# post01-yanisa-cheeppensuk

*Yanisa*

*9/22/2017*

# Colors in R

## Introduction

Data visualization can be much more effective when you use colors to specifying the trend in your data, the different factor levels, how much your data deviate from the mean. Usually, in these cases, it becomes redundant to manually type out the different colors for each data point so R has some built-in functions and packages that can help you color your reports meaningfully. We will be discussing colors in {base_r} plotting functions and the package `ggplot2`.

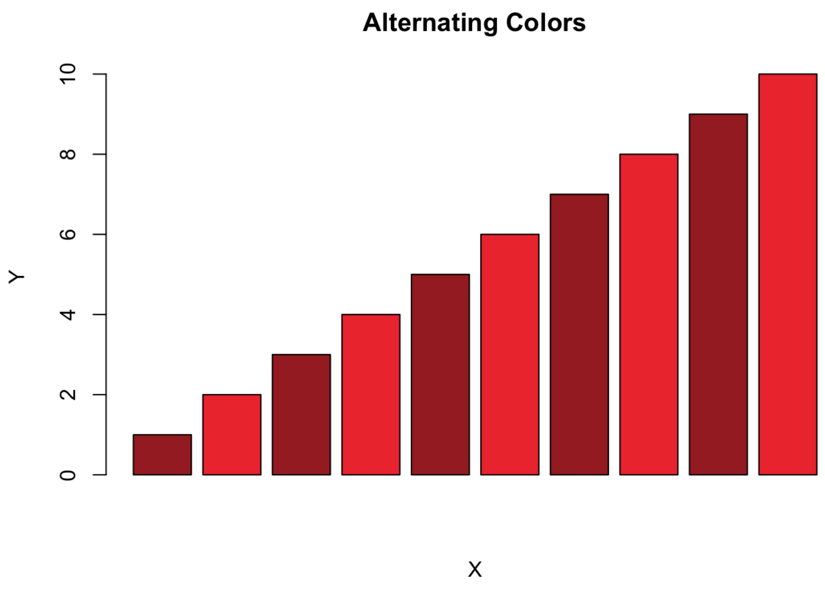
## Using {base_r} Functions

There are multiple ways where you can color your data plots, with the most common way being to pass the argument `col =` into the {base_r} plotting function. There is an extensive list of colors you can choose for your graph:

- http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf for color previews
- Call the function `colors()` to see the full list of colors available in R.

To color a graph in one single color, pass in one color as the argument. However, you may want to color the different plot points in different colors. To do this, the simplest way is to pass in the colors you want as a vector. This vector function also gets recycled i.e. if you have 10 plot points and you pass in a vector of length 2, each color is displayed 5 times.

```
barplot(seq(1:10), col = c("brown", "brown2"),
        main = 'Alternating Colors',
        ylab = 'Y', xlab = 'X')
```

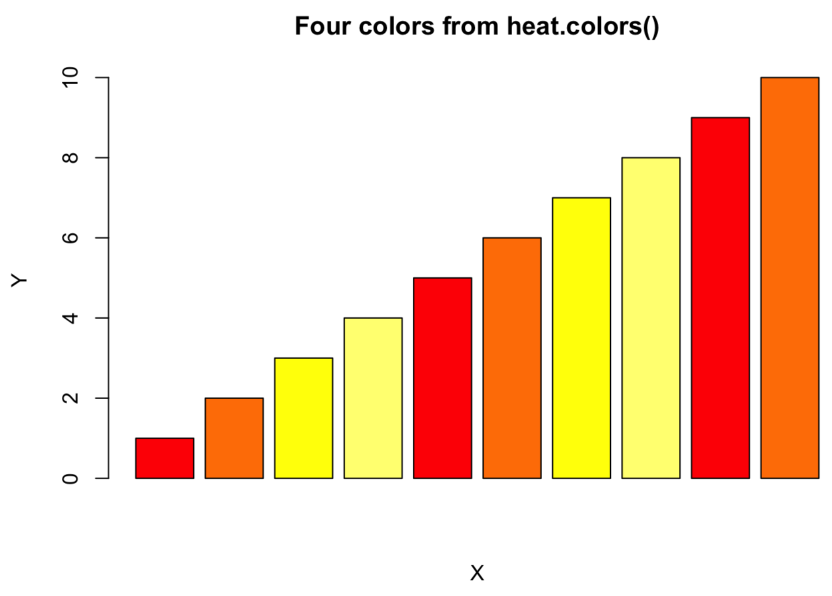


To avoid manually typing out every color names in a color vector, you can choose from some of the color hues built-in in R. Arguments for these functions are the numbers of color from the palette you want to use.

- `rainbow()`
- `heat.colors()` # for colors like red, orange and yellow
- `terrain.colors()` # for colors like green, yellow and brown
- `topo.colors()`
- `cm.colors()` # for colors like light pinks and blues

Here, we choose 4 colors from the palette `heat.colors()` which are recycled:

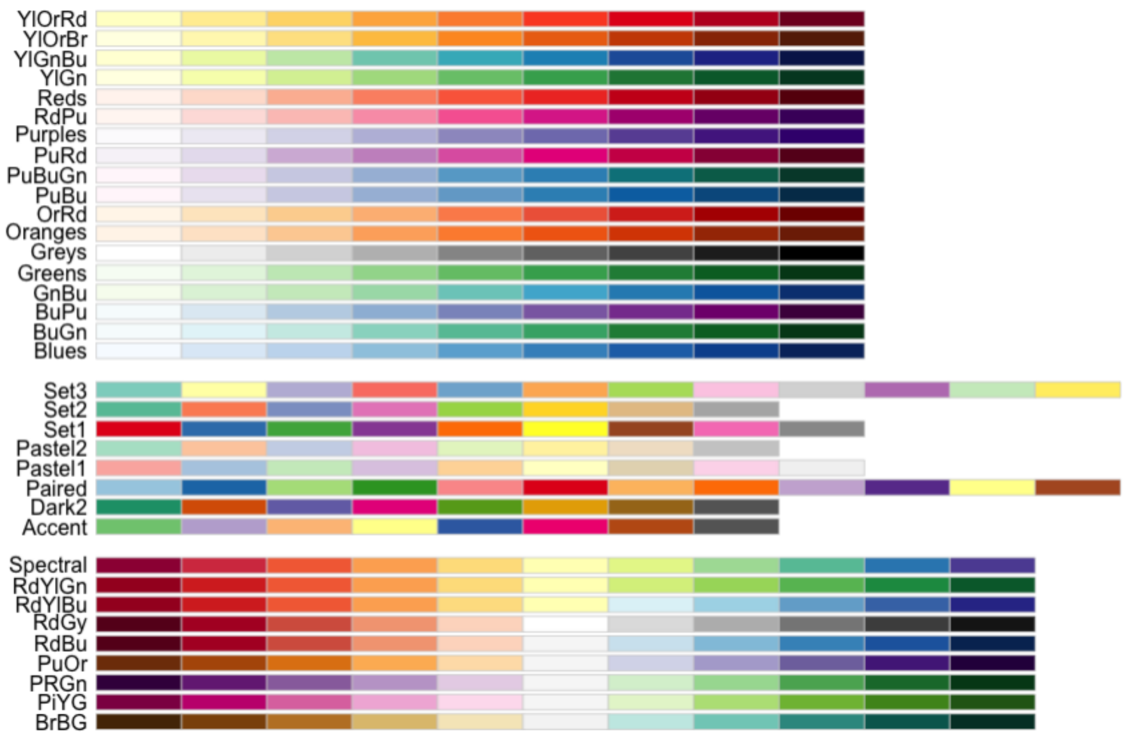
```
barplot(seq(1:10), col = heat.colors(4),
        main = 'Four colors from heat.colors()',
        ylab = 'Y', xlab = 'X')
```



## Using built-in color functions from `ggplot2`

```r
# install and load the package RColorBrewer and ggplot2
library(RColorBrewer)
library(ggplot2)
```

`ggplot()` offers more functions to color graphs, other than just `colour =`. Some useful functions also rely on `RColorBrewer`, which is a useful package that allows you to build your own color palette from the three different types of palettes available: sequential, qualitative and divergent (displayed in order below).



> **Sequential Color Palette** is helpful when you want to display data that increases from low to high. **Divergent** is similar to Sequential but allows you to have one color as the standard color, usually for the mean or the 0 value, while **Qualitative** have distinct colors and are useful for coloring in the different factor levels as it does not display the magnitude of difference between the data points.

The package `ggplot()` has built-in functions you can use to color your graphs, these functions also automatically generate a legend for the colors used to plot the graphs.

**Discrete Colors** use functions such as `scale_colour_brewer()` or `scale_fill_brewer()` to color your graphs based on discrete categories, such as factor levels. They work similarly, with the latter being for fill colours, such as for bar graphs.

We will be using this data frame of 30 individuals to plot our graphs. There are four columns: Height, Sex, Weight, Age.

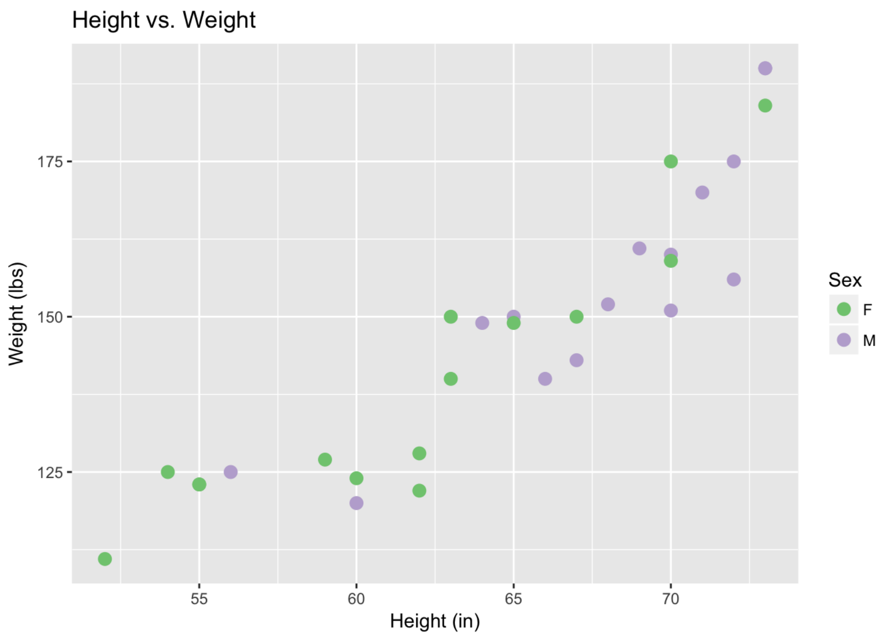
```
# data frame used to generate plots in this post
biodata <- data.frame(
  col1 = c(63,70,70,66,73,55,60,62,72,73,
           56,55,60,65,70,63,52,67,64,70,
           54,59,62,65,67,68,69,71,72,73),
  col2 = c("F","M","F","M","M","F","M","F","M","F",
           "M","F","F","M","F","F","F","M","M","M",
           "F","F","F","F","F","M","M","M","M","M"),
  col3 = c(140,160,159,140,190,123,120,122,156,184,
           125,123,124,150,175,150,111,143,149,151,
           125,127,128,149,150,152,161,170,175,190),
  col4 = c(16,19,22,15,19,14,16,22,23,20,
           19,18,17,18,22,19,17,18,19,21,
           15,17,18,19,19,20,21,22,21,22),
  stringsAsFactors = TRUE
)
names(biodata) <- c('Height','Sex','Weight','Age')
head(biodata)
```

```
##   Height Sex Weight Age
## 1     63   F    140  16
## 2     70   M    160  19
## 3     70   F    159  22
## 4     66   M    140  15
## 5     73   M    190  19
## 6     55   F    123  14
```

```
# scatterplot of height vs. weight, color by sex
plot <- ggplot(biodata) +
  geom_point(aes(x = biodata$Height, y = biodata$Weight, colour = biodata$Sex),
             size = 3) +
  labs(title = 'Height vs. Weight',
       x = 'Height (in)', y = 'Weight (lbs)')
```

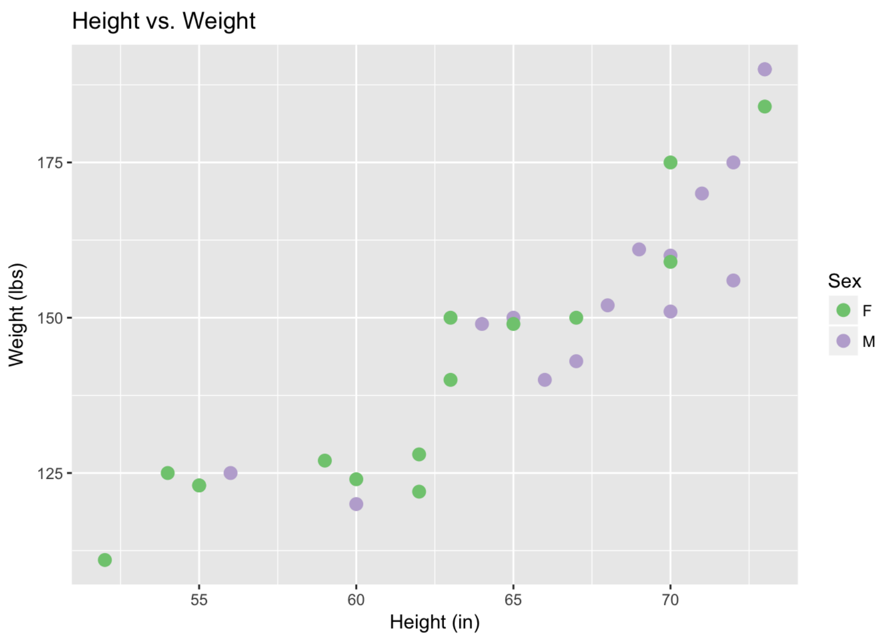
We can color our plot using the palette ‘Accent’:

```
plot + scale_colour_brewer('Sex', palette = 'Accent')
```



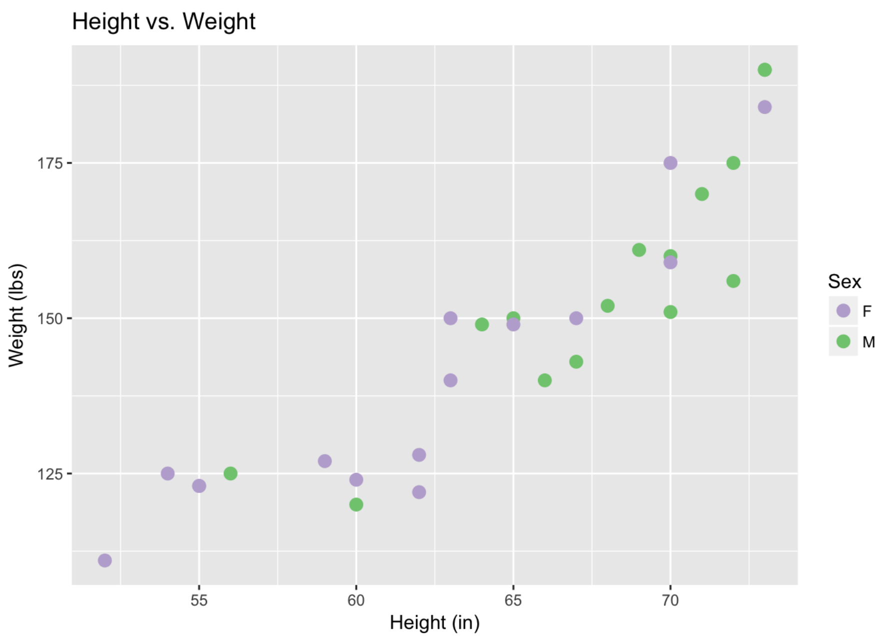
The same results can be achieved by referring to the relative position of each palette in its palette category:

```
plot + scale_colour_brewer('Sex', type = 'qual', palette = 1)
```



The order can also be reversed using the argument direction:

```
plot + scale_colour_brewer('Sex', palette = 'Accent', direction = -1)
```



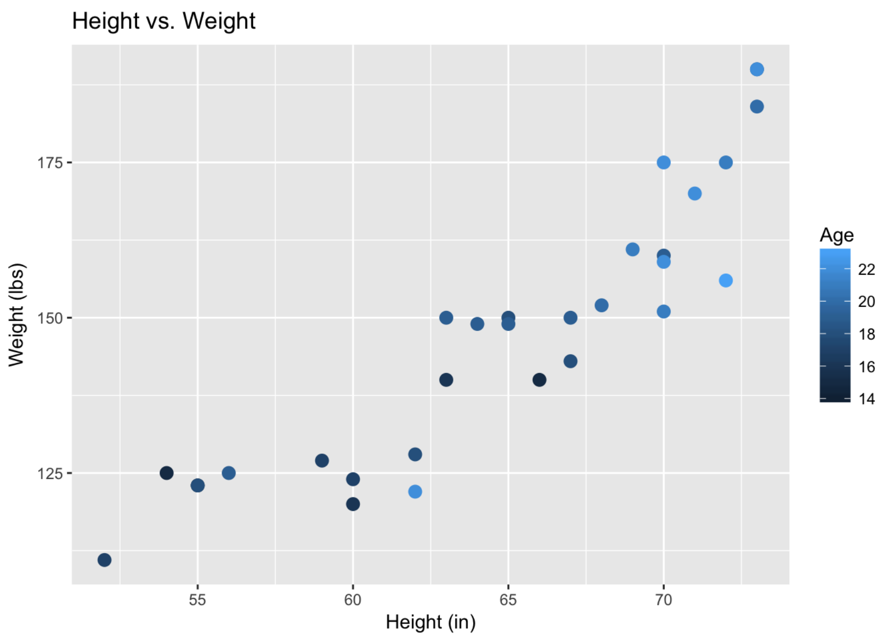
**Continuous Colors** use functions such as `scale_colour_gradient()`, `scale_colour_gradient2()` and `scale_colour_gradientn()` to color your graph in increasing/decreasing shades. You can use the default color scale, specify your own colors or use one of the palettes available.

- `scale_colour_gradient()`: Default blue colors or builds a gradient from two colors
- `scale_colour_gradient2()`: Builds a gradient from colors representing low-, mid- and high-values
- `scale_colour_gradientn()`: Uses one of the color palette in R

```
# scatterplot of height vs. weight, color by age
plot2 <- ggplot(biodata) +
  geom_point(aes(x = biodata$Height, y = biodata$Weight, colour = biodata$Age),
             size = 3) +
  labs(title = 'Height vs. Weight',
       x = 'Height (in)', y = 'Weight (lbs)')
```

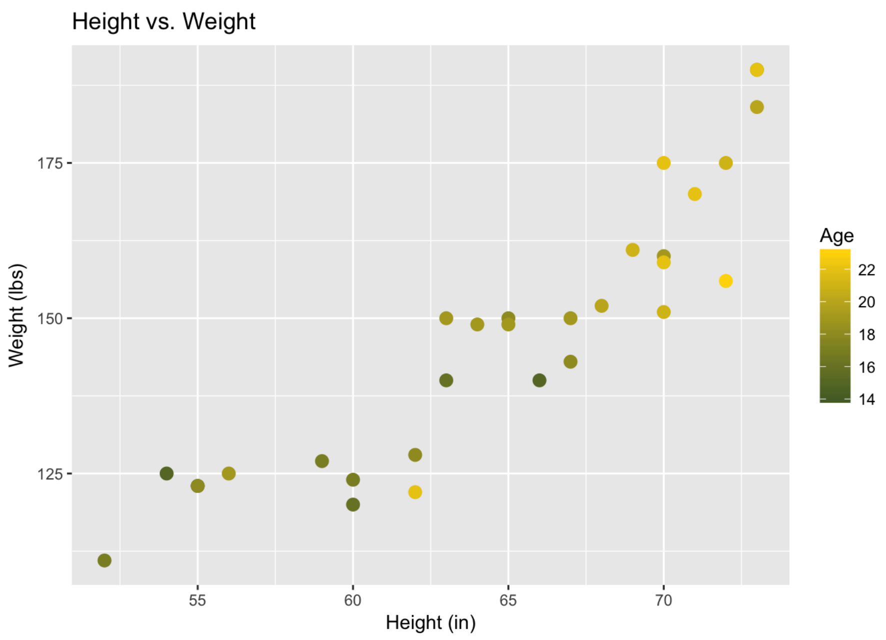
Default color gradient in R is from dark blue to light blue and the argument 'Age' labels the legend:

```
plot2 + scale_colour_gradient('Age')
```



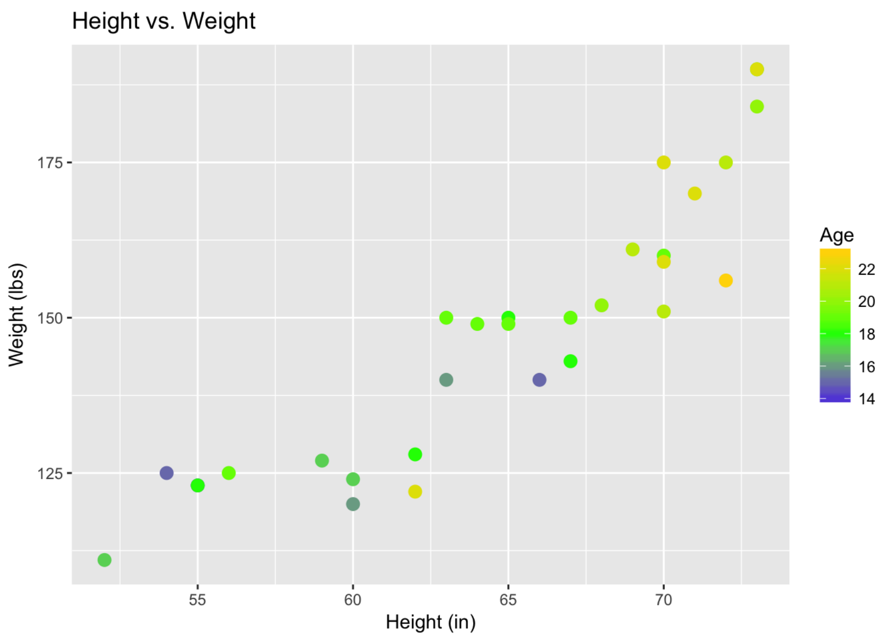
You can choose the gradient manually by specifiying the colors for low and high values:

```
plot2 + scale_colour_gradient('Age', low = 'darkolivegreen', high = 'gold')
```



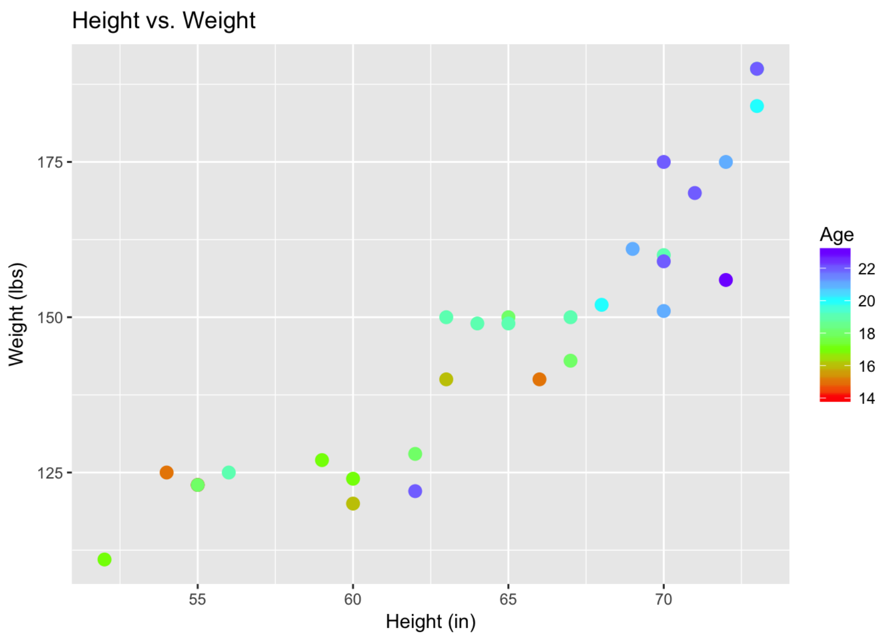
Use a diverging color palette by specifying low-mid-high colors:

```
plot2 + scale_colour_gradient2('Age',
                               low = 'blue', mid = 'green', high = 'gold',
                               midpoint = 18) # midpoint value is defaulted to 0
```



Using the palette rainbow available in R:

```
plot2 + scale_colour_gradientn('Age', colours = rainbow(4))
```



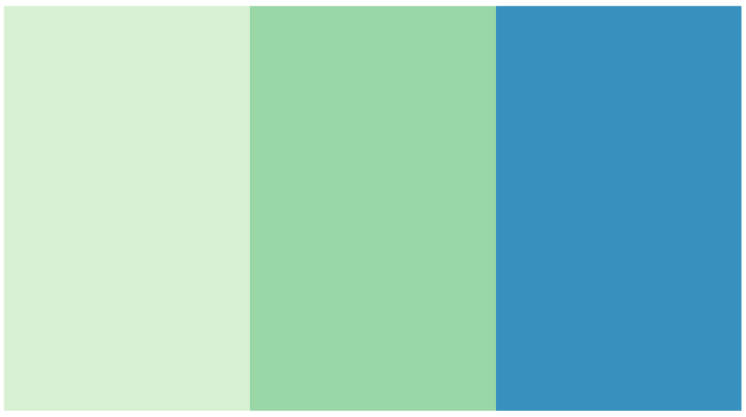
- Careful! the palettes available in RColorBrewer cannot simply be passed into the argument `colours =` in the function above because it will result in an error. To do so, we have to make the palette available in R first, more on that below.

## More on `RColorBrewer`

`RColorBrewer` is a useful package that allows you to build you own color palette, using the function `brewer.pal()`. It takes in the argument which specifies the colors that is to make up that new palette, making an RColorBrewer palette available in R.

First, we choose the colors from the `RColorBrewer` palette we want:

```
# choose three colors from green and blue palette
display.brewer.pal(3, 'GnBu')
```

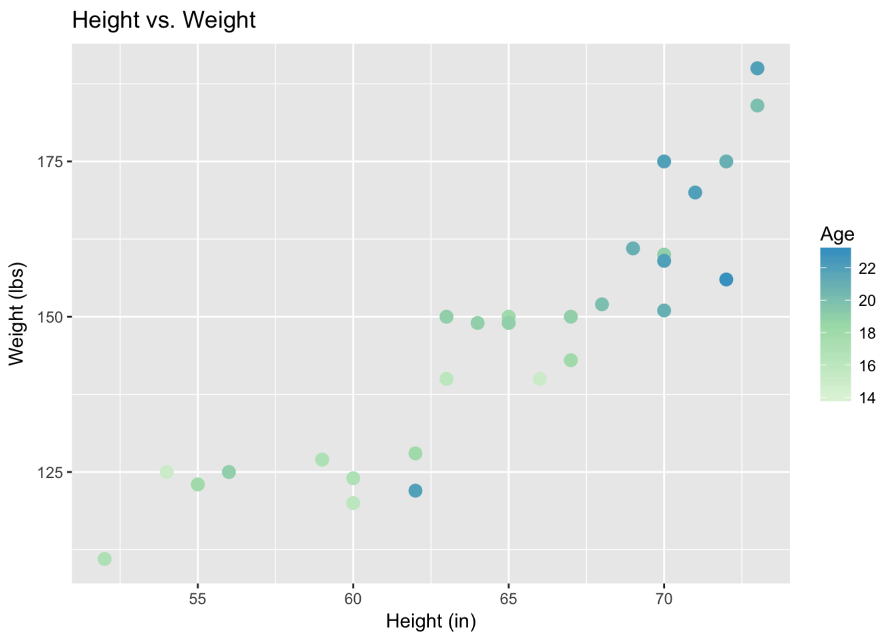


GnBu (sequential)

```r
# creating a palette from the 3 colors chosen
minigb <- brewer.pal(3, 'GnBu')
```

Now we can use colors from `RcolorBrewer` palettes in the function `scale_colour_gradientn()` :

```r
plot2 + scale_colour_gradientn('Age', colours = minigb)
```



> Fun Fact! RColorBrewer has the optional argument `colorblindFriendly` which is defaulted to FALSE. By choosing TRUE, only such colors will be chosen and displayed.

## Conclusion

Colors in R are more useful than just making our data visualization less boring. They could pack more information into our graphs on top of the axes we choose. `ggplot2` has helpful functions to help us color discrete and continuous data sets, with `scale_colour_brewer()` and `scale_colour_gradient()` being a few of them.

As we explore further into the colors available in R, there are functions which allow us to build color palettes from scratch (see: HCL colors) which may be useful when we want to use a palette with exact shades of colors to symbolize what the data points represent in real life.

## References

- https://stat.ethz.ch/R-manual/R-devel/library/grDevices/html/palettes.html
- https://www.google.com/search?client=safari&rls=en&q=brewer.pal&ie=UTF-8&oe=UTF-8
- http://ggplot2.tidyverse.org/reference/scale_brewer.html
- https://cran.r-project.org/web/packages/RColorBrewer/RColorBrewer.pdf
- https://www.rdocumentation.org/packages/RColorBrewer/versions/1.1-2/topics/RColorBrewer
- http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/
- http://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3