STATS 7022 - Data Science
Practical Exam

Nguyen Phuc Thai

Trimester 1, 2023

```
ID <- 1846505
my_data <- start_exam(ID)
```

```
## Please fit a random forest model to the data returned by this function, specifying pri
```

```
##### when generating this, I already saved my tuning result, and set the corresponding
#code chunk to eval=F, and just add a code chunk to load the tuning result back in.
#So if you want to run the code you have to undo that
```

1 Analysis

The first five row of the dataset is as follow

```
knitr::kable(head(my_data), caption = "First five observations")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 1: First five observations

price	cut	table	x	carat	depth	z	color	y
5599	Good	54	6.31	1.01	63.9	4.05	G	6.37
802	Premium	60	4.39	0.31	61.2	2.68	H	4.37
1192	Premium	58	4.84	0.41	60.6	2.92	F	4.79
774	Very Good	58	4.27	0.30	61.4	2.64	G	4.33
3512	Good	63	6.36	1.01	60.8	3.88	G	6.40
1154	Ideal	53	5.06	0.50	62.2	3.17	H	5.14

There are some categorical variables, namely the color of diamonds and the cut of diamonds. They will be coded as factor variable. Three columns x, y and z are ambiguous. From the help

document on the diamonds dataset, they are the length, width and depth of the diamonds in millimeters, respectively. Hence they will be renamed accordingly. These are done as follow:

```
## factor variable
my_data=my_data%>%
  mutate(color=as.factor(color),
         cut=as.factor(cut))

## renaming ambiguous columns
my_data <- my_data %>% rename( "diamond_length"='x',
                                          "diamond_width" = "y",
                                          "diamond_depth" = "z")
```

A summary of the data is shown as follow:

```
knitr::kable(summary(my_data[,1:4]), caption = "data summary")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 2: data summary

	price	cut	table	diamond_length
	Min. : 326	Fair : 1321	Min. :43.00	Min. : 0.000
	1st Qu.: 956	Good : 3873	1st Qu.:56.00	1st Qu.: 4.720
	Median : 2425	Very Good: 9637	Median :57.00	Median : 5.700
	Mean : 3952	Premium :11100	Mean :57.46	Mean : 5.738
	3rd Qu.: 5341	Ideal :17221	3rd Qu.:59.00	3rd Qu.: 6.540
	Max. :18818	NA	Max. :95.00	Max. :10.740

```
knitr::kable(summary(my_data[,5:9]), caption = "data summary")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 3: data summary

	carat	depth	diamond_depth	color	diamond_width
	Min. :0.2000	Min. :43.00	Min. : 0.000	D:5432	Min. : 0.000
	1st Qu.:0.4000	1st Qu.:61.00	1st Qu.: 2.910	E:7805	1st Qu.: 4.730
	Median :0.7000	Median :61.80	Median : 3.530	F:7643	Median : 5.720
	Mean :0.8007	Mean :61.75	Mean : 3.543	G:8975	Mean : 5.741
	3rd Qu.:1.0400	3rd Qu.:62.50	3rd Qu.: 4.040	H:6671	3rd Qu.: 6.540
	Max. :5.0100	Max. :79.00	Max. :31.800	I:4371	Max. :58.900
	NA	NA	NA	J:2255	NA

Aside from the ambiguous name and data type, the data seems to be cleaned at this point. Although it should be noted that there are some very large values in carat (max is 5.01, while 3rd quartile is 1.04), diamond depth (max 31.8 while 3rd quartile is 4.04), diamond width (max 58.9 while 3rd quartile is 6.54) and some other numerical variables. However, looking at the description of the diamond data, there's no indication that these data points are wrong. Hence they will be kept as is.

2 EDA

2.1 The response variable

```
my_data%>%
  ggplot(aes(x=price))+
  geom_histogram()+
  ggtitle('price distribution')
```

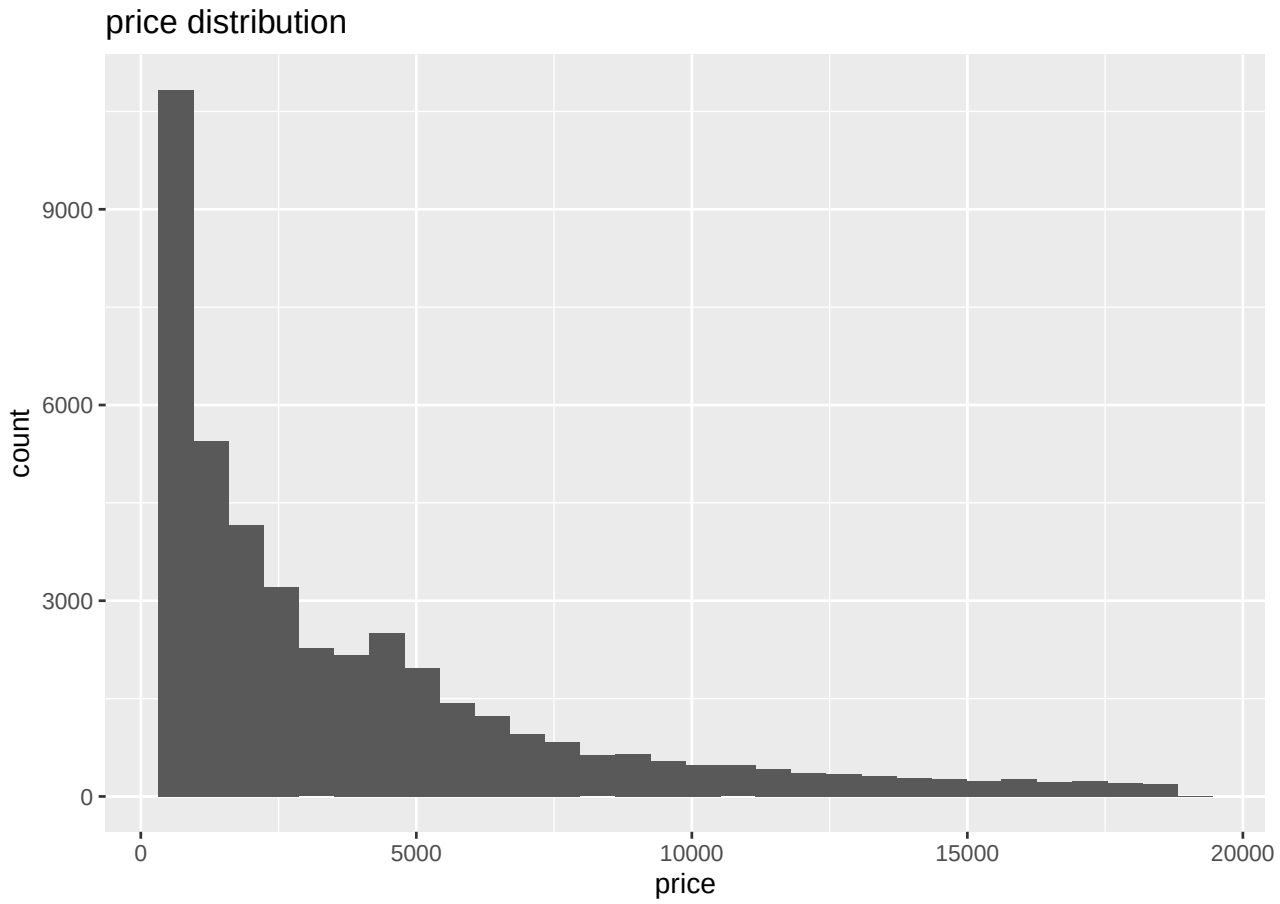

Figure 1: price distribution

Price of diamonds is right-skewed. Consequently, the data should be transformed to create a more balanced data so the model does not tend to predict response at value range that are more frequent. This is done below by taking the log of the price

```
my_data_tranformed<-my_data%>%
  mutate(price=log(price))
```

```
my_data_tranformed%>%
  ggplot(aes(x=price))+
  geom_histogram()+
  ggtitle('logged price distribution')+
  xlab('logged price')
```

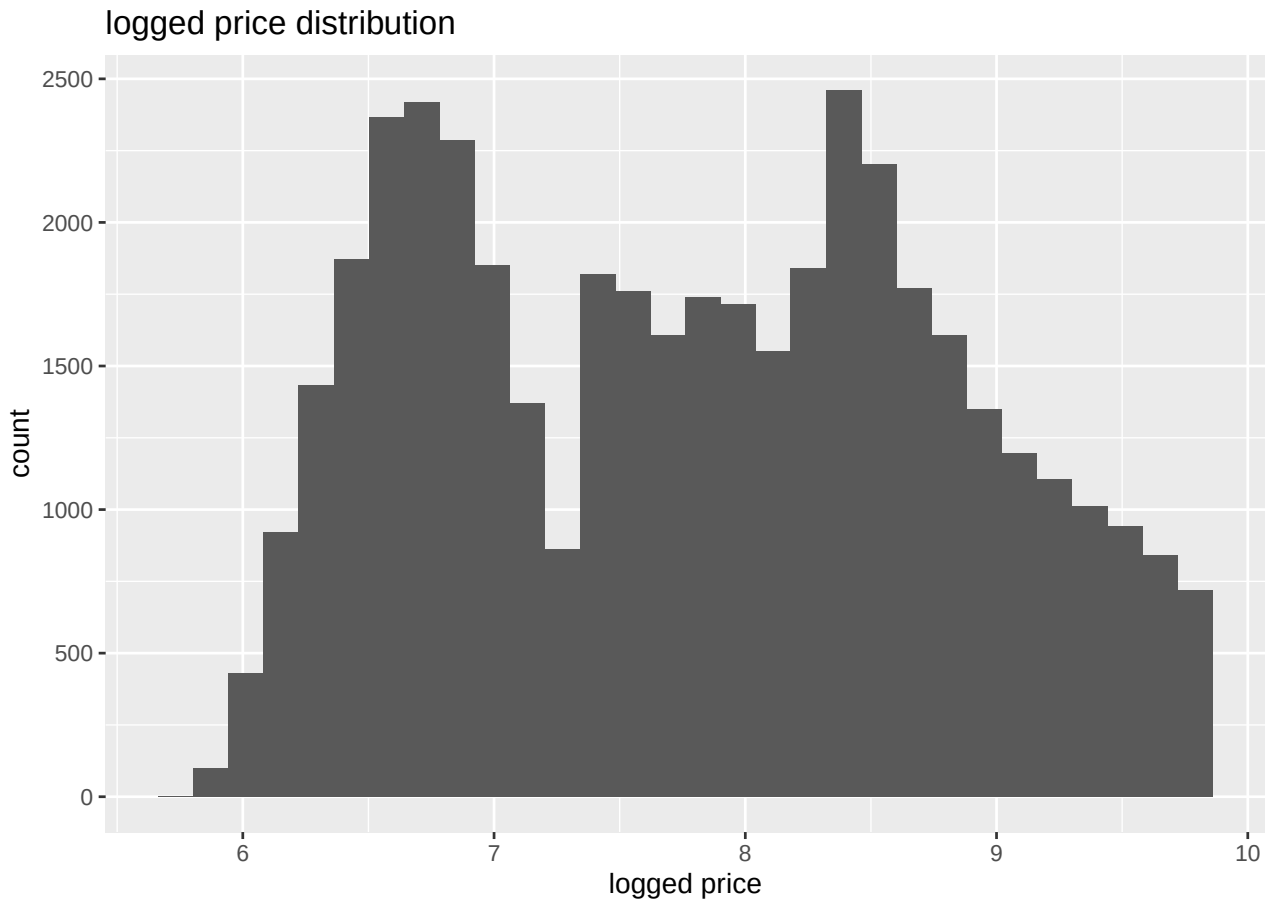

Figure 2: log of price distribution

After taking the log, the data now seems more balanced as shown in figure 2

2.2 Predictors

```
my_data_tranformed%>%
  select_if(is.numeric)%>%
  select(-price)%>%
  cor()%>%
  corrplot(title='correlations between all predictors',
           mar=c(0,0,1,0))
```

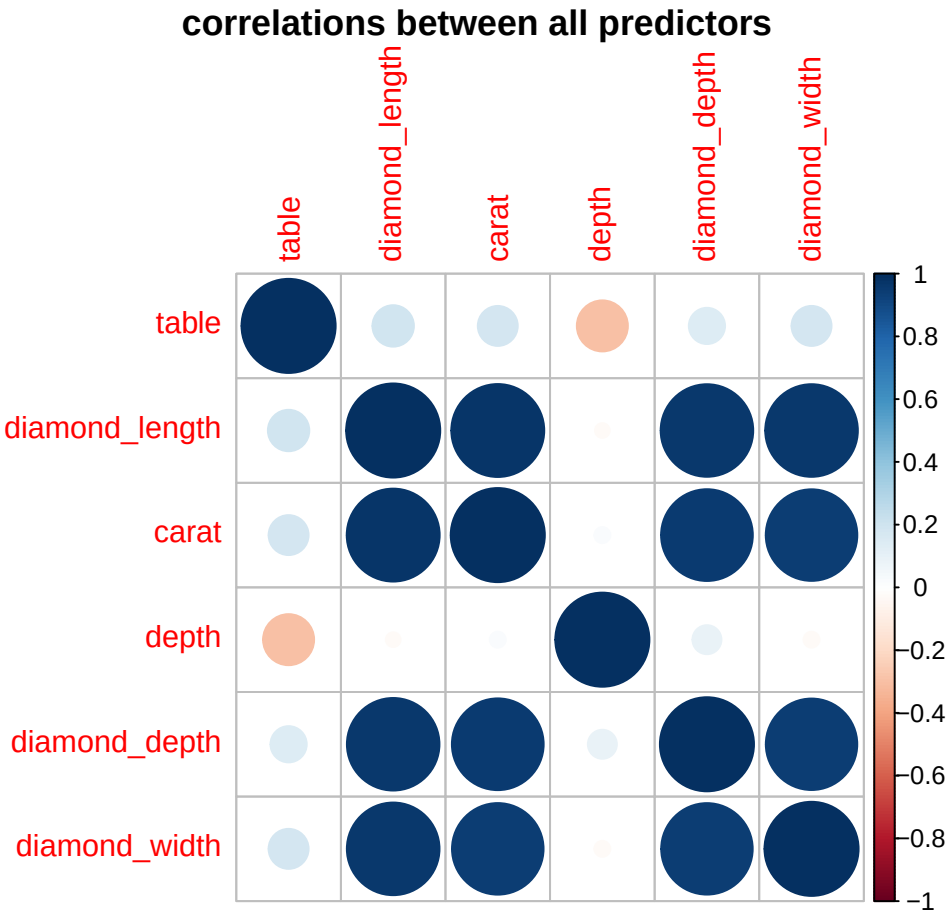

Figure 3: correlations between all predictors

Figure 3 shows the correlation plot between all numerical predictor variables. There are some predictors that are very strongly correlated to each other, they are all information regarding the size of the diamonds (length, width, depth and carat or weight of the diamond), which can be expected. Consequently, having one feature is informative about others features. Hence, all highly correlated features will need to be removed except for one of them so the model does not contains too many redundant features

```
my_data_tranformed%>%
  ggplot(aes(y=price,x=color,fill=color))+
  geom_boxplot()+
  ggtitle('Color and price')+
  ylab('logged price')
```

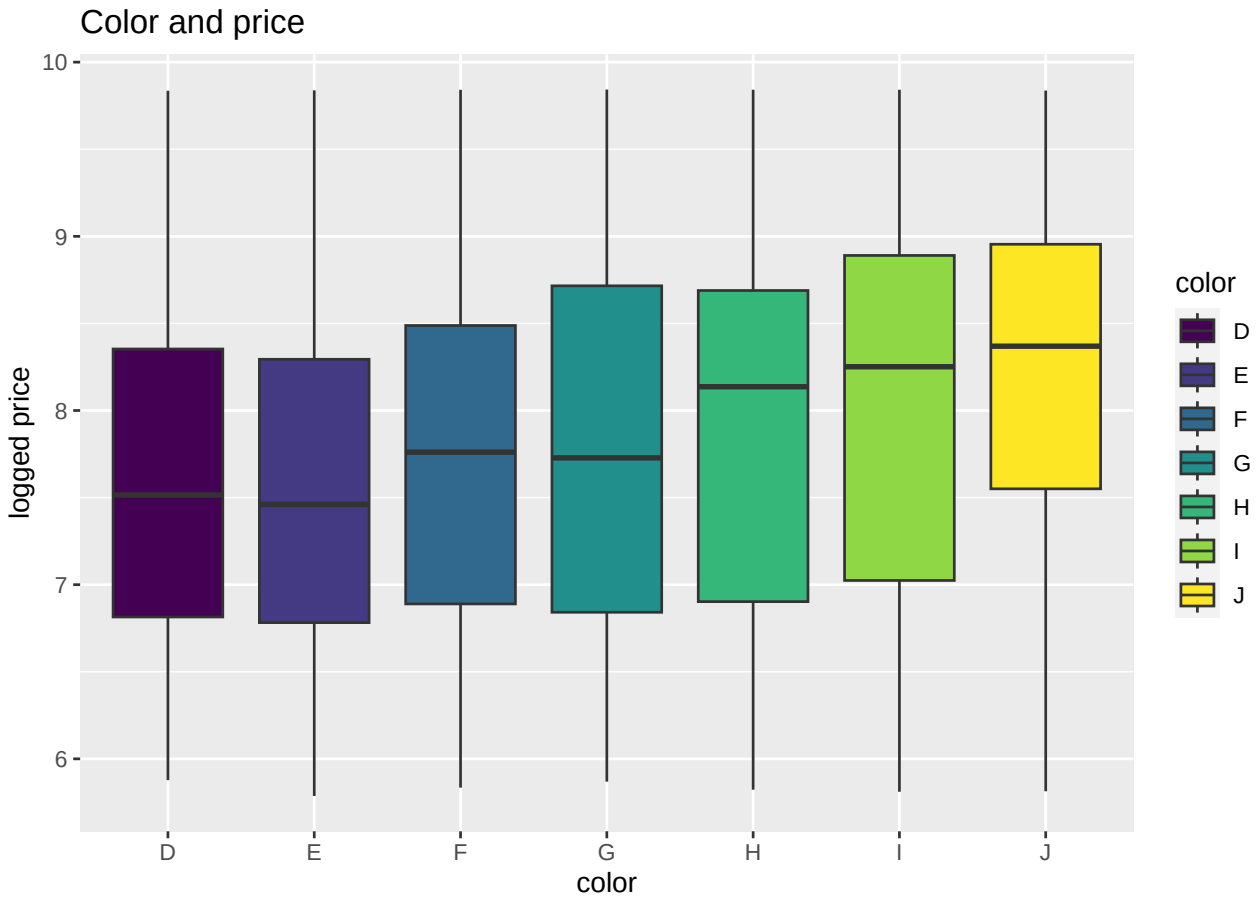

Figure 4: Price distribution for each color

Color seems to have some effect on the price of the diamonds as shown in figure 4. With diamonds of color D and E tend to have the lowest price (lowest median, and lowest interquartile spread), then followed by F and G, then H and I while color J diamonds tend to have the highest price. However, color does not seem to have a very strong effect on price as the distribution seen on the boxplots are relatively close

```
my_data_tranformed%>%
  ggplot(aes(y=price,x=cut,fill=cut))+
  geom_boxplot()+
  ggtitle('Cut and price')+
  ylab('logged price')
```

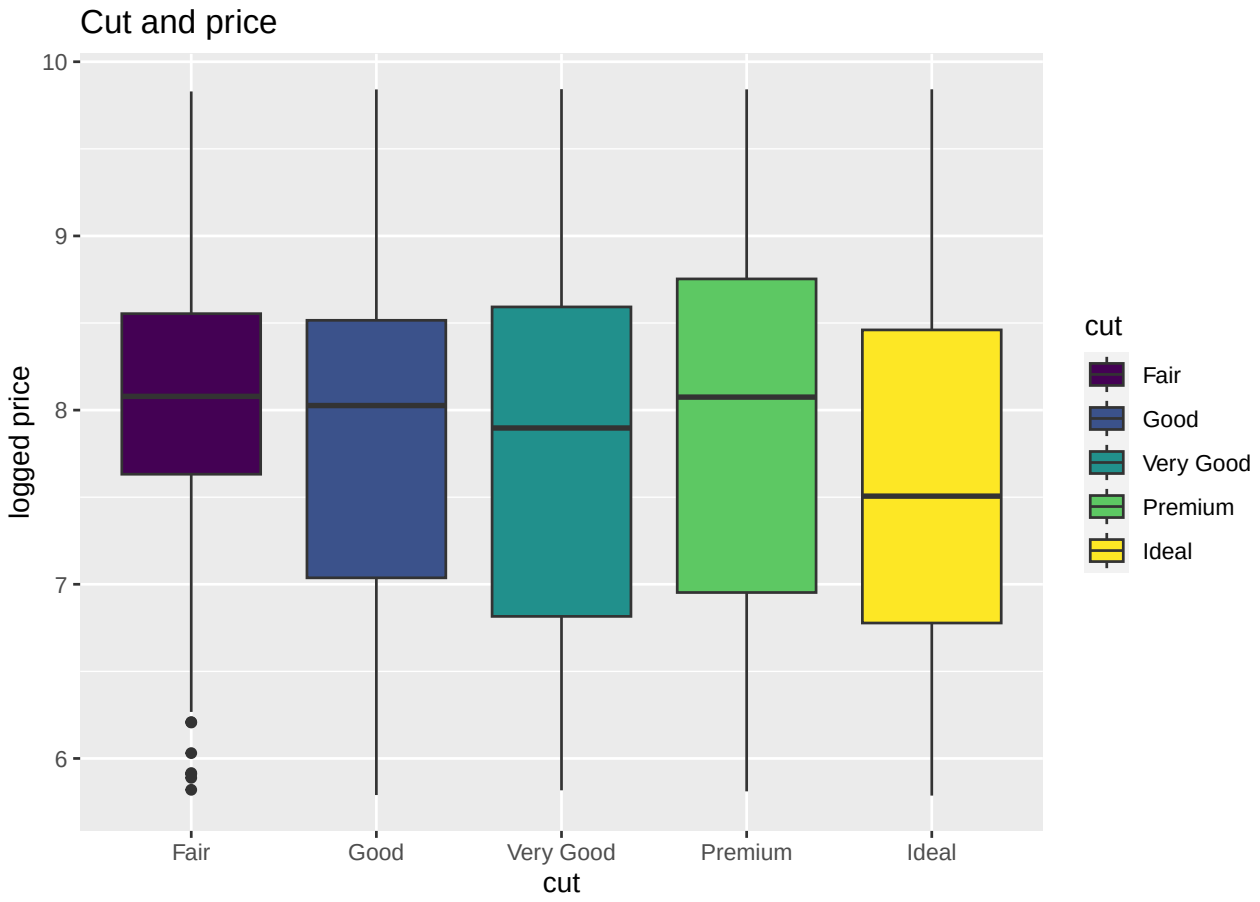

Figure 5: Price distribution for each cut

Looking at 5, diamonds with fair cut tend to have prices within a smaller range compared to others cut. While all cut have around the similar mid-point price, except for ideal cut, whose median price is noticeably lower. However, same as color, cut does not seem to have a very strong effect on price as the distribution seen on the boxplots are relatively close

3 Model fitting

3.1 Splitting data and creating folds

For testing and validating models, the data will be split as 80% for training and 20% for testing, i.e 34522 observations for the training data and 8630 observations for the testing data. For tuning the parameter of the regression random forest model, 5-fold cross validation will be used.

```
set.seed(2023)
## Splitting data
data_split=initial_split(my_data_tranformed,prop=0.8,strata=price)
```

```r
train_data=training(data_split)
test_data=testing(data_split)

## Cross validation folds
my_folds <- vfold_cv(train_data, v = 5,  strata = price)
```

3.2 Recipe for preprocessing

Price will be the response, while all other variables will be used as predictors. Preprocessing step will include normalization of the numerical variable, removal of highly correlated predictors with default threshold of 0.9 absolute correlation, lumping very low frequent categories within categorical variables with threshold 0.01 (for this dataset, this step will not be necessary as no categories fall below the threshold).

```r
my_recipe=recipe(price~.,data=train_data)%>%
  step_normalize(all_numeric_predictors())%>% # normalization
  step_corr(all_numeric_predictors())%>% #correlation
  step_other(all_nominal_predictors(),threshold = 0.01) #lumping factors
```

After preprocessing, the dataset is as follow

```r
knitr::kable(head(my_recipe%>%prep()%>%bake(new_data = NULL)),
  caption = "data set after going through preprocessing")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 4: data set after going through preprocessing

cut	table	carat	depth	color	price
Premium	1.1375496	-1.0308431	-0.3839058	H	6.687109
Very Good	0.2434331	-1.0518616	-0.2440289	G	6.651572
Ideal	-0.6506834	-1.0308431	-0.0342135	E	6.641182
Ideal	-0.6506834	-1.0308431	0.6651711	D	6.595781
Premium	0.2434331	-0.9467693	0.3154788	E	6.825460
Ideal	-0.6506834	-0.9677877	0.0357249	E	6.390241

It can be seen that there is only 6 columns (including the logged price response) and 3 variables (diamond's length, width and depth, the depth columns is a different measure) has been removed and only carat is kept.

3.3 model and workflow

For the random forest, the number of predictors used to build the each tree and the minimum number of observation for a node to be split will be the hyperparameter to tune. The model will build 500 trees before making the prediction. The model and the workflow is as follow:

```
## model
rf_model <- rand_forest(mtry = tune(),
                        min_n = tune(),
                        trees = 500) %>%
  set_mode("regression") %>%
  set_engine("ranger", importance = "impurity")

## workflow
wf <- workflow() %>%
  add_recipe(my_recipe) %>%
  add_model(rf_model)
```

3.4 Tuning model

3.4.1 Tune grid

For the number of predictors, values between 1 and 5 will be tested, and for the minimum number of observations at a leaf node for a tree to continue to be split, values of 2, 11, 21, 30 and 40 will be tested. So in total 25 models will be tested for the tuning process

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

3.4.2 Tuning and saving results

```
doParallel::registerDoParallel()

## tuning
rf_tune <- tune_grid(wf,
                     resamples = my_folds,
                     grid = my_grid)

## saving to my director the tuning result
write_rds(rf_tune,"rf_tune.rds")
```

Tuning result within a fold looks as follow in table 5

```
knitr::kable(head(rf_tune[[3]][[1]]), caption = "tuning result")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 5: tuning result

mtry	min_n	.metric	.estimator	.estimate	.config
1	2	rmse	standard	0.3253926	Preprocessor1_Model01
1	2	rsq	standard	0.9327912	Preprocessor1_Model01
2	2	rmse	standard	0.2109693	Preprocessor1_Model02
2	2	rsq	standard	0.9568985	Preprocessor1_Model02
3	2	rmse	standard	0.2158221	Preprocessor1_Model03
3	2	rsq	standard	0.9548483	Preprocessor1_Model03

```
rf_tune%>%autoplot()
```

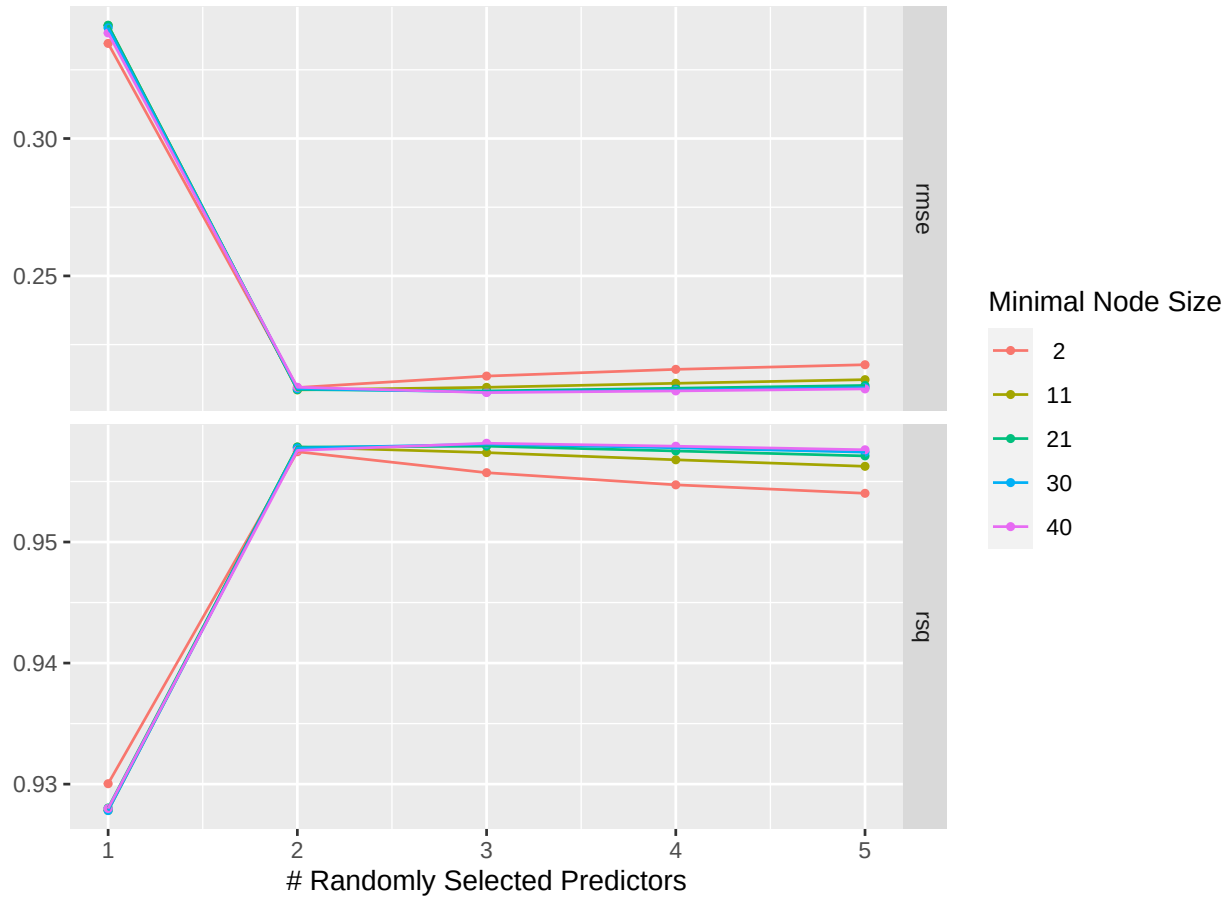

Figure 6: Tuning result

Figure 6 illustrates that between all tested models, only models using a single predictors to build trees seems to perform worse (higher RMSE, and lower RSQ), while other models seem

to perform very similarly to each other with RMSE at around 0.22 and R squared close to 0.96. Because of how close these models are, the simpler the model will be preferred, which in this case means the model that make less rigorous split (i.e min_n=40 would be the least rigorous splitting). However, it is worth having a look at the best model from the tuning table:

```
select_best(rf_tune, metric = "rsq")
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     3    40 Preprocessor1_Model23
```

Coincidentally, the model that perform marginally better than others is also the one with min_n=40, and mtry=3. Hence this model will be used.

3.5 Performance on the test set

```
doParallel::registerDoParallel()

## finalizing the workflow with the best model on R squared metric
wf <- wf %>%
  finalize_workflow(select_best(rf_tune, metric = "rsq"))
## fitting the model on the test set
test_fit <- wf %>% last_fit(split = data_split)
## collecting performance metrics
test_fit %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.209 Preprocessor1_Model1
## 2 rsq     standard       0.958 Preprocessor1_Model1
```

The model performance on the test set is very similar to the performance of this model during cross validation, which means it also performs relatively well. A scatter plot comparing actual price (no log) versus the predicted price (the exponent of the predicted value) is shown below

```
test_fit %>% collect_predictions() %>%
  ggplot(aes(exp(price), exp(.pred))) +
  geom_point() +
  geom_smooth(method = "lm") +
  geom_abline(intercept = 0, slope = 1)+
  xlab('actual price')+
  ylab('predicted price')
```

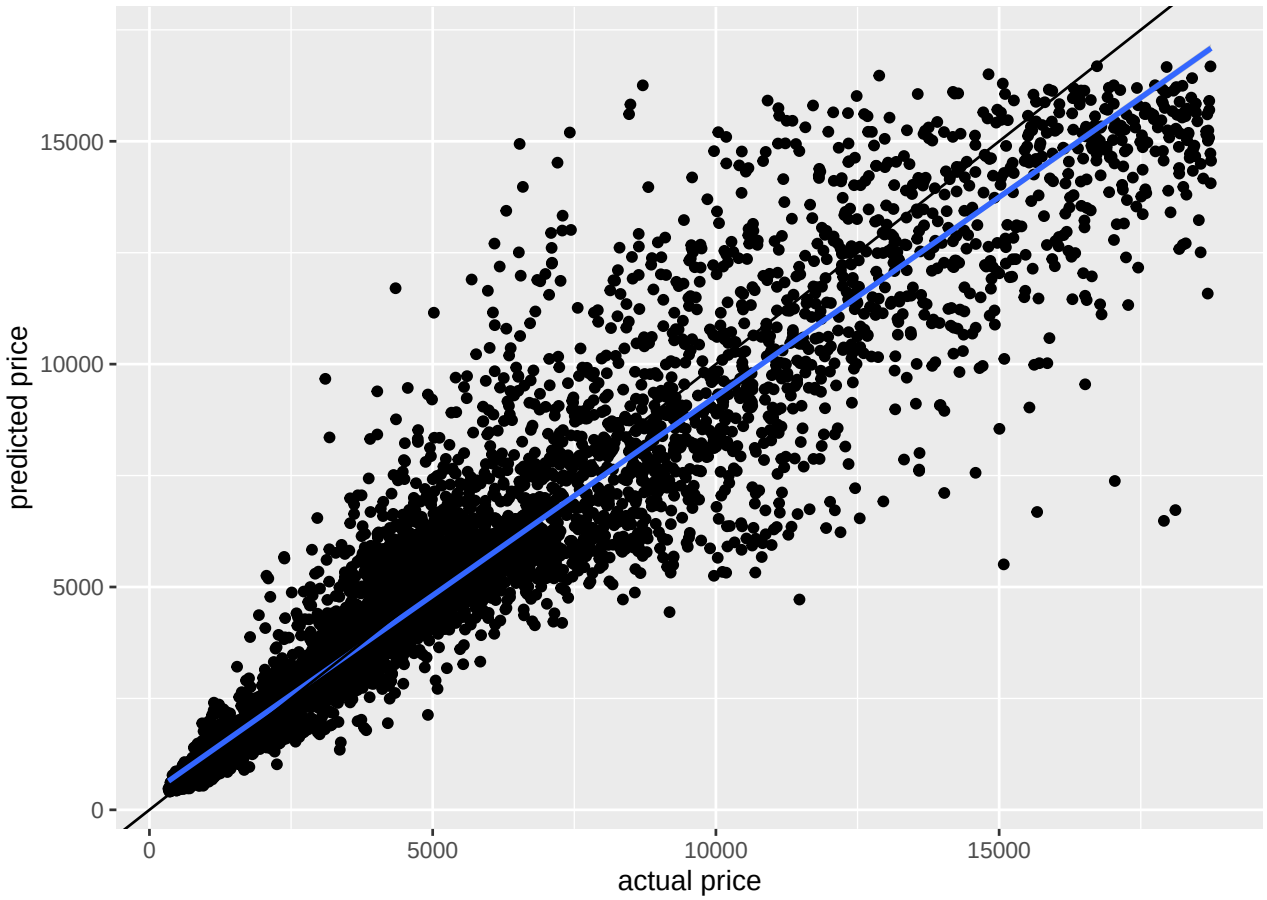

Figure 7: Test prediction vs actual test logged price

Overall, the model performs much better at the lower price (at around $7000 or less), but tend to underestimates price of more expensive diamonds.

3.6 Variable importance

```
final_fit=wf%>%
  fit(train_data)
```

```
final_fit %>% extract_fit_parsnip() %>% vip()+
  ggtitle('Variable importance')
```

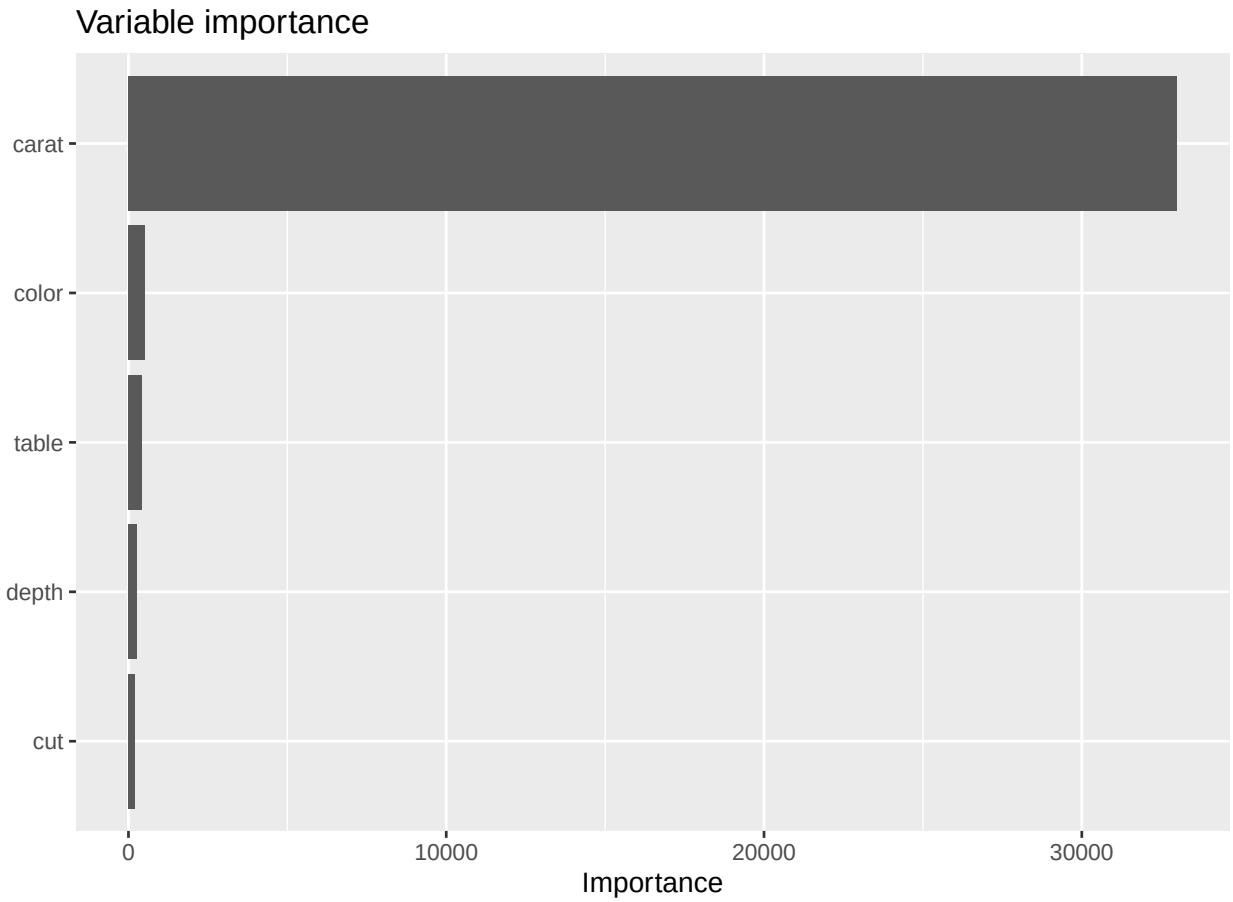

Figure 8: variable importance plot

Further inspection of the model (figure 8) shows that the model prediction is heavily influenced by carat, which means that according to the model, bigger diamonds tend to have higher price. while other predictors did not contribute as much.

4 Prediction on made up data

```
## generating the new data
set.seed(1)
madeup_data=tibble(carat=runif(2,0.2,1.04),
                   cut=c('Fair','Very Good'),
                   color=c('D', 'F'),
                   depth=runif(2,43,79),
                   diamond_depth=runif(2,2.91,4.04),
                   diamond_width=runif(2,4.73,6.54),
                   diamond_length=runif(2,4.72,6.54),
```

```
                    table=runif(2,56,59)
                    )
```

The new made-up diamonds characteristic is shown below

```
knitr::kable(madeup_data[,1:5], caption = "made up data")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 6: made up data

carat	cut	color	depth	diamond_depth
0.4230273	Fair	D	63.62272	3.137901
0.5125841	Very Good	F	75.69548	3.925180

```
knitr::kable(madeup_data[,6:8], caption = "made up data")%>%
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Table 7: made up data

diamond_width	diamond_length	table
6.439862	5.864988	56.61792
5.926044	4.832451	56.52967

And the model predict their price as follow (actual price, not logged price)

```
as.list(final_fit%>%
  predict(new_data = madeup_data))%>%
  lapply(exp)
```

```
## $.pred
## [1]  864.4705 1434.7393
```