VIBI SURVEY123 DATA - FROM SPREADSHEETS TO WETLANDS DATABASE

STANDARD OPERATING PROCEDURES

This SOP describes procedures and R scripts for uploading VIBI Survey123-generated Excel spreadsheets into the HTLNWetlands MS Access database using R tidyverse.

Workflow

This SOP describes the uploading process of Excel files generated by Survey123 surveys for VIBI field data into the HTLNWetlands database . The survey data is collected for VIBI herbaceous and VIBI woody analysis. VIBI woody data also includes data for big trees (>= 40 cm in diameter). For all three datasets (herbaceous, woody, and big trees) the workflow is the same. Excel spreadsheets derived from Survey123 surveys are loaded as csv files into R scripts and transformed to match their destination tables in Access. The R scripts generate “loadfiles” which are imported into the HTLNWetlands Access database. The imported load files are appended to each respective tables (tbl\_VIBI\_herb, tbl\_VIBI\_woody, and tbl\_BigTrees) in MS Access. The code listings for each script are described in the following text. Once the data is appended to Access tables, it is exported back out as .csv files for end-to-end validation. The validation tests consist of (1) verifying no duplicate records are contained in either the load file or the exported datasets, (2) verifying no missing values anywhere in the measured fields and (3) matching record counts between loadfiles and data exports. See the following diagram for the overall workflow.

Graphical user interface, diagram, application

Description automatically generated

PROCEDURES

Review the Survey123 Excel files:

1. Begin by reviewing the Survey123 Excel files.
2. Make working copies of the original spreadsheets and store the original files in a safe location.
3. Open the spreadsheets using Excel.
4. Review column headings. Remove any spaces or special characters. Spaces should be replaced with underscores. Special characters such as commas, apostrophes, percent signs, and math symbols should be replaced with text.
5. Ensure that all columns contain data. Columns that do not contain data should be deleted.
6. Remove all commas in the comments column to prevent parsing errors associated with the comma delimiter.

Processing VIBI herbaceous data

1. Export the VIBI spreadsheets from Excel into csv format. Use the CSV UTF-8 format.
2. Run the script Survey123\_VIBI\_herb\_final.R
3. The following is an annotated code listing explaining VIBI\_herb\_final.R

################################################################################

#

# VIBI\_herb\_final.R

#

# Gareth Rowell, 4/26/2024

#

# Generates loadfile for tbl\_VIBI\_herb and provides end2end

# tests for exported table tbl\_VIBI\_herb after its been appended with the

# 2023 <or latest field season> data.

#

#

################################################################################

library(tidyverse)

#setwd("./VIBI-herbaceous")

Step 1 loads the csv files into dataframes and creates the working Access\_data dataframe.

#################

#

# Step 1 - load spreadsheet csv files and appended them

#

#################

# load the Survey123 data

load\_file1 <- read\_csv("CUVA\_VIBI\_herb1.csv")

load\_file2 <- read\_csv("CUVA\_VIBI\_herb2.csv")

load\_file3 <- read\_csv("CUVA\_VIBI\_herb3.csv")

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

In Step 2, HerbSiteName and CoverClass\_LT\_6m columns are renamed to match up with the table tbl\_VIBI\_herb. The mm/dd/yyyy date format is converted to the universal date format yyyy-mm-dd.

#################

#

# Step 2 - select and rename columns, convert date to yyyy-mm-dd

#

#################

Access\_data <- Access\_data |>

select(Species, Comments, Module, CoverClass\_LT\_6m,

CoverClassAll, EditDate, HerbSiteName) |>

mutate(

FeatureID = HerbSiteName,

CoverClass = CoverClass\_LT\_6m,

EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))

)

glimpse(Access\_data)

In Step 3, I’m removing missing records and duplicate records. In Step3a, I am replacing the “NA” symbols. NA means “not available” and is used widely in R tidyverse. Here, I substitute it with -9999 in numerical columns. -9999 is the designated value for numeric fields in MS Access. Then I inspect the -9999 rows. After inspection, I remove all the - 9999 records from the dataframe.

#################

#

# Step 3a - Substitute NA with -9999 in CoverClass and CoverClassAll

# Then remove those with -9999 in CoverClass

#

#################

Access\_data$CoverClass <- Access\_data$CoverClass |> replace\_na(-9999)

Access\_data$CoverClassAll <- Access\_data$CoverClassAll |> replace\_na(-9999)

Access\_data |>

filter(CoverClass == -9999)

Access\_data <- Access\_data |>

filter(CoverClass != -9999)

# then test

Access\_data |>

filter(CoverClass == -9999)

In Step 3b, I am inspecting and removing duplicate rows.

##########

#

# Step 3b - check for duplicates

#

##########

# test for dups

Access\_data |>

count(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

) |>

filter(n > 1)

# Remove dups with distinct()

Access\_data <- Access\_data |>

distinct(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

)

# test for dups one more time

Access\_data |>

count(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

) |>

filter(n > 1)

In Step 4, the code generates the EventID. Note that I’ve created a lookup table called Month\_LUT to generate the 3 letter month code that replaces the numeric value MM for months.

#################

#

# Step 4a - Generate EventID from EditDate

#

#################

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

*Reconciling the location information.* In step 5, the new LocationID column in the dataframe is generated using the tbl\_Locations table and the values from the field data based on the FeatureID. This step tests for any mismatches between tbl\_Locations and the site values used in the field. These values must match exactly before proceeding to Step 6. The best practice is to use the field locations “as-is” and modify the tbl\_Locations.csv file as needed to make things match in Step 5. The updated tbl\_Locations table can optionally be loaded back into the HTLNWetlands database if needed.

Do not attempt to update locations information (or any other data) in the field data spreadsheets. The spreadsheets are very large and updates are extremely difficult to validate. These corrections can instead be made by editing the tbl\_Locations lookup table, then importing the new look-up table back into MS Access if needed.

#################

#

# Step 5 - Create the LocationID column from the FeatureID column

#

#################

# First test for NA's in FeatureID column

subset(Access\_data,is.na(FeatureID))

# there are no NAs in FeatureID column

Locations\_LUT <- read\_csv("tbl\_Locations\_fixed.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

# test for NA's in LocationID

df <- subset(Access\_data,is.na(LocationID))

glimpse(df)

# locate any FeatureIDs aren't matching up

df <- df |>

distinct(LocationID, FeatureID)

df

Access\_data <- Access\_data |>

filter(!is.na(LocationID))

# should be zero rows for the above tests once locations are fixed

Finally, in this last step, Step 6, I am just selecting the columns that will be needed to append the data to tbl\_VIBI\_herb in the database. The last line writes the dataframe to the loadfile.

#################

#

# Step 6 - Clean up columns and write load file

#

#################

# clean up columns

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Species, Comments, Module,

CoverClass, CoverClassAll, EditDate )

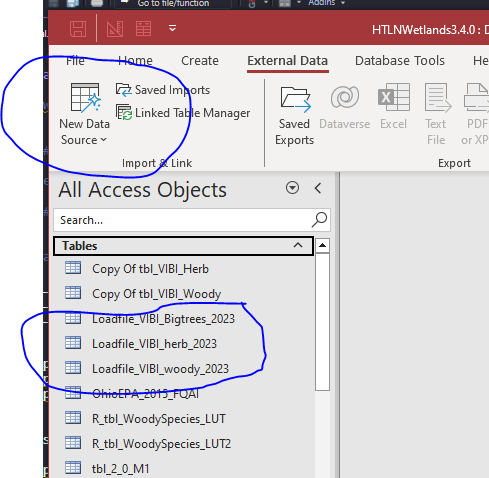
# comment out the write set when re-using this script for end2end tests

writexl::write\_xlsx(Access\_data, "Load\_VIBI\_herb\_2023.xlsx")

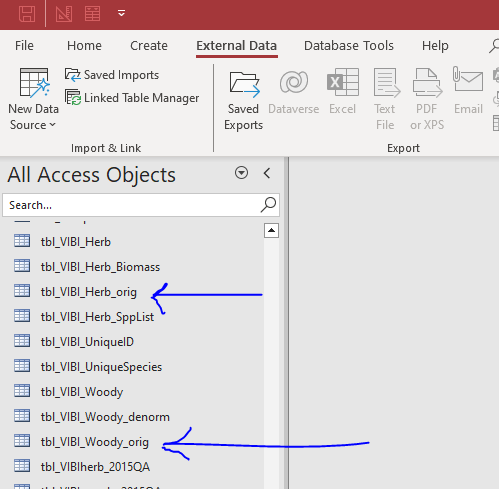
The load file should be ready to import into Access to be joined to tbl\_VIBI\_herb.

Appending VIBI herbaceous data in HTLNWetlandsDB

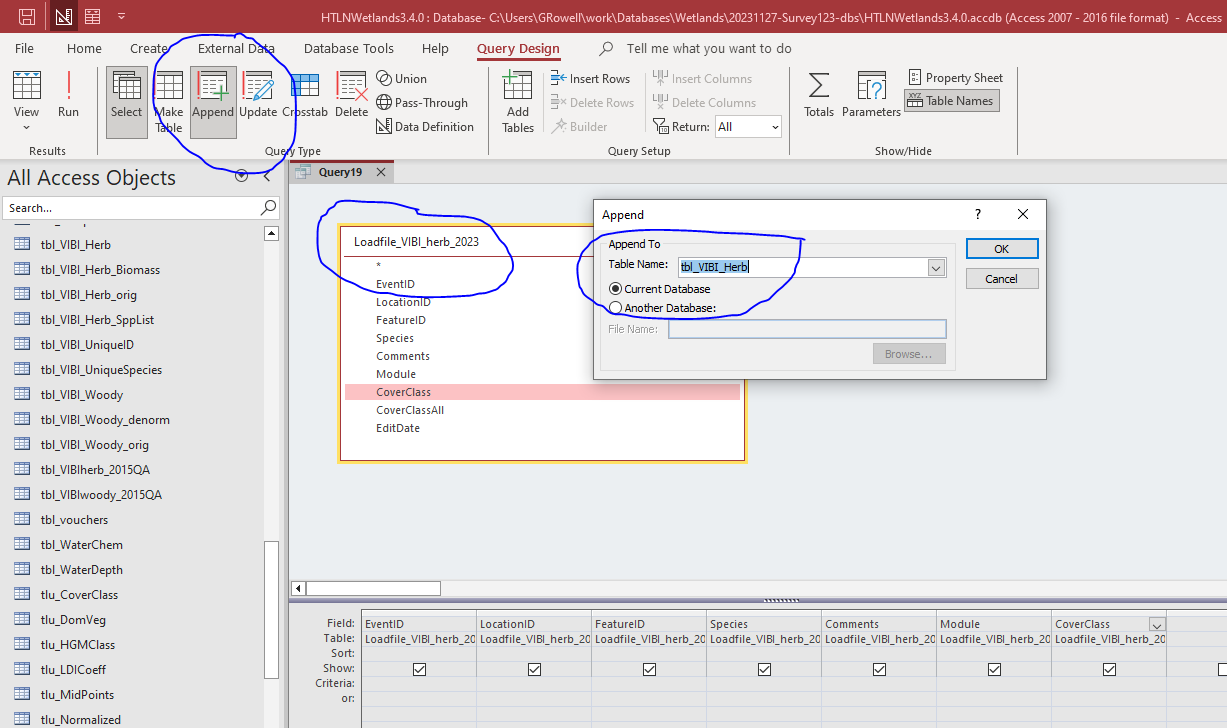
1. Open the latest version of HTLNWetlandsDB.
2. Go to the External Data tab and Import “Load\_VIBI\_2023.xlsx” into table Loadfile\_VIBI\_herb\_2023



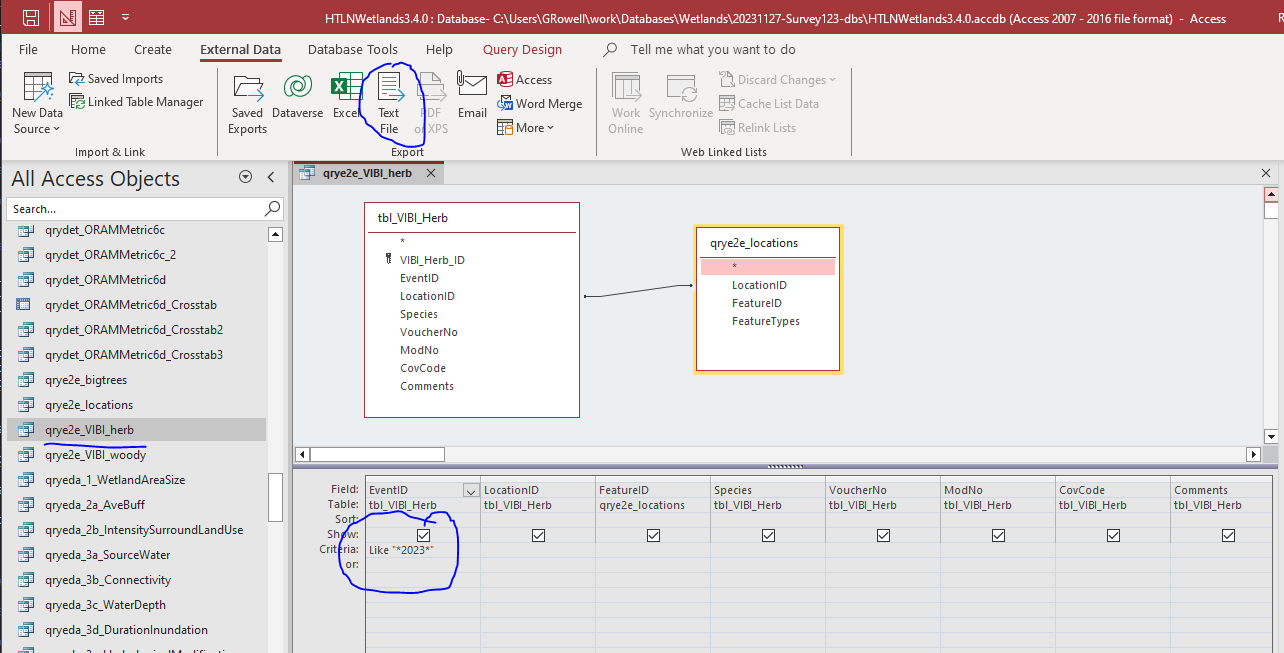
1. Make a copy of the table tbl\_VIBI\_herbaceous and label it tbl\_VIBI\_herbaceous\_orig. You will need to roll back to original table if anything fails at the end-to-end test at the very end.



1. Append the loadfile table to tbl\_VIBI\_herbaceous.



1. Export the tbl\_Herbaceous using the Access query labeled “qrye2e\_VIBI\_herb”. This query joins e2e\_Locations (VIBI locations only) to the tbl\_VIBI\_herb table so you can test both the LocationID and FeatureID in the end to end test. It also filters for only 2023 (or the current field season) data.



1. The exported query, called qrye2e\_VIBI\_herb.csv, is used in the end-to-end tests.

End2end testing for VIBI herbaceous data.

The purpose of the end2end test is to verify the data exported from HTLNWetlands matches the Access\_data data frame. This test will verify that the new data has been appended correctly in the Access database.

1. Open R Studio and reload the previously used R script, VIBI\_herb\_final.R, and go to the bottom of the script. The end-to-end tests are located at the very bottom of the script. Select the entire script up to the beginning of the end-to-end testing with your mouse in R Studio in press Ctl + Enter. This will run the code to recreate the object called Access\_data.
2. Once the Access\_data object is created, proceed to the end-to-end tests. The first step is to load the end-to-end export file called qrye2e\_VIBI\_herb.csv. The steps that follow below are to ensure the column ands and selected columns match exactly between the Access\_data dataframe and the end2end dataframe. These must match exactly for the anti\_join() test that follows.

#------------------------------------------------------------------------------

# End2End test begins here

end2end <- read\_csv("qrye2e\_VIBI\_herb.csv")

glimpse(Access\_data)

glimpse(end2end)

# matching column names

Access\_data <- Access\_data |>

mutate(

ModNo = Module,

CovCode = CoverClass

)

# select only shared columns

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Species, ModNo, CovCode, Comments

)

glimpse(Access\_data)

glimpse(end2end)

# removing voucherno and comments from end2end

end2end <- end2end |>

select(EventID, LocationID, FeatureID, Species, ModNo, CovCode

)

glimpse(Access\_data)

glimpse(end2end)

1. Once the columns match exactly, we test to verify that the are no *record* differences between Access\_data and end2end dataframes. I am using the function anti\_join()

my\_columns = c('EventID', 'LocationID', 'FeatureID', 'Species', 'ModNo', 'CovCode')

view(anti\_join(Access\_data, end2end, by=my\_columns))

1. If there are differences between the two dataframes, the output can be assigned to a dataframe. In the code below, I am capturing the differences in FeatureIDs between the Access\_data and end2end datafreams.

df <- anti\_join(end2end, Access\_data, by=my\_columns)

df |> distinct(FeatureID)

1. In the next code, I am testing for duplicate records in both the Access\_data and end2end dataframes.Access\_data |>

count(EventID, LocationID, Species, ModNo, CovCode) |>

filter(n > 1)

end2end |>

count(EventID, LocationID, Species, ModNo, CovCode) |>

filter(n > 1)

1. In the last step, I am verifying that the total sample size is the same in both dataframes.

Access\_data |>

count(EventID, LocationID, Species, ModNo, CovCode)

end2end |>

count(EventID, LocationID, Species, ModNo, CovCode)

Processing data for VIBI woody and VIBI big trees

The workflow for VIBI woody and VIBI big trees is virtually identical to VIBI herbaceous data. The complete code listings in R tidyverse are given below:

################################################################################

#

# VIBI\_woody\_final.R

#

# Gareth Rowell, 2/28/2024

#

# This end2end test compares the original 2023 data against

# the exported table tbl\_VIBI\_woody after its been appended with the

# 2023 data.

#

#

################################################################################

library(tidyverse)

#setwd("../HTLN-Data-Capture-Scripts/wetlands/src")

#setwd("./VIBI-woody")

#################

#

# Step 1 - load spreadsheet csv files and appended them

# Note - species codes were only used in CUVA\_VIBI\_woody1.csv

# and also CUVA\_VIBI\_woody3.csv

# They were joined to WoodySpecies\_LUT2.csv to create WoodySpecies

#

#################

# species codes were only used in CUVA\_VIBI\_woody1.csv

# join to WoodySPecies\_LUT2.csv to create WoodySpecies

load\_file1 <- read\_csv("CUVA\_VIBI\_woody1.csv")

problems(load\_file1)

WoodySpecies\_LUT <- read\_csv("WoodySpecies\_LUT2.csv")

load\_file1 <- load\_file1 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file1)

glimpse(WoodySpecies\_LUT)

# view(load\_file1)

# check for NAs in WoodySpecies

load\_file1 |>

select(SpeciesCode, WoodySpecies) |>

filter(is.na(WoodySpecies)) |>

distinct()

load\_file2 <- read\_csv("CUVA\_VIBI\_woody2.csv")

problems(load\_file2)

load\_file3 <- read\_csv("CUVA\_VIBI\_woody3.csv")

problems(load\_file3)

load\_file3 <- load\_file3 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file3)

load\_file4 <- read\_csv("CUVA\_VIBI\_woody4.csv")

problems(load\_file4)

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

glimpse(load\_file4)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file4)

glimpse(Access\_data)

load\_file <- Access\_data # for normalization test in step 8

#################

#

# Step 2 - select and rename columns, convert date to yyyy-mm-dd

#

#################

Access\_data <- Access\_data |>

select(WoodyModule, WoodySpecies, EditDate, WoodySiteName, ShrubClump, D0to1,

D1to2\_5, D2\_5to5, D5to10, D10to15, D15to20, D20to25, D25to30, D30to35,

D35to40, Dgt40, Dgt40\_1, Dgt40\_2, Dgt40\_3, Dgt40\_4, Dgt40\_5)

Access\_data <- Access\_data |>

mutate( FeatureID = WoodySiteName) |>

mutate(EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))) |>

mutate( Module\_No = WoodyModule )

glimpse(Access\_data)

#################

#

# Step 3 - Rename columns using DiamID values for pivot\_longer

#

#################

Access\_data$Col1 <- Access\_data$ShrubClump

Access\_data$Col2<- Access\_data$D0to1

Access\_data$Col3 <- Access\_data$D1to2\_5

Access\_data$Col4 <- Access\_data$D2\_5to5

Access\_data$Col5 <- Access\_data$D5to10

Access\_data$Col6 <- Access\_data$D10to15

Access\_data$Col7 <- Access\_data$D15to20

Access\_data$Col8 <- Access\_data$D20to25

Access\_data$Col9 <- Access\_data$D25to30

Access\_data$Col10 <- Access\_data$D30to35

Access\_data$Col11 <- Access\_data$D35to40

Access\_data$Col12 <- Access\_data$Dgt40

glimpse(Access\_data)

#################

#

# Step 4 - Generate EventID from EditDate

#

#################

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

glimpse(Access\_data)

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

#################

#

# Step 5 - create the LocationID column from the FeatureID column

# and a lookup table from HTLNWetlands

# Using the fixed locations lookup table

#

#################

# Locations\_LUT <- read\_csv("tbl\_Locations\_20230316.csv")

Locations\_LUT <- read\_csv("tbl\_Locations\_fixed.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

glimpse(Access\_data)

#################

#

# Step 6 - set up columns before normalization, then pivot longer

#

#################

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, WoodySpecies,

EditDate, WoodySiteName, Col1, Col2, Col3, Col4, Col5,

Col6, Col7, Col8, Col9, Col10, Col11, Col12)

glimpse(Access\_data)

# pivot longer (normalize)

Access\_data <- Access\_data |>

pivot\_longer(

cols = starts\_with("Col"),

names\_to = "DiamID",

values\_to = "Count",

values\_drop\_na = TRUE

)

glimpse(Access\_data)

#################

#

# Step 7 - Join the diameter information from a LUT

#

#################

Diam\_LUT <- read\_csv("Diam\_LUT.csv")

glimpse(Diam\_LUT)

Access\_data <- Access\_data |>

left\_join(Diam\_LUT, join\_by(DiamID))

#################

#

# Step 8 - Validate normalization and join using

# Sum of counts in initial load file

# against total\_counts for each diameter

# in final version

#

#################

Initial\_load <- load\_file |>

select(ShrubClump, D0to1,

D1to2\_5, D2\_5to5, D5to10, D10to15, D15to20, D20to25, D25to30, D30to35,

D35to40, Dgt40)

colSums(Initial\_load, na.rm=TRUE)

Access\_data |>

group\_by(Diam\_Desc) |>

summarize(

total\_count = sum(Count)

)

#################

#

# Step 9a - Substitute NA with -9999 in CoverClass and CoverClassAll

# Then remove those with -9999 in CoverClass

#

#

#################

glimpse(Access\_data)

Access\_data$Count <- Access\_data$Count |> replace\_na(-9999)

Access\_data |>

filter(Count == -9999)

Access\_data <- Access\_data |>

filter(Count != -9999)

# then test

Access\_data |>

filter(Count == -9999)

##########

#

# Step 9b - check for duplicates

#

##########

# test for dups

Access\_data |>

count(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc) |>

filter(n > 1)

# Remove dups with distinct()

Access\_data <- Access\_data |>

distinct(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc)

Access\_data

# test for dups

Access\_data |>

count(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc) |>

filter(n > 1)

##########

#

# Step 10 - Write load file

#

##########

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, Diam\_Code,

Count)

# writexl::write\_xlsx(Access\_data, "Load\_VIBI\_woody\_2023.xlsx")

#------------------------------------------------------------------------------

# End2End test begins here

# need to test for duplicate records

Access\_data |>

count(EventID, FeatureID, Module\_No, WoodySpecies, Diam\_Code, Count) |>

filter(n > 1)

# test for presence of -9999 in anything

Access\_data |>

filter(Count == -9999)

end2end <- read\_csv("qrye2e\_VIBI\_woody.csv")

problems(end2end)

glimpse(end2end)

glimpse(Access\_data)

# matching column names

Access\_data <- Access\_data |>

mutate(

DiamID = Diam\_Code,

Scientific\_Name = WoodySpecies

)

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, DiamID,

Count)

# testing for PK - unique no-nulls

Access\_data |>

count(EventID, LocationID, Module\_No, Scientific\_Name, DiamID, Count) |>

filter(n > 1)

end2enddups <- end2end |>

count(EventID, LocationID, Module\_No, Scientific\_Name, DiamID, Count) |>

filter(n > 1) |>

print(n = 45)

# what are the record differences between these two dataframes

my\_columns <- c('EventID', 'LocationID', 'FeatureID', 'Module\_No',

'Scientific\_Name','DiamID', 'Count')

view(anti\_join(Access\_data, end2end, by=my\_columns))

# all the extract Access\_data records have the result of corrected locations

# verify this

df <- anti\_join(end2end, Access\_data, by=my\_columns)

df |>

distinct(FeatureID)

################################################################################

#

# VIBI\_BigTree\_final.R

#

# Gareth Rowell, 2/28/2024

#

# This end2end test compares the original 2023 data against

# the exported table tbl\_BigTrees after its been appended with the

# 2023 data.

#

#

################################################################################

library(tidyverse)

# setwd("./VIBI-BigTrees")

#################

#

# Step 1 - load spreadsheet csv files and appended them

# Note - species codes were only used in CUVA\_VIBI\_woody1.csv

# and also CUVA\_VIBI\_woody3.csv

# They were joined to WoodySpecies\_LUT2.csv to create WoodySpecies

#

#################

# species codes were only used in CUVA\_VIBI\_woody1.csv

# join to WoodySPecies\_LUT2.csv to create WoodySpecies

load\_file1 <- read\_csv("CUVA\_VIBI\_woody1.csv")

problems(load\_file1)

WoodySpecies\_LUT <- read\_csv("WoodySpecies\_LUT2.csv")

load\_file1 <- load\_file1 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file1)

glimpse(WoodySpecies\_LUT)

# view(load\_file1)

# check for NAs in WoodySpecies

load\_file1 |>

select(SpeciesCode, WoodySpecies) |>

filter(is.na(WoodySpecies)) |>

distinct()

load\_file2 <- read\_csv("CUVA\_VIBI\_woody2.csv")

problems(load\_file2)

load\_file3 <- read\_csv("CUVA\_VIBI\_woody3.csv")

problems(load\_file3)

load\_file3 <- load\_file3 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file3)

load\_file4 <- read\_csv("CUVA\_VIBI\_woody4.csv")

problems(load\_file4)

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

glimpse(load\_file4)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file4)

glimpse(Access\_data)

##########

#

# Step 2 - select columns for Access import

# for big trees, need columns 40\_1 onwards

#

##########

Access\_data <- Access\_data |>

select(WoodyModule, WoodySpecies, EditDate, WoodySiteName,

Dgt40\_1, Dgt40\_2, Dgt40\_3, Dgt40\_4, Dgt40\_5)

glimpse(Access\_data)

# n = 1731

##########

#

# Step 3 - create columns for FeatureID and Module\_No

# and convert date format

#

##########

Access\_data <- Access\_data |>

mutate( FeatureID = WoodySiteName) |>

mutate(EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))) |>

mutate( Module\_No = WoodyModule )

Access\_data

##########

#

# Step 4 - Generate EventID from EditDate

#

##########

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

glimpse(Access\_data)

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

glimpse(Months\_LUT)

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

glimpse(Access\_data)

##########

#

# Step 5 - create the LocationID column from the FeatureID column

# and a lookup table from HTLNWetlands

#

# corrected for fixed FeatureIDs

#

##########

# Locations\_LUT <- read\_csv("tbl\_Locations\_20230316.csv")

Locations\_LUT <- read\_csv("tbl\_Locations\_fixed.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

glimpse(Access\_data)

##########

#

# Step 6 - Normalize DBH columns

#

#

##########

# Rename columns using DiamID values for pivot\_longer

Access\_data$Tree1 <- Access\_data$Dgt40\_1

Access\_data$Tree2 <- Access\_data$Dgt40\_2

Access\_data$Tree3 <- Access\_data$Dgt40\_3

Access\_data$Tree4 <- Access\_data$Dgt40\_4

Access\_data$Tree5 <- Access\_data$Dgt40\_5

Access\_data <- Access\_data |>

mutate( Scientific\_Name = WoodySpecies) |>

mutate( SampleDate = EditDate) |>

select(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, Tree1,

Tree2, Tree3, Tree4, Tree5)

glimpse(Access\_data)

# pivot longer (normalize)

Access\_data <- Access\_data |>

pivot\_longer(

cols = starts\_with("Tree"),

names\_to = "TreeName",

values\_to = "DBH",

values\_drop\_na = TRUE

)

glimpse(Access\_data)

##########

#

# Step 7 - check for duplicates

#

##########

# test for dups

Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, DBH)

Access\_data |>

group\_by(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, DBH) |>

summarize(

n = n(),

) |>

filter(n > 1)

# Remove duplicates

Access\_data <- Access\_data |>

distinct(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, DBH)

Access\_data |>

group\_by(EventID, LocationID, FeatureID, Module\_No, Scientific\_Name, DBH) |>

summarize(

n = n(),

) |>

filter(n > 1)

#view(Access\_data)

# writexl::write\_xlsx(Access\_data, "Load\_VIBI\_BigTrees\_2023.xlsx")

#------------------------------------------------------------------------------

# End2End test begins here

end2end <- read\_csv("qrye2e\_bigtrees.csv")

problems(end2end)

glimpse(end2end)

glimpse(Access\_data)

# matching column names

Access\_data <- Access\_data |>

mutate(

ModNo = Module\_No

)

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, ModNo, Scientific\_Name, DBH)

glimpse(end2end)

glimpse(Access\_data)

# testing for PK - unique no-nulls

Access\_data |>

count(EventID, LocationID, ModNo, Scientific\_Name, DBH) |>

filter(n > 1)

end2end |>

count(EventID, LocationID, ModNo, Scientific\_Name, DBH) |>

filter(n > 1)

# what are the record differences between these two dataframes

my\_columns <- c('EventID', 'LocationID', 'FeatureID', 'ModNo',

'Scientific\_Name','DBH')

df <- anti\_join(end2end, Access\_data, by=my\_columns)

df

df <- anti\_join(Access\_data, end2end, by=my\_columns)

df