PREDICTION OF FAILURE ON ROLLER BEARINGS USING VIBRATION ANALYSIS

Executive Summary

This project is specific to mechanical engineering. I am a mechanical engineer and have always wanted to learn this technique so have chosen this subject for the project to add my professional skill set.

Readers/reviewers who do not have engineering background might struggle to understand the concept of vibration analysis. I will try to explain every step clearly and provide satisfactory reasoning for each step.

The purpose of the project is to imitate the role of a vibration analysis expert who has the skills and experience to assess vibration data which is collected with sensors and make reasonable predictions whether the bearings of the interest would fail or not in certain time period. Roller bearings are the most problematic component of rotating equipment to which special maintenance attention are given to prevent dramatic consequences.

Three types of prediction algorithms have been used in this project. Those are Random Forest, KNN and Stochastic Gradient Boosting. These three algorithms are compared in terms of overall accuracy in the Results section. It turns out that Random Forest model provides the highest accuracy.

Table of Content

Section 1 - Introduction

The data that is used in this project has been generated by the NSF I/UCR Center for Intelligent Maintenance Systems (www.imscenter.net) It is stored in the Prognostics Data Repository hosted by NASA

The data was collected in a run-to-failure experiment performed in 2006. It consists four roller bearings on a loaded shaft that rotates at a constant speed. Three datasets were generated in the experiment. In set 1, two high precision accelerometers were installed on each bearing and measurements were taken with all 8 accelerometers whereas one accelerometer was used in datasets 2 and 3. Each dataset is formatted in individual files, each containing a 1-second vibration signal snapshot that was recorded every other 10 minutes. Each file consists of 20,480 points with a sampling rate set of 20 kHz. The file name indicates when the data was collected. Each record (row) in the data file is a data point.

In this project, we will focus on the dataset 1 since two accelerometers on each bearing provide more complete data.

The dataset is loaded to the Github. You will need to save it to your computer if you want to run this code.

Section 1.1 - What is Vibration Analysis?

Vibration analysis is defined as a process for measuring the vibration levels and frequencies of machinery or a component and analyzing how healthy the machines and their components are. It all starts with using an accelerometer to measure vibration. Anytime a piece of machinery is running, it vibrates. An accelerometer attached to the machine generates a voltage signal that corresponds to the amount and frequency of vibration the machine usually how many times per second or minute the vibration occurs.

All data collected from the accelerometer goes directly into a data collector (software) which records the signal as either amplitude vs. time (known as time waveform), amplitude vs. frequency (known as Fast Fourier Transfor spectrum) or both. All of this data is analyzed by computer program algorithms, which in turn is analyzed by engineers or trained vibration analysts to determine the health of the machine and identify possible impending problems like looseness, unbalance, misalignment, lubrication issues and more. The focus of this project is on roller bearings, not on another component and complete machine.

Section 2 - Analysis

The approach followed in this project is Supervised Learning, meaning that we have prior knowledge of what the output values for our samples should be. A vibration analysis expert has reviewed the data and drawn the conclusions below about the failure patterns for the bearings included in the experiment:

Time frame is given in this format: YYYY.MM.DD.HR.MN.SN

Bearing 1

- Early signs of failure are detected between these time frame: 2003.10.22.12.06.24 - 2003.10.23.09.14.13 (this is due to the initial break-in period of the bearing)
- Normal operation of the bearing occurred between these time frame: 2013.1023.09.24.13 - 2003.11.19.21.06.07
- A possible failure is suspected between these time frame: 2003.11.19.21.16.07 - 2003.11.24.20.47.32
- More obvious failure is observed between these time frame: 2003.11.24.20.57.32 - 2003.11.25.23.39.56
- Bearing 1 did not fail afterall even if it showed strong signs of possible failure

Bearing 2

- Early signs of failure: 2003.10.22.12.06.24 - 2003.11.01.21.41.44 (due to break-in period)
- Normal operation: 2003.11.01.21.51.44 - 2003.11.24.01.01.24
- Possible failure: 2003.11.24.01.11.24 - 2003.11.25.10.47.32
- More obvious failure: 2003.11.25.10.57.32 - 2003.11.25.23.39.56
- Bearing 2 did not fail either even if it showed strong signs of possible failure

Bearing 3

- Early signs of failure: 2003.10.22.12.06.24 - 2003.11.01.21.41.44
- Normal operation: 2003.11.01.21.51.44 - 2003.11.22.09.16.56
- Possible failure: 2003.11.22.09.26.56 - 2003.11.25.10.47.32
- Inner race failure: 2003.11.25.10.57.32 - 2003.11.25.23.39.56
- Bearing 3 failed, it was inner race failure.

Bearing 4

- Early signs of failure: 2003.10.22.12.06.24 - 2003.10.29.21.39.46
- Normal operation: 2003.10.29.21.49.46 - 2003.11.15.05.08.46
- Possible failure: 2003.11.15.05.18.46 - 2003.11.18.19.12.30
- Rolling element failure: 2003.11.19.09.06.09 - 2003.11.22.17.36.56
- Stage 2 failure: 2003.11.22.17.46.56 - 2003.11.25.23.39.56

The purpose of this project is to develop a machine learning algorithm that will imitate the function of a vibration analysis expert to predict bearing failure before it happens.

Section 2.1 - Data Wrangling

The following packages are needed in my script. The lines of code below will download the packages from cran:

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(entropy)) install.packages("entropy", repos = "http://cran.us.r-project.org")
if(!require(quantmod)) install.packages("quantmod", repos = "http://cran.us.r-project.org")
if(!require(pracma)) install.packages("pracma", repos = "http://cran.us.r-project.org")
if(!require(ggcorrplot)) install.packages("ggcorrplot", repos = "http://cran.us.r-project.org")
```

The folder that contains the files with vibration analysis data is stored in Github and has also been uploaded to Edx. There are 2156 files in this folder. Each file contains 20480 lines of vibration measurement. Let us took at the titles of first a few files:

```r
files <- list.files("./1st_test")
head(files)
```

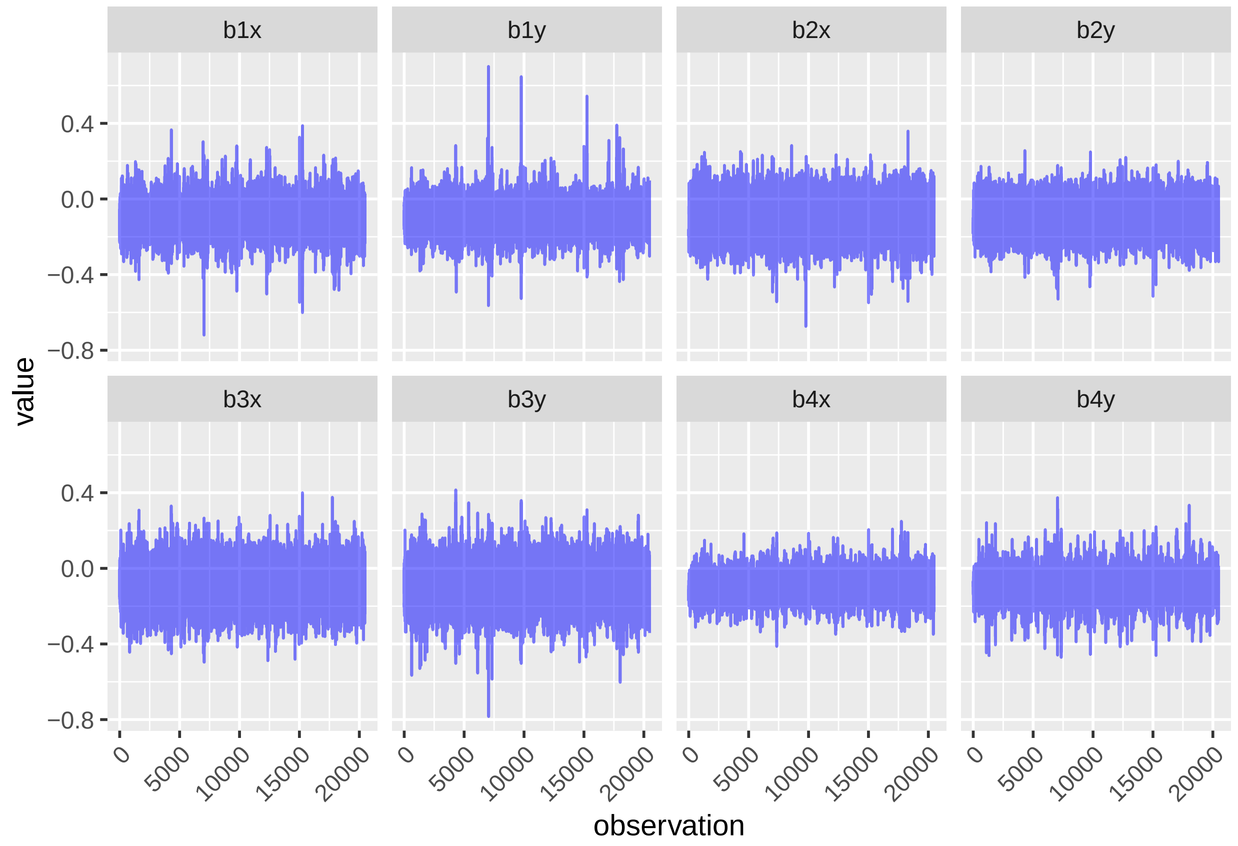
```
## [1] "2003.10.22.12.06.24" "2003.10.22.12.09.13" "2003.10.22.12.14.13"
## [4] "2003.10.22.12.19.13" "2003.10.22.12.24.13" "2003.10.22.12.29.13"
```

Remember these files have been titled in this format: YYYY.MM.DD.HR.MM.SS
Let's have a look at the first file:

```
##      b1x    b1y    b2x    b2y    b3x    b3y    b4x    b4y
## 1 -0.022 -0.039 -0.183 -0.054 -0.105 -0.134 -0.129 -0.142
## 2 -0.105 -0.017 -0.164 -0.183 -0.049  0.029 -0.115 -0.122
## 3 -0.183 -0.098 -0.195 -0.125 -0.005 -0.007 -0.171 -0.071
## 4 -0.178 -0.161 -0.159 -0.178 -0.100 -0.115 -0.112 -0.078
## 5 -0.208 -0.129 -0.261 -0.098 -0.151 -0.205 -0.063 -0.066
## 6 -0.232 -0.061 -0.281 -0.125  0.046 -0.088 -0.078 -0.078
```

As you can see in the dataframe, there are two sets of vibration reading for each bearing (4 bearing total x 2 readings for each). x and y represent the locations of the accelerometers on a bearing. One accelerometer measures vibration occurring vertically, the other one horizontally.

Now let us plot the values taken by the each accelerometer versus time in the first file. The variable "file" is matched to the first file. There will be 8 different graphs total:



You will notice that the mean vibration value on each graph is negative. This is due to the accelerometers having negative voltage when measuring and negative values do not mean problem. Furthermore, the y-axis vibration on bearing 1 (second figure from the top left corner) seems to have outlines but they do appear at regular-ish intervals.

Section 2.2 - Analyzing the Vibration Data

IMPORTANT NOTE: In Section 2.2, the raw vibration data (measurements taken by accelerometers) is processed to obtain a dataset which will be used in prediction models. There-

fore, the reader/reviewer is free to skip the Section 2.2 and continue on Section 2.3.

Working with the raw vibration data is not the best approach for failure prediction. Therefore, some certain analysis techniques are needed for further analysis. These are

- Time-Domain Analysis (Peak, Average, Peak-to-Peak, RMS, Skewness, Kurtosis, Entropy)
- Frequency-Domain Analysis (Fast Fourier Transform (FFT) Spectrum Analysis)
- Bearing Defect Frequencies
- Post Processing

Section 2.2.1 - Time-Domain Anaylsis Features

In this section, features of time-domain analysis of a sinusoidal signal are provided in the following lines of codes:

```
# This function will read the file and tabulates the vibration readings
# in easy-to-read format:
read_file <- function(filename)
{
  raw_vib <- read.table(paste("./1st_test/", filename, sep=""),
                        col.names = paste("b", rep(1:4, each = 2), c("x", "y"), sep = ""),
                        sep = "\t")
  return(raw_vib)
}

# This function calculates the Root Mean Square (RMS) of a sinusoidal signal
rms <- function(x){sqrt(mean(x^2))}

# This function calculates the peak value in a sinusoidal signal:
max_abs <- function(x){max(abs(x))}

# This function calculates the Peak-to-Peak (positive peak + negative peak) value
# in a sinusoidal signal:
p2p <- function(x){abs(max(x)) + abs(min(x))}

# This function calculates the Shannon entropy (cut signal to 500 bins)
# of a sinusoidal signal:
entropy <- function(x){entropy::entropy(table(cut(x, 500)))}

# This function calculates the Autoregressive (AR) coefficients of a sinusoidal signal:
ar_coeffs <- function(raw_signal, ar_order)
{
  # Get the AR coeffs for each of the 8 signals and return them in a vector
  ar_coeffs <- apply(raw_signal, 2, function(x){ar(x, aic = F, order.max = ar_order)$ar}) %>%
    as.vector(.)

  # Generate a vector of names, to append them to rest of the features later on
  ar_coeffs_names <- paste("b", rep(1:4, each = 2), c("x", "y"),".ar", rep(1:ar_order, each = ar_order)
    matrix(., ncol = ar_order)
  ar_coeffs_names <- (as.vector(t(ar_coeffs_names)))

  # Return a 1 by n dataframe as the result
  out <- t(data.frame(ar_coeffs, row.names = ar_coeffs_names))
```

```r
  return(out)
}


# Now let us combine all the functions defined previously into a single function
# for Time-Domain Analysis a sinusoidal signal:

time_features <- function(raw_vib_file)
{
  # Extract statistical moments and rms, abs and peak-to-peak values
  time_feats <- raw_vib_file %>% summarise_all(funs(mean,
                                                    sd,
                                                    skewness,
                                                    kurtosis,
                                                    entropy,
                                                    rms,
                                                    max_abs,
                                                    p2p))


  # Calculate quantiles
  quants <- apply(raw_vib_file, 2, quantile)[2:4, ] # Drop the 0% and 100% quantiles
  col.names <- paste(rep(colnames(quants), each = 3), rep(rownames(quants, 8)), sep = ".")
  quants <-quants %>% as.vector %>% data.frame %>% t
  colnames(quants) <- col.names

  # Extract AR model coefficients and bind all the features of the vibration signal together
  time_feats <- cbind(time_feats, quants, ar_coeffs(raw_vib_file, ar_order = 8))

  # Remove row names (appear due to the preprocessing done in the AR coeff function)
  rownames(time_feats) <- NULL

  # Return a 1 x n dataframe containing the extracted features per file
  return(time_feats)
}
```

Section 2.2.2 - Frequency-Domain Analysis Features (Fast Fourier Transform (FFT) Spectrum Analysis)

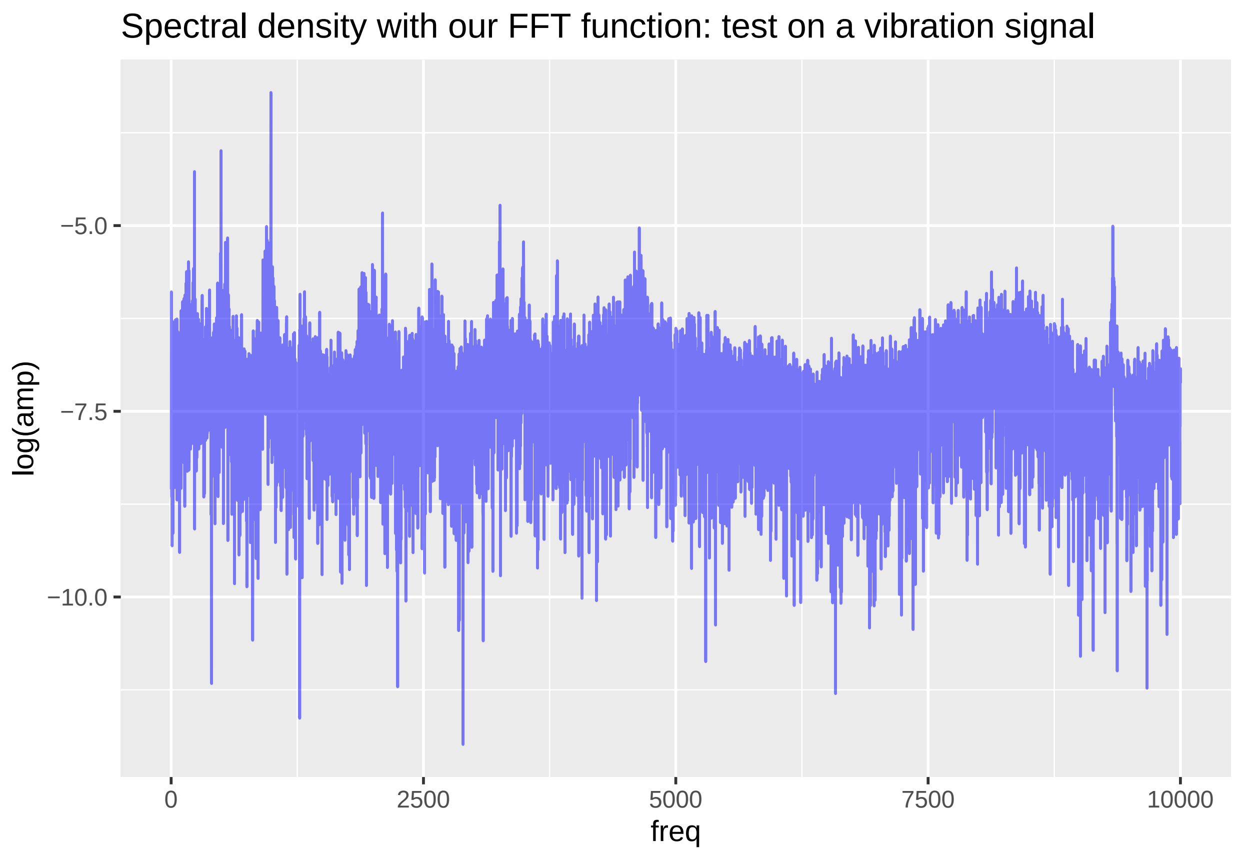
A method of viewing the vibration signal in a way that is more useful for analysis is to apply a Fast Fourier Transformation (FFT). In non-mathematical terms, this means that the signal is broken down into specific amplitudes at various component frequencies. An FFT spectrum is an incredibly useful tool. If a machinery problem exists, FFT spectra provide information to help determine the location of the problem, the cause of the problem and with trending, how long until the problem becomes critical.

Because we know that certain machinery problems occur at certain frequencies, we analyze the FFT spectrum by looking for amplitude changes in certain frequency ranges.

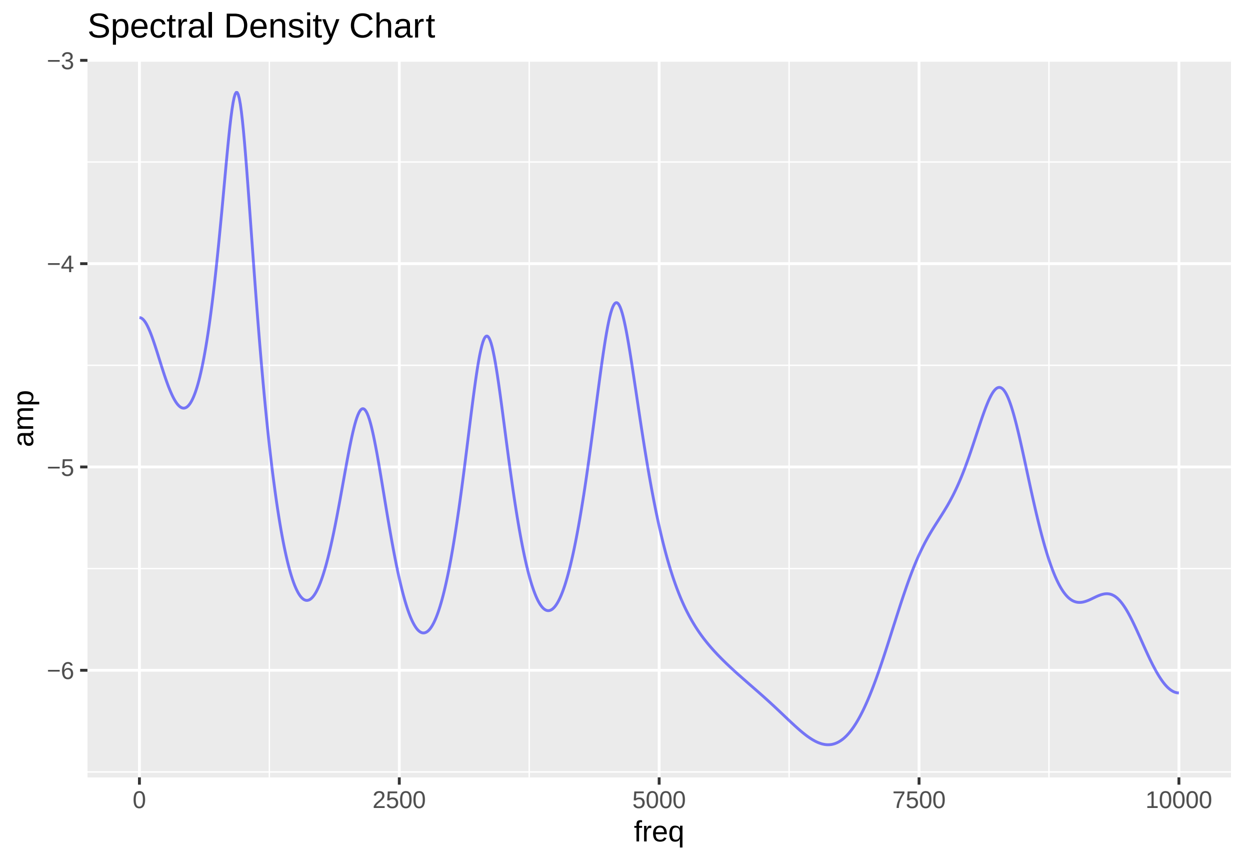
In this section, features of Frequency-Domain Analysis of a sinusoidal signal are provided in the following lines of codes:

```r
apply_FFT <- function(x, sampling_rate = 20000)
{
  # Center the signal first
  x <- x - mean(x)
  # Get the amplitudes
  amp <- fft(x)[1:(length(x) / 2)]%>% # Drop the second half - it's just a mirror of the first half
    Mod # Calculate the amplitude of the complex output of the previous line

  # Make a vector containing the frequencies
  freq <- seq(0, sampling_rate / 2, length.out = length(x) / 2)

  # and make a dataframe out of them (remove the dc term)
  fft_out <- data.frame("amp" = amp[-1] / length(amp), "freq" = freq[-1])

  return(fft_out)
}
```

And a quick test on a random file:



Looks about right (qualitatively), noisy but more or less as expected. R has a base function to approximate the spectral density of a stationary signal, by fitting an autoregressive model on it. This might be helpful as the expected result will be much less noisy. Let us try it out:



That is a nice result. The peaks are clearly defined and the result is less noisy overall. We will be using this function for the rest of the project. Before we move any further, we should calculate the characteristic frequencies of the bearings. We will be keeping an eye out on the FFT amplitude at these frequencies.

Section 2.2.3 - Bearing Defect Frequencies

A failure in one of the bearing components will produce a pronounced peak in a characteristic frequency of the frequency spectrum. This characteristic frequency will allow quick and easy identification of the failure. The four bearing defect frequencies are:

- Ball Pass Frequency Inner (BPFI)
- Ball Pass Frequency Outer (BPFO)
- Ball Spin Frequency (BSF)
- Fundamental Train Frequency (FTF)

The FTF defect will always occur at about 40 percent of running speed. Of the remaining three bearing defect frequencies BSF is always the lowest frequency. The ball pass–inner race frequency is always the highest.

The following lines of codes calculate the Bearing Defect Frequencies of the bearings that were used in the experiment:

```r
N <- 16 # No of rolling elements
n <- 2000 / 60 # Rotational speed [Hz]
Bd <- 0.331 # Rolling element diameter [in]
```

```r
Pd <- 2.815 # Pitch diameter [in]
phi <- 15.17 * pi / 180 # Contact diameter [rad]

# Get the frequencies
bearing_freqs <- list("BPFI" = N / 2 * (1 + Bd / Pd * cos(phi)) * n,
                      "BPFO" = N / 2 * (1 - Bd / Pd * cos(phi)) * n,
                      "BSF" = Pd / (2 * Bd) * (1 - (Bd / Pd * cos(phi)) ^2) * n,
                      "FTF" = 0.5 * (1 - Bd / Pd * cos(phi)) * n)

bearing_freqs

# Function to return a dataframe containing spectral density values at specific frequencies
get_spectrum <- function(signal, spectrum.points = 10000, AR.order = 100, sampling_rate = 20000)
{
  # Get spectral density
  spectrum <- spec.ar(x = signal - mean(signal), # Center the signal
                      n.freq = spectrum.points, # Generate 1000 points
                      order = AR.order,
                      plot = F) # Do not plot

  # Convert to a dataframe
  spectrum <- data.frame("freq" = seq(0, sampling_rate / 2, length.out = spectrum.points),
                         "amp" = log(spectrum$spec))

  return(spectrum)
}

# Function to do linear interpolation on the spectral density at a given frequency
interpolate_spectrum <- function(spectrum, f0)
{
  p1 <- spectrum[max(which(spectrum$freq <= f0)), ]
  p2 <- spectrum[min(which(spectrum$freq >= f0)), ]
  out <- (p2$amp - p1$amp) / (p2$freq - p1$freq) * (f0 - p1$freq) + p1$amp
  return(out)
}
```

Function to return spectral density values at the characteristic bearing frequencies

```r
get_spectra_at_char_freqs <- function(spectrum, bearing_frequencies)
{
  # Find the log-amplitude of the spectral density at the characteristic bearing frequencies
  spec_val_char_freqs <- sapply(bearing_frequencies, interpolate_spectrum, spectrum = spectrum) %>%
    as.data.frame %>%
    t

  # Unname rows (result of the preprocessing done in the previously called function)
  rownames(spec_val_char_freqs) <- NULL
  return(spec_val_char_freqs)
}
```

Next, let us write a function to return the top 10 frequencies, in terms of spectral density amplitude

```r
# Function to return the top n freqs (in terms of spectral content)
top_content_freqs <- function(spectrum, no_freqs)
{
  # Find the indices at which peaks occur
  peak_idx <- quantmod::findPeaks(spectrum$amp)

  # Isolate these instances, and get the top <no_freqs>
  peak_freqs <- spectrum[peak_idx, ] %>%
    arrange(desc(amp)) %>%
    head(., no_freqs) %>%
    select(freq) %>%
    t

  return(peak_freqs)
}
```

Now, a function to return the statistical and some other features from a spectrum:

```r
# Function to calculate integral based on the trapezoidal rule
trapz <- function(x, y)
{
  # Re-center y values to zero min
  y <- y + abs(min(y))

  # Calculate the area using the trapezoidal method by taking the average of
  # the "left" and "right" y-values.
  area <- sum(diff(x) * (head(y, -1) + tail(y, -1))) / 2

  return(area)
}

# Function to calculate the statistical moments of the spectrum
get_spectral_moments <- function(spectrum)
{
  f <- spectrum$freq
  s <- spectrum$amp + abs(min(spectrum$amp)) # Center to zero min

  fc <- trapz(x = f, y = s * f) / trapz(x = f, y = s) # Defined outside the list,
  # as it will be used within the list (vf) while the latter is being created
  feats <-list("fc" = fc, # frequency center
               "rmsf" = sqrt(trapz(x = f, y = s * f * f) / trapz(x = f, y = s)),
               "vf" = sqrt(trapz(x = f, y = (f - fc) ^ 2  * s) / trapz(x = f, y = s)),
               "sp_mean" = mean(spectrum$amp),
               "sp_sd" = sd(spectrum$amp),
               "sp_skew" = skewness(spectrum$amp),
               "sp_kurtosis" = kurtosis(spectrum$amp),
               "sp_entropy" = entropy(spectrum$amp),
               "power" = sum(exp(spectrum$amp))) # Power of the signal
                                                  # (sum of the FFT spectrum components)

  return(feats)
}
```

Next up, a function to split a spectrum into the three different frequency areas:

```r
# Function to split the spectrum into three different areas
split_spectrum <- function(spectrum)
{
  # Area below rotational speed of the shaft
  sub_spectrum <- spectrum %>%
    filter(freq < 2000 / 60) # Rotational speed of the shaft = 2000 rpm

  # Area between rotational speed of the shaft, up to ten times of it
  mid_spectrum <- spectrum %>%
    filter(freq >= 2000 / 60) %>%
    filter(freq < 10 * 2000 / 60)

  # Area above ten times the rotational speed of the shaft
  high_spectrum <- spectrum %>%
    filter(freq >= 10 * 2000 / 60)

  out <- list("low_spectr" = sub_spectrum,
              "mid_spectr" = mid_spectrum,
              "hi_spectr" = high_spectrum)

  return(out)
}
```

Wrapper for all the frequency features:

```r
frequency_features <- function(raw_vib_file, bearing_frequencies)
{
  # Get the spectra
  spectra <- apply(raw_vib_file, 2, get_spectrum)

  # Calculate spectral densities at the characteristic bearing frequencies
  bear_f_spectra <- sapply(spectra, get_spectra_at_char_freqs, bearing_frequencies) %>%
    as.vector %>%
    t %>%
    data.frame

  colnames(bear_f_spectra) <- paste(rep(colnames(raw_vib_file), each = 4),
                                    rep(c("BPFI", "BPFO", "BSF", "FTF"), 4), sep = ".")


  no_freqs <- 15 # Return top n freqs
  top_freqs <- sapply(spectra, top_content_freqs, no_freqs = no_freqs) %>%
    as.vector %>%
    t %>%
    data.frame

  colnames(top_freqs) <- paste(rep(colnames(raw_vib_file), each = no_freqs),
                               "freq", rep(1:no_freqs, 4), sep = ".")

  # Split the spectra into three frequecy areas
  spectra <- lapply(spectra, split_spectrum)

  # Convert the list of lists to list
  spectra <- unlist(spectra, recursive = F)

  # For the entire spectrum
  moments <- sapply(spectra, get_spectral_moments)
  col.names <- paste(rep(colnames(moments), each = 9), rep(rownames(moments), 4), sep = ".")
  moments <- moments %>%
    do.call(cbind, .) %>%
    as.vector %>% t %>%
    data.frame
  colnames(moments) <- col.names

  # Combine all
  freq_feats <- cbind(bear_f_spectra, top_freqs, moments)

  # Remove rownames (appear due to the preprocessing done in the AR coeff function)
  rownames(freq_feats) <- NULL

  # Return a 1 x n dataframe containing the extracted features per file
  return(freq_feats)
}
```

Section 2.2.4 - Post Processing

A small wrapper to combine time- and frequency- domain features together:

```
# Read a vibration file
calculate_features <- function(filename, bearing_frequencies)
{
  # Read the vibrations file
  vib_file <- read_file(filename)

  # Calculate the features
  feats <- cbind(time_features(vib_file),
                 frequency_features(vib_file, bearing_frequencies))

  # Return them (1 by n list)
  return(feats)
}
```

We will also need to assign the labels to the dataset. We will use the failure patters that are provided in Section 2 as labels:

```
labels <- list("b1.state" = c(rep("early", each = 151),
                              rep("suspect", each = 449),
                              rep("normal", each = 899),
                              rep("suspect", each = 599),
                              rep("imminent_failure", each = 58)),
               "b2.state" = c(rep("early", 500),
                              rep("normal", 1500),
                              rep("suspect", 120),
                              rep("imminent_failure", 36)),
               "b3.state" = c(rep("early", 500),
                              rep("normal", 1290),
                              rep("suspect", 330),
                              rep("inner_race_failure", 36)),
               "b4.state" = c(rep("early", 200),
                              rep("normal", 800),
                              rep("suspect", 435),
                              rep("inner_race_failure", 405),
                              rep("stage_2_failure", 316))) %>%
  data.frame
```

And finally, let us write a small function to perform a bit of post-processing on the dataset, to bring it into a format which is suitable for further analysis:

```
# Perform post processing on the column names
postprocess_dset <- function(data, bearing)
{
  # Split into individual bearing datasets
  bearing_dset <- data[, grepl(bearing, colnames(data))]
  colnames(bearing_dset) <- gsub(bearing, "", colnames(bearing_dset))
  colnames(bearing_dset) <- gsub(".state", "state", colnames(bearing_dset)) # leftovers not captured fr

  return(bearing_dset)
}
```

All done! Let us extract the features for the entire dataset and store them in a .csv file. Note that we do not necessairly need the filenames themselves as the dataset is already chronologically ordered due to the filename format.

```
# Function to read the vibration files, extract features, post-process the dataset, and write to an out
extract_features <- function(input_dir, output_file)
{
  # Extract the features
  features <- lapply(list.files(input_dir), calculate_features, bearing_freqs) %>%
    do.call("rbind", .)

  # Bind them with the labels
  dset <- cbind(features, labels)

  # Postprocess and write to .csv
  dset <- lapply(paste("b", rep(1:4, 1), sep = ""), postprocess_dset, data = dset) %>%
    do.call(rbind, .)


  return(write.csv(dset, file = output_file))
}
```

This process will take for a while if decide to run it:

```
extract_features("./1st_test", "processed_data.csv")
```

Section 2.3 - Exploratory Data Analysis on the Extracted Features

Let us start exploring the features we created:

```r
# Read in the data
data <- read.csv("processed_data.csv", row.names = 1, stringsAsFactors = T)

dim(data)
```

```
## [1] 8624  131
```

We have 8624 datapoints, 130 continuous independent variables and a nominal response.

Section 2.4 - Spliting the Dataset into Train and Validation Sets

Let us split the dataset into train and validation sets. Validation set will only be used to test the selected algorithm.

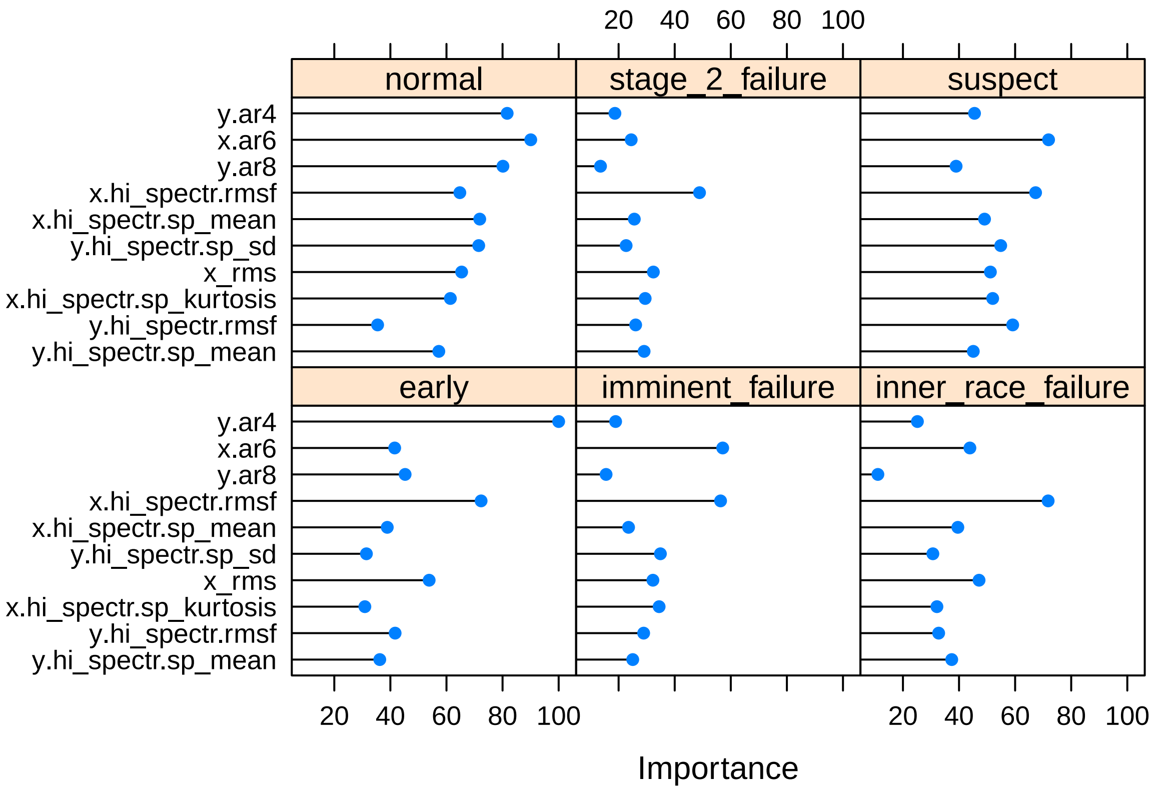
```r
set.seed(10)
train_idx <- createDataPartition(data$state, p = 0.7, list = F)
train <- data[train_idx, ]
validation <- data[-train_idx, ]
```

Section 2.5 - Feature Selection

Let's first assess predictor importance. We'll be using a model-based approach, based on a random forest classifier. The reason for choosing a model-based approach is that, being tied to model performance, it may be able to incorporate the correlation structure between the predictors into the importance calculation.

Let's train a random forest classifier on the training set:

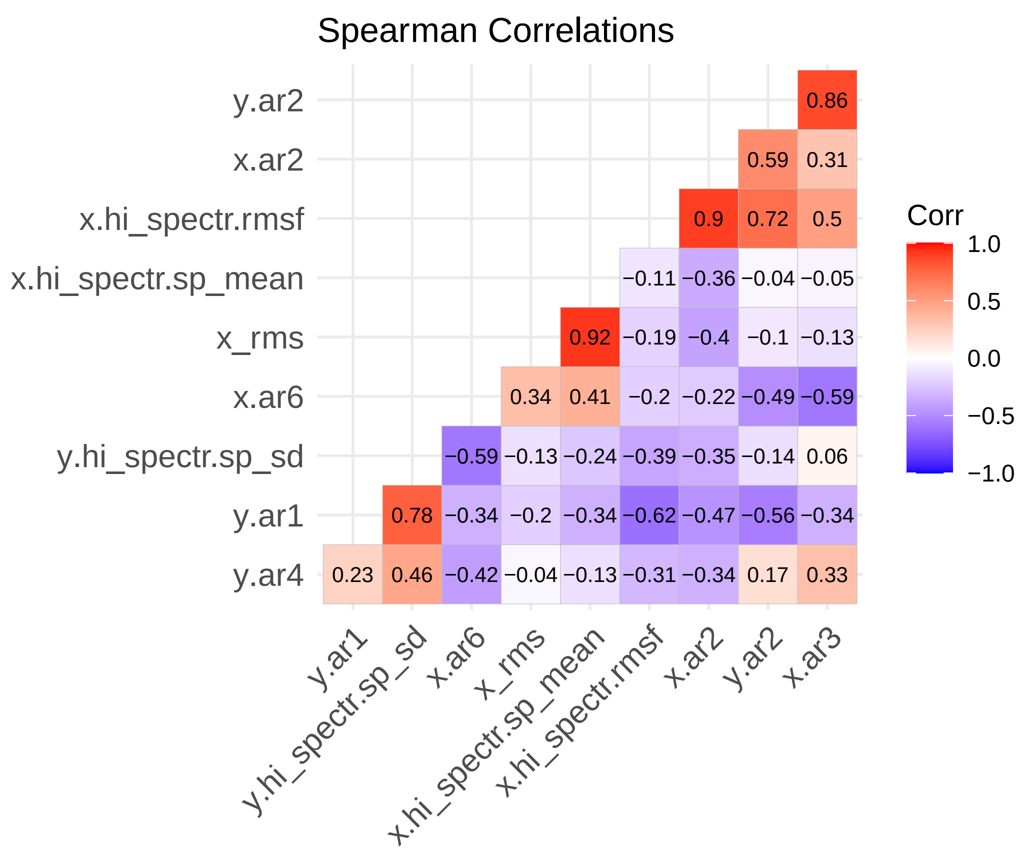
```r
set.seed(1)
# set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later
rf <- train(state~.,
            data = train,
            method = "rf",
            trControl = trainControl(method = "oob"),
            importance = T,
            verbose = F)

# and get the importance of each dependent variable:
rfImp <- varImp(rf, scale = T)
```



We can see that each predictor has different importance for each of the classes (reading the documentation of varImp, that is to be expected from tree-based algorithms). It is also interesting to note that vibration power levels at characteristic frequencies are not in the top 10 predictors.

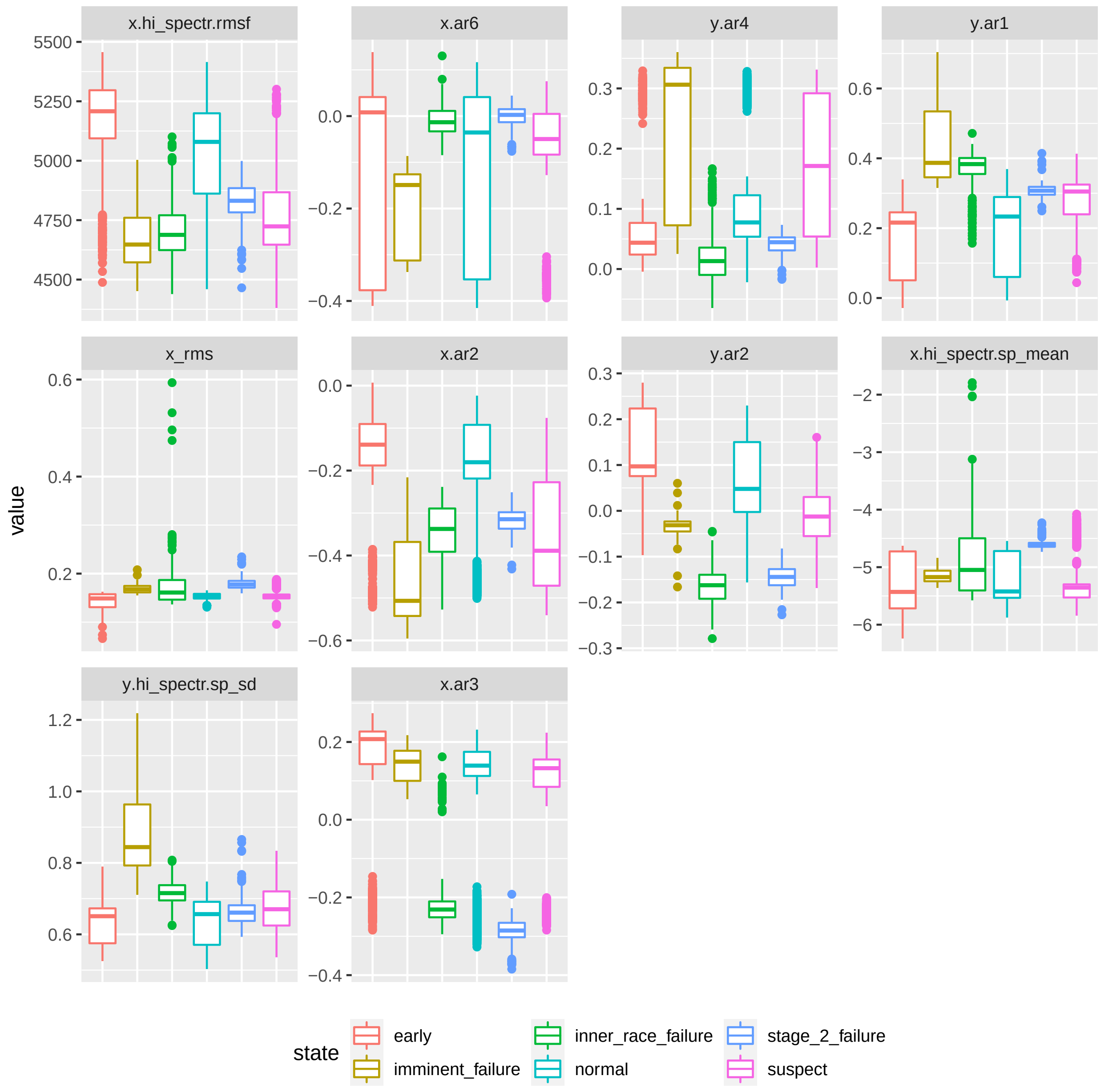
Anyway, let us isolate the top predictors and see how these are correlated:

```
n_predictors <- 10 # Get the top ten predictors

# Extract the top predictors from the dataset
top_predictors <- data.frame("predictor" = rownames(rfImp$importance)) %>%
  mutate(pred_power = rowMeans(rfImp$importance)) %>%
  arrange(desc(pred_power)) %>%
  head(n_predictors) %>%
  select(predictor) %>%
  as.list %>%
  unlist
```



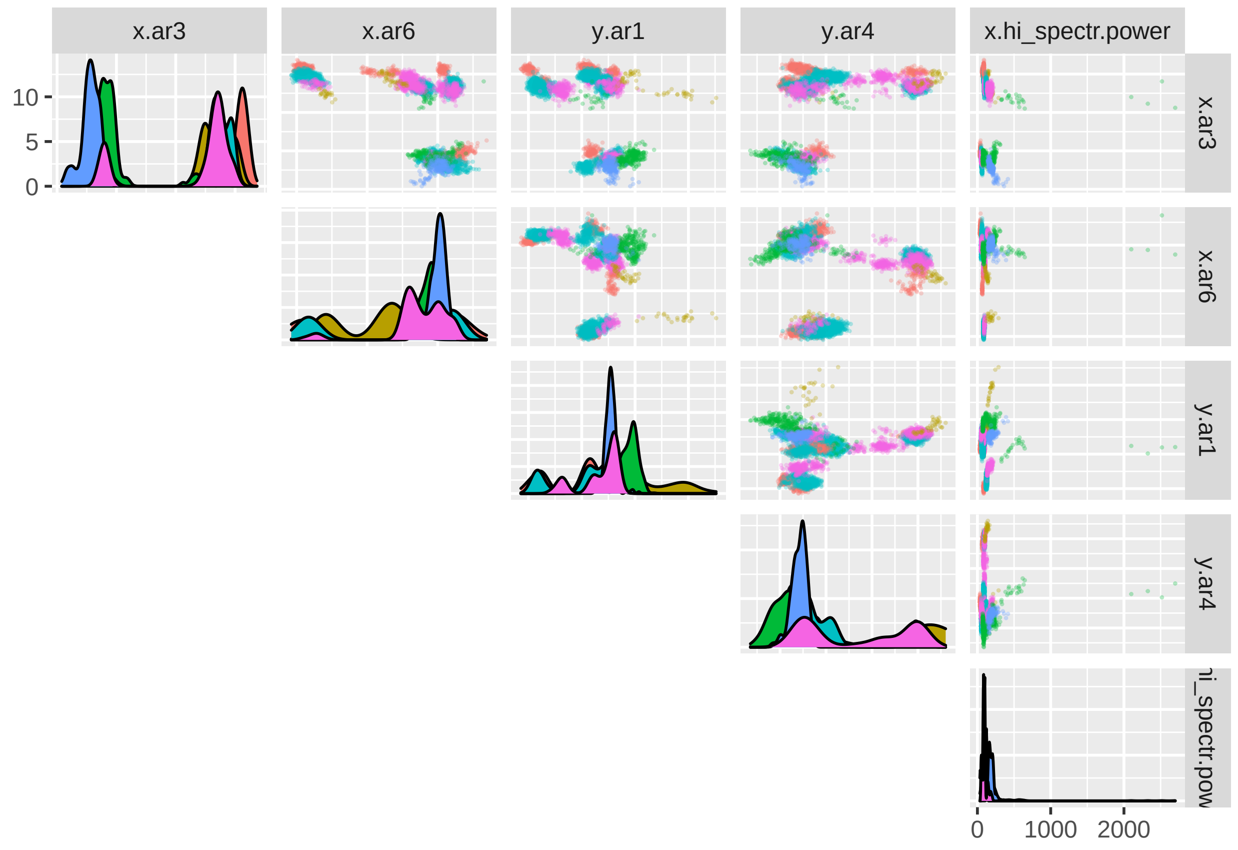
Highest correlation coefficient is 0.92. We have moderately correlated data to this point. Note that these are monotonic relations and not necessarily linear.

Let's make a boxplot to visualize the underlying distributions:



There are noticeable differences between groups for variables x_entropy, x.ar3 (inner race failure and stage 2 failure), x.ar6, y.ar1, y.ar4, x.ar5. Let's isolate these predictors and make a pair plot:

```
cols <- c("x.ar3","x.ar6","y.ar1","y.ar4","x.ar5")
```



Indeed, some clusters have started to emerge but nothing easily separable. Let us begin modeling and depending on the results, we might return to more advanced feature selection methods.

Section 2.6 - Predictive Modeling

Three different algorithms were selected for predictive modeling to choose the one with the best overall accuracy. These algorithms are Random Forest, KNN and Stochastic Gradient Boosting. We will start with KNN.
However, let us create training and test datasets out of train dataset and define the control parameter for each algorithm first.

Firstly, I will split the train dataset that was created earlier to another train and test datasets to train my candidate algorithms:

```r
set.seed(2)
trainIndex <- createDataPartition(train$state, p = .7, list = F)
train.set <- train[trainIndex, ]
test.set <- train[-trainIndex, ]


# Train control (10 fold cross validation)
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
```

Section 2.6.1 - KNN

Now let us start with KNN model:

```r
set.seed(3)
knn <- train(x = train.set[, cols],
              y = train.set$state,
              method = "knn",
              trControl = control,
              tuneGrid = data.frame(k = c(3,5,7)))

y_hat_knn <- predict(knn, test.set[, cols])
confusionMatrix(y_hat_knn, as.factor(test.set$state))$overall["Accuracy"]
```

We obtain Overall Accuracy of 0.782 and Kappa of 0.654. Prevalence among the classes are within reasonable levels. Accuracy is pretty low with KNN model.

Section 2.6.2 - Stochastic Gradient Boosting

And now stochastic gradient boosting:

```r
set.seed(4)
gbm_mdl <- train(x = train.set[, cols],
                 y = train.set$state,
                 method = "gbm",
                 verbose = F,
                 metric = "Kappa",
                 trControl = control)

y_hat_gbm <- predict(gbm_mdl, test.set[, cols])
confusionMatrix(y_hat_gbm, as.factor(test.set$state))$overall["Accuracy"]
```

We obtain Overall Accuracy of 0.9242 and Kappa of 0.8819. Prevalence among the classes are within reasonable levels. Accuracy is better than what was obtained with KNN.

Section 2.6.3 - Random Forest Model

Finally, Random Forest model:

```r
random_forest <- train(x = train.set[, cols],
                       y = train.set$state,
                       method = "rf",
                       trControl = control,
                       metric = "Kappa",
                       verbose = F)

y_hat_rf <- predict(random_forest, test.set[, cols])
confusionMatrix(y_hat_rf, as.factor(test.set$state))$overall["Accuracy"]
```

We obtain Overall Accuracy of 0.9391 and Kappa of 0.9065. Besides, Prevalence among the classes are within reasonable levels. It is highest accuracy obtained among these three models.

Section 2.7 - Final Model by Using the Validation Set

Let's re-train over the entire training set by using the Validation Set and see the final accuracy:

```r
set.seed(5)
random_forest_fnl <- train(x = train.set[, cols],
                           y = train.set$state,
                           method = "rf",
                           trControl = control,
                           metric = "Kappa",
                           verbose = F)

y_hat_rf_fnl <- predict(random_forest_fnl, validation[, cols])
confusionMatrix(y_hat_rf_fnl, as.factor(validation$state))$overall["Accuracy"]
```

We obtain Overall Accuracy of 0.933 and Kappa of 0.8964. Prevalence among the classes are within reasonable levels.

Indeed, we get similar results on the prediction set as before. Small confusion on the suspect class, very little to no confusion between early and normal health state, as well as the different failure modes.

Section 3 - Results

The results that were obtained by each model are presented and discussed in this section. Firstly, let us tabulate the results in the table below:

Method	Accuracy
KNN	0.782
Stochastic Gradient Boosting	0.882
Random Forest	0.939
Final Model (Random Forest)	0.933

As seen the results in the table above, the highest accuracy was obtained with the Random Forest model. Therefore, Random Forest was selected to compute the Final Accuracy with the Validation Set in Sec. 2.7.

Section 4 - Conclusion

We have built a classifier that can determine the health status of roller bearings and experimented quite a lot with feature extraction (and reduction) which led us to choose 8 features from the two vibration signals (x- and y- axis). We have managed to get a 93% accuracy and 90% Kappa value on the validation set but the errors are to be expected: There is small level of confusion between early and normal data as well as between suspect and the different failure modes. It is also nice to see that there is very little confusion between the classes relating to good health and those of bad health. In addition, the failure classes are only ever classified as different types of failures and never as normal behavior. The most confusion seems to be in the suspect class but that is understandable, considering that the suspect class is a just a transition from normal to a failure pattern.

As explained earlier, this is a classifier model. However, a regression approach could also be followed to predict the Remaining Useful Life (RUL) of the bearings. Another model that satisfies the RUL approach could be applied as further analysis if the goal is to understand how much time is left before failure occurs.