SEACAR Discrete Water Quality Analysis: Lab Bottom Chlorophyll a uncorrected for pheophytin

Last compiled on 08 June, 2023

Contents

Important Notes

These scripts were created by J.E. Panzik (jepanzik@usf.edu) for SEACAR.

All scripts and outputs can be found on the SEACAR GitHub repository:
https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses

This markdown file is designed to be compiled by SEACAR_WC_Discrete_ReportRender.R (https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses/blob/main/WC_Discrete/SEACAR_WC_Discrete_ReportRender.R).

Note: The top 2% of data is excluded when computing mean and standard deviations in plotting sections solely for the purpose of getting y-axis scales. The exclusion of the top 2% is not used in any statistics that are exported.

Libraries and Settings

Loads libraries used in the script. The inclusion of `scipen` option limits how frequently R defaults to scientific notation. Sets default settings for displaying warning and messages in created document, and sets figure dpi.

```
library(knitr)
library(data.table)
library(dplyr)
library(lubridate)
library(ggplot2)
library(ggpubr)
library(scales)
library(EnvStats)
library(tidyr)
library(kableExtra)
options(scipen=999)
knitr::opts_chunk$set(
   warning=FALSE,
   message=FALSE,
   dpi=200
)
```

File Import

Imports file that is determined in the SEACAR_WC_Discrete_ReportRender.R script.

The command `fread` is used because of its improved speed while handling large data files. Only columns that are used by the script are imported from the file, and are designated in the `select` input.

The script then gets the name of the parameter as it appears in the data file and units of the parameter.

The latest version of WC Discrete data is available at: https://usf.box.com/s/fbimxw4hrmazfn5b1d4jbn0addmcsld8

The file being used for the analysis is: **Combined_WQ_WC_NUT_Chlorophyll_a_uncorrected_for_pheophytin 2023-Jun-01.txt**

```
data <- fread(file_in, sep="|", header=TRUE, stringsAsFactors=FALSE,
        select=c("ManagedAreaName", "ProgramID", "ProgramName",
              "ProgramLocationID", "SampleDate", "Year", "Month",
              "RelativeDepth", "ActivityType", "ParameterName",
              "ResultValue", "ParameterUnits", "ValueQualifier",
              "SEACAR_QAQCFlagCode", "Include"), na.strings="")

parameter <- unique(data$ParameterName)
```

```
unit <- unique(data$ParameterUnits)
cat(paste("The data file(s) used:", file_short, sep="\n"))
```

```
## The data file(s) used:
## Combined_WQ_WC_NUT_Chlorophyll_a_uncorrected_for_pheophytin-2023-Jun-01.txt
```

Data Filtering

Most data filtering is performed on export from the database, and is indicated by the `Include` variable. `Include` values of 1 indicate the data should be used for analysis, values of 0 indicate the data should not be used for analysis. Documentation on the database filtering is provided here: SEACAR Documentation-Analysis Filters and Calculations.pdf

The filtering that is performed by the script at this point removes rows that are missing values for `ResultValue`, and only keeps data that is measured at the relative depth (surface, bottom, etc.) and activity type (field or sample) of interest. This is partly handled on export with the `RelativeDepth` variable, but there are some measurements that are considered both surface and bottom based on measurement depth and total depth. By default, these are marked as `Surface` for `RelativeDepth` and receive a `SEACAR_QAQCFlag` indicator of `12Q`. Data passes the filtering the process if it is from the correct depth and has an `Include` value of 1. The script also only looks at data of the desired `ActivityType` which indicates whether it was measured in the field (`Field`) or in the lab (`Sample`).

After the initial filtering, a second filter variable is created to determine whether enough time is represented in the monitoring location, which is that each monitoring location has 10 year or more of unique year entries and have at least 2 consecutive years of observations with at least 2 repeating months for observations that pass the initial filter. If data passes the first set of filtering criteria and the time criteria, they are used in the analysis.

The function that determines whether a managed area has at least 2 consecutive years of observations with at least 2 repeating months takes the data, creates a list of the managed areas and cycles through each managed area. For each managed area cycle:

1. List the unique years and put them in ascending order
2. If there are fewer than 2 unique years, skip to the next area
3. If there are 2 or more unique years, start a loop that compares adjacent year entries for the area
    - Start with the first two year entries
4. See if the year entries are subsequent years (1 year apart)
    - If not, skip to next pair of years
5. For the two years being compared, get the list of months for each
6. Compare the two lists of months to see what months are the same
    - If there are two or more months that are the same, the managed area passes the criteria and is stored in a variable
7. The list of managed areas that pass the 2 consecutive years with at least 2 repeating months is returned and used to determine if there is sufficient data for analysis.

A data frame is created that stores summary information for each managed area. This information is stored and combined with the results of the Seasonal Kendall Tau analysis and export to a data file once combined.

The sufficient data qualifier is merged with the original data, and a variable `Use_In_Analysis` is created to indicate what data should be used.

```
# Removes data rows with missing ResultValue
data <- data[!is.na(data$ResultValue),]
# Changes "Sample" to "Lab" for ActivityType
data$ActivityType <- gsub("Sample", "Lab", data$ActivityType)

# Gets data for the specific activity type for Chlorophyll, salinity, TSS,
# and Turbidity
if((param_name=="Chlorophyll_a_uncorrected_for_pheophytin" |
    param_name=="Salinity" | param_name=="Total_Suspended_Solids_TSS" |
    param_name=="Turbidity") & activity!="All"){
   data <- data[grep(activity, data$ActivityType[!is.na(data$ActivityType)]),]
}

# Changes RelativeDepth to Bottom for the QAQC flag 12Q that indicates
# measurements are both surface and bottom if the relative depth is bottom
if(depth=="Bottom"){
   data$RelativeDepth[grep("12Q", data$SEACAR_QAQCFlagCode[
      data$RelativeDepth=="Surface"])] <- "Bottom"
}
# Removes missing RelativeDepth data and data for RelativeDepth not of interest
# from all parameters except Secchi_Depth
if(param_name!="Secchi_Depth" & depth!="All"){
   data <- data[!is.na(data$RelativeDepth),]
   data <- data[data$RelativeDepth==depth,]
}

# Removes data rows that have "Blank" as an ActivityType
if(length(grep("Blank", data$ActivityType))>0){
   data <- data[-grep("Blank", data$ActivityType),]
}

# Removes data rows with ResultValue below 0, or -2 for Water_Temperature
if(param_name=="Water_Temperature"){
   data <- data[data$ResultValue>=-2,]
} else{
   data <- data[data$ResultValue>=0,]
}
# Changes Include to be either TRUE or FALSE
data$Include <- as.logical(data$Include)
# Changes Include to be TRUE for ProgramID 476 if it had the H value qualifier
data$Include[grep("H", data$ValueQualifier[data$ProgramID==476])] <- TRUE
# Change Include to be FALSE for Secchi_Depth with U value qualifier
if(param_name=="Secchi_Depth"){
   data$Include[grep("U", data$ValueQualifier)] <- FALSE
}
# Gets AreaID for data by merging data with the managed area list
data <- merge.data.frame(MA_All[,c("AreaID", "ManagedAreaName")],
            data, by="ManagedAreaName", all=TRUE)
# Creates function to checks managed area for at least 2 years of
# continuous consecutive data
DiscreteConsecutiveCheck <- function(con_data){
   # Gets AreaIDs
   IDs <- unique(con_data$AreaID[con_data$Include==TRUE &
```

```
                    !is.na(con_data$Include)])
  # Loops through each AreaID
  for(i in 1:length(IDs)) {
     # Gets list of Years for AreaID
     Years <- unique(con_data$Year[con_data$AreaID==IDs[i] &
                       con_data$Include==TRUE &
                       !is.na(con_data$Include)])
     # Puts Years in order
     Years <- Years[order(Years)]
     # If there are fewer than 2 years, skip to next AreaID
     if(length(Years)<2) {
        next
     }
     # Starts loop to make sure there are at least 2 consecutive years
     # with consecutive months of data
     for(j in 2:length(Years)) {
        # If adjacent year entries are not 1 year apart, skip to the
        # next set of year entries
        if(Years[j]-Years[j-1]!=1) {
           next
        }
        # Gets the list of months from the first year
        Months1 <- unique(con_data$Month[
           con_data$AreaID==IDs[i] &
              con_data$Year==Years[j-1] &
              con_data$Include==TRUE &
              !is.na(con_data$Include)])
        # Gets list of months for the second year
        Months2 <- unique(con_data$Month[
           con_data$AreaID==IDs[i] &
              con_data$Year==Years[j] &
              con_data$Include==TRUE &
              !is.na(con_data$Include)])
        # If there are more than 2 months shared between the two
        # years, the AreaID passes the check and is stored
        if(length(intersect(Months1, Months2))>=2) {
           # Creates variable for stored AreaID if it
           # doesn't exist
           if(exists("consecutive")==FALSE){
              consecutive <- IDs[i]
              break
           # Adds to variable for storing AreaID if does exist
           } else{
              consecutive <- append(consecutive, IDs[i])
              break
           }
        }
     }
  }
  # After going through all AreaID, return variable with list of all
  # that pass
  return(consecutive)
}
```

```
# Stores the AreaID that pass the consecutive year check
consMonthIDs <- DiscreteConsecutiveCheck(data)

# Creates data frame with summary for each managed area
MA_Summ <- data %>%
   group_by(AreaID, ManagedAreaName) %>%
   summarize(ParameterName=parameter,
          RelativeDepth=depth,
          ActivityType=activity,
          N_Data=length(ResultValue[Include==TRUE & !is.na(ResultValue)]),
          N_Years=length(unique(Year[Include==TRUE & !is.na(Year)])),
          EarliestYear=min(Year[Include==TRUE & N_Data!=0]),
          LatestYear=max(Year[Include==TRUE & N_Data!=0]),
          LastSampleDate=max(SampleDate[Include==TRUE]),
          ConsecutiveMonths=ifelse(unique(AreaID) %in%
                         consMonthIDs==TRUE, TRUE, FALSE),
          # Determines if monitoring location is sufficient for analysis
          # based on having more than 0 data entries, more than the
          # sufficient number of year, and the consecutive month criteria
          SufficientData=ifelse(N_Data>0 & N_Years>=suff_years &
                       ConsecutiveMonths==TRUE, TRUE, FALSE),
          Median=median(ResultValue[Include==TRUE & N_Data!=0], na.rm=TRUE))

MA_Summ$ConsecutiveMonths <- NULL
# Creates column in data that determines how many years from the start for each
# managed area
data <- data %>%
   group_by(AreaID, ManagedAreaName) %>%
   mutate(YearFromStart=Year-min(Year))
# Adds SufficientData column to data table based on managed area
data <- merge.data.frame(data, MA_Summ[,c("ManagedAreaName", "SufficientData")],
            by="ManagedAreaName")
# Creates Use_In_Analysis column for data that is determined if the row has
# Include value of TRUE and SufficientData value of TRUE
data$Use_In_Analysis <- ifelse(data$Include==TRUE & data$SufficientData==TRUE,
                TRUE, FALSE)
# Rearranges the summary data frame columns to be AreaID, ManagedAreaName,
# ParameterName RelativeDepth, ActivityType, SufficientData, everything else
MA_Summ <- MA_Summ %>%
   select(AreaID, ManagedAreaName, ParameterName, RelativeDepth, ActivityType,
       SufficientData, everything())
# Puts summary data in order based on managed area
MA_Summ <- as.data.frame(MA_Summ[order(MA_Summ$ManagedAreaName), ])
# Put SampleDate as date object
data$SampleDate <- as.Date(data$SampleDate)
# Creates character object for Month and Year
data$YearMonth <- paste0(data$Month, "-", data$Year)
# Creates variable that puts year and month into a decimal year format
data$YearMonthDec <- data$Year + ((data$Month-0.5) / 12)
# Converts ampleDate to a decimal date
data$DecDate <- decimal_date(data$SampleDate)

# Get list of and number of managed areas that are to be used in analysis
```

```
MA_Include <- MA_Summ$ManagedAreaName[MA_Summ$SufficientData==TRUE]
n <- length(MA_Include)
# Get list of and number of managed areas that are excluded from analysis
MA_Exclude <- MA_Summ[MA_Summ$N_Years<10 & MA_Summ$N_Years>0,]
MA_Exclude <- MA_Exclude[,c("ManagedAreaName", "N_Years")]
z <- nrow(MA_Exclude)
```

Data Impacted by Specific Value Qualifiers

Reports the amount of data impacted by the H (for dissolved oxygen & pH in program 476), I, Q, S (for Secchi depth), and U value qualifiers. It determines how much of the data for the given `ParametetrName`, `RelativeDepth`, and `ActivityType` is impacted by each value qualifier. Percentages are determined using 100*(# of value qualifier)/(# of total data)

A variable is also created that determines if scatter plot points should be a different color based on value qualifiers of interest.

A summary data frame is created that determines the amount of data and percentage of data impacted by the value qualifiers for each managed area by year and is written to a csv file in the output directory. Columns with `N` are the number impacted by the value qualifier, and those with `perc` are the percent of the data for that managed area and year impacted by the value qualifier. + WC Discrete Output Files in SEACAR GitHub (https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses/tree/main/WQ_Discrete/output)

```
# Find out how much total data exists and how much passed the initial filters
total <- length(data$Include)
pass_filter <- length(data$Include[data$Include==TRUE])
# Get the number and percentage of data entries impacted by value qualifier H
count_H <- length(grep("H", data$ValueQualifier[data$ProgramID==476]))
perc_H <- 100*count_H/length(data$ValueQualifier)
# Get the number and percentage of data entries impacted by value qualifier I
count_I <- length(grep("I", data$ValueQualifier))
perc_I <- 100*count_I/length(data$ValueQualifier)
# Get the number and percentage of data entries impacted by value qualifier Q
count_Q <- length(grep("Q", data$ValueQualifier))
perc_Q <- 100*count_Q/length(data$ValueQualifier)
# Get the number and percentage of data entries impacted by value qualifier S
count_S <- length(grep("S", data$ValueQualifier))
perc_S <- 100*count_S/length(data$ValueQualifier)
# Get the number and percentage of data entries impacted by value qualifier U
count_U <- length(grep("U", data$ValueQualifier))
perc_U <- 100*count_U/length(data$ValueQualifier)
# Copy ValueQualifier to a new VQ_Plot to create codes for plots
data$VQ_Plot <- data$ValueQualifier
# Determine if data with value qualifier H should be included for plots based
# on the parameter being observed
inc_H <- ifelse(param_name=="pH" | param_name=="Dissolved_Oxygen" |
          param_name=="Dissolved_Oxygen_Saturation", TRUE, FALSE)
# Loops through conditions to determine what indicators to include in plots.
# If H should be included
if (inc_H==TRUE){
      # Remove any Value qualifiers that aren't H or U
      data$VQ_Plot <- gsub("[^HU]+", "", data$VQ_Plot)
      # Standardize order of qualifiers. Puts UH as HU
```

```
      data$VQ_Plot <- gsub("UH", "HU", data$VQ_Plot)
      # Remove anything from ValueQualifier that isn't U from programs and that
      # aren't ProgramID 476
      data$VQ_Plot[na.omit(data$ProgramID!=476)] <-
            gsub("[^U]+", "", data$VQ_Plot[na.omit(data$ProgramID!=476)])
      # Changes blank character strings to NA
      data$VQ_Plot[data$VQ_Plot==""] <- NA
      # Prints the number and percentage of H, I, Q, U value qualifiers
      cat(paste0("Number of Measurements: ", total,
                 ", Number Passed Filter: ", pass_filter, "\n",
                 "Program 476 H Codes: ", count_H, " (", round(perc_H, 6), "%)\n",
                 "I Codes: ", count_I, " (", round(perc_I, 6), "%)\n",
                 "Q Codes: ", count_Q, " (", round(perc_Q, 6), "%)\n",
                 "U Codes: ", count_U, " (", round(perc_U, 6), "%)"))
# If Parameter is Secchi_Depth
} else if (param_name=="Secchi_Depth") {
      # Count the number of S ValueQualifier
      count_S <- length(grep("S", data$ValueQualifier))
      # Get percentage of S ValueQualifier
      perc_S <- 100*count_S/length(data$ValueQualifier)
      # Remove anything from ValueQualifier that isn't S or U
      data$VQ_Plot <- gsub("[^SU]+", "", data$VQ_Plot)
      # Change all ValueQualifier that are US to be US, standardizes codes
      data$VQ_Plot <- gsub("US", "SU", data$VQ_Plot)
      # Sets any blank character ValueQualifier to be NA
      data$VQ_Plot[data$VQ_Plot==""] <- NA
      # Prints the number and percentage of I, Q, S, U
      cat(paste0("Number of Measurements: ", total,
                 ", Number Passed Filter: ", pass_filter, "\n",
                 "I Codes: ", count_I, " (", round(perc_I, 6), "%)\n",
                 "Q Codes: ", count_Q, " (", round(perc_Q, 6), "%)\n",
                 "S Codes: ", count_S, " (", round(perc_S, 6), "%)\n",
                 "U Codes: ", count_U, " (", round(perc_U, 6), "%)"))
# For all other scenarios
} else{
      # Remove all ValueQualifier except U
      data$VQ_Plot <- gsub("[^U]+", "", data$VQ_Plot)
      # Sets any blank character ValueQualifier to be NA
      data$VQ_Plot[data$VQ_Plot==""] <- NA
      # Prints the number and percentage of I, Q, U
      cat(paste0("Number of Measurements: ", total,
                 ", Number Passed Filter: ", pass_filter, "\n",
                 "I Codes: ", count_I, " (", round(perc_I, 6), "%)\n",
                 "Q Codes: ", count_Q, " (", round(perc_Q, 6), "%)\n",
                 "U Codes: ", count_U, " (", round(perc_U, 6), "%)"))
}
```

```
## Number of Measurements: 953, Number Passed Filter: 953
## I Codes: 0 (0%)
## Q Codes: 0 (0%)
## U Codes: 919 (96.432319%)
```

```
# Creates a data table that summarizes the number and percentage of
# ValueQualifier H, I, Q, S, and U for each managed area each year
data_summ <- data %>%
      group_by(AreaID, ManagedAreaName, Year) %>%
      summarize(ParameterName=parameter,
                RelativeDepth=depth,
                ActivityType=activity,
                N_Total=length(ResultValue),
                N_AnalysisUse=length(ResultValue[Use_In_Analysis==TRUE]),
                N_H=length(grep("H", ValueQualifier[ProgramID==476])),
                perc_H=100*N_H/length(ValueQualifier),
                N_I=length(grep("I", ValueQualifier)),
                perc_I=100*N_I/length(ValueQualifier),
                N_Q=length(grep("Q", ValueQualifier)),
                perc_Q=100*N_Q/length(ValueQualifier),
                N_S=length(grep("S", ValueQualifier)),
                perc_S=100*N_S/length(ValueQualifier),
                N_U=length(grep("U", ValueQualifier)),
                perc_U=100*N_U/length(ValueQualifier))
# Orders the data table rows based on managed area name
data_summ <- as.data.table(data_summ[order(data_summ$ManagedAreaName,
                                           data_summ$Year), ])
# Writes the ValueQualifier summary to a csv file
fwrite(data_summ, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
                         activity, "_", depth, "_VQSummary.csv"), sep=",")
rm(data_summ)
```

Managed Area Statistics

Gets summary statistics for each managed area. Excluded managed areas are not included into whether the data should be used or not. Uses piping from dplyr package to feed into subsequent steps. The following steps are performed:

1. Take the `data` variable and only include rows that have a `SufficientData` value of TRUE
2. Group data that have the same `ManagedAreaName`, `Year`, and `Month`.
   - Second summary statistics do not use the `Month` grouping and are only for `ManagedAreaName` and `Year`.
   - Third summary statistics do not use `Year` grouping and are only for `ManagedAreaName` and `Month`
3. For each group, provide the following information: Number of Entries (N), Lowest Value (Min), Largest Value (Max), Median, Mean, Standard Deviation, and a list of all Program IDs included in these measurements.
4. Sort the data in ascending (A to Z and 0 to 9) order based on `ManagedAreaName` then `Year` then `Month`
5. Write summary stats to a pipe-delimited .txt file in the output directory
   - WC Discrete Output Files in SEACAR GitHub (https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses/tree/main/WQ_Discrete/output)

```
# Create summary statistics for each managed area based on Year and Month
# intervals.
MA_YM_Stats <- data[data$Use_In_Analysis==TRUE, ] %>%
   group_by(AreaID, ManagedAreaName, Year, Month) %>%
```

```
   summarize(ParameterName=parameter,
       RelativeDepth=depth,
       ActivityType=activity,
       N_Data=length(ResultValue),
       Min=min(ResultValue),
       Max=max(ResultValue),
       Median=median(ResultValue),
       Mean=mean(ResultValue),
       StandardDeviation=sd(ResultValue),
       Programs=paste(sort(unique(ProgramName), decreasing=FALSE),
               collapse=', '),
       ProgramIDs=paste(sort(unique(ProgramID), decreasing=FALSE),
               collapse=', '))
# Puts the data in order based on ManagedAreaName, Year, then Month
MA_YM_Stats <- as.data.table(MA_YM_Stats[order(MA_YM_Stats$ManagedAreaName,
                        MA_YM_Stats$Year,
                        MA_YM_Stats$Month), ])
# Writes summary statistics to file
fwrite(MA_YM_Stats, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
               activity, "_", depth, "_MA_MMYY_Stats.txt"), sep="|")
# Get year from start for each managed area
MA_YM_Stats <- MA_YM_Stats %>%
   group_by(AreaID, ManagedAreaName) %>%
   mutate(YearFromStart=Year-min(Year))
# Create decimal value of year and month values
MA_YM_Stats$YearMonthDec <- MA_YM_Stats$Year + ((MA_YM_Stats$Month-0.5) / 12)
# Create summary statistics for each managed area based on Year intervals.
MA_Y_Stats <- data[data$Use_In_Analysis==TRUE, ] %>%
   group_by(AreaID, ManagedAreaName, Year) %>%
   summarize(ParameterName=parameter,
       RelativeDepth=depth,
       ActivityType=activity,
       N=length(ResultValue),
       Min=min(ResultValue),
       Max=max(ResultValue),
       Median=median(ResultValue),
       Mean=mean(ResultValue),
       StandardDeviation=sd(ResultValue),
       Programs=paste(sort(unique(ProgramName), decreasing=FALSE),
               collapse=', '),
       ProgramIDs=paste(sort(unique(ProgramID), decreasing=FALSE),
               collapse=', '))
# Puts the data in order based on ManagedAreaName then Year
MA_Y_Stats <- as.data.table(MA_Y_Stats[order(MA_Y_Stats$ManagedAreaName,
                      MA_Y_Stats$Year), ])
# Writes summary statistics to file
fwrite(MA_Y_Stats, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
               activity, "_", depth, "_MA_Yr_Stats.txt"), sep="|")
rm(MA_Y_Stats)
# Create summary statistics for each managed area based on Month intervals.
MA_M_Stats <- data[data$Use_In_Analysis==TRUE, ] %>%
   group_by(AreaID, ManagedAreaName, Month) %>%
   summarize(ParameterName=parameter,
```

```
        RelativeDepth=depth,
        ActivityType=activity,
        N=length(ResultValue),
        Min=min(ResultValue),
        Max=max(ResultValue),
        Median=median(ResultValue),
        Mean=mean(ResultValue),
        StandardDeviation=sd(ResultValue),
        Programs=paste(sort(unique(ProgramName), decreasing=FALSE),
                collapse=', '),
        ProgramIDs=paste(sort(unique(ProgramID), decreasing=FALSE),
                collapse=', '))
# Puts the data in order based on ManagedAreaName then Month
MA_M_Stats <- as.data.table(MA_M_Stats[order(MA_M_Stats$ManagedAreaName,
                        MA_M_Stats$Month), ])
# Writes summary statistics to file
fwrite(MA_M_Stats, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
                activity, "_", depth, "_MA_Mo_Stats.txt"), sep="|")
rm(MA_M_Stats)
```

Monitoring Location Statistics

Gets monitoring location statistics, which is defined as a unique combination of `ManagedAreaName`, `ProgramID`, `ProgramAreaName`, and `ProgramLocationID`, using piping from dplyr package. The following steps are performed:

1. Take the `data` variable and only include rows that have a `SufficientData` value of `TRUE`
2. Group data that have the same `ManagedAreaName`, `ProgramID`, `ProgramName`, and `ProgramLocationID`.
3. For each group, provide the following information: Earliest Sample Date (EarliestSampleDate), Latest Sample Date (LastSampleDate), Number of Entries (N), Lowest Value (Min), Largest Value (Max), Median, Mean, and Standard Deviation.
4. Sort the data in ascending (A to Z and 0 to 9) order based on `ManagedAreaName` then `ProgramName` then `ProgramID` then `ProgramLocationID`
5. Write summary stats to a pipe-delimited .txt file in the output directory
    - WC Discrete Output Files in SEACAR GitHub (https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses/tree/main/WQ_Discrete/output)

```
# Gets summary statistics for monitoring locations, which are defined as unique
# combinations of ManagedAreaName, ProgramID, And ProgramLocationID
Mon_Stats <- data[data$Use_In_Analysis==TRUE, ] %>%
   group_by(AreaID, ManagedAreaName, ProgramID, ProgramName,
            ProgramLocationID) %>%
   summarize(ParameterName=parameter,
        RelativeDepth=depth,
        ActivityType=activity,
        EarliestSampleDate=min(SampleDate),
        LastSampleDate=max(SampleDate),
        N_Data=length(ResultValue),
        Min=min(ResultValue),
        Max=max(ResultValue),
        Median=median(ResultValue),
```

```
      Mean=mean(ResultValue),
      StandardDeviation=sd(ResultValue))
# Order data rows by ManagedAreaName, ProgramName, ProgramID, then
# ProgramLocationID
Mon_Stats <- as.data.table(Mon_Stats[order(Mon_Stats$ManagedAreaName,
                     Mon_Stats$ProgramName,
                     Mon_Stats$ProgramID,
                     Mon_Stats$ProgramLocationID), ])
# Write summary statistics to file
fwrite(Mon_Stats, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
               activity, "_", depth, "_MonLoc_Stats.txt"), sep="|")
rm(Mon_Stats)
```

Seasonal Kendall Tau Analysis

Gets seasonal Kendall Tau statistics using the `kendallSeasonalTrendTest` from the `EnvStats` package. The `Trend` parameter is determined from a user-defined function based on the median, Senn slope, and p values from the data. Analysis modified from code created by Jason Scolaro that performed at The Water Atlas: https://sarasota.wateratlas.usf.edu/water-quality-trends/#analysis-overview

The following steps are performed:

1. Define the functions used in the analysis
2. Check to see if there are any groups to run analysis on.
3. Take the `data` variable and only include rows that have a `SufficientData` value of `TRUE`
4. Group data that have the same `ManagedAreaName`.
5. For each group, provides the following information: Earliest Sample Date (EarliestSampleDate), Latest Sample Date (LastSampleDate), Number of Entries (N), Lowest Value (Min), Largest Value (Max), Median, Mean, Standard Deviation, tau, Senn Slope (SennSlope), Senn Intercept (SennIntercept), and p.
    - The analysis is run with the `kendallSeasonalTrendTest` function using the `Year` values for year, and `Month` as the seasonal qualifier, and Trend.
    - An independent.obs value of `TRUE` indicates that the data should be treated as not being serially auto-correlated. An independent.obs value of `FALSE` indicates that it is treated as being serially auto-correlated, but also requires one observation per season per year for the full time of observation.
6. Reformat columns in the data frame from export.
7. Write summary stats to a pipe-delimited .txt file in the output directory
    - WC Discrete Output Files in SEACAR GitHub (https://github.com/FloridaSEACAR/SEACAR_Trend_Analyses/tree/main/WQ_Discrete/output)

```
# Creates function to get the Kendall Tau statistics
tauSeasonal <- function(dat, independent, stats.median, stats.minYear,
               stats.maxYear) {
   tau <- NULL
   # Stores results from seasonal Kendall Tau
   tryCatch({ken <- kendallSeasonalTrendTest(
   y=dat$Mean,
   season=dat$Month,
   year=dat$YearFromStart,
```

```
   independent.obs=independent)
   # Gets the values of interest from the trend fit
   tau <- ken$estimate[1]
   p <- ken$p.value[2]
   slope <- ken$estimate[2]
   intercept <- ken$estimate[3]
   chi_sq <- ken$statistic[1]
   p_chi_sq <- ken$p.value[1]
   trend <- trend_calculator(slope, stats.median, p)
   rm(ken)
   # Prints warnings if a fit does not exist and stores values as NA
   }, warning=function(w) {
   print(w)
   }, error=function(e) {
   print(e)
   }, finally={
   if (!exists("tau")) {
      tau <- NA
   }
   if (!exists("p")) {
      p <- NA
   }
   if (!exists("slope")) {
      slope <- NA
   }
   if (!exists("intercept")) {
      intercept <- NA
   }
   if (!exists("trend")) {
      trend <- NA
   }
   })
   # Puts variables in a vector for the managed area currently being analyzed
   KT <-c(unique(dat$AreaID),
       unique(dat$ManagedAreaName),
       independent,
       tau,
       p,
       slope,
       intercept,
       chi_sq,
       p_chi_sq,
       trend)
   # Returns the fit parameters
   return(KT)
}
# Function that determines statistics from data
runStats <- function(dat, med, minYr, maxYr) {
   # Get basic stats
   dat$Mean <- as.numeric(dat$Mean)
   stats.median <- med
   stats.minYear <- minYr
   stats.maxYear <- maxYr
```

```
   # Calculate Kendall Tau and Slope stats assuming they are serially
   # independent, then store in variable
   KT <- tauSeasonal(dat, TRUE, stats.median,
            stats.minYear, stats.maxYear)
   # If variable returned is empty, run again assuming they are NOT serially
   # independent
   if (is.null(KT[9])) {
   KT <- tauSeasonal(dat, FALSE, stats.median,
            stats.minYear, stats.maxYear)
   }
   # If KT.Stats does not exist, create it and store values
   if (is.null(KT.Stats)==TRUE) {
   KT.Stats <- KT
   # If KT.Stats does exist, add values to it
   } else{
   KT.Stats <- rbind(KT.Stats, KT)
   }
   return(KT.Stats)
}
# Function to determine trend of Kendal Tau
trend_calculator <- function(slope, median_value, p) {
   # Trend depends on series of conditions
   trend <-
   # If the p value is less than 5% and the slope is greater than 10% of the
   # median value, the trend is large (2).
   if (p < .05 & abs(slope) > abs(median_value) / 10.) {
      if (slope > 0) {
      2
      }
      else {
      -2
      }
   }
   # If the p value is less than 5% and the slope is less than 10% of the
   # median value, there is a trend (1).
   else if (p < .05 & abs(slope) < abs(median_value) / 10.) {
   if (slope > 0) {
      1
   }
   else {
      -1
   }
   # Otherwise, there is no trend (0)
   } else
   0
   return(trend)
}
# Creates a null data frame for storing kendall tau results
KT.Stats <- NULL
# List for column names
c_names <- c("AreaID", "ManagedAreaName", "Independent", "tau", "p",
       "SennSlope", "SennIntercept", "ChiSquared", "pChiSquared", "Trend")
# Determines if there are any monitoring locations to analyze
```

```
if(n==0){
   # Creates data frame to store analysis values in
   KT.Stats <- data.frame(matrix(ncol=length(c_names),
                  nrow=length(MA_Summ$ManagedAreaName)))
   colnames(KT.Stats) <- c_names

   KT.Stats[, c("AreaID", "ManagedAreaName")] <-
   MA_Summ[, c("AreaID", "ManagedAreaName")]
} else{
   # Starts cycling through managed areas to determine seasonal Kendall Tau
   for (i in 1:n) {
   # Gets the number of rows of data for the managed area
   x <- nrow(MA_YM_Stats[MA_YM_Stats$ManagedAreaName==MA_Include[i], ])
   # Perform analysis if there is more than 1 row
   if (x>0) {
      # Store the managed area summary statistics to be used in trend analysis
      SKT.med <- MA_Summ$Median[MA_Summ$ManagedAreaName==MA_Include[i]]
      SKT.minYr <- MA_Summ$EarliestYear[MA_Summ$ManagedAreaName==
                        MA_Include[i]]
      SKT.maxYr <- MA_Summ$LatestYear[MA_Summ$ManagedAreaName==MA_Include[i]]
      # Get seasonal Kendall Tau statistics by running data for managed areas
      # through the functions
      KT.Stats <- runStats(MA_YM_Stats[MA_YM_Stats$ManagedAreaName==
                        MA_Include[i], ],
               SKT.med, SKT.minYr, SKT.maxYr)
   }
   }
   # Stores as data frame
   KT.Stats <- as.data.frame(KT.Stats)
   # If there was only one location, it is stored as a column vector. Change to
   # row vector
   if(dim(KT.Stats)[2]==1){
   KT.Stats <- as.data.frame(t(KT.Stats))
   }
   # Sets column and row names for KT.Stats
   colnames(KT.Stats) <- c_names
   rownames(KT.Stats) <- seq(1:nrow(KT.Stats))
   # Sets variables to proper format and rounds values if necessary
   KT.Stats$tau <- round(as.numeric(KT.Stats$tau), digits=4)
   KT.Stats$p <- round(as.numeric(KT.Stats$p), digits=4)
   KT.Stats$SennSlope <- as.numeric(KT.Stats$SennSlope)
   KT.Stats$SennIntercept <- as.numeric(KT.Stats$SennIntercept)
   KT.Stats$ChiSquared <- round(as.numeric(KT.Stats$ChiSquared), digits=4)
   KT.Stats$pChiSquared <- round(as.numeric(KT.Stats$pChiSquared), digits=4)
   KT.Stats$Trend <- as.integer(KT.Stats$Trend)
}
# Combines the KT.Stats with MA_Summ
KT.Stats <-  merge.data.frame(MA_Summ, KT.Stats,
               by=c("AreaID", "ManagedAreaName"), all=TRUE)

KT.Stats <- as.data.table(KT.Stats[order(KT.Stats$ManagedAreaName), ])
# Writes combined statistics to file
fwrite(KT.Stats, paste0(out_dir_param,"/WC_Discrete_", param_abrev, "_",
```

```
                activity, "_", depth, "_KendallTau_Stats.txt"),
    sep="|")
# Removes data rows for managed areas with no ResultValue
data <- data[!is.na(data$ResultValue),]
# Gets x and y values for starting point for trendline
KT.Plot <- KT.Stats %>%
   group_by(AreaID, ManagedAreaName) %>%
   summarize(x=EarliestYear,
       y=SennIntercept)
# Gets x and y values for ending point for trendline
KT.Plot2 <- KT.Stats %>%
   group_by(AreaID, ManagedAreaName) %>%
   summarize(x=decimal_date(LastSampleDate),
       y=(x-EarliestYear)*SennSlope+SennIntercept)
# Combines the starting and endpoints for plotting the trendline
KT.Plot <- bind_rows(KT.Plot, KT.Plot2)
rm(KT.Plot2)
KT.Plot <- as.data.table(KT.Plot[order(KT.Plot$ManagedAreaName), ])
KT.Plot <- KT.Plot[!is.na(KT.Plot$y),]
```

Appendix I: Scatter Plot of Entire Dataset

This part will create a scatter plot of the all data that passed initial filtering criteria with points colored based on specific value qualifiers. The values determined at the beginning (`year_lower`, `year_upper`, `min_RV`, `mn_RV`, `x_scale`, and `y_scale`) are solely for use by the plotting functions and are not output as part of the computed statistics.

```
# Defines standard plot theme: black and white, no major or minor grid lines,
# Arial font. Title is centered, size 12, and blue (hex coded). Subtitle is
# centered, size 10, and blue (hex coded). Legend title is size 10 and the
# legend is left-justified. X-axis title is size 10 and the margins are padded
# at the top and bottom to give more space for angled axis labels. Y-axis title
# is size 10 and margins are padded on the right side to give more space for
# axis labels. Axis labels are size 10 and the x-axis labels are rotated -45
# degrees with a horizontal justification that aligns them with the tick mark
plot_theme <- theme_bw() +
   theme(panel.grid.major = element_blank(),
      panel.grid.minor = element_blank(),
      text=element_text(family="Arial"),
      plot.title=element_text(hjust=0.5, size=12, color="#314963"),
      plot.subtitle=element_text(hjust=0.5, size=10, color="#314963"),
      legend.title=element_text(size=10),
      legend.text.align = 0,
      axis.title.x = element_text(size=10, margin = margin(t = 5, r = 0,
                                  b = 10, l = 0)),
      axis.title.y = element_text(size=10, margin = margin(t = 0, r = 10,
                                  b = 0, l = 0)),
      axis.text=element_text(size=10),
      axis.text.x=element_text(angle = 60, hjust = 1))
# Gets first and most recent years from data set
year_lower <- min(data$Year)
year_upper <- max(data$Year)
```

```
# Gets minimum, mean, and standard deviation of ResultValue for setting y-axis
# scale
min_RV <- min(data$ResultValue)
mn_RV <- mean(data$ResultValue[data$ResultValue <
                  quantile(data$ResultValue, 0.98)])
sd_RV <- sd(data$ResultValue[data$ResultValue <
                quantile(data$ResultValue, 0.98)])
x_scale <- ifelse(year_upper - year_lower > 30, 10, 5)
y_scale <- mn_RV + 4 * sd_RV

# Create plot object for auto-scaled y-axis plot
p1 <- ggplot(data=data[data$Include==TRUE,],
       aes(x=SampleDate, y=ResultValue, fill=VQ_Plot)) +
   geom_point(shape=21, size=3, color="#333333", alpha=0.75) +
   labs(subtitle="Autoscale",
     x="Year", y=paste0("Values (", unit, ")"),
     fill="Value Qualifier") +
   plot_theme +
   theme(legend.position="top", legend.box="horizontal",
      legend.justification="right") +
   scale_x_date(labels=date_format("%Y")) +
   {if(inc_H==TRUE){
   scale_fill_manual(values=c("H"= "#F8766D", "U"= "#00BFC4",
                  "HU"="#7CAE00"), na.value="#cccccc")
   } else if(param_name=="Secchi_Depth"){
   scale_fill_manual(values=c("S"= "#F8766D", "U"= "#00BFC4",
                  "SU"="#7CAE00"), na.value="#cccccc")
   } else {
   scale_fill_manual(values=c("U"= "#00BFC4"), na.value="#cccccc")
   }}
# Create plot object for y-axis scaled plot
p2 <- ggplot(data=data[data$Include==TRUE,],
       aes(x=SampleDate, y=ResultValue, fill=VQ_Plot)) +
   geom_point(shape=21, size=3, color="#333333", alpha=0.75) +
   ylim(min_RV, y_scale) +
   labs(subtitle="Scaled to 4x Standard Deviation",
     x="Year", y=paste0("Values (", unit, ")")) +
   plot_theme +
   theme(legend.position="none") +
   scale_x_date(labels=date_format("%Y")) +
   {if(inc_H==TRUE){
   scale_fill_manual(values=c("H"= "#F8766D", "U"= "#00BFC4",
                  "HU"="#7CAE00"), na.value="#cccccc")
   } else if(param_name=="Secchi_Depth"){
   scale_fill_manual(values=c("S"= "#F8766D", "U"= "#00BFC4",
                  "SU"="#7CAE00"), na.value="#cccccc")
   } else {
   scale_fill_manual(values=c("U"= "#00BFC4"), na.value="#cccccc")
   }}
# Create legend object
leg <- get_legend(p1)
# Arrange plots and legend
pset <- ggarrange(leg, p1 + theme(legend.position="none"), p2,
```

```
        ncol=1, heights=c(0.1, 1, 1))
# Create title object
p0 <- ggplot() + labs(title="Scatter Plot for Entire Dataset") +
   plot_theme + theme(panel.border=element_blank(),
            panel.grid.major=element_blank(),
            panel.grid.minor=element_blank(),
            axis.line=element_blank())
# Arrange and print title with plots
ggarrange(p0, pset, ncol=1, heights=c(0.1, 1))
```

Appendix II: Dataset Summary Box Plots

Box plots are created by using the entire data set and excludes any data that has been previously filtered out. The scripts that create plots follow this format

1. Use the data set that only has `SufficientData` of `TRUE`
2. Set what values are to be used for the x-axis, y-axis, and the variable that should determine groups for the box plots
3. Set the plot type as a box plot with the size of the outlier points
4. Create the title, x-axis, y-axis, and color fill labels
5. Set the y and x limits
6. Make the axis labels bold
7. Plot the arrangement as a set of panels

This set of box plots are grouped by year.

```
# Get minimum, mean, and standard deviation of the data
min_RV <- min(data$ResultValue[data$Include==TRUE])
mn_RV <- mean(data$ResultValue[data$Include==TRUE &
                  data$ResultValue <
                  quantile(data$ResultValue, 0.98)])
sd_RV <- sd(data$ResultValue[data$Include==TRUE &
                 data$ResultValue <
                 quantile(data$ResultValue, 0.98)])
# Sets y scale based on data
y_scale <- mn_RV + 4 * sd_RV
# Create plot object for auto-scaled y-axis plot
p1 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=Year, y=ResultValue, group=Year)) +
   geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
          outlier.size=3, outlier.color="#333333",
          outlier.fill="#cccccc", outlier.alpha=0.75) +
   labs(subtitle="Autoscale", x="Year",
     y=paste0("Values (", unit, ")")) +
   plot_theme
# Create plot object for y-axis scaled plot
p2 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=Year, y=ResultValue, group=Year)) +
   geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
          outlier.size=3, outlier.color="#333333",
          outlier.fill="#cccccc", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation", x="Year",
     y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   plot_theme
# Create plot object for y-axis scaled plot for past 10 years
p3 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=as.integer(Year), y=ResultValue, group=Year)) +
   geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
          outlier.size=3, outlier.color="#333333",
          outlier.fill="#cccccc", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
     x="Year", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(max(data$Year) - 10.5, max(data$Year)+1),
             breaks=seq(max(data$Year) - 10, max(data$Year), 2)) +
   plot_theme
# Arrange plot objects
set <- ggarrange(p1, p2, p3, ncol=1)
```

```
# Create title object for plots
p0 <- ggplot() + labs(title="Summary Box Plots for Entire Data",
             subtitle="By Year") + plot_theme +
   theme(panel.border=element_blank(), panel.grid.major=element_blank(),
      panel.grid.minor=element_blank(), axis.line=element_blank())
# Arrange title on plots
Yset <- ggarrange(p0, set, ncol=1, heights=c(0.07, 1))
```

This set of box plots are grouped by year and month with the color being related to the month.

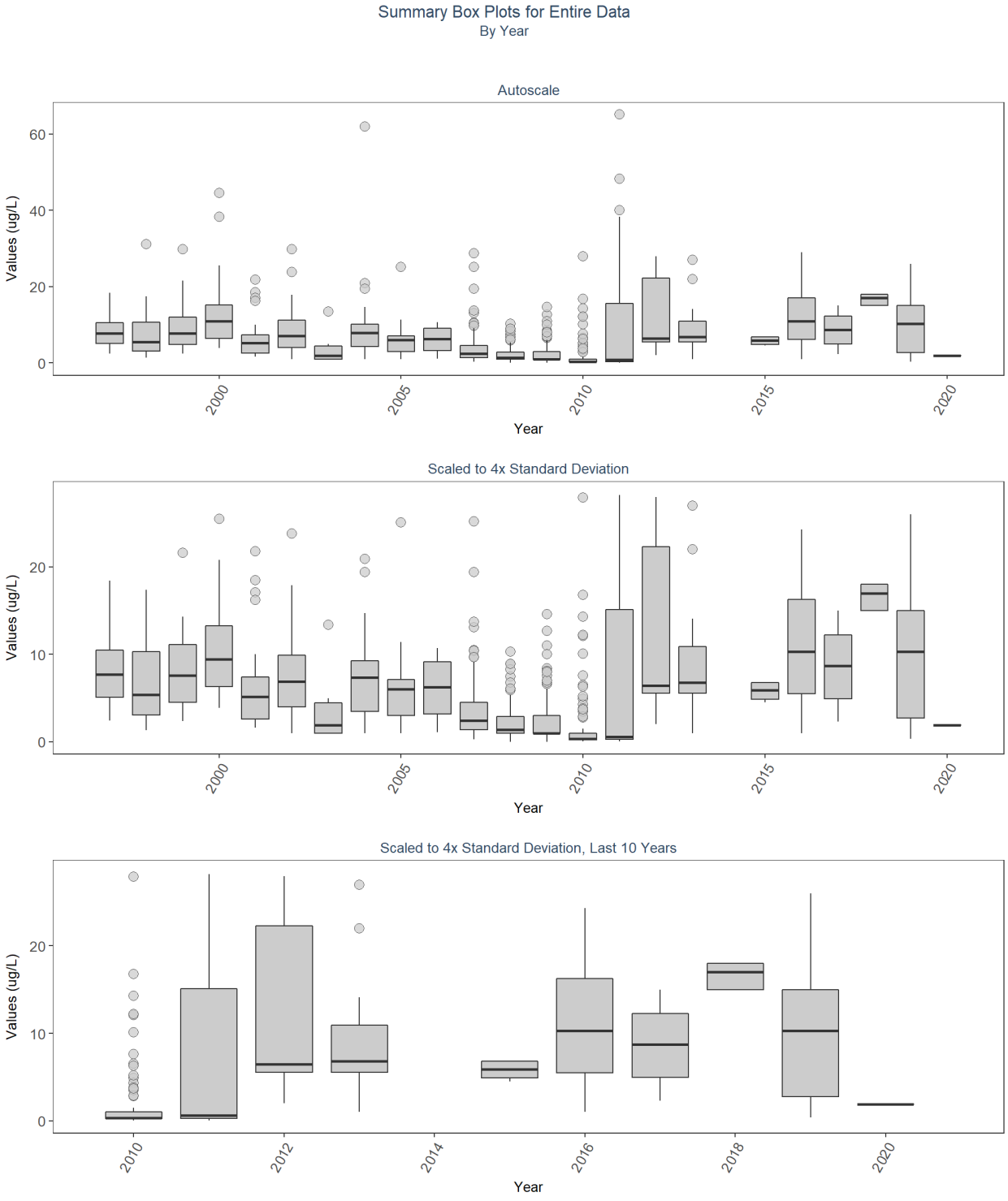
```
# Create plot object for auto-scaled y-axis plot
p1 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Autoscale", x="Year",
     y=paste0("Values (", unit, ")"), color="Month") +
   plot_theme +
   theme(legend.position="top", legend.box="horizontal") +
   guides(color=guide_legend(nrow=1))
# Create plot object for y-axis scaled plot
p2 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation",
     x="Year", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   plot_theme +
   theme(legend.position="none", axis.text.x=element_text(face="bold"),
      axis.text.y=element_text(face="bold"))
# Create plot object for y-axis scaled plot for past 10 years
p3 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
     x="Year", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(max(data$Year) - 10.5, max(data$Year)+1),
             breaks=seq(max(data$Year) - 10, max(data$Year), 2)) +
   plot_theme +
   theme(legend.position="none")
# Create legend item
leg <- get_legend(p1)
# Arrange plots and legend
set <- ggarrange(leg, p1 + theme(legend.position="none"), p2, p3, ncol=1,
           heights=c(0.1, 1, 1, 1))
# Create plot title object
p0 <- ggplot() + labs(title="Summary Box Plots for Entire Data",
             subtitle="By Year & Month") + plot_theme +
   theme(panel.border=element_blank(), panel.grid.major=element_blank(),
      panel.grid.minor=element_blank(), axis.line=element_blank())
```

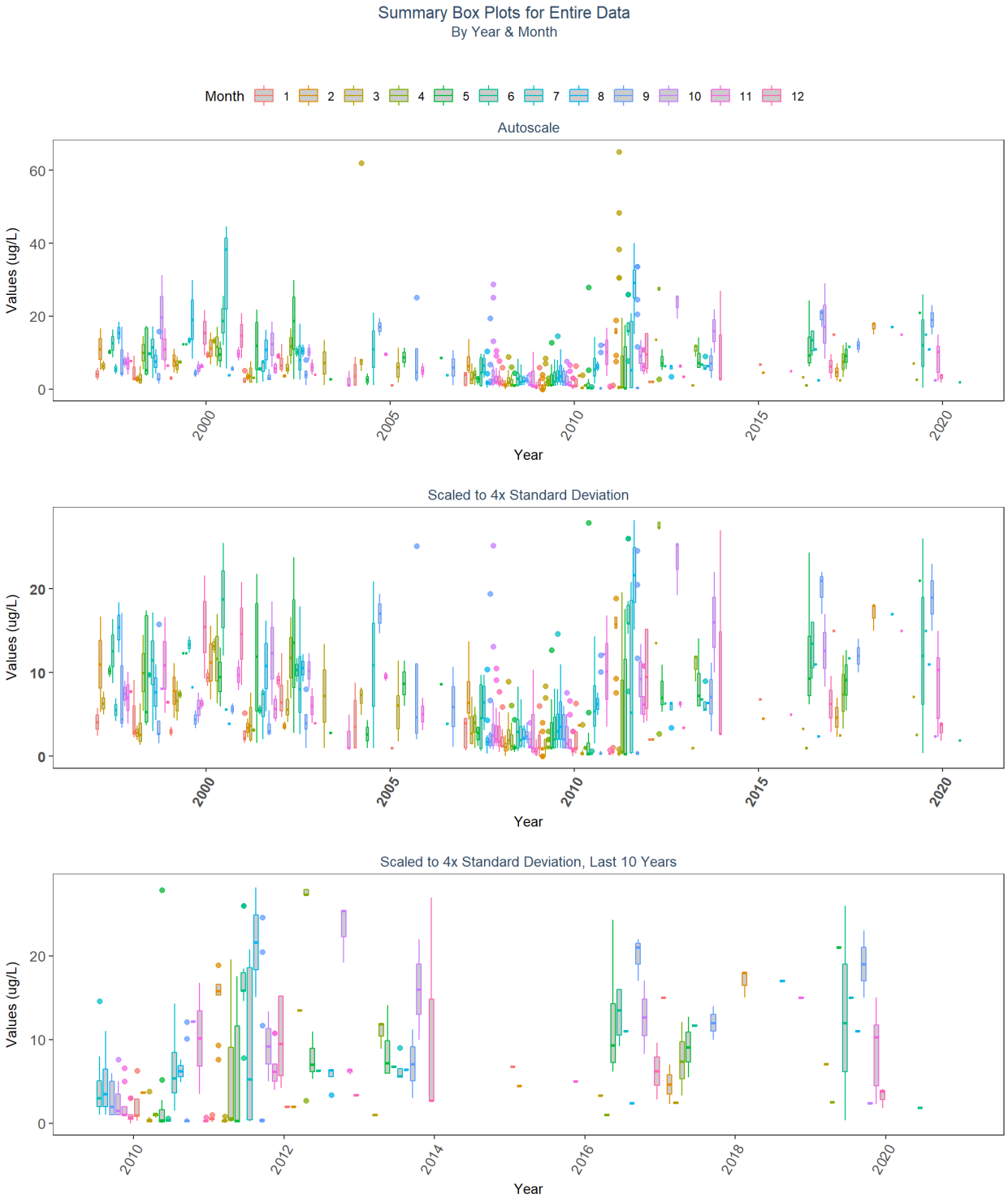
```
# Arrange plots and title
YMset <- ggarrange(p0, set, ncol=1, heights=c(0.07, 1))
```

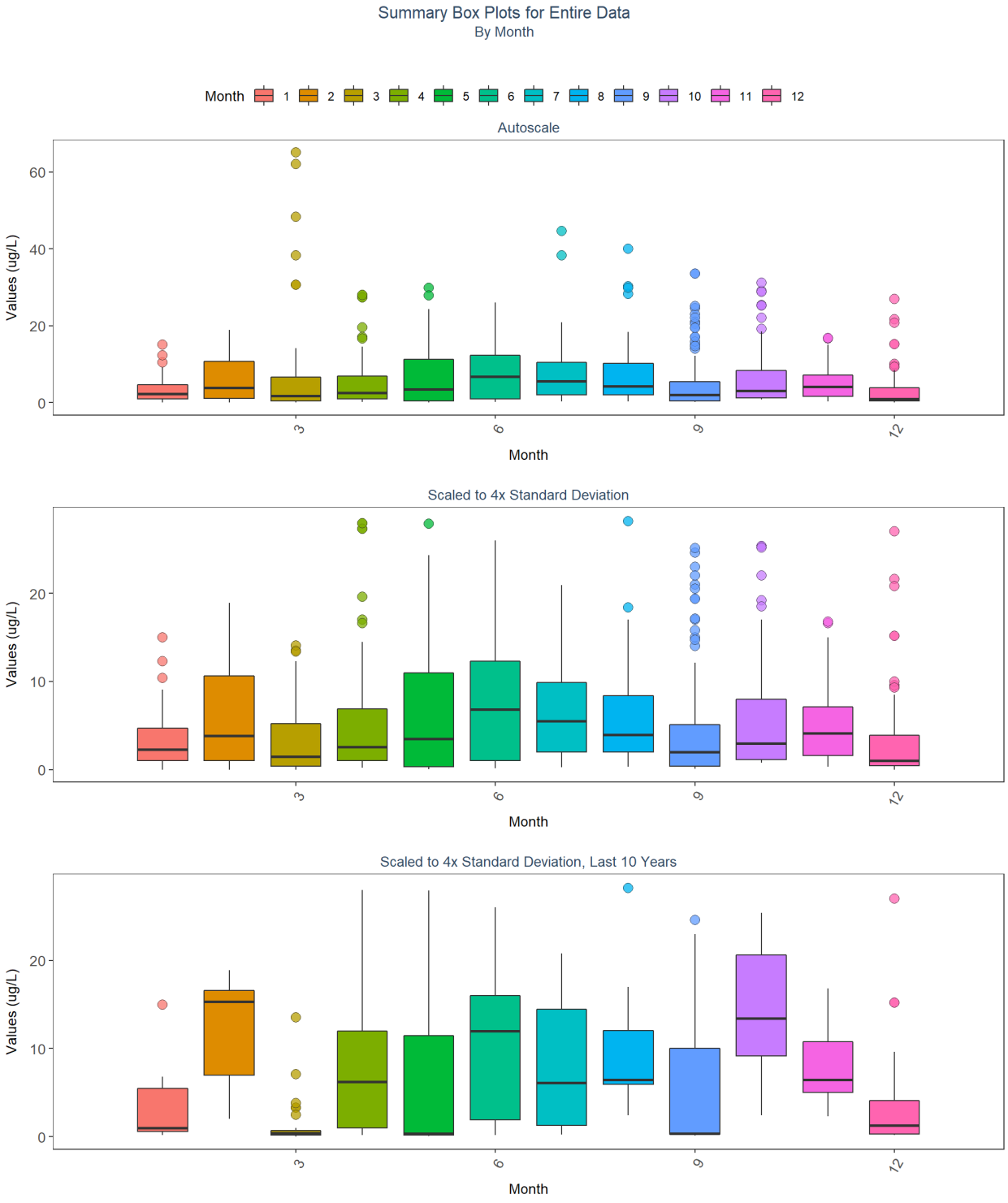
The following box plots are grouped by month with fill color being related to the month. This is designed to view potential seasonal trends.

```
# Create plot object for auto-scaled y-axis plot
p1 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
          outlier.color="#333333", outlier.alpha=0.75) +
   labs(subtitle="Autoscale", x="Month",
     y=paste0("Values (", unit, ")"), fill="Month") +
   scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
   plot_theme +
   theme(legend.position="top", legend.box="horizontal") +
   guides(fill=guide_legend(nrow=1))
# Create plot object for y-axis scaled plot
p2 <- ggplot(data=data[data$Include==TRUE, ],
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
          outlier.color="#333333", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation",
     x="Month", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
   plot_theme +
   theme(legend.position="none")
# Create plot object for y-axis scaled plot for past 10 years
p3 <- ggplot(data=data[data$Include==TRUE &
              data$Year >= max(data$Year) - 10, ],
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
          outlier.color="#333333", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
     x="Month", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
   plot_theme +
   theme(legend.position="none")
# Create legend object
leg <- get_legend(p1)
# Arrange plots and legend
set <- ggarrange(leg, p1 + theme(legend.position="none"), p2, p3, ncol=1,
           heights=c(0.1, 1, 1, 1))
# Create title object for plots
p0 <- ggplot() + labs(title="Summary Box Plots for Entire Data",
             subtitle="By Month") + plot_theme +
   theme(panel.border=element_blank(), panel.grid.major=element_blank(),
      panel.grid.minor=element_blank(), axis.line=element_blank())
```

```
# Arrange plots and title
Mset <- ggarrange(p0, set, ncol=1, heights=c(0.07, 1))
```




Summary Box Plots for Entire Data
By Year & Month
Month 1 2 3 4 5 6 7 8 9 10 11 12
Autoscale
Values (ug/L)
0
20
40
60
2000
2005
2010
2015
2020
Year
Scaled to 4x Standard Deviation
Values (ug/L)
0
10
20
2000
2005
2010
2015
2020
Year
Scaled to 4x Standard Deviation, Last 10 Years
Values (ug/L)
0
10
20
2010
2012
2014
2016
2018
2020
Year



Appendix III: Managed Area Trendlines

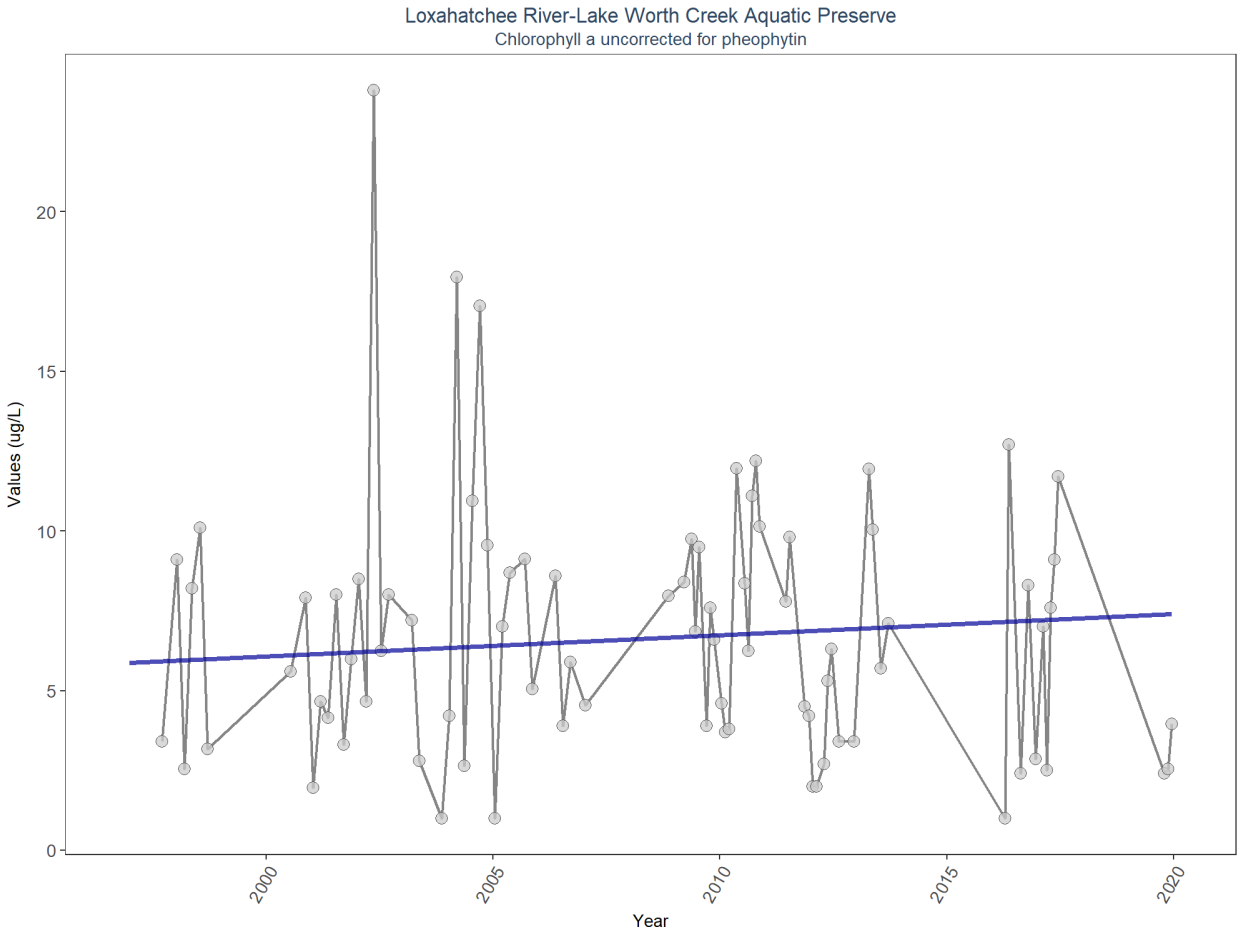
The plots created in this section are designed to show the general trend of the data. Data is taken and grouped by `ManagedAreaName`. The trendlines on the plots are created using the Senn slope and intercept

from the seasonal Kendall Tau analysis. The scripts that create plots follow this format

1. Use the data set that only has `SufficientData` of `TRUE` for the desired managed area
2. Determine the earliest and latest year of the data to create x-axis scale and intervals
3. Determine the minimum, mean, and standard deviation for the data to be used for y-axis scales
    - Excludes the top 2% of values to reduce the impact of extreme outliers on the y-axis scale
4. Set what values are to be used for the x-axis, y-axis, and the variable that should determine groups for the plots
5. Set the plot type as a point plot with the size of the points
6. Add the linear trend
7. Create the title, x-axis, y-axis, and color fill labels
8. Set the y and x limits
9. Make the axis labels bold
10. Plot the arrangement as a set of panels

```
# Determines whether analyzed managed areas exist. If they do, begins
# looping through them
if(n==0){
   print("There are no managed areas that qualify.")
} else {
   # Begins looping through each managed area
   for (i in 1:n) {
   # Gets data to be used in plot for managed area
   plot_data <- MA_YM_Stats[MA_YM_Stats$ManagedAreaName==MA_Include[i],]
   # Gets trendline data for managed area
   KT.plot_data <- KT.Plot[KT.Plot$ManagedAreaName==MA_Include[i],]
   #Determine max and min time (Year) for plot x-axis
   t_min <- min(plot_data$Year)
   t_max <- max(plot_data$YearMonthDec)
   t_max_brk <- as.integer(round(t_max, 0))
   t <- t_max-t_min
   min_RV <- min(plot_data$Mean)

   # Sets break intervals based on the number of years spanned by data
   if(t>=30){
      brk <- -10
   }else if(t<30 & t>=10){
      brk <- -5
   }else if(t<10 & t>=4){
      brk <- -2
   }else if(t<4){
      brk <- -1
   }

   # Create plot object with data and trendline
   p1 <- ggplot(data=plot_data,
          aes(x=YearMonthDec, y=Mean)) +
      geom_line(size=0.75, color="#333333", alpha=0.6) +
      geom_point(shape=21, size=3, color="#333333", fill="#cccccc",
           alpha=0.75) +
      geom_line(data=KT.plot_data, aes(x=x, y=y),
          color="#000099", size=1.2, alpha=0.7) +
      labs(title=paste0(MA_Include[i]),
```

```
        subtitle=parameter,
        x="Year", y=paste0("Values (", unit, ")")) +
      scale_x_continuous(limits=c(t_min-0.25, t_max+0.25),
                breaks=seq(t_max_brk, t_min, brk)) +
      plot_theme
   # Creates ResultTable to display statistics below plot
   ResultTable <- KT.Stats[KT.Stats$ManagedAreaName==MA_Include[i], ] %>%
      select(RelativeDepth, N_Data, N_Years, Median, Independent, tau, p,
          SennSlope, SennIntercept, ChiSquared, pChiSquared, Trend)
   # Create table object
   t1 <- ggtexttable(ResultTable, rows=NULL,
            theme=ttheme(base_size=10)) %>%
      tab_add_footnote(text="p < 0.00005 appear as 0 due to rounding.\n
              SennIntercept is intercept value at beginning of
              record for monitoring location",
              size=10, face="italic")
   # Arrange and display plot and statistic table
   print(ggarrange(p1, t1, ncol=1, heights=c(0.85, 0.15)))
   # Add extra space at the end to prevent the next figure from being too
   # close.
   cat("\n \n \n")
   rm(plot_data)
   rm(KTset, leg)
   rm(plot_data)
   rm(KTset, leg)
   }
}
```



RelativeDepth	N_Data	N_Years	Median	Independent	tau	p	SennSlope	SennIntercept	ChiSquared	pChiSquared	Trend
Bottom	143	19	5.6	TRUE	0.0177	0.4085	0.06666667	5.865491	8.8583	0.635	0

p < 0.00005 appear as 0 due to rounding.

SennIntercept is intercept value at beginning of record for monitoring location

Appendix IV: Managed Area Summary Box Plots

Data is taken and grouped by `ManagedAreaName`. The scripts that create plots follow this format

1. Use the data set that only has `SufficientData` of `TRUE` for the desired managed area
2. Determine the earliest and latest year of the data to create x-axis scale and intervals
3. Determine the minimum, mean, and standard deviation for the data to be used for y-axis scales
    - Excludes the top 2% of values to reduce the impact of extreme outliers on the y-axis scale
4. Set what values are to be used for the x-axis, y-axis, and the variable that should determine groups for the box plots
5. Set the plot type as a box plot with the size of the outlier points
6. Create the title, x-axis, y-axis, and color fill labels
7. Set the y and x limits
8. Make the axis labels bold
9. Plot the arrangement as a set of panels

The following plots are arranged by `ManagedAreaName` with data grouped by `Year`, then `Year` and `Month`, then finally `Month` only. Each managed area will have 3 sets of plots, each with 3 panels in them. Each panel goes as follows:

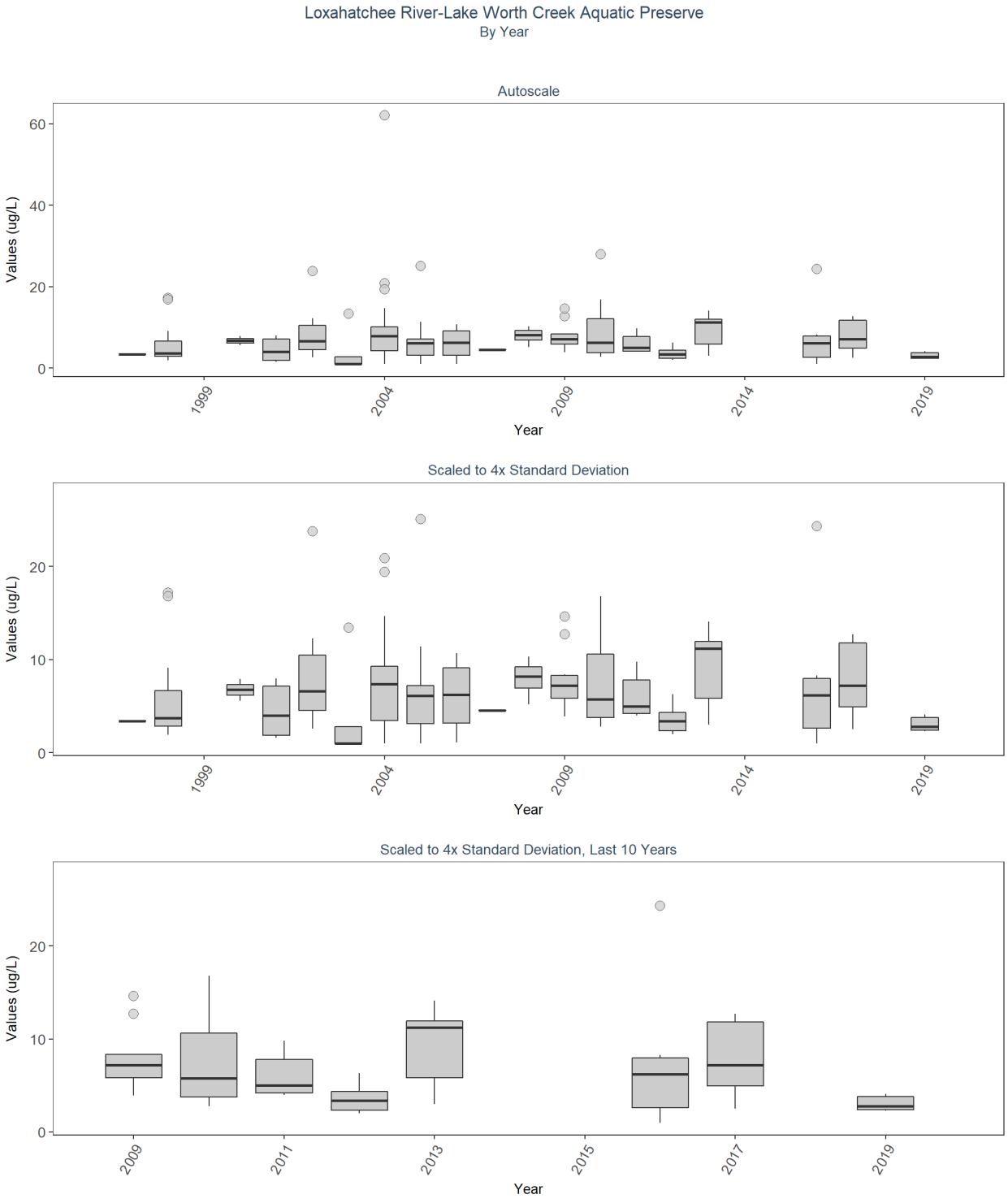
1. Y-axis autoscaled
2. Y-axis set to be mean + 4 times the standard deviation
3. Y-axis set to be mean + 4 times the standard deviation for most recent 10 years of data

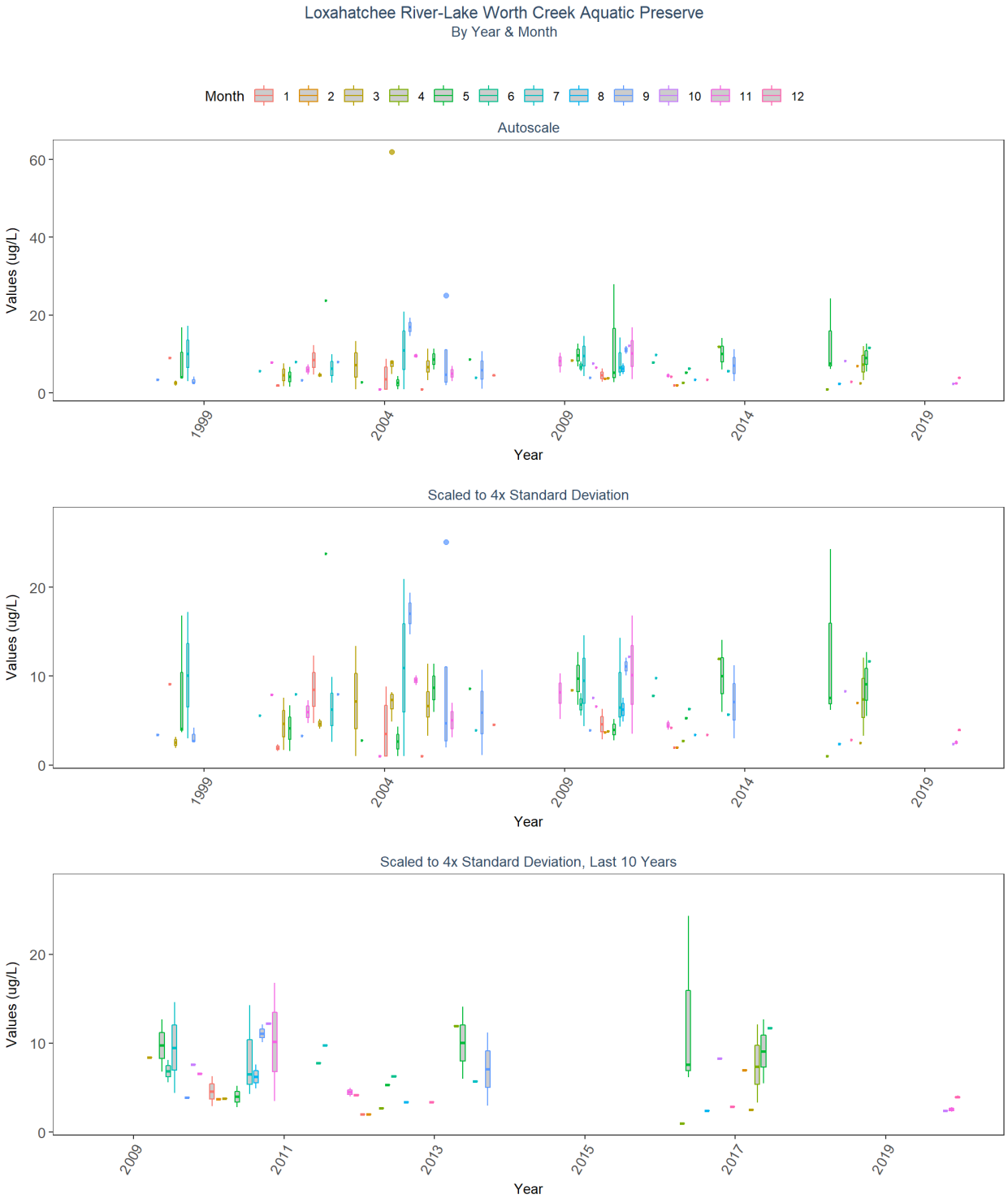
```
# Determines whether analyzed managed area exist. If they do, begins
# looping through them
if(n==0){
   print("There are no managed areas that qualify.")
} else {
   # Begin looping through managed area
   for (i in 1:n) {
   # Determine upper and lower bounds of time for x-axis
   plot_data <- data[data$SufficientData==TRUE &
               data$ManagedAreaName==MA_Include[i],]
   year_lower <- min(plot_data$Year)
   year_upper <- max(plot_data$Year)
   # Determine upper and lower bounds of ResultValue for y-axis
   min_RV <- min(plot_data$ResultValue)
   mn_RV <- mean(plot_data$ResultValue[plot_data$ResultValue <
                       quantile(data$ResultValue, 0.98)])
   sd_RV <- sd(plot_data$ResultValue[plot_data$ResultValue <
                     quantile(data$ResultValue, 0.98)])
   # Sets x- and y-axis scale
   x_scale <- ifelse(year_upper - year_lower > 30, 10, 5)
   y_scale <- mn_RV + 4 * sd_RV

   ##Year plots
   # Create plot object for auto-scaled y-axis plot
   p1 <- ggplot(data=plot_data,
          aes(x=Year, y=ResultValue, group=Year)) +
      geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
             outlier.size=3, outlier.color="#333333",
             outlier.fill="#cccccc", outlier.alpha=0.75) +
      labs(subtitle="Autoscale",
        x="Year", y=paste0("Values (", unit, ")")) +
      scale_x_continuous(limits=c(year_lower - 1, year_upper + 1),
                breaks=rev(seq(year_upper,
                      year_lower, -x_scale))) +
      plot_theme
   # Create plot object for y-axis scaled plot
   p2 <- ggplot(data=plot_data,
          aes(x=Year, y=ResultValue, group=Year)) +
      geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
             outlier.size=3, outlier.color="#333333",
             outlier.fill="#cccccc", outlier.alpha=0.75) +
      labs(subtitle="Scaled to 4x Standard Deviation",
        x="Year", y=paste0("Values (", unit, ")")) +
      ylim(min_RV, y_scale) +
      scale_x_continuous(limits=c(year_lower - 1, year_upper + 1),
                breaks=rev(seq(year_upper,
```

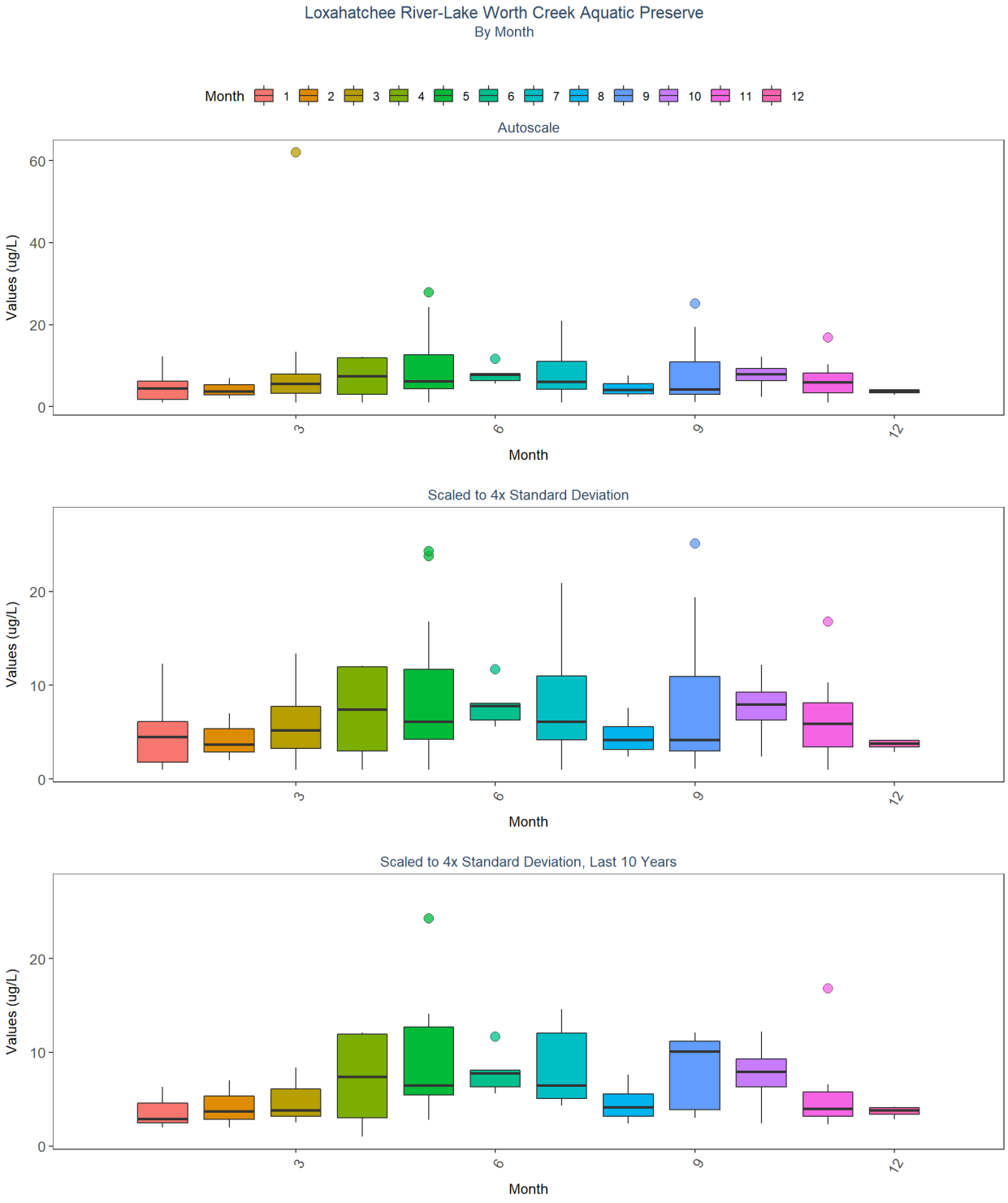
```
                    year_lower, -x_scale))) +
   plot_theme
# Create plot object for y-axis scaled plot for past 10 years
p3 <- ggplot(data=plot_data[plot_data$Year >= year_upper - 10, ],
       aes(x=Year, y=ResultValue, group=Year)) +
   geom_boxplot(color="#333333", fill="#cccccc", outlier.shape=21,
          outlier.size=3, outlier.color="#333333",
          outlier.fill="#cccccc", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
     x="Year", y=paste0("Values (", unit, ")")) +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(year_upper - 10.5, year_upper + 1),
             breaks=rev(seq(year_upper, year_upper - 10,-2))) +
   plot_theme
# Arrange plot objects
Yset <- ggarrange(p1, p2, p3, ncol=1)
# Create plot title object
p0 <- ggplot() + labs(title=paste0(MA_Include[i]),
             subtitle="By Year") +
   plot_theme + theme(panel.border=element_blank(),
             panel.grid.major=element_blank(),
             panel.grid.minor=element_blank(),
             axis.line=element_blank())


## Year & Month Plots
# Create plot object for auto-scaled y-axis plot
p4 <- ggplot(data=plot_data,
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Autoscale",
     x="Year", y=paste0("Values (", unit, ")"), color="Month") +
   scale_x_continuous(limits=c(year_lower - 1, year_upper + 1),
             breaks=rev(seq(year_upper,
                    year_lower, -x_scale))) +
   plot_theme +
   theme(legend.position="none")
# Create plot object for y-axis scaled plot
p5 <- ggplot(data=plot_data,
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation",
     x="Year", y=paste0("Values (", unit, ")"), color="Month") +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(year_lower - 1, year_upper + 1),
             breaks=rev(seq(year_upper,
                    year_lower, -x_scale))) +
   plot_theme +
   theme(legend.position="top", legend.box="horizontal") +
   guides(color=guide_legend(nrow=1))
# Create plot object for y-axis scaled plot for past 10 years
```

```
p6 <- ggplot(data=plot_data[plot_data$Year >= year_upper - 10, ],
       aes(x=YearMonthDec, y=ResultValue,
           group=YearMonth, color=as.factor(Month))) +
   geom_boxplot(fill="#cccccc", outlier.size=1.5, outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
     x="Year", y=paste0("Values (", unit, ")"), color="Month") +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(year_upper - 10.5, year_upper + 1),
            breaks=rev(seq(year_upper, year_upper - 10,-2))) +
   plot_theme +
   theme(legend.position="none")
# Create legend object
leg1 <- get_legend(p5)
# Arrange plots and legend
YMset <- ggarrange(leg1, p4, p5 + theme(legend.position="none"), p6,
          ncol=1, heights=c(0.1, 1, 1, 1))
# Create plot title object
p00 <- ggplot() + labs(title=paste0(MA_Include[i]),
              subtitle="By Year & Month") + plot_theme +
   theme(panel.border=element_blank(),
      panel.grid.major=element_blank(),
      panel.grid.minor=element_blank(), axis.line=element_blank())

## Month Plots
# Create plot object for auto-scaled y-axis plot
p7 <- ggplot(data=plot_data,
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
          outlier.color="#333333", outlier.alpha=0.75) +
   labs(subtitle="Autoscale",
     x="Month", y=paste0("Values (", unit, ")"), fill="Month") +
   scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
   plot_theme +
   theme(legend.position="none")
# Create plot object for y-axis scaled plot
p8 <- ggplot(data=plot_data,
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
          outlier.color="#333333", outlier.alpha=0.75) +
   labs(subtitle="Scaled to 4x Standard Deviation",
     x="Month", y=paste0("Values (", unit, ")"), fill="Month") +
   ylim(min_RV, y_scale) +
   scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
   plot_theme +
   theme(legend.position="top", legend.box="horizontal") +
   guides(fill=guide_legend(nrow=1))
# Create plot object for y-axis scaled plot for past 10 years
p9 <- ggplot(data=plot_data[plot_data$Year >= year_upper - 10, ],
       aes(x=Month, y=ResultValue,
           group=Month, fill=as.factor(Month))) +
   geom_boxplot(color="#333333", outlier.shape=21, outlier.size=3,
```

```
            outlier.color="#333333", outlier.alpha=0.75) +
      labs(subtitle="Scaled to 4x Standard Deviation, Last 10 Years",
        x="Month", y=paste0("Values (", unit, ")"), fill="Month") +
      ylim(min_RV, y_scale) +
      scale_x_continuous(limits=c(0, 13), breaks=seq(3, 12, 3)) +
      plot_theme +
      theme(legend.position="none")
   # Create legend object
   leg2 <- get_legend(p8)
   # Arrange plots and legend
   Mset <- ggarrange(leg2, p7, p8 + theme(legend.position="none"), p9,
            ncol=1, heights=c(0.1, 1, 1, 1))
   # Create title object
   p000 <- ggplot() + labs(title=paste0(MA_Include[i]),
               subtitle="By Month") + plot_theme +
      theme(panel.border=element_blank(),
         panel.grid.major=element_blank(),
         panel.grid.minor=element_blank(), axis.line=element_blank())
   # Arrange and display plots with titles for all combinations
   print(ggarrange(p0, Yset, ncol=1, heights=c(0.07, 1)))
   print(ggarrange(p00, YMset, ncol=1, heights=c(0.07, 1)))
   print(ggarrange(p000, Mset, ncol=1, heights=c(0.07, 1, 0.7)))

   rm(plot_data)
   rm(p1, p2, p3, p4, p5, p6, p7, p8, p9, p0, p00, p000, leg1, leg2,
      Yset, YMset, Mset)
   }
}
```


Loxahatchee River-Lake Worth Creek Aquatic Preserve
By Year
Autoscale
Values (ug/L)
0
20
40
60
1999
2004
2009
2014
2019
Year
Scaled to 4x Standard Deviation
Values (ug/L)
0
10
20
1999
2004
2009
2014
2019
Year
Scaled to 4x Standard Deviation, Last 10 Years
Values (ug/L)
0
10
20
2009
2011
2013
2015
2017
2019
Year

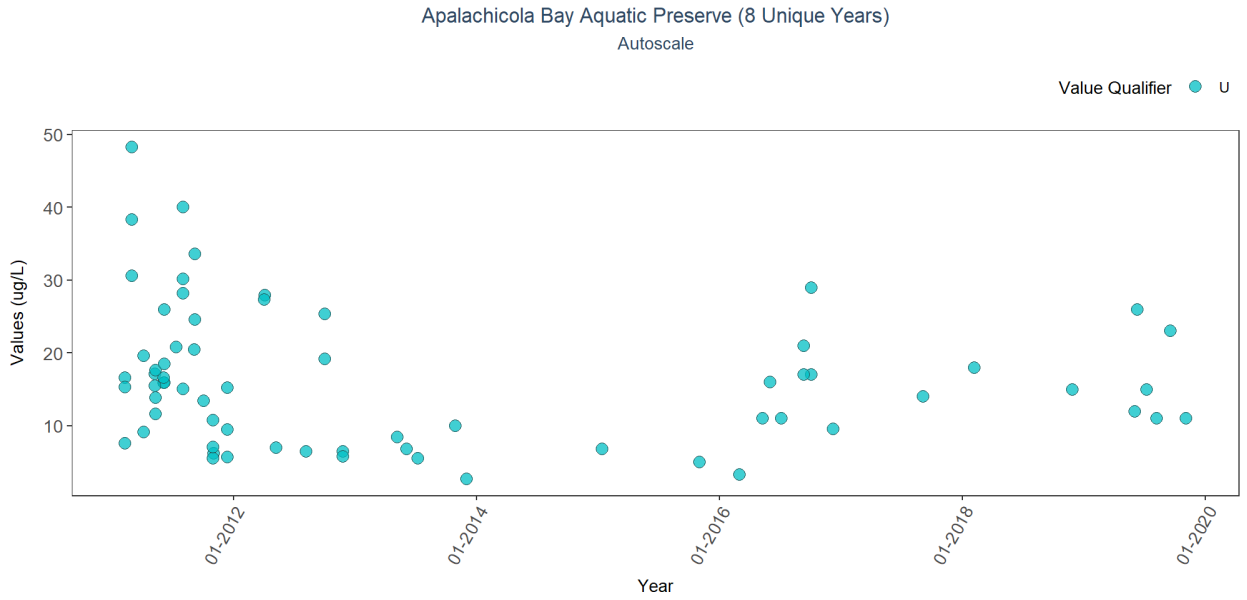

Loxahatchee River-Lake Worth Creek Aquatic Preserve
By Year & Month
Month 1 2 3 4 5 6 7 8 9 10 11 12
Autoscale
Values (ug/L)
Year
Scaled to 4x Standard Deviation
Values (ug/L)
Year
Scaled to 4x Standard Deviation, Last 10 Years
Values (ug/L)
Year

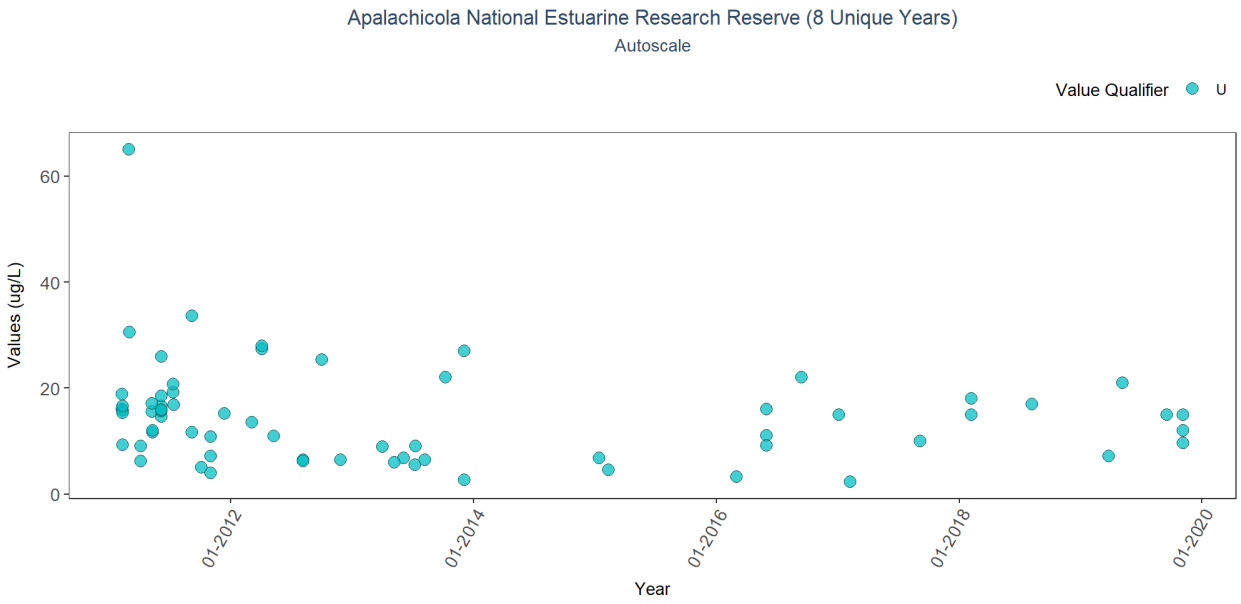


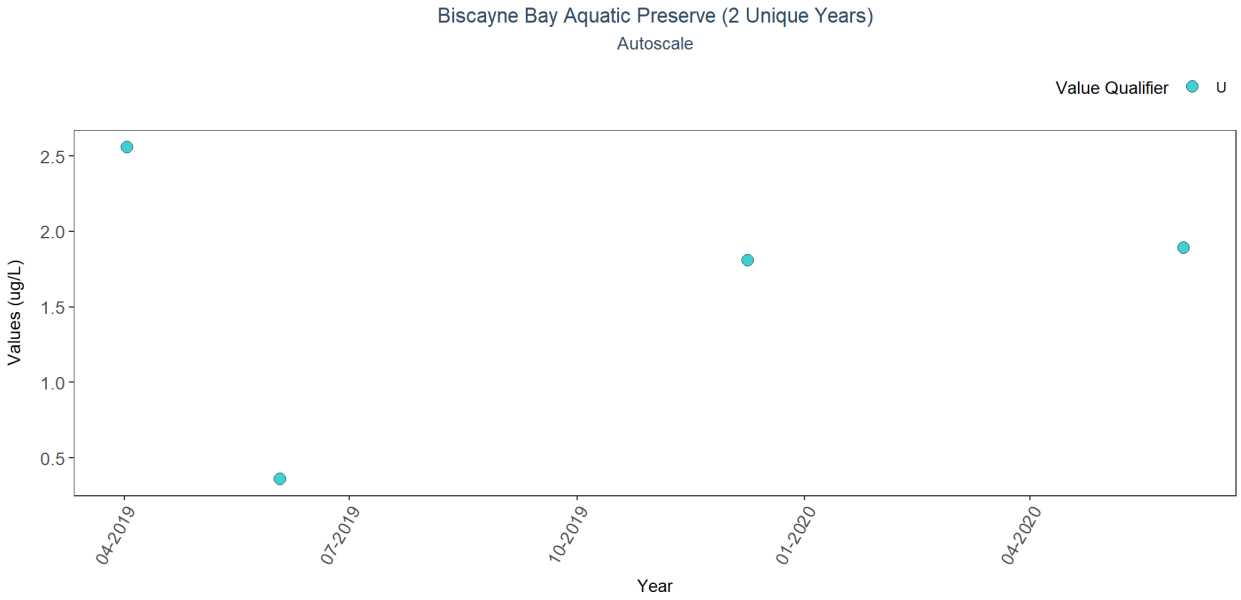
Appendix V: Excluded Managed Areas

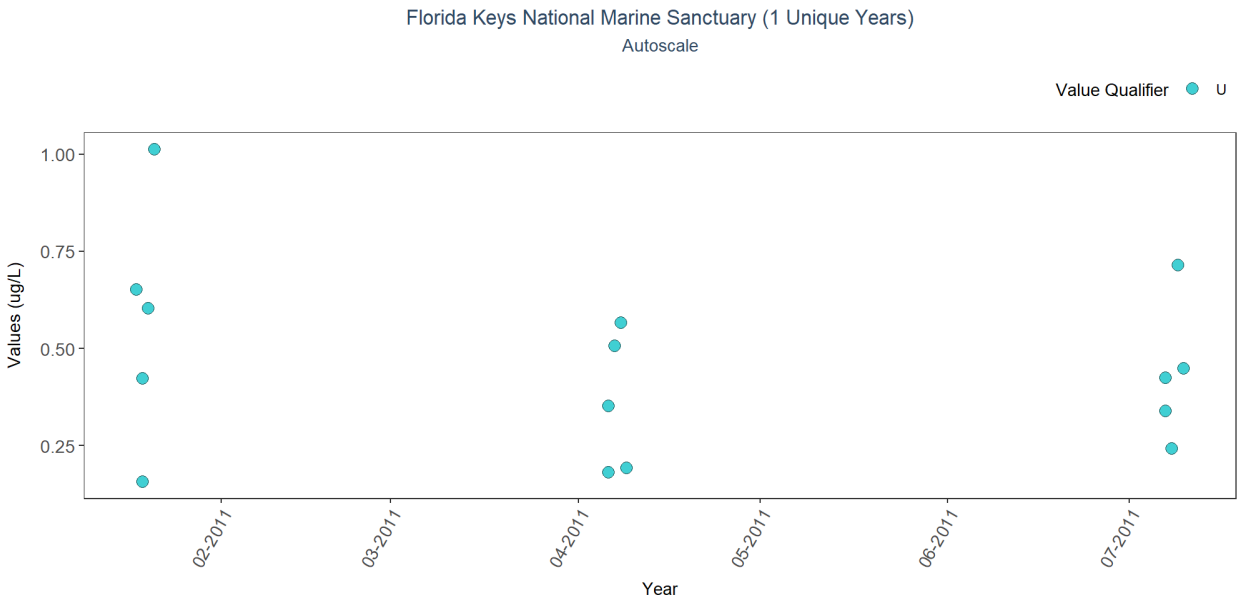
Scatter plots of data values are created for managed areas that have fewer than 10 separate years of data entries. Data points are colored based on specific value qualifiers of interest.

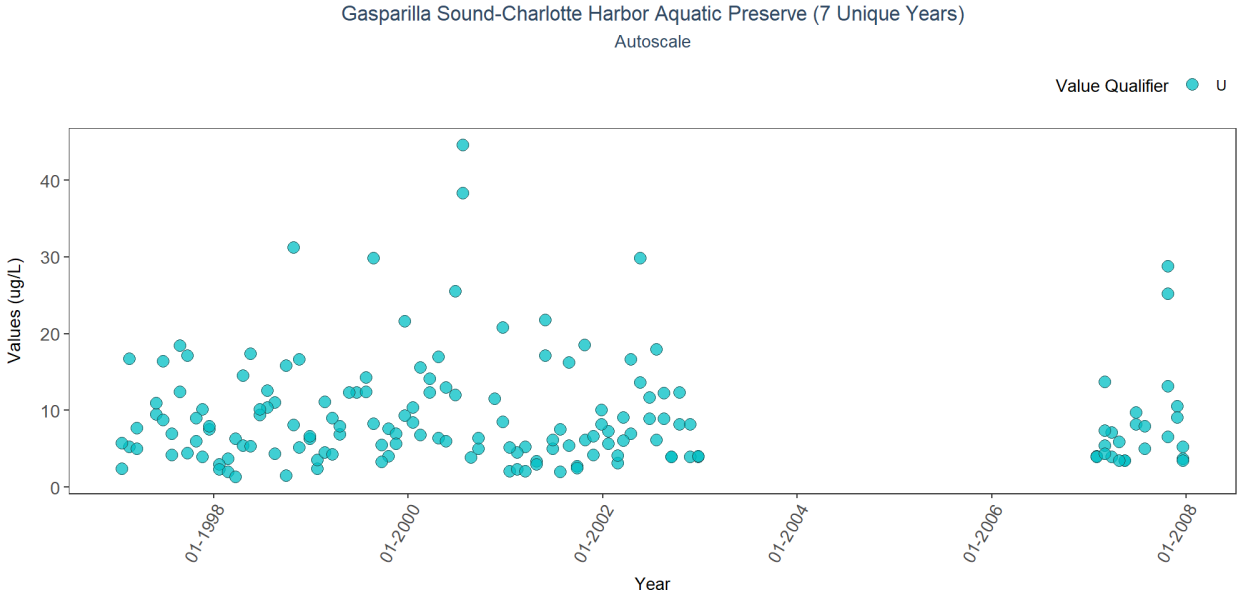
```
# Determines whether excluded managed areas exist. If they do, begins
# looping through them
if(z==0){
   print("There are no managed areas that qualify.")
} else {
   for(i in 1:z){
   # Create scatter plot with data
   p1<-ggplot(data=data[data$ManagedAreaName==MA_Exclude$ManagedAreaName[i]&
              data$Include==TRUE, ],
          aes(x=SampleDate, y=ResultValue, fill=VQ_Plot)) +
      geom_point(shape=21, size=3, color="#333333", alpha=0.75) +
      labs(title=paste0(MA_Exclude$ManagedAreaName[i], " (",
              MA_Exclude$N_Years[i], " Unique Years)"),
        subtitle="Autoscale", x="Year",
        y=paste0("Values (", unit, ")"), fill="Value Qualifier") +
      plot_theme +
      theme(legend.position="top", legend.box="horizontal",
         legend.justification="right") +
      scale_x_date(labels=date_format("%m-%Y")) +
      {if(inc_H==TRUE){
      scale_fill_manual(values=c("H"= "#F8766D", "U"= "#00BFC4",
                      "HU"="#7CAE00"), na.value="#cccccc")
      } else if(param_name=="Secchi_Depth"){
      scale_fill_manual(values=c("S"= "#F8766D", "U"= "#00BFC4",
                      "SU"="#7CAE00"), na.value="#cccccc")
      } else {
      scale_fill_manual(values=c("U"= "#00BFC4"), na.value="#cccccc")
      }}
   print(p1)
   }
}
```

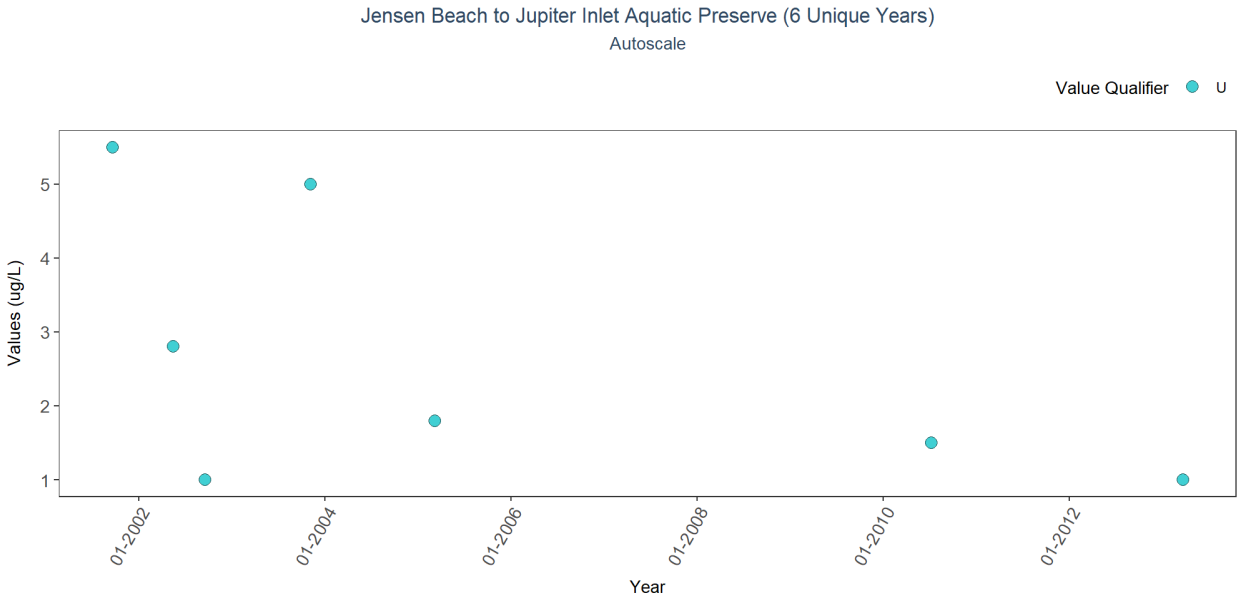


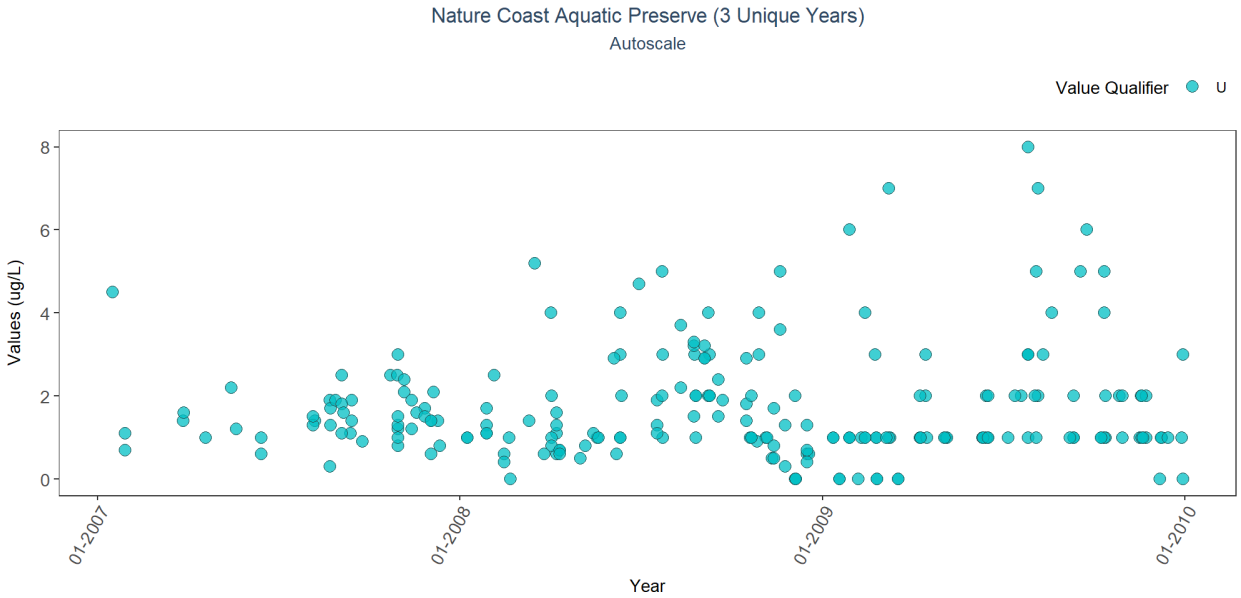

Apalachicola National Estuarine Research Reserve (8 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
60
40
20
0
01-2012
01-2014
01-2016
01-2018
01-2020
Year

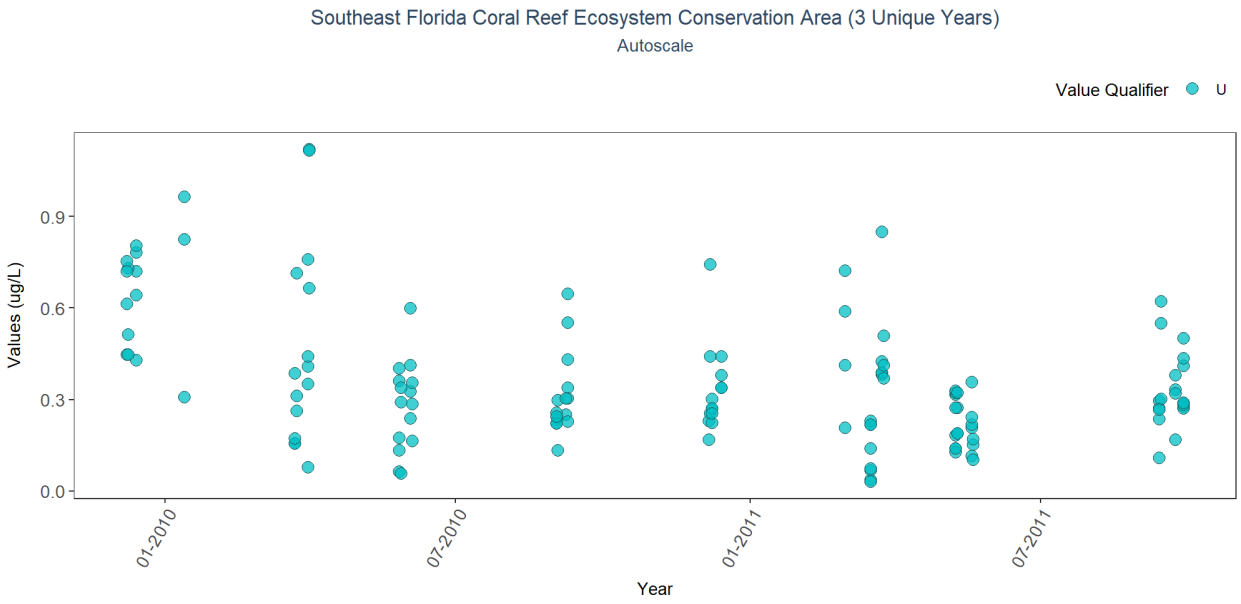

Biscayne Bay Aquatic Preserve (2 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
2.5
2.0
1.5
1.0
0.5
04-2019
07-2019
10-2019
01-2020
04-2020
Year

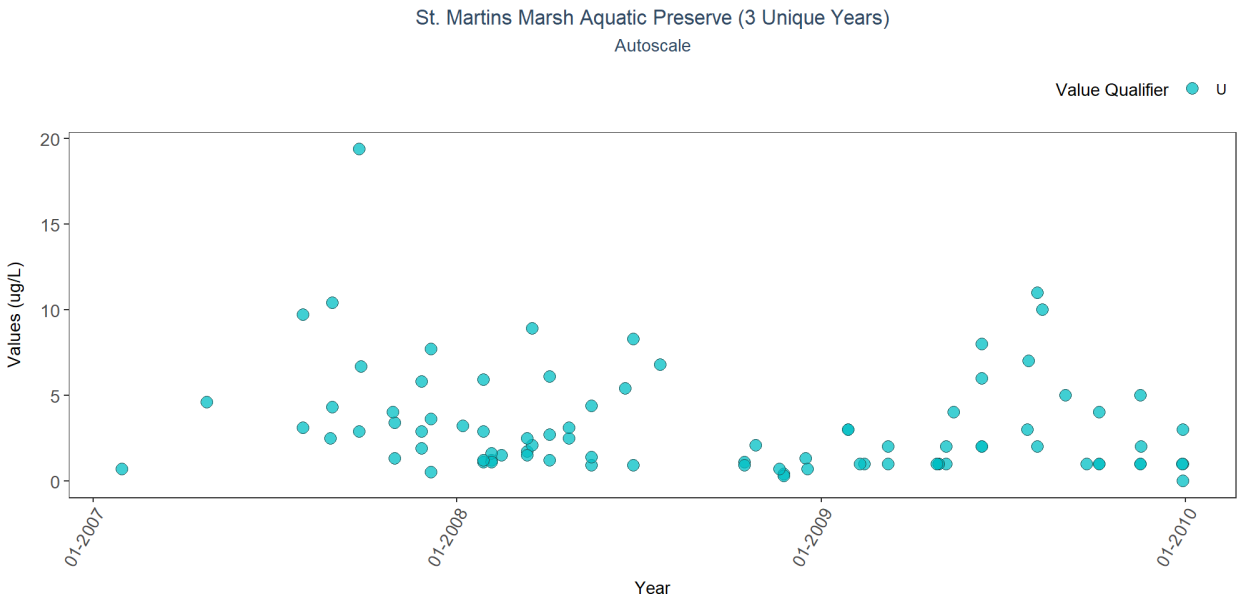

Florida Keys National Marine Sanctuary (1 Unique Years)
Autoscale
Value Qualifier
U
1.00
0.75
0.50
0.25
Values (ug/L)
02-2011
03-2011
04-2011
05-2011
06-2011
07-2011
Year


Gasparilla Sound-Charlotte Harbor Aquatic Preserve (7 Unique Years)
Autoscale
Value Qualifier
U
40
30
20
10
0
Values (ug/L)
01-1998
01-2000
01-2002
01-2004
01-2006
01-2008
Year


Jensen Beach to Jupiter Inlet Aquatic Preserve (6 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
5
4
3
2
1
01-2002
01-2004
01-2006
01-2008
01-2010
01-2012
Year


Nature Coast Aquatic Preserve (3 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
8
6
4
2
0
01-2007
01-2008
01-2009
01-2010
Year


Southeast Florida Coral Reef Ecosystem Conservation Area (3 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
0.0
0.3
0.6
0.9
01-2010
07-2010
01-2011
07-2011
Year


St. Martins Marsh Aquatic Preserve (3 Unique Years)
Autoscale
Value Qualifier
U
Values (ug/L)
0
5
10
15
20
01-2007
01-2008
01-2009
01-2010
Year