9: Data Visualization Advanced

Environmental Data Analytics | Kateri Salk

Spring 2020

LESSON OBJECTIVES

1. Perform advanced edits on ggplot objects to follow best practices for data visualization
2. Troubleshoot visualization challenges

SET UP YOUR DATA ANALYSIS SESSION

```
getwd()
```

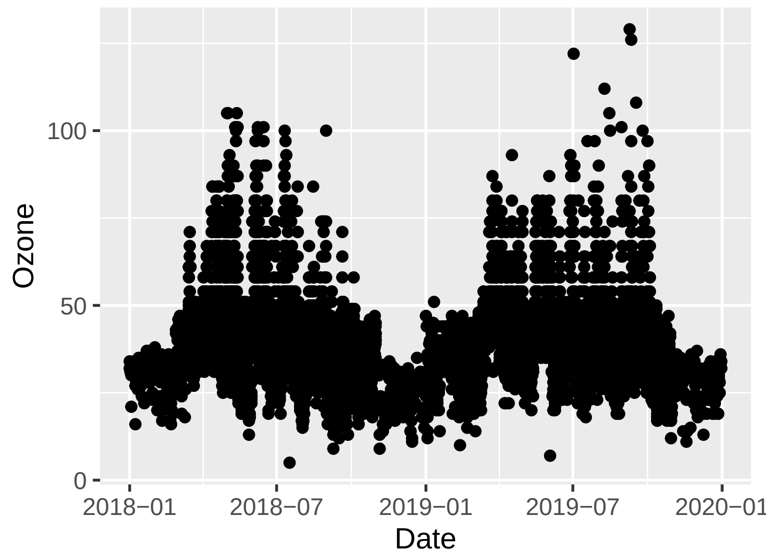
```
## [1] "/Users/emilymcnamara/Desktop/Env Data Analytics/Environmental_Data_Analytics_2020"
```

```
library(tidyverse)

PeterPaul.chem.nutrients <-
  read.csv("./Data/Processed/NTL-LTER_Lake_Chemistry_Nutrients_PeterPaul_Processed.csv")
PeterPaul.chem.nutrients.gathered <-
  read.csv("./Data/Processed/NTL-LTER_Lake_Nutrients_PeterPaulGathered_Processed.csv")
EPAair <- read.csv("./Data/Processed/EPAair_O3_PM25_NC1819_Processed.csv")

EPAair$Date <- as.Date(EPAair$Date, format = "%Y-%m-%d")
PeterPaul.chem.nutrients$sampledate <- as.Date(
  PeterPaul.chem.nutrients$sampledate, format = "%Y-%m-%d")
PeterPaul.chem.nutrients.gathered$sampledate <- as.Date(
  PeterPaul.chem.nutrients.gathered$sampledate, format = "%Y-%m-%d")
```

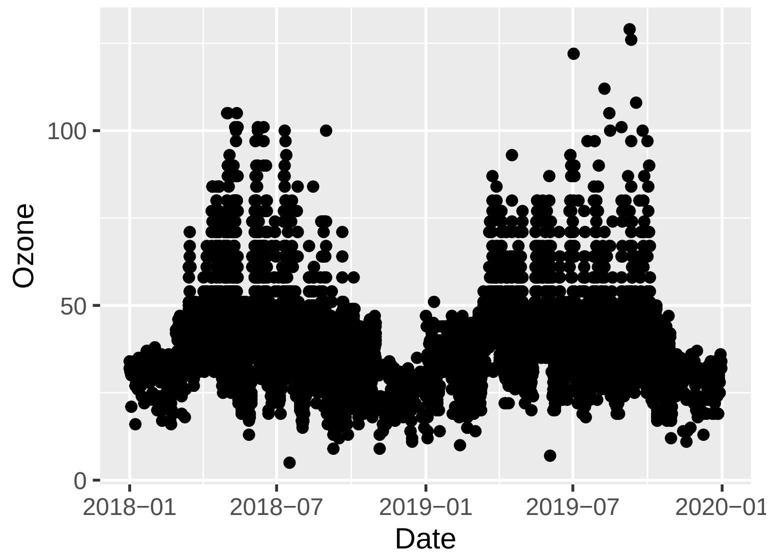
Themes

Often, we will want to change multiple visual aspects of a plot. Ggplot comes with pre-built themes that will adjust components of plots if you call that theme.

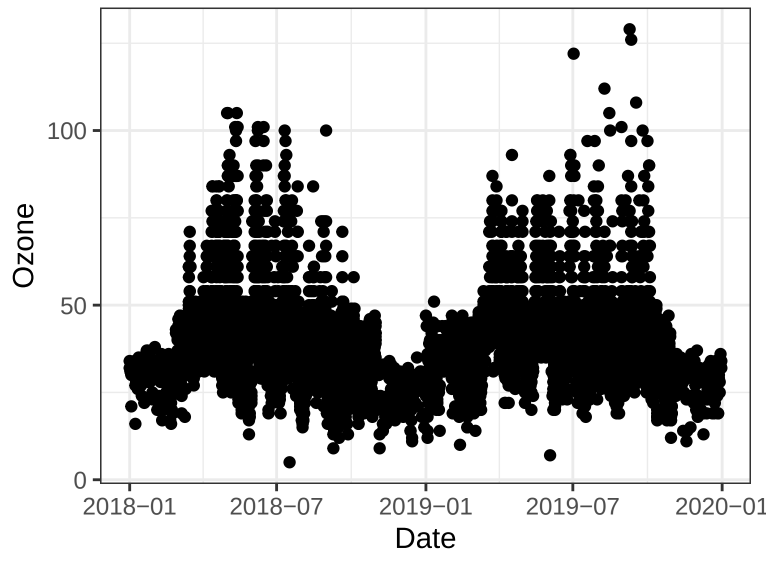
```
O3plot <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone))
print(O3plot)
```



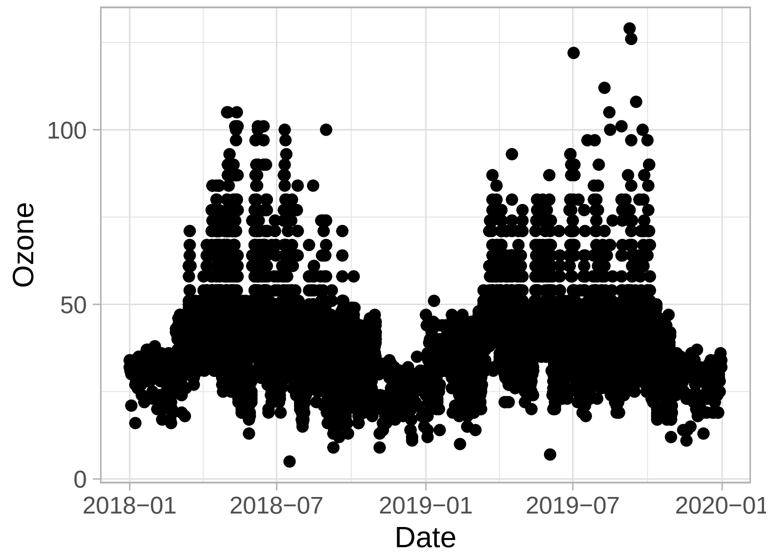
```r
O3plot1 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  theme_gray()
print(O3plot1)
```



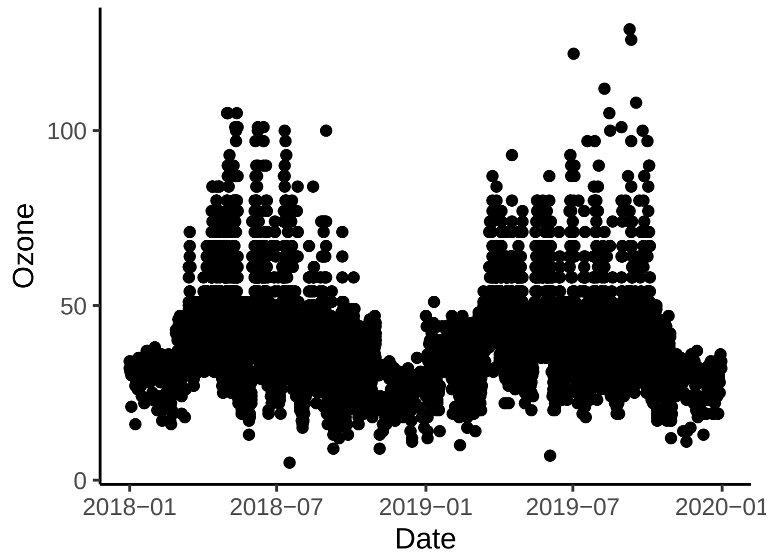
```r
# GGplot comes with several pre-made themes. i.e. theme_gray

O3plot2 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  theme_bw()
print(O3plot2)
```



```
# Theme_bw includes dark line all around plot and light line to show dif. units of axis

O3plot3 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  theme_light()
print(O3plot3)
```



```
O3plot4 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  theme_classic()
print(O3plot4)
```



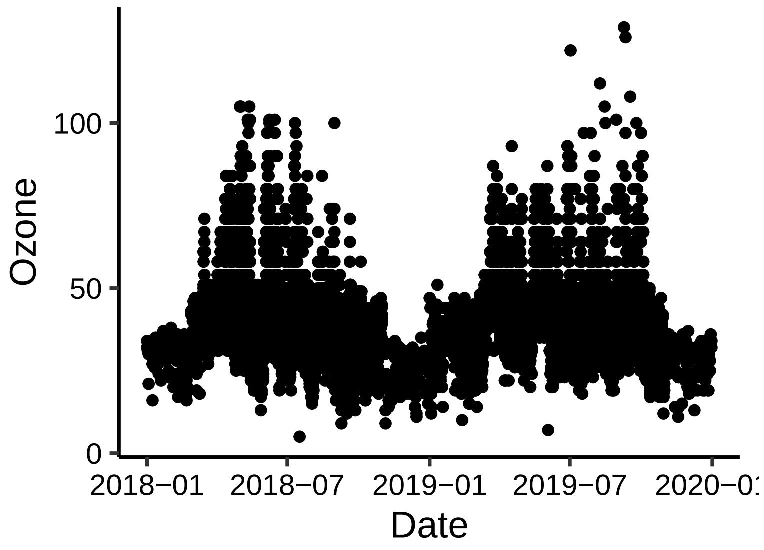
```
# Kateri's favorite. just outlines axis.
#Themes don't adjust text size, label colors, anything to due with legend (position/justification)
```

Notice that some aspects of your graph have not been adjusted, including:

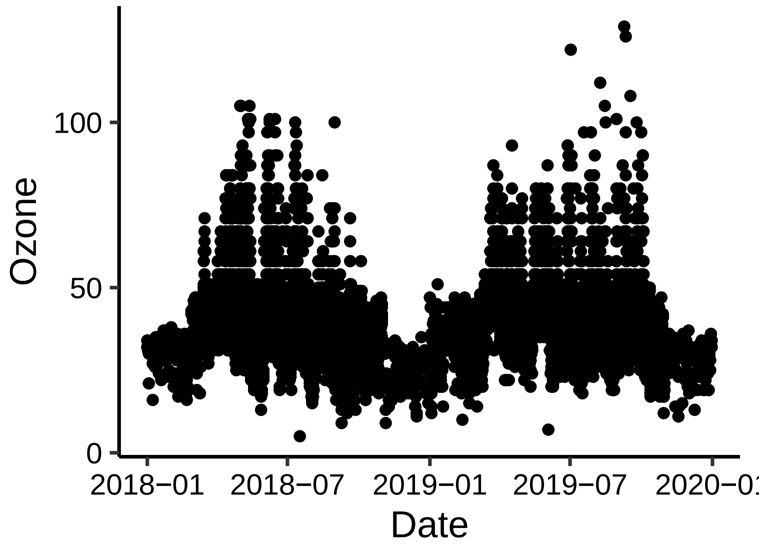
- text size
- axis label colors
- legend position and justification

If you would like to set a common theme across all plots in your analysis session, you may define a theme and call up that theme for each graph. This eliminates the need to add multiple lines of code in each plot.

```
mytheme <- theme_classic(base_size = 14) +
  theme(axis.text = element_text(color = "black"),
        legend.position = "top") #alternative: legend.position + legend.justification

# Says: start with theme_classic, make base size 14, plus change things aside from general theme like a
# options: call the theme in each plot or set the theme at the start.

# Can add "my_theme" to any plot

O3plot5 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  mytheme
print(O3plot5)
```



```
# colors on tickmarks are black, text is a little bigger but overall pulling from theme_classic.

theme_set(mytheme)
# can use this to set it as theme at beginning of session and then all plots will use it so you don't h

O3plot6 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone))
print(O3plot6)
```

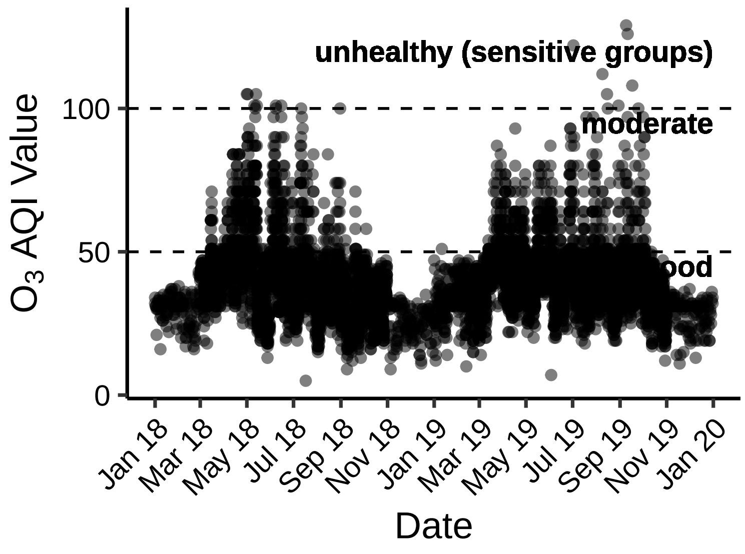


```
# so now this runs with all the elements of "my_theme" without having to write it in code
# help function has a lot of info on different elements you can change
# designated as "large gg" in 'Environment' because we used a theme, but shows up regular once theme is
# text size is a common thing to change (figure for paper vs. presentation will necessitate different f
```

Adjusting multiple components of your plots

While the theme allows us to set multiple aspects of plots, ggplot allows us to adjust other parts of plots outside of the theme.

```r
O3plot7 <- ggplot(EPAair, aes(x = Date, y = Ozone)) +
  geom_hline(yintercept = 50, lty = 2) +
  geom_hline(yintercept = 100, lty = 2) +
  geom_point(alpha = 0.5, size = 1.5) +
  geom_text(x = as.Date("2020-01-01"), y = 45, label = "good", hjust = 1, fontface = "bold") +
  geom_text(x = as.Date("2020-01-01"), y = 95, label = "moderate", hjust = 1, fontface = "bold") +
  geom_text(x = as.Date("2020-01-01"), y = 120, label = "unhealthy (sensitive groups)", hjust = 1, font
  scale_x_date(limits = as.Date(c("2018-01-01", "2019-12-31")),
    date_breaks = "2 months", date_labels = "%b %y") +
  ylab(expression("O"[3]* " AQI Value")) +
  theme(axis.text.x = element_text(angle = 45,  hjust = 1))
print(O3plot7)
```



```r
# New plot includes: Same things as before in first line of code followed by several layers
# AQI values are set to standard scale across all different pollutants thats associated with 'good' 'he
#geom_hline: lty means line type. lty = 2 means dashed line. can also adjust thickness.
# geom_point aesthetic added after adding horizontal line because ggplot adds things in layers so it's
# Geom_text: have to specify where you want to put it on plot (x and y coords), where label will be, fo
#alpha = .5 means that it's 50% transparent with size 1.5
# scale_x_date = start and end point. Makes a break every 2/3/4/etc months by saying date_breaks = '2 m
# date_labels: %b = 3 letter month, %B = whole month spelled out, %y = 2 digit year
# ylab = express the units as well. anything within parenthesis will be specified as is and [3] outside
# Put angle at 45 degrees and hjust = 1 makes it on right side
```

Color palettes

Color palettes are an effective way to communicate additional aspects of our data, often illustrating a third categorical or continuous variable in addition to the variables on the x and y axes. A few rules for choosing colors:

- Consider if your plot needs to be viewed in black and white. If so, choose a sequential palette with varying color intensity.

- Choose a palette that is color-blind friendly
- Maximize contrast (e.g., no pale colors on a white background)
- Diverging color palettes should be used for diverging values (e.g., warm-to-cool works well for values on a scale encompassing negative and positive values)

Does your color palette communicate additional and necessary information? If the answer is no, then you might consider removing it and going with a single color. Common instances of superfluous or redundant color palettes include:

- Color that duplicates an axis
- Color that distinguishes categories when labels already exist (exception: if category colors repeat throughout a series of interrelated visualizations and help the reader build a frame of reference across a report)
- Color that reduces the conciseness of a plot

Perception is key! Choose palettes that are visually pleasing and will communicate what you are hoping your audience to perceive.

RColorBrewer (package). Can choose number of data classes you have, pull specific hex codes, can pick various palettes

- http://colorbrewer2.org
- https://moderndata.plot.ly/create-colorful-graphs-in-r-with-rcolorbrewer-and-plotly/

viridis and viridisLite (packages). One of Kateri's favorites

- https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html
- https://ggplot2.tidyverse.org/reference/scale_viridis.html

colorRamp (function; comes with base R as part of the grDevices package)

- https://bookdown.org/rdpeng/exdata/plotting-and-color-in-r.html#colorramp

LaCroixColoR (package)

- https://github.com/johannesbjork/LaCroixColoR

wesanderson (package)

- https://github.com/karthik/wesanderson

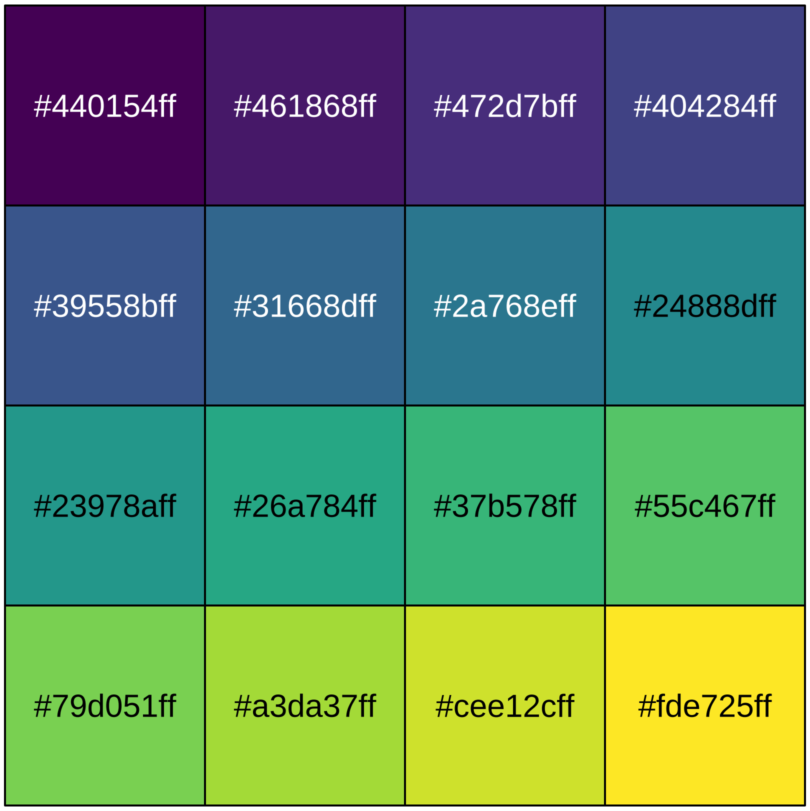
nationalparkcolors (package). Includes code to install color palettes

- https://github.com/katiejolly/nationalparkcolors

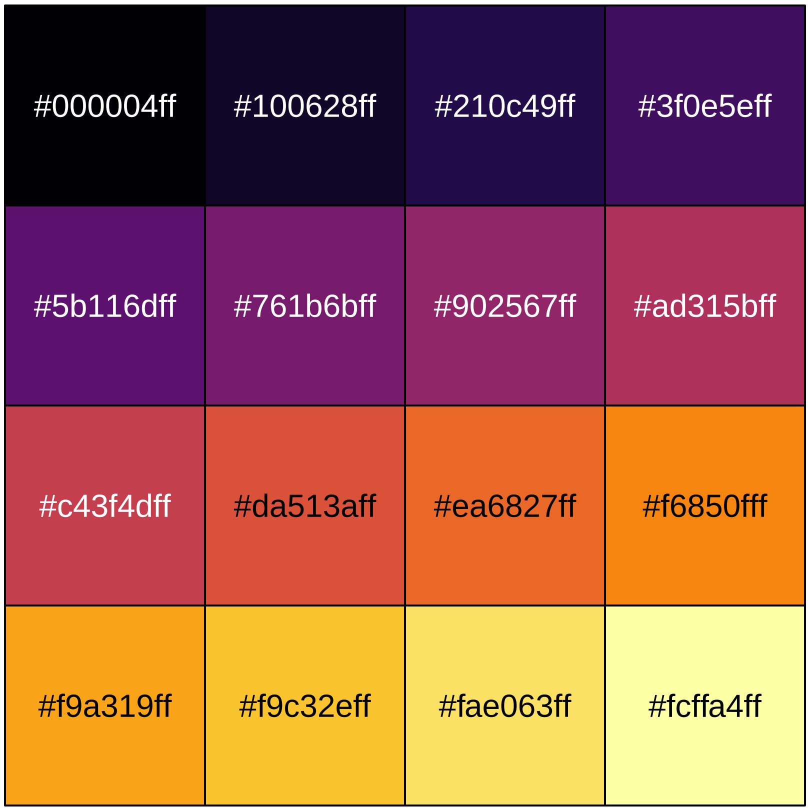
```r
#install.packages("viridis")
#install.packages("RColorBrewer")
#install.packages("colormap")
library(viridis)
```

```
## Loading required package: viridisLite
```

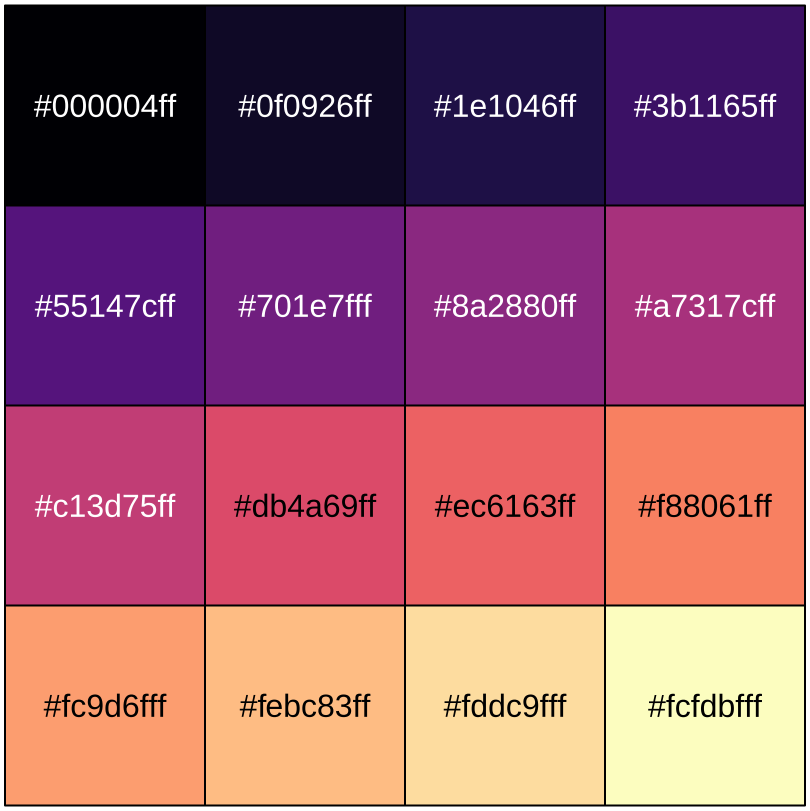
```r
library(RColorBrewer)
library(colormap)

scales::show_col(colormap(colormap = colormaps$viridis, nshades = 16))
```



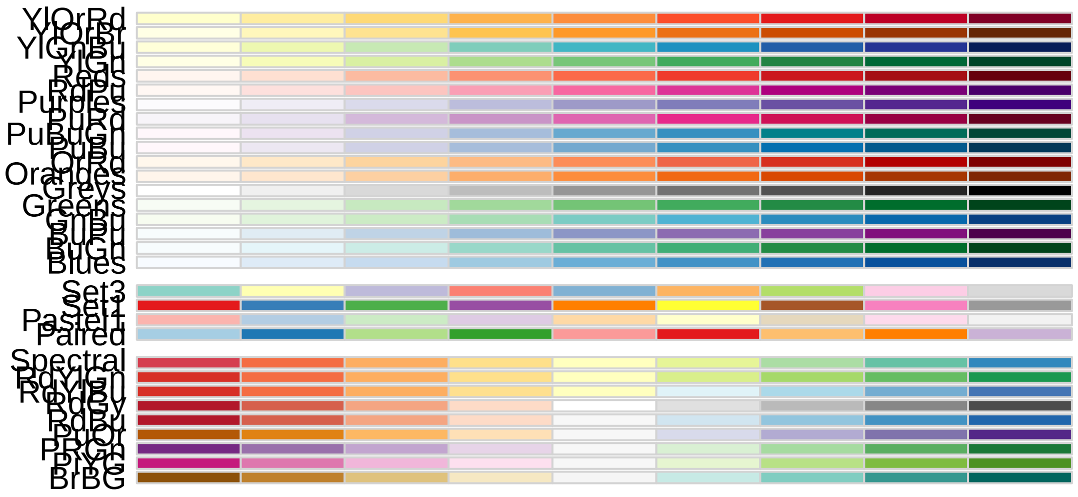
```
scales::show_col(colormap(colormap = colormaps$inferno, nshades = 16))
```



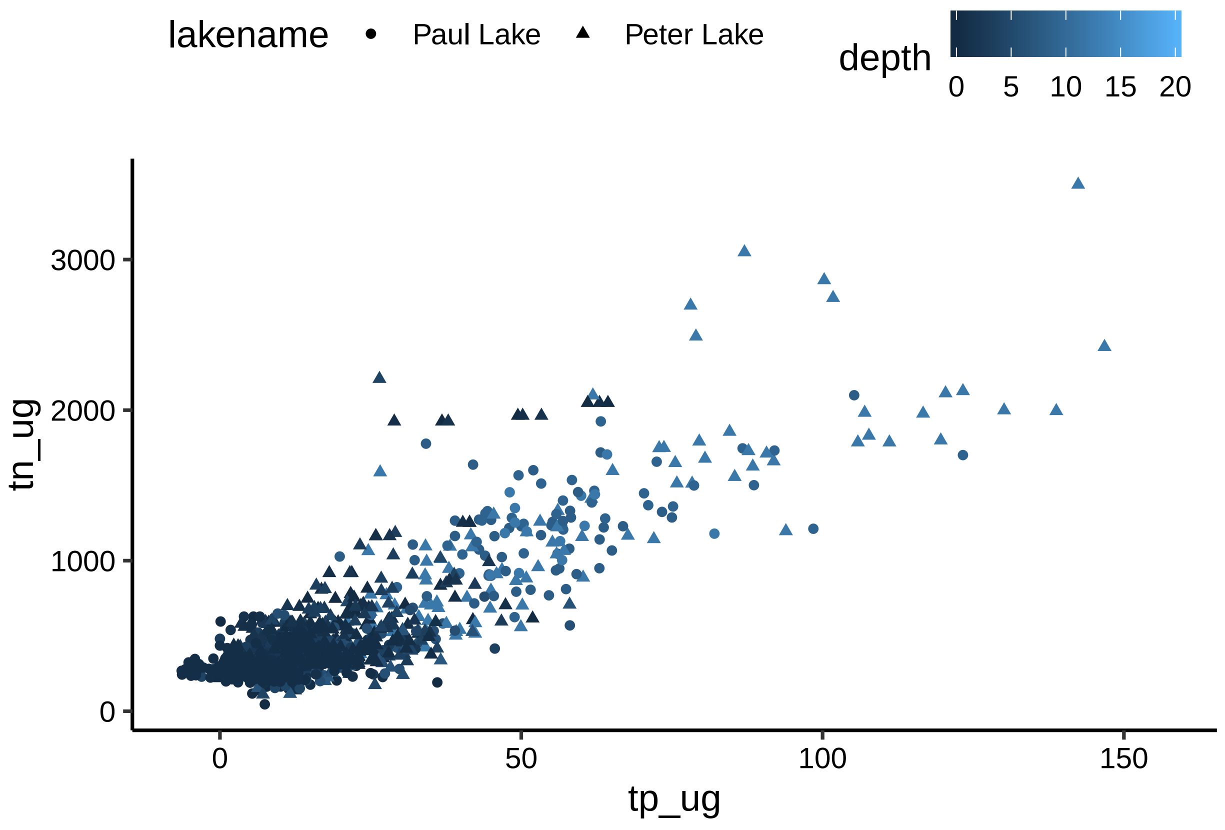
```
scales::show_col(colormap(colormap = colormaps$magma, nshades = 16))
```



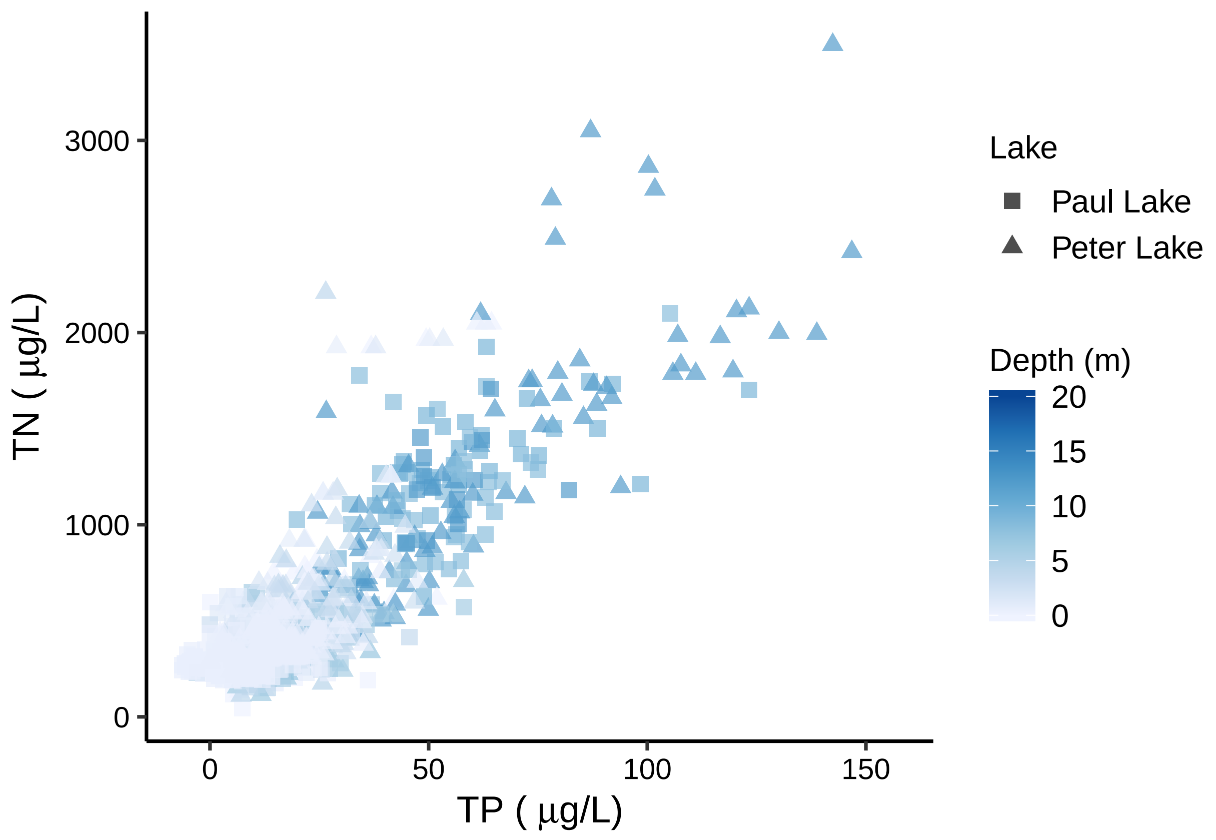
```r
display.brewer.all(n = 9)
```



```
NvsP <-
  ggplot(PeterPaul.chem.nutrients, aes(x = tp_ug, y = tn_ug, color = depth, shape = lakename)) +
  geom_point()
print(NvsP)
```



```
# let's first make the plot look better.
# change your axis labels to reflect TN and TP in micrograms per liter.
# change your legend labels
NvsP2 <-
  ggplot(PeterPaul.chem.nutrients, aes(x = tp_ug, y = tn_ug, color = depth, shape = lakename)) +
  geom_point(alpha = 0.7, size = 2.5) +
  labs(x = expression(paste("TP ( ", mu,"g/L)")),
       y = expression(paste("TN ( ", mu,"g/L)")),
       color = "Depth (m)", shape = "Lake") + #could specify epmty parenthesis so it just says Paul and
  scale_shape_manual(values = c(15, 17)) +
  scale_color_distiller(palette = "Blues", direction = 1) + # use scale_color_brewer for discrete varia
  #scale_color_viridis(option = "magma", direction = -1, end = 0.8) +
  theme(legend.position = "right",
        legend.text = element_text(size = 12), legend.title = element_text(size = 12))
print(NvsP2)
```



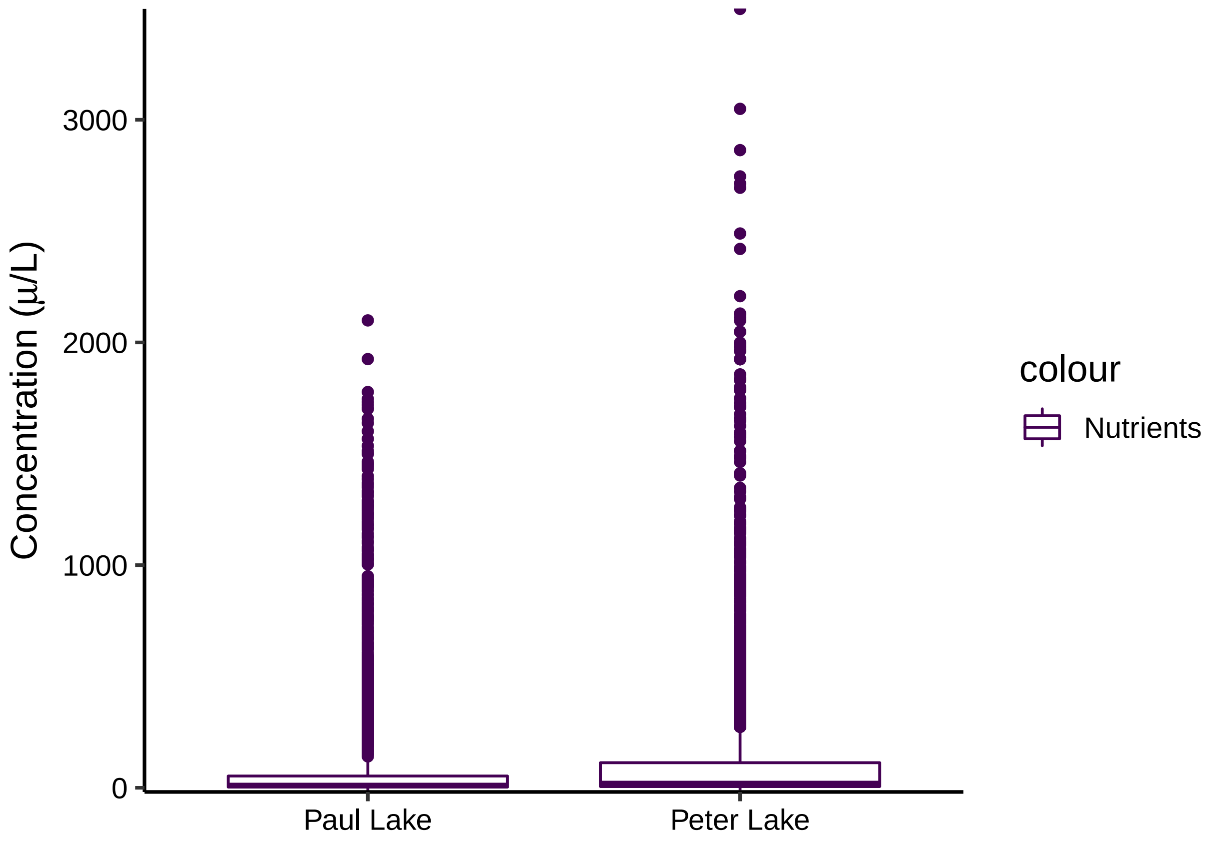
```
# one of scale color viridis or distiller should be commented out.



# change your y axis label to list concentration in micrograms per liter
# remove your x axis label
# change labels for nutrients in the legend
# try out the different color palette options and choose one (or edit)

  ylab(expression("O"[3]* " AQI Value"))
```

```
## $y
## expression("O"[3] * " AQI Value")
##
## attr(,"class")
## [1] "labels"
```

```
Nutrientplot <-
  ggplot(PeterPaul.chem.nutrients.gathered, aes(x = lakename, y = concentration, color = "Nutrients")) +
  geom_boxplot() +
  labs(y=expression(paste("Concentration (",mu,"/L)")))+
  theme(axis.title.x = element_blank())+
# place your additional edits here
  scale_y_continuous(expand = c(0, 0)) +
  #scale_color_discrete(labels=c("Nutrient 1", "Nutrient 2", "Nutrient 3", "Nutrient 4", "Nutrient 5"))
  scale_color_brewer(palette = "YlGnBu") +
  #scale_color_manual(values = c("#7fcdbb", "#41b6c4", "#1d91c0", "#225ea8", "#0c2c84")) +
  scale_color_viridis(discrete = TRUE, end = 0.8) +
  theme(legend.position = "right")
```

```
## Scale for 'colour' is already present. Adding another scale for 'colour',
## which will replace the existing scale.
```

```
print(Nutrientplot)
```



Multiple plots on a page

In situations where facets don't fill our needs to place multiple plots on a page, we can use the package `cowplot` to arrange plots. The `plot_grid` function is extremely flexible in its ability to arrange plots in specific configurations. A useful guide can be found here: https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html.

A useful guide for aligning plots by axis can be found here: https://wilkelab.org/cowplot/articles/aligning_plots.html

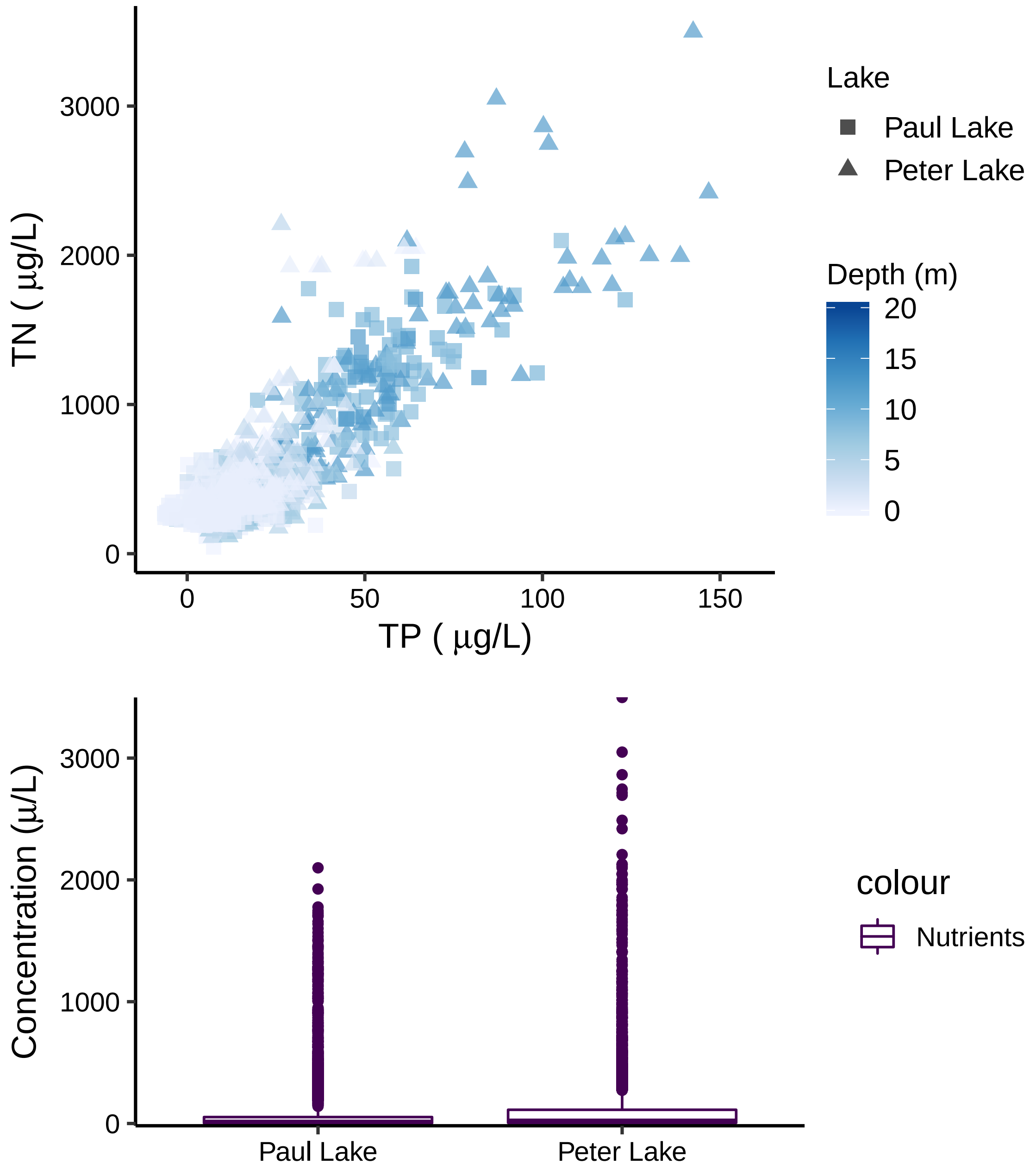
```
#install.packages("cowplot")
library(cowplot)
```

```
##
## ********************************************************

## Note: As of version 1.0.0, cowplot does not change the

##   default ggplot2 theme anymore. To recover the previous

##   behavior, execute:
```

```
##   theme_set(theme_cowplot())
## ********************************************************
```

```
plot_grid(NvsP2, Nutrientplot, nrow = 2, align = 'h', rel_heights = c(1.25, 1))
```

```
## Warning: Removed 21587 rows containing missing values (geom_point).
```



```
# allows to to plot both graphs on same plot
```

Saving plots

The `ggsave` function allows you to save plots in jpg, png, eps, pdf, tiff, and other formats. The following information can be supplied:

- filename and relative path, with file extension and in quotes (required)
- plot object (required)
- width, height, units
- resolution (dpi)

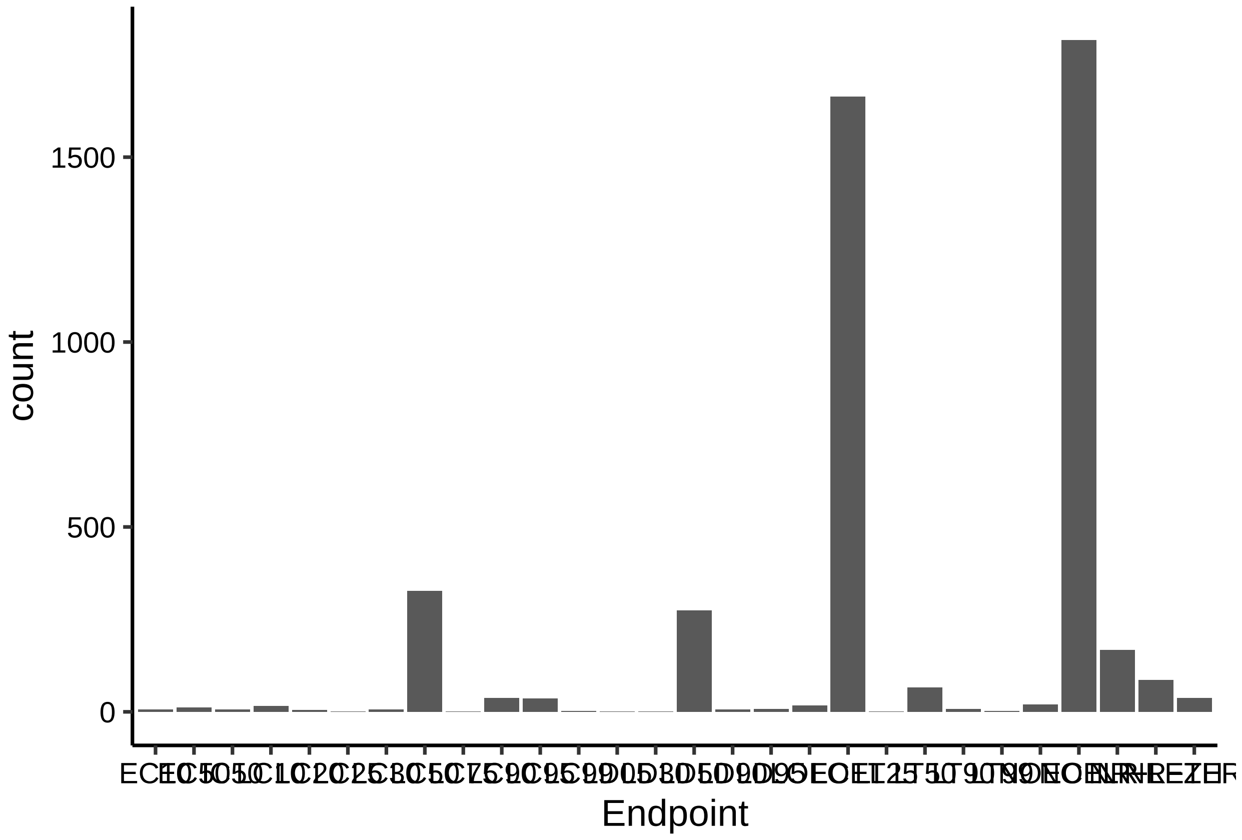
For example: `ggsave("./Output/PMplot.jpg", PMplot.faceted, height = 4, width = 6, units = "in", dpi = 300)`

Visualization challenge

The following graph displays the counts of specific endpoints measured in neonicotinoid ecotoxicology studies. The way it is visualized, however, is not effective. Make the following coding changes to improve the graph:

1. Change the ordering of the "Endpoint" factor (function: `reorder`) so that the highest counts are listed first (hint: FUN = length)
2. Plot the barplot with the reordered factor levels. Add this line of code to make the bars show up left to right: scale_x_discrete(limits = rev(levels(Neonics$Endpoint)))
3. Adjust the x axis labels so they appear at a 45 degree angle.
4. Change the color and/or border on the bars. Should you have a consistent color across all bars, or a different color for each bar?

```r
Neonics <- read.csv("./Data/Raw/ECOTOX_Neonicotinoids_Insects_raw.csv")
ggplot(Neonics) +
  geom_bar(aes(x = Endpoint))
```


count
Endpoint