Diamonds

Francisco Arrieta, Emily Schmidt and Lucia Camenisch

2022-12-17

Contents


```r
################ General Use #####################
library(car)            #for statistic functions
library(DataExplorer)   #for graphing missing value percentages
library(data.table)     #for reading data.tables
library(dplyr)          #for data manipulation
library(fastDummies)    #for creating dummies
library(e1071)          #for skewness
library(ellipse)        #for mapping correlation
library(GGally)         #for making graphs
library(ggplot2)        #for making graphs
library(ggpubr)         #for plot alignment
library(gridExtra)
library(kableExtra)     #for more elaborate tables
library(knitr)
library(tidyr)          #for changing the shape and hierarchy of a data set
library(naniar)         #for missing values
library(RColorBrewer)   #for graph colors
library(rattle)         #Graphical Data Interface


################ For Predictions #####################
library(caret)          #for preProcess() and accuracy()
library(forecast)       # for accuracy() measures
library(FNN)            #for finding k nearest neighbor
library(gbm)            #for boosting
```

```r
library(ipred)          #for bagging
library(vip)            #for variable importance
library(randomForest)   #for randomForest
library(rpart)          #for regression trees
library(rpart.plot)     #for plot trees
library(keras)          #front-end library for neural networks
library(magrittr)
library(tensorflow)     #backend python library for neural network


################ Personalized Functions #####################
source("VIF.R")         #for calculating VIF (KEEP/DELETE???????????????)
source("ProcStep.R")    #for variable selection (forw, backw, setpw)
source("GlobalCrit.R")  #for variable selection (exhaustive search)

################ Additonal Actions #####################
options(scipen = 999)              #for removing scientific notation
tf$constant("Hello Tensorflow!")   #for initializing tensoflow environment
```

```
## tf.Tensor(b'Hello Tensorflow!', shape=(), dtype=string)
```

Data Exploration

```r
diamonds <- fread("diamonds.csv", sep=",", header = T) # Load your data, diamonds.csv

diamonds$V1 <- NULL # Remove column 'V1' as it is similar to an ID variable - no additional meaning der

# Rename columns for more precise names
colnames(diamonds)[5] <- "depth_ratio" # depth to depth_ratio
colnames(diamonds)[8] <- "length" # x to length
colnames(diamonds)[9] <- "width"  # y to width
colnames(diamonds)[10] <- "depth" # z to depth
```

Dimension Summary

```r
dim(diamonds) # Dimensions of data
```

```
## [1] 53940    10
```

```r
summary(diamonds) # Produce result summaries of all variables
```

```
##      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_ratio        table           price           length
##  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
##      width            depth
##  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
```

```
str(diamonds) # Type of variables
```

```
## Classes 'data.table' and 'data.frame':   53940 obs. of  10 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" ...
##  $ color      : chr  "E" "E" "E" "I" ...
##  $ clarity    : chr  "SI2" "SI1" "VS1" "VS2" ...
##  $ depth_ratio: 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 ...
##  $ length     : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ width      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ depth      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# Number of unique values in each variable
sapply(diamonds, function(x) length(unique(x)))
```

```
##       carat         cut       color     clarity depth_ratio       table
##         273           5           7           8         184         127
##       price      length       width       depth
##       11602         554         552         375
```

Missing Values

```
# Missing values analysis
gg_miss_var(diamonds) + ggtitle("Missing values")
```

```
############### carat no problems ################

############### cut ###############################
unique(diamonds$cut) # Review unique values for cut
```

```
## [1] "Ideal"     "Premium"   "Good"      "Very Good" "Fair"
```

```
# Factor the cut to five level
diamonds$cut <- as.factor(diamonds$cut)

# Ordered from worst to best
diamonds$cut <- ordered(diamonds$cut, levels = c("Fair", "Good", "Very Good", "Premium", "Ideal"))

############### color ###########################
# Review unique values for color
unique(diamonds$color)
```

```
## [1] "E" "I" "J" "H" "F" "G" "D"
```

```
# Factor the color to seven levels
diamonds$color <- as.factor(diamonds$color)

# Ordered from worst to best
diamonds$color <- ordered(diamonds$color, levels = c("J", "I", "H", "G", "F", "E", "D"))

############### clarity ##########################
# Review unique values for clarity
unique(diamonds$clarity)
```

```
## [1] "SI2"  "SI1"  "VS1"  "VS2"  "VVS2" "VVS1" "I1"   "IF"
```

```
# Factor the clarity to eight levels
diamonds$clarity <- as.factor(diamonds$clarity)

# Ordered from worst to best
diamonds$clarity <- ordered(diamonds$clarity, levels = c("I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS

############### table no problems################

############### price no problems################

############### Other Checks #####################

# Remove values of 0 for for dimensions which includes zeros in length and width
nrow(diamonds[depth %in% 0,]) # Remove 20 rows due to depth = 0.0
```

```
## [1] 20
```

```
diamonds <- diamonds[depth > 0, ] # Include only values with depth greater than zero

# Create formula to check the absolute value of length to width, comparison
diamonds[, subtraction := abs(length - width)]
nrow(diamonds[subtraction>10,]) # Remove 2 rows due their extreme subtraction value (~59 and ~26)
```

```
## [1] 2
```

```
diamonds <- diamonds[subtraction <= 10, ] # Include only values with subtraction less than ten

# Check if the Depth_Ratio value corresponds to formula indicated in the description
diamonds[, depth_check := round(100*(2*depth)/((length + width)), 1)]
diamonds[, diff := abs(depth_check-depth_ratio)]

# Create histogram to look at the differences between how much price is off between a calculated value
hist(diamonds[diff >= 0.2 & diff < 1, diff], breaks =50, col = "#D8B365", border = "#D8B365", main = "T
```

```
# Threshold set to 0.3 due to ... (report)
nrow(diamonds[diff > 0.3,]) # We remove 268 rows
```

```
## [1] 253
```

```
diamonds <- diamonds[diff <= 0.3,]
```

```r
# Removed created columns needed to clean the data
diamonds[, subtraction := NULL]
diamonds[, depth_check := NULL]
diamonds[, diff := NULL]
# Total rows remove: 275 observations
```

```r
# Reorder data table to group like variable types
diamonds <- diamonds[, c(7, 2:4, 1, 8:10, 5:6)]
```

```r
#Used ggpairs to create a scatterplot matrix between quantitative variables
ggpairs(diamonds[, c(1, 5:10)], title = "Scatterplot Matrix",
         proportions = "auto",
         columnLabels = c("Price", "Carat", "Length", "Width", "Depth","Depth Ratio","Table"),
         upper = list(continuous = wrap('cor',size = 3)),) + theme_light()
```

```r
# Create plot that looks at carat and price
PCl1 <- ggplot(aes(x = carat, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Clarity', reverse = T,override.aes = li

# Create plot that looks at length and price
PCl2 <- ggplot(aes(x = length, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, positio
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Clarity', reverse = T,override.aes = li

# Create plot that looks at width and price
PCl3 <- ggplot(aes(x = width, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Clarity', reverse = T,override.aes = li

# Create plot that looks at depth and price
PCl4 <- ggplot(aes(x = depth, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Clarity', reverse = T,override.aes = li

# Create plot that looks at depth_ratio and price
PCl5 <- ggplot(aes(x = depth_ratio, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, po

# Create plot that looks at table and price
PCl6 <- ggplot(aes(x = table, y = price), data = diamonds) +  geom_point(alpha = 0.5, size = 1, positio

# Arrange ggplots into one frame
ggarrange(PCl1, PCl2, PCl3,PCl4, PCl5, PCl6,
                    ncol = 2, nrow = 3)
```

```r
# Create plot that looks at carat and price
PCo1 <- ggplot(aes(x = carat, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Color', reverse = T,override.aes = list

# Create plot that looks at length and price
PCo2 <- ggplot(aes(x = length, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, positio
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Color', reverse = T,override.aes = list

# Create plot that looks at width and price
PCo3 <- ggplot(aes(x = width, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Color', reverse = T,override.aes = list
```

```r
# Create plot that looks at depth and price
PCo4 <- ggplot(aes(x = depth, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Color', reverse = T,override.aes = list

# Create plot that looks at depth_ratio and price
PCo5 <- ggplot(aes(x = depth_ratio, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, pos

# Create plot that looks at table and price
PCo6 <- ggplot(aes(x = table, y = price), data = diamonds) +  geom_point(alpha = 0.5, size = 1, position

# Arrange ggplots into one frame
ggarrange(PCo1, PCo2, PCo3,PCo4, PCo5, PCo6,
                    ncol = 2, nrow = 3)
```

```r
# Create plot that looks at carat and price
PCu1 <- ggplot(aes(x = carat, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Cut', reverse = T,override.aes = list(a

# Create plot that looks at length and price
PCu2 <- ggplot(aes(x = length, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Cut', reverse = T,override.aes = list(a

# Create plot that looks at width and price
PCu3 <- ggplot(aes(x = width, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Cut', reverse = T,override.aes = list(a

# Create plot that looks at depth and price
PCu4 <- ggplot(aes(x = depth, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, position
  scale_color_brewer(type = 'div', guide = guide_legend(title = 'Cut', reverse = T,override.aes = list(a

# Create plot that looks at depth_ratio and price
PCu5 <- ggplot(aes(x = depth_ratio, y = price), data = diamonds) + geom_point(alpha = 0.5, size = 1, pos

# Create plot that looks at table and price
PCu6 <- ggplot(aes(x = table, y = price), data = diamonds) +  geom_point(alpha = 0.5, size = 1, position

# Arrange ggplots into one frame
ggarrange(PCu1, PCu2, PCu3,PCu4, PCu5, PCu6,
                    ncol = 2, nrow = 3)
```

```r
# Correlation between price and quantitative variables
price_correlation <- with(diamonds,
     data.frame(cor_length_price = cor(length, price), cor_width_price = cor(width, price), cor_depth_pr

# Transpose data and put into kable format
transpose <- t(sort(round(price_correlation,4),decreasing = FALSE))
kable_corr <- kable(transpose) %>% kable_classic()
kable_corr
```

cor_depth_ratio_price	-0.0104
cor_table_price2	0.1273
cor_depth_price	0.8829
cor_length_price	0.8876
cor_width_price	0.8892
cor_carat_price3	0.9218

Variable Visualisation

```
ggplot(gather(data = diamonds[, c(1, 5:10)]), aes(value)) +
  geom_histogram(aes(y = after_stat(density)),
                 bins = 10,
                 color = "white",
                 fill = "#D8B365") + # Creates bin sizing and sets the lines as white
  geom_density(alpha = .2, fill = "#D8B365") +
  facet_wrap(~ key, scales = "free") + # Converting the graphs into panels
  ggtitle("Quantitative Variable Analysis") + # Title name
  ylab("Count") + xlab("Value") + # Label names
  theme_classic() # A classic theme, with x and y axis lines and no grid lines
```

```
# Create heatmap to show variable correlation
# Round the correlation coefficient to two decimal places
cormat <- round(cor(diamonds[, c(1, 5:10)]), 2)

# Use correlation between variables as distance
reorder_cormat <- function(cormat){
dd <- as.dist((1-cormat)/2)
hc <- hclust(dd)
cormat <-cormat[hc$order, hc$order]
return(cormat)
}

# Reorder the correlation matrix
cormat <- reorder_cormat(cormat)

# Keeping only upper triangular matrix
# upper_tri returns TRUE/FALSE for each coordinate (TRUE -> part of upper triangle)
# multiplying will thus keep the upper triangle values and set the others to 0
cormat <- cormat*upper.tri(cormat, diag = TRUE)
# Values of the lower triangle (0) are replaced by NA
cormat[cormat == 0] <- NA

# Melt the correlation matrix
cormat <- reshape2::melt(cormat, na.rm = TRUE)

# Create a ggheatmap with multiple characteristics
ggplot(cormat, aes(Var2, Var1, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "#D8B365", high = "#15DDD8", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab", name="Pearson\nCorrelation") +
  ggtitle("Correlation Heatmap") + # Title name
  theme_minimal() + # Minimal theme, keeps in the lines
```

```
  theme(axis.text.x = element_text(angle = 45, vjust = 1, size = 12, hjust = 1)) +
  coord_fixed() +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 2)
```

```
rm(cormat, reorder_cormat)
```

```
#plot Correlation ellipses
plotcorr(cor(diamonds[, -c(2:4)]), col = "#D8B365",
         main = "Pearson correlation ellipses for numerical variables")
```

```
# set seed for reproducing the partition
set.seed(111)

# generating training set index
train.index <- sample(c(1:nrow(diamonds)), 0.5*nrow(diamonds))

# generating validation set index taken from the complementary of training set
valid.index <- sample(setdiff(c(1:nrow(diamonds)), train.index), 0.3*nrow(diamonds))

# defining test set index as complementary of (train.index + valid.index)
test.index <- as.numeric(setdiff(row.names(diamonds), union(train.index, valid.index)))

# creating data tables Train, Valid and Test using the indexes
Train <- diamonds[train.index, ]
Valid <- diamonds[valid.index, ]
Test <- diamonds[test.index, ]
```

Variable Prediction and Model Performance Evaluation

Linear Regression

```
Train_lr <- diamonds[train.index, ]
Valid_lr <- diamonds[valid.index, ]
Test_lr  <- diamonds[ test.index, ]
```

```
VIF(y = diamonds$price, matx = diamonds[, -c(1)])
```

```
##
##                   GVIF Df GVIF^(1/(2*Df))
## cut            2.45522  4         1.11882
## color          1.18398  6         1.01417
## clarity        1.36848  7         1.02266
## carat         25.91780  1         5.09095
## length      1091.42000  1        33.03670
## width       1143.44000  1        33.81480
## depth       2008.33000  1        44.81440
## depth_ratio   31.99350  1         5.65628
## table          1.80396  1         1.34311
##
##   Mean: 165.105
```

```r
VIF(y = diamonds$price, matx = diamonds[, -c(1, 6, 7, 8)])
```

```
##
##                 GVIF Df GVIF^(1/(2*Df))
## cut         1.93382  4         1.08593
## color       1.17045  6         1.01320
## clarity     1.30388  7         1.01913
## carat       1.32381  1         1.15057
## depth_ratio 1.38914  1         1.17862
## table       1.79505  1         1.33980
##
##   Mean: 1.98352
```

```r
plotcorr(cor(diamonds[, -c(2:4, 6:8)]), col = "#D8B365",
         main = "Pearson correlation ellipses for numerical variables")
```

```r
sapply(Train_lr[, c(1, 5:10)], skewness)
```

```
##       price       carat      length       width       depth depth_ratio
##  1.62597473  1.09675143  0.39846889  0.39220078  0.39210931  0.01824979
##       table
##  0.87288692
```

```r
Train_lr$price <- log(Train_lr$price)
Train_lr$carat <- log(Train_lr$carat)
Train_lr$table <- log(Train_lr$table)
sapply(Train_lr[, c(1, 5:10)], skewness)
```

```
##       price       carat      length       width       depth depth_ratio
##  0.11305083  0.09668960  0.39846889  0.39220078  0.39210931  0.01824979
##       table
##  0.64541910
```

```r
Valid_lr$price <- log(Valid_lr$price)
Valid_lr$carat <- log(Valid_lr$carat)
Valid_lr$table <- log(Valid_lr$table)
```

```r
ggplot(gather(data = Train_lr[, c(1, 5:10)]), aes(value)) +
  geom_histogram(aes(y = after_stat(density)),
                 color = "white",
                 fill = "#D8B365") +             # Creates bin sizing with colors
  geom_density(alpha = .2, fill = "#D8B365") +
  facet_wrap(~ key, scales = "free") +           # Converting the graphs into panels
  ggtitle("Histograms of numerical variables") + # Title name
  ylab("Count") + xlab("Value") +                # Label names
  theme_classic()                                # Theme with x and y axis lines and no grid lines
```

```r
norm.values <- preProcess(Train_lr[, c(1, 5:10)], method=c("center", "scale"))

Train_lr[, c(1, 5:10)] <- predict(norm.values, Train_lr[, c(1, 5:10)])
Valid_lr[, c(1, 5:10)] <- predict(norm.values, Valid_lr[, c(1, 5:10)])
 Test_lr[, c(1, 5:10)] <- predict(norm.values,  Test_lr[, c(1, 5:10)])
```

```r
LM_complete = lm(price ~. , data = Train_lr)
summary(LM_complete)
```

```
##
## Call:
## lm(formula = price ~ ., data = Train_lr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04059 -0.08291  0.00028  0.08156  1.42487
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0747373  0.0016289 -45.882 < 0.0000000000000002 ***
## cut.L        0.1168565  0.0037189  31.422 < 0.0000000000000002 ***
## cut.Q       -0.0342102  0.0030248 -11.310 < 0.0000000000000002 ***
## cut.C        0.0174498  0.0027023   6.457      0.0000000001084 ***
## cut^4        0.0003697  0.0020867   0.177              0.85937
## color.L      0.4366728  0.0028463 153.417 < 0.0000000000000002 ***
## color.Q     -0.0974458  0.0025856 -37.687 < 0.0000000000000002 ***
## color.C      0.0126123  0.0024180   5.216      0.0000001841568 ***
## color^4      0.0134193  0.0022210   6.042      0.0000000015434 ***
## color^5      0.0001336  0.0021044   0.063              0.94940
## color^6      0.0026592  0.0019108   1.392              0.16404
## clarity.L    0.9080235  0.0050105 181.224 < 0.0000000000000002 ***
## clarity.Q   -0.2486235  0.0046594 -53.360 < 0.0000000000000002 ***
## clarity.C    0.1370069  0.0039810  34.415 < 0.0000000000000002 ***
## clarity^4   -0.0685371  0.0031777 -21.568 < 0.0000000000000002 ***
## clarity^5    0.0290689  0.0025896  11.225 < 0.0000000000000002 ***
## clarity^6   -0.0022942  0.0022444  -1.022              0.30670
## clarity^7    0.0312779  0.0019833  15.771 < 0.0000000000000002 ***
## carat        0.9997686  0.0082204 121.621 < 0.0000000000000002 ***
## length       0.1775342  0.0266043   6.673      0.0000000000255 ***
## width       -0.0200231  0.0270778  -0.739              0.45963
## depth       -0.0702733  0.0359774  -1.953              0.05080 .
## depth_ratio  0.0117907  0.0045226   2.607              0.00914 **
## table        0.0011885  0.0010849   1.096              0.27330
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.131 on 26808 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9828
## F-statistic: 6.685e+04 on 23 and 26808 DF,  p-value: < 0.00000000000000022
```

```r
LM_forward_complete = step(LM_complete, direction = "forward")
```

```
## Start:  AIC=-109064.2
## price ~ cut + color + clarity + carat + length + width + depth +
##     depth_ratio + table
```

```r
summary(LM_forward_complete)    # no selection
```

```
##
## Call:
## lm(formula = price ~ cut + color + clarity + carat + length +
```

```
##     width + depth + depth_ratio + table, data = Train_lr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04059 -0.08291  0.00028  0.08156  1.42487
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0747373  0.0016289 -45.882 < 0.0000000000000002 ***
## cut.L        0.1168565  0.0037189  31.422 < 0.0000000000000002 ***
## cut.Q       -0.0342102  0.0030248 -11.310 < 0.0000000000000002 ***
## cut.C        0.0174498  0.0027023   6.457      0.0000000001084 ***
## cut^4        0.0003697  0.0020867   0.177              0.85937
## color.L      0.4366728  0.0028463 153.417 < 0.0000000000000002 ***
## color.Q     -0.0974458  0.0025856 -37.687 < 0.0000000000000002 ***
## color.C      0.0126123  0.0024180   5.216      0.0000001841568 ***
## color^4      0.0134193  0.0022210   6.042      0.0000000015434 ***
## color^5      0.0001336  0.0021044   0.063              0.94940
## color^6      0.0026592  0.0019108   1.392              0.16404
## clarity.L    0.9080235  0.0050105 181.224 < 0.0000000000000002 ***
## clarity.Q   -0.2486235  0.0046594 -53.360 < 0.0000000000000002 ***
## clarity.C    0.1370069  0.0039810  34.415 < 0.0000000000000002 ***
## clarity^4   -0.0685371  0.0031777 -21.568 < 0.0000000000000002 ***
## clarity^5    0.0290689  0.0025896  11.225 < 0.0000000000000002 ***
## clarity^6   -0.0022942  0.0022444  -1.022              0.30670
## clarity^7    0.0312779  0.0019833  15.771 < 0.0000000000000002 ***
## carat        0.9997686  0.0082204 121.621 < 0.0000000000000002 ***
## length       0.1775342  0.0266043   6.673      0.0000000000255 ***
## width       -0.0200231  0.0270778  -0.739              0.45963
## depth       -0.0702733  0.0359774  -1.953              0.05080 .
## depth_ratio  0.0117907  0.0045226   2.607              0.00914 **
## table        0.0011885  0.0010849   1.096              0.27330
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.131 on 26808 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9828
## F-statistic: 6.685e+04 on 23 and 26808 DF,  p-value: < 0.00000000000000022
```

```r
LM_backward_complete = step(LM_complete, direction = "backward")
```

```
## Start:  AIC=-109064.2
## price ~ cut + color + clarity + carat + length + width + depth +
##     depth_ratio + table
##
##               Df Sum of Sq    RSS     AIC
## - width        1      0.01 459.84 -109066
## - table        1      0.02 459.85 -109065
## <none>                     459.83 -109064
## - depth        1      0.07 459.89 -109062
## - depth_ratio  1      0.12 459.95 -109059
## - length       1      0.76 460.59 -109022
## - cut          4     19.38 479.21 -107965
## - carat        1    253.72 713.54  -97277
## - color        6    428.53 888.36  -91407
```

```
## - clarity      7    874.13 1333.96  -80501
##
## Step:  AIC=-109065.7
## price ~ cut + color + clarity + carat + length + depth + depth_ratio +
##     table
##
##                Df Sum of Sq     RSS     AIC
## - table        1      0.02  459.86 -109066
## <none>                      459.84 -109066
## - depth        1      0.18  460.02 -109057
## - depth_ratio  1      0.30  460.14 -109050
## - length       1      0.75  460.59 -109024
## - cut          4     19.38  479.22 -107966
## - carat        1    256.52  716.35  -97173
## - color        6    428.96  888.80  -91395
## - clarity      7    877.59 1337.43  -80433
##
## Step:  AIC=-109066.4
## price ~ cut + color + clarity + carat + length + depth + depth_ratio
##
##                Df Sum of Sq     RSS     AIC
## <none>                      459.86 -109066
## - depth        1      0.18  460.04 -109058
## - depth_ratio  1      0.29  460.15 -109052
## - length       1      0.75  460.61 -109025
## - cut          4     24.58  484.44 -107677
## - carat        1    261.25  721.11  -96998
## - color        6    429.14  889.00  -91391
## - clarity      7    877.84 1337.70  -80430
```

```
summary(LM_backward_complete)   # like LM_CpAIC_complete
```

```
##
## Call:
## lm(formula = price ~ cut + color + clarity + carat + length +
##     depth + depth_ratio, data = Train_lr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04014 -0.08295  0.00025  0.08161  1.42593
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0741493  0.0015667 -47.329 < 0.0000000000000002 ***
## cut.L        0.1150923  0.0034602  33.262 < 0.0000000000000002 ***
## cut.Q       -0.0339481  0.0029582 -11.476 < 0.0000000000000002 ***
## cut.C        0.0163231  0.0025452   6.413      0.0000000001447 ***
## cut^4       -0.0001739  0.0020487  -0.085              0.93235
## color.L      0.4366337  0.0028441 153.523 < 0.0000000000000002 ***
## color.Q     -0.0974187  0.0025833 -37.711 < 0.0000000000000002 ***
## color.C      0.0125915  0.0024179   5.208      0.0000001927323 ***
## color^4      0.0133674  0.0022204   6.020      0.0000000017651 ***
## color^5      0.0001194  0.0021042   0.057              0.95475
## color^6      0.0026739  0.0019108   1.399              0.16171
## clarity.L    0.9075881  0.0049934 181.757 < 0.0000000000000002 ***
```

```
## clarity.Q   -0.2483814  0.0046523 -53.389 < 0.0000000000000002 ***
## clarity.C    0.1367692  0.0039749  34.408 < 0.0000000000000002 ***
## clarity^4   -0.0684754  0.0031760 -21.560 < 0.0000000000000002 ***
## clarity^5    0.0289678  0.0025881  11.193 < 0.0000000000000002 ***
## clarity^6   -0.0022952  0.0022444  -1.023              0.30649
## clarity^7    0.0313111  0.0019831  15.789 < 0.0000000000000002 ***
## carat        1.0002821  0.0081052 123.413 < 0.0000000000000002 ***
## length       0.1751101  0.0264602   6.618      0.0000000000371 ***
## depth       -0.0884193  0.0271461  -3.257              0.00113 **
## depth_ratio  0.0136166  0.0033272   4.093      0.0000427910418 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.131 on 26810 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9828
## F-statistic: 7.321e+04 on 21 and 26810 DF,  p-value: < 0.00000000000000022
```

```
LM_stepwise_complete = step(LM_complete, direction = "both")
```

```
## Start:  AIC=-109064.2
## price ~ cut + color + clarity + carat + length + width + depth +
##     depth_ratio + table
##
##               Df Sum of Sq     RSS     AIC
## - width        1      0.01  459.84 -109066
## - table        1      0.02  459.85 -109065
## <none>                      459.83 -109064
## - depth        1      0.07  459.89 -109062
## - depth_ratio  1      0.12  459.95 -109059
## - length       1      0.76  460.59 -109022
## - cut          4     19.38  479.21 -107965
## - carat        1    253.72  713.54  -97277
## - color        6    428.53  888.36  -91407
## - clarity      7    874.13 1333.96  -80501
##
## Step:  AIC=-109065.7
## price ~ cut + color + clarity + carat + length + depth + depth_ratio +
##     table
##
##               Df Sum of Sq     RSS     AIC
## - table        1      0.02  459.86 -109066
## <none>                      459.84 -109066
## + width        1      0.01  459.83 -109064
## - depth        1      0.18  460.02 -109057
## - depth_ratio  1      0.30  460.14 -109050
## - length       1      0.75  460.59 -109024
## - cut          4     19.38  479.22 -107966
## - carat        1    256.52  716.35  -97173
## - color        6    428.96  888.80  -91395
## - clarity      7    877.59 1337.43  -80433
##
## Step:  AIC=-109066.4
## price ~ cut + color + clarity + carat + length + depth + depth_ratio
##
##               Df Sum of Sq     RSS     AIC
```

```
## <none>                         459.86 -109066
## + table         1      0.02  459.84 -109066
## + width         1      0.01  459.85 -109065
## - depth         1      0.18  460.04 -109058
## - depth_ratio   1      0.29  460.15 -109052
## - length        1      0.75  460.61 -109025
## - cut           4     24.58  484.44 -107677
## - carat         1    261.25  721.11  -96998
## - color         6    429.14  889.00  -91391
## - clarity       7    877.84 1337.70  -80430
```

```r
summary(LM_stepwise_complete)   # like LM_CpAIC_complete
```

```
##
## Call:
## lm(formula = price ~ cut + color + clarity + carat + length +
##     depth + depth_ratio, data = Train_lr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04014 -0.08295  0.00025  0.08161  1.42593
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0741493  0.0015667 -47.329 < 0.0000000000000002 ***
## cut.L        0.1150923  0.0034602  33.262 < 0.0000000000000002 ***
## cut.Q       -0.0339481  0.0029582 -11.476 < 0.0000000000000002 ***
## cut.C        0.0163231  0.0025452   6.413      0.0000000001447 ***
## cut^4       -0.0001739  0.0020487  -0.085              0.93235
## color.L      0.4366337  0.0028441 153.523 < 0.0000000000000002 ***
## color.Q     -0.0974187  0.0025833 -37.711 < 0.0000000000000002 ***
## color.C      0.0125915  0.0024179   5.208      0.0000001927323 ***
## color^4      0.0133674  0.0022204   6.020      0.0000000017651 ***
## color^5      0.0001194  0.0021042   0.057              0.95475
## color^6      0.0026739  0.0019108   1.399              0.16171
## clarity.L    0.9075881  0.0049934 181.757 < 0.0000000000000002 ***
## clarity.Q   -0.2483814  0.0046523 -53.389 < 0.0000000000000002 ***
## clarity.C    0.1367692  0.0039749  34.408 < 0.0000000000000002 ***
## clarity^4   -0.0684754  0.0031760 -21.560 < 0.0000000000000002 ***
## clarity^5    0.0289678  0.0025881  11.193 < 0.0000000000000002 ***
## clarity^6   -0.0022952  0.0022444  -1.023              0.30649
## clarity^7    0.0313111  0.0019831  15.789 < 0.0000000000000002 ***
## carat        1.0002821  0.0081052 123.413 < 0.0000000000000002 ***
## length       0.1751101  0.0264602   6.618      0.0000000000371 ***
## depth       -0.0884193  0.0271461  -3.257              0.00113 **
## depth_ratio  0.0136166  0.0033272   4.093      0.0000427910418 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.131 on 26810 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9828
## F-statistic: 7.321e+04 on 21 and 26810 DF,  p-value: < 0.00000000000000022
```

```r
options(scipen = 999)
GlobalCrit(LM_complete)
```

```
##
## ---------------------------------------------------
##   GLOBAL VARIABLE SELECTION PROCEDURE
##
##   ( Data =  Train_lr )
##
##   A = cut
##   B = color
##   C = clarity
##   D = carat
##   E = length
##   F = width
##   G = depth
##   H = depth_ratio
##   I = table
##
##   Models      |  Cp             |  AIC           |
##   ------------------------------------------------
##   ABCDEF      |      2.84  (10) |  32937.47  (10) |
##   ABCDEH      |      2.49  ( 9) |  32937.81  ( 9) |
##   ABCDEFG     |  -   0.63  ( 7) |  32940.94  ( 7) |
##   ABCDEFH     |  -   3.12  ( 5) |  32943.44  ( 5) |
##   ABCDEGH     |  -   6.12  ( 1) |  32946.43  ( 1) |
##   ABCDEFGH    |  -   4.80  ( 3) |  32945.11  ( 3) |
##   ABCDEFGI    |      0.80  ( 8) |  32939.51  ( 8) |
##   ABCDEFHI    |  -   2.18  ( 6) |  32942.50  ( 6) |
##   ABCDEGHI    |  -   5.45  ( 2) |  32945.77  ( 2) |
##   ABCDEFGHI   |  -   3.00  ( 4) |  32944.32  ( 4) |
##
## ---------------------------------------------------
```


```
##
## ---------------------------------------------------
##   GLOBAL VARIABLE SELECTION PROCEDURE
##
##   ( Data =  Train_lr )
##
##   A = cut
##   B = color
##   C = clarity
##   D = carat
##   E = length
##   F = width
##   G = depth
##   H = depth_ratio
##   I = table
##
##   Models      |  Cp             |  AIC           |
##   ------------------------------------------------
##   ABCDEF      |      2.84  (10) |  32937.47  (10) |
##   ABCDEH      |      2.49  ( 9) |  32937.81  ( 9) |
##   ABCDEFG     |  -   0.63  ( 7) |  32940.94  ( 7) |
##   ABCDEFH     |  -   3.12  ( 5) |  32943.44  ( 5) |
##   ABCDEGH     |  -   6.12  ( 1) |  32946.43  ( 1) |
```

```
##   ABCDEFGH    |  -   4.80   ( 3) |  32945.11  ( 3) |
##   ABCDEFGI    |      0.80   ( 8) |  32939.51  ( 8) |
##   ABCDEFHI    |  -   2.18   ( 6) |  32942.50  ( 6) |
##   ABCDEGHI    |  -   5.45   ( 2) |  32945.77  ( 2) |
##   ABCDEFGHI   |  -   3.00   ( 4) |  32944.32  ( 4) |
##
## ------------------------------------------------------
```

```
LM_CpAIC_complete = lm(price ~ . , data = Train_lr[, c(1:6, 8, 9)])
summary(LM_CpAIC_complete)
```

```
##
## Call:
## lm(formula = price ~ ., data = Train_lr[, c(1:6, 8, 9)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04014 -0.08295  0.00025  0.08161  1.42593
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0741493  0.0015667 -47.329 < 0.0000000000000002 ***
## cut.L        0.1150923  0.0034602  33.262 < 0.0000000000000002 ***
## cut.Q       -0.0339481  0.0029582 -11.476 < 0.0000000000000002 ***
## cut.C        0.0163231  0.0025452   6.413      0.0000000001447 ***
## cut^4       -0.0001739  0.0020487  -0.085              0.93235
## color.L      0.4366337  0.0028441 153.523 < 0.0000000000000002 ***
## color.Q     -0.0974187  0.0025833 -37.711 < 0.0000000000000002 ***
## color.C      0.0125915  0.0024179   5.208      0.0000001927323 ***
## color^4      0.0133674  0.0022204   6.020      0.0000000017651 ***
## color^5      0.0001194  0.0021042   0.057              0.95475
## color^6      0.0026739  0.0019108   1.399              0.16171
## clarity.L    0.9075881  0.0049934 181.757 < 0.0000000000000002 ***
## clarity.Q   -0.2483814  0.0046523 -53.389 < 0.0000000000000002 ***
## clarity.C    0.1367692  0.0039749  34.408 < 0.0000000000000002 ***
## clarity^4   -0.0684754  0.0031760 -21.560 < 0.0000000000000002 ***
## clarity^5    0.0289678  0.0025881  11.193 < 0.0000000000000002 ***
## clarity^6   -0.0022952  0.0022444  -1.023              0.30649
## clarity^7    0.0313111  0.0019831  15.789 < 0.0000000000000002 ***
## carat        1.0002821  0.0081052 123.413 < 0.0000000000000002 ***
## length       0.1751101  0.0264602   6.618      0.0000000000371 ***
## depth       -0.0884193  0.0271461  -3.257              0.00113 **
## depth_ratio  0.0136166  0.0033272   4.093      0.0000427910418 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.131 on 26810 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9828
## F-statistic: 7.321e+04 on 21 and 26810 DF,  p-value: < 0.00000000000000022
```

```
Train_minus_corr <- Train_lr[, -c(6:8)]
LM_minus_corr = lm(price ~ ., data = Train_minus_corr)
summary(LM_minus_corr)
```

```
##
```

```
## Call:
## lm(formula = price ~ ., data = Train_minus_corr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98683 -0.08456 -0.00054  0.08206  1.42888
##
## Coefficients:
##                Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -0.0733380  0.0016248  -45.136 < 0.0000000000000002 ***
## cut.L        0.1174782  0.0037111   31.656 < 0.0000000000000002 ***
## cut.Q       -0.0331237  0.0029650  -11.172 < 0.0000000000000002 ***
## cut.C        0.0132619  0.0025653    5.170         0.0000002361 ***
## cut^4       -0.0022453  0.0020453   -1.098               0.2723
## color.L      0.4322410  0.0028233  153.096 < 0.0000000000000002 ***
## color.Q     -0.0958030  0.0025851  -37.059 < 0.0000000000000002 ***
## color.C      0.0133140  0.0024238    5.493         0.0000000399 ***
## color^4      0.0125884  0.0022260    5.655         0.0000000157 ***
## color^5     -0.0001023  0.0021099   -0.048               0.9613
## color^6      0.0028366  0.0019160    1.481               0.1388
## clarity.L    0.9068901  0.0050015  181.324 < 0.0000000000000002 ***
## clarity.Q   -0.2447781  0.0046522  -52.615 < 0.0000000000000002 ***
## clarity.C    0.1346227  0.0039816   33.811 < 0.0000000000000002 ***
## clarity^4   -0.0686749  0.0031846  -21.564 < 0.0000000000000002 ***
## clarity^5    0.0288881  0.0025955   11.130 < 0.0000000000000002 ***
## clarity^6   -0.0024999  0.0022505   -1.111               0.2667
## clarity^7    0.0320730  0.0019874   16.138 < 0.0000000000000002 ***
## carat        1.0865804  0.0009254 1174.133 < 0.0000000000000002 ***
## depth_ratio -0.0016945  0.0009461   -1.791               0.0733 .
## table       -0.0001141  0.0010774   -0.106               0.9157
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1313 on 26811 degrees of freedom
## Multiple R-squared:  0.9828, Adjusted R-squared:  0.9828
## F-statistic: 7.644e+04 on 20 and 26811 DF,  p-value: < 0.00000000000000022
```

```
LM_backward_minus_corr = step(LM_minus_corr, direction = "backward")
```

```
## Start:  AIC=-108920.2
## price ~ cut + color + clarity + carat + depth_ratio + table
##
##               Df Sum of Sq     RSS     AIC
## - table        1       0.0   462.4 -108922
## <none>                       462.4 -108920
## - depth_ratio  1       0.1   462.5 -108919
## - cut          4      20.1   482.6 -107784
## - color        6     429.9   892.3  -91293
## - clarity      7     882.5  1344.9  -80288
## - carat        1   23776.4 24238.8   -2687
##
## Step:  AIC=-108922.2
## price ~ cut + color + clarity + carat + depth_ratio
##
##               Df Sum of Sq     RSS     AIC
```

```
## <none>                         462.4 -108922
## - depth_ratio  1       0.1    462.5 -108921
## - cut          4      26.3    488.7 -107445
## - color        6     430.0    892.4  -91293
## - clarity      7     883.1   1345.5  -80277
## - carat        1   24022.4 24484.8   -2418
```

```
summary(LM_backward_minus_corr)   # like LM_CpAIC_minus_corr
```

```
##
## Call:
## lm(formula = price ~ cut + color + clarity + carat + depth_ratio,
##     data = Train_minus_corr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98668 -0.08451 -0.00056  0.08207  1.42904
##
## Coefficients:
##                Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -0.0733824  0.0015697  -46.750 < 0.0000000000000002 ***
## cut.L        0.1176217  0.0034546   34.048 < 0.0000000000000002 ***
## cut.Q       -0.0331066  0.0029605  -11.183 < 0.0000000000000002 ***
## cut.C        0.0133117  0.0025217    5.279         0.0000001310 ***
## cut^4       -0.0022101  0.0020180   -1.095               0.2734
## color.L      0.4322448  0.0028231  153.112 < 0.0000000000000002 ***
## color.Q     -0.0958088  0.0025845  -37.070 < 0.0000000000000002 ***
## color.C      0.0133162  0.0024237    5.494         0.0000000396 ***
## color^4      0.0125930  0.0022255    5.658         0.0000000154 ***
## color^5     -0.0001032  0.0021098   -0.049               0.9610
## color^6      0.0028362  0.0019159    1.480               0.1388
## clarity.L    0.9069041  0.0049997  181.393 < 0.0000000000000002 ***
## clarity.Q   -0.2447790  0.0046521  -52.616 < 0.0000000000000002 ***
## clarity.C    0.1346281  0.0039812   33.816 < 0.0000000000000002 ***
## clarity^4   -0.0686735  0.0031846  -21.565 < 0.0000000000000002 ***
## clarity^5    0.0288920  0.0025952   11.133 < 0.0000000000000002 ***
## clarity^6   -0.0024996  0.0022504   -1.111               0.2667
## clarity^7    0.0320720  0.0019873   16.138 < 0.0000000000000002 ***
## carat        1.0865705  0.0009207 1180.212 < 0.0000000000000002 ***
## depth_ratio -0.0016535  0.0008632   -1.916               0.0554 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1313 on 26812 degrees of freedom
## Multiple R-squared:  0.9828, Adjusted R-squared:  0.9828
## F-statistic: 8.047e+04 on 19 and 26812 DF,  p-value: < 0.00000000000000022
```

```
LM_forward_minus_corr = step(LM_minus_corr, direction = "forward")
```

```
## Start:  AIC=-108920.2
## price ~ cut + color + clarity + carat + depth_ratio + table
```

```
summary(LM_forward_minus_corr)    # no selection
```

```
##
## Call:
```

```
## lm(formula = price ~ cut + color + clarity + carat + depth_ratio +
##     table, data = Train_minus_corr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98683 -0.08456 -0.00054  0.08206  1.42888
##
## Coefficients:
##               Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -0.0733380  0.0016248  -45.136 < 0.0000000000000002 ***
## cut.L        0.1174782  0.0037111   31.656 < 0.0000000000000002 ***
## cut.Q       -0.0331237  0.0029650  -11.172 < 0.0000000000000002 ***
## cut.C        0.0132619  0.0025653    5.170         0.0000002361 ***
## cut^4       -0.0022453  0.0020453   -1.098               0.2723
## color.L      0.4322410  0.0028233  153.096 < 0.0000000000000002 ***
## color.Q     -0.0958030  0.0025851  -37.059 < 0.0000000000000002 ***
## color.C      0.0133140  0.0024238    5.493         0.0000000399 ***
## color^4      0.0125884  0.0022260    5.655         0.0000000157 ***
## color^5     -0.0001023  0.0021099   -0.048               0.9613
## color^6      0.0028366  0.0019160    1.481               0.1388
## clarity.L    0.9068901  0.0050015  181.324 < 0.0000000000000002 ***
## clarity.Q   -0.2447781  0.0046522  -52.615 < 0.0000000000000002 ***
## clarity.C    0.1346227  0.0039816   33.811 < 0.0000000000000002 ***
## clarity^4   -0.0686749  0.0031846  -21.564 < 0.0000000000000002 ***
## clarity^5    0.0288881  0.0025955   11.130 < 0.0000000000000002 ***
## clarity^6   -0.0024999  0.0022505   -1.111               0.2667
## clarity^7    0.0320730  0.0019874   16.138 < 0.0000000000000002 ***
## carat        1.0865804  0.0009254 1174.133 < 0.0000000000000002 ***
## depth_ratio -0.0016945  0.0009461   -1.791               0.0733 .
## table       -0.0001141  0.0010774   -0.106               0.9157
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1313 on 26811 degrees of freedom
## Multiple R-squared:  0.9828, Adjusted R-squared:  0.9828
## F-statistic: 7.644e+04 on 20 and 26811 DF,  p-value: < 0.00000000000000022
```

```r
LM_stepwise_minus_corr = step(LM_minus_corr, direction = "both")
```

```
## Start:  AIC=-108920.2
## price ~ cut + color + clarity + carat + depth_ratio + table
##
##               Df Sum of Sq     RSS     AIC
## - table        1       0.0   462.4 -108922
## <none>                       462.4 -108920
## - depth_ratio  1       0.1   462.5 -108919
## - cut          4      20.1   482.6 -107784
## - color        6     429.9   892.3  -91293
## - clarity      7     882.5  1344.9  -80288
## - carat        1   23776.4 24238.8   -2687
##
## Step:  AIC=-108922.2
## price ~ cut + color + clarity + carat + depth_ratio
##
##               Df Sum of Sq     RSS     AIC
```

```
## <none>                        462.4 -108922
## - depth_ratio  1       0.1   462.5 -108921
## + table        1       0.0   462.4 -108920
## - cut          4      26.3   488.7 -107445
## - color        6     430.0   892.4  -91293
## - clarity      7     883.1  1345.5  -80277
## - carat        1   24022.4 24484.8   -2418
```

```
summary(LM_stepwise_minus_corr)   # like LM_CpAIC_minus_corr
```

```
##
## Call:
## lm(formula = price ~ cut + color + clarity + carat + depth_ratio,
##     data = Train_minus_corr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98668 -0.08451 -0.00056  0.08207  1.42904
##
## Coefficients:
##                Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -0.0733824  0.0015697  -46.750 < 0.0000000000000002 ***
## cut.L        0.1176217  0.0034546   34.048 < 0.0000000000000002 ***
## cut.Q       -0.0331066  0.0029605  -11.183 < 0.0000000000000002 ***
## cut.C        0.0133117  0.0025217    5.279         0.0000001310 ***
## cut^4       -0.0022101  0.0020180   -1.095               0.2734
## color.L      0.4322448  0.0028231  153.112 < 0.0000000000000002 ***
## color.Q     -0.0958088  0.0025845  -37.070 < 0.0000000000000002 ***
## color.C      0.0133162  0.0024237    5.494         0.0000000396 ***
## color^4      0.0125930  0.0022255    5.658         0.0000000154 ***
## color^5     -0.0001032  0.0021098   -0.049               0.9610
## color^6      0.0028362  0.0019159    1.480               0.1388
## clarity.L    0.9069041  0.0049997  181.393 < 0.0000000000000002 ***
## clarity.Q   -0.2447790  0.0046521  -52.616 < 0.0000000000000002 ***
## clarity.C    0.1346281  0.0039812   33.816 < 0.0000000000000002 ***
## clarity^4   -0.0686735  0.0031846  -21.565 < 0.0000000000000002 ***
## clarity^5    0.0288920  0.0025952   11.133 < 0.0000000000000002 ***
## clarity^6   -0.0024996  0.0022504   -1.111               0.2667
## clarity^7    0.0320720  0.0019873   16.138 < 0.0000000000000002 ***
## carat        1.0865705  0.0009207 1180.212 < 0.0000000000000002 ***
## depth_ratio -0.0016535  0.0008632   -1.916               0.0554 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1313 on 26812 degrees of freedom
## Multiple R-squared:  0.9828, Adjusted R-squared:  0.9828
## F-statistic: 8.047e+04 on 19 and 26812 DF,  p-value: < 0.00000000000000022
```

```
GlobalCrit(LM_minus_corr)
```

```
##
## -----------------------------------------------------
##   GLOBAL VARIABLE SELECTION PROCEDURE
##
##   ( Data =  Train_minus_corr )
```

```
##
##   A = cut
##   B = color
##   C = clarity
##   D = carat
##   E = depth_ratio
##   F = table
##
##   Models      |  Cp             |  AIC          |
##   ------------------------------------------------
##   BCD         |   1686.62  ( 8) |  31157.06  ( 8) |
##   ABCD        |  -   7.32  ( 2) |  32800.58  ( 2) |
##   ACDE        |  24920.99  (10) |  15162.85  ( 9) |
##   BCDE        |   1514.54  ( 7) |  31319.38  ( 7) |
##   BCDF        |   1498.62  ( 6) |  31334.46  ( 6) |
##   ABCDE       |  -   8.99  ( 1) |  32802.26  ( 1) |
##   ABCDF       |  -   5.79  ( 4) |  32799.06  ( 4) |
##   ACDEF       |  24919.19  ( 9) |  15162.82  (10) |
##   BCDEF       |   1158.93  ( 5) |  31658.16  ( 5) |
##   ABCDEF      |  -   7.00  ( 3) |  32800.27  ( 3) |
##
## -----------------------------------------------------
```

```
##
## -----------------------------------------------------
##   GLOBAL VARIABLE SELECTION PROCEDURE
##
##   ( Data =  Train_minus_corr )
##
##   A = cut
##   B = color
##   C = clarity
##   D = carat
##   E = depth_ratio
##   F = table
##
##   Models      |  Cp             |  AIC          |
##   ------------------------------------------------
##   BCD         |   1686.62  ( 8) |  31157.06  ( 8) |
##   ABCD        |  -   7.32  ( 2) |  32800.58  ( 2) |
##   ACDE        |  24920.99  (10) |  15162.85  ( 9) |
##   BCDE        |   1514.54  ( 7) |  31319.38  ( 7) |
##   BCDF        |   1498.62  ( 6) |  31334.46  ( 6) |
##   ABCDE       |  -   8.99  ( 1) |  32802.26  ( 1) |
##   ABCDF       |  -   5.79  ( 4) |  32799.06  ( 4) |
##   ACDEF       |  24919.19  ( 9) |  15162.82  (10) |
##   BCDEF       |   1158.93  ( 5) |  31658.16  ( 5) |
##   ABCDEF      |  -   7.00  ( 3) |  32800.27  ( 3) |
##
## -----------------------------------------------------
```

```r
LM_CpAIC_minus_corr = lm(price ~ . , data = Train_lr[, c(1:5, 9)])
summary(LM_CpAIC_minus_corr)
```

```
##
```

```
## Call:
## lm(formula = price ~ ., data = Train_lr[, c(1:5, 9)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98668 -0.08451 -0.00056  0.08207  1.42904
##
## Coefficients:
##               Estimate Std. Error  t value             Pr(>|t|)
## (Intercept) -0.0733824  0.0015697  -46.750 < 0.0000000000000002 ***
## cut.L        0.1176217  0.0034546   34.048 < 0.0000000000000002 ***
## cut.Q       -0.0331066  0.0029605  -11.183 < 0.0000000000000002 ***
## cut.C        0.0133117  0.0025217    5.279         0.0000001310 ***
## cut^4       -0.0022101  0.0020180   -1.095               0.2734
## color.L      0.4322448  0.0028231  153.112 < 0.0000000000000002 ***
## color.Q     -0.0958088  0.0025845  -37.070 < 0.0000000000000002 ***
## color.C      0.0133162  0.0024237    5.494         0.0000000396 ***
## color^4      0.0125930  0.0022255    5.658         0.0000000154 ***
## color^5     -0.0001032  0.0021098   -0.049               0.9610
## color^6      0.0028362  0.0019159    1.480               0.1388
## clarity.L    0.9069041  0.0049997  181.393 < 0.0000000000000002 ***
## clarity.Q   -0.2447790  0.0046521  -52.616 < 0.0000000000000002 ***
## clarity.C    0.1346281  0.0039812   33.816 < 0.0000000000000002 ***
## clarity^4   -0.0686735  0.0031846  -21.565 < 0.0000000000000002 ***
## clarity^5    0.0288920  0.0025952   11.133 < 0.0000000000000002 ***
## clarity^6   -0.0024996  0.0022504   -1.111               0.2667
## clarity^7    0.0320720  0.0019873   16.138 < 0.0000000000000002 ***
## carat        1.0865705  0.0009207 1180.212 < 0.0000000000000002 ***
## depth_ratio -0.0016535  0.0008632   -1.916               0.0554 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1313 on 26812 degrees of freedom
## Multiple R-squared:  0.9828, Adjusted R-squared:  0.9828
## F-statistic: 8.047e+04 on 19 and 26812 DF,  p-value: < 0.00000000000000022
```

```r
kable_styling(
  kable(
  data.table(Model = c("LM_complete", "LM_forward_complete", "LM_backward_complete",
                       "LM_stepwise_complete", "LM_CpAIC_complete",
                       "LM_minus_corr", "LM_forward_minus_corr", "LM_backward_minus_corr",
                       "LM_stepwise_minus_corr", "LM_CpAIC_minus_corr"),
             Cut           = c("X","X","X","X","X","X","X","X","X","X"),
             Color         = c("X","X","X","X","X","X","X","X","X","X"),
             Clarity       = c("X","X","X","X","X","X","X","X","X","X"),
             Carat         = c("X","X","X","X","X","X","X","X","X","X"),
             Length        = c("X","X","X","X","X"," "," "," "," "," "),
             Width         = c("X","X"," "," "," "," "," "," "," "," "),
             Depth         = c("X","X","X","X","X"," "," "," "," "," "),
             `Depth Ratio` = c("X","X","X","X","X","X","X","X","X","X"),
             Table         = c("X","X"," "," "," ","X","X"," "," "," ")
             ),
  align = 'lccccccccc') %>% kable_classic(),
  full_width = TRUE)
```

Model	Cut	Color	Clarity	Carat	Length	Width	Depth	Depth Ratio	Table
LM_complete	X	X	X	X	X	X	X	X	X
LM_forward_complete	X	X	X	X	X	X	X	X	X
LM_backward_complete	X	X	X	X	X		X	X	
LM_stepwise_complete	X	X	X	X	X		X	X	
LM_CpAIC_complete	X	X	X	X	X		X	X	
LM_minus_corr	X	X	X	X				X	X
LM_forward_minus_corr	X	X	X	X				X	X
LM_backward_minus_corr	X	X	X	X				X	
LM_stepwise_minus_corr	X	X	X	X				X	
LM_CpAIC_minus_corr	X	X	X	X				X	

In both cases, the forward selection doesn't discard any variables, whereas backward, stepwise and global selections all choose the same model with less variables than initially.

Thus, we have four different models emerging. We will keep `LM_complete`, `LM_CpAIC_complete`, `LM_minus_corr` and `LM_CpAIC_minus_corr` and remove the other models which are duplicates.

```
rm(LM_forward_complete, LM_backward_complete, LM_stepwise_complete, LM_forward_minus_corr, LM_backward_m
```

```
# predicting prices of validation set on the validation data

LM_Predictions =
  data.table(
    LM_complete_pred = predict(object = LM_complete, newdata = Valid_lr),
    LM_CpAIC_complete_pred = predict(object = LM_CpAIC_complete, newdata = Valid_lr),
    LM_minus_corr_pred = predict(object = LM_minus_corr, newdata = Valid_lr),
    LM_CpAIC_minus_corr_pred = predict(object = LM_CpAIC_minus_corr, newdata = Valid_lr)
    )

# we have to scale back the price, to do so we fetch the mean and std value from norm.values
# dsplaying all means and stds
norm.values$mean
```

```
##       price       carat      length       width       depth depth_ratio       table
##   7.7876577  -0.3941013   5.7331235   5.7353183   3.5414058  61.7640019   4.0504070
```

```
norm.values$std
```

```
##       price       carat      length       width       depth depth_ratio       table
##  1.01430151  0.58510643  1.12051551  1.11223832  0.69253953  1.41492652  0.03808011
```

```
# fetching for price
mean_price = norm.values$mean[1]
std_price = norm.values$std[1]

# scaling back (Y*mu + sigma), then exp() (we had transformed price with a log for skewness)
LM_Predictions = LM_Predictions*std_price + mean_price
LM_Predictions = exp(LM_Predictions)


# taking real prices of validation data from diamonds (which has not been touched -> original scale)
LM_Predictions[, real_prices := diamonds[valid.index, price]]

Acc1 = accuracy(object = LM_Predictions$LM_complete_pred, x = LM_Predictions$real_prices)
```

Model	ME	RMSE	MAE	MPE	MAPE
LM_complete	36.36648	846.5888	408.5364	-0.8377036	10.33643
LM_CpAIC_complete	36.32774	845.5018	408.1349	-0.8408540	10.33659
LM_minus_corr	50.38994	810.1522	405.0486	-0.8420621	10.39559
LM_CpAIC_minus_corr	50.39878	810.1610	405.0616	-0.8418121	10.39568

```
Acc1 = as.data.table(Acc1)
Acc2 = accuracy(object = LM_Predictions$LM_CpAIC_complete_pred, x = LM_Predictions$real_prices)
Acc2 = as.data.table(Acc2)
Acc3 = accuracy(object = LM_Predictions$LM_minus_corr_pred, x = LM_Predictions$real_prices)
Acc3 = as.data.table(Acc3)
Acc4 = accuracy(object = LM_Predictions$LM_CpAIC_minus_corr_pred, x = LM_Predictions$real_prices)
Acc4 = as.data.table(Acc4)
Accs = list(Acc1, Acc2, Acc3, Acc4)
Accs = rbindlist(Accs)
#rm(Acc1, Acc2, Acc3, Acc4)
Accs[, Model := c("LM_complete", "LM_CpAIC_complete", "LM_minus_corr", "LM_CpAIC_minus_corr")]
Accs <- Accs[, c(6, 1:5)]
kable_styling(kable(Accs)%>% kable_classic(), full_width = TRUE)
```

k-NN

```
diamonds_dummies <- diamonds
diamonds_dummies <- dummy_cols(diamonds_dummies,
                               select_columns = c("cut", "color", "clarity"),
                               remove_selected_columns = TRUE)

#rename column to avoid issues with Neural net function
colnames(diamonds_dummies)[10] = "cut_Very_Good"
```

```
#create data frames to normalize
knn_train <- diamonds_dummies[train.index,]
knn_valid <- diamonds_dummies[valid.index,]

knn_train_norm <- knn_train
knn_valid_norm <- knn_valid

# use preProcess() to normalize non-categorical variables
norm_values <- preProcess(knn_train[, c(1:7)], method=c("center", "scale"))

knn_train_norm[, c(1:7)] <- predict(norm_values, knn_train[, c(1:7)])
knn_valid_norm[,c(1:7)] <- predict(norm_values, knn_valid[, c(1:7)])
```

```
# computing kNN with knnreg from caret package
kNN = knnreg(x = knn_train_norm[, -c(1)] , y = knn_train_norm$price, k = 1)

# predicting prices of validation set on the validation data
knn_pred_y = predict(object = kNN, newdata = knn_valid_norm[, -c(1)])

# we have to scale back the price, to do so we fetch the mean and std value from norm.values
# displaying all means and standard deviations
norm_values$mean
```

	ME	RMSE	MAE	MPE	MAPE
Test set	20.59665	954.5954	487.1664	-1.833533	13.73317

```
##        price       carat      length       width       depth depth_ratio       table
## 3934.4128280   0.7987388   5.7331235   5.7353183   3.5414058  61.7640019  57.4628205
```

```
norm_values$std
```

```
##        price       carat      length       width       depth depth_ratio       table
## 3991.7064934   0.4742740   1.1205155   1.1122383   0.6925395   1.4149265   2.2176345
```

```
# fetching for price
mean_price = norm_values$mean[1]
std_price = norm_values$std[1]

# scaling back (Y*mu + sigma)
knn_rescaled_prices = std_price * knn_pred_y + mean_price

# taking real prices of validation data from diamonds (which has not been touched -> original scale)
real_prices = diamonds[valid.index, price]

# computing accuracy measures
kNN_aa1 <- accuracy(object = knn_rescaled_prices, x = real_prices)

kable_styling(kable(kNN_aa1)%>% kable_classic(), full_width = TRUE)
```

```
#create a data frame to store accuracy results
accuracy_df <- data.frame(k = seq(1, 20, 1), accuracy = rep(0, 20))

# use validation data set to compute various values of Knn
for(i in 1:20) {
temp_knn_pred <- knnreg(x = knn_train_norm[, -c(1)] , y = knn_train_norm$price, k = i)

temp_knn_pred_y = predict(object = temp_knn_pred, newdata = knn_valid_norm[, -c(1)])
temp_knn_rescaled_prices = std_price * temp_knn_pred_y + mean_price

#update the accuracy table with results of confusion matrix accuracy from the loop
accuracy_df[i, 2] <- accuracy(object = temp_knn_rescaled_prices, x = real_prices)[2]
}

#display results
row_spec(kable_classic(kbl(accuracy_df)), 4, bold = T, color = "white", background = "#D8B365")
```

```
#plot results
ggplot (data = accuracy_df, aes(x = k, y = accuracy)) +
  geom_line (size = 1.2, color = "black") +
  geom_point(data= accuracy_df[4,], aes(x = k, y = accuracy), color = "#D8B365", size = 3) +
  labs(x = "Amount of Neighbors", y = "RMSE per Model", title = "Behavior of RMSE per amount of kNN")+
  theme_classic()
```

```
# computing kNN with knnreg from caret package
opt_kNN = knnreg(x = knn_train_norm[, -c(1)] , y = knn_train_norm$price, k = 4)

# predicting prices of validation set on the validation data
```

k	accuracy
1	954.5954
2	861.0887
3	827.1196
4	814.5940
5	819.2859
6	818.5541
7	822.7554
8	833.2104
9	839.9301
10	847.3985
11	850.7010
12	854.8280
13	860.4922
14	867.3541
15	874.8586
16	879.9069
17	886.0747
18	891.9780
19	896.3521
20	900.0938

	ME	RMSE	MAE	MPE	MAPE
Test set	40.86142	814.594	431.2913	-2.45264	12.13511

```
opt_knn_pred_y = predict(object = opt_kNN, newdata = knn_valid_norm[, -c(1)])

# scaling back (Y*mu + sigma)
opt_knn_rescaled_prices = std_price * opt_knn_pred_y + mean_price

# computing accuracy measures
opt_accuracy <- accuracy(object = opt_knn_rescaled_prices, x = real_prices)
kable_styling(kable(opt_accuracy)%>% kable_classic(), full_width = TRUE)
```

Regression Tree

Partitioning

```
#Rename data specifically for regression trees
diamonds_tree <- diamonds

# Creating data tables Train, Valid and Test using the indexes for the regression tree section
Train_rg <- diamonds[train.index, ]
Valid_rg <- diamonds[valid.index, ]
```

Regression Tree

```
# Use rpart() to run tree on continuous response
RegressTree <- rpart(price ~ .,
              data = Train_rg,
              method = "anova")
```

	ME	RMSE	MAE	MPE	MAPE
Test set	6.097627	1267.154	846.9286	-14.54158	33.08388

```r
# Generates a cost complexity parameter table that provides the complexity parameter value
#summary(RegressTree)

# Plots a regression tree
fancyRpartPlot(RegressTree, caption = NULL, main = "Regression Tree", palettes = "YlGnBu", digits = -3)
```

```r
# Count number of leaves
length(RegressTree$frame$var[RegressTree$frame$var == "<leaf>"])
```

```
## [1] 8
```

```r
# kable and kable_styling as before
# We multiply by 100, divide by the sum and round the percentages to 2 decimals
kable_styling(kable(round(100*RegressTree$variable.importance / sum(RegressTree$variable.importance), 2
```

	Importance %
width	25.50
length	24.30
carat	23.90
depth	22.57
clarity	2.53
color	1.08
depth_ratio	0.09
table	0.02
cut	0.01

```r
# Predict errors using accuracy()
tree_aa1 <- forecast::accuracy(predict(RegressTree, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa1)%>% kable_classic(), full_width = TRUE)
```

```r
# Predict the diamond price with validation
pred_Diamond_test <- predict(RegressTree, newdata = Valid_rg)

# Display first 14 observations
head(pred_Diamond_test,14)
```

```
##        1        2        3        4        5        6        7        8        9       10       11
## 5440.419 1059.257 1059.257 8560.343 1059.257 1059.257 3203.633 8414.236 3203.633 1059.257 5440.419 1
##       13       14
## 1059.257 5440.419
```

'Exlcusion' Regression Tree

```r
RegressTree2 <- rpart(price ~ length+width+depth+carat+cut+color+clarity,
              data = Train_rg,
              method = "anova")

# Generates a cost complexity parameter table that provides the complexity parameter value
#summary(RegressTree2)
```

	ME	RMSE	MAE	MPE	MAPE
Test set	6.097627	1267.154	846.9286	-14.54158	33.08388

```
# Plots a regression tree
fancyRpartPlot(RegressTree2, caption = NULL, main = "Exclusion Regression Tree", palettes = "YlGnBu", di
```

```
# kable and kable_styling as before
# We multiply by 100, divide by the sum and round the percentages to 2 decimals
kable_styling(kable(round(100*RegressTree2$variable.importance / sum(RegressTree2$variable.importance),
```

	Importance %
width	25.51
length	24.31
carat	23.91
depth	22.58
clarity	2.55
color	1.08
cut	0.05

```
# Predict errors using accuracy()
tree_aa2 <- forecast::accuracy(predict(RegressTree2, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa2)%>% kable_classic(), full_width = TRUE)
```

```
# Predict the diamond price with validation
pred_Diamond_test <- predict(RegressTree, newdata = Valid_rg)

# Display first 14 observations
head(pred_Diamond_test,14)
```

```
##        1        2        3        4        5        6        7        8        9       10       11
## 5440.419 1059.257 1059.257 8560.343 1059.257 1059.257 3203.633 8414.236 3203.633 1059.257 5440.419 10
##       13       14
## 1059.257 5440.419
```

```
#Get the lowest CP value from CP table
min.xerror <- RegressTree2$cptable[which.min(RegressTree2$cptable[,"xerror"]),"CP"]

min.xerror
```

```
## [1] 0.01
```

```
#Plot the optimal Cp value
plotcp(RegressTree2)
```

Pruned Regression Tree

```
RegressTree_pruned <- prune(RegressTree2, cp = min.xerror)

# Draw the prune tree
fancyRpartPlot(RegressTree_pruned, caption = NULL, main = "'Pruned' Regression Tree", palettes = "YlGnBu
```

	ME	RMSE	MAE	MPE	MAPE
Test set	6.097627	1267.154	846.9286	-14.54158	33.08388

	ME	RMSE	MAE	MPE	MAPE
Test set	-3.898022	3660.422	2765.963	-144.4894	171.6511

```
# Predict errors using accuracy()
tree_aa3 <- forecast::accuracy(predict(RegressTree_pruned, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa3)%>% kable_classic(), full_width = TRUE)
```

Boosted Tree

```
# Boosted tree
set.seed(111)
tree_boost10 <- gbm(price ~., data = Train_rg, distribution = "gaussian", n.trees = 10, interaction.dept

# Interaction Depth specifies the maximum depth of each tree( i.e. highest level of variable interactio
# Shrinkage is considered as the learning rate. It is used for reducing, or shrinking, the impact of ea
#n.trees: Integer specifying the total number of trees to fit. This is equivalent to the number of iter


tree_boost10
```

```
## gbm(formula = price ~ ., distribution = "gaussian", data = Train_rg,
##     n.trees = 10, interaction.depth = 6, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 10 iterations were performed.
## There were 9 predictors of which 5 had non-zero influence.
```

```
vip::vip(tree_boost10, aesthetics = list(fill = "#D8B365")) +
  ggtitle("Boosted Tree Variable Importance") + # Title name
  xlab("Variable") + # Label names
  theme_classic() # A classic theme, with x and y axis lines and no grid lines
```

```
# Predict errors using accuracy()
tree_aa4 <- forecast::accuracy(predict(tree_boost10, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa4)%>% kable_classic(), full_width = TRUE)
```

```
# Boosted tree
set.seed(111)
tree_boost30 <- gbm(price ~., data = Train_rg, distribution = "gaussian", n.trees = 30)

vip::vip(tree_boost30, aesthetics = list(fill = "#D8B365")) +
  ggtitle("Boosted Tree Variable Importance") + # Title name
  xlab("Variable") + # Label names
  theme_classic() # A classic theme, with x and y axis lines and no grid lines
```

	ME	RMSE	MAE	MPE	MAPE
Test set	2.5438	1531.521	985.6974	-35.25731	46.85606

	ME	RMSE	MAE	MPE	MAPE
Test set	9.002539	1170.295	680.2182	-12.85125	26.22023

```r
tree_boost30
```

```
## gbm(formula = price ~ ., distribution = "gaussian", data = Train_rg,
##     n.trees = 30)
## A gradient boosted model with gaussian loss function.
## 30 iterations were performed.
## There were 9 predictors of which 4 had non-zero influence.
```

```r
# Predict errors using accuracy()
tree_aa5 <- forecast::accuracy(predict(tree_boost30, Train_rg), Train_rg$price)

kable_styling(kable(tree_aa5)%>% kable_classic(), full_width = TRUE)
```

```r
# Boosted tree
set.seed(111)
tree_boost100 <- gbm(price ~., data = Train_rg, distribution = "gaussian", cv.folds = 3)

# cv.folds: Number of cross-validation folds to perform. If cv.folds>1 then gbm, in addition to the usu

vip::vip(tree_boost100, aesthetics = list(fill = "#D8B365")) +
  ggtitle("Boosted Tree Variable Importance") + # Title name
  xlab("Variable") + # Label names
  theme_classic() # A classic theme, with x and y axis lines and no grid lines
```

```r
tree_boost100
```

```
## gbm(formula = price ~ ., distribution = "gaussian", data = Train_rg,
##     cv.folds = 3)
## A gradient boosted model with gaussian loss function.
## 100 iterations were performed.
## The best cross-validation iteration was 100.
## There were 9 predictors of which 6 had non-zero influence.
```

```r
# Predict errors using accuracy()
tree_aa6 <- forecast::accuracy(predict(tree_boost100, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa6)%>% kable_classic(), full_width = TRUE)
```

```r
pred_boost <- predict.gbm(tree_boost100, newdata = Valid_rg)

head(pred_boost, 14)
```

```
##  [1] 6112.0846 1582.6043  743.7071 6352.0316 1140.2701 1491.0859 2806.4714 9855.1238 2681.4129 1330.4
## [11] 7595.4454 1741.6017 1380.0364 6132.5267
```

	ME	RMSE	MAE	MPE	MAPE
Test set	0.9333074	1248.769	808.9297	-14.35086	31.45019

Bagging Tree

```
# Bagged tree
set.seed(111)
tree_bagging <- bagging(price ~., data = Train_rg, coob = TRUE)
tree_bagging
```

```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = price ~ ., data = Train_rg, coob = TRUE)
##
## Out-of-bag estimate of root mean squared error:  1268.231
```

```
ss<-varImp(tree_bagging)

data<-data.table(name=row.names(ss),value=ss$Overall)

data[,ggplot(.SD, aes(x=reorder(name, ss$Overall), y=ss$Overall)) +
  geom_bar(stat = "identity", fill = "#D8B365") +
  xlab("Variable") + # Label names +
  ylab("Importance") + # Label names +
  ggtitle("Bagged Tree Variable Importance") + # Title name
  theme_classic() + # A classic theme, with x and y axis lines and no grid lines
  coord_flip(),]
```

```
tree_aa7 <- forecast::accuracy(predict(tree_bagging, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa7)%>% kable_classic(), full_width = TRUE)
```

```
pred_bagged <- predict(tree_bagging, newdata = Valid_rg)

head(pred_bagged,14)
```

```
##  [1]  5396.881  1104.498  1046.993  8084.493  1104.498  1104.498  3123.791 10274.884  3123.791  1046
## [11]  6052.800  1104.498  1046.993  5953.609
```

RandomForest

```
options(scipen = 9999)

# Create randomForest
set.seed(111)
rdf_model <- randomForest(price~ ., ntree= 60, data = Train_rg)
rdf_model
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = Train_rg, ntree = 60)
##                Type of random forest: regression
```

	ME	RMSE	MAE	MPE	MAPE
Test set	2.969775	577.63	289.5139	-1.402886	7.235286

```
##                      Number of trees: 60
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 335950
##                     % Var explained: 97.89
```

```
plot(rdf_model,
         main = "Number of Optimal Trees")
```

```
# Number of trees with lowest MSE
# Compared with 80 and 100 trees, the trade-off with both of those were small so we stuck with 60 trees
which.min(rdf_model$mse)
```

```
## [1] 59
```

```
options(scipen = 9999)

# Create variable importance chart
vip::vip(rdf_model, aesthetics = list(fill = "#D8B365")) +
  ggtitle("RandomTree Variable Importance") + # Title name
  xlab("Variable") + # Label names
  theme_classic() # A classic theme, with x and y axis lines and no grid lines
```

```
tree_aa8 <- forecast::accuracy(predict(rdf_model, Valid_rg), Valid_rg$price)

kable_styling(kable(tree_aa8)%>% kable_classic(), full_width = TRUE)
```

```
pred_random <- predict(rdf_model, newdata = Valid_rg)

head(pred_random,14)
```

```
##         1         2         3         4         5         6         7         8         9        10
## 5012.2100 1782.6600  591.9428 9185.5925 1319.5133 1344.4381 2423.8250 8620.2650 2562.5817  936.9519
##        12        13        14
## 1788.5992 1172.3436 7014.9981
```

Create Regression Tree Summary Table

```
Tree_res <- data.frame("Model_Tree" = c("Reg. Tree",
                                       "Exclus. RT",
                                       "Pruned RT",
                                       "Boost10",
                                       "Boost30",
                                       "Boost100",
                                       "Bagging",
                                       "RndmFrst"),
                      "ME" = "",
                      "RMSE" = "",
                      "MAE"= "",
```

	ME	RMSE	MAE	MPE	MAPE
Reg. Tree	6.1	1267.15	846.93	-14.54	33.08
Exclus. RT	6.1	1267.15	846.93	-14.54	33.08
Pruned RT	6.1	1267.15	846.93	-14.54	33.08
Boost10	-3.9	3660.42	2765.96	-144.49	171.65
Boost30	2.54	1531.52	985.7	-35.26	46.86
Boost100	9	1170.3	680.22	-12.85	26.22
Bagging	0.93	1248.77	808.93	-14.35	31.45
RndmFrst	2.97	577.63	289.51	-1.4	7.24

```
                     "MPE" = "",
                     "MAPE" = "")

rownames(Tree_res) <- Tree_res$Model_Tree
Tree_res$Model_Tree = NULL

for (i in 1:8) {
  Tree_res[i,]= round(get(paste("tree_aa", i, sep ="")),2)
}

kable_styling(kable(Tree_res)%>% kable_classic(), full_width = TRUE)
```

```
RGRMSEplotdata <- t(Tree_res$RMSE)
colnames(RGRMSEplotdata) <- t(sort(rownames(Tree_res),decreasing = TRUE))
RGRMSEplotdata
```

```
##      RndmFrst  Reg. Tree Pruned RT Exclus. RT Boost30   Boost100 Boost10   Bagging
## [1,] "1267.15" "1267.15" "1267.15" "3660.42"  "1531.52" "1170.3" "1248.77" "577.63"
```

```
barplot(RGRMSEplotdata, col = "#D8B365",border = "#D8B365" ,
         main = "Regression Tree Comparison", ylab = "RMSE", las = 3, ylim=c(0,4000))
```

NeuralNetworks

Data preprocessing

```
#create dummy columns
diamonds_dummies <- dummy_cols(diamonds, select_columns = c("cut", "color", "clarity"))
diamonds_dummies <- diamonds_dummies[,-c(2:4)]

#rename column to avoid issues with Neural net function
colnames(diamonds_dummies)[10] = "cut_Very_Good"
```

Partitioning

```
Train_nn <- diamonds_dummies[train.index,]
Valid_nn <- diamonds_dummies[valid.index,]
```

Normalizing Data

```
# use preProcess() to normalize non-categorical variables
norm_values <- preProcess(Train_nn[1:7], method= c("range"))

Train_nn_norm <- predict(norm_values, Train_nn)
Valid_nn_norm <- predict(norm_values, Valid_nn)
```

Model 1 (Layers = 1, Nodes = 1)

- Creating Model 1

```
#create data sets to train and validate the model
#train
x_train <- c(t(Train_nn_norm[, -c(1)]))
x_train <- as.array(x_train,
                    dim(t(Train_nn_norm[, -c(1)])),
                    dimnames = list(rownames(x_train), colnames(x_train)))
x_train <- as_tensor(x_train, shape = dim(Train_nn_norm[, -c(1)]))
y_train <- as_tensor(Train_nn_norm$price)

#validation
x_valid <- c(t(Valid_nn_norm[, -c(1)]))
x_valid <- as.array(x_valid,
                    dim(t(Valid_nn_norm[, -c(1)])),
                    dimnames = list(rownames(x_valid), colnames(x_valid)))
x_valid <- as_tensor(x_valid, shape = dim(Valid_nn_norm[, -c(1)]))
y_valid <- as_tensor(Valid_nn_norm$price)

#create model
rm(model_1_1) #to prevent retraining of existing model
tf$random$set_seed(111)
model_1_1 <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(1, activation = "sigmoid", name = "HiddenLayer") %>%  # 1st hidden layer (26 nodes)
  layer_dense(1, activation = "sigmoid", name = "outputLayer")     # output layer (1 node)


#compile the model to be trained
model_1_1 %>% compile(
  optimizer = optimizer_adam(),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )

#train the model and find the lowest rmse
history <- model_1_1 %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 1

	ME	RMSE	MAE	MPE	MAPE
Test set	-6.616364	3982.992	3025.611	-157.4749	187.2922

```
# prediction values of validation set
y_valid_pred <- predict(model_1_1, x_valid)

# fetching a and b from standardization for scaling back price
a <- norm_values$ranges[1,1]
b <- norm_values$ranges[2,1]

valRescale <- function (x) {
  value <- (x * (b-a) + a)
  return(value)
}

# scaling back price predictions
y_valid_pred <- valRescale(y_valid_pred)

# we have to change the class of y_valid_pred
class(y_valid_pred)
```

```
## [1] "matrix" "array"
```

```
y_valid_pred <- as.numeric(y_valid_pred)
class(y_valid_pred)
```

```
## [1] "numeric"
```

```
# taking real values of validation set from original data (which wasn't standardized!)
y_valid_real <- diamonds[valid.index, price]

# copmuting accuracy measures of validation set
acc_1 <- accuracy(object = y_valid_pred, x = y_valid_real)

kable_styling(kable(acc_1)%>% kable_classic(), full_width = TRUE)
```

Model 2 (Layers = 1, Nodes = 26)

- Creating Model 2

```
#create model
rm(model_1_26) #to prevent retraining of existing model
tf$random$set_seed(111)
model_1_26 <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer") %>%  # 1st hidden layer (26 nodes)
  layer_dense(1, activation = "sigmoid", name = "outputLayer")      # output layer (1 node)


#compile the model to be trained
model_1_26 %>% compile(
  optimizer = optimizer_adam(),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )
```

	ME	RMSE	MAE	MPE	MAPE
Test set	-83.27564	745.1569	450.941	-11.36538	17.91012

```
#train the model and find the lowest rmse
history <- model_1_26 %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 2

```
# prediction values of validation set
y_valid_pred_1_26 <- predict(model_1_26, x_valid)

# scaling back price predictions
y_valid_pred_1_26 <- valRescale(y_valid_pred_1_26)

# we have to change the class of y_valid_pred
y_valid_pred_1_26 <- as.numeric(y_valid_pred_1_26)

# computing accuracy measures of validation set
acc_2 <- accuracy(object = y_valid_pred_1_26, x = y_valid_real)

kable_styling(kable(acc_2)%>% kable_classic(), full_width = TRUE)
```

Model 3 (Layers = 1, Nodes = 13)

- Creating Model 3

```
#create model
rm(model_1_13) #to prevent retraining of existing model
tf$random$set_seed(111)
model_1_13 <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(13, activation = "sigmoid", name = "HiddenLayer") %>%  # 1st hidden layer (26 nodes)
  layer_dense(1, activation = "sigmoid", name = "outputLayer")     # output layer (1 node)


#compile the model to be trained
model_1_13 %>% compile(
  optimizer = optimizer_adam(),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )

#train the model and find the lowest rmse
history <- model_1_13 %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 3

	ME	RMSE	MAE	MPE	MAPE
Test set	-86.71198	744.0357	454.3788	-12.8878	19.21285

```r
# prediction values of validation set
y_valid_pred_1_13 <- predict(model_1_13, x_valid)

# scaling back price predictions
y_valid_pred_1_13 <- valRescale(y_valid_pred_1_13)

# we have to change the class of y_valid_pred
y_valid_pred_1_13 <- as.numeric(y_valid_pred_1_13)

# copmuting accuracy measures of validation set
acc_3 <- accuracy(object = y_valid_pred_1_13, x = y_valid_real)

kable_styling(kable(acc_3)%>% kable_classic(), full_width = TRUE)
```

Model 4 (Layers = 2, Nodes = 26)

- Creating Model 4

```r
#create model
rm(model_2_26) #to prevent retraining of existing model
tf$random$set_seed(111)
model_2_26 <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer") %>%  # 1st hidden layer (26 nodes)
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer2") %>%  # 2nd hidden layer (26 nodes)
  layer_dense(13, activation = "sigmoid", name = "HiddenLayer3") %>%  # 3rd hidden layer (26 nodes)
  layer_dense(1, activation = "sigmoid", name = "outputLayer")     # output layer (1 node)


#compile the model to be trained
model_2_26 %>% compile(
  optimizer = optimizer_adam(),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )

#train the model and find the lowest rmse
history <- model_2_26 %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 4

```r
# prediction values of validation set
y_valid_pred_2_26 <- predict(model_2_26, x_valid)

# scaling back price predictions
y_valid_pred_2_26 <- valRescale(y_valid_pred_2_26)
```

	ME	RMSE	MAE	MPE	MAPE
Test set	30.21931	627.652	364.5433	-5.429278	13.48542

```
# we have to change the class of y_valid_pred
y_valid_pred_2_26 <- as.numeric(y_valid_pred_2_26)

# copmuting accuracy measures of validation set
acc_4 <- accuracy(object = y_valid_pred_2_26, x = y_valid_real)

kable_styling(kable(acc_4)%>% kable_classic(), full_width = TRUE)
```

Model 5 with Initializer (Layers = 2, Nodes = 26, GlorotNormal)

- Creating Model 5

```
#create model
rm(model_2_26G) #to prevent retraining of existing model
tf$random$set_seed(111)
model_2_26G <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer", kernel_initializer = "GlorotNormal") %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer2", kernel_initializer = "GlorotNormal") %
    layer_dense(26, activation = "sigmoid", name = "HiddenLayer3", kernel_initializer = "GlorotNormal")
  layer_dense(1, activation = "sigmoid", name = "outputLayer")     # output layer (1 node)


#compile the model to be trained
model_2_26G %>% compile(
  optimizer = optimizer_adam(),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )

#train the model and find the lowest rmse
history <- model_2_26G %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 5

```
# prediction values of validation set
y_valid_pred_2_26G <- predict(model_2_26G, x_valid)

# scaling back price predictions
y_valid_pred_2_26G <- valRescale(y_valid_pred_2_26G)

# we have to change the class of y_valid_pred
y_valid_pred_2_26G <- as.numeric(y_valid_pred_2_26G)

# copmuting accuracy measures of validation set
acc_5 <- accuracy(object = y_valid_pred_2_26G, x = y_valid_real)
```

	ME	RMSE	MAE	MPE	MAPE
Test set	-32.61097	630.0157	358.2125	-6.623257	12.71689

	ME	RMSE	MAE	MPE	MAPE
Test set	8.980655	577.0847	322.9464	-2.03136	10.22919

```
kable_styling(kable(acc_5)%>% kable_classic(), full_width = TRUE)
```

Model 6 with Initializer and Learning Rate (Layers = 2, Nodes = 26, GlorotNormal, LR = 0.005)

- Creating model 6

```
#create model
rm(model_2_26GLR) #to prevent retraining of existing model
tf$random$set_seed(111)
model_2_26GLR <- keras_model_sequential(input_shape = ncol(x_train)) %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer", kernel_initializer = "GlorotNormal") %>%
  layer_dense(26, activation = "sigmoid", name = "HiddenLayer2", kernel_initializer = "GlorotNormal") %
    layer_dense(13, activation = "sigmoid", name = "HiddenLayer3", kernel_initializer = "GlorotNormal")
  layer_dense(1, activation = "sigmoid", name = "outputLayer")     # output layer (1 node)


#compile the model to be trained
model_2_26GLR %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.009),
  loss = loss_mean_squared_error(),
  metric = metric_root_mean_squared_error()
  )

#train the model and find the lowest rmse
history <- model_2_26GLR %>% fit(
    x = x_train,
    y = y_train,
    validation_data = c(x_valid, y_valid),
    verbose = FALSE
    )
```

- Measuring Error Model 6

```
# prediction values of validation set
y_valid_pred_2_26GLR <- predict(model_2_26GLR, x_valid)

# scaling back price predictions
y_valid_pred_2_26GLR <- valRescale(y_valid_pred_2_26GLR)

# we have to change the class of y_valid_pred
y_valid_pred_2_26GLR <- as.numeric(y_valid_pred_2_26GLR)

# copmuting accuracy measures of validation set
acc_6 <- accuracy(object = y_valid_pred_2_26GLR, x = y_valid_real)

kable_styling(kable(acc_6)%>% kable_classic(), full_width = TRUE)
```

	ME	RMSE	MAE	MPE	MAPE
L1 N1	-6.62	3982.99	3025.61	-157.47	187.29
L1 N26	-83.28	745.16	450.94	-11.37	17.91
L1 N13	-86.71	744.04	454.38	-12.89	19.21
L2 N26	30.22	627.65	364.54	-5.43	13.49
L2 N26 G	-32.61	630.02	358.21	-6.62	12.72
L2 N26 G LR	8.98	577.08	322.95	-2.03	10.23

Create Neural Net Summary Table

```
NN_res <- data.frame("Model" = c("L1 N1",
                                 "L1 N26",
                                 "L1 N13",
                                 "L2 N26",
                                 "L2 N26 G",
                                 "L2 N26 G LR"),
                     "ME" = "",
                     "RMSE" = "",
                     "MAE"= "",
                     "MPE" = "",
                     "MAPE" = "")

rownames(NN_res) <- NN_res$Model
NN_res$Model = NULL

for (i in 1:6) {
  NN_res[i,]= round(get(paste("acc_", i, sep ="")),2)
}

kable_styling(kable(NN_res)%>% kable_classic(), full_width = FALSE)
```

```
NNRMSEplotdata <- t(NN_res$RMSE)
colnames(NNRMSEplotdata) <- t(rownames(NN_res))

barplot(NNRMSEplotdata, col = "#D8B365",border = "#D8B365" ,
         main = "Neural Network Comparison", ylab = "RMSE", las = 3, ylim=c(0,4000))
```

Neural network graphic description

Ensembles

```
#create Table with 0 values
ensemble_summ <- data.frame("MLR" = 0,
                            "RegTrees" = 0,
                            "kNN" = 0,
                            "NeuralNets" = 0)
#create empty rows
ensemble_summ[length(valid.index),] <- 0

#Add values per columns
ensemble_summ[,"MLR"] = LM_Predictions$LM_CpAIC_minus_corr_pred
ensemble_summ[,"RegTrees"] = pred_random
```

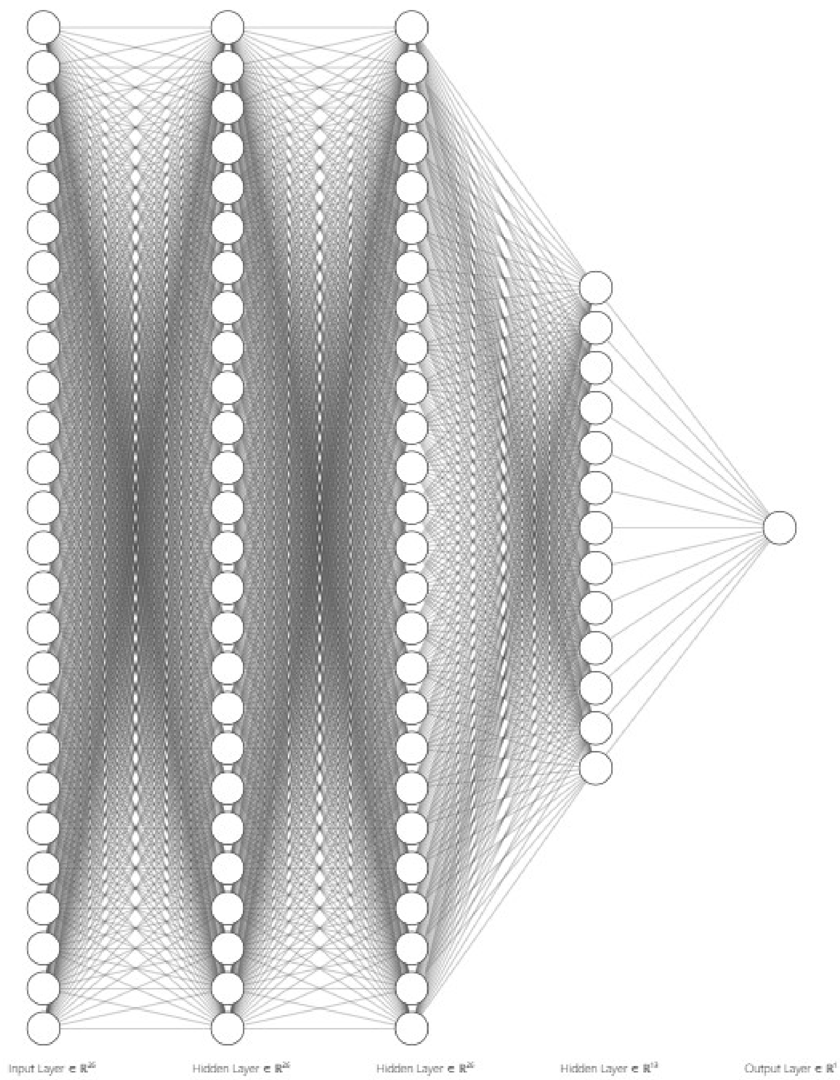

Figure 1: Visual description of Neural Network

	ME	RMSE	MAE	MPE	MAPE
Test set	25.80266	582.3353	306.1574	-1.682175	8.448477

```r
ensemble_summ[,"kNN"] = opt_knn_rescaled_prices
ensemble_summ[,"NeuralNets"] = y_valid_pred_2_26GLR

#Calculate Average
ensemble_summ$AveragePred <- rowMeans(ensemble_summ[,1:4])
ensemble_summ$RealPrices <- real_prices

ens_error = accuracy(ensemble_summ$AveragePred, ensemble_summ$RealPrices)

kable_styling(kable(ens_error)%>% kable_classic(), full_width = TRUE)
```

```
ens_rmse_summ <- NN_res[0,] #NeuralNet Error
ens_rmse_summ[1, ] <- round(Acc4, 2)
ens_rmse_summ[2, ] <- Tree_res[8,]
ens_rmse_summ[3, ] <- round(opt_accuracy, 2)
ens_rmse_summ[4, ] <- NN_res[6,]
ens_rmse_summ[5, ] <- round(ens_error, 2)

row.names(ens_rmse_summ) <- c("Multiple Linear Regression",
                              "Regression Tree",
                              "k-Nearest Neighbor",
                              "Neural Network",
                              "Ensemble")
#display results
kable_styling(kable(ens_rmse_summ)%>% kable_classic(), full_width = FALSE)
```

Model Performance Summary

```r
predtable_lr <- data.frame("Real" = real_prices,
                    "Predicted" = ensemble_summ[, 1],
                    "Color" = diamonds[valid.index, 3],
                    "Cut" = diamonds[valid.index, 2],
                    "Clarity" = diamonds[valid.index, 4])
predtable_rg <- data.frame("Real" = real_prices,
                    "Predicted" = ensemble_summ[, 2],
                    "Color" = diamonds[valid.index, 3],
                    "Cut" = diamonds[valid.index, 2],
                    "Clarity" = diamonds[valid.index, 4])
predtable_knn <- data.frame("Real" = real_prices,
                    "Predicted" = ensemble_summ[, 3],
                    "Color" = diamonds[valid.index, 3],
                    "Cut" = diamonds[valid.index, 2],
                    "Clarity" = diamonds[valid.index, 4])
predtable_nn <- data.frame("Real" = real_prices,
                    "Predicted" = ensemble_summ[, 4],
                    "Color" = diamonds[valid.index, 3],
                    "Cut" = diamonds[valid.index, 2],
                    "Clarity" = diamonds[valid.index, 4])
```

```r
predtable_ens <- data.frame("Real" = real_prices,
                    "Predicted" = ensemble_summ[, 5],
                    "Color" = diamonds[valid.index, 3],
                    "Cut" = diamonds[valid.index, 2],
                    "Clarity" = diamonds[valid.index, 4])


pred_plot_lr <- ggplot(data = predtable_lr, aes(x= Real, y = Predicted, color = clarity))+
  geom_point(show.title = FALSE, size = 0.5)+
  scale_color_brewer(type = 'div', guide = guide_legend(reverse = T, override.aes = list(alpha = 1, size
  labs(x= "Multiple Linear Regression", y = "Predicted Values")+
  theme(legend.key.size = unit(0.5, 'cm'))+
  theme_classic()

pred_plot_rg <- ggplot(data = predtable_rg, aes(x= Real, y = Predicted, color = clarity))+
  geom_point(size = 0.5)+
  scale_color_brewer(type = 'div', guide = guide_legend(reverse = T, override.aes = list(alpha = 1, size
  labs(x= "NN (Regression Trees", y = "")+
  theme_classic()

pred_plot_knn <- ggplot(data = predtable_knn, aes(x= Real, y = Predicted, color = clarity))+
  geom_point(size = 0.5)+
  scale_color_brewer(type = 'div', guide = guide_legend(reverse = T, override.aes = list(alpha = 1, size
  labs(x= "k Neares Neighbor", y = "")+
  theme_classic()

pred_plot_nn <- ggplot(data = predtable_nn, aes(x= Real, y = Predicted, color = clarity))+
  geom_point(size = 0.5)+
  scale_color_brewer(type = 'div', guide = guide_legend(reverse = T, override.aes = list(alpha = 1, size
  labs(x= "Neural Networks", y = "")+
  theme_classic()


pred_plot_ens <- ggplot(data = predtable_ens, aes(x= Real, y = Predicted, color = clarity))+
  geom_point(size = 0.5)+
  scale_color_brewer(type = 'div', guide = guide_legend(reverse = T, override.aes = list(alpha = 1, size
  labs(x= "Ensemble", y = "")+
  theme_classic()

#pred_plot_full <- grid.arrange(ggplotGrob("Hello"),ggarrange(pred_plot_lr, pred_plot_rg, pred_plot_knn
#pred_plot_full
```

Conclusions