Diamond Price Prediction using R

BUS 235A Introduction to Business Analytics
Spring 2022

Submitted to,
Prof. Bilgehan Yildiz

Submitted by:
Akshay Kumar Govindappa (014634633)
Kaustubh Dalal (015278003)
Payal Patel (015354040)
Sindhu Hegde (015307331)

Introduction

In this project, we are trying to implement the fundamental Data Visualization techniques to an interesting data set called diamonds dataset from Kaggle repository. The dataset contains the prices and other attributes of almost 54,000 diamonds. The goal of this project is to predict the price of the diamond based on the other attributes.

1. About the Dataset

Dataset column details:

Column Name	***Class***	***Description***
X	Numeric	# Index counter
carat	Numeric	Weight of the diamond
cut	Character	Describes the cut quality of the diamond. Quality in increasing order Fair, Good, Very Good, Premium, Ideal
color	Character	Color of the diamond, with D being the best and J the worst
clarity	Character	How obvious inclusions are within the diamond:(in order from best to worst, FL = flawless, I3= level 3 inclusions) FL, IF, VVS1,
depth	Numeric	Depth percentage: The height of a diamond, measured from the culet to the table, divided by its average girdle diameter
table	Numeric	Width of top of the diamond relative to widest point (43--95)
price	Integer	The price of the diamond (\$326--\$18,823)
x	Numeric	Length in mm (0--10.74)
y	Numeric	Width in mm (0--58.9)
z	Numeric	Depth in mm (0--31.8)

Table 1. Column description of the dataset

1.1 Dataset Description

A diamond's four basic characteristics are clarity, cut, carat, and color. Each feature has an impact on the cost and price of a diamond, so knowing what they are and how they affect the price of a diamond is critical. This data contains a total of 53,940 round-cut diamonds. There are 11 columns in all. The first column, X, is made up entirely of index numbers. As a result, 10 variables measure various aspects of diamonds. The names of these variables are in lowercase.

1.2 Categorical Columns

Cut, color, and clarity are three variables that have an ordered factor structure. The categorical values are arranged in low-to-high rank order by an ordered factor. For example,

1. cut: Fair, Good, Very Good, Premium, and Ideal
2. color: from J (worst) to D (best)
3. clarity: I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)

The barplot of prominent categorical columns(characteristics) of the dataset is shown below.

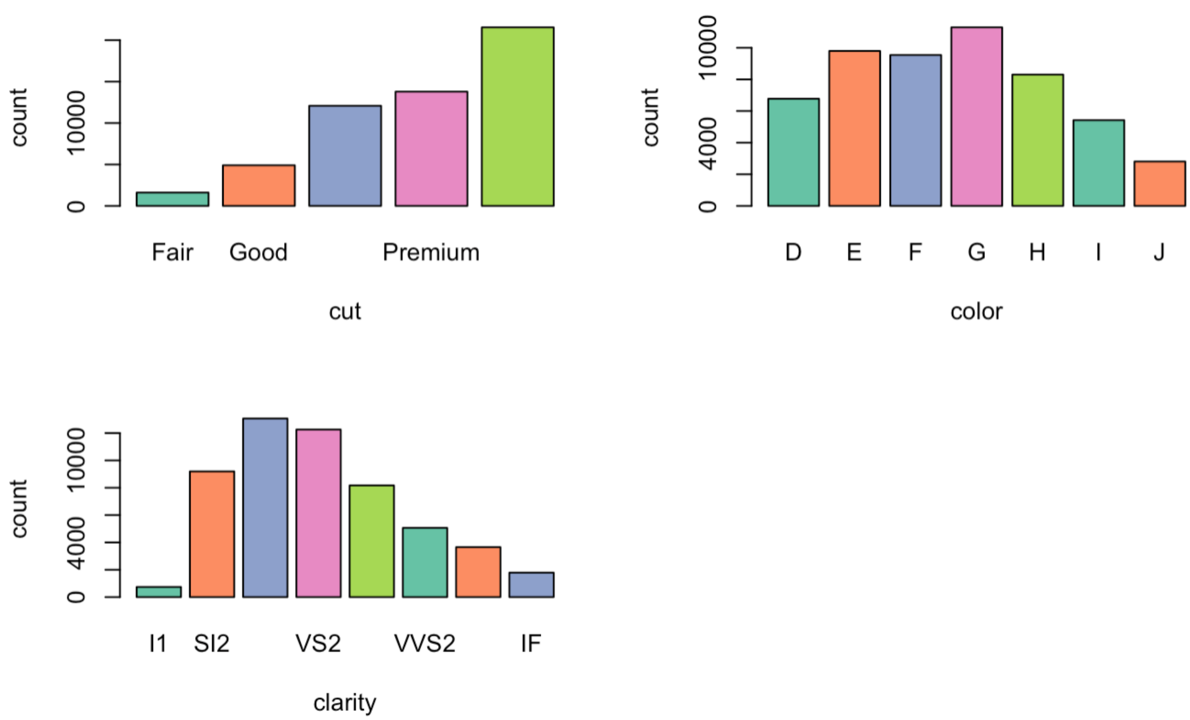

Figure1: Bar plot of categorical columns

The visualization images above illustrate that the following characteristics are popular in the data set with the best and worst ranges.

- cut - Ideal and Premium
- color - G & E
- clarity - S12-VS2

1.3 Numeric Variables

There are six numeric-structured variables: carat, depth, table, x, y, and z. Scatter plots for all predictor numeric columns are given below:

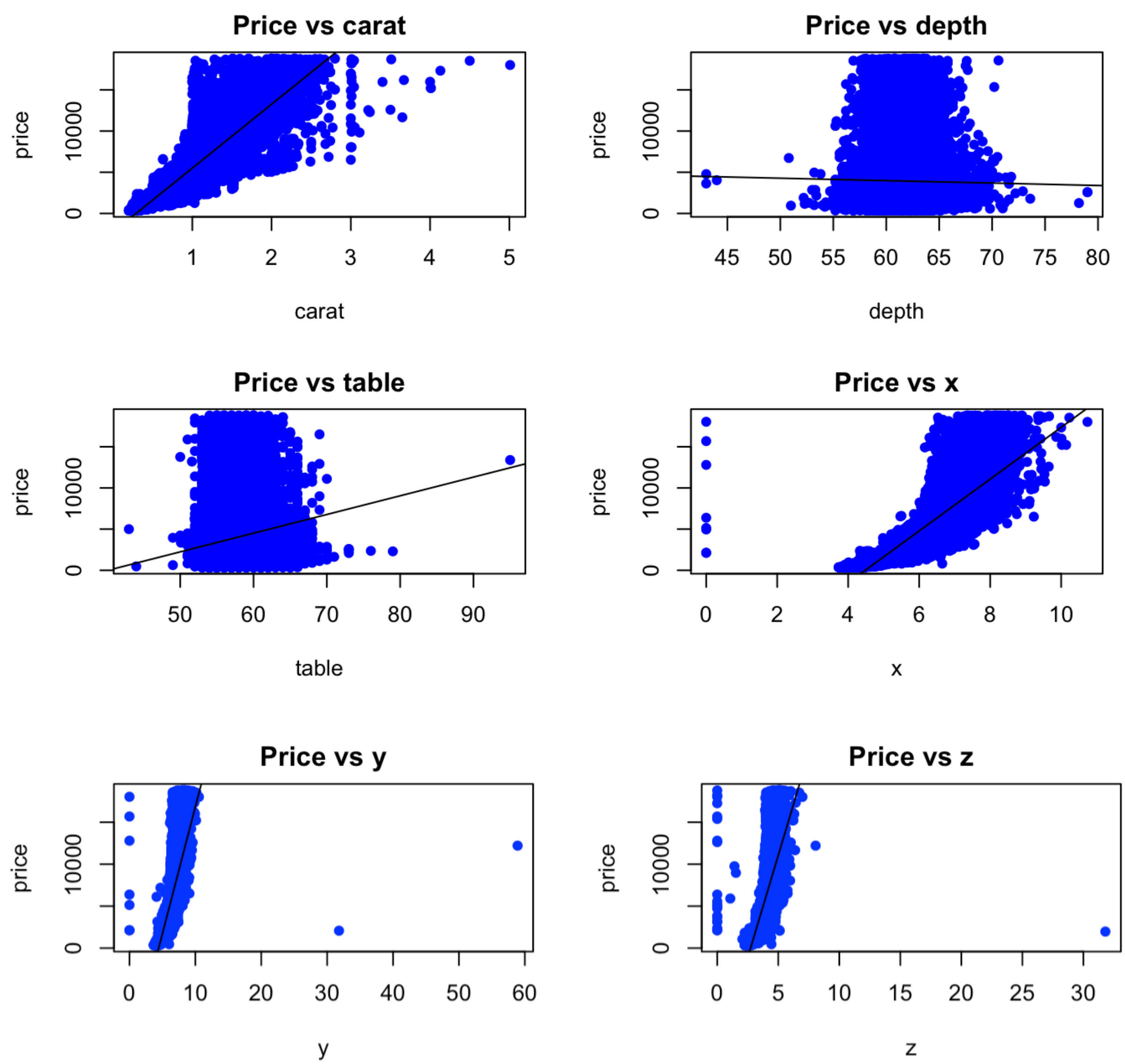

Figure 2: Scatter plot of numerical columns

Numeric columns can be classified into the following groups using the scatter plots shown above:

- Positive relation with a price: carat, table, x,y,z
- Negative relation with a price: depth

1.4 Distribution of the Response variable: Price

There is only one integer-structured variable that is price.

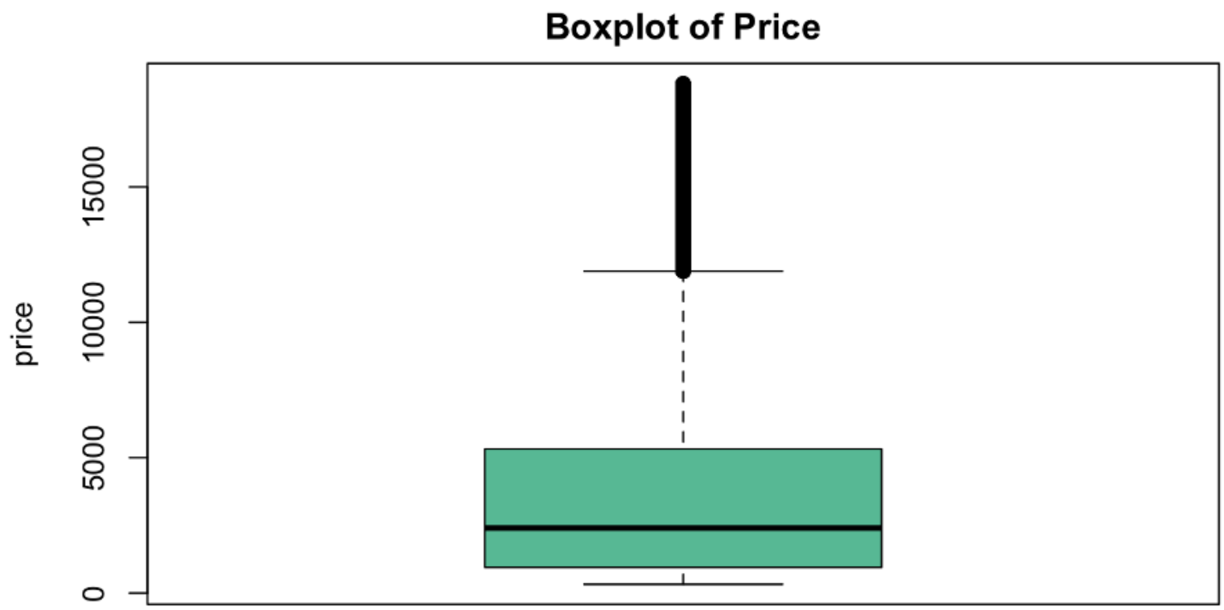

Figure 3: Boxplot of price

Figure 3 shows that the longer section of the box is to the right (or above) of the median, indicating that the data is skewed right. It has a large concentration of observations below the $5,000 level in the United States. Demand indicates that fewer consumers are prepared to pay more for higher-quality diamonds.

2. Applying Data Visualizations Techniques on the Dataset

The following plots give some insights into the data set.

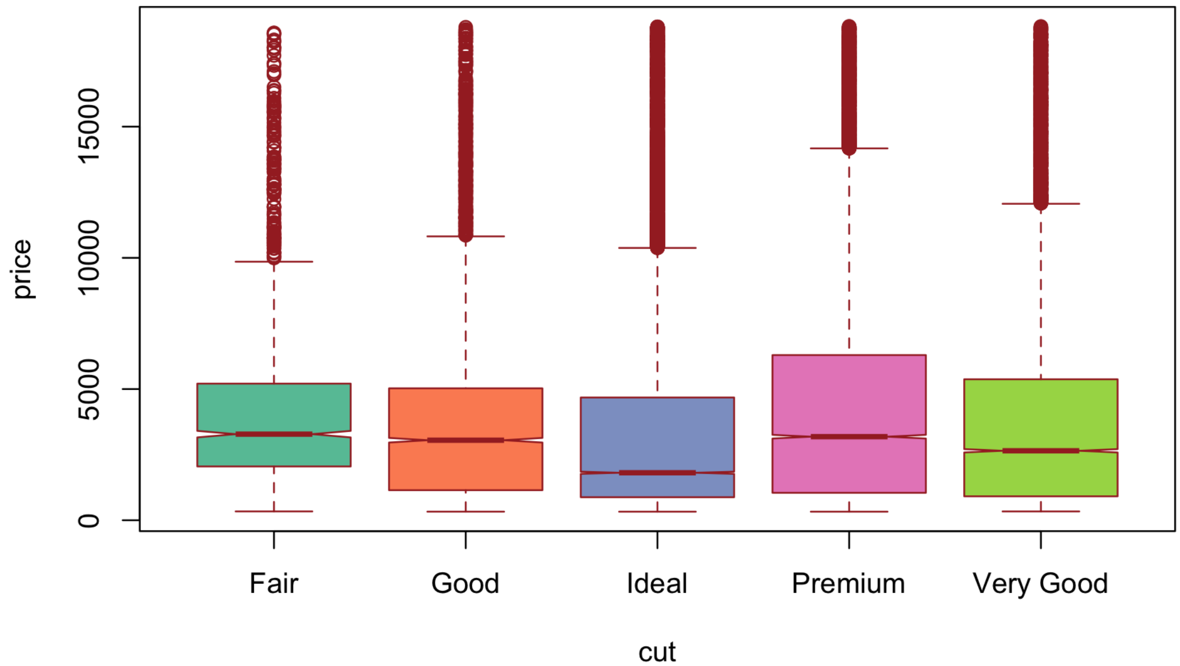

Figure 4: Boxplot of price vs. cut

The cut of a diamond can help assess its quality and whether or not it will be costly. However, as seen in Figure 4, the price does not appear to be affected significantly by the cut.

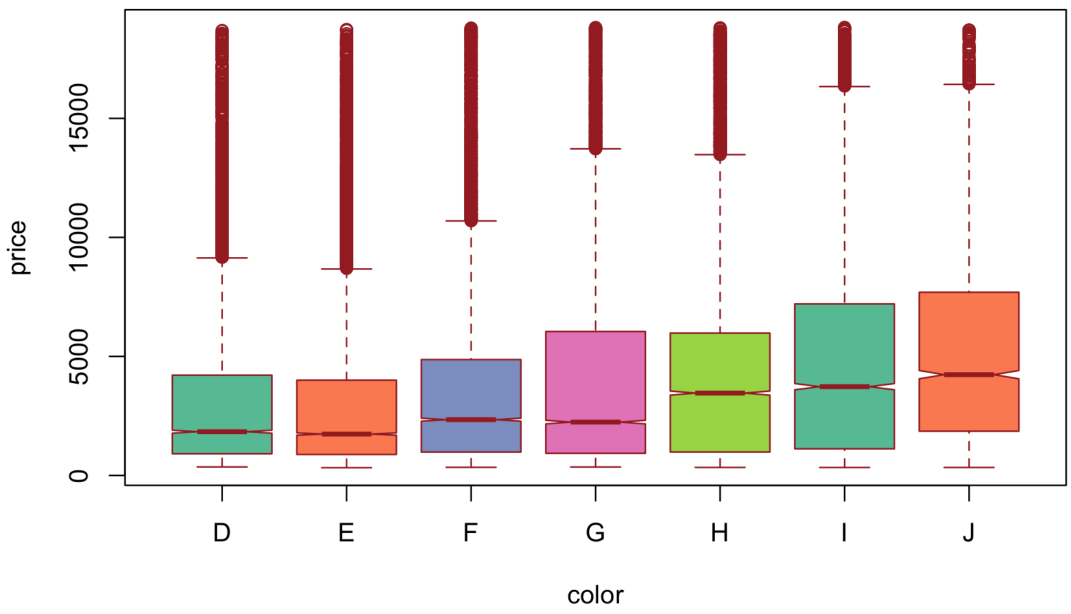

Figure 5: Boxplot of price vs. color

From figure 5, it is evident that Color appears to have an impact on the quality of a diamond and whether or not it will be pricey. Color is an important factor to consider.

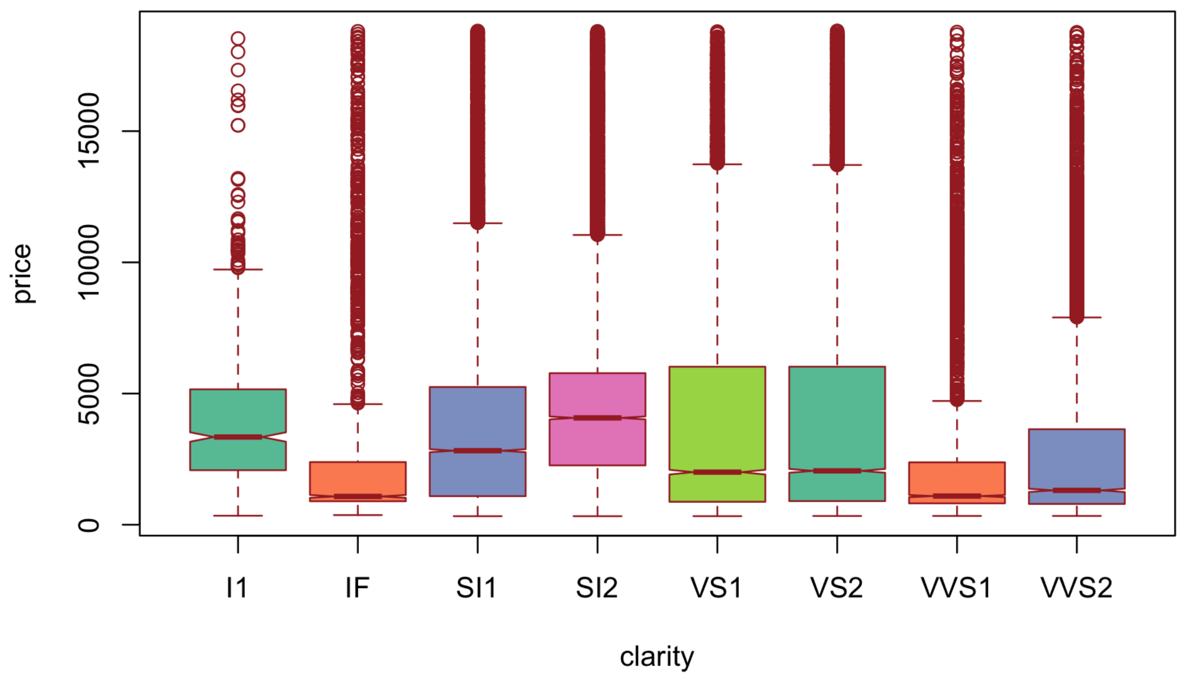

Figure 6: Boxplot of price vs. clarity

Figure 6 shows that, in comparison to cut, clarity appears to be a significant variable.

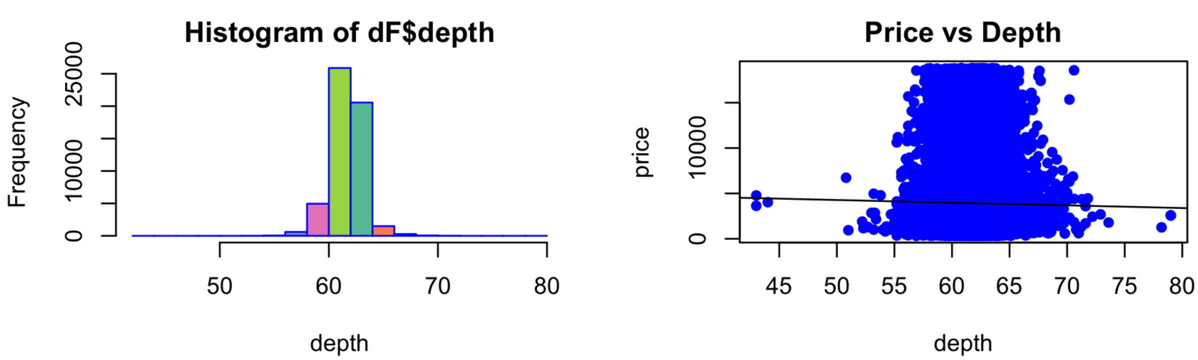

Figure 7: Histogram of depth & Scatter plot of price vs. depth

Figure 7 shows that the price for the same depth can vary greatly.

The variables height, width, and length are multiplied to get a new variable, volume. This arrangement is made to see if there is any difference in the comparison of price to these three parameters. The scatter plot of price vs. volume(x*y*z) is given below.

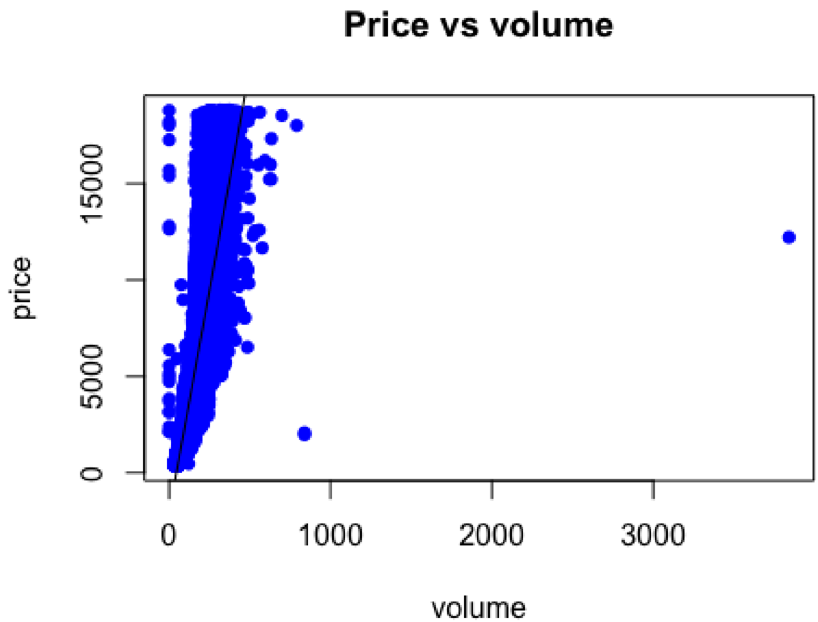

Figure 8: Scatter plot of price vs. volume

Correlation of numeric data

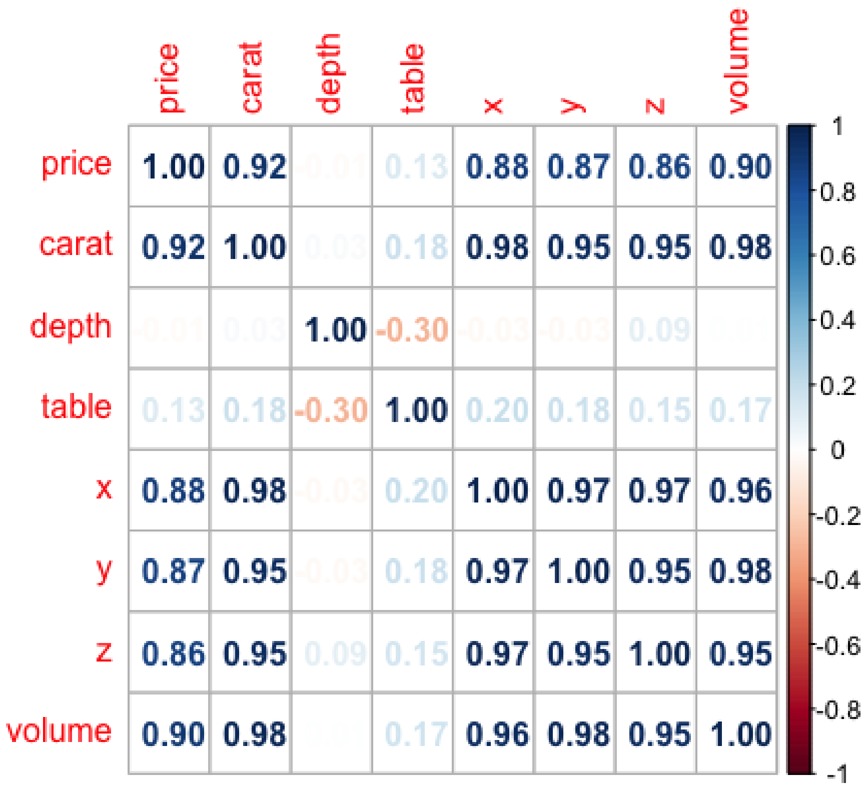

Figure 9: Correlation matrix of numeric columns

The correlation matrix shows that the price has the highest correlation with carat with 0.92 and least correlation with depth.

3. Applying Linear Regression on the Dataset

3.1 Data Preparation

Checking null values: There are no null values in the dataset.

3.2 Factor Conversion

The three categorical columns cut, clarity, and color are converted into factor columns.

3.3 Partitioning data into training and test datasets

Linear Regression is applied to the dataset to predict the price of the diamond by varying various variables. The data is divided into two sets: training and testing: the test partition is used to evaluate the final model's performance with unknown data. Test data will not be used in training to ensure that the model is evaluated objectively. After partitioning the data,

Total dataset size: 53940
Training set size: 85% of the dataset - 50344
Testing set size: 15% of the dataset - 3596

The result of various linear regression implementations is summarized in the table below.

Model Description	***R - squared***	***RMSE Training***	***RMSE Test***
Model with all the variables	0.9204	1128.14	1121.65
Model without depth & table variable	0.9203	1128.81	1127.28
Model without x, y, z variables	0.9161	1157.65	1129.73
Model without volume variable	0.9198	1131.92	1100.70
Model with just the carat value	0.8496	1550.53	1520.45
Model without table variable	0.9203	1128.68	1123.67
Model with all variables and 5-fold cross validation	0.9204	1128.14	1121.65
Regression with L1 constraint on the parameters	0.8496	1128.81	1127.28

Table 2. Summary of Linear Regression of the dataset

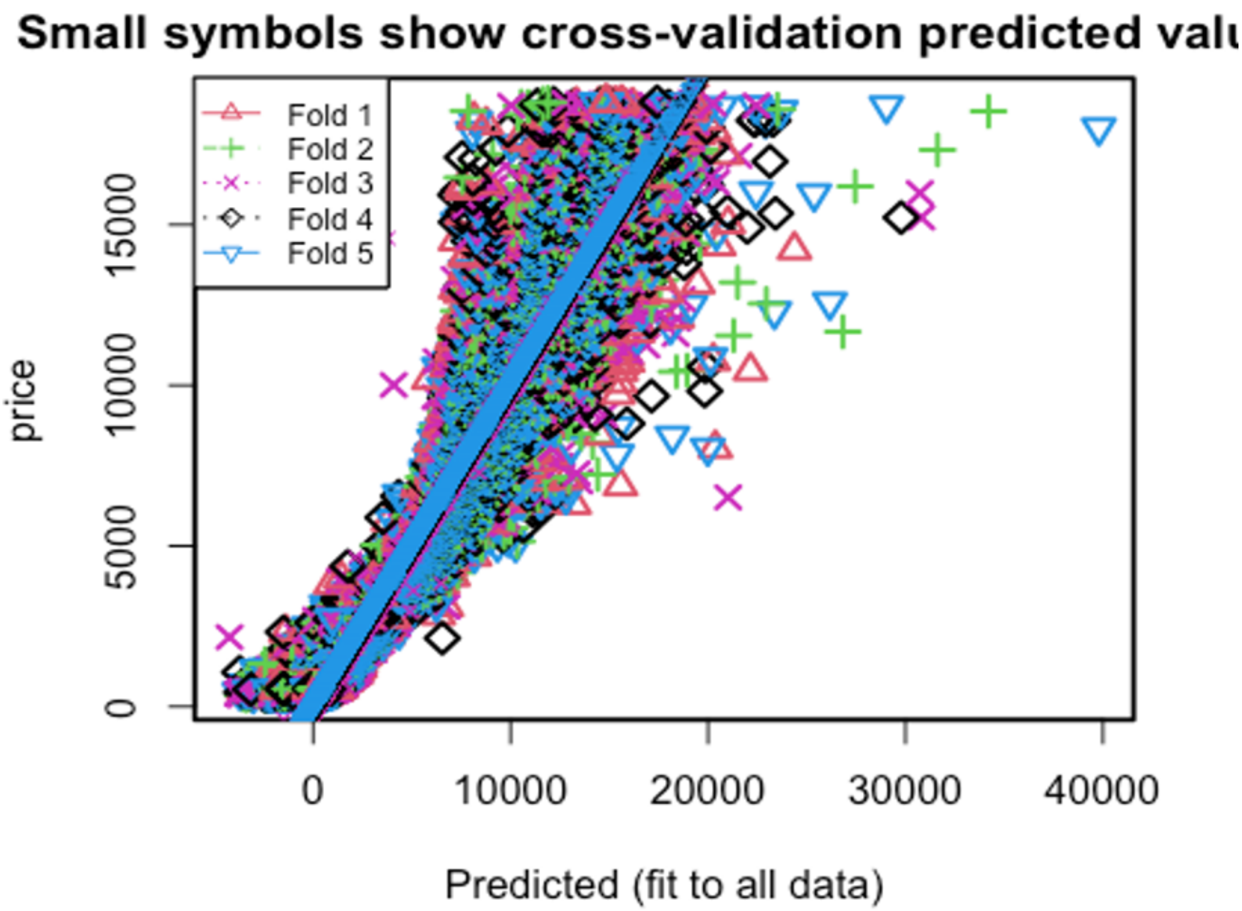

Figure 10. Price vs all variables and 5 fold cross-validation

We can see that, the cross validation lines of all the foldes are parallel to each other and the color distribution is spread equally across the lines. Hence all the 5 folds of cross validation has given very similar results.

4. Analysis of data

Some analysis based on the application of linear regression model on the data:

- Carat values are highly correlated to the price value. Both the correlation plot and the linear regression experiments second this observation.
- Categorical variables – cut, clarity, and color – play an important role in price determination.
- Our dataset has few outliers. For example, in the clarity or color box plots, we see a lot of values beyond 1.5 * IQR (interquartile range)
- From price vs carat or price vs dimensions plots, we see an obvious trend of increasing price value as the other variables increase.

We observe that in both the train and test graph, data is equally distributed along the x=y line with some outliers.

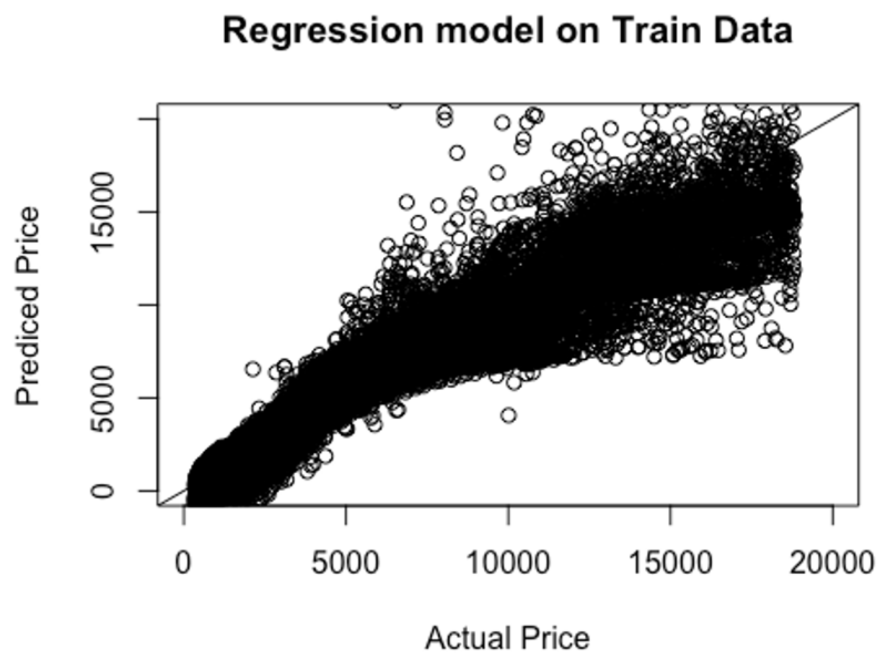

Figure 11. Regression model on train and test data

We observe that in both the train and test graph, data is equally distributed along the x=y line with some outliers.

5. Conclusion

This project gave us a good understanding of data visualization and linear regression concepts. Diamond price determination with features like color, carat, clarity, cut, depth, table, and dimensions are challenging.

This project was done in two parts. Initially, we tried to explore the data based on the types. Clarity, color, and cut were categorical values and other features were numerical. We converted the categorical values into factors to help us with the regression. One interesting find was that, even though we found a good trend for a single feature vs price plot, we did find a lot of outliers. This explained that a single feature was not sufficient to explain the data well.

In the second part of the project, we tried several regression techniques for the dataset. From the correlation data, we see that price was highly correlated to the carat value and least correlated to depth value. Similarly, we did see no significant RMSE difference when we dropped depth and table values in the regression. We also experimented with n-fold cross-validation and lasso regression techniques. All the regressions were compared based on the p-value, r-squared value, and RMSE on the train and test dataset. Out of all the experiments, including all the variables against price showed the best performance on both train and test sets.

As a future scope, we could experiment with advanced machine learning techniques – like neural networks and random forest regressions – to improve the performance. We believe there is a scope for feature engineering – like nonlinear mapping of features and feature normalizations – to help us predict the price better.

6. References

- *https://www.kaggle.com/shivam2503/diamonds*
- *https://medium.com/swlh/simple-guide-to-data-visualization-6ef6fa726e38*
- *https://www.scribbr.com/statistics/simple-linear-regression/*
- *https://www.scribbr.com/statistics/linear-regression-in-r/*

Appendix

```
library(janitor)

##

## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':

##

##      chisq.test, fisher.test

library(dplyr)

##

## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##

##      filter, lag

## The following objects are masked from 'package:base':

##

##      intersect, setdiff, setequal, union

library(corrplot)

## corrplot 0.90 loaded

library(RColorBrewer)

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(ggplot2)

coul <- brewer.pal(5, "Set2")
```

Loading Data

```
data1 <- read.csv("diamonds.csv", header = TRUE)
```

Data Cleaning

Remove empty rows and columns of Data

```
data2 <- remove_empty(data1, which = c("rows","cols"), quiet = FALSE)

## No empty rows to remove.

## No empty columns to remove.
```

The argument quiet = FAlSE used for to display message whether there is empty rows/columns in the data1 or not.

Remove Duplicate Rows of Data using distinct() function available in dplyr R package

```
diamonds_df <- distinct(data2)
```

There is no duplicate data to be removed.

Summarizing Data

```
summary(diamonds_df)

##       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 :26970   Median :0.7000   Mode  :character   Mode  :character
## Mean   :26970   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
```

Number of rows and columns

```
cat(sprintf("Number of rows: %d, Number of columns: %d", nrow(diamonds_df),
ncol(diamonds_df)))
## Number of rows: 53940, Number of columns: 11
```

Droping column X as we already have the index

```
dF <- subset(diamonds_df, select = -c(X) )
sapply(dF,class)
##       carat         cut       color     clarity       depth       table
##   "numeric" "character" "character" "character"   "numeric"   "numeric"
##       price           x           y           z
##   "integer"   "numeric"   "numeric"   "numeric"
```

Categorical Columns Plots

```
# 1. cut
par(mfrow = c(2,2))
count1 <- table(dF$cut)
barplot(count1, ylab = "count", xlab = "cut", col = coul)


# 2. color
```

```
count2 <- table(dF$color)
barplot(count2, ylab = "count", xlab = "color", col = coul)


# 3. clarity
count3 <- table(dF$clarity)
barplot(count3, ylab = "count", xlab = "clarity", col = coul)
```

Numeric Columns

Response Variable : Price

```
summary(dF)
##      carat            cut               color              clarity
## Min.   :0.2000   Length:53940      Length:53940          Length:53940
## 1st Qu.:0.4000   Class :character   Class :character   Class :character
## Median :0.7000   Mode  :character   Mode  :character   Mode  :character
## Mean   :0.7979
## 3rd Qu.:1.0400
## Max.   :5.0100
##      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
boxplot(dF$price, col = coul, ylab = "price", main = "Boxplot of Price")
```

scatter plot

```
par(mfrow = c(2,2))

# 1. carat
plot(price ~ carat, data = dF, main = "Price vs carat", col ="blue", pch = 16)
abline(lm(price ~ carat, data = dF))

# 2. depth
plot(price ~ depth, data = dF, main = "Price vs depth", col ="blue", pch = 16)
abline(lm(price ~ depth, data = dF))

# 3. table
plot(price ~ table, data = dF, main = "Price vs table", col ="blue", pch = 16)
abline(lm(price ~ table, data = dF))

# 4. x
plot(price ~ x, data = dF, main = "Price vs x", col ="blue", pch = 16)
abline(lm(price ~ x, data = dF))
# 5. y
plot(price ~ y, data = dF, main = "Price vs y", col ="blue", pch = 16)
```

```
abline(lm(price ~ y, data = dF))


# 6. z
plot(price ~ z, data = dF, main = "Price vs z", col ="blue", pch = 16)
abline(lm(price ~ z, data = dF))
```

Visualizing the Data with boxplot

```
# 1. price vs. cut
boxplot(dF$price ~ dF$cut,
        col = coul,
        border = "brown",
        notch = TRUE,
        ylab = "price",
        xlab = "cut")
# 2. price vs. color
boxplot(dF$price ~ dF$color,
        col = coul,
        border = "brown",
        notch = TRUE,
        ylab = "price",
        xlab = "color")
# 3. price vs. clarity
boxplot(dF$price ~ dF$clarity,
        col = coul,
        border = "brown",
        notch = TRUE,
        ylab = "price",
```

```
        xlab = "clarity")
```

Depth and Price

```
# Histogram of Depth

par(mfrow = c(2,2))

hist(dF$depth,xlab = "depth",col = coul,border = "blue")


#  Depth vs Price

plot(price ~ depth, data = dF, main = "Price vs Depth", col ="blue", pch = 16)

abline(lm(price ~ depth, data = dF))


# We can infer from the plot that the Price can vary heavily for the same Depth.
```

Dimnesion - x, y, z

```
plot(density(dF$x))

lines(density(dF$y))

lines(density(dF$z))
```

Consutruct volume from width, height and length

```
dF$volume <- dF$x * dF$y * dF$z

# 6. volume

plot(price ~ volume, data = dF, main = "Price vs volume", col ="blue", pch = 16)

abline(lm(price ~ volume, data = dF))
```

Correlation of numeric data

```
corrplot(cor(dF[,c('price', 'carat','depth','table', 'x','y','z','volume')]), type="full", method = "number"
)
```

factor coversion

```
# cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)
```

```
# color diamond colour, from J (worst) to D (best)

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

cut_levels <- c("Fair", "Good", "Very Good", "Premium", "Ideal")

clarity_levels <- c("I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1", "IF")

color_levels <- c("D", "E", "F", "G", "H", "I", "J")

dF$color = factor(dF$color, levels=color_levels)

dF$cut = factor(dF$cut, levels=cut_levels)

dF$clarity = factor(dF$clarity, levels=clarity_levels)
```

Splitting dataset into 15% test and 85% training data

```
# all variables

set.seed(2)

samples <- sample(1:nrow(dF), size=nrow(dF)/15)

dF.test <- dF[samples, ]

dF.train <- dF[-samples, ]

cat(sprintf("Size of full dataset: %d\n", nrow(dF)))

## Size of full dataset: 53940

cat(sprintf("Size of train dataset: %d\n", nrow(dF.train)))

## Size of train dataset: 50344

cat(sprintf("Size of test dataset: %d\n", nrow(dF.test)))

## Size of test dataset: 3596
```

Naive model

Model with all the variables

```
model <- lm(price~., data=dF.train)

summary(model)

##
```

```
## Call:
## lm(formula = price ~ ., data = dF.train)
##
## Residuals:
##       Min      1Q   Median    3Q       Max
## -21775.0   -583.0   -182.0  369.5  10687.9
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  203.9590   436.5525   0.467   0.6404
## carat        9029.2102   131.1043  68.870  < 2e-16 ***
## cutGood      621.9299        34.8076  17.868  < 2e-16 ***
## cutVery Good   766.7082  33.3973  22.957  < 2e-16 ***
## cutPremium          746.1936        33.2678  22.430  < 2e-16 ***
## cutIdeal     847.7635        34.5369  24.547  < 2e-16 ***
## colorE       -209.7999       18.5294 -11.323  < 2e-16 ***
## colorF       -268.9276       18.7255 -14.362  < 2e-16 ***
## colorG       -486.7055       18.3244 -26.561  < 2e-16 ***
## colorH       -985.2102       19.5112 -50.495  < 2e-16 ***
## colorI       -1474.8514     21.9012 -67.341  < 2e-16 ***
## colorJ       -2369.6234     27.0546 -87.587  < 2e-16 ***
## claritySI2   2714.1696      45.2125  60.031  < 2e-16 ***
## claritySI1   3688.4406      45.0198  81.929  < 2e-16 ***
## clarityVS2   4286.5634      45.2373  94.757  < 2e-16 ***
## clarityVS1   4598.6247      45.9542 100.070  < 2e-16 ***
## clarityVVS2   4969.9614   47.3136 105.043  < 2e-16 ***
```

```
## clarityVVS1   5027.5068   48.6416 103.358  < 2e-16 ***
## clarityIF     5363.3394     52.6398 101.887  < 2e-16 ***
## depth         -29.2826      5.0695  -5.776 7.68e-09 ***
## table         -21.0513      3.0268  -6.955 3.57e-12 ***
## x             135.5151     71.2574   1.902   0.0572 .
## y             -965.8150    57.3278 -16.847  < 2e-16 ***
## z             -530.4601    42.7830 -12.399  < 2e-16 ***
## volume        15.5787       0.8473  18.385  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1128 on 50319 degrees of freedom
## Multiple R-squared:  0.9204, Adjusted R-squared:  0.9204
## F-statistic: 2.424e+04 on 24 and 50319 DF,  p-value: < 2.2e-16
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f RMSE train: %.2f RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
## r-squared: 0.9204  RMSE train: 1128.14  RMSE test: 1121.65
#
```

Model without depth & table variable

```
model <- lm(price~carat+cut+color+clarity+x+y+z+volume, data=dF.train)
summary(model)
##
## Call:
## lm(formula = price ~ carat + cut + color + clarity + x + y +
```

```
##       z + volume, data = dF.train)
##
## Residuals:
##       Min      1Q   Median    3Q       Max
## -21731.9   -584.1   -180.9  369.8  10726.3
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2903.2801   100.7675 -28.812  < 2e-16 ***
## carat          8740.3631   118.2067  73.941  < 2e-16 ***
## cutGood      669.2425        34.1236  19.612  < 2e-16 ***
## cutVery Good   840.9800  31.8104  26.437  < 2e-16 ***
## cutPremium          809.3008        31.9653  25.318  < 2e-16 ***
## cutIdeal      960.6723        31.3365  30.657  < 2e-16 ***
## colorE        -210.6275       18.5393 -11.361  < 2e-16 ***
## colorF        -268.3558       18.7360 -14.323  < 2e-16 ***
## colorG        -487.1515       18.3288 -26.578  < 2e-16 ***
## colorH        -987.3128       19.5130 -50.598  < 2e-16 ***
## colorI         -1477.7137     21.9065 -67.456  < 2e-16 ***
## colorJ            -2372.8899  27.0650 -87.674  < 2e-16 ***
## claritySI2    2721.2285      45.2200  60.178  < 2e-16 ***
## claritySI1    3694.5034      45.0334  82.039  < 2e-16 ***
## clarityVS2  4294.8182      45.2405  94.933  < 2e-16 ***
## clarityVS1  4609.2537      45.9438 100.324  < 2e-16 ***
## clarityVVS2   4980.9884   47.3020 105.302  < 2e-16 ***
## clarityVVS1   5039.4059   48.6230 103.642  < 2e-16 ***
```

```
## clarityIF      5381.0920      52.5923 102.317  < 2e-16 ***
## x               297.7640      61.9615   4.806 1.55e-06 ***
## y             -1064.4957      54.1047 -19.675  < 2e-16 ***
## z              -631.2227      35.2467 -17.909  < 2e-16 ***
## volume           17.2940       0.7804  22.160  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1129 on 50321 degrees of freedom
## Multiple R-squared:  0.9203, Adjusted R-squared:  0.9203
## F-statistic: 2.641e+04 on 22 and 50321 DF,  p-value: < 2.2e-16
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f   RMSE train: %.2f   RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
## r-squared: 0.9203  RMSE train: 1128.81  RMSE test: 1127.28
```

Model without x, y, z variables

```
model <- lm(price~carat+cut+color+clarity+depth+table+volume, data=dF.train)
summary(model)
##
## Call:
## lm(formula = price ~ carat + cut + color + clarity + depth +
##     table + volume, data = dF.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -16852.8   -679.1   -198.0  464.7  10337.9
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -4523.7563   388.6553 -11.640  < 2e-16 ***
## carat         8780.6256    50.6026 173.521  < 2e-16 ***
## cutGood        607.4251    35.5942  17.065  < 2e-16 ***
## cutVery Good   762.3545    34.0982  22.358  < 2e-16 ***
## cutPremium     801.7848    34.1151  23.502  < 2e-16 ***
## cutIdeal       863.0519    35.3867  24.389  < 2e-16 ***
## colorE        -210.4457    19.0131 -11.068  < 2e-16 ***
## colorF        -305.0745    19.2014 -15.888  < 2e-16 ***
## colorG        -514.5705    18.7947 -27.378  < 2e-16 ***
## colorH        -979.8656    20.0197 -48.945  < 2e-16 ***
## colorI       -1441.7183    22.4635 -64.180  < 2e-16 ***
## colorJ       -2318.6562    27.7429 -83.576  < 2e-16 ***
## claritySI2    2620.9618    46.3554  56.541  < 2e-16 ***
## claritySI1    3577.6910    46.1415  77.537  < 2e-16 ***
## clarityVS2    4219.7049    46.3963  90.949  < 2e-16 ***
## clarityVS1    4534.3120    47.1309  96.207  < 2e-16 ***
## clarityVVS2   4968.9327    48.5414 102.365  < 2e-16 ***
## clarityVVS1   5074.9538    49.8910 101.721  < 2e-16 ***
## clarityIF     5413.0967    53.9875 100.266  < 2e-16 ***
## depth          -21.3052     4.2471  -5.016 5.28e-07 ***
## table          -25.0485     3.0932  -8.098 5.71e-16 ***
## volume           0.7539     0.3030   2.488   0.0128 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1158 on 50322 degrees of freedom
## Multiple R-squared:  0.9162, Adjusted R-squared:  0.9161
## F-statistic: 2.619e+04 on 21 and 50322 DF,  p-value: < 2.2e-16
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f RMSE train: %.2f RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
## r-squared: 0.9161  RMSE train: 1157.65  RMSE test: 1129.73
```

Model without volume variable

```
model <- lm(price~carat+cut+color+clarity+depth+table+x+y+z, data=dF.train)
summary(model)
##
## Call:
## lm(formula = price ~ carat + cut + color + clarity + depth +
##     table + x + y + z, data = dF.train)
##
## Residuals:
##      Min       1Q   Median     3Q      Max
## -21391.8   -593.3   -183.8  375.6  10691.3
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2299.377    422.821   5.438 5.41e-08 ***
```

```
## carat         11255.538      50.416 223.254  < 2e-16 ***
## cutGood     575.979          34.834  16.535  < 2e-16 ***
## cutVery Good   714.972    33.390  21.413  < 2e-16 ***
## cutPremium          761.211          33.369  22.812  < 2e-16 ***
## cutIdeal      822.050          34.624  23.742  < 2e-16 ***
## colorE        -208.021         18.591 -11.189  < 2e-16 ***
## colorF        -273.200         18.787 -14.542  < 2e-16 ***
## colorG        -489.215         18.385 -26.609  < 2e-16 ***
## colorH        -981.380         19.575 -50.134  < 2e-16 ***
## colorI      -1468.750         21.972 -66.847  < 2e-16 ***
## colorJ        -2363.037       27.143 -87.060  < 2e-16 ***
## claritySI2    2702.102        45.359  59.572  < 2e-16 ***
## claritySI1    3672.786        45.162  81.324  < 2e-16 ***
## clarityVS2  4273.755        45.383  94.171  < 2e-16 ***
## clarityVS1  4584.650        46.102  99.447  < 2e-16 ***
## clarityVVS2   4960.637     47.469 104.503  < 2e-16 ***
## clarityVVS1   5020.496     48.803 102.873  < 2e-16 ***
## clarityIF      5351.928        52.812 101.339  < 2e-16 ***
## depth         -65.487          4.687 -13.972  < 2e-16 ***
## table         -26.888          3.020  -8.903  < 2e-16 ***
## x             -1009.168        34.774 -29.020  < 2e-16 ***
## y             10.086           21.727   0.464 0.642
## z             -44.237          33.742  -1.311 0.190
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1132 on 50320 degrees of freedom
## Multiple R-squared:  0.9199, Adjusted R-squared:  0.9198
## F-statistic: 2.511e+04 on 23 and 50320 DF,  p-value: < 2.2e-16
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f  RMSE train: %.2f  RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
## r-squared: 0.9198  RMSE train: 1131.92  RMSE test: 1100.70
```

Model with just the carat value

```
model <- lm(price~carat, data=dF.train)
summary(model)
##
## Call:
## lm(formula = price ~ carat, data = dF.train)
##
## Residuals:
##      Min      1Q  Median     3Q     Max
## -18632.1  -805.6  -17.8  539.1  12725.7
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2261.10      13.52  -167.2   <2e-16 ***
## carat        7766.71      14.56   533.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1551 on 50342 degrees of freedom

## Multiple R-squared:  0.8496, Adjusted R-squared:  0.8496

## F-statistic: 2.844e+05 on 1 and 50342 DF,  p-value: < 2.2e-16

rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))

rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))

cat(sprintf("r-squared: %.4f  RMSE train: %.2f  RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))

## r-squared: 0.8496  RMSE train: 1550.53  RMSE test: 1520.45
```

Model without table variable

```
model <- lm(price~carat+cut+color+clarity+depth+x+y+z+volume, data=dF.train)

summary(model)

##

## Call:

## lm(formula = price ~ carat + cut + color + clarity + depth +

##     x + y + z + volume, data = dF.train)

##

## Residuals:

##      Min       1Q   Median     3Q      Max

## -21764.0   -583.1   -181.8  370.3  10730.7

##

## Coefficients:

##                Estimate Std. Error t value Pr(>|t|)

## (Intercept)  -1855.6710   320.9006  -5.783 7.39e-09 ***

## carat         8929.6192   130.3813  68.488  < 2e-16 ***

## cutGood        649.5732    34.5962  18.776  < 2e-16 ***

## cutVery Good   815.2536    32.6751  24.950  < 2e-16 ***
```

```
## cutPremium          783.1083       32.8571  23.834  < 2e-16 ***
## cutIdeal     937.0730       32.0760  29.214  < 2e-16 ***
## colorE       -210.8903      18.5374 -11.376  < 2e-16 ***
## colorF       -268.2159      18.7340 -14.317  < 2e-16 ***
## colorG       -485.6126      18.3323 -26.489  < 2e-16 ***
## colorH       -985.2201      19.5204 -50.471  < 2e-16 ***
## colorI       -1475.8152     21.9111 -67.355  < 2e-16 ***
## colorJ       -2371.2504     27.0663 -87.609  < 2e-16 ***
## claritySI2   2716.9897      45.2319  60.068  < 2e-16 ***
## claritySI1   3691.1055      45.0394  81.953  < 2e-16 ***
## clarityVS2  4290.2065       45.2555  94.800  < 2e-16 ***
## clarityVS1  4603.3611       45.9708 100.137  < 2e-16 ***
## clarityVVS2   4974.8401    47.3307 105.108  < 2e-16 ***
## clarityVVS1   5032.5065    48.6592 103.424  < 2e-16 ***
## clarityIF     5372.2910    52.6489 102.040  < 2e-16 ***
## depth        -16.1932       4.7094  -3.438 0.000585 ***
## x            178.3530       71.0241   2.511 0.012037 *
## y            -1001.4035     57.1259 -17.530  < 2e-16 ***
## z            -548.1593      42.7273 -12.829  < 2e-16 ***
## volume       16.1968        0.8431  19.212  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1129 on 50320 degrees of freedom
## Multiple R-squared:  0.9203, Adjusted R-squared:  0.9203
## F-statistic: 2.527e+04 on 23 and 50320 DF,  p-value: < 2.2e-16
```

```
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))

rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))

cat(sprintf("r-squared: %.4f RMSE train: %.2f RMSE test: %.2f",
summary(model)$adj.r.squared, rmse.train, rmse.test))

## r-squared: 0.9203  RMSE train: 1128.68  RMSE test: 1123.67
```

Model with all variables and 5 fold cross validation

```
model <- lm(price~., data=dF.train)

summary(model)

##

## Call:

## lm(formula = price ~ ., data = dF.train)

##

## Residuals:

##      Min      1Q   Median   3Q      Max

## -21775.0   -583.0   -182.0  369.5  10687.9

##

## Coefficients:

##               Estimate Std. Error t value Pr(>|t|)

## (Intercept)  203.9590   436.5525   0.467   0.6404

## carat        9029.2102   131.1043  68.870  < 2e-16 ***

## cutGood      621.9299     34.8076  17.868  < 2e-16 ***

## cutVery Good   766.7082  33.3973  22.957  < 2e-16 ***

## cutPremium      746.1936    33.2678  22.430  < 2e-16 ***

## cutIdeal     847.7635     34.5369  24.547  < 2e-16 ***

## colorE       -209.7999    18.5294 -11.323  < 2e-16 ***

## colorF       -268.9276    18.7255 -14.362  < 2e-16 ***
```

```
## colorG       -486.7055       18.3244 -26.561  < 2e-16 ***
## colorH       -985.2102       19.5112 -50.495  < 2e-16 ***
## colorI       -1474.8514      21.9012 -67.341  < 2e-16 ***
## colorJ       -2369.6234      27.0546 -87.587  < 2e-16 ***
## claritySI2   2714.1696       45.2125  60.031  < 2e-16 ***
## claritySI1   3688.4406       45.0198  81.929  < 2e-16 ***
## clarityVS2   4286.5634       45.2373  94.757  < 2e-16 ***
## clarityVS1   4598.6247       45.9542 100.070  < 2e-16 ***
## clarityVVS2   4969.9614      47.3136 105.043  < 2e-16 ***
## clarityVVS1   5027.5068      48.6416 103.358  < 2e-16 ***
## clarityIF    5363.3394       52.6398 101.887  < 2e-16 ***
## depth        -29.2826        5.0695  -5.776 7.68e-09 ***
## table        -21.0513        3.0268  -6.955 3.57e-12 ***
## x            135.5151        71.2574   1.902   0.0572 .
## y            -965.8150       57.3278 -16.847  < 2e-16 ***
## z            -530.4601       42.7830 -12.399  < 2e-16 ***
## volume       15.5787         0.8473  18.385  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1128 on 50319 degrees of freedom
## Multiple R-squared:  0.9204, Adjusted R-squared:  0.9204
## F-statistic: 2.424e+04 on 24 and 50319 DF,  p-value: < 2.2e-16
rmse.train <- sqrt(mean((predict(model, newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(model, newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f RMSE train: %.2f RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
```

```
## r-squared: 0.9204  RMSE train: 1128.14  RMSE test: 1121.65

library(DAAG)

cvlm.model <- CVlm(data=dF.train,model, m=5, seed=2, plotit = TRUE, printit = FALSE)

## Warning in CVlm(data = dF.train, model, m = 5, seed = 2, plotit = TRUE, :

##

##  As there is >1 explanatory variable, cross-validation

##  predicted values for a fold are not a linear function

##  of corresponding overall predicted values.  Lines that

##  are shown for the different folds are approximate

library(lasso2)

## R Package to solve regression problems while imposing

##   an L1 constraint on the parameters. Based on S-plus Release 2.1

## Copyright (C) 1998, 1999

## Justin Lokhorst   <jlokhors@stats.adelaide.edu.au>

## Berwin A. Turlach <bturlach@stats.adelaide.edu.au>

## Bill Venables         <wvenable@stats.adelaide.edu.au>

##

## Copyright (C) 2002

## Martin Maechler <maechler@stat.math.ethz.ch>

bounds = c(0.001, 0.01, 0.1, 1, 2, 4, 8, 16, 32, 64, 128, 256, 1024, 2048, 4096, 2^13, 2^14,
2^14, 2^15, 2^16)

lasso.models <- l1ce(price~carat+cut+color+clarity+x+y+z+volume, data=dF.train, absolute.t =
TRUE, standardize = TRUE, bound = bounds)

cat("Index\t   Bounds\tMSE\n")

## Index         Bounds   MSE

for (index in 1:length(bounds)) {

 mse <- sqrt(mean((predict(lasso.models[index]) - dF.train$price)^2))
```

```
cat(sprintf("%4d\t%10.2f\t%.4f\n", index, bounds[index], mse));
}
##      1         0.00  3998.4785
##      2         0.01  3998.4702
##      3         0.10  3998.3872
##      4         1.00  3997.5577
##      5         2.00  3996.6360
##      6         4.00  3994.7928
##      7         8.00  3991.1067
##      8         16.00  3983.7364
##      9         32.00  3969.0031
##   10           64.00  3939.5661
##   11 128.00  3880.8138
##   12           256.00  3763.8227
##   13 1024.00  3080.3162
##   14 2048.00  2255.2092
##   15 4096.00  1460.3442
##   16 8192.00  1228.0749
##   17   16384.00  1136.8715
##   18   16384.00  1136.8715
##   19   32768.00  1128.8115
##   20   65536.00  1128.8115
#summary(lasso.models[19])
rmse.train <- sqrt(mean((predict(lasso.models[19], newdata = dF.train) - dF.train$price)^2))
rmse.test <- sqrt(mean((predict(lasso.models[19], newdata = dF.test) - dF.test$price)^2))
cat(sprintf("r-squared: %.4f RMSE train: %.2f RMSE test: %.2f", summary(model)$adj.r.squared, rmse.train, rmse.test))
```

```
## r-squared: 0.9204  RMSE train: 1128.81  RMSE test: 1127.28
```

Plotting the best model output on the test and train data

```
model <- lm(price~carat+cut+color+clarity+depth+x+y+z+volume, data=dF.train)
plot(predict(model, newdata = dF.train) ~ dF.train$price,
        main = "Regression model on Train Data",
        xlab = "Actual Price",
    ylab = "Prediced Price",
        xlim = c(0, 20000),
        ylim = c(0, 20000))
abline(0,1)
plot(predict(model, newdata = dF.test) ~ dF.test$price,
        main = "Regression model on Test Data",
        xlab = "Actual Price",
        ylab = "Prediced Price",
        xlim = c(0, 20000),
        ylim = c(0, 20000))
abline(0,1)
```