Stats 101C Regression Project Report

Maxwell Chu, Bella Gordon, Tran Nguyen, Vanda Suklar, Nicole Yamachika

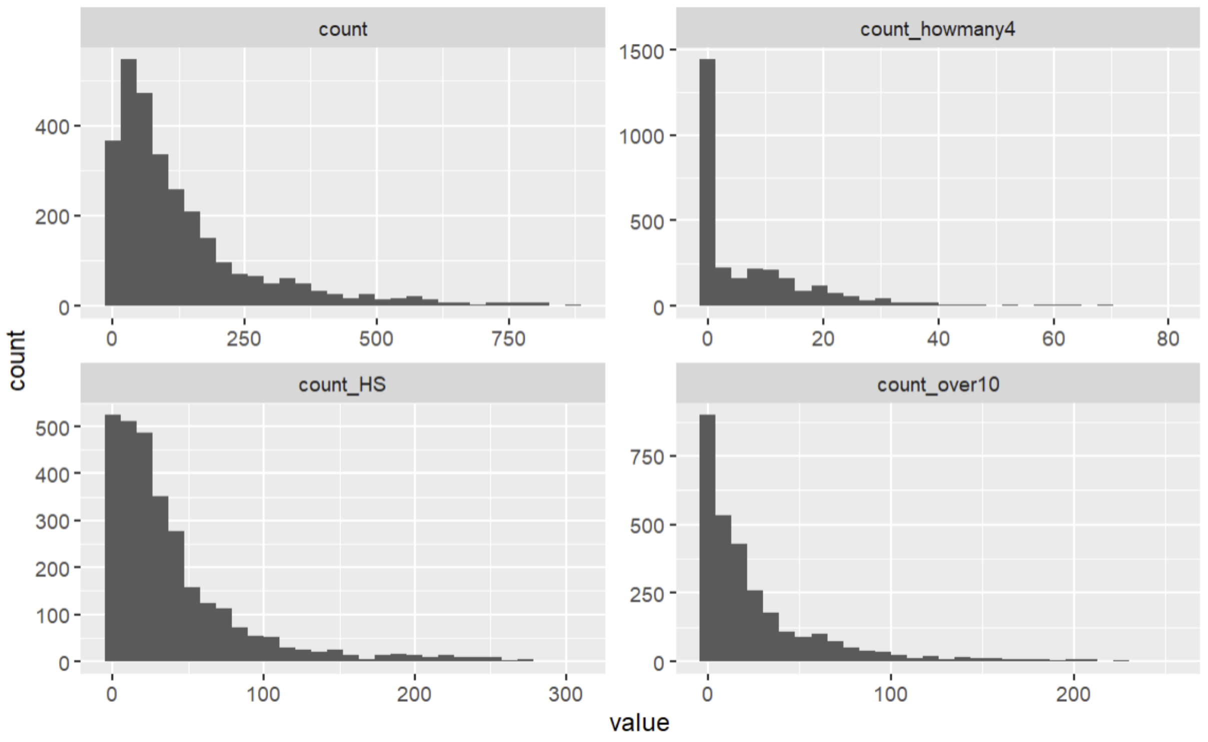
Table of Contents

Introduction
Exploratory Data Analysis
Preprocessing
Candidate Models, Model Evaluation, & Tuning
Final Model
Appendix: Final Annotated Script
Appendix: Team Member Contributions

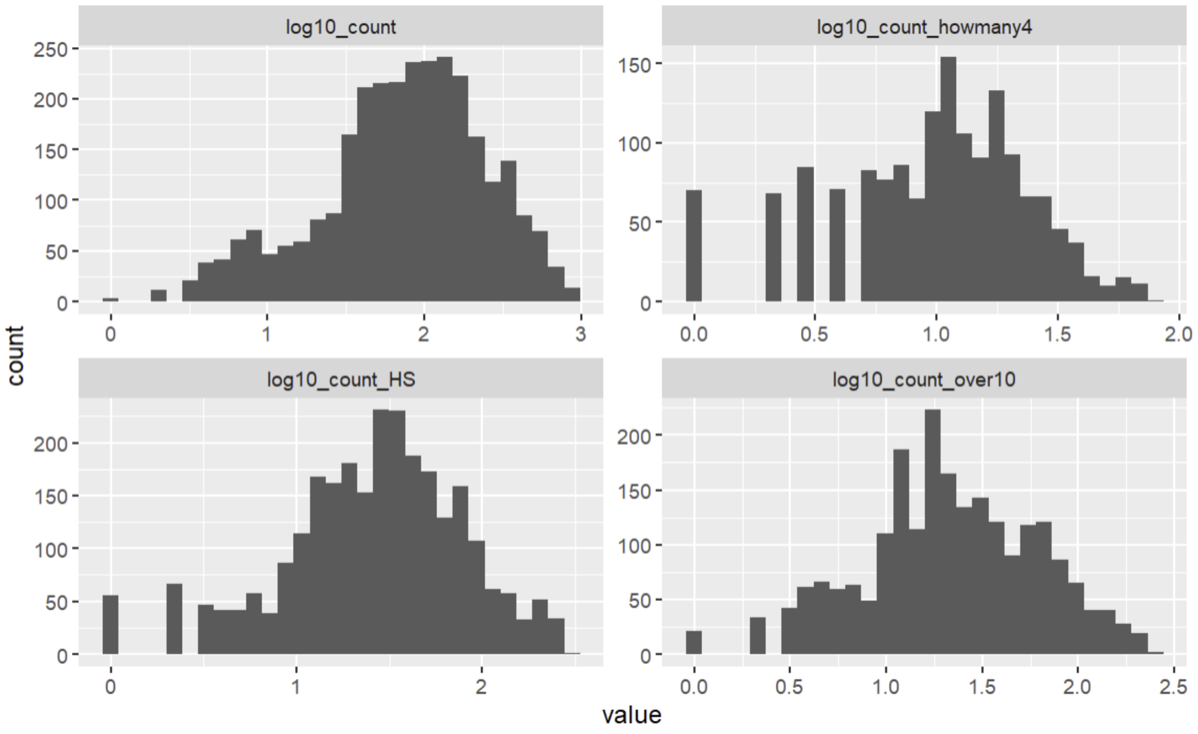
Introduction

Our dataset of interest regards a survey of Amazon customers and their orders, with our assigned response variable being the log amount (labeled as *log_total*) of how many orders were placed in a US state during a given month and year. In approaching the task of creating a predictive model for this data, identifying key variables associated with the response variable was crucial for us to determine how to improve our models and their accuracy. Some variables of interest were state, age distribution, household size, income, and education level as we believed that these variables likely influence customer purchasing behavior and, consequently, *log_total*. By understanding the associations between the predictive variables and *log_total*, we can improve our model's predictive performance (as evaluated with Mean Squared Error, *MSE*) and gain insights into customer purchasing patterns.

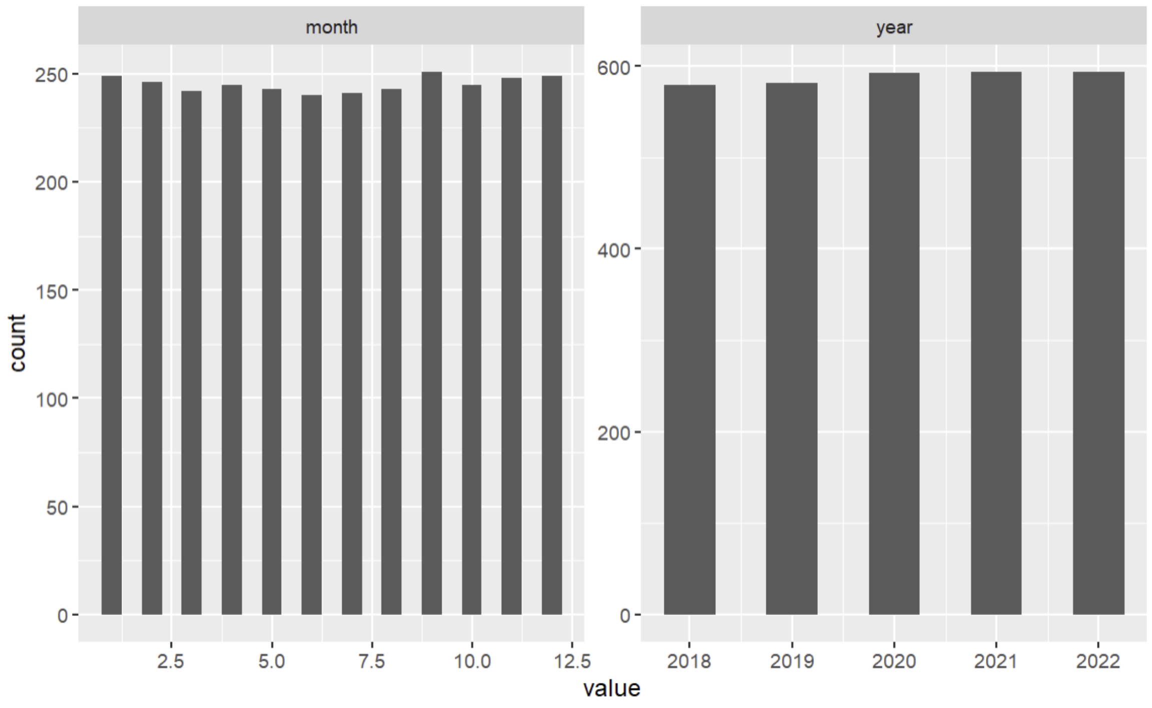
Exploratory Data Analysis



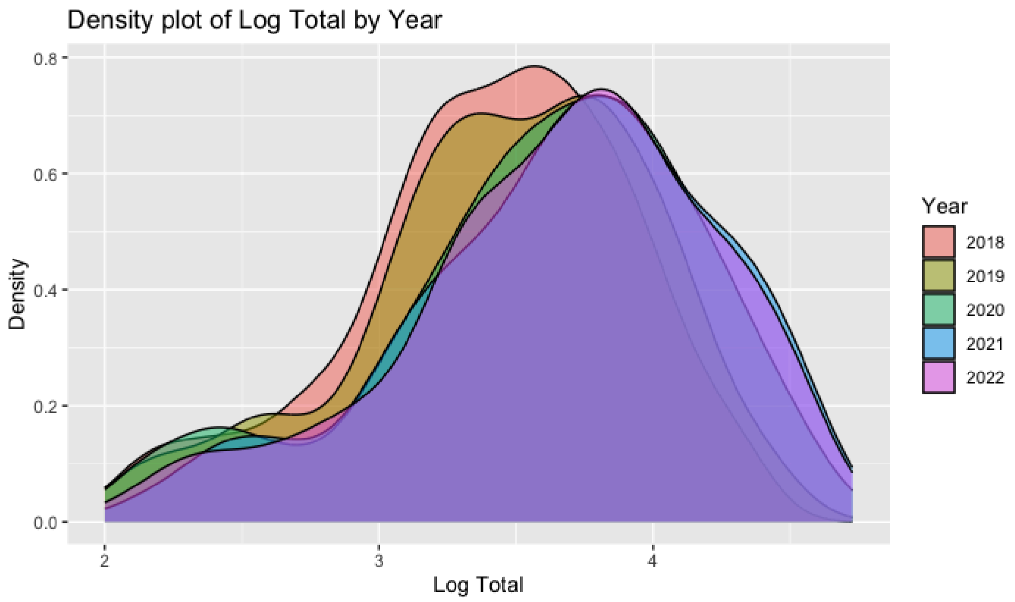
Visual 1 - All the predictors beginning with "count" are similar in that they are right-skewed. This makes sense; across all Amazon customers, regardless of survey result, people are less likely and less able to order more items. People are also more likely and more able to order fewer items.



Visual 2 - The same variables from Visual 1, log-transformed. This transformation removes skew and generally makes the distribution of these variables more similar to a normal distribution.

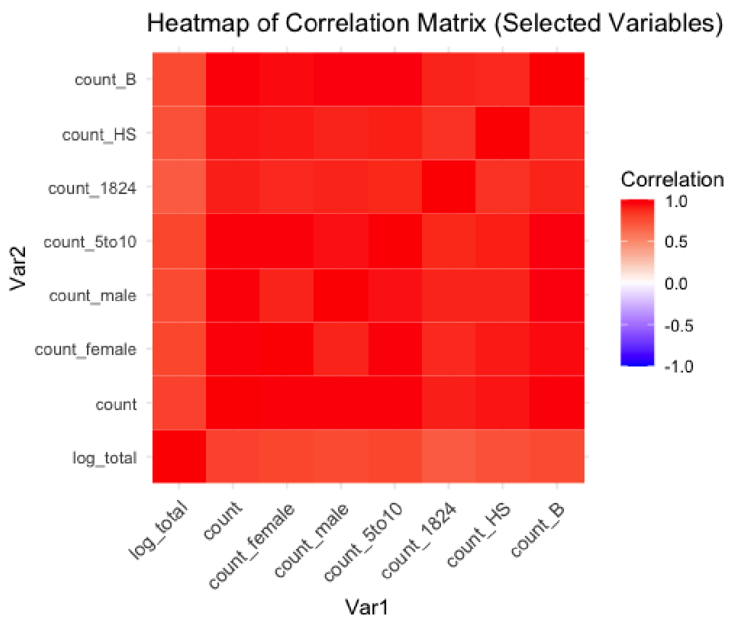


Visual 3 - The variables "month" and "year" can only take on twelve and five values, respectively. Thus, they should be converted from numeric to categorical variables using the factor function. Ordinary numeric variables usually are not so limited in the range of values they can take on.

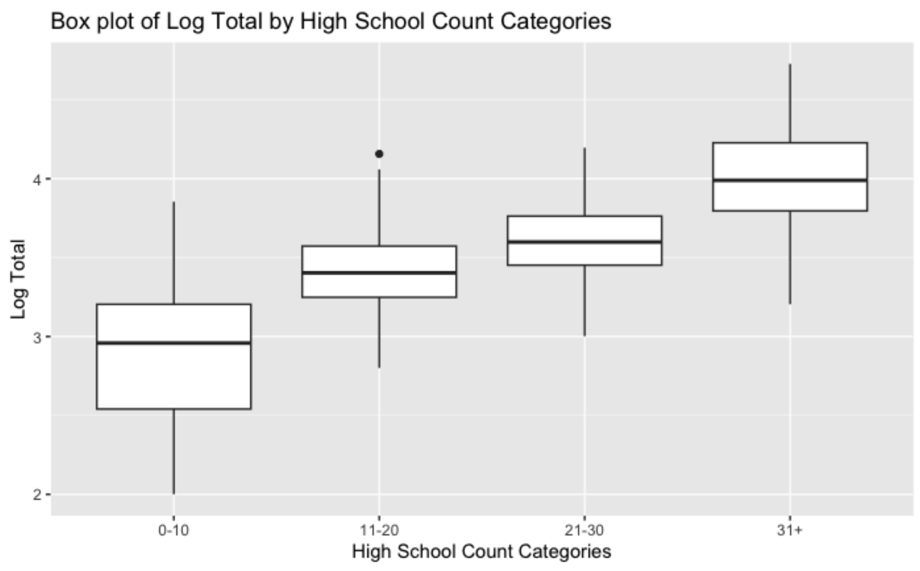


Visual 4 - This density plot shows the distribution of log_total across different years. It highlights how the distribution of order totals has changed over time. The density plot provides

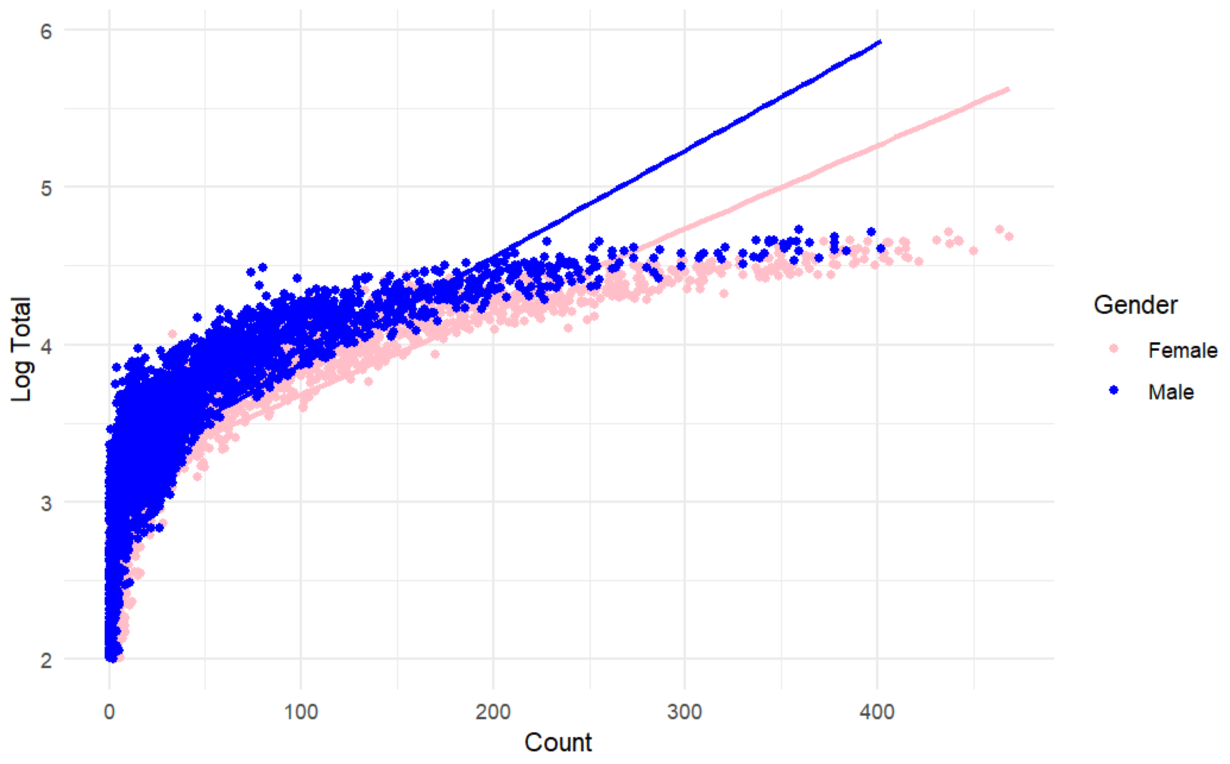
insights into how the distribution of order totals has evolved over the years. It can reveal trends, shifts, and the overall shape of the data distribution for different years.



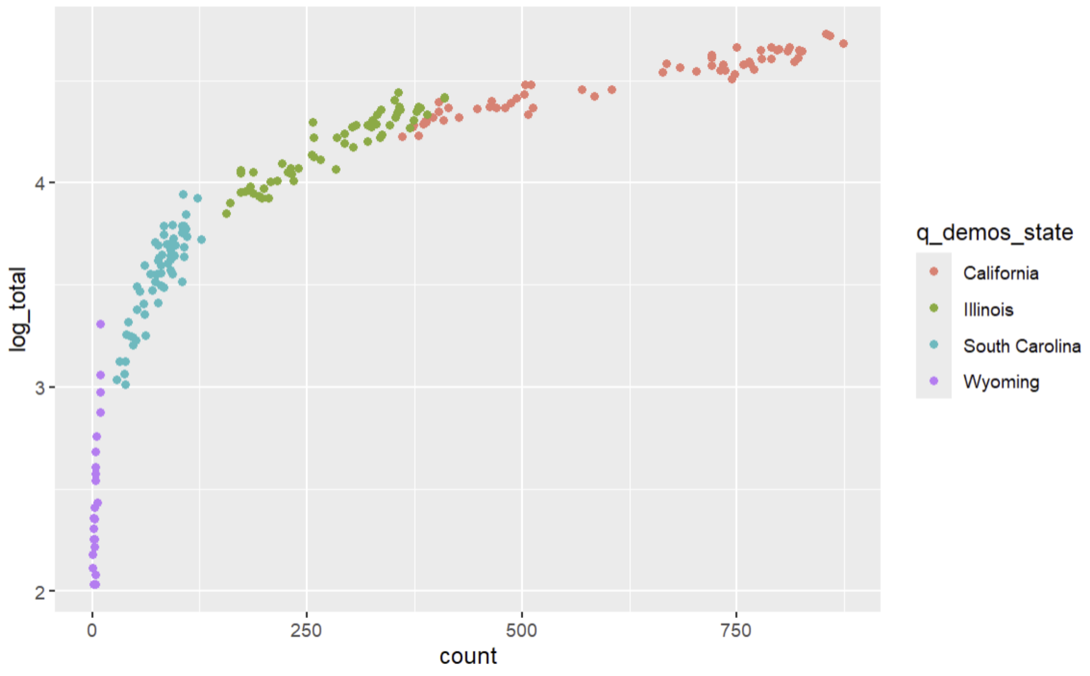
Visual 5 - The heatmap shows the correlation between selected numeric variables. This subset of variables helps in identifying multicollinearity issues and guiding variable selection for the regression model. High correlations among predictors can lead to multicollinearity, which can affect the stability and performance of regression models.



Visual 6 - This box plot shows the distribution of log_total across different high school count categories. Grouping the data improves readability and helps in identifying the impact of education level on order totals. Higher counts of high school education are associated with higher log_totals, suggesting a possible link between education level and purchasing behavior.



Visual 7 - A scatterplot of log_total vs count_female/count_male. log_total represents the log of total costs of items, and count_female/count_male represent the number of orders placed by male and female customers, respectively. Thus, male customers may purchase higher-cost items than female customers in general, though, as the plot illustrates for larger values of count_female/count_male, female customers may in general purchase more items than male customers.



Visual 8 - A scatterplot of log_total vs count with selected states. Clearly, the relationship between count and log_total is partially dependent on the state; we would expect a customer in California to order more items than a customer in Wyoming, and thus, generally speaking, have a higher value for log_total.

Preprocessing

In order to evaluate the model's performance robustly, and to prevent our models from overfitting to the training set, we split the data into 10-fold cross-validation sets, stratified by *'log_total'*. Stratification ensures that each fold has a similar distribution of the response variable, improving the reliability of the cross-validation results.

Only candidate models 2, 4, and 7 used preprocessing steps in their recipes. The first step was step_mutate_at(q_demos_state, year, month, fn = factor), which converts those three variables into factors/categorical variables. Since each of those variables may only take on a limited number of values, it makes sense to make them categorical variables. The model would not correctly use the year or month predictors if they were numeric.

The second step was step_log(contains("count"), offset = 1). This step log-transforms all numeric predictors in the data set. The optional argument offset = 1 adds one to the values of each transformed variable before employing the log-transform. This prevents the case where log(0) is attempted, which is an undefined value that messes up the model fitting. Since the lowest value in any numeric predictor variable in the data set is 0, there is no worry about negative values screwing up the log-transform. Performing this transformation unskews the numeric data and makes them more similar to a normal distribution.

As Visuals 4 and 8 illustrated above, the q_demos_state, year, and month variables have some relationship with the count variable. Thus, we used step_interact(~ q_demos_state:count + year:count + month:count) to create interaction terms that may help capture those relationships between the predictors, which may translate to a more accurate prediction of the response.

Finally, the last step is a simple step_dummy(all_nominal()) to create dummy variables in order to make the three categorical variables quantitatively useful to the model.

Candidate Models, Model Evaluation, & Tuning

ID	*Type of Model*	*Model Engine*	*Recipe*	*Hyperparameters*

1	Random Forest	Ranger	simple_train_recipe <- recipe(log_total ~ ., data = train)
			rf_workflow <- workflow() %>%
			add_recipe(simple_train_recipe) %>% add_model(rf_model)
				trees = 500
				mtry = 5
				min_n = 5

2	Random Forest	Ranger	rf_recipe <- recipe(log_total ~ ., data = rf_train)
			step_mutate_at(q_demos_state, year, month, fn = factor)
			step_log(contains("count"), offset = 1)
			step_interact(~ q_demos_state:count + year:count + month:count)
			step_dummy(all_nominal())
				trees = 378
				mtry = 13
				min_n = 12

3	Random Forest	Ranger	rf_recipe <- recipe(log_total ~ ., data = train)	mtry = 15
				trees = 1500
				min_n = 25

4	Linear Regression	lm	lm_recipe <- recipe(log_total ~ ., data = train)
			step_mutate_at(q_demos_state, year, month, fn = factor)
			step_log(contains("count"), offset = 1)
			step_interact(~ q_demos_state:count + year:count + month:count)
			step_dummy(all_nominal())
				N/A

5	Random Forest	Ranger	rf_workflow <- workflow() %>%
			add_recipe(recipe(log_total ~ ., data = train)) %>%
			add_model(rf_model)
				N/A

6	Random Forest	Ranger	workflow_rf <- workflow() %>%
			add_model(rf_model) %>%
			add_formula(log_total ~ .)
				N/A

7	Boosted Tree	XGBoost	bt_recipe <- recipe(log_total ~ ., data = train)
			step_mutate_at(q_demos_state, year, month, fn = factor)
			step_log(contains("count"), offset = 1)
			step_interact(~ year:count + month:count)
			step_dummy(all_nominal())
				mtry = 4
				trees = 836
				min_n = 16
				tree_depth = 13
				learn_rate = 0.028903

Candidate model #1: One simple candidate model used the random forest model with engine ranger. The arguments on rand_forest() were left as the default, and the predictions performed with a MSE of 0.01778. Assessing the model, using collect_metrics(), the results for the RMSE is 0.1144501. The number of trees used by the model were 500, the target node size was 5, and the Rsquared was 0.9588428.

Candidate model #2: A random forest model with tuned hyperparameters trees, mtry, and min_n. All numeric predictors were log-transformed to reduce skewness. The predictors "year" and "month" were converted from numeric to categorical variables, and then interaction terms between each categorical variable and the numeric predictor "count" were used. In testing, the RSME of the predictions was 0.1097141, with a standard error of 0.002628477.

Candidate Model #3: Our third candidate model was another random forest model. For this model, no hyperparameters for the independent variables were included, but we did set hyperparameters of the random forest model during initialization. The reasoning behind each hyperparameter: mtry was set to 15 so as to sample just below half of the independent variables at each split of the trees, trees was set to 1500 in an attempt to increase the complexity of the model and account for as many combinations of variables as possible without overfitting the model, and min_n was set to 25 to include a moderate amount of data points in a node prior to splitting to both increase model complexity and avoid overfitting.

Candidate Model #4: A barebones linear regression model with no hyperparameters. It uses the same recipe as model #2. The Kaggle RMSE for this model was much worse than those of the other models.

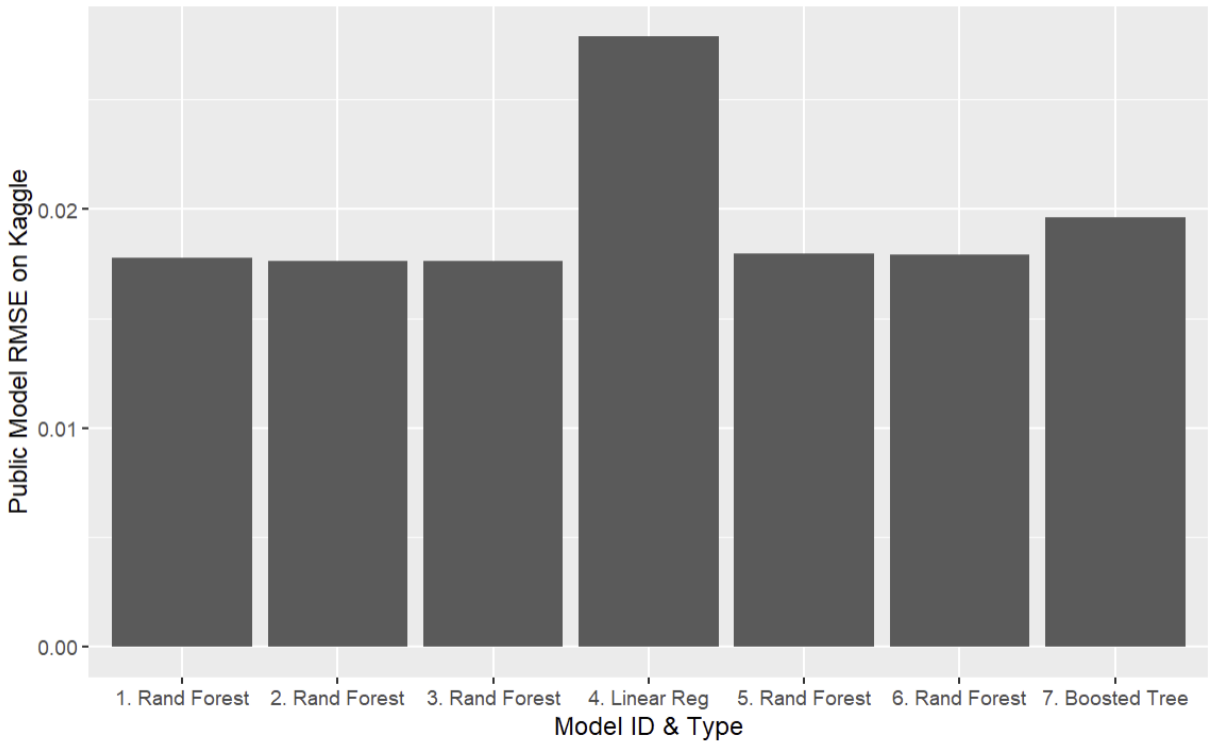
Candidate Model #5: A straightforward candidate model utilized a random forest approach with the ranger engine and the regression mode. The default settings were maintained for the rand_forest() function, resulting in predictions with a mean squared error (MSE) of 0.0129899. When evaluating the model with collect_metrics(), the root mean squared error (RMSE) was 0.1139732, and the R-squared value was 0.9585583.

Candidate Model #6: This model employed a random forest approach using the ranger engine in regression mode. The model was trained using the default settings of the rand_forest() function. After evaluating the model with cross-validation, the performance metrics were obtained using the collect_metrics() function. The model achieved a root mean squared error (RMSE) of 0.1136208 and an R-squared value of 0.9587198. The standard error of the RMSE was 0.002894019, indicating the model's predictions are fairly precise.

Candidate Model #7: A boosted decision tree model with two extra hyperparameters compared to the random forest models, tree_depth and learn_rate. The engine used was XGBoost. Its Kaggle RMSE was only slightly worse than those of the random forests. The same recipe as models #2 and #4 was used.

Model ID	*Type of Model*	*MSE (Public)*	*RMSE (R)*	*Standard Error*
1	Random Forest	0.01778	0.1144501	0.002854840
2	Random Forest	0.01764	0.1097141	0.002628477
3	Random Forest	0.01763	0.1124903	0.002646305
4	Linear Regression	0.02788	0.1147144	0.000649198
5	Random Forest	0.01797	0.1139732	0.002868967
6	Random Forest	0.01792	0.1136208	0.002894019
7	Boosted Tree	0.01960	0.1123863	0.001211790

Comparison of Models



The above visual illustrates that, in reality, all the random forest models performed quite similarly, especially compared to the other two types of models, the linear regression model (#4) and the boosted decision tree model (#7). The mean squared error (MSE) of our two best-performing models, #2 and #3, differed by 0.00001, which is extremely minor; the other three random forest models were not far behind at all.

The poor performance of the linear regression model is unsurprising because there are no tunable hyperparameters, and the regression method itself is a simple, high-bias low-variance model. Generally, it sacrifices predictive power in exchange for easier interpretability. Meanwhile, the boosted tree model performed only marginally worse than the random forest models, its MSE falling behind by about 0.002.

Overall, at least in our attempts and for this particular data set, the random forest model seems most befitting for regression modeling.

Final Model

In selecting our final model, our team opted to choose between the two models submitted onto Kaggle with the best public MSE scores – which were Candidate Models #2 and #3 with MSEs of 0.01764 and 0.01763 respectively. When the private testing results were revealed on Kaggle, both Candidate Models #2 and #3 were still our best performing models, with Model #3 performing marginally better with an MSE value of 0.01635 (thus making it our final model).

We believe that this model performed the best out of our models due to our strategic balancing of two crucial ideas: creating a thorough model that can account for (and thus accurately predict the response to) the many independent variables included in the data and their potential values and managing the flexibility of the model to ensure that we do not overfit the training data and can, as a result, accurately predict any testing data after the creation of the model. This was accomplished through adjusting the hyperparameters of the model when setting it as a random forest model. When initializing the model, we set mtry to 15, trees to 1500, and min_n to 25 – all to try to simultaneously accomplish both goals. Additionally, when setting the recipe for our model and workflow, we opted to use a basic recipe that included every independent variable in an attempt to be amenable to and flexible enough for any additional data that would be used to test the model. These two choices resulted in a model that appropriately fit testing data and had no signs of overfitting the data.

Our final model's key weakness lies in the fact that we were wary of potentially overfitting the data. Our model, overall, is fairly simple: we included every variable provided in the dataset and only adjusted the hyperparameters of the random forest model. It is highly plausible that the cautious selections we made in creating the model and its recipe were too vigilant and may have caused us to not create as precise of a model as we possibly could have made.

Evidently, there is room for improvement in the model as our final model placed twelfth in the Kaggle competition on the private leaderboard. We believe that these adjustments can be made in three potential ways. First: we can tune the hyperparameters of some of our independent variables; no tuning was done on any of the independent variables, which reduced the chances of overfitting the training data but, consequently, came at the cost of reducing our model's precision. Second: we could further adjust our hyperparameters of the random forest model itself and could consider removing some potential independent variables (for which our reasoning is articulated in our weaknesses paragraph). Third: we could potentially make a completely different model with a different engine. Our team opted to focus mostly on different random forest models as well as a boosted trees model and a general linear model. It is plausible that these three types of models were not the best option for the data we were given and, thus, potentially resulted in decent models but not one that precisely fit the testing data.

Appendix: Final Annotated Script

```
library(tidyverse)
library(tidymodels)

train <- read_csv("train.csv")
train <- train %>% select(-"order_totals")

rf_model <- rand_forest(mtry = 15, trees = 1500, min_n = 25) %>%
  set_engine("ranger") %>%
  set_mode("regression")

rf_recipe <- recipe(log_total ~ ., data = train)

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

rf_workflow_fit <- rf_workflow %>%
  fit(data = train) # This fitted workflow has an MSE of about
0.01288 and an R-squared value of about 95.89%

test <- read_csv("test.csv")

rf_predictions <- predict(rf_workflow_fit,
                          new_data = test)

rf_results <- test %>%
  select(id) %>%
  bind_cols(rf_predictions)

colnames(rf_results) <- c("id", "log_total")

write_csv(rf_results, file = "team_1_submission.csv")
# In Kaggle, this prediction set was labeled as
"rf_submission_3.csv"
```

Appendix: Team Member Contributions

- Max
    - Created Kaggle teams and outlined project reports
    - Submitted sixteen models/predictions to Kaggle, one of which tied with the best-performing model on the private leaderboard
    - Created and wrote descriptions/comparisons for a random forest model, a linear regression model, and a boosted decision tree model
    - Spent a lot of time tweaking the models, recipes, and hyperparameter tuning
    - Made the two tables and filled half the info in them
    - Made exploratory data analysis visualizations 1-3 and 8
    - Wrote most of the preprocessing section
- Bella
    - Submitted fifteen total sets of predictions to Kaggle (including one of our team's best performing model on the private leaderboard)
    - Wrote the description for Candidate Model #3 and included its information in the two Candidate Model tables
    - Wrote the Final Model section and the corresponding Final Annotated Script section
        - Submitted the Final Script on Canvas
    - Edited general aesthetics of the report
- Tran
    - Wrote the Introduction and the Preprocessing
    - Submitted Kaggle predictions
    - Wrote one of the descriptions for Candidate Model
- Vanda
    - Submitted on Kaggle
    - EDA viz #4 - 6
    - Wrote one candidate model
- Nicole
    - Submitted some kaggle preds
    - Made visualization 7
    - Wrote about my candidate model