Wine Quality

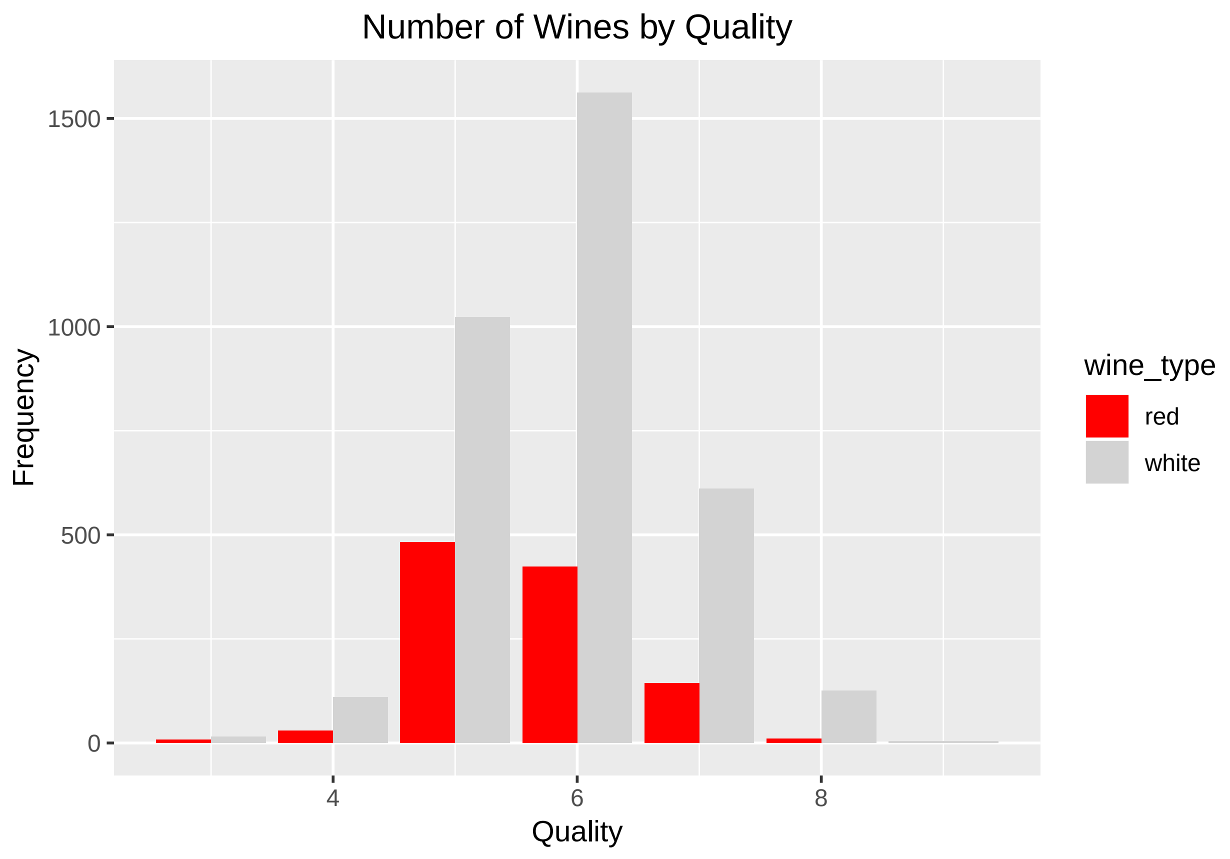
Fahim Hussain
Justin Deringer
Feanna Bergman

09 December 2018

Abstract

We were assigned the Wine Quality Data Set from UCI Machine Learning Repository. It has 4898 observations, 12 predictors, and 1 response variable. The variable of interest is the quality of the wines. The datasets are related to red and white variants of the Portuguese "Vinho Verde" wine, from the north of Portugal. The goal is to model wine quality based on physicochemical tests. There were no information provided on the meaning of the columns, but it is self explanatory by the given naming convention. Some of the considerations we had to make for the analysis was, 1) Given all the predictors, what are the chances of the wine being red or white? 2)Which predictor has the largest effect on wine quality? 3) Are all the predictors useful? We try to have these questions on the back of our mind while performing our analysis.

```r
theme_update(plot.title = element_text(hjust = 0.5))
ggplot(data=wine,aes(quality, fill=wine_type))+
  geom_bar(position = "dodge")+
  scale_fill_manual(values=c("red","lightgray"))+
  xlab("Quality")+ylab("Frequency")+ggtitle("Number of Wines by Quality")
```



```r
table(wine$quality, wine$wine_type)
```

```
##
##      red white
##   3    8    15
##   4   30   110
##   5  482  1023
##   6  423  1562
##   7  143   611
##   8   10   126
##   9    0     4
```

We can see that most of the quality is around the center, with 6 being the most frequent quality for white wines and 5 being the most frequent quality for red wines. There are also far more white wine data than there is red wine. The distribution seems pretty normal and symmetric.

```r
wine <- wine[complete.cases(wine),]
set.seed(13)
sample = sample.split(wine, SplitRatio = .80)
train = subset(wine, sample == TRUE)
test  = subset(wine, sample == FALSE)
```

I created a training set and a test set. The purpose of the training set is to create some models and then figure out the best model to use on my test set. There were some missing data from the original data set, so I removed those rows that contained missing data. There were not many, not enough to have a significant impact. I also decided to use an 80/20 split, my resampling techniques will help justify the split.

Logistic Regression with CV

```r
train_control <- trainControl(method="repeatedcv", number=10, repeats=5)
logmodel <- train(wine_type~.-quality, data=train,
                  trControl=train_control, method="glm",
                  family="binomial")
pred = predict(logmodel, newdata = test)
confusionMatrix(data=pred, test$wine_type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   237     5
##      white   3   803
##
##                Accuracy : 0.9924
##                  95% CI : (0.985, 0.9967)
##     No Information Rate : 0.771
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9784
##  Mcnemar's Test P-Value : 0.7237
##
##             Sensitivity : 0.9875
##             Specificity : 0.9938
##          Pos Pred Value : 0.9793
##          Neg Pred Value : 0.9963
##              Prevalence : 0.2290
##          Detection Rate : 0.2261
##    Detection Prevalence : 0.2309
##       Balanced Accuracy : 0.9907
##
##        'Positive' Class : red
##
```

I decided to use 10 fold repeated cross validation for my resampling technique.This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k–1 folds.(ISLR, 181). I also ignored the quality predictor, because I believe it is more aligned to a response variable and has been catered to as one, according to the source of the dataset. Although the goal of the project is to focus on the quality response, I believe having an analysis on wine type can help with our report in the long run. As we can see, we predicted 237 correct red wines and 803 correct white wines. Our sensitivity and specificity is 98.75% and 99.38% respectively. The accuracy of my model is 99.24% with a 95% confidence interval between 98.5% and 99.67%. These are great percentages and allows me to have the utmost confidence that my model will perform well with any similar dataset given to me to test.

KNN Classification

```
#creating dummy variables for wine color
set.seed (13)
wine_type_dummy <- rep(0, length(wine[,1]))
for (i in 1:length(wine[,1])) {
  if (wine[,1][i]=="red"){
    wine_type_dummy[i] <- 1
  }
}

#scale of each input may be different so we normalize them
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

wine_n <- as.data.frame(lapply(wine[2:12], normalize)) #normalizes all numerical inputs
wine_n <- cbind(wine_type_dummy,wine_n[1:11], wine[13]) #reincludes wine color and quality output
```

Finding out optimal k

```
k_vector <- rep(0,91)
for (i in 10:100) {
  wine_test_pred <- knn(train_n[1:12], test_n[1:12], train_n[,13], k=i)
  #uncomment the following if you want to see all k values and error ratios
  #print(paste("k:",i, " ",mean(wine_test_pred != test_n[,13])),quote=F)

  k_vector[i-9] <- mean(wine_test_pred != test_n[,13])
}

print(paste("Error Ratio:",(min(k_vector))),quote=F)
```

```
## [1] Error Ratio: 0.443702290076336
```

```
print(paste("k:",match(min(k_vector),k_vector)+9),quote=F) #finds index of the error ratio of optimal k
```

```
## [1] k: 90
```

It appears that KNN performs only slightly better than LDA

```
test_set<- read.csv("test_data.csv")
wine_test_dummy <- rep(0, length(test_set[,1]))
for (i in 1:length(test_set[,1])) {
  if (test_set[,1][i]=="red"){
    wine_test_dummy[i] <- 1
  }
}
test_set <- cbind(wine_test_dummy,test_set[2:12])
predictions <- knn(wine_n[1:12], test_set[1:12], wine_n[,13], k=89) #using k found previously
summary(predictions)
```

```
##    3    4    5    6    7    8    9
##    0    0  241 1693   16    0    0
```

Let's see if we can predict wine color based on other inputs

```
k_vector_color <- rep(0,91)
for (i in 10:100) {
  wine_color_pred <- knn(train_n[2:12], test_n[2:12], train_n[,1], k=i)

  #uncomment the following if you want to see all k values and error ratios
  #print(paste("k:",i, " ",mean(wine_color_pred != test_n[,1])),quote=F)

  k_vector_color[i-9] <- mean(wine_color_pred != test_n[,1])
}

print(paste("Error Ratio:",(min(k_vector_color))),quote=F)
```

```
## [1] Error Ratio: 0.00667938931297710
```

```
print(paste("k:",match(min(k_vector_color),k_vector_color)+9),quote=F) #finds index of the error ratio
```

```
## [1] k: 64
```

Wow, over 99% accurate at calculating wine color!

LDA

Measuring covariance of each numerical category compared to quality. Using it to make a plot to visualize data

```
for (i in 2:12){
  print(paste(i,":",cov(wine[i],wine$quality)), quote=F)
}
```

```
## [1] 2 : -0.0870880131587102
## [1] 3 : -0.0389631225886673
## [1] 4 : 0.0105893984644943
## [1] 5 : -0.161101112311935
## [1] 6 : -0.00592627524636266
## [1] 7 : 0.822832608965122
## [1] 8 : -2.24955082876245
## [1] 9 : -0.000783677544294923
## [1] 10 : 0.00322284006569671
## [1] 11 : 0.00461150331582454
## [1] 12 : 0.446263556563208
```

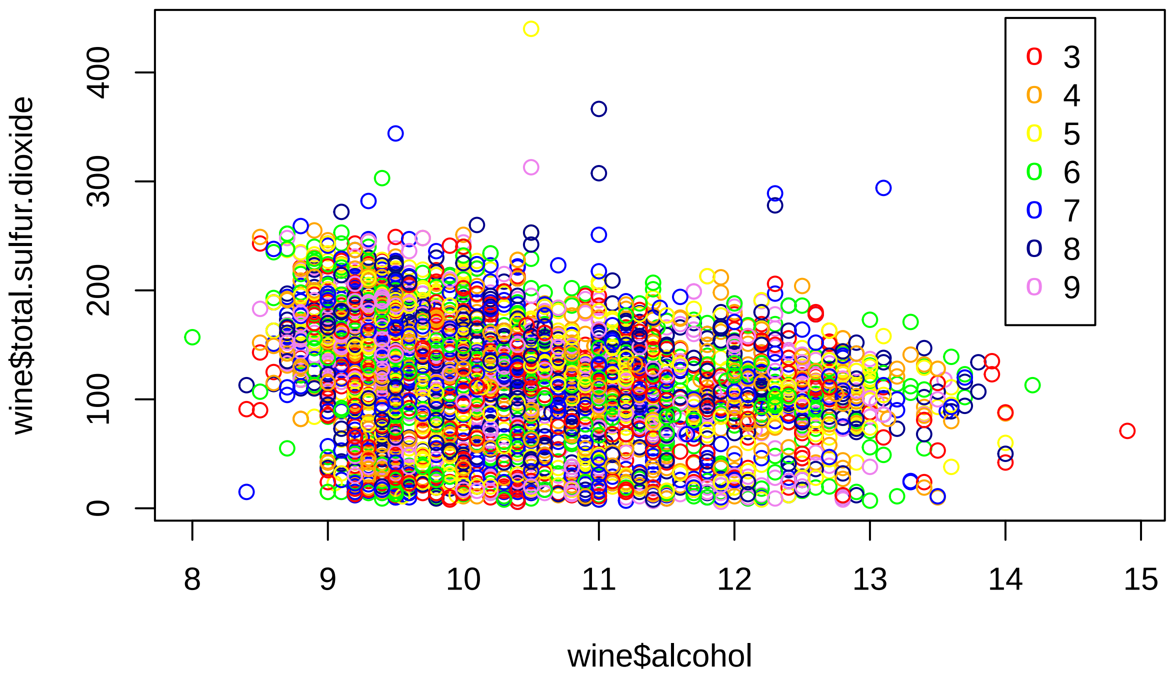
7: free sulfur dioxide

8: total sulfur dioxide

12: alcohol

I selected 8 and 12 becuase 7 and 8 were too similar

```
plot(wine$alcohol,wine$total.sulfur.dioxide,
     col=c("red","orange","yellow", "green", "blue", "dark blue", "violet"))
legend(14,450,legend=c(min(wine$quality):max(wine$quality)),
       col=c("red","orange","yellow", "green", "blue", "dark blue", "violet"),pch="o")
```



Even though the inputs with highest covariance were used in plotting, there is no clear apparent trend. The article "Wine-tasting: it's junk science" says that wine ratings, even ones given by experts, are seemingly given arbitrarily. Experts could not taste the difference between a red wine and a white wine dyed red. They also gave the same wine vastly different scores when tasted at different occasions

Taking these into account, I predict that it will be difficult to determine wine quality from the given inputs.

The article:

https://www.theguardian.com/lifeandstyle/2013/jun/23/wine-tasting-junk-science-analysis

Using LDA with the 2 inputs with highest covariance

```
lda.fit = lda(train[,13] ~ alcohol+total.sulfur.dioxide, data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.480916
```

Results are not good enough, need more inputs

Using LDA with all inputs

```
lda.fit = lda(train[,13] ~ .,  data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.4570611
```

```
lda.fit = lda(train[,13] ~ .-wine_type  ,  data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.4570611
```

```
lda.fit = lda(train[,13] ~ .-wine_type  -residual.sugar,  data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.4475191
```

```
lda.fit = lda(train[,13] ~ .-wine_type  -residual.sugar -pH -sulphates,  data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.4541985
```

```
lda.fit = lda(train[,13] ~ .-wine_type  -residual.sugar -pH -sulphates -chlorides,  data = train[1:12])
lda.pred = predict(lda.fit, test[1:12])
mean(lda.pred$class != test[,13])
```

```
## [1] 0.4541985
```

All the inputs with lowest covariance with quality were removed. I could not remove anything else that wouldn't increase the error ratio. The accuracy improved but it is still not great. Perhaps LDA is not the best way of prediction

Let's see what we get anyway

```
test_set<- read.csv("test_data.csv")
lda.fit = lda(wine[,13] ~ .-wine_type  -residual.sugar -pH -sulphates -chlorides,  data = wine[1:12])
lda.pred = predict(lda.fit, test_set[1:12])
summary(lda.pred$class)
```

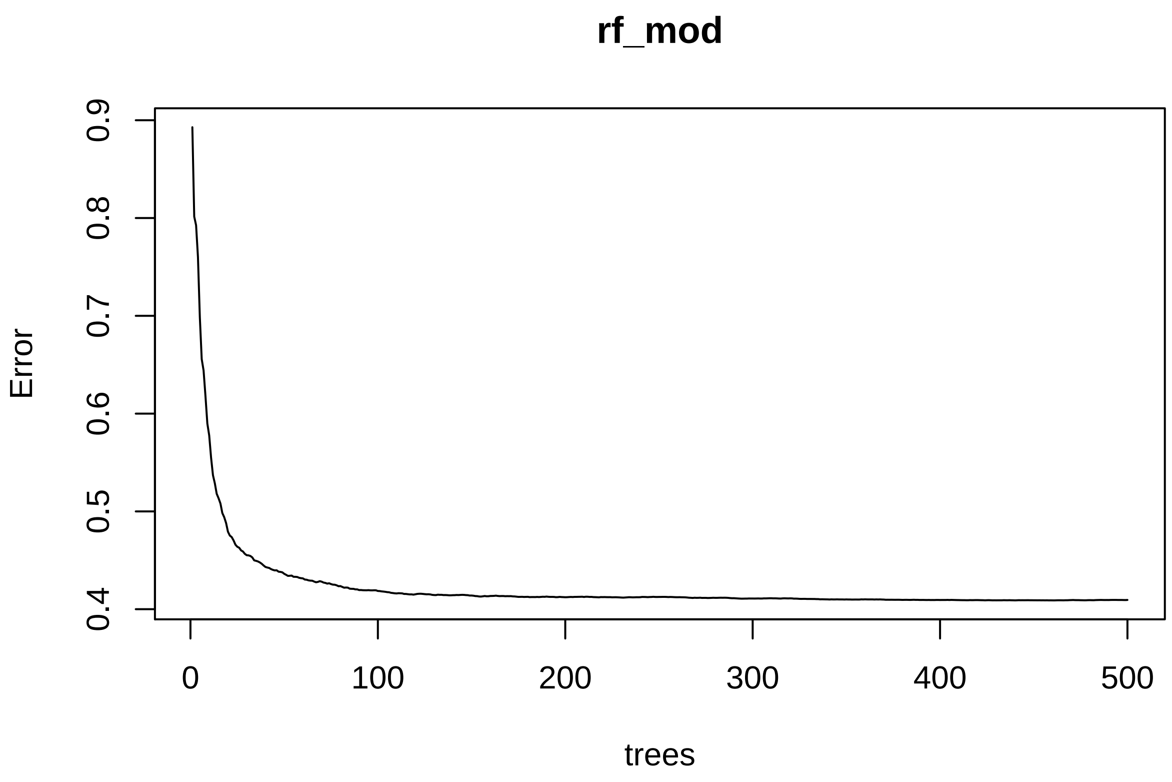
```
##    3    4    5    6    7    8    9
##    7    0  637 1110  196    0    0
```

Random Forest

```
set.seed(13)
rf_mod <- randomForest(quality~., data=train)
rf_mod
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.409441
##                     % Var explained: 46.36
```

```r
plot(rf_mod)
```



We can see that the error rate drops as the number of trees increases. From the original model, we have that mtry=4 and the number of trees = 500. Our error is 40.07% and R^2 is 46.62%. These values are not great, and can possibly be better if we use a different tuning pattern.

```r
#Do not run this code
set.seed(13)
oob.err = double(12)
test.err = double(12)
for(mtry in 1:12)
{
  rf=randomForest(quality ~ . , data = train,
                  mtry=mtry,ntree=500)
  oob.err[mtry] = rf$mse[500]

  pred<-predict(rf,test)
  test.err[mtry]= with(test, mean( (quality - pred)^2))
}
```

This code will run different mtry values to help identify the best value.

```
test.err
```

```
##  [1] 0.4374069 0.3888922 0.3862525 0.3878580 0.3869541 0.3911510 0.3882823
##  [8] 0.3875853 0.3883616 0.3896888 0.3905412 0.3899126
```

```
oob.err
```

```
##  [1] 0.4524254 0.4096483 0.4077176 0.4109751 0.4071729 0.4102752 0.4111538
##  [8] 0.4119924 0.4127557 0.4151919 0.4172444 0.4177343
```

mtry of 3 is the best, according to the errors.

```
bestrf <- randomForest(quality~., data=train, mtry=3)
bestrf
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = train, mtry = 3)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.4081998
##                     % Var explained: 46.52
```

```
postResample(pred=predict(bestrf,test),obs=test$quality)
```

```
##      RMSE  Rsquared       MAE
## 0.6232783 0.4919401 0.4634134
```

```
postResample(pred=predict(rf_mod,test),obs=test$quality)
```

```
##      RMSE  Rsquared       MAE
## 0.6210688 0.4940470 0.4613818
```

Even though the mtry at 3 is the best, according to the errors, the model actually does not improve when using an mtry at 4. The first output is with mtry = 3 and the second output is mtry=4. Although the values are extremely similar, mtry=4 produces the better output. The random forest method produces a MSE of 0.6210688^2 = 38.57%

Stepwise Regression

```
step.model <- train(quality~., data=train,
                    method="lmStepAIC",
                    trControl=train_control)
```

```
round(coef(step.model$finalModel),5)
```

```
##          (Intercept)       wine_typewhite        fixed.acidity
##            100.76650             -0.31114              0.08273
##     volatile.acidity       residual.sugar            chlorides
##             -1.43172              0.06100             -0.75505
##  free.sulfur.dioxide total.sulfur.dioxide              density
##              0.00602             -0.00154           -100.07497
##                   pH            sulphates              alcohol
##              0.54432              0.77062              0.21621
```

```
postResample(pred=predict(step.model,test),obs=test$quality)
```

```
##      RMSE  Rsquared       MAE
## 0.7277658 0.2967940 0.5604845
```

My step model only removed one predictor, citric.acid. According to the model, all the other predictors are valuable and contributes to the quality response. We can see that if we hold all the predictors to 0, we have a quality of over a 100, which doesn't make sense. Looks like the density predictor helps cancel out the high intercept. Surprisingly, if the wine is white, it actually brings down the quality of the wine. This contradicts the graph and table representation of the original data set. The volatility of the acidity has a strong negative effect on the quality of the wine, which could make sense because people like having a constant flavor. For model, my MSE is really high, at $0.7277658^2 =$ 52.964%. This model performs worst than the random forest model.

SVM Regression

```
model.svm <- train(quality~., data=train,
                   trControl=train_control,
                   method="svmLinear",
                   preProcess=c("center","scale"),
                   tuneLength=10)
model.svm
```

```
## Support Vector Machines with Linear Kernel
##
## 3497 samples
##   12 predictor
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 3148, 3147, 3148, 3146, 3148, 3148, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.7433607  0.2799241  0.5686144
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
postResample(pred=predict(model.svm,test),obs=test$quality)
```

```
##      RMSE  Rsquared       MAE
## 0.7314075 0.2919625 0.5598494
```

Using the support vector machine algorithm, I have a higher MSE than any of the other models I used. SVM is not a good model for this data set.

Conclusion

From all the models we used, the random forest model ended up being the best with a MSE of 38.57%. From our step model, we only had one predictor that was not relevant for the model, citric.acid, although the MSE for the step model didn't end up being our best model. We used both the regression approach and the classification approach. We ended up using the regression model for our final model because it is hard to quantify the classification approach because we can only go through using accuracy as a means of error. Despite this, we wanted to see how "good" it would be if we did take that approach. If the accuracy ended up being really high, then we most likely would have stuck with choosing the classification method. We started off the analysis with visualizing the original data set and then predicting the wine_type response. The reason why we chose to predict wine type is because having prior information on the type of wine will help us give a good estimate on whether or not our quality response make sense. For example, white wine looked to have a higher quality, on average, than the red wine. This means that if the quality is high, more likely than not, we expect the wine type to be white. This helped us question any outrageous quality prediction. Our analysis could have been improved if we received better data. From my regression models, I get a high intercept and an extremely negative predictor,density, which I believe played a huge role in the error rate. Since the p value was significant, I could not ignore the density predictor and had to leave it in my model. If we were to get data that could counteract the large coefficient value of density, then I believe we could have gotten an extremely low MSE.