DS 116 - Data Visualization

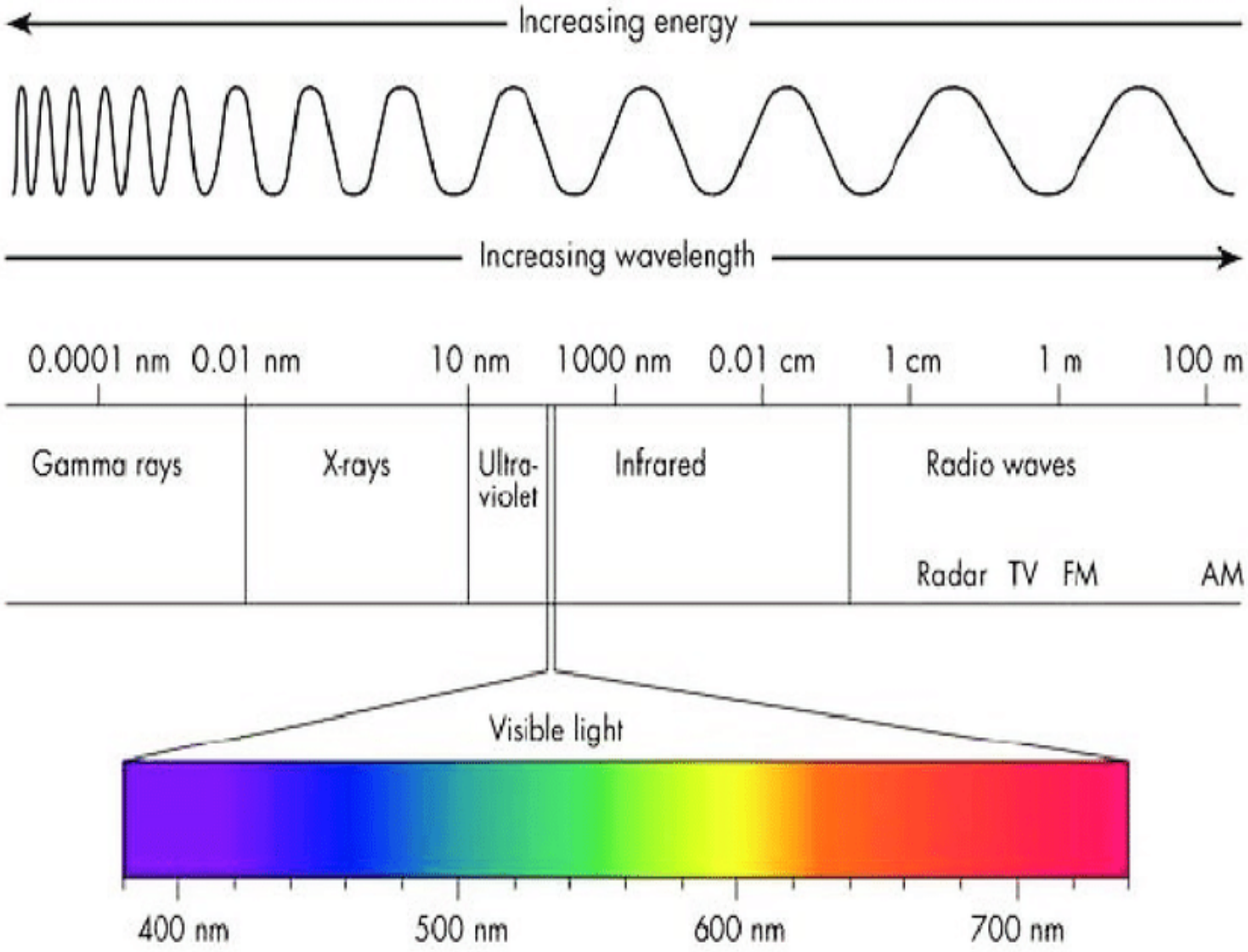
Colors

Habet Madoyan

American University of Armenia

Color

How do we see color ?



Color

- Rods and cones are the receptors in the retina responsible for your sense of sight.
- They are the part of the eye responsible for converting the light that enters your eye into electrical signals that can be decoded by the vision-processing center of the brain.
- Cones are responsible for color vision.

Color

- As a rule, humans have three different types of cones with photo-pigments that sense three different portions of the visible spectrum.
- Each cone is tuned to perceive primarily long wavelengths (red), middle wavelengths (green), or short wavelengths (blue), referred to as L-, M-, and S- cones respectively.

Color

My red is not the same as your red

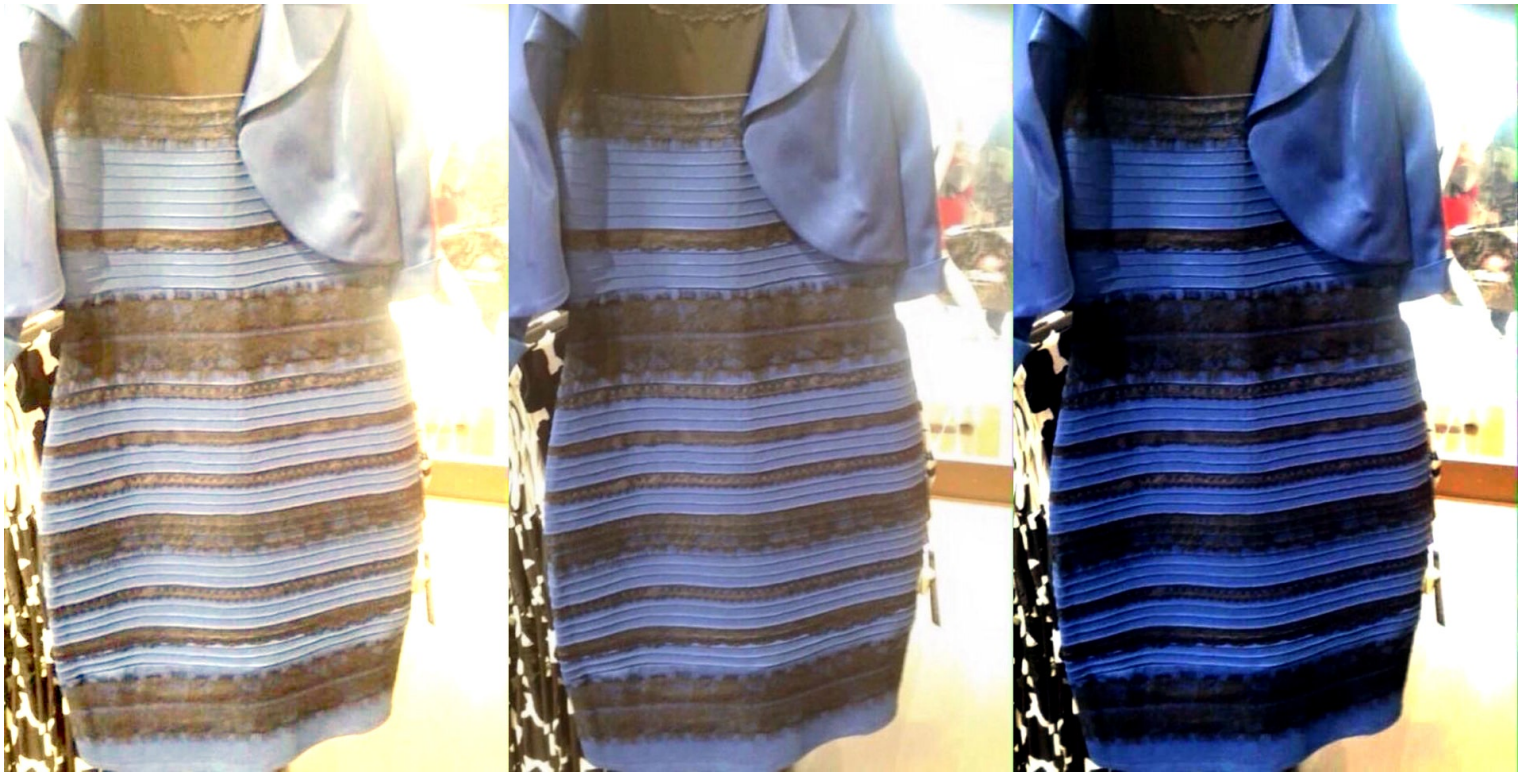
Different people can see the same color differently. Color perception depends on a lot of things, such as distribution of red, green, blue cones, context, etc.
That is how we are ending up with the color blindness
Color blindness can occur when one or more of the cone types are not functioning as expected. Cones can be absent, non-functioning or detect a different color than normal. Red-green color blindness is the most common, followed by blue-yellow color blindness. Men are more likely to have color blindness than women.

Color

Context:
The dress is a photograph that became a viral internet sensation on 26 February 2015, when viewers disagreed over whether the dress pictured was coloured black and royal blue, or white and gold.

Color

Still no consensus. . .



Color

Color perception is a property of an object.

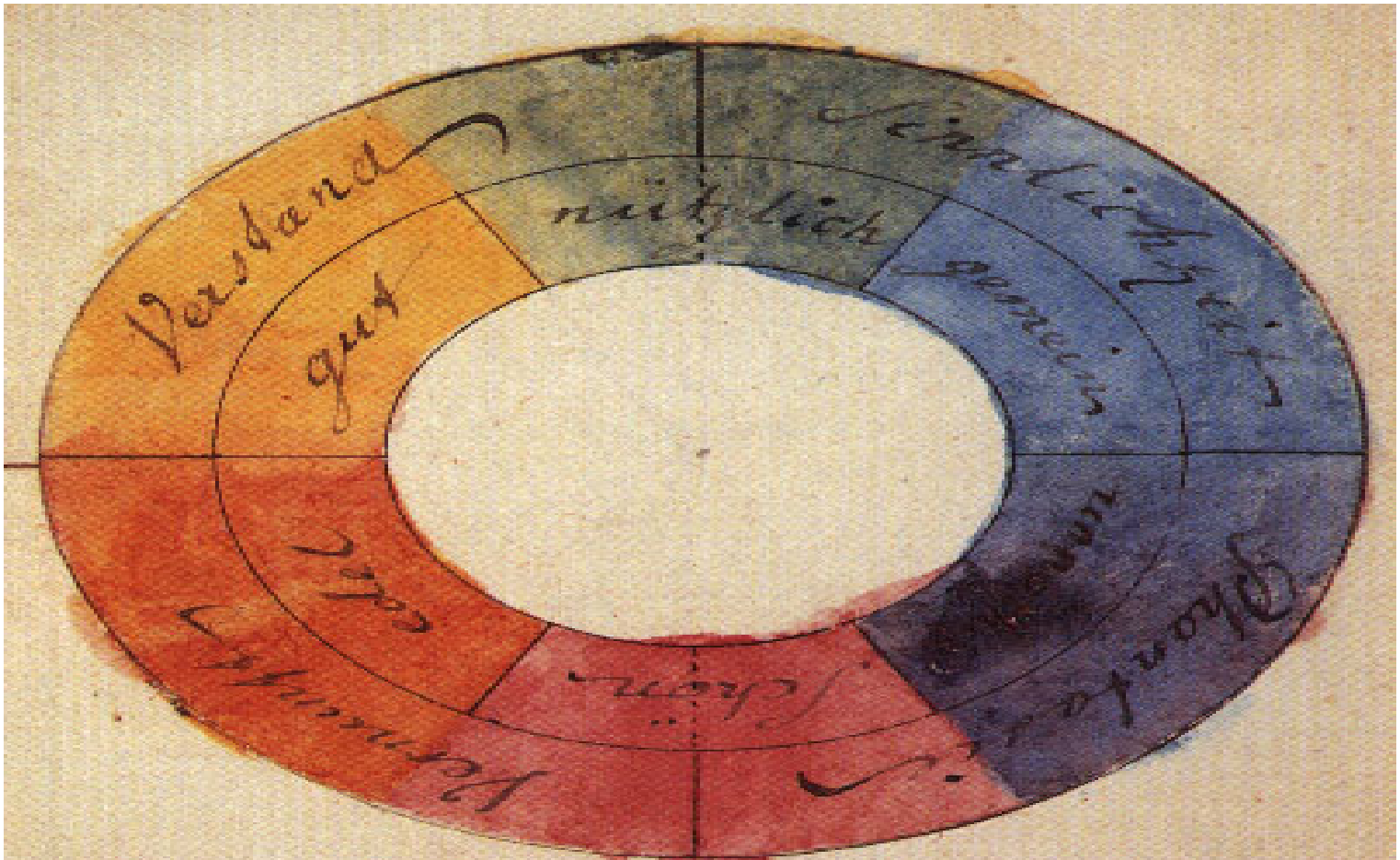
When you look at your white car, your brain instantly identifies it as a white, regardless of how the light is reflected from the car.

Color

A “color model” is an abstract mathematical model describing the way colors can be represented as tuples of numbers (e.g. triples in RGB or quadruples in CMYK). The first pioneers of color theory were Isaac Newton and Johann Wolfgang von Goethe.

Color

Goethe presented a circular diagram (color wheel) in which the three primary colors of red, blue and yellow alternate with the three secondary colors of orange, violet and green.



Color Model:

Color model can be described by primary colors and how other colors are created.
Primary colors are also called primary **hues**.
There are numerous color models:

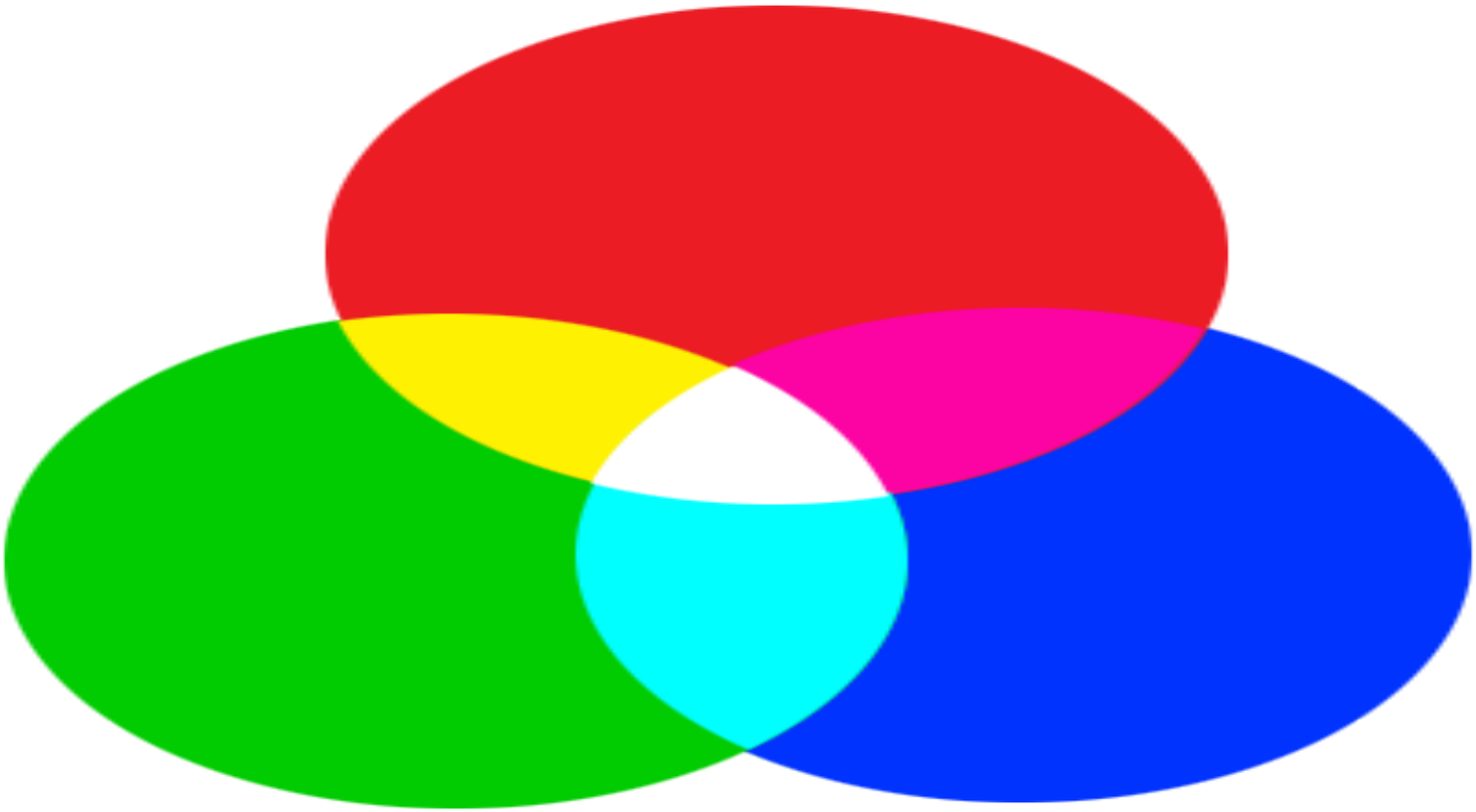
- *RGB* (Red Green Blue) - color model used by computers and for digital visuals
- *CMYK* (Cyan, Magenta, Yellow, Key(Black)) - color model used for printing
- *HCL* - Hue Chroma Luminance and its variations such as HLS, etc

Section 1

Color models

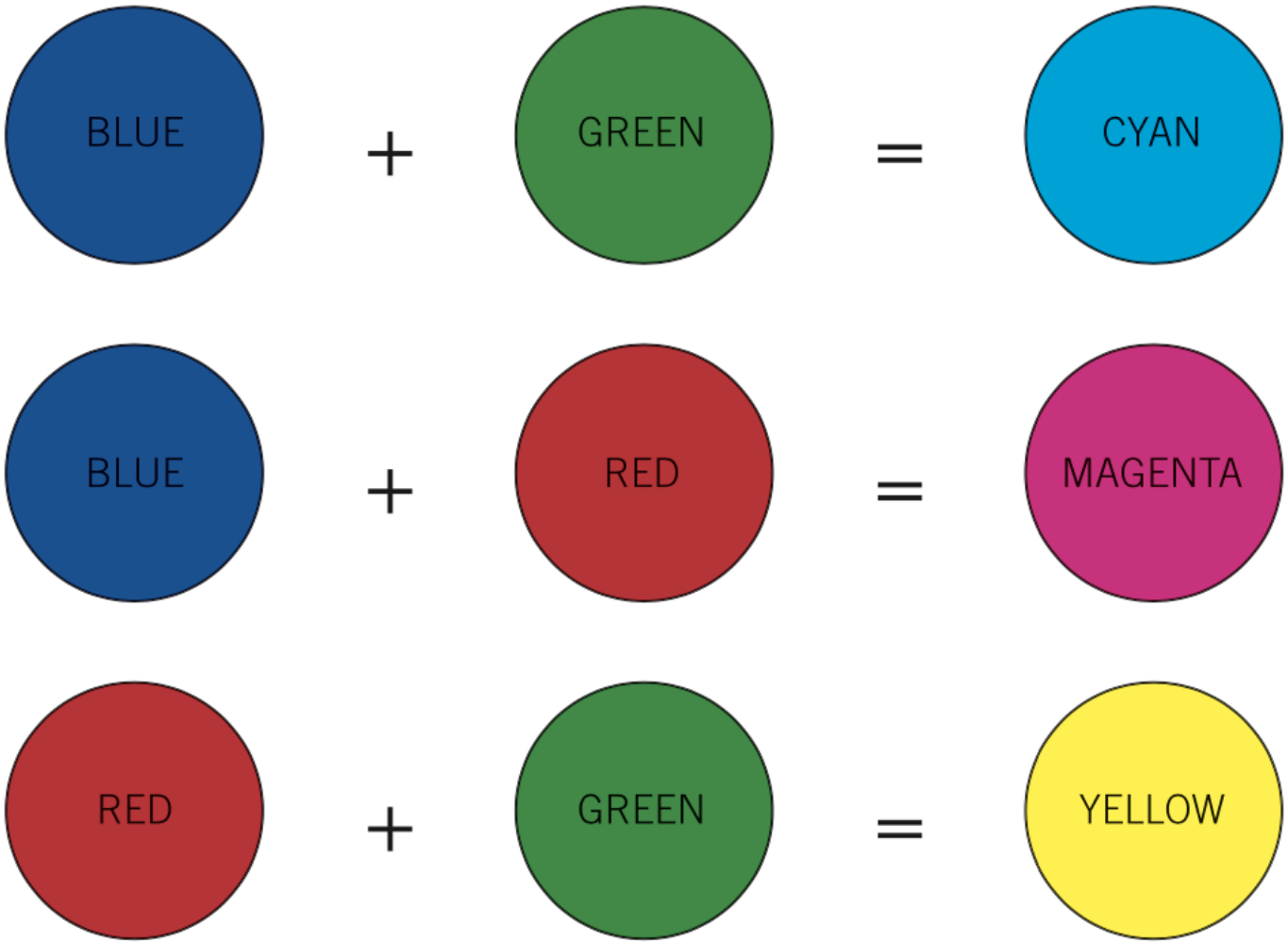
Color model: RGB

RGB: Red, Green, Blue color model is based on the three primary colors of light.



Color Model: RGB

Secondary colors in RGB model are created by adding primary colors to each other:

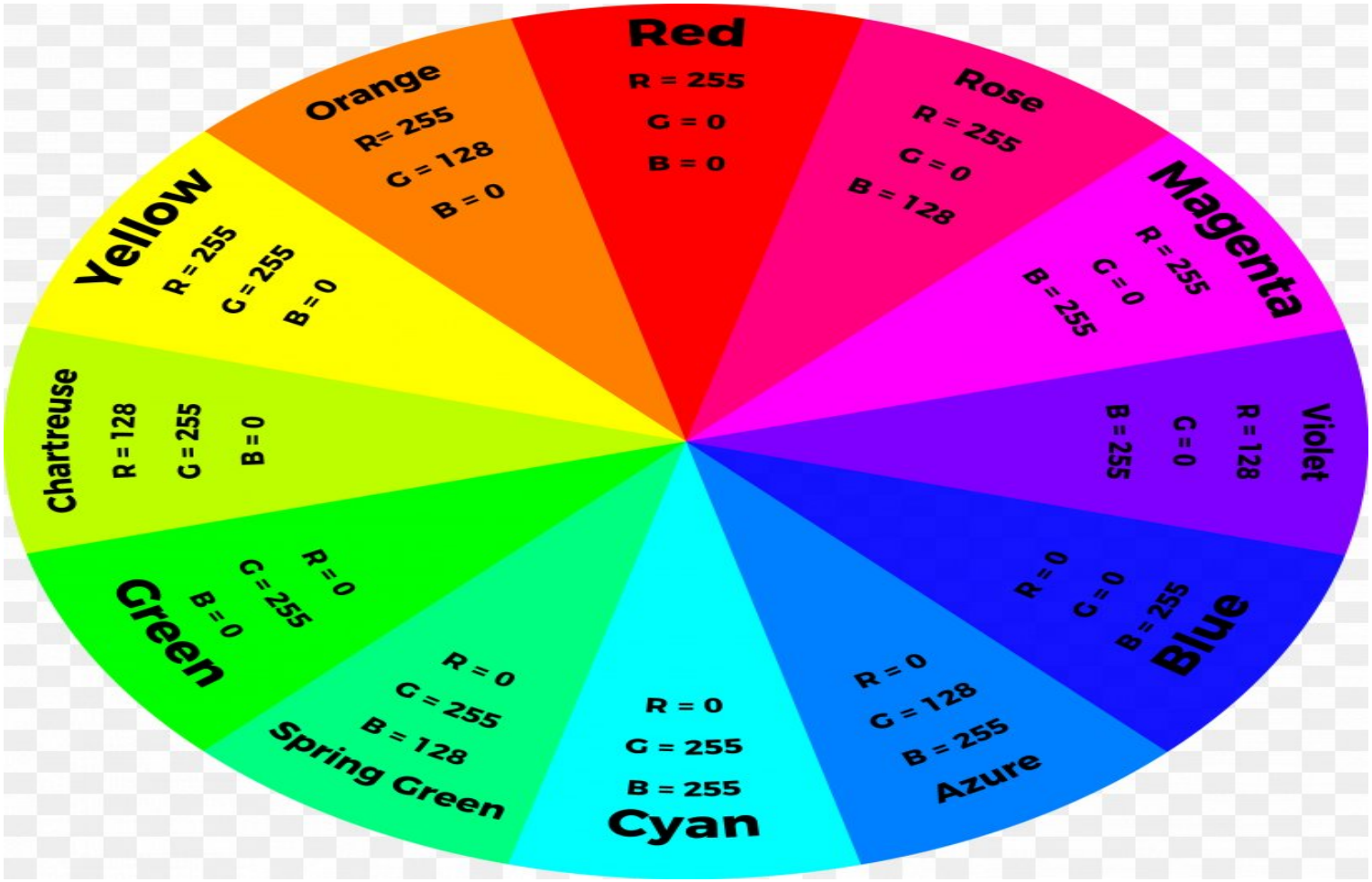


Color Model: RGB

Colors are defined with the values from 0 to 255, where 255 is the highest expression of the color, while 0 is its absence.
That is why white will have an RGB [255,255,255] and black will have RGB of [0,0,0]

Color Model: RGB

Tertiary colors are achieved by a mixture of primary and secondary colors.



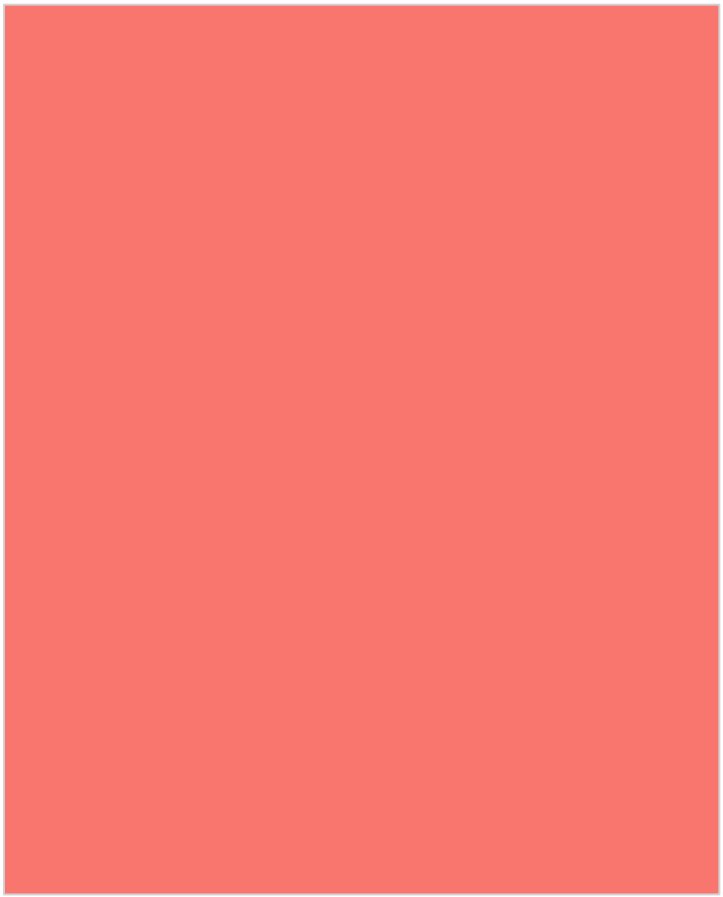
Color Model: RGB

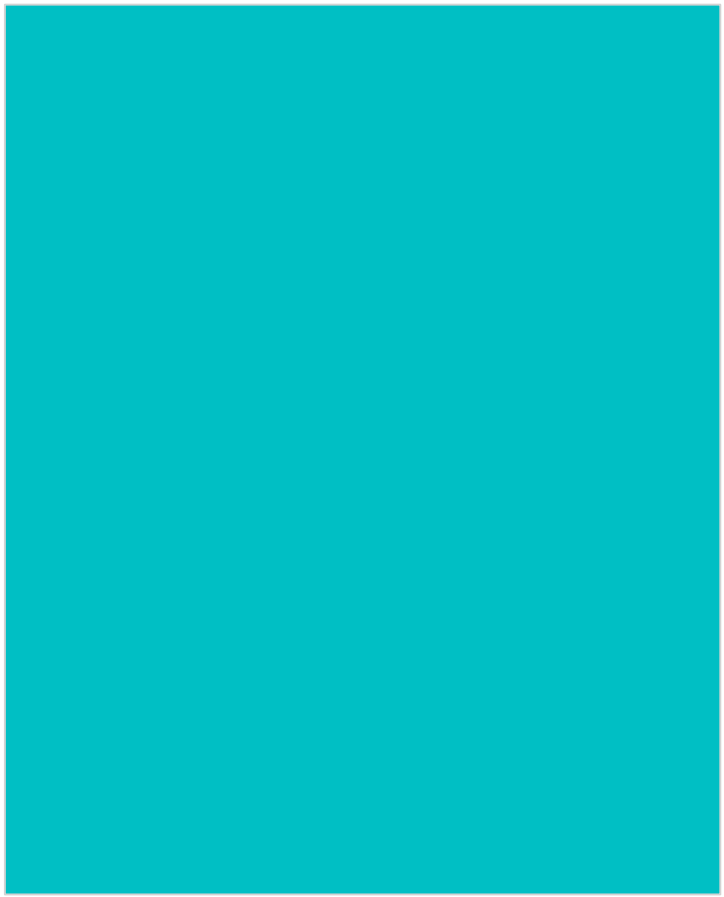
Complementary colors are pairs of colors which, when combined or mixed, cancel each other out by producing a grayscale color like white or black.When placed next to each other, they create the strongest contrast for those two colors. Complementary colors may also be called “opposite” colors.
Complementary colors are located on the opposite sides of the color wheel, examples of complementary colors:

- Cyan - Red
- Green - Magenta
- Yellow - Blue

Color model

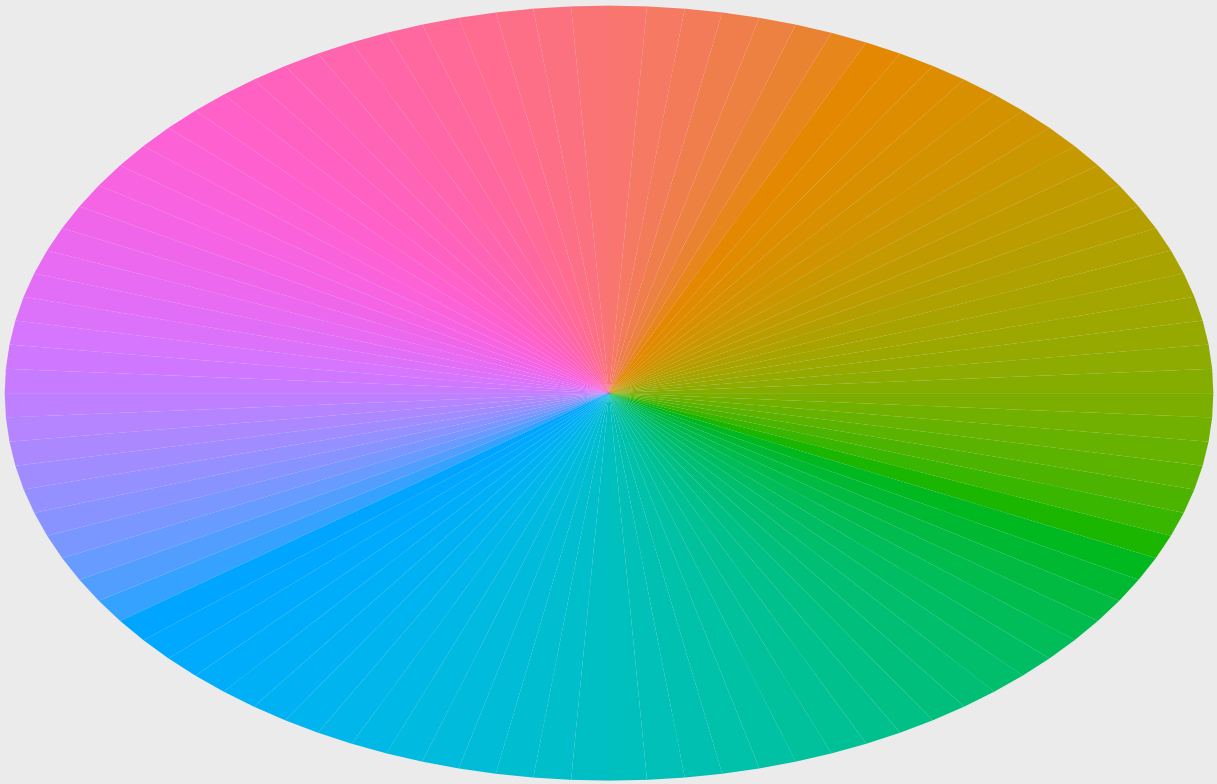
If you are plotting categorical data, then use complementary colors. Here is how ggplot is generating the default palette for categorical variables
If there are only two categories, then choose them as opposite to each other





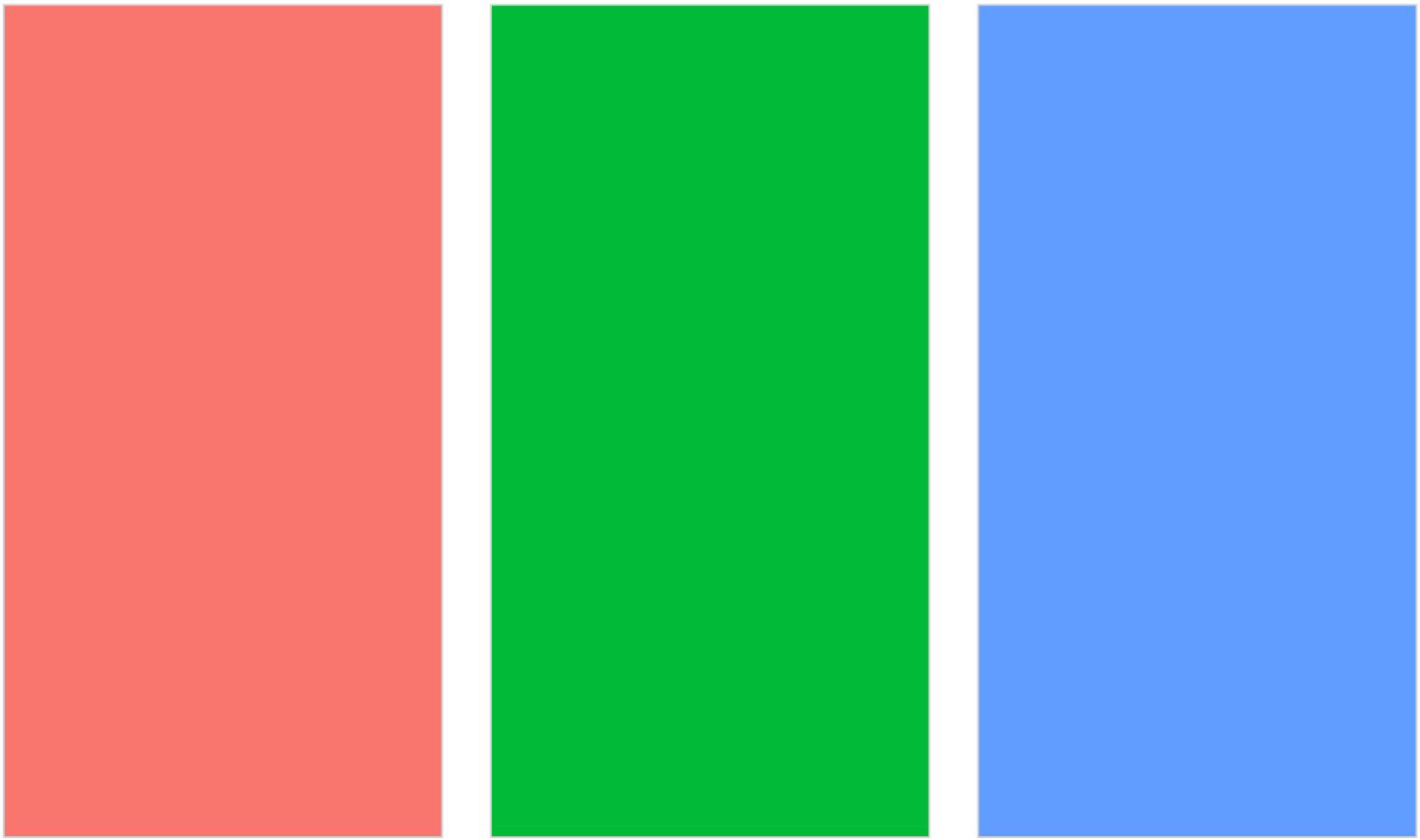
Color model

The color wheel (approximate)



Color model

If there are three categories, then choose the colors in such a way that they will be 120° apart (360/3)



Color model

With 4 categories, 90° apart

HCL model

HCL stands for *Hue*, *Chroma*, *Luminance*
in simpler terms:

- **Hue**: defines the color, the name of the color, or the dominant wavelength
- **Chroma**: defines the colorness
- **Luminance**: defines the brightness

HCL model

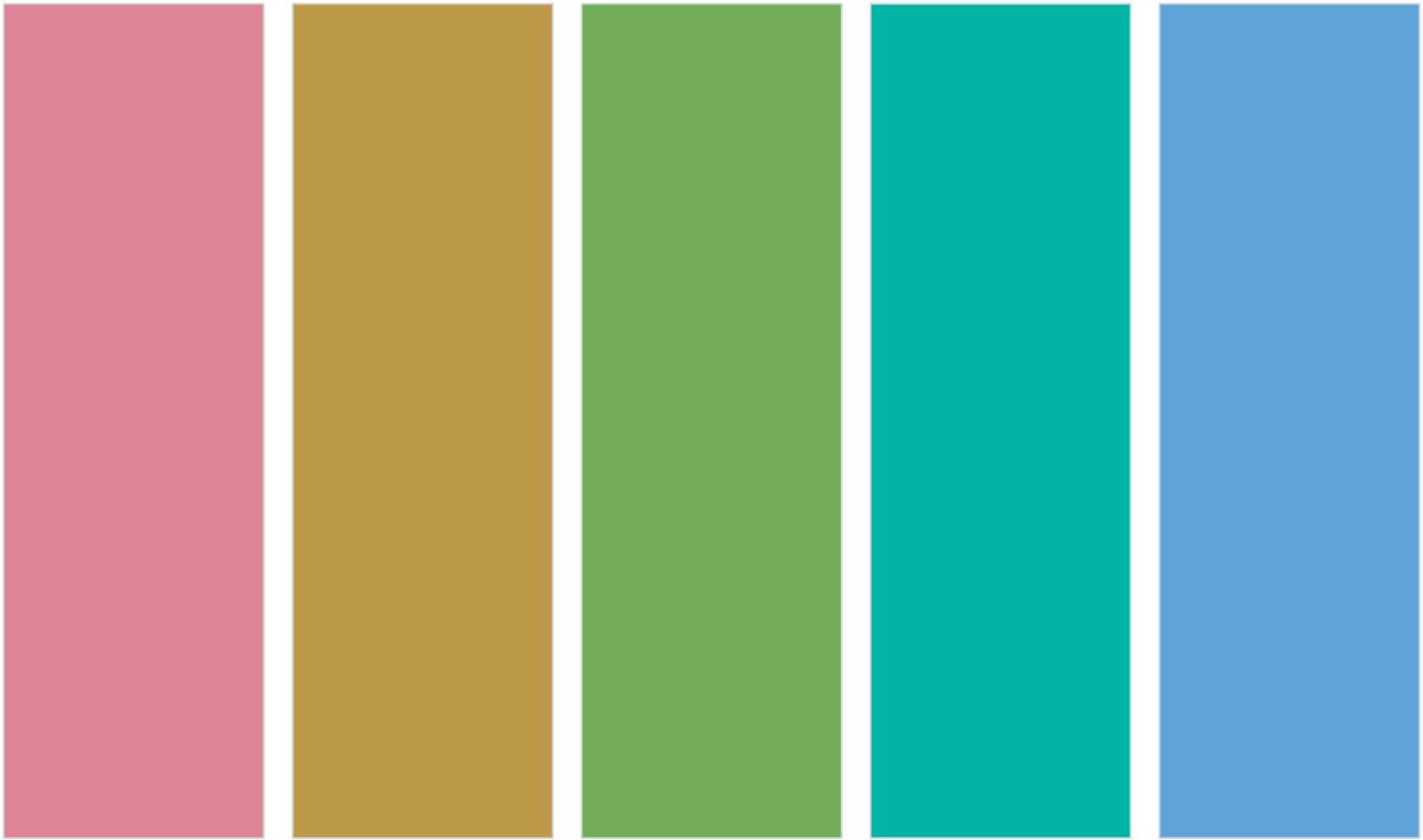
- Hue is a number between 0 and 360 (an angle on the color wheel).
- Chroma is the purity of a colour. A chroma of 0 is gray, and the maximum value of chroma varies with luminance.
- Luminance is the lightness of the colour. A luminance of 0 produces black, and a luminance of 1 produces white.

HCL model: Hue

```
cols <- hcl(h = c(0,60,120,180, 240), c = 60, l = 65)
cols
## [1] "#DF8396" "#BE994A" "#71AD59" "#00B4A5" "#5FA4D9"
```

HCL model: Hue

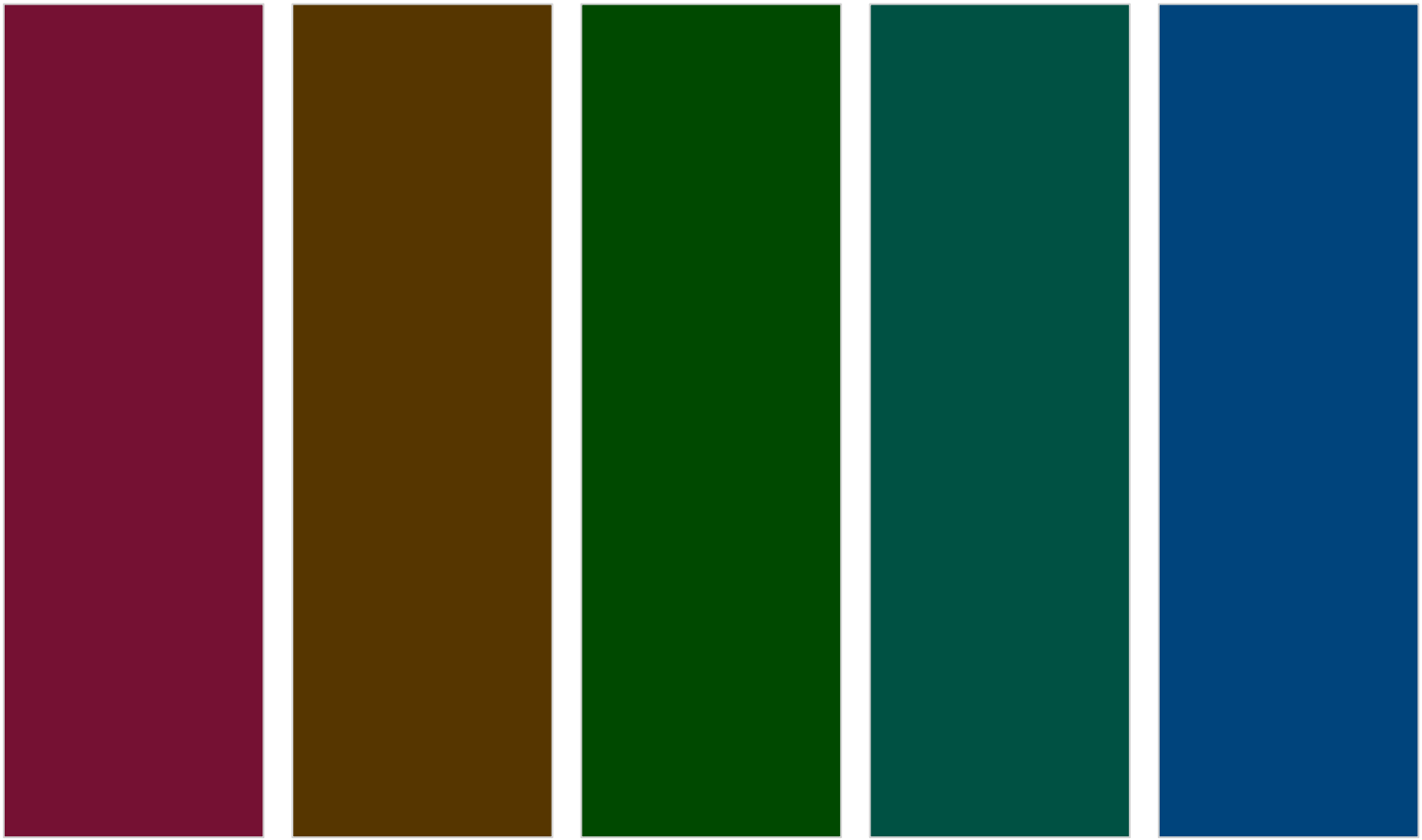
```
swatchplot(cols)
```



HCL model: Luminance

Decreasing the luminance makes the plot darker

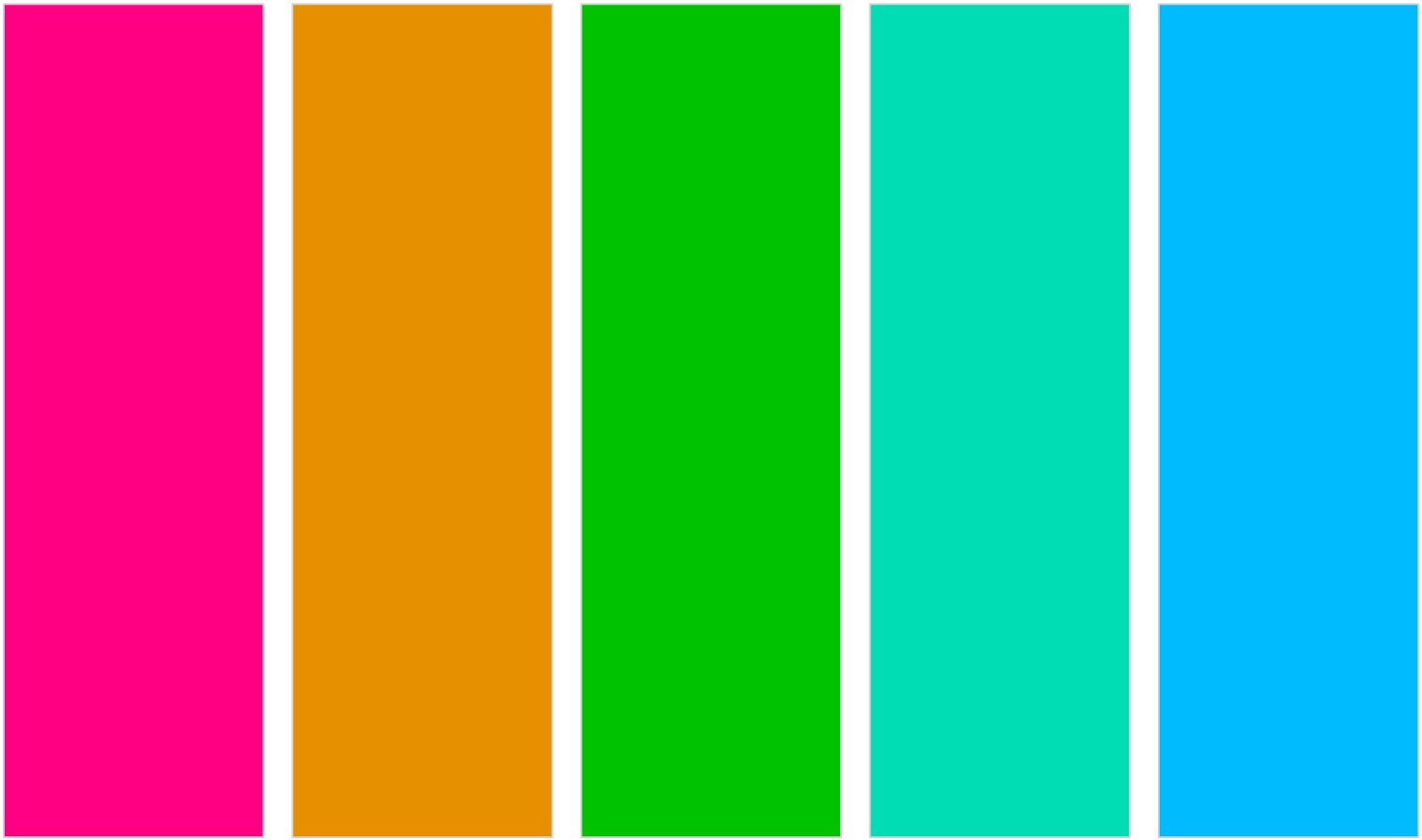
```
cols <- hcl(h = c(0,60,120,180, 240), c = 60, l = 25)
swatchplot(cols)
```



HCL model: Chroma

Increase of chroma brings more colorfulness.

```
cols <- hcl(h = c(0,60,120,180, 240), c = 200, l = 65)
swatchplot(cols)
```



HCL model: Chroma

Setting chroma to 0 will give achromatic color

```
cols <- hcl(h = c(0,60,120,180, 240), c = 0, l = 85)
swatchplot(cols)
```

HCL model

What will happen, if chroma is 0 and you gradually increase the luminance from 0 to 50?

HCL model

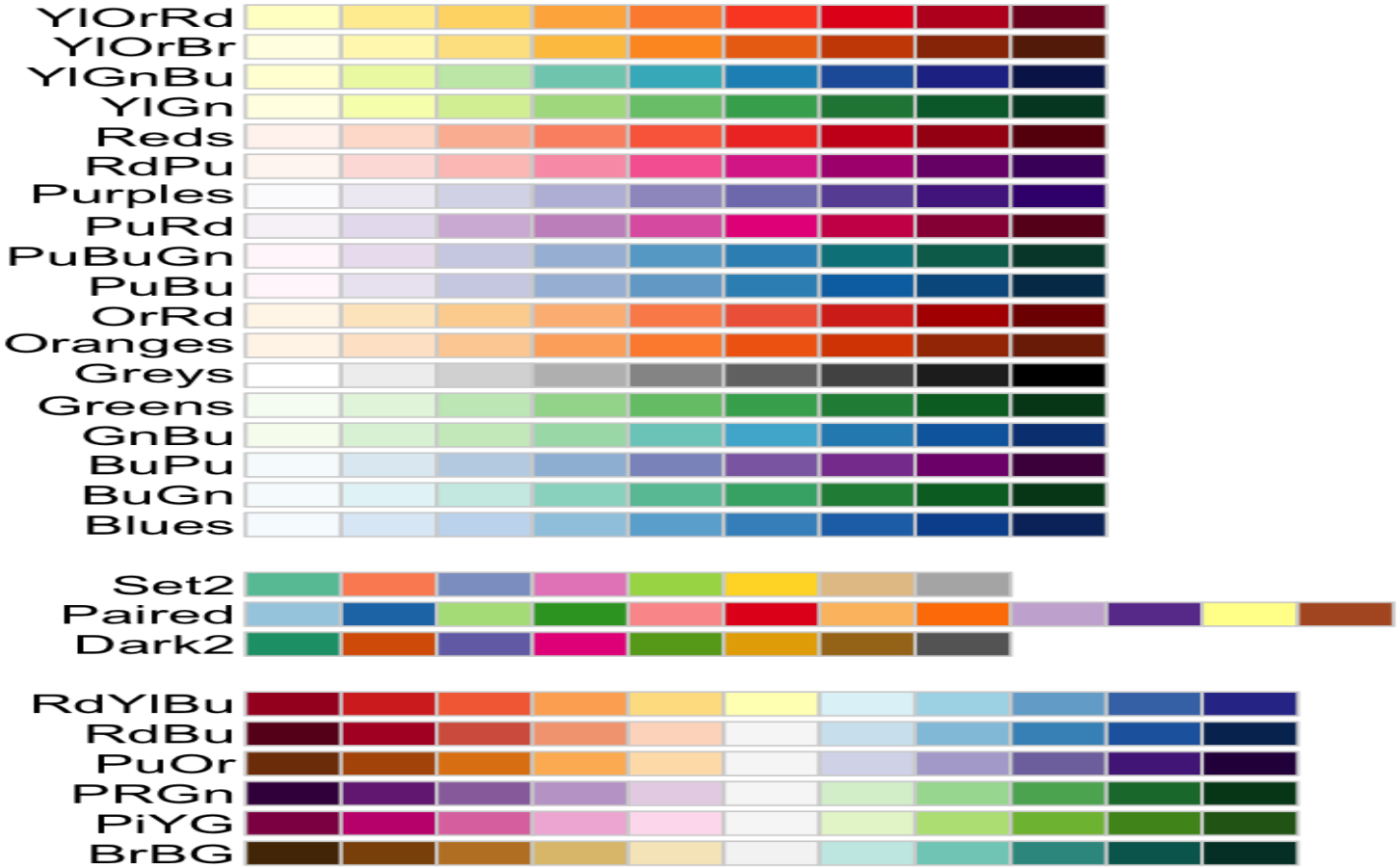
```
cols <- hcl(h = 1, c = 0, l = 0:50)
swatchplot(cols)
```

HCL model: color palettes

1. Sequential palettes, which are suited to ordered data that progress from low to high (gradient) - are used with ordinal data.
2. Qualitative palettes (second list of colors), are best suited to represent nominal or categorical data. They do not imply magnitude differences between groups.
3. Diverging palettes put equal emphasis on mid-range critical values and extremes at both ends of the data range.

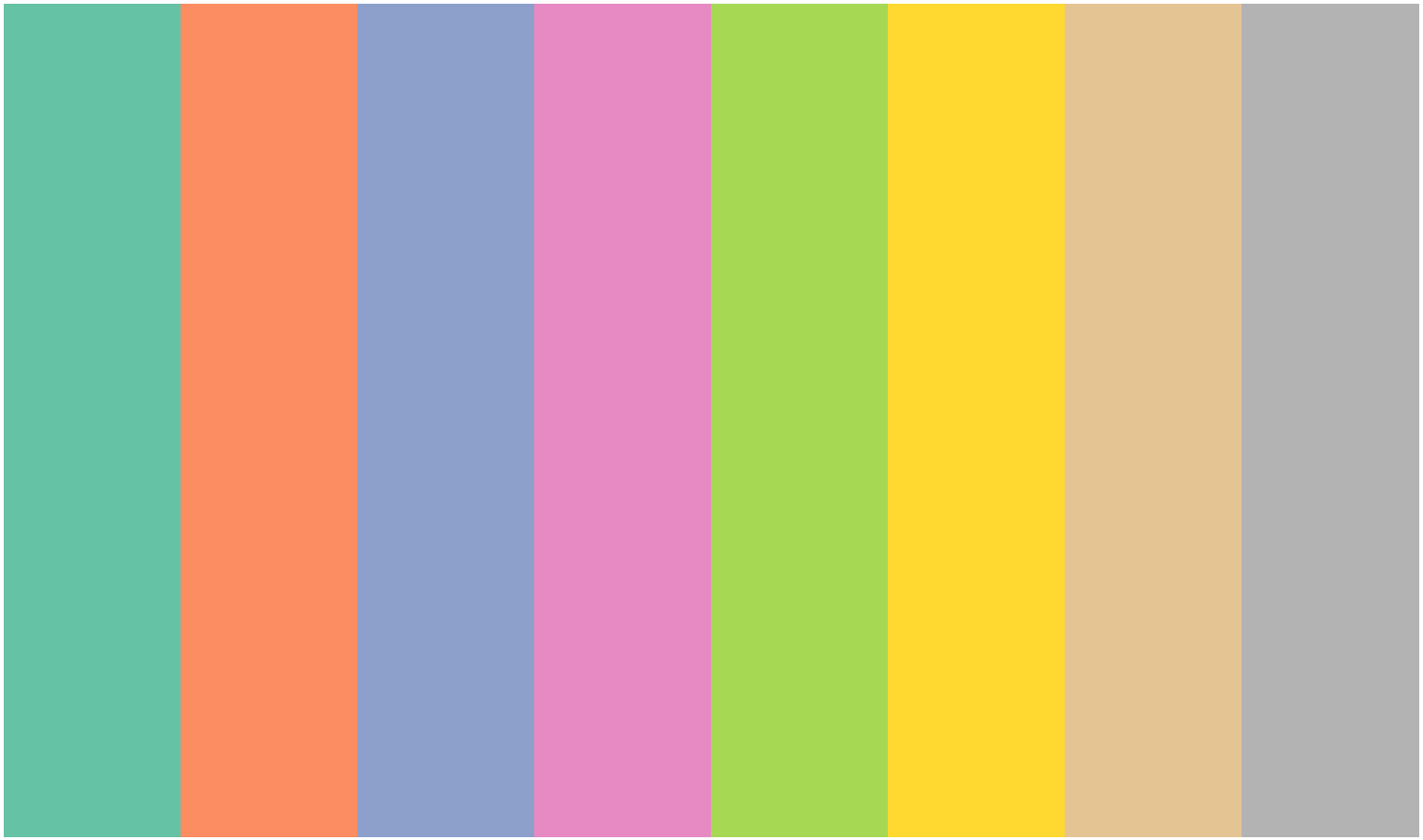
HCL model: color palettes

RColorbrewer provides set of palettes for plotting



HCL model: color palettes

```
display.brewer.pal(n=8, 'Set2')
```



Set2 (qualitative)

HCL model: color palettes

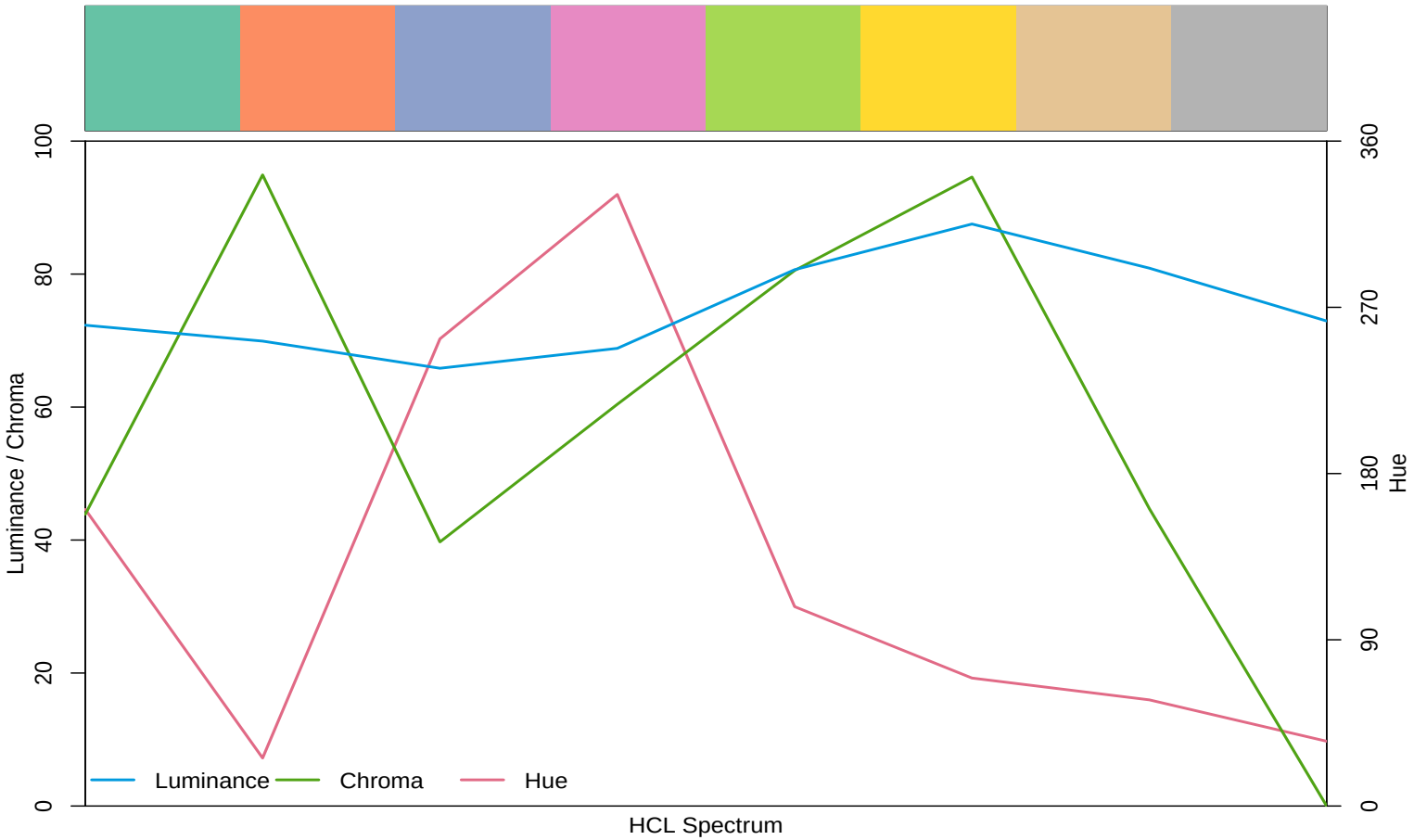
Get the hex codes for the palettes

```
brewer.pal(8, 'Set2')
## [1] "#66C2A5" "#FC8D62" "#8DA0CB" "#E78AC3" "#A6D854" "#FFD92F" "#E5C494
## [8] "#B3B3B3"
```

HCL model: color palettes

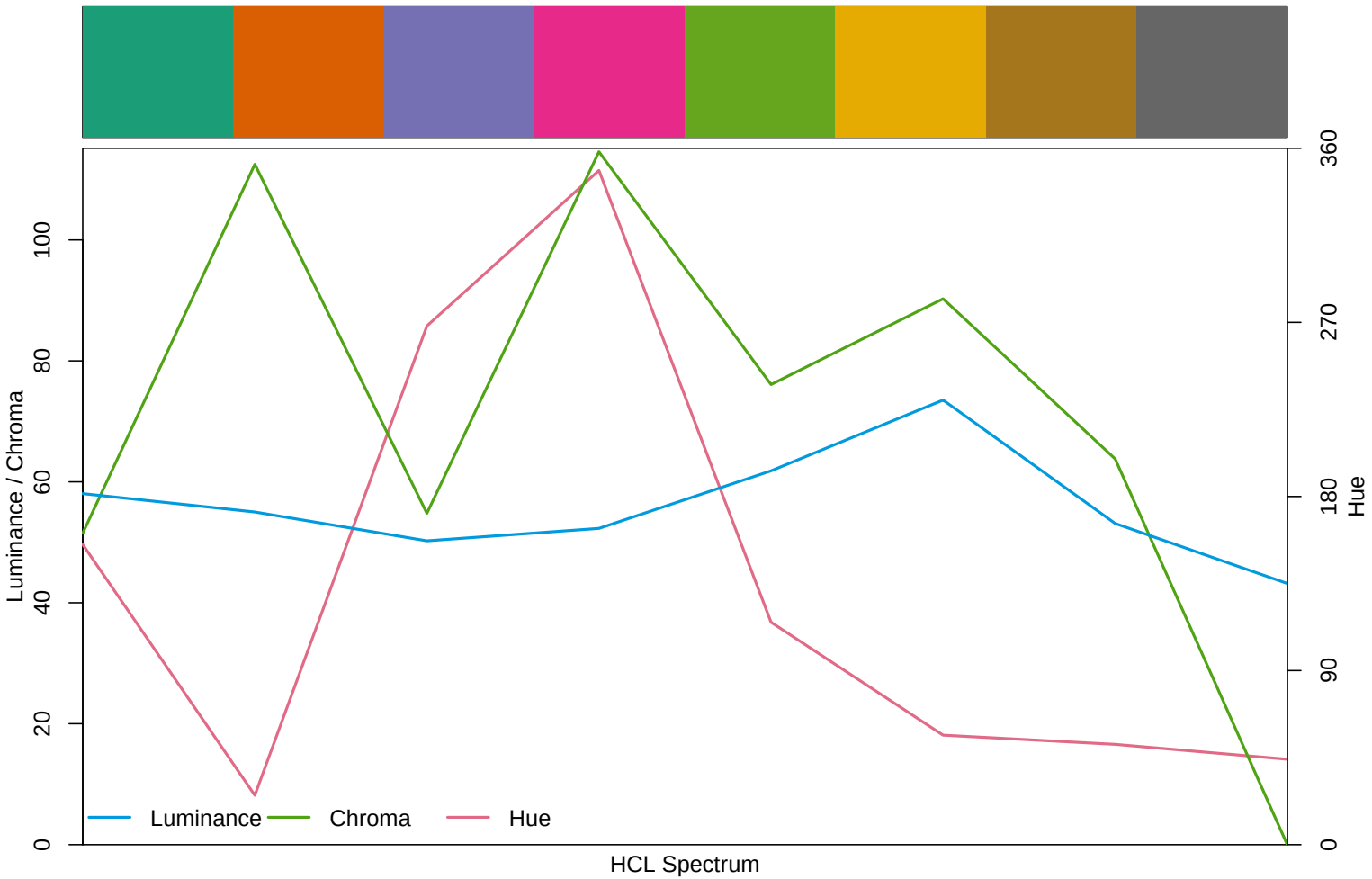
We can use specplot() from colorspace to visualize the palette showing how hue, chroma, and luminance are changing.
Qualitative palettes

```
specplot(brewer.pal(8, 'Set2'))
```



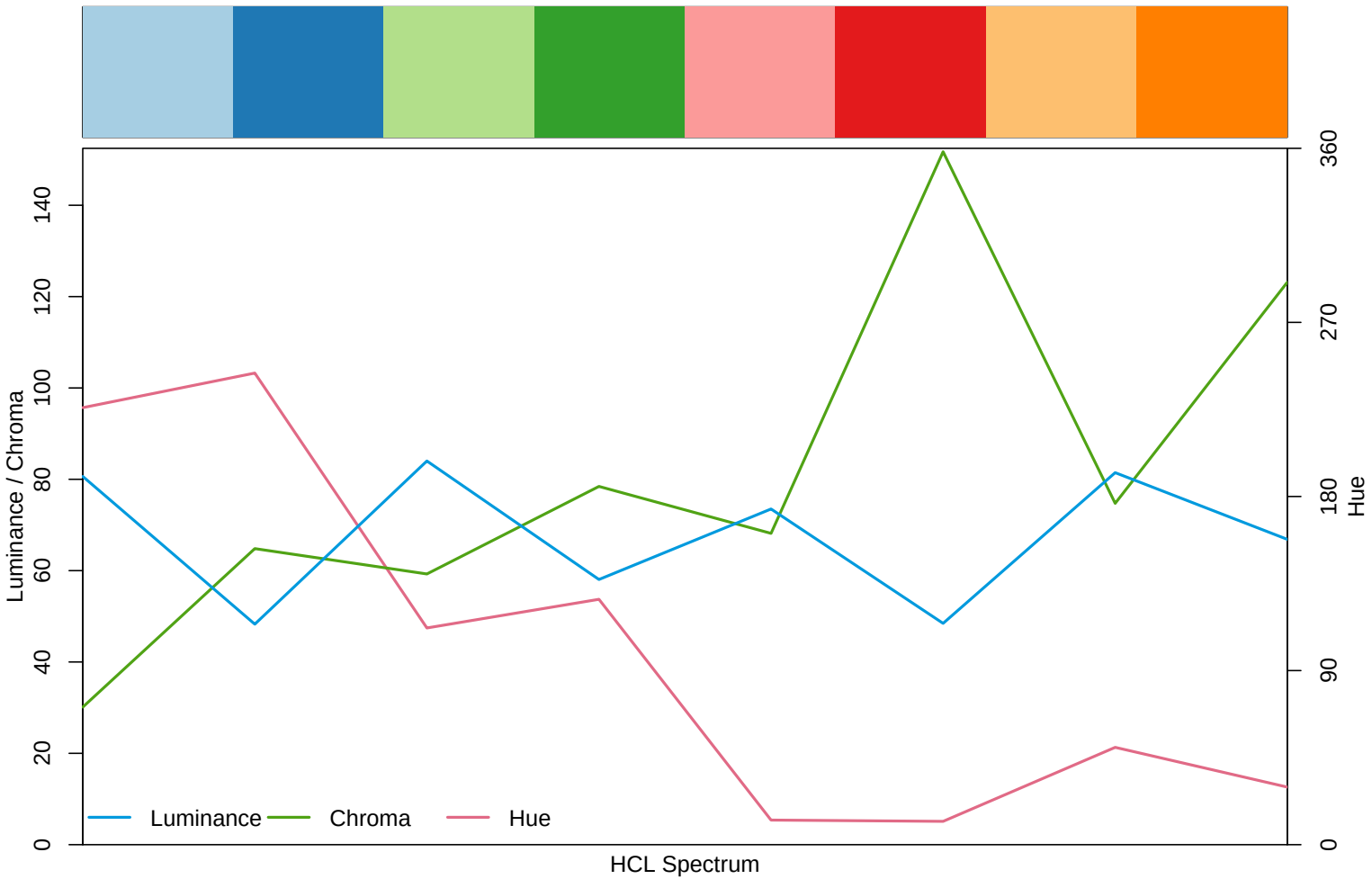
HCL model: color palettes

```
specplot(brewer.pal(8, 'Dark2'))
```



HCL model: color palettes

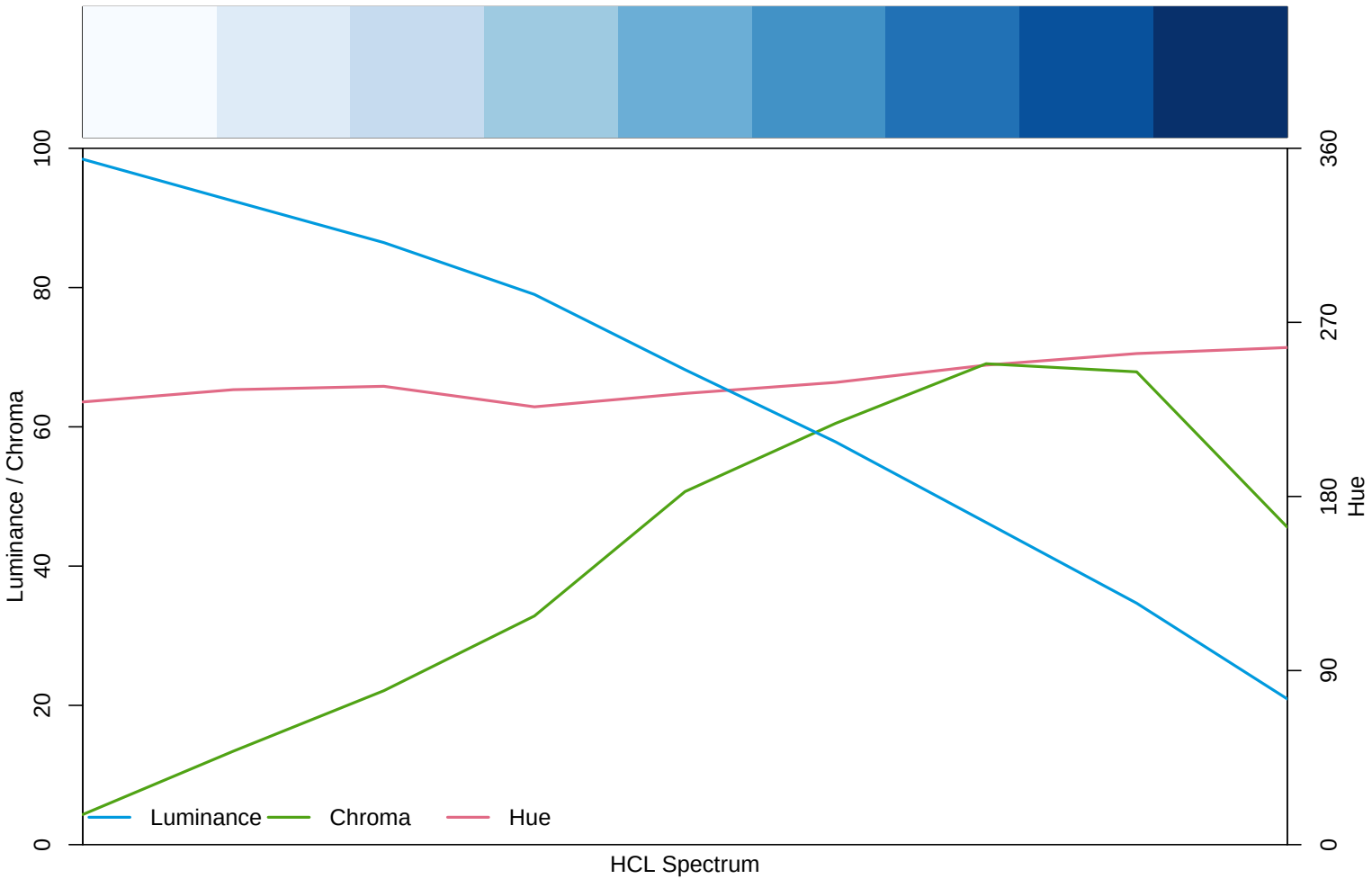
```
specplot(brewer.pal(8, 'Paired'))
```



HCL model: color palettes

Sequential palettes

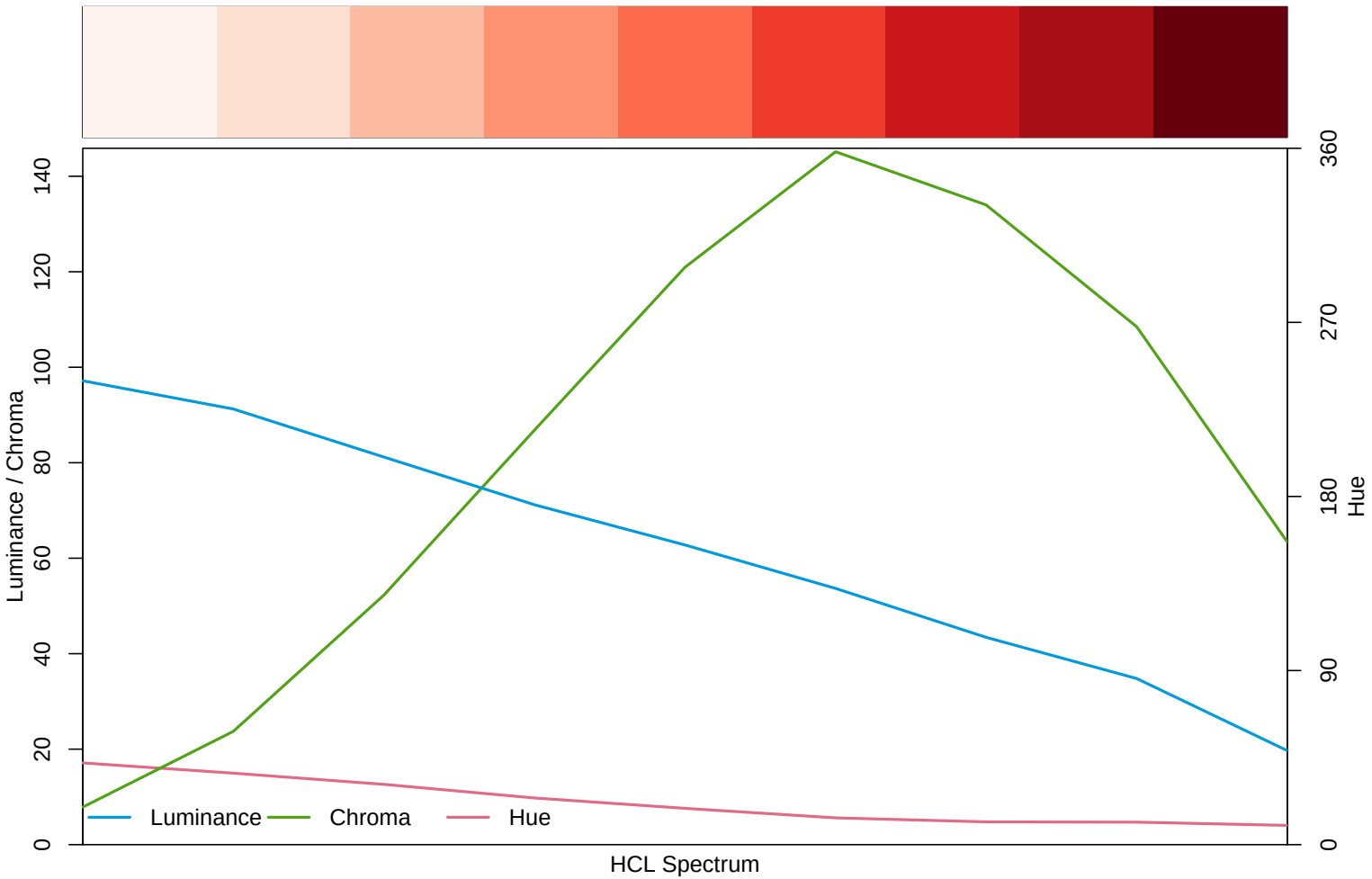
```
specplot(brewer.pal(9, 'Blues'))
```



HCL model: color palettes

Sequential palettes

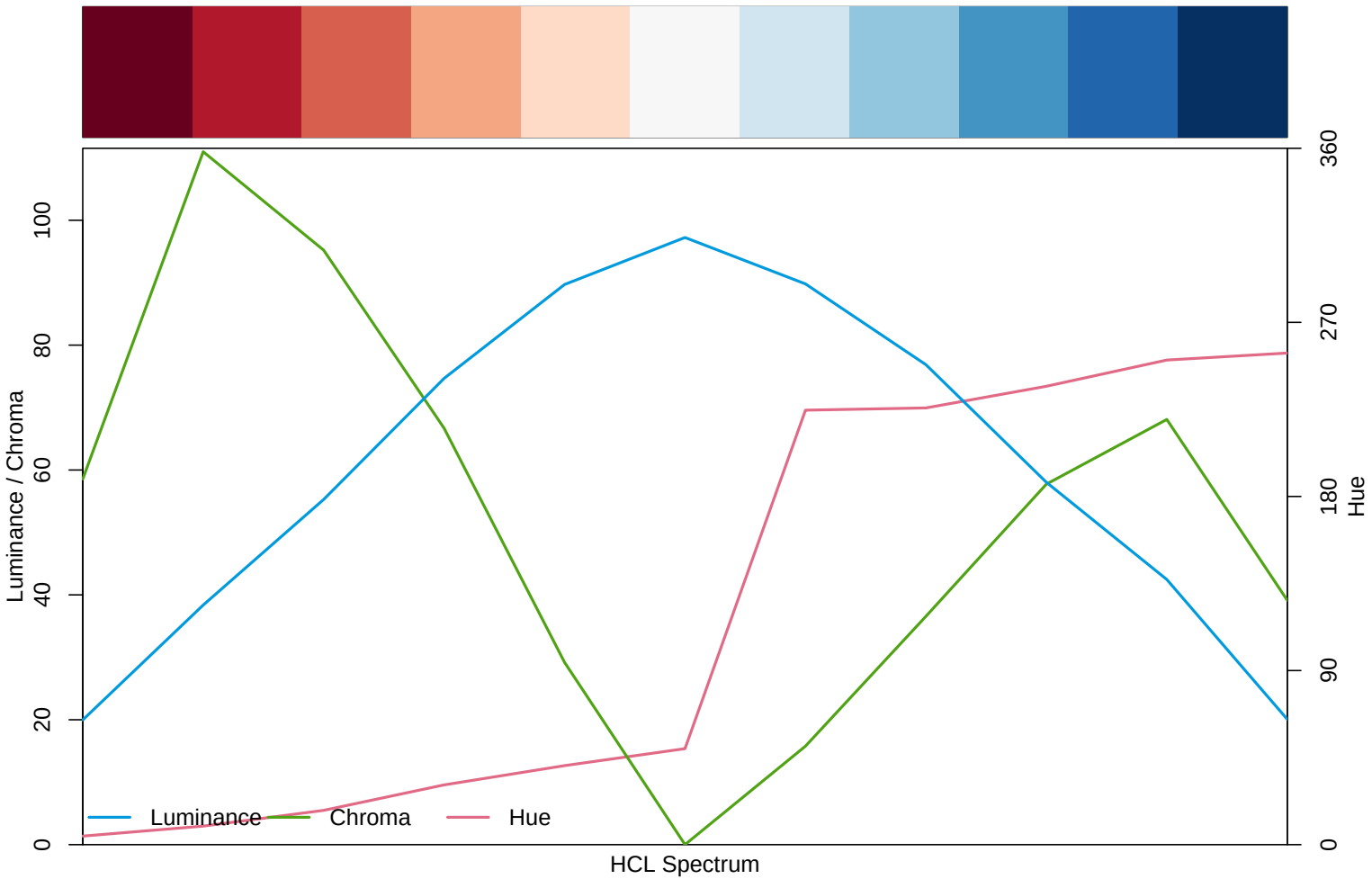
```
specplot(brewer.pal(9, 'Reds'))
```



HCL model: color palettes

Diverging palette

```
specplot(brewer.pal(11, 'RdBu'))
```



HCL model: Create your own palette

We can use colorspace to add additional palettes or create the ones we need

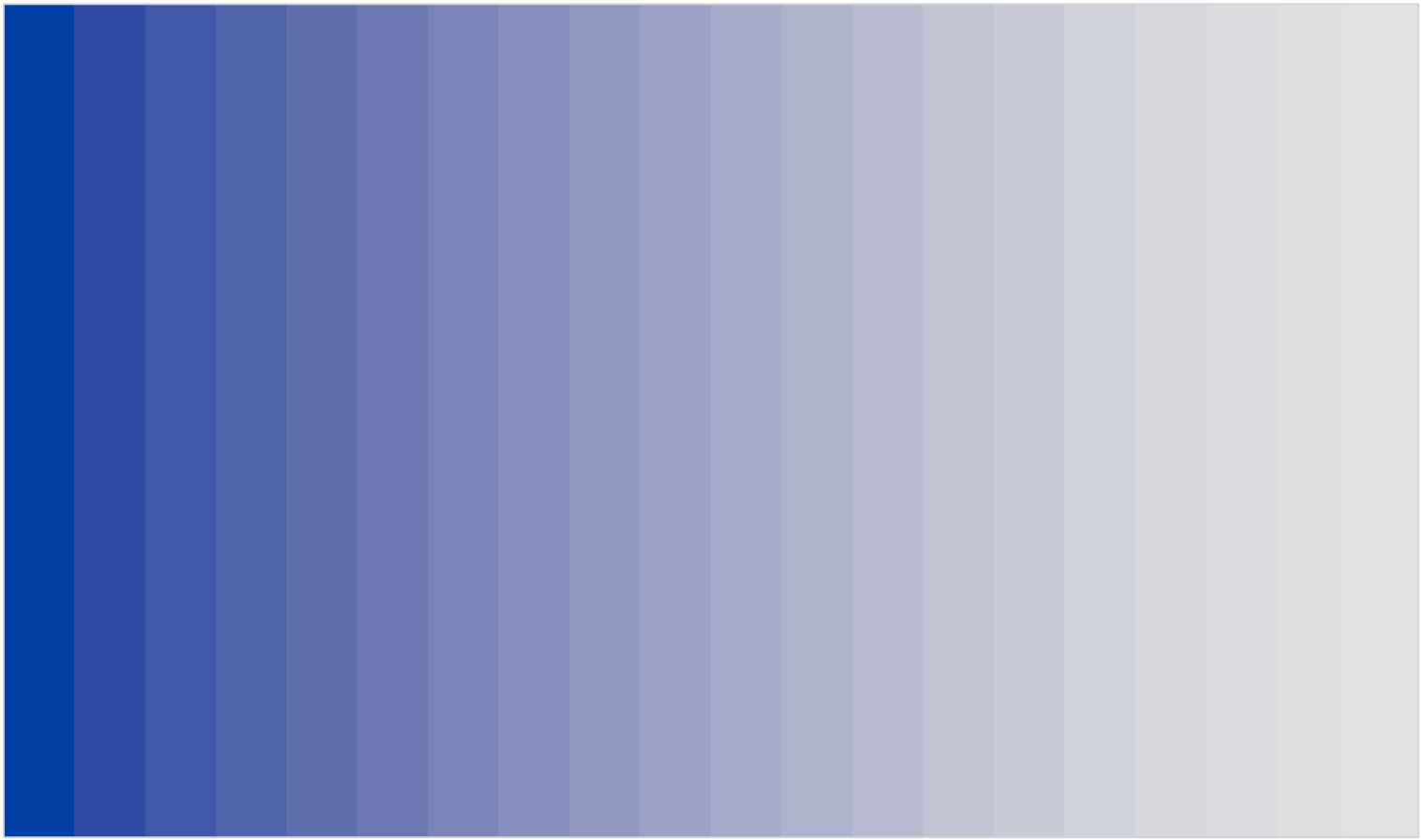
```
hcl_palettes()
## HCL palettes
##
## Type:  Qualitative
## Names: Pastel 1, Dark 2, Dark 3, Set 2, Set 3, Warm, Cold, Harmonic, Dyn
##
## Type:  Sequential (single-hue)
## Names: Grays, Light Grays, Blues 2, Blues 3, Purples 2, Purples 3, Reds
##        Reds 3, Greens 2, Greens 3, Oslo
##
## Type:  Sequential (multi-hue)
## Names: Purple-Blue, Red-Purple, Red-Blue, Purple-Orange, Purple-Yellow,
##        Blue-Yellow, Green-Yellow, Red-Yellow, Heat, Heat 2, Terrain,
##        Terrain 2, Viridis, Plasma, Inferno, Rocket, Mako, Dark Mint,
##        Mint, BluGrn, Teal, TealGrn, Emrld, BluYl, ag_GrnYl, Peach,
##        PinkYl, Burg, BurgYl, RedOr, OrYel, Purp, PurpOr, Sunset,
##        Magenta, SunsetDark, ag_Sunset, BrwnYl, YlOrRd, YlOrBr, OrRd,
##        Oranges, YlGn, YlGnBu, Reds, RdPu, PuRd, Purples, PuBuGn, PuBu,
##        Greens, BuGn, GnBu, BuPu, Blues, Lajolla, Turku, Hawaii, Batlow
```

HCL model: Sequential palettes

- Single-hue: In each palette the hue is fixed and chroma decreases monotonically (along with increasing luminance). This is typically sufficient to clearly bring out the extreme colors (dark/colorful vs. light gray).
- Single-hue (advanced): The hue is fixed (as above) but the chroma trajectory is triangular. Compared to the basic single-hue palette above this better distinguishes the colors in the middle and not only the extremes.
- Multi-hue (advanced): As in the advanced single-hue palette the chroma trajectory is triangular but additionally the hue varies slightly. This can further enhance the distinction of colors in the middle of the palette.

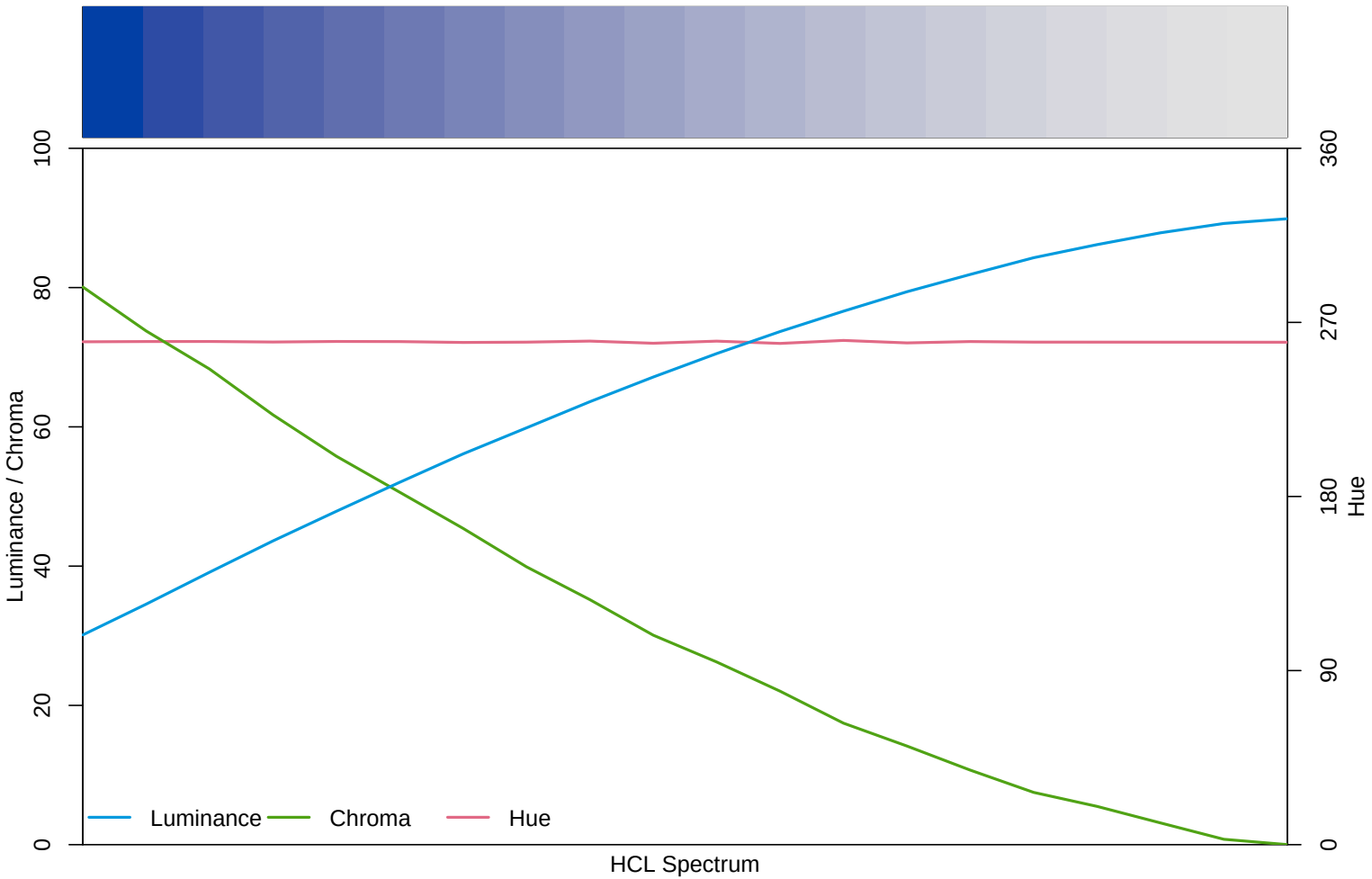
HCL model: Single hue

```
s_hue <- sequential_hcl(n=20, h = 260, l = c(30, 90))
swatchplot(s_hue)
```



HCL model: Single hue

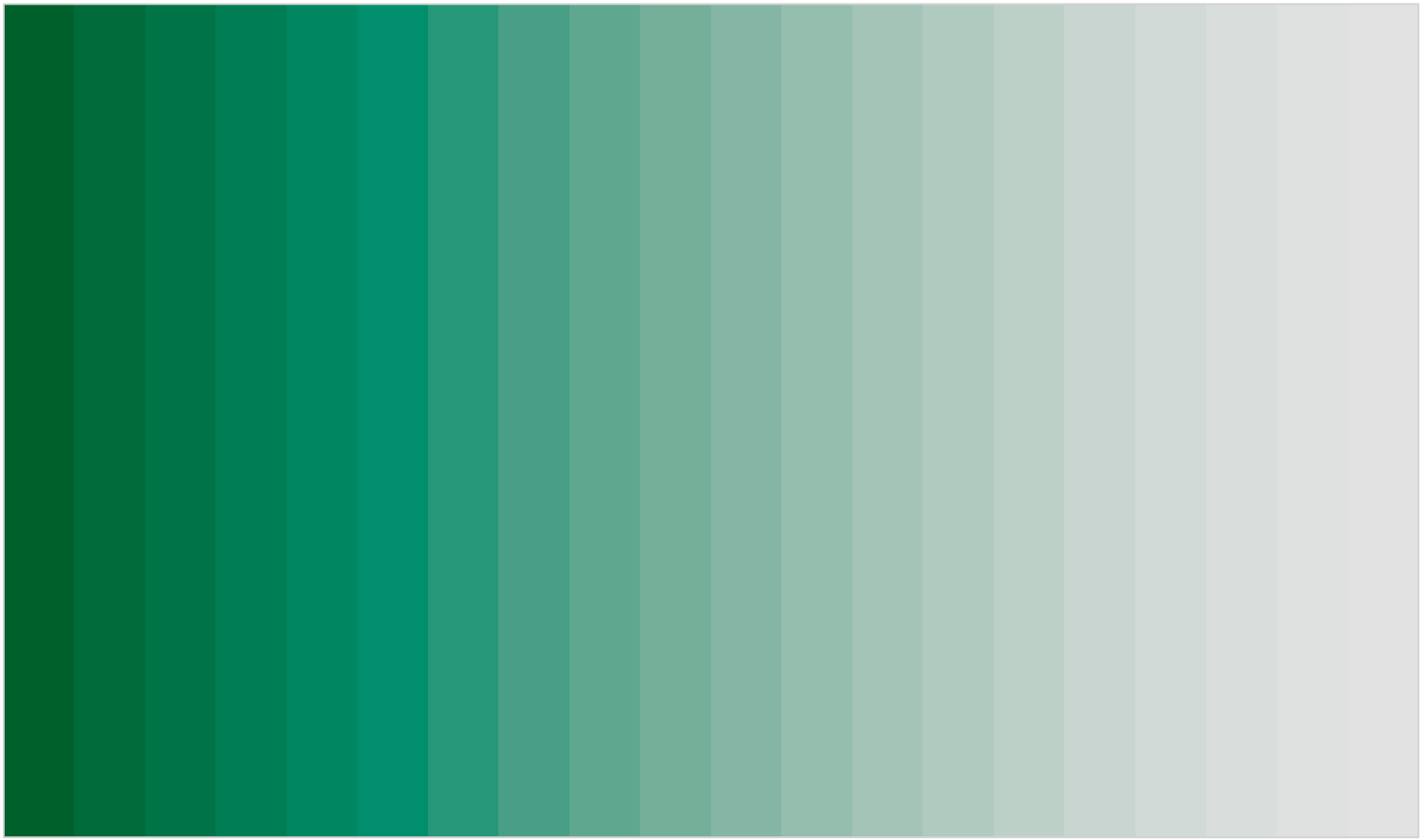
```
specplot(s_hue)
```



HCL model: Single hue

Change the hue

```
s_hue <- sequential_hcl(n=20, h = 160, l = c(30, 90))
swatchplot(s_hue)
```

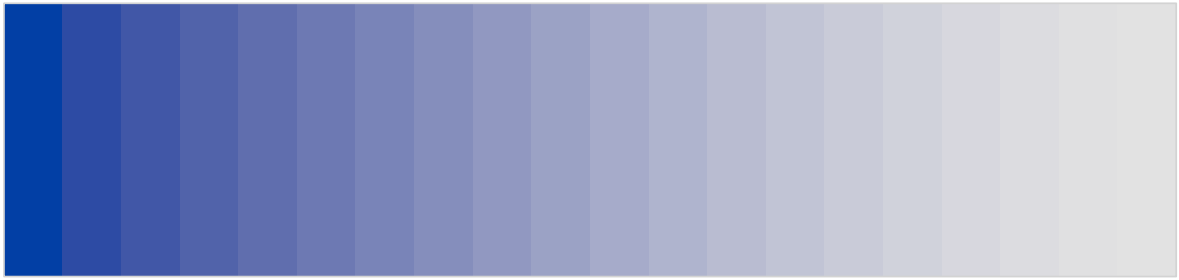


HCL model:

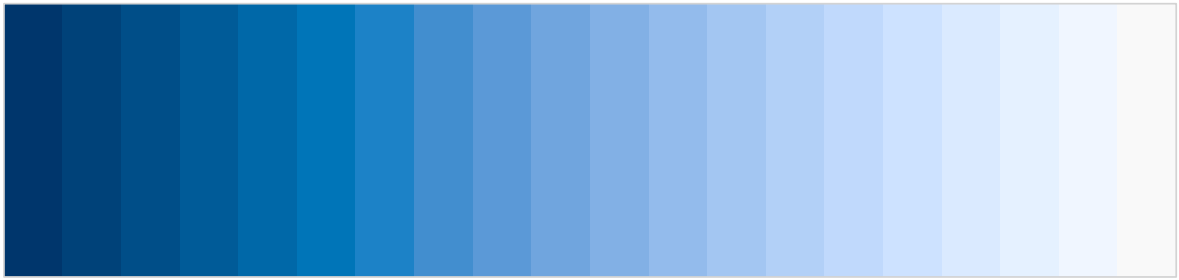
```
swatchplot('Single Hue' = sequential_hcl('Blues 2', n=20),
           'Single Hue Advanced' = sequential_hcl('Blues 3', n=20),
           'Multi Hue (Red-Blue)' = sequential_hcl('Red-Blue', n=20))
```

HCL model:

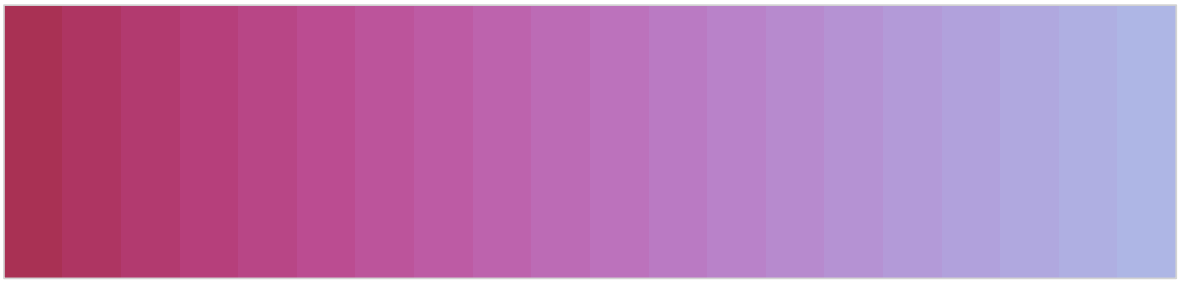
Single Hue



Single Hue Advanced

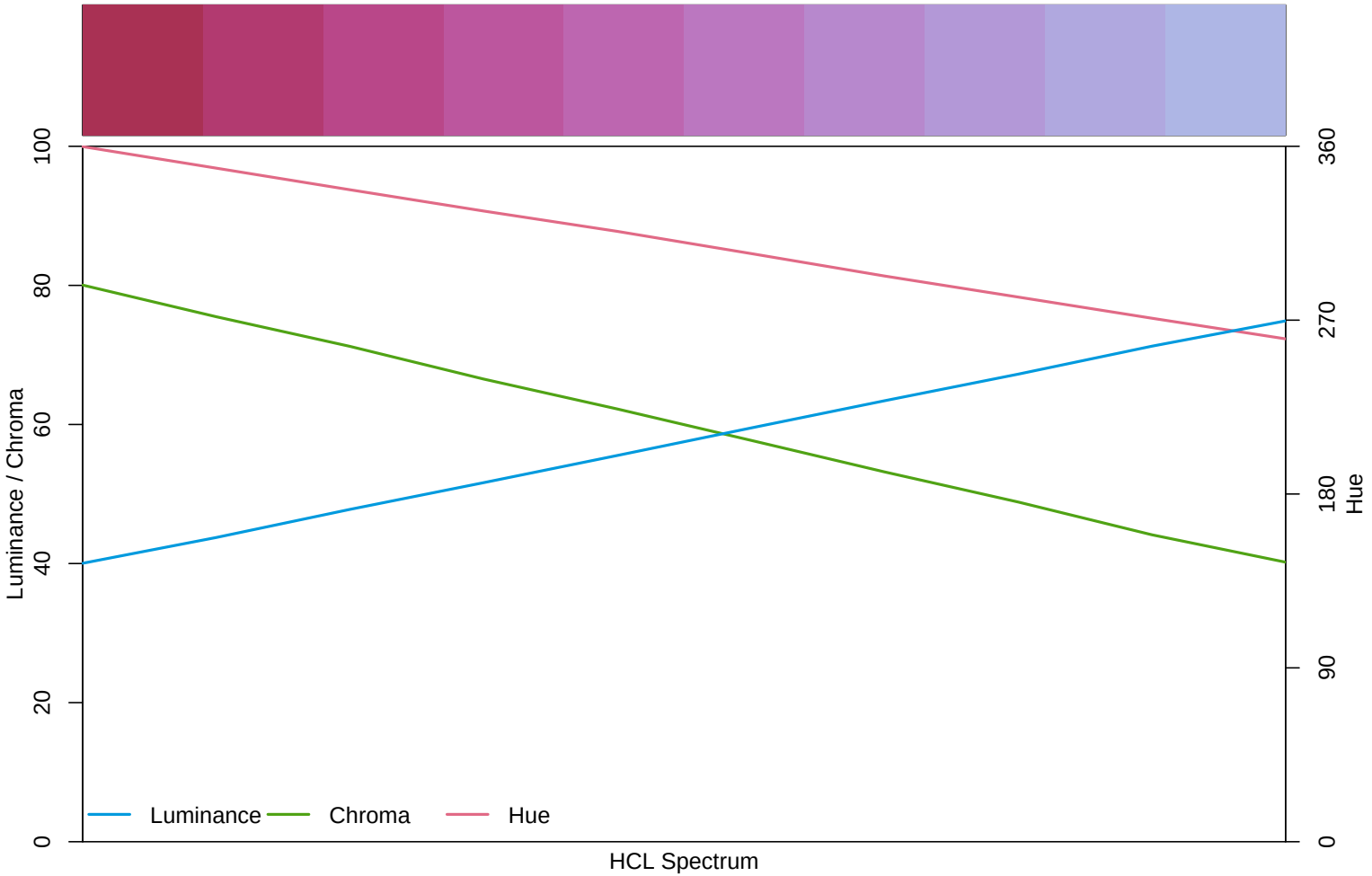


Multi Hue (Red−Blue)



HCL model: Single-hue (advanced)

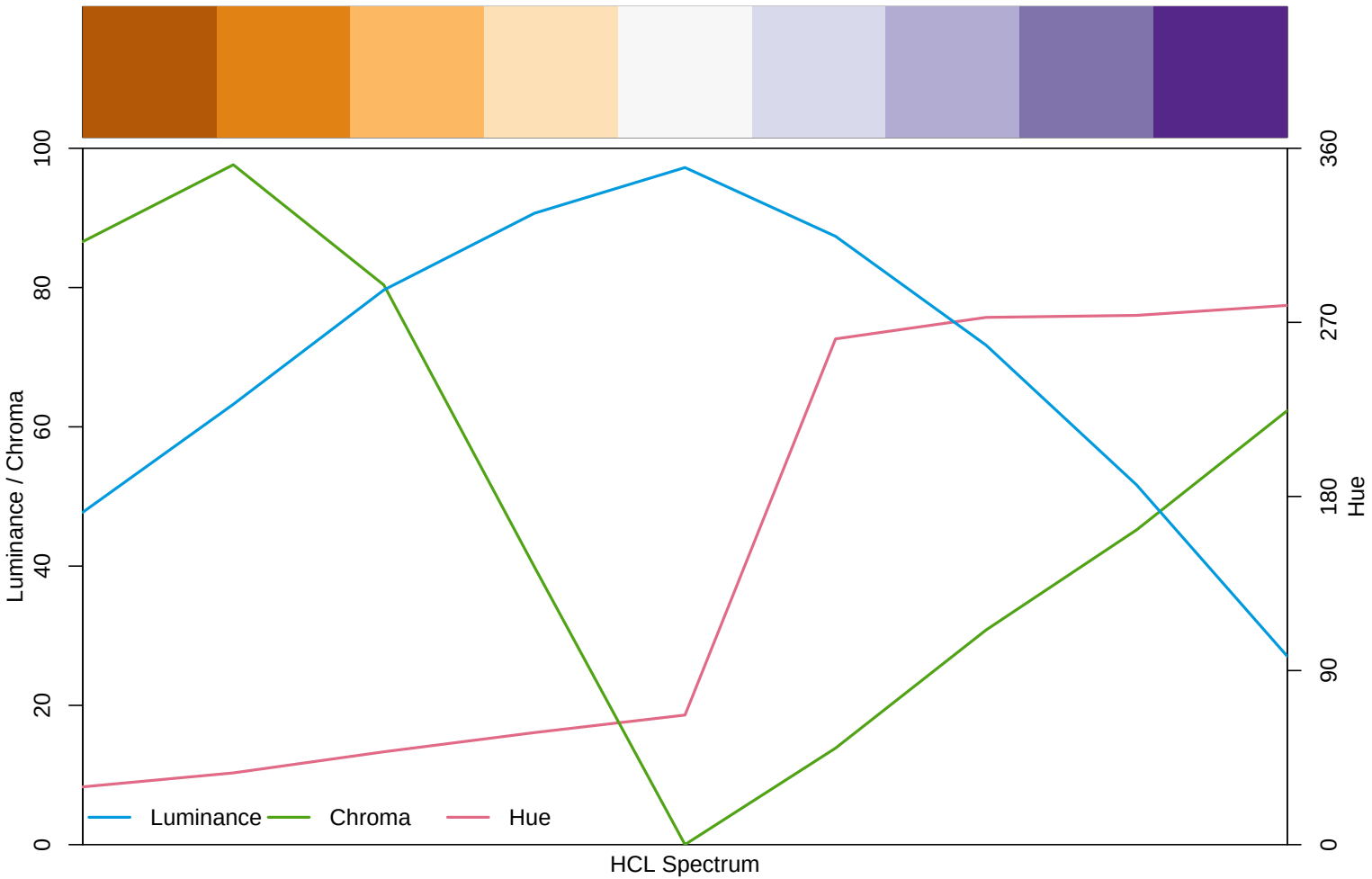
```
specplot(sequential_hcl('Red-Blue', n = 10))
```



HCL model: Diverging palettes

In RColorBrewer

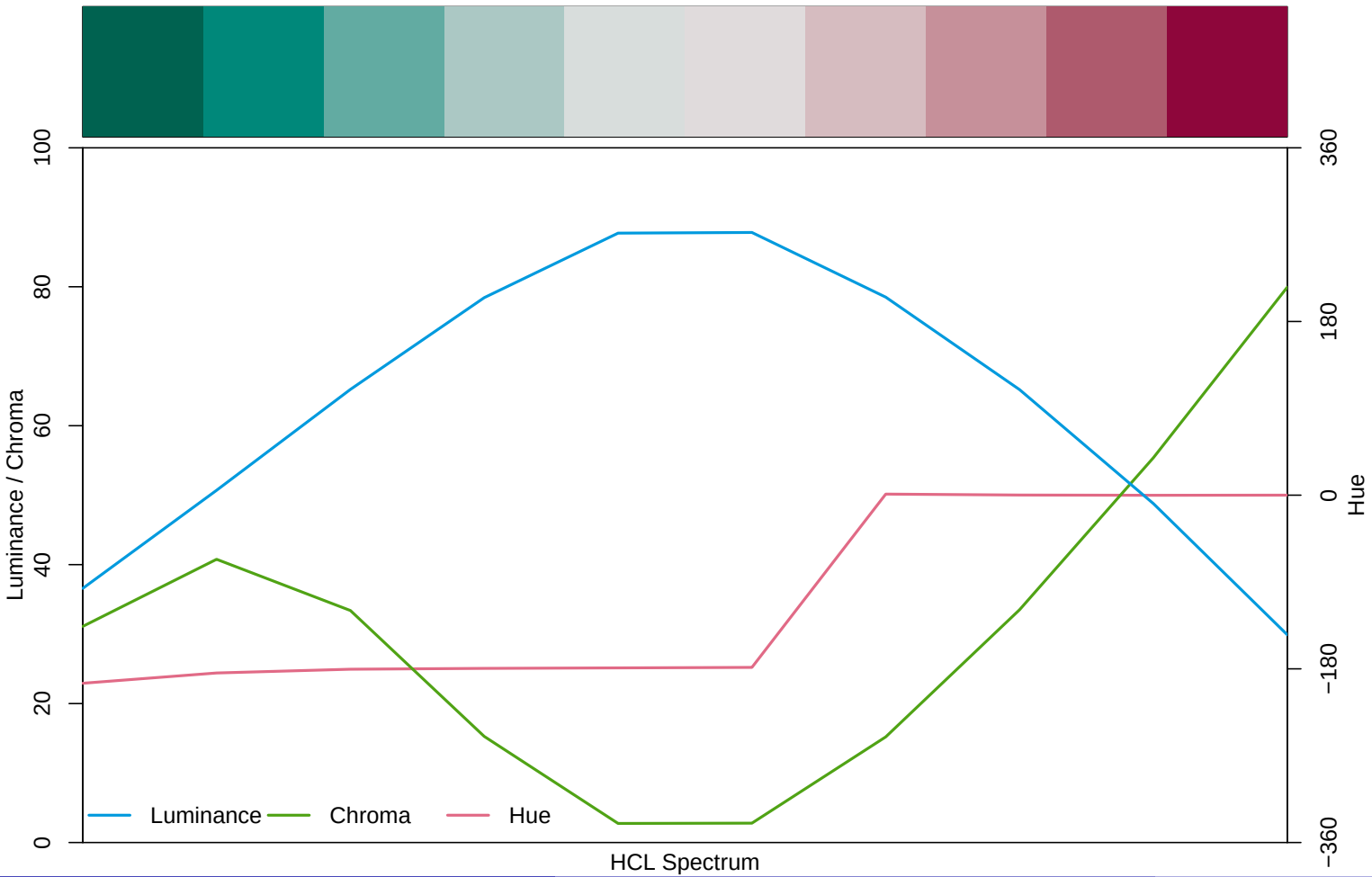
```
specplot(brewer.pal(9, 'PuOr'))
```



HCL model: Diverging palettes

From colorspace

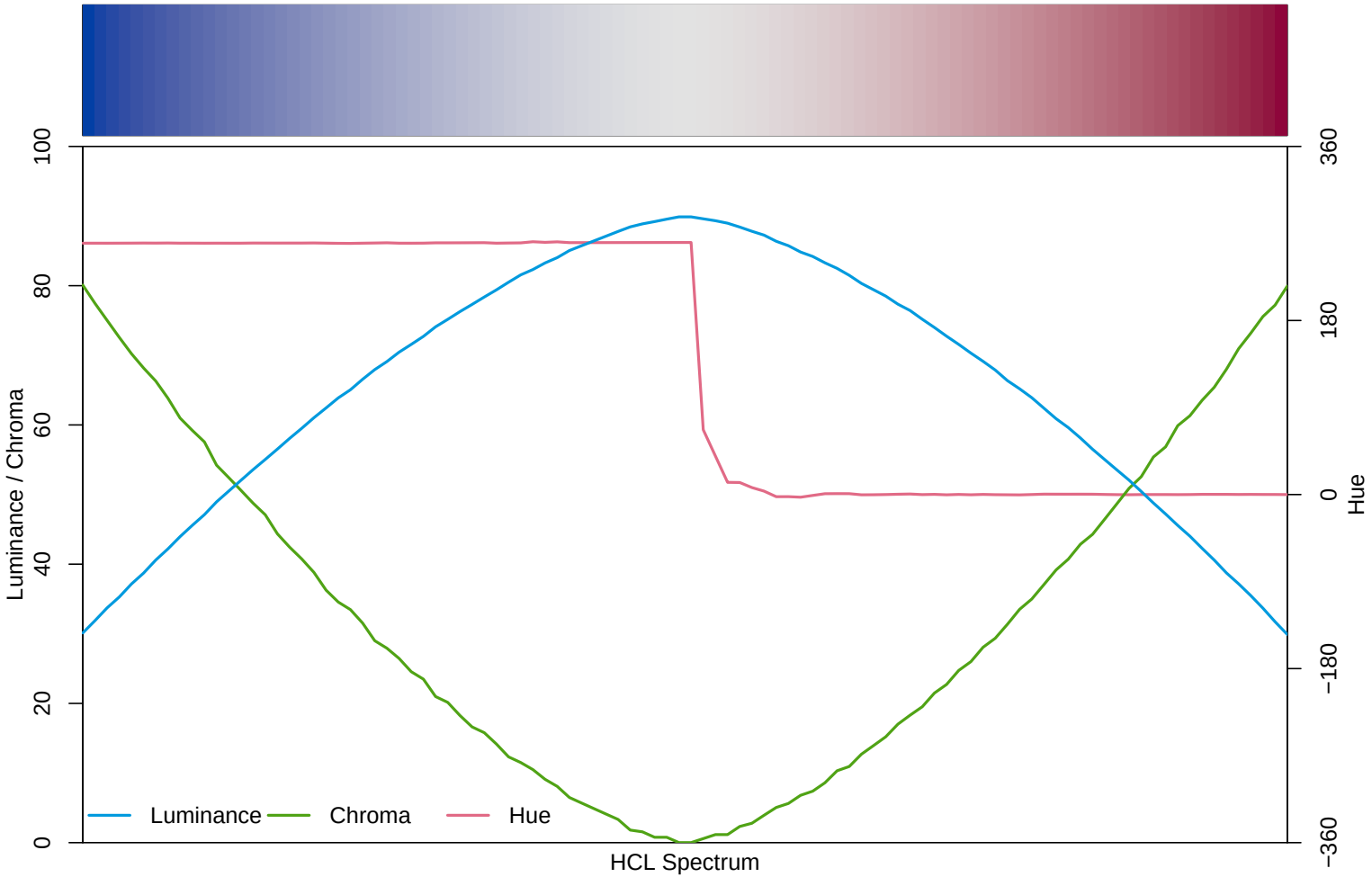
```
pal <- diverging_hcl(10, h = c(180, 0), c = 80, l = c(30, 90))
specplot(pal)
```



HCL model: Single-hue (advanced)

Diverging Palette in colorspace

```
specplot(diverge_hcl('Blue - Red',n= 100))
```

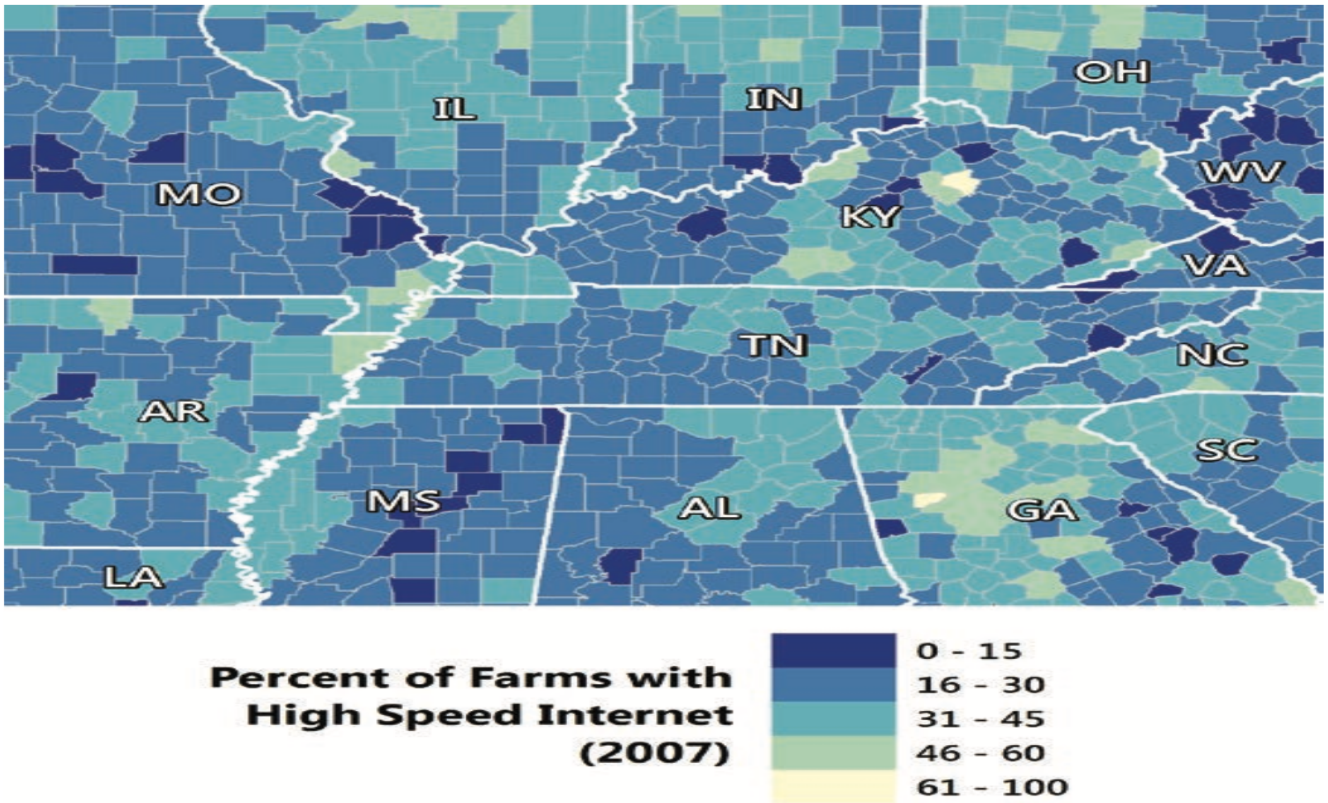


Palettes in use

- RColorbrewer and its packages are based on the colorbrewer.org project.
- The colorbrewer project is based on the work of Cynthia Brewer - Designing "Better Maps: A Guide for GIS Users"

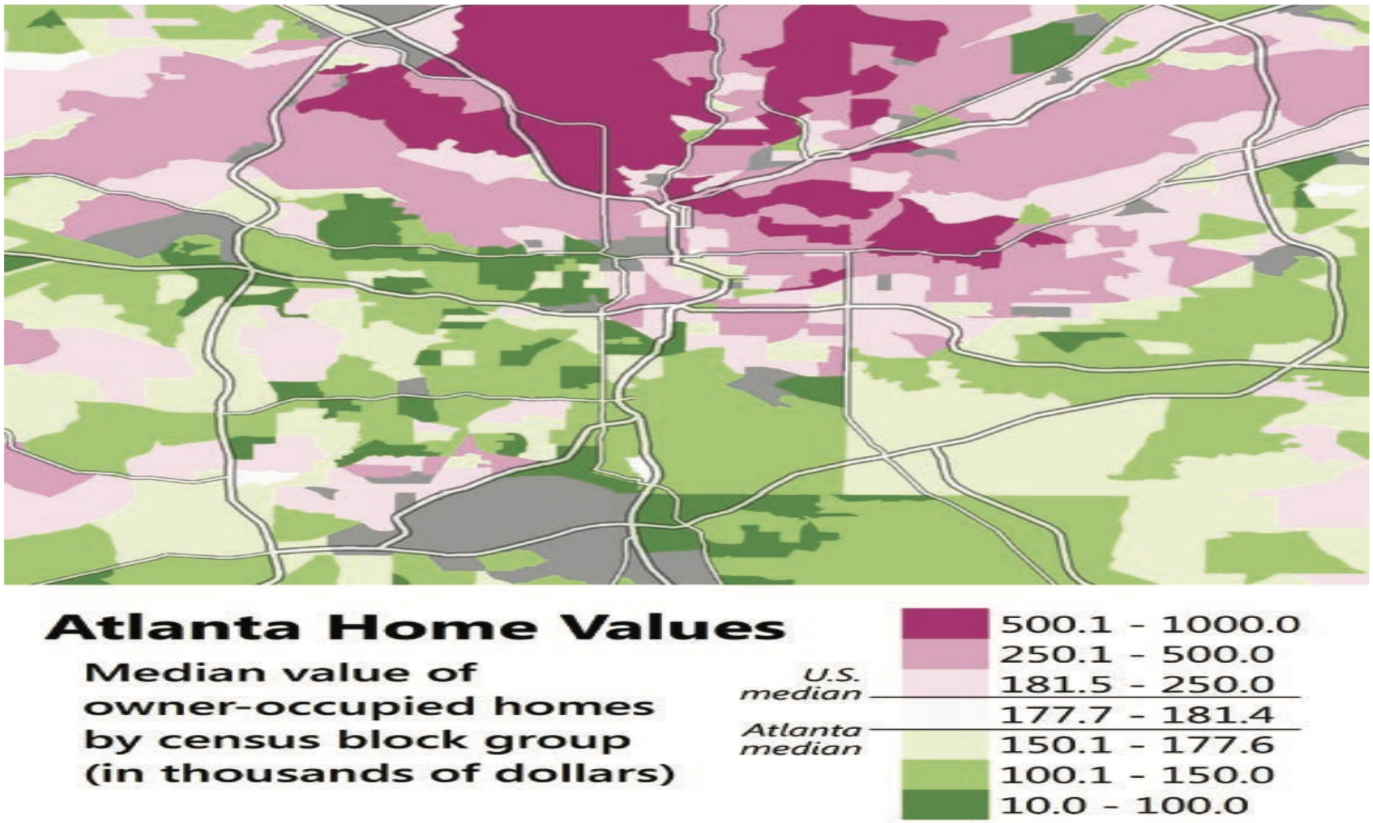
Palettes in use

An example of the usage of sequential palette
Some counties had almost no farms with access to high-speed Internet in 2007. The color sequence is reversed from the standard: "dark represents high" convention to emphasize lack of access (p 153).



Palettes in use: Diverging scale

This map shows median house value by census block group in Atlanta, Georgia. It uses a diverging color scheme with two hues (magenta and green) and a light-gray middle class for block groups between overall median values for Atlanta and the United States.(p. 154)



Section 2

Colors in R

Colors in R

R has a predefined, named list of colors

```
length(colors())
## [1] 657
colors()[1:10]
##  [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"
##  [5] "antiquewhite2"  "antiquewhite3"  "antiquewhite4"  "aquamarine"
##  [9] "aquamarine1"    "aquamarine2"
```

Colors in R

function col2rgb() translates any color to RGB notation

```
col2rgb(col=1:6)
##       [,1] [,2] [,3] [,4] [,5] [,6]
## red      0  223   97   34   40  205
## green    0   83  208  151  226   11
## blue     0  107   79  230  229  188
```

or refer by name

```
col2rgb("seagreen")
##       [,1]
## red     46
## green  139
## blue    87
```

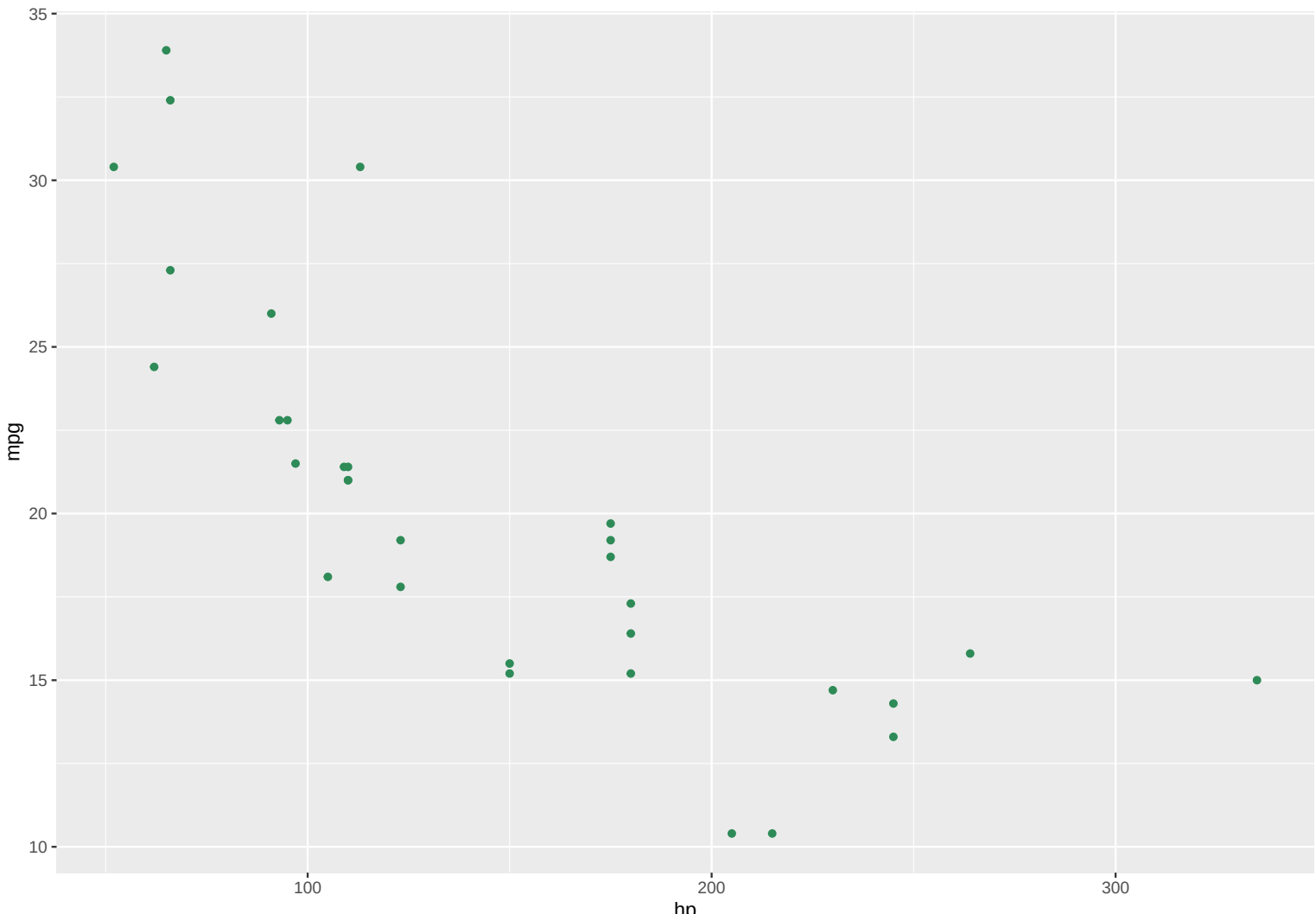
get the hexadecimal code

```
rgb(46,139,87, maxColorValue = 255)
## [1] "#2E8B57"
```

Colors in ggplot2

You can use either the color name or its hexadecimal code in ggplot

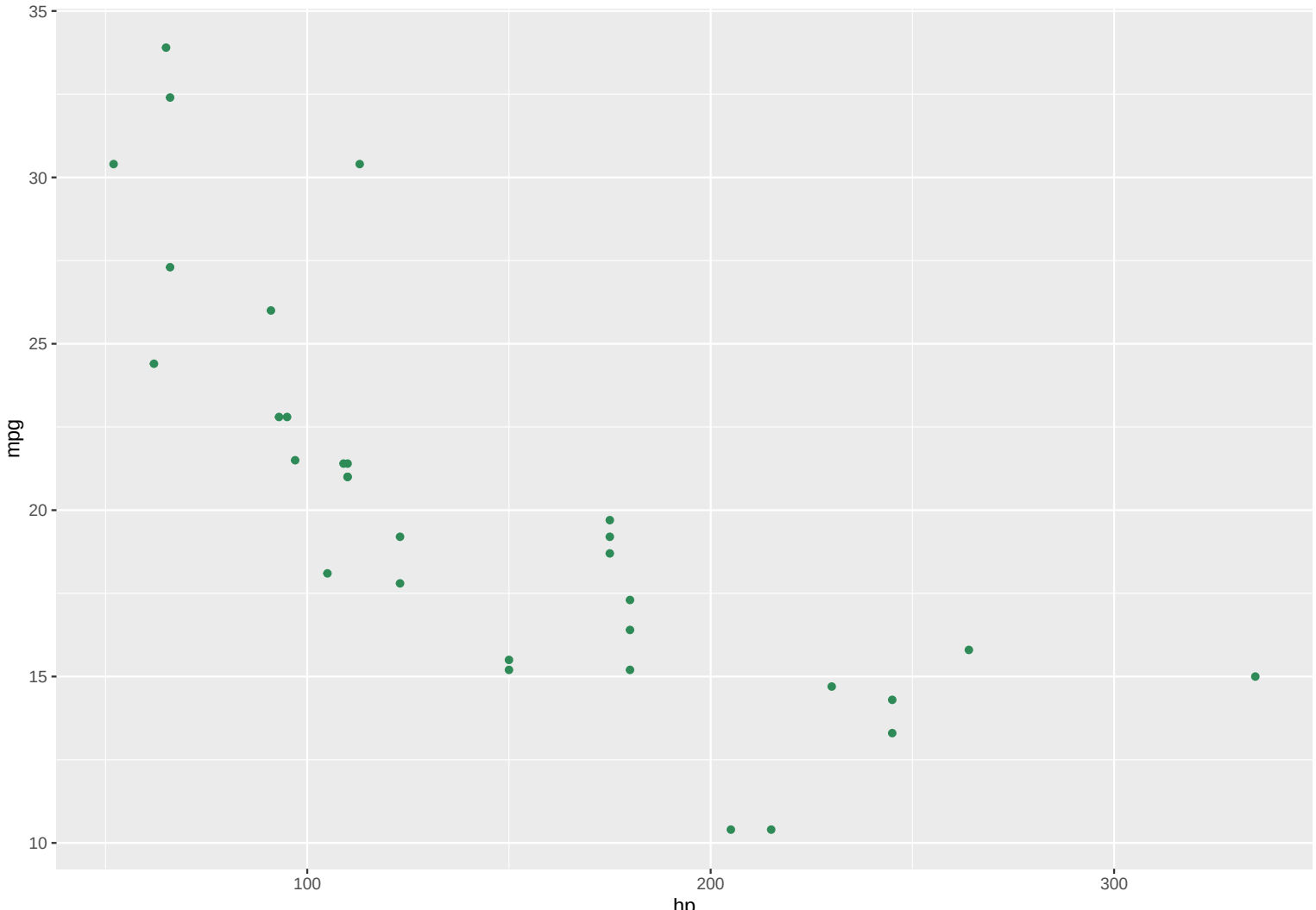
```
ggplot(mtcars, aes(x = hp, y = mpg)) + geom_point(color = "seagreen")
```



Colors in ggplot2

The same with the hexadecimal notation

```
ggplot(mtcars, aes(x = hp, y = mpg)) + geom_point(color = "#2E8B57")
```

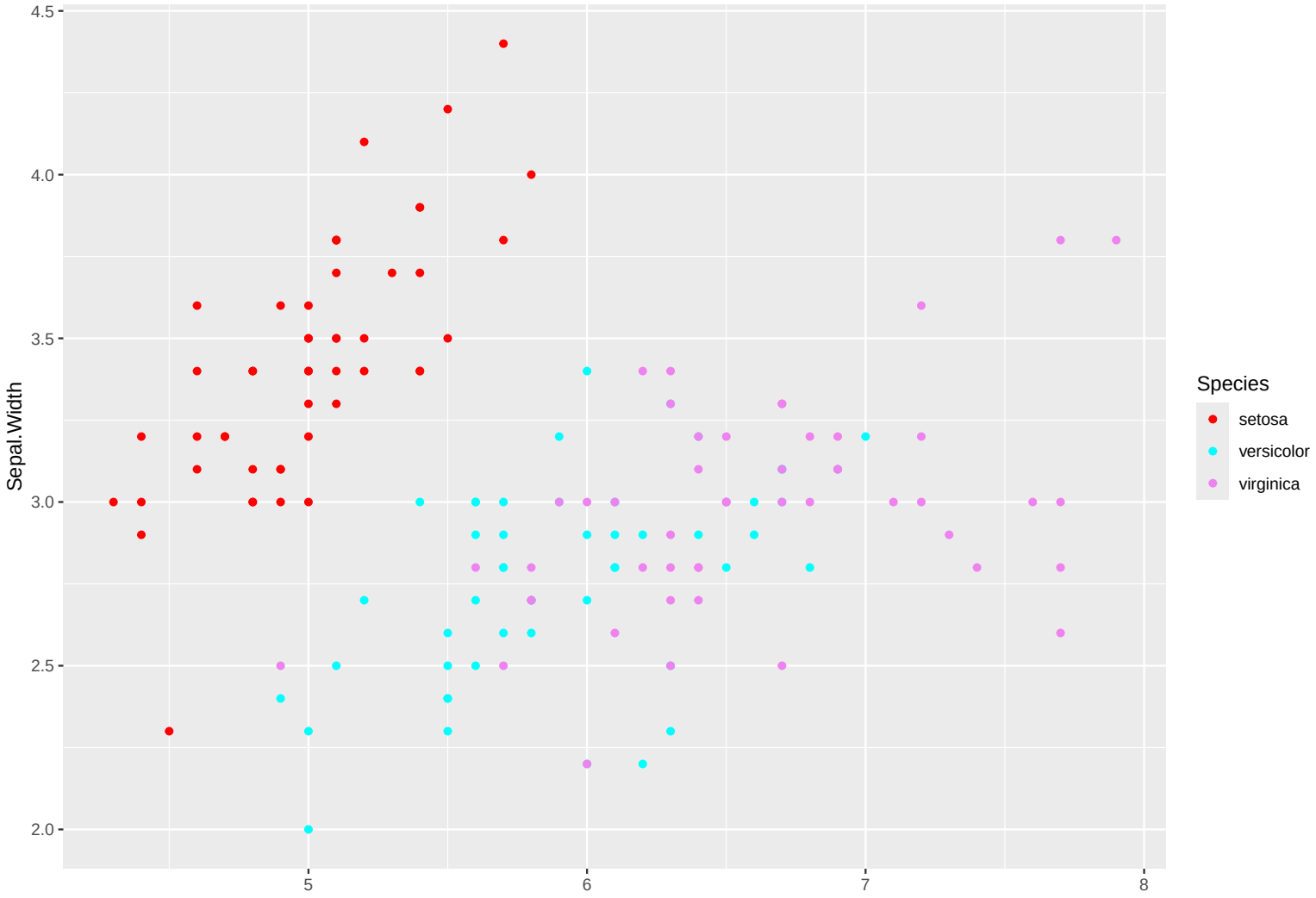


Colors in ggplot2

- Colors for different aesthetics are controlled by the scale.
- Depending on how you are planning to access and use colors, different layers are available
  - scale_XXX_manual - for manual entry of the colors
  - scale_XXX_brewer - to use color_brewer
  - and many more

Colors in ggplot2

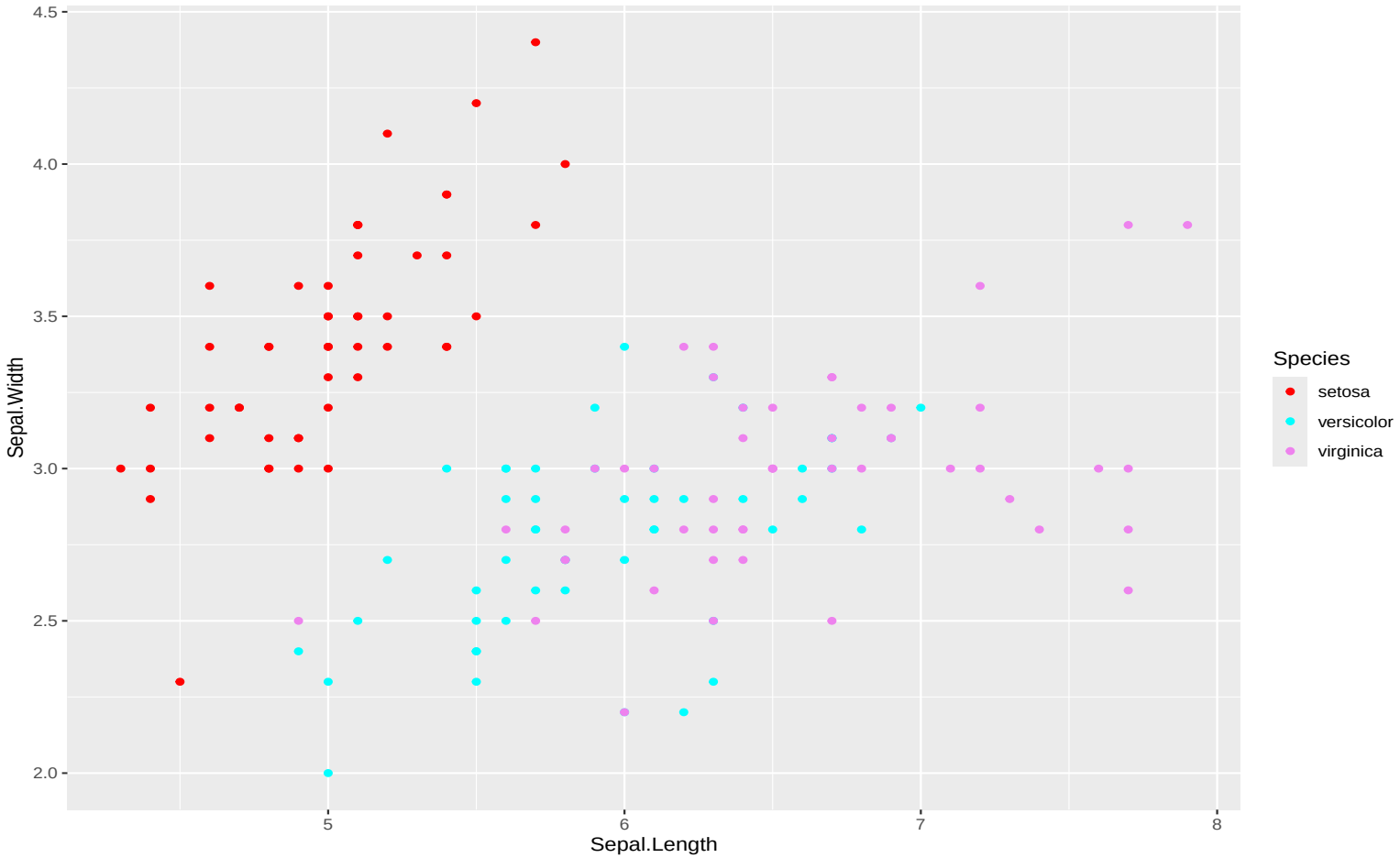
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point()+ scale_color_manual(values = c("red", "cyan", "violet"))
```



Colors in ggplot2

The same with hex codes

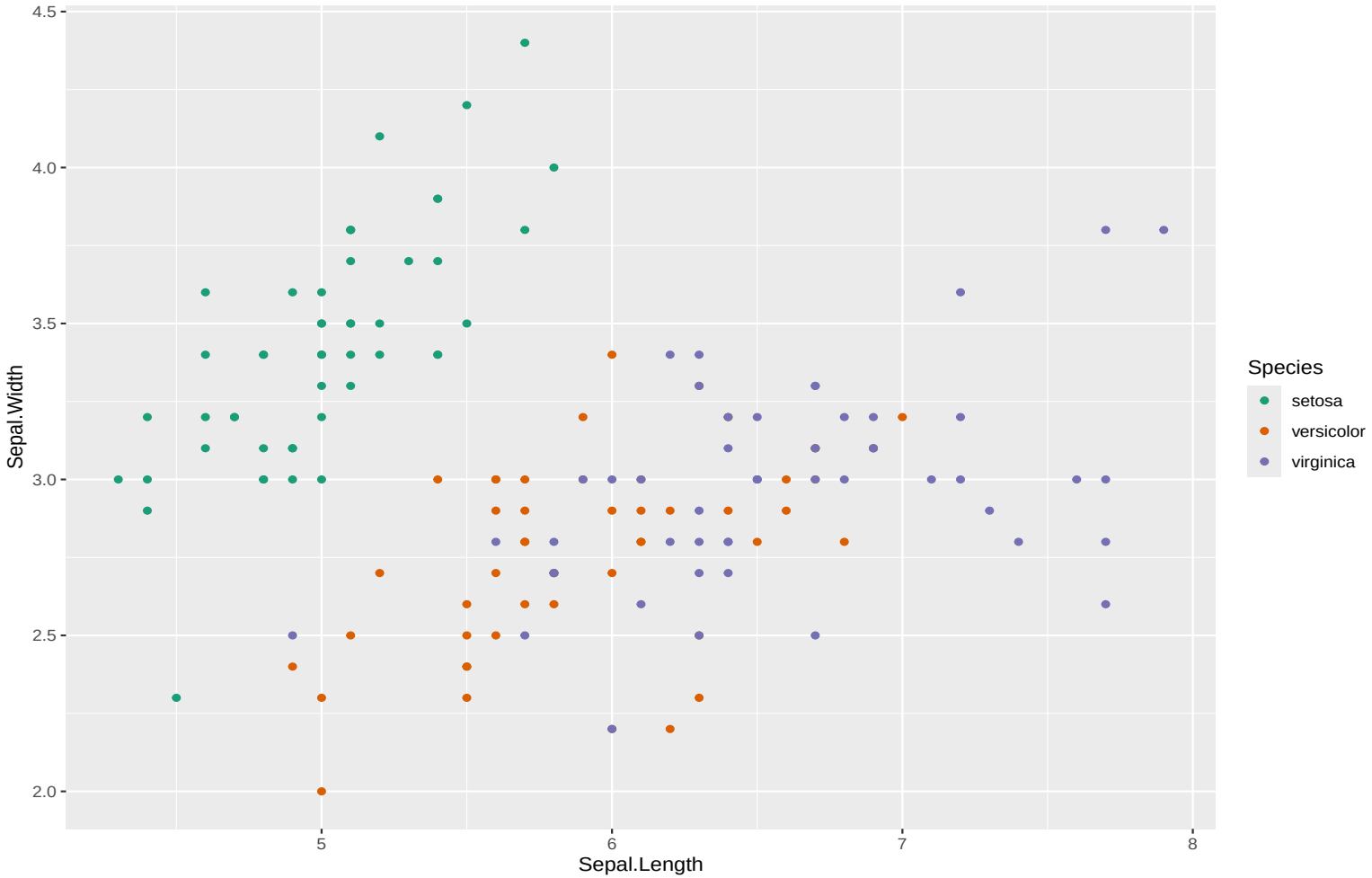
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point()+
  scale_color_manual(values = c("#FF0000", "#00FFFF", "#EE82EE"))
```



Colors in ggplot2

Color palettes with Brewer

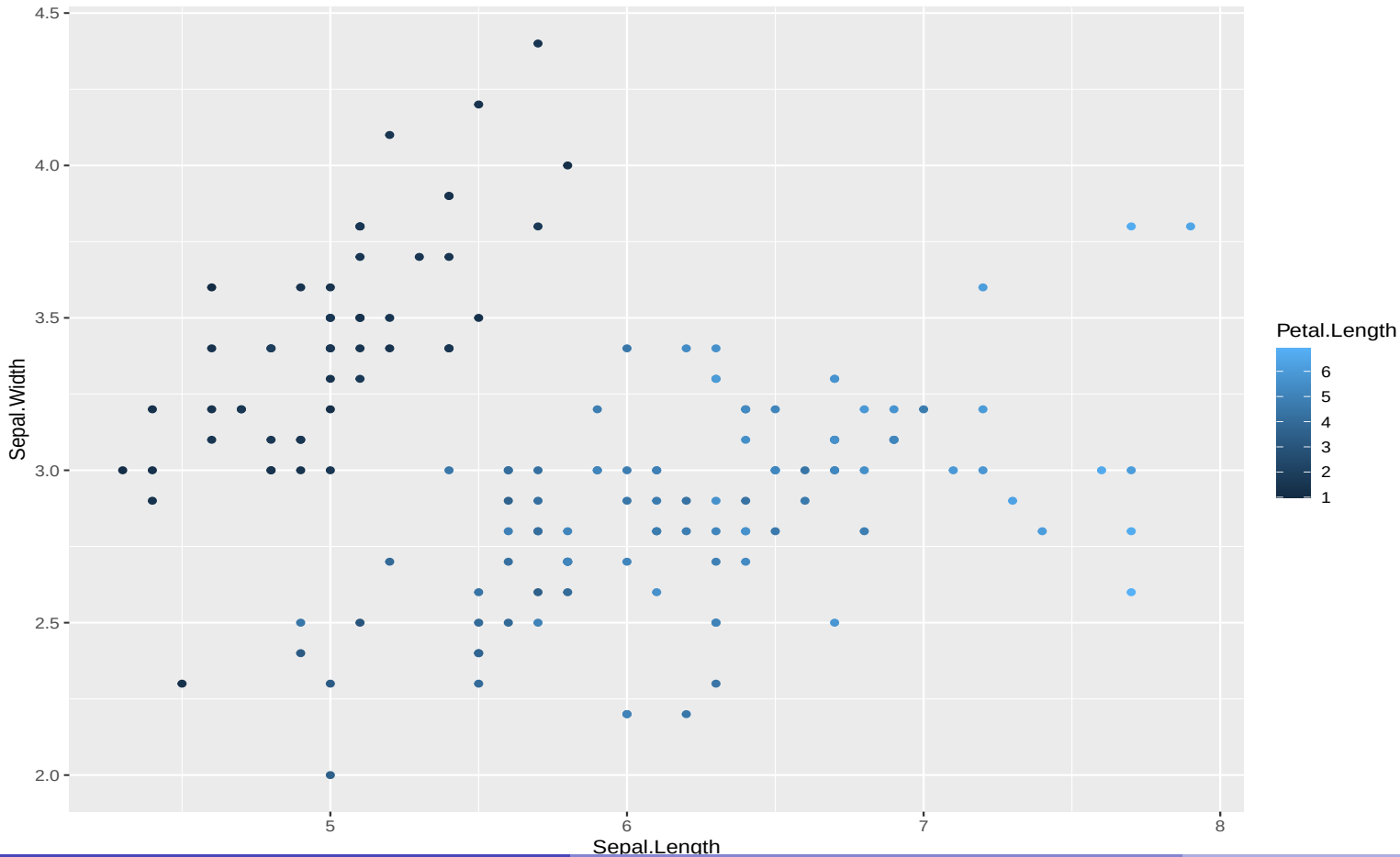
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point() + scale_color_brewer(palette = 'Dark2')
```



Colors in ggplot2

With the continuous variable we are going to use color gradient that can be either diverging or sequential. By default the Blues palette is used.

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length))+
  geom_point()
```



Colors in ggplot2: Color gradient

When you have continuous variable, you cannot use diverging or sequential palettes as it is. Instead you use color gradient by specifying

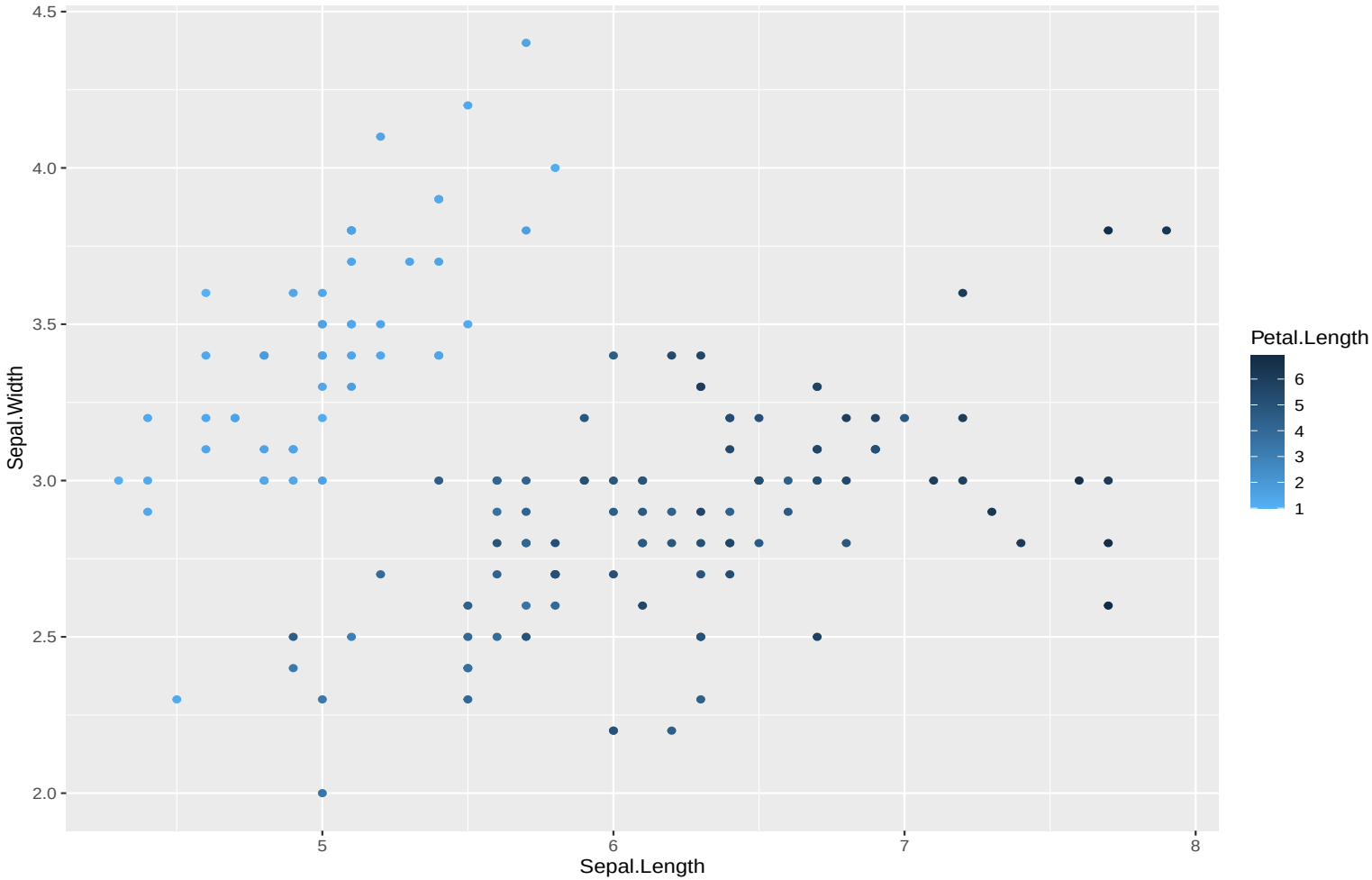
- low and high hues for sequential palettes
- low, high and optionally midpoint

The colour gradient scale maps a segment of the real line to a path through a colour space.

Colors in ggplot2: Color gradient

Define high and low values

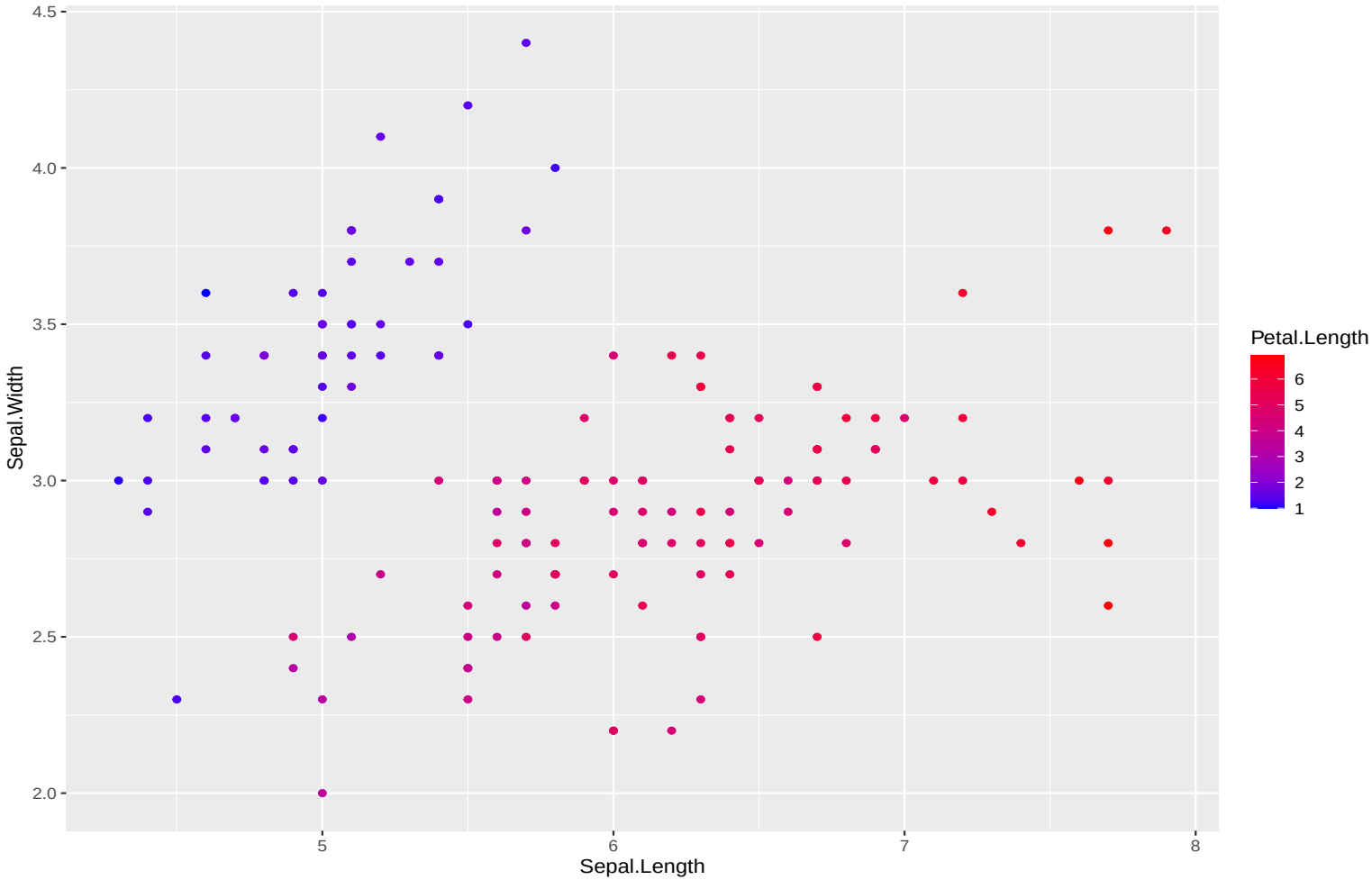
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length)) +
  geom_point() + scale_color_continuous(low = "#56B1F7", high =  "#132B43")
```



Colors in ggplot2

Any two hues will work

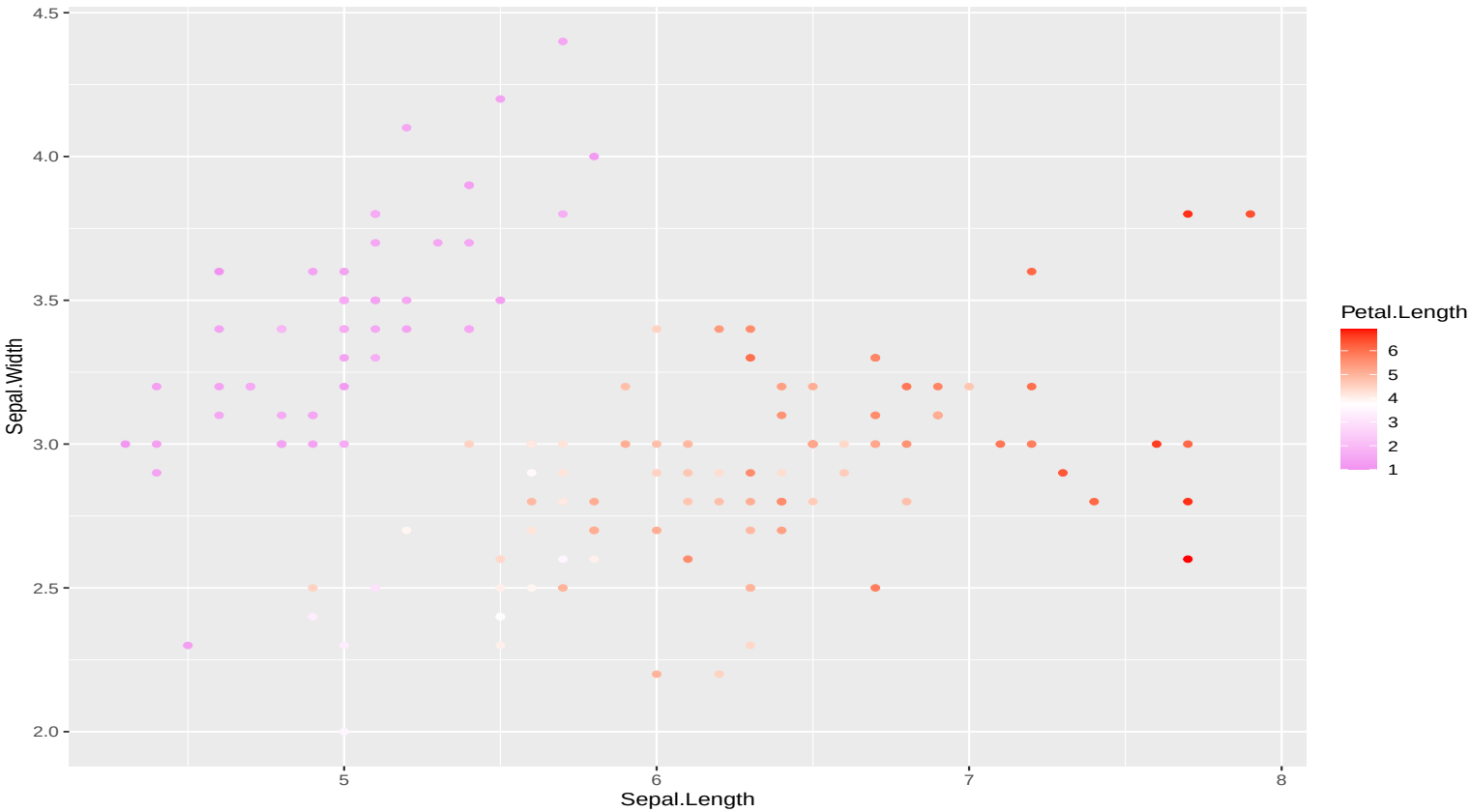
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length)) +
  geom_point() + scale_color_continuous(low = "blue", high = "red")
```



Colors in ggplot2

- scale_XXX_gradient() gives some more capabilities
- scale_XXX_gradient2() for diverging colors

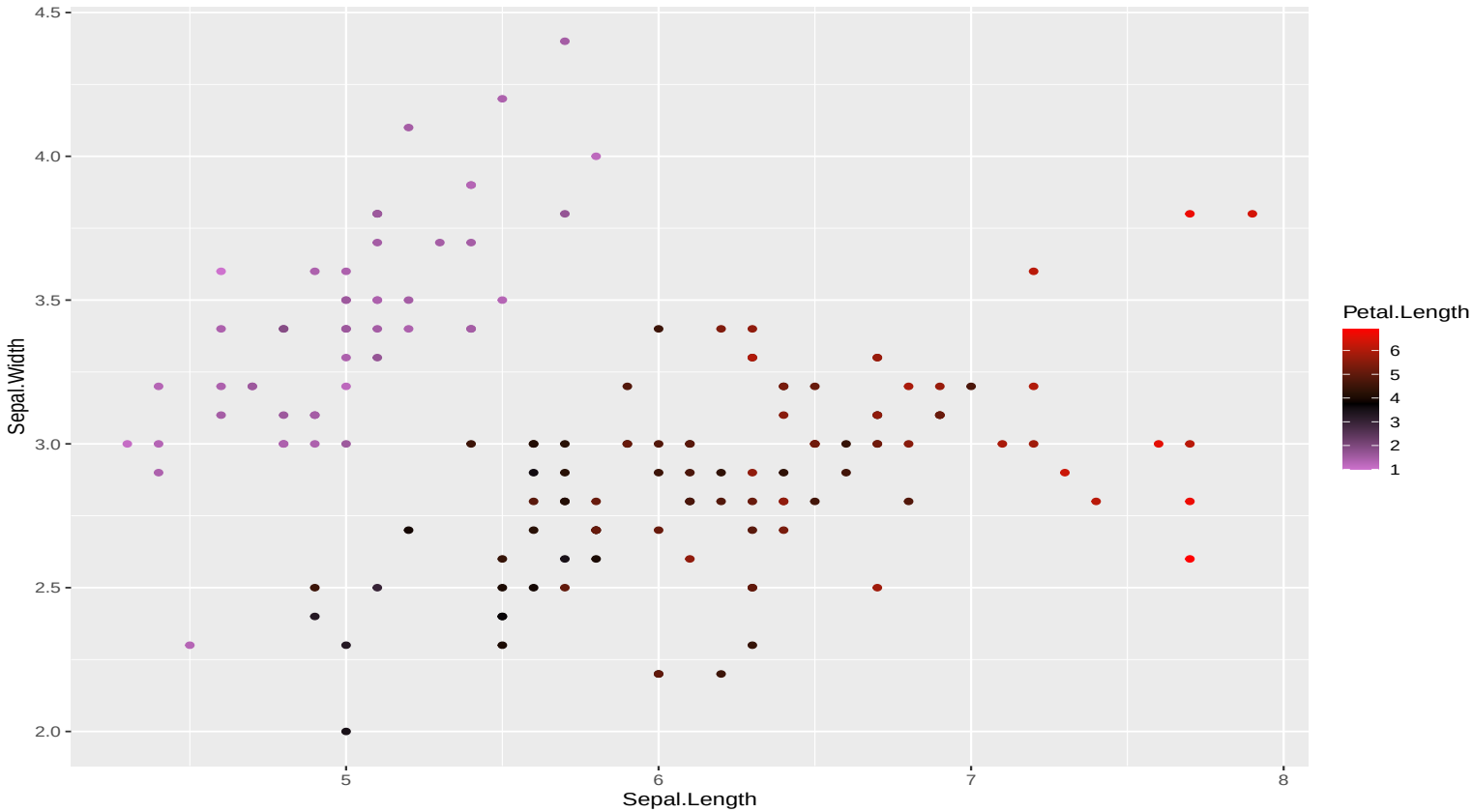
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length)) +
  geom_point() + scale_color_gradient2(low = "violet", high = "red",
                                       midpoint = mean(iris$Petal.Length))
```



Colors in ggplot2

Define the color for midpoint

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length)) +
  geom_point() +
  scale_color_gradient2(low = "violet", high = "red",
                        midpoint = mean(iris$Petal.Length), mid = 'black')
```



Viridis color palette

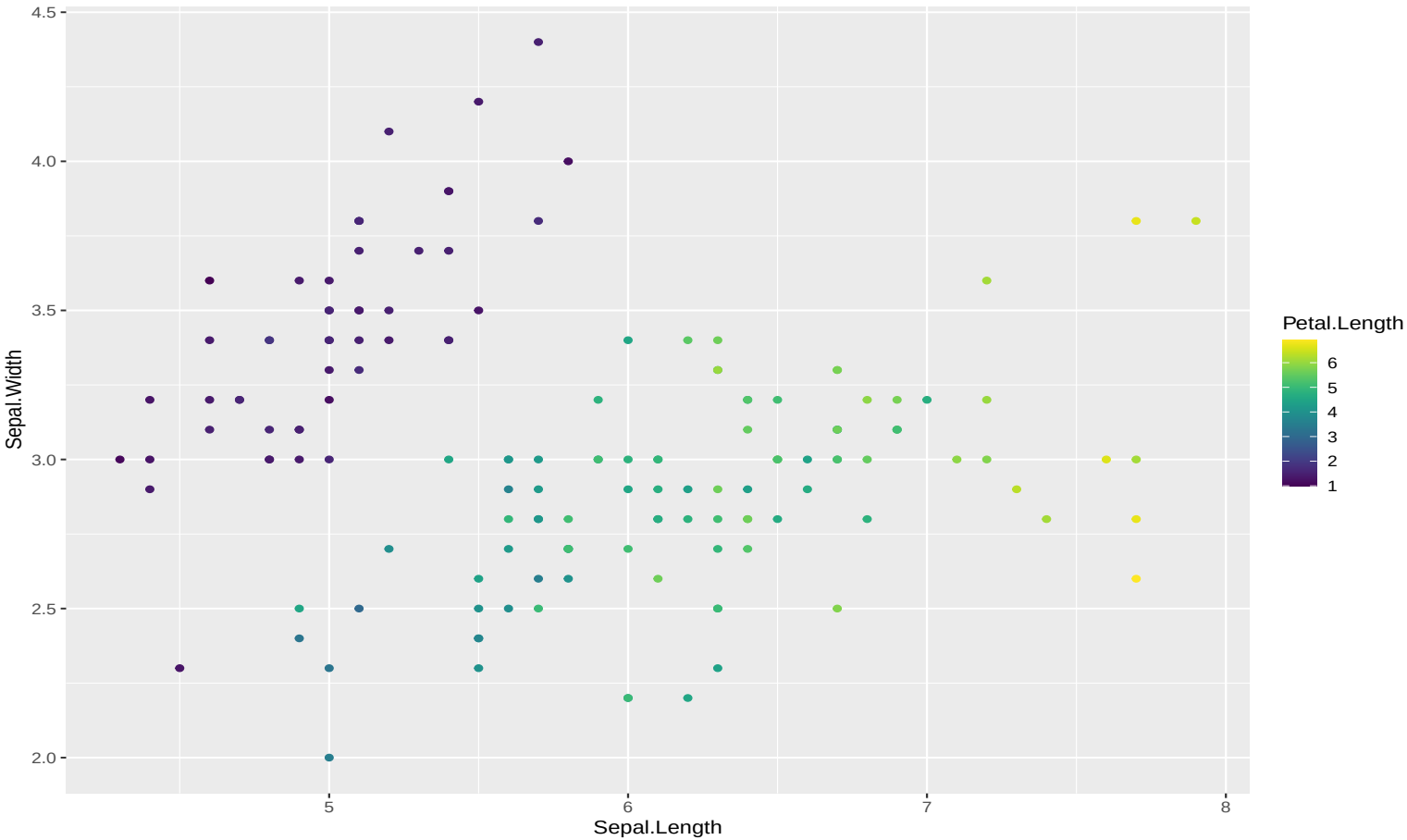
The viridis package developed by Stéfan van der Walt and Nathaniel Smith for the Python matplotlib library, to replace ugly MATLAB palettes.
Viridis color palettes is designed as:

- Colorful, spanning as wide palette as possible to make differences easy to see,
- Perceptually uniform, meaning that values close to each other have similar-appearing colors and values far away from each other have more different-appearing colors, consistently across the range of values,
- Robust to colorblindness, so that the above properties hold true for people with common forms of colorblindness,
- Well suited for gray scale printing, and
- Pretty (according to the authors)

Viridis color palette

scale_color_viridis() comes from a package viridis (even though ggplot has its own layers for viridis)

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Petal.Length)) +
  geom_point() + scale_color_viridis()
```



Viridis color palette

Four viridis perceptually uniform colormaps are available

- **inferno** perceptually uniform shades of black-red-yellow
- **magma** perceptually uniform shades of black-red-white
- **plasma** perceptually uniform shades of blue-red-yellow
- **viridis** perceptually uniform shades of blue-green-yellow