Data Processing User Guide

Tetra Tech

## Required Packages

The required packages for the data\_pull.R code are *TADA*, *tidyverse, leaflet, scales,* and *sf.* Installation for the *TADA* and *tidyverse* packages can be found in the Data Pull Guide. Here are the lines to install and load these packages:

# install.packages('leaflet')  
library(leaflet)  
  
# install.packages('scales')  
library(scales)  
  
# install.packages('sf')  
library(sf)  
  
library(TADA)  
library(sf)

## Required Inputs

This code has specific exterior files that are required in order for it to run. These inputs are not all loaded in at the beginning, but instead are read in when they are needed. The inputs are as follows:

1. Csv outputs from data\_pull.R - broken up by site type
2. ‘WQ\_Column\_Manager.csv’ to quickly subset WQ dataset fields
3. ‘ML\_AU\_Crosswalk.csv’ to crosswalk Monitoring Locations with AUs
4. ‘AK\_DataSufficiency\_Crosswalk\_20231012.csv’ to crosswalk WQ dataset with data sufficiency table

## Load Data

The data processing in this file is being applied to the csv outputs from data\_pull.R. The following lines read in the csvs broken up by site type and purposefully excluding the ‘data\_pull\_all.csv’ to allow for this code to be used for different combinations of files:

#Find all file names that end in .csv from the data\_pull output folder  
csv\_names1 <- list.files('Data/data\_pull', pattern = '.csv', full.names = T)  
csv\_names <- csv\_names1[!str\_detect(csv\_names1, pattern = 'all')]  
   
#Read in csvs and combine into one table  
all\_input\_data <- tibble()  
for(i in 1:length(csv\_names)) {  
 csv <- read\_csv(csv\_names[i])  
 all\_input\_data <- all\_input\_data %>%  
 rbind(csv)  
 remove(csv)  
}

## Section 1: Identifying TADA Flags

After the initial setup and data read, the data processing is broken up into numbered steps that are divided into 5 different sections. Steps #1 through #13 are in the ‘Identify Flags’ section. These steps are made up of different *TADA* functions that scan the input dataset for a given error or lack thereof and produce a new column or columns with the prefix ‘TADA’ that identifies the flag for each row. The comments under each step name list the column names that the *TADA* function adds to the data. Here is the example for Step #1:

#####1. Check Result Unit Validity#####  
# This function adds the TADA.ResultUnit.Flag to the dataframe.  
data\_1 <- TADA\_FlagResultUnit(all\_input\_data, clean = 'none')

The specifics for each TADA function can be found using the ‘?’ function in R or by reading through the [Module 1 TADA vignette](https://github.com/USEPA/TADA/blob/develop/vignettes/TADAModule1.Rmd). The following functions are used to flag potential issues in the data:

1. TADA\_FlagResultUnit: checks for errors in units
2. TADA\_FlagFraction: checks for invalid characteristic-fraction combinations
3. TADA\_FlagSpeciation: checks for invalid characteristic-method speciation
4. TADA\_HarmonizeSynonyms: checks for duplicate naming and assigns a name based on a synonym reference table
5. TADA\_FlagAboveThreshold/TADA\_FlagBelowThreshold: checks for values above or below the threshold for that ‘CharacteristicName’
6. TADA\_FindContinuousData: checks metadata to flag any potential aggregated continuous data submitted to WQP
7. TADA\_FlagMethod: checks for invalid characteristic-analytical method combinations
8. TADA\_FindpotentialDuplicatesMultipleOrgs/TADA\_FindPotentialDuplicatesSingleOrg: checks for duplicate samples within other organizations/within the same organization
9. TADA\_FindQCActivities: identifies QC samples
10. TADA\_FlagCoordinates: checks for coordinates outside of the United States
11. TADA\_FlagMeasureQualifierCode: checks the ‘MeasureQualifierCode’ for any known suspect codes
12. TADA\_SimpleCensoredMethods: determines if a sample is non-detect and if it needs to reassigned - currently set to assign non-detects to 0.5 the detection limit

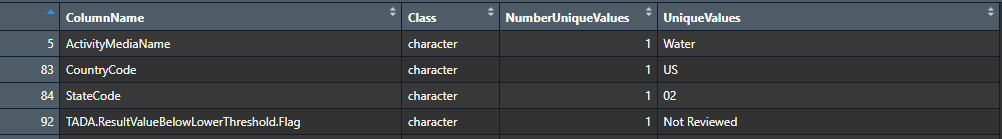
Step #11 is reliant on a list of known codes from AKDEC that the TADA package cannot identify. The codes from AK can be found in ‘IR Data QA .xlsx’. Any further unknown qualifier codes are labeled as ‘uncategorized’. After these functions are applied Step #13 removes any columns that are entirely NA from the dataset.

#####13. Identify columns with all NA values#####  
# Check whether you expect data in any of the columns listed below.  
(cols\_NA <- data\_12 %>%   
 keep(~all(is.na(.x))) %>%   
 names)  
  
# Eliminate any columns with all NA values  
data\_13 <- data\_12 %>%   
 select(where(~sum(!is.na(.x)) > 0))

## Section 2: Evaluate and Trim Data

The second section aims to use the flags from Section 1 to remove data. This section is composed of Steps #14 through #18. The following is a breakdown of each step:

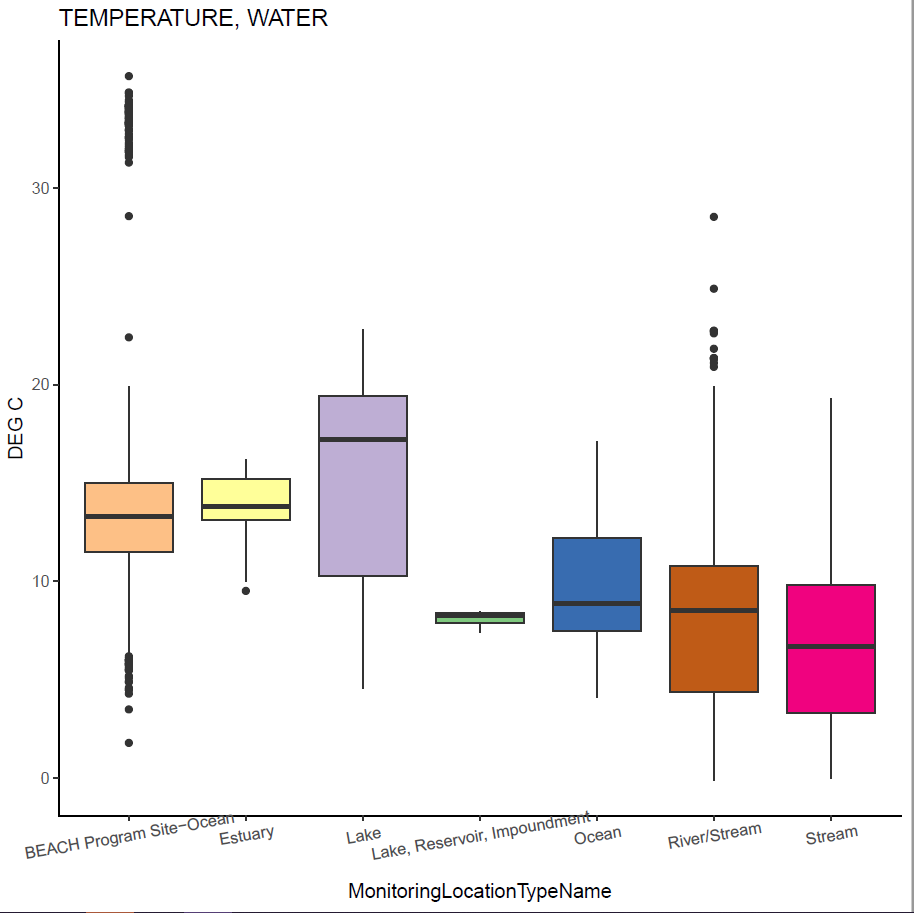
1. Create a data summary table with each row being a column from the Step #13 output in Section 1. The columns include the column name, class, number of unique values, and a list of those unique values if the list is 10 or less in length. The data\_summary table provides the values for data processing and removal.

* 

1. The WQ\_Column\_Manager.csv is read in and the ‘Keep\_YN’ column is used to filter out unnecessary columns. If the columns in the manager csv do not match those from the output of Step #13, an error message is printed instead.
2. Using the data\_summary table values, flags with values that need to be removed are filtered out. Some *TADA* outputs do not match their vignettes and so the following is assumed:

* # Assume the following:  
  # Not Reviewed <- "Not Reviewed"   
  # Valid <- c("Accepted", "Y")  
  # Invalid <- c("Rejected", "Rejected ", "N")  
  # NonStandardized <- c("NonStandardized",  
  # "InvalidMediaUnit",  
  # "InvalidChar",  
  # "MethodNeeded")  
    
  data\_16 <- data\_15 %>%   
   filter(TADA.ResultUnit.Flag != "Rejected") %>% # Step 1  
   filter(TADA.SampleFraction.Flag != "Rejected") %>% # Step 2  
   filter(TADA.MethodSpeciation.Flag != "Rejected") %>% # Step 3  
   filter(TADA.AnalyticalMethod.Flag != "Rejected") %>% # Step 7  
   filter(TADA.ActivityType.Flag == 'Non\_QC') %>% # Step 9  
   filter(TADA.MeasureQualifierCode.Flag != 'Suspect') %>% # Step 11  
   filter(TADA.ActivityMediaName == 'WATER') # Remove non-water samples  
  # censored data are retained in this dataset.

1. Create boxplots and log10 boxplots of each unique ‘TADA.CharacteristicName’ and site type through a for loop. Before the boxplots are created, NA values in ‘TADA.ResultMeasureValue’ are filtered out. Each site type is manually assigned a different color. These boxplots are exported to pdf for visual review. The following is an example boxplot for water temperature:

* 

1. Create an ‘ultra trim’ dataset that only contains the columns ‘OrganizerIdentifier’, ‘ActivityStartDate’, ‘MonitoringLocationIdentifier’, ‘MonitoringLocationName’, ‘MonitoringLocationTypeName’, ‘TADA.CharacteristicName’, ‘TADA.ResultMeasureValue’, ‘TADA.LatitudeMeasure’, and ‘TADA.LongitudeMeasure’.

* #####18. Ultra trim data#####  
  data\_18 <- data\_16 %>%   
   select(OrganizationIdentifier  
   ,ActivityStartDate  
   ,MonitoringLocationIdentifier  
   ,MonitoringLocationName  
   ,MonitoringLocationTypeName  
   ,TADA.CharacteristicName  
   ,TADA.ResultMeasureValue  
   ,TADA.ResultMeasure.MeasureUnitCode  
   ,TADA.LatitudeMeasure  
   ,TADA.LongitudeMeasure)

## Section 3: Match Data to AUs

This section matches the processed data with the proper Assessment Unit (AU) for the site type and location. This section is made up of Steps #19 through #20, although Step #20 is broken into six sub-steps.

1. Read in ‘ML\_AU\_Crosswalk.csv’ and join that with the output of Section 2. Use *leaflet* to create an interactive map in the ‘Viewer’ tab of the samples by monitoring location type.
2. Assigning sample locations to their appropriate AU if it is not specified in the crosswalk from Step #19.
   1. Read in the AU shapefiles and transform each of them to EPSG: 3338
   2. Break the sample locations with missing AUs into their appropriate site type (beach, lake, marine, or river). Select beach locations and find the nearest beach AU and calculate the distance. Join the neartest AU and calculated distance to the beach site locations. The following code is for the beach site type:
   * ######20b. Beaches #####  
     miss\_ML\_beaches <- missing\_ML %>% # filter appropriate sites  
      filter(MonitoringLocationTypeName == "BEACH Program Site-Ocean")  
       
     ### QC check  
     num\_sites <- nrow(miss\_ML\_beaches)  
       
     if(num\_sites == 0){  
      print(paste("There are NO beach monitoring locations missing AU data."  
      ,"Skip to the next section."))  
     } else {  
      print(paste("There ARE", num\_sites, "beach monitoring locations missing AU data."  
      ,"Continue to assign MLs to AUs using spatial joins."))  
     }# end if/else statement  
       
     ### convert to geospatial layer (sf object)  
     beach\_pts <- sf::st\_as\_sf(x = miss\_ML\_beaches, coords = c("TADA.LongitudeMeasure"  
      ,"TADA.LatitudeMeasure")  
      , crs = "+proj=longlat +datum=WGS84")%>%   
      sf::st\_transform(st\_crs(beach\_shp))  
       
     ### plot to see how they relate  
     ggplot() +  
      geom\_sf(data = AK\_shp)+  
      geom\_sf(data = beach\_shp, color = "red") +  
      geom\_sf(data = beach\_pts, color = "red") +  
      theme\_minimal()  
       
     ### spatial join  
     beach\_SpatJoin <- sf::st\_join(beach\_pts, beach\_shp, join = st\_nearest\_feature) %>% # join points and AUs  
      select(MonitoringLocationIdentifier, MonitoringLocationName  
      , MonitoringLocationTypeName, AUID\_ATTNS, Name\_AU, HUC10) # trim unneccessary columns  
       
     ### determine distance (m) between points and nearest feature  
     near\_feat <- sf::st\_nearest\_feature(beach\_pts, beach\_shp)  
     dist\_to\_AU\_m <- sf::st\_distance(beach\_pts, beach\_shp[near\_feat,], by\_element = TRUE)  
       
     ### join distance measurements to join results  
     beach\_SpatJoin2 <- cbind(beach\_SpatJoin, dist\_to\_AU\_m)  
       
     ### results and export data  
     miss\_ML\_beach\_results <- beach\_SpatJoin2 %>%  
      sf::st\_transform(4326) %>%   
      mutate(Longitude = unlist(map(geometry,1)),  
      Latitude = unlist(map(geometry,2))) %>%   
      sf::st\_drop\_geometry()
   1. Repeat for each site type.
   2. Create an interactive map using *leaflet* with each of the AU types and the sample locations that were just assigned to them.

## Section 4: Organize Data by AU

This section aims to explore the data within each AU and provide summary statistics for each.

1. Create a boxplot of the frequency of a given number of monitoring locations within an AU. Create a summary table by AU and ‘TADA.CharacteristicName’ that provides the number of samples, minimum, median, maximum, 25th quantile, and 75th quantile values for each.

## Section 5: Data Sufficiency

The final section of the code compares the data sufficiency needs for Alaska with the actual samples taken. This section aims to provide a final output that designates whether for a given AU and characteristic if there is enough data to draw conclusions on that AU.

1. Read in ‘AK\_DataSufficiency\_Crosswalk\_20231012.csv’, look for any constituents that aren’t in the sufficiency crosswalk or the processed data, summarize the output of Section 4 into number of samples and number of distinct years in which a sample was taken by AU and constituent, compare the summarized data to the sufficiency requirements