Exploring Regression Techniques - Diamonds Dataset

Ryan Yahnker

Set Up

Need to read the dataset, reclassify categorical variables, sample the data, and check that everything has been done correctly.

```
diamonds <- read.csv('Diamonds Prices2022.csv')
head(diamonds)
```

```
##   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
```

```
summary(diamonds)
```

```
##        X             carat            cut               color          
##  Min.   :    1   Min.   :0.2000   Length:53943       Length:53943      
##  1st Qu.:13486   1st Qu.:0.4000   Class :character   Class :character  
##  Median :26972   Median :0.7000   Mode  :character   Mode  :character  
##  Mean   :26972   Mean   :0.7979                                        
##  3rd Qu.:40458   3rd Qu.:1.0400                                        
##  Max.   :53943   Max.   :5.0100                                        
##    clarity              depth           table           price      
##  Length:53943       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  
```

```r
diamonds$cut = factor(diamonds$cut,
                      levels = c('Fair', 'Good', 'Very Good', 'Premium', 'Ideal'))
diamonds$color = factor(diamonds$color,
                        levels = c('D', 'E', 'F', 'G', 'H', 'I', 'J'))
diamonds$clarity = factor(diamonds$clarity,
                          levels = c('I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF'))

set.seed(101)

sample_diamonds <- diamonds[sample(1:nrow(diamonds), size = 500, replace = FALSE), ]
str(sample_diamonds)
```

```
## 'data.frame':    500 obs. of  11 variables:
##  $ X      : int  2873 43103 19665 21855 35772 46326 38688 14688 2531 43324 ...
##  $ carat  : num  1 0.5 1.41 1.24 0.4 0.6 0.37 1.25 0.7 0.51 ...
##  $ cut    : Factor w/ 5 levels "Fair","Good",..: 3 5 5 4 3 2 5 4 5 3 ...
##  $ color  : Factor w/ 7 levels "D","E","F","G",..: 6 5 7 1 2 4 2 6 4 1 ...
##  $ clarity: Factor w/ 8 levels "I1","SI2","SI1",..: 2 5 5 4 4 5 4 4 6 3 ...
##  $ depth  : num  62.4 61.6 61.6 59.3 61.2 60.1 61.6 62.2 61 61.7 ...
##  $ table  : num  63 58 56 58 60 61 57 57 56 58 ...
##  $ price  : int  3276 1384 8275 9916 912 1757 1041 5925 3205 1403 ...
##  $ x      : num  6.44 5.08 7.19 7.09 4.79 5.44 4.65 6.92 5.74 5.09 ...
##  $ y      : num  6.35 5.11 7.22 7.03 4.68 5.5 4.61 6.84 5.77 5.12 ...
##  $ z      : num  3.99 3.14 4.44 4.19 2.9 3.29 2.85 4.28 3.51 3.15 ...
```

Regression

Now we look to fit a linear regression model removing x, y, and z variables due to high correlation with carat variable.

We will use price as our dependent variable.

```r
diamonds_model <- lm(price ~ depth + cut + carat + color + table, data = sample_diamonds)
summary(diamonds_model)
```

```
##
## Call:
## lm(formula = price ~ depth + cut + carat + color + table, data = sample_diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10582.0   -654.7    -50.7    457.3   9114.2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2905.72    4758.01  -0.611 0.541683
## depth          -28.85      52.78  -0.547 0.584896
## cutGood        920.52     442.43   2.081 0.037993 *
## cutVery Good  1549.96     426.94   3.630 0.000313 ***
## cutPremium    1229.85     431.90   2.848 0.004593 **
## cutIdeal      1620.29     440.68   3.677 0.000263 ***
```

```
## carat         7557.20     153.80  49.136  < 2e-16 ***
## colorE         -87.95     259.33  -0.339 0.734635
## colorF        -196.31     262.70  -0.747 0.455257
## colorG        -104.83     243.15  -0.431 0.666571
## colorH        -796.27     264.72  -3.008 0.002767 **
## colorI        -946.71     290.32  -3.261 0.001189 **
## colorJ       -1497.65     353.88  -4.232 2.77e-05 ***
## table           25.47      38.96   0.654 0.513534
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1454 on 486 degrees of freedom
## Multiple R-squared:  0.841,  Adjusted R-squared:  0.8367
## F-statistic: 197.7 on 13 and 486 DF,  p-value: < 2.2e-16
```

Checking for multicollinearity (VIF & Condition Index)

Now that we have fit a model we need to check and adjust the model as need to find the best regression line.

First, we will check for multicollinearity. One way we can explore this is by calculating the Variance inflation factor (VIF).

Calculating VIF using R, we will use the vif function in the faraway library.

```
vif_data <- faraway::vif(diamonds_model)
vif_values <- data.frame(
  Coefficient=names(vif_data),
  VIF=as.numeric(vif_data)
)
kable(vif_values)
```

Coefficient	VIF
depth	1.327361
cutGood	4.385521
cutVery Good	8.080182
cutPremium	7.810604
cutIdeal	10.860483
carat	1.121647
colorE	2.243471
colorF	2.215037
colorG	2.648693
colorH	2.135606
colorI	1.888445
colorJ	1.512476
table	1.722300

We can see that the VIF of cutIdeal has a value over 10, which suggests that there could exist multicollinearity. However, this variable is a dummy variable, so this is not a cause for concern.

We will now look at other ways to explore multicollinearity in our model.

Another way is using condition index. The square root of the largest eigen value divided by the smallest eigen value gives us the condition number. When this number is larger than 30, there could be multicollinearity.

For this we will to use the correlation matrix from our regression equation.

```
diamonds_matrix <- model.matrix(diamonds_model)[,-1]
diamonds_corr <- cor(diamonds_matrix)
eigenvalue <- eigen(diamonds_corr)$values
coefficient_names <- colnames(diamonds_matrix)
condition_index <- sqrt(max(eigenvalue)/eigenvalue)
CI_df <- data.frame(Coefficiant=coefficient_names, CI=condition_index)
kable(CI_df)
```

Coefficiant	CI
depth	1.000000
cutGood	1.196043
cutVery Good	1.269965
cutPremium	1.286559
cutIdeal	1.323254
carat	1.335179
colorE	1.353269
colorF	1.422821
colorG	1.496735
colorH	1.693845
colorI	2.175808
colorJ	4.151526
table	7.795474

We can see here that there are no values over 30, so we will explore other options for testing our model.

Stepwise Elemination

We will now use stepwise elimination to identify the most relevant predictors for the model.

These methods help to avoid overfitting leaving out predictors that are not significantly increasing the accuracy of the model.

We will check both forward and backward stepwise elimination to see if we get the same result. We will use a k value of 2, because this is the standard k value for minimizing prediction error in models.

```
#backwards stepwise
diamonds_stepB <- stepAIC(diamonds_model, direction="backward", k = 2, trace = 0)
summary(diamonds_stepB)
```

```
##
## Call:
## lm(formula = price ~ cut + carat + color, data = sample_diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10706.3   -677.3    -68.6    467.0   9056.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   -3251.7      429.8  -7.566 1.94e-13 ***
## cutGood         981.9      427.2   2.299  0.02194 *
## cutVery Good   1593.9      401.5   3.970 8.27e-05 ***
## cutPremium     1301.4      402.2   3.235  0.00130 **
## cutIdeal       1611.6      395.6   4.074 5.40e-05 ***
## carat          7568.4      152.8  49.521  < 2e-16 ***
## colorE         -103.9      258.7  -0.402  0.68805
## colorF         -199.4      262.3  -0.760  0.44766
## colorG         -119.3      242.6  -0.492  0.62310
## colorH         -807.0      264.1  -3.055  0.00237 **
## colorI         -954.8      289.9  -3.294  0.00106 **
## colorJ        -1508.5      353.4  -4.269 2.36e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1453 on 488 degrees of freedom
## Multiple R-squared:  0.8406, Adjusted R-squared:  0.837
## F-statistic:   234 on 11 and 488 DF,  p-value: < 2.2e-16
```

Backwards stepwise elimination yields a final model includes only the variables cut, carat, and color.

This model appears to be slightly better than our original diamonds_model because the Adjusted R-squared value is slightly higher, with a value of 0.837 rather than 0.8367. This is a minor improvement, but an improvement nonetheless.

We will now confirm this adjusted model with forward stepwise elimination to see if we get the same result.

```
#forward stepwise
diamonds_stepF <- stepAIC(diamonds_model, direction="forward", k = 2, trace = 0)
summary(diamonds_stepF)
```

```
##
## Call:
## lm(formula = price ~ depth + cut + carat + color + table, data = sample_diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10582.0   -654.7    -50.7    457.3   9114.2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2905.72    4758.01  -0.611 0.541683
## depth          -28.85      52.78  -0.547 0.584896
## cutGood        920.52     442.43   2.081 0.037993 *
## cutVery Good  1549.96     426.94   3.630 0.000313 ***
## cutPremium    1229.85     431.90   2.848 0.004593 **
## cutIdeal      1620.29     440.68   3.677 0.000263 ***
## carat         7557.20     153.80  49.136  < 2e-16 ***
## colorE         -87.95     259.33  -0.339 0.734635
## colorF        -196.31     262.70  -0.747 0.455257
## colorG        -104.83     243.15  -0.431 0.666571
## colorH        -796.27     264.72  -3.008 0.002767 **
## colorI        -946.71     290.32  -3.261 0.001189 **
## colorJ       -1497.65     353.88  -4.232 2.77e-05 ***
```

```
## table           25.47      38.96   0.654 0.513534
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1454 on 486 degrees of freedom
## Multiple R-squared:  0.841,  Adjusted R-squared:  0.8367
## F-statistic: 197.7 on 13 and 486 DF,  p-value: < 2.2e-16
```

This yields a different result than backwards stepwise elimination. We will now perform bidirectional stepwise elimination.

```r
#bidirectional stepwise
diamonds_step <- stepAIC(diamonds_model, direction="both", k = 2, trace = 0)
summary(diamonds_step)
```

```
##
## Call:
## lm(formula = price ~ cut + carat + color, data = sample_diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10706.3   -677.3    -68.6    467.0   9056.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3251.7      429.8  -7.566 1.94e-13 ***
## cutGood         981.9      427.2   2.299  0.02194 *
## cutVery Good   1593.9      401.5   3.970 8.27e-05 ***
## cutPremium     1301.4      402.2   3.235  0.00130 **
## cutIdeal       1611.6      395.6   4.074 5.40e-05 ***
## carat          7568.4      152.8  49.521  < 2e-16 ***
## colorE         -103.9      258.7  -0.402  0.68805
## colorF         -199.4      262.3  -0.760  0.44766
## colorG         -119.3      242.6  -0.492  0.62310
## colorH         -807.0      264.1  -3.055  0.00237 **
## colorI         -954.8      289.9  -3.294  0.00106 **
## colorJ        -1508.5      353.4  -4.269 2.36e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1453 on 488 degrees of freedom
## Multiple R-squared:  0.8406, Adjusted R-squared:  0.837
## F-statistic:   234 on 11 and 488 DF,  p-value: < 2.2e-16
```

Performing bidirectional stepwise elimination returns the same result as backwards stepwise elimination. This suggests that the best model includes only predictors cut, carat, and color. Forward stepwise elimination returns the best model to include only predictors cut, carat, color, and table.

The inclusion of the table predictor in this model could be due to forward stepwises inability to remove predictors after adding them. If when table was added to the model, it improved the fit then after other predictors are added it weakened the fit, it cannot remove table.

Backward and bidirectional stepwise elimination are more computationally taxing then forward stepwise elimination, but they generally return better results.

Cross Validation

We can check if our new model results in less Mean Squared Error using Cross Validation.

We will do so by creating a CV function, then applying it to our two models to check each models MSE.

```
#CV function
kFoldCV <- function(data, response, formula, k, seed = FALSE, seed_num, shuffle = FALSE) {
  if (seed == TRUE) {
    set.seed(seed_num)
  }
  if(shuffle == TRUE) {
    shuffle_index = sample(1:nrow(data), replace = FALSE)
    data = data[shuffle_index,]
    response = response[shuffle_index]
  }
  folds <- cut(seq(1,nrow(data)),breaks=k,labels=FALSE)
  mse = numeric()
  for(i in 1:k){
    testIndexes <- which(folds == i, arr.ind=TRUE)
    testData <- data[testIndexes, ]
    trainData <- data[-testIndexes, ]

    diamonds_model.train = lm(formula, data = trainData)
    mse[i] = (1/length(testIndexes))*sum((response[testIndexes]
                                          - predict(diamonds_model.train, newdata = testData))^2)
  }
  rmse = sqrt(mse)
  cv_k_mse = sum(mse)/k
  cv_k_rmse = sum(rmse)/k
  return(list(CV_MSE = cv_k_mse, CV_RMSE = cv_k_rmse))
}
```

```
#CV on original model
kFoldCV(response = diamonds$price, formula =price ~ depth + cut + carat + color + table,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 1)
```

```
## $CV_MSE
## [1] 2045164
##
## $CV_RMSE
## [1] 1429.692
```

```
#CV on new model
kFoldCV(response = diamonds$price, formula = price ~ cut + carat + color, data = diamonds,
        k = 10, shuffle = TRUE, seed = TRUE, seed_num = 1)
```

```
## $CV_MSE
## [1] 2051609
##
## $CV_RMSE
## [1] 1431.963
```

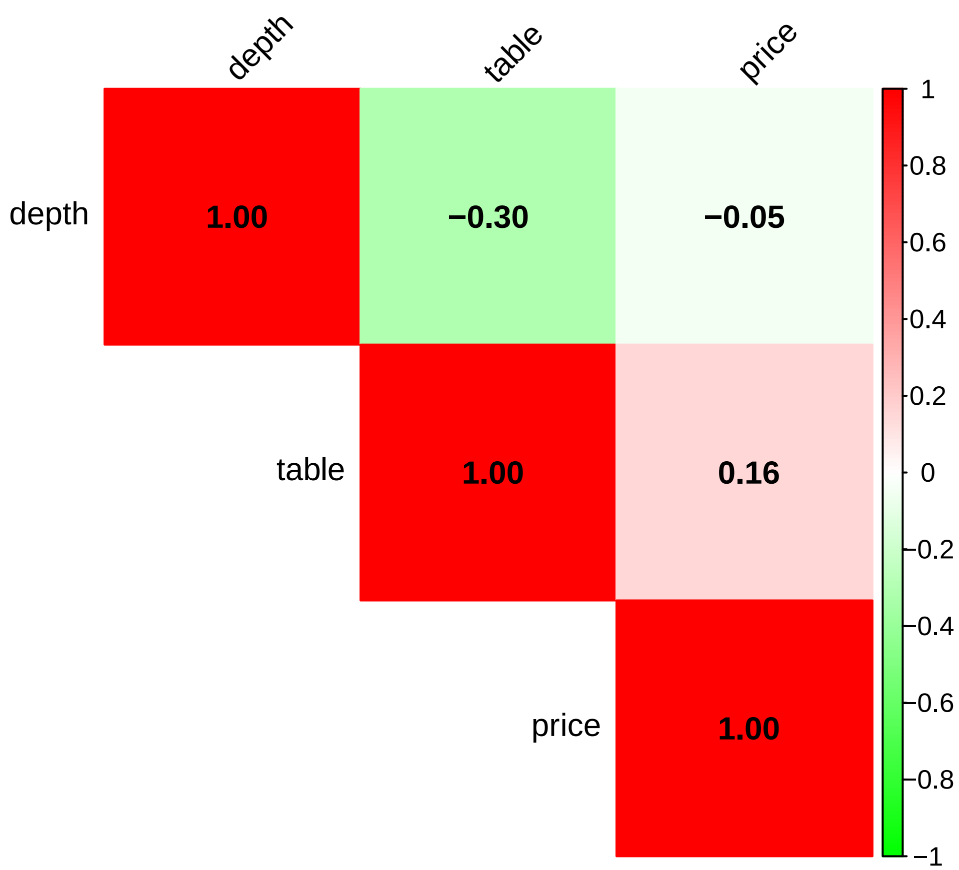
Cross validation is giving us an interesting result. We can see that MSE for our simplified model is higher than the MSE for our original model.

The results of cross validation pose need for a look into the correlation between depth and price and table and price.

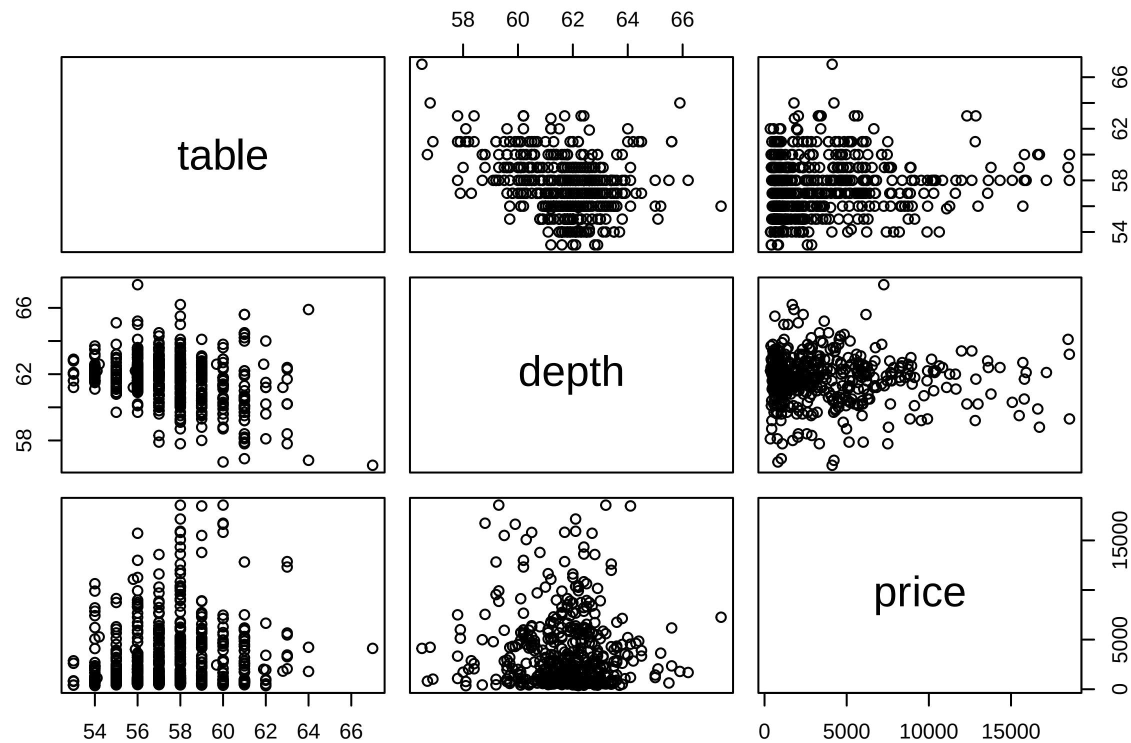
For this we can use a corr plot.

```
#correlation heatmap
correlation_diamonds <- cor(sample_diamonds %>% dplyr::select_if(is.numeric) %>%
                              dplyr::select(-X, -x, -y, -z, -carat))

corrplot(correlation_diamonds, method="color",
         col=colorRampPalette(c("green", "white", "red"))(200),
         type="upper", tl.col="black", tl.srt=45, addCoef.col = "black")
```




```
#pairs plot
pairs(sample_diamonds[, c('table', 'depth', 'price')])
```



It appears that neither table, depth, or price are significantly correlated. It seems strange that MSE increases when they are removed from the model.

It is possible the seed used is giving an unlucky result. Since the MSE increase is strange, let's explore other seeds.

```
kFoldCV(response = diamonds$price, formula =price ~ depth + cut + carat + color + table,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 43)
```

```
## $CV_MSE
## [1] 2045041
##
## $CV_RMSE
## [1] 1430.011
```

```
kFoldCV(response = diamonds$price, formula =price ~ cut + carat + color,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 43)
```

```
## $CV_MSE
## [1] 2051500
##
## $CV_RMSE
## [1] 1432.27
```

```r
kFoldCV(response = diamonds$price, formula =price ~ depth + cut + carat + color + table,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 713)
```

```
## $CV_MSE
## [1] 2045273
##
## $CV_RMSE
## [1] 1429.948
```

```r
kFoldCV(response = diamonds$price, formula =price ~ cut + carat + color,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 713)
```

```
## $CV_MSE
## [1] 2051743
##
## $CV_RMSE
## [1] 1432.203
```

```r
kFoldCV(response = diamonds$price, formula =price ~ depth + cut + carat + color + table,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 123)
```

```
## $CV_MSE
## [1] 2045532
##
## $CV_RMSE
## [1] 1429.976
```

```r
kFoldCV(response = diamonds$price, formula =price ~ cut + carat + color,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 123)
```

```
## $CV_MSE
## [1] 2051922
##
## $CV_RMSE
## [1] 1432.201
```

```r
kFoldCV(response = diamonds$price, formula =price ~ depth + cut + carat + color + table,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 600493)
```

```
## $CV_MSE
## [1] 2045483
##
## $CV_RMSE
## [1] 1429.728
```

```r
kFoldCV(response = diamonds$price, formula =price ~ cut + carat + color,
        data = diamonds, k = 10, shuffle = TRUE, seed = TRUE, seed_num = 600493)
```

```
## $CV_MSE
## [1] 2051736
##
## $CV_RMSE
## [1] 1431.936
```

Conclusion

After trying multiple seeds, the result is showing the same. It is still possible this is a sampling error, since there is not significant correlation between price, table, and depth. For now we will assume there is a sampling error and conclude that a model with only the predictors cut, carat, and color is better due to a larger adjusted R^2 value.