Regression

Kathryn Kingsley KLK170230

In this notebook, I explore a data set regarding the prices of diamonds using linear regression, kNN, and decision trees. The data set had 53940 observations and 11 varaibles. Data can be found here:

`https://www.kaggle.com/shivam2503/diamonds`

I want to see if we can predict price accurately.

First

I import my data. It has almost 54k rows with 11 variables

```
df <- data.frame(read.csv("diamonds.csv"))
```

Clean the data up. Color, clarity, and cut are non-numeric, so I changed those to factors. Then I removed any NAs and useless columns

```
# remove NAs
df <- df[!(df$carat=="" | df$cut==""| df$color=="" | df$clarity==""|
             df$depth==""| df$table=="" | df$price==""| df$x=="" | df$y==""|
             df$z==""),]

df$color <- as.factor(df$color)
df$clarity <- as.factor(df$clarity)
df$cut <- as.factor(df$cut)

df$color <- as.numeric(df$color)
df$clarity <- as.numeric(df$clarity)
df$cut <- as.numeric(df$cut)

# remove useless columns
df <- df[ , c("carat", "cut", "color", "clarity", "depth", "table", "price",
              "x","y", "z")]
```

Next

I explore my data a bit- taking a look at some basic characteristics. Then I want to see what range of price and size data we're working with. (X, y, and z are length, width, depth respectively.

Finally I output all the correlations. There is a high correlation between price carat, length, width, and depth. (df$depth is total depth percentage; z is physical depth)

```
summary(df)
```

```
##      carat             cut            color          clarity     
##  Min.   :0.2000   Min.   :1.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:0.4000   1st Qu.:3.000   1st Qu.:2.000   1st Qu.:3.000  
##  Median :0.7000   Median :3.000   Median :4.000   Median :5.000  
##  Mean   :0.7979   Mean   :3.553   Mean   :3.594   Mean   :4.835  
##  3rd Qu.:1.0400   3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:6.000  
##  Max.   :5.0100   Max.   :5.000   Max.   :7.000   Max.   :8.000  
##      depth           table           price             x         
##  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000  
##  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710  
##  Median :61.80   Median :57.00   Median : 2401   Median : 5.700  
##  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731  
##  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540  
##  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740  
##        y                z        
##  Min.   : 0.000   Min.   : 0.000  
##  1st Qu.: 4.720   1st Qu.: 2.910  
##  Median : 5.710   Median : 3.530  
##  Mean   : 5.735   Mean   : 3.539  
##  3rd Qu.: 6.540   3rd Qu.: 4.040  
##  Max.   :58.900   Max.   :31.800
```

```
names(df)
```

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

```
head(df)
```

```
##   carat cut color clarity depth table price    x    y    z
## 1  0.23   3     2       4  61.5    55   326 3.95 3.98 2.43
## 2  0.21   4     2       3  59.8    61   326 3.89 3.84 2.31
## 3  0.23   2     2       5  56.9    65   327 4.05 4.07 2.31
## 4  0.29   4     6       6  62.4    58   334 4.20 4.23 2.63
## 5  0.31   2     7       4  63.3    58   335 4.34 4.35 2.75
## 6  0.24   5     7       8  62.8    57   336 3.94 3.96 2.48
```

```
tail(df)
```

```
##       carat cut color clarity depth table price    x    y    z
## 53935  0.72   4     1       3  62.7    59  2757 5.69 5.73 3.58
## 53936  0.72   3     1       3  60.8    57  2757 5.75 5.76 3.50
## 53937  0.72   2     1       3  63.1    55  2757 5.69 5.75 3.61
## 53938  0.70   5     1       3  62.8    60  2757 5.66 5.68 3.56
## 53939  0.86   4     5       4  61.0    58  2757 6.15 6.12 3.74
## 53940  0.75   3     1       4  62.2    55  2757 5.83 5.87 3.64
```

```
dim(df)
```

```
## [1] 53940    10
```

```
range(df$price)
```

```
## [1]   326 18823
```

```
range(df$x)
```

```
## [1]  0.00 10.74
```

```
range(df$y)
```

```
## [1]  0.0 58.9
```

```
range(df$z)
```

```
## [1]  0.0 31.8
```

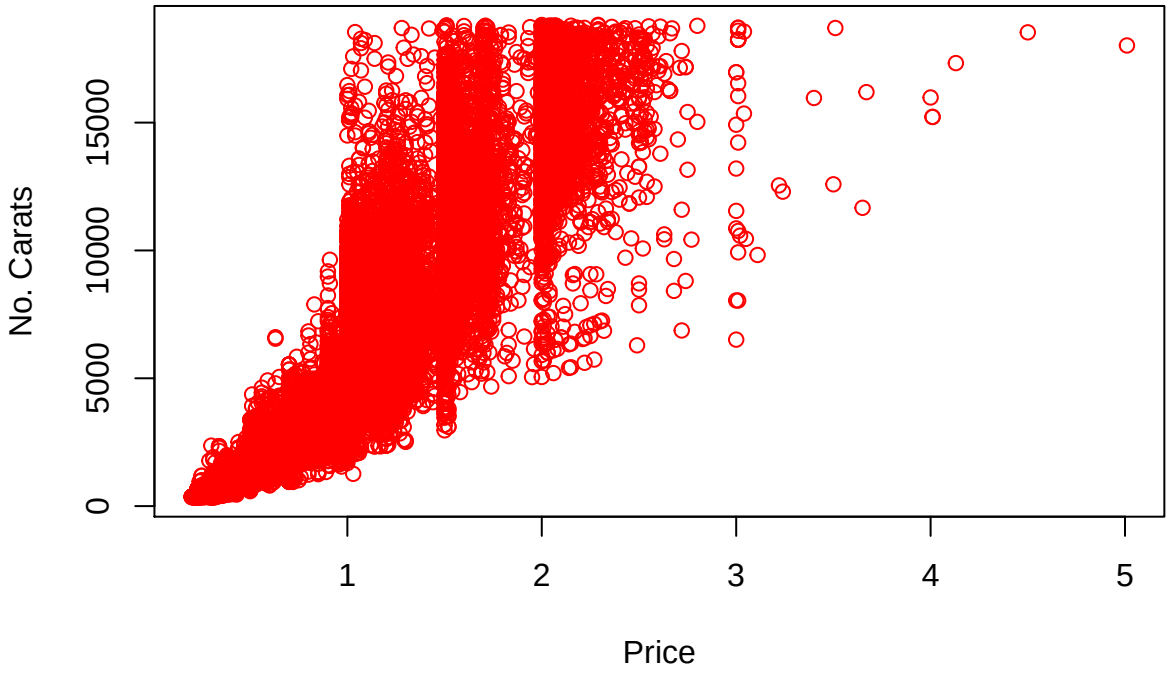
```
cor(df, use="complete")
```

```
##               carat           cut         color     clarity       depth
## carat    1.00000000  0.0171237362  0.2914367543 -0.21429037  0.02822431
## cut      0.01712374  1.0000000000  0.0003042479  0.02823537 -0.19424856
## color    0.29143675  0.0003042479  1.0000000000 -0.02779550  0.04727923
## clarity -0.21429037  0.0282353656 -0.0277954960  1.00000000 -0.05308011
## depth    0.02822431 -0.1942485626  0.0472792348 -0.05308011  1.00000000
## table    0.18161755  0.1503270263  0.0264652011 -0.08822266 -0.29577852
## price    0.92159130  0.0398602909  0.1725109282 -0.07153497 -0.01064740
## x        0.97509423  0.0223419276  0.2702866854 -0.22572144 -0.02528925
## y        0.95172220  0.0275720250  0.2635844027 -0.21761579 -0.02934067
## z        0.95338738  0.0020373568  0.2682268757 -0.22426307  0.09492388
##                table       price           x           y           z
## carat    0.18161755  0.92159130  0.97509423  0.95172220  0.953387381
## cut      0.15032703  0.03986029  0.02234193  0.02757203  0.002037357
## color    0.02646520  0.17251093  0.27028669  0.26358440  0.268226876
## clarity -0.08822266 -0.07153497 -0.22572144 -0.21761579 -0.224263069
## depth   -0.29577852 -0.01064740 -0.02528925 -0.02934067  0.094923882
## table    1.00000000  0.12713390  0.19534428  0.18376015  0.150928692
## price    0.12713390  1.00000000  0.88443516  0.86542090  0.861249444
## x        0.19534428  0.88443516  1.00000000  0.97470148  0.970771799
## y        0.18376015  0.86542090  0.97470148  1.00000000  0.952005716
## z        0.15092869  0.86124944  0.97077180  0.95200572  1.000000000
```

I want to visually see these correlations, so I made a few maps here. One to see price as a function of carats. One to see the spread of clarity. Three more quick ones to see if length, width, or depth has the greatest impact on price.

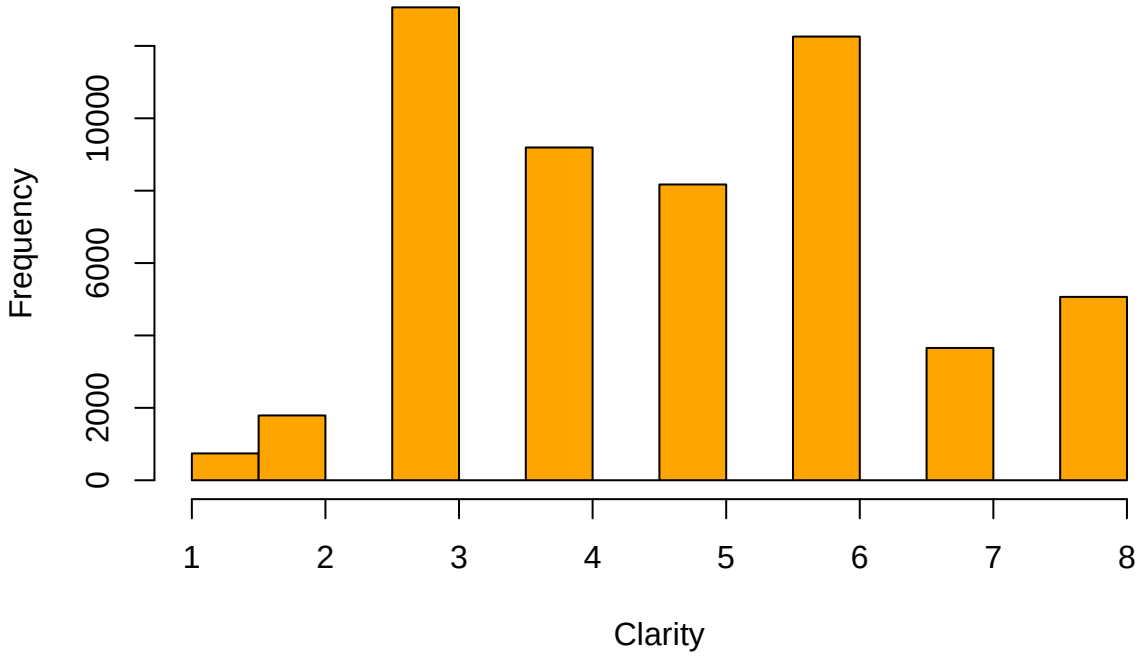
```
# hist(df$price, col="slategray", main="Range of Price",xlab="Price")
plot(df$price~ df$carat,col="red", main="Figure 1.1", xlab="Price",
     ylab="No. Carats")
```

Figure 1.1



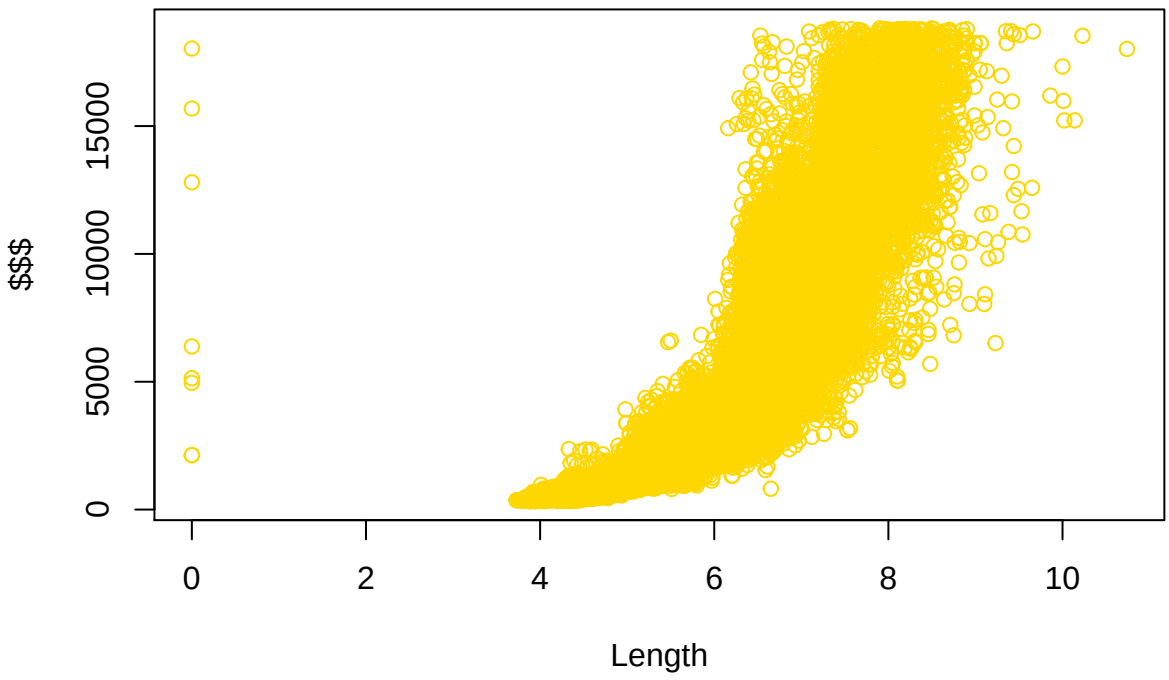

```
hist(df$clarity, col="orange", main="Figure 1.2", xlab="Clarity",
     ylab="Frequency")
```

Figure 1.2



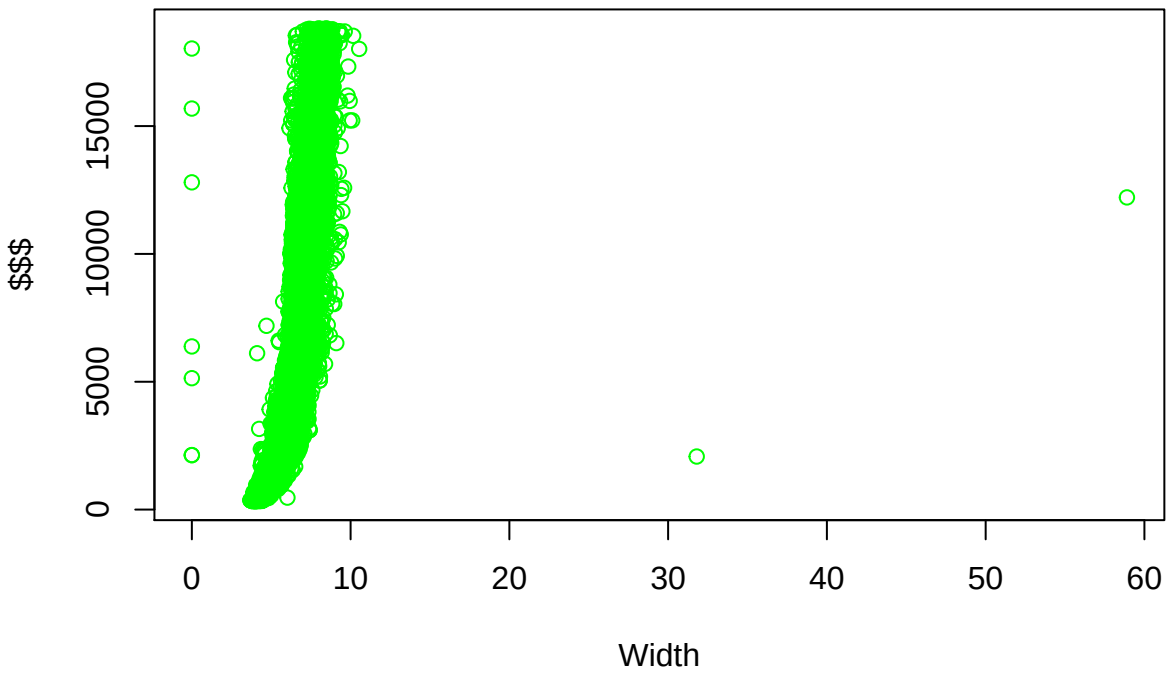

```
plot(df$price~df$x, col="gold",  main="Figure 1.3",xlab="Length", ylab="$$$")
```

Figure 1.3



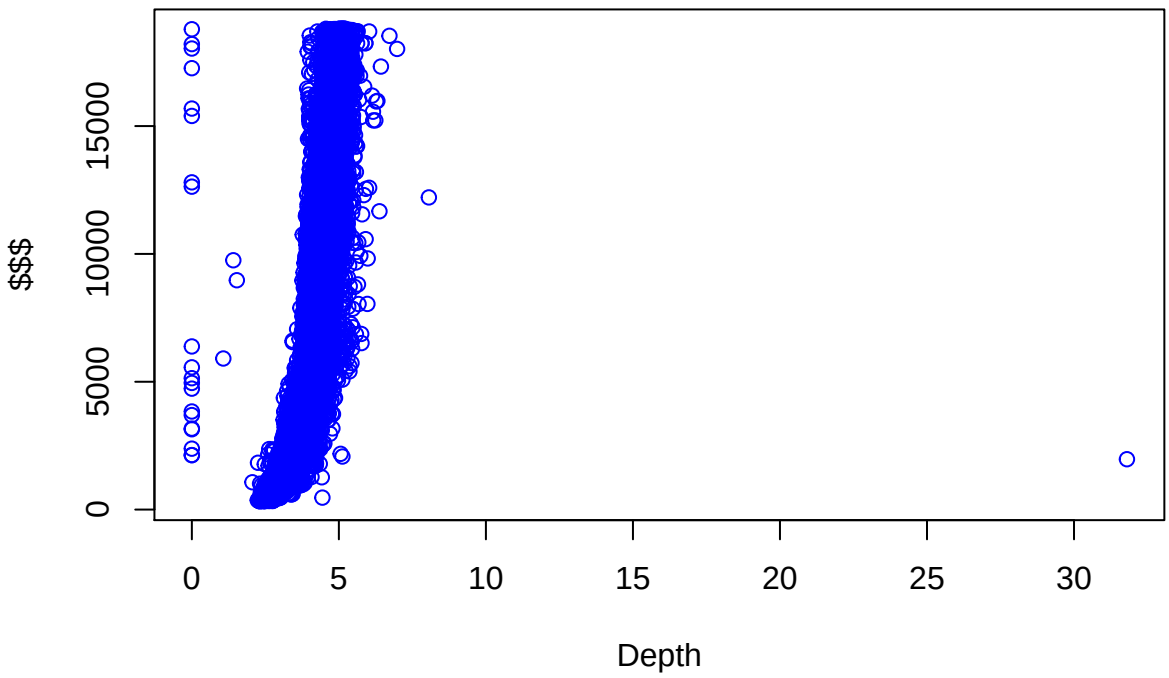

```
plot(df$price~df$y, col="green",  main="Figure 1.4",xlab="Width", ylab="$$$")
```

Figure 1.4




```
plot(df$price~df$z, col="blue",  main="Figure 1.5",xlab="Depth", ylab="$$$")
```

Figure 1.5



X,y, and z increase the price almost exponentially from the looks of the graphs, and all have a few outliers. Length appears to have a more steady pattern, so I anticipate this to be the best predictor of the 3 size variables.

Then

Separate into train and test sets

```
set.seed(1234)
i <- sample(1:nrow(df), nrow(df)*0.75, replace = FALSE)
train1 <- df[i,]
test1 <- df[-i,]
```

Linear Regression

In order to not overfit the model, I only used carat as the predictor.

```
start_time1 <- Sys.time()
lm1 <- lm(price~ carat, data=train1)
end_time1 <- Sys.time()
final_time1 <- end_time1 - start_time1
summary(lm1)
```

```
##
```

```
## Call:
## lm(formula = price ~ carat, data = train1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14544.5   -805.8    -16.4    544.9  12727.6
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2273.21      15.08  -150.8   <2e-16 ***
## carat        7776.57      16.26   478.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1545 on 40453 degrees of freedom
## Multiple R-squared:  0.8496, Adjusted R-squared:  0.8496
## F-statistic: 2.286e+05 on 1 and 40453 DF,  p-value: < 2.2e-16
```

The model looks good with a very low P value.

Now, I'll predict on the test data.

```
pred1 <- predict(lm1, newdata=test1)
```

Calculate and output the correlation, MSE, RMSE, and time.

```
cor1 <- cor(pred1, test1$price)
mse1<- mean((pred1 - test1$price)^2)
rmse1 <- sqrt(mse1)
# print out
print(paste("LinReg 1 Time taken: ", final_time1))
```

```
## [1] "LinReg 1 Time taken:  0.0159578323364258"
```

```
print(paste("LinReg 1 Correlation: ", cor1))
```

```
## [1] "LinReg 1 Correlation:  0.921113090335987"
```

```
print(paste("LinReg 1 MSE: ", mse1))
```

```
## [1] "LinReg 1 MSE:  2427828.48549402"
```

```
print(paste("LinReg 1 RMSE: ",rmse1))
```

```
## [1] "LinReg 1 RMSE:  1558.14905753398"
```

While the algorithm is fast, the model prediction was off by about $1,560.00. That's quite a bit. It has a strong correlation however, indicating a good model. I tried with more predictors just to check, and the change in RMSE was only about $20.00.

I want to see if I add a few more dimensions, if I can get that correlation a bit higher.

```
start_time4 <- Sys.time()
lm2 <- lm(price~ carat+x+y, data=train1)
end_time4 <- Sys.time()
final_time4 <- end_time4 - start_time4
summary(lm2)
```

```
##
## Call:
## lm(formula = price ~ carat + x + y, data = train1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17180.9   -644.6    -15.6    359.6  12992.2
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2066.61     120.62  17.133  < 2e-16 ***
## carat       10356.50      72.87 142.118  < 2e-16 ***
## x           -1204.61      39.51 -30.492  < 2e-16 ***
## y              88.33      25.82   3.421 0.000624 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1521 on 40451 degrees of freedom
## Multiple R-squared:  0.8545, Adjusted R-squared:  0.8544
## F-statistic: 7.916e+04 on 3 and 40451 DF,  p-value: < 2.2e-16
```

Here are the results.

```
pred4 <- predict(lm2, newdata=test1)
cor4 <- cor(pred4, test1$price)
mse4<- mean((pred4 - test1$price)^2)
rmse4 <- sqrt(mse4)
# print out
print(paste("LinReg 2 Time taken: ", final_time4))
```

```
## [1] "LinReg 2 Time taken:  0.0149600505828857"
```

```
print(paste("LinReg 2 Correlation: ", cor4))
```

```
## [1] "LinReg 2 Correlation:  0.922279229690552"
```

```
print(paste("LinReg 2 MSE: ", mse4))
```

```
## [1] "LinReg 2 MSE:  2394410.11341317"
```

```
print(paste("LinReg 2 RMSE: ",rmse4))
```

```
## [1] "LinReg 2 RMSE:  1547.38815861217"
```

Adding more dimensions had very little impact on correlation and RMSE.

kNN

First, I split into train and test sets. Then I remove the labels.

```
library(caret) # needed for knnreg
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(1234)
i <- sample(1:nrow(df), nrow(df)*0.75, replace = FALSE)
train2 <- df[i,]
test2 <- df[-i,]

train2Labs <- train2$price
train2<- train2[ , c("carat", "cut", "color", "clarity", "depth", "table",
                     "x","y", "z")]

test2Labs <- test2$price
test2<- test2[ , c("carat", "cut", "color", "clarity", "depth", "table",
                   "x","y", "z")]
```

Here I actually use knnreg. I build and predict at the same time. Output calculations here.

```
start_time2 <- Sys.time()
fit1 <- knnreg(train2, train2Labs, k=8)
end_time2 <- Sys.time()
final_time2 <- end_time2 - start_time2
pred2 <- predict(fit1, test2)
cor2 <- cor(pred1, test2Labs)
mse2 <- mean((pred2-test2Labs)^2)
rmse2 <- sqrt(mse2)

# print out
print(paste("kNN Time taken: ", final_time2))
```

```
## [1] "kNN Time taken:  0.00199484825134277"
```

```
print(paste("kNN Correlation: ", cor2))
```

```
## [1] "kNN Correlation:  0.921113090335987"
```

```
print(paste("kNN MSE: ", mse2))
```

```
## [1] "kNN MSE:  795444.768339987"
```

```
print(paste("kNN RMSE: ",rmse2))
```

```
## [1] "kNN RMSE:  891.87710383213"
```

Even with a k value of 1, this was much better. The correlation was the same at 92% but the RMSE went down by about $500. That's down nearly 33%! After playing with the k value a bit, I noticed the higher I went, the better kNN performed. With a k value of 8, I could lower the RMSE by almost $700. Any higher than 8, and the RMSE started to rise again, but the correlation stayed relatively constant at 92%. On top of that, the time was very fast! Only a fraction of the time for linear regression.

Decision Tree

Now let's try a decision tree on our data. I can use the same train and test set as linear regression

```
library(tree) # needed for decision trees
```

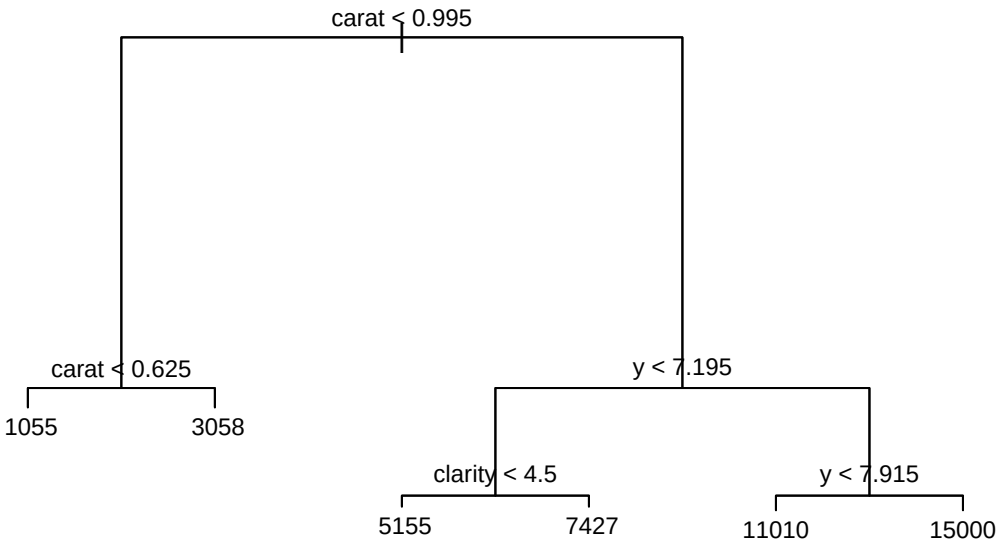
```
## Warning: package 'tree' was built under R version 4.0.4
```

```
start_time3 <- Sys.time()
tree1 <- tree(price~., data=train1)
end_time3 <- Sys.time()
final_time3 <- end_time3 - start_time3
summary(tree1)
```

```
##
## Regression tree:
## tree(formula = price ~ ., data = train1)
## Variables actually used in tree construction:
## [1] "carat"   "y"       "clarity"
## Number of terminal nodes:  6
## Residual mean deviance:  2023000 = 8.184e+10 / 40450
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -9953.0  -535.2  -172.9     0.0   512.1 13540.0
```

I see we have 6 leaf nodes, and carat, y, and clarity were found to be the most impactful. Here is a visualization of the tree.

```
plot(tree1)
text(tree1, cex=0.75, pretty=0)
```



Predict on the test set. I don't use type = "class" here because this is regression and not classification.

```
pred3 <- predict(tree1, newdata=test1)
```

Calculate and output the correlation, MSE, RMSE, and time.

```
cor3 <- cor(pred3, test1$price)
mse3 <- mean((pred3-test1$price)^2)
rmse3 <- sqrt(mse2)

# print
print(paste("Time taken for DT: ", final_time3))
```

```
## [1] "Time taken for DT:  0.180520057678223"
```

```
print(paste("DT Correlation: ", cor3))
```

```
## [1] "DT Correlation:  0.936119146202091"
```

```
print(paste("DT MSE: ", mse3))
```

```
## [1] "DT MSE:  1980256.5054407"
```

```
print(paste("DT RMSE: ",rmse3))
```

```
## [1] "DT RMSE:  891.87710383213"
```

The decision tree algorithm was slightly slower than the others at 0.19, but it was the simplest for me to make and the most accurate. On the first try, with all possible predictors, it chose the best ones and had almost a 94% accuracy rate. In addition to that, the RMSE is as low as I could get it with kNN, and that required a lot of trial and error!

Results Analysis

Linear regression performed well, but since linear regression has such a high bias, I could do little to raise the correlation. It was, however, fast and simple to implement once the data was prepped and cleaned.

KNN gave better results than linear regression, but required some fussing. I had tried to implement k- fold cross validation, but it proved to be challenging, so I went to good old fashioned trial and error. I settled on k=8 which lowered RMSE, but correlation refused to budge still. I think since there were 9 possible variables to choose from, this limited its performance.

Lastly I used the Decision Tree algorithm. This had the the best regression results hands down! Even though it was a tiny bit slower than linear regression and kNN, it was super easy to work with and provided the best correlation and RMSE without any effort from me. I think this is because, as the graphs indicated, the underlying function for the model wasn't as linear so much as exponential, so it could predict trends more accurately.