Final Project

Xilong Li (3467966)

2022-06-09

Contents

Introduction

Loading Data and Packages

```
library(tidymodels)
library(tidyverse)
library(MASS)
library(glmnet)
library(janitor)
library(discrim)
library(poissonreg)
library(klaR)
library(rpart.plot)
library(vip)
library(randomForest)
library(xgboost)
library(corrplot)
library(ranger)
```

- gross_profit:
- category:
- volume:

- volume_range:
- package:
- flavor:
- price:
- StorePerOrder:
- AmountPerOrder:
- Sale_Store:
- initial_days:

```
original_data <- read.csv("Translated_data.csv")
head(original_data)
```

gross_profit	category	volume	volume_range	package	flavor	price	StorePerOrder	AmountPerOrder	Sale_Store	initial_days	X
15.059149	Biscuit	115g	100-299g/ml	Bag	Original	3.5	7.2631	1.8930085	198	2841	NA
0.000000	Cookies	44g	0-149g/ml	Bag	Chocolate	4.9	0.0000	0.0000000	0	1220	NA
4.763544	Pastry	60g	0-149g/ml	Box	Vanilla	5.0	10.5572	2.2596862	80	2312	NA
103.966349	Pastry	84g	0-149g/ml	Bag	Original	5.5	1.9736	0.2974709	215	2841	NA
41.006804	Biscuit	168g	150-199g/ml	Bag	Original	6.5	5.4526	0.8248753	172	2841	NA
9.234831	Biscuit	60g	0-149g/ml	Box	Strawberry	6.9	15.9553	2.5869891	32	420	NA

```
dim(original_data)
```

```
## [1] 550  12
```

```
set.seed(2216)
```

Data Cleansing

```
original_data <- original_data %>%
  clean_names()

head(original_data)
```

gross_profit	category	volume	volume_range	package	flavor	price	store_per_order	amount_per_order	sale_store	initial_days	x
15.059149	Biscuit	115g	100-299g/ml	Bag	Original	3.5	7.2631	1.8930085	198	2841	NA
0.000000	Cookies	44g	0-149g/ml	Bag	Chocolate	4.9	0.0000	0.0000000	0	1220	NA
4.763544	Pastry	60g	0-149g/ml	Box	Vanilla	5.0	10.5572	2.2596862	80	2312	NA
103.966349	Pastry	84g	0-149g/ml	Bag	Original	5.5	1.9736	0.2974709	215	2841	NA
41.006804	Biscuit	168g	150-199g/ml	Bag	Original	6.5	5.4526	0.8248753	172	2841	NA
9.234831	Biscuit	60g	0-149g/ml	Box	Strawberry	6.9	15.9553	2.5869891	32	420	NA

```r
selected_data <- original_data %>%
  mutate(category = factor(category),
         volume_range = factor(volume_range),
         package = factor(package),
         flavor = factor(flavor),
         volume = unlist(strsplit(volume, split='g', fixed=TRUE)),
         volume = as.numeric(volume)) %>%
  dplyr::select(
    gross_profit,
    category,
    volume,
    volume_range,
    package,
    flavor,
    price,
    store_per_order,
    amount_per_order,
    sale_store,
    initial_days) %>%
  filter(gross_profit != 0)

head(selected_data)
```

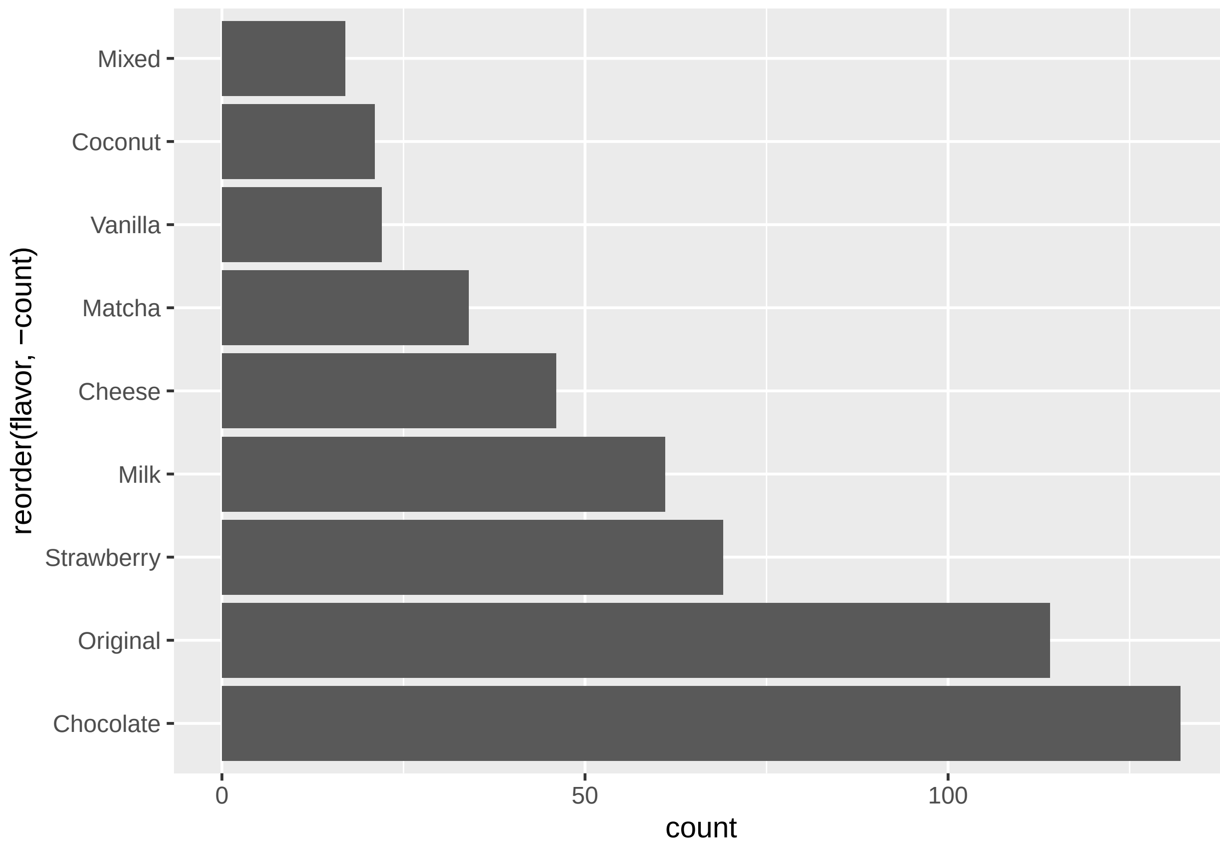
gross_profit	category	volume	volume_range	package	flavor	price	store_per_order	amount_per_order	sale_store	initial_days
15.059149	Biscuit	115	100-299g/ml	Bag	Original	3.5	7.2631	1.8930085	198	2841
4.763544	Pastry	60	0-149g/ml	Box	Vanilla	5.0	10.5572	2.2596862	80	2312
103.966349	Pastry	84	0-149g/ml	Bag	Original	5.5	1.9736	0.2974709	215	2841
41.006804	Biscuit	168	150-199g/ml	Bag	Original	6.5	5.4526	0.8248753	172	2841
9.234831	Biscuit	60	0-149g/ml	Box	Strawberry	6.9	15.9553	2.5869891	32	420
48.648763	Pastry	40	0-149g/ml	Bag	Chocolate	6.9	8.8013	1.1966070	31	198

Data Splitting and Cross-Validation

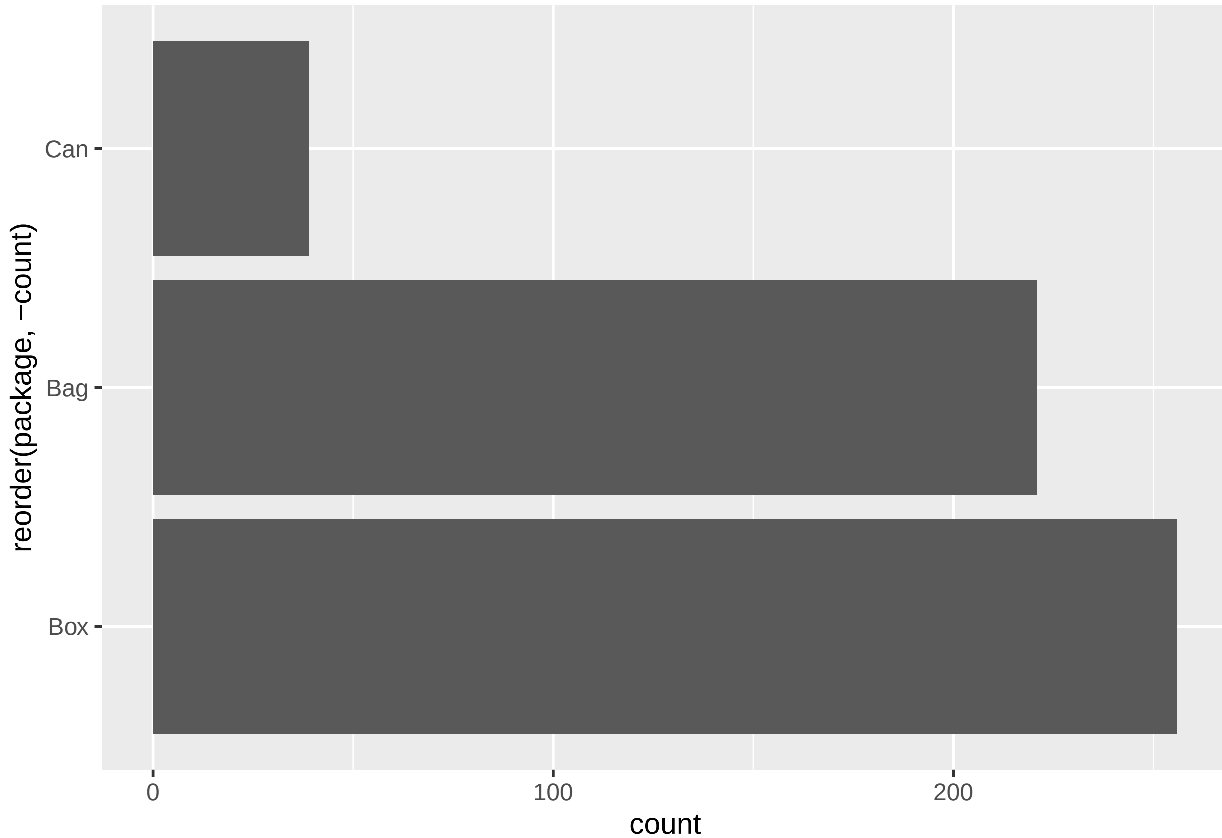
```
sales_split <- initial_split(selected_data, prop = 0.80)

sales_train <- training(sales_split)
sales_test <- testing(sales_split)

train_folds <- vfold_cv(sales_train, v = 5)
```

Exploratory Data Analysis

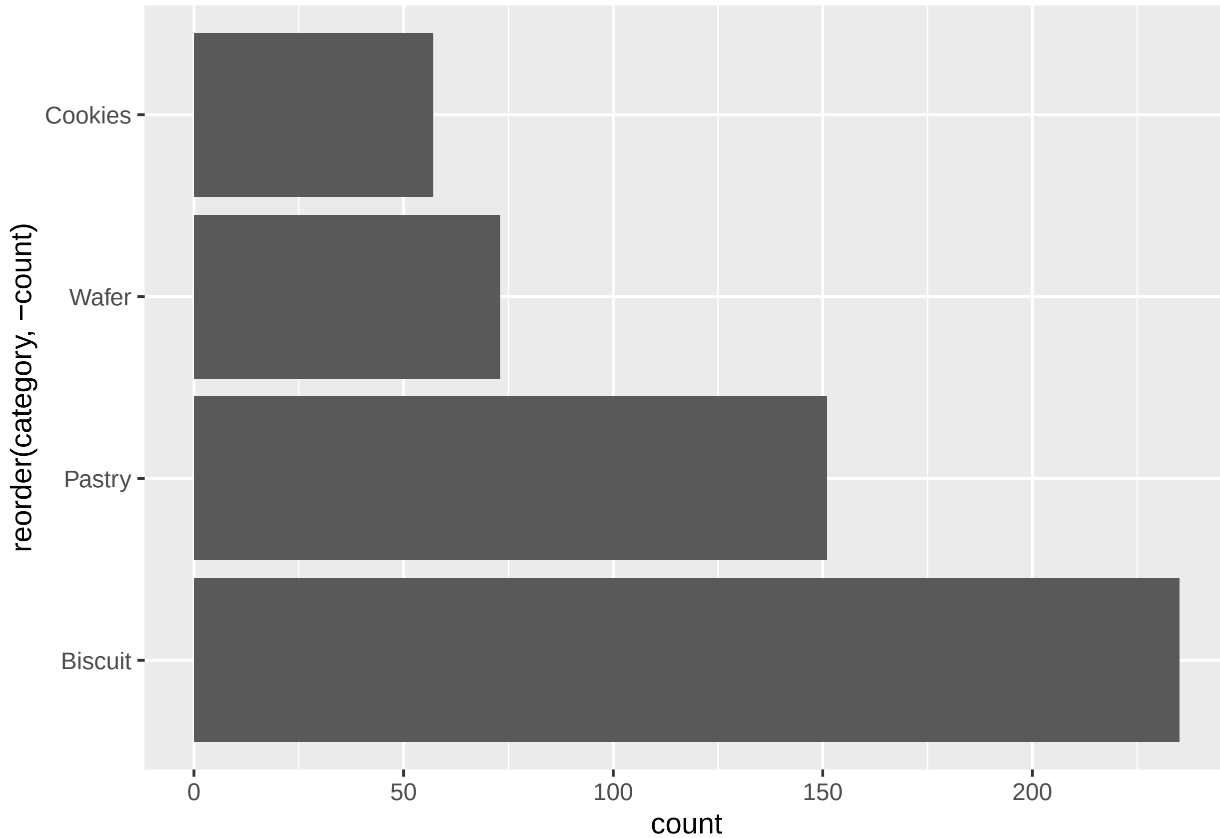
```
ordered_data_1 <- selected_data %>%
  group_by(flavor) %>%
  summarise(count = n()) %>%
  arrange(count)
ggplot(ordered_data_1, aes(x = count, y = reorder(flavor, -count))) + geom_bar(stat = "identity")
```



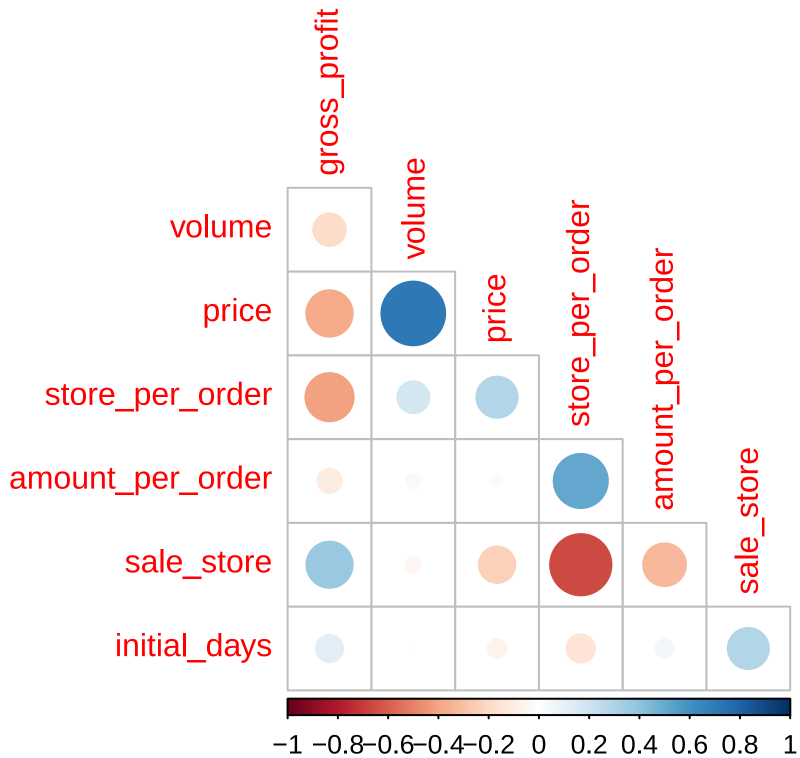

```
ordered_data_2 <- selected_data %>%
  group_by(package) %>%
  summarise(count = n()) %>%
  arrange(count)
ggplot(ordered_data_2, aes(x = count, y = reorder(package, -count))) + geom_bar(stat = "identity")
```




```
ordered_data_3 <- selected_data %>%
  group_by(category) %>%
  summarise(count = n()) %>%
  arrange(count)
ggplot(ordered_data_3, aes(x = count, y = reorder(category, -count))) + geom_bar(stat = "identity")
```




```
cor_data <- sales_train %>%
  dplyr::select(where(is.numeric))
corrplot(cor(cor_data), type = 'lower',diag = FALSE)
```



Model Fitting

Setting Up Recipe

```
sales_recipe <- recipe(gross_profit ~
                         category+
                         volume+
                         package+
                         flavor+
                         price+
                         store_per_order+
                         amount_per_order+
                         sale_store+
                         initial_days,
                       data = sales_train) %>%
  step_dummy(category, package, flavor) %>%
  step_interact(~ volume:starts_with("package")) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

sales_recipe
```

```
## Recipe
```

```
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          9
##
## Operations:
##
## Dummy variables from category, package, flavor
## Interactions with volume:starts_with("package")
## Centering for all_predictors()
## Scaling for all_predictors()
```

1. Linear Regression Model

```
lm_model <- linear_reg() %>%
  set_engine("lm")

lm_wflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(sales_recipe)

lm_fit <- fit(lm_wflow, sales_train)
lm_fit %>%
  # This returns the parsnip object:
  extract_fit_parsnip() %>%
  # Now tidy the linear model object:
  tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	11.5634192	4.089863	2.8273368	0.0049356
volume	0.7183806	9.929324	0.0723494	0.9423609
price	-49.1794437	7.091401	-6.9350815	0.0000000
store_per_order	-28.4652776	6.362336	-4.4740289	0.0000101
amount_per_order	6.5702737	5.015461	1.3100038	0.1909656
sale_store	17.0108606	5.906655	2.8799483	0.0041968
initial_days	-0.2732062	4.482523	-0.0609492	0.9514309
category_Cookies	14.7310239	4.771307	3.0874189	0.0021634
category_Pastry	5.4114702	4.736760	1.1424413	0.2539715
category_Wafer	9.2944727	4.755106	1.9546300	0.0513409
package_Box	-3.4113878	7.378907	-0.4623161	0.6441124
package_Can	-3.0026409	6.716523	-0.4470529	0.6550851
flavor_Chocolate	-22.5086284	7.238155	-3.1097191	0.0020102
flavor_Coconut	-4.9532011	5.080586	-0.9749271	0.3302007
flavor_Matcha	-7.2183828	5.336465	-1.3526526	0.1769503
flavor_Milk	-11.7126578	5.988005	-1.9560200	0.0511763
flavor_Mixed	-0.8497340	5.205699	-0.1632315	0.8704207
flavor_Original	-9.0915400	7.402852	-1.2281131	0.2201452
flavor_Strawberry	-16.2154065	6.150582	-2.6364019	0.0087137
flavor_Vanilla	-13.4799544	5.263861	-2.5608491	0.0108172

term	estimate	std.error	statistic	p.value
volume_x_package_Box	20.1282264	11.225443	1.7930897	0.0737333
volume_x_package_Can	17.4678678	7.531874	2.3191929	0.0209005

```
sales_train_res <- predict(lm_fit, new_data = sales_train %>%
                             dplyr::select(-gross_profit))
sales_train_res %>%
  head()
```

.pred
-81.355342
-34.205174
24.775899
24.692187
-3.752874
19.656663

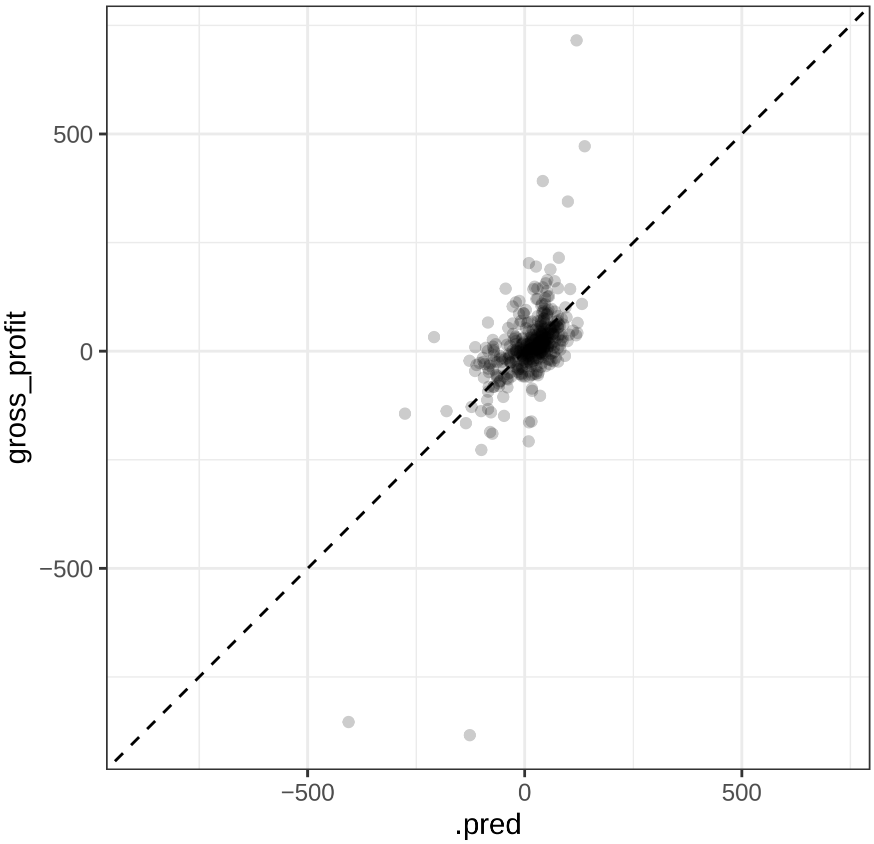
Now we attach a column with the actual observed gross_profit observations:

```
sales_train_res <- bind_cols(sales_train_res, sales_train %>%
                               dplyr::select(gross_profit))
sales_train_res %>%
  head()
```

.pred	gross_profit
-81.355342	-33.520424
-34.205174	11.762315
24.775899	3.082706
24.692187	5.254099
-3.752874	29.757399
19.656663	5.973426

```
# Write Out Results & Workflow:
save(lm_fit, lm_wflow, file = "Data/model_fitting/lm_fit.rda")
```

```
sales_train_res %>%
  ggplot(aes(x = .pred, y = gross_profit)) +
  geom_point(alpha = 0.2) +
  geom_abline(lty = 2) +
  theme_bw() +
  coord_obs_pred()
```



2. Ridge Regression Model

Set up the model specification.

```
ridge_spec <-
  linear_reg(penalty = tune(), mixture = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")
```

Create a workflow object.

```
ridge_workflow <- workflow() %>%
  add_recipe(sales_recipe) %>%
  add_model(ridge_spec)
```

Creates a grid of evenly spaced parameter values.

```
penalty_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 50)
penalty_grid
```

penalty
1.000000e-05

penalty
1.600000e-05
2.560000e-05
4.090000e-05
6.550000e-05
1.048000e-04
1.677000e-04
2.683000e-04
4.292000e-04
6.866000e-04
1.098500e-03
1.757500e-03
2.811800e-03
4.498400e-03
7.196900e-03
1.151400e-02
1.842070e-02
2.947050e-02
4.714870e-02
7.543120e-02
1.206793e-01
1.930698e-01
3.088844e-01
4.941713e-01
7.906043e-01
1.264855e+00
2.023590e+00
3.237458e+00
5.179475e+00
8.286428e+00
1.325711e+01
2.120951e+01
3.393222e+01
5.428675e+01
8.685114e+01
1.389495e+02
2.222996e+02
3.556480e+02
5.689866e+02
9.102982e+02
1.456348e+03
2.329952e+03
3.727594e+03
5.963623e+03
9.540955e+03
1.526418e+04
2.442053e+04
3.906940e+04
6.250552e+04
1.000000e+05

Fit the model.

```r
ridge_tune <- tune_grid(
  ridge_workflow,
  resamples = train_folds,
  grid = penalty_grid
)

# Write Out Results & Workflow:
save(ridge_tune, ridge_workflow, file = "Data/model_fitting/ridge_tune.rda")
```

3. Tree Decision Model

```r
reg_tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("regression")

reg_tree_wf <- workflow() %>%
  add_model(reg_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(sales_recipe)
```

```r
param_grid <- grid_regular(cost_complexity(range = c(-4, -1)), levels = 10)

tree_tune <- tune_grid(
  reg_tree_wf,
  resamples = train_folds,
  grid = param_grid
)

# Write Out Results & Workflow:
save(tree_tune, reg_tree_wf, file = "Data/model_fitting/tree_tune.rda")
```

4. Random Forest Model

```r
# ?rand_forest
rf_spec <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

rf_wf <- workflow() %>%
  add_recipe(sales_recipe) %>%
  add_model(rf_spec)

rf_grid <- grid_regular(mtry(range = c(3,8)), trees(range = c(5,2000)),
                        min_n(range = c(3,8)), levels = 5)
```

```r
rf_tune <- tune_grid(
  rf_wf,
  resamples = train_folds,
  grid = rf_grid
)
```

```r
# Write Out Results & Workflow:
save(rf_tune, rf_wf, file = "Data/model_fitting/rf_tune.rda")
```

5. Boosted Tree Model

```r
boost_spec <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boost_wf <- workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(sales_recipe)

boost_grid <- grid_regular(trees(range = c(10,2000)), levels = 10)
```

```r
boost_tune <- tune_grid(
  boost_wf,
  resamples = train_folds,
  grid = boost_grid
)

# Write Out Results & Workflow:

save(boost_tune, boost_wf, file = "Data/model_fitting/boost_tune.rda")
```

Model Selection and Performance

Load Previous Saved Data

```r
load("data/model_fitting/lm_fit.rda")
load("data/model_fitting/ridge_tune.rda")
load("data/model_fitting/tree_tune.rda")
load("data/model_fitting/rf_tune.rda")
load("data/model_fitting/boost_tune.rda")
```

1. Linear Regression Model

- After fitting the Linear Regression Model, we now compare the fitted value of gross profit to the actual observed value.

```r
sales_train_res <- predict(lm_fit, new_data = sales_train %>%
                              dplyr::select(-gross_profit))
sales_train_res <- bind_cols(sales_train_res, sales_train %>%
                              dplyr::select(gross_profit))
```

- Thus a generated datafame is shown below:

```r
sales_train_res %>%
  head()
```

.pred	gross_profit
-81.355342	-33.520424
-34.205174	11.762315
24.775899	3.082706
24.692187	5.254099
-3.752874	29.757399
19.656663	5.973426

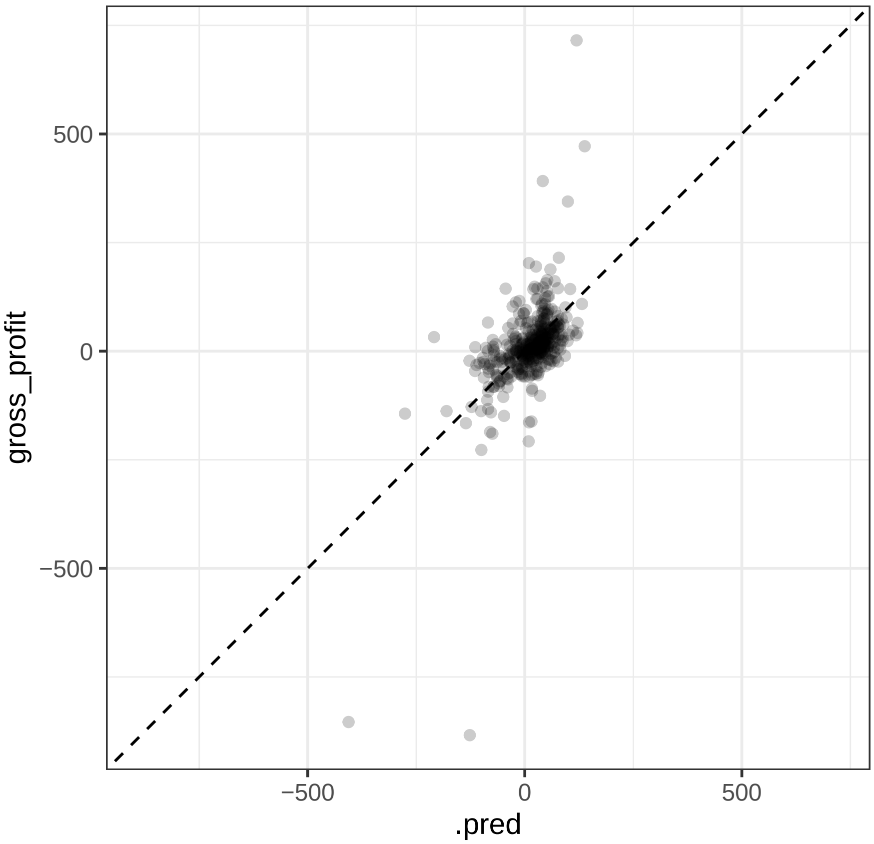
- Then, we decide to calcultae "rmse", "rsq", "mae" values to evaluate the performance of this model:

```r
sales_metrics <- metric_set(rmse, rsq, mae)
sales_metrics(sales_train_res, truth = gross_profit,
                estimate = .pred)
```

.metric	.estimator	.estimate
rmse	standard	80.7683158
rsq	standard	0.3348636
mae	standard	47.2504689

- As we can see, the rmse value is larger than 80, which is really high, and thus this result indicates that the linear regression model might not be suitable to achieve our goal.
- We can also plot and visualize the result:

```r
sales_train_res %>%
  ggplot(aes(x = .pred, y = gross_profit)) +
  geom_point(alpha = 0.2) +
  geom_abline(lty = 2) +
  theme_bw() +
  coord_obs_pred()
```

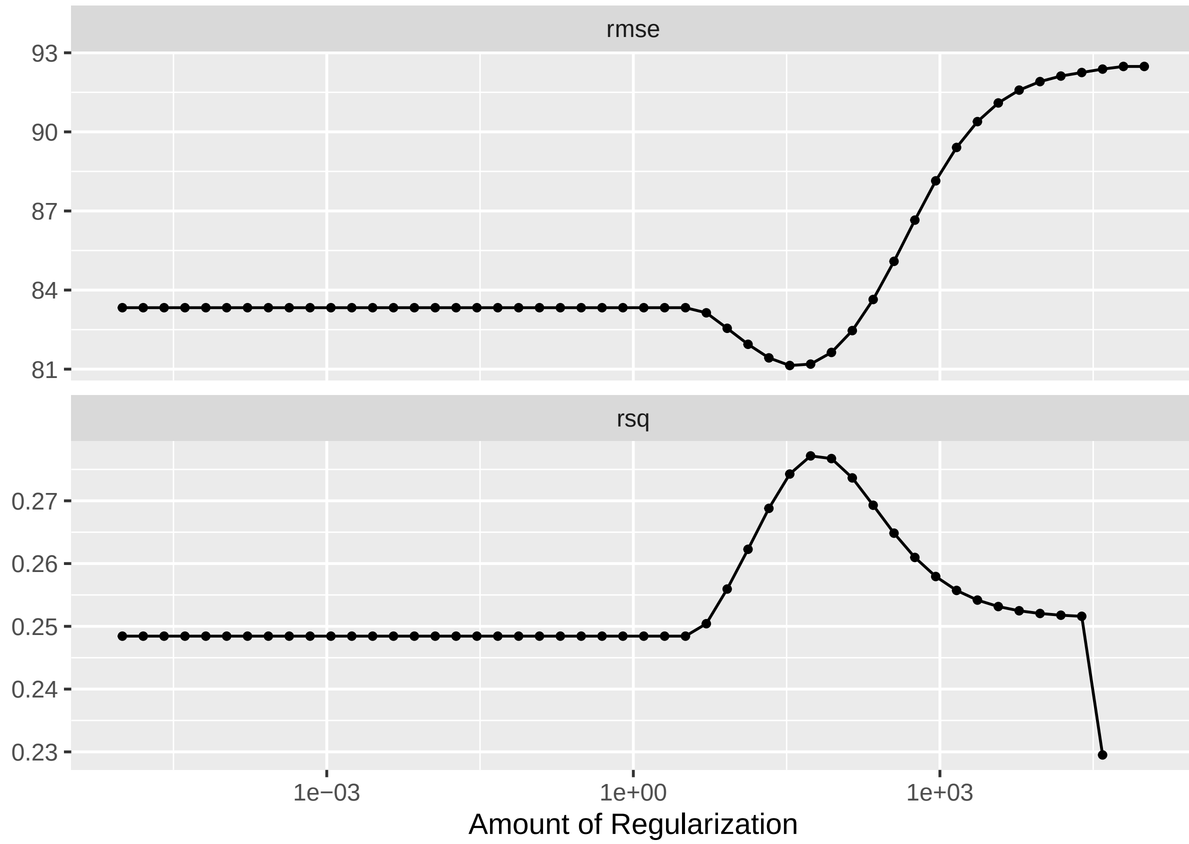


- As it is shown above, all the observed values of gross profit concentrate in a certain area, and there are also many outliers that can influence the prediction.
- Therefore, this linear regression model might NOT be a good way to predict the gross profit.

2. Ridge Regression Model

- After obtaining the output of tune_grid(),we first use autoplot() to create a visualization of the result:

```
autoplot(ridge_tune)
```



- As we can see in the graph, there is a certain point that rmse reaches the lowest level;
- Thus we use show_best() to find the lowest rmse of the best turning ridge model:

```
show_best(ridge_tune, metric = "rmse") %>% dplyr::select(-.estimator, -.config)
```

penalty	.metric	mean	n	std_err
33.93222	rmse	81.13850	5	16.70875
54.28675	rmse	81.19072	5	17.18792
21.20951	rmse	81.42716	5	16.21721
86.85114	rmse	81.63386	5	17.60988
13.25711	rmse	81.94270	5	15.76210

- As it is shown above, the lowest rmse is 77.36626 for the ridge regression model, which is lower than the linear regression model;
- It is also shown that the best penalty is at the level of 33.93222;

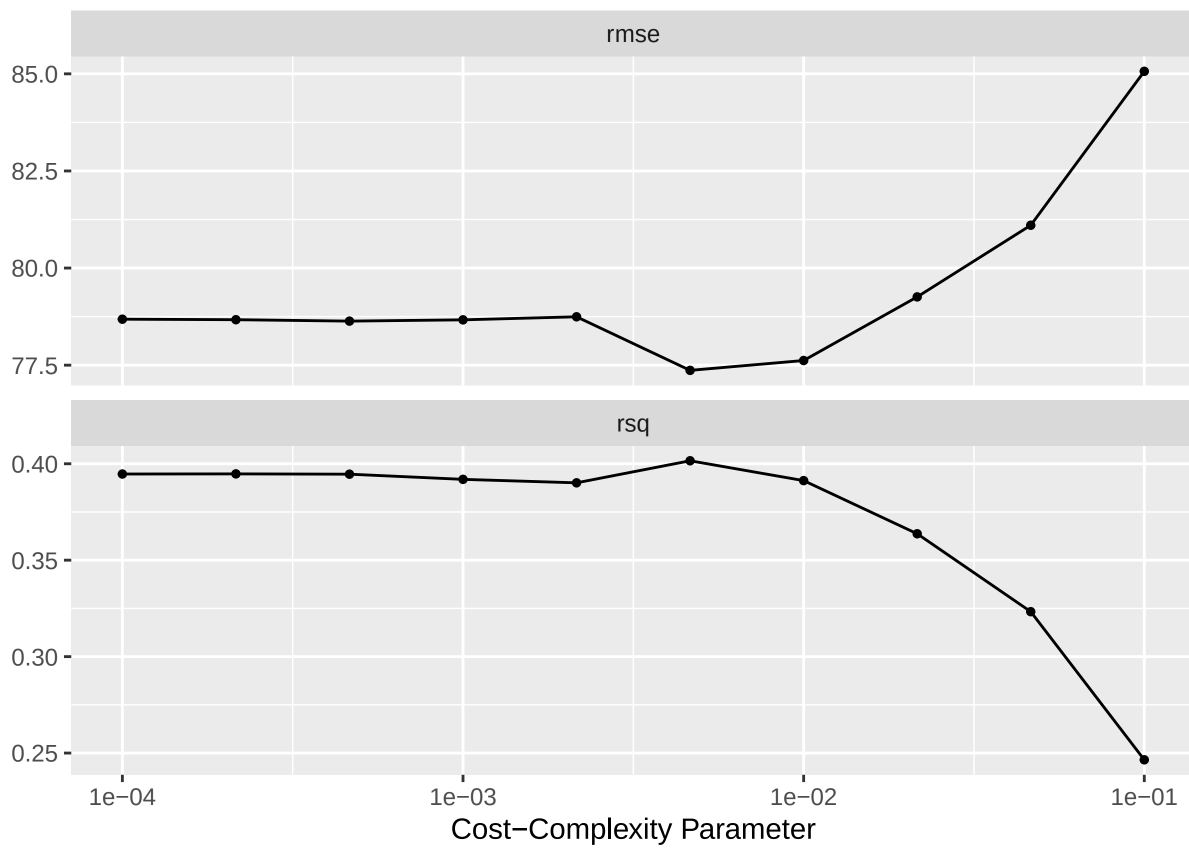
```
best_penalty <- select_best(ridge_tune, metric = "rmse")
best_penalty
```

penalty	.config
33.93222	Preprocessor1_Model33

3. Tree Decision Model

- For the Tree Decision Model, we use the same analysis procedure;
- We first use the autoplot() function to visulize the tuning result:

```
autoplot(tree_tune)
```



- The graph above shows that rmse reaches the lowest level when the cost-complexity parameter is in between 1e-03 and 1e-02;
- Then we use show_best() and notice that the best rmse of the Tree Decision model is very close to the lowest rmse of Ridge Regression model;

```
show_best(tree_tune)
```

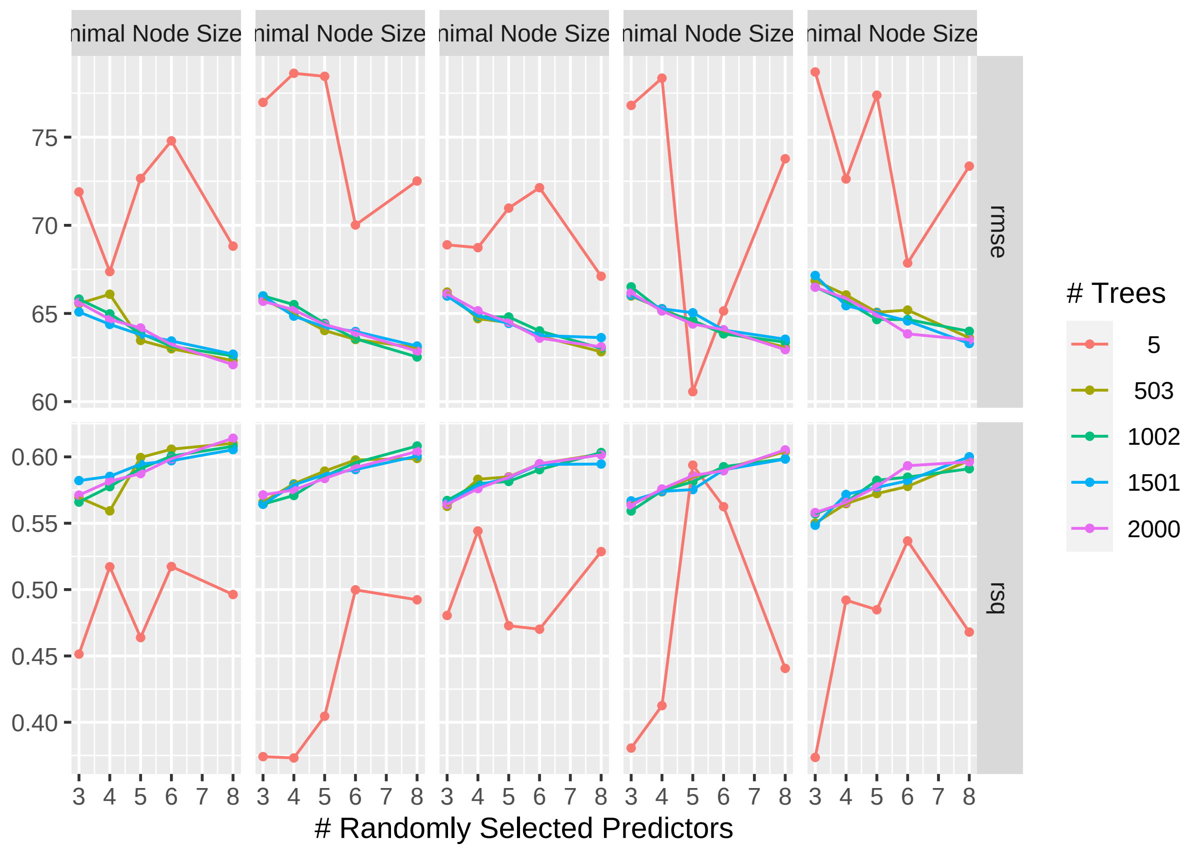
```
## Warning: No value of ‘metric‘ was given; metric ’rmse’ will be used.
```

cost_complexity	.metric	.estimator	mean	n	std_err	.config
0.0046416	rmse	standard	77.36626	5	11.38199	Preprocessor1_Model06
0.0100000	rmse	standard	77.61994	5	11.31526	Preprocessor1_Model07
0.0004642	rmse	standard	78.63397	5	11.89005	Preprocessor1_Model03
0.0010000	rmse	standard	78.66672	5	11.89160	Preprocessor1_Model04
0.0002154	rmse	standard	78.67046	5	11.89987	Preprocessor1_Model02

4. Random Forest Model

- We first visualize the tuning result of Random Forest Model:

```
autoplot(rf_tune)
```



- As it is shown above, when the tree number is only five, the results are highly variable, but when the number of trees gets really high, the graph shows that a larger node size results a smaller value of rmse;
- Then we select the specific best tuning model:

```
show_best(rf_tune)
```

```
## Warning: No value of ‘metric‘ was given; metric ’rmse’ will be used.
```

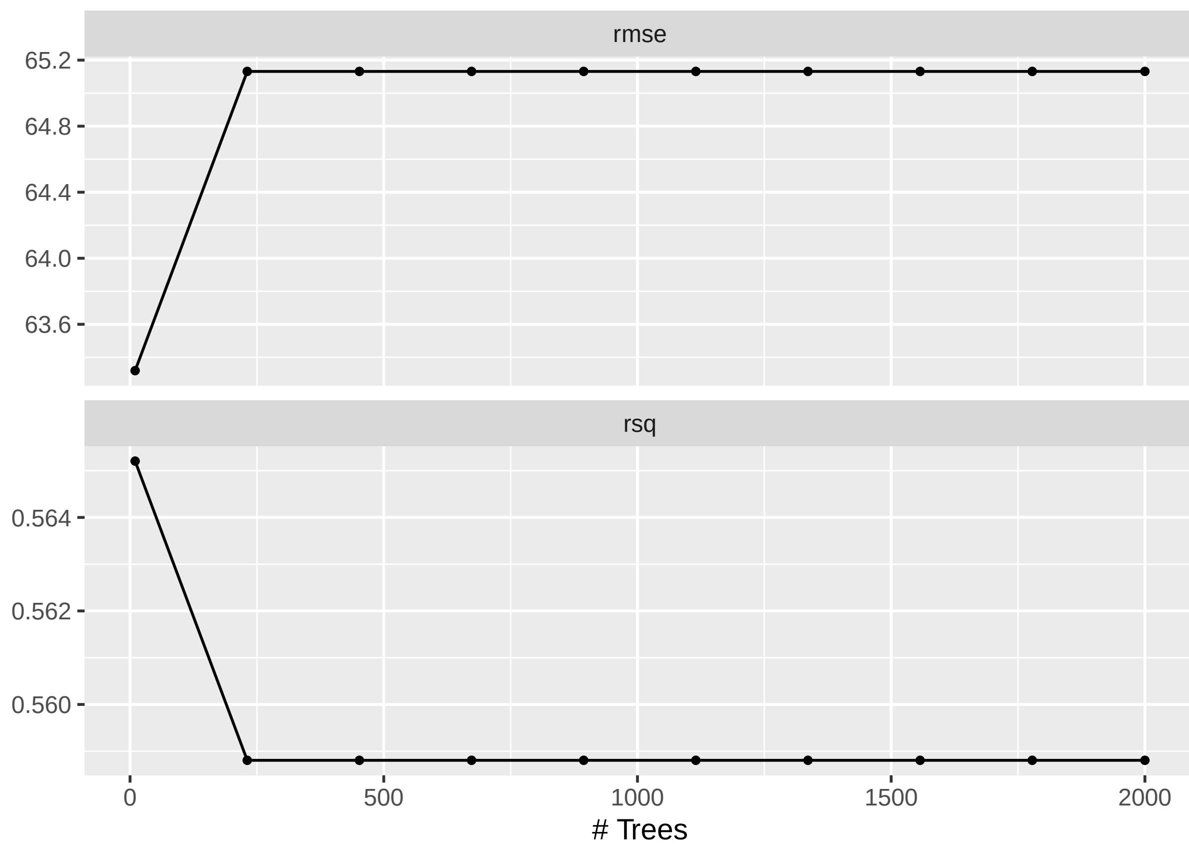
mtry	trees	min_n	.metric	.estimator	mean	n	std_err	.config
5	5	6	rmse	standard	60.56217	5	12.81039	Preprocessor1_Model078
8	2000	3	rmse	standard	62.09721	5	15.13394	Preprocessor1_Model025
8	503	3	rmse	standard	62.32769	5	14.98286	Preprocessor1_Model010
8	1002	4	rmse	standard	62.53059	5	15.09913	Preprocessor1_Model040
8	1002	3	rmse	standard	62.59266	5	15.19247	Preprocessor1_Model015

- It can be noticed above that the overall rmse is significantly smaller than three previous models, which indicates that random forest might be a better model;
- However, it also shows that the influence of the number of trees on the rmse is highly variable.

5. Boosted Tree Model

- Again, we first visualize the tuning result:
- what's interesting is that the graph below shows a contradicting result to the random forest model:
- it shows that when the smaller the number of trees is, the smaller the rmse is.

```
autoplot(boost_tune)
```



- We then select the best tuning model:

```
show_best(boost_tune)
```

```
## Warning: No value of `metric` was given; metric 'rmse' will be used.
```

trees	.metric	.estimator	mean	n	std_err	.config
10	rmse	standard	63.31936	5	15.31624	Preprocessor1_Model01
231	rmse	standard	65.13161	5	14.71023	Preprocessor1_Model02
452	rmse	standard	65.13161	5	14.71023	Preprocessor1_Model03
673	rmse	standard	65.13161	5	14.71023	Preprocessor1_Model04
894	rmse	standard	65.13161	5	14.71023	Preprocessor1_Model05

- Therefore, it shows that the rmse is lowest when the number of trees is 10, which is the lowest level as well;
- And it also contains a lower rmse than the first three models, while it is still higher than the rmse of the random forest model;

Final Model Selection and Testing

- According to the analysis above, we find that the Random forest model has the lowest average rmse level, and thus I decide to use the Random Forest model as the final model to do the testing.
- Thus, we first select the best Random Forest Model, which is the model when mtry = 5, trees = 5, min_n = 5;

```
best_model <- select_best(rf_tune, metric = "rmse")
best_model
```

mtry	trees	min_n	.config
5	5	6	Preprocessor1_Model078

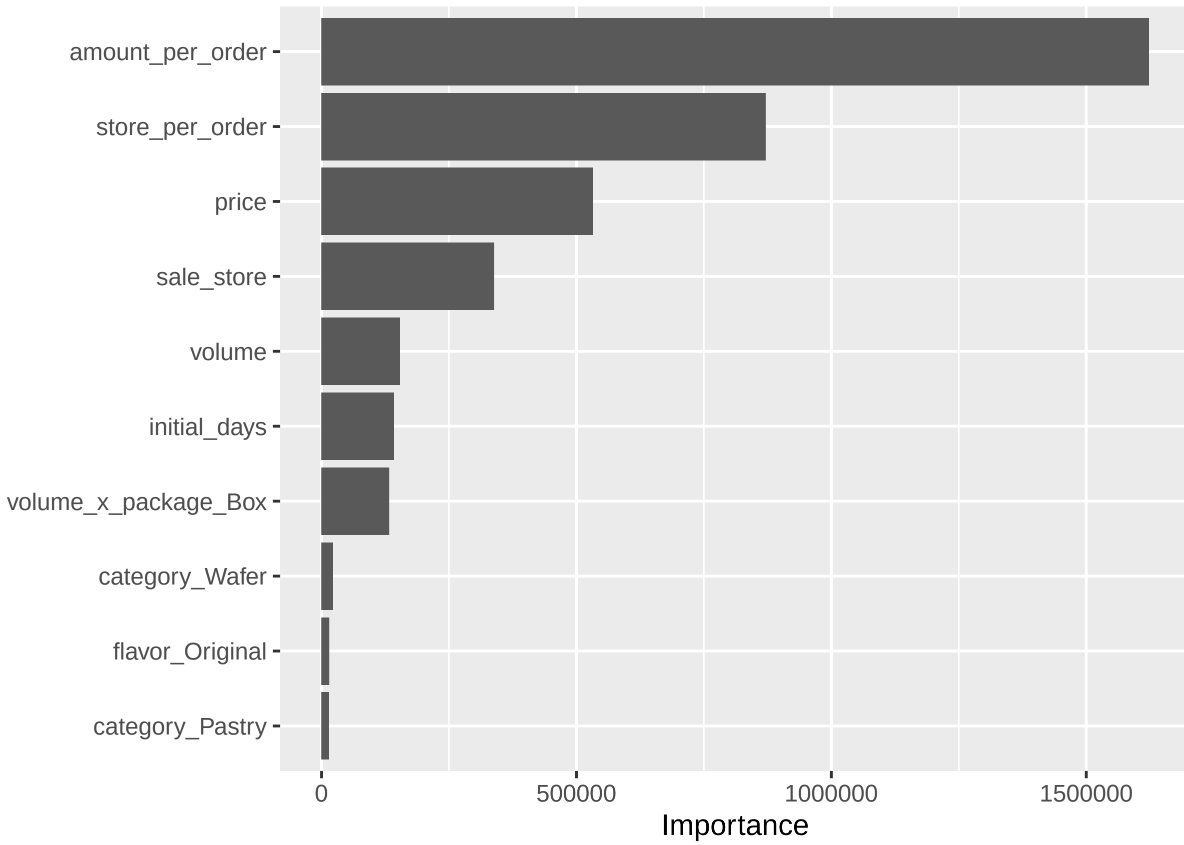
- Then, we fit the model with

```
final_rf <- finalize_workflow(rf_wf, best_model)
final_fit <- fit(final_rf, data = sales_train)

augmented_result <- augment(final_fit, new_data = sales_test)

augment(final_fit, new_data = sales_test) %>%
  rmse(truth = gross_profit, estimate = .pred)
```

.metric	.estimator	.estimate
rmse	standard	48.71332

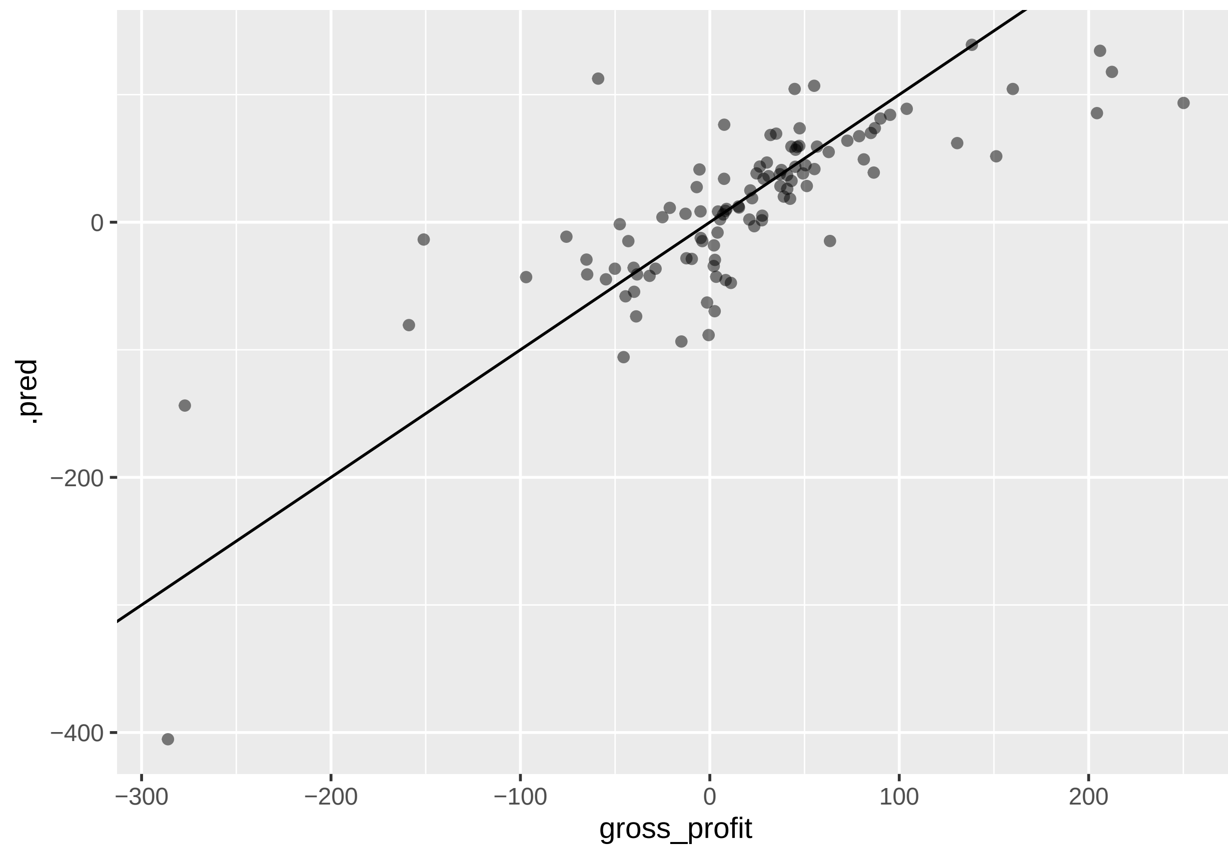
- As we can see, the tested fit of rmse is 62.58235, which is similar to the rmse level in the training set;
- We can also use the vip() function to evaluate the importance of each predictors in this model:

```r
final_fit %>%
  extract_fit_engine() %>%
  vip()
```



- Finally, we use ggplot() to visualize our prediction:

```r
augment(final_fit, new_data = sales_test) %>%
  ggplot(aes(gross_profit, .pred)) +
  geom_abline() +
  geom_point(alpha = 0.5)
```



- As it is shown in the graph above, this prediction looks better than the linear regression model, since the observed points are no longer highly concentrated.
- However, there are still many outliers that can reduce the accuracy of prediction

Conclusion

- In conclusion, this project strives to use five different models in order to predict the possible gross profits of the retail commodities about cakes.
- Among the five models, the Random Forest Model has the best performance and is thus selected to test.
- However, even though the Random Forest model does a relatively better job, its resulting rmse is still much higher than expected, which means that it is still not a good model to predict the gross profit.
- There are many possible explanations of this:

  1. It might be because that the training and testing data is highly insufficient, and thus the model cannot be trained well to make good predictions;
  2. It might be because that there are insufficient number of predictors in this model, which result in an inaccurate trained model;
  3. It is also worth noticing that two predictors (store_per_order & amount_per_order) were incorrectly selected, because these two predictors are originally calculated by the number of sold orders, which directly contribute the gross profit.

- Therefore, this model can definitely have more and better improvements in the future developments.