DSE6111 Final Project

Daniel Jackson

December 15th, 2023

# Quantitative Regression, Principal Component Regression and Qualitative Regression

## Introduction

The goal of this research paper is to effectively demonstrate how to use forms of regression to accurately predict real world outcomes from real world data sets. Our main goal is to see how well we can model the data and use that modeling from training data sets within the data to test how accurate our predictions are using test data sets. Rstudio was used to read in, clean up, and manipulate data, and the code that was proudced during my analysis can be found in the the appendix of this research paper.

In the first section of this project, we will be using data from Condition Based Maintenance of Naval Propulsion Plants data set (Coraddu et al., 2014). Quantitative regression approaches and principal component regression approaches were utilized during analysis to create a model that predicts the gas turbine decay coefficient using predictors provided in data set. I will go into further detail of analysis later in this project.

In the second section of this project, we will be using data from Wine Quality data set (Cortez et al., 2009). Qualitative regression approaches were utilized during analysis to create a model that predicts the quality of wine into two categories, good or bad, based on the wine quality score given to each wine by using numerous predictors in data set. I will also go into detail further of this analysis later in this project.

The overall goal of this project is to use Rstudio and the various modeling methods within various packages to read in data, manipulate it, and find models that best fit the data by creating training sets and training models to predict test data. Both the training data and the test data are subsets of the data provided for each data set in this project. We will begin our analysis with the qualitative regression approaches that were used to predict the gas turbine decay coefficient response variable.

# Quantitative Regression

## Data Introduction

As mentioned in the introduction section above, we will be using the data from the Condition Based Maintenance of Naval Propulsion Plants data set (Coraddu et al., 2014). This data was collected in 2014 by a “numerical simulator of a naval vessel (Frigate) characterized by a Gas Turbine (GT) propulsion plant” (Coraddu et al., 2014). The foundational blocks of the simulator in the data have been developed and updated over the years using real propulsion plants. Therefore, we will assume that the data we are working with accurately reflects the results of a real gas turbine propulsion plant. Our goal of this analysis is to use quantitative regression to predict the gas turbine decay coefficient response variable using the predictors given by the data set.

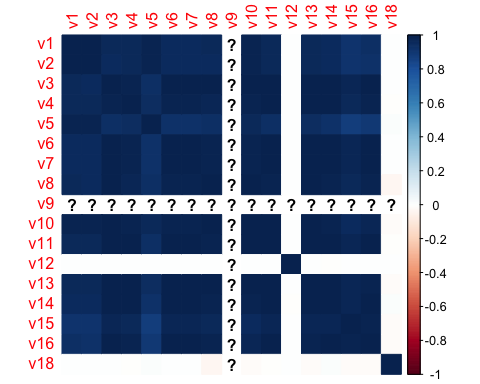
In our analysis, we will explore and fit various models including linear regression, ridge regression, lasso regression, principal component regression, partial least squares regression, regression trees, bagging/random forests, boosting and Bayesian Additive Regression Treees (BART). To start it off, we will being with linear regression after cleaning up and refining our data.

## Refining Data

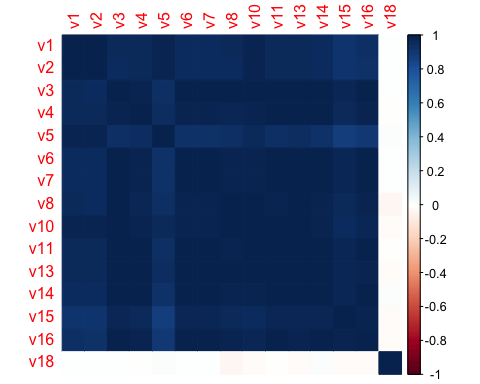
In the Condition Based Maintenance of Naval Propulsion plants (Coraddu et al., 2014) data, (we will use CBM for abbreviation in this analysis), there are 18 total variables provides. Below is a list of the variables and the the abbreviated variable names that we will be using in our analysis:

v1 - Lever position.  
v2 - Ship speed.  
v3 - Gas turbine shaft torque.  
v4 - Gas turbine rate of revolutions.  
v5 - Gas tenerator rate of revolutions.  
v6 - Starboard propeller torque.  
v7 - Port propeller torque.  
v8 - High pressure turbine exit temperature.  
v9 - Gas turbine compressor inlet air temperature.  
v10 - Gas turbine compressor outlet air temperature.  
v11 - High pressure turbine exit pressure.  
v12 - Gas turbine compressor inlet air pressure.  
v13 - Gas turbine compressor outlet air pressure.  
v14 - Gas turbine exhaust gas pressure.  
v15 - Turbine injection control.  
v16 - Fuel flow.  
v17 - Gas turbine compressor decay state coefficient.  
v18 - Gas turbine decay state coefficient.

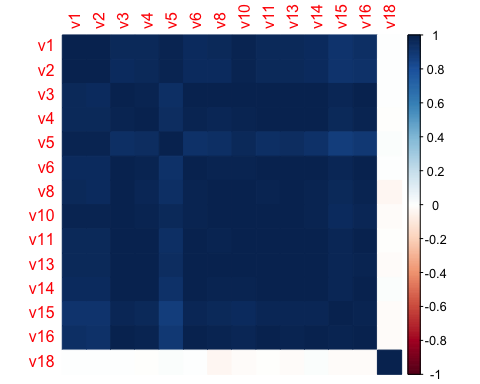
Gas turbine decay state coefficient (v18) is our response variable. Based on the data set’s information file, I know that gas turbine compressor decay state coefficient (v17) is also a response variable. Since we are only focused on the v18 response variable, we will remove v17 from our data set. Let’s start by looking at a correlation matrix of the predictors to see how correlated they are.



We see that the gas turbine compressor inlet air temperature (v9) variable and the gas turbine compressor inlet air pressure (v12) variables are producing either nothing or a question mark within the correlation matrix. We even get an error message in R saying standard deviation is 0. This is probably due to muilticollinearity or perfect collinearity, meaning that the v9 and v12 variables are linearly dependent on each other. Using R to check for which column sums have zero or near-zero variance, both v9 and v12 are returned. Therefore let’s remove them and look at the correlation matrix again.



Digging into the data more, I noticed that the starboard propeller torque (v6) predictor and the port propeller torque (v7) predictor had the same exact values. Using R to check, I was able to confirm that using R. Therefore, I removed the v7 predictor to avoid having the same predictor in our models twice. Let us look at the correlation matrix one more time.



We see that there is heavy correlation between all of the predictors in the new data set using only 13 predictors since we have removed the v7, v9 and v12 predictor. We can now proceed with our analysis since we have cleaned up our data.

## Training and Test Data

There are 11,934 observations in the CBM data set. For the training data and test data that we will be using to predict the v18 response variable, we will split the data in half. One half of the data will be used for training our models and the other data will be used for testing our trained model.

11934 / 2

## [1] 5967

set.seed(1)  
train = sample(1:nrow(cbm\_df), 5967)  
cbm\_train = cbm\_df[train,]  
cbm\_test = cbm\_df[-train,]

Since there is an even number of observations in the data set, we can randomly split the data in half and use half of the data for training models, and the other half for testing. There will be 5967 observations in both the training and testing data sets.

## Multiple Linear Regression

In this section, we will try and model the data using multiple linear regression using the various predictors in our data set. We will use the training data to fit our models and try and predict the test data v18 response variables. Our goal is to find which model produces the lowest test mean squared error. The one that produces the smallest test mean squared error will be the data that we choose.

### Best Subset Selection, Forward Selection, Backward Selection

Now we want to look at using best subset selecition, forward selection and backward selection to determine how many variables we want to use in our model. We will first start by using best subset selection on the data with v18 as our response variable. We want to see how many variables we can use to model the data that will maximize the adjusted R^2 value and minimizes the Bayestion Information Criterion (BIC) and Mallow’s Cp (Cp) values.

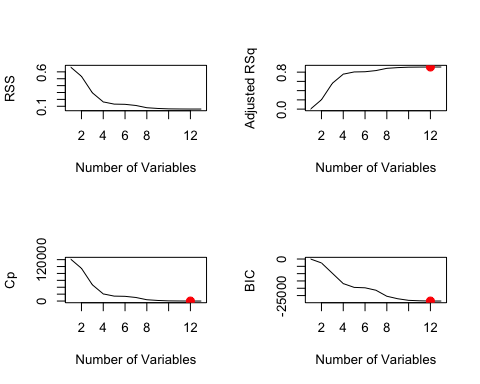
## [1] "which" "rsq" "rss" "adjr2" "cp" "bic" "outmat" "obj"

## [1] 0.001479472 0.202891738 0.559237682 0.756206430 0.804173571 0.808886591  
## [7] 0.833977086 0.882402098 0.898193760 0.907201814 0.909639453 0.910994155  
## [13] 0.910998476

## [1] 12

## [1] 12

## [1] 12



Using best subset selection, we see that 12 out of the 13 predictors will maximize our adjusted R^2 value and minimize the BIC and Cp values. Using both the forward selection and backward selection methods in R, both models that incorporate 12 total predictors do not include the lever position (v1) predictor.

Since v1 was not included in both the forward and backward selection models, I will keep that variable in mind as the first one removed in the models moving forward. My approach for the multiple linear regression will combine both forward and backward selection by using training data on test data while examining p-values and absolute values of t to add and remove variables from models. My goal is to try and make the model as simple as possible by maximizing the R^2 value and minimizing the test mean squared error.

## Multiple Linear Regression continued

Before I ran fitted a linear multiple regression model with all of the predictors on the training data, I wanted to see what the model looked like on the entire data set as a whole taking in all 5967 observations.

cbm\_lm = lm(v18 ~., cbm\_df)  
summary(cbm\_lm)

##   
## Call:  
## lm(formula = v18 ~ ., data = cbm\_df)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.008886 -0.001277 -0.000050 0.001271 0.009509   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.430e-01 7.196e-02 1.987 0.047 \*   
## v1 1.220e-03 1.604e-03 0.761 0.447   
## v2 5.800e-03 5.011e-04 11.574 <2e-16 \*\*\*  
## v3 1.381e-05 1.277e-07 108.166 <2e-16 \*\*\*  
## v4 6.543e-05 1.058e-06 61.869 <2e-16 \*\*\*  
## v5 3.148e-05 3.105e-07 101.400 <2e-16 \*\*\*  
## v6 -1.583e-03 1.123e-05 -140.896 <2e-16 \*\*\*  
## v8 -6.809e-04 7.014e-06 -97.080 <2e-16 \*\*\*  
## v10 1.828e-04 1.109e-05 16.486 <2e-16 \*\*\*  
## v11 8.168e-02 3.364e-03 24.281 <2e-16 \*\*\*  
## v13 -5.377e-02 2.991e-04 -179.730 <2e-16 \*\*\*  
## v14 9.313e-01 7.063e-02 13.186 <2e-16 \*\*\*  
## v15 -2.387e-04 6.374e-06 -37.456 <2e-16 \*\*\*  
## v16 3.589e-01 4.286e-03 83.724 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.002239 on 11920 degrees of freedom  
## Multiple R-squared: 0.911, Adjusted R-squared: 0.9109   
## F-statistic: 9385 on 13 and 11920 DF, p-value: < 2.2e-16

If we were to state that the null hypothesis is that none of the predictors in the linear model were statistically significant when it came to predicting v18, we could then reject the null hypothesis because the p-value for each predictor in this linear model is less than 0.05. Therefore, we know that each predictor plays a statistically significant part when it comes to predicting our response variable v18. The only variable that stands out in this model is v1 as it barely passes the p-value test with a p-value of 0.047. From the the knowledge that I gained from the best subset models and the forward/backward selection models I fit earlier, this result matches the fact that those models stated the that the 12 best predictor model would not include v1. Therefore, I decided to fit a linear model removing v1 as a predictor.

cbm\_lm = lm(v18 ~. -v1, cbm\_df)  
summary(cbm\_lm)

##   
## Call:  
## lm(formula = v18 ~ . - v1, data = cbm\_df)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.0089066 -0.0012723 -0.0000474 0.0012679 0.0095210   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.562e-01 6.984e-02 2.236 0.0254 \*   
## v2 6.178e-03 5.650e-05 109.352 <2e-16 \*\*\*  
## v3 1.384e-05 1.224e-07 113.083 <2e-16 \*\*\*  
## v4 6.503e-05 9.176e-07 70.873 <2e-16 \*\*\*  
## v5 3.142e-05 2.983e-07 105.311 <2e-16 \*\*\*  
## v6 -1.583e-03 1.123e-05 -140.954 <2e-16 \*\*\*  
## v8 -6.821e-04 6.839e-06 -99.744 <2e-16 \*\*\*  
## v10 1.878e-04 8.912e-06 21.076 <2e-16 \*\*\*  
## v11 8.179e-02 3.360e-03 24.339 <2e-16 \*\*\*  
## v13 -5.371e-02 2.896e-04 -185.427 <2e-16 \*\*\*  
## v14 9.170e-01 6.808e-02 13.470 <2e-16 \*\*\*  
## v15 -2.376e-04 6.213e-06 -38.248 <2e-16 \*\*\*  
## v16 3.581e-01 4.162e-03 86.046 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.002239 on 11921 degrees of freedom  
## Multiple R-squared: 0.911, Adjusted R-squared: 0.9109   
## F-statistic: 1.017e+04 on 12 and 11921 DF, p-value: < 2.2e-16

All of the predictors p-values in this model which did the v1 predictor were less than 0.05, allowing us to reject the null hypothesis. This model also provided the same R^2 value, 91.1%, as the first model that included all of the predictors. Therefore, we did not lower the amount of variance explained by removing the v1 predictor.

I did try some further removal of the v1 predictor and predictors with negative coefficients in the linear model: starboard propellor torque (v6), high point turbine exit temperature (v8), gas turbine compressor outlet air pressure (v13), and turbine injection control (v15). This unfortunately caused many significant predictor’s p-values in the model to be greater than 0.05, therefore we could not reject the null hypothesis. And the R^2 value decreased all the way to 24.33%.

cbm\_lm = lm(v18 ~. - v1 - v6 - v8 - v13 - v15, cbm\_df)  
summary(cbm\_lm)

##   
## Call:  
## lm(formula = v18 ~ . - v1 - v6 - v8 - v13 - v15, data = cbm\_df)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.0183873 -0.0048101 0.0000355 0.0048412 0.0176498   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -2.030e+00 1.950e-01 -10.408 < 2e-16 \*\*\*  
## v2 -2.130e-03 1.071e-04 -19.893 < 2e-16 \*\*\*  
## v3 7.992e-06 1.691e-07 47.268 < 2e-16 \*\*\*  
## v4 5.001e-05 2.510e-06 19.926 < 2e-16 \*\*\*  
## v5 6.187e-08 5.907e-07 0.105 0.91659   
## v10 6.568e-05 2.068e-05 3.177 0.00149 \*\*   
## v11 -1.659e-01 6.119e-03 -27.104 < 2e-16 \*\*\*  
## v14 3.056e+00 1.888e-01 16.188 < 2e-16 \*\*\*  
## v16 -1.114e-01 5.143e-03 -21.660 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.006526 on 11925 degrees of freedom  
## Multiple R-squared: 0.2433, Adjusted R-squared: 0.2428   
## F-statistic: 479.4 on 8 and 11925 DF, p-value: < 2.2e-16

Now it is time to let the training data train the linear model that does not include the v1 predictor. We will then take the training model predictions and test them on the actual values of the v18 response variable in our test data set. The code I used is below and this will be the R code format that I used to find the test mean squared error for each of the linear regression models that I fitted in this section.

train\_cbm\_lm = lm(v18 ~. - v1, cbm\_train)  
summary(train\_cbm\_lm)

##   
## Call:  
## lm(formula = v18 ~ . - v1, data = cbm\_train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.0083554 -0.0012567 -0.0000527 0.0012653 0.0094434   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 7.834e-02 9.737e-02 0.805 0.421   
## v2 6.148e-03 7.942e-05 77.418 <2e-16 \*\*\*  
## v3 1.381e-05 1.721e-07 80.224 <2e-16 \*\*\*  
## v4 6.517e-05 1.266e-06 51.486 <2e-16 \*\*\*  
## v5 3.111e-05 4.178e-07 74.453 <2e-16 \*\*\*  
## v6 -1.573e-03 1.580e-05 -99.575 <2e-16 \*\*\*  
## v8 -6.771e-04 9.564e-06 -70.801 <2e-16 \*\*\*  
## v10 1.880e-04 1.232e-05 15.261 <2e-16 \*\*\*  
## v11 7.951e-02 4.680e-03 16.987 <2e-16 \*\*\*  
## v13 -5.347e-02 4.059e-04 -131.748 <2e-16 \*\*\*  
## v14 9.942e-01 9.489e-02 10.477 <2e-16 \*\*\*  
## v15 -2.335e-04 8.744e-06 -26.707 <2e-16 \*\*\*  
## v16 3.554e-01 5.831e-03 60.949 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.00221 on 5954 degrees of freedom  
## Multiple R-squared: 0.9125, Adjusted R-squared: 0.9123   
## F-statistic: 5174 on 12 and 5954 DF, p-value: < 2.2e-16

pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)

## [1] 5.146441e-06

All of the p-values for each predictor is less than 0 allowing us to reject the null hypothesis for the model fitted on the training data. The test mean squared error is equal to 0.00000515 which is very close to 0. This means that this model does a fantastic job of predicting the actual response values of v18 in our test data set.

Since all of the p-values are less than 0.05 in this model, I wanted to try and reduce the amount of predictors to simplify the model by removing predictors with the lowest absolute value of t without significantly impacting the R^2 value or the test mean squared error. After many trials of this to try and simplify the model with less predictors, I landed on a model that removed the removed the v1 predictor, the gas turbine compressor outlet air temperature (v10) predictor, and the gas turbine exhaust gas pressure (v14) predictor. This model has two less predictors than the previous model that just removed v1. The R^2 value is 90.81% and the test mean squared error is 0.00000536. Both R^2 and test mean squared error similar that previous model taht just removed the v1 predictor.

train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10, cbm\_train)  
summary(train\_cbm\_lm)

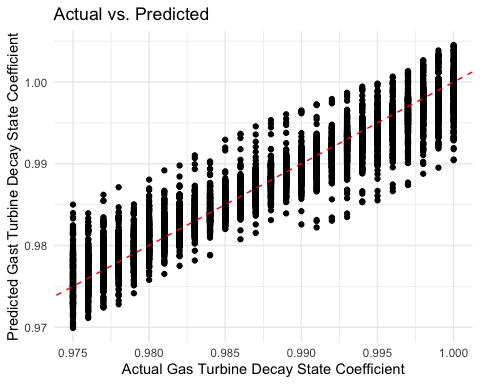
##   
## Call:  
## lm(formula = v18 ~ . - v1 - v14 - v10, data = cbm\_train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.0095219 -0.0012895 0.0000077 0.0012729 0.0096878   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.148e+00 3.327e-03 344.98 <2e-16 \*\*\*  
## v2 6.414e-03 7.772e-05 82.53 <2e-16 \*\*\*  
## v3 1.313e-05 1.666e-07 78.82 <2e-16 \*\*\*  
## v4 7.014e-05 1.206e-06 58.14 <2e-16 \*\*\*  
## v5 3.080e-05 4.075e-07 75.60 <2e-16 \*\*\*  
## v6 -1.514e-03 1.546e-05 -97.93 <2e-16 \*\*\*  
## v8 -6.272e-04 8.769e-06 -71.53 <2e-16 \*\*\*  
## v11 1.054e-01 4.397e-03 23.98 <2e-16 \*\*\*  
## v13 -5.519e-02 3.990e-04 -138.32 <2e-16 \*\*\*  
## v15 -2.271e-04 8.858e-06 -25.64 <2e-16 \*\*\*  
## v16 3.429e-01 5.886e-03 58.25 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.002264 on 5956 degrees of freedom  
## Multiple R-squared: 0.9081, Adjusted R-squared: 0.908   
## F-statistic: 5887 on 10 and 5956 DF, p-value: < 2.2e-16

# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)

## [1] 5.363148e-06

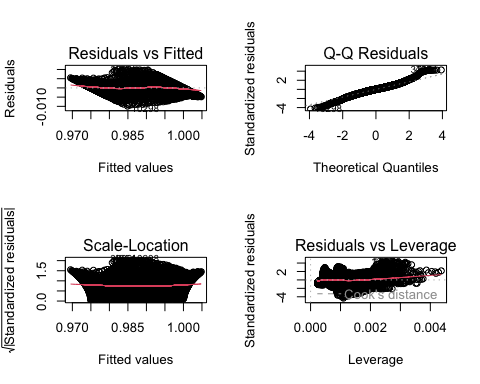
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 90.81%%

Let us look at a scatter plot comparing the actual values of the response variable v18 in the test data versus the prediction values of v18 from our model fit on the training data.



We see from this plot of the actual versus prediction values of our v18 response variable, our predictions follow the line of perfect predictions very well, which further suggests that our model that we fit is a good model to use when making predictions.

Let us now plot this model on the entire data set:



Looking at the random scattering of the residuals around 0 in the residuals vs fitted plot, we do not see any major pattern, which means our model is a good fit. Looking at the Q-Q residuals plot we see that a majority of the data fitted by the model follows normal distribution but with a curve off on the extremities meaning there may be some extreme values in this plot. There also seems to be a few leverage points in the data but not too many.

After fitting this model, I then tried some transformations and interactions on the predictors to see if I could lower my test mean squared anymore and also raise my R^2 value. I tried a square root transformation on all of the predictors, a squared transformation on all of the predictors, a cubed transformation on all of the predictors and a log transformation on all of the predictors. Of the transformations that I did, the square root transformation produced a test mean square error of 0.00000389 and a R^2 value of 93.21%. This test mean squared error is smaller than the model with no transformations and a the R^2 on the training data is also higher in this mode by about 3%.

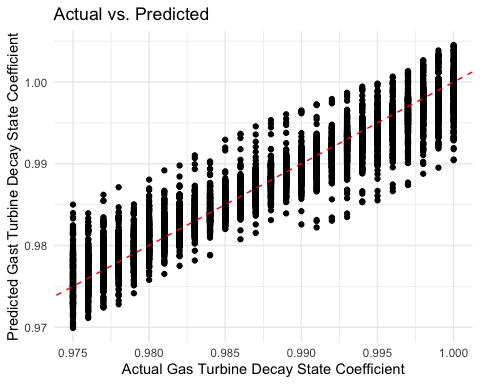
I created a model off of the square root transformation model. It is model where the highest correlated predictors were transformed and then interacted with each other. Here are the list of predictors that I had interact with each other in this model:

sqrt(v2) and sqrt(v5). sqrt(v4) and sqrt(v11) sqrt(v3) and sqrt(v6) sqrt(v8) and sart(v13) and v16 on its own.

This model:

lm(v18 ~ sqrt(v2)\*sqrt(v5) + sqrt(v4)\*sqrt(v11) + sqrt(v3)\*sqrt(v6) +  
 sqrt(v8)\*sqrt(v13) + sqrt(v16), cbm\_train)

provided a R^2 value on the training data of 94.82% and the mean squared test error rate of 0.00000295 was lowest rate of all the models we fit. Let us plot the predicted values of v18 from this transformed model versus the actual values of v18.



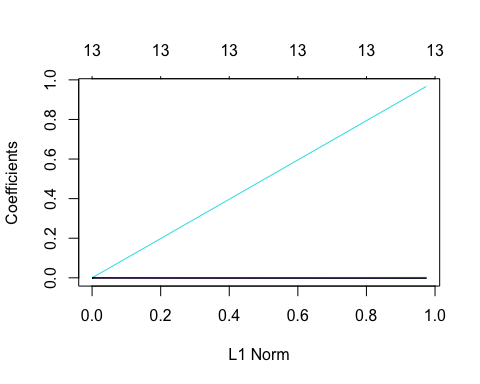
As you can see, this model does a great job of predicting the actual values of v18 from the square root transformed model that does not factor in the v1, v10 or v14 predictor. Just to summarize what predictors we are using in this square root transformed linear model:

v2 - Ship speed. v3 - Gas turbine shaft torque.  
v4 - Gas turbine rate of revolutions.  
v5 - Gas tenerator rate of revolutions.  
v6 - Starboard propeller torque.  
v8 - High pressure turbine exit temperature. V11 - High pressure turbine exit pressure.  
v13 - Gas turbine compressor outlet air pressure.  
v15 - Turbine injecton control.  
v16 - Fuel flow.

Now, let us look at a ridge regression model to see what sort of results we get there.

## Ridge Regression

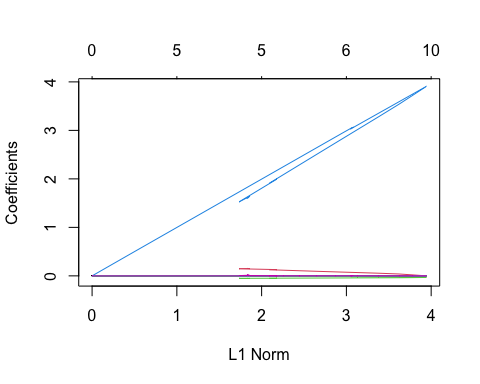
Using ridge regression, our goal here is to see if we can produce a test mean squared error less than the the mean squared error that we found in the square root transformed model from the linear regression section above. What we will first is create a training matrix and a test matrix for our ridge regression. Using cross-validation, I found that the best shrinking parameter, lambda, to use for this model is when lambda = 0.00004. Using lambda = 0.00004 in my ridge regression model using the training matrix to predict the response v18 values in the test data, I found a test mean squared error of 0.000048. This test mean squared error, although small, is not as small as the test error from the model we created in the linear regression section above.



The blue line shows how the coefficients in the ridge regression model change as the the lambda value varies. The thin black line represents the lambda value of 0.00004 that we used as the shrinking paramater. Now, let us fit a lasso regression model.

## Lasso Regression

Now we will fit a lasso regression model to the data. We will use the same training and test matrices created in the ridge regression model from the previous section to fit this lasso regression model. The best shrinking parameter, lambda, to use in the lasso regression model is when lambda = 0.00000549. Using that lambda in the regression model, we get a test mean squared error value of 0.0000095. This test mean squared error is not the smallest test mean squared error that we have seen so far.

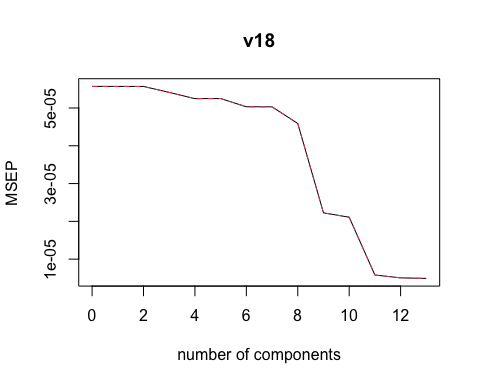


Just like in the ridge regression model, the blue line represents how the coefficients in the lasso model are affected by the shrinking parameter, lambda. The red line indicates the coefficient paths of the predictors as lambda changes and the green line represents the value of lambda that we got from cross-validation, which equaled 0.00000549. Now, let us look at principal components regression.

## Principal Components Regression

The main goal of principal components regression is to reduce the dimensions of the model by removing predictors to simplify the model. Let us fit a principal components regression model using the training data to then predict the response variable v18 in the test data.

## Data: X dimension: 5967 13   
## Y dimension: 5967 1  
## Fit method: svdpc  
## Number of components considered: 13  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 0.007464 0.007464 0.007463 0.007357 0.007244 0.007245 0.007094  
## adjCV 0.007464 0.007464 0.007463 0.007357 0.007243 0.007245 0.007093  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 0.007093 0.006775 0.004717 0.004593 0.002409 0.002246 0.002215  
## adjCV 0.007092 0.006774 0.004717 0.004592 0.002409 0.002246 0.002215  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps  
## X 97.4381 99.3413 99.605 99.770 99.907 99.973 99.990 100.0  
## v18 0.0466 0.0896 2.937 5.947 5.952 9.887 9.948 17.9  
## 9 comps 10 comps 11 comps 12 comps 13 comps  
## X 100.00 100.00 100.00 100.00 100.00  
## v18 60.21 62.33 89.64 90.99 91.25



We see that where M = 10, 11 and 12 produces the lowest mean squared error prediction value. However, we are trying our best to reduce the dimensions of the model.  
 When we fit a principal component regression using M = 11, we get a mean squared test error of 0.00000605.

When we fit a principal component regression using M = 10, we get a mean squared test error of 0.0000223.

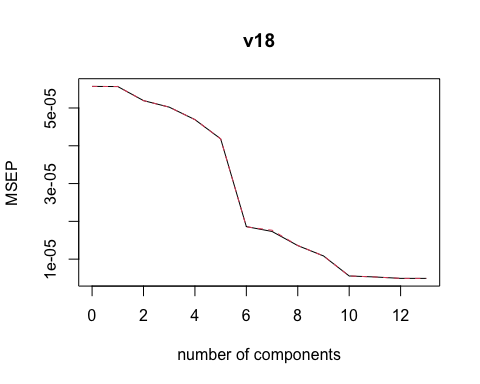
When we fit a principal component regression using M = 9, we get a mean squared test error of 0.0000223. Same mean squared test error when M = 10.

Any other lower M value will drastically lower the mean squared test error value. Although the goal of principal component regression model is to reduce the dimensions, the model where M = 11 results in the lowest mean squared test error value without using all of the 12 predictors in the model. Now let us fit a partial least squares model.

## Partial Least Squares

We will now try and reduce the dimensions by fitting a partial least squares model. Our goal, once again, is to create a more simple model.

## Data: X dimension: 5967 13   
## Y dimension: 5967 1  
## Fit method: kernelpls  
## Number of components considered: 13  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 0.007464 0.007461 0.007208 0.007087 0.006850 0.006469 0.004314  
## adjCV 0.007464 0.007461 0.007208 0.007086 0.006849 0.006468 0.004307  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 0.004162 0.003686 0.003294 0.002355 0.002299 0.002216 0.002215  
## adjCV 0.004202 0.003691 0.003294 0.002355 0.002298 0.002216 0.002215  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps  
## X 97.33913 98.226 99.56 99.71 99.82 99.84 99.96 99.99  
## v18 0.08965 6.952 10.05 16.04 25.13 66.79 68.57 75.65  
## 9 comps 10 comps 11 comps 12 comps 13 comps  
## X 100.00 100.0 100.00 100.00 100.00  
## v18 80.61 90.1 90.57 91.24 91.25



Once again, we see that the lowest mean squared error prediction occurs when M = 12, but we want to reduce our dimensions in this model. Looking at the graph let us try when M = 10.

With M = 10 provides us with a mean squared test error rate of 0.00000578.

With M = 9, we get a mean squared test error rate of 0.0000113.

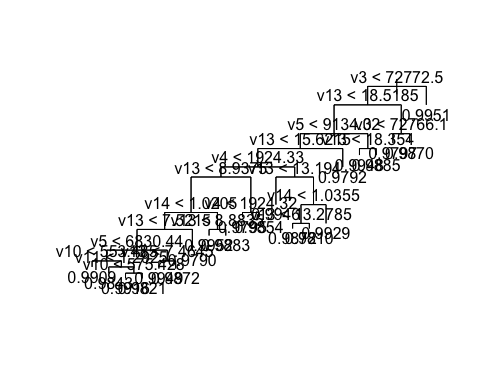
With M = 8, we get a mean squared test error rate of 0.0000144.

As we can see, the lowest mean squared error rate occurs when M = 10 in our partial least squares model. This is one lest predictor than what we saw in the principal component regression model we fitted in the previous model, which is a step in the right direct with what we are trying to achieve with these dimension reducing regression methods. Now, let us look at regression trees.

## Regression Trees

We will now fit a regression tree using our training data set and analyze how many terminal nodes we will have.

##   
## Regression tree:  
## tree(formula = v18 ~ ., data = cbm\_train)  
## Variables actually used in tree construction:  
## [1] "v3" "v13" "v5" "v4" "v14" "v10" "v11"  
## Number of terminal nodes: 21   
## Residual mean deviance: 1.758e-05 = 0.1045 / 5946   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -1.656e-02 -2.840e-03 -6.006e-06 0.000e+00 2.916e-03 1.575e-02

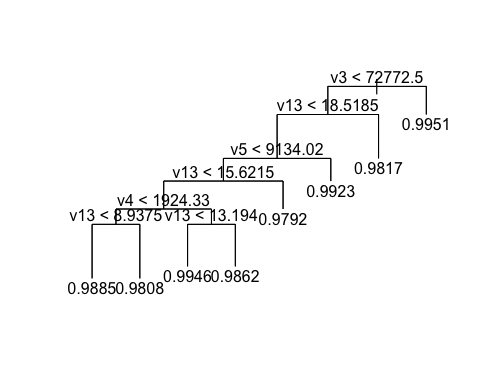


We see that this tree is quite messy and very hard to interpret. This is due to the fact that there are 21 terminal nodes in this regression tree. We do see that the only predictors used in the regression tree are v3, v13, v5, v4, v14, v10 and v11. The test mean squared error for this regression tree is 0.0000191.

Since this tree is quite confusing to understand, we want to try and prune the tree using cross-validation. We see that the variance is lowest when we have 21 terminal nodes. Using this graph above, let us try a tree with 15, 11, 9 and 8 terminal nodes without sacrificing our variance to see which one produces a low test mean squared error rate while creating a more interpretable tree.

The mean squared test error rate with 15 terminal nodes is 0.0000231.  
The mean squared test error rate with 11 terminal nodes is 0.0000284. The mean squared test error rate with 9 terminal nodes is 0.0000206. The mean squared test error rate with 8 terminal nodes is 0.0000363.

The pruned tree with 8 terminal nodes has slightly higer test mean squared error than regression tree with 21 terminal nodes, but this tree is much simplier to understand. Now, let us fit a bagging and random forest model.



## Bagging and Random Forests

Now we will fit a bagging model and multiple random forest models and find the test mean squared error rate to see how they compare to previous models. Let us first fit a bagging model using 500 trees that utilizes all 13 predictors at each split.

set.seed(1)  
cbm\_bag = randomForest(v18 ~., data = cbm\_train, mtry = 13, importance = TRUE)  
cbm\_bag

##   
## Call:  
## randomForest(formula = v18 ~ ., data = cbm\_train, mtry = 13, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 13  
##   
## Mean of squared residuals: 8.619948e-07  
## % Var explained: 98.45

yhat\_bag = predict(cbm\_bag, newdata = cbm\_test)  
mean((yhat\_bag - cbm\_test$v18)^2)

## [1] 8.5755e-07

In a bagging model using all 13 predictors, 98.45% of the variance is explained in the training data set and the test mean squared error rate is 0.000000858. This is the lowest test mean squared error of any model that we have fit to this point. Now, let us look at some random forests where we try different number of trees and variables used at each split to try and simplify our bagging model. We will first try 25 trees and use all 13 predictors at each split.

set.seed(1)  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 25,  
 mtry = 13, importance = TRUE)  
cbm\_rf

##   
## Call:  
## randomForest(formula = v18 ~ ., data = cbm\_train, ntree = 25, mtry = 13, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 25  
## No. of variables tried at each split: 13  
##   
## Mean of squared residuals: 1.135508e-06  
## % Var explained: 97.96

yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)

## [1] 9.690138e-07

In this random forest model, 96.95% of the variance is explained and the mean squared test error rate is 0.000000969.

Now, let us try with 10 predictors and 25 trees.

set.seed(1)  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 25,  
 mtry = 10, importance = TRUE)  
cbm\_rf

##   
## Call:  
## randomForest(formula = v18 ~ ., data = cbm\_train, ntree = 25, mtry = 10, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 25  
## No. of variables tried at each split: 10  
##   
## Mean of squared residuals: 1.189997e-06  
## % Var explained: 97.86

yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)

## [1] 1.025548e-06

97.86% of variance explained in this model and the mean squared test error rate is 0.00000102.  
Now, let us try with 10 variables and 10 trees.

set.seed(1)  
cbm\_bag = randomForest(v18 ~., data = cbm\_train, ntree = 10,  
 mtry = 10, importance = TRUE)  
cbm\_rf

##   
## Call:  
## randomForest(formula = v18 ~ ., data = cbm\_train, ntree = 25, mtry = 10, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 25  
## No. of variables tried at each split: 10  
##   
## Mean of squared residuals: 1.189997e-06  
## % Var explained: 97.86

yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)

## [1] 1.025548e-06

97.86% of variance explained in this model and a mean squared test error rate of 0.00000102.

As you reduce the number of variables and the trees to less than 10 variables and less than 25 trees, you are not making significant impact on the mean squared test error. The best model so far from the random forest models that we have fit is when there are 13 variables and 25 trees. Although the mean squared test error rate was smallest when we used all 13 variables and 500 trees, the processing time on that model was just not as quick as the one with 25 trees. And there was not a significant delta in the mean squared test error rate to justify a model with that slow of a repsonse time to fit. Now, we will fit a boosted model.

## Boosting

In this section, we will fit a boosted model using the training data and will make predicitions on that model using the test data to compare. We will try a few different shrinking paramater lambda values to compare mean squared test error rates of models. Our boosted models that we will fit will use lambda values of 0.001, 0.01. 0.1, and 0.2 and 1000 trees.

# lambda = 0.001:  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)

## [1] 1.789212e-05

# lamda = 0.01  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.01)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)

## [1] 4.480735e-05

# lamda = 0.1  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.1)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)

## [1] 1.789212e-05

# lamda = 0.2  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.2)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)

## [1] 1.002661e-05

The mean squared test error rate with shrinking parameter of 0.001 is 0.0000179.  
The mean squared test error rate with shrinking parameter of 0.01 is 0.0000448.  
The mean squared test error rate with shrinking parameter of 0.1 is 0.0000179.  
The mean squared test error rate with shrinking parameter of 0.2 is 0.00001.

The smallest mean squared test error rate occurred when we had a shrinking parameter lamda value of 0.2. Now, we will fit our final regression model on this data: the Bayesian Additive Regression Trees (BART).

## Bayesian Additive Regression Trees (BART)

We will now fit a BART model to this data. We will need to re-establish our training and test data for this model first before we can create the model. Then we will find the mean squared test error rate using the new training data and test data to compare predictions.

yhat\_bart = cbm\_bart$yhat.test.mean   
mean((ytest1 - yhat\_bart)^2)

## [1] 4.139308e-06

The mean squared test error rate of the BART model is 0.00000414.

## Test Mean Squared Error Summary

Here are the mean squared test error rates of each model that we fit.

Linear regression model: 0.00000295 with R^2 value of 94.82%.

Ridge Regression Model: 0.000048.

Lasso Regression Model: 0.0000095.

Principal Component Regression Model: 0.00000605.

Partial Least Squares Model: 0.00000578.

Regression Tree Model: 0.0000363 with tree that has 8 terminal nodes (this was the most interpretable regression tree).

Bagging Model: 0.000000858 using 500 trees and trying all 13 predictors at each split, 98.45% of variance explained in training data.

Random Forest Model: 0.000000969. 25 trees and 13 predictors at each split. 97.96% of variance explained in training data.

Boosting Model: 0.00001 using lambda shrinking parameter of 0.2.

BART Model: 0.00000414.

## Conclusion

As we can see from the mean squared test error values in the section above, all of the models seemed to do a great job at predicting our response variable, the gas turbine decay state coefficient, as all of the test mean squared error values are very close to 0. I ultimately believe that you can choose any one of these models to confidently predict that response variable using the predictors given in each model. I believe the best way to choose which model best predicts our response variable does not rely strictly on the mean squared test error, but on which model is easiest to interpret and which one takes the least amount of processing time to fit training data and predict test data.

For instance, of all of the quantitative regression models that we fit, the bagging model that used 500 trees and all 13 predictors at each split produced the lowest test error rate of 0.000000858. However, the processing time that the code took in order to produce the model took far too long compared to the random forest model that used 25 trees and 13 predictors at each split. This random forest model produced a mean squared test error rate of 0.000000969 and the model on the training data explained 97.96%.

Since all of the models produced a mean squared test error close to 0, I am going to choose the linear regression model where we squared the the predictors and interacted multiple predictors. The model is below:

lm(v18 ~ sqrt(v2)\*sqrt(v5) + sqrt(v4)\*sqrt(v11) + sqrt(v3)\*sqrt(v6) +  
 sqrt(v8)\*sqrt(v13) + sqrt(v16), cbm\_train)

I do not believe that there is a big enough difference in the various mean squared test error rates that we obtained to forgo the linear model that we created in the multiple linear regression section of this research project. Therefore, I believe the linear model that I created using the square root transformation and certain interactions between those square rooted variables is the easiest model to grasp for those who are familiar with linear models and those who are not. Being able to provide a model that allows the user to plug in values for each predictor to help them predict the gas turbine decay state coefficient response value, allows a user-friendly, and easy-to-understand experience. Plus, a linear model is the most basic form of linear regression to work with which furthermore enhances the convenience of conveying the model to one another to enhance their understanding of it.

# Qualitative Regression

## Data Introduction

In our qualitative regression analysis, we will be using the Wine Quality data set which models wine quality based on physicochemical tests using a data set of red and white vinho verde samples gathered in northern Portugal in 2009 (Cortez et al., 2009).

This wine data includes two separate sets of data. The first data set contains 1599 observations of red wine tests and 4898 observations of white wine tests. In order to maximize my sample size, I combined the two data sets together to create one data set that included 6497 observations. In the combined data set, the wine quality scoring ranged from 3 to 9. In order to make predictions on this data using qualitative regression, I manipulated the data so that if the wine quality score was between 3 and 5, we labeled the wine quality score as “poor” quality and if the wine quality score was between 6 and 9, we labeled the wine quality score as “good” quality. Our goal is to use qualitative regression to fit a model that accurately classifies the wine quality score as either “poor” or “good” based on the predictors given to us in the data set.

In this analysis, we will use and fit various qualitative regression models such as logistic regression, linear discriminant analysis (LDA), quadratic descriptive analysis (QDA), K-nearest neighbors (KNN) regression, naive Bayes regression, poisson regression, and classification trees (such as pruning trees, bagging, random forests, boosting and Bayesian Additive Trees (Bart)). We wil start with logistic regression once we clean up and refined data.

## Refining Data

In the Wine Quality data set, when we combined the red and the white wine data sets, we created one data set that included 6497 observations. This means that we have 6497 different wines from northern Portugal to analyze to help us fit our model. In this model, there are 11 total predictors that we can use in our analysis to help us predict our wine quality response variable. Below is a list of those predictors:

Fixed acidity.  
Volatile acidity.  
Citric acid.  
Residual sugar.  
Chlorides.  
Free sulfur dioxide.  
Total sulfur dioxide.  
Density.  
pH.  
Sulphates.  
Alcohol.

In order to utilize a qualitative regression approach, we will manipulate the quality response variable into a bernoulli variable that will take on the value of 0 if the wine quality is poor (wine quality score from 3-6) or the value of 1 if the wine quality is good (wine quality score from 7-9).

## Training and Test Data

Just like in our qualitative regression analysis from earlier, we want to create a training and test data set using the 6497 observations that we have. Just like before, we will split this in half. Since the number of observations is an odd number, one of our data sets will have one more observation than the other. For no particular reason, I had the test data set have the one more variable than the training data. Therefore the training data set has 3248 observations and the test data set has 3249 observations. Now, we will start with our logistic regression.

set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
wine\_train = wine\_df[train,]  
wine\_test = wine\_df[-train,]

## Logistic Regression

Looking at our logistic regression model using all 11 of the predictors in our wine quality data set, and using a null hypothesis that none of the predictors are statistically significant in predicting our quality response variable, we can reject the null hypothesis on a total of 8 predictors as their p-values are less than 0.05.

glm\_wine = glm(quality~., wine\_df, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ ., family = binomial, data = wine\_df)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 4.849e+01 3.981e+01 1.218 0.2232   
## fixed\_acidity 8.228e-02 5.096e-02 1.615 0.1064   
## volatile\_acidity -4.424e+00 2.750e-01 -16.086 < 2e-16 \*\*\*  
## citric\_acid -5.889e-01 2.536e-01 -2.322 0.0203 \*   
## residual\_sugar 8.814e-02 1.693e-02 5.205 1.94e-07 \*\*\*  
## chlorides -9.203e-01 1.033e+00 -0.891 0.3732   
## free\_sulfur\_dioxide 1.663e-02 2.514e-03 6.614 3.75e-11 \*\*\*  
## total\_sulfur\_dioxide -7.669e-03 8.924e-04 -8.594 < 2e-16 \*\*\*  
## density -5.951e+01 4.060e+01 -1.466 0.1428   
## ph 7.029e-01 2.988e-01 2.352 0.0187 \*   
## sulphates 2.119e+00 2.656e-01 7.979 1.48e-15 \*\*\*  
## alcohol 8.685e-01 5.918e-02 14.676 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 8541.0 on 6496 degrees of freedom  
## Residual deviance: 6706.2 on 6485 degrees of freedom  
## AIC: 6730.2  
##   
## Number of Fisher Scoring iterations: 4

The 8 predictors that we do not reject are: volatile acidity, citric acid, residual sugar, free sulfur dioxide, total sulfur dioxide, pH, sulphates, and alcohol.

We do see that 3 of the predictors have p-values greater than 0.05, so we fail to reject the null hypothesis on those predictors. With that being said, we remove them. Here are the 3 predictors being removed from the model: fixed acidity, chlorides, and density.

glm\_wine = glm(quality ~. - fixed\_acidity - chlorides - density,   
 wine\_df, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ . - fixed\_acidity - chlorides - density,   
## family = binomial, data = wine\_df)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -9.7229883 0.7803911 -12.459 < 2e-16 \*\*\*  
## volatile\_acidity -4.6118529 0.2391270 -19.286 < 2e-16 \*\*\*  
## citric\_acid -0.5627994 0.2312561 -2.434 0.0149 \*   
## residual\_sugar 0.0669939 0.0075820 8.836 < 2e-16 \*\*\*  
## free\_sulfur\_dioxide 0.0162968 0.0025035 6.510 7.54e-11 \*\*\*  
## total\_sulfur\_dioxide -0.0074298 0.0008427 -8.816 < 2e-16 \*\*\*  
## ph 0.3831002 0.2109088 1.816 0.0693 .   
## sulphates 1.9394319 0.2320192 8.359 < 2e-16 \*\*\*  
## alcohol 0.9480857 0.0339259 27.946 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 8541.0 on 6496 degrees of freedom  
## Residual deviance: 6710.3 on 6488 degrees of freedom  
## AIC: 6728.3  
##   
## Number of Fisher Scoring iterations: 4

When we fit a model that removes those those 3 predictors, we see that we also fail to reject the null hypothesis on the pH predictor as it’s p-value is greater than 0.05.

glm\_wine = glm(quality ~. - fixed\_acidity - chlorides - density - ph,   
 wine\_df, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ . - fixed\_acidity - chlorides - density -   
## ph, family = binomial, data = wine\_df)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.5385439 0.4245033 -20.114 < 2e-16 \*\*\*  
## volatile\_acidity -4.5736835 0.2379080 -19.225 < 2e-16 \*\*\*  
## citric\_acid -0.6814050 0.2214555 -3.077 0.00209 \*\*   
## residual\_sugar 0.0648793 0.0074726 8.682 < 2e-16 \*\*\*  
## free\_sulfur\_dioxide 0.0165659 0.0024996 6.627 3.42e-11 \*\*\*  
## total\_sulfur\_dioxide -0.0074854 0.0008419 -8.891 < 2e-16 \*\*\*  
## sulphates 2.0022471 0.2305610 8.684 < 2e-16 \*\*\*  
## alcohol 0.9529509 0.0338748 28.132 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 8541.0 on 6496 degrees of freedom  
## Residual deviance: 6713.6 on 6489 degrees of freedom  
## AIC: 6729.6  
##   
## Number of Fisher Scoring iterations: 4

After removing the pH predictor, all of the predictors left have p-values that are less than 0.05. Therefore we can reject the null hypothesis and say that these predictors have statistical significance when modeling the entire data set with quality as the response variable.  
Now, let us predict the quality response variable using the entire data set. We will convert probabilities to binary predictions, either 0 for poor or 1 for good. We can then use a confusion matrix to predict the quality response variable as either 0 or 1 and measure results.

# Predict on the entire dataset using the logistic model  
pred\_probs = predict(glm\_wine, newdata = wine\_df, type = "response")  
  
# Convert probabilities to binary predictions (0 or 1)  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
  
# Construct confusion matrix:  
conf\_matrix = table(wine\_df$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 1357 1027  
## 1 649 3464

accurate\_pred = (1357 + 3464)/6497  
# Returns 0.7420  
# Results in prediction error of:  
1 - accurate\_pred

## [1] 0.2579652

# 0.2579 or 25.8%

When we predict the quality response variable to be either 0 or 1, we get a prediction rate of 74.20%, or a 25.80% prediction error rate. This is a fairly good prediction rate but we know that this prediction is over-fitted because we used the entire data set to train the model, then used the entire model to test our predictions.

Let us now create a logistic regression using the training data and we can then use the test data to see how accurate our predictions of the quality response variable are.

# Let's use training and test data. Use training data to create model then  
## test data to test the model:  
glm\_wine = glm(quality ~., wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ ., family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 9.817e+01 5.216e+01 1.882 0.0598 .   
## fixed\_acidity 1.472e-01 6.841e-02 2.151 0.0315 \*   
## volatile\_acidity -4.356e+00 3.896e-01 -11.182 < 2e-16 \*\*\*  
## citric\_acid -7.554e-01 3.565e-01 -2.119 0.0341 \*   
## residual\_sugar 1.118e-01 2.340e-02 4.776 1.79e-06 \*\*\*  
## chlorides -1.309e-01 1.429e+00 -0.092 0.9270   
## free\_sulfur\_dioxide 1.486e-02 3.513e-03 4.231 2.32e-05 \*\*\*  
## total\_sulfur\_dioxide -7.163e-03 1.259e-03 -5.689 1.28e-08 \*\*\*  
## density -1.108e+02 5.327e+01 -2.079 0.0376 \*   
## ph 1.037e+00 4.223e-01 2.455 0.0141 \*   
## sulphates 2.372e+00 3.848e-01 6.164 7.11e-10 \*\*\*  
## alcohol 8.186e-01 7.837e-02 10.444 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3352.8 on 3236 degrees of freedom  
## AIC: 3376.8  
##   
## Number of Fisher Scoring iterations: 4

# Looking at p-values, we will leave out following variables:  
# chlorides  
glm\_wine = glm(quality~. - chlorides, wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ . - chlorides, family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 9.897e+01 5.141e+01 1.925 0.0542 .   
## fixed\_acidity 1.482e-01 6.739e-02 2.200 0.0278 \*   
## volatile\_acidity -4.363e+00 3.837e-01 -11.370 < 2e-16 \*\*\*  
## citric\_acid -7.603e-01 3.525e-01 -2.157 0.0310 \*   
## residual\_sugar 1.122e-01 2.288e-02 4.905 9.33e-07 \*\*\*  
## free\_sulfur\_dioxide 1.485e-02 3.509e-03 4.232 2.32e-05 \*\*\*  
## total\_sulfur\_dioxide -7.149e-03 1.249e-03 -5.723 1.04e-08 \*\*\*  
## density -1.116e+02 5.245e+01 -2.128 0.0334 \*   
## ph 1.045e+00 4.138e-01 2.524 0.0116 \*   
## sulphates 2.363e+00 3.730e-01 6.335 2.37e-10 \*\*\*  
## alcohol 8.189e-01 7.825e-02 10.466 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3352.8 on 3237 degrees of freedom  
## AIC: 3374.8  
##   
## Number of Fisher Scoring iterations: 4

# Remove predictors that are barely significant to see if that changes   
## prediction rate to simplify model:  
# citric\_acid  
# density  
# ph  
glm\_wine = glm(quality~. - chlorides - citric\_acid -   
 density - ph, wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ . - chlorides - citric\_acid - density -   
## ph, family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.612504 0.661470 -13.020 < 2e-16 \*\*\*  
## fixed\_acidity -0.031701 0.034960 -0.907 0.365   
## volatile\_acidity -4.273106 0.313836 -13.616 < 2e-16 \*\*\*  
## residual\_sugar 0.065877 0.010687 6.164 7.09e-10 \*\*\*  
## free\_sulfur\_dioxide 0.014873 0.003494 4.257 2.07e-05 \*\*\*  
## total\_sulfur\_dioxide -0.007260 0.001200 -6.052 1.43e-09 \*\*\*  
## sulphates 2.126029 0.339919 6.255 3.99e-10 \*\*\*  
## alcohol 0.948235 0.048107 19.711 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3365.9 on 3240 degrees of freedom  
## AIC: 3381.9  
##   
## Number of Fisher Scoring iterations: 4

# fixed\_acidity no longer significant  
glm\_wine = glm(quality~. - chlorides - citric\_acid -   
 density - ph - fixed\_acidity, wine\_train, family = binomial)  
# Same as:  
glm\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates +  
 alcohol, wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ volatile\_acidity + residual\_sugar + free\_sulfur\_dioxide +   
## total\_sulfur\_dioxide + sulphates + alcohol, family = binomial,   
## data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.877768 0.594132 -14.942 < 2e-16 \*\*\*  
## volatile\_acidity -4.280592 0.313537 -13.653 < 2e-16 \*\*\*  
## residual\_sugar 0.065661 0.010681 6.148 7.87e-10 \*\*\*  
## free\_sulfur\_dioxide 0.015071 0.003489 4.320 1.56e-05 \*\*\*  
## total\_sulfur\_dioxide -0.007084 0.001183 -5.987 2.14e-09 \*\*\*  
## sulphates 2.058056 0.330986 6.218 5.04e-10 \*\*\*  
## alcohol 0.952920 0.047840 19.919 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3366.7 on 3241 degrees of freedom  
## AIC: 3380.7  
##   
## Number of Fisher Scoring iterations: 4

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 671 533  
## 1 312 1733

accurate\_pred = (671 + 1733) / 3249  
# Returns 0.73992  
1 - accurate\_pred

## [1] 0.26008

# Prediction error of 0.26 or 26%.

Following the same process that we did when we fit our first regression model, we first removed the chloride predictor as we could not reject the null hypothesis on it as its p-value was greater than 0.05. When we re-fit the model without the chloride predictor, we had 3 predictors that had p-values barely less than 0.05, meaning they are barely significant enough. Those 3 predictors were: citric acid, density and pH. When we fit the new model without those 3 predictors, we saw that the fixed acidity predictor was no longer significant as we failed to reject the null hypothesis on it.

Our final logistic regression model included the following predictors that all rejected the null hypothesis: volatile acidity, residual sugar, free sulfur dioxide, total sulfure dioxide, sulphates and alcohol. Usin this final model to predict the quality response value as either 0 or 1 led to a 73.99% prediction accuracy, or 26.01% prediction error rate. This prediction rate is almost identical to the prediction accuracy rate from the model fitted on the entire data set plus it is less bias as we tested the model predictions on the test data, which was not used to fit the model.

### Transformations and Interactions

I tried multiple interaction terms on logistic model we fit above using the training data and testing our predictions on the test data. I could produce a model that had greater prediction rate than the one we fitted. Therefore I moved onto some transformations.

# Interaction terms:  
glm\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates\*alcohol,   
 wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ volatile\_acidity + residual\_sugar + free\_sulfur\_dioxide +   
## total\_sulfur\_dioxide + sulphates \* alcohol, family = binomial,   
## data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -6.915839 1.720925 -4.019 5.85e-05 \*\*\*  
## volatile\_acidity -4.274501 0.314156 -13.606 < 2e-16 \*\*\*  
## residual\_sugar 0.065077 0.010683 6.091 1.12e-09 \*\*\*  
## free\_sulfur\_dioxide 0.014952 0.003491 4.283 1.84e-05 \*\*\*  
## total\_sulfur\_dioxide -0.006995 0.001188 -5.890 3.86e-09 \*\*\*  
## sulphates -1.878272 3.281599 -0.572 0.567   
## alcohol 0.759494 0.166335 4.566 4.97e-06 \*\*\*  
## sulphates:alcohol 0.387393 0.321713 1.204 0.229   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3365.2 on 3240 degrees of freedom  
## AIC: 3381.2  
##   
## Number of Fisher Scoring iterations: 5

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 670 534  
## 1 312 1733

accurate\_pred = (670 + 1733) / 3249  
# This produced similar prediction rate of 74%

I first tried a square root transformation on all of the predictors.

# Try sqrt:  
glm\_wine = glm(quality ~ sqrt(volatile\_acidity) + sqrt(residual\_sugar) +  
 sqrt(free\_sulfur\_dioxide) + sqrt(total\_sulfur\_dioxide) +   
 sqrt(sulphates) + sqrt(alcohol), wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ sqrt(volatile\_acidity) + sqrt(residual\_sugar) +   
## sqrt(free\_sulfur\_dioxide) + sqrt(total\_sulfur\_dioxide) +   
## sqrt(sulphates) + sqrt(alcohol), family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -18.90170 1.18697 -15.924 < 2e-16 \*\*\*  
## sqrt(volatile\_acidity) -5.75764 0.39890 -14.434 < 2e-16 \*\*\*  
## sqrt(residual\_sugar) 0.34742 0.05421 6.409 1.47e-10 \*\*\*  
## sqrt(free\_sulfur\_dioxide) 0.24693 0.04080 6.053 1.43e-09 \*\*\*  
## sqrt(total\_sulfur\_dioxide) -0.17841 0.02452 -7.276 3.45e-13 \*\*\*  
## sqrt(sulphates) 3.32665 0.51496 6.460 1.05e-10 \*\*\*  
## sqrt(alcohol) 6.28658 0.30664 20.502 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3322.8 on 3241 degrees of freedom  
## AIC: 3336.8  
##   
## Number of Fisher Scoring iterations: 4

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 691 513  
## 1 312 1733

accurate\_pred = (691 + 1733) / 3249  
# 74.6%. Highest prediction rate yet.

The square root model produced a 74.6% prediction rate.

I also performed a squared transformation on all of the predictors in the model.

glm\_wine = glm(quality ~ I(volatile\_acidity)^2 + I(residual\_sugar)^2 +  
 I(free\_sulfur\_dioxide)^2 + I(total\_sulfur\_dioxide)^2 +   
 I(sulphates)^2 + I(alcohol)^2, wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ I(volatile\_acidity)^2 + I(residual\_sugar)^2 +   
## I(free\_sulfur\_dioxide)^2 + I(total\_sulfur\_dioxide)^2 + I(sulphates)^2 +   
## I(alcohol)^2, family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.877768 0.594132 -14.942 < 2e-16 \*\*\*  
## I(volatile\_acidity) -4.280592 0.313537 -13.653 < 2e-16 \*\*\*  
## I(residual\_sugar) 0.065661 0.010681 6.148 7.87e-10 \*\*\*  
## I(free\_sulfur\_dioxide) 0.015071 0.003489 4.320 1.56e-05 \*\*\*  
## I(total\_sulfur\_dioxide) -0.007084 0.001183 -5.987 2.14e-09 \*\*\*  
## I(sulphates) 2.058056 0.330986 6.218 5.04e-10 \*\*\*  
## I(alcohol) 0.952920 0.047840 19.919 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3366.7 on 3241 degrees of freedom  
## AIC: 3380.7  
##   
## Number of Fisher Scoring iterations: 4

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 671 533  
## 1 312 1733

accurate\_pred = (671 + 1733) / 3249  
# 73.99%.

The model that squared all of the predictors produced a 73.99% prediction rate.

I performed a cubed transformation on all of the predictors.

glm\_wine = glm(quality ~ I(volatile\_acidity)^3 + I(residual\_sugar)^3 +  
 I(free\_sulfur\_dioxide)^3 + I(total\_sulfur\_dioxide)^3 +   
 I(sulphates)^3 + I(alcohol)^3, wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ I(volatile\_acidity)^3 + I(residual\_sugar)^3 +   
## I(free\_sulfur\_dioxide)^3 + I(total\_sulfur\_dioxide)^3 + I(sulphates)^3 +   
## I(alcohol)^3, family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -8.877768 0.594132 -14.942 < 2e-16 \*\*\*  
## I(volatile\_acidity) -4.280592 0.313537 -13.653 < 2e-16 \*\*\*  
## I(residual\_sugar) 0.065661 0.010681 6.148 7.87e-10 \*\*\*  
## I(free\_sulfur\_dioxide) 0.015071 0.003489 4.320 1.56e-05 \*\*\*  
## I(total\_sulfur\_dioxide) -0.007084 0.001183 -5.987 2.14e-09 \*\*\*  
## I(sulphates) 2.058056 0.330986 6.218 5.04e-10 \*\*\*  
## I(alcohol) 0.952920 0.047840 19.919 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3366.7 on 3241 degrees of freedom  
## AIC: 3380.7  
##   
## Number of Fisher Scoring iterations: 4

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 671 533  
## 1 312 1733

accurate\_pred = (671 + 1733) / 3249  
# 73.99%.

The cubed model produced a prediction rate of 73.99%, which was the same as the model that squared all of the predictors above.

I then performed a log transformation on all of the predictors.

# Try log transformation:  
glm\_wine = glm(quality ~ log(volatile\_acidity) + log(residual\_sugar) +  
 log(free\_sulfur\_dioxide) + log(total\_sulfur\_dioxide) +   
 log(sulphates) + log(alcohol), wine\_train, family = binomial)  
summary(glm\_wine)

##   
## Call:  
## glm(formula = quality ~ log(volatile\_acidity) + log(residual\_sugar) +   
## log(free\_sulfur\_dioxide) + log(total\_sulfur\_dioxide) + log(sulphates) +   
## log(alcohol), family = binomial, data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -23.75355 1.20227 -19.757 < 2e-16 \*\*\*  
## log(volatile\_acidity) -1.76585 0.11937 -14.794 < 2e-16 \*\*\*  
## log(residual\_sugar) 0.36776 0.05852 6.285 3.28e-10 \*\*\*  
## log(free\_sulfur\_dioxide) 0.76844 0.10092 7.614 2.65e-14 \*\*\*  
## log(total\_sulfur\_dioxide) -0.85100 0.10715 -7.942 1.99e-15 \*\*\*  
## log(sulphates) 1.24522 0.19322 6.444 1.16e-10 \*\*\*  
## log(alcohol) 10.31773 0.48966 21.071 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 4256.8 on 3247 degrees of freedom  
## Residual deviance: 3292.5 on 3241 degrees of freedom  
## AIC: 3306.5  
##   
## Number of Fisher Scoring iterations: 4

pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 710 494  
## 1 320 1725

accurate\_pred = (710 + 1725) / 3249  
# 74.96%.

This model produced a prediction rate of 74.96%. This model has produced the highest prediction rate thus far compared to the previous models fitted using the training data. Let us know perform a linear discrimination analysis using our training data to predict the test data.

## Linear Discrimination Analysis (LDA)

Now we will run a linear discrimination analysis using our training data. We will use the same predictors that we obtained from our logistic regression model from the section above.

## lda\_class 0 1  
## 0 665 319  
## 1 539 1726

accurate\_pred = mean(lda\_class == wine\_test$quality)  
# Prediction rate = 0.7371 or 73.56%  
# Prediction error rate of:  
1 - accurate\_pred

## [1] 0.2640813

# 0.2629 or 26.29%

The linear discrimination analysis model produced an accurate prediction rate of 73.56% which is very close to the prediction rate produced by our logistic regression model that we already fit. Let us now move onto a quadratic descriptive analysis.

## Quadratic Descriptive Analysis (QDA)

Let us see if we can enhance our prediction rate by fitting a a quadratic descriptive analysis model to our training data.

## qda\_class 0 1  
## 0 622 298  
## 1 582 1747

accurate\_pred = mean(qda\_class == wine\_test$quality)  
# Prediction rate of 0.7291 or 72.91%  
# Prediction error rate of:  
1 - accurate\_pred

## [1] 0.2708526

# 0.2709 or 27.09%

This model produced a prediction rate of 72.91%. Now let us fit a K-nearest neighbor model using various values of K.

## K-Nearest Neighbors Model

We will now fit a K-nearest neighbors model using the training data to then predict the test data. We will use the variables selected from our logistic regression model: volatile acidity, residual sugar, free sulfur dioxide, total sulfur dioxide, sulphates and alcohol. We will use a few different numbers of K. We will start with K = 1, meaning the instance will be assigned to the same class as its single nearest neighbor.

K = 1:

This produced a prediction rate of 69.95%. This is the worst prediction rate thus far.

Try K = 2, where the model will check the closest two neighbors for each instance:

K = 2 KNN model produced a prediction rate of 64.45%. This is now the worst prediction rate that we have seen so far.

Try K = 5:

K = 5 produced a prediction rate of 66.57%. So far, the KNN model has had the worst overall prediction rate of our fitted models, regardless of what value of K we use. We will now fit a naive Bayes model.

## Naive Bayes Model

We will now fit a naive Bayes model using the same predictors that we just used in the KNN model in the previous section above.

accurate\_pred = mean(nb\_class == wine\_test$quality)  
# Prediction accuracy is 70.33%  
# Prediction error rate is:  
1 - accurate\_pred

## [1] 0.2967067

# 29.67%

Our naive Bayes model produced a prediction rate of 70.33%. Now we will look at the poisson regression model.

## Poisson Regression

Using the same predictors that we used in our KNN and naive Bayes models, we will fit a poisson regression model.

poiss\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates +  
 alcohol, wine\_train, family = poisson)  
summary(poiss\_wine)

##   
## Call:  
## glm(formula = quality ~ volatile\_acidity + residual\_sugar + free\_sulfur\_dioxide +   
## total\_sulfur\_dioxide + sulphates + alcohol, family = poisson,   
## data = wine\_train)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -2.8122012 0.2671929 -10.525 < 2e-16 \*\*\*  
## volatile\_acidity -1.5968655 0.1746346 -9.144 < 2e-16 \*\*\*  
## residual\_sugar 0.0200026 0.0055070 3.632 0.000281 \*\*\*  
## free\_sulfur\_dioxide 0.0041270 0.0018206 2.267 0.023395 \*   
## total\_sulfur\_dioxide -0.0021914 0.0006579 -3.331 0.000865 \*\*\*  
## sulphates 0.5519403 0.1536420 3.592 0.000328 \*\*\*  
## alcohol 0.2433312 0.0190649 12.763 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for poisson family taken to be 1)  
##   
## Null deviance: 1867.2 on 3247 degrees of freedom  
## Residual deviance: 1575.4 on 3241 degrees of freedom  
## AIC: 5725.4  
##   
## Number of Fisher Scoring iterations: 5

# We can reject null hypothesis for all predictors.  
pred\_probs = predict(poiss\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)

## pred\_classes  
## 0 1  
## 0 698 506  
## 1 346 1699

accurate\_pred = (698 + 1699) / (3249)  
# Accurate prediction percentage is 73.78%  
# Prediction error rate is:  
1 - accurate\_pred

## [1] 0.2622345

# 26.22%.

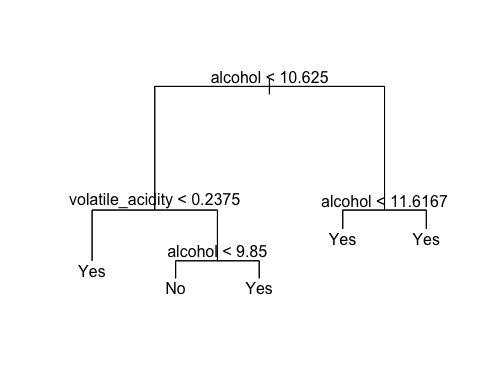
Since the p-values of all of the predictors in our poisson regression model are less than 0.05, we can reject null hypothesis each of them and dictate them significant predictors. This model produced a prediction error rate of 73.78%. Let us know fit our data to classification trees.

## Classification Trees

For our classification trees that we will be creating, we need to re-read in our data. Instead of having our quality response variable be represented by either 0 for poor or 1 for good, we want to change our response variable to “good” and want our reponses variables to be either “Yes” or “No”. We will also convert our training and test data set to have these response variables to keep continuity with our analysis so we can confidently compare prediction rates.

# New training and test data:  
set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
wine\_train = wine\_df[train,]  
wine\_test = wine\_df[-train,]

Now that we have re-fined our data, we can now perform a classification tree. Let us perform a classification tree on the entire wine quality data set using all of the predictors.



We see that our classification tree has 5 terminal nodes and a training error rate of 26.43%, which means there is a training prediction rate of 73.57%. We also see that alcohol is the most important predictor in our classification tree as it shows up three seperate times on our tree. Let us now fit our classification tree using the training data and test our predictions using our test data.

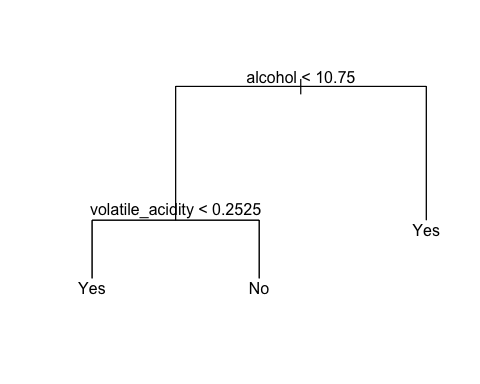
## pred\_tree No Yes  
## No 794 480  
## Yes 410 1565

accurate\_pred = (794 + 1565) / (3249)  
# Accurate prediction rate is 72.61%  
# Prediction error rate is:  
1 - accurate\_pred

## [1] 0.2739304

# 27.39%.

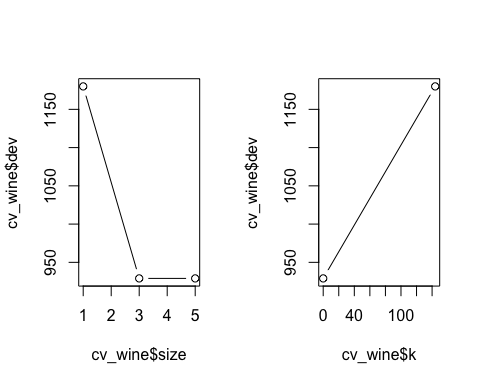
This classification tree produced a prediction rate of 72.61%. Now, let us prune our tree using cross-validation to see if that increases our prediction test rate.



# Test pruned tree performance on test data set.  
pred\_tree = predict(prune\_wine, wine\_test, type = "class")  
table(pred\_tree, wine\_test$good)

##   
## pred\_tree No Yes  
## No 794 480  
## Yes 410 1565

accurate\_pred = (794 + 1565) / (3249)



This pruned tree produced the same prediction rate of 72.61% as our first tree. Since this pruned tree only has 3 terminal nodes, we will use this tree over our first classification tree that had 5 terminal nodes, becuase it is easier to interpret. Now, let us look at the bagging model.

## Bagging Models

We will now fit our model using the bagging approach using the randomForest class in R.

set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11,  
 importance = TRUE)  
bag\_wine

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 11, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 11  
##   
## OOB estimate of error rate: 19.98%  
## Confusion matrix:  
## No Yes class.error  
## No 827 353 0.2991525  
## Yes 296 1772 0.1431335

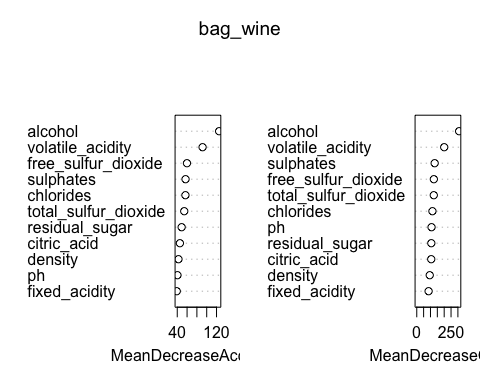
# 500 trees considering all 11 predictors   
# Out-of-bag estimate of error rate is: 19.98  
# Accuracy rate is:  
1 - 0.1998

## [1] 0.8002

# 80 %. Highest prediction rate so far. But this is only on training set.  
# Try on test set  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)

##   
## pred\_bag No Yes  
## No 824 257  
## Yes 380 1788

accurate\_pred = (824 + 1788) / (3249)  
# 80.39% accurate prediction rate.  
  
varImpPlot(bag\_wine)



The bagging model produced a prediction rate of 80.39%, which is the highest prediction rate that we have seen thus far! As we move forward with our models, we will see if we can get any higher than 80.39%.

Now, let us fit our bagging model with 100 trees:

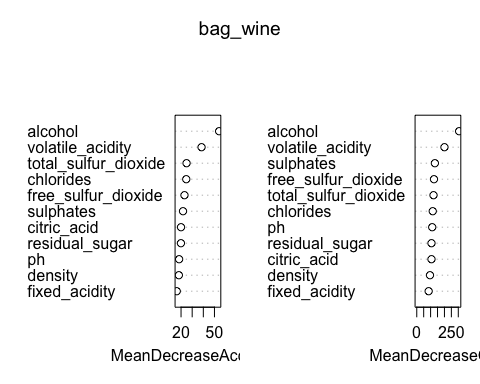
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11, ntree = 100,  
 importance = TRUE)  
bag\_wine

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 11, ntree = 100, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 100  
## No. of variables tried at each split: 11  
##   
## OOB estimate of error rate: 20.14%  
## Confusion matrix:  
## No Yes class.error  
## No 816 364 0.3084746  
## Yes 290 1778 0.1402321

# Slightly larger OOB estimate of error rate:  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)

##   
## pred\_bag No Yes  
## No 826 262  
## Yes 378 1783

accurate\_pred = (826 + 1783) / (3249)  
# 80.30% prediction rate  
  
varImpPlot(bag\_wine)



This bagging model with 100 trees produced a prediction rate of 80.3%. Very close to our bagginf model with 500 trees.

Now, let us fit our bagging model with 25 trees:

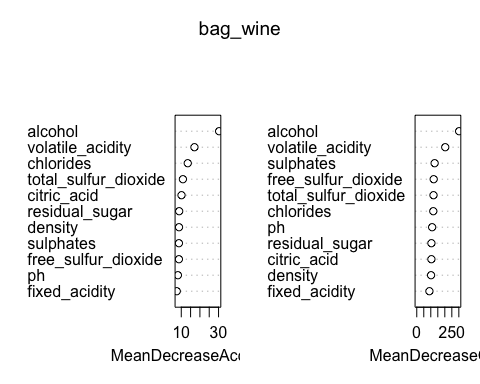
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11, ntree = 25,  
 importance = TRUE)  
bag\_wine

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 11, ntree = 25, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 25  
## No. of variables tried at each split: 11  
##   
## OOB estimate of error rate: 22.35%  
## Confusion matrix:  
## No Yes class.error  
## No 790 390 0.3305085  
## Yes 336 1732 0.1624758

# Slightly larger OOB estimate of error rate:  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)

##   
## pred\_bag No Yes  
## No 808 271  
## Yes 396 1774

accurate\_pred = (808 + 1744) / (3249)  
# 79.47% prediction rate  
  
  
# See most important variables  
varImpPlot(bag\_wine)



This model produced a 79.47% prediction rate. Very close to our bagging approach with 500 and 100 trees. Now, let us fit various random forest models to see if we increase that prediction rate percentage.

## Random Forest Models

We will now fit a random forest model, which is very similar to our bagging model, but we will use smaller predictor values compared to using all 11 predictors. We will first try with 5 variables.

set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5,  
 importance = TRUE)  
rf\_wine

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 5, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 5  
##   
## OOB estimate of error rate: 19.83%  
## Confusion matrix:  
## No Yes class.error  
## No 819 361 0.3059322  
## Yes 283 1785 0.1368472

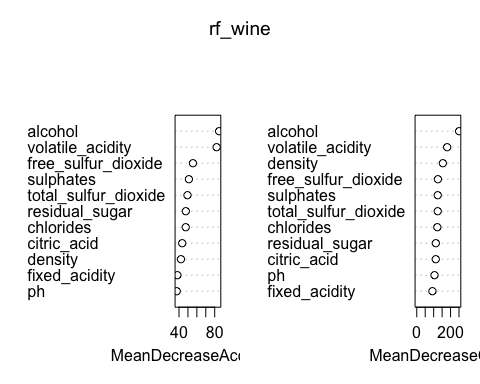
# OOB Estimate of error rate: 19.83%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)

##   
## pred\_rf No Yes  
## No 823 253  
## Yes 381 1792

accurate\_pred = (823 + 1792) / (3249)  
accurate\_pred

## [1] 0.804863

# Prediction rate: 80.49%  
  
  
# See most important variables  
varImpPlot(rf\_wine)



This random forest model produced a prediction rate of 80.49%. Now let us try with 5 predictors and 100 trees.

set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5, ntree = 100,  
 importance = TRUE)  
rf\_wine

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 5, ntree = 100, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 100  
## No. of variables tried at each split: 5  
##   
## OOB estimate of error rate: 20.26%  
## Confusion matrix:  
## No Yes class.error  
## No 816 364 0.3084746  
## Yes 294 1774 0.1421663

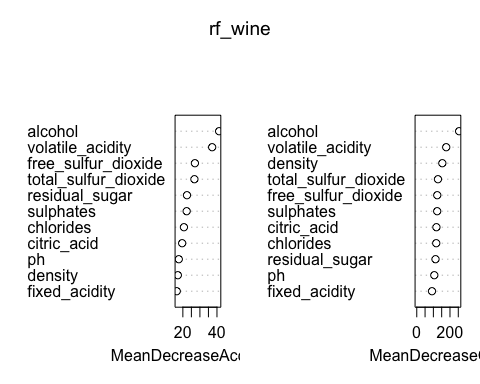
# OOB Estimate of error rate: 20.26%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)

##   
## pred\_rf No Yes  
## No 837 249  
## Yes 367 1796

accurate\_pred = (837 + 1796) / (3249)  
accurate\_pred

## [1] 0.8104032

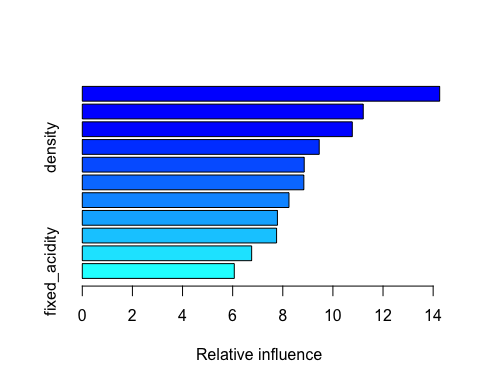
# See most important variables  
varImpPlot(rf\_wine)



This random forest model produced a prediction rate of 81.04%. This has now become our highest prediction rate so far! Alcohol and volatile\_acidity seem to be the two most important predictors in this model. After a few more attempts of changing the number of trees and number of variables used in the model, we were not able to produce a higher prediction rate than the 81.04%. Now we will fit a boosted model.

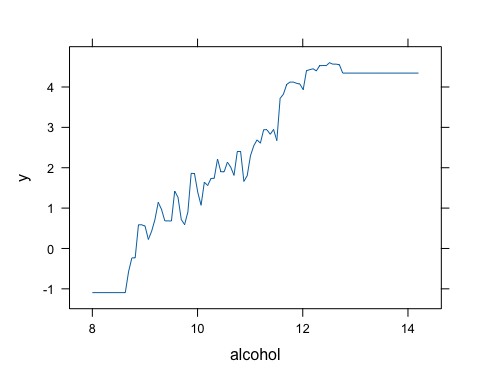
## Boosting Model

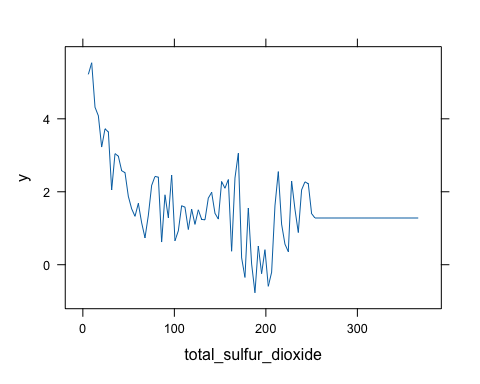
In order to fit a boosted model, we will need to re-read in our data again and change the quality response variable back to a bernoulli response variable where it takes on the value of 0 if the wine quality is poor (scores 3-6) and takes on the value of 1 if the wine quality is good (scores 7-9). We will also re-create our test data used in the previous regression models before we fit our classification trees.

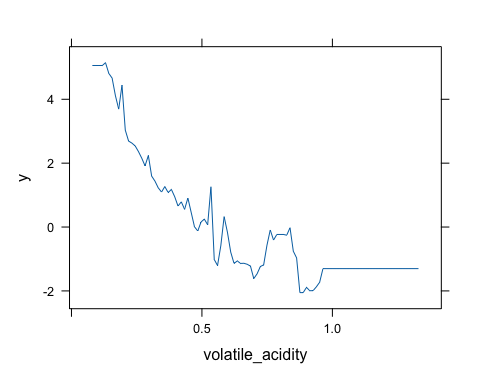
When fitting a boosted model, our default shrinking parameter lambda will equal 0.001. We will also test out our boosted models using shrinking parameters: 0.01, 0.1, and 0.2 and will compare the prediction rates for each model.

## var rel.inf  
## alcohol alcohol 14.260587  
## total\_sulfur\_dioxide total\_sulfur\_dioxide 11.210168  
## volatile\_acidity volatile\_acidity 10.773269  
## density density 9.453383  
## free\_sulfur\_dioxide free\_sulfur\_dioxide 8.855012  
## residual\_sugar residual\_sugar 8.839961  
## chlorides chlorides 8.247174  
## sulphates sulphates 7.785987  
## ph ph 7.752294  
## citric\_acid citric\_acid 6.757711  
## fixed\_acidity fixed\_acidity 6.064454

# We see that alcohol, total\_sulfur dioxide, and volatile\_acidity  
# are most important variables in boosted model.







In this model above, we see that the alcohol, total sulfur dioxide and volatile acidity are the most important variables when it comes to predicting the quality response variable. Now let us find the test prediction percentage.

## pred\_classes 0 1  
## 0 828 312  
## 1 376 1733

accurate\_pred = (828 + 1733) / (3249)  
accurate\_pred

## [1] 0.7882425

This model produced a prediction rate of 78.82%. Now let us try with the shrinking parameter lambda equal to 0.01

## pred\_classes 0 1  
## 0 791 284  
## 1 413 1761

accurate\_pred = (791 + 1761) / (3249)  
accurate\_pred

## [1] 0.7854725

This model produced a prediction rate of 78.54%. Let us try with the 0.1 shrinking parameter.

## pred\_classes 0 1  
## 0 828 312  
## 1 376 1733

accurate\_pred = (828 + 1733) / (3249)  
accurate\_pred

## [1] 0.7882425

This model produced a prediction rate of 78.82% which is the same prediction rate as the boosted model with the 0.001 shrinking parameter. Let us now try with the 0.2 shrinking parameter.

## pred\_classes 0 1  
## 0 812 313  
## 1 392 1732

accurate\_pred = (812 + 1732) / (3249)  
accurate\_pred

## [1] 0.7830102

This model produced a prediction rate of 78.30%. The boosted model with the highest prediction rate occured when we used shrinking paramater value of 0.001 and 0.1. The prediction rate for both of those models was 78.82%. Now we will turn our analysis to our last model that we will fit, which is our Bayesian Additive Regression Tree (BART).

## Bayesian Additive Regression Tree (BART)

In order to fit our BART model, we will need to create a set of both training and test matrices using our training and test data from our original wine data.

Once our training and testing data matrices are created, we can fit our BART model using the training data and test its predictions on the test data.

# Use $prob.test.mean extraction that represents the mean probs across  
## iterations or trees for each observation in test set  
pred\_classes = ifelse(pred\_bart$prob.test.mean > 0.5, 1, 0)  
table(pred\_classes, ytest)

## ytest  
## pred\_classes 0 1  
## 0 754 313  
## 1 450 1732

accurate\_pred = (754 + 1732) / 3249  
accurate\_pred

## [1] 0.7651585

This BART model produced a prediction rate of 76.52%. Now we can look at all of the models that we fit and see which one produced the highest prediction rate to help us determine which model we will choose that best fits our data of predicting wine quality as either poor or good.

## Prediction Rate Summary

Let us look at each prediction rate based on what model we fit using our qualitative regression methods from each section above.

Logistic regression with log transformations on predictors used: 74.96%.

Linear Discrimination Analysis (LDA): 73.56%.

Quadratic Descriptive Analysis (QDA): 72.91%. K-Nearest Neighbors Model: 69.95% when K = 1.  
Pruned classification tree with 3 terminal nodes: 72.61%.

Bagging Model: 80.39%.

Random forest using 5 most important variables: 81.04%.

Boosting Model: 78.82% when shrinking parameter is either 0.001 or 0.1.

BART Model: 76.52%.

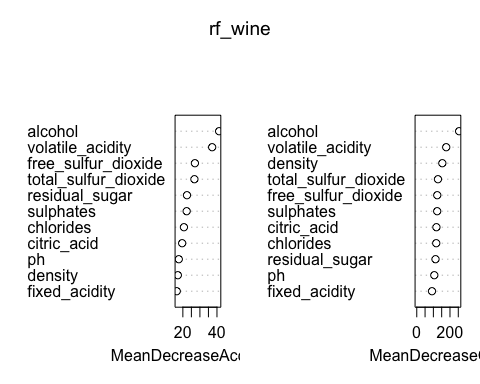
As we can see, the model that produced the highest prediction rate of 81.04% was our random forest using the 5 most important variables in our model and 100 trees. This means that our fitted model has a prediction error rate of 18.96% and the five predictors used were alcohol, volatile acidity, free sulfur dioxide, total sulfur dioxide and residual sugar.

## The following objects are masked from wine\_df (pos = 3):  
##   
## alcohol, chlorides, citric\_acid, density, fixed\_acidity,  
## free\_sulfur\_dioxide, ph, quality, residual\_sugar, sulphates,  
## total\_sulfur\_dioxide, volatile\_acidity

##   
## Call:  
## randomForest(formula = good ~ . - quality, data = wine\_train, mtry = 5, ntree = 100, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 100  
## No. of variables tried at each split: 5  
##   
## OOB estimate of error rate: 20.26%  
## Confusion matrix:  
## No Yes class.error  
## No 816 364 0.3084746  
## Yes 294 1774 0.1421663

##   
## pred\_rf No Yes  
## No 837 249  
## Yes 367 1796

## [1] 0.8104032



## Conclusion

Compared to our quantitative regression analysis, the prediction rate percentages varied slightly more in our qualitative regression analysis. than what we saw in our condition based maintenance regression from our first portion of our project. We saw some prediction rate percentages range from approximately 72% to approximately 81%. This may not seem like a big difference between the models and for the most part, we would be happy with a prediction rate of 72% as that is better than flipping a coin and havina 50% chance have predicting whether the wine quality will be poor or good.

Our final random forest model that we chose to model and predict our data outperformed random guessing by about 31%, with a prediction rate of approximately 81%. Any one of these models would be a better model to use than using random guessing when it comes to predicting the quality of these wines. This shows that the qualitative regression models that we fit all have statistical significance when it comes to predicting our response variable. And that was our goal, to fit qualitative regression models, to predict and test our predictions to see how well we can predict the wine quality of our wine samples in northern Portugal. If we were to further this analysis using the model we fit using the data from 2009, we would try and obtain a new data set of all new wine qualities and put our model to the test to see how well we could predict wine quality without training our model on the new data.

# Project Conclusion

In this research project, we were able to show how we can use both quantitative and qualitative regression models to model data and accurately predict our response variables. Using the various classes, functions, and models in Rstudio, we were able to fit a square root transformed linear regression model in our quantitative regression problem that had a mean squared test error rate of 0.000000969 when it came to predicting our gas turbine decay state coefficient.

We were also able to fit a random forest model in our qualitative regression that was able to accurately predict our wine quality response variable as either poor or good 81.04% of the time. Rather than assuming that certain predictors are important when it comes to modeling data, we were able to use statistics and test predictions to measure how accurate our fitted models truly were.

# Appendix

Coraddu,Andrea, Oneto,Luca, Ghio,Alessandro, Savio,Stefano, Anguita,Davide, and Figari,Massimo. (2014). Condition Based Maintenance of Naval Propulsion Plants. UCI Machine Learning Repository. <https://doi.org/10.24432/C5K31K>.

Cortez,Paulo, Cerdeira,A., Almeida,F., Matos,T., and Reis,J.. (2009). Wine Quality. UCI Machine Learning Repository. <https://doi.org/10.24432/C56S3T>.

Code Used for Quantitative Analysis

# Quantitative Linear Regression Final Project R Code -------------------------  
  
# Data link: https://archive.ics.uci.edu/dataset/316/condition+based+maintenance+of+naval+propulsion+plants  
  
# Citation:  
# Coraddu,Andrea, Oneto,Luca, Ghio,Alessandro, Savio,Stefano, Anguita,Davide, and Figari,Massimo. (2014).   
# Condition Based Maintenance of Naval Propulsion Plants. UCI Machine Learning Repository.  
# https://doi.org/10.24432/C5K31K.  
  
# Use predictors to predict Gas Turbine (TB) decay state coefficient.  
  
# Read in and clean up data -----------------------------------------------  
cbm\_df = read.table("Final\_Project/uci\_bm\_dataset/data.txt", header = FALSE, colClasses = "numeric")  
colnames(cbm\_df) = tolower(colnames(cbm\_df))  
dim(cbm\_df)  
# 11934 observations  
# 18 variables  
  
# Variables names:  
# v1 - Lever position (lp) [ ]  
# v2 - Ship speed (v) [knots]  
# v3 - Gas Turbine shaft torque (GTT) [kN m]  
# v4 - Gas Turbine rate of revolutions (GTn) [rpm]  
# v5 - Gas Generator rate of revolutions (GGn) [rpm]  
# v6 - Starboard Propeller Torque (Ts) [kN]  
# v7 - Port Propeller Torque (Tp) [kN]  
# v8 - HP Turbine exit temperature (T48) [C]  
# v9 - GT Compressor inlet air temperature (T1) [C]  
# 10 - GT Compressor outlet air temperature (T2) [C]  
# 11 - HP Turbine exit pressure (P48) [bar]  
# 12 - GT Compressor inlet air pressure (P1) [bar]  
# 13 - GT Compressor outlet air pressure (P2) [bar]  
# 14 - Gas Turbine exhaust gas pressure (Pexh) [bar]  
# 15 - Turbine Injecton Control (TIC) [%]  
# 16 - Fuel flow (mf) [kg/s]  
# 17 - GT Compressor decay state coefficient.  
# 18 - GT Turbine decay state coefficient.   
  
  
names(cbm\_df)  
# V18 is our response variable  
# V17 is also a response variable so we want to remove that column and just focus  
## on V18.  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v17")]  
names(cbm\_df)  
dim(cbm\_df)  
# 11934 observations  
# 17 variables. 16 predictors. 1 response.  
  
table(cbm\_df$v17)  
# 459 occurrences of each response output in data set. 26 different responses  
# for V18.  
  
# Response Variable Summary -----------------------------------------------  
# Using 16 predictors to predict Gas Turbine (TB) decay state coefficient extracted  
# from experiments that have been carried out by means of a numerical simulator of   
# a naval vessel (Frigate) characterized by a Gas Turbine (GT) propulsion plant.  
# Use training data and fit model that accurately predicts test data.  
  
# Linear Regression -------------------------------------------------------  
cbm\_lm = lm(v18 ~., cbm\_df)  
summary(cbm\_lm)  
# v7, v9 and v12 are showing as NA coefficients.   
# This indicates muilticollinearity or perfect collinearity, meaning that one or   
## more predictors are linearly dependent on others.   
# Check for columns with near-zero or zero variance  
near\_zero\_var = colSums(var(cbm\_df)) < 1e-10  
# Display columns with near-zero or zero variance  
print(names(cbm\_df)[near\_zero\_var])  
# v9 and v12 were returned. Yet v7 also showing NA  
cbm\_df$v9  
# All observations in v9 are the same at 288.  
cbm\_df$v12  
# Same with v12 where all the same at 0.998.  
  
# Let's remove v9 and v12.  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v9")]  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v12")]  
# For v7, lets look at correlation matrix  
cor(cbm\_df)  
install.packages("knitr")  
install.packages("corrplot")   
library(knitr)  
library(corrplot)  
correlation\_matrix = cor(cbm\_df, use = "pairwise.complete.obs")  
# v6 and v7 are perfectly correlated.   
are\_same = all(cbm\_df$v6 == cbm\_df$v7)  
if (are\_same) {  
 print("v6 and v7 are exactly the same.")  
} else {  
 print("v6 and v7 are different.")  
}  
# v6 = Starboard Propeller Torque (Ts) [kN] and v7 = Port Propeller Torque (Tp) [kN]  
## These are the same values, so let's remove v7.  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v7")]  
  
# Training and Test Data --------------------------------------------------  
# Here is training and test data that we will use. There is 11934 observations.  
# Split data in half for training and test data.  
11934 / 2  
set.seed(1)  
train = sample(1:nrow(cbm\_df), 5967)  
cbm\_train = cbm\_df[train,]  
cbm\_test = cbm\_df[-train,]  
  
# Linear Regression Cont. -------------------------------------------------  
  
# Let's try best subset selection on entire data:  
library(leaps)  
best\_cbm = regsubsets(v18 ~., cbm\_df, nvmax = 13)  
best\_cbm\_summary = summary(best\_cbm)  
names(best\_cbm\_summary)  
# This returns R^2, RSS, adjusted R^2, Cp and BIC.  
best\_cbm\_summary$rsq  
# Plot RSS, adjusted R^2, Cp and BIC  
par(mfrow = c(2, 2))  
plot(best\_cbm\_summary$rss, xlab = "Number of Variables",  
 ylab = "RSS", type = "l")  
plot(best\_cbm\_summary$adjr2, xlab = "Number of Variables",  
 ylab = "Adjusted RSq", type = "l")  
# Plot red dot to indicate model with largest adjusted R^2  
which.max(best\_cbm\_summary$adjr2)  
# Returned 12  
points(12, best\_cbm\_summary$adjr2[12], col = "red", cex = 2, pch = 20)  
# Plot Cp and BIC stats and indicate models with smallest stat  
plot(best\_cbm\_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")  
which.min(best\_cbm\_summary$cp)  
# Returns 12  
points(12, best\_cbm\_summary$cp[12], col = "red", cex = 2,  
 pch = 20)  
which.min(best\_cbm\_summary$bic)  
# Returns 12  
plot(best\_cbm\_summary$bic, xlab = "Number of Variables",  
 ylab = "BIC", type = "l")  
points(12, best\_cbm\_summary$bic[12], col = "red", cex = 2,  
 pch = 20)  
# Best subset selection says to have 12 variables.  
  
# Use forward selection:  
fwd\_cbm = regsubsets(v18 ~., cbm\_df, nvmax = 13, method = "forward")  
summary(fwd\_cbm)  
# Best 12 predictor model removes v1  
  
# Use backward selection:  
bwd\_cbm = regsubsets(v18 ~., cbm\_df, nvmax = 13, method = "backward")  
summary(bwd\_cbm)  
# Best 12 predictor model only removes v1  
# Since v1 was not included in both the forward and backward selection models,   
# I will keep that variable in mind as the first one removed in the models moving   
# forward. My approach for the multiple linear regression will combine both forward   
# and backward selection by using training data on test data while examining p-values   
# and absolute values of t to add and remove variables from models. My goal is to   
# try and make the model as simple as possible by maximizing the R^2 value and   
# minimizing the test mean squared error.  
  
# Only 14 variables.  
cbm\_lm = lm(v18 ~., cbm\_df)  
summary(cbm\_lm)  
# All p-values are less than 0.05, so we can reject the null hypothesis and   
## say all values are statistically signicant.  
# R-squared value is 91.1%  
# v1 barely passes hyothesis test.  
# Try model without v1  
cbm\_lm = lm(v18 ~. -v1, cbm\_df)  
summary(cbm\_lm)  
# All p-values are less than 0.05 once again.  
# R-squared value still 91.1%  
# Looking at coefficients, we see some ones with negative coefficients:  
## v6, v8, v13, and v15  
# Try running model with just positive coefficients  
cbm\_lm = lm(v18 ~. - v1 - v6 - v8 - v13 - v15, cbm\_df)  
summary(cbm\_lm)  
# This makes model way worse.  
# Go back to model with all but v1 variable  
cbm\_lm = lm(v18 ~. -v1, cbm\_df)  
summary(cbm\_lm)  
  
# Let's try letting our train data predict our test model   
train\_cbm\_lm = lm(v18 ~. - v1, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# Means squared error is almost close to 0.  
# This means that this model did a really good job at predicting response variable  
## in test data.  
  
# Can we lose some predictors on training data to make data more interpretable?  
# Let's then remove each predictor, one at a time, if there p-values are less than  
## 0.05 or if all p-values are less than, 0.05, remove predictor with smallest  
### absolute t-value.  
  
# Remove v1 from like we did in previous model.  
train\_cbm\_lm = lm(v18 ~. - v1, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values pass hypothesis test.  
# Test MSE is close to zero.  
# R^2 is 91.25%  
# Let's remove predictor with lowest absolute t-value which is v14  
  
# Remove v14  
train\_cbm\_lm = lm(v18 ~. - v1 - v14, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values besides pass hypothesis test.  
# Test MSE is still close to 0.  
# R^2 is 91.1%  
# Let's remove predictor with lowest absolute t-value which is v10.  
  
# Remove v10  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 90.81%%  
# Let's remove predictor with lowest absolute t-value which is v11  
  
# Remove v11  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10 - v11, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 89.93%%  
# Let's remove predictor with lowest absolute t-value which is v15  
  
# Remove v15  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10 - v11 - v15, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 88.43%%  
# Let's remove predictor with lowest absolute t-value which is v16  
  
# Remove v16  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10 - v11 - v15 - v16, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 76.48%%  
# R-squared value dropped dramatically.   
# I want to keep that value above 90%.  
  
# So let's go back to most recent model that was above 90% for R-squared value:  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10, cbm\_train)  
summary(train\_cbm\_lm)  
# All predictors p-values are less than 0.05  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# MSE still close to 0.  
# R-squared is 90.81%%  
# 10 predictors in this model. 2 less predictors than previous model that just   
## dropped v1 predictor.  
  
  
# Create a scatter plot comparing actual vs. predicted values  
par(mfrow = c(1, 1))  
plot\_df = data.frame(Actual = cbm\_test$v18, Predicted = pred\_cbm\_lm)  
  
ggplot(plot\_df, aes(x = Actual, y = Predicted)) +  
 geom\_point() +  
 geom\_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") + # Adds a line of perfect prediction  
 labs(x = "Actual Gas Turbine Decay State Coefficient",   
 y = "Predicted Gast Turbine Decay State Coefficient",   
 title = "Actual vs. Predicted") +  
 theme\_minimal()  
  
# Plot this model on entire data set:  
library(ggplot2)  
cbm\_lm = lm(v18 ~. - v1 - v14 - v10, cbm\_df)  
par(mfrow = c(2, 2))  
plot(cbm\_lm)  
  
# Try some transformations of predictors on model we have  
train\_cbm\_lm = lm(v18 ~. - v1 - v14 - v10, cbm\_train)  
summary(train\_cbm\_lm)  
# Same as this model:  
train\_cbm\_lm = lm(v18 ~ v2 + v3 + v4 + v5 + v6 + v8 + v11 +  
 v13 + v15 + v16, cbm\_train)  
summary(train\_cbm\_lm)  
  
# Try sqrt():  
train\_cbm\_lm = lm(v18 ~ sqrt(v2) + sqrt(v3) + sqrt(v4) + sqrt(v5) + sqrt(v6) +  
 sqrt(v8) + sqrt(v11) + sqrt(v13) + v15 + sqrt(v16), cbm\_train)  
summary(train\_cbm\_lm)  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05 besides v15.  
# Test MSE still close to 0.  
# R-squared is 93.21%%  
# Try model without square ro0ting v15.  
# p-value still greater than 0.05.  
# Remove v15 predictor  
train\_cbm\_lm = lm(v18 ~ sqrt(v2) + sqrt(v3) + sqrt(v4) + sqrt(v5) + sqrt(v6) +  
 sqrt(v8) + sqrt(v11) + sqrt(v13) + sqrt(v16), cbm\_train)  
summary(train\_cbm\_lm)  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# Test MSE still close to 0.  
# R-squared is 93.21%%  
  
# Try squared:  
train\_cbm\_lm = lm(v18 ~ I(v2)^2 + I(v3)^2 + I(v4)^2 + I(v5)^2 + I(v6)^2 +  
 I(v8)^2 + I(v11)^2 + I(v13)^2 + I(v15)^2 + I(v16)^2, cbm\_train)  
summary(train\_cbm\_lm)  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# Test MSE still close to 0.  
# R-squared is 90.81%%  
  
# Try cubed:  
train\_cbm\_lm = lm(v18 ~ I(v2)^3 + I(v3)^3 + I(v4)^3 + I(v5)^3 + I(v6)^3 +  
 I(v8)^3 + I(v11)^3 + I(v13)^3 + I(v15)^3 + I(v16)^3, cbm\_train)  
summary(train\_cbm\_lm)  
pred\_cbm\_lm = predict(train\_cbm\_lm, newdata = cbm\_test)  
mean((pred\_cbm\_lm - cbm\_test$v18)^2)  
# All p-values less than 0.05.  
# Test MSE still close to 0.  
# R-squared is 90.81%%  
  
# Sqrt() transformation is highest R-squared value so far with test MSE being  
## 0.  
  
# Try interactions of sqrt() model  
# Look at most correlated variables using these predictors:  
cor(cbm\_df)  
best\_train\_cbm\_lm = lm(v18 ~ sqrt(v2)\*sqrt(v5) + sqrt(v4)\*sqrt(v11) + sqrt(v3)\*sqrt(v6) +  
 sqrt(v8)\*sqrt(v13) + sqrt(v16), cbm\_train)  
summary(best\_train\_cbm\_lm)  
best\_pred\_cbm\_lm = predict(best\_train\_cbm\_lm, newdata = cbm\_test)  
mean((best\_pred\_cbm\_lm - cbm\_test$v18)^2)  
# This raised R-squared to 94.82%  
# Tried other variations. This was the best R-squared value that I got.  
# Test MSE = 0.00000295  
  
# Plot this model:  
par(mfrow = c(1, 1))  
plot\_df\_1 = data.frame(Actual = cbm\_test$v18, Predicted = pred\_cbm\_lm)  
  
ggplot(plot\_df, aes(x = Actual, y = Predicted)) +  
 geom\_point() +  
 geom\_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") + # Adds a line of perfect prediction  
 labs(x = "Actual Gas Turbine Decay State Coefficient",   
 y = "Predicted Gast Turbine Decay State Coefficient",   
 title = "Actual vs. Predicted") +  
 theme\_minimal()  
  
# Ridge Regression Model --------------------------------------------------  
library(glmnet)  
# Perform ridge regression on data  
set.seed(1)  
train\_matrix = model.matrix(v18~., cbm\_train)  
test\_matrix = model.matrix(v18~., cbm\_test)  
# Now we need to select lambda using cross-validation  
cv\_out = cv.glmnet(train\_matrix, cbm\_train$v18, alpha = 0)  
best\_lam = cv\_out$lambda.min  
best\_lam  
# Lambda chosen by cross-validation is 0.00004  
# Now we fit ridge regression model and make predictions:  
cbm\_ridge = glmnet(train\_matrix, cbm\_train$v18, alpha = 0)  
pred\_cbm\_ridge = predict(cbm\_ridge, s = best\_lam, newx = test\_matrix)  
# Find test error:  
mean((pred\_cbm\_ridge - cbm\_test$v18)^2)  
# Test Error is very close to 0  
# Test error is 0.000048  
  
  
# Lasso Regression Model --------------------------------------------------  
# Perform lasso regression  
set.seed(1)  
# Select lambda using cross-validation  
cv\_lam = cv.glmnet(train\_matrix, cbm\_train$v18, alpha = 1)  
best\_lam = cv\_lam$lambda.min  
best\_lam  
# Lambda chosen by cross-validation is 0.00000505  
# Now we fit lasso regression model and make predictions:  
cbm\_lasso = glmnet(train\_matrix, cbm\_train$v18, alpha = 1)  
pred\_cbm\_lasso = predict(cbm\_lasso, s = best\_lam, newx = test\_matrix)  
# Find test error:  
mean((pred\_cbm\_lasso - cbm\_test$v18)^2)  
# Test error is very close to 0  
# Test error is 0.0000095.  
# Ridge regression has lower test MSE.  
  
# Principal Components Regression -----------------------------------------  
# Perform principal components regression and try to reduce dimensions of data.  
library(pls)  
set.seed(1)  
cbm\_pcr = pcr(v18~., data = cbm\_train,   
 scale = TRUE, validation = "CV")  
summary(cbm\_pcr)  
validationplot(cbm\_pcr, val.type = "MSEP")  
# Where M = 11, 12, and 13 have lowest MSEP values. But we want to try and reduce  
# dimensions.  
# Let's try where M = 10.  
pred\_pcr = predict(cbm\_pcr, cbm\_test, ncomp = 10)  
mean((pred\_pcr - cbm\_test$v18)^2)  
# Test MSE = 0.0000223  
  
# Try where M = 9  
pred\_pcr = predict(cbm\_pcr, cbm\_test, ncomp = 9)  
mean((pred\_pcr - cbm\_test$v18)^2)  
# Test MSE = 0.0000234  
  
# And try where M = 11  
pred\_pcr = predict(cbm\_pcr, cbm\_test, ncomp = 11)  
mean((pred\_pcr - cbm\_test$v18)^2)  
# Test MSE = 0.00000605  
# Even though we are trying to reduce dimensions. M = 11 provides the lowest  
## test error.  
  
# Partial Least Squares ---------------------------------------------------  
set.seed(1)  
cbm\_pls = plsr(v18~., data = cbm\_train,   
 scale = TRUE, validation = "CV")  
summary(cbm\_pls)  
validationplot(cbm\_pls, val.type = "MSEP")  
# Lowest MSEP occurs when M = 12 but we are trying to reduce dimension  
# Try with M = 8  
pred\_pls = predict(cbm\_pls, cbm\_test, ncomp = 8)  
mean((pred\_pls - cbm\_test$v18)^2)  
# Test MSE = 0.0000144  
  
# Try M = 9  
pred\_pls = predict(cbm\_pls, cbm\_test, ncomp = 9)  
mean((pred\_pls - cbm\_test$v18)^2)  
# Test MSE = 0.0000113  
  
# Try M = 10  
pred\_pls = predict(cbm\_pls, cbm\_test, ncomp = 10)  
mean((pred\_pls - cbm\_test$v18)^2)  
# Test MSE = 0.00000578  
# M = 10 provides lowest test MSE of Partial Least Squares  
# Lower than PCR with M = 11.  
  
# Regression Trees --------------------------------------------------------  
# Fit a regression tree on training data  
library(tree)  
cbm\_tree = tree(v18 ~ ., cbm\_train)  
summary(cbm\_tree)  
plot(cbm\_tree)  
text(cbm\_tree, pretty = 0)  
# Number of terminal nodes: 21  
# Only variables used in tree: v3, v13, v5, v4, v14, v10 and v11  
# Find test MSE  
yhat = predict(cbm\_tree, newdata = cbm\_test)  
mean((yhat - cbm\_test[, "v18"])^2)  
# Test MSE = 0.0000191  
cbm\_tree  
  
# Pruning Regression Trees  
# Use cv.tree() function to prune tree to help make it more interpretable.  
cv\_tree\_cbm = cv.tree(cbm\_tree)  
plot(cv\_tree\_cbm$size, cv\_tree\_cbm$dev, type = "b")  
# Try with 15 terminal nodes instead of 21  
cbm\_prune = prune.tree(cbm\_tree, best = 15)  
plot(cbm\_prune)  
text(cbm\_prune, pretty = 0)  
summary(cbm\_prune)  
# Find test MSE on pruned tree:  
yhat = predict(cbm\_prune, newdata = cbm\_test)  
mean((yhat - cbm\_test[, "v18"])^2)  
# Test MSE = 0.0000231  
# Close to test MSE of original tree and more interpretable.  
# Still not very easy to interpret.  
  
# Try with 11 terminal nodes  
cbm\_prune = prune.tree(cbm\_tree, best = 11)  
plot(cbm\_prune)  
text(cbm\_prune, pretty = 0)  
summary(cbm\_prune)  
# Find test MSE on pruned tree:  
yhat = predict(cbm\_prune, newdata = cbm\_test)  
mean((yhat - cbm\_test[, "v18"])^2)  
# Test MSE = 0.0000284  
# Similar test MSE. More interpretable tree.  
  
# Try with 9 terminal nodes  
cbm\_prune = prune.tree(cbm\_tree, best = 9)  
plot(cbm\_prune)  
text(cbm\_prune, pretty = 0)  
summary(cbm\_prune)  
# Find test MSE on pruned tree:  
yhat = predict(cbm\_prune, newdata = cbm\_test)  
mean((yhat - cbm\_test[, "v18"])^2)  
# Test MSE = 0.0000284  
# Similar test MSE to tree with 11 terminal nodes  
  
# Try with 8 terminal nodes  
cbm\_prune = prune.tree(cbm\_tree, best = 8)  
plot(cbm\_prune)  
text(cbm\_prune, pretty = 0)  
summary(cbm\_prune)  
# Find test MSE on pruned tree:  
yhat = predict(cbm\_prune, newdata = cbm\_test)  
mean((yhat - cbm\_test[, "v18"])^2)  
# Test MSE = 0.0000363  
# Slightly higher test MSE to tree with 9 terminal nodes. More easy to read.  
  
# Pruned tree preferred as it has small test MSE and it is easier to understand compared  
## to tree with 21 terminal nodes.  
  
  
# Bagging and Random Forests ---------------------------------------------------  
# Perform bagging model and Random Forest. Find test MSE  
library(randomForest)  
set.seed(1)  
cbm\_bag = randomForest(v18 ~., data = cbm\_train, mtry = 13, importance = TRUE)  
cbm\_bag  
yhat\_bag = predict(cbm\_bag, newdata = cbm\_test)  
mean((yhat\_bag - cbm\_test$v18)^2)  
# Using 500 trees and trying all 13 predictors at each split, 98.45%  
## of variance explained.   
# Test MSE = 0.000000858  
# Lowest test MSE so far.  
  
# Try with number of trees as 25 to simplify model  
set.seed(1)  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 25,  
 mtry = 13, importance = TRUE)  
cbm\_rf  
yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)  
# Using 25 trees and trying all 13 predictors at each split, 97.96%  
## of variance explained.   
# Test MSE = 0.000000969  
# Still very small MSE.  
  
# Try with 10 variables instead of 13 and 25 trees  
set.seed(1)  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 25,  
 mtry = 10, importance = TRUE)  
cbm\_rf  
yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)  
# Using 25 trees and trying 10 predictors at each split, 97.86%  
## of variance explained.   
# Test MSE = 0.00000103  
# Still very small MSE and variance barely dropped.  
  
# Try with 10 variables instead of 13 and 10 trees  
set.seed(1)  
cbm\_bag = randomForest(v18 ~., data = cbm\_train, ntree = 10,  
 mtry = 10, importance = TRUE)  
cbm\_rf  
yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)  
# Using 10 trees and trying 10 predictors at each split, 96.97%  
## of variance explained.   
# Test MSE = 0.00000112  
# Still very small MSE and variance barely dropped.  
  
# Try with 8 variables and 5 trees  
set.seed(1)  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 5,  
 mtry = 8, importance = TRUE)  
cbm\_rf  
yhat\_rf = predict(cbm\_rf, newdata = cbm\_test)  
mean((yhat\_rf - cbm\_test$v18)^2)  
# Using 5 trees and trying 8 predictors at each split, 95.53%  
## of variance explained.   
# Test MSE = 0.00000148  
# Still very small MSE and variance barely dropped.  
# As you bring M down, and cut down on tree sizes, Test MSE barely affected  
## and variance explained by model does not change drastically.  
  
  
# Boosting ----------------------------------------------------------------  
# Use gbm() library  
library(gbm)  
# lambda = 0.001:  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)  
# Test MSE = 0.0000179  
  
# lamda = 0.01  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.01)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)  
# Test MSE = 0.0000448  
# Lambda = 0.001 produced smaller test MSE  
  
# lambda = 0.1  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.1)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)  
# Test MSE = 0.0000179  
# Lambda = 0.001 produced same test MSE as using lambda = 0.001  
  
# lambda = 0.2  
set.seed(1)  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.2)  
yhat\_boost = predict(cbm\_boost, newdata = cbm\_test,   
 n.trees = 1000)  
mean((yhat\_boost - cbm\_test$v18)^2)  
# Test MSE = 0.00001  
# Lambda = 0.2 produced smallest test MSE of all boosted models.  
  
  
# Bayesian Additive Regression Trees (BART) -------------------------------  
# Reread in data for BART model  
cbm\_df = read.table("Final\_Project/uci\_bm\_dataset/data.txt", header = FALSE, colClasses = "numeric")  
colnames(cbm\_df) = tolower(colnames(cbm\_df))  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v9")]  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v12")]  
cbm\_df = cbm\_df[, -which(names(cbm\_df) == "v7")]  
  
# Training data for BART  
set.seed(1)  
train = sample(1:nrow(cbm\_df), 5967)  
  
# Perform BART  
library(BART)  
  
x1 = cbm\_df[, 1:13]  
y1 = cbm\_df[,"v18"]  
  
xtrain1 = x1[train,]  
ytrain1 = y1[train]  
  
xtest1 = x1[-train,]  
ytest1 = y1[-train]  
set.seed(1)  
cbm\_bart = gbart(xtrain1, ytrain1, x.test = xtest1)  
# Now compute test MSE  
yhat\_bart = cbm\_bart$yhat.test.mean   
mean((ytest1 - yhat\_bart)^2)  
# Test MSE is 0.00000414  
  
# Test MSE by Model -------------------------------------------------------  
# Liner Model:  
cor(cbm\_df)  
best\_cbm\_lm = lm(v18 ~ sqrt(v2)\*sqrt(v5) + sqrt(v4)\*sqrt(v11) + sqrt(v3)\*sqrt(v6) +  
 sqrt(v8)\*sqrt(v13) + sqrt(v16), cbm\_train)  
summary(best\_cbm\_lm)  
best\_pred\_cbm\_lm = predict(best\_train\_cbm\_lm, newdata = cbm\_test)  
mean((best\_pred\_cbm\_lm - cbm\_test$v18)^2)  
# This raised R-squared to 94.82%  
# Tried other variations. This was the best R-squared value that I got.  
# Test MSE = 0.00000295  
  
# Ridge Regression Model  
# Test MSE = 0.000048  
  
# Lasso Regression Model  
# Test MSE = 0.0000095  
  
# Principal Component Regression Model  
pred\_pcr = predict(cbm\_pcr, cbm\_test, ncomp = 11)  
mean((pred\_pcr - cbm\_test$v18)^2)  
# Test MSE = 0.00000605  
  
# Partial Least Squares Model  
pred\_pls = predict(cbm\_pls, cbm\_test, ncomp = 10)  
mean((pred\_pls - cbm\_test$v18)^2)  
# Test MSE = 0.00000578  
  
# Regression Tree  
cbm\_tree = tree(v18 ~ ., cbm\_train)  
# Test MSE = 0.0000191  
  
# Bagging  
cbm\_bag = randomForest(v18 ~., data = cbm\_train, mtry = 13, importance = TRUE)  
# Using 500 trees and trying all 13 predictors at each split, 98.45% of variance explained.   
# Test MSE = 0.000000858  
  
# Random Forest  
cbm\_rf = randomForest(v18 ~., data = cbm\_train, ntree = 25,  
 mtry = 13, importance = TRUE)  
# Test MSE = 0.000000969.  
  
# Boosting  
cbm\_boost = gbm(v18 ~., data = cbm\_train, distribution = "gaussian",  
 n.trees = 1000, shrinkage = 0.2)  
# Test MSE = 0.00001  
  
  
# BART Model  
cbm\_bart = gbart(xtrain1, ytrain1, x.test = xtest1)  
# Test MSE = 0.00000414  
  
# Best Model:  
# Bagging using all 13 predictors at each split provided best Test MSE and explained  
## 98.45% of the variance in the data set.   
# On that conclusion, we will say that this model best fits our training data to   
## predict our test data.

Code for Qualitative Analysis:

# Qualitative Regression Final Project R code ----------------------------------  
  
# Data link: https://archive.ics.uci.edu/dataset/186/wine+quality  
  
# Citation:  
# Cortez,Paulo, Cerdeira,A., Almeida,F., Matos,T., and Reis,J.. (2009). Wine Quality.   
# UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.  
  
# Use wine data to predict wine quality as bad and good.  
# Bad wine quality will be ranked from 3-5.  
# Good wine quality will be ranked from 6-9.  
  
# Read in and clean up data -----------------------------------------------  
red\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_red.csv")  
white\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_white.csv")  
wine\_df = rbind(red\_wine\_df, white\_wine\_df)  
# Change column names to snake case:  
colnames(wine\_df) = gsub("\\.", "\_", colnames(wine\_df))  
lowercase\_cols = tolower(colnames(wine\_df))  
colnames(wine\_df) = lowercase\_cols  
dim(wine\_df)  
# 6497 observations and 12 variables.  
  
# Response Variable Summary -----------------------------------------------  
# quality will be our response variable.  
# Can we use qualitative regression to find model to accurately predict quality response variable  
## using 11 other predictors? Let's find out.  
# First let's categorize the quality response variable into bad or good.  
# First range of quality:  
range(wine\_df$quality)  
# 3 to 9.  
# Use for loop to categorize wine quality 3-5 as poor and 6-9 as good:  
# 0 will equal bad.   
# 1 will equal good.  
for (i in 1:length(wine\_df$quality)) {  
 if (wine\_df$quality[i] >= 3 & wine\_df$quality[i] <= 5) {  
 wine\_df$quality[i] = 0  
 } else if (wine\_df$quality[i] >= 6 & wine\_df$quality[i] <= 9) {  
 wine\_df$quality[i] = 1  
 } else {  
 wine\_df$quality[i] = NA # Handle values outside the specified ranges  
 }  
}  
# Count how many poors:  
sum(wine\_df$quality == 0)  
# 2384  
# Count how many goods:  
sum(wine\_df$quality == 1)  
# 4113  
  
#  
  
# Training and Test Data --------------------------------------------------  
# Here is training and test data that we will use. There is 6497 observations.  
# Split data in half for training and test data.  
set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
wine\_train = wine\_df[train,]  
wine\_test = wine\_df[-train,]  
  
# Check correlated variables  
cor(wine\_df[-12])  
  
# Logistic Regression -----------------------------------------------------  
glm\_wine = glm(quality~., wine\_df, family = binomial)  
summary(glm\_wine)  
# Using null hypothesis where no predictor is significant, we see that that we can  
## reject the null hypothesis for the following predictors as their p-values are all  
### less than 0.05:  
# volatile\_acidity  
# citric\_acid  
# residual\_sugar  
# free\_sulfur\_dioxide  
# total\_sulfur\_dioxide  
# ph  
# sulphates  
# alcohol  
  
# Three of the predictors have p-values greater than 0.05, so we fail to reject  
## the null hypothesis on those predictors.   
# With that being said, we remove them:  
# fixed\_acidity  
# chlorides  
# density  
glm\_wine = glm(quality ~. - fixed\_acidity - chlorides - density,   
 wine\_df, family = binomial)  
summary(glm\_wine)  
# After creating this model, we fall to reject the null hypothesis for ph  
## variable. Remove that variable from the model:  
glm\_wine = glm(quality ~. - fixed\_acidity - chlorides - density - ph,   
 wine\_df, family = binomial)  
summary(glm\_wine)  
# All variables p-values are less than 0.  
  
# Predict on the entire dataset using the logistic model  
pred\_probs = predict(glm\_wine, newdata = wine\_df, type = "response")  
  
# Convert probabilities to binary predictions (0 or 1)  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
  
# Construct confusion matrix:  
conf\_matrix = table(wine\_df$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (1357 + 3464)/6497  
# Returns 0.7420  
# Results in prediction error of:  
1 - accurate\_pred  
# 0.2579 or 25.8%  
  
# This data is definitely overfitted using just entire data set.  
# Let's use training and test data. Use training data to create model then  
## test data to test the model:  
glm\_wine = glm(quality ~., wine\_train, family = binomial)  
summary(glm\_wine)  
# Looking at p-values, we will leave out following variables:  
# chlorides  
glm\_wine = glm(quality~. - chlorides, wine\_train, family = binomial)  
summary(glm\_wine)  
# Remove predictors that are barely significant to see if that changes   
## prediction rate to simplify model:  
# citric\_acid  
# density  
# ph  
glm\_wine = glm(quality~. - chlorides - citric\_acid -   
 density - ph, wine\_train, family = binomial)  
summary(glm\_wine)  
# fixed\_acidity no longer significant  
glm\_wine = glm(quality~. - chlorides - citric\_acid -   
 density - ph - fixed\_acidity, wine\_train, family = binomial)  
# Same as:  
glm\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates +  
 alcohol, wine\_train, family = binomial)  
summary(glm\_wine)  
  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (671 + 1733) / 3249  
# Returns 0.73992  
1 - accurate\_pred  
# Prediction error of 0.26 or 26%. Higher than prediction error rate on  
## over fitted data, but this one is less bias as we used training data and  
# tested it on test data.  
  
  
# Transformations and Interactions ----------------------------------------  
# Interaction terms:  
glm\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates\*alcohol,   
 wine\_train, family = binomial)  
summary(glm\_wine)  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (670 + 1733) / 3249  
# This produced similar prediction rate of 74%  
  
# After multiple tries at different interactions between different variables,  
## I could not produce a significantly better test prediction percentage.  
  
# Try sqrt:  
glm\_wine = glm(quality ~ sqrt(volatile\_acidity) + sqrt(residual\_sugar) +  
 sqrt(free\_sulfur\_dioxide) + sqrt(total\_sulfur\_dioxide) +   
 sqrt(sulphates) + sqrt(alcohol), wine\_train, family = binomial)  
summary(glm\_wine)  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (691 + 1733) / 3249  
# 74.6%. Highest prediction rate yet.  
  
# Try squared:  
glm\_wine = glm(quality ~ I(volatile\_acidity)^2 + I(residual\_sugar)^2 +  
 I(free\_sulfur\_dioxide)^2 + I(total\_sulfur\_dioxide)^2 +   
 I(sulphates)^2 + I(alcohol)^2, wine\_train, family = binomial)  
summary(glm\_wine)  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (671 + 1733) / 3249  
# 73.99%.  
  
# Try cubed:  
glm\_wine = glm(quality ~ I(volatile\_acidity)^3 + I(residual\_sugar)^3 +  
 I(free\_sulfur\_dioxide)^3 + I(total\_sulfur\_dioxide)^3 +   
 I(sulphates)^3 + I(alcohol)^3, wine\_train, family = binomial)  
summary(glm\_wine)  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (671 + 1733) / 3249  
# Sames as cubed. 73.99%  
  
# Try log transformation:  
glm\_wine = glm(quality ~ log(volatile\_acidity) + log(residual\_sugar) +  
 log(free\_sulfur\_dioxide) + log(total\_sulfur\_dioxide) +   
 log(sulphates) + log(alcohol), wine\_train, family = binomial)  
summary(glm\_wine)  
  
pred\_probs = predict(glm\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (710 + 1725) / 3249  
# 74.96%.   
  
# Linear Discrimination Analysis ------------------------------------------  
# Use lda() function on training set and predict test data.  
library(MASS)  
lda\_wine = lda(quality ~., wine\_train)  
summary(lda\_wine)  
lda\_pred = predict(lda\_wine, wine\_test)  
lda\_class = lda\_pred$class  
table(lda\_class, wine\_test$quality)  
accurate\_pred = mean(lda\_class == wine\_test$quality)  
# Prediction rate = 0.7371 or 73.56%  
# Prediction error rate of:  
1 - accurate\_pred  
# 0.2629 or 26.29%  
# Same prediction error rate from logistic regression.  
  
# Using QDA ---------------------------------------------------------------  
qda\_wine = qda(quality ~., wine\_train)  
qda\_wine  
  
qda\_class = predict(qda\_wine, wine\_test)$class  
table(qda\_class, wine\_test$quality)  
accurate\_pred = mean(qda\_class == wine\_test$quality)  
# Prediction rate of 0.7291 or 72.91%  
# Prediction error rate of:  
1 - accurate\_pred  
# 0.2709 or 27.09%  
  
# Using KNN ---------------------------------------------------------------  
# K = 1  
library(class)  
# Predictor variables used in glm\_wine  
predictor\_vars = c('volatile\_acidity', 'residual\_sugar',  
 'free\_sulfur\_dioxide', 'total\_sulfur\_dioxide',  
 'sulphates', 'alcohol')  
  
# Extract predictor variables from wine\_train  
train\_data = wine\_train[, predictor\_vars]  
  
# Define the test data  
test\_data = wine\_test[, predictor\_vars]  
  
# Perform KNN prediction  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 1)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (692 + 1581) / 3249  
# Prediction rate is 0.6995  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3004 or 34.04%.  
# Worst prediction error rate so far.  
  
# Try K = 2  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 2)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (603 + 1491) / 3249  
# Prediction rate is 0.6445  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3555 or 35.55%.  
  
# Try K = 3  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 3)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (555 + 1560) / 3249  
# Prediction rate is 0.651  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.349 or 34.9%.  
  
# Try K = 4  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 4)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (548 + 1559) / 3249  
# Prediction rate is 0.6485  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3515 or 35.15%.  
# Not much better than K = 1  
  
# Try K = 5  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 5)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (557 + 1606) / 3249  
# Prediction rate is 0.6657  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3343 or 33.43%.  
# Not much better than K = 1  
  
# Try K = 6  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 6)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (531 + 1600) / 3249  
# Prediction rate is 0.6559  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3441 or 34.41%.  
# Not much better than K = 1  
  
# Try K = 7  
set.seed(1)  
knn\_wine = knn(train\_data, test\_data, wine\_train$quality, k = 7)  
# Use confusion matrix to test predictions  
conf\_matrix = table(wine\_test$quality, knn\_wine)  
print(conf\_matrix)  
accurate\_pred = (504 + 1687) / 3249  
# Prediction rate is 0.6743  
# Prediction error rate is:  
1 - accurate\_pred  
# 0.3256 or 32.56%.  
# Not much better than K = 1 or K = 5  
  
# KNN has worst prediction rate of all models that we ran.  
  
# Naive Bayes -----------------------------------------------------  
library(e1071)  
set.seed(1)  
nb\_wine = naiveBayes(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates +  
 alcohol, wine\_train)  
nb\_wine  
nb\_class = predict(nb\_wine, wine\_test)  
table(nb\_class, wine\_test$quality)  
accurate\_pred = mean(nb\_class == wine\_test$quality)  
# Prediction accuracy is 70.33%  
# Prediction error rate is:  
1 - accurate\_pred  
# 29.67%  
# Low. but not as low as logistic regression, QDA or LDA.  
  
# Poisson Regression ------------------------------------------------------  
poiss\_wine = glm(quality ~ volatile\_acidity + residual\_sugar +  
 free\_sulfur\_dioxide + total\_sulfur\_dioxide + sulphates +  
 alcohol, wine\_train, family = poisson)  
summary(poiss\_wine)  
# We can reject null hypothesis for all predictors.  
pred\_probs = predict(poiss\_wine, newdata = wine\_test, type = "response")  
pred\_classes = ifelse(pred\_probs > 0.5, 1, 0)  
conf\_matrix = table(wine\_test$quality, pred\_classes)  
print(conf\_matrix)  
accurate\_pred = (698 + 1699) / (3249)  
# Accurate prediction percentage is 73.78%  
# Prediction error rate is:  
1 - accurate\_pred  
# 26.22%.  
# Similar to prediction error rates for logistic regression, LDA and QDA.  
  
# Classification Tree -----------------------------------------------------  
red\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_red.csv")  
white\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_white.csv")  
wine\_df = rbind(red\_wine\_df, white\_wine\_df)  
# Change column names to snake case:  
colnames(wine\_df) = gsub("\\.", "\_", colnames(wine\_df))  
lowercase\_cols = tolower(colnames(wine\_df))  
colnames(wine\_df) = lowercase\_cols  
  
library(tree)  
attach(wine\_df)  
good = factor(ifelse(quality <= 5, "No", "Yes"))  
wine\_df = data.frame(wine\_df, good)  
# New training and test data:  
set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
wine\_train = wine\_df[train,]  
wine\_test = wine\_df[-train,]  
  
tree\_wine = tree(good ~. - quality, wine\_df)  
summary(tree\_wine)  
# Terminal nodes: 5  
# Training error rate = 26.43%.  
  
# Plot tree  
plot(tree\_wine)  
text(tree\_wine, pretty = 0)  
# Most important indicator of quality is alcohol.  
# Shows up three times on tree.  
tree\_wine  
  
# To properly evaluate performance of classification tree, we must estimate test   
## error:  
tree\_wine = tree(good ~ . - quality, wine\_train)  
pred\_tree = predict(tree\_wine, wine\_test, type = "class")  
table(pred\_tree, wine\_test$good)  
accurate\_pred = (794 + 1565) / (3249)  
# Accurate prediction rate is 72.61%  
# Prediction error rate is:  
1 - accurate\_pred  
# 27.39%.  
  
# Pruning tree:  
# Now, let's prune the tree to see if that leads to improved results.  
set.seed(1)  
cv\_wine = cv.tree(tree\_wine, FUN = prune.misclass)  
names(cv\_wine)  
cv\_wine  
# dev corresponds to number of cv errors. Same cv errors at five and 3 node tree  
par(mfrow = c(1,2))  
plot(cv\_wine$size, cv\_wine$dev, type = "b")  
plot(cv\_wine$k, cv\_wine$dev, type = "b")  
# Apply prune.misslcass function in order to purne tree to obtain 3 node tree  
prune\_wine = prune.misclass(tree\_wine, best = 3)  
plot(prune\_wine)  
text(prune\_wine, pretty = 0)  
# Test pruned tree performance on test data set.  
pred\_tree = predict(prune\_wine, wine\_test, type = "class")  
table(pred\_tree, wine\_test$good)  
accurate\_pred = (794 + 1565) / (3249)  
# Same prediction rate an prediction error rate as 5 node tree.  
# Go with 3 node tree because it is more interpretable than 5 node tree.  
  
# Bagging Approach:  
library(randomForest)  
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11,  
 importance = TRUE)  
bag\_wine  
# 500 trees considering all 11 predictors   
# Out-of-bag estimate of error rate is: 19.98  
# Accuracy rate is:  
1 - 0.1998  
# 80 %. Highest prediction rate so far. But this is only on training set.  
# Try on test set  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)  
accurate\_pred = (824 + 1788) / (3249)  
# 80.39% accurate prediction rate.  
# Highest so far!  
  
# Try with number of trees as 25 instead of 500 to simplify model:  
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11, ntree = 25,  
 importance = TRUE)  
bag\_wine  
# Slightly larger OOB estimate of error rate:  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)  
accurate\_pred = (808 + 1774) / (3249)  
# 79.47%. Not as high as bagging model with 500 Trees  
  
# Try 100 trees  
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11, ntree = 100,  
 importance = TRUE)  
bag\_wine  
# Slightly larger OOB estimate of error rate:  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)  
accurate\_pred = (826 + 1783) / (3249)  
# 80.30%   
# About same as 500 trees, and much more simpler model with only 100 trees.  
  
# Try with 50 trees  
set.seed(1)  
bag\_wine = randomForest(good ~. - quality, wine\_train, mtry = 11, ntree = 50,  
 importance = TRUE)  
bag\_wine  
# Slightly larger OOB estimate of error rate:  
pred\_bag = predict(bag\_wine, wine\_test, type = "class")  
table(pred\_bag, wine\_test$good)  
accurate\_pred = (822 + 1761) / (3249)  
# 79.50%.  
# Slightly lower than model with 100 trees.  
# So far 100 tree bagging model has highest prediction rate.  
  
# Random Forests:  
# Same as bagging but we use smaller mtry argument  
# By default, random forest uses sqrt(p) when building a random forest of classification  
## trees.  
# Let's try mtry = 5  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 19.83%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (823 + 1792) / (3249)  
# Prediction rate: 80.49%  
# Slightly higher than bagging model with 100 trees  
  
# Try random forest with 100 trees and mtry = 5.  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5, ntree = 100,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 20.26%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (837 + 1796) / (3249)  
# 81.04%  
# Highest prediction rate so far!  
  
# Try same model as above with mtry = 4  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 4, ntree = 100,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 20.14%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (827 + 1790) / (3249)  
# 80.55%.  
# Not as high when mtry = 5.  
  
# Try when mtry = 6  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 6, ntree = 100,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 19.92%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (837 + 1778) / (3249)  
# 80.49%  
  
# Random forest model with m try = 5 and ntree = 100 has best prediction rate  
## on test data: 81.04%  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5, ntree = 100,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 20.26%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (837 + 1796) / (3249)  
# importance() function shows importance of each variable  
importance(rf\_wine)  
# Plot these importance measures:  
varImpPlot(rf\_wine)  
# alcohol and volatile\_acidity seem to be most important predictors in this model.  
  
# Boosting:   
# Call gbm library for boosting.  
# Reset data back to binary classification for quality variable:  
red\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_red.csv")  
white\_wine\_df = read.csv("Final\_Project/wine\_quality/wine\_quality\_white.csv")  
wine\_df = rbind(red\_wine\_df, white\_wine\_df)  
# Change column names to snake case:  
colnames(wine\_df) = gsub("\\.", "\_", colnames(wine\_df))  
lowercase\_cols = tolower(colnames(wine\_df))  
colnames(wine\_df) = lowercase\_cols  
  
for (i in 1:length(wine\_df$quality)) {  
 if (wine\_df$quality[i] >= 3 & wine\_df$quality[i] <= 5) {  
 wine\_df$quality[i] = 0  
 } else if (wine\_df$quality[i] >= 6 & wine\_df$quality[i] <= 9) {  
 wine\_df$quality[i] = 1  
 } else {  
 wine\_df$quality[i] = NA # Handle values outside the specified ranges  
 }  
}  
# Re-establish test and training data:  
set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
wine\_train = wine\_df[train,]  
wine\_test = wine\_df[-train,]  
  
# Default lambda value in model is 0.001  
library(gbm)  
set.seed(1)  
boost\_wine = gbm(quality ~., wine\_train, distribution = "bernoulli",  
 n.trees = 5000, interaction.depth = 4)  
summary(boost\_wine)  
# We see that alcohol, total\_sulfur dioxide, and volatile\_acidity  
# are most important variables in boosted model.  
plot(boost\_wine, i = "alcohol")  
plot(boost\_wine, i = "total\_sulfur\_dioxide")  
plot(boost\_wine, i = "volatile\_acidity")  
  
# Use model to predict test data  
pred\_boost = predict(boost\_wine, wine\_test, n.trees = 5000, type = "response")  
pred\_classes = ifelse(pred\_boost > 0.5, 1, 0)  
table(pred\_classes, wine\_test$quality)  
accurate\_pred = (828 + 1733) / (3249)  
acurate\_pred  
# 78.82% prediction rate  
  
# Try with n.trees = 100. Same shinking parameter.  
set.seed(1)  
boost\_wine = gbm(quality ~., wine\_train, distribution = "bernoulli",  
 n.trees = 100, interaction.depth = 4)  
# Use model to predict test data  
pred\_boost = predict(boost\_wine, wine\_test, n.trees = 100, type = "response")  
pred\_classes = ifelse(pred\_boost > 0.5, 1, 0)  
table(pred\_classes, wine\_test$quality)  
accurate\_pred = (752 + 1745) / (3249)  
# 76.85%. Smaller than previous model.  
  
# Try with 5000 trees and shrinking parameter of 0.01  
set.seed(1)  
boost\_wine = gbm(quality ~., wine\_train, distribution = "bernoulli",  
 n.trees = 5000, interaction.depth = 4, shrinkage = 0.01,  
 verbose = FALSE)  
# Use model to predict test data  
pred\_boost = predict(boost\_wine, wine\_test, n.trees = 5000, type = "response")  
pred\_classes = ifelse(pred\_boost > 0.5, 1, 0)  
table(pred\_classes, wine\_test$quality)  
accurate\_pred = (791 + 1761) / (3249)  
# 78.54%  
  
# Shrinking parameter of 0.1  
set.seed(1)  
boost\_wine = gbm(quality ~., wine\_train, distribution = "bernoulli",  
 n.trees = 5000, interaction.depth = 4, shrinkage = 0.1,  
 verbose = FALSE)  
# Use model to predict test data  
pred\_boost = predict(boost\_wine, wine\_test, n.trees = 5000, type = "response")  
pred\_classes = ifelse(pred\_boost > 0.5, 1, 0)  
table(pred\_classes, wine\_test$quality)  
accurate\_pred = (828 + 1733) / (3249)  
# 78.82  
  
# Shrinking parameter of 0.2  
set.seed(1)  
boost\_wine = gbm(quality ~., wine\_train, distribution = "bernoulli",  
 n.trees = 5000, interaction.depth = 4, shrinkage = 0.2,  
 verbose = FALSE)  
# Use model to predict test data  
pred\_boost = predict(boost\_wine, wine\_test, n.trees = 5000, type = "response")  
pred\_classes = ifelse(pred\_boost > 0.5, 1, 0)  
table(pred\_classes, wine\_test$quality)  
accurate\_pred = (812 + 1732) / (3249)  
# 78.30  
  
# Boosting model with 5000 trees and shrinking parameter of 0.1 produced  
## best prediction rate.  
# Still not best prediction rate we have seen.  
  
# BART  
# Use BART library  
library(BART)  
# Create matrices of predictors  
xtrain = wine\_df[wine\_train,]  
ytrain = wine\_train[, "quality"]  
  
xtest = wine\_test[, 1:11]  
ytest = wine\_test[, "quality"]  
  
# Create training model and test using test data  
set.seed(1)  
train = sample(1:nrow(wine\_df), 3248)  
x = wine\_df[, 1:11]  
y = wine\_df[, "quality"]  
  
xtrain = x[train,]  
ytrain = y[train]  
  
xtest = x[-train,]  
ytest = y[-train]  
  
bart\_wine = lbart(xtrain, ytrain, x.test = xtest)  
  
# Train and test model on test data:  
bart\_wine = lbart(xtrain, ytrain)  
pred\_bart = predict(bart\_wine, newdata = xtest)  
# Use $prob.test.mean extraction that represents the mean probs across  
## iterations or trees for each observation in test set  
pred\_classes = ifelse(pred\_bart$prob.test.mean > 0.5, 1, 0)  
table(pred\_classes, ytest)  
accurate\_pred = (754 + 1732) / 3249  
accurate\_pred  
# Prediction rate of 76.52%  
  
# Model with Best Prediction Rate -----------------------------------------  
# Random forest model with 100 trees and mtry = 5.  
# 81.04% predictin rate on test data.  
set.seed(1)  
rf\_wine = randomForest(good ~. - quality, wine\_train, mtry = 5, ntree = 100,  
 importance = TRUE)  
rf\_wine  
# OOB Estimate of error rate: 20.26%  
pred\_rf = predict(rf\_wine, wine\_test, type = "class")  
table(pred\_rf, wine\_test$good)  
accurate\_pred = (837 + 1796) / (3249)