Analysis of Amazon Orders

Brandon Nava
July 28, 2024

Contents

1 Introduction **2**

2 Data Analysis **2**
2.1 Relationships & Visualizations 2

3 Preprocessing/Recipes **11**

4 Model Evaluation **11**

5 Discussion **13**

6 Appendix & Contributions **9**

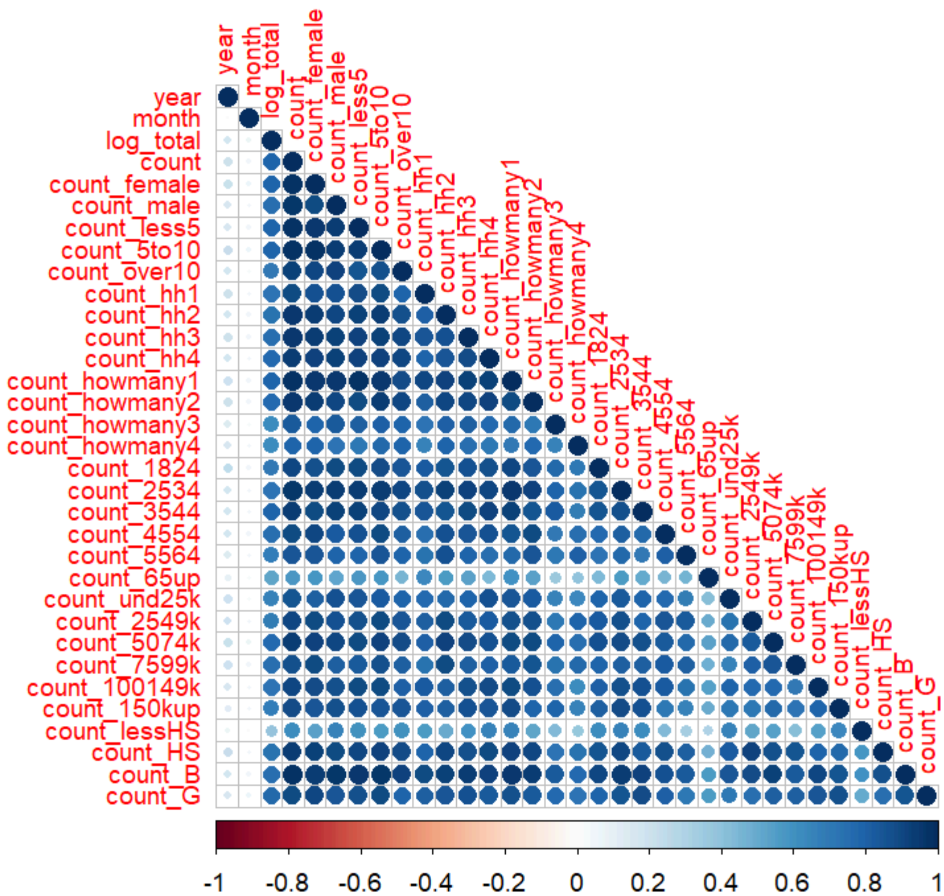
1 Introduction

This report analyzes factors associated with the response variable, ‘log_total’, which represents the log10 of order totals for Amazon customers. By having a data set with variables such as ‘count’ (total number of orders), orders based on sex such as ‘count_female’ (orders placed by females), and order frequency categories such as ‘count_less5’, ‘count_5to10’, and ‘count_over10’. In this study, I hypothesized that the predictors related to factors such as household size, account sharing, age, income and education levels have an influence on log_total.

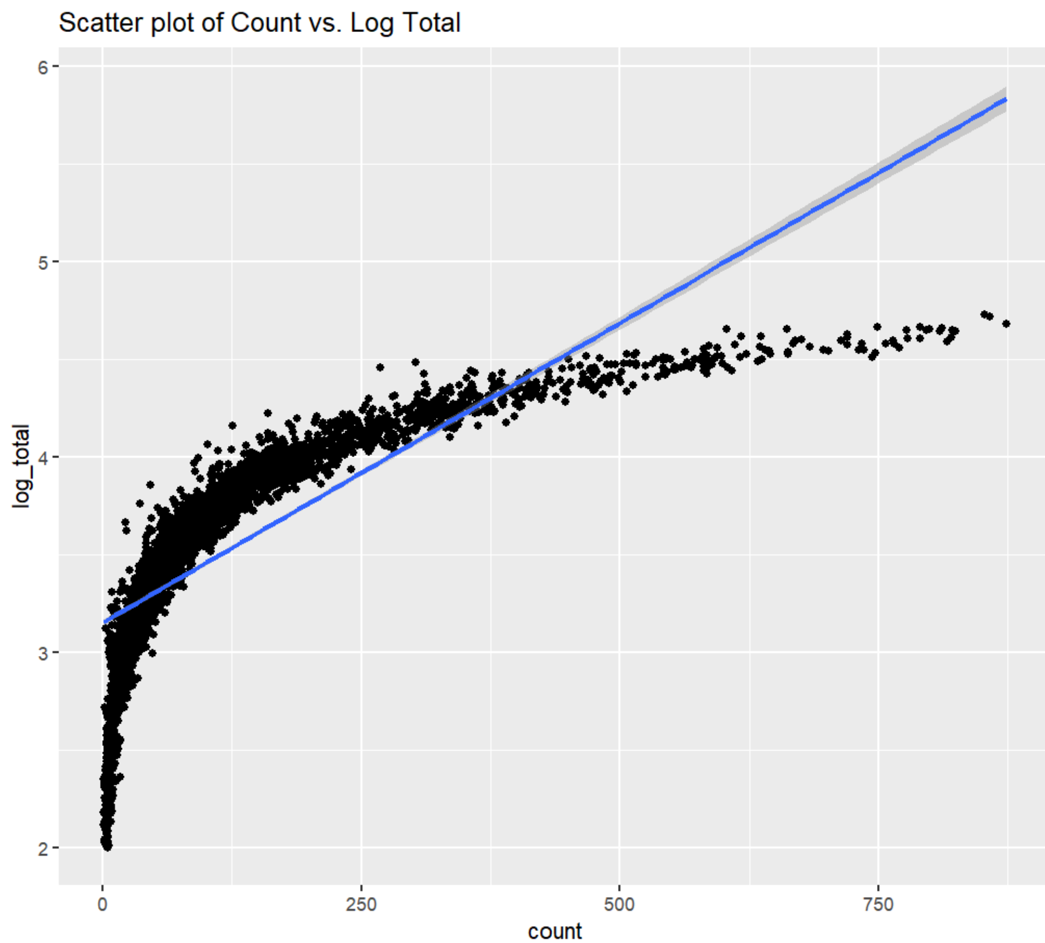
2 Data Analysis

2.1 Relationships & Visualizations

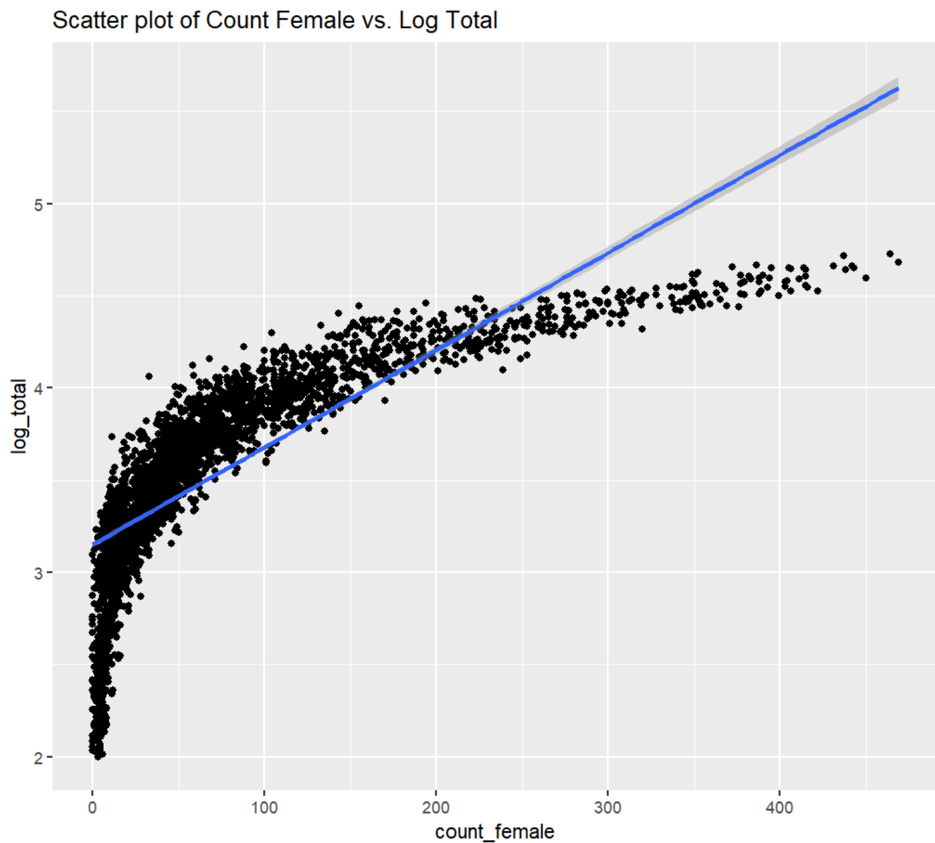
In order to visualize the relationships between the variables, I included 8 graphics to identify patterns, trends, and other potential factors that affect purchasing behavior. I used a combination of correlation analysis, scatter plots, box plots, and histograms in order to provide a comprehensive understanding of the data given.



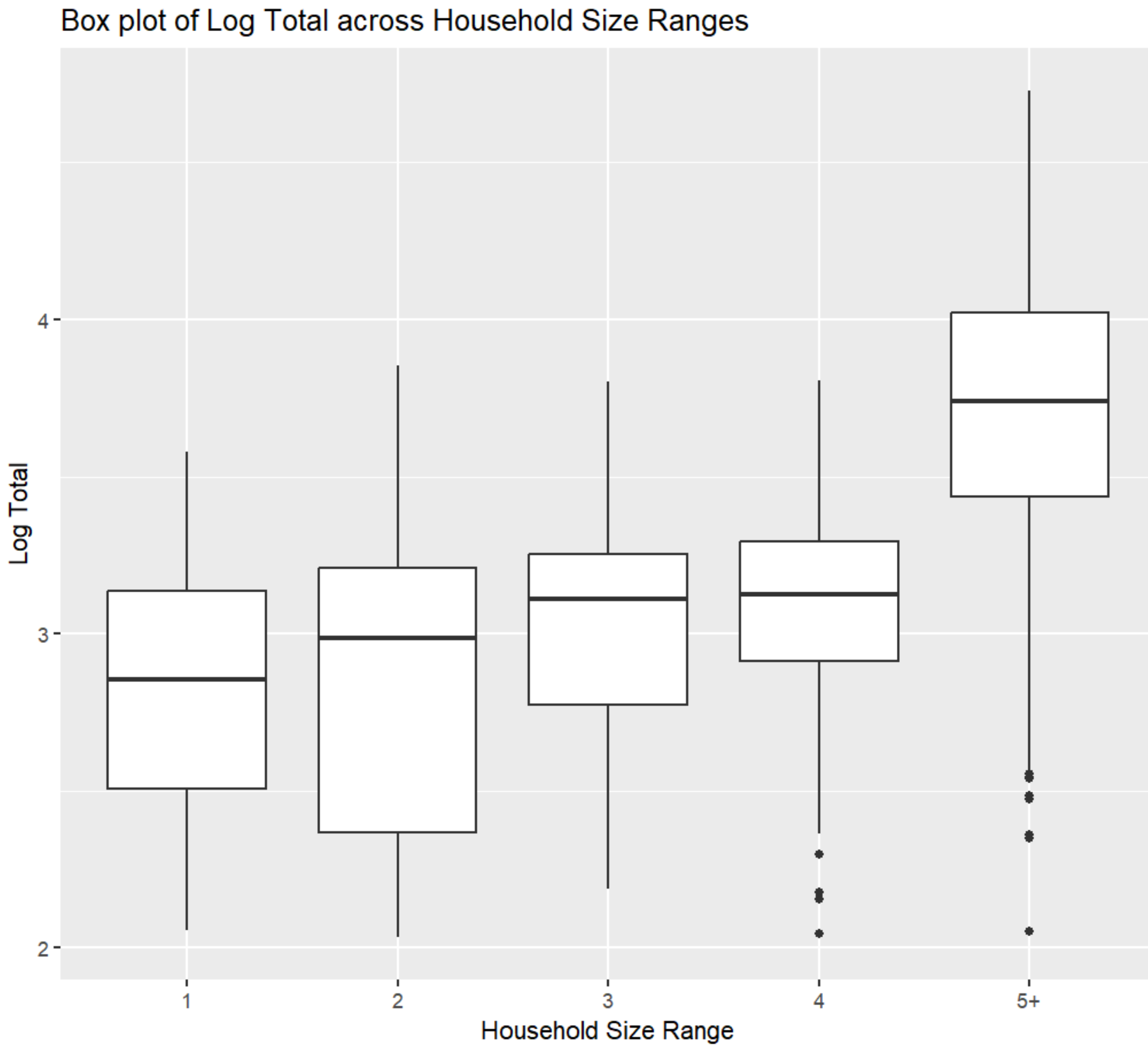
This correlation matrix visual helps us to see the relationships between the numeric variables in the dataset. Here we can identify pairs of variables strongly correlated with each other as well as log_total. Since most have a strong correlation as seen on the scale, we can explore their relationships through other visualizations.



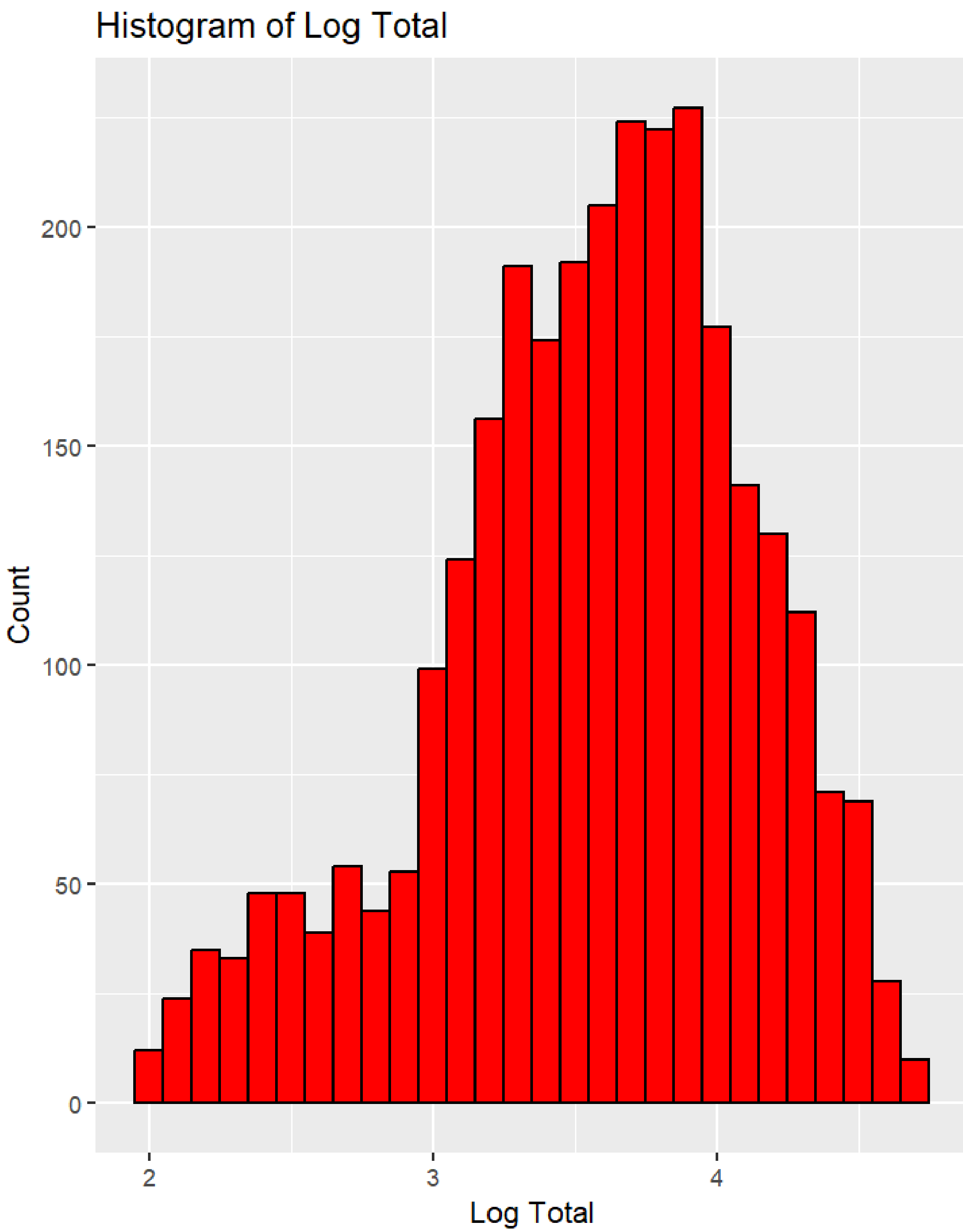
In this scatter plot of count (total number of orders) vs log_total we see a positive correlation between both variables. This plot indicates that as orders increase, log_total increases, with there being more linearity for higher orders. We can continue exploring other aspects of this through more visuals.



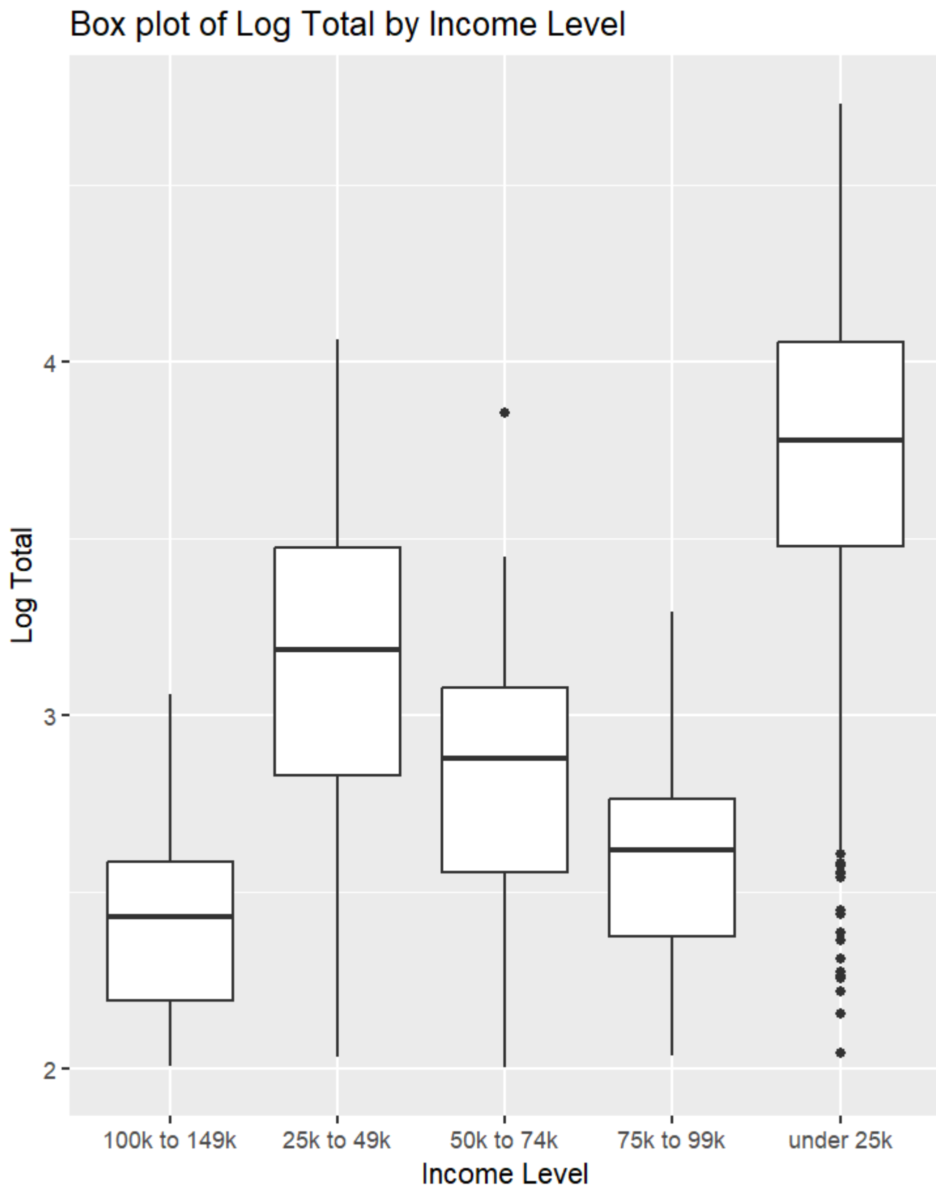
In this scatter plot we focus solely on the orders made by females, where we see a similar distribution. This similarity indicates that count_female plays a significant role in the overall distribution of log_total.



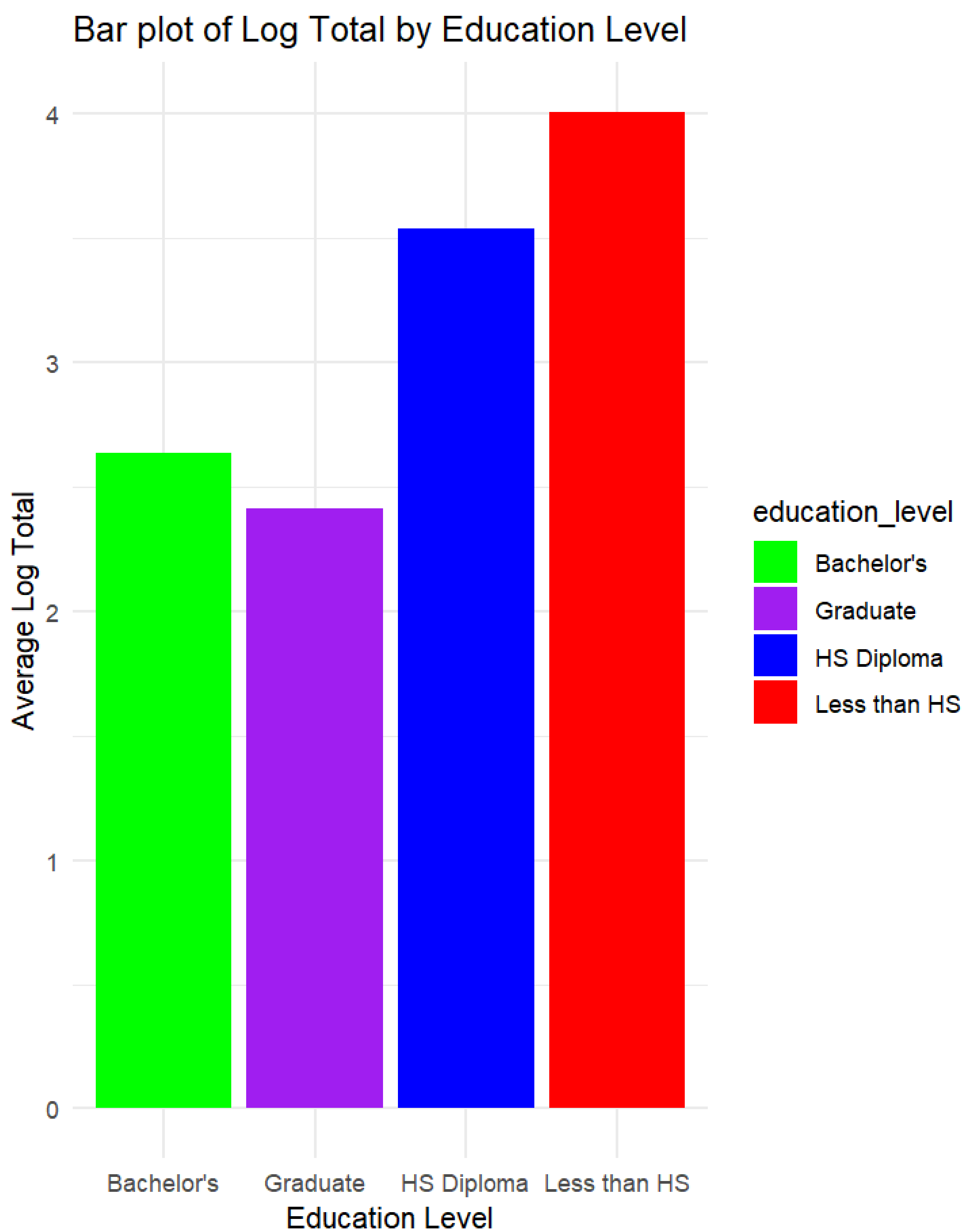
This box plot shows us how the median for log_total increases as the household size increases. There are a few outliers in the 4 and 5+ areas which gives us even more room to explore possible factors causing these outliers and if there is a greater significance to them.



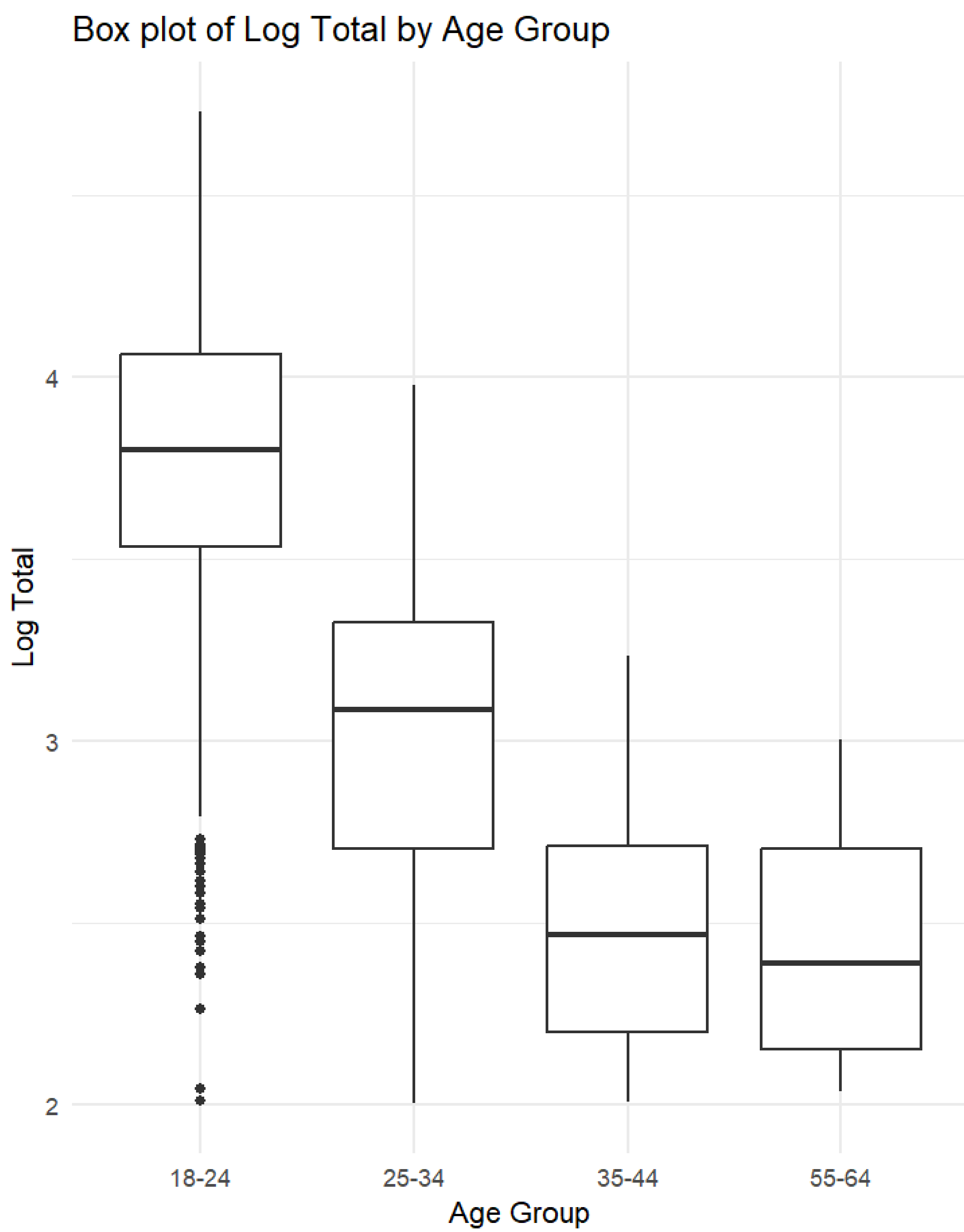
In this histogram, we see a roughly normal distribution centered around 3 and 4. The data is slightly skewed to the left, however the order totals behave relatively well. This tells us that the log transformation assisted in normalizing the original order totals.



In this box plot by income level, we see a negative linear distribution of orders according to income level. We see that income levels under 2k have the highest median of log_total, and as the income range increases, the median of orders decreases overall. Our next plot also gives us an interesting contrast of categories.



This colored bar plot indicates the significant differences between education levels and their order frequencies. According to the data, individuals with lower education levels have higher average order totals compared to those with higher education levels. This can potentially be due to differences in purchasing behaviors affected by underlying socioeconomic factors.



In this box plot by age group, we can see that younger individuals (18-24) have a higher average order than other age groups. There is a linear relationship here as the older the age groups get, the less frequency we see in orders.

3 Preprocessing/Recipes

For preprocessing I used, `recipe(log_total ~ ., data = train)` in order to set up the recipe with `log_total` as the response and all other variables as predictors. I then used step_novel(all_nominal_predictors())` to manage unseen categorical levels in the test data, `step_dummy(all_nominal_predictors())` to convert any categorical variables to binary dummy variables, `step_zv(all_predictors())` to remove predictors with no variation, `step_impute_mean(all_numeric_predictors())` to fill missing numeric values with the mean, and `step_normalize(all_numeric_predictors())` to scale numeric predictors to have a mean of zero and standard deviation of one. I took all these steps in order to ensure that the data is clean and suitable for modeling.

5 Model Evaluation

I used the following models for my report:

- **Linear Regression**: Used to predict log_total based on all available predictors. Linear regression models the relationship between the dependent variable and one or more independent variables using a linear approach.
- **Gradient Boosting Machine (GBM)**:: A method that builds trees sequentially, where each tree corrects errors made by the previous ones. It uses the xgboost implementation for gradient boosting.
- **Random Forest**: Used to build multiple decision trees and merge them together to get a more accurate and stable prediction. Each tree is trained on a random subset of the data.
- **Decision Tree**: Splits the data into branches to make predictions based on feature values. It is easy to interpret but prone to overfitting.
- **K-Nearest Neighbors (KNN)**: Predicts the value of a new data point based on the majority class or average value of its k-nearest neighbors.

I have also included a table of all the candidate models:

Model Identifier	**Type of model**	**Engine**	**Recipe, variables**	**Hyperparameters**
Linear Model	Linear Regression	lm	step_novel(), step_dummy(), step_zv(), step_impute_mean(), step_normalize()	none
GBM Model	Gradient Boosting	xgboost	step_novel(), step_dummy(), step_zv(), step_impute_mean(), step_normalize()	trees = 1000, learn_rate, tree_depth
Random Forest Model	Random Forest	ranger	step_novel(), step_dummy(), step_zv(), step_impute_mean(), step_normalize()	Default settings
Decision Tree Model	Decision Tree	rpart	step_novel(), step_dummy(), step_zv(), step_impute_mean(), step_normalize()	cost_complexity, tree_depth
KNN Model	K-Nearest Neighbors	kknn	step_novel(), step_dummy(), step_zv(), step_impute_mean(), step_normalize()	neighbors

The evaluation was conducted using 10-fold cross-validation, ensuring robust and reliable performance metrics. Hyperparameter tuning was performed for models with tunable parameters, such as GBM, Decision Tree, and KNN, to optimize their predictive performance. V-fold cross-validation (with v = 10) was used to measure the performance of the candidate models. This method splits the data into 10 subsets (folds), trains the model on 9 folds, and validates it on the remaining fold. This process is repeated 10 times, with each fold used exactly once as the validation data. Hyperparameters for the GBM, Decision Tree, and KNN models were tuned using grid search. The best hyperparameters were selected based on the RMSE metric. The following table shows how each candidate performed:

Model identifier	**Metric**	**Estimator**	**Mean RMSE**	**n**	**SE of RMSE**	**Configuration**
Linear	RMSE	Standard	0.147	10	0.00348	Preprocessor1_Model1
GBM	RMSE	Standard	0.117	10	0.00273	Preprocessor1_Model01
Random Forest	RMSE	Standard	0.113	10	0.00261	Preprocessor1_Model1
Decision Tree	RMSE	Standard	0.346	10	0.00306	Preprocessor1_Model1
KNN	RMSE	Standard	0.137	10	0.00418	Preprocessor1_Model1

6 Discussion

Based on the RMSE metrics, the Random Forest model was the best-performing model, followed closely by the GBM model. However, I decided to try the GBM model for final predictions to see how different it would perform from the Random Forest model, which was previously used in homework 3. Some of the strengths of this model include high accuracy, which combines multiple weak learners to strengthen the model. It handles complex data well due to its learning process and is very flexible for regression analysis. Some of the weaknesses include its computational complexity, which was slow in the process and resource heavy. The hyperparameters need careful tuning to avoid over/underfitting. It is also sensitive to noisy data and can be greatly affected by not using proper processing.

7 Appendix & Contributions

This is my final annotated R script file. I performed on my own in team 17.

```
#Nava, Brandon - Regression Script Submission

install.packages("readr")
install.packages("tidyverse")
install.packages("tidyr")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("tidymodels")
install.packages("ranger")
install.packages("stacks")
install.packages("corrplot")
install.packages("GGally")
install.packages("kknn")
install.packages("rpart")
install.packages("xgboost")
install.packages("recipes")
install.packages("doParallel")


# Load necessary packages
library(doParallel)
library(recipes)
library(xgboost)
library(rpart)
library(kknn)
library(tidyverse)
library(readr)
library(tidyr)
library(dplyr)
library(tidymodels)
library(ranger)
library(ggplot2)
library(stacks)
library(corrplot)
```

```
library(GGally)

## Did this during processing due to computations
# Detect the number of cores
num_cores <- detectCores()
# Register parallel backend
cl <- makePSOCKcluster(num_cores)
registerDoParallel(cl)

# Set directory
setwd("C:/Users/RGRIND")

# Read in train data
train <- read_csv("train.csv")

# Remove order_total in order total to focus on log total
train <- train %>% select(-order_totals)

# Make sure log_total is numeric
train$log_total <- as.numeric(train$log_total)

# Correlation matrix for numeric variables
cor_matrix <- cor(train %>% select_if(is.numeric), use =
"complete.obs")

# Plot the correlation matrix
corrplot(cor_matrix, method = "circle", type = "lower", tl.cex =
0.8)

# Scatter plot for count vs. log_total
ggplot(train, aes(x = count, y = log_total)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Scatter plot of Count vs. Log Total")

# Scatter plot for count_female vs. log_total
ggplot(train, aes(x = count_female, y = log_total)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Scatter plot of Count Female vs. Log Total")
```

```
# Store household size in different ranges
train_data <- train %>%
  mutate(household_size_range = cut(count_hh4, breaks = c(0, 1,
2, 3, 4, Inf),
                                    labels = c("1", "2", "3",
"4", "5+")))

# Remove rows with NA in household_size_range
train_data <- train_data %>%
filter(!is.na(household_size_range))

# Box plot for log_total across household sizes
ggplot(train_data, aes(x = household_size_range, y = log_total))
+
  geom_boxplot() +
  labs(title = "Box plot of Log Total across Household Size
Ranges",
       x = "Household Size Range", y = "Log Total")

# Histogram of log_total
ggplot(train, aes(x = log_total)) +
  geom_histogram(binwidth = 0.1, fill = "red", color = "black")
+
  labs(title = "Histogram of Log Total", x = "Log Total", y =
"Count")

# Variable for income level category
train_income <- train %>%
  mutate(income_category = case_when(
    count_und25k > 0 ~ "under 25k",
    count_2549k > 0 ~ "25k to 49k",
    count_5074k > 0 ~ "50k to 74k",
    count_7599k > 0 ~ "75k to 99k",
    count_100149k > 0 ~ "100k to 149k",
    count_150kup > 0 ~ "over 150k"
  ))

# Remove rows with NA in income category
train_income <- train_income %>% filter(!is.na(income_category))

# Box plot for log_total by income category
```

```
ggplot(train_income, aes(x = income_category, y = log_total)) +
  geom_boxplot() +
  labs(title = "Box plot of Log Total by Income Level", x =
"Income Level", y = "Log Total")

# Variable for education level category
train_ed <- train %>%
  mutate(education_level = case_when(
    count_lessHS > 0 ~ "Less than HS",
    count_HS > 0 ~ "HS Diploma",
    count_B > 0 ~ "Bachelor's",
    count_G > 0 ~ "Graduate",
    TRUE ~ NA_character_  # Handle cases where none of the
conditions are met
  ))

# Remove rows with NA in education_level
train_ed <- train_ed %>% filter(!is.na(education_level))

# Stacked bar plot for log_total by education level
ggplot(train_ed, aes(x = education_level, y = log_total, fill =
education_level)) +
  geom_bar(stat = "summary", fun = "mean") +
  labs(title = "Bar plot of Log Total by Education Level",
       x = "Education Level", y = "Average Log Total") +
  scale_fill_manual(values = c("Less than HS" = "red", "HS
Diploma" = "blue", "Bachelor's" = "green", "Graduate" =
"purple")) +
  theme_minimal()

# Variable for age group category
train_age <- train %>%
  mutate(age_group = case_when(
    count_1824 > 0 ~ "18-24",
    count_2534 > 0 ~ "25-34",
    count_3544 > 0 ~ "35-44",
    count_4554 > 0 ~ "45-54",
    count_5564 > 0 ~ "55-64",
    count_65up > 0 ~ "65+",
    TRUE ~ NA_character_  # Handle cases where none of the
conditions are met
```

```
  ))

# Remove rows with NA in age_group
train_age <- train_age %>% filter(!is.na(age_group))

# Box plot for log_total across age groups
ggplot(train_age, aes(x = age_group, y = log_total)) +
  geom_boxplot() +
  labs(title = "Box plot of Log Total by Age Group",
       x = "Age Group", y = "Log Total") +
  theme_minimal()


# Initial data split for cross-validation
set.seed(1)
data_split <- initial_split(train)
train_set <- training(data_split)

# V-fold cross-validation
set.seed(1)
cv_fold <- vfold_cv(train_set, v = 10, strata = log_total)

# Recipe for data
data_recipe <- recipe(log_total ~ ., data = train_set) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_impute_mean(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())


### Building of models

## Random Forest Model
# Create 10-fold cross-validation set with stratification on
log_total
cv_fold <- vfold_cv(train, v = 10, strata = log_total)

data_rec <- recipe(log_total ~ ., data = train_set) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
```

```
  step_impute_mean(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors())

# Define control settings for stacking
ctrl_grid <- control_stack_grid()
ctrl_res <- control_stack_resamples()

# Create random forest model
rf <- rand_forest(mode = "regression") %>%
  set_engine("ranger")

# Create random forest model for regression
rf_mod <- rand_forest(
  mode = "regression"
) %>%
  set_engine("ranger")

# Create workflow and add formula
rf_flow <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(data_rec)

# Fit workflow to set of resamples
results <- fit_resamples(
  rf_flow,
  resamples = cv_fold,
  metrics = metric_set(rmse, rsq),
  control = control_resamples(save_pred = TRUE)
)

# Collect metrics
metrics <- collect_metrics(results)
print(metrics)

## KNN Model

knn_spec <- nearest_neighbor(
  mode = "regression",
  neighbors = tune()
) %>%
  set_engine("kknn")
```

```
# KNN workflow
knn_workflow <- workflow() %>%
  add_model(knn_spec) %>%
  add_recipe(data_rec)

# Define tune grid
knn_grid <- grid_regular(
  neighbors(range = c(1, 20)),
  levels = 10
)

# Tune KNN Model
knn_results <- tune_grid(
  knn_workflow,
  resamples = cv_fold,
  grid = knn_grid,
  metrics = metric_set(rmse, rsq),
  control = control_grid(save_pred = TRUE)
)

# Collect metrics
knn_metrics <- collect_metrics(knn_results)
print(knn_metrics)

## Decision Tree Model

tree_spec <- decision_tree(
  mode = "regression",
  cost_complexity = tune(),
  tree_depth = tune()
) %>%
  set_engine("rpart")

# Decision tree workflow
tree_workflow <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(data_rec)

# Define tuning grid
tree_grid <- grid_regular(
```

```
  cost_complexity(),
  tree_depth(),
  levels = 10
)

# Fit to resamples
tree_results <- tune_grid(
  tree_workflow,
  resamples = cv_fold,
  grid = tree_grid,
  metrics = metric_set(rmse, rsq),
  control = control_grid(save_pred = TRUE)
)

# Collect metrics
tree_metrics <- collect_metrics(tree_results)
print(tree_metrics)

## Gradient Boosting Machine Model

gbm_spec <- boost_tree(
  mode = "regression",
  trees = 1000,
  learn_rate = tune(),
  tree_depth = tune()
) %>%
  set_engine("xgboost")

# GBM workflow
gbm_workflow <- workflow() %>%
  add_model(gbm_spec) %>%
  add_recipe(data_rec)

# Tune grid
gbm_grid <- grid_regular(
  learn_rate(range = c(0.01, 0.3)),
  tree_depth(range = c(1, 5)),
  levels = 5
)
```

```
# Fit to resamples
gbm_results <- tune_grid(
  gbm_workflow,
  resamples = cv_fold,
  grid = gbm_grid,
  metrics = metric_set(rmse, rsq),
  control = control_grid(save_pred = TRUE, parallel_over =
"everything")
)

# Collect metrics
gbm_metrics <- collect_metrics(gbm_results)
print(gbm_metrics)


## Linear Regression
lin_reg_spec <- linear_reg() %>%
  set_engine("lm")

# Create the workflow
lin_reg_workflow <- workflow() %>%
  add_model(lin_reg_spec) %>%
  add_recipe(data_rec)

# Fit to resamples
lin_reg_results <- fit_resamples(
  lin_reg_workflow,
  resamples = cv_fold,
  metrics = metric_set(rmse, rsq),
  control = control_resamples(save_pred = TRUE)
)

# Collect metrics
lin_reg_metrics <- collect_metrics(lin_reg_results)
print(lin_reg_metrics)


## GDM full fit

# Extract best hyperparameters
best_gbm_params <- select_best(gbm_results, metric = "rmse")
```

```r
# Print the best hyperparameters
print(best_gbm_params)

# Update GBM model parameters
final_gbm_spec <- boost_tree(
  mode = "regression",
  trees = 1000,
  learn_rate = best_gbm_params$learn_rate,
  tree_depth = best_gbm_params$tree_depth
) %>%
  set_engine("xgboost")

# Update workflow
final_gbm_workflow <- workflow() %>%
  add_model(final_gbm_spec) %>%
  add_recipe(data_rec)

# Fit the GBM model to the full training data
final_gbm_model <- final_gbm_workflow %>%
fit(train)

# Read the test CSV file
test_data <- read_csv("test.csv")

predictions <- predict(final_gbm_model, test)

# Bind the predicted values to the id column in the test data
if("id" %in% colnames(test)) {
  final_output <- test %>%
    mutate(predicted_log_total = predictions$.pred) %>%
    select(id, predicted_log_total)
} else {
  final_output <- test_data %>%
    mutate(predicted_log_total = predictions$.pred)
}

print(head(final_output, 15))

# Create csv file for final output
write_csv(final_output, "final_predictions.csv")
```