The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–24

The brewscheme toolkit for Data Visualization in Stata

William R. Buchanan
Minneapolis Public Schools
Minneapolis, MN, USA
William.Buchanan@mpls.k12.mn.us

Abstract. This article describes the `brewscheme` package, providing tools to help users to generate customized scheme files and a tool to proof data visualizations for perceptability among individual with color sight impairments. The `brewscheme` toolkit provides more than 10 different commands to help Stata users leverage the data visualization capabilities provided by the graph commands in Stata. Although Stata provides ample flexibility for customizing/specifying the aesthetic properties of data visualizations, customizing the graphs could require a substantial increase in the amount of code needed to generate the graph; the problem is compounded in production environments where standardized aesthetics may be required. The `brewscheme` package attempts to make it easier for users to reduce the amount of code they need to write to create graphs that meet their aesthetic needs and to minimize the code needed to implement those aesthetics across graphs, programs, and datasets.

Keywords: st0000, graphics, data visualization, colorblind, accessibility, brewcolors, brewproof, brewscheme, brewtheme, libbrewscheme

1 Introduction

While Stata provides a robust platform for developing data visualizations, users regularly encounter challenges when trying to leverage these capabilities for their use. Cox (2013), Cox (2014), Pisati (2007), & Radyakin (2009) illustrate different methods for generating customized data visualizations and methods to properly prepare data to generate the data visualizations that users would like to create from Stata. The Statalist and StataJournal also contain numerous resources for data visualization in Stata. In particular, Mitchell (2012) provides a comprehensive treatment of Stata's native graphics capabilities and an exploration of how the optional parameters available to the graph commands can be used to alter the aesthetics of the visualizations. Mitchell (2012) also includes a brief introduction to `.scheme` files in Stata. However, not all users share positive opinions of Stata's graphics capabilities as noted in Anonymous (2013), Bischof (2015), Briatte (2013), & Hsiang (2013) for example, and summarized in Buchanan (2015). In short, many users are dissatisfied with the default aesthetic choices, particularly with the `s2color` scheme.

Despite several attempts to provide users with the resources needed to create scheme files Rising (2010), to use the graph recorder functionality to simulate altered schemes Crow (2008), or by providing fixed alternate schemes such as those provided by Atz

(2011), Bischof (2015), Briatte (2013), Hsiang (2013), and Juul (2003), no comprehensive solution for programmatically generating scheme files was available until an earlier version of `brewscheme` discussed at the 2015 US Stata Users Group conference (Buchanan (2015)). The earliest implementation of `brewscheme` was not without flaws either. In particular, the earliest version of the package only allowed users to specify the colors that could be used for different types of graphs. Unlike the solutions proposed by Atz (2011), Bischof (2015), Briatte (2013), Hsiang (2013), and Juul (2003), the `brewscheme` package provides a significantly more flexible toolkit where the number of schemes that can be created — while finite — approaches inf.

1.1 What makes `brewscheme` different?

Several authors have implemented some similar features to those available in the `brewscheme` package. Briatte (2013), Hsiang (2013), and Pisati (2007) all include capabilities related to the color palettes developed by Brewer (2002). In the case of Briatte (2013) and Hsiang (2013) the schemes focus on a single palette, and while Pisati (2007) provides more comprehensive coverage of the ColorBrewer (Brewer (2002)) palettes it is not extensible and is limited to only those palettes that are hardcoded into the program. Others, such as Atz (2011) and Juul (2003) have attempted to integrate suggestions of Tufte (2001), and in one instance, Bischof (2015), there is an attempt to address color sight impairment and emulation of other popular aesthetic palettes (e.g., the ggplot2 package in R Wickham (2009)).

Unlike these packages, `brewscheme` parses the color palettes developed by Brewer (2002) from their source when building the dataset with the available color palettes, includes color palettes implemented in the D3js visualization library developed by Bostock et al. (2011), includes color palettes with semantic meanings researched by Lin et al. (2013), includes the default color palettes available in ggplot2 Wickham (2009), and includes culturally derived color palettes commonly found in data visualizations popular in the K-12 educational community Buchanan (2014). Additionally, unlike previous attempts to implement the work of Tufte (2001) in Stata scheme files, the `brewtheme` command provides a dafualt set of parameter values that define this type of behavior while providing users with the flexibility to deviate from these settings at their discretion. Lastly, while Bischof (2015) provided a `.scheme` file that is hoped to be sensitive to the needs of individuals with color sight impairments, the `brewproof` command allows users to see how their graphs might look to individuals with achromatopsia (complete color sight impairment), protanopia (impairment in the perception/differentiation of the color red), deuteranopia (impairment in the perception/differentiation of the color green), and tritanopia (impairment in the perception/differentiation of the color blue).

Colorspaces

One of the challenges of integrating all of these sources is the different use of colorspace by the different authors/sources. For example, Brewer (2002) provides an RGB values for colors in the color palettes, while Bostock et al. (2011) use hexadecimal values

to represent the colors in RGB colorspace, and Wickham (2009) uses a simple linear interpolation over the hue parameter in HSB colorspace to generate the colors used by default in the ggplot2 package in R. Although the conversion of base 16 to base 10 values may not present a major challenge, conversion between other color spaces can be more difficult and require intermediate transformations across several color spaces (see Lindbloom (2001) for additional information). At present, `brewscheme` provides limited tools for colorspace conversions, but does include a hexadecimal to RGB conversion command as well as a Java-based plugin that provides some colorspace transformation capabilities as part of its primary function of providing color interpolation methods.

Color sight impairment

Brettel et al. (1997) and Viénot et al. (1999) provide expositions on methods to transform colors in ways that simulate color sight impairments, more specifically protanopia, deuteranopia, and tritanopia. Viénot et al. (1999) build on their earlier work in Brettel et al. (1997), to provide a description of the methodology required to transform a given color in RGB colorspace to LMS colorspace, apply the necessary manipulations to simulate color sight impairments, and transform the values back to RGB color space. An implementation of these algorithms in JavaScript is available from Wickline (2014). And was implemented in Mata in the `brewscheme` package.

The remainder of the article will focus on the use of the package by end-users and will include brief examples, with references to view the color images.

2 Installation and Getting Started

The `brewscheme` package can be installed using:

```
net inst brewscheme, from(``http://wbuchanan.github.io/brewscheme/'')
```

The first time you run the program and after an update the the `libbrewscheme` Mata library, the programs will check when the Mata source code was created and will update itself if necessary. Additionally, the first time the `brewscheme` program is run, it may take upwards of a few minutes — depending on your internet connection speed and your computer as well — because the program will need to build the database of color palettes used to generate the `.scheme` files which includes parsing the color palettes from https://www.ColorBrewer2.org. After installing the package, the next step is to build the database of named color styles that already exist in your Stata using the `brewcolordb` command.

`brewcolordb` [`,` `display` `refresh`]

`display` is an option that writes the named color style and corresponding RGB values to the console.

`refresh` is an option to overwrite an existing color database.

Macros
`r(`*colorname*`)` RGB value

The `brewcolordb` command searches for named color styles, parses the contents of the files and builds a database of these files along with the RGB values used to simulate how the color would be perceived by individuals with achromatopsia, protanopia, deuteranopia, and tritanopia. Additionally, based on the information provided by Wiggins (2004), the program also installs named color styles corresponding to the colorsight impaired versions of the colors. The modified colors can all be accessed using the naming convention [color name]_[impairment name]. For example, ltblue_tritanopia would select the tritanopia simulated value for the color ltblue.

3 Creating customized scheme files

3.1 brewtheme

The `.theme` file is specific to `brewscheme` and provides a method to encapsulate aesthetic parameters which may be global in scope in a reusable way for the generation of `.scheme` files. The `brewtheme` command generates these files for you, but is not required to generate customized `.scheme` files. The optional arguments for `brewtheme` all use key/value pairs delimited by quotation marks. In other words, to pass an argument to any of the options they should use the following form:

```
optionname(``key1 value1'' ``key2 value2'' ``...''  ``keyn valuen'')
```

brewtheme API

`brewtheme` *theme name* [, `style(`*string*`)` `anglestyle(`*string*`)` `areastyle(`*string*`)`
`arrowstyle(`*string*`)` `axisstyle(`*string*`)` `barlabelpos(`*string*`)`
`barlabelstyle(`*string*`)` `barstyle(`*string*`)` `bygraphstyle(`*string*`)`
`clegendstyle(`*string*`)` `clockdir(`*string*`)` `color(`*string*`)` `compass2dir(`*string*`)`
`compass3dir(`*string*`)` `connectstyle(`*string*`)` `dottypestyle(`*string*`)`
`graphsize(`*string*`)` `graphstyle(`*string*`)` `gridlinestyle(`*string*`)`
`gridringstyle(`*string*`)` `gridstyle(`*string*`)` `gsize(`*string*`)` `horizontal(`*string*`)`
`labelstyle(`*string*`)` `legendstyle(`*string*`)` `linepattern(`*string*`)`
`linestyle(`*string*`)` `linewidth(`*string*`)` `margin(`*string*`)` `medtypestyle(`*string*`)`
`numstyle(`*string*`)` `numticks(`*string*`)` `piegraphstyle(`*string*`)`
`pielabelstyle(`*string*`)` `plotregionstyle(`*string*`)` `relativepos(`*string*`)`
`relsize(`*string*`)` `special(`*string*`)` `starstyle(`*string*`)` `sunflowerstyle(`*string*`)`

symbol(*string*) symbolsize(*string*) textboxstyle(*string*)
tickposition(*string*) tickstyle(*string*) ticksetstyle(*string*)
verticaltext(*string*) yesno(*string*) zyx2rule(*string*) zyx2style(*string*)
loadthemedata]

abovebelow an optional argument with a single key: star.

anglestyle an optional argument with the following keys: horizontal_tick, vertical_tick, clegend, p, parrow, and parrowbarb. See [G-4] ***anglestyle*** or use `graph query anglestyle` for additional information.

areastyle an optional argument with the following keys: foreground, background, plotregion, inner_plotregion, twoway_plotregion, twoway_iplotregion, bar_plotregion, bar_iplotregion, hbar_plotregion, hbar_iplotregion, dot_plotregion, dot_iplotregion, box_plotregion, box_iplotregion, hbox_plotregion, hbox_iplotregion, combine_plotregion, combine_iplotregion, bygraph_plotregion, bygraph_iplotregion, matrixgraph_plotregion, matrixgraph_iplotregion, matrix_plotregion, matrix_iplotregion, legend, legend_key_region, legend_inkey_region, inner_legend, clegend, clegend_preg, clegend_inpreg, clegend_outer, clegend_inner, graph, inner_graph, bygraph, inner_bygraph, piegraph, piegraph_region, inner_pieregion, inner_piegraph, combinegraph, combinegraph_inner, matrix_label, matrix_ilabel, ci, ci2, histogram, dendrogram, dotchart, sunflower, sunflowerlb, and sunflowerdb. See [G-4] ***areastyle*** or use `graph query areastyle` for additional information.

arrowstyle an optional argument with the following keys: default and editor. See [G-4] ***arrowstyle*** or use `graph query arrowstyle` for additional information.

axisstyle is an optional argument with the following keys: horizontal_default, vertical_default, horizontal_nogrid, vertical_nogrid, bar_super, dot_super, bar_group, dot_group, bar_var, dot_var, bar_scale_horiz, bar_scale_vert, dot_scale_horiz, dot_scale_vert, box_scale_horiz, box_scale_vert, matrix_horiz, matrix_vert, sts_risktable, and clegend. See [G-4] ***axisstyle*** or use `graph query axisstyle` for additional information.

barlabelpos an optional argument with a single key: bar.

barlabelstyle an optional argument with a single key: bar.

barstyle an optional argument with the following keys: default, dot, and box.

bygraphstyle an optional argument with the following keys: default, bygraph, and combine. See [G-4] ***bystyle*** or use `graph query bystyle` for additional information.

clegendstyle an optional argument with a single key: default. See [G-4] ***clegendstyle*** or use `graph query clegendstyle` for additional information.

clockdir an optional argument with the following keys: title_position, subtitle_position, caption_position, note_position, legend_position, zyx2legend_position, by_legend_position, ilabel,matrix_marklbl, p, legend_title_position, legend_subtitle_position, legend_caption_position, legend_note_position, and clegend_title_position.

color an optional argument with the following keys: background, foreground, symbol, backsymbol, text, body, small_body, heading, subheading, axis_title, matrix_label, label, key_label, tick_label, tick_biglabel, matrix_marklbl, sts_risk_label, sts_risk_title, box, textbox, mat_label_box, text_option, text_option_line, text_option_fill, filled_text, filled, bylabel_outline, reverse_big, reverse_big_line, reverse_big_text, grid, major_grid, minor_grid, axisline, tick, minortick, matrix, matrixmarkline, histback, plotregion, plotregion_line, matrix_plotregion, matplotregion_line, legend, legend_line, clegend, clegend_outer, clegend_inner, clegend_line, pboxlabelfill, plabelfill, pmarkback, and pmarkbkfill.

compass2dir an optional argument with the following keys: p, key_label, legend_fillpos, legend_key, text_option, graph_aspect, and editor.

compass3dir an optional argument with a single key: p.

connectstyle an optional argument with a single key: p. See [G-4] ***connectstyle*** or use `graph query connectstyle` for additional information.

dottypestyle an optional argument with a single key: dot.

graphsize an optional argument allowing users to specify the x and y values defining the width and height of the graph image.

graphstyle an optional argument with the following keys: default, graph, and matrix-graph.

gridlinestyle an optional argument with a single key: default.

gridringstyle an optional argument with the following keys: spacers_ring, title_ring, subtitle_ring, caption_ring, note_ring, legend_ring, zyx2legend_ring, clegend_ring, by_legend_ring, legend_title_ring, legend_subtitle_ring, legend_caption_ring, legend_note_ring, and clegend_title_ring.

gridstyle an optional argument with the following keys: major and minor. See [G-4] ***gridstyle*** or use `graph query gridstyle` for additional information.

gsize an optional argument with the following keys: gap, text, body, small_body, heading, subheading, axis_title, matrix_label, label, small_label, matrix_marklbl, key_label, note, star, text_option, dot_rectangle, axis_space, axis_title_gap, tick, minortick, tickgap, notickgap, tick_label, tick_biglabel, minortick_label, filled_text, reverse_big, alternate_gap, title_gap, key_gap, key_linespace, star_gap, legend_colgap, label_gap, matrix_mlblgap, barlabel_gap, legend_row_gap, legend_col_gap, legend_key_gap, legend_key_xsize, legend_key_ysize, zyx2legend_key_gap, zyx2legend_key_xsize, zyx2legend_key_ysize, zyx2rowgap, zyx2colgap, clegend_width, clegend_height, pie_explode, pielabel_gap, plabel, pboxlabel, sts_risktable_space, sts_risktable_tgap, sts_risktable_lgap, sts_risk_label, sts_risk_title, and sts_risk_tick. These keys take a string value like: zero, third_tiny, half_tiny, tiny, minuscule, vsmall, small, medsmall, medium, medlarge, large, huge, or vhuge.

horizontal an optional argument with the following keys: heading, subheading, label, key_label, body, small_body, axis_title, matrix_label, filled, text_option, editor,

sts_risk_label, and sts_risk_title.

`labelstyle` an optional argument with the following keys: ilabel, matrix, editor, and sunflower.

`legendstyle` an optional argument with the following keys: default and zyx2. See [G-4] ***legendstyle*** or use `graph query legendstyle` for additional information.

`linepattern` an optional argument with the following keys: foreground, background, ci, ci_area, histogram, dendrogram, grid, major_grid, minor_grid, axisline, tick, minortick, xyline, refline, refmarker, matrixmark, dots, dot, dot_area, dotmark, pie, legend, clegend, plotregion, sunflower, matrix_plotregion, text_option, zyx2, p, and pmark.

`linestyle` an optional argument with the following keys: background, foreground, symbol, boxline, textbox, axis, axis_withgrid, zero_line, tick, minortick, star, ci, ci_area, ci2, ci2_area, histogram, histback, dendrogram, grid, major_grid, minor_grid, xyline, refline, refmarker, matrixmark, matrix, dotchart, dotchart_area, dotmark, box_whiskers, box_median, pie_lines, legend, clegend, clegend_outer, clegend_inner, clegend_preg, mat_label_box, reverse_big, plotregion, matrix_plotregion, dots, editor, sunflower, sunflowerlb, sunflowerlf, sunflowerdb, sunflowerdf, text_option, sts_risktable, zyx2, pmarkback, pboxmarkback, plabel, and pboxlabel. See [G-4] ***linestyle*** or use `graph query linestyle` to see the available linestyles on your system.

`linewidth` an optional argument with the following keys: thin, medium, p, foreground, background, grid, major_grid, minor_grid, axisline, tick, tickline, minortick, ci, ci_area, ci2, ci2_area, histogram, dendrogram, xyline, refline, refmarker, matrixmark, dots, dot_line, dot_area, dotmark, plotregion, legend, clegend, pie, reverse_big, sunflower, matrix_plotregion, text_option, zyx2, and pbar.

`margin` an optional argument with the following keys: graph, twoway, bygraph, combinegraph, combine_region, matrixgraph, piegraph, piegraph_region, matrix_plotreg, matrix_label, mat_label_box, by_indiv, text, textbox, body, small_body, heading, subheading, axis_title, label, key_label, text_option, plotregion, star, bargraph, boxgraph, dotgraph, hbargraph, hboxgraph, hdotgraph, legend, legend_key_region, legend_boxmargin, clegend, cleg_title, clegend_boxmargin, key_label, filled_textbox, filled_box, editor, plabel, plabelbox, pboxlabel, and pboxlabelbox.

`medtypestyle` an optional argument with a single key: boxplot.

`numstyle` an optional argument with the following keys: grid_outer_tol, legend_rows, legend_cols, zyx2rows, zyx2cols, graph_aspect, max_wted_symsize, bar_num_dots, dot_num_dots, dot_extend_high, dot_extend_low, pie_angle that take numeric values. .

`numticks` an optional argument with the following keys: major, horizontal_major, vertical_major, horizontal_minor, vertical_minor, horizontal_tmajor, vertical_tmajor, horizontal_tminor, and vertical_tminor.

`piegraphstyle` an optional argument with a single key: piegraph.

pielabelstyle an optional argument with a single key: default.

plotregionstyle an optional argument with the following keys: graph, twoway, by-graph, combinegraph, combineregion, matrixgraph, bargraph, hbargraph, boxgraph, hboxgraph, piegraph, matrix, matrix_label, legend_key_region, and clegend.

relativepos an optional argument with the following keys: zyx2legend_pos, clegend_pos, and clegend_axispos.

relsize an optional argument with the following keys: bar_gap, bar_groupgap, bar_supgroupgap, bar_outergap, dot_gap, dot_groupgap, dot_supgroupgap, dot_outergap, box_gap, box_groupgap, box_supgroupgap, box_outergap, box_fence, and box_fencecap. The values associated with these keys should be of the form [neg]#pct. Where *neg* would indicate a negative relative size, the # represents a numeric value, and pct is a string literal for percentage.

special an optional argument with the following keys: default_slope1, default_knot1, default_slope2, by_slope1, by_knot1, by_slope2, combine_slope1, combine_knot1, combine_slope2 matrix_slope1, matrix_knot1, matrix_slope2 take numeric values and the keys: matrix_yaxis and matrix_xaxis take string values.

starstyle an optional argument with a single key: default.

symbol is an optional argument with the following keys: sunflower, none, histogram, histback, dots, ci, ci2, ilabel, matrix, refmarker, p, pback, pbarback, and pdotback.

symbolsize an optional argument with the following keys: smallsymbol, star, histogram, histback, dots, ci, ci2, matrix, refmarker, sunflower, backsymbol, backsymspace, p, pback, and parrow.

textboxstyle an optional argument with the following keys: title, subtitle, caption, note, leg_title, leg_subtitle, leg_caption, leg_note, cleg_title, cleg_subtitle, cleg_caption, cleg_note, t1title, t2title, b1title, b2title, r1title, r2title, l1title, l2title, heading, subheading, body, text_option, legend_key, barlabel, axis_title, matrix_label, pielabel, tick, minortick, bigtick, sts_risktable, label, ilabel, key_label, small_label, matrix_marklbl, star, bytitle, and editor. See [G-4] ***textboxstyle*** or use `graph query textboxstyle` for additional information.

tickposition an optional argument with a single key: axis_tick.

tickstyle an optional argument with the following keys: default, major, minor, major_nolabel, minor_nolabel, major_notick, minor_notick, major_notickbig, minor_notickbig, and sts_risktable. See [G-4] ***tickstyle*** or use `graph query tickstyle` for additional information.

ticksetstyle an optional argument with the following keys: major_horiz_default, major_vert_default, minor_horiz_default, minor_vert_default, major_horiz_withgrid, major_vert_withgrid, major_horiz_nolabel, major_vert_nolabel, minor_horiz_nolabel, minor_vert_nolabel, major_horiz_notick, major_vert_notick, minor_horiz_notick, minor_vert_notick, major_horiz_notickbig, major_vert_notickbig, sts_risktable, and major_clegend.

verticaltext an optional argument with the following keys: heading, subheading, label, key_label, body, small_body, axis_title, matrix_label, legend, text_option, and filled.

yesno an optional argument with the following keys: textbox, text_option, connect_missings, cmissings, pcmissings, extend_axes_low, extend_axes_high, extend_axes_full_low, extend_axes_full_high, draw_major_grid, draw_minor_grid, draw_majornl_grid, draw_minornl_grid, draw_major_hgrid, draw_minor_hgrid, draw_majornl_hgrid, draw_minornl_hgrid, draw_major_vgrid, draw_minor_vgrid, draw_majornl_vgrid, draw_minornl_vgrid, draw_major_nl_vgrid, draw_minor_nl_vgrid, draw_majornl_nl_vgrid, draw_minornl_nl_vgrid, draw_major_nl_hgrid, draw_minor_nl_hgrid, draw_majornl_nl_hgrid, draw_minornl_nl_hgrid, draw_major_nt_vgrid, draw_minor_nt_vgrid, draw_majornl_nt_vgrid, draw_minornl_nt_vgrid, draw_major_nt_hgrid, draw_minor_nt_hgrid, draw_majornl_nt_hgrid, draw_minornl_nt_hgrid, draw_major_nlt_vgrid, draw_minor_nlt_vgrid, draw_majornl_nlt_vgrid, draw_minornl_nlt_vgrid, draw_major_nlt_hgrid, draw_minor_nlt_hgrid, draw_majornl_nlt_hgrid, draw_minornl_nlt_hgrid, extend_grid_low, extend_grid_high, extend_minorgrid_low, extend_minorgrid_high, extend_majorgrid_low, extend_majorgrid_high, grid_draw_min, grid_draw_max, grid_force_nomin, grid_force_nomax, xyline_extend_low, xyline_extend_high, alt_xaxes, alt_yaxes, x2axis_ontop, y2axis_onright, use_labels_on_ticks, alternate_labels, swap_bar_scaleaxis, swap_bar_groupaxis, swap_dot_scaleaxis, swap_dot_groupaxis, swap_box_scaleaxis, swap_box_groupaxis, extend_dots, bar_reverse_scale, dot_reverse_scale, box_reverse_scale, box_hollow, box_custom_whiskers, pie_clockwise, by_edgelabel, by_alternate_xaxes, by_alternate_yaxes, by_skip_xalternate, by_skip_yalternate, by_outer_xtitles, by_outer_ytitles, by_outer_xaxes, by_outer_yaxes, by_indiv_xaxes, by_indiv_yaxes, by_indiv_xtitles, by_indiv_ytitles, by_indiv_xlabels, by_indiv_ylabels, by_indiv_xticks, by_indiv_yticks, by_indiv_xrescale, by_indiv_yrescale, by_indiv_as_whole, by_shrink_plotregion, by_shrink_indiv, mat_label_box, mat_label_as_textbox, legend_col_first, legend_text_first, legend_stacked, legend_force_keysz, legend_force_draw, legend_force_nodraw, title_span, subtitle_span, caption_span, note_span, legend_span, zyx2legend_span, clegend_title_span, adj_xmargins, adj_ymargins, plabelboxed, pboxlabelboxed, contours_outline, contours_reversekey, and contours_colorlines.

zyx2rule an optional argument with a single key: contour. See [G-4] ***zyx2rulestyle*** or use `graph query zyx2rulestyle` for additional information.

zyx2style an optional argument with a single key: default. See [G-4] ***zyx2style*** or use `graph query zyx2style` for additional information.

loadthemedata is an optional argument used to load a dataset containing the lines of the `.scheme` file that are copied from the `.theme` file as well as to show the default values used if no `theme` is passed to `brewscheme`.

Examples

If, for example, you wanted to generate a theme that emulated the aesthetics found in graphs created by the ggplot2 package in R (Wickham [2009]), you could use the following syntax:

Example

```
    #d ;
brewtheme ggplot2, numticks(‘‘major 5’’ ‘‘horizontal_major 5’’ ‘‘vertical_major
5’’ ‘‘horizontal_minor 10’’ ‘‘vertical_minor 10’’) color(‘‘plotregion gs15’’
‘‘matrix_plotregion gs15’’ ‘‘background gs15’’ ‘‘textbox gs15’’ ‘‘legend gs15’’
‘‘box gs15’’ ‘‘mat_label_box gs15’’ ‘‘text_option_fill gs15’’ ‘‘clegend gs15’’
‘‘histback gs15’’ ‘‘pboxlabelfill gs15’’ ‘‘plabelfill gs15’’ ‘‘pmarkbkfill
gs15’’ ‘‘pmarkback gs15’’) linew(‘‘major_grid medthick’’ ‘‘minor_grid thin’’
‘‘legend medium’’ ‘‘clegend medium’’) clockdir(‘‘legend_position 3’’) yesno(‘‘draw_major_grid
yes’’ ‘‘draw_minor_grid yes’’ ‘‘legend_force_draw yes’’ ‘‘legend_force_nodraw
no’’ ‘‘draw_minor_vgrid yes’’ ‘‘draw_minor_hgrid yes’’ ‘‘extend_grid_low yes’’
‘‘extend_grid_high yes’’ ‘‘extend_axes_low no’’ ‘‘extend_axes_high no’’) gridsty(‘‘minor
minor’’) axissty(‘‘horizontal_default horizontal_withgrid’’ ‘‘vertical_default
vertical_withgrid’’) linepattern(‘‘major_grid solid’’ ‘‘minor_grid solid’’)
linesty(‘‘major_grid major_grid’’ ‘‘minor_grid minor_grid’’) ticksty(‘‘minor
minor_notick’’ ‘‘minor_notick minor_notick’’)
ticksetsty(‘‘major_vert_withgrid minor_vert_nolabel’’
‘‘major_horiz_withgrid minor_horiz_nolabel’’ ‘‘major_horiz_nolabel major_horiz_default’’
‘‘major_vert_nolabel major_vert_default’’) gsize(‘‘minortick_label zero’’ ‘‘minortick
tiny’’) numsty(‘‘legend_cols 1’’ ‘‘legend_rows 0’’) verticaltext(‘‘legend
top’’);
```

◁

However, others may want to use aesthetics that more closely resemble the s2color scheme. Here is an example of generating a scheme file that will restore the majority of those default values:

Example

```
    #d ;
brewtheme s2default, graphsi(‘‘x 5.5’’ ‘‘y 4’’) numsty(‘‘legend_cols 2’’)
gsize(‘‘text medium’’ ‘‘body medsmall’’ ‘‘small_body vsmall’’ ‘‘heading large’’
‘‘axis_title medsmall’’ ‘‘matrix_label medlarge’’ ‘‘matrix_marklbl small’’
‘‘key_label medsmall’’ ‘‘note small’’ ‘‘star medsmall’’ ‘‘text_option medsmall’’
‘‘minor_tick half_tiny’’ ‘‘tick_label medsmall’’ ‘‘tick_biglabel medium’’ ‘‘title_gap
vsmall’’ ‘‘key_gap vsmall’’ ‘‘key_linespace vsmall’’ ‘‘legend_key_xsize 13’’
‘‘legend_key_ysize medsmall’’ ‘‘clegend_width huge’’ ‘‘pielabel_gap zero’’
‘‘plabel small’’ ‘‘pboxlabel small’’ ‘‘sts_risktable_space third_tiny’’ ‘‘sts_risktable_tgap
zero’’ ‘‘sts_risktable_lgap zero’’) relsize(‘‘bar_groupgap 67pct’’ ‘‘dot_supgroupgap
67pct’’ ‘‘box_gap 33pct’’ ‘‘box_supgroupgap 200pct’’ ‘‘box_outgap 20pct’’ ‘‘box_fence
67pct’’) symbolsi(‘‘smallsymbol small’’ ‘‘histogram medlarge’’ ‘‘ci medium’’
‘‘ci2 medium’’ ‘‘matrix medium’’ ‘‘refmarker medlarge’’ ‘‘parrowbarb zero’’)
color(‘‘background ltbluishgray’’ ‘‘foreground black’’ ‘‘backsymbol gs8’’
‘‘heading dknavy’’ ‘‘box bluishgray’’ ‘‘textbox bluishgray’’ ‘‘mat_label_box
bluishgray’’ ‘‘text_option_line black’’ ‘‘text_option_fill bluishgray’’ ‘‘filled
```

```
bluishgray'' ``bylabel_outline bluishgray'' ``reverse_big navy'' ``reverse_big_line
navy'' ``grid ltbluishgray'' ``major_grid ltbluishgray'' ``minor_grid gs5''
``matrix navy'' ``matrixmarkline navy'' ``histback gold'' ``legend_line black''
``clegend white'' ``clegend_line black'' ``pboxlabelfill bluishgray'' ``plabelfill
bluishgray'') linepattern(``foreground solid'' ``background solid'' ``grid
solid'' ``major_grid solid'' ``minor_grid dot'' ``text_option solid'') linesty(``textbox
foreground'' ``grid grid'' ``major_grid major_grid'' ``minor_grid minor_grid''
``legend legend'') linewidth(``p medium'' ``foreground thin'' ``background
thin'' ``grid medium'' ``major_grid medium'' ``minor_grid thin'' ``tick thin''
``minortick thin'' ``ci_area medium'' ``ci2_area medium'' ``histogram medium''
``dendrogram medium'' ``xyline medium'' ``refmarker medium'' ``matrixmark
medium'' ``dots vvthin'' ``dot_area medium'' ``dotmark thin'' ``plotregion
thin'' ``legend thin'' ``clegend thin'' ``pie medium'' ``sunflower medium''
``text_option thin'' ``pbar vvvthin'') textboxsty(``note small_body'' ``leg_caption
body'') axissty(``bar_super horizontal_nolinetick'' ``dot_super horizontal_nolinetick''
``bar_scale_horiz horizontal_withgrid'' ``bar_scale_vert vertical_withgrid''
``box_scale_horiz horizontal_withgrid'' ``box_scale_vert vertical_withgrid'')
clockdir(``caption_position 7'' ``legend_position 6'' ``by_legend_position
6'' ``p 3'' ``legend_caption_position 7'') gridringsty(``caption_ring 5'' ``legend_caption_ring
5'') anglesty(``vertical_tick vertical'') yesno(``extend_axes_low no'' ``extend_axes_high
no'' ``draw_major_vgrid yes'' ``use_labels_on_ticks no'' ``title_span no''
``subtitle_span no'' ``caption_span no'' ``note_span no'' ``legend_span no'')
barlabelsty(``bar none'');
```

◁

The verbosity of the second example also helps to illustrate differences in parameter values between the s2color scheme and the default values used by `brewtheme`. It is also important to reiterate, that this step is only necessary if you wish to change parameters that are generally more global in scope than the modifications that will occur using the `brewscheme` command. Additionally, while we only specified a single theme file in the command, the `brewtheme` command also constructs parallel versions of the theme where any color values are substituted for one of the simulated color sight impairment types. You can access these theme files directly by appending "_achromatopsia", "_protanopia", "_deuteranopia", or "_tritanopia" to the theme name.

3.2 brewscheme

Like the `brewtheme` command, the `brewscheme` command also generates parallel versions of your scheme for you. The reason for generating these additional `.scheme` files will be discussed later, but the same logic is used for naming of the parallel schemes. However, unlike the `brewtheme` command, the `brewscheme` command has three different methods available to use it:

1. A single color palette used for all graph types
2. A default color palette used for unspecified graph types and separate palettes for

specified graph types, and

3. Individual color palettes for each graph type.

The parameter names for the command all follow a standardized naming convention that will help to shorten the discussion of the individual parameters into groups based on the use cases described above.

brewscheme API

`brewscheme, schemename(`*string*`)` [`allstyle(`*string*`) allcolors(`#`)`
`allsaturation(`#`) barstyle(`*string*`) barcolors(`#`) barsaturation(`#`)`
`scatstyle(`*string*`) scatcolors(`#`) scatsaturation(`#`) areastyle(`*string*`)`
`areacolors(`#`) areasaturation(`#`) linestyle(`*string*`) linecolors(`#`)`
`linesaturation(`#`) boxstyle(`*string*`) boxcolors(`#`) boxsaturation(`#`)`
`dotstyle(`*string*`) dotcolors(`#`) dotsaturation(`#`) piestyle(`*string*`)`
`piecolors(`#`) piesaturation(`#`) sunstyle(`*string*`) suncolors(`#`)`
`sunsaturation(`#`) histstyle(`*string*`) histcolors(`#`) histsaturation(`#`)`
`cistyle(`*string*`) cicolors(`#`) cisaturation(`#`) matstyle(`*string*`)`
`matcolors(`#`) matsaturation(`#`) reflstyle(`*string*`) reflcolors(`#`)`
`reflsaturation(`#`) refmstyle(`*string*`) refmcolors(`#`) refmsaturation(`#`)`
`constart(`*string*`) conEnd(`*string*`) consaturation(`#`) somestyle(`*string*`)`
`somecolors(`#`) somesaturation(`#`) refresh themefile(`*string*`)`
`symbols(`*string*`)`]

`schemename` an option taking a string value that will name the scheme that is created.

`*style` these options are used to specify the name of the color palette to use for that graph type.

`*colors` allows users to specify the number of colors from the palette to use for a given graph type.

`*saturation` a multiplier used to modify the intensity/saturation of the colors for this graph type.

`refresh` is an optional argument used to rebuild the database of color palettes.

`themefile` is an optional argument used to pass the name of a theme to be used to set the global aesthetic parameters.

`symbols` is an optional argument used to set the symbol types used for different layers/graphs.

Examples

If, for example, you wanted to specify a single color palette to use for all graph types, you could pass arguments to the `allstyle` at a minimum:

Example

```
brewscheme, scheme(s2gg1) allsty(ggplot2)
```

◁

In this case, `brewscheme` will use the default `.theme` file – which is created the first time you call `brewscheme` if it does not already exist – and will use a palette of three colors based on the default colors of ggplot2 when only three colors are used Wickham (2009). If, you wanted to do something similar, but using the theme settings that try to emulate the s2color scheme it would only require an additional argument being passed to the `themefile` parameter:

Example

```
brewscheme, scheme(s2gg2) allsty(ggplot2) themef(s2default)
```

◁

And if you wanted to create a scheme that would mirror the ggplot2 (Wickham (2009)) aesthetics with up to five colors per graph type, you would only need to add an additional parameter:

Example

```
brewscheme, scheme(s2gg3) allsty(ggplot2) allc(5) themef(ggplot2)
```

◁

However, this can easily create difficult to decipher graphs when multiple graph types are overlaid using the twoway type graphs. In those cases, it may be helpful to specify default values for graphs which you use less commonly and specify palette and color combinations for graphs that you use more regularly:

Example

```
brewscheme, scheme(manycolorsexample) somest(ggplot2) somec(7) linest(dark2)
linec(3) cist(pastel2) cic(6) scatsty(category10) scatc(10)
```

◁

This last example highlights one of the changes made to `brewscheme` from Buchanan (2015). Internally, `brewscheme` makes calls to the mata function `recycle` — which is distributed with `brewscheme` — to deal with `.scheme` files using only a single value for the `pcycles` attribute. In the case of the example above, `brewscheme` looks across all of the *colors parameters to find the highest argument passed to all of them. Then the `recycle` function is called to automatically recycle the values you specified enough

times to avoid any potential error/warning messages that would be cause if the `pcycles` attribute was set to a higher value than the number of colors defined for a particular graph type; in other words, if pcycle was set to 10 and you created a graph with four or more calls to `twoway line`, Stata would print an error message to the screen indicating that it could not find the color to use defined in the `.scheme` file. It is also possible to specify a distinct color palette for each type of graph:

▷ **Example**

```
   brewscheme, scheme(myriadColorPalettes) barst(paired) barc(12) dotst(prgn)
dotc(7) scatstyle(set1) scatc(9) linest(pastel2) linec(8) boxstyle(accent)
boxc(8) areast(dark2) areac(8) piest(mdepoint) sunst(greys) histst(veggiese)
cist(activitiesa) matst(spectral) reflst(purd) refmst(set3) const(ylgn) cone(puor)
```

◁

4 Proofing your graphs for color impaired perceptibility

Checking the readability and perceptability of your data visualizations is important to ensure your message is easily and consistently understood. One of the major challenges with the use of color in data visualizations is how easily those colors can be perceived by individuals with different forms of color sight impairments. The `brewproof` prefix command was developed to make this process faster and easier for end users. The primary reason that the `brewtheme` and `brewproof` commands generate parallel versions of your `.scheme` and `.theme` files is to make it faster to proof a graph across each of the forms of color sight impairments. The `brewproof` prefix is a wrapper which calls your graph command multiple times, and passes the modified `.scheme` files as arguments on each iteration before combining each of the graphs into a single "proof" copy.

`brewproof, scheme(`*string*`):` *graph command*

`scheme` the scheme file containing the aesthetics you wish to proof.

graph command is any Stata graph command that accepts a scheme parameter

4.1 Examples

If you wanted to see how your data visualizations may be perceived by individuals with color sight impairments, the `brewproof` prefix provides a convenience command to do just that.

▷ **Example**

```
   brewproof, scheme(ggtest2) :  tw lowess mpg weight || scatter mpg weight
if rep78 == 1 || scatter mpg weight if rep78 == 2 || scatter mpg weight if
```

```
rep78 == 3 || scatter mpg weight if rep78 == 4 || scatter mpg weight if rep78
== 5, legend(order(2 ‘‘1978 Repair Record = 1’’ 3 ‘‘1978 Repair Record = 2’’
4 ‘‘1978 Repair Record = 3’’ 5 ‘‘1978 Repair Record = 4’’ 6 ‘‘1978 Repair
Record = 5’’))
```

◁

The result of the proofer program can be viewed at the project page https://wbuchanan.github.io/brewscheme/brewproof/. The example shown there uses the ggplot2 (Wickham [2009]) inspired `.theme` and `.scheme` files described above (with minor modifications). As you'll see, the combination of the color palette used as default by the ggplot2 package would be especially difficult for individuals with protanopia and deuteranopia to perceive with only a marginal improvement for individuals with tritanopic vision.

5 Utilities

In addition to the core functionality described above, the `brewscheme` package also provides a set of utilities and internals that other users may find helpful or useful. The utility commands can be thought of as commands related to the overall goal of the package and are intended for direct use by end users, while the internals that will be described later are used primarily by the commands described in this and the previous section but may have uses for other users.

brewcolors

A posting to the StataList from Wiggins (2004) prompted the development of the brewcolors package. In the post, Wiggins (2004) is responding to a query from Bill Rising in which he describes the structure of named color styles in Stata. Although there are many named colors already available in Stata, users — for one reason or another — may wish to define named color styles that can be more easily referenced by a name than the corresponding color space values. In addition to providing a tool to help facilitate the installation of named color styles, the `brewcolors` command also updates the database of named color styles that the `brewscheme` package uses to look up named color styles' RGB values and their corresponding RGB values for color sight impairment simulations.

`brewcolors` ***xkcd*** `|` ***new*** [`,` `make` `install` `colors(`*string*`)` `refresh`]

xkcd is an option used to construct a dataset containing the 900+ named colors from the 2010 XKCD survey Monroe (2010).

new is an option used to construct new named color styles based on user input.

`make` is an optional argument used to make — if the program is called prior to brewcolordb — and update the color database file with the additional colors.

`install` is an optional argument used to install the named colors to make them available

to Stata graph commands and in menus for creating graphs.

`colors` is an optional argument used in to pass a color constructor string (when the new syntax is used) or to provide a list of colors from the XKCD color survey Monroe (2010) which should be installed or added to the color database.

`refresh` is an option used to rebuild the color database.

Examples Like the `brewscheme` and `brewtheme` commands, the `brewcolors` command also automates the creation of parallel versions of the named colors for each of the forms of color sight impairment. To make all of the 900+ colors defined in the XKCD color survey (Monroe (2010)) available to the `brewscheme` package you would use:

▷ **Example**

```
brewcolors xkcd, ma
```

◁

After doing this, if you view any of the GUI menus for Stata graphs you'll notice that the colors are not available as named color styles. However, this has added these colors to the color database used/maintained by `brewscheme`. If you instead wanted to make the colors available as named color styles, pass the `install` parameter to the command:

▷ **Example**

```
brewcolors xkcd, ma inst
```

◁

If you wanted to define your own named color styles you can do that as well:

▷ **Example**

```
   brewcolors new, ma inst colors(‘‘‘117 255 47’’’)
brewcolors new, ma inst colors(‘‘‘‘‘mycolor’’ ‘‘117 255 47’’’’’)
```

◁

The two syntaxes above are equivalent. In the first example, the color would be named uc11725547, while the same color would be named "mycolor" in the second example. Users can specify multiple colors by wrapping each key/value pair (e.g., color name and RGB values) with compound double quotes. Internally, the program parses the colors argument by counting the words in the string and separating them into color names and RGB values.

brewextra

In addition to providing users with methods that can be used to add named color styles to their Stata installations, the `brewextra` command provides a mechanism to add data

to the color palette database. When called without options (which happens automatically the first time the `brewscheme` command is used), the command adds additional color palettes to the database containing the ColorBrewer (Brewer (2002)) palettes and adds the palettes defined in the D3js library Bostock et al. (2011), colors with semantic meanings Lin et al. (2013), and colors with socio-culturally defined meanings Buchanan (2014, 2015).

`brewextra` [`, files(`*string*`) refresh`]

`files(`*string*`)` is an option used to pass a string of file names containing the data to be added to the color palette database.

`refresh` an optional argument used to rebuild the database.

Examples Table 1 shows the file specification that must be followed to include a new palette in your palette database.

Table 1: File specification for `brewscheme` palettes

variable name	storage type	display format	value label	variable label
palette	str11	%11s		Name of Color Palette
colorblind	byte	%10.0g	colorblind	Colorblind Indicator
print	byte	%10.0g	print	Print Indicator
photocopy	byte	%10.0g	photocopy	Photocopy Indicator
lcd	byte	%10.0g	lcd	LCD/Laptop Indicator
colorid	byte	%10.0g		Within pcolor ID for individual color look ups
pcolor	byte	%10.0g		Palette by Colors Selected ID
rgb	str11	%11s		Red-Green-Blue Values to Build Scheme Files
maxcolors	byte	%10.0g		Maximum number of colors allowed for the palette
seqid	str13	%13s		Sequential ID for property lookups
meta	str13	%13s		Meta-Data Palette Characteristics

Using `viewsource brewextra.ado` can also help you to see how the data are constructed from text that constructs a file that is created by the command and used to add the additional palettes to the database internally.

brewmeta

An additional tool is available to look up the attributes of given color palettes, although it is primarily relevant to the colors palettes defined by Brewer (2002).

`brewmeta` *palette name*, `colorid(#)` [`colors(#)` `properties`[(‘‘’’, ‘‘all’’, ‘‘colorblind’’, ‘‘lcd’’, ‘‘print’’, ‘‘photocopy’’, ‘‘meta’’)] `refresh`]

`colorid` the specific color of which you are interested (e.g., color colorid of colors for a palette)

`colors` the total number of colors from which the colorid should be selected (e.g., if the palette has up to 12 colors and you were interested in color 5 when only 6 colors are used you would pass a value of 6 to colors and a value of 5 to colorid)

`properties` an optional argument to define the specific attributes/properties of the color/palette to look up.

Examples This command is used to quickly look up available attributes related to a given combination of colors, palettes, and specific color values within those color x palette definitions. For example, you can check whether or not the color has been shown to be robust to color sight impairments by passing the colorblind argument to the properties parameter:

▷ Example

```
brewmeta pastel2, colorid(5) colors(7) prop(colorblind)
```

◁

Would return a result of `The color 5 of palette pastel2 with 7 colors is Not color blind friendly` printed to the results window, as well as being returned in the macro `r(pastel275_colorblind)`. You can also access additional metadata attributes of the ColorBrewer palettes using the “meta” option passed to the properties parameter:

▷ Example

```
brewmeta dark2, colorid(3) prop(meta)
```

◁

Would return a result of `The color 3 of palette dark2 with 7 colors is Qualitative`. If instead, you wanted to know about all of the attributes defined for a color within a palette you can omit the properties parameter to retrieve all of the attributes:

▷ Example

```
brewmeta puor, colorid(6)
```

◁

Would print the following to the results window:
`The color 6 of palette puor with 10 colors is Missing Data on Colorblind Friendliness`

```
The color 6 of palette puor with 10 colors is LCD friendly
The color 6 of palette puor with 10 colors is Not photocopy friendly
The color 6 of palette puor with 10 colors is Possibly print friendly
The color 6 of palette puor with 10 colors is Divergent
```

brewcbsim

While the `brewproof` command is useful for proofing graphs defined by existing schemes, you may also want similar capabilities for individual colors. The `brewcbsim` command is useful for proofing an individual - or collection of individual - colors in a single graph. Because the `brewcolors` and `brewcolordb` commands create a database of named color styles, the `brewcbsim` command is able to accept either named color styles or RGB values.

`brewcbsim` *`RGB Strings | named color styles`*

Examples The `brewcbsim` command takes a single argument consisting of one or more named color styles and/or RGB strings.

brewsearch

`brewsearch` *`RGB String | named color style`*

brewtransform

`brewtransform` *varname*

brewviewer

`brewviewer` *`palette names`* [, colors(*`numlist`*) combine seq]

colors

combine

seq

hextorgb

`hextorgb,` hexcolor(*string*| *varname*)

hexcolor

5.1 Java Plugins

brewterpolate

brewterpolate , scolor(*string*) ecolor(*string*) colors(*#*) [,
luminance(*string*) icspace(*string*) rcspace(*string*) inverse]

scolor

ecolor

colors

luminance(*string*)

icspace(*string*)

rcspace(*string*)

inverse

filesys

filesys *filename* [, attributes display global readable(*string*)
writable(*string*) xecutable(*string*) readonly]

attributes

display

global

readable

writable

xecutable

readonly

6 Internals

The commands described in this section are designed primarily for calls made by other programs in the `brewscheme` package. They are included here for interested readers and to furhter document how the program works and functions.

6.1 Stata Internal Commands

brewlibcheck

`brewlibcheck`

brewdb

`brewdb [, refresh]`

`refresh`

dirfile

`dirfile, path(`*string*`) [, rebuild]`

`rebuild`

6.2 Mata Internals

Recycle

The `recycle` function is not defined in an `ado` file, but can be called from Stata using the syntax below.

```
mata:  recycle(real scalar shortVec, real scalar longVec)
```

The function takes two arguments, which contain the length of the shorter and longer vectors. From the example above, the call to recycle for the case of line graphs would be:

▷ **Example**

```
mata:recycle(3, 10)
```

◁

In this case, the function returns the value "1 2 3 1 2 3 1 2 3 1" in the local macro `sequence`. These values are treated as indices to select the appropriate RGB values to use for each of the 10 line color attributes.

libbrewscheme

`libbrewscheme [, display replace size(#) ]`

`display`

`replace`

`size(#)`

7 References

Anonymous. 2013. to all Stata graph makers. http://www.econjobrumors.com/topic/to-all-the-stata-graph-makers.

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

Bischof, D. 2015. Figure Schemes for Decent Stata Figures: blind & color-blind. http://danbischof.com/2015/02/04/stata-figure-schemes/. Article downloadable from https://www.dropbox.com/s/m5viis9oybgkept/FigureScheme.pdf?dl=0.

Bostock, M., V. Ogievetsky, and J. Heer. 2011. D3: data driven documents. *IEEE Transactions on Visualization & Computer Graphics* 17(12): 2301–2309. Retrieved from http://vis.stanford.edu/papers/d3.

Brettel, H., F. Viénot, and J. D. Mollon. 1997. Computerized simulation of color appearance for dichromats. *Journal of the Optical Society of America A* 14(10): 2647–2655.

Brewer, C. A. 2002. Color Brewer 2. [Computer Software]. State College, PA: Cynthia Brewer, Mark Harrower, and The Pennsylvania State University. Retrieved from http://www.ColorBrewer2.org.

Briatte, F. 2013. Plotting with the BuRd scheme. http://srqm.tumblr.com/post/44632966728/plotting-with-burd.

Buchanan, B. 2014. Using Stata for Educational Accountability & Compliance Reporting. Presentation to US Stata Users Group Meeting, Bostom, MA, 31 July. Downloadable from http://www.stata.com/meeting/boston14/abstracts/materials/boston14_buchanan.pdf.

———. 2015. Brewing color schemes in Stata: Making it easier for end users to customize Stata graphs. Presentation to US Stata Users Group Meeting, Columbus, OH, 31 July. Downloadable from http://www.stata.com/meeting/columbus15/abstracts/materials/columbus15_buchanan.pdf.

Cox, N. J. 2013. Strategy and Tactics for Graphic Multiples in Stata. Presentation to London Stata Users Group Meeting, London, England,. Downloadable from http://www.timberlake.co.uk/media/pdf/proceedings/materials/uk13_cox.ppt.

———. 2014. *Speaking Stata Graphics*. College Station, TX: Stata Press.

Crow, K. 2008. Stata tip 72: Using the Graph Recorder to create a pseudograph scheme. *The Stata Journal* 8(4): 592–593.

Hsiang, S. 2013. Prettier graphs with less headache: use schemes in Stata. http://www.fight-entropy.com/2013/01/prettier-graphs-with-less-headache-use.html.

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

Lin, S., J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. 2013. Selecting Semantically-Resonant Colors for Data Visualization. *Computer Graphics Forum* 32(3): 401–410. Retrieved from http://vis.stanford.edu/files/2013-SemanticColor-EuroVis.pdf.

Lindbloom, B. 2001. RGB working space information. Retrieved from: http://www.brucelindbloom.com/WorkingSpaceInfo.html. Retrieved on 24nov2015.

Mitchell, M. 2012. *A Visual Guide to Stata Graphics*. 3rd ed. College Station, TX: Stata Press.

Monroe, R. P. 2010. Color Survey Results. http://blog.xkcd.com/2010/05/03/color-survey-results/. Source data downloadable from http://xkcd.com/color/rgb.txt.

Pisati, M. 2007. SPMAP: Stata module to visualize spatial data. Statistical Software Components, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s456812.html.

Radyakin, S. 2009. Advanced Graphics Programming in Stata. Presentation to US Stata Users Group Meeting, Washington, D.C., 29 July. Downloadable from http://www.stata.com/meeting/dcconf09/dc09_radyakin.pdf.

Rising, B. 2010. Getting Graphs a Good Look: Schemes and the Graph Editor. Presentation to Portuguese Stata Users Group Meeting, Braga, Portugal, 17 September. Downloadable from http://www.stata.com/meeting/portugal10/portugal10_rising.pdf.

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

Viénot, F., H. Brettel, and J. D. Mollon. 1999. Digital Video Colourmaps for Checking the Legibility of Displays by Dichromats. *COLOR research and application* 24(4): 243–252.

Wickham, H. 2009. *ggplot2: Elegant Graphics for Data Analysis*. New York City, NY: Springer Science+Business Media LLC.

Wickline, M. 2014. Color.Vision.Simulate. Retrieved from: http://galacticmilk.com/labs/Color-Vision/Javascript/Color.Vision.Simulate.js. Retrieved on: 24nov2015. Version 0.1.

Wiggins, V. 2004. defining new colors or scheming too hard. Statalist Archives. http://www.stata.com/statalist/archive/2004-10/msg00209.html.

About the authors

William Buchanan is currently a data scientist in the Office of Research, Assessment, & Evaluation at the Minneapolis Public Schools District, following two years as a Strategic Data Fellow at the Mississippi Department of Education. in addition to working more directly in the field of education, he also worked as a methodological/statistical programming consultant following grad school.