DSCI 451 SemProj 3: Predictors and Responses of Optical Coating Performance

Michael Morken

12/06/2016

Contents

1 Introduction to the DATA

I am a masters student in the physics entrepreneurship program working at VisiMax Technologies. As you have heard throughout the semester I am working on looking at what factors affect the performance of optical coatings. As I have narrowed things down moving towards a deliverable for this semester project I have been looking exclusively at a coating that is a 7 layer anti-reflection stack on fused silica wafers. If you remember earlier I had a bunch of data for coatings on different polymers and was presenting some general summary statistics and a convoluted SQL schema through which I thought I could access my data set. I ended up running into trouble sorting out the SQL linking problem and had to write programs to pull data out of excel sheets that the company also maintained to produce a data set for one type of anti-reflection coating that is produced in two different coating chambers. My raw data contains the machine data file for the coating run and the spectral data of the final piece of glass. I have read each of these files and created variables for the various measurements inside of the coating chamber for each layer of the 7 layer deposition and variables for the shape of the scans of the reflection of the coating that come from a spectrometer. My data set consists of 86 observations or pairs of files for one machine and 94 observations for the other machine. I was not able to get a larger reliable data set but I still have hopes in the future to connect the data processing pipeline that I have developed here to the SQL database and run this kind of analysis on the 100,000 runs of this coating that are produced annually.

To process the raw files I ended up using the Python programming language. It was a bit faster than R but the size of the files (~50KB) was not a factor in choosing Python. I just felt more comfortable reading and parsing files in Python so somewhere in the middle of trying to read the raw machine files that were not in a nice format to process in R I just tried Python and never went back. Frankly I couldn't get away from thinking in list comprehensions and worked initially in Python. However once I read the raw files into a CSV file that could be read into a data frame I went back to R to finish the analysis. It proved to be far superior with its amazing packages for performing multiple linear regressions and selecting and testing models. I show select portions of Python and R code chunks in this write up. The full scripts are available in my project repository.

1.1 Linking Machine Data Files and Spectra Files

I originally hoped to use an SQL database to link the machine files to the spectra files. However the schema was not setup in such a way that a proper junction table existed to complete the type of join that I needed to do to build my data frame. I ended up having to write a script that read the internal run number out of the spectral file, looked up that run number in an excel spreadsheet (referred to as a run log in the comments), and matched the time stamp of the spreadsheet to the time stamp of the machine file. Python's list comprehension functionality ended up aiding me in this endeavor and was primary why Python was favored over R here. Without going into too much explanation here is the code for this linking part of the process:

```python
#!/usr/bin/python3
import os, shutil, datetime, openpyxl, ntpath

#machine = 'TLF'  #Switch this to move the files for the other machine
machine = 'DLF'
spectraPath = "./data/spectra/" + machine

#runLogPath = "./data/runlog/Run Log TLF & Z - 6-15-12.xlsx"
runLogPath = "./data/runlog/Run Log DLS & DLF - 3-23-13.xlsx"

def getRunTime(runNumbers, runLogPath):
    '''
    This opens the runLog and produces a list of run times for a list of run numbers.
    '''
    wb = openpyxl.load_workbook(runLogPath)
    sheet = wb.get_sheet_by_name(machine)
    runTimes = []
    for i in range(10,sheet.max_row):
        for runNumber in runNumbers:
            if runNumber == sheet.cell(row=i,column=2).value:
                try:
                    #print(sheet.cell(row=i,column=9).value)
                    runTimes.append(datetime.datetime.combine(
                                        datetime.datetime.strptime(
                                        runNumber.split('-')[0],"%y%m%d"),
                                        sheet.cell(row=i,column=9).value))
                except:
                    runTimes.append('N/A')
    return runTimes

#List Spectra paths and get runnumbers and times
spectra = [os.path.join(spectraPath,f) for f in os.listdir(spectraPath) if '.asc' in f]
runNumbers = [ntpath.basename(f).split('.')[0][0:9].strip(' ') for f in spectra]

#runTimes = getRunTime([runNumbers[runNumbers.index('151216-4')]],runLogPath)
```

```
#print(runTimes) #Testing Prints

runTimes = getRunTime(runNumbers, runLogPath)

## List Machine Files and get closest run Times - Machine Time within an hour, Moxtek in Recipe

#Get machine file times
machineFilesPath = "./data/fullMachineData/" + machine
machineFiles = [os.path.join(machineFilesPath,f) for f in os.listdir(machineFilesPath) if '.RUN' in f]
machineFileTimes = [datetime.datetime.strptime(ntpath.basename(x)[0:17],
                    "%Y%m%d_%H-%M-%S") for x in machineFiles]

#Get the index of the time closest to the run time and then move that machine file with the name
#of the run number plus Machine

indexMachineFiles = []
for runTime in runTimes:
    copyFile = False
    try:
        closest = sorted(machineFileTimes, key=lambda d: abs(runTime - d))[0]
        #print(closest)
        if abs((runTime - closest).total_seconds()) < 3600*2:
            index = machineFileTimes.index(closest)
            with open(machineFiles[index], encoding = 'latin-1') as f:
                if 'moxtek' in f.readline().lower():
                    copyFile = True
            if copyFile:
                #print(machineFiles[index])
                shutil.copyfile(machineFiles[index],"./data/machine/" + machine + "/" + runNumbers[runTi
                               + " " + machine + ".RUN")
    except:
        print('cannot match this run Number:' + runNumbers[runTimes.index(runTime)]  + "on the " + machi


print('It worked! The files have been Moved!')
```

Running the above script would give me a directory populated with the machine files named with the internal run number so that I could match them to the spectral files. Essentially it created a linked pair of files for each machine with machine data files and spectra file named consistently so that they could be cross referenced.

1.2 Machine Variables

The machine files are essentially a continuous readout of a multitude of different sensors on the inside of a vacuum chamber. In order to turn this soup of information into a data frame I parsed these files between the time when the shutter was open or closed, in other words, when the machine was depositing material and actually "making"" the coating. Then I synthesized this information into variables for each layer that I add to my data frame. This is what one thin layer of deposition would look like in a machine file. I have slightly edited the lines so that it appears here in a sensible format.

```
09:24:16 00:43:40 shutter open
                  Pressure Temper. Power Thickness  Rate
                      (Pa)    (C)    (%)    (nm)    (nm/s)
```

```
09:24:16 00:43:40  1.91E-3  446.5   20.0     0.00    0.00
09:24:21 00:43:45  2.15E-3  446.0   25.3     0.20    0.08
09:24:26 00:43:50  2.14E-3  446.5   19.9     2.00    0.29
09:24:31 00:43:55  2.11E-3  446.1   19.9     3.00    0.20
09:24:36 00:44:00  2.08E-3  446.5   19.3     4.00    0.22
09:24:41 00:44:05  2.07E-3  446.0   19.1     5.10    0.20
09:24:46 00:44:10 shutter close - last thickness read (nm) = 5.90
09:24:46 00:44:10  1.95E-3  446.5    0.0     0.00    0.21
09:24:48 00:44:12 summary: Deposition Time=00:00:30  Completion Mode=NORMAL
                           Rate=0.19 nm/s  Rate Dev.=-0.01 nm/s    Thickness=5.80 nm
                           Average Power=20.5 %  Ending Power=18.9 %
```

The code that I wrote to parse variables for each deposition layer would return the following 17 variables for this deposition layer:

```
"Layer1_MaxPressure","Layer1_MinPressure","Layer1_AvgPressure","Layer1_StdPressure",
"Layer1_MaxPower","Layer1_MinPower","Layer1_AvgPower","Layer1_StdPower",
"Layer1_MaxRate","Layer1_MinRate","Layer1_AvgRate","Layer1_StdRate",
"Layer1_MaxO2Flow","Layer1_MinO2Flow","Layer1_AvgO2Flow", "Layer1_StdO2Flow",
"Layer1_FinalThickness"
```

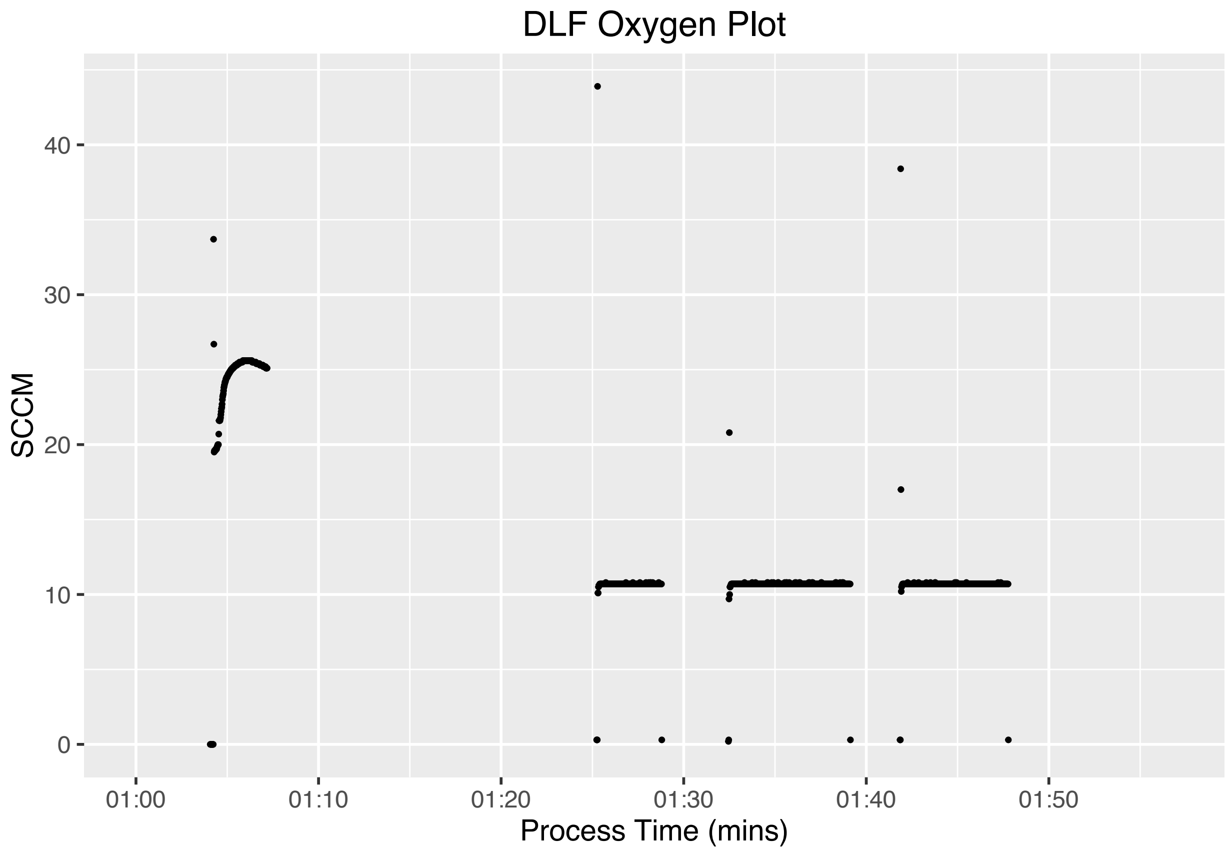
With 7 total layers in the following anti-reflection stack I would end up with a total of 119 layer variables and the addition of three variables characterizing all of the layers that make up this given coating and help to label this specific batch:

```
"run_number", "machine", "runTimeStamp"
```

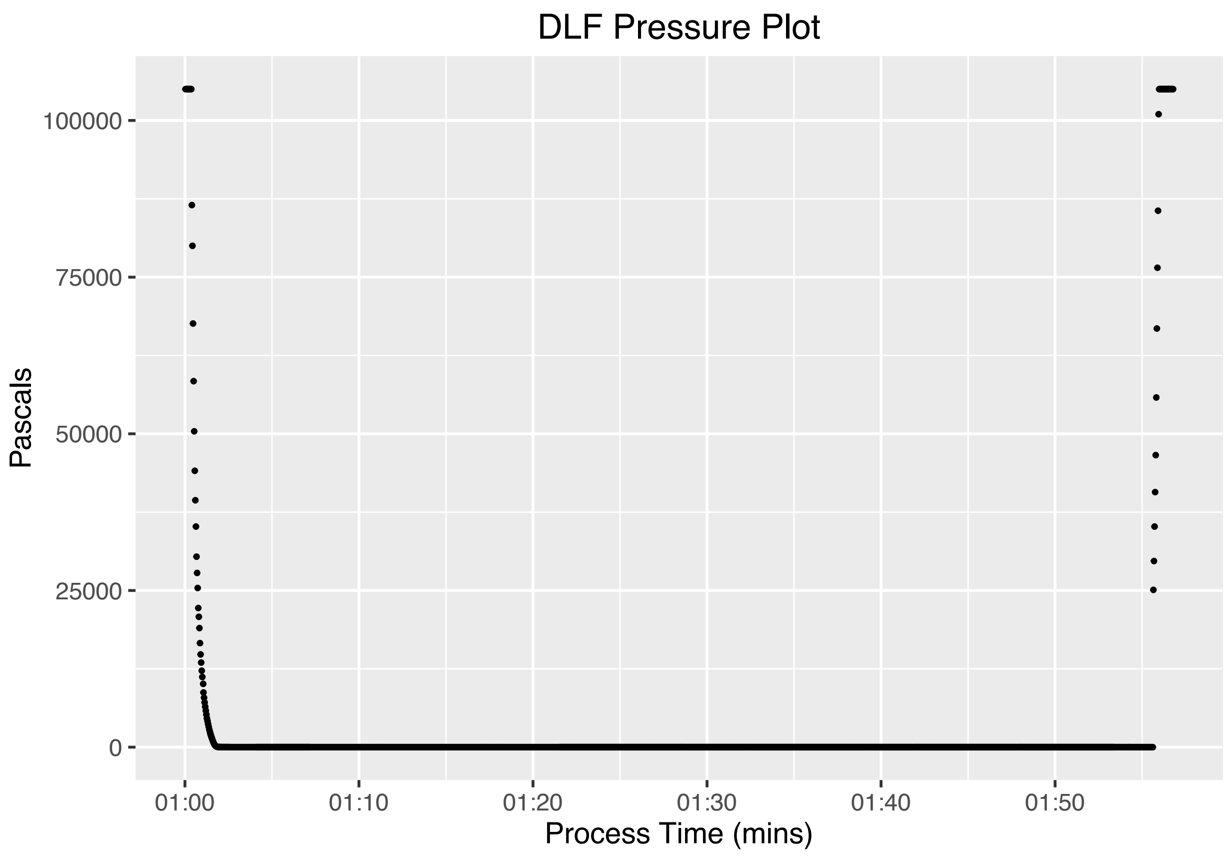
This gives me a total of 122 variables that come from the machine file. Since I am trying to infer how different machine settings affect the final performance of the optical coating these variables can be directly changed with difference machine control settings are my “predictors” for each observation.

I can plot one of the machine variables through the deposition to get a picture of what is going on.

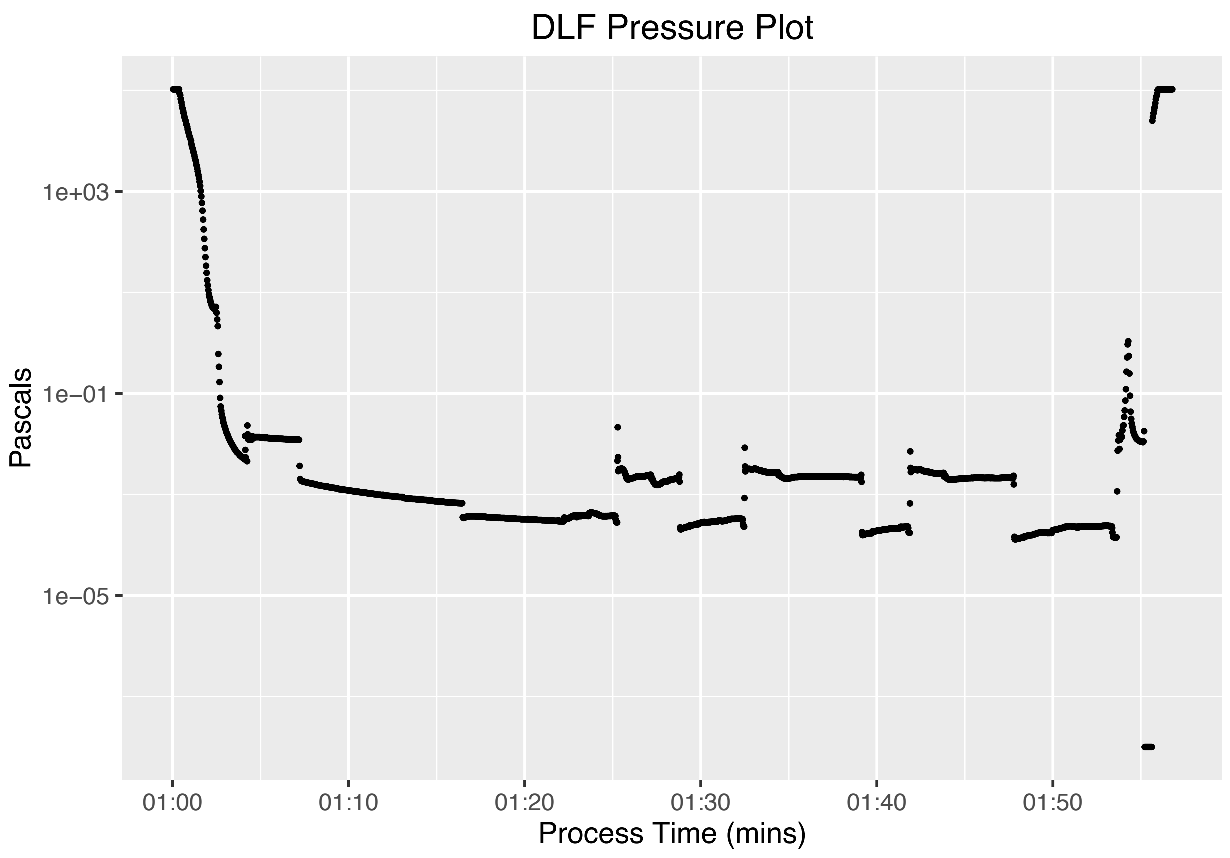
```r
library(ggplot2)
mungedData <- function(filename) {
    x <- read.table(filename, sep = "", header = F, fill = T, na.strings = "",
        stringsAsFactors = F)
    df <- data.frame(matrix(0, nrow = nrow(x), ncol = 100))
    df[, 1:12] <- x
    ind_temp1 <- grep("[A-Za-z]|\\(", x[, 1])  #Get any non-numeric entries
    ind_temp2 <- grep("[A-Za-z]|\\(", x[, 2])  #Get any non-numeric entries
    ind_temp3 <- grep("[0-9]|\\(", x[, 3])  #Find all the numeric values
    ind_temp4 <- grep("IonGun", x[, 3])  #Find all instances of 'IonGun'
    ind_temp5 <- grep("(sccm)", x[, 6])  #Find index of all '(sccm)' - will be used to shuffle the gass
    df[ind_temp1, 14:27] <- x[ind_temp1, 1:12]  #Shift all non-numeric entries to the right half of the
    df[ind_temp1, 1:13] <- NA  #Delete these entries in the left half of the data frame
    df[ind_temp2, 14:27] <- x[ind_temp2, 1:12]  #Shift all non-numeric entries to the right half of the
    df[ind_temp2, 1:13] <- NA  #Delete these entries in the left half of the data frame
    df[-(ind_temp3), 14:27] <- x[-(ind_temp3), 3:12]  #Any entry not containing a numeric value is shif
    df[-(ind_temp3), 3:13] <- NA  #Delete the left half of the data frame (disregarding the first two c
    df[ind_temp4, 8] <- x[ind_temp4, 7]
    df[ind_temp4, 9:13] <- x[ind_temp4, c(4, 5, 6, 8, 9)]
    df[ind_temp4, 15:20] <- NA
    df[(ind_temp5 + 1), 11] <- x[(ind_temp5 + 1), 6]
    df[1, 19:20] <- df[1, 1:2]
    df[1, 19:20] <- NA
    df$X1[is.na(df$X1)] <- "00:00:00"
    df$X2[is.na(df$X2)] <- "00:00:00"
    format_dmy = "%d/%m/%Y"
    format_hms = "%H:%M:%S"
    recipe_date <- as.POSIXct(df$X1[1], format = format_dmy)
    start_time <- as.POSIXct(df$X2[1], format = format_hms)
    df$X1 <- as.POSIXct(df$X1, format = format_hms)
    df$X2 <- as.POSIXct(df$X2, format = format_hms)
    df[, 3:13] <- apply(df[, 3:13], 2, as.numeric)

    return(df)
}

library(scales)
df <- mungedData("./data/machine/DLF/151119-2 DLF.RUN")
ggplot(df, aes_string(x = "X2", y = "X8")) + ggtitle("DLF Oxygen Plot") + geom_point(size = 0.5) +
    ylab("SCCM") + xlab("Process Time (mins)") + scale_x_datetime(date_breaks = "10 mins",
    labels = date_format(("%H:%M"), tz = "Etc/GMT+4"), limits = c(df$X2[2],
        df$X2[length(df$X2)]))
```



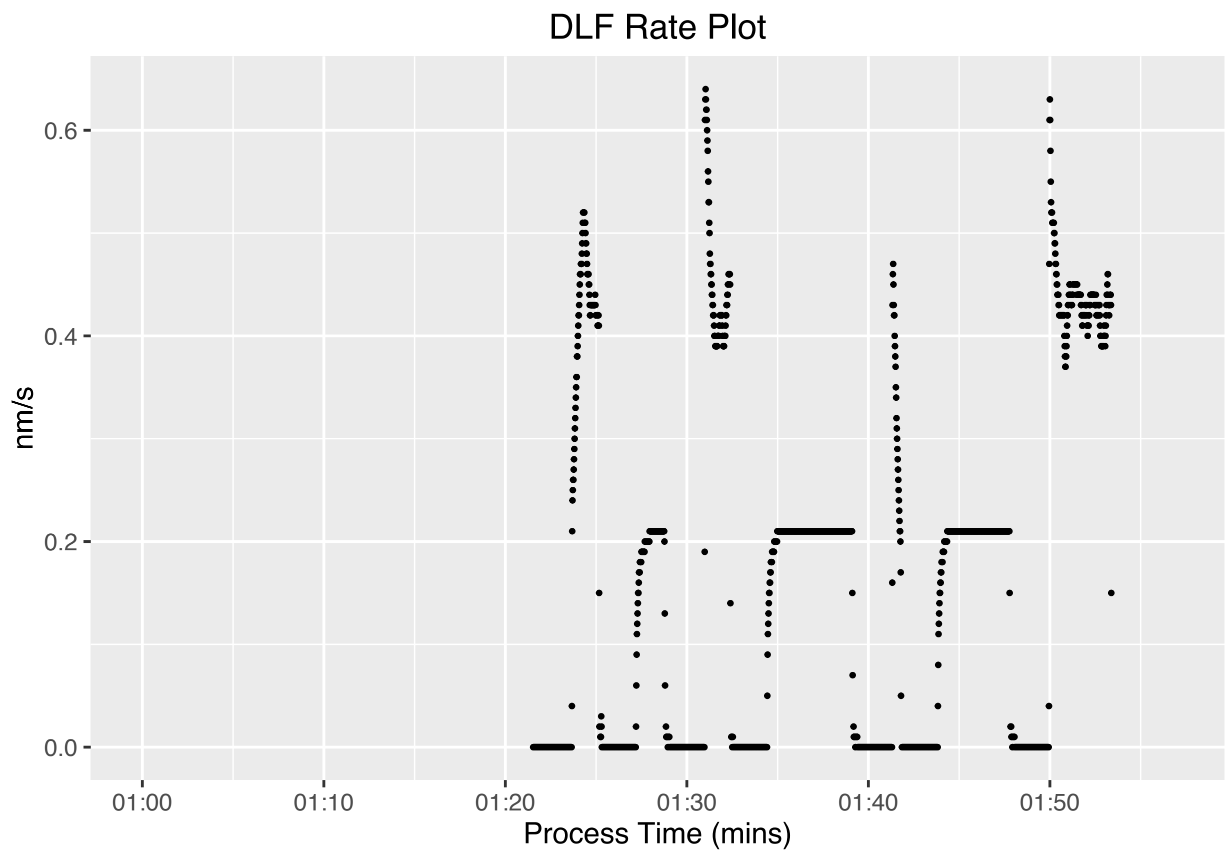
```
ggplot(df, aes_string(x = "X2", y = "X3")) + ggtitle("DLF Pressure Plot") +
    geom_point(size = 0.5) + ylab("Pascals") + xlab("Process Time (mins)") +
    scale_x_datetime(date_breaks = "10 mins", labels = date_format(("%H:%M"),
        tz = "Etc/GMT+4"), limits = c(df$X2[2], df$X2[length(df$X2)]))
```



```
ggplot(df, aes_string(x = "X2", y = "X3")) + ggtitle("DLF Pressure Plot") +
    geom_point(size = 0.5) + ylab("Pascals") + xlab("Process Time (mins)") +
    scale_x_datetime(date_breaks = "10 mins", labels = date_format(("%H:%M"),
        tz = "Etc/GMT+4"), limits = c(df$X2[2], df$X2[length(df$X2)])) + scale_y_log10()
```



```r
ggplot(df, aes_string(x = "X2", y = "X7")) + ggtitle("DLF Rate Plot") + geom_point(size = 0.5) +
    ylab("nm/s") + xlab("Process Time (mins)") + scale_x_datetime(date_breaks = "10 mins",
    labels = date_format(("%H:%M"), tz = "Etc/GMT+4"), limits = c(df$X2[2],
        df$X2[length(df$X2)]))
```

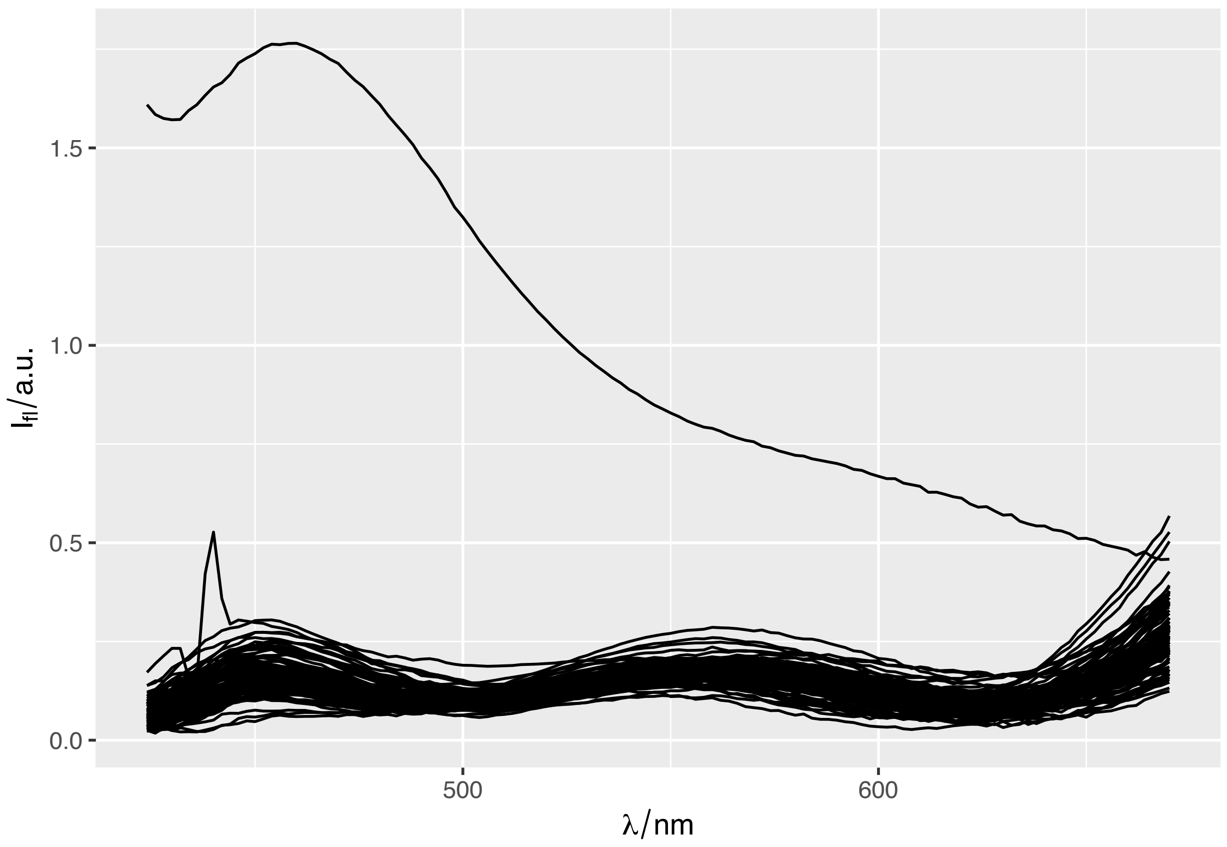


1.3 Spectra Variables

As I alluded to above if the machine variables are the predictors for an observation the responses are the optical performance. This is characterized by a file that is generated with a spectrometer where the reflection is measured every two nano meters from 424nm - 670nm. With this file, like with the machine file, I wrote a function to define variables that will represent the optical performance. These variables go into the data frame for each observation in addition to all of the machine variables. After looking at the plots of the spectra I ended up making different spectral variables based on the machine that produced the coating. Here are the graphs that lead me to this conclusion. The names of the two different machines are the TLF and DLF.

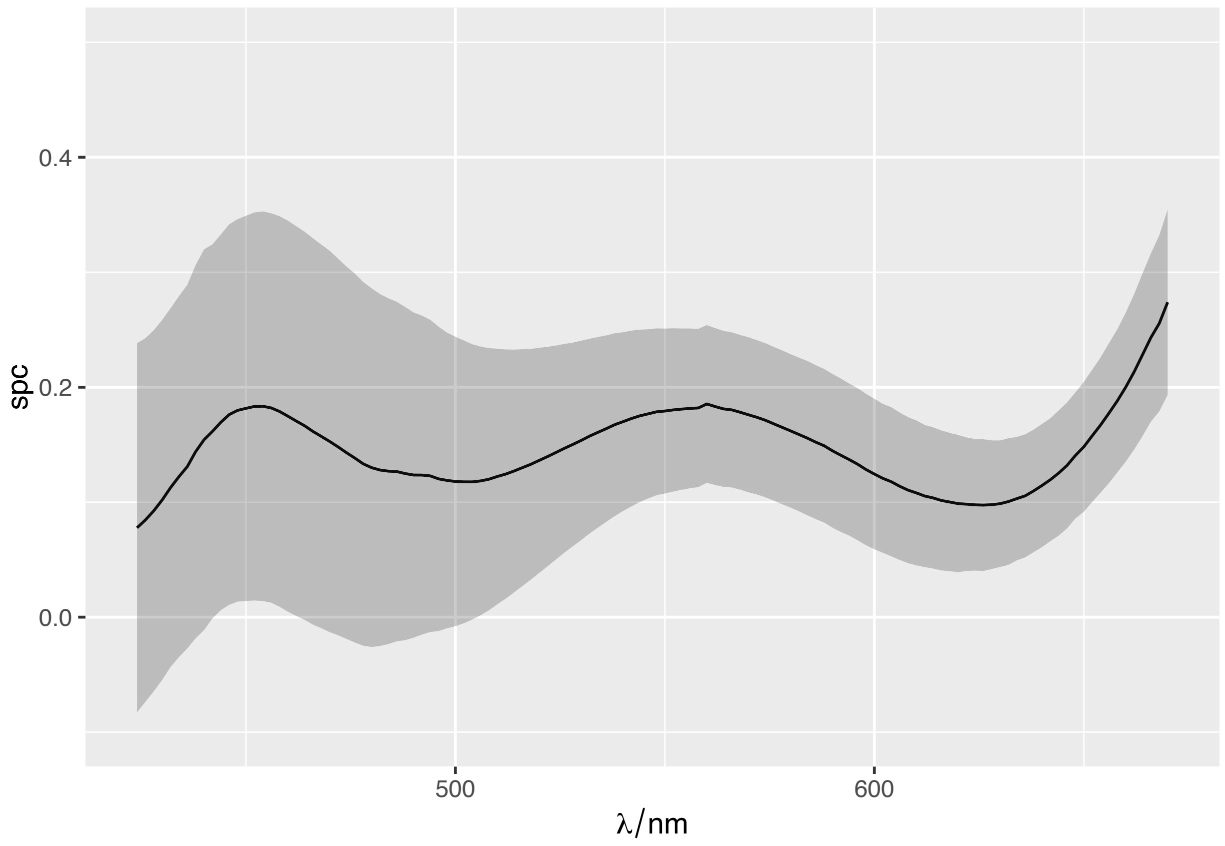
1.3.1 DLF Spectra

```
library(ggplot2)  #plotting
library(hyperSpec)  #This package will make quick work of the spectra
TLF <- read.csv("./csvFiles/TLF.csv")  #These are already ordered by timestamp from the python script
DLF <- read.csv("./csvFIles/DLF.csv")  #These are already ordered by timestamp from the python script

source("./importSpectraFiles.R")  #Custom import function I wrote for my files to make hyperspec object
spectraDLF <- importSpectra(as.vector(DLF$spectraFile), skip = 90, nmax = 248)
spectraTLF <- importSpectra(as.vector(TLF$spectraFile), skip = 90, nmax = 248)

qplotspc(spectraDLF, spc.nmax = length(spectraDLF))  #All of the spectra from the DLF
```



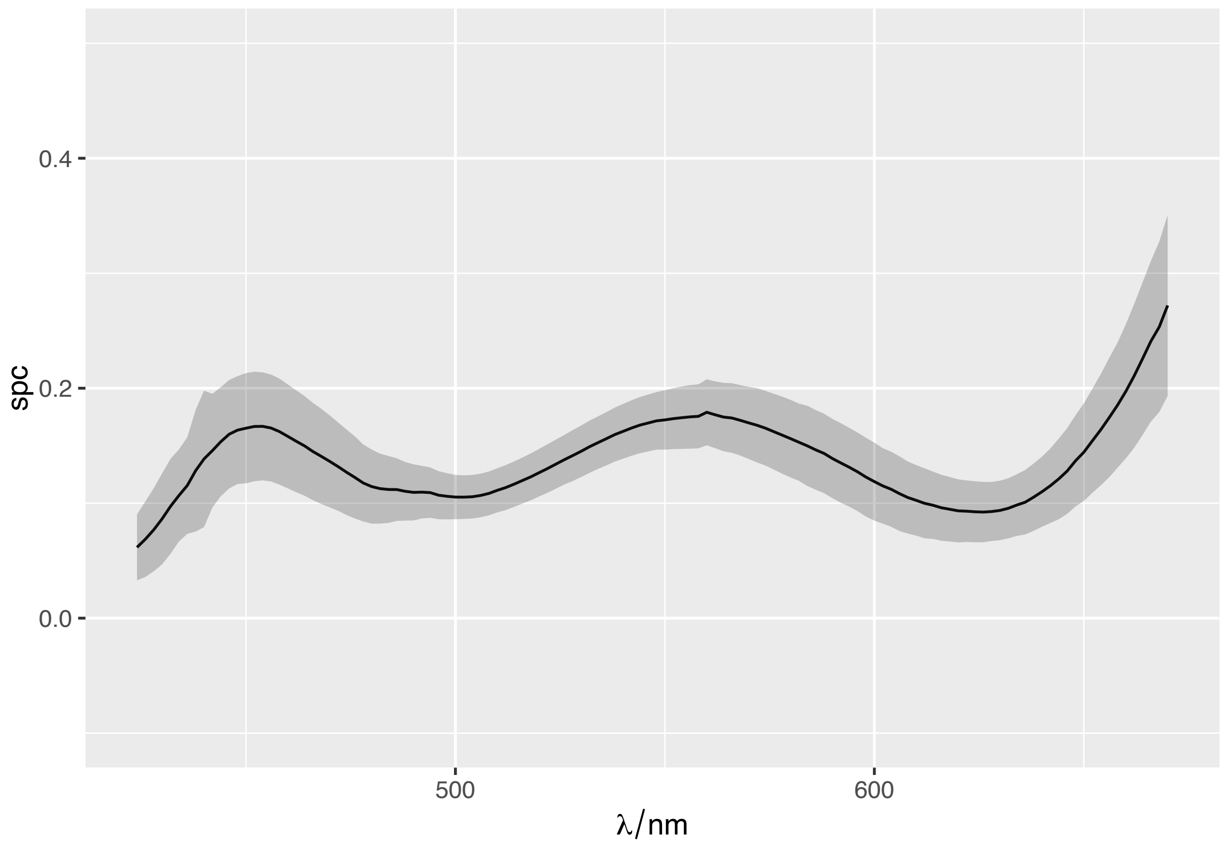
Here we can see one outlier that I need to investigate. Also, Looking at the right most side of the plot or the high wavelength end, we can see a group of three spectra that start a bit higher than the others. Aside from that we have a nice grouping. Since these were made in the same machine this is what we expect to find and they have a consistent shape. To get a better idea of what variables or responses to make from the spectra files I plotted the mean curve and the standard deviation around it:

```
qplotspc(mean(spectraDLF)) + geom_ribbon(aes(ymin = mean + sd, ymax = mean -
    sd, y = 0, group = NA), alpha = 0.25, data = as.t.df(mean_sd(spectraDLF))) +
    xlim(424, 670) + ylim(-0.1, 0.5)
```



I noticed that the one outlier has a drastic effect on this standard deviation around this mean. Here is the same plot with that point removed:

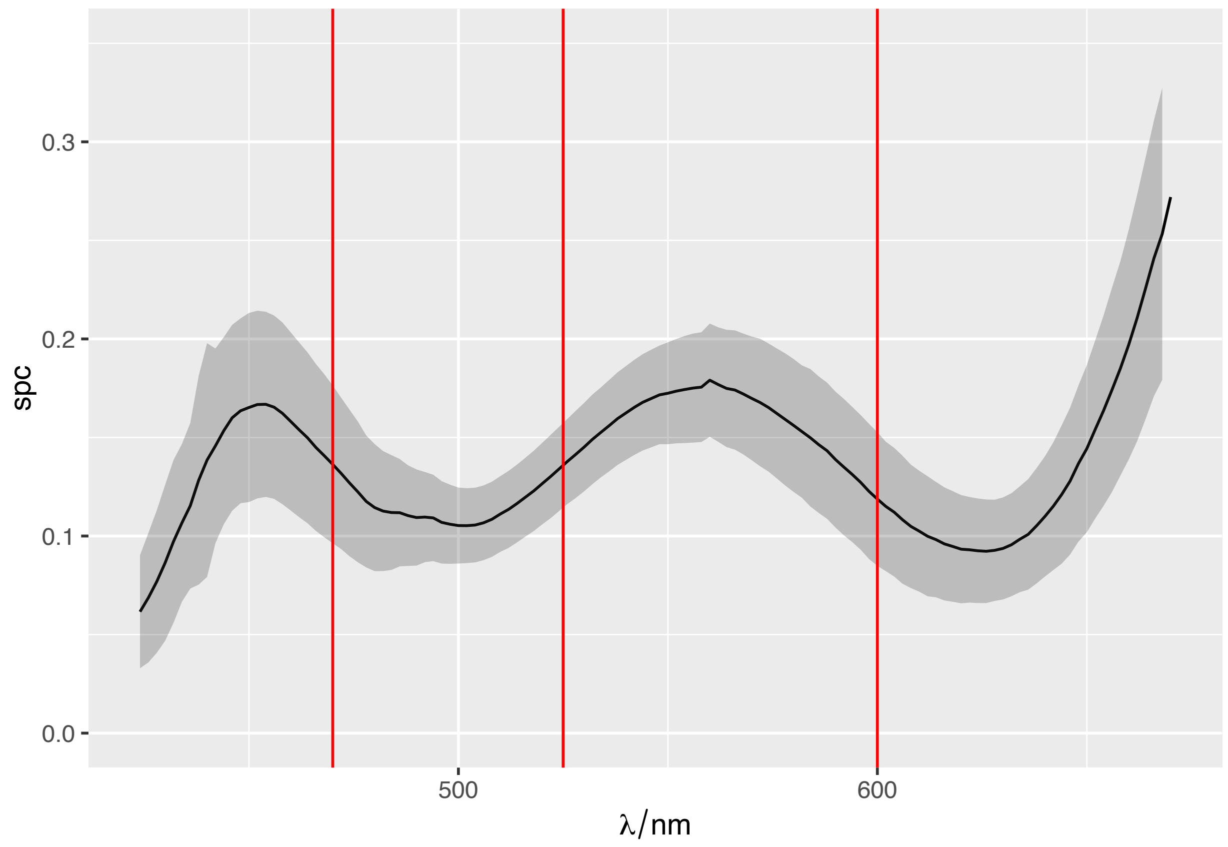
```
noOutlierDLF <- DLF[-c(which.max(DLF$rAvg)), ]

spectraDLF_noOutlier <- importSpectra(as.vector(noOutlierDLF$spectraFile), skip = 90,
    nmax = 248)

qplotspc(mean(spectraDLF_noOutlier)) + geom_ribbon(aes(ymin = mean + sd, ymax = mean -
    sd, y = 0, group = NA), alpha = 0.25, data = as.t.df(mean_sd(spectraDLF_noOutlier))) +
    xlim(424, 670) + ylim(-0.1, 0.5)
```



Here we see a much tighter grouping by removing that one erroneous run. Now looking into that outlier on this machine is something that will be important for this analysis. Especially since the machine file and the scan file seem to be genuine from what I can tell at a cursory investigation. The only thing that seems suspicious is the fact that the run was put into the machine at 1:31 AM and wasn't scanned until 6:43 AM. This means that someone working third shift waited until the end of their shift to scan this part. While it isn't completely convicting, it suggests that perhaps there is a greater than average chance that this scan was misnamed or misplaced as shifts were changing on the production floor. Anyway I will use the tight grouping to determine my spectral variables for the DLF Spectra, which are going to be my predictors.

I am going to break each curve into 4 distinct groups to try to capture the features of the graph. Looking again at the graph I have labeled the different groups that I am going to break the spectra into:

```r
qplotspc(mean(spectraDLF_noOutlier)) + geom_ribbon(aes(ymin = mean + sd, ymax = mean -
    sd, y = 0, group = NA), alpha = 0.25, data = as.t.df(mean_sd(spectraDLF_noOutlier))) +
    xlim(424, 670) + ylim(0, 0.35) + geom_vline(xintercept = 600, colour = "red") +
    geom_vline(xintercept = 525, colour = "red") + geom_vline(xintercept = 470,
    colour = "red")
```

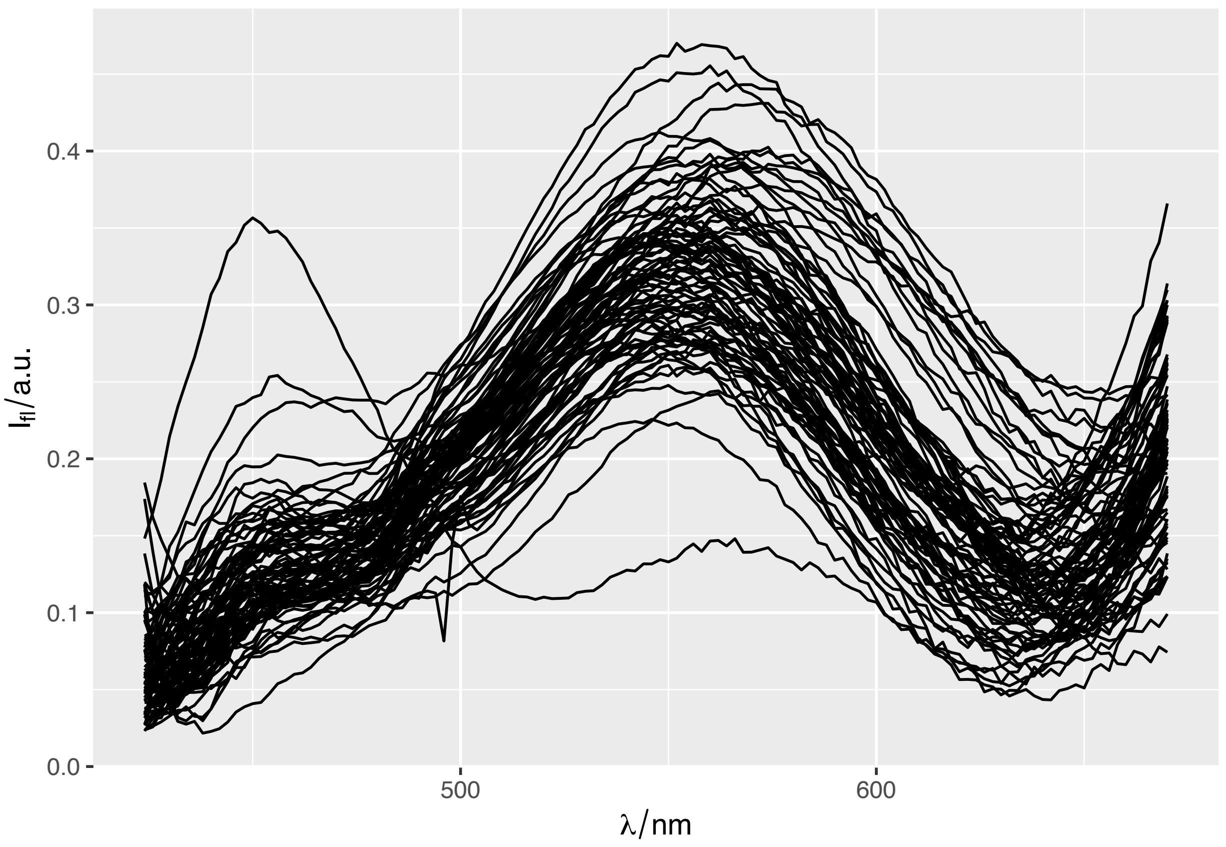


```
The regions that I am describing can be shown between the vertical bars on the above plot. In each regi
```

1.3.2 TLF Spectra

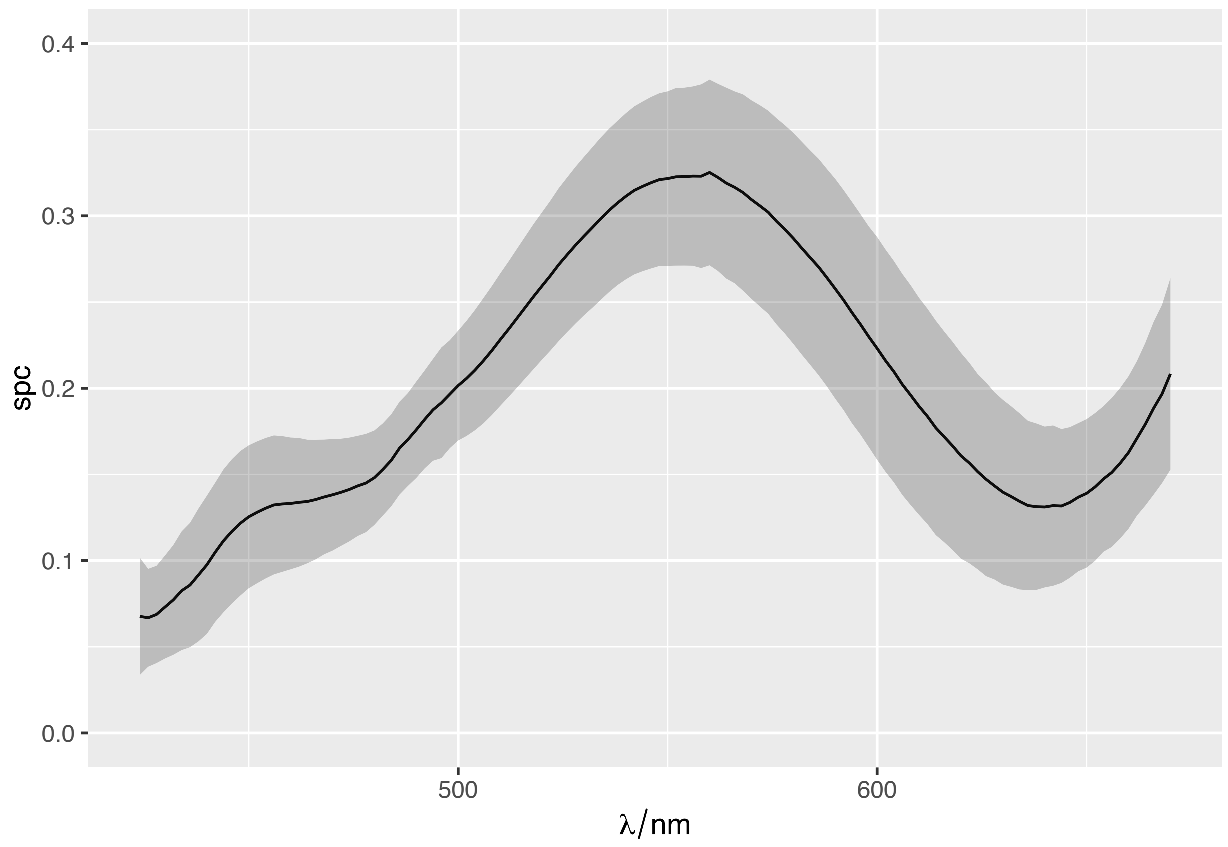
I am treating the spectral response variables slightly differently for each machine. This is because on an initial inspection the curves that these two machines were producing were slightly different and I wanted to try to tune my analysis to the particularities between coating chambers, as each one has its quirks and is setup slightly differently. So while the machine file readouts are standardized since the machines are made by the same company the shape of the curves that they are making are a bit different. here is the plot of all spectra I have in my data set for the TLF:

```r
qplotspc(spectraTLF, spc.nmax = length(spectraTLF))  #All of the spectra from the TLF
```



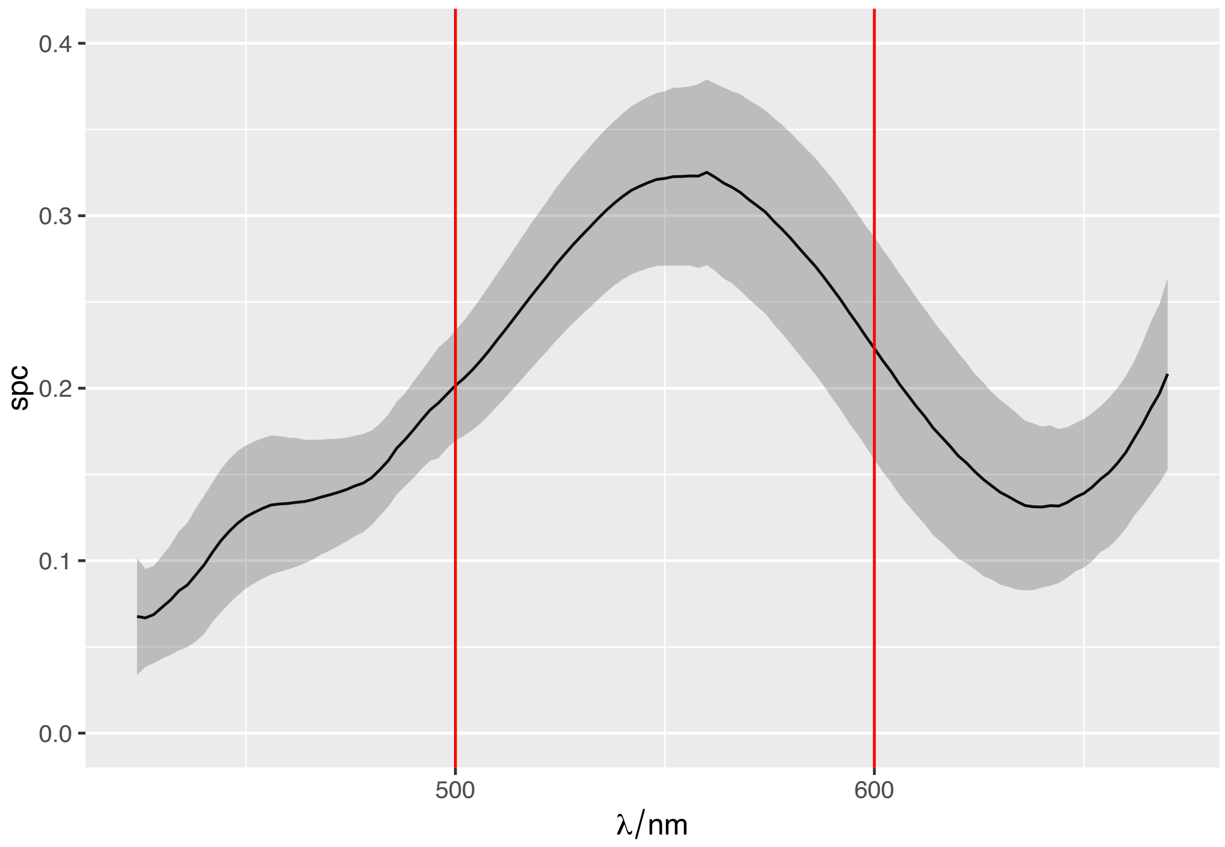
Here we don't see any crazy outliers like we saw when looking at the DLF. However there is definitely some variation in the shape of these curves that I wouldn't capture from simply doing summary statistics on the curve. In fact there seems to be a cluster of curves that seem to have a maximum value around 450 where there could be something interesting going on. To create response variables for these curves I am also going to look at the plot with the mean and standard deviation plotted to get some idea of the distribution before I start binning the curves.

```
qplotspc(mean(spectraTLF)) + geom_ribbon(aes(ymin = mean + sd, ymax = mean -
    sd, y = 0, group = NA), alpha = 0.25, data = as.t.df(mean_sd(spectraTLF))) +
    xlim(424, 670) + ylim(0, 0.4)
```



The general shape here is slightly different from the spectra on the DLF so I am going to bin it differently to make my response variables.

```
qplotspc(mean(spectraTLF)) + geom_ribbon(aes(ymin = mean + sd, ymax = mean -
    sd, y = 0, group = NA), alpha = 0.25, data = as.t.df(mean_sd(spectraTLF))) +
    xlim(424, 670) + ylim(0, 0.4) + geom_vline(xintercept = 600, colour = "red") +
    geom_vline(xintercept = 500, colour = "red")
```

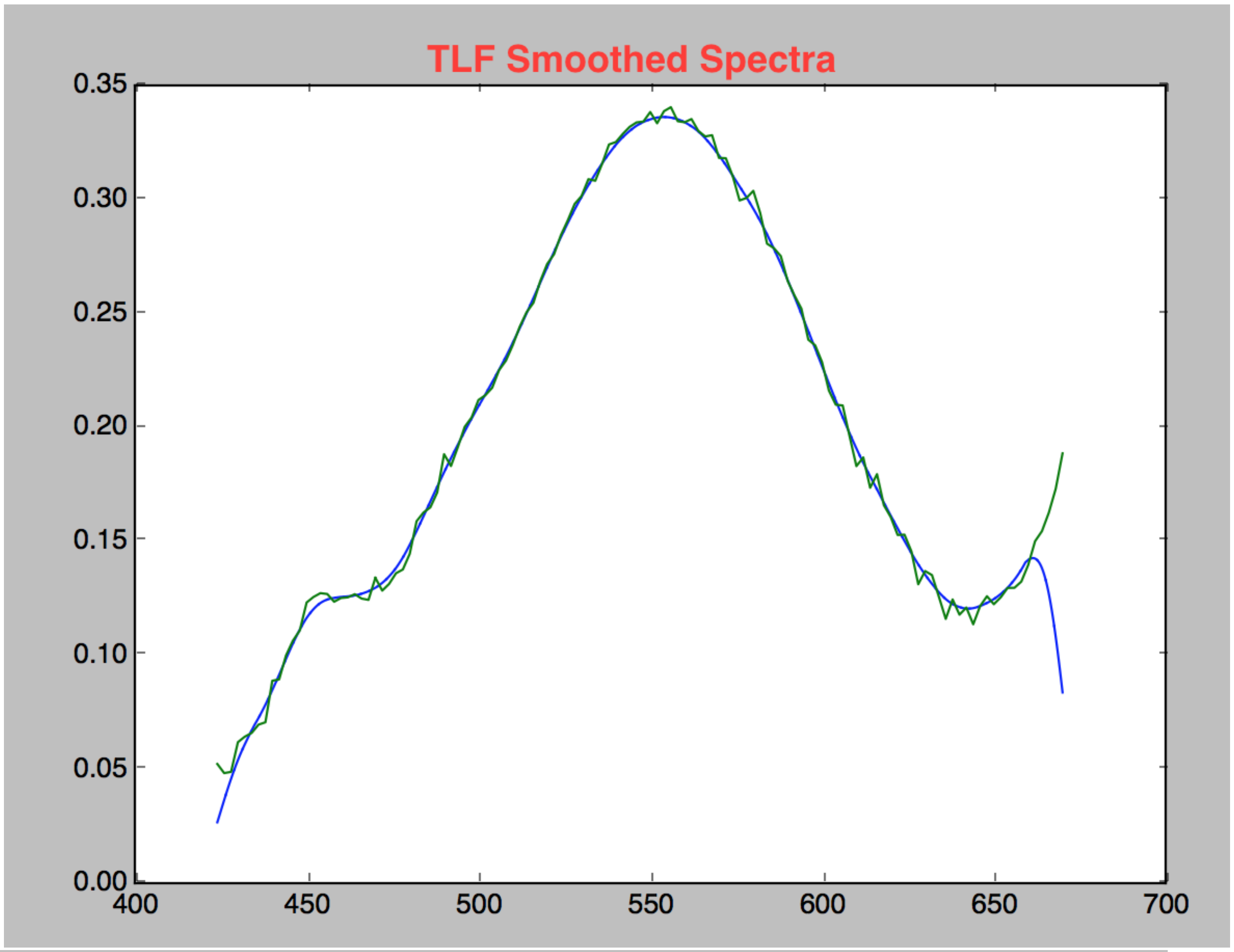


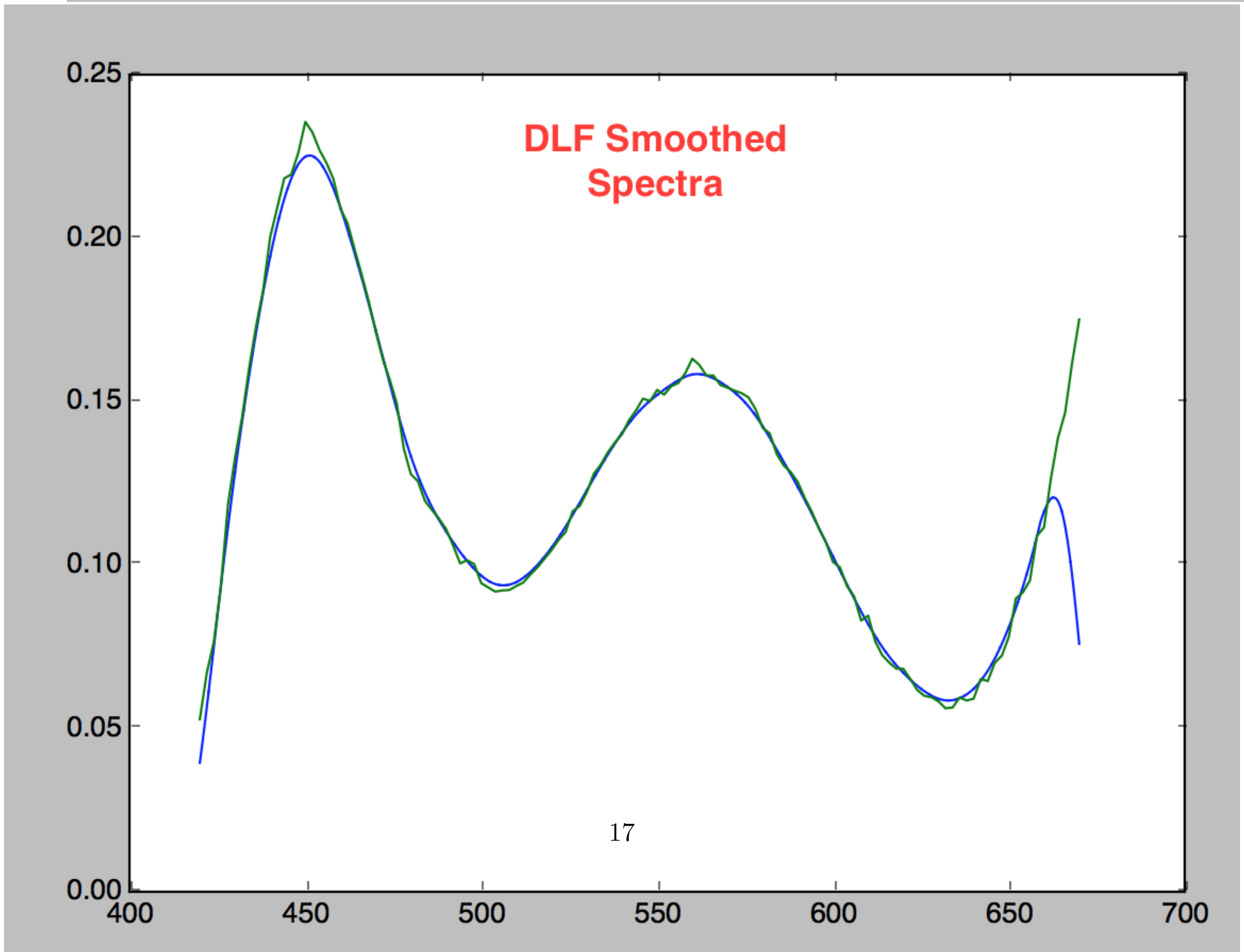
This one looked like it would make sense to split into three bins however there is a feature around 450nm that I am not able to capture with this binning system. Looking at the over-plot of all of the individual spectra I can see that some of the spectra have a maximum right around 450 that I would like to capture. I think that I will keep the bins like this but depending on the height of the 450nm maxima this setup might not capture what I want, which was the cluster of curves that looked different in the over-plot that I did initially. I might add in a function to find a local maximum near 450nm if I find the response variables inadequate. Like the DLF spectra I will also be calculating the summary statistics for the entire curve and fitting with a 7th order polynomial. So currently the TLF spectra will have 24 response variables.

1.3.3 Fitting the Spectra

To process the spectra I used a Python script. I used the scipy signal processing library to smooth and model the spectra. Here is my function and some images of smoothed curves from the TLF and DLF respectively.

```python
def read_spectra(spectraFilePath, machine,on):
    '''
    Gets the desired information out of a moxtek reflection scan. Because I defined the regions
    differently for the spectra coming from each machine this will need to take the machine as and
    arguement.
    '''
    #Process the Spectra file and get lists for wavelength and cooresponding Reflectance
    with open(spectraFilePath,'r') as f:
        sampleLines = f.readlines()
    wavelength = []
    reflectance = []
    for line in sampleLines[90:]:
        wavelength.append(float(line.split('\t')[0]))
        reflectance.append(float(line.split('\t')[-1].strip('\n')))

    #Smoothing and interpolating for better estimates
    interpFunction = scipy.interpolate.interp1d(numpy.array(wavelength),numpy.array(reflectance),kind=
    xx = numpy.linspace(min(wavelength),max(wavelength), 10000) #Setting up to resample with 10000 samp
    yy = interpFunction(xx) #Computing the interpolation Function over the new range
    window = scipy.signal.gaussian((max(wavelength)-min(wavelength)/2 + min(wavelength)), 200) #Make a 
    smoothed = scipy.signal.convolve(yy, window/window.sum(), mode='same') #Convolve the gaussian peak 
    #indexes = peakutils.peak.indexes(numpy.array(smoothed), thres=0.5, min_dist=50) #Find peaks in smo
    #print('Peaks are: %s' % xx[indexes]) #For Testing and verifying the threshold
    if on:
        plt.plot(xx, smoothed,numpy.array(wavelength),numpy.array(reflectance))
        plt.show()

    #Removing strange smoothing end behavior in region 600 - 670
    highEndR = smoothed[find_closest(xx,600):find_closest(xx,670)]
    highEndW = xx[find_closest(xx,600):find_closest(xx,670)]
    splitIndexHighEnd = peakutils.peak.indexes(numpy.array(highEndR), thres=0.2, min_dist=50)
    if len(splitIndexHighEnd) >0:
        wavelengthStrangePeak = highEndW[splitIndexHighEnd][0]
        splitIndex = find_closest(xx,600)
        smoothR = smoothed[:splitIndex]
        smoothW = xx[:splitIndex]
    else:
        smoothR = smoothed
        smoothW = xx
    if machine == 'DLF':
        #Finding the peaks in the smoothed data
        indexes = peakutils.peak.indexes(numpy.array(smoothR), thres=max(smoothR)-numpy.mean(smoothR), 

        #Returning Values:
        #Number of Peaks, p1w,p1r,p1rw,p2w,p2r,p2rw
        if (len(indexes) == 0) | (len(indexes) > 2):
            return ['NA']*7
        elif len(indexes) == 1:
            return [len(indexes), smoothW[indexes[0]], smoothR[indexes[0]], smoothR[indexes[0]] * smoot
        elif len(indexes) == 2:
            return [len(indexes), smoothW[indexes[0]], smoothR[indexes[0]], smoothR[indexes[0]] * smoot
    elif machine == 'TLF':
        #Finding the peaks in the smoothed data
        indexes = peakutils.peak.indexes(numpy.array(smoothR), thres=numpy.mean(smoothR)/0.3, min_dist=
        #print('Peaks are at the wavelength: %s' % smoothW[indexes])    numpy.mean(smoothR)/0.3

        #Returning Values:
        #Number of Peaks, p1w,p1r,p1rw,p2w,p2r,p2rw
        if (len(indexes) == 0) | (len(indexes) > 2):
            return ['NA']*7
        elif len(indexes) == 1:
            return [numpy.mean(numpy.array(reflectance)),len(indexes), smoothW[indexes[0]], smoothR[inde
        elif len(indexes) == 2:
            return [numpy.mean(numpy.array(reflectance)),len(indexes), smoothW[indexes[0]], smoothR[inde
```


TLF Smoothed Spectra
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
400
450
500
550
600
650
700


DLF Smoothed Spectra
0.25
0.20
0.15
0.10
0.05
0.00
400
450
500
550
600
650
700

1.4 Final Machine Dataframes

After creating the "Predictor" variables and the "Response" variables from the machine and spectral files respectively I can begin to do some analysis on how these different variables are related. Before getting into the analysis this is the dimensions of the data frames for each of the machines:

```
cat("DLF Number of Observables: ", length(rownames(DLF)), " - DLF Number of Variables: ",
    length(colnames(DLF)))
```

```
## DLF Number of Observables:  96  - DLF Number of Variables:  132
```

```
cat("\n")  #Probably only necessary to make it work in the code editor but whatever
```

```
cat("TLF Number of Observables: ", length(rownames(TLF)), " - TLF Number of Variables: ",
    length(colnames(TLF)))
```

```
## TLF Number of Observables:  84  - TLF Number of Variables:  132
```

```
head(DLF)  #For an example
```

```
##   run_number machine         runTimeStamp Layer1_MaxPressure
## 1   150211-6     DLF 2015-02-11 09:58:45           0.000374
## 2   151119-2     DLF 2015-11-19 13:34:44           0.000438
## 3   151121-2     DLF 2015-11-21 11:56:51           0.001360
## 4   151122-6     DLF 2015-11-22 05:49:43           0.001150
## 5  151123-14     DLF 2015-11-23 15:40:33           0.001200
## 6   151202-3     DLF 2015-12-02 01:49:51           0.000962
##   Layer1_MinPressure Layer1_AvgPressure Layer1_StdPressure Layer1_MaxPower
## 1           0.000358       0.0003670323       4.939313e-06             5.0
## 2           0.000370       0.0003934270       2.453243e-05             5.1
## 3           0.001100       0.0012721277       6.439361e-05             6.5
## 4           0.001090       0.0011122472       1.727197e-05             5.2
## 5           0.000973       0.0011289375       5.469762e-05             6.1
## 6           0.000804       0.0009298617       2.366953e-05             5.7
##   Layer1_MinPower Layer1_AvgPower Layer1_StdPower Layer1_MaxRate
## 1             3.9        4.101075       0.2670003           0.53
## 2             3.5        4.250562       0.5134426           0.52
## 3             3.5        5.879787       0.7557693           0.54
## 4             3.5        4.095506       0.5199201           0.54
## 5             3.5        5.500000       0.5387099           0.47
## 6             3.5        5.051064       0.4385044           0.49
##   Layer1_MinRate Layer1_AvgRate Layer1_StdRate Layer1_MaxO2Flow
## 1           0.00      0.4392473     0.06474493                0
## 2           0.00      0.4121348     0.09168181                0
## 3           0.00      0.3984043     0.11232839                0
## 4           0.00      0.4158427     0.08747146                0
## 5           0.03      0.3871875     0.07487077                0
## 6           0.00      0.3956383     0.09210954                0
##   Layer1_MinO2Flow Layer1_AvgO2Flow Layer1_StdO2Flow Layer1_FinalThickness
## 1                0                0                0                 41.68
## 2                0                0                0                 37.53
## 3                0                0                0                 37.92
## 4                0                0                0                 37.43
## 5                0                0                0                 37.61
## 6                0                0                0                 37.64
##   Layer2_MaxPressure Layer2_MinPressure Layer2_AvgPressure
## 1           0.000786           0.000264       0.0003985895
## 2           0.002340           0.001540       0.0018368421
## 3           0.002990           0.002420       0.0027170652
## 4           0.003280           0.002290       0.0026417391
## 5           0.003310           0.002140       0.0025197849
## 6           0.002000           0.001250       0.0016297802
##   Layer2_StdPressure Layer2_MaxPower Layer2_MinPower Layer2_AvgPower
## 1       6.769117e-05            22.6            21.1        21.34526
## 2       1.946509e-04            25.5            21.0        24.76842
## 3       1.853231e-04            24.3            21.0        23.64565
## 4       1.764111e-04            24.0            21.0        23.44783
## 5       2.253503e-04            24.5            21.0        23.73333
## 6       2.256711e-04            23.9            21.0        23.15714
##   Layer2_StdPower Layer2_MaxRate Layer2_MinRate Layer2_AvgRate
## 1       0.2725198           0.27           0.03      0.2127368
## 2       1.0387506           0.21           0.00      0.1907368
## 3       0.7220763           0.21           0.04      0.1964130
## 4       0.6873829           0.22           0.00      0.1960870
## 5       0.8283074           0.21           0.00      0.1941935
## 6       0.7830825           0.22           0.00      0.1958242
##   Layer2_StdRate Layer2_MaxO2Flow Layer2_MinO2Flow Layer2_AvgO2Flow
## 1     0.02273148              4.3              4.2         4.201053
## 2     0.03787075             10.8             10.7        10.708421
## 3     0.02434273             10.7             10.6        10.654348
## 4     0.03313203             10.7             10.6        10.650000
## 5     0.03290006             10.7             10.6        10.641935
## 6     0.02749889             10.7             10.6        10.635165
##   Layer2_StdO2Flow Layer2_FinalThickness Layer3_MaxPressure
## 1       0.01020564                 20.33           0.000323
## 2       0.02777033                 18.27           0.000334
## 3       0.04981061                 18.25           0.000951
## 4       0.05000000                 18.32           0.000951
## 5       0.04934535                 18.17           0.001050
## 6       0.04774849                 18.19           0.000474
##   Layer3_MinPressure Layer3_AvgPressure Layer3_StdPressure Layer3_MaxPower
## 1           0.000270       0.0003028000       1.384470e-05             5.0
## 2           0.000295       0.0003169286       1.403125e-05             5.0
## 3           0.000899       0.0009181765       1.133259e-05             5.0
## 4           0.000889       0.0009052857       1.189295e-05             5.0
## 5           0.000940       0.0010017273       3.609764e-05             5.8
## 6           0.000366       0.0004442247       2.329091e-05             5.6
##   Layer3_MinPower Layer3_AvgPower Layer3_StdPower Layer3_MaxRate
## 1             3.8        4.192632       0.2715422           0.60
## 2             3.1        3.655952       0.4302135           0.64
## 3             3.8        4.207059       0.3787870           0.49
## 4             3.3        3.563095       0.3330164           0.67
## 5             4.8        5.261364       0.3517592           0.48
## 6             4.8        5.100000       0.2848220           0.49
##   Layer3_MinRate Layer3_AvgRate Layer3_StdRate Layer3_MaxO2Flow
## 1           0.00      0.4316842     0.07113594                0
## 2           0.00      0.4427381     0.08977951                0
## 3           0.07      0.4428235     0.04769047                0
## 4           0.06      0.4517857     0.07570558                0
## 5           0.00      0.4230682     0.07031355                0
## 6           0.00      0.4228090     0.06091048               0
##   Layer3_MinO2Flow Layer3_AvgO2Flow Layer3_StdO2Flow Layer3_FinalThickness
## 1                0                0                0                 41.82
## 2                0                0                0                 38.13
## 3                0                0                0                 38.03
## 4                0                0                0                 38.19
## 5                0                0                0                 38.07
## 6                0                0                0                 38.02
##   Layer4_MaxPressure Layer4_MinPressure Layer4_AvgPressure
## 1           0.000695           0.000279       0.0004106931
## 2           0.002740           0.002070       0.0022287589
## 3           0.003030           0.002590       0.0026941993
## 4           0.003500           0.002560       0.0026926335
## 5           0.003280           0.002450       0.0026892883
## 6           0.002620           0.002070       0.0024077224
##   Layer4_StdPressure Layer4_MaxPower Layer4_MinPower Layer4_AvgPower
## 1       3.744499e-05            22.5              21        21.24521
## 2       6.886274e-05            25.6              21        25.20035
## 3       6.230578e-05            24.4              21        24.25231
## 4       8.888097e-05            24.4              21        24.11566
## 5       9.153996e-05            24.7              21        24.37153
## 6       1.710377e-04            24.8              21        24.55338
##   Layer4_StdPower Layer4_MaxRate Layer4_MinRate Layer4_AvgRate
## 1       0.2432628           0.25           0.09      0.2107591
## 2       0.5287814           0.21           0.00      0.2039716
## 3       0.4840554           0.21           0.00      0.2048399
## 4       0.4494010           0.22           0.00      0.2048043
## 5       0.4849047           0.22           0.00      0.2044484
## 6       0.4633903           0.21           0.00      0.2049110
##   Layer4_StdRate Layer4_MaxO2Flow Layer4_MinO2Flow Layer4_AvgO2Flow
## 1    0.008773914              4.3              4.2          4.20099
## 2    0.024563694             10.8             10.7         10.70922
## 3    0.021327625             10.7             10.6         10.64982
## 4    0.022242135             10.7             10.6         10.63986
## 5    0.022836219             10.7             10.6         10.63737
## 6    0.020735821             10.7             10.6         10.63772
##   Layer4_StdO2Flow Layer4_FinalThickness Layer5_MaxPressure
## 1       0.00990099                 64.01           0.000228
## 2       0.02893061                 57.69           0.000231
## 3       0.04999968                 57.73           0.000419
## 4       0.04896052                 57.74           0.000374
## 5       0.04837764                 57.69           0.000433
## 6       0.04846918                 57.80           0.000342
##   Layer5_MinPressure Layer5_AvgPressure Layer5_StdPressure Layer5_MaxPower
## 1           0.000197       0.0002188077       6.531104e-06             3.9
## 2           0.000213       0.0002267143       3.989783e-06             4.5
## 3           0.000378       0.0003993846       1.062077e-05             4.8
## 4           0.000350       0.0003636000       4.454211e-06             4.5
## 5           0.000366       0.0004167826       2.042900e-05             5.8
## 6           0.000282       0.0003253778       1.647798e-05             4.8
##   Layer5_MinPower Layer5_AvgPower Layer5_StdPower Layer5_MaxRate
## 1             3.5        3.730769       0.1538462           0.22
## 2             2.0        2.857143       0.8351341           0.47
## 3             3.2        4.169231       0.5720650           0.31
## 4            2.1        2.930000       0.8128756           0.42
## 5            4.1        5.121739       0.5408982           0.32
## 6            4.1        4.586667       0.2156128           0.25
##   Layer5_MinRate Layer5_AvgRate Layer5_StdRate Layer5_MaxO2Flow
## 1           0.00      0.1903846     0.04887038                0
## 2           0.00      0.3164286     0.10560081                0
## 3           0.01      0.2369231     0.07463256                0
## 4           0.00      0.3026667     0.09674480                0
## 5           0.00      0.2004348     0.10992006                0
## 6           0.00      0.2015556     0.05046255                0
##   Layer5_MinO2Flow Layer5_AvgO2Flow Layer5_StdO2Flow Layer5_FinalThickness
## 1                0                0                0                 10.33
## 2                0                0                0                  9.25
## 3                0                0                0                  9.47
## 4                0                0                0                  9.27
## 5                0                0                0                  9.47
## 6                0                0                0                  9.30
##   Layer6_MaxPressure Layer6_MinPressure Layer6_AvgPressure
## 1            0.00058           0.000189       0.0003064246
## 2            0.00265           0.001950       0.0021046835
## 3            0.00283           0.002210       0.0023062979
## 4            0.00268           0.002240       0.0022995763
## 5            0.00265           0.002090       0.0022590678
## 6            0.00259           0.002310       0.0023853814
##   Layer6_StdPressure Layer6_MaxPower Layer6_MinPower Layer6_AvgPower
## 1       3.398176e-05            22.6            20.7        20.89365
## 2       7.721692e-05            24.9            21.0        24.61350
## 3       6.176411e-05            24.1            21.0        23.90128
## 4       5.616100e-05            24.2            21.0        23.92203
## 5       6.727111e-05            24.5            21.0        24.20254
## 6       3.287532e-05            24.4            21.0        24.13729
##   Layer6_StdPower Layer6_MaxRate Layer6_MinRate Layer6_AvgRate
## 1       0.2763843           0.27           0.04      0.2111111
## 2       0.5463990           0.21           0.00      0.2029114
## 3       0.3839859           0.21           0.00      0.2048085
## 4       0.4410380           0.22           0.00      0.2038559
## 5       0.4984658           0.21           0.00      0.2035169
## 6       0.4758216           0.22          -0.01      0.2034746
##   Layer6_StdRate Layer6_MaxO2Flow Layer6_MinO2Flow Layer6_AvgO2Flow
## 1     0.01352377              4.2              4.2          4.20000
## 2     0.02705606             10.8             10.7         10.70253
## 3     0.02128493             10.7             10.6         10.64000
## 4     0.02451142             10.7             10.6         10.63305
## 5     0.02437185             10.7             10.6         10.63263
## 6     0.02535768             10.7             10.6         10.64025
##   Layer6_StdO2Flow Layer6_FinalThickness Layer7_MaxPressure
## 1       0.00000000                 53.56           0.000213
## 2       0.01570845                 48.40           0.000244
## 3       0.04898979                 48.38           0.000338
## 4       0.04703962                 48.33           0.000338
## 5       0.04688478                 48.30           0.000378
## 6       0.04904100                 48.34           0.000316
##   Layer7_MinPressure Layer7_AvgPressure Layer7_StdPressure Layer7_MaxPower
## 1           0.000182       0.0002056889       4.879081e-06             5.2
## 2           0.000180       0.0002272059       1.216620e-05             5.0
## 3           0.000327       0.0003328137       2.861956e-06             5.2
## 4           0.000292       0.0003292353       7.249985e-06             5.0
## 5           0.000320       0.0003335415       1.213077e-05             6.5
## 6           0.000244       0.0002886117       1.029774e-05             6.2
##   Layer7_MinPower Layer7_AvgPower Layer7_StdPower Layer7_MaxRate
## 1             3.8        4.146222       0.3696202           0.48
## 2             3.4        3.730392       0.2692725           0.63
## 3             3.7        3.931373       0.3851357           0.51
## 4             3.3        3.659314       0.2537204           0.66
## 5             4.0        4.419512       0.6761511           0.57
## 6             4.2        4.623786       0.5756948           0.52
##   Layer7_MinRate Layer7_AvgRate Layer7_StdRate Layer7_MaxO2Flow
## 1           0.09      0.4347111     0.02845708                0
## 2           0.04      0.4349020     0.04704493                0
## 3           0.09      0.4358824     0.03508681                0
## 4           0.00      0.4332843     0.05983226                0
## 5           0.00      0.4306829     0.07629546                0
## 6           0.00      0.4308252     0.05877222                0
##   Layer7_MinO2Flow Layer7_AvgO2Flow Layer7_StdO2Flow Layer7_FinalThickness
## 1                0                0                0                 98.34
## 2                0                0                0                 89.08
## 3                0                0                0                 88.84
## 4                0                0                0                 88.78
## 5                0                0                0                 88.80
## 6                0                0                0                 89.14
##        rAvg numPeaks      p1w       p1r     p1rw      p2w       p2r
## 1 0.1174496        2 453.8182 0.1443307 65.49990 554.0978 0.1557683
## 2 0.1500958        2 449.2421 0.1654892 74.34470 565.2919 0.2006995
## 3 0.1271214        2 451.1611 0.1016201 45.84703 563.3729 0.1863977
## 4 0.1273762        2 451.1365 0.1420721 64.09392 559.3627 0.1752241
## 5 0.1480437        2 458.8617 0.1159608 53.20997 567.8506 0.2241602
## 6 0.1150107        2 455.3435 0.1030191 46.90909 560.8143 0.1806068
##        p2rw                                 spectraFile
## 1  86.31088  ./data/spectra/DLF/150211-6 dlf.Sample.asc
## 2 113.45380  ./data/spectra/DLF/151119-2 DLF.Sample.asc
## 3 105.01141  ./data/spectra/DLF/151121-2 DLF.Sample.asc
## 4  98.01385  ./data/spectra/DLF/151122-6 DLF.Sample.asc
## 5 127.28952 ./data/spectra/DLF/151123-14 DLF.Sample.asc
## 6 101.28688  ./data/spectra/DLF/151202-3 DLF.Sample.asc
##                                 runFile
## 1  ./data/machine/DLF/150211-6 DLF.RUN
## 2  ./data/machine/DLF/151119-2 DLF.RUN
## 3  ./data/machine/DLF/151121-2 DLF.RUN
## 4  ./data/machine/DLF/151122-6 DLF.RUN
## 5 ./data/machine/DLF/151123-14 DLF.RUN
## 6  ./data/machine/DLF/151202-3 DLF.RUN
```

Now this isn't the ideal dimensions for a data frame. It should have more observables than column names. In this regard I was limited by the method that I had to use to capture reliable data. I also took lots of liberty creating predictor and response variables for all kinds of things that didn't really have any basis in the physics of optical coatings. I suspect that the vast number of my variables have no relation to most of the responses and even some of the curve fitting responses are probably meaningless. However I still will try to follow through with an analysis to see if I can tease out any conclusions from what I have to work with.

The dimension problem, as I will call it, is something I will have to be mindful of as I try to perform multiple regression fits of my data.

Since I have more predictors than data points I am not in a good situation to run a multiple regression with an F-static since p > n. However I plan to only use a few of the predictors at a time in the multiple regression. I will have to use the forward selection sub-setting technique to try to narrow down which variables are important and get to a model that uses only those variables. This is the direction that I will take in my exploratory analysis of the data.

2 The Analysis

As I alluded to I hope to use the forward step-wise subset selection technique to determine which variables I should use in my model. This is how I will proceed. The leaps package will be used for the forward sub-setting and model evaluation process. The general technique is the technique that is laid in ISLR 6.5 Lab 1: Subset Selection Methods. Much of the code used here was adopted from that R lab.

2.1 Modeling Reflection Average with RSQ

2.1.1 TLF

To start off I am looking broadly at the average reflection for an entire spectra. This is the spec that the customers that buy this coating actually care about so I decided to try to go for this one first to see if anything nice comes out. To crank out a multi-regression I put together the following code using the regsubsets function of the leap package:

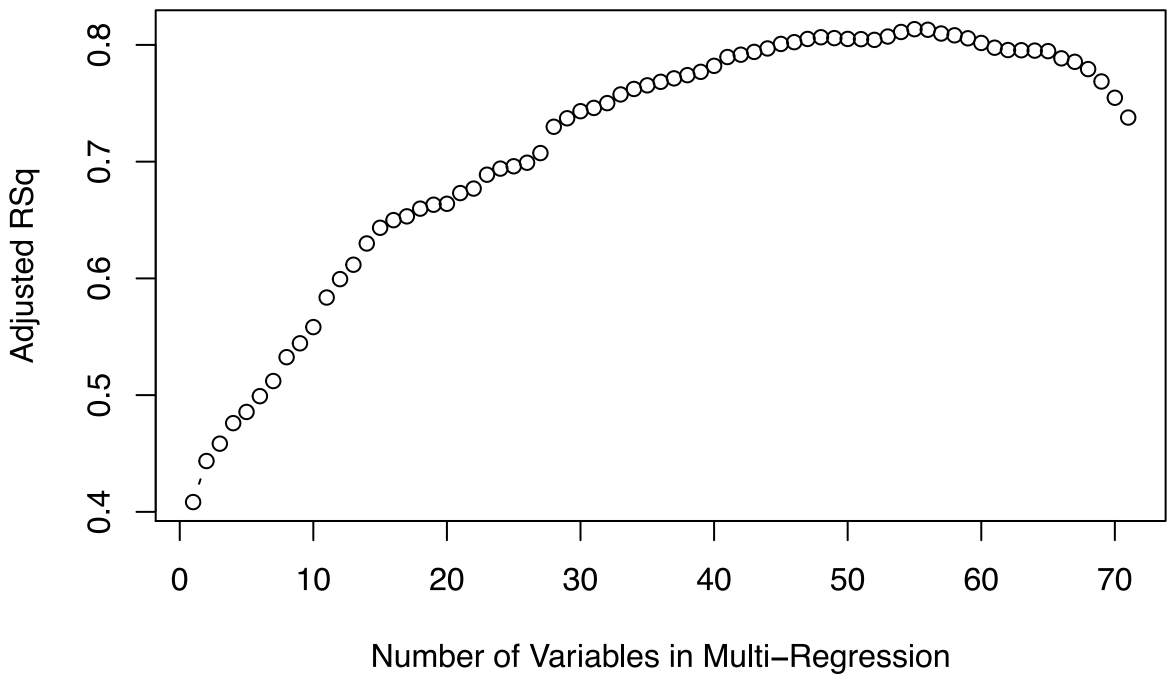
```r
library(leaps)
predictors.tlf <- TLF[4:122]  #All of the machine variables without the rest of the info
lm.tlf <- predictors.tlf
lm.tlf$rAvg <- na.omit(TLF$rAvg)  #The variable to use as the response #ravg
# cols <- colnames(lm.tlf)
fwdfit.tlf <- regsubsets(rAvg ~ ., data = lm.tlf, nvmax = 70, method = "forward")
```

```
## Reordering variables and trying again:
```

```r
# summary(fwdfit.tlf) #Nasty Console Output for RMD
tlf.summary <- summary(fwdfit.tlf)
which.max(tlf.summary$adjr2)
```

```
## [1] 55
```

```r
plot(tlf.summary$adjr2, xlab = "Number of Variables in Multi-Regression", ylab = "Adjusted RSq",
    type = "b")
```



This is a bit troubling. Especially since it takes a model with 55 variables to produce a maximum in the adjusted RSQ statistic for this multi-regression model for rAvg. This seems to me to be evidence of over-fitting the small amount of data that I have. Another troubling thing is that with a 55 model variable I wouldn't be able to nicely split up my data into testing data and training data for a cross-validation. While I am limited in the amount of data that I have I will still try to do a cross-validation approach, rather than maximizing the RSQ, to try to choose a model. I am hoping for a model that might provide more interchangeability than the 63 variable monster and not just be a one variable intercept guess model.

```
library(leaps)
set.seed(1)
train <- sample(c(TRUE, FALSE), nrow(lm.tlf), rep = TRUE)
test <- (!train)

regfit.fwd.tlf <- regsubsets(rAvg ~ ., data = lm.tlf[train, ], nvmax = 40, method = "forward")  #model
```

```
## Reordering variables and trying again:
```

```
# coef(regfit.fwd.tlf,40) #Ugly Console Output
test.mat.tlf <- model.matrix(rAvg ~ ., data = lm.tlf[test, ])  #Predict Test Data
val.errors.tlf <- rep(NA, 40)
for (i in 1:40) {
    coefi <- coef(regfit.fwd.tlf, id = i)
    pred <- test.mat.tlf[, names(coefi)] %*% coefi
    val.errors.tlf[i] <- mean((lm.tlf$rAvg[test] - pred)^2)
}

which.min(val.errors.tlf)
```

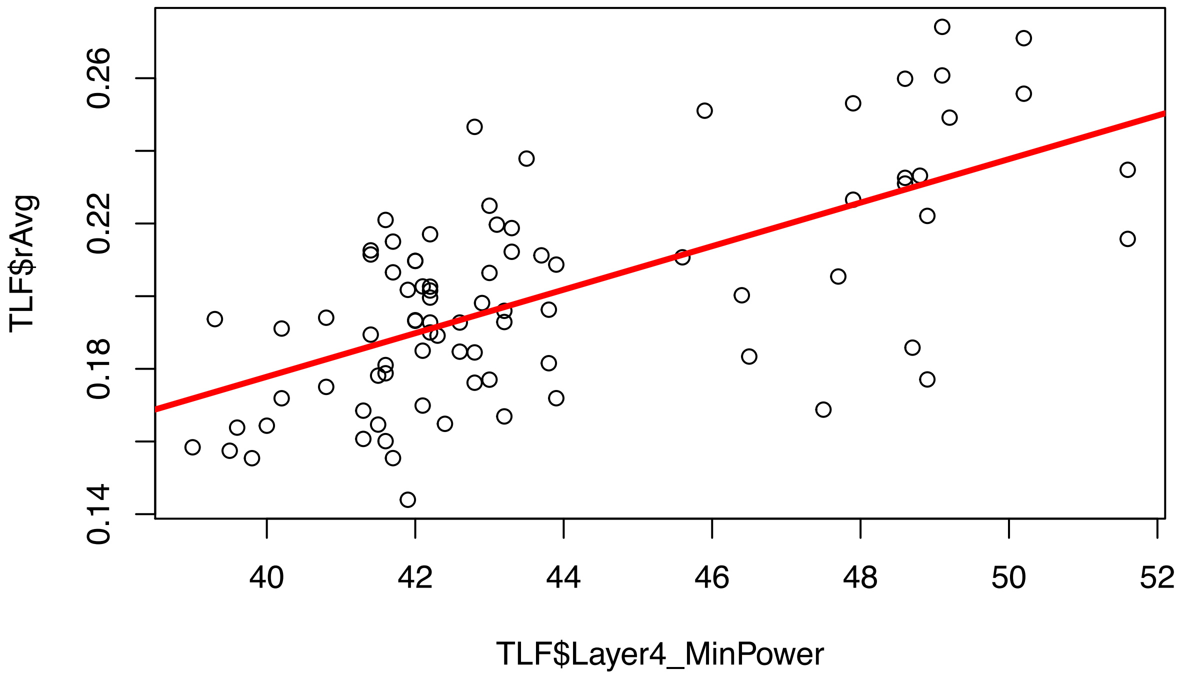
```
## [1] 1
```

```
# val.errors.tlf
coef(regfit.fwd.tlf, which.min(val.errors.tlf))
```

```
##     (Intercept) Layer4_MinPower
##     -0.07683728      0.00644544
```

This attempt at cross validation confirms my fears that the Adjusted RSQ method of evaluating a model was severely over-fitting. This attempt at cross-validation shows that the best model for the testing data takes only an intercept and the minimum power of the fourth layer, which is a high index layer. I don't think that this model is very useful. I can produce a scatter plot of the average reflectance and the minimum power in layer 4 from just a simple linear model without the multiple regression.

```r
lm.layer4MinPower_rAvg.tlf <- lm(rAvg ~ Layer4_MinPower, data = TLF)
plot(TLF$Layer4_MinPower, TLF$rAvg)
abline(lm.layer4MinPower_rAvg.tlf, lwd = 3, col = "red")
```



```r
summary(lm.layer4MinPower_rAvg.tlf)
```

```
##
## Call:
## lm(formula = rAvg ~ Layer4_MinPower, data = TLF)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.054066 -0.016218  0.000371  0.016351  0.052012
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -0.0618405  0.0344565  -1.795   0.0764 .
## Layer4_MinPower  0.0059915  0.0007863   7.620 3.94e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02263 on 82 degrees of freedom
## Multiple R-squared:  0.4145, Adjusted R-squared:  0.4074
## F-statistic: 58.06 on 1 and 82 DF,  p-value: 3.94e-11
```

So I see that there appears to be some relationship here with the one variable. Perhaps the minimum power in layer 4 is somewhat related to the average reflection. However I realize that characterizing the whole reflection curve by an average value seems to lose a lot of information and potentially gloss over some physical mechanism as to how different materials behave at different wavelengths. For example here a rise in the average reflection really only suggests that there was some problem with the deposition. An increase in the minimum power in layer 4 might be loosely related to this problem but this doesn't really inform how the settings would be changed to produce more reliable curves. As a side note this scatter plot also appears to segregate into two clusters. I am wondering if this could have stemmed from the rate set point being changed during my data collection and it is something that I should try to track down.

2.1.2 DLF

Now I am going to repeat this procedure for the data I have for the DLF.

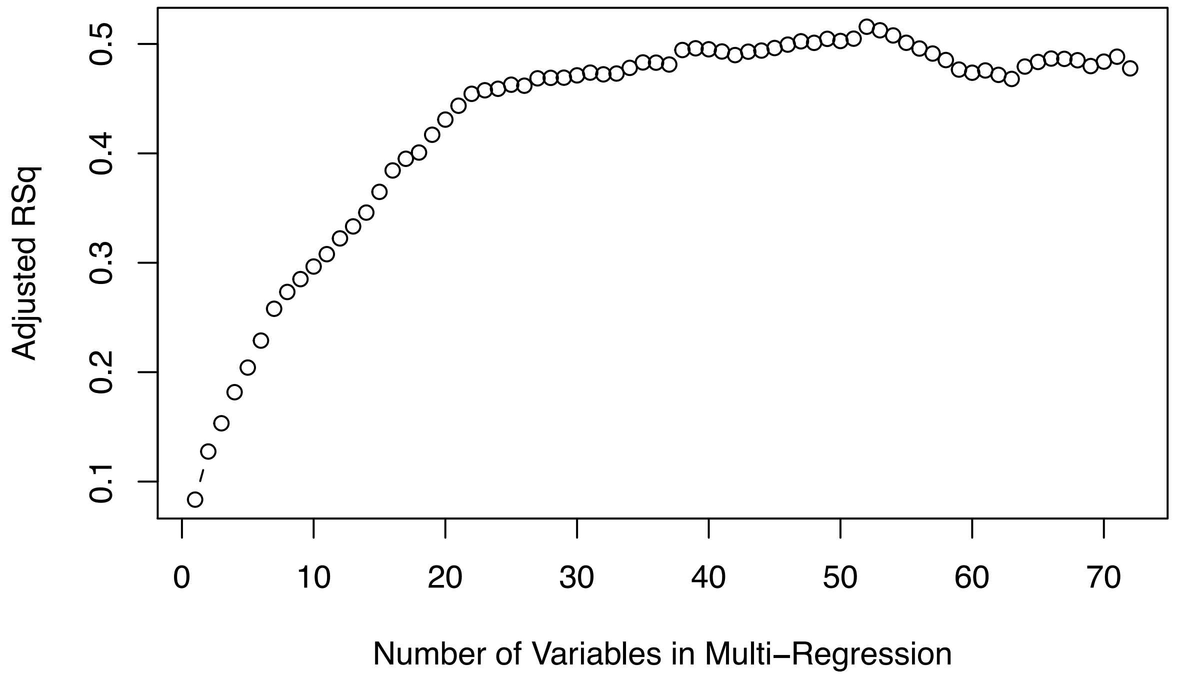
```
library(leaps)
predictors.dlf <- noOutlierDLF[4:122]  #All of the machine variables without the rest of the info
lm.dlf <- predictors.dlf
lm.dlf$rAvg <- na.omit(noOutlierDLF$rAvg)  #The variable to use as the response #ravg
# cols <- colnames(lm.tlf)
fwdfit.dlf <- regsubsets(rAvg ~ ., data = lm.dlf, nvmax = 70, method = "forward")
```

```
## Reordering variables and trying again:
```

```
dlf.summary <- summary(fwdfit.dlf)
which.max(dlf.summary$adjr2)
```

```
## [1] 52
```

```
plot(dlf.summary$adjr2, xlab = "Number of Variables in Multi-Regression", ylab = "Adjusted RSq",
    type = "b")
```



```
# Basic Cross Validation for DLF to avoid overfitting
set.seed(1)
train <- sample(c(TRUE, FALSE), nrow(lm.dlf), rep = TRUE)
test <- (!train)

regfit.fwd.dlf <- regsubsets(rAvg ~ ., data = lm.dlf[train, ], nvmax = 40, method = "forward")  #model
```

```
## Reordering variables and trying again:
```

```
test.mat.dlf <- model.matrix(rAvg ~ ., data = lm.dlf[test, ])  #Predict Test Data
val.errors.dlf <- rep(NA, 40)
for (i in 1:40) {
    coefi <- coef(regfit.fwd.dlf, id = i)
    pred <- test.mat.dlf[, names(coefi)] %*% coefi
    val.errors.dlf[i] <- mean((lm.dlf$rAvg[test] - pred)^2)
}

which.min(val.errors.dlf)
```

```
## [1] 2
```

```
# val.errors.dlf
coef(regfit.fwd.dlf, which.min(val.errors.dlf))
```

```
##           (Intercept) Layer4_FinalThickness    Layer7_StdPressure
##           0.298251120          -0.002898858         499.138491159
```

So the DLF picked a 9 variable model with cross validation. This is interesting but I suspect that since I have a small data set it is still very much dependent on the random seeding. So I also am wary of just using the average so I will try to investigate the best response to model.

2.1.3 Finding The Most "Modelable"" Response (MMR)

So I suspect that the average across an entire spectrum is a bad response because it isn't very sensitive to any one predictor and is kind of a general indicator of a problem. This is why it is useful to use for customers whey buying anti-reflective coatings for general use through the visible range. However I think that it might not be the best response variable to use here. What I want to do is to find out if any of my curve modeling parameters will be more "modelable"" in a way that might suggest some more nuance to the relationship. To try to do this without over-fitting I am going to run through the cross validation framework for every response variable. I want to try to find out if any of the responses can be modeled with a model that isn't just a single variable regression. Here is the code that I have developed to do this tagging of the useful response variables. It finds the model with the least error and records how many variables that model had for each response variable. Then I change the random seeding that sets the training data and re-run the analysis.

```
getFormula <- function(target, df) {
    # This function allows the response~. notation to be programatically added
    # into regsubsets function
    i <- grep(target, colnames(df))
    as.formula(paste(colnames(df)[i], ".", sep = " ~ "))
}

bestResponse <- function(machineData, numVars, printResult, method) {
    # I want to take the list of response variables and find the one that has
    # the best model for the test error, not the training error

    # Setup
    seeds <- seq(1, 10, 1)
    predictors <- seq(4, 122, 1)
    responses <- seq(123, 130, 1)
    finalCols <- colnames(machineData[responses])
    best.model <- vector("list", length(responses))  #This list will hold the number of vars in best mo
    avg.model <- NULL
```

```r
    # Looping over random seeds, response variables, and models for each
    # response variable loop of different random seeds
    for (k in seeds) {
        set.seed(seeds[k])
        train <- sample(c(TRUE, FALSE), nrow(machineData), rep = TRUE)
        test <- (!train)
        for (i in responses) {
            # Loop over different response variables
            lm.machine <- machineData[c(4:122, i)]
            cols <- colnames(lm.machine)
            lm.machine <- na.omit(lm.machine)
            log <- capture.output({
                # Hides the Console Vomit!!!
                fwdfit.machine <- regsubsets(as.formula(paste(cols[120], ".",
                  sep = "~")), data = lm.machine[train, ], nvmax = numVars,
                  method = method)  #Make Model with train data for Variable
            })
            test.mat.machine <- model.matrix(getFormula(cols[120], lm.machine[test,
                ]), data = lm.machine[test, ])  #Predict Test Data with model
            val.errors.machine <- rep(NA, numVars)
            test.machine <- lm.machine[, 120]
            for (j in 1:numVars) {
                # Loop over the models produced with different numbers of variables and run
                # cross-validation
                coefj <- coef(fwdfit.machine, id = j)
                pred <- test.mat.machine[, names(coefj)] %*% coefj  #Makes prediction with the model
                val.errors.machine[j] <- mean((test.machine[test] - pred)^2)  #Calculates model errors
            }
            best.model[[which(responses == i)]][k] <- which.min(val.errors.machine)  #Store the number
        }
        for (n in 1:length(best.model)) {
            avg.model[n] <- mean(unlist(best.model[[n]]))  #Average the number of variables of the best
        }
    }
    if (printResult) {
        # Print nice result if we desire/set the parameter to TRUE
        print(rbind(finalCols, avg.model))
    }
    return.index <- responses[which.max(avg.model)]  #Find the index of the average model that had the
    cols <- colnames(machineData)  #Find the name of the response variable that the model was for
    return(c(cols[return.index], round(max(avg.model))))  #return Information
    # return(ceiling(max(avg.model)))
}

# I also should randomize the number of variables
varNums <- seq(1, 20, 1)
ideal.variable.number <- NULL
response.variable <- NULL
for (i in varNums) {
    model.response <- bestResponse(noOutlierDLF, varNums[i], FALSE, "forward")  #Running the procedure

    # Printing the output
    ideal.variable.number[i] <- as.numeric(model.response[2])  #Getting the number of varibles on avg i
    response.variable[i] <- as.character(model.response[1])  #Getting the response variable associated
    if (i == 20) {
        print.out <- rbind(response.variable, ideal.variable.number)  #So we can see how it grows throu
        print(print.out)
        cat("\n")
    }
}
```

```
##                       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
## response.variable     "rAvg" "p1rw" "p1rw" "rAvg" "p1w"  "rAvg" "rAvg"
## ideal.variable.number "1"    "2"    "2"    "2"    "2"    "2"    "2"
##                       [,8]  [,9]  [,10] [,11] [,12] [,13] [,14] [,15]
## response.variable     "p1w" "p1w" "p1w" "p1w" "p1w" "p1w" "p1w" "p1w"
## ideal.variable.number "2"   "2"   "4"   "4"   "4"   "4"   "4"   "5"
##                       [,16] [,17] [,18] [,19] [,20]
## response.variable     "p1w" "p1w" "p1w" "p1w" "p1w"
## ideal.variable.number "5"   "5"   "5"   "5"   "5"
```

```
# mean(ideal.variable.number)
# bestResponse(DLF,ceiling(mean(ideal.variable.number))+3,TRUE,'forward')
```

Here I was looking for responses that could be predicted by more than just one response. This for loop that is printing out is iterating through the number of variables that the forward step-wise subset selection multi regression model was able to use. The number of variables allowed is the number shown in brackets the number under the variable is the number of variable that the best model in the subset of models used. The function that is called to do most of the work here models a multi-regression on randomized test and training data. Then the best model is chosen and the number of variables the model has is selected. I then change the randomized training and testing data seed 10 times and re-do the whole thing. Then I take the average number of variables that the best model contained for each of the response variables. Here I print out the responses that on average had the best model of the test data with the most variables. It is interesting to see how this grows with the number of variables allowed to the function. This is interesting, but to me looks like it could be evidence of over-fitting. Also in the higher variable end of this half pyramid it seems to trend to Ravg as the parameter that is best fit with more variables. I suspect that the average is a type of catch all but it is interesting. I might pull out some of these higher variable models to look at how they look on the data.

```
varNums <- seq(1, 20, 1)
ideal.variable.number.tlf <- NULL
response.variable.tlf <- NULL
for (i in varNums) {
    model.response <- bestResponse(TLF, varNums[i], FALSE, "forward")  #Running the procedure with diff

    # Printing the output
    ideal.variable.number.tlf[i] <- as.numeric(model.response[2])  #Getting the number of varibles on a
    response.variable.tlf[i] <- as.character(model.response[1])  #Getting the response variable associa
    if (i == 20) {
        print.out <- rbind(response.variable.tlf, ideal.variable.number.tlf)  #So we can see how it gro
        print(print.out)
        cat("\n")
    }
}
```

```
##                           [,1]   [,2]  [,3]   [,4]   [,5]   [,6]   [,7]
## response.variable.tlf     "rAvg" "p1w" "p1rw" "p1rw" "p1rw" "p1rw" "p1rw"
## ideal.variable.number.tlf "1"    "1"   "2"    "2"    "2"    "2"    "2"
##                              [,8]   [,9]   [,10]  [,11] [,12] [,13] [,14]
## response.variable.tlf       "p1rw" "p1rw" "p1rw" "p1r" "p1r" "p1r" "rAvg"
## ideal.variable.number.tlf "2"    "2"    "2"    "2"   "2"   "2"   "2"
##                              [,15]  [,16]  [,17]  [,18]  [,19]  [,20]
## response.variable.tlf       "rAvg" "rAvg" "rAvg" "rAvg" "rAvg" "rAvg"
## ideal.variable.number.tlf "2"    "2"    "3"    "3"    "3"    "3"
```

Now again we see the results. The variables and variable numbers of the best model on average for the data set.

2.2 Modeling for Specfic Spectral Features

For now I am going to look at some of the lower number of variable models to try to have a chance at interpreting the results. It appears that the wavelength of the first peak (p1w) seems to be the favored response for many of the lower variable models. This is the wavelength at which the first peak occurs. It is incredibly important for the color of the reflection of an anti-reflection coating so it is a worthwhile thing to try to predict. Based on my investigation into the possibility of modeling all of the variables I am going to look specifically at the first peak for DLF spectra. For the TLF spectra I will look at the product of the first peak reflection and wavelength. This is the variable that comes up many times in selecting the best model and will allow for two different response variables to be analyzed for each machine.

2.2.1 Modeling for the DLF second Peak Wavelength (p1w)

I will now model specifically for the wavelength of the first peak in the DLF spectra. I want to find what coefficients the model is choosing and then do a multi-regression with just those coefficients on the data. I will try the same procedure I have been using allowing 15, 10 and 5 variables. Then I will examine the models.

```r
library(leaps)
predictors.dlf <- noOutlierDLF[4:122]  #All of the machine variables without the rest of the info
lm.dlf <- predictors.dlf
lm.dlf$p1w <- na.omit(noOutlierDLF$p1w)  #The variable to use as the response #ravg

# Basic Cross Validation for DLF to avoid overfitting
set.seed(8)
train <- sample(c(TRUE, FALSE), nrow(lm.dlf), rep = TRUE)
test <- (!train)

regfit.fwd.dlf <- regsubsets(p1w ~ ., data = lm.dlf[train, ], nvmax = 20, method = "forward")  #model
```

```
## Reordering variables and trying again:
```

```r
test.mat.dlf <- model.matrix(p1w ~ ., data = lm.dlf[test, ])  #Predict Test Data
val.errors.dlf <- rep(NA, 20)
for (i in 1:20) {
    coefi <- coef(regfit.fwd.dlf, id = i)
    pred <- test.mat.dlf[, names(coefi)] %*% coefi
    val.errors.dlf[i] <- mean((lm.dlf$p1w[test] - pred)^2)
}

which.min(val.errors.dlf)
```

```
## [1] 4
```

```r
# val.errors.dlf
coef(regfit.fwd.dlf, which.min(val.errors.dlf))
```

```
##        (Intercept)    Layer3_MaxPower     Layer5_MinRate
##        449.1976105         -0.6058003        -58.1546077
## Layer6_MaxPressure     Layer6_MinRate
##       2444.4049999        116.8106887
```

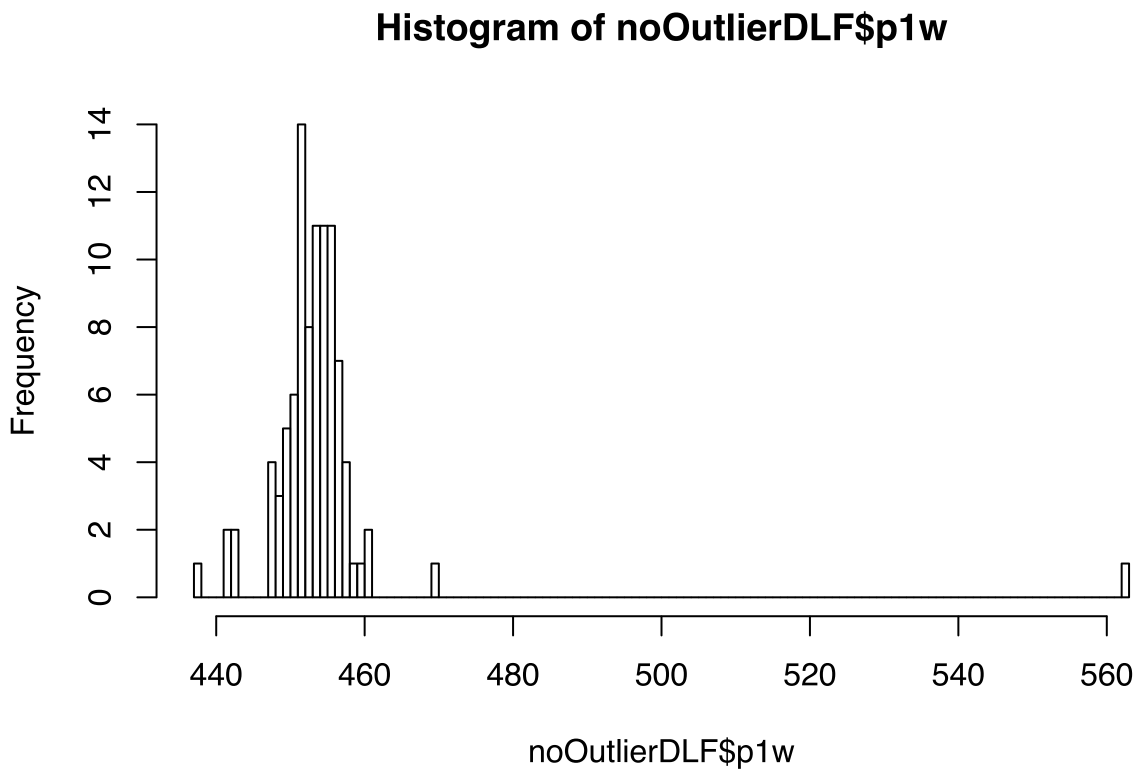
The predictors that this process select again and again are the minimum rate in layer 6. By varying the seed other variables are selected but it isn't consistent for all the data. I will try to look this linear model as well as information for all of the other predictors together individually.

```r
library(MASS)
lm.fit.dlf.p1w <- lm(p1w ~ ., data = lm.dlf)
summary.dlf.p1w <- summary(lm.fit.dlf.p1w)
summary.dlf.p1w$coefficients
```

```
##                             Estimate   Std. Error     t value  Pr(>|t|)
## (Intercept)            -7.217151e+03 7.688770e+03 -0.93866135 0.5201357
## Layer1_MaxPressure     -6.479128e+05 2.886819e+05 -2.24438332 0.2668409
## Layer1_MinPressure      7.670789e+05 3.208512e+05  2.39076249 0.2522046
## Layer1_AvgPressure     -1.244124e+05 8.844711e+04 -1.40663041 0.3934415
## Layer1_StdPressure      3.140700e+06 1.111600e+06  2.82538763 0.2165621
## Layer1_MaxPower        -4.582624e+01 2.715554e+01 -1.68754655 0.3405553
## Layer1_MinPower         5.815671e+03 2.399797e+03  2.42340124 0.2491462
## Layer1_AvgPower        -1.866797e+00 1.313353e+01 -0.14213975 0.9101131
## Layer1_StdPower         2.170282e+02 6.040796e+01  3.59270949 0.1728232
## Layer1_MaxRate          6.144987e+02 3.109038e+02  1.97649151 0.2981888
## Layer1_MinRate         -1.566941e+03 5.032717e+02 -3.11350805 0.1978449
## Layer1_AvgRate          2.500820e+03 5.641648e+02  4.43278273 0.1412518
## Layer1_StdRate         -2.494702e+03 1.181760e+03 -2.11100535 0.2816365
## Layer1_FinalThickness   3.369879e+01 3.357820e+01  1.00359121 0.4988589
## Layer2_MaxPressure     -1.549636e+05 5.780029e+04 -2.68101682 0.2272795
## Layer2_MinPressure     -9.418558e+04 3.431588e+04 -2.74466427 0.2224319
## Layer2_AvgPressure      1.896058e+05 7.397507e+04  2.56310347 0.2368149
## Layer2_StdPressure     -3.231068e+05 1.361365e+05 -2.37340405 0.2538604
## Layer2_MaxPower         1.557544e+02 4.708205e+01  3.30814888 0.1868802
## Layer2_AvgPower        -1.750926e+02 4.115365e+01 -4.25460598 0.1469632
## Layer2_StdPower         1.517902e+02 6.149125e+01  2.46848403 0.2450357
## Layer2_MaxRate         -2.861141e+03 1.893390e+03 -1.51112097 0.3721668
## Layer2_MinRate          6.652836e+02 7.643598e+02  0.87038017 0.5440482
## Layer2_AvgRate          1.989236e+04 6.701282e+03  2.96844088 0.2068611
## Layer2_StdRate          1.936008e+04 5.094768e+03  3.79999333 0.1638176
## Layer2_MaxO2Flow       -1.867364e+03 5.184963e+02 -3.60149908 0.1724218
## Layer2_AvgO2Flow       -9.587750e+02 3.602859e+02 -2.66115066 0.2288342
## Layer2_StdO2Flow       -1.050732e+03 1.759093e+03 -0.59731439 0.6572169
## Layer2_FinalThickness   2.203864e+01 7.534064e+01  0.29251992 0.8188306
## Layer3_MaxPressure      1.345603e+06 5.902089e+05  2.27987504 0.2631471
## Layer3_MinPressure     -1.103701e+05 1.681074e+05 -0.65654522 0.6301477
## Layer3_AvgPressure     -1.080842e+06 6.993241e+05 -1.54555283 0.3655953
## Layer3_StdPressure     -6.426854e+05 6.642825e+05 -0.96748814 0.5105189
## Layer3_MaxPower        -1.561233e+02 3.770392e+01 -4.14076945 0.1508558
## Layer3_MinPower        -5.769850e+01 8.311879e+01 -0.69416924 0.6136979
## Layer3_AvgPower        -6.064502e+00 4.119998e+01 -0.14719673 0.9069598
## Layer3_StdPower         1.316245e+02 9.742866e+01  1.35098352 0.4056544
## Layer3_MaxRate         -1.207857e+03 6.633129e+02 -1.82094578 0.3197117
## Layer3_MinRate          8.186544e+02 2.773364e+02  2.95184580 0.2079433
## Layer3_AvgRate          3.599357e+03 3.197470e+03  1.12568883 0.4624014
## Layer3_StdRate         3.545787e+03 1.386288e+03  2.55775616 0.2372655
## Layer3_FinalThickness  5.828003e+01 3.468041e+01  1.68048867 0.3417266
## Layer4_MaxPressure    -3.008235e+04 1.286153e+04 -2.33894013 0.2572094
## Layer4_MinPressure    -4.570928e+04 6.526784e+04 -0.70033393 0.6110571
## Layer4_AvgPressure     1.318200e+05 7.134959e+04  1.84752246 0.3158350
## Layer4_StdPressure     6.965079e+05 1.505416e+05  4.62668063 0.1355130
## Layer4_MaxPower       -1.629698e+02 6.854810e+01 -2.37745148 0.2534725
## Layer4_AvgPower        2.196439e+02 8.944138e+01  2.45573031 0.2461854
## Layer4_StdPower       -3.775540e+02 1.108073e+02 -3.40730410 0.1817365
## Layer4_MaxRate        -1.805250e+03 2.350897e+03 -0.76789811 0.5831045
## Layer4_MinRate         1.211551e+03 6.640501e+02  1.82448744 0.3191901
## Layer4_AvgRate         2.929226e+04 3.375951e+04  0.86767447 0.5450295
## Layer4_StdRate         1.939451e+04 7.572347e+03  2.56122804 0.2369728
## Layer4_AvgO2Flow       3.052067e+03 4.189462e+03  0.72851053 0.5991808
## Layer4_StdO2Flow       7.404167e+03 3.038384e+03  2.43687629 0.2479039
## Layer4_FinalThickness -1.637385e+02 8.354731e+01 -1.95982976 0.3003653
## Layer5_MaxPressure     8.081503e+05 5.058741e+05  1.59753267 0.3560571
## Layer5_MinPressure     6.673340e+05 2.951206e+05  2.26122443 0.2650761
## Layer5_AvgPressure    -1.544644e+06 4.727589e+05 -3.26729681 0.1890826
## Layer5_StdPressure    -3.079761e+06 1.683757e+06 -1.82910024 0.3185130
## Layer5_MaxPower       -2.167737e+01 1.888592e+02 -0.11478053 0.9272468
## Layer5_MinPower       -1.351871e+02 1.611306e+02 -0.83899074 0.5555962
## Layer5_AvgPower        1.598602e+01 3.007725e+01  0.53149885 0.6889934
## Layer5_StdPower        3.332686e+02 4.774510e+02  0.69801637 0.6120481
## Layer5_MaxRate         1.258398e+02 4.271100e+02  0.29463101 0.8175932
## Layer5_MinRate        -1.501372e+02 6.495092e+02 -0.23115483 0.8553823
## Layer5_AvgRate        -8.341996e+03 3.433670e+03 -2.42946927 0.2485853
## Layer5_StdRate        -1.600460e+03 1.571754e+03 -1.01826380 0.4942392
## Layer5_FinalThickness  1.857280e+01 4.205895e+01  0.44158963 0.7352694
## Layer6_MaxPressure     2.506974e+05 7.195861e+04  3.48391087 0.1779474
## Layer6_MinPressure    -6.026158e+05 2.010384e+05 -2.99751566 0.2049910
## Layer6_AvgPressure     4.834392e+05 1.781542e+05  2.71360000 0.2247728
## Layer6_StdPressure    -1.878102e+06 7.400711e+05 -2.53773212 0.2389672
## Layer6_MaxPower        1.389358e+02 4.744877e+01  2.92812179 0.2095094
## Layer6_AvgPower       -2.105254e+02 1.022582e+02 -2.05876404 0.2878573
## Layer6_StdPower        5.886929e+02 2.400349e+02  2.45253029 0.2464755
## Layer6_MaxRate         2.837922e+03 1.444455e+03  1.96470043 0.2997260
## Layer6_MinRate        -2.309376e+03 1.241623e+03 -1.85996553 0.3140494
## Layer6_AvgRate        -5.167079e+03 1.496566e+04 -0.34526231 0.7883571
## Layer6_StdRate        -2.140255e+04 9.997401e+03 -2.14081137 0.2781986
## Layer6_AvgO2Flow      -4.745872e+01 3.794149e+03 -0.01250839 0.9920373
## Layer6_StdO2Flow      -7.369106e+03 1.482400e+03 -4.97106329 0.1263784
## Layer6_FinalThickness -2.047383e+02 1.015365e+02 -2.01640201 0.2930925
## Layer7_MaxPressure     2.970737e+06 9.853993e+05  3.01475419 0.2038976
## Layer7_MinPressure    -3.667502e+06 1.525033e+06 -2.40486764 0.2508742
## Layer7_AvgPressure     4.330035e+05 5.996674e+05  0.72207287 0.6018663
## Layer7_StdPressure    -4.598770e+06 2.500517e+06 -1.83912782 0.3170502
## Layer7_MaxPower       -2.008774e+02 1.341230e+02 -1.49771117 0.3747829
## Layer7_MinPower       -4.852867e+01 5.229986e+01 -0.92789299 0.5237997
## Layer7_AvgPower        2.336545e+02 1.497765e+02  1.56002124 0.3628951
## Layer7_StdPower        3.192566e+02 1.866011e+02  1.71090434 0.3367300
## Layer7_MaxRate         1.462402e+03 1.050127e+03  1.39259565 0.3964612
## Layer7_MinRate         3.145112e+02 1.253573e+02  2.50891866 0.2414571
## Layer7_AvgRate        -1.686344e+04 3.430414e+03 -4.91586279 0.1277599
```

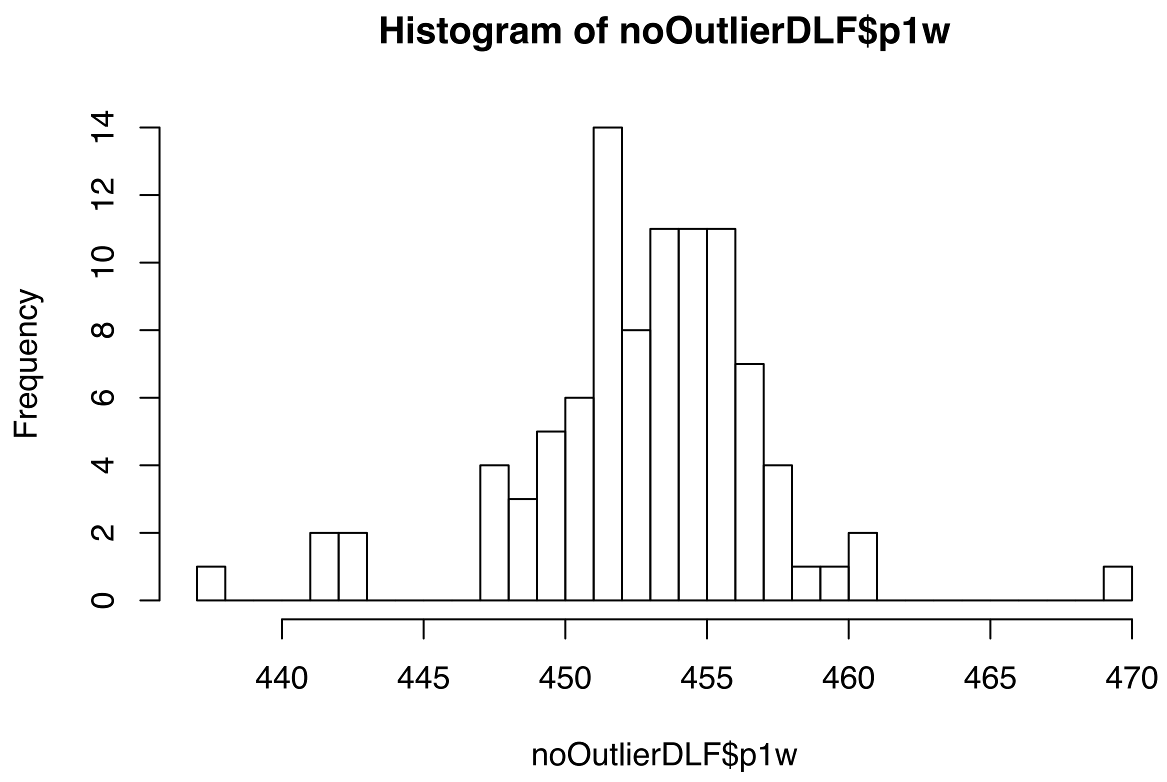
Looking at the p-values nothing appears to be significant. I can also inspect a plot of p1w.

```r
hist(noOutlierDLF$p1w, breaks = 100)
```



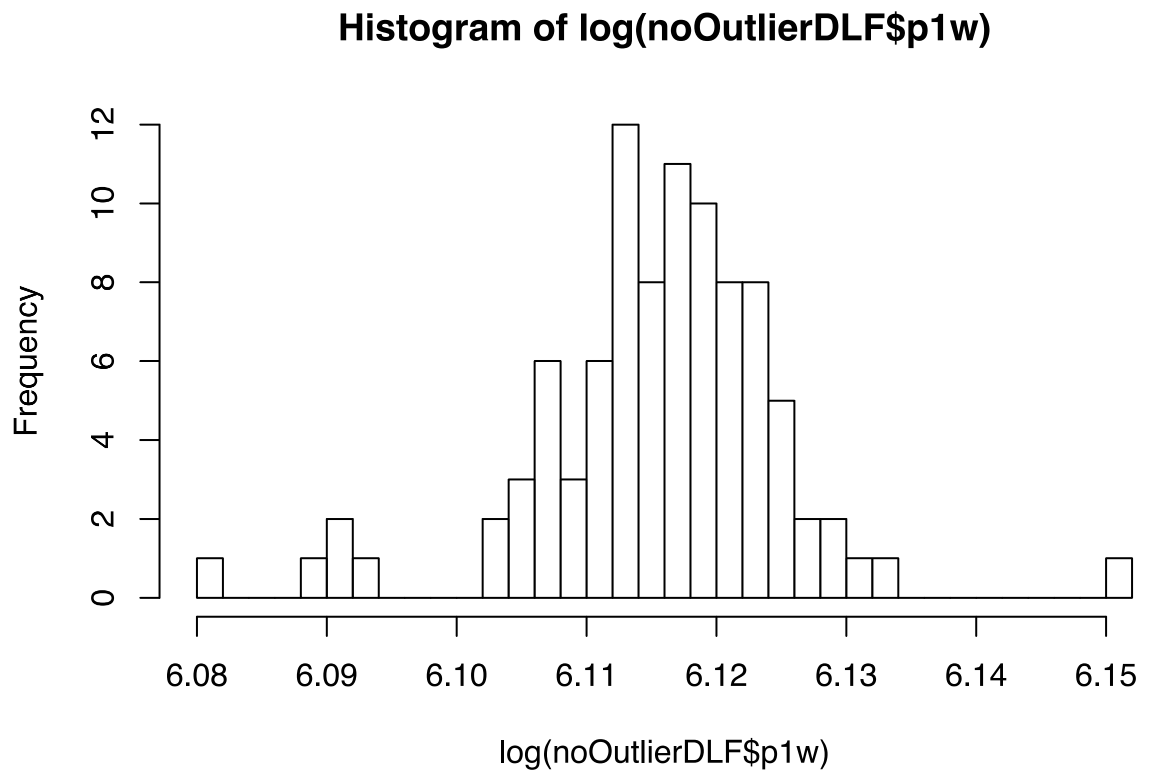
From the histogram I can see that the data looks roughly normally distributed with one large outlier that most likely is having a strong effect on the model. It looks like that point represents a time where the peak fitting algorithm that I wrote in Python grabbed a the high wavelength peak and didn't find the low wavelength peak. I will remove this data-point for now. It also could be a TLF spectra that got into the DLF data.

```r
noOutlierDLF <- noOutlierDLF[-c(which.max(noOutlierDLF$p1w)), ]
hist(noOutlierDLF$p1w, breaks = 25)
```



Now the "no outlier" histogram lives up to its name. Here we see a left skewed histogram of the location of the first peak in the DLF spectra. I will run the linear model again without the outlier which represented a wrongly identified peak. I also will look at the log of the response.

```
hist(log(noOutlierDLF$p1w), breaks = 25)
```



```r
library(MASS)
lm.fit.dlf.p1w <- lm(log(p1w) ~ ., data = lm.dlf)
summary.dlf.p1w <- summary(lm.fit.dlf.p1w)
summary.dlf.p1w$coefficients
```

```
##                            Estimate   Std. Error     t value  Pr(>|t|)
## (Intercept)           -8.674976e+00 1.691737e+01 -0.51278507 0.6983547
## Layer1_MaxPressure    -1.310142e+03 6.351782e+02 -2.06263671 0.2873874
## Layer1_MinPressure     1.553769e+03 7.059593e+02  2.20093260 0.2714979
## Layer1_AvgPressure    -2.479843e+02 1.946075e+02 -1.27427916 0.4235924
## Layer1_StdPressure     6.333825e+03 2.445820e+03  2.58965285 0.2346019
## Layer1_MaxPower       -9.087352e-02 5.974953e-02 -1.52090785 0.3702778
## Layer1_MinPower        1.185855e+01 5.280202e+00  2.24585194 0.2666861
## Layer1_AvgPower       -4.351346e-03 2.889732e-02 -0.15057956 0.9048529
## Layer1_StdPower        4.276801e-01 1.329138e-01  3.21772438 0.1918236
## Layer1_MaxRate         1.240281e+00 6.840723e-01  1.81308481 0.3208752
## Layer1_MinRate        -3.147933e+00 1.107334e+00 -2.84280393 0.2153345
## Layer1_AvgRate         5.089365e+00 1.241315e+00  4.09997884 0.1523003
## Layer1_StdRate        -4.974449e+00 2.600192e+00 -1.91310862 0.3066277
## Layer1_FinalThickness  6.613544e-02 7.388112e-02  0.89516020 0.5351818
## Layer2_MaxPressure    -3.127853e+02 1.271763e+02 -2.45946284 0.2458479
## Layer2_MinPressure    -1.936428e+02 7.550422e+01 -2.56466189 0.2366839
## Layer2_AvgPressure     3.829410e+02 1.627652e+02  2.35272097 0.2558602
## Layer2_StdPressure    -6.711396e+02 2.995370e+02 -2.24058983 0.2672415
## Layer2_MaxPower        3.154726e-01 1.035932e-01  3.04530128 0.2019875
## Layer2_AvgPower       -3.543164e-01 9.054916e-02 -3.91297244 0.1592854
## Layer2_StdPower        3.043338e-01 1.352974e-01  2.24936960 0.2663161
## Layer2_MaxRate        -6.076346e+00 4.165969e+00 -1.45856729 0.3826078
## Layer2_MinRate         1.382176e+00 1.681798e+00  0.82184395 0.5620570
## Layer2_AvgRate         4.041439e+01 1.474463e+01  2.74095621 0.2227089
## Layer2_StdRate         3.928696e+01 1.120987e+01  3.50467688 0.1769466
## Layer2_MaxO2Flow      -3.743917e+00 1.140832e+00 -3.28174269 0.1882981
## Layer2_AvgO2Flow      -1.953387e+00 7.927263e-01 -2.46413841 0.2454263
## Layer2_StdO2Flow      -2.309844e+00 3.870481e+00 -0.59678478 0.6574655
## Layer2_FinalThickness  5.323408e-02 1.657698e-01  0.32113261 0.8021831
## Layer3_MaxPressure     2.710614e+03 1.298619e+03  2.08730462 0.2844274
## Layer3_MinPressure    -2.244036e+02 3.698818e+02 -0.60668996 0.6528359
## Layer3_AvgPressure    -2.175110e+03 1.538702e+03 -1.41360039 0.3919567
## Layer3_StdPressure    -1.204129e+03 1.461601e+03 -0.82384230 0.5612984
## Layer3_MaxPower       -3.180383e-01 8.295882e-02 -3.83368858 0.1624397
## Layer3_MinPower       -1.055956e-01 1.828838e-01 -0.57739204 0.6666467
## Layer3_AvgPower       -2.065730e-02 9.065108e-02 -0.22787709 0.8573646
## Layer3_StdPower        2.739190e-01 2.143694e-01  1.27778980 0.4227420
## Layer3_MaxRate        -2.468957e+00 1.459468e+00 -1.69168347 0.3398721
## Layer3_MinRate         1.678982e+00 6.102151e-01  2.75145871 0.2219261
## Layer3_AvgRate         7.222208e+00 7.035299e+00  1.02656741 0.4916547
## Layer3_StdRate         7.352408e+00 3.050209e+00  2.41046076 0.2503503
## Layer3_FinalThickness  1.126273e-01 7.630627e-02  1.47599048 0.3790898
## Layer4_MaxPressure    -5.880540e+01 2.829885e+01 -2.07801382 0.2855355
## Layer4_MinPressure    -8.681438e+01 1.436069e+02 -0.60452794 0.6538430
## Layer4_AvgPressure     2.501069e+02 1.569884e+02  1.59315551 0.3568431
## Layer4_StdPressure     1.409372e+03 3.312322e+02  4.25493683 0.1469522
## Layer4_MaxPower       -3.227928e-01 1.508244e-01 -2.14019023 0.2782694
## Layer4_AvgPower        4.426208e-01 1.967952e-01  2.24914414 0.2663398
## Layer4_StdPower       -7.767340e-01 2.438059e-01 -3.18586995 0.1936260
## Layer4_MaxRate        -3.553280e+00 5.172609e+00 -0.68694144 0.6168135
## Layer4_MinRate         2.373678e+00 1.461090e+00  1.62459443 0.3512654
## Layer4_AvgRate         5.737392e+01 7.428005e+01  0.77240021 0.5813055
## Layer4_StdRate         3.879883e+01 1.666121e+01  2.32869229 0.2582214
## Layer4_AvgO2Flow       6.335555e+00 9.217949e+00  0.68730632 0.6166557
## Layer4_StdO2Flow       1.516655e+01 6.685267e+00  2.26865220 0.2643047
## Layer4_FinalThickness -3.384307e-01 1.838267e-01 -1.84103158 0.3167739
## Layer5_MaxPressure     1.625792e+03 1.113060e+03  1.46065156 0.3821839
## Layer5_MinPressure     1.337622e+03 6.493452e+02  2.05995508 0.2877126
## Layer5_AvgPressure    -3.107646e+03 1.040197e+03 -2.98755394 0.2056281
## Layer5_StdPressure    -6.244407e+03 3.704722e+03 -1.68552678 0.3408897
## Layer5_MaxPower       -5.457126e-02 4.155414e-01 -0.13132570 0.9168712
## Layer5_MinPower       -2.721266e-01 3.545308e-01 -0.76756831 0.5832367
## Layer5_AvgPower        2.676469e-02 6.617808e-02  0.40443435 0.7553322
## Layer5_StdPower        7.192424e-01 1.050521e+00  0.68465289 0.6178044
## Layer5_MaxRate         2.755919e-01 9.397574e-01  0.29325856 0.8183975
## Layer5_MinRate        -2.272152e-01 1.429096e+00 -0.15899229 0.8996225
## Layer5_AvgRate        -1.733939e+01 7.555003e+00 -2.29508674 0.2615933
## Layer5_StdRate        -3.223840e+00 3.458283e+00 -0.93220842 0.5223267
## Layer5_FinalThickness  3.965729e-02 9.254106e-02  0.42853722 0.7422563
## Layer6_MaxPressure     5.041296e+02 1.583284e+02  3.18407596 0.1937285
## Layer6_MinPressure    -1.208666e+03 4.423388e+02 -2.73244473 0.2233471
## Layer6_AvgPressure     9.837899e+02 3.919873e+02  2.50974932 0.2413847
## Layer6_StdPressure    -3.784092e+03 1.628356e+03 -2.32387229 0.2586999
## Layer6_MaxPower        2.818448e-01 1.044001e-01  2.69965998 0.2258387
## Layer6_AvgPower       -4.315338e-01 2.249956e-01 -1.91796552 0.3059655
## Layer6_StdPower        1.184015e+00 5.281417e-01  2.24185126 0.2671082
## Layer6_MaxRate         5.707826e+00 3.178192e+00  1.79593468 0.3234404
## Layer6_MinRate        -4.784851e+00 2.731906e+00 -1.75147000 0.3302685
## Layer6_AvgRate        -1.200149e+01 3.292850e+01 -0.36447126 0.7774962
## Layer6_StdRate        -4.362998e+01 2.199698e+01 -1.98345297 0.2972881
## Layer6_AvgO2Flow      -2.745130e-01 8.348154e+00 -0.03288308 0.9790735
## Layer6_StdO2Flow      -1.487366e+01 3.261682e+00 -4.56011890 0.1374305
## Layer6_FinalThickness -4.054438e-01 2.234077e-01 -1.81481586 0.3206183
## Layer7_MaxPressure     6.038077e+03 2.168145e+03  2.78490501 0.2194681
## Layer7_MinPressure    -7.468038e+03 3.355485e+03 -2.22562124 0.2688332
## Layer7_AvgPressure     8.922298e+02 1.319430e+03  0.67622351 0.6214724
## Layer7_StdPressure    -9.319379e+03 5.501812e+03 -1.69387439 0.3395112
## Layer7_MaxPower       -4.110715e-01 2.951067e-01 -1.39295869 0.3963825
## Layer7_MinPower       -1.071553e-01 1.150738e-01 -0.93118768 0.5226745
## Layer7_AvgPower        4.787875e-01 3.295488e-01  1.45285768 0.3837732
## Layer7_StdPower        6.411827e-01 4.105728e-01  1.56167846 0.3625881
## Layer7_MaxRate         2.998829e+00 2.310563e+00  1.29787784 0.4179315
## Layer7_MinRate         6.508494e-01 2.758199e-01  2.35968994 0.2551831
## Layer7_AvgRate        -3.434516e+01 7.547837e+00 -4.55033138 0.1377170
```

Still nothing looks particularly statistically significant. I will try the cross validation model to see if removing the outlier and using a log transform helped.

```
library(leaps)
predictors.dlf <- noOutlierDLF[4:122]  #All of the machine variables without the rest of the info
lm.dlf <- predictors.dlf
lm.dlf$p1w <- na.omit(log(noOutlierDLF$p1w))  #The variable to use as the response #ravg
```

```
# Basic Cross Validation for DLF to avoid overfitting
set.seed(2)
train <- sample(c(TRUE, FALSE), nrow(lm.dlf), rep = TRUE)
test <- (!train)

regfit.fwd.dlf <- regsubsets(p1w ~ ., data = lm.dlf[train, ], nvmax = 20, method = "forward")  #model
```

```
## Reordering variables and trying again:
```

```
test.mat.dlf <- model.matrix(p1w ~ ., data = lm.dlf[test, ])  #Predict Test Data
val.errors.dlf <- rep(NA, 20)
for (i in 1:20) {
    coefi <- coef(regfit.fwd.dlf, id = i)
    pred <- test.mat.dlf[, names(coefi)] %*% coefi
    val.errors.dlf[i] <- mean((lm.dlf$p1w[test] - pred)^2)
}

which.min(val.errors.dlf)
```

```
## [1] 3
```

```
# val.errors.dlf
coef(regfit.fwd.dlf, which.min(val.errors.dlf))
```

```
##      (Intercept)  Layer3_MaxPower  Layer2_MinPower Layer6_AvgO2Flow
##    -36.760901642     -0.004083456      2.026836278      0.031611928
```

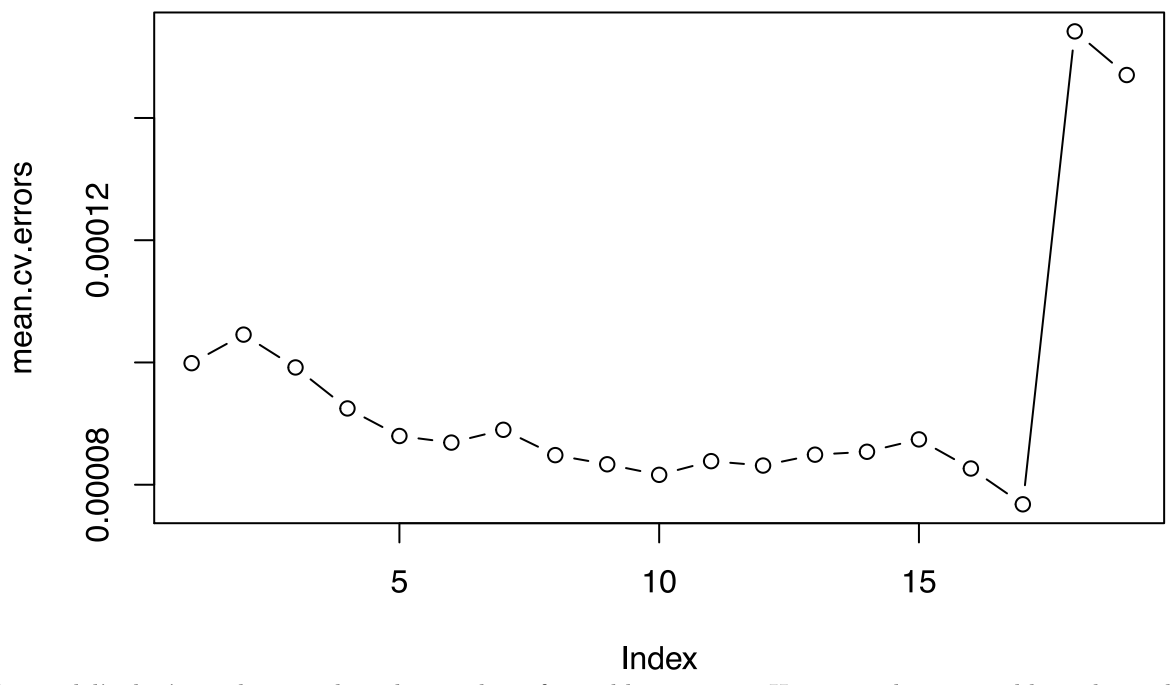
Again I have the problem of constantly changing number of variables with different random seeds. My Most Modelable investigation did not help here. So instead I am going to have to take another crack at a cross validation model.

```
predict.regsubsets <- function(object, newdata, id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id = id)
    xvars <- names(coefi)
    mat[, xvars] %*% coefi
}
predictors.dlf <- noOutlierDLF[4:122]  #All of the machine variables without the rest of the info
lm.dlf <- predictors.dlf
lm.dlf$p1w <- na.omit(log(noOutlierDLF$p1w))  #The variable to use as the response THe log is here!
k <- 10
set.seed(1)
folds <- sample(1:k, nrow(lm.dlf), replace = TRUE)
cv.errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
    log <- capture.output({
        # Hides the Console Vomit!!!
        best.fit <- regsubsets(p1w ~ ., data = lm.dlf[folds != j, ], nvmax = 19,
            method = "forward")
    })
    for (i in 1:19) {
        pred <- predict(best.fit, lm.dlf[folds == j, ], id = i)
        cv.errors[j, i] <- mean((lm.dlf$p1w[folds == j] - pred)^2)
    }
}
```

```
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##            1            2            3            4            5
## 9.988231e-05 1.045580e-04 9.918768e-05 9.247950e-05 8.797124e-05
##            6            7            8            9           10
## 8.689070e-05 8.898467e-05 8.484345e-05 8.334475e-05 8.161923e-05
##           11           12           13           14           15
## 8.385558e-05 8.313655e-05 8.491509e-05 8.538888e-05 8.741533e-05
##           16           17           18           19
## 8.265166e-05 7.679787e-05 1.541635e-04 1.470255e-04
```

```
plot(mean.cv.errors, type = "b")
```

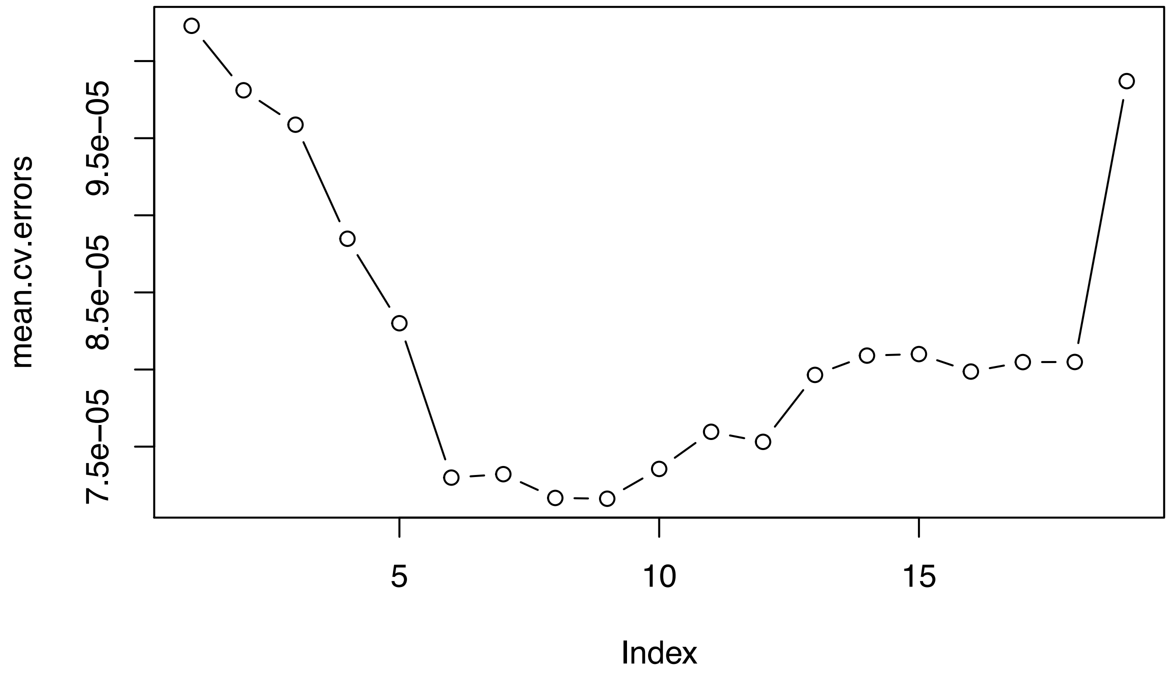


So the model's don't get better when the number of variables goes up. However what is troubling about this is that when I change the number of folds (k) I get different results. The set with k=10 seems to show that more variables doesn't help the situation at all. However when I set (k=5):

```
k <- 5
set.seed(1)
folds <- sample(1:k, nrow(lm.dlf), replace = TRUE)
cv.errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
    log <- capture.output({
        # Hides the Console Vomit!!!
        best.fit <- regsubsets(p1w ~ ., data = lm.dlf[folds != j, ], nvmax = 19,
            method = "forward")
    })
    for (i in 1:19) {
        pred <- predict(best.fit, lm.dlf[folds == j, ], id = i)
        cv.errors[j, i] <- mean((lm.dlf$p1w[folds == j] - pred)^2)
    }
}
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##            1            2            3            4            5 
## 1.022849e-04 9.810419e-05 9.587464e-05 8.847923e-05 8.300256e-05 
##            6            7            8            9           10 
## 7.299317e-05 7.321513e-05 7.167404e-05 7.162370e-05 7.355518e-05 
##           11           12           13           14           15 
## 7.596272e-05 7.531119e-05 7.965064e-05 8.090211e-05 8.100206e-05 
##           16           17           18           19 
## 7.986679e-05 8.048230e-05 8.048956e-05 9.869996e-05
```

```
plot(mean.cv.errors, type = "b")
```



Here it loo,s like a 10 variable model is best. Also the general error is less than when the number of folds was 5. I guess this is because there were twice as many data-points in each fold so it could get a better fit… But this still seems a bit sketchy.

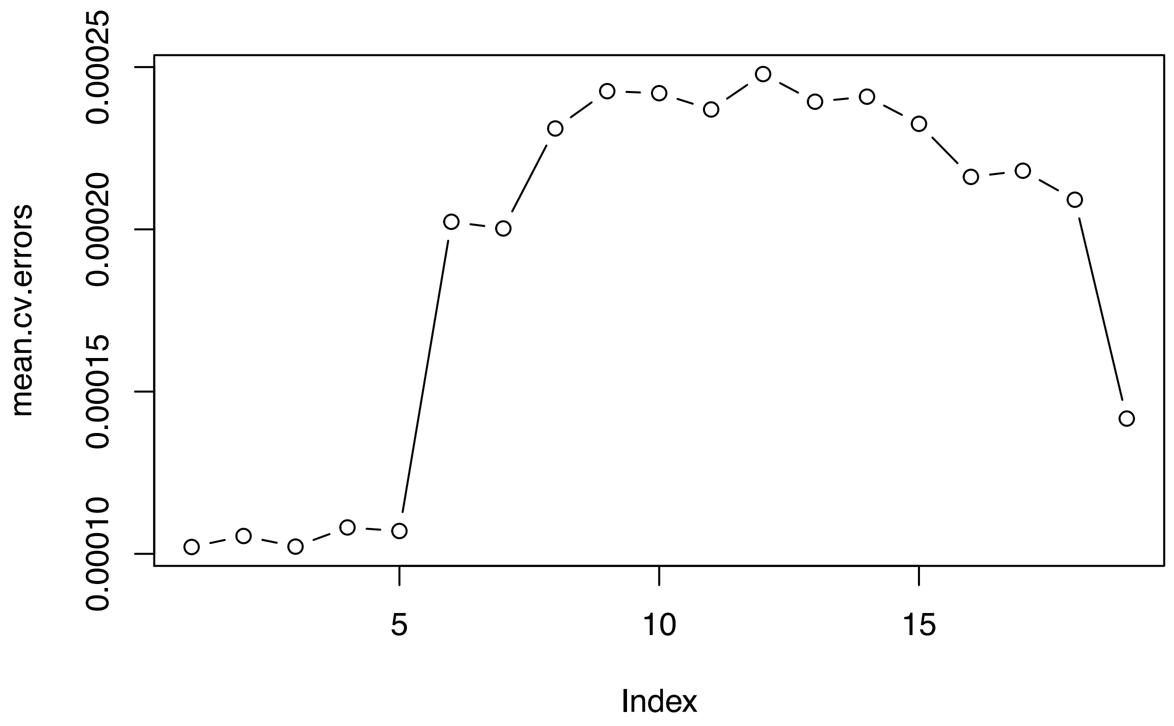
```
k <- 5
set.seed(3)
folds <- sample(1:k, nrow(lm.dlf), replace = TRUE)
cv.errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
    best.fit <- regsubsets(p1w ~ ., data = lm.dlf[folds != j, ], nvmax = 19,
        method = "forward")
    for (i in 1:19) {
        pred <- predict(best.fit, lm.dlf[folds == j, ], id = i)
        cv.errors[j, i] <- mean((lm.dlf$p1w[folds == j] - pred)^2)
    }
}
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```r
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##            1            2            3            4            5
## 0.0001021177 0.0001055020 0.0001022316 0.0001081423 0.0001070473
##            6            7            8            9           10
## 0.0002023286 0.0002002652 0.0002310802 0.0002425851 0.0002419573
##           11           12           13           14           15
## 0.0002369344 0.0002478464 0.0002393308 0.0002408817 0.0002325184
##           16           17           18           19
## 0.0002161577 0.0002180606 0.0002091299 0.0001416816
```

```r
plot(mean.cv.errors, type = "b")
```



Here is kept the number of folds at 5 and simply changed my random seed. Sure enough it looks like it is pointing to a completely different model than before. To follow through with the exercise I am going to try to get the best 5 variable on the model, since all of the error plots looked decent at 5 ad beyond that it got ugly. This is also what my most modelable analysis would land at as the best model for the higher order models.

```r
# Best 5 variable model
reg.best.dlf <- regsubsets(p1w ~ ., data = lm.dlf, nvmax = 19, method = "forward")
```

```
## Reordering variables and trying again:
```

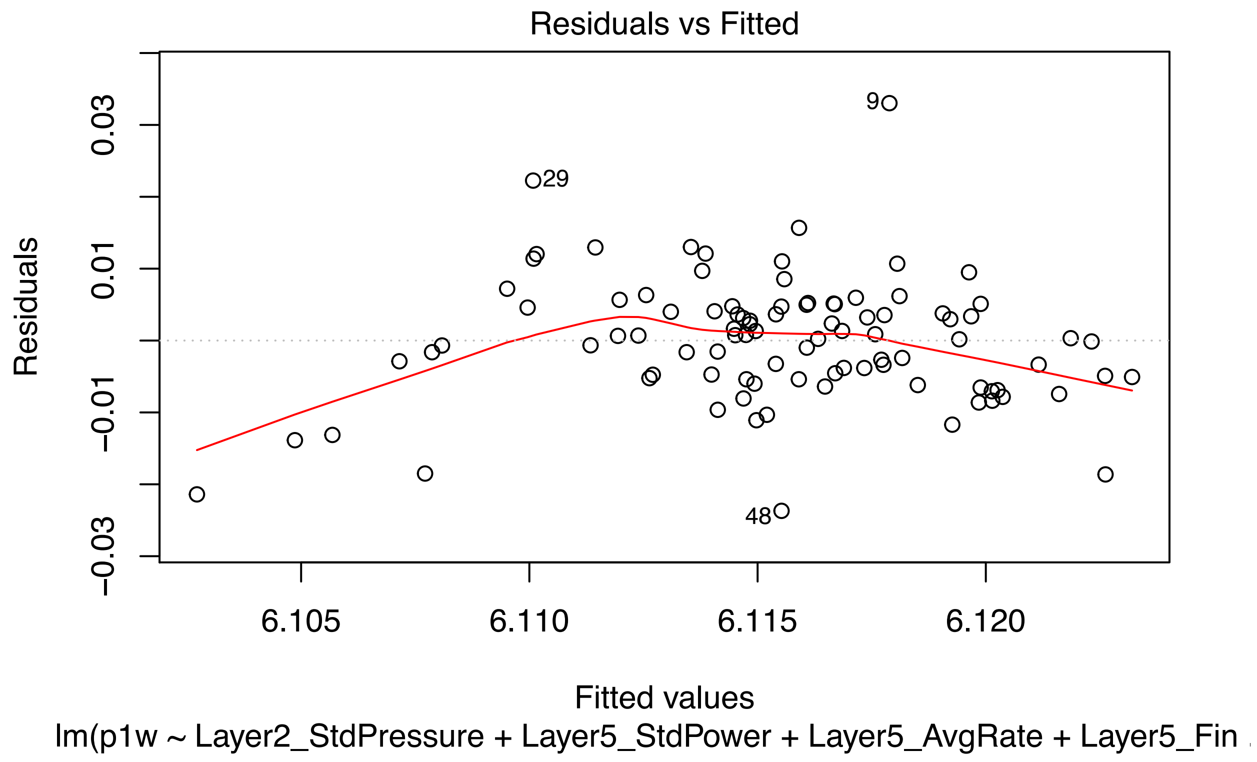
```r
coef(reg.best.dlf, 5)  #Best Model
```

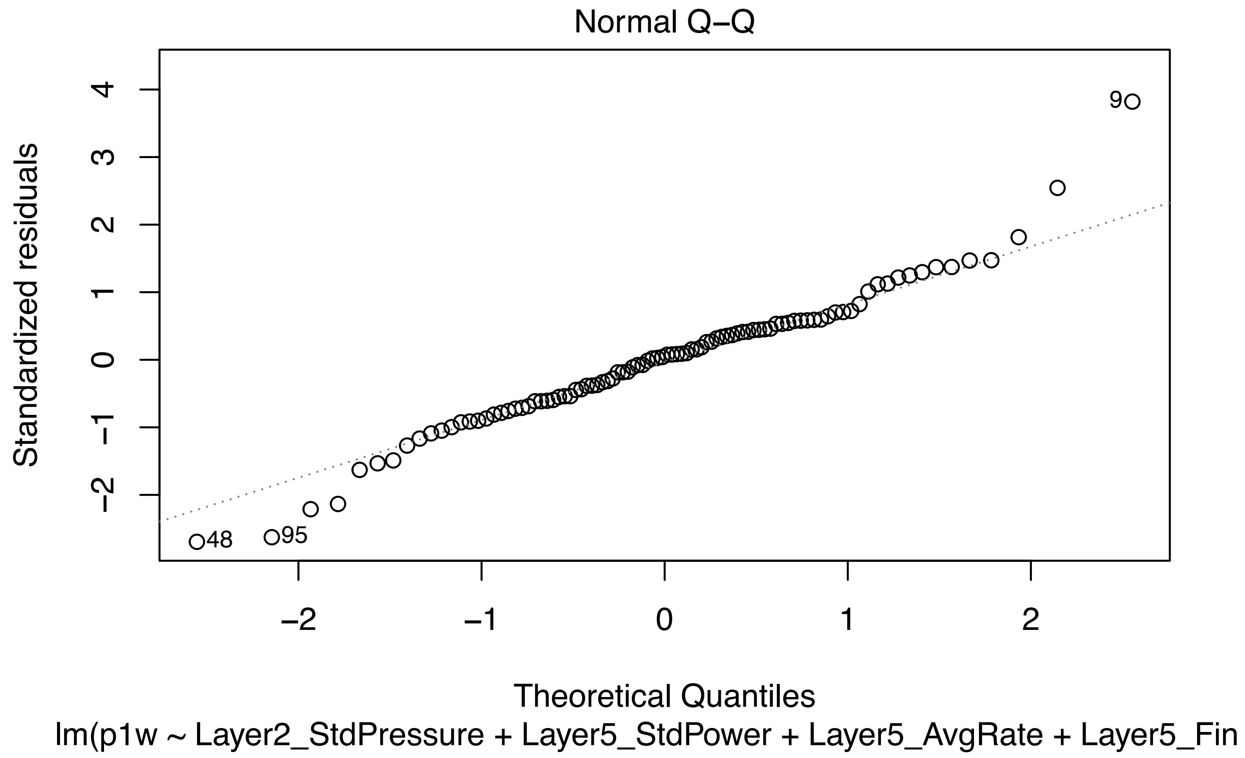
```
##           (Intercept)    Layer2_StdPressure       Layer4_AvgPower
##          5.9449963439         18.6048658393          0.0005191072
##        Layer5_AvgRate Layer5_FinalThickness        Layer7_MinRate
##          0.0466274539          0.0154718723         -0.0658999598
```

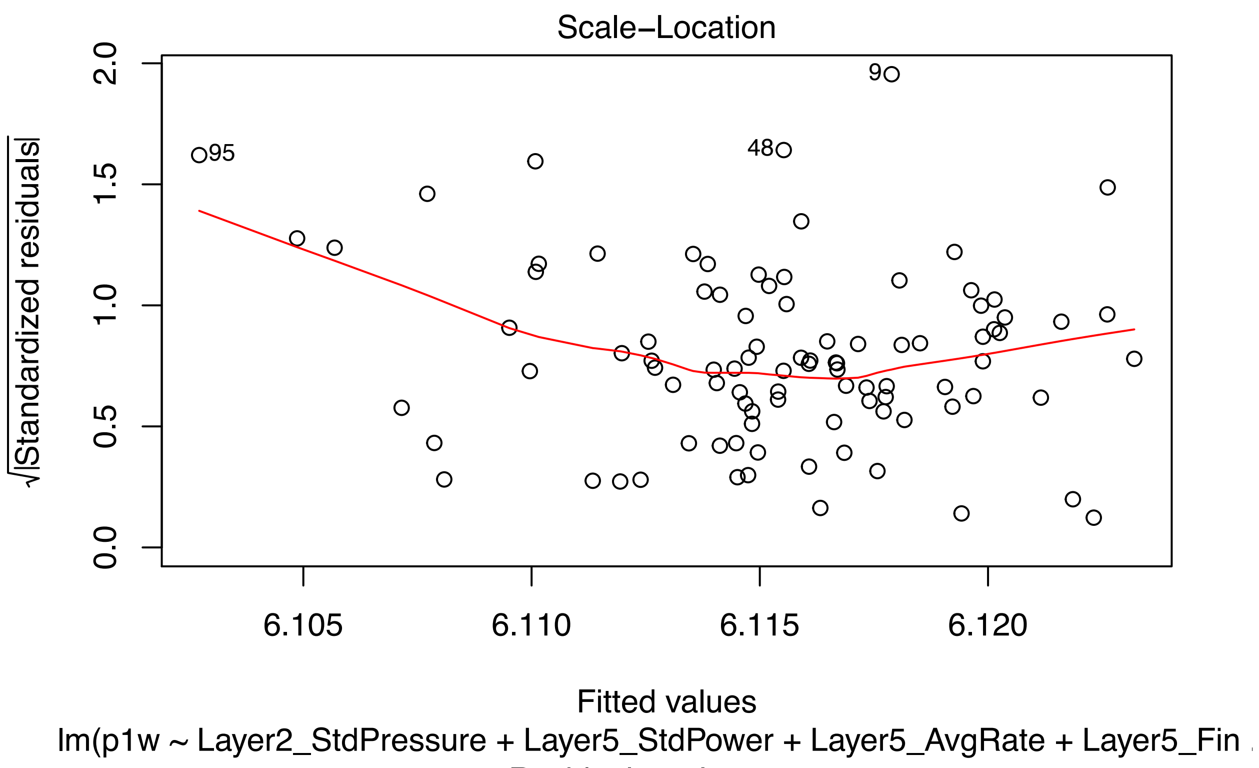
I can try the same with the liner model on those same variables

```r
lm.best.dlf <- lm(p1w ~ Layer2_StdPressure + Layer5_StdPower + Layer5_AvgRate +
    Layer5_FinalThickness + Layer7_AvgRate, data = lm.dlf)
```

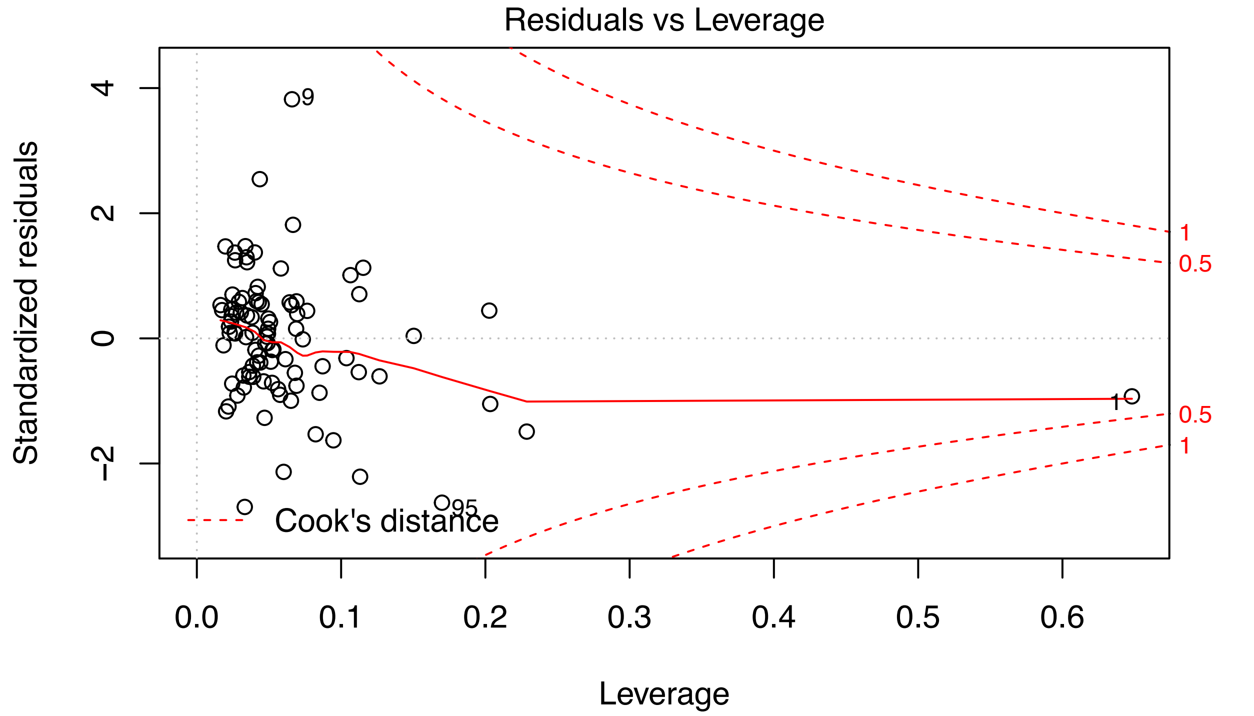
```r
plot(lm.best.dlf)
```







Fitted values
lm(p1w ~ Layer2_StdPressure + Layer5_StdPower + Layer5_AvgRate + Layer5_Fin



Leverage
lm(p1w ~ Layer2_StdPressure + Layer5_StdPower + Layer5_AvgRate + Layer5_Fin

```
summary(lm.best.dlf)
```

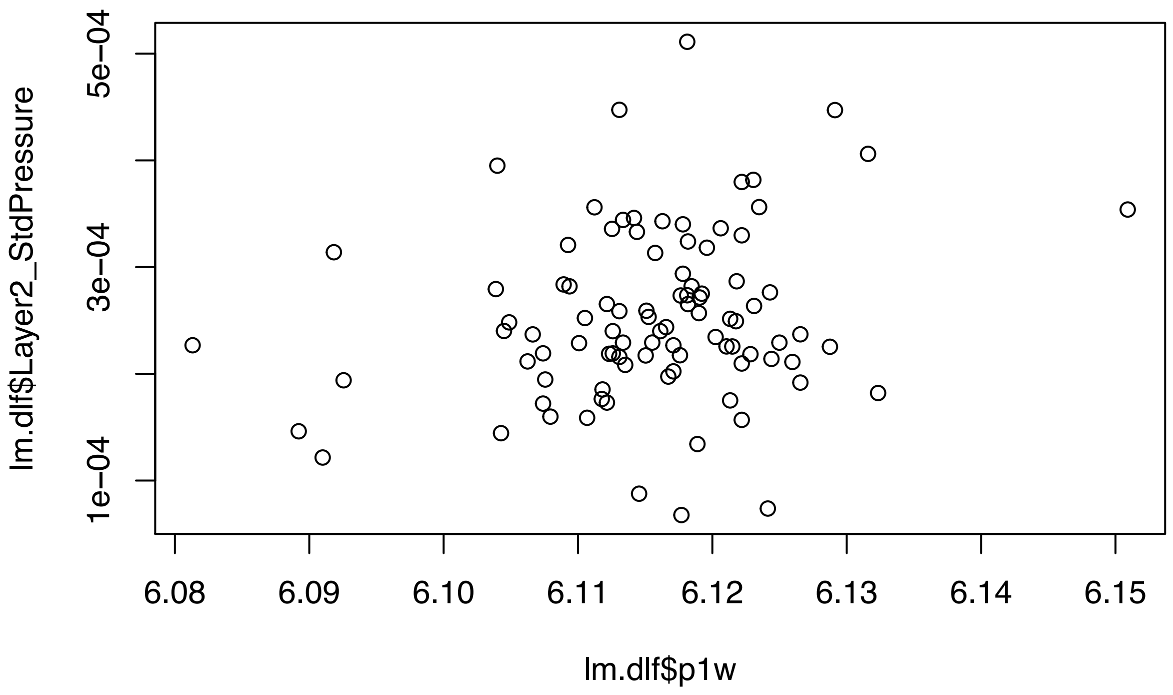
```
##
## Call:
## lm(formula = p1w ~ Layer2_StdPressure + Layer5_StdPower + Layer5_AvgRate +
##     Layer5_FinalThickness + Layer7_AvgRate, data = lm.dlf)
```

```
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.023693 -0.005193  0.000487  0.004756  0.033025
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            6.565203   0.192103  34.175  < 2e-16 ***
## Layer2_StdPressure    21.436989  13.013542   1.647  0.10307
## Layer5_StdPower       -0.002560   0.007950  -0.322  0.74818
## Layer5_AvgRate         0.105814   0.055407   1.910  0.05942 .
## Layer5_FinalThickness  0.016098   0.007554   2.131  0.03587 *
## Layer7_AvgRate        -1.449432   0.437913  -3.310  0.00135 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.008943 on 88 degrees of freedom
## Multiple R-squared:  0.1785, Adjusted R-squared:  0.1319
## F-statistic: 3.825 on 5 and 88 DF,  p-value: 0.003498
```

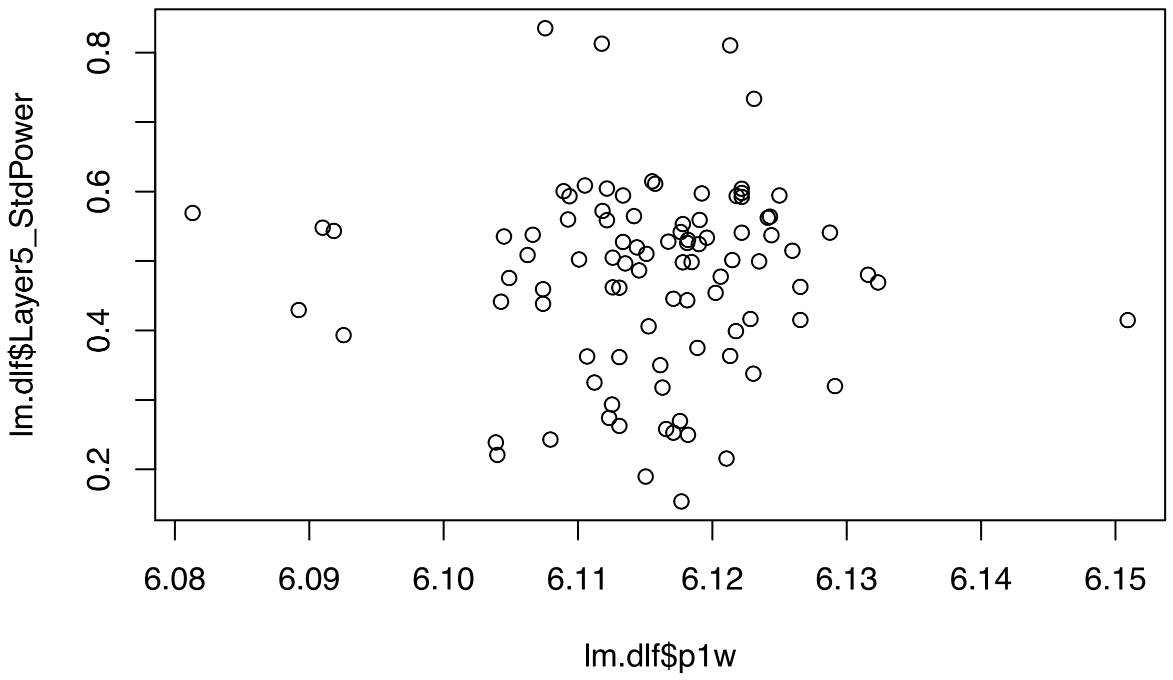
Here we can see the information about the linear model that was made for the data. I suspect that over-fitting has been a huge problem throughout this whole analysis but it is nice to see that these kind of things can be setup to work in a pipeline so that if more data did become available the analysis is scaleable. One thing is that the p-values for the variables in the linear model are much lower than they were for those predictor variables on their own.

I will also plot the scatter plots for the variables used in the model.

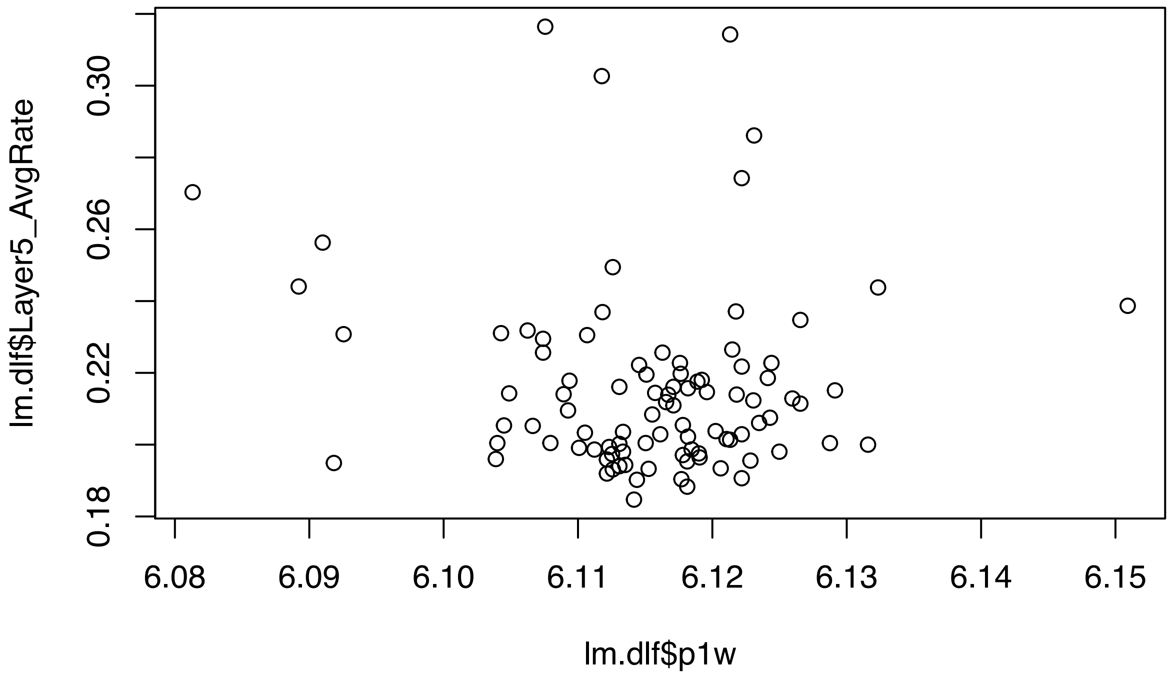
```
plot(lm.dlf$p1w, lm.dlf$Layer2_StdPressure)
```



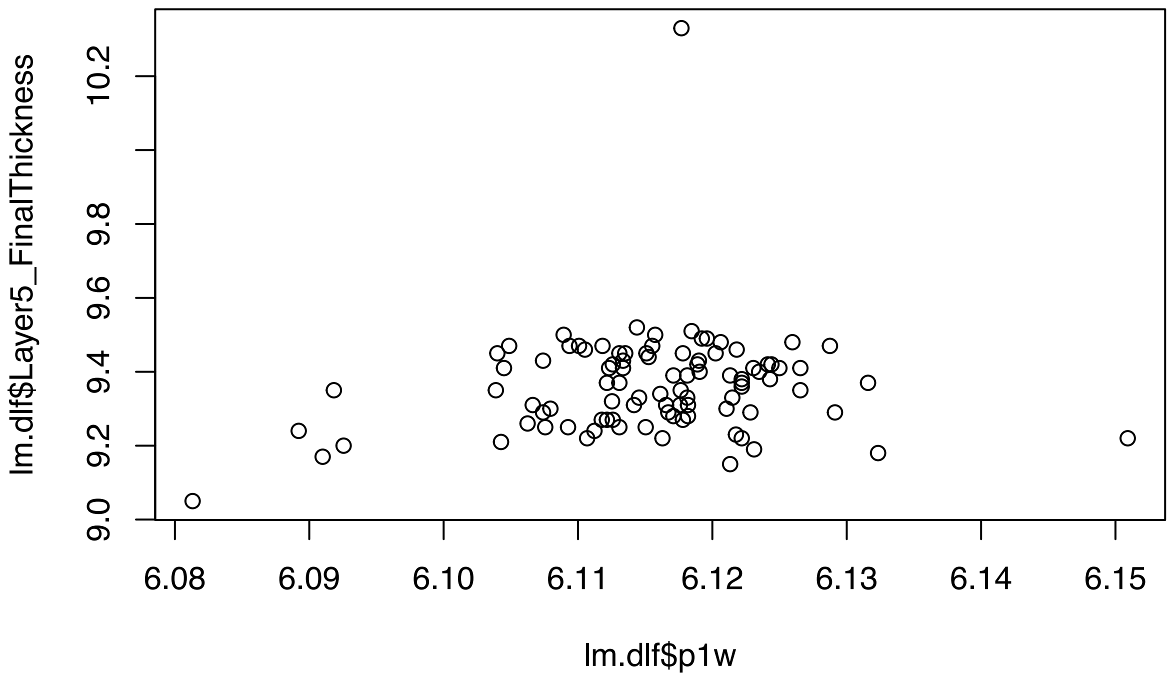
```r
plot(lm.dlf$p1w, lm.dlf$Layer5_StdPower)
```



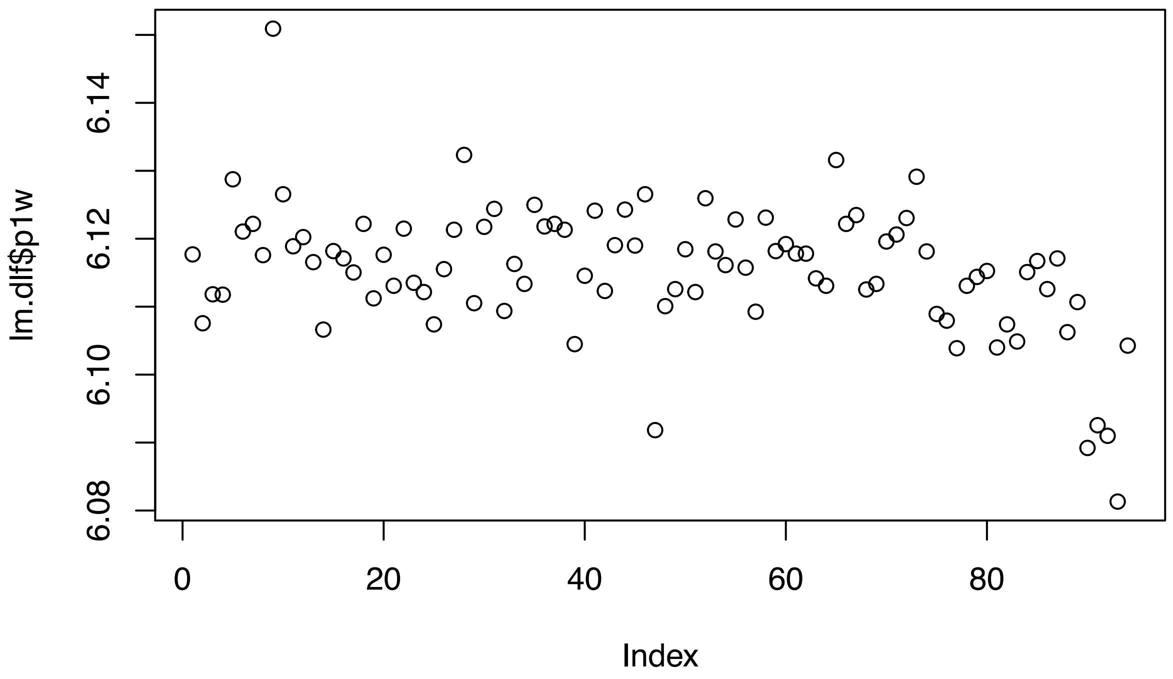
```r
plot(lm.dlf$p1w, lm.dlf$Layer5_AvgRate)
```



```r
plot(lm.dlf$p1w, lm.dlf$Layer5_FinalThickness)
```



```r
plot(lm.dlf$p1w, lm.dlf$Layer2_StdPressureLayer7_AvgRate)
```

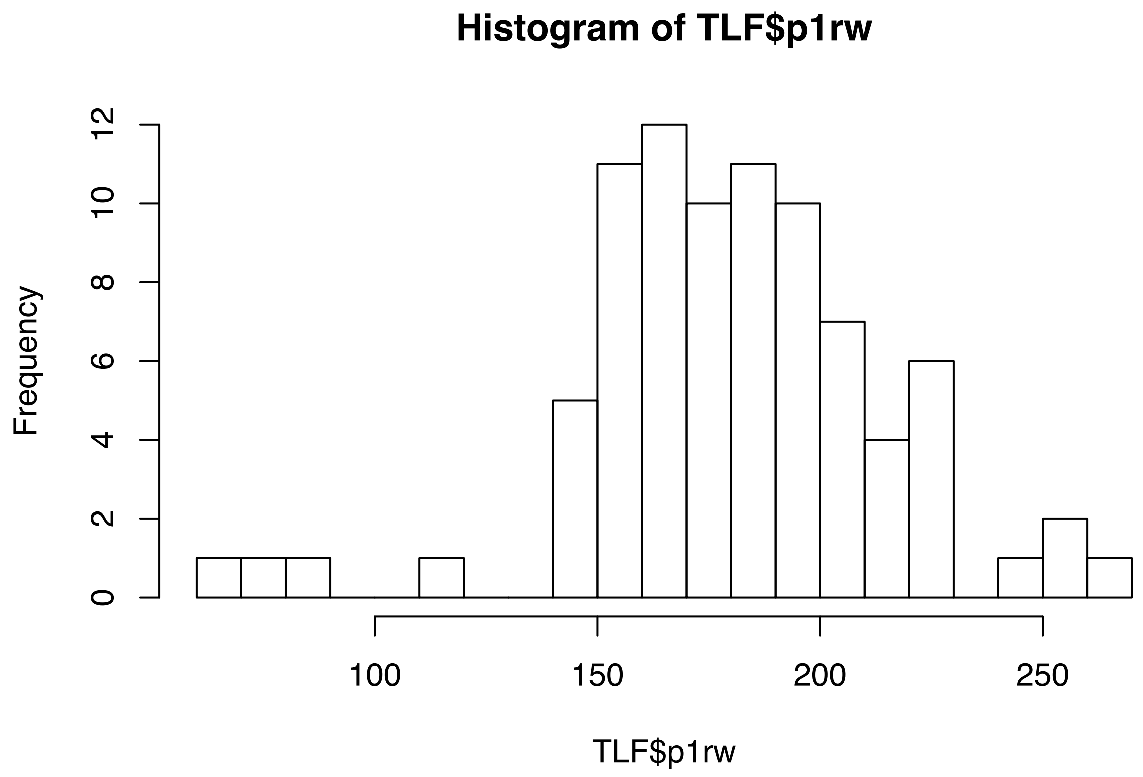


From these scatter plots it seems like there is hardly a discernible relationship between the variables that got incorporated into this model. I think that this is dangerous and I don't have a lot of faith in the model.

2.2.2 Modeling for the TLF product of the first peak's wavelength and reflectance

I am going to try to replicate a similar analysis for the product of the wavelength and the reflectance for the first peak of the TLF.

```r
# Histogram
hist(TLF$p1rw, breaks = 20)
```



Now I will use the MASS package to see if anything is related to this response!

```r
library(MASS)
predictors.tlf <- TLF[4:122]  #All of the machine variables without the rest of the info
lm.tlf <- predictors.tlf
lm.tlf$p1rw <- na.omit(TLF$p1rw)
lm.fit.tlf.p1rw <- lm(p1rw ~ ., data = lm.tlf)
summary.tlf.p1rw <- summary(lm.fit.tlf.p1rw)
summary.tlf.p1rw$coefficients
```

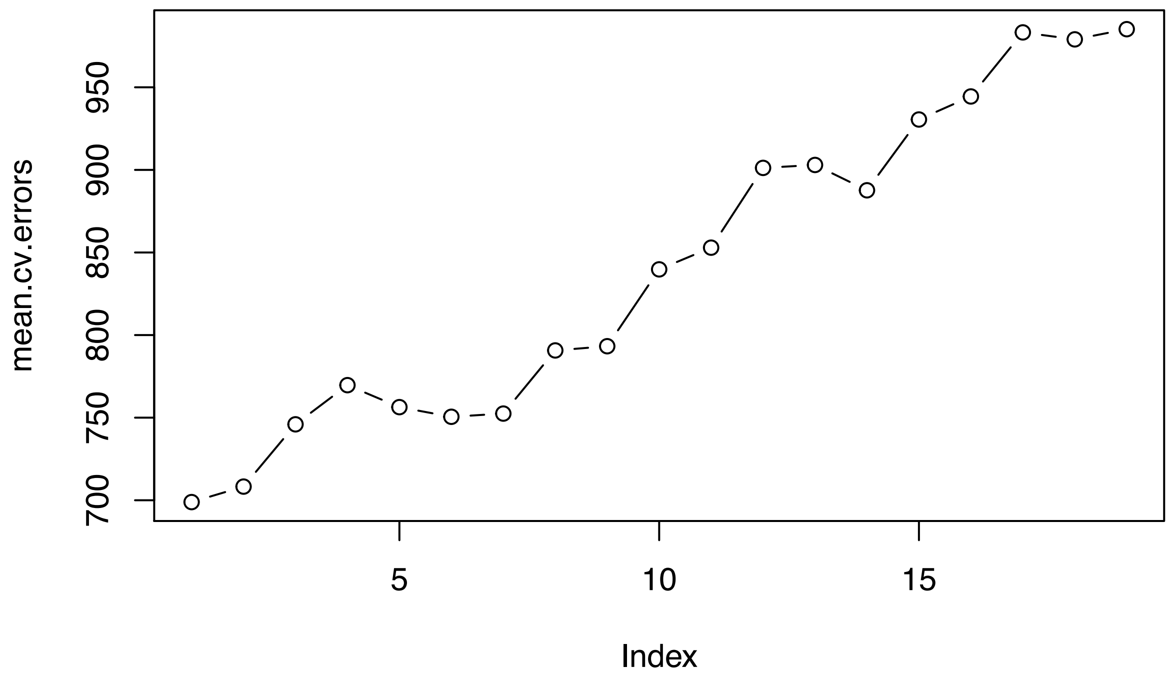
```
##                          Estimate   Std. Error     t value  Pr(>|t|)
## (Intercept)          1.294678e+05 3.364589e+05  0.38479522 0.7162127
## Layer1_MaxPressure  -6.339726e+06 1.489936e+07 -0.42550333 0.6881627
## Layer1_MinPressure   1.265498e+07 3.031690e+07  0.41742349 0.6936870
## Layer1_AvgPressure  -6.102786e+06 1.487471e+07 -0.41027949 0.6985896
## Layer1_StdPressure   9.447014e+06 2.151724e+07  0.43904391 0.6789540
## Layer1_MaxPower      3.445651e+03 7.282433e+03  0.47314554 0.6560446
## Layer1_MinPower     -1.757251e+03 4.399126e+03 -0.39945452 0.7060500
## Layer1_AvgPower      1.572856e+03 4.499546e+03  0.34955880 0.7409111
## Layer1_StdPower     -1.453465e+04 3.096677e+04 -0.46936284 0.6585655
## Layer1_MaxRate       3.574204e+03 1.256315e+04  0.28449900 0.7874344
## Layer1_MinRate       8.083335e+03 2.189886e+04  0.36912131 0.7271529
## Layer1_AvgRate      -3.206197e+05 7.954596e+05 -0.40306222 0.7035594
## Layer1_StdRate       3.102059e+04 1.061708e+05  0.29217629 0.7818871
## Layer1_FinalThickness -9.951221e+02 1.418252e+03 -0.70165390 0.5142010
## Layer2_MaxPressure  -5.119621e+05 1.620341e+06 -0.31595953 0.7647967
## Layer2_MinPressure   8.684542e+05 2.282276e+06  0.38052113 0.7191885
## Layer2_AvgPressure    -1.896320e+06 4.299474e+06 -0.44105865 0.6775892
## Layer2_StdPressure     5.335287e+06 1.357977e+07  0.39288482 0.7105961
## Layer2_MaxPower        1.100358e+03 2.278867e+03  0.48285311 0.6495991
## Layer2_MinPower       -4.431965e+02 1.116781e+03 -0.39685157 0.7078495
## Layer2_AvgPower       -1.206572e+03 2.339378e+03 -0.51576610 0.6280058
## Layer2_StdPower       -2.230687e+03 4.353641e+03 -0.51237259 0.6302133
## Layer2_MaxRate        -2.342042e+03 2.780077e+04 -0.08424380 0.9361316
## Layer2_MinRate         3.964945e+04 7.915938e+04  0.50088131 0.6377210
## Layer2_AvgRate        -6.788146e+04 9.196925e+04 -0.73808868 0.4936237
## Layer2_StdRate        -3.601901e+03 8.145640e+04 -0.04421876 0.9664417
## Layer2_MaxO2Flow       7.077559e+01 2.124920e+02  0.33307417 0.7525908
## Layer2_MinO2Flow       1.306884e+02 2.577866e+02  0.50696345 0.6337411
## Layer2_AvgO2Flow      -2.648005e+02 6.349665e+02 -0.41703066 0.6939562
## Layer2_StdO2Flow       1.187076e+01 1.398048e+02  0.08490951 0.9356284
## Layer2_FinalThickness  1.347838e+03 2.050271e+03  0.65739482 0.5399801
## Layer3_MaxPressure     3.154548e+06 6.612038e+06  0.47709166 0.6534204
## Layer3_MinPressure    -1.586326e+07 3.532725e+07 -0.44903742 0.6721980
## Layer3_AvgPressure     1.219428e+07 2.750332e+07  0.44337492 0.6760219
## Layer3_StdPressure    -1.510730e+07 2.945629e+07 -0.51287186 0.6298882
## Layer3_MaxPower        2.626402e+03 5.981568e+03  0.43908254 0.6789279
## Layer3_MinPower       -1.934961e+03 5.006329e+03 -0.38650290 0.7150253
## Layer3_AvgPower       -2.471084e+03 4.801710e+03 -0.51462585 0.6287470
## Layer3_StdPower       -1.412715e+04 3.161007e+04 -0.44691920 0.6736272
## Layer3_MaxRate         6.127204e+04 1.456553e+05  0.42066475 0.6914683
## Layer3_MinRate         1.918218e+03 2.415775e+04  0.07940383 0.9397915
## Layer3_AvgRate        -4.891549e+05 1.354893e+06 -0.36102832 0.7328309
## Layer3_StdRate        -2.959835e+05 6.688272e+05 -0.44254102 0.6765859
## Layer3_FinalThickness -9.311892e+02 1.641541e+03 -0.56726516 0.5950540
## Layer4_MaxPressure    -1.120932e+04 4.134068e+05 -0.02711450 0.9794173
## Layer4_MinPressure     2.986379e+06 7.565158e+06  0.39475429 0.7093010
## Layer4_AvgPressure     5.207047e+06 8.139243e+06  0.63974588 0.5504958
## Layer4_StdPressure     1.852695e+07 4.711227e+07  0.39325107 0.7103423
## Layer4_MaxPower        4.685120e+02 8.077906e+02  0.57999192 0.5870733
## Layer4_MinPower       -4.463302e+02 7.050029e+02 -0.63308981 0.5544961
## Layer4_AvgPower        6.673493e+02 1.497708e+03  0.44558037 0.6745312
## Layer4_StdPower        1.351568e+03 3.884884e+03  0.34790436 0.7420798
## Layer4_MaxRate        -9.218401e+04 2.088945e+05 -0.44129466 0.6774294
## Layer4_MinRate         1.412776e+04 1.334259e+04  1.05884669 0.3381180
## Layer4_AvgRate         2.263357e+05 5.532237e+05  0.40912148 0.6993858
## Layer4_StdRate         3.475395e+05 6.932166e+05  0.50134337 0.6374182
## Layer4_MaxO2Flow       9.174788e+01 2.245684e+02  0.40855207 0.6997775
## Layer4_MinO2Flow      -3.073708e+02 7.259020e+02 -0.42343293 0.6895762
## Layer4_AvgO2Flow       3.338872e+02 7.902751e+02  0.42249493 0.6902170
## Layer4_StdO2Flow      -6.870817e+02 1.930395e+03 -0.35592807 0.7364193
## Layer4_FinalThickness -3.009012e+01 5.978224e+02 -0.05033287 0.9618060
## Layer5_MaxPressure     1.030123e+07 2.284448e+07  0.45092848 0.6709235
## Layer5_MinPressure    -7.796364e+06 1.552043e+07 -0.50232912 0.6367724
## Layer5_AvgPressure    -2.683408e+06 1.509472e+07 -0.17777129 0.8658803
## Layer5_StdPressure    -1.900553e+07 4.563255e+07 -0.41649060 0.6943263
## Layer5_MaxPower       -9.917229e+02 1.687564e+03 -0.58766521 0.5822934
## Layer5_MinPower        5.153617e+02 1.370986e+03  0.37590588 0.7224081
## Layer5_AvgPower       -2.455812e+01 1.152370e+03 -0.02131097 0.9838219
## Layer5_StdPower        2.548025e+03 5.105453e+03  0.49907907 0.6389030
## Layer5_MaxRate        -3.063644e+04 6.946361e+04 -0.44104302 0.6775998
## Layer5_MinRate         2.908846e+04 6.172863e+04  0.47123124 0.6573197
## Layer5_AvgRate        -1.097650e+04 2.257429e+04 -0.48623921 0.6473590
## Layer5_StdRate         7.702049e+04 1.712524e+05  0.44974830 0.6717188
## Layer5_FinalThickness  2.201409e+03 5.140176e+03  0.42827510 0.6862726
## Layer6_MaxPressure     5.934361e+05 7.873610e+05  0.75370273 0.4849856
## Layer6_MinPressure     1.829088e+05 4.748236e+05  0.38521408 0.7159214
## Layer6_AvgPressure    -6.715675e+06 1.029566e+07 -0.65228235 0.5430125
## Layer6_StdPressure    -9.535675e+05 9.589791e+05 -0.99435698 0.3657038
## Layer6_MaxPower       -2.793823e+02 6.634826e+02 -0.42108460 0.6911811
```

Again nothing looks to correlate with the data very well. But I can continue with cross validation.

```r
predict.regsubsets <- function(object, newdata, id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id = id)
    xvars <- names(coefi)
    mat[, xvars] %*% coefi
}

k <- 10
set.seed(1)
folds <- sample(1:k, nrow(lm.tlf), replace = TRUE)
cv.errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
    log <- capture.output({
        # Hides the Console Vomit!!!
        best.fit <- regsubsets(p1rw ~ ., data = lm.tlf[folds != j, ], nvmax = 19,
            method = "forward")
    })
    for (i in 1:19) {
        pred <- predict(best.fit, lm.tlf[folds == j, ], id = i)
        cv.errors[j, i] <- mean((lm.tlf$p1rw[folds == j] - pred)^2)
    }
}
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8
## 698.8961 708.2723 745.9720 769.6761 756.3733 750.5277 752.4714 790.6994
##        9       10       11       12       13       14       15       16
## 793.2481 839.7715 852.9219 901.2051 902.9833 887.6460 930.5076 944.4606
##       17       18       19
## 983.2298 979.0155 985.1791
```

```r
plot(mean.cv.errors, type = "b")
```

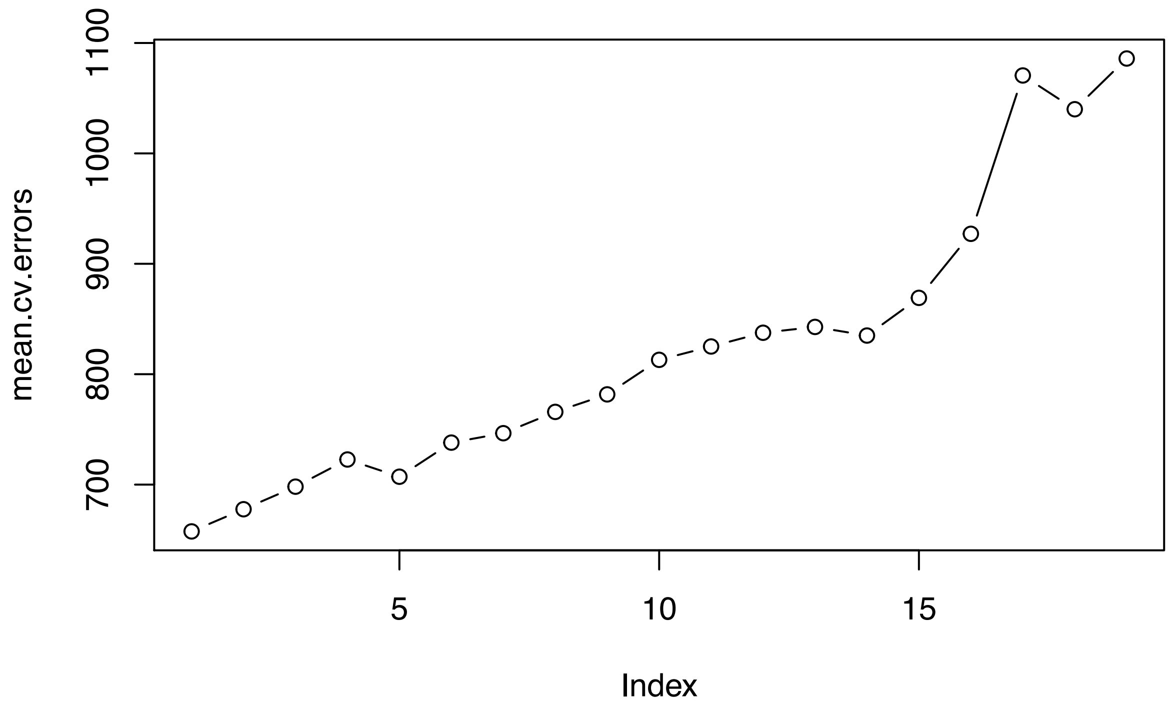


Here things look bleak for a multi-variable model. I will try changing the number of samples in each fold though to see how that affects things.

```r
k <- 5
set.seed(1)
folds <- sample(1:k, nrow(lm.tlf), replace = TRUE)
cv.errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))
for (j in 1:k) {
    log <- capture.output({
        # Hides the Console Vomit!!!
        best.fit <- regsubsets(p1rw ~ ., data = lm.tlf[folds != j, ], nvmax = 19,
            method = "forward")
    })
    for (i in 1:19) {
        pred <- predict(best.fit, lm.tlf[folds == j, ], id = i)
        cv.errors[j, i] <- mean((lm.tlf$p1rw[folds == j] - pred)^2)
    }
}
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##         1         2         3         4         5         6         7
##  657.6080  677.6952  698.1361  722.7445  707.1289  738.0212  746.5466
##         8         9        10        11        12        13        14
##  765.8425  781.7111  812.9983  825.1042  837.5335  842.8025  835.0123
##        15        16        17        18        19
##  869.1306  927.1395 1070.5716 1039.9934 1085.9245
```

```r
plot(mean.cv.errors, type = "b")
```



Again nothing. So for the TLF I would be best off with just a single variable near model.

```r
reg.best.tlf <- regsubsets(p1rw ~ ., data = lm.tlf, nvmax = 19, method = "forward")
```

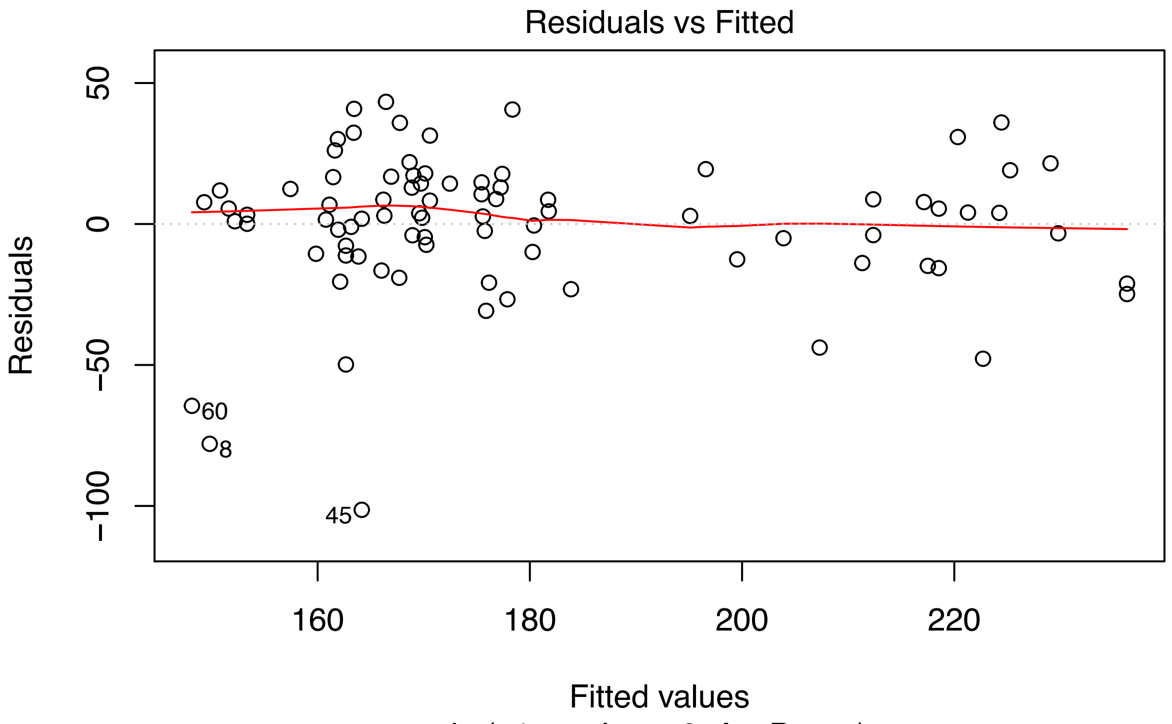
```
## Reordering variables and trying again:
```

```r
coef(reg.best.tlf, 1)  #Best Model
```

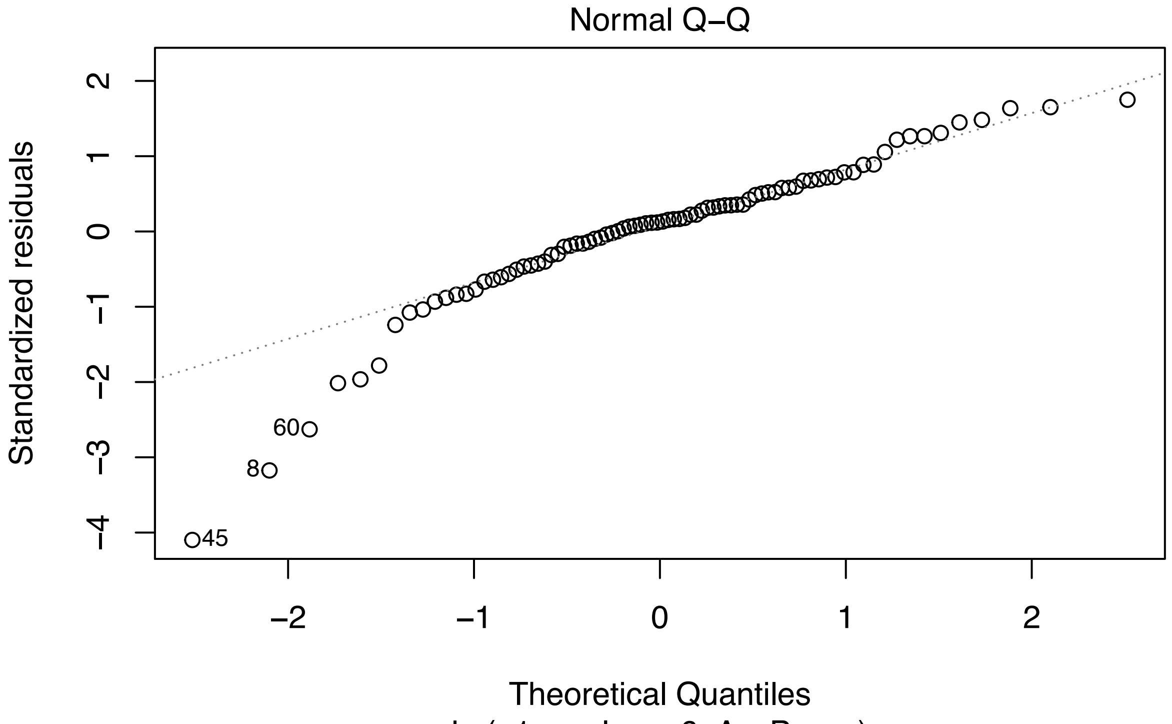
```
##     (Intercept) Layer6_AvgPower
##      -149.99018         7.30365
```

So layer 6 average power is the "best" variable. Now I will plug this into the linear model.

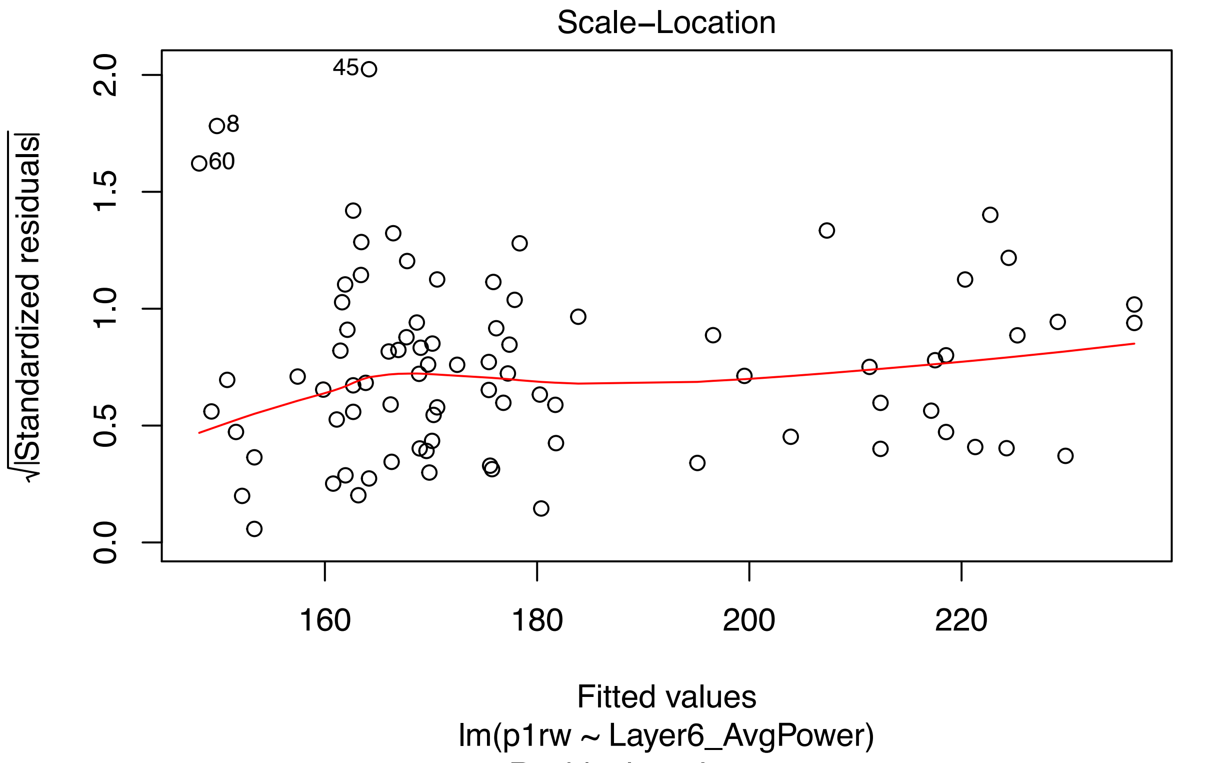
```r
lm.best.tlf <- lm(p1rw ~ Layer6_AvgPower, data = lm.tlf)
plot(lm.best.tlf)
```

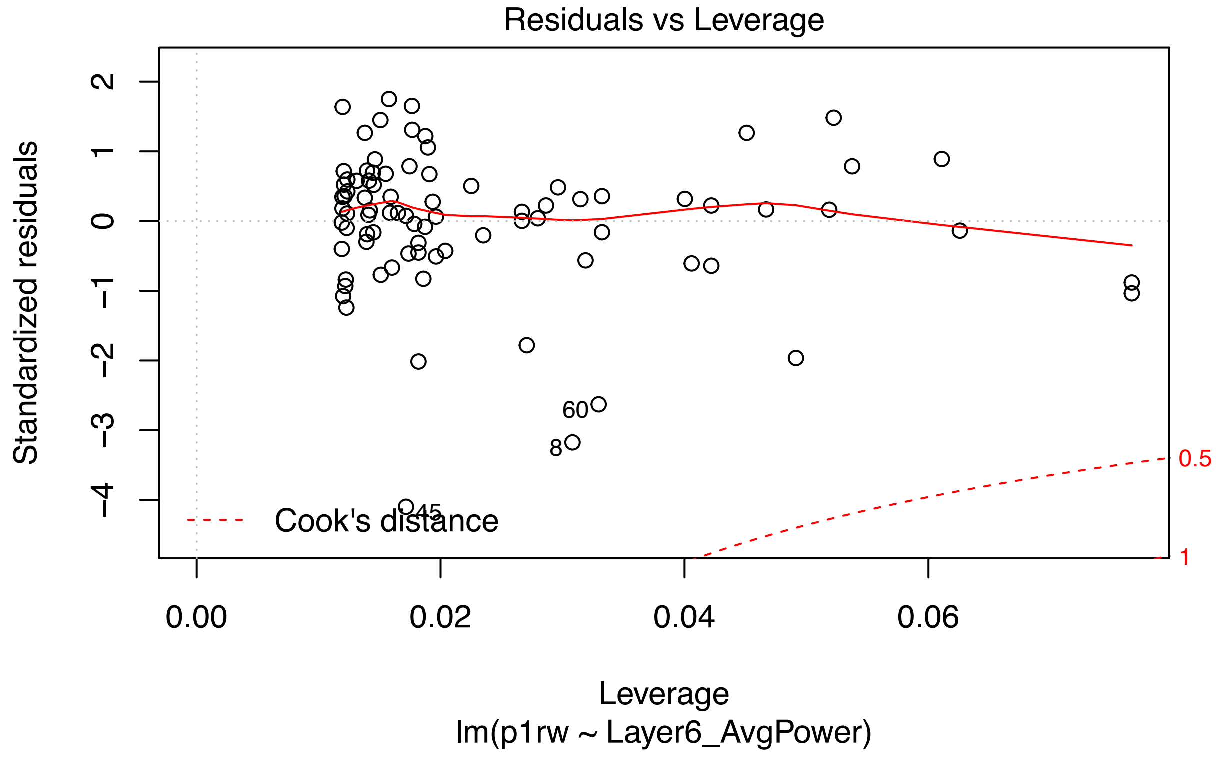


lm(p1rw ~ Layer6_AvgPower)



lm(p1rw ~ Layer6_AvgPower)





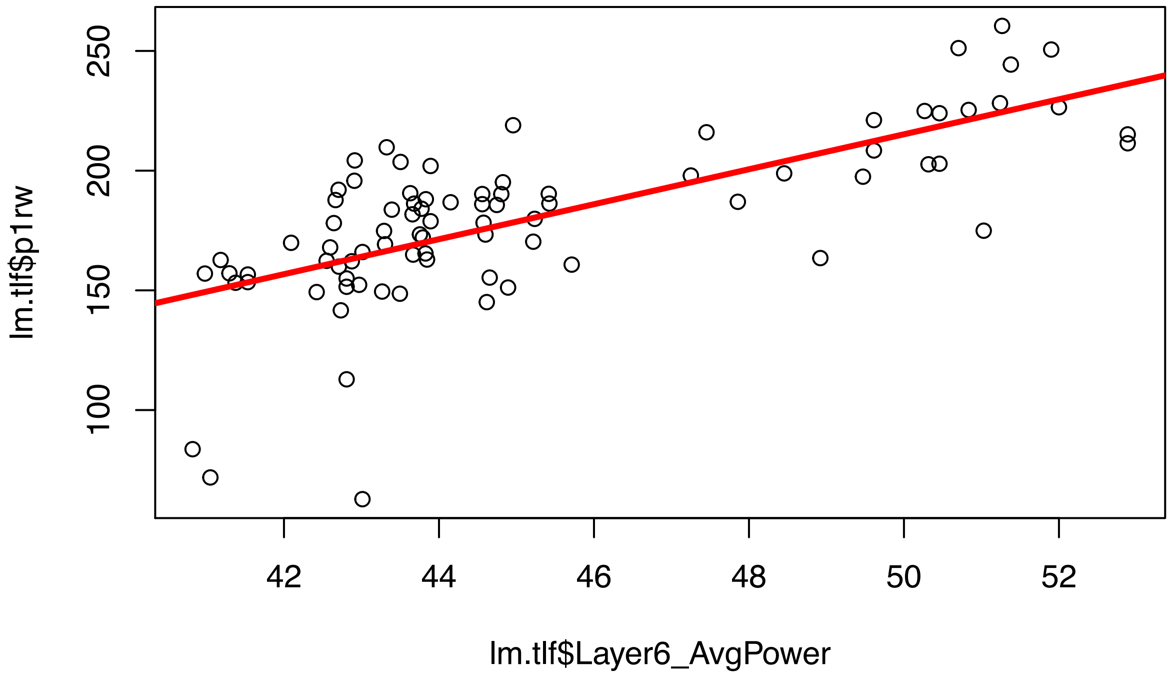
```r
summary(lm.best.tlf)
```

```
##
## Call:
## lm(formula = p1rw ~ Layer6_AvgPower, data = lm.tlf)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -101.371  -10.711    3.103   14.329   43.317 
## 
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)    
## (Intercept)     -149.9902    37.4515  -4.005 0.000136 ***
## Layer6_AvgPower    7.3037     0.8263   8.839 1.51e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 24.95 on 82 degrees of freedom
## Multiple R-squared:  0.4879, Adjusted R-squared:  0.4816 
## F-statistic: 78.12 on 1 and 82 DF,  p-value: 1.508e-13
```

I can plot this model and the data with it since it is just a simple linear model.

```r
lm.best.tlf <- lm(p1rw ~ Layer6_AvgPower, data = lm.tlf)
plot(lm.tlf$Layer6_AvgPower, lm.tlf$p1rw)
abline(lm.best.tlf, lwd = 3, col = "red")
```



3 Colnclusions

Overall I don't feel very confident in this analysis. I did not have a large sample of data and the resolution on the numbers in the machine files was low. So there wasn't a lot of variation in the data. I think that a problem where things aren't changing very much is solved by taking a lot more data and incorporating it into the analysis. I was limited by what I could select and I think that that really almost invalidates this analysis. However I think that the framework for identifying predictors can be scaled up to large samples and that is what I hope to continue to do as I work at VisiMax and finish up my masters thesis.