

5: Part 2 - Data Visualization Advanced

Environmental Data Analytics | John Fay and Luana Lima | Developed by Kateri Salk

Fall 2022

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] "/home/guest/R/EDA-Fall2022"
```

```
library(tidyverse)
```

```
PeterPaul.chem.nutrients <- read.csv("./Data/Processed/NTL-LTER_Lake_Chemistry_Nutrients_PeterPaul_Proc  
  stringsAsFactors = T)
```

```
PeterPaul.chem.nutrients.gathered <- read.csv("./Data/Processed/NTL-LTER_Lake_Nutrients_PeterPaulGather  
  stringsAsFactors = T)
```

```
EPAair <- read.csv("./Data/Processed/EPAair_03_PM25_NC1819_Processed.csv", stringsAsFactors = T)
```

```
EPAair$Date <- as.Date(EPAair$Date, format = "%Y-%m-%d")
```

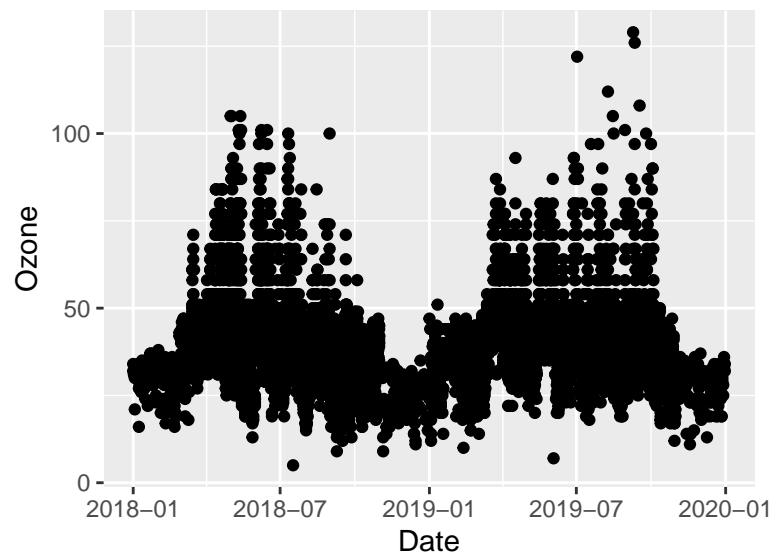
```
PeterPaul.chem.nutrients$sampldate <- as.Date(PeterPaul.chem.nutrients$sampldate,  
  format = "%Y-%m-%d")
```

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

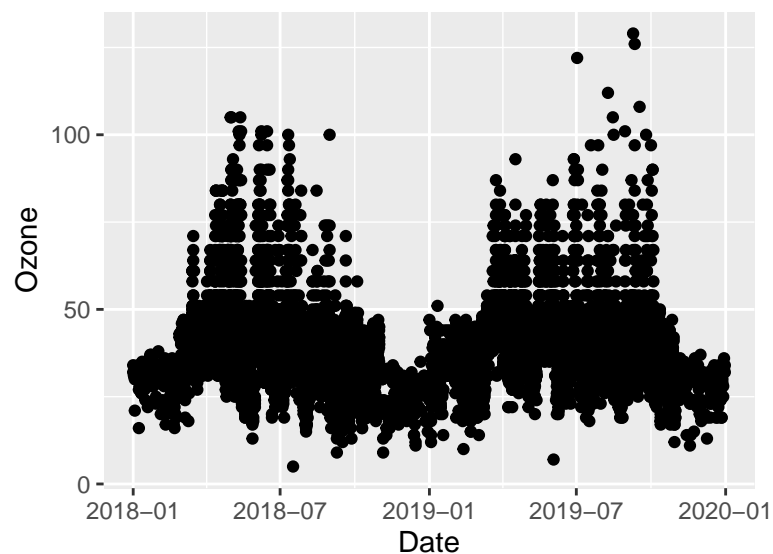
Themes

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. where legends should go set these for graphs

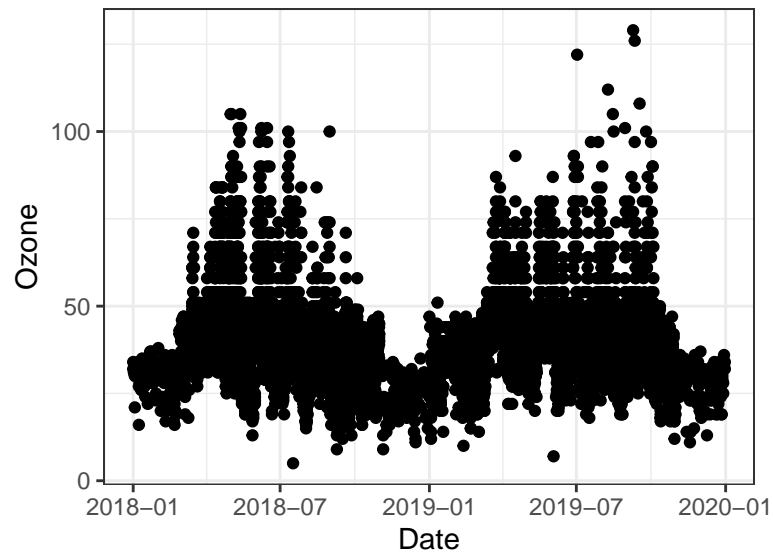
```
O3plot <- ggplot(EPAair) + geom_point(aes(x = Date, y = Ozone))  
print(O3plot)
```



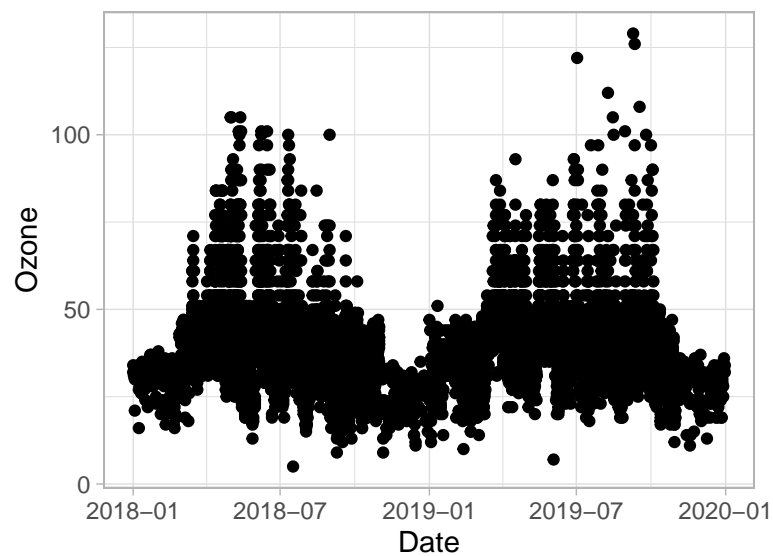
```
O3plot1 <- ggplot(EPAair) + geom_point(aes(x = Date, y = Ozone)) + theme_gray() #default ggi plot them
print(O3plot1)
```



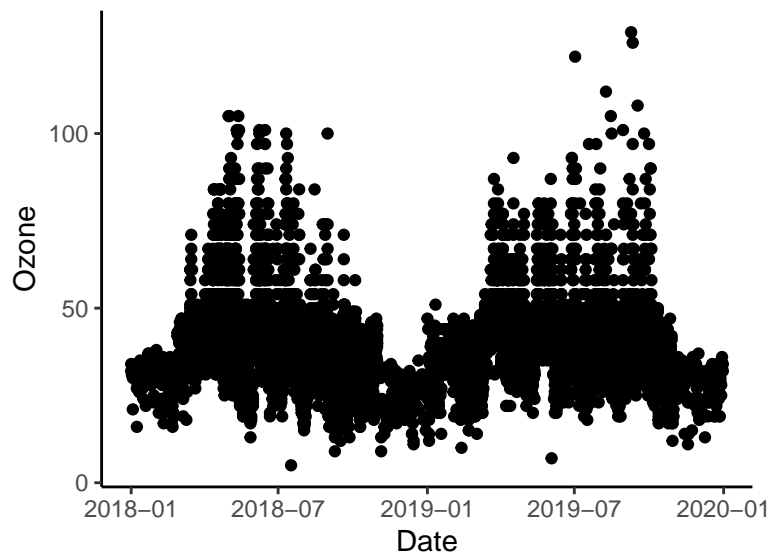
```
# theme bw - is dark on white theme- background white
O3plot2 <- ggplot(EPAair) + geom_point(aes(x = Date, y = Ozone)) + theme_bw()
print(O3plot2)
```



```
03plot3 <- ggplot(EPAair) + geom_point(aes(x = Date, y = Ozone)) + theme_light() #background is white
print(03plot3)
```



```
03plot4 <- ggplot(EPAair) + geom_point(aes(x = Date, y = Ozone)) + theme_classic() #no gridlines x and y
print(03plot4)
```



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

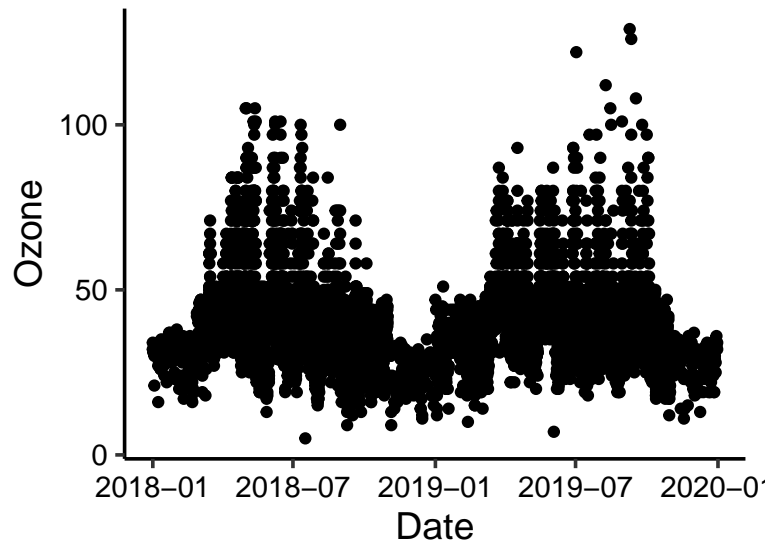
- text size
- axis label colors
- legend position and justification

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) + #define theme - call you theme -
  theme(axis.text = element_text(color = "black"), #change axis text and legend position top
        legend.position = "top") #alternative: legend.position + legend.justification

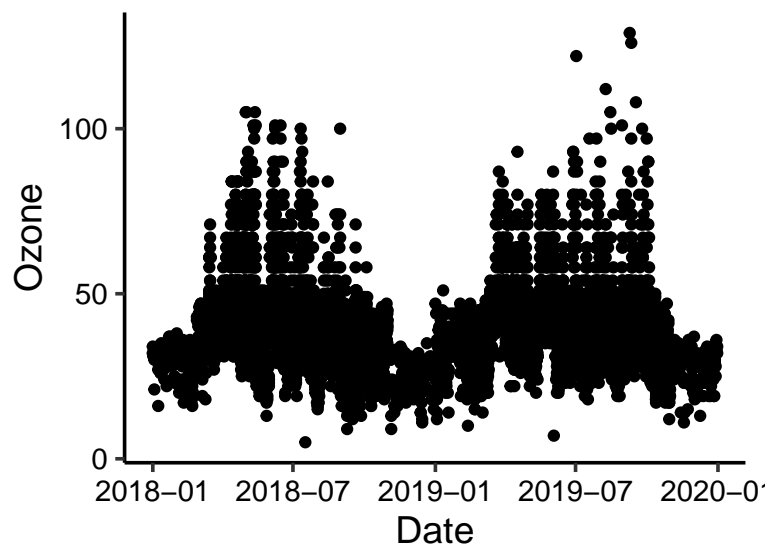
# options: call the theme in each plot or set the theme at the start.

O3plot5 <- ggplot(EPAair) +
  geom_point(aes(x = Date, y = Ozone)) +
  mytheme #setting theme using plot environment from code above
print(O3plot5)
```



```
theme_set(mytheme) #set theme for all subsequent plots

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



Adjusting multiple components of your plots

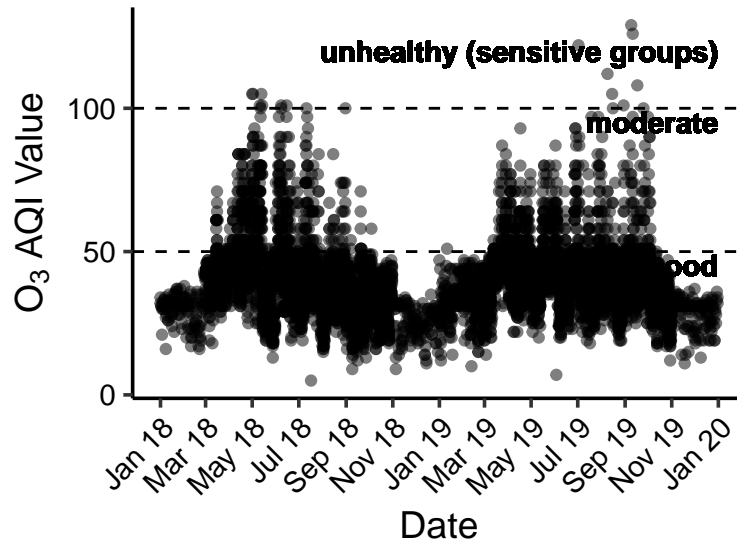
While the theme allows us to set multiple aspects of plots, ggplot allows us to adjust other parts of plots outside of the theme.

```
O3plot7 <- ggplot(EPAair, aes(x = Date, y = Ozone)) +
  geom_hline(yintercept = 50, lty = 2) + #lty specify type of line - dashed line
  geom_hline(yintercept = 100, lty = 2) + #another horizontal line at point 100
```

```

geom_point(alpha = 0.5, size = 1.5) + #add points representing data set - alpha determine transparency
geom_text(x = as.Date("2020-01-01"), y = 45, label = "good", hjust = 1, fontface = "bold") + # hjust
geom_text(x = as.Date("2020-01-01"), y = 95, label = "moderate", hjust = 1, fontface = "bold") + #mod
geom_text(x = as.Date("2020-01-01"), y = 120, label = "unhealthy (sensitive groups)", hjust = 1, fontface = "bold") + #unhealthy
scale_x_date(limits = as.Date(c("2018-01-01", "2019-12-31")), #change scale on x axis
  date_breaks = "2 months", date_labels = "%b %y") + #month labels number of didgets that appear
ylab(expression("O3"[3]* " AQI Value")) +
theme(axis.text.x = element_text(angle = 45, hjust = 1)) #so text on axis does not overlap
print(O3plot7)

```



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)

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

viridis and viridisLite (packages)

- <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)

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

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

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

```
library(RColorBrewer)
library(colormap)

# looks at color maps and different shades
scales::show_col(colormap(colormap = colormaps$viridis, nshades = 16))
```

#440154ff	#461868ff	#472d7bff	#404284ff
#39558bff	#31668dff	#2a768eff	#24888dff
#23978aff	#26a784ff	#37b578ff	#55c467ff
#79d051ff	#a3da37ff	#cee12cff	#fde725ff

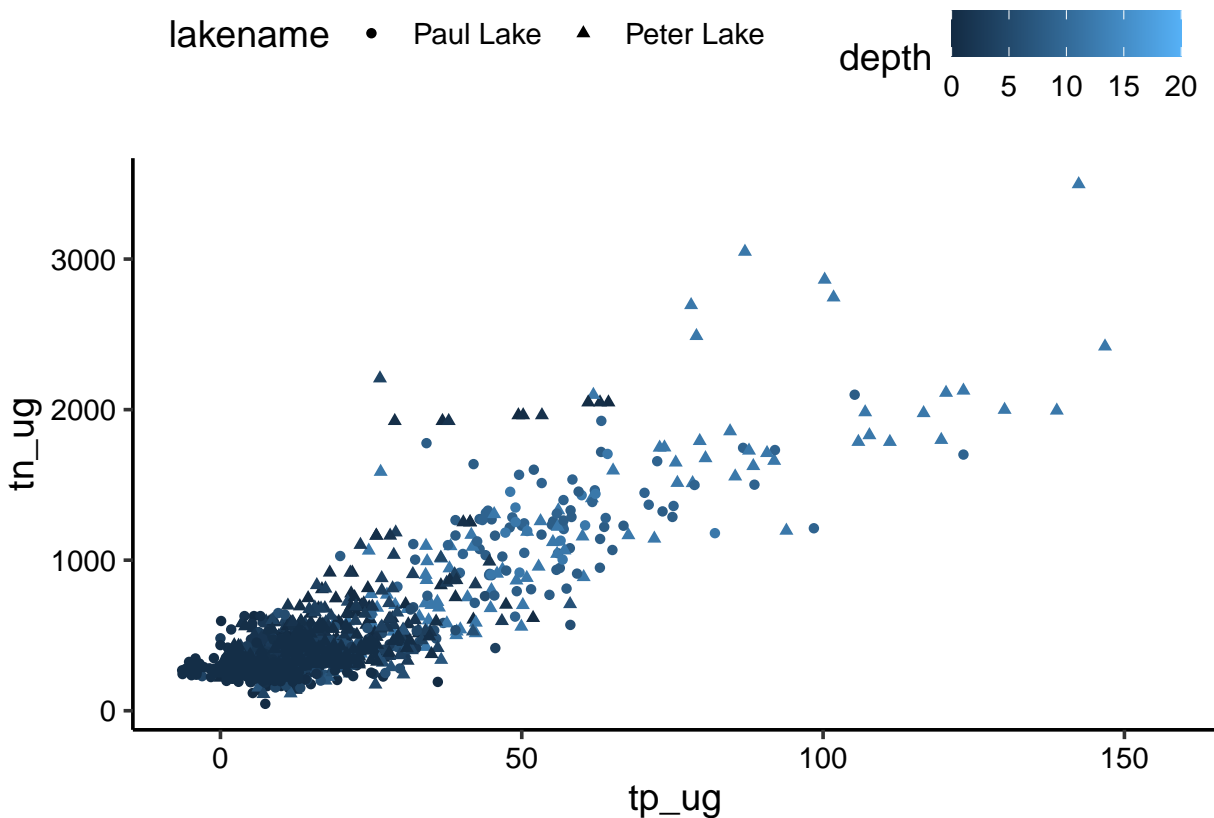
```
scales::show_col(colormap(colormap = colormaps$inferno, nshades = 16))
```


#000004ff	#100628ff	#210c49ff	#3f0e5eff
#5b116dff	#761b6bff	#902567ff	#ad315bff
#c43f4dff	#da513aff	#ea6827ff	#f6850fff
#f9a319ff	#f9c32eff	#fae063ff	#fcffa4ff

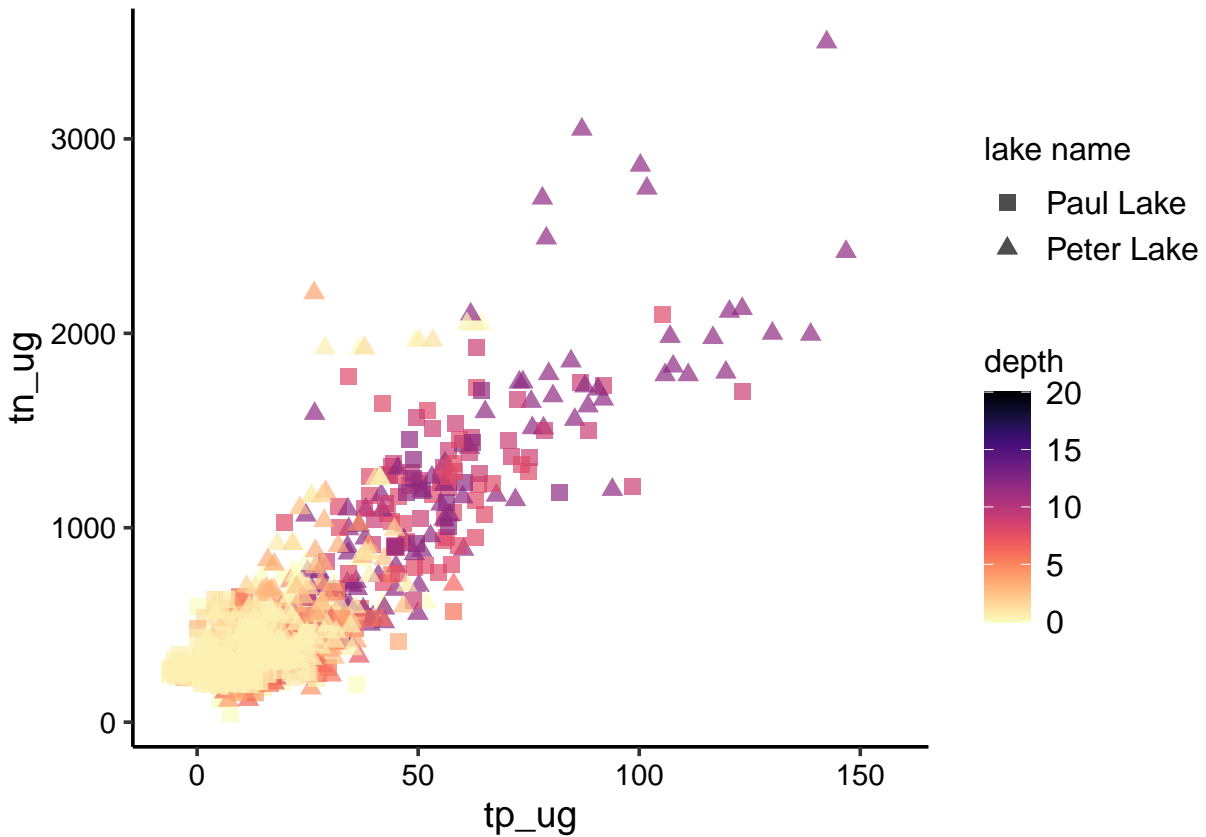
```
scales::show_col(colormap(colormap = colormaps$magma, nshades = 16))
```

#000004ff	#0f0926ff	#1e1046ff	#3b1165ff
#55147cff	#701e7fff	#8a2880ff	#a7317cff
#c13d75ff	#db4a69ff	#ec6163ff	#f88061ff
#fc9d6fff	#febcb83ff	#fddc9fff	#fcfdbfff

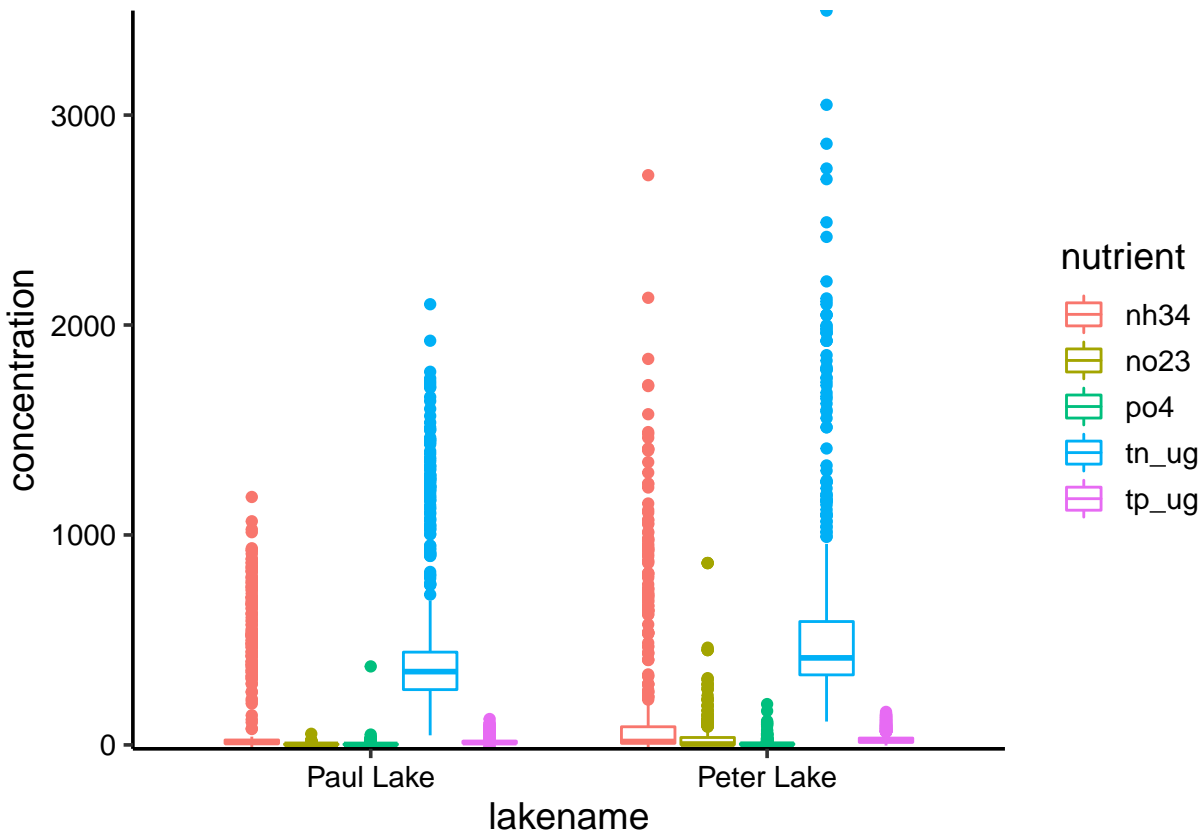
```
display.brewer.all(n = 9) #display all colors
```

```
# 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) + #add some transparency to the points
  labs(color="depth", shape="lake name") + # change your legend labels here
  scale_shape_manual(values = c(15, 17)) +
  #scale_color_distiller(palette = "Blues", direction = 1) + # use scale_color_brewer for discrete vari
  scale_color_viridis(option = "magma", direction = -1) +
  theme(legend.position = "right",
        legend.text = element_text(size = 12), legend.title = element_text(size = 12))
print(NvsP2)
```



```
# 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)
Nutrientplot <-
  ggplot(PeterPaul.chem.nutrients.gathered, aes(x = lakename, y = concentration, color = nutrient)) +
  geom_boxplot() +
  # place your additional edits here
  scale_y_continuous(expand = c(0, 0)) +
  #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")
print(Nutrientplot)
```



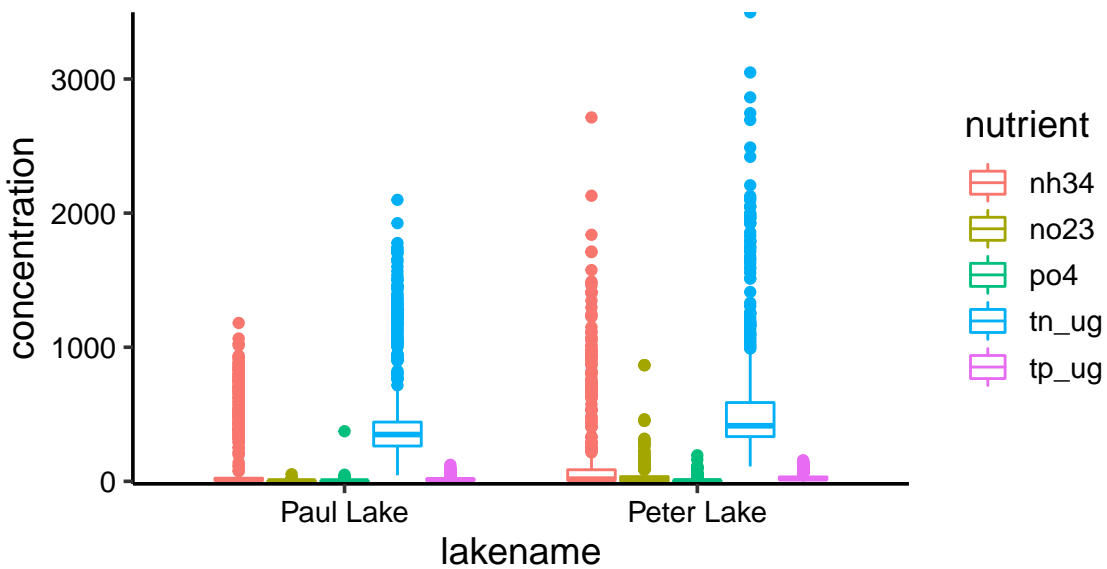
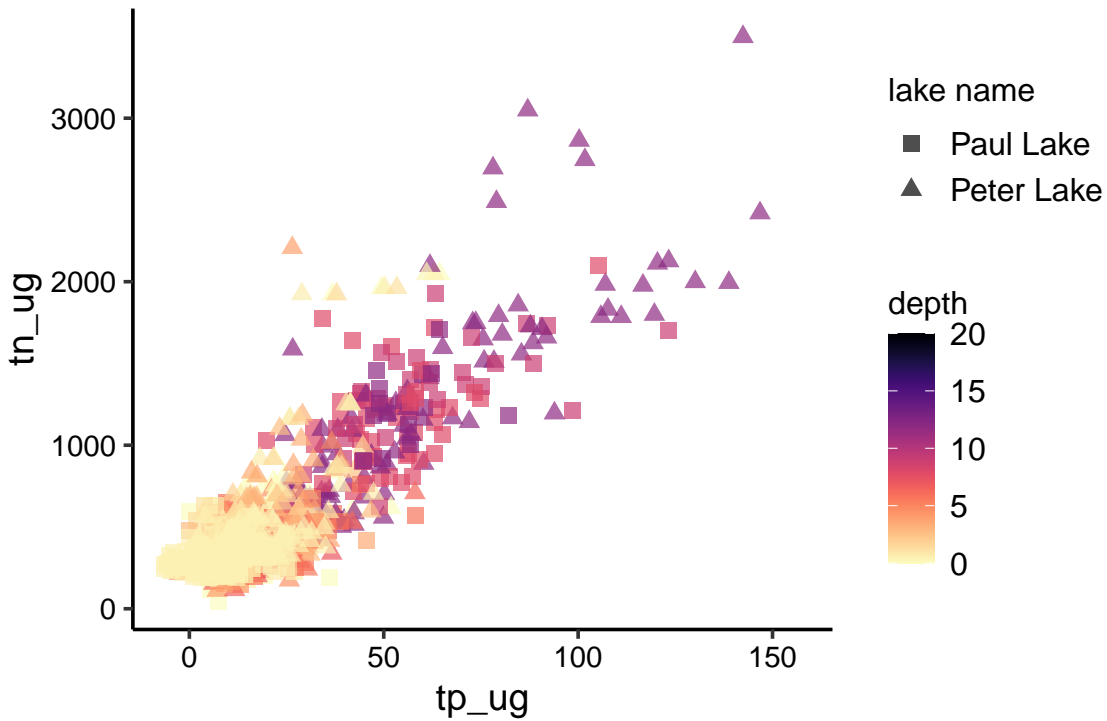
Multiple plots on a page

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

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



*# plot grid allows you to have multiple plots in same window. specify how many
rows with how many plots. 2 plots rel_height adjusts space of plots have in
window- one plot may need more space*

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?

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

