8: Data Wrangling

Environmental Data Analytics | Kateri Salk

Spring 2019

LESSON OBJECTIVES

1. Describe the usefulness of data wrangling and its place in the data pipeline
2. Wrangle datasets with dplyr functions
3. Apply data wrangling skills to a real-world example dataset

SET UP YOUR DATA ANALYSIS SESSION

```
getwd()
```

```
## [1] "/Users/ks501/Documents/GithubRepos/ENV872"
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------

## v ggplot2 3.1.0     v purrr   0.3.0
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

## Warning: package 'tibble' was built under R version 3.5.2

## Warning: package 'purrr' was built under R version 3.5.2

## -- Conflicts ------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
NTL.phys.data.PeterPaul <- read.csv("./Data/Processed/NTL-LTER_Lake_ChemistryPhysics_PeterPaul_Processe
NTL.nutrient.data <- read.csv("./Data/Raw/NTL-LTER_Lake_Nutrients_Raw.csv")
```

REVIEW OF BASIC DATA EXPLORATION AND WRANGLING

```
# Data summaries for physical data
head(NTL.phys.data.PeterPaul)
```

```
##   lakeid  lakename year4 daynum sampledate depth temperature_C
## 1      L Paul Lake  1984    148    5/27/84  0.00          14.5
## 2      L Paul Lake  1984    148    5/27/84  0.25            NA
## 3      L Paul Lake  1984    148    5/27/84  0.50            NA
## 4      L Paul Lake  1984    148    5/27/84  0.75            NA
## 5      L Paul Lake  1984    148    5/27/84  1.00          14.5
## 6      L Paul Lake  1984    148    5/27/84  1.50            NA
##   dissolvedOxygen irradianceWater irradianceDeck comments
## 1             9.5            1750           1620     <NA>
## 2              NA            1550           1620     <NA>
## 3              NA            1150           1620     <NA>
## 4              NA             975           1620     <NA>
## 5             8.8             870           1620     <NA>
## 6              NA             610           1620     <NA>
```

```
colnames(NTL.phys.data.PeterPaul)
```

```
##  [1] "lakeid"          "lakename"        "year4"
##  [4] "daynum"          "sampledate"      "depth"
##  [7] "temperature_C"   "dissolvedOxygen" "irradianceWater"
## [10] "irradianceDeck"  "comments"
```

```
dim(NTL.phys.data.PeterPaul)
```

```
## [1] 21613    11
```

```
summary(NTL.phys.data.PeterPaul$comments)
```

```
## DO Probe bad - Doesn't go to zero     DO taken with Jones Lab Meter
##                               132                               112
##                              NA's
##                             21369
```

```
class(NTL.phys.data.PeterPaul$sampledate)
```

```
## [1] "factor"
```

```
# Format sampledate as date
NTL.phys.data.PeterPaul$sampledate <- as.Date(NTL.phys.data.PeterPaul$sampledate, format = "%m/%d/%y")

# Select Peter and Paul Lakes from the nutrient dataset
NTL.nutrient.data.PeterPaul <- filter(NTL.nutrient.data, lakename == "Paul Lake" | lakename == "Peter La

# Data summaries for nutrient data
head(NTL.nutrient.data.PeterPaul)
```

```
##   lakeid  lakename year4 daynum sampledate depth_id depth tn_ug tp_ug nh34
## 1      L Paul Lake  1991    140    5/20/91        1  0.00   538    25   NA
## 2      L Paul Lake  1991    140    5/20/91        2  0.85   285    14   NA
## 3      L Paul Lake  1991    140    5/20/91        3  1.75   399    14   NA
## 4      L Paul Lake  1991    140    5/20/91        4  3.00   453    14   NA
## 5      L Paul Lake  1991    140    5/20/91        5  4.00   363    13   NA
## 6      L Paul Lake  1991    140    5/20/91        6  6.00   583    37   NA
##   no23 po4 comments
## 1   NA  NA
## 2   NA  NA
## 3   NA  NA
## 4   NA  NA
## 5   NA  NA
## 6   NA  NA
```

```
colnames(NTL.nutrient.data.PeterPaul)
```

```
##  [1] "lakeid"     "lakename"   "year4"      "daynum"     "sampledate"
##  [6] "depth_id"   "depth"      "tn_ug"      "tp_ug"      "nh34"
## [11] "no23"       "po4"        "comments"
```

```
dim(NTL.nutrient.data.PeterPaul)
```

```
## [1] 2770   13
```

```
summary(NTL.nutrient.data.PeterPaul$comments)
```

```
##                                                sample missing
##                            2770                             0
## TP value suspect, far too high
##                               0
```

```
class(NTL.nutrient.data.PeterPaul$sampledate)
```

```
## [1] "factor"
```

```
NTL.nutrient.data.PeterPaul$sampledate <- as.Date(NTL.nutrient.data.PeterPaul$sampledate, format = "%m/%

# Save processed nutrient file
write.csv(NTL.nutrient.data.PeterPaul, row.names = FALSE, file = "./Data/Processed/NTL-LTER_Lake_Nutrie

# Remove columns that are not of interest for analysis
NTL.phys.data.PeterPaul.skinny <- select(NTL.phys.data.PeterPaul,
                                         lakename, year4, sampledate:irradianceDeck)

NTL.nutrient.data.PeterPaul.skinny <- select(NTL.nutrient.data.PeterPaul,
                                             lakename, year4, sampledate, depth:po4)
```

TIDY DATASETS

For most situations, data analysis works best when you have organized your data into a tidy dataset. A tidy dataset is defined as:

- Each variable is a column
- Each row is an observation (e.g., sampling event from a specific date and/or location)
- Each value is in its own cell

However, there may be situations where we want to reshape our dataset, for example if we want to facet numerical data points by measurement type (more on this in the data visualization unit). We can program this reshaping in a few short lines of code using the package `tidyr`, which is conveniently included in the `tidyverse` package.

```
# Gather nutrient data into one column
NTL.nutrient.data.PeterPaul.gathered <- gather(NTL.nutrient.data.PeterPaul.skinny, "nutrient", "concent
NTL.nutrient.data.PeterPaul.gathered <- subset(NTL.nutrient.data.PeterPaul.gathered, !is.na(concentrati
count(NTL.nutrient.data.PeterPaul.gathered, nutrient)
```

```
## # A tibble: 5 x 2
##   nutrient     n
##   <chr>    <int>
## 1 nh34      1204
## 2 no23      1235
## 3 po4       1246
## 4 tn_ug     1729
## 5 tp_ug     2583
```

```
write.csv(NTL.nutrient.data.PeterPaul.gathered, row.names = FALSE,
          file ="./Data/Processed/NTL-LTER_Lake_Nutrients_PeterPaulGathered_Processed.csv")

# Spread nutrient data into separate columns
NTL.nutrient.data.PeterPaul.spread <- spread(NTL.nutrient.data.PeterPaul.gathered, nutrient, concentrat
# Split components of cells into multiple columns
# Opposite of 'separate' is 'unite'
NTL.nutrient.data.PeterPaul.dates <- separate(NTL.nutrient.data.PeterPaul.skinny, sampledate, c("Y", "m
```

JOINING MULTIPLE DATASETS

In many cases, we will want to combine datasets into one dataset. If all column names match, the data frames can be combined with the `rbind` function. If some column names match and some column names don't match, we can combine the data frames using a "join" function according to common conditions that exist in the matching columns. We will demonstrate this with the NTL-LTER physical and nutrient datasets, where we have specific instances when physical and nutrient data were collected on the same date, at the same lake, and at the same depth.

In dplyr, there are several types of join functions:

- `inner_join`: return rows in x where there are matching values in y, and all columns in x and y (mutating join).
- `semi_join`: return all rows from x where there are matching values in y, keeping just columns from x (filtering join).
- `left_join`: return all rows from x, and all columns from x and y (mutating join).
- `anti_join`: return all rows from x where there are *not* matching values in y, keeping just columns from x (filtering join).
- `full_join`: return all rows and all columns from x and y. Returns NA for missing values (mutating join).

Let's say we want to generate a new dataset that contains all possible physical and chemical data for Peter and Paul Lakes. In this case, we want to do a full join.

```
NTL.phys.nutrient.data.PeterPaul <- full_join(NTL.phys.data.PeterPaul.skinny, NTL.nutrient.data.PeterPa
```

```
## Joining, by = c("lakename", "year4", "sampledate", "depth")
```

```
## Warning: Column `lakename` joining factors with different levels, coercing
## to character vector
```

LUBRIDATE

A package that makes coercing date much easier is `lubridate`. A guide to the package can be found at https://lubridate.tidyverse.org/. The cheat sheet within that web page is excellent too. This package can do many things (hint: look into this package if you are having unique date-type issues), but today we will be using two of its functions for our NTL dataset.

```
#install.packages(lubridate)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
# add a month column to the dataset
NTL.phys.nutrient.data.PeterPaul <- mutate(NTL.phys.nutrient.data.PeterPaul, month = month(sampledate))
```

```
# reorder columns to put month with the rest of the date variables
NTL.phys.nutrient.data.PeterPaul <- select(NTL.phys.nutrient.data.PeterPaul, lakename, year4, month, sam

# find out the start and end dates of the dataset
interval(NTL.phys.nutrient.data.PeterPaul$sampledate[1], NTL.phys.nutrient.data.PeterPaul$sampledate[233
```

```
## [1] 1984-05-27 UTC--2014-08-29 UTC
```

```
interval(first(NTL.phys.nutrient.data.PeterPaul$sampledate), last(NTL.phys.nutrient.data.PeterPaul$sampl
```

```
## [1] 1984-05-27 UTC--2014-08-29 UTC
```

SPLIT-APPLY-COMBINE

dplyr functionality, combined with the pipes operator, allows us to split datasets according to groupings (function: `group_by`), then run operations on those groupings and return the output of those operations. There is a lot of flexibility in this approach, but we will illustrate just one example today.

```
NTL.PeterPaul.summaries <-
  NTL.phys.nutrient.data.PeterPaul %>%
  filter(depth == 0) %>%
  group_by(lakename) %>%
  filter(!is.na(temperature_C) & !is.na(tn_ug) & !is.na(tp_ug)) %>%
  summarise(meantemp = mean(temperature_C),
            sdtemp = sd(temperature_C),
            meanTN = mean(tn_ug),
            sdTN = sd(tn_ug),
            meanTP = mean(tp_ug),
            sdTP = sd(tp_ug))
```

ALTERNATIVE METHODS FOR DATA WRANGLING

If you want to iteratively perform operations on your data, there exist several options. We have demonstrated the pipe as one option. Additional options include the `apply` function (https://www.rdocumentation.org/packages/base/versions/3.5.2/topics/apply) and `for` loops (https://swcarpentry.github.io/r-novice-inflammation/15-supp-loops-in-depth/). These options are good options as well (again, multiple ways to get to the same outcome). A word of caution: loops are slow. This may not make a difference for small datasets, but small time additions will make a difference with large datasets.