#Loading Possible Libraries

library(tidyverse)

#library(carat)

library(caret)

library(leaps)

library(glmnet)

library(leaps)

library(ggplot2)

library(earth)

library(mgcv)

library(class)

library(readr)

library(xgboost)

library(Ckmeans.1d.dp)

library(pdp)

library(ROCR)

library(randomForest)

library(plotROC)

library(sqldf)

library(readr)

#Setting the Seed so that when I use random functions it will be the same random pull each time so my numbers don't change

set.seed(123)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#loading in the data

diamond <- read\_csv("ENOVA/training.csv")

diamond <- data.frame(diamond)

colnames(diamond)

sapply(diamond, typeof)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Turning Measurements into 3 separate columns so I can work with Continuous Data

#Making sure the new values are read in as doubles/numeric

diamond <- separate\_wider\_delim(diamond, cols = Measurements, delim = "x", names = c("Length", "Width", "Depth2"))

diamond$Length <- as.double(diamond$Length)

diamond$Depth <- as.double(diamond$Depth)

diamond$Depth2 <- as.double(diamond$Depth2)

diamond$Width <- as.double(diamond$Width)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Checking for Missing values for each column

MV <-NULL

for (i in 1:21){

MV[i] <- sum(is.na(diamond[i]))

print(names(diamond)[i])

print(MV[i])

}

#MISSING VALUES

#CUT 3922 CHARACTER

#DEPTH 1440 DOUBLE

#POLISH 899 CHARACTER

#SYMMETRY 899 CHARACTER

#TABLE 2531 DOUBLE

#Creating functions to deal with missing values

#Imputes Median

impute\_median <- function(x){

ind\_na <- is.na(x)

x[ind\_na] <- median(x[!ind\_na])

as.numeric(x)

}

#Imputes 'Missing' for character columns

impute\_m <- function(x){

ind\_na <- is.na(x)

x[ind\_na] <- 'Missing'

factor(x)

}

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Using SQL to manage data and create new variables I feel more comfortable to do it this way

#Creating a new Depth % column with Actual values and Estimated Values where there were NAs

#Creating a Depth % flag for where the rows that initially had NAs

#Grouping Colors together based on research found from diamond experts groups were based on their input

#Grouping Clarity together based on research found from diamond experts groups were based on their input

#WHEN Clarity like 'FL' then 'Flawless'

diamond <- sqldf("select \*

, Case when Depth is NULL or Depth = 0.0 then Round((Depth2/Width)\*100,1) else Depth end as Depth\_imputed

, Case when Depth is NULL or Depth = 0.0 then 1 else 0 end as Depth\_imputed\_flag

, CASE WHEN Color in ('D', 'E', 'F') or Color like 'F%' then 'Colorless'

WHEN Color in ('G', 'H', 'I', 'J') or Color like 'G%' then 'NearColorless'

WHEN Color in ('K', 'L', 'M') or Color like 'L%' then 'FaintColor'

Else 'ColoredDiamond' end as Color\_Adj

, CASE

when Clarity in ('I1', 'I2', 'I3', 'N') then 'Included'

when Clarity in ('IF', 'FL') or Clarity like 'None' then 'Internally Flawless'

when Clarity in ('SI1', 'SI2') then 'Slightly Included'

when Clarity in ('VS1', 'VS2') then 'Very Slightly Included'

when Clarity in ('VVS1', 'VVS2') then 'Very Very Slightly Included'

else 'Other' end as Clarity\_Adj

from diamond")

#Trying to Table input median value for Table grouped by shape but each same had the same median Table value

#so Table overall Median was inputted instead of median by Shape

max(diamond$Table, na.rm = TRUE)

min(diamond$Table, na.rm = TRUE)

diamond %>%

group\_by(Shape)%>%

summarise(median(Table, na.rm = TRUE))

diamond[, "TABLE\_MISSING\_FLAG"] <- as.numeric(is.na(diamond$Table))

diamond <- diamond %>%

mutate(Table = impute\_median(diamond$Table))

#Fixing Shape

#values needed capitalized as some had all caps

unique(diamond$Shape)

n\_distinct(diamond$Shape)

diamond$Shape <- toupper(diamond$Shape)

#Fixing and Imputing Missing values for Symmetry

#values needed capitalized as some had all caps

#one value was misspelled

unique(diamond$Symmetry)

diamond %>%

filter(Symmetry == "Execllent") %>%

nrow()

diamond <- diamond %>%

mutate(Symmetry = replace(Symmetry,Symmetry == "Execllent", "Excellent"))

diamond <- diamond %>%

mutate(Symmetry = impute\_m(diamond$Symmetry))

diamond$Symmetry <- toupper(diamond$Symmetry)

#Imputing Cut

diamond <- diamond %>%

mutate(Cut = impute\_m(diamond$Cut))

#Imputing Polish

diamond <- diamond %>%

mutate(Polish = impute\_m(diamond$Polish))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Making sure that the columns were either numeric or factors so modeling would be easier

diamond <- diamond %>%

mutate(LogPrice = as.numeric(LogPrice),

Price = as.numeric(Price),

Carats = as.numeric(Carats),

Cert = as.factor(Cert),

Clarity = as.factor(Clarity),

Clarity\_Adj = as.factor(Clarity\_Adj),

Color = as.factor(Color),

Color\_Adj = as.factor(Color\_Adj),

Cut = as.factor(Cut),

Depth = as.numeric(Depth),

Known\_Conflict\_Diamond = as.factor(Known\_Conflict\_Diamond),

Length = as.numeric(Length),

Width = as.numeric(Width),

Depth2 = as.numeric(Depth2),

Polish = as.factor(Polish),

Regions = as.factor(Regions),

Shape = as.factor(Shape),

Symmetry = as.factor(Symmetry),

Table = as.numeric(Table),

Vendor = as.factor(Vendor),

Depth\_imputed = as.numeric(Depth\_imputed),

Depth\_imputed\_flag = as.factor(Depth\_imputed\_flag),

TABLE\_MISSING\_FLAG = as.factor(TABLE\_MISSING\_FLAG)

)

#Splitting dataframe into training and Validation/Test sets by id column

train <- diamond %>% sample\_frac(0.7)

testing <- anti\_join(diamond, train, by = 'id')

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Random Forest

#Initial Model with all columns in it

rf\_diamond <- randomForest(Price ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG

, data=train, ntree = 500,

importance = TRUE)

#Variable Importance Plot

#This is shows the most important variables to help the model predict Price

varImpPlot(rf\_diamond,

sort = TRUE,

n.var = 10,

main = "Look for Variables Below Random Variable"

)

# This plot shows how MSE decreases and smoothes out as the # of trees increases

plot(rf\_diamond, main = "Number of Trees Compared to MSE")

#Creating another train dataset so the original isn't disturbed

train\_select <- train %>% select(Price, Carats , Clarity\_Adj , Color\_Adj , Cut , Known\_Conflict\_Diamond , Length , Width ,

Depth2 , Polish , Regions , Shape , Symmetry , Table , Depth\_imputed ,

Depth\_imputed\_flag , TABLE\_MISSING\_FLAG)

#Tuning Random Forest

#Finding the best Random Forest Parameters

tuneRF(x = train\_select[,-1], y = train\_select[,1],

plot = TRUE, ntreeTry = 300, stepFactor = .5)

#Addinga column to the new data set with Random values from a normal distribution

train\_select$random <- rnorm(5635)

#Creating a new model with random this way we can see if any columns have less importance than a random variable

rf\_diamond\_VS <- randomForest(Price ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG + random

, data=train\_select,

ntree = 500, mtry = 10, importance = TRUE)

#Variable Importance Plot2

varImpPlot(rf\_diamond\_VS,

sort = TRUE,

n.var = 15,

main = "Look for Variables Below Random Variable"

)

importance((rf\_diamond\_VS))

#FINAL MODEL

#Removed most variables as they were less important than the random variable created

rf\_diamond2 <- randomForest(Price ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2 + Symmetry, data=train,

ntree = 500, mtry = 7, importance = TRUE)

#Variable Importance same as the plot, but just the numbers

importance(rf\_diamond2)

#predicting the values of the test set

predictions <- predict(rf\_diamond2, newdata = testing, type = "response")

# Calculate the mean absolute error

MAE <- mean(abs(predictions - testing$Price))

print(paste("Mean Absolute Error:", MAE))

# Calculate the mean squared error

MSE <- mean((predictions - testing$Price)^2)

print(paste("Mean Squared Error:", MSE))

# Calculate R-squared value

SSR <- sum((predictions - mean(testing$Price))^2)

SST <- sum((testing$Price - mean(testing$Price))^2)

R\_squared <- SSR/SST

print(paste("R-squared Value:", R\_squared))

#Partial Plot to help interpret the relationship between carat and price

partialPlot(rf\_diamond2, train,Carats)

View(data.frame(id = testing$id, actual = testing$Price, predicted = predictions, diff = testing$Price-predictions))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#XGBOOST

#Creating a matrix to use for the XGBOOST

train\_x <- model.matrix(Price ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG, data = train)[,-1]

#Traget Variable for model

train\_y <- train$Price

#BUILDING INITIAL MODEL

set.seed(123)

xgb\_insur <- xgboost(data =train\_x, label= train\_y, subsample =.5, nrounds = 100)

#CROSS VALIDATION MODEL

set.seed(123)

xgb\_insur2 <- xgb.cv(data =train\_x, label= train\_y, subsample =.5, nrounds = 100, nfold= 10)

summary(xgb\_insur2)

#TUNING OUR MODEL

tune\_grid <- expand.grid(

nrounds = 10, #CHOOSE THE NUMBER OF ROUNDS THAT IS BEST FOR YOUR MODEL

eta = c(0.1, 0.15, 0.2, 0.25, 0.3),

max\_depth = c(1:10),

gamma = c(0),

colsample\_bytree = 1,

min\_child\_weight = 1,

subsample = c(0.25, 0.5, 0.75, 1)

)

set.seed(123)

xgb.insur.caret <- train(x = train\_x, y = train\_y,

method = "xgbTree",

tuneGrid = tune\_grid,

trControl = trainControl(method = 'cv', # Using 10-fold cross-validation

number = 10))

#Ploting our graphs to see how the tuning affected each model

plot(xgb.insur.caret)

#You can determine the best tune from the plots above, but this is the simplest way to see the best parameters

xgb.insur.caret$bestTune

#MODEL AFTER TUNING

xgb\_insur3 <- xgboost(data =train\_x, label= train\_y, subsample =1, nrounds = 10, eta = .3, max\_depth = 4, objective = "reg:squarederror")

#Variable Importance Plot

xgb.importance(feature\_names = colnames(train\_x), model = xgb\_insur3)

xgb.ggplot.importance(xgb.importance(feature\_names = colnames(train\_x), model = xgb\_insur3))

#Random Variable if needed for Variable selection

#train$random <- rnorm(5635)

#VARIABLE SELECTION PROCESS

#Creating a new matrix with random variable

train\_x\_VS <- model.matrix(Price ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG + random, data = train)[,-1]

set.seed(123)

#Creating the validation Model with the Parameters above

xgb\_insur\_VS <- xgboost(data =train\_x\_VS, label= train\_y, subsample =1, nrounds = 10, eta = .3, max\_depth = 4, objective = "reg:squarederror")

#Variable Importance Plot

xgb.ggplot.importance(xgb.importance(feature\_names = colnames(train\_x\_VS), model = xgb\_insur\_VS))

xgb.importance(feature\_names = colnames(train\_x\_VS), model = xgb\_insur\_VS)

#Final Model

#Creating the Model with the Parameters above

train\_x\_final <- model.matrix(Price ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2, data = train)[,-1]

xgb\_diamond\_final <- xgboost(data =train\_x\_final, label= train\_y, subsample =.1, nrounds = 10, eta = .3, max\_depth = 4, objective = "reg:squarederror")

#Creating this test matrix to get the predictions

test\_x <- model.matrix(Price ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2, data = testing)[,-1]

#Predicting the values in test matrix so we can measure our model

predictions2 <- predict(xgb\_diamond\_final, newdata = test\_x, type = "response")

#Making sure names were similar

#xgb\_diamond\_final[["feature\_names"]]

#colnames(test\_x)

# Calculate the mean absolute error

MAE2 <- mean(abs(predictions2 - testing$Price))

print(paste("Mean Absolute Error:", MAE2))

# Calculate the mean squared error

MSE2 <- mean((predictions2 - testing$Price)^2)

print(paste("Mean Squared Error:", MSE2))

# Calculate R-squared value

SSR2 <- sum((predictions2 - mean(testing$Price))^2)

SST2 <- sum((testing$Price - mean(testing$Price))^2)

R\_squared2 <- SSR2/SST2

print(paste("R-squared Value:", R\_squared2))

#Partial Plot to help interpret the relationship between carat and price

partial(xgb\_diamond\_final, pred.var = "Carats",

plot = TRUE, rug = TRUE, alpha = 0.1,

plot.engine = "lattice", train = train\_x\_final)

View(data.frame(actual = testing$Price, predicted = predictions2, diff = testing$Price-predictions2))

View(data.frame(id = testing$id, actual = testing$Price, RFpredicted = predictions, XGBpredicted = predictions2, RFdiff = testing$Price-predictions, XGBdiff = testing$Price-predictions2))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Comparing Models

print(paste("XGB Mean Absolute Error:", MAE2))

print(paste("RF Mean Absolute Error:", MAE))

# Calculate the mean squared error

print(paste("XGB Mean Squared Error:", MSE2))

print(paste("RF Mean Squared Error:", MSE))

# Calculate R-squared value

print(paste("XGB R-squared Value:", R\_squared2))

print(paste("RF R-squared Value:", R\_squared))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Random Forest for LogPrice

#Initial Model with all columns in it

rf\_diamondLP <- randomForest(LogPrice ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG

, data=train, ntree = 500,

importance = TRUE)

varImpPlot(rf\_diamondLP,

sort = TRUE,

n.var = 10,

main = "Look for Variables Below Random Variable"

)

plot(rf\_diamondLP, main = "Number of Trees Compared to MSE")

train\_selectLP <- train %>% select(LogPrice, Carats , Clarity\_Adj , Color\_Adj , Cut , Known\_Conflict\_Diamond , Length , Width ,

Depth2 , Polish , Regions , Shape , Symmetry , Table , Depth\_imputed ,

Depth\_imputed\_flag , TABLE\_MISSING\_FLAG)

#Tuning Random Forest

tuneRF(x = train\_selectLP[,-1], y = train\_selectLP[,1],

plot = TRUE, ntreeTry = 300, stepFactor = 0.5)

train\_selectLP$random <- rnorm(5635)

rf\_diamond\_VSLP <- randomForest(LogPrice ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG + random

, data=train\_selectLP,

ntree = 300, mtry = 10, importance = TRUE)

varImpPlot(rf\_diamond\_VSLP,

sort = TRUE,

n.var = 11,

main = "Look for Variables Below Random Variable"

)

#set.seed(14)

#FINAL MODEL

rf\_diamond2LP <- randomForest(LogPrice ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2, data=train,

ntree = 300, mtry = 6, importance = TRUE)

importance(rf\_diamond2)

summary(rf\_diamond2)

typeof(list(rf\_diamond2$predicted))

sapply(testing, typeof)

sapply(train, typeof)

str(testing2)

str(train)

unique(train$Color\_Adj)

predictions3 <- predict(rf\_diamond2LP, newdata = testing, type = "response")

#train$prob <- predict(rf\_diamond2, type = "prob")[,2]

#plotROC(train$Price, train$prob)

#AUROC(train$Price, training$prob)

# Calculate the mean absolute error

MAE3 <- mean(abs(predictions3 - testing$LogPrice))

print(paste("Mean Absolute Error:", MAE3))

# Calculate the mean squared error

MSE3 <- mean((predictions3 - testing$LogPrice)^2)

print(paste("Mean Squared Error:", MSE3))

# Calculate R-squared value

SSR3 <- sum((predictions3 - mean(testing$LogPrice))^2)

SST3 <- sum((testing$LogPrice - mean(testing$LogPrice))^2)

R\_squared3 <- SSR3/SST3

print(paste("R-squared Value:", R\_squared3))

View(data.frame(actual = testing$LogPrice, predicted = predictions3, diff = testing$LogPrice-predictions3))

View(data.frame(actual = testing$Price, predicted = exp(predictions3), diff = testing$Price-exp(predictions3)))

predictions4 <- exp(predictions3)

#train$prob <- predict(rf\_diamond2, type = "prob")[,2]

#plotROC(train$Price, train$prob)

#AUROC(train$Price, training$prob)

# Calculate the mean absolute error

MAE4 <- mean(abs(predictions4 - testing$Price))

print(paste("Mean Absolute Error:", MAE4))

# Calculate the mean squared error

MSE4 <- mean((predictions4 - testing$Price)^2)

print(paste("Mean Squared Error:", MSE4))

# Calculate R-squared value

SSR4 <- sum((predictions4 - mean(testing$Price))^2)

SST4 <- sum((testing$Price - mean(testing$Price))^2)

R\_squared4 <- SSR4/SST4

print(paste("R-squared Value:", R\_squared4))

print(paste("R-squared Value:", R\_squared))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Final Predictions

#Bringing in Offers as the final data frame

offers <- read\_csv("ENOVA/offers.csv")

offers <- data.frame(offers)

colnames(offers)

sapply(offers, typeof)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Turning Measurements into 3 separate columns so I can work with Continuous Data

#Making sure the new values are read in as doubles/numeric

offers$Measurements <- chartr('\*', "x", offers$Measurements)

offers <- separate\_wider\_delim(offers, cols = Measurements, delim = "x", names = c("Length", "Width", "Depth2")) #, too\_few = "debug")

offers$Length <- as.double(offers$Length)

offers$Depth <- as.double(offers$Depth)

offers$Depth2 <- as.double(offers$Depth2)

offers$Width <- as.double(offers$Width)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Checking for Missing values for each column

MV2 <-NULL

for (i in 1:18){

MV2[i] <- sum(is.na(offers[i]))

print(names(offers)[i])

print(MV2[i])

}

#MISSING VALUES

#CUT 3922 CHARACTER

#DEPTH 1440 DOUBLE

#POLISH 899 CHARACTER

#SYMMETRY 899 CHARACTER

#TABLE 2531 DOUBLE

#WHEN Clarity like 'FL' then 'Flawless'

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

offers <- sqldf("select \*

, Case when Depth is NULL or Depth = 0.0 then Round((Depth2/Width)\*100,1) else Depth end as Depth\_imputed

, Case when Depth is NULL or Depth = 0.0 then 1 else 0 end as Depth\_imputed\_flag

, CASE WHEN Color in ('D', 'E', 'F') or Color like 'F%' then 'Colorless'

WHEN Color in ('G', 'H', 'I', 'J') or Color like 'G%' then 'NearColorless'

WHEN Color in ('K', 'L', 'M') or Color like 'L%' then 'FaintColor'

Else 'ColoredDiamond' end as Color\_Adj

, CASE when Clarity in ('I1', 'I2', 'I3', 'N') then 'Included'

when Clarity in ('IF', 'FL') or Clarity like 'None' then 'Internally Flawless'

when Clarity in ('SI1', 'SI2') then 'Slightly Included'

when Clarity in ('VS1', 'VS2') then 'Very Slightly Included'

when Clarity in ('VVS1', 'VVS2') then 'Very Very Slightly Included'

else 'Other' end as Clarity\_Adj

from offers")

offers <- offers %>%

mutate(Symmetry = impute\_m(offers$Symmetry))

offers$Symmetry <- toupper(offers$Symmetry)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Making sure that the columns were either numeric or factors so modeling would be easier

offers <- offers %>%

mutate(Carats = as.numeric(Carats),

Cert = as.factor(Cert),

Clarity = as.factor(Clarity),

Clarity\_Adj = as.factor(Clarity\_Adj),

Color = as.factor(Color),

Color\_Adj = as.factor(Color\_Adj),

Cut = as.factor(Cut),

Depth = as.numeric(Depth),

Known\_Conflict\_Diamond = as.factor(Known\_Conflict\_Diamond),

Length = as.numeric(Length),

Width = as.numeric(Width),

Depth2 = as.numeric(Depth2),

Polish = as.factor(Polish),

Regions = as.factor(Regions),

Shape = as.factor(Shape),

Symmetry = as.factor(Symmetry),

Table = as.numeric(Table),

Vendor = as.factor(Vendor),

Depth\_imputed = as.numeric(Depth\_imputed),

Depth\_imputed\_flag = as.factor(Depth\_imputed\_flag),

)

#Bringing only the columns used in the model

x\_final <- offers %>% select(Carats, Clarity\_Adj, Color\_Adj, Length , Width ,

Depth2, Symmetry)

#Predicting the prce of the offers diamonds with the original random forest model

final\_predictions <- predict(rf\_diamond2, newdata = x\_final, type = "response")

#Creating a data frame

offers\_final <- data.frame(offers, predictions = final\_predictions)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

diamond2 <- diamond %>%

mutate(Retail = as.numeric(Retail),

Carats = as.numeric(Carats),

Cert = as.factor(Cert),

Clarity = as.factor(Clarity),

Clarity\_Adj = as.factor(Clarity\_Adj),

Color = as.factor(Color),

Color\_Adj = as.factor(Color\_Adj),

Cut = as.factor(Cut),

Depth = as.numeric(Depth),

Known\_Conflict\_Diamond = as.factor(Known\_Conflict\_Diamond),

Length = as.numeric(Length),

Width = as.numeric(Width),

Depth2 = as.numeric(Depth2),

Polish = as.factor(Polish),

Regions = as.factor(Regions),

Shape = as.factor(Shape),

Symmetry = as.factor(Symmetry),

Table = as.numeric(Table),

Vendor = as.factor(Vendor),

Depth\_imputed = as.numeric(Depth\_imputed),

Depth\_imputed\_flag = as.factor(Depth\_imputed\_flag),

TABLE\_MISSING\_FLAG = as.factor(TABLE\_MISSING\_FLAG)

)

#Splitting dataframe into training and Validation/Test sets by id column

train2 <- diamond2 %>% sample\_frac(0.7)

testing2 <- anti\_join(diamond2, train2, by = 'id')

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Random Forest For Offers

#Initial Model with all columns in it

rf\_diamond\_o <- randomForest(Retail ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG

, data=train2, ntree = 500,

importance = TRUE)

#Variable Importance Plot

#This is shows the most important variables to help the model predict Price

varImpPlot(rf\_diamond\_o,

sort = TRUE,

n.var = 10,

main = "Look for Variables Below Random Variable"

)

# This plot shows how MSE decreases and smoothes out as the # of trees increases

plot(rf\_diamond\_o, main = "Number of Trees Compared to MSE")

#Creating another train2 dataset so the original isn't disturbed

train2\_select <- train2 %>% select(Retail, Carats , Clarity\_Adj , Color\_Adj , Cut , Known\_Conflict\_Diamond , Length , Width ,

Depth2 , Polish , Regions , Shape , Symmetry , Table , Depth\_imputed ,

Depth\_imputed\_flag , TABLE\_MISSING\_FLAG)

#Tuning Random Forest

#Finding the best Random Forest Parameters

tuneRF(x = train2\_select[,-1], y = train2\_select[,1],

plot = TRUE, ntreeTry = 300, stepFactor = .5)

#Adding a column to the new data set with Random values from a normal distribution

train2\_select$random <- rnorm(5635)

#Creating a new model with random this way we can see if any columns have less importance than a random variable

rf\_diamond\_o\_VS <- randomForest(Retail ~ Carats + Clarity\_Adj + Color\_Adj + Cut + Known\_Conflict\_Diamond + Length + Width +

Depth2 + Polish + Regions + Shape + Symmetry + Table + Depth\_imputed +

Depth\_imputed\_flag + TABLE\_MISSING\_FLAG + random

, data=train2\_select,

ntree = 500, mtry = 10, importance = TRUE)

#Variable Importance Plot2

varImpPlot(rf\_diamond\_o\_VS,

sort = TRUE,

n.var = 15,

main = "Look for Variables Below Random Variable"

)

importance((rf\_diamond\_o\_VS))

#FINAL MODELS for Retail Price

#Removed most variables as they were less important than the random variable created

rf\_diamond\_o2 <- randomForest(Retail ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2 + Symmetry + Known\_Conflict\_Diamond +Regions, data=train2,

ntree = 500, mtry = 7, importance = TRUE)

rf\_diamond\_o3 <- randomForest(Retail ~ Carats + Clarity\_Adj + Color\_Adj + Length + Width +

Depth2 + Symmetry + Known\_Conflict\_Diamond +Regions, data=train2,

ntree = 500, mtry = 7, importance = TRUE)

#Variable Importance same as the plot, but just the numbers

importance(rf\_diamond\_o2)

predictions5 <- predict(rf\_diamond\_o2, newdata = testing2, type = "response")

MAE5 <- mean(abs(predictions5 - testing$Retail))

print(paste("Mean Absolute Error:", MAE5))

# Calculate the mean squared error

MSE5 <- mean((predictions5 - testing$Retail)^2)

print(paste("Mean Squared Error:", MSE5))

# Calculate R-squared value

SSR5 <- sum((predictions5 - mean(testing$Retail))^2)

SST5 <- sum((testing$Retail - mean(testing$Retail))^2)

R\_squared5 <- SSR5/SST5

print(paste("R-squared Value:", R\_squared5))

print(paste("R-squared Value:", R\_squared))

#predicting the values of the test set

x\_final2 <- offers %>% select(Carats, Clarity\_Adj, Color\_Adj, Length , Width ,

Depth2, Symmetry, Known\_Conflict\_Diamond, Regions)

predictionso <- predict(rf\_diamond\_o2, newdata = x\_final2, type = "response")

offers\_final <- data.frame(offers, predictions = final\_predictions, retail\_pred = predictionso, gain = predictionso-final\_predictions)

write\_xlsx(offers\_final, "C:\\Users\\mikey\\OneDrive\\Documents\\ENOVA\\offers\_final.xlsx")

write\_xlsx(diamond, "C:\\Users\\mikey\\OneDrive\\Documents\\ENOVA\\diamond.xlsx")

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

#Short answer Problems Analysis

#Problem 1

diamond %>%

group\_by(Vendor)%>%

summarise(median(Price, na.rm = TRUE))

diamond %>%

group\_by(Vendor)%>%

summarise(median(Carats, na.rm = TRUE))

diamond %>%

group\_by(Vendor)%>%

summarise(mean(Price, na.rm = TRUE))

res\_aov <- aov(Price ~ Vendor, data = diamond)

summary(res\_aov)

par(mfrow = c(1, 2)) # combine plots

# histogram

hist(res\_aov$residuals)

# QQ-plot

qqPlot(res\_aov$residuals,

id = FALSE # id = FALSE to remove point identification

)

crosstab(Color\_Adj, Vendor, data = diamond)

crosstab(Clarity\_Adj, Vendor, data = diamond)

crosstab(Shape, Vendor, data = diamond)

#PROBLEM 2

cor(diamond$Carats, diamond$Price)

diamond %>%

group\_by(round(Carats,digits = 0))%>%

summarise(median(Price, na.rm = TRUE))

diamond %>%

group\_by(round(Carats,digits = 0))%>%

summarise(mean(Price, na.rm = TRUE))

ggplot(diamond, aes(x = Carats, y = Price))+

geom\_point()+

geom\_smooth(method = lm, se = FALSE)

#Extremely Helpful Charts to help me select which Diamonds to Purchase after looking at the biggest gain

View(diamond %>%

group\_by(Color\_Adj,round(Carats,digits = 0))%>%

summarise(median(Price, na.rm = TRUE)))

View(diamond %>%

group\_by(Clarity\_Adj,round(Carats,digits = 0))%>%

summarise(median(Price, na.rm = TRUE)))

diamond %>%

group\_by(round(Carats,digits = 0))%>%

summarise(median(Price, na.rm = TRUE))

diamond %>%

group\_by(round(Depth2,digits = 0))%>%

summarise(median(Price, na.rm = TRUE))

diamond %>%

group\_by(Regions)%>%

summarise(median(Price, na.rm = TRUE))

diamond %>%

group\_by(Clarity\_Adj)%>%

summarise(median(Price, na.rm = TRUE))