Proj 2 - Diamonds

Val Wong - vmw170030

Diamonds Data Set

A dataset containing the prices and other attributes of almost 54,000 diamonds.

Sources: https://vincentarelbundock.github.io/Rdatasets/articles/data.html

Step 1 - Loading the data

```
Diamonds <- read.csv("diamonds.csv")
numOfRows <- NROW(Diamonds)
print(paste("Total Number of Rows in Diamonds: ", numOfRows, " rows."))
```

```
## [1] "Total Number of Rows in Diamonds:  53940  rows."
```

Step 2 - Data Cleaning

The Diamonds data set has been cleaned beforehand and contains no NA values. However let's determine if the columns we are interested in contain outliers.

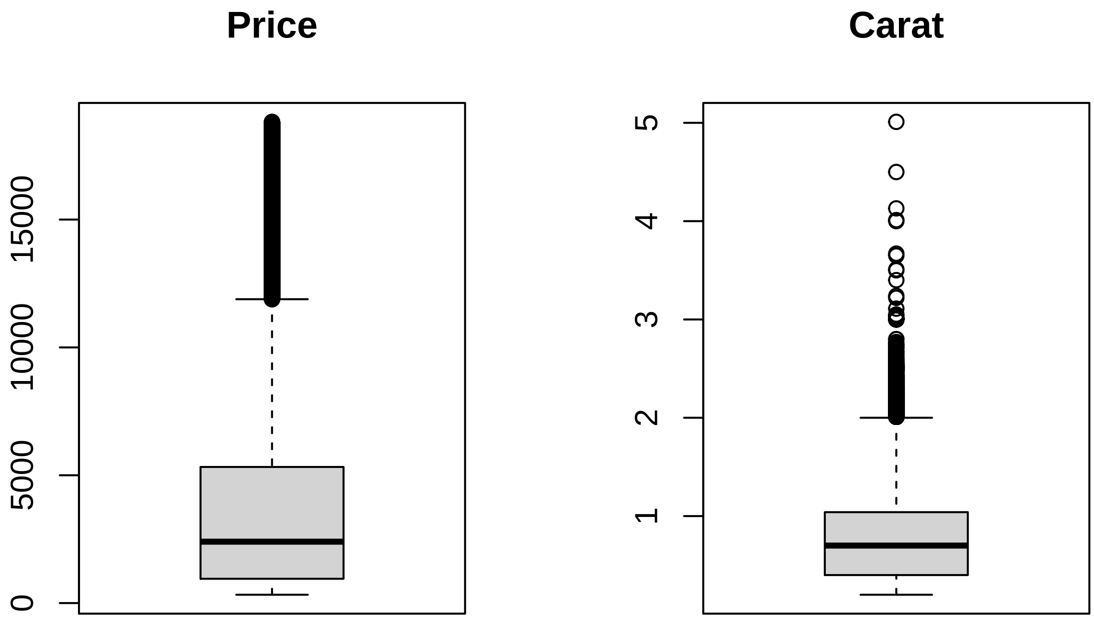
```
# Check NAs
print("Overall NAs:")
```

```
## [1] "Overall NAs:"
```

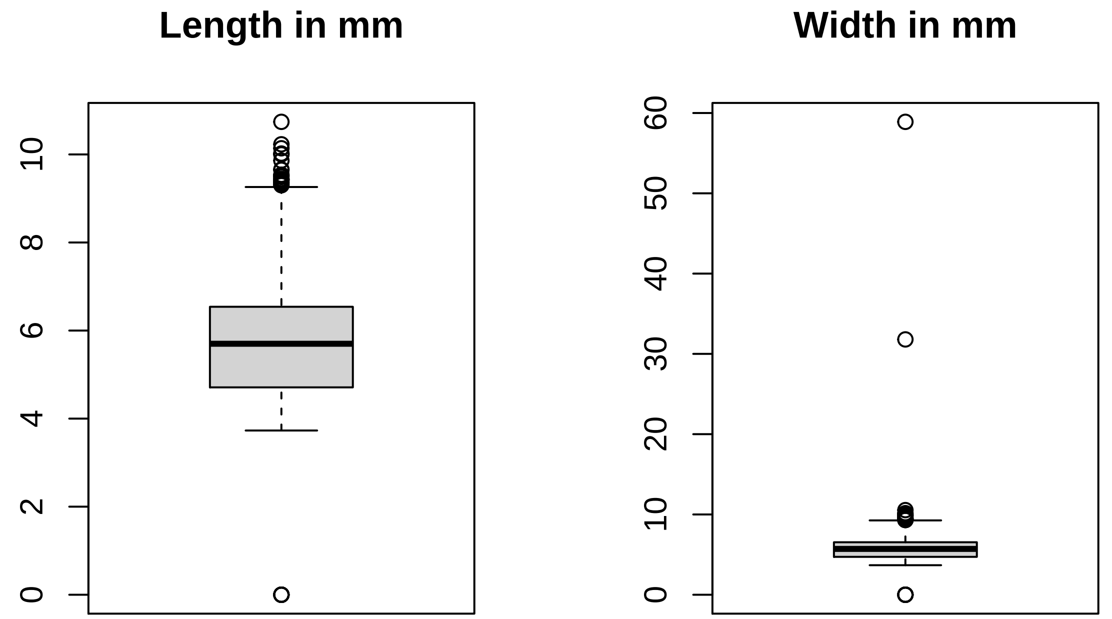
```
sum(is.na(Diamonds) == TRUE)
```

```
## [1] 0
```

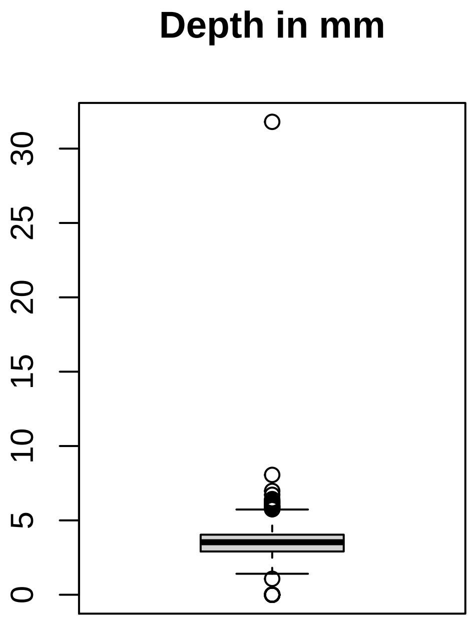
```
par(mfrow=c(1,2))
boxplot(Diamonds$price, main="Price")
boxplot(Diamonds$carat, main="Carat")
```



```
boxplot(Diamonds$x, main="Length in mm")
boxplot(Diamonds$y, main="Width in mm")
```



```
boxplot(Diamonds$z, main="Depth in mm")
```



It's important to keep in mind the clear amount of outliers in price and carat as it will be evident in the regression models.

Step 3 - Data Exploration

Display Column Names and Structure of NBA Shot logs data set.

```
names(Diamonds) # data exploration function 1
```

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

```
str(Diamonds) # data exploration function 2
```

```
## 'data.frame':    53940 obs. of  11 variables:
##  $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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" ...
##  $ color  : chr  "E" "E" "E" "I" ...
##  $ 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 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Preview First and Last Few Rows

```
head(Diamonds) # data exploration function 3
```

```
##   X carat       cut color clarity depth table price    x    y    z
## 1 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
tail(Diamonds) # data exploration function 4
```

```
##           X carat       cut color clarity depth table price    x    y    z
## 53935 53935  0.72   Premium     D     SI1  62.7    59  2757 5.69 5.73 3.58
## 53936 53936  0.72     Ideal     D     SI1  60.8    57  2757 5.75 5.76 3.50
## 53937 53937  0.72      Good     D     SI1  63.1    55  2757 5.69 5.75 3.61
## 53938 53938  0.70 Very Good     D     SI1  62.8    60  2757 5.66 5.68 3.56
## 53939 53939  0.86   Premium     H     SI2  61.0    58  2757 6.15 6.12 3.74
## 53940 53940  0.75     Ideal     D     SI2  62.2    55  2757 5.83 5.87 3.64
```

Summary of entire Diamonds data set.

```
summary(Diamonds) # data exploration function 5
```

```
##        X             carat            cut               color          
##  Min.   :    1   Min.   :0.2000   Length:53940       Length:53940      
##  1st Qu.:13486   1st Qu.:0.4000   Class :character   Class :character  
##  Median :26971   Median :0.7000   Mode  :character   Mode  :character  
##  Mean   :26971   Mean   :0.7979                                        
##  3rd Qu.:40455   3rd Qu.:1.0400                                        
##  Max.   :53940   Max.   :5.0100                                        
##    clarity              depth           table           price      
##  Length:53940       Min.   :43.00   Min.   :43.00   Min.   :  326  
##  Class :character   1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950  
##  Mode  :character   Median :61.80   Median :57.00   Median : 2401  
##                     Mean   :61.75   Mean   :57.46   Mean   : 3933  
##                     3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324  
##                     Max.   :79.00   Max.   :95.00   Max.   :18823  
##        x                y                z         
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
##  Median : 5.700   Median : 5.710   Median : 3.530  
##  Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
##  3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
##  Max.   :10.740   Max.   :58.900   Max.   :31.800  
```

Data Exploration - Miscellaneous

```
print(paste("Average Price of Diamonds: $", mean(Diamonds$price))) # data exploration function 6
```

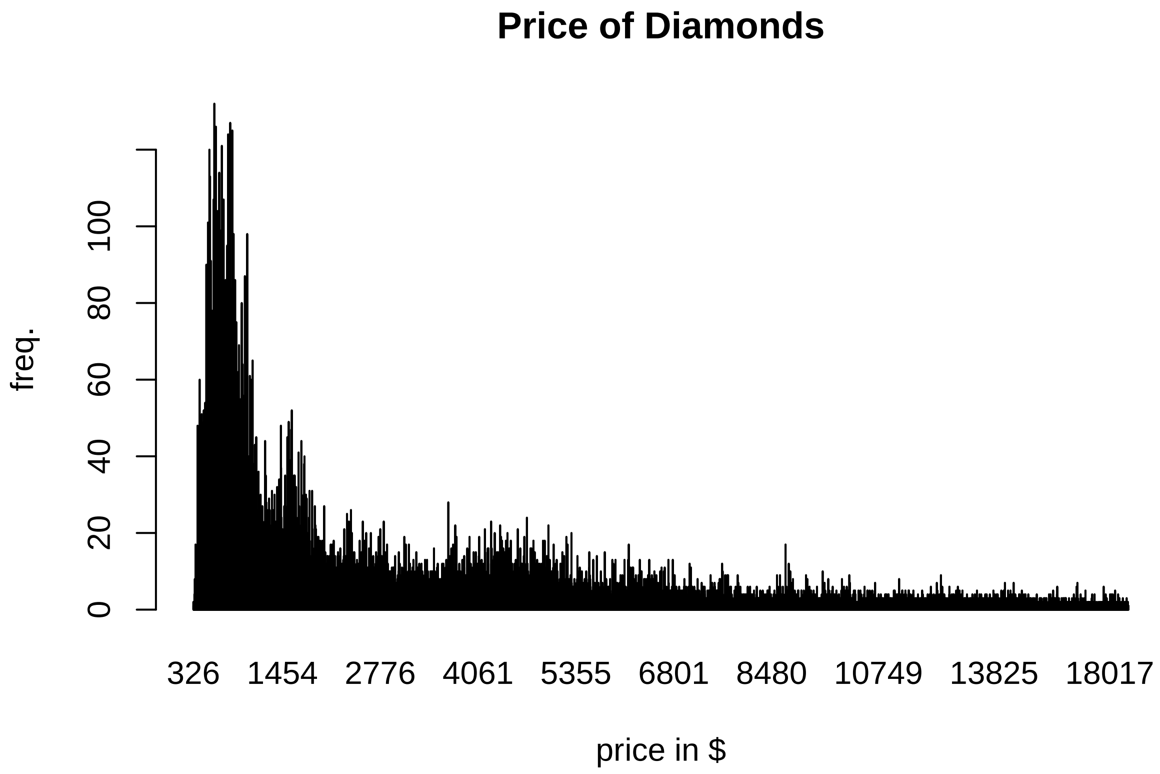
```
## [1] "Average Price of Diamonds: $ 3932.79972191324"
```

```
print(paste("Average Carat of Diamonds: ", mean(Diamonds$carat))) # data exploration function 7
```

```
## [1] "Average Carat of Diamonds:  0.797939747868001"
```

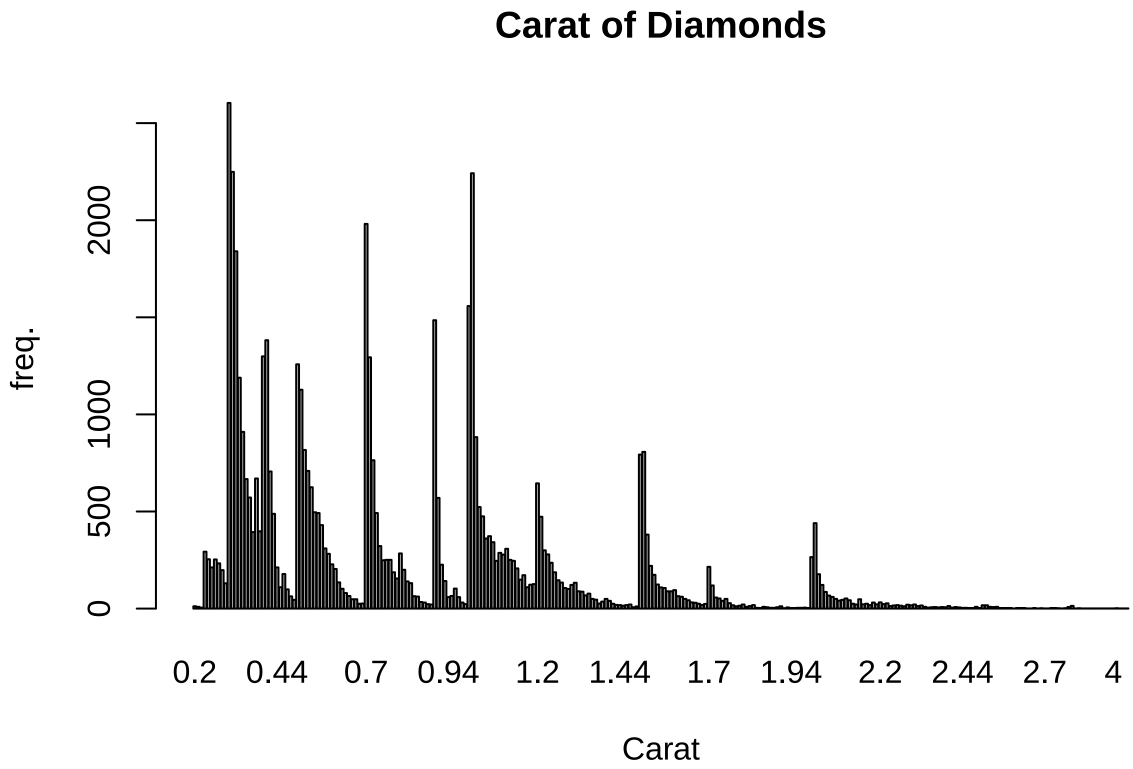
Data Visualization - Price Column Distribution

```
plot(factor(Diamonds$price),
     main= "Price of Diamonds",
     xlab= "price in $",
     ylab= "freq.") # data visual exploration function 1
```



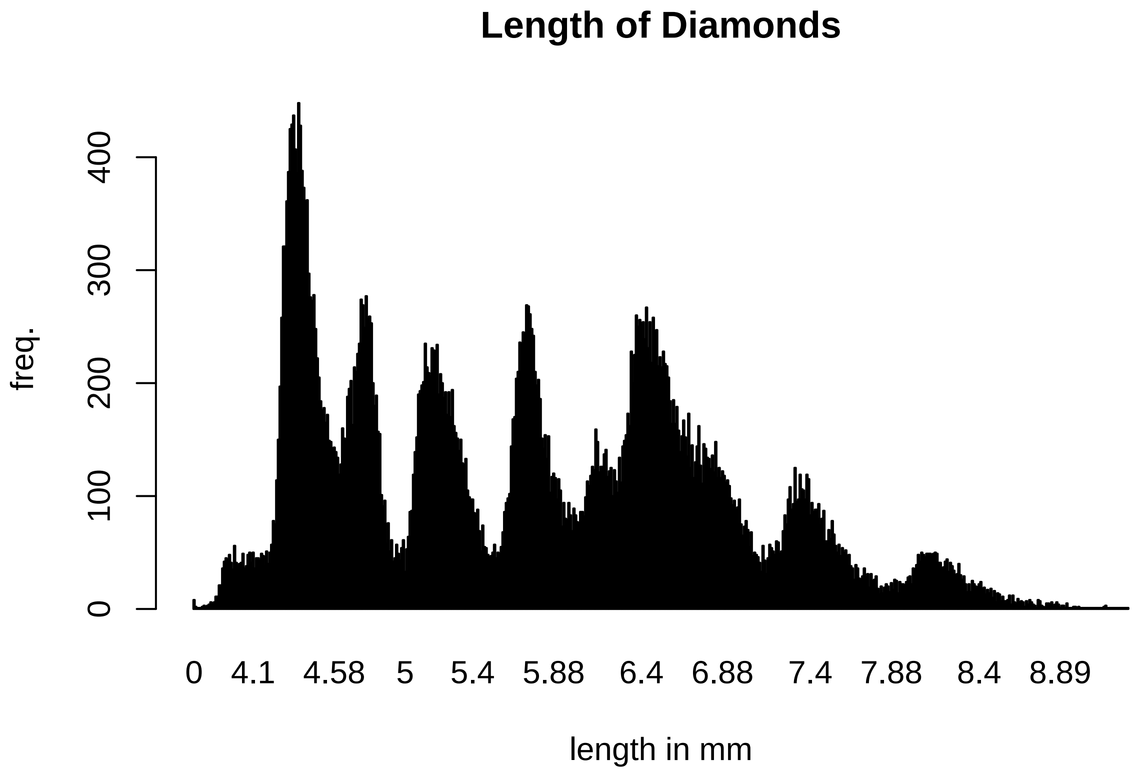
Data Visualization - Carat Column Distribution

```
plot(factor(Diamonds$carat),
     main= "Carat of Diamonds",
     xlab= "Carat",
     ylab= "freq.") # data visual exploration function 1
```



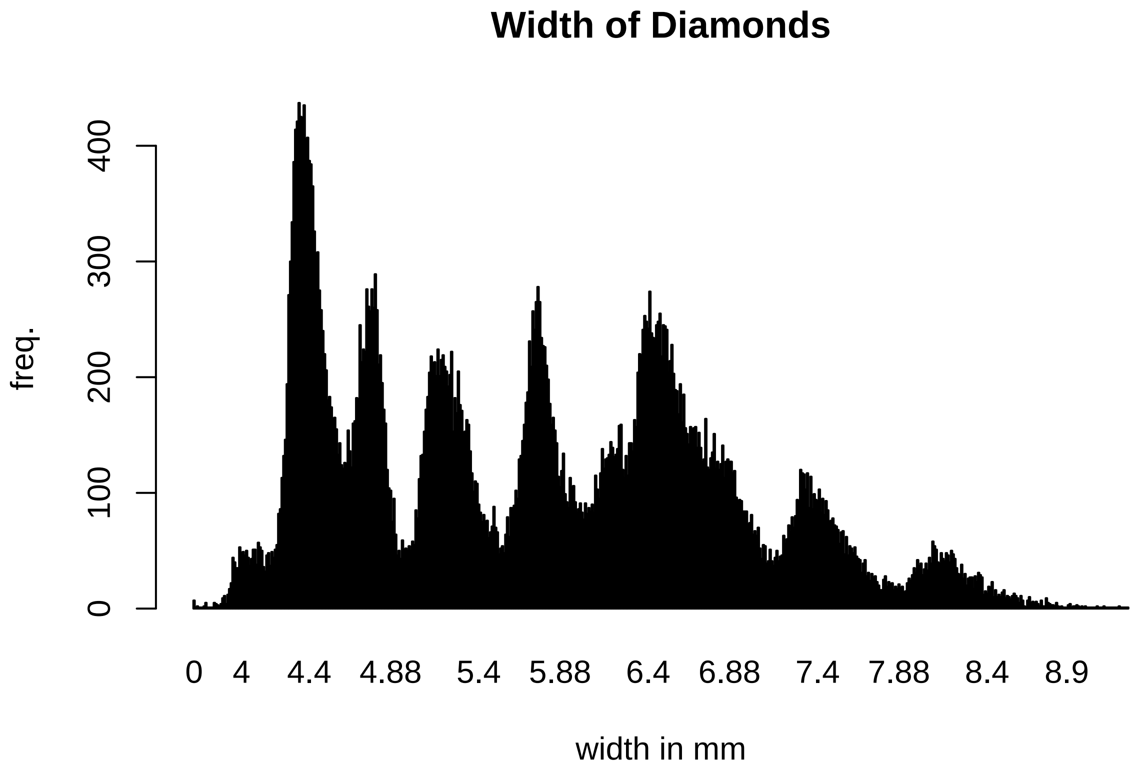
Data Visualization - Length Column Distribution

```
plot(factor(Diamonds$x),
     main= "Length of Diamonds",
     xlab= "length in mm",
     ylab= "freq.") # data visual exploration function 2
```



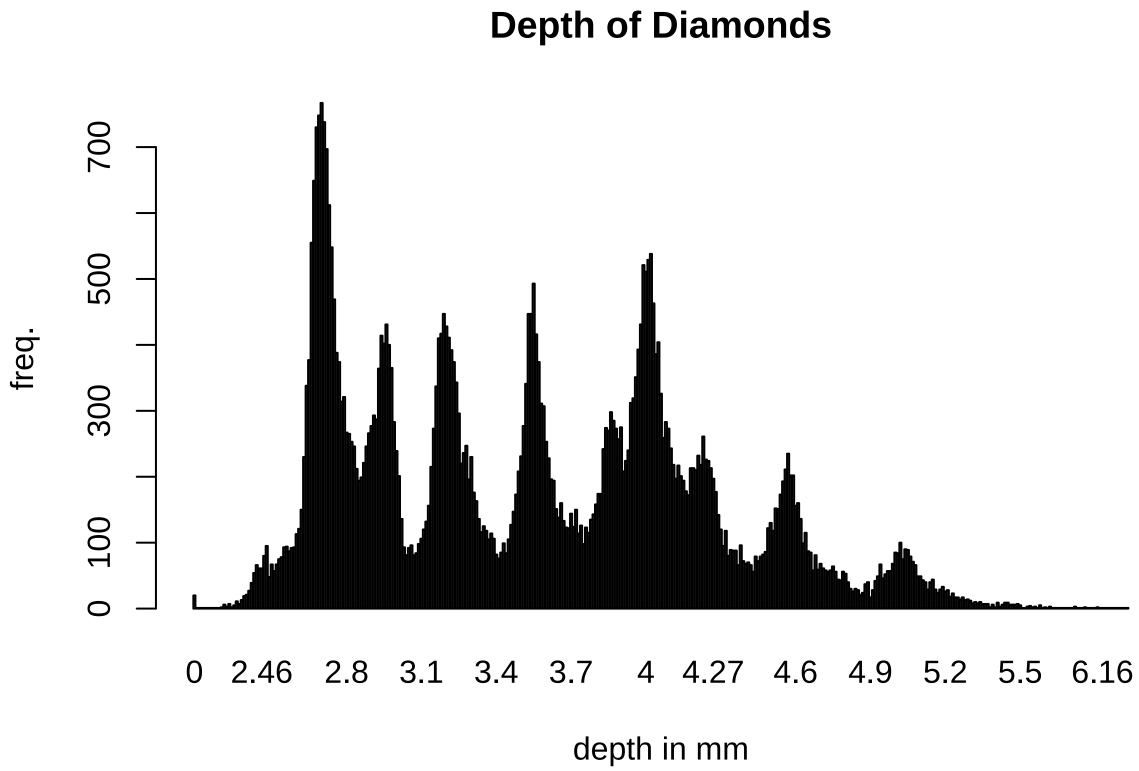
Data Visualization - Width Column Distribution

```
plot(factor(Diamonds$y),
     main= "Width of Diamonds",
     xlab= "width in mm",
     ylab= "freq.") # data visual exploration function 3
```



Data Visualization - Depth Column Distribution

```
plot(factor(Diamonds$z),
     main= "Depth of Diamonds",
     xlab= "depth in mm",
     ylab= "freq.") # data visual exploration function 4
```



Step 4.0 - Dividing into train and test sets.

Divide data randomly. 75% into train and 25% into test.

```
set.seed(1234)
i <- sample(1:nrow(Diamonds), nrow(Diamonds)*0.75, replace=FALSE)
train <- Diamonds[i,] # 75% in train
test <- Diamonds[-i,] # 25% in test
```

Step 4.1 - Linear Regression

I selected the linear regression model for the relative simplicity of the algorithm and to act as a good baseline comparison to other regression algorithms. Furthermore, with diamonds and pricing relationship, the presumption that bigger diamonds would yield higher price can strengthen a case for data exbiting a linear pattern.

```
set.seed(1234)
lm <- lm(price ~ carat + x + y + z, data=train)
summary(lm)
```

```
##
## Call:
## lm(formula = price ~ carat + x + y + z, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17273.7   -635.6    -16.4    348.4  13735.3
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2228.49     121.30   18.37  < 2e-16 ***
## carat        10448.76      73.23  142.69  < 2e-16 ***
## x             -988.00      43.94  -22.48  < 2e-16 ***
## y              119.54      25.93    4.61 4.03e-06 ***
## z             -467.91      41.84  -11.18  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1518 on 40450 degrees of freedom
## Multiple R-squared:  0.8549, Adjusted R-squared:  0.8549
## F-statistic: 5.958e+04 on 4 and 40450 DF,  p-value: < 2.2e-16
```

Evaluation on Test

```
set.seed(1234)
predlm <- predict(lm, newdata=test)
corlm <- cor(predlm, test$price)
mselm <- mean((predlm - test$price)^2)
rmselm <- sqrt(mean((predlm - test$price)^2))

print(paste("Linear Regression COR: ", corlm))
```

```
## [1] "Linear Regression COR:  0.922758755303959"
```

```
print(paste("Linear Regression MSE: ", mselm))
```

```
## [1] "Linear Regression MSE:  2380014.33574145"
```

```
print(paste("Linear Regression RMSE: ", rmselm))
```

```
## [1] "Linear Regression RMSE:  1542.72950828765"
```

Linear Regression Model resulted in an correlation value of 0.922758755303959, MSE of 2380014.33574145, and RMSE of 1542.72950828765. The correlation value being so close to 1 illustrates that the model and evaluation on the test data are pretty similar in relationship (strong positive). The MSE and RMSE indicate high residual values which can be seen in huge jumps of price values in the data set.

Step 4.2 - kNN Regression

I selected kNN regression for its advantage in assuming no shape concerning the data. However, its important to acknowledge the sensitivity of the algorithm to outliers as it chooses neighbors based on distance.

```
library(caret)
```

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

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

```
# carat = 2, price = 8, x = 9, y = 10, z = 11
knnreg <- knnreg(train[c(2,9,10,11)], train[,8], k=1)

predknn <- predict(knnreg, test[c(2,9,10,11)])
corknn <- cor(predknn, test$price)
mseknn <- mean((predknn - test$price)^2)
rmseknn <- sqrt(mean((predknn - test$price)^2))
```

```
print(paste("kNN Regression COR: ", corknn))
```

```
## [1] "kNN Regression COR:  0.897254333737754"
```

```
print(paste("kNN Regression MSE: ", mseknn))
```

```
## [1] "kNN Regression MSE:  3275547.92933657"
```

```
print(paste("kNN Regression RMSE: ", rmseknn))
```

```
## [1] "kNN Regression RMSE:  1809.84748786647"
```

kNN Regression Model resulted in an correlation value of 0.897254333737754, MSE of 3275547.92933657, and RMSE of 1809.84748786647. The resulting correlation value is lower than linear regression model but is still close to 1 illustrating that the model and evaluation on the test data are pretty similar in relationship (strong positive). The outliers are also more impactful in this algorithm as predicted illustrated in the higher values of MSE and RMSE.

Step 4.3 - Decision Tree (Regression)

I selected the Decision Tree algorithm for the advantage of being easy to interpret and performing without the requirement of normalization of the data.

```
library(tree)
```

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

```
tree <- tree(price ~ carat + depth, data=train)
summary(tree)
```

```
##
## Regression tree:
## tree(formula = price ~ carat + depth, data = train)
## Variables actually used in tree construction:
## [1] "carat"
## Number of terminal nodes:  5
## Residual mean deviance:  2364000 = 9.561e+10 / 40450
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -9752.0  -567.5  -181.0     0.0   544.3 12580.0
```

Evaluate

```
predtree <- predict(tree, newdata=test)
cortree <- cor(predtree, test$price)
msetree <- mean((predtree - test$price)^2)
rmsetree <- sqrt(mean((predtree - test$price)^2))

print(paste("Tree Regression COR: ", cortree))
```

```
## [1] "Tree Regression COR:  0.922150171471487"
```

```
print(paste("Tree Regression MSE: ", msetree))
```

```
## [1] "Tree Regression MSE:  2395913.01360515"
```

```
print(paste("Tree Regression RMSE: ", rmsetree))
```

```
## [1] "Tree Regression RMSE:  1547.873707253"
```

Tree Pruning

```r
tree_pruned <- prune.tree(tree, best=5)

predtreepruned <- predict(tree_pruned, newdata=test)
cortreepruned <- cor(predtreepruned, test$price)
msetreepruned <- mean((predtreepruned - test$price)^2)
rmsetreepruned <- sqrt(mean((predtreepruned - test$price)^2))

print(paste("Tree Regression COR: ", cortreepruned))
```

```
## [1] "Tree Regression COR:  0.922150171471487"
```

```r
print(paste("Tree Regression MSE: ", msetreepruned))
```

```
## [1] "Tree Regression MSE:  2395913.01360515"
```

```r
print(paste("Tree Regression RMSE: ", rmsetreepruned))
```

```
## [1] "Tree Regression RMSE:  1547.873707253"
```

The normal tree and pruned tree outputted the same results which is acceptable as pruning the tree has no guarantee of improving performance. Decision Tree Model resulted in an correlation value of 0.922150171471487, MSE of 2395913.01360515, and RMSE of 1547.873707253. The correlation value achieved by Decision tree was almost exact as linear regression which could likely show that the data set isn't particularly linear in shape otherwise would give linear model a bigger advantage. Additionally, we continue to see the effects of outliers within the model's high MSE and RMSE values.

Step 5 - Results Analysis

Metrics Comparison: **Correlation:** - Linear Regression: 0.922758755303959 - kNN Regression: 0.897254333737754 - Decision Tree Regression: 0.922150171471487

MSE: - Linear Regression: 2400312.93680125 - kNN Regression: 3275547.92933657 - Decision Tree Regression: 2395913.01360515

RMSE: - Linear Regression: 1549.29433510913 - kNN Regression: 1809.84748786647 - Decision Tree Regression: 1547.873707253

Overall, the Linear Regression model and Decision tree model were neck and neck but Linear Regression did just a tad bit better in terms of correlation. kNN Regression definitely performed poorest due to the sensitivity to outliers which this data set was a victim to. Also restating, the correlation value achieved was almost exact as linear regression which could likely show that the data set isn't particularly linear in shape otherwise would give linear model a bigger advantage.

The script was able to identify notable weaknesses and strengths of the algorithms but in terms of what it was able to learn from the data, would be considered not useful. The outliers were a major factor of variance and residual impact on the models, exemplified even more for kNN Regression.