



*The World's Largest Open Access Agricultural & Applied Economics Digital Library*

**This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.**

**Help ensure our sustainability.**

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

[aesearch@umn.edu](mailto:aesearch@umn.edu)

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

*No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.*

# Color palettes for Stata graphics: An update

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

**Abstract.** This article is an update to Jann (2018a, *Stata Journal* 18: 765–785). 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. Furthermore, a new command called `colorcheck` is presented that can be used to evaluate whether colors are distinguishable by people suffering from color vision deficiency.

**Keywords:** `gr0075_1`, palettes, `colorpalette`, `colorcheck`, ColrSpace, graph, graphics, color, color spaces, color interpolation, color vision deficiency, grayscale conversion, perceptually uniform

This article has extensive use of colors, so the electronic copy has been published in color, while the printed copy is in monochrome. If you are reading the printed copy and are having trouble following the text, you may want to switch to the electronic copy.

## 1 Introduction

In Jann (2018a), I introduced command `colorpalette` to enhance support for colors in Stata graphics. The command has primarily been written as a backend for `grstyle set`, a command that customizes the look of Stata graphics (Jann 2018b). However, `colorpalette` can also be called directly to retrieve colors for use in subsequent graph commands. Substantial changes and additions have been made to `colorpalette` since its first publication. Most fundamentally, `colorpalette` is now based on a full-fledged color management system written in Mata that provides functionality such as translation of colors among a variety of color spaces, high-quality color interpolation, and other tools such as grayscale conversion or color-deficiency simulation. Furthermore, a large collection of named colors and a variety of additional palettes have been added.

The current article documents the revised version of `colorpalette`. Because of the wide scope of the changes, full documentation of `colorpalette` is provided (rather than just discussing the additions), and readers are not expected to be familiar with Jann (2018a). In addition, the article also introduces a new convenience command called `colorcheck` that can be used to evaluate whether the colors are distinguish-

able in (noncolor) print or by people who suffer from color vision deficiency (CVD). Documentation of `ColrSpace`, the Mata-based color management system that is the engine behind `colorpalette` and `colorcheck`, is provided in an online supplement (see section 6).

## 2 Syntax

### 2.1 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 3.4 and section 4), a named palette (see section 3.9 and section 5), or `mata(name)`, where *name* is the name of a `ColrSpace` object (see the online supplement).

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.

#### 2.1.1 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 interpolated colors.

`n(#)` specifies the size of the palette (the 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`.

`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()`.

`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.

`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.

`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.

`opacity(numlist)` sets the opacity level or levels (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:

`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` in the online supplement for details.

**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 toward the lower bound. If you interpolate between two colors, the first color will dominate most of the interpolation range (slow to fast transition). A value between 0 and 1 squishes the positions toward 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 endpoints 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.

`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:

`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:

`cspace` specifies the color space in which the colors are manipulated. The 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 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 CVD (based on Machado, Oliveira, and Fernandes [2009]), where `numlist` in  $[0, 1]$  specifies the severity of the deficiency. The default severity is 1 (maximum severity, that is, deuteranopia, protanopia, or tritanopia). 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 [https://en.wikipedia.org/wiki/Color\\_blindness](https://en.wikipedia.org/wiki/Color_blindness) for basic information on color blindness.

**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.

**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.

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

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

*other\_options* are additional palette-specific options. See the descriptions of the palettes below (section 5). When collecting results from multiple palettes, you can specify palette options at the global level to define default settings for all palettes or at the local level of an individual palette. For general palette options, you can override defaults set at the global level by repeating an option at the local level. Such repetitions are not allowed for palette-specific options.

### 2.1.2 Macro options

The following options are available only 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 section 3.7 for an example. **globals()** disables graph display unless option **graph** is specified. The syntax of *spec* is

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

**suffix(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.

`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 3.8 for an example. Style files will be created only 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

```
[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 `pwd` in [D] **cd**.

`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.

### 2.1.3 Graph options

#### Common graph options

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

`nonumbers` suppresses the numbers identifying the colors in the graph.

`gropts(twoway_options)` provides options to be passed through to the `graph` command; see [G-3] *twoway\_options*.



**Additional graph options for syntax 1**

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

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

`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.

`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.

`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 the fill.

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

**2.1.4 Stored results**

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

Scalars:

`r(n)`                      number of colors

Macros:

<code>r(ptype)</code>	<code>color</code>	<code>r(pname)</code>	name of palette or custom
<code>r(pclass)</code>	palette class (if available)	<code>r(pnote)</code>	palette description (if available)
<code>r(psource)</code>	palette source (if available)	<code>r(p)</code>	space-separated list of colors
<code>r(p#)</code>	<code>#th</code> color	<code>r(p#name)</code>	name of <code>#th</code> color (if available)
<code>r(p#info)</code>	info of <code>#th</code> color (if available)		

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

## 2.2 Syntax of `colorcheck`

The `colorcheck` command analyzes the colors returned by `colorpalette` by applying grayscale conversion and CVD transformation and by computing minimum color differences among the converted colors. Results from `colorpalette` are required to be in memory. The syntax is

```
colorcheck [ , options ]
```

### 2.2.1 Options

`metric(metric)` selects the color difference metric to be used. *metric* can be `E76`, `E94`, `E2000`, or `Jab`; see the documentation of `ColrSpace` in the online supplement for details. The 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)`, that is, full conversion) and *method* selects the conversion method. Default is `LCh`; see the documentation of `ColrSpace` in the online supplement for available methods.

`cvd(#)`, with *#* in  $[0, 1]$ , sets the severity of CVD. The 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 has an effect only 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.

## 2.2.2 Stored results

`colorcheck` adds (or updates) the following results. The results from `colorpalette` are kept in memory.

Scalars:

<code>r(mono)</code>	proportion of gray
<code>r(cvd)</code>	CVD severity

Macros:

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

Matrix:

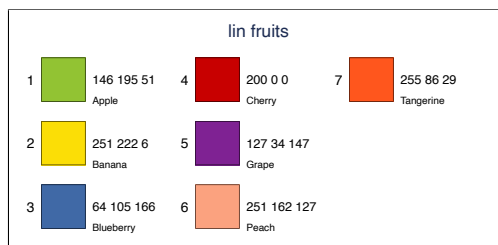
<code>r(delta)</code>	color difference statistics
-----------------------	-----------------------------

## 3 Basic usage

### 3.1 View a palette (syntax 1)

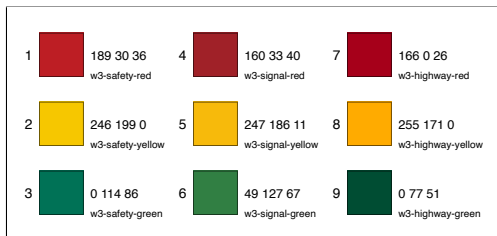
To display a single palette, type `colorpalette` followed by the name of the palette. The graph produced by `colorpalette` displays the colors as well as their color codes and, possibly, some additional information. For example, to view the `fruits` palette from Lin et al. (2013), type

```
. colorpalette lin fruits, rows(3)
```



Option `rows()` sets the (minimum) number of rows in the graph. 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 signaling colors from different `w3` palettes:

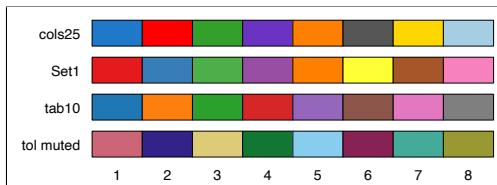
```
. colorpalette
>   w3 safety, select(1 3 4) /
>   w3 signal, select(3 1 6) /
>   w3 highway, select(2 5 6) /
>   , title("") rows(3)
```



## 3.2 View multiple palettes (syntax 2)

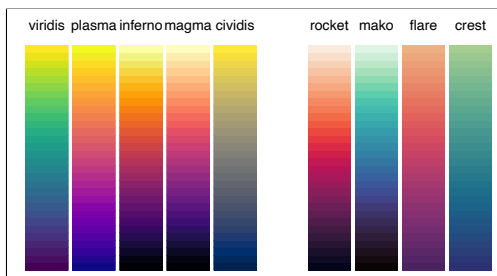
To display an overview of multiple palettes in a single graph, first type `colorpalette` and possibly add some global options, then type a colon followed by a list of palettes separated by forward slashes. Here is a simple example that displays colors 1–8 from several categorical palettes; option `lcolor(black)` has been specified to draw black lines around the color fields:

```
. colorpalette,
>   select(1/8) lcolor(black):
>   cols25 / Set1 / tab10 /
>   tol muted
```



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 width of the color bars), `nonumbers` (suppress the color index), and `gropts()` (pass options through to the underlying [G-2] `graph twoway` command) and also shows how empty slots can be introduced by typing a dot (missing); in addition, the example uses option `n()` to determine the number of colors to be generated per palette:

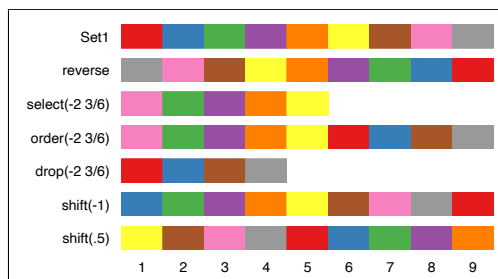
```
. colorpalette, n(30) vertical
>   barwidth(.9) nonumbers
>   gropts(yscale(noreverse)):
>   /* viridis colormaps */
>   viridis / plasma / inferno
>   / magma / cividis
>   /* add gap */
>   / . /
>   /* seaborn colormaps */
>   rocket / mako / flare, reverse
>   / crest, reverse
```



### 3.3 Select and order colors in a palette

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, and 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)
```



### 3.4 Specify a custom list of colors

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

```
[ ( ) colorspec [ colorspec ... ] ( ) ]
```

where *colorspec* is

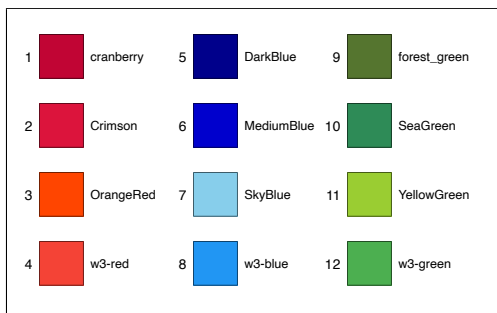
```
[ " ] color[ %# ] [ *# ] [ " ]
```

You may use parentheses around the list, 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:

<i>name</i>	a color name; this includes official Stata's color names as listed in [G] <i>colorstyle</i> , possible user additions provided through style files, and a large collection of named colors provided by <i>colorpalette</i> (see section 4)
<i>#rrggbb</i>	6-digit hex RGB value; for example, white = #FFFFFF or #ffffff, navy = #1A476F or #1a476f
<i>#rgb</i>	3-digit abbreviated hex RGB value; for example, white = #FFF or #fff
<i># # #</i>	RGB value in 0–255 scaling; for example, navy = "26 71 111"
<i># # # #</i>	CMYK value in 0–255 or 0–1 scaling; for example, navy = "85 40 0 144" or ".333 .157 0 .565"
<i>cspace ...</i>	color value in one of the color spaces supported by ColrSpace; for example, 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 the online supplement for details)

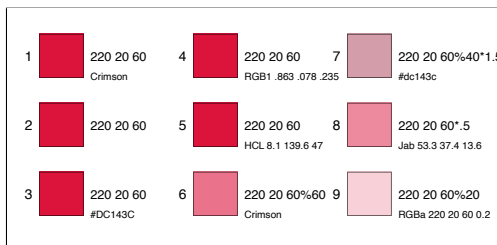
Here is an example displaying some of Stata's named colors (see [G] *colorstyle*) as well as some of the additional named colors provided by *colorpalette* (see section 4):

```
. colorpalette cranberry Crimson
> OrangeRed w3-red DarkBlue
> MediumBlue SkyBlue w3-blue
> forest_green SeaGreen
> YellowGreen w3-green
> , rows(4) title("")
> names noinfo
```



Colors can also be specified as color codes in a variety of formats (see section 6.1 in the online supplement). *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"
> Crimson%60 #dc143c%40*1.5
> "Jab 53.3 37.4 13.6*0.5"
> "RGBa 220 20 60 0.2"
> , rows(3) title("")
```



## 3.5 Manipulate and analyze colors

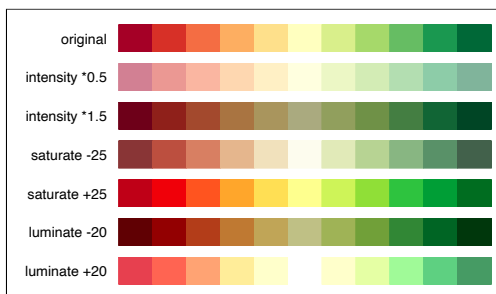
### 3.5.1 Color interpolation

See section 5.3.2.

### 3.5.2 Change intensity, saturation, and luminance

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)
```

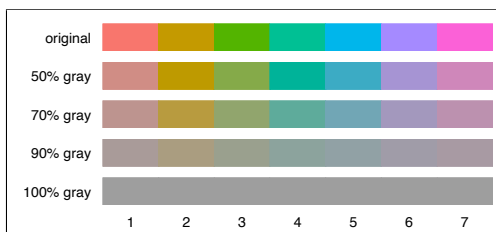


The values specified in `saturate()` and `luminate()` are addends to the chroma and luminance channels, respectively. Reasonable values are in a range of about  $\pm 50$ . The values specified in `intensify()` are intensity factors; typical values are between 0 (white) and about 10 (black).

### 3.5.3 Grayscale conversion

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

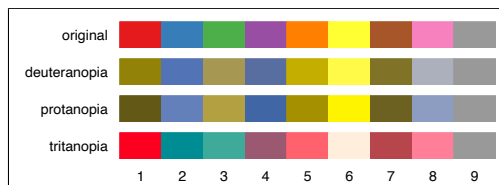
```
. colorpalette, n(7) labels(original
>   "50% gray" "70% gray"
>   "90% gray" "100% gray"):
>   hue
>   / hue, gscale(.5)
>   / hue, gscale(.7)
>   / hue, gscale(.9)
>   / hue, gscale
```



### 3.5.4 Color vision deficiency simulation

A substantial fraction of people suffer from 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)`.

### 3.5.5 Analyze colors using colorcheck

`colorcheck` is a convenience command to evaluate whether colors will be distinguishable by people who suffer from CVD and also whether colors will be distinguishable in (noncolor) 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 general procedure is to obtain colors by running `colorpalette` and then apply `colorcheck` one or several times to analyze the colors. Example:

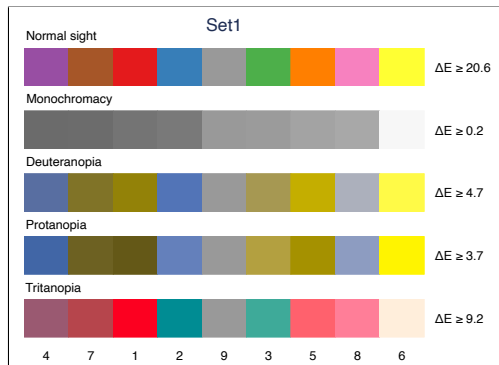
```
. colorpalette Set1, nograph
. colorcheck, nograph
      Number of colors      =      9
      N. of comparisons    =     36
      CVD severity         =      1
      Proportion of gray   =      1
      Grayscale method     =     LCh
      Difference metric    =     Jab
```

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, you will find the `sort()` option can be helpful. For example, specify `sort(mono)` to sort the colors by gray scale:

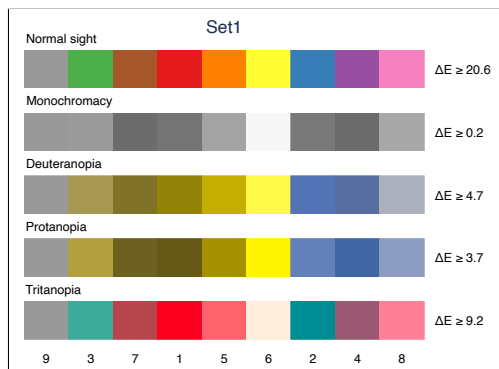


```
. colorcheck, sort(mono)
```



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

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



See [https://en.wikipedia.org/wiki/Color\\_blindness](https://en.wikipedia.org/wiki/Color_blindness) for background information on CVD. See [https://en.wikipedia.org/wiki/Color\\_difference](https://en.wikipedia.org/wiki/Color_difference) for information on color differences.

### 3.6 Retrieve colors from colorpalette

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(pdtype) : "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\""

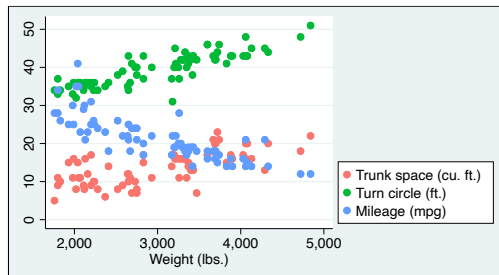
```

Options such as `n()` or `select()` will affect the information stored in `r()`. That is, the stored information reflects the configuration of the palette after you apply all palette options. Here is an example that uses colors from the `hue` generator in a scatterplot:

```

. sysuse auto, clear
(1978 automobile data)
. colorpalette hue, n(3) nograph
. scatter trunk turn mpg weight,
>   mcolor(`r(p)')
>   ysize(3) scale(1.5)
>   legend(position(4) cols(1))

```



Note that many commands, including most graph commands, clear `r()`. If you want to use the same colors in multiple graphs without having to call `colorpalette` repeatedly, it may thus be convenient to copy `r(p)` to a local or global macro (see [P] **macro**). Alternatively, see section 3.7 and section 3.8 on how to make colors available as local macros, global macros, or system colors. Yet another possibility is to use the `grstyle set` command to change the default colors used in Stata graphs; `grstyle set` calls `colorpalette` internally (see Jann [2018b]).

### 3.7 Make colors available as globals or locals

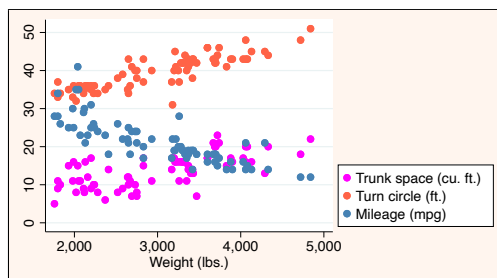
Instead of retrieving color codes from `r()` as described in section 3.6, you can also directly store the colors returned by `colorpalette` as local or global macros by applying option `locals()` or `globals()`, respectively. These local or global macros can then be used in subsequent graph commands 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))
>   ysize(3) scale(1.5)
>   legend(position(4) cols(1))
```

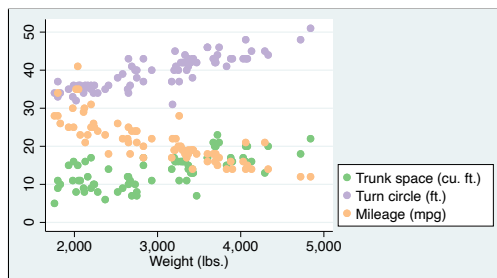


The defined globals will remain in memory until you restart Stata or until you drop them using command `macro drop`.

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 (for example, `$p1`, `$p2`, etc.; see section 2.1.2 for details on naming conventions). Furthermore, depending on context (for example, 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')
>   ysize(3) scale(1.5)
>   legend(position(4) cols(1))
```



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

### 3.8 Make colors permanently available

You can also make colors 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:<sup>1</sup>

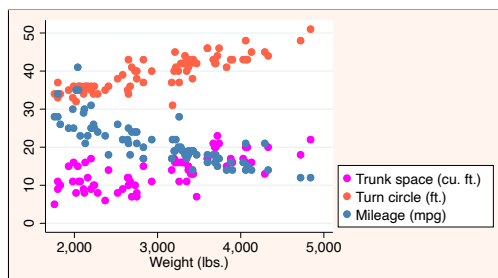
```
. colorpalette Fuchsia Tomato SteelBlue SeaShell, stylefiles
directory style does not exist
press any key to create the directory, or Break to abort
color styles:
      Fuchsia : "255 0 255"
      Tomato  : "255 99 71"
      SteelBlue : "70 130 180"
      SeaShell : "255 245 238"

(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.

```
. sysuse auto, clear
(1978 automobile data)

. scatter trunk turn mpg weight,
>     mc(Fuchsia Tomato SteelBlue)
>     graphr(color(SeaShell))
>     ysize(3) scale(1.5)
>     legend(position(4) cols(1))
```



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

```
. dir style, wide
color-Fuchsia.style    color-SteelBlue.style
color-SeaShell.style   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 [P] `sysdir`.

### 3.9 Provide custom palettes

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*

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 `discard` (see [P] `discard`) or `clear all` (see [D] `clear`) to flush the working memory.

will then be available to `colorpalette` like any other palette. Your program should be designed according to the following principles.

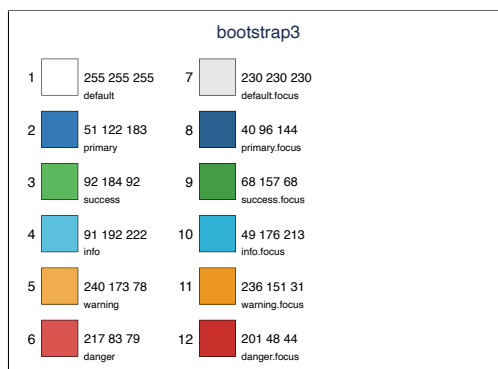
1. The program must return the color definitions as a comma-separated list in local macro `P`. All types of color specifications supported by `colorpalette`, including opacity and intensity operators, are allowed for the individual colors in the list (see section 3.4).
2. If input is parsed using `[P] 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 always 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

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

140 HTML colors

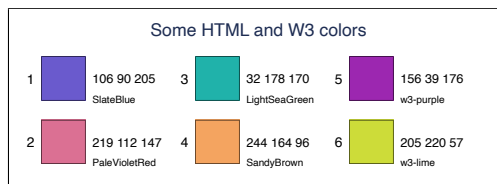
30 W3.CSS default colors

Further color collections from W3.CSS (using names as provided by W3.CSS): Flat UI Colors, Metro UI Colors, Windows 8 Colors, iOS Colors, U.S. Highway Colors, U.S. 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 and 4.2 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, which can then be used, for example, in a graph command (see section 3.6). Example:

```
. colorpalette SlateBlue
>   PaleVioletRed LightSeaGreen
>   SandyBrown w3-purple w3-lime,
>   title(Some HTML and W3 colors)
>   rows(2)
```



## 4.1 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		



## 4.2 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-win8-lime	w3-win8-green	w3-win8-emerald	w3-win8-teal
w3-win8-cyan	w3-win8-blue	w3-win8-cobalt	w3-win8-indigo
w3-win8-violet	w3-win8-pink	w3-win8-magenta	w3-win8-crimson
w3-win8-red	w3-win8-orange	w3-win8-amber	w3-win8-yellow
w3-win8-brown	w3-win8-olive	w3-win8-steel	w3-win8-mauve
w3-win8-taupe	w3-win8-sienna		

### w3 ios

w3-ios-dark-blue	w3-ios-deep-blue	w3-ios-blue	w3-ios-light-blue
w3-ios-green	w3-ios-pink	w3-ios-red	w3-ios-orange
w3-ios-yellow	w3-ios-grey	w3-ios-light-grey	w3-ios-background

**w3 highway**

w3-highway-brown	w3-highway-red	w3-highway-orange	w3-highway-schoolbus
w3-highway-yellow	w3-highway-green	w3-highway-blue	

**w3 safety**

w3-safety-red	w3-safety-orange	w3-safety-yellow	w3-safety-green
w3-safety-blue	w3-safety-purple		

**w3 signal**

w3-signal-yellow	w3-signal-orange	w3-signal-red	w3-signal-violet
w3-signal-blue	w3-signal-green	w3-signal-grey	w3-signal-brown
w3-signal-white	w3-signal-black		

**w3 2019**

w3-2019-fiesta	w3-2019-jester-red	w3-2019-turmeric	w3-2019-living-coral
w3-2019-pink-peacock	w3-2019-pepper-stem	w3-2019-aspen-gold	w3-2019-princess-blue
w3-2019-toffee	w3-2019-mango-mojito	w3-2019-terrarium-moss	w3-2019-sweet-lilac
w3-2019-soybean	w3-2019-eclipse	w3-2019-sweet-corn	w3-2019-brown-granite
w3-2019-chili-pepper	w3-2019-biking-red	w3-2019-creme-de-peche	w3-2019-peach-pink
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-food-apple	w3-food-asparagus	w3-food-apricot	w3-food-aubergine
w3-food-avocado	w3-food-banana	w3-food-butter	w3-food-blueberry
w3-food-carrot	w3-food-cherry	w3-food-chocolate	w3-food-cranberry
w3-food-coffee	w3-food-egg	w3-food-grape	w3-food-kiwi
w3-food-lemon	w3-food-lime	w3-food-mango	w3-food-mushroom
w3-food-mustard	w3-food-mint	w3-food-olive	w3-food-orange
w3-food-pea	w3-food-peach	w3-food-pear	w3-food-pistachio
w3-food-plum	w3-food-pomegranate	w3-food-pumpkin	w3-food-raspberry
w3-food-saffron	w3-food-salmon	w3-food-spearmint	w3-food-squash
w3-food-strawberry	w3-food-tomato	w3-food-wheat	w3-food-wine

**w3 camo**

w3-camo-brown	w3-camo-red	w3-camo-olive	w3-camo-field
w3-camo-earth	w3-camo-sand	w3-camo-tan	w3-camo-sandstone
w3-camo-dark-green	w3-camo-forest	w3-camo-light-green	w3-camo-green
w3-camo-dark-gray	w3-camo-gray	w3-camo-black	

**w3 ana**

w3-ana-501	w3-ana-502	w3-ana-503	w3-ana-504
w3-ana-505	w3-ana-506	w3-ana-507	w3-ana-508
w3-ana-509	w3-ana-510	w3-ana-511	w3-ana-512
w3-ana-513	w3-ana-514	w3-ana-515	w3-ana-516
w3-ana-601	w3-ana-602	w3-ana-603	w3-ana-604
w3-ana-605	w3-ana-606	w3-ana-607	w3-ana-608
w3-ana-609	w3-ana-610	w3-ana-611	w3-ana-612
w3-ana-613	w3-ana-614	w3-ana-615	w3-ana-616
w3-ana-617	w3-ana-618	w3-ana-619	w3-ana-620
w3-ana-621	w3-ana-622	w3-ana-623	w3-ana-624
w3-ana-625	w3-ana-626	w3-ana-627	w3-ana-628

**w3 traffic**

w3-traffic-yellow	w3-traffic-orange	w3-traffic-red	w3-traffic-purple
w3-traffic-green	w3-traffic-blue	w3-traffic-grey	w3-traffic-white
w3-traffic-black			

## 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);<sup>2</sup> 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; and 3) color generators that construct colors based on specific equations and parameter settings.

Full palette names are given below. Note that the names can be abbreviated and typed in lowercase letters when you call 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 generators is used.

### 5.1 Palettes

#### 5.1.1 Stata palettes

The Stata palettes are named after the graphics scheme (see [G-4] **Schemes intro**) in which the colors are used. The palettes are as follows:

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

Palette **s2** is the default used by `colorpalette` if no palette is specified.

2. 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.

### 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 Statistical Software Components Archive. The following palettes have been constructed after some of these contributions:



Eight colorblind-friendly colors suggested by Okabe and Ito (2002); these colors are used in schemes by Bischof (2017b).



Like **okabe**, but including an additional gray as suggested at <http://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).



Fifteen 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.



Six 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 <https://fivethirtyeight.com/>.



Seven 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.



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



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

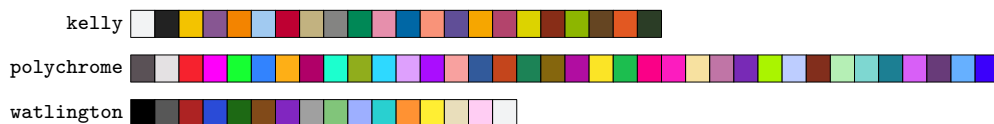


Fifteen gray scales used for areas in plots 1 to 15 in schemes **lean1** and **lean2** by Juul (2003).

### 5.1.3 Categorical palettes from pals

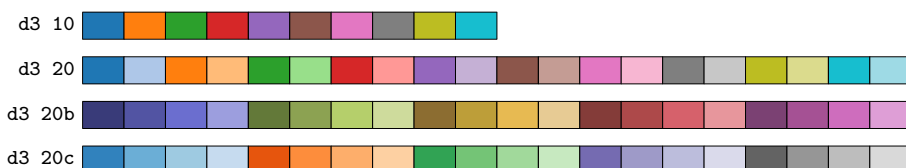
The **pals** collection provides categorical palettes that have been obtained from the **pals** package in R; see <http://github.com/kwstat/pals>. The palettes are as follows:



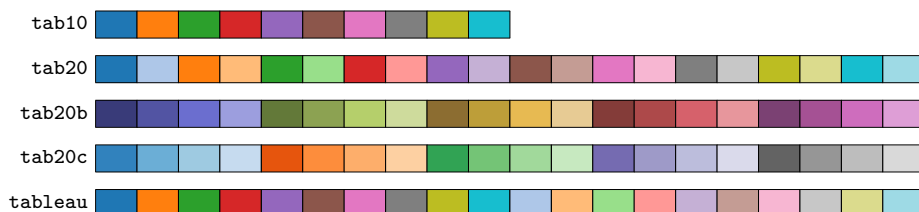


### 5.1.4 D3.js palettes

The d3 collection provides color schemes from <http://d3js.org> using the color values found at GitHub. The schemes are as follows:

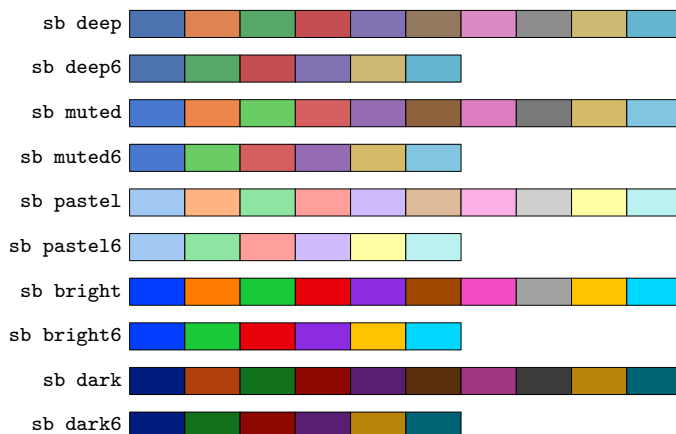


Typing `d3` without an argument is equivalent to `d3 10`. The palettes appear to originate from an earlier version of Tableau and are thus also provided as `tab10`, `tab20`, `tab20b`, and `tab20c`. Furthermore, a variant on `d3 20` is palette `tableau` (same colors but in different order), which has been obtained from Lin et al. (2013).



### 5.1.5 Qualitative palettes from seaborn

The `sb` collection provides categorical color schemes from <https://seaborn.pydata.org/> (same basic colors as `tab10` but in different tones). The schemes are as follows:













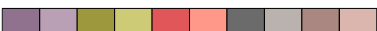




```
sb colorblind 
sb colorblind6 
```

Typing `sb` without an argument is equivalent to `sb deep`; typing `sb6` is equivalent to `sb deep6`.

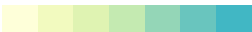






### 5.1.6 Tableau 10 color schemes

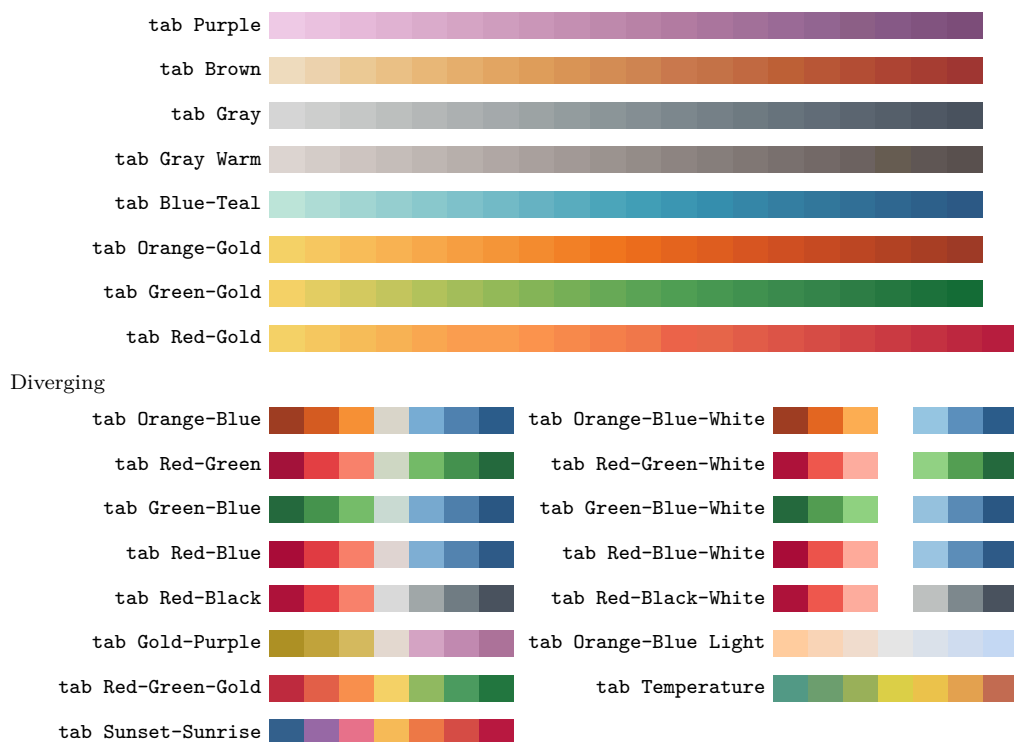
The `tab` collection provides various color schemes from Tableau 10 (the color values have been obtained from the `ggthemes` package in R). The schemes are as follows (typing `tab` without an argument is equivalent to `tab 10`):

Qualitative

```
tab 10 
tab 20 
tab Color Blind 
tab Seattle Grays 
tab Traffic 
tab Miller Stone 
tab Superfishel Stone 
tab Nuriel Stone 
tab Jewel Bright 
tab Summer 
tab Winter 
tab Green-Orange-Teal 
tab Red-Blue-Brown 
tab Purple-Pink-Gray 
tab Hue Circle 
```

Sequential

```
tab Blue-Green 
tab Blue Light 
tab Orange Light 
tab Blue 
tab Orange 
tab Green 
tab Red 
```

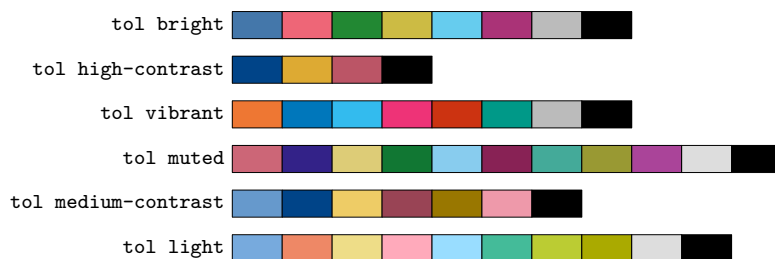


### 5.1.7 Color schemes by Paul Tol

#### New palettes

The `tol` collection provides various color schemes presented by Paul Tol at <https://personal.sron.nl/~pault/>. The schemes are as follows (typing `tol` without an argument is equivalent to `tol muted`):

#### Qualitative

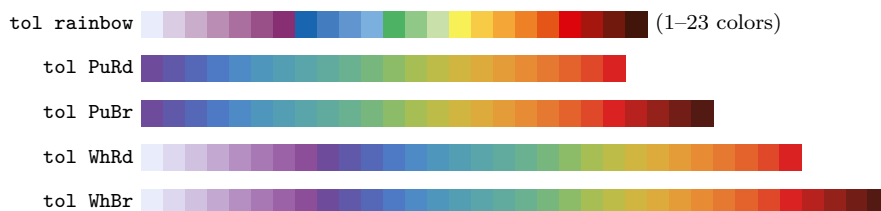


#### Sequential

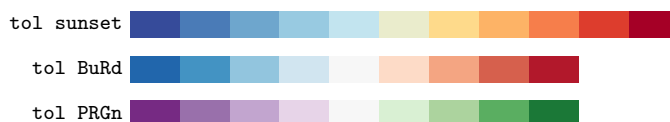




## Rainbow



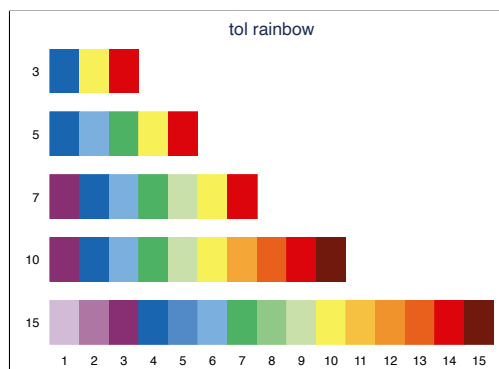
## Diverging



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 <https://personal.sron.nl/~pault/> (for example, with respect to the order of colors in the qualitative schemes).

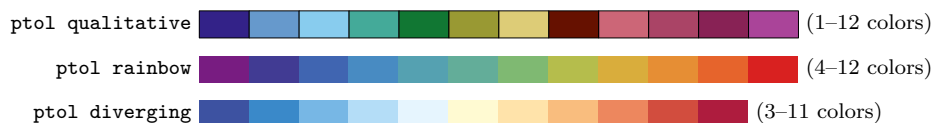
For `tol rainbow`, the selection of colors depends on the size of the palette, as illustrated in the following example:

```
. 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 (2021)

The `ptol` collection provides color schemes as suggested by Tol (2021). The schemes are as follows:



Typing `ptol` without an argument is equivalent to `ptol qualitative`. The selection of colors depends on the size of the palette; displayed is the variant using the maximum number of colors.

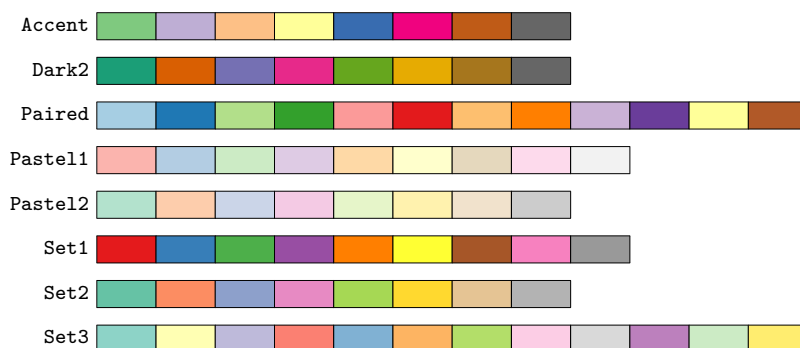
### 5.1.8 ColorBrewer palettes

ColorBrewer is a set of color schemes developed by Brewer, Hatchard, and Harrower ([2003]; also see Brewer [2015]). For more information on ColorBrewer, also see <http://colorbrewer2.org/>.<sup>3</sup> The syntax for the ColorBrewer palettes is

*scheme* [*cmyk*]

where argument *cmyk* selects the CMYK variant (the default is to use the RGB variant) and *scheme* is one of the following (the maximum number of colors is displayed for those schemes that come in different sizes).

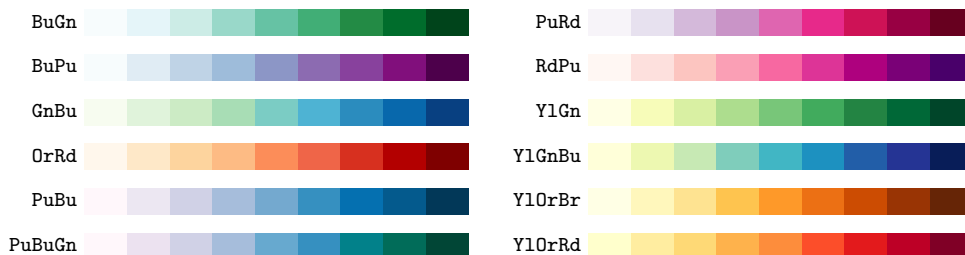
#### Qualitative



#### Single-hue sequential (3–9 colors)

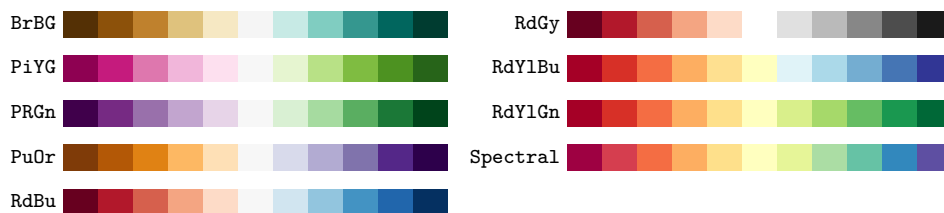


#### Multihue sequential (3–9 colors)



3. The colors are licensed under Apache License Version 2.0; see the copyright notes at [http://www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer\\_updates.html](http://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 [http://www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer\\_RGB.html](http://www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer_RGB.html). The CMYK values have been taken from file `cb.csv` provided at <https://github.com/axismaps/colorbrewer/>. ColorBrewer palettes for Stata have also been provided by Gomez (2015) and by Buchanan (2015).

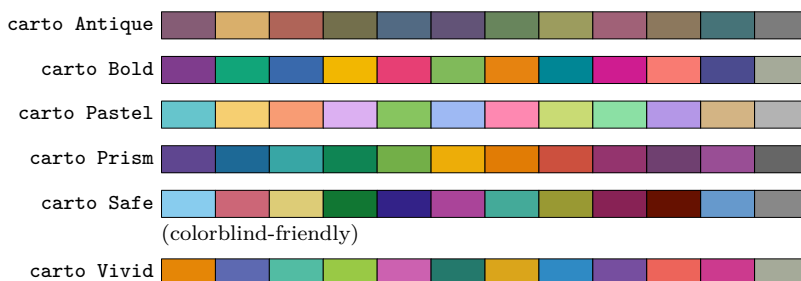
## Diverging (3–11 colors)



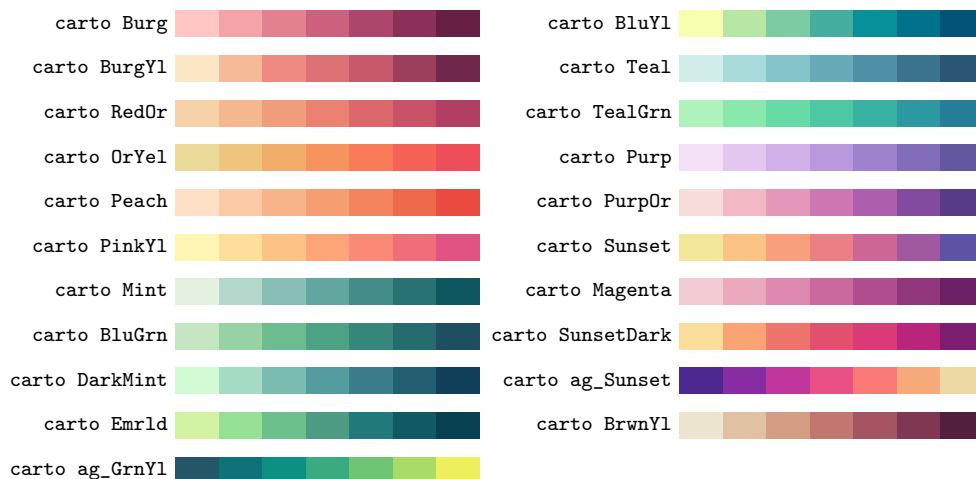
## 5.1.9 Color schemes from Carto

The `carto` collection provides various color schemes from Carto (using color codes from `cartocolor.js` at <https://github.com/CartoDB/>). The schemes are as follows:

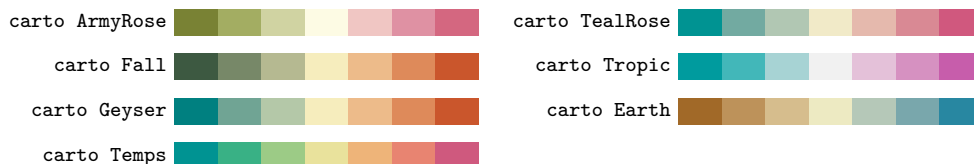
## Qualitative (2–11 colors, plus one color for missing data)



## Sequential (2–7 colors)



Diverging (2–7 colors)



Typing `carto` without an argument is equivalent to `carto Bold`. The selection of colors depends on the size of the palette; displayed is the variant using the maximum number of colors.

### 5.1.10 Semantic colors by Lin et al. (2013)

The `lin` collection provides semantic color schemes suggested by Lin et al. (2013).<sup>4</sup> The syntax is

```
lin [scheme [algorithm]]
```

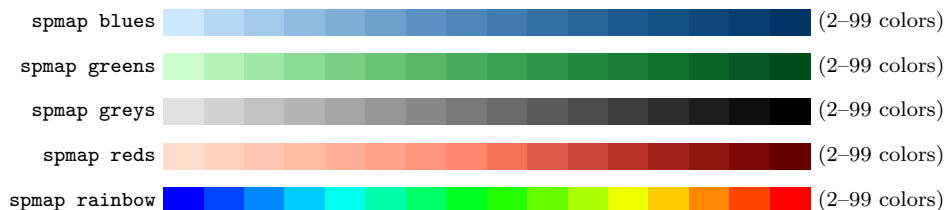
where *scheme* is one of the following,

<b>carcolor</b>	6 car colors; the default	<b>food</b>	7 food colors
<b>features</b>	5 feature colors	<b>activities</b>	5 activity colors
<b>fruits</b>	7 fruit colors	<b>vegetables</b>	7 vegetable colors
<b>drinks</b>	7 drink colors	<b>brands</b>	7 brand colors

and argument `algorithm` requests algorithm-selected colors. The default is to return the colors selected by Turkers (in the case of `carcolor`, `food`, `features`, `activities`) or by the expert (in the case of `fruits`, `vegetables`, `drinks`, `brands`). Figure 1 displays the schemes, including labels (“T” stands for “Turkers”, “E” for “Expert”, and “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 schemes are as follows (typing `spmap` without an argument is equivalent to `spmap blues`; displayed are 16-color variants):



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).

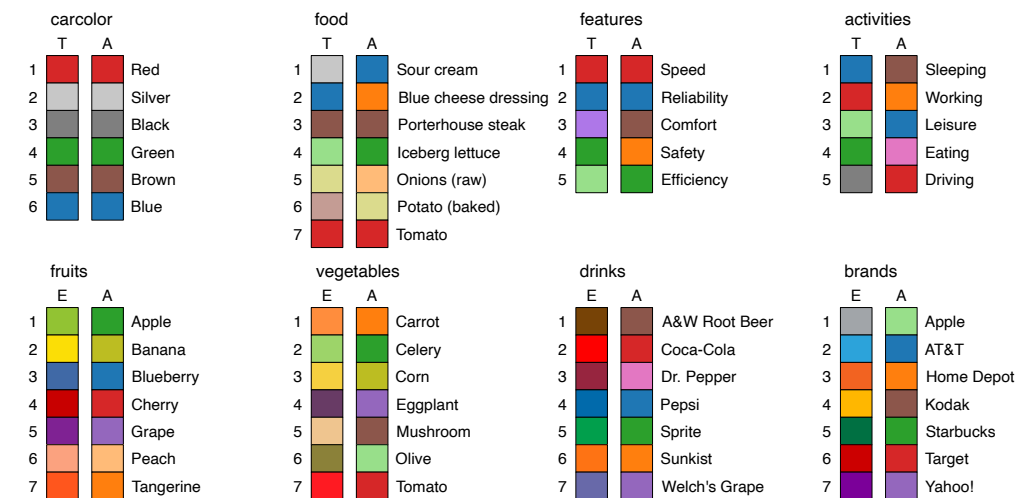
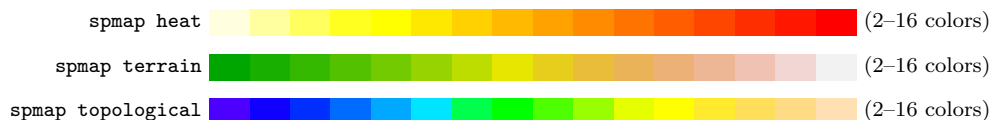


Figure 1. Semantic color schemes by Lin et al. (2013)



### 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 [cmk]]
```

where argument **cmk** requests the CMYK variant (the default is to use the RGB variant) and **sfso scheme** is one of the following:

Qualitative

**sfso languages**

colors used for languages (German, French, Italian, Rhaeto-Romanic, English)

**sfso parties**

colors used for Swiss parties (FDP, CVP, SP, SVP, GLP, BDP, Grüne, small left-wing parties, small middle parties, small rightwing parties, other parties)

Sequential

**sfso brown**

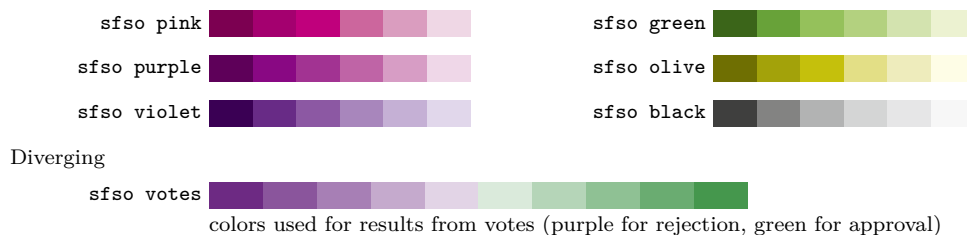
**sfso blue**

**sfso orange**

**sfso ltblue**

**sfso red**

**sfso turquoise**



Typing `sfso` without an argument is equivalent to `sfso blue`. When using `sfso blue` for color gradients, select colors 1–6 only; the 7th color is an extra color for special purposes.

### 5.1.13 HTML colors

The HTML collection provides HTML colors from <https://www.w3schools.com/>. The schemes are as follows:

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

See section 4.1 for an overview of the colors in these schemes. Also see [https://www.w3schools.com/colors/colors\\_names.asp](https://www.w3schools.com/colors/colors_names.asp) or [https://www.w3schools.com/colors/colors\\_groups.asp](https://www.w3schools.com/colors/colors_groups.asp). Typing `HTML` without an argument returns all 148 HTML colors (alphabetically sorted).

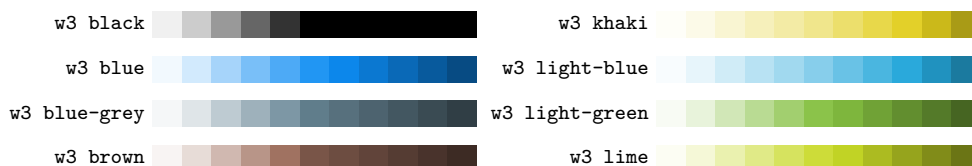
### 5.1.14 W3.CSS colors

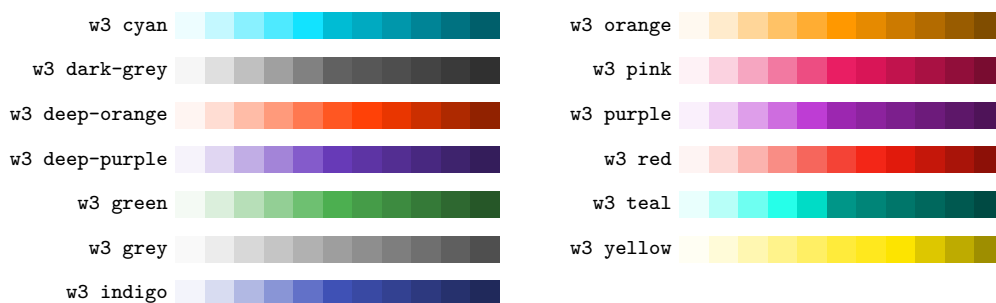
The `w3` collection provides color schemes from W3.CSS. The schemes are as follows (typing `w3` without an argument is equivalent to `w3 default`):

Qualitative collections (see section 4.2)

w3 default	30 Default Colors	w3 flat	20 Flat UI Colors
w3 metro	17 Metro UI Colors	w3 win8	22 Windows 8 Colors
w3 ios	12 iOS Colors	w3 highway	7 U.S. Highway Colors
w3 safety	6 U.S. Safety Colors	w3 signal	10 European Signal Colors
w3 2019	32 Fashion Colors 2019	w3 2018	30 Fashion Colors 2018
w3 2017	20 Fashion Colors 2017	w3 vivid	21 Vivid Colors
w3 food	40 Food Colors	w3 camo	15 Camouflage Colors
w3 ana	44 Army Navy Aero Colors	w3 traffic	9 Traffic Colors

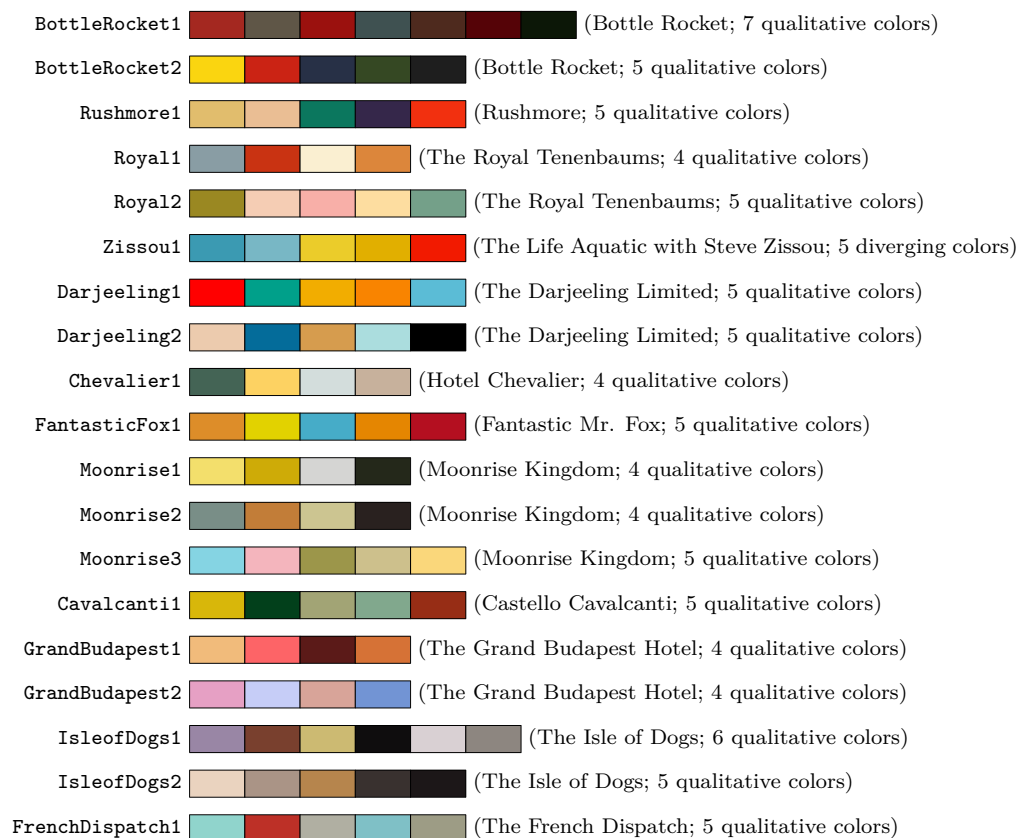
Sequential themes (11 colors)





### 5.1.15 Wes Anderson palettes

The Wes Anderson collection provides palettes from <http://wesandersonpalettes.tumblr.com> (the color codes have been obtained from <http://github.com/karthik/wesanderson>). The palettes are as follows:



## 5.2 Colormaps

Colormaps are palettes used to generate color gradients with an arbitrary number of colors. The general syntax is

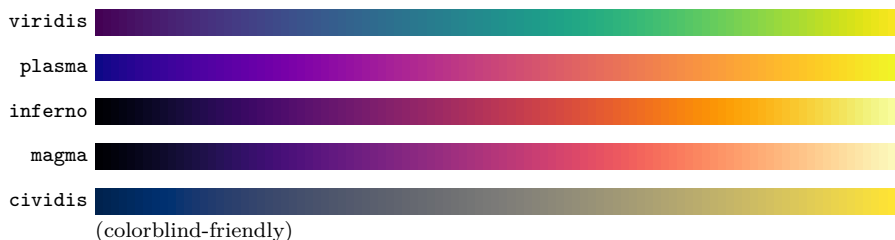
```
palette [, range(lb [ub]) ...]
```

where *palette* is the name of the colormap and option **range()**, with *lb* and *ub* in  $[0, 1]$ , selects the range of the colormap to be used. The default is **range(0 1)**, that is, to use the full range. If *lb* is larger than *ub*, the colors are returned in reverse order. Option **range()** has no effect for cyclic (circular) colormaps.

### 5.2.1 Viridis colormaps

The **viridis** collection provides perceptually uniform colormaps from matplotlib (also see <http://bids.github.io/colormap/>). The color values have been taken from file `_cm_listed.py` at <https://github.com/matplotlib/>. The colormaps are as follows:

Sequential



Cyclic



### 5.2.2 Seaborn colormaps

The **seaborn** collection provides perceptually uniform colormaps from <http://seaborn.pydata.org/>. The colormaps are as follows:

Sequential





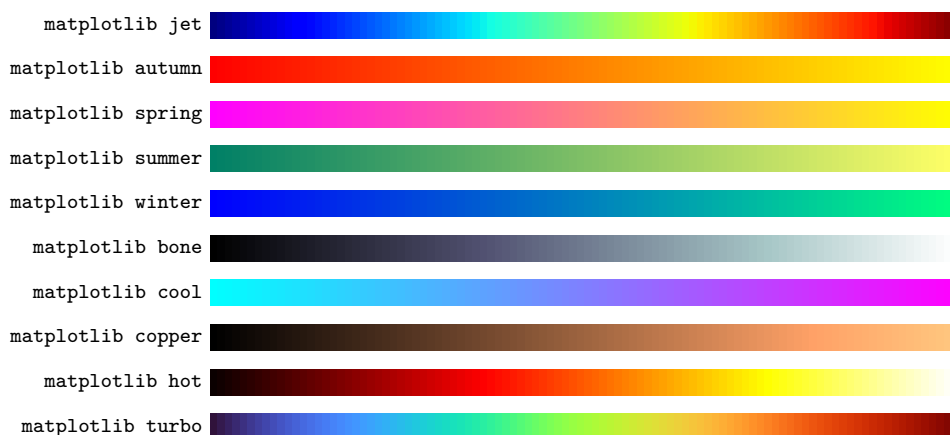
Diverging



### 5.2.3 Other matplotlib colormaps

The `matplotlib` collection provides several colormaps from `matplotlib` (Hunter 2007). The definitions of the colormaps have been taken from file `_cm.py` at <https://github.com/matplotlib/>. The colormaps are as follows (typing `matplotlib` without an argument is equivalent to `matplotlib jet`):

Sequential



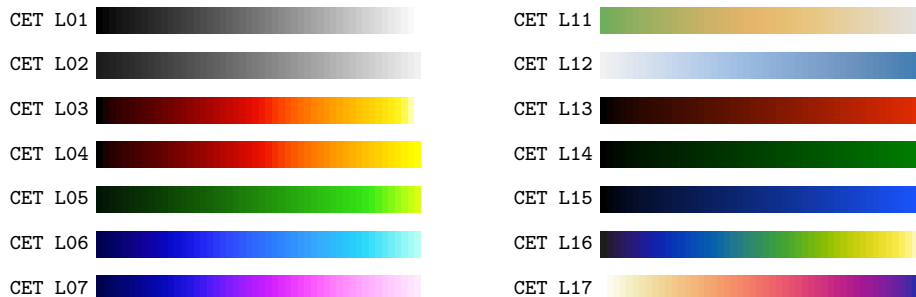
Diverging

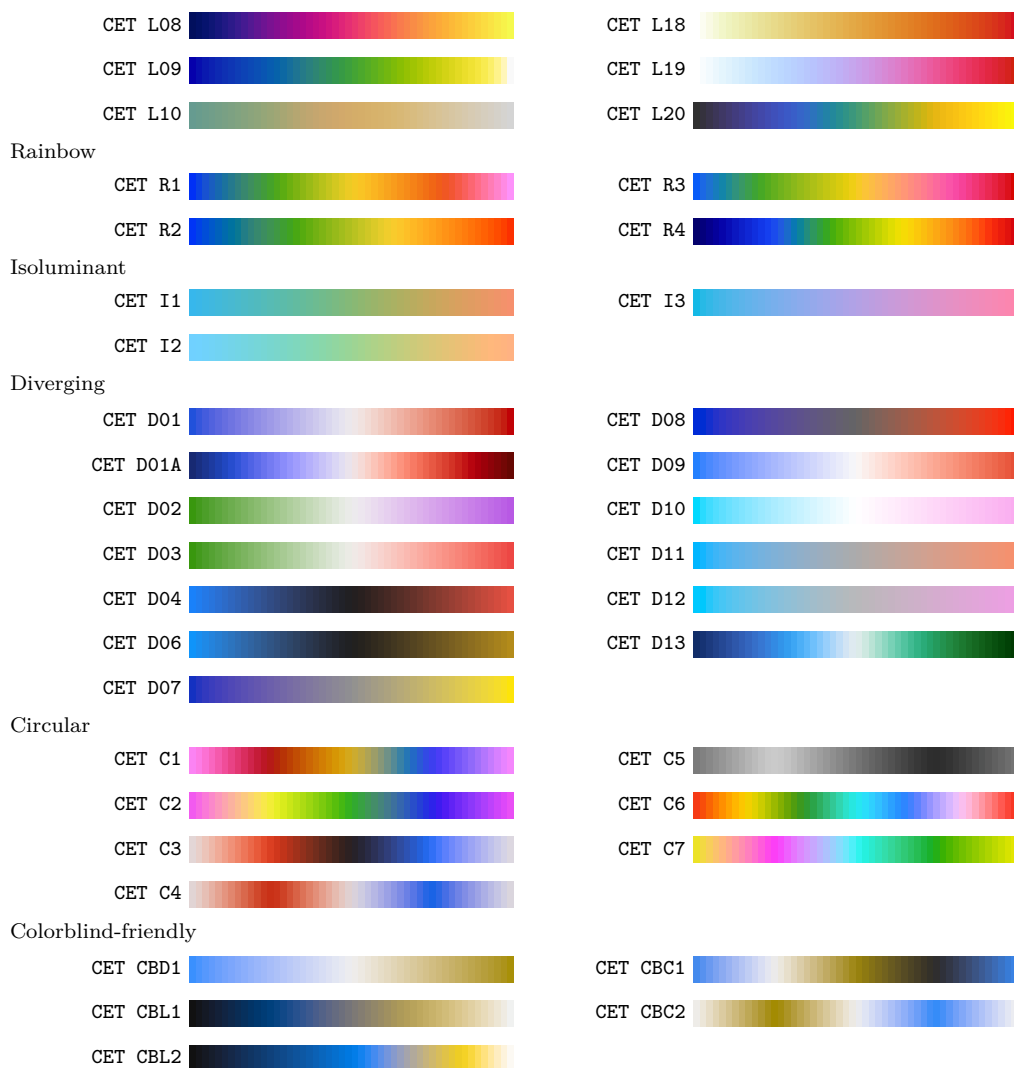


### 5.2.4 Colormaps by Kovesi (2015)

The CET collection provides perceptually uniform colormaps by Kovesi (2015); see <https://colorcet.com/>. The colormaps are as follows (typing `CET` without an argument is equivalent to `CET L20`):

Linear



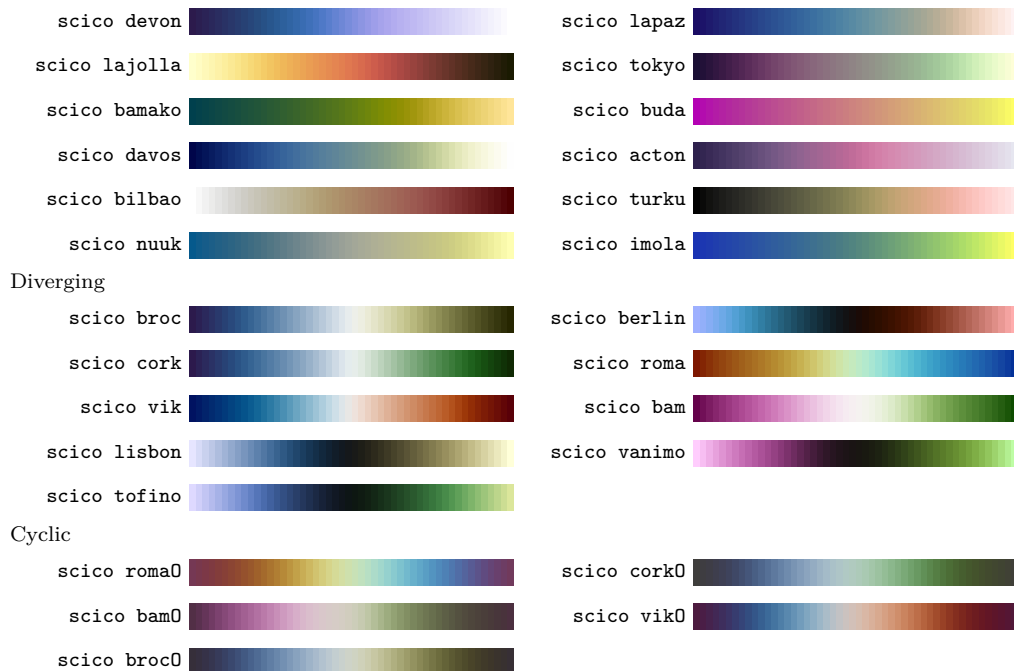


### 5.2.5 Scientific color maps by Crameri (2018)

The `scico` collection provides perceptually uniform colorblind-friendly colormaps by Crameri (2018); see <https://www.fabiocrameri.ch/colourmaps/>. The colormaps are as follows (typing `scico` without an argument is equivalent to `scico batlow`):

#### Sequential



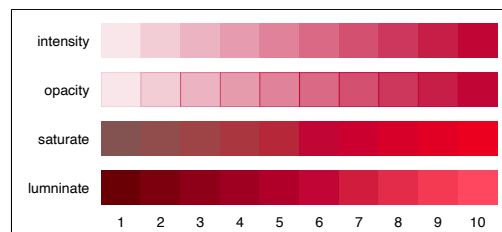


## 5.3 Color generators

### 5.3.1 Generate colors over a range of intensity, opacity, saturation, or luminance levels

The `intensity()` (or `intensify()`), `opacity()`, `saturate()`, and `luminare()` options support number lists as an 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 intensity, opacity, saturation, or luminance levels, as in the following example:

```
. colorpalette, labels(intensity
>   opacity saturate luminare):
>   cranberry, intensity(0.1(.1)1)
>   / cranberry, opacity(10(10)100)
>   / cranberry, saturate(-50(10)40)
>   / cranberry, luminare(-25(5)20)
```

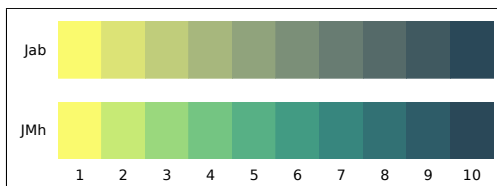


### 5.3.2 Generate colors by interpolation

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 (see the description of the `ipolate()` option for further details)

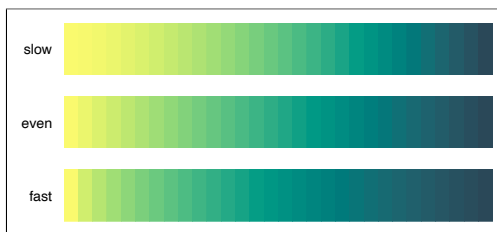
The default is to interpolate in the CIECAM02-based  $J'a'b'$  space, which is supposed to be perceptually uniform and does not travel around the color wheel. If you want to interpolate around the color wheel, you could, for example, use the CIECAM02-based  $J'M'h$  space:

```
. colorpalette, labels(Jab JMh)
>   gropts(ysize(2) scale(2.25)):
>   #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)
>   gropts(ysize(2.5) scale(1.8)):
>   #fafa6e #2a4858,
>   ipolate(30, HCL power(1.5))
>   / #fafa6e #2a4858,
>   ipolate(30, HCL)
>   / #fafa6e #2a4858,
>   ipolate(30, HCL power(0.7))
```

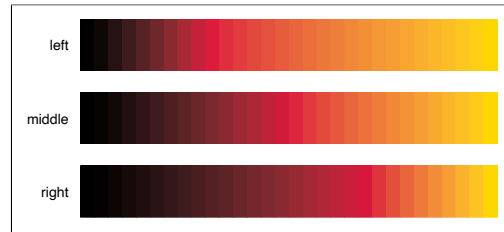


When you interpolate between more than two colors, the default is to arrange the origin colors on a regular grid. This can be changed by the `positions()` suboption. In the following example, the `positions()` suboption is used to shift the middle color to the left or to the right:

```

. colorpalette, nonumbers
>   labels(left middle right)
>   gropts(ysize(2.5) scale(1.8)):
>     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 <https://github.com/hadley/scales>). The syntax is

```
hue [ , parameter_options ... ]
```

where *parameter\_options* are the following:

**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 `hue` palette with default options produces the same colors as the `intense` scheme of the HCL color generator (see below).

### 5.3.4 HCL, LCh, and JMh color generators

The HCL, LCh, and JMh palettes 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. (2022); also see Zeileis, Hornik, and Murrell (2009) and <https://hclwizard.org/>. The LCh and JMh generators are implemented analogously.

Let  $i$  be an index from 1 to  $n$  with  $n$  as the number of requested colors,  $h_1$  and  $h_2$  be two hues on the 360-degree color wheel,  $c_1$  and  $c_2$  be two chroma values,  $l_1$  and  $l_2$  be two luminance values, and  $p_1$  and  $p_2$  be two power parameters. Depending on the type of scheme, the colors are then generated using the following equations:

Scheme	$H_i$	$C_i$	$L_i$	$j$
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	$ j ^{p_1} c_1$	$l_2 -  j ^{p_2}(l_2 - l_1)$	$\frac{n-2j+1}{n-1}$

The color generator syntax selects the color space, the type of scheme, and the parameter settings. It is

`{HCL|LCh|JmH} [scheme] [, parameter_options ...]`

where *parameter\_options* are the following:

`hue`( $h_1$  [ $h_2$ ]) overrides the default values for  $h_1$  and  $h_2$  (hues on the 360-degree color wheel).











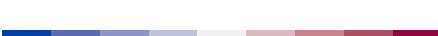
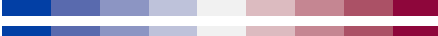
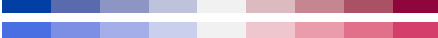
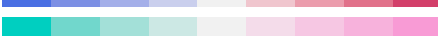
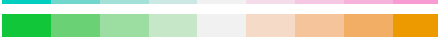
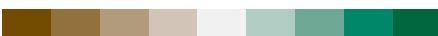

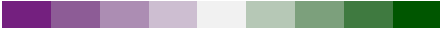

`chroma`( $c_1$  [ $c_2$ ]) overrides the default values for  $c_1$  and  $c_2$ ,  $c \geq 0$  (colorfulness;  $M'$  in the case of JmH).

`luminance`( $l_1$  [ $l_2$ ]) overrides the default values for  $l_1$  and  $l_2$ ,  $l \in [0, 100]$  (brightness;  $J'$  in the 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  $p = 1$ ;  $p > 1$  makes the transition faster, and  $p < 1$  makes the transition slower).

*scheme* sets the type of scheme and the default parameters. The available predefined HCL schemes are as follows (using  $n = 9$ ; typing HCL without an argument is equivalent to HCL qualitative):

	$h_1$	$h_2$	$c_1$	$c_2$	$l_1$	$l_2$	$p_1$	$p_2$	
Qualitative									
HCL qualitative	15	$h^*$	60	—	70	—	—	—	
HCL intense	15	$h^*$	100	—	65	—	—	—	
HCL dark	15	$h^*$	80	—	60	—	—	—	
HCL light	15	$h^*$	50	—	80	—	—	—	
HCL pastel	15	$h^*$	35	—	85	—	—	—	
with $h^* = h_1 + 360(n-1)/n$									
Sequential									
HCL sequential	260	$h_1$	80	10	25	95	1	$p_1$	
HCL blues	260	$h_1$	80	10	25	95	1	$p_1$	
HCL greens	145	125	80	10	25	95	1	$p_1$	

HCL grays	0	$h_1$	0	0	15	95	1	$p_1$	
HCL oranges	40	$h_1$	100	10	50	95	1	$p_1$	
HCL purples	280	$h_1$	70	10	20	95	1	$p_1$	
HCL reds	10	20	80	10	25	95	1	$p_1$	
HCL heat	0	90	100	30	50	90	.2	1	
HCL heat2	0	90	80	30	30	90	.2	2	
HCL terrain	130	0	80	0	60	95	.1	1	
HCL terrain2	130	30	65	0	45	90	.5	1.5	
HCL viridis	300	75	35	95	15	90	.8	1.2	
HCL plasma	100	$h_1$	60	100	15	95	2	.9	
HCL redblue	0	-100	80	40	40	75	1	1	
Diverging									
HCL diverging	260	0	80	-	30	95	1	$p_1$	
HCL bluered	260	0	80	-	30	95	1	$p_1$	
HCL bluered2	260	0	100	-	50	95	1	$p_1$	
HCL bluered3	180	330	60	-	75	95	1	$p_1$	
HCL greenorange	130	45	100	-	70	95	1	$p_1$	
HCL browngreen	55	160	60	-	35	95	1	$p_1$	
HCL pinkgreen	340	128	90	-	35	95	1	$p_1$	
HCL purplegreen	300	128	60	-	30	95	1	$p_1$	

Equivalent schemes are also provided for LCh and JMh, using adjusted parameter values such that the endpoints of the generated colors are similar to the ones generated by HCL (see source file `colrspace_library_generators.sthlp` for details).

### 5.3.5 HSV and HSL color generators

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 `colrspace` by Ihaka et al. (2022). The used formulas are analogous to the formulas of the HCL generator (replacing chroma by saturation and replacing luminance by value or lightness).<sup>5</sup> The syntax is

```
{HSV|HSL} [scheme] [, parameter_options ...]
```

where *parameter\_options* are the following:

`hue( $h_1$  [ $h_2$ ])` overrides the default values for  $h_1$  and  $h_2$  (hues on the 360-degree color wheel).

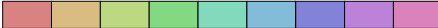


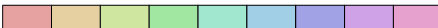













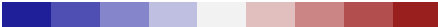
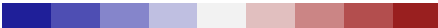
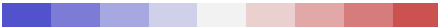
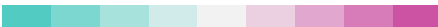
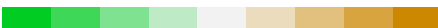
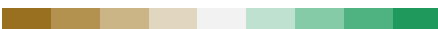
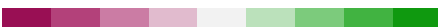
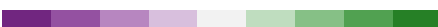
5. HSV `heat0` and HSV `terrain0`, however, use somewhat different formulas; see the source code of `ColrSpace` for details.

`saturation(s1 [s2])` overrides the default values for  $s_1$  and  $s_2$ ,  $s_i \in [0, 1]$  (colorfulness).

`value(v1 [v2])` overrides the default values for  $v_1$  and  $v_2$ ,  $v_i \in [0, 1]$  (brightness).

`power(p1 [p2])` overrides the default values for  $p_1$  and  $p_2$ ,  $p > 0$ , that determine the shape of the transition between saturation and value levels (for linear transitions, set  $p = 1$ ;  $p > 1$  makes the transition faster, and  $p < 1$  makes the transition slower).

`scheme` sets the type of scheme and the default parameters. The available predefined HSV schemes are as follows (using  $n = 9$ ; typing HSV without an argument is equivalent to HSV qualitative):

	$h_1$	$h_2$	$s_1$	$s_2$	$v_1$	$v_2$	$p_1$	$p_2$	
Qualitative									
HSV qualitative	15	$h^*$	.4	—	.85	—	—	—	
HSV intense	15	$h^*$	.6	—	.9	—	—	—	
HSV dark	15	$h^*$	.6	—	.7	—	—	—	
HSV light	15	$h^*$	.3	—	.9	—	—	—	
HSV pastel	15	$h^*$	.2	—	.9	—	—	—	
HSV rainbow	15	$h^*$	1	—	1	—	—	—	
with $h^* = h_1 + 360(n - 1)/n$									
Sequential									
HSV sequential	240	$h_1$	.8	.05	.6	1	1.2	$p_1$	
HSV blues	240	$h_1$	.8	.05	.6	1	1.2	$p_1$	
HSV greens	140	120	1	.1	.3	1	1.2	$p_1$	
HSV grays	0	$h_1$	0	0	.1	.95	1.0	$p_1$	
HSV oranges	30	$h_1$	1	.1	.9	1	1.2	$p_1$	
HSV purples	270	$h_1$	1	.1	.6	1	1.2	$p_1$	
HSV reds	0	20	1	.1	.6	1	1.2	$p_1$	
HSV heat	0	60	1	.2	1	1	0.3	$p_1$	
HSV terrain	120	0	1	0	.65	.95	0.7	1.5	
HSV heat0	0	20	1	0	1	—	—	—	
HSV terrain0	120	0	1	0	.65	.9	—	—	
Diverging									
HSV diverging	240	0	.8	—	.6	.95	1.2	$p_1$	
HSV bluered	240	0	.8	—	.6	.95	1.2	$p_1$	
HSV bluered2	240	0	.6	—	.8	.95	1.2	$p_1$	
HSV bluered3	175	320	.6	—	.8	.95	1.2	$p_1$	
HSV greenorange	130	40	1	—	.8	.95	1.2	$p_1$	
HSV browngreen	40	150	.8	—	.6	.95	1.2	$p_1$	
HSV pinkgreen	330	120	.9	—	.6	.95	1.2	$p_1$	
HSV purplegreen	290	120	.7	—	.5	.95	1.2	$p_1$	



The HSL color generator supports only the schemes **qualitative**, **sequential**, and **diverging**, using adjusted parameter values such that the endpoints of the generated colors are similar to the ones generated by HSL for these schemes.

## 6 Online supplement

In essence, `colorpalette` is a user-friendly interface for `ColrSpace`, a class-based color management system written in Mata. The functionality offered by `colorpalette` should be sufficient in most applied situations, but working with `ColrSpace` directly may be convenient in programming contexts. Extensive documentation of `ColrSpace` can be found in the online supplement available from <https://journals.sagepub.com/doi/suppl/10.1177/10.1177/1536867X231175264>.

## 7 Programs and supplemental materials

Commands `colorpalette` and `colorcheck` are part of the `palettes` package and require `ColrSpace` to be installed on the system. To install `palettes` and `ColrSpace`, type

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

Alternatively, the packages are available from GitHub (see <https://github.com/benjann/palettes> and <https://github.com/benjann/colrspace>). Stata 14.2 or newer is required.

## 8 References

- Bischof, D. 2017a. g538schemes: Stata module to provide graphics schemes for <http://fivethirtyeight.com>. Statistical Software Components S458404, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s458404.html>.
- . 2017b. New graphic schemes for Stata: plotplain and plottig. *Stata Journal* 17: 748–759. <https://doi.org/10.1177/1536867X1701700313>.
- Brewer, C. A. 2015. *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: 5–32. <https://doi.org/10.1559/152304003100010929>.
- Briatte, F. 2013. scheme-burd: Stata module to provide a ColorBrewer-inspired graphics scheme with qualitative and blue-to-red diverging colors. Statistical Software Components S457623, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s457623.html>.

- Buchanan, W. 2015. brewscheme: Stata module for generating customized graph scheme files. Statistical Software Components S458050, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s458050.html>.
- 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. <https://doi.org/10.5281/zenodo.1243862>.
- Gomez, M. 2015. Stata command to generate color schemes. GitHub. <http://github.com/matthieugomez/stata-colorscheme>.
- Hunter, J. D. 2007. Matplotlib: A 2D graphics environment. *Computing in Science and Engineering* 9: 90–95. <https://doi.org/10.1109/MCSE.2007.55>.
- Ihaka, R., P. Murrell, K. Hornik, J. C. Fisher, R. Stauffer, C. O. Wilke, C. D. McWhite, and A. Zeileis. 2022. colorspace: A toolbox for manipulating and assessing colors and palettes. R package version 2.0-3. <https://cran.r-project.org/web/packages/colorspace/>.
- Jann, B. 2018a. Color palettes for Stata graphics. *Stata Journal* 18: 765–785. <https://doi.org/10.1177/1536867X1801800402>.
- . 2018b. Customizing Stata graphs made easy (part 2). *Stata Journal* 18: 786–802. <https://doi.org/10.1177/1536867X1801800403>.
- Juul, S. 2003. Lean mainstream schemes for Stata 8 graphics. *Stata Journal* 3: 295–301. <https://doi.org/10.1177/1536867X0300300306>.
- Kovesi, P. 2015. Good colour maps: How to design them. ArXiv Working Paper No. arXiv:1509.03700. <https://doi.org/10.48550/arXiv.1509.03700>.
- Lin, S., J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. 2013. Selecting semantically-resonant colors for data visualization. *Computer Graphics Forum* 32: 401–410. <https://doi.org/10.1111/cgf.12127>.
- 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: 1291–1298. <https://doi.org/10.1109/TVCG.2009.113>.
- Morris, T. 2013. scheme-mrc: Stata module to provide graphics scheme for UK Medical Research Council. Statistical Software Components S457703, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s457703.html>.
- . 2015. scheme-tfl: Stata module to provide graph scheme, based on Transport for London’s corporate colour palette. Statistical Software Components S458103, Department of Economics, Boston College. <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. <https://jfly.uni-koeln.de/color/>.
- Pisati, M. 2007. spmap: Stata module to visualize spatial data. Statistical Software Components S456812, Department of Economics, Boston College. <https://ideas.repec.org/c/boc/bocode/s456812.html>.
- Tol, P. 2021. Colour schemes. Technical Note SRON/EPS/TN/09-002, SRON. <https://personal.sron.nl/~pault/data/colourschemes.pdf>.
- Zeileis, A., K. Hornik, and P. Murrell. 2009. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics and Data Analysis* 53: 3259–3270. <https://doi.org/10.1016/j.csda.2008.11.033>.

**About the author**

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