The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–48

The brewscheme toolkit for Data Visualization in Stata

William R. Buchanan
Office of Research, Evaluation, & Assessment
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

© *yyyy* StataCorp LP

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).

Another difference between this program and existing programs is the documentation provided to end users for scheme entries. Unlike the official Stata documentation, the help files for `brewtheme` are organized based on the structure of the scheme file (e.g., all yes/no entries are contained in a single page of the brewtheme documentation). Additionally, some scheme file entries are not explicitly documented in the official documentation from StataCorp (e.g., clegend* entries), most likely due to a combination of demand for the entries to be documented and use of scheme file documentation by the broader community. When official documentation is available for scheme entries, the `brewtheme` documentation provides explicit and direct references to the relevant entries; if official documentation is unavailable, `brewtheme` provides direct references to the location in the `s2color` scheme file.

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 development branch of 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 programs will check when the Mata library `libbrewscheme` was created/modified and will compile itself if needed. On first installation, it would still be useful to use the convenience wrapper to compile the library locally on your system and to build the color dataset containing the Stata named color styles RGB values along with versions of each color that represent how they might be perceieved by individuals with achromatopsia, protanopia, deuteranopia, and/or tritanopia (color sight impairments):

```
libbrewscheme, replace
```

```
brewcolordb, replace override
```

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 the JavaScript source at 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` `override`]

`display` is an optional argument to display the color information in the results window during the program's execution.

`refresh` is an optional argument used to overwrite an existing copy of the library.

`override` is an optional argument used to override a user prompt before clearing data from memory.

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:

$\texttt{optionname("key}_1\ \texttt{value}_1\texttt{" "key}_2\ \texttt{value}_2\texttt{" "..." "key}_n\ \texttt{value}_n\texttt{")}$

brewtheme API

`brewtheme` *theme name* [, `abovebelow(`*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: clegend, horizontal_tick, p, parrow, parrowbarb, and vertical_tick. See [G-4] ***anglestyle*** or use `graph query anglestyle` for additional information.

`areastyle` an optional argument with the following keys: background, bar_iplotregion, bar_plotregion, box_iplotregion, box_plotregion, bygraph, bygraph_iplotregion, bygraph_plotregion, ci, ci2, clegend, clegend_inner, clegend_inpreg, clegend_outer, clegend_preg, combine_iplotregion, combine_plotregion, combinegraph, combinegraph_inner, dendrogram, dot_iplotregion, dot_plotregion, dotchart, foreground, graph, hbar_iplotregion, hbar_plotregion, hbox_iplotregion, hbox_plotregion, histogram, inner_bygraph, inner_graph, inner_legend, inner_piegraph, inner_pieregion, inner_plotregion, legend, legend_inkey_region, legend_key_region, matrix_ilabel, matrix_iplotregion, matrix_label, matrix_plotregion, matrixgraph_iplotregion, matrixgraph_plotregion, piegraph, piegraph_region, plotregion, sunflower, sunflowerdb, sunflowerlb, twoway_iplotregion, and twoway_plotregion. 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: bar_group, bar_scale_horiz, bar_scale_vert, bar_super, bar_var, box_scale_horiz, box_scale_vert, clegend (line 1394)*, dot_group, dot_scale_horiz, dot_scale_vert, dot_super, dot_var, horizontal_default, horizontal_nogrid, matrix_horiz, matrix_vert, sts_risktable (line 1393)*, vertical_default, and vertical_nogrid. 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: box, default, and dot.

`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: by_legend_position, caption_position, clegend_title_position, ilabel, legend_caption_position, legend_note_position, legend_position, legend_subtitle_position, legend_title_position, matrix_marklbl, note_position, p, subtitle_position, title_position, and zyx2legend_position. See [G-4] ***clockposstyle*** or use `graph query clockpos` for additional information.

`color` an optional argument with the following keys: axis_title, axisline, background, backsymbol, body, box, bylabel_outline, clegend, clegend_inner, clegend_line, clegend_outer, filled, filled_text, foreground, grid, heading, histback, key_label, label, legend, legend_line, major_grid, mat_label_box, matplotregion_line, matrix, matrix_label, matrix_marklbl, matrix_plotregion, matrixmarkline, minor_grid, minortick, pboxlabelfill, plabelfill, plotregion, plotregion_line, pmarkback, pmarkbkfill, reverse_big, reverse_big_line, reverse_big_text, small_body, sts_risk_label, sts_risk_title, subheading, symbol, text, text_option, text_option_fill, text_option_line, textbox, tick, tick_biglabel, and tick_label. See [G-4] ***colorstyle*** or use `graph query color` for additional information.

`compass2dir` an optional argument with the following keys: editor, graph_aspect, key_label, legend_fillpos, legend_key, p, and text_option. See [G-4] ***compassdirstyle*** or use `graph query compassdir` for additional information.

`compass3dir` an optional argument with a single key: p. See `graph query compass3dirstyle` for additional information.

`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. See `graph query dottypestyle` for additional information.

`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 matrixgraph. See `graph query graphstyle` for additional information.

`gridlinestyle` an optional argument with a single key: default. See `graph query gridlinestyle` for additional information.

`gridringstyle` an optional argument with the following keys: by_legend_ring, caption_ring, clegend_ring, clegend_title_ring, legend_caption_ring, legend_note_ring, legend_ring, legend_subtitle_ring, legend_title_ring, note_ring, spacers_ring, subtitle_ring, title_ring, and zyx2legend_ring. See [G-4] ***ringposstyle*** for additional information.

`gridstyle` an optional argument with the following keys: major and minor. See `graph query gridstyle` for additional information.

`gsize` an optional argument with the following keys: alternate_gap, axis_space, axis_title, axis_title_gap, barlabel_gap, body, clegend_height, clegend_width, dot_rectangle, filled_text, gap, heading, key_gap, key_label, key_linespace, label, label_gap, legend_col_gap, legend_colgap, legend_key_gap, legend_key_xsize, legend_key_ysize, legend_row_gap, matrix_label, matrix_marklbl, matrix_mlblgap, minortick, minortick_label, note, notickgap, pboxlabel, pie_explode, pielabel_gap, plabel, reverse_big, small_body, small_label, star, star_gap, sts_risk_label, sts_risk_tick, sts_risk_title, sts_risktable_lgap, sts_risktable_space, sts_risktable_tgap, subheading, text, text_option, tick, tick_biglabel, tick_label, tickgap, title_gap, zyx2colgap, zyx2legend_key_gap, zyx2legend_key_xsize, zyx2legend_key_ysize, and zyx2rowgap. See [G-4] ***textsizestyle*** or use `graph query textsizestyle` for additional information.

`horizontal` an optional argument with the following keys: axis_title, body, editor, filled, heading, key_label, label, matrix_label, small_body, sts_risk_label, sts_risk_title, subheading, and text_option. See [G-4] ***justificationstyle*** or use `graph query justificationstyle` for additional information.

`labelstyle` an optional argument with the following keys: editor, ilabel, matrix, and sunflower. See [G-4] ***labelstyle*** or use `graph query labelstyle` for additional information.

`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: axisline, background, ci, ci_area, clegend, dendrogram, dot, dot_area, dotmark, dots, foreground, grid, histogram, legend, major_grid, matrix_plotregion, matrixmark, minor_grid, minortick, p, pie, plotregion, pmark, refline, refmarker, sunflower, text_option, tick, xyline, and zyx2. See [G-4] ***linepatternstyle*** or use `graph query linepatternstyle` for additional information.

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

`margin` an optional argument with the following keys: axis_title, bargraph, body, boxgraph, by_indiv, bygraph, cleg_title, clegend, clegend_boxmargin, combine_region, combinegraph, dotgraph, editor, filled_box, filled_textbox, graph, hbargraph, hboxgraph, hdotgraph, heading, key_label, label, legend, legend_boxmargin, legend_key_region, mat_label_box, matrix_label, matrix_plotreg, matrixgraph, pboxlabel, pboxlabelbox, piegraph, piegraph_region, plabel, plabelbox, plotregion, small_body, star, subheading, text, text_option, textbox, and twoway. See [G-4] ***marginstyle*** or use `graph query marginstyle` for additional information.

`medtypestyle` an optional argument with a single key: boxplot. See `graph query medtypestyle` for additional information.

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

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

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

`pielabelstyle` an optional argument with a single key: default. See `graph query pielabelstyle` for additional information.

`plotregionstyle` an optional argument with the following keys: bargraph, boxgraph, bygraph, clegend, combinegraph, combineregion, dotgraph, hbargraph, hboxgraph, legend_key_region, matrix, matrix_label, matrixgraph, piegraph, and twoway. See [G-4] ***plotregionstyle*** or use `graph query plotregionstyle` for additional information.

`relativepos` an optional argument with the following keys: clegend_axispos, clegend_pos, and zyx2legend_pos. See `graph query relative_posn` for additional information.

`relsize` an optional argument with the following keys: bar_gap, bar_groupgap, bar_outergap, bar_supgroupgap, box_fence, box_fencecap, box_gap, box_groupgap, box_outergap, box_supgroupgap, dot_gap, dot_groupgap, dot_outergap, and dot_supgroupgap.

`special` an optional argument with the following keys: by_knot1, by_slope1, by_slope2, combine_knot1, combine_slope1, combine_slope2, default_knot1, default_slope1, default_slope2, matrix_knot1, matrix_slope1, matrix_slope2, matrix_xaxis, and matrix_yaxis. See [G-4] ***axis_options*** or view the help file for `scheme_by_scaling` for additional information.

`starstyle` an optional argument with a single key: default. See `graph query starstyle` for additional information.

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

`symbol` is an optional argument with the following keys: ci, ci2, dots, histback, histogram, ilabel, matrix, none, p, pback, pbarback, pdotback, refmarker, and sunflower. See [G-4] ***symbolstyle*** or use `graph query symbolstyle` for additional information.

`symbolsize` an optional argument with the following keys: backsymbol, backsymspace, ci, ci2, dots, histback, histogram, matrix, p, parrow, parrowbarb, pback, refmarker, smallsymbol, star, sunflower, and symbol. See [G-4] ***markersizestyle*** or use `graph query markersizestyle` for additional information.

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

`tickposition` an optional argument with a single key: axis_tick.

`ticksetstyle` an optional argument with the following keys: major_clegend, major_horiz_default, major_horiz_nolabel, major_horiz_notick, major_horiz_notickbig, major_horiz_withgrid, major_vert_default, major_vert_nolabel, major_vert_notick, major_vert_notickbig, major_vert_withgrid, minor_horiz_default, minor_horiz_nolabel, minor_horiz_notick, minor_vert_default, minor_vert_nolabel, minor_vert_notick, and sts_risktable. See [G-4] ***ticksetstyle*** or use `graph query ticksetstyle` for additional information.

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

`verticaltext` an optional argument with the following keys: axis_title, body, filled, heading, key_label, label, legend, matrix_label, small_body, subheading, and text_option. See [G-4] ***alignmentstyle*** or use `graph query alignmentstyle` for additional information.

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

`zyx2rule` an optional argument with a single key: contour. See `graph query zyx2rulestyle` for additional information.

`zyx2style` an optional argument with a single key: default. See `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

The two following examples illustrate how a .theme file could be constructed to simulate the aesthetics of the ggplot2 Wickham (2009) in Stata as well as a .theme file that emulates the aesthetics of the s2color scheme.

▷ Example

```
. do `"$articledir/brewthemeExamples.do"'
. /* brewtheme example theme files */
.
. // Change the end of line delimiter
. #d ;
delimiter now ;
. // Generate the theme file used to simulate ggplot2 aesthetics
> brewtheme ggtheme, 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 minuscule"
> "minortick tiny") numsty("legend_cols 1" "legend_rows 0" "zyx2rows 0"
> "zyx2cols 1") verticaltext("legend top");
Directory exists and rebuild option not specified.  No further action
. // Generates a theme in the style of s2color
> brewtheme s2theme, graphsi("x 5.5" "y 4") numsty("legend_cols 2" "legend_rows 0"
> "zyx2rows 0" "zyx2cols 1") gsize("label medsmall" "small_label small"
> "text medium" "body medsmall" "small_body small" "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" "minortick half_tiny"
> "pie_explode medium") relsize("bar_groupgap 67pct" "dot_supgroupgap 67pct"
> "box_gap 33pct" "box_supgroupgap 200pct" "box_outergap 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");
Directory exists and rebuild option not specified.  No further action

.
end of do-file
```

◁

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

The following examples illustrate the creation of scheme files that use a single color palette for all graphs, a combination of a default color palette and graph specific color palettes, and specifying palettes for each type of graph.

▷ Example

```
. do `"$articledir/brewschemeExamples.do"´

. /* brewscheme examples */
.
. // Create a mono color scheme with three colors; this will cause all layers
. // beyond the third to not be drawn (e.g., there won´t be colors defined for
. // Stata to use to assign colors to points/lines, etc...)
. brewscheme, scheme(onecolorex1) allsty(ggplot2)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // Use the ggplot2 color palette with s2color theme settings; this uses 4
. // colors to help highlight how these cases are handled by Stata
. brewscheme, scheme(onecolorex2) allsty(ggplot2) allc(4) ///
> themef(s2theme)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // Now five colors from same palette using the ggplot2
. // inspired theme
. brewscheme, scheme(ggplot2ex1) allsty(ggplot2) allc(5)  ///
> themef(ggtheme)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // An Example showing the use of the some parameters
. brewscheme, scheme(somecolorex1) somest(ggplot2)                ///
> somec(7) linest(dark2) linec(3) cist(pastel2) cic(6)    ///
> scatsty(category10) scatc(10)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
```

```
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // An example showing a different color palette/number
. // of colors for each graph type
. brewscheme, scheme(manycolorex1) barst(paired) barc(12) ///
> dotst(prgn) dotc(7) scatstyle(set1) scatc(8)                    ///
> linest(pastel2) linec(7) boxstyle(accent) boxc(4)           ///
> areast(dark2) areac(5) piest(mdepoint) sunst(greys)     ///
> histst(veggiese) cist(activitiesa) matst(spectral)              ///
> reflst(purd) refmst(set3) const(ylgn) cone(puor)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // Using different numbers of colors from the same scheme
. // to highlight differences and showing the use of the
. // symbols parameter
. brewscheme, scheme(ggplot2ex2) const(orange) cone(blue) ///
> consat(20) scatst(ggplot2) scatc(5) piest(ggplot2)              ///
> piec(6) barst(ggplot2) barc(2) linest(ggplot2) linec(2) ///
> areast(ggplot2) areac(5) somest(ggplot2) somec(15)              ///
> cist(ggplot2) cic(3) themef(ggtheme)                                    ///
> symbols(diamond triangle square)
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

For bugs/issues, please submit issues to:
http://github.com/wbuchanan/brewscheme
For additional information about the program visit:
http://wbuchanan.github.io/brewscheme

.
. // Load the auto.dta dataset
. sysuse auto.dta, clear
(1978 Automobile Data)

.
. // Loop over the schemes
. foreach scheme in onecolorex1 onecolorex2 ggplot2ex1    ///
> somecolorex1 manycolorex1 ggplot2ex2 {
  2.
.         // Create the same graph with each of the different schemes
.         tw      fpfitci 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,  scheme(`scheme´)   ///
>                 legend(order(1 "Frac Poly" 2 "Frac Poly" 3 "1" 4 "2"  ///
>                 5 "3" 6 "4" 7 "5")) name(`scheme´, replace)
  3.
.         // Export to an eps file
.         qui: gr export `"$articledir/brewscheme_`scheme´.eps"´,   ///
>         as(eps) replace
  4.
. }  // End of Loop over scheme files
.
end of do-file
```

◁

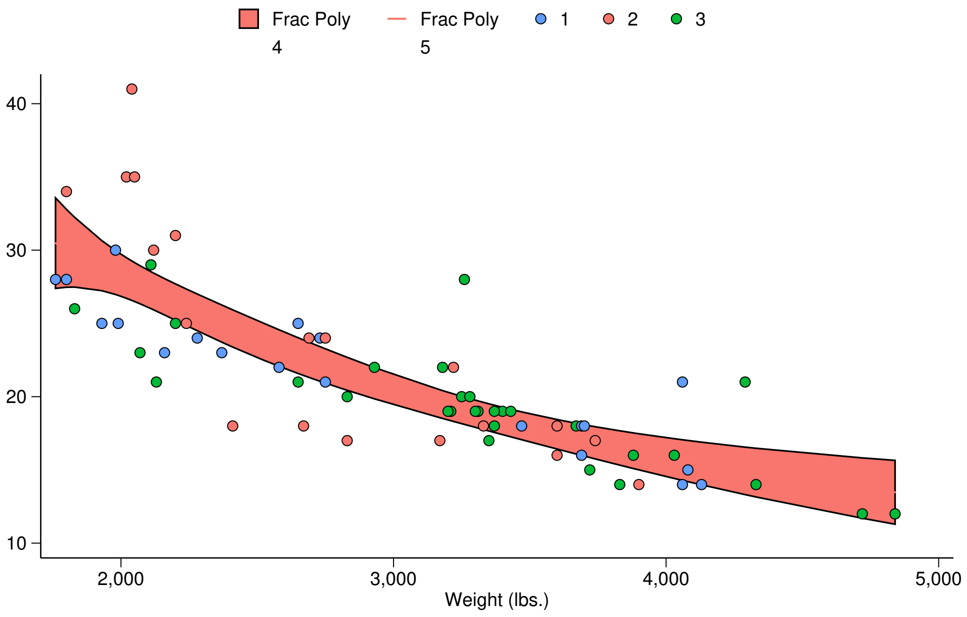

Figure 1: `brewscheme` graph with default `.theme` file and single color palette for all graphs

These examples also highlight a change 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.

Additionally, unlike the version discussed in Buchanan (2015), the current version of `brewscheme` uses `.theme` files to encapsulate and modularize the creation of the `.scheme` files. The primary difference between the first three examples exists in their respective `.theme` files that establish parameters that tend to be independent of specific graph

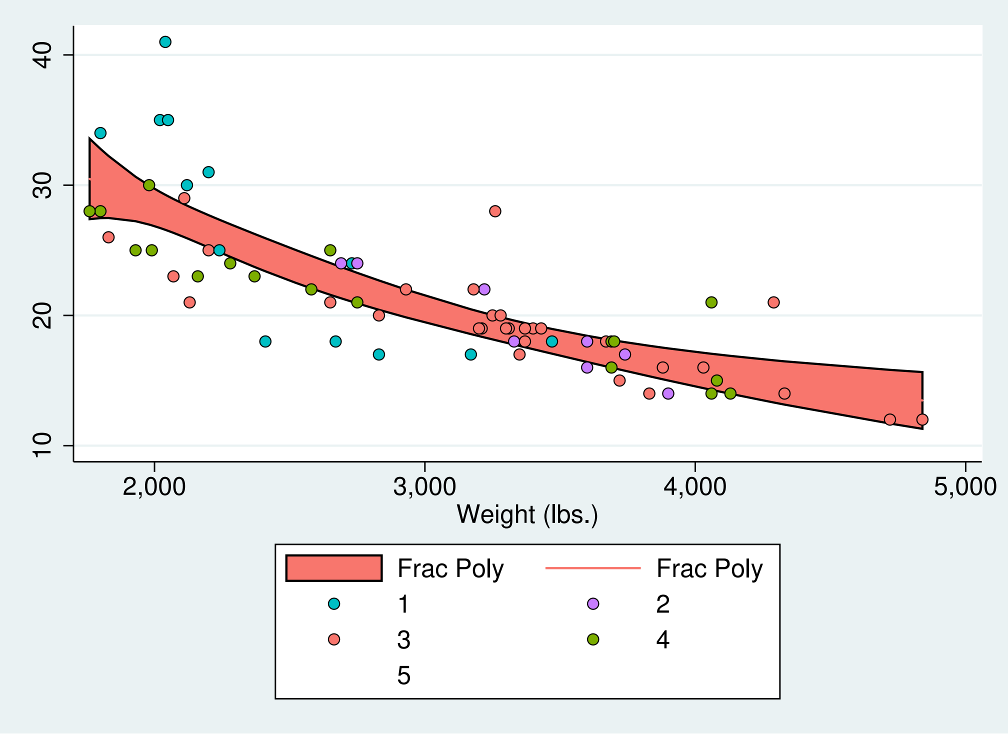

Figure 2: `brewscheme` graph with `s2color` inpired `.theme` file and single color palette for all graphs

types (e.g., displaying horizontal reference lines in the plot area, graph region fill colors, etc...). The major advantage to this development is the flexibility it provides to set generic parameters that can be reused across multiple `.scheme` files that may specify different configurations of color palettes for graphs.

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.

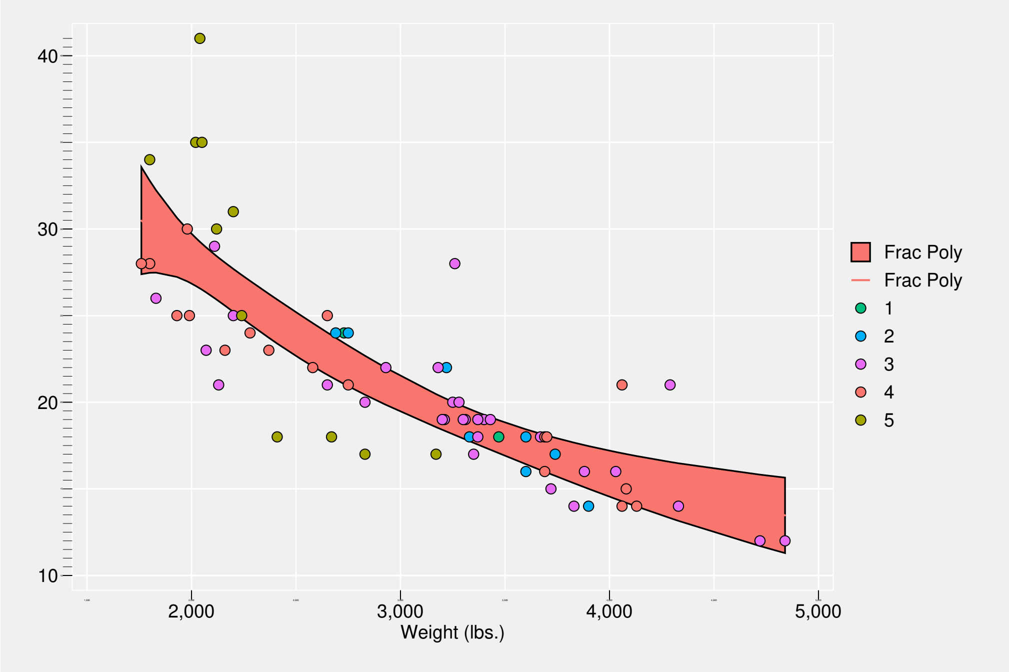

Figure 3: `brewscheme` graph with default `.theme` file and single color palette for all graphs

`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.

```
. do `"$articledir/brewproofExamples.do"´
. /* brewproof examples based on the brewscheme examples graphs */
.
. // Load the auto.dta dataset
. sysuse auto.dta, clear
(1978 Automobile Data)

.
. // Loop over the schemes
. foreach scheme in onecolorex1 onecolorex2 ggplot2ex1              ///
> somecolorex1 manycolorex1 ggplot2ex2 {
```

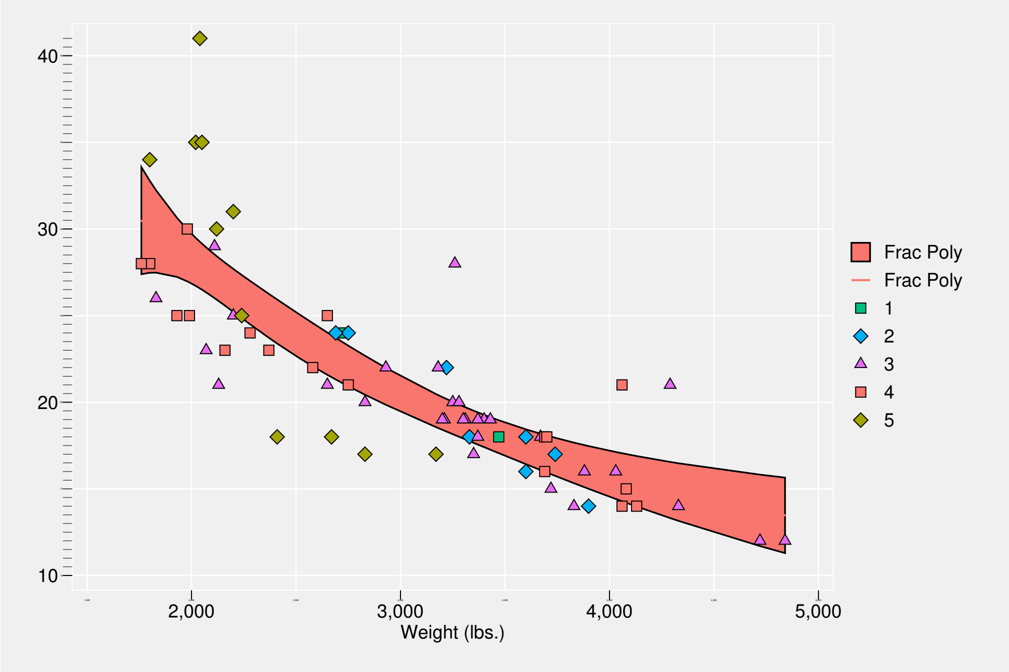

Figure 4: `brewscheme` graph with default `.theme` file and single color palette for all graphs

```
  2.
.         // Create the same graph with each of the different schemes
.         brewproof, scheme(`scheme´) : tw fpfitci 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(1 "Frac Poly" 2 "Frac Poly" 3 "1" 4 "2"  ///
>                 5 "3" 6 "4" 7 "5")) name(`scheme´, replace)
  3.
.         // Export to an eps file
.         qui: gr export `"$articledir/brewProof_`scheme´.eps"´,    ///
>         as(eps) replace
  4.
. } // End of Loop over scheme files
  1
  1
  1
  1
  1
  1

.
end of do-file
```

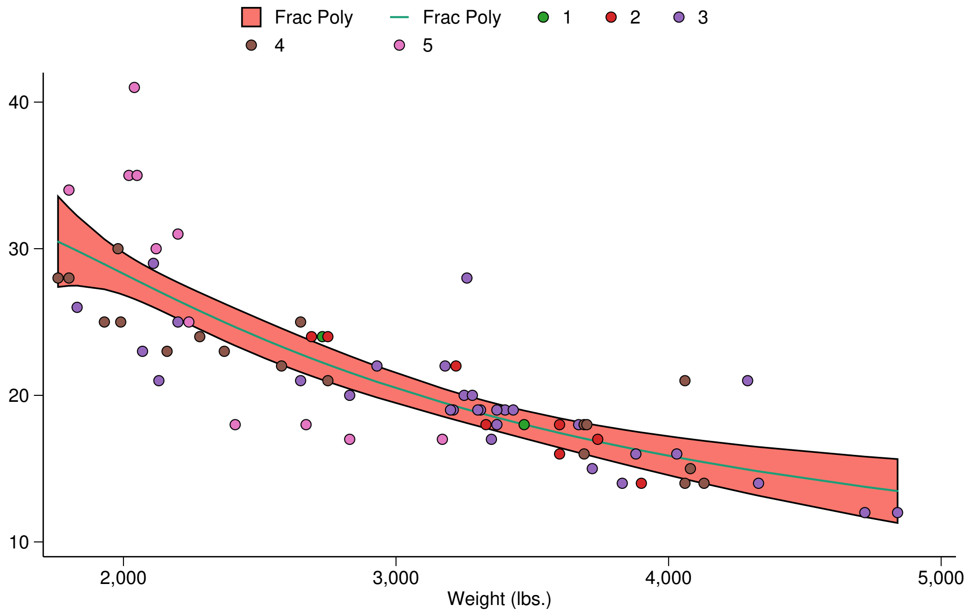

Figure 5: `brewscheme` graph with default `.theme` file, default color palette, and palettes specified for some graphs

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.

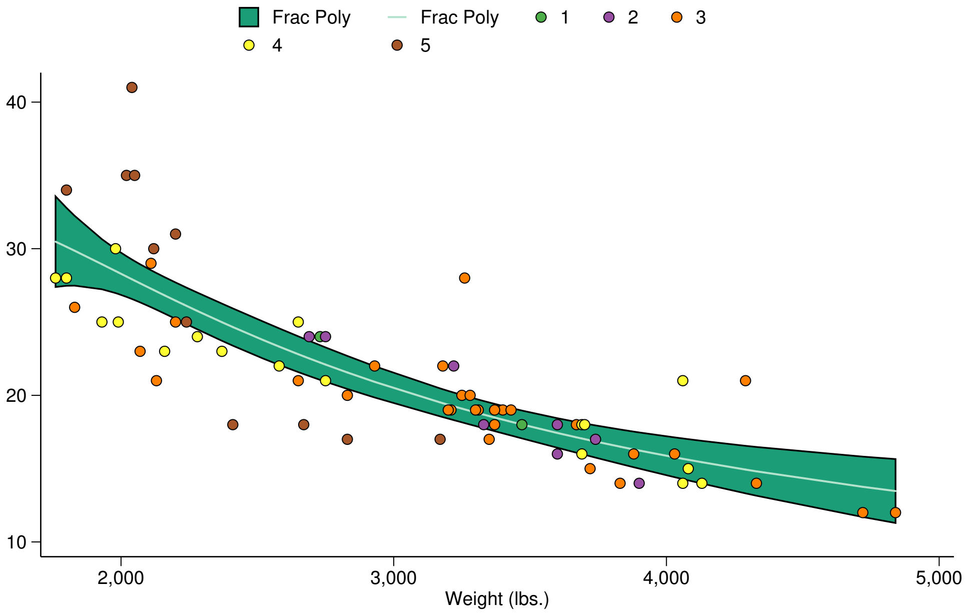

Figure 6: `brewscheme` graph with default `.theme` file and palettes specified for all graphs

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.

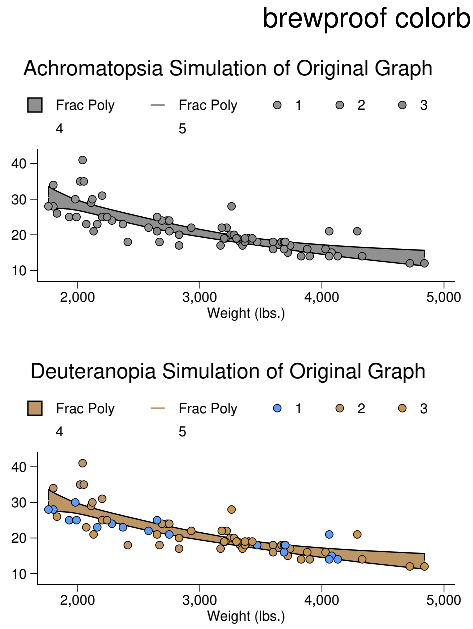

Figure 7: `brewproof` graph based on figure 1

`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. The first example below shows how the Monroe (2010) named colors can be installed to the local color database. This, however, does not expose these colors as named color styles in Stata[1]. To do that, you must also specify the install option, which writes the color style file and places it along the `ADOPATH`.

▷ Example

```
. do `"$articledir/brewcolorsExamples.do"´
```

[1] A screen shot showing these colors installed on the developer's system can be viewed at: http://wbuchanan.github.io/brewscheme/about.html for those interested

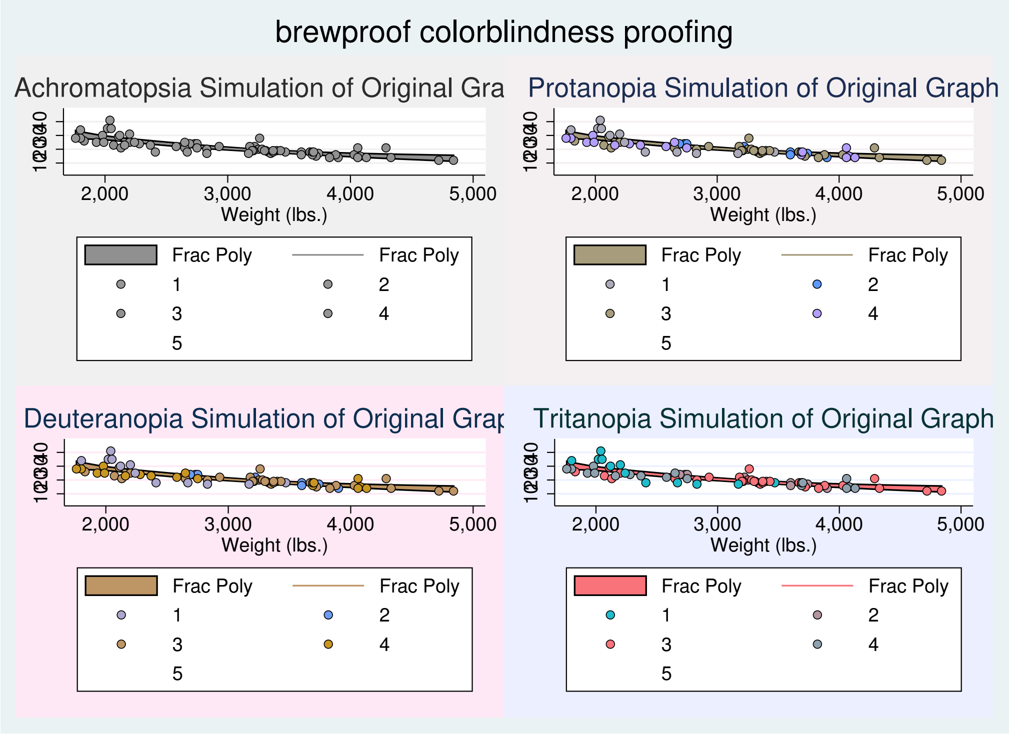

Figure 8: `brewproof` graph based on figure 2

```
. /* brewcolors examples */
.
. // Make the color database for the XKCD colors
. brewcolors xkcd, ma
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action
(2 vars, 950 obs)

.
. // Make the color database for the XKCD colors and install the named color styles
. brewcolors xkcd, ma inst
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action
(2 vars, 950 obs)

.
. // Add a new color to the color database
. brewcolors new, ma inst colors("117 200 47")
Directory exists and rebuild option not specified.  No further action
Directory exists and rebuild option not specified.  No further action

.
. // Add the same color but use the name mycolor
. brewcolors new, ma inst colors(`""mycolor 117 200 47""´)
Directory exists and rebuild option not specified.  No further action
```

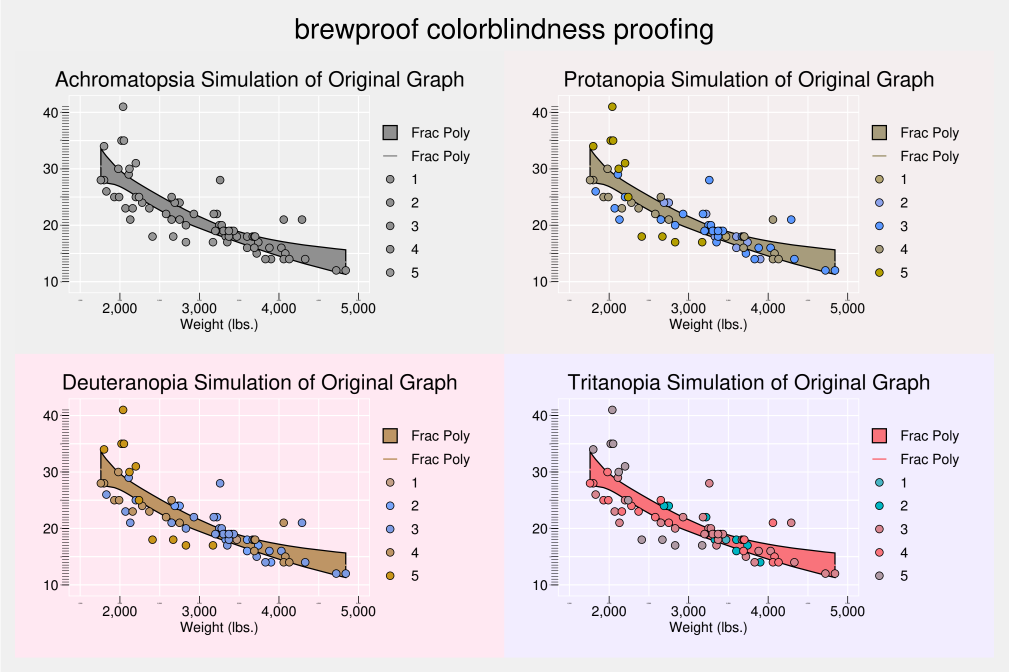

Figure 9: `brewproof` graph based on figure 3

```
Directory exists and rebuild option not specified.  No further action
.
end of do-file
```

◁

This program is also designed to help users define their own named color styles with their RGB values; this functionality, in particular, can be extremely valuable when a project requires data visualizations to reenforce branding through a common company color palette. The last two syntaxes in the examples above are equivalent. In the former, the color would be named uc11720047, while the same color would be named ”mycolor” in the later. Users can specify multiple colors by wrapping each key/value pair (e.g., color name and RGB values) with compound double quotes.

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

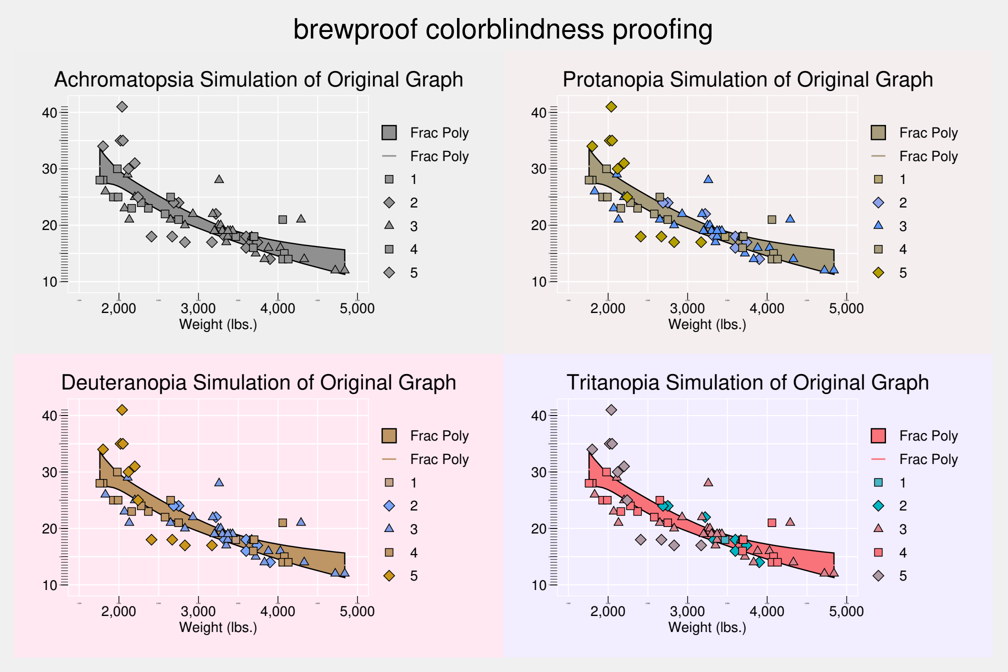

Figure 10: `brewproof` graph based on figure 4

meanings Lin et al. (2013), and colors with socio-culturally defined meanings Buchanan (2014, 2015).

`brewextra` [, `files(`*string*`)` <u>`ref`</u>`resh`]

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

<u>`ref`</u>`resh` 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.

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).

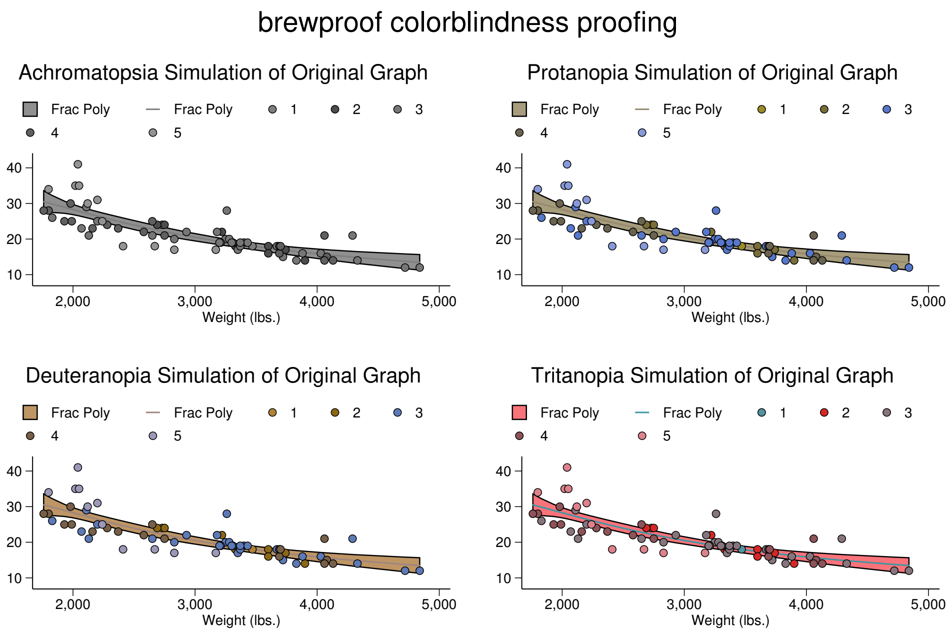

Figure 11: `brewproof` graph based on figure 5

`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.

Macros

`r(`*palette##_colorblind*`)`	Colorblind	Friendliness
`r(`*palette##_lcd*`)`	LCD	Friendliness
`r(`*palette##_photocopy*`)`	Photocopy	Friendliness
`r(`*palette##_print*`)`	Print	Friendliness
`r(`*palette##_meta*`)`	Additional	Characteristics

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

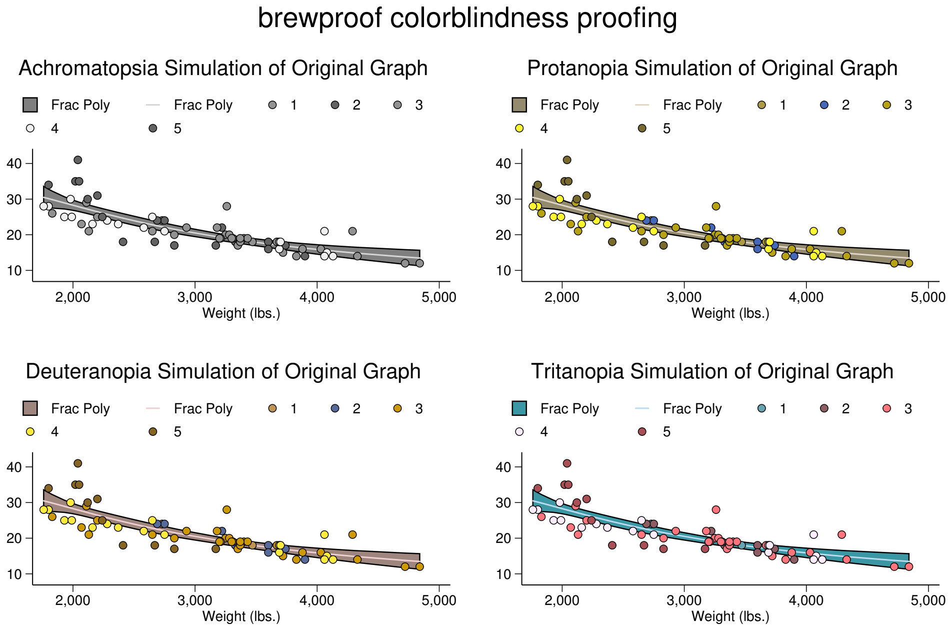

Figure 12: `brewproof` graph based on figure 6

palette definitions.

▷ Example

```
. do `"$articledir/brewmetaExamples.do"´

. /* brewmeta examples */
.
. // Get the color blind attribute for the pastel2 palette with 7 colors for color
. // number 5
. brewmeta pastel2, colorid(5) colors(7) prop(colorblind)
The color 5 of palette pastel2 with 7 colors is Not color blind friendly

.
. // Get the meta attribute for the dark2 palette with maximum number of colors for
. // the third color
. brewmeta dark2, colorid(3) prop(meta)
The color 3 of palette dark2 with 7 colors is Qualitative

.
. // Get all of the attributes for the puor palette with the maximum number of
. // colors for the 6th color
. brewmeta puor, colorid(6)
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
```

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

```
The color 6 of palette puor with 10 colors is Divergent
.
.
end of do-file
```

◁

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`*

Macros

`r(`*`original#`*`)`	RGB	Value
`r(`*`achromatopsic#`*`)`	Achromatopsia	Simulated
`r(`*`protanopic#`*`)`	Protanopia	Simulated
`r(`*`deuteranopic#`*`)`	Deuteranopia	Simulated
`r(`*`tritanopic#`*`)`	Tritanopia	Simulated

Examples The `brewcbsim` command takes a single argument consisting of one or more named color styles and/or RGB strings. The example below shows how the program can be used with user specified colors, a Stata named color style, and a named color style installed by the `brewcolors` command.

▷ Example

```
. do `"$articledir/brewcbsimExamples.do"´
. /* brewcbsim examples */
.
. // Simulation with XKCD installed color, RGB strings, and a Stata named color style
. brewcbsim xkcd119 "63 210 142" "8 151 233" "182 33 43" bluishgray8
.
. qui: gr export `"$articledir/brewcbsimEx1.eps"´, as(eps) replace
.
. // Colors typically associated with color sight impairments
. brewcbsim red green blue yellow
.
. qui: gr export `"$articledir/brewcbsimEx2.eps"´, as(eps) replace
.
end of do-file
```

◁

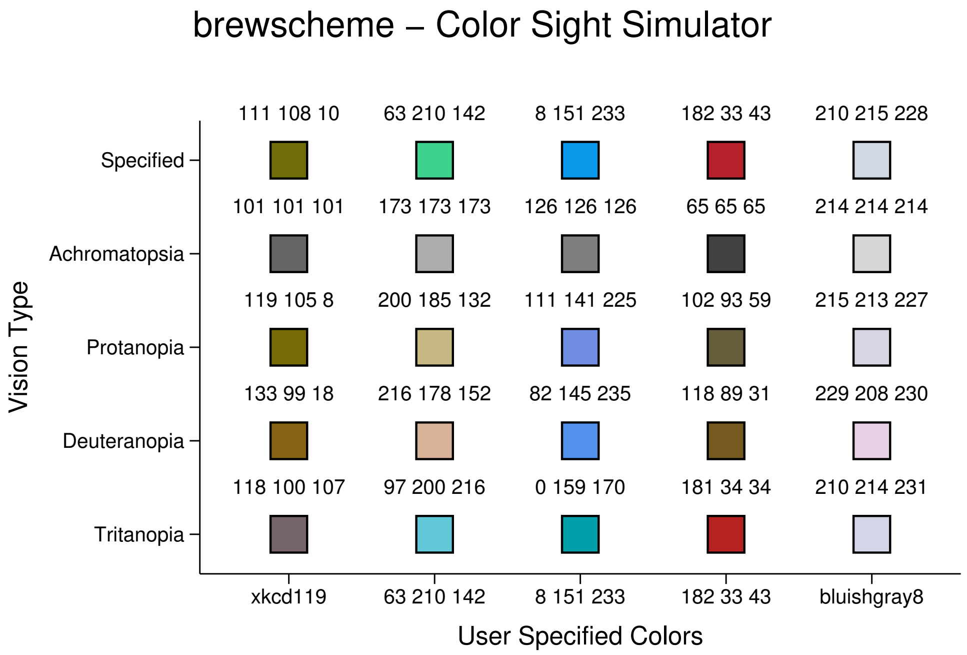

Figure 13: `brewcbsim` graph with combination of named color styles and RGB values passed as arguments

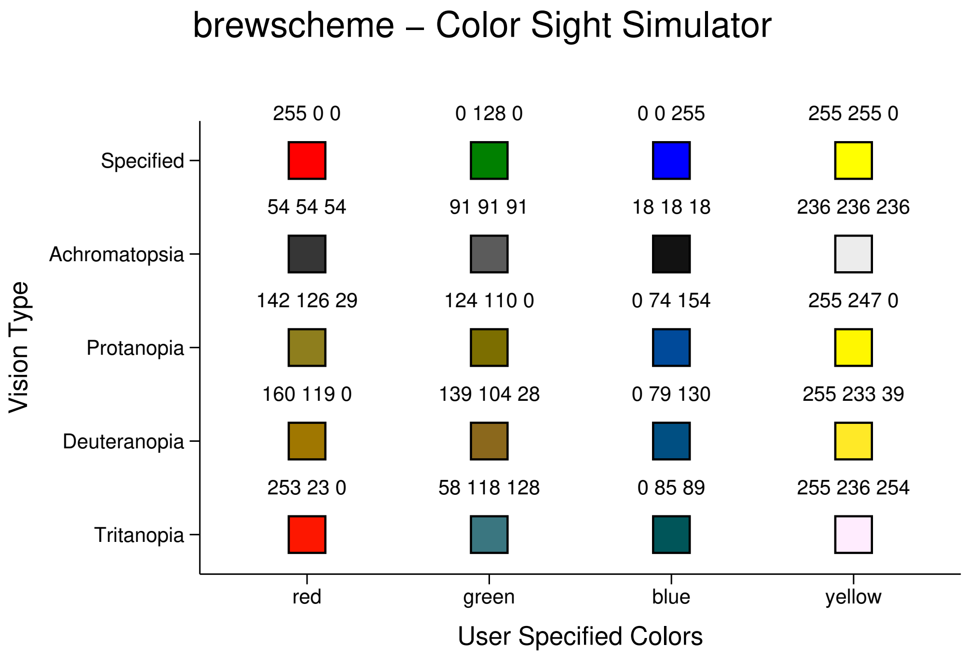

Figure 14: `brewcbsim` graph with colors typically associated with color sight impairments

brewviewer

The `brewviewer` command provides a previewer for the palettes made available by `brewscheme`. In addition to the basic previewer capabilities, the program also allows users to view copies of the palette(s) that are transformed to simulate the different forms of color sight impairments.

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

`colors` the number of colors to display from a given palette or the maximum number of colors to show if the sequential option is used.

`combine` an option to combine graphs for separate palettes into a single graph.

`seq` an option used to treat the values passed to the colors parameter as the maximum number of colors to display from the palette (e.g., a value of 6 will display the palette with 3, 4, 5, and 6 colors). Without this option, the values passed to the colors command are treated as discrete values (e.g., a value of 6 will display a single set of colors for a palette with 6 colors).

impaired is an option used include the color sight impaired simulated colors in the preview.

ExamplesExample

```
. do `"$articledir/brewviewerExamples.do"´
. /* brewviewer examples */
.
. // Use the D3js palette with upto 6 colors (e.g., 3, 4, 5, and 6) and include
. // how the colors would appear with different forms of color sight impairments
. brewviewer category10, im seq c(6)
.
. qui: gr export `"$articledir/brewviewerEx1.eps"´, as(eps) replace
.
. // Specify a different number of colors for each palette graphing the colors with
. // the sequential option and combining the results into a single image
. brewviewer category10 category20 category20b category20c, c(5 8 10 12)  comb seq
.
. qui: gr export `"$articledir/brewviewerEx2.eps"´, as(eps) replace
.
. // Use the same number of colors for multiple palettes and combine the results
. brewviewer dark2 mdebar accent pastel2 set1 tableau, c(5) seq comb
.
. qui: gr export `"$articledir/brewviewerEx3.eps"´, as(eps) replace
.
. // Show the same portion of each palette listed in a combined graph
. brewviewer dark2 mdebar accent pastel2 set1 tableau, c(5) comb
.
. qui: gr export `"$articledir/brewviewerEx4.eps"´, as(eps) replace
.
.
end of do-file
```

◁

hextorgb

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

hexcolor

Macros

r(*red#*)	Red Channel Value	r(*green#*)	Green Channel Value
r(*blue#*)	Blue Channel Value	r(*rgb#*)	Stata RGB String
r(*rgbcomma#*)	Comma-Delimited RGB String		

Examples The examples below show how the hextorgb command was used to convert the color palettes used by the D3js (Bostock et al. (2011)) library to RGB values used by the brewextra command to add those color palettes to the brewscheme package.

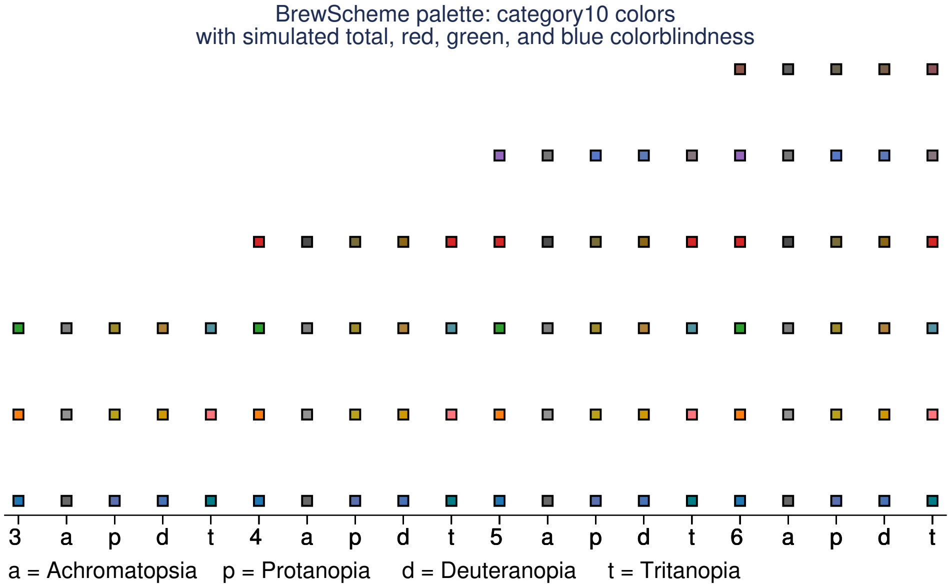

Figure 15: `brewviewer` example with single palette, single sequential color, and color sight impairment simulated values

▷ Example

```
. do `"$articledir/hextorgbExamples.do"´

. /* hextorgb examples */
.
. // Using the first three colors from the category10 palette from D3js
. hextorgb, hex("#1f77b4" "#ff7f0e" "#2ca02c")
--------------------------------------------------------------------------------
          Red        Green      Blue       RGB                 RGB String
--------------------------------------------------------------------------------
          31         119        180        31, 119, 180        "31 119 180"
          255        127        14         255, 127, 14        "255 127 14"
          44         160        44         44, 160, 44         "44 160 44"
--------------------------------------------------------------------------------

.
. // Display the returned results
. ret li

macros:
          r(rgbcomma3) : "44, 160, 44"
               r(rgb3) : "44 160 44"
              r(blue3) : "44"
             r(green3) : "160"
               r(red3) : "44"
          r(rgbcomma2) : "255, 127, 14"
```

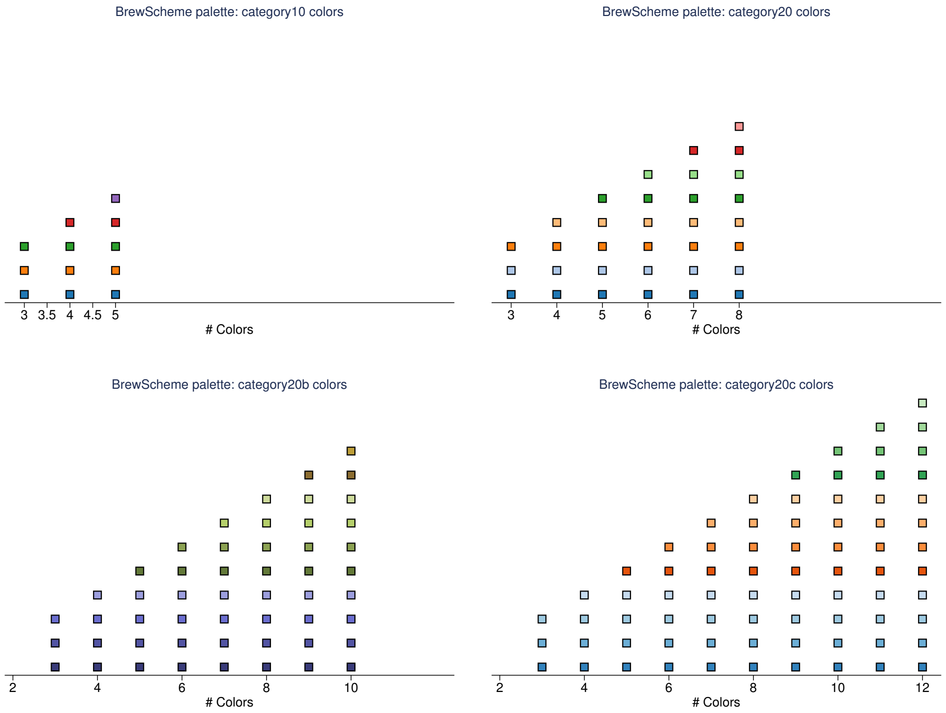

Figure 16: `brewviewer` example with multiple palettes and multiple sequential colors

```
               r(rgb2) : "255 127 14"
              r(blue2) : "14"
             r(green2) : "127"
               r(red2) : "255"
          r(rgbcomma1) : "31, 119, 180"
               r(rgb1) : "31 119 180"
              r(blue1) : "180"
             r(green1) : "119"
               r(red1) : "31"

.
. // Can also pass a longer list of values to convert (e.g., the entire category10
. // palette).
. hextorgb, hex("#1f77b4" "#ff7f0e" "#2ca02c" "#d62728" "#9467bd" "#8c564b"        ///
> "#e377c2" "#7f7f7f" "#bcbd22" "#17becf")
--------------------------------------------------------------------------------
          Red       Green     Blue      RGB                 RGB String
--------------------------------------------------------------------------------
          31       119       180       31, 119, 180       "31 119 180"
          255      127       14        255, 127, 14       "255 127 14"
          44       160       44        44, 160, 44        "44 160 44"
          214      39        40        214, 39, 40        "214 39 40"
          148      103       189       148, 103, 189      "148 103 189"
```

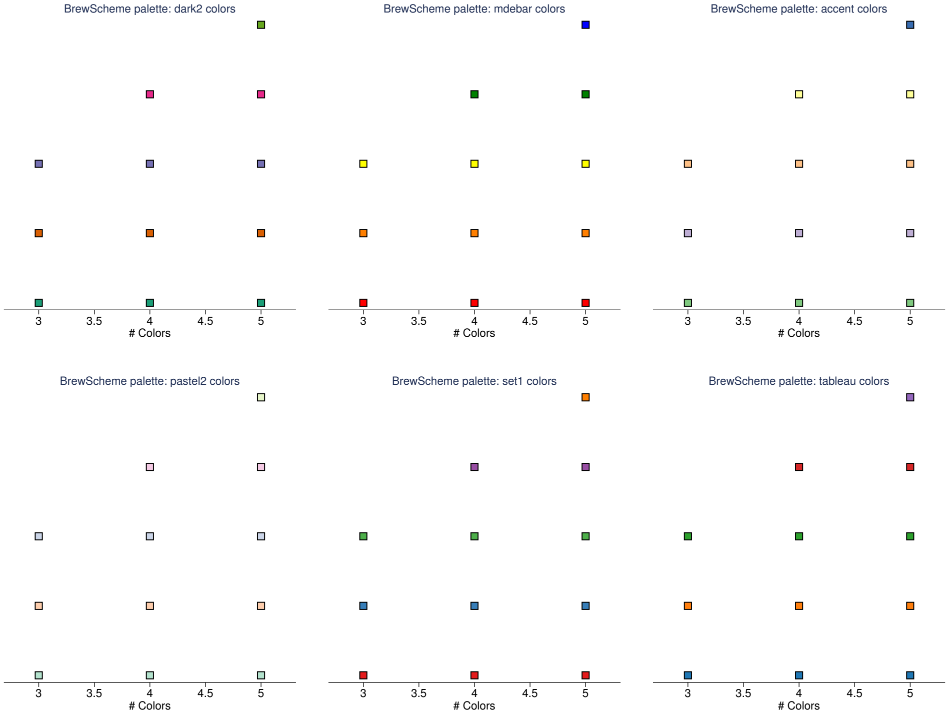

Figure 17: `brewviewer` example with multiple palettes and single sequential color

```
          140      86       75       140, 86, 75       "140 86 75"
          227      119      194      227, 119, 194     "227 119 194"
          127      127      127      127, 127, 127     "127 127 127"
          188      189      34       188, 189, 34      "188 189 34"
          23       190      207      23, 190, 207      "23 190 207"
--------------------------------------------------------------------------------
.
. // Or with a larger list of values
. hextorgb, hex("#1f77b4" "#aec7e8" "#ff7f0e" "#ffbb78" "#2ca02c" "#98df8a"        ///
> "#d62728" "#ff9896" "#9467bd" "#c5b0d5" "#8c564b" "#c49c94" "#e377c2"            ///
> "#f7b6d2" "#7f7f7f" "#c7c7c7" "#bcbd22" "#dbdb8d" "#17becf" "#9edae5")
--------------------------------------------------------------------------------
          Red       Green     Blue      RGB                 RGB String
--------------------------------------------------------------------------------
          31       119      180      31, 119, 180      "31 119 180"
          174      199      232      174, 199, 232     "174 199 232"
          255      127      14       255, 127, 14      "255 127 14"
          255      187      120      255, 187, 120     "255 187 120"
          44       160      44       44, 160, 44       "44 160 44"
          152      223      138      152, 223, 138     "152 223 138"
          214      39       40       214, 39, 40       "214 39 40"
```

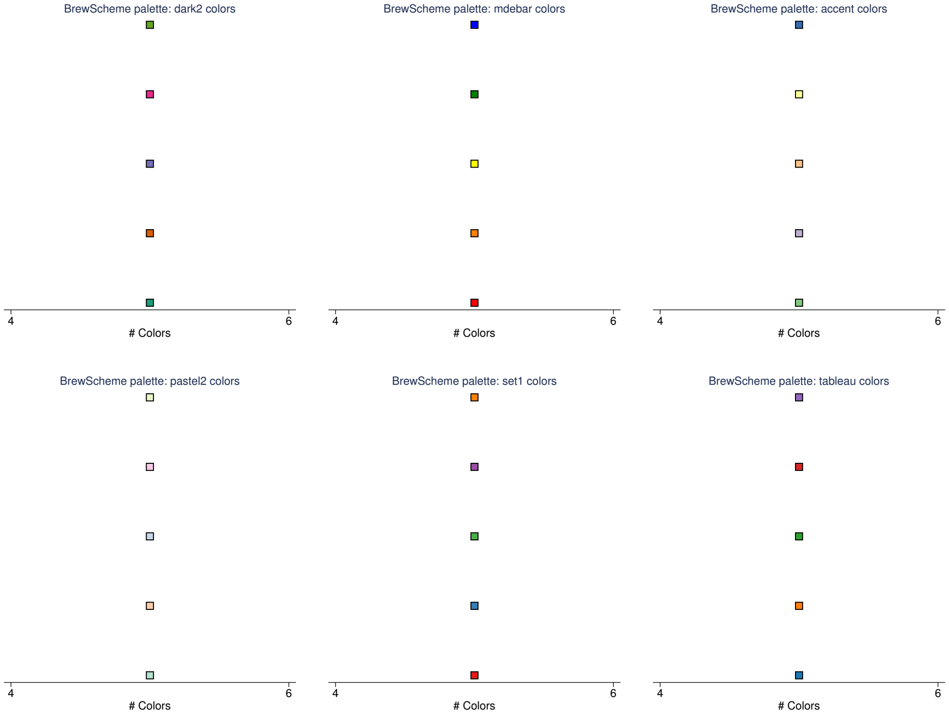
BrewScheme palette: dark2 colors
BrewScheme palette: mdebar colors
BrewScheme palette: accent colors
BrewScheme palette: pastel2 colors
BrewScheme palette: set1 colors
BrewScheme palette: tableau colors

Figure 18: `brewviewer` example with multiple palette and single color

```
          255      152      150      255, 152, 150     "255 152 150"
          148      103      189      148, 103, 189     "148 103 189"
          197      176      213      197, 176, 213     "197 176 213"
          140      86       75       140, 86, 75       "140 86 75"
          196      156      148      196, 156, 148     "196 156 148"
          227      119      194      227, 119, 194     "227 119 194"
          247      182      210      247, 182, 210     "247 182 210"
          127      127      127      127, 127, 127     "127 127 127"
          199      199      199      199, 199, 199     "199 199 199"
          188      189      34       188, 189, 34      "188 189 34"
          219      219      141      219, 219, 141     "219 219 141"
          23       190      207      23, 190, 207      "23 190 207"
          158      218      229      158, 218, 229     "158 218 229"
--------------------------------------------------------------------------------
.
end of do-file
```

◁

5.1 Java Plugins

brewterpolate

Although the commands discussed thus far provide significant options that can be used to create/install new named color styles and generate new scheme files, there has yet to be any discussion of generating any type of gradients and/or quantitative color scales that provide a mapping — or interpolation — between two points in a color space. The `brewterpolate` command is a Java-plugin that provides this capability to Stata users who have Java 8 or above installed. The command requires users to specify a starting and ending color value and the number of points between them that should be interpolated. There are also options available

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

`scolor` starting color

`ecolor` ending color

`colors` number of points to interpolate between starting and ending colors

`cmod(`*string*`)` is an optional argument that can take one of the following values: brighter, darker, saturated, desaturated, or nothing and is used to modify the interpolated colors. A value of "brighter" will return colors that are arbitrarily brighter than the normal interpolated value, "darker" will return colors that are arbitrarily darker than the normal interpolated colors, "saturated" will return arbitrarily more saturated colors, and "desaturated" will return arbitrarily less saturated colors. If no argument is passed to this parameter, the colors are interpolated without modification.

`icspace(`*string*`)` the colorspace of the starting and ending colors (see table 2 for more information).

`rcspace(`*string*`)` the colorspace in which the values are to be returned(see table 2 for more information).

`inverse` is an optional argument used to return the inverse of the interpolated colors. This is implemented after any of the luminance modifications have been made to the colors.

`grayscale` is an optional argument used to force the returned color values into a grayscale. This is the last transformation that is applied to the colors. In other words, if you requested the inverse of the colors that are arbitrarily less saturated, the method would first get the less saturated interpolated color, invert it, and then transform it to a gray scale value.

Table 2: Colorspaces available for `brewterpolate`

Argument	Colorspace
rgb	Red, Green, Blue (ex., 0 0 255)
rgba	Red, Green, Blue, Alpha (ex., 0 0 255 0.5)
srgb	RGB Decimal (ex., 0.0 0.0 1.0)
srgba	RGB Decimal (ex., 0.0 0.0 1.0 0.5)
hsb	Hue, Saturation, Brightness (ex., 270.0 1.0 1.0)
hsba	Hue, Saturation, Brightness (ex., 270.0 1.0 1.0 0.5)
web	Hex string (ex., 0000FF) [returned with leading # added]
weba	Hex string w/Decimal Alpha (ex., 0000FF .27) [returned with leading # added]
weba	Hex string w/Hex Alpha (ex., 0000FF00) [returned with leading # added]
hex	Hexadecimal (ex., 0000FF)
hexa	Hexadecimal w/Alpha (ex., 0000FF 0.5)
hexa	Hex w/Alpha scaled as RGB Integer (e.g., 0000FFFF)
hexa	web Web Hexadecimal w/Alpha (ex.,)
hsl	Hue, Saturation, Lightness (ex., 0.22, .75, .3725)
hsla	Hue, Saturation, Lightness w/Alpha (ex., 0.22 .75 .3725 0.275)

Macros

`r(`*start*`)`	Starting Color Value
`r(`*end*`)`	Ending Color value
`r(`*totalcolors*`)`	# of colors returned
`r(`*terpcolor#*`)`	#th Interpolated Color
`r(`*interpstart*`)`	Start Index in colors/colorsdelim
`r(`*interpend*`)`	End Index in colors/colorsdelim
`r(`*colorstring*`)`	Space-Delimited Colors
`r(`*colorsdelim*`)`	Comma-Delimited Colors

Examples The example below shows a basic usage of the `brewterpolate` command. Regardless of whether the colors are passed with or with out comma-delimiters, the program will handle the values appropriately. In the case where no input color space is defined, RGB is assumed.

▷ Example

```
. do `"$articledir/brewterpolateExamples.do"´

. /* brewterpolate examples */
.
. // Four colors interpolated in RGB color space
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(2)

.
. // Display the returned values
. ret li

macros:
        r(colorsdelim) : "197 115 47", "133 89 114", "69 63 182", "5 37 249"
        r(colorstring) : "197 115 47" "133 89 114" "69 63 182" "5 37 249"
          r(interpend) : "3"
        r(interpstart) : "2"
        r(totalcolors) : "4"
                r(end) : "5 37 249"
              r(start) : "197 115 47"
         r(terpcolor4) : "5 37 249"
         r(terpcolor3) : "69 63 182"
         r(terpcolor2) : "133 89 114"
         r(terpcolor1) : "197 115 47"

.
. // Four colors interpolated and returned as web formatted hexadecimal strings
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(3) rcs(web)

.
. // Display the returned values
. ret li

macros:
        r(colorsdelim) : "#c5732f", "#956062", "#654c94", "#3539c6", "#0525f9"
        r(colorstring) : "#c5732f" "#956062" "#654c94" "#3539c6" "#0525f9"
          r(interpend) : "4"
        r(interpstart) : "2"
        r(totalcolors) : "5"
                r(end) : "#0525f9"
              r(start) : "#c5732f"
         r(terpcolor5) : "#0525f9"
         r(terpcolor4) : "#3539c6"
         r(terpcolor3) : "#654c94"
         r(terpcolor2) : "#956062"
         r(terpcolor1) : "#c5732f"

.
. // Four colors less saturated colors returned as hex strings with alpha parameters
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(3) rcs(hexa) cm(desaturated)

.
. // Display the returned values
. ret li

macros:
        r(colorsdelim) : "c5732f 1.0", "6a9f9d 1.0", "9ab36b 1.0", "cac639 1.0", "fada06 1.0"
        r(colorstring) : "c5732f 1.0" "6a9f9d 1.0" "9ab36b 1.0" "cac639 1.0" "fada06 1.0"
          r(interpend) : "4"
        r(interpstart) : "2"
        r(totalcolors) : "5"
                r(end) : "fada06 1.0"
              r(start) : "c5732f 1.0"
         r(terpcolor5) : "fada06 1.0"
         r(terpcolor4) : "cac639 1.0"
         r(terpcolor3) : "9ab36b 1.0"
         r(terpcolor2) : "6a9f9d 1.0"
         r(terpcolor1) : "c5732f 1.0"

.
. // Three interpolated colors returned in RGB as a gray scale
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(2) g

.
. // Display the returned values
. ret li

macros:
        r(colorsdelim) : "197 115 47", "122 166 141", "186 192 73", "250 218 6"
        r(colorstring) : "197 115 47" "122 166 141" "186 192 73" "250 218 6"
          r(interpend) : "3"
        r(interpstart) : "2"
        r(totalcolors) : "4"
                r(end) : "250 218 6"
              r(start) : "197 115 47"
         r(terpcolor4) : "250 218 6"
         r(terpcolor3) : "186 192 73"
         r(terpcolor2) : "122 166 141"
         r(terpcolor1) : "197 115 47"
.
. // Arbitrariliy brighter version of example above
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(3) g cm(brighter)
.
. // Display the returned values
. ret li
macros:
        r(colorsdelim) : "197 115 47", "42 119 116", "111 146 44", "187 182 0", "250 217 0"
        r(colorstring) : "197 115 47" "42 119 116" "111 146 44" "187 182 0" "250 217 0"
          r(interpend) : "4"
        r(interpstart) : "2"
        r(totalcolors) : "5"
                r(end) : "250 217 0"
              r(start) : "197 115 47"
         r(terpcolor5) : "250 217 0"
         r(terpcolor4) : "187 182 0"
         r(terpcolor3) : "111 146 44"
         r(terpcolor2) : "42 119 116"
         r(terpcolor1) : "197 115 47"
.
. /* The use of mata below is primarily for the display/formatting of results but
> would otherwise be completely superfluous. */
.
. // Initalize null matrices to store results for he next three examples
. mata: hsb1 = J(6, 3, .)
.
. // Return the inverse of the original results in HSB color space
. brewterpolate, sc("197 115 47") ec("5, 37, 249") c(4) rcs("hsb") inv
.
. // Loop over returned results
. forv i = 1/6 {
  2.
.         // Store the results from the command above in a Mata matrix
.         mata: hsb1[`i´, .] = strtoreal(tokens(st_global("r(terpcolor`i´)")))
  3.
. } // End Loop over returned results
.
. // Return the matrices to Stata
. mata: st_matrix("hsb1", hsb1)
.
. // Add column names to each of the matrices
. mat colnames hsb1 = "Hue" "Saturation" "Brightness"
.
. // Add rownames to each of the matrices
. mat rownames hsb1 = "Color 1" "Color 2" "Color 3" "Color 4" "Color 5" "Color 6"
.
. // Print the first result set to the screen
. // RGB input returned in HSB color space
. mat li hsb1
```

```
hsb1[6,3]
                 Hue  Saturation  Brightness
Color 1    27.199999   .76142132   .77254903
Color 2    10.112354    .6413258   .62196076
Color 3    289.63636   .49183989   .50117648
Color 4       248.16   .84932905   .65960789
Color 5    236.65859           1   .81803924
Color 6    232.13115           1   .97647059

.
end of do-file
```

◁

To make the returned values more useful to others, the starting and ending values are reported as `terpcolors`. If you wanted to loop through only the values that were actually interpolated, you could used a `forvalues` loop like:

```
forv i = ‘r(interpstart)’/‘r(interpend)’ {
   di ‘"‘r(terpcolor‘i’)’"’
}
```

filesys

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

`attributes`

`display` Prints a table of the returned attributes to the results window.

`global` using inconjunction with the readable, writable, and executable options. Setting this parameter will apply the setting(s) passed to these arguments for all users. Without this parameter, the settings will be applied only for the current system's user.

`readable` accepts either ”on” or ”off” to make the given file readable or not-readable. When used with the global option, this can make the file globally readable or unreadable.

`writable` accepts either ”on” or ”off” to make the given file readable or not-readable. When used with the global option, this can make the file globally readable or unreadable.

`xecutable` accepts either ”on” or ”off” to make the given file readable or not-readable. When used with the global option, this can make the file globally readable or unreadable.

`readonly`

Macros

r(*created*)	String Created Date
r(*creatednum*)	SIF Created Date
r(*modified*)	String Modified Date
r(*modifiednum*)	SIF Modified Date
r(*accessed*)	String Last Access Date
r(*accessednum*)	SIF Last Access Date
r(*symlink*)	Symbolic Link Indicator
r(*regularfile*)	Regular File Indicator
r(*filesize*)	Filesize
r(*absolutepath*)	Absolute Filepath
r(*canonicalpath*)	Canonical Filepath
r(*isexecutable*)	Executable Attribute Set
r(*filename*)	Filename
r(*ishidden*)	Hidden File Indicator
r(*parentpath*)	Filepath to Parent Directory
r(*isreadable*)	Readable Attribute Set
r(*iswritable*)	Writable Attribute Set

Examples One difference between the `brewscheme` package and other Stata programs that include Mata libraries, is the method by which the `.mlib` file is created for users. Because one of the primary methods for distributing the program is from it's GitHub repository, the package attempts to detect the age of the Mata library on the user's system and will recompile it if needed. This is handled by the `brewlibcheck` command, which uses the `filesys` command to access file system attributes. This plugin and command are discussed here since it is likely to be useful to a wider audience of user-programmers. The examples below show how the program can be used interactively to inspect these properties, as well as programmatically via returned macros.

▷ Example

```
. do `"$articledir/filesysExamples.do"´

. /* filesys examples */
.
. // Get the file system attributes for the auto.dta file and print to screen
. filesys `c(sysdir_base)´a/auto.dta, attr dis

────────────────────────────────────────────────────────────────────────────────
    Attribute                 File Attribute Value
────────────────────────────────────────────────────────────────────────────────
    Created Date              20nov2015 05:44:54
    Modified Date             20nov2015 05:44:54
    Last Accessed Date        05apr2016 13:13:27
    Absolute File Path        /Applications/Stata/ado/base/a/auto.dta
    Canonical File Path       /Applications/Stata/ado/base/a/auto.dta
    Parent Path               /Applications/Stata/ado/base/a
    File Name                 auto.dta
    Is Symbolic Link          false
    Is Regular File           true
    Is Executable             false
    Is Hidden                 false
    Is Readable               true
    Is Writable               true
────────────────────────────────────────────────────────────────────────────────
```

```
.
. // Display the SIF version of the last accessed date with display formatting
. di %tc `r(accessednum)´
05apr2016 13:13:27

.
. // Make the data set globally executable
. filesys `c(sysdir_base)´a/auto.dta, x(on) glo dis

────────────────────────────────────────────────────────────────────────────────
    Attribute                 File Attribute Value
────────────────────────────────────────────────────────────────────────────────
    Created Date              20nov2015 05:44:54
    Modified Date             20nov2015 05:44:54
    Last Accessed Date        05apr2016 13:13:27
    Absolute File Path        /Applications/Stata/ado/base/a/auto.dta
    Canonical File Path       /Applications/Stata/ado/base/a/auto.dta
    Parent Path               /Applications/Stata/ado/base/a
    File Name                 auto.dta
    Is Symbolic Link          false
    Is Regular File           true
    Is Executable             true
    Is Hidden                 false
    Is Readable               true
    Is Writable               true
────────────────────────────────────────────────────────────────────────────────

.
. // And undo the change that was just made
. filesys `c(sysdir_base)´a/auto.dta, x(off) glo dis

────────────────────────────────────────────────────────────────────────────────
    Attribute                 File Attribute Value
────────────────────────────────────────────────────────────────────────────────
    Created Date              20nov2015 05:44:54
    Modified Date             20nov2015 05:44:54
    Last Accessed Date        05apr2016 13:13:27
    Absolute File Path        /Applications/Stata/ado/base/a/auto.dta
    Canonical File Path       /Applications/Stata/ado/base/a/auto.dta
    Parent Path               /Applications/Stata/ado/base/a
    File Name                 auto.dta
    Is Symbolic Link          false
    Is Regular File           true
    Is Executable             false
    Is Hidden                 false
    Is Readable               true
    Is Writable               true
────────────────────────────────────────────────────────────────────────────────

.
end of do-file
```

◁

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

brewlibcheck

This program is a wrapper used to check the user's system for the libbrewscheme Mata library. If the library does not exist, the program compiles it from source locally. If the file does exist, the program calls the `filesys` program to check when the library file was created. If the created date is earlier than the distribution date in the file, it will recompile the library for the user. Although this is a highly specific use case, it serves as an example of how other developers could use the `filesys` command to remove maintenance of mata libraries from the users.

```
brewlibcheck
```

brewdb

The `brewdb` command is used to parse and build the initial palette database for the `brewscheme` command to use. The program is called internally by `brewscheme` if the palette database is not found. Calling this program with the refresh option will result in all of the additional palettes — installed by `brewextra` — being removed. If you wish to rebuild the database locally, call the `brewextra` command with the refresh option. However, if you were interested in seeing how the javascript source code for the ColorBrewer Brewer (2002) palettes is parsed and structured, the source code in this file will show you how it was done.

```
brewdb [, refresh]
```

`refresh` an optional argument that will erase an existing instance of the color palette database if it exists before rebuilding the ColorBrewer palettes.

dirfile

The `dirfile` command is used to test whether or not specific filepaths exist and includes an option to create them if they do not exist. If the directory has files in it, this command also includes prompts that let the user determine if they wish to delete the contents of the subdirectory.

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

`path` is a required parameter that takes the filepath to be tested.

`rebuild` is an option used to rebuild the directory passed in the path parameter and provides an interactive method for users to approve/deny removal of files within the directory

brewsearch

The `brewsearch` command is used internally to search for named color styles and/or RGB values. If the value is found, the macro rgb will contain the passed value, and the remaining returned values contain the transformed RGB values. If the value is not found, the program returns the passed value in each of the macros. This command is used by `brewtheme` to test the arguments passed to the parameters of the program.

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

Macros

r(*rgb*)	RGB	Value
r(*achromatopsia*)	Achromatopsia	Simulated
r(*protanopia*)	Protanopia	Simulated
r(*deuteranopia*)	Deuteranopia	Simulated
r(*tritanopia*)	Tritanopia	Simulated

ExamplesExample

```
. do `"$articledir/brewsearchExamples.do"´
. /* brewsearch examples */
.
. // Search an RGB color string
. brewsearch "255 127 14"

.
. // Display the returned values
. ret li
macros:
        r(tritanopia) : "255 117 126"
      r(deuteranopia) : "206 153 0"
        r(protanopia) : "183 162 25"
     r(achromatopsia) : "146 146 146"
               r(rgb) : "255 127 14"

.
. // Search a named color style that does not exist on the system
. brewsearch "xkcd7327"

.
. // Display the returned values
. ret li
macros:
        r(tritanopia) : "xkcd7327"
      r(deuteranopia) : "xkcd7327"
        r(protanopia) : "xkcd7327"
     r(achromatopsia) : "xkcd7327"
               r(rgb) : "xkcd7327"

.
. // Search a named color style that does exist if the user installed the XKCD colors
. brewsearch "xkcd327"

.
. // Display the returned values
. ret li
```

```
macros:
         r(tritanopia) : "198 236 255"
       r(deuteranopia) : "255 218 50"
         r(protanopia) : "255 231 0"
      r(achromatopsia) : "218 218 218"
                r(rgb) : "168 255 4"
.
. // Display a known color style
. brewsearch "ltbluishgray"
.
. // Display the returned values
. ret li
macros:
         r(tritanopia) : "236 239 255"
       r(deuteranopia) : "255 232 245"
         r(protanopia) : "244 239 241"
      r(achromatopsia) : "240 240 240"
                r(rgb) : "234 242 243"
.
.
end of do-file
```

◁

brewtransform

The `brewtransform` program is used to create four variables containing the transformed RGB values in a variable in the current file. The variables created are: achromatopsia, protanopia, deuteranopia, and tritanopia and are added to the current dataset before populating them with the simulated values corresponding to the RGB string in the variable passed to the command. This is used internally to add these variables to user specified colors/palettes when updating/modifying the color and/or palette databases. The program is used internally to install the simulated versions of the XKCD (Monroe (2010)) named colors and the Stata named color styles.

`brewtransform` *varname*

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

To make installation easier for users, the `brewscheme` package includes an .ado file that handles the compilation of the libbrewscheme Mata library. The syntax below describe the use of the .ado file used to compile the library and is followed by an explanation of the mata library itself.

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

`display` is an option to bring up a help file that describes the mata library.

`replace` overwrites any existing version of the libbrewscheme mata library.

`size(#)` an option to pass a size argument to the `mata: mata mlib create` command.

The libbrewscheme Mata library consists of several objects and methods that will be briefly described here.

The Protanopia, Deuteranopia, and Tritanopia all inherit from the cbtype class. Each of these classes, when initializes, sets the member variables x, y, m, and yint to the values needed to transform an inputted RGB value into a simulated RGB value for each of those color sight impairments. These separate objects are initialized by the colorblind class object and the accessor methods defined in the cbtype class are used to extract the members when transforming an inputted RGB value. Because the typical use of Stata is more along the lines of a procedural/functional language, the library also includes a standalone mata function named `translateColor` which takes a red, green, and blue scalar arguments and returns the resulting colors to the user in the local macros: achromatopsia, protanopia, deuteranopia, and tritanopia.

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 & colorblind. 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.