Analysis about IKEA furniture.

Fangning Tian

1 Load Libraries

```
pacman::p_load(tidyverse, tidymodels,randomForest,ranger)
```

2 Load Data

```
diamonds2 <- readRDS("~/Assignment/data taming/R code/diamonds2.rds")
diamonds2
```

```
## # A tibble: 5,151 × 10
##    carat depth table price     x     y cut       colour clarity     z
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>     <fct>  <fct>   <dbl>
##  1  1.11  62.3    56  4645  6.59  6.64 ideal     F      SI1      4.12
##  2  1.24  58.9    59  5972  7.08  7.02 premium   F      SI1      4.15
##  3  1     64.5    59  4541  6.25  6.31 good      A      SI2      4.05
##  4  1.13  61.4    57  4936  6.72  6.69 ideal     F      SI1      4.12
##  5  0.31  63.5    56   571  4.29  4.31 good      A      SI1      2.73
##  6  0.51  61.5    55  1438  5.16  5.18 ideal     B      SI1      3.18
##  7  0.32  60.9    57   612  4.46  4.44 ideal     D      SI1      2.71
##  8  0.3   63.2    57   675  4.3   4.25 very good B      SI1      2.7
##  9  1.21  61.8    56  8863  6.82  6.86 ideal     B      SI1      4.23
## 10  0.78  64.6    56  2359  5.86  5.79 fair      D      SI1      3.76
## # ℹ 5,141 more rows
```

Table 1: The diamonds2.rds dataset.

The data is contained within the file diamonds2.rds. The data is read in to R. There are 5,151 rows in the diamonds2 data.

Log-transform the price variable

```
diamonds2 <- diamonds2 %>%
  mutate(price = log(price))
diamonds2
```

```
## # A tibble: 5,151 × 10
##    carat depth table price     x     y cut       colour clarity     z
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>     <fct>  <fct>   <dbl>
##  1  1.11  62.3    56  8.44  6.59  6.64 ideal     F      SI1      4.12
##  2  1.24  58.9    59  8.69  7.08  7.02 premium   F      SI1      4.15
##  3  1     64.5    59  8.42  6.25  6.31 good      A      SI2      4.05
##  4  1.13  61.4    57  8.50  6.72  6.69 ideal     F      SI1      4.12
##  5  0.31  63.5    56  6.35  4.29  4.31 good      A      SI1      2.73
##  6  0.51  61.5    55  7.27  5.16  5.18 ideal     B      SI1      3.18
##  7  0.32  60.9    57  6.42  4.46  4.44 ideal     D      SI1      2.71
##  8  0.3   63.2    57  6.51  4.3   4.25 very good B      SI1      2.7
##  9  1.21  61.8    56  9.09  6.82  6.86 ideal     B      SI1      4.23
## 10  0.78  64.6    56  7.77  5.86  5.79 fair      D      SI1      3.76
## # ℹ 5,141 more rows
```

Table 2: The diamonds2.rds dataset after transforming.

Split the data into training and testing sets

```
set.seed(2024)
data_split <- initial_split(diamonds2,strata = price)
train_data <- training(data_split)
test_data <- testing(data_split)
```

We want to ensure that our training and testing sets are balanced according to price, so we specify strata = price in the function initial_split().

Perform cross-validation

```
set.seed(2024)
cv_folds <- vfold_cv(train_data,strata = price)
cv_folds
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 × 2
##    splits             id
##    <list>             <chr>
##  1 <split [3474/388]> Fold01
##  2 <split [3474/388]> Fold02
##  3 <split [3474/388]> Fold03
##  4 <split [3474/388]> Fold04
##  5 <split [3474/388]> Fold05
##  6 <split [3476/386]> Fold06
##  7 <split [3478/384]> Fold07
##  8 <split [3478/384]> Fold08
##  9 <split [3478/384]> Fold09
## 10 <split [3478/384]> Fold10
```

Table 3: 10 folds cross-validation.

Next, we will partition our training set into 10 folds.

Define the recipe for preprocessing

```
pacman::p_load(textrecipes)
diamonds_recipe <- recipe(price ~ carat + depth + table + x + y + z + cut + colour + clarity, d
ata = train_data)%>%
  step_impute_mean(carat,depth, table, x,y,z)
diamonds_recipe %>% prep() %>% bake(new_data=NULL)
```

```
## # A tibble: 3,862 × 10
##    carat depth table     x     y     z cut       colour clarity price
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>     <fct>  <fct>   <dbl>
##  1  0.31  63.5    56  4.29  4.31  2.73 good      A      SI1      6.35
##  2  0.32  60.9    57  4.46  4.44  2.71 ideal     D      SI1      6.42
##  3  0.3   63.2    57  4.3   4.25  2.7  very good B      SI1      6.51
##  4  0.31  61.6    55  4.36  4.37  2.69 ideal     C      VS2      6.44
##  5  0.3   63      55  4.28  4.29  2.7  very good C      VS2      6.26
##  6  0.26  62.2    55  4.06  4.11  2.54 ideal     E      VS1      6.14
##  7  0.25  59.8    59  4.07  4.09  2.44 very good B      VVS2     6.35
##  8  0.3   62.2    57  4.27  4.28  2.66 ideal     E      SI1      6.11
##  9  0.41  63.5    56  4.7   4.75  3    good      C      SI1      6.56
## 10  0.3   59.8    59  4.3   4.43  2.61 very good C      VS1      6.56
## # ℹ 3,852 more rows
```

Table 4:The recipe for preprocessing

Define the random forest model

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

we specify a random forest model with 500 trees.

Create a workflow

```
rf_workflow <- workflow() %>%
  add_recipe(diamonds_recipe) %>%
  add_model(rf_model)
rf_workflow
```

```
## ═══ Workflow ════════════════════════════════════════════════════
═══════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────
──────────────────────────────────
## 1 Recipe Step
##
## • step_impute_mean()
##
## ── Model ───────────────────────────────────────────────────────
──────────────────────────────────
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 500
##   min_n = tune()
##
## Computational engine: ranger
```

we can combine our recipe and model into a workflow.

Define the tuning grid

```
tune_grid <- grid_regular(
  mtry(range = c(1, 7)),
  min_n(),
  levels = 7
)

tune_grid
```

```
## # A tibble: 49 × 2
##     mtry min_n
##    <int> <int>
##  1     1     2
##  2     2     2
##  3     3     2
##  4     4     2
##  5     5     2
##  6     6     2
##  7     7     2
##  8     1     8
##  9     2     8
## 10     3     8
## # ℹ 39 more rows
```

Table 4:The hyperparameters between 1 and 5 variables

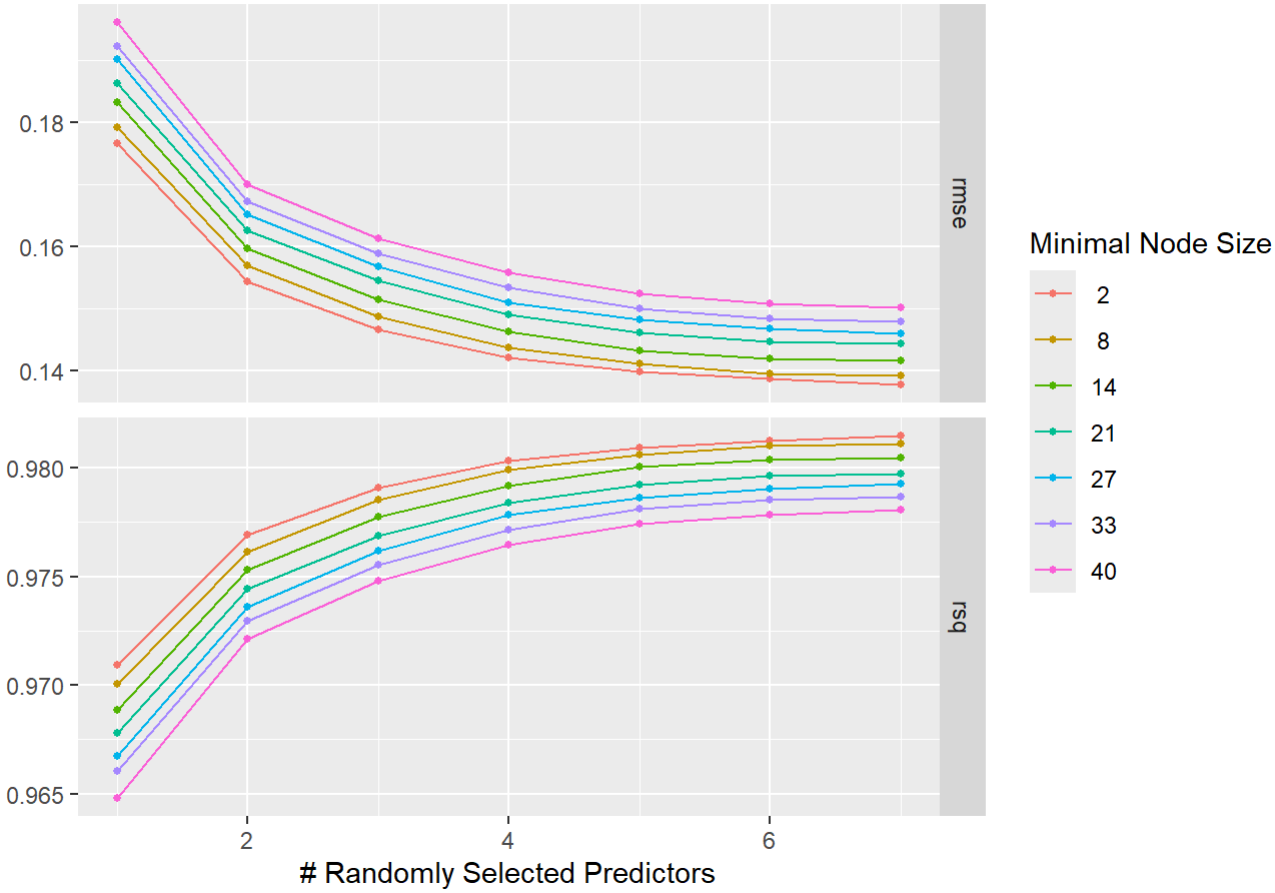
We tune hyperparameters in our random forest.

Tune the model

```
doParallel::registerDoParallel()
diamond_tune <- tune_grid(rf_workflow,
                      resamples = cv_folds,
                      grid = tune_grid)

diamond_tune %>% autoplot()+
  labs(title = "Figure 1: The result of each hyperparameter affects our model performance.")
```

Figure 1: The result of each hyperparameter affects our model performance.



We can fit our random forest models. For each hyperparameter combination in our grid, we will build 500 trees. From the figure, we can see 2 of min_n gives the best model.

select the best model

```
show_best(diamond_tune, metric = "rmse")
```

```
## # A tibble: 5 × 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     7     2 rmse    standard   0.138    10 0.00302 Preprocessor1_Model107
## 2     6     2 rmse    standard   0.139    10 0.00282 Preprocessor1_Model106
## 3     7     8 rmse    standard   0.139    10 0.00288 Preprocessor1_Model114
## 4     6     8 rmse    standard   0.140    10 0.00275 Preprocessor1_Model113
## 5     5     2 rmse    standard   0.140    10 0.00272 Preprocessor1_Model105
```

Table 5:The result of best models through rmse.

From the table, we can see 7 of of mtry gives the best model.

Finalize the workflow with the best parameters

```
rf_workflow <- rf_workflow %>%
  finalize_workflow(select_best(diamond_tune, metric = "rmse"))
rf_workflow
```

```
## ═══ Workflow ════════════════════════════════════════════════════════════
════════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────────────
────────────────────────────────────
## 1 Recipe Step
##
## • step_impute_mean()
##
## ── Model ───────────────────────────────────────────────────────────────
────────────────────────────────────
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = 7
##   trees = 500
##   min_n = 2
##
## Computational engine: ranger
```

Fitting the Final Model

```
diamond_fit <- rf_workflow %>% last_fit(split = data_split)
```

With our finalised workflow, we are ready to evaluate the performance of our chosen model using our testing data.

#calculate the regular performance metrics.

```
diamond_fit %>% collect_metrics()
```

```
## # A tibble: 2 × 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.142 Preprocessor1_Model1
## 2 rsq     standard       0.980 Preprocessor1_Model1
```

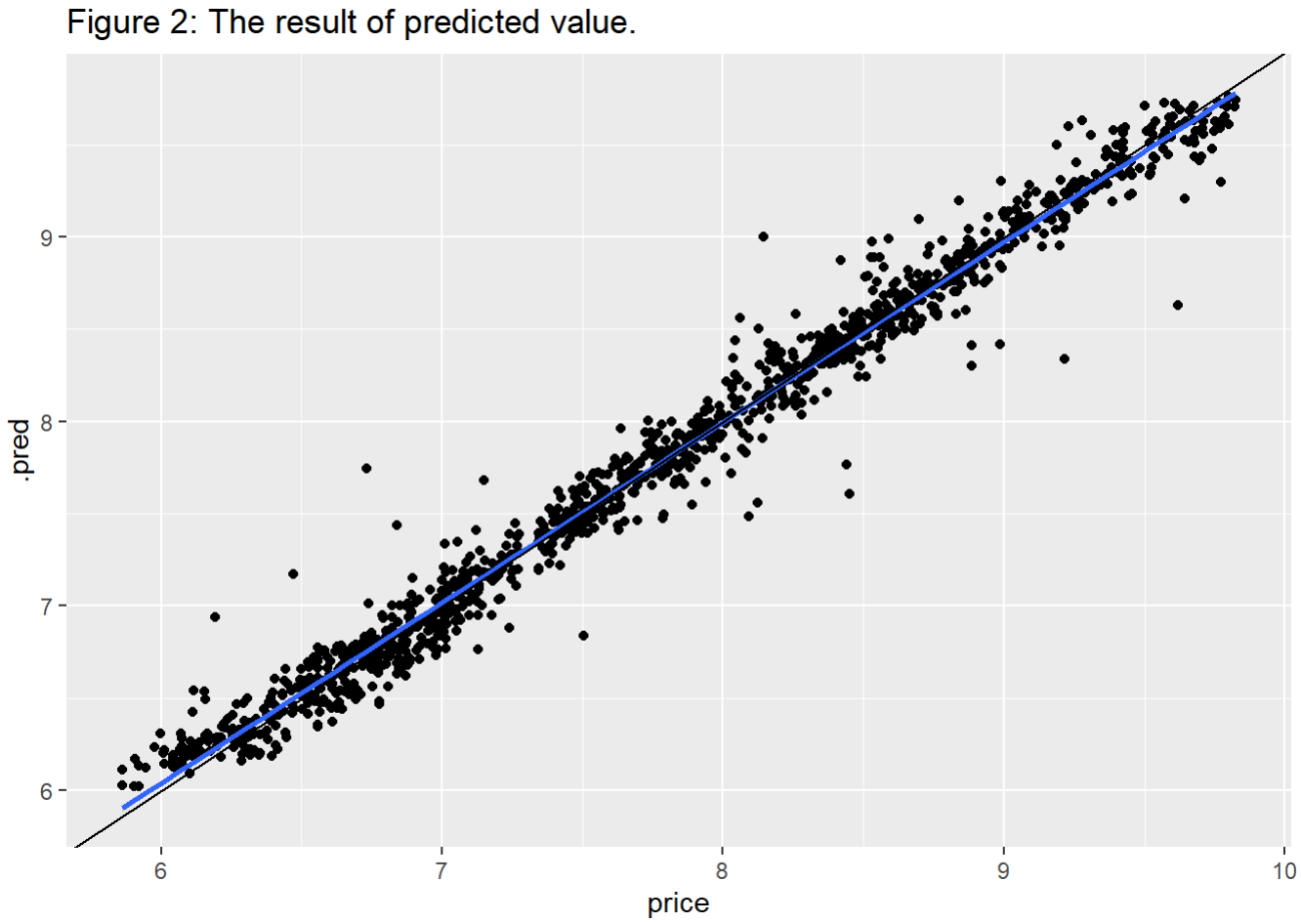
Table 6:The result of rmse and rsq.

For the final model fitted to the testing data, rmse is 0.142 and rsq is 0.980.

visualise the predicted values

```
diamond_fit %>% collect_predictions() %>%
  ggplot(aes(price, .pred)) +
  geom_point() +
  geom_smooth(method = "lm") +
  geom_abline(intercept = 0, slope = 1) +
  labs(title = "Figure 2: The result of predicted value.")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



From the figure, we can see our chosen model tend to overestimate the price for cheaper diamond and underestimate for more expensive ones.