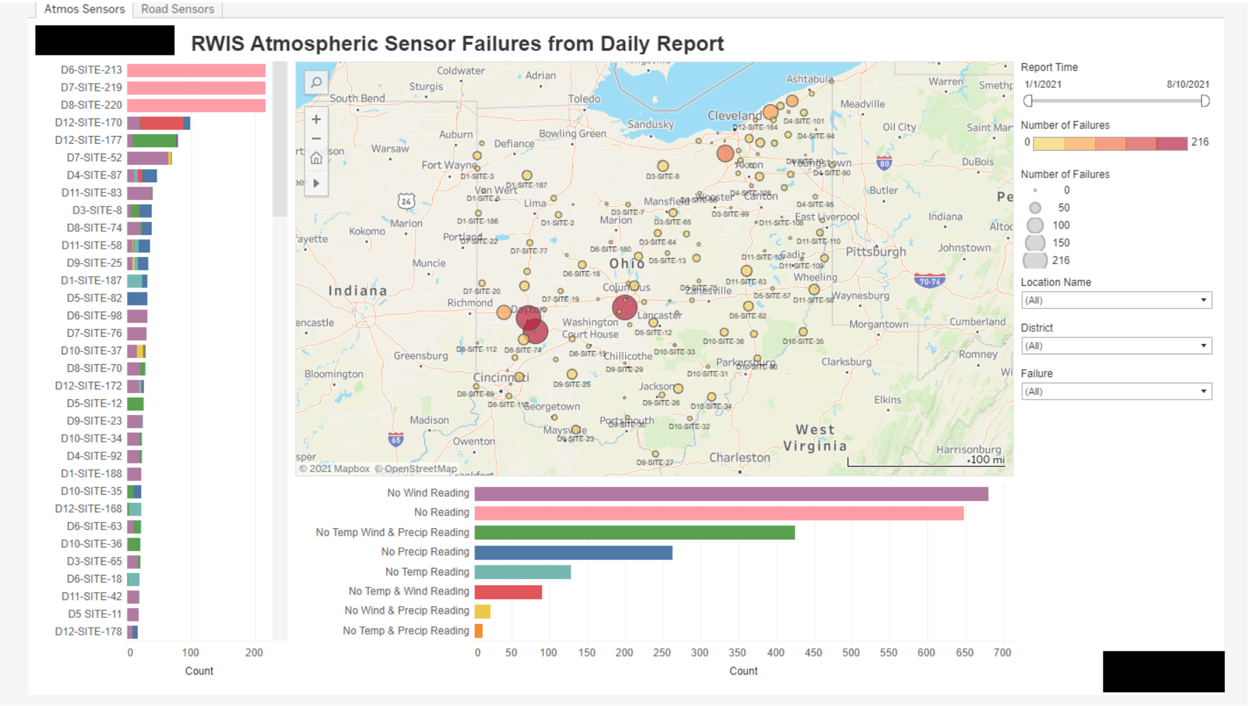
Kyle Vertin
Internship May 2021 - August 2021

Dashboards:


[redacted]

RWIS Atmospheric Sensor Failures from Daily Report

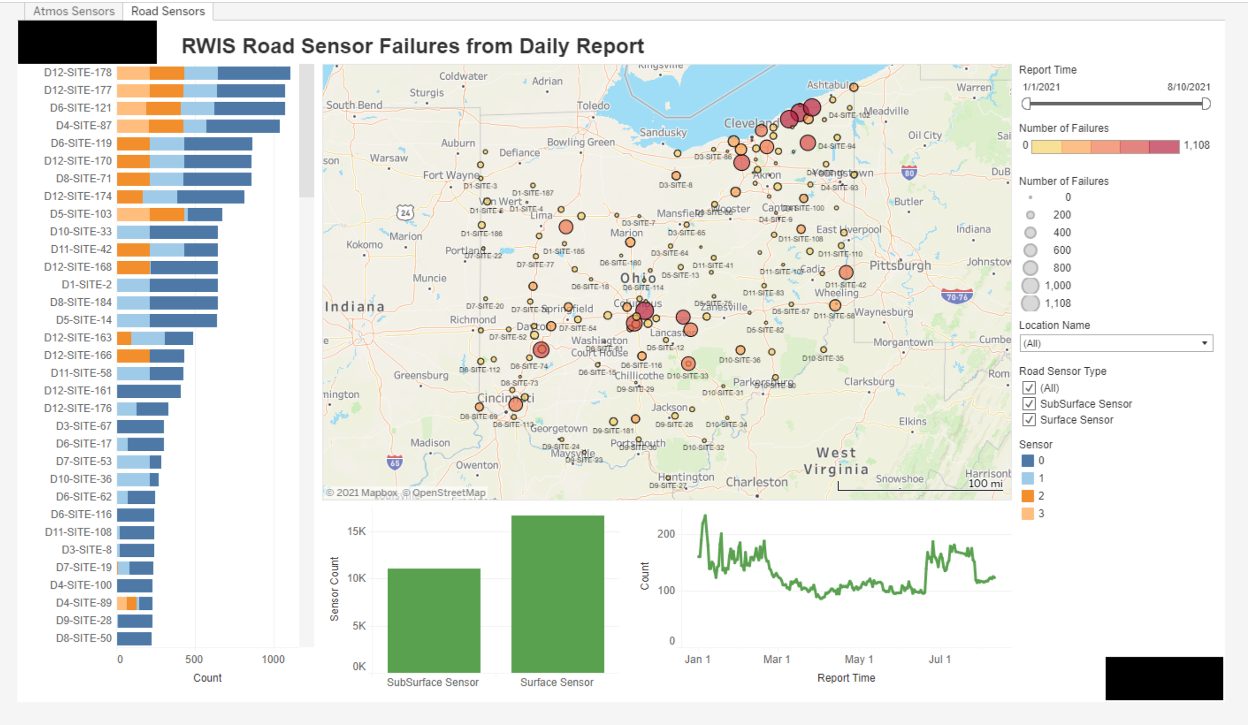
Report Time 1/1/2021 – 8/10/2021

Location Name (All)

District (All)

Failure (All)

[redacted]


[redacted]

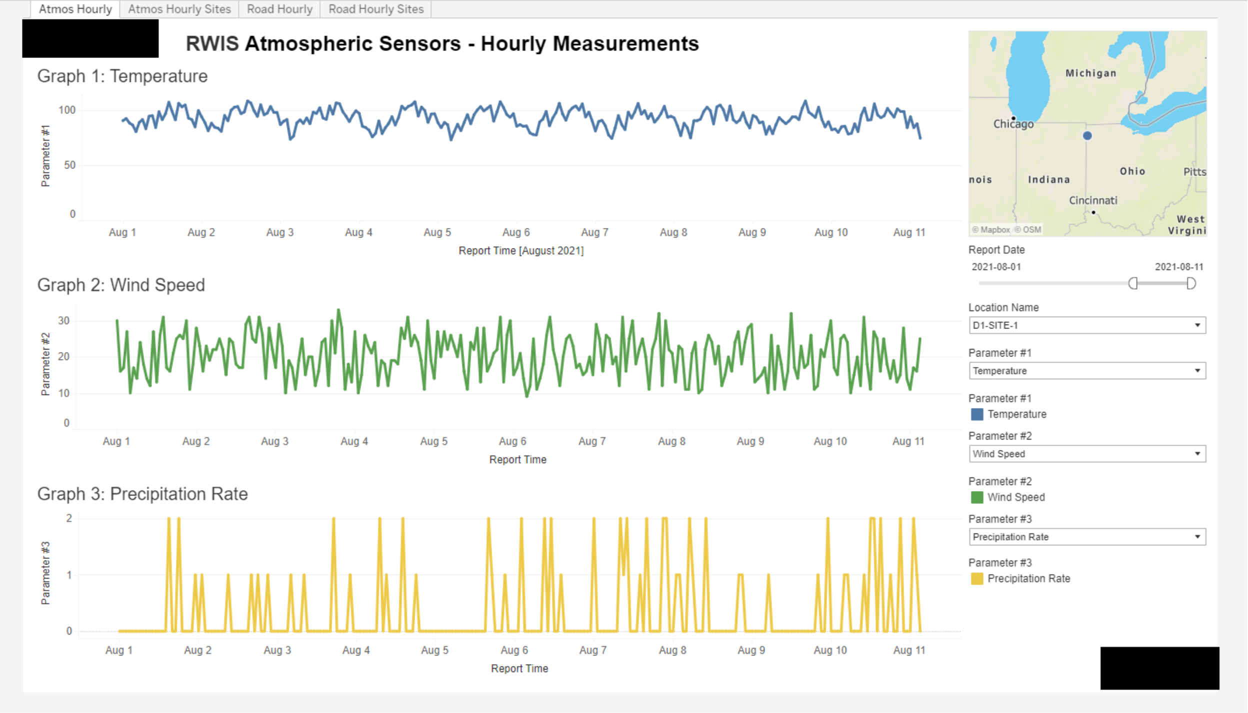
RWIS Road Sensor Failures from Daily Report

Report Time 1/1/2021 – 8/10/2021

Location Name (All)

Road Sensor Type
☑ (All)
☑ SubSurface Sensor
☑ Surface Sensor

[redacted]



[redacted]

RWIS Atmospheric Sensors - Hourly Measurements

Graph 1: Temperature

Graph 2: Wind Speed

Graph 3: Precipitation Rate

Report Date
2021-08-01 2021-08-11

Location Name: D1-SITE-1

Parameter #1: Temperature

Parameter #1
🟦 Temperature

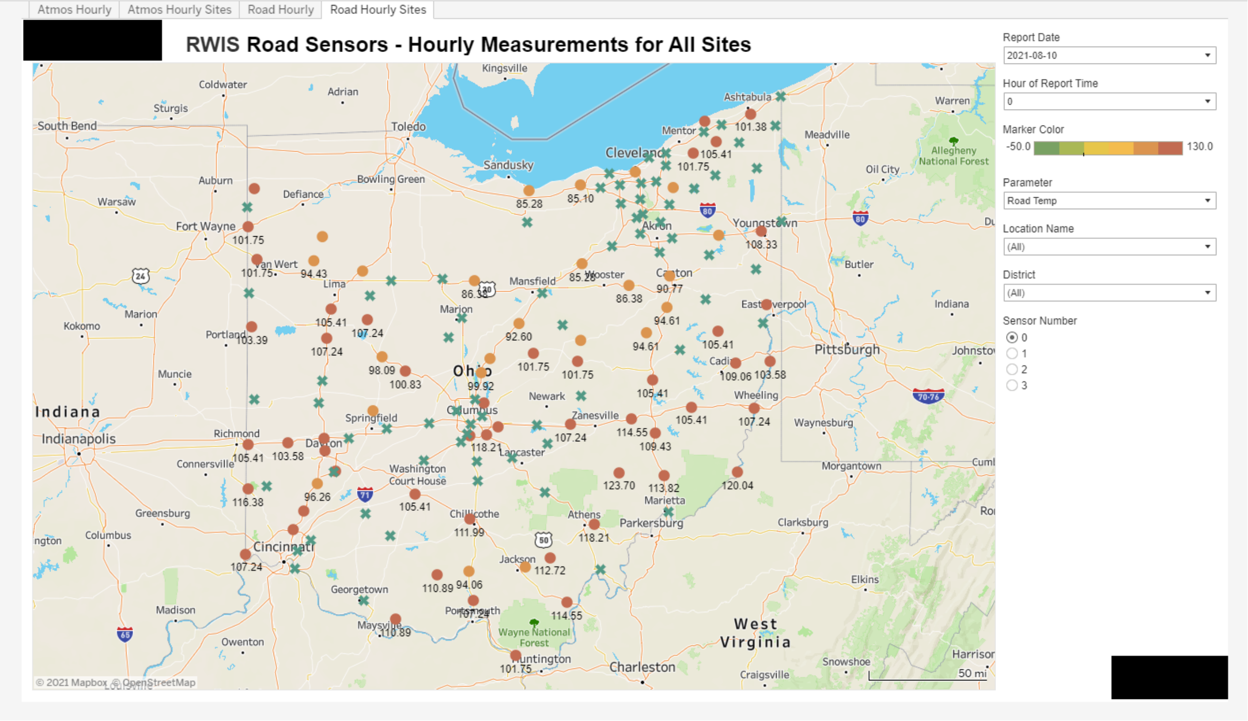
Parameter #2: Wind Speed

Parameter #2
🟩 Wind Speed

Parameter #3: Precipitation Rate

Parameter #3
🟨 Precipitation Rate

[redacted]



[redacted]

RWIS Road Sensors - Hourly Measurements for All Sites

Report Date: 2021-08-10

Hour of Report Time: 0

Marker Color
-50.0 130.0

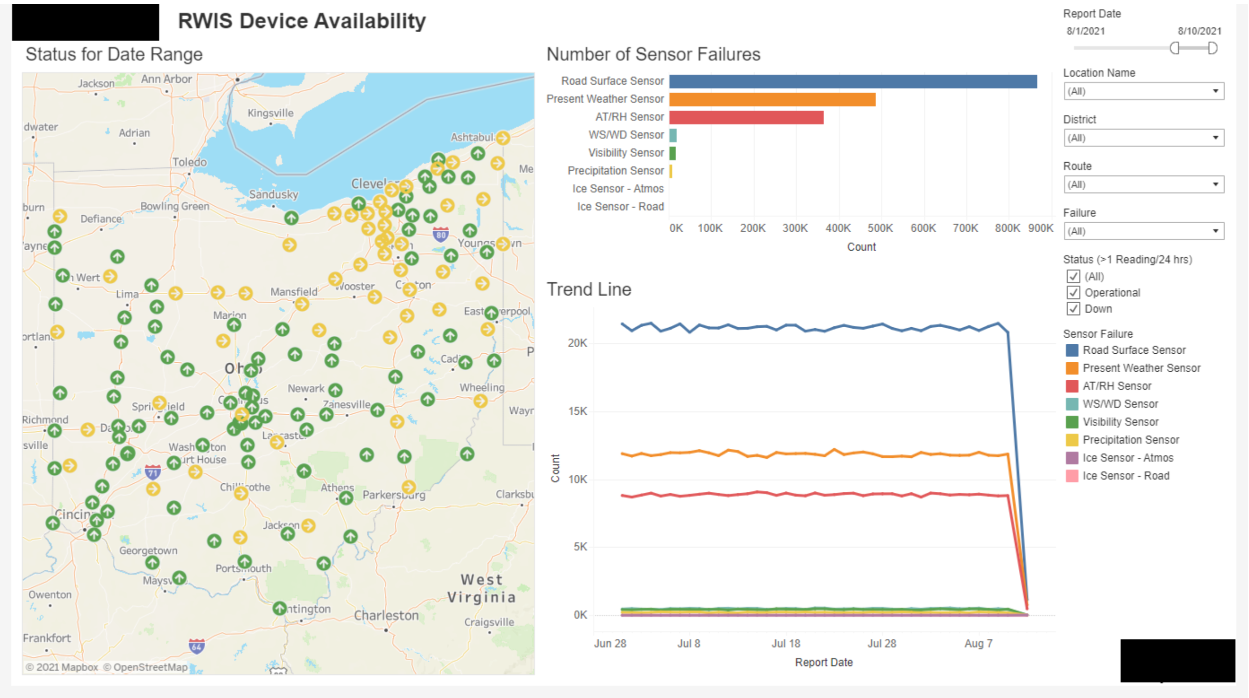
Parameter: Road Temp

Location Name: (All)

District: (All)

Sensor Number
● 0
○ 1
○ 2
○ 3

[redacted]



[redacted] RWIS Device Availability

Status for Date Range

Number of Sensor Failures

Trend Line

Report Date 8/1/2021 8/10/2021

Location Name (All)

District (All)

Route (All)

Failure (All)

Status (>1 Reading/24 hrs)
☑ (All)
☑ Operational
☑ Down

Sensor Failure
🟦 Road Surface Sensor
🟧 Present Weather Sensor
🟥 AT/RH Sensor
🟦 WS/WD Sensor
🟩 Visibility Sensor
🟨 Precipitation Sensor
🟪 Ice Sensor - Atmos
🟥 Ice Sensor - Road

[redacted]

Scripts:

----------------------PARSE THE DAILY REPORT AND STORE IN DATABASE-------------------------

```
library(lubridate)
library(stringr)
library(plyr)
library(dplyr)
library(openxlsx)
library(tools)

rm(list=ls())
Sys.setenv(TZ='America/New_York')

home <- "setwd('~/analytics/scripts')"
eval(parse(text = home))

# Read reference data for normalization purposes
setwd("../database")
r_failure <- read.csv("r_failure.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
r_failure <- subset(r_failure, select = c(failure_id, failure, failure_datasource))

t_location <- read.csv("t_location.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
```

```
#t_location$location <- t_location$location_number
t_location$location_number <- as.integer(gsub("^.*-", "", t_location$location_number))
t_location <- t_location[!is.na(t_location$location_number),]
t_location <- subset(t_location, select = c(location_id, location_description, location_number))

# Read the database tables that store daily report data

t_dailyatmos <- read.csv("t_dailyatmos.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
t_dailysensor <- read.csv("t_dailysensor.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))

# Denormalize

        t_dailyatmos  <- join(t_dailyatmos,
r_failure[r_failure$failure_datasource=="daily_report_atmospheric",], by = "failure_id",) %>%
                join(.,t_location, by="location_id")

        # warning if failure or location_number are not in reference tables
        if(sum(is.na(t_dailyatmos$failure)) > 0) { warning("daily_parse: The failure in t_dailyatmos
is NA") }
        if(sum(is.na(t_dailyatmos$location)) > 0) { warning("daily_parse: The location in
t_dailyatmos is NA") }

        t_dailyatmos <-  t_dailyatmos[,c("location_number", "failure", "location_description",
"report_time",
                                  "dailyatmos_inserted", "dailyatmos_inserted_by", "dailyatmos_updated")]



#Denormalize t_dailysensor



        t_dailysensor  <- join(t_dailysensor,
r_failure[r_failure$failure_datasource=="daily_report_sensor",], by = "failure_id") %>%
                join(.,t_location, by="location_id")



        t_dailysensor <-  t_dailysensor[,c("location_number", "failure", "location_description",
"sensor", "sensor_name", "report_time",
                                          "dailysensor_inserted", "dailysensor_inserted_by",
"dailysensor_updated")]
```

```
# Replace all your names:  location became location, location_description became location_description.....report_data to report_time
# Run each part of the code and change the columns name to match the schema

setwd("../input_files/daily_report")
input_files <- list.files(pattern = ".txt", all.files = TRUE)

# Rename the files with their dates

    for(file in input_files) {

        #file <- input_files[1]
        rdr <- readLines(file)
        rdr <- data.frame(x = rdr, stringsAsFactors = FALSE)

        report_time <- str_sub(rdr$x[1], 43, nchar(rdr$x[1]))
        # gsub use regex, meaning regular expressions
        report_time <- gsub("\\.", "", report_time)

        # Put in UTC format
        report_time <- date(strptime(report_time, "%m/%d/%Y", tz = ""))
        # print(report_time)

        newfile <- paste0("daily_report_", report_time, ".txt")

        if(!newfile %in% input_files) {
            file.rename(file, newfile)
        }
    }

# Check if the data for each report is already in the database

    input_files <- sort(list.files(pattern = ".txt", all.files = TRUE))
    outage_flag <- 0
    sensor_flag <- 0

for(file in input_files) {

    #file <- input_files[1]
    rdr <- readLines(file)
    rdr <- data.frame(x = rdr, stringsAsFactors = FALSE)

    report_time <- str_sub(rdr$x[1], 43, nchar(rdr$x[1]))
    # gsub use regex, meaning regular expressions
    report_time <- gsub("\\.", "", report_time)

    # Put in UTC format
    report_time <- date(strptime(report_time, "%m/%d/%Y", tz = ""))
    # print(report_time)

    # Initialize
    rdr$location_number <- NA
    rdr$location_description <- NA
    rdr$sensor <- NA
    rdr$sensor_name <- NA
    rdr$failure <-   NA
    # Use ?strptime codes
    rdr$report_time <- paste0(format(report_time, "%Y-%m-%d"), " 10:00:00")

    # Remove blank lines
    rdr$x <- gsub("^ *$", "", rdr$x)
    rdr <- rdr[rdr$x != "",]

    row.names(rdr) <- NULL

    if(sum(t_dailyatmos$report_time == report_time) == 0) {

        # Flag = 1 means the records for this report are being added to the database and the
end
        outage_flag <- 1
        #Top
        start <- which(grepl("RWIS ATMOSPHERIC REPORT", rdr$x))
        end <-  which(grepl("RWIS SURFACE SENSOR REPORT", rdr$x))
        t1 <- rdr[start:(end-1),]

        t1 <- as.data.frame(t1)

        t1 <- t1[grepl("^[0-9]", t1$x),]
        t1$report_name <- "RWIS ATMOSPHERIC REPORT"

        t1$dailyatmos_inserted <- t1$report_time
        t1$dailyatmos_inserted_by <- "daily_parse"
        t1$dailyatmos_updated <- t1$report_time

        d_start <- c(1,11,71)
        d_end <- c(10,70,255)

        for(i in 1:nrow(t1)) {
```

```
        t1$location_number[i] <- substr(t1$x[i],d_start[1],d_end[1])
        t1$location_description[i] <- substr(t1$x[i],d_start[2],d_end[2])
        t1$failure[i] <- substr(t1$x[i],d_start[3],d_end[3])
    }

    t1$failure <- gsub("^.*No", "No", t1$x)
    t1$location_number <- as.integer(t1$location_number)
    t1$location_description <- gsub(" *$", "", t1$location_description)
    t1$failure <- gsub(" *$", "", t1$failure)

    t1 <- subset(t1, select = -c(x, sensor, sensor_name, report_name))

    if(nrow(t1) > 0) {
        # Column names must be identical for rbind
        t_dailyatmos <- rbind(t_dailyatmos,t1)
    }
}

if(sum(t_dailysensor$report_time == report_time) == 0) {

    sensor_flag <- 1
    # Middle
    start <- which(grepl("RWIS SURFACE SENSOR REPORT", rdr$x))
    end <-  which(grepl("RWIS SUB SENSOR REPORT", rdr$x))
    t2 <- rdr[start:(end-1),]
    t2 <- t2[grepl("^[0-9]", t2$x),]
    t2$report_name <- "RWIS SURFACE SENSOR REPORT"

    t2$dailysensor_inserted <- t2$report_time
    t2$dailysensor_inserted_by <- "daily_parse"
    t2$dailysensor_updated <- t2$report_time


    d_start <- c(1,11,71,81,131)
    d_end <- c(10,70,80,130,255)

    for(i in 1:nrow(t2)) {
        t2$location_number[i] <- substr(t2$x[i],d_start[1],d_end[1])
        t2$location_description[i] <- substr(t2$x[i],d_start[2],d_end[2])
        t2$sensor[i] <- substr(t2$x[i],d_start[3],d_end[3])
        t2$sensor_name[i] <- substr(t2$x[i],d_start[4],d_end[4])
        t2$failure[i] <- substr(t2$x[i],d_start[5],d_end[5])
    }
```

```
 t2$location_number <- as.integer(t2$location_number)
 t2$sensor <- as.integer(t2$sensor)
 t2$location_description <- gsub(" *$", "", t2$location_description)
 t2$sensor_name <- gsub(" *$", "", t2$sensor_name)
 t2$failure <- gsub(" *$", "", t2$failure)

 # t2 <- subset(t2, select = -c(x,location_description,report_name))

 # Bottom

 start <- which(grepl("RWIS SUB SENSOR REPORT", rdr$x))
 end <-  nrow(rdr)
 t3 <- rdr[start:end,]
 t3 <- t3[grepl("^[0-9]", t3$x),]
 t3$report_name <- "RWIS SURFACE SENSOR REPORT"

 t3$dailysensor_inserted <- t3$report_time
 t3$dailysensor_inserted_by <- "daily_parse"
 t3$dailysensor_updated <- t3$report_time


 d_start <- c(1,11,71,81,131)
 d_end <- c(10,70,80,130,255)

 for(i in 1:nrow(t3)) {
        t3$location_number[i] <- substr(t3$x[i],d_start[1],d_end[1])
        t3$location_description[i] <- substr(t3$x[i],d_start[2],d_end[2])
        t3$sensor[i] <- substr(t3$x[i],d_start[3],d_end[3])
        t3$sensor_name[i] <- substr(t3$x[i],d_start[4],d_end[4])
        t3$failure[i] <- substr(t3$x[i],d_start[5],d_end[5])
 }

 t3$location_number <- as.integer(t3$location_number)
 t3$sensor <- as.integer(t3$sensor)
 t3$location_description <- gsub(" *$", "", t3$location_description)
 t3$sensor_name <- gsub(" *$", "", t3$sensor_name)
 t3$failure <- gsub(" *$", "", t3$failure)


# t3 <- subset(t3, select = -c(x,location_description, report_name))

 t2 <- subset(t2, select = -c(x,report_name))
 t3 <- subset(t3, select = -c(x,report_name))
```

```
        if(nrow(t2) > 0 | nrow(t3) > 0 ) {
            # Column names must be identical for rbind
            t_dailysensor <- rbind(t_dailysensor, rbind(t2,t3))
    }
    }
    } # end of for loop for the files


# Normalize: Using the location, get the location_id and failure_id

    t_dailyatmos <- join(t_dailyatmos, t_location, by = "location_number") %>%
        join(., r_failure[r_failure$failure_datasource=="daily_report_atmospheric",], by =
"failure")

    # Create a warning that the failure is not in r_failure table, to prompt you to add it
    if(sum(is.na(t_dailyatmos$failure_id)) > 0) {
        warning("daily_parse: Daily outage failure is not in r_failure table, add it")
        print("daily_parse: Daily outage failure is not in r_failure table, add it")
        print(sort(unique(t_dailyatmos$failure[is.na(t_dailyatmos$failure_id)])))
    }

    # Drop the records that do not have a match in t_location (ICE sites)
    t_dailyatmos <- t_dailyatmos[!is.na(t_dailyatmos$location_id),]


    # This key needs to be assigned for the whole table for database simulation purposes only
    t_dailyatmos$dailyatmos_id <- 1:nrow(t_dailyatmos)

    # Keep the columns that were in the original database table
    t_dailyatmos <-
t_dailyatmos[,c("dailyatmos_id","location_id","failure_id","location_description", "report_time",

"dailyatmos_inserted","dailyatmos_inserted_by","dailyatmos_updated")]


    t_dailysensor <- join(t_dailysensor, t_location, by = "location_number") %>%
        join(., r_failure[r_failure$failure_datasource=="daily_report_sensor",], by = "failure")

    # Drop the records that do not have a match in t_location (ICE sites)
    t_dailysensor <- t_dailysensor[!is.na(t_dailysensor$location_id),]

    # This key needs to be assigned for the whole table for database simulation purposes only
    t_dailysensor$dailysensor_id <- 1:nrow(t_dailysensor)
```

```
        # Keep the columns that were in the original database table
        t_dailysensor <-
t_dailysensor[,c("dailysensor_id","location_id","failure_id","location_description",
"sensor","sensor_name","report_time",

"dailysensor_inserted","dailysensor_inserted_by","dailysensor_updated")]

# Write the database tables

        setwd("../../database")

        if(outage_flag == 1) {
                write.table(t_dailyatmos, "t_dailyatmos.csv", row.names = FALSE, sep=",")
        }

        if(sensor_flag == 1) {
                write.table(t_dailysensor, "t_dailysensor.csv", row.names = FALSE, sep=",")
        }
```

------------------SCRIPT TO READ API DATA AND INSERT INTO DATABASE---------------------

```
suppressMessages(library(lubridate))
suppressMessages(library(plyr))
suppressMessages(library(dplyr))
suppressMessages(library(stringr))
suppressMessages(library(openxlsx))
suppressMessages(library(tools))
suppressMessages(library(httr))
suppressMessages(library(jsonlite))
suppressMessages(library(digest))

#suppressMessages(library(RMariaDB))
#suppressMessages(library(rsyslog))

# FIXES

# skip sites that aren't in t_location
# For sites that don't report_assign the failure "Not Reporting", this only applies to t_atmos

# Change location in the noramlization/demormalization, get rid of r_site_crossref.csv
# Some sites can be missing from the API query - how to handle?

# API Information

	#Some sample requests using this API key are below.  The top request is for sensorpheric observations for station OH001.  The bottom request is for surface observations for station OH001.

	# WeatherSentry API
# https://api.com/random/random/exampletext/

# How to HTTP in R
# https://medium.com/@traffordDataLab/querying-apis-in-r-39029b73d5f1`


#path <- "https://api.com/random/random/exampletext/"

# atmospheric
# path <- "https://api.com/random/random/exampletext/"

# Sensors
#path <- "https://api.com/random/random/exampletext/"

	# API Connection info
	#myquery$failure_code
	#myquery$request

rm(list = ls())
options(max.print=10000)
options(digits=6)
Sys.setenv(TZ='America/New_York')

# Staging

	# OPTIONS FOR TESTING
	# These options are for testing and are not going to be in the production code
	# Option 1: Make input database tables t_atmos, t_atmos_item, t_sesnor, t_sensor_item
	# blank with header only if empty_tables == 1, leave as-is if == 0
	empty_tables <- 0
	# Option 2: Use simulated report_time dates if simulation == 1, do not simulate if simulation == 0
	simulation <- 1
```

```
    tday <- Sys.time()
    attr(tday,"tzone") <- "America/Denver"
    tdayf2 <- format(tday, "%y%m%d_%H%M%S")
    attr(tday,"tzone") <- "UTC"
    tdayf <- format(tday, "%Y-%m-%d %H:%M:%S")


    # Choose one type of query, and one site.  Site is "OH001" in the URL.
    home <- "setwd('~/analytics/scripts')"
    eval(parse(text = home))

    # Read reference data for normalization purposes
    setwd("../database")
    site_cref <- read.csv("r_site_crossref.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
    setwd("../input_files")

    setwd("../database")

    t_location <- read.csv("t_location.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    t_location <- subset(t_location, select = c(location_id, ws_location_name,
ws_location_number))

    r_failure <- read.csv("r_failure.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    r_failure <- subset(r_failure, select = c(failure_id, failure, failure_datasource))

    # Read the database tables that store the WeatherSentry data
    t_atmos <- read.csv("t_atmos.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    t_atmos_item <- read.csv("t_atmos_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
    t_sensor <- read.csv("t_sensor.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    t_sensor_item <- read.csv("t_sensor_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))

    # Option for testing
    if(empty_tables == 1) {
        t_atmos <- t_atmos[0,]
        t_atmos_item <- t_atmos_item[0,]
        t_sensor <- t_sensor[0,]
        t_sensor_item <- t_sensor_item[0,]
    }

    # Options for testing - simulate the report_time
    # This applies only to data already in t_atmos and t_sensor
    # Scramble up the dates
    if(simulation == 1 & nrow(t_atmos) > 0) {
        # Date simulation is going to go in the ws_collect script
        adt <- as.POSIXct(paste0(date(tdayf), "06:00:00"))
        attr(adt,"tzone") <- "UTC"
        for(k in 1:30) {
            adt <- c(adt, adt[1] - days(k))
        }
        adt <- format(adt,"%Y-%m-%d %H:%M:%S")
        t_atmos$report_time <- sample(adt, nrow(t_atmos),replace = TRUE)
        t_sensor$report_time <- sample(adt, nrow(t_sensor),replace = TRUE)
    }

# Denormalize the historical data

    # t_atmos  <- join(t_atmos, t_location, by="location_id")

    atmos_cols <-
c("location_id","atmos_item_set_id","report_time","temperature","dew_point","relative_humidity",
"wind_speed","wind_direction","wind_gust",

"visibility","precipitation_rate","atmos_inserted","atmos_inserted_by","atmos_updated")

    t_atmos <- t_atmos[,atmos_cols]

    atmos_item_cols <- c("atmos_item_set_id","failure_id","count","atmos_item_inserted",
                  "atmos_item_inserted_by","atmos_item_updated")

    t_atmos_item <- t_atmos_item[,atmos_item_cols]




# Get Atmospheric Measurements from WS API query for  t_atmos

    # df has all data for all sites
    atmos <- NULL
    # failure has failure data for all sites
    failure <- NULL
    # StationId
    i <- 0

    for(j in 1:250) {
        i <- i + 1
```

```
print(i)

# Skip if the site is not in t_location
if(i %in% t_location$ws_location_number) {

a <- str_pad(as.character(i), 3, "left", pad = "0")
path <- paste0("example", a, "/example")
myquery <-  GET(url = path)
output <- httr::content(myquery, as = "text")
z <- fromJSON(output)

if("stationId" %in% names(z)) {  # If true this is a DTS site that is reporting

    # Process the qcFailures list, if there is even one failure over past hour include it
    if("qcFailures" %in% names(z) & sum(!is.na(z$qcFailures)) > 0) {
        temp <- bind_rows(z$qcFailures)
        temp$StationId <- paste0("OH",a)

        # Assemble a dataframe of all values in case we want to analyze
        if(is.null(failure)){
            failure <- temp
        }else {
            failure <- rbind(failure, temp)
        }
    }

    # Extract from embedded dataframe
    z$precipitation_rate <- z$precipitation$rate

    keepcols <- c("stationId",
              #"latitude",
              #"longitude",
              "utcTime",
              "temperature",
              "dewPoint",
              #"wetBulbTemp",
              "relativeHumidity",
              "windSpeed",
              "windDirection",
              "windGust",
              "visibility",
              "precipitation_rate"
              #"pressureSeaLevel"
    )
```

```
            z <- z[nrow(z), keepcols]

            if(is.null(atmos)){
                atmos <- z
                report_time <- as.POSIXct(strptime(z$utcTime[nrow(z)],
"%Y-%m-%dT%H:%M:%SZ"), tz = "UTC")
                print(paste0("Time of the WeatherSentry API Query in UTC = ",
report_time))
            }
            else {
                atmos <- bind_rows(atmos, z)
            }
        }else {
            # The DTS site is not reporting
            temp <- data.frame(attribute = "Not Reporting", arrayIndex = NA, causes = "",
                        StationId = paste0("OH",a))
            if(is.null(failure)){
                failure <- temp
            }else {
                failure <- rbind(failure, temp)
            }
        } # Go to the next site
        } # Skip the site if it is not in t_location, it is not a site in DTS PFP project
    } # end for loop

    # The key for joining to t_atmos_item, use 0 if no failures by default, and set key below for
sites with failures
    atmos$atmos_item_set_id <- 0
    atmos <- subset(atmos, select = -c(utcTime))
    report_time <- format(report_time,"%Y-%m-%d %H:%M:%S")

    atmos$report_time <- report_time
    atmos$atmos_inserted <- tdayf
    atmos$atmos_inserted_by <- "ws_collect"
    atmos$atmos_updated <- tdayf

    names(atmos) <- c(
        "ws_location_name",
        "temperature",
        "dew_point",
        "relative_humidity",
        "wind_speed",
        "wind_direction",
        "wind_gust",
        "visibility",
        "precipitation_rate",
        "atmos_item_set_id",
        "report_time",
        "atmos_inserted",
        "atmos_inserted_by",
        "atmos_updated"
    )

# Get Atmospheric Failures from WS API query for t_atmos_item

    # Here are the types of failures we are recording
    sort(unique(failure$attribute))
    # We want these in our df
    failure_all_col <- c("temperature","windSpeed","windDirection","windGusts",
"wetBulbTemp",
               "dewPoint","precipitation.accumulation.6H","precipitation.accumulation.24H",
               "precipitation.precipDetected","precipitation.rate","relativeHumidity",
"visibility",
               "precipitation.accumulation.1H","precipitation.accumulation.3H",
               "precipitation.accumulation.12H", "Not Reporting")

    # Warn if we are missing a failure
    if(sum(!failure$attribute %in% failure_all_col) > 0) {
        print("Need to add a failure attribute to the df data frame: ")
        print(unique(failure$attribute[!failure$attribute %in% failure_all_col]))
    }

    failure <- failure[failure$attribute %in% failure_all_col,c("attribute","StationId")]

    # Aggregate the failure data so we can get it into df
    failure$num <- 1
    fagg <- aggregate(num ~ attribute + StationId, data = failure, sum)
    fagg$attribute <- gsub("\\.","_",fagg$attribute)

    # Names in schema
    names(fagg) <- c("failure","ws_location_name","count")
    fagg$atmos_item_set_id <- 0
    fagg$atmos_item_inserted <- tdayf
    fagg$atmos_item_inserted_by <- "ws_collect"
    fagg$atmos_item_updated <- tdayf

    # Find the index in t_atmos
    if(nrow(t_atmos) == 0) {
        index <- 0
    }else {
        index <- max(t_atmos$atmos_item_set_id)
    }

    sites <- unique(atmos$ws_location_name)

    # Assign the key to link the tables
    for(i in sites) {
        # for(j in c(0,1,2,3)) would need another loop for sensors here
        # Check if there is a failure for the location_name
        if(i %in% fagg$ws_location_name) {
            index <- index + 1
            atmos$atmos_item_set_id[atmos$ws_location_name == i] <- index
            fagg$atmos_item_set_id[fagg$ws_location_name == i] <- index
        }
    }




    atmos <- join(atmos, t_location, by = "ws_location_name")
    atmos <- atmos[,atmos_cols]

    t_atmos <- rbind(t_atmos, atmos)

    # This key needs to be assigned for the whole table for database simulation purposes only
    t_atmos$atmos_id <- 1:nrow(t_atmos)
    t_atmos <- t_atmos[,c("atmos_id",atmos_cols)]

    write.table(t_atmos, "t_atmos.csv", row.names = FALSE, sep=",")

# Normalize failure_id for fagg

    fagg <- join(fagg, r_failure, by="failure")
    fagg <- fagg[,atmos_item_cols]

    t_atmos_item <- rbind(t_atmos_item, fagg)

    # This key needs to be assigned for the whole table for database simulation purposes only
    t_atmos_item$atmos_item_id <- 1:nrow(t_atmos_item)
    t_atmos_item <- t_atmos_item[,c("atmos_item_id",atmos_item_cols)]
    write.table(t_atmos_item, "t_atmos_item.csv", row.names = FALSE, sep=",")



# NOW DO THE SAME THING FOR THE PAVEMENT SENSOR DATA  - AND COMBINE IN
THE SAME DATAFRAME "ws".

# JOIN THE API ATMOS DATA TO API ROAD SESNOR DATA  AND THEN TO site_cref and
webdata

########PAVEMENT SENSORS



    # Denormalize

    # t_sensor  <- join(t_sensor, t_location, by="location_id")

    sensor_cols <- c("location_id","sensor_item_set_id","report_time","sensor_number",
             "sensor_name",
             "road_temp",
             "bridge_temp",
             "freeze_temp",
             "pavement_sensor_obs_error",
             "subsurface_temperature",
             "sensor_inserted","sensor_inserted_by","sensor_updated")

    t_sensor <- t_sensor[,sensor_cols]

    sensor_item_cols <- c("sensor_item_set_id","failure_id","count","sensor_item_inserted",
              "sensor_item_inserted_by","sensor_item_updated")

    t_sensor_item <- t_sensor_item[,sensor_item_cols]



# df has all data for all sites
sensor <- NULL
# failure has failure data for all sites
failure <- NULL
# StationId
i <- 0
```

```
    for(j in 1:25) {
        i <- i + 1
        print(i)

        # Skip if the site is not in t_location
        if(i %in% t_location$ws_location_number) {

        a <- str_pad(as.character(i), 3, "left", pad = "0")
        path <- paste0("example", a, "/example")

        myquery <-  GET(url = path)
        output <- httr::content(myquery, as = "text")
        z2 <- fromJSON(output)

        if("stationId" %in% names(z2)) {

            # Process the qcFailures list, if there is even one failure over past hour include it
            if("qcFailures" %in% names(z2) & sum(!is.na(z2$qcFailures)) > 0) {

                temp <- NULL

                for(k in 1:nrow(z2)) {
                    if(is.null(z2$qcFailures[[k]]$attribute)) {
                        fail <- data.frame(attribute = NA,
                                  StationId = paste0("OH",a),
                                  sensorId = z2$sensorId[k], stringsAsFactors =
FALSE)
                    }else {
                        fail <- data.frame(attribute = z2$qcFailures[[k]]$attribute,
                                  StationId = paste0("OH",a),
                                  sensorId = z2$sensorId[k], stringsAsFactors =
FALSE)
                    }
                    if(is.null(temp)) {
                        temp <- fail
                    }else {
                        temp <- rbind(temp,fail)
                    }

                }

                # Assemble a dataframe of all values in case we want to analyze
                if(is.null(failure)){
                    failure <- temp
                }else {
                    failure <- rbind(failure, temp)
                }
            }

            if("subsurfaceTemps" %in% names(z2) & sum(!is.na(z2$subsurfaceTemps)) > 0
) {

                sstemp <- NULL

                for(x in 1:nrow(z2)) {
                    if(is.null(z2$subsurfaceTemps[[x]]$temperature)) {
                        ssvec <- NA
                    }else {
                        # For sites that are erroneously returning two temps for one
sensor
                        # take the first one (example "OH098")
                        if(length(z2$subsurfaceTemps[[x]]$temperature) > 1) {
                            print(paste0("WS API has returned more than 1 temperature
per sensor: OH", a))
                        }
                        ssvec <- z2$subsurfaceTemps[[x]]$temperature[1]
                    }
                    if(is.null(sstemp)) {
                        sstemp <- ssvec
                    }else {
                        sstemp <- c(sstemp, ssvec)
                    }
                }
                z2$subsurfaceTemps <- sstemp
            } else {
                z2$subsurfaceTemps <- NA
            }

            z2 <- z2[z2$utcTime == max(z2$utcTime),
                      c("stationId",
                      #"latitude",
                      #"longitude",
                      "utcTime",
                      #"precipitation" this is an embedded dataframe may need to
process later
                      "sensorId",
                      "sensorName",
                      #"surfaceCondition",
                      #"mobileFriction",
                      "roadTemp",
                      "bridgeTemp",
                      "freezeTemp",
                      #"chemicalPercent",
                      #"chemicalFactor",
                      #"waterLevel",
                      #"icePercent",
                      #"conductivity",
                      #"salinity",
                      "subsurfaceTemps", # We may need to add this back in later
                      "pavementSensorObservationError"
            )]

            if(is.null(sensor)){
                sensor <- z2
                report_time <- as.POSIXct(strptime(z2$utcTime[nrow(z2)],
"%Y-%m-%dT%H:%M:%SZ"), tz = "UTC")
                attr(report_time,"tzone") <- "America/New_York"
                print(paste0("Time of the WeatherSentry API Query in Eastern Time = ",
report_time))
            }
            else {
                sensor <- bind_rows(sensor, z2)
            }
        } # Go to the next site
        } # Skip the site if it is not in t_location, it is not a site in DTS PFP project
    } # for loop

# The key for joining to t_sensor_item, use 0 if no failures by default, and set key below for sites
with failures

    sensor$sensor_item_set_id <- 0
    sensor <- subset(sensor, select = -c(utcTime))
    report_time <- format(report_time,"%Y-%m-%d %H:%M:%S")

    sensor$report_time <- report_time
    sensor$sensor_inserted <- tdayf
    sensor$sensor_inserted_by <- "ws_collect"
    sensor$sensor_updated <- tdayf

    names(sensor) <- c(
        "ws_location_name",
        "sensor_number",
        "sensor_name",
        #"surface_condition",
        #"mobile_friction",
        "road_temp",
        "bridge_temp",
        "freeze_temp",
        "subsurface_temperature",
        "pavement_sensor_obs_error",
        "sensor_item_set_id",
        "report_time",
        "sensor_inserted",
        "sensor_inserted_by",
        "sensor_updated"
    )


# Here are the types of failures we are recording
    sort(unique(failure$attribute))
    # We want these in our df2
    failure_all_col <- c("salinity", "subSurfaceTemps", "surfaceCondition", "surfaceTemp")

# Remove causes and columns we wont use
    if(sum(!failure$attribute %in% failure_all_col) > 0) {
        print("Need to add a failure attribute to the df2 data frame: ")
        print(unique(failure$attribute[!failure$attribute %in% failure_all_col]))
    }

    failure <- failure[failure$attribute %in% failure_all_col,c("attribute","StationId", "sensorId")]

    # Aggregate the failure data so we can get it into df
    failure$num <- 1
    fagg <- aggregate(num ~ attribute + StationId, data = failure, sum)
    fagg$attribute <- gsub("\\.","_",fagg$attribute)

    # Names in schema
    names(fagg) <- c("failure","ws_location_name","count")
    fagg$sensor_item_set_id <- 0
    fagg$sensor_item_inserted <- tdayf
    fagg$sensor_item_inserted_by <- "ws_collect"
    fagg$sensor_item_updated <- tdayf

    # Find the index in t_sensor
    if(nrow(t_sensor) == 0) {
        index <- 0
    }else {
        index <- max(t_sensor$sensor_item_set_id)
    }

    sites <- unique(sensor$ws_location_name)

    # Assign the key to link the tables
    for(i in sites) {
        # for(j in c(0,1,2,3)) would need another loop for sensors here
        # Check if there is a failure for the location_name
        if(i %in% fagg$ws_location_name) {
            index <- index + 1
            sensor$sensor_item_set_id[sensor$ws_location_name == i] <- index
            fagg$sensor_item_set_id[fagg$ws_location_name == i] <- index
        }
    }



    #using location_id and ws_location_name for normalization and denormalization
    #Get the t_sensor and t_sensor_item tables updating first, with normalization and
denormalization
    #Test by sourcing the script multiple times to confirm data are being added to the tables
each time it is run
    #Once it is fully working, weâ ™ll meet to discuss t_sensor/t_sensor_item and how it will
differ for storing sensor data for the site.

    sensor <- join(sensor, t_location, by = "ws_location_name")
    sensor <- sensor[,sensor_cols]

    t_sensor <- rbind(t_sensor, sensor)

    # This key needs to be assigned for the whole table for database simulation purposes only
    t_sensor$sensor_id <- 1:nrow(t_sensor)
    t_sensor <- t_sensor[,c("sensor_id",sensor_cols)]

    write.table(t_sensor, "t_sensor.csv", row.names = FALSE, sep=",")

    # Normalize failure_id for fagg

    fagg <- join(fagg, r_failure, by="failure")
    fagg <- fagg[,sensor_item_cols]

    t_sensor_item <- rbind(t_sensor_item, fagg)
    # This key needs to be assigned for the whole table for database simulation purposes only
    t_sensor_item$sensor_item_id <- 1:nrow(t_sensor_item)
    t_sensor_item <- t_sensor_item[,c("sensor_item_id",sensor_item_cols)]

    write.table(t_sensor_item, "t_sensor_item.csv", row.names = FALSE, sep=",")
```

-----SCRIPT TO READ API DATA AND INSERT INTO DATABASE WITH AGGREGATIONS----

```
suppressMessages(library(lubridate))
suppressMessages(library(plyr))
suppressMessages(library(dplyr))
suppressMessages(library(stringr))
suppressMessages(library(openxlsx))
suppressMessages(library(tools))
suppressMessages(library(httr))
suppressMessages(library(jsonlite))
suppressMessages(library(digest))

#suppressMessages(library(RMariaDB))
#suppressMessages(library(rsyslog))

# FIXES

# Change location in the noramlization/demormalization, get rid of r_site_crossref.csv
# Some sites can be missing from the API query - how to handle?

# API Information

#Some sample requests using this API key are below.  The top request is for sensorpheric
observations for station OH001.  The bottom request is for surface observations for station
OH001.

# WeatherSentry API
# https://api.com/random/random/exampletext/

# How to HTTP in R
# https://medium.com/@traffordDataLab/querying-apis-in-r-39029b73d5f1`


#path <- "https://api.com/random/random/exampletext/"
```

```
# atmospheric
# path <- "https://api.com/random/random/exampletext/"

# Sensors
#path <- "https://api.com/random/random/exampletext/"

# API Connection info
#myquery$failure_code
#myquery$request

rm(list = ls())
options(max.print=10000)
options(digits=6)
Sys.setenv(TZ='America/New_York')

# Staging

    home <- "setwd('~/analytics/scripts')"
    eval(parse(text = home))

    # Specify the retention days for data in t_atmos and t_sensor tables and their item tables
    retention_days <- 10

    tday <- Sys.time()
    attr(tday,"tzone") <- "America/Denver"
    tdayf2 <- format(tday, "%y%m%d_%H%M%S")
    attr(tday,"tzone") <- "UTC"
    tdayf <- format(tday, "%Y-%m-%d %H:%M:%S")

# Read reference data for normalization purposes
setwd("../database")
site_cref <- read.csv("r_site_crossref.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
setwd("../input_files")

setwd("../database")

# Read the database tables that store the WeatherSentry data

t_atmos <- read.csv("t_atmos.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
t_atmos_item <- read.csv("t_atmos_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
t_sensor <- read.csv("t_sensor.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
t_sensor_item <- read.csv("t_sensor_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
```

```
t_availability <- read.csv("t_availability.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))

r_failure <- read.csv("r_failure.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
unneeded <-
c("precipitation_accumulation_3H","precipitation_accumulation_6H","precipitation_accumulation
_12H",
           "salinity","surfaceCondition")
r_failure <- r_failure[!r_failure$failure %in% unneeded,]
r_failure_atmos <- r_failure$failure_id[r_failure$failure_datasource == "api_atmospheric"]
r_failure_sensor <- r_failure$failure_id[r_failure$failure_datasource == "api_sensor"]

t_location <- read.csv("t_location.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
t_location_vec <- t_location$location_id[t_location$active == 1]

t_asset <- read.csv("t_asset.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
t_asset <- t_asset[,c("asset_status_id","location_id")]

# Make this just availability_cols


# Consolidate this to a single table t_availability and remove unused statements



t_availability_cols <- c("location_id",
             "failure_id",
             "asset_status_id",
             "report_date",
             "count",
             "availability_inserted",
             "availability_inserted_by",
             "availability_updated")

t_availability <- t_availability[,t_availability_cols]


#
# Determine if date(t_atmos$report_time) %in% t_availability$report_date
# If it is in, there is nothing to do.

      alldates <- as.character(unique(date(t_atmos$report_time)))
      newdates <- alldates[!alldates %in% t_availability$report_date]
```

```
    #atmos

    if(length(newdates) > 0) {

        t_atmos <- t_atmos[as.character(date(t_atmos$report_time)) %in% newdates, ]

        # Convert report_time time zone to Eastern before availabilityregation (convert string
to as.POSIXct(), then use attr() to change tz to Eastern)
        #Give some new name:
        t_atmos_item <- join(t_atmos, t_atmos_item, by = "atmos_item_set_id")

        t_atmos_item$report_time <-  as.POSIXct(t_atmos_item$report_time, tz = "UTC")
        attr(t_atmos_item$report_time, "tzone") <- "America/New_York"

        t_atmos_item$report_date <- date(t_atmos_item$report_time)

        #as.Date(t_atmos_item$report_time, tz = "EST", format = "%Y-%m-%d-%h")
        #format = "%Y-%m-%d-%h-%m-%s"

        aagg <- aggregate(count ~ location_id + failure_id + report_date, t_atmos_item, sum)

        aagg$report_date <- format(aagg$report_date, "%Y-%m-%d")

        df <- expand.grid(location_id = t_location_vec, failure_id = r_failure_atmos,
report_date = newdates)

        aagg <- join(df, aagg, by = c("location_id", "failure_id", "report_date"))
        aagg$count[is.na(aagg$count)] <- 0

        # Assign the current asset_status_id START HERE - APPLY THIS TO SENSOR

        aagg <- join(aagg, t_asset, by = "location_id")

        aagg$availability_inserted <- tdayf
        aagg$availability_inserted_by <- "ws_aggregate"
        aagg$availability_updated <- tdayf
        names(aagg)[names(aagg) == "count"] <- "count"

        aagg <- aagg[,t_availability_cols]

        # now check that you match the schema exactly before doing rbind (t_availability)
        t_availability <- rbind(t_availability, aagg)
    }
```

```
#sensor

        # Determine if date(t_sensor$report_time) %in% t_availability$report_date
        # If it is in, there is nothing to do.

        # Can't do this because newdates already added to t_availability above
        #alldates <- unique(date(t_sensor$report_time))
        #newdates <- alldates[!alldates %in% t_availability$report_date]

        if(length(newdates) > 0) {




                #t_sensor <- t_sensor[as.character(date(t_sensor$report_time)) %in% newdates, ]

                # Convert report_time time zone to Eastern before aggregation (convert string to
as.POSIXct(), then use attr() to change tz to Eastern)
                #Give some new name:
                t_sensor_item <- join(t_sensor, t_sensor_item, by = "sensor_item_set_id")

                t_sensor_item$report_time <-  as.POSIXct(t_sensor_item$report_time, tz = "UTC")
                attr(t_sensor_item$report_time, "tzone") <- "America/New_York"

                t_sensor_item$report_date <- date(t_sensor_item$report_time)

                #as.Date(t_sensor_item$report_time, tz = "EST", format = "%Y-%m-%d-%h")
                #format = "%Y-%m-%d-%h-%m-%s"

                sagg <- aggregate(count ~ location_id + failure_id + report_date, t_sensor_item, sum)

                sagg$report_date <- format(sagg$report_date, "%Y-%m-%d")

                df <- expand.grid(location_id = t_location_vec, failure_id = r_failure_sensor,
report_date = newdates)

                sagg <- join(df, sagg, by=c("location_id", "failure_id", "report_date"))
                sagg$count[is.na(sagg$count)] <- 0



                sagg <- join(sagg, t_asset, by = "location_id")
        sagg$availability_inserted <- tdayf
        sagg$availability_inserted_by <- "ws_aggregate"
        sagg$availability_updated <- tdayf
        names(sagg)[names(sagg) == "count"] <- "count"

        sagg <- sagg[,t_availability_cols]

        # now check that you match the schema exactly before doing rbind (t_availability)
        t_availability <- rbind(t_availability, sagg)
}

    # Assign key and get columns in the right order before writing to database
    t_availability$availability_id <- 1:nrow(t_availability)
    t_availability_cols <- c("availability_id",t_availability_cols)
    t_availability <- t_availability[,t_availability_cols]

    t_availability$availability_id <- 1:nrow(t_availability)
    t_availability <- t_availability[,c("availability_id",t_availability_cols)]

    write.table(t_availability, "t_availability.csv", row.names = FALSE, sep=",")  # Do this after
t_sensor


# Next: Delete records in t_atmos and t_atmos_item that are older than "retention_days"

    # Hints
    # You need to determine atmos_item_set_id keys that are older than the retention_days
and
    # delete from t_atmos_item table using the keys
    # Find retention_cutoff_datetime <- tdayf - days(retention_days)

    # delete_keys  <- t_atmos$atmos_item_set_id[t_atmos$report_time <
retention_cutoff_datetime]
    # t_atmos_item <- t_atmos_item[!t_atmos_item$atmos_item_set_id %in% delete_keys]
    # Do the same thing for t_sensor and t_sensor_item

    # Read the database tables that store the WeatherSentry data
    # Need to read again becuase tables changed per above
    t_atmos <- read.csv("t_atmos.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    t_atmos_item <- read.csv("t_atmos_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
    t_sensor <- read.csv("t_sensor.csv", stringsAsFactors = FALSE, na.strings = c("NA",""))
    t_sensor_item <- read.csv("t_sensor_item.csv", stringsAsFactors = FALSE, na.strings =
c("NA",""))
```

```
        retention <- Sys.Date() - 10

        retain_keys  <- t_atmos$atmos_item_set_id[date(t_atmos$report_time) >= retention &
t_atmos$atmos_item_set_id != 0]
        t_atmos <- t_atmos[date(t_atmos$report_time) >= retention, ]
        t_atmos_item <- t_atmos_item[t_atmos_item$atmos_item_set_id %in% retain_keys,]

        retain_keys  <- t_sensor$sensor_item_set_id[date(t_sensor$report_time) >= retention &
t_sensor$sensor_item_set_id != 0]
        t_sensor <- t_sensor[date(t_sensor$report_time) >= retention, ]
        t_sensor_item <- t_sensor_item[t_sensor_item$sensor_item_set_id %in% retain_keys,]
```