Non-linear regression

Suzana Iacob

18/03/2020

Understanding Diamond Prices

Preprocessing

```
DiamondsTrain = read.csv("diamonds_train.csv")
DiamondsTest = read.csv("diamonds_test.csv")
head(DiamondsTrain)
```

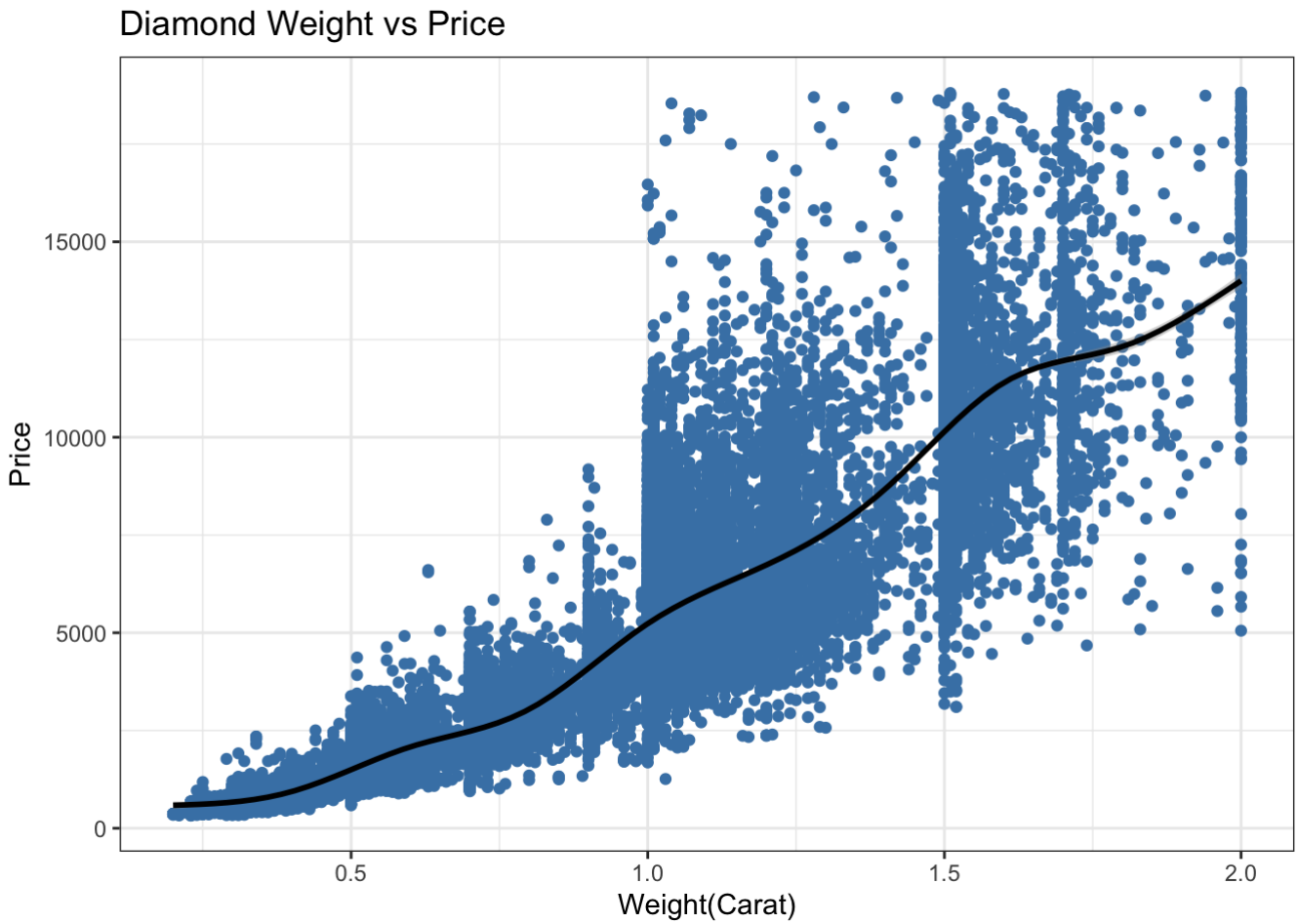
```
##   X carat       cut color clarity    price length width depth
## 1 1  0.23     Ideal     E     SI2 323.7269   3.95  3.98  2.43
## 2 2  0.21   Premium     E     SI1 327.2024   3.89  3.84  2.31
## 3 3  0.23      Good     E     VS1 326.4125   4.05  4.07  2.31
## 4 4  0.29   Premium     I     VS2 335.1308   4.20  4.23  2.63
## 5 5  0.31      Good     J     SI2 335.8074   4.34  4.35  2.75
## 6 7  0.24 Very Good     I    VVS1 338.7874   3.95  3.98  2.47
```

Plotting Diamond Prices

We now plot the price as a function of weight.

```
ggplot(DiamondsTrain, aes(x = carat, y = price)) +
   geom_point(color = "steelblue")  +
   geom_smooth(color = "black") +
   xlab("Weight(Carat)")+ylab("Price") +
   ggtitle("Diamond Weight vs Price") + theme_bw()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



We notice there the general increasing trend, however, there is high variance in price for a particular carat-weight (e.g. at 1.5 carats the price ranges from approximately 3,000 to 18,000). This could be due to other factors such as clarity, cut or color. Let us add clarity to the above plot.

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

is now apparent that for the same carat weight, clarity is a differentiator (e.g. diamonds with clarity VS2 and VS1 tend to be more expensive than those with clarity I1).

Building an Initial Linear Regression Model

We begin with a simple regression model, taking into account the “4 C” variables.

```
DiamondsLM = lm(price ~ carat + cut + color + clarity, data=DiamondsTrain)
summary(DiamondsLM)
```

```
##
## Call:
## lm(formula = price ~ carat + cut + color + clarity, data = DiamondsTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4950.6  -645.1  -186.0   426.2 10429.9
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -6318.64      60.03 -105.26   <2e-16 ***
## carat         8762.74      15.28  573.46   <2e-16 ***
## cutGood        556.71      37.92   14.68   <2e-16 ***
## cutIdeal       876.94      34.66   25.30   <2e-16 ***
## cutPremium     771.21      35.01   22.03   <2e-16 ***
## cutVery Good   743.77      35.34   21.05   <2e-16 ***
## colorE        -243.42      20.05  -12.14   <2e-16 ***
## colorF        -349.35      20.33  -17.19   <2e-16 ***
## colorG        -525.61      19.97  -26.32   <2e-16 ***
## colorH       -1009.40      21.32  -47.35   <2e-16 ***
## colorI       -1491.30      24.23  -61.55   <2e-16 ***
## colorJ       -2227.64      30.56  -72.90   <2e-16 ***
## clarityIF     4571.48      59.55   76.77   <2e-16 ***
## claritySI1    2709.42      51.80   52.30   <2e-16 ***
## claritySI2    1765.45      52.22   33.80   <2e-16 ***
## clarityVS1    3700.20      52.71   70.20   <2e-16 ***
## clarityVS2    3369.85      52.00   64.80   <2e-16 ***
## clarityVVS1   4227.10      55.41   76.28   <2e-16 ***
## clarityVVS2   4110.48      54.14   75.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1059 on 36416 degrees of freedom
## Multiple R-squared:  0.9033, Adjusted R-squared:  0.9032
## F-statistic: 1.889e+04 on 18 and 36416 DF,  p-value: < 2.2e-16
```

The regression model gives us the in-sample R-squared, which is extremely high. This could be due to the 4 C explaining diamond prices extremely well, or we could be overfitting our data.

We now inspect the out-of-sample R-squared, which is similarly high. At first glance, we might conclude that this is an accurate model fitting the data well and having a good performance of the testing set.

```
DiamondsSST = sum((mean(DiamondsTrain$price) - DiamondsTest$price)^2)
DiamondsLMR2 = summary(DiamondsLM)$r.squared
print(summary(DiamondsLM)$r.squared)
```

```
## [1] 0.9032806
```

```
DiamondsLMPredictTest = predict(DiamondsLM, newdata=DiamondsTest)
DiamondsLMSSE = sum((DiamondsLMPredictTest - DiamondsTest$price)^2)
DiamondsLMOSR2 = 1 - DiamondsLMSSE/DiamondsSST
print(DiamondsLMOSR2)
```

```
## [1] 0.9036701
```

We also note that all of the variables meet the 95% significance threshold. We will now attempt to improve the model, while preserving the “4 C” variables which we know to be significant, not only from the data, but from industry expertise as well.

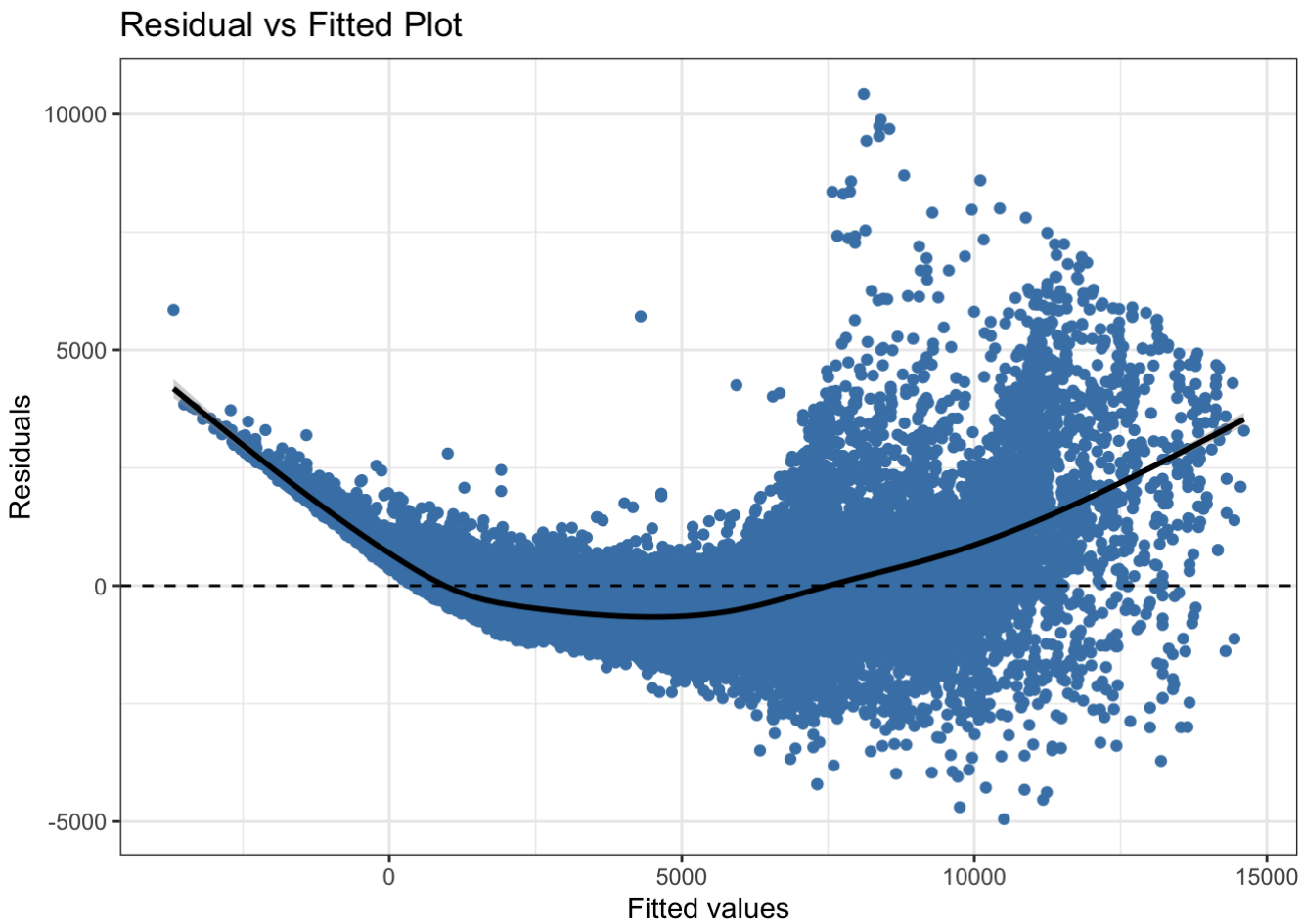
Residual Analysis

We will now plot the residuals of the model as a function of the fitted values (i.e. predicted prices). We also include a horizontal line at x=0. This represents the expected mean of residuals. Residual analysis helps us assess the quality of our model and whether we meet the regression assumptions. Linear regression assumes an underlying linear relationship between the dependent and independent variable, and a normally distributed error term with constant variance.

```
DiamondsLMResiduals = data.frame(predicted = predict(DiamondsLM),
                          residuals = residuals(DiamondsLM))

ggplot(DiamondsLMResiduals,  aes(x = predicted, y = residuals)) +
  geom_point(color = "steelblue") +
  geom_smooth(color = "black") +
  geom_hline(yintercept=0, linetype="dashed") +
  xlab("Fitted values")+ylab("Residuals") +
  ggtitle("Residual vs Fitted Plot") + theme_bw()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Ideally, the residuals plot should appear random across the X-axis. The mean of residuals should be around 0 (marked by the dashed horizontal line). Instead, we see a clear pattern in our residuals - we are overfitting for small and very large diamond prices and underfitting for medium prices.

We also notice heteroskedasticity in the residual plot, i.e. the variance of the residuals is unequal, with low variance at the beginning, and large variance for high diamond prices. Linear regression has a homoscedasticity assumption, and we see that in this case this assumption is violated, hence linear regression in this form may not be an appropriate model. This could also be due to the fact that we have fewer data for very expensive diamonds.

Heuristic Prediction

We would now like to fit a model given by the formula $T A \dot{X}[b6$ where P is the price, W is the weight in carats and α is the typical price of a 1-carat diamond. We first calculate the α value from our data by taking the average price of 1-carat diamonds.

```
pricepercarat = DiamondsTrain$price/DiamondsTrain$carat
alpha = mean(pricepercarat)

DiamondsHeuristicPredictTrain = alpha * (DiamondsTrain$carat ^ 2 )
DiamondsHeuristicSSE1 = sum((DiamondsHeuristicPredictTrain - DiamondsTrain$price)^2)
DiamondsHeuristicSST = sum((mean(DiamondsTrain$price) - DiamondsTrain$price)^2)
DiamondsHeuristicR2 = 1 - DiamondsHeuristicSSE1/DiamondsHeuristicSST

DiamondsHeuristicPredictTest = alpha * (DiamondsTest$carat ^ 2 )
DiamondsLMSSE2 = sum((DiamondsHeuristicPredictTest - DiamondsTest$price)^2)
DiamondsHeuristicSST2 = sum((mean(DiamondsTrain$price) - DiamondsTest$price)^2)
DiamondsHeuristicOSR2 = 1 - DiamondsLMSSE2/DiamondsHeuristicSST2

print(DiamondsHeuristicR2)
```

```
## [1] 0.7798389
```

```
print(DiamondsHeuristicOSR2)
```

```
## [1] 0.7804181
```

The parameter alpha is 3897.054 calculated as the average price/carat. The model uses experts' knowledge and performs relatively well. Lacking sophisticated statistical tools, one could confidently use such a model for a quick approximation.

Natural Splines Model

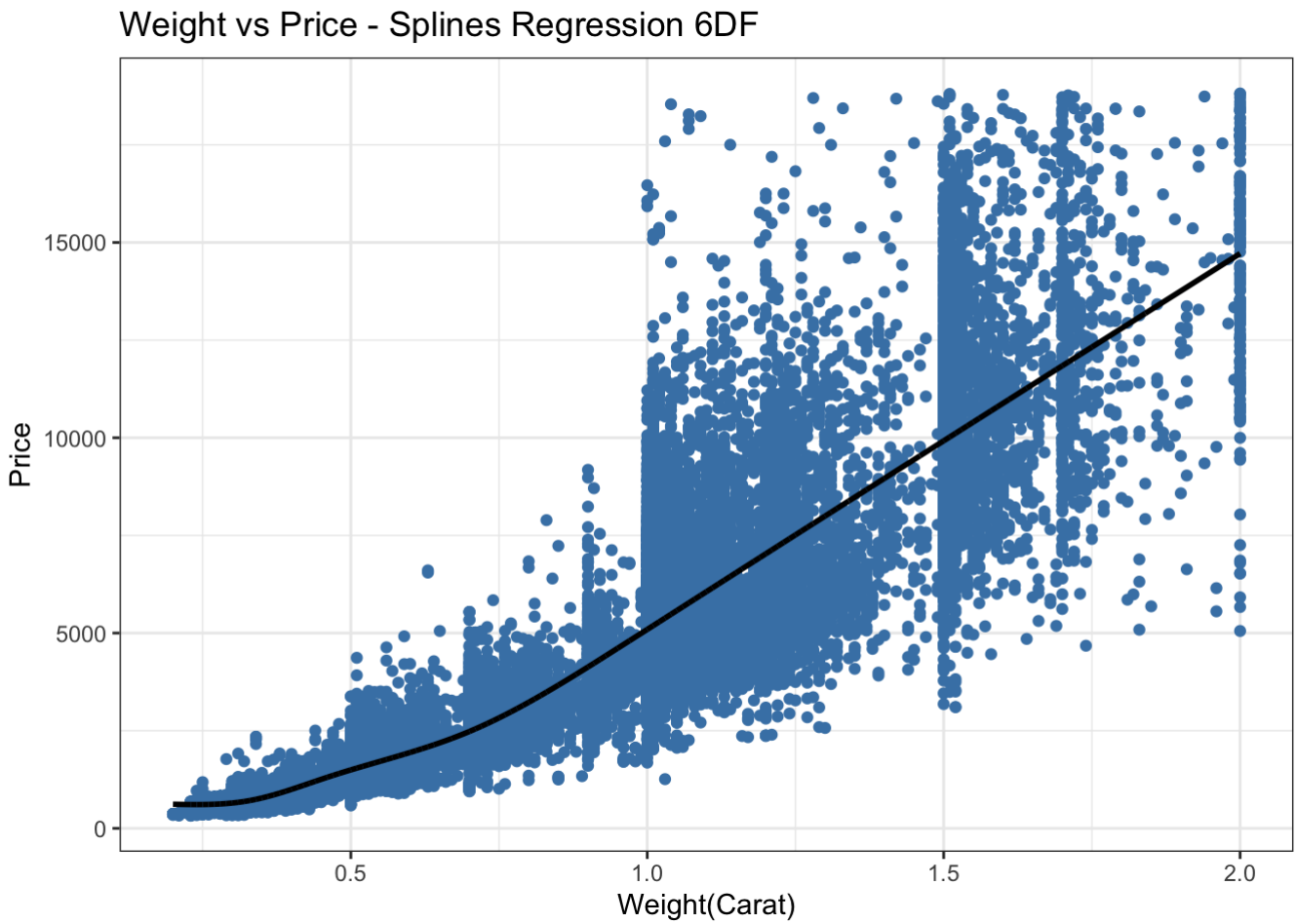
We create a natural (cubic) spline model with 6 degrees of freedom. We evaluate the in-sample and out-of-sample performance.

```
library(splines)
DiamondsSplines=lm(price~ns(carat,6),data=DiamondsTrain)
DiamondsSplinesR2 = summary(DiamondsSplines)$r.squared
print(summary(DiamondsSplines)$r.squared)
```

```
## [1] 0.839658
```

```
DiamondsSplinesPredictTest = predict(DiamondsSplines, newdata=DiamondsTest)
DiamondsSplinesSSE = sum((DiamondsSplinesPredictTest - DiamondsTest$price)^2)
DiamondsSplinesOSR2 = 1 - DiamondsSplinesSSE/DiamondsSST
print(DiamondsSplinesOSR2)
```

```
## [1] 0.8394498
```



The Spline model ourperforms standard regression, but we notice that it still does not model the relationship perfectly well.

General Aditive Model

We also attempt a GAM model, which also performs very well.

```
library(gam)
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.16.1
```

```
DiamondsGAM = gam(price~ns(carat,6) + factor(cut) + factor(color) + factor(clarity),
 data=DiamondsTrain)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts
## argument ignored
```

```
DiamondsGAMPredictTrain = predict(DiamondsGAM, newdata=DiamondsTrain)
DiamondsGAMSSE1 = sum((DiamondsGAMPredictTrain - DiamondsTrain$price)^2)
DiamondsGAMSST = sum((mean(DiamondsTrain$price) - DiamondsTrain$price)^2)
DiamondsGAMR2 = 1 - DiamondsGAMSSE1/DiamondsGAMSST


DiamondsGAMPredictTest = predict(DiamondsGAM, newdata=DiamondsTest)
DiamondsGAMSSE = sum((DiamondsGAMPredictTest - DiamondsTest$price)^2)
DiamondsGAMOSR2 = 1 - DiamondsGAMSSE/DiamondsSST
print(DiamondsGAMR2)
```

```
## [1] 0.9155495
```

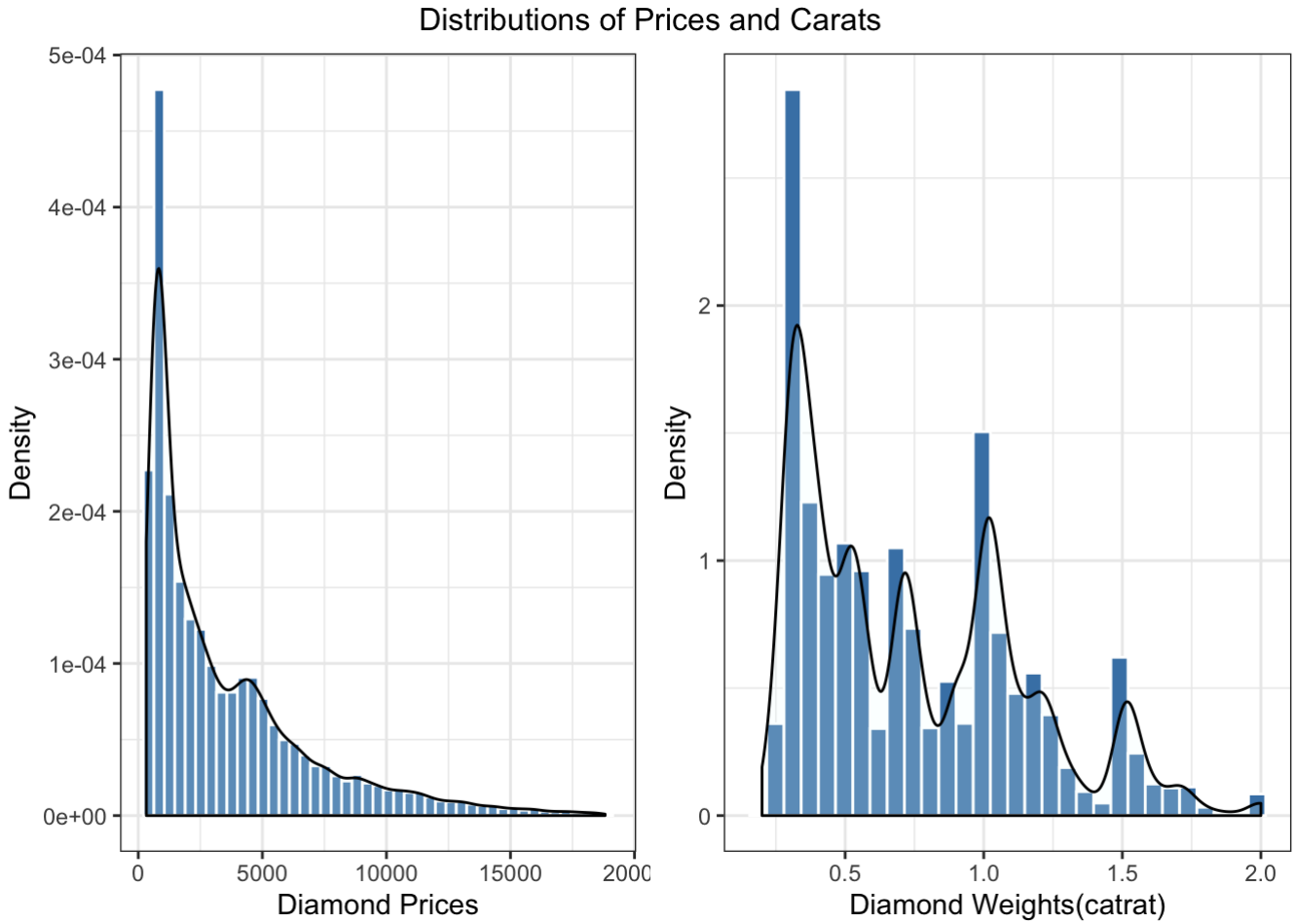
```
print(DiamondsGAMOSR2)
```

```
## [1] 0.916354
```

Logarithmic Model

We now want to fit a logarithmic model. We have seen that the residuals in linear regression follow a pattern of polynomial shape, which may indicate that a logarithmic model would be suitable. A log transformation will keep the variance in residuals closer to the line.

Moreover, by inspecting the distribution of prices we notice a long tail to the right, indicating that we have much more data for lower-priced diamonds and less data for expensive diamonds. This is usually a sign that we need to take the log of the price. A similar observation can be made about carats, although not as prominent. Predicting the log of the price using the log of carat might be a better fit.

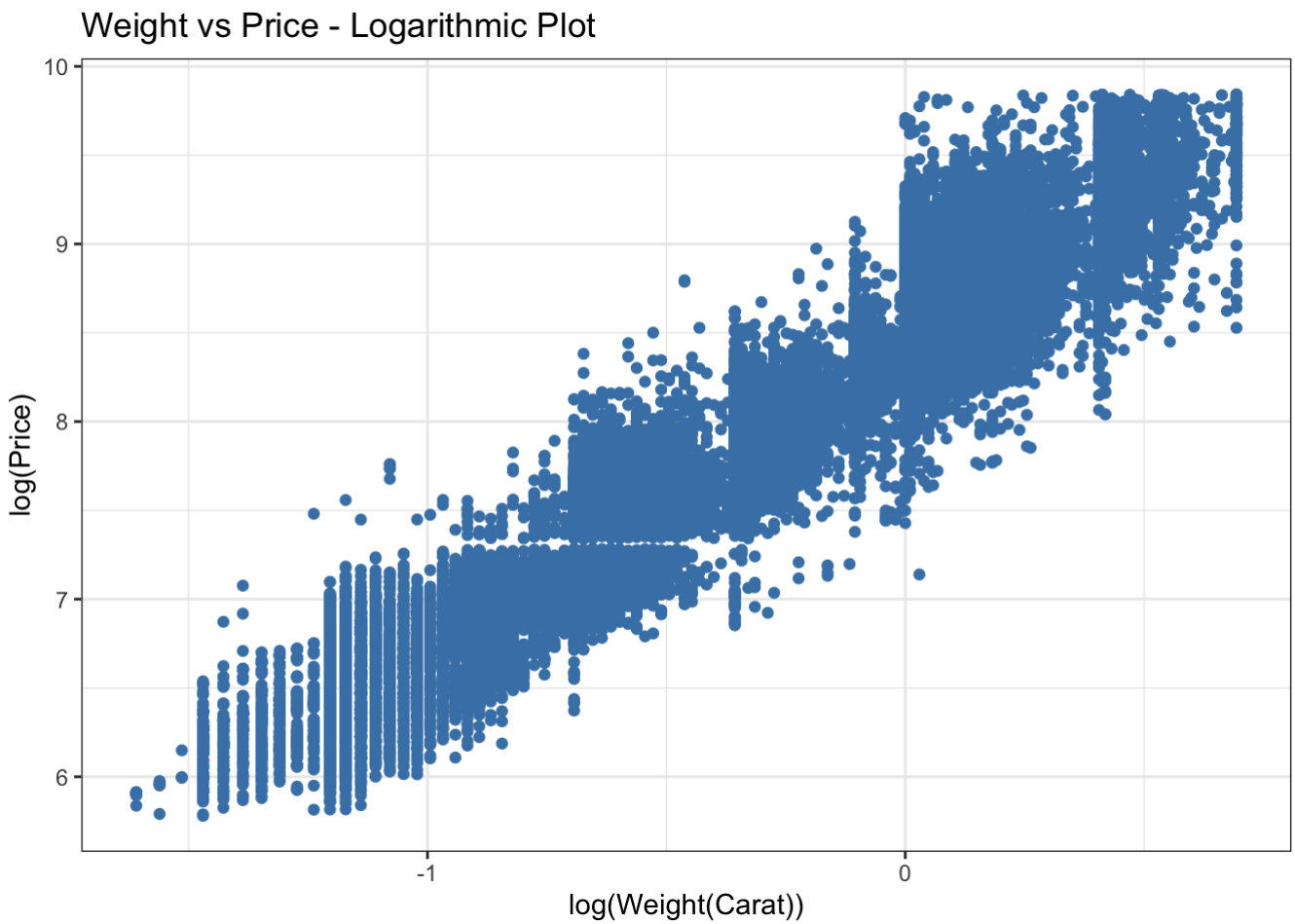


Lastly, when we attempt to plot the log of price and carat, the relationship appears much more linear.

```
DiamondsTrain$logPrice = log(DiamondsTrain$price)
DiamondsTrain$logCarat = log(DiamondsTrain$carat)

DiamondsTest$logPrice = log(DiamondsTest$price)
DiamondsTest$logCarat = log(DiamondsTest$carat)

ggplot(data=DiamondsTrain, aes(x= logCarat)) +
  geom_point(aes(y=logPrice), color="steelblue") +
  theme_bw() +
  xlab("log(Weight(Carat))") +
  ylab("log(Price)") +
  ggtitle("Weight vs Price - Logarithmic Plot") + theme_bw()
```




```
DiamondsLog = lm(logPrice ~ logCarat + cut + color + clarity, data=DiamondsTrain)
DiamondsLogR2 = summary(DiamondsLog)$r.squared
print(summary(DiamondsLog)$r.squared)
```

```
## [1] 0.981189
```

```
DiamondsLogPredictTest = predict(DiamondsLog, newdata=DiamondsTest)
DiamondsLogSSE = sum((DiamondsLogPredictTest - DiamondsTest$logPrice)^2)
DiamondsLogSST = sum((mean(DiamondsTrain$logPrice) - DiamondsTest$logPrice)^2)
DiamondsLogOSR2 = 1 - DiamondsLogSSE/DiamondsLogSST
DiamondsLogOSR2
```

```
## [1] 0.9822383
```

This is our final model. Let us summarize the perfomance of the models we have seen thus far.

```
##                ModelName In_Sample_R2 Out_Of_Sample_R2
## 1           Linear Model    0.9032806        0.9036701
## 2 Tavernier's Regression    0.7798389        0.7804181
## 3        Natural Splines    0.8396580        0.8394498
## 4                    GAM    0.9155495        0.9163540
## 5              Log Model    0.9811890        0.9822383
```

The logarithmic model appears by far the best choice, having superior performance both in-sample and out of sample. Interestingly, the logarithmic R-squared outperforms the in-sample R-squared, which is uncommon but possible if the data in the test set is more centered around the mean. It is possible that for example, the test set has fewer low-priced diamonds.

Lastly, the residual analysis demonstrates that the residuals are centered around the 0-line and do not display any patters. The original residual plot showed a systematic error, in terms of over/underfitting as well as heteroskedasticity. The logarithmic residual plot has a relatively equal number of points above and below the line and the variance is constant. This strengthens our belief in the model's suitability.

```
DiamondsLogResiduals <- data.frame(predicted = predict(DiamondsLog),
                           residuals = residuals(DiamondsLog))


ggplot(DiamondsLogResiduals, aes(x = predicted, y = residuals)) +
  geom_point(color = "steelblue") +
  geom_smooth(color = "black") +
  geom_hline(yintercept=0, linetype="dashed", color="black") +
  xlab("Fitted values")+ylab("Residuals") +
  ggtitle("Residual vs Fitted Plot - Logarithmic Model") + theme_bw()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```