PNNL-16136

Compensating For Changes in MOS Sensors

Brett Matzke
Pacific Northwest National Laboratory

September 29, 2006

1 Introduction

"AirAdvice uses metal oxide semiconductor (MOS) sensors for measuring total volatile organic compounds (TVOC) in air. These sensors are incorporated into AirAdvice's indoor air quality (IAQ) monitors. The IAQ monitors are designed so that they require annual calibration to maintain acceptable accuracy. Since the MOS TVOC sensors used in the monitors change in sensitivity with time and exposure to gases, AirAdvice has developed an algorithm-based process that automatically compensates for changes in the sensors. The proposed project is to have PNNL analyze data provided by AirAdvice with these objectives: (1) assess how effective AirAdvice's automatic drift correction process is, (2) identify any problems in the process, and (3) propose improvements to the process."

- From **TVOC Sensor Accuracy Drift Correction -- PNNL Project for AirAdvice**
  Jonathan Lay, AirAdvice, Inc.
  August 18, 2006

2 Summary

With AirAdvice's objectives in mind, the following changes to the drift correction process are recommended for future research, and are explained in more detail in throughout this report. Numbers 2-5 all pertain to changes to AirAdvice's current methodology, and it is recommended that AirAdvice validate these methods to assure they will achieve their desired goals.

1) Use the free statistics package R instead of Excel for fitting models and generating plots
2) Stop using the simple linear regression model, and instead use a non-parametric slope parameter with no intercept
3) Use a warm-up time of at least 30 minutes, and no more than 2 hours
4) Devise a method for detecting erroneous "spikes" in the data caused by glitches in the sensors. Replace these with "NA" (missing values)
5) Do not use a model if the raw data is from a very narrow range or if one of the sensors appears to be "dead"

3 Data Provided for Analysis

The final data used for this analysis were received from Jonathan Lay of AirAdvice on September 12, 2006 in the *AiradviceToPNNL.mdb* file. This file contained data on more than twenty IAQ monitors. Each IAQ monitor contains two MOS sensors. One is the main sensor, which runs whenever the IAQ monitor is turned on. The other is a control sensor which turns on periodically for 6 to 8 hours to gather data for checking against the main sensor, then turns off. Each time the control sensor turns on, this is considered a trial. There were 336 usable trials in the data. Each sensor takes a reading every minute or every two minutes. When both sensors are on, they take readings at the same times. AirAdvice "cooks" the raw sensor data into a value measured in parts-per-billion (ppb) ranging from zero to several thousand.

4 Outcomes and Recommendations

4.1 Use R instead of Excel

One of the first problems identified was that AirAdvice was using Excel to generate regression models and plots for the more than 300 trials. Using Excel for this was a very manual process. Additionally, the plots were on individual tabs, which make it very difficult to view many plots at one time. AirAdvice needed a more automated method of running regression models and producing plots.

The free statistics package R is recommended since it can automate this process for AirAdvice and they can easily implement its use. R can be downloaded from www.r-project.org. R code is also easy to interpret (has many similarities to C++), and it would not be difficult to later convert to another programming language. Attached in Appendix A is sample code that reads in AirAdvice's raw data, performs data preprocessing, runs regression models, and outputs all of the 672 plots to a single PDF file.

4.2 Use a Non-parametric Slope Estimate with No Intercept

AirAdvice was using a simple linear regression model which contains an intercept and slope parameter. It is recommended that an intercept not be used. The reason is that these models are fit on one set of data, and are later applied to other sets of data. Often there will be extrapolation in these cases, and with an intercept in the model, it is possible to get negative ppb values, which is undesirable.

It also is recommended that AirAdvice use a non-parametric method of estimating the slope. Sensors vary in their ability to "bounce back" from a sudden escalation in sensor values. One sensor may quickly shoot up by 1000 ppb while another may go up gradually and catch up to the other sensor up to one or two hours later. Otherwise, when sensors are operating normally, the ratio of their ppb values typically follow a linear pattern, meaning one sensor's values are approximately a proportion of the other's. The proposed formula for adjusting the main sensors readings is:

$$\text{Adjusted Main Sensor} = \beta_1 * \text{Main Sensor}$$

where β_1 is the slope parameter. The non-parametric method of computing β_1 is as follows. Say the readings for the main sensor are $a_1, a_2, a_3, \ldots, a_n$, and the corresponding readings for the control sensor are $b_1, b_2, b_3, \ldots, b_n$. We expect most of the ratios b_i/a_i to be around the same value (the data usually follows a linear pattern). Let β_1 be the median of the set of ratios $b_1/a_1, b_2/a_2, b_3/a_3, \ldots, b_n/a_n$. This allows for an accurate estimate of β_1 when sensors are under typical conditions. Escalations in ppb values and erroneous data influence the slope parameter very little using this method.

One additional point about using a non-parametric parameter is that it may eliminate the needs of using a warm-up time (Section 3.3) and detecting erroneous spikes (Section 3.4). It is recommended that AirAdvice research how robust the parameter is when data from the warm-up time and erroneous spikes is included.

4.3 Use a Warm-Up Time of 30 minutes to 2 hours

A study was done to determine the optimal warm-up time for the control sensors. This is a period where the control sensor adapts to the environment after being turned on. In Table 1, the columns are percentiles of R-square values from the 336 models. The columns that best demonstrate changes in R-squared are the 25th and 50th percentiles. R-squared values appear to go up as warm up time increases to 30 minutes. Then R-squared values remain stable up through 2 hours and then some models appear to start falling apart, possibly because of removing so much data. It is recommended that 30 minutes of warm up time be used.

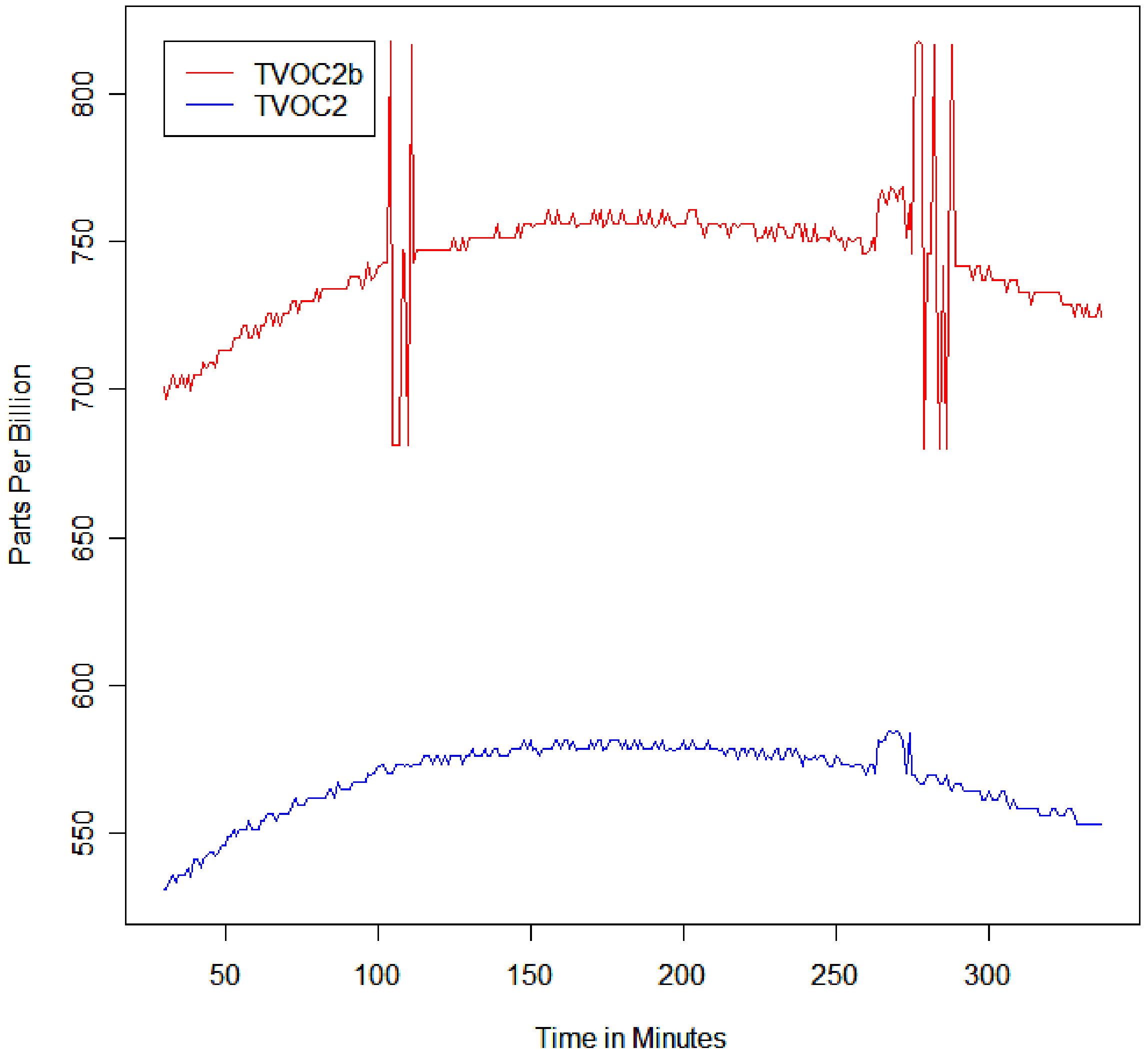
Table 1. R-squared Values of Models with Different Warm-Up Times (in minutes)

time	0%	10%	25%	50%	75%	90%	100%
0	0.03	0.26	0.57	0.87	0.97	0.99	1.00
10	0.00	0.13	0.57	0.87	0.96	0.99	1.00
20	0.00	0.23	0.68	0.90	0.98	0.99	1.00
30	0.00	0.21	0.74	0.95	0.99	1.00	1.00
40	0.00	0.18	0.68	0.93	0.99	1.00	1.00
50	0.00	0.24	0.73	0.93	0.99	1.00	1.00
60	0.00	0.14	0.70	0.94	0.99	1.00	1.00
70	0.00	0.27	0.72	0.92	0.99	1.00	1.00
80	0.00	0.23	0.68	0.92	0.99	1.00	1.00
90	0.00	0.31	0.73	0.93	0.99	1.00	1.00
100	0.00	0.22	0.70	0.92	0.99	1.00	1.00
110	0.00	0.25	0.71	0.92	0.99	1.00	1.00
120	0.00	0.32	0.74	0.93	0.99	1.00	1.00
150	0.00	0.20	0.61	0.89	0.98	1.00	1.00
180	0.00	0.22	0.61	0.90	0.99	1.00	1.00
240	0.00	0.07	0.40	0.82	0.98	1.00	1.00

4.4 Detect Erroneous Spikes

One of the most noticeable observations when initially plotting the data was that on some trials one sensor would “spike” while the other sensor would not (See Figure 2). This was shared with AirAdvice who investigated and determined these were erroneous values. AirAdvice is looking into preventing such spikes, but their occurrences in this data warranted a method for detecting them.

Figure 1. TVOC2b sensor values (red) contains erroneous spikes while TVOC2 (blue) does not.



An initial method was developed that loops through each point in time for each sensor and analyzes the surrounding points (for example, the 5 readings to the left, and 5 readings to the right), recording the mean and standard deviation. If the point in question was more than 5 standard deviations away from the mean of the other points, it is most likely an erroneous point, and is replaced with a missing value. However, the initial method was not robust against cases where multiple spikes occur close together. Since there was not sufficient time to expand on the original method, AirAdvice plans to develop its own method for detecting erroneous spikes using the same philosophy as the initial method. AirAdvice may also elect not to tackle this problem should they implement the model which uses a non-parametric estimate of the slope parameter.

4.5 Avoid Narrow Model Ranges and Dead Sensors

An observation from the data was that most trials have a fairly wide range of ppb values where ppb can fluctuate by several hundred or more ppb over several hours. However, sometimes ppb does not fluctuate much, and the range can be 100 ppb or less. This causes problems in that the model is fit on a tight cluster of values, but is later applied to a wide range. It is recommended that models only be applied if the raw data had a sufficient range of values.

Another observation is that there were some “dead” sensors that did not appear to be responding any longer. It is recommended that AirAdvice develop methods for detecting these sensors and not apply regression models to them.

5 Conclusions

Sensors within the air monitoring appear to have an approximately linear relationship when both are working properly. A linear model is therefore a good choice. A non-parametric method of computing the slope parameter has been suggested since this computes an accurate parameter while not allowing erroneous spikes and differences in response time between sensors to heavily influence the model.

Appendix A – Sample R Code

```
## Prior to running R code, export 3 access database files into .csv format
## PNNL-CookedData1                export as   PNNL-CookedData1.csv
## PNNL-RCVLVOCAdjust2             export as   PNNL-RCVLVOCAdjust2.csv
## PNNL-TVOC2UncorrelatedLog       export as   PNNL-TVOC2UncorrelatedLog.csv
## Be sure to export the field names
## Place these into a folder

## Next read the 3 data files into R
## Make sure the extensions in the filename commands are correct
## It make take a few minutes for the data to get read in

## Read in the Cooked data values
## Make sure the extension on the "filename" line is correct
filename    <-          "C:\\Air Advice\\PNNL-CookedData1.csv"
cooked                  <-          read.table(filename, header=T, sep=",", row.names=NULL)
rm(filename)  ## clear filename

## Read in the Unadjusted data values
## Make sure the extension on the "filename" line is correct
filename    <-          "C:\\Air Advice\\PNNL-TVOC2UncorrelatedLog.csv"
uncorrelated            <-          read.table(filename, header=T, sep=",", row.names=NULL)
rm(filename)  ## clear filename

## Read in the Adjustments data
## Make sure the extension on the "filename" line is correct
filename    <-          "C:\\Air Advice\\PNNL-RCVLVOCAdjust2.csv"
adjust                  <-          read.table(filename, header=T, sep=",", row.names=NULL)
rm(filename)  ## clear filename

## Add row number variables to cooked data
cooked$cookedrownum               <-          rep(1,length(cooked$MonID))
cooked$cookedrownum               <-          cumsum(cooked$cookedrownum)

## Find MonIDs that occur in both the cooked and adjust files

cooked.MonID.names                <-          names(table(cooked$MonID))
adjust.MonID.names                <-          names(table(adjust$MonID))
cooked.adjust.MonID.names         <-          c(cooked.MonID.names,adjust.MonID.names)
cooked.adjust.MonID.counts        <-          table(cooked.adjust.MonID.names)
rm(cooked.adjust.MonID.names)
cooked.adjust.MonID.names         <-          names(cooked.adjust.MonID.counts)
cooked.adjust.MonID.names         <-          cooked.adjust.MonID.names[cooked.adjust.MonID.counts > 1.5]   ## values > 2 indicate
the MonID is in both data tables
rm(cooked.adjust.MonID.counts)

cooked.adjust.MonID.names         ## print out the names of these MonIDs

## In cooked data - remove records that do not have a TVOC2b value

ind.na                  <-          is.na(cooked[, "TVOC2b"])
cooked                  <-          cooked[!ind.na, ]

## In uncorrelated data - remove MonIDs not in the remaining cooked data

ind.MonID<-             is.element(uncorrelated$MonID,cooked.adjust.MonID.names)
uncorrelated            <-          uncorrelated[ind.MonID, ]

## Rename TVOC2 in the cooked data to TVOC2corrected

colnames(cooked)[colnames(cooked)=="TVOC2"]        <-          "TVOC2corrected"

## Merge cooked and uncorrelated by common MonID and Stamp

cooked$mergevalue <- paste(cooked$MonID,cooked$Stamp,sep="")
uncorrelated$mergevalue <- paste(uncorrelated$MonID,uncorrelated$Stamp,sep="")
ind.dupvars             <-          is.element(names(uncorrelated),c("MonID","Stamp"))
```

```
uncorrelated <- uncorrelated[,!ind.dupvars]
cookeduncorr <- merge(cooked,uncorrelated,by = "mergevalue")
ind.dupvars                <-              is.element(names(cookeduncorr),c("mergevalue"))
cookeduncorr <- cookeduncorr[,!ind.dupvars]

## The data gets sorted differently - Re-sort so cookedrownum is ascending
sort.data.frame <- function(x, key, ...) {
   if (missing(key)) {
      rn <- rownames(x)
      if (all(rn %in% 1:nrow(x))) rn <- as.numeric(rn)
      x[order(rn, ...), , drop=FALSE]
   } else {
      x[do.call("order", c(x[key], ...)), , drop=FALSE]
   }
}

cookeduncorr <- sort.data.frame(cookeduncorr, key = "cookedrownum")


## Assign PlotID for each group of data that will be plotted
## Note that this method assumes that there is extra data in between trials that is deleted - so it looks for changes in row number
greater than 1.
## However - we have seen a couple of cases where this did not occur.
## Recommend that Air Advice change this code so that it reads in the date and time, and instead looks to see if the time interaval is
more than 1 or 2 minutes.
## First see if records were deleted
cookedrownumprevrow <- c(-1000,cookeduncorr$cookedrownum[1:length(cookeduncorr$cookedrownum)-1])
## Next see if the MonIDs change
cookedprevMonID <- c(-1000,cookeduncorr$MonID[1:length(cookeduncorr$MonID)-1])
## If either happens, mark the record
rownummissed <- 1*((cookeduncorr$cookedrownum-cookedrownumprevrow) > 1.5 | (cookeduncorr$MonID!=cookedprevMonID))
cookeduncorr$PlotID <- cumsum(rownummissed)

## Create GroupID-MonID variable (puts the Group letter in front of the MonID)
cookeduncorr$GroupID <- cookeduncorr$MonID
cookeduncorr$GroupID[(cookeduncorr$MonID==11148)] <- 'F11148'
cookeduncorr$GroupID[(cookeduncorr$MonID==11260)] <- 'A11260'
cookeduncorr$GroupID[(cookeduncorr$MonID==11497)] <- 'H11497'
cookeduncorr$GroupID[(cookeduncorr$MonID==11535)] <- 'G11535'
cookeduncorr$GroupID[(cookeduncorr$MonID==11552)] <- 'B11552'
cookeduncorr$GroupID[(cookeduncorr$MonID==11559)] <- 'C11559'
cookeduncorr$GroupID[(cookeduncorr$MonID==11564)] <- 'C11564'
cookeduncorr$GroupID[(cookeduncorr$MonID==11604)] <- 'F11604'
cookeduncorr$GroupID[(cookeduncorr$MonID==11608)] <- 'B11608'
cookeduncorr$GroupID[(cookeduncorr$MonID==11643)] <- 'A11643'
cookeduncorr$GroupID[(cookeduncorr$MonID==11653)] <- 'D11653'
cookeduncorr$GroupID[(cookeduncorr$MonID==11670)] <- 'C11670'
cookeduncorr$GroupID[(cookeduncorr$MonID==11672)] <- 'D11672'
cookeduncorr$GroupID[(cookeduncorr$MonID==11699)] <- 'B11699'
cookeduncorr$GroupID[(cookeduncorr$MonID==11708)] <- 'A11708'
cookeduncorr$GroupID[(cookeduncorr$MonID==11713)] <- 'D11713'
cookeduncorr$GroupID[(cookeduncorr$MonID==11752)] <- 'F11752'
cookeduncorr$GroupID[(cookeduncorr$MonID==11769)] <- 'H11769'
cookeduncorr$GroupID[(cookeduncorr$MonID==11790)] <- 'H11790'
cookeduncorr$GroupID[(cookeduncorr$MonID==11801)] <- 'G11801'
cookeduncorr$GroupID[(cookeduncorr$MonID==11802)] <- 'G11802'
cookeduncorr$GroupID[(cookeduncorr$MonID==12151)] <- 'E12151'
cookeduncorr$GroupID[(cookeduncorr$MonID==13212)] <- 'K13212'
cookeduncorr$GroupID[(cookeduncorr$MonID==13273)] <- 'K13273'
cookeduncorr$GroupID[(cookeduncorr$MonID==13295)] <- 'K13295'
cookeduncorr$GroupID[(cookeduncorr$MonID==13925)] <- 'E13925'
cookeduncorr$GroupID[(cookeduncorr$MonID==13963)] <- 'E13963'

## Create alphabetical list of GroupIDs
GroupIDlist <- names(table(cookeduncorr$GroupID))

## Count the number of PlotIDs
numPlotID <- max(cookeduncorr$PlotID)

## Run through all of the data and create plots
```

```
pdf(file="AAplots.pdf")  ## start sending plots to a pdf file
plot(c(1,1),c(1,1),type="n",xlab="x-axis",ylab="y-axis")  ## create a junk plot for the first plot - allows for easier viewing of 2 pages
at a time in Adobe
title("The following slides contain plots")

## Create variables to store regression information
MonIDstore <- c(rep(NA,numPlotID))          ## Monitor IDs
GroupIDstore <- c(rep("",numPlotID))
PlotIDStore <- MonIDstore
Timestore <- GroupIDstore
## Regression Model with Intercept
Intercept.R.squared.val <- MonIDstore
Intercept.val <- MonIDstore
Slope.Intercept.Model <- MonIDstore
Intercept.Model.Fstat.Pval <- MonIDstore
Intercept.val.pval <- MonIDstore
Slope.pval.Intercept.Model <- MonIDstore
## Regression Model Without Intercept
No.Intercept.R.squared.val <- MonIDstore
Slope.No.Intercept.Model <- MonIDstore
No.Intercept.Model.Fstat.Pval <- MonIDstore
Slope.pval.No.Intercept.Model <- MonIDstore
## Use Median Ratio for Slope Parameter
Slope.Median <- MonIDstore


## This indicates the number of the trial that is currently being worked on
rec.num.store <- 1

## Loop through each GroupID (Group ID letter and MonID)
for (j in 1:length(GroupIDlist)) {
  PlotIDs.in.Group <- names(table(cookeduncorr$PlotID[cookeduncorr$GroupID==GroupIDlist[j]]))
  Num.PlotIDs.in.Group <- length(PlotIDs.in.Group)
## For the current GroupID, loop through each set of data for plotting
  for (i in 1:Num.PlotIDs.in.Group) {
    grab.ind <-             is.element(cookeduncorr$PlotID,PlotIDs.in.Group[i])
    tempdat                 <-             cookeduncorr[grab.ind, ]
    ## Determine minute interval
    ## Default is 1 minute, add a second minute if it is sensor 12151, 13925, or 13963
    minute.interval <- 1 + 1*(tempdat$MonID[1]==12151)
    minute.interval <- 1 + 1*(tempdat$MonID[1]==13925)
    minute.interval <- 1 + 1*(tempdat$MonID[1]==13963)
    ## Count the number of records
    num.records <- dim(tempdat)[1]
    ## If sensors have 2 minute intervals, delete first 15 records
    ## If sensors have 1 minute intervals, delete first 30 records
    if (minute.interval==2) {
      tempdat                         <-             tempdat[!c(rep(TRUE,min(15,num.records)),rep(FALSE,length(tempdat$MonID)-
min(15,num.records))), ]
    }
      if (minute.interval==1) {
        tempdat                       <-             tempdat[!c(rep(TRUE,min(30,num.records)),rep(FALSE,length(tempdat$MonID)-
min(30,num.records))), ]
    }

    ## Count the number of records
    num.records <- dim(tempdat)[1]

    ## Proceed if there are records
    if (num.records > 0) {
    minutes <- seq(from=0, to=(minute.interval*(length(tempdat$MonID)-1)), by=minute.interval )
    minutes <- minutes + 30

    record.num <- c(1:num.records)

    ## Identify erroneous "spikes"
    std.dev.to.use <- 5                ## number of standard deviations to indicate as erroneous
    min.std.dev <- 7                   ## minimum standard deviation to use
    ## Note - when std.dev.to.use is 5 and min.std.dev is 2, the spike must be at least 10 higher or lower than the surrounding points to
have a chance of being erroneous
```

```
    erroneousb <- rep(0,length(tempdat$TVOC2b))
    erroneous <- erroneousb
    if (length(tempdat$TVOC2b) > 5) {
      for (k in 3:(length(tempdat$TVOC2b)-2)) {
        sampvecb <- c(tempdat$TVOC2b[k-2],tempdat$TVOC2b[k-1],tempdat$TVOC2b[k+1],tempdat$TVOC2b[k+2])
        sampvec  <- c(tempdat$TVOC2[k-2],tempdat$TVOC2[k-1],tempdat$TVOC2[k+1],tempdat$TVOC2[k+2])
        std.dev.offb <- abs(tempdat$TVOC2b[k]-mean(sampvecb))/max(sd(sampvecb),min.std.dev)
        std.dev.off <- abs(tempdat$TVOC2[k]-mean(sampvec))/max(sd(sampvec),min.std.dev)
        if (std.dev.offb > std.dev.to.use) {
          erroneousb[k] <- 1
        }
        if (std.dev.off > std.dev.to.use) {
          erroneous[k] <- 1
        }
      } ## end k for loop
    }  ## end if
    ## correlation plots first
    ## The if statements below color code the points by time so that we can more easily view the correlation plots.
    ## Without color, it is more difficult to see patterns since you don't know what points were to what point in time.
    print(paste(j,i,sep=" "))
    plot(tempdat$TVOC2b[erroneousb==0],tempdat$TVOC2[erroneousb==0],xlab="TVOC2b ppb",ylab="TVOC2 ppb")
    if (minute.interval == 1) {
      if (num.records > 30) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 30],tempdat$TVOC2[erroneousb==0 & record.num > 30],col="red")
      }
      if (num.records > 90) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 90],tempdat$TVOC2[erroneousb==0 & record.num > 90],col="blue")
      }
      if (num.records > 150) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 150],tempdat$TVOC2[erroneousb==0 & record.num >
150],col="green")
      }
      if (num.records > 210) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 210],tempdat$TVOC2[erroneousb==0 & record.num >
210],col="orange")
      }
    }
    if (minute.interval == 2) {
      if (num.records > 15) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 15],tempdat$TVOC2[erroneousb==0 & record.num > 15],col="red")
      }
      if (num.records > 45) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 45],tempdat$TVOC2[erroneousb==0 & record.num > 45],col="blue")
      }
      if (num.records > 75) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 75],tempdat$TVOC2[erroneousb==0 & record.num >
75],col="green")
      }
      if (num.records > 105) {
        points(tempdat$TVOC2b[erroneousb==0 & record.num > 105],tempdat$TVOC2[erroneousb==0 & record.num >
105],col="orange")
      }
    }
    legend(min(tempdat$TVOC2[erroneousb==0]),max(tempdat$TVOC2[erroneousb==0]),c("1st Hour","2nd Hour","3rd Hour","4th
Hour","4+ Hours"),pch=c(1,1,1,1,1),col=c("black","red","blue","green","orange"))
    title(paste("Monitor ",tempdat$GroupID[1]," VOC2b vs Uncompensated VOC2\nFrom ",tempdat$Stamp[1]," to
",tempdat$Stamp[length(tempdat$Stamp)],sep=""))
    ## Commenting out lines command since we are now color coding the points
    ##lines(tempdat$TVOC2b[erroneousb==0 & erroneous==0],tempdat$TVOC2[erroneousb==0 & erroneous==0])


    ## time plots second
    plot(c(minutes,minutes,minutes,),c(tempdat$TVOC2,tempdat$TVOC2b,tempdat$TVOC2corrected),type="n",xlab="Time in
Minutes",ylab="Parts Per Billion")
    ##title line below is for debugging only
    ## title(paste(j,i))
    ## Plot the TVOC2b line
    lines(minutes[erroneousb==0],tempdat$TVOC2b[erroneousb==0],col="red")
    points(minutes[erroneousb==1],tempdat$TVOC2b[erroneousb==1],col="red")
    ## Plot the TVOC2 line
    lines(minutes[erroneous==0],tempdat$TVOC2[erroneous==0],col="blue")
    points(minutes[erroneous==1],tempdat$TVOC2[erroneous==1],col="blue")
    ## Commenting out TVOCcorrected since were looking at regression lines created by this code
    ## lines(minutes[erroneous==0],tempdat$TVOC2corrected[erroneous==0],col="green")
    ##points(minutes[erroneous==1],tempdat$TVOC2corrected[erroneous==1],col="green")
    ##
legend(0,max(c(tempdat$TVOC2b,tempdat$TVOC2,tempdat$TVOC2corrected)),c("TVOC2b","TVOC2","TVOC2corrected"),lty=c(
1,1,1),col=c("red","blue","green"))

    ## Do Regressions
      ## First regression with intercept
      regintercept30 <- lm(tempdat$TVOC2b[erroneous==0 & erroneousb==0] ~ tempdat$TVOC2[erroneous==0 & erroneousb==0])
      MonIDstore[rec.num.store] <- tempdat$MonID[1]
            GroupIDstore[rec.num.store] <- tempdat$GroupID[1]
            PlotIDStore[rec.num.store] <- tempdat$PlotID[1]
            Timestore[rec.num.store] <- paste(tempdat$Stamp[1]," to ",tempdat$Stamp[length(tempdat$Stamp)],sep="")
            Intercept.R.squared.val[rec.num.store] <- summary(regintercept30)$r.squared
            Intercept.val[rec.num.store] <- regintercept30$coefficients[1]
            Slope.Intercept.Model[rec.num.store] <- regintercept30$coefficients[2]
            Intercept.Model.Fstat.Pval[rec.num.store] <- (((anova(regintercept30))[5])*1)[1,1]
            fitreg <- regintercept30$coef[1] + regintercept30$coef[2]*tempdat$TVOC2[erroneous==0 & erroneousb==0]
            ## Plot the Simple Linear Regression predicted values for the adjusted TVOC2
            lines(minutes[erroneous==0 & erroneousb==0],fitreg,col="green")
            rm(fitreg)
            ## get p-values of terms in model
            regintercept30 <- summary(regintercept30)$coefficients
            regintercept30 <- regintercept30[,4]
            Intercept.val.pval[rec.num.store] <- regintercept30[1]
            Slope.pval.Intercept.Model[rec.num.store] <- regintercept30[2]
            rm(regintercept30)

      ## Second regression without intercept
      ## Putting a -1 as first X variable yields no intercept
      regintercept30 <- lm(tempdat$TVOC2b[erroneous==0 & erroneousb==0] ~ -1 + tempdat$TVOC2[erroneous==0 &
erroneousb==0])
            No.Intercept.R.squared.val[rec.num.store] <- summary(regintercept30)$r.squared
            Slope.No.Intercept.Model[rec.num.store] <- regintercept30$coefficients[1]
            No.Intercept.Model.Fstat.Pval[rec.num.store] <- (((anova(regintercept30))[5])*1)[1,1]
            fitreg <- regintercept30$coef[1]*tempdat$TVOC2[erroneous==0 & erroneousb==0]
            ## Plot the Linear Regression With No Intercept predicted values for the adjusted TVOC2
            lines(minutes[erroneous==0 & erroneousb==0],fitreg,col="orange")
            rm(fitreg)
            ## get p-values of terms in model
            regintercept30 <- summary(regintercept30)$coefficients
            regintercept30 <- regintercept30[,4]
            Slope.pval.No.Intercept.Model[rec.num.store] <- regintercept30[1]
            rm(regintercept30)

            ## Median Ratio Method
            ratio2bto2 <- tempdat$TVOC2b[erroneous==0 & erroneousb==0]/tempdat$TVOC2[erroneous==0 & erroneousb==0]
            medianratio <- median(ratio2bto2)
            Slope.Median[rec.num.store] <- medianratio
            fitreg <- medianratio*tempdat$TVOC2[erroneous==0 & erroneousb==0]
            ## Plot the predicted values for the adjusted TVOC2 using the median ratio of TVOC2b/TVOC2 as the slope with no
intercept
            lines(minutes[erroneous==0 & erroneousb==0],fitreg,col="purple")

            legend(30,max(c(tempdat$TVOC2b,tempdat$TVOC2)),c("TVOC2b","TVOC2","Regression Line","No
Intercept","Median Ratio"),lty=c(1,1,1,1,1),col=c("red","blue","green","orange","purple"))

      rec.num.store <- rec.num.store+1

    } ## end if Proceed

  }  ## end i for loop

}  ## end j for loop


dev.off()  ## stop sending plots to the pdf file
```

```
## look at regression results

## Put all of the stored regression results into a data frame
stored.data <-
data.frame(MonIDstore,GroupIDstore,PlotIDStore,Timestore,Intercept.R.squared.val,Intercept.val,Slope.Intercept.Model,Intercept.M
odel.Fstat.Pval,Intercept.val.pval,Slope.pval.Intercept.Model,No.Intercept.R.squared.val,Slope.No.Intercept.Model,No.Intercept.Mode
l.Fstat.Pval,Slope.pval.No.Intercept.Model,Slope.Median)

## In stored data - if any - remove records with missing values
ind.na              <-          is.na(stored.data[, "MonIDstore"])
stored.data <-      stored.data[!ind.na, ]

## Write the data frame out to a comma delimited file.  Default directory is in Program Files - R - and then the R version folder
write.table(stored.data, file="Trials.csv",sep="~",quote=F, col.names=NA)
```