Method for Graphic Display of Results of GAM Analysis

Curtis C. Bohlen, Casco Bay Estuary Partnership

1/28/2022

Contents

Introduction

One of my concerns with presenting the results of GAMs in a paper is that the models are complex, and the results can't usually be readily represented in a simple table. So, we need to think about potential graphic representation of the model results.

One possible way to display GAM model results (at least for GAMs without any two or three dimensional smoothers) is by showing "slices" through the model space associated with each predictor. The graphics would show the raw data, plus predicted values from `emmeans()`. The graphics can be "stacked" or "wrapped" into one graphic for compactness.

In this notebook, I work out steps needed to do that using `ggplot()`.

It may be worth noting that while poking around online, I also found an interesting package, called `mgcVis` that has graphics facilities for GAMS, but I have not explored that packages capabilities.

Load Libraries

```
library(tidyverse)
library(readxl)
library(mgcv)      # for GAM models
library(emmeans)   # For extracting useful "marginal" model summaries

theme_set(theme_classic())
```

Folder Reference

```
data_folder <- "Original_Data"
```

Load Data

```
filename.in <- "penob.station.data EA 3.12.20.xlsx"
file_path <- file.path(data_folder, filename.in)
station_data <- read_excel(file_path,
                           sheet="Final", col_types = c("skip", "date",
```

```
                                              "numeric", "text", "numeric",
                                              "text", "skip", "skip",
                                              "skip",
                                              rep("numeric", 10),
                                              "text",
                                              rep("numeric", 47),
                                              "text",
                                              rep("numeric", 12))) %>%
  rename_with(~ gsub(" ", "_", .x)) %>%
  rename_with(~ gsub("\\.", "_", .x)) %>%
  rename_with(~ gsub("\\?", "", .x)) %>%
  rename_with(~ gsub("%", "pct", .x)) %>%
  rename_with(~ gsub("_Abundance", "", .x)) %>%
  filter(! is.na(date))
#> New names:
#> * `` -> ...61
```

Station names are arbitrary, and Erin previously expressed interest in renaming them from Stations 2, 4, 5 and 8 to Stations 1,2,3,and 4.

The `factor()` function by default sorts levels before assigning numeric codes, so a convenient way to replace the existing station codes with sequential numbers is to create a factor and extract the numeric indicator values with `as.numeric()`.

```
station_data <- station_data %>%
  mutate(station = factor(as.numeric(factor(station))))
```

Subsetting to Desired Data Columns

I base selection of predictor variables on the ones used in the manuscript.

```
base_data <- station_data %>%
  rename(Date = date,
         Station = station,
         Year = year) %>%
  select(-c(month, month_num)) %>%
  mutate(Month = factor(as.numeric(format(Date, format = '%m')),
                                              levels = 1:12,
                                              labels = month.abb),
         DOY = as.numeric(format(Date,format = '%j'))) %>%
  mutate(season = factor(season, levels = c('Spring', 'Summer', 'Fall'))) %>%
  rename(Season = season,
         Temp = ave_temp_c,
         Sal = ave_sal_psu,
         Turb = sur_turb,
         AvgTurb = ave_turb_ntu,
         DOsat = ave_DO_Saturation,
         Chl = ave_chl_microgperl,
         RH = Herring
         ) %>%
  select(Date, Station, Year, Month, Season, DOY, riv_km, Temp, Sal, Turb, AvgTurb,
         DOsat, Chl, RH,
         combined_density,H, SEI,
```

```
        Acartia, Balanus, Eurytemora, Polychaete, Pseudocal, Temora) %>%
  arrange(Date, Station)
```

```
rm(station_data)
```

A Test GAM Model

I picked the gamma GLM for total zooplankton abundance, in part because it is a complicated model. If I can make graphics work for this one, the same methods will work for other models too.

```
test_gam <- gam(combined_density ~
                                Station +
                                s(Temp, bs="ts") +
                                s(Sal, bs="ts") +
                                s(log(Turb), bs="ts") +
                                s(log(Chl), bs="ts") +
                                s(log1p(RH),bs="ts"),
                                data = base_data, family = 'Gamma')
summary(test_gam)
#>
#> Family: Gamma
#> Link function: inverse
#>
#> Formula:
#> combined_density ~ Station + s(Temp, bs = "ts") + s(Sal, bs = "ts") +
#>     s(log(Turb), bs = "ts") + s(log(Chl), bs = "ts") + s(log1p(RH),
#>     bs = "ts")
#>
#> Parametric coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  2.711e-04  5.118e-05   5.296 2.81e-06 ***
#> Station2     8.833e-05  7.198e-05   1.227    0.226
#> Station3    -4.316e-05  6.567e-05  -0.657    0.514
#> Station4     1.672e-05  7.486e-05   0.223    0.824
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>                    edf Ref.df     F  p-value
#> s(Temp)      5.789e-01      9 0.249   0.0493 *
#> s(Sal)       3.049e+00      9 0.799   0.0593 .
#> s(log(Turb)) 1.629e+00      9 2.186 5.01e-05 ***
#> s(log(Chl))  1.184e-05      9 0.000   0.4354
#> s(log1p(RH)) 8.180e-06      9 0.000   0.4491
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.445   Deviance explained = 37.2%
#> GCV = 0.5638  Scale est. = 0.3217    n = 58
```

The important finding is that only the first three predictors are likely to have any value.

Marginal Predictions

We want to look at marginal predictions across each predictor. The tool for this is the`emmeans` package.

A "marginal" prediction estimates the mean value of the response variable, while varying the value of only one predictor, and holding the value of all **other** predictors fixed. Ordinarily, the other predictors are fixed to some "typical" level. I use median values, but the mean is more common, and would also make sense.

An Example

Here's what I have in mind, calculating and then plotting the marginal means (predictions) for various levels of Temperature, while holding all other predictors constant.

I calculate predictions at 40 locations. The number of locations affects the size of the data frame you are building and the resolution of your graphics. In my experience, the number is not critical for smooth curves on small graphics, but if you see straight line segments in the output, up the number of fitted points.

```
emm_temp <- as.data.frame(emmeans(test_gam, "Temp",
                    at = list(Temp = seq(1, 20, 0.5)),
                    cov.reduce = median,
                    type = 'response'))
```

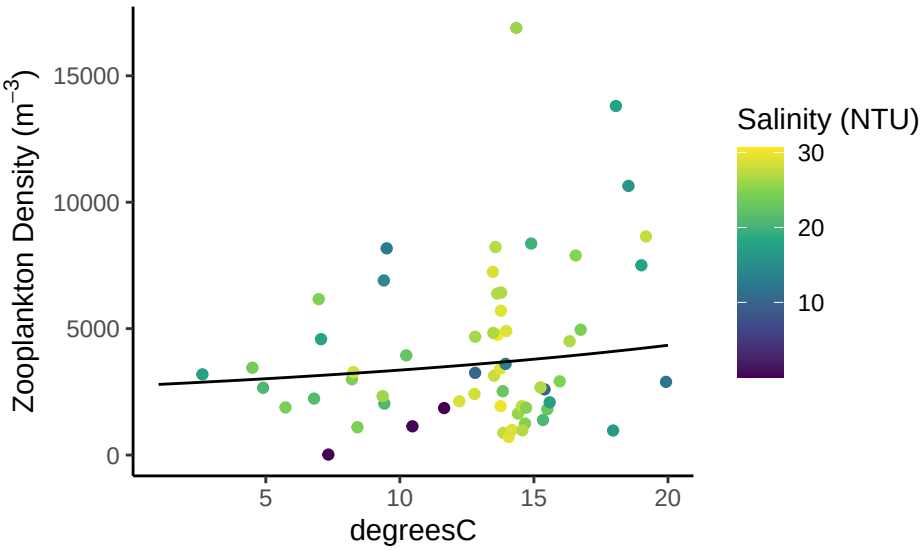
We build a `ggplot()` graphic showing the marginal effect of temperature
like this:

First a scatter plot of the underlying data. . . .

```
base_plt <- base_data %>%
  ggplot(aes(x = Temp, y = combined_density)) +
  geom_point(aes(color = Sal)) +
  scale_color_viridis_c(name = 'Salinity (NTU)') +
  xlab(expression(paste(degrees, 'C'))) +
  ylab(expression (paste('Zooplankton Density (', m^-3, ')')))
```

Then add the prediction line. that is a statistically significant increase in zooplankton abundance, associated with higher temperatures.

```
base_plt +
  geom_line(data = emm_temp, aes(x = Temp, y = response))
```



Calculate `emmeans()` for other predictors

I have to decide the range over which to plot the predictions for each predictor. I do that by looking at the observed range of that predictor, and selecting "pretty" limits just outside that range.

```
range(base_data$Sal)
#> [1]  0.02000 30.67636
emm_sal <- as.data.frame(emmeans(test_gam, "Sal",
                    at = list(Sal = seq(0, 30, 0.5)),
                    cov.reduce = median,
                    type = 'response'))
```

Turbidity entered the model as a log transform. The package `emmeans()` remembers that it did so, and handles the transform "behind the scenes", so you can specify the untransformed predictor. The same is true for the remaining transformed predictors.

Turbidity

```
range(base_data$Turb)
#> [1]  1.95000 13.64306
emm_turb <- as.data.frame(emmeans(test_gam, "Turb",
                    at = list(Turb = seq(1, 15, 0.25)),
                    cov.reduce = median,
                    type = 'response'))
```

Chlorophyll

```
range(base_data$Chl)
#> [1]  1.314348 17.792852
```

```
emm_chl <- as.data.frame(emmeans(test_gam, "Chl",
                    at = list(Chl = seq(1, 18, 0.5)),
                    cov.reduce = median,
                    type = 'response'))
```

River Herring

```
range(base_data$RH, na.rm = TRUE)
#> [1]    0.000 3021.257
emm_rh <- as.data.frame(emmeans(test_gam, "RH",
                    at = list(RH = seq(0, 3000, 50)),
                    cov.reduce = median,
                    type = 'response'))
```

Data For One Combined Graphic

We need to construct a data set suited to building a multi-panel display using `facet_wrap()`. That requires us to build a "long" data frame that replicates the observations once for every predictor, so we can facet according to the predictor.

A "Long" Version of the Data

I change the names of the predictors just to make facet labels a little easier to read. I'll get fancier with facet labels later.

```
long_data <- base_data %>%
  select(-c(H:Temora,  AvgTurb, DOsat)) %>%
  pivot_longer(-c(Date:riv_km, combined_density),
               names_to = 'Predictor', values_to = 'Values') %>%
  mutate(Predictor = factor(Predictor,
                            levels = c('Temp', 'Sal', 'Turb',
                                       'Chl', 'RH'),
                            labels = c("Temperature", "Salinity", "Turbidity",
                                       "Chlorophyll", "River Herring")))
```

Long Data for Predictions.

Now we generate a data set to provide the prediction lines. The goal is similar: a long data frame coded by the predictor we want on the X axis.

```
predicts <-
   bind_rows(Temperature = emm_temp, Salinity = emm_sal, Turbidity = emm_turb,
             Chlorophyll = emm_chl, 'River Herring' = emm_rh,
             .id = 'Predictor')
```

As we load each prediction data frame, it creates a new column named after the predictor in the GAM model. Each column is only defined for the values loaded from that data frame. It is `NA` everywhere else. A

sum using `a + b + c` would always return NA. But, we can add multiple columns together while ignoring any `NA`s using `rowSums(..., na.rm = TRUE)`.

If the two data frames are going plot correctly on one faceted plot, the `Predictor` factor here has to match the `Predictor` factor from the previous data frame (name and levels).

```
Value = rowSums(predicts[,c('Temp', 'Sal', 'Turb', 'Chl', 'RH')],
                          na.rm=TRUE)
predicts <- predicts %>%
  mutate(Value = Value) %>%
  mutate(Predictor = factor(Predictor,
                            levels = c("Temperature", "Salinity", "Turbidity",
                                       "Chlorophyll", "River Herring"))) %>%
  select(-c(Temp, Sal, Turb, Chl, RH))
```

Add Fancy Labels for Figures

These fancy labels are expressions, which allows me to include non-standard glyphs, Greek letters, superscripts, etc. They come in handy later, but I add them here because they are part of building the data frames for plotting.

```
plain_labels = levels = c("Temperature",
                          "Salinity",
                          "Turbidity",
                          "Chlorophyll",
                          "River Herring")

fancy_labels =  c(expression("Temperature (" * degree * "C)"),
                  expression("Salinity" ~ "(PSU)"),
                  expression("Turbidity" ~ "(NTU)"),
                  expression("Chlorophyll (" * mu * g * L ^-1 ~")"),
                  expression("River Herring" ~  "(CPUE)"))

long_data <- long_data %>%
  mutate(fancy_p = factor(Predictor, levels = plain_labels,
                          labels = fancy_labels))
predicts <- predicts %>%
  mutate(fancy_p = factor(Predictor, levels = plain_labels,
                          labels = fancy_labels))
```

Add Transformed Predictors

Again, we won't need this for a while, but it's handy later on.

```
new_values <- base_data %>%
  select(-c(H:Temora,  AvgTurb, DOsat)) %>%
  mutate(Turb = log(Turb),
         Chl = log(Chl),
         RH = log1p(RH)) %>%
  pivot_longer(-c(Date:riv_km, combined_density),
               names_to = 'Predictor', values_to = 'Values_2') %>%
  pull('Values_2')
```

```
fancy_labels_2 =  c(expression(atop("Temperature", "(" * degree * "C)")),
                    expression(atop("Salinity", "(PSU)")),
                    expression(atop("Log(Turbidity)", "(NTU)")),
                    expression(atop("Log(Chlorophyll)",
                                    "(" * mu * g * L ^-1 ~")")),
                    expression(atop("Log(1 + River Herring)",  "(CPUE)")))

long_data <- long_data %>%
  mutate(Transformed_Values = new_values,
         fancy_p_2 = factor(Predictor, levels = plain_labels,
                            labels = fancy_labels_2))

predicts <- predicts %>%
  mutate(Transformed_Values = case_when(
                        Predictor %in% c('Temperature', 'Salinity') ~ Value,
                        Predictor %in% c('Turbidity', 'Chlorophyll') ~ log(Value),
                        Predictor == 'River Herring' ~ log1p(Value)),
         fancy_p_2 = factor(Predictor, levels = plain_labels,
                            labels = fancy_labels_2))
```

Draft Graphic

It turns out, `ggplot()` automatically rescales axes to contain the values to be plotted. Usually, that's a convenience, but here, the confidence intervals are extremely wide, so the default y axis does not show the raw data well if I include confidence intervals. I need to specify the y axis. The y range should be slightly larger than the range of the observed response variable.

```
range(base_data$combined_density)
#> [1]    20.26076 16891.94108
```

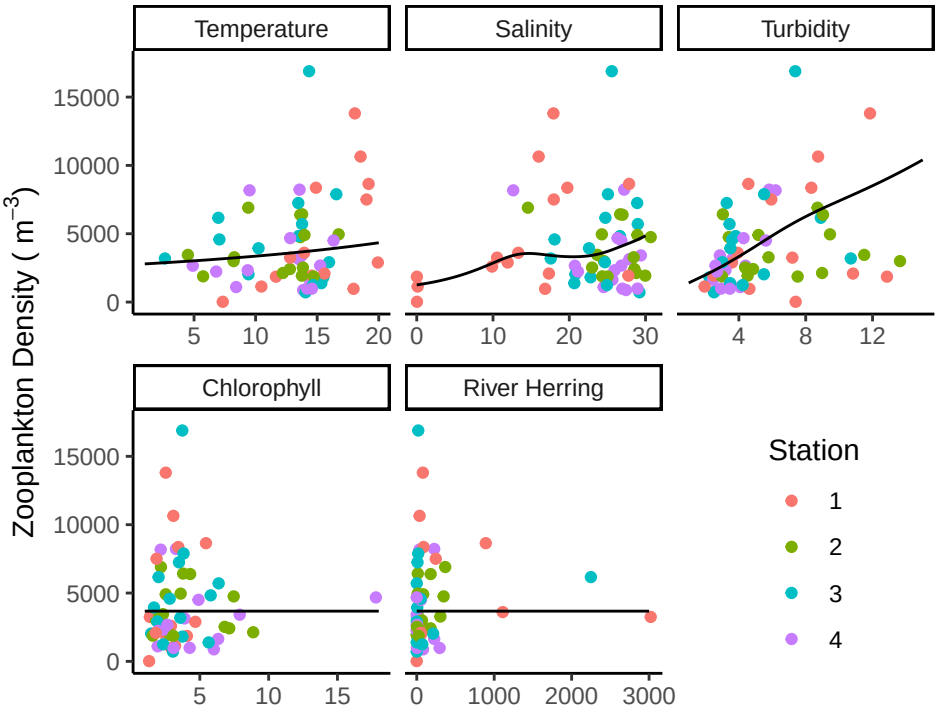
You can specify the axis limits two ways in `ggplot()`. `ylim()` drops data outside the specified range. That means any smoothers or summaries are based only on data visible on teh plot. Often, that's not what you want. You often want to change the plot boundaries as though you are just "zooming in" on an existing plat, without changing any calculations. The tool for that is the `coord_cartesian()` function, used here.

```
base_graphic = long_data %>%
  ggplot() +
  geom_point(data = long_data, aes(x = Values,
                                   y = combined_density,
                                   color = Station)) +
    geom_line(data = predicts,
              mapping = aes(x = Value, y =response)) +
  xlab('') +
  ylab(expression (paste('Zooplankton Density ( ', m^-3, ')'))) +
    coord_cartesian(ylim = c(0, 17500)) +

  facet_wrap(~Predictor, scales = 'free_x', nrow = 2) +

  theme(legend.position = c(.85, .2),
        legend.box = 'horizontal',
        legend.title.align = 0.5,)
```

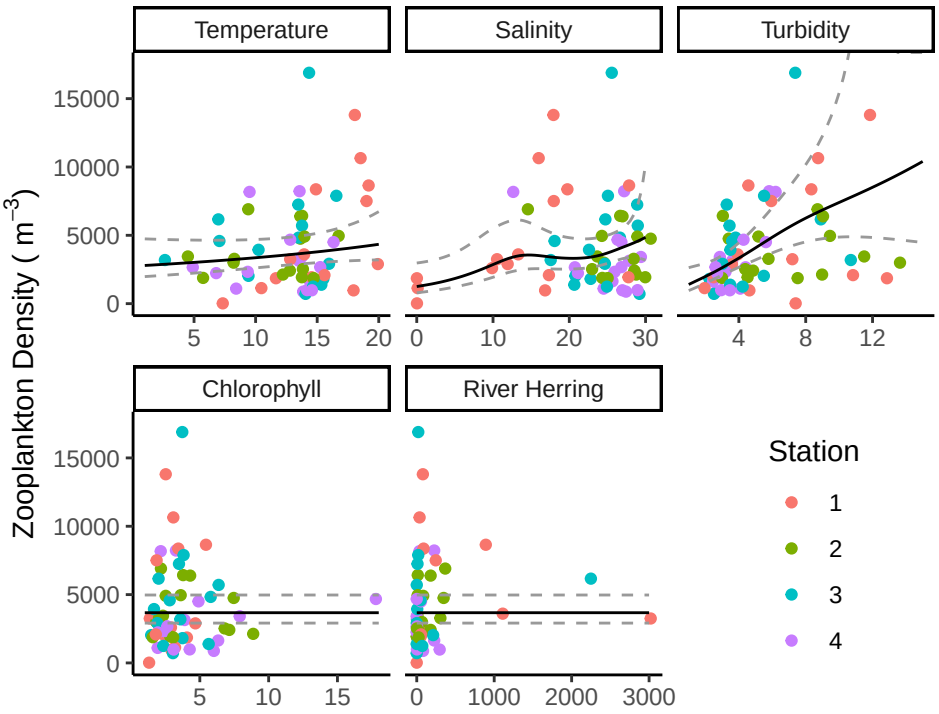
```
base_graphic
#> Warning: Removed 2 rows containing missing values (geom_point).
```



Add Error Bands

I could not get `geom_ribbon()` to work quite right. It won't plot the error band around the Turbidity predictor correctly, whether I specify the coordinated with `coord_cartesian()` or `ylim()`. So I ended up just drawing dashed lines to show the 95% confidence intervals.

```
base_graphic +
  geom_line(data = predicts,
            mapping = aes(x = Value, y = lower.CL), lty = 2, color = 'gray60') +
  geom_line(data = predicts,
            mapping = aes(x = Value, y = upper.CL), lty = 2, color = 'gray60')
#> Warning: Removed 2 rows containing missing values (geom_point).
```



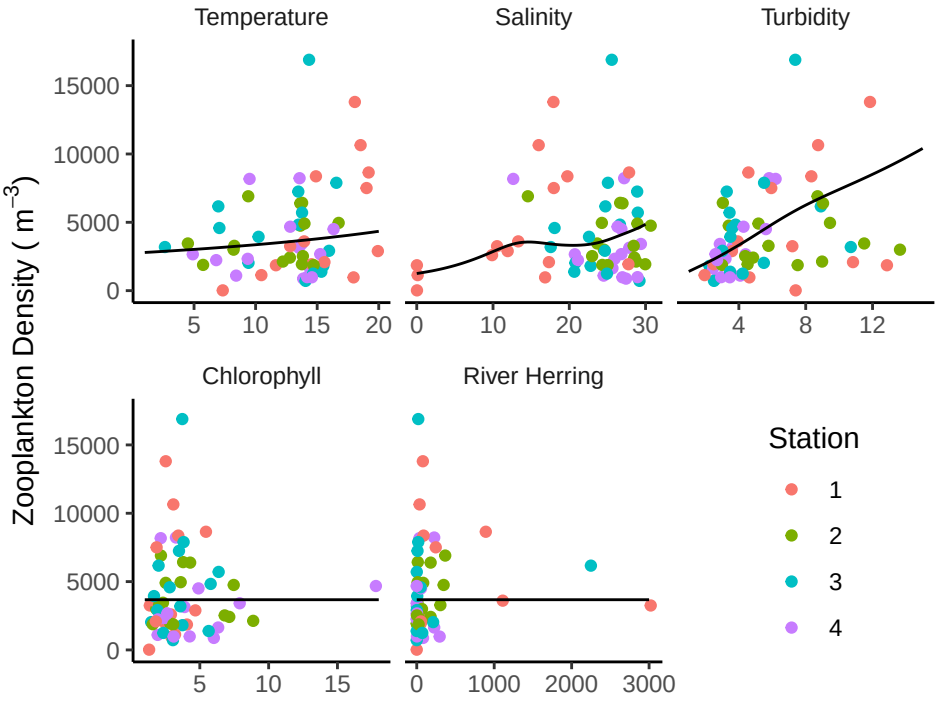
Cleaning Things Up

Several things I'd like to do:

1. Remove the ugly box around the facet labels
2. Add units to the panel labels
3. Experiment with different graphic formats

Remove Box Around Facet Labels

```
base_graphic +
  theme(strip.background = element_blank())
#> Warning: Removed 2 rows containing missing values (geom_point).
```



Make Better Facet Labels

The problem of making better facet labels is a bit tricky, since I need to create a new factor with values carefully tailored to generate labels that include units. The major trick here is that if I want to use Greek letters, superscripts, etc. in the facet labels, I need to create a factor with labels defined with `expression()`, which will be parsed by `ggplot()`. You tell `ggplot()` that you will pass it expressions by specifying `labeller = label_parsed` in the call to `facet_wrap()`.

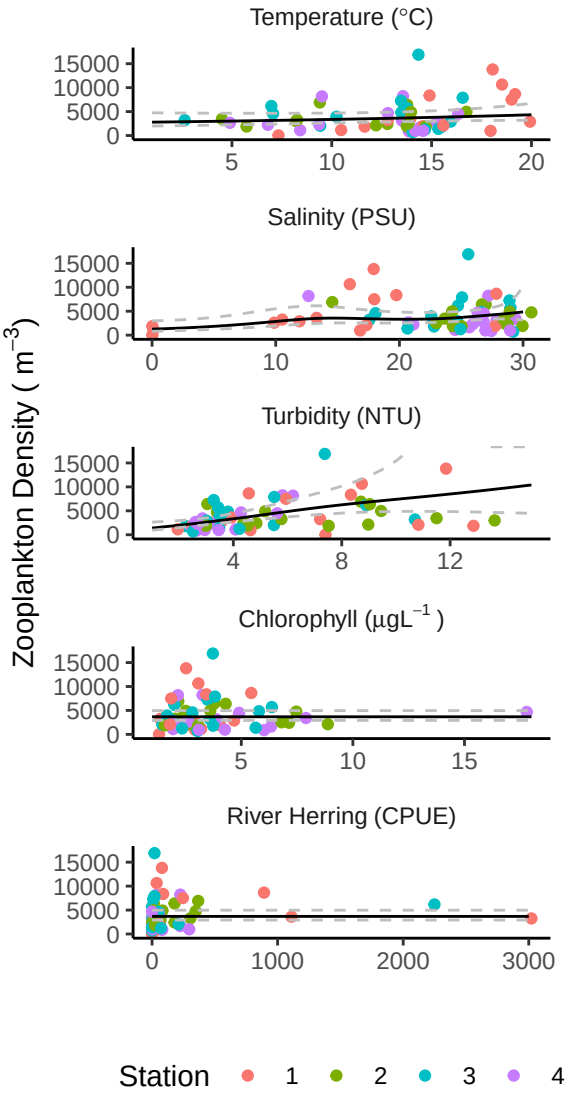
We built the necessary machinery into the data frame already, here we see how it works.

Three Alternative Layouts

```
fancy_graphic = long_data %>%
  ggplot(aes(Values, combined_density)) +
  geom_point(aes(color = Station)) +
  geom_line(data = predicts,
            mapping = aes(x = Value, y =response)) +
  geom_line(data = predicts,
              mapping = aes(x = Value, y = lower.CL), lty = 2, color = 'gray') +
  geom_line(data = predicts,
              mapping = aes(x = Value, y = upper.CL), lty = 2, color = 'gray') +
  xlab('') +
  ylab(expression (paste('Zooplankton Density ( ', m^-3, ')'))) +
  ylim(0, 17500) +
```

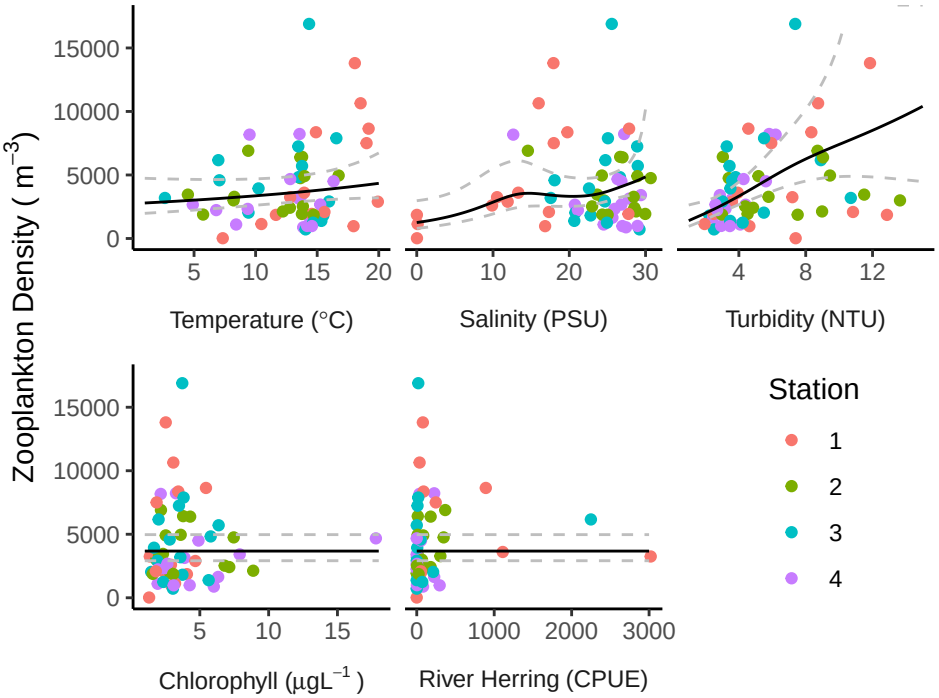
```
 theme(legend.position="bottom",
        strip.placement = 'outside',
        strip.background = element_blank())


fancy_graphic +
  facet_wrap(~fancy_p,
             scales = 'free_x',
             nrow = 5,
             strip.position = 'top',
             labeller = label_parsed)
#> Warning: Removed 2 rows containing missing values (geom_point).
```



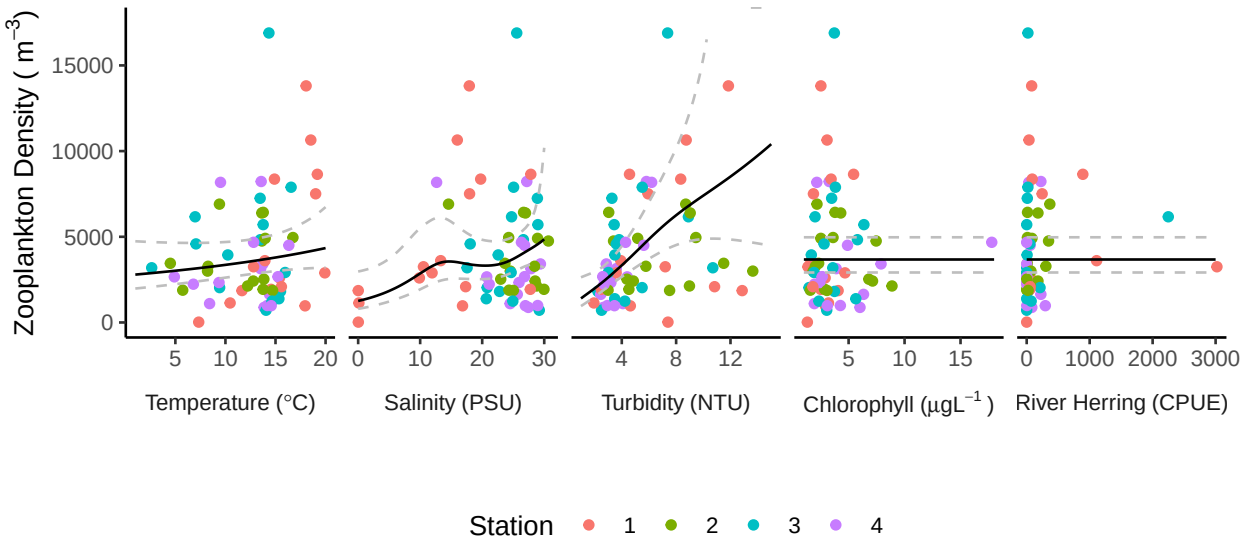
I also like `strip.position = 'bottom'`. It makes the facet labels function like X axis labels.

```
fancy_graphic +
  facet_wrap(~fancy_p,
             scales = 'free_x',
             nrow = 2,
             strip.position = 'bottom',
             labeller = label_parsed) +
  theme(legend.position = c(.85, .2),
        legend.box = 'horizontal',
        legend.title.align = 0.5,)
#> Warning: Removed 2 rows containing missing values (geom_point).
```



I also like the horizontal layout.

```
fancy_graphic +
  facet_wrap(~fancy_p,
             scales = 'free_x',
             nrow = 1,
             strip.position = 'bottom',
             labeller = label_parsed)
#> Warning: Removed 2 rows containing missing values (geom_point).
```



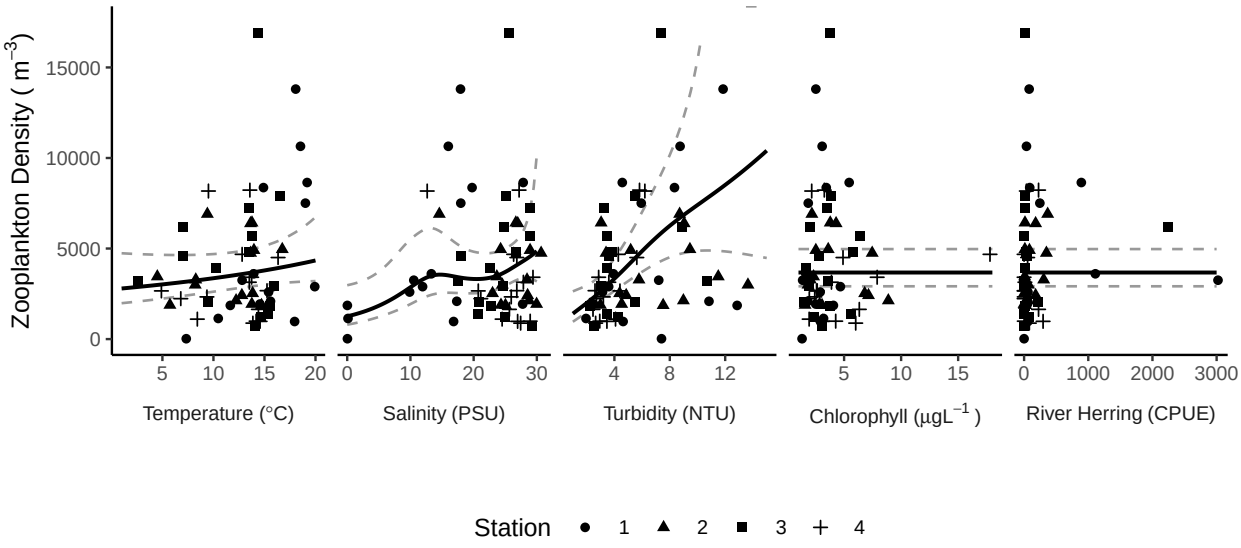
Use Shapes, Not Color for the Station

```
alt_graphic = long_data %>%
  ggplot(aes(Values, combined_density)) +

  geom_line(data = predicts,
            mapping = aes(x = Value, y =response), size = 0.75) +
  geom_line(data = predicts,
              mapping = aes(x = Value, y = lower.CL),
            lty = 2, color = 'gray60') +
  geom_line(data = predicts,
              mapping = aes(x = Value, y = upper.CL),
            lty = 2, color = 'gray60') +
  geom_point(aes(shape = Station)) +
  xlab('') +
  ylab(expression (paste('Zooplankton Density ( ', m^-3, ')'))) +
  ylim(0, 17500) +

  theme_classic(base_size = 10) +
  theme(legend.position="bottom",
        strip.placement = 'outside',
        strip.background = element_blank())

alt_graphic +
  facet_wrap(~fancy_p,
             scales = 'free_x',
             nrow = 1,
             strip.position = 'bottom',
             labeller = label_parsed)  #+
#> Warning: Removed 2 rows containing missing values (geom_point).
```




```
  # theme(legend.position = c(.85, .2),
  #       legend.box = 'horizontal',
  #       legend.title.align = 0.5,)
```

Add A Couple of Alternative Ideas

Since the GAM model was based on transformed predictors, you may want to present the results using transformed x axis values.

It might also be nice to NOT show prediction lines and error bands when the predictors are not meaningful. That just involves deleting the predicted values associated with certain from the predicts data frame.

I call `ggplot()` without arguments here, as it is slightly easier to control layering of graphics from multiple data frames that way. Each layer can now refer to its own data source.

The biggest problem with this design is that the long axis labels are problematic. Here I split them nto two rows, which works, but hte layout is not great looking.

```
predicts_2 <- predicts %>%
   filter(Predictor %in% c('Temperature', 'Salinity', 'Turbidity'))

fancy_graphic_2 <-
  ggplot() +
  geom_line(data = predicts_2,
            mapping = aes(x = Transformed_Values, y = response), size = 0.75) +
  geom_line(data = predicts_2,
              mapping = aes(x = Transformed_Values, y = lower.CL),
            lty = 2, color = 'gray') +
  geom_line(data = predicts_2,
              mapping = aes(x = Transformed_Values, y = upper.CL),
            lty = 2, color = 'gray') +
  geom_point(data = long_data,
             mapping = aes(x = Transformed_Values,
                           y = combined_density,
```

```
                          color = Station) ) +
  xlab('') +
  ylab(expression (paste('Zooplankton Density ( ', m^-3, ')'))) +
  ylim(0, 17500) +

  facet_wrap(~fancy_p_2,
             scales = 'free_x',
             nrow = 1,
             strip.position = 'bottom',
             labeller = label_parsed) +

  theme_classic(base_size = 9) +
  theme(legend.position = "bottom",
        legend.box = 'horizontal',
        legend.title.align = 0.5,
        strip.placement = 'outside',
        strip.background = element_blank(),
        #strip.text = element_text(size = 6)
        )

fancy_graphic_2
#> Warning: Removed 2 rows containing missing values (geom_point).
```