Final Project: A Data Mining Approach to Diamonds

Janavi Kumar, PSTAT 131, Spring 2017

June 2nd, 2017

Abstract Summary:

I used a data set detailing diamonds from the Kaggle online database system.[1]. I analyzed this data via predictive tactics in order to understand relationships between price of the diamond and its other attributes. The data mining techniques used **linear regression, classification trees,** and **kNN classification**. The key results led me to reinforce that there was indeed a definite tradeoff in carat and both color and clarity. Therefore in conclusion, price of diamond can be predicted if the size variables are held constant, and that the most expensive diamonds don't necessarily correspond to the "best" diamond, best alluding to the least sacrifice amongst the 4 C's (carat, cut, color, clarity), which are the most important attributes people take into consideration when purchasing a diamond.

Figure 1: Diamond.

Introduction:

I am using a data set containing a data frame of record data containing 53940 observations and 10 attributes, some ordinal categorical (clarity, color of diamond, cut) as well as continuous numeric ratio (depth, price, carat Iight, x, y, and z length, table width of diamond).

I formed my key questions and objectives AFTER visualizing the data, so I could do the relevant preprocessing and visualization that would allow me to see relationships between the attributes and see what was a viable objective.

The questions I am trying to address:

- People always seem to talk about the carat when showing off their diamonds - however, can price of a diamond be predicted based solely on the value of the cut, clarity, and color? Or is the carat the main driving force behind diamond price?
- Are the most expensive diamonds really the "best?"
- What are some attribute tradeoffs amongst all the attributes? If any, what combination of attributes would give you the "best" diamond?

I approached this problem mainly as a classification problem, to see if certain combinations of the 4 C's would be considered "Expensive" or not, and I did this via classification decision trees and support vector machines - however, I also modelled a regression based on the data to see if I could do predictive analysis on the continuous variable price.

Mining the Data

Data Preprocessing and Visualization

To begin, I read in the data. I then gathered summary statistics, but must do some preprocessing in order to do so. I relabelled the attributes and omitted any missing values for ease of calculation and changed the necessary data to numerical format for double precision:

```
dat = read.table("diamonds.txt", header = TRUE, sep = ",", quote = "",
    fill = TRUE)
diamonds <- na.omit(diamonds)

diamonds <- data.frame(dat)
diamonds <- diamonds %>% select(-X)
colnames(diamonds) <- c("Carat", "Cut", "Color", "Clarity", "Depth",
    "Table", "Price", "X", "Y", "Z")
head(diamonds)
```

```
##   Carat Cut Color Clarity Depth Table Price    X    Y    Z
## 1  0.23   5     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   4     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23   2     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   4     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31   2     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24   3     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
(meanPrice = mean(dat$price))
```

```
## [1] 3932.8
```

```
(varPrice = var(dat$price))
```

```
## [1] 15915629
```

```
(medianPrice = median(dat$price))
```

```
## [1] 2401
```

```
(MADPrice = mad(dat$price))
```

```
## [1] 2475.942
```

I came to the conclusion that the median and MAD are less sensitive to outliers than the mean and variance, so I used the median as our chief summary statistic. Now that I have acquired the appropriate summary statistic, I can continue to preprocess the data and create an binary valued attribute "Expensive" based on the median. The structure of the data is revealed below as well:

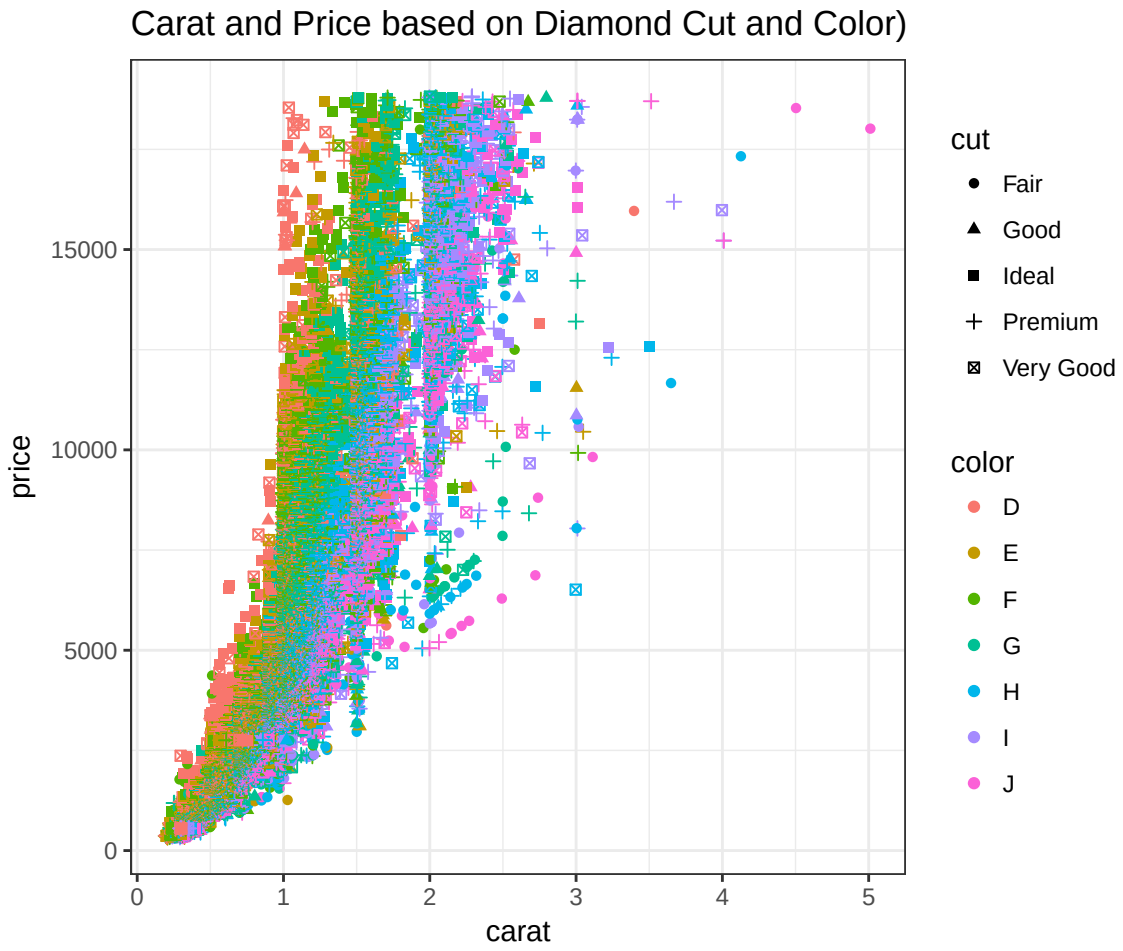
```
diamonds = diamonds %>% mutate(Expensive = as.factor(ifelse(Price <
    median(Price), "No", "Yes")))
str(diamonds)
```

```
## 'data.frame':    53940 obs. of  11 variables:
##  $ Carat    : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ Cut      : int  5 4 2 4 2 3 3 3 1 3 ...
##  $ Color    : Factor w/ 7 levels "D","E","F","G",..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ Clarity  : Factor w/ 8 levels "I1","IF","SI1",..: 4 3 5 6 4 8 7 3 6 5 ...
##  $ Depth    : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ Table    : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ Price    : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ X        : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ Y        : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ Z        : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
##  $ Expensive: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Next, I explored the data in order to gain some insight into the relationships between the attributes to see if further preprocessing or tweaking to my objectives was necessary. First, in order to do so, I gave discrete values to Cut, as they are currently being read in as continuous by ggplot which would not allow that. The data is represented graphically below:

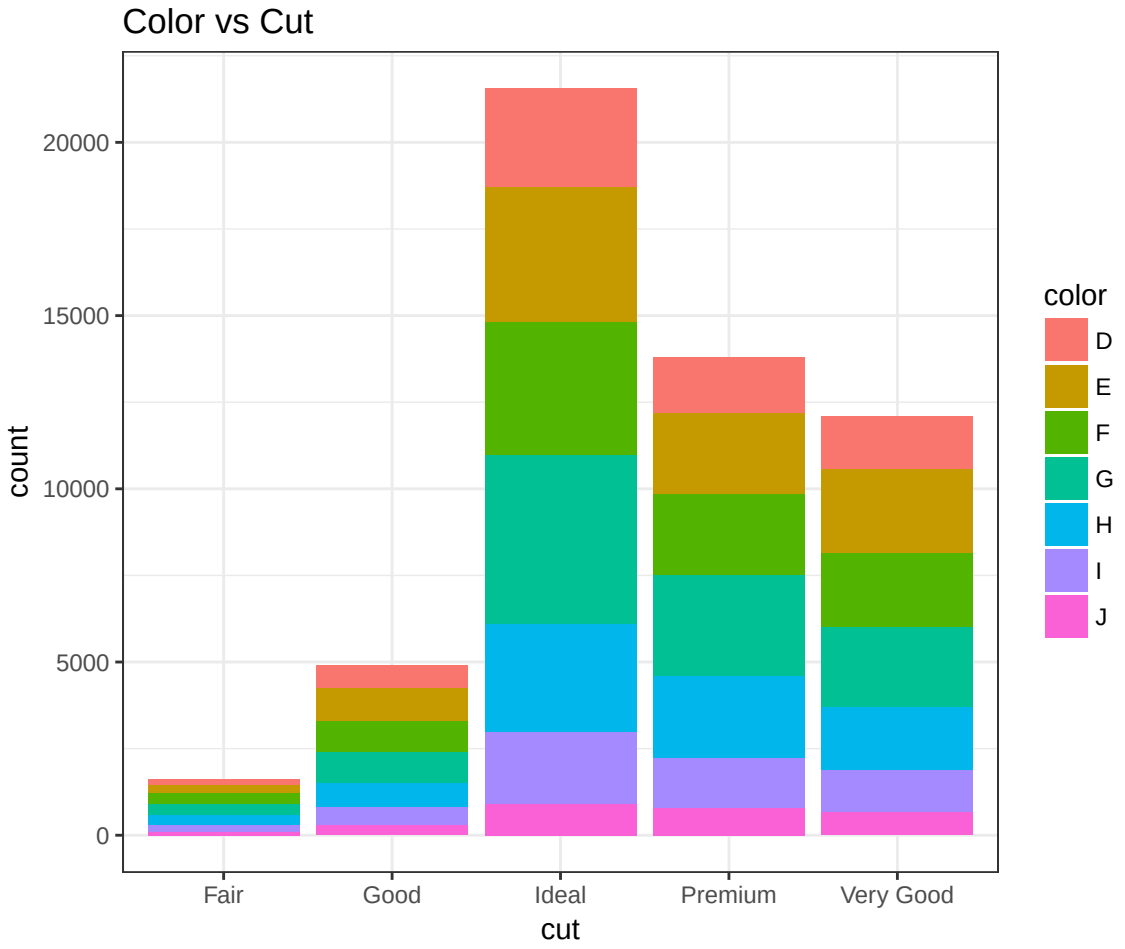
```
for (i in 1:53940) {
    if (diamonds$Cut[i] == 1) {
        dat$cut[i] = "Fair"
    }
    if (diamonds$Cut[i] == 2) {
        dat$cut[i] = "Good"
    }
    if (diamonds$Cut[i] == 3) {
        dat$cut[i] = "Very Good"
    }
    if (diamonds$Cut[i] == 4) {
        dat$cut[i] = "Premium"
    }
    if (diamonds$Cut[i] == 5) {
        dat$cut[i] = "Ideal"
    }

}
```

```
theme_set(theme_bw())
ggplot(dat, aes(x = carat, y = price)) + geom_jitter(aes(color = color,
    shape = cut)) + labs(title = "Carat and Price based on Diamond Cut and Color)")
```



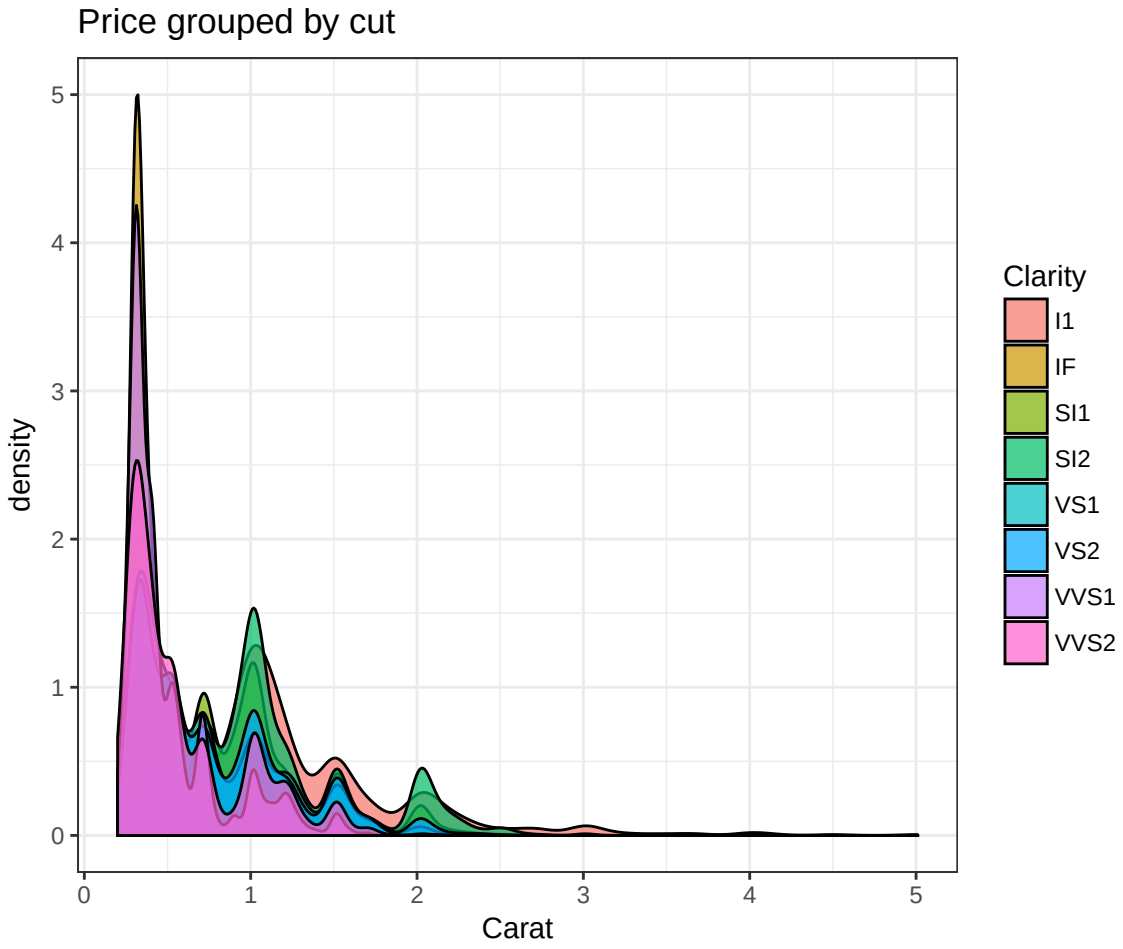
From the above graph, I attempted to see the relationship between 3 of the 4 C's and price. I can see that there is a definite inverse relationship between color and carat, as higher carat values correspond more to lower or worsening colors. This struck me as interesting, and nods towards an assumption that there are indeed attribute tradeoffs when pricing a diamond. We can test this, to see if purchasing a fancier, higher carat diamond would actually give you the best diamond - it seems that you might have to sacrifice one attribute for another. There seems to be an even spread of cut over the price values, although it is hard to tell due to data density. Now that we see a relationship between color and carat, let's explore a relationship between color and cut, keeping in mind the inverse relationship between color and carat:

```
ggplot(dat, aes(x = cut, fill = color)) + geom_bar() + labs(title = "Color vs Cut")
```



This visualization shows that there seems to be a slightly skewed left, normal distribution of cut. Further, within each cut, there seems to be an even spread of color, which may show that there is no clear relationship between cut and color, and therefore little between cut and carat. This reflects what I gathered from the first graph, where there was an even spread of cut. Now let's explore the relationship between carat and clarity:

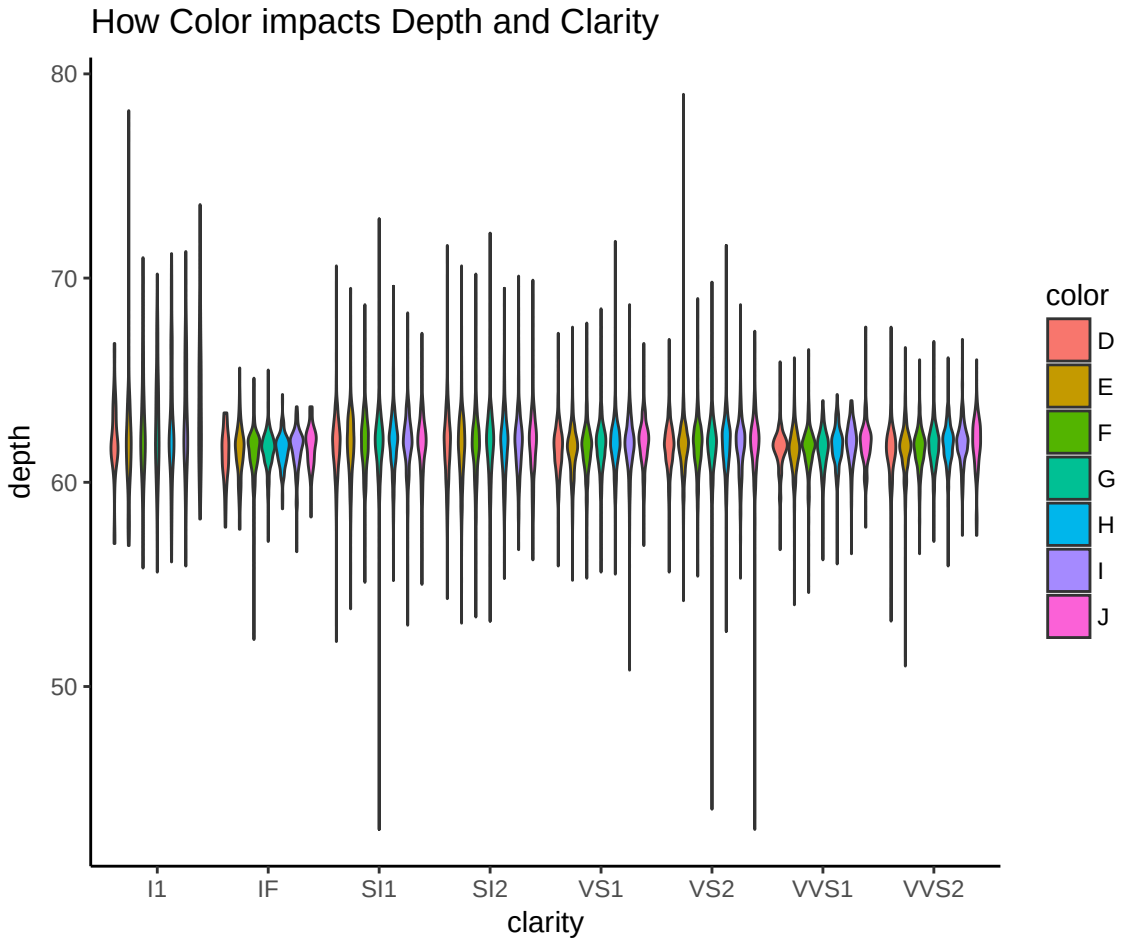
```
ggplot(dat, aes(x = carat)) + geom_density(aes(fill = factor(clarity)),
    alpha = 0.7) + labs(title = "Price grouped by cut", x = "Carat",
    fill = "Clarity")
```



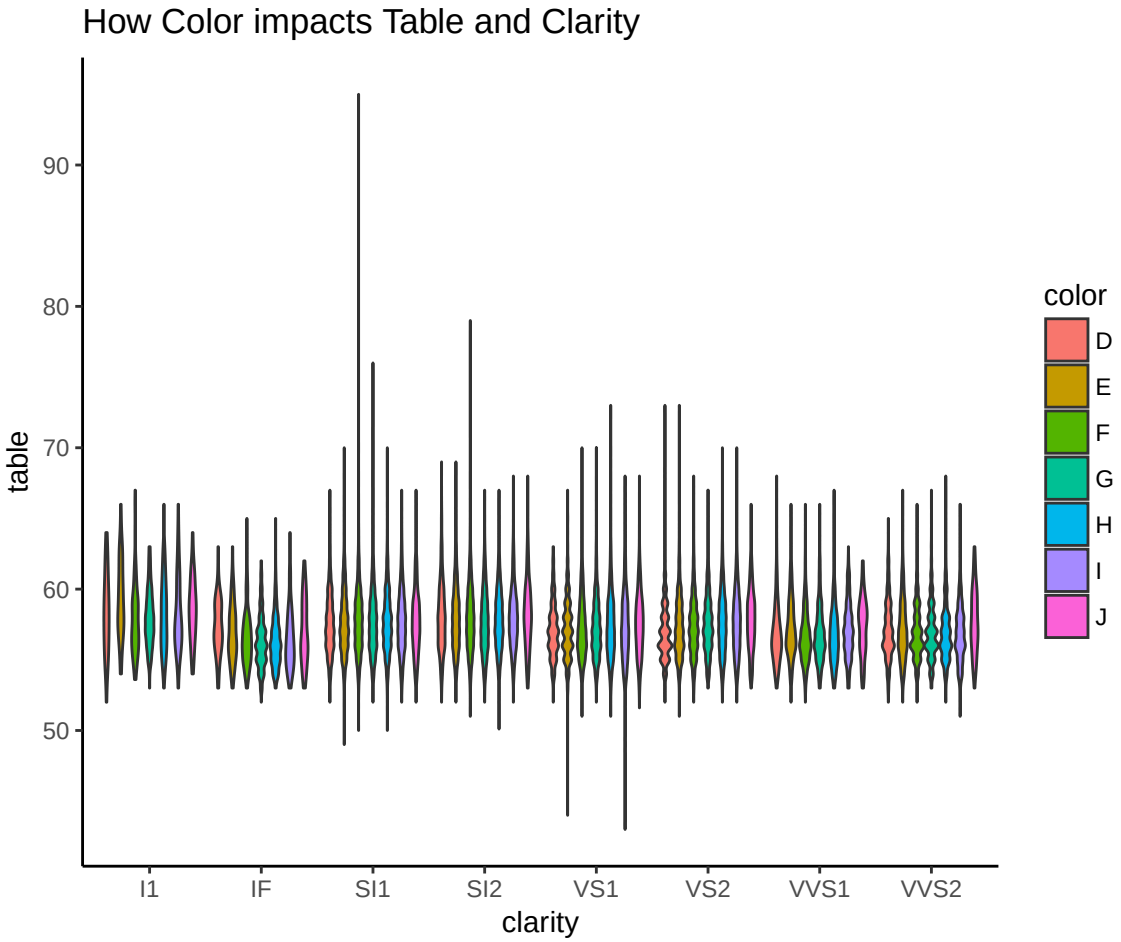
The graph above makes it clear that for the diamonds with the worst clarity, the carat is also lower. However, diamonds with the best clarity seem to have more density around a carat of 1, which is not the highest carat by any means. This allows us to assume that there may be some tradeoff between clarity and carat as well.

Further, I can visualize relationships between some of these attributes and the size. We are not sure how cut affects carat, but there is reason to assume color and clarity may both be tradeoffs. So, let's explore how color impacts clarity and depth in the first graph, as well as table in the second:

```
ggplot(dat, aes(x = clarity, y = depth)) + geom_violin(aes(fill = color)) +
    theme_classic() + labs(title = "How Color impacts Depth and Clarity")
```

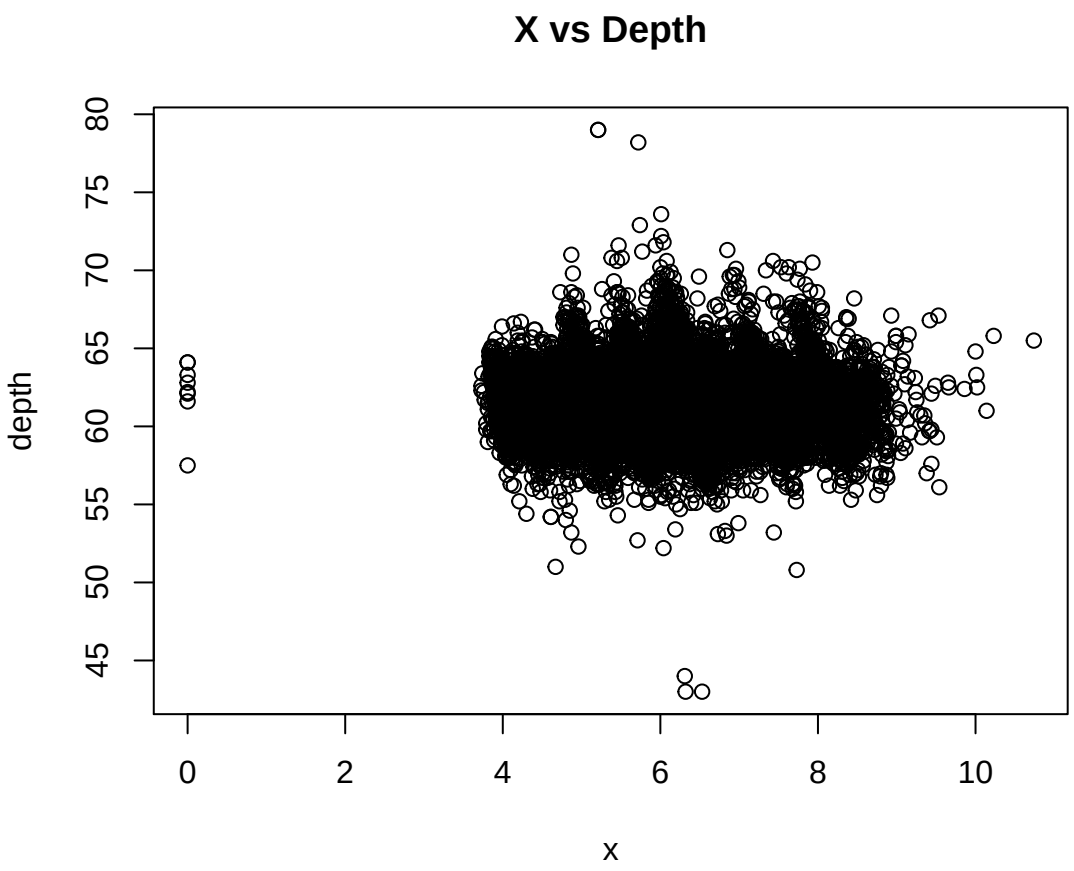



```
ggplot(dat, aes(x = clarity, y = table)) + geom_violin(aes(fill = color)) +
    theme_classic() + labs(title = "How Color impacts Table and Clarity")
```

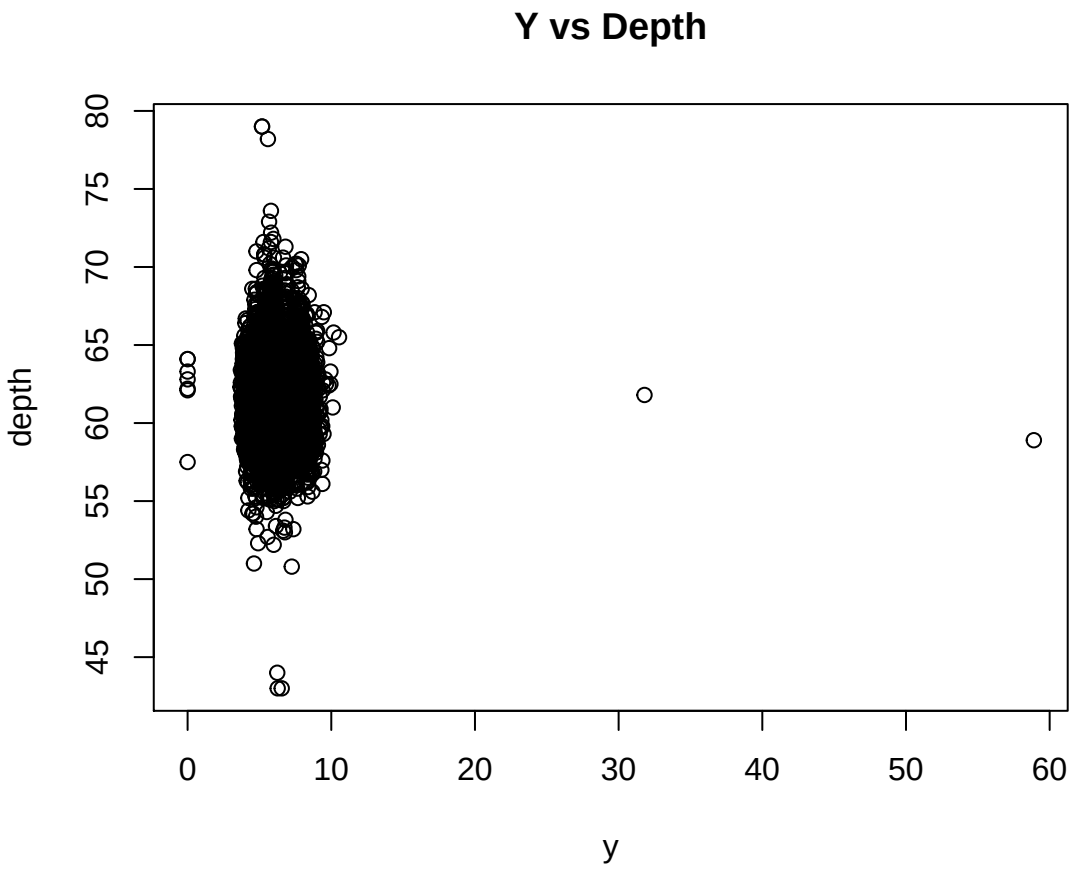


Through some research, I found that the "best" depth for a diamonds lies between 59.5 and 62.9[2], while ideal table varies on the type of diamond, which we do not have information on (princess, etc.). So, depth would seem to be more useful in our data mining tasts. We can see here that the ideal depth does not correspond to the spread of most diamonds in terms of clarity, but perhaps very slightly corresponds to better colors. So, we can assume that the more "ideal" depths of a diamond do in fact are tradeoffs (or perhaps affecting) carat - we should see small but definite fluctuations in carat depending on size measurements of the diamond. Finally, let's break up which measurements affect depth the most - this might be key in seeing which measurements are tradeoffs with carat the most.

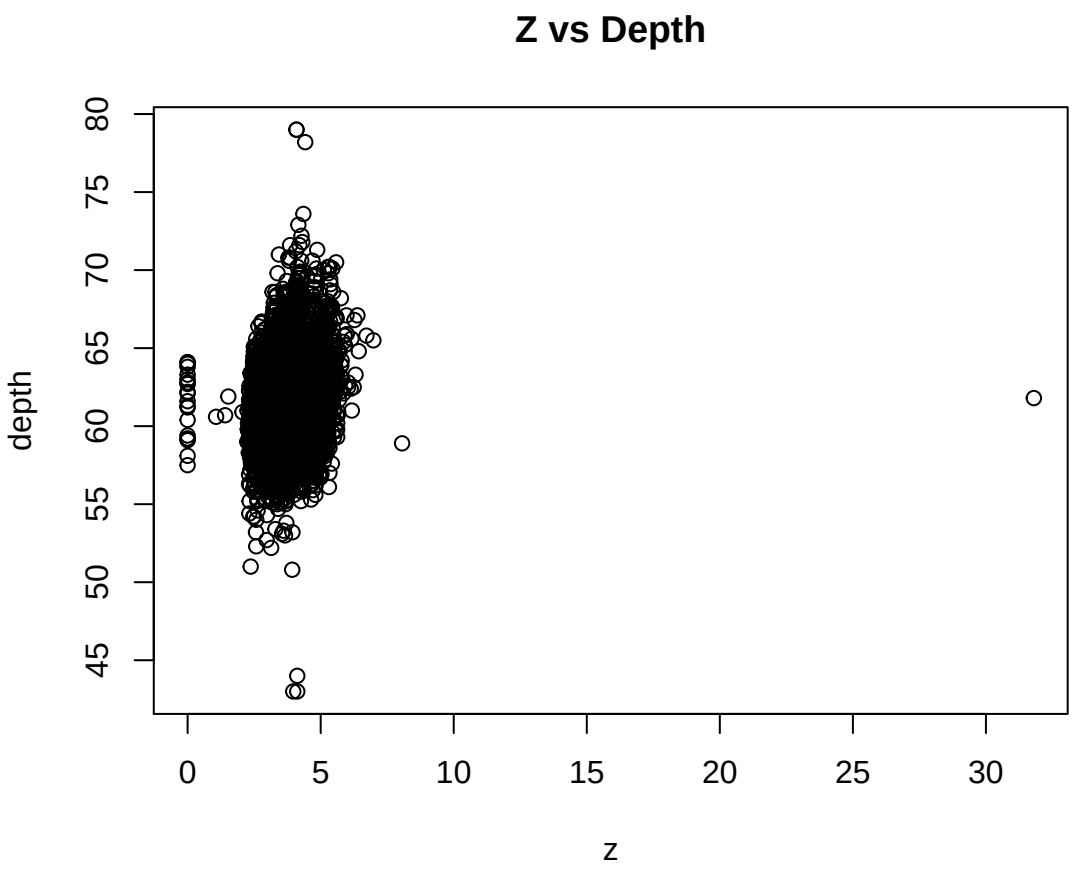
```
plot(depth ~ x, data = dat, main = "X vs Depth")
```




```
plot(depth ~ y, data = dat, main = "Y vs Depth")
```



```
plot(depth ~ z, data = dat, main = "Z vs Depth")
```



From these plots, we can see that y and z correspond much more to variability in depth but little variability in and of themselves. Further, the x does not correspond to much change in depth at all. This leads me to think that depth is perhaps the best indicator of size, as the individual components do not allow for proper analysis due to the manner of spread. This analysis allows me to see which variables I needn't bother with as predictors in my model. At this point, I developed my key questions and observations which I noted above.

Creating a Training and Test Set

Before beginning further analysis via predictive mining, let us split the data into a training and test set.

```
# create a training and test set
set.seed(1)
train = sample(1:nrow(diamonds), 0.75 * dim(diamonds)[1])
diamonds.train = diamonds[train, ]
diamonds.test = diamonds[-train, ]

# Let us preemptively create Price.test, to allow us to store
# the true labels of the test cases
Expensive.test = diamonds.test$Expensive
head(diamonds.test)
```

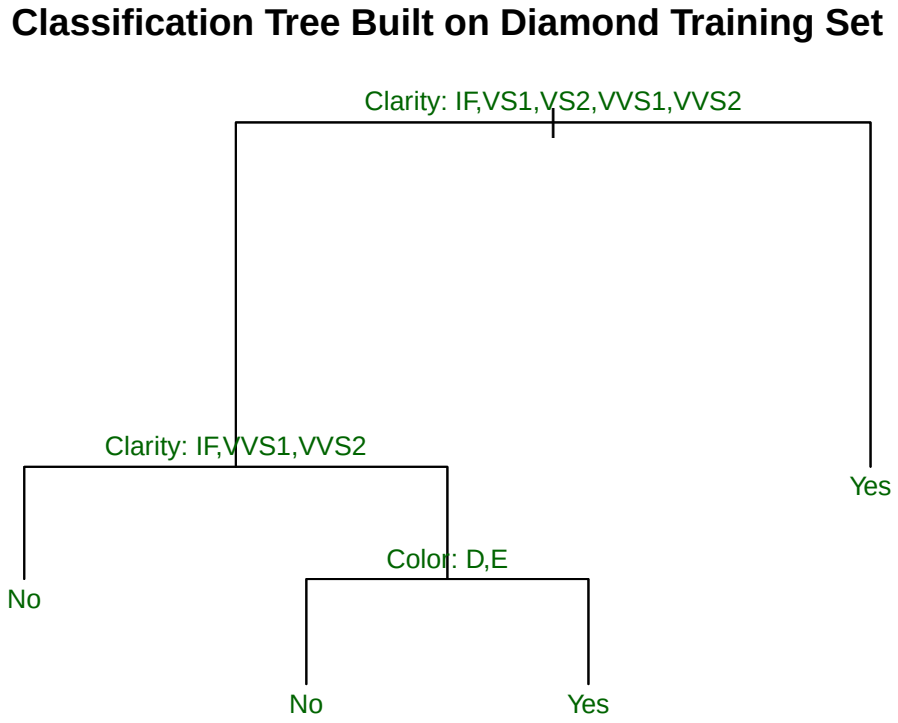
```
##    Carat Cut Color Clarity Depth Table Price    X    Y    Z Expensive
## 2   0.21   4     E     SI1  59.8    61   326 3.89 3.84 2.31        No
## 4   0.29   4     I     VS2  62.4    58   334 4.20 4.23 2.63        No
## 8   0.26   3     H     SI1  61.9    55   337 4.07 4.11 2.53        No
```

```
## 9   0.22   1     E     VS2  65.1    61   337 3.87 3.78 2.49        No
## 13  0.22   4     F     SI1  60.4    61   342 3.88 3.84 2.33        No
## 15  0.20   4     E     SI2  60.2    62   345 3.79 3.75 2.27        No
```

Classification Method: Building a Decision Tree from the Data

I used classification to help answer the question if we could predict the price of the diamond based on its attributes cut, color, and clarity since "Expensive" is a binary valued trait. Further, the outcome of the error in this test could tell us more about specific tradeoffs in acquiring the "best" diamond that were reflected in the graphs. We construct a tree based on the training set of the data:

```
tree.diamonds = tree(Expensive ~ Color + Cut + Clarity, data = diamonds,
    subset = train)
plot(tree.diamonds)
text(tree.diamonds, pretty = 0, cex = 0.8, col = "dark green")
title("Classification Tree Built on Diamond Training Set")
```



We can see from our tree that clarity and color are enough to make assumptions about whether or not a diamond will be expensive (although it is error prone). Our data shows us that clarity may be the most important in determining this, and therefore may be the biggest tradeoff concerning attribute tradeoffs. Now that we have our tree model, we can make predictions on the test set of our data. I have printed out the predictions for the first few entries in the test set.

```
tree.pred = predict(tree.diamonds, diamonds.test, type = "class")
head(tree.pred)
```

```
## [1] Yes Yes Yes No  Yes Yes
## Levels: No Yes
```

To gain insight into whether this classification tree is a good model, I calculated the test error rate by creating a confusion matrix:

```
error = table(tree.pred, Expensive.test)
error
```

```
##          Expensive.test
## tree.pred   No  Yes
##       No  2822 1269
##       Yes 3885 5509
```

```
# Test accuracy rate
(testAccuracyRate = sum(diag(error))/sum(error))
```

```
## [1] 0.6177976
```

```
# Test error rate (Classification Error)
(testErrorRate = 1 - testAccuracyRate)
```

```
## [1] 0.3822024
```

I can see that this approach leads to correct predictions 61.78% of the time - the test error rate is thus around 38.24%. In other words, given the cut, clarity, and color values of a diamond, this classification model can accurately predict if a diamond price is above the median diamond price about 61.78% of the time. I now pruned the tree in the hopes of acquiring a lower test error rate:

```
prune = prune.tree(tree.diamonds, k = 0:10, method = "misclass")
best.prune = prune$size[which.min(prune$dev)]
best.prune
```

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

Now, we see that the best size for the number of nodes to create the smallest error is 4. However, we can see from our tree above that we already have 4 nodes; thus, concerning the relationship between Expensiveness of diamond and the Clarity and Color, this is the most accurate result we will be able to get regardless of how we prune the tree.

Another approach is to perform a k-fold cross validation in order to get the optimal level of tree complexity. Performing a 10-fold cross validation, we get the following results:

```
set.seed(1)
cv = cv.tree(tree.diamonds, FUN = prune.misclass, K = 10)
best.cv = cv$size[which.min(cv$dev)]
best.cv
```

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

Once again, we get 4. Therefore, our original tree for which we ran the error analysis for is our final classification decision tree model for our diamond data.

RegressionModel

```
glm.fit = glm(Expensive ~ Color + Clarity + Cut, data = diamonds.train,
    family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Expensive ~ Color + Clarity + Cut, family = binomial,
##     data = diamonds.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0427  -1.0603  -0.5702   1.0729   1.9697
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.71010    0.10261   6.921 4.50e-12 ***
## ColorE       0.05019    0.03851   1.303 0.192467
## ColorF       0.43152    0.03883  11.113  < 2e-16 ***
## ColorG       0.51903    0.03788  13.701  < 2e-16 ***
## ColorH       0.78585    0.04015  19.572  < 2e-16 ***
## ColorI       0.97658    0.04483  21.782  < 2e-16 ***
## ColorJ       1.05014    0.05586  18.800  < 2e-16 ***
## ClarityIF   -1.79956    0.11406 -15.777  < 2e-16 ***
## ClaritySI1  -0.42209    0.09587  -4.403 1.07e-05 ***
## ClaritySI2   0.33264    0.09750   3.412 0.000646 ***
## ClarityVS1  -0.87134    0.09733  -8.953  < 2e-16 ***
## ClarityVS2  -0.72839    0.09610  -7.580 3.47e-14 ***
## ClarityVVS1 -1.72174    0.10381 -16.585  < 2e-16 ***
## ClarityVVS2 -1.23343    0.10003 -12.330  < 2e-16 ***
## Cut         -0.13903    0.00966 -14.392  < 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: 56082  on 40454  degrees of freedom
## Residual deviance: 51582  on 40440  degrees of freedom
## AIC: 51612
##
## Number of Fisher Scoring iterations: 4
```

```
prob.training = predict(glm.fit, type = "response")
invisible(round(prob.training, digits = 2))
diamonds.train = diamonds.train %>% mutate(predEXPENSIVE = as.factor(ifelse(prob.training <=
    0.5, "No", "Yes")))
table(pred = diamonds.train$predEXPENSIVE, true = diamonds.train$Expensive)
```

```
##      true
## pred     No   Yes
##   No  13236  7436
##   Yes  7016 12767
```

We can interpret our coefficients as such: The variable cut has a coefficient of -0.13903. For every one unit change in Cut, the log odds of being classified as expensive decreases by .13903 holding all other variables fixed. Similarly, we can look at the coefficients for all the other variables associated with different colors and clarities and see how a certain color or clarity would increase or decrease the log odds of being classified as expensive.

Further, from our results we see that out of 40455 cases, the model classifies 26003 correctly (64.27%) out of 20252 inexpensive cases, the model classifies 13234 correctly (65.34%) out of 20203 expensive cases, the model classifies 12767 correctly (63.19%)

Now, to get a better understanding of our model, we can predict whether or not a diamond is going to be expensive based on its cut, color, and clarity and estimate the probabilities. We use the testing data for this. I have pritned out the first ten observations.

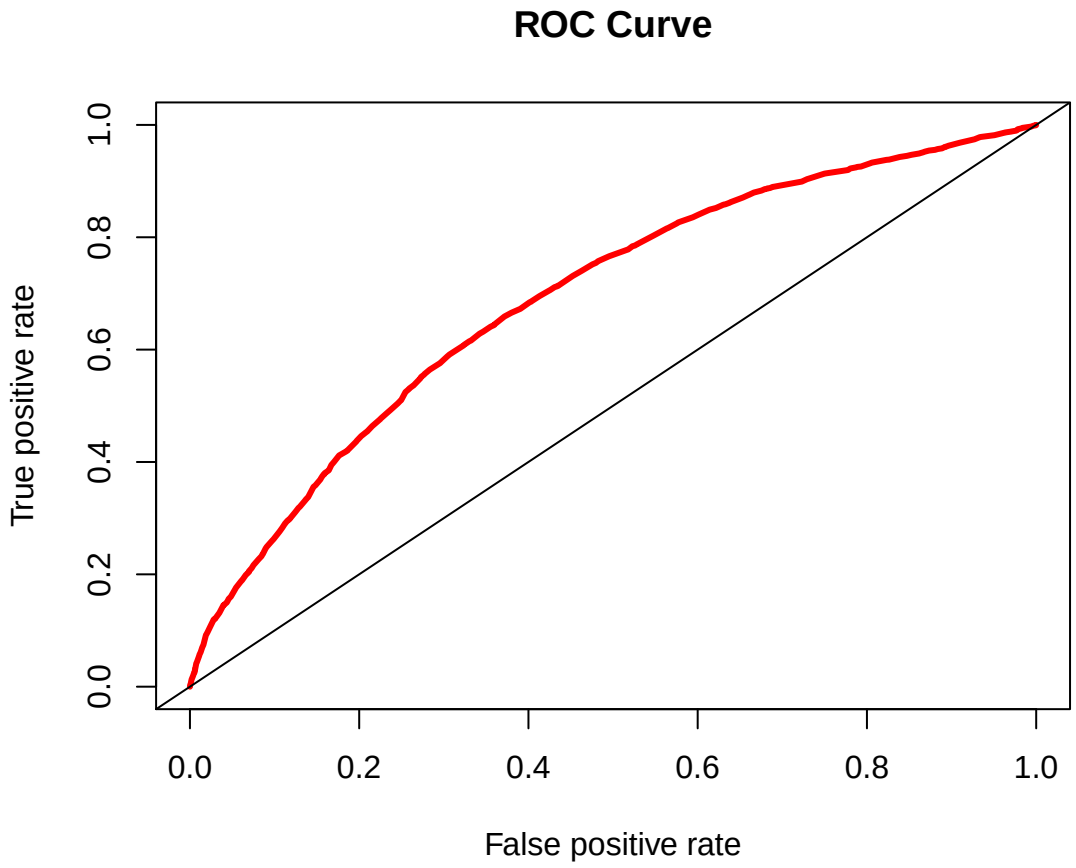
```
prob.test = round(predict(glm.fit, diamonds.test, type = "response"),
    digits = 5)
diamonds.test1 = diamonds.test %>% select(-Expensive) %>% mutate(Probability = prob.test)
diamonds.test1[1:10, ]
```

```
##    Carat Cut Color Clarity Depth Table Price    X    Y    Z Probability
## 1   0.21   4     E     SI1  59.8    61   326 3.89 3.84 2.31     0.44573
## 2   0.29   4     I     VS2  62.4    58   334 4.20 4.23 2.63     0.59920
## 3   0.26   3     H     SI1  61.9    55   337 4.07 4.11 2.53     0.65853
## 4   0.22   1     E     VS2  65.1    61   337 3.87 3.78 2.49     0.47324
## 5   0.22   4     F     SI1  60.4    61   342 3.88 3.84 2.33     0.54076
## 6   0.20   4     E     SI2  60.2    62   345 3.79 3.75 2.27     0.63107
## 7   0.32   4     E      I1  60.9    58   345 4.38 4.42 2.68     0.55086
## 8   0.30   2     J     SI1  63.4    54   351 4.23 4.29 2.70     0.74271
## 9   0.31   3     J     SI1  59.4    62   353 4.39 4.43 2.62     0.71525
## 10  0.30   3     J     VS2  62.2    57   357 4.28 4.30 2.67     0.64902
```

The most glaring observation we can make is that the worse the clarity (J) the more likely a diamond is to be classified as expensive as well. We can see that the mid range clarities are related to less drastic probabilities (around .50) *unless* also associated with better or worse colors. Therefore, both clarity and color are important when determining if a diamond will be expensive or not. Better clarities more often than not correspond to probabilities below .5, as are better colors.

We can now gather some information about the types of errors this model brings us.

```
pred = prediction(prob.training, diamonds.train$Expensive)
perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf, col = 2, lwd = 3, main = "ROC Curve")
abline(0, 1)
```



We can also construct the AUC

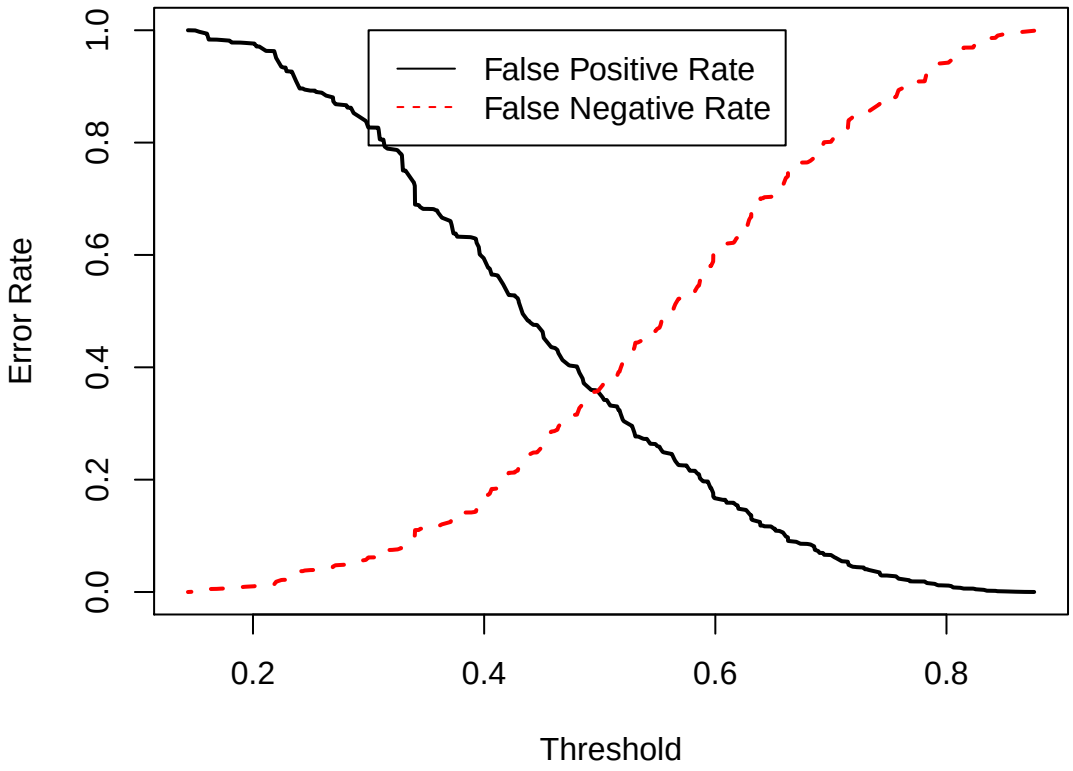
```
auc = performance(pred, "auc")@y.values
auc
```

```
## [[1]]
## [1] 0.6904892
```

We acquire an AUC (area under the curve) level of 0.69. So, about 70% of predicted observations are accurately represented by this model - this, merely by error rate, seems to give us a better predictive model than does the decision tree classifies the observations as expensive or not.

Now, we can determine the best **threshold** value. A threshold provides a cutoff level where below and above is where our binary labeled factor can be determined with the highest level of accuracy. In other words, where do we set the cutoff so above is labelled as "Expensive" and below as "Not Expensive." We used the median earlier, but since linear regression does not work on binary attributes but is suited to continuous attributes, we had to use the model itself to create the cutoff level. The way I do this is by trying to find a probability threshold at which both the false negative rate and false positive rate are BOTH as small as possible - one way we can do this is by calculating the euclidean distance between (False Negative Rate, False Positive Rate) and (0,0). I have calculated the best pair of values for which the euclidean distance is minimized below:

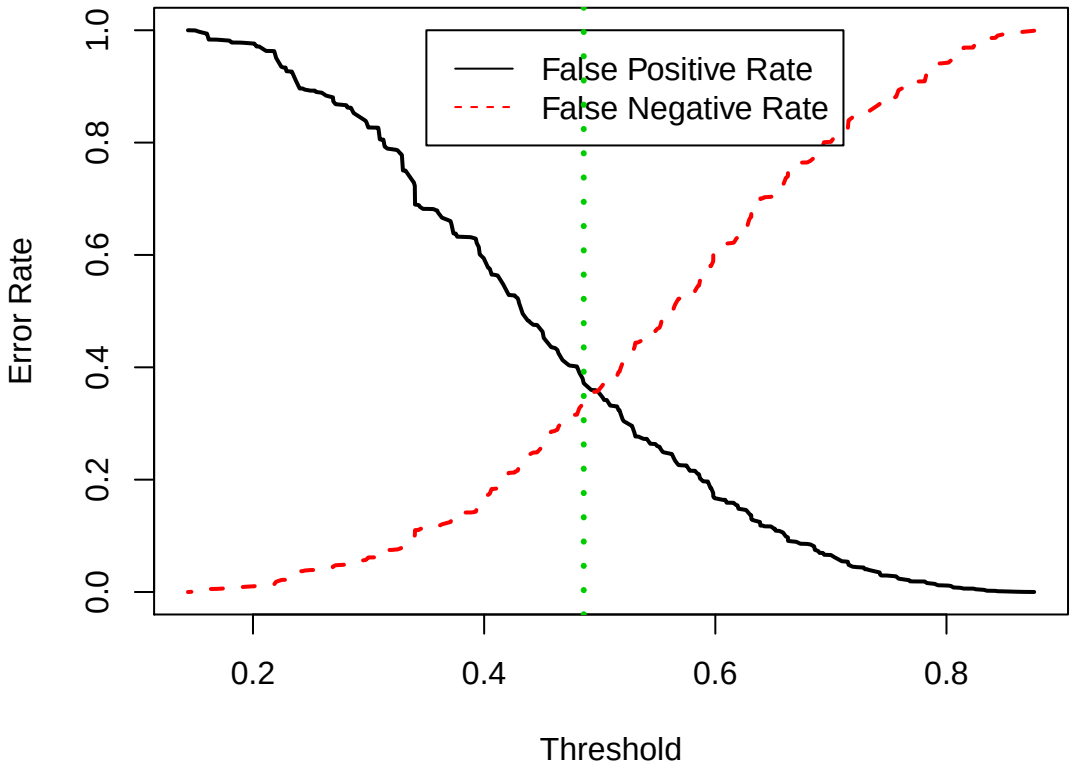
```
fpr = performance(pred, "fpr")@y.values[[1]]
cutoff = performance(pred, "fpr")@x.values[[1]]
fnr = performance(pred, "fnr")@y.values[[1]]
matplot(cutoff, cbind(fpr, fnr), type = "l", lwd = 2, xlab = "Threshold",
    ylab = "Error Rate")  # Add legend to the plot
legend(0.3, 1, legend = c("False Positive Rate", "False Negative Rate"),
    col = c(1, 2), lty = c(1, 2))
```




```
rate = as.data.frame(cbind(Cutoff = cutoff, FPR = fpr, FNR = fnr))
rate$distance = sqrt((rate[, 2])^2 + rate[, 3]^2)
index = which.min(rate$distance)
best = rate$Cutoff[index]
best
```

```
## [1] 0.486153
```

```
matplot(cutoff, cbind(fpr, fnr), type = "l", lwd = 2, xlab = "Threshold",
    ylab = "Error Rate")
legend(0.35, 1, legend = c("False Positive Rate", "False Negative Rate"),
    col = c(1, 2), lty = c(1, 2))
abline(v = best, col = 3, lty = 3, lwd = 3)
```



We get a threshold value of about .4862. This corresponds to the smallest euclidean distance of 0.397. So, according to this threshold value, probabilities less than 0.4862 should be predicted as NO for "Expensive" and probabilities higher than 0.4862 should be predicted as YES for "Expensive."

kNN Classifier Method

Now I will perform a classifying method for k-NN, which is a type of instance-based learning where the function is approximated only locally unlike regression and classification trees. Although we suffer slightly at the curse of dimensionality due to a large number of observations, we first run a simply k-NN through the data as is, and then run k-NN on a much smaller subset of the data to gather a visual representation:

First, we read in the data numerically, and once again mutate the data set to create an "Expensive" factor according to the median value of the price. Then, we once again will split the data in the same train and test set (we can assure this by setting the seed in the same way we did for our previous methods.)

```
dat = read.table("diamondsnumeric.txt", header = TRUE, sep = ",",
    quote = "", fill = TRUE)
diamonds <- data.frame(dat)
diamonds <- diamonds %>% select(-X)
colnames(diamonds) <- c("Carat", "Cut", "Color", "Clarity", "Depth",
    "Table", "Price", "X", "Y", "Z")
diamonds = diamonds %>% mutate(Expensive = as.factor(ifelse(Price <
    median(Price), 0, 1)))

set.seed(1)
train = sample(1:nrow(diamonds), 0.75 * dim(diamonds)[1])
diamonds.train = diamonds[train, ]
diamonds.test = diamonds[-train, ]

# Acquire the train and test split
YTrain = diamonds.train$Expensive
```

```
XTrain = diamonds.train %>% select(-Expensive)
XTrain = XTrain %>% select(Cut, Color, Clarity)
YTest = diamonds.test$Expensive
XTest = diamonds.test %>% select(-Expensive)
XTest = XTest %>% select(Cut, Color, Clarity)
```

Now that we have established our data we can train a kNN classifier on the data. We do this by running the knn function and storing the results in a prediction matrix, from which we develop a confusion matrix to test the error and accuracy rates based on the false positives and false negatives and correct values that we have computed. We do this for both the training and the test set to see if our predicted values are accurately represented by the model.

```
# Train a kNN classifier and calculate error rates
set.seed(444)
pred.YTrain = knn(train = XTrain, test = XTrain, cl = YTrain,
    k = 2)

# Get confusion matrix
(conf.train = table(predicted = pred.YTrain, true = YTrain))
```

```
##          true
## predicted     0     1
##         0 13045  6973
##         1  7207 13230
```

```
# Calculate the test accuracy rate on training set
sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.6494871
```

```
# Calculate the test error rate on training set
1 - sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.3505129
```

```
# Calculate the test error rate - make predictions with TEST
# set
set.seed(555)
pred.YTest = knn(train = XTrain, test = XTest, cl = YTrain, k = 2)

# Get confusion matrix
(conf.test = table(predicted = pred.YTest, true = YTest))
```

```
##          true
## predicted    0    1
##         0 4278 2472
##         1 2429 4306
```

```
# Get test accuracy
sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.6365591
```

```
# Test error rate
1 - sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.3634409
```

The test error obtained by 2-NN classifier is okay, but a relatively high, since 36% of the test observations are incorrectly predicted. However, since our model is so large, any larger number for k is impractical, since leave out one cross validation and k-fold CV are both computationally very expensive for larger numbers of k. However, what this k-NN classifier tell us is that it is relatively accurate to predict the expense factor of a diamond based on the the three attributes of cut, color, and clarity.

Since these processes are computationally very expensive for data sets that have large number of dimensions, in order to get an idea how this would work on a subset of the data and the trends it would show us, we evaluate kNN on a smaller subset of the data. So, on order to get a visual representation of our data and our k-NN method, I take a much smaller portion of the data as the training and test set and repeat the process.

```
dat = read.table("diamondsnumeric.txt", header = TRUE, sep = ",",
    quote = "", fill = TRUE)
diamonds <- data.frame(dat)
diamonds <- diamonds %>% select(-X)
colnames(diamonds) <- c("Carat", "Cut", "Color", "Clarity", "Depth",
    "Table", "Price", "X", "Y", "Z")
diamonds = diamonds %>% mutate(Expensive = as.factor(ifelse(Price <
    median(Price), 0, 1)))
diamonds = diamonds[1:1000, ]

set.seed(1)
train = sample(1:nrow(diamonds), 0.75 * dim(diamonds)[1])
diamonds.train = diamonds[train, ]
diamonds.test = diamonds[-train, ]

# Acquire the train and test split
YTrain = diamonds.train$Expensive
XTrain = diamonds.train %>% select(-Expensive)
XTrain = XTrain %>% select(Cut, Color, Clarity)
YTest = diamonds.test$Expensive
XTest = diamonds.test %>% select(-Expensive)
XTest = XTest %>% select(Cut, Color, Clarity)
# Train a kNN classifier and calculate error rates
set.seed(444)
pred.YTrain = knn(train = XTrain, test = XTrain, cl = YTrain,
    k = 2)

# Get confusion matrix
(conf.train = table(predicted = pred.YTrain, true = YTrain))
```

```
##          true
## predicted   0   1
##         0  38  15
##         1  61 636
```

```
# Calculate the test accuracy rate on training set
sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.8986667
```

```r
# Calculate the test error rate on training set
1 - sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.1013333
```

```r
# Calculate the test error rate - make predictions with TEST
# set
set.seed(555)
pred.YTest = knn(train = XTrain, test = XTest, cl = YTrain, k = 2)

# Get confusion matrix
(conf.test = table(predicted = pred.YTest, true = YTest))
```

```
##          true
## predicted   0   1
##         0  10  10
##         1  41 189
```

```r
# Get test accuracy
sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.796
```

```r
# Test error rate
1 - sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.204
```

This do.chunk method is provided by Professor Sang-Yun Oh from the University of California, Santa Barbara. [3] This function returns a data frame that shows you all the possible values that fold can take, the corresponding training error, and the validation error for each fold. This allows us to select the optimal number of neighbors and the test error rate. We use it to perform a 3-fold cross validation.

```r
do.chunk <- function(chunkid, folddef, Xdat, Ydat, ...) {
    train = (folddef != chunkid)
    Xtr = Xdat[train, ]  # Get training set by the above index
    Ytr = Ydat[train]  # Get true labels in training set
    Xvl = Xdat[!train, ]  # Get validation set
    Yvl = Ydat[!train]  # Get true labels in validation set
    predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, ...)
    predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, ...)
    data.frame(fold = chunkid, train.error = mean(predYtr !=
        Ytr), val.error = mean(predYvl != Yvl))
}

nfold = 3
set.seed(66)
folds = cut(1:nrow(diamonds.train), breaks = nfold, labels = FALSE) %>%
    sample()
error.folds = NULL
allK = 1:50
set.seed(888)
```

```
for (j in allK) {
    tmp = ldply(1:nfold, do.chunk, folddef = folds, Xdat = XTrain,
        Ydat = YTrain, k = j)
    tmp$neighbors = j
    error.folds = rbind(error.folds, tmp)
}
```

At this point, we want the types of errors and the corresponding values from the data. We find the errors for each fold and for each neighbor. Finally, we can see which errors correspond to the optimal number of neighbors, which we find to be 11.

```
errors = melt(error.folds, id.vars = c("fold", "neighbors"),
    value.name = "error")

val.error.means = errors %>% filter(variable == "val.error") %>%
    group_by(neighbors, variable) %>% summarise_each(funs(mean),
    error) %>% ungroup() %>% filter(error == min(error))

numneighbor = max(val.error.means$neighbors)
numneighbor
```

```
## [1] 11
```

Now, we can finally train a 49 Nearest neighbor classifier on our data and calculate the test error rate.

```
set.seed(99)
pred.YTest = knn(train = XTrain, test = XTest, cl = YTrain, k = numneighbor)
conf.matrix = table(predicted = pred.YTest, true = YTest)
sum(diag(conf.matrix)/sum(conf.matrix))
```

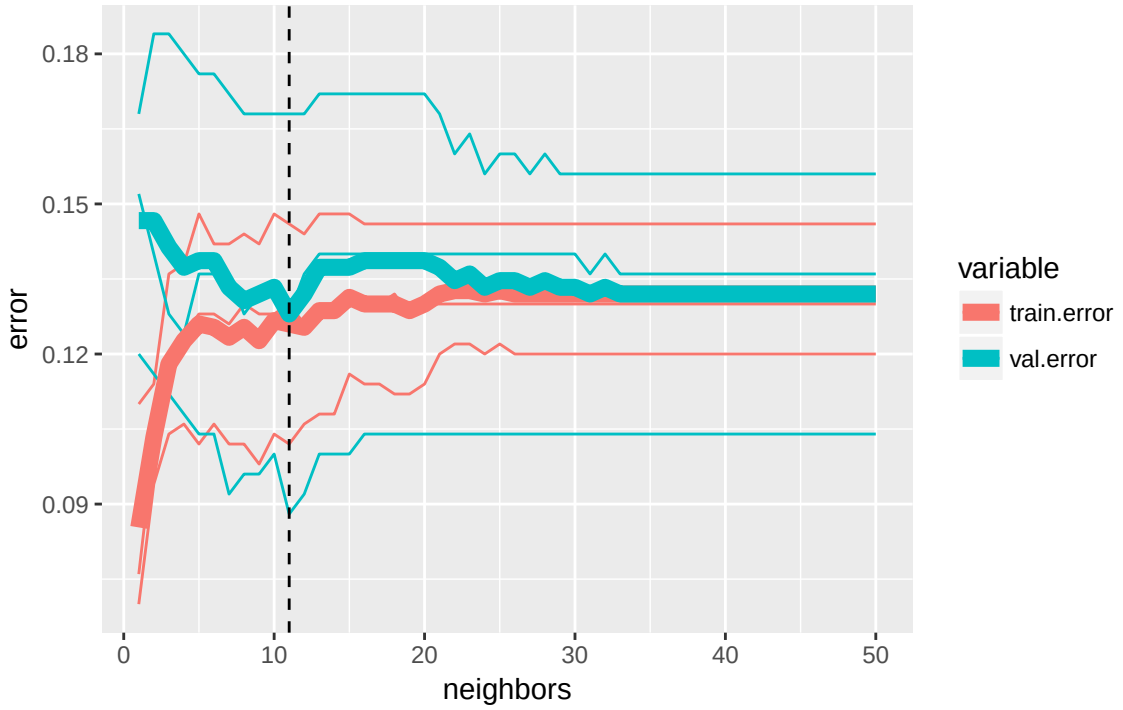
```
## [1] 0.792
```

```
1 - sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.208
```

*NOTE: Our error, while not comparable with our other models due to different data set separarations, is much lower when we do k-fold cross validation. This begs the question that if k-fold cross validation were not such a costly process to perform on data sets that suffer from the curse of dimensionality (read: very large data sets) then perhaps it would be the optimal model. However, we cannot use this error to compare to the other models since it is not a fair comparison across the board.

Now, we can plot our training and validation errors alongside the number of neighbors. As we can see, there is a vertical line at 11, which for our SMALLER data set, we found to be the smallest number of neighbors that corresponded to the minimum of the average validation errors.



How to find the "best" diamond:

Now that we have completed our three data mining processes, we can see that there is indeed a tradeoff between the attributes, but that price can be predicted at some level of accuracy. So, this begs the question - what's the best bang for your buck? Assuming that the 4 C's are what is important when purchasing a diamond, I calculated a "C-Score" for each diamond. I did this by standardizing all the values for the 4 C's, and then adding them together across the diamond. The maximum cscore was stored in a diamond that did NOT have the highest carat, but a carat of 1.07, a cut of 5 (Ideal), a Color of D(best) and a Clarity of IF (Best). So, the other 3 attributes were held more "important"" than the highest carat level, and this corresponded to *not* the most expensive diamond at the end of the day.

```
dat = read.table("diamondsnumeric.txt", header = TRUE, sep = ",",
    quote = "", fill = TRUE)
sdiamonds = scale(dat, center = TRUE, scale = TRUE)
sdiamonds <- sdiamonds[, -1]
sdiamonds <- sdiamonds[, -5:-10]
cscore = rowSums(sdiamonds)
max_cscore = max(cscore)
index = which(cscore == max_cscore)
bestDiamond = (sdiamonds[index, ])
bestDiamond
```

```
##     carat       cut     color   clarity
## 0.5739532 0.9814642 1.5250073 2.3974828
```

*Note: these are the STANDARDIZED values of the carat, cut, color, and clarity of the best diamond. In order to find the true value of these attributes, we must check this index in the data set. We do this in our analysis and conclusion below.

Which is the best model to use, and why?

In order to find out which is the best model, we compare error rates and see if there is a significant difference between the models. We can see that by comparing all the three models, the linear regression is the best model. However, as linear regression treats price as a continuous variable, it is not technically considered a "classification" method in the sense that it does not operate on a discrete resultant variable. If we were being nitpicky, then our classification tree would be the optimal classification model when trying to determine if a certain diamond, depending on its color, cut, and clarity would be deemed as expensive or not.

Conclusion

We *can* predict the price of a diamond based on its attributes. Through predictive modeling, it was found that the most expensive diamonds are not the "best" when trying to optimize all attributes - purchasing the most expensive diamonds will only result in a purchase of a diamond with the highest carat level, but a mediocre color and clarity. Further, our predictive regression model is only correct about 70% of the time. This is due to variation in the other quantitative attributes of the diamond such as depth and table. We also can see that if you hold the 4 C's to be the ultimate teller of "best" diamond, then the most expensive diamonds are NOT the best. These attributes are tradeoffs with the carat level, and the predicted price of the "best" diamond, is one with a carat of 1.07, color of D, a clarity level of IF, and a cut of 5(Ideal). Our best model places this as an "Expensive" diamond, but *not* the most expensive diamond according to our linear regression, which is accurate as our data tells us that the true price of the specific diamond in the records was $17,042.

Figure 2: Diamond.

References:

[1]https://www.kaggle.com/shivam2503/diamonds

[2]https://www.brilliance.com/diamonds/ideal-depth-table-round-cut-diamonds

[3]https://gauchospace.ucsb.edu/courses/mod/folder/view.php?id=827867