Hidden Gems: What makes a diamond precious?

Ammar Hoque

12/10/2019

Introduction:

Diamonds are an often coveted resource that many people love to purchase. These precious gems are cared for because of how valuable they can be. For this report, my Y value, or variable we will analyze, is the price of diamonds in dollars, because people care a lot about selling/buying diamonds. the main goal of this machine learning project is to see what determines the price of a diamond? We are given a dataset of 10 variables of a diamond, so let's interpret it:

For our Y Value, it is the price of diamonds in dollars. Let's see what other variables are present:

```
CSV <- read.csv('diamonds.csv')
names(CSV) # our variables
```

```
##  [1] "X"       "carat"   "cut"     "color"   "clarity" "depth"   "table"
##  [8] "price"   "x"       "y"       "z"
```

```
Data = CSV[ -c(1) ]
```

The X variables are listed below:

carat: The weight, in carats, of the diamond. 1 carat = 200mg

cut: quality of the cut

color: diamond colour. Goes D(best), E, F, G, H, I, J(worst)

clarity: a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x, y, z: The length, width, and depth of the diamond, respectively

depth: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)

table: width of top of diamond relative to widest point (43–95)

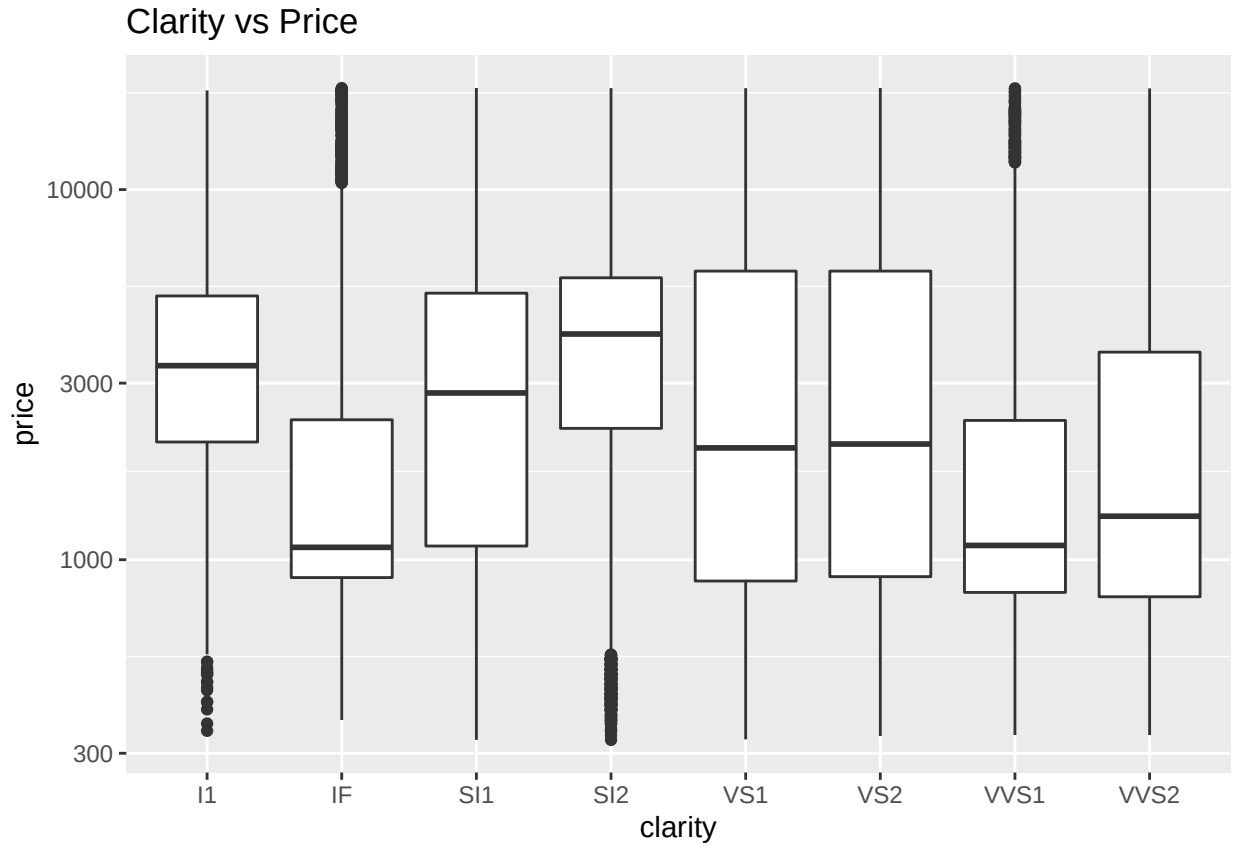
Plots

My outcome (Y) will the the price of the diamonds. The price will depend on the different values, such as 'carat', 'cut', 'clarity', etc. I picked price because most people associate diamonds with making money, so seeing what the diamonds would sell for is the most interesting aspect to me.

Diamonds are typically evaluated with the 4 C's: cut, clarity, color, carat. I will be picking clarity, color, and carat as my 3 variables to analyze with price are 'carat', 'cut', and 'clarity' because that is how people typically determine the price of a diamond

here is a box plot and scatter plot for comparing the diamond clarity with the price:

```
ggplot(Data, aes(x=clarity, y=price)) + geom_boxplot()+ labs(title= "Clarity vs Price") + scale_y_log10
```



It seems that 'I1' has the highest average prices, and 'VVS1' has the lowest. 'IF' has one of the lowest average prices, despite being the most clear. However, if we view the scatter plot that copares the clarity to price, 'I1' is typically bought higher prices, while 'IF' is bought at lower prices. The clarity does appears to not affect the price if the diamond's clarity is 'SL1', 'SL2', 'VS1', 'VS2', and to a lesser extent, 'VVS1' and 'VVS2'.

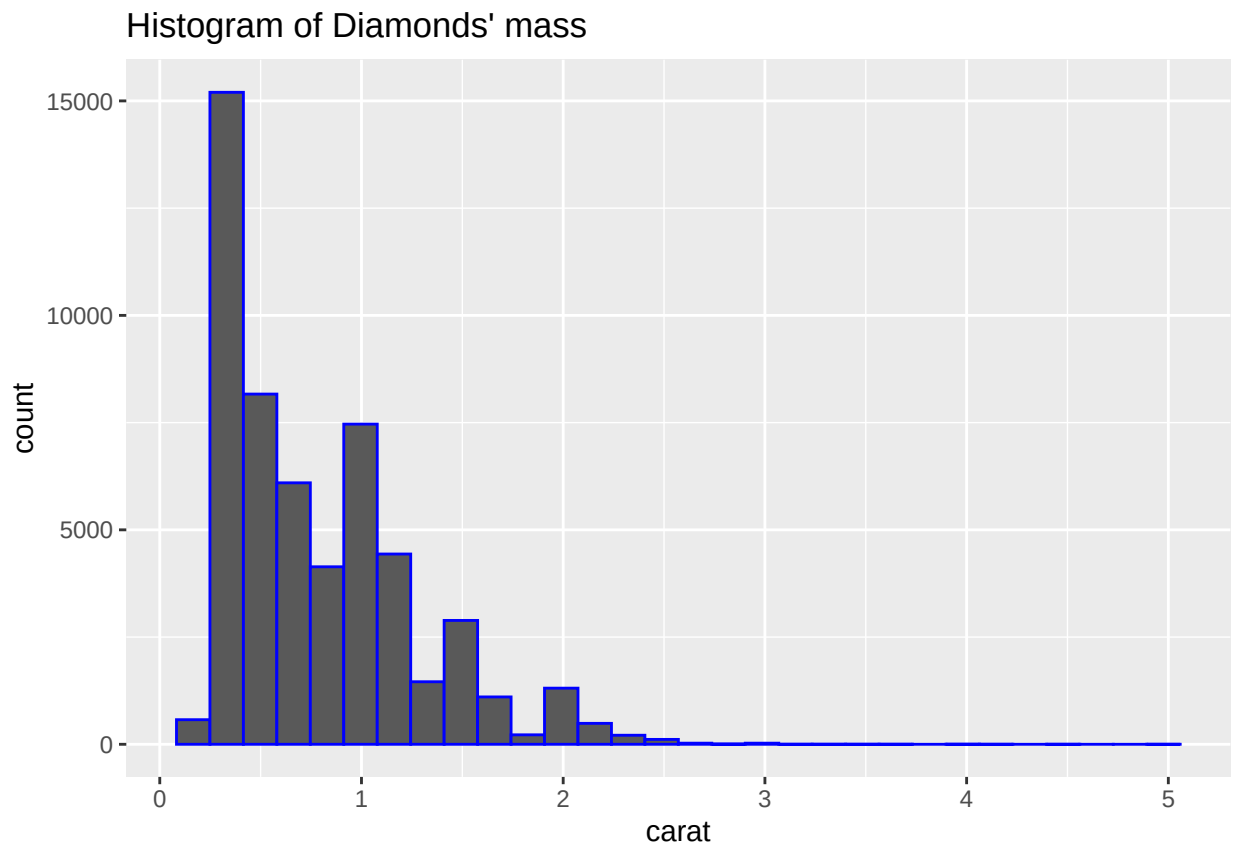
here is a box plot and a scatter plot for comparing diamond color with the price:

```
ggplot(Data, aes(x=color, y=price)) + geom_boxplot() + labs(title= "Color vs Price") + scale_y_log10()
```

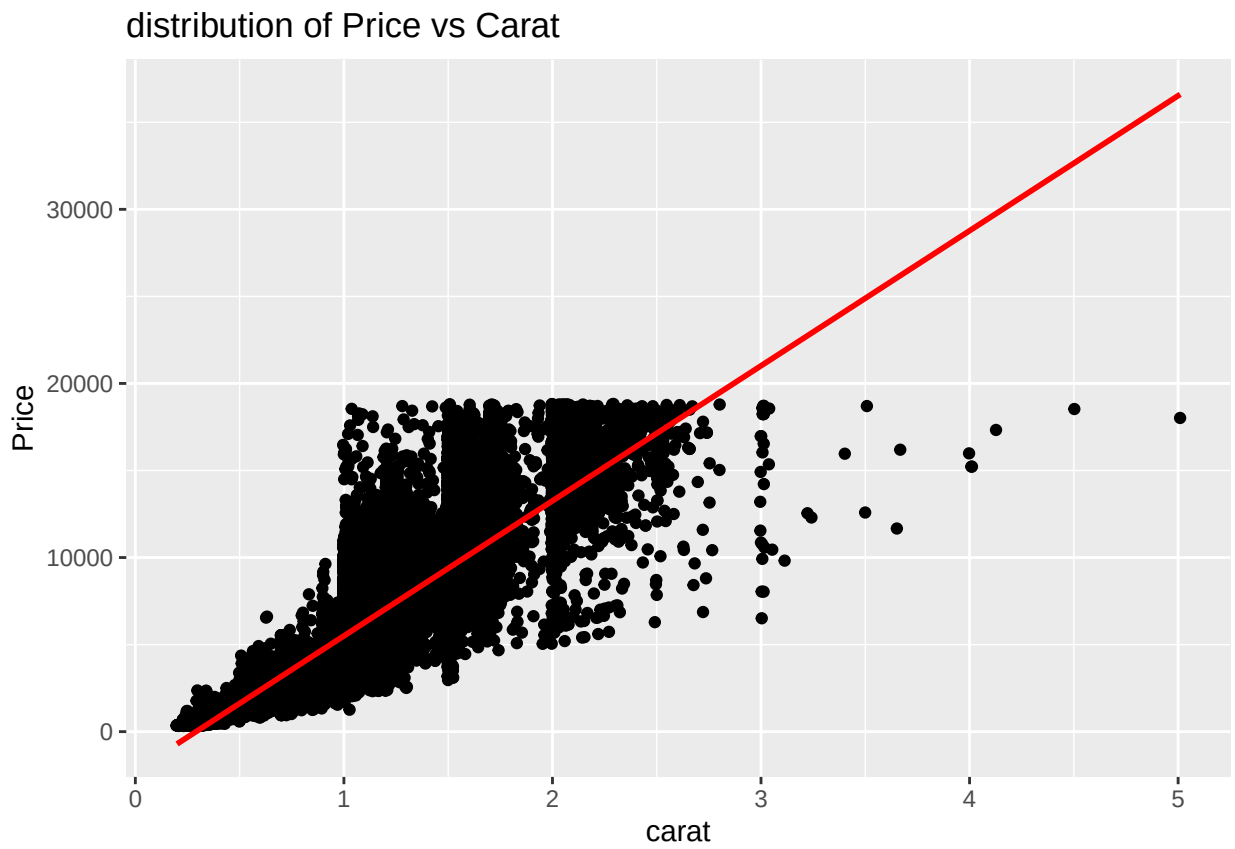
despite the fact D to J goes from best to worst color, it appears the worse-colored diamonds sell for higher on average

here is a histogram to compare the number of diamonds that have a particular weight. I am also including a scatter plot to compare the diamond carat to its price

```
#histogram for carat vs price
ggplot(Data)+
    aes(carat)+
    geom_histogram(bins=30, color="blue")+
    labs(title= "Histogram of Diamonds' mass")+
    xlab("carat")+
    ylab("count")
```




```
ggplot(Data)+
    aes(carat, price)+
    geom_jitter()+
    labs(title= "distribution of Price vs Carat")+
    xlab("carat")+
    ylab("Price")+
    stat_smooth(method = "lm", col = "red")
```



from the histogram, we can see the data is skewed right, since more diamond samples are of lower weight. From the scatter plot, we see a positive correlation between diamond price and its weight. This can be clearly seen with the positive trend line

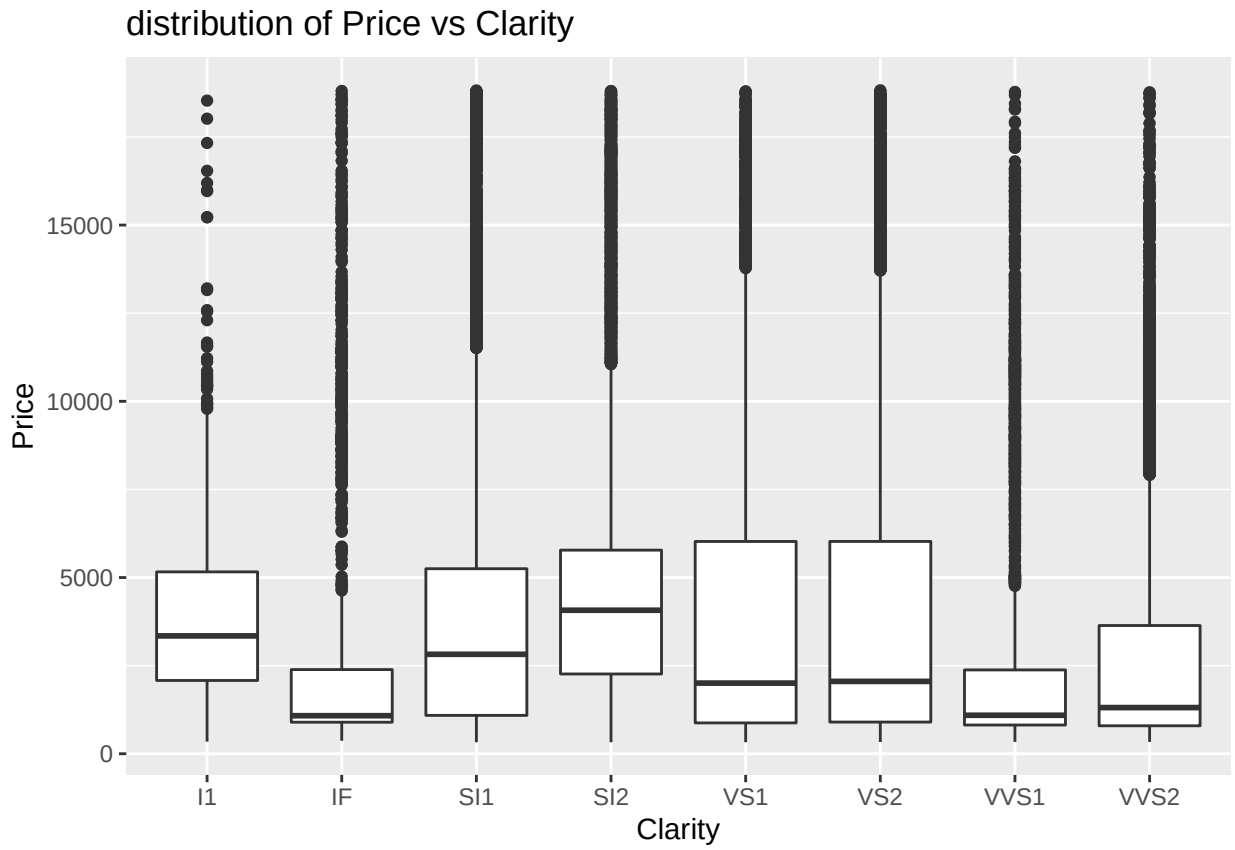
Linear regressions

Simple linear regression

now let's do regressions between price and each of our 3 variables:

Price vs Clarity

```
#scatter plot for the price and clarity

ggplot(Data)+
    aes(clarity, price)+
    geom_boxplot()+
    labs(title= "distribution of Price vs Clarity")+
    xlab("Clarity")+
    ylab("Price")+
    stat_smooth(method = "lm", col = "red")
```




```
#code for regression and getting summary statistics:
linearMod <- lm(price ~ clarity, data=Data)  # build linear regression model on full data
print(linearMod)
```

```
##
## Call:
## lm(formula = price ~ clarity, data = Data)
##
## Coefficients:
```

```
## (Intercept)    clarityIF   claritySI1   claritySI2   clarityVS1   clarityVS2
##   3924.1687   -1059.3296      71.8325    1138.8599     -84.7133       0.8207
## clarityVVS1  clarityVVS2
##  -1401.0541    -640.4316
```

```
summary(linearMod)
```

```
##
## Call:
## lm(formula = price ~ clarity, data = Data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -4737  -2727  -1429   1262  16254
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3924.1687   144.5619  27.145  < 2e-16 ***
## clarityIF   -1059.3296   171.8990  -6.163 7.21e-10 ***
## claritySI1     71.8325   148.6049   0.483    0.629
## claritySI2   1138.8599   150.2746   7.579 3.55e-14 ***
## clarityVS1    -84.7133   150.9746  -0.561    0.575
## clarityVS2      0.8207   148.8672   0.006    0.996
## clarityVVS1 -1401.0541   158.5401  -8.837  < 2e-16 ***
## clarityVVS2  -640.4316   154.7737  -4.138 3.51e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3935 on 53932 degrees of freedom
## Multiple R-squared:  0.02715,    Adjusted R-squared:  0.02702
## F-statistic:   215 on 7 and 53932 DF,  p-value: < 2.2e-16
```

SI1, SI2, and VS2 have a positive effect on the price of the diamonds. SI2 is the only variable who's coefficient has a p-value $< .05$. This means that we reject the null hypothesis that SI2 clarity is insignificant in the correlation with diamond price. SI1 and VS2, however, have p-values $> .05$. We do not reject the null hypothesis that these are insignificant, so these clarities have no effect on price.

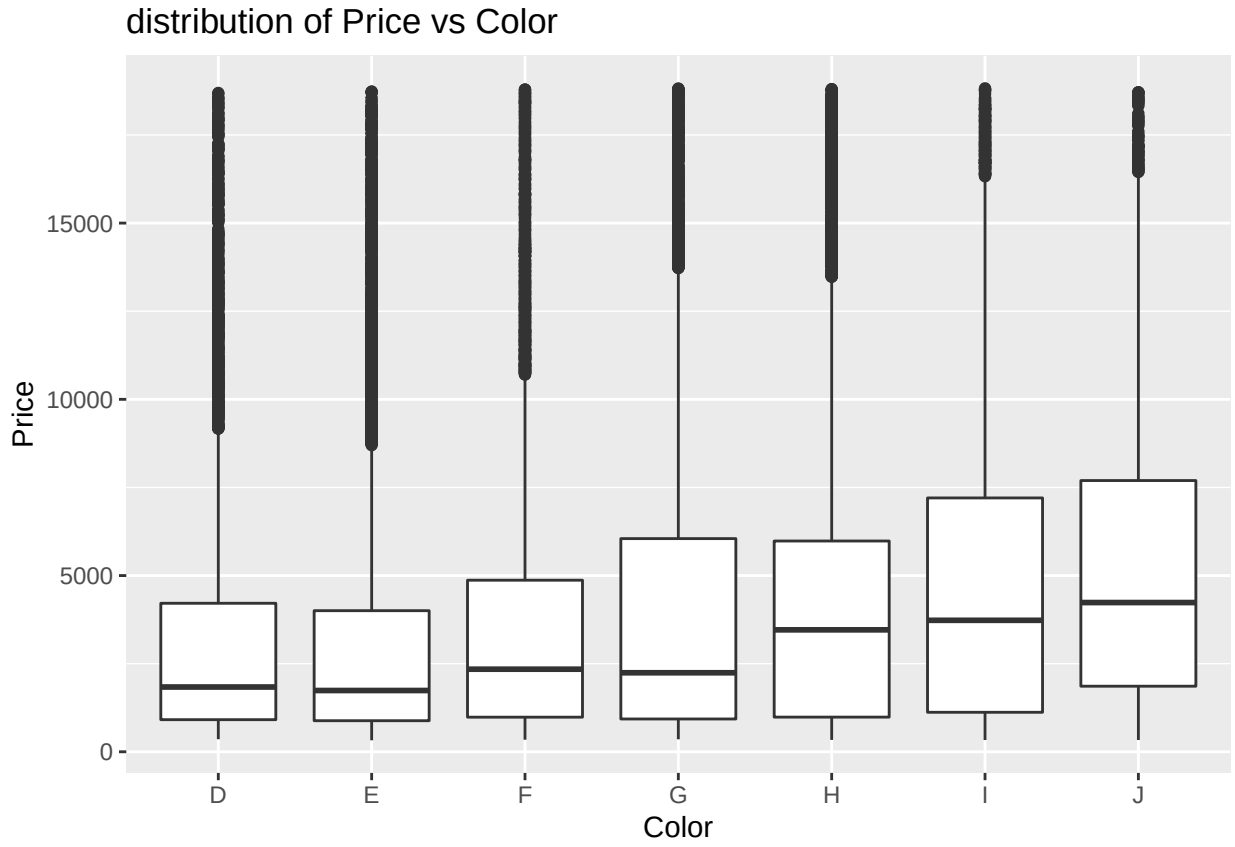
IF, VS1, VVS1, and VVS2 clarities have a negative effect on the price of the diamonds. IF, VVS1, and VVS2 all have p-values $< .05$ so they are not insignificant, while VVS1 has a p-value $> .05$ (Therefore insignificant). This is interesting because these are the higher end clarities, but they give the diamonds a lower price.

When using the F test, it gives a p value that is less than .05, which indicates that the coefficients are altogether not insignificant, or clarity does have an effect on the price of a diamond.

Price vs Color:

Now let's do regressions on color:

```
#scatter plot for price vs color
ggplot(Data)+
    aes(color, price)+
    geom_boxplot()+
    labs(title= "distribution of Price vs Color")+
    xlab("Color")+
    ylab("Price")+
    stat_smooth(method = "lm", col = "red")
```




```
linearMod <- lm(price ~ color, data=Data)  # build linear regression model on full data
print(linearMod)
```

```
##
## Call:
## lm(formula = price ~ color, data = Data)
##
## Coefficients:
## (Intercept)       colorE       colorF       colorG       colorH       colorI
##      3170.0        -93.2        554.9        829.2       1316.7       1921.9
##      colorJ
##      2153.9
```

```
summary(linearMod)
```

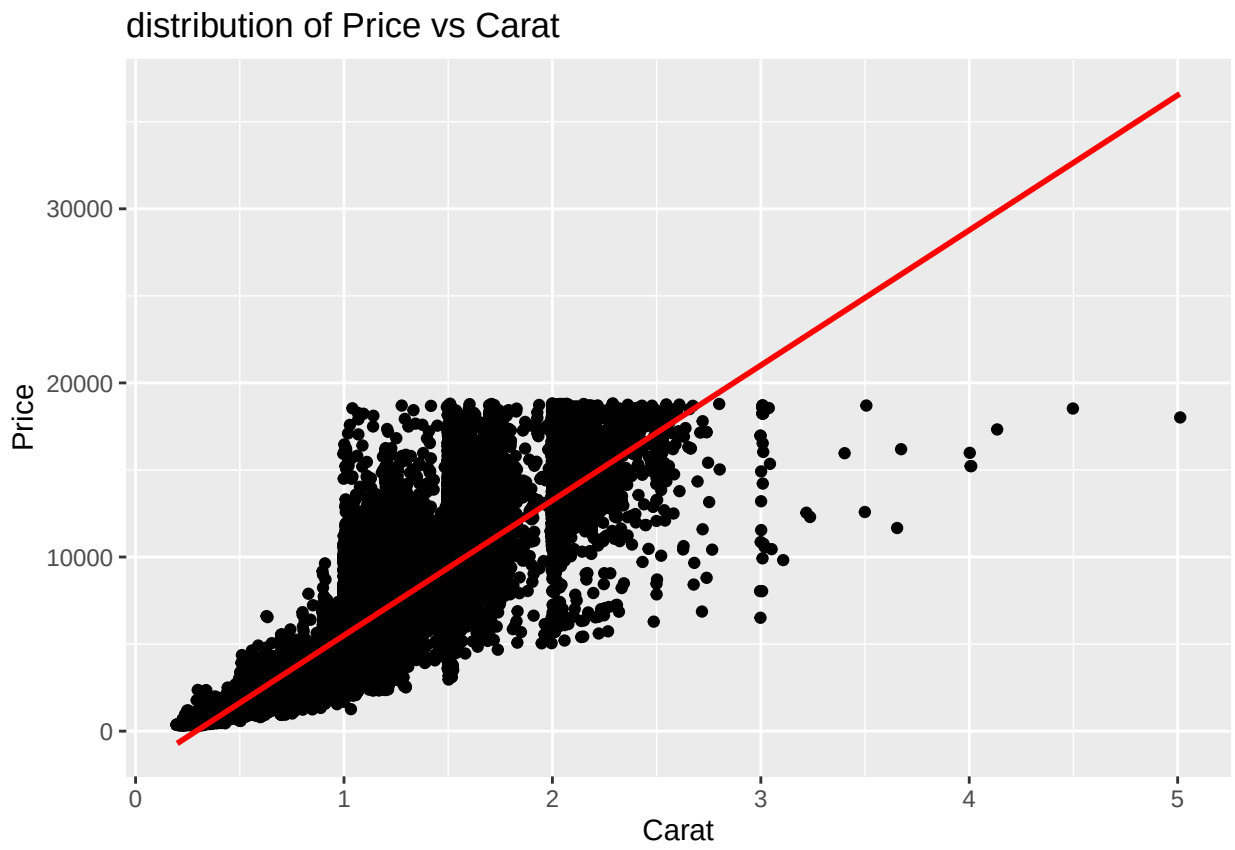
```
##
## Call:
## lm(formula = price ~ color, data = Data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -4989  -2619  -1376   1374  15654
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3169.95      47.71  66.446   <2e-16 ***
## colorE        -93.20      62.05  -1.502    0.133
```

```
## colorF        554.93      62.39   8.895   <2e-16 ***
## colorG        829.18      60.34  13.741   <2e-16 ***
## colorH       1316.72      64.29  20.482   <2e-16 ***
## colorI       1921.92      71.55  26.860   <2e-16 ***
## colorJ       2153.86      88.13  24.439   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3927 on 53933 degrees of freedom
## Multiple R-squared:  0.03128,    Adjusted R-squared:  0.03117
## F-statistic: 290.2 on 6 and 53933 DF,  p-value: < 2.2e-16
```

Having color D has no effect. Having a color of E has a negative effect on price, but it is insignificant because the coefficient of D has a p-value $> .05$. Colors F, G, H, I, and J all have a positive effect on price, and they are all significant because their p-values $< .05$. The F test gives a p-value $< .05$, so the color coefficients have a significant effect on the price of a diamond.

Price vs Carats

```
ggplot(Data)+
    aes(carat, price)+
    geom_jitter()+
    labs(title= "distribution of Price vs Carat")+
    xlab("Carat")+
    ylab("Price")+
    stat_smooth(method = "lm", col = "red")
```



```r
linearMod <- lm(price ~ carat, data=Data)  # build linear regression model on full data
print(linearMod)
```

```
##
## Call:
## lm(formula = price ~ carat, data = Data)
##
## Coefficients:
## (Intercept)        carat
##       -2256         7756
```

```r
summary(linearMod)
```

```
##
## Call:
## lm(formula = price ~ carat, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18585.3   -804.8    -18.9    537.4  12731.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2256.36      13.06  -172.8   <2e-16 ***
## carat        7756.43      14.07   551.4   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1549 on 53938 degrees of freedom
## Multiple R-squared:  0.8493, Adjusted R-squared:  0.8493
## F-statistic: 3.041e+05 on 1 and 53938 DF,  p-value: < 2.2e-16
```

It appears that the carats have a positive effect on the price. An increase by 1 carat increases price by $7756.43 USD. The coefficient of carats has a p-value $< .05$, so we reject the null hypothesis that carats is insignificant. Carats have a significant impact on the price of diamonds

It appears that color, carat, and clarity play a significant part in determining the price of a diamond due from the tests that we have run. However, color and clarity have some insignificant attributes. It seems that in the department of clarity, VVS1 has no effect on determining the price of a diamond. In the department of color, color D has no effect on determining price. Overall, each color and clarity other than the ones mentioned will have some effect on the price, whether it be boosting or reducing the price.

Multiple linear regressions of many x-values

Let's remember what our X values were:

carat: The weight, in carats, of the diamond. 1 carat = 200mg

cut: quality of the cut

color: diamond colour. Goes D,E,F,G,H,I,J

clarity: a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x, y, z: The length, width, and depth of the diamond, respectively

depth: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)

table: width of top of diamond relative to widest point (43–95). This is the upper flat facet of the diamond

Here we will do some multiple linear regressions. Let's start with the 4 C's: carat, cut, color, and clarity. The 4 C's are said to be the "universal method for assessing the quality of any diamond, anywhere in the world (Gemological Institute of America)."

```
#4 C's
multiLinearMod1 <- lm(price ~ clarity + cut + color + carat, data=Data)
```

The adjusted R^2 value is 0.9159. Each of these variables appears to be statistically significant, as the p-val is less than 2.2E-6. I wonder if the 4 C's are the end-all be-all for determining the price of a diamond, or if there are other variables we can use. Let's try the values of x, y, z. These are the length, width, and height of the diamond, which determines its size.

```
#size
multiLinearMod <- lm(price ~ x + y + z, data=Data)
summary(multiLinearMod)
```

```
##
## Call:
## lm(formula = price ~ x + y + z, data = Data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -10958  -1269   -187    975  32147
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -14113.05      41.76 -337.936  < 2e-16 ***
## x             2790.32      40.98   68.082  < 2e-16 ***
## y              218.79      31.56    6.932 4.21e-12 ***
## z              225.91      47.57    4.749 2.05e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1860 on 53936 degrees of freedom
## Multiple R-squared:  0.7825, Adjusted R-squared:  0.7825
## F-statistic: 6.47e+04 on 3 and 53936 DF,  p-value: < 2.2e-16
```

The adjusted R^2 is 0.7825. this is a pretty significant drop-off from the 4 C's. From the p-values, we see that the ceofficients are all significant. I thought that the size variables would determine the price better. Let's try to add an extra parameter with perhaps the most importance out of everything: carat. Carat is typically what most people use to determine the price of a diamond.

```r
#size and weight
multiLinearMod <- lm(price ~ x + y + z + carat, data=Data)
summary(multiLinearMod)
```

```
##
## Call:
## lm(formula = price ~ x + y + z + carat, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23406.8   -647.1    -16.1    358.0  16824.9
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1921.17     104.37  18.407  < 2e-16 ***
## x            -884.21      40.47 -21.848  < 2e-16 ***
## y             166.04      25.86   6.421 1.36e-10 ***
## z            -576.20      39.28 -14.668  < 2e-16 ***
## carat       10233.91      62.94 162.607  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1524 on 53935 degrees of freedom
## Multiple R-squared:  0.8541, Adjusted R-squared:  0.8541
## F-statistic: 7.892e+04 on 4 and 53935 DF,  p-value: < 2.2e-16
```

The adjusted R^2 increased up to 0.8541, and all the variables are significant, according to the p-value of the F-test. Adding carat changed the sign of the coefficients on x and z. The coefficient on carat is the largest, meaning that carat plays a significant factor in the price of diamonds. However, it seems that the x,y,z values are not as significant. This could perhaps be because people like the other features of a diamond, such as how well they can shine, etc.

Let's try another regression with depth, carat, and table. This makes sense because we determine the shine and how much diamond someone is getting with a gem.

```r
#carat, depth, table
multiLinearMod <- lm(price ~ depth + carat + table, data=Data)
summary(multiLinearMod)
```

```
##
## Call:
## lm(formula = price ~ depth + carat + table, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18288.0   -785.9    -33.2    527.2  12486.7
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13003.441    390.918   33.26   <2e-16 ***
## depth        -151.236      4.820  -31.38   <2e-16 ***
## carat        7858.771     14.151  555.36   <2e-16 ***
## table        -104.473      3.141  -33.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1526 on 53936 degrees of freedom
## Multiple R-squared:  0.8537, Adjusted R-squared:  0.8537
## F-statistic: 1.049e+05 on 3 and 53936 DF,  p-value: < 2.2e-16
```

Adjusted R^2 is 0.8537. it is worth noting that dropping either depth or table drops the adjusted R^2 by very little (only changes in the thousandths place). Running this regression with just depth and table gives aadjusted R^2 of 0.017. It appears that depth and table have little to do with price. I find this quite strange because if the table (upper flat facet) is too large then light will not play off of any of the crown's angles or facets and will not create the sparkly rainbow colors that are so beautiful in diamonds. If a diamond's depth percentage is too large or small the diamond will become dark in appearance because it will no longer return an attractive amount of light. Perhaps depth and table are not the biggest factors in price when compared to other variables in the dataset.

From testing 4 variable combinations so far, it seems that the 4 C's and the x,y,z values of the diamond play the most significant roles in determining the price. Let's try and make a linear regression that combines all of them. I will combine the 4 C's and the x,y,z values to see if the size of the diamond also helps determine the price.

```
#4 C's and x, y, z
multiLinearMod <- lm(price ~ clarity + color + carat + cut + x + y + z , data=Data)
summary(multiLinearMod)
```

```
##
## Call:
## lm(formula = price ~ clarity + color + carat + cut + x + y +
##     z, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21132.4   -597.5   -180.5    379.7  10737.9
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3642.49      91.78 -39.686  < 2e-16 ***
## clarityIF     5386.29      51.05 105.520  < 2e-16 ***
## claritySI1    3678.40      43.71  84.152  < 2e-16 ***
## claritySI2    2719.67      43.89  61.967  < 2e-16 ***
## clarityVS1    4603.10      44.60 103.207  < 2e-16 ***
## clarityVS2    4286.45      43.92  97.597  < 2e-16 ***
## clarityVVS1   5039.06      47.20 106.758  < 2e-16 ***
## clarityVVS2   4978.31      45.90 108.449  < 2e-16 ***
## colorE        -209.02      17.93 -11.659  < 2e-16 ***
## colorF        -273.60      18.13 -15.093  < 2e-16 ***
## colorG        -486.19      17.74 -27.401  < 2e-16 ***
## colorH        -986.08      18.87 -52.269  < 2e-16 ***
## colorI       -1472.11      21.20 -69.440  < 2e-16 ***
## colorJ       -2374.88      26.18 -90.714  < 2e-16 ***
## carat        11129.29      47.71 233.255  < 2e-16 ***
## cutGood        671.05      33.08  20.287  < 2e-16 ***
## cutIdeal      1027.67      30.29  33.925  < 2e-16 ***
## cutPremium     907.86      30.71  29.564  < 2e-16 ***
## cutVery Good   866.79      30.86  28.088  < 2e-16 ***
## x             -869.65      30.62 -28.403  < 2e-16 ***
## y               32.20      19.30   1.669   0.0952 .
## z             -227.70      29.72  -7.661 1.87e-14 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1132 on 53918 degrees of freedom
## Multiple R-squared:  0.9195, Adjusted R-squared:  0.9194
## F-statistic: 2.931e+04 on 21 and 53918 DF,  p-value: < 2.2e-16
```

Adjusted R^2 is 0.9194. This is an increase by 0.0035 from the model with 4 C's. However, according to the summary, it seems that y is insignificant (its p-value is 0.0952. This is greater than .05) in determining the price, so I will remove that and see how price changes.

```
#4 C's and x, z
multiLinearMod <- lm(price ~ clarity + color + carat + cut + x + z , data=Data)
summary(multiLinearMod)
```

```
##
## Call:
## lm(formula = price ~ clarity + color + carat + cut + x + z, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21139.0   -597.5   -180.2    379.8  10737.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3643.92      91.78 -39.702  < 2e-16 ***
## clarityIF     5387.77      51.04 105.563  < 2e-16 ***
## claritySI1    3679.42      43.71  84.182  < 2e-16 ***
## claritySI2    2720.76      43.88  61.998  < 2e-16 ***
## clarityVS1    4604.38      44.59 103.250  < 2e-16 ***
## clarityVS2    4287.50      43.92  97.629  < 2e-16 ***
## clarityVVS1   5040.30      47.20 106.796  < 2e-16 ***
## clarityVVS2   4979.56      45.90 108.488  < 2e-16 ***
## colorE        -208.95      17.93 -11.655  < 2e-16 ***
## colorF        -273.56      18.13 -15.091  < 2e-16 ***
## colorG        -486.25      17.74 -27.404  < 2e-16 ***
## colorH        -986.09      18.87 -52.269  < 2e-16 ***
## colorI       -1472.22      21.20 -69.444  < 2e-16 ***
## colorJ       -2374.99      26.18 -90.716  < 2e-16 ***
## carat        11130.45      47.71 233.300  < 2e-16 ***
## cutGood        673.85      33.04  20.398  < 2e-16 ***
## cutIdeal      1030.58      30.24  34.077  < 2e-16 ***
## cutPremium     909.74      30.69  29.645  < 2e-16 ***
## cutVery Good   870.21      30.79  28.260  < 2e-16 ***
## x             -841.19      25.43 -33.079  < 2e-16 ***
## z             -222.53      29.56  -7.528 5.23e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1132 on 53919 degrees of freedom
## Multiple R-squared:  0.9195, Adjusted R-squared:  0.9194
## F-statistic: 3.078e+04 on 20 and 53919 DF,  p-value: < 2.2e-16
```

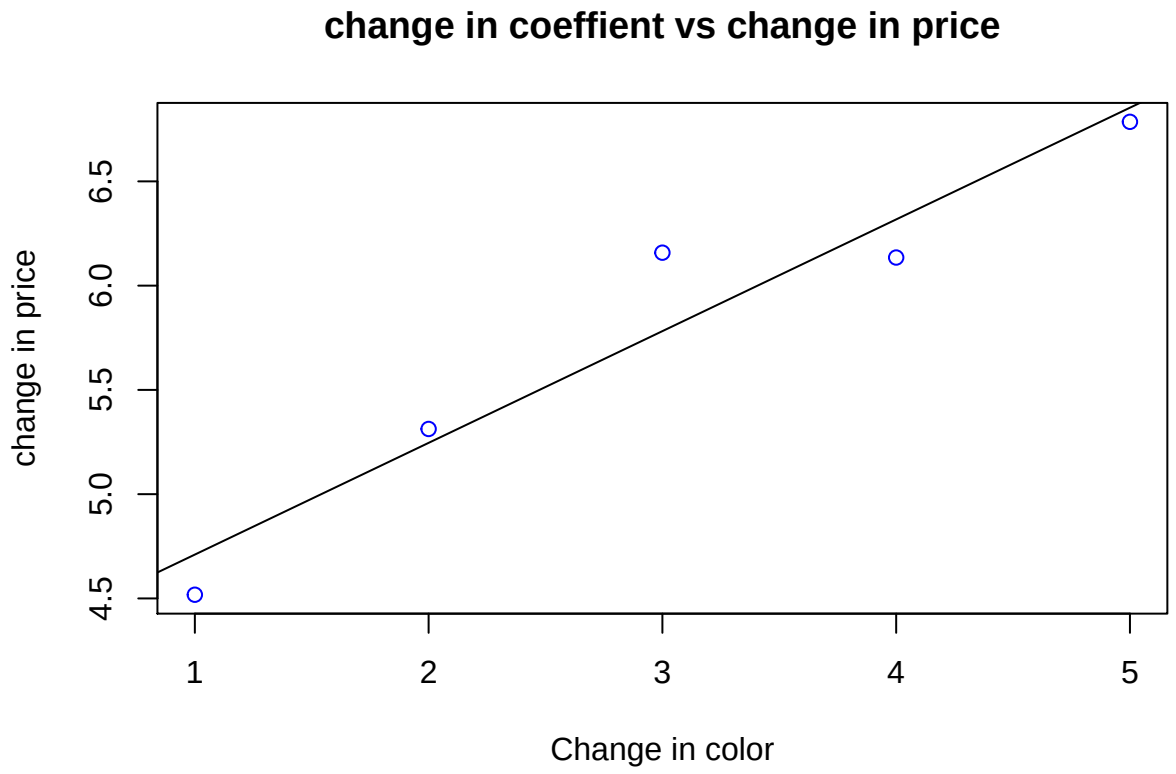
```
#this is the model I will go with
```

the adjusted R^2 value kept the same, despite dropping a parameter. I believe that this is the optimal model.

Interpreting the coefficients of the model

Categorical Variables For the categorical variables (clarity, color, cut), these parameters hold dummy variables (either 0 or 1) for each of their elements. All of these coefficients have $p < 0.5$. This means that all coefficients are significant. for color, changing from E to F creates a decrease in the price by \$91.63. As the color grade keeps getting worse, the decrease in price decreases:

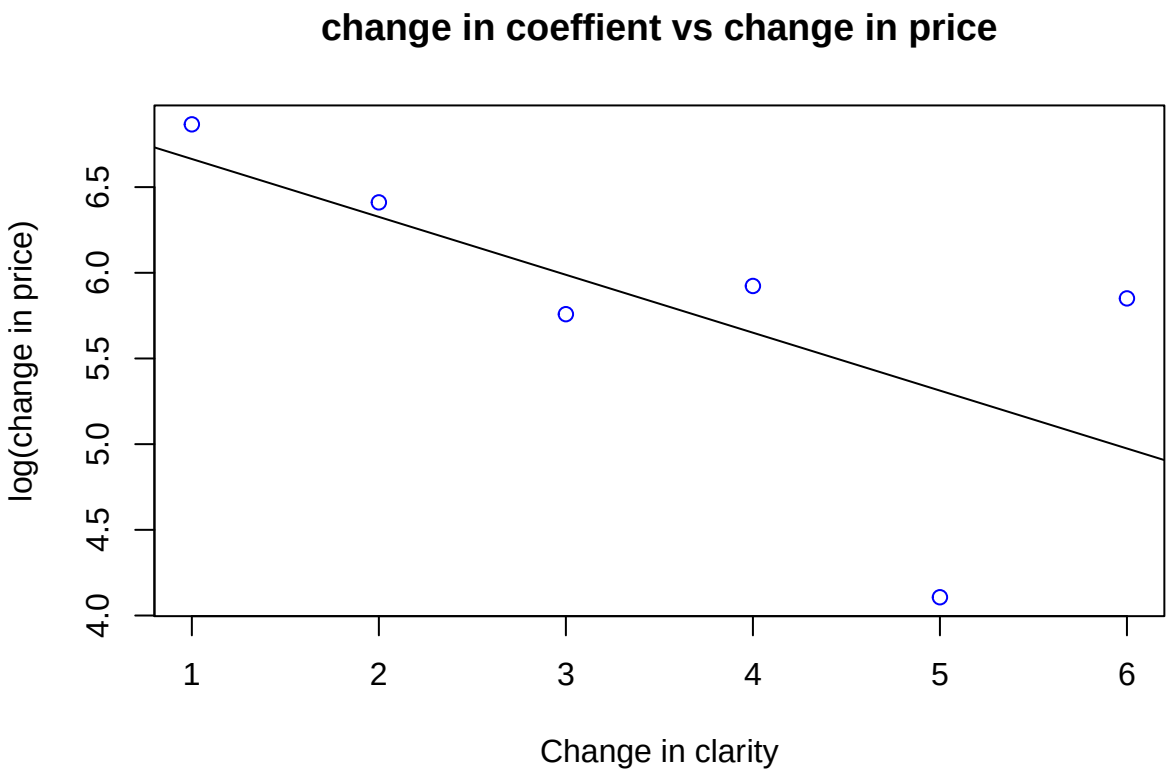
```
valueVec <- c(log(91.63), log(202.89), log(472.5), log(461.6), log(884.92))
variableVec <- c(1, 2, 3, 4, 5)
myline.fit <- lm(valueVec ~ variableVec)

plot(x=variableVec, y=valueVec, main= "change in coeffient vs change in price", xlab="Change in color",
abline(myline.fit)
```



let the elements of 1, 2, 3, 4, 5 be mapped respectively to the vector E-F, F-G, G-H, H-I, I-J from the graph, change in color keeps decreasing the price at a non-linear rate. I took the log and it was more linear.

for clarity, changing to a better clarity increases price:

```
valueVec <- c(log(959.42), log(608.5), log(316.88), log(373.56), log(60.74), log(347.47)) # these are t

variableVec <- c(1, 2, 3, 4, 5, 6)

myline.fit <- lm(valueVec ~ variableVec)

plot(x=variableVec, y=valueVec, main= "change in coeffient vs change in price", xlab="Change in clarity
abline(myline.fit)
```



let the elements of 1, 2, 3, 4, 5, 6 be mapped respectively to the vector 'r c("SI2-SI1", "SI1-VS2", "VS2-VS1", "VS1-VVS2", "VVS2-VVS1", "VVS1-IF") change in price due to clarity keeps decreasing, other than changing from VVS1-IF. It seems that changes in clarity change thhe price exponentially aswell.

for cut, (where it goes from worst to best: Good, Very Good, Premium, Ideal), the change in price due to change cut quality is inconsistent. moving from good to very good gives an increase of 196.36 USD, very good to premium gives 39.53 USD increase, and change from premium to ideal gives an increase 120.84 USD.

Continuous Variables

For continuous variables (x, z, carat) these coefficients mean that an increase in their value by 1 gives and increase equivalent to their coefficient.

For x, its coefficient is -841.19. Increase x by 1 and the price of the diamond decreases by 841.19 USD.

For z, its coefficient is -222.53. Increase z by 1 and the price decreases by 222.53 USD. This is not as big of a decrease when compared to x. it appears the value of width has a greater impact than the height.

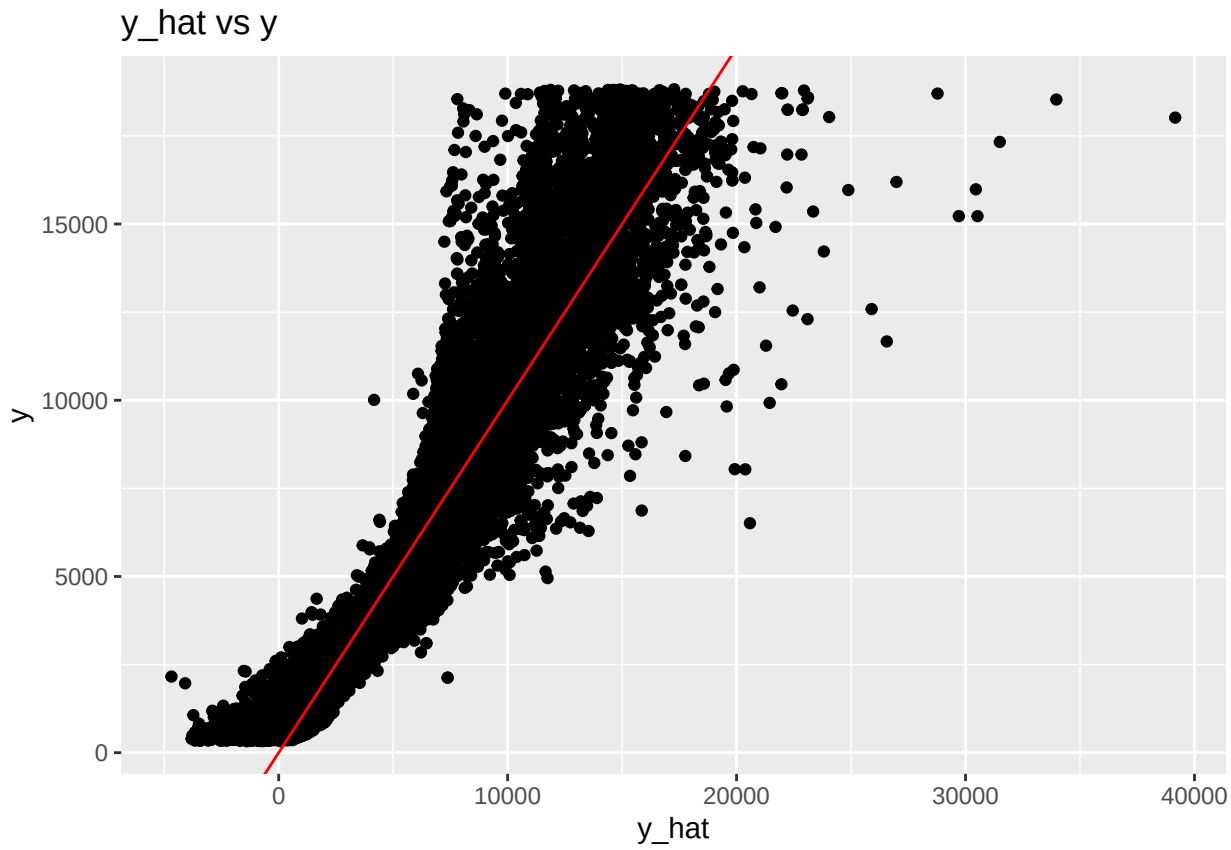
For carat, its coefficient is 1130.45. Increase a diamond's carat by 1 and the price increases by 1130.45 USD. This is the biggest coefficient out of all the coefficients in our regression. This is perhaps the most important variable for determining price of a diamond. The F-stat gives us a p-value that is less than .05. This means all the coefficients are significant in the model.

Y vs Yhat

Here is a plot for y_hat vs y:

```
yhat <- lm(price ~ clarity + color + carat + cut + x + z , data=Data)
y = Data$price
```

```
ggplot(Data, aes(x = fitted(yhat), y = Data$price)) +
    labs(title= "y_hat vs y")+
    xlab("y_hat")+
    ylab("y")+
    geom_point() +
    geom_abline(slope=1, intercept=0, col="red") #this is where y_hat=y on the graph
```



the bulk of the points lie very close to the line y=x. This means that our model (y_hat) is pretty close to the actual value of y (price). However, at higher values of price, our model seems to fall off.

Coefficients vs standard errors

Here is our summary of the coefficients:

```
summary(yhat)$coefficients
```

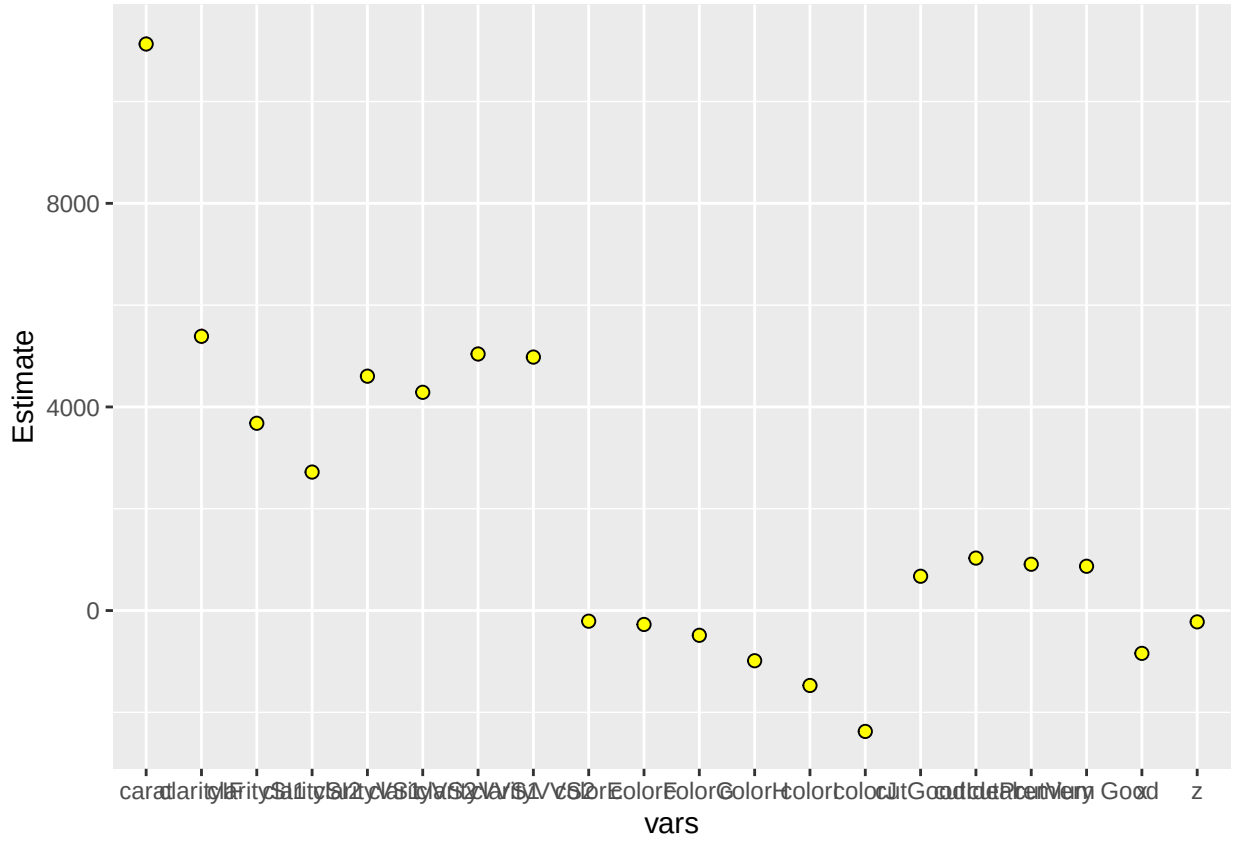
```
##                Estimate Std. Error    t value      Pr(>|t|)
## (Intercept)  -3643.9195   91.78137 -39.702170  0.000000e+00
## clarityIF     5387.7699   51.03833 105.563211  0.000000e+00
## claritySI1    3679.4178   43.70802  84.181761  0.000000e+00
## claritySI2    2720.7565   43.88475  61.997760  0.000000e+00
## clarityVS1    4604.3826   44.59471 103.249521  0.000000e+00
## clarityVS2    4287.5027   43.91610  97.629397  0.000000e+00
## clarityVVS1   5040.2952   47.19547 106.796176  0.000000e+00
## clarityVVS2   4979.5638   45.89948 108.488459  0.000000e+00
## colorE        -208.9524   17.92829 -11.654898  2.362529e-31
## colorF        -273.5647   18.12801 -15.090718  2.374097e-51
```

```
## colorG        -486.2456   17.74357 -27.404048 3.297004e-164
## colorH        -986.0886   18.86573 -52.268781  0.000000e+00
## colorI       -1472.2184   21.19999 -69.444302  0.000000e+00
## colorJ       -2374.9856   26.18035 -90.716326  0.000000e+00
## carat        11130.4477   47.70882 233.299574  0.000000e+00
## cutGood        673.8470   33.03571  20.397533  3.923240e-92
## cutIdeal      1030.5802   30.24302  34.076636 7.861686e-252
## cutPremium     909.7371   30.68762  29.645090 1.366892e-191
## cutVery Good   870.2063   30.79282  28.260038 2.018145e-174
## x             -841.1949   25.42991 -33.078952 1.441989e-237
## z             -222.5268   29.55966  -7.528058  5.229976e-14
```

each coefficient has $p < 0.5$. Therefore, all of these coefficients are significant.

Below is the coefficients with their standard errors in a 1.96 standard deviation confidence interval:

```
yhat.summary = summary(yhat)

coeffs = as.data.frame(yhat.summary$coefficients[-1,1:2]) # -1 is to exclude the intercept
names(coeffs)[2] = "se"
coeffs$vars = rownames(coeffs)
ggplot(coeffs, aes(vars, Estimate)) +
geom_errorbar(aes(ymin=Estimate - 1.96*se, ymax=Estimate + 1.96*se), lwd=1, colour="red", width=0) +
geom_errorbar(aes(ymin=Estimate - se, ymax=Estimate + se), lwd=1.5, colour="blue", width=0) +
geom_point(size=2, pch=21, fill="yellow")
```

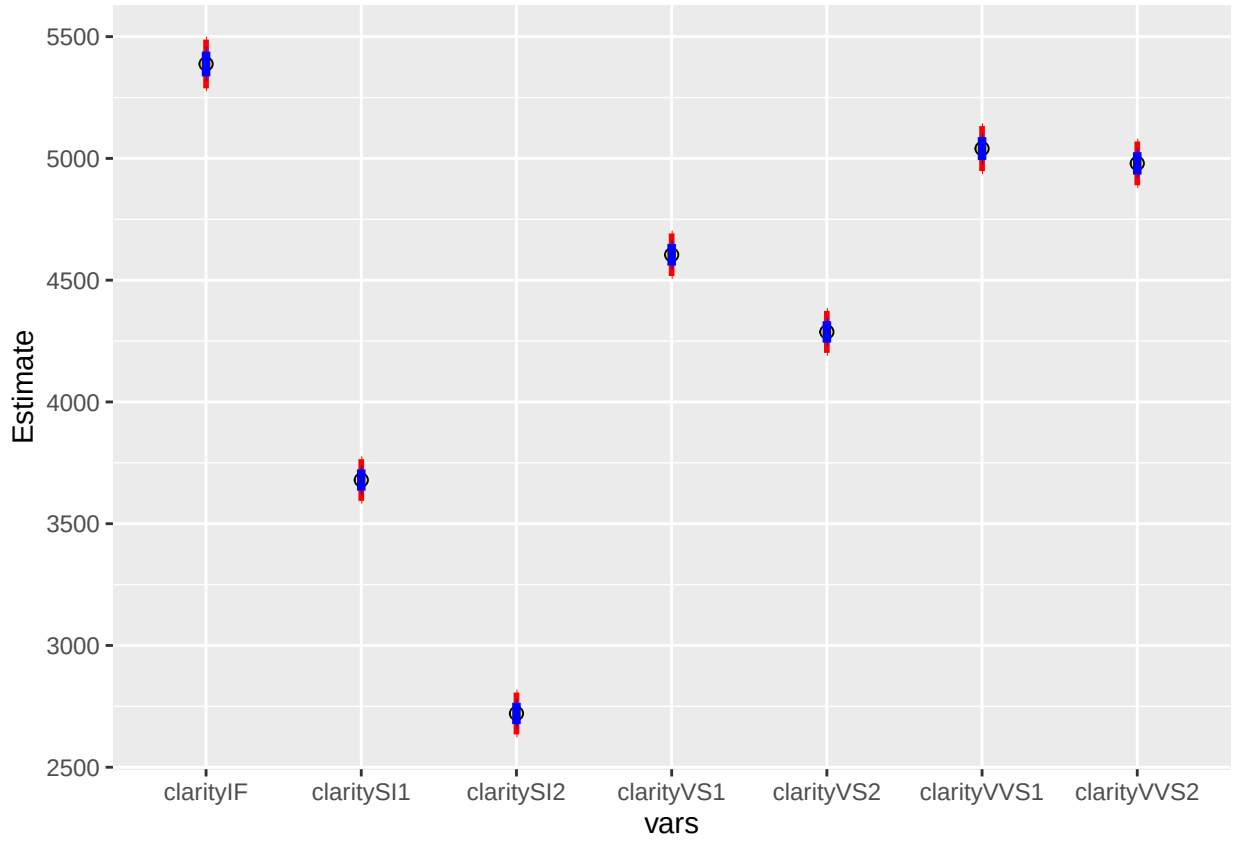


from what we can see here, carat has the highest effect on determining the price of a diamond. this is followed by the different clarities. cut appears to have a bigger effect than color in this graph aswell. It is

surprising that x, z, and color have such little effects on determining price.

I will break this graph down to make it easier to interpret. Below are the clarity coefficients:

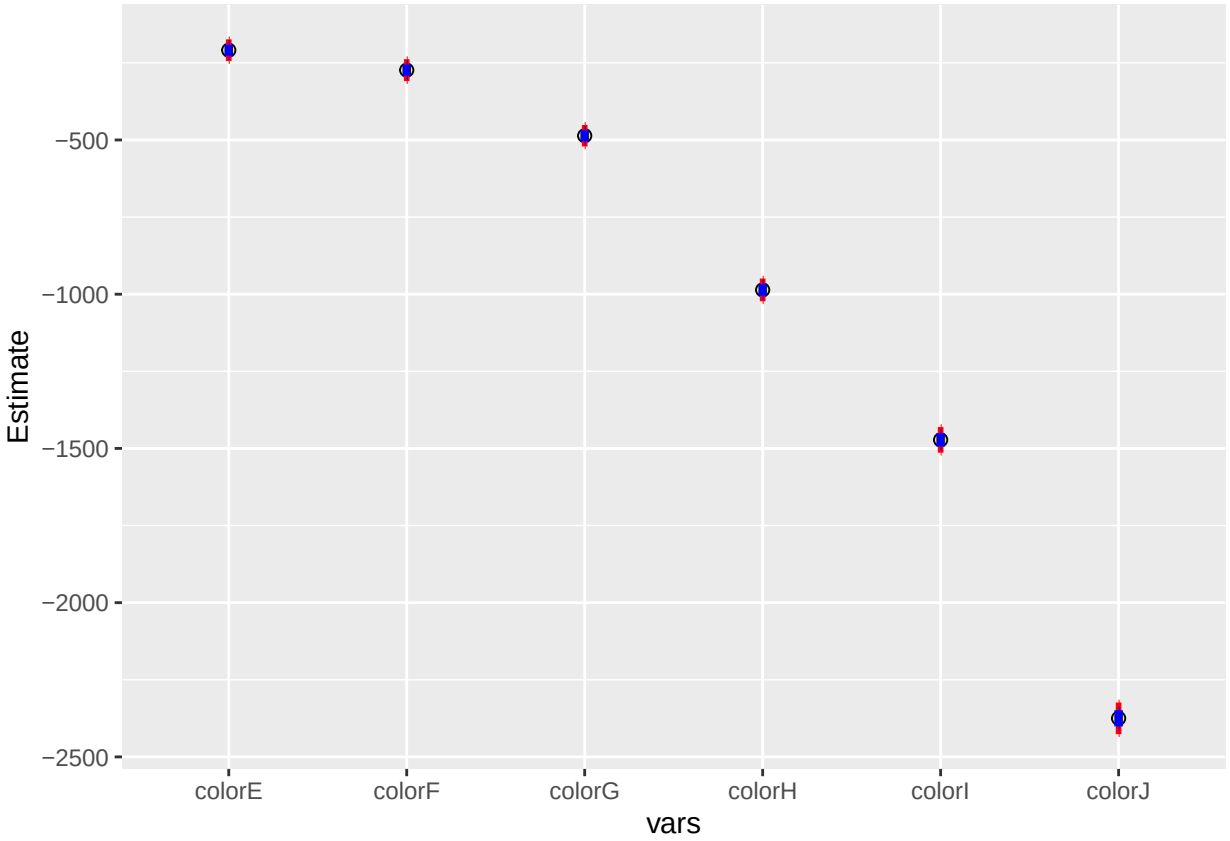
```
yhat.summary = summary(yhat)

coeffs = as.data.frame(yhat.summary$coefficients[2:8,1:2]) # -1 is to exclude the intercept
names(coeffs)[2] = "se"
coeffs$vars = rownames(coeffs)
ggplot(coeffs, aes(vars, Estimate)) +
geom_point(size=2, pch=21, fill="yellow") +
geom_errorbar(aes(ymin=Estimate - 1.96*se, ymax=Estimate + 1.96*se), lwd=1, colour="red", width=0) +
geom_errorbar(aes(ymin=Estimate - se, ymax=Estimate + se), lwd=1.5, colour="blue", width=0)
```



In clarity, it appears that having IF has the highest effect on price

Below are the color coefficients:

```
coeffs = as.data.frame(yhat.summary$coefficients[9:14,1:2])
names(coeffs)[2] = "se"
coeffs$vars = rownames(coeffs)
ggplot(coeffs, aes(vars, Estimate)) +
geom_point(size=2, pch=21, fill="yellow") +
geom_errorbar(aes(ymin=Estimate - 1.96*se, ymax=Estimate + 1.96*se), lwd=1, colour="red", width=0) +
geom_errorbar(aes(ymin=Estimate - se, ymax=Estimate + se), lwd=1.5, colour="blue", width=0)
```

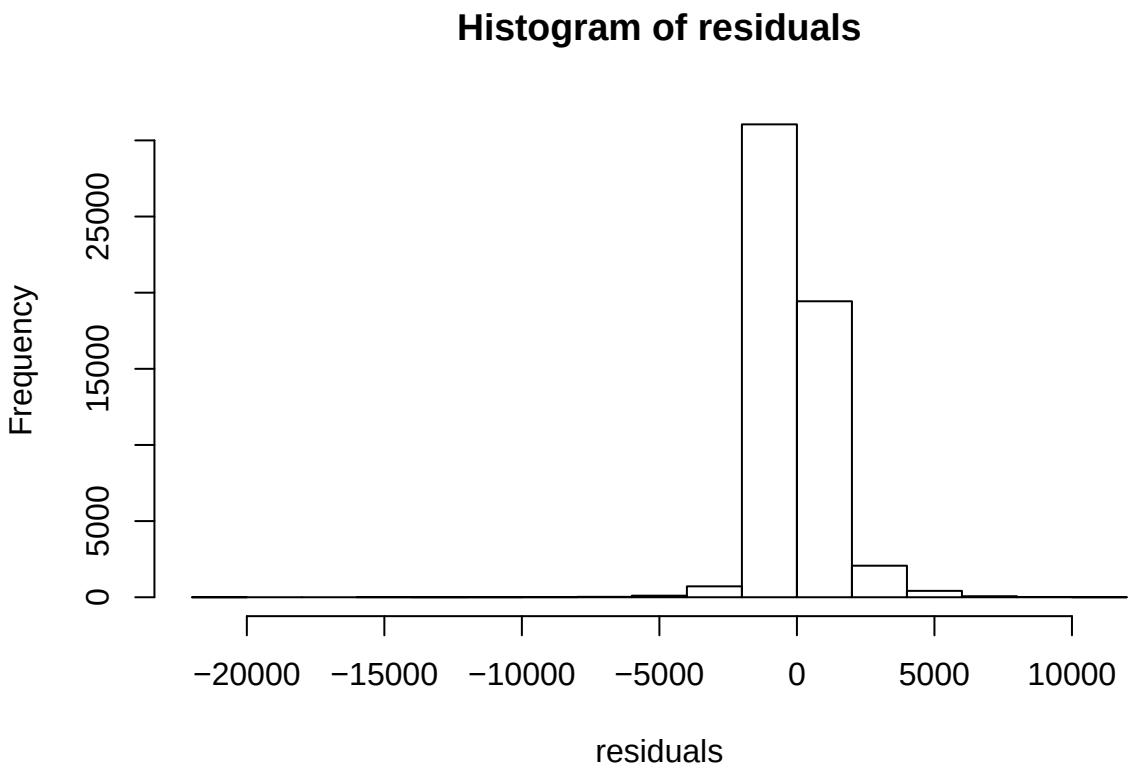


Having Color J gives the highest effect on the price out of all the colors, according to the standard error plot. However,

Residuals

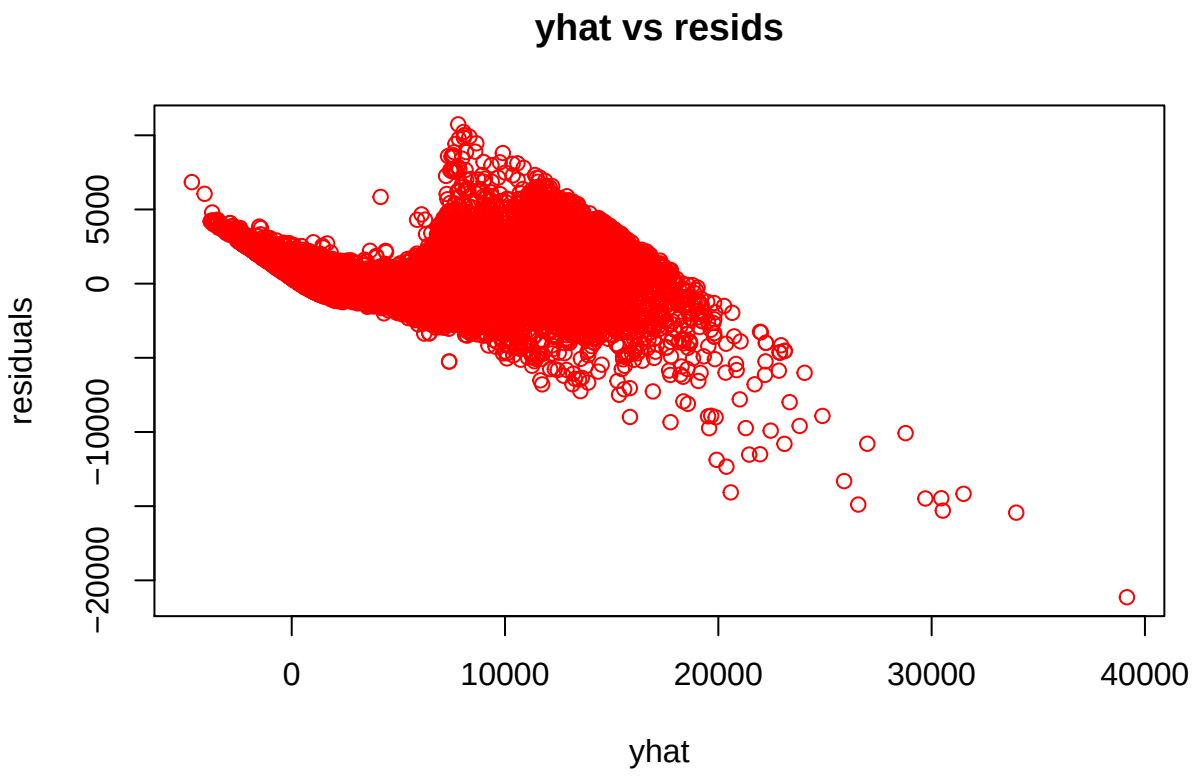
here is the histogram of residuals:

```
residuals = resid(yhat)
hist(residuals)
```



Most of the residuals are close to zero. This means that most of our residuals are very small, so our model is quite accurate

```
plot(x=fitted(yhat), y=residuals, main= "yhat vs resids", ylab="residuals", xlab = "yhat", col="red")
```



From this graph here, we can see that the higher the price gets, the greater the residual becomes. When there is a diamond that sells exceptionally high, it appears that this model will lose some accuracy in finding out its 4 C's, and the x and z values. In a linear model, we assume heteroskedasticity, or equal variance accross different samples. From this data, we see that the property of equal variance stands.

Part 3

Question 1: Dividing the test and train data

this question asks us "Divide your data to a train and a test subset and use your train subset for tuning and fitting the model and the test subset for testing its performance. (5 points)"

I will divide the model into a test and train set:

```
train = Data %>% sample_frac(0.5)
test = Data %>% setdiff(train)

x = model.matrix(price~., Data)[,-1]
y = Data$price

x_train = model.matrix(price~., train)[,-1]
x_test = model.matrix(price~., test)[,-1]
y_train = train$price
y_test = test$price
```

Ridge and lasso regression

Ridge Regression:

I will be using glmnet to do the cross-validation:

```
##RIDGE VERSION
cv.out = cv.glmnet(x_train, y_train, alpha = 0) # Fit ridge regression model on training data
bestlam = cv.out$lambda.min  # Select lamda that minimizes training MSE
bestlam
```

```
## [1] 367.7704
```

our best lambda value (the one that minimizes cross-validation training error or MSE) is $\lambda = 367.7704286$.

here is our list of λ:

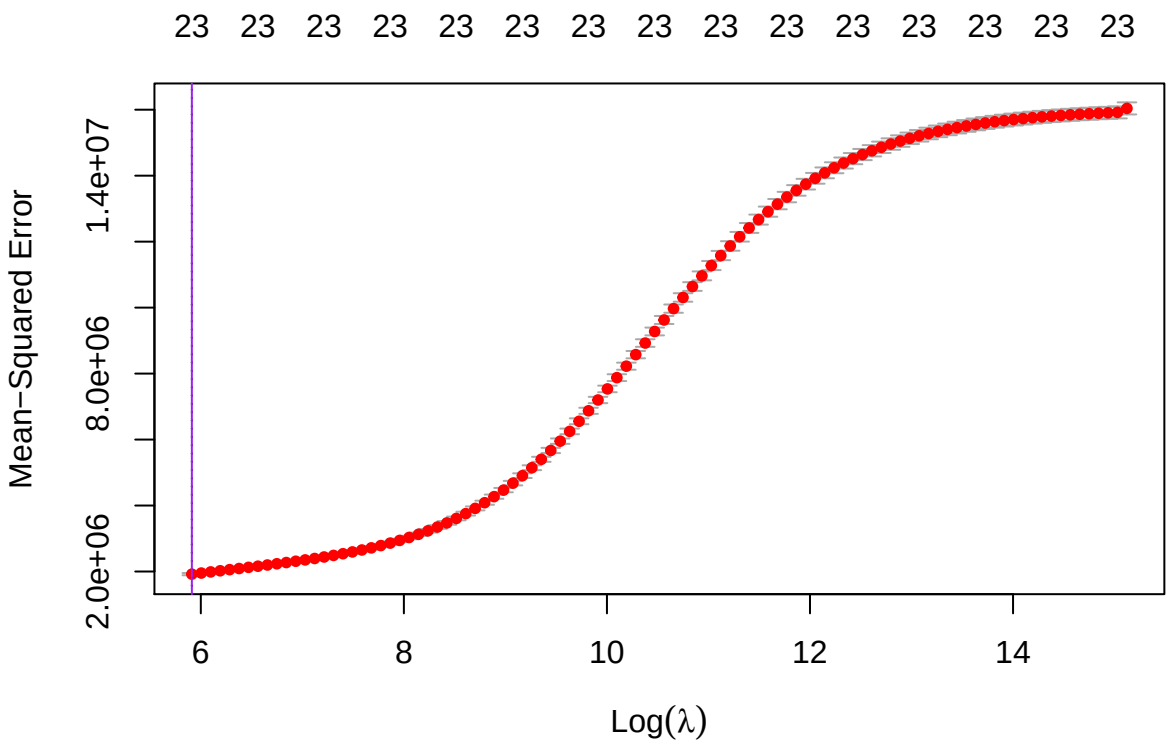
```
cv.out$lambda.1se
```

```
## [1] 367.7704
```

our lambda value within 1 standard error of the variable bestlam is 367.7704286.

Here is the cross-validation error and the chosen λ in a graph:

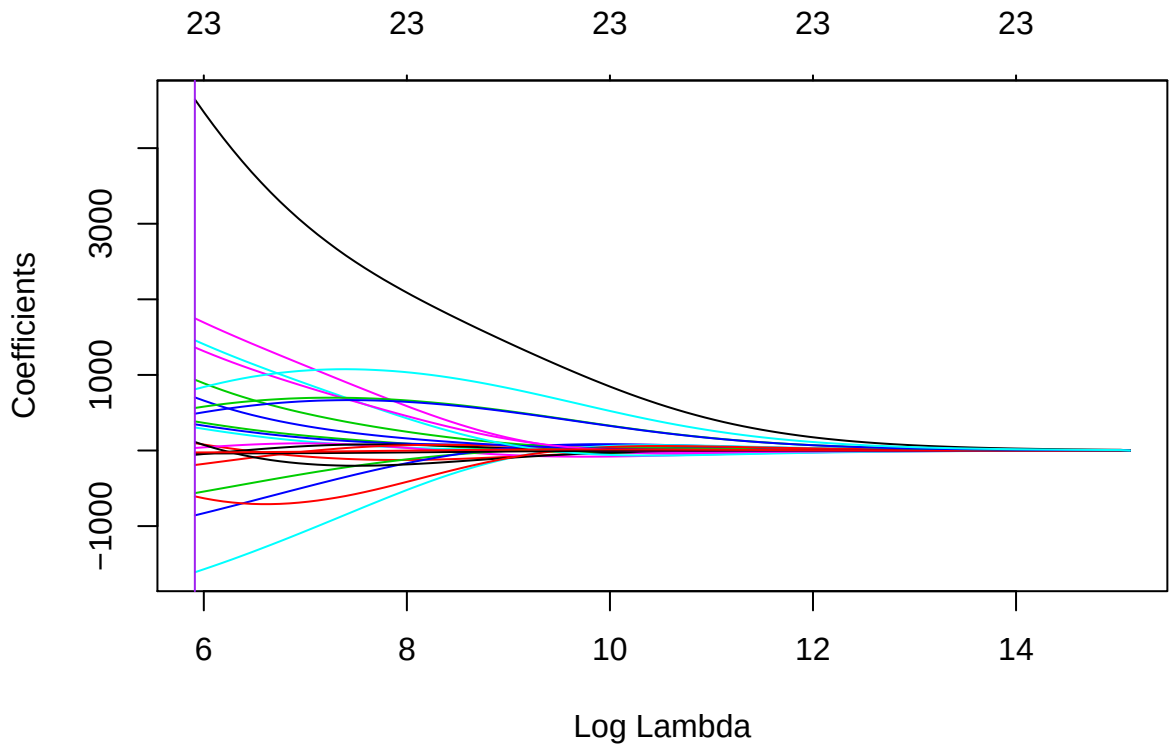
```
chosen_lam = cv.out$lambda.1se
plot(cv.out) # Draw plot of training MSE as a function of lambda
abline(v=log(chosen_lam), col="purple")
```



the purple line here shows the location of the chosen λ. It seems that this λ value is very small with a very small mean-squared error.

I wish to show how the coefficients vary with λ in a graph. I will start by making the ridge regression model first, which will benecessary in order to graph the coefficeints varying with λ. Here is the graph:

```
out = glmnet(x_train, y_train, alpha = 0) # Fit ridge regression model on the train dataset
plot(out, xvar = "lambda")
abline(v=log(chosen_lam), col="purple")
```



it appears that alot of these coefficients reduce significantly when $log(\lambda) = 9$. However, none of these coefficients will go to zero as we increase λ, as this is ridge regression.

running a prediction with ridge regression on the training dataset we get the following coefficients:

```
train_ridge = glmnet(x_train, y_train, alpha = 0) # our model
predict(train_ridge, type = "coefficients", s = chosen_lam)[1:20,]
```

```
##  (Intercept)        carat      cutGood     cutIdeal   cutPremium cutVery Good
## -6178.552059  4496.754193   108.433561   359.315023   357.041550   299.179545
##       colorE       colorF       colorG       colorH       colorI       colorJ
##    -4.555407  -104.884426  -194.413946  -555.326408  -872.741176 -1584.736343
##    clarityIF   claritySI1   claritySI2   clarityVS1   clarityVS2  clarityVVS1
##  1727.111823   128.099583  -590.382897   931.636983   699.088155  1414.290716
##  clarityVVS2        depth
##  1366.857621   -23.735840
```

Here is the MSE for the ridge model:

```
ridge_pred = predict(train_ridge, s = chosen_lam, newx = x_test) # Use chosen almbda to predict test da
ridge_MSE = mean((ridge_pred - y_test)^2) # Calculate test MSE
ridge_MSE
```

```
## [1] 1979398
```

Here is the OLS MSE:

```
OLS_model = lm(price ~ ., data = train)

OLS_MSE = mean((test$price-predict(OLS_model, test))^2)
OLS_MSE
```

```
## [1] 1502315
```

MSE for OLS model is less than the MSE for the ridge regression. it seems that ridge regression may not be the best choice for this data. We can even see this on the graph in d) where the coefficients are very spread apart.

Lasso Regression:

We will start by tuning the model by using cross-validation to find the flexibly of the model (λ):

```
## LASSO VERSION
cv.out.lasso = cv.glmnet(x_train, y_train, alpha = 1) # Fit lasso regression model on training data. al
bestlam = cv.out.lasso$lambda.min  # Select lamda that minimizes training MSE
bestlam
```

```
## [1] 0.8495974
```

Here is my chosen λ. It is within 1 standard error of the minimum λ:

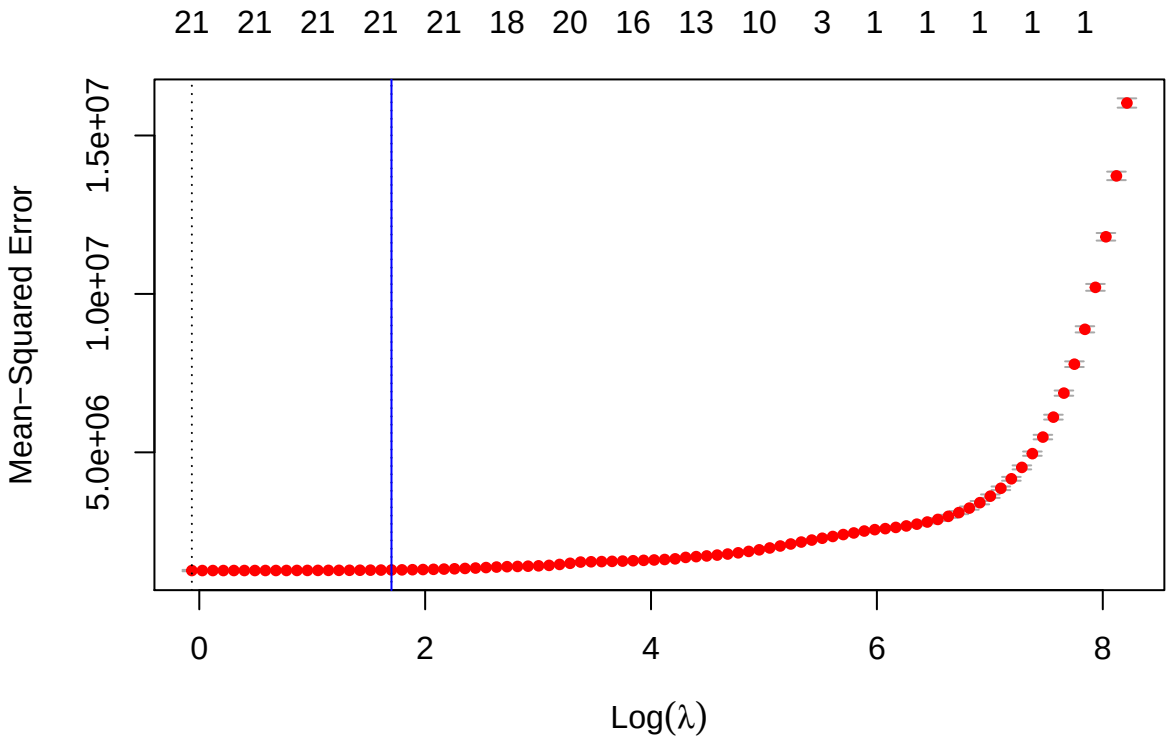
```
chosenlam = cv.out.lasso$lambda.1se
chosenlam
```

```
## [1] 6.578124
```

our chosen lambda will be 7.961706, as that is within 1 standard error of the minumum lambda.

Here is the cross-validation error and the chosen λ in a graph

```
plot(cv.out.lasso) # Draw plot of training MSE as a function of lambda
abline(v=log(chosenlam), col="blue")
```

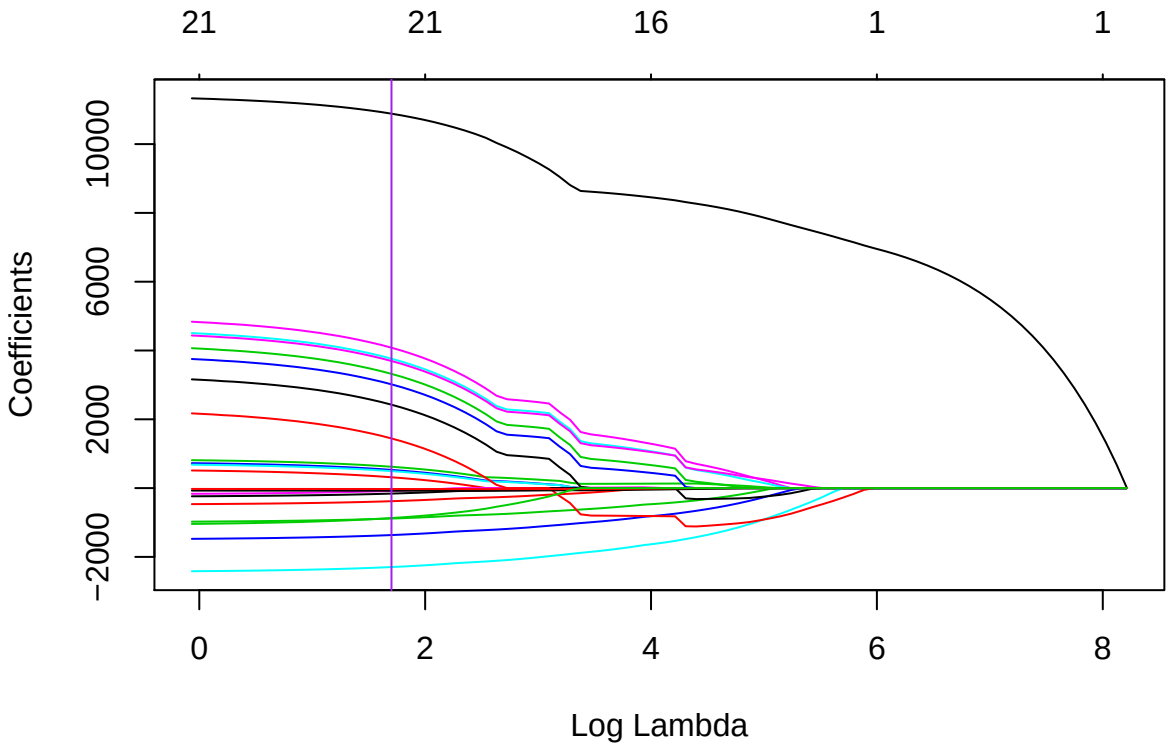


our value of lambda keeps MSE small right before the MSE starts growing with lambda.

Here is how the coefficients vary with λ in a graph

I will start by making the LASSO regression model first, which will be necessary in order to graph the coefficeints varying with λ. Here is the graph:

```
lasso_out = glmnet(x_train, y_train, alpha = 1) # Fit LASSO regression model on the train dataset
plot(lasso_out, xvar = "lambda")
abline(v=log(chosenlam), col="purple")
```



at this chosen lambda, we have some coefficients that have gone to zero.

Here are the coefficients correspondent with the chosen λ:

```
train_lasso = glmnet(x_train, y_train, alpha = 1) # our model
predict(train_lasso, type = "coefficients", s = chosenlam)[1:20,]
```

```
##  (Intercept)        carat      cutGood     cutIdeal   cutPremium cutVery Good
##   3891.68244  10849.37870    296.83608    547.38273    498.06791    462.63388
##       colorE       colorF       colorG       colorH       colorI       colorJ
##    -95.89697   -192.23586   -366.76020   -857.49945 -1336.19314 -2239.38456
##    clarityIF   claritySI1   claritySI2   clarityVS1   clarityVS2  clarityVVS1
##   4036.71714   2402.41692   1446.90566   3275.26163   2984.24293   3676.75619
##  clarityVVS2        depth
##   3656.05795    -71.80732
```

It looks like the cut, color, and clarity, and carat all make a significant impact in the model. Depth also plays a role in determining the price of a diamond.

the MSE in the test subset is as follows:

```
lasso_pred = predict(train_lasso, s = chosenlam, newx = x_test) # Use chosen almbda to predict test dat
lasso_MSE = mean((lasso_pred - y_test)^2) # Calculate test MSE
```

our MSE in the test subset is 1.323981×10^6

Let's compare this to the OLS model:

```
OLS_model = lm(price ~ ., data = train)
```

```
OLS_MSE = mean((test$price-predict(OLS_model, test))^2)
OLS_MSE
```

```
## [1] 1502315
```

OLS with all the parameters has still given us a model with a lower test MSE than LASSO regression. However, the MSE difference between Lasso and OLS(1.7833416×10^5) is less than the difference between ridge and OLS(-4.7708302×10^5). LASSO appears to be more accurate than ridge. In addition, thanks to the LASSO, we are able to make a more interpretable model than the OLS with the sacrifice of some MSE.

#Decision tree

before we start anything, we must first construct our tree:

```
tree_diamonds=tree(price~., train)
summary(tree_diamonds)
```

```
##
## Regression tree:
## tree(formula = price ~ ., data = train)
## Variables actually used in tree construction:
## [1] "carat"   "y"       "clarity"
## Number of terminal nodes:  6
## Residual mean deviance:  1903000 = 5.131e+10 / 26960
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -9798.0  -534.2  -165.1     0.0   547.8 10200.0
```

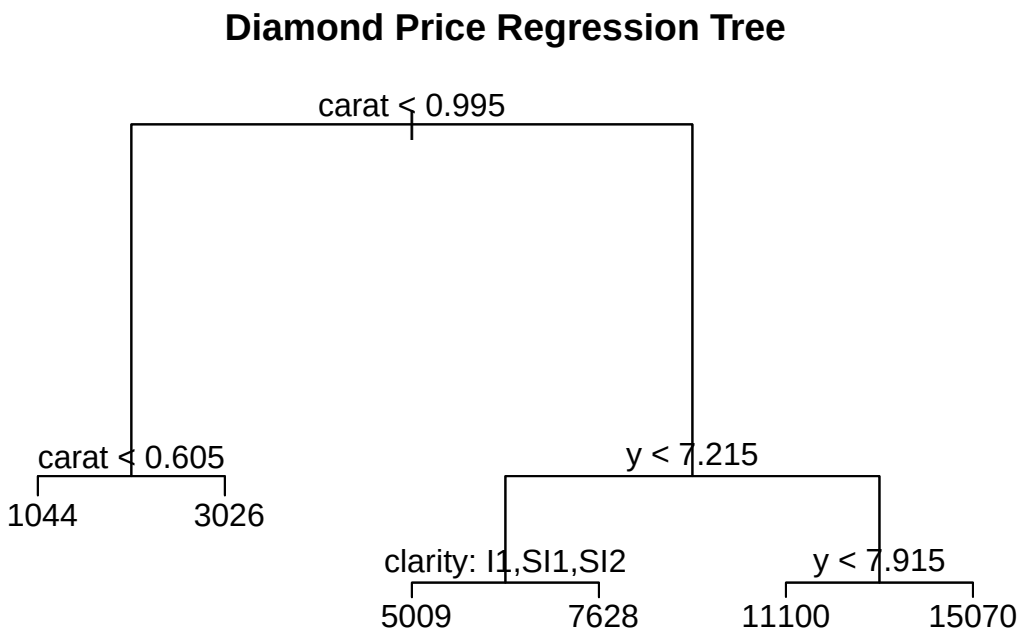
The tree uses 3 variables: “carat”, “y”, and “clarity”.

I will use cross validation pruning for my tree. Let's plot our cross-validation error of the tree with the size of its tree:

```
cv.diamonds = cv.tree(tree_diamonds)
plot(cv.diamonds$size, cv.diamonds$dev, type = 'b')
```

This tree has the lowest cross-validation error rate when there are 6 terminal nodes. In addition, this is the maximum amount of nodes presented. Since our data has some paramaters that are more valuable than others, our model should have low bias. Using bias-variance-tradeoff, I decided to pick 6 terminal nodes instead of 5. Therefore, we will prune the tree to have 6 terminal nodes.

```
prune_diamonds = prune.tree(tree_diamonds, best = 6)
plot(prune_diamonds)
text(prune_diamonds, pretty = 0)
title("Diamond Price Regression Tree")
```



the pruned tree has a total size of 4. There are 6 terminal nodes for the tree.

Interpretation

The tree uses 3 variables: 'carat', 'clarity', and 'y'. This is a bit different from the models I envisioned for a regression model. I would typically think that a model for determining the price of a diamond would be reliant on color, cut, clarity, and carat. Although I was correct about carat and clarity, I was wrong about the insignificance of 'y'. It appears that carat dominates all other variables. After carat, another factor that determines price is 'y', or width of the diamond. After 'y', clarity is the next major determinant of price.

When the mass of a diamond is less than 0.995 carats, then clarity does not matter. This is perhaps due to the fact that for a smaller diamond, its clarity is hard to see, so people do not care about it as much for a diamond. For the presence of 'y', a diamond with a greater width will be able to reflect more light, which will make it look "shinier."

When the weight of the diamond is greater than 0.995 carats and width is less than 7.195, then clarity plays a role in determining the price of a diamond. We can see that diamonds with the lower clarities (I1, SI1, SI2, VS2) have a lower average price than the ones with the higher clarities.

If the width of a diamond is greater than 7.195 (with weight $>$ 0.995 carats), then clarity is not as important, according to the tree. The greater width will have a higher price. This is expected, since greater width makes shinier looking diamonds.

OLS vs Decision Tree vs Ridge vs Lasso:

here's the MSE of the regression tree:

```
single_tree_estimate = predict(prune_diamonds,
                               newdata = test)

tree_MSE = mean((single_tree_estimate - test$price)^2)
tree_MSE
```

```
## [1] 1915642
```

Tree MSE is 1.9156421×10^6. This is greater than MSE of OLS with all covariates (1.5023152×10^6) and LASSO MSE (1.323981×10^6). This is expected because our data has a linear (or log linear) relationship between the features (or 'X' values) and Y (price). However, the tree has about the same MSE as ridge regression (1.9793982×10^6). This is perhaps due to the fact that ridge is not the best model for our dataset, as some parameters are more valueable than others for determining price. From LASSO and tree, we have learned that some parameters are not as significant on the effect of price as other parameters. For instance, 'carat' and 'clarity' are more important than 'depth'. OLS with all parameters will only risk overfitting due to increased variance.

Overall, Lasso, OLS, and Trees are all great options to use for a regression. If we do not want so many parameters, we can trade some accuracy from OLS to switch over to a LASSO or tree model. These models give better interpretability on what parameters explain the price of a diamond the most. Overall, LASSO appears to be the best model for this data due to its balance of accuracy and interpretability.

Tree based methods Part 1: Bagging

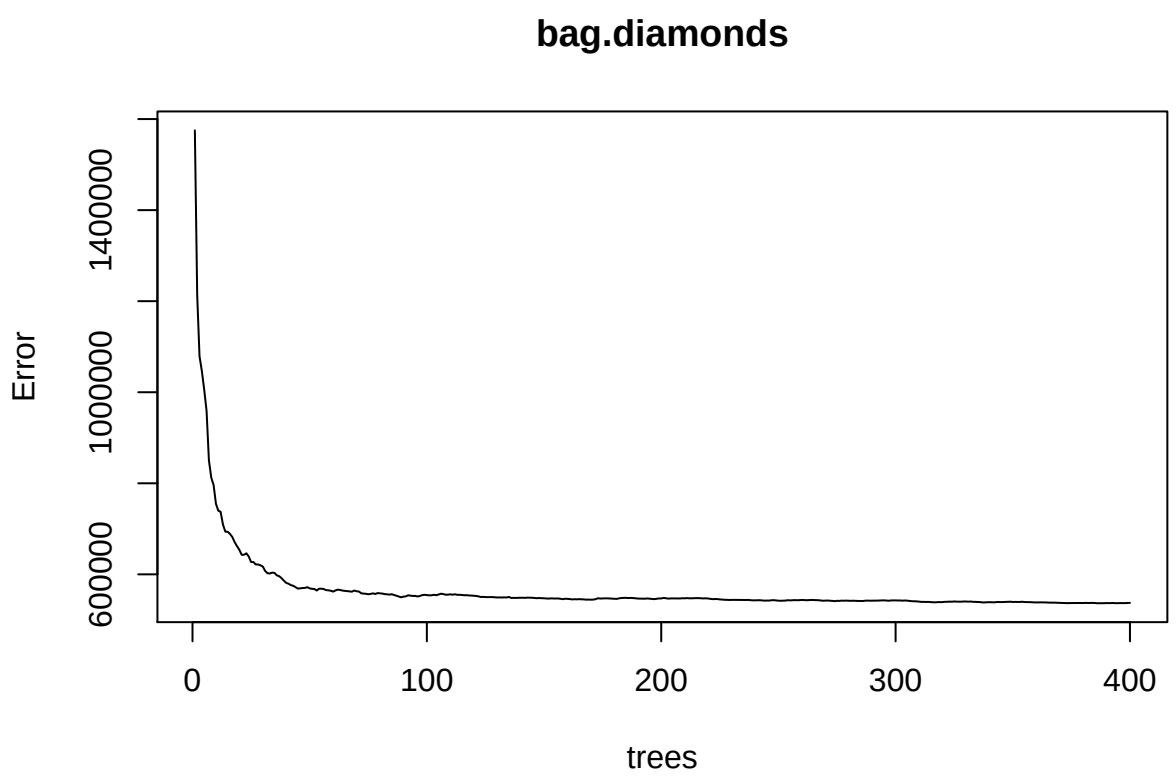
First, I have 53940 observations over 10 variables. This is a lot of data that will make it very slow to run these tree-based models. I will resample the data I will divide the dataset into a train subset and test set. The train subset is from the subset data ("Data_sample") while the test data is straigt from the full dataset in order to compare the tree models with ridge/LASSO:

```
Data_sample = Data[sample(1:nrow(Data), 5000,
   replace=FALSE),]

train = Data_sample %>% sample_frac(0.5)#this is my resampled training data to run the models faster

train_real =  Data %>% sample_frac(0.5)
test = Data %>% setdiff(train) #I will use the same data for the trees in order to have a better compar

x = model.matrix(price~., Data_sample)[,-1]
y = Data$price

x_train = model.matrix(price~., train)[,-1] #for XGBoost
x_test = model.matrix(price~., test)[,-1]
y_train = train$price
y_test = test$price

dtrain <- xgb.DMatrix(data = x_train, label = y_train) #for XGBoost
```

Fitting a Bagging regression

Plotting the out-of-bag error rate as a function of number of trees:

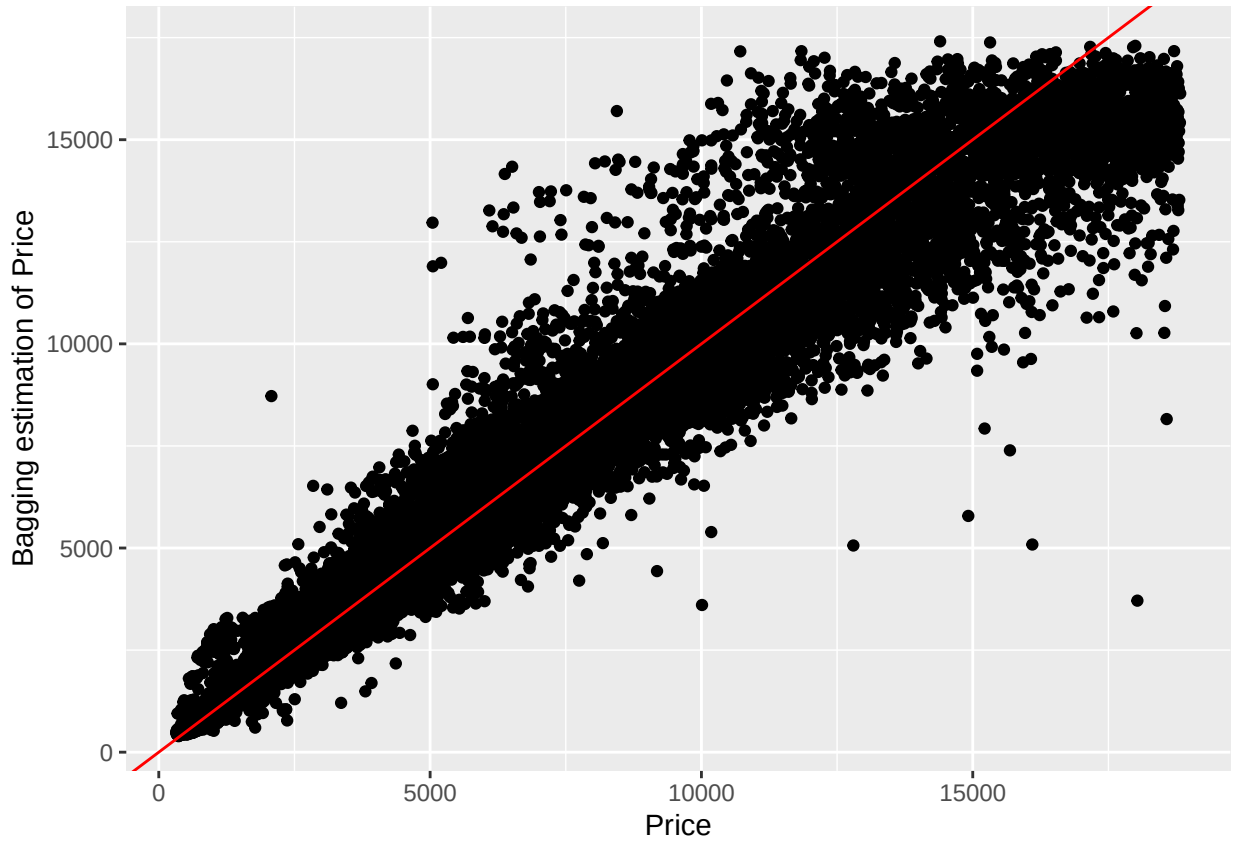
```
bag.diamonds = randomForest(price~., data=train, mtry=9, ntree=400, importance=TRUE)
plot(bag.diamonds)
```



the error keeps decreasing as we increase the number of trees that we grow. In bagging, we do not have to worry about overfitting when increasing the number of trees, so I will go with 400 trees in my model.

Here is the plot of the yhat(bag estimation of price) vs y(price)

```
yhat.bag = predict(bag.diamonds,
                           newdata = test)

ggplot() +
    geom_point(aes(x = test$price, y = yhat.bag)) +
    geom_abline(col="red")+
    labs(x="Price", y="Bagging estimation of Price")
```



As price increases, there appears to be a decrease in prediction accuracy, as more points are farther away from the line. The line represents $\hat{y} = y$

Let's get the MSE:

```
bag_MSE = round(mean((yhat.bag-test$price)^2), 2)
bag_MSE
```

```
## [1] 519724
```

MSE from bagging is equal to 5.1972395×10^5. This is less than the ridge MSE (1.9793982×10^6) and Lasso MSE (1.323981×10^6) This is unexpected, since the normal tree had a higher MSE than Ridge and LASSO

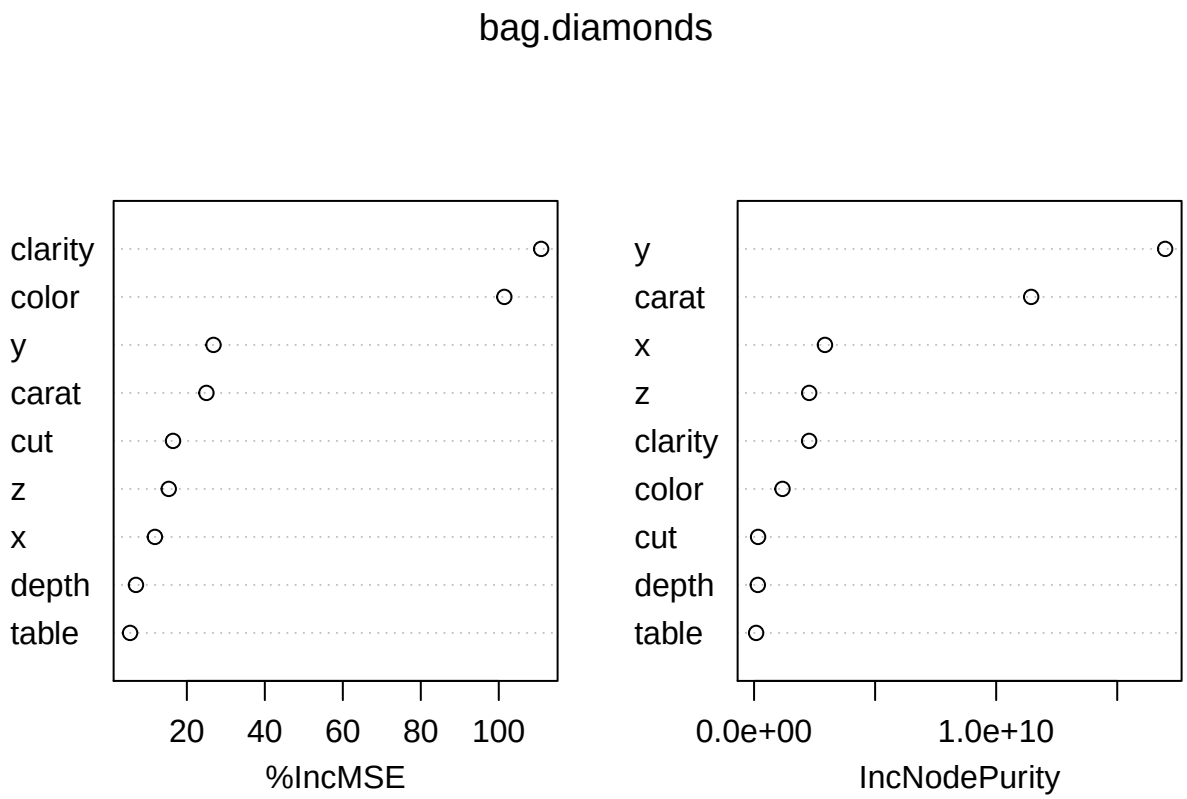
Here is the importance matrix:

```
importance(bag.diamonds)
```

```
##             %IncMSE IncNodePurity
## carat     21.0882148   11093001520
## cut        8.4905726     117960745
## color     85.9634384    1134216989
## clarity  120.4189745    2507099662
## depth      5.3861025     157061276
## table      0.7431479      87895934
## x         10.1739981    3271319436
## y         29.2141782   20857964107
## z         12.0575511    2553390340
```

Here is a plot of these importance measures:

```r
varImpPlot(bag.diamonds)
```



According to increase in node purity, 'y' and 'carat' are the two most important variables, followed by 'clarity', and 'color'. According to precent increase in MSE, 'clarity', 'color', 'z', and 'cut' are the top 4 important variables. It is interesting to see that 'carat' is not as high in the percent increase in MSE; however, it is on the same relative level as 'z' and 'cut'in terms of percent increase in MSE. From these graphs, 'color' and 'clarity' are definitely two important variables.

Random Forest

Plotting the out-of-bag error rate as a function of number of predictors considered in each split

I will be using hyperparameter tuning to find the best mtry:

```r
oob.err<-double(9)
test.err<-double(9)

#mtry is no of Variables randomly chosen at each split
for(mtry in 1:9)
{
  rf=randomForest(price ~ . , data = train, mtry=mtry, ntree=400)
  oob.err[mtry] = rf$mse[400] #Error of all Trees fitted on training

  pred<-predict(rf,test) #Predictions on Test Set for each Tree
  test.err[mtry]= with(test, mean( (price - pred)^2)) # "Test" Mean Squared Error
```

```
 # print(mtry)

}
```

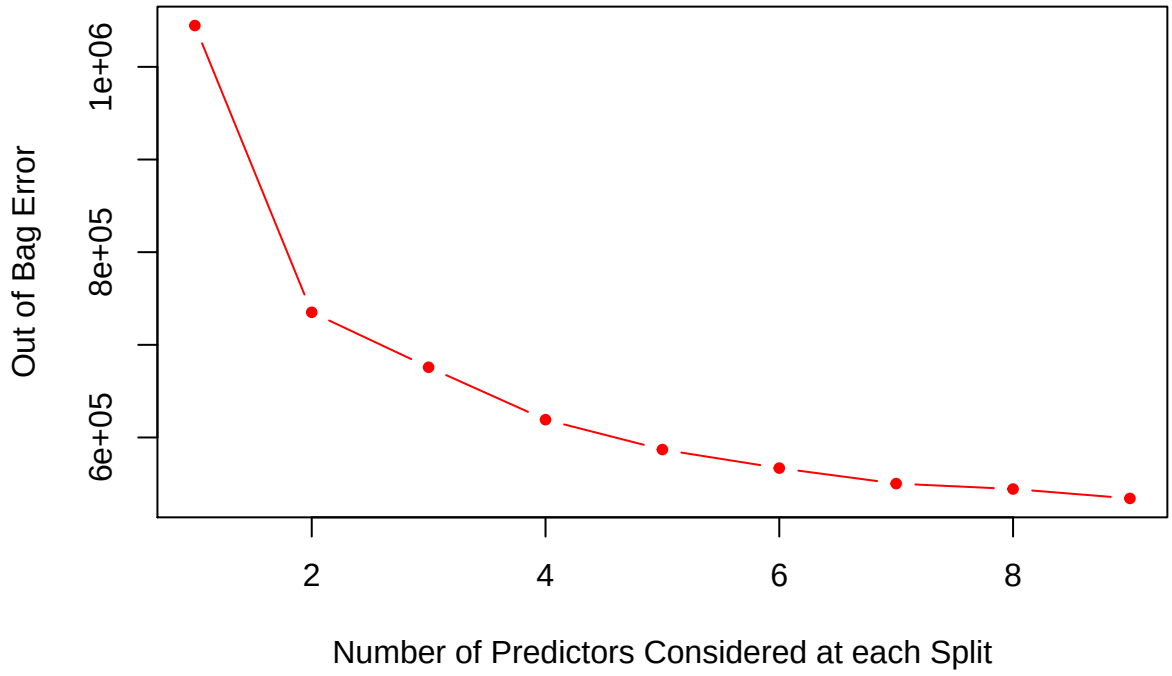
here are our out of bag error values mapped with each mtry value 1:9:

```
round(oob.err ,2)
```

```
## [1] 1235989.6  923322.9  817822.7  773108.4  737422.4  708601.5  686265.8
## [8]  687254.1  674544.3
```

here is a plot of the out-of-bag error vs the mtry value:

```
matplot(1:mtry , cbind(oob.err), pch=20 , col=c("red"),type="b",ylab="Out of Bag Error",xlab="Number of
```



in our case, $p/3 = 3$, since $p = 9$. However, at $mtry = 3$, the out-of-bag error rate is not settled down I will use $mtry = 5$, as this is close enough to 3, while the error rate is settled down. I do not want to select $mtry = 9$ despite the low error rate because in random forest, we want to use a random subsample of the parameters for each tree we grow in order to uncorrelate the trees.

From the out of bag error to tune the number of predictors in each split (mtry) [note: you do not need to run cross validation and you can tune your model using the outof-bag error]

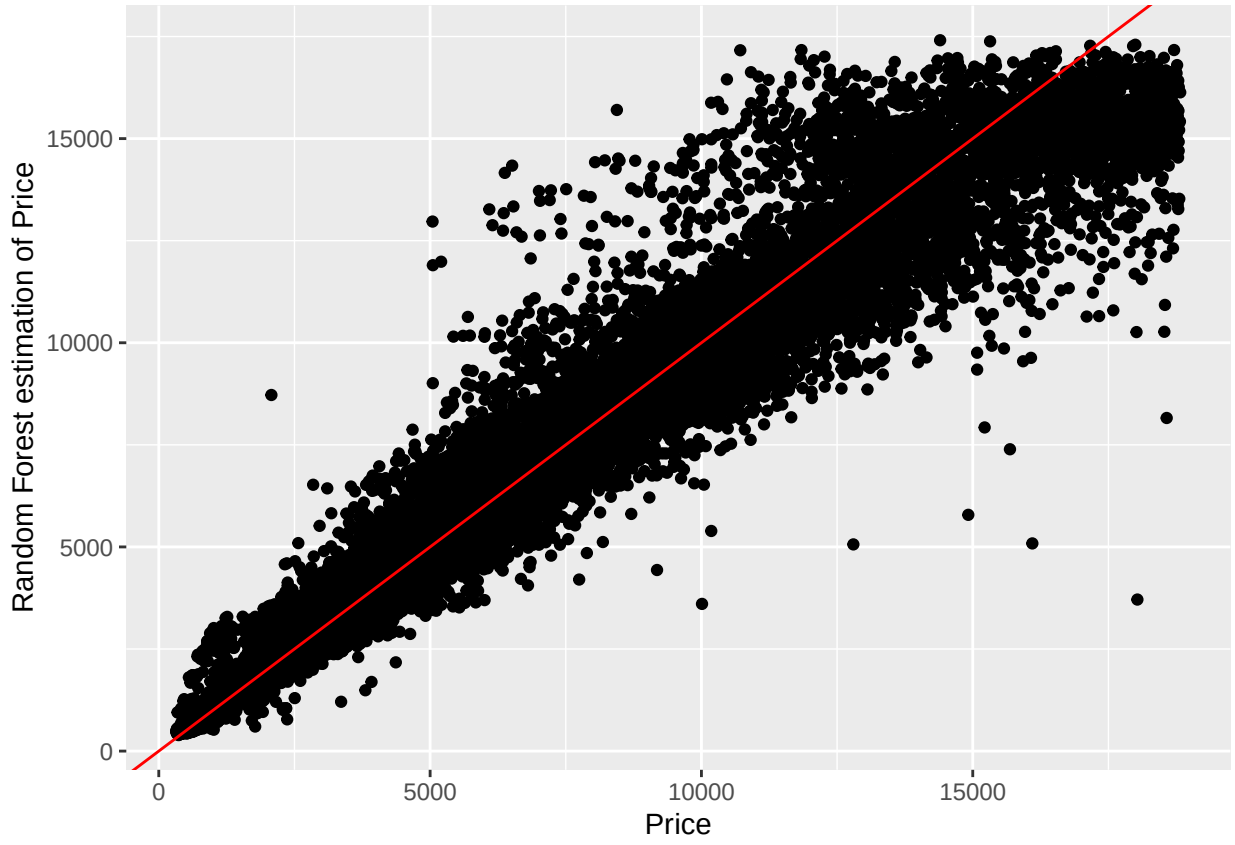
Here is the random forest model:

```
rf.diamonds = randomForest(price~., data=train,
                           mtry=5, ntree=400,
                           importance=TRUE,
                           do.trace=100)
```

```
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  100 | 7.665e+05     4.58 |
##  200 | 7.553e+05     4.51 |
##  300 | 7.534e+05     4.50 |
##  400 | 7.506e+05     4.48 |
```

Since I am doing a regression, here is a plot of $\hat{y}$ vs y:

```
yhat.rf = predict(rf.diamonds,
                          newdata = test)

ggplot() +
    geom_point(aes(x = test$price, y = yhat.bag)) +
    geom_abline(col="red")+
    labs(x="Price", y="Random Forest estimation of Price")
```



as usual, an increase in price leads to a decrease in prediction accuracy, as more points are farther away from the line. The red line represents $\hat{y} = y$

Random forest vs bagging, lasso, and ridge:

```
rf_MSE = mean((yhat.rf - test$price)^2)
rf_MSE
```

```
## [1] 584364.9
```

The MSE for Random Forest is 5.8436488×10^5. This is slightly higher than bag_MSE, but it is still lower

than Lasso and Ridge. this is intresting because Random Forest is meant to decorrelate the trees by randomly selecting other predictors in the tree-based model.

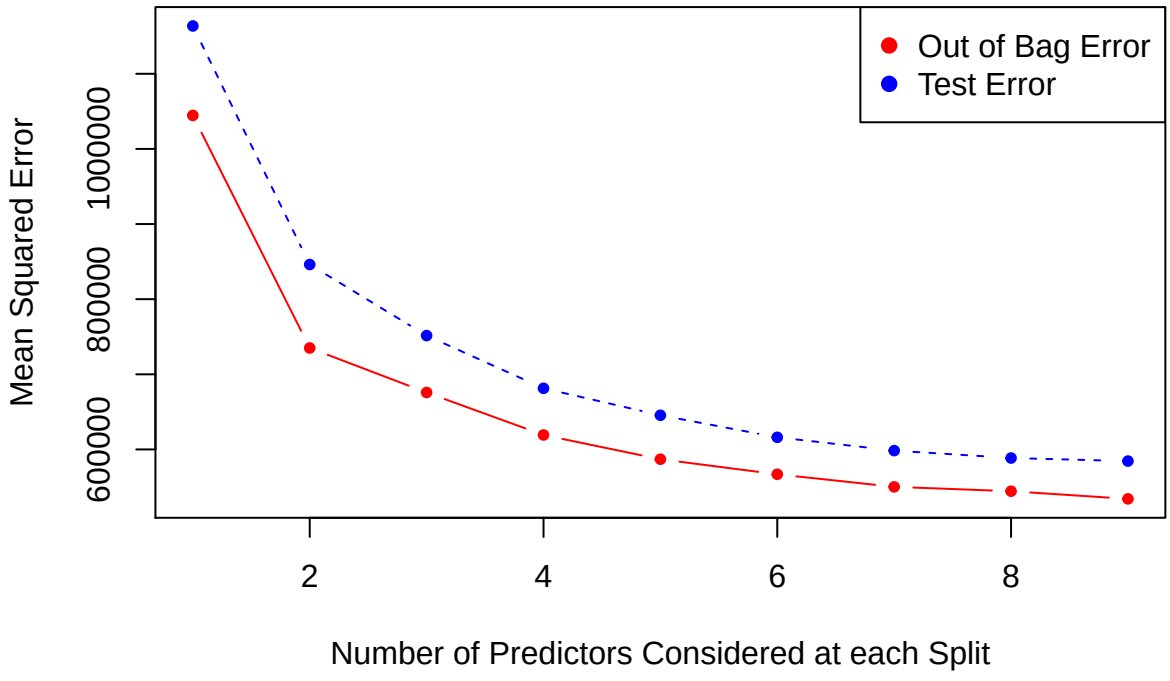
here is the importance matrix for random forest:

```
importance(rf.diamonds)
```

```
##           %IncMSE IncNodePurity
## carat   19.087224    9896369616
## cut      7.846763     182320052
## color   61.193206    1039224613
## clarity 96.008523    2310144586
## depth    9.655057     258755569
## table    1.733036     126273984
## x       14.003781    6552214474
## y       25.517787   17572669700
## z       12.381895    3672613940
```

Plotting the test error and out-of-bag error:

```
matplot(1:mtry , cbind(oob.err,test.err), pch=20 , col=c("red","blue"),type="b",ylab="Mean Squared Erro
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blue"))
```



from these plots, the Out-of-Bag error and Test error are nearly identical in their pattern, making my decision of setting $mtry = 5$ a valid one.

Boosting regression

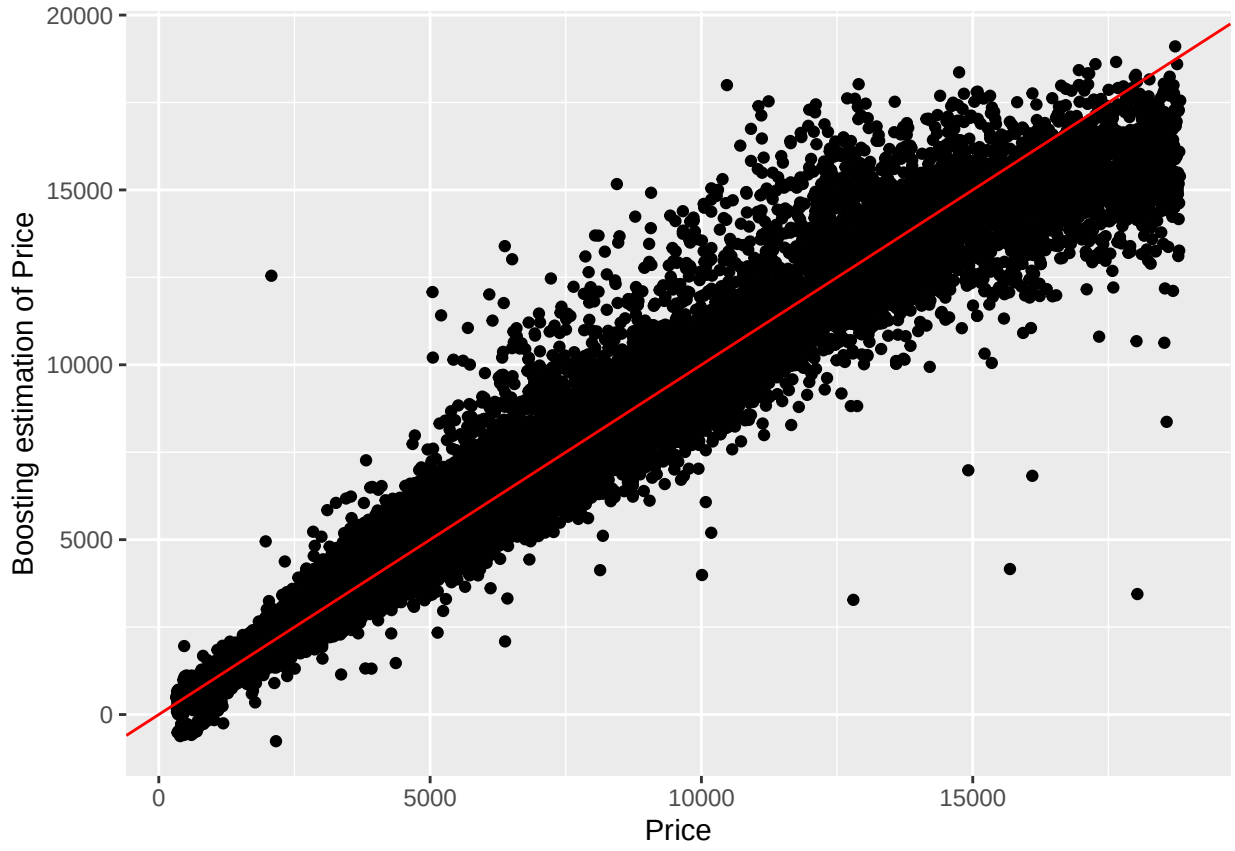
Here is the boosted model:

```
boost.diamonds = gbm(price~.,
                     data =train,
                     distribution = "gaussian", #"bernoulli" for logitic regression
                     n.trees = 200,
                     interaction.depth = 4)
```

I halved the number of trees grown to prevent overfitting. Boosting has the concern with overfitting while bagging/random forest do not.

here is a plot of our prediction:

```
yhat.boost = predict(boost.diamonds,
                           newdata = test,
                           n.trees = 200)

ggplot() +
    geom_point(aes(x = test$price, y = yhat.boost)) +
    geom_abline(col="red")+
    labs(x="Price", y="Boosting estimation of Price")
```



Per usual, an increase in price leads to a decrease in prediction accuracy, as more points are farther away from the line. However, this graph appears to have more values that are farther away from the line at lower values of price. The red line represents $\hat{y} = y$.

#Boosting vs Lasso/Ridge/Bagging/RandomForest models

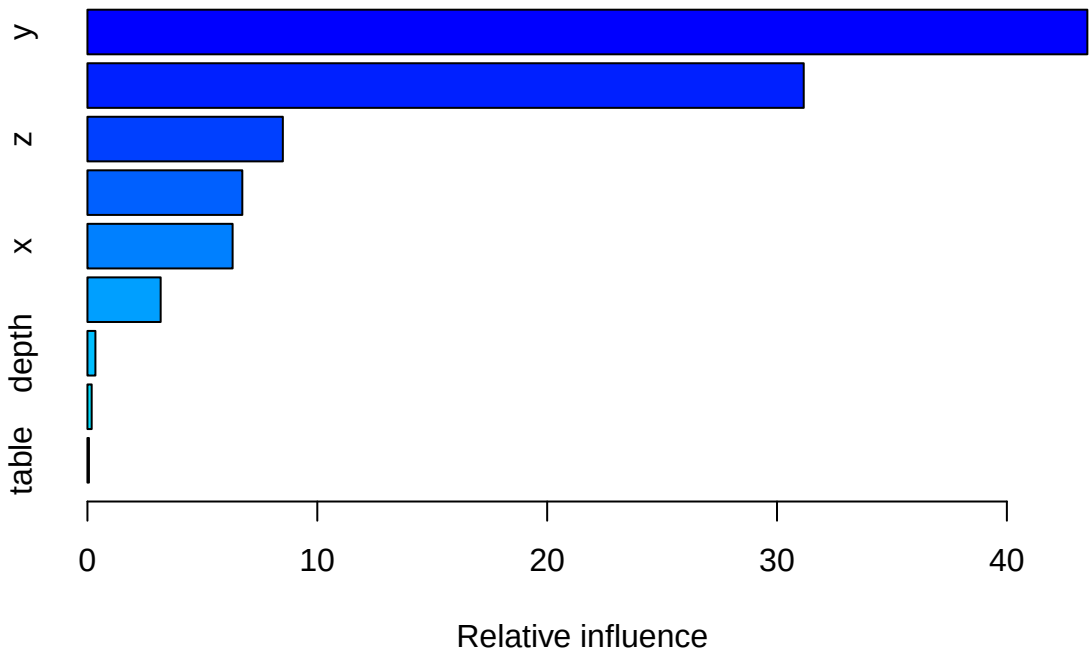
```
boost_MSE = round(mean((yhat.boost - test$price)^2),2)
boost_MSE
```

```
## [1] 420903.5
```

Boost MSE is 4.2090351×10^5. It appears I misinterpreted the previous graph. This is lower than Lasso(1.323981×10^6), ridge(1.9793982×10^6), bagging(5.1972395×10^5), and random forest(5.8436488×10^5)

rather than an importance matrix, gradient boosting has an influence matrix:

```
summary(boost.diamonds)
```




```
##             var     rel.inf
## y             y 40.42405671
## carat     carat 38.47074016
## z             z  7.80831113
## clarity clarity  6.78968292
## color     color  3.07048927
## x             x  2.91004731
## cut         cut  0.25444831
## depth     depth  0.17868779
## table     table  0.09353641
```

contrary to all the other models, x,y, and z are the three most important variables in the boosting model. From this model, it appears that the 4 C's are not as important in determining price. This makes sense because the x, y, and z values of a diamond best determine its carat, as the density of diamond is constant. It is interesting to note, however that 'y' (the width of the diamond) has the highest value. This is perhaps because a wider diamond allows more light to come into its face, thereby allowing more reflections inside the

diamond, which gives the diamond the “shinier” look.

XGboost

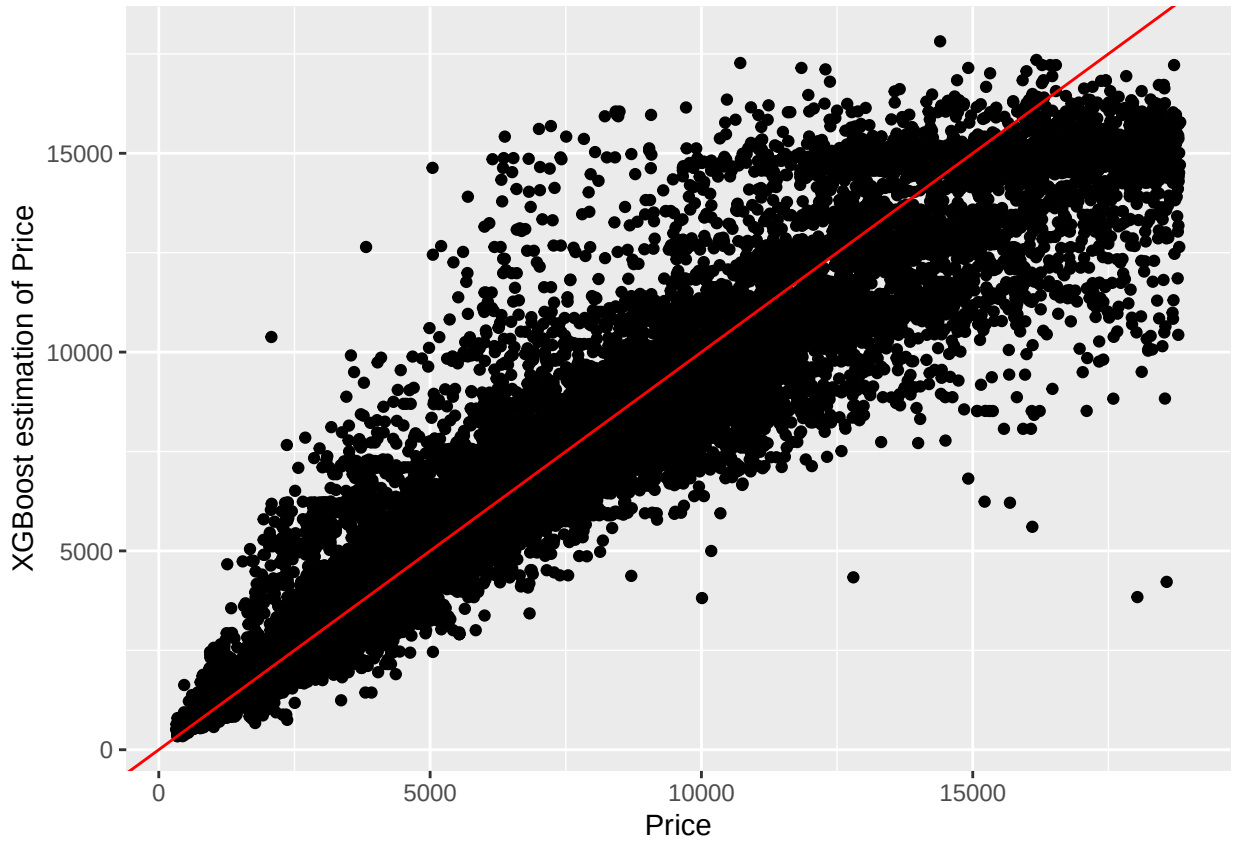
here is the XGboost regression. I will also tune the hyperparameters such as number of iterations,:

```
diamonds.xgb = xgboost(data=dtrain,
                       max_depth=5,
                       eta = 0.1,
                       nrounds=40, # max number of boosting iterations (trees)
                       lambda=0,
                       print_every_n = 10,
                       objective="reg:linear") # for classification: objective = "binary:logistic"
```

```
## [1]  train-rmse:5190.343750
## [11] train-rmse:2053.591553
## [21] train-rmse:1074.232056
## [31] train-rmse:784.714539
## [40] train-rmse:670.941162
```

Here is the predicted Y vs actual Y:

```
yhat.xgb <- predict(diamonds.xgb, x_test)

ggplot() +
    geom_point(aes(x = test$price, y = yhat.xgb)) +
    geom_abline(col="red")+
    labs(x="Price", y="XGBoost estimation of Price")
```



The $\hat{y}$ vs y relation appears to be more logarithmic. The points are also more scattered than they are in bagging, random forest, and boosting. The higher the price is, the more the predictions fall of from the actual values.

XGBoost vs Lasso/Ridge/Bagging/RandomForest/Boosting models

here is the MSE of XGBoost:

```
xgb_MSE = round(mean((yhat.xgb - test$price)^2),2)
xgb_MSE
```

```
## [1] 937584.7
```

The XGboosting MSE is 9.3758468×10^5 This is higher than bagging (5.1972395×10^5), boosting (4.2090351×10^5), and random forest (5.8436488×10^5). XGboost MSE is still lower than Lasso(1.323981×10^6) and ridge (1.9793982×10^6). My obesrvations about the XGBoost plot are correct. It appears that using "slower" methods for developing trees, such as boost and bagging, are better than XGboost.

here is the importance matrix:

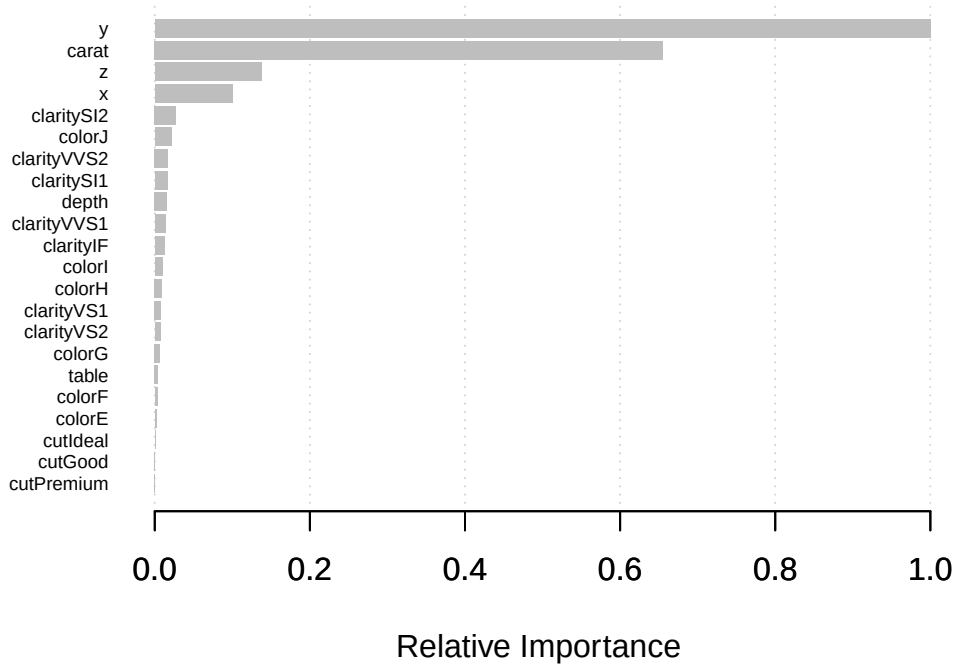
```
xgb_importance <- xgb.importance(colnames(x_train),model=diamonds.xgb)
xgb_importance
```

```
##          Feature         Gain        Cover    Frequency
##  1:            y 5.661808e-01 2.004357e-01 0.0988626422
##  2:        carat 2.990726e-01 1.725175e-01 0.1819772528
##  3:            z 3.950275e-02 1.777223e-01 0.0988626422
##  4:   claritySI2 1.615095e-02 5.289183e-02 0.0673665792
##  5:            x 1.038400e-02 7.635730e-02 0.0621172353
```

```
##  6:       colorJ 9.367441e-03 4.833715e-02 0.0542432196
##  7:        depth 9.329099e-03 1.776063e-02 0.0699912511
##  8:  clarityVVS2 9.207653e-03 2.377352e-02 0.0271216098
##  9:   claritySI1 7.630044e-03 2.854022e-02 0.0332458443
## 10:    clarityIF 6.490212e-03 2.842221e-02 0.0279965004
## 11:       colorI 6.285903e-03 4.866320e-02 0.0481189851
## 12:       colorH 3.457122e-03 4.038398e-02 0.0376202975
## 13:   clarityVS1 3.183923e-03 1.475418e-02 0.0201224847
## 14:  clarityVVS1 3.104287e-03 3.039050e-02 0.0411198600
## 15:   clarityVS2 2.819236e-03 8.499258e-03 0.0209973753
## 16:       colorF 1.990179e-03 4.590679e-03 0.0183727034
## 17:       colorE 1.564561e-03 3.304489e-03 0.0148731409
## 18:     cutIdeal 1.316595e-03 5.452807e-03 0.0148731409
## 19:   cutPremium 1.007983e-03 2.322344e-03 0.0122484689
## 20:        table 9.082547e-04 5.756852e-03 0.0306211724
## 21:       colorG 8.147077e-04 9.025336e-03 0.0122484689
## 22:      cutGood 1.945039e-04 8.401243e-05 0.0061242345
## 23: cutVery Good 3.721863e-05 1.400207e-05 0.0008748906
##          Feature         Gain        Cover    Frequency
```

and here is a plot of the importance matrix data:

```
xgb.plot.importance(xgb_importance, rel_to_first=TRUE, xlab="Relative Importance")
```



XGBoosting brings back carat as an important variable, although 'y' is top priority in importance. After y, carat, z, and clarity and color take over, but only certain types help with determining price. It is surprising to see color so low, despite being one of the "4 C's" among diamonds. It is interesteing see that both Boosting

and XGBoosting have very similar rankings, where 'y' is the most important covariate, followed by 'carat'. In addition, both Boost and XGBoost rank clarity and color next important after 'z' and 'x' in terms of relative importance. Overall, both models appear to be similar.

It appears that Bagging or Random Forest are the two best tree-based options for our dataset. In addition, these two models perform better than LASSO and Ridge. This contradicts my belief in part 3 that trees were inferior to the linear models. The MSEs for Bagging, Random Forest, and both boosts had MSEs that were one digit lower than the Ridge and LASSO MSE, despite me using a subset of the dataset for the sake of runtime.

It appears that price has a non-linear relationship with the covariates present. It is also surprising to see that the top 4 most important variables were not always the 4 C's; instead, in each model, there was typically a combination of carat, z, y, x and another C as the top 4 important variables. All the tree models have x, y, z, carat, clarity, and color as the most important covariates, while ridge and LASSO make carat, clarity, and color the 3 most important inputs in our model, while neglecting x,y,z. Given that the tree based models have a lower MSE than ridge and LASSO, it appears that x, y, z play a bigger role in determining the price of a diamond than clariy, color, and cut (3 out of the 4 C's), contrary to popular belief.

Neural Networks

For the final part, we shall dive into one of the most popular models of today: the neural network (I shall refer to it as a "neural net.") It acts like neurons in your brain or some shit idk....HERE'S A NEURAL NET. First, however, we must prepare our data to be in a matrix format for the neural net to read

First, I will prepare the data. I am turning some discrete attributes such as cut and clarity as numerics so they can run through the network:

```
Data_sample = Data[sample(1:nrow(Data), 5000,
   replace=FALSE),]

attach(Data_sample)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     x, y
```

```
Data_sample$cut = as.numeric(Data_sample$cut)
Data_sample$color = as.numeric(Data_sample$color)
Data_sample$clarity = as.numeric(Data_sample$clarity)
detach(Data_sample)

train = Data_sample %>% sample_frac(0.7)#this is my resampled training data to run the models faster

test = Data_sample %>% setdiff(train) #I will use the same data for the trees in order to have a better

##converting to matrix format
train_labels <- as.matrix(train[,"price"]) # This is the Y variable that we are testing
train_data <- as.matrix(train[!names(train) %in% c("price")]) #These are the X variables
test_data <- as.matrix(test[!names(train) %in% c("price")])
test_labels <- as.matrix(test[,"price"])
```

Now I will normalize the variables:

```
train_data <- scale(train_data)
test_data <- scale(test_data)
```

Now here is our model. It takes in 64 units for the hidden layers and 1 output layer for y. For the activation

function, I am using an ReLU, since that gave me a lower MSE when I worked with this dataset before:

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1)
```

now I will compile the model and define our loss function. I am choosing the MSE as loss:

```
model %>% compile(
  loss = "mse",
  optimizer = optimizer_rmsprop(),
  metrics = list("mean_absolute_error")
)
```

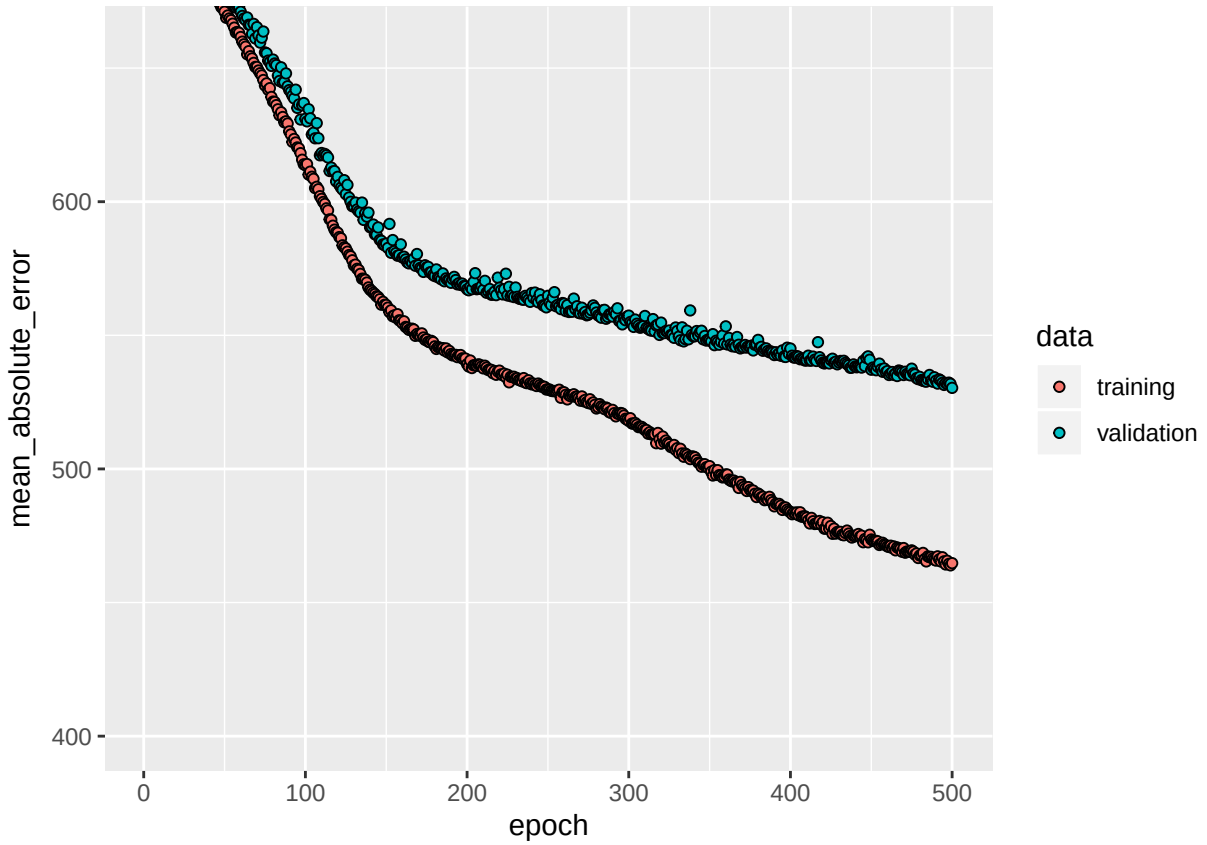
here is the fit:

```
epochs <- 500
history <- model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = 0.2
)
```

the validation_split specifies what portion of the train data should be used as a validation set. I have chosen 1/5 of the data for validation

I will plot the mean absolute error of both train and validation sets of the model in order to see how many epochs I will need:

```
plot(history, metrics = "mean_absolute_error", smooth = FALSE) +
coord_cartesian(ylim = c(400, 660))
```



the mean absolute error is the lowest at 500 epochs, so I will be keeping the model at 500. Here is the MSE of the neural network:

```
test_predictions <- model %>% predict(test_data)
NN_MSE = mean((test_labels - test_predictions)^2)
```

The MSE of this network is $\{3.2129017 \times 10^6\}$.

##BONUS: Support Vector Regression with Hyperparameter Tuning: First, let me prepare the data again. I will be using the resampled version that I used for the trees and neural net:

```
Data_sample = Data[sample(1:nrow(Data), 5000,
   replace=FALSE),]

train = Data_sample %>% sample_frac(0.5)#this is my resampled training data to run the models faster

train_real =  Data %>% sample_frac(0.5)
test = Data %>% setdiff(train) #I will use the same data for the trees in order to have a better compar
```

To start off, I will tune our model parameters for the support vector regression. I will be performing a grid search:

```
#grid search
tuneResult <- tune(svm, price ~ .,  data = train,
              kernel = "linear",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1,5,10,100)))
```

```
summary(tuneResult)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 1303008
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-03 2999260   476956.8
## 2 1e-02 1561170   184991.7
## 3 1e-01 1344992   335421.7
## 4 1e+00 1303008   508512.2
## 5 5e+00 1304639   561245.7
## 6 1e+01 1320203   608299.1
## 7 1e+02 1329279   634875.8
```

beyond a value of $cost = 100$, there is very little decrease in error, so I will use 100 as our cost. here is our new model:

```
svm.diamonds = tuneResult$best.model #picked the best model from tune()

summary(svm.diamonds)
```

```
##
## Call:
## best.tune(method = svm, train.x = price ~ ., data = train, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.04166667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1150
```

```
#svm.diamonds$index
```

and as predicted, $cost = 100$ is the chosen value. Now let's predict the SVM model:

```
svr.predict = predict(svm.diamonds, test)
```

now let's get the test MSE:

```
svm_MSE = mean(test$price - svr.predict)^2
```

the MSE of the support-vector regression is 1.572259×10^4. This is the lowest MSE of them all. It appears that the data that we have is linear in its structure, as linear hyperplanes are able to be fit in it. This is hard to visualize, as we have a dataset that is 10-dimensional.

Comparing all models

```
model_names = c("OLS", "Ridge", "Lasso", "Pruned Tree", "Bagging", "Random Forest", "SVM", "Neural Netw
model_MSEs = c(OLS_MSE, ridge_MSE, lasso_MSE, tree_MSE, bag_MSE, rf_MSE, svm_MSE, NN_MSE)

MSE_df = data.frame(Model = model_names, MSE = model_MSEs)
print(MSE_df)
```

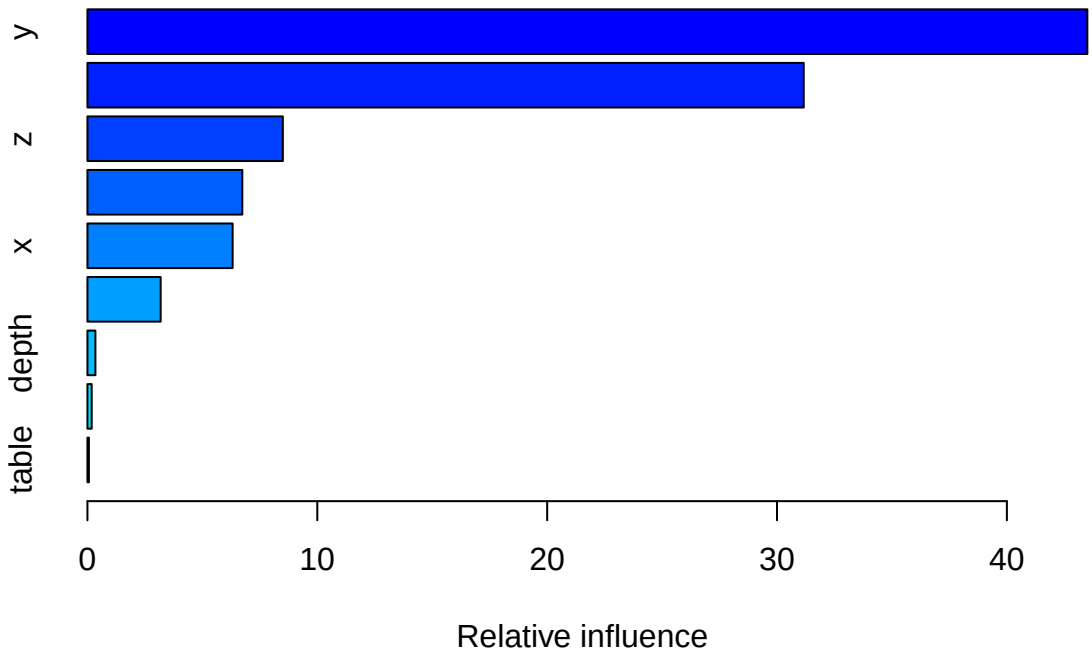
```
##            Model        MSE
## 1            OLS 1502315.17
## 2          Ridge 1979398.18
## 3          Lasso 1323981.00
## 4    Pruned Tree 1915642.14
## 5        Bagging  519723.95
## 6  Random Forest  584364.88
## 7            SVM   15722.59
## 8 Neural Network 3212901.71
```

The lowest MSE is from support vector regression 1.572259×10^4. While this would be very good for prediction, it is not very interpretable, especially with the high number of variables(which gives a high dimension) The MSE of this network is $\{3.2129017 \times 10^6\}$. This is lower than the linear regressions, but it still does not beat the tree-based MSE models with their MSE. Despite me using the smae resampled train and test data that I used for tree-based methods, The neural network still performed better than the linear models. The linear models had the cheat of having more observations which should reduce variance, but the neural network fit the data better with fewer observations. However, the neural net still has a higher MSE than the tree-based models, which use the same data as the neural net. The linear models are not so great to interpret, and they have very high MSEs. They are not the best models to use for interpretation or accuracy. Furthermore, despite the popularity of the neural network, it is not the optimal model for determining the price of a diamond with my dataset. In addition, it is very hard to interpret. The tree-based models give both the best interpretability with the lowest MSE out of all the models. In particular, boosting gives the best results.

Conclusion: What determines the price of a diamond?

According to the importance matrix, It appears that the 4 most important variables are, from most to least important, 'y', 'carat', 'z', and 'x' followed by 'clarity'. This is contradictory to the popular belief that the 4 C's are what determine price. It is also worth noting the relative importances from the boosting tree:

```
summary(boost.diamonds)
```




```
##             var     rel.inf
## y             y 40.42405671
## carat     carat 38.47074016
## z             z  7.80831113
## clarity clarity  6.78968292
## color     color  3.07048927
## x             x  2.91004731
## cut         cut  0.25444831
## depth     depth  0.17868779
## table     table  0.09353641
```

I believe the importance of 'x', 'y', and 'z' and 'carat' is due to the fact that size matters. A bigger sized diamond will allow more light to come in, which allows more light to come into the diamond and shine. Furthermore, the 5th most important variable, 'clarity', allows the intensity of the light to be preserved as it goes inside of the diamond, so it is deserving of being the 5th most important factor. It appears that the 4 C's are overrated. There are only two factors that matter when determining the price of a diamond: its size and how bright it can shine. These two factors go hand-in-hand aswell.