STATS 7022 - Data Science Practical Exam

Prithviraj Banerjee

Trimester 1, 2023

Overview

This practical exam is an opportunity for you to demonstrate your ability to analyse data in `R` using the `tidymodels` package.

You will be analysing the `diamonds` dataset from the `ggplot2` package to predict the price of a diamond.

Instructions

1. Download the file DSPG_Practical_Exam.Rmd from MyUni.
2. Within this .Rmd file, use the function `start_exam()` to obtain
   - the data,
   - your specified method, and
   - your specified response variable.
3. Perform a complete analysis of the data. Your analysis should include at least the following:
   a. exploratory data analysis
   b. data cleaning and pre-processing
   c. fitting a model using your specified method
   d. hyperparameter tuning
   e. assessing the fit of your model
   f. choosing a single model for prediction
   g. using your model to make predictions, using an example of your choice
4. You may use this file as a template for your analysis.
5. From the time the exam opens on MyUni, you have 24 hours to complete and submit the analysis.

Submission

You **MUST** submit your analysis at the Practical Exam submission portal on MyUni.

You **MUST** submit your analysis as a .pdf file. You may export your analysis directly to .pdf, or export as an .html file and convert to .pdf.

You **MUST** submit your code as a .R or .Rmd file. Your marker may run your code. You may lose marks if your code does not produce the output contained in your analysis.

No other file submissions will be accepted.

Start the Exam

Enter your student ID in the code block below, then run the code block to start the exam.

You **MUST** enter your ID as an integer. For example, if your student number is a1234567, then please enter your ID as `ID <- 1234567`.

You **MUST** enter your own student ID. Using a different ID may result in a loss of marks.

Importing the necessary libraries

```
library(rsample)
#install.packages("parsnip")
library(parsnip)
#install.packages("tune")
library(ranger)
library(tune)
library(dplyr)
#install.packages("workflows")
library(workflows)
library(doParallel)
```

Loading the data and the method

```
ID <- 1753181
data <- start_exam(ID)
```

```
## Please fit a random forest model to the data returned by this function, specifying price as the response varia
ble.
```

Analysis

Inspect the data

```
head(data)
```

price	carat	cut	table	depth	color	z	x	clarity
<int>	<dbl>	<ord>	<dbl>	<dbl>	<ord>	<dbl>	<dbl>	<ord>
939	0.39	Ideal	56.1	61.7	F	2.89	4.65	VS1
1908	0.53	Ideal	55.0	62.2	F	3.23	5.22	VS1
1340	0.56	Ideal	57.0	61.4	F	3.25	5.28	SI1
4824	1.21	Good	60.8	61.1	J	4.18	6.83	SI2
5138	1.08	Premium	58.0	62.1	D	4.10	6.62	SI2
5408	1.13	Very Good	60.0	62.3	D	4.14	6.59	SI2

6 rows

We get an idea about the data.The data contains 43152 rows and 9 predictors with price as the response variable.

EDA

Display tables

```
skimr::skim_without_charts(data)
```

Data summary

Name	data
Number of rows	43152
Number of columns	9
_______________________	
Column type frequency:	
factor	3
numeric	6
________________________	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
cut	0	1	TRUE	5	Ide: 17252, Pre: 11031, Ver: 9703, Goo: 3859
color	0	1	TRUE	7	G: 9104, E: 7844, F: 7589, H: 6617
clarity	0	1	TRUE	8	SI1: 10461, VS2: 9753, SI2: 7351, VS1: 6574

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
price	0	1	3933.10	3979.33	326.0	953.00	2418.50	5334.25	18823.00
carat	0	1	0.80	0.47	0.2	0.40	0.70	1.04	5.01
table	0	1	57.45	2.24	44.0	56.00	57.00	59.00	95.00
depth	0	1	61.75	1.43	43.0	61.00	61.80	62.50	79.00
z	0	1	3.54	0.71	0.0	2.91	3.53	4.03	31.80
x	0	1	5.73	1.12	0.0	4.72	5.70	6.54	10.74

```
mean(data$price)
```

```
## [1] 3933.105
```

```
min(data$price)
```

```
## [1] 326
```

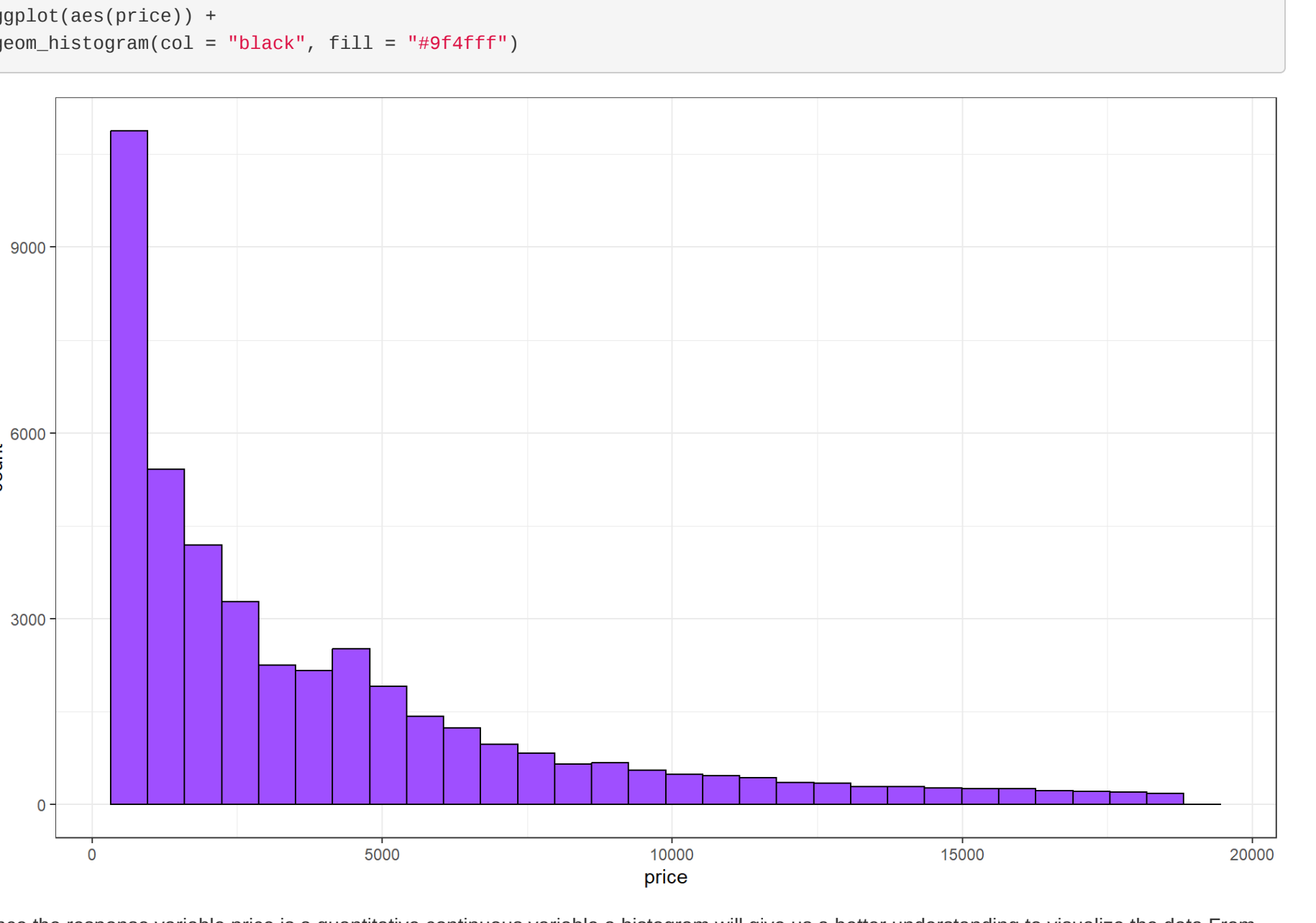
```
max(data$price)
```

```
## [1] 18823
```

The sample mean of the response variable (price) is 3933.105 and the minimum value of price is 326 and the max value is 18823 from the table above.

Visualising the response

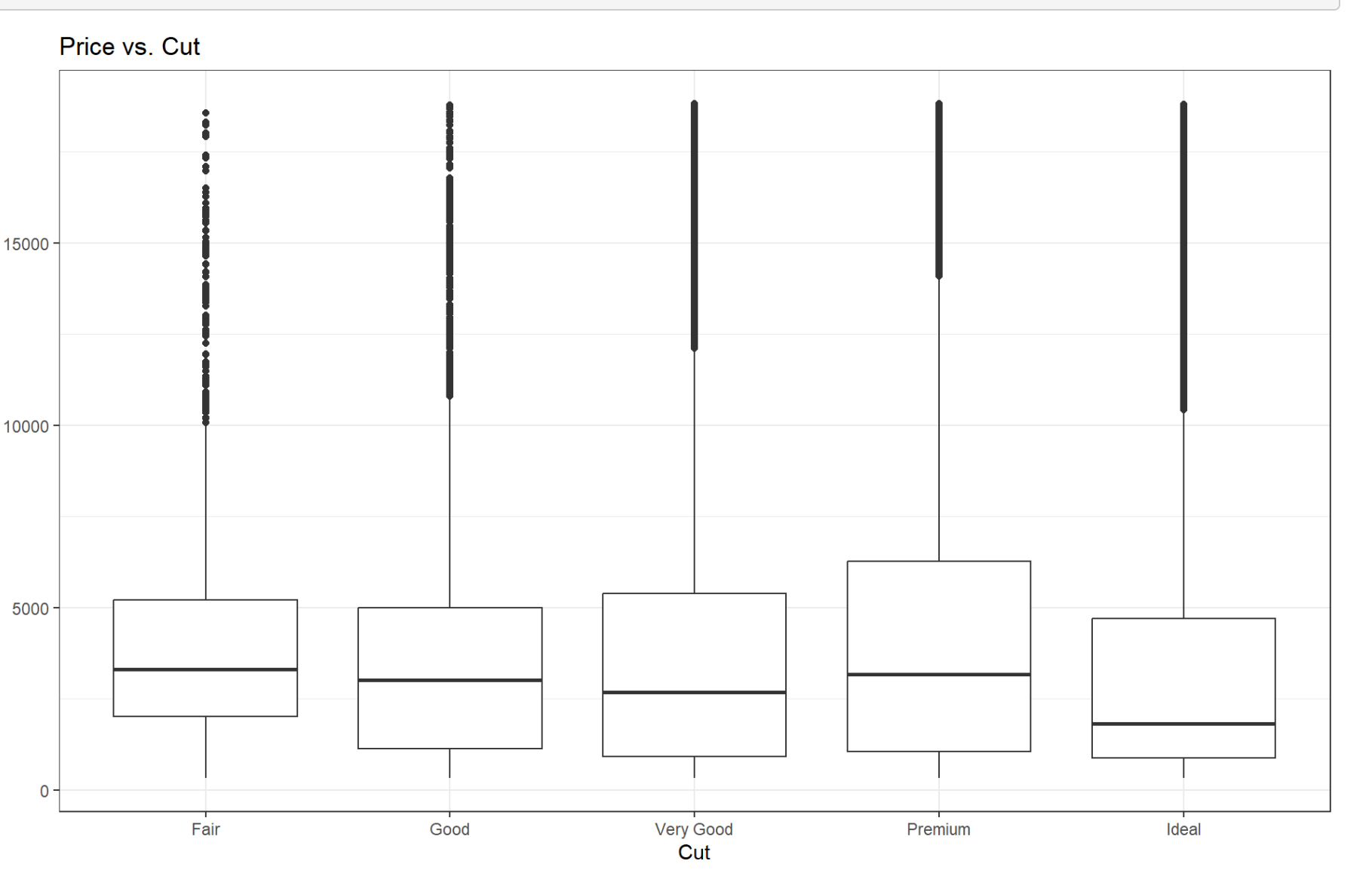
```
data %>%
ggplot(aes(price)) +
geom_histogram(col = "black", fill = "#9f4fff")
```



Since the response variable price is a quantitative continuous variable a histogram will give us a better understanding to visualize the data.From the histogram it is clearly visible that it is a right-skewed uni modal distribution.

A box plot can be obtained between the response variable (price) and the predictor variable (cut)

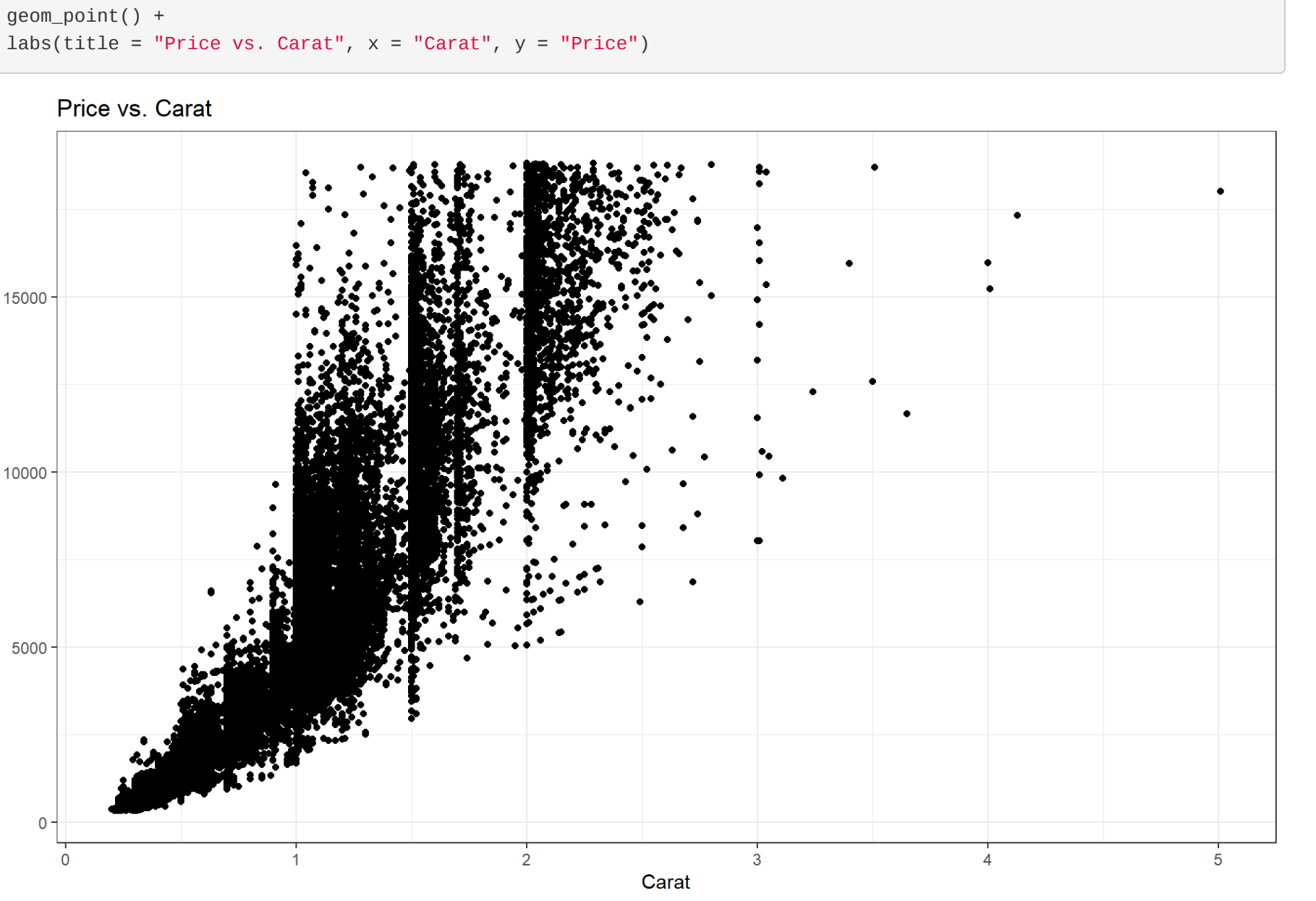
```
ggplot(data, aes(x = cut, y = price)) +
  geom_boxplot() +
  labs(title = "Price vs. Cut", x = "Cut", y = "Price")
```



Each box represents the interquartile range of prices for the level of cut, with the median represented by the horizontal line. The whiskers indicate the range of prices, while the points outside the whiskers represent the potential outliers. From this plot, we can see that diamonds with higher cut quality tend to have higher prices, with Premium cuts having the highest median prices and Ideal cuts having the lowest. This suggests that cut quality is an important factor in determining the price of a diamond.

The second plot shows the relationship between the price of diamonds and their carat weight

```
ggplot(data, aes(x = carat, y = price)) +
  geom_point() +
  labs(title = "Price vs. Carat", x = "Carat", y = "Price")
```



The x-axis represents the carat weight, while the y-axis represents the price. Each point on the plot represents a diamond, and the position of the point indicates its carat weight and price.From this plot, we can see that there is a strong positive co-relationship between the carat weight and price of diamonds, as expected. This means that, in general, larger diamonds tend to be more expensive than smaller diamonds.

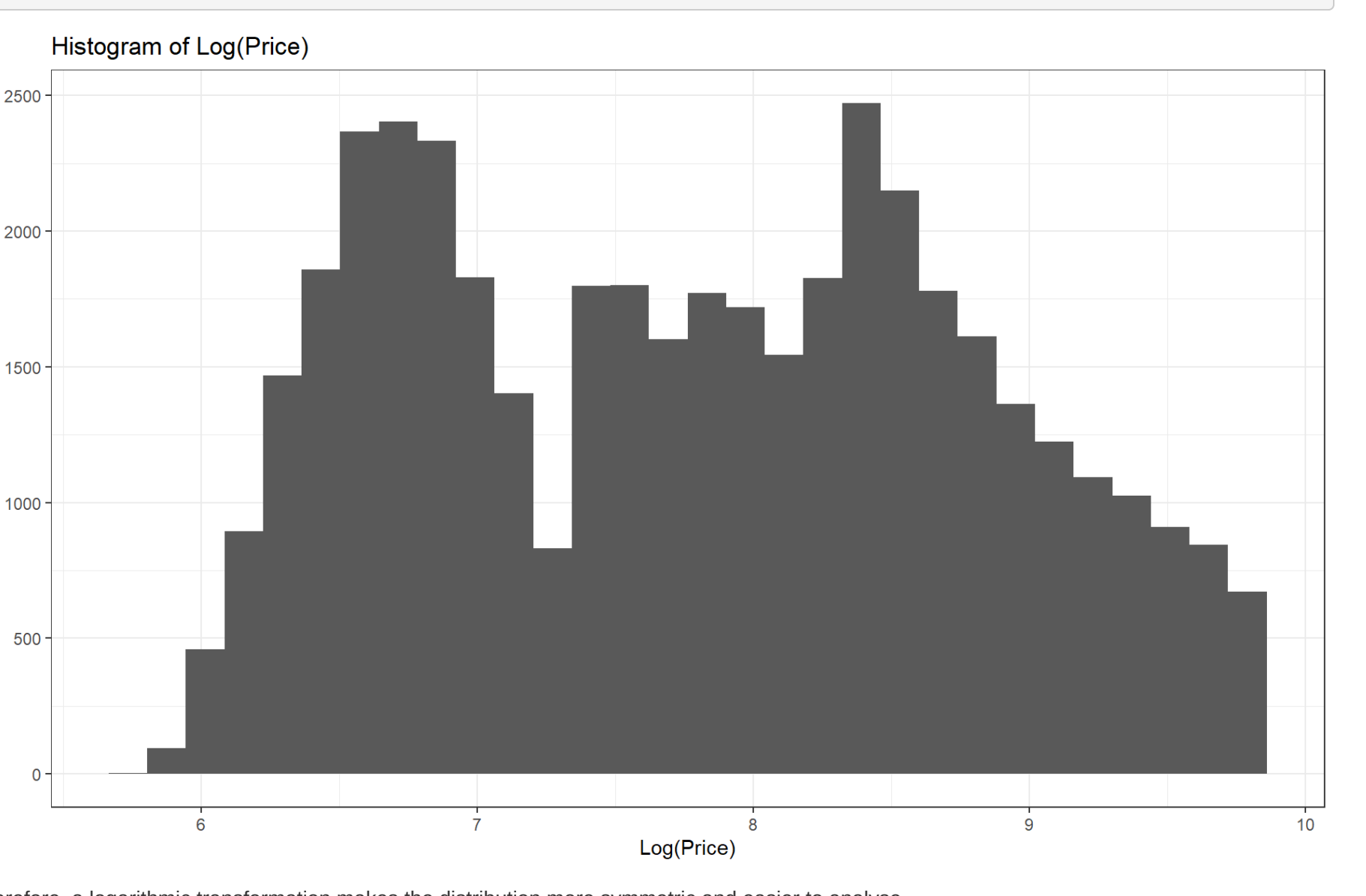
Cleaning

After looking at the distribution of price, it is clearly visible that the response variable is rightly skewed. This suggests that there may be extreme outliers or values that may drive up the price.

```
# Apply a logarithmic transformation to price
data$log_price <- log(data$price)

# Removing the actual price column
data <- data[,-c(1)]

# Create a histogram of log(price)
ggplot(data, aes(x = log_price)) +
  geom_histogram(bins = 30) +
  labs(title = 'Histogram of Log(Price)', x = 'Log(Price)', y = 'Count')
```



Therefore, a logarithmic transformation makes the distribution more symmetric and easier to analyse.

Convert ordinal to nominal

```
# Convert the cut, color and clarity variable to a factor
data$cut <- factor(data$cut, levels = c('Fair', 'Good', 'Very Good', 'Premium', 'Ideal'))
data$color <- factor(data$color)
data$clarity <- factor(data$clarity, levels = c("I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1", "IF"))
```

Checking for null values

```
sum(is.na(data))
```

```
## [1] 0
```

There are no missing values in our data. So no imputation is required.

Pre-processing

Splitting the data into test and training sets

```
set.seed(2022)
data_split <- initial_split(data, strata = log_price)
data_train <- training(data_split)
data_test <- testing(data_split)
# we will partition our training set into 10 folds.
data_folds <- vfold_cv(data_train, v = 10, strata = log_price)
data_folds
```

splits	id
<list>	<chr>
<S3: vfold_split>	Fold01
<S3: vfold_split>	Fold02
<S3: vfold_split>	Fold03
<S3: vfold_split>	Fold04
<S3: vfold_split>	Fold05
<S3: vfold_split>	Fold06
<S3: vfold_split>	Fold07
<S3: vfold_split>	Fold08
<S3: vfold_split>	Fold09
<S3: vfold_split>	Fold10

1-10 of 10 rows

Creating the recipe and performing centering and scaling

```
pacman::p_load(textrecipes)
data_recipe <- recipe(log_price ~ ., data = data_train) %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())
data_recipe %>% prep() %>% bake(new_data=NULL)
```

carat	cut	table	depth	color	z	x	clarity	log_price
<dbl>	<ord>	<dbl>	<dbl>	<ord>	<dbl>	<dbl>	<ord>	<dbl>
-0.863691556	Ideal	-0.60427336	-0.03514611	F	-0.9118663755	-0.966169687	VS1	6.844815
-1.053855059	Very Good	-0.64899862	-0.66528384	G	-1.2483625900	-1.233724217	VS1	6.371612
-0.948208669	Ideal	-0.64899862	0.24491511	E	-1.0240317803	-1.082109983	VS2	6.642487
-1.201760005	Very Good	0.24550665	-0.66528384	E	-1.5427967777	-1.572626622	VS2	5.996452
-0.990467225	Ideal	-1.99075652	0.59499163	G	-1.0660938071	-1.144539374	VVS2	6.803505
-1.011596503	Ideal	-0.20174598	-0.24519202	D	-1.1642385364	-1.180213311	SI1	6.378426
-0.969337947	Very Good	1.14001192	0.10488450	H	-1.1081558339	-1.171294827	VS2	6.285998
-1.032725781	Premium	0.69275929	0.31493041	G	-1.1782592120	-1.224805733	VVS2	6.810142
-0.990467225	Very Good	0.24550665	-0.24519202	I	-1.1081558339	-1.135620889	SI1	6.098074
-0.884820834	Very Good	-0.20174598	1.15511406	H	-0.8698043487	-0.984006656	SI2	6.553933

1-10 of 10,000 rows Previous 1 2 3 4 5 6 ... 1000 Next

We create a recipe object and perform pre-processing steps like centering and scaling to all the numeric predictors. We do this on the training data.

Modelling and Tuning

we specify a random forest model. Note that we set_mode() to regression because our response variable is a quantitative variable.We specify tune() for the hyperparameters we want to tune in our model:

```
data_model <- rand_forest(mtry = tune(),
 min_n = tune(),
 trees = 10) %>%
 set_mode("regression") %>%
 set_engine("ranger")
```

Create the workflow

```
data_wf <- workflow() %>%
 add_recipe(data_recipe) %>%
 add_model(data_model)
data_wf
```

```
## ══ Workflow ════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 2 Recipe Steps
##
## • step_center()
## • step_scale()
##
## ── Model ───────────────────────────────────────────────────────────────────
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 10
##   min_n = tune()
##
## Computational engine: ranger
```

Creating the tuning grid

```
data_grid <- grid_regular(mtry(c(1,5)),
 min_n(),
levels = 5)
data_grid
```

mtry	min_n
<int>	<int>
1	2
2	2
3	2
4	2
5	2
1	11
2	11
3	11
4	11
5	11

1-10 of 25 rows Previous 1 2 3 Next

Tuning the model for optimal hyperparameters

```
doParallel::registerDoParallel()
data_tune <- tune_grid(data_wf,
 resamples = data_folds,
grid = data_grid)
```

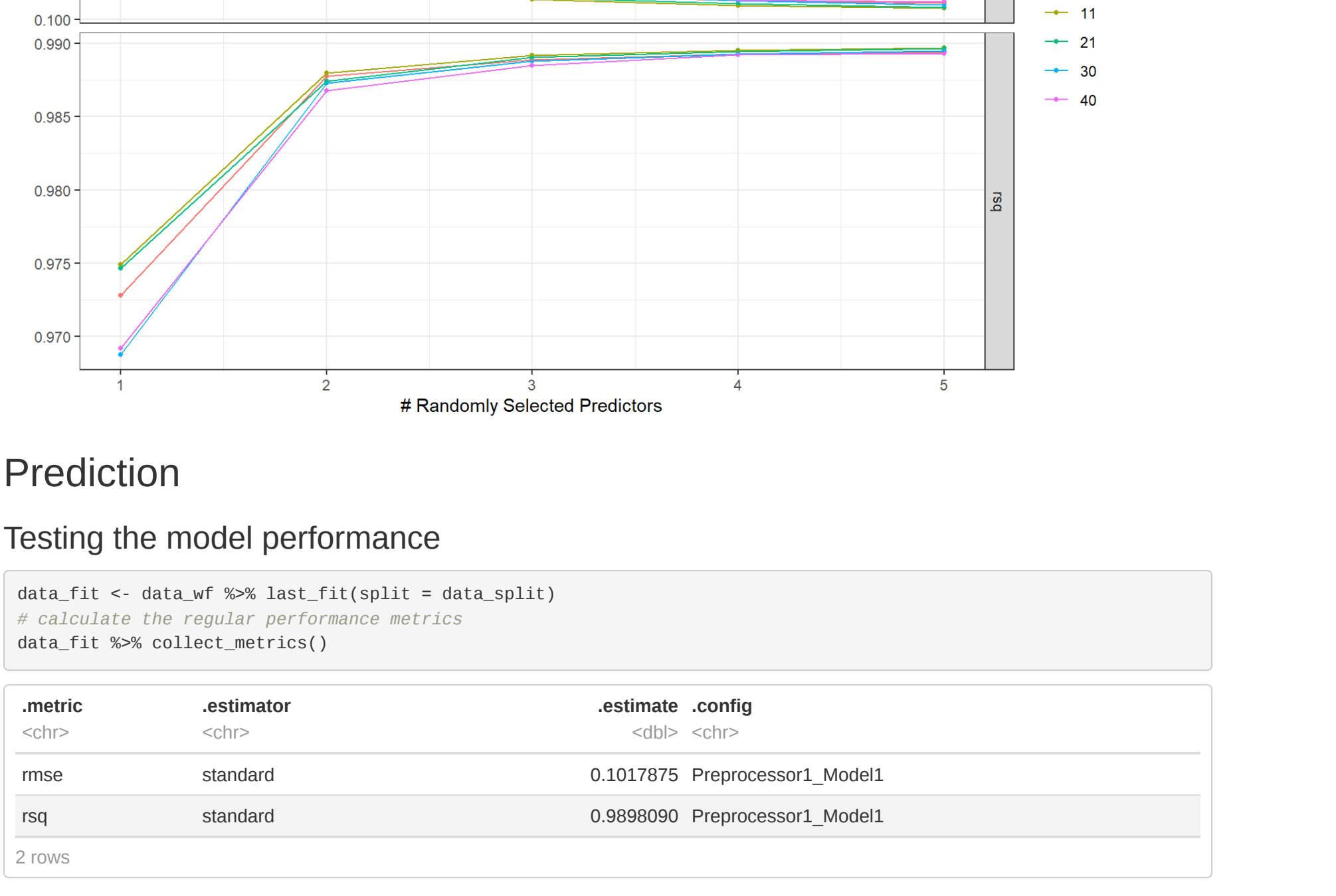
Choose the optimal hyperparameters based on tuning results

```
data_wf <- data_wf %>%
 finalize_workflow(select_best(data_tune, metric = "rmse"))
data_wf
```

```
## ══ Workflow ════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 2 Recipe Steps
##
## • step_center()
## • step_scale()
##
## ── Model ───────────────────────────────────────────────────────────────────
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = 5
##   trees = 10
##   min_n = 11
##
## Computational engine: ranger
```

Plot the tuning model

```
data_tune %>% autoplot()
```



Prediction

Testing the model performance

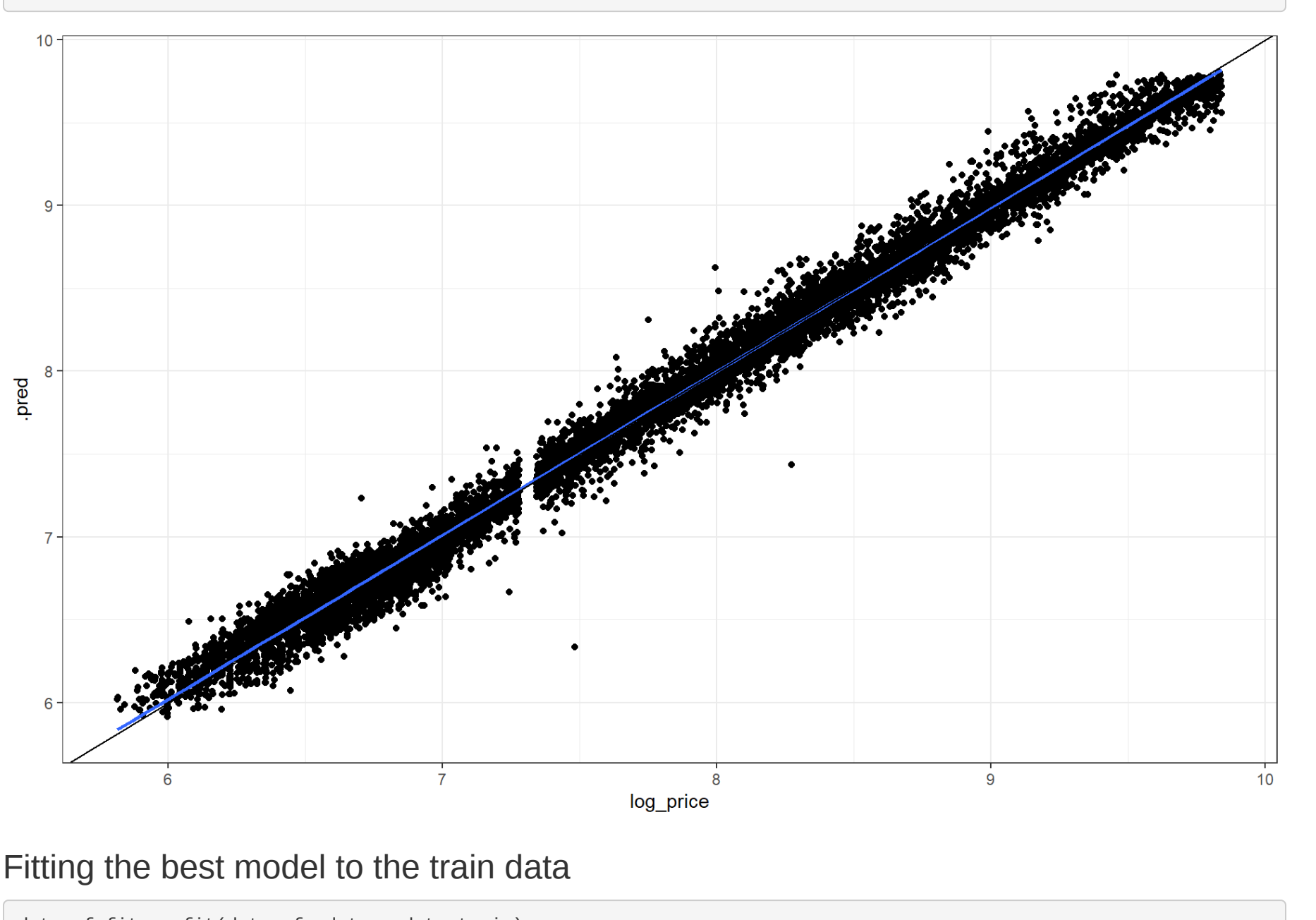
```
data_fit <- data_wf %>% last_fit(split = data_split)
# calculate the regular performance metrics
data_fit %>% collect_metrics()
```

.metric	.estimator	.estimate	.config
<chr>	<chr>	<dbl>	<chr>
rmse	standard	0.1017875	Preprocessor1_Model1
rsq	standard	0.9898090	Preprocessor1_Model1

2 rows

Visualising the model performance

```
data_fit %>% collect_predictions() %>%
 ggplot(aes(log_price, .pred)) +
 geom_point() +
 geom_smooth(method = "lm") +
 geom_abline(intercept = 0, slope = 1)
```



Fitting the best model to the train data

```
data_wf_fit <- fit(data_wf, data = data_train)
```

##Prediction

Prediction for new observation

```
# Prediction for the first observation in test data
prediction <- predict(data_wf_fit, new_data = data_test[1,])

# View the predicted log_price
prediction$.pred
```

```
## [1] 8.481783
```