Deployment Data Cleanup

D Goldsmith, R Wilkins

December 18, 2012

1 Introduction

Below is a description of the data summary process, how the information is processed an yields calculated.

It takes the form of a walk though of the yield calculation process, and includes code snippets from the R script used to generate the data.

The document was generated in R and Latex, using the Knitr package

2 Summary Table

A new table added to the database, the summary table is intended to hold summary statistics on deployments. This means that future work can avoid having to process entire data sets when dealing with yield, or other summarised functions.

The summary table takes the same form as the reading table, with an additional *summary type* column, these summary types are taken from a lookup table in the database.

Database Rows, and expected inputs are given below

Row	Type	Description
Time	PK,Required	Timestamp of summary, In general I would expect this to use midnight to summarise a complete day. However, if more detailed summaries (such as hourly) are needed, this should not be a problem.
nodeId	PK,Required	Id of node that this summary is from
sensorTypeId	PK	Id of sensor that this summary is from, this can be left NULL to indicate whole node summary samples (for example yield)
summaryTypeId	FK	Id of summary type.
locationId	FK	Id of location this node is from, to keep parity with the reading table
value	float	Value of the summary
textValue	string(30)	Optional text description of the summary, for example “Hot” if we are dealing with exposure graphs.

Table 1: Summary Table Description

3 Scripts

This section has a description of the scripts used process the data, and combine all samples into one database.

These scripts are designed to work with the new format (location aware) database format.

They can found in the *dataclense* directory of the *cogent-house/djgoldsmith-devel* repository.

processCC.py Transfers current cost data from the old style sqlite database, into the new format database.

processAr.py Transfers data from an Archrock postgresql database into the new format database.

getStats.R R script that calculates yields for each deployment in a given database

calcKwh.R R script to caluclate KwH usage from current cost readings.

Further details of these scripts are given below

4 getStats.R

This script calculates summary statistics for all houses in a given database. The statistics are output in two formats.

- *.csv* file with summary output for this database
- update rows in the *summary* table given these statistics

To run the script modify the source file with the relevant database access name. Then run the script through R.

4.1 Script initialisation

- Load the relevant R librarys
- Connects to the database
- Loads the Relevant Lookup tables into memory.
  - Houses Table
  - Sensor Table (For Calibration)
  - Sensor Type Table
  - Summary Type Table

```
# Load Relevant Libraries
library(RMySQL)
library(ggplot2)
library(plyr)

# Setup Database Connection
drv <- dbDriver("MySQL")
# con <- dbConnect(drv,dbname='mainStore',user='chuser')
con <- dbConnect(drv, dbname = "mainStore", user = "root", password = "Ex31S4ga")

# Load the Relevant lookup tables into memeory
allHouses <- dbGetQuery(con, statement = "SELECT * FROM House WHERE address != 'ERROR-DATA'")
summaryData <- dbReadTable(con, "SummaryType")
calibrationData <- dbReadTable(con, "Sensor")
sensorType <- dbReadTable(con, "SensorType")

## Sensors we are interested in (For Yield Calculateions)
sensorTypeList <- subset(sensorType, name == "Temperature" | name ==
    "Humidity" | name == "Light PAR" | name == "Light TSR" | name ==
```

```
    "CO2" | name == "Air Quality" | name == "VOC" | name == "Battery Voltage" |
    name == "Power")

# Create a temporary table to hold summary informtion
houseData <- data.frame(address = allHouses$address, dbStart = NA, dbEnd = NA,
    dataStart = NA, dataEnd = NA, totalNodes = NA, coNodes = NA, yield = NA,
    yieldSD = NA, yieldMin = NA, yieldMax = NA, totalSamples = NA, yieldDays = NA)

# Choose a particular house (for the demo)
i <- 7
THEHOUSE <- allHouses[i, ]
hseName <- THEHOUSE$address
# Output the House for Debugging
print(hseName)

## [1] "2 Berryfields"
```

At the end of this we have :

1. A connection to the Database
2. A collection of lookup tables used later in the application
3. One main dataframe, to hold the summary information generated during the summarising process.
4. Selected a house, in this case 2 Berryfields

4.2 Loading the Inital Dataset

This section of code fetches the relevant data for each property.

First the relevant house is fetched, and the start and end dates processed to convert to a format R is comfortable with.

Next the Relevant locations are fetched using the following query This query also fetches the corresponding room names for each location for use in a lookup table if required.

```
# As the function parameter is a row number, Work out which House
# we are dealing with. print(paste('Processing House ',hseName))
rowNo <- which(houseData$address == hseName)
cleanName <- gsub(" ", "", hseName)

# Get House from the database
houseQry <- paste("SELECT * FROM House WHERE address = '", hseName, "'",
    sep = "")
theHouse <- dbGetQuery(con, statement = houseQry)

# Process start and end dates into a format that R is happy with
theHouse$sd <- tryCatch({
    as.POSIXlt(theHouse$startDate, tz = "GMT")
}, error = function(e) {
    NA
})

theHouse$ed <- tryCatch({
    as.POSIXlt(theHouse$endDate, tz = "GMT")
}, error = function(e) {
    NA
})
```

```
# Build the Location Query
locQry <- paste("SELECT * FROM Location as Loc ", " LEFT OUTER JOIN Room as Room ",
    " ON Loc.roomId = Room.id ", " WHERE houseId =", theHouse$id, " AND Room.name NOT LIKE 'PhyNet%'
    " AND Room.name NOT IN ('Hot Water','Cold Water','Flow','Return','Hot','Cold','HotWater','ColdWa
    sep = "")

# Fetch Locations from the DB
locations <- dbGetQuery(con, statement = locQry)

# Convert into a string so we can put the data into the next query
locationIds <- paste(locations$id, collapse = ",")
```

Next the relevant dataset is fetched. Rather than fetch the entire dataset, and process it on the local machine, the query used makes use of SQL summarising functions to group the data by day. The query fetches the following values:

nodeId Used to Identify the Node

locationId Used to Identify the Location

date Timestamp converted to a date (i.e. with the time removed)

minTime Time of first sample on this date

maxTime Time of last sample on this date

minVal Minimum reported Value on this date

maxVal Maximum reported value on this date

meanVal Average value reported on this date

```
houseData[rowNo, ]$dbStart <- theHouse$startDate
houseData[rowNo, ]$dbEnd <- theHouse$endDate

# Setup the Query
sTypes <- paste(sensorTypeList$id, collapse = ",")
dataQry <- paste("SELECT nodeId, type, locationId, DATE(time) as date,count(*) as count,",
    " min(time) as minTime,max(time) as maxTime,", " min(value) as minVal, max(value) as maxVal, avg
    " FROM Reading WHERE locationId IN (", locationIds, ") ", " AND type IN (",
    sTypes, ") ", " GROUP BY nodeId,type,DATE(time)", sep = "")

# Fetch the data
houseSummary <- dbGetQuery(con, statement = dataQry)
# Add a time object so that makes sense to R
houseSummary$dt <- as.POSIXlt(houseSummary$date, tz = "GMT")
houseSummary$DT <- as.Date(houseSummary$dt)
# And add some extra columns to hold summary information on bad
# samples
houseSummary$countWithBad <- houseSummary$count
```

We should end up with a load of uncalibrated data as shown in figure **??**

```
# Plot the Uncalibrated Data
plt <- ggplot(subset(houseSummary, type == 0))
plt <- plt + geom_point(aes(dt, meanVal))
plt <- plt + geom_errorbar(aes(dt, ymin = minVal, ymax = maxVal))
```

```
plt <- plt + opts(title = paste("Uncalibrated Temperature data for ",
    hseName))
plt <- plt + xlab("Date") + ylab("Reading")
plt + facet_grid(nodeId ~ .)
```

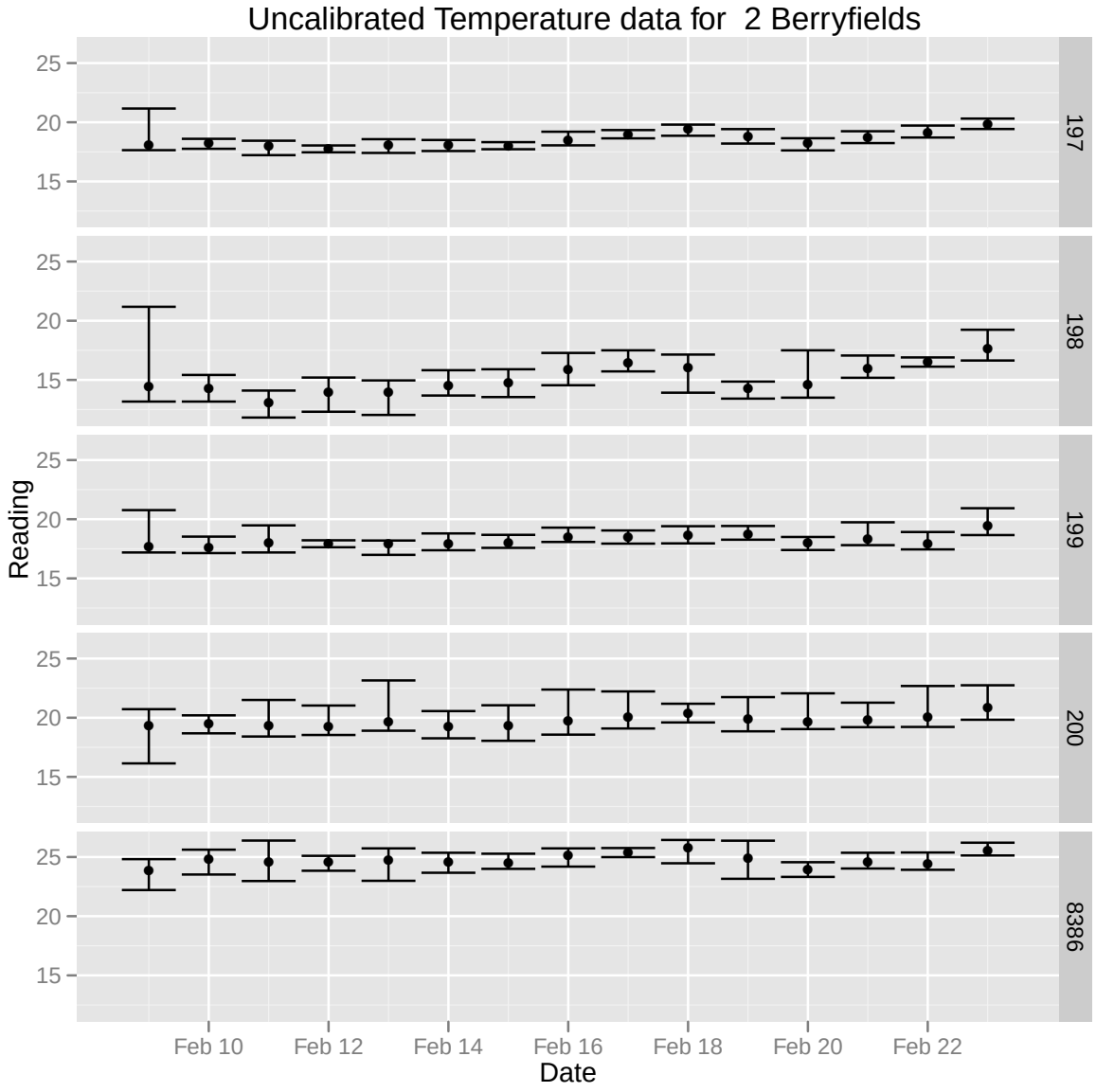


Figure 1: Uncalibrated Data

4.3 Calibration

For the purpouses of error detection, it should be sufficent to examine the minimum and maximum values.

The dataset is calibrated by merging with the calibration table, to append the calibration slope and offset (where known), where there is no calibration data, a default value of 1 for slope, and 0 for offset are used. From this a calibrated version of the Min,Max and Mean values is calculated.

```
# Merge with Calibration Data
tmp <- merge(houseSummary, calibrationData, by.x = c("nodeId", "type"),
    by.y = c("nodeId", "sensorTypeId"), all.x = TRUE)

# Add a dafualt calibration for nodes that have none
noCalibIdx <- which(is.na(tmp$id) == TRUE)
tmp[noCalibIdx, ]$calibrationSlope <- 1
tmp[noCalibIdx, ]$calibrationOffset <- 0

# Calibrate
tmp$calibMin <- (tmp$minVal * tmp$calibrationSlope) + tmp$calibrationOffset
tmp$calibMax <- (tmp$maxVal * tmp$calibrationSlope) + tmp$calibrationOffset
tmp$calibMean <- (tmp$meanVal * tmp$calibrationSlope) + tmp$calibrationOffset
# Remove Data outside of given bands
tmp$badValue <- NA
```

Figure 2 shows calibrated temperature data, before any error handling takes place.

4.4 Error Detection

Errors in the data are detected using a basic threshold based approach: Dates where the minimum or maximum values fall out of the ranges in Table 2 are marked as invalid.

NOTE: Given that with the Telos the data for temperature and humidity sensors tends to go bad when the battery voltage falls below 2.3V, it may be appropriate to filter any temperature values when the battery drops below this level.

Parameter	Minimum Threshold	Maximum Threshold
Temperature	-10	50
Humidity	0	100
Co2	0	6000

Table 2: Accepted Ranges

```
# Set a flag for data outside of certain bands
tmp$badValue <- NA

# Temperatre
badRows <- which(tmp$type == 0 & (tmp$calibMin < -10 | tmp$calibMax >
    50))
if (length(badRows) > 0) {
    tmp[badRows, ]$badValue <- TRUE
}
# Humidity
badRows <- which(tmp$type == 2 & (tmp$calibMin < 0 | tmp$calibMax > 100))
if (length(badRows) > 0) {
    tmp[badRows, ]$badValue <- TRUE
}
# Co2
badRows <- which(tmp$type == 8 & (tmp$calibMin < 0 | tmp$calibMax > 6000))
if (length(badRows) > 0) {
    tmp[badRows, ]$badValue <- TRUE
}
# TODO Rather than threshold by anything else, Threshold
# Electricity by something sensible
badRows <- which(tmp$type == 6 & (tmp$calibMin < 0 | tmp$calibMax > 5))
```

```
if (length(badRows) > 0) {
    tmp[badRows, ]$badValue <- TRUE
}
```

Where data is marked as invalid, the full data stream for that day and node is collected.

This full data stream is then calibrated and the bad readings removed using the method above.

Each day is then summarised, and the original entry in the *houseSummary* data frame replaced with the new summary values. Additionally, a new column *badCount* is added to the data table to represent the total number of samples before any data is removed. This may allow some insight into the number of bad samples per deployment.

```
# Work out the correct SQL statement
sqlValues <- subset(tmp, badValue == TRUE)

if (nrow(sqlValues) > 0) {
    uniqueNode <- paste(unique(sqlValues$nodeId), collapse = ",")
    uniqueDate <- paste(shQuote(unique(sqlValues$dt)), collapse = ",")
    theQry <- paste("SELECT * FROM Reading WHERE", " NodeId IN (", uniqueNode,
        ")", " AND Date(time) IN (", uniqueDate, ")", " AND type IN (",
        paste(sensorTypeList$id, collapse = ","), ")", sep = "")

    # Fetch all that data
    fixData <- dbGetQuery(con, statement = theQry)

    # Merge with Calibration Stuff
    fixCalib <- merge(fixData, calibrationData, by.x = c("nodeId", "type"),
        by.y = c("nodeId", "sensorTypeId"), all.x = TRUE)
    noCalibIdx <- which(is.na(fixCalib$id) == TRUE)
    rowcount <- length(noCalibIdx)
    if (rowcount > 0) {
        fixCalib[noCalibIdx, ]$calibrationSlope <- 1
        fixCalib[noCalibIdx, ]$calibrationOffset <- 0
    }

    # Calibrate
    fixCalib$calibValue <- (fixCalib$value * fixCalib$calibrationSlope) +
        fixCalib$calibrationOffset
    fixCalib$ts <- as.POSIXlt(fixCalib$time, tz = "GMT")
    fixCalib$dt <- as.Date(fixCalib$ts)

    # Remove all the bad data
    fixCalib$badValue <- FALSE
    badRows <- which(fixCalib$type == 0 & (fixCalib$calibValue < -10 |
        fixCalib$calibValue > 50))
    if (length(badRows) > 0) {
        fixCalib[badRows, ]$badValue <- TRUE
        fixCalib[badRows, ]$value <- NA
    }
    badRows <- which(fixCalib$type == 2 & (fixCalib$calibValue < 0 |
        fixCalib$calibValue > 100))
    if (length(badRows) > 0) {
        fixCalib[badRows, ]$badValue <- TRUE
        fixCalib[badRows, ]$value <- NA
    }
    badRows <- which(fixCalib$type == 8 & (fixCalib$calibValue < 0 |
```

```
        fixCalib$calibValue > 6000))
    if (length(badRows) > 0) {
        fixCalib[badRows, ]$badValue <- TRUE
        fixCalib[badRows, ]$value <- NA
    }
    # We could do with removing any temperture / humidity data where
    # the battery level is below XXX
    badRows <- which(fixCalib$type == 6 & (fixCalib$calibValue < 0 |
        fixCalib$calibValue > 5))
    if (length(badRows) > 0) {
        fixCalib[badRows, ]$badValue <- TRUE
        fixCalib[badRows, ]$value <- NA
    }

    # Summarise so its in the same format as the overall data
    fixSummary <- ddply(fixCalib, .(nodeId, locationId, type, dt), summarise,
        minVal = min(value, na.rm = TRUE), maxVal = max(value, na.rm = TRUE),
        meanVal = mean(value, na.rm = TRUE), minTime = min(ts), maxTime = max(ts),
        count = length(value), naCount = sum(is.na(value)), tCount = length(value) -
            sum(is.na(value)))

    # Remove the Infs put in by sumary functions where there is no
    # data.
    infDates <- which(is.infinite(fixSummary$minVal) == TRUE)
    if (length(infDates) > 0) {
        fixSummary[which(is.infinite(fixSummary$minVal) == TRUE), ]$minVal <- NA
        fixSummary[which(is.infinite(fixSummary$maxVal) == TRUE), ]$maxVal <- NA
    }

    # And Replace the original Values
    for (i in 1:nrow(fixSummary)) {
        thisRow <- fixSummary[i, ]
        rowIdx <- which(houseSummary$nodeId == thisRow$nodeId & houseSummary$locationId ==
            thisRow$locationId & houseSummary$type == thisRow$type &
            houseSummary$DT == thisRow$dt)
        houseSummary[rowIdx, ]$count <- thisRow$tCount
        houseSummary[rowIdx, ]$countWithBad <- thisRow$count
        houseSummary[rowIdx, ]$minVal <- thisRow$minVal
        houseSummary[rowIdx, ]$maxVal <- thisRow$maxVal
    }
}  #End of IF Statement
```

Figure 3, demonstrates calibration and removal of erroneous data, samples in *blue* are removed from the database

4.5 Calculating Per Node Yields

First a Yield per Sensor is Calculated, This is based on the *ideal* yield of 288 samples per day (5 minute sampling interval)

$$\text{Yield} = \text{count} * 288/100.0 \tag{1}$$

Next summary yields per node / location are calculated, these summaries include minimum, maximum and average yields per sensor. Minimum and Maximum values are included as in some cases, the yield for individual sensors varies on each node. For Example, consider a case where the battery is running low (below 2.3V), in this case the readings from the attached Temperature and Humidity sensors will

have errors, while other sensors (such as light levels) will continue to be reported correctly. Therefore each sensor may have differing yields.

4.6 Calculating Deployment Yield

4.7 Insert Summary Information into Main Database

Summary information is inserted into the main database.

```
plt <- ggplot(subset(tmp, type == 0))
plt <- plt + geom_point(aes(dt, calibMean, color = "calibrated"))
plt <- plt + geom_errorbar(aes(dt, ymin = calibMin, ymax = calibMax,
    color = "calibrated"))
plt <- plt + geom_point(aes(dt, meanVal, color = "uncalibrated"))
plt <- plt + opts(title = paste(hseName, "Summary Temperature Data (calibrated)"))
plt <- plt + xlab("Date") + ylab("Value")
plt <- plt + facet_grid(nodeId ~ .)
plt
```

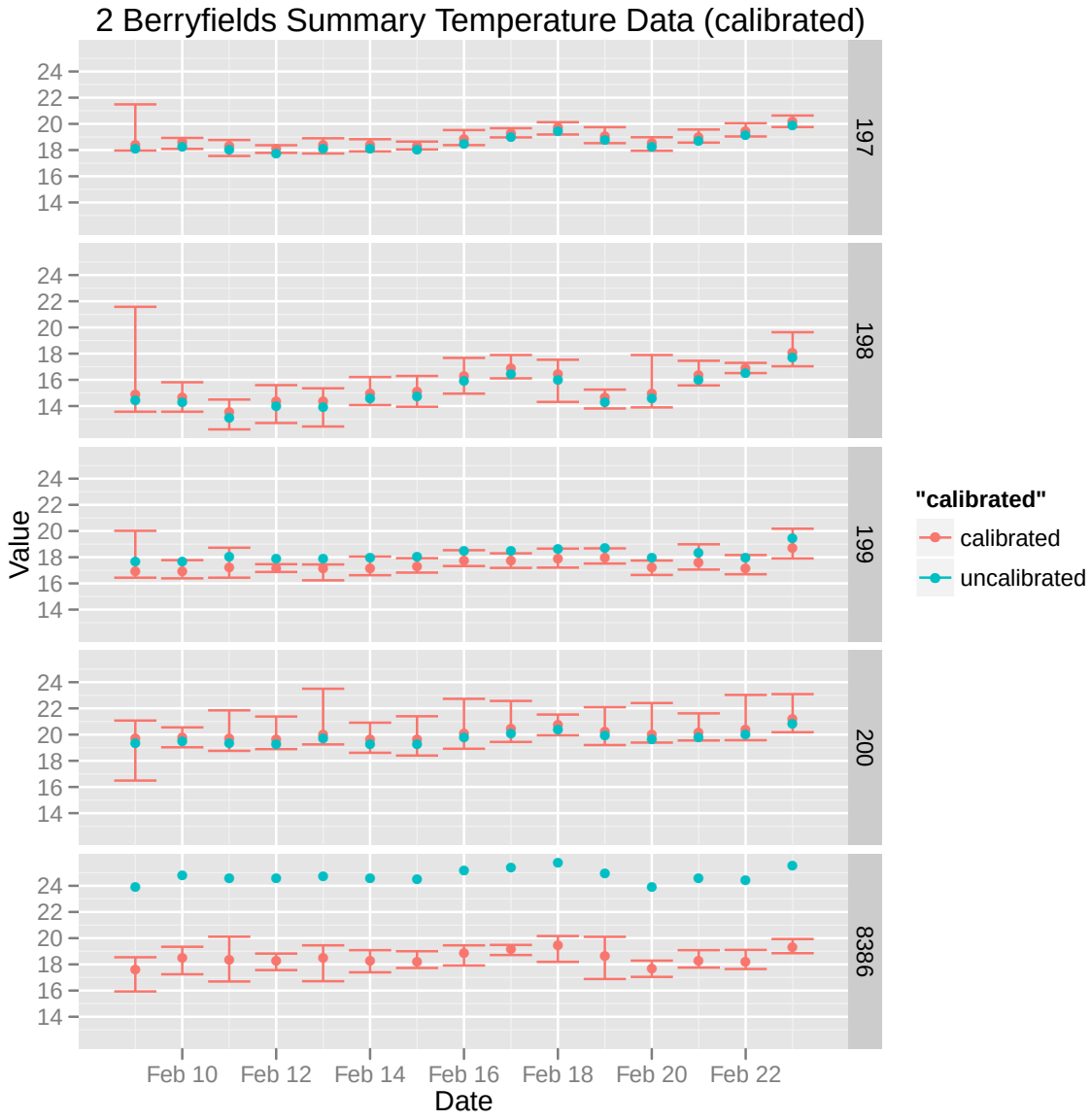


Figure 2: Calibrated Temperature Data

```
plt <- ggplot(subset(fixCalib, type == 0))

## Error:  object 'fixCalib' not found

plt <- plt + geom_point(aes(dt, calibValue, color = badValue))
plt <- plt + ylab("Value") + xlab("Date")
plt <- plt + opts(title = "Calibrating and Removing 'Bad' data")
plt <- plt + facet_grid(nodeId ~ .)
print(plt)

## Error:  object 'calibValue' not found
```

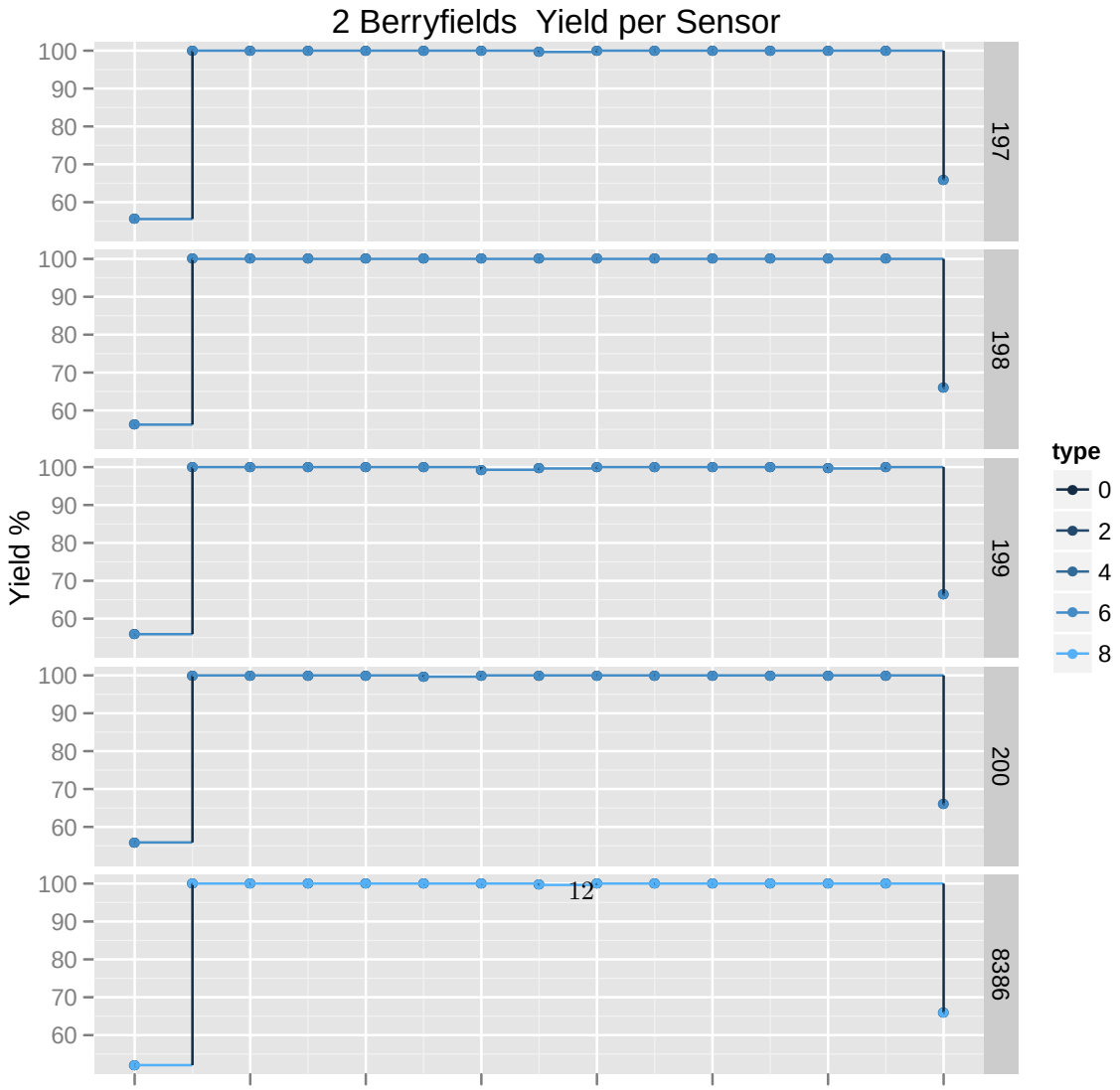
Figure 3: Calibrate and Remove "bad" samples

```
# Work out the first and last samples so we can get yields for each
# node / Day print('FIRST AND LAST SAMPLES') First and Last Samples
firstSample <- as.character(min(houseSummary$dt))
lastSample <- as.character(max(houseSummary$dt))
houseData[rowNo, ]$dataStart <- firstSample
houseData[rowNo, ]$dataEnd <- lastSample

# Error check, if there is no start date / end date in the
# database, We use the date of the first and last sample
hSd <- as.POSIXlt(theHouse$sd, tz = "GMT")
if (is.na(hSd)) {
    hSd <- as.POSIXlt(firstSample, tz = "GMT")
}
hEd <- as.POSIXlt(theHouse$ed, tz = "GMT")
if (is.na(hEd)) {
    hEd <- as.POSIXlt(lastSample, tz = "GMT")
}

# Calculate the Yield per Node / Sensor / Day
houseSummary$dayYield <- (houseSummary$count/288) * 100

plt <- ggplot(houseSummary)
plt <- plt + geom_point(aes(dt, dayYield, color = type))
plt <- plt + geom_step(aes(dt, dayYield, color = type))
plt <- plt + ylab("Yield %") + xlab("Date")
plt <- plt + opts(title = paste(hseName, " Yield per Sensor"))
plt <- plt + facet_grid(nodeId ~ .)
plt
```



```
# Averge out the Yields by Node / Location / Date (IE Combine all
# Sensors together)
avgYield <- ddply(houseSummary, .(dt, nodeId, locationId), summarise,
    min = min(dayYield), max = max(dayYield), dayYield = mean(dayYield))

dayCountId <- summaryData[which(summaryData$name == "Day Count"), ]$id
dayCountCleanId <- summaryData[which(summaryData$name == "Day Count (Clean)"),
    ]$id
# print(paste('Day Count Id ',dayCountId)) print(paste('Day Count
# Clean Id ',dayCountCleanId))

plt <- ggplot(avgYield)
plt <- plt + geom_point(aes(dt, dayYield))
plt <- plt + geom_errorbar(aes(dt, ymin = min, ymax = max))
plt <- plt + facet_grid(nodeId ~ .)
print(plt)
```

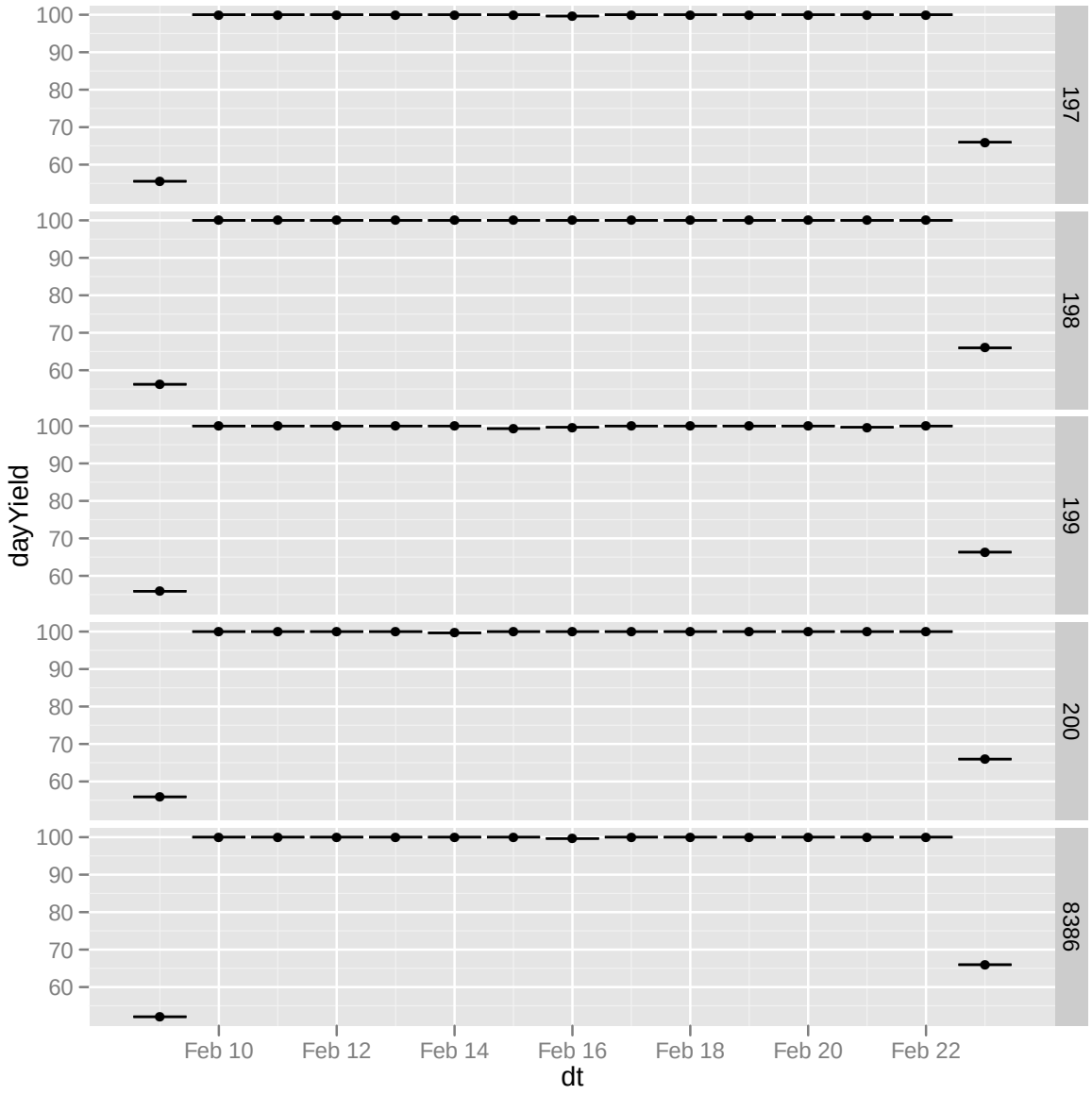


Figure 5: Yield per Node