Boxplots are often used to show data distributions, and ggplot2 is often used to visualize data. A question that comes up is what exactly do the box plots represent? The ggplot2 box plots follow standard Tukey representations, and there are many references of this online and in standard statistical text books. The base R function to calculate the box plot limits is boxplot.stats. The help file for this function is very informative, but it’s often non-R users asking what exactly the plot means. Therefore, this blog post breaks down the calculations into (hopefully!) easy-to-follow chunks of code for you to make your own box plot legend if necessary. Some additional goals here are to create boxplots that come *close* to USGS style. Features in this blog post take advantage of enhancements to ggplot2 in version 3.0.0 or later.

First, let’s get some data that might be typically plotted in a USGS report using a boxplot. Here we’ll use chloride data (parameter code “00940”) measured at a USGS station on the Fox River in Green Bay, WI (station ID “04085139”). We’ll use the package dataRetrieval to get the data (see [this tutorial](https://owi.usgs.gov/R/dataRetrieval.html) for more information on dataRetrieval), and plot a simple boxplot by month using ggplot2:

library(dataRetrieval)

library(ggplot2)

# Get chloride data using dataRetrieval:

chloride <- readNWISqw("04085139", "00940")

# Add a month column:

chloride$month <- month.abb[as.numeric(format(chloride$sample\_dt, "%m"))]

chloride$month <- factor(chloride$month, labels = month.abb)

# Pull out the official parameter and site names for labels:

cl\_name <- attr(chloride, "variableInfo")[["parameter\_nm"]]

cl\_site <- attr(chloride, "siteInfo")[["station\_nm"]]

# Create basic ggplot graph:

ggplot(data = chloride,

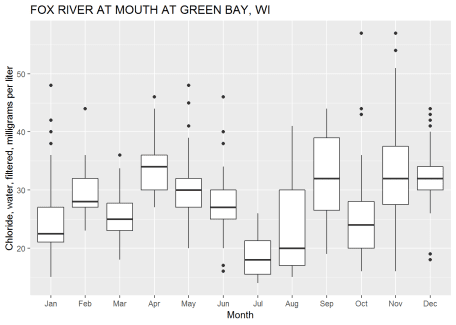
aes(x = month, y = result\_va)) +

geom\_boxplot() +

xlab("Month") +

ylab(cl\_name) +

labs(title = cl\_site)



Is that graph great? YES! And for presentations and/or journal publications, that graph might be appropriate. However, for an official USGS report, USGS employees need to get the graphics approved to assure they follow specific style guidelines. The approving officer would probably come back from the review with the following comments:

| **Reviewer’s Comments** | **ggplot2 element to modify** |
| --- | --- |
| Remove background color, grid lines | Adjust theme |
| Add horizontal bars to the upper and lower whiskers | Add stat\_boxplot |
| Have tick marks go inside the plot | Adjust theme |
| Tick marks should be on both sides of the y axis | Add sec.axis to scale\_y\_continuous |
| Remove tick marks from discrete data | Adjust theme |
| y-axis needs to start exactly at 0 | Add expand\_limits |
| y-axis labels need to be shown at 0 and at the upper scale | Add breaks and limits to scale\_y\_continuous |
| Add very specific legend | Create function ggplot\_box\_legend |
| Add the number of observations above each boxplot | Add custom stat\_summary |
| Change text size | Adjust geom\_text defaults |
| Change font (we’ll use "serif" in this blog, although that is not the official USGS font) | Adjust geom\_text defaults |

As you can see, it will not be as simple as creating a single custom ggplot theme to comply with the requirements. However, we can string together ggplot commands in a list for easy re-use. This blog is *not* going to get you perfect compliance with the USGS standards, but it will get much closer. Also, while these style adjustments are tailored to USGS requirements, the process described here may be useful for other graphic guidelines as well.

So, let’s skip to the exciting conclusion and use some code that will be described later (boxplot\_framework and ggplot\_box\_legend) to create the same plot, now closer to those USGS style requirements:

library(cowplot)

# NOTE! This is a preview of the FUTURE!

# We'll create the functions ggplot\_box\_legend and boxplot\_framework

# later in this blog post.

# So....by the end of this post, you will be able to:

legend\_plot <- ggplot\_box\_legend()

chloride\_plot <- ggplot(data = chloride,

aes(x = month, y = result\_va)) +

boxplot\_framework(upper\_limit = 70) +

xlab(label = "Month") +

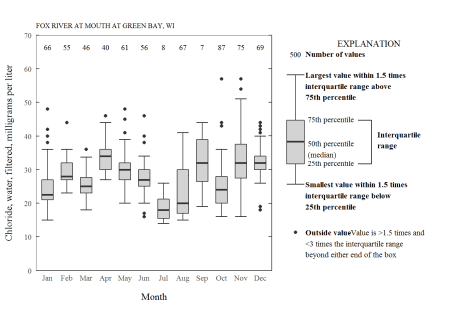
ylab(label = cl\_name) +

labs(title = cl\_site)

plot\_grid(chloride\_plot,

legend\_plot,

nrow = 1, rel\_widths = c(.6,.4))



As can be seen in the code chunk, we are now using a function ggplot\_box\_legend to make a legend, boxplot\_framework to accommodate all of the style requirements, and the cowplot package to plot them together.

**boxplot\_framework: Styling the boxplot**

Let’s get our style requirements figured out. First, we can set some basic plot elements for a theme. We can start with the theme\_bw and add to that. Here we remove the grid, set the size of the title, bring the y-ticks inside the plotting area, and remove the x-ticks:

theme\_USGS\_box <- function(base\_family = "serif", ...){

theme\_bw(base\_family = base\_family, ...) +

theme(

panel.grid = element\_blank(),

plot.title = element\_text(size = 8),

axis.ticks.length = unit(-0.05, "in"),

axis.text.y = element\_text(margin=unit(c(0.3,0.3,0.3,0.3), "cm")),

axis.text.x = element\_text(margin=unit(c(0.3,0.3,0.3,0.3), "cm")),

axis.ticks.x = element\_blank(),

aspect.ratio = 1,

legend.background = element\_rect(color = "black", fill = "white")

)

}

Next, we can change the defaults of the geom\_text to a smaller size and font.

update\_geom\_defaults("text",

list(size = 3,

family = "serif"))

We also need to figure out what other ggplot2 functions need to be added. The basic ggplot code for the chloride plot would be:

n\_fun <- function(x){

return(data.frame(y = 0.95\*70,

label = length(x)))

}

ggplot(data = chloride,

aes(x = month, y = result\_va)) +

stat\_boxplot(geom ='errorbar', width = 0.6) +

geom\_boxplot(width = 0.6, fill = "lightgrey") +

stat\_summary(fun.data = n\_fun, geom = "text", hjust = 0.5) +

expand\_limits(y = 0) +

theme\_USGS\_box() +

scale\_y\_continuous(sec.axis = dup\_axis(label = NULL,

name = NULL),

expand = expand\_scale(mult = c(0, 0)),

breaks = pretty(c(0,70), n = 5),

limits = c(0,70))

Breaking that code down:

| **Function** | **What’s happening?** |
| --- | --- |
| stat\_boxplot(geom =’errorbar’) | The "errorbars" are used to make the horizontal lines on the upper and lower whiskers. This needs to happen first so it is in the back of the plot. |
| geom\_boxplot | Regular boxplot |
| stat\_summary(fun.data = n\_fun, geom = "text", hjust = 0.5) | The stat\_summary function is very powerful for adding specific summary statistics to the plot. In this case, we are adding a geom\_text that is calculated with our custom n\_fun. That function comes back with the count of the boxplot, and puts it at 95% of the hard-coded upper limit. |
| expand\_limits | This forces the plot to include 0. |
| theme\_USGS\_box | Theme created above to help with grid lines, tick marks, axis size/fonts, etc. |
| scale\_y\_continuous | A tricky part of the USGS requirements involve 4 parts: Add ticks to the right side, have at least 4 "pretty" labels on the left axis, remove padding, and have the labels start and end at the beginning and end of the plot. Breaking that down further: |
| scale\_y\_continuous(sec.axis = dup\_axis | Handy function to add tick marks to the right side of the graph. |
| scale\_y\_continuous(expand = expand\_scale(mult = c(0, 0)) | Remove padding |
| scale\_y\_continuous(breaks = pretty(c(0,70), n = 5)) | Make pretty label breaks, assuring 5 pretty labels if the graph went from 0 to 70 |
| scale\_y\_continuous(limits = c(0,70)) | Assure the graph goes from 0 to 70. |

Let’s look at a few other common boxplots to see if there are other ggplot2 elements that would be useful in a common boxplot\_framework function.

**Logrithmic boxplot**

For another example, we might need to make a boxplot with a logarithm scale. This data is for phosphorus measurements on the Pheasant Branch Creek in Middleton, WI.

site <- "05427948"

pCode <- "00665"

# Get phosphorus data using dataRetrieval:

phos\_data <- readNWISqw(site, pCode)

# Create a month column:

phos\_data$month <- month.abb[as.numeric(format(phos\_data$sample\_dt, "%m"))]

phos\_data$month <- factor(phos\_data$month, labels = month.abb)

# Get site name and paramter name for labels:

phos\_name <- attr(phos\_data, "variableInfo")[["parameter\_nm"]]

phos\_site <- attr(phos\_data, "siteInfo")[["station\_nm"]]

n\_fun <- function(x){

return(data.frame(y = 0.95\*log10(50),

label = length(x)))

}

prettyLogs <- function(x){

pretty\_range <- range(x[x > 0])

pretty\_logs <- 10^(-10:10)

log\_index <- which(pretty\_logs < pretty\_range[2] &

pretty\_logs > pretty\_range[1])

log\_index <- c(log\_index[1]-1,log\_index, log\_index[length(log\_index)]+1)

pretty\_logs\_new <- pretty\_logs[log\_index]

return(pretty\_logs\_new)

}

fancyNumbers <- function(n){

nNoNA <- n[!is.na(n)]

x <-gsub(pattern = "1e",replacement = "10^",

x = format(nNoNA, scientific = TRUE))

exponents <- as.numeric(sapply(strsplit(x, "\\^"), function(j) j[2]))

base <- ifelse(exponents == 0, "1", ifelse(exponents == 1, "10","10^"))

exponents[base == "1" | base == "10"] <- ""

textNums <- rep(NA, length(n))

textNums[!is.na(n)] <- paste0(base,exponents)

textReturn <- parse(text=textNums)

return(textReturn)

}

phos\_plot <- ggplot(data = phos\_data,

aes(x = month, y = result\_va)) +

stat\_boxplot(geom ='errorbar', width = 0.6) +

geom\_boxplot(width = 0.6, fill = "lightgrey") +

stat\_summary(fun.data = n\_fun, geom = "text", hjust = 0.5) +

theme\_USGS\_box() +

scale\_y\_log10(limits = c(0.001, 50),

expand = expand\_scale(mult = c(0, 0)),

labels=fancyNumbers,

breaks=prettyLogs) +

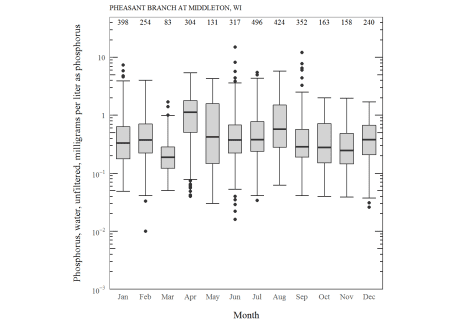
annotation\_logticks(sides = c("rl")) +

xlab("Month") +

ylab(phos\_name) +

labs(title = phos\_site)

phos\_plot



What are the new features we have to consider for log scales?

| **Function** | **What’s happening?** |
| --- | --- |
| stat\_boxplot | The stat\_boxplot function is the same, but our custom function to calculate counts need to be adjusted so the position would be in log units. |
| scale\_y\_log10 | This is used instead of scale\_y\_continuous. |
| annotation\_logticks(sides = c("rl")) | Adds nice log ticks to the right ("r") and left ("l") side. |
| prettyLogs | This function forces the y-axis breaks to be on every 10^x. This could be adjusted if a finer scale was needed. |
| fancyNumbers | This is a custom formatting function for the log axis. This function could be adjusted if other formatting was needed. |

**Grouped boxplots**

We might also want to make grouped boxplots. In ggplot, it’s pretty easy to add a “fill” to the aes argument. Here we’ll plot temperature distributions at 4 USGS stations. We’ll group the measurements by a “daytime” and “nighttime” factor. Temperature might be a parameter that would not be required to start at 0.

library(dplyr)

# Get water temperature data for a variety of USGS stations

temp\_q\_data <- readNWISuv(siteNumbers = c("04026561", "04063700",

"04082400", "05427927"),

parameterCd = '00010',

startDate = "2018-06-01",

endDate = "2018-06-03")

temperature\_name <- attr(temp\_q\_data, "variableInfo")[["variableName"]]

# add an hour of day to create groups (daytime or nighttime)

temp\_q\_data <- temp\_q\_data %>%

renameNWISColumns() %>%

mutate(hourOfDay = as.numeric(format(dateTime, "%H")),

timeOfDay = case\_when(hourOfDay < 20 & hourOfDay > 6 ~ "daytime",

TRUE ~ "nighttime" # catchall

))

n\_fun <- function(x){

return(data.frame(y = 0.95\*30,

label = length(x)))

}

temperature\_plot <- ggplot(data = temp\_q\_data,

aes(x=site\_no, y=Wtemp\_Inst, fill=timeOfDay)) +

stat\_boxplot(geom ='errorbar', width = 0.6) +

geom\_boxplot(width = 0.6) +

stat\_summary(fun.data = n\_fun, geom = "text",

aes(group=timeOfDay),

hjust = 0.5, position = position\_dodge(0.6)) +

expand\_limits(y = 0) +

scale\_y\_continuous(sec.axis = dup\_axis(label = NULL,

name = NULL),

expand = expand\_scale(mult = c(0, 0)),

breaks = pretty(c(10,30), n = 5),

limits = c(10,30)) +

theme\_USGS\_box() +

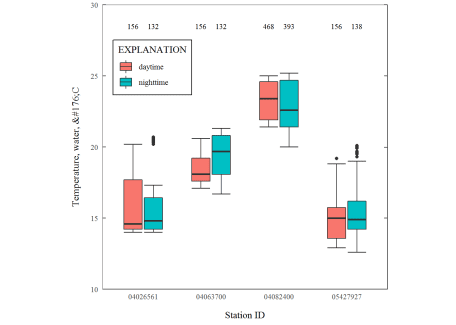
xlab("Station ID") +

ylab(temperature\_name) +

scale\_fill\_discrete(name = "EXPLANATION") +

theme(legend.position = c(0.175, 0.78))

temperature\_plot



What are the new features we have to consider for log scales?

| **Function** | **What’s happening?** |
| --- | --- |
| stat\_summary(position) | We need to move the counts to above the boxplots. This is done by shifting them the same amount as the width. |
| stat\_summary(aes(group=timeOfDay)) | We need to include how the boxplots are grouped. |
| scale\_fill\_discrete | Need include a fill legend. |

Additionally, the parameter name that comes back from dataRetrieval could use some formatting. The following function can fix that for both ggplot2 and base R graphics:

unescape\_html <- function(str){

fancy\_chars <- regmatches(str, gregexpr("&#\\d{3};",str))

unescaped <- xml2::xml\_text(xml2::read\_html(paste0("", fancy\_chars, "")))

fancy\_chars <- gsub(pattern = "&#\\d{3};",

replacement = unescaped, x = str)

fancy\_chars <- gsub("Â","", fancy\_chars)

return(fancy\_chars)

}

We’ll use this function in the next section.

**Framework function**

Finally, we can bring all of those elements together into a single list for ggplot2 to use. While we’re at it, we can create a function that is flexible for both linear and logarithmic scales, as well as grouped boxplots. It’s a bit clunky because you need to specify the upper and lower limits of the plot. Much of the USGS style requirements depend on specific upper and lower limits, so I decided this was an acceptable solution for this blog post. There’s almost certainly a slicker way to do that, but for now, it works:

boxplot\_framework <- function(upper\_limit,

family\_font = "serif",

lower\_limit = 0,

logY = FALSE,

fill\_var = NA,

fill = "lightgrey", width = 0.6){

update\_geom\_defaults("text",

list(size = 3,

family = family\_font))

n\_fun <- function(x, lY = logY){

return(data.frame(y = ifelse(logY, 0.95\*log10(upper\_limit), 0.95\*upper\_limit),

label = length(x)))

}

prettyLogs <- function(x){

pretty\_range <- range(x[x > 0])

pretty\_logs <- 10^(-10:10)

log\_index <- which(pretty\_logs < pretty\_range[2] &

pretty\_logs > pretty\_range[1])

log\_index <- c(log\_index[1]-1,log\_index,

log\_index[length(log\_index)]+1)

pretty\_logs\_new <- pretty\_logs[log\_index]

return(pretty\_logs\_new)

}

fancyNumbers <- function(n){

nNoNA <- n[!is.na(n)]

x <-gsub(pattern = "1e",replacement = "10^",

x = format(nNoNA, scientific = TRUE))

exponents <- as.numeric(sapply(strsplit(x, "\\^"), function(j) j[2]))

base <- ifelse(exponents == 0, "1", ifelse(exponents == 1, "10","10^"))

exponents[base == "1" | base == "10"] <- ""

textNums <- rep(NA, length(n))

textNums[!is.na(n)] <- paste0(base,exponents)

textReturn <- parse(text=textNums)

return(textReturn)

}

if(!is.na(fill\_var)){

basic\_elements <- list(stat\_boxplot(geom ='errorbar', width = width),

geom\_boxplot(width = width),

stat\_summary(fun.data = n\_fun,

geom = "text",

position = position\_dodge(width),

hjust =0.5,

aes\_string(group=fill\_var)),

expand\_limits(y = lower\_limit),

theme\_USGS\_box())

} else {

basic\_elements <- list(stat\_boxplot(geom ='errorbar', width = width),

geom\_boxplot(width = width, fill = fill),

stat\_summary(fun.data = n\_fun,

geom = "text", hjust =0.5),

expand\_limits(y = lower\_limit),

theme\_USGS\_box())

}

if(logY){

return(c(basic\_elements,

scale\_y\_log10(limits = c(lower\_limit, upper\_limit),

expand = expand\_scale(mult = c(0, 0)),

labels=fancyNumbers,

breaks=prettyLogs),

annotation\_logticks(sides = c("rl"))))

} else {

return(c(basic\_elements,

scale\_y\_continuous(sec.axis = dup\_axis(label = NULL,

name = NULL),

expand = expand\_scale(mult = c(0, 0)),

breaks = pretty(c(lower\_limit,upper\_limit), n = 5),

limits = c(lower\_limit,upper\_limit))))

}

}

**Examples with our framework**

Let’s see if it works! Let’s build the last set of example figures using our new function boxplot\_framework. I’m also going to use the ‘cowplot’ package to print them all together. I’ll also include the ggplot\_box\_legend which will be described in the next section.

legend\_plot <- ggplot\_box\_legend()

chloride\_plot <- ggplot(data = chloride,

aes(x = month, y = result\_va)) +

boxplot\_framework(upper\_limit = 70) +

xlab(label = "Month") +

ylab(label = cl\_name) +

labs(title = cl\_site)

phos\_plot <- ggplot(data = phos\_data,

aes(x = month, y = result\_va)) +

boxplot\_framework(upper\_limit = 50,

lower\_limit = 0.001,

logY = TRUE) +

xlab(label = "Month") +

#Shortened label since the graph area is smaller

ylab(label = "Phosphorus in milligraphs per liter") +

labs(title = phos\_site)

temperature\_plot <- ggplot(data = temp\_q\_data,

aes(x=site\_no, y=Wtemp\_Inst, fill=timeOfDay)) +

boxplot\_framework(upper\_limit = 30,

lower\_limit = 10,

fill\_var = "timeOfDay") +

xlab(label = "Station ID") +

ylab(label = unescape\_html(temperature\_name)) +

labs(title = "Daytime vs Nighttime Temperature Distribution") +

scale\_fill\_discrete(name = "EXPLANATION") +

theme(legend.position = c(0.225, 0.72),

legend.title = element\_text(size = 7))

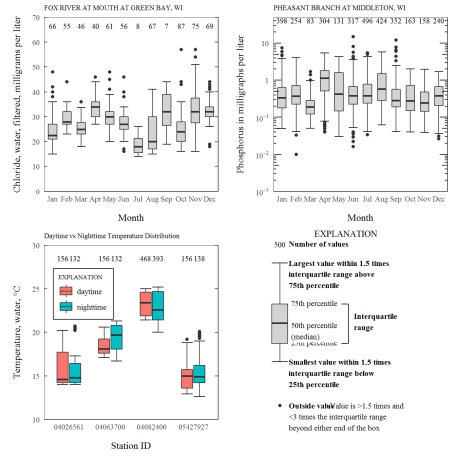
plot\_grid(chloride\_plot,

phos\_plot,

temperature\_plot,

legend\_plot,

nrow = 2)



**ggplot\_box\_legend: What is a boxplot?**

A non-trivial requirement to the USGS boxplot style guidelines is to make a detailed, prescribed legend. In this section we’ll first verify that ggplot2 boxplots use the same definitions for the lines and dots, and then we’ll make a function that creates the prescribed legend. To start, let’s set up random data using the R function sample and then create a function to calculate each value.

set.seed(100)

sample\_df <- data.frame(parameter = "test",

values = sample(500))

# Extend the top whisker a bit:

sample\_df$values[1:100] <- 701:800

# Make sure there's only 1 lower outlier:

sample\_df$values[1] <- -350

Next, we’ll create a function that calculates the necessary values for the boxplots:

ggplot2\_boxplot <- function(x){

quartiles <- as.numeric(quantile(x,

probs = c(0.25, 0.5, 0.75)))

names(quartiles) <- c("25th percentile",

"50th percentile\n(median)",

"75th percentile")

IQR <- diff(quartiles[c(1,3)])

upper\_whisker <- max(x[x < (quartiles[3] + 1.5 \* IQR)])

lower\_whisker <- min(x[x > (quartiles[1] - 1.5 \* IQR)])

upper\_dots <- x[x > (quartiles[3] + 1.5\*IQR)]

lower\_dots <- x[x < (quartiles[1] - 1.5\*IQR)]

return(list("quartiles" = quartiles,

"25th percentile" = as.numeric(quartiles[1]),

"50th percentile\n(median)" = as.numeric(quartiles[2]),

"75th percentile" = as.numeric(quartiles[3]),

"IQR" = IQR,

"upper\_whisker" = upper\_whisker,

"lower\_whisker" = lower\_whisker,

"upper\_dots" = upper\_dots,

"lower\_dots" = lower\_dots))

}

ggplot\_output <- ggplot2\_boxplot(sample\_df$values)

What are those calculations?

* Quartiles (25, 50, 75 percentiles), 50% is the median
* Interquartile range is the difference between the 75th and 25th percentiles
* The upper whisker is the maximum value of the data that is within 1.5 times the interquartile range over the 75th percentile.
* The lower whisker is the minimum value of the data that is within 1.5 times the interquartile range under the 25th percentile.
* Outlier values are considered any values over 1.5 times the interquartile range over the 75th percentile or any values under 1.5 times the interquartile range under the 25th percentile.

Let’s check that the output matches boxplot.stats:

# Using base R:

base\_R\_output <- boxplot.stats(sample\_df$values)

# Some checks:

# Outliers:

all(c(ggplot\_output[["upper\_dots"]],

ggplot\_output[["lowerdots"]]) %in%

c(base\_R\_output[["out"]]))

## [1] TRUE

# whiskers:

ggplot\_output[["upper\_whisker"]] == base\_R\_output[["stats"]][5]

## [1] TRUE

ggplot\_output[["lower\_whisker"]] == base\_R\_output[["stats"]][1]

## [1] TRUE

**Boxplot Legend**

Let’s use this information to generate a legend, and make the code reusable by creating a standalone function that we used in earlier code (ggplot\_box\_legend). There is a *lot* of ggplot2 code to digest here. Most of it is style adjustments to approximate the USGS style guidelines for a boxplot legend.

Show/Hide Code

ggplot\_box\_legend <- function(family = "serif"){

# Create data to use in the boxplot legend:

set.seed(100)

sample\_df <- data.frame(parameter = "test",

values = sample(500))

# Extend the top whisker a bit:

sample\_df$values[1:100] <- 701:800

# Make sure there's only 1 lower outlier:

sample\_df$values[1] <- -350

# Function to calculate important values:

ggplot2\_boxplot <- function(x){

quartiles <- as.numeric(quantile(x,

probs = c(0.25, 0.5, 0.75)))

names(quartiles) <- c("25th percentile",

"50th percentile\n(median)",

"75th percentile")

IQR <- diff(quartiles[c(1,3)])

upper\_whisker <- max(x[x < (quartiles[3] + 1.5 \* IQR)])

lower\_whisker <- min(x[x > (quartiles[1] - 1.5 \* IQR)])

upper\_dots <- x[x > (quartiles[3] + 1.5\*IQR)]

lower\_dots <- x[x < (quartiles[1] - 1.5\*IQR)]

return(list("quartiles" = quartiles,

"25th percentile" = as.numeric(quartiles[1]),

"50th percentile\n(median)" = as.numeric(quartiles[2]),

"75th percentile" = as.numeric(quartiles[3]),

"IQR" = IQR,

"upper\_whisker" = upper\_whisker,

"lower\_whisker" = lower\_whisker,

"upper\_dots" = upper\_dots,

"lower\_dots" = lower\_dots))

}

# Get those values:

ggplot\_output <- ggplot2\_boxplot(sample\_df$values)

# Lots of text in the legend, make it smaller and consistent font:

update\_geom\_defaults("text",

list(size = 3,

hjust = 0,

family = family))

# Labels don't inherit text:

update\_geom\_defaults("label",

list(size = 3,

hjust = 0,

family = family))

# Create the legend:

# The main elements of the plot (the boxplot, error bars, and count)

# are the easy part.

# The text describing each of those takes a lot of fiddling to

# get the location and style just right:

explain\_plot <- ggplot() +

stat\_boxplot(data = sample\_df,

aes(x = parameter, y=values),

geom ='errorbar', width = 0.3) +

geom\_boxplot(data = sample\_df,

aes(x = parameter, y=values),

width = 0.3, fill = "lightgrey") +

geom\_text(aes(x = 1, y = 950, label = "500"), hjust = 0.5) +

geom\_text(aes(x = 1.17, y = 950,

label = "Number of values"),

fontface = "bold", vjust = 0.4) +

theme\_minimal(base\_size = 5, base\_family = family) +

geom\_segment(aes(x = 2.3, xend = 2.3,

y = ggplot\_output[["25th percentile"]],

yend = ggplot\_output[["75th percentile"]])) +

geom\_segment(aes(x = 1.2, xend = 2.3,

y = ggplot\_output[["25th percentile"]],

yend = ggplot\_output[["25th percentile"]])) +

geom\_segment(aes(x = 1.2, xend = 2.3,

y = ggplot\_output[["75th percentile"]],

yend = ggplot\_output[["75th percentile"]])) +

geom\_text(aes(x = 2.4, y = ggplot\_output[["50th percentile\n(median)"]]),

label = "Interquartile\nrange", fontface = "bold",

vjust = 0.4) +

geom\_text(aes(x = c(1.17,1.17),

y = c(ggplot\_output[["upper\_whisker"]],

ggplot\_output[["lower\_whisker"]]),

label = c("Largest value within 1.5 times\ninterquartile range above\n75th percentile",

"Smallest value within 1.5 times\ninterquartile range below\n25th percentile")),

fontface = "bold", vjust = 0.9) +

geom\_text(aes(x = c(1.17),

y = ggplot\_output[["lower\_dots"]],

label = "Outside value"),

vjust = 0.5, fontface = "bold") +

geom\_text(aes(x = c(1.9),

y = ggplot\_output[["lower\_dots"]],

label = "-Value is >1.5 times and"),

vjust = 0.5) +

geom\_text(aes(x = 1.17,

y = ggplot\_output[["lower\_dots"]],

label = "<3 times the interquartile range\nbeyond either end of the box"),

vjust = 1.5) +

geom\_label(aes(x = 1.17, y = ggplot\_output[["quartiles"]],

label = names(ggplot\_output[["quartiles"]])),

vjust = c(0.4,0.85,0.4),

fill = "white", label.size = 0) +

ylab("") + xlab("") +

theme(axis.text = element\_blank(),

axis.ticks = element\_blank(),

panel.grid = element\_blank(),

aspect.ratio = 4/3,

plot.title = element\_text(hjust = 0.5, size = 10)) +

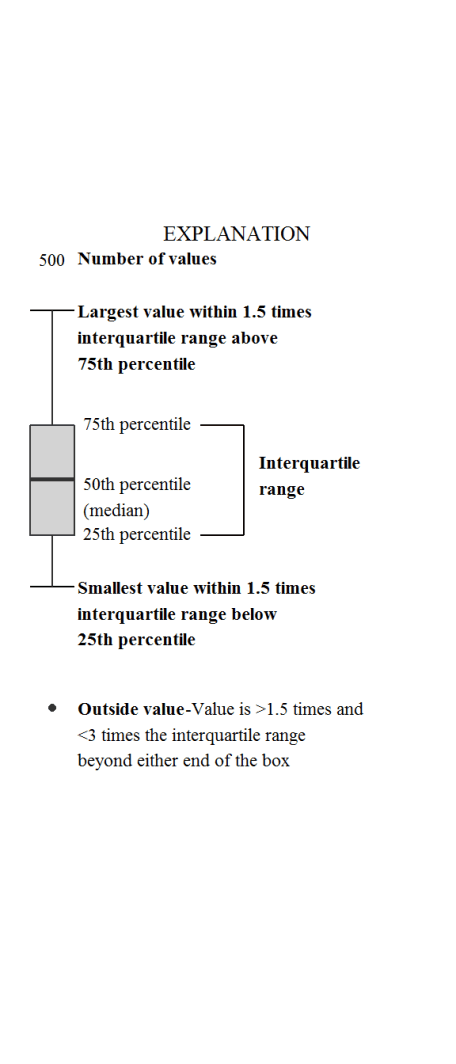
coord\_cartesian(xlim = c(1.4,3.1), ylim = c(-600, 900)) +

labs(title = "EXPLANATION")

return(explain\_plot)

}

ggplot\_box\_legend()



**Bring it together**

What’s nice about leaving this in the world of ggplot2 is that it is still possible to use other ggplot2 elements on the plot. For example, let’s add a reporting limit as horizontal lines to the phosphorous graph:

phos\_plot\_with\_DL <- phos\_plot +

geom\_hline(linetype = "dashed",

yintercept = 0.01)

explain\_plot\_DL <- ggplot\_box\_legend() +

geom\_segment(aes(y = -650, yend = -650,

x = 0.6, xend = 1.6),

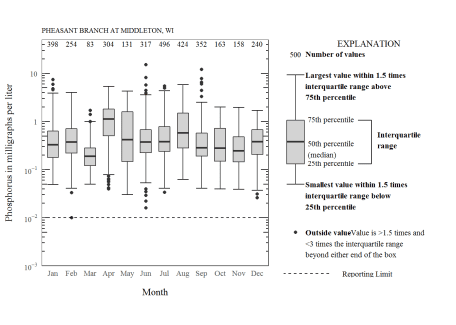
linetype="dashed") +

geom\_text(aes(y = -650, x = 1.8, label = "Reporting Limit"))

plot\_grid(phos\_plot\_with\_DL,

explain\_plot\_DL,

nrow = 1, rel\_widths = c(.6,.4))



I hoped you like my “deep dive” into ggplot2 boxplots. Many of the techniques here can be used to modify other ggplot2 plots.