Cost Prediction for Diamonds Dataset using Multiple Linear Regression, KNN, Naïve Bayes and SVM models.

Table of Contents

1. Introduction

Diamond trades are going on for a long time, a diamond is purchased from a trader by paying for its worth cost based on the quality of the diamond. The models that I am building is going to be used to predict the cost of the diamond based on the features mentioned in the data set. The data set used here has information related to diamonds. Diamonds are priced based on its clarity, carat, color, etc. The dataset has these attributes as the feature variable. The models built will be predicting the values of the cost variable. This can be used by the diamond traders and the shop owners to price the diamonds to its worth.

2. Data

The dataset used for this project is the diamond dataset. The data can be referred from https://www.kaggle.com/shivam2503/diamonds. The dataset has **53940 rows and 10 variables**. The variables present in the data frame are: Index counter, **carat** (Carat weight of the diamond), **cut** (Describe cut quality of the diamond. Quality in increasing order Fair, Good, Very Good, Premium, Ideal), **color** (Color of the diamond, with D being the best and J the worst), **clarity** (How obvious inclusions are within the diamond:(in order from best to worst, FL = flawless, I3= level 3 inclusions) FL,IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3), **depth** (depth % :The height of a diamond, measured from the culet to the table, divided by its average girdle diameter), **table** (table%: The width of the diamond's table expressed as a percentage of its average diameter), **price** (the price of the diamond), **x** (length mm), **y** (width mm), **z** (depth mm). Datatypes of the variables are as follows:

Cut, color, clarity: ordinal variables (Qualitative datatype).

Id, carat, depth, table, price, x, y, z: Discrete variables (Quantitative datatype).

	carat	cut	color	clarity	depth	table	price	x	y	z
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
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61	338	4	4.05	2.39
11	0.3	Good	J	SI1	64	55	339	4.25	4.28	2.73
12	0.23	Ideal	J	VS1	62.8	56	340	3.93	3.9	2.46
13	0.22	Premium	F	SI1	60.4	61	342	3.88	3.84	2.33
14	0.31	Ideal	J	SI2	62.2	54	344	4.35	4.37	2.71
15	0.2	Premium	E	SI2	60.2	62	345	3.79	3.75	2.27
16	0.32	Premium	E	I1	60.9	58	345	4.38	4.42	2.68
17	0.3	Ideal	I	SI2	62	54	348	4.31	4.34	2.68
18	0.3	Good	J	SI1	63.4	54	351	4.23	4.29	2.7

3. Problems to be Solved

The problem statement is: **To predict and to classify the cost of diamond based on the attribute or the features used to determine its worth.** The better the features of the diamond the more the worth of it. More the degrading the features are the lesser the worth is. In the given dataset we have several attributes that affect the cost of the diamonds, so it is important to find out how much each attribute has a contribution on the changing value of the cost variable and also how the changes in these factors can affect changes in the cost of the diamonds.

4. Solutions

To address the problem mentioned in the problem statement supervised learning models are used, the **predictive model** and the **classification models.** In any supervised learning model, the built model must go through training and testing phase.

The **predictive model** can be used to predict the values of cost variable with changing values of the x variables (attribute values). To build a predictive model we initially need to have a regression model of the data set selected. For building this model we need to preprocess the data variables. Build a multiple regression model, validate the model. To get the best model we need to select those features which have the maximum effect on the price variable. For this predictive model we have one **dependent** variable which will be predicting the values of the cost of the diamond. This variable is the **price** variable. The **independent variables** for this predictive model will be the attributes or the features used by one to determine the worth of the diamond. The independent variables present in the dataset are – **cut, color, clarity, carat, depth, table, price, x, y, z**.

The **classification models** are used to classify the price into labels; Inexpensive or Expensive. There are four models built. To build a classification model there is some basic preprocessing of the data we must do in order to make the data ideal for the model. Every model has a different way of data [reprocessing. The data set is then divided into the training and testing data set (supervised learning). Hold out evaluation is used for splitting the dataset. The model is then built based on the training set. Predictions are checked on the test data to calculate the accuracy. There are different metrics used by different models to calculate the accuracy. There is one **dependent** variable which will be predicting the values of the cost of the diamond, **price**. The **independent variables** for this predictive model will be the attributes or the features used by one to determine the worth of the diamond, **cut, color, clarity, carat, depth, table, price, x, y, z**.

5. Experiments and Results

5.1. Methods and Process

The initial stage of building a model is getting to know the data. **Descriptive analysis** has been performed on the data and the outcomes from that are:

- carat weight of the diamond is in the range 0.2--5.01, skewness = right, Mean = 0.797 and variance = 0.224
- cut quality of the cut, labels = Fair, Good, Very Good, Premium, Ideal, maximum used label = Ideal, least used label = Fair.
- color diamond color from J (worst) to D (best) labels = D, E, F, G, H, I and J, maximum used label = G, least used label = j.
- clarity a measurement of how clear the diamond is, labels = I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1 and IF (best), maximum used label = SI1, least used label = I1.
- x length in mm is in the range 0--10.74, skewness = symmetric, Mean = 5.7311 and variance = 1.25
- y width in mm is in the range 0--58.9, skewness = right, Mean = 5.734 and variance = 1.3
- z depth in mm is in the range 0--31.8, skewness = right, Mean = 3.538 and variance = 0.498
- depth is in the range of 43—79, skewness = symmetric, Mean = 61.749 and variance = 2.05

- table width of top of diamond relative to widest point is in the range 43—95, skewness = right, Mean = 57.45 and variance = 4.99

Now, analyzing the correlation and collinearity between the variables.

```
> cor(num)
            data.carat  data.depth data.table      data.x      data.y      data.z data.price
data.carat  1.00000000  0.02822431  0.1816175  0.97509423  0.95172220 0.95338738  0.9215913
data.depth  0.02822431  1.00000000 -0.2957785 -0.02528925 -0.02934067 0.09492388 -0.0106474
data.table  0.18161755 -0.29577852  1.0000000  0.19534428  0.18376015 0.15092869  0.1271339
data.x      0.97509423 -0.02528925  0.1953443  1.00000000  0.97470148 0.97077180  0.8844352
data.y      0.95172220 -0.02934067  0.1837601  0.97470148  1.00000000 0.95200572  0.8654209
data.z      0.95338738  0.09492388  0.1509287  0.97077180  0.95200572 1.00000000  0.8612494
data.price  0.92159130 -0.01064740  0.1271339  0.88443516  0.86542090 0.86124944  1.0000000
```

From the screenshot provided above it is evident that there is high **collinearity** between carat, x, y and z variable, this makes the variables redundant as any changes made in any one of the variables will get reciprocated in rest of the variables. There is no significant contribution of all the variables, hence it is preferred to remove such redundancy from the model, by keeping only one variable and discarding the rest. While building the MLR model, this collinearity is handled in the postprocessing stage. Whereas in the classification model, the variables have been removed from the dataset to have better prediction will less feature variable.

To check for the **correlation**, we have the following observations. x, y, and z have a very strong relation with price, but depth doesn't have a significant relation with price. Carat has a strong relation with price. Table doesn't have a significant relation with price or any other variable as well. Depth is inversely related to Price. This is because if a Diamond's Depth percentage is too large or small the Diamond will become 'Dark' in appearance because it will no longer return an Attractive amount of light. The Price of the Diamond is highly correlated to Carat, and its Dimensions (x, y, z). The Carat of a diamond has the most significant impact on its Price, the larger a stone is, the rarer it is. The Length(x), Width(y) and Height(z) seems to be highly related to Price and even each other. The correlation of the table and depth variable is too low, in order to increase its transformation of the data was done. This did not affect the correlation value to a great extent hence it was not used.

Continuous

To predict the exact cost value of a diamond we need a predictive model. The model will take the values of the feature variable as an input and give the user the exact price of the diamond. For this we will be developing a Multiple Linear Regression model.

1. Multiple Linear Regression Model

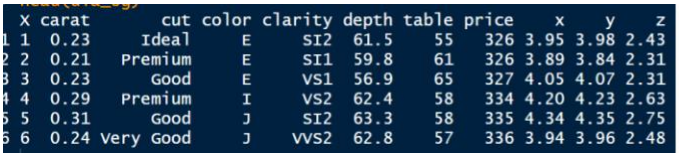
The multiple Linear Regression Model (MLR) requires the data in a preprocessed format. Hence, preprocessing of the data will be the first step.

Data Preprocessing:

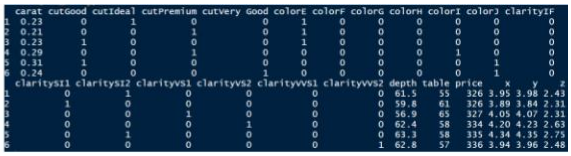
The dataset has one unnamed column which is the serial number for the dataset rows. This information is redundant in building the model; hence the column is removed.

The dataset is then checked for null values and missing values. From the output of R Studio there were no null values or missing values in the data frame.

The most important data preprocessing required for MLR model is removing the categorical variables and replacing them with their dummy variables. There are three dummy variables in the dataset: cut, color and clarity; these variables are converted into their dummy variables. One column of dummy variable is dropped from each categorical variable, as for N factors we only need to create N-1 dummy variables, these variables indirectly tell us the value if the dropped dummy variable. The following screenshots show the data before preprocessing and once preprocessing is done.



	X	carat	cut	color	clarity	depth	table	price	x	y	z
1	1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
4	4	0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
5	5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
6	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48



	carat	cutGood	cutIdeal	cutPremium	cutVery Good	colorE	colorF	colorG	colorH	colorI	colorJ	clarityIF
1	0.23	0	1	0	0	1	0	0	0	0	0	0
2	0.21	0	0	1	0	1	0	0	0	0	0	0
3	0.23	1	0	0	0	1	0	0	0	0	0	0
4	0.29	0	0	1	0	0	0	0	0	1	0	0
5	0.31	1	0	0	0	0	0	0	0	0	1	0
6	0.24	0	0	0	1	0	0	0	0	0	0	1

	claritySI1	claritySI2	clarityVS1	clarityVS2	clarityVVS1	clarityVVS2	depth	table	price	x	y	z
1	0	1	0	0	0	0	61.5	55	326	3.95	3.98	2.43
2	1	0	0	0	0	0	59.8	61	326	3.89	3.84	2.31
3	0	0	1	0	0	0	56.9	65	327	4.05	4.07	2.31
4	0	0	0	1	0	0	62.4	58	334	4.20	4.23	2.63
5	0	1	0	0	0	0	63.3	58	335	4.34	4.35	2.75
6	0	0	0	0	0	1	62.8	57	336	3.94	3.96	2.48

Splitting of the data:

In a supervised learning model, the models need to be trained and tested. For training we use training dataset and for testing dataset is used. The total dataset is needed to be split into train and test dataset. As the dataset is huge, **hold out evaluation** will be used for this purpose. Following screenshot shows the code for holdout evaluation and the no of rows after splitting the data. Before splitting the data set, sample function is performed on the data. This shuffles the dataset rows to give a better set of data.

```
#hold out evaluation
dia_dummy$id <- NULL
dia_dummy <- dia_dummy[sample(nrow(dia_dummy)),]
select.data <- sample (1:nrow(dia_dummy), 0.8*nrow(dia_dummy))
train.data <- dia_dummy[select.data,]
test.data <- dia_dummy[-select.data,]
head(train.data)
head(test.data)
```

```
> nrow(train.data)
[1] 43152
> nrow(test.data)
[1] 10788
> nrow(dia_dummy)
[1] 53940
```

Building a model:

Using all the features as the x variable and the price as the y variable a full regression model is build. This model is the basic model with all the features in it. The summary of this model tells us a lot about the model and the feature variables and their contribution towards building the model. Inference can be made that the p value of the model is less than the alpha value (here 0.05), we can conclude that this is a good model. The adjacent R2 value is 0.9192, which means that in this model 91.92% of variation can be shown by x variables. The first column of the summary shows the intercept for each x variable which is the beta value in the regression model. The last column shows the individual p values. From this we can conclude that the p value for y and z is greater than the

```
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             1926.332    455.423   4.230 2.34e-05 ***
train.data$carat       11256.210     54.521 206.457  < 2e-16 ***
train.data$cutGood       588.306     37.673  15.616  < 2e-16 ***
train.data$cutIdeal      837.741     37.399  22.400  < 2e-16 ***
train.data$cutPremium    775.586     36.095  21.488  < 2e-16 ***
train.data$`cutVery Good` 736.430    36.132  20.382  < 2e-16 ***
train.data$colorE       -212.088     20.077 -10.564  < 2e-16 ***
train.data$colorF       -279.349     20.270 -13.781  < 2e-16 ***
train.data$colorG       -499.157     19.883 -25.105  < 2e-16 ***
train.data$colorH       -988.986     21.200 -46.651  < 2e-16 ***
train.data$colorI      -1474.792     23.709 -62.204  < 2e-16 ***
train.data$colorJ      -2380.010     29.229 -81.425  < 2e-16 ***
train.data$clarityIF    5383.720     57.204  94.115  < 2e-16 ***
train.data$claritySI1   3669.276     48.892  75.048  < 2e-16 ***
train.data$claritySI2   2697.312     49.093  54.943  < 2e-16 ***
train.data$clarityVS1   4578.735     49.910  91.740  < 2e-16 ***
train.data$clarityVS2   4268.380     49.139  86.864  < 2e-16 ***
train.data$clarityVVS1  5001.170     52.904  94.532  < 2e-16 ***
train.data$clarityVVS2  4960.778     51.379  96.553  < 2e-16 ***
train.data$depth         -58.481      5.012 -11.668  < 2e-16 ***
train.data$table         -27.748      3.275  -8.473  < 2e-16 ***
train.data$x           -1011.528     36.118 -28.006  < 2e-16 ***
train.data$y               9.794     21.818   0.449    0.654
train.data$z             -43.882     34.661  -1.266    0.206
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1134 on 43128 degrees of freedom
Multiple R-squared:  0.9192,    Adjusted R-squared:  0.9192
```

alpha value. If the p value of x the variable is less than alpha value, then it implies that the x variable does not contribute much towards the y variable.

Feature Selection:

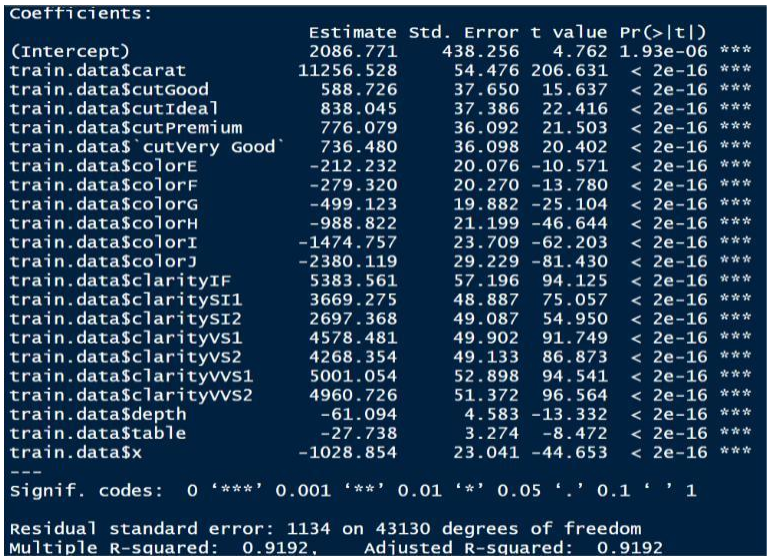
The full model from the previous step has all the feature variables contributing towards the y variable but as we saw in the summary of the model there are variables which are not so significant towards the model such variables need not be a part of the model. These variables are removed by the process of feature selection. In this project two criteria are used. These two criteria are: The p-value in individual parameter test and Akaike/Bayes Information Criterion (AIC/BIC). The search algorithm used are: Backward elimination using p value. Backward and forward by using AIC value. In the backward model we start the feature selection from the full mode and get to a more better regression model.

In **backward selection using p value** we select x variables based on their individual p values. If the p value for a variable is greater than the alpha value (0.05), then we can conclude that the contribution of that x variable is not significant on the model hence we can remove it from the model. This selection method is a manual method. For building this model we are using the train data set. Each independent p value will be compared with the alpha value and decision whether to keep the feature will be taken. The regression model is shown above. The summary of the model is shown in the screenshot.

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               2086.771    438.256   4.762 1.93e-06 ***
train.data$carat         11256.528     54.476 206.631  < 2e-16 ***
train.data$cutGood         588.726     37.650  15.637  < 2e-16 ***
train.data$cutIdeal        838.045     37.386  22.416  < 2e-16 ***
train.data$cutPremium      776.079     36.092  21.503  < 2e-16 ***
train.data$`cutVery Good`  736.480     36.098  20.402  < 2e-16 ***
train.data$colorE         -212.232     20.076 -10.571  < 2e-16 ***
train.data$colorF         -279.320     20.270 -13.780  < 2e-16 ***
train.data$colorG         -499.123     19.882 -25.104  < 2e-16 ***
train.data$colorH         -988.822     21.199 -46.644  < 2e-16 ***
train.data$colorI        -1474.757     23.709 -62.203  < 2e-16 ***
train.data$colorJ        -2380.119     29.229 -81.430  < 2e-16 ***
train.data$clarityIF      5383.561     57.196  94.125  < 2e-16 ***
train.data$claritySI1     3669.275     48.887  75.057  < 2e-16 ***
train.data$claritySI2     2697.368     49.087  54.950  < 2e-16 ***
train.data$clarityVS1     4578.481     49.902  91.749  < 2e-16 ***
train.data$clarityVS2     4268.354     49.133  86.873  < 2e-16 ***
train.data$clarityVVS1    5001.054     52.898  94.541  < 2e-16 ***
train.data$clarityVVS2    4960.726     51.372  96.564  < 2e-16 ***
train.data$depth           -61.094      4.583 -13.332  < 2e-16 ***
train.data$table           -27.738      3.274  -8.472  < 2e-16 ***
train.data$x             -1028.854     23.041 -44.653  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1134 on 43130 degrees of freedom
Multiple R-squared:  0.9192,    Adjusted R-squared:  0.9192
F-statistic: 2.338e+04 on 21 and 43130 DF,  p-value: < 2.2e-16
```

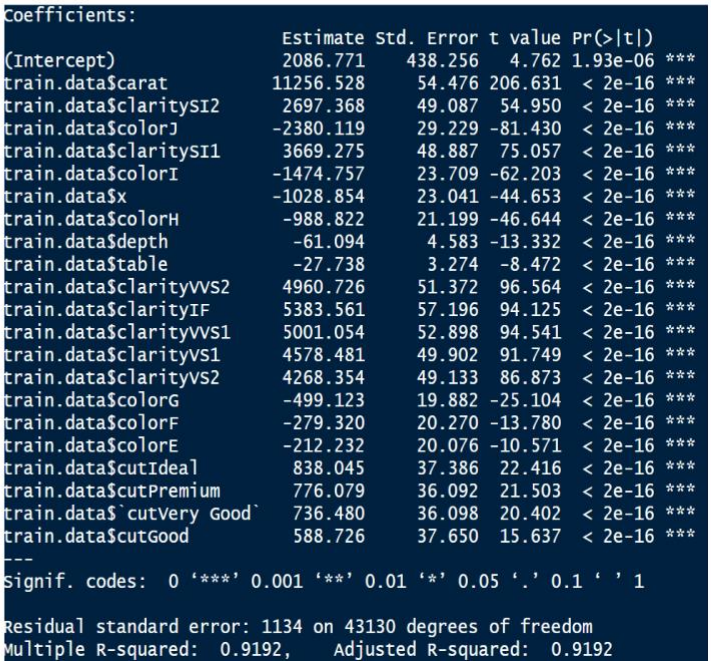
According to the summary of the model, there are two features for whom the p value is greater than the alpha value hence, we will drop one feature at a time. First, we will drop y feature and then run the model again. After executing the updated model p value for z is greater than alpha hence, we drop z next and run the model again. Now all the features are removed who do not contribute enough for the model. In this model we have all the significant variables. Hence this our final model. The adj r2 value for this model is 0.9192. The adjacent figure is the summary of the model.

In **backward selection using AIC** values step () function is used to build the model. It uses full model and from this it starts to select the significant variables based on the AIC metric. It is an automated way to build a model. The summary of the model is shown in the screenshot. According to the summary of the model, some variables have been dropped to make our model better. Only the significant variables are kept in the model. The AIC value of the model is 6070771. The adjacent r2 of this model is 0.9192. Which means 91.92% of variations in the y variable can be explained by change in the x variables.



```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               2086.771    438.256   4.762 1.93e-06 ***
train.data$carat         11256.528     54.476 206.631  < 2e-16 ***
train.data$cutGood         588.726     37.650  15.637  < 2e-16 ***
train.data$cutIdeal        838.045     37.386  22.416  < 2e-16 ***
train.data$cutPremium      776.079     36.092  21.503  < 2e-16 ***
train.data$`cutVery Good`  736.480     36.098  20.402  < 2e-16 ***
train.data$colorE         -212.232     20.076 -10.571  < 2e-16 ***
train.data$colorF         -279.320     20.270 -13.780  < 2e-16 ***
train.data$colorG         -499.123     19.882 -25.104  < 2e-16 ***
train.data$colorH         -988.822     21.199 -46.644  < 2e-16 ***
train.data$colorI        -1474.757     23.709 -62.203  < 2e-16 ***
train.data$colorJ        -2380.119     29.229 -81.430  < 2e-16 ***
train.data$clarityIF      5383.561     57.196  94.125  < 2e-16 ***
train.data$claritySI1     3669.275     48.887  75.057  < 2e-16 ***
train.data$claritySI2     2697.368     49.087  54.950  < 2e-16 ***
train.data$clarityVS1     4578.481     49.902  91.749  < 2e-16 ***
train.data$clarityVS2     4268.354     49.133  86.873  < 2e-16 ***
train.data$clarityVVS1    5001.054     52.898  94.541  < 2e-16 ***
train.data$clarityVVS2    4960.726     51.372  96.564  < 2e-16 ***
train.data$depth           -61.094      4.583 -13.332  < 2e-16 ***
train.data$table           -27.738      3.274  -8.472  < 2e-16 ***
train.data$x             -1028.854     23.041 -44.653  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1134 on 43130 degrees of freedom
Multiple R-squared:  0.9192,    Adjusted R-squared:  0.9192
```

In **forward selection using AIC** values step () function is used to build the model. The direction of building the model is given as forward explicitly. This function uses a base model and from this it starts to build the final model using the AIC metric. The summary of the model is shown in the screenshot. According to the summary of the model, some variables have been dropped to make our model better. Only the significant variables are kept in the model. The base model has carat as feature variable as the correlation of the variable is very high with the price variable hence it is important to retain this variable. The AIC value of the model is 6073191. The adj r2 of this model is 0.9192.



```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               2086.771    438.256   4.762 1.93e-06 ***
train.data$carat         11256.528     54.476 206.631  < 2e-16 ***
train.data$claritySI2     2697.368     49.087  54.950  < 2e-16 ***
train.data$colorJ        -2380.119     29.229 -81.430  < 2e-16 ***
train.data$claritySI1     3669.275     48.887  75.057  < 2e-16 ***
train.data$colorI        -1474.757     23.709 -62.203  < 2e-16 ***
train.data$x             -1028.854     23.041 -44.653  < 2e-16 ***
train.data$colorH         -988.822     21.199 -46.644  < 2e-16 ***
train.data$depth           -61.094      4.583 -13.332  < 2e-16 ***
train.data$table           -27.738      3.274  -8.472  < 2e-16 ***
train.data$clarityVVS2    4960.726     51.372  96.564  < 2e-16 ***
train.data$clarityIF      5383.561     57.196  94.125  < 2e-16 ***
train.data$clarityVVS1    5001.054     52.898  94.541  < 2e-16 ***
train.data$clarityVS1     4578.481     49.902  91.749  < 2e-16 ***
train.data$clarityVS2     4268.354     49.133  86.873  < 2e-16 ***
train.data$colorG         -499.123     19.882 -25.104  < 2e-16 ***
train.data$colorF         -279.320     20.270 -13.780  < 2e-16 ***
train.data$colorE         -212.232     20.076 -10.571  < 2e-16 ***
train.data$cutIdeal        838.045     37.386  22.416  < 2e-16 ***
train.data$cutPremium      776.079     36.092  21.503  < 2e-16 ***
train.data$`cutVery Good`  736.480     36.098  20.402  < 2e-16 ***
train.data$cutGood         588.726     37.650  15.637  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1134 on 43130 degrees of freedom
Multiple R-squared:  0.9192,    Adjusted R-squared:  0.9192
```

The model to be selected will be compared on a few values. The AIC value of the model should be less, residual error must be less and the adjacent R2 must be as high as possible. The root mean squared error is also calculated for the model. Based on all these factors the backward model with AIC value is selected. The next step no will be to validate the model that has been selected.

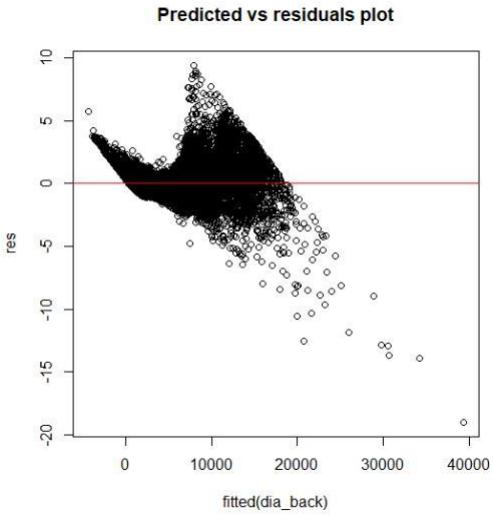
Goodness of fit:

The regression model must be validated and checked if the model is qualified or not. Goodness of fit has three components: F test, Individual Parameter test and Coefficient of determination R2. In the **F test** the whole model is checked if it is good or not. P-value of the model is less than 0.05. hence, we can conclude that at 95% confidence level, we say that at least one X variable has significant linear relationship with Y variable, and it can affect the value of the Y variable. In **Individual parameter test** each variable is check for its significance towards the model built. According to the summary of the model All the feature variables have p value less than 0.05 are making a significant impact on the model. In **Coefficient of determination R2**: The R2 value of the model given in the summary tells us how much variation in y can be explained by the x variable. But the value of R2 changes with addition on x variable that does not improve the model. Hence, we use the adj R2 value, higher the value better the model is. Here the value of adj R2 is 0.9192 that means 91% of variation in y variable can be explained using x variable

Residual analysis:

While building the model assumptions about the data are made such as there exists a linear relationship between the variables, there exists constant variance and

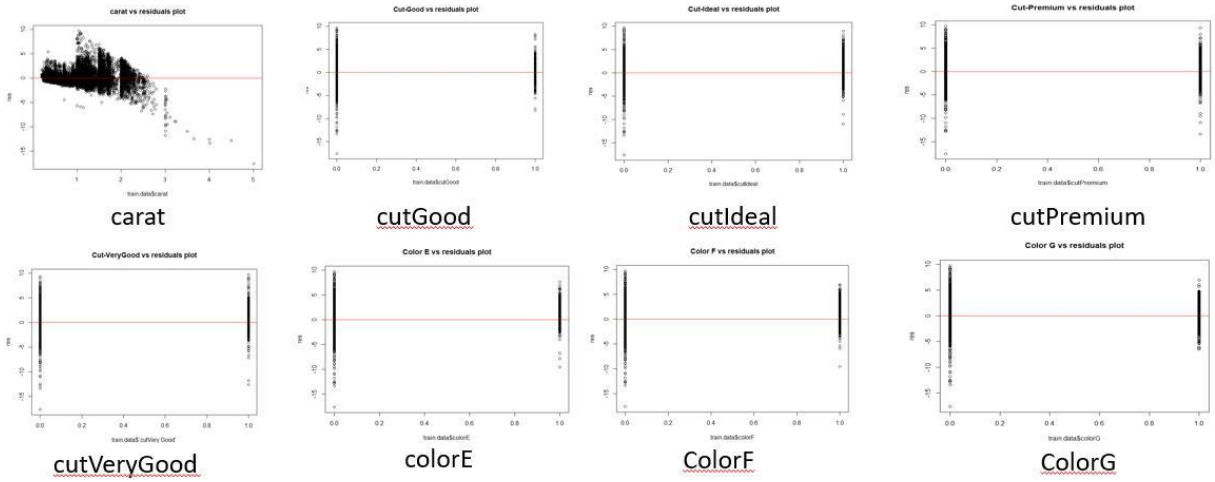
that the data is normally distributed. These assumptions need to be validated s for our model. This is done in residual analysis. The goals in Residual Analysis:



Validate the constant Variance:

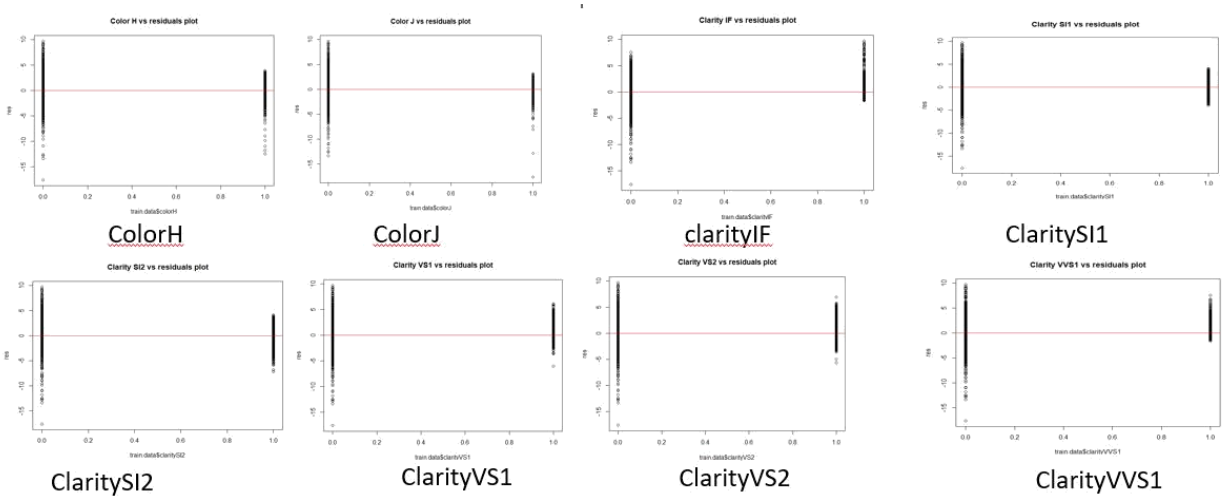
Plot residual vs predicted is used to check constant variance. Some of the data points in this plot are not constantly spread in the initial part of the graph. This graph is for few data points. But the area where maximum data points are mapped have a constant variance. If we draw parallel lines the data points will fall inside it. The pattern of spread for majority of data points is showing that there is approximate constant variance. There are very few outliners to this plot. Hence, based on the explanation given above we approve of this plot.

Validate the linear relationship:

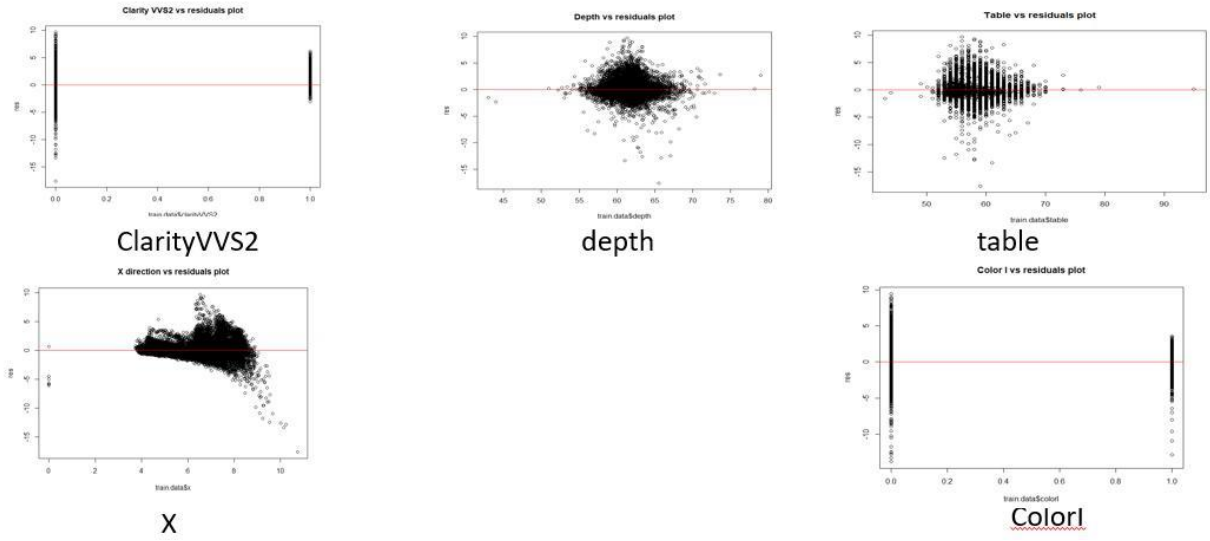


For the numerical columns:

Carat, x and z, the spread of the data points is constant. There are a few outliners present in the plot. If we draw parallel lines the data points will fall in it. There is no such evident pattern for the plot of the data. Hence, we conclude that there exists constant variance and there is a Linear relationship between the x and y variable.

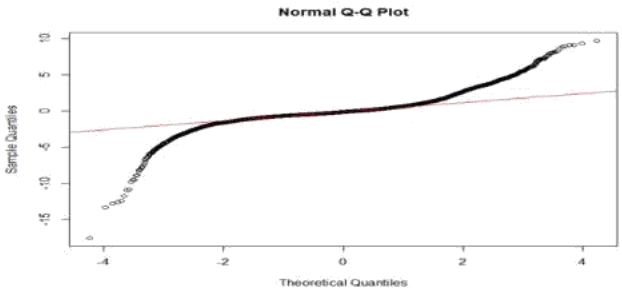


For variables depth and table, the data points are concentrated near the center but for the overall data the spread of the data points is constant. There are a few outliners present in the plot. If we draw parallel lines the data points will fall in it. There is no such evident pattern for the plot of the data. Hence, we conclude that there exists constant variance and there is a Linear relationship between the x and y variable. For dummy variable: The graph is almost same for all the dummy variables. Not much can be concluded for them as they have discrete values (either 0 or1). If we draw parallel lines the data points will fall in it. There is no such evident pattern for the plot of the data. Hence, we conclude that there exists constant variance and there is a Linear relationship between the x and y variable.



Validate normal distribution of residuals:

Validate normal distribution of residuals by QQ plot and by Jarque-Bera Normality Test. As there are outliners present in the data the normality plot is skewed on both the sides. But as we can observe from the plot maximum of the data points follows the line. Hence, we can say that the data points are close to normal distribution. P value of the model is less than 0.05 hence we say it follows normal distribution at 95% confidence level. This test is performed on data with data having more than 5k rows. All the three tests tell us that the model selected is a good and validated model.




```
install.packages("tseries")
library(tseries)
jarque.bera.test(res)
> jarque.bera.test(res)

        Jarque Bera Test

data:  res
X-squared = 354085, df = 2, p-value < 2.2e-16
```

Post Processing:

In post processing of the model, improving of the current model is done. As the data was not preprocessed with collinearity of the feature variables. There is a need to check the multicollinearity of the variables. For this we will be computing **variance inflation factor** (VIF). A large value of VIF (larger than 4) is a sign of strong multicollinearity. In such a case when the VIF is greater than 4 for two variables then both the variables change with each other. This makes one of the variable redundant as the change is already measure with the other variable. One of the variables should be dropped from the model. As we can see from the output there are many variables having high vif value. That means there is high collinearity between these x variables. According to the screenshot there is high collinearity between few of the variables. That means there is a need to drop a few x variables and run the model again to check the vif value. We have dropped the **cutIdeal, colorI, claritySI1 and x** feature variable. After deleting those variables, we are getting the vif value for all the feature variables below 4. The entire model was built again without these variables. The best model used after feature selection and after measuring the predictive performance was the backward model build by AIC metrics. The following screen shots shows the change in the values.

```
> vif(dia_back) # variance inflation factors
          train.data$carat           train.data$cutGood          train.data$cutIdeal
                  22.332646                    3.925265                   11.254627
   train.data$cutPremium train.data$`cutVery Good`          train.data$colorE
                   8.299308                    7.582132                    2.010371
       train.data$colorF           train.data$colorG          train.data$colorH
                   2.017982                    2.191589                    1.941223
       train.data$colorI           train.data$colorJ         train.data$clarityIF
                   1.707300                    1.426413                    3.507275
  train.data$claritySI1       train.data$claritySI2       train.data$clarityVS1
                  14.718297                   11.452685                   10.744545
  train.data$clarityVS2       train.data$clarityVVS1      train.data$clarityVVS2
                  14.194244                    5.873929                    7.543738
       train.data$depth           train.data$table                train.data$x
                   1.450696                    1.793842                   22.359010
```

```
> vif(dia_back1)
          train.data$carat           train.data$cutGood   train.data$cutPremium train.data$`cutVery Good`
                  1.247245                    1.306589                    1.556387                    1.348212
       train.data$colorE           train.data$colorF           train.data$colorG          train.data$colorH
                  1.497443                    1.479249                    1.555223                    1.429105
       train.data$colorJ         train.data$clarityIF       train.data$claritySI2       train.data$clarityVS1
                  1.201697                    1.145108                    1.417407                    1.387752
  train.data$clarityVS2       train.data$clarityVVS1      train.data$clarityVVS2           train.data$depth
                  1.480602                    1.244905                    1.296324                    1.179342
          train.data$table
                  1.549290
```

After removing the redundant variables there is a need to identify the influential points which are significant for model. The **influential points** can miss lead a model; hence they are removed from the dataset. In order to do so we need to find the Cooks distance. To find it, Cooks Distance metrics is used. Cooks Distance has a threshold of 4/N,

```
cooksd1 <- cooks.distance(dia_back1)
# influential row numbers
influential1 <- as.numeric(names(cooksd1)[(cooksd1 > 4/nrow(train.data))])
head(train.data[influential1, ])
train.data1 <- train.data[-c(19534, 31109, 17573,37886,32318), ]
```

```
      carat cutGood cutIdeal cutPremium cutVery Good colorE colorF colorG colorH
52533  0.70       0        1          0            0      0      0      0      0
8379   0.31       0        1          0            0      0      1      0      0
17651  1.08       0        1          0            0      0      1      0      0
52971  0.40       0        1          0            0      0      0      0      0
41013  0.50       0        1          0            0      1      0      0      0
38591  0.57       0        0          0            1      0      0      0      0
      colorI colorJ clarityIF claritySI1 claritySI2 clarityVS1 clarityVS2 clarityVVS1
52533      1      0         0          0          0          0          0          0
8379       0      0         0          0          0          0          1          0
17651      0      0         0          0          0          0          1          0
52971      0      1         0          1          0          0          0          0
41013      0      0         0          0          1          0          0          0
38591      0      1         0          1          0          0          0          0
      clarityVVS2 depth table price    x    y    z
52533           1  61.1    56  2530 5.73 5.76 3.52
8379            0  61.9    57   583 4.32 4.34 2.68
17651           0  61.9    55  7110 6.58 6.64 4.09
52971           0  62.2    57   552 4.71 4.74 2.94
41013           0  61.1    58  1185 5.09 5.13 3.12
38591           0  60.6    61  1037 5.36 5.40 3.26
```

where N is the size of data. Here it is 80% of the total data set. It is a very small distance as the data size is very huge. In the provided screenshot we can see that after running the function there are 5 influential points shown to us. Here I have written the code which will give us the rom number for the influential points. We need to remove these influential points from our data set which we were using. Here the variable is train. Data. After removing the data points given by the cooks distance calculations the same model was built again and rmse was calculated. Initially the rmse value was 11014.93 but now after removing the influential points the rmse value reduces by a good amount for the model to 10980.

FINAL MULTIPLE REGRESSION MODEL:

The multiple regression model we have is:

Price = β0 + β1*carat + β2*cutGood + β3*cutPremium + β4*cutVeryGood + β5*colorE + β5*colorF +β6*colorG + β7*colorH + β9*colorJ + β11* claritySI2 + β12* clarityVS1 + β13* clarityVS2 + β14* clarityVVS1 + β15* clarityVVS2 + β16* clarityIF + β17*depth + β18*table

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               5727.729    396.106  14.460  < 2e-16 ***
train.data1$carat         8615.753     14.579 590.956  < 2e-16 ***
train.data1$cutGood        -38.400     24.595  -1.561    0.118
train.data1$cutPremium      95.585     17.699   5.401 6.67e-08 ***
train.data1$`cutVery Good`  90.961     17.235   5.278 1.31e-07 ***
train.data1$colorE         376.604     19.619  19.196  < 2e-16 ***
train.data1$colorF         278.404     19.651  14.168  < 2e-16 ***
train.data1$colorG          97.087     18.965   5.119 3.08e-07 ***
train.data1$colorH        -349.722     20.597 -16.979  < 2e-16 ***
train.data1$colorJ       -1615.792     30.377 -53.191  < 2e-16 ***
train.data1$clarityIF     1929.624     37.014  52.132  < 2e-16 ***
train.data1$claritySI2    -710.606     19.555 -36.339  < 2e-16 ***
train.data1$clarityVS1    1062.737     20.307  52.335  < 2e-16 ***
train.data1$clarityVS2     815.480     17.968  45.385  < 2e-16 ***
train.data1$clarityVVS1   1557.832     27.573  56.498  < 2e-16 ***
train.data1$clarityVVS2   1529.518     24.112  63.433  < 2e-16 ***
train.data1$depth          -85.513      4.678 -18.279  < 2e-16 ***
train.data1$table          -69.027      3.445 -20.035  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1284 on 43129 degrees of freedom
Multiple R-squared:  0.8964,    Adjusted R-squared:  0.8964
F-statistic: 2.196e+04 on 17 and 43129 DF,  p-value: < 2.2e-16
```

The model tells us that if all the variables are kept constant except for carat or any other feature variable there will be beta(coefficient) times change in the price variable. This is the final regression model. This model can be used by the traders and the sellers to price the diamonds accurately based on the value of each feature the model has. **This model has an accuracy of 89.64%.** The summary of the model is shown in the adjacent figure.

Classification Model

The classification model is a binary classification model, it is a part of supervised learning. The model will predict the output into one of the labels. Here the question is which label of diamond do we have with all the features given. The label variable is the price variable. It has two factors Inexpensive and Expensive. The feature variables are used to predict the label. The data must be preprocessed and make it fit for performing classification. The feature variables must be numerical values and the y variable is the categorical variable with labels. There is a need to normalize the data. There are three classification models built. The preprocessing of the data is almost same for the classification models, for example: converting the y variable into label, normalization of data, etc.

Preprocessing:

The y variable for classification must be a factor variable. The price variable is converted into **a factor variable** by creating two factors Inexpensive and Expensive. If the value of the price is less than equal to 4000 it is Inexpensive if the value is greater than 4000 then it is Expensive.

The data is checked for **null values and missing values**. The dataset does not have any missing or null value data.

The dataset has one **unnamed column** which is the serial number for the dataset rows. This information is redundant in building the model; hence the column is removed.

The most important data preprocessing needed for a classification model is **normalization.** The data values are brought in a range of [0,1], just so that the weightage of each data value will be equal and not column of dataset will have more impact on the classification decision. Normalization is performed only on the numerical columns.

	carat	cut	color	clarity	depth	table	price
	0.16632017	Premium	F	VVS1	0.5000000	0.3269231	Expensive
	0.03534304	Ideal	E	VS2	0.5444444	0.2500000	Inexpensive
	0.06444906	Good	G	SI2	0.5611111	0.2692308	Inexpensive
	0.14553015	Premium	G	VS2	0.5500000	0.2884615	Expensive
	0.06237006	Premium	F	VS2	0.5194444	0.3269231	Inexpensive
	0.02286902	Premium	F	VS2	0.5277778	0.3269231	Inexpensive

```
> #collinearity between carat x,y and z is very high so we remove the variables
> num$mydata.x <- NULL
> cor(num)
             mydata.carat mydata.depth mydata.table    mydata.y   mydata.z
mydata.carat   1.00000000   0.02822431    0.1816175  0.95172220 0.95338738
mydata.depth   0.02822431   1.00000000   -0.2957785 -0.02934067 0.09492388
mydata.table   0.18161755  -0.29577852    1.0000000  0.18376015 0.15092869
mydata.y       0.95172220  -0.02934067    0.1837601  1.00000000 0.95200572
mydata.z       0.95338738   0.09492388    0.1509287  0.95200572 1.00000000
> num$mydata.y <- NULL
> cor(num)
             mydata.carat mydata.depth mydata.table   mydata.z
mydata.carat   1.00000000   0.02822431    0.1816175 0.95338738
mydata.depth   0.02822431   1.00000000   -0.2957785 0.09492388
mydata.table   0.18161755  -0.29577852    1.0000000 0.15092869
mydata.z       0.95338738   0.09492388    0.1509287 1.00000000
> num$mydata.z <- NULL
> cor(num)
             mydata.carat mydata.depth mydata.table
mydata.carat   1.00000000   0.02822431    0.1816175
mydata.depth   0.02822431   1.00000000   -0.2957785
mydata.table   0.18161755  -0.29577852    1.0000000
>
```

The next step is **removing the features with high collinearity**. Previously it has been mentioned that carat, x, y, z has the highest collinearity. Each variable one at a time was removed and the collinearity was checked for it and at the end x, y and z was deleted and only carat was retained. We have not used correlation as the y variable is a categorical variable.

1. K Nearest Neighbor

In the KNN classification model we find the distance of the target from the instances present and arrange that distance from minimum to maximum. Based on the number of how many nearest neighbors we want to analyze (k) we will decide the label. K stands for the number of neighbors we look for to classify the target. K should be an odd number. The final label is decided by choosing the majority class labels for those neighbors. There are two ways the distance is calculated between target and instance; Euclidean and Manhattan.

Preprocessing:

Creating N-1 dummy variables. This needs to be done as the classification model does not accept categorical variable as a feature. All the categories are converted to N-1 dummy variables, for N factors we create N-1 dummy variable. For example: Cut: levels-Fair, Good, Ideal, Premium, Very Good. We dropped- Fair.

```
> head(data)
      carat cutGood cutIdeal cutPremium cutVery Good colorE colorF colorG colorH colorI colorJ clarityIF claritySI1
1 0.006237006       0        1          0           0      1      0      0      0      0      0         0          0
2 0.002079002       0        0          1           0      1      0      0      0      0      0         0          1
3 0.006237006       1        0          0           0      1      0      0      0      0      0         0          0
4 0.018711019       0        0          1           0      0      0      0      0      1      0         0          0
5 0.022869023       1        0          0           0      0      0      0      0      0      1         0          0
6 0.008316008       0        0          0           1      0      0      0      0      0      1         0          0
  claritySI2 clarityVS1 clarityVS2 clarityVVS1 clarityVVS2     depth     table       price         x         y
1          1          0          0           0           0 0.5138889 0.2307692 Inexpensive 0.3677840 0.06757216
2          0          0          0           0           0 0.4666667 0.3461538 Inexpensive 0.3621974 0.06519525
3          0          1          0           0           0 0.3861111 0.4230769 Inexpensive 0.3770950 0.06910017
4          0          0          1           0           0 0.5388889 0.2884615 Inexpensive 0.3910615 0.07181664
5          1          0          0           0           0 0.5638889 0.2884615 Inexpensive 0.4040968 0.07385399
6          0          0          0           0           1 0.5500000 0.2692308 Inexpensive 0.3668529 0.06723260
           z
1 0.07641509
2 0.07264151
3 0.07264151
4 0.08270440
5 0.08647799
6 0.07798742
```

Splitting of data:

For performing classification, we need to split the data into training set and testing set. As the dataset is huge, hold out evaluation will be used for this purpose. Following screenshot shows the code for **Holdout evaluation** and the no of rows after splitting the data. Before splitting the data set, sample function is performed on the data. This shuffles the dataset rows to give a better set of data.

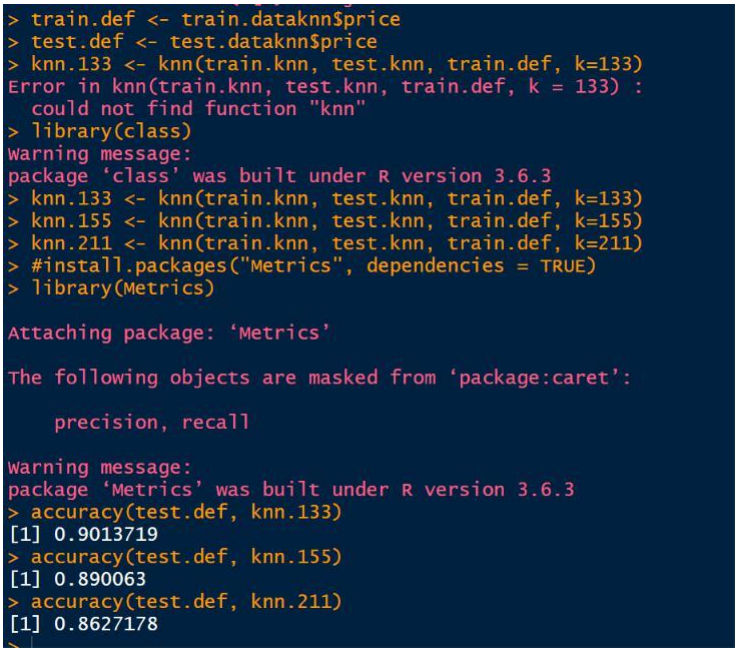
```
#hold out evaluation
knndata <- knndata[sample(nrow(knndata)),]
select.dataknn <- sample (1:nrow(knndata), 0.8*nrow(knndata))
train.dataknn <- knndata[select.dataknn,]
test.dataknn <- knndata[-select.dataknn,]
test.knn <- test.dataknn
train.knn <- train.dataknn
head(train.knn)

nrow(test.svm)
nrow(train.svm)

train.knn$price<-NULL
test.knn$price<-NULL
train.def <- train.dataknn$price
test.def <- test.dataknn$price

library(class)
knn.133 <- knn(train.knn, test.knn, train.def, k=133)
knn.155 <- knn(train.knn, test.knn, train.def, k=155)
knn.211 <- knn(train.knn, test.knn, train.def, k=211)
#install.packages("Metrics", dependencies = TRUE)
library(Metrics)
accuracy(test.def, knn.133)
accuracy(test.def, knn.155)
accuracy(test.def, knn.211)
```

Building a model:

We will be building three knn models with varying k values. The number of nearest neighbors we will be checking is defined by k. As the data is large enough, we will be using greater values for k. In this model I have defined k as 133, 155 and 211. The value for k must be odd. The function knn () is used it is passed with train data and test data without the label and the train label are also passed the k value defines the number of nearest neighbors used to decide about the test labels.

Evaluation of the model:

To measure the accuracy, we are using the accuracy () function. It is passed with test data and the knn model. The first variable for accuracy is the truth value or the true label for the test data and in the second argument we give the label our model has predicted. KNN classification does not much learning, no optimizations which implies there will be no overfitting problem. Based on the model prepared, the accuracy of the model with different k value is different.



For k = 133 accuracy is 0.9013

For k = 155 accuracy is 0.89

For k as 211 accuracy is 0.8627

Knn.133 is the model with highest accuracy of prediction. The best accuracy our model gives are for k = 133 which is **90.13%.**

There are other 3 classification models implemented to compare the accuracy.

2. Naïve Bayes

Naive Bayes is among one of the most simple and powerful algorithms for classification based on Bayes'

Theorem. The Bayes theorem states that "The posterior probability equals the prior probability times the likelihood ratio". Naive Bayes model is easy to build and particularly useful for very large data sets. In Naïve Bayes all these features independently contribute to the probability of the outcome. First, we will create a frequency table using each attribute of the dataset. For each frequency table, we will generate a likelihood tables. So, with the data, we must predict whether diamonds are expensive or not.

```
#hold out evaluation
mydata <- mydata[sample(nrow(mydata)),]
select.datan <- sample (1:nrow(mydata), 0.8*nrow(mydata))
train.datan <- mydata[select.data,]
test.datan <- mydata[-select.data,]
test.naive <- test.datan
train.naive <- train.datan

nrow(test.naive)
nrow(train.naive)

#naive model
#install.packages("e1071")
library(e1071)

naive <- naiveBayes(train.naive$price~. , data = train.naive)
naive

#predict
#install.packages("caret")
library(caret)
pre_naive = predict(naive, test.naive, type = "raw")
head(pre_naive)
confusionMatrix(table(pre_naive,test.naive$price))
```

Preprocessing:

The naïve Bayes model does not require the conversion of categorical data into dummy variables. The naïve Bayes model can have features as both numerical and categorical columns.

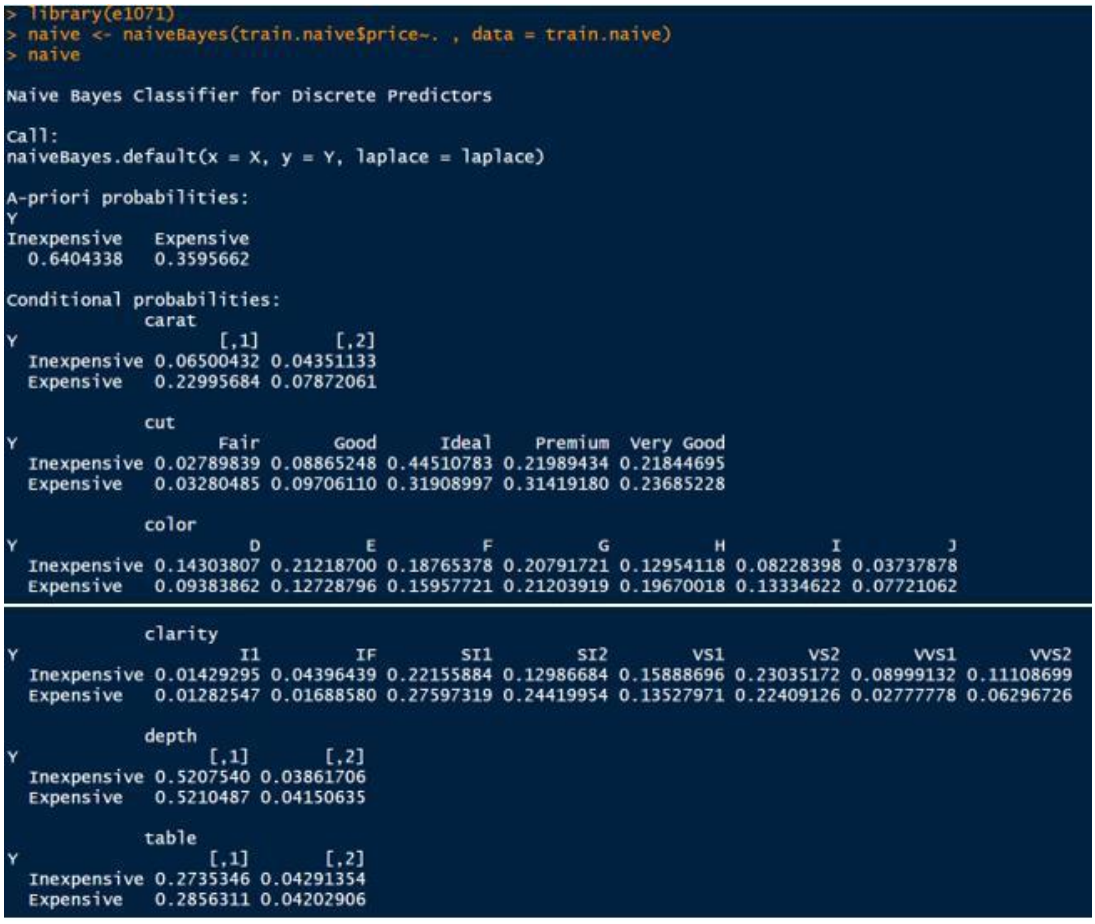
carat	cut	color	clarity	depth	table	price
0.16632017	Premium	F	VVS1	0.5000000	0.3269231	Expensive
0.03534304	Ideal	E	VS2	0.5444444	0.2500000	Inexpensive
0.06444906	Good	G	SI2	0.5611111	0.2692308	Inexpensive
0.14553015	Premium	G	VS2	0.5500000	0.2884615	Expensive
0.06237006	Premium	F	VS2	0.5194444	0.3269231	Inexpensive
0.02286902	Premium	F	VS2	0.5277778	0.3269231	Inexpensive

Splitting of data:

For performing classification, we need to split the data into training set and testing set. As the dataset is huge, hold out evaluation will be used for this purpose. Following screenshot shows the code for **Holdout evaluation** and the no of rows after splitting the data. Before splitting the data set, sample function is performed on the data. This shuffles the dataset rows to give a better set of data.

Building the Naïve Bayes model:

To build a naïve Bayes model we are using the naïvebayes () function. The parameter to this is the full regression model. The data used is the train data as we are just training the model to predict the class. The output of the model is given in the adjacent screenshot. Under the Apriori Probabilities there is a table with values for Inexpensive and Expensive. It says that 64% of the diamonds are in the Inexpensive category and the rest are in Expensive category. The tables under conditional probability are the likelihood tables for out variables



```
> library(e1071)
> naive <- naiveBayes(train.naive$price~. , data = train.naive)
> naive

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
Inexpensive   Expensive
  0.6404338   0.3595662

Conditional probabilities:
            carat
Y                [,1]       [,2]
  Inexpensive 0.06500432 0.04351133
  Expensive   0.22995684 0.07872061

            cut
Y                Fair       Good      Ideal    Premium  Very Good
  Inexpensive 0.02789839 0.08865248 0.44510783 0.21989434 0.21844695
  Expensive   0.03280485 0.09706110 0.31908997 0.31419180 0.23685228

            color
Y                   D          E          F          G          H          I          J
  Inexpensive 0.14303807 0.21218700 0.18765378 0.20791721 0.12954118 0.08228398 0.03737878
  Expensive   0.09383862 0.12728796 0.15957721 0.21203919 0.19670018 0.13334622 0.07721062

            clarity
Y                  I1         IF        SI1        SI2        VS1        VS2       VVS1       VVS2
  Inexpensive 0.01429295 0.04396439 0.22155884 0.12986684 0.15888696 0.23035172 0.08999132 0.11108699
  Expensive   0.01282547 0.01688580 0.27597319 0.24419954 0.13527971 0.22409126 0.02777778 0.06296726

            depth
Y                [,1]       [,2]
  Inexpensive 0.5207540 0.03861706
  Expensive   0.5210487 0.04150635

            table
Y                [,1]       [,2]
  Inexpensive 0.2735346 0.04291354
  Expensive   0.2856311 0.04202906
```

For a numerical value probability are not calculated rather the mean and standard deviation is calculated. For example: Carat is a numerical variable. The conditional probability tables say that on an average carat value as 0.065 falls in inexpensive category with a standard deviation of 0.043 and average carat value for Expensive diamond is 0.2299 with a standard deviation of 0.078. For a categorical value, let's consider color (D); it can be read as the conditional probability of the diamond being expensive and has a colorD is 0.09 whereas with colorD there is much higher probability of the diamond being inexpensive; 0.14.

Prediction and Accuracy:

We use the predict () function to predict the test output. The parameters are the naïve Bayes model and the testing dataset. To evaluate the model, we are using a **confusion matrix**. The accuracy of this model is 92.47. On the top left corner there is an expensive and inexpensive table. It tells that there are 6324 Inexpensive observations which were correctly predicted by our model into the inexpensive label and there are 3652 expensive observations correctly predicted. Whereas 211 expensive observations were wrongly predicted as inexpensive and vice versa for 601 observations. **Accuracy is given by sum of the correctly predicted values / the total observations.** The CI entry tells that the model is 95% confident of giving an accuracy between 91.96 and 92.96. P value of the model is less than alpha, which makes it a good model. The sensitivity and specificity manage the incorrect predictions.

```
> confusionMatrix(table(pre_naive,test.naive$price))
Confusion Matrix and Statistics


pre_naive     Inexpensive Expensive
  Inexpensive        6324       211
  Expensive           601      3652

               Accuracy : 0.9247
                 95% CI : (0.9196, 0.9296)
    No Information Rate : 0.6419
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8398

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9132
            Specificity : 0.9454
         Pos Pred Value : 0.9677
         Neg Pred Value : 0.8587
             Prevalence : 0.6419
         Detection Rate : 0.5862
   Detection Prevalence : 0.6058
      Balanced Accuracy : 0.9293

       'Positive' Class : Inexpensive
```

Using the raw type, we can see the probability for each of the entry. For the first diamond entry we can see it probability of being expensive=e and inexpensive.

```
> pre_naive = predict(naive, test.naive, type = "raw")
> head(pre_naive)
     Inexpensive   Expensive
[1,] 4.132322e-03 0.995867678
[2,] 9.955023e-01 0.004497695
[3,] 9.958531e-01 0.004146882
[4,] 1.499133e-01 0.850086661
[5,] 1.512267e-14 1.000000000
[6,] 9.894210e-01 0.010579013
```

The accuracy of the Naïve Bayes model is 92.47%

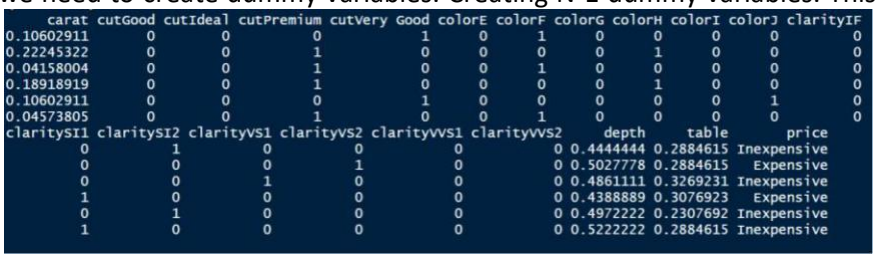
3. Support Vector Machines

SVM (Support Vector Machine) is a supervised machine learning algorithm which is mainly used to classify data into different classes. SVM makes use of a hyperplane which acts like a decision boundary between the various classes. SVM can be used to generate multiple separating hyperplanes such that the data is divided into segments and each segment contains only one kind of data. The support vectors are the points which are closest data points to the hyperplane. The hyperplane is drawn based on these support vectors and an optimum hyperplane will have a maximum distance from each of the support vectors. And this distance between the hyperplane and the support vectors is known as the margin. SVM trains on a set of labeled data. SVM studies the labeled training data and then classifies any new input data depending on what it learned in the training phase. SVM can also work on nonlinear data, it uses kernel function to transform the data into other dimensions.

```
select.datasvm <- sample (1:nrow(data), 0.8*nrow(data))
train.datasvm <- data[select.datasvm,]
test.datasvm <- data[-select.datasvm,]
test.svm <- test.datasvm
train.svm <- train.datasvm
head(train.svm)
nrow(test.svm)
nrow(train.svm)

#svm model
#install.packages("e1071")
library(e1071)
#install.packages("caret")
library(caret)
#install.packages("kernlab")
library(kernlab)

svm1 <- svm(train.svm$price~. , data = train.svm, kernel = "linear", cost = 0.1, scale = F)
#summary(svm1)
#plot(svm1,train.svm[,])

#predict
#install.packages("caret")
library(caret)
pre_svm = predict(svm1, test.svm, type = "class")
#plot(pre_svm)
table(pre_svm, test.svm$price)
mean(pre_svm==test.svm$price)

confusionMatrix(table(pre_svm,test.svm$price))
```

Preprocessing:

For a svm classification model we need to create dummy variables. Creating N-1 dummy variables. This needs to be done as the classification model does not accept categorical variable as a feature. All the categories are converted to N-1 dummy variables, for N factors we create N-1 dummy variable. For example: Cut: levels- Fair, Good, Ideal, Premium, Very Good. We dropped-Fair.



	carat	cutGood	cutIdeal	cutPremium	cutVery Good	colorE	colorF	colorG	colorH	colorI	colorJ	clarityIF
0.10602911		0	0	0	1	0	1	0	0	0	0	0
0.22245322		0	0	1	0	0	0	0	1	0	0	0
0.04158004		0	0	1	0	0	1	0	0	0	0	0
0.18918919		0	0	1	0	0	0	0	1	0	0	0
0.10602911		0	0	0	1	0	0	0	0	0	1	0
0.04573805		0	0	1	0	0	1	0	0	0	0	0

	clarity SI1	clarity SI2	clarity VS1	clarity VS2	clarity VVS1	clarity VVS2	depth	table	price
0	1	0	0	0	0	0.4444444	0.2884615	Inexpensive	
0	0	0	1	0	0	0.5027778	0.2884615	Expensive	
0	0	1	0	0	0	0.4861111	0.3269231	Inexpensive	
1	0	0	0	0	0	0.4388889	0.3076923	Expensive	
0	1	0	0	0	0	0.4972222	0.2307692	Inexpensive	
1	0	0	0	0	0	0.5222222	0.2884615	Inexpensive	

Splitting of data:

For performing classification, we need to split the data into training set and testing set. As the dataset is huge, hold out evaluation will be used for this purpose. Following screenshot shows the code for **Holdout evaluation** and the no of rows after splitting the data. Before splitting the data set, sample function is performed on the data. This shuffles the dataset rows to give a better set of data.

Building the SVM model:

The output of the SVM model shows that it has performed C type classification. The given values for the cost variable was 0.1. as there was enough linear relationship in our data as mentioned when performing residual analysis, the linear kernel was selected to perform this classification. The number of support vectors present in the dataset were 12203. This was performed on the train data set. There are two classes Expensive and Inexpensive.

```
Call:
svm(formula = train.svm$price ~ ., data = train.svm, kernel = "linear",
    cost = 0.1, scale = F)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  12203

 ( 6102 6101 )


Number of Classes:  2

Levels:
 Inexpensive Expensive
```

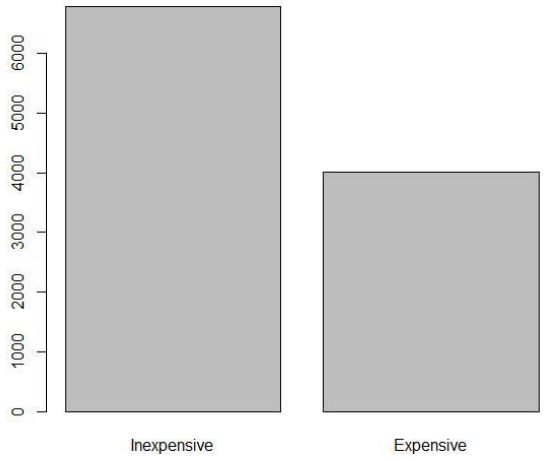
Prediction and Accuracy:

We use the predict () function to predict the test output. The parameters are the SVM model and the testing dataset. To evaluate the model, we are using a **confusion matrix.** The accuracy of this model is 96.7%. On the top left corner there is an expensive and inexpensive table. It tells that there are 6661 Inexpensive observations which were correctly predicted by our model into the inexpensive label and there are 3771 expensive observations correctly predicted. Whereas 118 expensive observations were wrongly predicted as inexpensive and vice versa for 238 observations. The correct prediction number has increased from what it was for the naïve Bayes model. **Accuracy is given by sum of the correctly predicted values / the total observations.**

```
> confusionMatrix(table(pre_svm,test.svm$price))
Confusion Matrix and Statistics


pre_svm       Inexpensive Expensive
  Inexpensive        6661       118
  Expensive           238      3771

               Accuracy : 0.967
                 95% CI : (0.9635, 0.9703)
    No Information Rate : 0.6395
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9289

 Mcnemar's Test P-Value : 2.845e-10

            Sensitivity : 0.9655
            Specificity : 0.9697
         Pos Pred Value : 0.9826
         Neg Pred Value : 0.9406
             Prevalence : 0.6395
         Detection Rate : 0.6174
   Detection Prevalence : 0.6284
      Balanced Accuracy : 0.9676

       'Positive' Class : Inexpensive
```

The CI entry tells that the model is 95% confident of giving an accuracy between 96.35% and 97.03%. P value of the model is less than alpha, which makes it a good model. The sensitivity and specificity manage the incorrect predictions.

A plot for how the labels is distributed for the test data is shown. We can see that the inexpensive label has highest number of diamonds in its category.



The accuracy of the SVM model is 96.7%

4. Random Forest

Random forest builds multiple decision trees (called the forest) and glues them together to get a more accurate and stable prediction. The forest it builds is a collection of Decision Trees, trained with the bagging method. it randomly selects a set of parameters and creates a decision tree for each set of chosen parameters. after creating multiple Decision trees using this method, each tree selects or votes the class, and the class receiving the most votes by a simple majority is termed as the predicted class.

```
#hold out evaluation
rfdata <- rfdata[sample(nrow(rfdata)),]
select.datarf <- sample (1:nrow(rfdata), 0.8*nrow(rfdata))
train.datarf <- rfdata[select.datarf,]
test.datarf <- rfdata[-select.datarf,]
test.rf <- test.datarf
train.rf <- train.datarf

nrow(test.rf)
nrow(train.rf)

#naive model
install.packages("randomForest")
library(randomForest)

rf <- randomForest(train.rf$price~. , data = train.rf)
rf
attributes(rf)

#predict
#install.packages("caret")
library(caret)
pre_rf = predict(rf, test.rf)
head(pre_rf)
head(test.rf$price)
confusionMatrix(table(pre_rf,test.rf$price))
plot(rf)
head(train.rf)
```

Preprocessing:

Other than the preprocessing done on the data we have to convert the label variable into a numerical class 0 or 1. Here 0 stand for Inexpensive and 1 stands for expensive. This will be used to predict the labels. The diamonds will be predicted into either 0 or 1 class.

```
> cost <- cut(cost, breaks = c(0,4000,1000000), labels = c(0,1) )
> head(cost)
[1] 0 0 0 0 0 0
Levels: 0 1
```

Splitting of data:

For performing classification, we need to split the data into training set and testing set. As the dataset is huge, hold out evaluation will be used for this purpose. Following screenshot shows the code for **Hold out evaluation** and the no of rows after splitting the data. Before splitting the data set, sample function is performed on the data. This shuffles the dataset rows to give a better set of data.

Building a model:

To build a random forest model we have used a random Forest function. This model is built on the training data. The output shows that the random forest has used classification technique. The number of trees used were 500. These number of trees can be changes also. The data as we know was split into two type. The OOB estimate for this model was given, it is the Out of Bag error, it is 2.35%, which is very good. The confusion Matrix mentioned in the output of summary show the error the model did while classifying. The attributes for building the random forest model are shown in the screen shot.

```
Call:
 randomForest(formula = train.rf$price ~ ., data = train.rf)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 2.35%
Confusion matrix:
      0     1 class.error
0 26984   582  0.02111297
1   431 15155  0.02765302
>
```

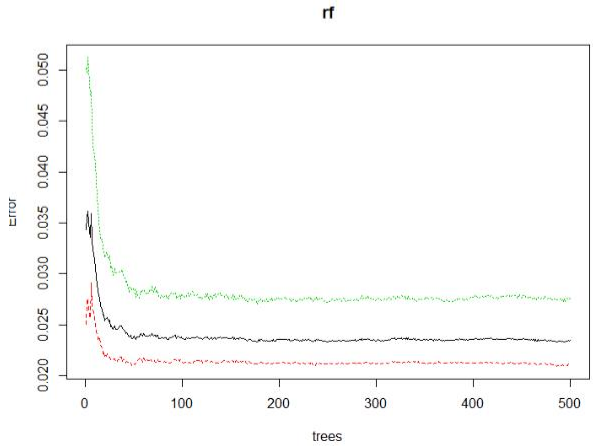
```
> attributes(rf)
$names
 [1] "call"            "type"            "predicted"
 [4] "err.rate"        "confusion"       "votes"
 [7] "oob.times"       "classes"         "importance"
[10] "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"
[16] "y"               "test"            "inbag"
[19] "terms"

$class
[1] "randomForest.formula" "randomForest"
```

Prediction and accuracy:

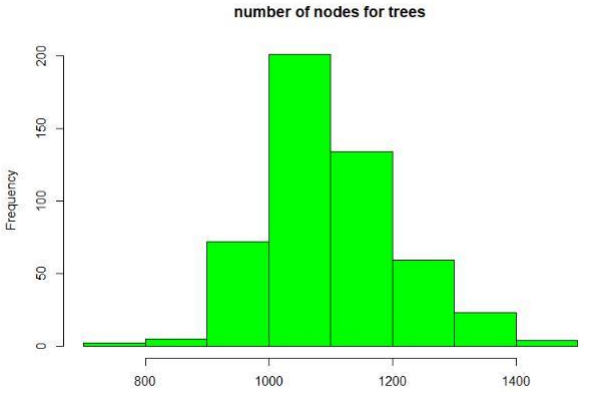
To calculate the accuracy of the model we have used the confusion matrix. We have used the predict () function to predict the test output. The parameters are the random forest model and the testing dataset. To evaluate the model, we are using a **confusion matrix.** The accuracy of this model is 97.6%. On the top left corner there is an expensive and inexpensive table. It tells that there are 6854 Inexpensive observations which were correctly predicted by our model into the inexpensive label and there are 3675 expensive observations correctly predicted. Whereas 118 expensive observations were wrongly predicted as inexpensive and vice versa for 141 observations. The correct prediction number has increased from what it was for the naïve Bayes and SVM models. Accuracy is given by sum of the correctly predicted values / the total observations. The CI entry tells that the model is 95% confident of giving an accuracy between 97.29% and 97.88%. P value of the model is less than alpha, which makes it a good model. The sensitivity and specificity manage the incorrect predictions.

```
Confusion Matrix and Statistics


pre_rf    0    1
     0 6854  118
     1  141 3675

               Accuracy : 0.976
                 95% CI : (0.9729, 0.9788)
    No Information Rate : 0.6484
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9474

 Mcnemar's Test P-Value : 0.1716

            Sensitivity : 0.9798
            Specificity : 0.9689
         Pos Pred Value : 0.9831
         Neg Pred Value : 0.9631
             Prevalence : 0.6484
         Detection Rate : 0.6353
   Detection Prevalence : 0.6463
      Balanced Accuracy : 0.9744

       'Positive' Class : 0
```

A snapshot for the first few value predictions is given. The error rate for the model is shown. According to it initially the error is very high, but it drops till 50; after 50 trees there is a stable graph, which tells us no improvement is seen.



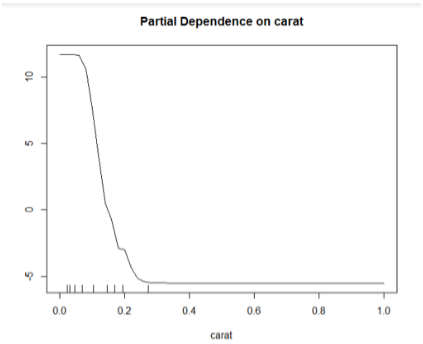
The number of trees is shown in a histogram. It tells us that there are maximum 200 trees that have 1100 number of nodes.

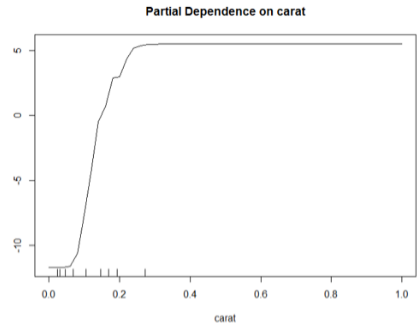


The varUsed function tells us how many times the feature variables are used. There are 6 values for 6 different features. It tells that depth has been used maximum times for classification.

```
> varUsed(rf)
[1] 126974  49031  73502  53449 146560 100889
```

We have plotted the graph for partial dependence for the carat variable for each 0 and 1 label. When carat is less than 0.2 it predicts class 0 more strongly and When carat is greater than 0.2 it predicts class 1 more strongly.





Then we have ran the code to analyze the first tree. It shows us how the tree is built, how many nodes it has and where does it terminate and what decision of label is taken for that specific node. When the status column converts into -1 the tree has terminated and the prediction column adjacent to it will show what prediction is taken for it.

```
> getTree(rf, 1, labelVar = T)
   left daughter right daughter split var split point status prediction
1              2              3     carat  0.15904366      1       <NA>
2              4              5     table  0.24903846      1       <NA>
3              6              7     table  0.34326923      1       <NA>
4              8              9     carat  0.13409563      1       <NA>
5             10             11   clarity 52.00000000      1       <NA>
6             12             13       cut  1.00000000      1       <NA>
7             14             15   clarity  9.00000000      1       <NA>
8             16             17   clarity 63.00000000      1       <NA>
9             18             19       cut 11.00000000      1       <NA>
10            20             21       cut 13.00000000      1       <NA>
11            22             23     depth  0.44583333      1       <NA>
12            24             25     color 34.00000000      1       <NA>
13            26             27   clarity  1.00000000      1       <NA>
14            28             29   clarity  1.00000000      1       <NA>
15            30             31       cut  1.00000000      1       <NA>
16            32             33     table  0.22980769      1       <NA>
17            34             35     carat  0.11538462      1       <NA>
18            36             37     carat  0.15072765      1       <NA>
19            38             39     color 31.00000000      1       <NA>
20            40             41     color 31.00000000      1       <NA>
21            42             43       cut  2.00000000      1       <NA>
22            44             45     color  2.00000000      1       <NA>
23            46             47     carat  0.14033264      1       <NA>
24            48             49     depth  0.40972222      1       <NA>
25            50             51     carat  0.20893971      1       <NA>
26            52             53     carat  0.22557173      1       <NA>
27            54             55     carat  0.17983368      1       <NA>
28            56             57     carat  0.24324324      1       <NA>
```

The accuracy of the random forest model is 97.6%

5.2. Evaluations and Results

There are 4 models build for our problem. The prediction of exact values was done using a multiple linear regression model. The multiple linear regression model was built after preforming the data preprocessing, Checking the relation between the x and y variables, splitting of data into train and test data, Building the regression model using feature selection, Goodness of fit, Residual analysis, Measuring predictive performance and Postprocessing. The model was checked for errors with the root mean square value. **The accuracy of multiple linear regression model was found out to be around 89%.**

The next problem we addressed was the classification of the diamonds as either Expensive or Inexpensive. There are four classification models (K nearest neighbor, Naïve Bayes, Support Vector Machines, Random

forest) built to perform the same task of classification of the diamonds into the labels specified. These models were trained on the train dataset and then tested on test data set the predict function was used to predict the outcomes for the test dataset. For the KNN model, accuracy function was used to find the accuracy of the model. There were three KNN models with different values for the neighboring dataset. The values for k was 133, 155 and 211. The accuracy for each model is; for k = 133 accuracy is 0.9013, for k = 155 accuracy is 0.89, for k as 211 accuracy is 0.8627. Among this the best accuracy value is given by the model with 133 neighbors. **The accuracy for knn is 90.13%.** The naïve Bayes model showed that there were 812 misclassifications. To find out the accuracy for the Naïve Bayes a confusion matrix was used according to that **the accuracy of a naïve Bayes classification model is 92.47%.** The Support Vector Machine model showed that there were 356 misclassifications. To find out the accuracy for the SVM a confusion matrix was used according to that **the accuracy of a Support Vector Machine classification model is 96.7%.** The random forest model showed that there were 259 misclassifications. To find out the accuracy for the random forest a confusion matrix was used according to that **the accuracy was 97.6%.** It is also seen from the confusion Matrix that the major class in the data set is the Inexpensive class. The data set has many entries for Inexpensive diamonds.

Based on these observations for the accuracy values for each of the model, it is concluded that the best accuracy to predict the diamond cost is given by **Random Forest** for the diamond's dataset.

Comparing the accuracy for each model:

SVM

```
> confusionMatrix(table(pre_svm,test.svm$price))
Confusion Matrix and Statistics


pre_svm       Inexpensive Expensive
  Inexpensive        6661       118
  Expensive           238      3771

               Accuracy : 0.967
                 95% CI : (0.9635, 0.9703)
    No Information Rate : 0.6395
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9289

 Mcnemar's Test P-Value : 2.845e-10

            Sensitivity : 0.9655
            Specificity : 0.9697
         Pos Pred Value : 0.9826
         Neg Pred Value : 0.9406
             Prevalence : 0.6395
         Detection Rate : 0.6174
   Detection Prevalence : 0.6284
      Balanced Accuracy : 0.9676

       'Positive' Class : Inexpensive
```

Naïve Bayes

```
> confusionMatrix(table(pre_naive,test.naive$price))
Confusion Matrix and Statistics


pre_naive     Inexpensive Expensive
  Inexpensive        6324       211
  Expensive           601      3652

               Accuracy : 0.9247
                 95% CI : (0.9196, 0.9296)
    No Information Rate : 0.6419
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8398

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9132
            Specificity : 0.9454
         Pos Pred Value : 0.9677
         Neg Pred Value : 0.8587
             Prevalence : 0.6419
         Detection Rate : 0.5862
   Detection Prevalence : 0.6058
      Balanced Accuracy : 0.9293

       'Positive' Class : Inexpensive
```

KNN

```
> train.def <- train.dataknn$price
> test.def <- test.dataknn$price
> knn.133 <- knn(train.knn, test.knn, train.def, k=133)
Error in knn(train.knn, test.knn, train.def, k = 133) :
  could not find function "knn"
> library(class)
Warning message:
package 'class' was built under R version 3.6.3
> knn.133 <- knn(train.knn, test.knn, train.def, k=133)
> knn.155 <- knn(train.knn, test.knn, train.def, k=155)
> knn.211 <- knn(train.knn, test.knn, train.def, k=211)
> #install.packages("Metrics", dependencies = TRUE)
> library(Metrics)

Attaching package: 'Metrics'

The following objects are masked from 'package:caret':

    precision, recall

Warning message:
package 'Metrics' was built under R version 3.6.3
> accuracy(test.def, knn.133)
[1] 0.9013719
> accuracy(test.def, knn.155)
[1] 0.890063
> accuracy(test.def, knn.211)
[1] 0.8627178
>
```

Multiple Linear Regression

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               5727.729    396.106  14.460  < 2e-16 ***
train.data1$carat         8615.753     14.579 590.956  < 2e-16 ***
train.data1$cutGood        -38.400     24.595  -1.561    0.118
train.data1$cutPremium      95.585     17.699   5.401 6.67e-08 ***
train.data1$`cutVery Good`  90.961     17.235   5.278 1.31e-07 ***
train.data1$colorE         376.604     19.619  19.196  < 2e-16 ***
train.data1$colorF         278.404     19.651  14.168  < 2e-16 ***
train.data1$colorG          97.087     18.965   5.119 3.08e-07 ***
train.data1$colorH        -349.722     20.597 -16.979  < 2e-16 ***
train.data1$colorJ        -1615.792     30.377 -53.191  < 2e-16 ***
train.data1$clarityIF      1929.624     37.014  52.132  < 2e-16 ***
train.data1$claritySI2     -710.606     19.555 -36.339  < 2e-16 ***
train.data1$clarityVS1     1062.737     20.307  52.335  < 2e-16 ***
train.data1$clarityVS2      815.480     17.968  45.385  < 2e-16 ***
train.data1$clarityVVS1    1557.832     27.573  56.498  < 2e-16 ***
train.data1$clarityVVS2    1529.518     24.112  63.433  < 2e-16 ***
train.data1$depth           -85.513      4.678 -18.279  < 2e-16 ***
train.data1$table           -69.027      3.445 -20.035  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1284 on 43129 degrees of freedom
Multiple R-squared:  0.8964,    Adjusted R-squared:  0.8964
F-statistic: 2.196e+04 on 17 and 43129 DF,  p-value: < 2.2e-16
```

Random Forest

```
Confusion Matrix and Statistics


pre_rf    0    1
     0 6854  118
     1  141 3675

               Accuracy : 0.976
                 95% CI : (0.9729, 0.9788)
    No Information Rate : 0.6484
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9474

 Mcnemar's Test P-Value : 0.1716

            Sensitivity : 0.9798
            Specificity : 0.9689
         Pos Pred Value : 0.9831
         Neg Pred Value : 0.9631
             Prevalence : 0.6484
         Detection Rate : 0.6353
   Detection Prevalence : 0.6463
      Balanced Accuracy : 0.9744

       'Positive' Class : 0
```

5.3. Findings

The main task was to find the best model for performing prediction. Five different models were built in order to do so. The accuracy of each model was used to compare the best prediction model for the diamonds data set. The accuracy for the Multiple Linear regression was its Adjacent R2 value, for the KNN model the accuracy matrix was used, for naïve Bayes, Random forest and SVM we used the confusion matrix. The Random forest gave the highest accuracy. From the confusion matrix output it was clear the number of misclassifications had reduced to a very small amount. This model had the maximum matching predictions and the observation categories same. The second best was given by SVM. As the data had linear relationship, a linear kernel was used. The next-best model was the Naïve Bayes, it had a greater number of misclassifications than the SVM. It is followed by the KNN model the accuracy of the KNN model was satisfactory it was almost near to the value of accuracy we got from the Multiple Regression model. The multiple Linear regression model need a lot of analysis to make it a good model.

6. Conclusions and Future Work

6.1. Conclusions

The main motive to build this project was to be able to predict the values or the class for a diamond with its other features or attributes listed for it. The prediction model build will be used to predict the exact cost of diamond based on the independent attributes listed. Where ass the classification models will be used for classifying the price of the diamonds into categories like Expensive or Inexpensive. These models could be used in market to predict the cost of the diamond to be sold. The cost of the diamond will change with respect to the changing features of the diamond. This change of cost will be very useful if could be predicted beforehand. It would help in giving a shape or size to the diamond to increase or decrease its worth. Based on the models built, the models are compared with respective to their accuracy value. This value tells us how much accurate and reliable our model is, how much perfect its predictions will be. As a real-life project, it will not be dependable if the value of accuracy of prediction is low. This would mean that with every decreasing value of accuracy there is an increase in the value of mistake. After comparing the accuracy values for all the four models built, it is concluded that the Random forest model is the best model for prediction for the diamond's dataset.

6.2. Limitations

In the premise of supervised continuous algorithm only one model was explored. Another models like Decision tree or random forest could have been explored. In the Multiple Linear regression model a few assumptions were made while selecting the best regression model (as the values of Adjacent r2 was same for the models). In the knn model only three values of k were explored, there could be other k values which give us a better accuracy for the model. In the Support Vector Machine, the kernel was set to a linear function. Tuning for the random forest was skipped as the R studio did not support its execution, but the accuracy we got for random forest was still the best.

6.3. Potential Improvements or Future Work

Different types of supervised algorithm can be tested on the dataset to give us different accuracy values. Unsupervised learning algorithms could be used for prediction purpose. The variations in the value of the price with the change in the value of every variable could be traced using various visualization tools. Different features can be added to the model and then its accuracy can be checked. Tuning for the models can be performed.