Group 5 Final Project

Jordan Coelho, Charlie Eastman, Noah Greco, Ishan Kaushal, Jake Simpson

2022-12-08

```
data <- read.csv("~/5th year/3DA3/RStudio Working Directory/maintenance.csv")
```

Introduction

The purpose of this report will be to explore a new dataset, Maintenance, and use the knowledge and tools gained in class to prepare, explore, and build prediction models for the data within the dataset.The Maintenance dataset showcases information from different machines and equipment involved in the process of various jobs/processes at a production facility. The variables assess the overall functionality of the machines and equipment in the scope of the various jobs. Variables include: ID, Quality, Temperature, Process Temperature, Rotation Speed, Intensity, Process Length, and Failed (our target variable). Given the variables provided we are testing to see if we can predict if a job will fail or not depending on the given input variables.

Understanding the Data

First, we’ll look at the dataset to understand a breakdown of the data we’re working with including, the size, the variables, and the types of variables. To do this, we’ll run the head and structure functions on the data.

```
head(data)
```

```
##         ID Quality  Temp ProcessTemp RotationSpeed Intensity ProcessLength
## 1 EQ106761       M 24.95       50.45          1551    Medium             0
## 2 EQ100798       M 25.05       50.55          1408      High             3
## 3 EQ108602       H 24.95       50.35          1498      High             5
## 4 EQ106046       H 25.05       50.45          1433    Medium             7
## 5 EQ109212       M 25.05       50.55          1408    Medium             9
## 6 EQ106902       H 24.95       50.45          1425    Medium            11
##   Failed
## 1      0
## 2      0
## 3      0
## 4      0
## 5      0
## 6      0
```

```
str(data)
```

```
## 'data.frame':    10000 obs. of  8 variables:
##  $ ID           : chr  "EQ106761" "EQ100798" "EQ108602" "EQ106046" ...
##  $ Quality      : chr  "M" "M" "H" "H" ...
##  $ Temp         : num  24.9 25.1 24.9 25.1 25.1 ...
##  $ ProcessTemp  : num  50.5 50.5 50.4 50.5 50.5 ...
##  $ RotationSpeed: int  1551 1408 1498 1433 1408 1425 1558 1527 1667 1741 ...
##  $ Intensity    : chr  "Medium" "High" "High" "Medium" ...
##  $ ProcessLength: int  0 3 5 7 9 11 14 16 18 21 ...
##  $ Failed       : int  0 0 0 0 0 0 0 0 0 0 ...
```

These numbers indicate that there are 10,000 rows within the dataset, which means this is a relatively large dataset and should help in identifying trends and building prediction models. There are 8 columns in the data set and we can see the columns are the characteristics of machines. The structure of the data shows us that the column variables consist of 4 categorical (ID, Quality, Intensity, Failed) and 4 numerical (Temperature, Process Temperature, Rotation Speed, Process Length). The categorical variable, Failed, indicates whether a machine has broke down or not based on a 0 or 1 score and will be the variable that we attempt to predict in our models.

Data Preparation

Before we begin exploring the data and the relationships between the variables, we must examine the data to determine if there are any missing values and determine how we need to address those. To do this we will use the sum() and is.na() functions on each variable to determine how many rows have missing data for each variable.

```
sum(is.na(data$Intensity))
```

```
## [1] 5
```

```
sum(is.na(data$Failed))
```

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

After running these functions, we can tell Intensity has missing values for 5 rows, and Failed has 4 rows with missing values. Since these are categorical variables, we are electing to remove these rows rather than replace the values with another. With the dataset being extremely large, removing the data from these rows should not skew our results in any direction.

```
data <- data[complete.cases(data), ]
nrow(data)
```

```
## [1] 9991
```

To remove these missing values from the data, we have updated the dataset to only include complete cases. This brings the total number of rows to 9991 with 9 being removed.

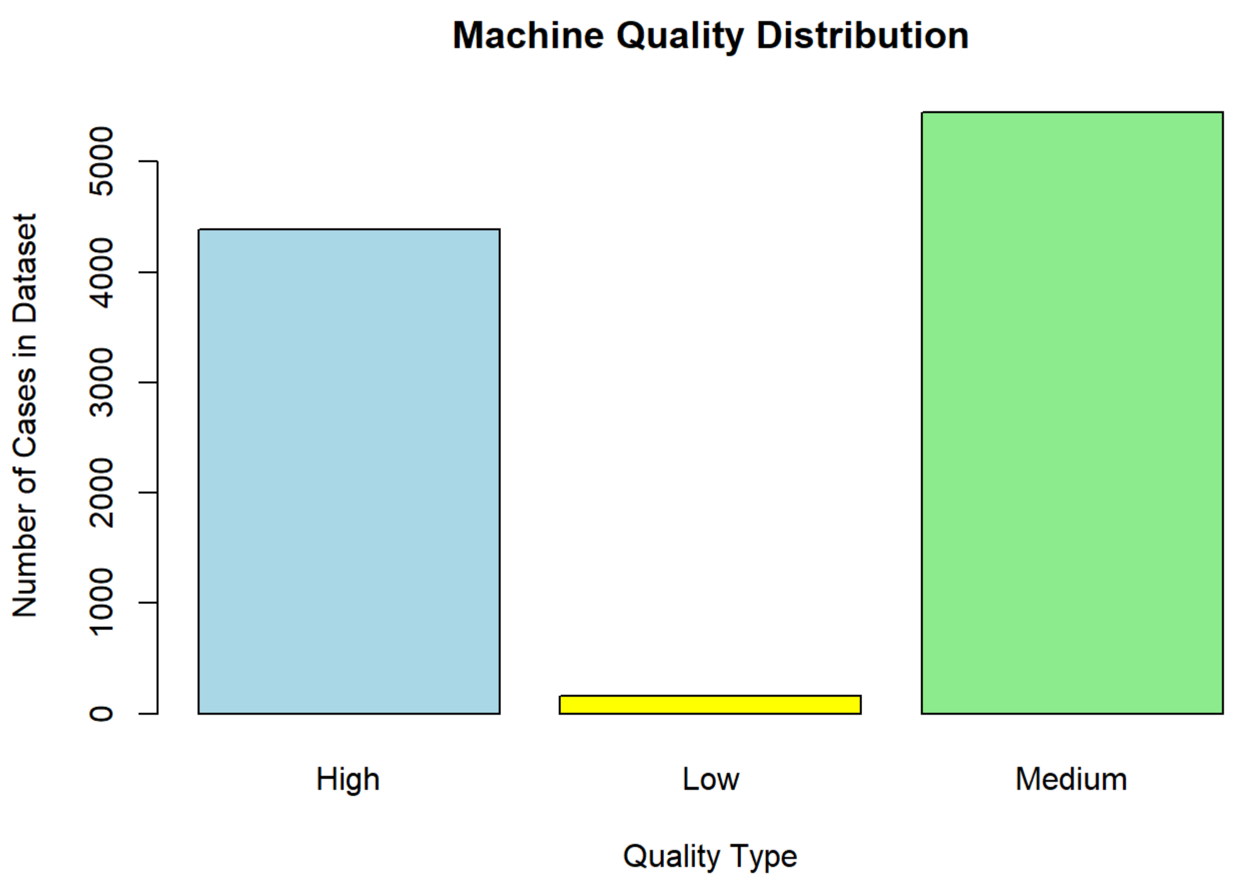
Data Exploration

In the data exploration section of the report, we want to further our knowledge of the variables within our dataset and some relationships that may exist between them, particularly the relationships between our independent variables and the target variable. To do this, we will attempt to visualize the variables and perform simple math functions on them if applicable.

Quality

Beginning with quality, we will view the distribution of the variable through a barplot to see the difference in the total number of cases. We will also use a prop.table() function to the get exact percentage distribution.

```
qual_table <- table(data$Quality)
names(qual_table) <- c("High", "Low", "Medium")

barplot(qual_table,
        main = "Machine Quality Distribution",
        xlab = "Quality Type",
        ylab = "Number of Cases in Dataset",
        col = c("lightblue", "yellow", "lightgreen")
        )
```



```
round(prop.table(qual_table),3)
```

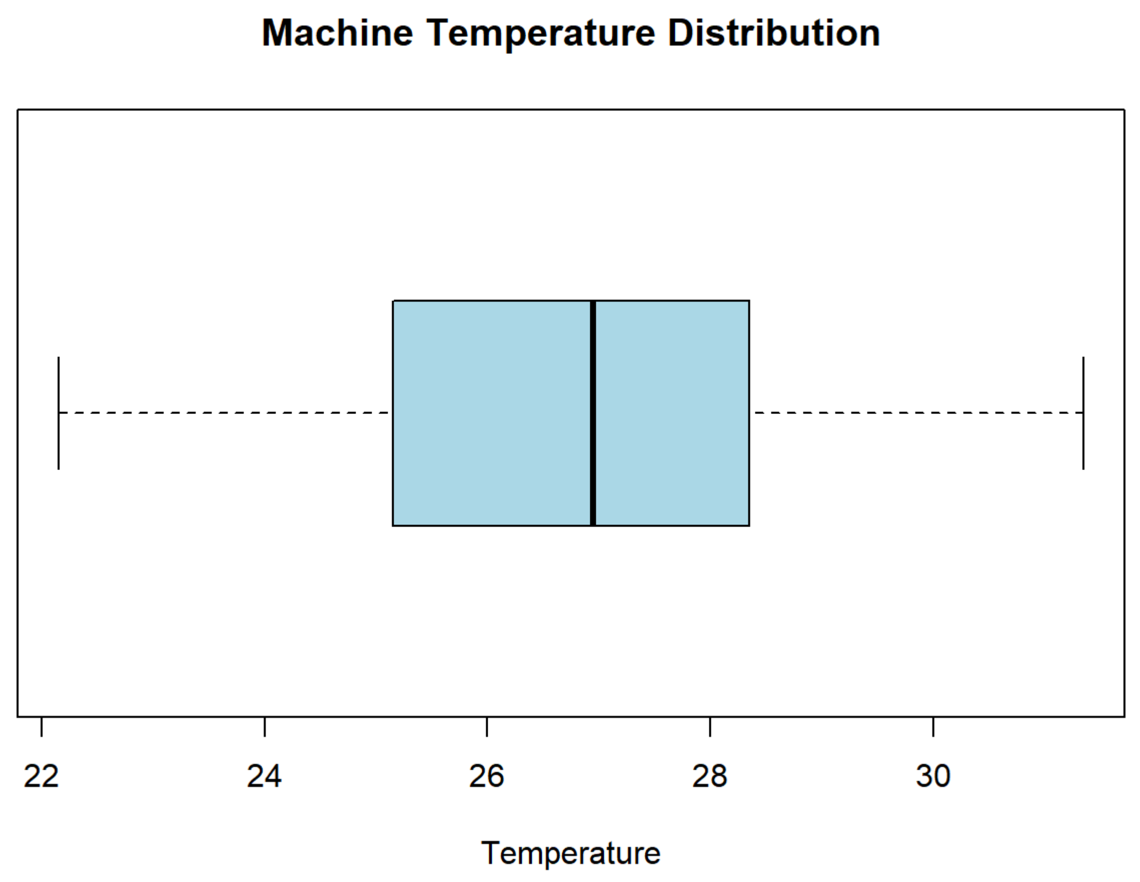
```
##   High    Low Medium 
##  0.439  0.016  0.545
```

Looking at the Quality variable, most of the machines are either High-Quality or Medium-Quality, with them accounting for roughly 44% and 54% respectively of the total cases, while Low-Quality accounts for roughly 2% of the cases. Therefore, the data is unevenly distributed for Quality.

Temperature

Next we’re going to visualize the Temperature variable distribution through a boxplot.

```
boxplot(data$Temp, main = "Machine Temperature Distribution",
       xlab = "Temperature", col = "lightblue", horizontal = TRUE)
```



```
round(median(data$Temp), 2)
```

```
## [1] 26.95
```

```
quantile(data$Temp, .25)
```

```
##   25%
## 25.15
```

```
quantile(data$Temp, .75)
```

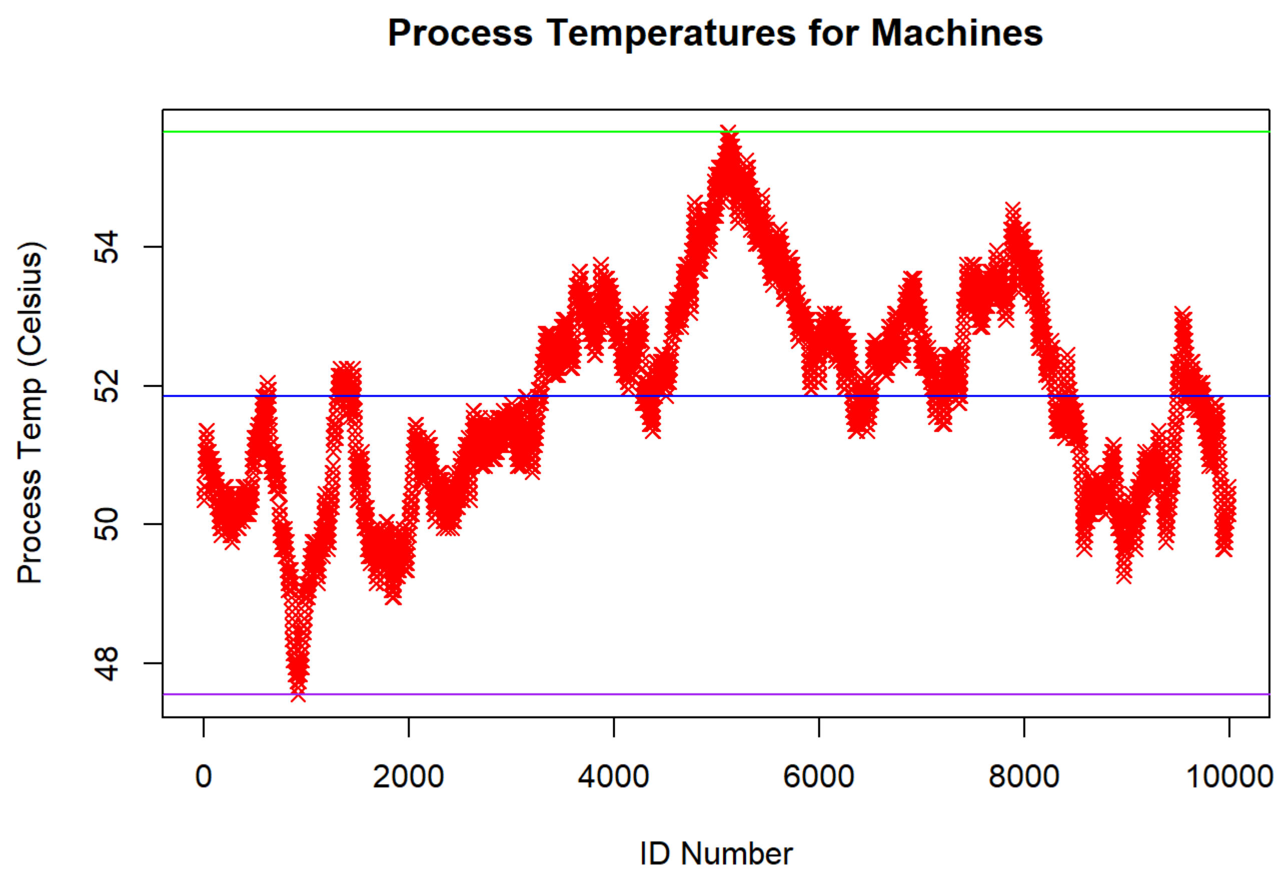
```
##   75%
## 28.35
```

Based on the analysis of the boxplot, we can see some interesting statistics regarding the data. The boxplot shows that the median for the data is around 26.95. The Lower quartile is around 25 to 25.3 while the upper quartile is approximately around 28.2-28.5. Which gives an inter quartile range of approximately 3. Most of the data is around the range of the upper and lower quartile however there are some outliers in the data which shows that the temperature of the process does vary among the machines.

Process Temperature

Next, we visualized the Process Temp variable by using the plot() function and inserted horizontal lines on the graph to illustrate some of the key figures such as the min, max, and mean of the data.

```
plot(data$ProcessTemp, main = "Process Temperatures for Machines",
     col = "red", xlab = "ID Number",
     ylab = "Process Temp (Celsius)", pch = 4)

abline(h = 47.55, col = "purple")
abline(h = 55.65, col = "green")
abline(h = 51.85545, col = "blue")
```



```
round(min(data$ProcessTemp), 2)
```

```
## [1] 47.55
```

```
round(max(data$ProcessTemp), 2)
```

```
## [1] 55.65
```

```
round(mean(data$ProcessTemp), 2)
```

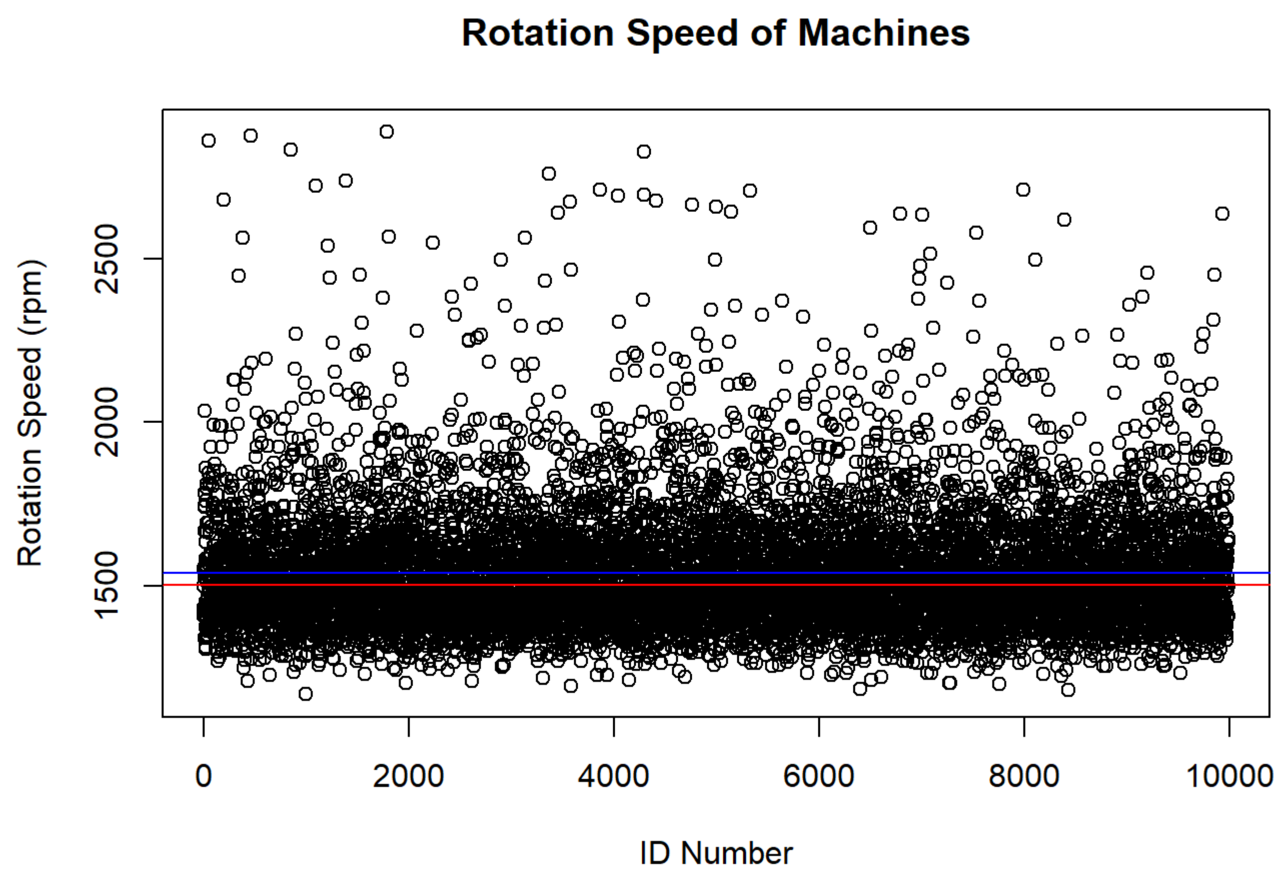
```
## [1] 51.86
```

The process temperature was indicated by a red x for each ID number. The graph indicates a variety of process temperatures as the data drops below 48 and gets above 54. The next step at looking into the process temperature was learning the min, max, and mean of the variable. We have indicated these measures in the graph by adding a purple, green, and blue line to indicate each value respectively.

Rotation Speed

Rotation Speed was visualized through a scatter plot where we were able to see the distribution of the variable by ID Number. Two lines were added for the mean and median of the data.

```
plot(data$RotationSpeed, main = "Rotation Speed of Machines",
     col = "black", xlab = "ID Number", ylab = "Rotation Speed (rpm)")

abline(h = 1538.797, col = "blue")
abline(h = 1503, col = "red")
```



```
round(mean(data$RotationSpeed), 2)
```

```
## [1] 1538.8
```

```
round(median(data$RotationSpeed), 2)
```

```
## [1] 1503
```

```
round(sd(data$RotationSpeed), 2)
```

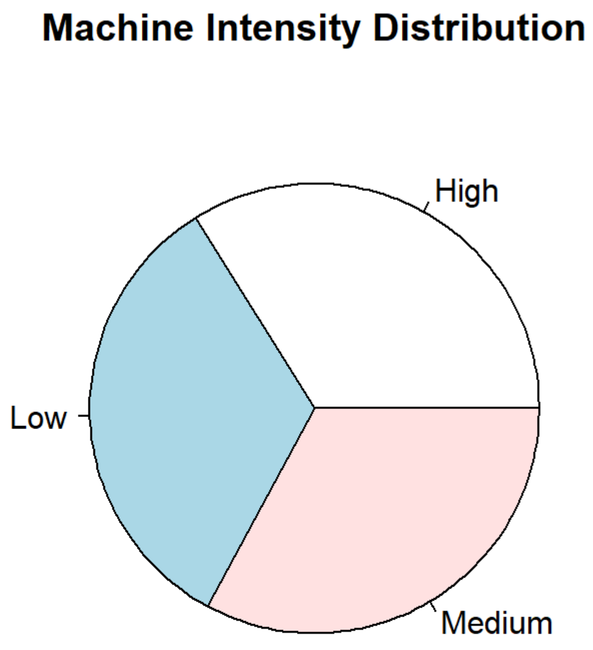
```
## [1] 179.29
```

The rotation speed is indicated by small black circles. We can see among the data that the circles are heavily populated in the lower region of the graph. Next, we looked at obtaining the mean, median, and standard deviation of rotation speed. The mean of rotation speed was 1538.78 and the median was 1503. We have added a blue and red line to the graph to represent the mean and median respectively. The standard deviation for this variable was approximately 180 meaning on average the process temp was about 180 from the mean.

Intensity

For Intensity, we decided to view the distribution through a pie graph and prop.table() to get the exact distribution percentage.

```
pie(table(data$Intensity),
    main = "Machine Intensity Distribution")
```



```
round(prop.table(table(data$Intensity)),3)
```

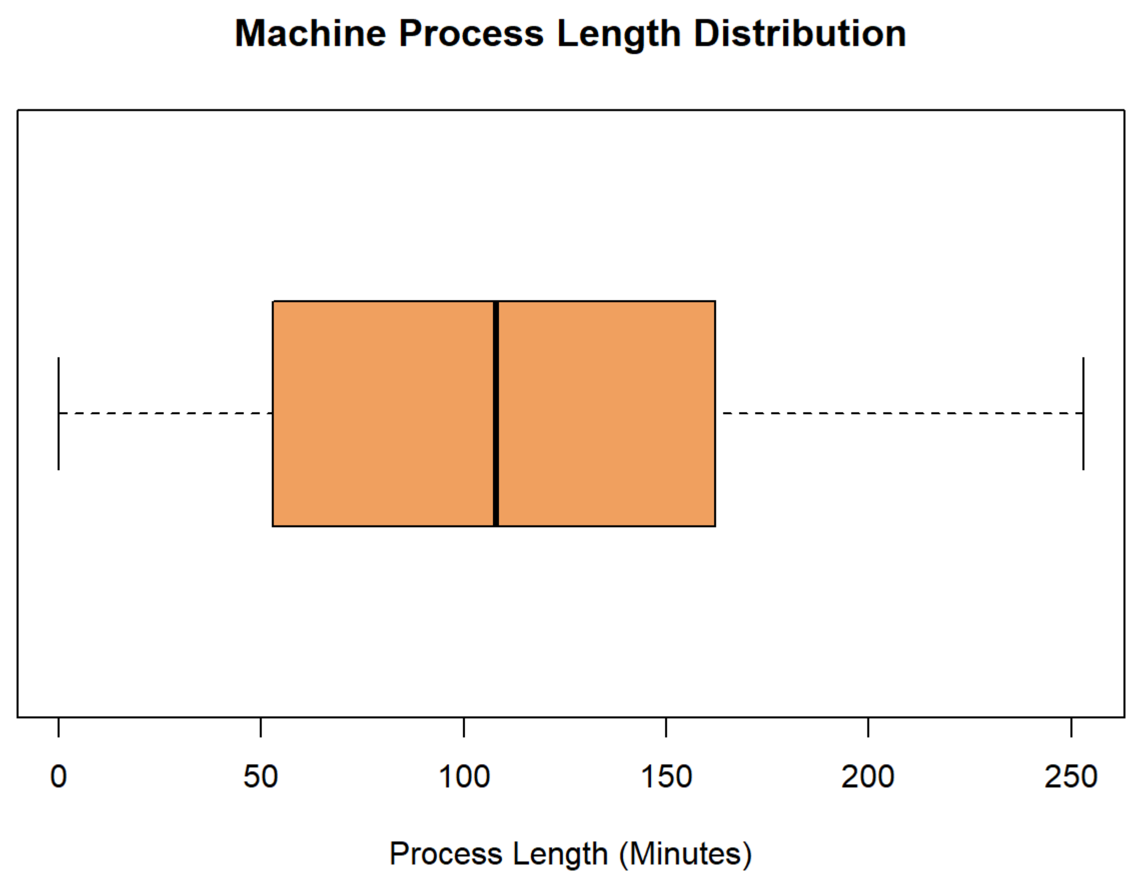
```
##
##   High    Low Medium
##  0.339  0.332  0.329
```

We discovered the intensity is relatively evenly split across low, medium and high process or job. Assessing the proportion more closely through a probability table, we can see that there are slightly more High Intensity jobs at 0.339 or 33.9%, however this is a marginal difference as low and medium jobs also fall around the 33% mark.

Process Length

We decided to view the Process Length variable through a boxplot graph and also find the mean, min, max, and standard deviation of the data.

```
boxplot(data$ProcessLength,
        main = "Machine Process Length Distribution",
        xlab = "Process Length (Minutes)",
        col = "sandybrown",
        horizontal = TRUE)
```



```
round(mean(data$ProcessLength), 2)
```

```
## [1] 107.94
```

```
round(min(data$ProcessLength), 2)
```

```
## [1] 0
```

```
round(max(data$ProcessLength), 2)
```

```
## [1] 253
```

```
round(sd(data$ProcessLength), 2)
```

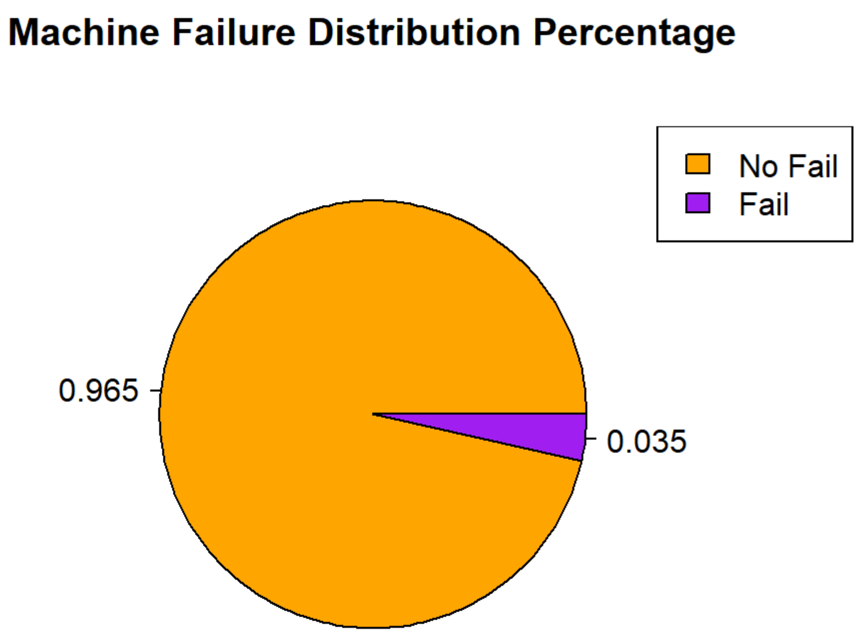
```
## [1] 63.64
```

The boxplot for Process Length shows the IQR of the data fall in the range of about 50 to 175 minutes, while the data is also slightly skewed to the left of the graph. The mean process length is about 108, the min is 0, and the max is 253 minutes. The standard deviation of the data is about 64 minutes meaning on average the process length is about 1 hour away from the mean.

Failed

For the target variable Failed, we decided to visualize it through a pie graph, showing the percentage of machines that break down or “Fail” against the number of machines that do not break down or experience “No Fail”. We also created a legend to indicate which percentage was for each possible result.

```
failed_table <- table(data$Failed)
names(failed_table) <- c(round((failed_table[1])/(failed_table[1] + failed_table[2]), 3), round((failed_table[2])/(failed_ta
ble[1] + failed_table[2]), 3))

par(mar = c(5, 5, 5, 5))
pie(failed_table,
    main = "Machine Failure Distribution Percentage",
    col = c("orange", "purple"))

legend("topright",legend = c("No Fail", "Fail"), fill = c("orange", "purple")
       )
```

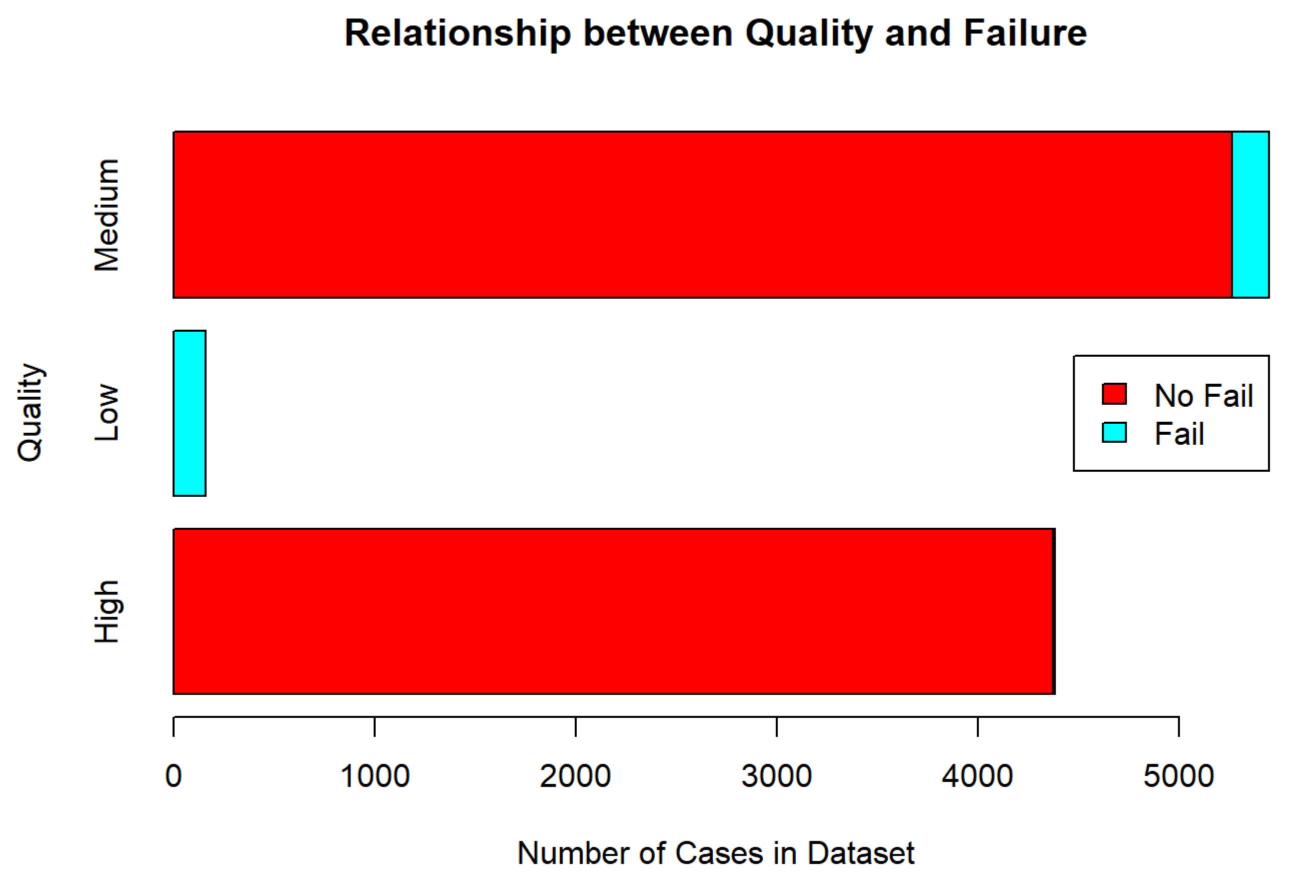


The distribution represented in the pie graph shows almost 97% of the machines will not fail while about 3-4% of the machines will break down or "Fail". When building our prediction models, it is important to note that most of the predictions should be No Fail.

Relationship between Quality and Failure

Next we decided to visualize the relationship between the type of Quality and Failure rate to see if any relationship exists. We showed this through a horizontal barplot with the bars indicating the number of machines that do not fail versus the number of machines that do fail. We also use a prop.table() to get the exact percentages and a legend was created.

```
temp_table <- table(data$Failed, data$Quality)
rownames(temp_table) <- c("No Fail", "Fail")
colnames(temp_table) <- c("High", "Low", "Medium")

barplot(temp_table,
        main = "Relationship between Quality and Failure",
        ylab = "Quality", xlab = "Number of Cases in Dataset",
        col = rainbow(2), horiz = TRUE)

legend("right",legend = c("No Fail", "Fail"), fill = rainbow(2)
       )
```



```
round(prop.table(temp_table, margin = 2), 3)
```

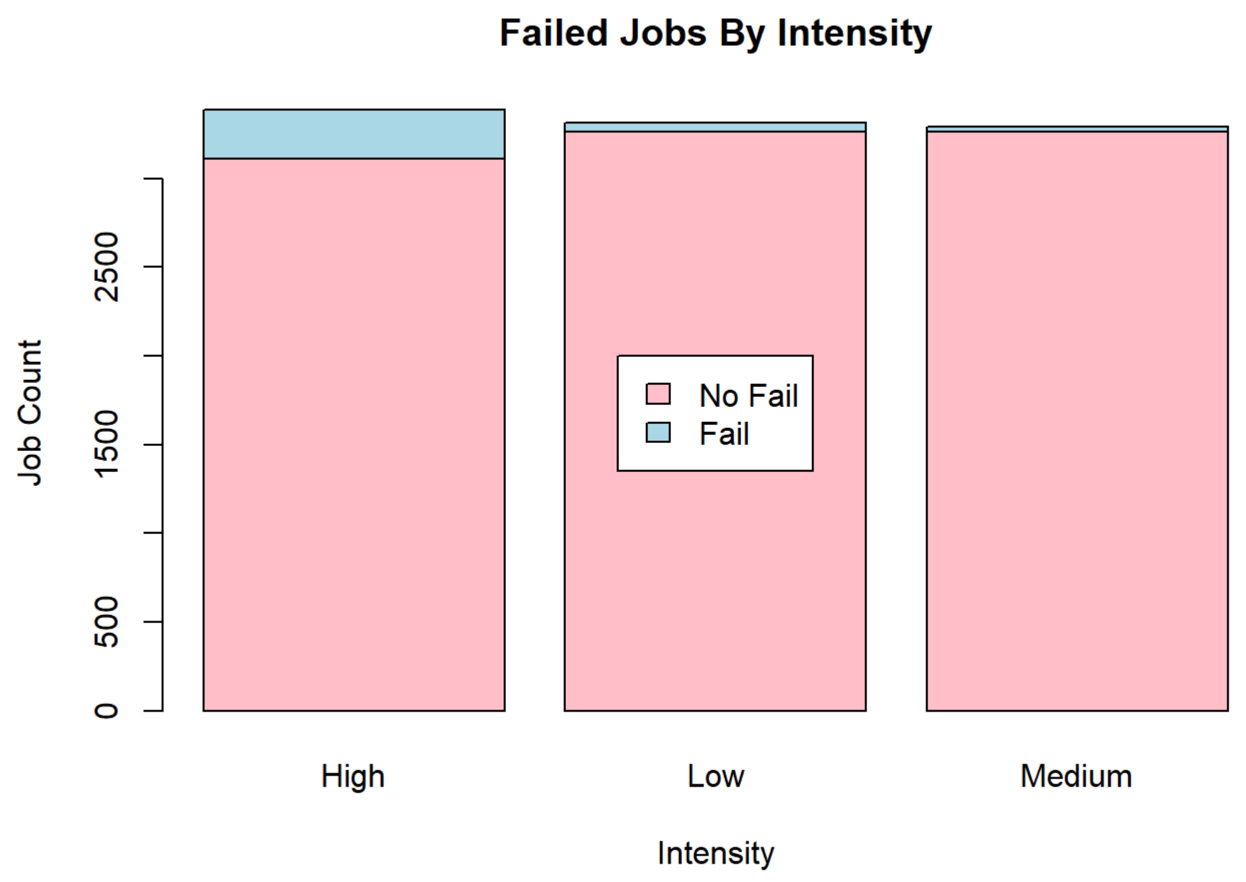
```
##
##             High   Low Medium
##   No Fail 0.997 0.000  0.966
##   Fail    0.003 1.000  0.034
```

The barplot shows that High-Quality and Medium Quality machines have relatively low fail rates with high failing about 0.3% of the time and medium failing about 3.4% of the time. Meanwhile Low-Quality machines fail at a surprising rate of 100% of the time. This relationship will be extremely useful in building our prediction models because all Low-Quality machines Fail and only a very small percentage of High and Medium Failing.

Relationship Between Intensity and Failure

Next we explored the relationship between Intensity and Failure. We first created the variable int_table indicating the data needed from both failed and intensity to understand what is going on between the two variables. We also created a legend for the barplot and used a probability table again.

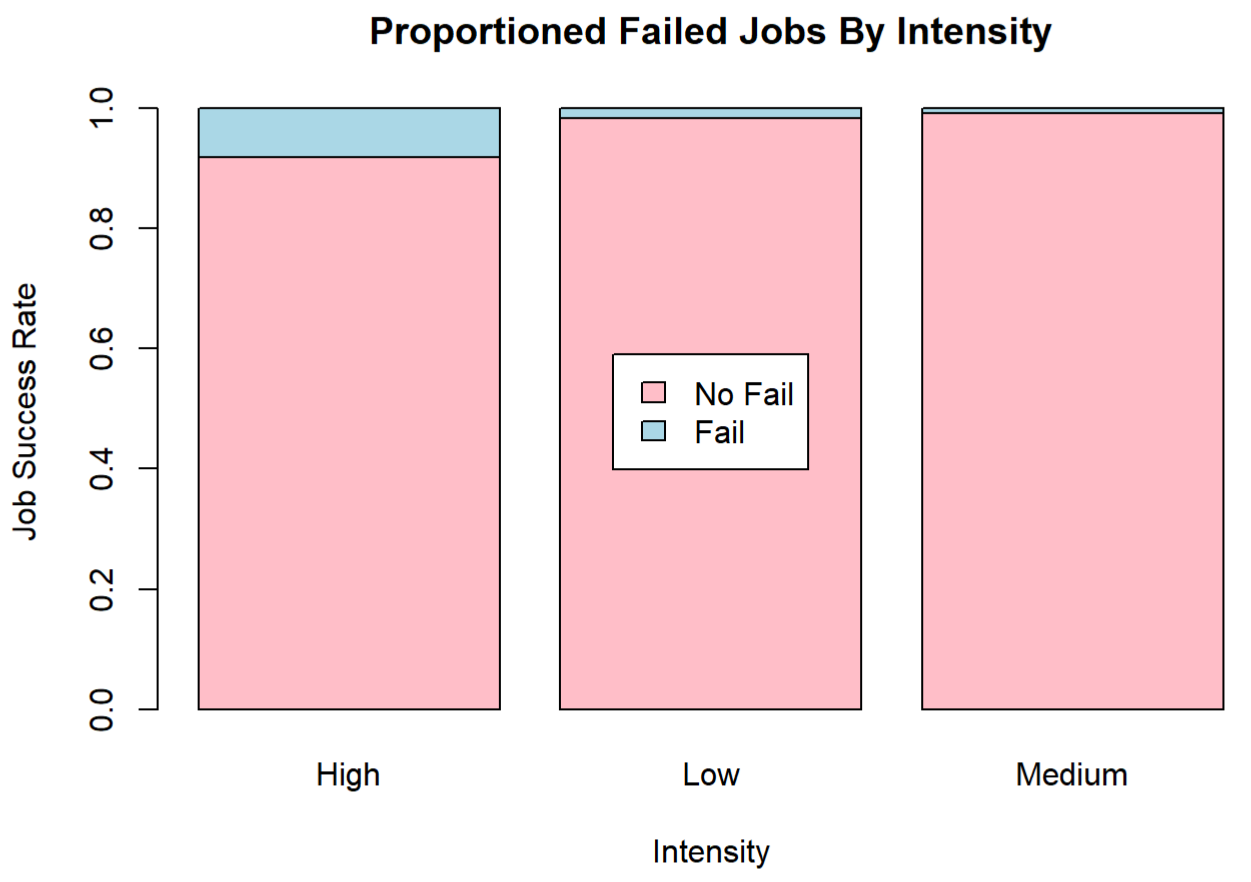
```
int_table <- table(data$Failed, data$Intensity)

barplot(int_table, main = "Failed Jobs By Intensity",
        xlab = "Intensity", ylab = "Job Count",
        col = c("pink","lightblue"), )

legend("center",legend = c("No Fail", "Fail"), fill = c("pink","lightblue"))
```



```
int_prop <- prop.table(int_table, margin = 2)
rownames(int_prop) <- c("No Fail", "Fail")
round(int_prop, 3)
```

```
##
##            High   Low Medium
##   No Fail 0.919 0.984  0.992
##   Fail    0.081 0.016  0.008
```

```
barplot(prop.table(int_table, margin = 2),
        main = "Proportioned Failed Jobs By Intensity",
        xlab = "Intensity", ylab = "Job Success Rate",
        col = c("pink","lightblue"))

legend("center",legend = c("No Fail", "Fail"), fill = c("pink","lightblue"))
```



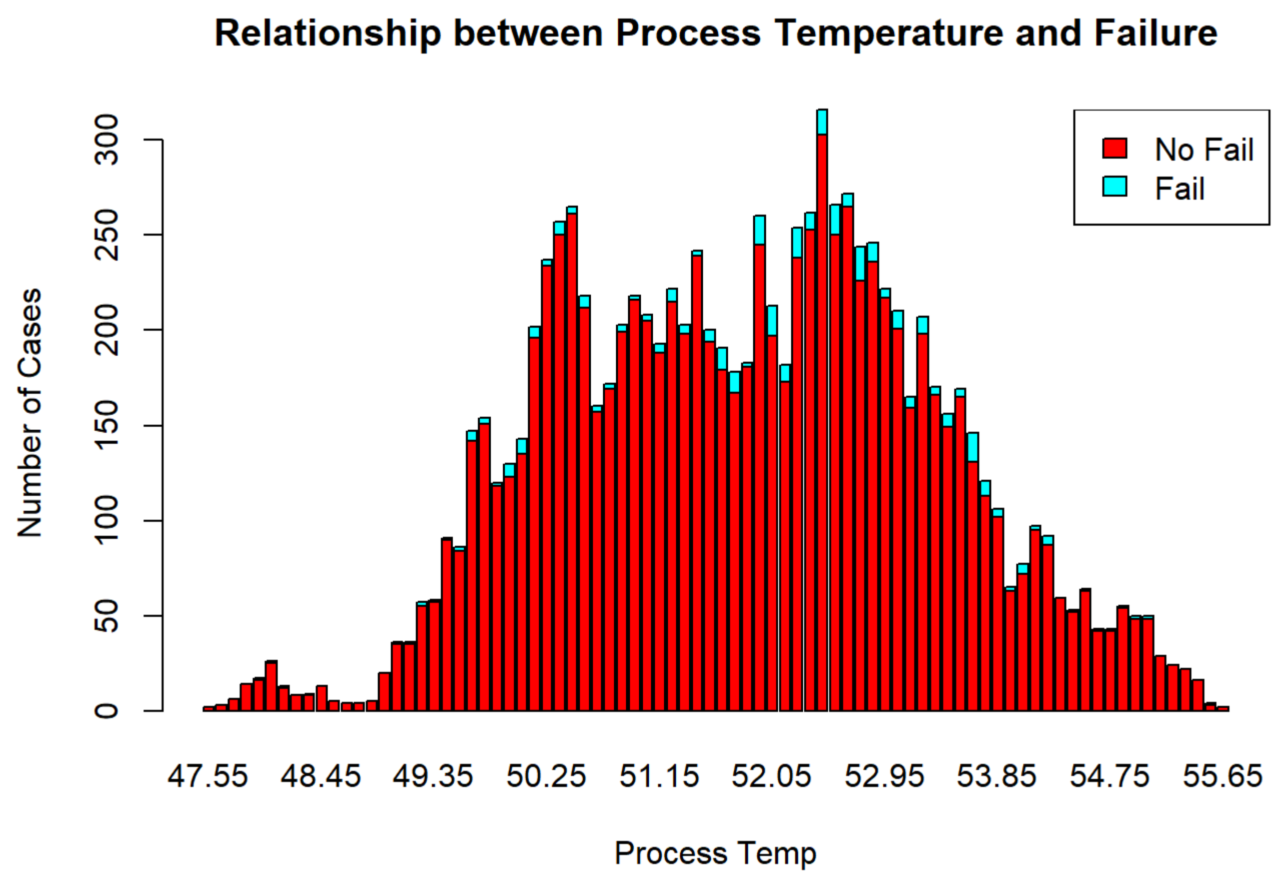
We first created a graph based on the number of jobs failed for each intensity, knowing there is a fairly equal split between jobs in “High” “Medium” and “Low” intensity types at ~33% and this graph is okay. However, to be more accurate we then produced a relative frequency chart and used this to create a graph. This graph is more accurate even though it shows similar results because each type is weighted based on the number of jobs in that category.

From this graph, we can tell the intensity of a machine has a huge impact on whether it will fail or not. The higher intensity of a job the more likely it is for the job to fail. As we can see the proportion of blue in the graph is larger in high jobs.

Relationship between Process Temperature and Failure

Next, we wanted to explore the relationship between one of the numerical variables, Process Temp, and the target variable Failed. We first started by plotting the relationship on a barplot.

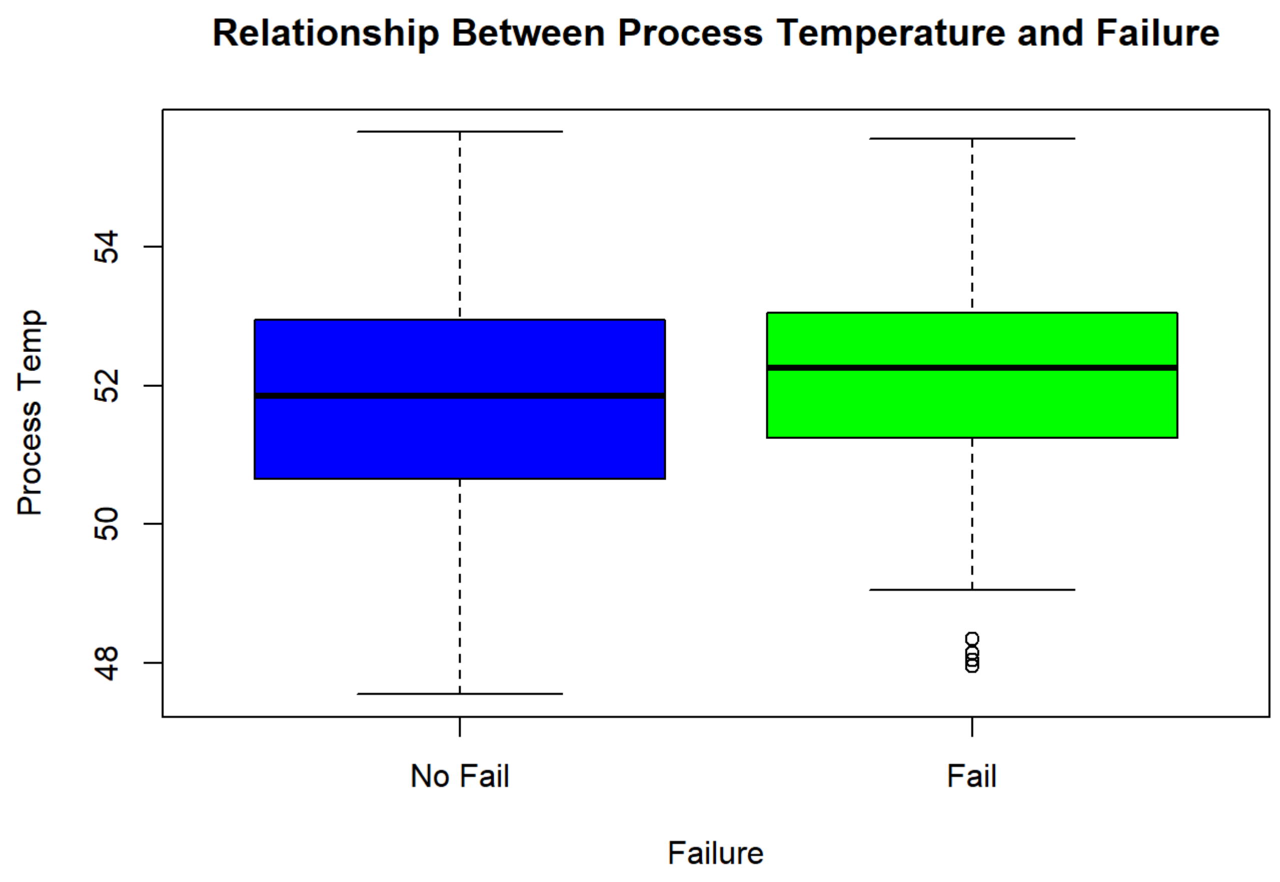
```
temp_table <- table(data$Failed, data$ProcessTemp)

rownames(temp_table) <- c("No Fail", "Fail")

barplot(temp_table,
        main = "Relationship between Process Temperature and Failure",
        ylab = "Number of Cases",
        xlab = "Process Temp",
        col = rainbow(2))

legend("topright",legend = c("No Fail", "Fail"), fill = rainbow(2) )
```



Looking at the graph we can tell visually that the temps in the middle have a higher percentage of failure as opposed to the lower and higher temps. However, we felt the graph does not give us a clear understanding on what the data is showing.

For us to try and get a clearer understanding, we thought a smaller dataset would be easier to visualize so we split up the data in half and used the first half of the data and visualized it through a plot this time.

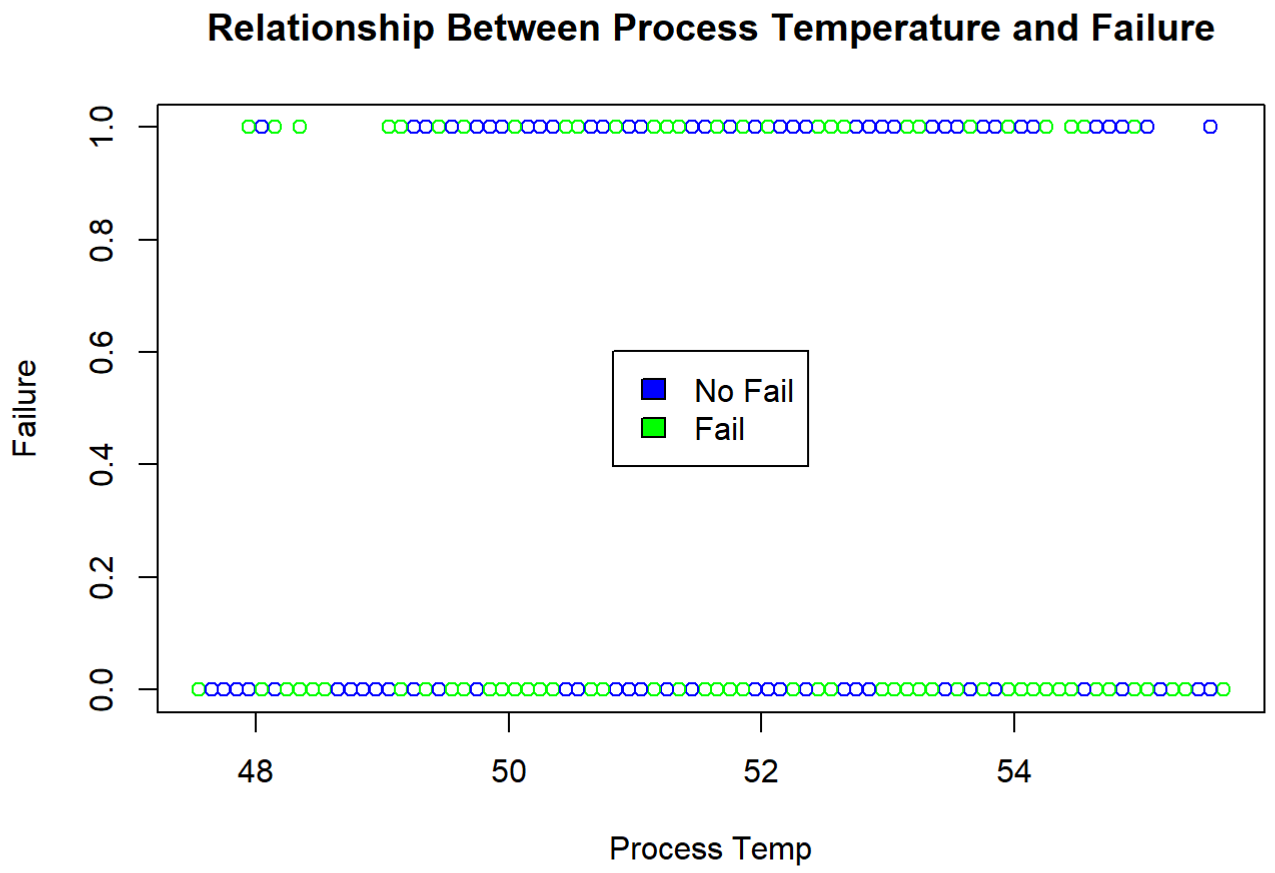
```
temp_var1 <- as.factor(data$Failed)
plot(temp_var1,data$ProcessTemp, xlab = "Failure", ylab = "Process Temp",
     main = "Relationship Between Process Temperature and Failure",
     col = c("blue", "green"),
     xaxt = "n" )

axis(side = 1, at = c(1, 2), labels = c("No Fail", "Fail") )
```



In this graph, we noticed that the max process temperature for machine that would fail or not was about the same, while the middle temperature was slightly higher for the machines that fail. The min temperature was also noticebly higher for machines that fail. While these results were encourging, we still wanted to see if there was a better way to view the relationship between the variables.

Next, we tried to visulaize the data through a plot by the failure rate.

```
plot(data$ProcessTemp, data$Failed, xlab = "Process Temp", ylab = "Failure",
     main = "Relationship Between Process Temperature and Failure",
     col = c("blue", "green"))

legend("center",legend = c("No Fail", "Fail"), fill = c("blue","green"))
```

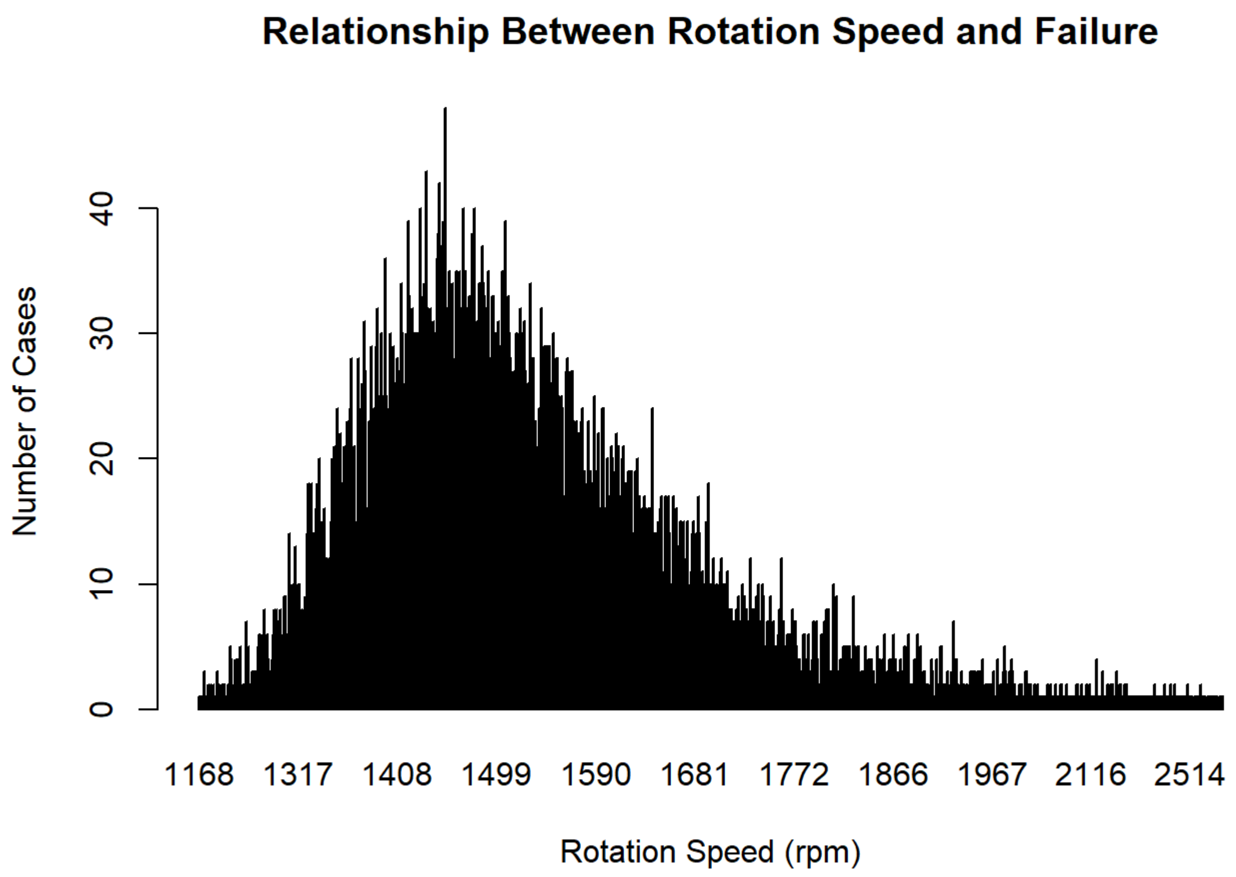


However, we did not get the desired results from this plot and therefore concluded that the other graphs were likely more useful in visualizing this relationship.

Relationship Between Rotation Speed and Failure

Next, we decided to closely look at the relationship between failure and rotation speed, which is another numerical variable.

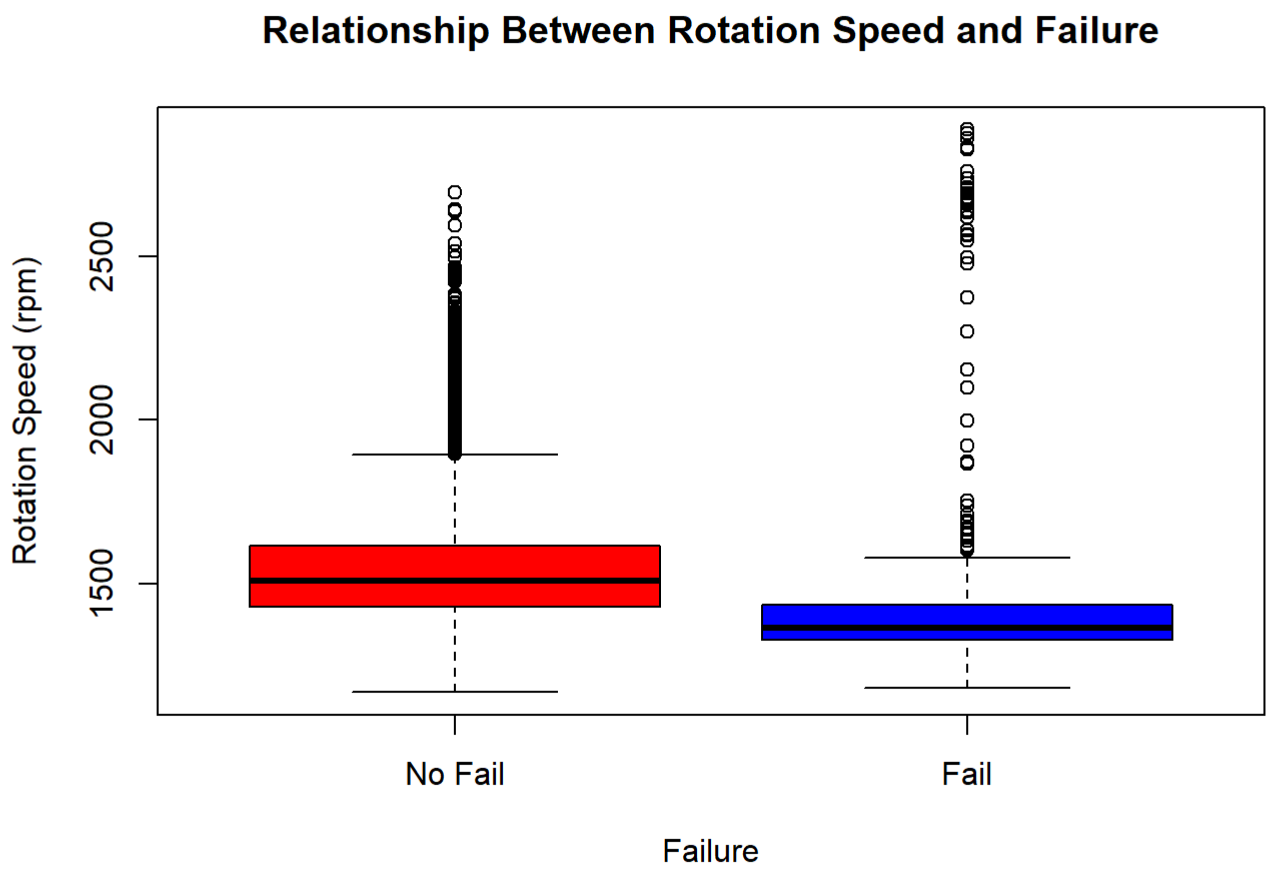
```r
ro_speed <- table(data$Failed,data$RotationSpeed)

barplot(ro_speed, col = rainbow(2),
     main = "Relationship Between Rotation Speed and Failure",
     xlab = "Rotation Speed (rpm)",
     ylab = "Number of Cases")
```



We noticed the graph represents the values of rotation speed well but is difficult to clearly see the failure aspect which tells us we need to switch up the diagram and add more features to the graph for a better representation.

After our first trial attempt, we then decided to display this relationship using a plot as it seems to represent our data well.

```
temp_var1 <- as.factor(data$Failed)

plot(temp_var1,data$RotationSpeed,xlab = "Failure", ylab = "Rotation Speed (rpm)",
     main = "Relationship Between Rotation Speed and Failure",
     col = c("red", "blue"),
     xaxt = "n")

axis(side = 1, at = c(1, 2), labels = c("No Fail", "Fail") )
```



In this graph, we noticed that the Q3 rotation Speed for machine that would fail was around the Q1 for rotation speed for machine that would not fail. We also noticed that the middle rotation speed for machines that would fail was lower than machines with no failure. We can roughly conclude that machines that fail likely have a lower rotation speed. Lastly, there were a substantial number of outliers on both boxplots which represents the extreme distribution.

Prediction Models

After exploring the variables, and some of the relationships between the variables, it was now time to build our prediction models which will be built using our independent variables and predict whether a machine would fail or not based on its characteristics.

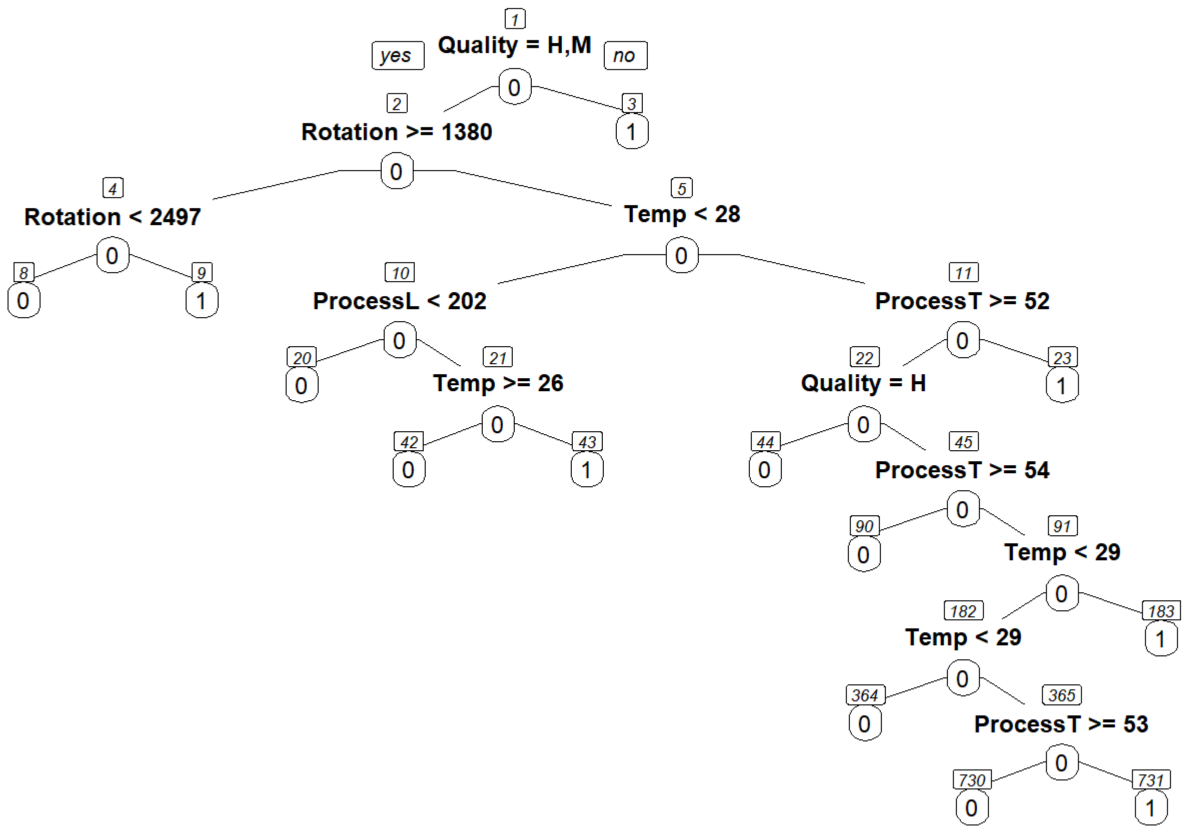
Prediction Model 1 - Classification Tree

Before we built our tree, we had to convert the categorical variables to factors by using the as.factor() function. We then used set.seed() so our model used the same set of random number each time. Lastly, we partitioned the data by creating the training index using the target variable and proportioning 65% of the data to the training set. The validation set was created by using the remaining 35% of data.

```
data$Failed <- as.factor(data$Failed)
data$Quality <- as.factor(data$Quality)
data$Intensity <- as.factor(data$Intensity)

set.seed(1)
train_index <- createDataPartition(data$Failed, p = 0.65, list = FALSE)

train_set <- data[train_index,]
val_set <- data[-train_index, ]
```

We then built the model tree by using the rpart() function, which required the target and predictor variables, and the class method. Finally we were able to visualize the tree using the prp() function.

```
model_tree <- rpart(formula = Failed ~ Quality + Temp + ProcessTemp +
                    RotationSpeed + Intensity + ProcessLength,
                    data = train_set, method = "class")

prp(model_tree, type = 1, nn = TRUE)
```



The classification tree depicted above shows a breakdown of Failed jobs/processes as the target variable based on predictors: Quality, Temp, ProcessTemp, RotationSpeed, Intensity, and ProcessLength.

The tree has 12 internal nodes and 13 leaf or ending nodes. Overall looking at the summary table we have a large complexity parameter at 0.447 for the row 1, coming down to the average of 0.01 by row 7, as anticipated ending the tree. Xerror is at its lowest in row 5, getting smaller from row 1-5, then increasing in rows 6 & 7. Our goals should be to end the tree before error starts to increase again to prevent overfitting / and decrease our sensitivity. The order of importance for the variables as in how much the variable is used to make a decision is: 1-Quality, 2-ProcessTemp, 3-Temp, 4-RotationSpeed, and 5-ProcessLength.

Summary for Model Tree

```
summary(model_tree)
```

```
## Call:
## rpart(formula = Failed ~ Quality + Temp + ProcessTemp + RotationSpeed +
##     Intensity + ProcessLength, data = train_set, method = "class")
##   n= 6495
##
##           CP nsplit rel error    xerror       xstd
## 1 0.44782609      0 1.0000000 1.0000000 0.06476003
## 2 0.04202899      1 0.5521739 0.5521739 0.04851611
## 3 0.02608696      4 0.4260870 0.4304348 0.04292936
## 4 0.02463768      5 0.4000000 0.4217391 0.04250016
## 5 0.01304348      8 0.3260870 0.3347826 0.03792519
## 6 0.01086957     10 0.3000000 0.3391304 0.03816769
## 7 0.01000000     12 0.2782609 0.3391304 0.03816769
##
## Variable importance
##       Quality   ProcessTemp          Temp RotationSpeed ProcessLength
##            56            15            14             9             4
##
## Node number 1: 6495 observations,    complexity param=0.4478261
##   predicted class=0  expected loss=0.03541186  P(node) =1
##     class counts:  6265   230
##    probabilities: 0.965 0.035
##   left son=2 (6392 obs) right son=3 (103 obs)
##   Primary splits:
##       Quality       splits as  LRL,        improve=194.757200, (0 missing)
##       RotationSpeed < 1379.5 to the right, improve= 34.068100, (0 missing)
##       ProcessLength < 202.5  to the left,  improve= 19.562620, (0 missing)
##       Intensity     splits as  RLL,        improve= 14.044790, (0 missing)
##       Temp          < 28.5   to the left,  improve=  5.140108, (0 missing)
##   Surrogate splits:
##       ProcessLength < 247.5  to the left,  agree=0.984, adj=0.019, (0 split)
##       RotationSpeed < 2556   to the left,  agree=0.984, adj=0.010, (0 split)
##
## Node number 2: 6392 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.01986859  P(node) =0.9841416
##     class counts:  6265   127
##    probabilities: 0.980 0.020
##   left son=4 (5565 obs) right son=5 (827 obs)
##   Primary splits:
```

```
##       RotationSpeed < 1379.5 to the right, improve=15.862800, (0 missing)
##       Intensity     splits as  RLL,        improve= 4.974525, (0 missing)
##       ProcessLength < 202.5  to the left,  improve= 4.087266, (0 missing)
##       Temp          < 28.4   to the left,  improve= 3.275935, (0 missing)
##       Quality       splits as  L-R,        improve= 3.123334, (0 missing)
## 
## Node number 3: 103 observations
##   predicted class=1  expected loss=0  P(node) =0.01585835
##     class counts:     0   103
##    probabilities: 0.000 1.000 
## 
## Node number 4: 5565 observations,    complexity param=0.02608696
##   predicted class=0  expected loss=0.006289308  P(node) =0.8568129
##     class counts:  5530    35
##    probabilities: 0.994 0.006 
##   left son=8 (5555 obs) right son=9 (10 obs)
##   Primary splits:
##       RotationSpeed < 2496.5 to the left,  improve=12.6222100, (0 missing)
##       ProcessLength < 224.5  to the left,  improve= 2.6065350, (0 missing)
##       Quality       splits as  L-R,        improve= 0.1396827, (0 missing)
##       ProcessTemp   < 49.9   to the right, improve= 0.1308280, (0 missing)
##       Temp          < 23.5   to the right, improve= 0.1266745, (0 missing)
## 
## Node number 5: 827 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.1112455  P(node) =0.1273287
##     class counts:   735    92
##    probabilities: 0.889 0.111 
##   left son=10 (634 obs) right son=11 (193 obs)
##   Primary splits:
##       Temp          < 28.4   to the left,  improve=24.4496200, (0 missing)
##       Quality       splits as  L-R,        improve=13.4058700, (0 missing)
##       ProcessLength < 193.5  to the left,  improve=10.6184400, (0 missing)
##       ProcessTemp   < 51.7   to the left,  improve= 4.2886570, (0 missing)
##       RotationSpeed < 1286.5 to the right, improve= 0.5100265, (0 missing)
##   Surrogate splits:
##       ProcessTemp < 53.5   to the left,  agree=0.834, adj=0.29, (0 split)
## 
## Node number 8: 5555 observations
##   predicted class=0  expected loss=0.004860486  P(node) =0.8552733
##     class counts:  5528    27
##    probabilities: 0.995 0.005 
## 
## Node number 9: 10 observations
##   predicted class=1  expected loss=0.2  P(node) =0.001539646
##     class counts:     2     8
##    probabilities: 0.200 0.800 
## 
## Node number 10: 634 observations,    complexity param=0.01086957
##   predicted class=0  expected loss=0.04416404  P(node) =0.09761355
##     class counts:   606    28
##    probabilities: 0.956 0.044 
##   left son=20 (599 obs) right son=21 (35 obs)
##   Primary splits:
##       ProcessLength < 201.5  to the left,  improve=7.9352090, (0 missing)
##       Quality       splits as  L-R,        improve=1.6642430, (0 missing)
##       Temp          < 26.2   to the right, improve=0.9645272, (0 missing)
##       RotationSpeed < 1268.5 to the right, improve=0.8391343, (0 missing)
##       ProcessTemp   < 52.3   to the right, improve=0.7073413, (0 missing)
## 
## Node number 11: 193 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.3316062  P(node) =0.02971517
##     class counts:   129    64
##    probabilities: 0.668 0.332 
##   left son=22 (162 obs) right son=23 (31 obs)
##   Primary splits:
##       ProcessTemp   < 52.4   to the right, improve=29.8905300, (0 missing)
##       Quality       splits as  L-R,        improve=17.7897000, (0 missing)
##       ProcessLength < 186.5  to the left,  improve= 3.6165870, (0 missing)
##       Temp          < 30.5   to the right, improve= 3.5387970, (0 missing)
##       RotationSpeed < 1338.5 to the right, improve= 0.4153887, (0 missing)
## 
## Node number 20: 599 observations
##   predicted class=0  expected loss=0.02504174  P(node) =0.09222479
##     class counts:   584    15
##    probabilities: 0.975 0.025 
## 
## Node number 21: 35 observations,    complexity param=0.01086957
##   predicted class=0  expected loss=0.3714286  P(node) =0.005388761
##     class counts:    22    13
##    probabilities: 0.629 0.371
##   left son=42 (14 obs) right son=43 (21 obs)
##   Primary splits:
##       Temp          < 26.1   to the right, improve=6.4380950, (0 missing)
##       Quality       splits as  L-R,        improve=4.4261900, (0 missing)
##       ProcessTemp   < 50.7   to the right, improve=3.1834370, (0 missing)
##       RotationSpeed < 1327   to the left,  improve=1.2595240, (0 missing)
##       ProcessLength < 214.5  to the left,  improve=0.4761905, (0 missing)
##   Surrogate splits:
##       ProcessTemp   < 52.45  to the right, agree=0.800, adj=0.500, (0 split)
##       Quality       splits as  L-R,        agree=0.743, adj=0.357, (0 split)
##       RotationSpeed < 1323.5 to the left,  agree=0.629, adj=0.071, (0 split)
##       Intensity     splits as  R-L,        agree=0.629, adj=0.071, (0 split)
##       ProcessLength < 213.5  to the left,  agree=0.629, adj=0.071, (0 split)
##
## Node number 22: 162 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.2098765  P(node) =0.02494226
##     class counts:   128    34
##    probabilities: 0.790 0.210
##   left son=44 (57 obs) right son=45 (105 obs)
##   Primary splits:
##       Quality       splits as  L-R,        improve=7.7474430, (0 missing)
##       ProcessTemp   < 53.7   to the right, improve=4.3573060, (0 missing)
##       ProcessLength < 186.5  to the left,  improve=3.4224170, (0 missing)
##       Temp          < 28.9   to the left,  improve=3.3772500, (0 missing)
##       RotationSpeed < 1338.5 to the right, improve=0.5987882, (0 missing)
##   Surrogate splits:
##       Temp          < 30.7   to the right, agree=0.667, adj=0.053, (0 split)
##       ProcessLength < 9.5    to the left,  agree=0.667, adj=0.053, (0 split)
##       ProcessTemp   < 55.1   to the right, agree=0.660, adj=0.035, (0 split)
##
## Node number 23: 31 observations
##   predicted class=1  expected loss=0.03225806  P(node) =0.004772902
##     class counts:     1    30
##    probabilities: 0.032 0.968
##
## Node number 42: 14 observations
##   predicted class=0  expected loss=0  P(node) =0.002155504
##     class counts:    14     0
##    probabilities: 1.000 0.000 
## 
## Node number 43: 21 observations
##   predicted class=1  expected loss=0.3809524  P(node) =0.003233256
##     class counts:     8    13
##    probabilities: 0.381 0.619 
## 
## Node number 44: 57 observations
##   predicted class=0  expected loss=0  P(node) =0.008775982
##     class counts:    57     0
##    probabilities: 1.000 0.000 
## 
## Node number 45: 105 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.3238095  P(node) =0.01616628
##     class counts:    71    34
##    probabilities: 0.676 0.324 
##   left son=90 (43 obs) right son=91 (62 obs)
##   Primary splits:
##       ProcessTemp   < 53.7   to the right, improve=6.2727750, (0 missing)
##       Temp          < 28.9   to the left,  improve=4.2536800, (0 missing)
##       ProcessLength < 184    to the left,  improve=3.4247730, (0 missing)
##       RotationSpeed < 1351   to the right, improve=0.5365079, (0 missing)
##   Surrogate splits:
##       Temp          < 29.3   to the right, agree=0.80, adj=0.512, (0 split)
##       ProcessLength < 166.5  to the right, agree=0.61, adj=0.047, (0 split)
##       RotationSpeed < 1339   to the right, agree=0.60, adj=0.023, (0 split)
## 
## Node number 90: 43 observations
##   predicted class=0  expected loss=0.1162791  P(node) =0.006620477
##     class counts:    38     5
##    probabilities: 0.884 0.116 
## 
## Node number 91: 62 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.4677419  P(node) =0.009545804
##     class counts:    33    29
##    probabilities: 0.532 0.468 
##   left son=182 (41 obs) right son=183 (21 obs)
##   Primary splits:
```

```
##       Temp          < 29.3   to the left,  improve=12.1299700, (0 missing)
##       ProcessLength < 184    to the left,  improve= 4.4555830, (0 missing)
##       RotationSpeed < 1313   to the left,  improve= 1.6614440, (0 missing)
##       ProcessTemp   < 53.6   to the left,  improve= 0.9592794, (0 missing)
##   Surrogate splits:
##       ProcessTemp   < 53.4   to the left,  agree=0.758, adj=0.286, (0 split)
##       ProcessLength < 33     to the right, agree=0.710, adj=0.143, (0 split)
##       RotationSpeed < 1254.5 to the right, agree=0.677, adj=0.048, (0 split)
## 
## Node number 182: 41 observations,    complexity param=0.01304348
##   predicted class=0  expected loss=0.2439024  P(node) =0.006312548
##     class counts:    31    10
##    probabilities: 0.756 0.244 
##   left son=364 (17 obs) right son=365 (24 obs)
##   Primary splits:
##       Temp          < 28.9   to the left,  improve=3.455285, (0 missing)
##       ProcessTemp   < 52.6   to the right, improve=2.887103, (0 missing)
##       RotationSpeed < 1325.5 to the left,  improve=2.529359, (0 missing)
##       ProcessLength < 147    to the left,  improve=2.240007, (0 missing)
##   Surrogate splits:
##       RotationSpeed < 1325.5 to the left,  agree=0.683, adj=0.235, (0 split)
##       ProcessLength < 62.5   to the left,  agree=0.634, adj=0.118, (0 split)
##       ProcessTemp   < 52.9   to the left,  agree=0.610, adj=0.059, (0 split)
##       Intensity     splits as  R-L,        agree=0.610, adj=0.059, (0 split)
## 
## Node number 183: 21 observations
##   predicted class=1  expected loss=0.0952381  P(node) =0.003233256
##     class counts:     2    19
##    probabilities: 0.095 0.905 
## 
## Node number 364: 17 observations
##   predicted class=0  expected loss=0  P(node) =0.002617398
##     class counts:    17     0
##    probabilities: 1.000 0.000 
## 
## Node number 365: 24 observations,    complexity param=0.01304348
##   predicted class=0  expected loss=0.4166667  P(node) =0.00369515
##     class counts:    14    10
##    probabilities: 0.583 0.417
##   left son=730 (12 obs) right son=731 (12 obs)
##   Primary splits:
##       ProcessTemp   < 52.9   to the right, improve=5.3333330, (0 missing)
##       RotationSpeed < 1331.5 to the left,  improve=2.0416670, (0 missing)
##       ProcessLength < 87     to the left,  improve=2.0416670, (0 missing)
##       Temp          < 29.1   to the right, improve=0.5555556, (0 missing)
##   Surrogate splits:
##       Temp          < 29.1   to the right, agree=0.708, adj=0.417, (0 split)
##       RotationSpeed < 1343.5 to the left,  agree=0.667, adj=0.333, (0 split)
##       ProcessLength < 87     to the left,  agree=0.583, adj=0.167, (0 split)
##
## Node number 730: 12 observations
##   predicted class=0  expected loss=0.08333333  P(node) =0.001847575
##     class counts:    11     1
##    probabilities: 0.917 0.083
##
## Node number 731: 12 observations
##   predicted class=1  expected loss=0.25  P(node) =0.001847575
##     class counts:     3     9
##    probabilities: 0.250 0.750
```

Confusion Matrix for Model Tree

Here, we cross validate the model by predicting the result of the validation set and use a confusion matrix to show the performance statistics of the tree.

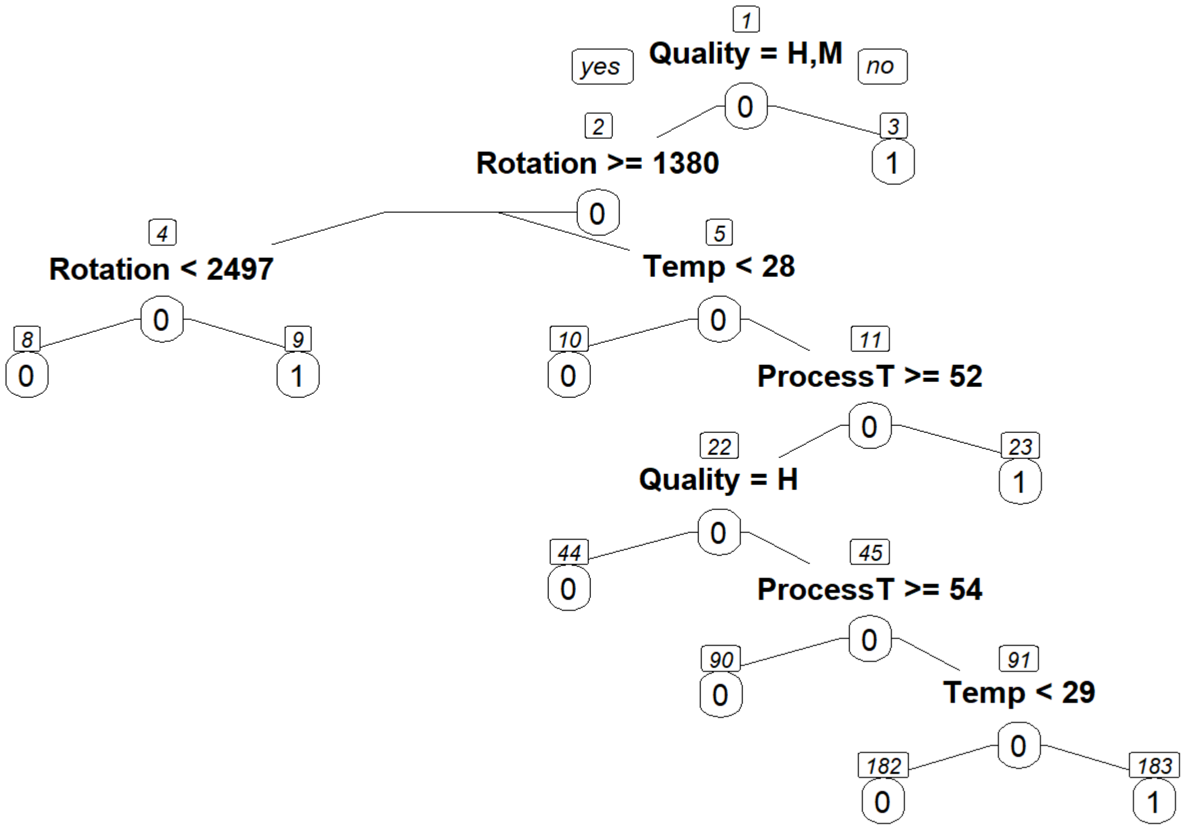
```
predicted_classes <- predict(object = model_tree, newdata =
val_set, type = "class")

probis <- predict(object = model_tree, newdata = val_set,
type = "prob")

confusionMatrix(predicted_classes, val_set$Failed,
positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3356   32
##          1   17   91
##
##                Accuracy : 0.986
##                  95% CI : (0.9815, 0.9896)
##     No Information Rate : 0.9648
##     P-Value [Acc > NIR] : 1.167e-14
##
##                   Kappa : 0.7807
##
##  Mcnemar's Test P-Value : 0.0455
##
##             Sensitivity : 0.73984
##             Specificity : 0.99496
##          Pos Pred Value : 0.84259
##          Neg Pred Value : 0.99055
##              Prevalence : 0.03518
##          Detection Rate : 0.02603
##    Detection Prevalence : 0.03089
##       Balanced Accuracy : 0.86740
##
##        'Positive' Class : 1
##
```

In reviewing our confusion matrix with "1" as our positive cases, indicated a failed job. The matrix for classification tree 1 (model_tree) shows we have correctly identified 3356 jobs as Not-Failed, and 91 jobs as Failed; compared to incorrectly predicting 17 jobs as Not-Failed, and 32 jobs as Failed when they didn't. An accuracy of 99%. A precision of 84%, (0.84), and true positivity/recall of 74% (0.74); resulting in an F-Score of ((2 x 0.74 x 0.84) / (0.74 + 0.84)) = 0.786. This is not the best and is due to a low precision indicating we have a high number of false positives, despite our model being relatively good at indicating if a project will Not-Fail as shown in a high specificity of 0.99.

Model Tree Version 2

```
model_tree_2 <- rpart(formula = Failed ~ Quality + Temp + ProcessTemp +
                          RotationSpeed + Intensity + ProcessLength,
                          data = train_set, method = "class", cp = 0.014)

prp(model_tree_2, type = 1, nn = TRUE)
```



Taking a closer look at the second (adjusted) classification tree, we see the tree is very similar, however we have adjusted to the cp (complexity parameter) to 0.014, resulting in an adjusted xerror that gets smaller from rows 1 to 5 and overall shortens the tree and addresses our previous overfitting. The new tree has only 8 internal nodes and 9 leaf or ending nodes. The order of importance for the variables as in how much the variable is used to make a decision is: 1-Quality, 2-ProcessTemp, 3-Temp, 4-RotationSpeed, and 5-ProcessLength with more importance being placed on the variable Quality, compared to the first classification tree.

Summary for Model Tree Version 2

```
summary(model_tree_2)
```

```
## Call:
## rpart(formula = Failed ~ Quality + Temp + ProcessTemp + RotationSpeed +
##     Intensity + ProcessLength, data = train_set, method = "class",
##     cp = 0.014)
##   n= 6495
##
##           CP nsplit rel error    xerror       xstd
## 1 0.44782609      0 1.0000000 1.0000000 0.06476003
## 2 0.04202899      1 0.5521739 0.5521739 0.04851611
## 3 0.02608696      4 0.4260870 0.4347826 0.04314226
## 4 0.02463768      5 0.4000000 0.4260870 0.04271533
## 5 0.01400000      8 0.3260870 0.3695652 0.03982182
##
## Variable importance
##       Quality   ProcessTemp          Temp RotationSpeed ProcessLength
##            62            14            12            10             2
##
## Node number 1: 6495 observations,    complexity param=0.4478261
##   predicted class=0  expected loss=0.03541186  P(node) =1
##     class counts:  6265   230
##    probabilities: 0.965 0.035
##   left son=2 (6392 obs) right son=3 (103 obs)
##   Primary splits:
##       Quality       splits as  LRL,        improve=194.757200, (0 missing)
##       RotationSpeed < 1379.5 to the right, improve= 34.068100, (0 missing)
##       ProcessLength < 202.5  to the left,  improve= 19.562620, (0 missing)
##       Intensity     splits as  RLL,        improve= 14.044790, (0 missing)
##       Temp          < 28.5   to the left,  improve=  5.140108, (0 missing)
##   Surrogate splits:
##       ProcessLength < 247.5  to the left,  agree=0.984, adj=0.019, (0 split)
##       RotationSpeed < 2556   to the left,  agree=0.984, adj=0.010, (0 split)
##
## Node number 2: 6392 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.01986859  P(node) =0.9841416
##     class counts:  6265   127
##    probabilities: 0.980 0.020
##   left son=4 (5565 obs) right son=5 (827 obs)
##   Primary splits:
##       RotationSpeed < 1379.5 to the right, improve=15.862800, (0 missing)
##       Intensity     splits as  RLL,        improve= 4.974525, (0 missing)
##       ProcessLength < 202.5  to the left,  improve= 4.087266, (0 missing)
##       Temp          < 28.4   to the left,  improve= 3.275935, (0 missing)
##       Quality       splits as  L-R,        improve= 3.123334, (0 missing)
##
## Node number 3: 103 observations
##   predicted class=1  expected loss=0  P(node) =0.01585835
##     class counts:     0   103
##    probabilities: 0.000 1.000
##
## Node number 4: 5565 observations,    complexity param=0.02608696
##   predicted class=0  expected loss=0.006289308  P(node) =0.8568129
##     class counts:  5530    35
##    probabilities: 0.994 0.006
##   left son=8 (5555 obs) right son=9 (10 obs)
##   Primary splits:
##       RotationSpeed < 2496.5 to the left,  improve=12.6222100, (0 missing)
##       ProcessLength < 224.5  to the left,  improve= 2.6065350, (0 missing)
##       Quality       splits as  L-R,        improve= 0.1396827, (0 missing)
##       ProcessTemp   < 49.9   to the right, improve= 0.1308280, (0 missing)
##       Temp          < 23.5   to the right, improve= 0.1266745, (0 missing)
##
## Node number 5: 827 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.1112455  P(node) =0.1273287
##     class counts:   735    92
##    probabilities: 0.889 0.111
##   left son=10 (634 obs) right son=11 (193 obs)
##   Primary splits:
##       Temp          < 28.4   to the left,  improve=24.4496200, (0 missing)
##       Quality       splits as  L-R,        improve=13.4058700, (0 missing)
##       ProcessLength < 193.5  to the left,  improve=10.6184400, (0 missing)
##       ProcessTemp   < 51.7   to the left,  improve= 4.2886570, (0 missing)
##       RotationSpeed < 1286.5 to the right, improve= 0.5100265, (0 missing)
##   Surrogate splits:
##       ProcessTemp < 53.5   to the left,  agree=0.834, adj=0.29, (0 split)
##
## Node number 8: 5555 observations
##   predicted class=0  expected loss=0.004860486  P(node) =0.8552733
##     class counts:  5528    27
##    probabilities: 0.995 0.005
##
## Node number 9: 10 observations
##   predicted class=1  expected loss=0.2  P(node) =0.001539646
##     class counts:     2     8
##    probabilities: 0.200 0.800
##
## Node number 10: 634 observations
##   predicted class=0  expected loss=0.04416404  P(node) =0.09761355
##     class counts:   606    28
##    probabilities: 0.956 0.044
##
## Node number 11: 193 observations,    complexity param=0.04202899
##   predicted class=0  expected loss=0.3316062  P(node) =0.02971517
##     class counts:   129    64
##    probabilities: 0.668 0.332
##   left son=22 (162 obs) right son=23 (31 obs)
##   Primary splits:
##       ProcessTemp   < 52.4   to the right, improve=29.8905300, (0 missing)
##       Quality       splits as  L-R,        improve=17.7897000, (0 missing)
##       ProcessLength < 186.5  to the left,  improve= 3.6165870, (0 missing)
##       Temp          < 30.5   to the right, improve= 3.5387970, (0 missing)
##       RotationSpeed < 1338.5 to the right, improve= 0.4153887, (0 missing)
##
## Node number 22: 162 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.2098765  P(node) =0.02494226
##     class counts:   128    34
##    probabilities: 0.790 0.210
##   left son=44 (57 obs) right son=45 (105 obs)
##   Primary splits:
##       Quality       splits as  L-R,        improve=7.7474430, (0 missing)
##       ProcessTemp   < 53.7   to the right, improve=4.3573060, (0 missing)
##       ProcessLength < 186.5  to the left,  improve=3.4224170, (0 missing)
##       Temp          < 28.9   to the left,  improve=3.3772500, (0 missing)
##       RotationSpeed < 1338.5 to the right, improve=0.5987882, (0 missing)
##   Surrogate splits:
##       Temp          < 30.7   to the right, agree=0.667, adj=0.053, (0 split)
##       ProcessLength < 9.5    to the left,  agree=0.667, adj=0.053, (0 split)
##       ProcessTemp   < 55.1   to the right, agree=0.660, adj=0.035, (0 split)
```

```
##
## Node number 23: 31 observations
##   predicted class=1  expected loss=0.03225806  P(node) =0.004772902
##     class counts:     1    30
##    probabilities: 0.032 0.968
##
## Node number 44: 57 observations
##   predicted class=0  expected loss=0  P(node) =0.008775982
##     class counts:    57     0
##    probabilities: 1.000 0.000
##
## Node number 45: 105 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.3238095  P(node) =0.01616628
##     class counts:    71    34
##    probabilities: 0.676 0.324
##   left son=90 (43 obs) right son=91 (62 obs)
##   Primary splits:
##       ProcessTemp   < 53.7   to the right, improve=6.2727750, (0 missing)
##       Temp          < 28.9   to the left,  improve=4.2536800, (0 missing)
##       ProcessLength < 184    to the left,  improve=3.4247730, (0 missing)
##       RotationSpeed < 1351   to the right, improve=0.5365079, (0 missing)
##   Surrogate splits:
##       Temp          < 29.3   to the right, agree=0.80, adj=0.512, (0 split)
##       ProcessLength < 166.5  to the right, agree=0.61, adj=0.047, (0 split)
##       RotationSpeed < 1339   to the right, agree=0.60, adj=0.023, (0 split)
##
## Node number 90: 43 observations
##   predicted class=0  expected loss=0.1162791  P(node) =0.006620477
##     class counts:    38     5
##    probabilities: 0.884 0.116
##
## Node number 91: 62 observations,    complexity param=0.02463768
##   predicted class=0  expected loss=0.4677419  P(node) =0.009545804
##     class counts:    33    29
##    probabilities: 0.532 0.468
##   left son=182 (41 obs) right son=183 (21 obs)
##   Primary splits:
##       Temp          < 29.3   to the left,  improve=12.1299700, (0 missing)
##       ProcessLength < 184    to the left,  improve= 4.4555830, (0 missing)
##       RotationSpeed < 1313   to the left,  improve= 1.6614440, (0 missing)
##       ProcessTemp   < 53.6   to the left,  improve= 0.9592794, (0 missing)
##   Surrogate splits:
##       ProcessTemp   < 53.4   to the left,  agree=0.758, adj=0.286, (0 split)
##       ProcessLength < 33     to the right, agree=0.710, adj=0.143, (0 split)
##       RotationSpeed < 1254.5 to the right, agree=0.677, adj=0.048, (0 split)
##
## Node number 182: 41 observations
##   predicted class=0  expected loss=0.2439024  P(node) =0.006312548
##     class counts:    31    10
##    probabilities: 0.756 0.244
##
## Node number 183: 21 observations
##   predicted class=1  expected loss=0.0952381  P(node) =0.003233256
##     class counts:     2    19
##    probabilities: 0.095 0.905
```

Confusion Matrix for Model Tree Version 2

```
predicted_classes_2 <- predict(object = model_tree_2, newdata =
val_set, type = "class")

probis_2 <- predict(object = model_tree_2, newdata = val_set,
type = "prob")

confusionMatrix(predicted_classes_2, val_set$Failed,
positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3364   40
##          1    9   83
##
##                Accuracy : 0.986
##                  95% CI : (0.9815, 0.9896)
##     No Information Rate : 0.9648
##     P-Value [Acc > NIR] : 1.167e-14
##
##                   Kappa : 0.765
##
##  Mcnemar's Test P-Value : 1.822e-05
##
##             Sensitivity : 0.67480
##             Specificity : 0.99733
##          Pos Pred Value : 0.90217
##          Neg Pred Value : 0.98825
##              Prevalence : 0.03518
##          Detection Rate : 0.02374
##    Detection Prevalence : 0.02632
##       Balanced Accuracy : 0.83606
##
##        'Positive' Class : 1
##
```

Comparing to the second confusion matrix for Model Tree Version 2, we can see 3360 jobs correctly predicted as no to fail, and 83 correctly predicted as to fail. An accuracy of 99%, with 9 incorrectly predicted to not-fail, and 40 incorrectly labelled as failed. A precision of 90% and a true positive / recall 67%. The F-Score is ((2 x 0.83 x 0.90) / (0.83 + 0.90)) = 0.86, again closer to 1, than compared to tree 1.

Prediction Model 2 - KNN Classification

In this section, we looked at our second prediction model, classification with KNN. To begin, we updated the dataset by creating a vector and scaling the numerical variables. For the other variables such as Failed, Quality, and Intensity, we converted them into a factor to make them numerical as well. With the pre-processing completed, we were able to move onto partitioning the data.

```
data2[, c(3,4,5,7)] <- scale(data2[, c(3,4,5,7)])
data2$Failed <- as.factor(data2$Failed)
data2$Quality <- as.factor(data2$Quality)
data2$Intensity <- as.factor(data2$Intensity)
```

To control the randomness, we used set.seed(1) to ensure the function would produce the same results. Next, we created a training index based on the target variable “Failed” and chose 65% of the data to be included in the training set. We then used the training index to split the data into two parts, the training and validation set. The training set uses the rows that are included in the training index while the validation set uses all the rows not included in the index.

```
set.seed(1)
train_index_2 <- createDataPartition(data2$Failed, p = 0.65, list = FALSE)

train_set_2 <- data2[train_index,]
val_set_2 <- data2[-train_index, ]
```

The next stage in classification with KNN is training the classification model. Our group used the train function to target Failed and included all the variables to predict whether it Failed. In the function, we used the training set as our data and chose the KNN as our method.

```
KNN_fit <- train(Failed ~ Quality + Temp + ProcessTemp + RotationSpeed + Intensity + ProcessLength, data = train_set_2, meth
od = "knn")

KNN_fit
```

```
## k-Nearest Neighbors
##
## 6495 samples
##    6 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 6495, 6495, 6495, 6495, 6495, 6495, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9780686  0.6282384
##   7  0.9787231  0.6215074
##   9  0.9789409  0.6117417
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

By looking at KNN_fit, we can see that the algorithm decided the k = 7 would be the best number of neighbours to maximize the accuracy.

We then used the predict function to predict the values of the variable Failed in the validation set. These predictions are based on probabilities, and if the probability is greater than 0.5, it will be predicted as a failure. We used KNN_class_prob to display the probabilities of each ID being a failure.

```
KNN_predictions <- predict(object = KNN_fit, newdata =
val_set_2)

KNN_Class_prob <- predict(object = KNN_fit, newdata =
val_set_2, type = "prob")

head(KNN_predictions)
```

```
## [1] 0 0 0 0 0 0
## Levels: 0 1
```

```
head(KNN_Class_prob)
```

```
##           0         1
## 1 1.0000000 0.0000000
## 2 1.0000000 0.0000000
## 3 1.0000000 0.0000000
## 4 0.8888889 0.1111111
## 5 1.0000000 0.0000000
## 6 1.0000000 0.0000000
```

Looking at the head of the predictions and prediction probabilities, we can see the model is predicting a 0 or 1, while the probabilities shows the probability for each case ID and chooses the one higher than 0.5.

Confusion Matrix for KNN Model

Afterwards, we created a confusion matrix to evaluate the classification model. The function was created by taking the predictions on the target variable, using the validation set values of Failed, and specifying Failed as “1”.

```
cm2 <- confusionMatrix(KNN_predictions, val_set_2$Failed,
positive = "1")
cm2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3368   53
##          1    5   70
##
##                Accuracy : 0.9834
##                  95% CI : (0.9786, 0.9874)
##     No Information Rate : 0.9648
##     P-Value [Acc > NIR] : 2.647e-11
##
##                   Kappa : 0.699
##
##  Mcnemar's Test P-Value : 6.769e-10
##
##             Sensitivity : 0.56911
##             Specificity : 0.99852
##          Pos Pred Value : 0.93333
##          Neg Pred Value : 0.98451
##              Prevalence : 0.03518
##          Detection Rate : 0.02002
##    Detection Prevalence : 0.02145
##       Balanced Accuracy : 0.78381
##
##        'Positive' Class : 1
##
```

This confusion matrix and statistics help reveal analysis to better understand the model. To begin, the confusion matrix shows us the true and false positives and negatives within the case to assess its accuracy. Within the validation set, there were 3,368 true negatives. These are instances in which the model predicted the model would not fail, and the result reflected it. The matrix indicates there were 5 false positives, 53 false negatives, and 70 true positives.

Other measures indicated by the confusion matrix statistics include accuracy, sensitivity, and specificity. The accuracy of the model was very impressive at 0.9834. This shows us that over 98% of the time, the model's prediction was successful. One area the model could improve is sensitivity. At roughly 57%, this measure shows the proportion of correct positive predictions to all true positive cases (70 / (70+53)). On the other hand, specificity is very impressive at over 99% as it displays the proportions of correct negative predictions to all true negative cases (3368 / (3368+5)).

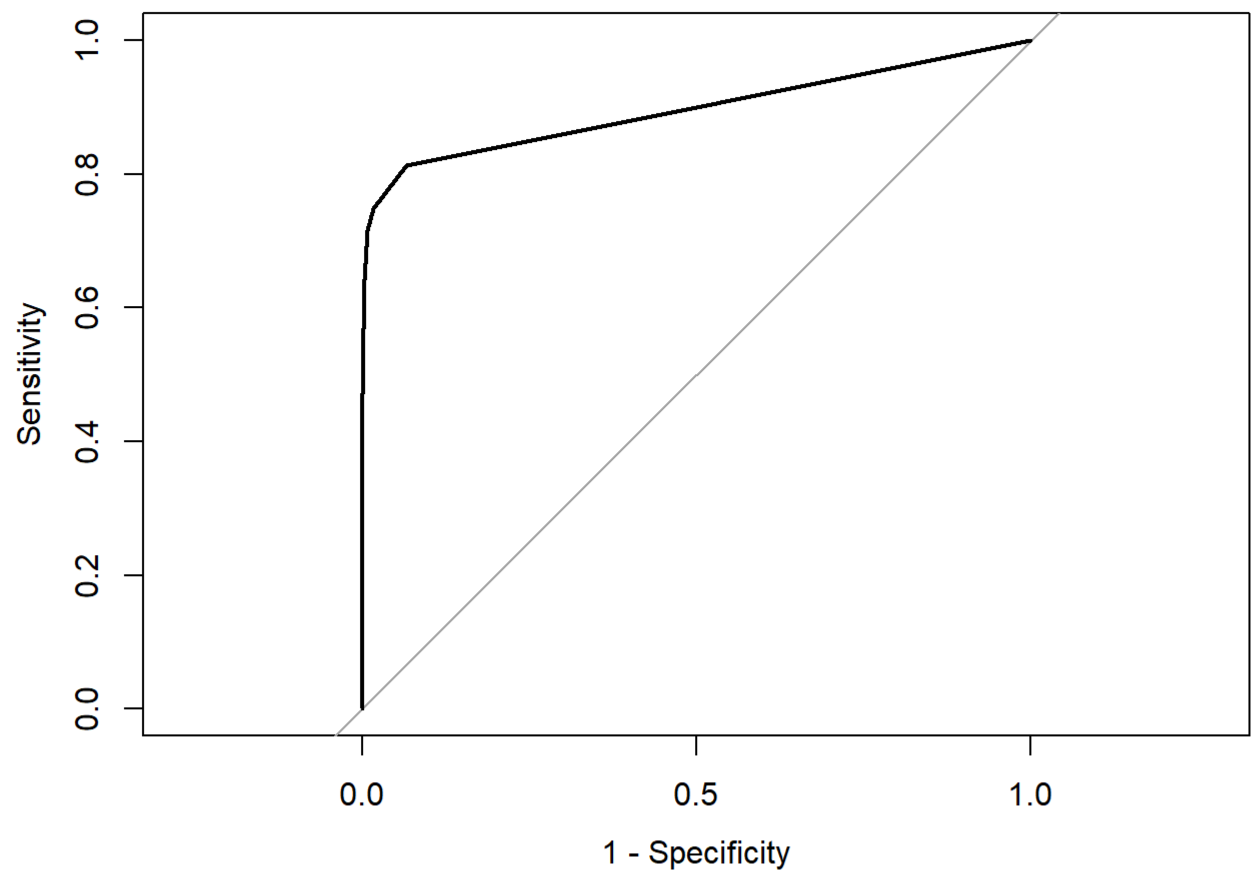
After creating our prediction model, we decided to create the ROC curve and learn about its area under the curve score. With an AUC score of 1, it would indicate a perfect model.We created the ROC curve and plotted it by using the function roc() and adding the actual target values, followed by the probabilities of Failed.

```
roc_object <- roc(val_set_2$Failed, KNN_Class_prob[ ,2])
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot.roc(roc_object, legacy.axes = TRUE)
```



```
auc(roc_object)
```

```
## Area under the curve: 0.8965
```

We plotted the ROC curve and then found its area under the curve score of 0.8965. While there is some room for improvement, this score indicates a very good model.

Finally, we used the coords function to show the cut-off points of the ROC curve along with the sensitivity and (1-specificity) for each threshold.

```
coords <- coords(roc_object, ret = c("threshold",
"sensitivity", "1-specificity"))
```

In the last step of this prediction model, we wanted to learn about the changes that may occur if we chose our value for k. In the model above, k was chosen automatically as 7 as that was seen as the most optimal. However, under KNN_fit_2, we forced the value of k to be 9 and trained the model based on this value. The accuracy under k = 9 was 0.9788, confirming that k = 7 was seen as more optimal.

```
KNN_fit_2 <- train(Failed ~ Quality + Temp + ProcessTemp + RotationSpeed + Intensity + ProcessLength, data = train_set_2, me
thod = "knn", tuneGrid = expand.grid(k = 9))

KNN_fit_2
```

```
## k-Nearest Neighbors
##
## 6495 samples
##    6 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 6495, 6495, 6495, 6495, 6495, 6495, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9788042  0.6106104
##
## Tuning parameter 'k' was held constant at a value of 9
```

Comparison Between Models

In this section, we will compare the performance between the two classification trees and the KNN model. The two models work in varying ways to split the data allowing us to see whether the equipment or machine will fail. In classification trees, the model is split by going though yes and no decisions based on the variables used within the model to predict the target. Meanwhile, the KNN model uses probabilities to split the data and determine whether a Failure will occur.

Looking at the confusion matrix between the classification trees and the KNN model, we can compare measures such as accuracy, sensitivity, and specificity. Beginning with accuracy, both classification trees were slightly more accurate with a total score of 0.986 respectively while the KNN model featured an accuracy score of 0.9834. In terms of sensitivity, the first classification tree's score was most impressive with a total score of

0.73984, compared to the second tree’s 0.67480, and the KNN model’s 0.56911. However, the KNN model featured the greatest specificity between the models with a score of 0.99852. The classification trees also scored highly in this regard with a score of 0.99496 for the first model and 0.99733 for the second model.

We created a decision matrix to help us decide which model to choose based upon those measures. We will award up to 3 stars for each model depending on the result of the measure. With superior accuracy, we awarded 3 stars to both classification trees, and gave 2 stars to KNN. Classification tree 1 showed the strongest sensitivity, allowing us to give the model 3 stars for the measure. This was followed by 2 stars for classification tree 2, and 1 star for the KNN model. Finally, as the KNN model displayed the highest specificity, we awarded it 3 stars, followed by classification tree 2 with 2 stars and rounding out with 1 star for classification tree 1. The Results are shown below:

1. Classification Tree 1: (3 Stars ACC) (3 Stars SENS) (1 Star SPEC) = (7 Stars)
2. Classification Tree 2: (3 Stars ACC) (2 Stars SENS) (2 Stars SPEC) = (7 Stars)
3. KNN Model: (2 Stars ACC) (1 Star SENS) (3 Stars SPEC) = (6 Stars)

Based upon the scores within the confusion matrix of the models and the results in our decision matrix, the classification trees are the recommended models when it comes to predicting whether the machines will fail or not.

The End.