Linear Regression

Sovanna Ramirez & Sanjana Jadhav

2023-02-17

In this notebook, linear regression will be explored through various practices on a given data set. Before getting into data exploration let's first understand what linear regression is and how it works. Linear regression is a machine learning algorithm most commonly used for statistics. With linear regression we are trying to determine if there is a linear relationship between predictor (x) and target (y). Some of the benefits of linear regression is that it is both simple and powerful, it works well when data follows a linear pattern, and it has low variance. The downside to linear regression is that it has high bias due to the fact that the model inherently follows a linear shape to the data.

Data Exploration

This example looks at the data set **Diamonds** as an intro into linear regression. The data set was downloaded from here: https://www.kaggle.com/datasets/shivam2503/diamonds

To start, the data set is read into and stored in a data frame.

```
df <- read.csv("~/Downloads/diamonds.csv")
```

Data Cleaning: Checking for N/A values

```
sapply(df,function(x) sum(is.na(x)==TRUE))
```

```
##       X   carat     cut   color clarity   depth   table   price       x       y
##       0       0       0       0       0       0       0       0       0       0
##       z
##       0
```

Divide into 80/20 train/test: From here we divide the data set. Where 80% of the original data is dedicated to the training set and 20% of the original data is dedicated to the testing set. Once the data is divided, we can use the training set to practice some data exploration...

```
i <- sample(1:nrow(df), nrow(df)*0.80, replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

The summary() function

With the **summary()** function we are provided with the summary statistics of the data from the training set. Our variables for this data set include carat (weight of the diamond), cut, color, clarity, depth, table, price, length in mm (x), width in mm (y), and depth in mm (z).

```
summary(train)
```

```
##        X             carat             cut                color
##  Min.   :    1   Min.   :0.2000   Length:43152       Length:43152
##  1st Qu.:13476   1st Qu.:0.4000   Class :character   Class :character
##  Median :26962   Median :0.7000   Mode  :character   Mode  :character
```

```
##  Mean   :26955   Mean   :0.7976
##  3rd Qu.:40413   3rd Qu.:1.0400
##  Max.   :53940   Max.   :5.0100
##    clarity              depth           table           price
##  Length:43152       Min.   :43.00   Min.   :43.00   Min.   :  326
##  Class :character   1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  946
##  Mode  :character   Median :61.80   Median :57.00   Median : 2409
##                     Mean   :61.75   Mean   :57.45   Mean   : 3939
##                     3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5329
##                     Max.   :79.00   Max.   :95.00   Max.   :18823
##        x                y               z
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
##  Median : 5.700   Median : 5.710   Median : 3.520
##  Mean   : 5.731   Mean   : 5.734   Mean   : 3.539
##  3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :10.740   Max.   :58.900   Max.   :31.800
```

The head() function

The **head()** function will output the first n rows from the training set. In our case, the head() function will print out the first 20 rows.

```
head(train, n=20)
```

```
##           X carat       cut color clarity depth table price    x    y    z
## 13960 13960  1.02      Fair     E     SI1  65.7    58  5681 6.23 6.31 4.12
## 5184   5184  0.90   Premium     G     SI1  61.3    60  3774 6.17 6.14 3.77
## 28071 28071  0.31     Ideal     G    VVS2  60.0    58   661 4.38 4.42 2.64
## 43495 43495  0.54     Ideal     E     SI1  61.3    56  1422 5.22 5.25 3.21
## 16668 16668  1.06 Very Good     G     VS1  62.8    54  6663 6.50 6.56 4.10
## 25989 25989  1.54     Ideal     G    VVS2  62.5    57 15197 7.36 7.30 4.58
## 19257 19257  1.39   Premium     H     SI1  62.7    58  7986 7.15 7.08 4.46
## 25926 25926  1.01      Good     D      IF  63.4    59 15081 6.26 6.39 4.01
## 29857 29857  0.30     Ideal     E     VS1  60.8    57   710 4.33 4.36 2.64
## 27851 27851  0.34     Ideal     H    VVS2  60.9    55   652 4.52 4.54 2.75
## 42002 42002  0.51 Very Good     F     SI1  62.0    54  1265 5.15 5.18 3.20
## 48603 48603  0.72     Ideal     J    VVS2  61.6    56  1996 5.76 5.73 3.54
## 29582 29582  0.41 Very Good     E     SI1  61.8    60   705 4.74 4.78 2.94
## 25938 25938  1.70   Premium     D     SI1  61.9    59 15100 7.61 7.64 4.72
## 8351   8351  1.05 Very Good     E     SI2  62.9    57  4398 6.47 6.51 4.08
## 36858 36858  0.50      Good     D     SI2  63.1    56   958 5.05 4.96 3.16
## 268     268  0.70   Premium     F     VS1  60.7    60  2792 5.78 5.75 3.50
## 911     911  0.71     Ideal     I     VS2  61.5    55  2878 5.76 5.78 3.55
## 30681 30681  0.40     Ideal     E     SI1  62.2    55   737 4.71 4.75 2.94
## 52354 52354  0.70 Very Good     G     VS2  59.6    60  2501 5.71 5.80 3.43
```

The tail() function

Similar to the head() function, the **tail()** function will output the last n rows from the training set. Here the tail() function will print out the last 20 rows of our training set.

```
tail(train, n=20)
```

```
##           X carat       cut color clarity depth table price    x    y    z
## 41295 41295  0.34     Ideal     D    VVS1  61.7    56  1211 4.48 4.46 2.76
```

```
## 9082   9082  1.03     Ideal     E     SI2  62.5    56  4522 6.48 6.45 4.04
## 2424   2424  0.63     Ideal     E    VVS1  61.1    58  3181 5.49 5.54 3.37
## 17241 17241  1.52      Good     J     SI1  63.6    60  6897 7.21 7.25 4.60
## 27191 27191  1.70      Good     D     VS2  63.6    56 17485 7.56 7.50 4.79
## 42094 42094  0.45 Very Good     H    VVS1  62.3    57  1274 4.90 4.93 3.06
## 10225 10225  1.00   Premium     F     VS2  58.7    56  4743 6.46 6.42 3.78
## 6917   6917  0.90   Premium     G     VS2  62.0    60  4137 6.19 6.14 3.82
## 43140 43140  0.58      Good     G     SI1  63.8    56  1388 5.30 5.33 3.39
## 20628 20628  1.54     Ideal     G     SI2  62.3    60  8923 7.29 7.34 4.56
## 36273 36273  0.30     Ideal     G     SI1  61.4    57   477 4.28 4.32 2.64
## 40128 40128  0.41     Ideal     F    VVS2  62.5    56  1115 4.72 4.75 2.96
## 14647 14647  1.05     Ideal     D     SI1  61.7    56  5914 6.53 6.59 4.05
## 15682 15682  1.22   Premium     G     SI1  62.3    58  6288 6.83 6.76 4.23
## 48869 48869  0.78 Very Good     J     SI1  59.4    62  2035 6.01 6.05 3.58
## 48731 48731  0.52     Ideal     G    VVS2  61.7    56  2012 5.16 5.14 3.18
## 49936 49936  0.71 Very Good     I     VS2  60.4    60  2185 5.72 5.80 3.48
## 10187 10187  1.24 Very Good     J     SI2  60.4    61  4737 6.85 6.92 4.16
## 2827   2827  0.90      Good     F     SI2  64.3    57  3267 6.06 6.16 3.93
## 53096 53096  0.75     Ideal     I     VS1  63.0    57  2613 5.80 5.82 3.66
```

The str() function

The **str()** function will output the structure of the data frame. The str() function provides us with the number of observations or instances in our training set. As well as the data type for each of the variables.

```
str(train)
```

```
## 'data.frame':    43152 obs. of  11 variables:
##  $ X      : int  13960 5184 28071 43495 16668 25989 19257 25926 29857 27851 ...
##  $ carat  : num  1.02 0.9 0.31 0.54 1.06 1.54 1.39 1.01 0.3 0.34 ...
##  $ cut    : chr  "Fair" "Premium" "Ideal" "Ideal" ...
##  $ color  : chr  "E" "G" "G" "E" ...
##  $ clarity: chr  "SI1" "SI1" "VVS2" "SI1" ...
##  $ depth  : num  65.7 61.3 60 61.3 62.8 62.5 62.7 63.4 60.8 60.9 ...
##  $ table  : num  58 60 58 56 54 57 58 59 57 55 ...
##  $ price  : int  5681 3774 661 1422 6663 15197 7986 15081 710 652 ...
##  $ x      : num  6.23 6.17 4.38 5.22 6.5 7.36 7.15 6.26 4.33 4.52 ...
##  $ y      : num  6.31 6.14 4.42 5.25 6.56 7.3 7.08 6.39 4.36 4.54 ...
##  $ z      : num  4.12 3.77 2.64 3.21 4.1 4.58 4.46 4.01 2.64 2.75 ...
```

The dim() function

The **dim()** function will output the dimensions of the data frame in terms of rows and columns. There are 43152 rows and 11 variables in our training set.

```
dim(train)
```

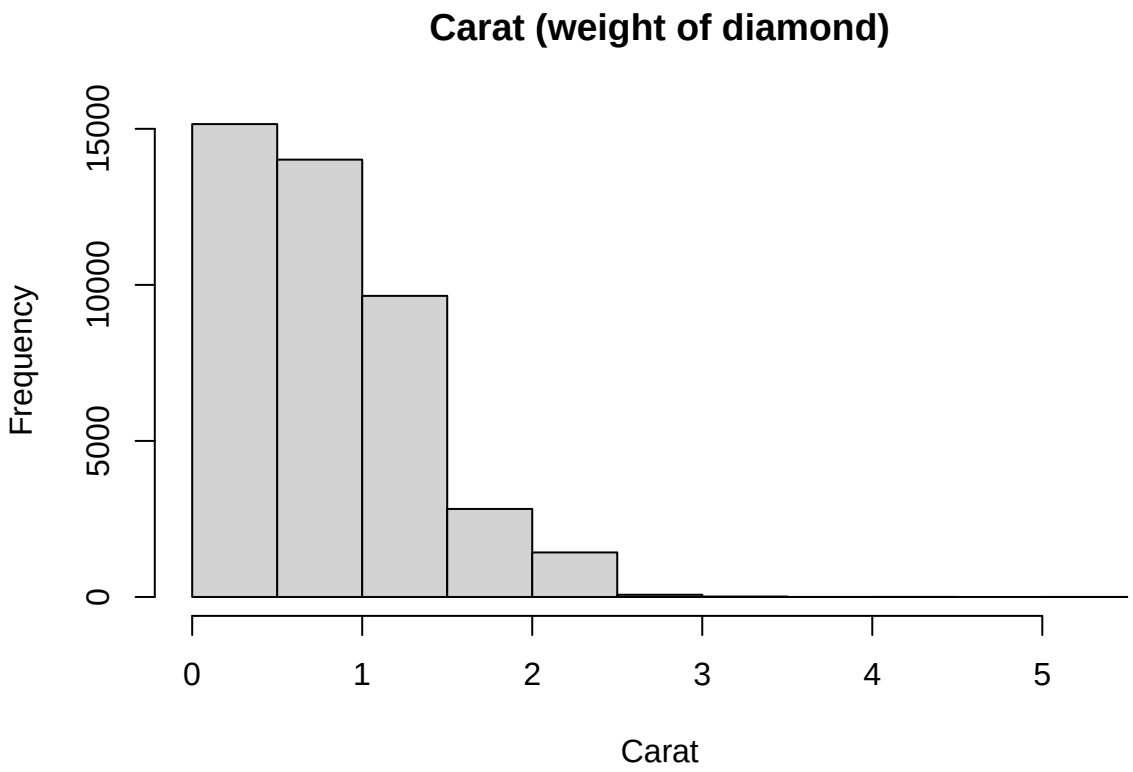
```
## [1] 43152    11
```

Data Visualization

Now that we have done some data exploration and sought to understand a few functions in r, we can go into data visualization. Data visualization allows us to understand data through visual representations. To start, let's explore some simple graphs using graphing functions in R.

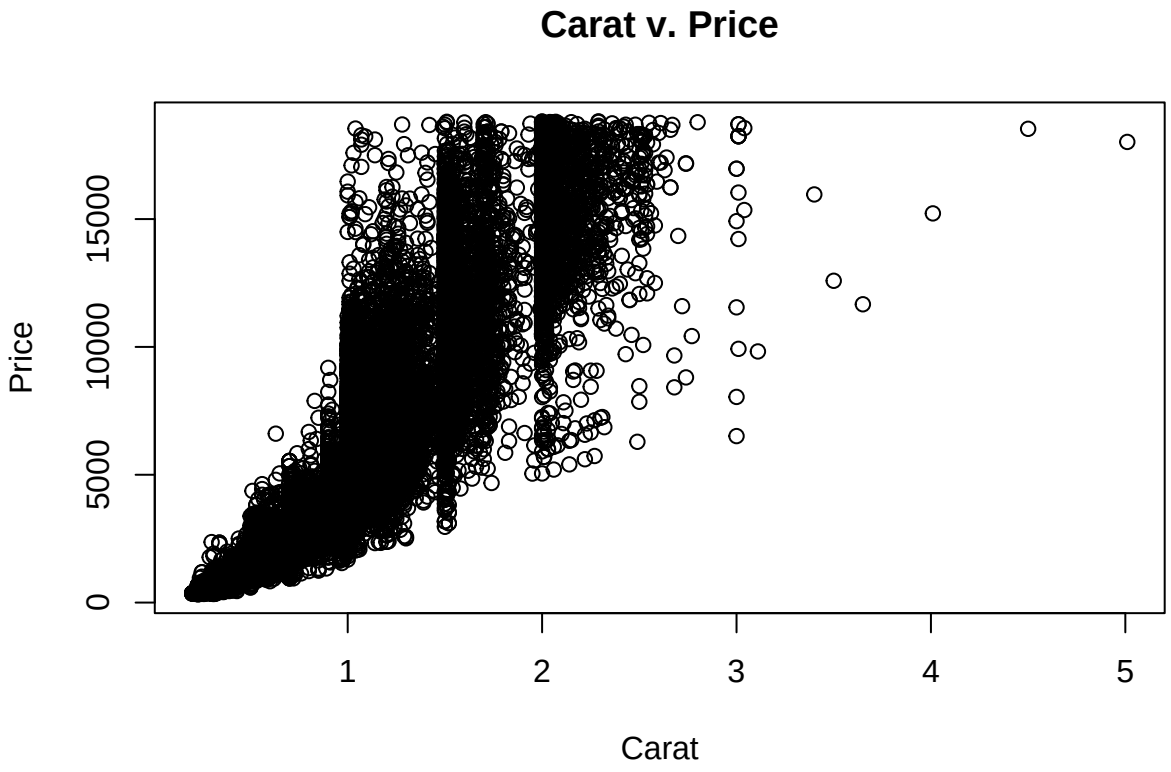
For the first graph, the **hist()** function will be used to create a histogram with carat as its parameter. Carat is the weight of a diamond ranging from 0.2-5.01.

```
hist(train$carat, main="Carat (weight of diamond)",
xlab="Carat")
```



For our second graph, the **plot()** function will be used to create a scatter plot with parameters carat and price. In this plot we start to see a potential correlation between the price of a diamond and the carat or weight of a diamond.

```
plot(train$carat, train$price, main="Carat v. Price", xlab = "Carat", ylab = "Price")
```



Simple Linear Regression Model

Now using the training data, we build a simple linear regression model which demonstrates how *carat* (weight of a diamond) has an affect on the *price* of a diamond. Our summary of lm1 provides the statistics to better understand the relationship between the predictor and our target.

- **Residual standard error** is meant to be a smaller number. For this demonstration, our RSE was 1547. A large RSE means that our data points are far from the fitted regression line.
- Typically the **r-squared value** should be closer to 1 to represent the "goodness of fit." In our case the data does fit the linear regression model decently with an r-squared value of 0.8507
- The **f-statistic and p-value** can be used together to determine how confident the model is. With our f-statistic at 2.458e+05 and a low p-value we may conclude that there is confidence in the model.

```
lm1 <-lm(price~carat, data=train)
summary(lm1)
```

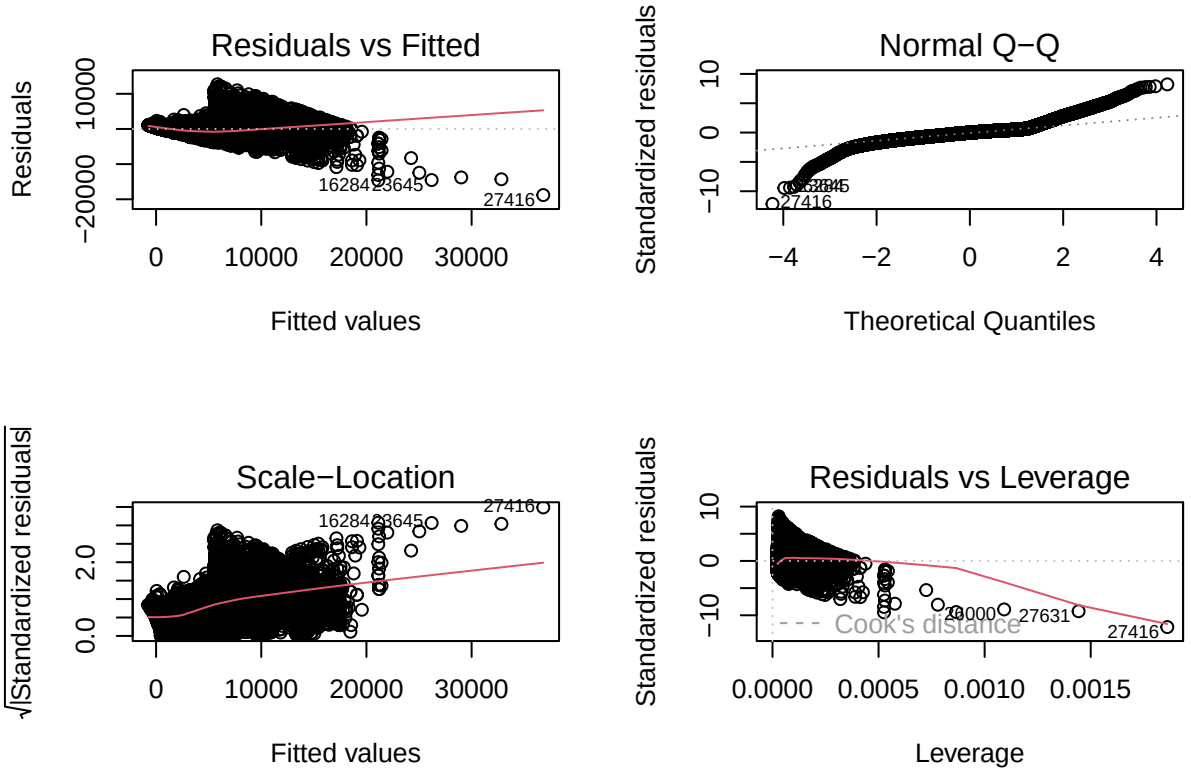
```
##
## Call:
## lm(formula = price ~ carat, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18798.2   -812.3    -15.3    549.2  12710.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2285.64      14.60  -156.6   <2e-16 ***
## carat        7804.76      15.74   495.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1547 on 43150 degrees of freedom
## Multiple R-squared:  0.8507, Adjusted R-squared:  0.8507
## F-statistic: 2.458e+05 on 1 and 43150 DF,  p-value: < 2.2e-16
```

Now to plot the residuals...

- The **Residuals vs Fitted** plot demonstrates no particular pattern with residuals which are not equally spread out. Thus, this plot indicates that we may not have a non-linear relationship between our predictor (carat) and our outcome (price).
- The **Normal Q-Q** plot shows that the residuals deviate from the line pretty severely. Thus, indicating that the residuals are not normally distributed.
- The **Scale-Location** plot shows that the line is not horizontal and the points are not equally spread out. We can conclude that the residuals are not spread equally among the range of predictors.
- The **Residuals vs Leverage** plot shows that there are cases which may be influential to the regression results. These cases are found at the lower right corner and if excluded will change our regression results.

```
par(mfrow=c(2,2))
plot(lm1)
```



Multiple Linear Regression Model

Using the training data, we will explore a multiple linear regression model with multiple predictors. First, let's explore how depth and carat affect the price of a diamond.

```
lm2 <- lm(price~depth + carat, data=train)
summary(lm2)
```

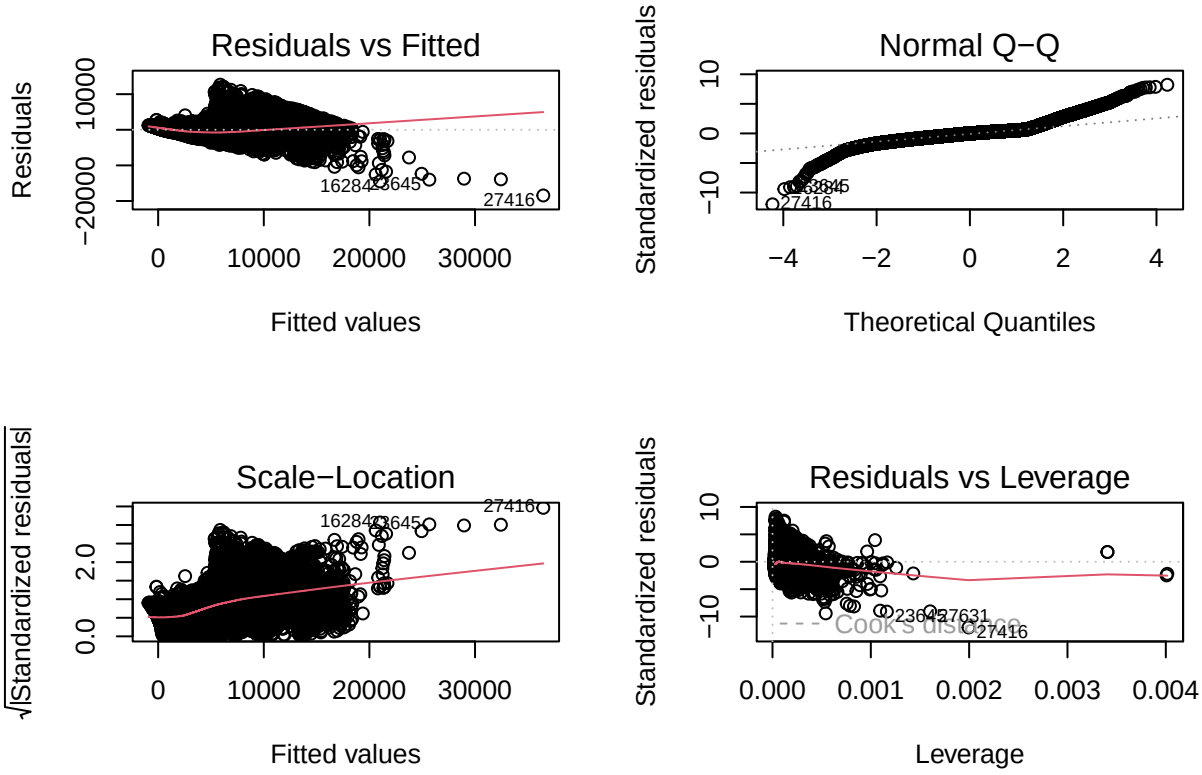
```
##
## Call:
## lm(formula = price ~ depth + carat, data = train)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18445.8   -808.3    -16.5    553.7  12662.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4062.679    319.711   12.71   <2e-16 ***
## depth       -102.914      5.178  -19.88   <2e-16 ***
## carat       7812.775     15.677  498.37   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1540 on 43149 degrees of freedom
## Multiple R-squared:  0.852,  Adjusted R-squared:  0.852
## F-statistic: 1.242e+05 on 2 and 43149 DF,  p-value: < 2.2e-16
```

Now to plot the residuals of the multiple linear regression model lm2...

After plotting the residuals on the training set it is still quite tricky to tell if the models fit the data well or not. However, we do see a slight improvement from lm1 (a simple linear regression model) to lm2 (a multiple linear regression model) using the RSE and r-squared values.

```
par(mfrow=c(2,2))
plot(lm2)
```



Now let's explore a different combination of predictors that may improve our results for our multiple linear regression model.

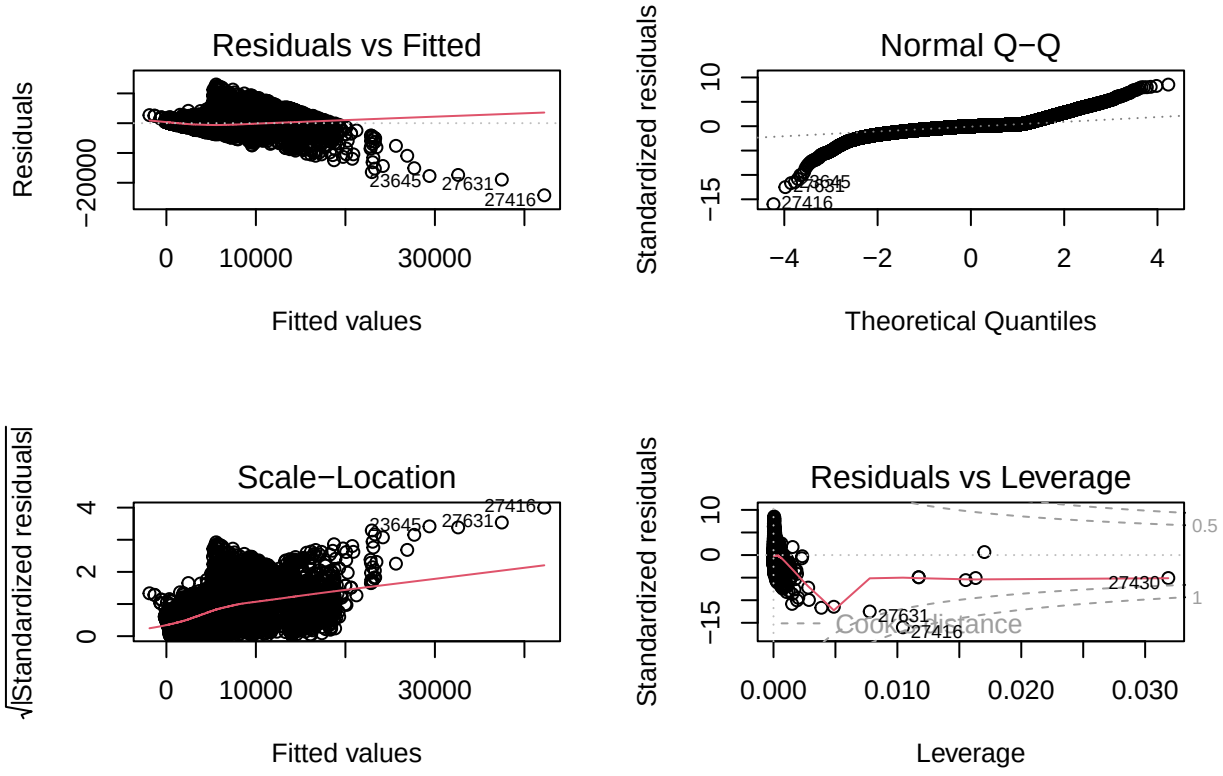
In this next model lm3 we will see how length in mm (x) and carat affect the price of a diamond.

```
lm3 <- lm(price~x + carat, data=train)
summary(lm3)
```

```
##
## Call:
## lm(formula = price ~ x + carat, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24201.7   -646.3    -10.4    363.0  12982.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2147.76     116.52   18.43   <2e-16 ***
## x           -1140.99      29.76  -38.34   <2e-16 ***
## carat       10444.35      70.56  148.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1521 on 43149 degrees of freedom
## Multiple R-squared:  0.8556, Adjusted R-squared:  0.8556
## F-statistic: 1.278e+05 on 2 and 43149 DF,  p-value: < 2.2e-16
```

The residual plots for lm3 seem to show some improvement. Again, looking at the summary statistics we can see that lm3 is an improvement upon lm2.

```
par(mfrow=c(2,2))
plot(lm3)
```



Summary of results

When comparing all three linear models lm1, lm2, and lm3 lm3 is the best model out of the three. To recap, lm1 was our simple linear regression model in which we wanted to examine if there was a linear relationship

between the carat, weight of a diamond, and the price of a diamond. Moving to lm2, lm2 examines how the combination of the depth and carat influence the price of a diamond. And finally lm3, examines how the combination of the length in mm (x) and the carat influence the price of a diamond. Although the residual plots were quite tricky to understand due to the mass amount of data we were performing on, the summary statistics provided us a clearer understanding of which was the best model. And to conclude, lm3's summary provided us the statistics needed to determine it was the best model. It had the lowest residual standard error of 1524, an r-squared closest to 1 with 0.855, and high f-statistic with low p-value. Thus, lm3 was the best model out of the three.

Predict and evaluate on test data

Using the three models lm1, lm2, and lm3 we can now evaluate our test data set using metrics correlation and MSE. These metrics will help us evaluate our prediction accuracy. Moreover, correlation helps us to understand how strong a relationship between variables is. With a correlation close to +1 would indicate a positive correlation between variables. While mean square error (MSE) helps us understand the amount of error in our regression models. With MSE we take the difference between the actual values and the predicted values. An MSE of 0 indicates no errors. Now, since our models were not the best representations of a good fit, we can expect to get poor results when calculating correlation and mse.

After performing metrics correlation and MSE we see that lm3 has the highest correlation closest to +1 with 0.92 and the lowest MSE. Although lm3's MSE is lowest in comparison to the other models, this model may not be able to make accurate predictions of the target variable. For future improvements in multiple linear regression models, we may want to consider including more than two predictors.

lm1:

```
pred1 <- predict(lm1, newdata=test)
```

```
pred1 <- predict(lm1, newdata=test)
cor1 <- cor(pred1, test$price)
mse1 <- mean((pred1-test$price)^2)
rmse1 <- sqrt(mse1)

print(paste('correlation:', cor1))
```

```
## [1] "correlation: 0.919022455150056"
```

```
print(paste('mse:', mse1))
```

```
## [1] "mse: 2421892.21649605"
```

```
print(paste('rmse:', rmse1))
```

```
## [1] "rmse: 1556.24298118772"
```

lm2:

```
pred2 <- predict(lm2, newdata=test)
cor2 <- cor(pred2, test$price)
mse2 <- mean((pred2-test$price)^2)
rmse2 <- sqrt(mse2)

print(paste('correlation:', cor2))
```

```
## [1] "correlation: 0.919718776754736"
```

```
print(paste('mse:', mse2))
```

```
## [1] "mse: 2401615.1954383"
```

```
print(paste('rmse:', rmse2))
```

```
## [1] "rmse: 1549.71455288976"
```

lm3:

```
pred3 <- predict(lm3, newdata=test)
cor3 <- cor(pred3, test$price)
mse3 <- mean((pred3-test$price)^2)
rmse3 <- sqrt(mse3)

print(paste('correlation:', cor3))
```

```
## [1] "correlation: 0.919472715655536"
```

```
print(paste('mse:', mse3))
```

```
## [1] "mse: 2415720.5352591"
```

```
print(paste('rmse:', rmse3))
```

```
## [1] "rmse: 1554.25883792214"
```