ESP2107 Project Report

Problem Statement:

Simulation in this domain is computationally expensive, and requires a significant amount of time. The intent of this project is to train a fast model to predict the pressure drop experienced within a drone's battery fuel cells.

The key steps involved will be to:

1. Download and pre-process data.
2. Visualise parameters for intuiting output results.
3. Train a model in R to predict the drop in pressure.
4. Quantify the model's accuracy and further optimise it.
5. Assess the model's capability for use.

Metrics:

The main method of quantifying the model's performance will be using relative error.

Relative error is defined as: $\frac{|\,True\ value - Predicted\ value\,|}{|\,True\ value\,|}$

Model performance outputs will be given with three separate relative errors, 1%, 5%, and 10%.

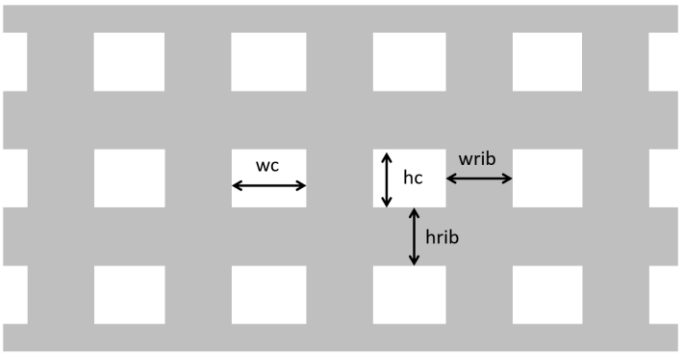
The model will be improved iteratively during training with root mean squared error also.

Data Exploration:

Parameters:

- Pct_length: The percentage length of the channel
- Wc: Width of the air channel (mm)
- Hc: Height of the air channel (mm)
- Hrib: Height of the cell rib (mm)
- Wrib: Width of the cell rib (mm)
- Uin: Wind speed entering the channel (m/s)
- Pressure: The true pressure drop (Pa)

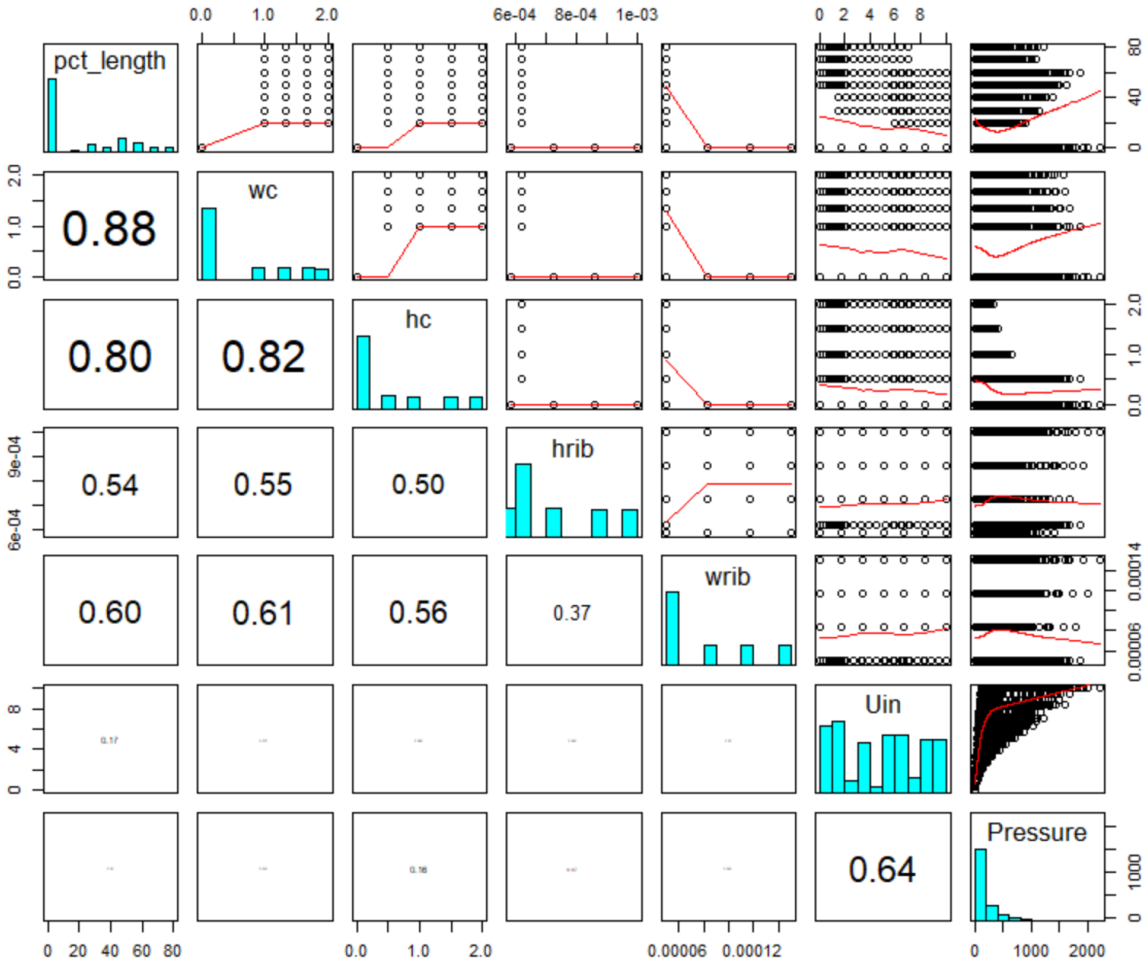
The fuel cell honeycomb structure is depicted below.



Interpret the graph below as follows:

- The main diagonal line of cells contain distributions of each variable in blue.
- The lower triangle shows the correlation between each of the two corresponding variables. The size of the text is correlated to the correlation, if it is too small to be read, the correlation is non-existent.
- The upper triangle shows the trend of the data as explained by each two corresponding variables.

Notice the high correlation between wind speed and pressure drop, relative to the other variables there is a significant difference. Also notice the high correlation between pct_length, wc, and hc. Principal Component Analysis can be performed on these parameters by orthogonally transforming them if an issue arises during modelling.

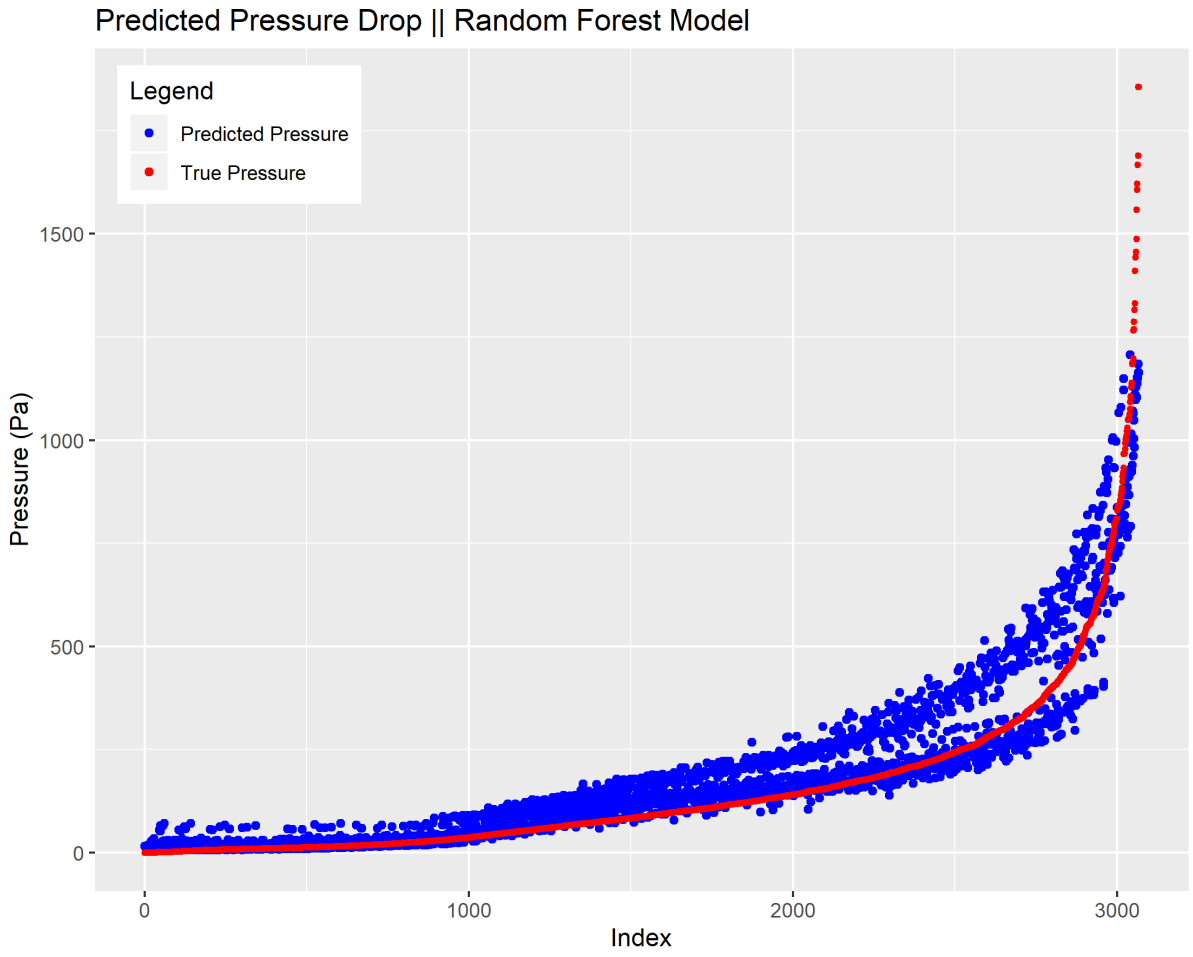


Algorithms & Techniques:

An extreme gradient boosting model was trained on this dataset. This type of model utilises both classification and regression, and iteratively retrains itself using root mean squared error as the specified loss function. Boosting works by training multiple weak models and adding them to a stronger final model. Similar to adding multiple layers of weak graphene to create something much stronger.

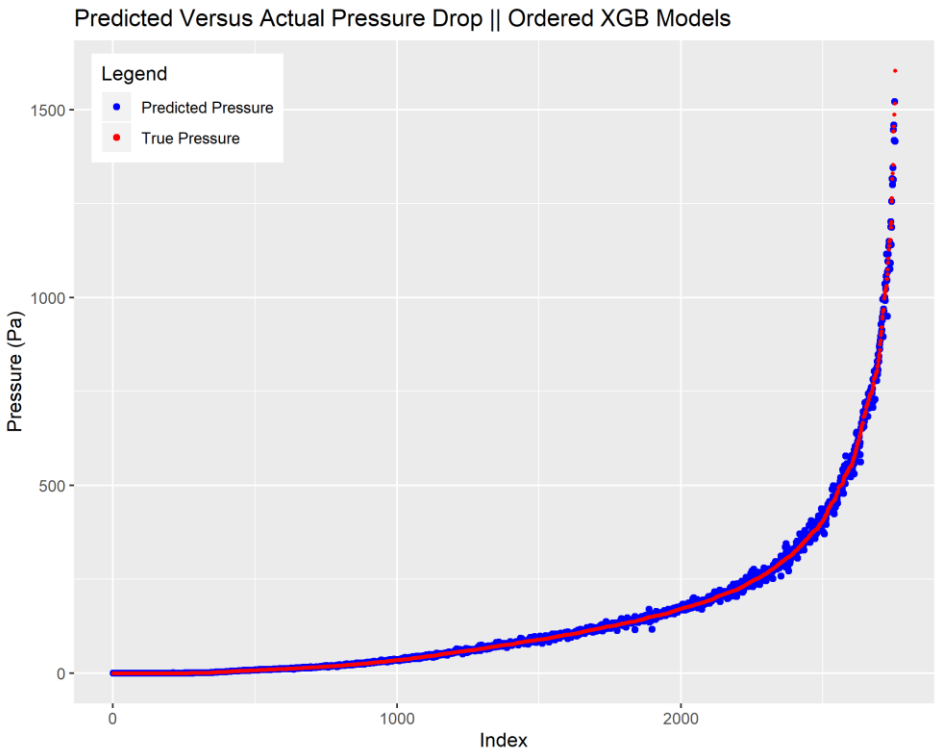
> "If linear regression was a Toyota Camry, then gradient boosting would be a UH-60 Blackhawk Helicopter. A particular implementation of gradient boosting, XGBoost, is consistently used to win machine learning competitions.." – Kaggle.com

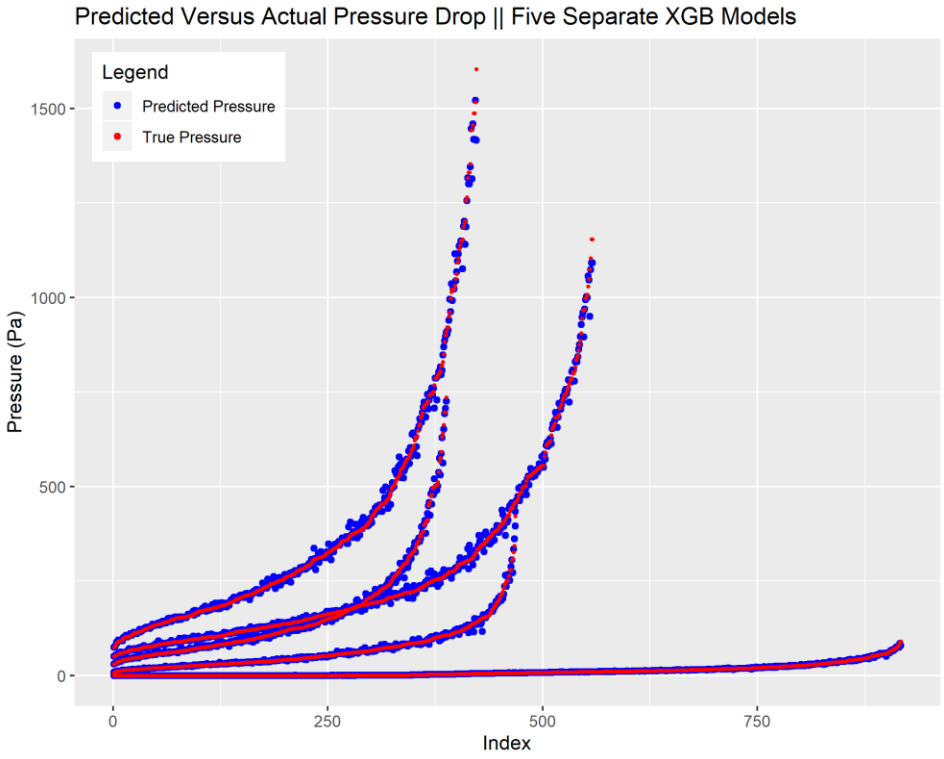
This type of algorithm is more suitable for this project than CART models, such as a random forest, which are unable to predict outliers effectively. Below is a graphical output of a random forest model. This is go to model of mine for almost all supervised machine learning tasks acting as a stage-gate for deciding which type of algorithm to explore next. Note the decent classification ability for the mean pressure drop, however also note that pure classification models underpredict outliers as shown below.



Be advised that the graph above is **not** of the final model.

Instead, five separate extreme gradient boosting models were each trained on 2 m/s bands of the wind speed (Uin) parameter using all supplied variables. This works significantly better than training one model on all of the data as Uin has such a large influence on the dependent variable and upstages the other variables' contributions. Below are the same data on two graphs, showing the outputs of the model in an ordered fashion and also showing all five separate models superimposed onto a single graph.



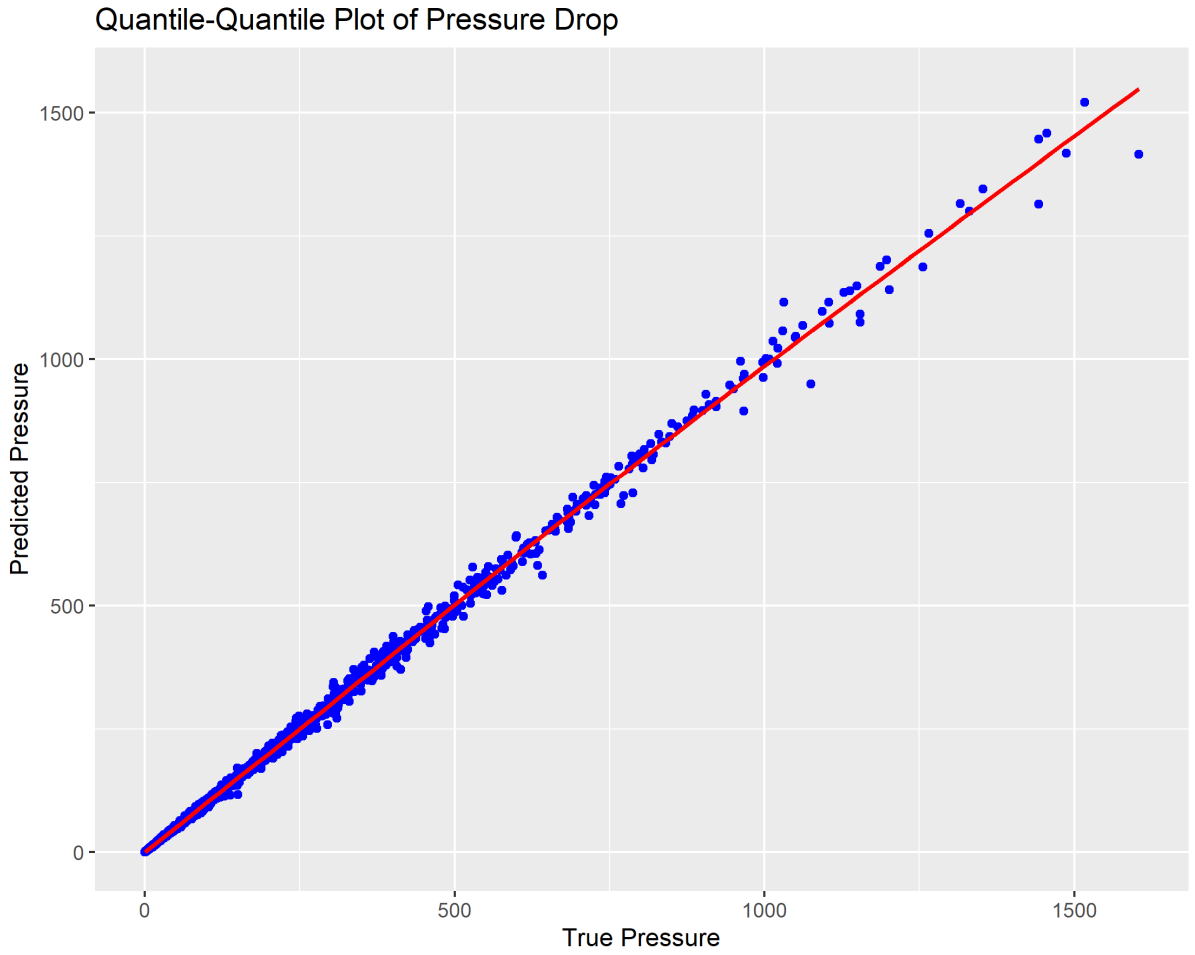


Uin, pct_length, hc, and wc are the most important variables throughout all the models, however the importance of variables differs between all five models. We see that in the lower wind speed models Uin plays a larger role than it does in the larger wind speed models. Pct_length and hc had similar contributions as the most important variables for the four largest wind speed models.

Results & Conclusions:

The models are largely accurate, the table shows that the models are highly accurate to within 5% relative error and also fully accurate to within 10% relative error. Note that the model trained on the [0, 2] m/s wind speed interval is the least accurate.

		Wind Speed				
		[0, 2]	[2, 4]	[4, 6]	[6, 8]	[8, 10]
Error	1%	0.3	0.32	0.27	0.44	0.47
	5%	0.67	0.85	0.88	0.94	0.9
	10%	0.75	0.99	0.98	1	0.98

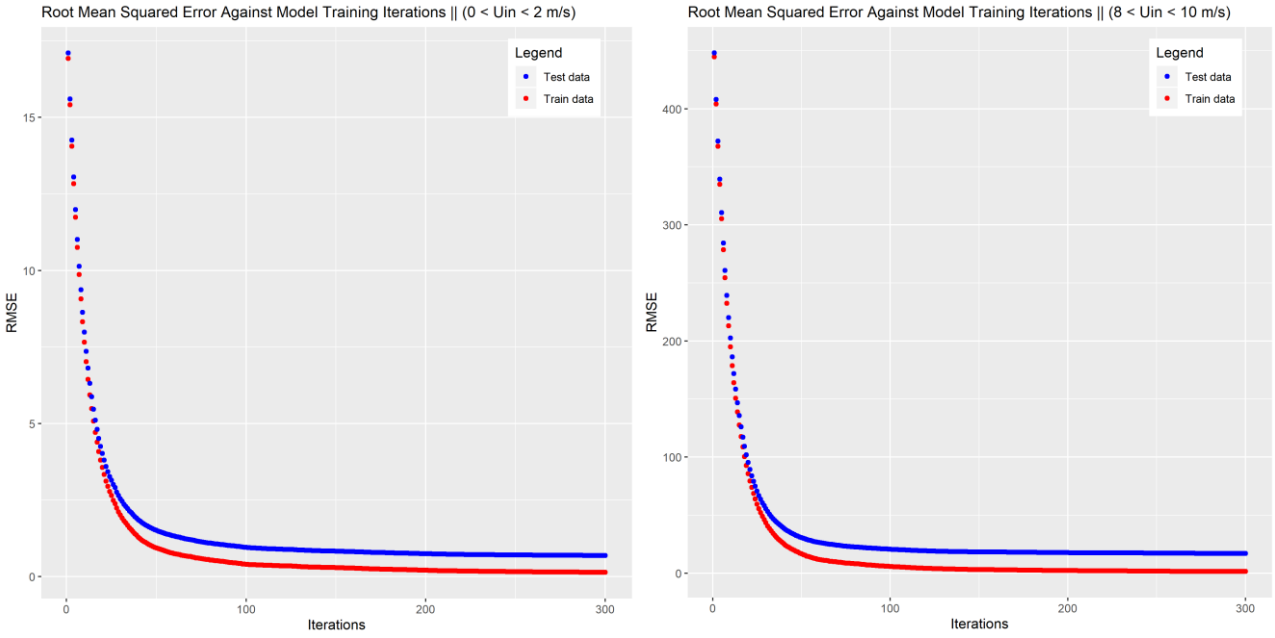


Speed – It took 26 seconds in total on a regular laptop to train all five models at once and to produce all the suite of graphs for each model. This can be further improved by a minor trade off in model accuracy by reducing the number of training iterations of each model.

A ten-second training time yields slightly less accurate models yet is a suitable demonstration of this model's speed and provides a framework for scaling the amount of input data.

		Wind Speed				
		[0, 2]	[2, 4]	[4, 6]	[6, 8]	[8, 10]
Error	1%	0.21	0.21	0.23	0.31	0.32
	5%	0.61	0.75	0.84	0.88	0.9
	10%	0.75	0.95	0.97	0.98	1

The model requires a large number of iterations to be able to train itself to an acceptable standard. The graphs below show that the models have a poor classification ability in the first 50 iterations, before the rate of improvement decreases. Instead of 300 iterations, we could have used 200 and only suffered a minor reduction in model accuracy. If we were to see the test RMSE (blue line) start to increase again, this would be a sign of overfitting, however 300 iterations is too small to see overfitting occur.



Improvements:

Finding the trade-off between the number of models and the data needed to train each model to an accuracy standard might be worthwhile. I arbitrarily picked five yet did not experiment with other numbers of models.

The [0, 2] m/s wind speed model is the worst performing model of the quintet. Exploring why this is the case may be worthwhile, however I would speculate that this is because of the very narrow band of small pressure drops in this model.

Note that in an effort to save space, I have removed the majority of the iteration RMSE value outputs from the markdown script below.

XGB_Model4.R

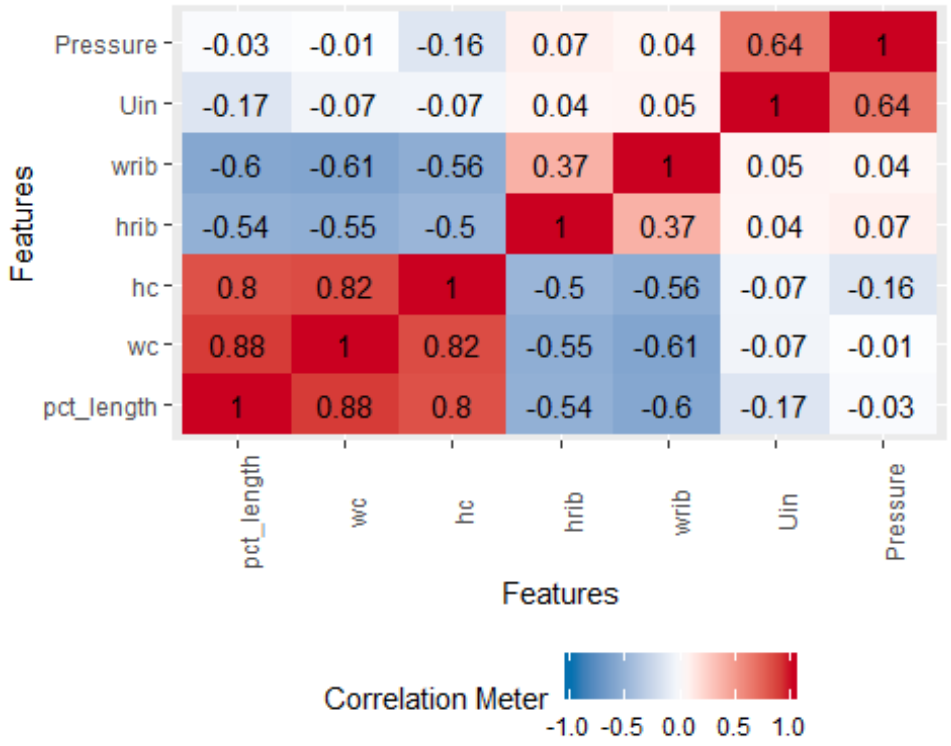
2019-11-06

```
# This script is used to train and test an extreme gradient boosting
# (XGBoost) machine learning model

# Author: Cal Roughan
# Course: ESP2107 - Numerical Methods and Statistics
# Date: 8th October 2019


# Libraries and packages
library("readtext")
library("s20x")
library("DataExplorer")
library("caTools")
library("randomForest")

## randomForest 4.6-14

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

library("plyr")
library("xgboost")
library("Matrix")
library("tidyverse")

## -- Attaching packages ---------------------------------------------
---------------- tidyverse 1.2.1 --

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------
----------- tidyverse_conflicts() --
## x dplyr::arrange()   masks plyr::arrange()
## x dplyr::combine()   masks randomForest::combine()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x tidyr::expand()    masks Matrix::expand()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x ggplot2::margin()  masks randomForest::margin()
## x dplyr::mutate()    masks plyr::mutate()
## x tidyr::pack()      masks Matrix::pack()
```

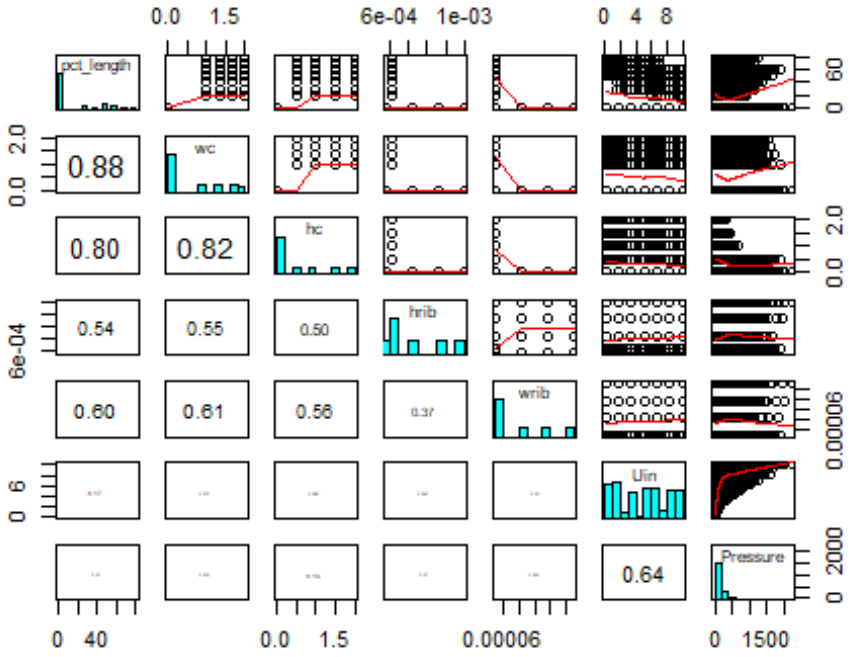
```
## x dplyr::rename()    masks plyr::rename()
## x dplyr::slice()     masks xgboost::slice()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks plyr::summarize()
## x tidyr::unpack()    masks Matrix::unpack()

# Read data
setwd("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\Plots")

df <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\DataPr
essureDrop2019.txt"
                 , fill = TRUE
                 , header = TRUE)
hs_t1 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\HS_
T1.txt"
                    , fill = TRUE
                    , header = TRUE)

hs_t2 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\HS_
T2.txt"
                    , fill = TRUE
                    , header = TRUE)

hs_t3 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\HS_
T3.txt"
                    , fill = TRUE
                    , header = TRUE)

ls_t1 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\LS_
T1.txt"
                    , fill = TRUE
                    , header = TRUE)

ls_t2 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\LS_
T2.txt"
                    , fill = TRUE
                    , header = TRUE)

ls_t3 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\LS_
T3.txt"
                    , fill = TRUE
                    , header = TRUE)


ls_t4 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\LS_
T4.txt"
                    , fill = TRUE
                    , header = TRUE)

ms_t1 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\MS_
T1.txt"
                    , fill = TRUE
                    , header = TRUE)
```

```
ms_t2 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\MS_
T2.txt"
                    , fill = TRUE
                    , header = TRUE)

ms_t3 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\MS_
T3.txt"
                    , fill = TRUE
                    , header = TRUE)

ms_t4 <- read.table("C:\\Users\\calro\\Documents\\ESP2107\\Assignment\\MS_
T4.txt"
                    , fill = TRUE
                    , header = TRUE)

# Concatenate data
new.df <- rbind(hs_t1
                , hs_t2
                , hs_t3
                , ls_t1
                , ls_t2
                , ls_t3
                , ls_t4
                , ms_t1
                , ms_t2
                , ms_t3
                , ms_t4)

# Assign default values as requested
new.df$wrib <- 5e-5
new.df$hrib <- 6.17e-4

# Reorder columns
new.df <- new.df[, c(1, 2, 3, 7, 6, 4, 5)]
df <- rbind(df, new.df)

# Clean dataframes
rm(new.df
   ,hs_t1
   , hs_t2
   , hs_t3
   , ls_t1
   , ls_t2
   , ls_t3
   , ls_t4
   , ms_t1
   , ms_t2
   , ms_t3
   , ms_t4)

# Output file
testMtrx <- setNames(data.frame(matrix(ncol = 9, nrow = 0))
                     , c(names(df)
                         , "pred.xgb"
```

```
                          , "gen_ID"))

# Define lower bands for each model
min_Uin = c(0, 2, 4, 6, 8)
min_Uin[1]

## [1] 0

output_table <- array(dim = c(3, length(min_Uin)))

introduce(df)

##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 7156       7                 0                   7                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0          7156               50092       403048

plot_correlation(df)
```




```
pairs20x(df)
```




```r
# Initiate for loop
for (i in 1:5)
{
  # Subset, prepare, and split the data
  df_temp <- df[df$Uin < min_Uin[i] + 2, ]
  df_temp <- df_temp[df_temp$Uin >= min_Uin[i], ]

  set.seed(42)
  split <- sample.split(df_temp, SplitRatio = 0.7)

  train <- df_temp[split,]
  test <- df_temp[!split,]

  test <- test[order(test$Pressure),]

  gen_ID <- 1:nrow(test)

  # Create watchlist
  test_m <- xgb.DMatrix(data = as.matrix(test[, c(1:6)])
                        , label = test[, 7])

  train_m <- xgb.DMatrix(data = as.matrix(train[, c(1:6)])
                         , label = train[, 7])

  watchlist <- list(train = train_m
                    , test = test_m)

  # XGB modelling
  model <- xgb.train(data = train_m
                     , label = as.matrix(train[, c(7)])
```

```
                      , nrounds = 300
                      , eta = 0.1
                      , subsample = 1
                      , objective = "reg:squarederror"
                      , watchlist = watchlist)

  # Run the test data and store
  pred.xgb <- abs(predict(model, as.matrix(test[, c(1:6)])))

  test <- cbind(test, pred.xgb)
  testMtrx <- rbind(testMtrx, cbind(test, gen_ID))

  # Plot individual graphs of pressure drop
  ggplot(test, aes(y = test$pred.xgb, x = gen_ID)) +
    geom_point(aes(colour = "Predicted Pressure")) +
    geom_point(aes(y = Pressure, x = gen_ID, colour = "True Pressure"), si
ze = 0.9) +
    ggtitle(paste("Pressure drop || XGB model || ("
                                    , min_Uin[i], " < Uin < "
                                    , min_Uin[i] + 2
                                    , " m/s)"
                                    , sep = "")) +
    labs(colour="Legend",x="Index",y="Pressure (Pa)")+
    theme(legend.position = c(0.02, .98), legend.justification = c(0, 1))
+
    scale_color_manual(values = c("blue","red"))

  ggsave(filename = paste("XGBoost_Plot_", i, ".png", sep = ""), plot = la
st_plot())

  # Model analysis
  print(xgb.importance(colnames(train[, -7])
                , model = model))

  err <- data.frame(model$evaluation_log)

  ggplot(err, aes(y = train_rmse, x = iter)) +
    geom_point(aes(colour = "Train data")) +
    geom_point(aes(y = test_rmse, x = iter, colour = "Test data")) +
    ggtitle(paste("Root Mean Squared Error Against Model Training Iteratio
ns || ("
                                    , min_Uin[i], " < Uin < "
                                    , min_Uin[i] + 2
                                    , " m/s)"
                                    , sep = "")) +
    theme(plot.title = element_text(size = 12)) +
    labs(colour="Legend",x="Iterations", y="RMSE")+
    theme(legend.position = c(0.79, .98), legend.justification = c(0, 1))
+
    scale_color_manual(values = c("blue","red"))

  ggsave(filename = paste("RMSE_Plot_", i, ".png", sep = ""), plot = last_
plot())
```

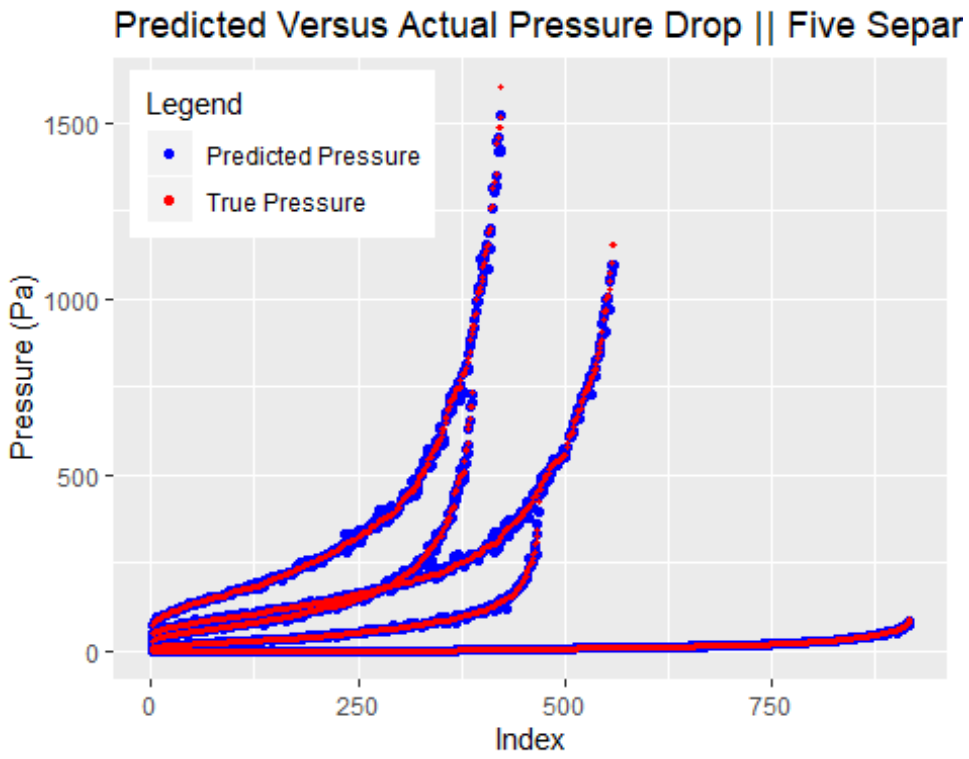
```
  # Quantify outputs

  temp_table <- table(abs(test$Pressure-test$pred.xgb)/test$Pressure <= 0.
01)

  output_table[1, i] <- round((temp_table[2])/(temp_table[1] + temp_table[
2]), 2)

  temp_table <- table(abs(test$Pressure-test$pred.xgb)/test$Pressure <= 0.
05)

  output_table[2, i] <- round((temp_table[2])/(temp_table[1] + temp_table[
2]), 2)

  temp_table <- table(abs(test$Pressure-test$pred.xgb)/test$Pressure <= 0.
1)

  output_table[3, i] <- round((temp_table[2])/(temp_table[1] + temp_table[
2]), 2)

  rm(temp_table)
}
```

```
## [1]  train-rmse:16.919746    test-rmse:17.099556
## [2]  train-rmse:15.410169    test-rmse:15.602957
## [3]  train-rmse:14.058943    test-rmse:14.260110
## [4]  train-rmse:12.837038    test-rmse:13.052180
## [5]  train-rmse:11.739763    test-rmse:11.982673
## [6]  train-rmse:10.750588    test-rmse:11.012337
## [295]    train-rmse:0.138513 test-rmse:0.688707
## [296]    train-rmse:0.137614 test-rmse:0.688074
## [297]    train-rmse:0.137148 test-rmse:0.687746
## [298]    train-rmse:0.136440 test-rmse:0.687258
## [299]    train-rmse:0.136047 test-rmse:0.687337
## [300]    train-rmse:0.135106 test-rmse:0.686634

## Saving 5 x 4 in image

##       Feature        Gain      Cover  Frequency
## 1:        Uin 0.484539906 0.23542839 0.21747708
## 2: pct_length 0.244150842 0.22939842 0.17976094
## 3:         hc 0.187412188 0.18467761 0.21910178
## 4:         wc 0.064878765 0.16170819 0.20645236
## 5:       hrib 0.012752691 0.10953118 0.08657305
## 6:       wrib 0.006265609 0.07925621 0.09063479

## Saving 5 x 4 in image

## [1]  train-rmse:82.604446    test-rmse:81.151131
## [2]  train-rmse:75.304207    test-rmse:74.127663
## [3]  train-rmse:68.757271    test-rmse:67.806549
## [4]  train-rmse:62.808712    test-rmse:62.133301
## [5]  train-rmse:57.482986    test-rmse:56.970379
## [6]  train-rmse:52.680923    test-rmse:52.411575
## [295]    train-rmse:0.493775 test-rmse:4.343574
```

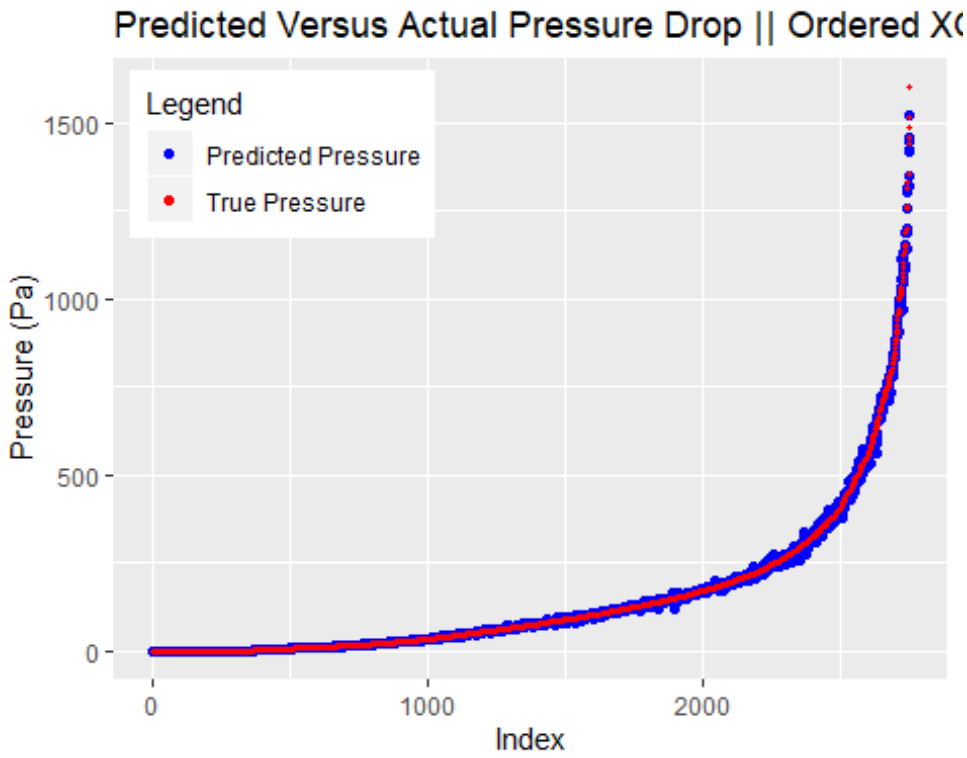
```
## [296]    train-rmse:0.491299 test-rmse:4.343818
## [297]    train-rmse:0.487387 test-rmse:4.341497
## [298]    train-rmse:0.486163 test-rmse:4.340413
## [299]    train-rmse:0.482670 test-rmse:4.338017
## [300]    train-rmse:0.479961 test-rmse:4.336661

## Saving 5 x 4 in image

##       Feature        Gain      Cover  Frequency
## 1: pct_length 0.419173356 0.27446768 0.24370526
## 2:         hc 0.250406585 0.19100845 0.22093310
## 3:        Uin 0.207410326 0.13623448 0.12034066
## 4:         wc 0.100274110 0.20619514 0.21716860
## 5:       hrib 0.015965261 0.09725581 0.10571464
## 6:       wrib 0.006770361 0.09483844 0.09213774

## Saving 5 x 4 in image

## [1]  train-rmse:187.164612   test-rmse:185.825195
## [2]  train-rmse:170.586029   test-rmse:169.551346
## [3]  train-rmse:155.473053   test-rmse:154.697739
## [4]  train-rmse:141.784241   test-rmse:141.038910
## [5]  train-rmse:129.528442   test-rmse:129.277023
## [6]  train-rmse:118.507347   test-rmse:118.712212
## [295]    train-rmse:0.690856 test-rmse:7.491752
## [296]    train-rmse:0.687392 test-rmse:7.490374
## [297]    train-rmse:0.683035 test-rmse:7.488680
## [298]    train-rmse:0.677346 test-rmse:7.487004
## [299]    train-rmse:0.672363 test-rmse:7.485129
## [300]    train-rmse:0.666934 test-rmse:7.480933

## Saving 5 x 4 in image

##       Feature        Gain      Cover Frequency
## 1:         hc 0.495366659 0.19909328 0.2356380
## 2: pct_length 0.351715311 0.26659455 0.2134596
## 3:         wc 0.098277984 0.20467486 0.2168912
## 4:        Uin 0.028565102 0.07127311 0.0769573
## 5:       hrib 0.018191568 0.14665432 0.1284316
## 6:       wrib 0.007883376 0.11170988 0.1286223

## Saving 5 x 4 in image

## [1]  train-rmse:308.514923   test-rmse:311.300110
## [2]  train-rmse:280.302643   test-rmse:283.654480
## [3]  train-rmse:254.685471   test-rmse:258.227570
## [4]  train-rmse:231.569366   test-rmse:235.338684
## [5]  train-rmse:210.848892   test-rmse:215.168198
## [6]  train-rmse:192.106537   test-rmse:196.547012
## [295]    train-rmse:1.444556 test-rmse:9.516602
## [296]    train-rmse:1.437536 test-rmse:9.511374
## [297]    train-rmse:1.432878 test-rmse:9.509795
## [298]    train-rmse:1.424848 test-rmse:9.502144
## [299]    train-rmse:1.419633 test-rmse:9.495366
## [300]    train-rmse:1.408628 test-rmse:9.486091
```

```
## Saving 5 x 4 in image

##       Feature        Gain      Cover  Frequency
## 1: pct_length 0.648767908 0.27301610 0.23815721
## 2:         hc 0.240705739 0.20553934 0.23503384
## 3:         wc 0.075513551 0.18887552 0.23067413
## 4:        Uin 0.015747875 0.14346151 0.09383134
## 5:       hrib 0.012839282 0.10769891 0.10482821
## 6:       wrib 0.006425644 0.08140862 0.09747527

## Saving 5 x 4 in image

## [1]  train-rmse:444.652069   test-rmse:448.005981
## [2]  train-rmse:404.132263   test-rmse:408.015442
## [3]  train-rmse:367.655609   test-rmse:372.098053
## [4]  train-rmse:334.761871   test-rmse:339.446259
## [5]  train-rmse:305.305847   test-rmse:310.520355
## [6]  train-rmse:278.516052   test-rmse:284.243622
## [295]    train-rmse:1.395277 test-rmse:17.000362
## [296]    train-rmse:1.390577 test-rmse:16.995840
## [297]    train-rmse:1.387786 test-rmse:16.994135
## [298]    train-rmse:1.383565 test-rmse:16.992071
## [299]    train-rmse:1.380717 test-rmse:16.989025
## [300]    train-rmse:1.376938 test-rmse:16.987425

## Saving 5 x 4 in image

##       Feature        Gain      Cover  Frequency
## 1:         hc 0.593099619 0.21763221 0.23537093
## 2: pct_length 0.268767829 0.23789006 0.21205149
## 3:         wc 0.103036563 0.20585902 0.21230023
## 4:       hrib 0.017407759 0.14278220 0.16180586
## 5:        Uin 0.009731716 0.06680303 0.05061874
## 6:       wrib 0.007956514 0.12903348 0.12785275

## Saving 5 x 4 in image

print(output_table)

##      [,1] [,2] [,3] [,4] [,5]
## [1,] 0.25 0.25 0.29 0.39 0.39
## [2,] 0.64 0.78 0.88 0.93 0.92
## [3,] 0.77 0.97 0.99 0.99 1.00

testMtrx <- testMtrx[order(testMtrx$Pressure),]
testMtrx$gen_ID2 <- 1:nrow(testMtrx)

# Plot all together - Five separate models, no reordering
ggplot(testMtrx, aes(y = testMtrx$pred.xgb, x = testMtrx$gen_ID)) +
  geom_point(aes(colour = "Predicted Pressure")) +
  geom_point(aes(y = Pressure, x = gen_ID, colour = "True Pressure"), size
= 0.7) +
  ggtitle(paste("Predicted Versus Actual Pressure Drop || Five Separate XG
B Models")) +
  labs(colour="Legend",x="Index",y="Pressure (Pa)")+
```

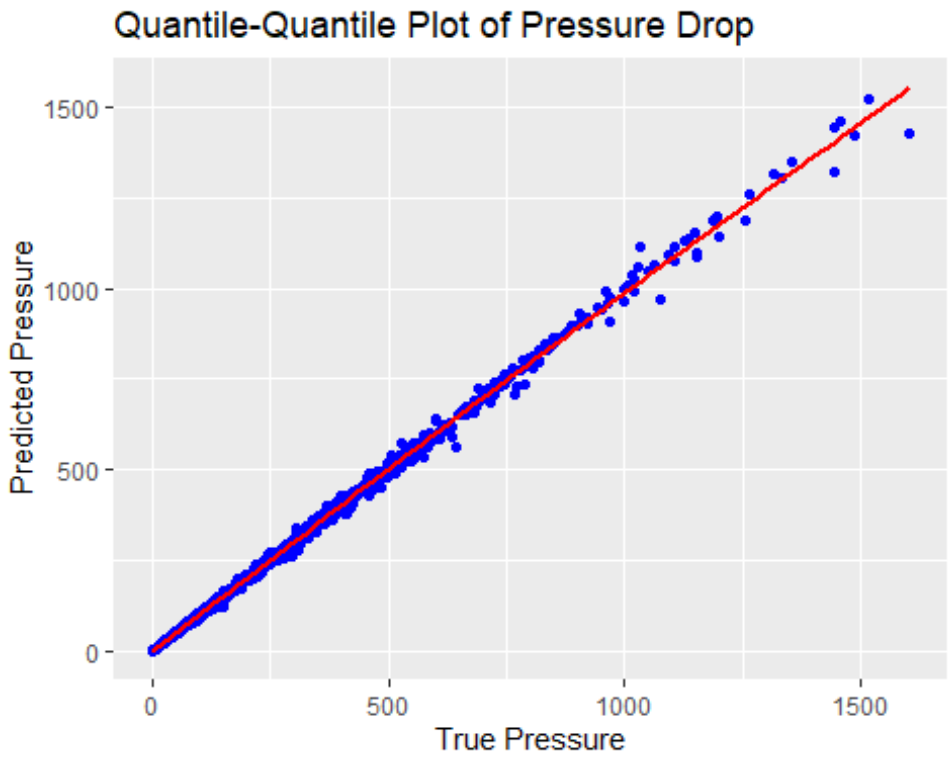
```
  theme(legend.position = c(0.02, .98), legend.justification = c(0, 1)) +
  scale_color_manual(values = c("blue","red"))
```




```
ggsave(filename = "XGB_Unordered_Model.png", plot = last_plot())

## Saving 5 x 4 in image

# Plot all together - Reordered
ggplot(testMtrx, aes(y = testMtrx$pred.xgb, x = testMtrx$gen_ID2)) +
  geom_point(aes(colour = "Predicted Pressure")) +
  geom_point(aes(y = Pressure, x = gen_ID2, colour = "True Pressure"), siz
e = 0.7) +
  ggtitle(paste("Predicted Versus Actual Pressure Drop || Ordered XGB Mode
ls")) +
  labs(colour="Legend",x="Index",y="Pressure (Pa)")+
  theme(legend.position = c(0.02, .98), legend.justification = c(0, 1)) +
  scale_color_manual(values = c("blue","red"))
```




```
ggsave(filename = paste("XGB_Ordered_Model.png"), plot = last_plot())

## Saving 5 x 4 in image

# Add relative error to matrix
testMtrx$error <- abs(testMtrx$Pressure-testMtrx$pred.xgb)/testMtrx$Pressu
re

# Compare predicted to actual pressure drop, size by error
ggplot(testMtrx, aes(y = pred.xgb, x = Pressure)) +
  geom_point(colour = "blue") +
  geom_smooth(color = "red") +
  ggtitle("Quantile-Quantile Plot of Pressure Drop") +
  labs(colour="Legend",x="True Pressure",y="Predicted Pressure")

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
ggsave(filename = paste("XGB_Quantile_Model.png"), plot = last_plot())
```

```
## Saving 5 x 4 in image
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
# # Plot same data but only error > 5%
# testMtrx2 <- testMtrx[testMtrx$error > 0.5, ]
# ggplot(testMtrx2, aes(y = pred.xgb, x = Pressure)) +
#   geom_point(aes(color = testMtrx2$Pressure, size = testMtrx2$error)) +
geom_smooth(color = "red")
```