is `citype(normal)`. Type `citype(logit)` to use the logit transformation to compute confidence intervals (only relevant for proportions).

2.2 Plot options

Main

modelopts are plot-specific model options as described above.

`label(`*string*`)` specifies a label to be used for the plot in the legend.

`offset(`#`)` provides a custom offset for the plot positions.

`pstyle(`*pstyle*`)` determines overall style of the plot; see [G] ***pstyle***.

`axis(`#`)` specifies the scale axis to be used for the plot, where $1 \leq \# \leq 9$.

Rendition of coefficient markers

marker_options change the look of markers (color, size, etc.); see [G] ***marker_options***.

`mlabel` adds coefficient values as marker labels.

marker_label_options change the look and position of marker labels; see [G] ***marker_label_options***.

`recast(`*plottype*`)` plots the markers using *plottype*; supported plot types are `scatter` (default), `line`, `connected`, `area`, `bar`, `spike`, `dropline`, and `dot`.

Rendition of confidence intervals

`cionly` plots confidence intervals only (i.e. omits coefficient markers).

`citop` draws confidence intervals in front of markers.

`ciopts(`*options*`)` affect the rendition of confidence intervals; *options* are *line_ options* to change the look of the spikes (see [G] ***line_ options***), `pstyle(`*pstyle*`)` to set the overall style, and `recast(`*plottype*`)` to set the plot type; supported plot types are `rspike` (default), `rarea`, `rbar`, `rcap`, `rcapsym`, `rscatter`, `rline`, and `rconnected`. [G] ***stylelists*** may be used within options in case of multiple confidence intervals.

`cismooth`[`(`*options*`)`] adds smoothed confidence intervals. *options* are `n(`#`)` to set the number of (equally) spaced confidence levels, default is `n(50)`; `lwidth(`*min max*`)` to set the range of (relative) line widths, default is `lwidth(2 15)`; and `intensity(`*min max*`)` to set the range of color intensities, default is `intensity(`*min* `100)`, where *min* depends on `n()` and is about 14 for `n(50)`.

2.3 Subgraph options

modelopts and *plotopts* are subgraph-specific model and plot options as described above.

`bylabel(`*string*`)` specifies a label to be used for the subgraph.

2.4 Global options

Main

modelopts, *plotopts*, and *subgropts* are global model, plot, and subgraph options as described above.

`horizontal` places coefficient values on the X axis; this is the general default.

`vertical` places coefficient values on the Y axis; this is the default with `at()`.

`order(`*coeflist*`)` orders coefficients, where *coeflist* is as above for `keep()`; may specify `.` instead of *coef* to introduce gaps; not allowed with `at()`.

`relocate(`*spec*`)` repositions coefficients, where *spec* is

[*eq*`:`]*coef* `=` # [[*eq*`:`]*coef* `=` # ...]

`bycoefs` arranges subgraphs by coefficients; not allowed with `at()`.

`norecycle` increments plot styles across subgraphs.

`grid(`*spec*`)` determines where grid lines are placed; not allowed with `at()`; *spec* may be `between` (grid lines between coefficients), `within` (grid lines within coefficients), or `none` (omit grid lines).

`nooffsets` suppresses automatic offsets of plot positions.

`format(`*format*`)` sets the display format for coefficient values; *format* may be a numeric format or a date format as described in [D] **format**.

Labels

`nolabels` uses variable names instead of labels.

`coeflabels(`*spec*`)` specifies custom labels for coefficients; not allowed with `at()`; *spec* is

[*coeflist* `=` *label* [*coeflist* `=` *label* ...]] [`,` `truncate(`#`)` `wrap(`#`)` `nobreak` *suboptions*]

where *coeflist* is as above for `keep()`, `truncate(`#`)` truncates coefficient labels to a maximum length of # characters, `wrap(`#`)` divides coefficient labels into multiple lines with a maximum length of # characters, `nobreak` prevents splitting long words, and *suboptions* are axis label suboptions as described in [G] ***axis_ label_ options***.

noeqlabels suppresses equation labels.

eqlabels(*spec*) specifies custom labels for equations; not allowed with at(); *spec* is

[*label* [*label* ...]] [, [no]gap[(#)] asheadings offset(#) truncate(#) wrap(#) nobreak *suboptions*]

gap() specifies the gap between equations; default is gap(1). asheadings treats equation labels as headings; see headings(). offset(#) offsets the labels by # (only allowed with asheadings). truncate(), wrap(), and nobreak are as described above under coeflabels(). *suboptions* are axis label suboptions as described in [G] ***axis_label_options***.

eqstrict specifies to be strict about equations.

headings(*spec*) adds headings between coefficients; not allowed with at(); *spec* is

coeflist = *label* [*coeflist* = *label* ...] [, [no]gap[(#)] offset(#) truncate(#) wrap(#) nobreak *suboptions*]

where *coeflist* is as above for keep(). gap() specifies the gap before headings; default is gap(1). offset(#) offsets the headings by #. truncate(), wrap(), and nobreak are as described above under coeflabels(). *suboptions* are axis label suboptions as described in [G] ***axis_label_options***.

groups(*spec*) adds labels for groups of coefficients; not allowed with at(); *spec* is

coeflist = *label* [*coeflist* = *label* ...] [, [no]gap[(#)] truncate(#) wrap(#) nobreak *suboptions*]

where *coeflist* is as above for keep(). gap() specifies the size of the gaps before and after the groups; default is gap(1). truncate(), wrap(), and nobreak are as described above under coeflabels(). *suboptions* are axis label suboptions as described in [G] ***axis_label_options***.

plotlabels(*spec*) specifies labels for the plots to be used in the legend; *spec* is

[*label* [*label* ...]] [, truncate(#) wrap(#) nobreak]

where truncate(), wrap(), and nobreak are as described above under coeflabels().

bylabels(*spec*) specifies labels for the subgraphs; *spec* is

[*label* [*label* ...]] [, truncate(#) wrap(#) nobreak]

where truncate(), wrap(), and nobreak are as described above under coeflabels().

Save results

generate[(*prefix*)] generates variables containing the graph data; default prefix is coefplot_.

replace overwrites existing variables.

Add plots

addplot(*plot*) adds other plots to the graph; see [G] ***addplot_option***.

Y axis, X axis, Titles, Legend, Overall, By

twoway_options are twoway options, other than by(), as documented in [G] ***twoway_options***.

byopts(*byopts*) determines how subgraphs are combined; *byopts* are as described in [G] ***by_option***.

3 Examples

3.1 Plotting a single model

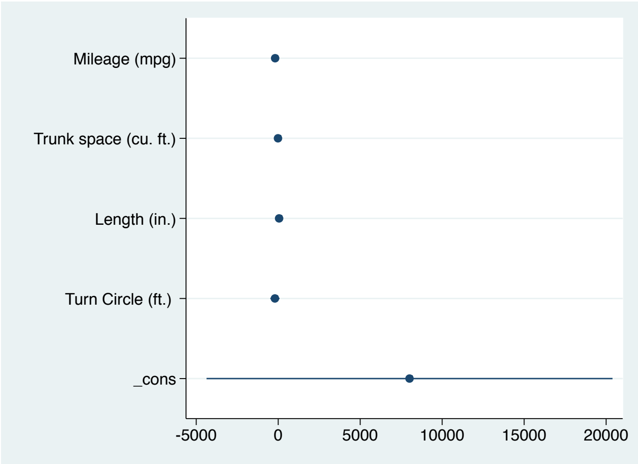
The syntax to produce a plot of the coefficients of a single model is

`coefplot` [*name*] [, *options*]

where *name* is the name of a stored model (see [R] **estimates**), or `.` or empty string for the active model.

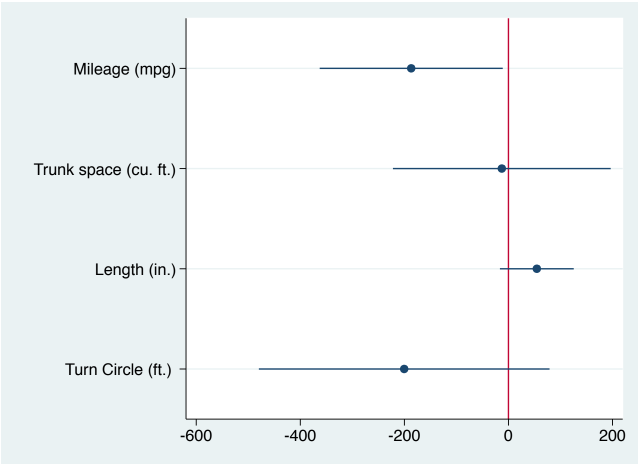
For example, to plot coefficients and 95% confidence intervals for the most recent model, type:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot
```



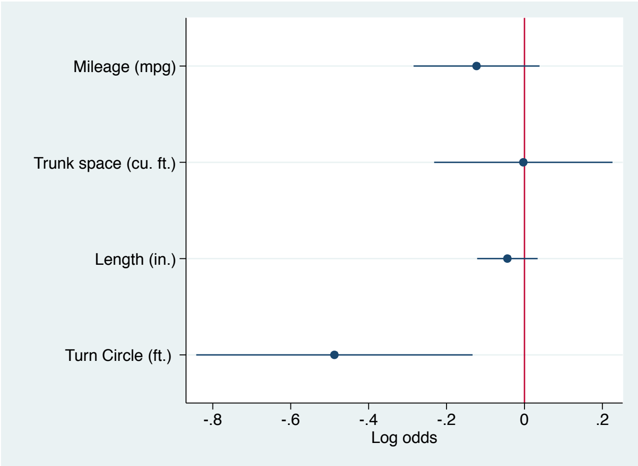
We may not be interested in the constant, so we can add `drop(_cons)` to remove it. Furthermore, `xline(0)` will add a reference line at zero so we can better see which coefficients are significantly different from zero:

```
. coefplot, drop(_cons) xline(0)
```



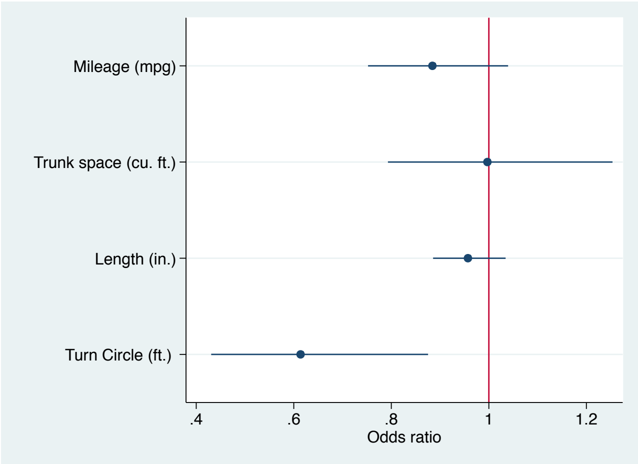
`coefplot` can graph results from almost any estimation command. For example, to plot coefficients from a logit model ([R] **logit**), type:

```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>     xtitle(Log odds)
```



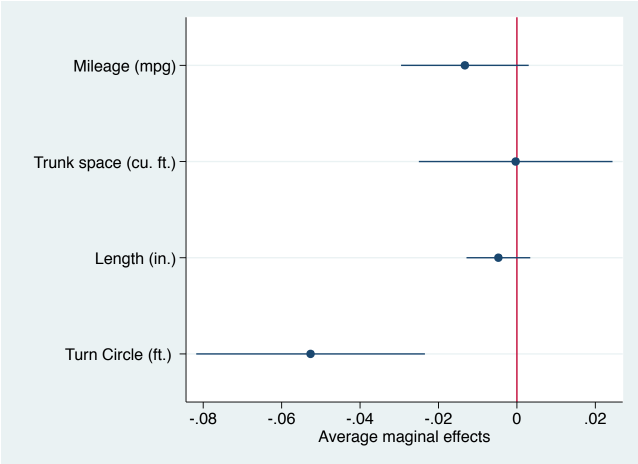
With logit models one is often interested in odds ratios instead of the raw coefficients. To plot odds ratios instead of log odds, use the `eform` option that causes `coefplot` to compute exponents of coefficients and confidence intervals:

```
. coefplot, drop(_cons) xline(1) eform
>     xtitle(Odds ratio)
```



Furthermore, if you want to plot average marginal effects instead of log odd or odds ratios, you can apply `margins` (see [R] **margins**):

```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg trunk length turn
  (output omitted)
. margins, dydx(*) post
  (output omitted)
. coefplot, xline(0)
>     xtitle(Average maginal effects)
```



It is essential to specify the `post` option with `margins` so that it posts its results in `e()`, from where `coefplot` collects the results to be displayed. If you do not specify the `post` option then `margins` leaves e() unchanged and coefplot uses the raw coefficients from the logit model that still reside in e().

3.2 Plotting multiple models

To include results from several commands in one graph save the results from each command using **estimates store** (see [R] **estimates**) and then provide the names of the stored estimation sets to coefplot. Three options to include multiple results in the graph are offered. First, models can be included as different "plots" in the same graph. By a "plot" I mean a set of markers and confidence spikes using same plot style. Second, separate subgraphs can be created, each containing one or more plots. Third, multiple models can be appended into the same "plot".

3.2.1 Models as plots

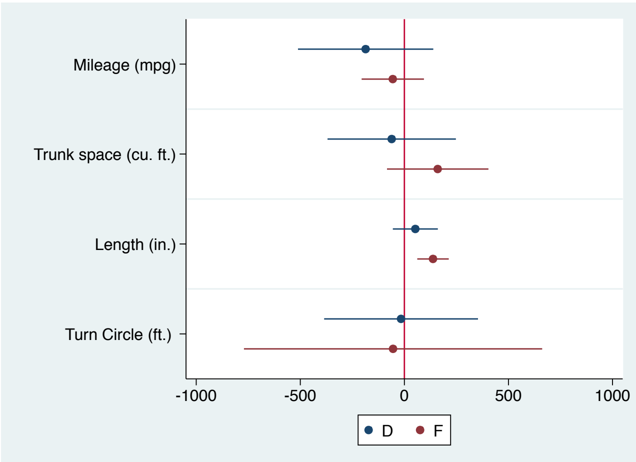
The syntax to include multiple models as separate plots is

coefplot [(]*name*[, *plotopts*)] [(*name*, *plotopts*) ...] [, *globalopts*]

where *name* is again the name of a stored model, or . or empty string for the active model. *plotopts* are options that apply to a single plot. They specify the information to be collected, affect the rendition of the plot, and provide a label for the plot in the legend. *globalopts* are options that apply to the overall graph, such as titles or axis labels, but may also contain any options allowed as plot options to provide defaults for the single plots.

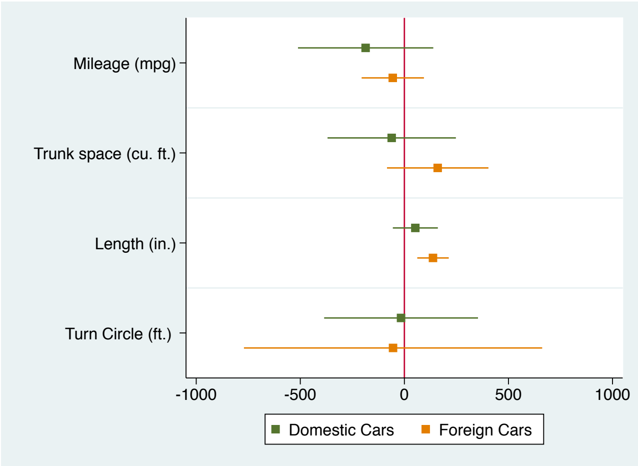
A basic example is as follows:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
>     if foreign==0
  (output omitted)
. estimates store D
. regress price mpg trunk length turn
>     turn if for==1
  (output omitted)
. estimates store F
. coefplot D F, drop(_cons) xline(0)
```



To specify separate options for an individual model, enclose the model and its options in parentheses. For example, to add a label for each plot in the legend, to use alternative plot styles, and to change the marker symbol, you could type:

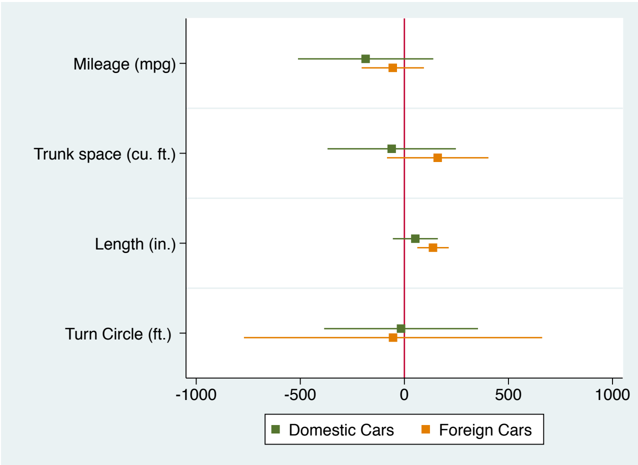
```
. coefplot (D, label(Domestic Cars)
>             pstyle(p3))
>          (F, label(Foreign Cars)
>             pstyle(p4))
>     , msymbol(S) drop(_cons) xline(0)
```



I specified `msymbol()` as a global option so that the same symbol is used for in both plots. To use different symbols, include an individual `msymbol()` option for each plot.

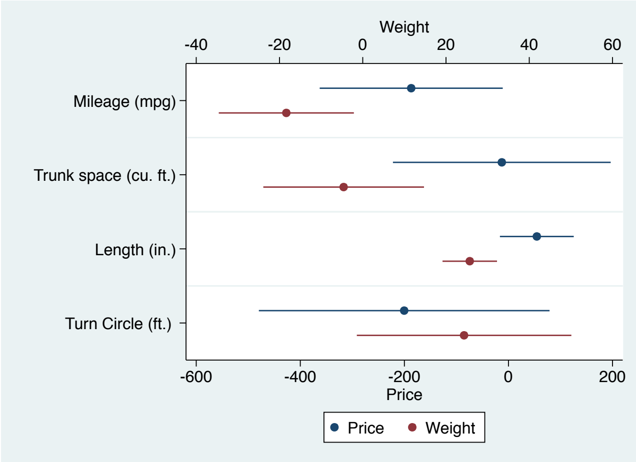
`coefplot` offsets the plot positions of the coefficients so that the confidence spikes do not overlap. To deactivate the automatic offsets, you can specify global option `nooffsets`. Alternatively, custom offsets may be specified by the `offset()` option (if `offset()` is specified for at least one model, automatic offsets are disabled). The spacing between coefficients is one unit, so usually offsets between -0.5 and 0.5 make sense. For example, if you want to use smaller offsets than the default, you could type:

```
. coefplot (D, label(Domestic Cars)
>             pstyle(p3) offset(0.05))
>          (F, label(Foreign Cars)
>             pstyle(p4) offset(-0.05))
>     , msymbol(S) drop(_cons) xline(0)
```



If the dependent variables of the models you want to include in the graph have different scales, it can be useful to employ the `axis()` plot option to assign specific axes to the models. For example, to include a regression on price and a regression on weight in the same graph, type:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. estimates store Price
. regress weight mpg trunk length turn
  (output omitted)
. estimates store Weight
. coefplot Price (Weight, axis(2)),
>     drop(_cons)
>     xtitle(Price) xtitle(Weight, axis(2))
```



3.2.2 Subgraphs

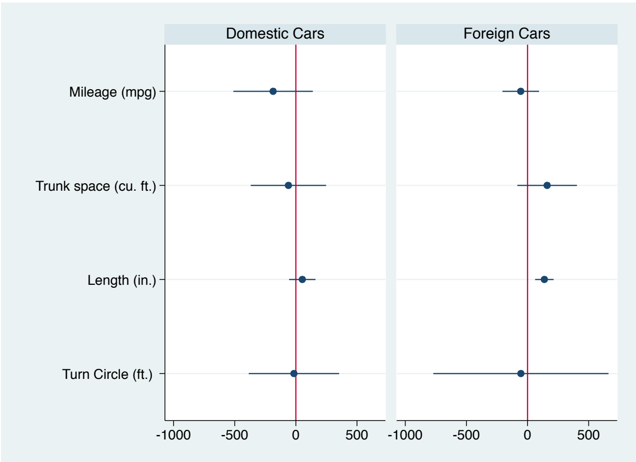
The syntax to create subgraphs is

`coefplot` *plotlist* [, *subgropts*] `||` [*plotlist*, *subgropts* `||` ...] [, *globalopts*]

where *plotlist* is a list of plots as in section 3.2.1 and *subgropts* are options that apply to a single subgraph.

An example with one model per subgraph is:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
>     if foreign==0
  (output omitted)
. estimates store D
. regress price mpg trunk length turn
>     turn if for==1
  (output omitted)
. estimates store F
. coefplot D, bylabel(Domestic Cars)
>       || F, bylabel(Foreign Cars)
>       ||, drop(_cons) xline(0)
```



An example with multiple models per subgraph is:

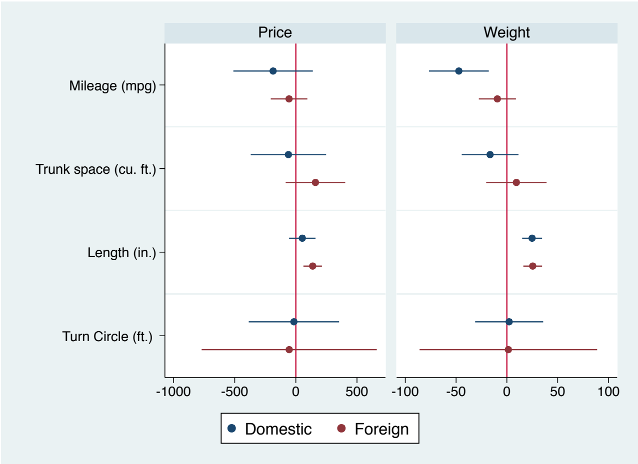
```
. regress weight mpg trunk length turn
>     if foreign==0
```

(output omitted)

```
. estimates store wD
. regress weight mpg trunk length turn
>     if foreign==1
```

(output omitted)

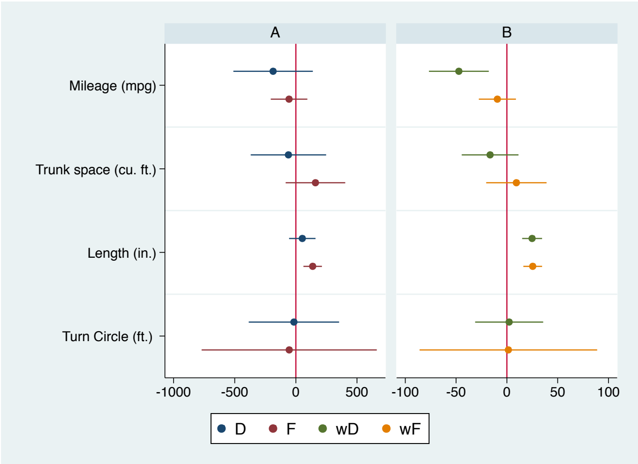
```
. estimates store wF
. coefplot
>     (D, label(Domestic))
>     (F, label(Foreign)), bylabel(Price)
>     || wD wF, bylabel(Weight)
>     ||, drop(_cons) xline(0)
>         byopts(xrescale)
```



Option `byopts(xrescale)` was specified so that each subgraph can have its own scale.

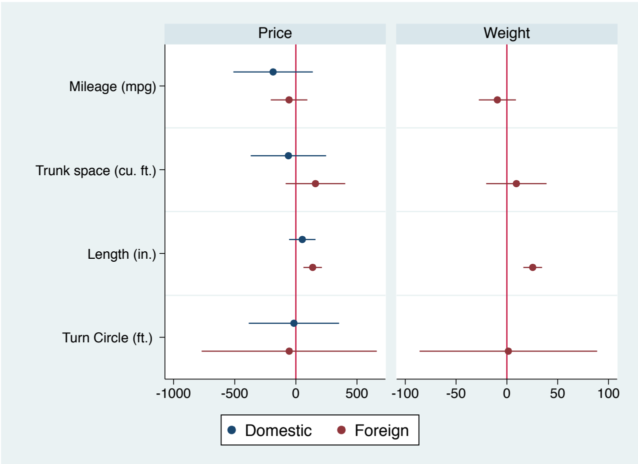
In the example above, plot labels for the legend were set within the first subgraph. They could also have been specified within the second subgraph, as plot styles are recycled with each new subgraph and plot options are collected across subgraphs. To prevent recycling of plot styles, add the `norecycle` option:

```
. coefplot D F, bylabel(A)
>     || wD wF, bylabel(B)
>     ||, drop(_cons) xline(0)
>       norecycle byopts(xrescale)
>       legend(rows(1))
```



Furthermore, to leave a plot position empty in one of the subgraphs, you can specify `_skip` in place of a plot:

```
. coefplot
>     (D, label(Domestic))
>     (F, label(Foreign)), bylabel(Price)
>     || _skip wF, bylabel(Weight)
>     ||, drop(_cons) xline(0)
>         byopts(xrescale)
```



3.2.3 Appending models

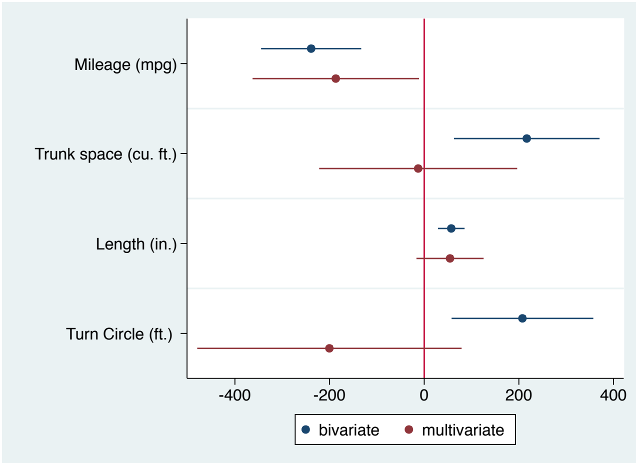
The syntax to append models within the same plot is

`coefplot` `(`*name*[`,` *modelopts*] `\` [*name*`,` *modelopts* `\` ...][`,` *plotopts*]`)` [...]

where *name* is again the name of a stored model, or `.` or empty string for the active model, and *modelopts* are options that apply to a single model.

For example, if you want to draw a graph comparing bivariate and multivariate effects, you could type:

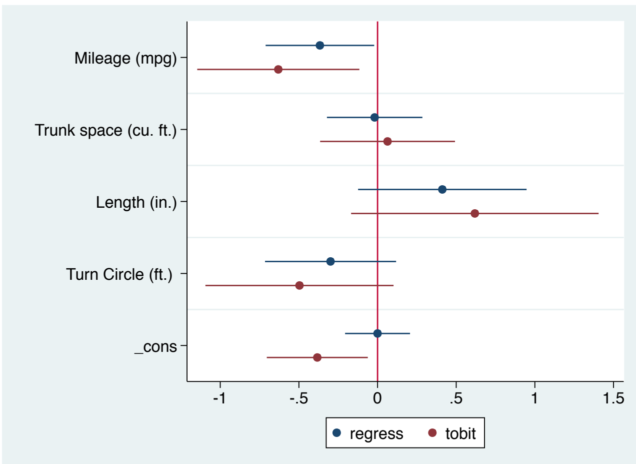
```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. estimates store multivariate
. foreach var in mpg trunk length turn {
  2.      quietly regress price `var´
  3.      estimates store `var´
  4. }
. coefplot (mpg \ trunk \ length \ turn,
>     label(bivariate)) (multivariate)
>     , drop(_cons) xline(0)
```



3.2.4 How coefficients and equations are matched

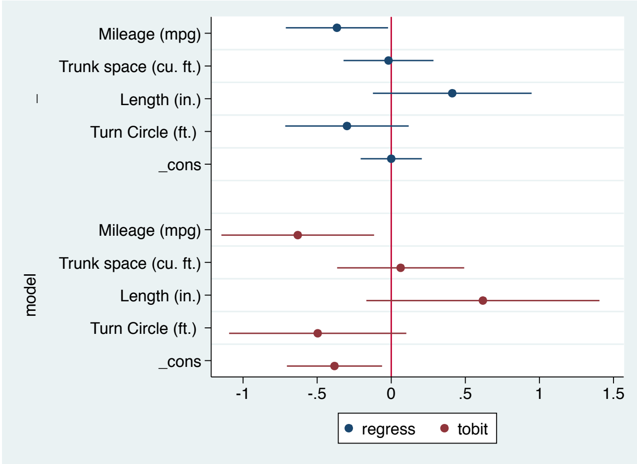
The default for `coefplot` is to use the first (nonzero) equation from each model and match coefficients across models by their names (ignoring equation names). For example, `regress` returns one (unnamed) equation containing the regression coefficients whereas `tobit` ([R] **tobit**) returns two equations, equation "`model`" containing the regression coefficients and equation "`sigma`" containing the standard error of the regression. Hence, the default for `coefplot` is to match the regression coefficients from the two models and ignore equation "`sigma`" from the Tobit model:

```
. sysuse auto, clear
(1978 Automobile Data)
. foreach v of var price mpg trunk length
>     turn {
  2.      quietly summarize `v´
  3.      quietly replace `v´ =
>           (`v´ - r(mean)) / r(sd)
  4. }
. regress price mpg trunk length turn
  (output omitted)
. estimate store regress
. tobit price mpg trunk length turn, ll(-.5)
  (output omitted)
. estimate store tobit
. coefplot regress tobit, xline(0)
```



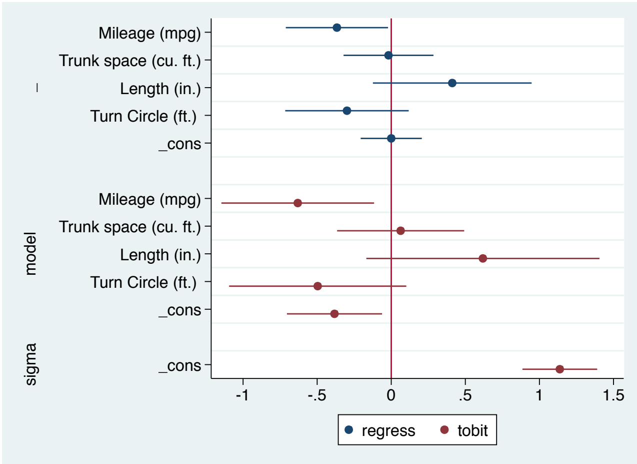
Even though the collected results from `regress` and `tobit` have different equation names ("`_`" and "`model`", respectively), `coefplot` matches their coefficients, that is, the equation names are ignored. This is the default if only one equation per model is collected. If you want to take equation names into account nonetheless, you can specify the `eqstrict` option:

```
. coefplot regress tobit, xline(0)
>     eqstrict
```



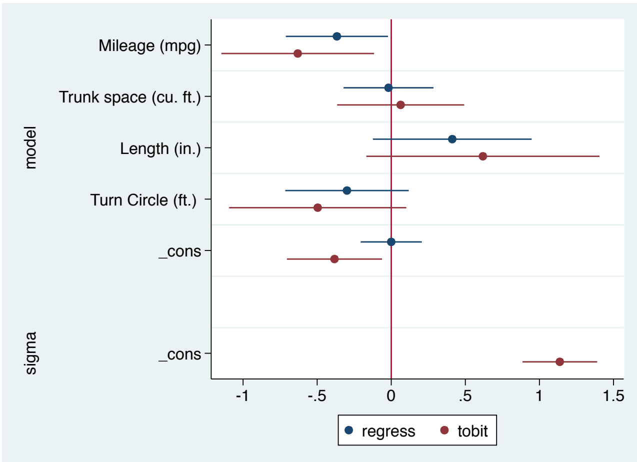
Although eqstrict causes equation names to be relevant, the second equation from the tobit model is still ignored. To include all equations, type:

```
. coefplot regress tobit, xline(0)
>     keep(*:)
```



Furthermore, to match the coefficients from regress with the first equation from tobit and also print the second equation from tobit, you can use asequation() to set the equation name of regress to "model":

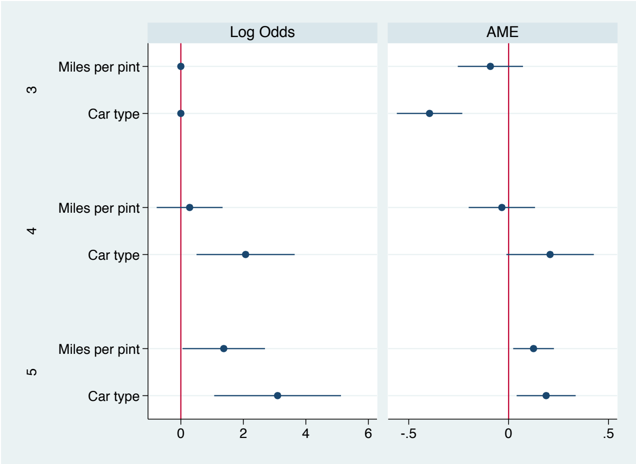
```
. coefplot (regress, asequation(model))
>          (tobit, keep(*:))
>          , xline(0)
```



Alternatively, you could also use eqrename(_ = model) to rename equation "_" to "model" or eqrename(model = _) to rename equation "model" to "_".

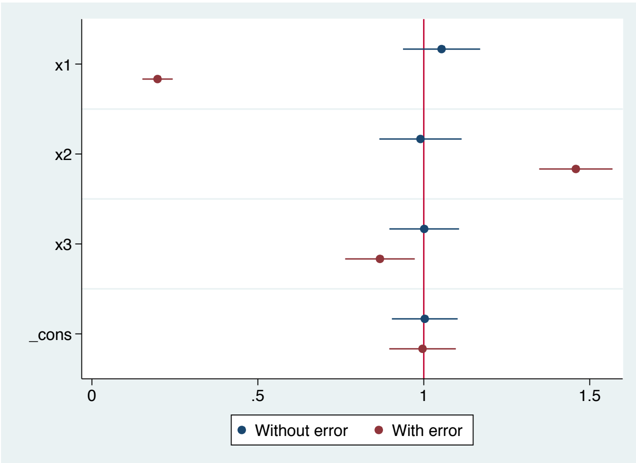
Another application of the `asequation()` option is when you want to assign equations to results from `margins`. The following example shows how to plot log odds of a multinomial logit ([R] **mlogit**) along with average marginal effects:

```
. sysuse auto, clear
(1978 Automobile Data)
. gen mpp = mpg/8
. label variable mpp "Miles per pint"
. mlogit rep78 mpp foreign if rep>=3
  (output omitted)
. estimates store mlogit
. forvalues i = 3/5 {
  2.      quietly margins, dydx(*)
>             predict(outcome(`i´)) post
  3.      estimates store ame`i´
  4.      quietly estimates restore mlogit
  5. }
. coefplot mlogit, keep(*:) drop(_cons)
>             omitted bylabel(Log Odds)
>     || (ame3, aseq(3) \ ame4, aseq(4)
>        \ ame5, aseq(5)), bylabel(AME)
>     ||, xline(0) byopts(xrescale)
```



Finally, if you want to match coefficients that have different names in the input models, you can apply the `rename()` option. Here is an example that illustrates the effect of measurement error in regression models:

```
. drop _all
. matrix C = (  1, .5,   0 \ .5,   1, .3
>              \  0, .3,   1 )
. drawnorm x1 x2 x3, n(10000) corr(C)
(obs 10000)
. generate y = 1 + x1 + x2 + x3 +
>             5 * invnorm(uniform())
. regress y x1 x2 x3
  (output omitted)
. estimates store m1
. generate x1err = x1 +
>             2 * invnorm(uniform())
. regress y x1err x2 x3
  (output omitted)
. estimates store m2
. coefplot (m1, label(Without error))
>          (m2, label(With error)), xline(1) rename(x1err = x1)
```

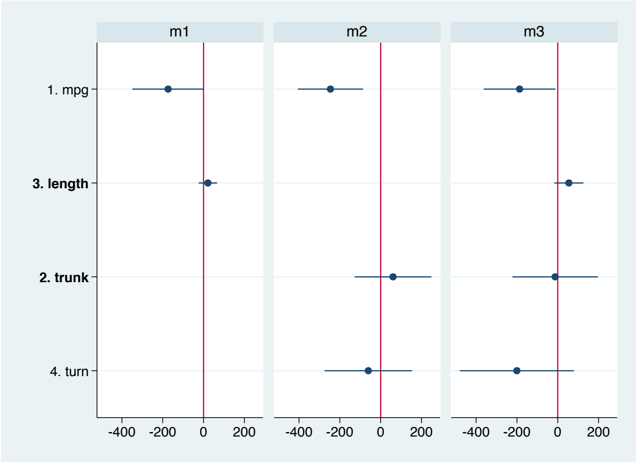


We can see how measurement error on `x1` distorts all slope coefficients in the model, even for variable `x3` that is uncorrelated with `x1` (due to the indirect correlation through `x2`).

3.2.5 How coefficients are ordered

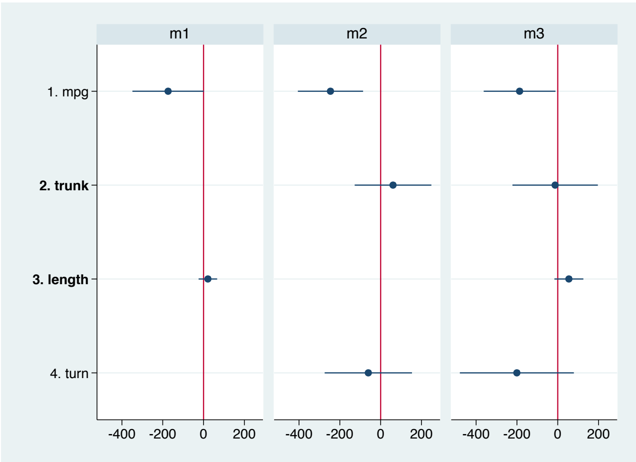
In general, coefficients are plotted in the same order (from top to bottom) as they appear in the input models. However, coefficients appearing only in later models are placed after coefficients from earlier models (with the exception of `_cons`, which is always placed last). Have a look at the following example:

```
. sysuse auto, clear
(1978 Automobile Data)
. label variable mpg    "1. mpg"
. label variable trunk  "{bf:2. trunk}"
. label variable length "{bf:3. length}"
. label variable turn   "4. turn"
. regress price mpg length
  (output omitted)
. estimate store m1
. regress price mpg trunk turn
  (output omitted)
. estimate store m2
. regress price mpg trunk length turn
  (output omitted)
. estimate store m3
. coefplot m1 || m2 || m3, xline(0) drop(_cons) byopts(row(1))
```



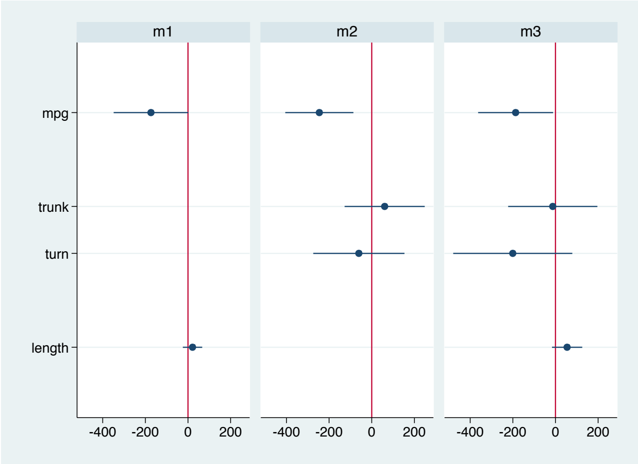
Even though in the full model (`m3`) `trunk` comes before `length`, the order of the two coefficients is reversed in the plot. This is because `length` but not `trunk` is part of the first model. That is, because `trunk` only appears in the later models, it is placed after `length` that appears already in the first model. To establish an order as in model `m3`, you can use the `order()` option:

```
. coefplot m1 || m2 || m3, xline(0)
>     drop(_cons) byopts(row(1))
>     order(mpg trunk length)
```



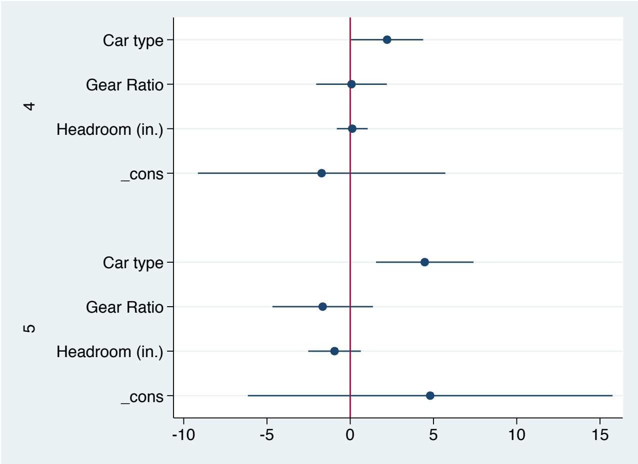
Within `order()`, you can use the `*` (any string) and `?` (any nonzero character) wildcards. Furthermore, you can type `.` to insert gaps (but also see the section on headings and groups below). Example:

```
. label variable mpg
. label variable trunk
. label variable length
. label variable turn
. coefplot m1 || m2 || m3, xline(0)
>     drop(_cons) byopts(row(1))
>     order(. mpg . t* . length .)
```



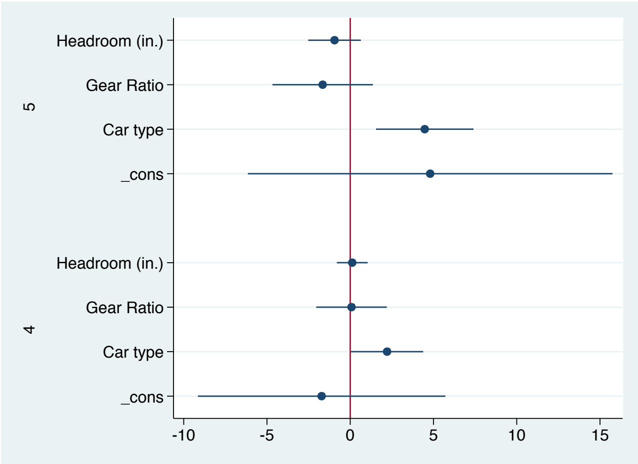
In case of multiple equation models, the default is to order coefficients by equations:

```
. sysuse auto, clear
(1978 Automobile Data)
. mlogit rep78 headroom gear_ratio foreign
>     if rep>=3
```
  (output omitted)
```
. coefplot, xline(0) keep(*:)
>     order(foreign gear* head*)
```

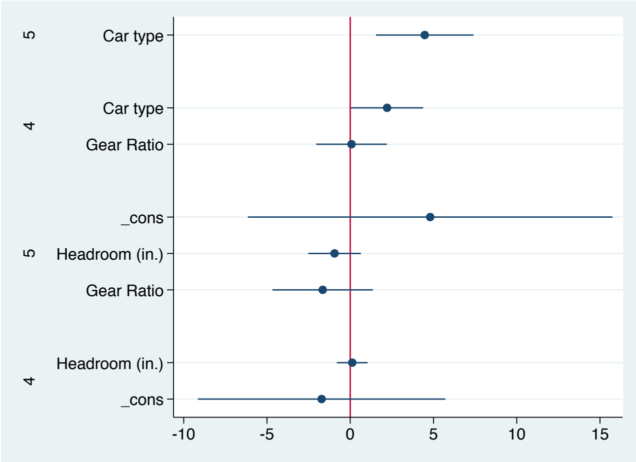


To reorder equations, to apply different orderings within equations, or to break equations apart, specify equation names within `order()`, as in the following examples:

```
. coefplot, xline(0) keep(*:) order(5: 4:)
```



```
. coefplot, xline(0) keep(*:)
>     order(5:foreign 4:foreign gear*
>           5:_cons *)
```

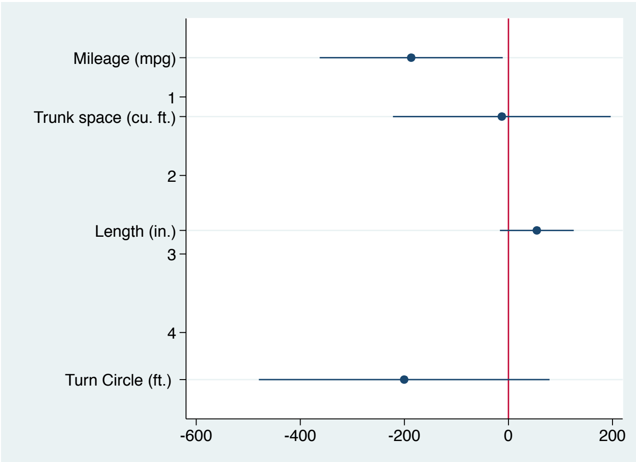


In the second example, `headroom` and `_cons` from equation 4 are placed last because they are remaining coefficients that have not been listed in `order()`.

Note that equation names have to be specified either for all elements in `order()` or for none. Hence, for example, typing `order(foreign 4:_cons)` would be invalid.

By default, coefficients (and gaps, if specified) are placed at integer values on the categorical axis (starting with 1 from top to bottom). If you want to place coefficients at nonstandard values, you can apply the `relocate()` option. `relocate()` is an end-of-pipe option, that is, after the categorical axis has been set up in the usual way, `relocate()` moves the specified coefficients, leaving empty the original positions. Here is an example:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, xline(0) drop(_cons)
>     ylabel(1(1)4, add)
>     relocate(mpg = 0.5 trunk = 1.25
>         length = 2.7 turn = 4.6)
```

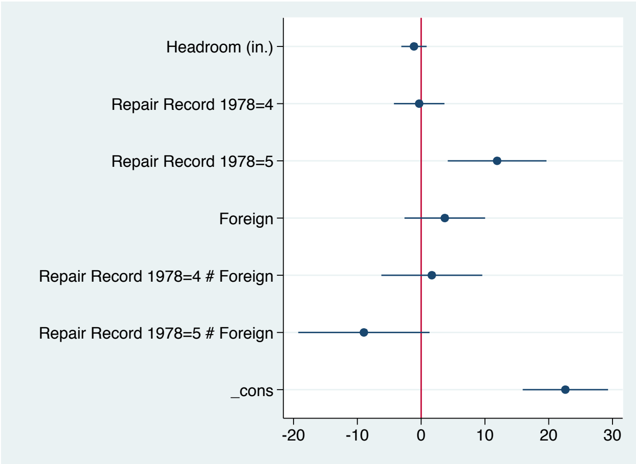


To illustrate the effect of `relocate()` the original plot positions have been marked with labels “1”, “2”, “3”, and “4” in the example.

3.3 Labeling the categorical axis

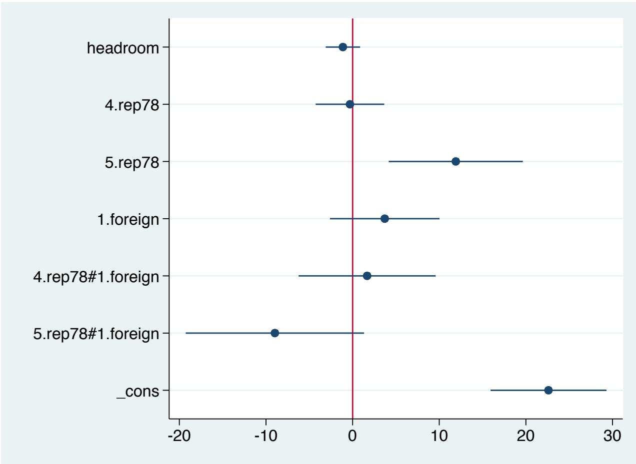
`coefplot` looks for variables corresponding to the collected coefficient names and then uses their variable labels for the categorical axis. For factor variables, `coefplot` additionally takes value labels into account (the rule is to print the value label, if a value label is defined, and otherwise print the variable label or name along with the level). Here is an example with categorical variables and interaction terms:

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0)
```



To use coefficient names instead of variable labels, specify the `nolabels` option:

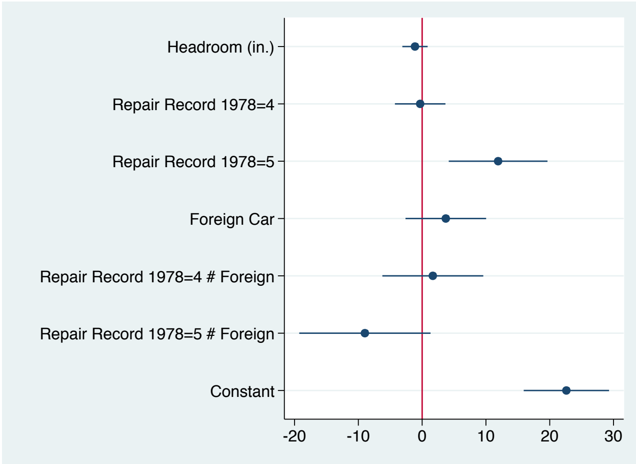
```
. coefplot, xline(0) nolabels
```



3.3.1 Custom coefficient labels

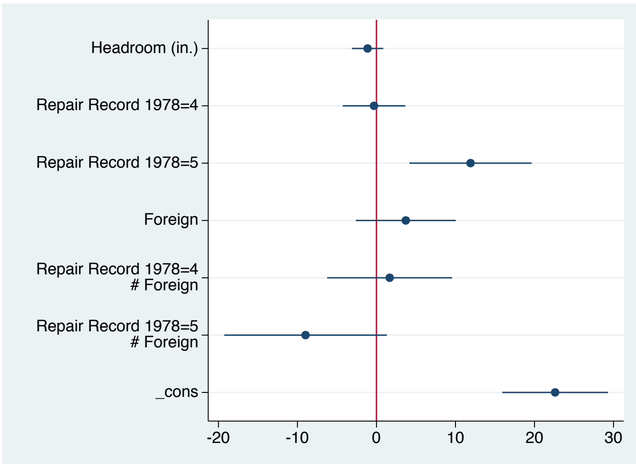
An easy way to provide labels for the coefficients is to define appropriate variable and value labels before applying `coefplot` (see [R] **label**). However, not all coefficients have corresponding variables (e.g. `_cons`). To provide labels for such coefficients or to assign custom labels to coefficients without manipulating variable labels, use the `coeflabels()` option:

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0)
>     coeflabel(1.foreign = "Foreign Car"
>               _cons = "Constant")
```



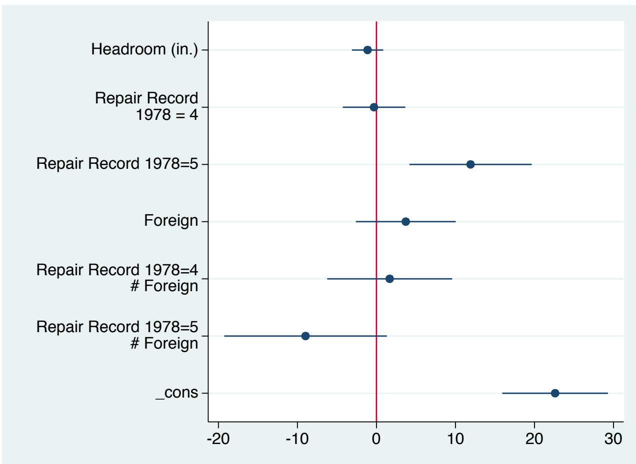
`coeflabels()` has a `wrap()` and a `truncate()` suboption to deal with long labels. These suboptions apply to all coefficient labels, whether they are automatically generated or provided within `coeflabels()`. For example, to limit the line with to 20 characters and wrap long labels to multiple lines, type:

```
. coefplot, xline(0) coeflabel(, wrap(20))
```



Multiline labels can also be created explicitly using compound double quotes within `coeflabels()`. Such labels will not be altered by `wrap()` or `truncate()`:

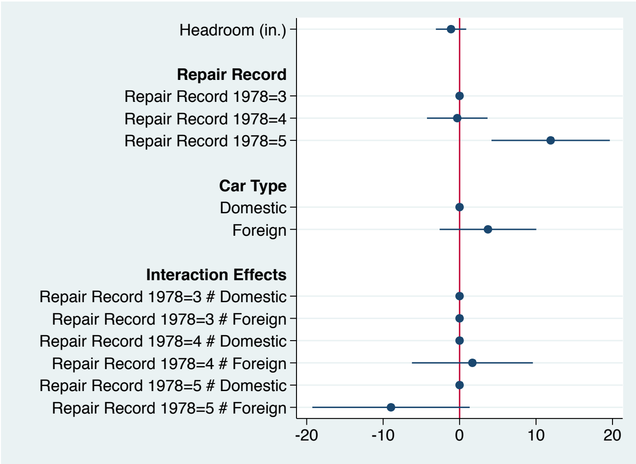
```
. coefplot, xline(0) coeflabel(4.rep78 =
>         `""Repair Record" "1978 = 4""'
>         , wrap(20))
```



3.3.2 Headings and groups

Sometimes it is useful to add headings between coefficients to better arrange a graph. This can be achieved by the `headings()` option:

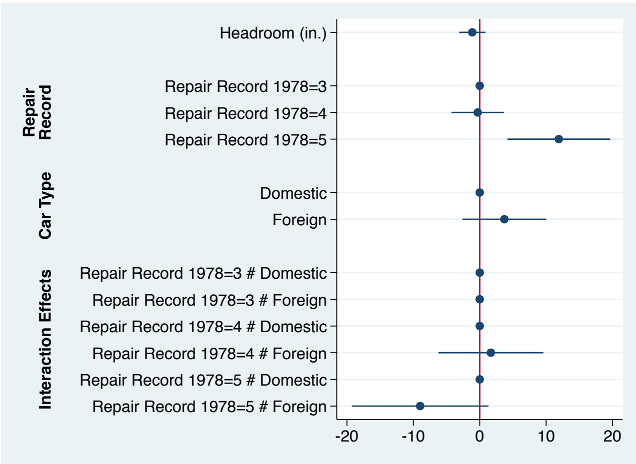
```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     headings(
>     3.rep78 = "{bf:Repair Record}"
>     0.foreign = "{bf:Car Type}"
>     3.rep78#0.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons)
```



In this example, `omit` requests to plot omitted coefficients and `baselevels` requests to plot base level coefficients. Omitted coefficients and base levels coefficients are always equal to zero, but in some cases it can be helpful to include them in a graph for reasons of clarity. The `{bf}` tag changes text to bold; see [G] **text** for details on text in graphs.

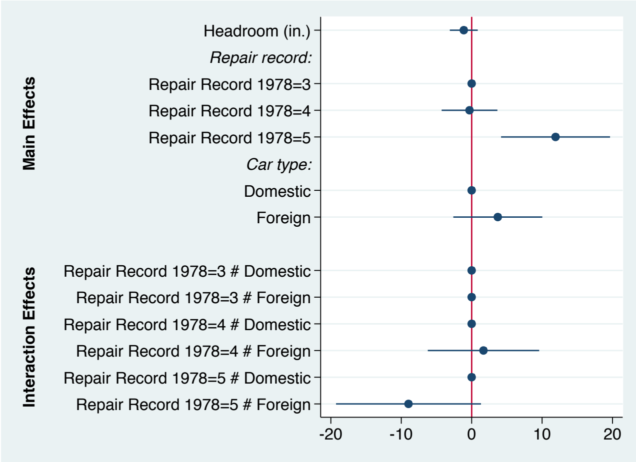
Instead of adding headings you can also define groups of coefficients and add group labels using the `groups()` option:

```
. coefplot, xline(0) omitted base
>     groups(?.rep78 =
>         `""{bf:Repair}" "{bf:Record}""'
>     ?.foreign = "{bf:Car Type}"
>     ?.rep78#?.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons)
```



Furthermore, `headings()` and `groups()` can be combined:

```
. coefplot, xline(0) omitted base
>     headings(
>     3.rep78 = "{it:Repair record:}"
>     0.foreign = "{it:Car type:}", nogap)
>     groups(headroom 1.foreign =
>         "{bf:Main Effects}"
>     ?.rep78#?.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons)
```

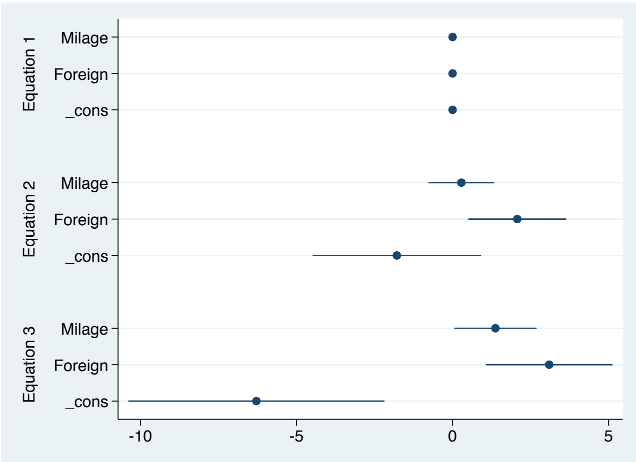


In this example, the nogap suboption was specified within headings() to prevent adding extra space before the headings.

3.3.3 Equation labels

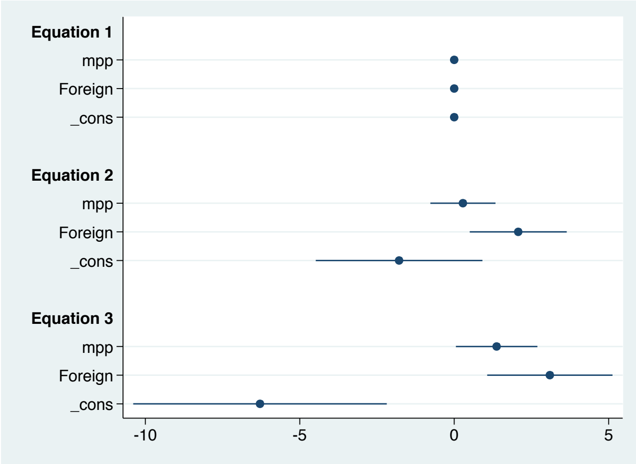
Equation labels provide yet another layer of labels. The default is to place the equation labels on the right hand side, similar to group labels:

```
. sysuse auto, clear
(1978 Automobile Data)
. gen mpp = mpg/8
. mlogit rep78 mpp i.foreign if rep>=3
  (output omitted)
. coefplot, omitted keep(*:)
>     coeflabels(mpp = "Milage")
>     eqlabels("Equation 1" "Equation 2"
>         "Equation 3")
```



However, you can also set the equation labels as headings between equations using the asheadings suboption:

```
. coefplot, omitted keep(*:)
>     coeflabels(mpg = "Milage")
>     eqlabels("{bf:Equation 1}"
>         "{bf:Equation 2}"
>         "{bf:Equation 3}", asheadings)
```

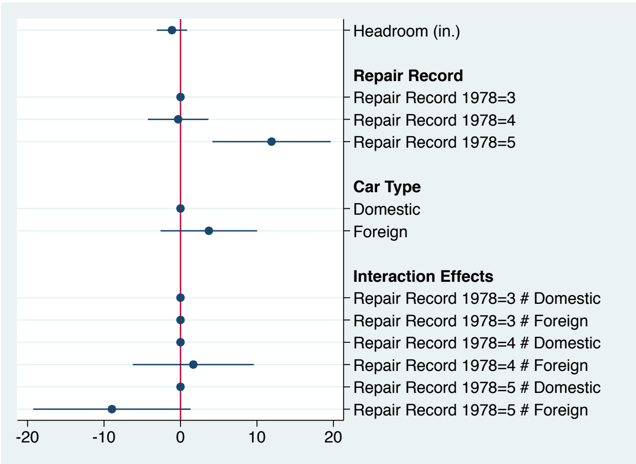


Note that, in this case, the `headings()` option is not allowed.

3.3.4 Labels on opposite side

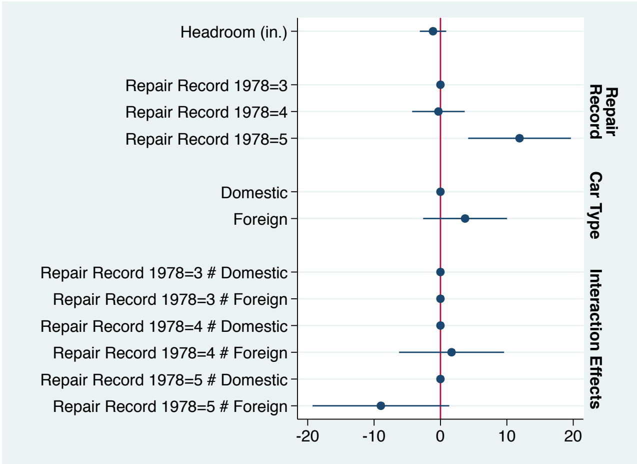
The default is to plot all labels on the left of the plot region. Use option `yscale(alt)` to move labels to the right (see [G] ***twoway_ options***):

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     headings(
>     3.rep78 = "{bf:Repair Record}"
>     0.foreign = "{bf:Car Type}"
>     3.rep78#0.foreign =
>         "{bf:Interaction Effects}")
>     drop(_cons) yscale(alt)
```



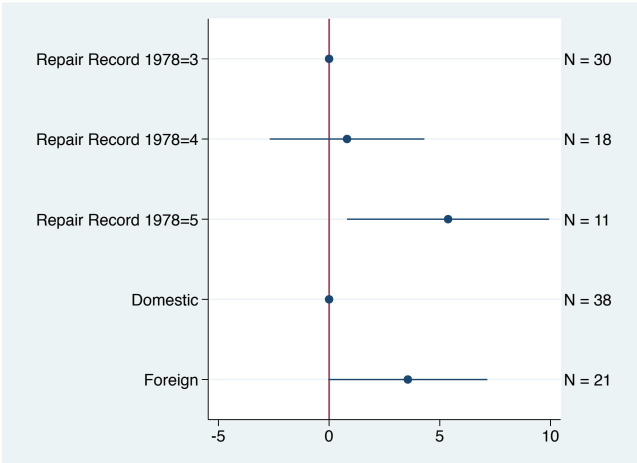
Group labels and equation labels are rendered as additional axes (axis 2 for group labels; axis 2 or 3 for equation labels, depending on whether groups were specified), so you have to employ the `axis()` suboption to move these:

```
. coefplot, xline(0) omitted base
>     groups(?.rep78 =
>         `""{bf:Repair}" "{bf:Record}""´
>     ?.foreign = "{bf:Car Type}"
>     ?.rep78#?.foreign =
>         "{bf:Interaction Effects}",
>         angle(rvertical))
>     drop(_cons) yscale(alt axis(2))
```



Moving group labels to the right can also be useful if you want to add an extra set of individual coefficient labels, without actually forming groups. Here is an example in which `groups()` is used to add information on the sample sizes of factor levels:

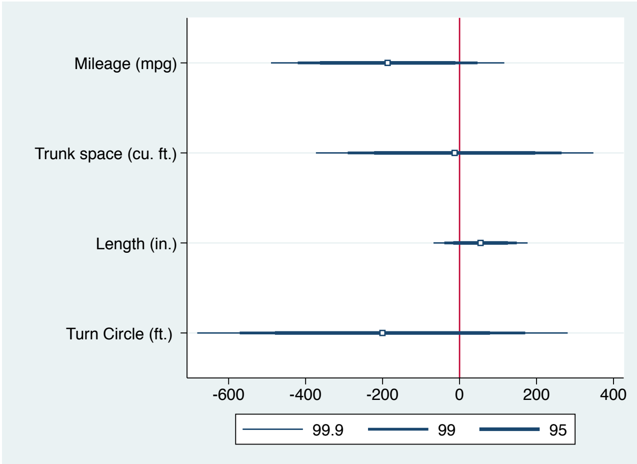
```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg i.rep i.foreign
  (output omitted)
. coefplot, xline(0) omitted baselevels
>     groups(3.rep78 = "N = 30"
>            4.rep78 = "N = 18"
>            5.rep78 = "N = 11"
>            0.foreign = "N = 38"
>            1.foreign = "N = 21"
>            , nogap angle(horizontal))
>    drop(_cons) yscale(alt axis(2))
```



3.4 Confidence intervals

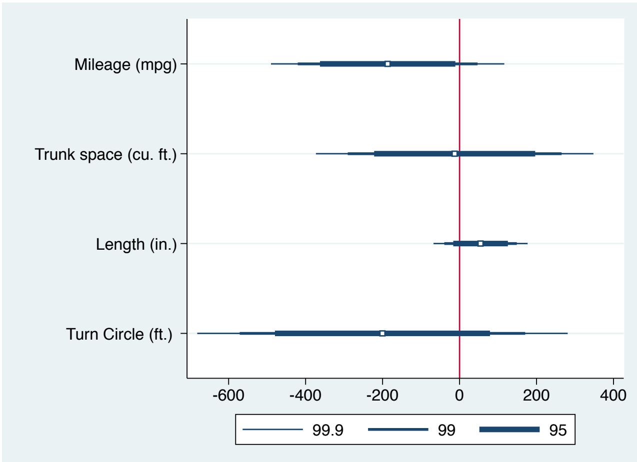
The default for `coefplot` is to draw spikes for 95% confidence intervals (or as set by [R] **level**). To specify a different level or to include multiple confidence intervals, use the `levels()` option. Here is an example with 99.9%, 99%, and 95% confidence intervals:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>     msymbol(s) mfcolor(white)
>     levels(99.9 99 95)
>     legend(order(1 "99.9" 2 "99" 3 "95")
>         row(1))
```



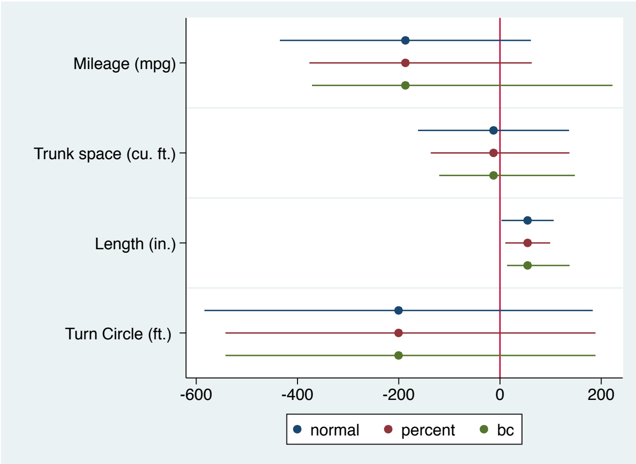
Line widths are (logarithmically) increased across the confidence intervals. To use different line widths specify the `lwidth()` suboption within `ciopts()`:

```
. coefplot, drop(_cons) xline(0)
>     msymbol(s) mfcolor(white)
>     levels(99.9 99 95)
>     legend(order(1 "99.9" 2 "99" 3 "95")
>         row(1))
>     ciopts(lwidth(*1 *2 *4))
```



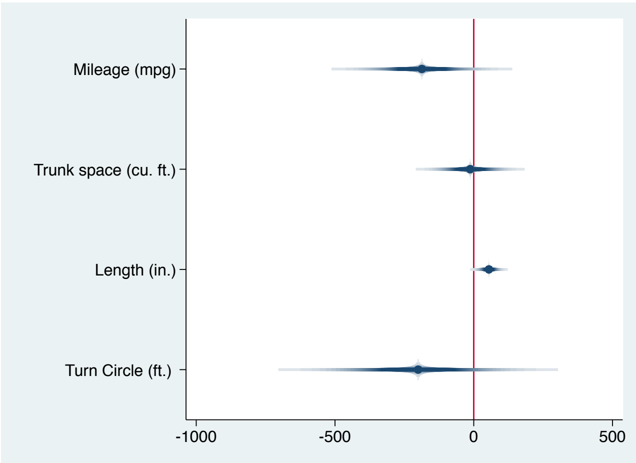
To compute confidence intervals, `coefplot` collects the variances of the coefficients from the diagonal of `e(V)` and then, depending on whether degrees of freedom are available in scalar `e(df_r)` (or, for estimates from [MI] **mi**, in matrix `e(df_mi)`), applies the standard formulas for confidence intervals based on the t-distribution or the normal distribution, respectively. If `e(V)` is not available, then `coefplot` looks for standard errors in vector `e(se)` and uses these for confidence interval computation. If a model does not provide degrees of freedom but you want to compute confidence intervals based on the t-distribution, you can provide the degrees of freedom through option `df()` (see the online help). If variances are stored in a matrix other than `e(V)`, use the `v()` option to provide the appropriate matrix name, or, alternatively use option `se()` to provide custom standard errors (in which case variances from `e(V)` will be ignored). Likewise, if your estimation command provides precomputed confidence intervals, use the `ci()` option to include them in the plot. For example, to plot the normal-approximation, percentile, and bias-corrected confidence intervals that are provided in `e(ci_normal)`, `e(ci_percentile)`, and `e(ci_bc)` by the bootstrap method, you could type:

```
. regress price mpg trunk length turn,
>     vce(bootstrap)
  (output omitted)
. coefplot
>     (, ci(ci_normal) label(normal))
>     (, ci(ci_percentile) label(percent))
>     (, ci(ci_bc) label(bc))
>     , drop(_cons) xline(0) legend(row(1))
```



In addition to `level()` and `ci()` you can also use option `cismooth` to add smoothed confidence intervals.[1] By default, `cismooth` generates confidence intervals for 50 equally spaced levels (1, 3, ..., 99) width graduated color intensities and varying line widths, as illustrated in the following example:

```
. coefplot, drop(_cons) xline(0)
>     cismooth grid(none)
```



The smoothed confidence intervals are produced independently from `levels()` and `ci()` and are not affected by `ciopts()`. Their appearance, however, can be set by a number of suboptions (see the online help). If `cismooth` is specified together with `levels()` or `ci()`, then the smoothed confidence intervals are placed behind the confidence intervals from `levels()` or `ci()`.

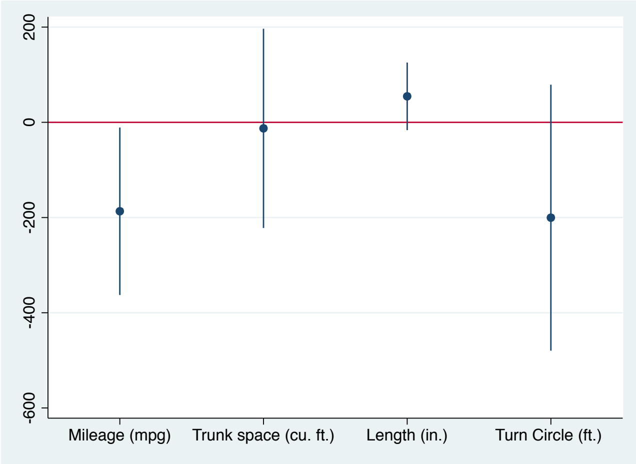
3.5 Alternate plot types and advanced examples

3.5.1 Vertical mode

By default, `coefplot` produces a horizontal graph with labels on the Y axis and values on the X axis. To flip axes specify the `vertical` option:

[1] The `cismooth` option has been inspired by code form David B. Sparks to produce smoothed confidence interval plots in R (see `http://dsparks.wordpress.com/2011/02/21/choropleth-tutorial-and-regression-coefficient-plots/`).

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) vertical yline(0)
```



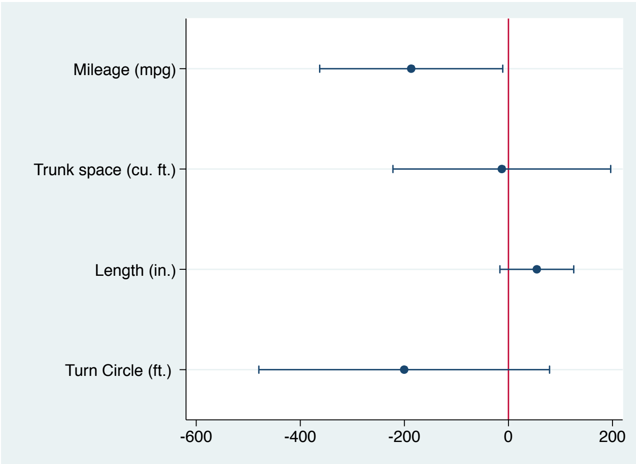
When changing from horizontal to vertical mode, options referring to specific axes need to be adjusted. This is why `yline(0)` was used in the example instead of `xline(0)` to draw the zero line.

3.5.2 Using the recast() option

To change the plot types used for coefficient markers and confidence intervals, you can use the `recast()` option. Available plot types for markers are standard twoway plots such as `scatter` (the default), `line`, `dot`, or `bar`. For confidence intervals use range plots such as `rspike` (the default), `rline`, `rcap`, or `rbar`.

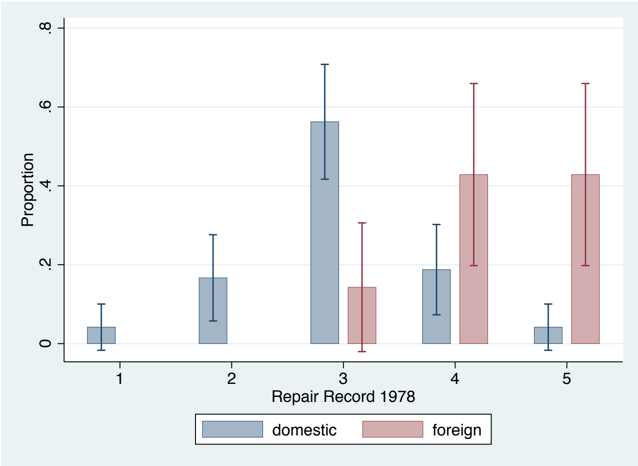
Capped spikes for confidence intervals For example, to display confidence intervals using capped spikes, you could type:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg trunk length turn
  (output omitted)
. coefplot, drop(_cons) xline(0)
>     ciopts(recast(rcap))
```



Bar charts of proportions Furthermore, a bar chart of proportions with capped confidence spikes can be produced as follows:

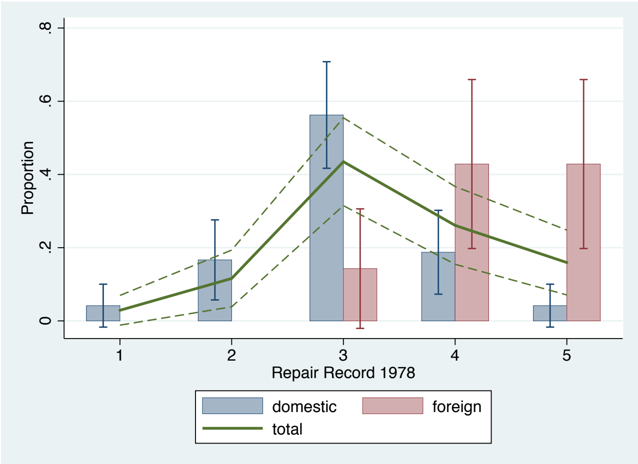
```
. sysuse auto, clear
(1978 Automobile Data)
. proportion rep if foreign==0
  (output omitted)
. estimates store domestic
. proportion rep if foreign==1
  (output omitted)
. estimates store foreign
. coefplot domestic foreign,
>     vertical recast(bar)
>     barwidth(0.25) fcolor(*.5)
>     ciopts(recast(rcap)) citop
>     xtitle(Repair Record 1978)
>     ytitle(Proportion)
```



In this example the `citop` option was used to prevent the lower limits of the confidence intervals from being hidden behind the bars.

Bars and lines Different plot types can be mixed, as the following example illustrates:

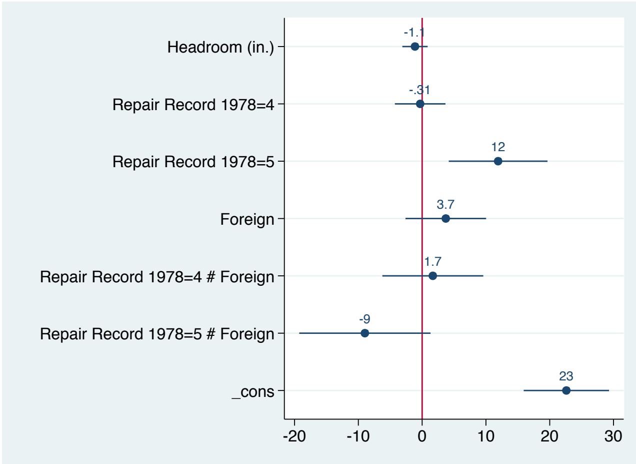
```
. proportion rep
  (output omitted)
. estimates store total
. coefplot
>     (domestic, offset(-.15) recast(bar)
>         barwidth(0.3) fcolor(*.5)
>         ciopts(recast(rcap)) citop)
>     (foreign, offset(.15) recast(bar)
>         barwidth(0.3) fcolor(*.5)
>         ciopts(recast(rcap)) citop)
>     (total, offset(0) recast(line)
>         lwidth(*2) ciopts(recast(rline)
>         lpattern(dash)))
>     , xtitle(Repair Record 1978)
>       ytitle(Proportion) vertical
```



3.5.3 Adding marker labels

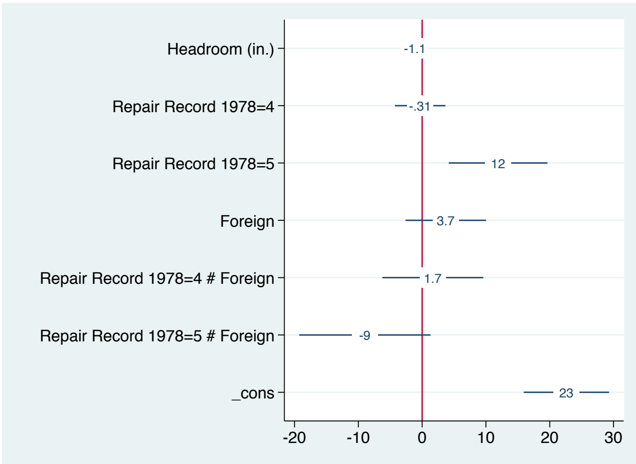
To add the values of the coefficients as marker labels, use the `mlabel` option, possibly together with `format()` to set the display format:

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(12) mlabgap(*2)
```



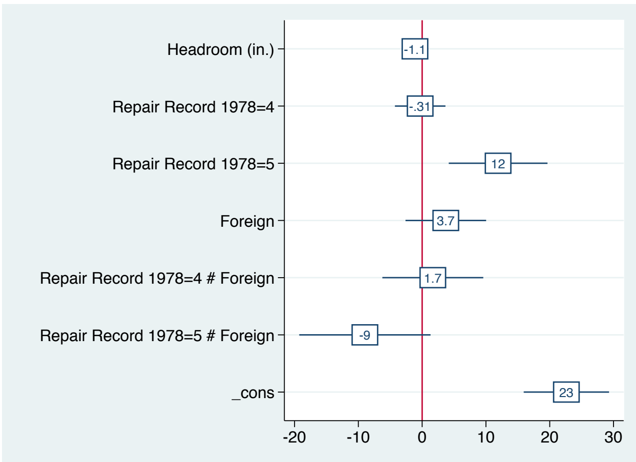
Stata graphs do not support background colors for marker labels, which makes labels unreadably if you place them on top of the markers using `mlabposition(0)`. However, here is a workaround. The trick is to add a second “confidence interval” that is a bar of fixed width (the dot in the suboptions within `ciopts()` specifies the “default” style; see [G] **stylelists**):

```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. mata: st_matrix("e(box)",
>     (st_matrix("e(b)") :- 2 \
>      st_matrix("e(b)") :+ 2))
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(0) msymbol(i)
>     ci(95 box) ciopts(recast(. rbar)
>         barwidth(. 0.35) color(. white))
```



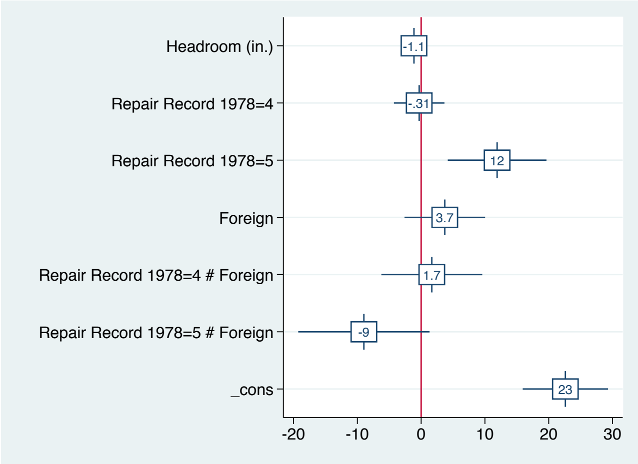
Here is a further example where a box is placed around the numbers:

```
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(0) msymbol(i)
>     ci(95 box) ciopts(recast(. rbar)
>         barwidth(. 0.35) fcolor(. white)
>         lwidth(. medium))
```



A bit unfortunate might be that due to the box the exact location of the coefficient can no longer be seen in the graph. Here is an example where an additional vertical spike is added to mark the point estimates.

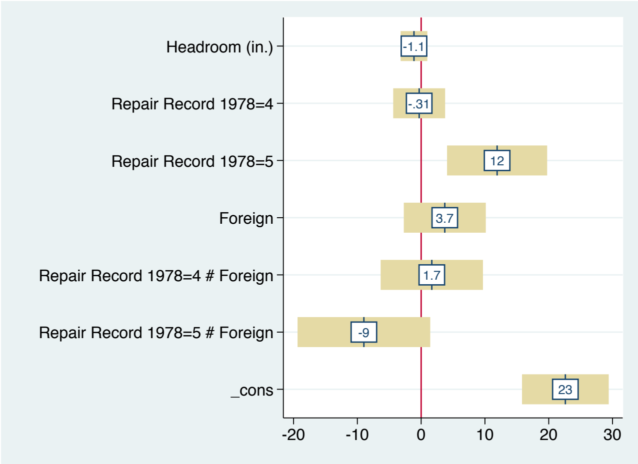
```
. sysuse auto, clear
(1978 Automobile Data)
. keep if rep78>=3
(10 observations deleted)
. regress mpg headroom i.rep##i.foreign
  (output omitted)
. mata: st_matrix("e(box)",
>     (st_matrix("e(b)") :- 2 \
>      st_matrix("e(b)") :+ 2))
. mata: st_matrix("e(spike)",
>     (st_matrix("e(b)") :- 1e-9 \
>      st_matrix("e(b)") :+ 1e-9 ))
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(0) msymbol(i)
>     ci(95 spike box)
>     ciopts(recast(. rbar rbar)
>         barwidth(. 0.6 0.35)
>         fcolor(. . white)
>         lwidth(. medium medium))
```



In the example, bars of close-to-zero width are used produce the vertical spikes. Zero width bars would be invisible. By adding tiny offsets of $\pm 10^{-9}$ the bars become visible.

The plot might still not be optimal since for the first coefficient, the confidence interval is hidden behind the marker label box. Plotting the confidence intervals as bars can, for example, solve this problem ("`..`" in `recast()` specifies to repeat style `rbar` until end; see [G] **stylelists**):

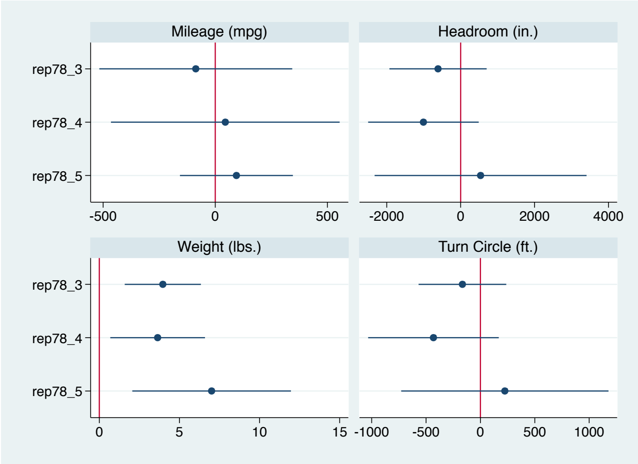
```
. coefplot, xline(0) mlabel format(%9.2g)
>     mlabposition(0) msymbol(i)
>     ci(95 spike box)
>     ciopts(recast(rbar ..)
>         barwidth(0.5 0.5 0.35)
>         fcolor(. . white) bstyle(ci2)
>         lwidth(. medium medium))
```



3.5.4 Arranging subgraphs by coefficients

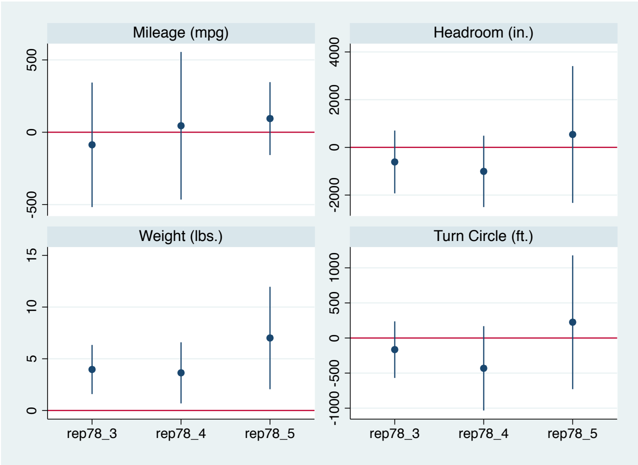
In some situations it is sensible to have a separate subgraph for each coefficient. This can be achieved by the `bycoefs` option. Technically, `bycoefs` flips coefficients and subgraphs, that is, the coefficients are treated as "subgraphs" and what was specified as subgraphs is treated as "coefficients". This seems difficult to understand, but should become clear in the following example:

```
. sysuse auto, clear
(1978 Automobile Data)
. forv i = 3/5 {
  2.     quietly regress price mpg
>            headroom weight turn
>            if rep78==`i´
  3.     estimate store rep78_`i´
  4. }
. coefplot rep78_3 || rep78_4 || rep78_5,
>     drop(_cons) xline(0)
>     bycoefs byopts(xrescale)
```



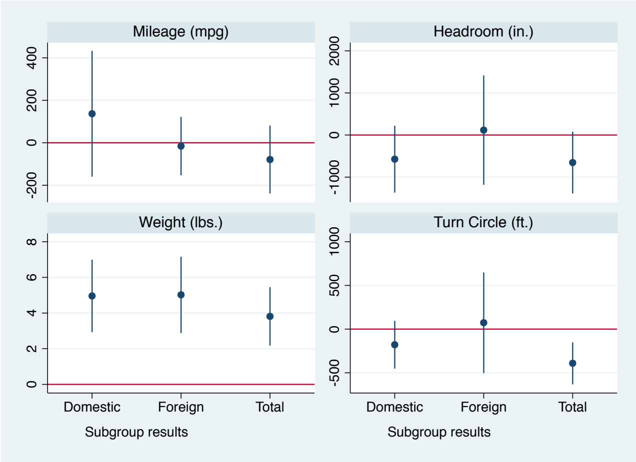
In the example, option `byopts(xrescale)` was specified so that each coefficient can have its own scale. As some people prefer vertical mode for such a graph, you might want to specify the `vertical` option:

```
. coefplot rep78_3 || rep78_4 || rep78_5,
>     drop(_cons) yline(0) vertical
>     bycoefs byopts(yrescale)
```



If `bycoefs` is specified, options `relocate()`, `headings()`, `groups()` apply to the elements on the categorical axis (instead of coefficients). To address the elements use integer numbers, `1`, `2`, `3` etc., as in the following example:

```
. sysuse auto, clear
(1978 Automobile Data)
. regress price mpg headroom weight turn
  (output omitted)
. estimates store Total
. regress price mpg headroom weight turn
>     if foreign==0
  (output omitted)
. estimates store Domestic
. regress price mpg headroom weight turn
>     if foreign==1
  (output omitted)
. estimates store Foreign
. coefplot Domestic || Foreign || Total,
>     drop(_cons) yline(0) vertical
>     bycoefs byopts(yrescale)
>     group(1 2 = "Subgroup results", nogap) ylabel(0, add)
```

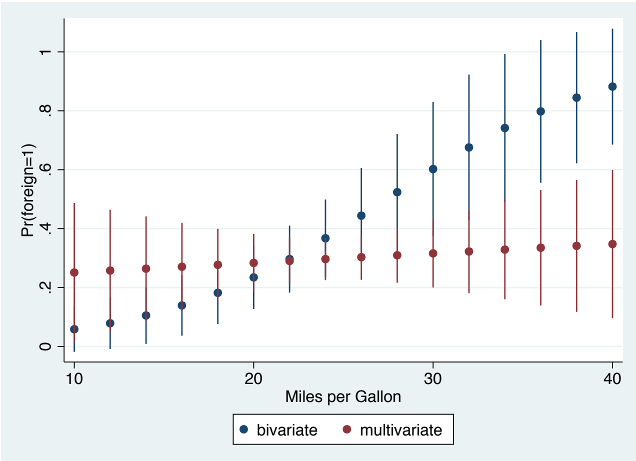


Option `ylabel(0, add)` has been added to ensure that zero is included in each subgraph.

3.5.5 Using a continuous axis

Coefficients provided to `coefplot` may represent estimates along a continuous dimension. Examples are predictive margins or marginal effects computed over values of a continuous variable. In such a case, use the `at()` option to provide the plot positions to `coefplot`. Here is an example where predictive margins of `foreign` are computed by level of `mpg`, once from a bivariate model and once from a multivariate model:

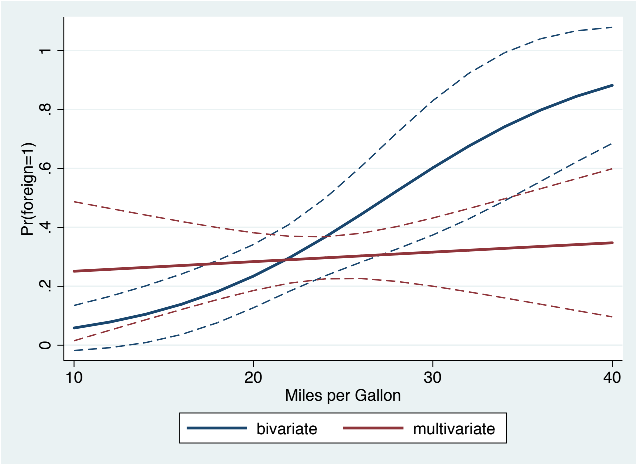
```
. sysuse auto, clear
(1978 Automobile Data)
. logit foreign mpg
  (output omitted)
. margins, at(mpg=(10(2)40)) post
  (output omitted)
. estimates store bivariate
. logit foreign mpg turn price
  (output omitted)
. margins, at(mpg=(10(2)40)) post
  (output omitted)
. estimates store multivariate
. coefplot bivariate multivariate, at
>     ytitle(Pr(foreign=1))
>     xtitle(Miles per Gallon)
```



`at()` causes `coefplot` to use a continuous axis with default labeling for the plotted estimates instead of compiling a categorical axis. It also causes `coefplot` to switch to vertical mode, as this is the more common way to display such results. As no categorical axis is constructed if `at()` is specified, options `order()`, `relocate()`, `grid()` `coeflabels()`, `eqlabels()`, `headings()`, `groups()`, and `bycoefs` are not allowed. Furthermore, note that continuous and categorical mode cannot be mixed. That is, `at()` has to be specified for all models or for none. In the example above, `at` was used without argument. This is suitable for results provided by `margins`, as `coefplot` contains some special code to retrieve the plot positions in this case. See the online help for alternative applications of `at()`.

`coefplot` does not change the plot type for markers and confidence intervals and hence still draws dots and spikes. Use the `recast()` option to change this, e.g., as follows:

```
. coefplot bivariate multivariate, at
>     ytitle(Pr(foreign=1))
>     xtitle(Miles per Gallon)
>     recast(line) lwidth(*2)
>     ciopts(recast(rline) lpattern(dash))
```



3.5.6 Plotting results from matrices

Finally, to plot results from a matrix ([P] **matrix**) instead of the `e()`-returns, use syntax

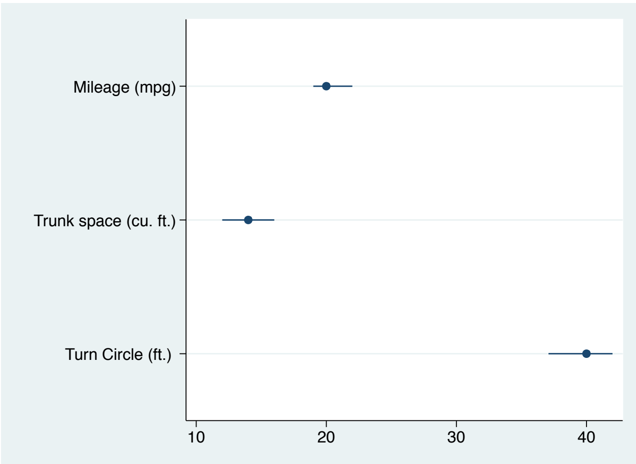
`coefplot` [(]`matrix(`*mspec*`)`[, *modelopts*] [...][)] [...]

where *mspec* is:

name	get point estimates from first row of matrix *name*
name`[`#`,.]`	get point estimates from row # of matrix *name*
name`[.,`#`]`	get point estimates from column # of matrix *name*

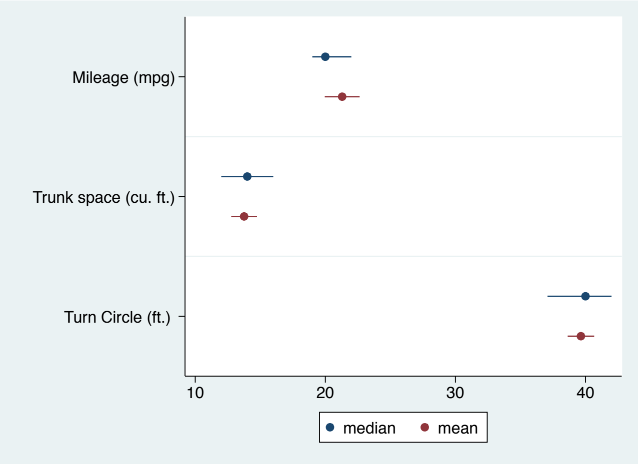
In this case, names given in `at()`, `v()`, `se()`, `df()`, and `ci()` will also be interpreted as matrix names. For example, to plot medians and their confidence intervals as computed by `centile` ([R] **centile**) you could type:

```
. sysuse auto, clear
(1978 Automobile Data)
. matrix res = J(3,3,.)
. matrix coln res = median ll95 ul95
. matrix rown res = mpg trunk turn
. local i 0
. foreach v of var mpg trunk turn {
  2.      local ++ i
  3.      quietly centile `v´
  4.      matrix res[`i´,1] = r(c_1),
>          r(lb_1), r(ub_1)
  5. }
. matrix list res
res[3,3]
          median       ll95       ul95
  mpg         20         19         22
trunk         14         12         16
 turn         40  37.078729         42
. coefplot matrix(res[.,1]), ci((res[.,2] res[.,3]))
```



A single `coefplot` command can contain both regular syntax and `matrix()` syntax. For example, to add means to the graph above you could proceed as follows:

```
. mean mpg trunk turn
  (output omitted)
. estimates store mean
. coefplot (matrix(res[,1]), label(median)
>         ci((res[,2] res[,3])))
>     (mean)
```



References

Gallup, J. L. 2012. A new system for formatting estimation tables. *The Stata Journal* 12(1): 3–28.

Jann, B. 2007. Making regression tables simplified. *The Stata Journal* 7(2): 227–244.

Kastellec, J. P., and E. L. Leoni. 2007. Using Graphs Instead of Tables in Political Science. *Perspectives on Politics* 5(4): 755–771.

Newson, R. 2003. Confidence intervals and p-values for delivery to the end user. *The Stata Journal* 3(3): 245–269.