# Project #2 - Regression

## Initial Setup

```
df <- read.csv("diamonds.csv", header=TRUE, stringsAsFactors = FALSE)
str(df)
```

```
## 'data.frame':    53940 obs. of  8 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : chr  "Ideal" "Premium" "Good" "Premium" ...
##  $ clarity: chr  "SI2" "SI1" "VS1" "VS2" ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ 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 ...
```

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##    carat      cut clarity    depth    price        x        y        z
##        0        0        0        0        0        0        0        0
```

## Data Cleaning

Links to data-set and documentation: 1. https://vincentarelbundock.github.io/Rdatasets/datasets.html 2. https://vincentarelbundock.github.io/Rdatasets/doc/ggplot2/diamonds.html

This is dataset is called "diamonds" and contains the prices and other attributes of almost 54,000 diamonds.

The columns "table", "color", and "index" were manually removed from the CSV file as the data was not necessary for obtaining our final solutions.

This chunk of code will convert two columns into factors, and will display the new structure of the data frame.

```
df$cut <- as.factor(df$cut)
df$clarity <- as.factor(df$clarity)
str(df)
```

```
## 'data.frame':    53940 obs. of  8 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Factor w/ 5 levels "Fair","Good",..: 3 4 2 4 2 5 5 5 1 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 ...
```

```
##  $ 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 ...
```

# Data Exploration

```r
head(df)
```

```
##   carat       cut clarity depth price    x    y    z
## 1  0.23     Ideal     SI2  61.5   326 3.95 3.98 2.43
## 2  0.21   Premium     SI1  59.8   326 3.89 3.84 2.31
## 3  0.23      Good     VS1  56.9   327 4.05 4.07 2.31
## 4  0.29   Premium     VS2  62.4   334 4.20 4.23 2.63
## 5  0.31      Good     SI2  63.3   335 4.34 4.35 2.75
## 6  0.24 Very Good    VVS2  62.8   336 3.94 3.96 2.48
```

```r
summary(df)
```

```
##      carat                 cut           clarity          depth      
##  Min.   :0.2000   Fair     : 1610   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good     : 4906   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Ideal    :21551   SI2    : 9194   Median :61.80  
##  Mean   :0.7979   Premium  :13791   VS1    : 8171   Mean   :61.75  
##  3rd Qu.:1.0400   Very Good:12082   VVS2   : 5066   3rd Qu.:62.50  
##  Max.   :5.0100                     VVS1   : 3655   Max.   :79.00  
##                                     (Other): 2531                  
##      price             x                y                z         
##  Min.   :  326   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
##  Median : 2401   Median : 5.700   Median : 5.710   Median : 3.530  
##  Mean   : 3933   Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
##  3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
##  Max.   :18823   Max.   :10.740   Max.   :58.900   Max.   :31.800  
## 
```

```r
contrasts(df$cut)
```

```
##           Good Ideal Premium Very Good
## Fair         0     0       0         0
## Good         1     0       0         0
## Ideal        0     1       0         0
## Premium      0     0       1         0
## Very Good    0     0       0         1
```

```r
contrasts(df$clarity)
```

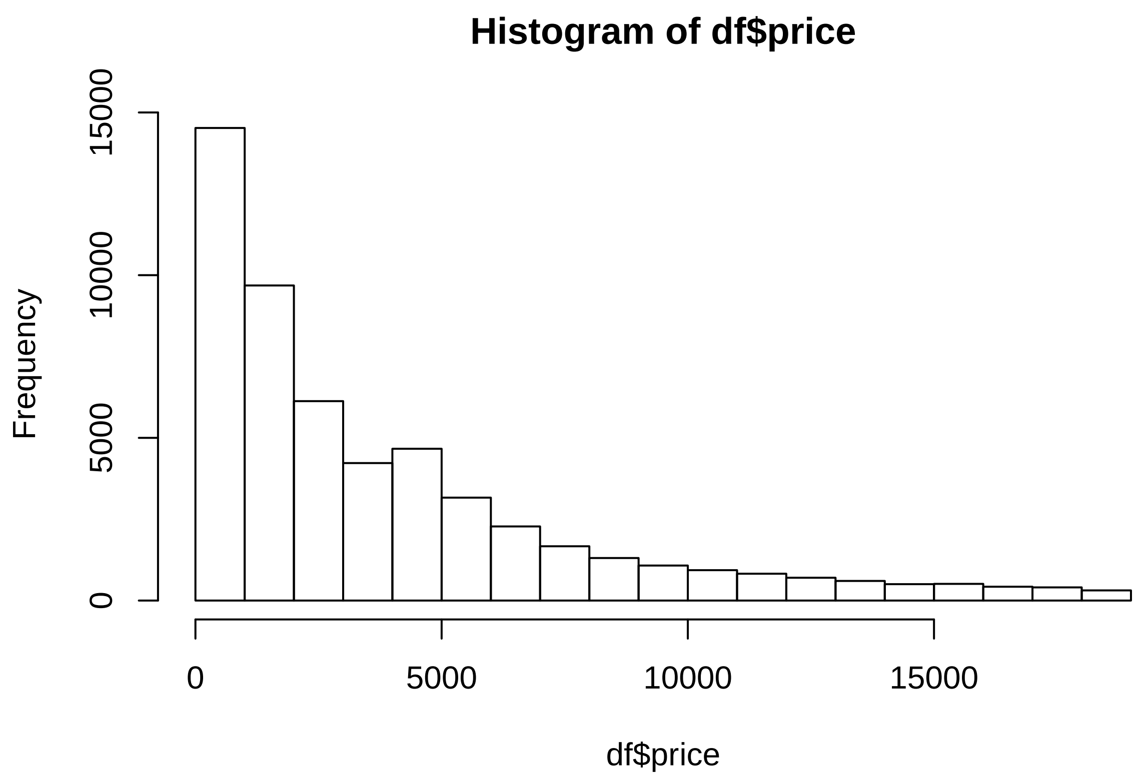
```
##      IF SI1 SI2 VS1 VS2 VVS1 VVS2
## I1    0   0   0   0   0    0    0
```

```
## IF     1   0   0   0   0    0    0
## SI1    0   1   0   0   0    0    0
## SI2    0   0   1   0   0    0    0
## VS1    0   0   0   1   0    0    0
## VS2    0   0   0   0   1    0    0
## VVS1   0   0   0   0   0    1    0
## VVS2   0   0   0   0   0    0    1
```

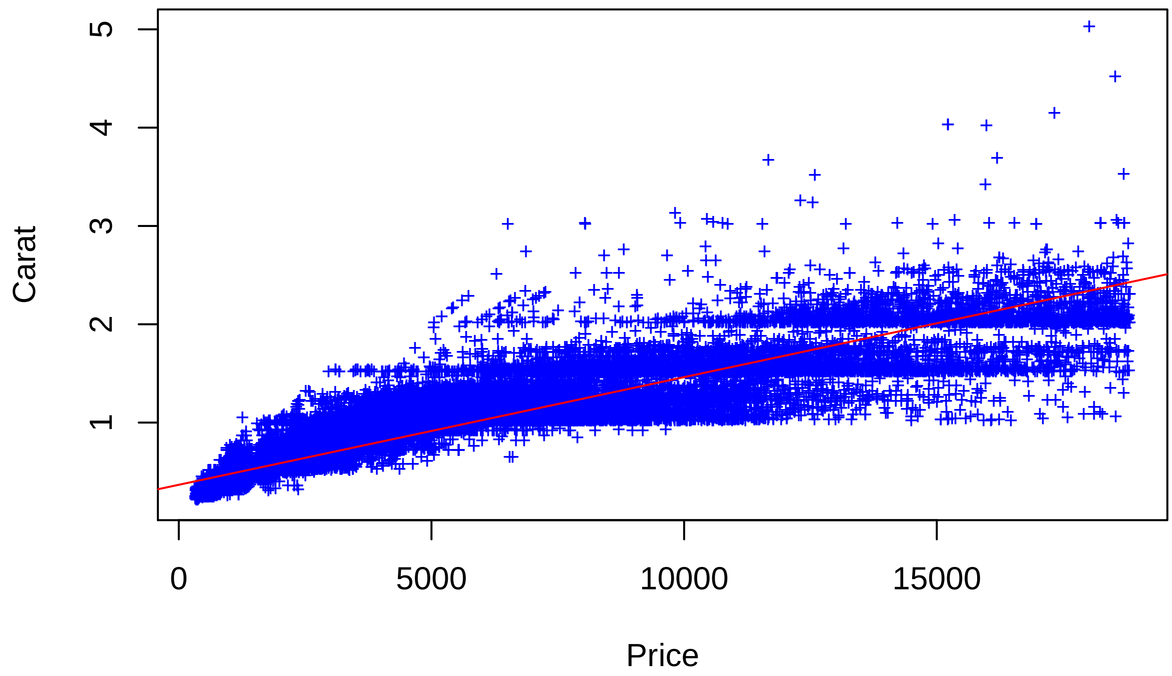
```
cor(df$x, df$y)
```

```
## [1] 0.9747015
```

```
hist(df$price)
```




```
plot(df$price, df$carat, pch='+', cex=0.75, col="blue", xlab="Price", ylab="Carat")
abline(lm(df$carat ~ df$price), col="red")
```




## Linear Regression

Linear regression was performed on this data frame because it is relatively simple and powerful, and provides strong information of the relationship held between two attribute values within a data frame. The values used in this model are "price" and "carat", with hopes to learn more of the relationship held between the two variables.

Upon completion of the linear regression model, we can see that there is a strong relationship held between the value of price and carat of a diamond. This can be seen as there is a high correlation value, and the plot follows the same pattern as the predicted abline values. This represents that the predicted data trend is accurate to the actual results depicted from the data frame.

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

lm1 <- lm(formula = price ~ carat, data=train)
summary(lm1)
```

```
##
## Call:
## lm(formula = price ~ carat, data = train)
##
```

```
## 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
```

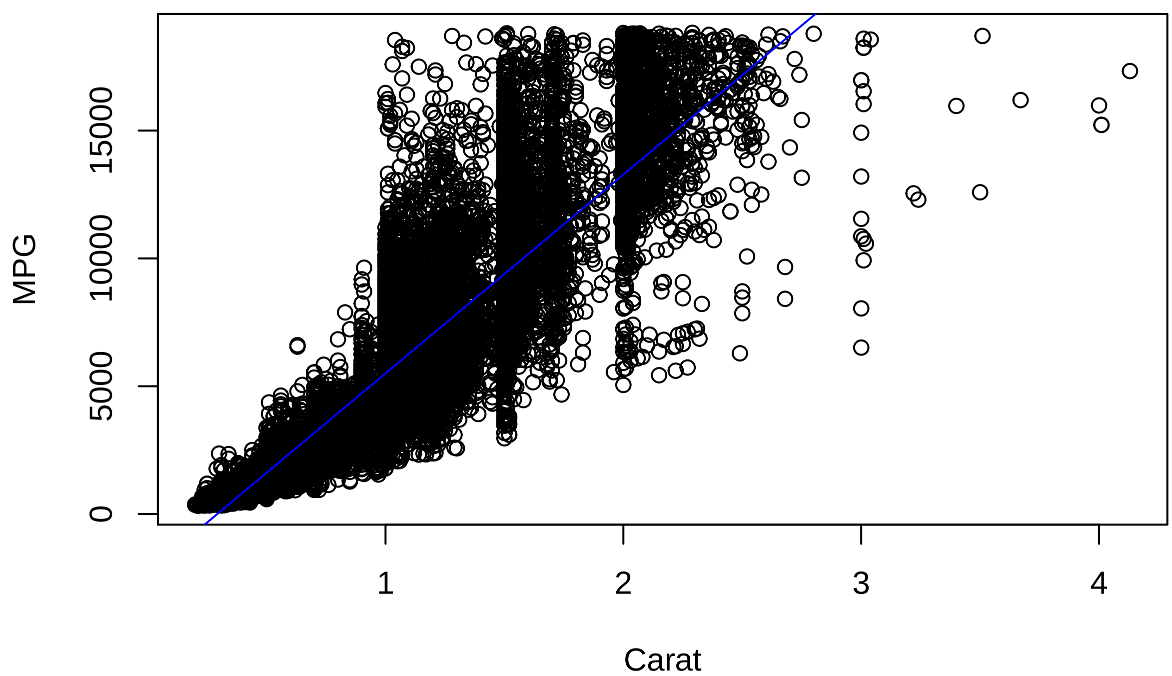
```r
pred <- predict(lm1, newdata= test)
cor_lm <- cor(pred, test$price)
rmse_lm <- sqrt(mean(pred-test$price)^2)
print(paste("cor = ", cor_lm))
```

```
## [1] "cor =  0.921113090335987"
```

```r
rmse <- sqrt(mean((pred - lm1$residuals)^2))
print(paste("RMSE: ", rmse))
```

```
## [1] "RMSE:  5633.40004929194"
```

```r
plot(train$price~train$carat, xlab="Carat", ylab = "MPG")
abline(lm1, col="blue")
```




```
predict1 <- predict(lm1, data.frame(carat = 98))
print(paste(predict1))
```

```
## [1] "759830.847232921"
```

## KNN Regression

KNN Regression was performed on this data frame because it provides intuitive information collected from the data, without actually forming varoius models. Rather, all of the training observations will be made in memory for simpler access. Then, once a new observation requires evaluation, the algorithm can reference in memory and find the closest neighbors to the observation.

Upon complete execution of the KNN regression algorithm, we can see that there is a strong correlation between the data and a lower MSE than the Linear Regression algorithm. As the value of k increases, there is a greater correlation and lower RMSE values, showing that k=15 provides the best results over the other values.

```
library(caret)
```

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

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

```r
lm_knn <- lm(price ~., data=train)
summary(lm_knn)
```

```
##
## Call:
## lm(formula = price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14995.3   -556.2   -111.2    399.4  11500.6
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    495.226    381.983   1.296    0.195
## carat        10667.308     62.770 169.942   <2e-16 ***
## cutGood        576.491     43.138  13.364   <2e-16 ***
## cutIdeal       899.558     40.189  22.383   <2e-16 ***
## cutPremium     741.113     41.130  18.019   <2e-16 ***
## cutVery Good   738.697     40.819  18.097   <2e-16 ***
## clarityIF     5091.976     65.415  77.841   <2e-16 ***
## claritySI1    3481.432     55.941  62.235   <2e-16 ***
## claritySI2    2638.998     56.215  46.945   <2e-16 ***
## clarityVS1    4301.875     57.072  75.377   <2e-16 ***
## clarityVS2    4079.027     56.232  72.540   <2e-16 ***
## clarityVVS1   4713.401     60.514  77.889   <2e-16 ***
## clarityVVS2   4751.697     58.801  80.810   <2e-16 ***
## depth          -67.566      5.351 -12.627   <2e-16 ***
## x             -959.553     39.762 -24.132   <2e-16 ***
## y                6.942     21.680   0.320    0.749
## z               -4.071     39.117  -0.104    0.917
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1261 on 40438 degrees of freedom
## Multiple R-squared:  0.8999, Adjusted R-squared:  0.8999
## F-statistic: 2.273e+04 on 16 and 40438 DF,  p-value: < 2.2e-16
```

```r
train_knn <- train[, c(1,4,6)]
test_knn <- test[, c(1,4,6)]
means <- sapply(train_knn, mean)
stdevs <- sapply(train_knn, sd)
train_knn <- scale(train_knn, center = means, scale = stdevs)
test_knn <- scale(test_knn, center=means, scale=stdevs)

for (i in c(5, 10, 15)){
  fit_knn <- knnreg(train_knn, train$price, k=i)
  pred_knn <- predict(fit_knn, test_knn)
  print(paste("k= ", i))
  print(paste("cor = ", cor(pred_knn, test$price)))
  print(paste("rmse = ", sqrt(mean(pred_knn-test$price)^2)))
}
```

```
## [1] "k=  5"
```

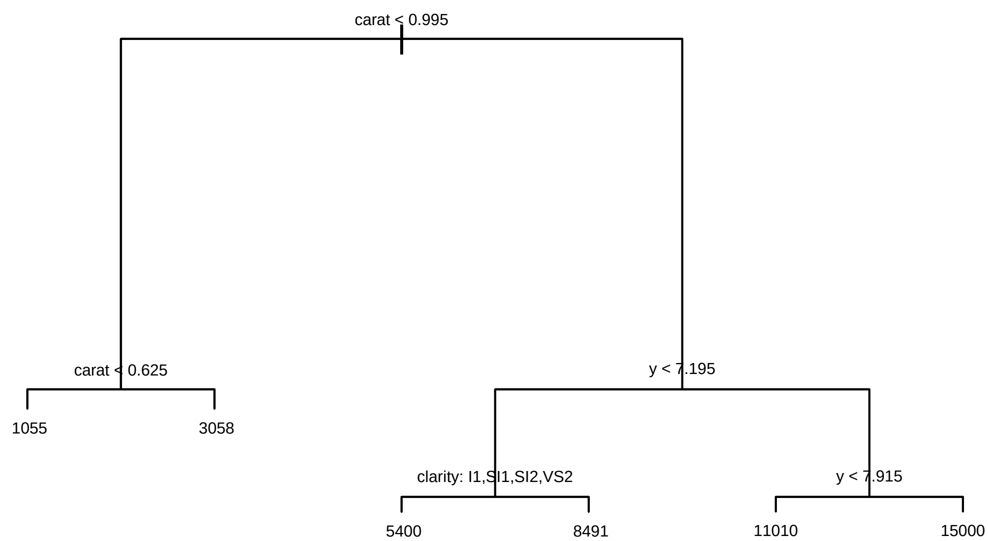
```
## [1] "cor =  0.931917047384187"
## [1] "rmse =  22.184064006214"
## [1] "k=  10"
## [1] "cor =  0.936320845235787"
## [1] "rmse =  15.220526195623"
## [1] "k=  15"
## [1] "cor =  0.937240614888268"
## [1] "rmse =  18.5050260177871"
```

## Decision Tree Regression

Decision Tree Regression was performed on this data frame because it allows for the splitting of observations into proper partitions, allowing for the data to be placed into particular boundaries of which they belong. Although this algorithm is not the best when it comes to performance, it presents the data in a very strong and comprehensible manner, which is extremely beneficial and obtaining a visual understanding of the data frame.

Upon complete execution of the Decision Tree Regression algorithm, we can see that there is a strong correlation between the predicted and actual data collected with a very low RMSE value. We can also see that the price of a diamond relies heavily on the amount of carats, and the width of each diamond.

```
library(tree)
tree1 <- tree(price ~., data=train)
plot(tree1)
text(tree1, cex=0.5, pretty=0)
```



```r
pred_tree <- predict(tree1, newdata=test)
cor_tree <- cor(pred_tree, test$price)
print(paste("cor = ", cor_tree))
```

```
## [1] "cor =  0.938679943379147"
```

```r
rmse_tree <- sqrt(mean(pred_tree - test$price)^2)
print(paste("rmse = ", rmse_tree))
```

```
## [1] "rmse =  8.53082450968764"
```

## XGBoost

XGBoost was performed on this data frame because it is extremely scalable and runs very fast when compared to other algorithms. XGBoost focuses on the speed of execution, while still providing optimal accuracy and clarity.

Upon execution of XGBoost, we can see that after 200 rounds of learning, the algorithm is able to predict the data with a correlation value of 99.9% and an extremely low RMSE. This represents the ability for the algorithm to learn successfully from the data and make accurate predictions over time.

```r
library(xgboost)
train_mat <- data.matrix(train[, -3])
xmod <- xgboost(data=train_mat, label=train$price, nrounds=200, objective='reg:squarederror')
```

```
## [1]  train-rmse:3921.111084
## [2]  train-rmse:2747.712158
## [3]  train-rmse:1925.410400
## [4]  train-rmse:1349.290283
## [5]  train-rmse:945.723511
## [6]  train-rmse:663.073242
## [7]  train-rmse:465.179718
## [8]  train-rmse:326.706665
## [9]  train-rmse:229.859314
## [10] train-rmse:162.185806
## [11] train-rmse:114.942627
## [12] train-rmse:82.147812
## [13] train-rmse:59.482750
## [14] train-rmse:43.879372
## [15] train-rmse:33.491718
## [16] train-rmse:26.810938
## [17] train-rmse:22.706810
## [18] train-rmse:19.904284
## [19] train-rmse:18.111624
## [20] train-rmse:17.020054
## [21] train-rmse:16.441479
## [22] train-rmse:15.653096
## [23] train-rmse:15.189156
## [24] train-rmse:14.722426
## [25] train-rmse:14.396504
## [26] train-rmse:14.044674
```

```
## [27] train-rmse:13.916530
## [28] train-rmse:13.535176
## [29] train-rmse:13.481875
## [30] train-rmse:13.349463
## [31] train-rmse:13.144892
## [32] train-rmse:13.084277
## [33] train-rmse:13.036746
## [34] train-rmse:12.877256
## [35] train-rmse:12.796413
## [36] train-rmse:12.744702
## [37] train-rmse:12.708138
## [38] train-rmse:12.641765
## [39] train-rmse:12.528550
## [40] train-rmse:12.123793
## [41] train-rmse:11.956040
## [42] train-rmse:11.658518
## [43] train-rmse:11.432349
## [44] train-rmse:11.262834
## [45] train-rmse:11.051462
## [46] train-rmse:10.876549
## [47] train-rmse:10.774953
## [48] train-rmse:10.511905
## [49] train-rmse:10.428731
## [50] train-rmse:10.271671
## [51] train-rmse:10.228831
## [52] train-rmse:10.017670
## [53] train-rmse:9.917238
## [54] train-rmse:9.808461
## [55] train-rmse:9.770369
## [56] train-rmse:9.716631
## [57] train-rmse:9.644032
## [58] train-rmse:9.632558
## [59] train-rmse:9.591115
## [60] train-rmse:9.572762
## [61] train-rmse:9.541162
## [62] train-rmse:9.529919
## [63] train-rmse:9.489415
## [64] train-rmse:9.458892
## [65] train-rmse:9.330058
## [66] train-rmse:9.181149
## [67] train-rmse:9.091463
## [68] train-rmse:8.944582
## [69] train-rmse:8.839486
## [70] train-rmse:8.733828
## [71] train-rmse:8.688491
## [72] train-rmse:8.670526
## [73] train-rmse:8.579525
## [74] train-rmse:8.520858
## [75] train-rmse:8.471761
## [76] train-rmse:8.456687
## [77] train-rmse:8.446122
## [78] train-rmse:8.384457
## [79] train-rmse:8.350221
## [80] train-rmse:8.334531
```

```
## [81] train-rmse:8.232047
## [82] train-rmse:8.131912
## [83] train-rmse:8.030093
## [84] train-rmse:7.929210
## [85] train-rmse:7.825773
## [86] train-rmse:7.730723
## [87] train-rmse:7.718632
## [88] train-rmse:7.683204
## [89] train-rmse:7.676072
## [90] train-rmse:7.548990
## [91] train-rmse:7.446944
## [92] train-rmse:7.346314
## [93] train-rmse:7.264205
## [94] train-rmse:7.236125
## [95] train-rmse:7.190091
## [96] train-rmse:7.125730
## [97] train-rmse:7.113976
## [98] train-rmse:7.043576
## [99] train-rmse:7.018410
## [100]    train-rmse:6.996046
## [101]    train-rmse:6.891916
## [102]    train-rmse:6.833212
## [103]    train-rmse:6.745467
## [104]    train-rmse:6.677728
## [105]    train-rmse:6.649980
## [106]    train-rmse:6.572954
## [107]    train-rmse:6.500912
## [108]    train-rmse:6.449457
## [109]    train-rmse:6.384882
## [110]    train-rmse:6.329360
## [111]    train-rmse:6.244195
## [112]    train-rmse:6.239938
## [113]    train-rmse:6.225387
## [114]    train-rmse:6.177069
## [115]    train-rmse:6.108219
## [116]    train-rmse:6.069561
## [117]    train-rmse:6.019928
## [118]    train-rmse:5.959866
## [119]    train-rmse:5.899639
## [120]    train-rmse:5.865479
## [121]    train-rmse:5.797110
## [122]    train-rmse:5.746074
## [123]    train-rmse:5.740941
## [124]    train-rmse:5.717888
## [125]    train-rmse:5.715441
## [126]    train-rmse:5.711261
## [127]    train-rmse:5.645539
## [128]    train-rmse:5.636548
## [129]    train-rmse:5.624739
## [130]    train-rmse:5.621527
## [131]    train-rmse:5.611495
## [132]    train-rmse:5.596295
## [133]    train-rmse:5.540775
## [134]    train-rmse:5.489931
```

```
## [135]	train-rmse:5.419221
## [136]	train-rmse:5.377131
## [137]	train-rmse:5.343208
## [138]	train-rmse:5.277297
## [139]	train-rmse:5.264592
## [140]	train-rmse:5.216985
## [141]	train-rmse:5.108963
## [142]	train-rmse:5.035273
## [143]	train-rmse:5.026814
## [144]	train-rmse:5.011512
## [145]	train-rmse:4.993924
## [146]	train-rmse:4.977694
## [147]	train-rmse:4.936043
## [148]	train-rmse:4.895182
## [149]	train-rmse:4.840315
## [150]	train-rmse:4.804748
## [151]	train-rmse:4.782284
## [152]	train-rmse:4.762999
## [153]	train-rmse:4.758668
## [154]	train-rmse:4.738183
## [155]	train-rmse:4.714849
## [156]	train-rmse:4.702100
## [157]	train-rmse:4.690609
## [158]	train-rmse:4.654119
## [159]	train-rmse:4.643760
## [160]	train-rmse:4.616132
## [161]	train-rmse:4.610760
## [162]	train-rmse:4.604997
## [163]	train-rmse:4.587253
## [164]	train-rmse:4.569396
## [165]	train-rmse:4.550006
## [166]	train-rmse:4.542857
## [167]	train-rmse:4.489128
## [168]	train-rmse:4.442485
## [169]	train-rmse:4.374753
## [170]	train-rmse:4.327721
## [171]	train-rmse:4.324047
## [172]	train-rmse:4.314894
## [173]	train-rmse:4.298628
## [174]	train-rmse:4.272822
## [175]	train-rmse:4.227800
## [176]	train-rmse:4.196464
## [177]	train-rmse:4.190054
## [178]	train-rmse:4.186926
## [179]	train-rmse:4.177448
## [180]	train-rmse:4.175037
## [181]	train-rmse:4.141765
## [182]	train-rmse:4.120569
## [183]	train-rmse:4.106136
## [184]	train-rmse:4.092699
## [185]	train-rmse:4.058883
## [186]	train-rmse:3.984486
## [187]	train-rmse:3.933908
## [188]	train-rmse:3.901233
```

```
## [189]	train-rmse:3.875955
## [190]	train-rmse:3.845248
## [191]	train-rmse:3.802611
## [192]	train-rmse:3.778607
## [193]	train-rmse:3.757210
## [194]	train-rmse:3.739088
## [195]	train-rmse:3.698071
## [196]	train-rmse:3.666694
## [197]	train-rmse:3.627323
## [198]	train-rmse:3.611624
## [199]	train-rmse:3.606393
## [200]	train-rmse:3.601969
```

```
test_mat <- data.matrix(test[, -3])
pred_x <- predict(xmod, newdata=test_mat)

corx <- cor(pred_x, test$price)
rmsx <- sqrt(mean(pred_x-test$price)^2)
print(paste("cor = ", corx))
```

```
## [1] "cor =  0.999999445487892"
```

```
print(paste("rmse = ", rmsx))
```

```
## [1] "rmse =  0.0632218284168463"
```

# Result Analysis

The algorithms ranked best to worst for this data frame are: 1. KNN 2. Decision Tree 3. Linear Regression

The reason behind these rankings are based upon the fact that the RMSE and correlation values for the KNN are the strongest, followed by the decision tree output, then the linear regression model values.

I believe the reason KNN was the best algorithm out of the three is because it was accurately able to depict the values within the data frame and test out different data values that would best fit the data, rather than having a single instance observation.

Overall, this script was able to learn the relationship between the price of a diamond based upon it's amount of carats, cut, clarity, depth, length, and width. With the final results, we are able to determine that the price of a diamond has the strongest relationship with the amount of carats inside, and the width of the diamond when measured in millimeters. As each of these values increase, the price of a diamond tends to increase as well.