The Stata Journal (2018)
18, Number 3, pp. 491–502

Customizing Stata graphs made easy (part 1)

Ben Jann
University of Bern
Bern, Switzerland
ben.jann@soz.unibe.ch

Abstract. The overall look of Stata's graphs is determined by so-called scheme files. Scheme files are system components; that is, they are part of the local Stata installation. In this article, I argue that style settings deviating from default schemes should be part of the script producing the graphs, rather than being kept in separate scheme files, and I present a simple tool called `grstyle` that supports such practice.

Keywords: gr0073, grstyle, graph, graphics, scheme files

1 Introduction

Graphs are ubiquitous in scientific research. They serve many purposes such as data analysis or presentation of results. Depending on properties of the data, the type of analysis, the nature of results, the context in which the graphs are used, or the audience to which the graphs are presented, graphs may look different. No universal best style for graphs serves all purposes, and in many cases, the style of graphs must be tailored to the specific application. To reduce the amount of typing required to generate graphs conforming to a particular overall look, Stata features so-called scheme files. Scheme files contain statements that define the default shape, color, and size of the various elements that compose a graph. There are several official scheme files, such as `s2color` (the factory default), `sj`, or `s1mono`, that are part of any Stata installation (see [G-4] **schemes intro**), but users can also create additional scheme files for their own needs and add them to the system. Furthermore, some users provide custom scheme files for public use that can, for example, be obtained from the *Stata Journal* net site or the SSC archive (see [R] **ssc**). Examples are Juul (2003), Newson (2005), Atz (2011), Briatte (2013), Morris (2013), Morris (2015), Bischof (2017a), and Bischof (2017b). Furthermore, Buchanan (2015) provides a powerful framework for generating new schemes.

In essence, scheme files are system components that can be added to the local installation to extend Stata's functionality. Adding custom schemes to the system may make sense if one regularly produces specific types of graphs that differ substantially from the default looks provided by official Stata. In many cases, however, just a few small modifications of a default scheme would do. In such cases, I believe that the style settings determining the look of the graphs should be part of the script producing the graphs. Having to maintain a separate scheme file is unnecessarily complicated and, worse, may prevent researchers from customizing their graphs at all.

In this article, I present a new command called `grstyle` that allows users to customize the overall look of graphs from within a do-file without having to fiddle with external scheme files. The advantage of using `grstyle` over manually editing a scheme file is that everything needed to reproduce the graphs can be included in a single do-file. No scheme files must be copied, for example, if the graphs are to be reproduced on a different computer.

`grstyle` does nothing that could not be done by manually editing a scheme file. In fact, `grstyle` simply creates a scheme file on the fly and loads it as the default scheme. The difference is that handling the scheme file is fully automated, so the user does not have to worry about it. Thus, `grstyle` makes customizing Stata graphs easy.

2 Syntax

Syntax and usage of `grstyle` are as follows:

`set scheme` *schemename*

`grstyle init` [*newscheme*`, path(`*path*`)` `replace`]

`grstyle` *scheme entry*

⋮

graph command

⋮

`grstyle clear` [`, erase`]

First, select a scheme to be used as the basis for the custom style settings; see [G-2] **set scheme**. The default scheme according to factory settings is `s2color`. Hence, `s2color` will be used as the basis if you omit the `set scheme` command.

After that, initialize the custom settings by typing `grstyle init`. Optionally, if *newscheme* is specified, a new scheme containing the custom settings will be created and stored in the file `scheme-`*newscheme*`.scheme` in the current working directory. Option `path()` specifies an alternative directory for storing the scheme file,[1] and option `replace` allows overwriting the file if it already exists. If *newscheme* is omitted, `grstyle` manages the settings in the background; `path()` and `replace` are not allowed in this case.

1. An absolute or relative path may be provided in `path()`. Note that the graphs using the scheme will display correctly only if the scheme file is stored in a location where it is found by Stata; see [U] **17.5 Where does Stata look for ado-files?** and [P] **sysdir**.

Then, record the custom style settings using one or more `grstyle` *scheme entry* commands. The syntax of *scheme entry* is described in `help scheme entries`; see below for examples.

After recording the desired settings, run the commands creating your graphs.

At the end, if you want to drop the custom settings, type `grstyle clear`. `grstyle clear` is needed only if you want to restore the original settings within the same Stata session; changes made by `grstyle` are temporary, and restarting Stata will remove the custom settings. Furthermore, `grstyle init` automatically runs `grstyle clear` before initializing new settings.[2] Option `erase` causes the scheme file created by `grstyle` to be erased from disk. Use this option if you specified *newscheme* when calling `grstyle init` and do not want to keep the scheme file. The default is not to erase the scheme file.

Two more commands are available. `grstyle type` views the current settings; it types the contents of the custom scheme file to Stata's Results window. Furthermore, `grstyle set` provides precoded collections of scheme entries and automizes handling certain attributes such as colors or sizes. `grstyle set` is discussed in Jann (Forthcoming).

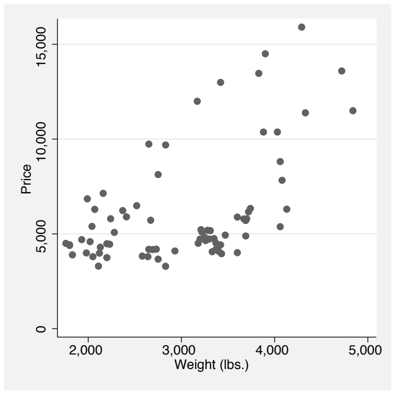
3 Examples

Basic usage

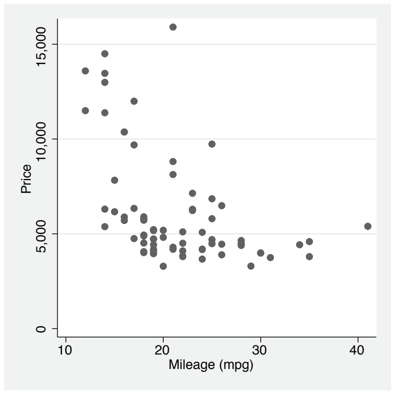
Say that you want the size of your graphs to be 2 × 2 inches instead of Stata's default size. Instead of adding options `ysize(2)` and `xsize(2)` to each graph command, you could use `grstyle` as follows:

2. However, note that `clear all` will not clear `grstyle`'s graph settings. You need to type `grstyle clear` or restart Stata to remove the settings.

```
. set scheme sj
. grstyle init
. grstyle graphsize x 2
. grstyle graphsize y 2
. sysuse auto
(1978 Automobile Data)
. scatter price weight
```



```
. scatter price mpg
```



```
. grstyle clear
```

Some useful scheme entries

As illustrated, using `grstyle` is straightforward and simple. The real difficulty, of course, is to know how the scheme entries must look. Type

```
. help scheme entries
```

to view the corresponding documentation. Scheme entries have their own idiosyncratic syntax, but the documentation is well structured, and it usually does not take long to find the relevant information. Here is an example that illustrates several modifications I find useful:[3]

```
. set scheme sj
. grstyle init
```

- Omit background shading.

```
. grstyle color background white
```

- Use horizontal text for tick labels on the Y (vertical) axis (the `sj` default is to use vertical text, which makes the labels hard to read).

```
. grstyle anglestyle vertical_tick horizontal
```

- Draw vertical grid lines; that is, draw grid lines on the X (horizontal) axis (the `sj` default is to draw horizontal grid lines only).

```
. grstyle yesno draw_major_hgrid yes
```

- Always include minimum and maximum grid lines (by default, minimum and maximum grid lines are omitted if there are no data in the proximity of these grid lines; I find this behavior odd, especially if one is producing a series of graphs that use the same scale for the axes for sake of comparison).

```
. grstyle yesno grid_draw_min yes
. grstyle yesno grid_draw_max yes
```

- Change color, width, and pattern of grid lines (by default, the grid lines have the same color as the background and are a bit thicker than the axis lines; this no longer makes sense if the background shading is removed).

```
. grstyle color major_grid gs8
. grstyle linewidth major_grid thin
. grstyle linepattern major_grid dot
```

- Place the legend on the lower right of the plot region, and remove the frame.

```
. grstyle clockdir legend_position 4
. grstyle numstyle legend_cols 1
. grstyle linestyle legend none
```

- Use thicker lines in line plots.

```
. grstyle linewidth plineplot medthick
```

3. Also see the schemes provided by Bischof (2017b) that contain similar modifications, among other things.

- Make markers transparent (only for the first two plot styles for case of exposition;[4] requires Stata 15).

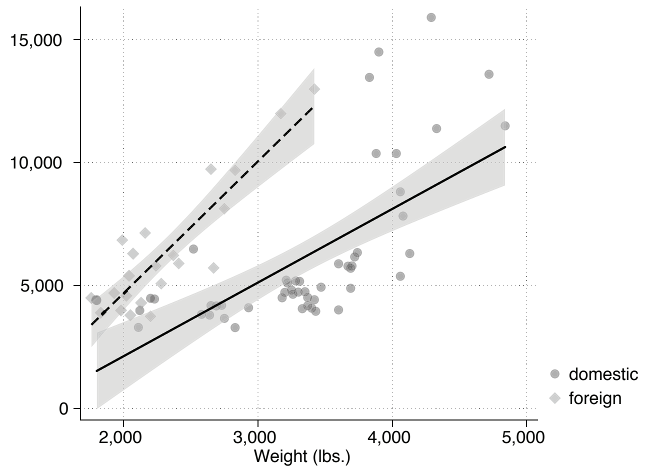
```
. grstyle color p1markline gs6%0
. grstyle color p1markfill gs6%50
. grstyle color p2markline gs10%0
. grstyle color p2markfill gs10%50
```

- Make confidence intervals transparent (requires Stata 15).

```
. grstyle color ci_area gs12%50
. grstyle color ci_arealine gs12%0
```

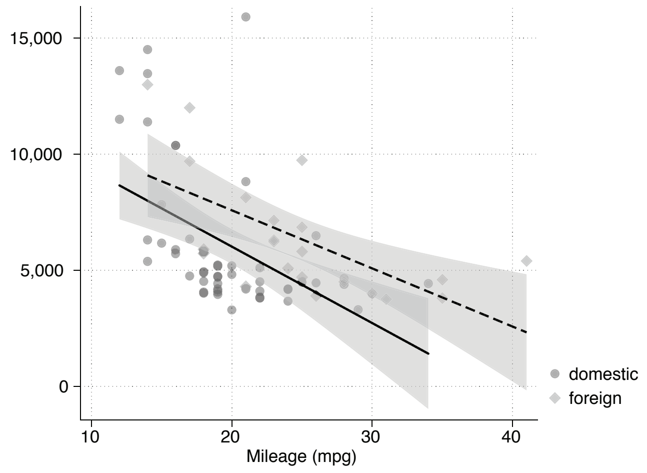
- Now, create some graphs using the modified style.

```
. sysuse auto
(1978 Automobile Data)
. twoway (scatter price weight if foreign==0)
>        (scatter price weight if foreign==1)
>        (lfitci price weight if foreign==0, clstyle(p1line))
>        (lfitci price weight if foreign==1, clstyle(p2line))
>        , legend(order(1 "domestic" 2 "foreign"))
```



4. Unfortunately, the colors have to be spelled out explicitly for each plot style, and it does not seem to be possible to just make all markers transparent while keeping the default colors; type `viewsource scheme-`*scheme*`.scheme`, and look for entries such as `color p1`, `color p2`, etc. to find out about the default colors of a scheme.

```
. twoway (scatter price mpg if foreign==0)
>        (scatter price mpg if foreign==1)
>        (lfitci price mpg if foreign==0, clstyle(p1line))
>        (lfitci price mpg if foreign==1, clstyle(p2line))
>        , legend(order(1 "domestic" 2 "foreign"))
```



Absolute text sizes and line widths

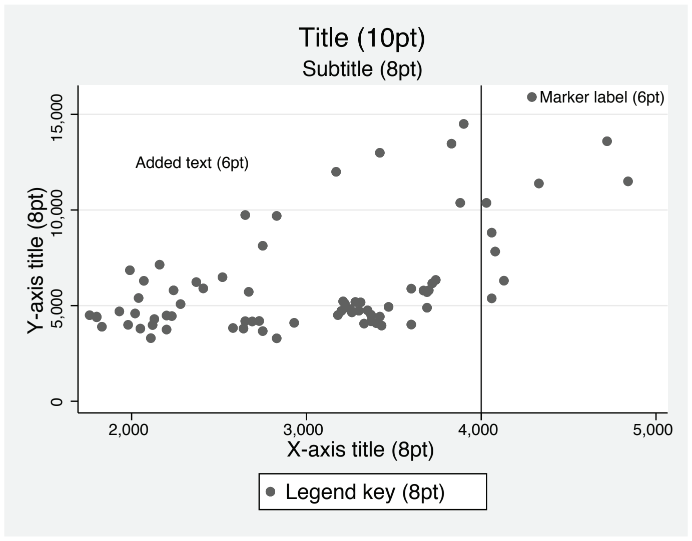
In some situations, for example, because of requirements by a publisher, one needs to set text sizes and line widths to specific absolute values. This is difficult to accomplish in Stata because the sizes of objects on a graph are determined relative to the size of the graph. That is, if you change the graph size, text sizes and line widths may change. Given a specific graph size, it is difficult to know how large exactly the text sizes and line widths will happen to be.

If you want to control the absolute size of text and line widths, you must create a scheme that sets the relative sizes such that for a given graph dimension, the specified relative sizes lead to the desired absolute sizes. Furthermore, you must create one such scheme for each graph size you intend to use. The formula to compute the relative sizes is not difficult, but it is a lot of work to do the computations and compile the corresponding schemes manually.

More convenient is to change the settings on the fly using `grstyle`. The width and height of Stata graphs are expressed in inches. One inch is equal to 72 points (or 2.54 centimeters). Relative sizes are expressed in percent of the minimum of width and height of the graph. That is, if a graph is 5.5 inches wide and 4 inches high, then the reference size is 4 inches (or $4 \times 72 = 288$ points), and a relative size of, say, 5 is equal to $5 \times (4 \times 72)/100 = 14.4$ points. Conversely, if you want the size of an object on this graph to be 10 points, you need to set its relative size to $10/(4 \times 72) \times 100 = 3.472222$. Here is an example in which I use `grstyle` to set various text sizes and line widths according to this formula. The goal is to construct a graph that is 9 cm wide and 7 cm high, uses 0.5 pt lines for axes, major grid lines, etc., and uses text sizes between 6 and 10 points depending on object:[5]

```
. local xsize = 9 / 2.54               // 9cm wide
. local ysize = 7 / 2.54               // 7cm high
. local rsize = min(`xsize´, `ysize´)  // reference size
. foreach pt in .5 3 6 8 10 {          // compute relative sizes
  2.     local nm: subinstr local pt "." "_" // so that ".#" is "_#"
  3.     local `nm´pt = `pt´ /(`rsize´*72)*100
  4. }
. grstyle init
. grstyle graphsize x         `xsize´
. grstyle graphsize y         `ysize´
. grstyle gsize heading        `10pt´ // title
. grstyle gsize subheading      `8pt´ // subtitle
. grstyle gsize axis_title      `8pt´
. grstyle gsize tick_label      `6pt´
. grstyle gsize key_label       `8pt´ // key labels in legend
. grstyle gsize plabel          `6pt´ // marker labels
. grstyle gsize text_option     `6pt´ // added text
. grstyle symbolsize p          `3pt´ // marker symbols
. grstyle linewidth axisline   `_5pt´
. grstyle linewidth tick       `_5pt´
. grstyle linewidth major_grid `_5pt´
. grstyle linewidth legend     `_5pt´ // legend outline
. grstyle linewidth xyline     `_5pt´ // added lines
. sysuse auto
(1978 Automobile Data)
. generate str mlab = "Marker label (6pt)" if price>15000
(73 missing values generated)
. twoway (scatter price weight, mlabel(mlab)), title("Title (10pt)")
>     subtitle("Subtitle (8pt)") xtitle("X-axis title (8pt)")
>     ytitle("Y-axis title (8pt)") legend(on order(1 "Legend key (8pt)"))
>     text(12500 2400 "Added text (6pt)") xline(4000)
```

5. The list of scheme entries in this example covers some of the most common elements but is not exhaustive. Depending on the application, you may need to set the size of additional elements. Furthermore, to fully control the layout of the graph, you would also have to set elements such as gaps, margins, and tick length.



To generate a graph with different dimensions that uses the same sizes for text and line widths, simply change the `xsize` and `ysize` macros on the first two lines. An alternative is to wrap the `grstyle` commands into a little program as follows:

```
program graphsetup
    args x y
    local xsize = `x´ / 2.54
    local ysize = `y´ / 2.54
    local rsize = min(`xsize´, `ysize´)
    foreach pt in .5 3 6 8 10 {
        local nm: subinstr local pt "." "_"
        local `nm´pt = `pt´ /(`rsize´*72)*100
    }
    grstyle init
    grstyle graphsize x          `xsize´
    grstyle graphsize y          `ysize´
    grstyle gsize heading         `10pt´
    grstyle gsize subheading       `8pt´
    grstyle gsize axis_title       `8pt´
    grstyle gsize tick_label       `6pt´
    grstyle gsize key_label        `8pt´
    grstyle gsize plabel           `6pt´
    grstyle gsize text_option      `6pt´
    grstyle symbolsize p           `3pt´
    grstyle linewidth axisline   `_5pt´
    grstyle linewidth tick       `_5pt´
    grstyle linewidth major_grid `_5pt´
    grstyle linewidth legend     `_5pt´
    grstyle linewidth xyline     `_5pt´
end
```

You can then use the program to quickly switch between different graph dimensions while keeping text sizes and line widths fixed:

```
. graphsetup 9 7    // 9cm wide and 7cm high
. graph commands
. graphsetup 9 12   // 9cm wide and 12cm high
. graph commands
. etc.
```

You do not need to run `grstyle clear` between the `graphsetup` commands, because `grstyle init`, which is called within `graphsetup`, will clear the previous settings.

4 Limitations and further remarks

Unless a custom scheme name is specified, `grstyle` works by creating a new scheme called `_GRSTYLE_` and storing it in file `scheme-_GRSTYLE_.scheme` in the `PERSONAL` ado-file directory (see [U] **17.5.2 Where is my personal ado-directory?** and [P] **sysdir**). The relevance of this information is that there may be two complications:

- Stata must have writing rights in the `PERSONAL` ado-file directory. Furthermore, if the `PERSONAL` ado-file directory does not exist, Stata must have the necessary rights to create the directory.
- If multiple Stata sessions are executed in parallel and the sessions use the same `PERSONAL` ado-file directory, then the `grstyle` commands in the different sessions will all write to the same file. To keep the style settings distinct in this case, you may want to provide a unique custom scheme name with `grstyle init` in each session.

Furthermore, note that `grstyle` does not check whether a submitted style setting is a valid scheme entry. It just copies the provided specification to the temporary scheme file as is. Hence, if you misspell the scheme entry, no warning message will be displayed by `grstyle`. Whether a subsequent graph command will display an error or just ignore the misspelled scheme entry depends on context. Scheme entries have the following syntax:

attribute element style

An example is "`color background white`", where "`color`" is the attribute to be set, "`background`" is the graph element to be affected, and "`white`" is the desired style. If you misspell *attribute* or *element* (that is, if you specify an attribute or element that is unknown to Stata), the most likely thing to happen is that the graph command ignores the scheme entry, it will have no effect, and no warning message will be displayed. If you misspell *style*, different things may happen. If the graph does not contain the affected element, the scheme entry will again be ignored, and no warning message will be displayed. Otherwise, the graph command will either abort with error (for example, if *style* is a numeric value and you specify a value outside the allowed range) or display a warning message stating that the style has not been found and that default attributes will be used for the graph. In any case, if a graph does not seem to adopt your style settings, it is always a good idea to double-check the spelling of your scheme entries.

Another reason why your style settings may not have an effect is that some of the higher-level graph commands (that is, commands other than `graph` that internally call `graph`) apply explicit style settings to certain elements and, hence, override the defaults provided by `grstyle`. An example is `marginsplot`, which internally applies option `pstyle(p1)` (or `pstyle(p2)`, `pstyle(p3)`, etc., depending on context) to the confidence intervals so that point estimates and confidence intervals are displayed using the same style. This makes it difficult to modify point estimates and confidence intervals individually. For example, if you apply option `recastci(rarea)` to `marginsplot` so that the confidence intervals are displayed as areas instead of capped spikes, using `grstyle` to set the attributes of elements `p1area` and `p1arealine` will have no effect unless you also add option `ci1opts(astyle(p1area))` to `marginsplot`. If your graph contains multiple series of estimates and you want all confidence areas to look the same, it is probably easiest to set the attributes of the `ci` elements (see `help scheme ci plots`) and then add option `ciopts(astyle(ci))` to `marginsplot`.

Finally, note that `grstyle` maintains global macros `GRSTYLE_FN` (path and name of the scheme file used to store the custom settings); `GRSTYLE_SN` (the corresponding scheme name); `GRSTYLE_SN0` (name of the scheme that was active when initializing `grstyle`); and, depending on context, `GRSTYLE_RSIZE` (the reference size for size calculation by `grstyle set`; see Jann [Forthcoming]). Do not modify these globals. To remove the globals and restore the initial settings, type `grstyle clear`.

5 References

Atz, U. 2011. scheme_tufte: Stata module to provide a Tufte-inspired graphics scheme. Statistical Software Components S457285, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457285.html.

Bischof, D. 2017a. g538schemes: Stata module to provide graphics schemes for http://fivethirtyeight.com. Statistical Software Components S458404, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458404.html.

———. 2017b. New graphic schemes for Stata: plotplain and plottig. *Stata Journal* 17: 748–759.

Briatte, F. 2013. scheme-burd: Stata module to provide a ColorBrewer-inspired graphics scheme with qualitative and blue-to-red diverging colors. Statistical Software Components S457623, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457623.html.

Buchanan, W. 2015. brewscheme: Stata module for generating customized graph scheme files. Statistical Software Components S458050, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458050.html.

Jann, B. Forthcoming. Customizing Stata graphs made easy (part 2). *Stata Journal*.

Juul, S. 2003. Lean mainstream schemes for Stata 8 graphics. *Stata Journal* 3: 295–301.

Morris, T. 2013. scheme-mrc: Stata module to provide graphics scheme for UK Medical Research Council. Statistical Software Components S457703, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457703.html.

———. 2015. scheme-tfl: Stata module to provide graph scheme, based on Transport for London's corporate colour pallette. Statistical Software Components S458103, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458103.html.

Newson, R. 2005. scheme_rbn1mono: Stata module to provide minimal monochrome graphics schemes. Statistical Software Components S456505, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s456505.html.

About the author

Ben Jann is a professor of sociology at the University of Bern, Switzerland. His research interests include social science methodology, statistics, social stratification, and labor market sociology. He is the principle investigator of TREE, a large-scale multicohort panel study in Switzerland on transitions from education to employment (http://www.tree.unibe.ch).