Team 12: Katherine Huynh, Victoria Aye, Oliver Siu, and Vivian Yee

Introduction

The dataset includes survey data from 5000 Amazon customers collected between January 2018 and December 2022, divided into training and test sets. The training set contains information on customer demographics, household size, and account sharing patterns, containing over 34 variables, with the response variable being the logarithm of total order costs (log_total). Key variables include order counts and demographics. According to Iowa State University's article "Income and Substitution Effects," higher income can increase demand for goods. Additionally, the article "What is the Relationship Between Family Size and Mercer Spendable Income Amounts?" notes that larger households tend to spend more. Thus, customer income and household size are expected to be significant predictors of 'log_total,' reflecting consumer behavior trends.

References:

https://www.kaggle.com/competitions/ucla-stats-101-c-2024-su-regression/overview
https://parsnip.tidymodels.org/
https://mobilityexchange.mercer.com/insights/article/what-is-the-relationship-between-family-size-and-mercer-spendable-income-amounts#:~:text=Insights-,What%20Is%20the%20Relationship%20Between%20Family%20Size%20and%20Mercer%20Spendable,but%20at%20a%20decreasing%20rate
https://www2.econ.iastate.edu/classes/econ101/hallam/Income_Substitution.pdf

EDA

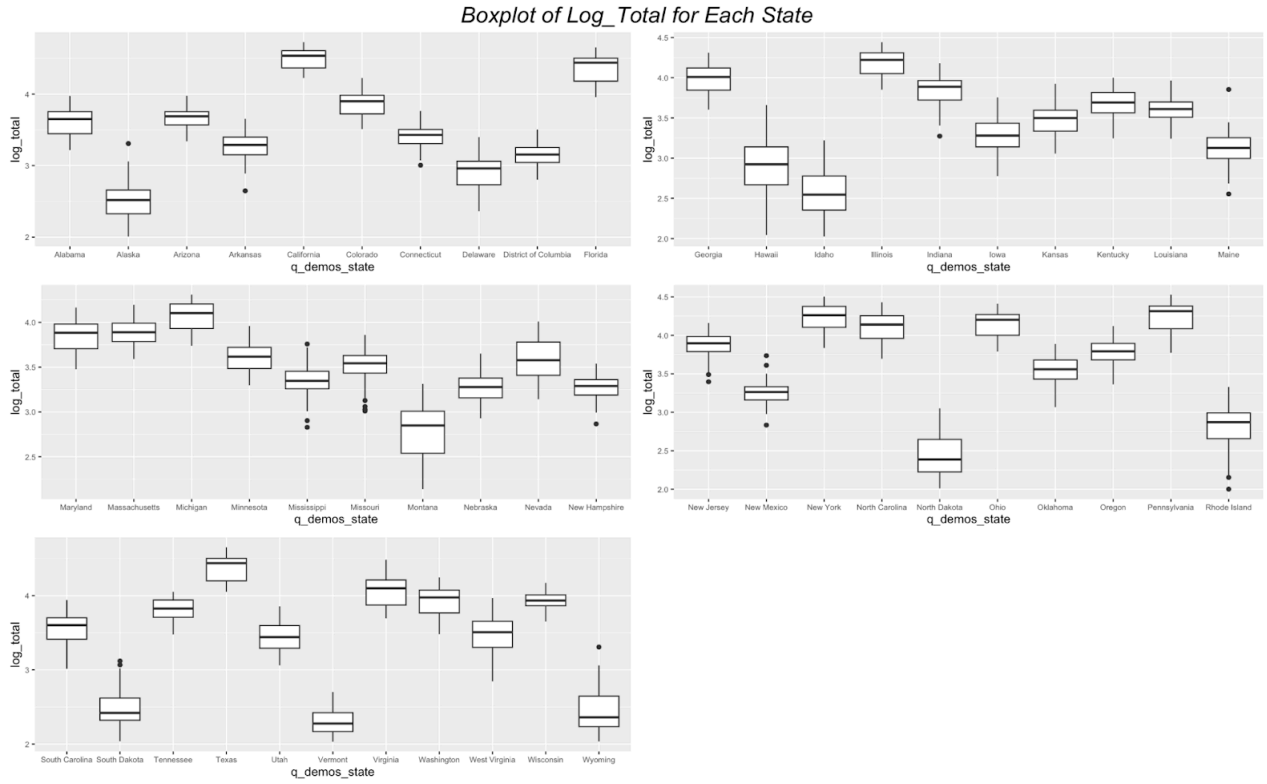
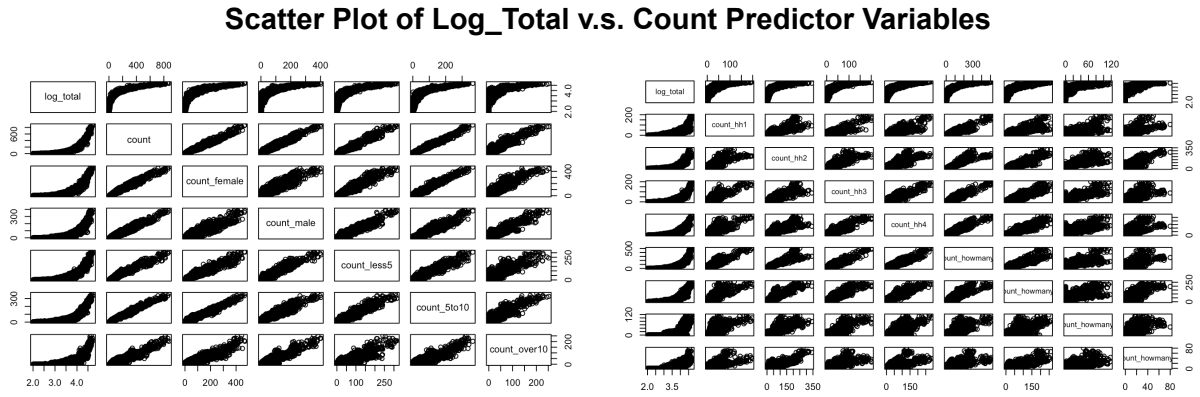


Figure 1

Most states have fairly high median order totals of around 10^3.5 to 10^4. However, some states like Alaska, Hawaii, Idaho, Montana, North Dakota, Rhode Island, South Dakota, Vermont, and Wyoming have a median order total of 10^2.5 to 10^3, suggesting a potential relationship between certain states and order total.



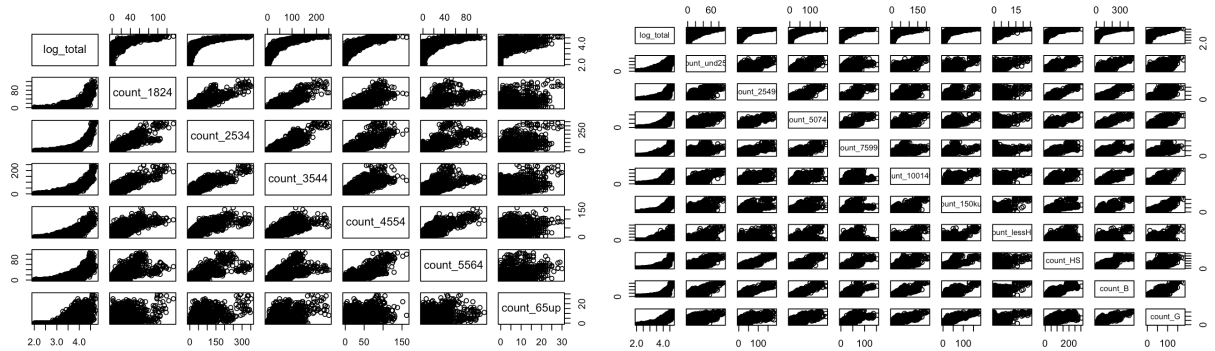


Figure 2

In this model, we plotted all the count predictor variables against log_total. We note that all count predictors have a slight positive correlation with log_total. So, we decided it could be worthwhile to investigate any linear relationships between log_total and the other count predictors using the "glm" engine.

Scatter Plot of Transformed Count Predictor Variables Against Log_Total using Power Transformation

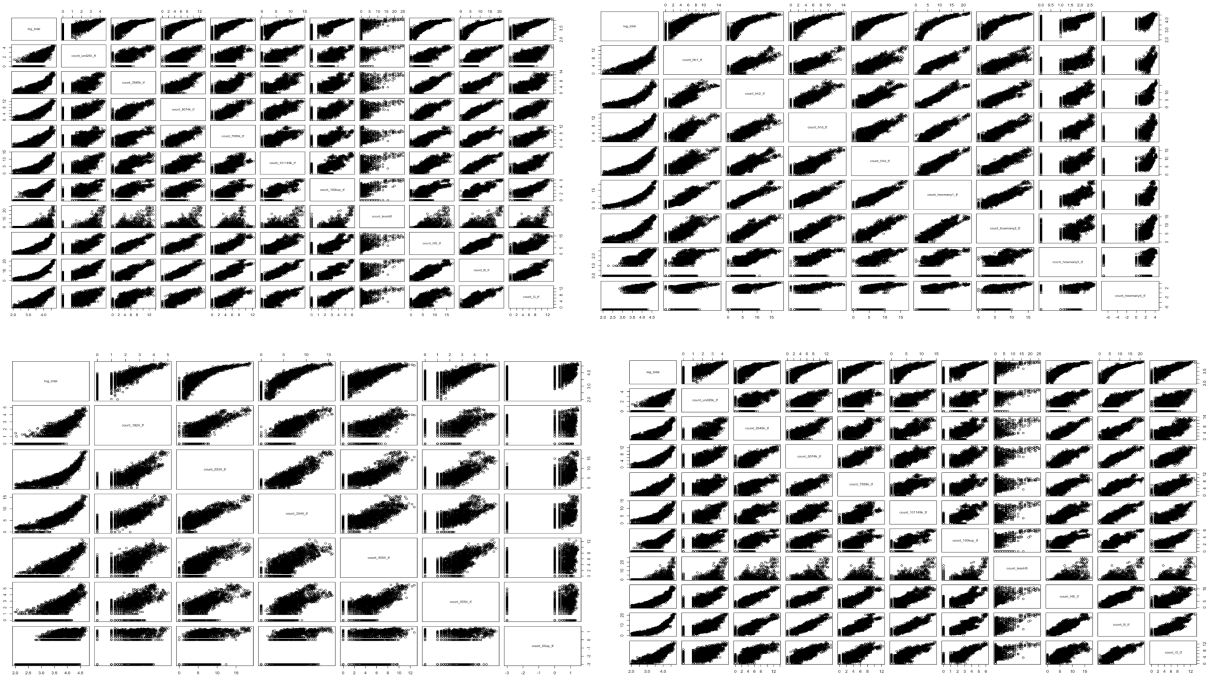


Figure 3

After applying the recommended power transformations, we noticed the scatterplots were overall slightly more linear than before. Thus, we noted that if we use linear regression models, it would be worthwhile to try transforming the count predictors so that it follows model assumptions.

Scatterplot Matrix for Logarithmic Transformation of All Numeric Predictors

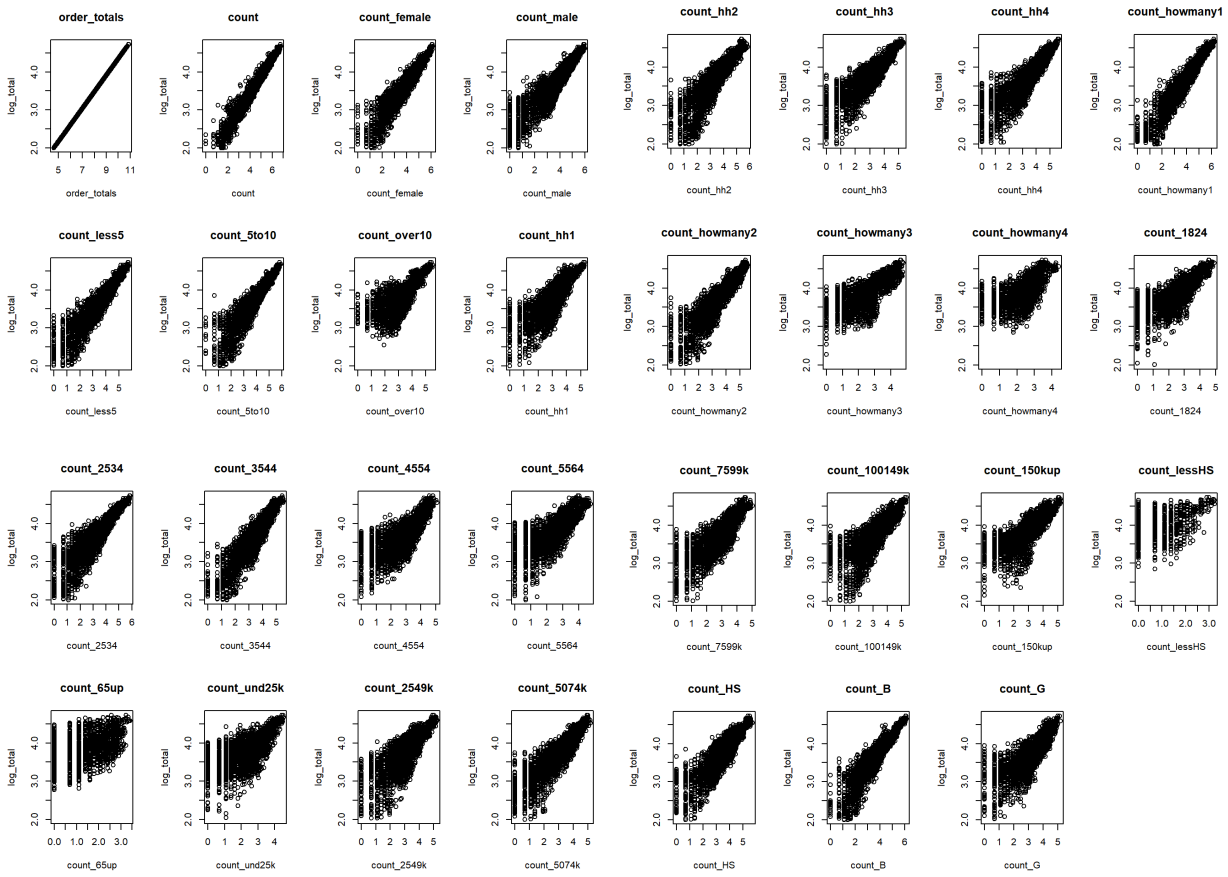


Figure 4

Because the response variable is a log transformation of order_totals, we also considered a log transformation of all numeric predictors as well. Most transformed predictors exhibited more linear relations with log_total, but the variances increased as the predictor values decreased, which indicated a potentially problematic pattern.

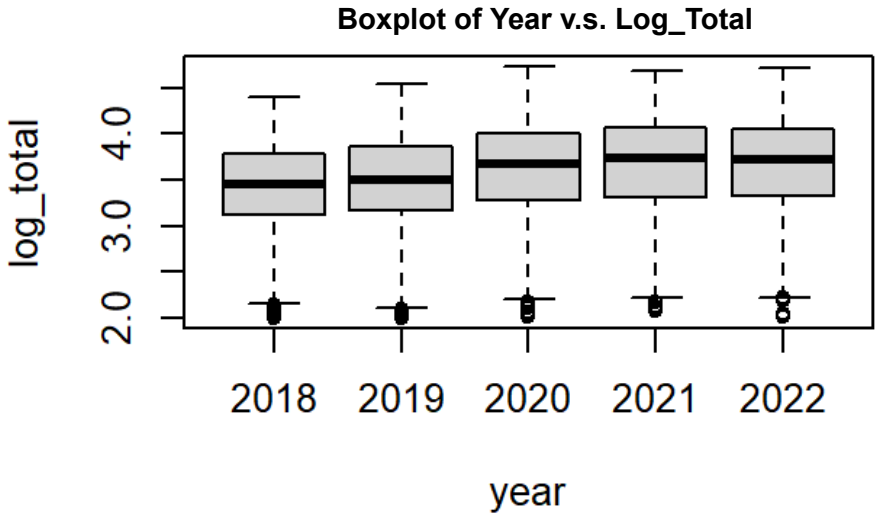


Figure 5

In this model, we plotted different years against log_total. We note that log_total does not vary amongst different years. So, it could be worthwhile to investigate dropping this variable when investigating different linear models.

Boxplot of Different Months v.s. Log_Total

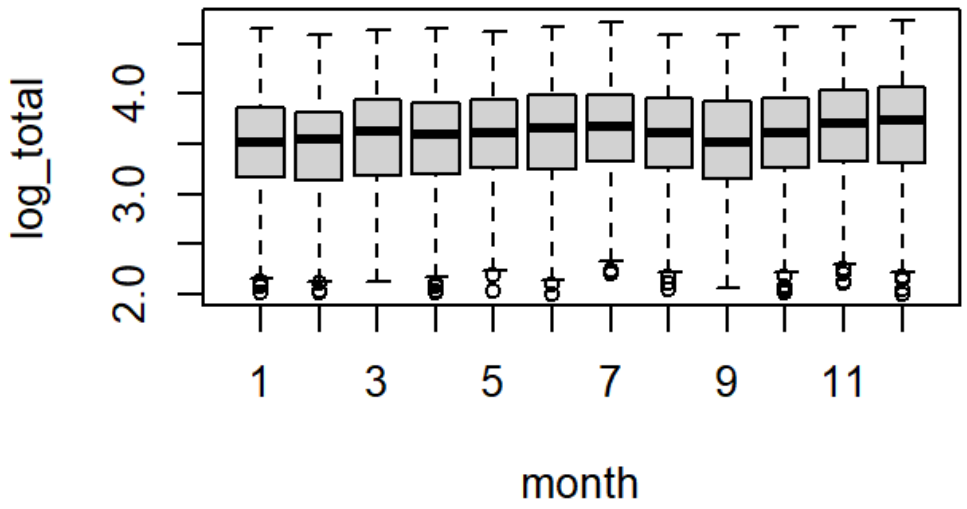


Figure 6

In this model, we plotted different months against log_total. We can note that the log_total does not vary amongst different months. So, it could be worthwhile to investigate dropping this variable, in addition to years, as well when investigating different lm models.

Diagnostic Plots for Linear Model of log_total v.s. factor(year), factor(month) and factor(a_demos_state)

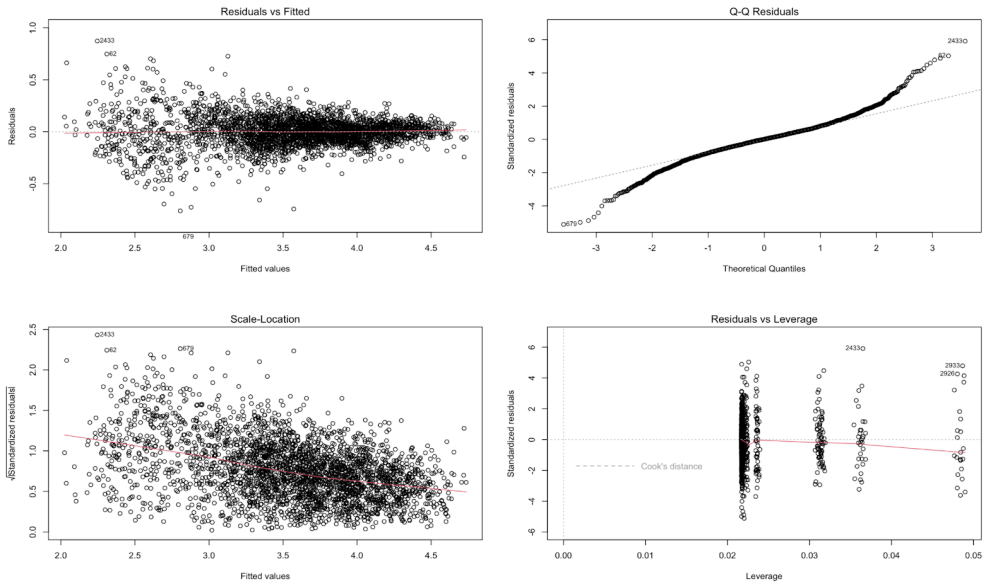


Figure 7

From this plot, we can note that the overall model appears to have an average residual of 0. There appears to be somewhat of a constant variance and normality of the error terms. Thus, this linear relationship appears to be following model assumptions and could be a potential model.

Assessing Multicollinearity of log_total v.s. factor(year), factor(month) and factor(a_demos_state)

	GVIF	Df	GVIF^(1/(2*Df))
factor(year)	1.007828	4	1.000975
factor(month)	1.009896	11	1.000448
factor(q_demos_state)	1.017419	50	1.000173

Figure 8

Because all of the VIFs are less than 5, we can be assured that this model does not have multicollinearity. Since these variables are not highly correlated, the linear regression model between state, month, and year would be a more reliable model and thus more worthwhile to investigate.

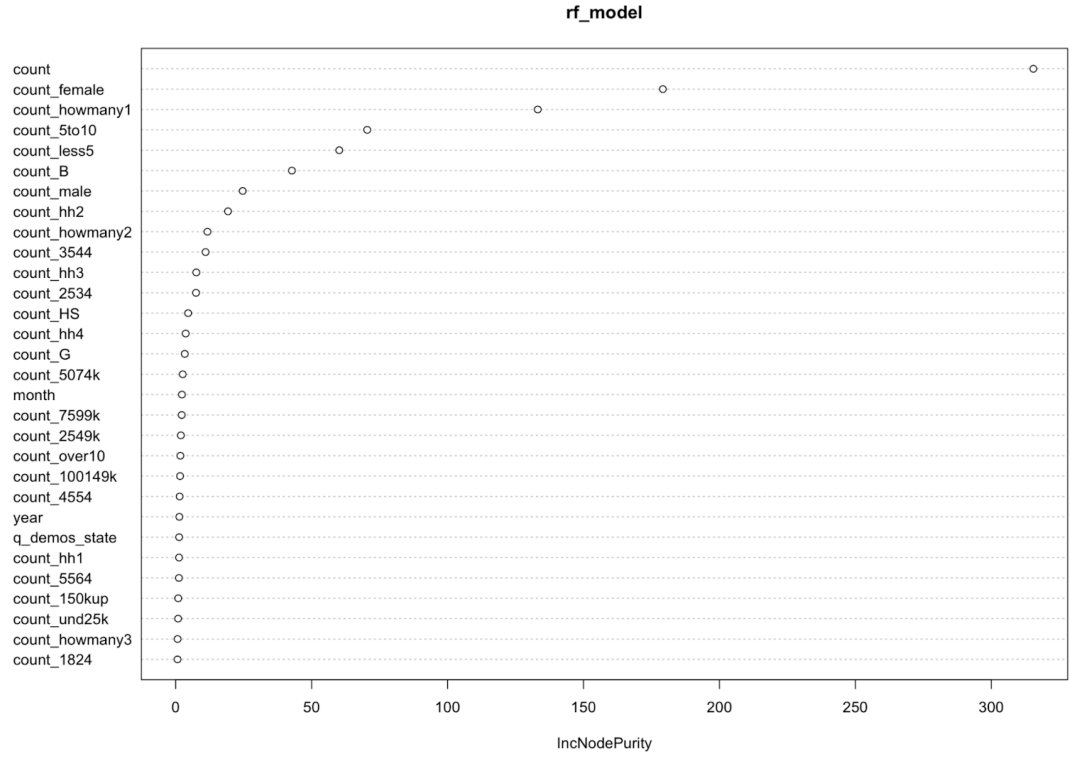


Figure 9

We plot the feature of importance for the rf_model (random forest model). This plot shows the significance of each predictor in predicting log_total. From the plot, the most important variables are count, count_female, count_howmany1, etc. The variables with low IncNodePurity could be considered for removal to improve models.

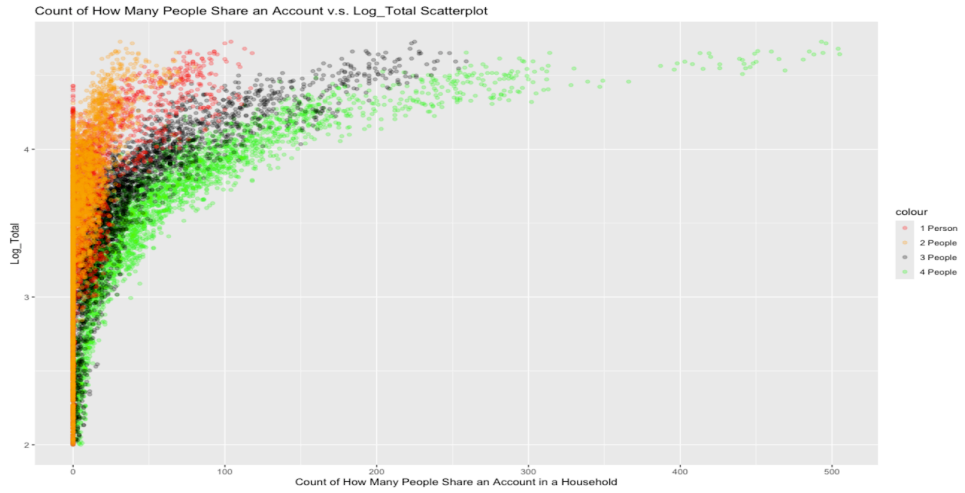


Figure 10

From this model, we can see that there are a lot more accounts that are shared between four people compared to accounts that are used by one person. Thus, it would be worthwhile to consider dividing the count_howmany predictor variables into each subcategory when predicting log_total.

Histogram Matrix for Predictor Distributions

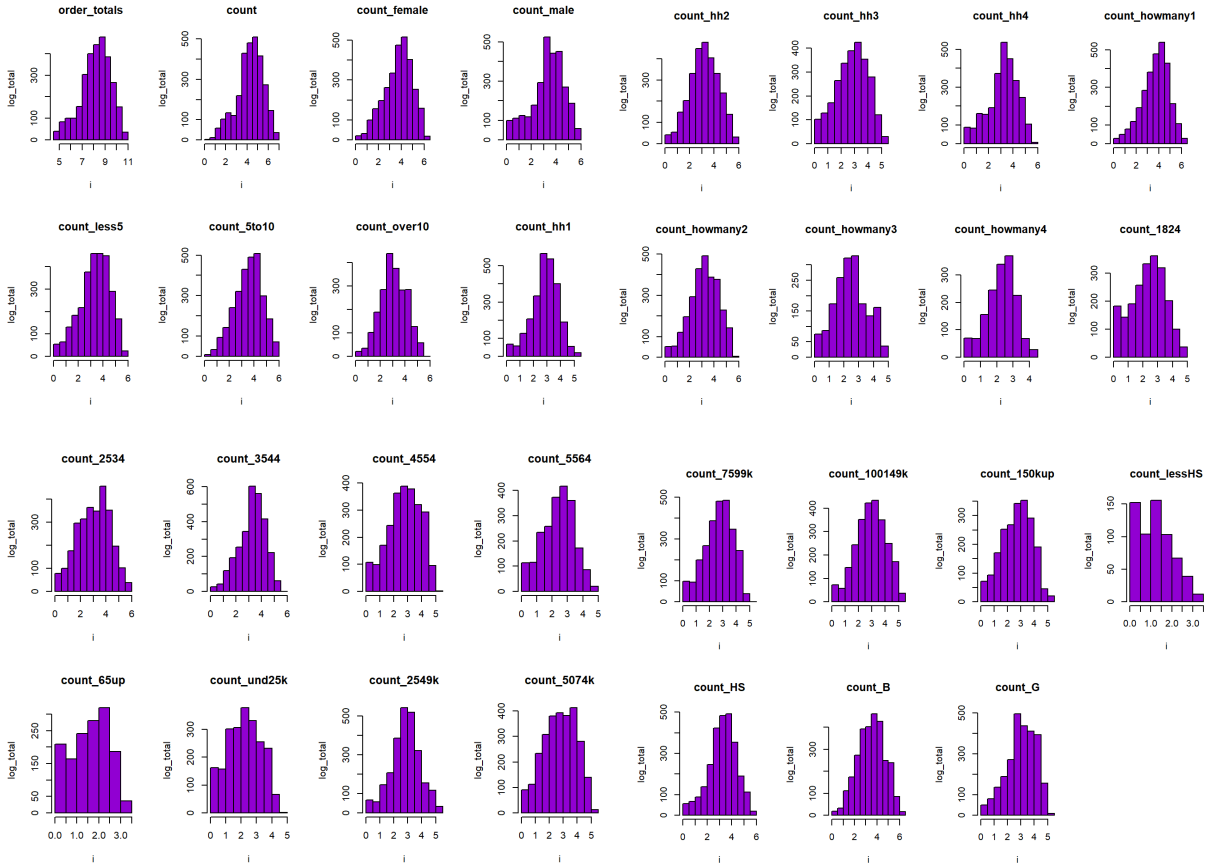


Figure 11

The predictors are generally left-skewed or normally distributed except for count_lessHS. count_lessHS has the least counts compared to the rest of the counts for education level, so removing count_lessHS was a potential consideration during variable selection since count_lessHS did not seem to follow a similar pattern as the rest of the predictors.

Preprocessing and Recipes

For preprocessing, order_totals was removed. To investigate removal of predictor variables, we decided to create two recipes. The first recipe was Base Recipe which was a basic recipe of log_total versus all of the other predictor variables in the training dataset. In this recipe, we decided to convert the month and year variables into factors so that R would not view higher values of these variables as more significant. For instance, December, or the value 12, in the month variable should not be more significant than January, which has a value of 1. We also decided to factor the q_demos_state variable because the states variable could easily be divided into 51 different levels.

Because our train data has a significant number of predictor variables, we decided to create an additional recipe to experiment with. For our second recipe, we wanted to get rid of any possible predictor variables that have multicollinearity. Thus, we applied the step_corr() function on all predictors and applied the tuning() function on threshold to investigate various thresholds. We then applied these two recipes to five different models: linear regression, decision tree, k nearest neighbor, random forest, and boosted tree. Then, we used v-fold cross validation with v = 10 on our train data as a way to compare the models in the workflow set. Using cross validation, we called the workflow_map() function to evaluate the MAE, RMSE, and SE of each of the 10 created models. To account for tuning, we added the "tune_grid" function to the workflow map so that R could try and tune each model. After creating the workflow map, we then called the rank_results() function to numerically display the different metrics of each model. In doing this, we discovered that the basic recipe (without the step_corr() filter) appeared to perform better. So, we primarily used this recipe of log_total against all predictors for a majority of our models.

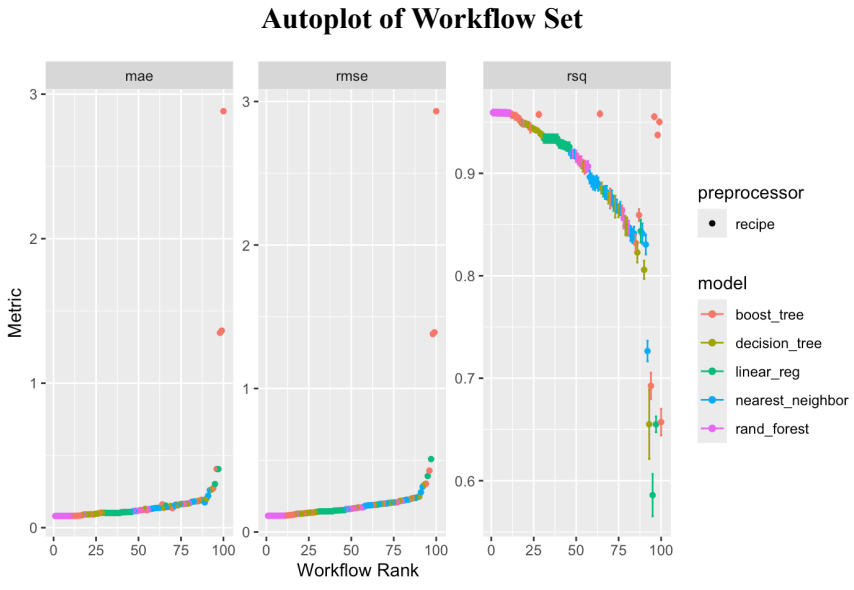


Figure 12

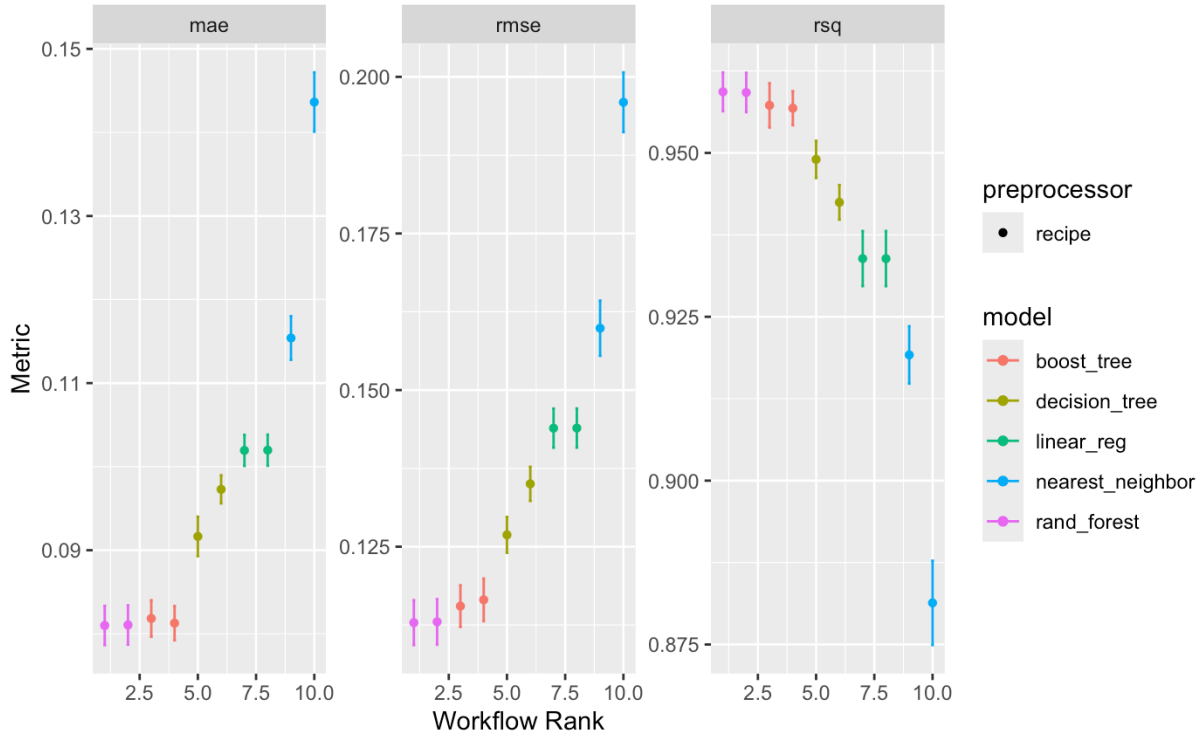


Figure 13

wflow_id <chr>	.config <chr>	.metric <chr>	mean <dbl>	std_err <dbl>	n <int>	preprocessor <chr>	model <chr>	rank <int>
simple_rf	Preprocessor1_Model07	mae	0.08099008	0.001425893	10	recipe	rand_forest	1
simple_rf	Preprocessor1_Model07	rmse	0.11287662	0.002177554	10	recipe	rand_forest	1
simple_rf	Preprocessor1_Model07	rsq	0.95932425	0.001801153	10	recipe	rand_forest	1
filter_rf	Preprocessor06_Model1	mae	0.08105252	0.001427922	10	recipe	rand_forest	2
filter_rf	Preprocessor06_Model1	rmse	0.11300379	0.002212721	10	recipe	rand_forest	2
filter_rf	Preprocessor06_Model1	rsq	0.95922643	0.001825923	10	recipe	rand_forest	2
filter_xgboost	Preprocessor02_Model1	mae	0.08182889	0.001332107	10	recipe	boost_tree	3
filter_xgboost	Preprocessor02_Model1	rmse	0.11551574	0.002024673	10	recipe	boost_tree	3
filter_xgboost	Preprocessor02_Model1	rsq	0.95726236	0.002058604	10	recipe	boost_tree	3
simple_xgboost	Preprocessor1_Model08	mae	0.08127219	0.001244646	10	recipe	boost_tree	4
simple_xgboost	Preprocessor1_Model08	rmse	0.11652024	0.002065288	10	recipe	boost_tree	4
simple_xgboost	Preprocessor1_Model08	rsq	0.95682633	0.001577767	10	recipe	boost_tree	4
simple_cart	Preprocessor1_Model04	mae	0.09164728	0.001438783	10	recipe	decision_tree	5
simple_cart	Preprocessor1_Model04	rmse	0.12688745	0.001740871	10	recipe	decision_tree	5
simple_cart	Preprocessor1_Model04	rsq	0.94900917	0.001728155	10	recipe	decision_tree	5
filter_cart	Preprocessor06_Model1	mae	0.09728290	0.001025020	10	recipe	decision_tree	6
filter_cart	Preprocessor06_Model1	rmse	0.13502912	0.001663440	10	recipe	decision_tree	6
filter_cart	Preprocessor06_Model1	rsq	0.94245857	0.001602712	10	recipe	decision_tree	6
simple_glmnet	Preprocessor1_Model08	mae	0.10194367	0.001130468	10	recipe	linear_reg	7
simple_glmnet	Preprocessor1_Model08	rmse	0.14391861	0.001898326	10	recipe	linear_reg	7
simple_glmnet	Preprocessor1_Model08	rsq	0.93387592	0.002555103	10	recipe	linear_reg	7
filter_glmnet	Preprocessor06_Model1	mae	0.10197212	0.001132584	10	recipe	linear_reg	8
filter_glmnet	Preprocessor06_Model1	rmse	0.14393697	0.001899641	10	recipe	linear_reg	8
filter_glmnet	Preprocessor06_Model1	rsq	0.93387279	0.002558156	10	recipe	linear_reg	8
simple_knn	Preprocessor1_Model05	mae	0.11539655	0.001590784	10	recipe	nearest_neighbor	9
simple_knn	Preprocessor1_Model05	rmse	0.15987696	0.002686169	10	recipe	nearest_neighbor	9
simple_knn	Preprocessor1_Model05	rsq	0.91918322	0.002660911	10	recipe	nearest_neighbor	9
filter_knn	Preprocessor06_Model1	mae	0.14363417	0.002159619	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor06_Model1	rmse	0.19594420	0.002899959	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor06_Model1	rsq	0.88135048	0.003914990	10	recipe	nearest_neighbor	10

30 rows

Figure 14

A Brief Overview of Each Candidate Model and its Results

For the first candidate model, we tested a random forest using a ranger engine and the base recipe. With this model, we wanted to understand whether a ranger engine or a randomForest engine would be a better model for our data. We were also curious to see what combination of hyperparameters would give us the best RMSE without overfitting. We used Latin Hypercube through grid_latin_hypercube() for model tuning, which introduces randomization to the hyperparameter search within the given range. Although we tuned the model and tested for overfit, this model didn't perform as well as our other candidates seen in the higher RMSE; therefore, we chose not to include this model in our autoplot and instead compare our other models.

For our second candidate model, we discovered from our workflow set that 3 out of the top 5 models used the base recipe (not the recipe with the step_corr() function); we decided to cut it off at top 5 because the mse afterwards would be at least greater than 0.0161. So, we decided to use the base recipe for our ensemble model. Then, out of the models with base recipes, we noted that the random forest model, boosted model, and the decision tree model performed the best. So, we created an ensemble model using these 3 models and the base recipe where the parameters (min_n, trees, and cost_complexity) would be tuned. Although this model did not perform as well as the other candidate models based on its RMSE, it was one of the lowest RMSE values we noticed out of all the ensemble models.

For the third candidate model, we tested for MLP and used a neural network with the nnet engine and the base recipe to predict the target variable. We focused on optimizing hyperparameters such as hidden units, penalty, and epochs using a Latin hypercube search, aiming for the best RMSE while avoiding overfitting. Despite tuning, this model's performance, indicated by a higher RMSE, was not as strong as our other models, so we decided not to include it in the final comparison.

For the fourth model, we wanted to try out different models to combine in an ensemble model. So, we decided to try out another 3 types of models: k nearest neighbor, linear regression, and support vector machine model. Overall, these were three models that were not used in the previous ensemble candidate model. So, we were curious if using different models would allow us to be able to outperform the previous ensemble model. Also, we were curious if there was any other models that could be worthwhile to combine with our previous ensemble model. This candidate ensemble model did not have as low of an RMSE as our previous ensemble model. Thus, for ensemble models overall, it would be better to consider the second model.

The fifth model was a random forest model using the aorsf engine and the base recipe. With the default parameters, the aorsf engine performed very well. It was our 4th best model based on the public scores and our best model based on the private scores. Regrettably, this model was not chosen for the final submission. Due to limited time, we opted to pursue tuning randomForest engine models instead as they tended to score higher on the public leaderboard.

The sixth model was a hyper parameterized random forest model that used the randomForest engine. Based on the public scores, our best model at the time was a randomForest engine model. This model was responsible for helping the team rank 1st on the public leaderboard for the extra credit opportunity. Therefore, we attempted to tune the model for better performance. Despite efforts to manually tune the model, the lowest training RMSE combination from a random grid search exhibited the best results with a lower training RMSE than the untuned model, but the public score was still the same as the untuned model. This model was still ultimately chosen for submission in hopes that the increase in variance would fit the private data better.

Candidate Models/Model Evaluation/Tuning

Candidate Model Number	Model Identifier	Type of Model	Engine	Recipes or Listing of Other Variables in Model:	Hyperparameters
1	rf_model	Rand_Forest	"ranger"	Base Recipe	trees(range = c(400, 500)), mtry(range = c(2, 10)), min_n(range = c(2, 10)), size = 20
2	stacks	stacks of random forest, boosted tree, and decision tree	"randomForest", "xgboost", "rpart"	Base Recipe	trees(range = c(400, 600)), mtry(range = c(1, 33)), min_n(range = c(2, 10)), cost_complexity (range = c(0, 1))
3	mlp_model	MLP	"nmet"	Base Recipe	hidden_units(range = c(1, 10)), penalty(range = c(0, 0.1)), epochs(range = c(50, 100)), size = 20
4	stacks2	Stack of k nearest neighbor, linear regression, and a support vector machine model	"kknn", "lm", "kernlab"	Base recipe, Filter recipe	Neighbors = tune("k"), Cost = tune("cost"), rbf_sigma = tune("sigma")
5	aorsf_2038_full	Random Forest	"aorsf"	Base Recipe	N/A
6	rf_2017_tune_auto	Random Forest	"randomForest"	Base Recipe	trees(range = c(400, 600)), mtry(range = c(1, 33)), mtry = 14 trees = 548 min_n = 27

Table of Metric Results

Model Identifier	RMSE	SE (if applicable)
aorsf_2038_full	0.111128	0.003197376
rf_2017_tune_auto	0.1130085	001968296
rf_model	0.113	0.00193
stacks	0.113	N/A
mlp_model	0.126	0.00382

stacks2	0.144	N/A

Evaluation and Comparison of Models (V-Fold Cross Validation and Tuning)

To evaluate and compare the candidate models, we used v-fold cross validation with 10 folds, which was stratified on the log_total response variable. This method splits the data into multiple folds and trains and tests the models on different subsets. We mainly evaluated the models using the Root Mean Squared Error (RMSE), which is the square root of the average of the squared differences between actual and predicted values. From the RMSE that R generated, we found that the aorsf_2038_full model performed the best since it had the lowest RMSE of 0.111128, while other models had RMSEs that closely followed. Additionally, for some models (if applicable), we considered the Standard Error (SE). The SE for the aorsf_2038_full model was 0.003197376, indicating low variability in the RMSE estimate and stable performance.

We also used the stacks autoplot function to better understand which type of model generally did the best at creating accurate predictions. As mentioned earlier, we used a workflow set to identify which recipes and engines would perform the best across a set of metrics (MAE, RMSE, and RSQ). After investigating various combinations, we noted that the boosted tree and random forest model seemed to perform the best, which is shown in how these models have the highest weight; this is also shown in the stacks autoplot results in figure 13 and figure 14. Additionally, it is interesting to note how stacks2 model had linear regression as the highest weight, which is shown in figure 21. However, we decided against including linear regression in our final model since the overall scatterplots between the predictor variables and the response variable log_total appeared to be more nonlinear than linear (as seen in figure 2). Also, linear regression did not have as big of a weight as the other models (as seen in figure 13).

Autopilots for the Candidate Models (except for mlp_model)

Note: We excluded the mlp_model because we decided that this model had a higher RMSE than the most of the models apart from stacks2, which could be observed through a clear glance at the table. So, we deemed it not worthwhile to investigate the autoplot for this model. However, because we only had two main ensemble models, we decided it could be worthwhile to further investigate the stacks2 model.

Autoplot comparing the 3 Random Forest Models

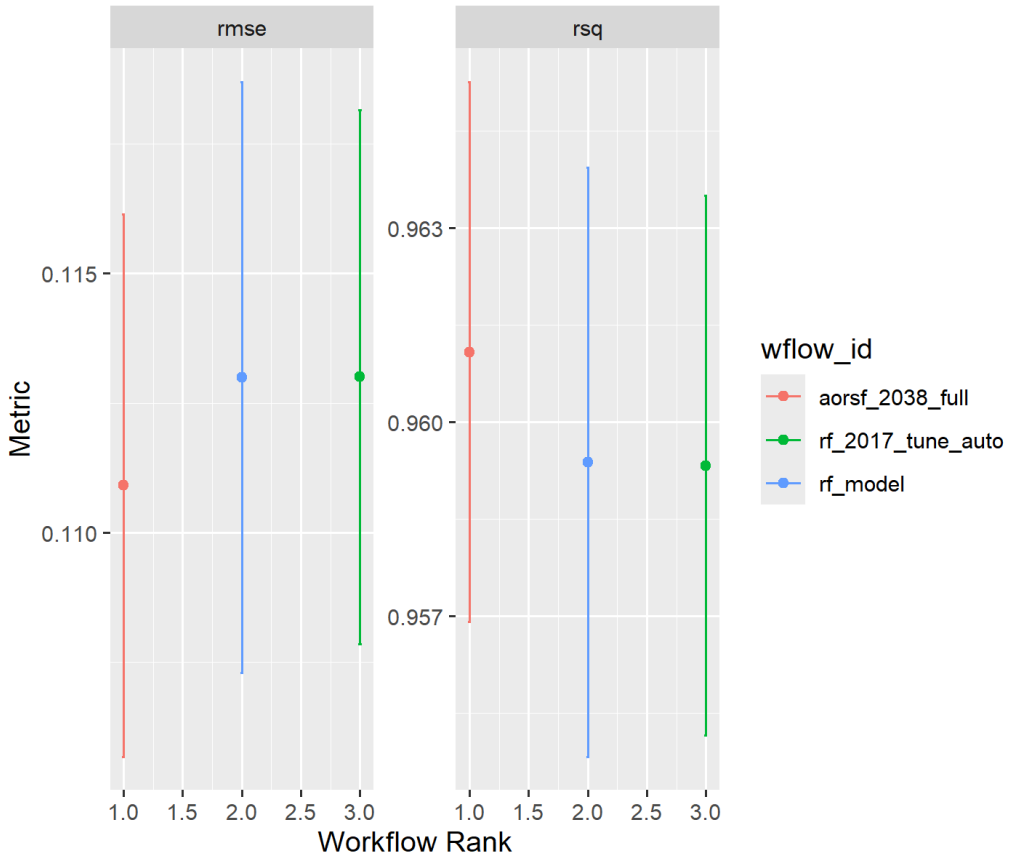
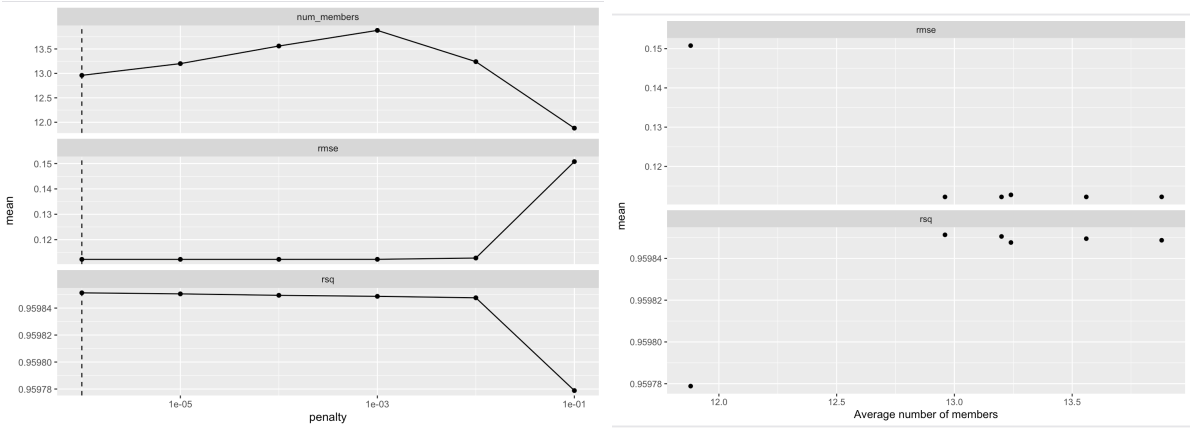


Figure 15

Autoplot of stacks Model



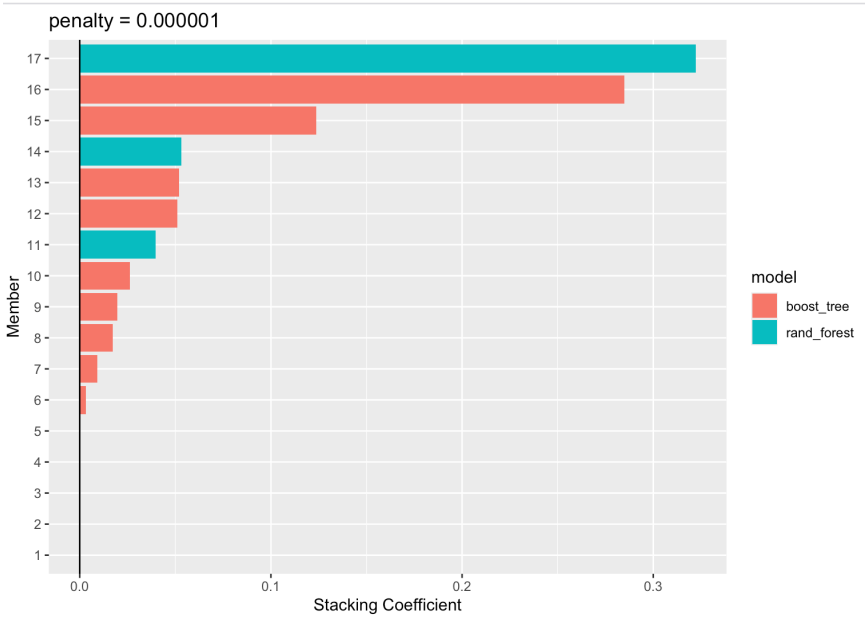
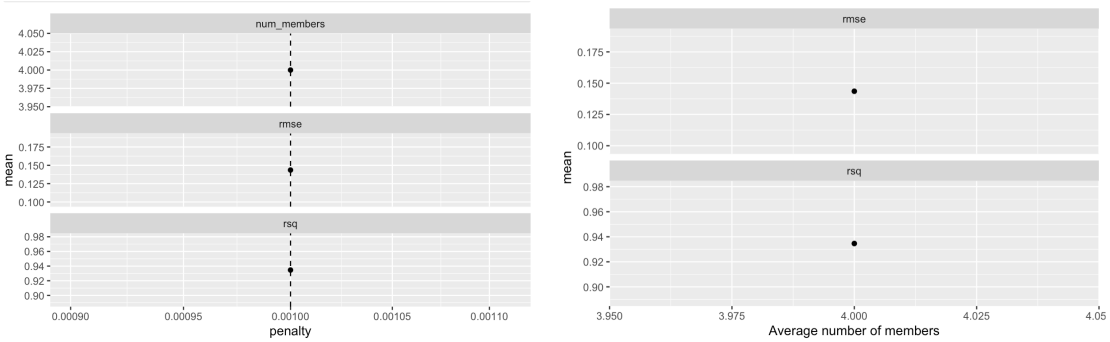


Figure 16 - 18

Autoplot of stacks2 Model



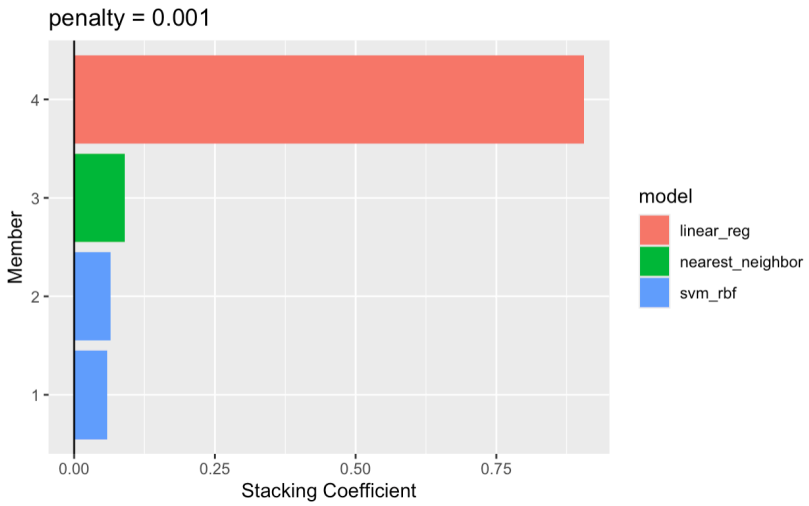


Figure 19 - 21

Lastly, to optimize model performance, we tuned hyperparameters for each model using grid search combined with cross validation with specified ranges. Using these methods, we identified the best hyperparameters and found our best model. For instance, we noted that the trees parameter had the best range of 400 to 600 and mty had the best range of 1 to 33. From these ranges, we had R auto select the best parameter, which was used in our rf_2017_tune_auto model.

Tuning for Ensemble Models (Stacks and Stacks2)

For the ensemble model, we were able to tune the model by setting each parameter to tune() when we were defining the different engines. Then, we used the tune_grid() function and sent it that specific model's workflow, the cross fold validation set, our specified grid with 3 levels, our specified metric set, and the base control grid we defined previously. In this base control grid, we specified save_pred to be TRUE and save_workflow = TRUE so that our assessment set predictions and the workflow used on the resamples are actually going to be stored inside the object. In doing this, we allow R to experiment with various levels to tune each hyperparameter. As previously mentioned, we noted that the stacks model had a lower RMSE than the stacks2 model. Thus, we decided it would be more beneficial to use the stacks model as our primary ensemble model candidate compared to stacks2.

Discussion of Final Models

Selecting our Final Model

When we were selecting our models, we primarily were deciding between rf_model/rf_2017_tune_auto, stacks, and the aorsf_2038_full model. Although they all had similar RMSE, we ultimately decided to utilize the public leaderboard score on Kaggle to assist our selection. For instance, even though the stacks model and the rf_2017_tune_auto had approximately the same RMSE on R Studio, we noticed that the rf_2017_tune_auto model actually performed better on the public leaderboard than the ensemble model. Thus, we ultimately decided to prioritize this model over the stacks model. Additionally, we also noticed that aorsf_2038_full and the rf_model had a lower RMSE than the rf_2017_tune_auto model. However, when submitting it to the Kaggle leaderboard, we noted that these two models actually performed worse on the public leaderboard. So, we assumed that these models would be riskier because it could have potentially overfit our data.

Thus, our final model was rf_2017_tune_auto. However, because rf_2017_tune_auto was a high-ranking potential model that could overfit our data, we decided to have our second model be a low-ranking model with less variance. Overall, we thought that the stacks ensemble model would be the safest option. By averaging three models, stacks is able to more likely prevent any overfitting that could occur should we choose to solely use one type of model as shown in rf_2017_tune_auto. So, the overall strength of this model is that we can better assure that the RMSE R estimates will be more similar to the RMSE we will get from the test/unseen data. Ultimately, the rf_2017_tune_auto performed better on all of the test data than the stacks model, so this model was the final choice.

Strengths and Limitations of the rf_2017_tune_auto Model

A strength of the random forest model like rf_2017_tune_auto is that random forests, unlike decision trees, are able to randomly select variables and thus avoid creating trees that are too similar. By creating many diverse trees, it is able to combine predictions of each of the trees to overall reduce the model's variance. This then ensures that random forest models, like our rf_2017_tune_auto mode, would be less prone to overfitting than a final model that uses a decision tree. Another benefit of this model is that this model can also accommodate for nonlinear relationships between variables. As shown in the scatterplot previously, the relationship between log_total and the various count_predictors appeared to be more non-linear. So, a linear model such as lm would perform worse than the random forest model.

One limitation of this model is that this rf_2017_tune_auto model utilizes a tree range of 400 to 600. Having such a high amount of trees could be potentially detrimental for our model because it could entail a possibility of us overfitting our data. Another limitation of this model is that this model, in its fundamental algorithm and also the use of all predictor variables, tends to be harder to interpret than other models such as linear regression models.

Final Thoughts

If we were to improve this model in the future, it would be beneficial for us to combine this model with a boosted tree model and the other aorsf_2038_full candidate model. Individually, each model has different pros and cons. For instance, the boosted tree model tends to have a higher predictive capability than the random forest model; however, the boosted tree model is harder to tune. However, these three models tended to perform the best amongst our candidate models. So, it could be beneficial for us to investigate an ensemble model that utilizes these three types of models to hopefully create a more

precise prediction. Additionally, this model primarily tried trees across a range of 400 to 600. In the future, it could be worthwhile to sacrifice more computational time so that R can try out more levels and better tune the trees parameters. For instance, we can play around with less trees to see if we can try to better ensure our model does not overfit.

Some additional data that could be useful is regional data, specifically whether an area is urban or rural. Even though this data returned the amount of order totals per state, this data did not provide any information about the amount of order totals across geographic regions. So, a potential relationship we would like to explore is whether rural areas would have less order totals. For instance, is it possible that rural areas, in being more difficult to transport packages, would make express delivery like Amazon more difficult? As a result, would individuals in rural regions be less likely to order goods through Amazon? Another factor that could be useful is the unemployment rate in that state during that time period. We would also be interested to see more information on household data, such as the count of individuals who are married or the count of how many children, to explore whether the actual distribution of household members impacts the amount people order goods. For instance, are there a lot of parents who use Amazon to order goods because they don't have enough time to physically purchase goods in person? Or, is there no strong relationship between these variables at all? Overall, these are the types of additional data we wish we could have had access to during this project so that we could create a model that could more precisely predict order totals.

Appendix

Final Model Submission

rf_2017_tune_auto.R by Oliver Siu

Screen recording of script running 2024-07-28 14-34-42.mp4

https://drive.google.com/file/d/1J1KYCUWd1MBk4RDE5ubWAFHwuVBDQJ4i/view?usp=sharing

```
rf_2017_tune_auto.R

######################### rf_2017_tune_auto.R ############################

# load libraries and set preferences
library(tidyverse)
library(tidymodels)
tidymodels_prefer()

# load data
train <- read_csv("train.csv")
orders_train <- train %>% select(!order_totals) # remove order_totals
test <- read_csv("test.csv")

# set seed
set.seed(2017)

# setup folds for k-fold (v-fold) cross-validation
orders_folds <- vfold_cv(orders_train, v = 10, strata = log_total)

# setup recipe
rf_recipe <-
  recipe(log_total ~ ., data = orders_train) %>%
  step_mutate(q_demos_state = factor(q_demos_state),
              year = factor(year),
              month = factor(month))

# setup model with tuning parameters
rf_model <-
  rand_forest(
    mtry = tune(),
    min_n = tune(),
    trees = tune()
  ) %>%
  set_engine("randomForest") %>%
  set_mode("regression")
```

```
# set tuning parameters ranges
model_param <-
  extract_parameter_set_dials(rf_model) %>%
  update(trees = trees(range = c(400, 600)),
         mtry = mtry(range = c(1, 33)))

# grid research (retained to preserve reproducibility)
rf_tune <-
  model_param %>%
  grid_random(size = 100)

# setup workflow
rf_workflow <-
  workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_model)

# fit workflow to tuning grid
rf_rand_tune <-
  rf_workflow %>%
  tune_grid(
    orders_folds,
    grid = model_param %>% grid_random(size = 10)
  )

# find the best tuning parameters based on RMSE
best_spec <-
  rf_rand_tune %>%
  select_best(metric = "rmse")

# finalize workflow using the best tuning parameters
rf_wf_auto <-
  rf_workflow %>%
  finalize_workflow(best_spec)

# k-fold (v-fold) cross-validation
rf_res_auto <-
  rf_wf_auto %>%
  fit_resamples(resamples = orders_folds)

# fitting model to training data
```

```
rf_wf_auto_fit <-
  rf_wf_auto %>%
  fit(data = orders_train)

# making predictions on test data
orders_test_res <-
  rf_wf_auto_fit %>%
  predict(new_data = test) %>%
  bind_cols(test %>% select(id)) %>%
  select(id, log_total = .pred)

# writing out csv with id and predicted log_total values
write_csv(orders_test_res, "submission.csv")
```

Team Member Contribution

A. Katherine Huynh
    a. Created boxplot of log_total v.s. States, scatter plot of count_variables v.s. Log_total, and scatter plot of log_total v.s. Transformed version of count_variables
    b. Organized combination of 2 recipes and 5 models and placed them into a workflow set, and placed them in the autoplot function to identify best performing models
    c. Created and organized zoom meetings for the group
    d. Created and submitted the stacks model as one of the 2 final prediction models
    e. Composed entire discussion of final models section
    f. Created figure 1, 2, 3, 7, 8, 10, 12-14, 16-20
B. Oliver Siu
    a. Created figures 4, 11, and 15
    b. Handled exploration and testing of all random forest engines
    c. Created the model that ranked number 1 for the extra credit opportunity
    d. Responsible for the top 4 best private score models and the best public score model
    e. Composed sections on and related to candidate models 5 & 6
C. Vivian Yee
    a. Created figure 9 (features of importance plot)
        i. Wrote significance
    b. Composed the evaluation and comparisons of models section
    c. Checked for interactions between variables by creating plots
    d. Attempted random forest model and tune hyperparameters
        i. Incorporated interaction between count of household and count to improve predictive accuracy
D. Victoria Aye
    a. Wrote introduction
    b. Test best hyperparameters after model tuning and doesn't overfit after identifying random forest as one of the better models
        i. Results in one of the top 5 private scores out of all our submissions but was not chosen for reasons stated above
    c. Tested MLP model and conclude that MLP wasn't a good model to use due to high RMSE
    d. Composed all writings related to mlp_model and rf_model