STATS 101C Final Project - Regression

2024 Summer

Regression Kaggle Competition Final Project - Amazon Order Survey

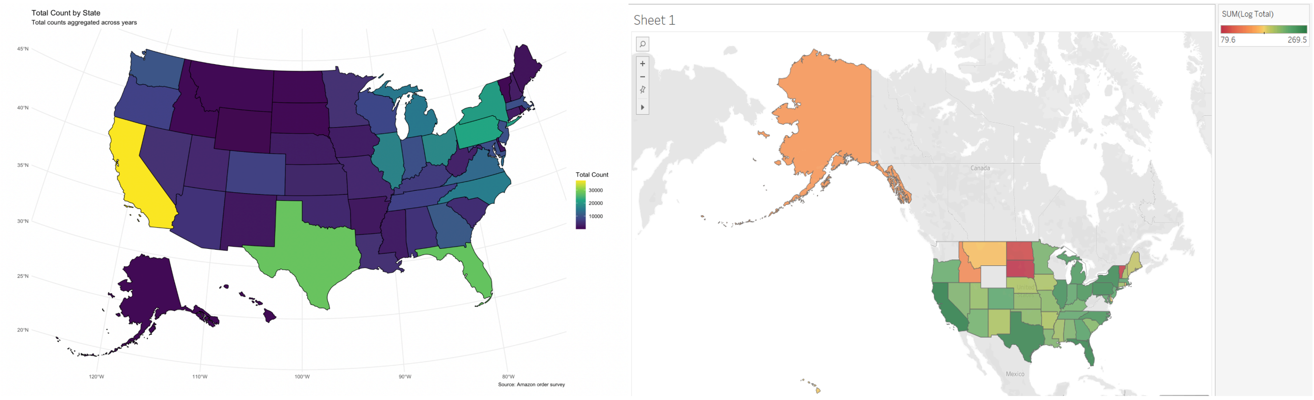
Yechen Cao, Jingran Zhang, Pinyi Li, Vieno Wu

1. Introduction

This report explains the process of constructing regression models for a train dataset through supervised learning. The dataset used in this study is retrieved from Kaggle, for which the data come from a survey conducted on approximately 5000 randomly selected Amazon customers from January 2018 to December 2022. The dataset contains eight files: train.csv, test.csv, customer_info_test, customer_info_train, amazon_order_details_test, amazon_order_details_train, data_descriptions.txt, and sample_submission.csv. For the purpose of this specific study, we use the train.csv and test.csv and modify them accordingly in the process of building the final regression model.

The objective of this study is to build a model such that it will predict the variable log_total, that addresses the skewness in the variable order_totals (calculated by totaling the cost of items in the amazon_order_details files) by taking log-base 10, based on various predictors. According to the data_description.txt, we select the predictor columns that include distinct age levels, income levels, and how many people are using the same account levels with statistical significance to our prediction.

2. Data Analysis



Source: Amazon order survey

Figure 1 Comparison of Total Counts and Log Values by US State (2018-2022)

These two maps show the data for each US state. Counts refers to the number of orders placed. Log value refers to the log of value of orders placed. The graph on the right highlights the total counts from 2018 to 2022 by different states. We can see that California, Texas and Florida have the highest counts among all states, possibly caused by their large population and wealth. The second map shows the total log value among all states. California, Texas and Florida also have the highest value in the United States. These two graphs show similar results, which means there could be a strong correlation between counts and log_total which is worth exploring later.

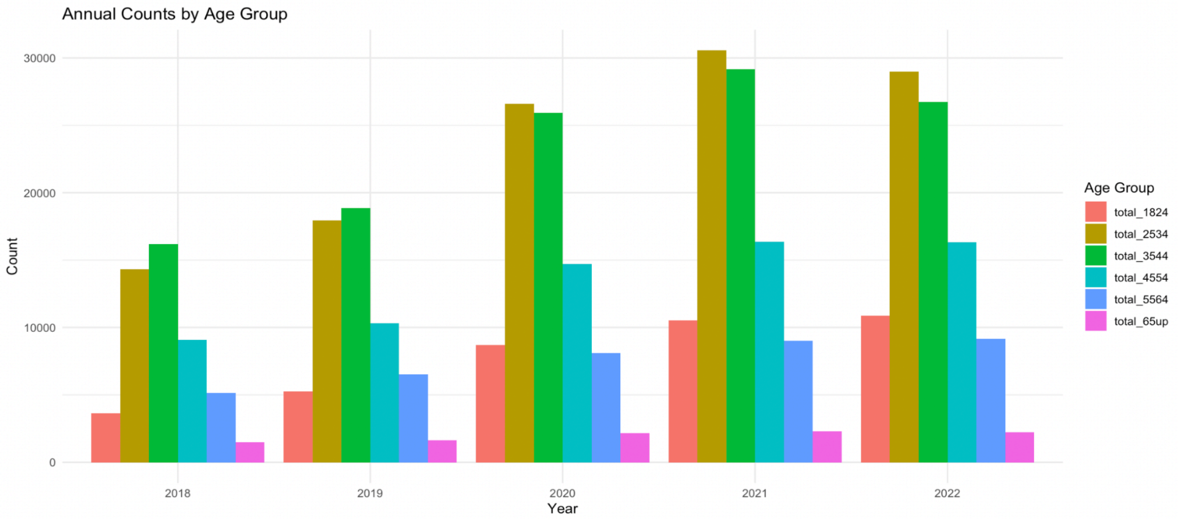


Figure 2 Annual Counts by Age Groups

This graph shows the annual counts by age groups over the year 2018-2022. Counts refers to the number of orders replaced by each age group each year. And the different colors represent different age groups. From the graph, we can see that age 25-34 and 35-44 consistently have the highest counts, while the age 65 and up nearly has minimal counts. Given the extreme low counts of the oldest age group, it might be beneficial to combine it with another group. Also the overall trend indicates counts are increasing over the years.

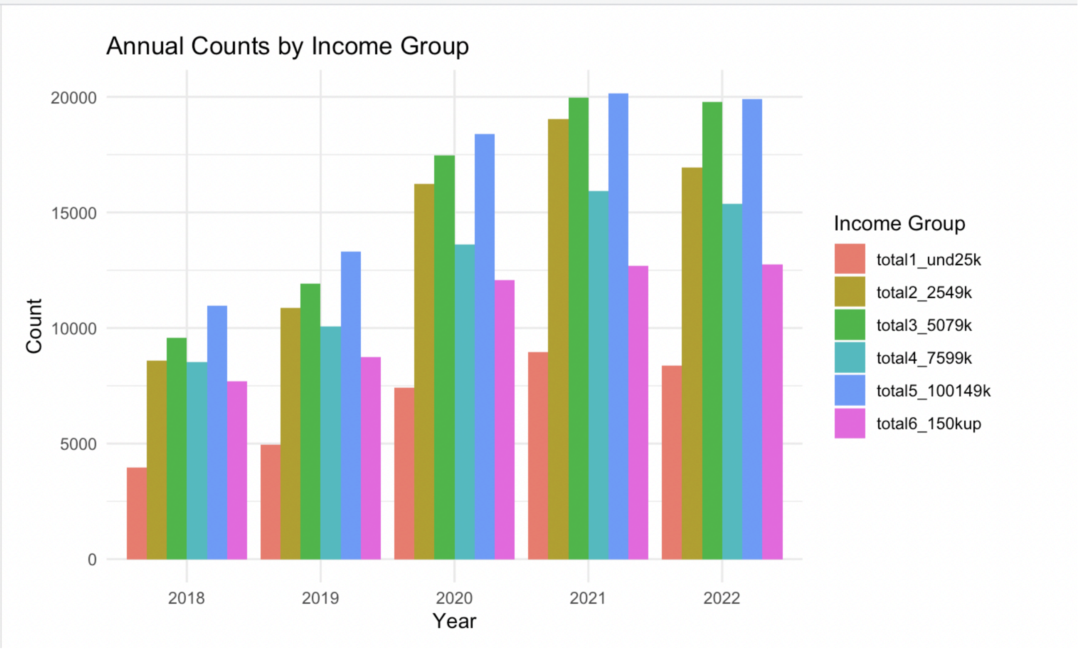


Figure 3 Annual Counts by Income Groups

This graph describes the annual counts by income groups from 2018 to 2022. Counts refers to the number of orders replaced by each income group each year. And the different colors represent different income groups. It highlights that income groups between 25k and 100k show higher counts than the other levels. The overall trend suggests a stable increase for counts for higher income groups over the years.

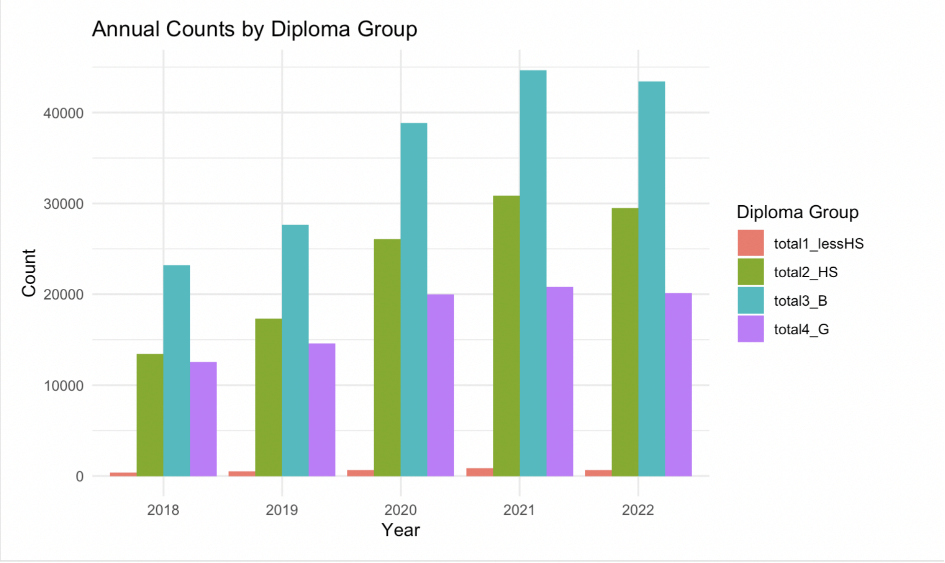


Figure 4 Annual Counts by Diploma Groups

This graph displays the annual counts by diploma groups from 2018 to 2022. Counts refers to the number of orders replaced by each disploma group each year. And the different colors represent different diploma groups. It shows that individuals with bachelor's degrees have the highest counts, while those with a diploma less than high school have nearly 0 counts which suggests that we should combine it with others. Also, the overall trend in counts increases for all groups over the years.

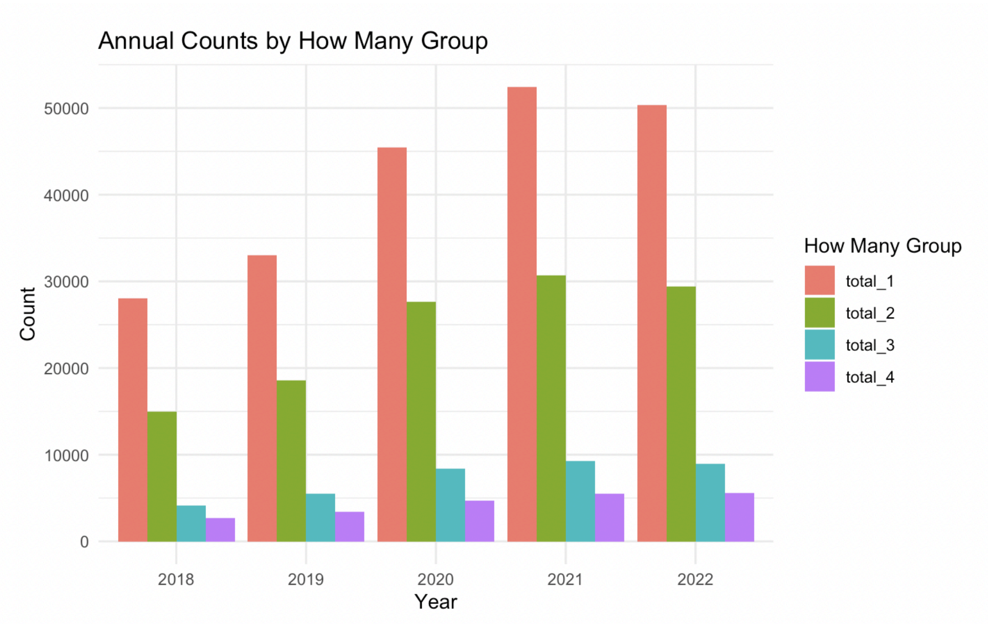


Figure 5 Annual Counts by Number of Users per Account

This graph represents the annual counts by how many people are using the same account (how_many). Counts refers to the number of orders replaced by different how_many groups each year. And the different colors represent different how_many groups. It clearly shows that groups with only one person using the account have much higher counts than the others. The overall trends are similar with the other three graphs.

Combining figures 2-5, we can conclude that there could be a strong correlation between all these four categories and count and year

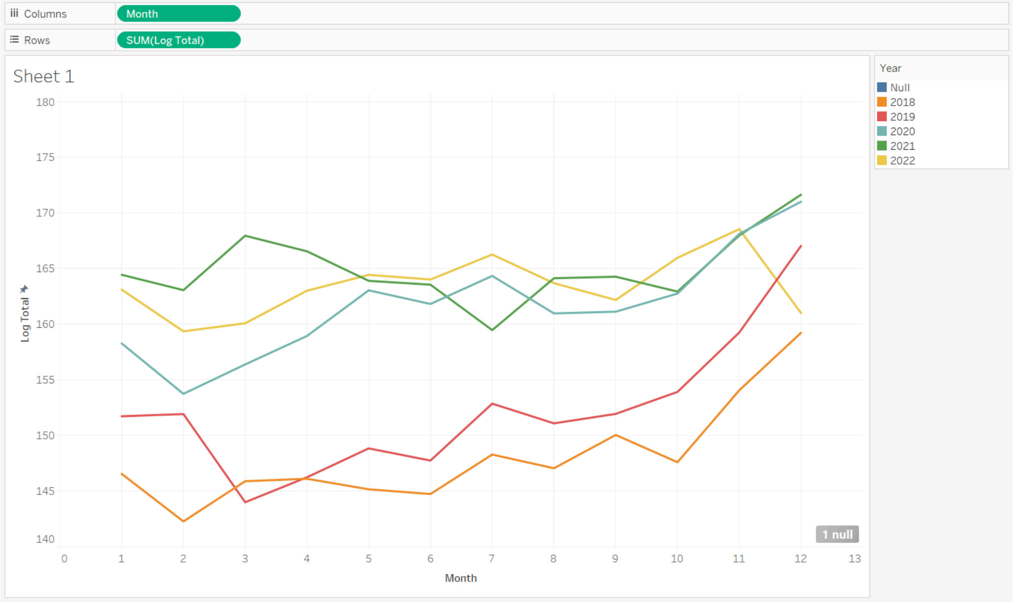


Figure 6: Monthly Trends of Log Total

This line chart shows the trend of the sum of our response variable log_total (which refers to the log of value of orders placed) over the course of 12 months from month January to month December for distinct years from 2018 to 2022. Examining this graph, we can see an overall pattern presented in all of the years such that log_total rises around the last quarter of the year (months October to December) and drops around the month of February. In addition, there exists a noticeable overall increase in the sum of log_total from the year of 2019 to the years it follows. Considering the effect of the real events that occurred during this time period, we can attribute such an increase to the impact of COVID-19 quarantine policy and the effect of economic inflation in the past few years.

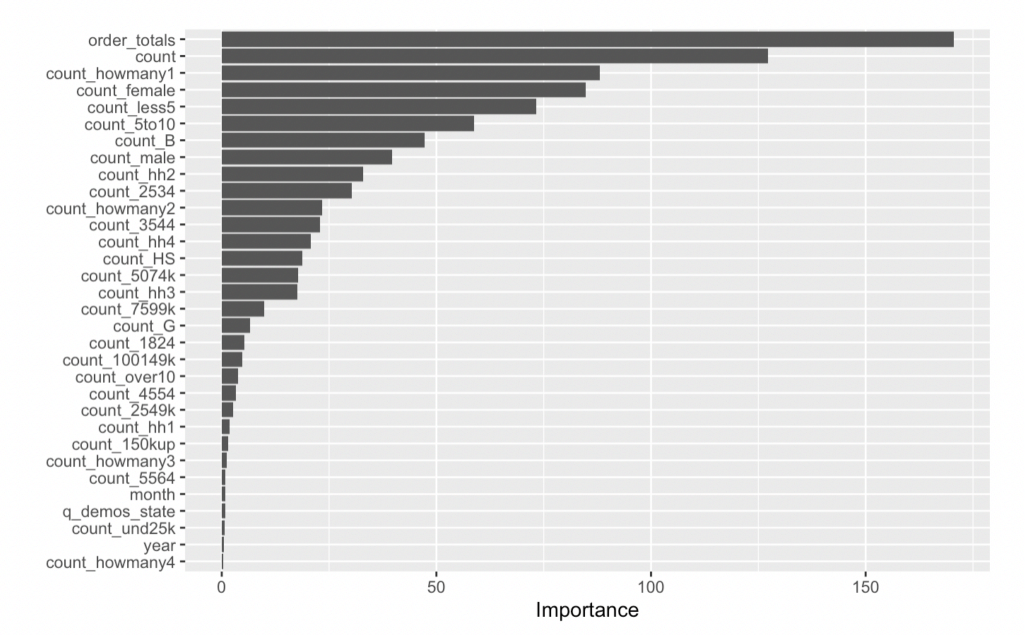


Figure 7 Predictor Importance for Random Forest Model

We used the importance function of the Random Forest model to identify and possibly reduce predictors with little importance in terms of statistical importance to the prediction of log_total. Despite removing order_total (to prevent multicollinearity in the regression fitting), and focusing on top predictors like count, count_howmany1, count_female, and others, our model's performance did not really improve. Therefore, we decided not to proceed with this approach of reducing predictors based on their importance scores.

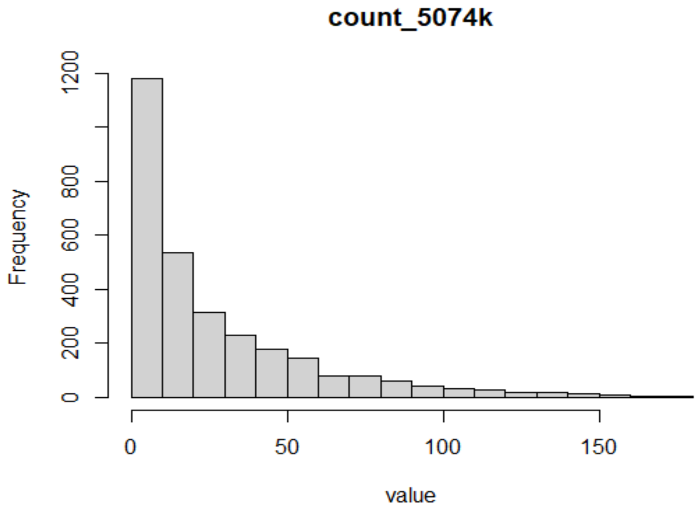


Figure 8a Histogram of Counts for Income Group 50-74k

To better model building in the data preprocessing stage, we combine columns with little to minimal effect in terms of their frequency of different levels of counts so that the newly partitioned levels better address the model fitting. The figure above is a histogram of the frequency distribution of count of the column count_5074k, that is the number of counts in the group of people with income from 50 to 74k. This is an example not suitable for categorizing the counts as low, medium, and high levels (in contrast with the two figures below) since the histogram does not display a clear pattern in distinct levels that can be mapped to low, medium, and high.

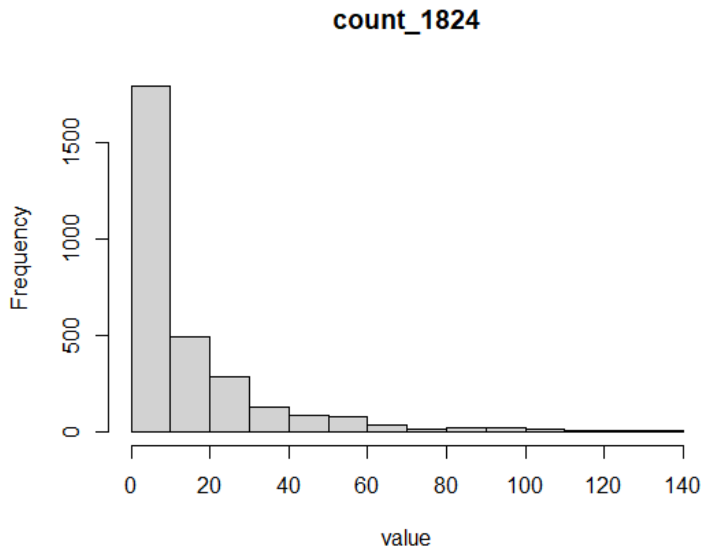


Figure 8b Histogram of Counts for Age Group 18-24

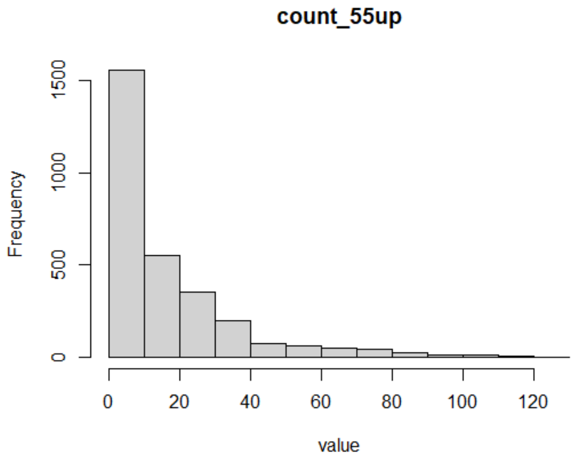


Figure 8c Histogram of Counts for Age Group Greater than 55

In contrast, these two figures, count_1824 (number of counts to people of age 18-24) and count_55up (number of counts to people of age 55 and up), represent frequencies histogram that presents some clear pattern that can be categorized into distinct levels of low, medium and high. Such data transformation from discrete numerical data to categorical data reduces the number of predictors that simplifies regression analysis by avoiding overfitting and multicollinearity. This improves the accuracy of the model when fitting to the regression models.

3. Preprocessing

There are two parts in preprocessing, where the first part involves selecting the appropriate variables and performing transformations and the second part involves defining a recipe that will be used in modeling workflows.

We first remove the predictors that are redundant or irrelevant from the training data. The "count" column is the sum of other columns with different levels so we remove the "count" variable. Next we combine the columns "count_65up" with "count_5565" and call this new column "count_55up"which encodes the counts of orders placed by users with age 55 and above because the "count_65up" column has small entries and we expect the model to generalize better with the combined column. Similarly, we combine the column "count_HSund" with "count_HS" by adding them together to make a new column called "count_HSund"which encodes the counts of orders placed by users with education level of High School and under. We also combined the variables "count_und25k" and "count_2549" following the same logic. By merging these columns, we reduced the number of predictors to 21, which indeed enhances our models performance in later modeling. Then we transformed the numerical counts in some columns to categorical values of "high", "medium", and "low" as there are evident dividing lines within each column. The numerical counts from "count_HSund" to "count_G" and from "count_1824" to "count_55up"are assigned levels of "high", "medium", and "low" according to the distribution of counts within the columns. Figure 8b, a histogram of the predictor "count_1824", shows a clear pattern of low,medium, and high where "low" is assigned to counts under 10, "medium" is assigned to counts above 10 but under 60 and "high" is assigned to counts above 60. Similarly, Figure 8c also shows a clear pattern of low, medium, and high in variable "count_B". Thus, we assign "low" to counts under 50, "medium" to count between 50 and 100 and "high" to counts above 100. After this transformation, we are left with 30 columns where some are numerical counts, some are categorical levels.

We defined a recipe that includes step_dummy, step_impute_mean, step_zv, and step_rm(aliased_vars). Step_dummy converts all categorical predictors into dummy encoded variables. This step is necessary as most machine learning models require numeric inputs. Step_impute_mean imputes missing values in numeric predictors with the mean value of each predictor, which handles the missing values in the data if any. Step_zv removes all variables that have zero variance, so predictors with the same values for all observations are removed. Step_rm(aliased_vars) removes variables identified as collinear and ensures that no two predictors are perfectly collinear. This recipe is used for all of our models except the Neural Network model where an extra step of normalization is added for Neural Network models.

4. Model Selection

Table 1 Summary of Candidate Models

Model Identifier	Type of Model	Engine	Recipe Steps	Hyperparameters
lr_model	Linear Regression	lm	Same as svm_rbf	No hyperparameter
lasso_spec	Lasso regression	glmnet	Same as svm_rbf	penalty

				mixture
lasso_trans	Lasso Regression	glmnet	Same as svm_rbf	penalty mixture
svm_rbf	Support vector machine with gaussian radial basis function kernel	kernlab	Dummy encoding Mean imputation Removal of near zero variance Removal of aliased variables	cost rbf_sigma margin
rf_model_notune	Random forest	ranger	Dummy encoding Mean imputation	No hyperparameter
rf_model	Random forest	ranger	Same as svm_rbf	mtry trees min_n
xgboost_model	eXtreme Gradient Boosting	lightgbm	Same as svm_rbf	learn_rate loss_reduction
nn_model	Neural Network	nnet	Same as svm_rbf with an extra step_normalize()	hidden_units penalty
model_stack	Combination of several model above	/	/	/

The six models used in our regression analysis can be divided into two categories, with tuning and without tuning. For each model analysis, the model is first defined and a recipe for data preprocessing is created. The workflow is constructed by combining the model and the recipe.

For the models that do not require tuning, that is, linear regression model and lasso regression model, a 10 fold cross-validation is performed to evaluate the model's performance using RMSE, Root Mean Square Error which is the measurement for error between predictions and actual value, and R-squared, which explained the variance that explained by the prediction model, metrics. After evaluating the model, the final model is fitted to the training dataset and predictions are made on test data.

For models that require hyperparameter tuning, hyperparameter tuning is performed using a grid search (grid_random) or bayes search. A 10 fold cross-validation is performed to identify the best model parameters based on RMSE. After selecting the best model, the workflow is finalized and evaluated through cross-validation. The final model is fitted to the training data, and predictions are made on the test data.

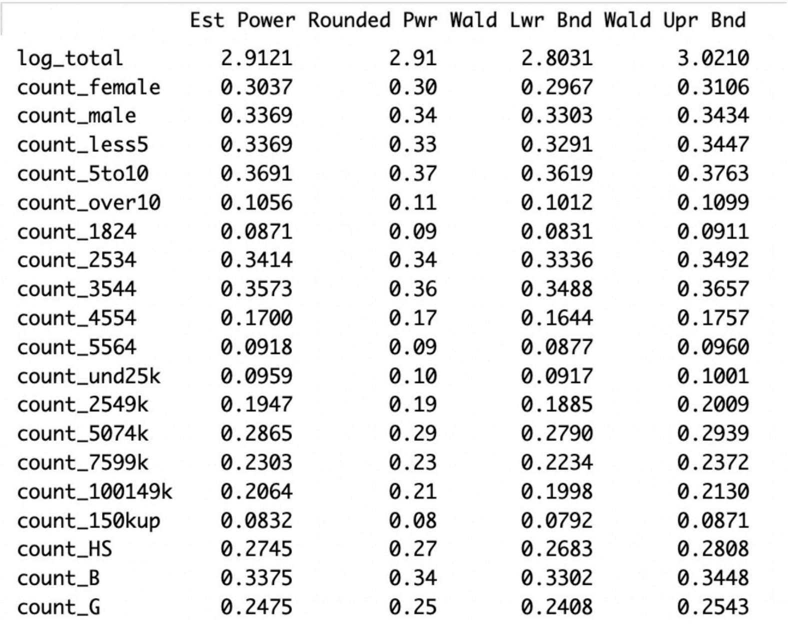
Linear Regression

Linear regression model assumes linear relationships between predictors and the outcome variable. We applied the linear regression model (lr_model) and used our preprocessing training data to fit the model with cross validation. However, the model doesn't give out a good estimation for the test data, so we switched to other more complex methods.

Lasso Regression

Lasso regression is a type of linear regression that performs variable selection and regularization to prevent overfitting and improve model accuracy. For the regression model (lasso_spec), we set the fixed penalty parameter to be 0.1, and the mixture to be 1. Using this model, we try to avoid the complex hyperparameter tuning, and merely focus on linear model training. However, this model does not give a good prediction for the test data. We decided to do some linear transformation to the dataset to improve model performance.

Lasso Transformation



```
               Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
log_total         2.9121        2.91       2.8031       3.0210
count_female      0.3037        0.30       0.2967       0.3106
count_male        0.3369        0.34       0.3303       0.3434
count_less5       0.3369        0.33       0.3291       0.3447
count_5to10       0.3691        0.37       0.3619       0.3763
count_over10      0.1056        0.11       0.1012       0.1099
count_1824        0.0871        0.09       0.0831       0.0911
count_2534        0.3414        0.34       0.3336       0.3492
count_3544        0.3573        0.36       0.3488       0.3657
count_4554        0.1700        0.17       0.1644       0.1757
count_5564        0.0918        0.09       0.0877       0.0960
count_und25k      0.0959        0.10       0.0917       0.1001
count_2549k       0.1947        0.19       0.1885       0.2009
count_5074k       0.2865        0.29       0.2790       0.2939
count_7599k       0.2303        0.23       0.2234       0.2372
count_100149k     0.2064        0.21       0.1998       0.2130
count_150kup      0.0832        0.08       0.0792       0.0871
count_HS          0.2745        0.27       0.2683       0.2808
count_B           0.3375        0.34       0.3302       0.3448
count_G           0.2475        0.25       0.2408       0.2543
```

Figure 9 Transformation of Variables

To get better predictions, we use powertransform to reduce skewness and stabilize variance. We apply rounded estimates for every predictor on the table, and perform a transformed lasso regression. For the transformed model (lasso_trans), the hyperparameter lambda is tuned using grid search to minimize the rmse. Lambda specifies the degree of regularization penalty for the model. The range of lambda for tuning is set to (1, 1*10^4). After data transformation and model tuning, from the bellowing Table 3, we can see that lasso_trans doesn't outperform the lasso_spec model. However, on Kaggle, lasso_trans does have a better score, suggesting that there is overfitting in the lasso_spec model. Nevertheless, it is still not better than the non-linear models. Thus, we decide to give up on the Lasso Regression Models.

SVM RBF Model

Support vector machine with gaussian radial basis function as the kernel uses nonlinear function of predictors to do regression analysis. It can optimize the loss function and make it only affected by very large model residuals. The two hyperparameters, cost and RBF sigma, in the model (svm_rbf) are tuned through grid search. Cost specifies the cost of predicting a sample within or on the wrong side of the margin. RBF sigma specifies the width of the RBF kernel. The range of cost for tuning is set to (0, 20), and the range of RBF sigma for tuning is set to (0, 0.6). The other hyperparameter margin, which specifies the epsilon in the SVM insensitive loss function for regression, is fixed to 0.01 in this model.

Random Forest Model without Tuning

Random forest constructs a multitude of decision trees at training time and then combines the outputs into a single result. We first tried an untuned random forest model (rf_model_notune) and applied dummy encoding and mean imputation as a recipe. We used our preprocessing training data to fit the model with cross validation. The model gave out a good estimation for the test data, so we selected this as one of our final models.

Random Forest Model

Given the good performance of the untuned random forest model, we tried to tune the model and expected for a better performance. For this model (rf_model), firstly, three hyperparameters, trees, mtry, and min_m, are tuned through grid search. Trees denotes the number of trees contained in the model; mtry specifies the number of predictors that will be randomly sampled at each split when creating the tree models; min_n specifies the minimum number of data points in a node that are required for the node to be split further. The range of trees for tuning is set to (50, 200), the range of mtry for tuning is set to (1, 5), and the range of min_n for tuning is set to (5, 20).

Beside grid search (grid_random), we also tried grid_regular and bayes tuning method. For bayes tuning, we set the initial to be 5, the iteration to be 50, and other hyperparameters same as grid_random. After applying prediction to the best model using the minimum rmse for the top 10 models, we find out that there is not much difference in rmse between different tuning methods for our random forest model. However, when we are comparing the rmse and rsq of other models, random forest is one of the models with smallest rmse and highest rsq. Therefore, we decided to use this model as one of our final models.

XGBoost with lightgbm

XGBoost builds a collection of decision trees in a sequential manner. In regression, the model makes an initial guess of the prediction by taking the mean of all observations. Then it adds decision trees based on the residuals of previous trees. The final prediction is a combination of all predictions from the trees.

The XGBoost model has hyperparameters like learning rate and loss reduction. Learning rate scales the contribution of each tree to the final prediction. Loss reduction (gamma) specifies the minimum loss reduction required to make a further divide on a node of the tree. These two parameters are tuned using iterative search with the tune_bayes() function, where 20 iterations of Bayes optimization are conducted after the initial search.The optimal parameters are found to be 1.023664 for learning rate and 1.226358 for loss_reduction.

Neural Network Model

Neural network is a model that optimizes the network's parameters to minimize the difference between the predicted outcome and the actual target values. It is adept at capturing non-linear relationships between predictors. Thus, we employed NN to preserve the possible nonlinearity between the predictors and the outcome.

For the model (nn_model), we implemented the nnet engine to train a multilayer perceptron (MLP), a feedforward neural network. There are several hyperparameters, including the type of activation function, number of hidden units and penalty associated with loss function. Using random grid search, the optimal parameters are found to be 4 for hidden units, and 1.05958 for penalty. The activation function is set to be the default, sigmoid.

Stacking Model

We use tidy model stacks to ensemble the output of several models together and generate a new model to better fit the data. In our stacking model, we use linear regression model, neural network model, random forest model, and xgboost model as candidate members to predict the output. As we can see from the table, the rmse for model stacking is much smaller than the others, but the prediction for the test is not better than the others. We think the stacking model is overfitting the training data.

Summary table

We evaluated each model with fit_resamples and set the metric to be rmse and r-square. This function fits the model to each resample (our cross-validation folds) and computes performance metrics, allowing us to assess the model's generalization ability.

The rmse and standard error of each model are extracted and compared. The details are shown in the table below.

Table 3 Summary of RMSE and Standard Error of Candidate Models

Model Identifier	Metric Score (RMSE)	SE of metric
lr_model	0.1435065	0.003854239
lasso_spec	0.2695638	0.003652302
lasso_trans	0.5720427	0.005643523
svm_rbf	0.2724036	0.004534390
rf_model_notune	0.1130457	0.001896869
rf_model	0.1248761	0.002357312
xgboost_model	0.1416653	0.003665463
nn_model	0.1226933	0.002988884
model_stack	0.08516663	NA

5. Final Model

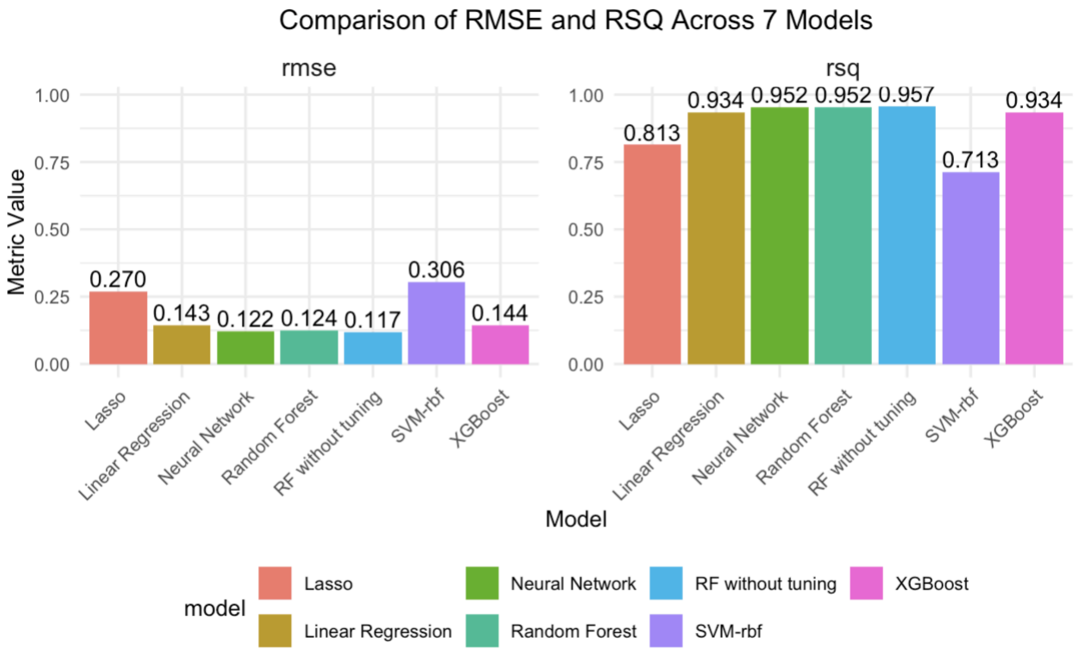


Figure 10 Comparison of RMSE and R^2 Across All Models

According to the public scoreboard, our best performing models are the Bayesian random forest with a score of 0.01748. Our final model selection, taking into account potential overfitting to the dataset, would be the Random forest model (without tuning) with a public score of 0.01805. After appropriate data preprocessing, recipe setup, and 10-fold cross validation setup described earlier, we set up the Random forest regression model from the ranger package. The strength of using Random forest model is its powerful algorithm that can capture nonlinear relationships between the predictors and the response variable in large, complex dataset like this amazon purchase dataset. This is one of the best-fitted models given our transformed train data with 31 variables and 2,952 rows. In addition, the skewness of our model is also addressed by the model's high interpretability on non-linear relationships. However, with such a strong model, one weakness is overfitting over train data that performs poorly on unseen data. We address this issue by selecting a model that does not have the best performance score in the public score. Another disadvantage in selecting this model is time-consuming in training and predicting large datasets, but the high performance makes it worthy to be selected in the final model.

Some potential improvements to our model building include more effective tuning techniques since our current tuning shows no significant improvement compared to the untuned model. In addition, our model is based primarily on the train dataset. The regression fitting might be improved if we include more data from other files, like customer_info and amazon_order_details files. Selecting appropriate columns from these external datasets is also crucial in building our supervised learning model.

---

6. Appendix: Final annotated script & Team member contribution

Using Windows 11 for running prediction

Final Annotated Script 1: Random Forest not tuned

```
# load our original train dataset
train <- read_csv('train.csv',show_col_types = FALSE)

#setting randomness for set.seed
RNGkind("Mersenne-Twister")  # Most common and default in R

#combine several columns with others (count_65up, count_und25k, count_lessHS)
train <- train %>%
  mutate(count_55up = count_5564+count_65up)%>%
  mutate(count_49kund = count_2549k+count_und25k) %>%
  mutate(count_HSund = count_lessHS + count_HS)%>%
  select(-count_5564,-count_65up,-count_2549k,-count_und25k)%>%
  relocate(count_55up,.before=count_5074k)%>%
  relocate(count_49kund,.before=count_5074k)%>%
  relocate(count_HSund,.before=count_B)%>%
  select(-count_HS,-count_lessHS)

# change all age and degree column to categorical variables
train <- train %>%
  mutate(count_1824_cat = case_when(
    count_1824 <= 10 ~ 'Low',
    count_1824 <= 60 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_2534_cat = case_when(
    count_2534 <= 20 ~ 'Low',
    count_2534 <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_3544_cat = case_when(
    count_3544 <= 40 ~ 'Low',
    count_3544 <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_4554_cat = case_when(
    count_4554 <= 10 ~ 'Low',
    count_4554 <= 80 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_55up_cat = case_when(
    count_55up <= 10 ~ 'Low',
    count_55up <= 40 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_HSund_cat = case_when(
    count_HSund <= 20 ~ 'Low',
    count_HSund <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
train <- train %>%
  mutate(count_B_cat = case_when(
    count_B <= 50 ~ 'Low',
    count_B <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
train<- train %>%
  mutate(count_G_cat = case_when(
    count_G <= 20 ~ 'Low',
    count_G <= 90 ~ 'Medium',
    TRUE ~ 'High'
  ))

# Deleting transformed numerical columns
train <- train %>%
  select(-order_totals,
-count_HSund,-count_B,-count_G,-count_1824,-count_2534,-count_3544,-count_4554
         ,-count_55up)

# Doing the same thing for our test dataset
test <- read_csv('test.csv')

#Combine the same column as trainning do
test <- test %>%
  mutate(count_55up = count_5564+count_65up)%>%
  mutate(count_49kund = count_2549k+count_und25k) %>%
  mutate(count_HSund = count_lessHS + count_HS)%>%
  select(-count_5564,-count_65up,-count_2549k,-count_und25k)%>%
  relocate(count_55up,.before=count_5074k)%>%
  relocate(count_49kund,.before=count_5074k)%>%
  relocate(count_HSund,.before=count_B)%>%
  select(-count_HS,-count_lessHS)

# Converting numeric values to categoricals ones for age and degree group
test <- test %>%
  mutate(count_1824_cat = case_when(
    count_1824 <= 10 ~ 'Low',
    count_1824 <= 60 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_2534_cat = case_when(
    count_2534 <= 20 ~ 'Low',
    count_2534 <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_3544_cat = case_when(
    count_3544 <= 40 ~ 'Low',
    count_3544 <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_4554_cat = case_when(
    count_4554 <= 10 ~ 'Low',
    count_4554 <= 80 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_55up_cat = case_when(
    count_55up <= 10 ~ 'Low',
    count_55up <= 40 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_HSund_cat = case_when(
    count_HSund <= 20 ~ 'Low',
    count_HSund <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_B_cat = case_when(
    count_B <= 50 ~ 'Low',
    count_B <= 100 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  mutate(count_G_cat = case_when(
    count_G <= 20 ~ 'Low',
    count_G <= 90 ~ 'Medium',
    TRUE ~ 'High'
  ))
test <- test %>%
  select(-count_HSund,-count_B,-count_G,-count_1824,-count_2534,-count_3544,-count_4554
        ,-count_55up)

#set up a cv_fold for all of our models with v = 10
set.seed(513)
folds <- vfold_cv(train, v = 10, strata = log_total)

# set up the engine and mode for random forest model
library(ranger)
```

```
rf_model <- rand_forest( ) %>%
  set_mode("regression") %>%
  set_engine("ranger")

# create recipe, and change categorical to numeric, and remove NA values
re_recipe <- recipe(log_total ~ ., data = train) %>%
  step_dummy(all_nominal_predictors(),-all_outcomes()) %>%
  step_impute_mean(all_numeric_predictors(), -all_outcomes())

# create workflow with rf_model and recipe
rf_workflow <- workflow() %>%
  add_recipe(re_recipe) %>%
  add_model(rf_model)

#fitting the workflow to the set of resamples = 10 folds
library(tune)
fitted_data <- fit_resamples(rf_workflow, resamples = folds)

fitted_data %>%
  collect_metrics()

# fit the model to all of the training dataset
rf_workflow_fit <- fit(rf_workflow, data = train)

# Use the model to predict the test dataset
test_pre <- predict(rf_workflow_fit, new_data = test)

# create a dataframe to collect IDs and predictions
results <- test %>%
  select(id) %>%
  bind_cols(test_pre)

head(results, n = 15)

#output predictions
write_csv(results, "prediction_notune.csv")
```

Final Annotated Script 2: Random Forest Model Tuned

Same Data preprocessing, using different model:

```
#delete unimportant columns
train <- train %>%
  select (- count_howmany4,- count_over10, -count_hh1, - count_hh2,-count_hh3, - count_hh4, - count_howmany1, - count_howmany2, - count_howmany3)

test <- test %>%
  select (- count_howmany4,- count_over10, -count_hh1, - count_hh2,-count_hh3, - count_hh4, - count_howmany1, - count_howmany2, - count_howmany3)


#set up a cv_fold for all of our models
set.seed(100)
cv_fold <- vfold_cv(train, v = 10, strata = log_total)

# set up the engine and mode for random forest model
set.seed(100)
rf_model <- rand_forest() %>%
  set_engine('ranger')%>%
  set_mode('regression')

# create recipe, and normalize all predictors
amazon_recipe <- recipe(log_total ~. ,data=train)%>%
  step_impute_mean(all_numeric(),-all_outcomes())%>%
  step_dummy(all_nominal_predictors(),-all_outcomes()) %>%
  step_nzv(all_predictors())

# create workflow with rf_model and recipe
rf_wkfl <- workflow() %>%
  add_model(rf_model)%>%
  add_recipe(amazon_recipe)

# set up a new model with grid tuning parameters
set.seed(100)
rf_tune_model <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) %>%
  set_engine('ranger')%>%
  set_mode('regression')

# set up a new workflow with the tuning parameters
rf_tune_wkfl <- rf_wkfl %>%
  update_model(rf_tune_model)
```

```
# Define the grid of hyperparameters
set.seed(100)
rf_grid <- grid_random(
  mtry(range = c(1, 5)),
  trees(range = c(50, 200)),
  min_n(range = c(5, 20)),
  size = 20
)

# Perform the tuning
set.seed(100)
rf_tuning <- tune_grid(
  rf_tune_wkfl,
  resamples = cv_fold,
  grid = rf_grid,
  metrics = metric_set(rmse)
)

# select the best model out of 5 candidates using rmse
rf_tuning %>%
  show_best(metric = 'rmse',n=5)

best_rf_model <- rf_tuning %>%
  select_best(metric = 'rmse')

# create the newest workflow with our best candidates
final_rf_wkfl <- rf_tune_wkfl %>%
  finalize_workflow(best_rf_model)


# fit the trainning data to our best model with vfold = 10, and calculate the rmse, rsq for our model
set.seed(100)
final_rf_res <- fit_resamples(
  final_rf_wkfl,
  resamples =cv_fold,
  metrics = metric_set(rmse, rsq)
)

final_rf_fit <-final_rf_wkfl %>%
  fit(data=train)


#using the best model to predict by test dataset
rf_predictions <- final_rf_fit %>%
  predict(new_data=test)

# create a dataframe with id and our new prediction
results <- test %>%
  select(id) %>%
  bind_cols(rf_predictions) %>%
  rename(id = id, log_total = .pred)

# check the predictions
head(results,15)

# write a csv file with the id and prediction.
write_csv(results,'predictions_rf_3.20.3.15.csv')
```

Team Member Contribution:

Yechen Cao: I am responsible for some of the visualizations, and improving the Lasso model, random forest model and stacking models. And also writing reports for those.

Vieno Wu: I am responsible for result visualizations, the construction of xgboost model, neural network model and stacking model. I am responsible for some of the data preprocessing. I also wrote the corresponding parts in this report.

Pinyi Li: I am responsible for some visualizations, and writing for the linear regression and svm rbf model. I also write reports for the models.

Jingran Zhang: I am responsible for data visualizations, support for data preprocessing and model testing. I am also responsible for the introduction, part of data analysis, and final model section of the report writing.