Predict Price of Diamonds

Ajay Shankar A

2023-12-10

Table of contents

## 1 Predicting Diamonds Price

### 1.1 Introduction

Building a model to **predict the price of the diamonds** using tidymodels.

Diamonds data set is readily available to use through the ggplot2 library in the tidyverse and we will be using this data set predict the prices of the other diamonds.

In the data set various parameters of diamonds are given and each of these parameters may or may not effect the **price** of the diamonds.

library(tidyverse)

data("diamonds")
diamonds

# A tibble: 53,940 × 10
 carat cut color clarity depth table price x y z
 <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43
 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31
 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31
 4 0.29 Premium I VS2 62.4 58 334 4.2 4.23 2.63
 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75
 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48
 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
# ℹ 53,930 more rows

Data has over 50,000 observations which is good for modeling.

### 1.2 Exploring the data

The diamonds data set is available to explore in ggplot2 library as mentioned above.

Let’s check for NA’s before exploring the data

diamonds %>% map( ~sum(is.na(.))) %>% unlist()

 carat cut color clarity depth table price x y z
 0 0 0 0 0 0 0 0 0 0

It’s really good that there are no NA’s but we have to be careful of the 0 in the numeric columns.

diamonds %>% select(carat, x, y, z) %>% arrange(x, y, z)

# A tibble: 53,940 × 4
 carat x y z
 <dbl> <dbl> <dbl> <dbl>
 1 1 0 0 0
 2 1.14 0 0 0
 3 1.56 0 0 0
 4 1.2 0 0 0
 5 2.25 0 0 0
 6 0.71 0 0 0
 7 0.71 0 0 0
 8 1.07 0 6.62 0
 9 0.2 3.73 3.68 2.31
10 0.2 3.73 3.71 2.33
# ℹ 53,930 more rows

Diamonds cannot have a x(length), y(width), z(depth) of 0 and have weight. So let’s replace these values with NA or we can remove them out completely too.

diamonds %>% mutate(x = if\_else(x == "0", NA, x),
 y = if\_else(y == "0", NA, y),
 z = if\_else(y == "0", NA, z))

# A tibble: 53,940 × 10
 carat cut color clarity depth table price x y z
 <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43
 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31
 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31
 4 0.29 Premium I VS2 62.4 58 334 4.2 4.23 2.63
 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75
 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48
 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
# ℹ 53,930 more rows

Now lets visualize the distribution of the diamonds.

diamonds %>% ggplot(aes(carat)) + geom\_freqpoly(binwidth = 0.05)

|  |
| --- |
| Figure 1: Frequency polygon plot |

From the [Figure 1](#fig-price_hist) we can observe that - Most of the diamonds are between 0.2 to 1.5 carats. - There are peaks which means higher number of diamonds at whole and common fractions.

My general knowledge is that the weight i.e, carat of the diamond influences the price most. Let’s visualize that.

diamonds %>% ggplot(aes(carat, price)) + geom\_hex(bins = 50)



The price tends to follow exponential curve the log2() curve, we can confirm this by another graph.

diamonds %>% filter(carat < 2.5) %>%
 mutate(log\_price = log10(price),
 log\_carat = log10(carat)) %>%
 ggplot(aes(log\_carat, log\_price)) + geom\_hex(bins = 50)

|  |
| --- |
| Figure 2: Log of carat vs Log of Price at base 2 |

The above [Figure 2](#fig-log_prc_vs_carat) shows that once we apply log2() to both price and carat the relationship mostly looks to be linear.

diamonds %>% filter(carat <= 2.5) %>% ggplot(aes(carat, price)) +
 geom\_point(alpha = 0.1, color = "purple")



We can see that price jumps when the weight is exactly or greater than to the whole and common fractions such as 0.5, 1.0, 1.5 and 2.

library(patchwork)

plot\_parameter <- function(param){
 ggplot(diamonds, aes(fct\_reorder({{param}}, price), price)) +
 geom\_boxplot() + stat\_summary(fun.y = mean, geom = "point") +
 labs(x = as\_label(substitute(param)))
}

(plot\_parameter(cut) + plot\_parameter(color)) /
 (plot\_parameter(clarity))



Low quality diamonds with Fair cut and low quality color seems to have very high price. So now lets use tidymodels to model the data using rand\_forest

### 1.3 Building a model

As every parameter in the data is important for the price prediction we are going to keep all the columns intact.

library(tidymodels)
set.seed(2023)

diamonds\_2 <- diamonds %>%
 mutate(price = log2(price), carat = log2(carat))

diamonds\_split <- initial\_split(diamonds\_2, strata = carat, prop = 0.8)
diamonds\_split

<Training/Testing/Total>
<43150/10790/53940>

diamonds\_train <- training(diamonds\_split)
diamonds\_test <- testing(diamonds\_split)

I am using strata with carat as most of the diamonds are not properly distributed yet all diamonds of different weight should be well represented.

diamonds\_folds <- vfold\_cv(diamonds\_train, v = 10)
diamonds\_folds

# 10-fold cross-validation
# A tibble: 10 × 2
 splits id
 <list> <chr>
 1 <split [38835/4315]> Fold01
 2 <split [38835/4315]> Fold02
 3 <split [38835/4315]> Fold03
 4 <split [38835/4315]> Fold04
 5 <split [38835/4315]> Fold05
 6 <split [38835/4315]> Fold06
 7 <split [38835/4315]> Fold07
 8 <split [38835/4315]> Fold08
 9 <split [38835/4315]> Fold09
10 <split [38835/4315]> Fold10

I think rand\_forest will work better on this data set but lets compare both **Linear Regression** models and **Random Forest** models.

lm\_spec <- linear\_reg() %>% set\_engine("glm")
lm\_spec

Linear Regression Model Specification (regression)

Computational engine: glm

rf\_spec <- rand\_forest(mtry = 3, trees = 500, min\_n = 5 ) %>%
 set\_mode("regression") %>%
 set\_engine("ranger")
rf\_spec

Random Forest Model Specification (regression)

Main Arguments:
 mtry = 3
 trees = 500
 min\_n = 5

Computational engine: ranger

We still need to maniplulate some parts of the data like price and carat so that they are optimised which can be done using recipe library.

diamonds\_recp <-
 recipe(price ~ ., data = diamonds\_train)

diamonds\_recp

── Recipe ──────────────────────────────────────────────────────────────────────

── Inputs

Number of variables by role

outcome: 1
predictor: 9

Next let’s start putting together a tidymodels workflow(), a helper object to help manage modeling pipelines with pieces that fit together like Lego blocks.

diamonds\_wf <- workflow() %>%
 add\_recipe(diamonds\_recp)

diamonds\_wf

══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: None

── Preprocessor ────────────────────────────────────────────────────────────────
0 Recipe Steps

Let’s fit the two models we prepared for the data. First code block contains linear regression model and the second contains the random\_forest model.

1. Linear Regression model

# Linear regression
glm\_rs <- diamonds\_wf %>%
 add\_model(lm\_spec) %>%
 fit\_resamples(
 resamples = diamonds\_folds,
 control = control\_resamples(save\_pred = TRUE)
 )

glm\_rs

# Resampling results
# 10-fold cross-validation
# A tibble: 10 × 5
 splits id .metrics .notes .predictions
 <list> <chr> <list> <list> <list>
 1 <split [38835/4315]> Fold01 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 2 <split [38835/4315]> Fold02 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 3 <split [38835/4315]> Fold03 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 4 <split [38835/4315]> Fold04 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 5 <split [38835/4315]> Fold05 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 6 <split [38835/4315]> Fold06 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 7 <split [38835/4315]> Fold07 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 8 <split [38835/4315]> Fold08 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 9 <split [38835/4315]> Fold09 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
10 <split [38835/4315]> Fold10 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>

1. Random Forest model

rf\_rs <- diamonds\_wf %>%
 add\_model(rf\_spec) %>%
 fit\_resamples(
 resamples = diamonds\_folds,
 control = control\_resamples(save\_pred = TRUE)
 )

rf\_rs

# Resampling results
# 10-fold cross-validation
# A tibble: 10 × 5
 splits id .metrics .notes .predictions
 <list> <chr> <list> <list> <list>
 1 <split [38835/4315]> Fold01 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 2 <split [38835/4315]> Fold02 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 3 <split [38835/4315]> Fold03 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 4 <split [38835/4315]> Fold04 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 5 <split [38835/4315]> Fold05 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 6 <split [38835/4315]> Fold06 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 7 <split [38835/4315]> Fold07 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 8 <split [38835/4315]> Fold08 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 9 <split [38835/4315]> Fold09 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
10 <split [38835/4315]> Fold10 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>

We have fit out training data to the models.

## 2 Evaluating models

Now let’s check the results and how well they performed.

# Linear regression model
collect\_metrics(glm\_rs)

# A tibble: 2 × 6
 .metric .estimator mean n std\_err .config
 <chr> <chr> <dbl> <int> <dbl> <chr>
1 rmse standard 0.194 10 0.00102 Preprocessor1\_Model1
2 rsq standard 0.983 10 0.000198 Preprocessor1\_Model1

# Random Forest model
collect\_metrics(rf\_rs)

# A tibble: 2 × 6
 .metric .estimator mean n std\_err .config
 <chr> <chr> <dbl> <int> <dbl> <chr>
1 rmse standard 0.133 10 0.000906 Preprocessor1\_Model1
2 rsq standard 0.992 10 0.000114 Preprocessor1\_Model1

Both of the models have high accuracy but I prefer **Random Forest Model** as it is preferred when there are more nominal and ordinal data points.