Group 9 Regression Report

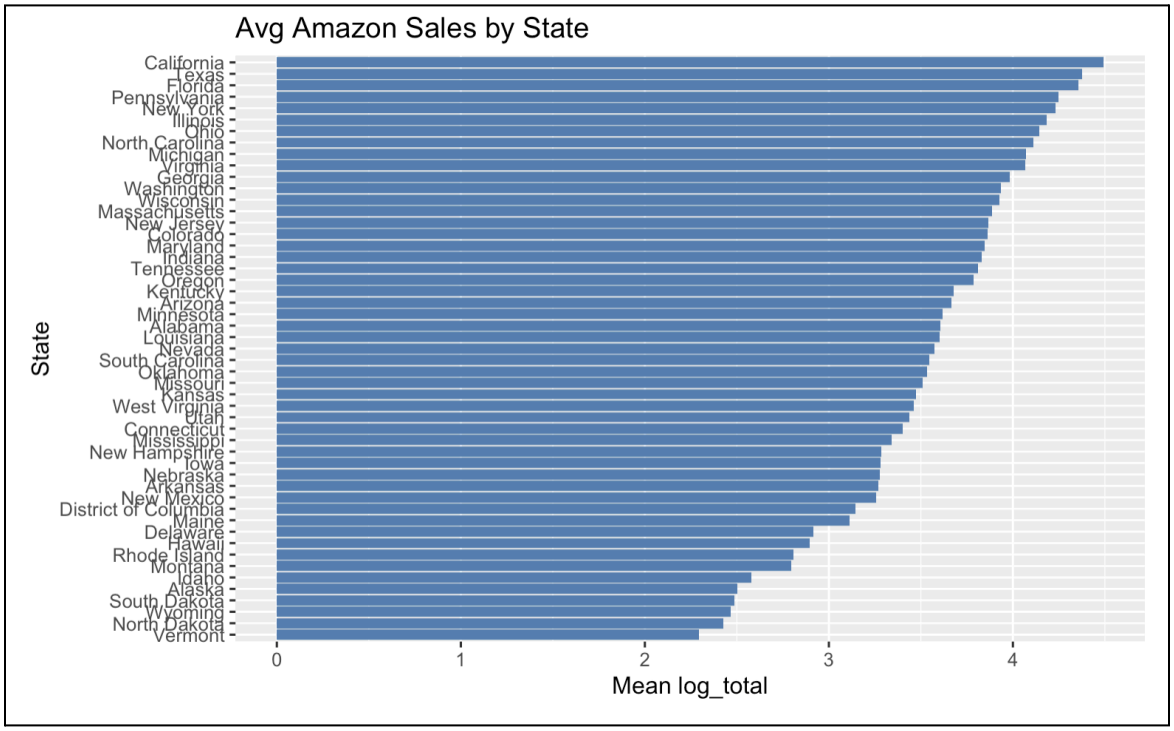
Introduction

This study investigates Amazon customer survey data (2018–2022) to model monthly state-level order totals. Prior research highlights that purchasing behavior is shaped not only by product categories such as clothing and electronics but also by demographic and behavioral factors (Sheriff & Paul, 2022)[1]. In particular, Amazon's strongest customer base consists of individuals aged 21–50 due to higher technological awareness, with students, employees, and business owners representing the majority of buyers. These findings motivated our initial data exploration, where we examined how age, income, household size, education, order frequency, account sharing, geographic location, and seasonality might be associated with variation in purchasing outcomes.

[1] Sheriff, M. I., & Paul, M. J. (2022). A study on consumer buying behaviour with preference to Amazon. *International Journal of Science and Research (IJSR), 11*(5), 481–485. https://doi.org/10.21275/MR22506143227

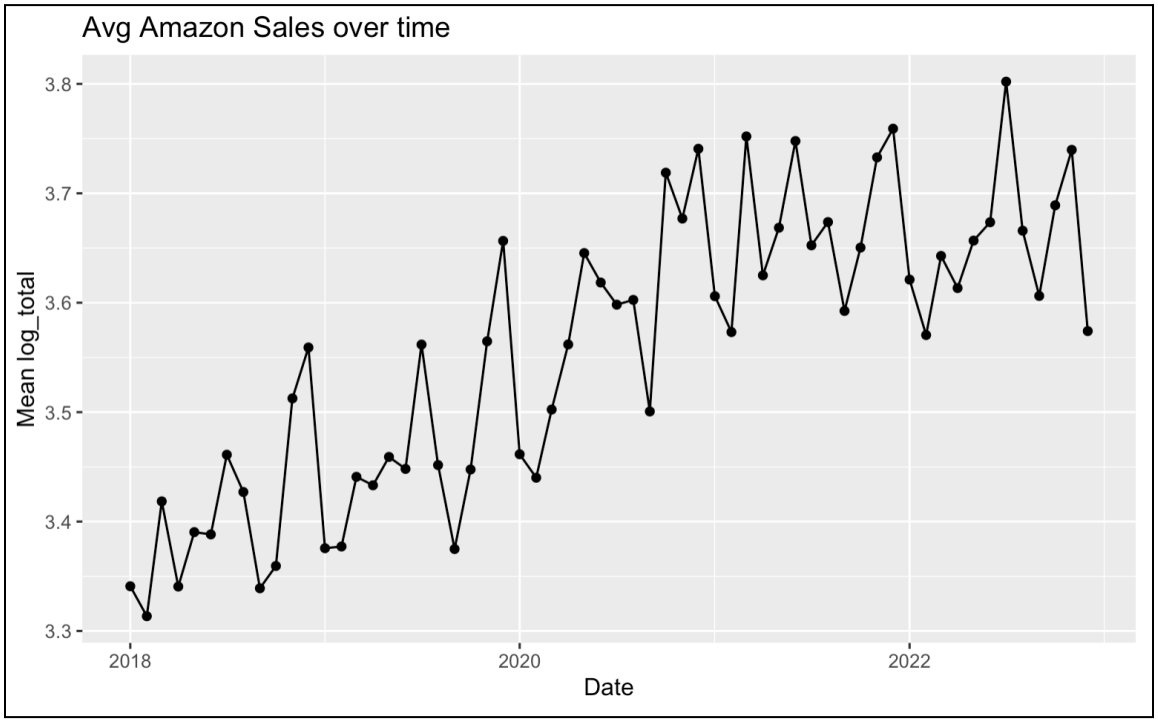
Exploratory Data Analysis

1. Bar Chart of Average Sales for Each State



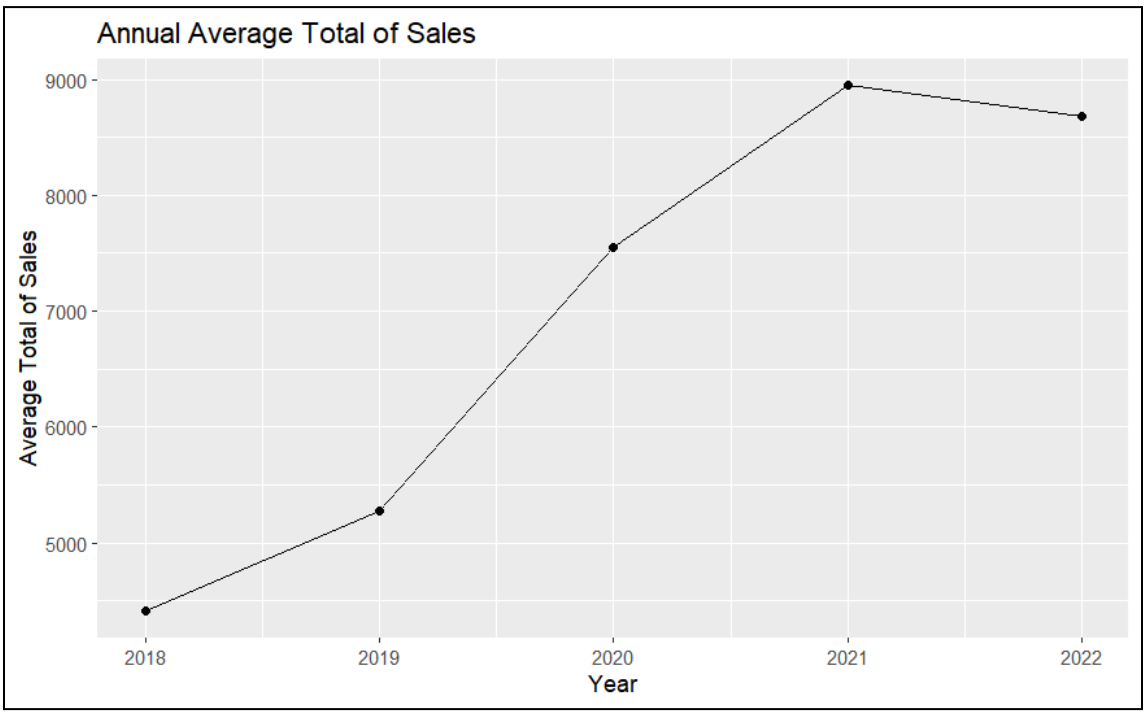
The bar chart above shows the relationship between the different states and the mean `log_total` (amount of sales) across all years. After ordering the different states from the largest average `log_total` to the smallest, we can see there is a clear relationship between the number of sales and which state we are observing. All of the states have roughly the same number of observations, so this would imply that some states spend more than other states. This suggests that making states a categorical variable may improve a model's predictive capabilities.

2. Line Chart of Average Sales Over Time



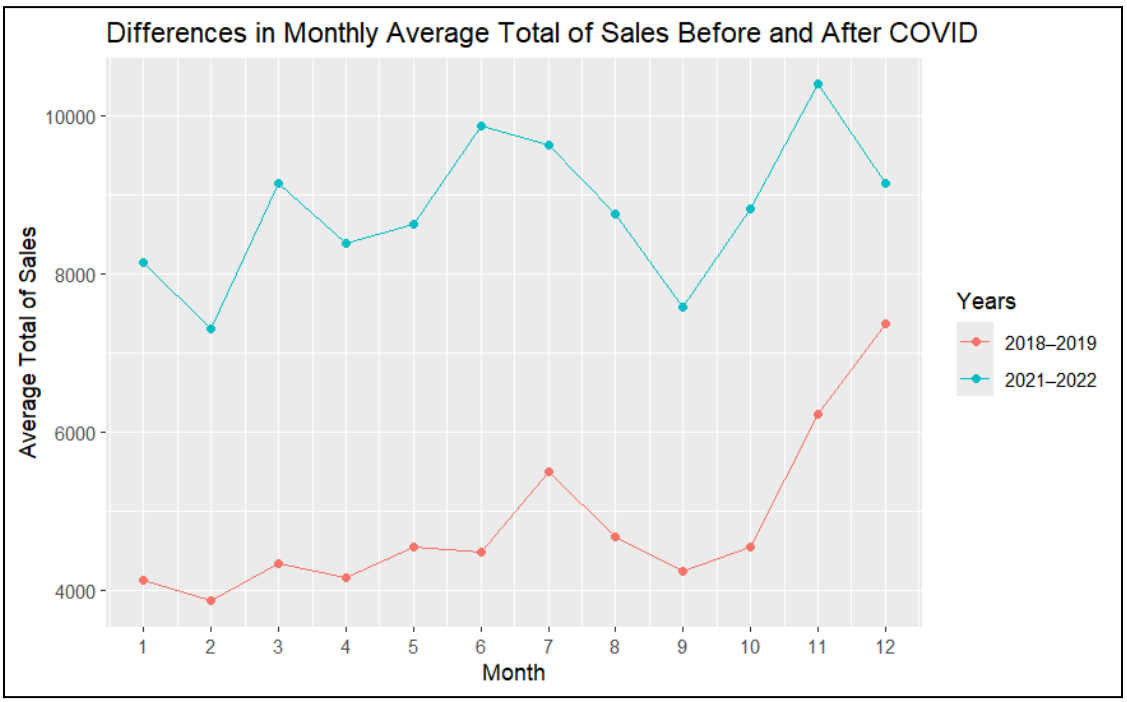
The line chart above models the mean `log_total` against time. We can see a general upwards trend of the `log_total`; however, more interestingly, we can see a somewhat cyclical pattern in the rise and fall of sales. This implies the possibility of the month playing an important role in predicting Amazon sales. By modelling seasonality into the model, this may capture the variation that is seen in the data.

3. Line Chart of Annual Average Sales Over Time



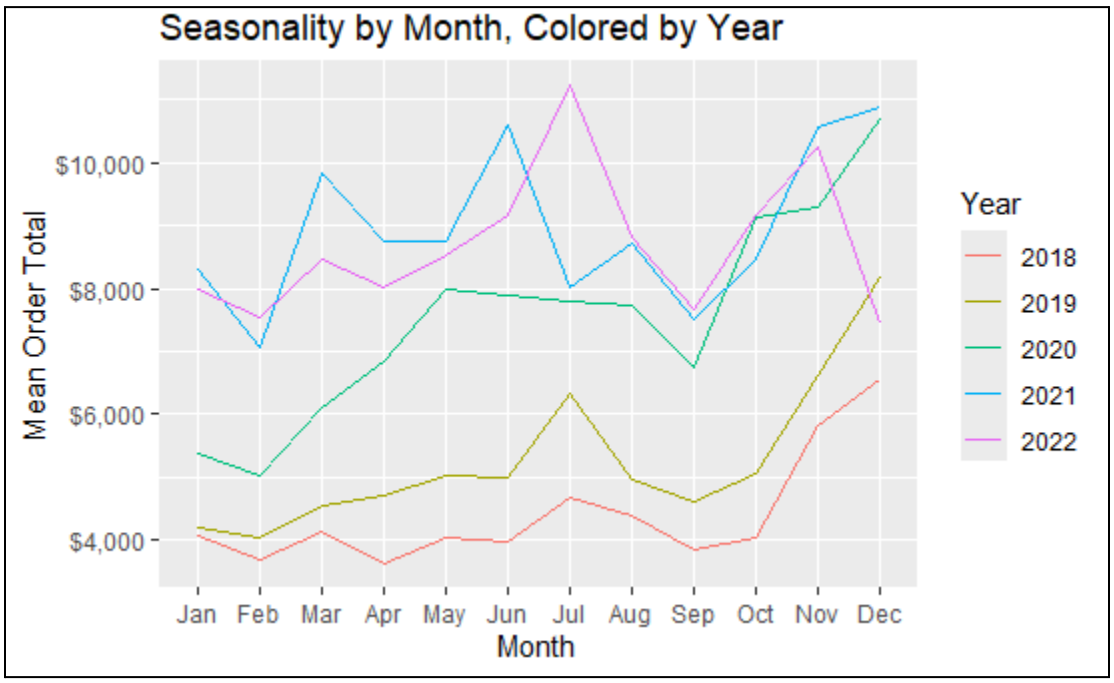
This line chart models the annual average sales against time. This chart is similar to Data Visualization 2, but it uses `order_totals` instead of the `log_totals` to emphasize the growth each year of the average total of sales. There is a noticeable increase in sales between 2019 and 2020 and 2020 and 2021, which we speculate could be attributed to the COVID-19 pandemic preventing people from physically shopping in stores and encouraging online shopping. It might be worth investigating further the influence of the COVID-19 pandemic on sales.

4. Line Chart of Monthly Average Sales Before and After COVID-19



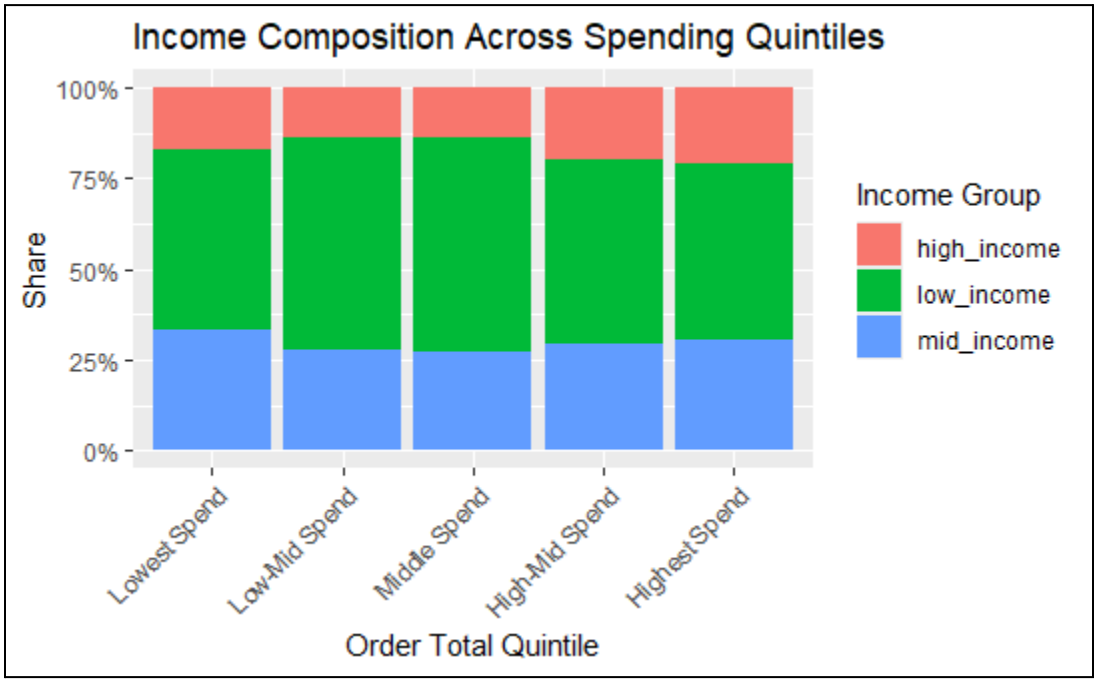
This line chart is an extension of Data Visualization 3 as it depicts the average monthly sales before and after COVID. The year 2020 is omitted from this graph because 2020 is used as the boundary between the pre-COVID and post-COVID time periods. The line chart visualizes how the average monthly sales have increased overall, but the months with the highest averages have changed between the two periods. This further emphasizes that it may be useful to explore whether grouping the data in 2018 and 2019 together and the data in 2021 and 2022 together may help our model better predict the overall trend.

5. Seasonality by Month, 2018-2022



Similar to Data Visualization 4, this line chart illustrates the average `order_totals` across all five years without grouping by year, allowing us to observe raw trends within each fiscal year. The data reveals consistent seasonal patterns, with recurring monthly peaks typically occurring from April to June and again from September to November. However, the 2022 year-end data deviates noticeably from the established trends observed between 2018 and 2021. As a result, when modeling seasonality for the 2018-2021 period, Month indicators alone may suffice as seasonal controls. In contrast, incorporating 2022 into the model likely requires Year fixed effects to control for overall differences across years.

6. Income Composition Across Spending Quintiles



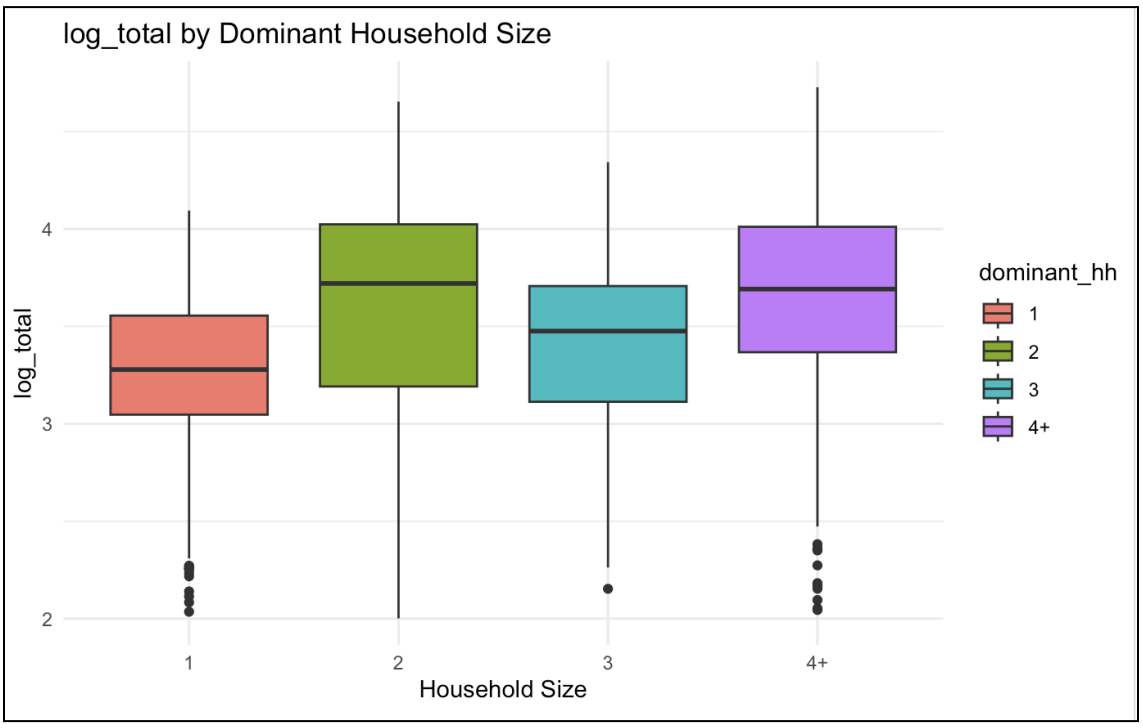
This graph examines spending patterns across income quintiles among a range of income groups and reveals a relatively stable distribution, with no pronounced increase in spending among higher-income households. This suggests that income, while influential, does not fully account for variation in total order volume, pointing to the potential role of other demographic or behavioral variables.

Income levels have been grouped into three buckets:

- Low income: $50,000 - $99,999
- Mid income: $100,000 - $149,999
- High income: $150,000+

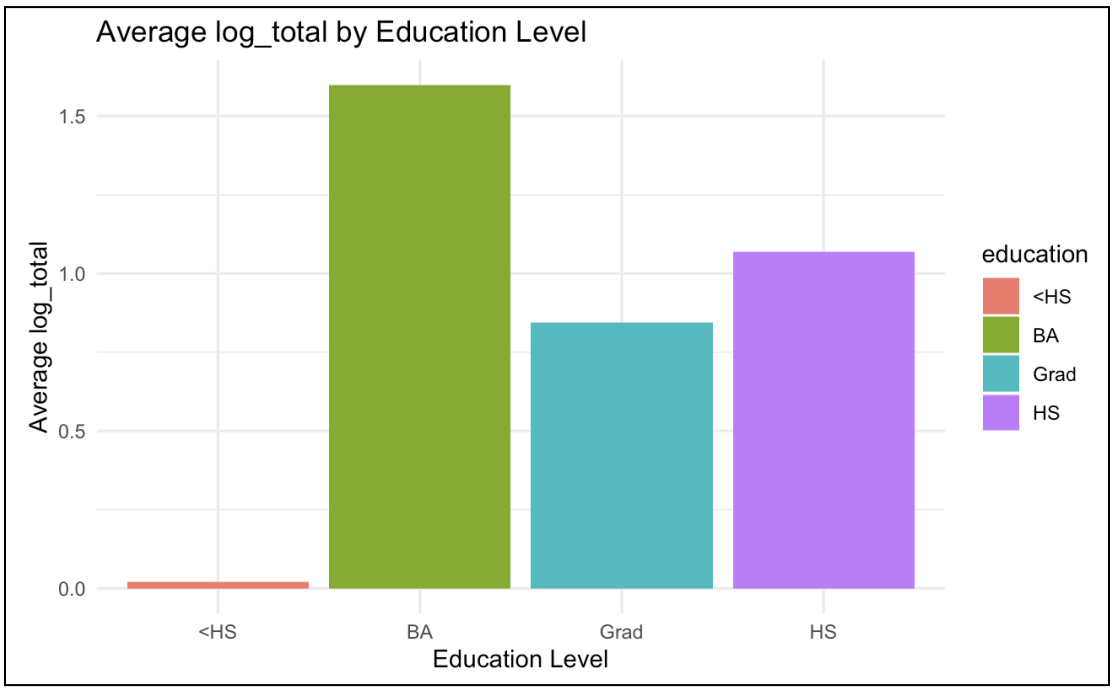
These groupings help isolate spending behavior across broader income categories and reinforce the observation that higher income does not necessarily correlate with higher order totals.

7. Spending Patterns by Household Size



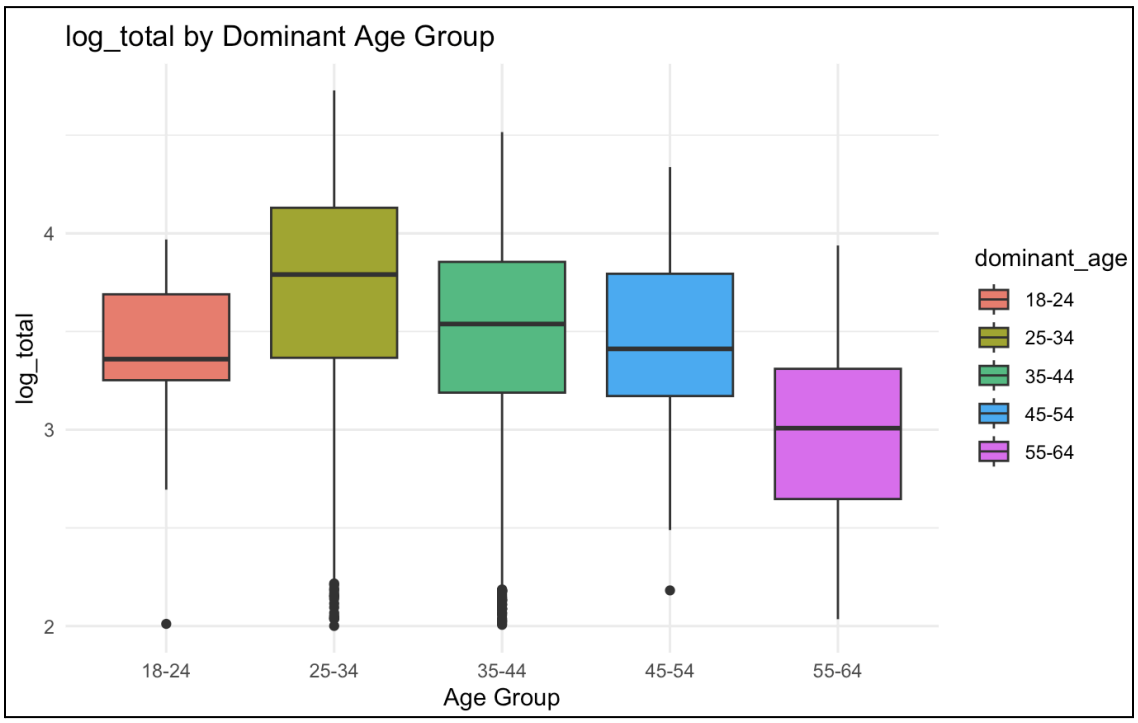
This boxplot shows how household size relates to Amazon spending, measured by log-transformed order totals. Single-person households (size 1) have the lowest median spending and smaller variability. Households with two or four-plus members show the highest spending, with both groups having higher medians and wider spreads. Three-person households spend more than single-person households but less than two- or four-person groups. Outliers appear across all categories, but more prominently in larger households. Overall, the data suggest that larger households tend to generate higher Amazon spending, likely because multiple members share the account and increase order frequency.

8. Average Spending by Education Level



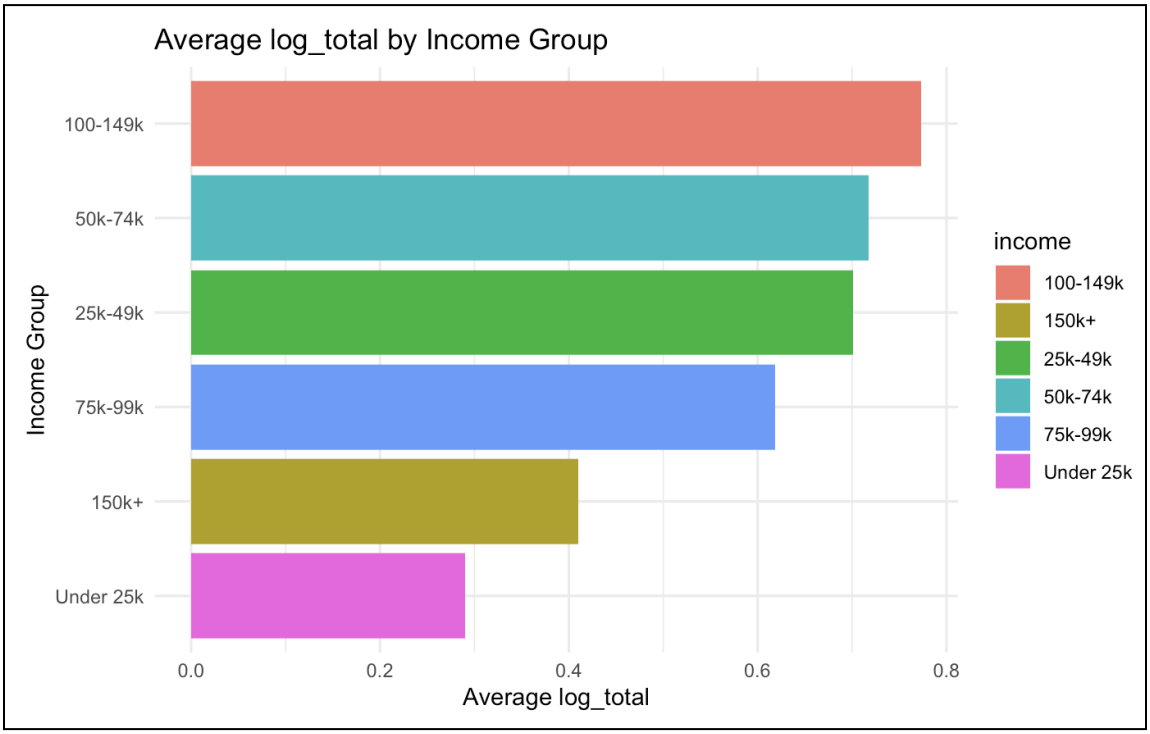
This bar chart compares average log-transformed order totals across four education levels. Customers with a Bachelor's degree (BA) have the highest average spending, significantly surpassing other groups. High school graduates follow, showing moderate levels of Amazon spending. Graduate or professional degree holders spend less than BA holders and high school graduates, which may suggest different purchasing preferences. Customers with less than a high school education (<HS) spend the least, with nearly negligible log_totals on average. The chart highlights that education is linked to consumer behavior on Amazon, with BA-level customers standing out as the most frequent or highest spenders.

9. Spending Distribution by Age Group



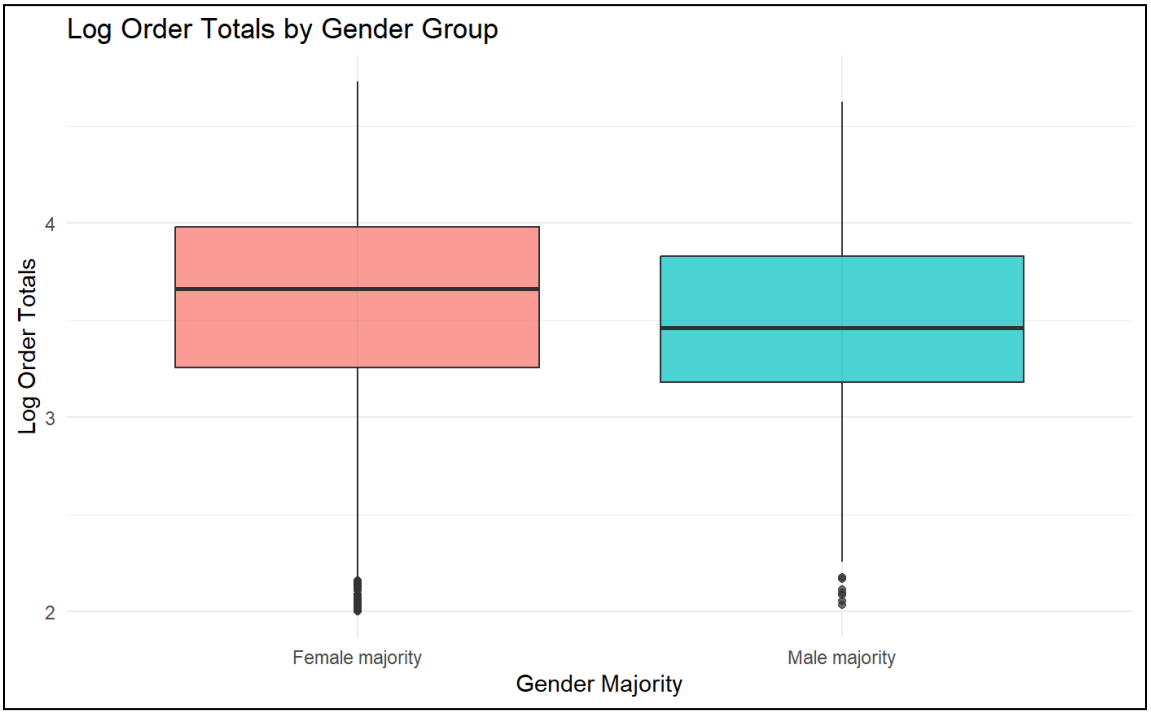
This boxplot examines Amazon spending across age groups. The 25–34 age group has the highest median `log_total` and the widest variability, suggesting they are the platform's most active and diverse spenders. Spending decreases with age, with the 35–44 and 45–54 groups spending moderately, while the 55–64 group spends the least. Younger adults (18–24) spend more than older groups but less than 25–34. Outliers are visible across categories, particularly in younger groups, reflecting a mix of light and heavy buyers. Overall, the chart indicates that Amazon spending peaks in early adulthood and declines steadily with age.

10. Average Spending by Income Group



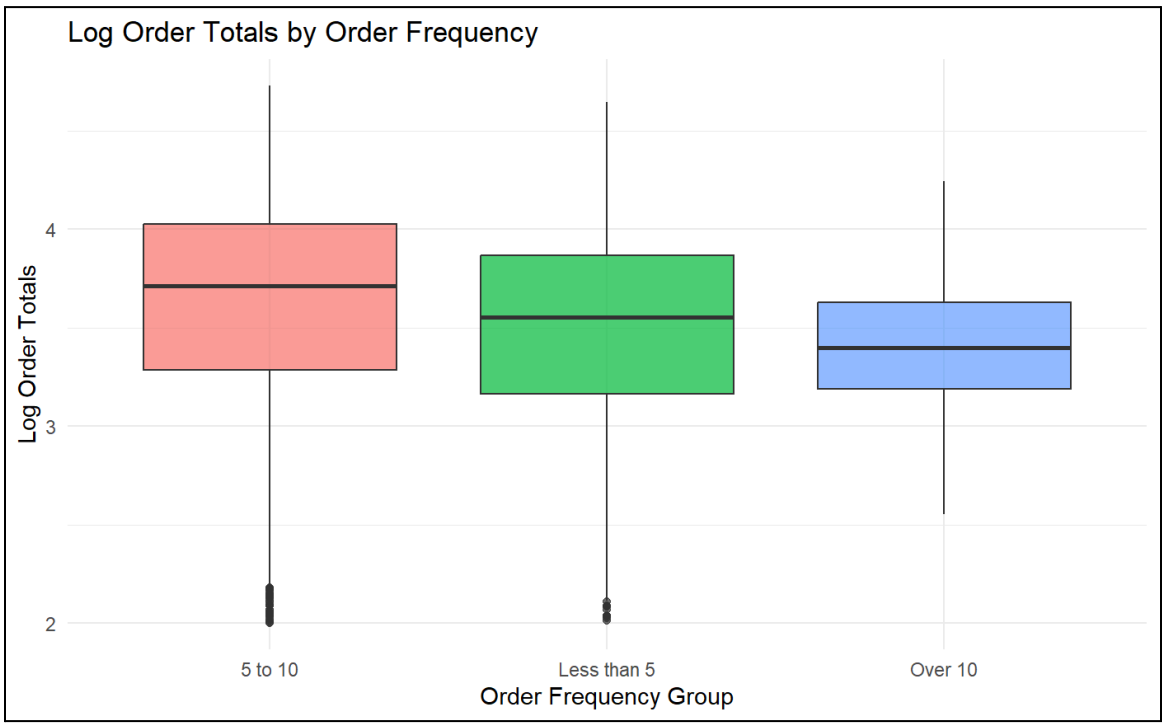
This bar chart illustrates how Amazon spending varies by income bracket. Customers earning between $100k–149k spend the most on average, followed by those in the $50k–74k and $25k–49k brackets. Interestingly, spending drops for the highest earners ($150k+), suggesting they may rely less on Amazon for purchases or diversify shopping elsewhere. The lowest income group, under $25k, predictably has the least spending. Middle-to-upper income brackets appear to be Amazon's strongest customer base, reflecting both purchasing power and reliance on the platform. The chart highlights a non-linear relationship, where middle-income groups can outspend even the wealthiest customers.

11. Boxplot of Spending Differences by Gender



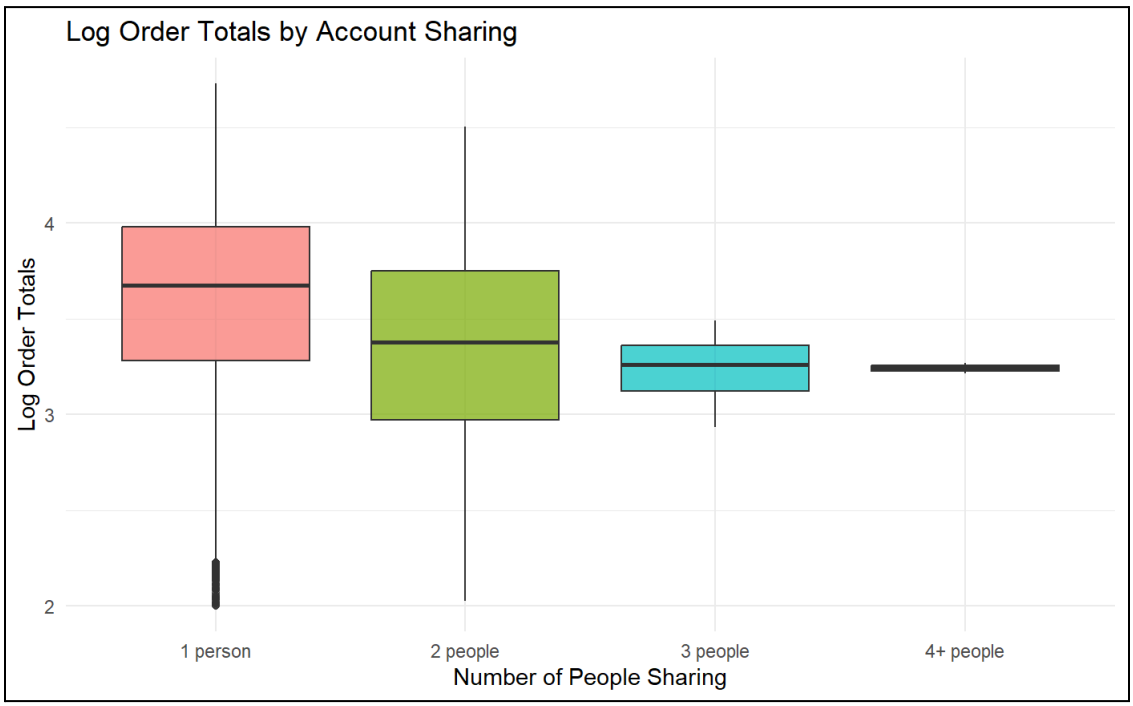
This boxplot illustrates plausible differences in spending patterns based on the category of gender. It was plotted based on whether a given row observation had a majority of female responders or male responders, as derived from the count_female and count_male variables. The variability of the female majority group is greater than that of the male majority group. Additionally, the `log_total` medians of the two groups are different, indicating possibly significant between-groups variation. However, the overall distributions of the two groups are very similar and may indicate that gender is not a significant enough predictor of `log_total`.

12. Boxplot of Spending Differences by Order Frequency



This boxplot compares spending patterns according to order frequency proportions. Each row observation was categorized based on the highest proportion of response counts that fell into one of the three order frequency groups. While each group has different median order totals, the boxplots appear to follow fairly similar distributions. Moderate-frequency shoppers (5 to 10) appear to produce the highest order totals. High-frequency (Over 10) shoppers could be habitual buyers who tend to have fewer items in their baskets at checkout. Low-frequency (Less than 5) shoppers have wide variability. These patterns suggest that order frequency is not linearly associated with order totals.

13. Boxplot of Spending Differences by Account Sharing



This boxplot compares spending patterns according to account sharing proportions in the dataset. Each row observation was categorized based on the highest proportion of response counts that fell into one of the four account-sharing groups. Because there were so few row observations whose greatest proportion in account sharing counts was responses of people who share with 4 or more people, we are unable to observe the variability of the 4+ people group in this plot. When comparing the other three groups, we can see that variability decreases as the number of people sharing increases. The median spending also decreases as the number of people sharing increases. There could be a moderate linear relationship between these two variables.

Preprocessing / Recipes

Linear Regression Recipe

```
lm_recipe <-
  recipe(log_total ~., data = train_data) |>
  step_rm(count_female, count_less5, count_hh1, count_howmany1, count_1824, count_und25k, count_lessHS) |>
  step_ns(year, deg_free = 3)
```

step_rm: For the linear regression model, we removed the variables `count_female`, `count_less5`, `count_hh1`, `count_howmany1`, `count_1824`, `count_und25k`, and `count_lessHS`. Since each category of variables (i.e., household size variables, customer income variables, etc.) sums up to the total `count`, if we included all the variables available, there would be perfect collinearity. Thus, we had to remove one variable from each category to ensure the model could still be fitted.

step_ns: The recipe also adds splines to the `year` variable. Since each year results in quite a different average `log_total` due to various reasons (such as Amazon performance or COVID-19), we hypothesized that it would not be an exact linear relationship and that adding splines to the year might improve the model.

The states were automatically converted to dummy variables.

Random Forest Recipe

```
rf_recipe <-
  recipe(log_total ~ ., data = train_data) |>
  step_mutate(
  month_sin = sin(2 * pi * month / 12),
  month_cos = cos(2 * pi * month / 12)) |>
  step_rm(month) |>
  step_interact(terms = ~ count_2534:count_100149k +
                  count_4554:count_150kup +
                  count_hh4:count_over10 +
                  count_hh1:count_less5)
```

step_mutate: Since months are cyclical in nature, if it is left just as a numeric variable, then the month of December (12) will not be taken to be next to January (1), and will only be taken to be a linear increasing relationship. To correct for this, we create two new variables that will account for the cyclical nature of the months.

step_rm: Since we have created new variables to model the months, we must remove the redundant `month` variable.

step_interact: We included some interaction variables in the hopes that they would result in improving the model's predictive performance. Due to the large number of possible interaction variables, the interacted terms were chosen based on rational choices and what made sense in reality. For example, we employed an interaction term between `count_2534` and `count_100149k` to model young working adults who have high annual income, as this group may spend more money on Amazon purchases. Another example would be `count_hh4` and `count_over10`, since large households that make more than 10 orders per month could indicate high spenders. These types of interactions were slowly added until the model did not improve in performance.

The states were automatically converted to dummy variables.

Boosted Tree Recipe (xgboost)

```
xgb_recipe <-
  recipe(log_total ~ ., data = train_data) |>
  step_mutate(
  month_sin = sin(2 * pi * month / 12),
  month_cos = cos(2 * pi * month / 12)) |>
  step_rm(month) |>
  step_interact(terms = ~ count_2534:count_100149k +
                  count_4554:count_150kup +
                  count_hh4:count_over10 +
                  count_hh1:count_less5) |>
  step_dummy(q_demos_state)
```

Most of this recipe is the same as the random forest recipe. The only addition is the step_dummy, since, unlike ranger, xgboost does not automatically create dummy variables. Thus, we had to manually change the states to dummy variables.

Boosted Tree Recipe (lightgbm)

```
lgbm_recipe <-
  recipe(log_total ~ ., data = train_data) |>
  step_mutate(
  month_sin = sin(2 * pi * month / 12),
  month_cos = cos(2 * pi * month / 12)) |>
  step_rm(month) |>
  step_dummy(q_demos_state)
```

Most of this recipe is the same as the xgboost boosted tree recipe, since lightgbm also does not automatically create dummy variables. We also decided to remove step_interact to see if it would improve the performance of the model by reducing the number of variables, and in this case, it did have an improvement.

Candidate Models

We created various models to predict the `log_total` of the test set. The following 5 models are the main models that were created:

1. lm_M1 (Multiple Linear Regression)
    a. We attempted a simple linear regression model to test as a base for predictive performance. There was no tuning required for this model since there are no hyperparameters.
2. rf_M4_monthinter_2 (Random Forest)
    a. Random Forest proved to give the best performance. Thus, there were several iterations of this model that were created, with different recipes and tuning ranges. The final chosen model is the one that is listed in this report.
3. xgb_M1 (Boosted Tree)
    a. A boosted tree model was attempted with the `xgboost` engine. A recipe similar to the random forest model was used for the boosted tree model; however, it resulted in performance that was worse than the random forest model.
4. lgbm_M1 (Boosted Tree)
    a. A different boosted tree model was also attempted with the `lightgbm` engine. This engine ran through the different cross-validation iterations much quicker than `xgboost`, allowing us to try different combinations of hyperparameters and search grids. However, it still resulted in performance that was worse than the random forest model.
5. stack_M1 (Stacked Model)
    a. This model is a stacked model that combines all of the previously created models together. We attempted to use this to improve the final model. However, it also resulted in performance that was worse than the random forest model. A possible explanation for this low performance is due to the performance of the other models being poor.

Attempted Candidate Models

Model Identifier	Type of Model	Engine	Recipe Used	Hyperparameters
lm_M1	Multiple Linear Regression	lm	lm_recipe	NIL
rf_M4_monthinter_2	Random Forest	ranger	rf_recipe	`trees`: 800 `mtry`: 5 `min_n`: 12
xgb_M1	Boosted Tree	xgboost	xgb_recipe	`trees`: 800 `mtry`: 60 `min_n`: 17 `tree_depth`: 4 `learn_rate`: 1.118203 `loss_reduction`: 1.535224 `sample_size`: 0.7897973
lgbm_M1	Boosted Tree	lightbgm	lgbm_recipe	`trees`: 2000 `mtry`: 48 `min_n`: 43 `tree_depth`: 5 `learn_rate`: 1.136642 `loss_reduction`: 1.038899 `sample_size`: 0.6128111 `num_leaves`: 36
stack_M1	Stacked	Multiple	Multiple	`penalty`: 0.001

Model Evaluation and Tuning

The different models' performances were evaluated based on their overall RMSE after a 10-fold cross-validation. All models were tested using the same fold (that was created using seed 101). Each model was tuned, with different grid types being used depending on the model, since different engines had differences in how computationally expensive they were. For example, the xgboost model is computationally expensive, so we chose to tune using a small random grid and use `tune_bayes` to potentially improve the model using iterative search. Unfortunately, there was no success in improving the model any further. On the other hand, the lightgbm model had fast computations, so we could afford to tune it on a larger grid.

Different models also had different hyperparameters that required tuning. The following hyperparameters and ranges that they were tuned on are listed below:

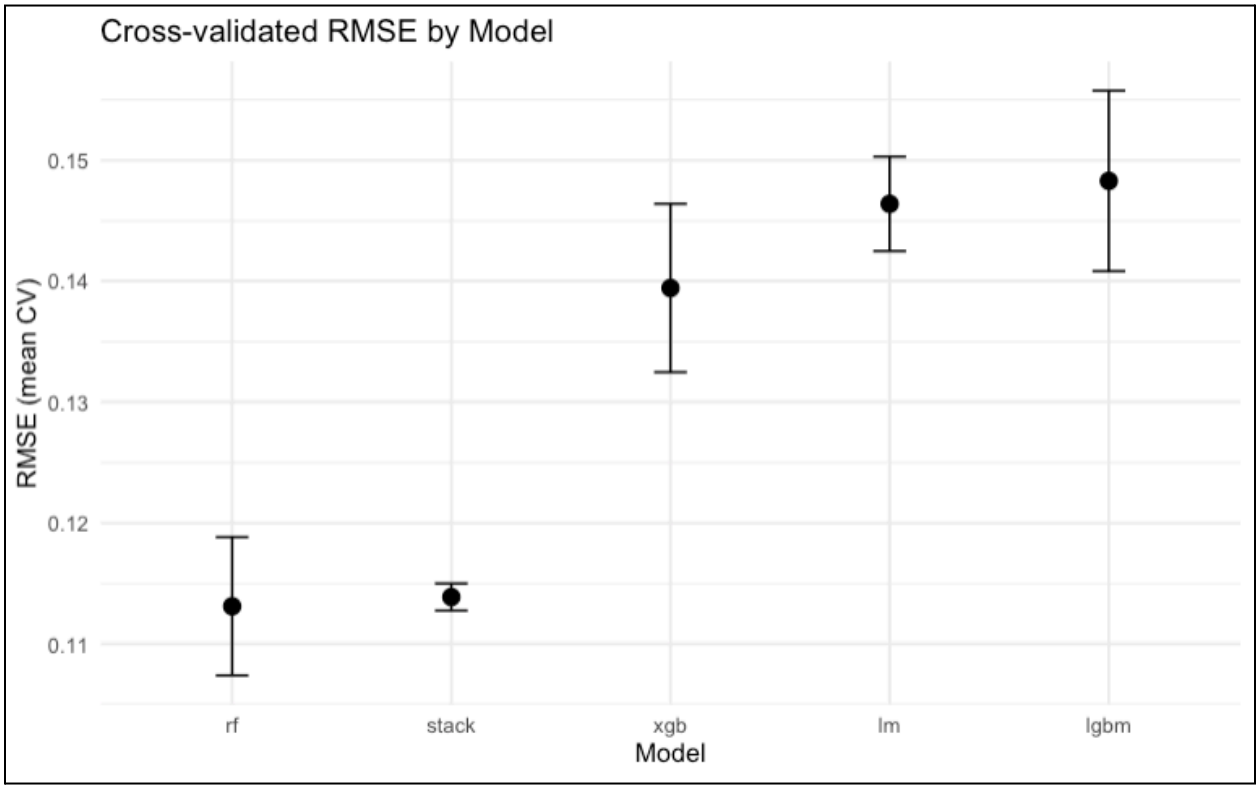
Tuning Details for Each Model

1. lm_M1 (Multiple Linear Regression)
    a. No tuning required since there are no hyperparameters for this model.
2. rf_M4_monthinter_2 (Random Forest)
    a. Tuned on a regular grid with 5 levels
    b. `trees` was set at 800
    c. `mtry` was tuned from a range of 1 to 20
    d. `min_n` was tuned from a range of 5 to 20
3. xgb_M1 (Boosted Tree)
    a. Tuned on a random grid with a size of 20
    b. `trees` was set at 800
    c. `mtry` was tuned from a range of 0.3 to 0.8, based on the proportion of predictors
    d. `tree_depth` was tuned from a range of 3 to 8
    e. `learn_rate` was tuned from a range of 0.01 to 0.1
    f. `loss_reduction` was tuned from a range of 0 to 5
    g. `min_n` was tuned from a range of 5 to 20
    h. `sample_size` was tuned from a range of 0.6 to 1
4. lgbm_M1 (Boosted Tree)
    a. Tuned on a latin hypercube grid with a size of 40
    b. `trees` was set at 2000
    c. `mtry` was tuned from a range of 0.3 to 0.8, based on the proportion of predictors
    d. `tree_depth` was tuned from a range of 3 to 8
    e. `learn_rate` was tuned from a range of 0.01 to 0.06
    f. `loss_reduction` was tuned from a range of 0 to 5
    g. `min_n` was tuned from a range of 10 to 100
    h. `sample_size` was tuned from a range of 0.6 to 0.95
    i. `num_leaves` was tuned from a range of 31 to 255
5. stack_M1 (Stacked Model)
    a. No tuning was required. The penalty is automatically tested.

Model Performance

Model Identifier	Metric Score (RMSE)	SE of Metric
lm_M1	0.1463834	0.001991862
rf_M4_monthinter_2	0.1131197	0.002917837
xgb_M1	0.1394274	0.003551220
lgbm_M1	0.1482810	0.003806781
stack_M1	0.1138850	0.0005701876

Model Comparison



Based on the model performance metric (RMSE), and autoplot comparing the models, we see that the best model is the random forest model (rf_M4_monthinter_2) which has the smallest RMSE score (0.1131197). It had also yielded the best overall score when input into the Kaggle competition site. Thus, we have chosen this as our final model.

Discussion of Final Model

Final Model Selection

Our final model was `rf_M4_monthinter_2`, a Random Forest regression model implemented with the `ranger` engine in `tidymodels`. The model was tuned on a regular grid with 5 levels, where the hyperparameters `mtry` (1-20) and `min_n` (5-20) were optimized, while the number of trees was fixed at 800.

The choice of Random Forest was motivated by its robustness and ability to handle non-linear relationships and interactions without requiring heavy manual feature engineering. Our recipe included preprocessing steps such as converting `month` into cyclical features and adding meaningful interaction terms between demographic and household variables. These additions allowed the model to capture seasonality and nuanced demographic effects in a way that simple linear models could not.

Strengths of the Model

Our Random Forest model captures interactions naturally, and we reinforced this by explicitly creating interaction terms in the recipe. Each preprocessing step (month transformation, interaction terms) was aligned with background knowledge about the seasonality of sales, as determined through our data exploration. The nature of Random Forest makes it less prone to overfitting compared to single decision trees, especially with a relatively large number of trees (800). The combination of preprocessing and Random Forest's flexibility likely explains our jump in the leaderboard placement from last to 9th place.

Weaknesses of the Model

While our preprocessing steps and use of a Random Forest model were effective, our approach lacked the sophistication of stacked models that combine multiple well-performing models. Other groups likely leveraged multiple single models to create an advanced stacked model. Although `month` was encoded cyclically, other time-related patterns may not have been fully captured in our model as well. We also did not experiment with alternative variable transformations, such as including grouped categorical variables in our model.

Possible Improvements

Our model could have benefited from additional variable tuning, such as in our submission `rf_M4_monthinter_2.1` that ended up performing better in the private competition than our best model in the public competition. We also could have organized model stacking if we had found additional well-performing singular models. Extra time could have been spent testing different models that could eventually go into an advanced stacked model. We could also have examined and reengineered our variables through binning, transformations, or grouping to improve predictive power. While we attempted to capture monthly seasonality, we could have made better use of our time variables by introducing other trend components. Additional interaction terms may help capture the joint effects of variables that influence purchasing behavior in ways a single feature can't explain, depending on the results of our feature selection.

Appendix: Final Annotated Script

Script for Final Model (rf_M4_monthinter_2)

```
library(tidyverse)
library(tidymodels)

# Load data
test_data <- read.csv("test.csv")
train_data <- read.csv("train.csv")
train_data <- train_data |> select(-order_totals)

# Create folds
set.seed(101)
cv_folds <- vfold_cv(train_data, v = 10, strata = log_total)

# 1. Recipes (Pre-processing)
rf_recipe <-
  recipe(log_total ~ ., data = train_data) |>
  step_mutate(
  month_sin = sin(2 * pi * month / 12),
  month_cos = cos(2 * pi * month / 12)) |>
  step_rm(month) |>
  step_interact(terms = ~ count_2534:count_100149k +
                  count_4554:count_150kup +
                  count_hh4:count_over10 +
                  count_hh1:count_less5)

rf_prep <- prep(rf_recipe, training = train_data) # Check changes
view(bake(rf_prep, new_data = NULL))

# 2. Create model
rf_model <- rand_forest(mtry = tune(),
                        trees = 800,
                        min_n = tune()) |>
  set_mode("regression") |>
  set_engine("ranger")

# 3. Create workflow
rf_workflow <- workflow() |>
  add_model(rf_model) |>
  add_recipe(rf_recipe)
```

```
# 4. Tuning grid
rf_param_set <- parameters(rf_workflow) |>
  update(min_n = min_n(c(5, 20)),
         mtry = mtry(c(1, 20)))

rf_grid <- grid_regular(rf_param_set, levels = 5)

ctrl <- control_grid(
  save_pred      = TRUE,
  save_workflow  = TRUE,
  verbose        = TRUE)

metric <- metric_set(rmse)

rf_res <-
  tune_grid(rf_workflow,
            resamples = cv_folds,
            grid = rf_grid,
            metrics = metric,
            control = ctrl)

# 5. Checking best parameters
show_best(rf_res, metric = "rmse")
best_params <- select_best(rf_res, metric = "rmse")

# 6. Finalize workflow
final_wf <- finalize_workflow(rf_workflow, best_params)

# 7. Fit on full training
final_fit <- fit(final_wf, data = train_data)

# 8. Predict on test set
results <- cbind(test_data |> select(id), final_fit |>
predict(test_data))
results <- rename(results, log_total = .pred)

# 9. Create submission file
write_csv(results, "rf_M4_month+inter_2_final.csv")
```

Appendix: Team Member Contributions

John Tan

- Submitted 20 models to Kaggle
- Wrote Prereprocessing/Recipes, Candidate Models section, and Appendix (Final Annotated Script)
- Created data visualizations 1 and 2 for Exploratory Data Analysis
- Checked in on team progress, organized meetings

Jillian Dessing

- Created 5 models
- Submitted 2 models to Kaggle, one of them being one of our initial best models
- Wrote Introduction
- Created data visualizations 7, 8, 9, 10 for Exploratory Data Analysis
- Participated in team meetings as scheduled
- Helped read over and revise final report

Phoebe Chen

- Submitted 1 model to Kaggle
- Created data visualizations 11, 12, and 13 for Exploratory Data Analysis
- Edited report and its formatting
- Participated in team meetings as scheduled

Joseph Lukas

- Submitted 14 models to Kaggle
- Created data visualizations 3 and 4 for Exploratory Data Analysis
- Read over report to ensure clarity
- Participated in team meetings as scheduled

Saranna Lay

- Submitted 14 models to Kaggle
- Wrote Discussion of Final Model
- Created data visualizations 5 and 6 for Exploratory Data Analysis
- Participated in team meetings as scheduled