SVM Regression

Regression Algorithms

One of the many applications that regression algorithms are used for is to predict the prices for a product. In this case, we will work with a cubic zirconia data set to predict the price of cubic zirconia gemstones based on attributes such as cut, color, and so on. Although R has a built-in 'diamonds' dataset included with the ggplot2 package, it would be interesting to see what attributes are important to distinguish the quality of cubic zirconia gemstones, which are generally views as a cheaper alternative to diamonds. For this exploration, we would like to implement and compare the performances of the following SVM kernels: linear, polynomial, and radial.

Load the Data

Dataset: 'Gemstone Price Prediction' dataset

```
df <- read.csv("cubic_zirconia.csv")
set.seed(1234)
i <- sample(1:nrow(df), 10000, replace=FALSE)
df <- df[i,]
head(df)
```

```
##           X carat       cut color clarity depth table    x    y    z price
## 7452   7452  1.24   Premium     D     SI1  62.4    59 6.82 6.86 4.27  7351
## 8016   8016  0.30     Ideal     F      IF    NA    57 4.29 4.31 2.68   886
## 7162   7162  1.01 Very Good     F     SI2  59.1    57 6.57 6.62 3.90  4304
## 8086   8086  0.38     Ideal     E     VS1  62.1    57 4.65 4.62 2.88  1097
## 23653 23653  1.20   Premium     H     VS2  61.4    59 6.79 6.75 4.16  6333
## 9196   9196  2.02   Premium     G     SI2  61.3    56 8.22 8.16 5.02 16929
```

Data Cleaning

Before dividing the data into train/test sets, it is good to handle row instances that may be missing information or is a duplicate. Since we may not know what attributes would significantly affect the outcome price of a cubic zirconia gemstone, we will simply omit rows with NA's instead of filling them in with some mean value. Row duplicates will also be omitted. We will also remove the first column since its value is the row instance number, which we do not need.

There are three qualitative attributes in this dataset: cut, color, clarity. Before examining the distributions of these three attributes, change these columns so that they are factors instead of characters.

```
print(paste("Number of rows (before): ", nrow(df)))
```

```
## [1] "Number of rows (before):  10000"
```

```r
# Delete rows with NA's or are duplicates
df <- na.omit(df)
df <- unique(df)

print(paste("Number of rows (after): ", nrow(df)))
```

```
## [1] "Number of rows (after):  9750"
```

```r
# Remove first column
df$X <- NULL

# Convert chr columns to be factors
df$cut <- factor(df$cut, levels = c("Fair", "Good", "Very Good", "Premium", "Ideal"))
df$color <- as.factor(df$color)
df$clarity <- factor(df$clarity, levels = c("I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1", "IF"))
str(df)
```

```
## 'data.frame':    9750 obs. of  10 variables:
##  $ carat  : num  1.24 1.01 0.38 1.2 2.02 1 0.41 0.53 2.01 2.07 ...
##  $ cut    : Factor w/ 5 levels "Fair","Good",..: 4 3 5 4 4 2 5 5 2 5 ...
##  $ color  : Factor w/ 7 levels "D","E","F","G",..: 1 3 2 5 4 7 6 3 6 4 ...
##  $ clarity: Factor w/ 8 levels "I1","SI2","SI1",..: 3 2 5 4 2 4 7 5 2 2 ...
##  $ depth  : num  62.4 59.1 62.1 61.4 61.3 58.7 61.3 60.7 56.9 62.5 ...
##  $ table  : num  59 57 57 59 56 61 56 55 59 55 ...
##  $ x      : num  6.82 6.57 4.65 6.79 8.22 6.5 4.85 5.32 8.41 8.2 ...
##  $ y      : num  6.86 6.62 4.62 6.75 8.16 6.52 4.8 5.25 8.3 8.13 ...
##  $ z      : num  4.27 3.9 2.88 4.16 5.02 3.82 2.96 3.21 4.74 5.11 ...
##  $ price  : int  7351 4304 1097 6333 16929 3769 1107 1697 11312 18804 ...
##  - attr(*, "na.action")= 'omit' Named int [1:250] 2 55 58 83 135 201 248 263 353 396 ...
##   ..- attr(*, "names")= chr [1:250] "8016" "23910" "902" "10905" ...
```

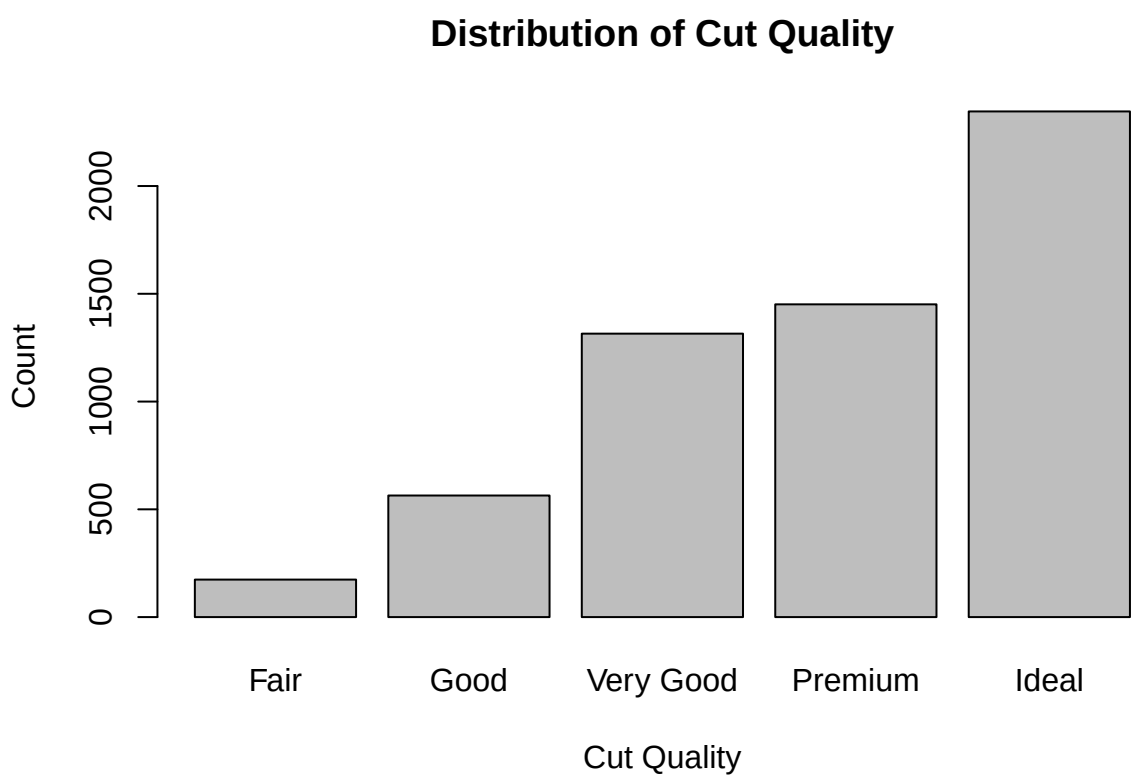
Divide Train/Test

```r
spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(df), nrow(df)*cumsum(c(0,spec)), labels=names(spec)), replace=FALSE)
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
```

Data Exploration

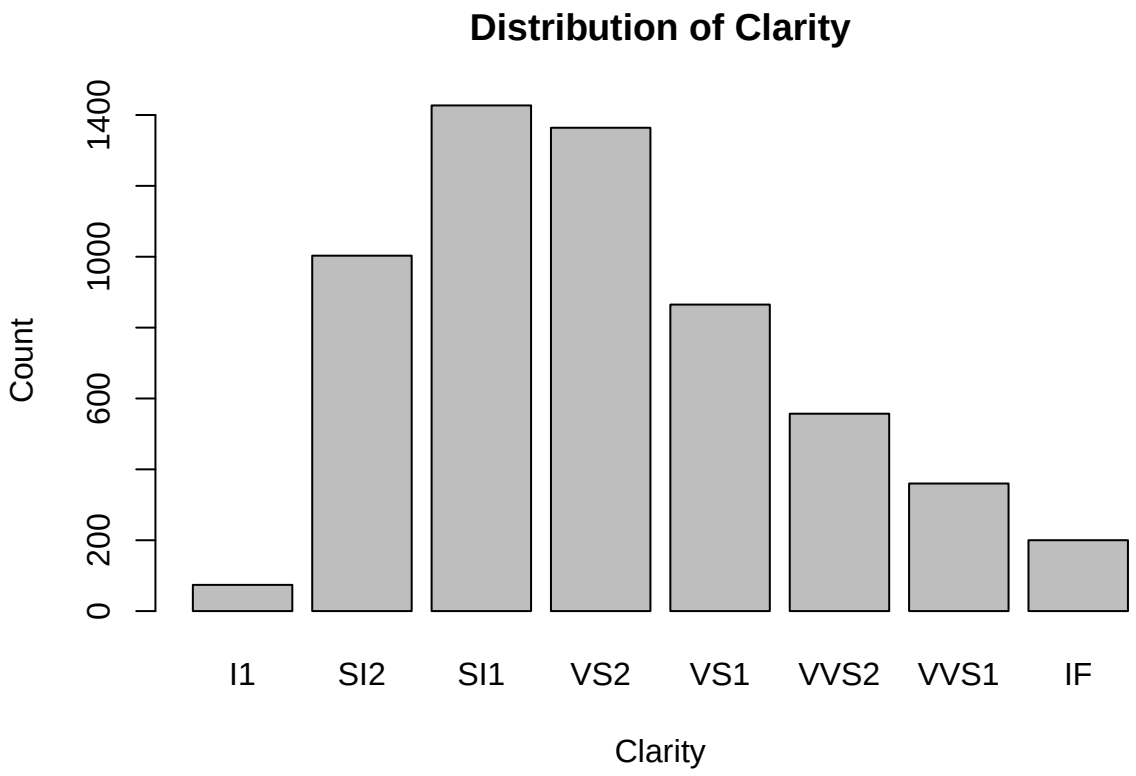
First, we will examine the distribution of the attributes individually with graph visualization. Then, we will analyze trends between the predictor attributes and the target attribute, price.

```r
#table(train$cut)
barplot(table(train$cut),
        main="Distribution of Cut Quality", xlab="Cut Quality", ylab="Count")
```

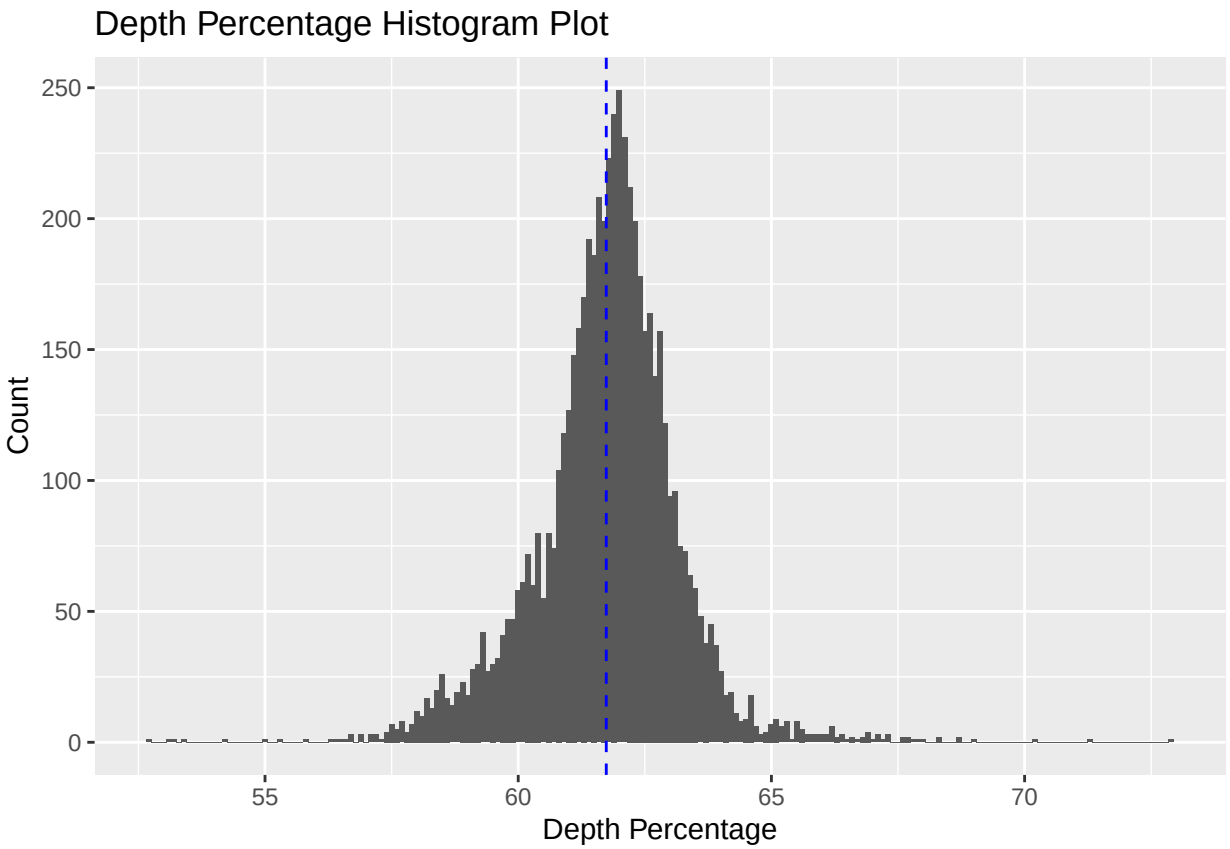



```
#table(train$color)
barplot(table(train$color),
        main="Distribution of Color", xlab="Color", ylab="Count")
```

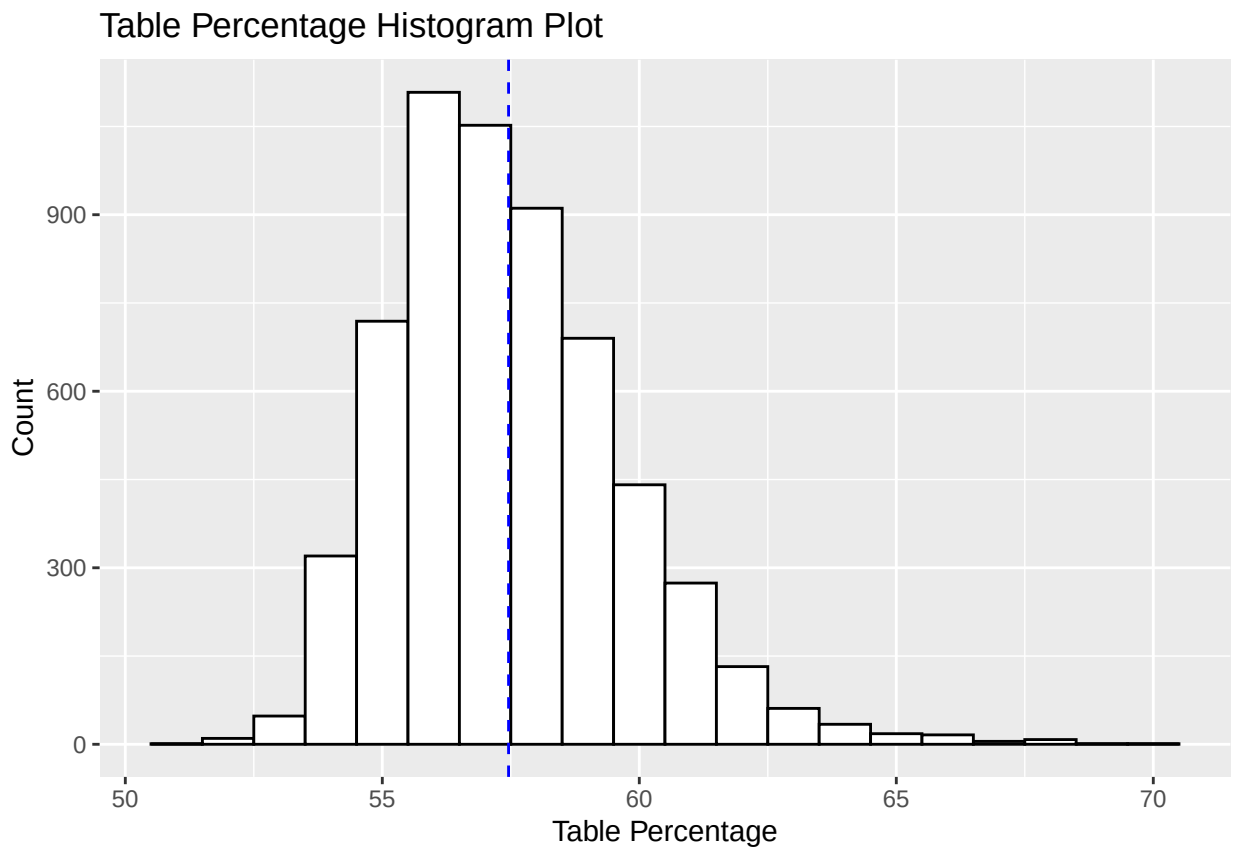

```
#table(train$clarity)
barplot(table(train$clarity),
        main="Distribution of Clarity", xlab="Clarity", ylab="Count")
```



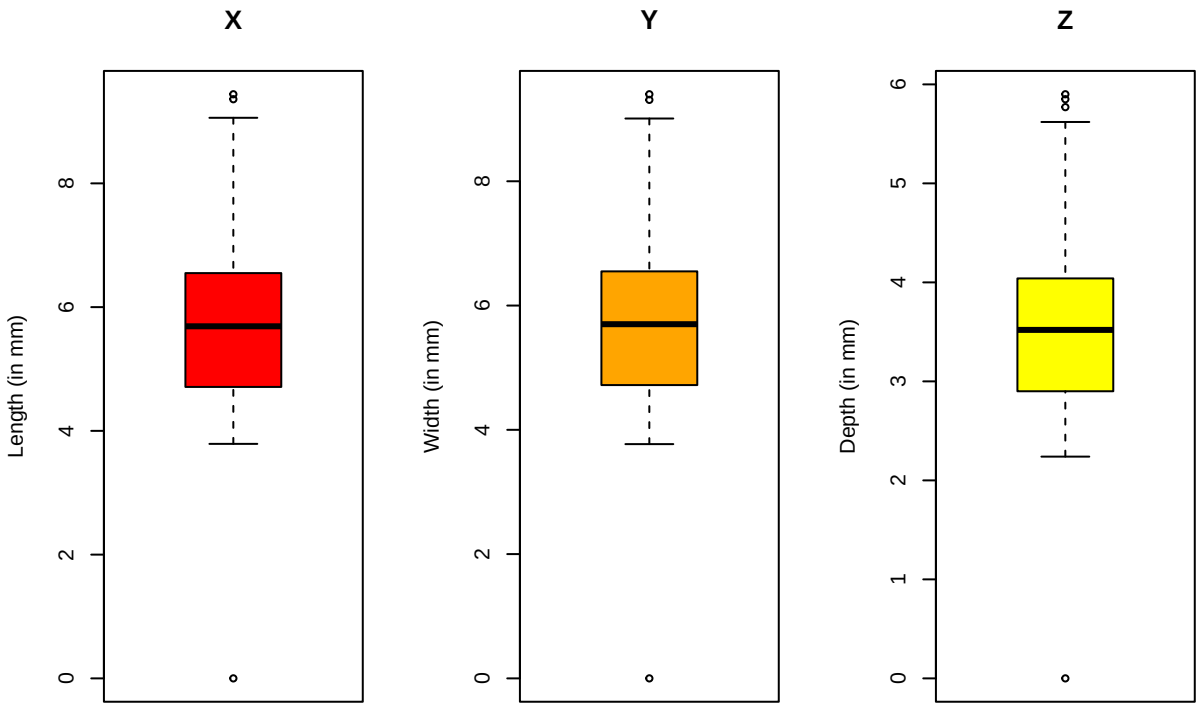

```
library(ggplot2)

# Total Depth Percentage
ggplot(train, aes(x=depth)) +
  geom_histogram(binwidth = 0.1) +
  geom_vline(aes(xintercept=mean(depth)), color = "blue", linetype = "dashed") +
  labs(title = "Depth Percentage Histogram Plot", x = "Depth Percentage", y = "Count")
```




```
# Table Percentage
ggplot(train, aes(x=table)) +
  geom_histogram(binwidth = 1, color = "black", fill = "white") +
  geom_vline(aes(xintercept=mean(table)), color = "blue", linetype = "dashed") +
  labs(title = "Table Percentage Histogram Plot", x = "Table Percentage", y = "Count")
```



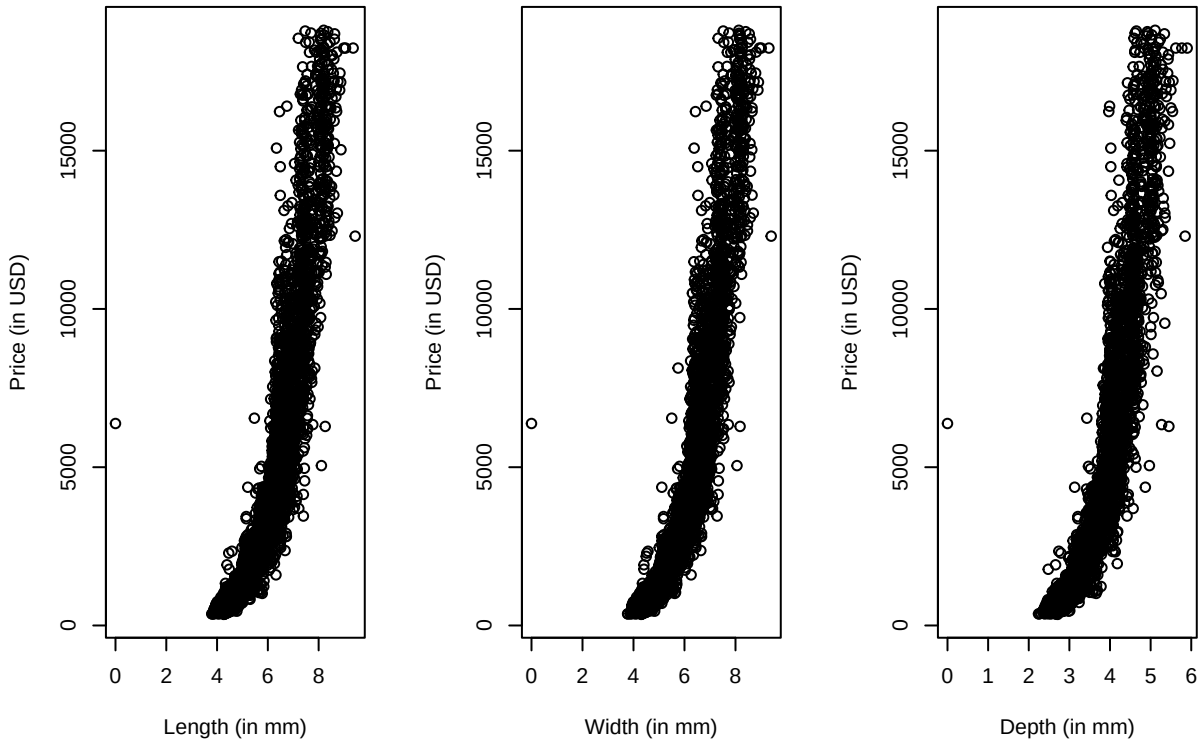

```
par(mfrow = c(1,3))

# Length, x
boxplot(train$x,
        main = "X",
        ylab = "Length (in mm)",
        col = "red")

# Width, y
boxplot(train$y,
        main = "Y",
        ylab = "Width (in mm)",
        col = "orange")

# Depth, z
boxplot(train$z,
        main = "Z",
        ylab = "Depth (in mm)",
        col = "yellow")
```




```
par(mfrow = c(1,1))
```

After analyzing the attributes individually, we can see that there are some outliers for dimensions x, y, and z that we may have to clean up. Let's plot these attributes in relation to price to have a better visualization for how many outliers there are.

```
par(mfrow = c(1,3))
plot(train$x, train$price,
      main = "", xlab = "Length (in mm)", ylab = "Price (in USD)")

plot(train$y, train$price,
     main = "", xlab = "Width (in mm)", ylab = "Price (in USD)")

plot(train$z, train$price,
     main = "", xlab = "Depth (in mm)", ylab = "Price (in USD)")

mtext("Cubic Zirconia Dimensions in Relation to Price", side = 3, line = -2, outer = TRUE)
```




```
par(mfrow = c(1,1))
```

There are a few zeros for the gemstone dimensions, but since there are very few of these cases, it would be alright to remove these instances from the dataset.

```
print(paste("Number of train rows (before): ", nrow(train)))
```

```
## [1] "Number of train rows (before):  5850"
```

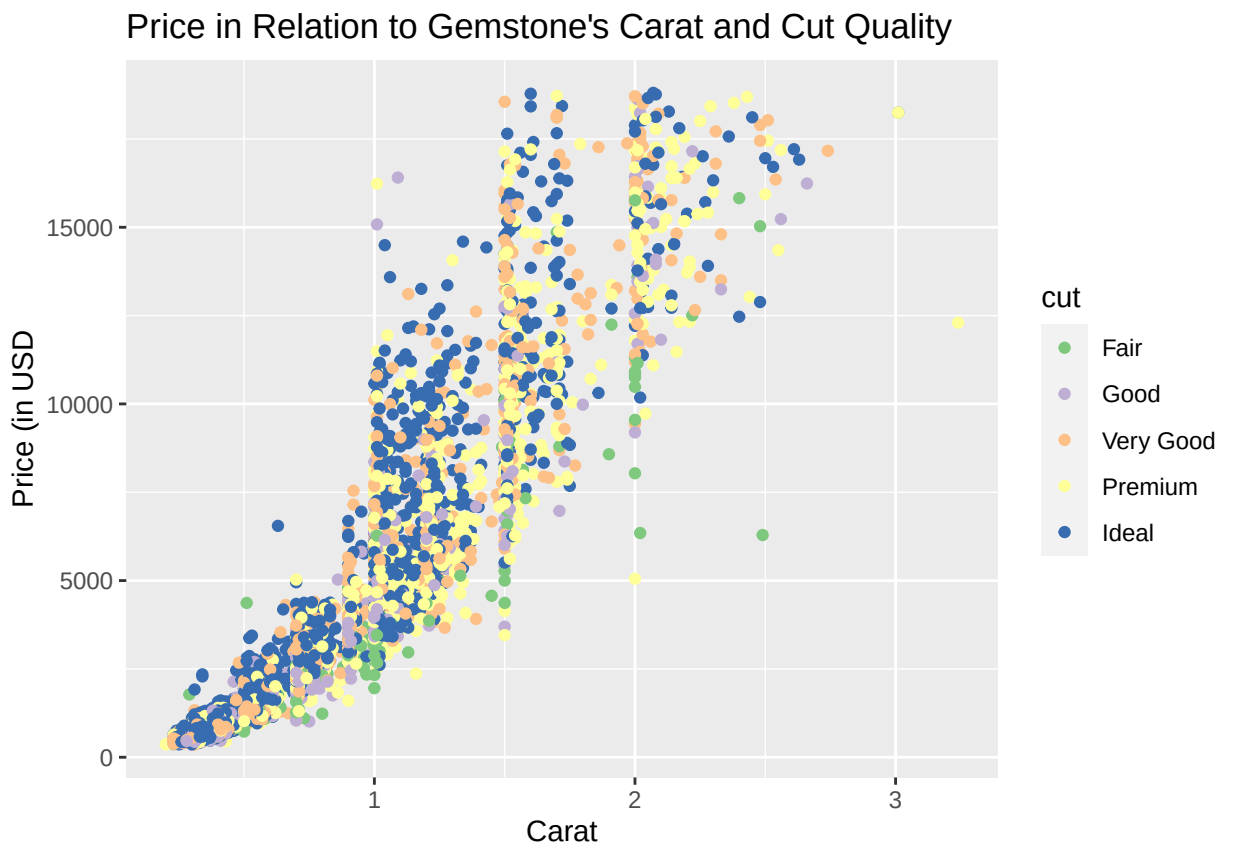
```
# Delete rows with dimension length of 0
train <- train[!(train$x == 0),]
train <- train[!(train$y == 0),]
train <- train[!(train$z == 0),]

print(paste("Number of train rows (after): ", nrow(train)))
```

```
## [1] "Number of train rows (after):  5849"
```

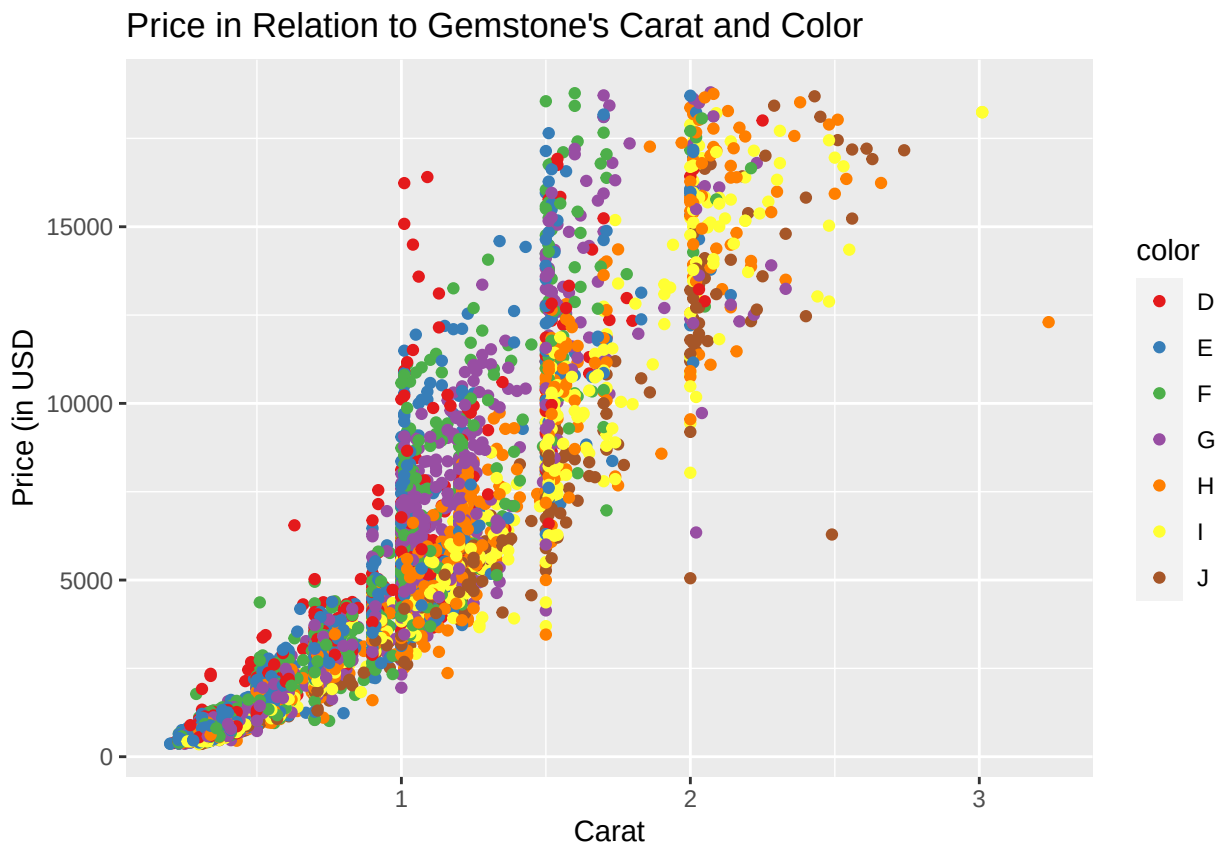
There are some outliers for the dimensions x, y, and z that can be seen from our analysis of these attributes so far. However, gemstones probably come in all sorts of different sizes and shapes, so it should be alright to leave the outliers alone.

Besides the physical size and shape of the cubic zirconia gemstone, its carat value or weight seems like it could be a good predictor for predicting price. Let's also examine how the categories of cut, color, and clarity also fit into the trend between carat and price.

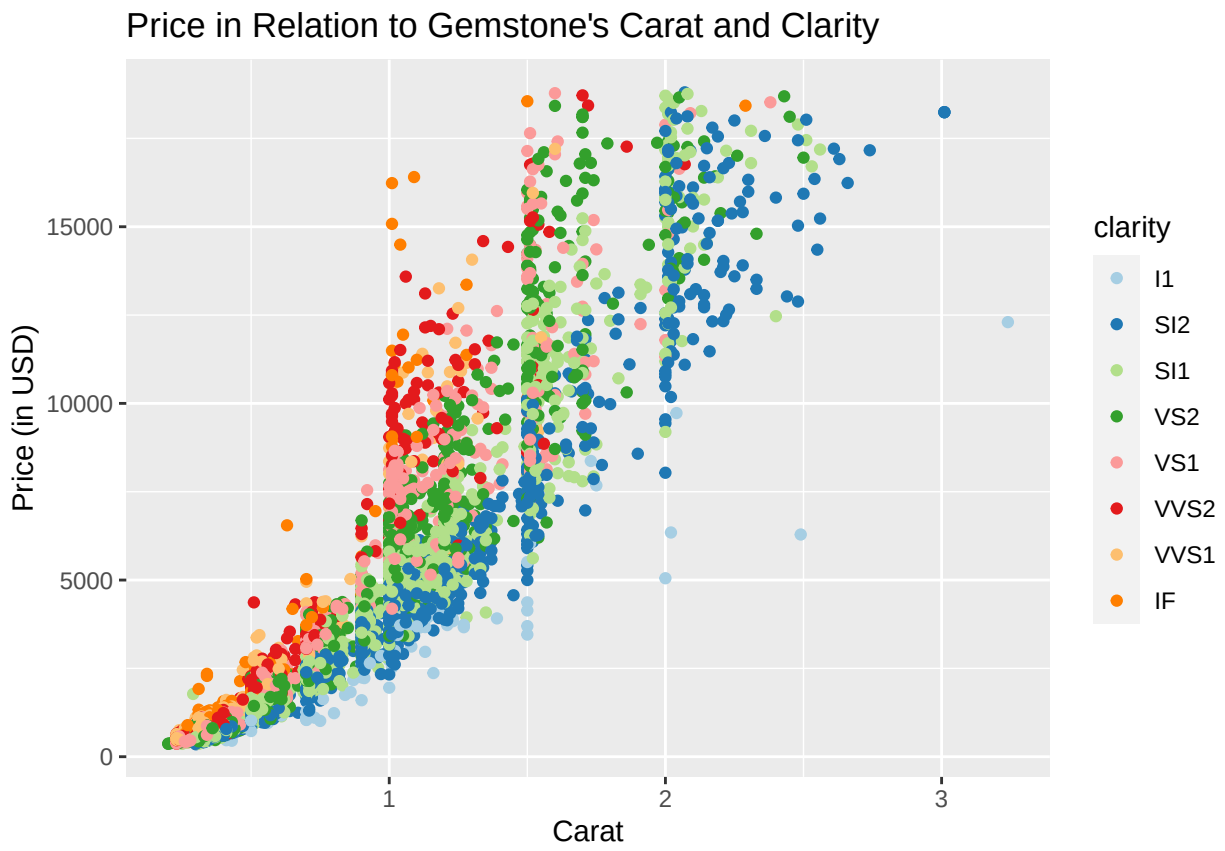
```
ggplot(train) + geom_point(aes(x=carat,y=price,colour=cut)) +
  labs(title = "Price in Relation to Gemstone's Carat and Cut Quality", x = "Carat", y = "Price (in USD
    scale_color_brewer(palette = "Accent")
```




```
ggplot(train) + geom_point(aes(x=carat,y=price,colour=color)) +
  labs(title = "Price in Relation to Gemstone's Carat and Color", x = "Carat", y = "Price (in USD") +
    scale_color_brewer(palette = "Set1")
```




```
ggplot(train) + geom_point(aes(x=carat,y=price,colour=clarity)) +
  labs(title = "Price in Relation to Gemstone's Carat and Clarity", x = "Carat", y = "Price (in USD)") +
  scale_color_brewer(palette = "Paired")
```



In the previous version of this notebook, which addressed linear and kNN regression, we found that physical dimensions are not good predictors, so in the models below only carat, cut, color, and clarity will be considered.

Convert strings to ints

```
train$cut <- as.integer(train$cut)
train$color <- as.integer(train$color)
train$clarity <- as.integer(train$clarity)

test$cut <- as.integer(test$cut)
test$color <- as.integer(test$color)
test$clarity <- as.integer(test$clarity)

vald$cut <- as.integer(vald$cut)
vald$color <- as.integer(vald$color)
vald$clarity <- as.integer(vald$clarity)
```

SVM Regression

Linear kernel

```
library(e1071)
tune_svm1 <- tune(svm, price~carat+cut+color+clarity, data=vald, kernel="linear", ranges=list(cost=c(0.
summary(tune_svm1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 2000947
##
## - Detailed performance results:
##    cost   error dispersion
## 1   0.1 2014606   571198.8
## 2   1.0 2001368   568210.0
## 3  10.0 2000947   568115.1
## 4 100.0 2000969   568183.1
```

```
pred <- predict(tune_svm1$best.model, newdata=test)
cor_svm1_tune <- cor(pred, test$price)
mse_svm1_tune <- mean((pred - test$price)^2)
paste("Correlation: ", cor_svm1_tune)
```

```
## [1] "Correlation:  0.948440795827825"
```

```
paste("MSE: ", mse_svm1_tune)
```

```
## [1] "MSE:  1810932.08428343"
```

Polynomial kernel

The polynomial kernel example has been removed here since it simply ran for a long time and crashed R. The following code demonstrates our attempt at using a polynomial kernel with SVM.

```
# tune_svm2 <- tune(svm, price~carat+cut+color+clarity, data=vald, kernel="polynomial", ranges=list(cos
# summary(tune_svm2)
# pred <- predict(tune_svm2$best.model, newdata=test)
# cor_svm2_tune <- cor(pred, test$price)
# mse_svm2_tune <- mean((pred - test$price)^2)
# paste("Correlation: ", cor_svm2_tune)
# paste("MSE: ", mse_svm2_tune)
```

Radial kernel

```
tune_svm3 <- tune(svm, price~carat+cut+color+clarity, data=vald, kernel="radial", ranges=list(cost=c(0.
summary(tune_svm3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 530451.1
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1    0.1   0.5 1581925.4   450540.5
## 2    1.0   0.5  542279.3   306424.0
## 3   10.0   0.5  530451.1   279135.0
## 4  100.0   0.5  743117.5   350680.7
## 5    0.1   1.0 3148615.6   608365.3
## 6    1.0   1.0  774114.5   380737.5
## 7   10.0   1.0  769115.1   405136.9
## 8  100.0   1.0 1055319.4   447369.9
## 9    0.1   2.0 6921339.3  1321312.1
## 10   1.0   2.0 1468181.5   450563.6
## 11  10.0   2.0 1370047.6   449007.7
## 12 100.0   2.0 1719348.8   535954.8
## 13   0.1   3.0 9655901.0  1836961.6
## 14   1.0   3.0 2291786.8   486970.4
## 15  10.0   3.0 2001579.6   511694.0
## 16 100.0   3.0 2450910.2   955008.9
```

```
pred <- predict(tune_svm3$best.model, newdata=test)
cor_svm3_tune <- cor(pred, test$price)
mse_svm3_tune <- mean((pred - test$price)^2)
paste("Correlation: ", cor_svm3_tune)
```

```
## [1] "Correlation:  0.983067152730392"
```

```
paste("MSE: ", mse_svm3_tune)
```

```
## [1] "MSE:  559650.963525154"
```

Analysis

The difference between SVM kernel types is much easier to visualize than to describe. Scikit-Learn has some excellent images on its documentation pages illustrating the difference between linear, polynomial, and

radial kernels. The linear kernel models the data according to a simple line, while the polynomial model approximates the form of the data using a polynomial equation. These are similar to how linear regression works. Perhaps more interesting is the radial kernel, which considers data points in more granular radial "clumps" rather than simple lines or polynomial equations.

The radial kernel seems to have performed best in this experiment, which makes sense as it emphasizes precision. The other parameters are not entirely intuitive, but tuning reveals that a cost of 10 and gamma of 0.5 are optimal for this dataset out of the tested combinations. It is interesting to note that a cost of 10 was best for both linear and radial kernels.