Stats 101C Final Regression Report

Group 8

By Tianlin Yue, Ziwei Guo, Rueiyu Chang, Kai Liang, Yiming Xue

July 28, 2024

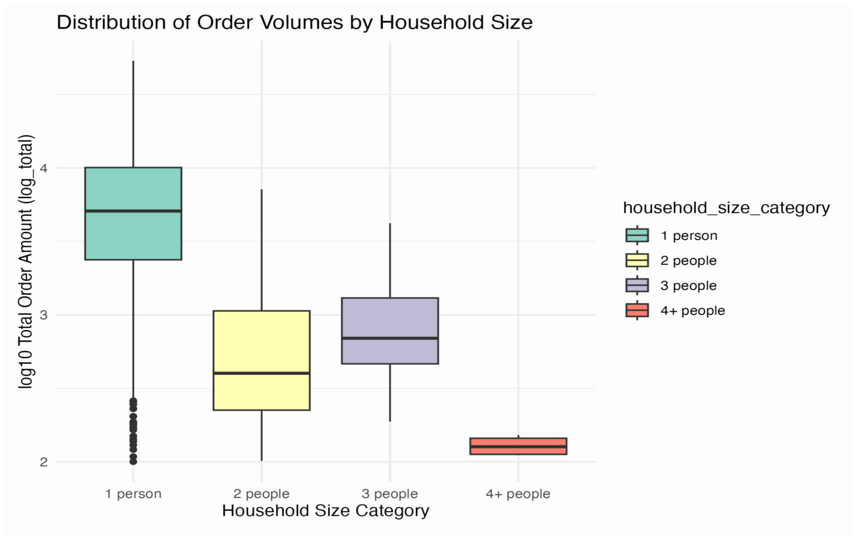
Table of Contents

1 Introduction

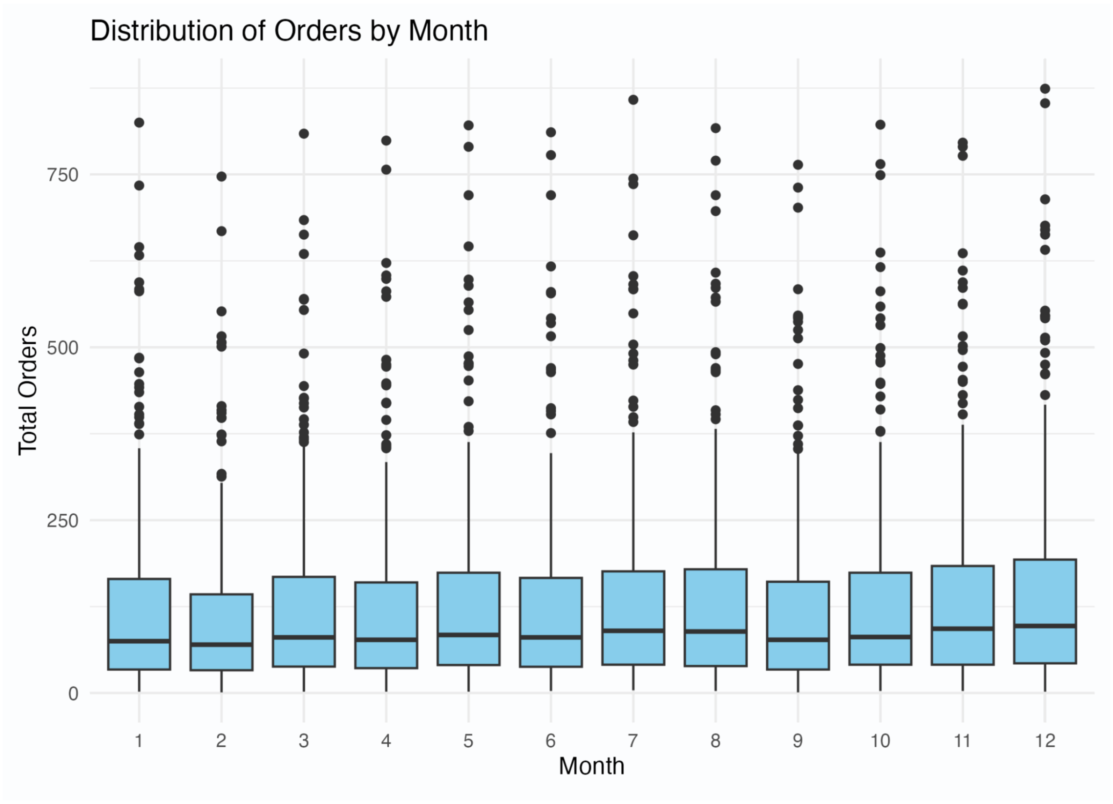
This project focuses on analyzing the dataset that is collected from a survey of approximately 5,000 Amazon customers between January 2018 and December 2022. The dataset is divided into training and test sets and each containing the customer information and order details. The primary goal is to predict the logarithm of total order amounts (log_total) by using the regression techniques. To achieve this, the analysis will consider the predictor variables including demographic (states), household size, account sharing, customer's age, income levels, and education level. Moreover, the research suggests that demographic and behavioral factors will significantly impact purchasing behavior (The Influence of Demographic Characteristics of Consumers on Decisions to Purchase Technical Products, 2018). In this report, it aims to identify the key predictors and develop the best predictive model through the regression modeling.

2 Data Analysis

2.1 Distribution

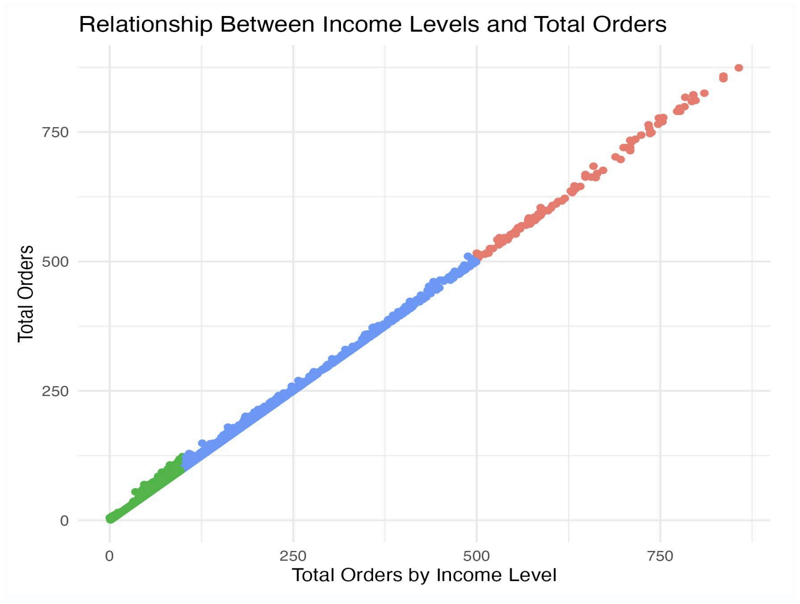


This box plot shows the distribution of total order amounts (log_total) for different household sizes. Households with 1 person have the highest median order amount, close to 3.5, while households with 4 or more people have the lowest median order amount, around 2. This indicates that smaller households tend to spend more on Amazon, possibly due to a higher propensity for online shopping. The presence of outliers suggests that some households have purchasing behaviors that differ from the majority, warranting further investigation.

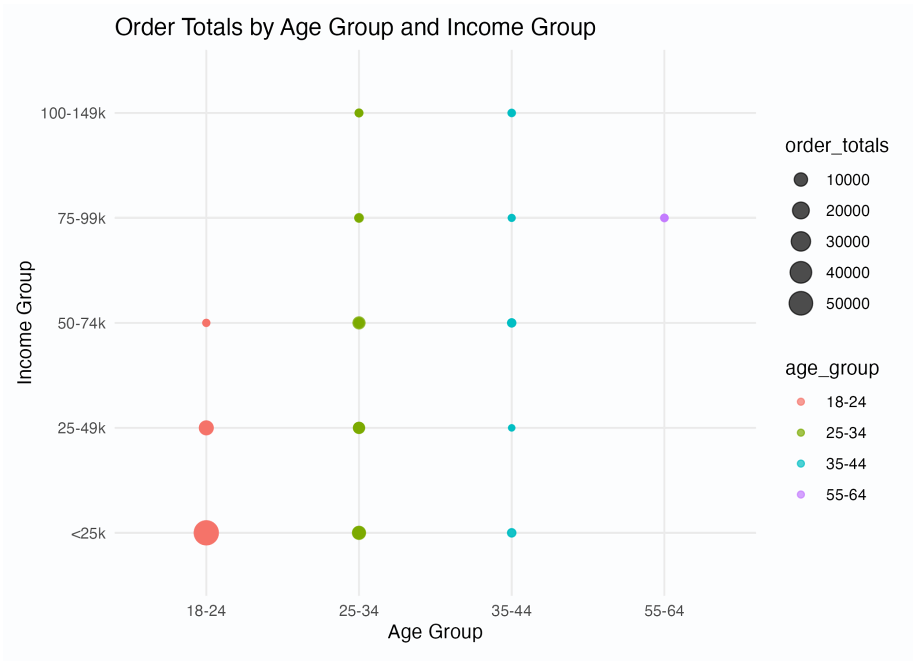


The box plot displays the distribution of total orders placed each month across all years in the dataset. Each box represents the interquartile range of orders for a given month, with outliers shown as dots. This plot highlights seasonal trends, showing that order volumes are relatively consistent throughout the year with slight increases in certain months, possibly due to holidays or promotional events.

2.2 Relationship

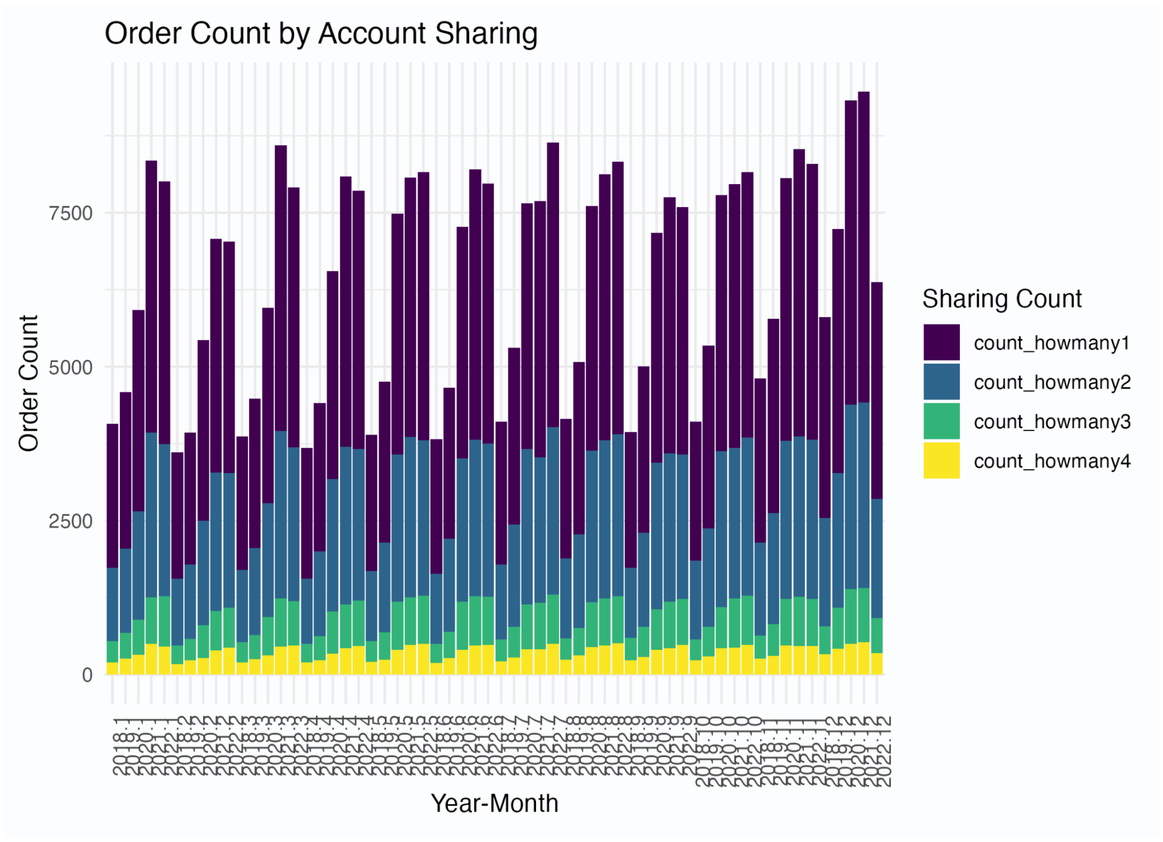


The scatter plot you've provided shows the relationship between the income and the total order amounts. The strong positive correlation observed indicates that as the income decreases (from green to red, high income to low income), the total order amount increases. This relationship suggests that income level is a strong predictor of total order amounts and should be prioritized in the model building process.

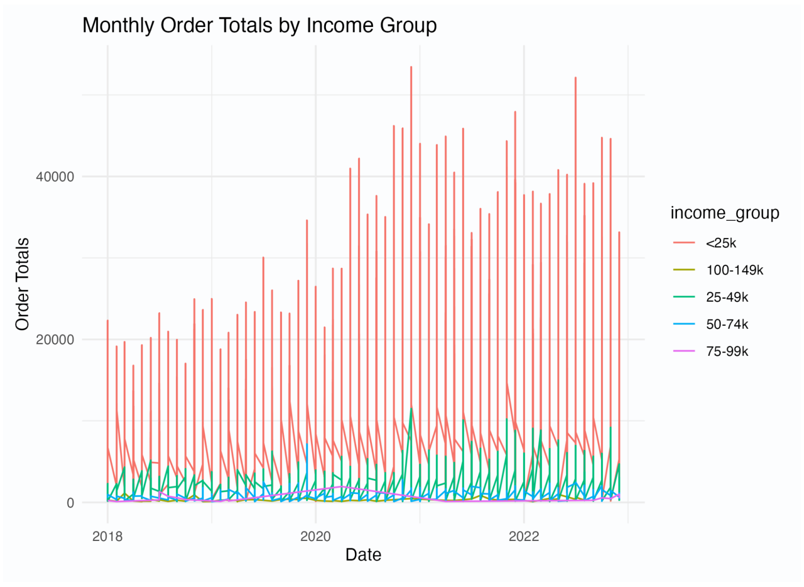


This chart visualizes the number of orders placed by Amazon customers, segmented by age group and income level. The x-axis represents different age groups, while the y-axis represents various income levels. The size and color of the circles indicate the volume of orders within each segment. Notably, the largest circle appears in the under $25k income bracket for the 18-24 age group, indicating this segment has the highest order count.

2.3 Time Series

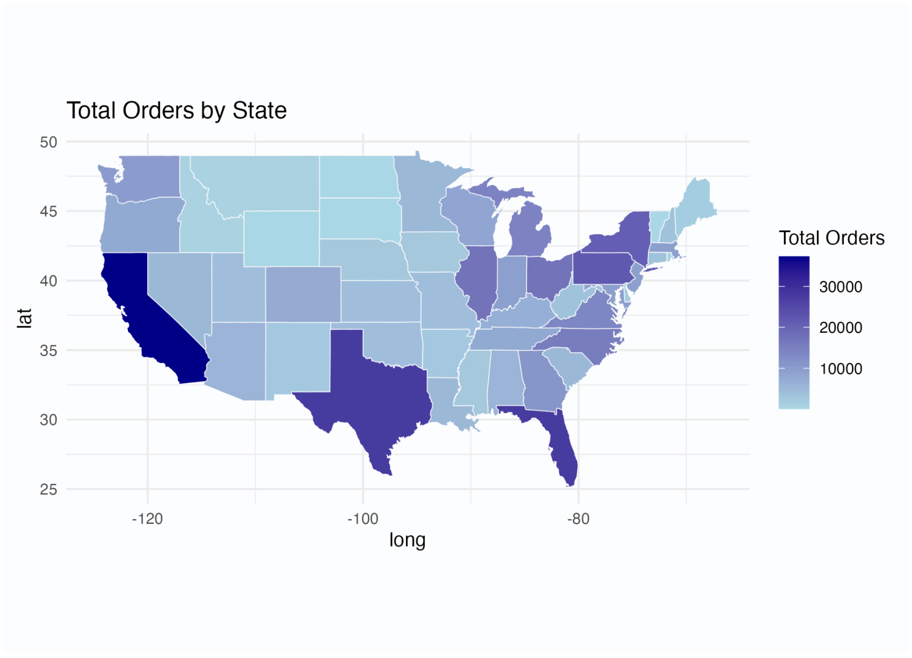


This stacked bar chart displays the total number of orders placed each month, categorized by the number of people in the household who use/share the Amazon account. Each color in the bars represents a different category The chart reveals consistent seasonal trends in order volumes, with peaks typically seen in November and December, likely due to holiday shopping. The varying heights of the colored segments show how the number of users per household contributes to the total order volume for each month.

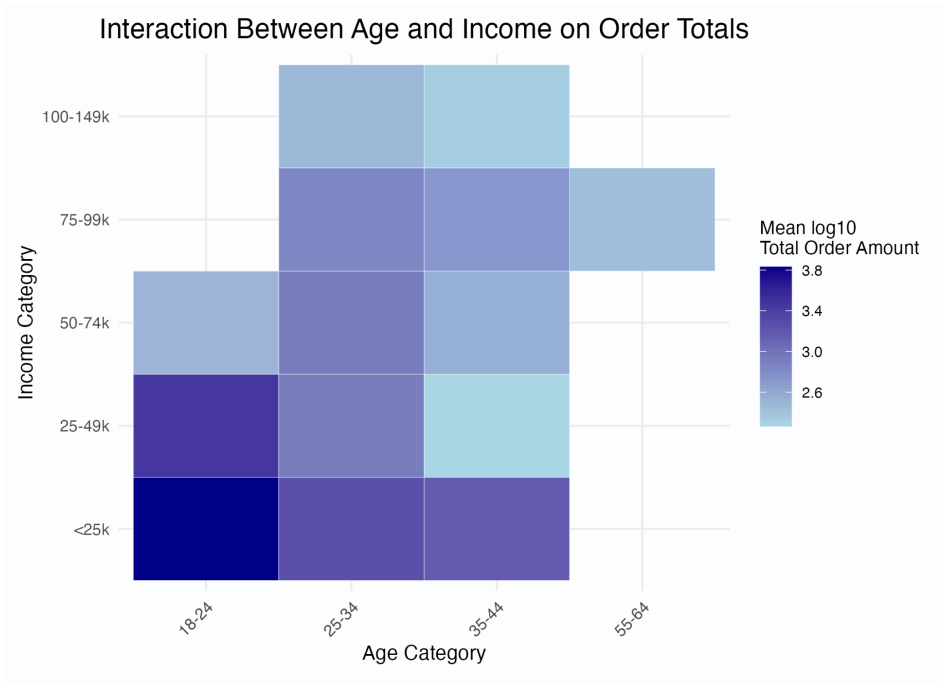


The chart shows the monthly order counts by Amazon customers from January 2018 to December 2022. The red line represents the low-income group, while other colored lines correspond to different income groups. The overall trend shows a significant increase in order counts over the years, with noticeable seasonal variations. It also fits the conclusion from the previous plot that the low income group has the highest order totals.

2.4 Map



This map illustrates the total number of orders placed in each state. Darker shades represent states with higher order totals. California, Texas, and New York show significantly higher order volumes compared to other states, indicating higher customer activity in these regions. This information is crucial for understanding regional differences in customer behavior.



This heat map displays the average order totals (log_total) across different age groups and income levels. Darker shades of blue indicate higher order totals. This chart helps us understand how different age groups and income levels interact to influence order totals. It reveals that younger individuals with lower incomes (<25k) tend to have higher order totals, while middle-aged and older individuals have more consistent but generally lower order totals across different income levels. This indicates that income and age are significant factors influencing purchasing behavior. The consistent patterns in older age groups suggest a more stable spending behavior, whereas younger age groups show more variation, particularly at lower income levels.

3 Preparation

3.1 Preprocessing/Recipes

First, we remove order_total as we target 'log_total' (log of order_total) as the response variable. Then, we transform the numeric values for year and month into factors for both the training and test datasets before we create recipes; additionally, we transform test id into factor so that it will not be log transformed or normalized. After these basic steps, we apply different recipes for various models.

3.1.1 Linear Model

For linear regression models, we start by transforming the character variable state into a factor, followed by log transformation of all numeric predictor variables. Since most of the variables are right-skewed, log transformation helps make them symmetric. To handle very small values close to zero, which can result in infinity after transformation, we convert these infinite values to NA and then impute them with the mean values of the corresponding variable. After these transformations, we normalize all numeric predictor variables. We also check the correlation of all variables and remove those with a correlation coefficient greater than 0.8.

For the linear regression model with interaction terms, we add the interaction terms of all numeric variables before removing closely related variables. It's worth noticing that we consider two-way interactions of numeric variables only.

In the recipe for regularized linear regression, we remove all categorical variables due to system warnings advising that all columns should be numeric. Additionally, because the regularization penalty already exists, we do not remove variables with high correlations.

3.1.2 Decision Tree and Random Forest Model

For decision tree and random forest models, we use the same recipe: first, we transform the state variable into a factor and then apply one-hot encoding to create dummy variables for categorical variables.

3.1.3 Boosting Tree Model with Xgboost and Gbm

For Boosting Tree Model with Xgboost, we first tried the preprocessing step and recipes that are similar previous models, where we transform year, month and id into factors, and we do step_dummy() for all nominal variables and step_corr() correlations higher than 0.8, with step_impute_mean for all numerical variables. However, as we found the abnormal result number for the test data, we decided to remove the step_corr() function in our recipe.

For the Boosting Tree Model with the gbm package (generalized boosted regression model package), we do not apply any recipe on it. Besides year and month, we convert state into factors.

4 Model Analysis

4.1 Candidate Models

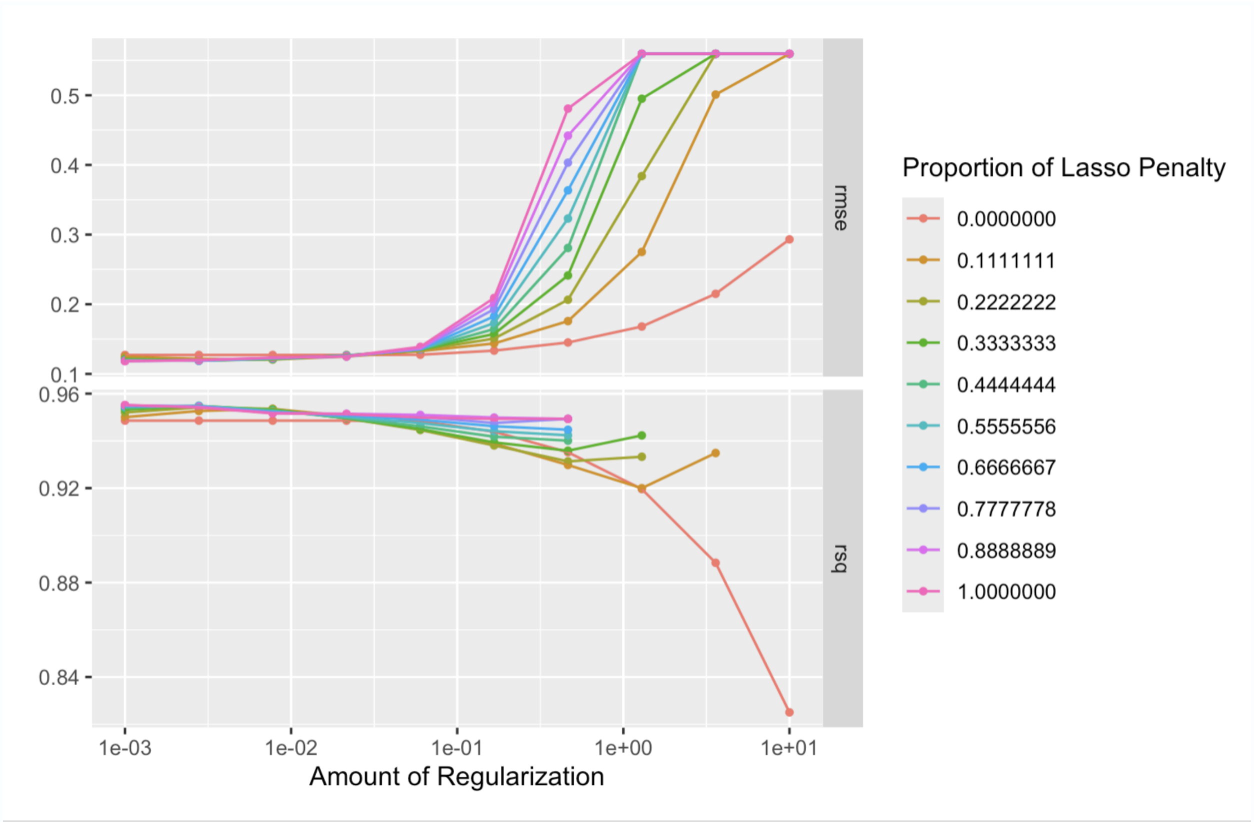
Throughout the process of reducing the root mean square error(RMSE) score of the data, we used multiple types of model, including linear regression model, decision tree model, random forest model, boosting tree model, and gbm model. For most models we tuned some parameters inside the model and tried to get a lowest score on RMSE, and the list of our settings is created in the table under this paragraph.

Model Iden	Model Type	Engine	Recipe	Hyperparameters
lm_model	Linear regression	lm	Included in Part 3.1.1	N/A
lm_model	Linear regression with interaction	lm	Included in Part 3.1.1	N/A
regularized_mod	Regularized linear regression	glmnet	Included in Part 3.1.1	penalty = 0.001 mixture = 1
dt_model	Decision Tree	rpart	Included in Part 3.1.2	Min_n = 2 Cost_complexity = 0.01
rf_model	Random Forest	ranger	Included in Part 3.1.2	min_n = 8 trees = 836
btree995-11-8-0.01672	Boosting Tree	xgboost	Included in Part 3.1.3	Tree numbers = 995, min_n = 11, Tree depth = 8, Learn_rate = 0.01672
btree837-11-8-0.01672	Boosting Tree	xgboost	Included in Part 3.1.3	Tree numbers = 837, Min_n = 20, tree_depth = 6, Learn_rate = 0.01672
btree670-14-5-0.01672	Boosting Tree	xgboost	Included in Part 3.1.3	Tree numbers = 670, Min_n = 14, tree_depth = 5, Learn_rate = 0.01672
boost	Gbm		N/A	n.trees = 1000, shrinkage = 0.04, n.minobsinnode = 28, distribution = "gaussian"

4.1.1 Linear regression Model

For simple linear regression model and linear regression model with interaction terms, we do not tune anything, the result doesn't vary too much as we use different values 0.8, 0.85, and 0.9 for correlation threshold. For regularized linear regression, we tune the 'penalty' and 'mixture' (the proportion of L1 regularizations) with range of penalty from -3 to 1 with log transformation of base 10 and mixture from 0 to 1 of 10 levels

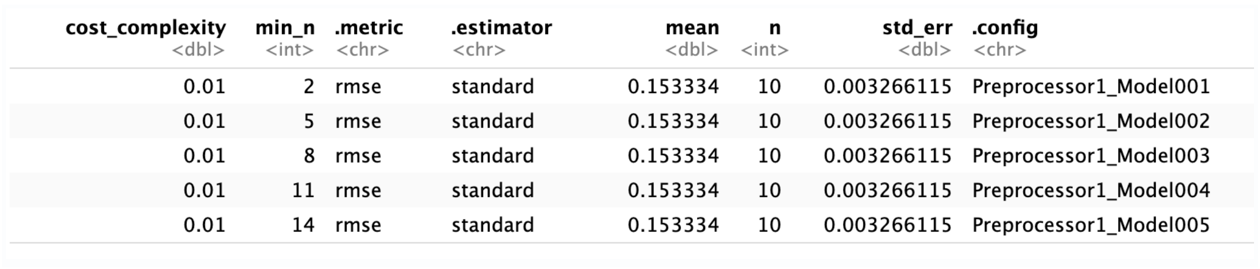
penalty <dbl>	mixture <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
0.001000000	1.0000000	rmse	standard	0.1183729	10	0.003419220	Preprocessor1_Model091
0.001000000	0.8888889	rmse	standard	0.1184931	10	0.003448392	Preprocessor1_Model081
0.001000000	0.7777778	rmse	standard	0.1186873	10	0.003453416	Preprocessor1_Model071
0.002782559	0.4444444	rmse	standard	0.1189252	10	0.003336970	Preprocessor1_Model042
0.001000000	0.6666667	rmse	standard	0.1189966	10	0.003441792	Preprocessor1_Model061



Finally, we choose 0.001 as penalty and 1 as mixture.

4.1.2 Decision Tree model

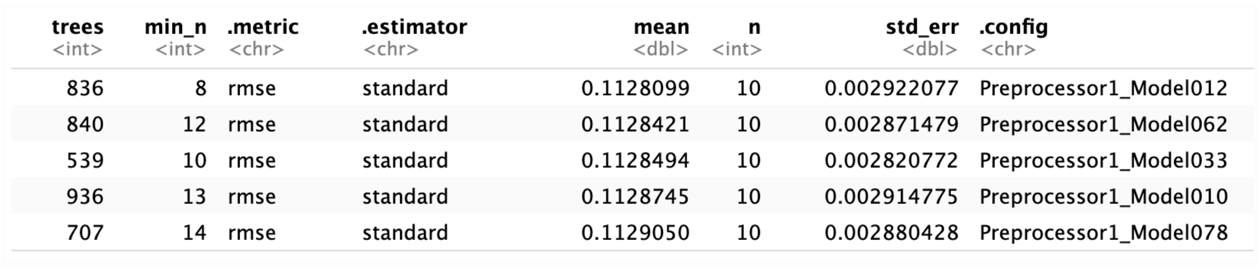
For the decision tree, we tune the hyperparameters ‘min_n’ and the ‘cost_complexity’ with the range 2 to 30 for minimum nodes and -2 to -0.5 with log10 transformation for cost complexity. Based on the result, we set them min_n = 2 and cost_complexity = 0.01 respectively.



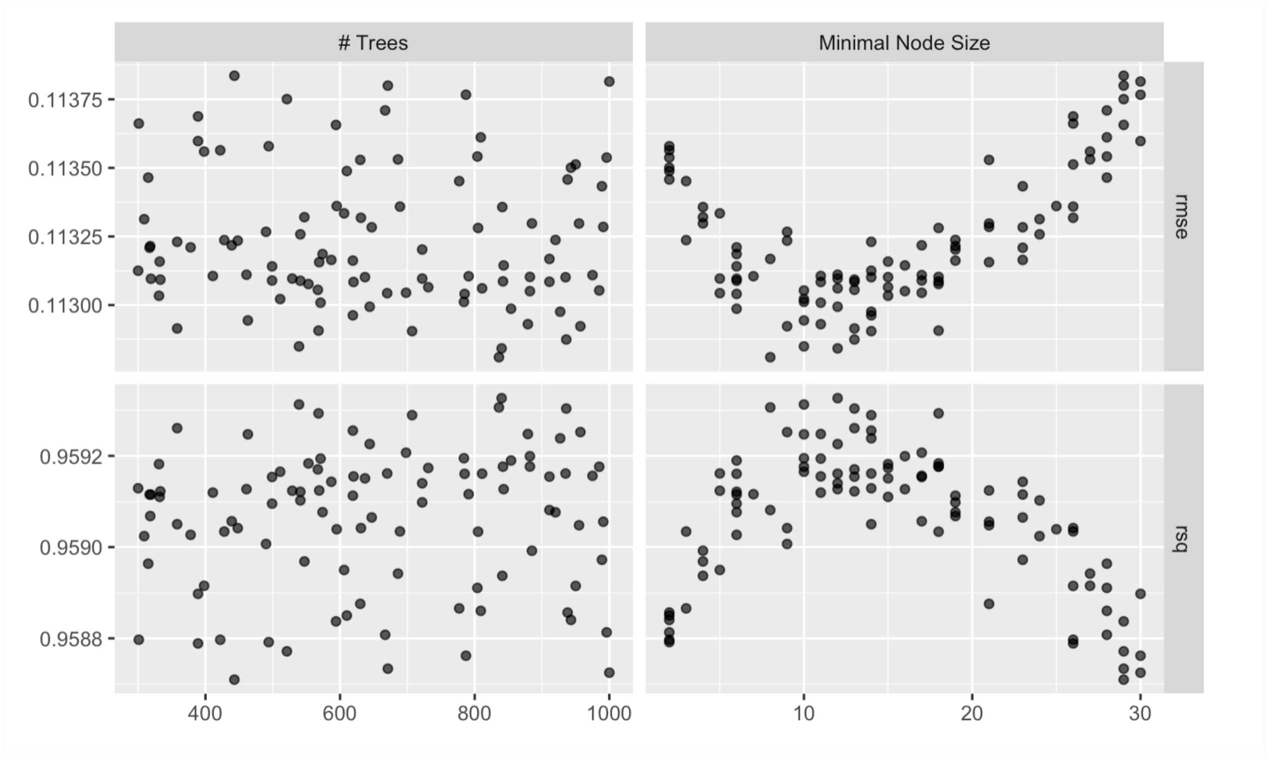
cost_complexity	min_n	.metric	.estimator	mean	n	std_err	.config
<dbl>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
0.01	2	rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model001
0.01	5	rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model002
0.01	8	rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model003
0.01	11	rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model004
0.01	14	rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model005

4.1.3 Random Forest model

For the random forest, we tune the hyperparameters ‘min_n’ and the ‘trees’ with the range 2 to 30 for minimum nodes and 300 to 1000 for trees. Based on the result, we set them min_n = 8 and trees = 836 separately.



trees	min_n	.metric	.estimator	mean	n	std_err	.config
<int>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
836	8	rmse	standard	0.1128099	10	0.002922077	Preprocessor1_Model012
840	12	rmse	standard	0.1128421	10	0.002871479	Preprocessor1_Model062
539	10	rmse	standard	0.1128494	10	0.002820772	Preprocessor1_Model033
936	13	rmse	standard	0.1128745	10	0.002914775	Preprocessor1_Model010
707	14	rmse	standard	0.1129050	10	0.002880428	Preprocessor1_Model078

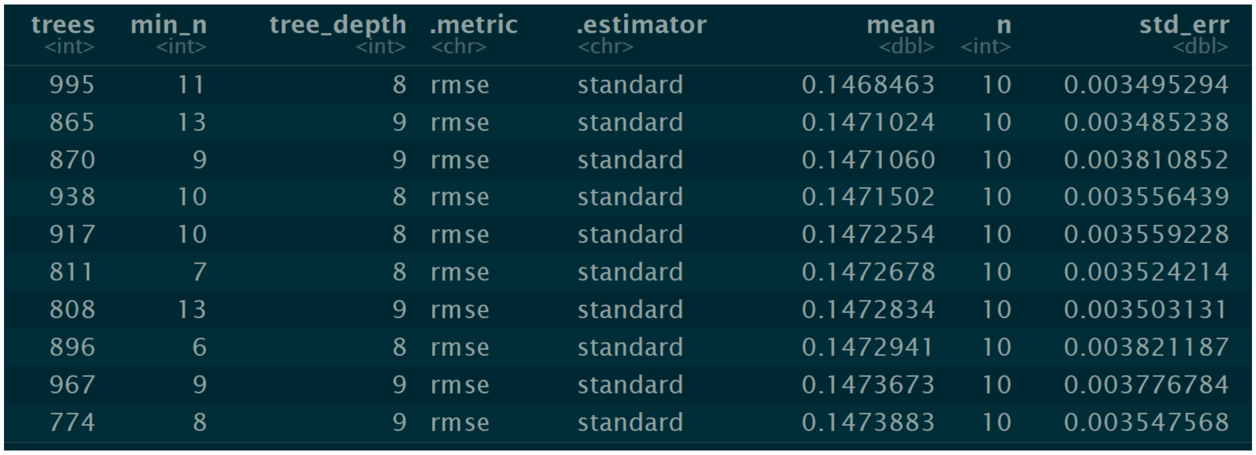


4.1.4 Boosting Tree Model

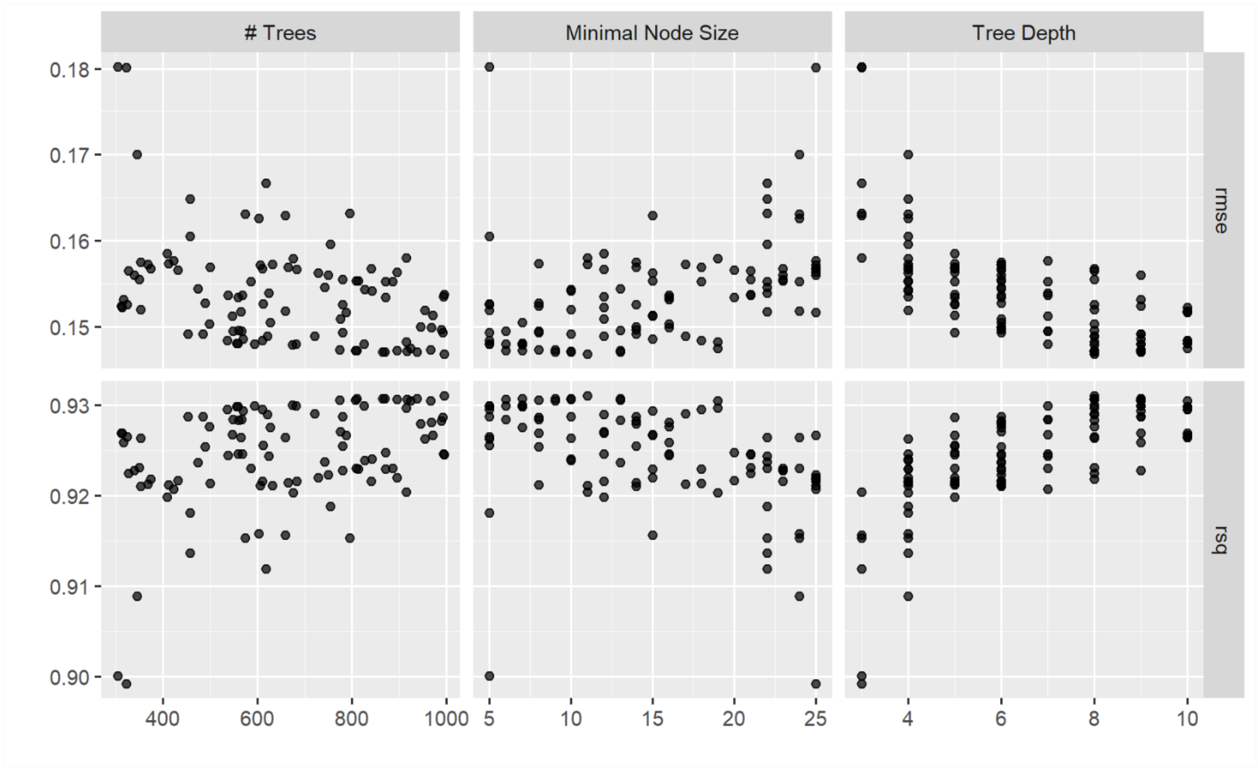
In the Boosting Tree Model, we mainly focused on the different versions of recipe and tuning hyperparameters. Though we know that this model should probably be the one of models that give the best test result, the actual output of testing RMSE is not as good as we thought. The training RMSE is very low after our tuning method, but when we apply the model to the test data, the RMSE is actually higher than what we expect. This suggests that there are some kinds of overfitting in our boosting tree model.

The Initial Boosting Tree Model

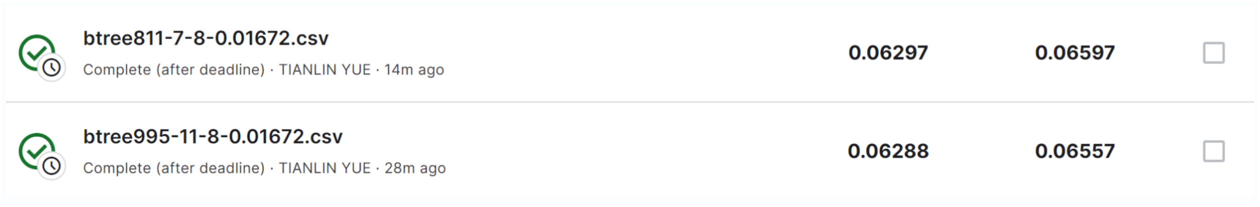
We first tuned the 'trees', 'min_n' and 'tree_depth' parameters in the model, and we found the result and an autoplot listed under.



trees	min_n	tree_depth	.metric	.estimator	mean	n	std_err
<int>	<int>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>
995	11	8	rmse	standard	0.1468463	10	0.003495294
865	13	9	rmse	standard	0.1471024	10	0.003485238
870	9	9	rmse	standard	0.1471060	10	0.003810852
938	10	8	rmse	standard	0.1471502	10	0.003556439
917	10	8	rmse	standard	0.1472254	10	0.003559228
811	7	8	rmse	standard	0.1472678	10	0.003524214
808	13	9	rmse	standard	0.1472834	10	0.003503131
896	6	8	rmse	standard	0.1472941	10	0.003821187
967	9	9	rmse	standard	0.1473673	10	0.003776784
774	8	9	rmse	standard	0.1473883	10	0.003547568



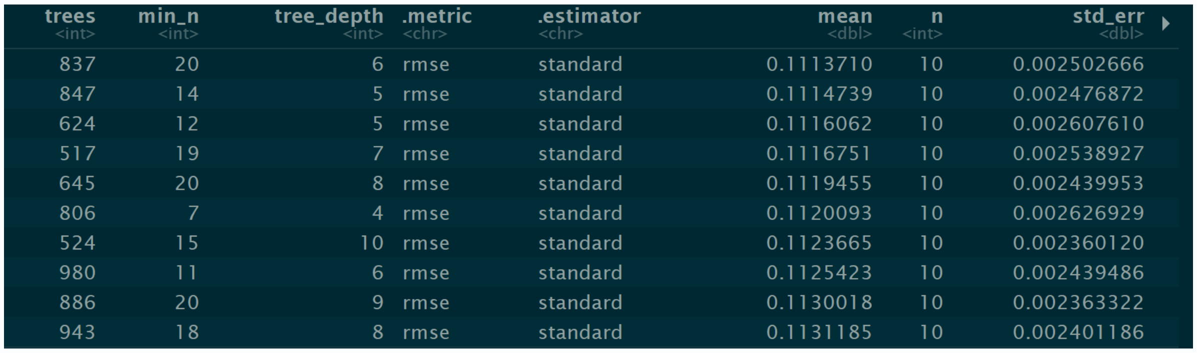
As we can see, tree number of 911, min_n of 11 and tree depth of 8 gives the best result, so we plug those numbers in our actual model for the test set and get our prediction. However, the actual score for both the public test set and private test score are all showing signs of mistakes, no matter which set of combinations we choose for the test set.



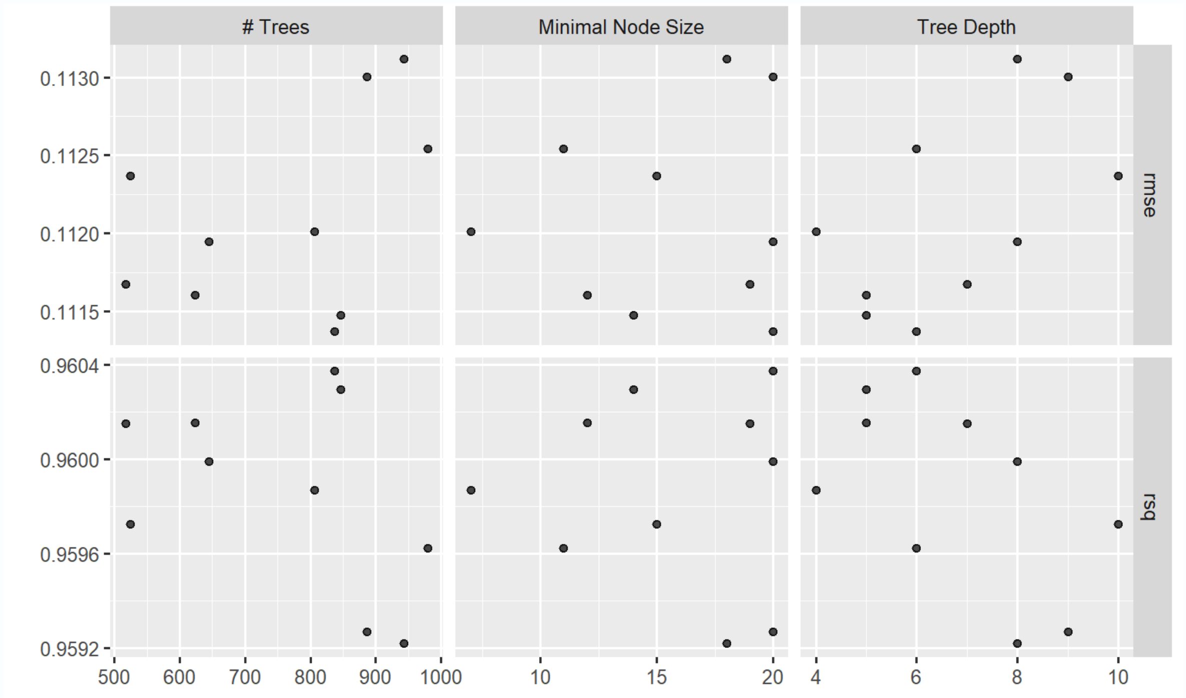
btree811-7-8-0.01672.csv Complete (after deadline) · TIANLIN YUE · 14m ago	0.06297	0.06597	☐
btree995-11-8-0.01672.csv Complete (after deadline) · TIANLIN YUE · 28m ago	0.06288	0.06557	☐

Model after Changing Recipe

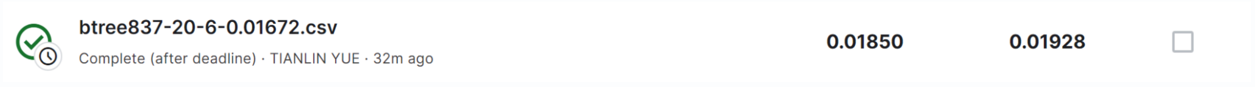
Because of the abnormal scores, we decided to alter our model condition and re-perform the tuning process. After some testing of the different settings, we speculate that the main problem happens on the step_corr functions in our recipe, therefore we decided to remove this step in the recipe and re-run the tuning process with the same conditions except smaller grid size. The result is:



trees <int>	min_n <int>	tree_depth <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
837	20	6	rmse	standard	0.1113710	10	0.002502666
847	14	5	rmse	standard	0.1114739	10	0.002476872
624	12	5	rmse	standard	0.1116062	10	0.002607610
517	19	7	rmse	standard	0.1116751	10	0.002538927
645	20	8	rmse	standard	0.1119455	10	0.002439953
806	7	4	rmse	standard	0.1120093	10	0.002626929
524	15	10	rmse	standard	0.1123665	10	0.002360120
980	11	6	rmse	standard	0.1125423	10	0.002439486
886	20	9	rmse	standard	0.1130018	10	0.002363322
943	18	8	rmse	standard	0.1131185	10	0.002401186



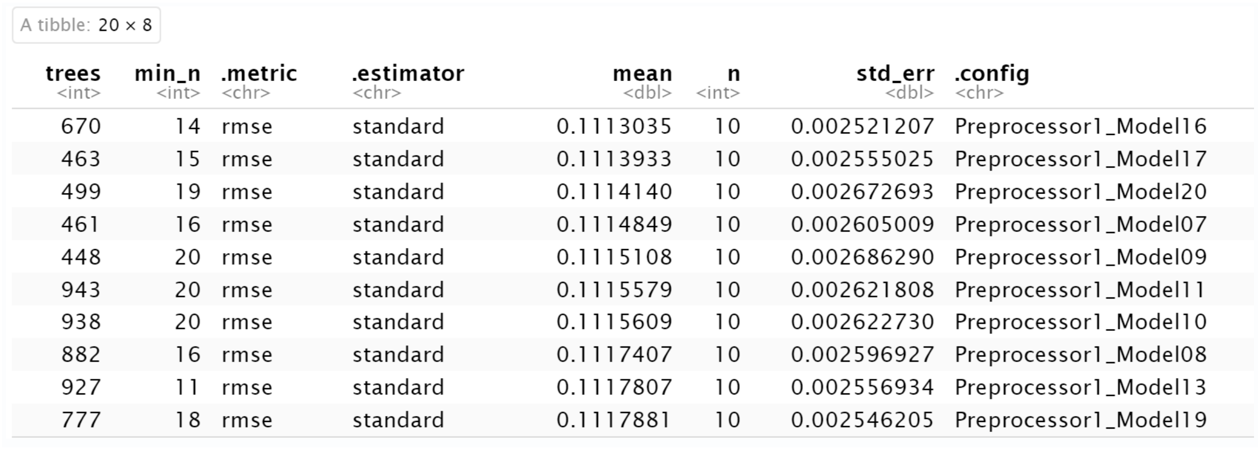
Plugging 837, 20 and 6 for trees, min_n, and tree_depth, we gained a result that looks much more normal, but still not high enough.



btree837-20-6-0.01672.csv
Complete (after deadline) · TIANLIN YUE · 32m ago
0.01850 0.01928

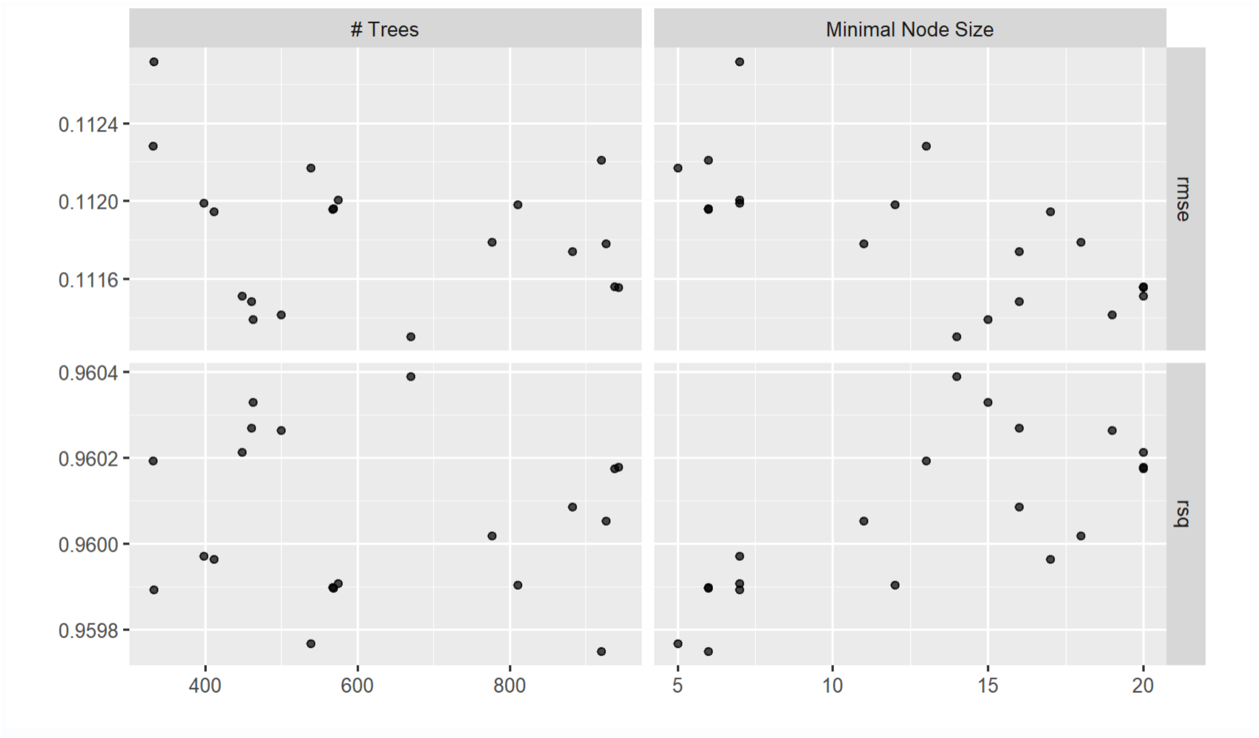
Boosting Tree with Further Improvement

Since the last result is still lower than we expected, we decided to try to further improve the score by finding a better min_n and tree value. Since the top 3 tree_depth are 5 and 6, we decide to limit our tree depth to 5 to test the other 2 parameters.

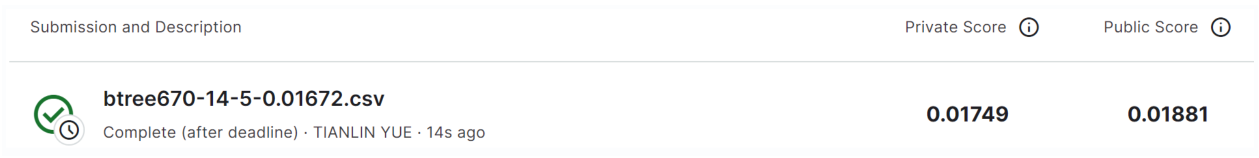


A tibble: 20 × 8

trees <int>	min_n <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
670	14	rmse	standard	0.1113035	10	0.002521207	Preprocessor1_Model16
463	15	rmse	standard	0.1113933	10	0.002555025	Preprocessor1_Model17
499	19	rmse	standard	0.1114140	10	0.002672693	Preprocessor1_Model20
461	16	rmse	standard	0.1114849	10	0.002605009	Preprocessor1_Model07
448	20	rmse	standard	0.1115108	10	0.002686290	Preprocessor1_Model09
943	20	rmse	standard	0.1115579	10	0.002621808	Preprocessor1_Model11
938	20	rmse	standard	0.1115609	10	0.002622730	Preprocessor1_Model10
882	16	rmse	standard	0.1117407	10	0.002596927	Preprocessor1_Model08
927	11	rmse	standard	0.1117807	10	0.002556934	Preprocessor1_Model13
777	18	rmse	standard	0.1117881	10	0.002546205	Preprocessor1_Model19



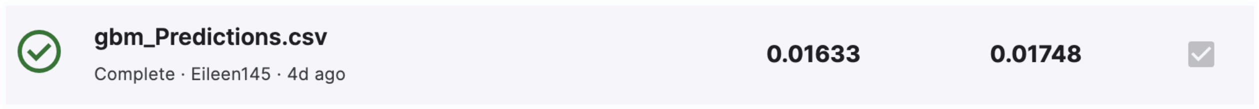
As we can see from both graphics, the result has been improved by a significant amount, but it is still not as good as we thought. As a few of the other tests all cannot give a result that is much better than what we previously got, we decide to transform to boosting tree with gbm.



Submission and Description	Private Score	Public Score
btree670-14-5-0.01672.csv Complete (after deadline) · TIANLIN YUE · 14s ago	0.01749	0.01881

Boosting Tree with gbm()

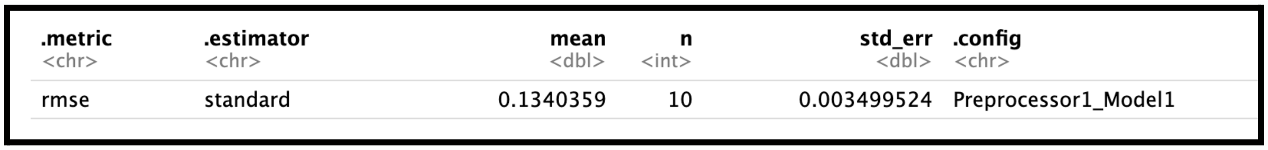
For this model, we remind the preprocessing of transferring numerical variables year, month into factor for training and testing datasets, then we use the generalized boosting tree from packages “gbm”. (With another seed for set.seed()) We tune the boosting tree model with hyperparameters of trees, minimum elements in terminal nodes, stop iteration, maximum tree depth, and learning rate (trees = 1000, min_n = 28, stop_iter = 5, tree_depth = 8, learn_rate = 0.04), so I set n.trees = 1000, shrinkage = 0.04, n.minobsinnode = 28, distribution = "gaussian". Because we don’t use resample() and vfold_cv() for cross validation here, we manually write some code with a similar function of cross validation. We repeatedly split the training data into training and validation subsets in an 8:2 ratio for 10 iterations, fit the boosting model on the training subset, and predict on the validation subset. The RMSE for each iteration is calculated and stored, and the mean RMSE across all iterations is computed to assess the model’s performance.



```
gbm_Predictions.csv                                   0.01633        0.01748
Complete · Eileen145 · 4d ago
```

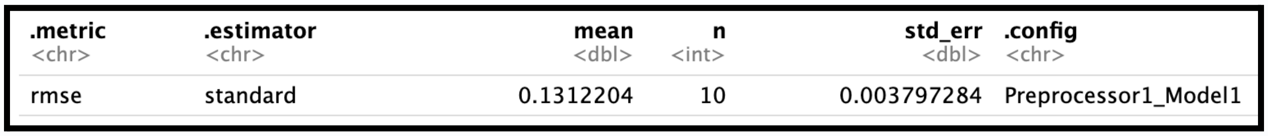
4.2 Model Evaluation

1. Linear regression



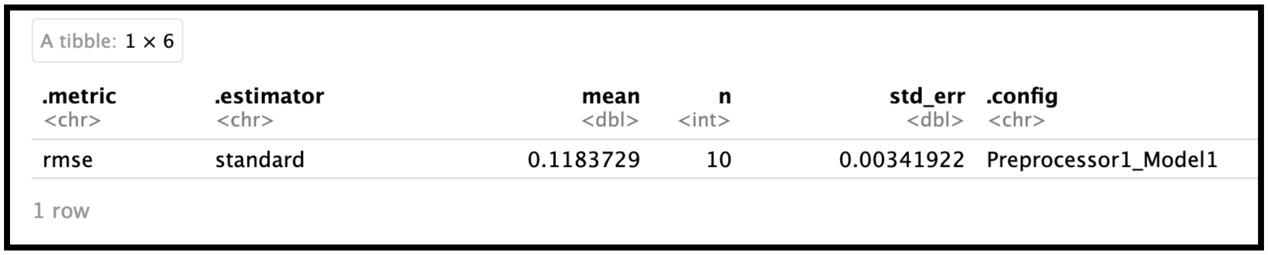
```
.metric    .estimator          mean      n        std_err  .config
<chr>      <chr>              <dbl>  <int>          <dbl>  <chr>
rmse       standard       0.1340359     10    0.003499524  Preprocessor1_Model1
```

2. Linear regression with interaction



```
.metric    .estimator          mean      n        std_err  .config
<chr>      <chr>              <dbl>  <int>          <dbl>  <chr>
rmse       standard       0.1312204     10    0.003797284  Preprocessor1_Model1
```

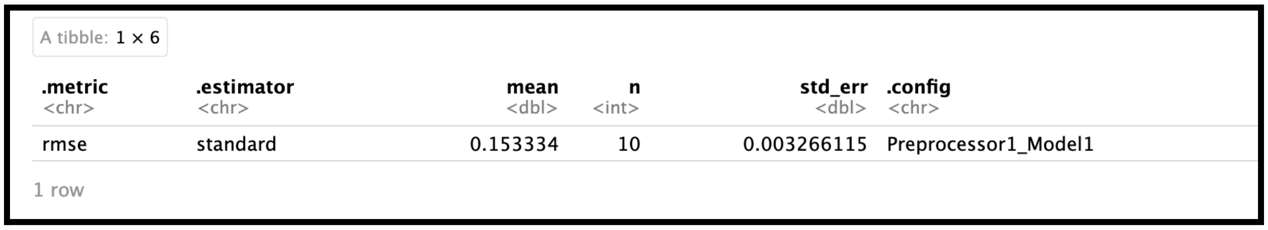
3. Regularized Linear Regression


A tibble: 1 × 6

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.1183729	10	0.00341922	Preprocessor1_Model1

1 row

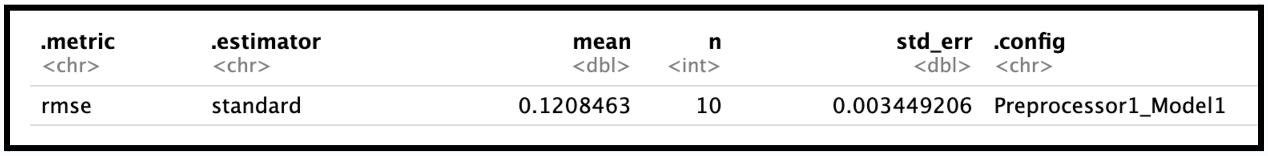
4. Decision tree


A tibble: 1 × 6

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.153334	10	0.003266115	Preprocessor1_Model1

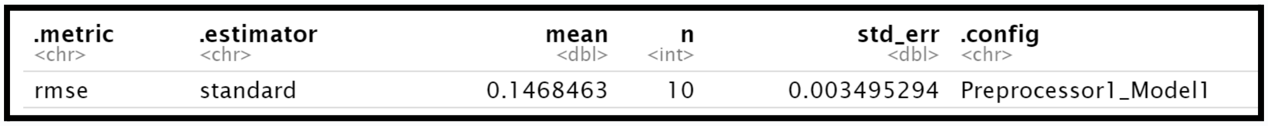
1 row

5. Random Forest



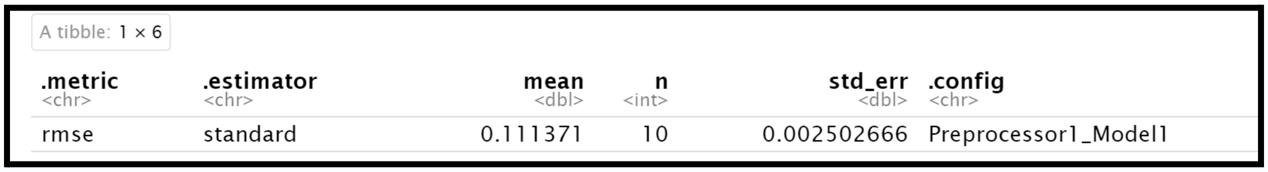
.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.1208463	10	0.003449206	Preprocessor1_Model1

6. Initial Boosting Tree



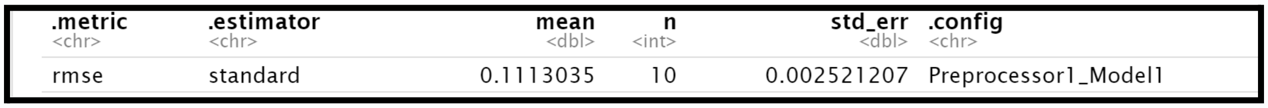
.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.1468463	10	0.003495294	Preprocessor1_Model1

7. Boosting Tree without Step_corr


A tibble: 1 × 6

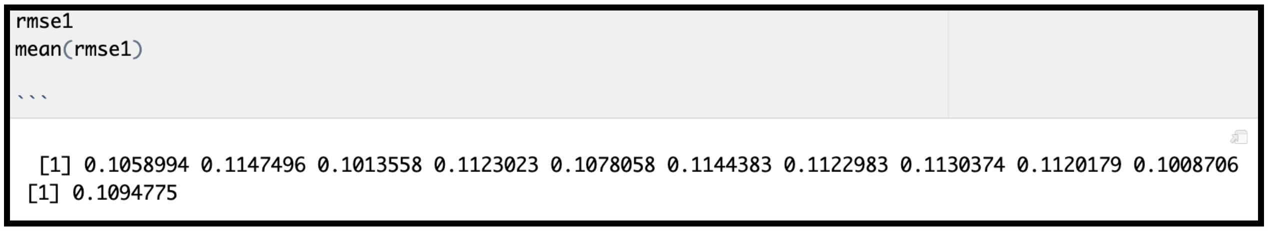
.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.111371	10	0.002502666	Preprocessor1_Model1

8. Boosting Tree with Tree_depth = 5



.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
rmse	standard	0.1113035	10	0.002521207	Preprocessor1_Model1

8. Boosting Tree with gbm()



```
rmse1
mean(rmse1)

```
```

```
 [1] 0.1058994 0.1147496 0.1013558 0.1123023 0.1078058 0.1144383 0.1122983 0.1130374 0.1120179 0.1008706
[1] 0.1094775
```

5 Discussion

5.1 Selection of the final model

In this project, we used the GBM model to predict the logarithm of the total orders (log_total). GBM model is a powerful ensemble learning technique that builds models iteratively, where each new model corrects the errors of the previous ones. We trained the model using the training dataset and tuned the hyperparameters such as the number of trees (n.trees), learning rate (shrinkage), and minimum number of observations in the terminal nodes (n.minobsinnode).

The final model was selected based on the RMSE obtained from cross-validation. We performed a grid search over a range of random seeds to ensure that our results were robust to random initialization. The best model, which yielded the lowest RMSE on the validation set, was chosen for predicting the test set.

5.2 Strengths and Weaknesses

GBM model has the advantages of Robustness, Accuracy and Flexibility. GBM model is less prone to overfitting compared to other methods like decision trees because it builds multiple trees in a sequential manner. The ensemble approach of GBM model often results in higher accuracy than individual models by combining the strengths of many weak learners. What's more, GBM model can handle different types of data (categorical, numerical) and missing values effectively.

As for the disadvantages of the GBM model, training GBM model can be slow and resource-intensive, especially with large datasets and a high number of trees. Gbm
```

model requires careful tuning of multiple hyperparameters, which can be complex and time-consuming. GBM requires careful tuning of multiple hyperparameters, which can be complex and time-consuming.

5.3 Possible Improvements

The first improvement would be to convert state into factors in test data. This is an obvious mistake we made. We convert the variable q_demos_state in training data, but we fail to do so in the test data, which leads to inconsistent results.

Then, we focus the improvement of the model mainly into two parts: data processing and model optimization. For the data, we can create new features from the existing data, such as polynomial features, could help the model capture more complex relationships. The usage of other two training data sets with additional information is currently at the minimum level, and we could find ways to transform those datasets and include them in our model to increase accuracy of prediction. We can also pay more attention to the temporal characteristics of the data. Including more granular temporal features, such as day of the week or holiday indicators, could improve the model's ability to capture seasonal patterns.

There are also methods we can use to optimize our model to make its predictions closer. For example, Hyperparameter Optimization of the model. Using techniques such as Bayesian optimization for hyperparameter tuning might lead to better performance than the grid search. In addition, we can implement regularization techniques that principal component analysis (PCA) could help in reducing overfitting and improving the model's generalizability.

Overall, we chose the GBM model because it is more accurate than other predictions. The accuracy of prediction is related to advantages such as the ensemble approach of the model itself. While the GBM model provides a robust and accurate model for this task, there are still ways we might make it perform better.

6 Conclusion

In this report, we aimed to predict the 'log_total' by using the dataset from the survey of approximately 5,000 Amazon customers over 5 years. Throughout the data analysis and model testing, we approach different regression techniques to identify the most effective predictive model.

We explored a variety of models, including linear regression, regularized linear regression, decision trees, random forests, and boosting tree models. And each model evaluation is based on the RMSE to ensure the highest accuracy for our predictions. We chose the GBM model as our final choice because it demonstrated the best performance and showed the lowest RMSE during the cross-validation. In addition, it corrects errors from the previous iterations and contributes significantly to its predictive accuracy.

Overall, the GBM model is the most effective tool for predicting the total order amounts in our dataset. The model's selection was based on its performance metrics and its ability of handling the different data types efficiently. Which highlights the advanced ensemble learning techniques of predictive modeling and provides the foundation for the future improvements.

7 Appendix

7.1 Final annotated script

```
rm(list = ls())
gc()

library(tidyverse)
library(tidymodels)
library(ggplot2)

# Load the training dataset
```

```r
train <- read_csv("train.csv")
train <- train %>% select(-order_totals)
# Convert year, month, state into factors
train$year <- factor(train$year)
train$month <- as.factor(train$month)
train$q_demos_state <- as.factor(train$q_demos_state)

# Load the test dataset
test <- read_csv("test.csv")
# Convert year, month, and id into factors
test$year <- as.factor(test$year)
test$month <- as.factor(test$month)
test$id <- as.factor(test$id)

#install.packages("gbm")
library(gbm)
set.seed(55)

rmse1 <- rep(NA, 10)

# Perform cross-validation
for (i in 1:10){
  data_split <- initial_split(train, prop = 0.8)
  # split into training data and cross validation data
  train_data <- training(data_split)
  valid_data <- testing(data_split)

  # Train the GBM model
  boost <- gbm(log_total ~ ., data = train_data, n.trees = 1000,
shrinkage = 0.04, n.minobsinnode = 28, distribution =
"gaussian")
  # shrinkage is learning rate, n.minobsinnode is min_n
  # Predict on the validation set
  valid_preds <- predict(boost, newdata = valid_data, n.trees =
1000)
 # Calculate RMSE for the validation set
  results <- valid_data %>% mutate(predictions = valid_preds)
  rmse1[i] <- rmse(results, truth = log_total, estimate =
predictions)$.estimate
}

# Display RMSE results (each iteration and mean)
rmse1
mean(rmse1)

# Predict on test data
pred <- predict(boost, newdata = test, n.trees = 1000)
# Combine predictions with test set IDs
predictions <- test %>% select(id) %>%
  bind_cols(pred) %>%
  rename(log_total = ...2)
# Save the prediction to a csv file
write_csv(predictions, "Predictions.csv")
```

7.2 Team member contributions

Tianlin Yue: Model development and tuning for Random Forest Model and Boosting Tree model(xgboost), preparation and model analysis of Boosting Tree(xgboost) in the report.

Rueiyu Chang: Finish the introduction and conclusion sections, and overall report coordination.
Kai Liang: : Finish the discussion section.
Yiming Xue: Conducted model preprocessing, built and tuned Linear Regression, Linear Regression with Interaction, Regularized Linear Regression, Decision Tree, Random Forest, and Boosting Tree models using gbm, and reported the results.
Ziwei Guo: Data Visualization and Data Analysis

8. Citation

Puška, A., Stojanović, I., Šadić, S., & Bečić, H. (2018). The influence of demographic characteristics of consumers on decisions to purchase technical products. *The European Journal of Applied Economics*, *15*(2), 1–16. https://doi.org/10.5937/ejae15-16576