Data Processing User Guide

Tetra Tech

May 22, 2024

## Introduction

The following user guide pertains to the [*data\_processing.R*](https://github.com/KateriSalk/Alaska_IR_Automation/blob/main/Code/5_Data_Processing/data_processing.R) script developed by Tetra Tech for the Alaska Department of Environmental Conservation (AK DEC). This guide assumes that the user has relatively recent versions of [R](https://www.r-project.org/) and [RStudio](https://posit.co/download/rstudio-desktop/) installed and that they are familiar with the R coding language. Please direct any questions regarding the usage of the *data\_processing.R* script to Amber Crawford, AK DEC (amber.crawford@alaska.gov).

### Purpose

The purpose of the *data\_processing.R* script is to first conduct quality control and data wrangling on Water Quality Portal ([WQP](https://www.waterqualitydata.us/)) data downloaded using the [*data\_pull.R*](https://github.com/KateriSalk/Alaska_IR_Automation/blob/main/Code/3_Data_Pull/data_pull.R) script. These steps heavily rely upon *TADA* functions as described below. WQ data are then matched to AK DEC’s Assessment Units (AUs) using a crosswalk table. Monitoring locations with WQ data that are not in the crosswalk table are assigned to AUs using spatial joins to existing AU shapefiles and the joins can be evaluated using interactive mapping. Finally, data sufficiency is determined for each AU/pollutant combination using a data sufficiency lookup table for each designated use and pollutant.

## Required Packages

The required packages for the *data\_processing.R* code are [*TADA*](https://github.com/USEPA/TADA), [*tidyverse*](https://www.tidyverse.org/)*,* [*leaflet*](https://rstudio.github.io/leaflet/)*,* [*scales*](https://scales.r-lib.org/)*,* and [*sf*](https://r-spatial.github.io/sf/)*.* Installation for the *TADA* and *tidyverse* packages can be found in the Data Pull Guide. Below is a code chunk that demonstrates how to install and load these packages. If you work from using a Virtual Private Network (VPN), you may have trouble downloading packages, and if so, simply turn the VPN off before package download.

# 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 for it to run. The inputs are listed below. Note, only the outputs from *data\_pull.R* are loaded at the beginning of the script, whereas all others are loaded when needed farther into the script.

Exterior files required by data\_processing.R to function properly, the purpose of each file within the code, and their source.

| File Name | Purpose | Source |
| --- | --- | --- |
| .CSV outputs (varies by site types from *data\_pull.R*) | WQ data to be processed. | WQP (via *data\_pull.R*). |
| WQ\_Column\_Manager.csv | Quickly subset WQ dataset fields. | Tetra Tech |
| ML\_AU\_Crosswalk.csv | Crosswalk Monitoring Locations with AUs/ | AK DEC |
| AK\_DataSufficiency\_Crosswalk\_20231012.csv | Determine data sufficiency per AU/pollutant. | AK DEC |
| Beaches.shp | Beaches AU shapefile for mapping. | AK DEC |
| Lakes.shp | Lakes AU shapefile for mapping. | AK DEC |
| MAUs\_FINAL\_2023.shp | Marine AU shapefile for mapping. | AK DEC |
| Rivers.shp | Rivers AU shapefile for mapping. | AK DEC |
| cb\_2018\_us\_state\_500k.shp | US States shapefile for mapping. | Tetra Tech |

## Load Data

Data pulled from the WQP via the *data\_pull.R* script are read in to the *data\_processing.R* script for processing and quality control. The following lines read in the .CSVs by site type and explicitly excludes the composite dataset (‘data\_pull\_all.csv’) which allows for this code to be used for different combinations of files from *data\_pull.R*. Data are then combined into a single dataset.

#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 five different sections. Steps #1 through #13 are in the ‘Identify TADA Flags’ section. Each step represents a different *TADA* function that scans the input dataset for quality control issues. Note, every *TADA* function has an argument that asks whether the data should be “cleaned” which automatically removes erroneous or suspect data. We did not use the “clean” arguments to ensure that the WQ data could be reviewed by an analyst at the end.

Each *TADA* function appends one or more columns to the input dataset and are either updated versions of existing columns or flags for potential erroneous data points (every new *TADA* field has a ‘TADA.’ prefix). Within *data\_processing.R*, 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 (in order of appearance in *data\_processing.R*):

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. Note, that any Measure Qualifier Codes not recognized by this function are flagged as ‘uncategorized’. A sub-step in Step #11 manually replaces ‘uncategorized’ flags with appropriate flags based on a list of known Measure Qualifier Codes from AK DEC (see code chunk below). Any new flags encountered after the development of *data\_processing.R* will require the analyst to manually append new flags into the code chunk.

* # add any changes by making a new row below  
  data\_11b <- data\_11a %>%   
   mutate(TADA.MeasureQualifierCode.Flag =   
   case\_when((MeasureQualifierCode == "H;U") ~ "Suspect"  
   , (MeasureQualifierCode == "RC;U") ~ "Non-Detect"  
   , (MeasureQualifierCode == "H;RC;U") ~ "Suspect"  
   , (MeasureQualifierCode == "J-R;TOC") ~ "Pass"  
   , (MeasureQualifierCode == "TOC;U") ~ "Non-Detect"  
   , (MeasureQualifierCode == "RC;SUS") ~ "Suspect"  
   , (MeasureQualifierCode == "O;RC") ~ "Pass"  
   , (MeasureQualifierCode == "H;RC") ~ "Suspect"  
   , (MeasureQualifierCode == "B;J-R") ~ "Pass"  
   , (MeasureQualifierCode == "H;J-R") ~ "Suspect"  
   , (MeasureQualifierCode == "IQCOL;U") ~ "Pass"  
   , (MeasureQualifierCode == "IQCOL;J-R") ~ "Pass"  
   , (MeasureQualifierCode == "LL;RC") ~ "Pass"  
   , (MeasureQualifierCode == "BQL;RC") ~ "Pass"  
   , (MeasureQualifierCode == "B;D") ~ "Pass"  
   , (MeasureQualifierCode == "SDROL;U") ~ "Suspect"  
   , TRUE ~ TADA.MeasureQualifierCode.Flag))

1. *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

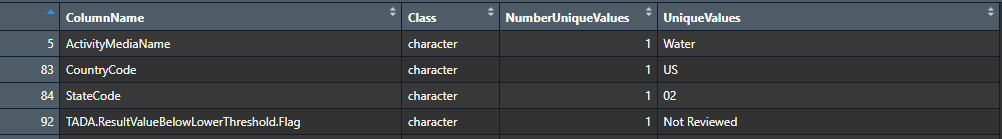
After the *TADA* functions are applied, Step #13 removes any columns that are entirely NA from the dataset.

#####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

Section #2 uses the flags from Section #1 to remove erroneous data. This section is composed of Steps #14 through #18. The following is a breakdown of each step.

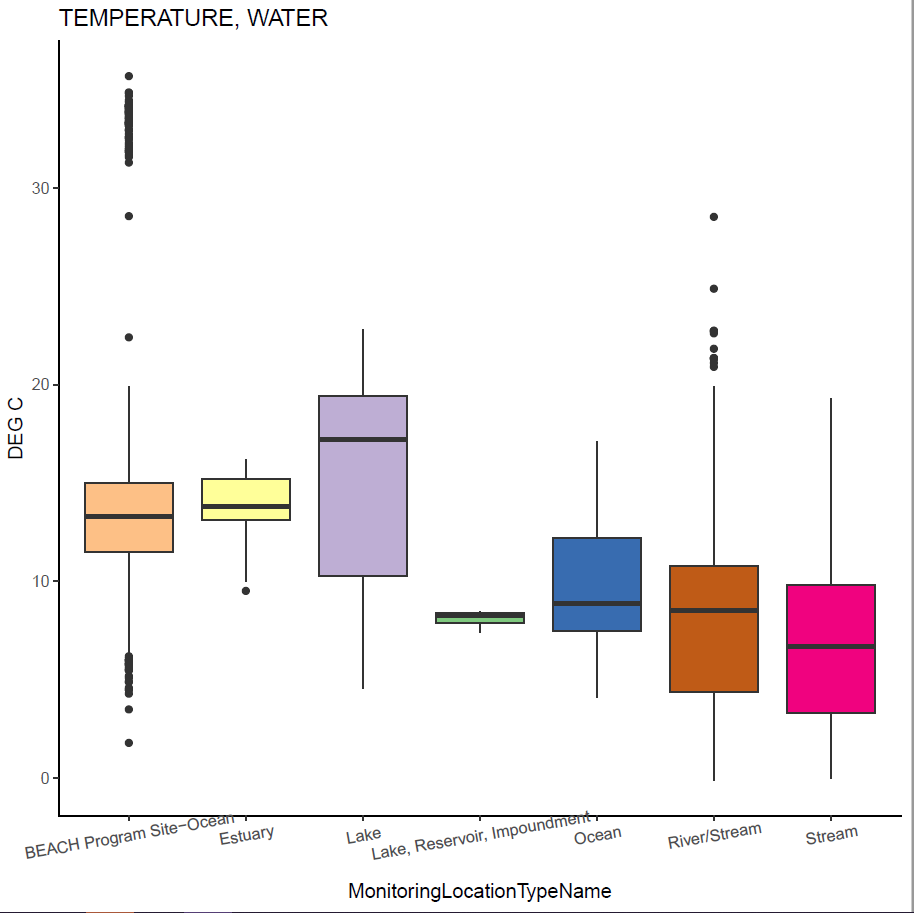
1. Create a data summary table with each row being a column from the Step #13 output in Section 1 (see screenshot of example output below). The fields within the summary table include the column name, class, number of unique values, and a list of those unique values (if the list is 10 or less in length). The data summary table provides the user with the means to quickly review field attributes which can be used to identify and remove suspect data.

* 

1. The *WQ\_Column\_Manager.csv* is read in and the ‘Keep\_YN’ column is used to filter out unnecessary columns. A quality control check ensures that the external manager file is up to date and prompts the user for an update if any new columns are detected in the WQ dataset.
2. The WQ dataset is trimmed by removing invalid data (mainly from *TADA* function fields) (see the code chunk below). The user can also review the data summary table generated in Step #14 to inform additional trimming steps. Note, that at the time of development, outputs from some *TADA* functions did not match the outputs described in the R documentation, therefore, the following crosswalk was assumed (see comments):

* # 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 to remove any unnecessary columns. At the time of development, only the columns listed in the code chunk below were kept. The user is welcome to retain any additional columns by appending their name to the code.

* #####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 WQ data with the appropriate AK DEC Assessment Unit (AU). 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 using the ‘MonitoringLocationIdentifier’ field. Monitoring locations from the WQ dataset that are already in the crosswalk table will be assigned an AU. This step also has optional code that uses *leaflet* to create an interactive map of the monitoring locations by waterbody type (displays in the RStudio ‘Viewer’ tab).
2. Assigning monitoring locations to their appropriate AU if it is not specified in the crosswalk from Step #19.
   1. Read in the AU shapefiles and transform them to NAD83 Alaska Albers (EPSG: 3338).
   2. Separate monitoring locations with missing AUs by waterbody type (beach, lake, marine, or river). Note, the waterbody types listed in ‘MonitoringLocationTypeName’ from WQP do not exactly match the waterbody types listed in *ML\_AU\_Crosswalk.csv*.
   3. Select beach monitoring locations and assign them to the nearest beach AU using a spatial join. Distance between the monitoring location points and the AU polygons are calculated. The following code is for the beach waterbody 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 waterbody type.
   2. Create an interactive map using *leaflet* with each of the AU waterbody types and the monitoring locations with missing AU assignments. Use this map and the outputs from the spatial joins to evaluate whether montioring locations were correctly assigned to AUs.
   3. Update *ML\_AU\_Crosswalk.csv* with correct monitoring location and AU information based on the results from the previous sub-steps. This should be done manually for analyst review of spatial joins.
   4. Rerun Section 3 once all correct and approved spatial joins have been added to *ML\_AU\_Crosswalk.csv.*

## 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 AK DEC’s Integrated Reporting process with the WQ data actually available. This section provides a final output that designates whether there is sufficient data for a given AU and pollutant combinations for assessment purposes.

1. Read in *AK\_DataSufficiency\_Crosswalk\_20231012.csv*. For each AU in the WQ dataset, the data sufficiency table is filtered by waterbody type and the corresponding constituents. The number of samples and number of years of data for each constituent within a given AU is compared to the minimum requirements. If sufficient data is available, that constituent can be used at the assessment level for the given AU. Conversely, if insufficient, the data can only be used at the screening level.