Orders Regression Report

Class : Statistics 101C

Group : Team 14

Members : Claire Nabours (206014740), Jackie Sung (806062422),

Naikang Wang (206791396), Yankai Wen (606792233）

Introduction

The online retail landscape is increasingly shaped by regional consumer trends, making predictive analytics vital for platforms like Amazon. This project utilizes a five-year survey dataset from approximately 5,000 customers to construct a regression model for predicting monthly, state-level sales totals. The central hypothesis is that variations in sales across regions are primarily influenced by the demographic profiles and purchasing patterns of the local consumer base. By analyzing the relationship between these variables and sales, this study seeks to identify the key drivers of consumer spending and provide insight into variations in market performance.

Exploratory Data Analysis

In this section we will explore the structure and quality of the dataset before modeling. We will examine the outcome variable (log_total), distributions of predictors, simple relationships, and interactions between variables from the train.csv file. These findings will motivate our transformations later used in the preprocessing section and help us to inform us of model specification when creating our models.

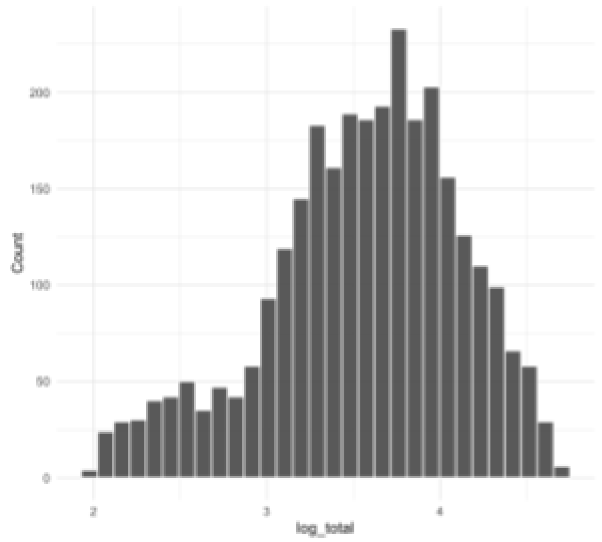


Figure 1: Distribution of Outcome Variables: Log Total

The histogram in Figure 1 displaying the distribution of Log_Total is unimodal and roughly symmetric with a slight left tail. Most observations fall between 3 and 4, peaking around 3.7. This shape suggests the log transform has stabilized the outcome reasonably well, so linear modeling on Log_Total is appropriate.

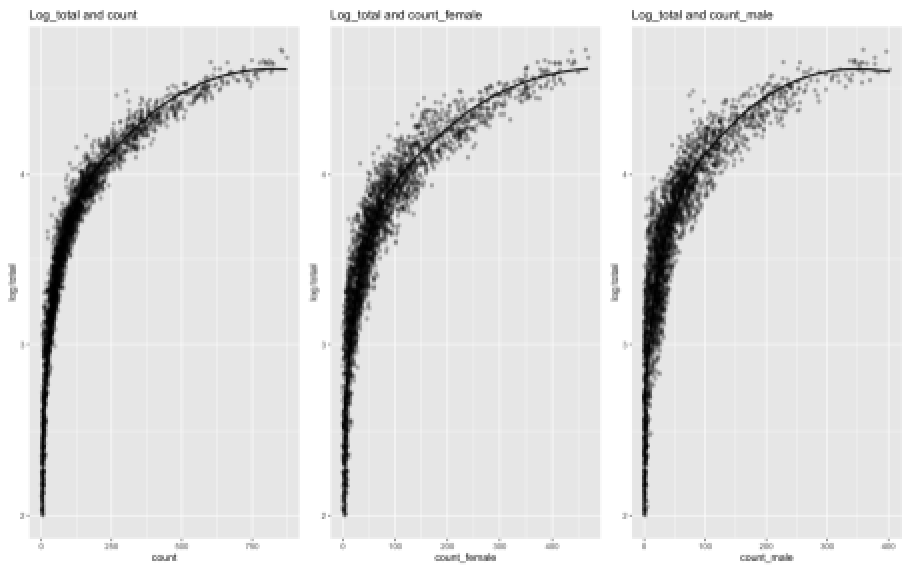


Figure 2: Relationship of count, count_female, and count_male with Log_total

Figure 2 shows a clear positive relationship between log_total and the variables count, count_female, and count_male. The trend lines rise quickly at low values and then level off as values increase. In fact, all count variables show the same curvature and the same overall pattern. This curvature indicates the possibility for the need for a log transform on the count features to make the relationship closer to linear and to stabilize the variance.

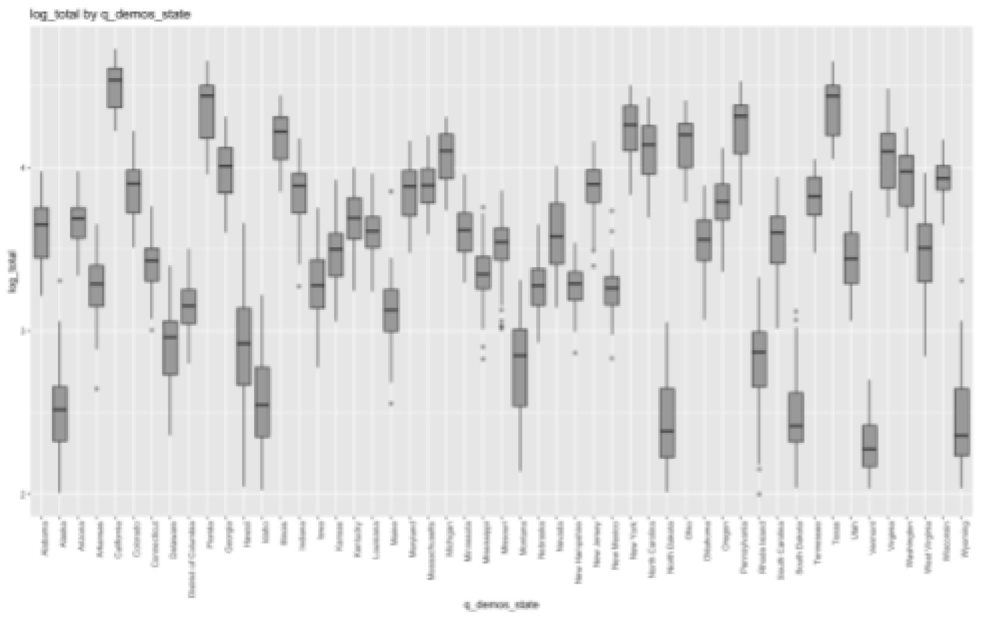


Figure 3: Relationship of log_total and state

Figure 3 demonstrates how log_total varies by state. Medians vary widely by state, showing a strong relationship between state and log_total . For example, larger population states like California tend to sit toward the higher end of log_total, which matches the idea that more people means more orders.

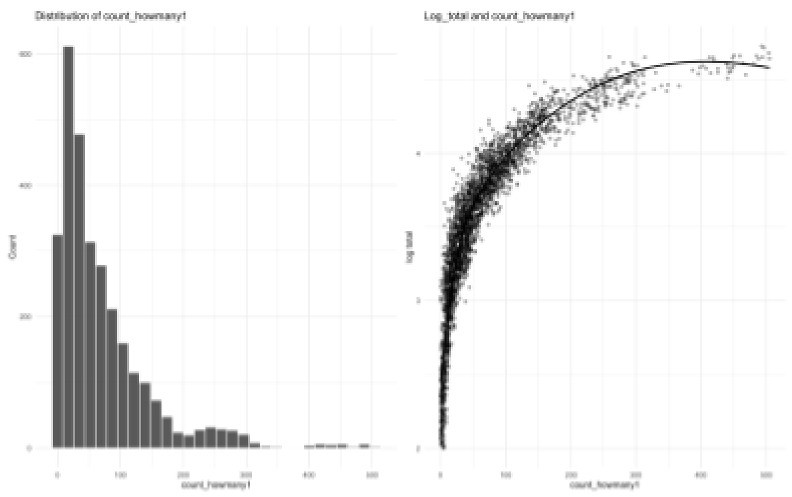


Figure 4: Distribution of count_how_many1 and scatter plot of count_how_many1 and log_total pre transformations

In figure 4, the left graph of the distribution of count_howmany1 is strongly right skewed with and there are a few very large values. Again, log_total rises fast at low counts and then levels off as counts grow, with slightly more spread at the high end. All of the count features show this same combo of right skew and a saturating relationship with log_total. To address both, we perform a log transform of the count variables, to compresses the tail, makes the trend closer to linear, and stabilizes the variance

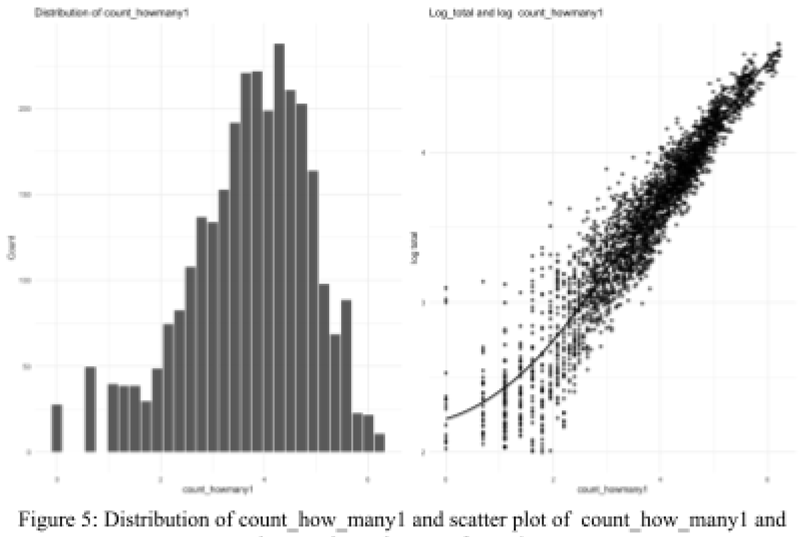


Figure 5: Distribution of count_how_many1 and scatter plot of count_how_many1 and log_total post log transformation

After logging the count feature, as seen in figure 5, the histogram is much more symmetric and the extreme right tail is compressed. In the scatter plot, log_total now tracks almost linearly with the logged count and the spread looks more even across the range. The same improvement holds for all count variables because they share the same skew and saturating pattern. In preprocessing we apply log(x + 1) transformation to every count feature to handle zeros and stabilize variance.

Year and month started as numeric, but keeping them numeric would make the model assume one straight line trend over time, which is unrealistic. For example, month wraps around with December next to January, and year effects can change abruptly from one year to the next.

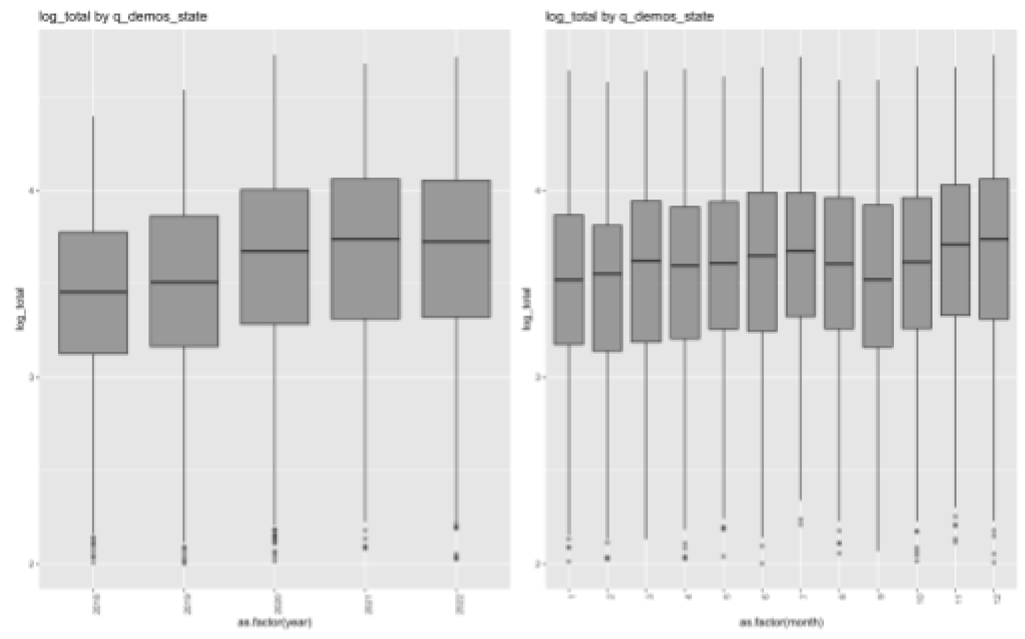


Figure 6: Relationship of state and month with log_total post converting to categorical

Now as categorical variables we can view the relationship of the variable with the outcome as seen in Figure 6. The boxplots show that log_total differs by both year and month. Log_total increase from 2018 until 2021 and minimally decrease in 2022. Log_total also varies slightly depending on months. Converting year and month into categorical variables allows us to learn separate effects and better capture the relationship between time and log_total. We apply this change in the preprocessing step.

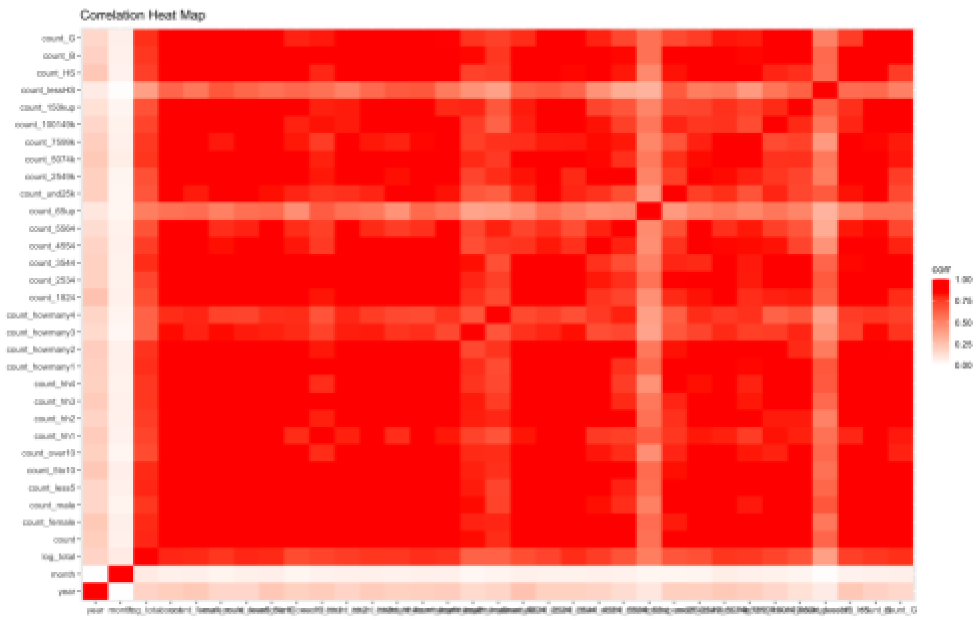


Figure 7: Heat Map

Examining the heat map in figure 7, we see high correlation amongst almost all predictors, seen through the dense red throughout the visual. High correlations amongst predictors poses the risk of multicollinearity which could result in unreliable and unstable coefficient estimates. In preprocessing, we convert key counts into proportions to reduce shared size effects. We also include interaction effects to capture non-additive relationships and improve predictive accuracy.

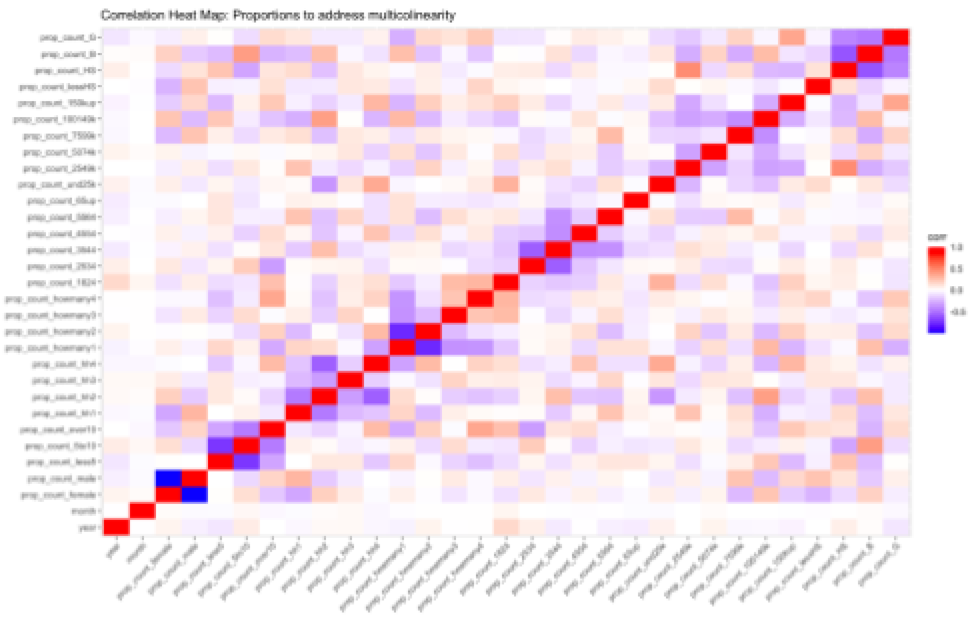


Figure 8: Heat map post conversion to proportions

As an example, a simple conversion of all count variables to proportions of the count, as seen in figure 8, alleviates some of the high correlation between predictors and helps reduce the risk of multicollinearity, demonstrating how adjusting variables can reduce multicollinearity risks. However, this approach can also lead to the loss of scale information. In the preprocessing section, we explored different transformations of the variables as well as investigated interaction effects between variables in our models and selected the approach that gave the best cross validation RMSE.

Overall, our exploratory analysis supports log transforming skewed counts, treating year and month as categorical variables, and adding key proportions and interaction terms. These choices reduce multicollinearity, linearize relationships, and improve fit, which we will validate through cross validation before finalizing the model.

Recipes and Preprocessing

We began by building a tidymodels recipe to make preprocessing reproducible and across resamples. We specify the recipe with the compact formula log_total ~ ., which tells tidymodels to predict log_total using all remaining predictors after preprocessing. First, we log-transform all count variables with step_log(starts_with("count"), offset = 1, base = 10). As seen in our EDA, counts are heavy-tailed and variance grows as count increases. The log transformation compresses large values, stabilizes variance, and makes linear trends more linear. The offset = 1 prevents taking log(0). We use base-10 to stay consistent with the response.

Next, we treat year and month as categorical variables via step_mutate(year = factor(year), month = factor(month)). Leaving them numeric would force a straight-line relationship. Now as factors, the model can learn separate effects for each year and month without implying linear distance. We then one-hot encode all nominal predictors with step_dummy(all_nominal_predictors()) so models that expect numeric inputs can use them.

We engineer features to capture structure we expect in retail demand. We add sin/cos pairs for month (month_sin, month_cos) so the model can learn the wrap-around nature of time. We include a simple Q4 indicator (is_q4) to let the model learn holiday surges. To compare places and periods fairly, we convert raw counts into ratios using total = pmax(count, 1). For example, we create female_ratio, high_income_ratio, college_plus_ratio, young_ratio, senior_ratio, and buyer-intensity shares. Ratios reduce scale effects and mitigate multicollinearity among raw counts.

We also compute an average household size by a weighted sum of household counts divided by total, and a shared account ratio to reflect multi-user behavior. We keep log_count = log1p(count) as an alternate scale of overall activity and a simple numeric state code (state_num) for models that can use ordinal encodings. Finally, we create interaction features such as income_education, high_income_ratio multiplied by college_plus_ratio, and heavy_buyer_income, heavy_buyer_ratio multiplied by high_income_ratio, to highlight segments that combine higher income with greater spending potential. After feature engineering and the creation of our recipe, we drop helper columns so only useful predictors are utilized in our models.

Candidate Models / Model Evaluation

We have six candidate models for predicting the outcome variable, log_total. The baseline models are Linear Regression, Ridge Regression, and Lasso Regression. Additionally, to capture nonlinear relationships, we have Random Forest, Gradient Boosting(XGBoost), and K-Nearest Neighbors (KNN).

Model descriptions:

1. **Model 1 (Linear Regression)**: A standard linear regression model using the lm engine with the preprocessed recipe.
2. **Model 2 (Ridge Regression):** A ridge regression model fit with glmnet using a penalty of 0.01 and mixture 0.
3. **Model 3 (Lasso Regression):** A lasso regression model fit with glmnet using a penalty of 0.01 and mixture 1.
4. **Model 4 (Random Forest):** A random forest model built with ranger using 500 trees, mtry = 15, and min_n = 5.
5. **Model 5 (K-Nearest Neighbors):** A KNN regression model fit with kknn using 20 neighbors, rectangular weights, and dist_power = 2.

6. **Model 6 (XGBoost)**: An XGBoost regression model trained on engineered features with tuned parameters including eta = 0.03, max_depth = 7, and subsample = 0.85.

Candidate Models Summary Table

The table below summarizes the candidate models, including their type, engine, preprocessing or features used, and the key hyperparameters for each approach.

Model #	**Model Type**	**Engine**	**Recipe / Variable Used**	**Hyperparameters**
Model 1 (LM)	Linear Regression	lm	Preprocessed recipe	-
Model 2 (Ridge)	Ridge Regression	glmnet	Preprocessed recipe	Penalty = 0.01, mixture = 1
Model 3 (Lasso)	Lasso Regression	glmnet	Preprocessed recipe	Penalty = 0.01, Mixture =1
Model 4 (Random Forest)	Random Forest	Ranger	Preprocessed recipe	Trees = 500 mtry = 5 Min_n = 5
Model 5 (KNN)	K-Nearest Ne	kknn	Preprocessed recipe	neighbors = 20 Dist_power = 2
Model 6 (XGBoost)	XGBoost	xgboost	Engineered features	Eta = 0.03, max_depth = 7, subsample = 0.85

(Note: “Preprocessed recipe” includes log-transformations for count variables, factor conversions for year/month, and one-hot encoding of categorical predictors. XGBoost additionally used engineered features such as demographic ratios, seasonality terms, and interaction variables.)

Model Evaluation and Tuning

We used 5-fold cross-validation to assess the performance of all candidate models, ensuring each model was tested on multiple subsets of the data for robust evaluation. Root mean squared error (RMSE) was selected as the primary metric because it effectively measures the average prediction error while penalizing larger deviations more heavily.

Hyperparameters were tuned at a basic level:

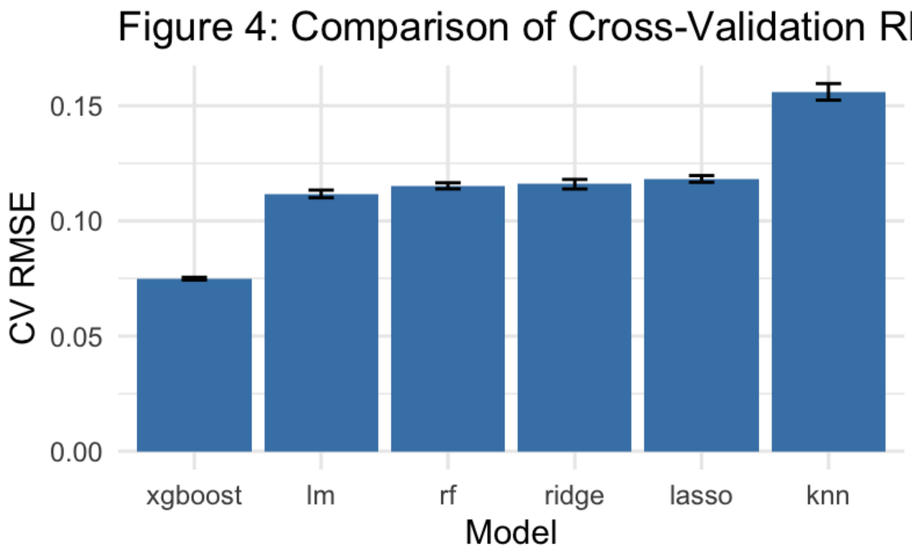
- Ridge and Lasso – penalty λ = 0.01, mixture = 0 (Ridge) and 1 (Lasso)
- Random Forest – trees, mtry, and min_n chosen from exploratory runs
- KNN – 20 neighbors with Euclidean distance
- XGBoost – parameters optimized via cross-validation with early stopping

Model #	**CV RMSE**	**Std. Error**
Model 1 (LM)	0.1117	0.00167
Model 2 (Ridge)	0.1159	0.00207
Model 3 (Lasso)	0.1182	0.00146
Model 4 (Random Forest)	0.1152	0.00132
Model 5 (KNN)	0.1560	0.00360
Model 6 (XGBoost)	0.0749	0.00059

Discussion of Results

The results show that regularized regression models (Model 2 and Model 3) had slightly higher RMSE than the linear baseline (Model 1), indicating no improvement in predictive performance. Model 4 captured nonlinearities and interactions, achieving lower RMSE. Model 5 performed better than linear models but was less stable in high-dimensional space. Model 6 achieved the best performance overall, with the lowest RMSE and strong generalization. This model benefited from both engineered features and boosting's ability to capture complex patterns.

The comparison plot below summarizes RMSE across all candidate models.



Discussion of final model

We selected XGBoost as the final model because it attained the lowest five fold cross validation error. The RMSE was 0.0749 and the standard error was 0.00059. This clearly improves on the linear baseline at 0.1117 and on the random forest at 0.1152. The reduction is large enough to be practically meaningful and the advantage was consistent across folds. The model used conservative regularization and early stopping. This controlled variance without heavy tuning and supports a stable choice for prediction.

The main strength of this model is flexibility. This is particularly helpful here because several of our engineered ratios are correlated by design. The model's boosting method provides a good trade-off between bias and variance, as shown by its low cross-validation error and stable performance. Another major benefit is that XGBoost naturally handles various feature types and scales, which reduced the need for extensive preprocessing.

The primary limitations involve interpretability and the evaluation. Unlike a linear model, XGBoost doesn't produce simple coefficients that explain a feature's effect. To get meaningful insights, we need to use post-hoc analyses. Furthermore, we relied on random cross-validation, which may give an overly optimistic picture if the model is used to predict future time periods or new geographic areas. A more rigorous approach would use time-series or group-based validation. It's also worth noting that because some predictors are ratios using "count" totals and we also include the count itself as a feature. We need to verify this data is available at prediction time and isn't unintentionally creating a proxy for the response. Finally, our hyperparameter tuning was intentionally basic, meaning there is likely still potential for better performance.

Future improvements fall into two main categories: the model and the data.

On the modeling side, we could benefit from a wider search for optimal hyperparameters and applying monotonic constraints to variables with known directional effects. It would also be valuable to stack our model with a simple linear model to improve its calibration, and to better interpret the results and prune less important features.

On the data side, we could capture more variation by incorporating new information. This could include data like holiday events or weather. Simple time-series features like moving averages or lagged variables would also be a strong addition.

Appendix: Final annotated script

```
#loading tidy models and ggplot
library(tidymodels)
library(ggplot2)
#patchwork allows us to put multiple ggplot visuals together as
one
#did this bc par does not work for ggplot but have visuals side
by side looks clean
library(patchwork)
library(xgboost)
library(kknn)
library(car)
#were gonna use large comment chunks to break up the sections of
our script!

#================================================================
==============
#                                  LOADING DATA
#================================================================
==============

#loading training and testing sets
train <- read.csv("train.csv")
test  <- read.csv("test.csv")
#head(train)

#================================================================
==============
#                                  EDA SECTION
#================================================================
==============

###############################################
##### variable Distributions section ##########
###############################################
#in this section we are just trying to get a feel for how the
variables are distributed
#trying to see the skewness of the variables and what not
#also trying to see the trends of distributions

#dist of outcome: log total
ggplot(train, aes(x = log_total)) +
  geom_histogram(bins = 30, color = "white", alpha = 0.9) +
  labs(
    title = "Distribution of Outcome: log_total",
    x = "log_total",
    y = "Count"
  ) +
  theme_minimal()

###########################
######## count vars #######
###########################

#mhy og code was super inefficient because I was replicating so
I created a function to help me plot

#creating a funcition that inputs our data set (train.csv) and
the variables for optimal plotting
#also allows less repitition!
#the ggplot function creates a histogram of the specified
variable!
plot_hist <- function(var, data) {
  ggplot(data, aes(x = .data[[var]])) +
    geom_histogram(bins = 30, color = "white", alpha = 0.9) +
    labs(
      title = paste("Distribution of", var),
      x = var,
      y = "Count"
    ) +
    theme_minimal()
}

#the variables will be all the predictors so in vars omitting
order total
#we need the names of the columns to be able to use the function
so use names()
vars <- train %>%
  names()


#generate histogram plots using lapply
dist_plots <- lapply(vars, plot_hist, data = train)

######################
#count, female, male #
######################
dist_plots[[6]] | dist_plots[[7]] | dist_plots[[8]]
```

```
######################
# order places vars ##
######################
dist_plots[[9]] | dist_plots[[10]] | dist_plots[[11]]

######################
# house hold size.  ##
######################
dist_plots[[12]] | dist_plots[[13]] | dist_plots[[14]] |
dist_plots[[15]]

######################
# user per account  ##
######################
dist_plots[[16]] | dist_plots[[17]] | dist_plots[[18]] |
dist_plots[[19]]

######################
#       ages.       ##
######################
dist_plots[[20]] | dist_plots[[21]] | dist_plots[[22]] |
dist_plots[[23]] | dist_plots[[24]] | dist_plots[[25]]

######################
#       income      ##
######################
dist_plots[[26]] | dist_plots[[27]] | dist_plots[[28]] |
dist_plots[[29]] | dist_plots[[30]] | dist_plots[[31]]

######################
#   education       ##
######################
dist_plots[[32]] | dist_plots[[33]] | dist_plots[[34]] |
dist_plots[[35]]




###############################################
##### Relationiships w/ outcome sect ##########
###############################################

###########
```

```
# numeric #
###########

#individual scatter plots with outocome variables
plot_scatter <- function(var, data) {
  ggplot(data, aes(x = .data[[var]], y = .data[["log_total"]]))
+
    geom_point(alpha = 0.3) +
    geom_smooth(method = "loess", se = FALSE, color = "black") +
    labs(title = paste("Log_total and", var),
         x = var,
         y = "log total")+
    theme_minimal()
}



#generate scatterplot plots using lapply
scatter_plots <- lapply(vars, plot_scatter, data = train)
######################
#count, female, male #
######################
scatter_plots[[6]] | scatter_plots[[7]] | scatter_plots[[8]]

######################
# order places vars ##
######################
scatter_plots[[9]] | scatter_plots[[10]] | scatter_plots[[11]]

######################
# house hold size.  ##
######################
scatter_plots[[12]] | scatter_plots[[13]] | scatter_plots[[14]]
| scatter_plots[[15]]

######################
# user per account  ##
######################
scatter_plots[[16]] | scatter_plots[[17]] | scatter_plots[[18]]
| scatter_plots[[19]]

######################
#       ages.      ##
######################
```

```
scatter_plots[[20]] | scatter_plots[[21]] | scatter_plots[[22]]
| scatter_plots[[23]] | scatter_plots[[24]] |
scatter_plots[[25]]

######################
#       income     ##
######################
scatter_plots[[26]] | scatter_plots[[27]] | scatter_plots[[28]]
| scatter_plots[[29]] | scatter_plots[[30]] |
scatter_plots[[31]]

######################
#   education      ##
######################
scatter_plots[[32]] | scatter_plots[[33]] | scatter_plots[[34]]
| scatter_plots[[35]]

##############################
######## TRANSFORMATIONS #####
##############################

################################
## log transformations: hists ##
################################

#checking the histogram distributions of the variables with a
log transformation
#the log transformation allows the

log_plot_hist <- function(var, data) {
  ggplot(data, aes(x = log(.data[[var]]))) +
    geom_histogram(bins = 30, color = "white", alpha = 0.9) +
    labs(
      title = paste("Distribution of", var),
      x = var,
      y = "Count"
    ) +
    theme_minimal()
}

#generate histogram plots using lapply
log_dist_plots <- lapply(vars, log_plot_hist, data = train)

######################
```

```
#count, female, male #
######################
log_dist_plots[[6]] | log_dist_plots[[7]] | log_dist_plots[[8]]

######################
# order places vars ##
######################
log_dist_plots[[9]] | log_dist_plots[[10]] |
log_dist_plots[[11]]

######################
# house hold size.  ##
######################
log_dist_plots[[12]] | log_dist_plots[[13]] |
log_dist_plots[[14]] | log_dist_plots[[15]]

######################
# user per account  ##
######################
log_dist_plots[[16]] | log_dist_plots[[17]] |
log_dist_plots[[18]] | log_dist_plots[[19]]

######################
#       ages.       ##
######################
log_dist_plots[[20]] | log_dist_plots[[21]] |
log_dist_plots[[22]] | log_dist_plots[[23]] |
log_dist_plots[[24]] | log_dist_plots[[25]]

######################
#       income      ##
######################
log_dist_plots[[26]] | log_dist_plots[[27]] |
log_dist_plots[[28]] | log_dist_plots[[29]] |
log_dist_plots[[30]] | log_dist_plots[[31]]

######################
#   education       ##
######################
log_dist_plots[[32]] | log_dist_plots[[33]] |
log_dist_plots[[34]] | log_dist_plots[[35]]

dist_plots[[10]]
log_dist_plots[[10]]
```

```
##################################
## log transformations: scatter ##
##################################

#individual histograms and boxplots of variables
log_plot_scatter <- function(var, data) {
  ggplot(data, aes(x = log(.data[[var]] + 1), y =
.data[["log_total"]])) +
    geom_point(alpha = 0.6) +
    geom_smooth(method = "loess", se = FALSE, color = "black") +
    labs(title = paste("Log_total and log ", var),
         x = var,
         y = "log total") +
    theme_minimal()
}


#generate scatterplot plots using lapply
log_scatter_plots <- lapply(vars, log_plot_scatter, data =
train)
######################
#count, female, male #
######################
log_scatter_plots[[6]] | log_scatter_plots[[7]] |
log_scatter_plots[[8]]

######################
# order places vars ##
######################
log_scatter_plots[[9]] | log_scatter_plots[[10]] |
log_scatter_plots[[11]]

######################
# house hold size.  ##
######################
log_scatter_plots[[12]] | log_scatter_plots[[13]] |
log_scatter_plots[[14]] | log_scatter_plots[[15]]

######################
# user per account  ##
######################
```

```
log_scatter_plots[[16]] | log_scatter_plots[[17]] |
log_scatter_plots[[18]] | log_scatter_plots[[19]]

######################
#  ages.           ##
######################
log_scatter_plots[[20]] | log_scatter_plots[[21]] |
log_scatter_plots[[22]] | log_scatter_plots[[23]] |
log_scatter_plots[[24]] | log_scatter_plots[[25]]

######################
#       income     ##
######################
log_scatter_plots[[26]] | log_scatter_plots[[27]] |
log_scatter_plots[[28]] | log_scatter_plots[[29]] |
log_scatter_plots[[30]] | log_scatter_plots[[31]]

######################
#   education      ##
######################
log_scatter_plots[[32]] | log_scatter_plots[[33]] |
log_scatter_plots[[34]] | log_scatter_plots[[35]]

dist_plots[[16]] | scatter_plots[[16]]
log_dist_plots[[16]] | log_scatter_plots[[16]]

#############
#categorical#
#############
# gonna use bar charts
#allows us to see the relationships of categorical variables
with outcome

#higher populations like california and texas cause more orders
ggplot(train, aes(x = q_demos_state, y = log_total)) +
  geom_boxplot(outlier.alpha = 0.4, fill = "darkgrey") +
  labs(title = "log_total by q_demos_state") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

year_af <- ggplot(train, aes(x = as.factor(year), y =
log_total)) +
  geom_boxplot(outlier.alpha = 0.4, fill = "darkgrey") +
  labs(title = "log_total by q_demos_state") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

```
month_af <- ggplot(train, aes(x = as.factor(month), y =
log_total)) +
  geom_boxplot(outlier.alpha = 0.4, fill = "darkgrey") +
  labs(title = "log_total by q_demos_state") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

year_af|month_af
################################
## heat map                   ##
################################
numeric_for_heat_map <- train %>%
  select(-order_totals, -q_demos_state)

#correlation matrix
cormat <- cor(numeric_for_heat_map, use =
"pairwise.complete.obs")
df <- as.data.frame(as.table(cormat))
names(df) <- c("var1", "var2", "corr")

#heat map in ggplot
ggplot(df, aes(x = var2, y = var1, fill = corr)) +
  geom_tile() +
  scale_fill_gradient2(low="blue", mid="white", high="red",
midpoint=0) +
  labs(x = NULL, y = NULL, title = "Correlation Heat Map")

##########################
# vif ######
##########################

train_processed <- train %>%
  mutate(across(starts_with("count_"), ~ .x / count, .names =
"prop_{.col}"))

#get rud if the original count columns
train_processed <- train_processed %>%
  select(-starts_with("count_"))

#keep predictors we need
df_predictors <- train_processed %>%
  select(-order_totals, -count, -log_total, -q_demos_state)

#correlation matric
```

```
corr_matrix <- cor(df_predictors, use = "pairwise.complete.obs")

df_corr <- as.data.frame(as.table(corr_matrix))
names(df_corr) <- c("var1", "var2", "corr")

#heat map with changing to
ggplot(df_corr, aes(x = var2, y = var1, fill = corr)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high =
"red", midpoint = 0) +
  labs(x = NULL, y = NULL, title = "Correlation Heat Map:
Proportions to address multicolinearity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))


#===============================================================
==============
#                                 PRE PROCESSING
#===============================================================
==============
#prep with transformations
#transformations used:
#all count variables are log tranformed to make them linear with
the outcome
#month and year are tranformed into categorical and made factors

#remove order
train_clean <- train %>%
  select(- order_totals)

#create recipe for preprocessing
rec <- recipe(log_total ~ ., data = train_clean) %>%
  # log-transform all count_ variables
  step_log(starts_with("count"), offset = 1, base = 10) %>%
  # convert year and month to factors
  step_mutate(
    year = factor(year),
    month = factor(month)
  )%>%
  # one-hot encode categorical variables
  step_dummy(all_nominal_predictors())

#####################
# feature engineer ##
```

```
#####################

engineer_features <- function(df) {
  df %>%
    mutate(
      month_sin = sin(2 * pi * month / 12),
      month_cos = cos(2 * pi * month / 12),
      is_q4 = as.numeric(month >= 10),
      total = pmax(count, 1),
      female_ratio = count_female / total,
      high_income_ratio = (count_100149k + count_150kup) /
total,
      college_plus_ratio = (count_B + count_G) / total,
      young_ratio = (count_1824 + count_2534) / total,
      senior_ratio = count_65up / total,
      avg_household = (count_hh1 + 2*count_hh2 + 3*count_hh3 +
4*count_hh4)/total,
      single_household_ratio = count_hh1/total,
      heavy_buyer_ratio = count_over10/total,
      moderate_buyer_ratio = count_5to10/total,
      shared_account_ratio = (count_howmany2 + count_howmany3 +
count_howmany4)/total,
      log_count = log1p(count),
      state_num = as.numeric(factor(q_demos_state)),
      income_education = high_income_ratio * college_plus_ratio,
      heavy_buyer_income = heavy_buyer_ratio * high_income_ratio
    ) %>%
    select(-total)
}

train <- engineer_features(train)
test <- engineer_features(test)

#================================================================
==============
#                                 MODEL CREATION
#================================================================
==============

######################
###  linear       ####
######################
lm_model <- linear_reg(
  mode = "regression"
```

```r
) %>%
  set_engine("lm")

lm_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lm_model)

lm_fit <- fit(lm_workflow, data = train_clean)

test_preds_lm <- predict(lm_fit, new_data = test)
head(test_preds_lm)

######################
###  lasso        ####
######################
#create lasso model
lasso_model <- linear_reg(penalty = 0.01, mixture = 1) %>%
  set_engine("glmnet")

#create laso workflow
lasso_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lasso_model)

#fit work flow
lasso_fit <- fit(lasso_workflow, data = train_clean)

#predict on test
test_preds_lasso <- predict(lasso_fit, new_data = test)


######################
###  ridge        ####
######################
ridge_model <- linear_reg(
  mode = "regression",
  penalty = 0.01,
  mixture = 0
) %>%
  set_engine("glmnet")

ridge_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(ridge_model)
```

```
ridge_fit <- fit(ridge_workflow, data = train_clean)

test_preds_ridge <- predict(ridge_fit, new_data = test)

######################
###  random forest  ##
######################
rf_model <- rand_forest(
  trees = 500,
  mtry = 15,
  min_n = 5
) %>%
  set_mode("regression") %>%
  set_engine("ranger")

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

rf_fit <- rf_workflow %>%
  fit(data = train_clean)

test_preds_rf <- rf_fit %>% predict(test)


######################
###  Knn.          ##
######################

knn_model <- nearest_neighbor(
  neighbors = 20,
  weight_func = "rectangular",
  dist_power = 2
) %>%
  set_mode("regression") %>%
  set_engine("kknn")

knn_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(knn_model)

knn_fit <- knn_workflow %>%
  fit(data = train_clean)
```

```
test_preds_knn <- knn_fit %>%
  predict(test)


######################
###  XGBoost        ##
######################

feature_cols <- c('state_num', 'year', 'month', 'month_sin',
'month_cos', 'is_q4',
                  'log_count', 'female_ratio',
'heavy_buyer_ratio', 'moderate_buyer_ratio',
                  'high_income_ratio', 'college_plus_ratio',
'young_ratio', 'senior_ratio',
                  'avg_household', 'single_household_ratio',
'shared_account_ratio',
                  'income_education', 'heavy_buyer_income')

X_train <- train %>% select(all_of(feature_cols)) %>%
as.matrix()
y_train <- train$log_total
X_test <- test %>% select(all_of(feature_cols)) %>% as.matrix()

dtrain <- xgb.DMatrix(data = X_train, label = y_train)
dtest <- xgb.DMatrix(data = X_test)

params <- list(
  objective = "reg:squarederror",
  eta = 0.03,
  max_depth = 7,
  min_child_weight = 5,
  subsample = 0.85,
  colsample_bytree = 0.85,
  gamma = 0.05,
  alpha = 0.1,
  lambda = 1,
  seed = 42
)

##############################
###  cross vaildation       ##
##############################
```

```
set.seed(114)
cv_result <- xgb.cv(params = params, data = dtrain, nrounds =
3000,
                    nfold = 5, early_stopping_rounds = 50,
print_every_n = 100)

best_rounds <- cv_result$best_iteration

model <- xgb.train(params = params, data = dtrain, nrounds =
best_rounds)
predictions <- predict(model, dtest)

set.seed(123)
folds <- vfold_cv(train_clean, v = 5)

# CV 수행
set.seed(123)
lm_res <- fit_resamples(lm_workflow, resamples = folds, metrics
= metric_set(rmse))

lasso_res <- fit_resamples(lasso_workflow, resamples = folds,
metrics = metric_set(rmse))

ridge_res <- fit_resamples(ridge_workflow, resamples = folds,
metrics = metric_set(rmse))

rf_res <- fit_resamples(rf_workflow, resamples = folds, metrics
= metric_set(rmse))

knn_res <- fit_resamples(knn_workflow, resamples = folds,
metrics = metric_set(rmse))

# tidymodels results table
tidymodels_results <- bind_rows(
  lm = collect_metrics(lm_res),
  lasso = collect_metrics(lasso_res),
  ridge = collect_metrics(ridge_res),
  rf = collect_metrics(rf_res),
  knn = collect_metrics(knn_res),
  .id = "model"
) %>%
  filter(.metric == "rmse") %>%
  select(model, mean, std_err)
```

```
# XGBoost CV RMSE
dtrain <- xgb.DMatrix(data = X_train, label = y_train)

set.seed(123)
xgb_cv <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 3000,
  nfold = 5,
  early_stopping_rounds = 50,
  verbose = FALSE,
  metrics = list("rmse")
)

best_iter <- xgb_cv$best_iteration
xgb_cv_rmse <- xgb_cv$evaluation_log[best_iter,
c("train_rmse_mean", "train_rmse_std")]

xgb_result <- tibble(
  model = "xgboost",
  mean = xgb_cv_rmse$train_rmse_mean,
  std_err = xgb_cv_rmse$train_rmse_std
)

#=========================
# Combine all results
#=========================
cv_results <- bind_rows(tidymodels_results, xgb_result)
cv_results

#=========================
# comparison plots
#=========================

# Table of CV RMSE + SE
cv_results %>%
  kable(digits = 5, caption = "Cross-Validation RMSE and
Standard Error for Candidate Models")

#  RMSE Comparison
ggplot(cv_results, aes(x = reorder(model, mean), y = mean)) +
  geom_col(fill = "steelblue") +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean +
std_err), width = 0.2) +
  labs(
    title = "Figure 4: Comparison of Cross-Validation RMSE
Across Models",
    x = "Model",
    y = "CV RMSE"
  ) +
  theme_minimal()

#==================================================================
==============
#                                 WRITE TO CSV
#==================================================================
==============
final_preds <- test_preds_rf %>% bind_cols(test %>% select(id))
%>%
  rename(log_total = .pred) %>%
  select(id, log_total)

write_csv(final_preds, "submission.csv")
```

Appendix: Team member contributions

Claire Nabours: Creation of EDA section of report, revision of preprocessing and recipe section, original preprocessing/recipe pre feature engineering, linear model code, ridge model code, and lasso model code.

Jackie Sung: Reported Candidate models / model evaluation part

Naikang Wang: Introduction, Discussion of final model, XGBoost Model

Yankai Wen: pre-processing