b
UNIVERSITÄT
BERN

Faculty of Business, Economics and
Social Sciences

Department of Social Sciences

University of Bern Social Sciences Working Paper No. 31

Color palettes for Stata graphics

Ben Jann

April 12, 2018

http://ideas.repec.org/p/bss/wpaper/31.html
http://econpapers.repec.org/paper/bsswpaper/31.htm

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

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

Color palettes for Stata graphics

Ben Jann
University of Bern

Abstract. This paper introduces a command called `colorpalette` that provides a wide variety of color palettes and color generators for use in Stata graphics. Supported are color palettes from official Stata's graph schemes, a selection of palettes that have been proposed by users, standard collections such as the ColorBrewer or D3.js palettes, as well as HSV and HCL color generators. As a byproduct, the paper also introduces commands for marker symbol and line pattern palettes.

Keywords: st0001, `palettes`, `colorpalette`, `symbolpalette`, `linepalette`, `graph`, graphics, color, color spaces

1 Introduction

Stata features a set of about 50 named colors that can be used in graphs (see [G] ***colorstyle***). Given the diverse needs of users, a set of 50 predefined colors is rather limited. Alternative colors are supported, but have to be specified by their RGB, CMYK, or HSV values. To increase the number of easily accessible colors, the new `colorpalette` command provides a wide variety of predefined palettes and also features HSV (Hue-Saturation-Value) and HCL (Hue-Chroma-Luminance) color generators. Furthermore, it supports additional input formats for custom colors, such as hex triplets, and allows generating colors over a range if intensity or opacity levels.

A byproduct of `colorpalette` are two additional commands, `symbolpalette` and `linepalette`, that provide palettes of marker symbols and line pattern. These commands are briefly presented in the Appendix.

To install the new commands and view the documentation, type:

```
. scc install palettes, replace
. help palettes
```

2 Syntax and basic usage

The `colorpalette` command has two syntax variants. Syntax 1 is used to retrieve colors from one or multiple palettes. The colors are returned in `r()` and, by default, displayed in a graph. The syntax is

`colorpalette` [*argument*] [`,` *palette_options graph_options*]

where *argument* is

palette [[, *palette_options*] / [*palette* [, *palette_options*] / ...]]

and *palette* is a named palette as described below, or a space-separated list of named colors, RGB values, CMYK values, or HSV values according to [G] ***colorstyle***, HCL values specified as `"hcl` *h c l*`"`, where *h* specifies the hue (dominant wavelength in degrees of the 360 degree color wheel), *c* specifies the chroma (colorfulness; $c \geq 0$), and *l* specifies the luminance (brightness, amount of gray; $l \in [0, 100]$), or hex triplets specified as `#`*rrggbb*, where *rr*, *gg*, and *bb* are the two digit hex codes (or one digit abbreviations) for red, green, and blue. The specified colors can include intensity adjustment and, since Stata 15, an opacity level specified as `"`*color*[`*`*int*][`%`*op*]`"` , where $int \in [0, 1)$ makes the color lighter, $int > 1$ makes the color darker, and *op* is a number between 0 (fully transparent) and 100 (fully opaque).

Syntax 2 is used to display an overview of multiple palettes in a single graph, without returning the colors in `r()`. The syntax is

`colorpalette` [, *palette_options graph_options*]`:` *pspec* [/ *pspec* / ...]

where *pspec* is

palette [, *palette_options*]

or `.` to insert a gap.

Palette options

`n(`*#*`)` specifies the size of the palette (the number of colors). In many cases this just selects the first # colors from the palette and is thus equivalent to `select(1/`*#*`)`. However, some color schemes return colors that adjust to the size of the palette.

`select(`*numlist*`)` selects and orders the colors retrieved from the palette.

`reverse` returns the palette in reverse order.

`intensity(`*numlist*`)` applies color intensity adjustment. Values between 0 and 1 make the colors lighter; values larger than one make the colors darker. Specify multiple values to use different adjustments across the selected colors. The list of adjustments will be recycled if it is shorter than the list of selected colors. Likewise, colors will be recycled if the list of adjustments is longer than the palette.

`opacity(`*numlist*`)` sets the opacity level(s) (requires Stata 15). Values must be between 0 (fully transparent) and 100 (fully opaque). Specify multiple values to use different opacity levels across the selected colors. The list of opacity levels will be recycled if it is shorter than the list of selected colors. Likewise, colors will be recycled if the list of opacity levels is longer than the palette.

The above options are supported by all palettes, but some palettes also have additional options; see the descriptions of the palettes below.

Common graph options

`title(`*string*`)` specifies a custom title for the graph.

`gropts(`*twoway_options*`)` provides options to be passed through to the graph command; see [G] ***twoway_options***.

Additional graph options for syntax 1

`nograph` suppresses the graph.

`rows(#)` specifies the minimum number of rows in the graph. The default is 5.

Additional graph options for syntax 2

`horizontal` displays the palettes horizontally. This is the default.

`vertical` displays the palettes vertically.

`plabels(`*strlist*`)` provides custom labels for the palettes. Enclose labels with spaces in double quotes.

`lcolor(`*colorstyle*`)` specifies a custom outline color. The default is to use the same color as for the fill.

`lwidth(`*linewidthstyle*`)` specifies a custom outline thickness. The default is `lwidth(vthin)`.

Stored results

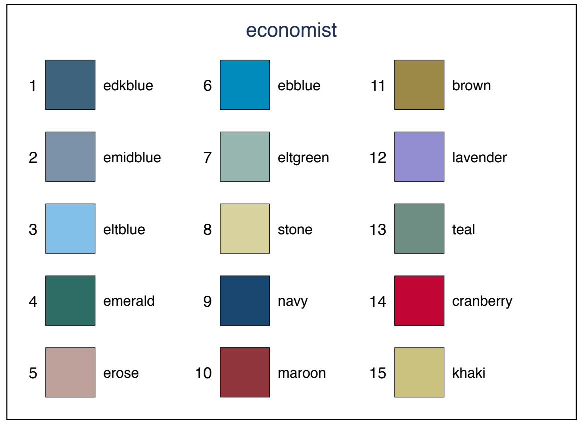
Under syntax 1, `colorpalette` stores the following in `r()`:

Scalars:			
`r(n)`	number of colors		
Macros:			
`r(ptype)`	`color`	`r(pname)`	name of palette or `custom`
`r(pnote)`	palette note (if available)	`r(p)`	space-separated list of colors
`r(p#)`	#th color	`r(p#info)`	info on #th color (if available)

2.1 Viewing a palette (syntax 1)

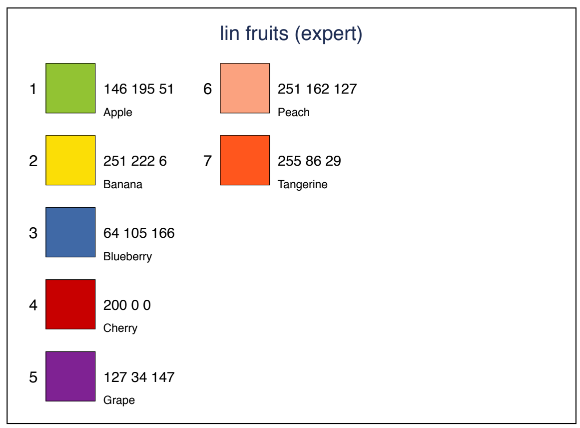
To display a single palette, type `colorpalette` followed by the name of the palette. For example, to view the `economist` palette, type:

```
. colorpalette economist
```



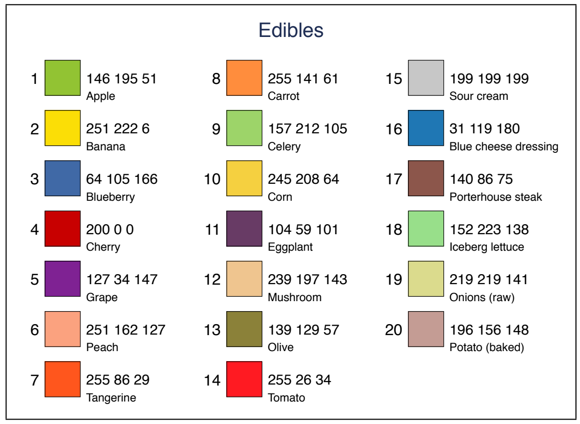
The graph produced by `colorpalette` displays the colors as well as their names or color codes and, possibly, some additional information. Here is an example of a semantic palette by Lin et al. (2013) with RGB codes and labels:

```
. colorpalette lin, fruits
```



It is also possible to combine colors from multiple palettes. Here is an example that uses the `fruits` palette, the `vegetable` palette, and a selection of colors from the `food` palette by Lin et al. (2013):

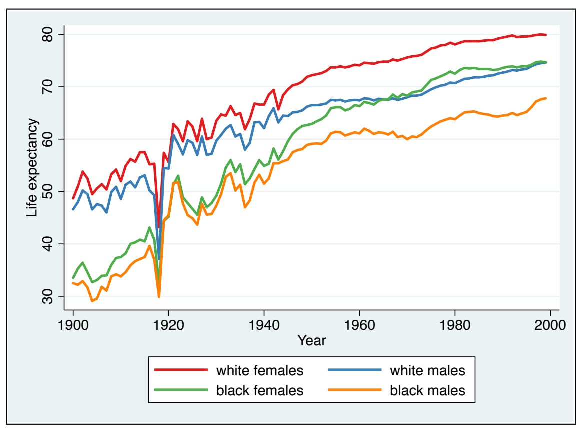
```
. colorpalette lin, fruits
>             / lin, vegetable
>             / lin, food select(1/6)
>               title("Edibles")
```



2.2 Retrieving colors from a palette (syntax 1)

`colorpalette` returns the values of the colors in `r()` so that they can be used in a subsequent graph command. `r(p)` will contain a space separated list of all colors, `r(p1)`, `r(p2)`, etc. will contain the single colors one by one. Here is an example that selects four colors from ColorBrewer's `Set1` palette and uses them in a line plot (option `nograph` is specified to prevent `colorpalette` from displaying the palette):

```
. sysuse uslifeexp, clear
(U.S. life expectancy, 1900-1999)
. lab var le_wfemale "white females"
. lab var le_wmale   "white males"
. lab var le_bfemale "black females"
. lab var le_bmale   "black males"
. colorpalette Set1, select(1/3 5)
>     nograph
. line le_wfemale le_wmale
>      le_bfemale le_bmale year
>      , lcolor(`r(p)') lwidth(*2 ..)
>      ytitle(Life expectancy)
```



Macro expansion notation `` `r(p)' `` instructs Stata to paste the contents of `r(p)` at the specified position within the command. Note that many commands, including most graph commands, clear `r()`. That is, if you want to use the same colors in multiple graphs without having to call `colorpalette` repeatedly, copy the colors to a local or global macro ([P] **macro**). For example, typing

```
. local mycolors `"`r(p)'"'
```

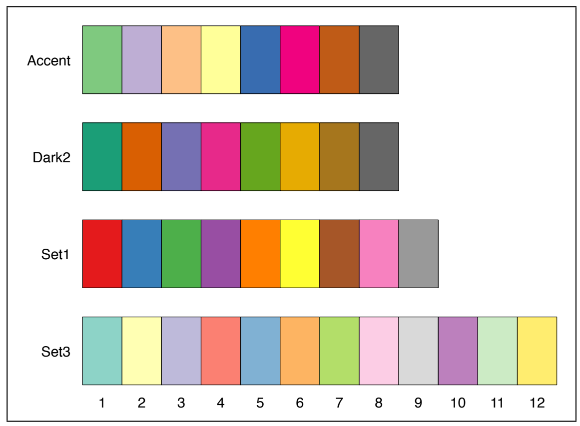
would copy the list of colors to local macro `mycolor`. You could then use the colors in subsequent graph commands by typing `` `mycolors' ``.

An alternative is to use the `grstyle` command to change the default colors used in Stata graphs; `grstyle` calls `colorpalette` internally (see Jann 2018).

2.3 Viewing multiple palettes (syntax 2)

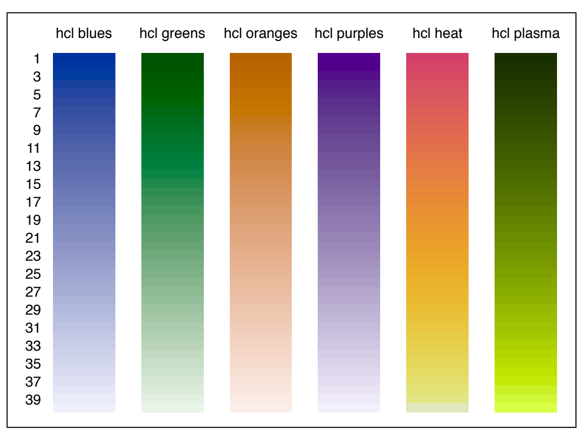
To display an overview of multiple palettes in a single graph, type `colorpalette`, a colon, and a list of palettes separated by forward slashes. The following example displays some of the categorical palettes from ColorBrewer (Brewer 2016; Brewer et al. 2003):

```
. colorpalette, lcolor(black):
>     Accent / Dark2 / Set1 / Set3
```



Option `lcolor(black)` has been specified to draw black lines around the color fields. Separate options can be specified for each palette. Here is an example that displays several default schemes of the HCL color generator. The example also illustrates the effect of the `vertical` option and the use of `n()` to determine the number of colors:

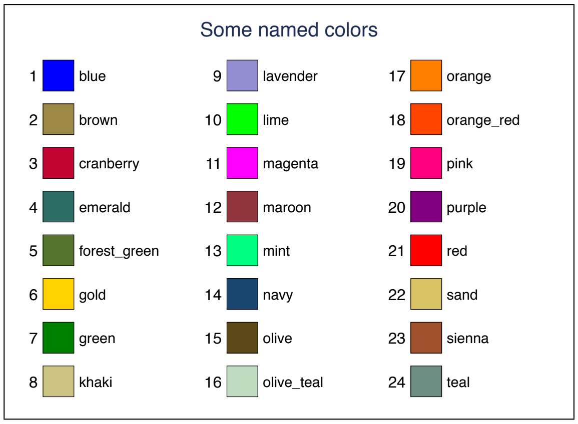
```
. colorpalette, vertical n(40):
>     hcl, blues / hcl, greens /
>     hcl, oranges / hcl, purples /
>     hcl, heat / hcl, plasma
```



2.4 Specifying a custom list of colors

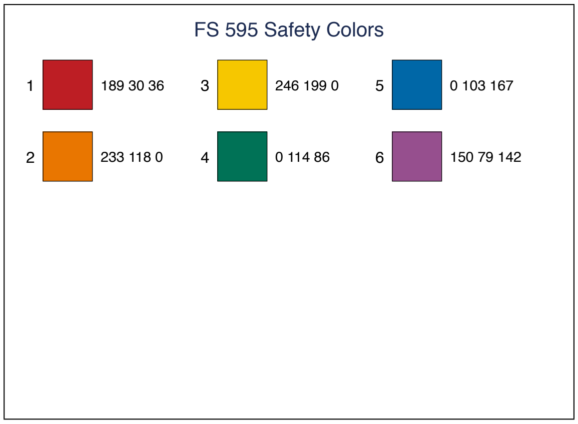
Instead of using a named palette you can provide a custom palette by specifying a list of *colorstyles* (named colors, RBG values, CMYK values, or HSV values; see [G] ***colorstyle***). Here is an example displaying some of Stata's named colors:

```
. colorpalette blue brown cranberry
>     emerald forest_green gold green
>     khaki lavender lime magenta
>     maroon mint navy olive
>     olive_teal orange orange_red
>     pink purple red sand sienna
>     teal, title(Some named colors)
```



In addition to the color specifications documented in [G] ***colorstyle***, you can specify colors using HCL codes (type `"hcl` *h c l*`"`, where *h*, *c*, and *l* are the values for hue, chroma, and luminance) or hex triplets. The specified colors will be translated to RGB. Here is an example displaying (approximate) Federal Standard 595 Safety Colors, using the hex codes found at www.w3schools.com/colors/colors_fs595.asp:

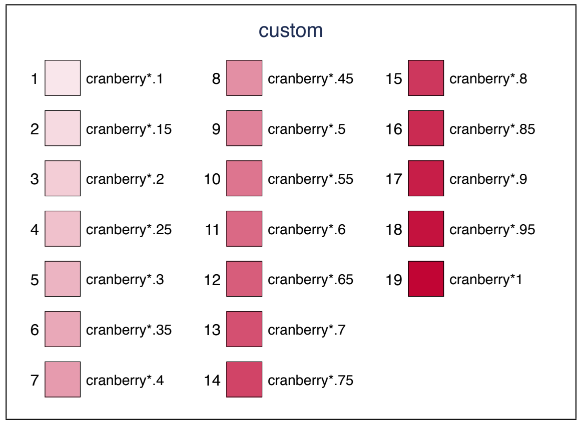
```
. colorpalette #bd1e24 #e97600
>     #f6c700 #007256 #0067a7
>     #964f8e, rows(2)
>     title(FS 595 Safety Colors)
```



2.5 Creating colors over a range of intensities or opacity levels

The `intensity()` and `opacity()` options can be used to apply intensity adjustment or assign opacity levels to the selected colors. Both options support number lists as argument (see [U] **11.1.8 numlist**). If the list of specified numbers is longer than the number of colors in the palette, the list of colors will be recycled. This allows creating colors over a range of intensities or opacity levels, as in the following example:

```
. colorpalette cranberry,
>     intensity(0.1(.05)1)
```

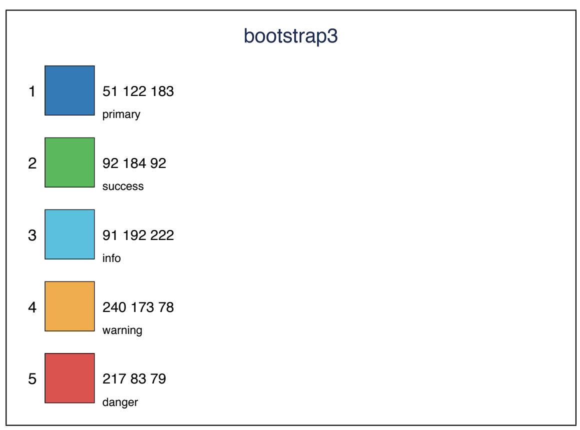


2.6 Custom palettes

If you want to create an own named color palette, you can define a program called `colorpalette_`*myname*, where *myname* is the name of your palette. Your program should return the color definitions (e.g., RGB values or hex codes) as a comma-separated list in local macro `P`. In addition you may provide a comma-separated list of descriptions in local macro `I`.

After defining the program, the new palette is available to `colorpalette` like any other palette. Here is an example providing a palette called `bootstrap3` containing semantic colors used for buttons in Bootstrap v3.3 (getbootstrap.com/docs/3.3):

```
. program colorpalette_bootstrap3
  1. c_local P #337ab7,#5cb85c,
>     #5bc0de,#f0ad4e,#d9534f
  2. c_local I primary,success,
>     info,warning,danger
  3. end
. colorpalette bootstrap3
```



More complicated definitions of palettes that take account of specific options are also possible. See the palette definitions in `colorpalette.ado` for examples.

If you intend to use the new palette in different analyses, you can store the program in an ado-file instead of including it in each of the do-files. If *myname* is the name of your palette, the program should be stored in file `colorpalette_`*myname*`.ado` in the working directory or somewhere along Stata's ado path (see [P] **sysdir**).

3 Predefined color palettes

This section provides an overview of the named palettes implemented in `colorpalette`. There are three types of palettes: Palettes providing the colors used for plots 1 to 15 in official Stata's graph schemes, palettes providing colors found in user-contributed schemes, and collections containing sets of palettes found in the literature.

3.1 Stata palettes

The Stata palettes are named after the schemes in which the colors are used. The palettes are:

`s1`	15 colors as in Stata's `s1color` scheme
`s1r`	15 colors as in Stata's `s1rcolor` scheme
`s2`	15 colors as in Stata's `s2color` scheme (the default palette)
`economist`	15 colors as in Stata's `economist` scheme
`mono`	15 gray scales as in Stata's monochrome schemes

Palette `s2` is the default used by `colorpalette` if no palette is specified. The left panel in Figure 1 displays an overview of the palettes.

3.2 User-contributed palettes

Stata users have contributed various scheme files in which alternative sets of colors are used, typically available from the Stata Journal site or from the SSC Archive. The following palettes have been constructed after some of these contributions.

`cblind` 9 colorblind-friendly colors suggested by Okabe and Ito (2002), including an additional gray as suggested at www.cookbook-r.com. The same colors are also used (in different order and using `gs10` for gray) in the `plotplainblind` and `plottigblind` schemes by Bischof (2017b).

`plottig` 15 colors used for plots 1 to 15 in the `plottig` scheme by Bischof (2017b). Most of these colors are the same as the colors produced by the `hue` color generator with default options (see below), although in different order.

`538` 6 colors used for plots 1 to 6 and 7 colors used for background, labels, axes, and confidence areas in the `538` scheme by Bischof (2017a). The palette replicates colors used at fivethirtyeight.com.

`mrc` 7 colors used for plots 1 to 7 in the `mrc` scheme by Morris (2013). These are colors according to guidelines by the UK Medical Research Council.

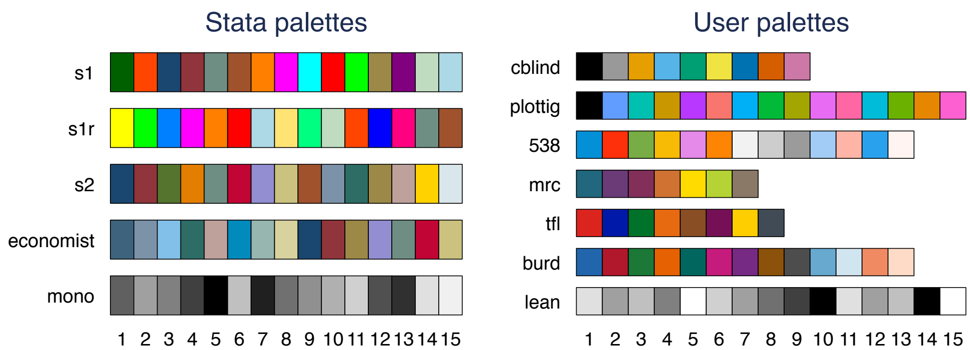


Figure 1: Stata palettes and user-contributed palettes

`tfl` 8 colors used for plots 1 to 8 in the `tfl` scheme by Morris (2015). The palette replicates Transport for London's corporate colors.

`burd` 9 colors used for plots 1 to 9 and 4 colors used for confidence areas in the `burd` scheme by Briatte (2013). The first 9 colors are a selection of colors from various ColorBrewer schemes.

`lean` 15 gray scales used for areas in plots 1 to 15 in schemes `lean1` and `lean2` by Juul (2003).

The right panel in Figure 1 displays an overview of these palettes.

3.3 Collections

ColorBrewer

ColorBrewer is a set of color schemes developed by Brewer et al. (2003; also see Brewer 2016). For more information on ColorBrewer also see colorbrewer2.org.[1] The syntax for the ColorBrewer palettes is

scheme [, `cmyk` *palette_options*]

where *palette_options* are general palette options as described above, `cmyk` requests the CMYK variant of the colors instead of the RGB variant, and `scheme` is one of the following:

1. The colors are licensed under Apache License Version 2.0; see the copyright notes at www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer_updates.html. The RGB values for the implementation of the colors in `colorpalette` have been taken from the Excel spreadsheet provided at www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer_RGB.html. The CMYK values have been taken from file `cb.csv` provided at github.com/axismaps/colorbrewer. ColorBrewer palettes for Stata have also been provided by Gomez (2015) and by Buchanan (2015).

Qualitative schemes

`Accent`	8 accented colors for qualitative data	`Dark2`	8 dark colors for qualitative data
`Paired`	12 paired colors for qualitative data	`Pastel1`	9 pastel colors for qualitative data
`Pastel2`	8 pastel colors for qualitative data	`Set1`	9 colors for qualitative data
`Set2`	8 colors for qualitative data	`Set3`	12 colors for qualitative data

Single-hue sequential schemes (3–9 colors)

`Blues`	light blue to blue	`Greens`	light green to green
`Greys`	light gray to gray	`Oranges`	light orange to orange
`Purples`	light purple to purple	`Reds`	light red to red

Multi-hue sequential schemes (3–9 colors)

`BuGn`	light blue to green	`BuPu`	light blue to purple
`GnBu`	light green to blue	`OrRd`	light orange to red
`PuBu`	light purple to blue	`PuBuGn`	light purple over blue to green
`PuRd`	light purple to red	`RdPu`	light red to purple
`YlGn`	light yellow to green	`YlGnBu`	light yellow over green to blue
`YlOrBr`	light yellow over orange to brown	`YlOrRd`	light yellow over orange to red

Diverging schemes (3–11 colors)

`BrBG`	brown to green, light gray mid	`PiYG`	pink to green, light gray mid
`PRGn`	purple to green, light gray mid	`PuOr`	purple to orange, light gray mid
`RdBu`	red to blue, light gray mid	`RdGy`	red to gray, white mid
`RdYlBu`	red to blue, yellow mid	`RdYlGn`	red to green, yellow mid
`Spectral`	red – orange – yellow – green – blue		

Figure 2 display the schemes (using the maximum number of colors for those schemes that come in different sizes).

Semantic colors by Lin et al.

The `lin` collection provides semantic color schemes suggested by Lin et al. (2013).[2] The syntax is

`lin` [, *scheme* `algorithm` *palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* is one of the following:

`tableau`	20 categorical colors; the default		
`carcolor`	6 car colors	`food`	7 food colors
`features`	5 feature colors	`activities`	5 activity colors
`fruits`	7 fruit colors	`vegetables`	7 vegetable colors
`drinks`	7 drinks colors	`brands`	7 brands colors

Option `algorithm` requests algorithm selected colors. The default is to return the colors selected by Turkers (in case of `carcolor`, `food`, `features`, `activities`) or by the expert (in case of `fruits`, `vegetables`, `drinks`, `brands`). Option `algorithm` has no effect for `tableau`. Figure 3 display the schemes.

2. The values of the semantic colors have been taken from the source code of the brewscheme package by Buchanan (2015) (`brewextra.ado`, version 1.0.0, 21 March 2016); the values of the `tableau` palette have been taken from code provided by Lin et al. at github.com/StanfordHCI/semantic-colors.

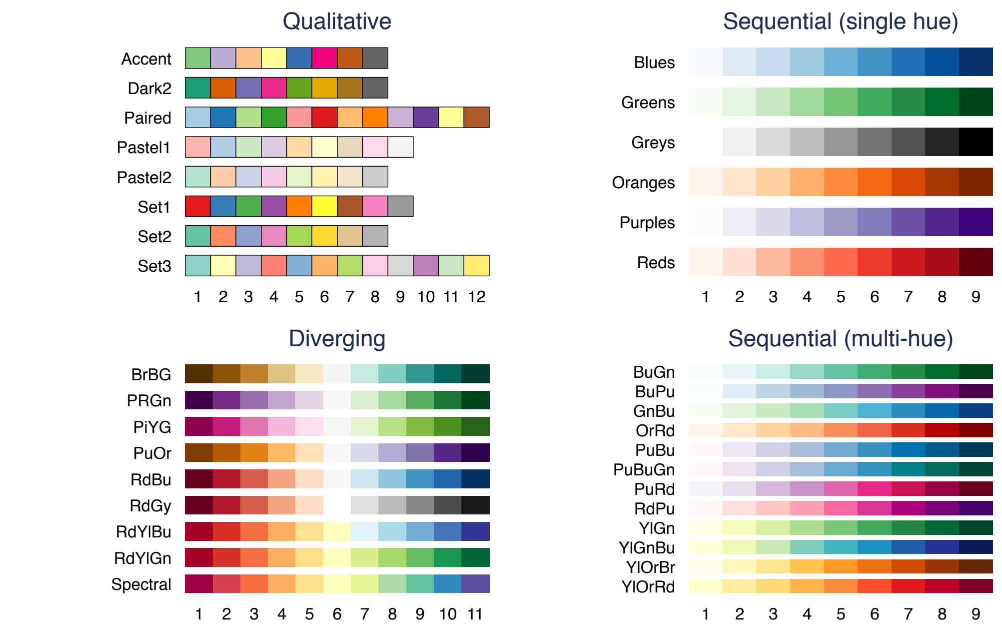


Figure 2: ColorBrewer schemes

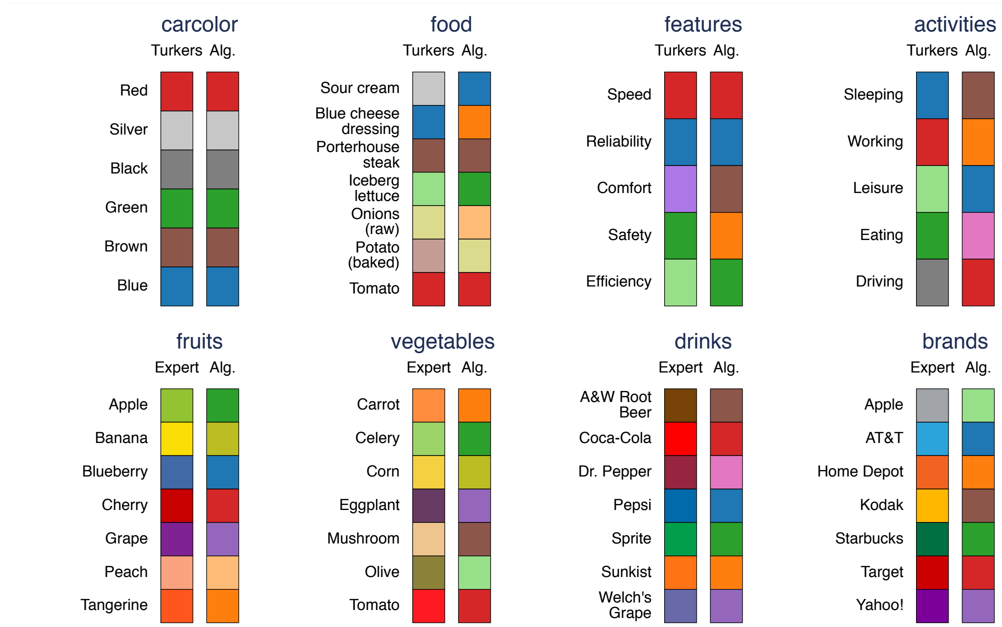


Figure 3: Semantic color schemes by Lin et al. (2013)

Color schemes by Paul Tol

The `ptol` collection provides color schemes as suggested by Tol (2012). The syntax is

`ptol` [, *scheme palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* is one of the following (displayed for for different numbers of colors in the upper left panel in Figure 4).

`qualitative`	1–12 qualitative colors; the default
`diverging`	3–11 diverging colors; very similar to reverse `RdYlBu` from ColorBrewer
`rainbow`	4–12 rainbow colors

D3.js

The `d3` collection provides color schemes from d3js.org, using the color values found at github.com/d3/d3-scale/blob/master/README.md#category-scales. The syntax is

`d3` [, *scheme palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* is one of the following (displayed in the upper right panel in Figure 4).

`10`	10 categorical colors; the default; same as the first 10 colors in the `tableau` scheme of the `lin` collection
`20`	20 categorical colors in pairs; same colors as in the `tableau` scheme of the `lin` collection, but in different order
`20b`	20 categorical colors in groups of four
`20c`	20 categorical colors in groups of four

Colors schemes from spmap

The `spmap` collection provides color schemes from the `spmap` package by Pisati (2007). The implementation is based on code from `spmap_color.ado` (version 1.3.0, 13 March 2017). The syntax is:

`spmap` [, *scheme palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* is one of the following (displayed for `n(16)` in the lower left panel in Figure 4).

`blues`	light blue to blue (2–99 colors); the default	`greens`	light green to green (2–99 colors)
`greys`	light gray to black (2–99 colors)	`reds`	light red to red (2–99 colors)
`rainbow`	2–99 rainbow colors	`heat`	2–16 heat colors
`terrain`	2–16 terrain colors	`topological`	2–16 topological colors

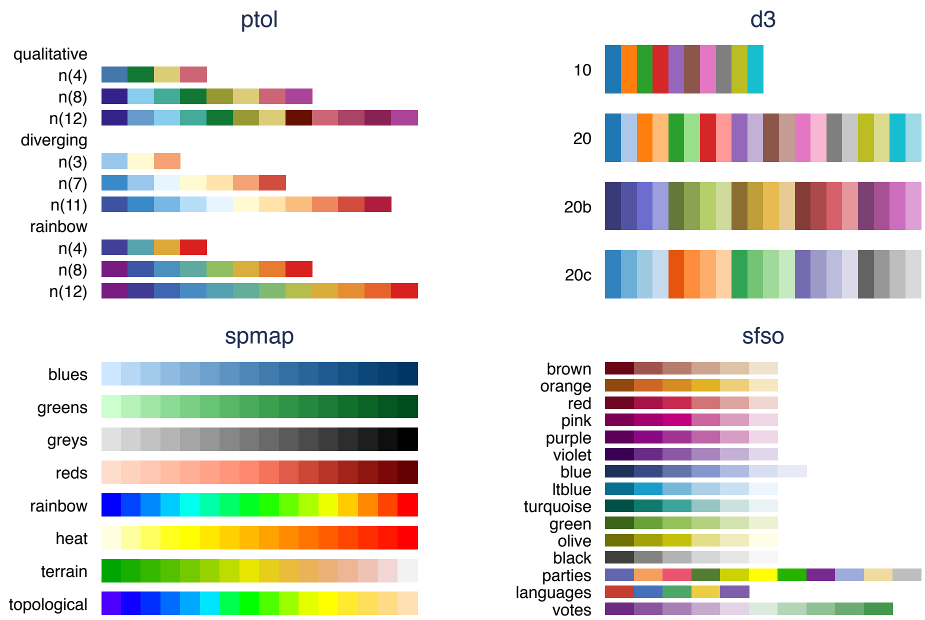


Figure 4: Various palettes

Swiss Federal Statistical Office colors

The `sfso` collection provides color schemes by the Swiss Federal Statistical Office (using hex and CMYK codes found in Bundesamt für Statistik 2017). The syntax is

`sfso` [, *scheme* `cmyk` *palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* is one of the following (displayed in the lower right panel in Figure 4).

Sequential schemes

`brown`	dark brown to light brown	`orange`	dark orange to light orange
`red`	dark red to light red	`pink`	dark pink to light pink
`purple`	dark purple to light purple	`violet`	dark violet to light violet
`blue`	dark blue to light blue; the default	`ltblue`	lighter version of `blue`
`turquoise`	dark turquoise to light turquoise	`green`	dark green to light green
`olive`	dark olive to light olive	`black`	dark gray to light gray

Semantic schemes

`parties`	colors used for Swiss parties	`languages`	colors used for languages
`votes`	colors used for results from votes		

Option `cmyk` requests the CMYK variant of a scheme. The default is to use the RGB variant.

4 Color generators

4.1 Evenly spaced HCL hues

The `hue` palette implements an algorithm that generates HCL colors with evenly spaced hues. The palette has been modeled after function `hue_pal()` from R's `scales` package by Hadley Wickham (see github.com/hadley/scales). This is the default color scheme used by R's `ggplot2` for categorical data (see ggplot2.tidyverse.org/reference/scale_hue.html). The `hue` palette with default options produces the same colors as the `intense` scheme of the `hcl` color generator (see below). The syntax of the `hue` color generator is

`hue` [, *hue_options palette_options*]

where *palette_options* are general palette options as discussed above and *hue_options* are:

`hue(`h_1 h_2`)` sets the range of hues on the 360 degree color wheel. The default is `hue(15 375)`. If the difference between start and end is a multiple of 360, end will be reduced by $360/n$, where n is the number of requested colors (so that the space between the last and the first color is the same as between the other colors).

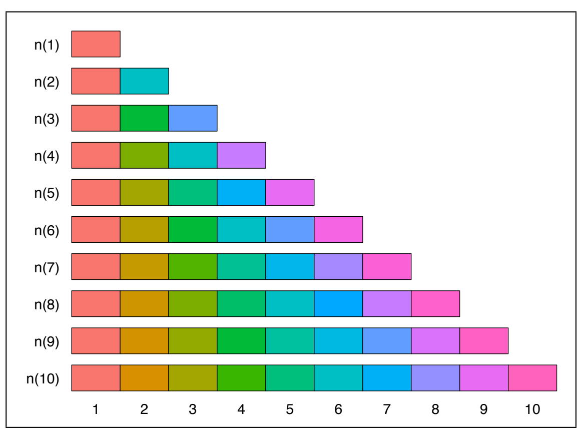
`chroma(`c`)` sets the colorfulness (color intensity), with $c \geq 0$. The default is `chroma(100)`.

`luminance(`l`)` sets the brightness (amount of gray), with $l \in [0, 100]$. The default is `luminance(65)`.

`direction(#)` determines the direction to travel around the color wheel. `direction(1)`, the default, travels clockwise; `direction(-1)` travels counter-clockwise.

The following graph illustrates how the colors change depending on option `n()`:

```
. colorpalette, plabels(n(1) n(2)
>         n(3) n(4) n(5) n(6) n(7)
>         n(8) n(9) n(10))
>     lcolor(black):
>     hue, n(1) / hue, n(2) /
>     hue, n(3) / hue, n(4) /
>     hue, n(5) / hue, n(6) /
>     hue, n(7) / hue, n(8) /
>     hue, n(9) / hue, n(10)
```



4.2 HCL color generator

The `hcl` palette is a HCL color generator (Hue-Chroma-Luminance). The implementation is based on R's `colorspace` package by Ihaka et al. (2016); also see Zeileis et al. (2009) and hclwizard.org.

Let h_1 and h_2 be two hues on the 360 degree color wheel, c_1 and c_2 two chroma levels, l_1 and l_2 two luminance levels, p_1 and p_2 two power parameters, and i an index from 1 to n, where n is the number of requested colors. The HCL colors are then generated according to the following formulas.

Qualitative schemes:

$$H = h_1 + j(h_2 - h_1), \quad C = c_1, \quad L = l_1, \quad j = \frac{i-1}{n-1}$$

Sequential schemes:

$$H = h_2 - j(h_2 - h_1), \quad C = c_2 - j^{p_1}(c_2 - c_1), \quad L = l_2 - j^{p_2}(l_2 - l_1), \quad j = \frac{n-i}{n-1}$$

Diverging schemes:

$$H = \begin{cases} h_1 & \text{if } j > 0 \\ h_2 & \text{else} \end{cases}, \quad C = |j|^{p_1} c_1, \quad L = l_2 - |j|^{p_2}(l_2 - l_1), \quad j = \frac{n-2j+1}{n-1}$$

The syntax of the `hcl` color generator is

`hcl` [`,` *scheme hcl_options palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* selects the type of scheme and the default parameters according to the following overview:

	h_1	h_2	c_1	c_2	l_1	l_2	p_1	p_2		h_1	h_2	c_1	c_2	l_1	l_2	p_1	p_2
Qualitative																	
`qualitative`	15	h^*	60	–	70	–	–	–	`intense`	15	h^*	100	–	65	–	–	–
`dark`	15	h^*	80	–	60	–	–	–	`light`	15	h^*	50	–	80	–	–	–
`pastel`	15	h^*	35	–	85	–	–	–		with $h^* = h_1 + 360(n-1)/n$							
Sequential																	
`sequential`	260	h_1	80	10	25	95	1	p_1	`blues`	260	h_1	80	10	25	95	1	p_1
`greens`	145	125	80	10	25	95	1	p_1	`grays`	0	h_1	0	0	15	95	1	p_1
`oranges`	40	h_1	100	10	50	95	1	p_1	`purples`	280	h_1	70	10	20	95	1	p_1
`reds`	10	20	80	10	25	95	1	p_1	`heat`	0	90	100	30	50	90	.2	1
`heat2`	0	90	80	30	30	90	.2	2	`terrain`	130	0	80	0	60	95	.1	1
`terrain2`	130	30	65	0	45	90	.5	1.5	`viridis`	300	75	35	95	15	90	.8	1.2
`plasma`	100	h_1	60	100	15	95	2	.9	`redblue`	0	-100	80	40	40	75	1	1
Diverging																	
`diverging`	260	0	80	–	30	95	1	p_1	`bluered`	260	0	80	–	30	95	1	p_1
`bluered2`	260	0	100	–	50	95	1	p_1	`bluered3`	180	330	60	–	75	95	1	p_1
`greenorange`	130	45	100	–	70	95	1	p_1	`browngreen`	55	160	60	–	35	95	1	p_1
`pinkgreen`	340	128	90	–	35	95	1	p_1	`purplegreen`	300	128	60	–	30	95	1	p_1

hcl_options are:

`hue(`h_1 $\left[h_2\right]$`)` overwrites the default values for h_1 and h_2 that determine the range of hues on the 360 degree color wheel.

`chroma(`c_1 $\left[c_2\right]$`)` overwrites the default values for c_1 and c_2, $c_i \geq 0$, that determine the colorfulness (color intensity).

`luminance(`l_1 $\left[l_2\right]$`)` overwrites the default values for l_1 and l_2, $l_i \in [0, 100]$, that determine the brightness (amount of gray).

`power(`p_1 $\left[p_2\right]$`)` overwrites the default values for p_1 and p_2, $p_i > 0$, that determine the shape of the transition between chroma and luminance levels. For linear transitions, set $p_i = 1$; $p_i > 1$ makes the transition faster, $p_i < 1$ makes the transition slower.

The left panel of Figure 5 displays the predefined HCL schemes with default parameters for $n = 15$.

4.3 HSV color generator

The `hsv` palette is a HSV color generator (Hue-Saturation-Value). The implementation is partially based on R's `grDevices` package (which is part of the R core) and partially on `colorspace` by Ihaka et al. (2016).

Let h_1 and h_2 be two hues on the 360 degree color wheel, s_1 and s_2 two saturation levels, v_1 and v_2 two value levels, p_1 and p_2 two power parameters, and i an index from 1 to n, where n is the number of requested colors. The HSV colors are then generated according to the following formulas.

Qualitative schemes:

$$H = h_1 + j(h_2 - h_1), \quad S = s_1, \quad V = v_1, \quad j = \frac{i-1}{n-1}$$

Sequential schemes:

$$H = h_2 - j(h_2 - h_1), \quad S = s_2 - j^{p_1}(s_2 - s_1), \quad V = v_2 - j^{p_2}(v_2 - v_1), \quad j = \frac{n-i}{n-1}$$

Diverging schemes:

$$H = \begin{cases} h_1 & \text{if } j > 0 \\ h_2 & \text{else} \end{cases}, \quad S = |j|^{p_1} s_1, \quad V = v_2 - |j|^{p_2}(v_2 - v_1), \quad j = \frac{n - 2j + 1}{n-1}$$

The syntax of the `hsv` color generator is

`hsv` [`,` *scheme hsv_options palette_options*]

where *palette_options* are general palette options as discussed above and *scheme* selects the type of scheme and the default parameters according to the following overview:

	h_1	h_2	s_1	s_2	v_1	v_2	p_1	p_2		h_1	h_2	s_1	s_2	v_1	v_2	p_1	p_2
Qualitative																	
`qualitative`	15	h^*	.4	–	.85	–	–	–	`intense`	15	h^*	.6	–	.9	–	–	–
`dark`	15	h^*	.6	–	.7	–	–	–	`light`	15	h^*	.3	–	.9	–	–	–
`pastel`	15	h^*	.2	–	.9	–	–	–	`rainbow`	15	h^*	1	–	1	–	–	–
										with $h^* = h_1 + 360(n-1)/n$							
Sequential																	
`sequential`	240	h_1	.8	.05	.6	1	1.2	p_1	`blues`	240	h_1	.8	.05	.6	1	1.2	p_1
`greens`	140	120	1	.1	.3	1	1.2	p_1	`grays`	0	h_1	0	0	.1	.95	1.0	p_1
`oranges`	30	h_1	1	.1	.9	1	1.2	p_1	`purples`	270	h_1	1	.1	.6	1	1.2	p_1
`reds`	0	20	1	.1	.6	1	1.2	p_1	`heat`	0	60	1	.2	1	1	0.3	p_1
`terrain`	120	0	1	0	.65	.95	0.7	1.5									
Diverging																	
`diverging`	240	0	–	.8	.6	.95	1.2	p_1	`bluered`	240	0	–	.8	.6	.95	1.2	p_1
`bluered2`	240	0	–	.6	.8	.95	1.2	p_1	`bluered3`	175	320	–	.6	.8	.95	1.2	p_1
`greenorange`	130	40	–	1	.8	.95	1.2	p_1	`browngreen`	40	150	–	.8	.6	.95	1.2	p_1
`pinkgreen`	330	120	–	.9	.6	.95	1.2	p_1	`purplegreen`	290	120	–	.7	.5	.95	1.2	p_1

hsv_options are:

`hue(`h_1 [h_2]`)` overwrites the default values for h_1 and h_2 that determine the range of hues on the 360 degree color wheel.

`saturation(`s_1 [s_2]`)` overwrites the default values for s_1 and s_2, $s_i \in [0,1]$, that determine the colorfulness (color intensity).

`value(`v_1 [v_2]`)` overwrites the default values for v_1 and v_2, $v_i \in [0,1]$, that determine the brightness (amount of gray).

`power(`p_1 [p_2]`)` overwrites the default values for p_1 and p_2, $p_i > 0$, that determine the shape of the transition between chroma and luminance levels. For linear transitions, set $p_i = 1$; $p_i > 1$ makes the transition faster, $p_i < 1$ makes the transition slower.

The right panel of Figure 5 displays the predefined HSV schemes with default parameters for $n = 15$.

5 Appendix: Symbol palettes and line pattern palettes

The `palettes` package also contains commands for symbol palettes and line pattern palettes. Their syntax and basic functionality is similar to the command for color palettes; see `help symbolpalette` and `help linepalette`. Figure 6 display an overview of the available named palettes.

6 References

Bischof, D. 2017a. G538SCHEMES: module to provide graphics schemes for http://fivethirtyeight.com. Statistical Software Components S458404, Boston College Department of Economics. Available from https://ideas.repec.org/c/boc/bocode/s458404.html.

———. 2017b. New graphic schemes for Stata: plotplain and plottig. *The Stata Journal*

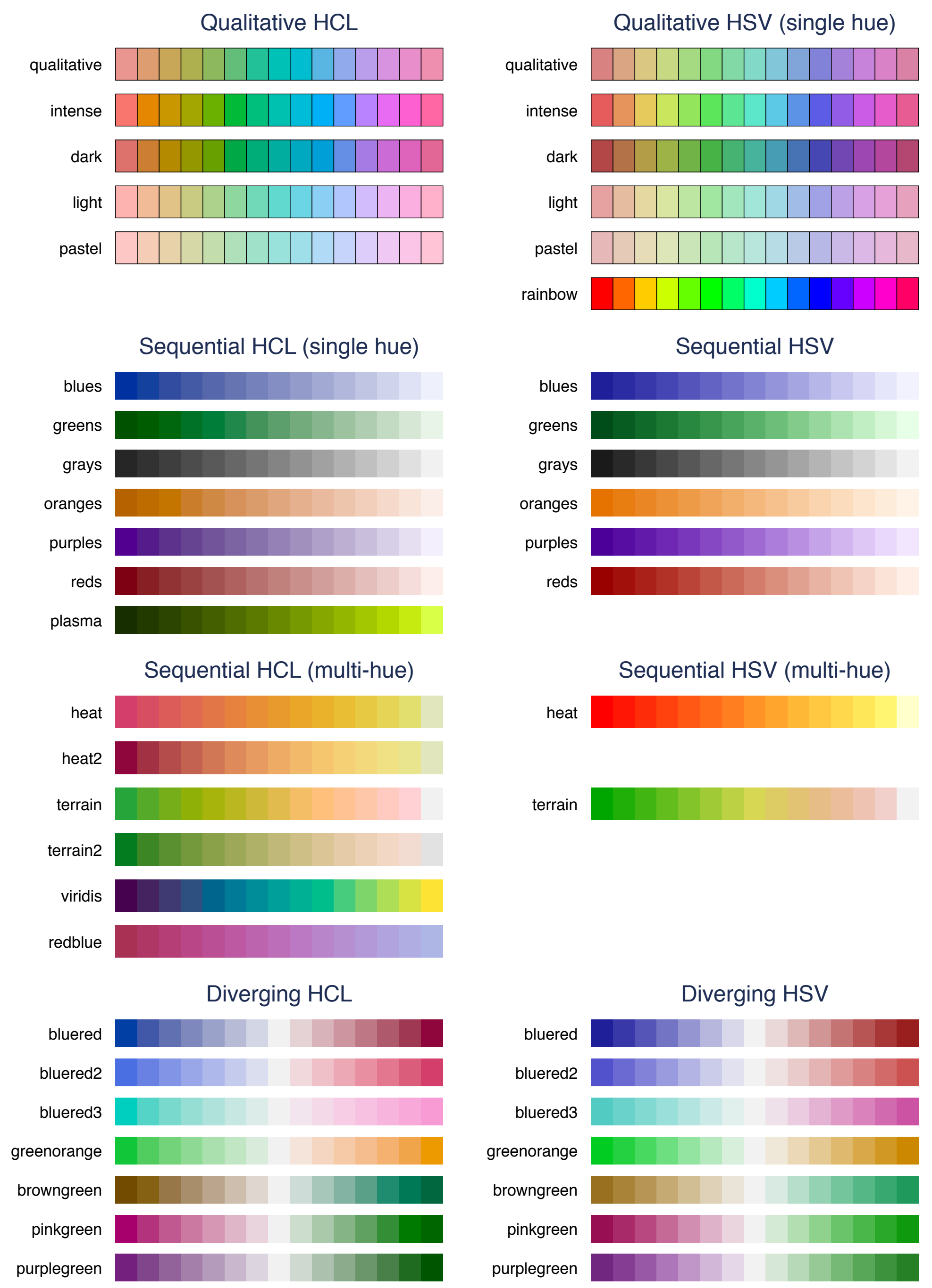


Figure 5: HCL and HCV color schemes

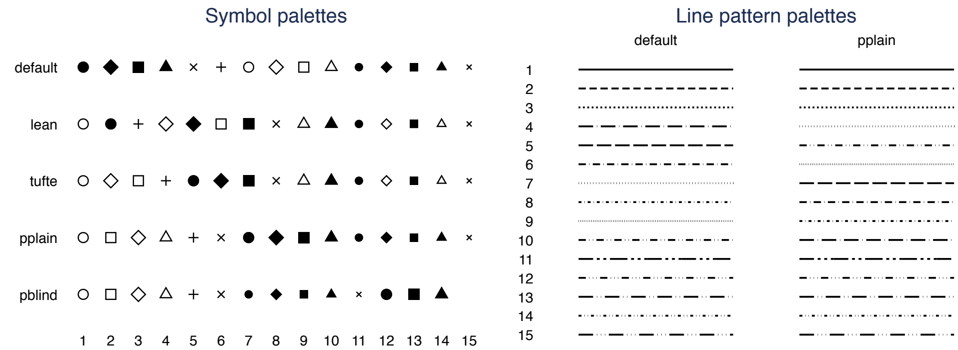


Figure 6: Symbol palettes and line pattern palettes

17(3): 748–759.

Brewer, C. A. 2016. *Designing Better Maps. A Guide for GIS Users.* 2nd ed. Redlands, CA: Esri Press.

Brewer, C. A., G. W. Hatchard, and M. A. Harrower. 2003. ColorBrewer in Print: A Catalog of Color Schemes for Maps. *Cartography and Geographic Information Science* 30(1): 5–32.

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

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

Bundesamt für Statistik. 2017. Layoutrichtlinien. Gestaltungs und Redaktionsrichtlinien für Publikationen, Tabellen und grafische Assets. Technical Report Version 1.1.1, Bundesamt für Statistik, Neuchâtel.

Gomez, M. 2015. Stata command to generate color schemes. Available from http://github.com/matthieugomez/stata-colorscheme.

Ihaka, R., P. Murrell, K. Hornik, J. C. Fisher, R. Stauffer, and A. Zeileis. 2016. colorspace: Color Space Manipulation. R package version 1.3-2. Available from http://CRAN.R-project.org/package=colorspace.

Jann, B. 2018. Customizing Stata graphs made even easier. University of Bern Social Sciences Working Papers 30, available from http://ideas.repec.org/p/bss/wpaper/30.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(3pt4): 401–410.

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

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

Okabe, M., and K. Ito. 2002. Color Universal Design (CUD). How to make figures and presentations that are friendly to Colorblind people. Available from http://jfly.iam.u-tokyo.ac.jp/color/.

Pisati, M. 2007. SPMAP: Stata module to visualize spatial data. Available from http://ideas.repec.org/c/boc/bocode/s456812.html.

Tol, P. 2012. Colour Schemes. SRON Technical Note, Doc. no. SRON/EPS/TN/09-002. Available from https://personal.sron.nl/~pault/colourschemes.pdf.

Zeileis, A., K. Hornik, and P. Murrell. 2009. Escaping RGBland: Selecting Colors for Statistical Graphics. *Computational Statistics & Data Analysis* 53: 3259–3270.

About the author

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