CRWA Flagging Model - 2023 Update

Ayushman Choudhury

20 January 2023

Introduction

My name is Ayushman Choudhury. I'm currently a sophomore at Brown University studying Applied Mathematics, Computer Science, and Music. For my Winternship with the Charles River Watershed Association, I updated the E. Coli Flagging Model, used to generate recommendations regarding water quality on the Charles River.

In this file, I have documented my process, including gathering data, wrangling it in R, and creating GLM and MLR models for each of CRWA's 4 locations along the Charles River.

Boilerplate Text - this is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When code is executed within the notebook, the results appear beneath the code. Code can be executed by clicking the *Run* button within the chunk or by placing the cursor inside it and pressing *Cmd+Shift+Enter*.

```
library(tidyverse)
library(lubridate)
library(dataRetrieval)
library(caret)
library(naniar)
set.seed(1)
```

Section 1: Gathering Data

1.1 - E. Coli

The first data source is a copy of CRWA's E. coli data from 2017 to 2022. These are the values that this model will predict. I did some basic selecting and Date/Time formatting in a Google Sheet, before loading the CSV in R. Here are the columns I used:

- Date_Collected - samples are weekly, typically on Thursdays
- Time_Collected - typically in the morning
- Site_ID - either 1NBS, 2LARZ, 3BU, or 4LONG
- Reporting_Result - E. coli. measured in cfu/100mL

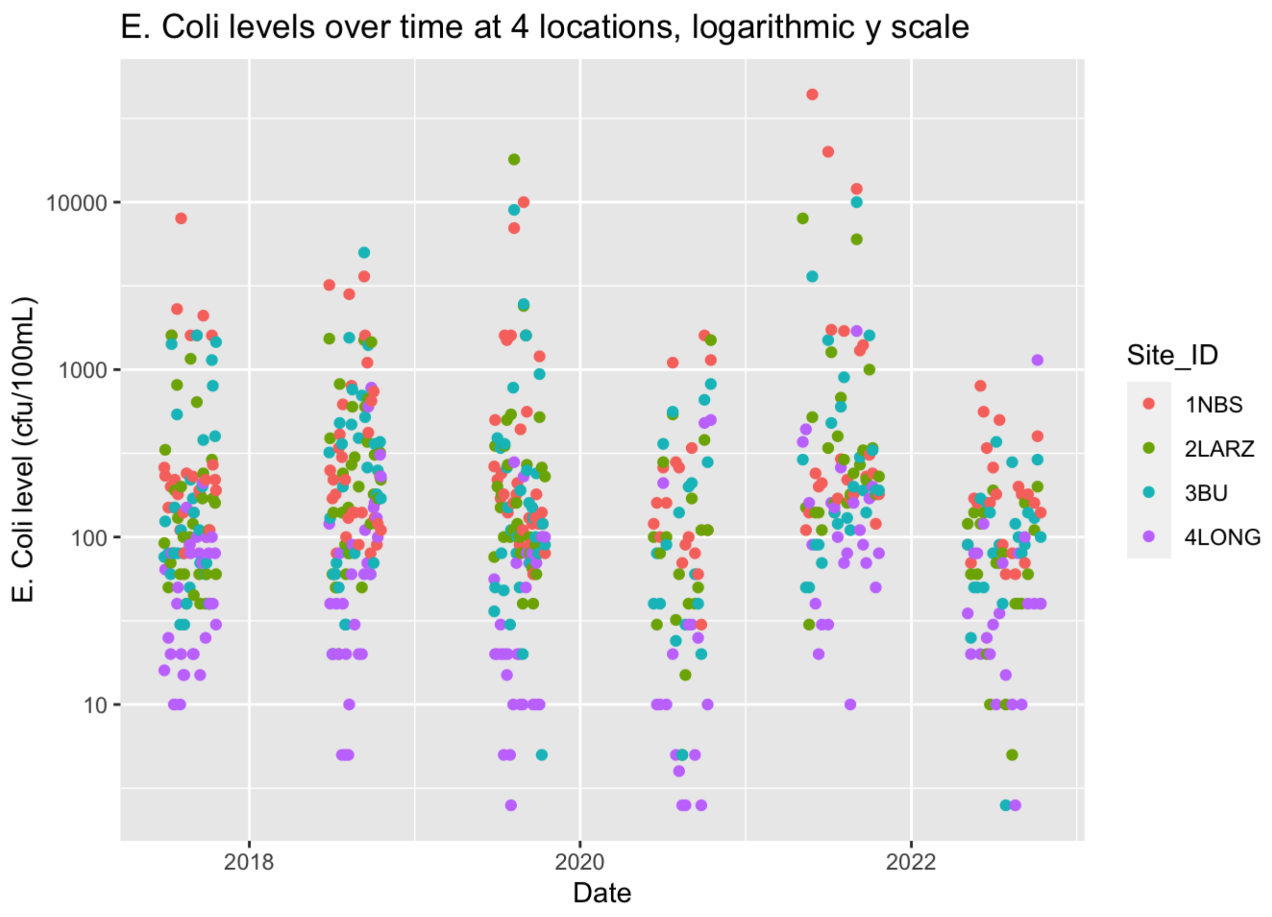
In the following code, I access this CSV from my local "sources" folder, combine Date and Time into `datetime` , and round it to the nearest 10-minute interval. The tibble and dot plot give a quick overview.

```
ecoliData <- read.csv("sources/ecoliWrangledCopy.csv") %>%
  mutate(datetime = round_date(
    as.POSIXct(
      paste(.$Date_Collected, .$Time_Collected),
      format = "%m/%d/%Y %H:%M:%S"),
    "10 minutes")) %>%
  select(datetime, Site_ID, Reporting_Result) %>%
  filter(.$Site_ID %in% c("1NBS", "2LARZ", "3BU", "4LONG"))

ecoliData %>% tibble()
```

```
## # A tibble: 627 × 3
##    datetime            Site_ID Reporting_Result
##    <dttm>              <chr>              <dbl>
##  1 2017-06-27 09:50:00 1NBS                 260
##  2 2017-06-27 09:20:00 2LARZ                 92
##  3 2017-06-27 09:00:00 3BU                   76
##  4 2017-06-27 08:40:00 4LONG                 16
##  5 2017-06-29 09:30:00 1NBS                 232
##  6 2017-06-29 08:50:00 2LARZ                332
##  7 2017-06-29 08:40:00 3BU                  124
##  8 2017-06-29 08:10:00 4LONG                 64
##  9 2017-07-06 09:50:00 1NBS                 150
## 10 2017-07-06 09:10:00 2LARZ                 50
## # … with 617 more rows
## # ℹ Use `print(n = ...)` to see more rows
```

```
ecoliData %>% ggplot(aes(x = datetime, y = Reporting_Result, color = Site_ID)) +
  geom_point() +
  scale_y_continuous(trans = 'log10') +
  ggtitle("E. Coli levels over time at 4 locations, logarithmic y scale") +
  labs(x = "Date", y = "E. Coli level (cfu/100mL)")
```



1.2 - Weather

The second data source is CRWA's weather data. This is stored in several CSV files by year, from 2017 to 2022, each of which has about 26,000 rows. Each file has a different amount of metadata to skip — `read.csv()` has a skip parameter, but I just removed these rows manually in a text editor. The following code accesses them from my local "sources" folder.

```
rawWeather2017 <- read.csv("sources/2017_Flagging_Season_2023_01_09_09_55_56_EST_1.csv")
rawWeather2018 <- read.csv("sources/2018_Flagging_Season_2023_01_09_09_54_55_EST_1.csv")
rawWeather2019 <- read.csv("sources/2019_Flagging_Season_2023_01_09_09_53_45_EST_1.csv")
rawWeather2020 <- read.csv("sources/2020_Flagging_Season_2023_01_09_09_44_11_EST_1.csv")
rawWeather2021 <- read.csv("sources/2021_Flagging_Season_2023_01_09_09_48_29_EST_1.csv")
rawWeather2022 <- read.csv("sources/2022_Flagging_Season_2023_01_09_09_50_04_EST_1.csv")
```

There's some wrangling to do here too:

1. Each file has its columns in a different order, so I standardized this order as Pressure, PAR, Rain, RH%, Wind Dir, Wind Speed, Gust Speed, DewPt, Air Temp, Water Temp. (Temperatures are in Fahrenheit).
2. For some data points, the values jump between several different columns. I merged these together.

```
#2017 --
colnames(rawWeather2017) = c("id", "time", "pressure", "par", "rain", "rh", "gustSpeed", "de
wPt", "windDir", "airTemp", "windSpeed", "waterTemp")

weather2017 <- rawWeather2017 %>%
  mutate(waterTemp = ifelse(waterTemp > 32, waterTemp, NA)) %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)

#2018 --
colnames(rawWeather2018) = c("id", "time", "pressure", "par", "rain", "rh", "windSpeed", "gu
stSpeed", "dewPt", "windDir", "waterTemp2", "airTemp3", "airTemp2", "airTemp1", "waterTemp
1", "waterTempNull")

weather2018 <- rawWeather2018 %>%
  mutate(waterTemp = ifelse(is.na(waterTemp2), waterTemp1, waterTemp2)) %>%
  mutate(airTemp = ifelse(is.na(airTemp3), ifelse(is.na(airTemp2), airTemp1, airTemp2), airT
emp3)) %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)

#2019 --
colnames(rawWeather2019) = c("id", "time", "pressure", "par", "rain", "rh", "windSpeed", "gu
stSpeed", "dewPt", "windDir", "waterTemp", "airTemp")

weather2019 <- rawWeather2019 %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)

#2020 --
colnames(rawWeather2020) = c("id", "time", "pressure", "par", "rain", "rh", "windSpeed", "gu
stSpeed", "dewPt1", "windDir1", "airTemp1", "waterTemp1", "waterTemp2", "dewPt2", "windDir
2", "airTemp2")

weather2020 <- rawWeather2020 %>%
  mutate(dewPt = ifelse(is.na(dewPt2), dewPt1, dewPt2)) %>%
  mutate(windDir = ifelse(is.na(windDir2), windDir1, windDir2)) %>%
  mutate(airTemp = ifelse(is.na(airTemp2), airTemp1, airTemp2)) %>%
  mutate(waterTemp = NA) %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)

#2021 --
colnames(rawWeather2021) = c("id", "time", "pressure", "par", "rain", "rh", "windSpeed", "gu
stSpeed", "dewPt1", "windDir1", "airTemp1", "waterTempNull", "dewPt2", "windDir2", "tempUnkn
own", "airTemp2")

weather2021 <- rawWeather2021 %>%
  mutate(dewPt = ifelse(is.na(dewPt2), dewPt1, dewPt2)) %>%
  mutate(windDir = ifelse(is.na(windDir2), windDir1, windDir2)) %>%
  mutate(airTemp = ifelse(is.na(airTemp2), airTemp1, airTemp2)) %>%
  mutate(waterTemp = tempUnknown) %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)

#2022 --
colnames(rawWeather2022) = c("id", "time", "pressure", "par", "rain", "rh", "dewPt", "windDi
r", "waterTemp", "airTemp", "windSpeed", "gustSpeed")

weather2022 <- rawWeather2022 %>%
  mutate(datetime = as.POSIXct(time, format = "%m/%d/%y %H:%M:%S")) %>%
  select(datetime, pressure, par, rain, rh, windDir, windSpeed, gustSpeed, dewPt, airTemp, w
aterTemp)
```

We can stack all of these together to create a large (158,116 x 11) dataset for all of the weather data. Here's a quick tibble to show the results.

```
weatherData <- rbind(weather2017, weather2018, weather2019, weather2020, weather2021, weathe
r2022)

weatherData %>% tibble()
```

```
## # A tibble: 158,116 × 11
##    datetime            pressure   par  rain    rh windDir windSp…¹ gustS…² dewPt
##    <dttm>                 <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>   <dbl> <dbl>
##  1 2017-05-01 00:00:00     30.2     1     0  74.9     157      3.4    11.9  40.8
##  2 2017-05-01 00:10:00     30.2     1     0  75.1     160      6      12.8  40.9
##  3 2017-05-01 00:20:00     30.2     1     0  74.8     163      2.6     9.4  41.3
##  4 2017-05-01 00:30:00     30.2     1     0  71.4     168      7.7    17.9  39.9
##  5 2017-05-01 00:40:00     30.2     1     0  70.1     164      6      17.9  39.6
##  6 2017-05-01 00:50:00     30.2     1     0  70.1     157      2.6    11.9  39.6
##  7 2017-05-01 01:00:00     30.2     1     0  70.5     170      3.4    11.1  39.6
##  8 2017-05-01 01:10:00     30.2     1     0  69.9     175      3.4    11.1  39.4
##  9 2017-05-01 01:20:00     30.2     1     0  68.6     167      6      13.6  38.9
## 10 2017-05-01 01:30:00     30.2     1     0  65.2     180      5.1    14.5  37.5
## # … with 158,106 more rows, 2 more variables: airTemp <dbl>, waterTemp <dbl>,
## #   and abbreviated variable names ¹windSpeed, ²gustSpeed
## # ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

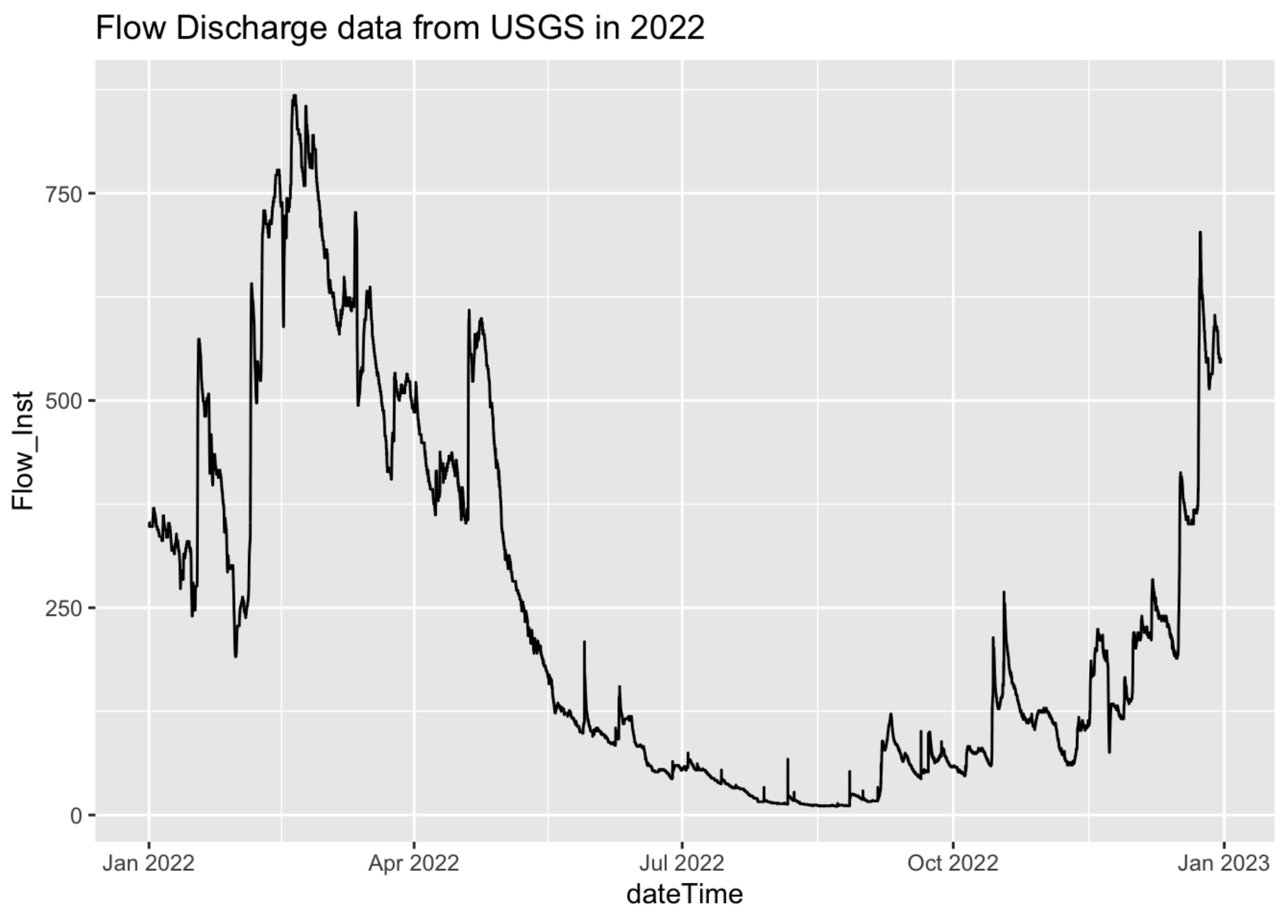
1.3 - USGS Water Data - discharge

The final data source is USGS's water data regarding discharge. USGS has a specific R library `dataRetrieval` which I use below. The location is in Waltham, MA on the Charles River (site number 01104500). Below, I access the discharge data (code 00060) from the beginning of 2017 to the end of 2022.

```
waltham <- readNWISuv(siteNumbers = "01104500",
                      parameterCd = "00060",
                      startDate = "2017-01-01",
                      endDate = "2022-12-31") %>%
  renameNWISColumns()
```

Retrieving this data from USGS can take some time, but once it is done we can include it as yet another predictor. Here's a quick plot of the flow during 2022.

```
waltham %>%
  filter(dateTime >= as.Date('2022-01-01'), dateTime <= as.Date('2022-12-31')) %>%
  ggplot(aes(dateTime, Flow_Inst)) +
  geom_line() +
  ggtitle("Flow Discharge data from USGS in 2022")
```



Section 2: Defining Features and Outcomes

2.1 - Strategy

We have 627 E. Coli measurements. To predict these measurements, we should have 627 sets of features from which to train our model. The simplest choice would be to only use the 627 rows at the times the E. Coli sample was recorded. However, since E. Coli populations can depend on the conditions of the water over the previous hour or day (and since we have a lot more than 627 rows), we can also include averages of the flow and weather values over the previous 1 hour, 12 hours, 24 hours, 48 hours, and 72 hours.

We can be even more specific with the rainfall values. CRWA has previously used these variables (credit to Eli Kane):

- 12 hour intervals: 0-12, 12-24, 24-36, 36-48, 48-60, 60-72, 72-84, 84-96
- 24 hour intervals: 0-24, 24-48, 48-72, 72-96
- 48 hour intervals: 0-48, 48-96
- Prior week: 0-168
- Days since last rain & Days since last major rain (> 0.1 inches)

For averages, I will be using the geometric mean over the arithmetic mean, to lessen the effect of outliers. This can be calculated as `exp(mean(log(data)))`.

2.2 - Flow Features

For each E. coli measurement time, I:

- Calculate its index in the flow data frame
- Get the Geometric means of the flow 1/12/24/48/72 hours prior using `sapply`

Once these values are stored, I transpose the matrix, cast it to a data frame, and adjust the column/row names.

```r
flowFeaturesSeparate <- sapply(ecoliData$datetime, function(x) {
  flowIndex <- length(which(waltham$dateTime <= x))

  flowTimes <- c(1,12,24,48,72) * 4 - 1

  sapply(flowTimes, function(x){
    exp(mean(log(waltham$Flow_Inst[(flowIndex - x):flowIndex])))
  })
})

flowFeatures <- flowFeaturesSeparate %>%
  data.frame() %>% t() %>% data.frame()

colnames(flowFeatures) = c("hourAvgFlow", "halfDayAvgFlow", "dayAvgFlow", "twoDayAvgFlow",
"threeDayAvgFlow")

row.names(flowFeatures) <- NULL
```

2.3 - Rainfall Total Features

Here, I use multiple sapply calls for each range.

```r
rainfallFeaturesSeparate <- sapply(ecoliData$datetime, function(x) {
  weatherIndex = length(which(weatherData$datetime <= x))

  halfDayIntervals <- sapply(seq(12,96,12), function(t) {
    sum(weatherData$rain[(weatherIndex - t*6):(weatherIndex - (t + 12)*6)])
  })

  dayIntervals <- sapply(seq(24,96,24), function(t) {
    sum(weatherData$rain[(weatherIndex - t*6):(weatherIndex - (t + 24)*6)])
  })

  twoDayIntervals <- sapply(seq(48,96,48), function(t) {
    sum(weatherData$rain[(weatherIndex - t*6):(weatherIndex - (t + 48)*6)])
  })

  weekRain <- sum(weatherData$rain[(weatherIndex - 6*24*7):(weatherIndex)])

  c(halfDayIntervals, dayIntervals, twoDayIntervals, weekRain)
})

rainfallFeatures <- rainfallFeaturesSeparate %>%
  data.frame() %>% t() %>% data.frame()

colnames(rainfallFeatures) = c(
  "rain0_12", "rain12_24", "rain24_36", "rain36_48", "rain48_60", "rain60_72", "rain72_84",
"rain84_96",
  "rain0_24", "rain24_48", "rain48_72", "rain72_96",
  "rain0_48", "rain48_96", "rainWeek")

row.names(rainfallFeatures) = NULL
```

2.4 - Periodic Rain Features

For each E. coli measure time, I run a loop:

- Using a day counter,
- While the time range is still valid, and the year is the same as the original measure time's year,
- I find the max rain value between the day counter and the original measure time.
    - If it is > 0.1, this is a major rain event. Break.
    - Else if it is > 0, this is a rain event. Continue looking for a major rain event.
    - Else, continue looking.

Since rain values aren't collected year-round, this metric might not be useful for E. coli observations during dry days at the beginning of the weather data collection period.

This is not an issue in this scope, because weather data is collected starting on May 1st and E. coli data is collected starting at the end of June, and there has always been some rainfall between these two months.

However, it could become an issue if E. coli data was collected closer to May 1st, before the first rainfall of the season.

```r
rainPeriodSeparate <- sapply(ecoliData$datetime, function(x) {
  year <- format(
    as.POSIXct(x, origin = "1970-01-01"),
    format = "%Y")

  originalIndex = length(which(weatherData$datetime <= x))

  daysSinceLastRain <- 365
  daysSinceLastMajorRain <- 365
  day <- 0

  while (((originalIndex - day*6*24) > 1)
         &
         (format(as.POSIXct(weatherData$datetime[originalIndex]), "%Y") == year)) {

    maxRain <- max(weatherData$rain[(originalIndex - day*6*24):originalIndex])

    if (maxRain > 0.1) {
      daysSinceLastMajorRain <- day
      if (day < daysSinceLastRain) {
        daysSinceLastRain <- day
      }
      break
    } else if (maxRain > 0) {
      if (day < daysSinceLastRain) {
        daysSinceLastRain <- day
      }
    }
    day <- day + 1
  }#Close while loop

  c(daysSinceLastRain, daysSinceLastMajorRain)
})

rainPeriod <- rainPeriodSeparate %>%
  data.frame() %>% t() %>% data.frame()

colnames(rainPeriod) <- c("daysSinceLastRain", "daysSinceLastMajorRain")

row.names(rainPeriod) <- NULL
```

2.5 - Other Weather Features

This is similar to the code for 2.2 - Flow Features except with more metrics in the sapply, that are concatenated together.

```r
weatherFeaturesSeparate <- sapply(ecoliData$datetime, function(x) {
  weatherIndex <- length(which(weatherData$datetime <= x))

  weatherTimes <- c(1, 12, 24, 28, 72) * 6

  weatherTempArray <- sapply(weatherTimes, function(x) {
    c(
      exp(mean(log(weatherData$pressure[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$par[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$rh[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$windSpeed[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$gustSpeed[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$dewPt[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$airTemp[(weatherIndex - x):weatherIndex]))),
      exp(mean(log(weatherData$waterTemp[(weatherIndex - x):weatherIndex])))
    )
  })

  c(t(weatherTempArray))
})

weatherFeatures <- weatherFeaturesSeparate %>%
  data.frame() %>% t() %>% data.frame()

colnames(weatherFeatures) <- c(
  "hourAvgPressure", "halfDayAvgPressure", "dayAvgPressure", "twoDayAvgPressure", "threeDayA
vgPressure",
  "hourAvgPAR", "halfDayAvgPAR", "dayAvgPAR", "twoDayAvgPAR", "threeDayAvgPAR",
  "hourAvgRH", "halfDayAvgRH", "dayAvgRH", "twoDayAvgRH", "threeDayAvgRH",
  "hourAvgWind", "halfDayAvgWind", "dayAvgWind", "twoDayAvgWind", "threeDayAvgWind",
  "hourAvgGust", "halfDayAvgGust", "dayAvgGust", "twoDayAvgGust", "threeDayAvgGust",
  "hourAvgDew", "halfDayAvgDew", "dayAvgDew", "twoDayAvgDew", "threeDayAvgDew",
  "hourAvgAirTemp", "halfDayAvgAirTemp", "dayAvgAirTemp", "twoDayAvgAirTemp", "threeDayAvgAi
rTemp",
  "hourAvgWaterTemp", "halfDayAvgWaterTemp", "dayAvgWaterTemp", "twoDayAvgWaterTemp", "three
DayAvgWaterTemp")

row.names(weatherFeatures) <- NULL
```

2.6 - Putting the Features Together

Now, we `cbind` all four data frames together. For each of the 627 observations, we now have `datetime` , `year` , and 62 environmental features, to use in the prediction models.

```
features <- cbind(ecoliData$datetime,
                  (format(as.POSIXct(ecoliData$datetime), "%Y")),
                  flowFeatures, rainfallFeatures, rainPeriod, weatherFeatures)

names(features)[1] <- 'datetime'
names(features)[2] <- 'year'

dim(features)
```

```
## [1] 627  64
```

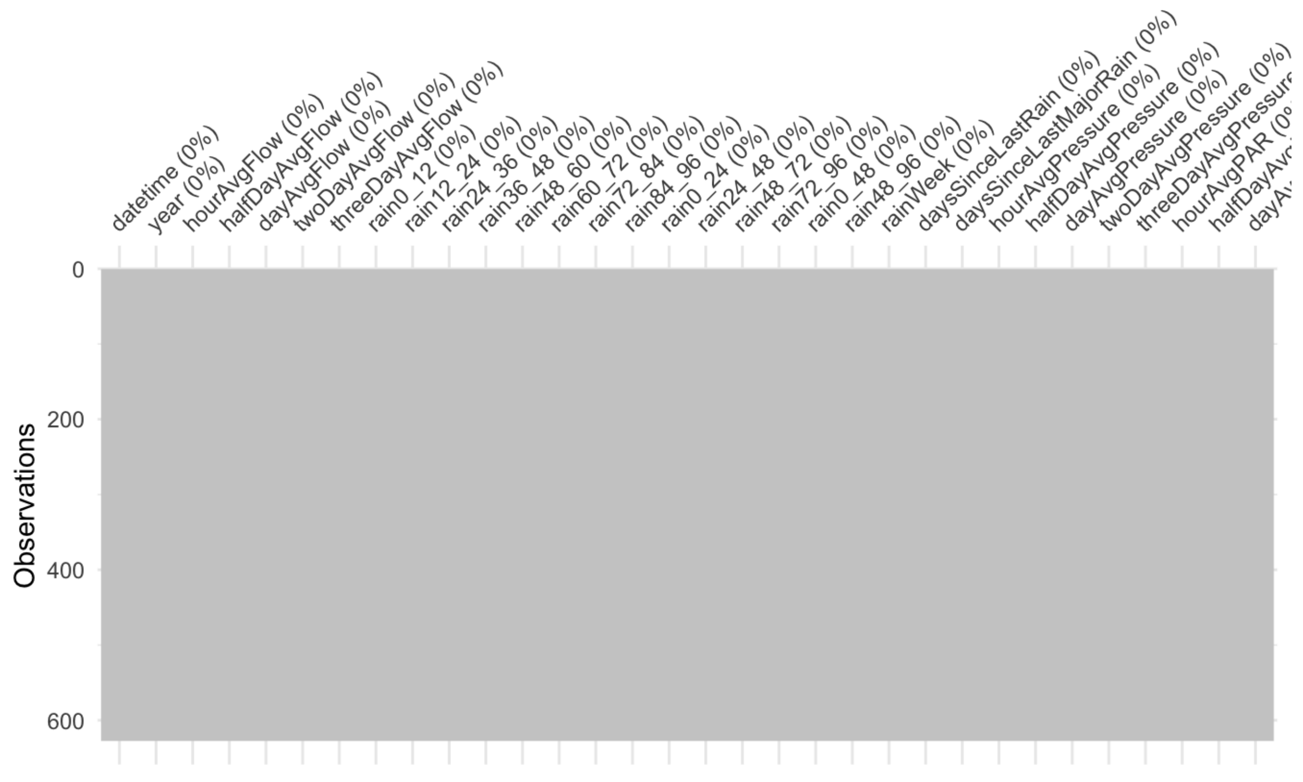
2.7 - Checking Completeness of Features

Before constructing the model, we should check the completeness of the Features data frame.

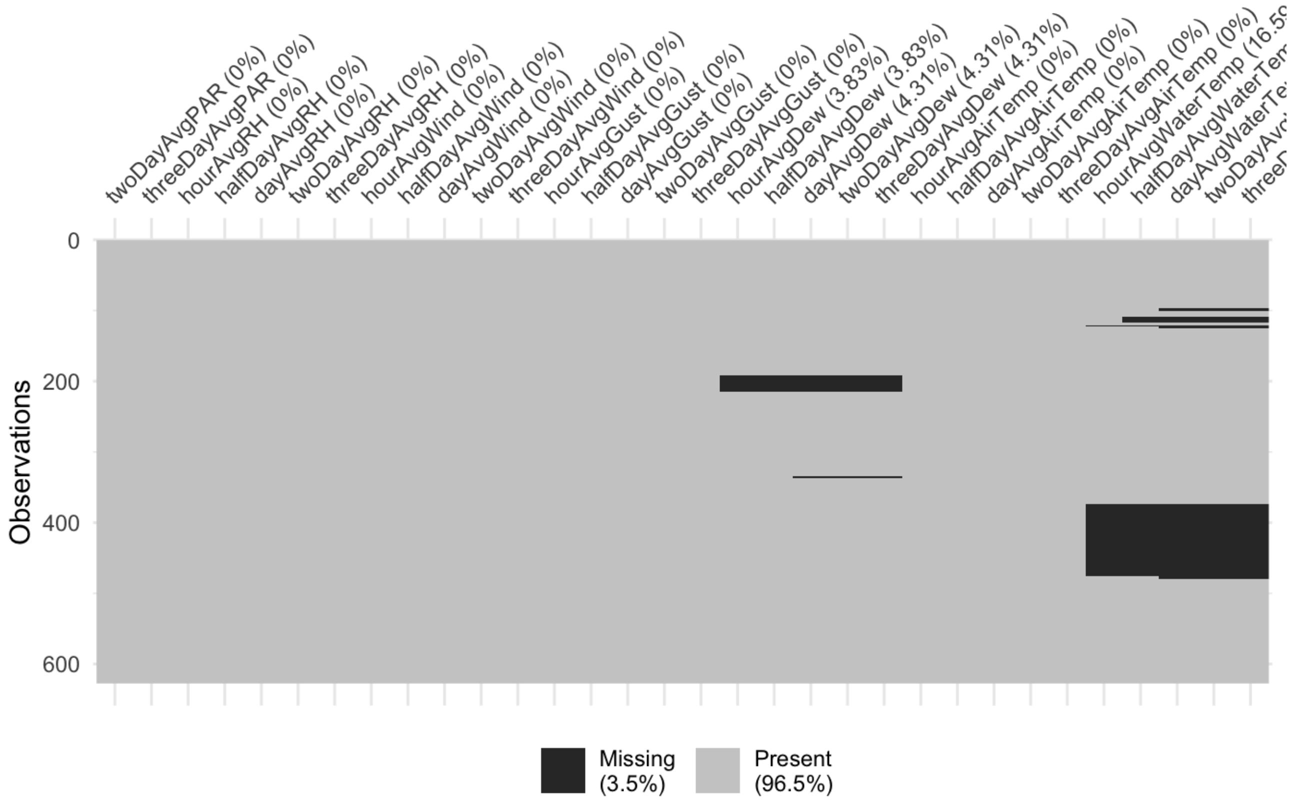
2.7.1 - NAs

First, we can use the naniar package (https://cran.r-project.org/web/packages/naniar/vignettes/naniar-visualisation.html) to look at NA values. The `vis_miss` graphs (which I split into two plots so the labels are easier to read) show that most of the data has 0% missing, but the DewPt data and the WaterTemp data have gaps. The `geom_miss_point` graphs show that the DewPt data is periodically missing in 2018 and 2019, and the WaterTemp data is periodically missing in 2017, and unavailable in 2020 and 2021.

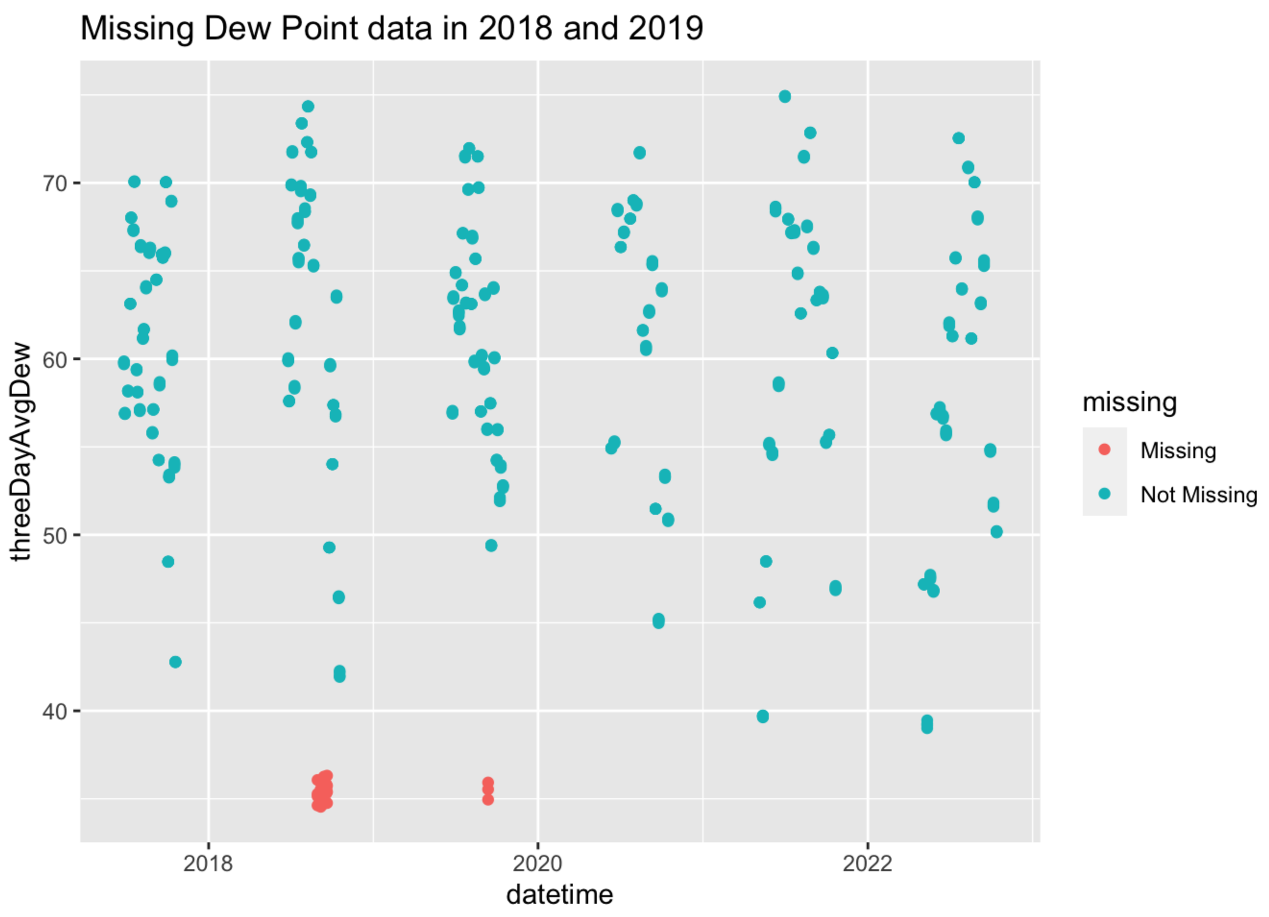
```
vis_miss(features[,1:32])
```



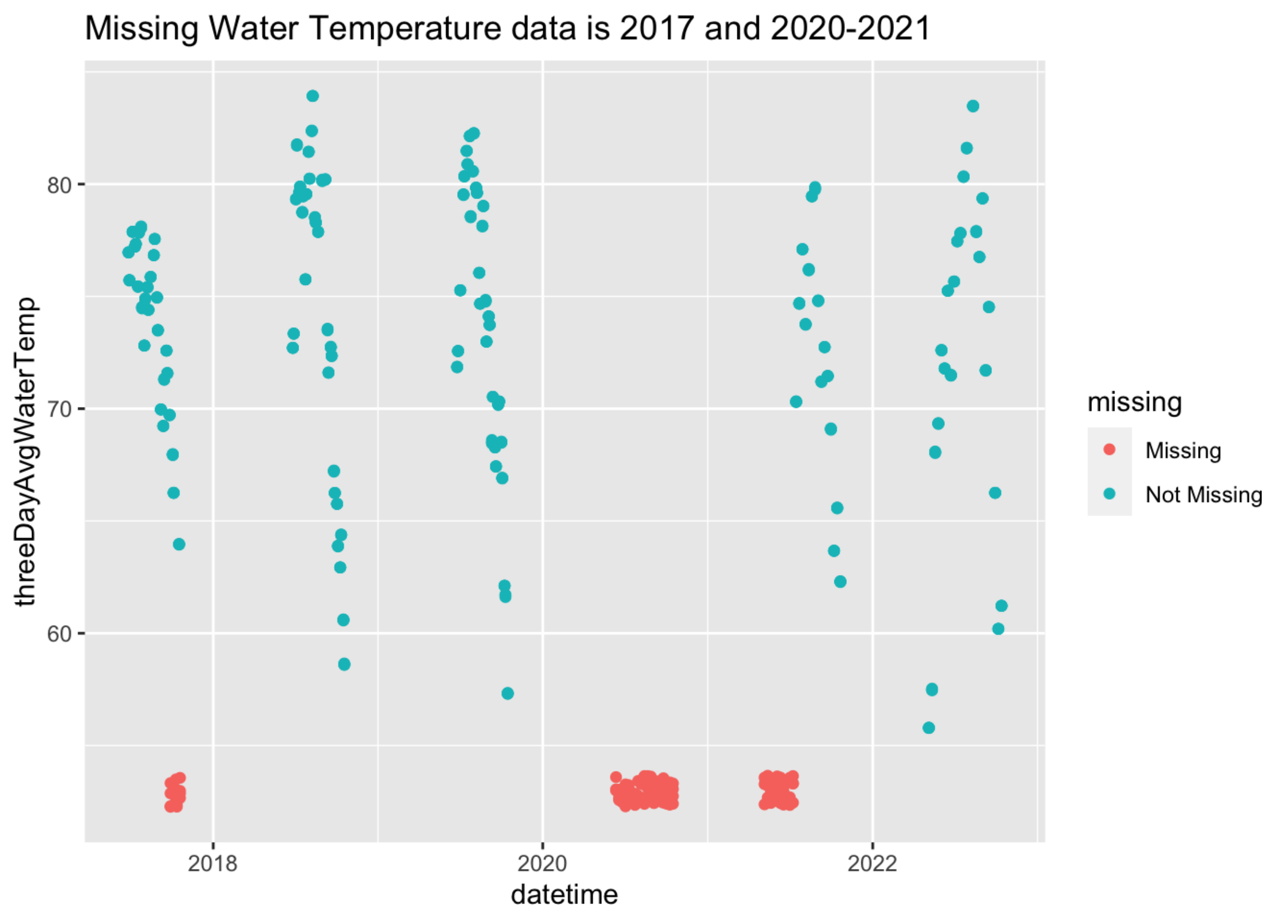
```
vis_miss(features[,33:64])
```



```
features %>% ggplot(aes(datetime, threeDayAvgDew)) +
  geom_miss_point() +
  ggtitle("Missing Dew Point data in 2018 and 2019")
```



```
features %>% ggplot(aes(datetime, threeDayAvgWaterTemp)) +
  geom_miss_point() +
  ggtitle("Missing Water Temperature data is 2017 and 2020-2021")
```



Based on these results,

- We will not be using the DewPt data, since it is not particularly relevant to E. coli.
- We want to use the waterTemp data, since it is relevant to E. coli, so we will remove the rows for which `threeDayAvgWaterTemp` is null.

```
ecoliData <- ecoliData %>%
  filter(!is.na(features$threeDayAvgWaterTemp))

features <- features %>%
  select(!c(hourAvgDew, halfDayAvgDew, dayAvgDew, twoDayAvgDew, threeDayAvgDew)) %>%
  filter(!is.na(features$threeDayAvgWaterTemp))
```

2.7.2 - Flawed Values

Just because the data exists doesn't mean that it is reliable. We should also check that the values are reasonable (no negative rainfalls, extreme temperatures, etc.) To do this, I created boxplots by year of every feature.

```
completionBoxPlots <- lapply(names(features)[3:59], function(colName) {
  ggplot(features, aes_string(x = "year", y = colName)) +
    geom_boxplot(outlier.color = "red", outlier.size = 3) +
    coord_flip()
})
```

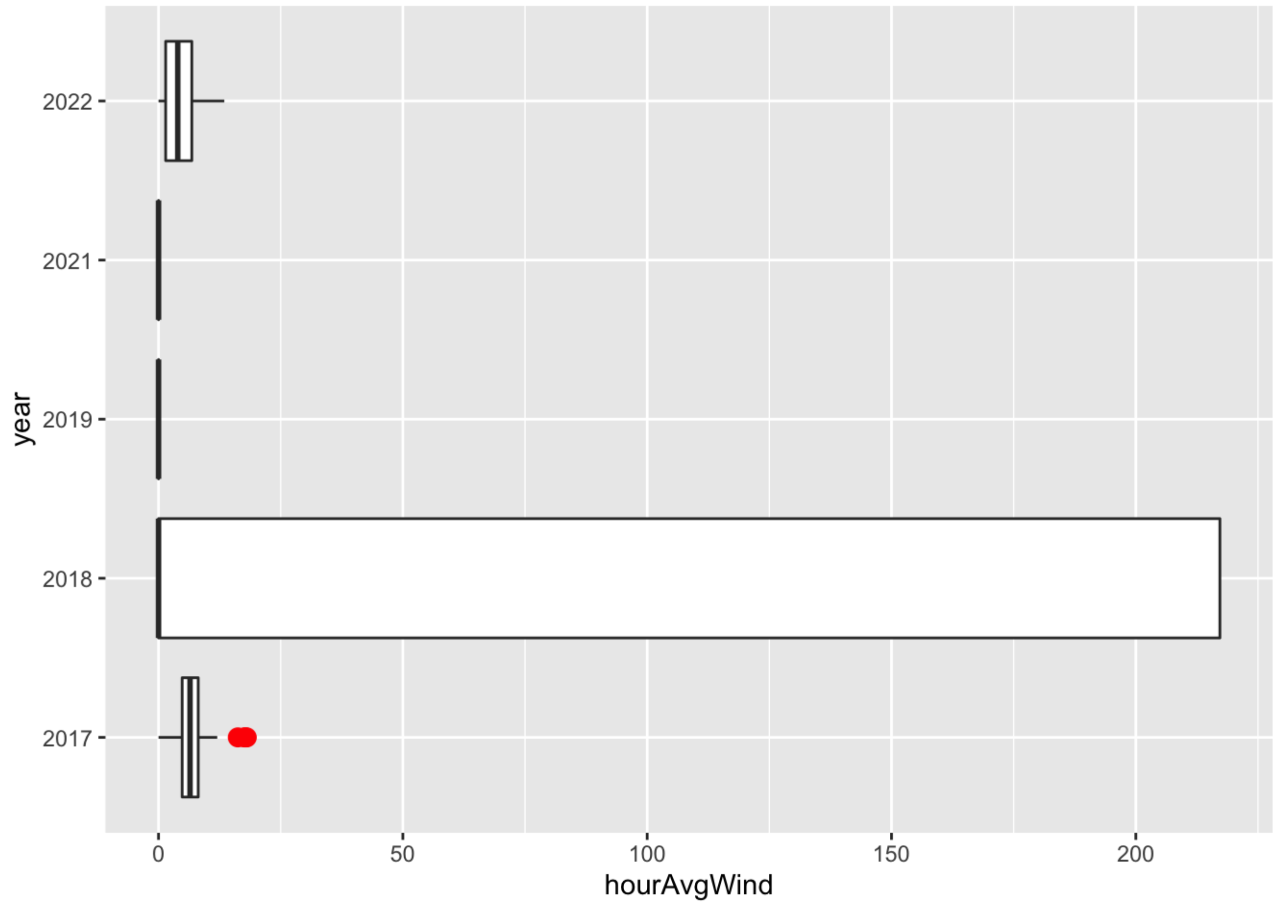
I don't include all 62 graphs in this report, but I will include the anomalies below.

First is the Wind and Gust data. From July to August of 2018, there are 39 rows where the wind values are around 217.2 mph, which is likely a device malfunction. Also, all of the data from 2019-2021 is 0.

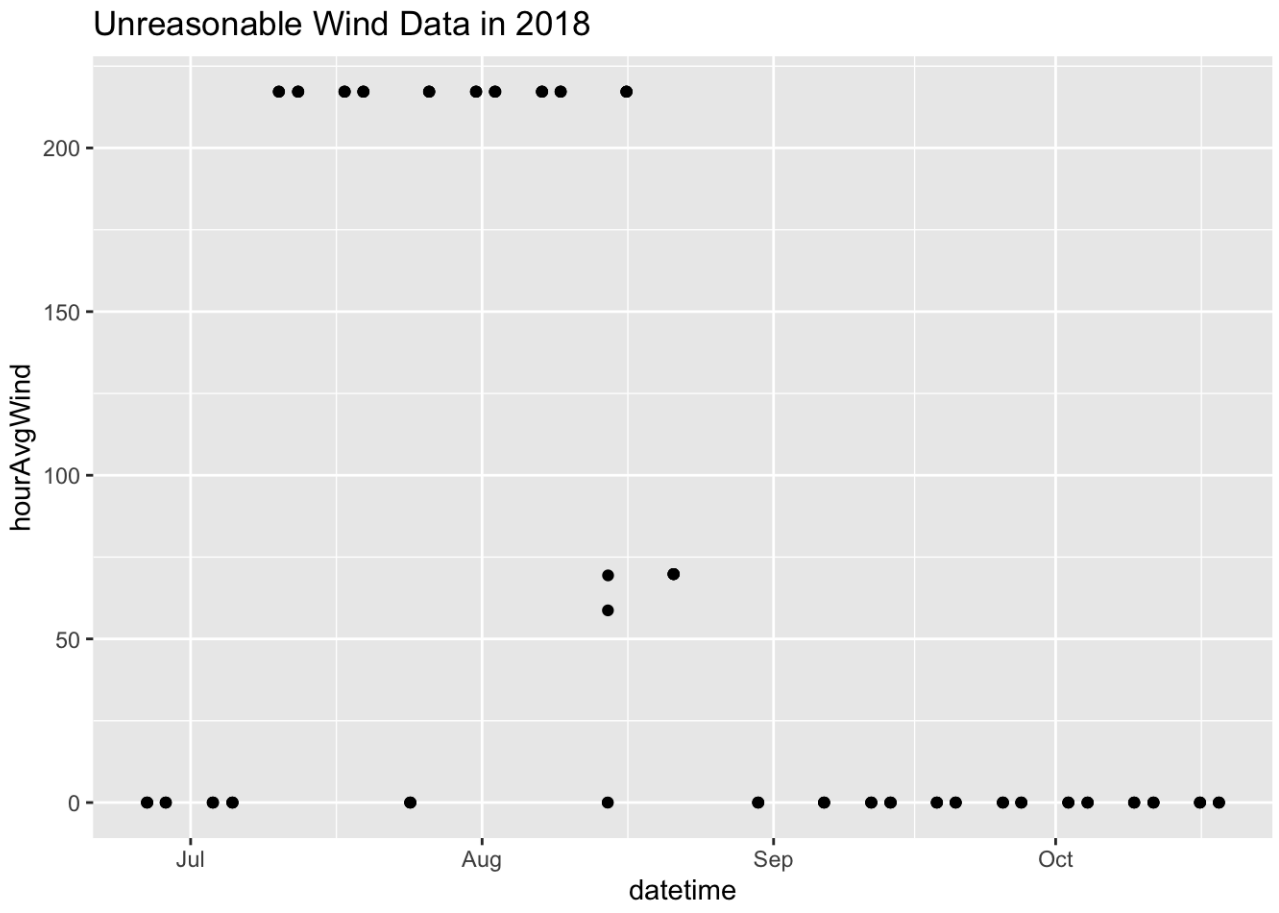
So, we will not consider wind speed or gust speed.

```
completionBoxPlots[which(names(features) == "hourAvgWind") - 2]
```

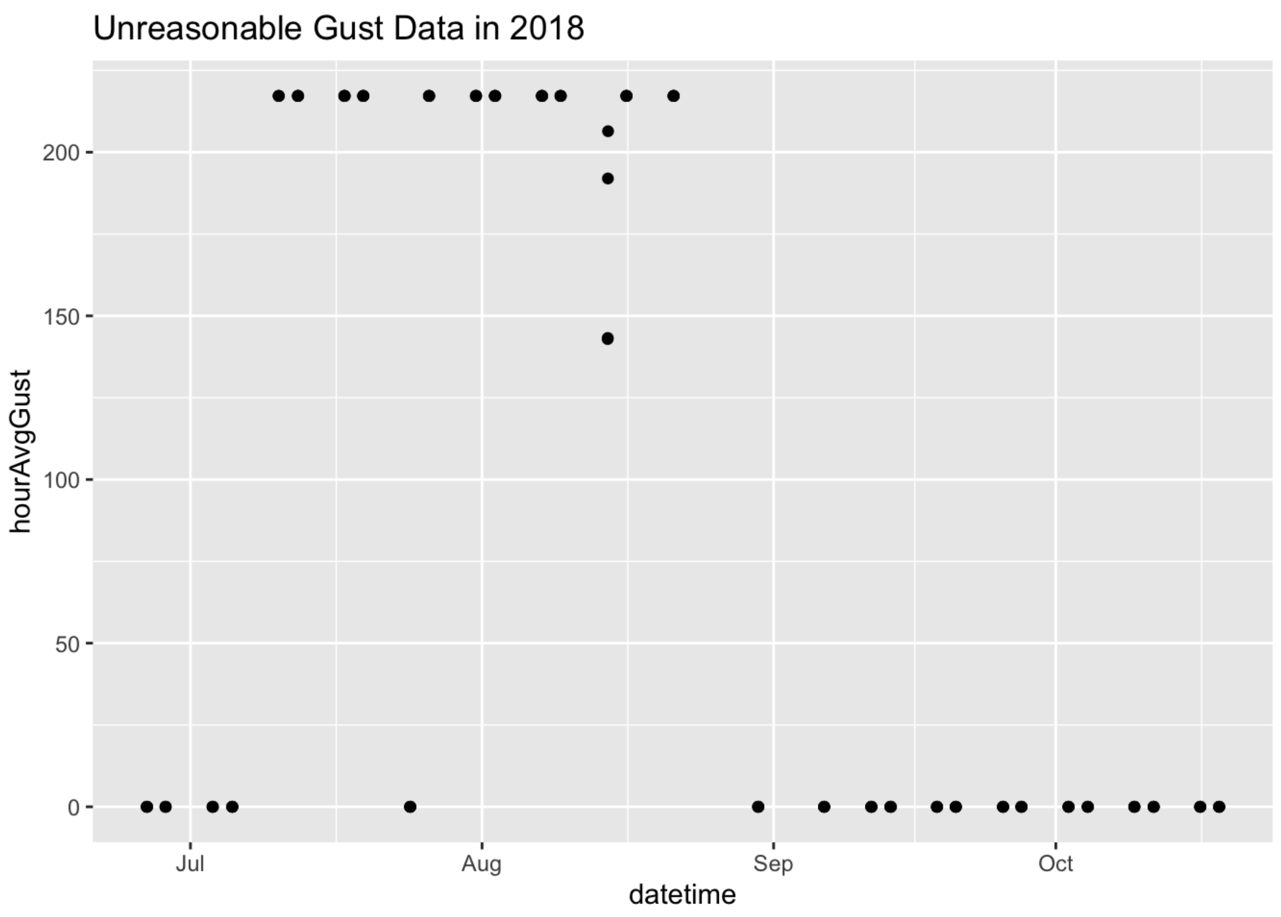
```
## [[1]]
```



```
features %>%
  filter(year == 2018) %>%
  ggplot(aes(datetime, hourAvgWind)) +
  geom_point() +
  ggtitle("Unreasonable Wind Data in 2018")
```



```
features %>%
  filter(year == 2018) %>%
  ggplot(aes(datetime, hourAvgGust)) +
  geom_point() +
  ggtitle("Unreasonable Gust Data in 2018")
```



```
features$datetime[which(features$hourAvgWind == max(features$hourAvgWind))]
```

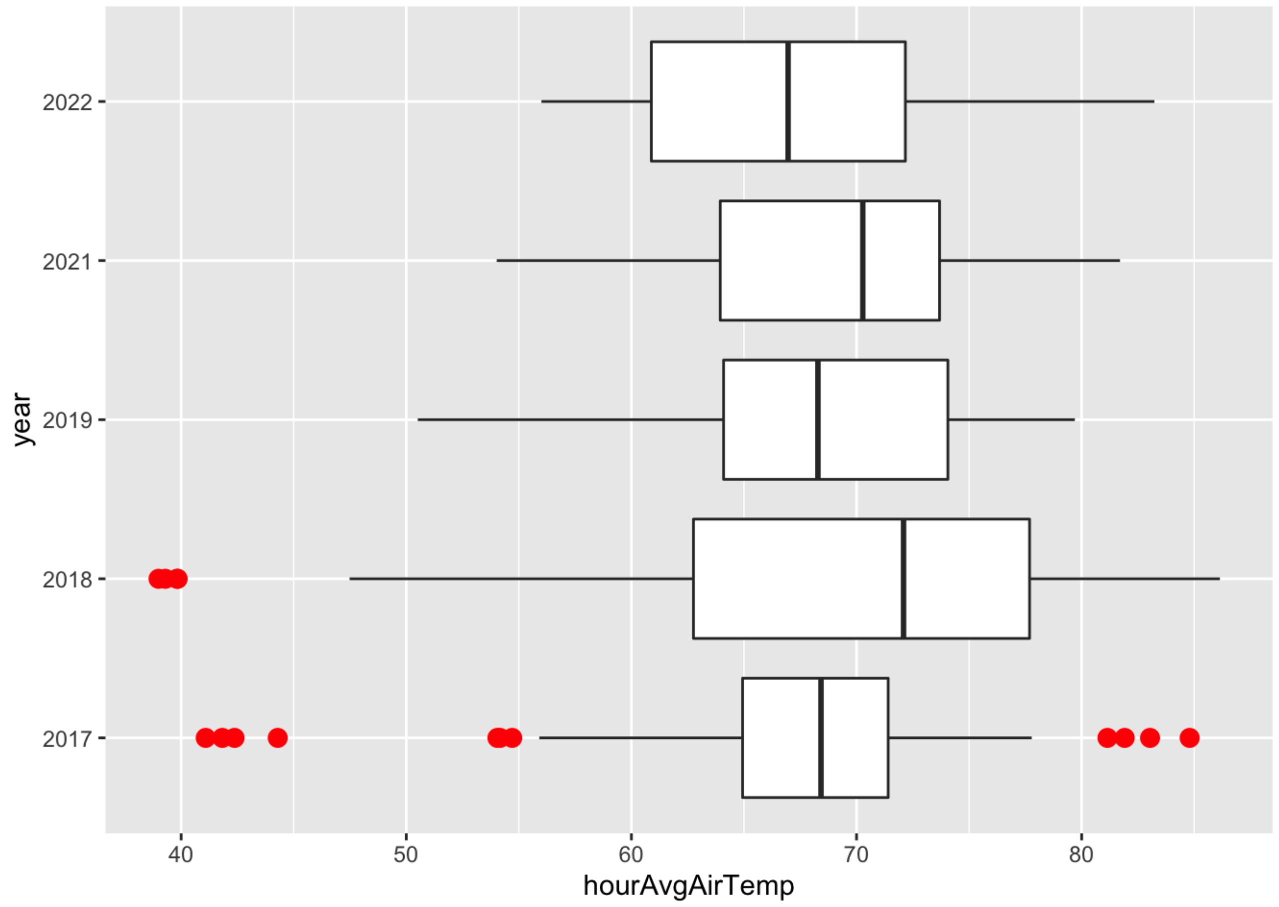
```
##  [1] "2018-07-10 09:40:00 EDT" "2018-07-10 09:10:00 EDT"
##  [3] "2018-07-10 08:50:00 EDT" "2018-07-12 11:00:00 EDT"
##  [5] "2018-07-12 10:30:00 EDT" "2018-07-12 10:10:00 EDT"
##  [7] "2018-07-12 09:50:00 EDT" "2018-07-17 09:40:00 EDT"
##  [9] "2018-07-17 09:10:00 EDT" "2018-07-17 08:40:00 EDT"
## [11] "2018-07-17 08:20:00 EDT" "2018-07-19 09:30:00 EDT"
## [13] "2018-07-19 09:00:00 EDT" "2018-07-19 08:40:00 EDT"
## [15] "2018-07-19 08:30:00 EDT" "2018-07-26 09:20:00 EDT"
## [17] "2018-07-26 08:50:00 EDT" "2018-07-26 08:40:00 EDT"
## [19] "2018-07-26 08:20:00 EDT" "2018-07-31 09:10:00 EDT"
## [21] "2018-07-31 08:40:00 EDT" "2018-07-31 08:20:00 EDT"
## [23] "2018-07-31 08:10:00 EDT" "2018-08-02 10:10:00 EDT"
## [25] "2018-08-02 09:00:00 EDT" "2018-08-02 08:40:00 EDT"
## [27] "2018-08-02 08:30:00 EDT" "2018-08-07 09:30:00 EDT"
## [29] "2018-08-07 09:00:00 EDT" "2018-08-07 08:40:00 EDT"
## [31] "2018-08-07 08:20:00 EDT" "2018-08-09 09:00:00 EDT"
## [33] "2018-08-09 08:30:00 EDT" "2018-08-09 08:20:00 EDT"
## [35] "2018-08-09 08:00:00 EDT" "2018-08-16 09:00:00 EDT"
## [37] "2018-08-16 08:30:00 EDT" "2018-08-16 08:20:00 EDT"
## [39] "2018-08-16 08:10:00 EDT"
```

```
features <- features %>%
  select(!c(hourAvgWind, halfDayAvgWind, dayAvgWind, twoDayAvgWind, threeDayAvgWind,
            hourAvgGust, halfDayAvgGust, dayAvgGust, twoDayAvgGust, threeDayAvgGust))
```

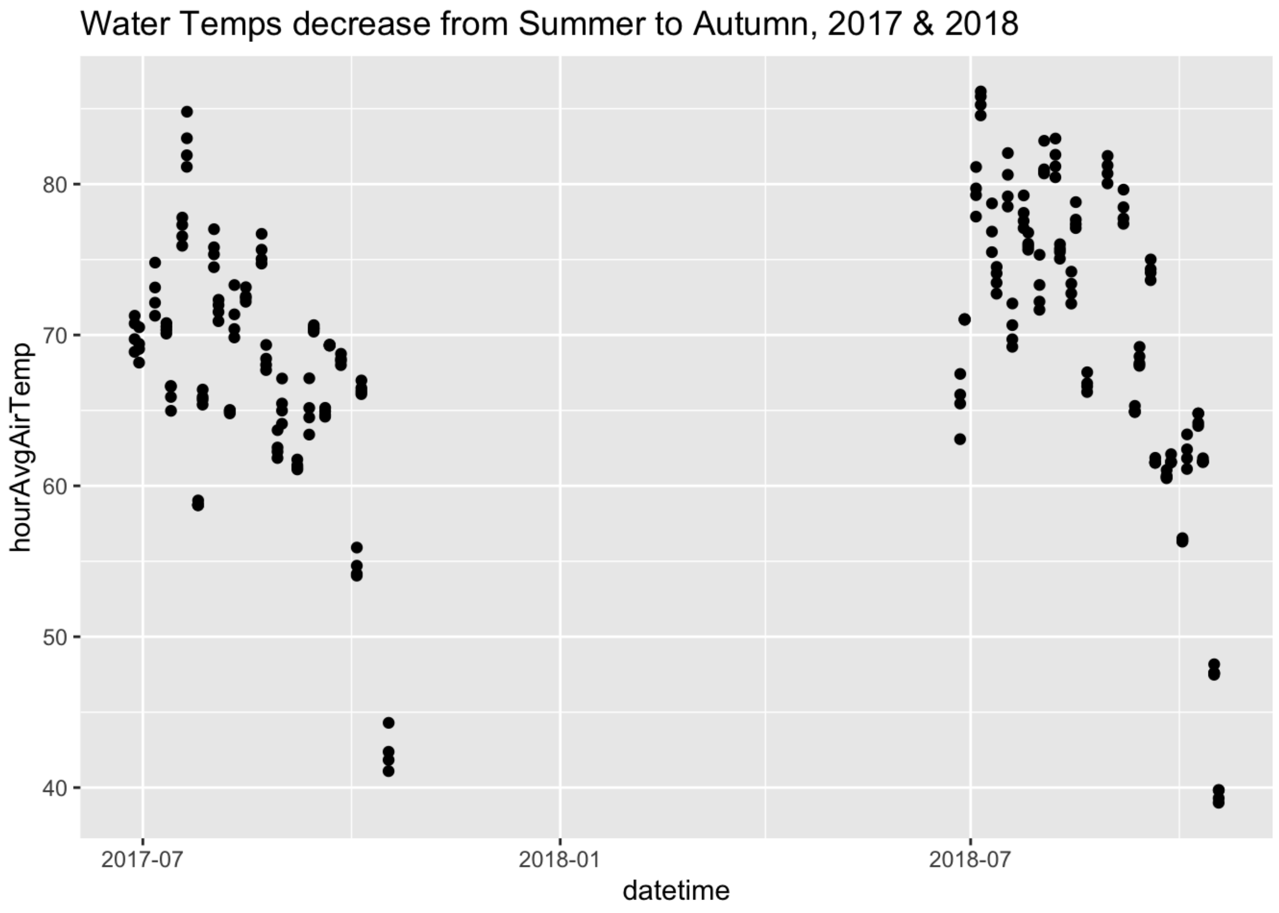
Second is the average air temperature, for which there are some low outliers in 2017 and 2018. This is expected, however, as October temperatures in Boston can be fairly cold. So, we will not remove any of this data.

```
completionBoxPlots[which(names(features) == "hourAvgAirTemp") - 2 + 10]
```

```
## [[1]]
```



```
features %>%
  filter(year < 2019) %>%
  ggplot(aes(datetime, hourAvgAirTemp)) +
  geom_point() +
  ggtitle("Water Temps decrease from Summer to Autumn, 2017 & 2018")
```



```
features$datetime[which(features$hourAvgAirTemp < 45)]
```

```
## [1] "2017-10-17 09:10:00 EDT" "2017-10-17 08:40:00 EDT"
## [3] "2017-10-17 08:30:00 EDT" "2017-10-17 08:10:00 EDT"
## [5] "2018-10-18 09:10:00 EDT" "2018-10-18 09:00:00 EDT"
## [7] "2018-10-18 08:30:00 EDT" "2018-10-18 08:10:00 EDT"
```

Section 3: Constructing Models

3.1 - GLM Model Setup

To prepare our analysis, I separate our data by site number, and remove the datetime/year columns. We will be constructing and evaluating 4 different models, for our 4 different sites.

The first model we will use is a GLM - a Generalized Linear Model, a type of logistic model. Instead of predicting the E. coli values, a GLM logistic model will make a binomial prediction - whether the E. Coli data is safe (below 630) or unsafe (above 630). So, we will cast this vector of numeric values to a vector of 0 (safe) or 1 (unsafe).

In my variable names, x refers to the predictors (weather, flow discharge) and y refers to the 0/1 E. coli safety levels.

```
indicesLoc1 <- which(ecoliData$Site_ID == "1NBS")
indicesLoc2 <- which(ecoliData$Site_ID == "2LARZ")
indicesLoc3 <- which(ecoliData$Site_ID == "3BU")
indicesLoc4 <- which(ecoliData$Site_ID == "4LONG")

xLoc1 <- features[indicesLoc1,-(1:2)]
xLoc2 <- features[indicesLoc2,-(1:2)]
xLoc3 <- features[indicesLoc3,-(1:2)]
xLoc4 <- features[indicesLoc4,-(1:2)]

safetyThreshold <- 630
ecoliData <- ecoliData %>%
  mutate(safety = ifelse(Reporting_Result < safetyThreshold, 0, 1))

yLoc1 <- ecoliData$safety[indicesLoc1]
yLoc2 <- ecoliData$safety[indicesLoc2]
yLoc3 <- ecoliData$safety[indicesLoc3]
yLoc4 <- ecoliData$safety[indicesLoc4]

dataLoc1 <- cbind(yLoc1, xLoc1)
dataLoc2 <- cbind(yLoc2, xLoc2)
dataLoc3 <- cbind(yLoc3, xLoc3)
dataLoc4 <- cbind(yLoc4, xLoc4)
```

In the below code, I randomly split the data into two partitions: a training set and a testing set. We will train the model on the former data, and test the model on the latter to evaluate its accuracy. This is to prevent **overtraining**, where our model is trained to work very well on the measured data, so the accuracy value we generate from this dataset may be unfairly high compared to data outside our sample.

Because the unsafe-ecoli data is sparse, I want to ensure that both the training and testing partitions have enough unsafe-ecoli data. So, I split these rows 50-50. (`rainWeek` is arbitrarily chosen in the code because the function `createDataPartition` acts on vectors, not data frames - `yLoc` cannot be used since it's homogeneous in those split safe/unsafe partitions.) The rest of the data is split 80-20, as is standard.

```
#Break the data into Unsafe and Safe

dataLoc1Unsafe <- dataLoc1 %>% filter(yLoc1 == 1)
dataLoc1Safe <- dataLoc1 %>% filter(yLoc1 == 0)
dataLoc2Unsafe <- dataLoc2 %>% filter(yLoc2 == 1)
dataLoc2Safe <- dataLoc2 %>% filter(yLoc2 == 0)
dataLoc3Unsafe <- dataLoc3 %>% filter(yLoc3 == 1)
dataLoc3Safe <- dataLoc3 %>% filter(yLoc3 == 0)
dataLoc4Unsafe <- dataLoc4 %>% filter(yLoc4 == 1)
dataLoc4Safe <- dataLoc4 %>% filter(yLoc4 == 0)

#Create data partitions

trainingIndicesLoc1Unsafe <- createDataPartition(dataLoc1Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc1Safe <- createDataPartition(dataLoc1Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc2Unsafe <- createDataPartition(dataLoc2Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc2Safe <- createDataPartition(dataLoc2Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc3Unsafe <- createDataPartition(dataLoc3Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc3Safe <- createDataPartition(dataLoc3Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc4Unsafe <- createDataPartition(dataLoc4Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc4Safe <- createDataPartition(dataLoc4Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

#rbind the data back together, using these partition indices

trainLoc1 <- rbind(dataLoc1Unsafe[trainingIndicesLoc1Unsafe,],
                   dataLoc1Safe[trainingIndicesLoc1Safe,])
testLoc1 <- rbind(dataLoc1Unsafe[-trainingIndicesLoc1Unsafe,],
                   dataLoc1Safe[-trainingIndicesLoc1Safe,])

trainLoc2 <- rbind(dataLoc2Unsafe[trainingIndicesLoc2Unsafe,],
                   dataLoc2Safe[trainingIndicesLoc2Safe,])
testLoc2 <- rbind(dataLoc2Unsafe[-trainingIndicesLoc2Unsafe,],
                   dataLoc2Safe[-trainingIndicesLoc2Safe,])

trainLoc3 <- rbind(dataLoc3Unsafe[trainingIndicesLoc3Unsafe,],
                   dataLoc3Safe[trainingIndicesLoc3Safe,])
testLoc3 <- rbind(dataLoc3Unsafe[-trainingIndicesLoc3Unsafe,],
                   dataLoc3Safe[-trainingIndicesLoc3Safe,])

trainLoc4 <- rbind(dataLoc4Unsafe[trainingIndicesLoc4Unsafe,],
                   dataLoc4Safe[trainingIndicesLoc4Safe,])
```

```
testLoc4 <- rbind(dataLoc4Unsafe[-trainingIndicesLoc4Unsafe,],
                  dataLoc4Safe[-trainingIndicesLoc4Safe,])
```

3.2 - Logistic Regression: binomial GLM

In this section, I use the Generalized Linear Model algorithm to create logit models on the training data.

For each location, I first use all available features, and calculate the AIC (Akaike Information Criterion). Then, I construct the ANOVA (Analysis of Variables) table, and choose the most significant variables to minimize the AIC.

Once I have the best model, I use the `caret::predict()` function, still on the training data, using type = "response" to get probabilities between 0 and 1. Without this parameter, the output is log odds. Using a selection of thresholds to convert this to either 0 or 1, I compare this to the actual observed E. coli using a Confusion Matrix, from which we can determine the TPR, FPR, TNR, and FNR. The value that we care about the most is "missed hits" - when the model predicts that the E. coli level is safe, but we observed that it was unsafe.

After choosing the best threshold using the training data, I then apply the best model with the best threshold to the Testing data, and create a final Confusion Matrix.

3.2.1 - Location 1 GLM

Using all variables, the AIC is 94. The ANOVA variables can vary by the random partitions, but the best variables at the time of my analysis are `dayAvgFlow`, `twoDayAvgPressure`, `halfDayAvgFlow`, `rain0_12`, and `daysSinceLastRain`.

A four-variable model (excluding `halfDayAvgFlow` since it's likely highly correlated with `dayAvgFlow`) yields an AIC of 35.9. A three-variable model (also excluding `twoDayAvgPressure` since pressure is known to be correlated with rain) yields an AIC of 36.5. I tested other combinations as well, but the four-variable model appears to be the best.

```
glmLoc1 <- glm(yLoc1 ~ ., family = binomial, data = trainLoc1)
glmLoc1$aic
```

```
## [1] 96
```

```
anovaLoc1 <- anova(glmLoc1, test = "Chisq")
anovaLoc1 %>% slice_min(`Pr(>Chi)`, n = 10)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: yLoc1
##
## Terms added sequentially (first to last)
##
##
##                 Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## rain48_60        1  22.0713        84      0.000 2.627e-06 ***
## dayAvgFlow       1  20.8280        91     51.871 5.024e-06 ***
## rain0_12         1  19.0506        88     27.116 1.273e-05 ***
## hourAvgFlow      1   4.9763        93     74.466   0.02570 *
## threeDayAvgFlow  1   4.3150        89     46.166   0.03778 *
## rain12_24        1   3.2335        87     23.882   0.07215 .
## halfDayAvgFlow   1   1.7665        92     72.699   0.18382
## twoDayAvgFlow    1   1.3903        90     50.481   0.23836
## rain36_48        1   0.9973        85     22.071   0.31797
## rain24_36        1   0.8135        86     23.069   0.36708
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fourVarGlmLoc1 <- glm(yLoc1 ~ dayAvgFlow + twoDayAvgPressure + rain0_12 + daysSinceLastRain,
                     family = binomial,
                     data = trainLoc1)
fourVarGlmLoc1$aic
```

```
## [1] 35.96206
```

```
threeVarGlmLoc1 <- glm(yLoc1 ~ dayAvgFlow + rain0_12 + daysSinceLastRain,
                     family = binomial,
                     data = trainLoc1)
threeVarGlmLoc1$aic
```

```
## [1] 36.53494
```

Using `sapply` to check various thresholds, I decided on 0.5 for this location.

```
trainPredictGlmLoc1 <- predict(fourVarGlmLoc1, trainLoc1, type = "response")
thresholdChoicesLoc1 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(trainPredictGlmLoc1 > x, 1, 0)),
    as.factor(trainLoc1$yLoc1))$table
  })
thresholdChoicesLoc1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   79   79   79   79   79   79   80   80   80    80    80    80    80    80
## [2,]    2    2    2    2    2    2    1    1    1     1     1     1     1     1
## [3,]    2    4    4    4    5    5    5    5    5     5     5     6     6     6
## [4,]   12   10   10   10    9    9    9    9    9     9     9     8     8     8
##      [,15] [,16]
## [1,]    80    80
## [2,]     1     1
## [3,]     7     7
## [4,]     7     7
```

```
thresholdLoc1 <- 0.5

testPredictGlmLoc1 <- predict(fourVarGlmLoc1, testLoc1, type = "response")
cmGlmLoc1 <- confusionMatrix(
  as.factor(ifelse(testPredictGlmLoc1 > thresholdLoc1, 1, 0)),
  as.factor(testLoc1$yLoc1))
cmGlmLoc1$table
```

```
##           Reference
## Prediction  0  1
##          0 18  5
##          1  1  7
```

Among the testing data, the accuracy is 81%, and the missed-hit rate is 42%.

3.2.2 - Location 2 GLM

Using all variables, the AIC is 96. At the time of my analysis, the best variables are `daysSinceLastMajorRain`, `hourAvgPressure`, several different Flow features, and `rain0_12`.

A four-variable model (using `dayAvgFlow` and the 3 specified variables above), a two-variable model using only Pressure and Rain, and a one-variable model using only Rain, all yield an AIC of 43.

```
glmLoc2 <- glm(yLoc2 ~ ., family = binomial, data = trainLoc2)
glmLoc2$aic
```

```
## [1] 96
```

```
anovaLoc2 <- anova(glmLoc2, test = "Chisq")
anovaLoc2 %>% slice_min(`Pr(>Chi)`, n = 10)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: yLoc2
##
## Terms added sequentially (first to last)
##
##
##                 Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## rain24_36        1  15.0335        91      0.000 0.0001056 ***
## rain12_24        1  10.5735        92     15.033 0.0011472 **
## rain0_12         1   9.6434        93     25.607 0.0019003 **
## hourAvgFlow      1   8.1640        98     47.590 0.0042730 **
## halfDayAvgFlow   1   5.4053        97     42.185 0.0200754 *
## dayAvgFlow       1   3.3170        96     38.868 0.0685661 .
## threeDayAvgFlow  1   2.1867        94     35.250 0.1392113
## twoDayAvgFlow    1   1.4305        95     37.437 0.2316798
## rain36_48        1   0.0000        90      0.000 0.9988836
## rain48_60        1   0.0000        89      0.000 0.9999498
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fourVarGlmLoc2 <- glm(yLoc2 ~ daysSinceLastMajorRain + hourAvgPressure + dayAvgFlow + rain0_
12,
                      family = binomial,
                      data = trainLoc2)
fourVarGlmLoc2$aic
```

```
## [1] 43.09682
```

```
twoVarGlmLoc2 <- glm(yLoc2 ~  hourAvgPressure + rain0_12,
                      family = binomial,
                      data = trainLoc2)
twoVarGlmLoc2$aic
```

```
## [1] 43.0339
```

```
oneVarGlmLoc2 <- glm(yLoc2 ~ rain0_12,
                     family = binomial,
                     data = trainLoc2)
oneVarGlmLoc2$aic
```

```
## [1] 43.57254
```

I decided to test all three of these models. For all of them, 0.6 was an acceptable criteria based on the training data.

```
fourVarTrainPredictGlmLoc2 <- predict(fourVarGlmLoc2, trainLoc2, type = "response")
fourVarThresholdChoicesLoc2 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(fourVarTrainPredictGlmLoc2 > x, 1, 0)),
    as.factor(trainLoc2$yLoc2))$table
  })
fourVarThresholdChoicesLoc2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   92   92   92   92   92   92   92   92   92    92    92    92    92    92
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    4    5    5    5    5    5    5    5    5     5     5     5     5     5
## [4,]    4    3    3    3    3    3    3    3    3     3     3     3     3     3
##      [,15] [,16]
## [1,]    92    92
## [2,]     0     0
## [3,]     5     5
## [4,]     3     3
```

```
twoVarTrainPredictGlmLoc2 <- predict(twoVarGlmLoc2, trainLoc2, type = "response")
twoVarThresholdChoicesLoc2 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(twoVarTrainPredictGlmLoc2 > x, 1, 0)),
    as.factor(trainLoc2$yLoc2))$table
  })
twoVarThresholdChoicesLoc2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   92   92   92   92   92   92   92   92   92    92    92    92    92    92
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    5    5    5    5    5    5    5    5    5     5     5     5     5     5
## [4,]    3    3    3    3    3    3    3    3    3     3     3     3     3     3
##      [,15] [,16]
## [1,]    92    92
## [2,]     0     0
## [3,]     5     5
## [4,]     3     3
```

```
oneVarTrainPredictGlmLoc2 <- predict(oneVarGlmLoc2, trainLoc2, type = "response")
oneVarThresholdChoicesLoc2 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(oneVarTrainPredictGlmLoc2 > x, 1, 0)),
    as.factor(trainLoc2$yLoc2))$table
  })
oneVarThresholdChoicesLoc2
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   91   91   92   92   92   92   92   92   92    92    92    92    92    92
## [2,]    1    1    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    5    5    5    5    5    5    5    5    5     5     5     5     6     6
## [4,]    3    3    3    3    3    3    3    3    3     3     3     3     2     2
##      [,15] [,16]
## [1,]    92    92
## [2,]     0     0
## [3,]     6     6
## [4,]     2     2
```

```
thresholdLoc2 <- 0.6

fourVarTestPredictGlmLoc2 <- predict(fourVarGlmLoc2, testLoc2, type = "response")
fourVarCmGlmLoc2 <- confusionMatrix(
  as.factor(ifelse(fourVarTestPredictGlmLoc2 > thresholdLoc2, 1, 0)),
  as.factor(testLoc2$yLoc2))
fourVarCmGlmLoc2$table
```

```
##           Reference
## Prediction  0  1
##          0 20  5
##          1  0  2
```

```
twoVarTestPredictGlmLoc2 <- predict(twoVarGlmLoc2, testLoc2, type = "response")
twoVarCmGlmLoc2 <- confusionMatrix(
  as.factor(ifelse(twoVarTestPredictGlmLoc2 > thresholdLoc2, 1, 0)),
  as.factor(testLoc2$yLoc2))
twoVarCmGlmLoc2$table
```

```
##           Reference
## Prediction  0  1
##          0 20  5
##          1  0  2
```

```
oneVarTestPredictGlmLoc2 <- predict(oneVarGlmLoc2, testLoc2, type = "response")
oneVarCmGlmLoc2 <- confusionMatrix(
  as.factor(ifelse(oneVarTestPredictGlmLoc2 > thresholdLoc2, 1, 0)),
  as.factor(testLoc2$yLoc2))
oneVarCmGlmLoc2$table
```

```
##           Reference
## Prediction  0  1
##          0 20  5
##          1  0  2
```

All three models behaved in the same way with regards to the testing data. The accuracy is 81%, and the ***missed-hit rate is 71%.***

3.2.3 - Location 3 GLM

Using all variables, the AIC is 94. At the time of my analysis, the best variables are `daysSinceLastMajorRain`, `dayAvgFlow`, `halfDayAvgFlow`, `hourAvgFlow`, and `rain24_36`.

A three-variable model (only using `dayAvgFlow` among the three flow features) yields an AIC of 57.9. Other models yield similar AICs, but to preserve the accuracy I will use this three-variable model.

```
glmLoc3 <- glm(yLoc3 ~ ., family = binomial, data = trainLoc3)
glmLoc3$aic
```

```
## [1] 94
```

```
anovaLoc3 <- anova(glmLoc3, test = "Chisq")
anovaLoc3 %>% slice_min(`Pr(>Chi)`, n = 10)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: yLoc3
##
## Terms added sequentially (first to last)
##
##
##                   Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## rainWeek           1   720.87        78     504.61 < 2.2e-16 ***
## daysSinceLastRain  1   504.61        77       0.00 < 2.2e-16 ***
## dayAvgFlow         1     8.21        94      46.76  0.004170 **
## threeDayAvgFlow    1     7.64        92      36.23  0.005708 **
## rain24_36          1     5.16        89      29.11  0.023096 *
## rain72_84          1     3.82        85      22.02  0.050779 .
## twoDayAvgFlow      1     2.90        93      43.87  0.088722 .
## rain48_60          1     2.76        87      25.83  0.096730 .
## rain12_24          1     1.95        90      34.27  0.162398
## rain84_96          1     1.32        84      20.69  0.250074
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
threeVarGlmLoc3 <- glm(yLoc3 ~ daysSinceLastMajorRain + dayAvgFlow + rain24_36,
                       family = binomial,
                       data = trainLoc3)
threeVarGlmLoc3$aic
```

```
## [1] 57.91771
```

When checking various thresholds, I noticed that only one sample was identified as unsafe.

```
trainPredictGlmLoc3 <- predict(threeVarGlmLoc3, trainLoc3, type = "response")
thresholdChoicesLoc3 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(trainPredictGlmLoc3 > x, 1, 0)),
    as.factor(trainLoc3$yLoc3))$table
  })
thresholdChoicesLoc3
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   90   90   90   90   90   90   90   90   90    90    90    90    90    90
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    7    7    7    7    7    7    7    7    7     7     7     7     7     7
## [4,]    1    1    1    1    1    1    1    1    1     1     1     1     1     1
##      [,15] [,16]
## [1,]    90    90
## [2,]     0     0
## [3,]     7     7
## [4,]     1     1
```

Setting a threshold of 0.5, the three-variable model **never identified an unsafe sample in the test data.**

Even when using all 47 variables, **the missed-hit rate is 63%.**

```
thresholdLoc3 <- 0.5
testPredictGlmLoc3 <- predict(threeVarGlmLoc3, testLoc3, type = "response")
cmGlmLoc3 <- confusionMatrix(
  as.factor(ifelse(testPredictGlmLoc3 > thresholdLoc3, 1, 0)),
  as.factor(testLoc3$yLoc3))
cmGlmLoc3$table
```

```
##           Reference
## Prediction  0  1
##          0 21  8
##          1  0  0
```

```
predictFull <- predict(glmLoc3, testLoc3, type = "response")
fullCmGlmLoc3 <- confusionMatrix(
  as.factor(ifelse(predictFull > thresholdLoc3, 1, 0)),
  as.factor(testLoc3$yLoc3))
fullCmGlmLoc3$table
```

```
##           Reference
## Prediction  0  1
##          0 20  5
##          1  1  3
```

3.2.4 - Location 4 GLM

Preface - I would not expect a GLM to be accurate for Location 4. Out of 125 observations, only 3 had dangerous levels of E. coli. 2 are in the training set, and 1 is in the test set.

Using all variables, the AIC is 96. The only reliable variables are the Flow variables.

A four-variable model using only Flow variables yields a better AIC of 10. A one-variable model, using the best of these (`twoDayAvgFlow`) yields an AIC of 23.3.

```
glmLoc4 <- glm(yLoc4 ~ ., family = binomial, data = trainLoc4)
glmLoc4$aic
```

```
## [1] 96
```

```
anovaLoc4 <- anova(glmLoc4, test = "Chisq")
anovaLoc4 %>% slice_min(`Pr(>Chi)`, n = 10)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: yLoc4
##
## Terms added sequentially (first to last)
##
##
##                 Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## twoDayAvgFlow    1  10.7896        95      0.000 0.001021 **
## hourAvgFlow      1   6.0220        98     13.586 0.014129 *
## halfDayAvgFlow   1   2.6802        97     10.906 0.101600
## dayAvgFlow       1   0.1160        96     10.790 0.733465
## rain12_24        1   0.0000        92      0.000 0.999973
## rain72_96        1   0.0000        82      0.000 0.999974
## threeDayAvgFlow  1   0.0000        94      0.000 0.999980
## rain24_36        1   0.0000        91      0.000 0.999983
## rain0_12         1   0.0000        93      0.000 0.999989
## dayAvgPressure   1   0.0000        74      0.000 0.999990
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fourVarGlmLoc4 <- glm(yLoc4 ~ hourAvgFlow + halfDayAvgFlow + dayAvgFlow + twoDayAvgFlow,
                       family = binomial,
                       data = trainLoc4)
fourVarGlmLoc4$aic
```

```
## [1] 10
```

```
oneVarGlmLoc4 <- glm(yLoc4 ~ twoDayAvgFlow,
                     family = binomial,
                     data = trainLoc4)
oneVarGlmLoc4$aic
```

```
## [1] 23.31041
```

Using `sapply` to check various thresholds, a threshold of 0.5 is appropriate. The model is able to flag these two hits.

```
trainPredictGlmLoc4 <- predict(fourVarGlmLoc4, trainLoc4, type = "response")
thresholdChoicesLoc4 <- sapply(seq(0.5, 0.65, 0.01), function(x) {
  confusionMatrix(
    as.factor(ifelse(trainPredictGlmLoc4 > x, 1, 0)),
    as.factor(trainLoc4$yLoc4))$table
  })
thresholdChoicesLoc4
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   98   98   98   98   98   98   98   98   98    98    98    98    98    98
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [4,]    2    2    2    2    2    2    2    2    2     2     2     2     2     2
##      [,15] [,16]
## [1,]    98    98
## [2,]     0     0
## [3,]     0     0
## [4,]     2     2
```

```
thresholdLoc4 <- 0.5

testPredictGlmLoc4 <- predict(fourVarGlmLoc4, testLoc4, type = "response")
cmGlmLoc4 <- confusionMatrix(
  as.factor(ifelse(testPredictGlmLoc4 > thresholdLoc4, 1, 0)),
  as.factor(testLoc4$yLoc4))
cmGlmLoc4$table
```

```
##           Reference
## Prediction  0  1
##          0 24  1
##          1  0  0
```

In the testing data, however, the model is unable to flag the one hit.

3.3 - MLR Model Setup

After the results of the GLM Regression Models, I wanted to see if I could create a better model using MLR Regression. In this type of model, I take the `log()` of the original observations, calculate a normal linear regression using `lm()`, and convert to 0/1 afterwards to calculate the Confusion Matrix.

First, however, I'll need to redefine the y parts of the model.

```
yLoc1 <- log(ecoliData$Reporting_Result[indicesLoc1])
yLoc2 <- log(ecoliData$Reporting_Result[indicesLoc2])
yLoc3 <- log(ecoliData$Reporting_Result[indicesLoc3])
yLoc4 <- log(ecoliData$Reporting_Result[indicesLoc4])

dataLoc1 <- cbind(yLoc1, xLoc1)
dataLoc2 <- cbind(yLoc2, xLoc2)
dataLoc3 <- cbind(yLoc3, xLoc3)
dataLoc4 <- cbind(yLoc4, xLoc4)
```

I still want to ensure that enough high E. coli river conditions are in both the training and the testing data. So, for each location, I partition the data 50-50 for the top 25% concentrations, and 80-20 for the rest.

```
dataLoc1Unsafe <- dataLoc1 %>% filter(yLoc1 >= quantile(yLoc1)[4])
dataLoc1Safe <- dataLoc1 %>% filter(yLoc1 < quantile(yLoc1)[4])
dataLoc2Unsafe <- dataLoc2 %>% filter(yLoc2 >= quantile(yLoc2)[4])
dataLoc2Safe <- dataLoc2 %>% filter(yLoc2 < quantile(yLoc2)[4])
dataLoc3Unsafe <- dataLoc3 %>% filter(yLoc3 >= quantile(yLoc3)[4])
dataLoc3Safe <- dataLoc3 %>% filter(yLoc3 < quantile(yLoc3)[4])
dataLoc4Unsafe <- dataLoc4 %>% filter(yLoc4 >= quantile(yLoc4)[4])
dataLoc4Safe <- dataLoc4 %>% filter(yLoc4 < quantile(yLoc4)[4])
```

The rest of the code is the same as before.

```
#Create data partitions

trainingIndicesLoc1Unsafe <- createDataPartition(dataLoc1Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc1Safe <- createDataPartition(dataLoc1Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc2Unsafe <- createDataPartition(dataLoc2Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc2Safe <- createDataPartition(dataLoc2Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc3Unsafe <- createDataPartition(dataLoc3Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc3Safe <- createDataPartition(dataLoc3Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

trainingIndicesLoc4Unsafe <- createDataPartition(dataLoc4Unsafe$rainWeek, times = 1, p = 0.
5, list = FALSE)
trainingIndicesLoc4Safe <- createDataPartition(dataLoc4Safe$rainWeek, times = 1, p = 0.8, li
st = FALSE)

#rbind the data back together, using these partition indices

trainLoc1 <- rbind(dataLoc1Unsafe[trainingIndicesLoc1Unsafe,],
                   dataLoc1Safe[trainingIndicesLoc1Safe,])
testLoc1 <- rbind(dataLoc1Unsafe[-trainingIndicesLoc1Unsafe,],
                   dataLoc1Safe[-trainingIndicesLoc1Safe,])

trainLoc2 <- rbind(dataLoc2Unsafe[trainingIndicesLoc2Unsafe,],
                   dataLoc2Safe[trainingIndicesLoc2Safe,])
testLoc2 <- rbind(dataLoc2Unsafe[-trainingIndicesLoc2Unsafe,],
                   dataLoc2Safe[-trainingIndicesLoc2Safe,])

trainLoc3 <- rbind(dataLoc3Unsafe[trainingIndicesLoc3Unsafe,],
                   dataLoc3Safe[trainingIndicesLoc3Safe,])
testLoc3 <- rbind(dataLoc3Unsafe[-trainingIndicesLoc3Unsafe,],
                   dataLoc3Safe[-trainingIndicesLoc3Safe,])

trainLoc4 <- rbind(dataLoc4Unsafe[trainingIndicesLoc4Unsafe,],
                   dataLoc4Safe[trainingIndicesLoc4Safe,])
testLoc4 <- rbind(dataLoc4Unsafe[-trainingIndicesLoc4Unsafe,],
                   dataLoc4Safe[-trainingIndicesLoc4Safe,])
```

3.4 - Multivariate Regression: logarithmic MLR

Our goal with the MLR model is to predict when the E. coli concentration is above 1260 CFU/100mL. This is the statistical threshold value above which we deem the water unsafe.

In order to create a conservative and safe model, I will implement a threshold of 235 CFU/100mL (the single sample limit for freshwater primary contact recreation, designated swimming area) in the creation of my MLR models. These thresholds will determine how we convert the predictions to a binary safe/unsafe.

For each location, I calculated the R-squared value for the model using all variables, and then use `varImp()` to select a reasonable subset of those variables. Then, I use the above thresholds to construct a Confusion Matrix. Again, we care most about minimizing the missed-hits rate.

3.4.1 - Location 1 MLR

Using all variables, the model has an R-squared of 0.5057.

A two-variable model using `rain0_12` and `hourAvgFlow` has an adjusted R-squared of 0.3105.

```
mlrLoc1 <- train(yLoc1 ~ ., data = trainLoc1, method = "lm")
varImp(mlrLoc1)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 47)
##
##                       Overall
## rain0_12               100.00
## hourAvgFlow             76.44
## daysSinceLastRain       39.26
## halfDayAvgAirTemp       35.29
## twoDayAvgPressure       27.44
## dayAvgFlow              26.23
## dayAvgPressure          25.92
## hourAvgAirTemp          25.63
## threeDayAvgRH           25.31
## threeDayAvgWaterTemp    24.46
## halfDayAvgPAR           24.05
## threeDayAvgAirTemp      23.54
## rain48_96               22.95
## hourAvgWaterTemp        22.09
## twoDayAvgRH             21.27
## halfDayAvgPressure      20.16
## threeDayAvgPAR          19.26
## rainWeek                19.09
## rain60_72               18.05
## hourAvgPressure         17.51
```

```
twoVarMlrLoc1 <- train(yLoc1 ~ rain0_12 + hourAvgFlow,
                    data = trainLoc1,
                    method = "lm")
summary(twoVarMlrLoc1)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.58628 -0.58504 -0.08267  0.22992  2.85079
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5.1681833  0.1276673  40.482  < 2e-16 ***
## rain0_12    4.6625798  0.7185292   6.489 4.59e-09 ***
## hourAvgFlow 0.0006113  0.0004202   1.455    0.149
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8768 on 90 degrees of freedom
## Multiple R-squared:  0.3255, Adjusted R-squared:  0.3105
## F-statistic: 21.72 on 2 and 90 DF,  p-value: 2.011e-08
```

I test this model on the testing data, and calculate a Confusion Matrix using the thresholds from above.

The accuracy level is 88%, and there were 0% missed hits.

```
testTwoVarMlrLoc1 <- 5.1681832 + (4.6625787 * testLoc1$rain0_12) + (0.0006113 * testLoc1$hou
rAvgFlow)

cmMlrLoc1 <- confusionMatrix(
  as.factor(ifelse(testTwoVarMlrLoc1 > log(235), 1, 0)),
  as.factor(ifelse(testLoc1$yLoc1 > log(1260), 1, 0))
)

cmMlrLoc1$table
```

```
##           Reference
## Prediction  0  1
##          0 21  0
##          1  4  8
```

3.4.2 - Location 2 MLR

Using all variables, the model has an R-squared of 0.4469.

A three-variable model using `rain0_12` , `threeDayAvgAirTemp` , and `daysSinceLastMajorRain` has an adjusted R-squared of 0.1619.

```
mlrLoc2 <- train(yLoc2 ~ ., data = trainLoc2, method = "lm")
varImp(mlrLoc2)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                        Overall
## rain0_12                100.00
## threeDayAvgAirTemp       84.77
## daysSinceLastMajorRain   65.11
## hourAvgFlow              59.35
## rain24_48                55.25
## rain48_60                53.28
## rain36_48                52.42
## hourAvgRH                51.48
## hourAvgPressure          45.66
## rain60_72                43.52
## rain0_48                 43.46
## rain72_84                43.15
## twoDayAvgAirTemp         43.14
## hourAvgWaterTemp         42.19
## halfDayAvgRH             39.09
## rain84_96                38.41
## halfDayAvgAirTemp        37.81
## threeDayAvgPAR           35.34
## twoDayAvgRH              34.65
## hourAvgAirTemp           34.12
```

```
threeVarMlrLoc2 <- train(yLoc2 ~ rain0_12 + threeDayAvgAirTemp + daysSinceLastMajorRain,
                         data = trainLoc2,
                         method = "lm")
summary(threeVarMlrLoc2)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8102 -0.5183 -0.0291  0.4905  4.7791
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             5.501886   0.984510   5.588 2.37e-07 ***
## rain0_12                2.997021   0.730157   4.105 8.82e-05 ***
## threeDayAvgAirTemp     -0.014088   0.013888  -1.014   0.3131
## daysSinceLastMajorRain  0.003538   0.001750   2.022   0.0461 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.028 on 91 degrees of freedom
## Multiple R-squared:  0.1887, Adjusted R-squared:  0.1619
## F-statistic: 7.054 on 3 and 91 DF,  p-value: 0.000257
```

On the testing data, the accuracy level is 94%, and there were 33% missed hits.

```
testThreeVarMlrLoc2 <- 5.501886 + (2.997021 * testLoc2$rain0_12) + (-0.014088 * testLoc2$thr
eeDayAvgAirTemp) + (0.003538 * testLoc2$daysSinceLastMajorRain)

cmMlrLoc2 <- confusionMatrix(
  as.factor(ifelse(testThreeVarMlrLoc2 > log(235), 1, 0)),
  as.factor(ifelse(testLoc2$yLoc2 > log(1260), 1, 0))
)
cmMlrLoc2$table
```

```
##           Reference
## Prediction  0  1
##          0 26  2
##          1  0  4
```

3.4.3 - Location 3 MLR

Using all variables, the model has an R-squared of 0.3712.

A three-variable model using `halfDayAvgFlow`, `dayAvgFlow`, and `rain0_12` has an adjusted R-squared of 0.3952.

```
mlrLoc3 <- train(yLoc3 ~ ., data = trainLoc3, method = "lm")
varImp(mlrLoc3)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 47)
##
##                      Overall
## halfDayAvgFlow        100.00
## dayAvgFlow             88.94
## rain0_12               67.78
## daysSinceLastRain      63.26
## halfDayAvgRH           53.50
## rain24_48              51.67
## dayAvgPressure         51.23
## twoDayAvgPressure      50.62
## rain0_48               48.26
## threeDayAvgPAR         48.00
## rain72_84              44.06
## rain60_72              43.91
## threeDayAvgWaterTemp   42.57
## hourAvgFlow            41.46
## halfDayAvgPressure     40.26
## twoDayAvgFlow          38.98
## hourAvgPAR             33.78
## rain48_72              29.59
## twoDayAvgWaterTemp     27.96
## rain48_96              26.36
```

```
threeVarMlrLoc3 <- train(yLoc3 ~ halfDayAvgFlow + dayAvgFlow + rain0_12,
                      data = trainLoc3,
                      method = "lm")
summary(threeVarMlrLoc3)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4820 -0.5512 -0.0102  0.5406  2.6197
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     4.380013   0.147400  29.715  < 2e-16 ***
## halfDayAvgFlow  0.011368   0.003212   3.540 0.000634 ***
## dayAvgFlow     -0.010225   0.003297  -3.101 0.002568 **
## rain0_12        3.765905   1.142571   3.296 0.001399 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.001 on 91 degrees of freedom
## Multiple R-squared:  0.4145, Adjusted R-squared:  0.3952
## F-statistic: 21.48 on 3 and 91 DF,  p-value: 1.322e-10
```

For this Confusion Matrix, the accuracy level is 88% and there were 50% missed hits.

```
testThreeVarMlrLoc3 <- 4.380013 + (0.011368 * testLoc3$halfDayAvgFlow) + (-0.010225 * testLo
c3$dayAvgFlow) + (3.765905 * testLoc3$rain0_12)

cmMlrLoc3 <- confusionMatrix(
  as.factor(ifelse(testThreeVarMlrLoc3 > log(235), 1, 0)),
  as.factor(ifelse(testLoc3$yLoc3 > log(1260), 1, 0))
)
cmMlrLoc3$table
```

```
##           Reference
## Prediction  0  1
##          0 26  2
##          1  2  2
```

3.4.4 - Location 4 MLR

Using all variables, the model has an R-squared of 0.3822.

A four-variable model using `rain48_96`, `hourAvgFlow`, `rainWeek`, and `rain0_12` has an adjusted R-squared of 0.1606.

```
mlrLoc4 <- train(yLoc4 ~ ., data = trainLoc4, method = "lm")
varImp(mlrLoc4)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 47)
##
##                     Overall
## rain48_96            100.00
## hourAvgFlow           86.06
## rainWeek              85.45
## rain0_12              82.13
## hourAvgRH             70.28
## halfDayAvgAirTemp     62.94
## rain48_60             59.21
## rain36_48             57.18
## hourAvgPressure       56.83
## rain72_96             55.52
## threeDayAvgFlow       49.60
## hourAvgAirTemp        48.63
## halfDayAvgRH          48.47
## threeDayAvgAirTemp    41.98
## rain0_48              40.64
## halfDayAvgWaterTemp   39.08
## halfDayAvgFlow        38.21
## halfDayAvgPressure    35.31
## hourAvgPAR            33.89
## rain72_84             33.66
```

```
fourVarMlrLoc4 <- train(yLoc4 ~ rain48_96 + hourAvgFlow + rainWeek + rain0_12,
                        data = trainLoc4,
                        method = "lm")
summary(fourVarMlrLoc4)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1728 -0.7051 -0.0476  0.6740  3.5837
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.0615367  0.1611623  18.997  < 2e-16 ***
## rain48_96   -0.5564919  0.3293548  -1.690  0.09468 .
## hourAvgFlow  0.0022405  0.0006652   3.368  0.00113 **
## rainWeek     0.2938508  0.2276028   1.291  0.20010
## rain0_12    -0.0305700  0.7107799  -0.043  0.96579
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.017 on 87 degrees of freedom
## Multiple R-squared:  0.1975, Adjusted R-squared:  0.1606
## F-statistic: 5.354 on 4 and 87 DF,  p-value: 0.0006674
```

For this Confusion Matrix, the accuracy level is 97% and there were 0% missed hits (out of 1 unsafe observation, the model caught it.)

```
testFourVarMlrLoc4 <- 3.0615415 + (-0.5564694 * testLoc4$rain48_96) + (0.0022405 * testLoc4
$hourAvgFlow) + (0.2938575 * testLoc4$rainWeek) + (-0.0305788 * testLoc4$rain0_12)

cmMlrLoc4 <- confusionMatrix(
  as.factor(ifelse(testFourVarMlrLoc4 > log(235), 1, 0)),
  as.factor(ifelse(testLoc4$yLoc4 > log(1260), 1, 0))
)
cmMlrLoc4$table
```

```
##           Reference
## Prediction  0  1
##          0 32  0
##          1  0  1
```

Section 4: Conclusion

4.1 - Final Models

After comparing the results of the logistic Generalized Linear Model, a normal Multivariate Linear Regression (not included in this document), and a log-transformed Multivariate Linear Regression, I conclude that the log-transformed Multivariate Linear Regression is the best option for this flagging model.

For each location, we predict the natural log of the E. coli population as a linear combination of various weather and river discharge factors, summarized below. If the output is greater than `log(235)` , then this model predicts that the water should be flagged.

It should also be noted that this model is only a prediction, and that it is derived from a rather small sample size. One of the main reasons why modeling whether the E. coli population is unsafe is difficult, is that the river is so often safe. For example, at CRWA's Location 4, over the 6 years from 2017 to 2022, an E. coli population above 1260 was only recorded once. The arithmetic mean at Location 4 is 94.7.

Without further ado, here are the final models:

For Location 1:

$$\log(y_1) \approx 5.1681832 + 4.6625787 * (A) + 0.0006113 * (B)$$

- A is the total rain in inches over the last 0-12 hours.
- B is the average flow discharge over the last 0-1 hour.

If $\log(y_1) > \log(235)$, the water should be flagged.

Here is the Confusion Matrix on the testing data, as well as across both the training and testing data.

```
cmMlrLoc1$table
```

```
##           Reference
## Prediction  0  1
##          0 21  0
##          1  4  8
```

```
entireTwoVarMlrLoc1 <- 5.1681832 + (4.6625787 * dataLoc1$rain0_12) + (0.0006113 * dataLoc1$h
ourAvgFlow)

confusionMatrix(
  as.factor(ifelse(entireTwoVarMlrLoc1 > log(235), 1, 0)),
  as.factor(ifelse(dataLoc1$yLoc1 > log(1260), 1, 0))
)$table
```

```
##           Reference
## Prediction  0  1
##          0 87  4
##          1 19 16
```

For Location 2:

$$\log(y_2) \approx 5.501886 + 2.997021 * (A) - 0.014088 * (B) + 0.003538 * (C)$$

- A is the total rain in inches over the last 0-12 hours.
- B is the average air temperature over the last 0-3 days.
- C is the number of days since the last “Major Rainfall” (more than 0.1 inches)

If $\log(y_2) > \log(235)$, the water should be flagged.

Here are the two Confusion Matrices.

```
cmMlrLoc2$table
```

```
##           Reference
## Prediction  0  1
##          0 26  2
##          1  0  4
```

```
entireThreeVarMlrLoc2 <- 5.501886 + (2.997021 * dataLoc2$rain0_12) + (-0.014088 * dataLoc2$t
hreeDayAvgAirTemp) + (0.003538 * dataLoc2$daysSinceLastMajorRain)

confusionMatrix(
  as.factor(ifelse(entireThreeVarMlrLoc2 > log(235), 1, 0)),
  as.factor(ifelse(dataLoc2$yLoc2 > log(1260), 1, 0))
)$table
```

```
##           Reference
## Prediction   0   1
##          0 116   3
##          1   3   5
```

For Location 3:

$\log(y_3) \approx 4.380013 + 0.011368 * (A) - 0.010225 * (B) + 3.765905 * (C)$

- A is the average flow discharge over the last 0-12 hours.
- B is the average flow discharge over the last 0-24 hours.
- C is the total rain in inches over the last 0-12 hours.

If $\log(y_3) > \log(235)$, the water should be flagged.

Here are the two Confusion Matrices.

```
cmMlrLoc3$table
```

```
##           Reference
## Prediction  0  1
##          0 26  2
##          1  2  2
```

```
entireThreeVarMlrLoc3 <- 4.380013 + (0.011368 * dataLoc3$halfDayAvgFlow) + (-0.010225 * data
Loc3$dayAvgFlow) + (3.765905 * dataLoc3$rain0_12)

confusionMatrix(
  as.factor(ifelse(entireThreeVarMlrLoc3 > log(235), 1, 0)),
  as.factor(ifelse(dataLoc3$yLoc3 > log(1260), 1, 0))
)$table
```

```
##           Reference
## Prediction   0   1
##          0 110   4
##          1   7   6
```

For Location 4:

$\log(y_4) \approx 3.0615415 - 0.5564694 * (A) + 0.0022405 * (B) + 0.2938575 * (C) - 0.0305788 * (D)$

- A is the total rain in inches over the last 48-96 hours.
- B is the average flow discharge over the last 0-1 hour.
- C is the total rain in inches over the last 0-7 days.
- D is the total rain in inches over the last 0-12 hours.

If $\log(y_4) > \log(235)$, the water should be flagged.

Here are the two Confusion Matrices.

```
cmMlrLoc4$table
```

```
##           Reference
## Prediction  0  1
##          0 32  0
##          1  0  1
```

```
entireFourVarMlrLoc4 <- 3.0615415 + (-0.5564694 * dataLoc4$rain48_96) + (0.0022405 * dataLoc
4$hourAvgFlow) + (0.2938575 * dataLoc4$rainWeek) + (-0.0305788 * dataLoc4$rain0_12)

confusionMatrix(
  as.factor(ifelse(entireFourVarMlrLoc4 > log(235), 1, 0)),
  as.factor(ifelse(dataLoc4$yLoc4 > log(1260), 1, 0))
)$table
```

```
##           Reference
## Prediction   0   1
##          0 123   0
##          1   1   1
```

4.2 - Closing Remarks

Thank you for reading this report! I hope you found this documentation informational and useful.

Thank you to the Charles River Watershed Association, especially Lisa Kumpf and Max Rome, for the opportunity to work on these models and for their support during my internship. Thank you as well to Eli Kane, a past intern whose documentation from several years ago served as a starting point for my data exploration.

Ayushman Choudhury | Brown University Class of 2025 | January 20, 2023