Title

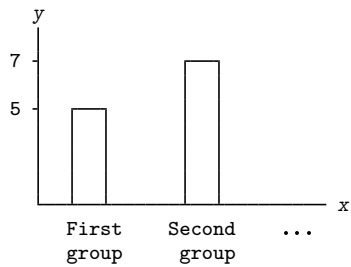
stata.com

graph bar — Bar charts

Description

`graph bar` draws vertical bar charts. In a vertical bar chart, the y axis is numerical, and the x axis is categorical.

```
. graph bar (mean) numeric_var, over(cat_var)
```

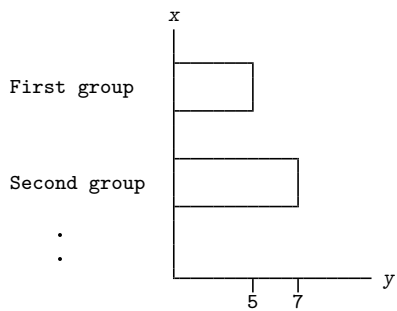


numeric_var must be numeric; statistics of it are shown on the *y* axis.

cat_var may be numeric or string; it is shown on the categorical *x* axis.

`graph hbar` draws horizontal bar charts. In a horizontal bar chart, the numerical axis is still called the y axis, and the categorical axis is still called the x axis, but y is presented horizontally, and x vertically.

```
. graph hbar (mean) numeric_var, over(cat_var)
```



same conceptual layout: *numeric_var* still appears on `y`, *cat_var* on `x`

The syntax for vertical and horizontal bar charts is the same; all that is required is changing `bar` to `hbar` or `hbar` to `bar`.

Quick start

Bar graph of percentages of observations for each level of categorical variable `catvar1`

```
graph bar, over(catvar1)
```

Bar graph of frequencies of observations for each level of `catvar1`

```
graph bar (count), over(catvar1)
```

Bar graph of the mean of `v1`

```
graph bar v1
```

Add the mean of `v2` to the graph

```
graph bar v1 v2
```

Same as above, but show the median of `v1` and `v2`

```
graph bar (median) v1 v2
```

Horizontal bar graph of the mean of `v1`

```
graph hbar v1
```

Same as above, but plot the mean of `v1` for each level of `catvar1`

```
graph hbar v1, over(catvar1)
```

Same as above, but with bars for each level of `catvar1` grouped by level of `catvar2`

```
graph hbar v1, over(catvar1) over(catvar2)
```

Same as above, but with each level of `catvar2` grouped by level of `catvar1`

```
graph hbar v1, over(catvar2) over(catvar1)
```

Bar graph of the mean of `v1` in separate graph areas for each level of `catvar2`

```
graph bar v1, by(catvar2)
```

Same as above, but with bars for each level of `catvar1` within each graph area

```
graph bar v1, over(catvar1) by(catvar2)
```

Bar graph of the sums of `v1` and `v2` with separate graph areas for levels of `catvar1`

```
graph bar (sum) v1 v2, by(catvar1)
```

Bar graph of the mean and median of `v1` for each level of `catvar1`

```
graph bar (mean) v1 (median) v1, over(catvar1)
```

Change the variable labels displayed in the legend

```
graph bar v1 v2, over(catvar1) legend(label(1 "Variable 1")  ///
     label(2 "Variable 2"))
```

Place a gap equal to 10% of the bar width between bars for `v1` and `v2`

```
graph bar v1 v2, bargap(10)
```

Overlap bars for `v1` and `v2` by 50% of the bar width

```
graph bar v1 v2, bargap(-50)
```

Display `v1` and `v2` as though they are categories of a single variable, as in an `over()` option

```
graph bar v1 v2, ascategory
```

Display the categories defined by catvar1 as though they are separate variables

```
graph bar v1, over(catvar1) asyvars
```

Sort bars in ascending order by the mean of v1

```
graph bar v1, over(catvar1, sort(1))
```

Same as above, but sort in descending order

```
graph bar v1, over(catvar1, sort(1) descending)
```

Stacked bar graph of the means of v1 and v2 with one bar for each level of catvar1

```
graph bar v1 v2, over(catvar1) stack
```

Stacked bar graph of the 25th and 50th percentiles of v1 for each level of catvar1

```
graph bar (p25) v1 (p50) v1, over(catvar1) stack
```

Plot summary statistics stored in v3 with labels defined by catvar3

```
graph bar (asis) v3, over(catvar3)
```

Menu

Graphics > Bar chart

Syntax

```
graph bar yvars [if] [in] [weight] [, options]

graph hbar yvars [if] [in] [weight] [, options]
```

where *yvars* is

```
(asis) varlist
```

or is

```
(percent) [varlist] | (count) [varlist]
```

or is

```
[(stat)] varname [[(stat)] ...]

[(stat)] varlist [[(stat)] ...]

[(stat)] [name=]varname[...] [[(stat)] ...]
```

where *stat* may be any of

```
mean median p1 p2 ... p99 sum count percent min max
```

or

any of the other *stats* defined in [D] **collapse**

yvars is optional if the option `over(`*varname*`)` is specified. `percent` is the default statistic, and percentages are calculated over *varname*.

`mean` is the default when *varname* or *varlist* is specified and *stat* is not specified. `p1` means the first percentile, `p2` means the second percentile, and so on; `p50` means the same as `median`. `count` means the number of nonmissing values of the specified variable.

options	Description
group_options	groups over which bars are drawn
yvar_options	variables that are the bars
lookofbar_options	how the bars look
legending_options	how *yvars* are labeled
axis_options	how the numerical y axis is labeled
title_and_other_options	titles, added text, aspect ratio, etc.

group_options	Description
over(*varname* [, *over_subopts*])	categories; option may be repeated
nofill	omit empty categories
missing	keep missing value as category
allcategories	include all categories in the dataset

yvar_options	Description
ascategory	treat *yvars* as first over() group
asyvars	treat first over() group as *yvars*
percentages	show percentages within *yvars*
stack	stack the *yvar* bars
cw	calculate *yvar* statistics omitting missing values of any *yvar*

lookofbar_options	Description
outergap([*]#)	gap between edge and first bar and between last bar and edge
bargap(#)	gap between *yvar* bars; default is 0
intensity([*]#)	intensity of fill
lintensity([*]#)	intensity of outline
pcycle(#)	bar styles before pstyles recycle
bar(#, *barlook_options*)	look of #th *yvar* bar

legending_options	Description
legend_options	control of *yvar* legend
nolabel	use *yvar* names, not labels, in legend
yvaroptions(*over_subopts*)	*over_subopts* for *yvars*; seldom specified
showyvars	label *yvars* on x axis; seldom specified
blabel(...)	add labels to bars

axis_options	Description
yalternate	put numerical y axis on right (top)
xalternate	put categorical x axis on top (right)
exclude0	do not force y axis to include 0
yreverse	reverse y axis
axis_scale_options	y-axis scaling and look
axis_label_options	y-axis labeling
ytitle(...)	y-axis titling

title_and_other_options	Description
`text(...)`	add text on graph; x range $[0, 100]$
`yline(...)`	add y lines to graph
aspect_option	constrain aspect ratio of plot region
std_options	titles, graph size, saving to disk
`by(`*varlist*`, ...)`	repeat for subgroups

The *over_subopts*—used in `over(`*varname*`,` *over_subopts*`)` and, on rare occasion, in `yvaroptions(`*over_subopts*`)`—are

over_subopts	Description
`relabel(# "`*text*`" ...)`	change axis labels
`label(`*cat_axis_label_options*`)`	rendition of labels
`axis(`*cat_axis_line_options*`)`	rendition of axis line
`gap(`[`*`]`#`)`	gap between bars within `over()` category
`sort(`*varname*`)`	put bars in prespecified order
`sort(#)`	put bars in height order
`sort((`*stat*`)` *varname*`)`	put bars in derived order
`descending`	reverse default or specified bar order
`reverse`	reverse scale to run from maximum to minimum

`aweights`, `fweights`, and `pweights` are allowed; see [U] **11.1.6 weight** and see note concerning weights in [D] **collapse**.

Options

Options are presented under the following headings:

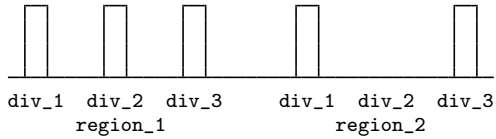
group_options

`over(`*varname*[`,` *over_subopts*]`)` specifies a categorical variable over which the *yvars* are to be repeated. *varname* may be string or numeric. Up to two `over()` options may be specified when multiple *yvars* are specified, and up to three `over()`s may be specified when one *yvar* is specified; options may be specified; see *Examples of syntax* and *Multiple over( )s (repeating the bars)* under *Remarks and examples* below.

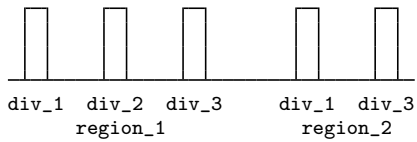
`nofill` specifies that missing subcategories be omitted. For instance, consider

```
. graph bar (mean) y, over(division) over(region)
```

Say that one of the divisions has no data for one of the regions, either because there are no such observations or because `y==.` for such observations. In the resulting chart, the bar will be missing:



If you specify `nofill`, the missing category will be removed from the chart:



`missing` specifies that missing values of the `over()` variables be kept as their own categories, one for `.`, another for `.a`, etc. The default is to act as if such observations simply did not appear in the dataset; the observations are ignored. An `over()` variable is considered to be missing if it is numeric and contains a missing value or if it is string and contains "".

`allcategories` specifies that all categories in the entire dataset be retained for the `over()` variables. When `if` or `in` is specified without `allcategories`, the graph is drawn, completely excluding any categories for the `over()` variables that do not occur in the specified subsample. With the `allcategories` option, categories that do not occur in the subsample still appear in the legend, and zero-height bars are drawn where these categories would appear. Such behavior can be convenient when comparing graphs of subsamples that do not include completely common categories for all `over()` variables. This option has an effect only when `if` or `in` is specified or if there are missing values in the variables. `allcategories` may not be combined with `by()`.

yvar_options

`ascategory` specifies that the *yvars* be treated as the first `over()` group; see *Treatment of bars* under *Remarks and examples* below. `ascategory` is a useful option.

When you specify `ascategory`, results are the same as if you specified one *yvar* and introduced a new first `over()` variable. Anyplace you read in the documentation that something is done over the first `over()` category, or using the first `over()` category, it will be done over or using *yvars*.

Suppose that you specified

```
. graph bar y1 y2 y3, ascategory whatever_other_options
```

The results will be the same as if you typed

```
. graph bar y, over(newcategoryvariable) whatever_other_options
```

with a long rather than wide dataset in memory.

`asyvars` specifies that the first `over()` group be treated as *yvars*. See *Treatment of bars* under *Remarks and examples* below.

When you specify `asyvars`, results are the same as if you removed the first `over()` group and introduced multiple *yvars*. If you previously had k *yvars* and, in your first `over()` category, G groups, results will be the same as if you specified $k \times G$ *yvars* and removed the `over()`. Anyplace you read in the documentation that something is done over the *yvars* or using the *yvars*, it will be done over or using the first `over()` group.

Suppose that you specified

```
. graph bar y, over(group) asyvars whatever_other_options
```

Results will be the same as if you typed

```
. graph bar y1 y2 y3 ..., whatever_other_options
```

with a wide rather than a long dataset in memory. Variables *y1*, *y2*, ..., are sometimes called the virtual *yvars*.

`percentages` specifies that bar heights be based on percentages that *yvar_i* represents of all the *yvars*. That is,

```
. graph bar (mean) inc_male inc_female
```

would produce a chart with bar height reflecting average income.

```
. graph bar (mean) inc_male inc_female, percentages
```

would produce a chart with the bar heights being $100 \times$ `inc_male`$/($`inc_male` $+$ `inc_female`$)$ and $100 \times$ `inc_female`$/($`inc_male` $+$ `inc_female`$)$.

If you have one *yvar* and want percentages calculated over the first `over()` group, specify the `asyvars` option. For instance,

```
. graph bar (mean) wage, over(i) over(j)
```

would produce a chart where bar heights reflect mean wages.

```
. graph bar (mean) wage, over(i) over(j) asyvars percentages
```

would produce a chart where bar heights are

$$100 \times \left(\frac{\text{mean}_{ij}}{\sum_i \text{mean}_{ij}} \right)$$

Option `stack` is often combined with option `percentages`.

`stack` specifies that the *yvar* bars be stacked.

```
. graph bar (mean) inc_male inc_female, over(region) percentages stack
```

would produce a chart with all bars being the same height, 100%. Each bar would be two bars stacked (percentage of `inc_male` and percentage of `inc_female`), so the division would show the relative shares of `inc_male` and `inc_female` of total income.

To stack bars over the first `over()` group, specify the `asyvars` option:

```
. graph bar (mean) wage, over(sex) over(region) asyvars percentages stack
```

`cw` specifies casewise deletion. If `cw` is specified, observations for which any of the *yvars* are missing are ignored. The default is to calculate the requested statistics by using all the data possible.

lookofbar_options

`outergap(*#)` and `outergap(#)` specify the gap between the edge of the graph to the beginning of the first bar and the end of the last bar to the edge of the graph.

`outergap(*#)` specifies that the default be modified. Specifying `outergap(*1.2)` increases the gap by 20%, and specifying `outergap(*.8)` reduces the gap by 20%.

`outergap(#)` specifies the gap as a percentage-of-bar-width units. `outergap(50)` specifies that the gap be half the bar width.

`bargap(#)` specifies the gap to be left between *yvar* bars as a percentage-of-bar-width units. The default is `bargap(0)`, meaning that bars touch.

`bargap()` may be specified as positive or negative numbers. `bargap(10)` puts a small gap between the bars (the precise amount being 10% of the width of the bars). `bargap(-30)` overlaps the bars by 30%.

`bargap()` affects only the *yvar* bars. If you want to change the gap for the first, second, or third `over()` groups, specify the *over_subopt* `gap()` inside the `over()` itself; see *Suboptions for use with over( ) and yvaroptions( )* below.

`intensity(#)` and `intensity(*#)` specify the intensity of the color used to fill the inside of the bar. `intensity(#)` specifies the intensity, and `intensity(*#)` specifies the intensity relative to the default.

By default, the bar is filled with the color of its border, attenuated. Specify `intensity(*#)`, `#`<1, to attenuate it more and specify `intensity(*#)`, `#`>1, to amplify it.

Specify `intensity(0)` if you do not want the bar filled at all. Specify `intensity(100)` if you want the bar to have the same intensity as the bar's outline.

`lintensity(#)` and `lintensity(*#)` specify the intensity of the line used to outline the bar. `lintensity(#)` specifies the intensity, and `lintensity(*#)` specifies the intensity relative to the default.

By default, the bar is outlined at the same intensity at which it is filled or at an amplification of that, which depending on your chosen scheme; see [G-4] **Schemes intro**. If you want the bar outlined in the darkest possible way, specify `intensity(255)`. If you wish simply to amplify the outline, specify `intensity(*#)`, `#` >1, and if you wish to attenuate the outline, specify `intensity(*#)`, `#` <1.

`pcycle(#)` specifies how many variables are to be plotted before the pstyle (see [G-4] ***pstyle***) of the bars for the next variable begins again at the pstyle of the first variable—`p1bar` (with the bars for the variable following that using `p2bar` and so). Put another way: `#` specifies how quickly the look of bars is recycled when more than `#` variables are specified. The default for most schemes is `pcycle(15)`.

`bar(#,` *barlook_options*`)` specifies the look of the *yvar* bars. `bar(1, ...)` refers to the bar associated with the first *yvar*, `bar(2, ...)` refers to the bar associated with the second, and so on. The most useful *barlook_option* is `color(`*colorstyle*`)`, which sets the color and opacity of the bar. For instance, you might specify `bar(1, color(green))` to make the bar associated with the first *yvar* green. See [G-4] ***colorstyle*** for a list of color choices, and see [G-3] ***barlook_options*** for information on the other *barlook_options*.

legending_options

legend_options controls the legend. If more than one *yvar* is specified, a legend is produced. Otherwise, no legend is needed because the `over()` groups are labeled on the categorical x axis. See [G-3] ***legend_options***, and see *Treatment of bars* under *Remarks and examples* below.

`nolabel` specifies that, in automatically constructing the legend, the variable names of the *yvars* be used in preference to "mean of *varname*" or "sum of *varname*", etc.

`yvaroptions(`*over_subopts*`)` allows you to specify *over_subopts* for the *yvars*. This is seldom done.

showyvars specifies that, in addition to building a legend, the identities of the *yvars* be shown on the categorical x axis. If showyvars is specified, it is typical also to specify legend(off).

blabel() allows you to add labels on top of the bars; see [G-3] ***blabel_option***.

axis_options

yalternate and xalternate switch the side on which the axes appear.

Used with graph bar, yalternate moves the numerical y axis from the left to the right; xalternate moves the categorical x axis from the bottom to the top.

Used with graph hbar, yalternate moves the numerical y axis from the bottom to the top; xalternate moves the categorical x axis from the left to the right.

If your scheme by default puts the axes on the opposite sides, then yalternate and xalternate reverse their actions.

exclude0 specifies that the numerical y axis need not be scaled to include 0.

yreverse specifies that the numerical y axis have its scale reversed so that it runs from maximum to minimum. This option causes bars to extend down rather than up (graph bar) or from right to left rather than from left to right (graph hbar).

axis_scale_options specify how the numerical y axis is scaled and how it looks; see [G-3] ***axis_scale_options***. There you will also see option xscale() in addition to yscale(). Ignore xscale(), which is irrelevant for bar charts.

axis_label_options specify how the numerical y axis is to be labeled. The *axis_label_options* also allow you to add and suppress grid lines; see [G-3] ***axis_label_options***. There you will see that, in addition to options ylabel(), ytick(), ..., ymtick(), options xlabel(), ..., xmtick() are allowed. Ignore the x*() options, which are irrelevant for bar charts.

ytitle() overrides the default title for the numerical y axis; see [G-3] ***axis_title_options***. There you will also find option xtitle() documented, which is irrelevant for bar charts.

title_and_other_options

text() adds text to a specified location on the graph; see [G-3] ***added_text_options***. The basic syntax of text() is

text(*#_y #_x* "*text*")

text() is documented in terms of twoway graphs. When used with bar charts, the "numeric" x axis is scaled to run from 0 to 100.

yline() adds horizontal (bar) or vertical (hbar) lines at specified y values; see [G-3] ***added_line_options***. The xline() option, also documented there, is irrelevant for bar charts. If your interest is in adding grid lines, see [G-3] ***axis_label_options***.

aspect_option allows you to control the relationship between the height and width of a graph's plot region; see [G-3] ***aspect_option***.

std_options allow you to add titles, control the graph size, save the graph on disk, and much more; see [G-3] ***std_options***.

by(*varlist*, ...) draws separate plots within one graph; see [G-3] ***by_option*** and see *Use with by()* under *Remarks and examples* below.

Suboptions for use with over( ) and yvaroptions( )

`relabel(`*# "text" ...*`)` specifies text to override the default category labeling. Pretend that variable `sex` took on two values and you typed

```
. graph bar ..., ... over(sex, relabel(1 "Male" 2 "Female"))
```

The result would be to relabel the first value of `sex` to be "Male" and the second value, "Female"; "Male" and "Female" would appear on the categorical x axis to label the bars. This would be the result, regardless of whether variable `sex` were string or numeric and regardless of the codes actually stored in the variable to record `sex`.

That is, *#* refers to category number, which is determined by sorting the unique values of the variable (here `sex`) and assigning 1 to the first value, 2 to the second, and so on. If you are unsure as to what that ordering would be, the easy way to find out is to type

```
. tabulate sex
```

If you also plan on specifying `graph bar`'s or `graph hbar`'s `missing` option,

```
. graph bar ..., ... missing over(sex, relabel(...))
```

then type

```
. tabulate sex, missing
```

to determine the coding. See [R] **tabulate oneway**.

Relabeling the values does not change the order in which the bars are displayed.

You may create multiple-line labels by using quoted strings within quoted strings:

`over(`*varname*`, relabel(1 ‘" "Male" "patients" "’ 2 ‘" "Female" "patients" "’))`

When specifying quoted strings within quoted strings, remember to use compound double quotes ‘" and "’ on the outer level.

`relabel()` may also be specified inside `yvaroptions()`. By default, the identity of the *yvars* is revealed in the legend, so specifying `yvaroptions(relabel())` changes the legend. Because it is the legend that is changed, using `legend(label())` is preferred; see *legending_options* above. In any case, specifying

`yvaroptions(relabel(1 "Males" 2 "Females"))`

changes the text that appears in the legend for the first *yvar* and the second *yvar*. *#* in `relabel(`*# ...*`)` refers to *yvar* number. Here you may not use the nested quotes to create multiline labels; use the `legend(label())` option because it provides multiline capabilities.

`label(`*cat_axis_label_options*`)` determines other aspects of the look of the category labels on the x axis. Except for `label(labcolor())` and `label(labsize())`, these options are seldom specified; see [G-3] ***cat_axis_label_options***.

`axis(`*cat_axis_line_options*`)` specifies how the axis line is rendered. This is a seldom specified option. See [G-3] ***cat_axis_line_options***.

`gap(#)` and `gap(*#)` specify the gap between the bars in this `over()` group. `gap(#)` is specified in percentage-of-bar-width units, so `gap(67)` means two-thirds the width of a bar. `gap(*#)` allows modifying the default gap. `gap(*1.2)` would increase the gap by 20%, and `gap(*.8)` would decrease the gap by 20%.

To understand the distinction between `over(..., gap())` and option `bargap()`, consider

```
. graph bar revenue profit, bargap(...) over(division, gap(...))
```

`bargap()` sets the distance between the revenue and profit bars. `over(,gap())` sets the distance between the bars for the first division and the second division, the second division and the third, and so on. Similarly, in

```
. graph bar revenue profit, bargap(...)
                            over(division, gap(...))
                            over(year,     gap(...))
```

`over(division, gap())` sets the gap between divisions and `over(year, gap())` sets the gap between years.

`sort(`*varname*`)`, `sort(#)`, and `sort((`*stat*`) `*varname*`)` control how bars are ordered. See *How bars are ordered* and *Reordering the bars* under *Remarks and examples* below.

`sort(`*varname*`)` puts the bars in the order of *varname*; see *Putting the bars in a prespecified order* under *Remarks and examples* below.

`sort(#)` puts the bars in height order. # refers to the *yvar* number on which the ordering should be performed; see *Putting the bars in height order* under *Remarks and examples* below.

`sort((`*stat*`) `*varname*`)` puts the bars in an order based on a calculated statistic; see *Putting the bars in a derived order* under *Remarks and examples* below.

`descending` specifies that the order of the bars—default or as specified by `sort()`—be reversed.

`reverse` specifies that the categorical scale run from maximum to minimum rather than the default minimum to maximum. Among other things, when combined with `bargap(-#)`, `reverse` causes the sequence of overlapping to be reversed.

Remarks and examples

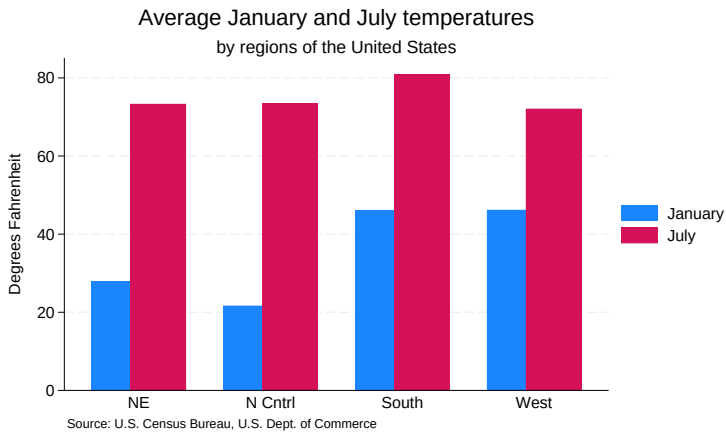
stata.com

Remarks are presented under the following headings:

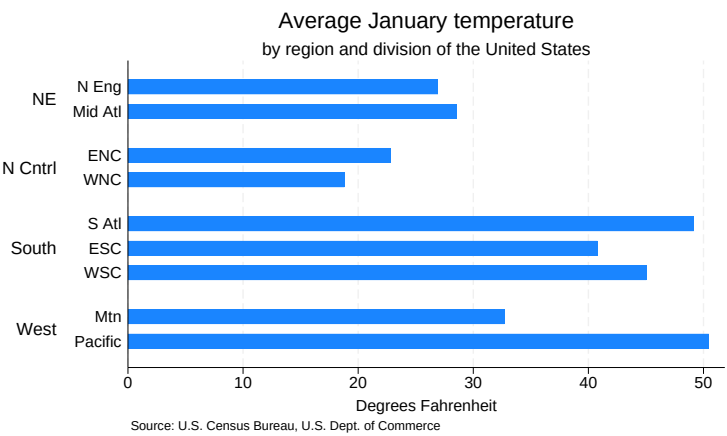
Introduction

Let us show you some bar charts:

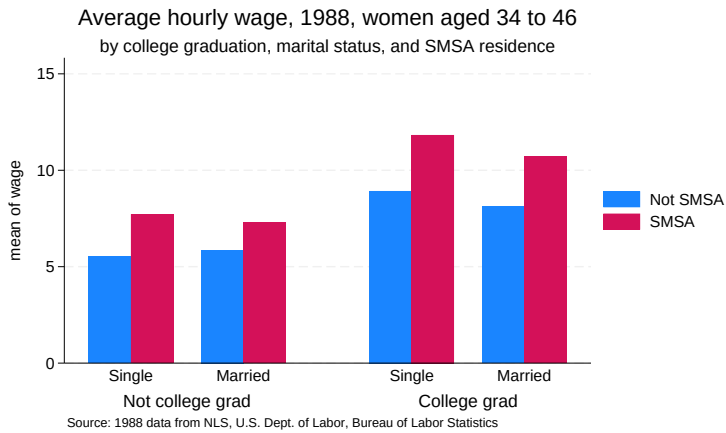
```
. use https://www.stata-press.com/data/r18/citytemp
(City temperature data)
. graph bar (mean) tempjan tempjuly, over(region)
        bargap(-30)
        legend( label(1 "January") label(2 "July") )
        ytitle("Degrees Fahrenheit")
        title("Average January and July temperatures")
        subtitle("by regions of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



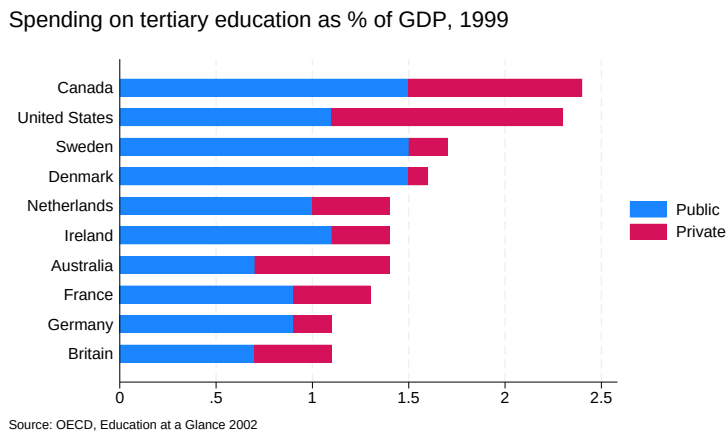

```
. use https://www.stata-press.com/data/r18/citytemp, clear
(City temperature data)
. graph hbar (mean) tempjan, over(division) over(region) nofill
        ytitle("Degrees Fahrenheit")
        title("Average January temperature")
        subtitle("by region and division of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



```
. use https://www.stata-press.com/data/r18/nlsw88, clear
(NLSW, 1988 extract)
. graph bar (mean) wage, over(smsa) over(married) over(collgrad)
        title("Average hourly wage, 1988, women aged 34 to 46")
        subtitle("by college graduation, marital status,
                  and SMSA residence")
        note("Source: 1988 data from NLS, U.S. Dept. of Labor,
              Bureau of Labor Statistics")
```




```
. use https://www.stata-press.com/data/r18/educ99gdp, clear
(Education and GDP)
. generate total = private + public
. graph hbar (asis) public private,
        over(country, sort(total) descending) stack
        title("Spending on tertiary education as % of GDP,
                1999", span position(11))
        subtitle(" ")
        note("Source: OECD, Education at a Glance 2002", span)
```



In the sections that follow, we explain how each of the above graphs—and others—are produced.

Examples of syntax

Below we show you some `graph bar` commands and tell you what each would do:

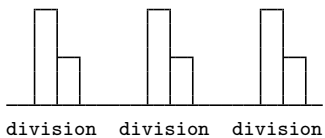
`graph bar, over(division)`
#_of_divisions bars showing the percentage of observations for each division.

`graph bar (count), over(division)`
#_of_divisions bars showing the frequency of observations for each division. `graph bar revenue` One big bar showing average revenue.

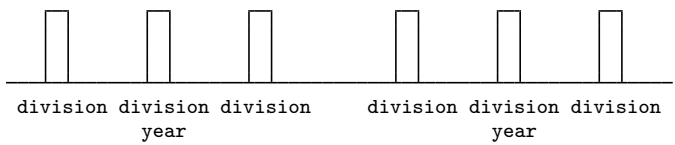
`graph bar revenue profit`
Two bars, one showing average revenue and the other showing average profit.

`graph bar revenue, over(division)`
#_of_divisions bars showing average revenue for each division.

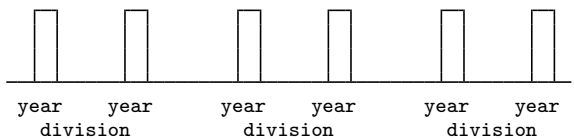
`graph bar revenue profit, over(division)`
$2 \times$ *#_of_divisions* bars showing average revenue and average profit for each division. The grouping would look like this (assuming three divisions):



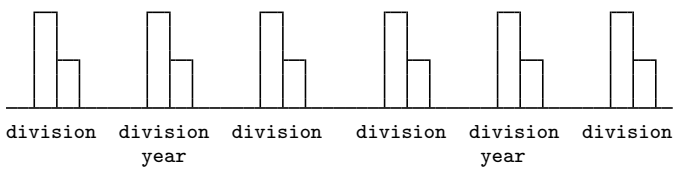
`graph bar revenue, over(division) over(year)`
#_of_divisions $\times$ *#_of_years* bars showing average revenue for each division, repeated for each of the years. The grouping would look like this (assuming three divisions and 2 years):



`graph bar revenue, over(year) over(division)`
same as above but ordered differently. In the previous example, we typed `over(division) over(year)`. This time, we reverse it:



`graph bar revenue profit, over(division) over(year)`
$2 \times$ *#_of_divisions* $\times$ *#_of_years* bars showing average revenue and average profit for each division, repeated for each of the years. The grouping would look like this (assuming three divisions and 2 years):



`graph bar (sum) revenue profit, over(division) over(year)`
$2 \times$ *#_of_divisions* $\times$ *#_of_years* bars showing the sum of revenue and sum of profit for each division, repeated for each of the years.

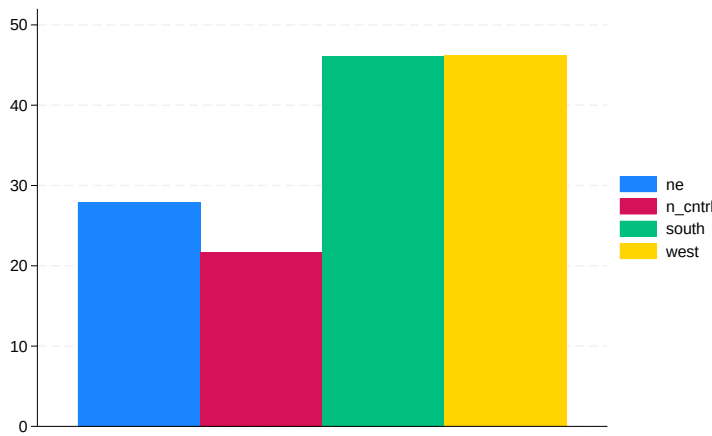
`graph bar (median) revenue profit, over(division) over(year)`
$2 \times$ *#_of_divisions* $\times$ *#_of_years* bars showing the median of revenue and median of profit for each division, repeated for each of the years.

`graph bar (median) revenue (mean) profit, over(division) over(year)`
$2 \times$ *#_of_divisions* $\times$ *#_of_years* bars showing the median of revenue and mean of profit for each division, repeated for each of the years.

Treatment of bars

Assume that someone tells you that the average January temperature in the Northeast of the United States is 27.9 degrees Fahrenheit, 21.7 degrees in the North Central, 46.1 in the South, and 46.2 in the West. You could enter these statistics and draw a bar chart:

```
. input ne n_cntrl south west
            ne         n_cntrl       south       west
  1. 27.9 21.7 46.1 46.2
  2. end
. graph bar (asis) ne n_cntrl south west
```

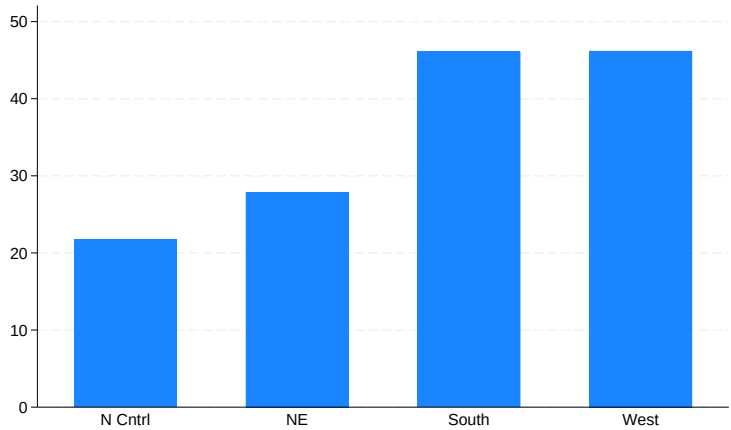


The above is admittedly not a great-looking chart, but specifying a few options could fix that. The important thing to see right now is that, when we specify multiple *yvars*, 1) the bars touch, 2) the bars are different colors (or at least different shades of gray), and 3) the meaning of the bars is revealed in the legend.

We could enter these data another way:

```
. clear
. input  str10 region  float tempjan
      region  tempjan
  1. NE 27.9
  2. "N Cntrl" 21.7
  3. South 46.1
  4. West 46.2
  5. end
```

```
. graph bar (asis) tempjan, over(region)
```



Observe that, when we generate multiple bars via an `over()` option, 1) the bars do not touch, 2) the bars are all the same color, and 3) the meaning of the bars is revealed by how the categorical x axis is labeled.

These differences in the treatment of the bars in the multiple *yvars* case and the `over()` case are general properties of `graph bar` and `graph hbar`:

	multiple *yvars*	`over()` groups
bars touch	yes	no
bars different colors	yes	no
bars identified via ...	legend	axis label

Option `ascategory` causes multiple *yvars* to be presented as if they were `over()` groups, and option `asyvars` causes `over()` groups to be presented as if they were *yvars*. Thus

```
. graph bar (asis) tempjan, over(region)
```

would produce the first chart and

```
. graph bar (asis) ne n_cntrl south west, ascategory
```

would produce the second.

Treatment of data

In the previous two examples, we already had the statistics we wanted to plot: 27.9 (Northeast), 21.7 (North Central), 46.1 (South), and 46.2 (West). We entered the data, and we typed

```
. graph bar (asis) ne n_cntrl south west
```

or

```
. graph bar (asis) tempjan, over(region)
```

We do not have to know the statistics ahead of time: `graph bar` and `graph hbar` can calculate statistics for us. If we had datasets with lots of observations (say, cities of the United States), we could type

```
. graph bar (mean) ne n_cntrl south west
```

or

```
. graph bar (mean) tempjan, over(region)
```

and obtain the same graphs. All we need to do is change `(asis)` to `(mean)`. In the first example, the data would be organized the wide way:

cityname	ne	n_cntrl	south	west
name of city	42	.	.	.
another city	.	28	.	.
...				

and in the second example, the data would be organized the long way:

cityname	region	tempjan
name of city	ne	42
another city	n_cntrl	28
...		

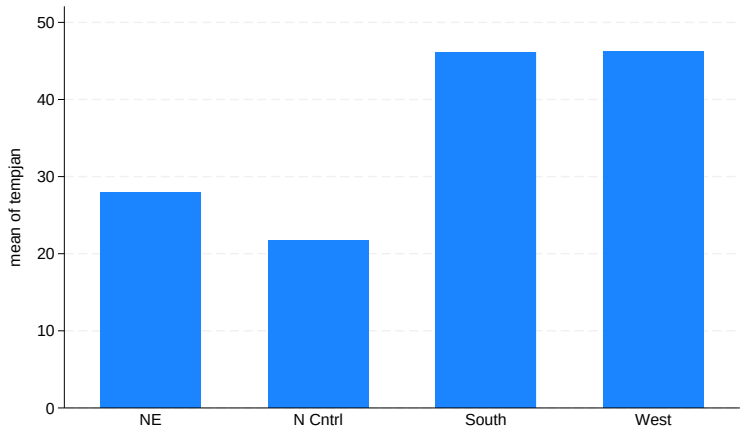
We have such a dataset, organized the long way. In `citytemp.dta`, we have information on 956 U.S. cities, including the region in which each is located and its average January temperature:

```
. use https://www.stata-press.com/data/r18/citytemp, clear
(City temperature data)
. list region tempjan if _n < 3 | _n > 954
```

	region	tempjan
1.	NE	16.6
2.	NE	18.2
955.	West	72.6
956.	West	72.6

With these data, we can type

```
. graph bar (mean) tempjan, over(region)
```



We just produced the same bar chart we previously produced when we entered the statistics 27.9 (Northeast), 21.7 (North Central), 46.1 (South), and 46.2 (West) and typed

```
. graph bar (asis) tempjan, over(region)
```

When we do not specify `(asis)` or `(mean)` (or `(median)` or `(sum)` or `(p1)` or any of the other *stats* allowed), `(mean)` is assumed. Thus `(...)` is often omitted when `(mean)` is desired, and we could have drawn the previous graph by typing

```
. graph bar tempjan, over(region)
```

Some users even omit typing `(...)` in the `(asis)` case because calculating the mean of one observation results in the number itself. Thus in the previous section, rather than typing

```
. graph bar (asis) ne n_cntrl south west
```

and

```
. graph bar (asis) tempjan, over(region)
```

We could have typed

```
. graph bar ne n_cntrl south west
```

and

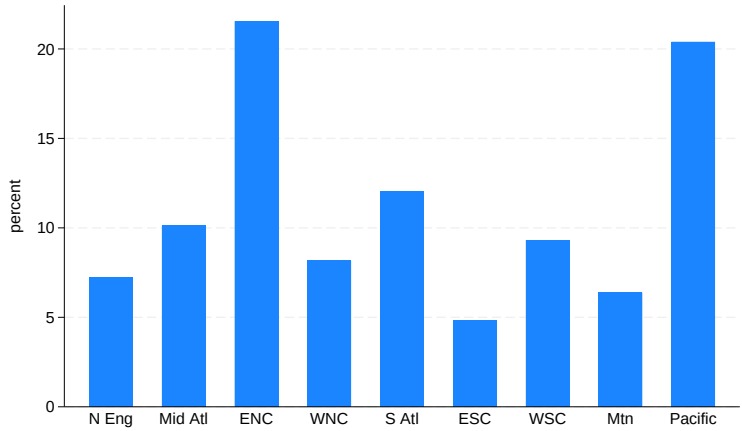
```
. graph bar tempjan, over(region)
```

Obtaining frequencies

The (percent) and (count) statistics work just like any other statistic with the graph bar command. In addition to the standard syntax, you may use the abbreviated syntax below to create bar graphs for percentages and frequencies over categorical variables.

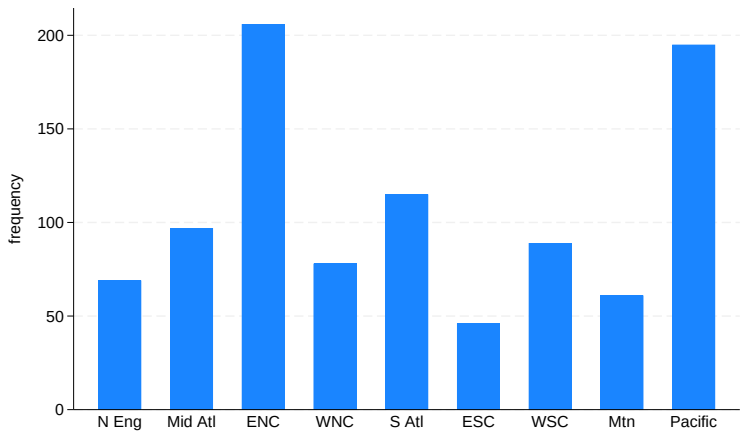
To graph the percentage of observations in each category of division, type

```
. use https://www.stata-press.com/data/r18/citytemp, clear
(City temperature data)
. graph bar, over(division)
```



To graph the frequency of observations in each category of division, type

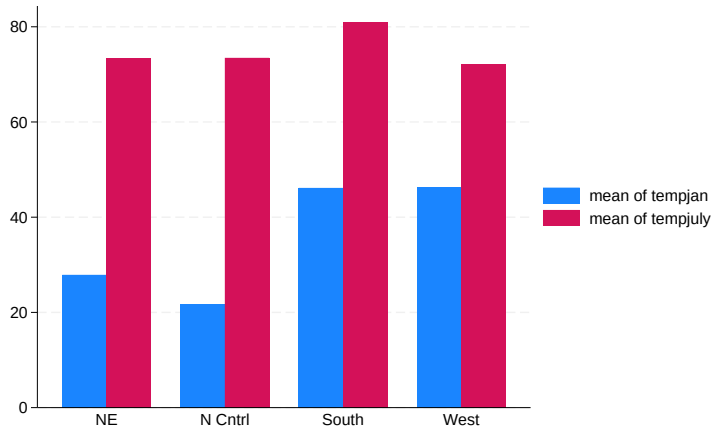
```
. graph bar (count), over(division)
```



Multiple bars (overlapping the bars)

In citytemp.dta, in addition to variable tempjan, there is variable tempjuly, which is the average July temperature. We can include both averages in one chart, by region:

```
. use https://www.stata-press.com/data/r18/citytemp, clear
(City temperature data)
. graph bar (mean) tempjan tempjuly, over(region)
```

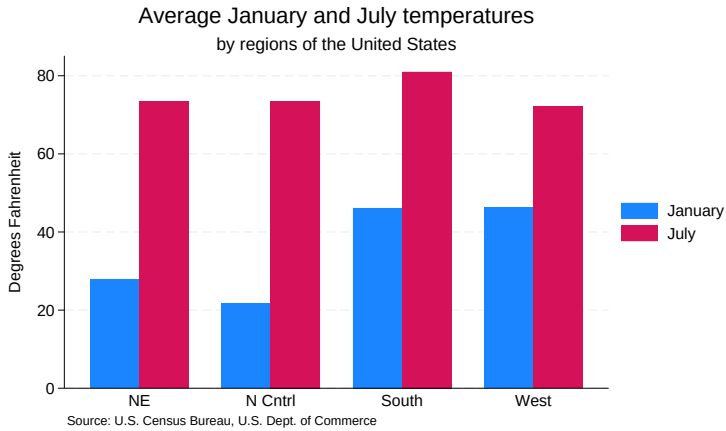


We can improve the look of the chart by

1. including the *legend_options* legend(label()) to change the text of the legend; see [G-3] ***legend_options***;
2. including the *axis_title_option* ytitle() to add a title saying "Degrees Fahrenheit"; see [G-3] ***axis_title_options***;
3. including the *title_options* title(), subtitle(), and note() to say what the graph is about and from where the data came; see [G-3] ***title_options***.

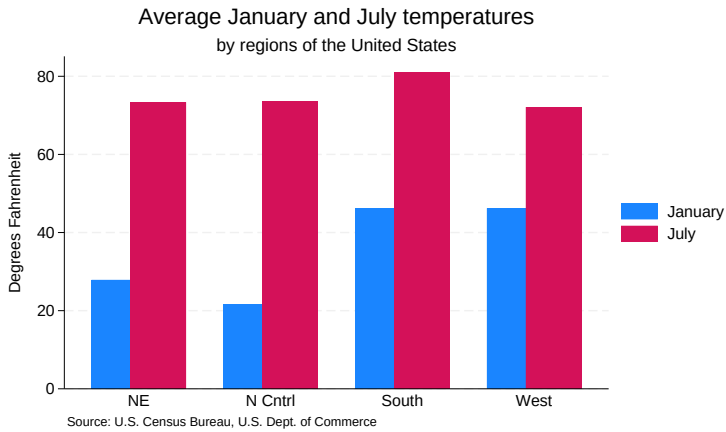
Doing all that produces

```
. graph bar (mean) tempjan tempjuly, over(region)
        legend( label(1 "January") label(2 "July") )
        ytitle("Degrees Fahrenheit")
        title("Average January and July temperatures")
        subtitle("by regions of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



We can make one more improvement to this chart by overlapping the bars. Below we add the option `bargap(-30)`:

```
. graph bar (mean) tempjan tempjuly, over(region)
        bargap(-30)                                          ← new
        legend( label(1 "January") label(2 "July") )
        ytitle("Degrees Fahrenheit")
        title("Average January and July temperatures")
        subtitle("by regions of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



`bargap(#)` specifies the distance between the *yvar* bars (that is, between the bars for `tempjan` and `tempjuly`); `#` is in percentage-of-bar-width units, so `barwidth(-30)` means that the bars overlap by 30%. `bargap()` may be positive or negative; its default is 0.

Controlling the text of the legend

In the above example, we changed the text of the legend by specifying the legend option:

```
legend( label(1 "January") label(2 "July") )
```

We could just as well have changed the text of the legend by typing

```
yvaroptions( relabel(1 "January" 2 "July") )
```

Which you use makes no difference, but we prefer `legend(label())` to `yvaroptions(relabel())` because `legend(label())` is the way to modify the contents of a legend in a twoway graph; so why do bar charts differently?

Multiple over()s (repeating the bars)

Option `over(`*varname*`)` repeats the *yvar* bars for each unique value of *varname*. Using `citytemp.dta`, if we typed

```
. graph bar (mean) tempjan tempjuly
```

we would obtain two (fat) bars. When we type

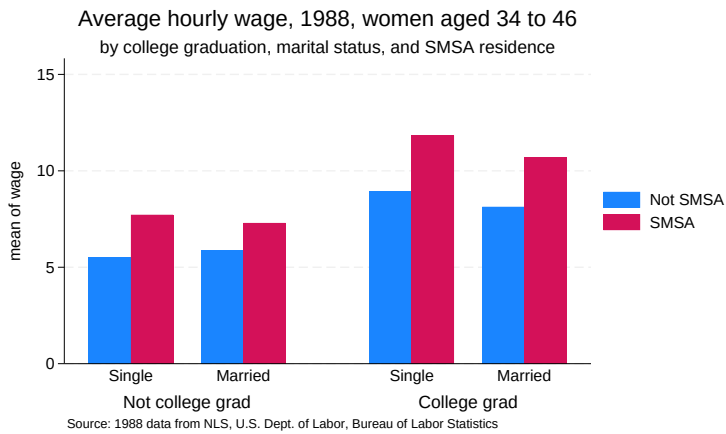
```
. graph bar (mean) tempjan tempjuly, over(region)
```

we obtain two (thinner) bars for each of the four regions. (We typed exactly this command in *Multiple bars* above.)

You may repeat the `over()` option. You may specify `over()` twice when you specify two or more *yvars* and up to three times when you specify just one *yvar*.

In `nlsw88.dta`, we have information on 2,246 women:

```
. use https://www.stata-press.com/data/r18/nlsw88, clear
(NLSW, 1988 extract)
. graph bar (mean) wage, over(smsa) over(married) over(collgrad)
        title("Average hourly wage, 1988, women aged 34 to 46")
        subtitle("by college graduation, marital status,
                  and SMSA residence")
        note("Source: 1988 data from NLS, U.S. Dept. of Labor,
              Bureau of Labor Statistics")
```



If you strip away the *title_options*, the above command reads

```
. graph bar (mean) wage, over(smsa) over(married) over(collgrad)
```

In this three-`over()` case, the first `over()` is treated as multiple *yvars*: the bars touch, the bars are assigned different colors, and the meaning of the bars is revealed in the legend. When you specify three `over()` groups, the first is treated the same way as multiple *yvars*. This means that if we wanted to separate the bars, we could specify option `bargap(#)`, #>0, and if we wanted them to overlap, we could specify `bargap(#)`, #<0.

Nested over( )s

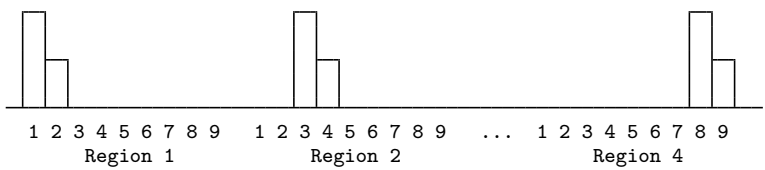
Sometimes you have multiple `over()` groups with one group explicitly nested within the other. In `citytemp.dta`, we have variables `region` and `division`, and `division` is nested within `region`. The Census Bureau divides the United States into four regions and into nine divisions, which work like this

Region	*Division*
1. Northeast	1. New England 2. Mid Atlantic
2. North Central	3. East North Central 4. West North Central
3. South	5. South Atlantic 6. East South Central 7. West South Central
4. West	8. Mountain 9. Pacific

Were we to type

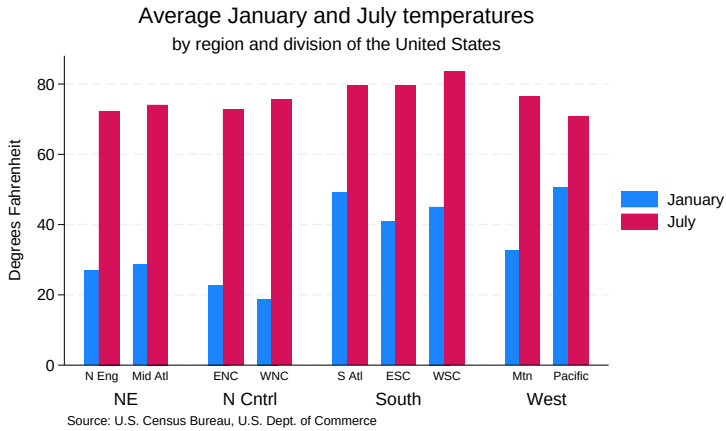
```
. graph bar (mean) tempjan tempjuly, over(division) over(region)
```

we would obtain a chart with space allocated for 9*4 = 36 groups, of which only nine would be used:



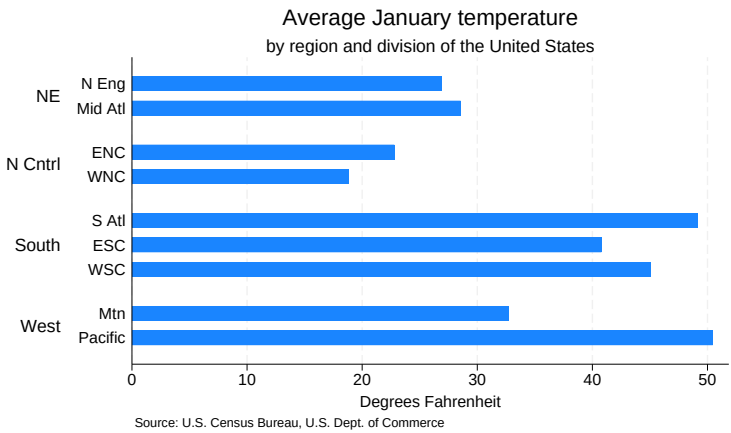
The `nofill` option prevents the chart from including the unused categories:

```
. use https://www.stata-press.com/data/r18/citytemp, clear
(City temperature data)
. graph bar tempjan tempjuly, over(division) over(region) nofill
        bargap(-30)
        ytitle("Degrees Fahrenheit")
        legend( label(1 "January") label(2 "July") )
        title("Average January and July temperatures")
        subtitle("by region and division of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



The above chart, if we omit one of the temperatures, also looks good horizontally:

```
. graph hbar (mean) tempjan, over(division) over(region) nofill
        ytitle("Degrees Fahrenheit")
        title("Average January temperature")
        subtitle("by region and division of the United States")
        note("Source: U.S. Census Bureau, U.S. Dept. of Commerce")
```



Charts with many categories

Using `nlsw88.dta`, we want to draw the chart

```
. use https://www.stata-press.com/data/r18/nlsw88
(NLSW, 1988 extract)
. graph bar wage, over(industry) over(collgrad)
```

Variable `industry` records industry of employment in 12 categories, and variable `collgrad` records whether the woman is a college graduate. Thus we will have 24 bars. We draw the above and quickly discover that the long labels associated with industry result in much overprinting along the horizontal x axis.

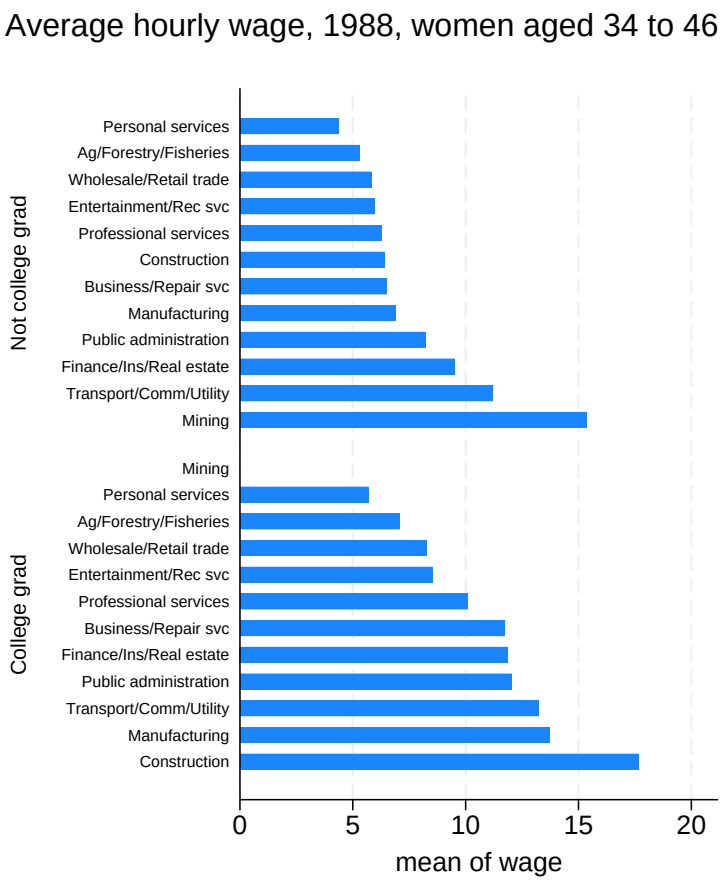
Horizontal bar charts work better than vertical bar charts when labels are long. We change our command to read

```
. graph hbar wage, over(ind) over(collgrad)
```

That works better, but now we have overprinting problems of a different sort: the letters of one line are touching the letters of the next.

Graphs drawn with the default manual scheme `stgcolor` are 3.987 inches wide by 2.392 inches tall. Here we need to make the chart taller, and that is the job of the *region_option* `ysize()`. Below we make a chart that is 5 inches tall:

```
. use https://www.stata-press.com/data/r18/nlsw88, clear
(NLSW, 1988 extract)
. graph hbar wage, over(ind, label(labsize(vsmall)) sort(1))
        over(collgrad, label(angle(90) labsize(small)))
        title("Average hourly wage, 1988, women aged 34 to 46",
              size(medlarge) span)
        subtitle(" ")
        note("Source: 1988 data from NLS, U.S. Dept. of Labor,
              Bureau of Labor Statistics", span)
        ysize(5)
```



The important option in the above is `ysize(5)`, which made the graph taller than usual; see [G-3] ***region_options***. Concerning the other options:

`over(ind, label(labsize(vsmall)) sort(1))`
`labsize(vsmall)` reduces the label text size for `ind`. `sort(1)` is specified so that the bars would be sorted on mean wage. The `1` says to sort on the first *yvar*; see *Reordering the bars* below.

`over(collgrad, label(angle(90) labsize(small)))`
`labsize(small)` reduces the label text size for `collgrad`. `angle(90)` displays the label text vertically (90 degrees).

`title("Average hourly wage, 1988, women aged 34 to 46", size(medlarge) span)`
`size(medlarge)` reduces the title text size. `span` is specified so that the title, rather than being centered over the plot region, would be centered over the entire graph. Here the plot region (the part of the graph where the real chart appears, ignoring the labels) is narrow, and centering over that was not going to work. See [G-3] ***region_options*** for a description of the graph region and plot region, and see [G-3] ***title_options*** and [G-3] ***textbox_options*** for a description of `span`.

`subtitle(" ")`
We specified this because the title looked too close to the graph without it. We could have done things properly and specified a `margin()` suboption within the `title()`, but we often find it easier to include a blank subtitle. We typed `subtitle(" ")` and not `subtitle("")`. We had to include the blank, or the subtitle would not have appeared.

`note("Source: 1988 data from NLS, ...", span)`
`span` is specified so that the note would be left-justified in the graph rather than just in the plot region.

How bars are ordered

The default is to place the bars in the order of the *yvars* and to order each set of `over(`*varname*`)` groups according to the values of *varname*. Let us consider some examples:

`graph bar (sum) revenue profit`
Bars appear in the order specified: `revenue` and `profit`.

`graph bar (sum) revenue, over(division)`
Bars are ordered according to the values of variable `division`.

If `division` is a numeric variable, the lowest division number comes first, followed by the next lowest, and so on. This is true even if variable `division` has a value label. Say that division 1 has been labeled "Sales" and division 2 is labeled "Development". The bars will be in the order Sales followed by Development.

If `division` is a string variable, the bars will be ordered by the sort order of the values of `division` (meaning alphabetically, but with capital letters placed before lowercase letters). If variable `division` contains the values "Sales" and "Development", the bars will be in the order Development followed by Sales.

`graph bar (sum) revenue profit, over(division)`
Bars appear in the order specified, `revenue` and `profit`, and are repeated for each division, which will be ordered as explained above.

`graph bar (sum) revenue, over(division) over(year)`
Bars appear ordered by the values of division, as previously explained, and then that is repeated for each of the years. The years are ordered according to the values of the variable `year`, following the same rules as applied to the variable `division`.

`graph bar (sum) revenue profit, over(division) over(year)`
Bars appear in the order specified, profit and revenue, repeated for division ordered on the values of variable `division`, repeated for year ordered on the values of variable `year`.

Reordering the bars

There are three ways to reorder the bars:

1. You want to control the order in which the elements of each `over()` group appear. Your divisions might be named Development, Marketing, Research, and Sales, alphabetically speaking, but you want them to appear in the more logical order Research, Development, Marketing, and Sales.
2. You wish to order the bars according to their heights. You wish to draw the graph

```
. graph bar (sum) empcost, over(division)
```

   and you want the divisions ordered by total employee cost.
3. You wish to order on some other derived value.

We will consider each of these desires separately.

Putting the bars in a prespecified order

We have drawn the graph

```
. graph (sum) bar empcost, over(division)
```

Variable `division` is a string containing "Development", "Marketing", "Research", and "Sales". We want to draw the chart, placing the divisions in the order Research, Development, Marketing, and Sales.

To do that, we create a new numeric variable that orders `division` as we would like:

```
. generate order = 1 if division=="Research"
. replace  order = 2 if division=="Development"
. replace  order = 3 if division=="Marketing"
. replace  order = 4 if division=="Sales"
```

We can name the variable and create it however we wish, but we must be sure that there is a one-to-one correspondence between the new variable and the `over()` group's values. We then specify the `over()`'s `sort(`*varname*`)` option:

```
. graph bar (sum) empcost, over( division, sort(order) )
```

If you want to reverse the order, you may specify the `descending` suboption:

```
. graph bar (sum) empcost, over(division, sort(order) descending)
```

Putting the bars in height order

We have drawn the graph

```
. graph bar (sum) empcost, over(division)
```

and now wish to put the bars in height order, shortest first. We type

```
. graph bar (sum) empcost, over( division, sort(1) )
```

If we wanted the tallest first, type

```
. graph bar empcost, over(division, sort(1) descending)
```

The `1` in `sort(1)` refers to the first (and here only) *yvar*. If we had multiple *yvars*, we might type

```
. graph bar (sum) empcost othcost, over( division, sort(1) )
```

and we would have a chart showing employee cost and other cost, sorted on employee cost. If we typed

```
. graph bar (sum) empcost othcost, over( division, sort(2) )
```

the graph would be sorted on other cost.

We can use `sort(#)` on the second `over()` group as well:

```
. graph bar (sum) empcost, over( division, sort(1) )
                           over( country,  sort(1) )
```

Country will be ordered on the sum of the heights of the bars.

Putting the bars in a derived order

We have employee cost broken into two categories: `empcost_direct` and `empcost_indirect`. Variable `emp_cost` is the sum of the two. We wish to make a chart showing the two costs, stacked, over `division`, and we want the bars ordered on the total height of the stacked bars. We type

```
. graph bar (sum) empcost_direct empcost_indirect,
                  stack
                  over(division, sort((sum) empcost) descending)
```

Reordering the bars, example

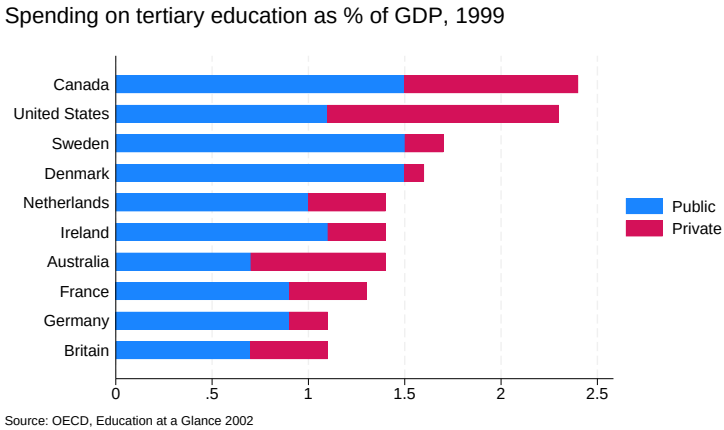
We have a dataset showing the spending on tertiary education as a percentage of GDP from the 2002 edition of *Education at a Glance: OECD Indicators 2002*:

```
. use https://www.stata-press.com/data/r18/educ99gdp, clear
(Education and GDP)
. list
```

	country	public	private
1.	Australia	.7	.7
2.	Britain	.7	.4
3.	Canada	1.5	.9
4.	Denmark	1.5	.1
5.	France	.9	.4
6.	Germany	.9	.2
7.	Ireland	1.1	.3
8.	Netherlands	1	.4
9.	Sweden	1.5	.2
10.	United States	1.1	1.2

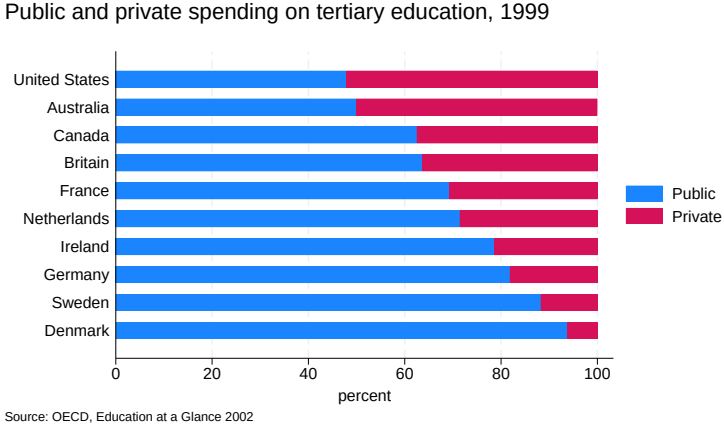
We wish to graph total spending on education and simultaneously show the distribution of that total between public and private expenditures. We want the bar sorted on total expenditures:

```
. generate total = private + public
. graph hbar (asis) public private,
        over(country, sort(total) descending) stack
        title("Spending on tertiary education as % of GDP, 1999",
                span position(11))
        subtitle(" ")
        note("Source: OECD, Education at a Glance 2002", span)
```



Or perhaps we wish to disguise the total expenditures and focus the graph exclusively on the share of spending that is public and private:

```
. generate frac = private/(private + public)
. graph hbar (asis) public private,
        over(country, sort(frac) descending) stack percentages
        title("Public and private spending on tertiary education, 1999",
               span position(11) )
        subtitle(" ")
        note("Source: OECD, Education at a Glance 2002", span)
```



The only differences between the two `graph hbar` commands are as follows:

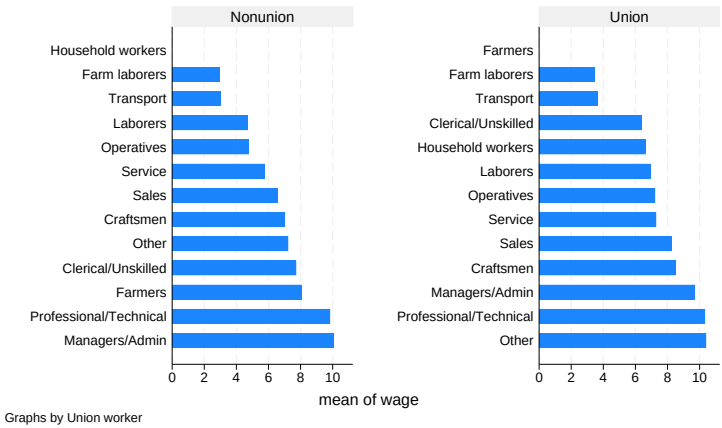
1. The `percentages` option was added to change the *yvars* `public` and `private` from spending amounts to percentages each is of the total.
2. The order of the bars was changed.
3. The title was changed.

Use with by()

`graph bar` and `graph hbar` may be used with `by()`, but in general, you want to use `over()` in preference to `by()`. Bar charts are explicitly categorical and do an excellent job of presenting summary statistics for multiple groups in one chart.

A good use of `by()`, however, is when you are ordering the bars and you wish to emphasize that the ordering is different for different groups. For instance,

```
. use https://www.stata-press.com/data/r18/nlsw88, clear
(NLSW, 1988 extract)
. graph hbar wage, over(occ, sort(1)) by(union)
```



The above graph orders the bars by height (hourly wage); the orderings are different for union and nonunion workers.

Video example

Bar graphs in Stata

History

The first published bar chart appeared in William Playfair's *Commercial and Political Atlas* (1786). See Tufte (2001, 32–33) or Beniger and Robyn (1978) for more historical information.

References

Beniger, J. R., and D. L. Robyn. 1978. Quantitative graphics in statistics: A brief history. *American Statistician* 32: 1–11. https://doi.org/10.2307/2683467.

Cox, N. J. 2004. Speaking Stata: Graphing categorical and compositional data. *Stata Journal* 4: 190–215.

——. 2005. Stata tip 24: Axis labels on two or more levels. *Stata Journal* 5: 469.

——. 2008. Speaking Stata: Spineplots and their kin. *Stata Journal* 8: 105–121.

——. 2011. Stata tip 102: Highlighting specific bars. *Stata Journal* 11: 474–477.

——. 2021. Stata tip 140: Shorter or fewer category labels withgraph bar. *Stata Journal* 21: 263–271.

Playfair, W. H. 1786. *Commercial and Political Atlas: Representing, by means of stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure, and Debts of England, during the Whole of the Eighteenth Century*. London: Corry.

Tufte, E. R. 2001. *The Visual Display of Quantitative Information*. 2nd ed. Cheshire, CT: Graphics Press.

Also see

[G-2] **graph dot** — Dot charts (summary statistics)

[D] **collapse** — Make dataset of summary statistics

[R] **table** — Table of frequencies, summaries, and command results