# EAPS 507 Final Project RMarkdown File

### Student: Isabella Zuffoletti

## Project Goal

In this project I will explore a dataset provided by Los Alamos National Lab (LANL), a US national laboratory concerned with national security. This dataset was produced by a series of experiments that aimed to replicate the stick-slip behavior of faults that produce major earthquakes. The data is a continous waveform with a quasi-periodic nature, and an associated time until ground failure. This dataset was released to the public as a Kaggle competition, so data scientists could compete for the best solution to this unknown problem- can you predict earthquakes based on seismic data alone?

This question proposed by LANL poses a simple regression problem of one variable- the waveform, with the predicted value being the time of ground failure. However, many different versions of the solution were produced, with combinations of signal pre-processing, regression model selection, and feature extraction.

The implication of a solid solution to this problem is the ability monitor the national seismic network, a produce a better early warning system for major earthquakes. Advancing early warning systems by even seconds can prevent significant costs. For example, pausing surgeries, evacuating buildings, and protecting critical infastructure can all be improved with a few nore seconds of warning.

Link to kaggle competition: https://www.kaggle.com/c/LANL-Earthquake-Prediction

# Define function to plot a waveform with time until failure  
# y1 is acoustic signal and y2 is time to failure  
plot\_wave <- function(x, y1, y2){  
 plot.new()  
plot.window(xlim=range(x), ylim=range(y1))  
lines(x, y1, col="red")  
axis(1)  
axis(2, col.axis="red")  
box()  
  
plot.window(xlim=range(x), ylim=range(y2))  
lines(x, y2, col="limegreen", pch=19)  
axis(4, col.axis="limegreen") }

# ============== DOWNSAMPLING ==============

The original waveform was downsampled by pulling only 1 our of every 250 rows in the original continuous train time series. This is still a continuous time series, but at a lower resolution. I break this continuous waveform, provided in “train.csv”, into test and train sets. This is because the test segments provided with the Kaggle competition do not have an attached “time to failure”. The goal of the competition is to best calculate the time to failure, although no one knows the true answer.

Original train file size: 9 Gigabytes  
Downsampled train file size: 1,500,000 rows, 2 columns, 36.7 megabytes  
Downsampled train file size: 1,016,582 rows, 2 columns, 22.7 megabytes

This code does not need to be run, but shows how the file was downsampled using the command line from RStudio. I import the results in the next block of code.

#file.pipe <- pipe("awk ' BEGIN{i=0}{i++;if (i%250==0) print $1}' < train.csv")  
 #df <- read.csv(file.pipe, nrows=1000000, header=FALSE)  
 #df  
  
 #file.pipe <- pipe("awk ' BEGIN{i=0}{i++;if (i%250==0) print $1}' < train.csv")  
 #df <- read.csv(file.pipe, nrows=5000000, header=FALSE)  
   
 #write.csv(df[1:1500000, ], "downsampled\_train.csv")  
 #write.csv(df[1500000:nrow(df), ], "downsampled\_test.csv")

### Read data and initial cleanup

The labeled train data provided was broken into test and train sets, so the model error can be tested. Because this is a continuous time series broken into segments, this may be more correlated than the random segments provided above. However, this is the only other labeled data we were provided with by LANL.

# Read data and cleanup column names  
train <- read.csv("downsampled\_train.csv")  
headers <- c("index", "acoustic", "time\_to\_failure")  
colnames(train) <- headers  
test <- read.csv("downsampled\_test.csv")  
colnames(test) <- headers  
# reorder our index (no longer a continuous waveform with train)  
test$index <- test$index-1499999  
# Cut off test set so that we can break into equal sized segments of 600 rows  
test <- test[1: ((nrow(test) %% 600)\*600),]  
  
# View train data properties  
summary(train)

## index acoustic time\_to\_failure   
## Min. : 1 Min. :-3146.000 Min. : 0.000196   
## 1st Qu.: 375001 1st Qu.: 2.000 1st Qu.: 2.553599   
## Median : 750000 Median : 5.000 Median : 5.259498   
## Mean : 750000 Mean : 4.566 Mean : 5.675172   
## 3rd Qu.:1125000 3rd Qu.: 7.000 3rd Qu.: 8.111299   
## Max. :1500000 Max. : 3644.000 Max. :16.107400

summary(test)

## index acoustic time\_to\_failure   
## Min. : 1 Min. :-546.00 Min. : 0.000496   
## 1st Qu.: 27301 1st Qu.: 2.00 1st Qu.: 6.108597   
## Median : 54600 Median : 4.00 Median : 7.881198   
## Mean : 54600 Mean : 4.47 Mean : 7.821159   
## 3rd Qu.: 81900 3rd Qu.: 7.00 3rd Qu.: 9.653799   
## Max. :109200 Max. : 157.00 Max. :11.426400

Similarly, this block of code does some pre-downsampling of the unlabeled test segments, in order to provide data with a consistent sampling interval to our model.

# original file = 150,000 rows  
# downsampled file = 600 rows  
#file.pipe <- pipe("awk ' BEGIN{i=0}{i++;if (i%250==0) print $1}' < seg\_00030f.csv")  
#df <- read.csv(file.pipe, nrows=600, header=FALSE)  
#write.csv(df[1:nrow(df), ], "seg\_00030f\_downsampled.csv")  
  
#file.pipe <- pipe("awk ' BEGIN{i=0}{i++;if (i%250==0) print $1}' < seg\_0012b5.csv")  
#df <- read.csv(file.pipe, nrows=600, header=FALSE)  
#write.csv(df[1:nrow(df), ], "seg\_0012b5\_downsampled.csv")  
  
# These are unlabeled test segments, which only include the acoustic measurements. The submission for the Kaggle competition was an estimated time of failure for each of these segments. I imported only 2.  
test1 <- read.csv("seg\_00030f\_downsampled.csv")  
colnames(test1) <- c("index", "acoustic")  
test2 <- read.csv("seg\_0012b5\_downsampled.csv")  
colnames(test2) <- c("index", "acoustic")  
head(test1)

## index acoustic  
## 1 1 -1  
## 2 2 14  
## 3 3 -1  
## 4 4 -15  
## 5 5 -5  
## 6 6 2

# ======= PART 1: EXPLORATORY DATA ANALYSIS =======

# From table view, depending on the rounding, the time\_to\_failure column may appear to have the same value for multiple rows. However, we can print with more significant digits to show the values are changing   
head(train)

## index acoustic time\_to\_failure  
## 1 1 4 1.469100  
## 2 2 0 1.469099  
## 3 3 5 1.469099  
## 4 4 8 1.469099  
## 5 5 -1 1.469099  
## 6 6 4 1.469098

print(train$time\_to\_failure[1:5], digits=10)

## [1] 1.469099710 1.469099435 1.469099160 1.469098885 1.469098610

# We can also calculate our sampling interval easily from the dataset. The difference in "time\_to\_failure" between each pair of points is constant. This is demonstrated below.  
sampling\_int <- train$time\_to\_failure[1] - train$time\_to\_failure[2]  
sampling\_int

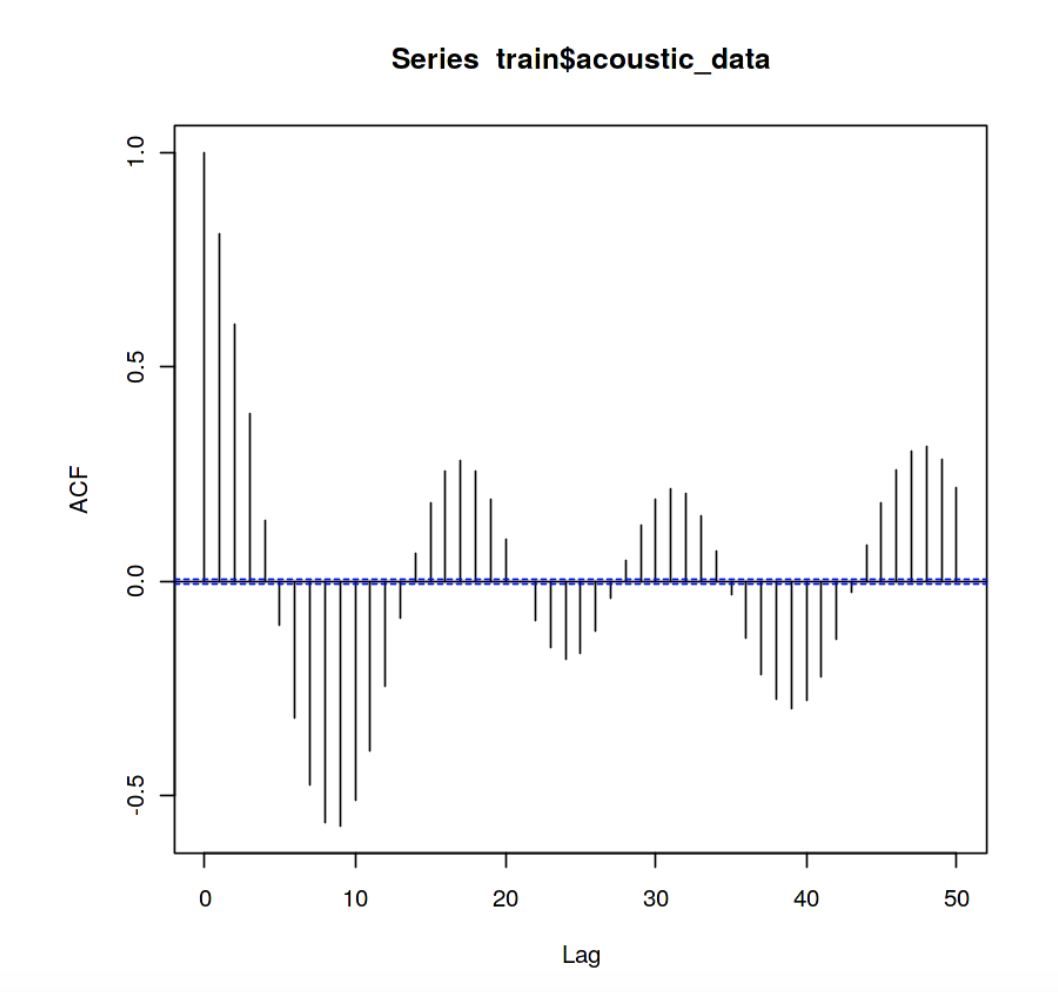
## [1] 2.75e-07

sampling\_int2 <- train$time\_to\_failure[2] - train$time\_to\_failure[3]  
sampling\_int2

## [1] 2.75e-07

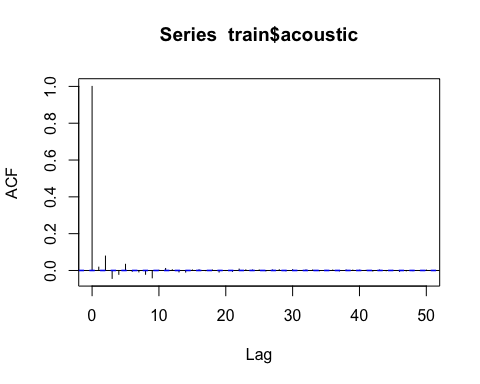
### Autocorrelation

# This is code used to plot the full data set, but to avoid importing the large file, I will include an image instead  
# acf(train$acoustic\_data, lag.max= 50)  
knitr::include\_graphics('ACF\_img.png')



The data set before down sampling (lowering the sampling interval by taking only one of every few rows) is beautifully autocorrelated for short lags. The alternating positive and negative correlation with increasing lag may be due to vibrations effects in the experimental setup for taking seismic measurements in a laboratory.

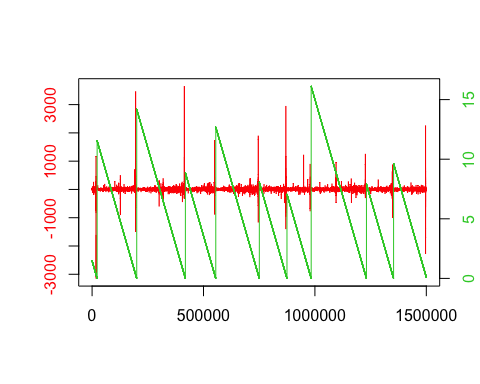
# This line plots our downsampled data autocorrelation  
acf(train$acoustic, lag.max= 50)



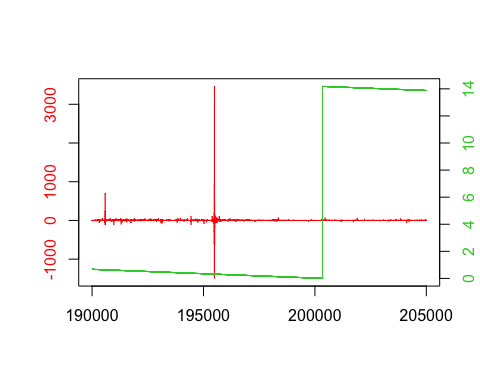
By downsampling the autocorrelation pattern seen above have almost completely disappeared. Therefore, I do not expect autocorrelation to be a significant predictor in the model to predict time of ground failure, made from the downsampled data set. However, this may be a more significant factor if the full 9 GB dataset was used.

### Plotting the waveform & failure cycles

# First we plot the full downsampled train dataset  
plot\_wave(train$index, train$acoustic, train$time\_to\_failure)



# With the time of ground failure labeled in this seismic dataset, we know which big signals are earthquakes and which are just noise.   
  
# Now we zoom in on one earthquake  
plot\_wave(train$index[190000:205000], train$acoustic[190000:205000], train$time\_to\_failure[190000:205000])

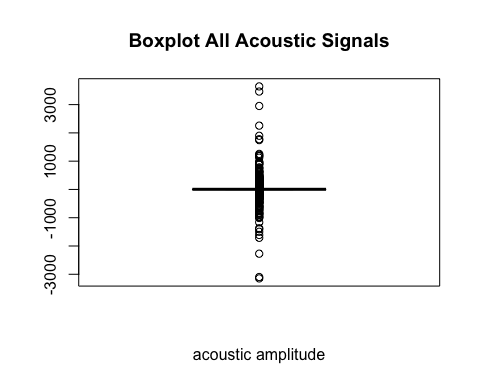


There is a clear lag time between the measured signal the the recorded ground failure. Our model should be able to place the time of ground failure a short time after the large amplitude signal.

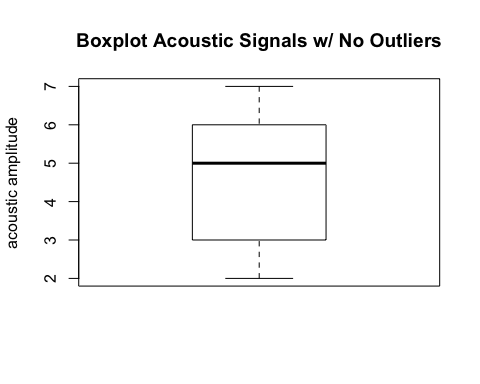
### Data distribution

We can look at the distribution of the acoustic signals to better understand how we can differentiate noise and earthquakes. Understanding how to differentiate the two can guide feature generation.

# The boxplots shows that the IQR includes a very small range. We can conclude that earthquakes are large outliers in our waveform data, which is useful for our analysis.  
boxplot(train$acoustic, main="Boxplot All Acoustic Signals", xlab="acoustic amplitude")



# Plotting only the interquartile range, we see the distribution of non-earthquake signals is almost Gaussian. However, it the non-earthquake signals are not centered on zero, so we will apply a normalization.  
q1 <- quantile(train$acoustic)[2]  
q3 <- quantile(train$acoustic)[4]  
boxplot(train$acoustic[train$acoustic >= q1 & train$acoustic <= q3], ylab = "acoustic amplitude", main="Boxplot Acoustic Signals w/ No Outliers")



### Data Normalization

Because the mean of the acoustic data is not zero, we should normalize this column before further analysis.

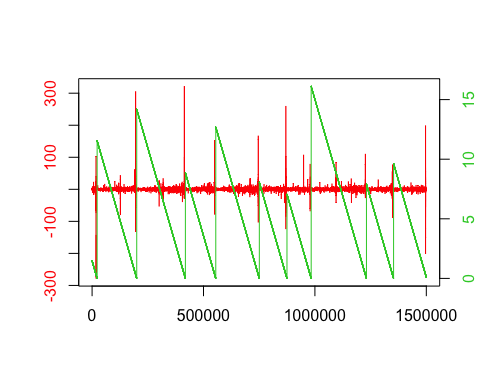
# First we make a copy of the train data to scale, so we can compare the two  
train\_scaled <- train  
train\_scaled$acoustic <- scale(train$acoustic)  
# Summary of before and after normalization for comparison  
summary(train$acoustic)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3146.000 2.000 5.000 4.566 7.000 3644.000

summary(train\_scaled$acoustic)

## V1   
## Min. :-278.0641   
## 1st Qu.: -0.2265   
## Median : 0.0383   
## Mean : 0.0000   
## 3rd Qu.: 0.2148   
## Max. : 321.2108

# Good! We still have large values where the earthquakes should be given the plotted time of failture, but have normalized our data, so the mean is now zero.  
plot\_wave(train\_scaled$index, train\_scaled$acoustic, train\_scaled$time\_to\_failure)

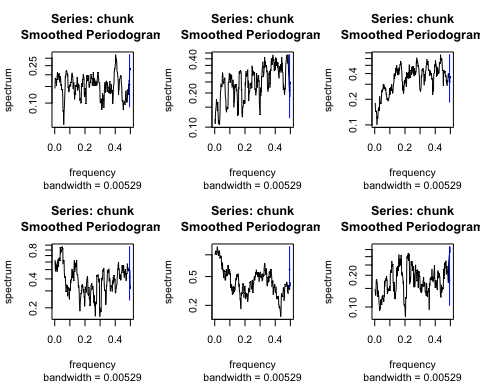


# Now that we have compared the data before and after normalization and like the results, we can overwrite our train data.  
train <- train\_scaled

### Periodogram

The periodogram plot shows the contributions of the individual frequencies to the time series. Significant patterns in the periodogram would suggest that further investigation into frequency as a predictor for earthquakes could be beneficial.

# The function spec.pgram() calculates the periodogram using a FFT, and optionally smooths the result with a series of modified Daniell smoother  
  
# First we plot an estimate of the spectral density of the acoustic signal for individual segments of the data. The smoothing kernel used is a daniell kernel that takes a moving average.  
kernel <- kernel("daniell", m = 5)  
acoustic\_ts <- ts(train$acoustic, frequency = 1/sampling\_int)  
seg\_train <- split(acoustic\_ts, (seq(nrow(train))-1) %/% 600)   
# Plot spectrums for a sample of segments  
par(mfrow = c(2,3))  
for (chunk in seg\_train[1:6]){  
 seg\_pgram <- spec.pgram(chunk, kernel, demean = TRUE, plot = FALSE)  
 plot(seg\_pgram)  
}

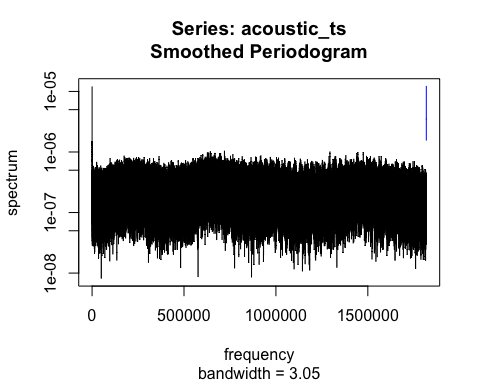


These periodograms contain a lot of spikes that make it difficult to analyze. This will only be more confusing if we plot the entire time series. The periodogram does not become smoother as the length of the time series increases but just includes more spikes packed closer together. Therefore smoothing would be beneficial before we plot the periodogram of the whole time series. Because of this I switch to a kernel that is able to smooth more. The kernel selected is a 5 by 5 daniell kernel, which takes a moving average of a moving average.

library("TTR")

## Warning: package 'TTR' was built under R version 3.6.2

kernel <- kernel("daniell", c(5, 5))  
full\_pgram <- spec.pgram(acoustic\_ts, kernel,spans=5, demean = TRUE, plot = FALSE, na.action = na.exclude)  
plot(full\_pgram)

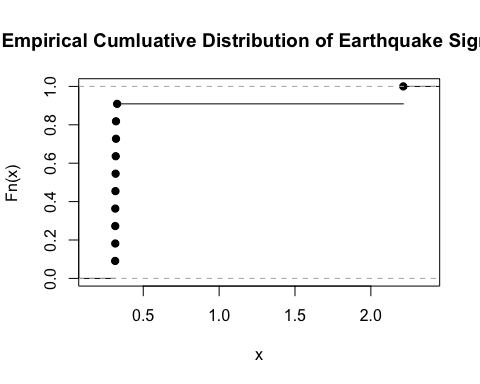


These results are not promising for using spectral density/ frequency as predictors for time of ground failure. The plots from individual segments do not show any trend. The lack of periodicity in this data is also evident in the plot of the entire time series, which appears as a solid mass, with a flat top. There are peaks at nearly every frequency of around the same height. The quasi-periodic trend of earthquake stick-slip behavior is likely not predictable by the frequency.

### Cumulative Distribution Plot

How close are large acoustic signals to ground failure?? By looking at the waveform plot, we can conclude that all earthquakes produce values greater than 100, and most well over 50. To answer this question we will look at the time to failure data for all signals that are >100. Let’s look further into the distribution of only large signals, which we attributed to laboratory produced earthquakes.

plot(ecdf(train$time\_to\_failure[train$acoustic> 100]), main = 'Empirical Cumluative Distribution of Earthquake Signals')

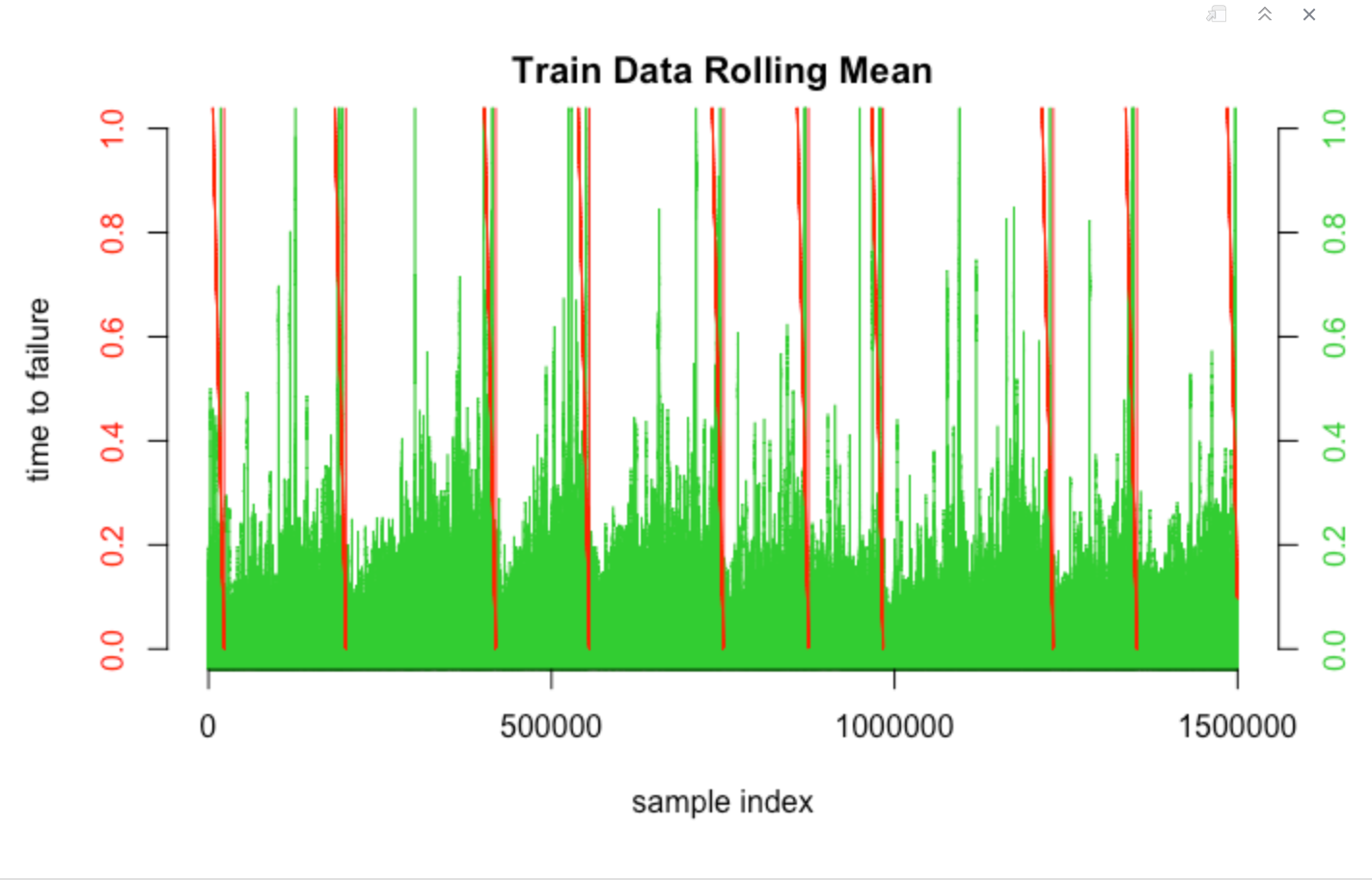


mean(train$time\_to\_failure[train$acoustic> 100])

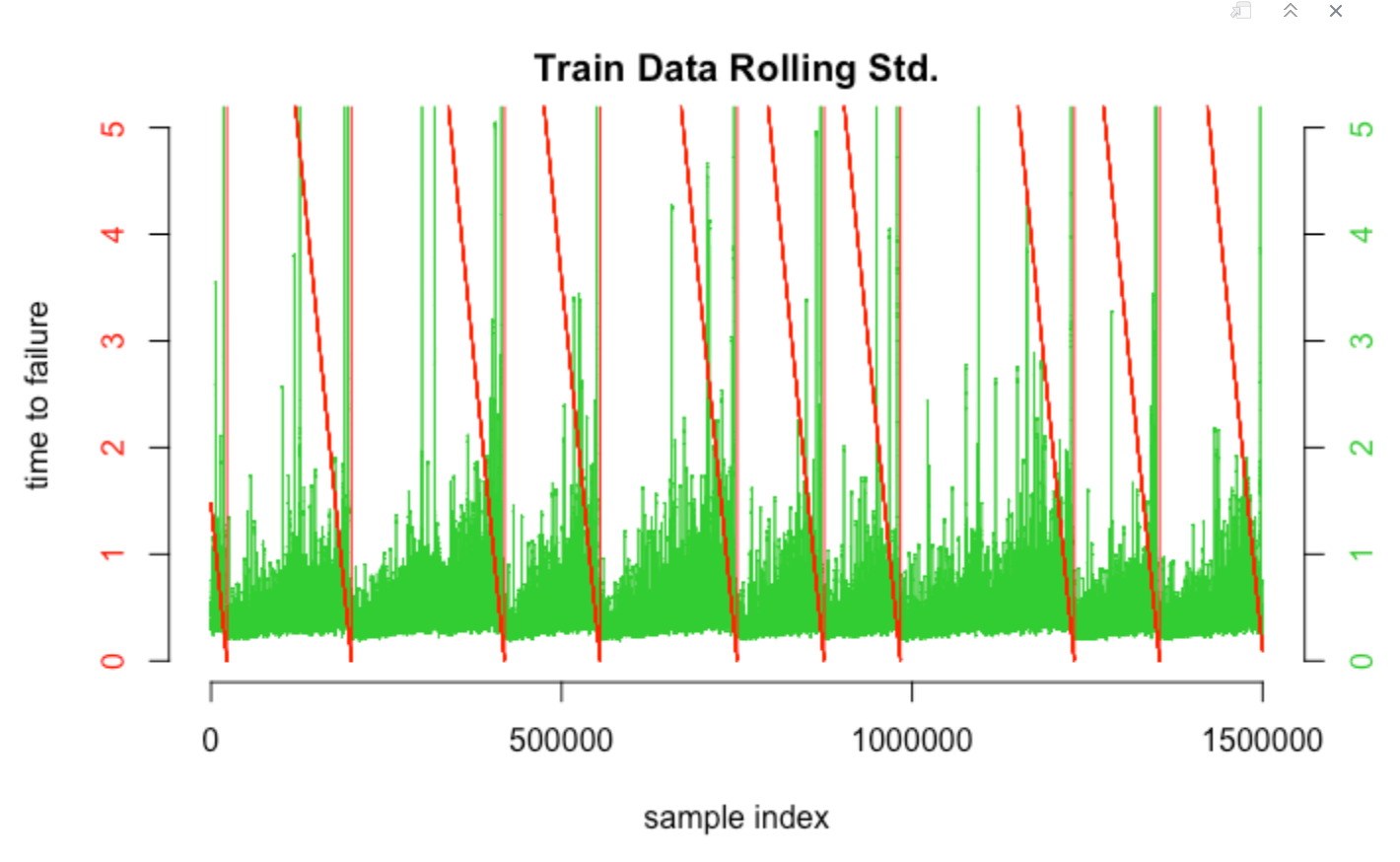
## [1] 0.4905788

The cumulative distribution plot of earthquake signals shows that they are concentrated at less than a second before ground failure, with a mean value of 0.49 seconds. About 85-90% of large signals occur within .5 seconds of ground failure. All large signals are within 1 second of ground failure. The occurance of large signals may be a strong predictor of earthquakes, and should be considered in feature generation. ### Rolling mean and standard deviation Before we generate features, let’s look at one last potential predictor. Because the signal is greatest closest to earthquake occurrence, a rolling average of the acoustic signal should correlate with the time\_to\_failure

# We will also calculate a rolling standard deviation and append it to the dataframe  
# The rolling window size is small enough that we can get most of the signal in a rolling average, not averaging in the earthquake produced acoustic signal.  
 # library(zoo)  
 #   
 # window <- 50  
 # padding <- rep(NA, window - 1)  
 # train$rollingavg <- c(padding, rollapply(train$acoustic,window, mean))  
 # window <- 100  
 # padding <- rep(NA, window - 1)  
 # train$rollingstd <- c(padding, rollapply(train$acoustic,window, sd))  
  
# We plot only small signals (acoustic<100), in order to look at how the earthquake building might be reflected in the waveform.  
 # plot.new()  
 # plot.window(xlim=range(train$index), ylim=c(0,1))  
 # title(main="Train Data Rolling Mean", xlab="sample index", ylab="time to failure",)  
 # lines(train$index[train$acoustic<100], train$rollingavg[train$acoustic<100], col="limegreen")  
 # lines(train$index[train$acoustic<100], train$time\_to\_failure[train$acoustic<100], col="red")  
 # axis(4, col.axis="limegreen", ylim=range(train$rollingavg), )  
 # axis(2, col.axis="red")  
 # axis(1)  
   
 # plot.new()  
 # plot.window(xlim=range(train$index), ylim=c(0,5))  
 # title(main="Train Data Rolling Std.", xlab="sample index", ylab="time to failure",)  
 # lines(train$index[train$acoustic<100], train$rollingstd[train$acoustic<100], col="limegreen")  
 # lines(train$index[train$acoustic<100], train$time\_to\_failure[train$acoustic<100], col="red")  
 # axis(4, col.axis="limegreen", ylim=range(train$rollingstd), )  
 # axis(2, col.axis="red")  
 # axis(1)  
  
  
# Because this calculation takes a long time, I imported the plots it produced as images instead of running the code in this file.   
knitr::include\_graphics('rollingmean.png')



knitr::include\_graphics('rollingstd.png')



This step is computationally intensive, but is the most promising feature for detection based on EDA. Therefore, calculating the rolling mean & rolling std for each segment in our dataframe may produce better results. Rolling standard deviation and rolling mean are expected to be important predictors

There are signs of an earthquake building, even without averaging in earthquakes (large, acoustic>100) signals. This is promising for trying to predict time of ground failure from the seismic waveform alone. We know this is true because in the rolling mean plot below, the rolling mean builds slightly toward the time of earthquake. This is made evident by the cone shape of the rolling average (lime green), growing toward the right, before each occurrence of an earthquake. Therefore, there is stronger signal noise before an earthquake in this data set. # ======== PART 2: FEATURE GENERATION ======== We next need to generate some features in order to create a model that predicts time of ground failure.

# First, we need to import packages and define some functions before we start feature generation in preparation for making a model.  
library(dplyr)

## Warning: package 'dplyr' was built under R version 3.6.2

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(kStatistics)

## Warning: package 'kStatistics' was built under R version 3.6.2

library(entropy)  
library(zoo)

## Warning: package 'zoo' was built under R version 3.6.2

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

gm\_mean = function(x, na.rm=TRUE){  
 exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x))  
}  
  
 add\_trend\_feature = function(x, y, abs\_values=FALSE){  
 if (abs\_values == TRUE){  
 x = abs(x)}  
 p = coef(lm(c(y)~c(x), na.action=na.exclude))  
 p[2]  
 }

### Feature Generation Function

This is a function to generate 36 different features. Most features included are based on the data exploration in the previous sections. At first I create some basic features, including mean, min, max, and quantiles, but by aggregating these features they can become more complex.

# Define the function  
generate\_features = function(chunk, summary\_df) {  
 window <- 35  
 padding <- rep(NA, window - 1)  
 rollingavg <- c(padding, rollapply(chunk$acoustic,window, mean))  
 rollingstd <- c(padding, rollapply(chunk$acoustic,window, sd))  
 autocorr = acf(chunk$acoustic, lag= 100, plot = FALSE, na.action = na.pass)  
 seg\_min = min(chunk$acoustic, na.rm=TRUE)  
 seg\_max = max(chunk$acoustic, na.rm=TRUE)  
 # Suppress Warnings is used because data contains a few NaNs.  
 suppressWarnings(  
 summary\_df <- rbind(summary\_df,   
 c(  
 seg\_min,  
 seg\_max,  
 seg\_05 = quantile(chunk$acoustic, 0.05, na.rm=TRUE),  
 seg\_50 = quantile(chunk$acoustic, 0.50, na.rm=TRUE),  
 seg\_95 = quantile(chunk$acoustic, 0.95, na.rm=TRUE),  
 seg\_mean = mean(chunk$acoustic, na.rm = TRUE),  
 seg\_sd = sd(chunk$acoustic, na.rm = TRUE),  
 seg\_range = (seg\_max - seg\_min),  
 max\_min\_ratio = abs(seg\_max) / abs(seg\_min),  
 seg\_big\_count35 = length(chunk$acoustic[chunk$acoustic > 35]),  
 seg\_big\_count50 = length(chunk$acoustic[chunk$acoustic > 50]),  
 seg\_big\_count75 = length(chunk$acoustic[chunk$acoustic > 75]),  
 seg\_sum = sum(chunk$acoustic, na.rm=TRUE),  
 mean\_change\_abs = mean(diff(chunk$acoustic, na.rm=TRUE)),  
 harmonic\_mean = 1/mean(1/abs(chunk$acoustic[chunk$acoustic>0]), na.rm=TRUE),  
 geometric\_mean = gm\_mean(chunk$acoustic, na.rm=TRUE),  
 seg\_variance=nKS(2, chunk$acoustic),  
 seg\_skewness = nKS(3, chunk$acoustic)/sqrt(nKS(2, chunk$acoustic))^3 ,  
 seg\_kurtosis= nKS(4, chunk$acoustic)/nKS(2, chunk$acoustic)^2 + 3 ,  
 meanAD = mad(chunk$acoustic, na.rm = TRUE),  
 autocorr\_1 = autocorr$acf[1],  
 autocorr\_10 = autocorr$acf[10],  
 autocorr\_100 = autocorr$acf[100],  
 entropy = entropy(chunk$acoustic),  
 slope = add\_trend\_feature(chunk$acoustic, chunk$index, abs\_values = FALSE),  
 slope\_abs = add\_trend\_feature(chunk$acoustic, chunk$index, abs\_values = TRUE),  
 rollingavg\_min = min(rollingavg, na.rm=TRUE),  
 rollingavg\_max = max(rollingavg, na.rm=TRUE),  
 rollingavg\_avg = mean(rollingavg, na.rm=TRUE),  
 rollingavg\_q05 = quantile(rollingavg, 0.05, na.rm=TRUE),  
 rollingavg\_q95 = quantile(rollingavg, 0.95, na.rm=TRUE),  
 rollingstd\_min = min(rollingstd, na.rm=TRUE),  
 rollingstd\_max = max(rollingstd, na.rm=TRUE),  
 rollingstd\_avg = mean(rollingstd, na.rm=TRUE),  
 rollingstd\_q05 = quantile(rollingstd, 0.05, na.rm=TRUE),  
 rollingstd\_q95 = quantile(rollingstd, 0.95, na.rm=TRUE)  
 )))  
   
}  
headers\_x <- c("seg\_min", "seg\_max", "seg\_05", "seg\_50", "seg\_95", "seg\_mean", "seg\_sd", "seg\_range", 'max\_min\_ratio', 'seg\_big\_count35','seg\_big\_count50','seg\_big\_count75', 'seg\_sum','mean\_change\_abs', 'harmonic\_mean', 'geometric\_mean', 'seg\_variance', 'seg\_skewness','seg\_kurtosis', 'meanAD','autocorr\_1', 'autocorr\_10', 'autocorr\_100', 'entropy',"slope", "slope\_abs", "rollingavg\_min", "rollingavg\_max", "rollingavg\_avg", "rollingavg\_q05","rollingavg\_q95","rollingstd\_min", "rollingstd\_max", "rollingstd\_avg", "rollingstd\_q05","rollingstd\_q95", "y")

For the remainder of our generated features, we can segment the data into chunks in order to create some features that generalize sections of the continuous time series. These features will be used to train our model. I break up feature generation into segments of 600 rows. Why 600? Test segments are 600 rows each with my chosen rate of downsampling. A chunk size of 600 rows iterates through the 1,500,000 row train dataframe, 0.04% at a time.

# Segmenting the train data and intializing dataframes to store the features produced from the segmented data  
seg\_train <- split(train, (seq(nrow(train))-1) %/% 600)   
train\_features <- data.frame(matrix(ncol = 36, nrow = 0))  
train\_y <- data.frame(matrix(ncol = 1, nrow = 0))

Now we apply the feature generation function, iterating through the segmented train data

# Generating features on our train data  
 # for (chunk in seg\_train){  
 # train\_features <- generate\_features(chunk, train\_features)  
 # train\_y <- rbind(train\_y, min(chunk$time\_to\_failure))  
 # }  
 # train\_features <- cbind(train\_features,train\_y)   
 # headers\_x <- c("seg\_min", "seg\_max", "seg\_05", "seg\_50", "seg\_95", "seg\_mean", "seg\_sd", "seg\_range", 'max\_min\_ratio', 'seg\_big\_count35','seg\_big\_count50','seg\_big\_count75', 'seg\_sum','mean\_change\_abs', 'harmonic\_mean', 'geometric\_mean', 'seg\_variance', 'seg\_skewness','seg\_kurtosis', 'meanAD','autocorr\_1', 'autocorr\_10', 'autocorr\_100', 'entropy',"slope", "slope\_abs", "rollingavg\_min", "rollingavg\_max", "rollingavg\_avg", "rollingavg\_q05","rollingavg\_q95","rollingstd\_min", "rollingstd\_max", "rollingstd\_avg", "rollingstd\_q05","rollingstd\_q95", "y")  
 # colnames(train\_features) <- headers\_x  
 #   
 # #write.csv(train\_features, "train\_features.csv")  
  
train\_features <- read.csv("train\_features.csv")[-1]  
head(train\_features)

## seg\_min seg\_max seg\_05 seg\_50 seg\_95 seg\_mean seg\_sd  
## 1 -2.168171 4.627730 -0.4912607 0.03828999 0.7443576 0.05991331 0.4669779  
## 2 -4.109857 3.127336 -0.5795192 0.03828999 0.6560992 0.03770160 0.5011886  
## 3 -3.403790 8.599360 -0.6677776 0.03828999 0.6560992 0.02784607 0.6256694  
## 4 -7.904971 4.274696 -0.5795192 0.03828999 0.7443576 0.04343840 0.6356123  
## 5 -10.640983 7.098966 -0.6677776 0.03828999 0.6560992 0.04049645 0.7660709  
## 6 -1.550362 3.215594 -0.4956737 0.03828999 0.6560992 0.03196480 0.4205745  
## seg\_range max\_min\_ratio seg\_big\_count35 seg\_big\_count50 seg\_big\_count75  
## 1 6.795901 2.1343929 0 0 0  
## 2 7.237193 0.7609354 0 0 0  
## 3 12.003150 2.5264075 0 0 0  
## 4 12.179667 0.5407605 0 0 0  
## 5 17.739949 0.6671345 0 0 0  
## 6 4.765957 2.0740923 0 0 0  
## seg\_sum mean\_change\_abs harmonic\_mean geometric\_mean seg\_variance  
## 1 35.94799 -0.000294686 0.1119650 0.4161782 0.2180683  
## 2 22.62096 0.000884058 0.1188273 0.4116882 0.2511900  
## 3 16.70764 0.000442029 0.1073915 0.3825594 0.3914622  
## 4 26.06304 -0.000147343 0.1201463 0.4385085 0.4040030  
## 5 24.29787 -0.000294686 0.1175703 0.4188594 0.5868646  
## 6 19.17888 0.000884058 0.1236751 0.4254909 0.1768830  
## seg\_skewness seg\_kurtosis meanAD autocorr\_1 autocorr\_10 autocorr\_100  
## 1 2.3595656 25.12110 0.261704 1 -0.043920084 -0.008432381  
## 2 -1.3727004 18.70644 0.261704 1 -0.023606390 -0.009697556  
## 3 4.2227367 68.29180 0.261704 1 0.005731254 0.030336486  
## 4 -3.1384235 53.61819 0.392556 1 -0.004383992 -0.181967022  
## 5 -3.1618694 84.37129 0.261704 1 -0.036658676 -0.007624027  
## 6 0.9489634 11.27079 0.392556 1 -0.023209561 -0.008938455  
## entropy slope slope\_abs rollingavg\_min rollingavg\_max rollingavg\_avg  
## 1 12.55356 -27.0651168 -89.849603 -0.1281402 0.2526319 0.05526894  
## 2 18.28858 -12.0677057 -12.680504 -0.1533569 0.2652403 0.03517132  
## 3 21.08149 13.2363684 9.736412 -0.1659653 0.2223719 0.02138232  
## 4 16.80216 7.6594090 -20.275283 -0.3676989 0.3232387 0.04186310  
## 5 17.17960 4.8341912 7.567422 -0.2340504 0.7166193 0.04282098  
## 6 19.38392 0.8992204 -5.662845 -0.1079669 0.2248936 0.03581733  
## rollingavg\_q05 rollingavg\_q95 rollingstd\_min rollingstd\_max rollingstd\_avg  
## 1 -0.07266350 0.1713081 0.1984434 1.1537603 0.3956748  
## 2 -0.11301022 0.1341135 0.2023630 0.7969423 0.4838097  
## 3 -0.08022851 0.1612214 0.1730716 1.6514103 0.5307327  
## 4 -0.26683210 0.2299369 0.1703651 1.5152495 0.5504300  
## 5 -0.14011817 0.5804491 0.1967540 1.9235825 0.5925549  
## 6 -0.06698974 0.1542868 0.2278919 0.7292086 0.4142506  
## rollingstd\_q05 rollingstd\_q95 y  
## 1 0.2304624 0.6721734 1.430797  
## 2 0.2635364 0.7729017 1.391499  
## 3 0.2432483 1.5944226 1.353196  
## 4 0.2375329 1.4941276 1.313798  
## 5 0.2298634 1.8652928 1.274400  
## 6 0.2693284 0.6151224 1.236097

And we repeat this process to generate test features

# Generating features on labeled test data  
 # seg\_test <- split(test, (seq(nrow(test))-1) %/% 600)   
 # test\_features <- data.frame(matrix(ncol = 36, nrow = 0))  
 # test\_y <- data.frame(matrix(ncol = 1, nrow = 0))  
 #   
 # for (chunk in seg\_test){  
 # test\_features <- generate\_features(chunk, test\_features)  
 # test\_y <- rbind(test\_y, min(chunk$time\_to\_failure))  
 # }  
 # test\_features <- cbind(test\_features,test\_y)   
 # colnames(test\_features) <- headers\_x  
 # test\_features  
   
 #write.csv(test\_features, "test\_features.csv")  
  
test\_features<- read.csv("test\_features.csv")[-1]  
head(test\_features)

## seg\_min seg\_max seg\_05 seg\_50 seg\_95 seg\_mean seg\_sd seg\_range  
## 1 -18 48 -1 5.0 10 4.826667 4.386573 66  
## 2 -15 40 -1 5.0 9 4.656667 3.678497 55  
## 3 -17 24 -1 5.0 10 4.751667 3.389072 41  
## 4 -23 45 0 5.0 10 4.796667 4.538316 68  
## 5 -29 62 -1 4.5 9 4.421667 4.088405 91  
## 6 -45 17 0 5.0 10 4.633333 4.054169 62  
## max\_min\_ratio seg\_big\_count35 seg\_big\_count50 seg\_big\_count75 seg\_sum  
## 1 2.6666667 1 0 0 2896  
## 2 2.6666667 1 0 0 2794  
## 3 1.4117647 0 0 0 2851  
## 4 1.9565217 2 0 0 2878  
## 5 2.1379310 1 1 0 2653  
## 6 0.3777778 0 0 0 2780  
## mean\_change\_abs harmonic\_mean geometric\_mean seg\_variance seg\_skewness  
## 1 0.005008347 3.768038 4.010081 19.24203 2.098839876  
## 2 0.010016694 3.791809 4.003486 13.53134 1.161020979  
## 3 0.001669449 3.813271 4.030513 11.48581 0.002314528  
## 4 -0.001669449 3.584448 3.973077 20.59632 2.359078575  
## 5 0.001669449 3.421556 3.754666 16.71506 3.767208363  
## 6 0.003338898 3.755384 4.086359 16.43628 -3.622292326  
## seg\_kurtosis meanAD autocorr\_1 autocorr\_10 autocorr\_100 entropy  
## 1 24.483628 2.9652 1 -0.01701581 0.14712732 6.336850  
## 2 20.272714 2.9652 1 -0.04145049 0.01748739 6.366944  
## 3 7.899948 2.9652 1 -0.03503974 -0.06176629 6.309772  
## 4 28.422867 2.9652 1 0.05273630 -0.01273140 6.321573  
## 5 75.559976 2.2239 1 0.04063376 -0.03880495 6.354080  
## 6 43.296660 2.9652 1 -0.00208322 -0.01456968 6.493729  
## slope slope\_abs rollingavg\_min rollingavg\_max rollingavg\_avg  
## 1 -0.2031064911 -0.07298349 2.600000 6.342857 4.845684  
## 2 -0.6805446002 2.41629034 3.285714 6.057143 4.608682  
## 3 -0.0009447677 1.05901721 3.514286 5.771429 4.755931  
## 4 -2.6878884933 -1.50592573 2.085714 7.800000 4.760575  
## 5 -4.8753443883 -2.62194113 2.657143 6.657143 4.403635  
## 6 1.1092565005 1.01845038 3.171429 5.914286 4.631600  
## rollingavg\_q05 rollingavg\_q95 rollingstd\_min rollingstd\_max rollingstd\_avg  
## 1 3.628571 5.935714 1.989468 8.457308 4.130287  
## 2 3.578571 5.600000 1.914123 9.111540 3.482443  
## 3 4.092857 5.314286 2.210745 6.245671 3.332192  
## 4 2.742857 7.142857 1.962677 10.855197 4.105357  
## 5 3.178571 5.935714 2.085565 10.648076 3.597480  
## 6 3.721429 5.564286 2.051849 9.494182 3.744447  
## rollingstd\_q05 rollingstd\_q95 y  
## 1 2.460698 8.065697 5.899860e-02  
## 2 2.327403 8.253434 2.069580e-02  
## 3 2.436763 5.792524 4.956083e-04  
## 4 2.300621 10.211132 1.136900e+01  
## 5 2.261479 10.207562 1.133070e+01  
## 6 2.268341 9.143140 1.129130e+01

# ======= PART 3: MODELING WITH RANDOM FORESTS =======

Why apply random forests for this particular problem? Randoms forests works well with a large number of highly correlated features. In the previous section we generate features that are highly related by series of nested calculations. Because earthquake prediction is an unsolved area of research, it is best to create a lot of features that may or may not be helpful for the model, and use a method that reduces the amount of predictors. Random forests achieves this by ensemble learning, or averaging smaller trees together into one larger tree. There are multiple benefits of using a random forest model. We can build a deep tree, including multiple predictors, by ensemble learning. By averaging smaller trees together there is a good variance to bias tradeoff. Because 37 predictors may or may not be included in each tree, this method limits both bias and variance. Random forest models are also fast and perform well on large datasets, which is important in this usage. Finally, random forests have easily interpretted results.

# This block makes a random forest model from the features created from the train data  
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

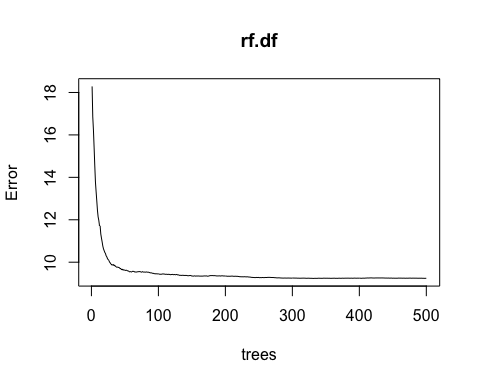
##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

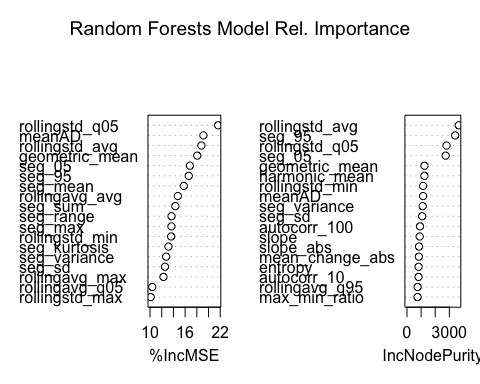
set.seed(101)  
rf.df = randomForest(y~., data=train\_features, ntree=500,  
 importance=TRUE,)  
rf.df

##   
## Call:  
## randomForest(formula = y ~ ., data = train\_features, ntree = 500, importance = TRUE, )   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 12  
##   
## Mean of squared residuals: 9.241306  
## % Var explained: 35.18

# We can plot how the error improves with more branches. More branches lower error, as would be expected. Averaging more trees improves the model, although the benefits are much less at 300-500 trees.  
plot(rf.df)

 ### Variable Importance

## Relative importance of variables: higher values mean more important  
# This plot shows the percentage of MSE explained by the whole model for each predictors, only showing the top 1/2 most important predictors  
varImpPlot(rf.df, sort=TRUE, n.var= nrow(rf.df$importance) / 2, main="Random Forests Model Rel. Importance")



round(importance(rf.df), 2)

## %IncMSE IncNodePurity  
## seg\_min 9.27 573.17  
## seg\_max 13.66 582.10  
## seg\_05 16.81 2737.33  
## seg\_50 3.35 77.29  
## seg\_95 16.62 3410.67  
## seg\_mean 15.77 686.69  
## seg\_sd 12.56 1073.39  
## seg\_range 13.68 547.23  
## max\_min\_ratio 9.13 746.21  
## seg\_big\_count35 0.90 1.67  
## seg\_big\_count50 0.78 1.57  
## seg\_big\_count75 -0.76 0.68  
## seg\_sum 14.32 685.46  
## mean\_change\_abs 2.21 838.28  
## harmonic\_mean 7.14 1234.13  
## geometric\_mean 18.07 1237.47  
## seg\_variance 12.76 1112.30  
## seg\_skewness 8.31 647.85  
## seg\_kurtosis 13.15 687.65  
## meanAD 19.12 1137.33  
## autocorr\_1 0.00 0.00  
## autocorr\_10 -1.75 817.15  
## autocorr\_100 -1.19 922.50  
## entropy 6.28 819.28  
## slope 1.51 896.76  
## slope\_abs -1.24 853.92  
## rollingavg\_min 8.06 617.78  
## rollingavg\_max 12.30 740.08  
## rollingavg\_avg 14.76 706.74  
## rollingavg\_q05 10.38 641.53  
## rollingavg\_q95 7.20 751.28  
## rollingstd\_min 13.61 1137.86  
## rollingstd\_max 10.11 649.91  
## rollingstd\_avg 18.78 3662.92  
## rollingstd\_q05 21.63 2800.21  
## rollingstd\_q95 9.75 632.64

### Analysis of Feature Importance

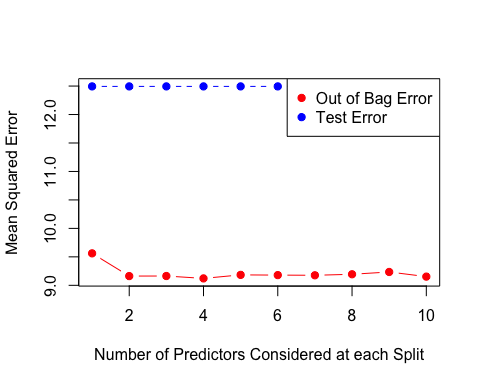
This plot shows that the most important predictors are rolling standard deviation, the 95th quantile, meanAD, and the geometric mean. I expected the rolling mean and rolling sum to be important predictors because of the increasing noise toward earthquake events. I am not surprised that the mean absolute deviation is also an important predictor because average distance of datapoints from the mean is a good measure of noise. The 95th quantile is a measure of how large the largest signals are. A high 95th quantile indicates an earhtquake would have just occurred, making the time to failure probably high. I was expecting the absolute slope to be a more important predictor, as it would show an increasing trend in the signal, similar to a rolling mean. However, this did not appear in the top half of most useful features for our model. However, the geometric mean was a top predictor. Geometric mean calculations use a product of the components instead of their sum to find the average. This statistic is often used in calculation of growth rates.

### Test Error Calculations

oob.err<-double(10)  
test.err<-double(10)  
for(mtry in 1:10) {  
 rf=randomForest(y ~ . , data = train\_features, mtry=mtry, ntree=500)   
 oob.err[mtry] = rf$mse[400] #Error of all Trees fitted  
   
 pred<-predict(rf.df,test\_features) #Predictions on Test Set for each Tree  
 test.err[mtry]= with(test\_features, mean( (y - pred)^2)) #Mean Squared Test Error  
   
 cat(mtry," ")  
 }

## 1 2 3 4 5 6 7 8 9 10

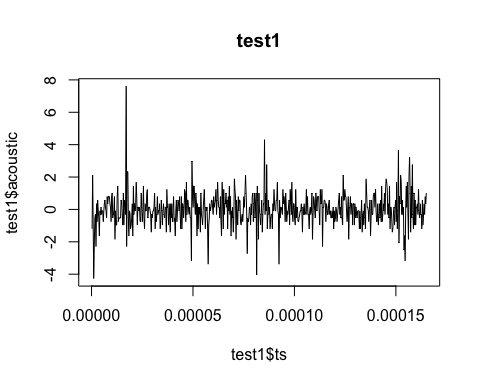
matplot(1:mtry , cbind(oob.err,test.err), pch=19 , col=c("red","blue"),type="b",ylab="Mean Squared Error",xlab="Number of Predictors Considered at each Split")  
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blue"))



The MSE if not far off between the test error and out of bag error and the two erros follow a similar trend for increasing predictors, which is ideal for the model. However, an MSE of 10-12 is not great. An ideal MSE is zero for a perfect model, although MSE values are relative, and there is not set “good” standard for MSE.

### Application of Model To Test Segments

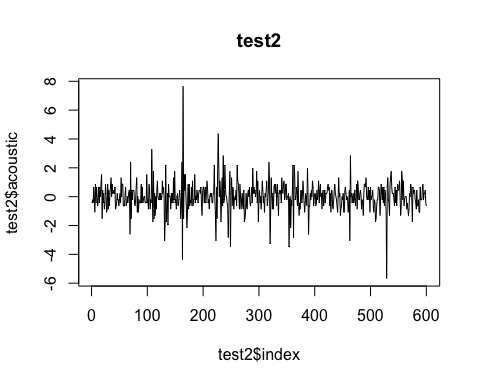
test\_seg\_fxn = function(segment, features){  
 features <- generate\_features(segment, features)  
 colnames(features) <- c("seg\_min", "seg\_max", "seg\_05", "seg\_50", "seg\_95", "seg\_mean", "seg\_sd", "seg\_range", 'max\_min\_ratio', 'seg\_big\_count35','seg\_big\_count50','seg\_big\_count75', 'seg\_sum','mean\_change\_abs', 'harmonic\_mean', 'geometric\_mean', 'seg\_variance', 'seg\_skewness','seg\_kurtosis', 'meanAD','autocorr\_1', 'autocorr\_10', 'autocorr\_100', 'entropy',"slope", "slope\_abs", "rollingavg\_min", "rollingavg\_max", "rollingavg\_avg", "rollingavg\_q05","rollingavg\_q95","rollingstd\_min", "rollingstd\_max", "rollingstd\_avg", "rollingstd\_q05","rollingstd\_q95")  
   
 return(features)   
}  
  
# Random forest on unlabeled test segments provided with Kaggle competition  
  
test1$acoustic <- scale(test1$acoustic)  
test1$ts <- test1$index \* sampling\_int  
plot(test1$ts, test1$acoustic, type="l", main="test1")



test1\_features <- data.frame(matrix(ncol = 37, nrow = 0))  
test1\_features <- test\_seg\_fxn(test1, test1\_features)  
pred<-predict(rf.df,test1\_features)  
print(pred)

## 1   
## 6.07869

test2$acoustic <- scale(test2$acoustic)  
plot(test2$index, test2$acoustic, type="l", main="test2")



test\_features2 <- data.frame(matrix(ncol = 37, nrow = 0))  
test\_features2<- test\_seg\_fxn(test2, test\_features2)  
pred<-predict(rf.df,test\_features2)  
print(pred)

## 1   
## 5.781491

# Model Conclusions:

The purpose of this project was to explore the possibility of predicting earthquakes. I wanted to share my results of data exploration on a smaller dataset, and show a model may be applied, and the reasoning behind my choices in handling the data. I think the test error of this model is reasonable, given the downsampling. This model would perform much better if it were not downsampled and the strong autocorrelation seen in the entire dataset could be used as a predictor. However, the top predictor in all top scoring models on this dataset was some form of rolling standard deviation, which was preserved despite working with a smaller dataset.

The “pred” result is the predicted time of ground failure. Unfortunately, this can not be plotted over the segment waveform because the sampling interval of the segments is not provided. However, we can compare these results to other Kaggle users submissions, who used different methods of feature generation and different models on the same test segments. The top scored responses had values between 3-7 seconds for “test1” and 4-6 seconds for “test2”. I think the results using a random forest model stack up to other models, and because of that, it is a viable method for attempting to predict earthquakes from the amplitude waveform alone.