UNIVERSITÄT
BERN

Faculty of Business, Economics and Social Sciences

Department of Social Sciences

University of Bern Social Sciences Working Paper No. 43

Color palettes for Stata graphics: an update

Ben Jann

Current version: April 25, 2022
First version: April 20, 2022

http://ideas.repec.org/p/bss/wpaper/43.html
http://econpapers.repec.org/paper/bsswpaper/43.htm

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

Tel. +41 (0)31 684 48 11
Fax +41 (0)31 684 48 17
info@sowi.unibe.ch
www.sowi.unibe.ch

Color palettes for Stata graphics: an update

Ben Jann
Institute of Sociology
University of Bern
ben.jann@unibe.ch

Abstract. This paper is an update to Jann (2018b). It contains a comprehensive discussion of the `colorpalette` command, including various changes and additions that have been made to the software since its first publication. Command `colorpalette` provides colors for use in Stata graphics. In addition to Stata's default colors, `colorpalette` supports a variety of named colors, a selection of palettes that have been proposed by users, numerous collections of palettes and colormaps from sources such as ColorBrewer, Carto, D3.js, or Matplotlib, as well as color generators in different color spaces. The command also provides features such as color interpolation or color vision deficiency simulation.

Keywords: Stata, `palettes`, `colorpalette`, `colorcheck`, `graph`, graphics, color, color spaces, color interpolation, color vision deficiency, grayscale conversion, perceptually uniform

Revision notes: Substantial changes and additions compared to the first release of the software are marked with tags such as (new) or (revised) in the margin.

Contents

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 `colorpalette` command provides a variety of named colors and predefined palettes and also features color generators in different color spaces such as HSV (Hue-Saturation-Value) or HCL (Hue-Chroma-Luminance). Furthermore, it supports additional input formats for custom colors, such as hex triplets, and allows color interpolation or generating colors over a range if intensity or opacity levels.

The primary purpose of `colorpalette` is to provide named colors, color palettes, color generators, and additional color input formats for use with `grstyle set`, a command that customizes the look of Stata graphics (Jann 2018c). `grstyle set` runs `colorpalette` in the background so that users typically do not have to call `colorpalette` directly. However, manually calling `colorpalette` can be useful to display a quick overview of one or several palettes (see Section 3.2 and Section 3.3). Furthermore, `colorpalette` can also be used independently of `grstyle set` to retrieve colors and then pass them through to a subsequent graph command (see Section 3.6) or to make additional named colors available as macros or system colors (see Section 4.2 (new)

and Section 4.3).

Command `colorpalette` is provided as part of the `palettes` package (Jann 2017). The package also contains an older version of the command called `colorpalette9` as well as utilities for the management of marker symbols and line patterns. Furthermore, the package contains command `colorcheck` that can be used to evaluate whether the colors are distinguishable in (non-color) print or by people who suffer from color vision deficiency (see Section 3.5.4). (new)

The engine behind `colorpalette` and `colorcheck` is the `ColrSpace` class, a color management system written in Mata (Jann 2018a). `ColrSpace` is documented in Jann (2022). (new)

2 Installation

To install `palettes` and `ColrSpace`, type

```
. ssc install palettes, replace
. ssc install colrspace, replace
```

Alternatively, the packages are also available from GitHub (see github.com/benjann/palettes and github.com/benjann/colrspace). To install from GitHub, type

```
. net from https://raw.githubusercontent.com/benjann/palettes/master/
. net install palettes, replace
. net from https://raw.githubusercontent.com/benjann/colrspace/master/
. net install colrspace, replace
```

`colorpalette`, `colorcheck`, and `ColrSpace` require version 14.2 of Stata or newer. Users of older Stata versions can use command `colorpalette9`, which runs under Stata 9.2 or newer, but has limited functionality and somewhat different syntax. (new)

3 Syntax and basic usage

3.1 Syntax of colorpalette

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* *macro_options* *graph_options*]

where *argument* is

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

and *palette* is a space-separated list of individual colors (see Section 3.7 and Section 4), a named palette (see Section 3.8 and Section 5), or `mata(`*name*`)`, where *name* is the name of a `ColrSpace` object. (new)

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.

3.1.1 Palette options

Options `n()` through `cblind()` are applied in the order as listed below. For example, if you apply options `ipolate()` and `cblind()`, the colors will first be interpolated. Color vision deficiency simulation will then be applied to the the interpolated colors.

`n(`#`)` specifies the size of the palette (number of colors). Many palettes such as the color generators or the sequential and diverging ColorBrewer palettes are adaptive to `n()` in the sense that they return different colors depending on `n()`. Other palettes such as `s2` contain a fixed set of colors.

If `n()` is different from the (maximum or minimum) number of colors defined by a palette, the colors are either recycled or interpolated, depending on the class of the palette; see option `class()`. To prevent automatic recycling or interpolation, specify option `noexpand`. (new)

`select(`*numlist*`)` selects (and reorders) the colors retrieved from the palette. Positive numbers refer to positions from the start; negative numbers refer to positions from the end. `select()` cannot be combined with `order()` or `drop()`. (revised)

`drop(`*numlist*`)` drops individual colors retrieved from the palette. Positive numbers refer to positions from the start; negative numbers refer to positions from the end. Only one of `drop()` and `select()` is allowed. (new)

`order(`*numlist*`)` reorders the colors. Positive numbers refer to positions from the start; negative numbers refer to positions from the end. Colors not covered in *numlist* will be placed last, in their original order. Only one of `order()` and `select()` is allowed. (new)

`reverse` returns the colors in reverse order.

`shift(`#`)` shifts the positions of the colors up (if $\# > 0$) or down (if $\# < 0$), wrapping positions around at the end. If # is in $(-1, 1)$, the colors are shifted by `trunc(`$\#\times n$`)` positions, where n is the size of the palette (proportional shift); if $|\#| \geq 1$, the colors are shifted by `trunc(`#`)` positions. (new)

`opacity(`*numlist*`)` sets the opacity level(s) (this requires Stata 15 or newer). The values in *numlist* must be between 0 (fully transparent) and 100 (fully opaque). Specify multiple values to use different opacity levels across the selected colors. If the number of specified opacity levels is smaller than the number of colors, the levels will be recycled; if the number of opacity levels is larger than the number of colors, the colors will be recycled. To skip assigning opacity to a particular color, you may set the corresponding element in *numlist* to `.` (missing).

`intensity(`*numlist*`)` sets the color intensity adjustment multipliers. The values in *numlist* must be between 0 and 255. Values below 1 make the colors lighter; values larger than 1 make the colors darker (although the allowed scale goes up to 255, values as low as 5 or 10 may already make a color black). General behavior of *numlist* is as in `opacity()`.

`ipolate(`n`[, `*suboptions*`])` interpolates the colors to a total of n colors (intensity multipliers and opacity levels, if defined, will also be interpolated). Suboptions are as follows. (new)

cspace selects the color space in which the colors are interpolated. The default space is `Jab` (perceptually uniform CIECAM02-based $J'a'b'$). Other possibilities are, for example, `RGB`, `lRGB`, `Lab`, `LCh`, `Luv`, `HCL`, `JMh`, or `HSV`; see the documentation of `ColrSpace` for details (Jann 2022).

`range(`*lb* [*ub*]`)` sets the interpolation range, where *lb* and *ub* are the lower and upper bounds. The default is `range(0 1)`. If *lb* is larger than *ub*, the colors are returned in reverse order. Extrapolation will be applied if the specified range exceeds $[0, 1]$.

`power(#)`, with $\# > 0$, determines how the destination colors are distributed across the interpolation range. The default is to distribute them evenly; this is equivalent to `power(1)`. A power value larger than 1 squishes the positions towards the lower bound. If interpolating between two colors, this means that the first color will dominate most of the interpolation range (slow to fast transition). A value between 0 and 1 squishes the positions towards the upper bound, thus making the second color more dominant (fast to slow transition). Another way to think of the effect of `power()` is that it moves the center of the color gradient up (if $\# > 1$) or down (if $0 < \# < 1$).

`positions(`*numlist*`)` specifies the positions of the origin colors. The default is to arrange them on a regular grid from 0 and 1. If the number of specified positions is smaller than the number of origin colors, default positions are used for the remaining colors. If the same position is specified for multiple colors, these colors will be averaged before applying interpolation.

`padded` requests padded interpolation. By default, the first color and the last color are taken as the end points of the interpolation range; these colors thus remain unchanged. Specify `padded` to interpret the first and last colors as interval midpoints on an equally-spaced grid. This increases the interpolation range by half an interval on each side and causes the first color and the last color to be affected

by the interpolation.

Circular interpolation will be used for palettes declared as "`cyclic`" or "`circular`"; see option `class()`. For such palettes, suboptions `range()`, `power()`, `positions()`, and `padded` have no effect.

`intensify(`*numlist*`)` modifies the intensity of the colors. Syntax is as for `intensity()`. `intensify()` applies the same kind of adjustment as implemented by the intensity adjustment multipliers set by `intensity()`. The difference between `intensify()` and `intensity()` is that `intensity()` only records the intensity multipliers (which are then returned as part of the color definitions), whereas `intensify()` directly applies the adjustment by transforming the RGB values. A second difference is that `intensity()` is applied before interpolation, whereas `intensify()` is applied after interpolation. (new)

`saturate(`*numlist*`[, `*cspace*` level])` modifies the saturation (colorfulness) of the colors. Positive numbers will increase the chroma channel of the colors by the specified amount, negative numbers will reduce chroma. General behavior of *numlist* is as in `opacity()`. Suboptions are as follows. (new)

cspace specifies the color space in which the colors are manipulated. Possible spaces are `LCh` (cylindrical representation of CIE $L^*a^*b^*$), `HCL` (cylindrical representation of CIE $L^*u^*v^*$), `JCh` (CIECAM02 JCh), and `JMh` (CIECAM02-based $J'M'h$). The default is `LCh`.

`level` specifies that the provided numbers are levels, not differences. The default is to adjust the chroma values of the colors by adding or subtracting the specified amounts. Alternatively, if `levels` is specified, the chroma values of the colors will be set to the specified levels. Chroma values of typical colors lie between 0 and 100 or maybe 150.

`luminate(`*numlist*`[, `*cspace*` level])` modifies the luminance (brightness) of the colors. Positive numbers will increase the luminance of the colors by the specified amount, negative numbers will reduce luminance. General behavior of *numlist* is as in `opacity()`. Suboptions are as follows. (new)

cspace specifies the color space in which the colors are manipulated. Possible spaces are `Lab` (CIE $L^*a^*b^*$), `Luv` (CIE $L^*u^*v^*$), `JCh` (CIECAM02 JCh), and `JMh` (CIECAM02-based $J'M'h$) (`LCh`, `HCL`, and `Jab` are also allowed, but result in the same colors as `Lab`, `Luv`, and `JMh`, respectively). The default is `JMh`.

`level` specifies that the provided numbers are levels, not differences. The default is to adjust the luminance values of the colors by adding or subtracting the specified amounts. Alternatively, if `levels` is specified, the luminance values of the colors will be set to the specified levels. Luminance values of typical colors lie between 0 and 100.

`gscale[(`[*numlist*]`[, `*cspace*`])]` converts the colors to gray, where *numlist* in $[0, 1]$ specifies the proportion of gray. The default proportion is 1 (full conversion). General behavior of *numlist* is as in `opacity()`. Suboption *cspace* specifies the color space in (new)

which the conversion is performed; it may be `LCh` (cylindrical representation of CIE $L^*a^*b^*$), `HCL` (cylindrical representation of CIE $L^*u^*v^*$), `JCh` (CIECAM02 JCh), and `JMh` (CIECAM02-based $J'M'h$). The default is `LCh`.

`cblind`[`(`[*numlist*][`,` *type*]`)`] simulates color vision deficiency (based on Machado et al. 2009), where *numlist* in $[0, 1]$ specifies the severity of the deficiency. The default severity is 1 (maximum severity, i.e. deuteranopia, protanopia, or tritanopia, respectively). General behavior of *numlist* is as in `opacity()`. Suboption *type* specifies the type of color vision deficiency, which may be `deuteranomaly` (the default), `protanomaly`, or `tritanomaly`. See en.wikipedia.org/wiki/Color_blindness for basic information on color blindness. (new)

`forcergb` enforces translation of all colors to RGB. By default, `colorpalette` does not translate colors specified as Stata color names. Specify `forcergb` to return these colors as RGB values. (new)

`noexpand` omits recycling or interpolating colors if the number of requested colors is larger than the maximum (or smaller than the minimum) number of colors defined in a palette. (new)

`class(`*class*`)` declares the class of the the palette, where *class* may be `qualitative` (or `categorical`), `sequential`, `diverging`, `cyclic` (or `circular`), or any other string. Palettes declared as `qualitative` or `categoprical` will be recycled, all other palettes will be interpolated (if recycling or interpolation is necessary). Specifying `class()` only affects palettes that do not set the class as part of their definition. (new)

`name(`*str*`)` assigns a custom name to the palette. (new)

other_options are additional palette-specific options. See the descriptions of the palettes below (Section 5). When collecting results from multiple palettes, palette options can be specified at the global level to define default settings for all palettes, or at the local level of an individual palette. For general palette options, defaults set at the global level can be overridden by repeating an option at the local level. Such repetitions are not allowed for palette-specific options.

3.1.2 Macro options

The following options are only available in syntax 1.

`globals`[`(`*spec*`)`] stores the color codes as global macros (see [P] **macro**). Use this option as an alternative to obtaining the color codes from `r()`; see the Section 4.2 for an example. `globals()` disables graph display unless option `graph` is specified. The syntax of *spec* is (new)

[*namelist*] [*stub*`*`] [`, prefix(`*prefix*`) suffix(`*suffix*`) nonames`]

where *namelist* provides custom names for the colors and *stub*`*` provides a stub for automatic names. If no name is found for a color in the palette definition and no

custom name is provided, an automatic name defined as *stub*#*suffix* will be used, where # is the number of the color in the palette. The default *stub* is `p` or as set by `prefix()`. Suboptions are as follows:

`prefix()` specifies a common prefix to be added to the names.

`suffix()` specifies a common suffix to be added to the names.

`nonames` prevents `colorpalette` from using the names found in the palette definition.

`locals`[`(`*spec*`)`] stores the color codes as local macros (see [P] **macro**). Syntax and functionality is as described for option `globals()`, with the exception that *stub* defaults to empty string. `locals()` disables graph display unless option `graph` is specified. (new)

`stylefiles`[`(`*spec*`)`] stores the color codes in style files on disk. This makes the colors permanently available by their name, just like official Stata's color names; see Section 4.3 for an example. Style files will only be created for colors that are represented by a simple RGB code; codes that include an intensity-adjustment or opacity operator and colors that are referred to by their Stata name will be skipped. `stylefiles()` disables graph display unless option `graph` is specified. The syntax of *spec* is (new)

[*namelist*] [*stub*`*`] [`, prefix(`*prefix*`) suffix(`*suffix*`) nonames personal path(path) replace `]

where *namelist* provides custom names for the colors and *stub*`*` provides a stub for automatic names. If no name is found for a color in the palette definition and no custom name is provided, an automatic name defined as *stub*#*suffix* will be used, where # is the number of the color in the palette. The default *stub* is empty string or as set by `prefix()`. Suboptions are as follows:

`prefix()` specifies a common prefix to be added to the names.

`suffix()` specifies a common suffix to be added to the names.

`nonames` prevents `colorpalette` from using the names found in the palette definition.

`personal` causes the style files to be stored in folder "style" within the `PERSONAL` ado-file directory; see [P] **sysdir**. The default is to store the style files in folder "style" within the current working directory; see [D] **pwd**.

`path(`*path*`)` provides a custom path for the style files. The default is to store the style files in folder "style" within the current working directory. `path()` and `personal` are not both allowed.

`replace` permits `colorpalette` to overwrite existing files.

3.1.3 Graph options

Common graph options

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

`nonumbers` suppresses the numbers identifying the colors in the graph. (new)

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

Additional graph options for syntax 1

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

`names` replaces the RGB values in the graph by the information found in `r(p#name)` (the color names), if such information is available. (new)

`noninfo` suppresses the additional color information that is sometimes printed below the RGB values or the color names. (new)

`nograph` suppresses the graph.

`graph` enforces drawing a graph even though *macro_options* have been specified.

Additional graph options for syntax 2

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

`vertical` displays the palettes vertically.

`span` adjusts the size of the color fields such that each palette spans the full plot region even if the palettes contain different numbers of colors. (new)

`barwidth(#)` sets the width of the color bars. The default is `barwidth(0.7)`. The available space per bar is 1 unit; specifying `barwidth(1)` will remove the gap between bars. (new)

`labels(`*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)`.

3.1.4 Stored results

Under syntax 1, `colorpalette` stores the following in `r()` (see [R] **Stored results**):

Scalars:

`r(n)`	number of colors

Macros:

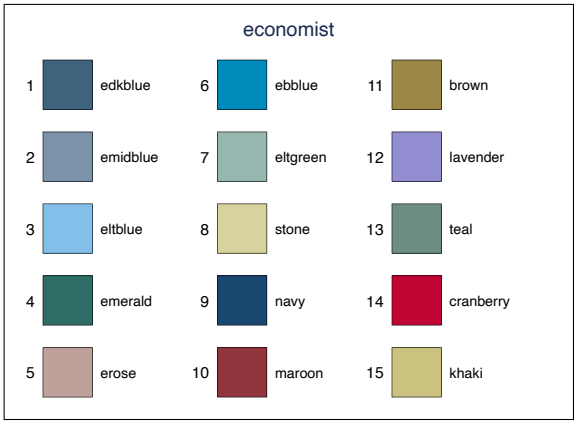
`r(ptype)`	`color`	`r(pname)`	name of palette or `custom`
`r(pclass)`	palette class (if provided)	`r(pnote)`	palette description (if provided)
`r(psource)`	palette source (if provided)	`r(p)`	space-separated list of colors
`r(p#)`	#th color	`r(p#name)`	name of #th color (if provided)
`r(p#info)`	info of #th color (if provided)		

Under syntax 2, `colorpalette` does not store any results.

3.2 View a palette (syntax 1)

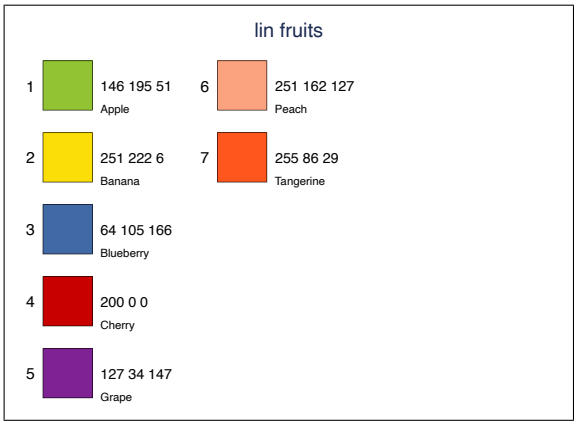
To display a single palette, type `colorpalette` followed by the name of the palette. For example, to view the `economist` palette, type:

```
. colorpalette economist
```



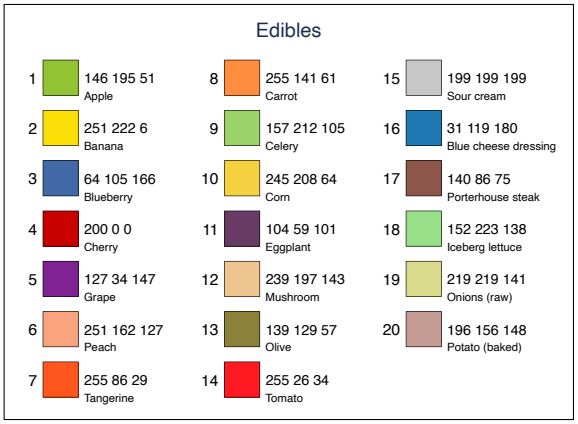
The graph produced by `colorpalette` displays the colors as well as their names or color codes and, possibly, some additional information. Here is an example of a semantic palette by Lin et al. (2013) with RGB codes and labels:

```
. colorpalette lin fruits
```



It is also possible to combine colors from multiple palettes by delimiting individual palette specifications using a forward slash. Specify "global" options, that is, options affecting the rendering of the graph as well as palette options to be applied to each palette, after the last forward slash. In addition, each palette can have "local" options. Options specified at the local level will take precedence over palette option specified at the global level. Here is an example that combines the `fruits` palette, the `vegetable` palette, and a selection of colors from the `food` palette by Lin et al. (2013):

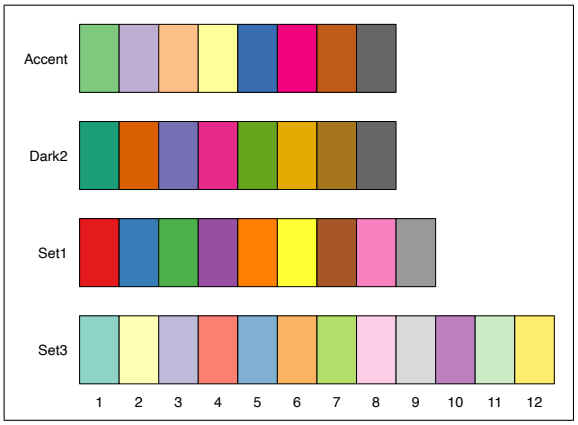
```
. colorpalette lin fruits
>            / lin vegetable
>            / lin food, select(1/6)
>            / , title("Edibles")
```



3.3 View multiple palettes (syntax 2)

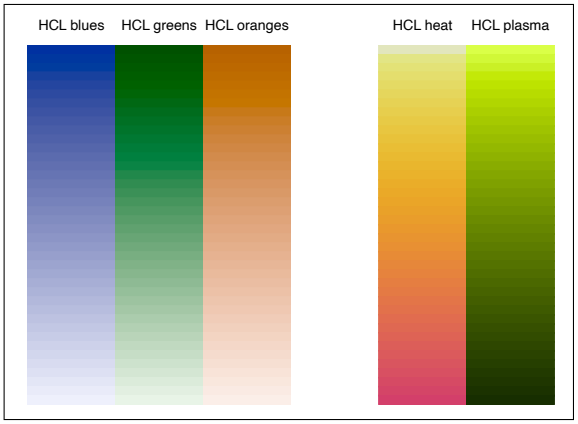
To display an overview of multiple palettes in a single graph, first type `colorpalette` and possibly add some global options, then type a colon followed by a list of palettes separated by forward slashes. Here is a simple example that displays some categorical palettes from ColorBrewer, where global option `lcolor(black)` has been specified to draw black lines around the color fields.

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



Again, each palette can have local options that take precedence over options specified at the global level. Furthermore, several options are available to change the rendering of the graph. The following example illustrates the effects options `vertical` (flip orientation), `barwidth()` (set the with of the color bars), and `nonumbers` (suppress the color index), and also shows how empty slots can be introduced typing a dot (missing); in addition, the example uses option `n()` to determine the number of colors to be generated per palette:

```
. colorpalette, n(40) vertical
>     barwidth(1) nonumbers:
>       hcl blues
>     / hcl greens
>     / hcl oranges
>     / .
>     / hcl heat, reverse
>     / hcl plasma, reverse
```



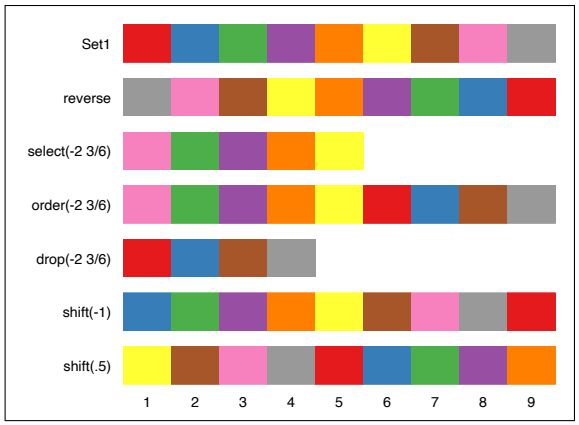
3.4 Select and order colors

(revised)

Some of the above examples already made use of options for selecting and ordering the colors in a palette. Five such options are available: `select()` to select and order colors, `drop()` to drop individual colors, `order()` to order colors without selecting, `reverse` to reverse the order of the colors, and `shift()` to shift the positions of the colors up or down. Positive numbers in `select()`, `drop()`, and `order()` refer to positions from the

start, negative numbers refer to positions from the end. A positive number in `shift()` shifts positions up, a negative number shifts positions down; furthermore, an absolute value smaller than one can be used to specify the shift in terms of a proportion of the number of the colors in the palette. The following example illustrates the effects of these options:

```
. colorpalette, labels(Set1
>     reverse "select(-2 3/6)"
>     "order(-2 3/6)" "drop(-2 3/6)"
>     shift(-1) shift(.5)):
>       Set1
>     / Set1, reverse
>     / Set1, select(-2 3/6)
>     / Set1, order(-2 3/6)
>     / Set1, drop(-2 3/6)
>     / Set1, shift(-1)
>     / Set1, shift(.5)
```



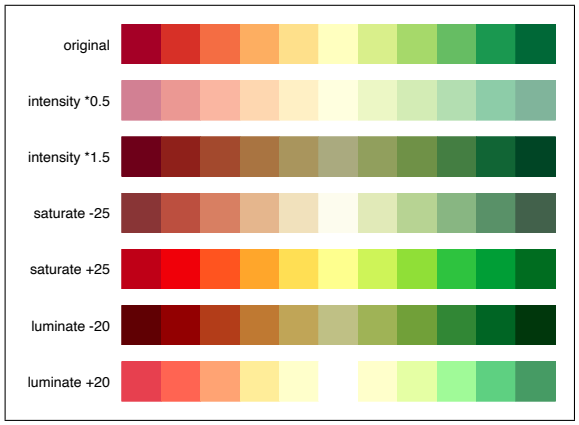
3.5 Manipulate and analyze colors

3.5.1 Change intensity, saturation, and luminance

(new)

Options `intensify()`, `saturate()`, and `luminate()` can be used to change the intensity (using same formulas as Stata's intensity operator), the saturation (colorfulness), and the luminance (brightness) of colors, respectively. The following example illustrates the effects of these options:

```
. colorpalette, nonumbers
>     labels(original "intensity *0.5"
>     "intensity *1.5" "saturate -25"
>     "saturate +25" "luminate -20"
>     "luminate +20"):
>       RdYlGn
>     / RdYlGn, intensify(0.5)
>     / RdYlGn, intensify(1.5)
>     / RdYlGn, saturate(-25)
>     / RdYlGn, saturate(25)
>     / RdYlGn, luminate(-20)
>     / RdYlGn, luminate(20)
```



The values specified in `saturate()` and `luminate()` are addends to the chroma and lu-

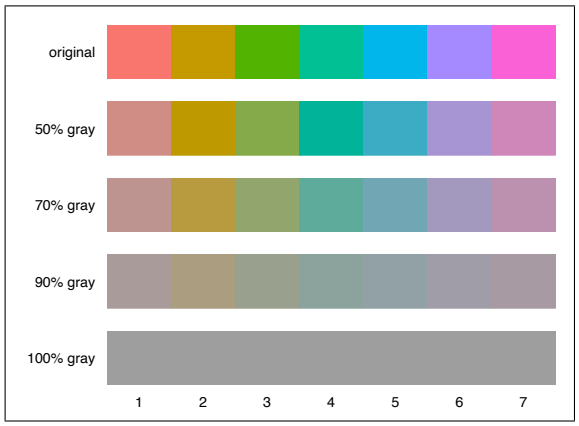
minance channels, respectively. Reasonable values are in a range of about ±50. The values specified in `intensify()` are intensity factors; typical values are between 0 (white) and about 10 (black).

3.5.2 Grayscale conversion

(new)

Option `gscale()` converts colors to shades of gray. Grayscale transformation works by reducing the chroma channel (colorfulness) towards zero. Here is an example, illustrating that colors from the `hue` generator will not be distinguishable in black and white print:

```
. colorpalette, n(7) labels(original
>     "50% gray" "70% gray"
>     "90% gray" "100% gray"):
>       hue
>     / hue, gscale(.5)
>     / hue, gscale(.7)
>     / hue, gscale(.9)
>     / hue, gscale
```

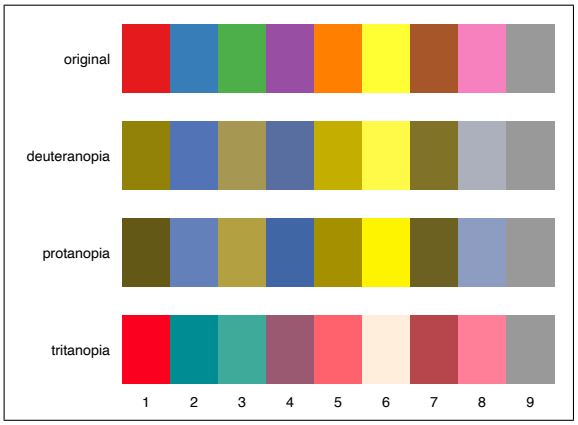


3.5.3 Color vision deficiency simulation

(new)

A substantial fraction of people suffer from color vision deficiency (CVD). Option `cblind()` can be used to simulate the three common types of CVD (deuteranomaly, protanomaly, and tritanomaly). Example:

```
. colorpalette, labels(original
>     deuteranopia protanopia
>     tritanopia):
>       Set1,
>     / Set1, cblind(1, deut)
>     / Set1, cblind(1, prot)
>     / Set1, cblind(1, trit)
```



`cblind(1)` requests simulation of full CVD severity; to simulate 50% CVD severity you could specify `cblind(0.5)`.

3.5.4 Analyze colors using colorcheck

(new)

The `colorcheck` command analyzes the colors returned by `colorpalette` by applying grayscale conversion and color vision deficiency transformation, and by computing minimum color differences among the converted colors. The purpose of the command is to evaluate whether the colors will be distinguishable by people who suffer from color vision deficiency and also whether the colors will be distinguishable in (non-color) print. The smallest noticeable difference between two colors has a color difference value (Delta E) of 1.0. A value of, say, 10 seems a reasonable minimum difference for colors used to illustrate different features of data.

The syntax of `colorcheck` is

`colorcheck` [`,` *options*]

where *options* are as follows.

`metric(`*metric*`)` selects the color difference metric to be used. *metric* can be `E76`, `E94`, `E2000`, or `Jab`; see the documentation of `ColrSpace` for details (Jann 2022). Default is `metric(Jab)`.

`mono(`[#] [`,` *method*]`)` determines the settings for grayscale conversion, where # in [0, 1] specifies the proportion of gray (default is `mono(1)`, i.e. full conversion) and *method* selects the conversion method. Default is `LCh`; see the documentation of `ColrSpace` for available methods (Jann 2022).

`cvd(`#`)`, with # in [0, 1], sets the severity of color vision deficiency. Default is `cvd(1)` (maximum severity).

`nograph` suppresses the graph.

`sort`[`(`*spec*`)`] sorts the colors in the graph, where *spec* may be `normal` (sort by hue of normal vision), `mono` (sort by shading of monochromacy vision), `deuter` (sort by hue of deuteranomaly vision), `prot` (sort by hue of protanomaly vision), or `trit` (sort by hue of tritanomaly vision). `sort` without argument is equivalent to `sort(normal)`. Sort only has an effect on how the colors are ordered in the graph, not on how they are stored in `r()`.

graph_options are graph options as for `colorpalette` in Syntax 2.

`colorcheck` requires results from `colorpalette` to be in memory and it adds (or updates) the following returns:

Scalars:

`r(mono)`	proportion of gray	`r(cvd)`	CVD severity

Macros:

`r(metric)`	color difference metric	`r(mono_method)`	grayscale conversion method
`r(p_mono)`	list of converted colors (grayscale)	`r(p_deut)`	list of converted colors (deteranomaly)
`r(p_prot)`	list of converted colors (protanomaly)	`r(p_trit)`	list of converted colors (tritanomaly)

Matrix:

`r(delta)`	color difference statistics

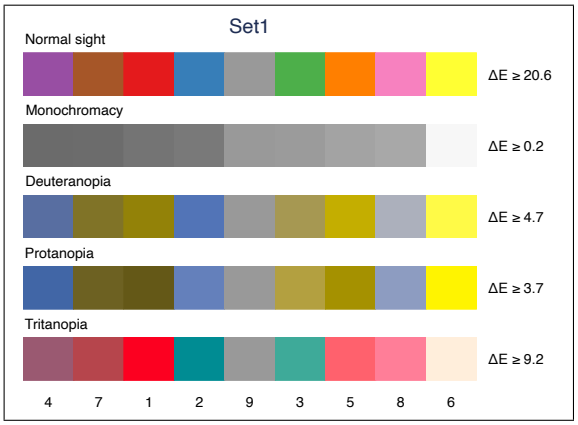
The general procedure is to obtain colors by running `colorpalette` and then apply `colorcheck` one or several times to analyze the colors. Example:

```
. colorpalette Set1, nograph
. colorcheck, nograph
                  Number of colors    =       9
                  N. of comparisons   =      36
                  CVD severity        =       1
                  Proportion of gray  =       1
                  Grayscale method    =     LCh
                  Difference metric   =     Jab

------------------------------------------------------------
      Delta E |    minimum    maximum       mean
--------------+---------------------------------------------
  normal sight |   20.59514   76.14433   43.95417
  monochromacy |   .2124847   50.00967    17.8777
deuteranomaly |   4.655909   72.64074   34.40387
  protanomaly |   3.744389   77.81498   37.00846
  tritanomaly |    9.20225   67.59568   37.00433
------------------------------------------------------------
```

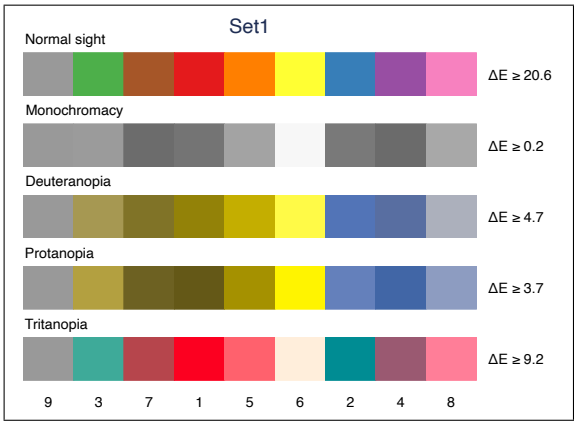
The minimum color difference under normal sight is about 20; all colors in the palette are sufficiently distinguishable. However, at least some of them cannot be distinguished after the colors have been transformed to gray. Also in the CVD scenarios the colors are substantially less distinct than under normal sight. To identify the problematic colors, the `sort()` option can be helpful. For example, specify `sort(mono)` to sort the colors by gray scale:

```
. colorcheck, sort(mono)
```



Likewise, specify `sort(deuter)` to sort the colors by hue under deuteranomaly vision:

```
. colorcheck, sort(deuter)
```



See en.wikipedia.org/wiki/Color_blindness for background information on color vision deficiency. See en.wikipedia.org/wiki/Color_difference for information on color differences.

3.6 Retrieve colors from a palette

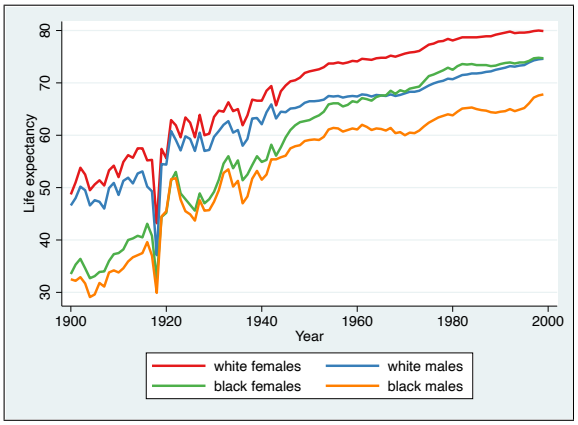
Under syntax 1, `colorpalette` returns the values of the colors in `r()` so that they can be used in a subsequent graph command. `r(p)` will contain a space separated list of all colors, `r(p1)`, `r(p2)`, etc. will contain the single colors one by one. Here is an example of the returns stored by `colorpalette` (option `nograph` is specified to prevent `colorpalette` from displaying the palette):

```
. colorpalette Set1, nograph
. return list
```

```
scalars:
                  r(n) =  9
macros:
              r(ptype) : "color"
              r(pname) : "Set1"
              r(pnote) : "categorical colors from colorbrewer2.org (Brewer e.."
            r(psource) : "http://www.personal.psu.edu/cab38/ColorBrewer/Colo.."
             r(pclass) : "qualitative"
                  r(p) : ""228 26 28" "55 126 184" "77 175 74" "152 78 163" .."
                 r(p9) : "153 153 153"
                 r(p8) : "247 129 191"
                 r(p7) : "166 86 40"
                 r(p6) : "255 255 51"
                 r(p5) : "255 127 0"
                 r(p4) : "152 78 163"
                 r(p3) : "77 175 74"
                 r(p2) : "55 126 184"
                 r(p1) : "228 26 28"
```

Options such as `select()` will affect the information stored in `r()`. That is, the stored information reflects the configuration of the palette after applying all palette options. Here is an example that selects four colors from ColorBrewer's `Set1` palette and uses them in a line plot:

```
. 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 `mycolors`. You could then use the colors in subsequent graph commands by typing `` `mycolors' ``.

An alternative is to make colors available as local macros, global macros, or system

colors by calling `colorpalette` with option `locals()`, `globals()`, or `stylefiles()`, respectively. See Section 4.2 and Section 4.3 for details. Yet another alternative is to use the `grstyle set` command to change the default colors used in Stata graphs; `grstyle set` calls `colorpalette` internally (see Jann 2018c).

3.7 Specify a custom list of colors

(revised)

Instead of selecting a named color palette you can also specify a custom list of colors using syntax

[(] *colorspec* [*colorspec* ...] [)]

where *colorspec* is

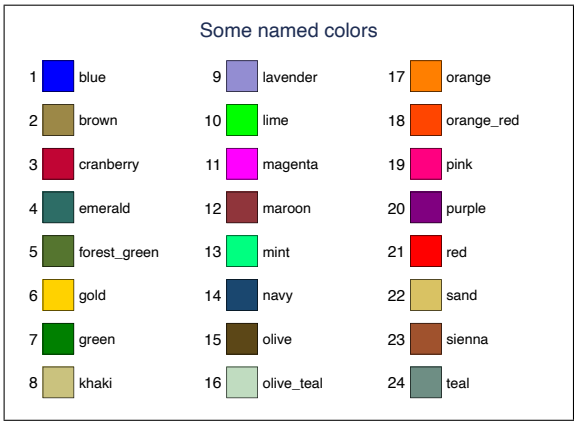
["] *color* [%#] [*#] ["]

Parentheses around the list may be used, if needed, to prevent name conflict with palette specifications. Color specifications containing spaces must be included in double quotes. Argument %# in *colorspec* sets the opacity (in percent; 0 = fully transparent, 100 = fully opaque; this requires Stata 15 or newer), *# adjusts the intensity (values between 0 and 1 make the color lighter; values larger than one make the color darker), and *color* is one of the following:

name	a color name; this includes official Stata's color names as listed in [G] ***colorstyle***, possible user additions provided through style files, as well as a large collection of named colors provided by `colorpalette` (see Section 4)
`#`*rrggbb*	6-digit hex RGB value; e.g., white = `#FFFFFF` or `#ffffff`, navy = `#1A476F` or `#1a476f`
`#`*rgb*	3-digit abbreviated hex RGB value; e.g., white = `#FFF` or `#fff`
# # #	RGB value in 0–255 scaling; e.g., navy = `"26 71 111"`
# # # #	CMYK value in 0–255 or 0–1 scaling; e.g., navy = `"85 40 0 144"` or `".333 .157 0 .565"`
cspace ...	color value in one of the color spaces supported by `ColrSpace`; e.g., navy = `"XYZ 5.55 5.87 15.9"` or `"Lab 29 -.4 -27.5"` or `"Jab 30.1 -8.9 -19"` (see Section 6.1 in Jann 2022 for details)

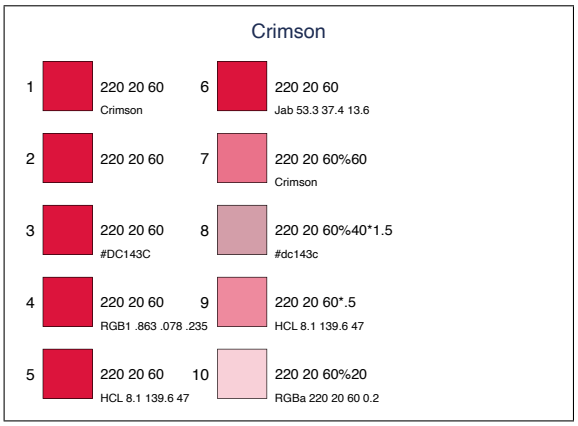
Here is an example displaying some of Stata's named colors (see [G] ***colorstyle***):

```
. 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)
```



For additional named colors provided by `colorpalette` see Section 4. Colors can also be specified as color codes in a variety of formats (see Section 6.1 in Jann 2022). `colorpalette` translates these colors to RGB. The following example illustrates some variants; the example also illustrates how to specify opacity and intensity operators:

```
. colorpalette
>     Crimson
>     "220 20 60"
>     #DC143C
>     "RGB1 .863 .078 .235"
>     "HCL 8.1 139.6 47"
>     "Jab 53.3 37.4 13.6"
>     Crimson%60
>     #dc143c%40*1.5
>     "HCL 8.1 139.6 47*0.5"
>     "RGBa 220 20 60 0.2"
>     , title(Crimson)
```



3.8 Provide custom palettes

(revised)

If you want to create a personal named color palette, you can define a program called `colorpalette_`*myname*, where *myname* is the name of your palette. Palette *myname* will then be available to `colorpalette` like any other palette. Your program should behave as follows.

1. It must return the color definitions as a comma-separated list in local macro `P`. All types of color specifications supported by `colorpalette`, including opacity and intensity operators, are allowed for the individual colors in the list (see Section 3.7).

2. If input is parsed using [P] **syntax**, option `n()` must be allowed. In addition to `n()`, all options not consumed by `colorpalette` will be passed through to `colorpalette_`*myname*. This makes it possible to support custom options in your program.

3. Color names can be returned as a comma-separated list in local macro `N`.

4. Color descriptions can be returned as a comma-separated list in local macro `I`.

5. The palette name can be returned in local macro `name` (*myname* is used as the palette name if no name is returned).

6. The palette class can be returned in local macro `class`.

7. A palette description can be returned in local macro `note`.

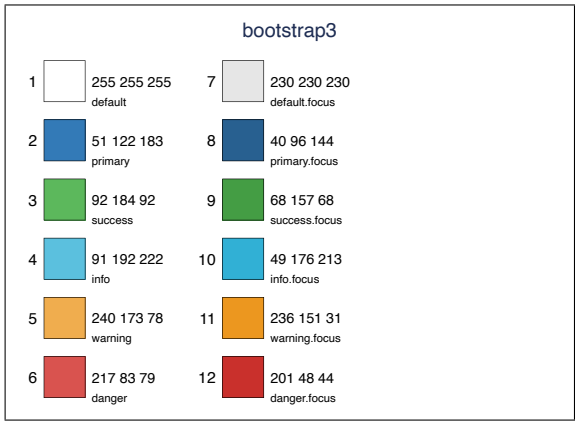
8. Information on the source of the palette can be returned in local macro `source`.

Here is an example providing a palette called `bootstrap3` containing semantic colors used for buttons in Bootstrap 3.3:

```
program colorpalette_bootstrap3
    syntax [, n(str) ] // n() not used
    c_local P      #ffffff,#337ab7,#5cb85c,#5bc0de,#f0ad4e,#d9534f, /*
                */ #e6e6e6,#286090,#449d44,#31b0d5,#ec971f,#c9302c
    c_local N      default,primary,success,   info,warning, danger, /*
                */ default.focus,primary.focus,success.focus,info.focus, /*
                */ warning.focus,danger.focus
    c_local class  qualitative
    c_local note   Button colors from Bootstrap 3.3
    c_local source https://getbootstrap.com/docs/3.3/
end
```

After defining the program, you can, for example, type:

```
. colorpalette bootstrap3, rows(6)
```



To make the new palette permanently available, store the program in file `colorpalette_`*myname*`.ado` in the working directory or somewhere along Stata's ado path (see [P] **sysdir**).

4 Named colors (new)

4.1 Overview

`colorpalette` supports a variety of named colors in addition to Stata's default colors documented in [G] ***colorstyle***. These additional colors are:

140 HTML colors

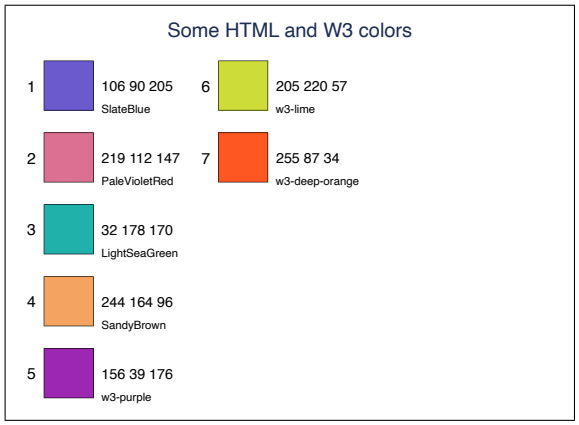
30 W3.CSS default colors

Further color collections from W3.CSS (using names as provided by W3.CSS): Flat UI Colors, Metro UI Colors, Windows 8 Colors, iOS Colors, US Highway Colors, US Safety Colors, European Signal Colors, Fashion Colors 2019, Fashion Colors 2018, Fashion Colors 2017, Vivid Colors, Food Colors, Camouflage Colors, ANA (Army Navy Aero) Colors, Traffic Colors

See Section 4.1.1 and Section 4.1.2 for an overview of the colors and their names. The names can be abbreviated and typed in lowercase letters. If abbreviation is ambiguous, the first matching name in the alphabetically ordered list will be used. In case of name conflict with a Stata color, the color from `colorpalette` will take precedence only if the specified name is an exact match including case. For example, `pink` will refer to official Stata's pink, whereas `Pink` will refer to HTML color pink.

To use the named colors in `colorpalette` simply type their names as you would for Stata's default colors. `colorpalette` will return the RGB codes of these colors that can then be used, for example, in a graph command (see Section 3.6). Example:

```
. colorpalette SlateBlue
>     PaleVioletRed LightSeaGreen
>     SandyBrown w3-purple w3-lime
>     w3-deep-orange,
>     title(Some HTML and W3 colors)
```



4.1.1 HTML colors

HTML pink

Pink LightPink HotPink DeepPink

PaleVioletRed MediumVioletRed

HTML purple

Lavender Thistle Plum Orchid

Violet Fuchsia Magenta MediumOrchid

DarkOrchid DarkViolet BlueViolet DarkMagenta

Purple MediumPurple MediumSlateBlue SlateBlue

DarkSlateBlue RebeccaPurple Indigo

HTML red

LightSalmon Salmon DarkSalmon LightCoral

IndianRed Crimson Red FireBrick

DarkRed

HTML orange

Orange DarkOrange Coral Tomato

OrangeRed

HTML yellow

Gold Yellow LightYellow LemonChiffon

LightGoldenRodYellow PapayaWhip Moccasin PeachPuff

PaleGoldenRod Khaki DarkKhaki

HTML green

GreenYellow Chartreuse LawnGreen Lime

LimeGreen PaleGreen LightGreen MediumSpringGreen

SpringGreen MediumSeaGreen SeaGreen ForestGreen

Green DarkGreen YellowGreen OliveDrab

DarkOliveGreen MediumAquaMarine DarkSeaGreen LightSeaGreen

DarkCyan Teal

HTML cyan

Aqua Cyan LightCyan PaleTurquoise

Aquamarine Turquoise MediumTurquoise DarkTurquoise

HTML blue

CadetBlue SteelBlue LightSteelBlue LightBlue

PowderBlue LightSkyBlue SkyBlue CornflowerBlue

DeepSkyBlue DodgerBlue RoyalBlue Blue

MediumBlue DarkBlue Navy MidnightBlue

HTML brown

Cornsilk BlanchedAlmond Bisque NavajoWhite

Wheat BurlyWood Tan RosyBrown

SandyBrown GoldenRod DarkGoldenRod Peru

Chocolate Olive SaddleBrown Sienna

Brown Maroon

HTML white

White Snow HoneyDew MintCream

Azure AliceBlue GhostWhite WhiteSmoke

SeaShell Beige OldLace FloralWhite

Ivory AntiqueWhite Linen LavenderBlush

MistyRose

HTML gray

Gainsboro LightGray Silver DarkGray

DimGray Gray LightSlateGray SlateGray

DarkSlateGray Black

HTML grey

Gainsboro	LightGrey	Silver	DarkGrey
DimGrey	Grey	LightSlateGrey	SlateGrey
DarkSlateGrey	Black		

4.1.2 W3.CSS colors

w3 default

w3-red	w3-pink	w3-purple	w3-deep-purple
w3-indigo	w3-blue	w3-light-blue	w3-cyan
w3-aqua	w3-teal	w3-green	w3-light-green
w3-lime	w3-sand	w3-khaki	w3-yellow
w3-amber	w3-orange	w3-deep-orange	w3-blue-grey
w3-brown	w3-light-grey	w3-grey	w3-dark-grey
w3-black	w3-white	w3-pale-red	w3-pale-yellow
w3-pale-green	w3-pale-blue		

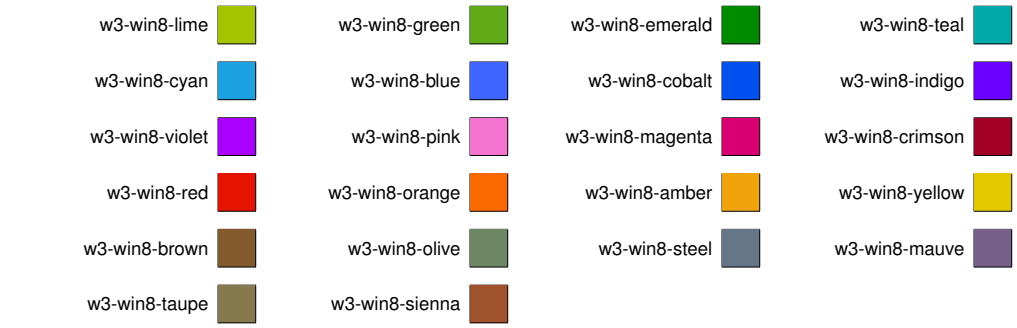
w3 flat

w3-flat-turquoise	w3-flat-emerald	w3-flat-peter-river	w3-flat-amethyst
w3-flat-wet-asphalt	w3-flat-green-sea	w3-flat-nephritis	w3-flat-belize-hole
w3-flat-wisteria	w3-flat-midnight-blue	w3-flat-sun-flower	w3-flat-carrot
w3-flat-alizarin	w3-flat-clouds	w3-flat-concrete	w3-flat-orange
w3-flat-pumpkin	w3-flat-pomegranate	w3-flat-silver	w3-flat-asbestos

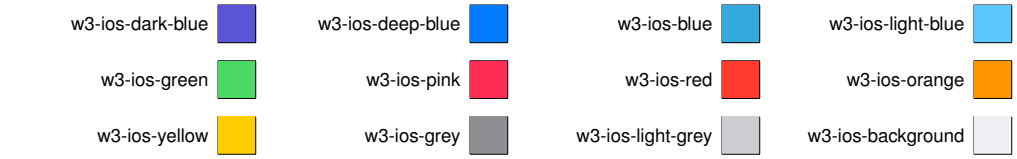
w3 metro

w3-metro-light-green	w3-metro-green	w3-metro-dark-green	w3-metro-magenta
w3-metro-light-purple	w3-metro-purple	w3-metro-dark-purple	w3-metro-darken
w3-metro-teal	w3-metro-light-blue	w3-metro-blue	w3-metro-dark-blue
w3-metro-yellow	w3-metro-orange	w3-metro-dark-orange	w3-metro-red
w3-metro-dark-red			

w3 win8



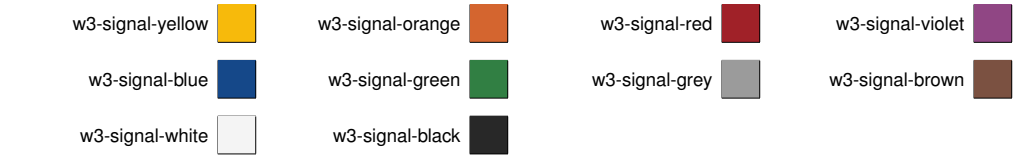
w3 ios



w3 highway

w3 safety

w3 signal



w3 2019

w3-2019-rocky-road
w3-2019-fruit-dove
w3-2019-sugar-almond
w3-2019-dark-cheddar
w3-2019-galaxy-blue
w3-2019-bluestone
w3-2019-orange-tiger
w3-2019-eden
w3-2019-vanilla-custard
w3-2019-evening-blue
w3-2019-paloma
w3-2019-guacamole

w3 2018

w3-2018-red-pear
w3-2018-valiant-poppy
w3-2018-nebulas-blue
w3-2018-ceylon-yellow
w3-2018-martini-olive
w3-2018-russet-orange
w3-2018-crocus-petal
w3-2018-limelight
w3-2018-quetzal-green
w3-2018-sargasso-sea
w3-2018-tofu
w3-2018-almond-buff
w3-2018-quiet-gray
w3-2018-meerkat
w3-2018-meadowlark
w3-2018-cherry-tomato
w3-2018-little-boy-blue
w3-2018-chili-oil
w3-2018-pink-lavender
w3-2018-blooming-dahlia
w3-2018-arcadia
w3-2018-emperador
w3-2018-ultra-violet
w3-2018-almost-mauve
w3-2018-spring-crocus
w3-2018-lime-punch
w3-2018-sailor-blue
w3-2018-harbor-mist
w3-2018-warm-sand
w3-2018-coconut-milk

w3 2017

w3-2017-greenery
w3-2017-grenadine
w3-2017-tawny-port
w3-2017-ballet-slipper
w3-2017-butterum
w3-2017-navy-peony
w3-2017-neutral-gray
w3-2017-shaded-spruce
w3-2017-golden-lime
w3-2017-marina
w3-2017-autumn-maple
w3-2017-niagara
w3-2017-primrose-yellow
w3-2017-lapis-blue
w3-2017-flame
w3-2017-island-paradise
w3-2017-pale-dogwood
w3-2017-pink-yarrow
w3-2017-kale
w3-2017-hazelnut

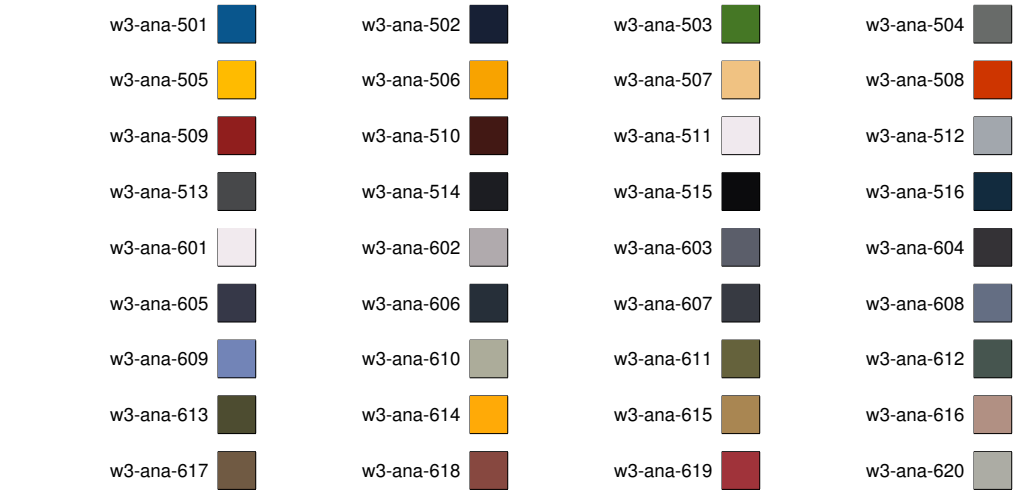
w3 vivid

w3-vivid-pink
w3-vivid-red
w3-vivid-orange
w3-vivid-yellow
w3-vivid-green
w3-vivid-blue
w3-vivid-black
w3-vivid-white
w3-vivid-purple
w3-vivid-purple
w3-vivid-yellowish-pink
w3-vivid-reddish-orange
w3-vivid-orange-yellow
w3-vivid-greenish-yellow
w3-vivid-yellow-green
w3-vivid-yellowish-green
w3-vivid-bluish-green
w3-vivid-greenish-blue
w3-vivid-purplish-blue
w3-vivid-reddish-purple
w3-vivid-purplish-red

w3 food

w3 camo

w3 ana



4.2 Make colors available as globals or locals

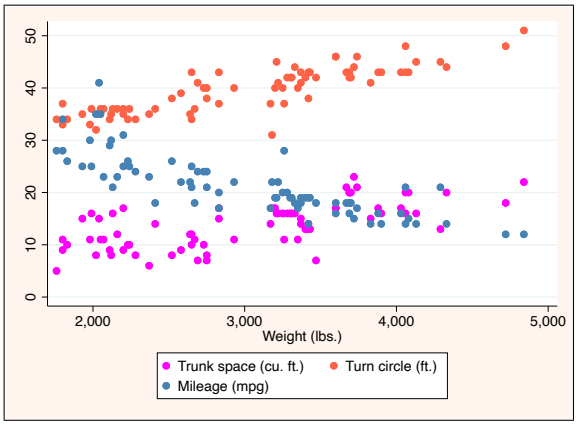
(new)

Instead of retrieving color codes from `r()` as described in Section 3.6, you can also directly store the colors returned by `colorpalette` as local or global macros by applying option `locals()` or `globals()`, respectively. These local or global macros can then be used in subsequent graph commands to address the colors. Here is an example that makes some HTML colors available as global macros (you could make all 140 HTML colors available by typing `colorpalette HTML, globals`):

```
. colorpalette Fuchsia Tomato SteelBlue SeaShell, globals
globals:
                Fuchsia : "255 0 255"
                 Tomato : "255 99 71"
              SteelBlue : "70 130 180"
               SeaShell : "255 245 238"
```

After that, type $*name* to select a color, where *name* is the name of the global:

```
. sysuse auto, clear
(1978 automobile data)
. scatter trunk turn mpg weight,
>     mc($Fuchsia $Tomato $SteelBlue)
>     graphr(color($SeaShell))
```

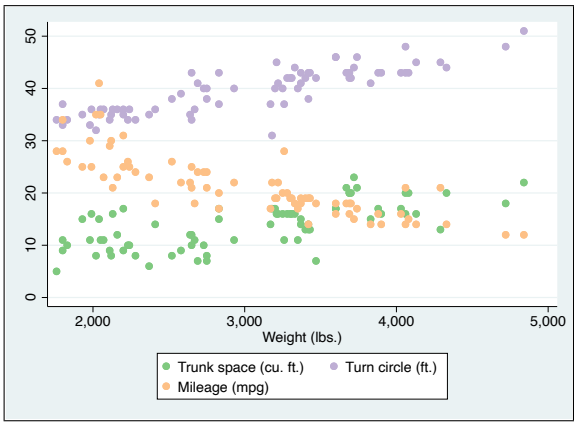


The defined globals will remain in memory until you restart Stata or until you drop them using command `macro drop`.

Note that storing colors as macros will also work with colors that have no names;

`colorpalette` will then make up names using the color index (e.g., `$p1`, `$p2`, etc.; see Section 3.1.2 for details on naming conventions). Furthermore, depending on context (e.g., within a do-file or program), it may be more convenient to make colors available as local macros instead of global macros. An example is as follows:

```
. colorpalette Accent, locals
locals:
                    1 : "127 201 127"
                    2 : "190 174 212"
                    3 : "253 192 134"
                    4 : "255 255 153"
                    5 : "56 108 176"
                    6 : "240 2 127"
                    7 : "191 91 23"
                    8 : "102 102 102"
. scatter trunk turn mpg weight,
>     mc(`1´ `2´ `3´)
```



The defined locals will remain in memory until the do-file or program concludes.

4.3 Make colors permanently available (new)

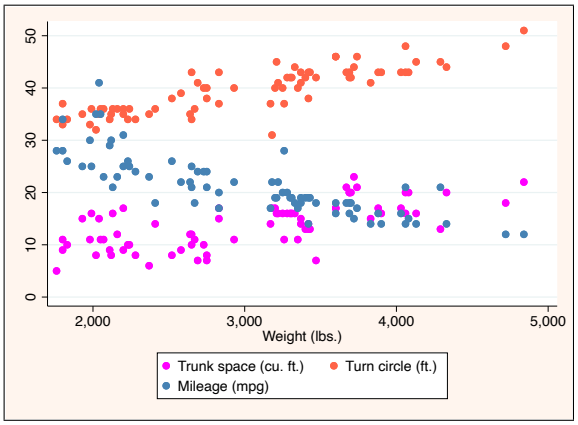
Colors can also be made available permanently by applying the `stylefiles()` option. Option `stylefiles()` will cause RGB color definitions to be stored in style files on disk (one file for each color), from where Stata will read the color definitions:[1]

```
. colorpalette Fuchsia Tomato SteelBlue SeaShell, stylefiles
directory style does not exist
press any key to create the directory, or Break to abort
color styles:
                Fuchsia : "255 0 255"
                 Tomato : "255 99 71"
              SteelBlue : "70 130 180"
               SeaShell : "255 245 238"
(style files written to directory style)
. discard // flush working memory to make the new colors available
```

The color names can then be used just like official Stata's color names.

[1] If the graph system has already been loaded, that is, if you already produced a graph in the current session, you will need to clear the graph memory before the stored colors become available; use [P] **discard** or [D] **clear all** to flush the working memory.

```
. sysuse auto, clear
(1978 automobile data)
. scatter trunk turn mpg weight,
>     mc(Fuchsia Tomato SteelBlue)
>     graphr(color(SeaShell))
```



By default, `colorpalette` will store the style files in folder "style" in the current working directory:

```
. dir style
total 32
-rw-r--r--  1 jann  staff  19 Apr 25 18:22 color-Fuchsia.style
-rw-r--r--  1 jann  staff  21 Apr 25 18:22 color-SeaShell.style
-rw-r--r--  1 jann  staff  20 Apr 25 18:22 color-SteelBlue.style
-rw-r--r--  1 jann  staff  19 Apr 25 18:22 color-Tomato.style
```

This means that the color definitions will be found by Stata as long as you do not change the working directory. To make the colors permanently available irrespective of the working directory, type `stylefiles, personal`. In this case the style files will be stored in folder "style" within the `PERSONAL` ado-file directory; see help [P] **sysdir**.

5 Palettes, colormaps, and color generators

This section provides an overview of the named palettes implemented in `colorpalette`. There are three types of palettes: (1) standard palettes defined as a fixed set of colors (or several fixed sets of varying size);[2] (2) colormaps that obtain color gradients by linear interpolation from a dense grid of RGB values or by linear segmentation between given RGB anchor points; (3) color generators that construct colors based on specific equations and parameter settings.

Full palette names are given below, some of them including uppercase letters. Note that the names can be abbreviated and typed in lowercase letters when calling the palettes in `colorpalette` (for example, "`BuGn`" could be typed as "`bugn`", "`lin carcolor algorithm`" could be typed as "`lin car a`"). If abbreviation is ambiguous, the first matching name in the sorted list of all predefined palettes, colormaps, and color

[2] However, be aware that `colorpalette` automatically applies interpolation or recycling if the number of requested colors is larger than the (maximum) number of colors provided by the palette. Specify option `noexpand` to prevent this behavior.

generators is used.

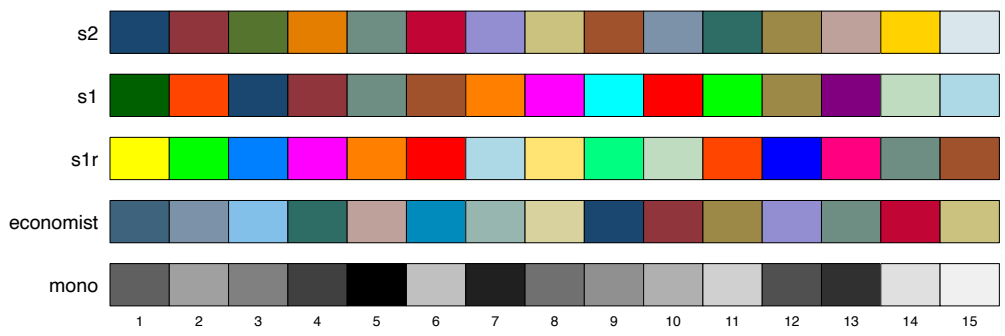
5.1 Palettes

5.1.1 Stata palettes

The Stata palettes are named after the graphics scheme (see [G] **Schemes intro**) 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 palettes look as follows.



5.1.2 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.

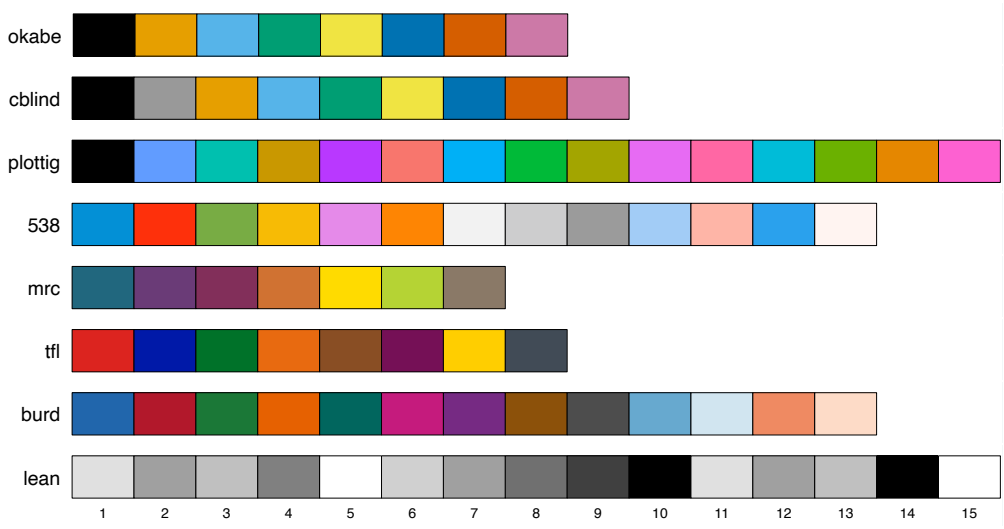
`okabe`	8 colorblind-friendly colors suggested by Okabe and Ito (2002); these colors are used in schemes by Bischof (2017b).
`cblind`	Like `okabe`, but including an additional gray as suggested at www.cookbook-r.com. The same colors are also used (in different order and using `gs10` for gray) in the `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.
`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 palettes look as follows.



5.1.3 D3.js palettes

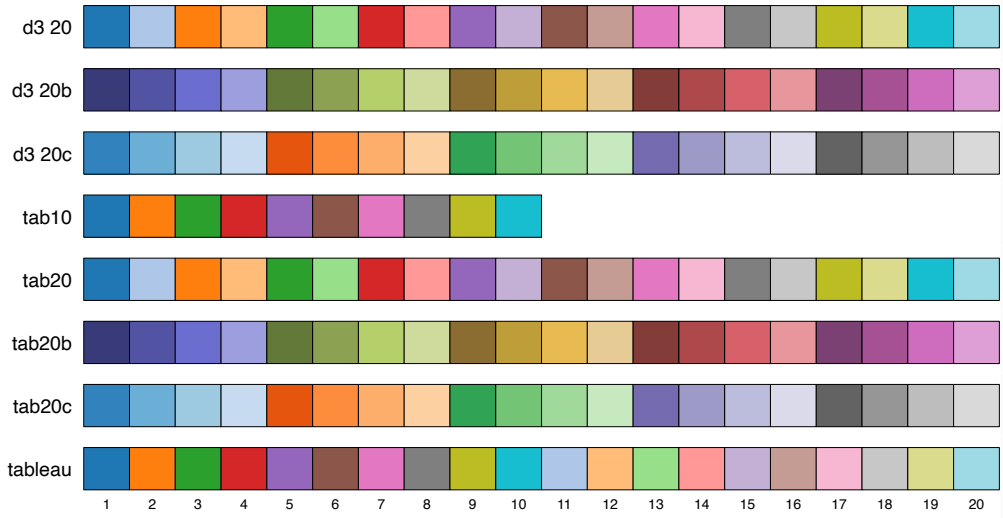
The `d3` collection provides color schemes from d3js.org, using the color values found at GitHub. The syntax is

`d3` [*scheme*] [`,` *palette_options*]

where *scheme* is one of the following:

`10` 10 categorical colors; the default
`20` 20 categorical colors in pairs
`20b` 20 categorical colors in groups of four
`20c` 20 categorical colors in groups of four

These colors appear to be colors that have been used in earlier versions of Tableau. The four palettes are thus also available as `tab10`, `tab20`, `tab20b`, and `tab20c`. A further variant on `d3 20` is palette `tableau` (same colors but in different order), which has been obtained from Lin et al. (2013). The palettes look as follows. (new)



5.1.4 Qualitative palettes from seaborn

(new)

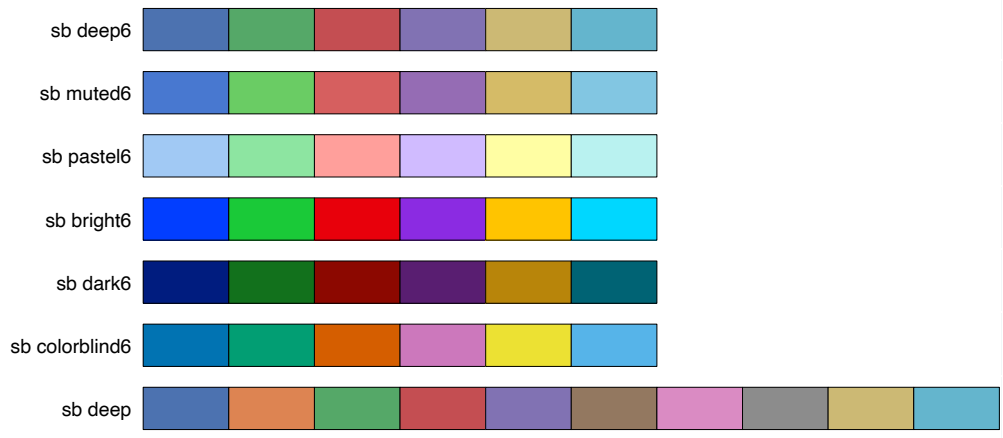
The `sb` collection provides categorical color schemes from seaborn.pydata.org (same basic colors as `tab10`, but in different tones). The syntax is

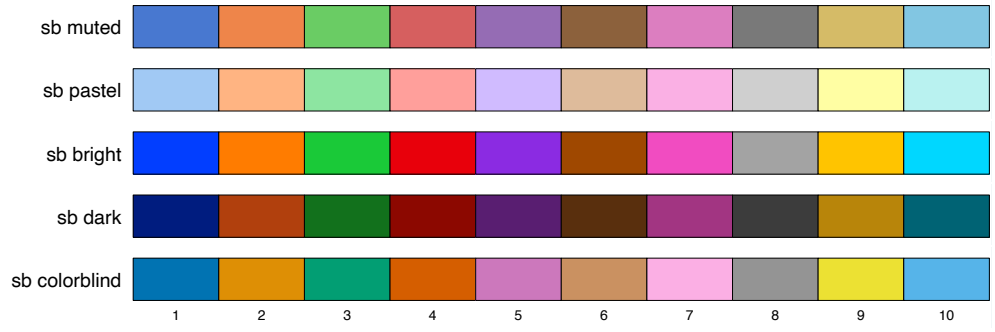
`sb` [*scheme*] [`,` *palette_options*]

where *scheme* is one of the following:

10 colors: `muted` (the default), `pastel`, `bright`, `dark`, `colorblind`
6 colors: `muted6`, `pastel6`, `bright6`, `dark6`, `colorblind6`

Palette `sb muted6` is also available as `sb6`. The palettes look as follows.





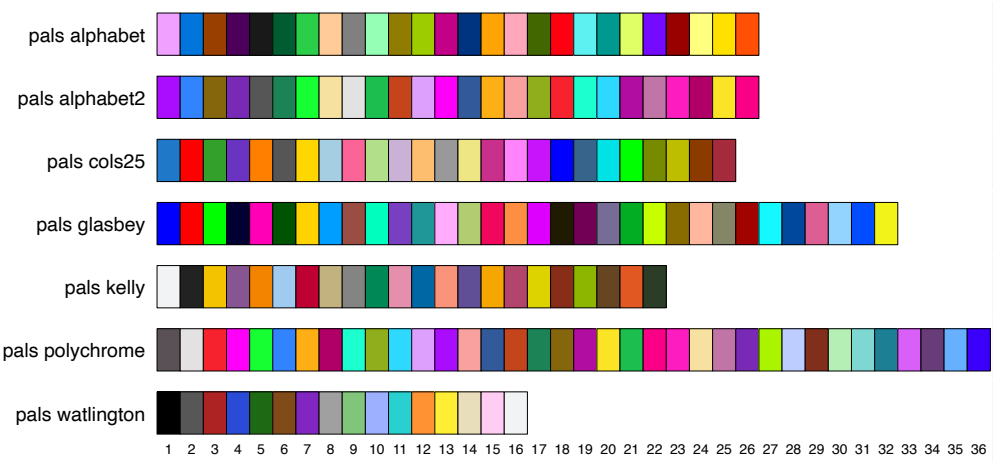
5.1.5 Categorical palettes from pals

(new)

The `pals` collection provides categorical color schemes that have been obtained from the `pals` package in R; see github.com/kwstat/pals. The syntax is

`pals` [*scheme*] [, *palette_options*]

where *scheme* is one of `alphabet` (26 colors), `alphabet2` (26 colors), `cols25` (25 colors), `glasbey` (32 colors), `kelly` (22 colors; the default), `polychrome` (36 colors), and `watlington` (16 colors). The palettes look as follows.



5.1.6 Tableau 10 color schemes

(new)

The `tab` collection provides various color schemes from Tableau 10 (the color values have been obtained from the `ggthemes` package in R). The syntax is

`tab` [*scheme*] [, *palette_options*]

where *scheme* is one of the following:

Qualitative
: `10` (default), `20`, `Color Blind` (10 colors), `Seattle Grays` (5 colors), `Traffic` (9 colors), `Miller Stone` (11 colors), `Superfishel Stone` (10 colors), `Nuriel Stone` (9 colors), `Jewel Bright` (9 colors), `Summer` (8 colors), `Winter` (10 colors), `Green-Orange-Teal` (12 colors), `Red-Blue-Brown` (12 colors), `Purple-Pink-Gray` (12 colors), `Hue Circle` (19 colors)

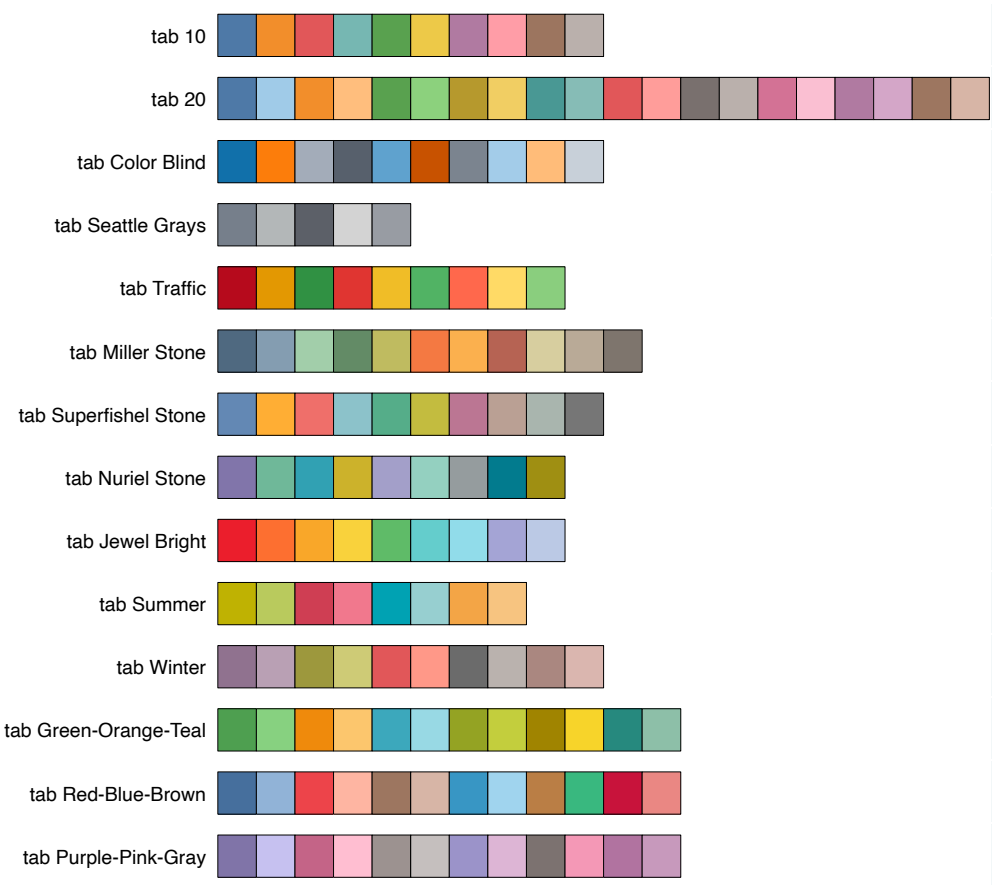
Sequential (7 colors)
: `Blue-Green`, `Blue Light`, `Orange Light`

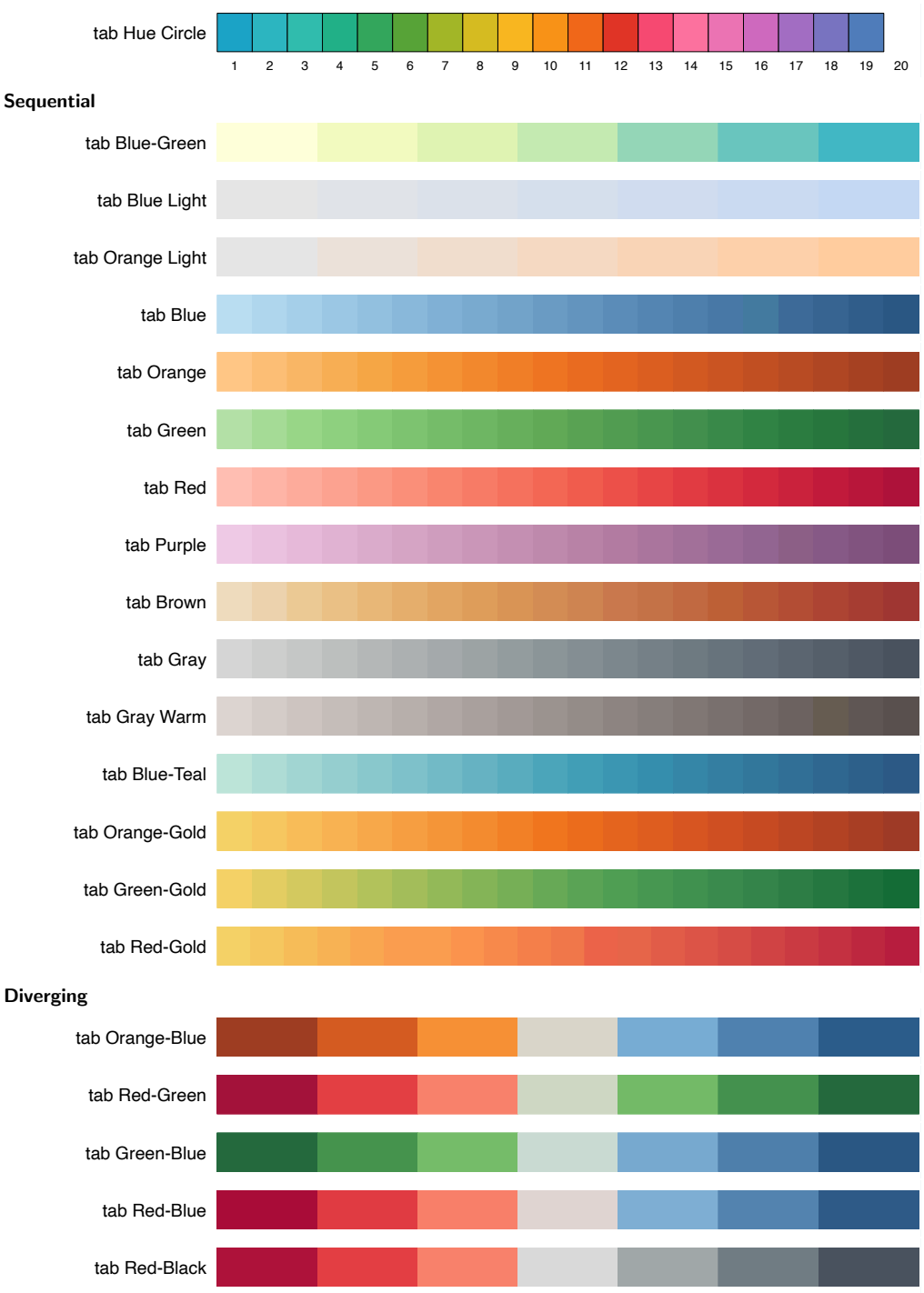
Sequential (20 colors)
: `Blue`, `Orange`, `Green`, `Red`, `Purple`, `Brown`, `Gray`, `Gray Warm`, `Blue-Teal`, `Orange-Gold`, `Green-Gold`, `Red-Gold` (21 colors)

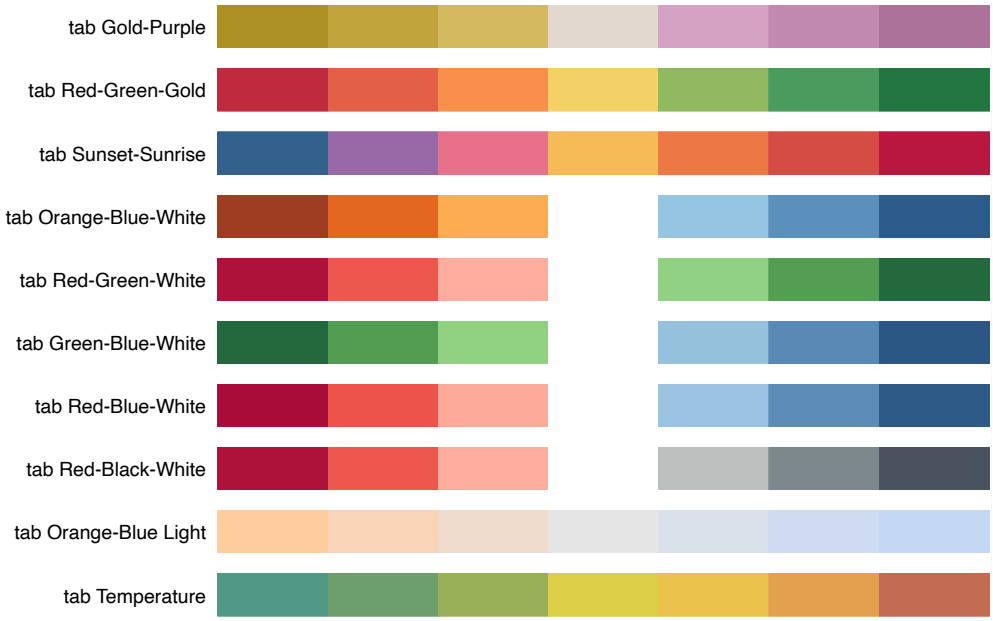
Diverging (7 colors)
: `Orange-Blue`, `Red-Green`, `Green-Blue`, `Red-Blue`, `Red-Black`, `Gold-Purple`, `Red-Green-Gold`, `Sunset-Sunrise`, `Orange-Blue-White`, `Red-Green-White`, `Green-Blue-White`, `Red-Blue-White`, `Red-Black-White`, `Orange-Blue Light`, `Temperature`

The palettes look as follows.

Qualitative




tab Hue Circle
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Sequential
tab Blue-Green
tab Blue Light
tab Orange Light
tab Blue
tab Orange
tab Green
tab Red
tab Purple
tab Brown
tab Gray
tab Gray Warm
tab Blue-Teal
tab Orange-Gold
tab Green-Gold
tab Red-Gold
Diverging
tab Orange-Blue
tab Red-Green
tab Green-Blue
tab Red-Blue
tab Red-Black



5.1.7 Color schemes by Paul Tol

New palettes (new)

The `tol` collection provides various color schemes presented by Paul Tol at personal.sron.nl/~pault. The syntax is

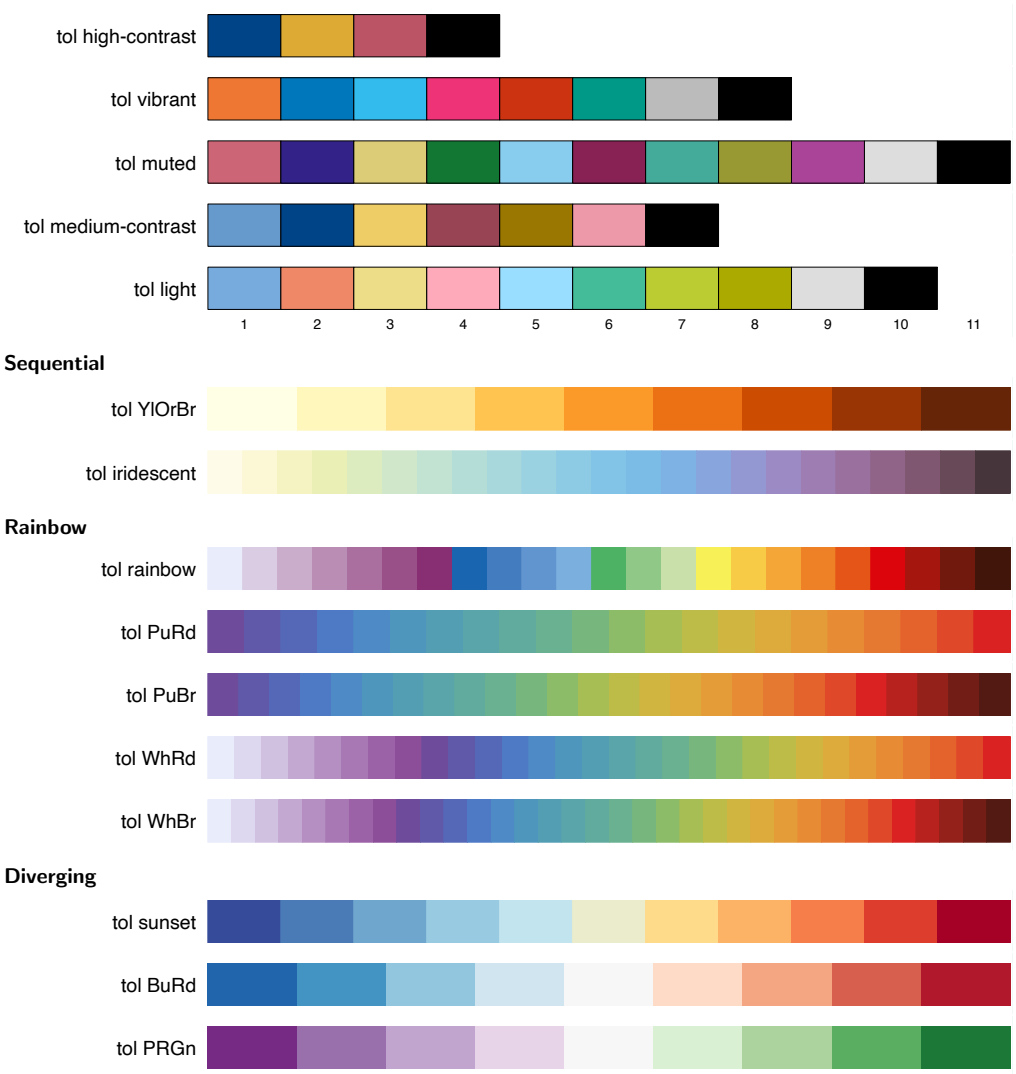
`tol` [*scheme*] [`,` *palette_options*]

where *scheme* is one of the following:

Qualitative:	`bright` (8 colors), `high-contrast` (4 colors), `vibrant` (8 colors), `muted` (11 colors; the default), `medium-contrast` (7 colors), `light` (10 colors)
Sequential:	`YlOrBr` (9 colors), `iridescent` (23 colors)
Rainbow:	`rainbow` (1–23 colors), `PuRd` (22 colors), `PuBr` (26 colors), `WhRd` (30 colors), `WhBr` (34 colors)
Diverging:	`sunset` (11 colors), `BuRd` (9 colors), `PRGn` (9 colors)

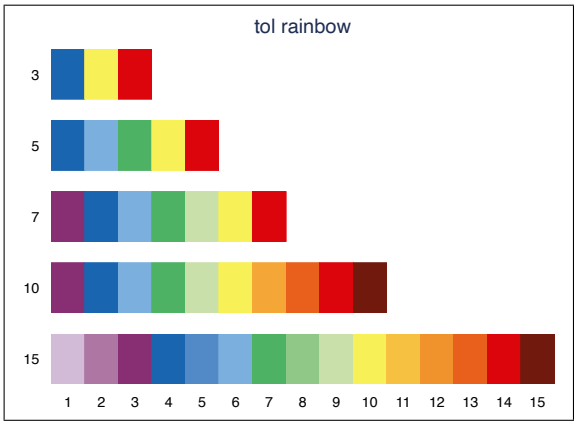
The definitions of the schemes have been obtained from source file tol_colors.py. These definitions may deviate from how the palettes are presented at personal.sron.nl/~pault (e.g., with respect to the order of colors in the qualitative schemes). The palettes look as follows.

Qualitative



For `tol rainbow`, the selection of colors depends on the size of the palette (1 to 23 colors), as illustrated in the following example:

```
. colorpalette, labels(3 5 7 10 15)
>     title(tol rainbow):
>       tol rainbow, n(3)
>     / tol rainbow, n(5)
>     / tol rainbow, n(7)
>     / tol rainbow, n(10)
>     / tol rainbow, n(15)
```



Palettes from Tol (2012)

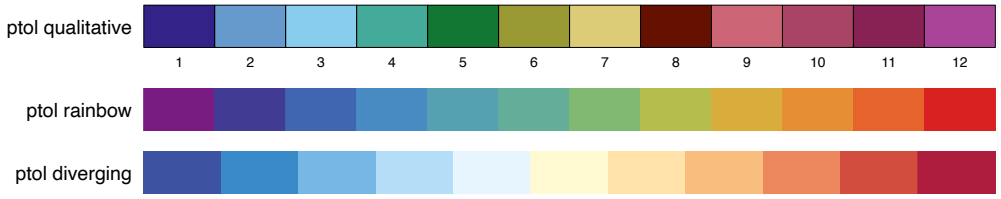
The `ptol` collection provides color schemes as suggested by Tol (2012). The syntax is

`ptol` [*scheme*] [, *palette_options*]

where *scheme* is one of the following:

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

The palettes look as follows (using the maximum number of colors).



5.1.8 ColorBrewer palettes

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.[3] The syntax for the ColorBrewer palettes is

[3]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).

scheme [cmyk] [, *palette_options*]

where *scheme* is one of the following:

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)

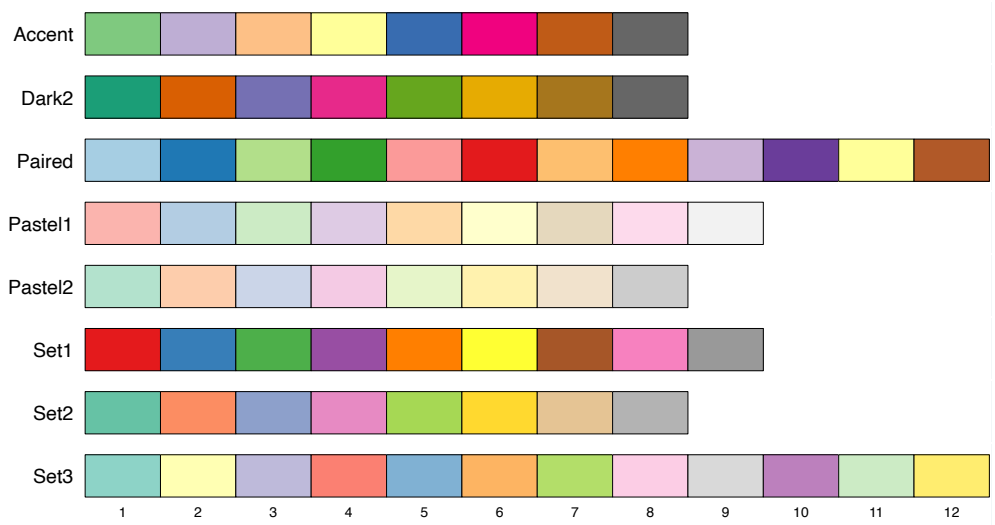
`BuGn`	light blue to green	`BuPu`	light blue to purple
`GnBu`	light green to blue	`OrRd`	light orange to red
`PuBu`	light purple to blue	`PuBuGn`	light purple over blue to green
`PuRd`	light purple to red	`RdPu`	light red to purple
`YlGn`	light yellow to green	`YlGnBu`	light yellow over green to blue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

Diverging schemes (3–11 colors)

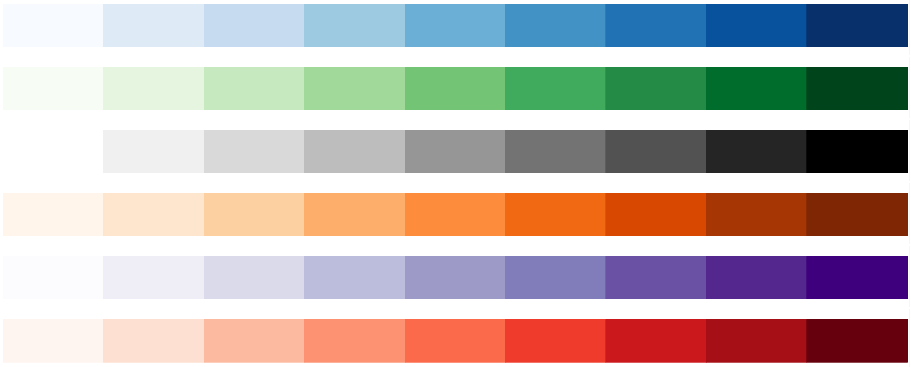
`BrBG`	brown to green, light gray mid	`PiYG`	pink to green, light gray mid
`PRGn`	purple to green, light gray mid	`PuOr`	purple to orange, light gray mid
`RdBu`	red to blue, light gray mid	`RdGy`	red to gray, white mid
`RdYlBu`	red to blue, yellow mid	`RdYlGn`	red to green, yellow mid
`Spectral`	red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

Argument `cmyk` selects the CMYK variant; the default is to use the RGB variant. The palettes look as follows (using the maximum number of colors for those schemes that come in different sizes).

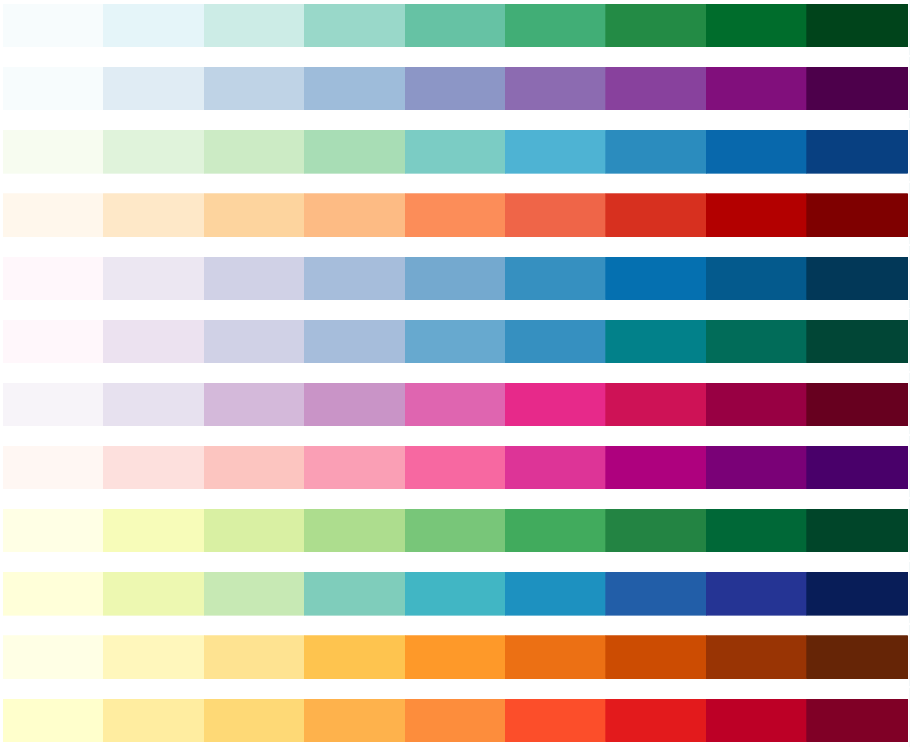
Qualitative



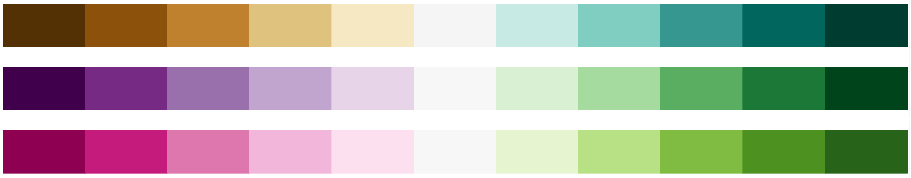
Sequential (single hue)

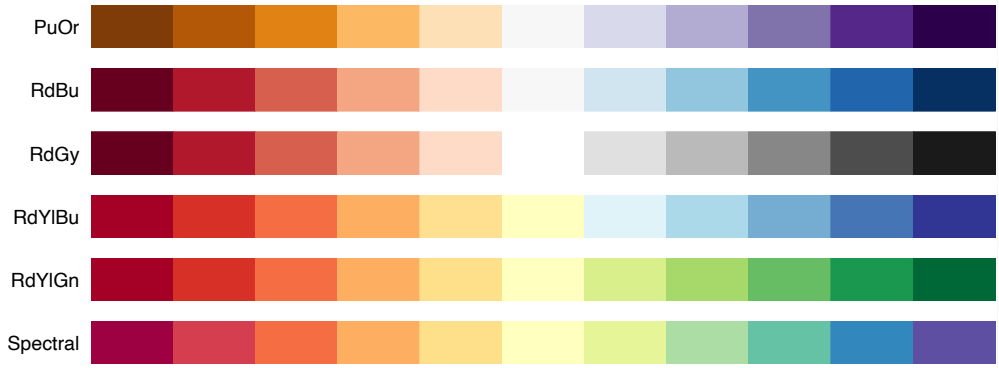


Sequential (multi-hue)



Diverging





5.1.9 Color schemes from Carto

(new)

The `carto` collection provides various color schemes from Carto (using colors codes from cartocolor.js at github.com/CartoDB). The syntax is

`carto` *scheme* [`,` *palette_options*]

where *scheme* is one of the following:

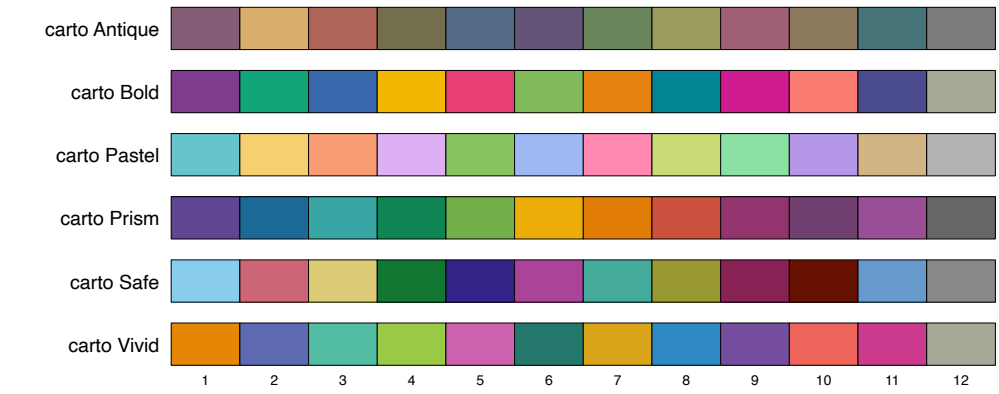
Qualitative (2–11 colors)
: `Antique`, `Bold` (the default), `Pastel`, `Prism`, `Safe` (colorblind-friendly), `Vivid`

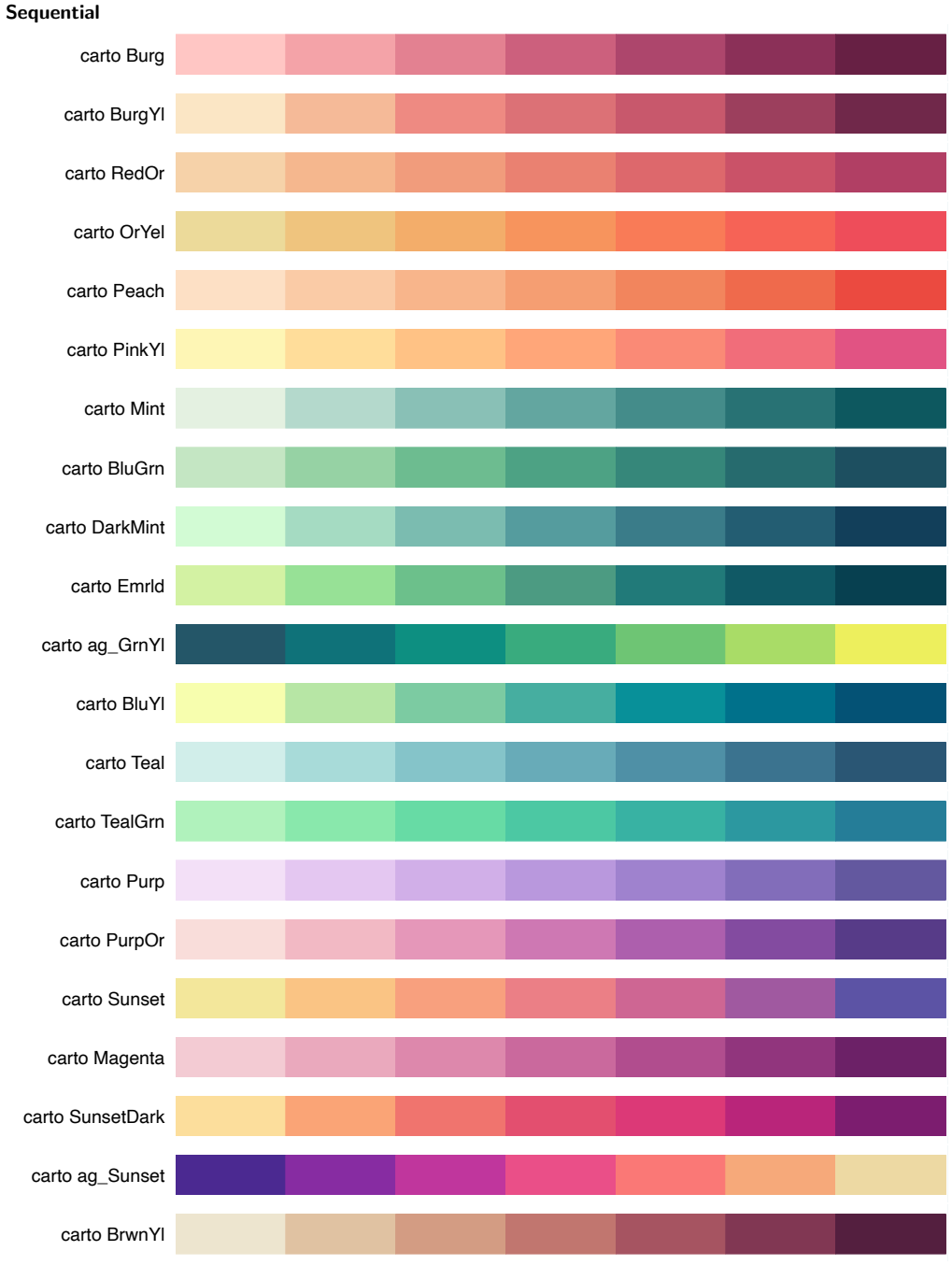
Sequential (2–7 colors)
: `Burg`, `BurgYl`, `RedOr`, `OrYel`, `Peach`, `PinkYl`, `Mint`, `BluGrn`, `DarkMint`, `Emrld`, `ag_GrnYl`, `BluYl`, `Teal`, `TealGrn`, `Purp`, `PurpOr`, `Sunset`, `Magenta`, `SunsetDark`, `ag_Sunset`, `BrwnYl`

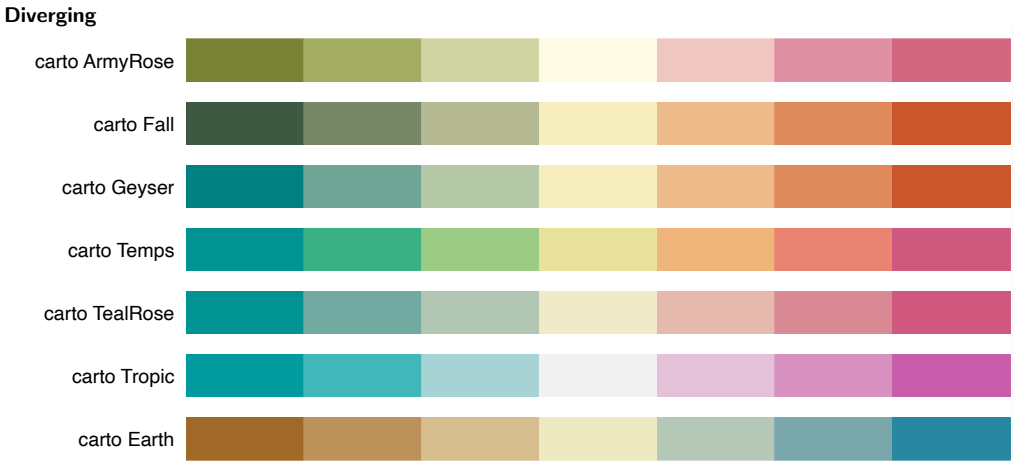
Diverging (2–7 colors)
: `ArmyRose`, `Fall`, `Geyser`, `Temps`, `TealRose`, `Tropic`, `Earth`

The palettes look as follows (using the maximum number of colors for those schemes that come in different sizes).

Qualitative




Sequential
carto Burg
carto BurgYl
carto RedOr
carto OrYel
carto Peach
carto PinkYl
carto Mint
carto BluGrn
carto DarkMint
carto Emrld
carto ag_GrnYl
carto BluYl
carto Teal
carto TealGrn
carto Purp
carto PurpOr
carto Sunset
carto Magenta
carto SunsetDark
carto ag_Sunset
carto BrwnYl



5.1.10 Semantic colors by Lin et al.

The `lin` collection provides semantic color schemes suggested by Lin et al. (2013).[4] The syntax is

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

where *scheme* is one of the following:

`carcolor`	6 car colors; the default	`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

Argument `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`). Figure 1 display the schemes including labels ("T" stands for "Turkers", "E" for "Expert", "A" for "Algorithm").

5.1.11 Colors schemes from spmap

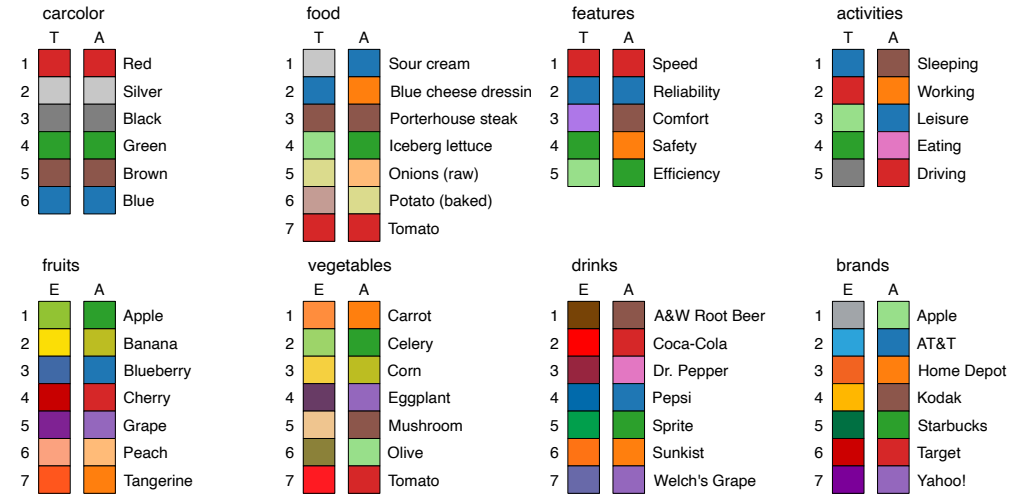
The `spmap` collection provides color schemes from the `spmap` package by Pisati (2007). The implementation is based on code from `spmap_color.ado` (version 1.3.0, 13 March 2017). The syntax is:

`spmap` [*scheme*] [`,` *palette_options*]

where *scheme* is one of the following:

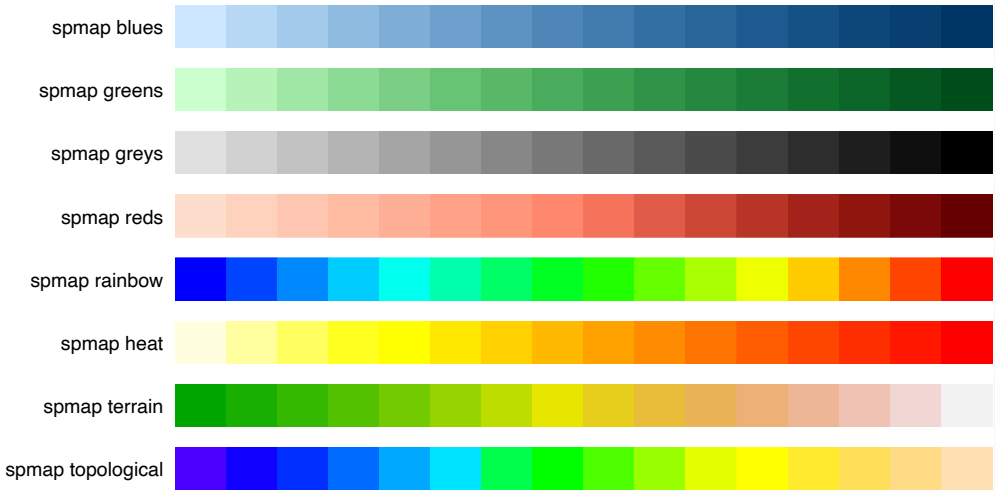
[4] The values of the semantic colors have been taken from the source code of the `brewscheme` package by Buchanan (2015) (`brewextra.ado`, version 1.0.0, 21 March 2016).

Figure 1: Semantic color schemes by Lin et al. (2013)



`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

The palettes look as follows (using 16 colors).



5.1.12 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 *scheme* is one of the following:

Qualitative

`languages`	colors used for languages	`parties`	colors used for Swiss parties

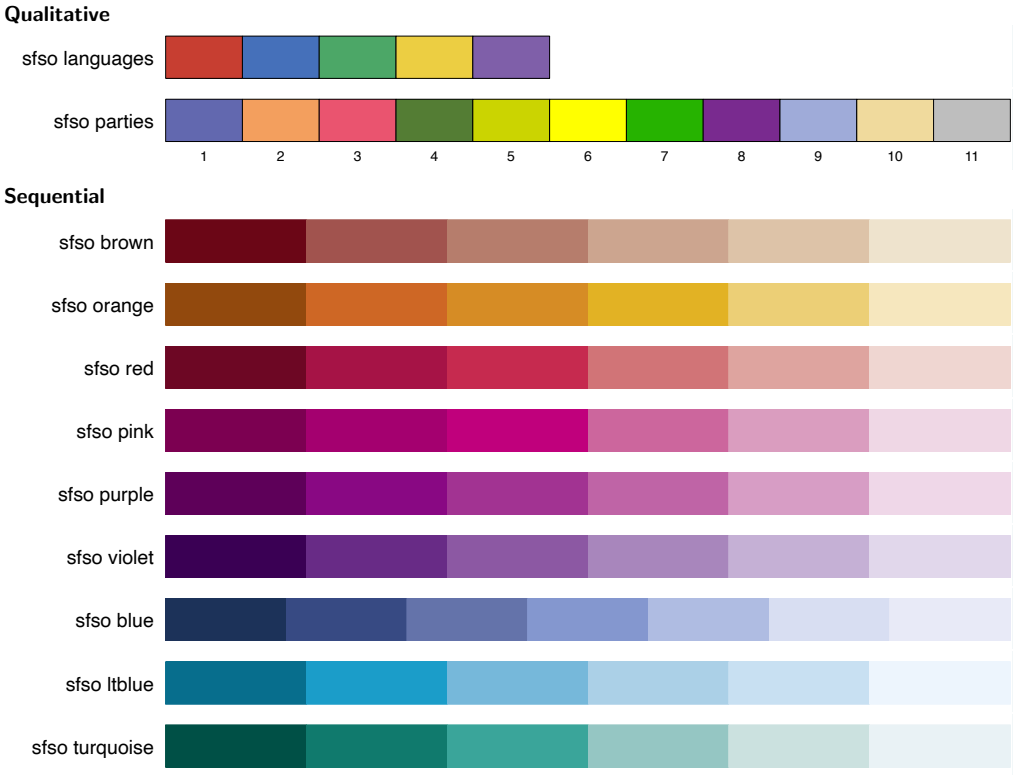
Sequential

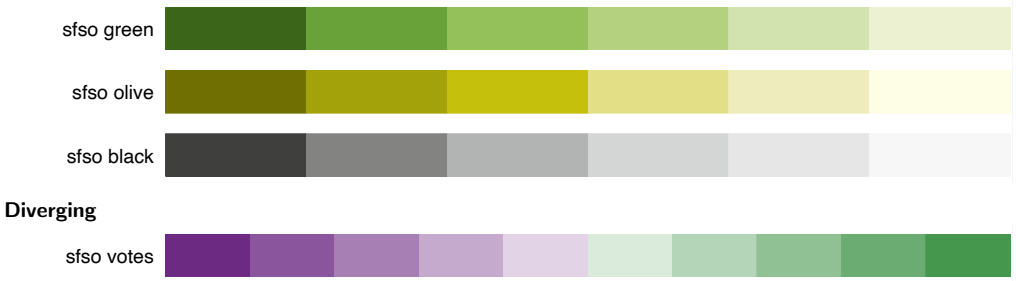
`brown`	dark brown to light brown	`orange`	dark orange to light orange
`red`	dark red to light red	`pink`	dark pink to light pink
`purple`	dark purple to light purple	`violet`	dark violet to light violet
`blue`	dark blue to light blue; the default	`ltblue`	lighter version of `blue`
`turquoise`	dark turquoise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

Diverging

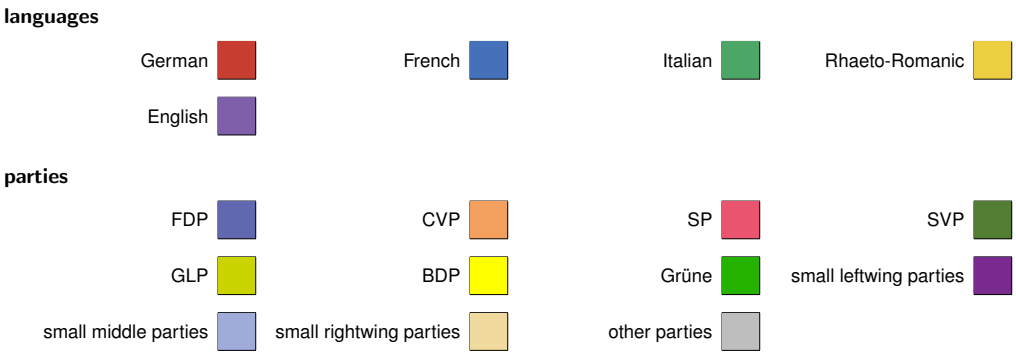
`votes`	colors used for results from votes

Argument `cmyk` requests the CMYK variant; the default is to use the RGB variant. The palettes look as follows.





When using `sfso blue` for color gradients, select colors 1–6 only; the 7th color is an extra color for special purposes. In the `votes` scheme, purple colors mean rejection and green colors mean for approval. The meanings of the colors in `languages` and `parties` are as follows:



5.1.13 HTML colors

(new)

The `HTML` collection provides named HTML colors from www.w3schools.com. The syntax is

`HTML` [*scheme*] [, *palette_options*]

where *scheme* is one of the following:

`pink`	6 pink colors	`purple`	19 purple colors
`red`	14 red and orange colors	`orange`	14 red and orange colors
`yellow`	11 yellow colors	`green`	22 green colors
`cyan`	8 cyan colors	`blue`	16 blue colors
`brown`	18 brown colors	`white`	17 white colors
`gray`	10 gray colors	`grey`	10 grey colors (same as `gray`)

See Section 4.1.1 for an overview of the colors in these schemes. All 148 HTML colors (alphabetically sorted) will be returned if *scheme* is omitted. Also see www.w3schools.com/colors/colors_names.asp or www.w3schools.com/colors/colors_groups.asp.

5.1.14 W3.CSS colors (new)

The `w3` collection provides colors from W3.CSS. The syntax is

`w3` [*scheme*] [, *palette_options*]

where *scheme* is one of the following:

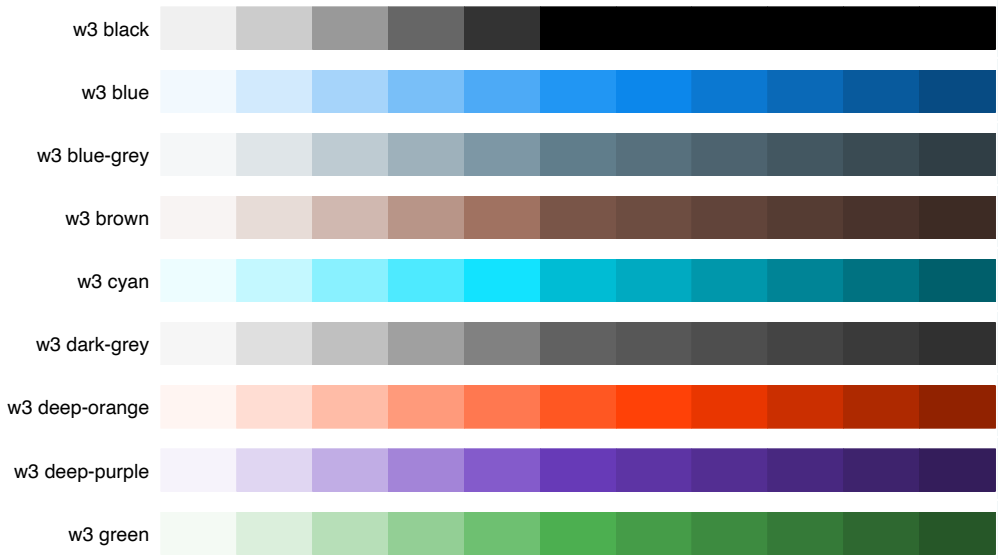
Qualitative collections

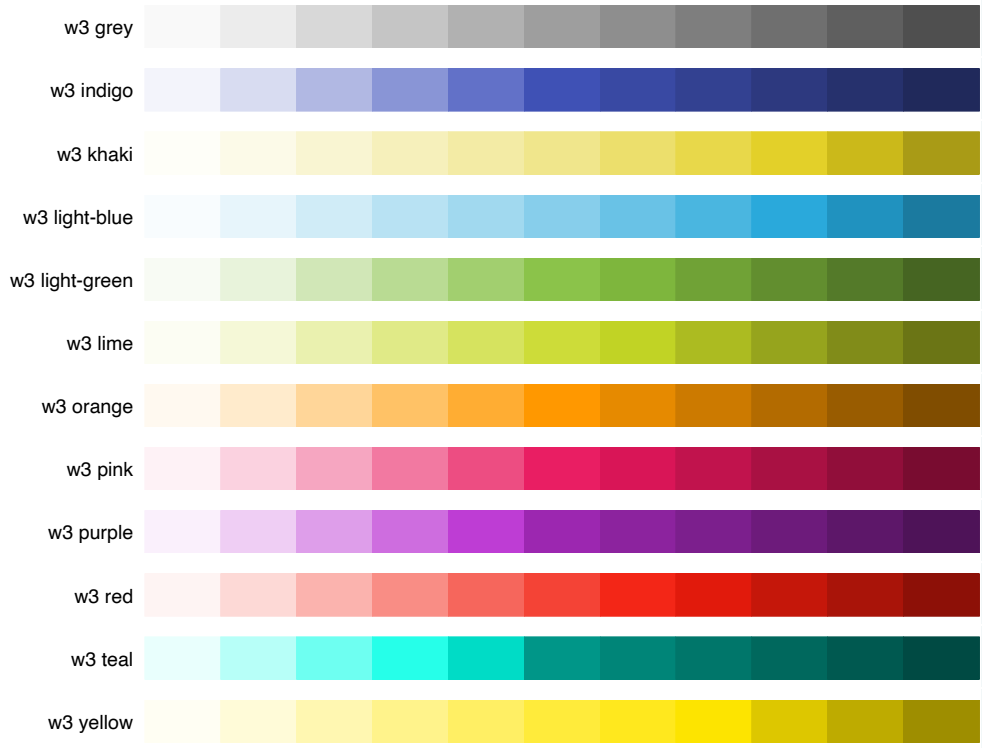
`default`	30 Default Colors (the default)
`flat`	20 Flat UI Colors
`metro`	17 Metro UI Colors
`win8`	22 Windows 8 Colors
`ios`	12 iOS Colors
`highway`	7 US Highway Colors
`safety`	6 US Safety Colors
`signal`	10 European Signal Colors
`2019`	32 Fashion Colors 2019
`2018`	30 Fashion Colors 2018
`2017`	20 Fashion Colors 2017
`vivid`	21 Vivid Colors
`food`	40 Food Colors
`camo`	15 Camouflage Colors
`ana`	44 Army Navy Aero Colors
`traffic`	9 Traffic Colors

Sequential themes (11 colors)

`black`, `blue`, `blue-grey`, `brown`, `cyan`, `dark-grey`, `deep-orange`, `deep-purple`, `green`, `grey`, `indigo`, `khaki`, `light-blue`, `light-green`, `lime`, `orange`, `pink`, `purple`, `red`, `teal`, `yellow`

See Section 4.1.2 for an overview of the colors in the qualitative collections. The sequential themes look as follows.





5.1.15 Wes Anderson palettes

(new)

The *wesanderson* collection provides Wes Anderson palettes from wesanderson-palettes.tumblr.com (the color codes have been obtained from github.com/karthik/wesanderson). The syntax is

scheme [, *palette_options*]

where *scheme* is one of the following:

`BottleRocket1`	7 qualitative colors from Bottle Rocket
`BottleRocket2`	5 qualitative colors from Bottle Rocket
`Rushmore1`	5 qualitative colors from Rushmore
`Royal1`	4 qualitative colors from The Royal Tenenbaums
`Royal2`	5 qualitative colors from The Royal Tenenbaums
`Zissou1`	5 diverging colors from The Life Aquatic with Steve Zissou
`Darjeeling1`	5 qualitative colors from The Darjeeling Limited
`Darjeeling2`	5 qualitative colors from The Darjeeling Limited
`Chevalier1`	4 qualitative colors from Hotel Chevalier
`FantasticFox1`	5 qualitative colors from Fantastic Mr. Fox
`Moonrise1`	4 qualitative colors from Moonrise Kingdom
`Moonrise2`	4 qualitative colors from Moonrise Kingdom
`Moonrise3`	5 qualitative colors from Moonrise Kingdom

`Cavalcanti1`	5 qualitative colors from Castello Cavalcanti
`GrandBudapest1`	4 qualitative colors from The Grand Budapest Hotel
`GrandBudapest2`	4 qualitative colors from The Grand Budapest Hotel
`IsleofDogs1`	6 qualitative colors from The Isle of Dogs
`IsleofDogs2`	5 qualitative colors from The Isle of Dogs
`FrenchDispatch1`	5 qualitative colors from The French Dispatch

The palettes look as follows.

5.2 Colormaps

5.2.1 Viridis colormaps

(new)

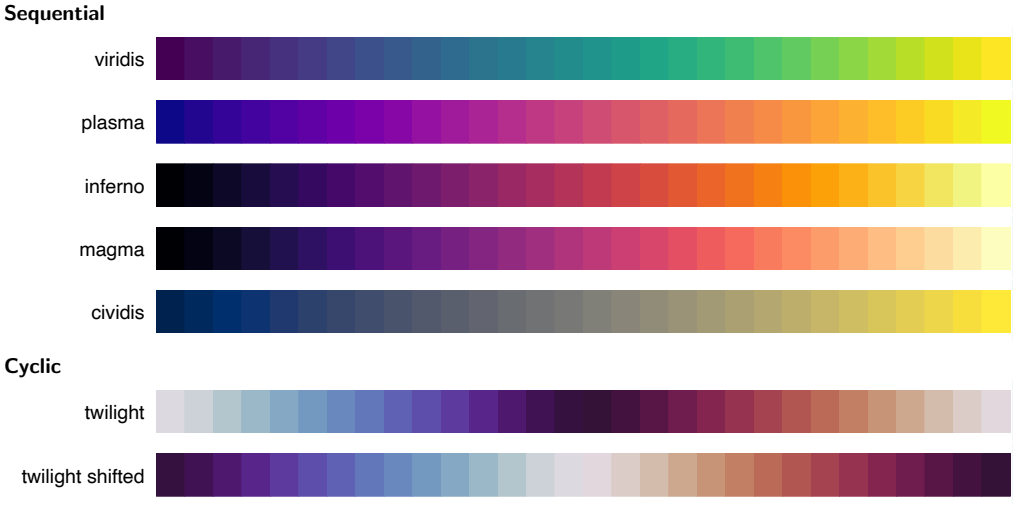
The *viridis* collection provides perceptually uniform colormaps from matplotlib (also see bids.github.io/colormap). The color values have been taken from file _cm_listed.py at github.com/matplotlib. The syntax for the *viridis* palettes is

scheme [, range(*lb* [*ub*]) *palette_options*]

where *scheme* is one of the following

`viridis`	blue – green – yellow (sequential)
`plasma`	blue – red – yellow (sequential)
`inferno`	black – blue – orange – yellow (sequential)
`magma`	black – blue – red – yellow (sequential)
`cividis`	blue – olive – yellow (sequential, colorblind-friendly)
`twilight`	blue – russet (cyclic)
`twilight shifted`	blue – russet (cyclic)

and option `range(`*lb* [*ub*]`)` selects the range of the colormap to be used, where *lb* and *ub* must be in $[0, 1]$. The default is `range(0 1)`. If *lb* is larger than *ub*, the colors are returned in reverse order. `range()` has not effect for cyclic colormaps. The colormaps look as follows.



5.2.2 Seaborn colormaps (new)

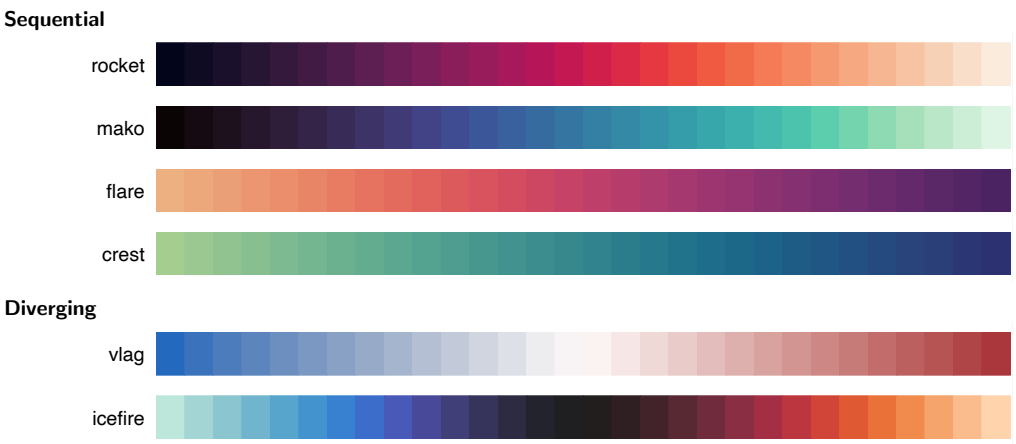
The *seaborn* collection provides perceptually uniform colormaps from seaborn.pydata.org. The syntax is

scheme [, range(*lb* [*ub*]) *palette_options*]

where *scheme* is one of the following

Sequential: `rocket`, `mako`, `flare`, `crest`
Diverging: `vlag`, `icefire`

and `range(`*lb* [*ub*]`)` is as described for *viridis*. The colormaps look as follows.



5.2.3 Other matplotlib maps (new)

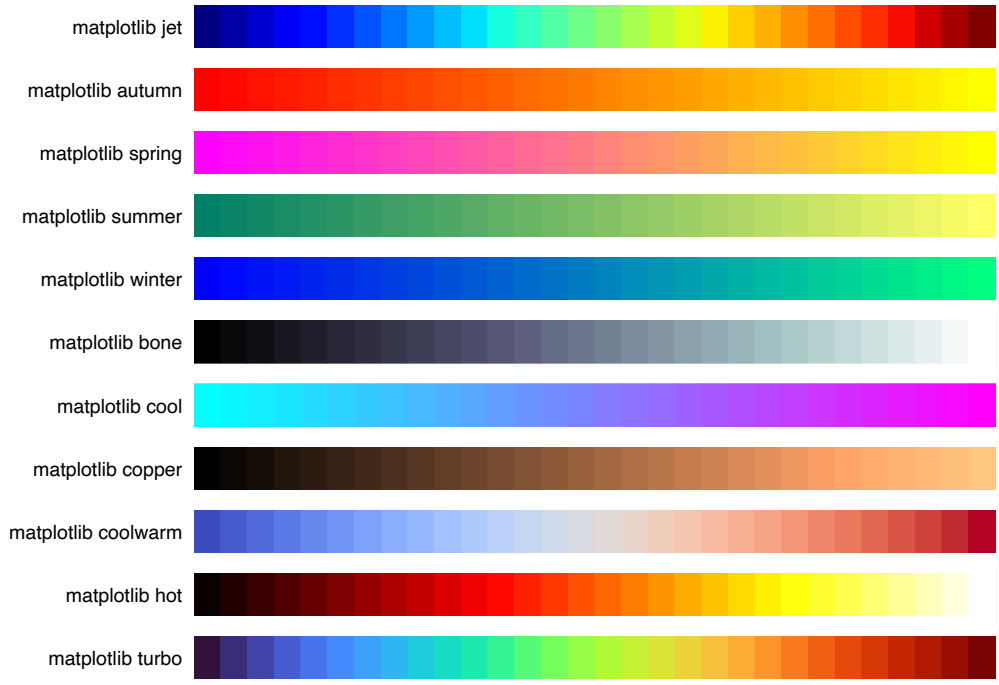
The `matplotlib` collection provides several colormaps from matplotlib (Hunter 2007). The definitions of the colormaps have been taken from file _cm.py at github.com/matplotlib. The syntax for the `matplotlib` collection is

`matplotlib` [*scheme*] [, range(*lb* [*ub*]) *palette_options*]

where *scheme* is one of the following

`jet`	blue – green – yellow – red; the default
`autumn`	red – yellow
`spring`	magenta – yellow
`summer`	green – yellow
`winter`	blue – lime
`bone`	black – bluish gray – white
`cool`	cyan – magenta
`copper`	black – orange/brown
`coolwarm`	blue – red (diverging)
`hot`	heat colors
`turbo`	similar to `jet`

and `range(`*lb* [*ub*]`)` is as described for *viridis*. The colormaps look as follows.



5.2.4 Colormaps by Kovesi (2015)

(new)

The `CET` collection provides perceptually uniform colormaps by Kovesi (2015); see colorcet.com. The syntax is

`CET` [*scheme*] [`, range(`*lb* [*ub*]`)` *palette_options*]

where *scheme* is one of the following

Linear
: `L01`, `L02`, `L03`, `L04`, `L05`, `L06`, `L07`, `L08`, `L09`, `L10`, `L11`, `L12`, `L13`, `L14`, `L15`, `L16`, `L17`, `L18`, `L19`, `L20` (default)

Rainbow
: `R1`, `R2`, `R3`, `R4`

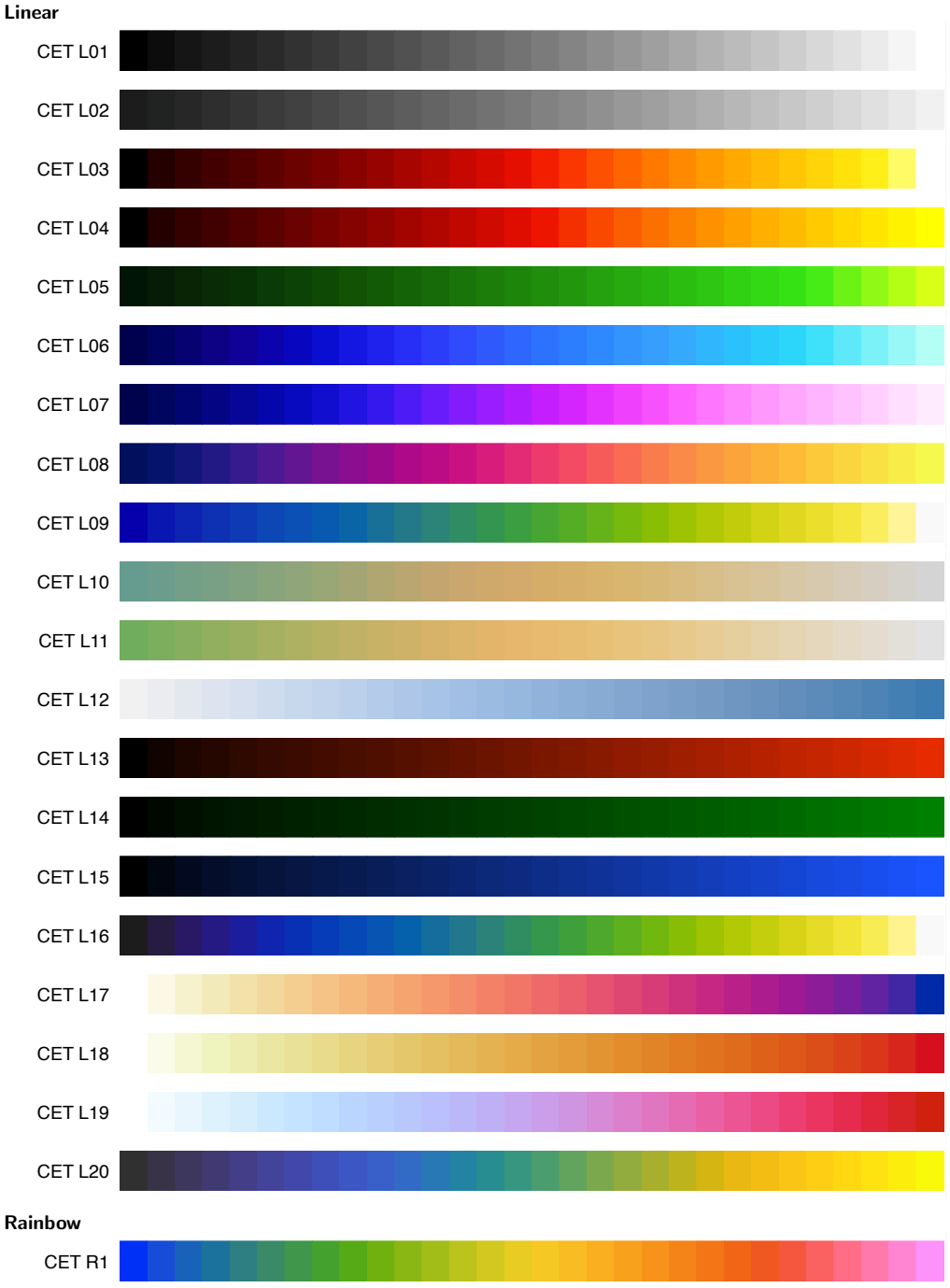
Isoluminant
: `I1`, `I2`, `I3`

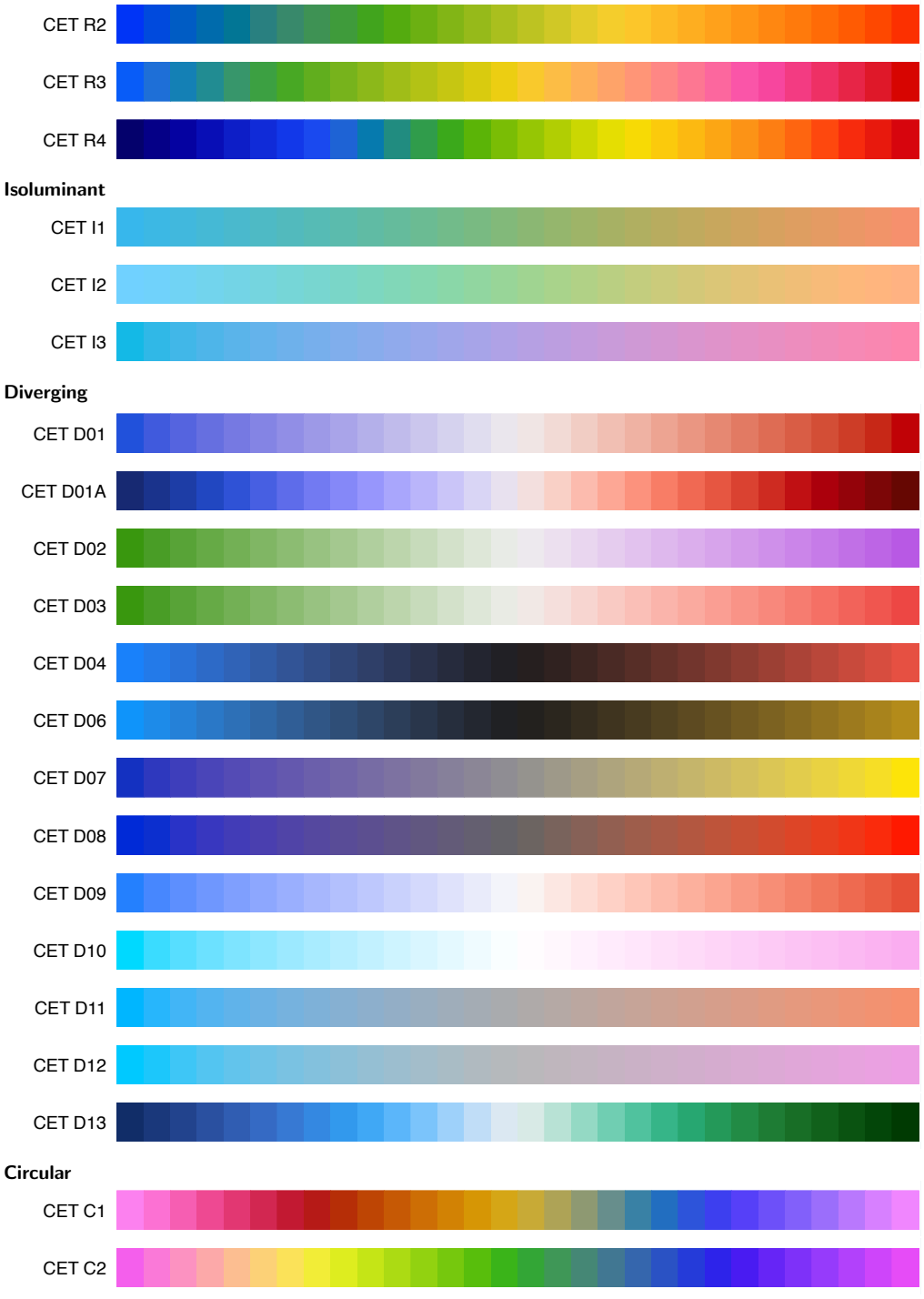
Diverging
: `D01`, `D01A`, `D02`, `D03`, `D04`, `D06`, `D07`, `D08`, `D09`, `D10`, `D11`, `D12`, `D13`

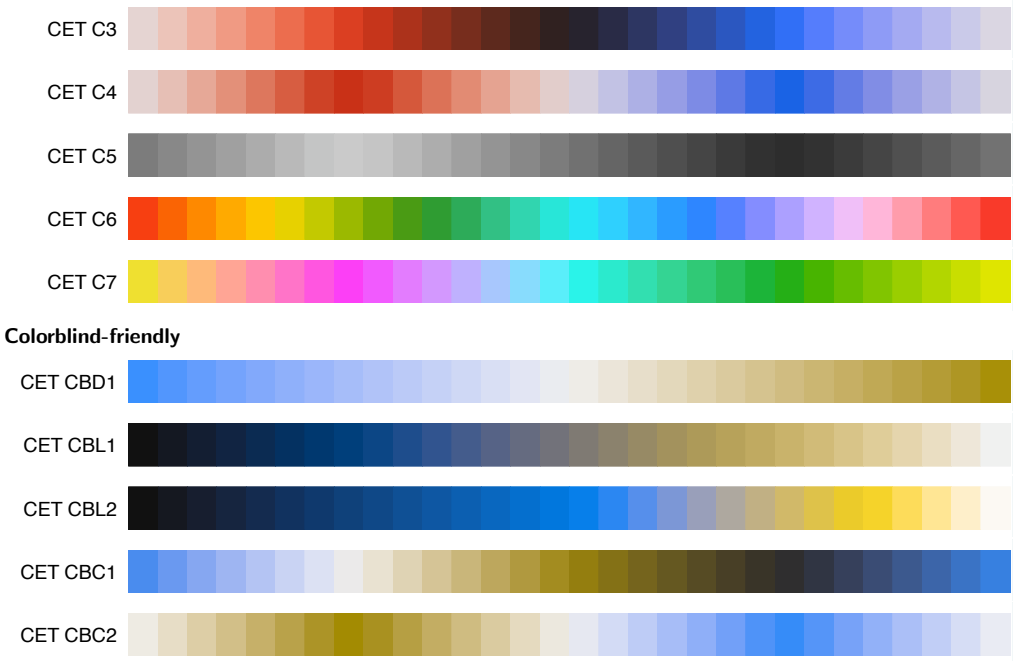
Circular
: `C1`, `C2`, `C3`, `C4`, `C5`, `C6`, `C7`

Colorblind-friendly
: `CBD1`, `CBL1`, `CBL2`, `CBC1`, `CBC2`

and `range(`*lb* [*ub*]`)` is as described for *viridis*. The colormaps look as follows.




CET R2
CET R3
CET R4
Isoluminant
CET I1
CET I2
CET I3
Diverging
CET D01
CET D01A
CET D02
CET D03
CET D04
CET D06
CET D07
CET D08
CET D09
CET D10
CET D11
CET D12
CET D13
Circular
CET C1
CET C2



5.2.5 Scientific colour maps (Crameri 2018)

(new)

The `scico` collection provides perceptually uniform colorblind-friendly colormaps by Crameri (2018); see www.fabiocrameri.ch/colourmaps. The syntax is

`scico` [*scheme*] [`, range(`*lb* [*ub*]`)` *palette_options*]

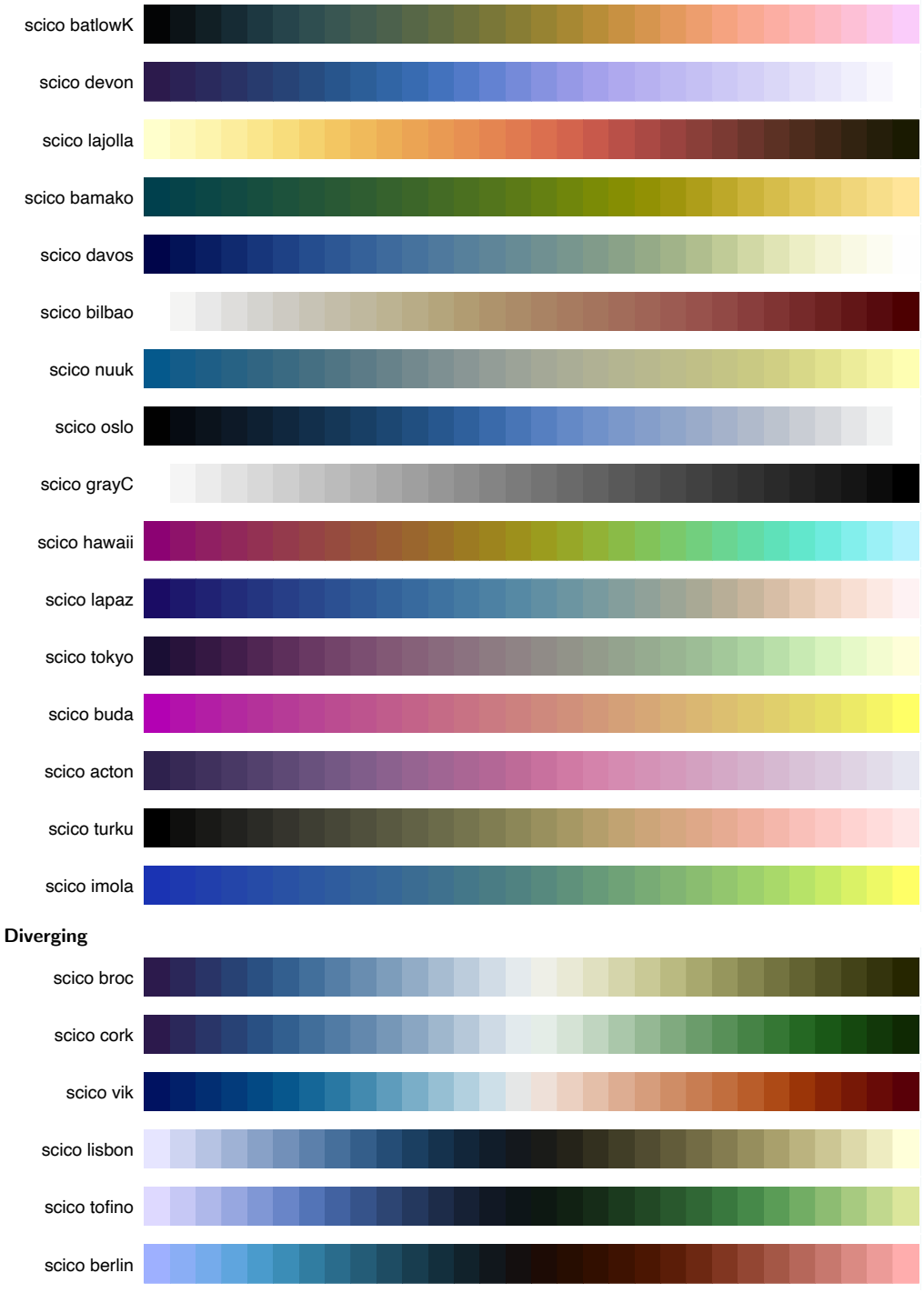
where *scheme* is one of the following

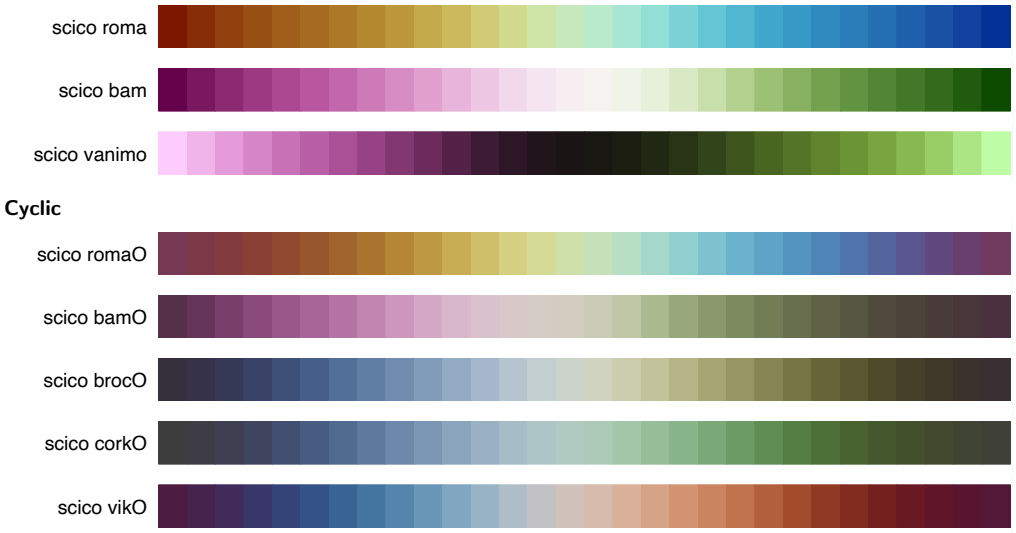
Sequential
: `batlow` (default), `batlowW`, `batlowK`, `devon`, `lajolla`, `bamako`, `davos`, `bilbao`, `nuuk`, `oslo`, `grayC`, `hawaii`, `lapaz`, `tokyo`, `buda`, `acton`, `turku`, `imola`

Diverging
: `broc`, `cork`, `vik`, `lisbon`, `tofino`, `berlin`, `roma`, `bam`, `vanimo`

Cyclic
: `romaO`, `bamO`, `brocO`, `corkO`, `vikO`

and `range(`*lb* [*ub*]`)` is as described for *viridis*. The colormaps look as follows.


scico batlowK
scico devon
scico lajolla
scico bamako
scico davos
scico bilbao
scico nuuk
scico oslo
scico grayC
scico hawaii
scico lapaz
scico tokyo
scico buda
scico acton
scico turku
scico imola
Diverging
scico broc
scico cork
scico vik
scico lisbon
scico tofino
scico berlin

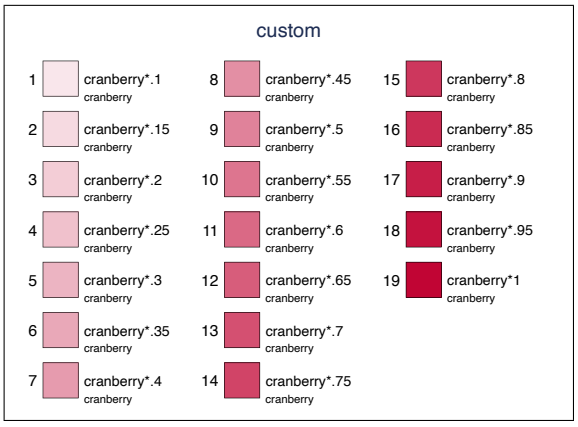


5.3 Color generators

5.3.1 Generate colors over a range of intensity 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)
```



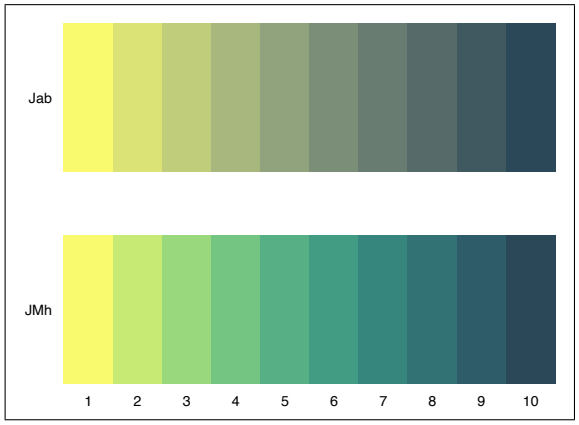
5.3.2 Generate colors by interpolation

(new)

A powerful color generator is provided via the `ipolate()` option. The procedure is to select a start color and an end color, and perhaps some intermediate colors, and then apply interpolation to generate a color scale. Several suboptions to select the interpolation space, set the positions of the origin colors, or affect the shape of the transition between the colors are available (dee the description of the `ipolate()` option further details)

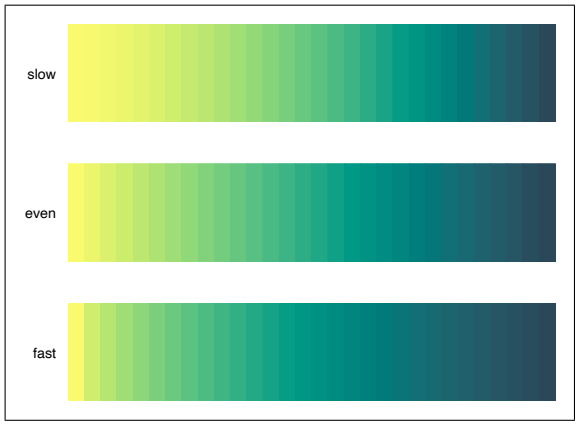
The default is to interpolate in the CIECAM02-based $J'a'b'$ space, which is supposed to be perceptually uniform and does not travel around the color wheel. If you want to interpolate around the color wheel, you could, for example, use the CIECAM02-based $J'M'h$ space:

```
. colorpalette, labels(Jab JMh):
>       #fafa6e #2a4858, ipolate(10)
>     / #fafa6e #2a4858,
>         ipolate(10, JMh)
```



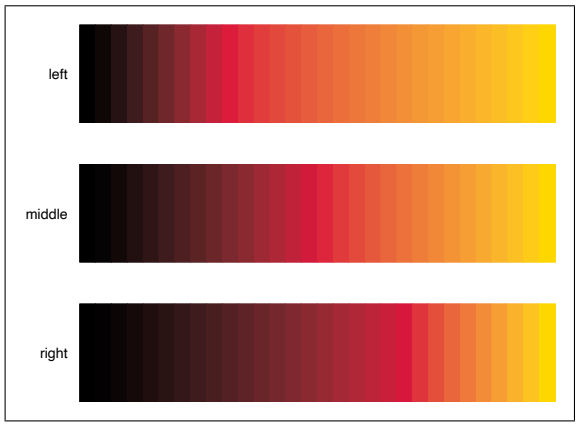
The `power()` suboption determines the speed of the transition between the colors. A value larger than one makes the first color more dominant (slow to fast transition); a value smaller than one makes the second color more dominant (fast to slow transition). Example:

```
. colorpalette, nonumbers
>     labels(slow even fast):
>       #fafa6e #2a4858,
>         ipolate(30, HCL power(1.5))
>     / #fafa6e #2a4858,
>         ipolate(30, HCL)
>     / #fafa6e #2a4858,
>         ipolate(30, HCL power(0.7))
```



When interpolating between more than two colors, the default is to arrange the origin colors on a regular grid. This can be changed by the `positions()` suboption. In the following example, the `positions()` suboption is used to shift the middle color to the left or to the right, respectively:

```
. colorpalette, nonumbers
>     labels(left middle right):
>       Black Crimson Gold,
>           ipolate(30, pos(0 .3 1))
>     / Black Crimson Gold,
>         ipolate(30)
>     / Black Crimson Gold,
>         ipolate(30, pos(0 .7 1))
```



5.3.3 Generate evenly spaced HCL hues

The `hue` generator 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). The `hue` palette with default options produces the same colors as the `intense` scheme of the `hcl` color generator (see below). The syntax is

`hue` [, *settings* *palette_options*]

where *settings* 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).

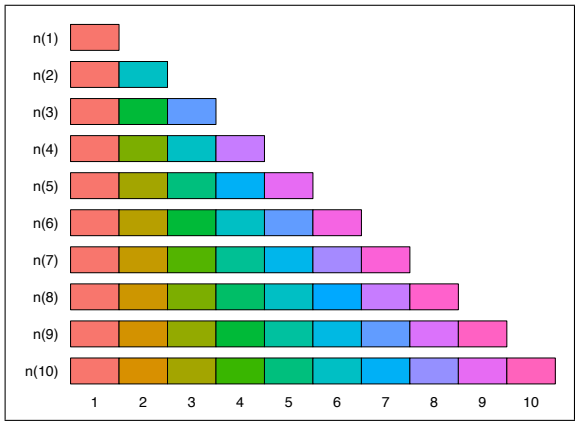
`chroma(`c`)` sets the colorfulness (color intensity), with $c \geq 0$. The default is `chroma(100)`.

`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 counterclockwise.

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)
```



5.3.4 HCL, LCh, and JMh color generators

(revised)

The `HCL`, `LCh`, and `JMh` palette are color generators in the HCL (Hue-Chroma-Luminance) space (cylindrical representation of CIE $L^*u^*v^*$), the LCh space (cylindrical representation of CIE $L^*a^*b^*$), and the CIECAM02-based $J'M'h$ space, respectively. The implementation is based on R's `colorspace` package by Ihaka et al. (2016); also see Zeileis et al. (2009) and hclwizard.org. The `LCh` and `JMh` generators are implemented analogously. The syntax is

{ `HCL` | `LCh` | `JMh` } [*scheme*] [`,` *settings* *palette_options*]

where *scheme* selects the type of scheme and the default parameters[5] according to the

[5]The shown parameter values are for `HCL`; `LCh` and `JMh` use adjusted values such that the end points

following overview (the default is `qualitative`)

	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

and *settings* are:

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

`chroma`$(c_1\ [c_2])$ overrides the default values for c_1 and c_2, $c \geq 0$, that determine the colorfulness (color intensity, M' in case of `JMh`).

`luminance`$(l_1\ [l_2])$ overrides the default values for l_1 and l_2, $l \in [0, 100]$, that determine the brightness (amount of gray; J' in case of `JMh`).

`power`$(p_1\ [p_2])$ overrides the default values for p_1 and p_2, $p > 0$, 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.

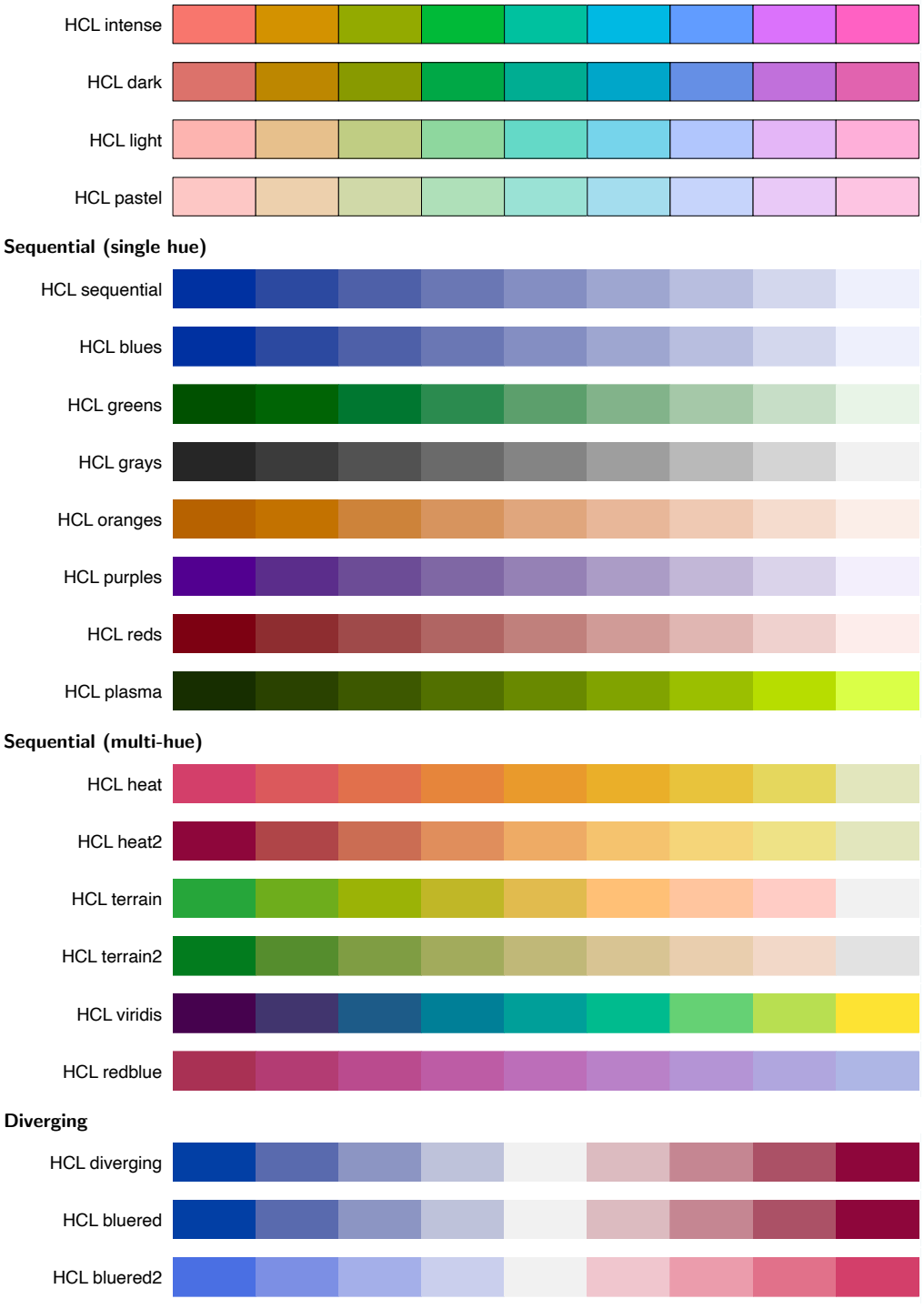
Given the parameters above, colors are generated according to the following equations, where i an index from 1 to n with n as the number of requested colors:

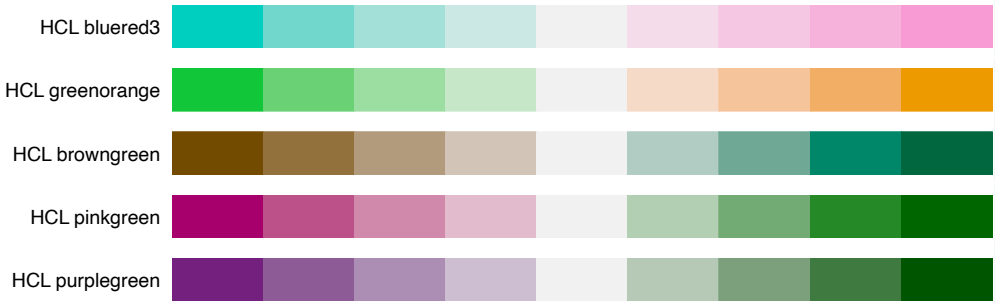
	H_i	C_i	L_i	j
Qualitative	$h_1 + j(h_2 - h_1)$	c_1	l_1	$\frac{i-1}{n-1}$
Sequential	$h_2 - j(h_2 - h_1)$	$c_2 - j^{p_1}(c_2 - c_1)$	$l_2 - j^{p_2}(l_2 - l_1)$	$\frac{n-i}{n-1}$
Diverging	h_1 if $j > 0$ h_2 else	$\lvert j\rvert^{p_1} c_1$	$l_2 - \lvert j\rvert^{p_2}(l_2 - l_1)$	$\frac{n-2j+1}{n-1}$

The predefined `HCL` schemes with default parameters for $n = 9$ look as follows.

Qualitative

of the generated colors are similar to the ones generated by `HCL` (see source file `colrspace_library_generators.sthlp`).


HCL intense
HCL dark
HCL light
HCL pastel
Sequential (single hue)
HCL sequential
HCL blues
HCL greens
HCL grays
HCL oranges
HCL purples
HCL reds
HCL plasma
Sequential (multi-hue)
HCL heat
HCL heat2
HCL terrain
HCL terrain2
HCL viridis
HCL redblue
Diverging
HCL diverging
HCL bluered
HCL bluered2



5.3.5 HSV and HSL color generators

(revised)

The `HSV` and `HSL` palettes are color generators in the HSV (Hue-Saturation-Value) and the HSL (Hue-Saturation-Lightness) spaces, respectively. 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). The used formulas are analogous to the formulas of the HCL generator (replacing chroma by saturation and replacing luminance by value or lightness).[6] The syntax is

{ `HSV` | `HSL` } [*scheme*] [`,` *settings* *palette_options*]

where *scheme* selects the type of scheme and the default parameters[7] according to the following overview (the default is `qualitative`)

	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	`heat0`	0	20	1	0	1	–	–	–
`terrain0`	120	0	1	0	.65	.9	–	–									
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

and *settings* are:

[6] Although `HSV heat0` and `HSV terrain0` use somewhat different formulas; see the source code of `ColrSpace` for details.

[7] The shown parameter values are for `HSV`; `HSL` only supports `qualitative`, `sequential`, and `diverging`, and uses adjusted values such that the end points of the generated colors are similar to the ones generated by `HSL` for these schemes (see source file `colrspace_library_generators.sthlp`).

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

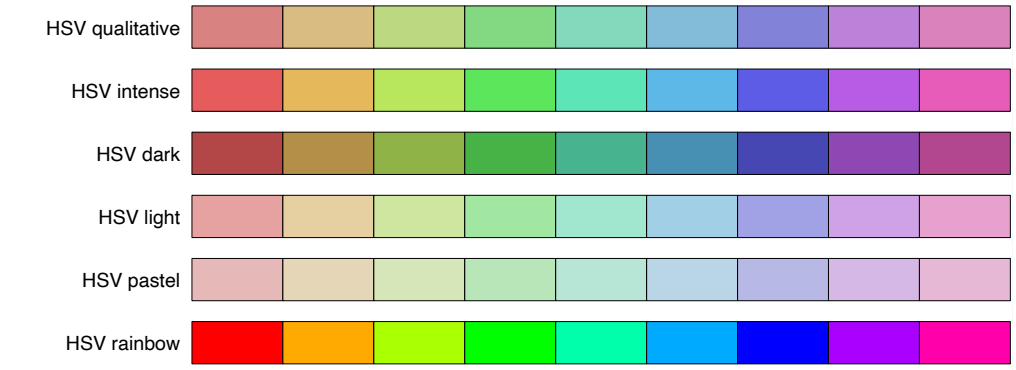
`saturation`(s_1 [s_2]) overrides the default values for s_1 and s_2, $s_i \in [0, 1]$, that determine the colorfulness (color intensity).

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

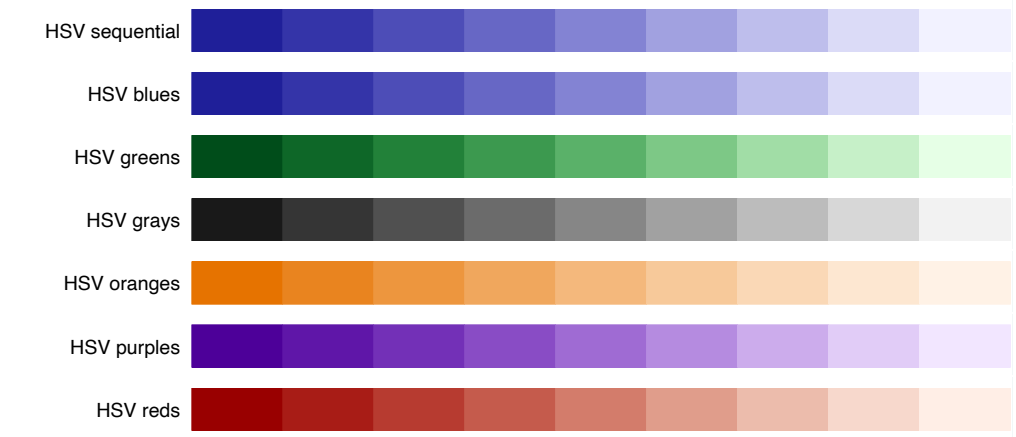
`power`(p_1 [p_2]) overrides the default values for p_1 and p_2, $p_i > 0$, that determine the shape of the transition between chroma and luminance levels. For linear transitions, set $p_i = 1$; $p_i > 1$ makes the transition faster, $p_i < 1$ makes the transition slower.

The predefined `HSV` schemes with default parameters for $n = 9$ look as follows.

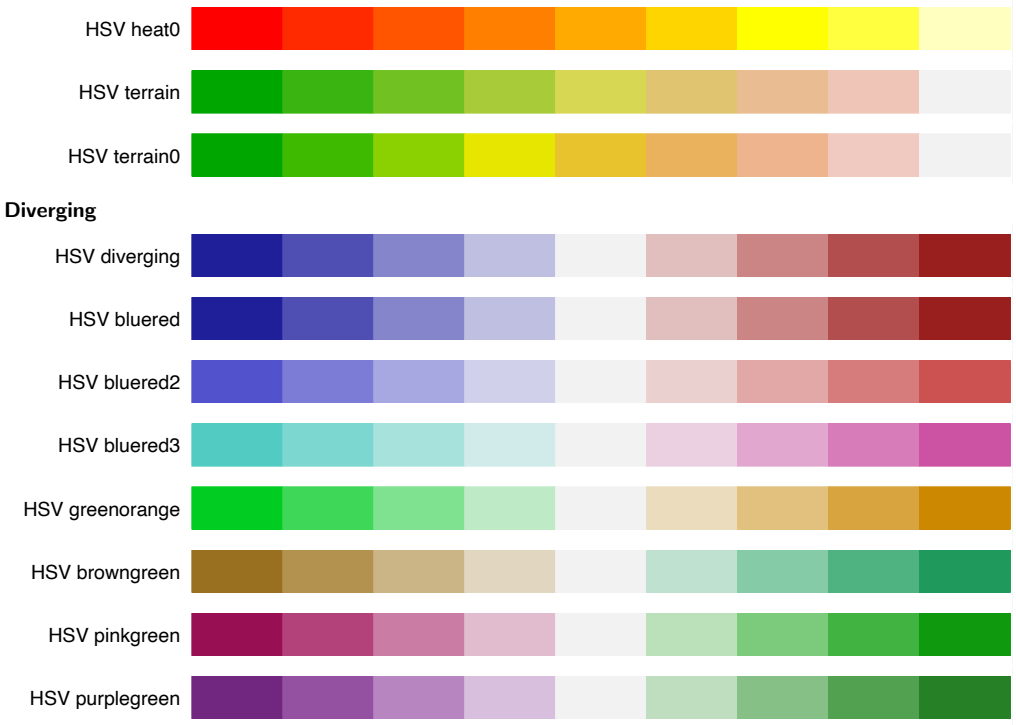
Qualitative



Sequential (single hue)



Sequential (multi-hue)



6 References

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

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

Crameri, F. 2018. Scientific colour maps. Zenodo. DOI: 10.5281/zenodo.1243862.

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

Hunter, J. D. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9(3): 90–95.

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. http://CRAN.R-project.org/package=colorspace.

Jann, B. 2017. PALETTES: Stata module to provide color palettes, symbol palettes, and line pattern palettes. Statistical Software Components S458444, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s458444.html.

———. 2018a. COLRSPACE: Stata module providing a class-based color management system in Mata. Statistical Software Components S458597, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s458597.html.

———. 2018b. Color palettes for Stata graphics. *The Stata Journal* 18(4): 765–785.

———. 2018c. Customizing Stata graphs made easy (part 2). *The Stata Journal* 18(4): 786–802.

———. 2022. ColrSpace: A Mata class for color management. University of Bern Social Sciences Working Papers 42. https://ideas.repec.org/p/bss/wpaper/42.html.

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

Kovesi, P. 2015. Good Colour Maps: How to Design Them. arXiv:1509.03700 [cs.GR].

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.

Machado, G. M., M. M. Oliveira, and L. A. F. Fernandes. 2009. A Physiologically-based Model for Simulation of Color Vision Deficiency. *IEEE Transactions on Visualization and Computer Graphics* 15(6): 1291–1298.

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. 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. 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. http://jfly.iam.u-tokyo.ac.jp/color/.

Pisati, M. 2007. SPMAP: Stata module to visualize spatial data. Statistical Software Components S456812, Boston College Department of Economics. http://ideas.repec.org/c/boc/bocode/s456812.html.

Tol, P. 2012. Colour Schemes. SRON Technical Note, Doc. no. SRON/EPS/TN/09-002. 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.