This document walks through an R script designed to clean up data from the Mattole meteorological station and produce monthly and yearly averages of variables.

The first part of the script describes what the script is supposed to do, who wrote it, and when it was edited last.

#Emily Kyker-Snowman

#February 26th, 2014

#This script reads in met station data, flags bad data, converts data to a time series and

#calculates monthly and yearly averages.

Next, the script sets the working directory and checks to see if the ‘plyr’ and ‘xts’ packages are installed. plyr allows multiple text files to be pulled into one dataframe, and xts provides some useful tools for efficiently working with time series data.

#Setting up packages

setwd("~/Desktop/R")

#library(lattice)

require(plyr)

library(plyr)

require(xts)

library(xts)

This part pulls in data from all the .txt files in a user-specified folder and pieces them into a single dataframe. It doesn’t matter if the columns are in different orders in different files; if the headers are the same, plyr will put the data in the right columns. If one file has a column that another lacks, the column will be added to the dataframe, but all the records from the files without that column will be given ‘NA’ for the value of that variable. Since the variable names in the Mattole files include the serial numbers of the sensors, this means that if a sensor is replaced by one with a different serial number, the two sensors will produce two separate columns in the dataframe.

#Pulling in all the .txt files of met stations records from a folder (duplicate records ok)

listOfFiles <- list.files(path = "~/Desktop/R/Mattole met station txt files", pattern= ".txt", full.names = TRUE)

metRecordsRaw <- ldply(listOfFiles, read.table, header=TRUE, sep="\t", skip=2, fill=TRUE) #stringsAsFactors=FALSE)

str(metRecordsRaw)

#attach(metRecordsRaw)

#names(metRecordsRaw)

#summary(metRecordsRaw)

This part checks if there are duplicate records based on the ‘Date’ field and removes any duplicates. This way, if the same file is downloaded multiple times, averages won’t be weighted by repeat records.

#Deduplicating records by date

metRecords = subset(metRecordsRaw, !duplicated(Date\_Time))

This part shortens the variable names at “\_” to remove the serial numbers and units. This makes them much easier to call later in the script. gsub uses regular expressions, which are hard to explain succinctly, but you can read more about them online if you’re interested in the weird-looking first argument given to gsub here.

#Shortening header names to everything that comes before '\_' (look up 'regular expressions' to understand the first argument in gsub)

names(metRecords) <- gsub("\_.\*","", names(metRecords))

This part looks for records in the dataset that might have something wrong with them (i.e. unusually high gust speeds that mean that the wind sensor is malfunctioning).

#Scanning for weird values in dataset

highGustSpeed = subset(metRecords, metRecords$Gust.Speed > 100)

highGustSpeed

lowTemp = subset(metRecords, metRecords$Temperature < 0)

lowTemp

This part turns the data into a time series that the xts package can use to perform efficient averaging over different time scales. The arguments in xts() tell the function which columns to convert to a time series, what format to use (POSIXct in this case), which column contains time, and which parts of that time column correspond to month/day/year etc.

#Converting data to a time series

x <- xts(metRecords[,2:8], as.POSIXct(metRecords$Date, format="%m/%d/%Y %H:%M:%S"))

rain <- xts(metRecords[,6], as.POSIXct(metRecords$Date, format="%m/%d/%Y %H:%M:%S"))

This part quickly calculates monthly and annual means. The apply.monthly function can also take other functions, like standard deviation. xts also includes apply.daily, apply.weekly, and apply.quarterly. (Changes in sampling rate, i.e. once every 5 minutes to once every 30 minutes, can bias data towards more frequent sampling. One way around this is to average daily and then average monthly, but for some reason the apply.daily function breaks up met station data at 00:20, 15:55, and 23:55 instead of just once per day. A function could be written using plyr to do what apply.daily does without the bugs if you created a new field for ‘Day’ and used ddply to average records by day (see LICOR script for details on ddply)).

#Calculating monthly means

monthlyMeans = apply.monthly(x, mean)

monthlyMeans

monthlyTotalRain = apply.monthly(rain, sum)

monthlyTotalRain

#Calculating annual means

annualMeans = apply.yearly(x, mean)

annualMeans

yearlyTotalRain = apply.yearly(rain, sum)

yearlyTotalRain