Analysis of Diamond Dataset

GEETA SOWMYA CHETHI [50442722]
AYUSHI DAKSH [50485625]
RAM KASHYAP CHERUKUMILLI [50468970]
SHIVANI BETHI [50442852]

Github Repository: https://github.com/raamkashyap/Analysis-Of-Diamond-Dataset.git

1. Introduction

1.1 Abstract

The history of the diamond industry from a political and social perspective is fascinating. By studying the characteristics of diamonds that are highly valued, we could gain insights into what factors make a diamond more or less valuable. Without considering the peculiarities of the diamonds, even an expert cannot incorporate as much price information as a picture of the entire market. We performed a statistical data analysis (outlier detection, pairwise association and PCA) and used different regression techniques (linear regression, penalized linear regression - ridge and lasso, regression tree and random forest) to predict the diamond price. In our analysis, we also identified the factors responsible for driving the price of a diamond. Among the techniques that we explored, we found that Random Forest performed the best for this task.

1.2 Dataset Description

We have used the Diamond dataset from openML repository
https://www.openml.org/search?type=data&sort=runs&id=42225&status=active

The dataset has approximately 54K observations and 10 primary variables - carat, cut, color, clarity, depth, table, price, x (length in mm), y (width in mm), and z. (depth in mm).

Detailed description of the variables-

Price	Price of the diamond in US dollars.
Carat	Weight of the diamond.
Cut	Quality of cut.
Color	Color of Diamond.
Clarity	How Clear the diamond is.
x	Length of the diamond in mm.
y	Width of the diamond in mm.
z	Depth of the diamond in mm.
depth	Total depth percentage of the diamond.(depth percantage = z/mean(x,y))
table	width of top to relatively widest point on the diamond.

There are 9 predictors-

- Carat, depth, table, x, y, z are numerical variables
- Cut, color clarity are categorical variables

Response variable is price.

2. Materials and Methods

2.1 Data exploration and pre-processing

Load libraries and set seed

We load all the libraries necessary to execute our code and set seed for reproducible results.

```
library(tidyverse)
library(rpart)
library(dplyr)
library(rpart.plot)
library(ISLR2)
library(plyr)
library(ggplot2)
library(GGally)
library(leaps)
library(randomForest)
library(Metrics)
library(caret)
library(leaps)
library(glmnet)
library(glmnetUtils)
library(pls)
library(boot)
library(corrplot)

set.seed(2)
```

Load Dataset

```
data(diamonds)

# view dataset summary
summary(diamonds)
```

```
##      carat               cut        color        clarity          depth      
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good     : 4906   E: 9797   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194   Median :61.80  
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171   Mean   :61.75  
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50  
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00  
##                                     J: 2808   (Other): 2531                  
##      table           price             x                y         
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720  
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710  
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735  
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540  
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900  
##                                                                   
##        z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

```
# view a few rows of the data
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Numerical Columns: Carat, x, y, z, depth and table are numerical columns.

```
num_cols <- select_if(diamonds, is.numeric)
colnames(num_cols)
```

```
## [1] "carat" "depth" "table" "price" "x"     "y"     "z"
```

Categorical Columns: Cut, color and clarity are categorical columns.

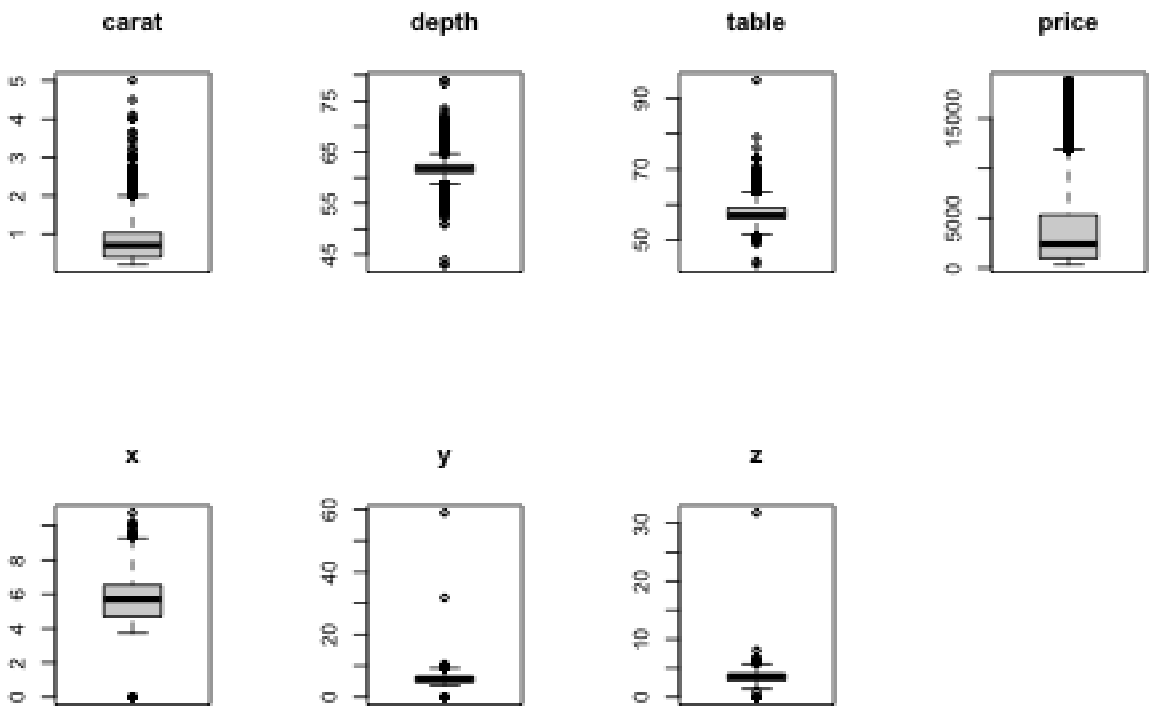
```
cat_col <- select_if(diamonds, negate(is.numeric))
colnames(cat_col)
```

```
## [1] "cut"     "color"   "clarity"
```

Outlier Detection and Filtering

The boxplot() method helps us visually identify if outliers exist in the dataset for each variable. This method defined the outliers as points which are outside the [2.5%, 97.5%] confidence interval. We perform outlier detection and filtering on numerical variables only.

```
par(mfrow=c(2,4))
for (i in colnames(num_cols)) {
  boxplot(diamonds[,i], main = i)
}
```



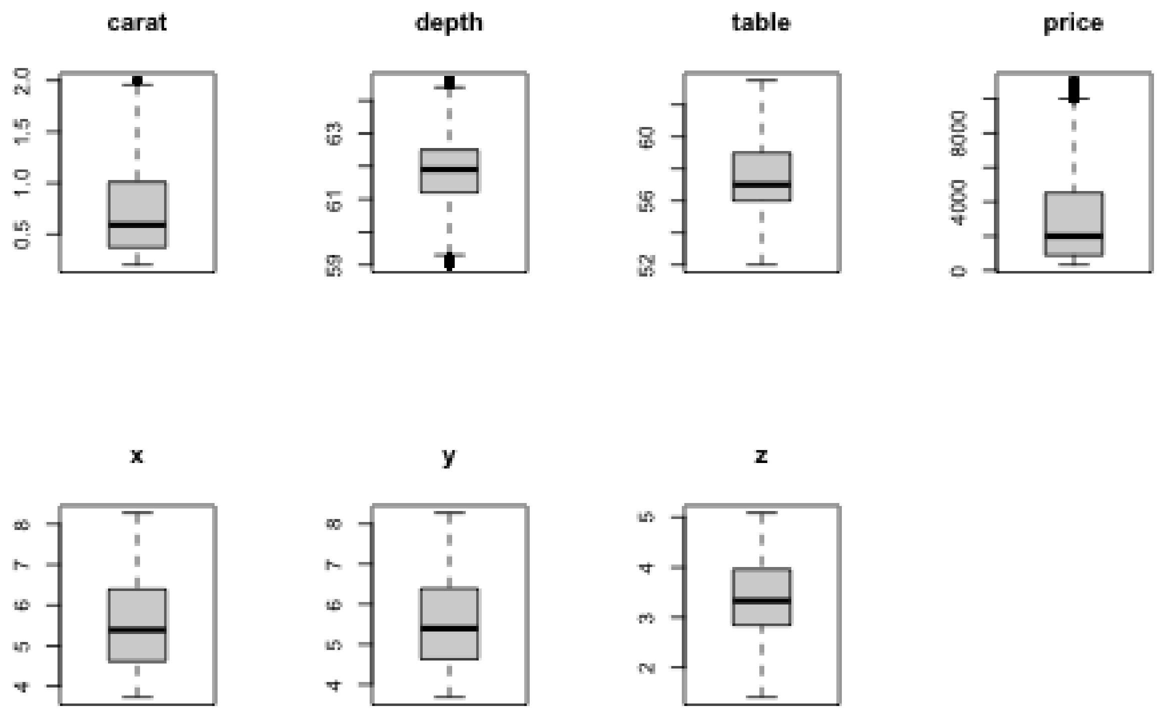
We observe that all the variables have some outliers which need to be filtered out.

For filtering outliers we use boxplot()$out which gives us the outliers for a particular column. Once we get those values, we remove them from the dataset.

```
outliers.carat <- boxplot(diamonds$carat, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$carat %in% outliers.carat),]
outliers.depth <- boxplot(diamonds$depth, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$depth %in% outliers.depth),]
outliers.table <- boxplot(diamonds$table, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$table %in% outliers.table),]
outliers.price <- boxplot(diamonds$price, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$price %in% outliers.price),]
outliers.x <- boxplot(diamonds$x, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$x %in% outliers.x),]
outliers.y <- boxplot(diamonds$y, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$y %in% outliers.y),]
outliers.z <- boxplot(diamonds$z, plot=FALSE)$out
diamonds <- diamonds[-which(diamonds$z %in% outliers.z),]
```

After outlier removal, we get the following boxplots for all columns-

```
par(mfrow=c(2,4))
for (i in colnames(num_cols)) {
  boxplot(diamonds[,i], main = i)
}
```



After outlier removal, we have retained approximately 47000 records.

Data Encoding

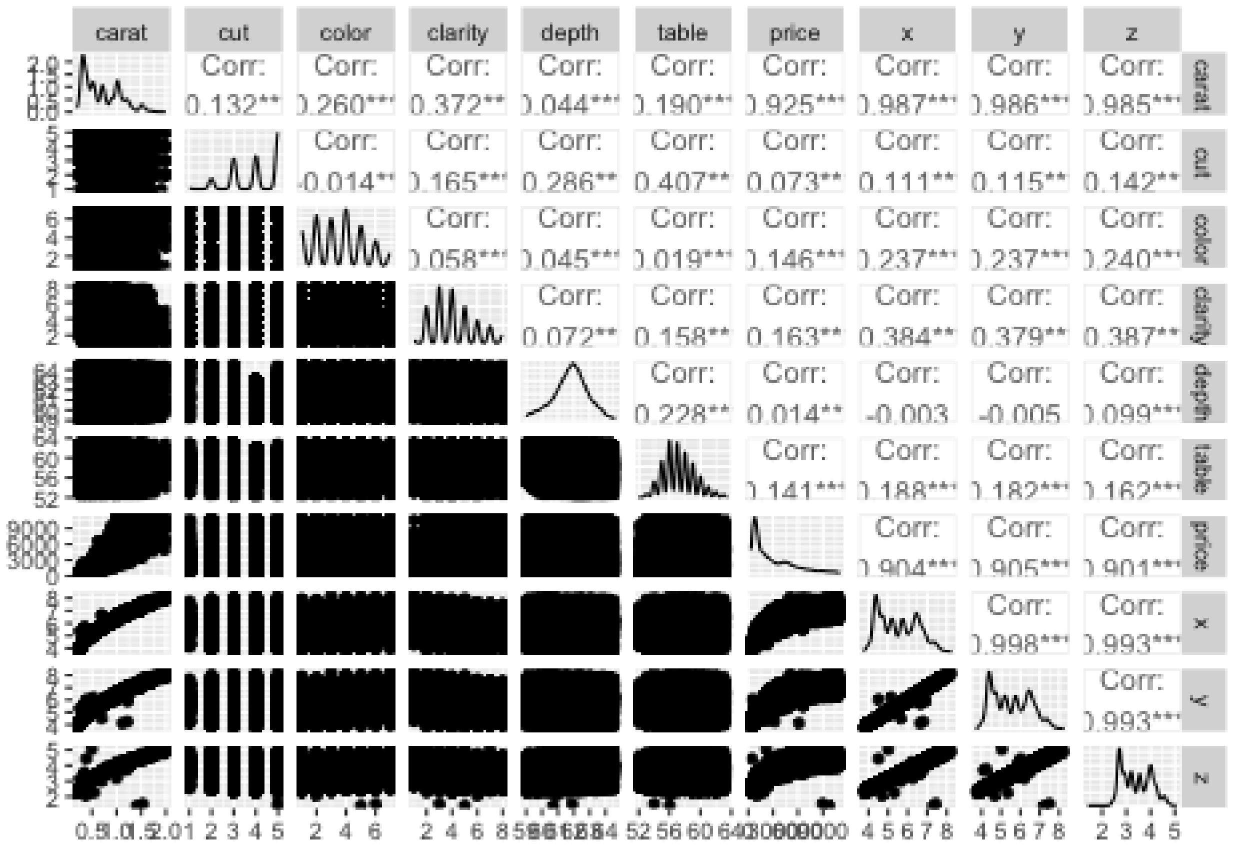
Convert categorical variables to numerical encoding.

```
diamonds$cut = as.numeric(unclass(diamonds$cut))
diamonds$clarity = as.numeric(unclass(diamonds$clarity))
diamonds$color = as.numeric(unclass(diamonds$color))
```

Pairwise Association

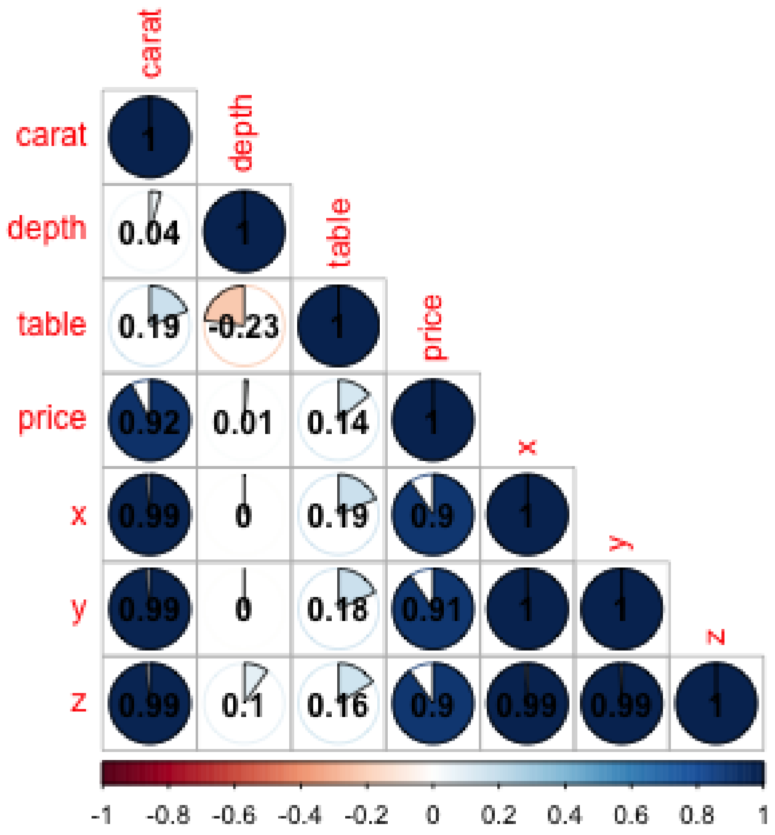
Pairwise association is a statistical technique that is used to examine the relationship between the variables by looking at each possible pair of variables and calculating the correlations between them. This helps us identify any variables that are highly correlated and might be influencing the relationship between the other variables. This can help us control for confounding variables and improve the accuracy and reliability of our analysis.

```
ggpairs(diamonds, progress = FALSE)
```



By looking at the association between variables, we can identify that carat is highly correlated with x, y, z. Thus, we can safely remove x, y, z from the dataset without losing any relevant information. Reducing the number of predictors reduces the complexity of the prediction task and can also improve the accuracy and reliability of the prediction algorithm.
The correlations for numerical columns can also be plotted as pie charts for more visually intuitive understanding.

```
cor_mat <- cor(diamonds[, -c(2:4)])
corrplot(cor_mat, method="pie", type="lower", addCoef.col = "black")
```

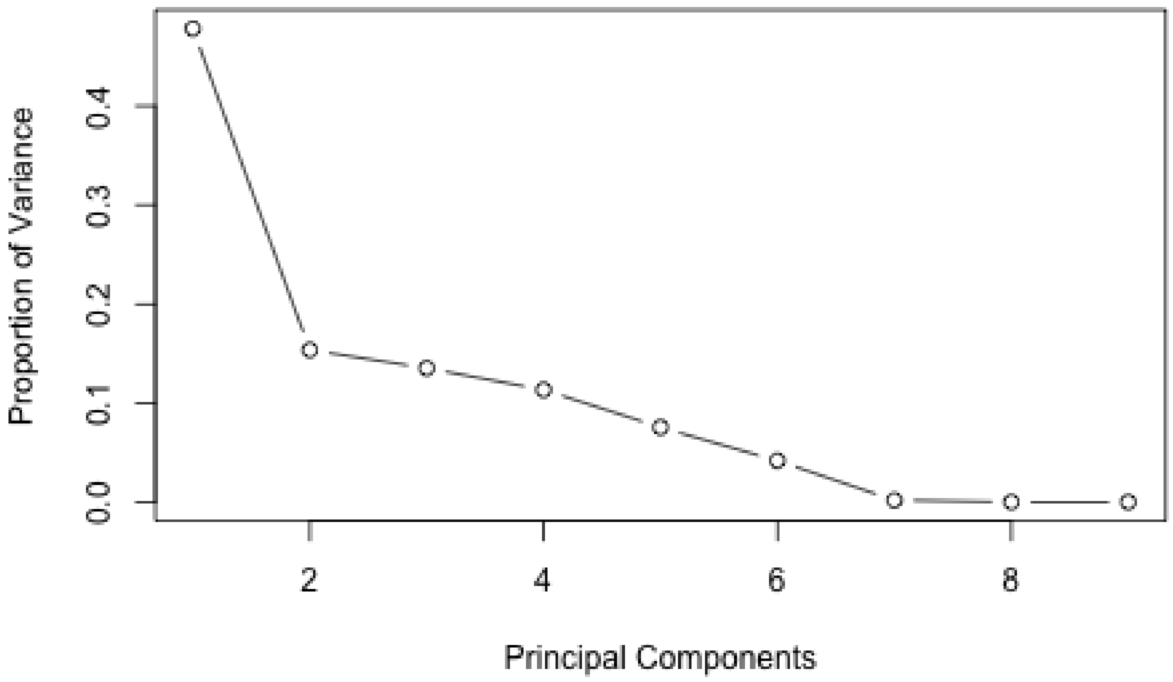


Principal Component Analysis

Principal component analysis (PCA) is an unsupervised statistical technique used to reduce the dimensionality of a dataset while preserving the maximum variance. It uses singular value decomposition to identify a suitable sub-space in the high dimensional dataset.

We are computing PCA to verify the results we obtained from pairwise association and to reduce the number of predictors.

```
pr.out <- prcomp(diamonds %>% select (-price), scale = TRUE)
std_dev <- pr.out$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
plot(prop_varex, xlab = "Principal Components",
     ylab = "Proportion of Variance", type = "b")
```



From the scree plot it is visible that at-least 6 components are necessary to represent 98% variance in the data.

Since depth already captures information from x, y, z, these four variables- depth, x, y, z have colinearity. PCA takes advantage of multi-colinearity and combines the highly correlated variables into a set of uncorrelated variables. Therefore, PCA can effectively eliminate multi-colinearity between features. This reduces the training time since only few weights have to be found given that feature vectors have been decreased.

These results are also inline with the pairwise association results, which also reduced the dimensionality by 3.

From the above results, we finally drop x, y, z variables from the dataset. Now we have 6 predictors in the dataset- carat, color, cut, clarity, depth, and table.

```
diamonds <- diamonds %>%
  select (-x, -y, -z)
```

Data Split

We split the data into 70% train and 30% test samples.

```
partition = sample(nrow(diamonds), as.integer(nrow(diamonds)*0.7))
train_data <- diamonds[partition,]
test_data <- diamonds[-partition,]
```

Data Normalization

We normalize the data to center it about the origin to have a unit variance across all features. This is needed for better performance and stability of the prediction algorithms.

```
# normalize/standardize the data
mean_vector = apply(train_data %>% select (-price), MARGIN=2, mean)
sd_vector = apply(train_data %>% select (-price), MARGIN=2, sd)
train_data[c(-7)] <- scale(train_data[c(-7)], center=mean_vector, scale=sd_vector)
```

```
test_data[c(-7)] <- scale(test_data[c(-7)], center=mean_vector, scale=sd_vector)

# Separate out predictors and response
train_X <- train_data %>%
  select(-price)
train_Y <- train_data$price
test_X <- test_data %>%
  select(-price)
test_Y <- test_data$price
```

2.2 Prediction Algorithms

We have trained different regression algorithms like Linear Regression and Penalized Linear Regression models-Ridge and Lasso, Regression Trees and Random Forest to predict the price of a diamond.

We have used cross validation for tuning of hyper-parameters (regularization weight, depth of decision tree etc.). Mean Squared Error (MSE) is used as a metric for evaluation on test data. Stability Estimation has been done using Bootstrap.

2.2.1 Linear Regression

Linear regression analysis is used to predict the value of a variable based on the value of another variable. It is of the form y = ax +b , where y is the response variable and x is the predictor variable. a and b are constants which are called the coefficients. Using R's lm() utilities, we construct a relationship model. The model's coefficients are calculated, then we use them to construct the mathematical equation. As a performance metric we find out the mean square error.

```
# Applying linear regression model using price as the target
# and other features as the feature space
lin_reg_obj <- lm(train_Y ~ ., data = train_X)
# Compute the predictions
predict_price_lin_reg <- predict(lin_reg_obj, test_X)
#Print MAE
print(mae(test_Y,predict_price_lin_reg))
```

```
## [1] 561.7629
```

```
# Print MSE
postResample(predict_price_lin_reg,test_Y)['RMSE']^2
```

```
##     RMSE
## 593280.4
```

```
# Calculating the R-squared value
postResample(predict_price_lin_reg,test_Y)['Rsquared']
```

```
##  Rsquared
## 0.9121809
```

2.2.2 Penalized Linear Regression - Ridge

Ridge regression is a shrinkage method; it shrinks the coefficients of variables having minor contribution to the outcome close to 0. This shrinkage is achieved by penalizing the model with an L2-norm penalty term (sum of squared coefficients). The amount of penalty is tuned using constant term λ. The best value of λ is the one that minimizes the cross-validation prediction error. We compute best λ using cv.glmnet() function that internally performs k-fold cross-validation. After getting the best λ, we fit a ridge regression model using that λ value and evaluate on the test set.

```r
# Find best lambda using cross validation
cv.ridge <- cv.glmnet(as.matrix(train_X), train_Y, alpha = 0)
best.lambda.ridge <- cv.ridge$lambda.1se
print(paste("Best lambda: ", best.lambda.ridge))
```

```
## [1] "Best lambda:  240.207095302286"
```

```r
# Fit a model using best lambda value
ridge.model <- glmnet(as.matrix(train_X), train_Y,
                      alpha = 0, lambda = best.lambda.ridge)

# Predictor importance
coef(ridge.model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 2999.473351
## carat       2417.869601
## cut           43.392618
## color       -250.570431
## clarity      442.393856
## depth         -9.379870
## table         -3.515143
```

```r
# Prediction on train data
ridge.train.pred <- predict(ridge.model, as.matrix(train_X))
print(paste("MSE on train data: ", mean((ridge.train.pred - train_Y)^2)))
```

```
## [1] "MSE on train data:  677572.202334575"
```

```r
# Prediction on test data
ridge.test.pred <- predict(ridge.model, as.matrix(test_X))
print(paste("MSE on test data: ", mean((ridge.test.pred - test_Y)^2)))
```

```
## [1] "MSE on test data:  668929.132491034"
```

2.2.3 Penalized Linear Regression - LASSO

LASSO stands for Least Absolute Shrinkage and Selection Operator. It is also a shrinkage method. The shrinkage is achieved by penalizing the model with an L1-norm penalty term (sum of absolute coefficients). The amount of penalty is tuned using constant term λ. Sometimes coefficients are shrunken exactly to 0, this acts as subset selection for reducing predictors to reduce model complexity. The best value of λ is the one that minimizes the cross-validation prediction error. We compute best λ using cv.glmnet() function that internally performs k-fold cross-validation. After getting the best λ, we fit a LASSO regression model using that λ value and evaluate on the test set.

```r
# Find best lambda using cross validation
cv.lasso <- cv.glmnet(as.matrix(train_X), train_Y, alpha = 1)
best.lambda.lasso <- cv.lasso$lambda.1se
print(paste("Best lambda: ", best.lambda.lasso))
```

```
## [1] "Best lambda:  27.6179788167974"
```

```r
# Fit a model using best lambda value
lasso.model <- glmnet(as.matrix(train_X), train_Y,
                      alpha = 1, lambda = best.lambda.lasso)
```

```
# Predictor importance
coef(lasso.model)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 2999.47335
## carat       2674.50179
## cut           36.31569
## color       -320.94335
## clarity      557.13176
## depth          .
## table         -6.40503
```

```
# prediction on train data
lasso.train.pred <- predict(lasso.model, as.matrix(train_X))
print(paste("MSE on train data: ", mean((lasso.train.pred - train_Y)^2)))
```

```
## [1] "MSE on train data:  601991.021954542"
```

```
#prediction on test data
lasso.test.pred <- predict(lasso.model, as.matrix(test_X))
print(paste("MSE on test data: ", mean((lasso.test.pred - test_Y)^2)))
```

```
## [1] "MSE on test data:  597434.903144139"
```

2.2.4 Regression Tree

Regression tree is a type of decision tree used for regression problems. To build a regression tree, the algorithm recursively splits the dataset based on individual features until the leaves of the tree are reached. We fit a regression tree using rpart() with method = "anova". From the fitted model, get the best complexity parameter (cp) value. Best cp is the one with least cross validation error (xerror). Then we use this cp to get a pruned tree with less number of predictors than the full tree. Pruning is done to avoid overfitting. Sometimes pruned tree is same as full tree with all predictors.

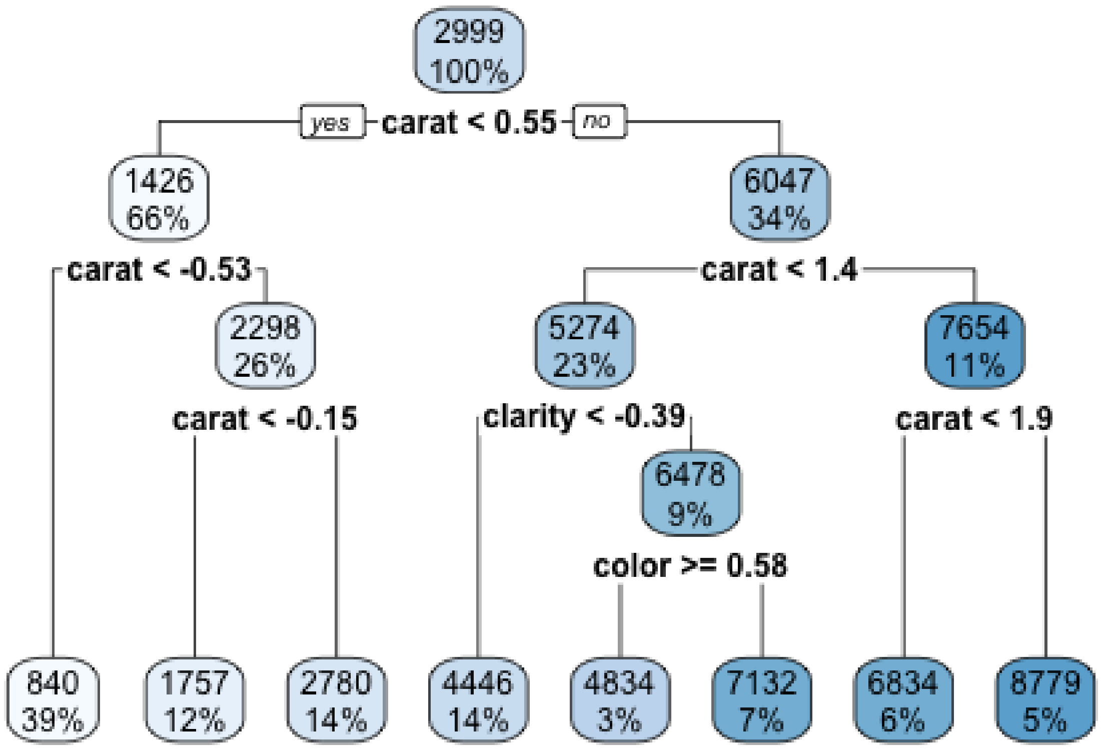
```
fit.tree <- rpart(train_Y ~ ., data = train_X)

# cp table for model
fit.tree$cptable
```

```
##           CP nsplit rel error    xerror        xstd
## 1 0.71027869      0 1.0000000 1.0000475 0.008671636
## 2 0.06266607      1 0.2897213 0.2899686 0.002930171
## 3 0.04993119      2 0.2270552 0.2280978 0.002431176
## 4 0.03393171      3 0.1771241 0.1784367 0.002385631
## 5 0.01511627      4 0.1431923 0.1435604 0.001811502
## 6 0.01492336      5 0.1280761 0.1282948 0.001672948
## 7 0.01023770      6 0.1131527 0.1134395 0.001485584
## 8 0.01000000      7 0.1029150 0.1070337 0.001467921
```

```
# Best cp with minimum xerror
best.cp <- fit.tree$cptable %>%
  as_tibble() %>%
  filter(xerror == min(xerror)) %>%
  head(1) %>%
  pull(CP)
print(paste("Best cp for prune tree ", best.cp))
```

```
## [1] "Best cp for prune tree  0.01"
```

```
# Prune tree with best cp
prune.tree <- prune(fit.tree, cp = best.cp)
rpart.plot(prune.tree)
```



```
# prediction on train data
prune.train.pred <- predict(
  prune.tree,
  train_X
)

#train error
print(paste("MSE on train data: ", mean((prune.train.pred - train_Y)^2)))
```

```
## [1] "MSE on train data:  694893.560620125"
```

```
# prediction on test data
prune.test.pred <- predict(
  prune.tree,
  test_X
)

#test error
print(paste("MSE on test data: ", mean((prune.test.pred - test_Y)^2)))
```

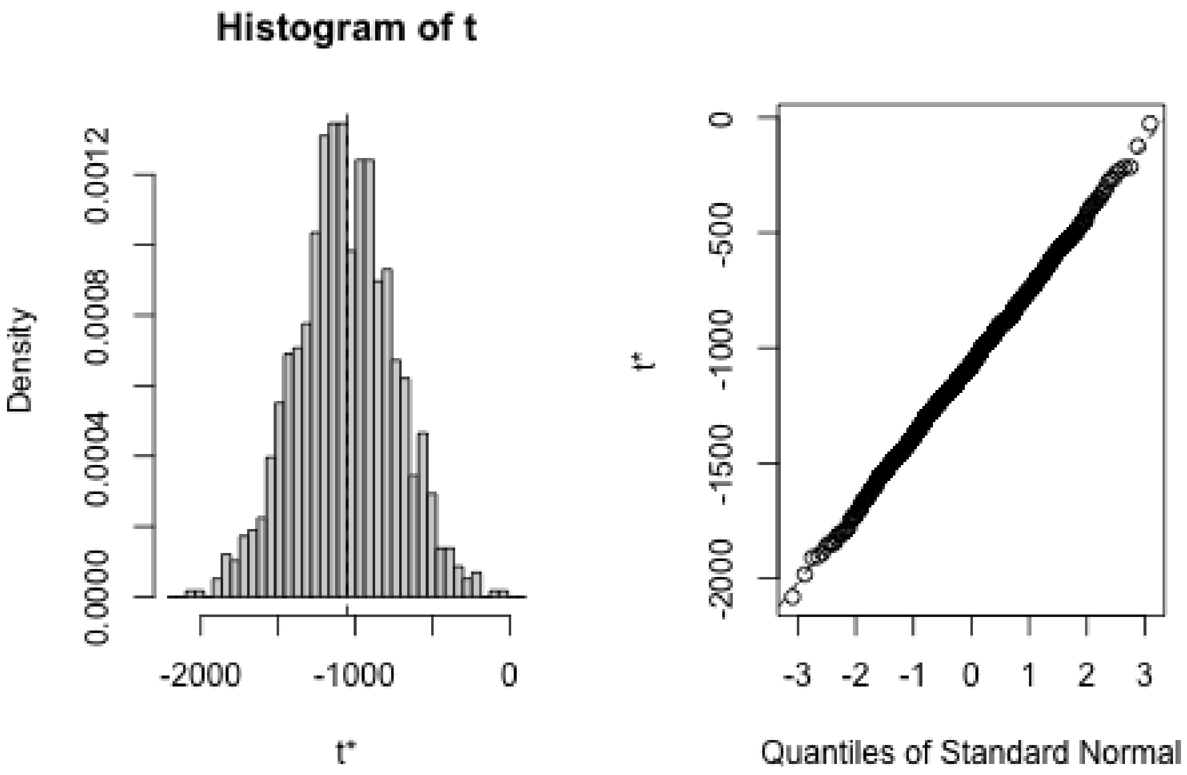
```
## [1] "MSE on test data:  692470.446771889"
```

2.2.5 Bootstrap

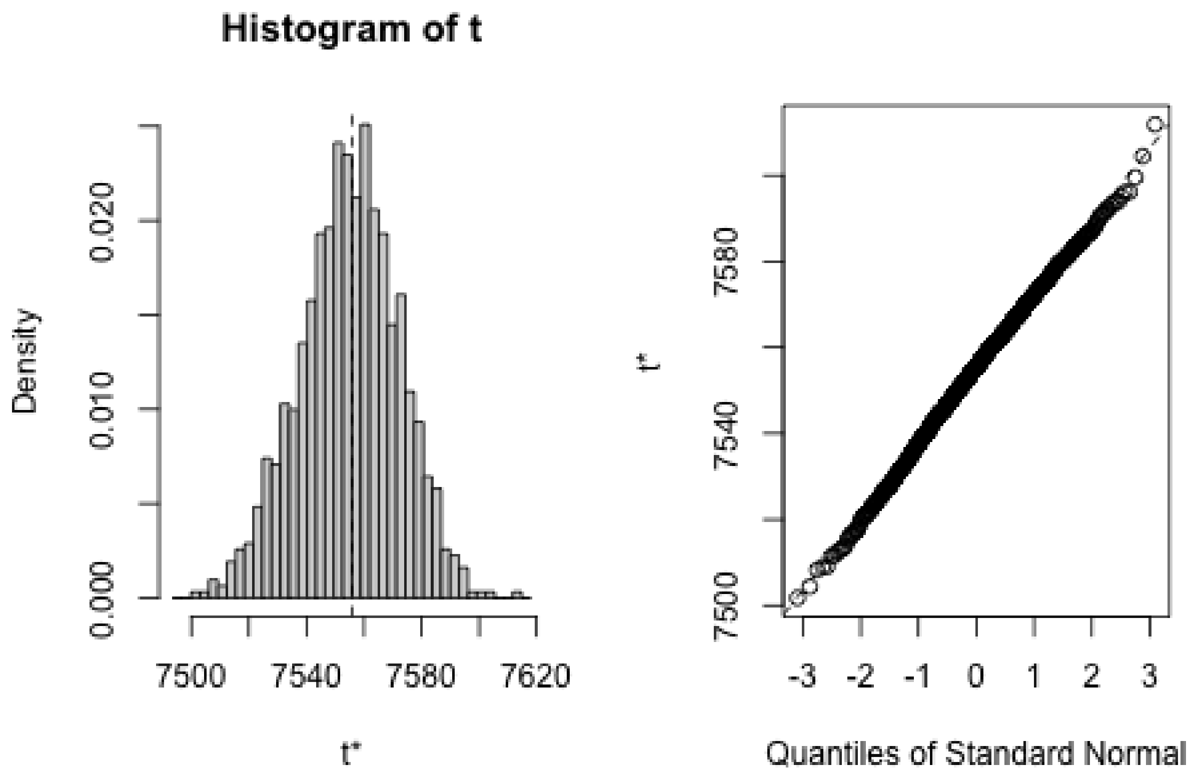
Bootstrapping is a re-sampling approach used in statistics and machine learning that involves periodically taking samples with replacement from our source data, frequently in order to estimate a population parameter.

The phrase “with replacement” refers to the possibility of numerous occurrences of the same data point in our re-sampled dataset.

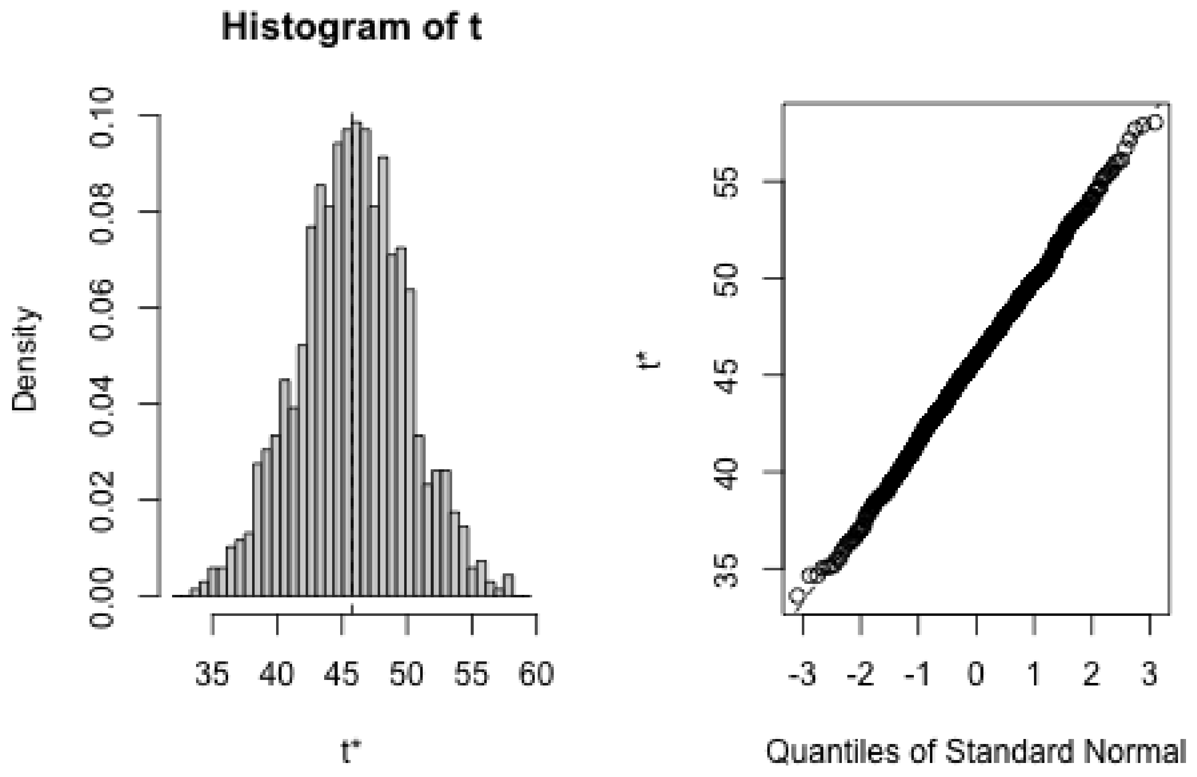
```
# Applying bootstrapping using multiple linear regression
# Creating a function to caluculate the coeffecients that have been fitted
bootstrap_func <- function(formula, data_frame, obs)
{
  bootstrap_sample <-  data_frame[obs, ] # Select a sample using bootstrap
  # Applying linear regression on the bootstrap sample
  lin_reg_model <- lm(formula, data = bootstrap_sample)
  return(coef(lin_reg_model)) # The function should return the weights
}
# Apply bootstrap for 1001 samples
bootstrap_final <- boot(data = diamonds, statistic=bootstrap_func, R = 1001, formula = price ~ .)
# Plot the distribution of the bootstrapped samples
plot(bootstrap_final,index = 1) # Intercept of the model
```



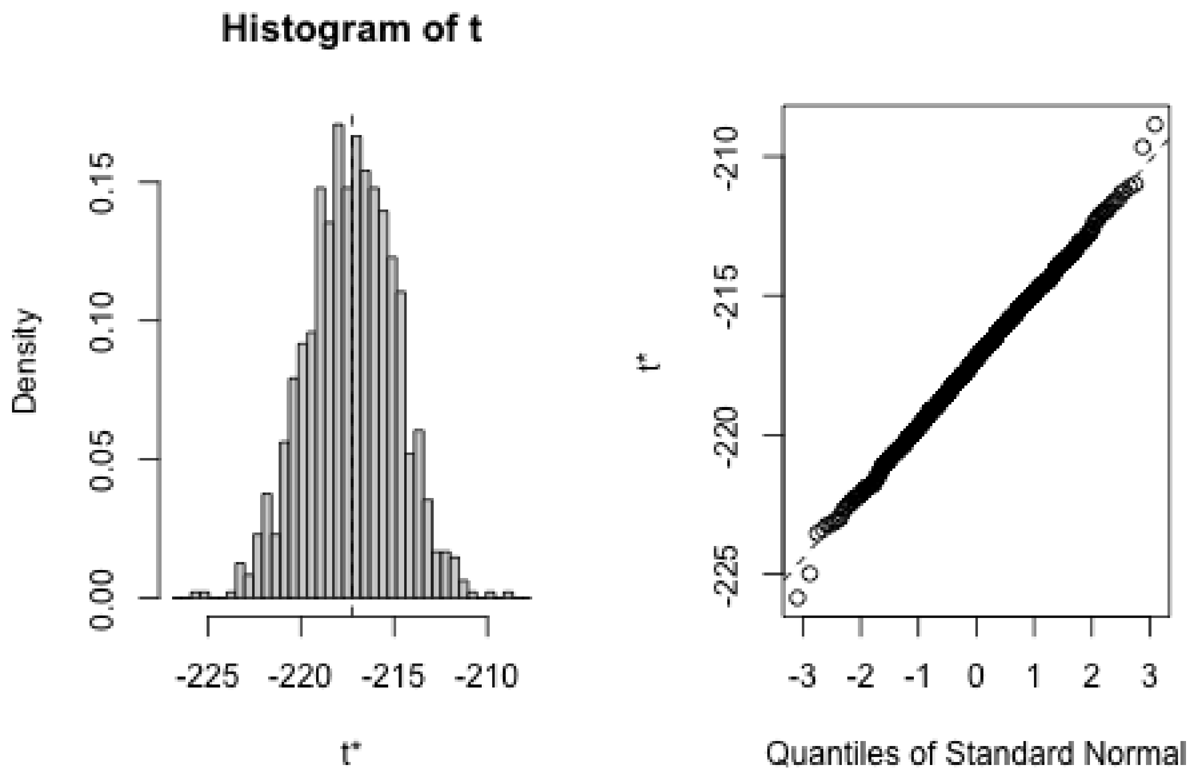
```
plot(bootstrap_final,index = 2) # Carat predictor variable
```



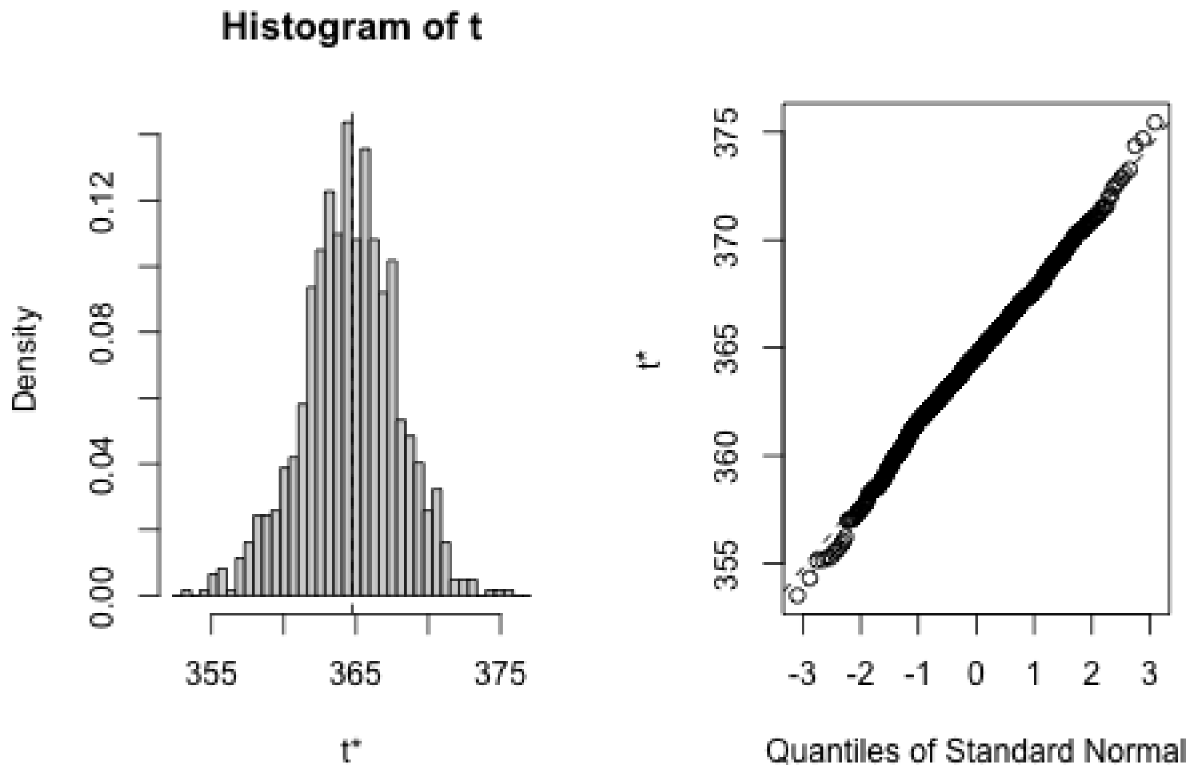
```
plot(bootstrap_final,index = 3) # Cut predictor variable
```



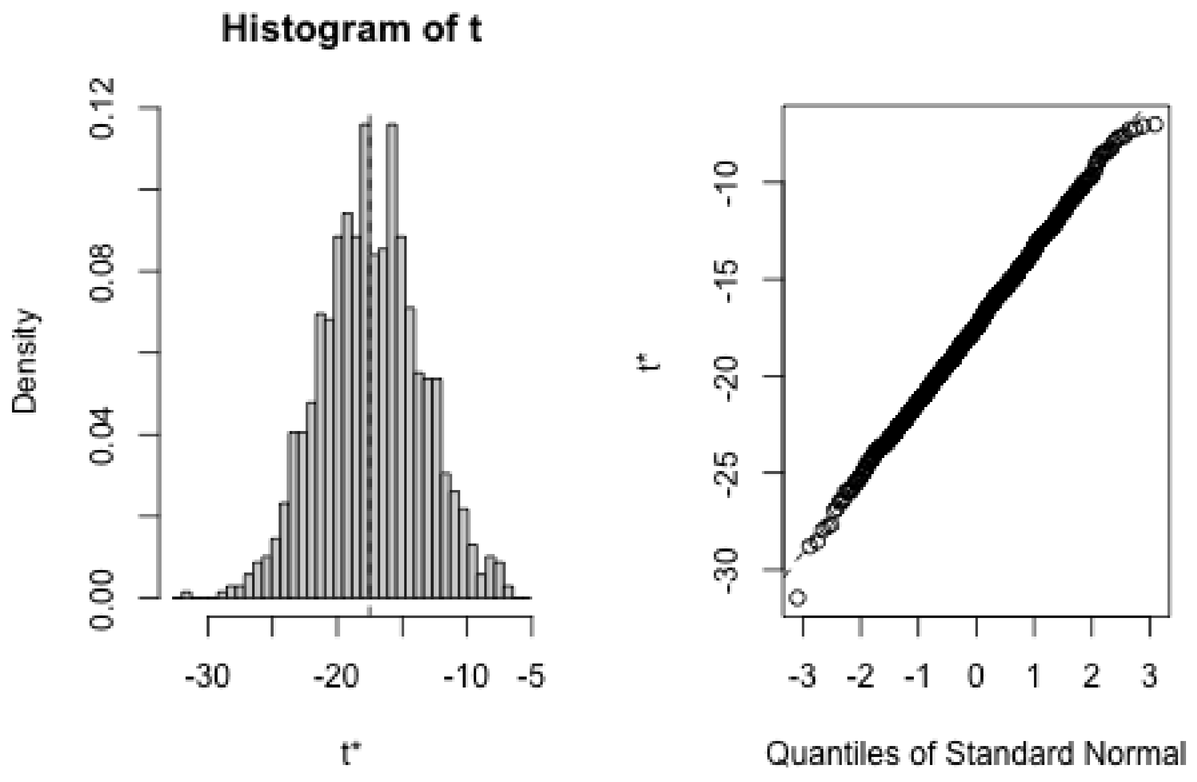
```
plot(bootstrap_final,index = 4) # Colour predictor variable
plot(bootstrap_final,index = 4) # Clarity predictor variable
```



```r
plot(bootstrap_final,index = 5) # Depth predictor variable
```



```r
plot(bootstrap_final,index = 6) # Table predictor variable
```



2.2.6 Random Forest Regression

A random forest is a machine learning method for tackling classification and regression issues. It makes use of ensemble learning, a method for solving complicated issues by combining a number of classifiers. In a random forest algorithm, there are many different decision trees. The random forest algorithm creates a "forest" that is trained via bagging or bootstrap aggregation. The accuracy of machine learning algorithms is increased by bagging, an ensemble meta-algorithm. Based on the predictions of the decision trees, the (random forest) algorithm determines the result. It makes predictions by averaging or averaging out the results from different trees. The accuracy of the result grows as the number of trees increases. The decision tree algorithm's shortcomings are eliminated with a random forest.

```
#Applying Random Forest algorithm
#install.packages("randomForest")
random_forest_model <- randomForest(x = train_X, y = train_Y, ntree = 100)
# Predicting on test set
predict_rf <- predict(random_forest_model, test_X)
#Displaying the testing error
#install.packages('Metrics')
# Calculating the mean absolute error
print(mae(test_Y,predict_rf))
```

```
## [1] 220.3869
```

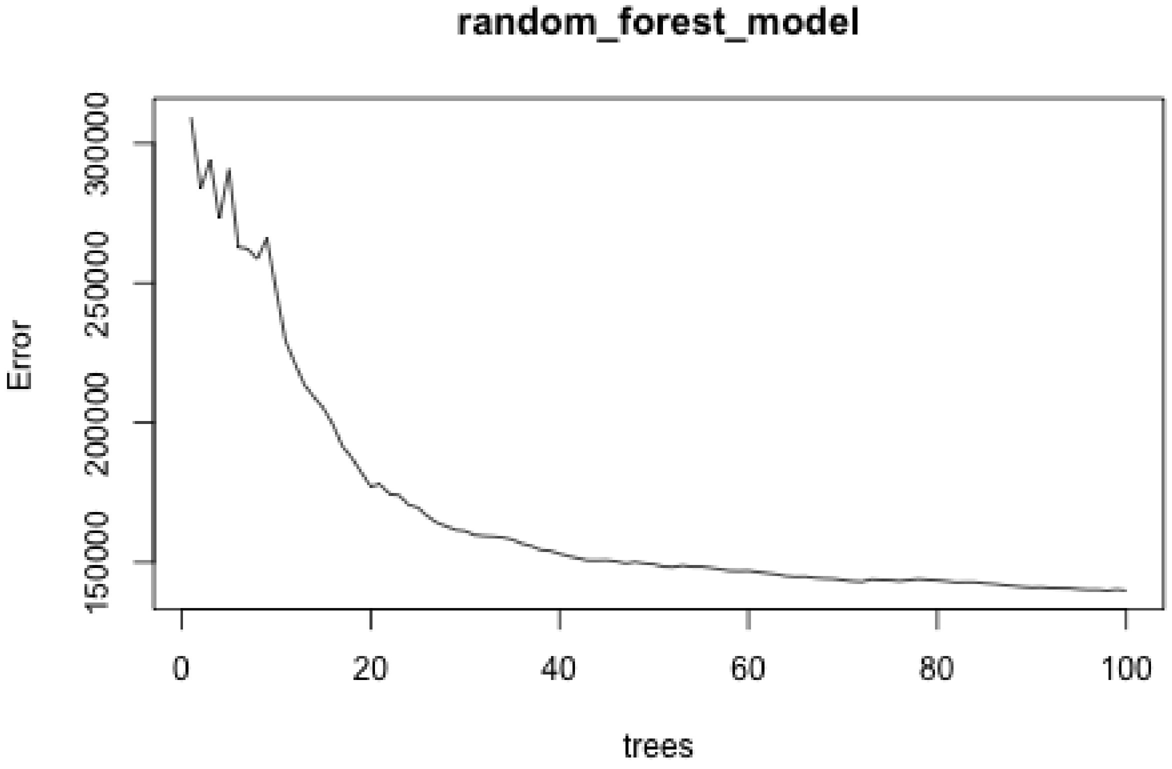
```
# Calculating the  mean square error
postResample(predict_rf,test_Y)['RMSE']^2
```

```
##     RMSE
## 139501.5
```

```
# Calculating the R^2 value
postResample(predict_rf,test_Y)['Rsquared']
```

```
##  Rsquared
## 0.9799271
```

```
plot(random_forest_model)
```



3. Results

A summary of the train and test MSEs for the algorithms used above-

Model	Train MSE	Test MSE
Linear Regression	596245.3	593280.4
Ridge Regression	677572.2	668929.1
LASSO	601991.0	597434.9
Regression Tree	694893.6	692470.4
Random Forest	140689.1	140845.7

From comparison of different models, we observe that Regression Tree performs the worst among all models as it has the maximum train and test MSE which looks right since single tree model are usually highly unstable and a poor predictor. Bagging regression tree should make the prediction more effective.

We also observe that Linear Regression with no penalization performs better than linear models with penalty – ridge and LASSO. This indicates that ridge and LASSO penality are causing the model to underfit the data.

Finally, Random Forest has the least Train and Test MSE and therefore we can conclude that it is the best model. This algorithm performs best because random forests are more flexible than linear models and can model non-linear relationships between predictors and response. Also, random forest avoids overfitting which is a problem with regression tree.

4. References

- https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
- https://www.ijrte.org/wp-content/uploads/papers/v7i6s2/F11170476S219.pdf
- https://bookdown.org/yih_huynh/Guide-to-R-Book/diamonds.html
- https://medium.com/swlh/exploratory-data-analysis-21bbf3887e28
- https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why

5. Contributions

- RAM KASHYAP CHERUKUMILLI : Data set analysis, Exploratory data analysis, PCA, Presentation slides, Report
- AYUSHI DAKSH : Penalized Linear Regression - Ridge and Lasso Regression, Regression Tree, Presentation slides, Report
- GEETA SOWMYA CHETHI : Bootstrapping, Random Forest regression algorithm, Presentation slides, Report
- SHIVANI BETHI : Linear Regression, Presentation slides, Report