Enhancing E-commerce Operations through Predictive Modeling and Churn classification

DataInsightX

Pratik Ghawate (827910727)

Tanvi Patil (828408419)

Harshal Patil (827817153)

Christopher Flatley (828570373)

Fowler School of Business

San Diego State University

MIS 720: E-Business Infrastructure: Data Science and Big Data

Dr. Aaron Elkins

Date: December 13th, 2023

Executive Summary

We examine a thorough data analysis of the Olist Store dataset in this project report. This dataset is an extensive collection of data about 100,000 orders that were placed on the Brazilian e-commerce platform between 2016 and 2018. The dataset, which comes from a variety of Brazilian marketplaces, offers a sophisticated understanding of a number of aspects, such as order dynamics, pricing schemes, payment procedures, delivery efficiency, consumer locations, product characteristics, and customer feedback. Recognizing that the dataset contains real commercial data that has been anonymized in order to protect sensitive information is important. The objective of this investigation is to derive significant insights that enhance comprehension of e-commerce operations.

In the process of preparing our dataset, our primary objective is to identify and retain customers who remain actively engaged with business. To achieve this, we focus exclusively on customers whose orders have been successfully delivered. Recognizing that missing delivery dates may indicate orders in different stages of processing, such as being shipped, canceled, or facing availability issues, we adopt a strategy to exclude records with missing datetime values, presuming that these orders were not delivered or are still undergoing various stages of fulfillment.

For records with missing delivery dates, we consider them as indicative of orders that have not been delivered or are in different stages of the fulfillment process, such as being shipped, canceled, or unavailable. Consequently, we decide to eliminate these records from our analysis to maintain the integrity of our dataset. Furthermore, when addressing missing values in datetime fields, our imputation strategy is selective. We only impute null values for orders that have been successfully delivered and exhibit missing values in any of the datetime fields. This

focused approach ensures that our data preparation steps align with our goal of identifying and retaining customers within our active business sphere

We investigated the use of random forest, decision tree, and logistic regression algorithms in model building to forecast results. A baseline that captured linear relationships between variables was provided by logistic regression. Clear decision paths and interpretability were provided by decision trees. The most promising model was the random forest model, which achieved a higher accuracy of 67% and AUC score of 0.727 which was highest among all the models. Random forest was the best option because it reduced overfitting and increased predictive power by utilizing an ensemble of decision trees. Its strong performance indicates that it can identify intricate patterns in the data, proving that it is effective in reaching a respectable degree of accuracy and that it is appropriate for our predictive modeling assignment.

The results highlight the efficacy of the random forest model in addressing the problem statement. The 67% accuracy underscores its suitability for predictive tasks. Insights suggest focusing on enhancing features contributing to predictive power. To capitalize on these findings, we consider refining data collection for crucial variables and explore advanced ensemble methods. Recommendations include ongoing model monitoring for adaptability. Aligning with the initial problem statement, the results reinforce the importance of leveraging robust predictive models for improved decision-making, guiding future actions towards a more refined, feature-rich approach to better address the specified problem.

RFM (Recency, Frequency, Monetary) analysis is a pivotal method empowering businesses to comprehend customer behavior. Recency assesses the time since the last purchase, segmenting customers by engagement proximity. Frequency gauges order occurrences, highlighting repeat purchasing trends. While some customers may show a frequency of 1, it remains insightful, especially when coupled with other metrics or in industries fostering repeat sales. Monetary value reflects the total spent by customers, pinpointing high-value patrons and their revenue contribution.

A churn prediction model utilizing Recency as the target variable was devised to categorize customers exceeding the average Recency as 'churned.' This enables businesses to foresee potential churners, prompting proactive strategies like tailored marketing, personalized incentives, or enhanced customer service. Such preemptive measures aid in customer retention, curtailing overall churn rates and fortifying long-term business sustainability. Identifying and addressing potential churners helps fortify customer relationships, fostering sustained business growth and customer loyalty.

Discovery and Data Preparation

The data set we used for our analysis contains a multitude of different data types, suited for a variety of different model types and hypotheses. The data originated from the Brazilian e-commerce system, which released various order information data sets from the Olist store. The data sets included information from 100,000 orders made within the Olist site network, containing columns such as price, delivery status, and consumer reviews.

Our source data contained 99,441 rows and 52 columns, all containing different categories of data. The first type of data observed was columns containing text data, compiled from a variety of consumer reviews and ratings. Many of the columns contained numeric data, including but not limited to prices and qualities of items, date shipped and received, as well as the cost of processing. Lastly, we had columns containing categorical data, such as the status of an order.

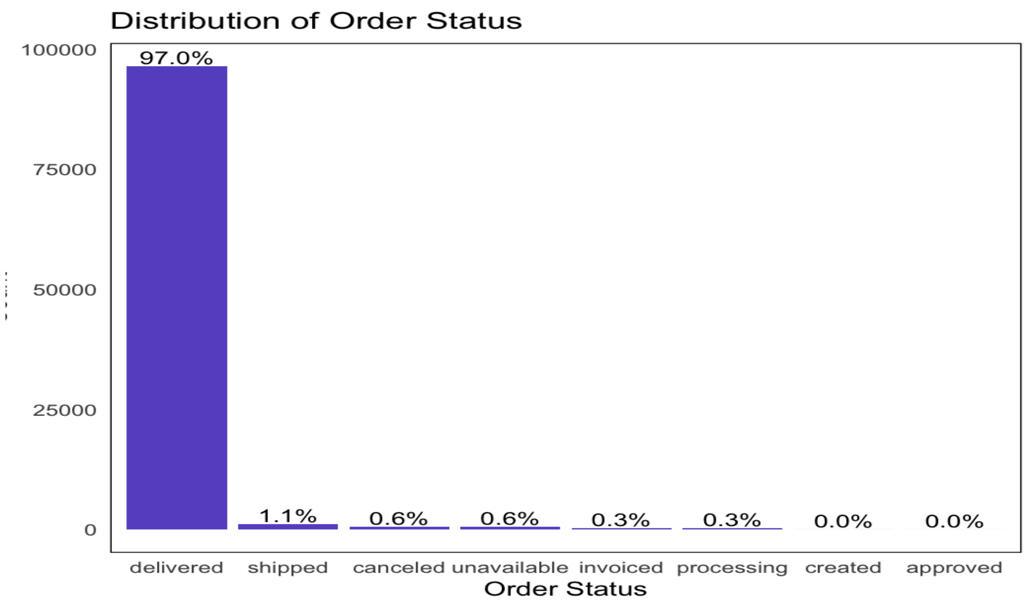


Figure 1: *The Distribution of Order Status for Observed Values*

A large part of our data planning model was to only focus on the orders that had reached their final state of processing. This would allow us to narrow down the orders that would contribute to sales growth over the period of observations between 2016 and 2018.

We began the data preparation process by reading in the various data sets, joining them together, and beginning the cleaning process. It was essential for us to clean the data of any missing information. We began to drop any rows or columns with missing data or data that was not helpful to the contribution of our overall regression analysis, as well as any orders that did not reach the final stage of the fulfillment process as we are trying to predict only the realized sales.

Following this process we created new columns to allow for an easier analysis process, we used existing columns such as the time an order was shipped, and calculated the difference between when the order arrived. This process provides a single numeric digit that we can then apply to the regression analysis. We repeated this process multiple times to create the column values for the “Purchase Approved”, “Approved Carrier”, “Carrier Delivered”, “Delivered Estimate”, and “Purchase Delivered”. These created columns would become the core for the models we would begin to build throughout our analysis project. Following the creation of these key columns, we began to further clean the data of any rows containing unrealistic or altered data. These filtered rows would include orders that were delivered immediately after their estimate, indicating a gap in the accuracy of the shipping date, and others similar. By factoring out these illegitimate entries, it allows us to create our model with the most accurate information to predict sales, as it removes most of the rows that would create outliers directly from the beginning of the model-building process.

The last steps we took during the data preparation process was to outline the various columns that we would focus on to build or analysis models. We decided that information such as the state and zip code would allow us to see where orders were increasing over time, which areas sold the most, and how different states were impacted by shipping times. We wanted to also account for logistic challenges of shipping orders, so we included the use of the parameters such as size and weight of the items being shipped, as well as the geolocations for each of the orders. The payment process is also a key factor in the efficiency of the online ordering system. To account for this in the model we kept the records of payment type, the amount of time payment took to be approved, and the overall prices for each order.

Model Planning and Building

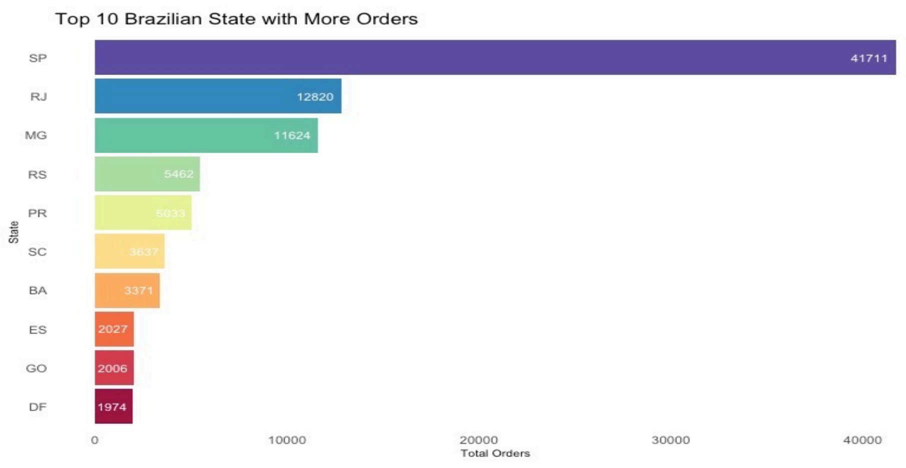


Figure 2: *Top 10 Brazilian States by Order Quantity*

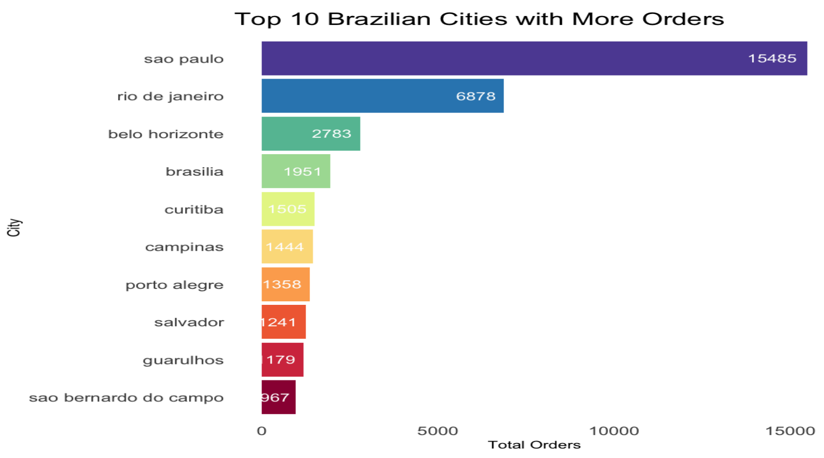


Figure 3 : *Top 10 Brazilian Cities by Order Quantity*

The data set we used was set in the country of Brazil. We needed to understand how different parts of the country interacted with the online market system to see if this would be a relevant topic to further base our model around. By understanding various market trends, broken down by both state and city, it would allow us to predict growth by area, and later predict where sales would increase or decrease in the future. The impact that these findings have is significant as it will allow us to target marketing resources to help boost regions that are underperforming according to our analysis.

From January to July 2018, sales of Brazilian e-commerce increased by a significant 154%, indicating a booming market. Through an analysis of the data, we created a plot that would show us where orders were being placed based on the Geolocation of the shipping address. We created this model with the columns defined in the data preparation process and found that Sao Paulo state, São Paulo city, and the Centro-Oeste region are all experiencing rapid growth, suggesting these areas as key locations for corporate expansion.

Another key piece of observations for our data set is understanding the types of products most commonly sold through this online market space. To do this, we created a code to arrange the orders into product categories, then order it from the least purchased to the most. In order to gain a visual representation of this information, we created a bar graph to show the frequency of orders within each category for the data set as a whole. We chose not to break down this model by geolocation because we assumed that this would not have a significant impact when predicting the future sales.

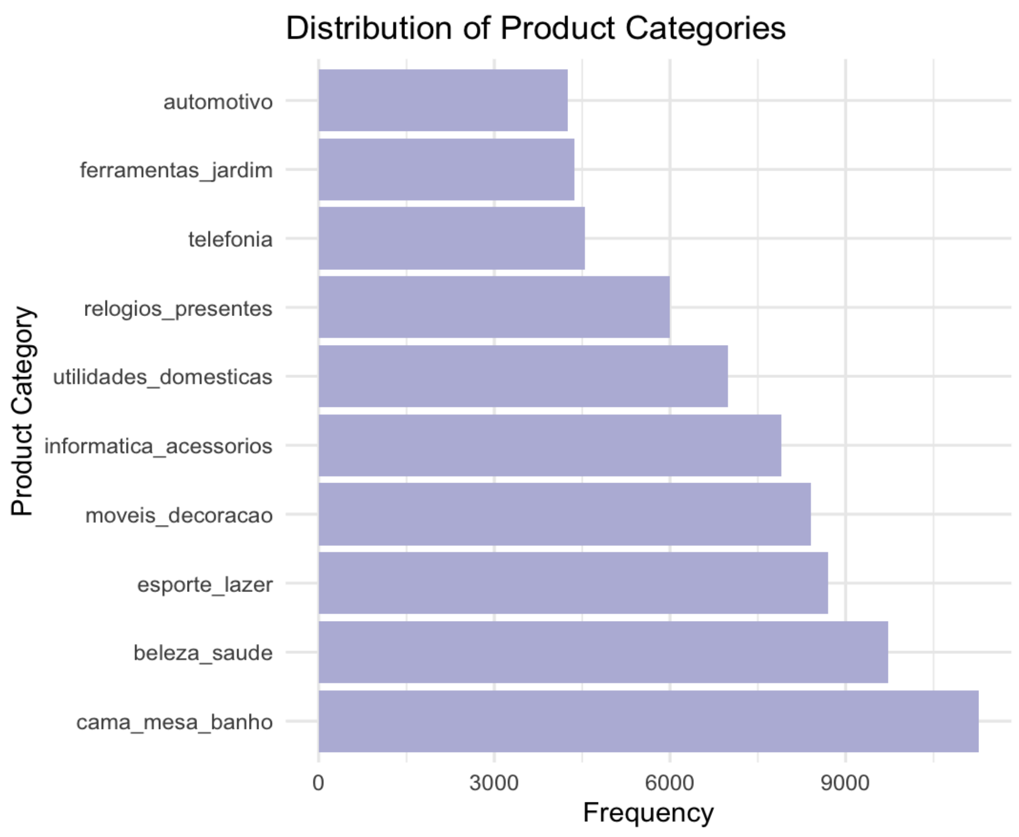


Figure 4 : *The Distribution of Fulfilled Products by Product Category*

What we observed through this process was that "Bedding, Bath, and Table" products were the most frequently ordered category. This was followed by the "Health and Beauty" category, and lastly the "Sporting Goods" category. This information would show us that the highest frequency of orders within this marketspace were for household essential items. This consumer behavior is important because oftentimes these items will have higher single item price points than some lower category items such as toys and entertainment.

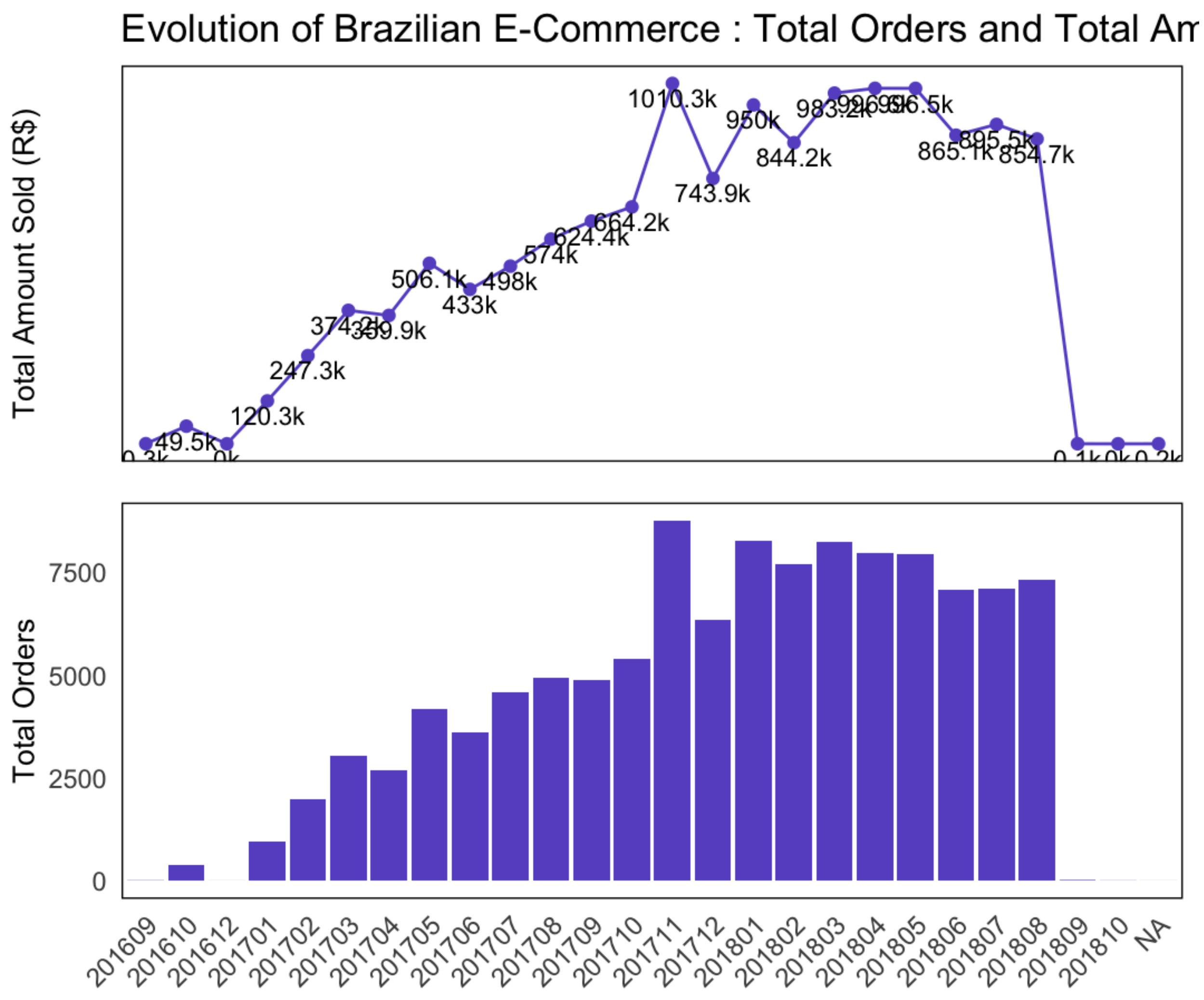


Figure 5: *The Total Amount Sold and the Correlating Number of Orders*

Another key piece of our model was understanding the current growth in the sales and number of total orders being placed with the e-commerce network. In order to do this, the team created a Growth of Sales Analysis tool. The team first simplified the date column into the various months of observed data, placing it alongside the correlating sales values. We then visualized this data by putting it into a line chart above a bar graph depicting the total number of orders being placed. What we observed is a very steady growth between 2016 and 2018, following a typical retail sales pattern of spikes just prior to the holiday season. What this information has told us is that there is a constant market growth, so it is essential for us to understand what the potential of this growth pattern will be.

One of the most important aspects of sales is returning consumer bases. In order to analyze the impact of consistent consumers, we needed to create a target variable, labeled “Churn”, what this value would tell us is that if a consumer’s “Recency” was greater than the mean value they would be assigned a “1”. This value would indicate if a consumer was at risk of disengagement, otherwise they would be labeled with a 0 for no risk. This is where the 'customer_unique_id' variable was extremely important, as it allowed us to accurately see behavior per individual and if they were a repeat shopper so we could target resources into improving sales on those at risk of churn. We would use a lambda function to assign these variables for the target column.

The distribution of the associated variables is revealed by the skewness values. A distribution that is perfectly symmetrical has a skewness value of 0, whereas skewness that is positive or negative denotes left or right skewness, respectively. The extreme right skewness of variables such as 'price,' 'freight_value,' 'product_weight_g,' 'product_length_cm,' 'product_height_cm,' 'product_width_cm,' 'payment_value,' and 'Monetary' suggests a heavy right tail with a concentration of lower values and a few outlier high values. This skewness raises the possibility of extreme values or outliers in these features, which could affect the performance of the model. To guarantee model robustness and precise interpretation of these variables, additional research and possible outlier treatment are advised.

To prepare to create the different model types we would use for our analysis we did multiple steps. Initially we decided on the various columns that we would keep for our model building. We conducted a final check for any missing values within the columns and rows of the data set. We trained our model using 10% of the original data, which was further separated into independent and dependent variables. We created dummy variables for the categorical variables within our data. This allowed us to train our data and split the test data using scaled features.

The first model we created was a logistic regression model. We built this model using the training data to predict the sales growth and accuracy of our model. We created a confusion matrix and classification report to further analyze the efficiency of this model. This is visualized in an ROC curve that will be analyzed further in the next section. We would then repeat this process for both the decision tree and random forest models.

Results and Performance

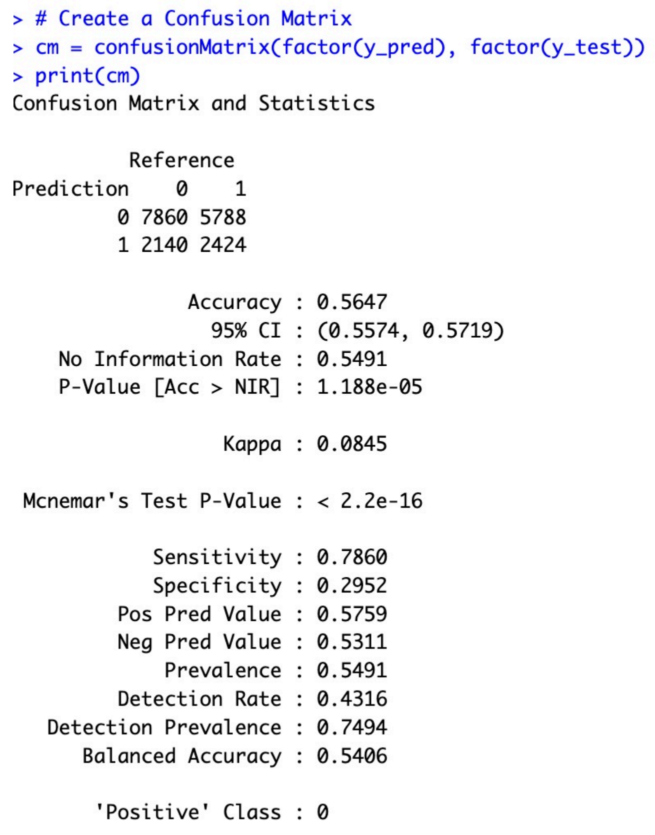
We built three models, Logistic regression, Decision trees and Random forest**.**

Logistic Regression:

Train-Test Split:

The dataset is split into training and test sets using the createDataPartition function. createDataPartition divides the data into 80% for training (X_train, y_train) and 20% for testing (X_test, y_test) the logistic regression model.

We trained the logistic regression model using the training data and evaluated it. Then we computed the confusion matrix, ROC curve and AUC.



```
> # Create a Confusion Matrix
> cm = confusionMatrix(factor(y_pred), factor(y_test))
> print(cm)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7860 5788
         1 2140 2424

               Accuracy : 0.5647
                 95% CI : (0.5574, 0.5719)
    No Information Rate : 0.5491
    P-Value [Acc > NIR] : 1.188e-05

                  Kappa : 0.0845

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7860
            Specificity : 0.2952
         Pos Pred Value : 0.5759
         Neg Pred Value : 0.5311
             Prevalence : 0.5491
         Detection Rate : 0.4316
   Detection Prevalence : 0.7494
      Balanced Accuracy : 0.5406

       'Positive' Class : 0
```

Figure 6: *Confusion matrix for logistic regression*

The logistic regression model exhibits 56.47% accuracy, outperforming the 54.91% No Information Rate. With a Kappa value of 0.0845, there's slight agreement between predicted and actual classes. Sensitivity is strong at 78.60%, but specificity is limited at 29.52%, indicating a challenge in classifying instances of class 1. Positive Predictive Value (PPV) and Negative Predictive Value (NPV) stand at 57.59% and 53.11%, respectively. While surpassing baseline performance, the model shows room for enhancement, especially in specificity and overall accuracy.

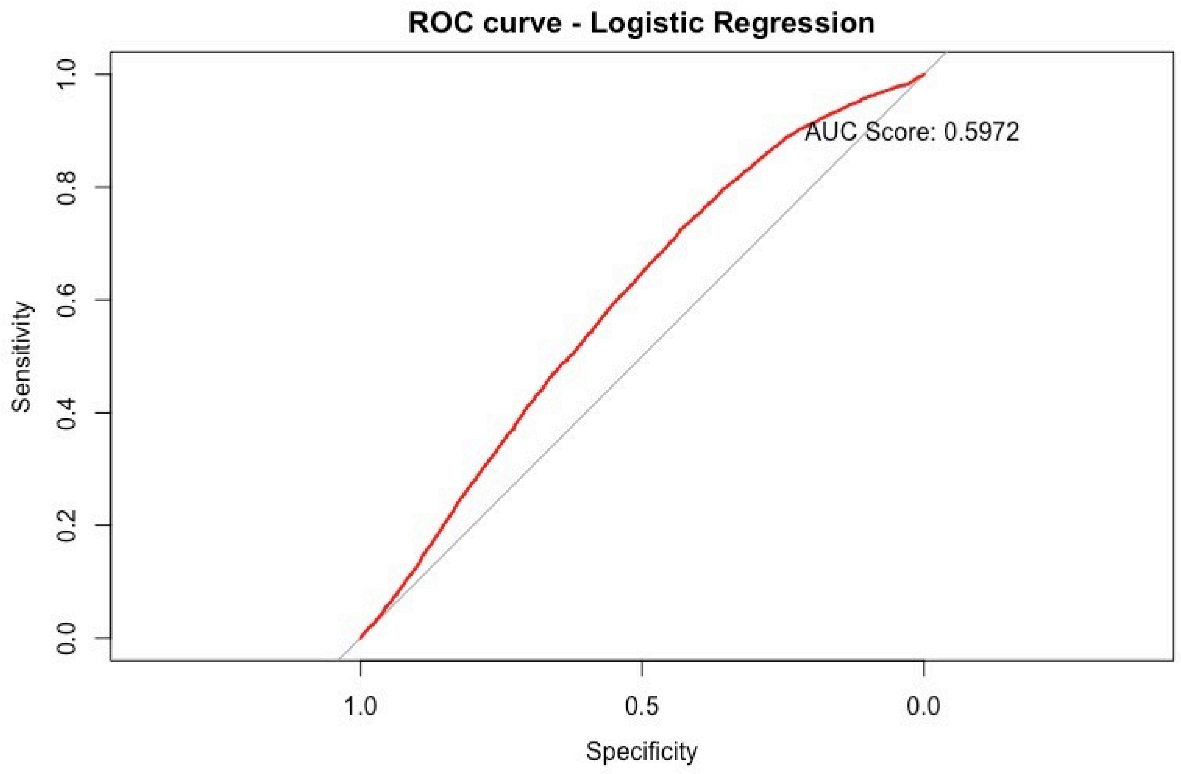


Figure 7: *ROC Curve for Logistic Regression*

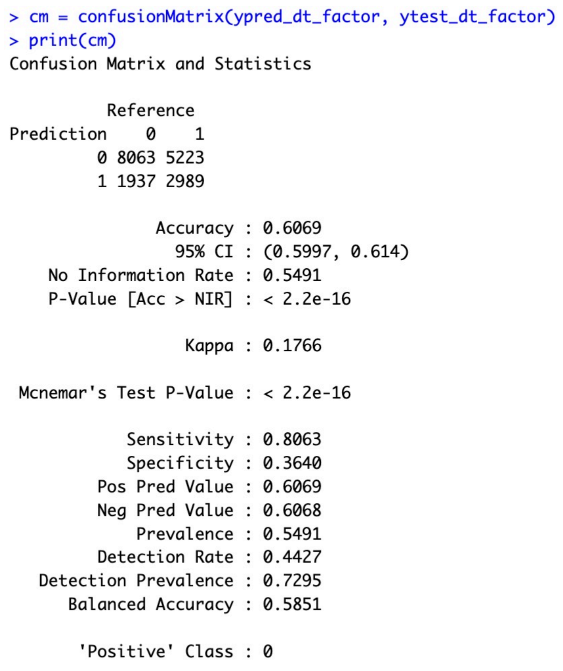
The output provided for the Logistic Regression model is an AUC Score of only 0.5972. This is an extremely low score when trying to predict the accuracy of how pricing and sales will be impacted within the e-commerce network. This model was our initial attempt, after analyzing the output, we realized that this type of model would not hold up in creating an accurate model. This then leads our team to move on and create the next model in our code, which is the Decision tree model. We believe that the poor performance of this model is because we are aiming to predict and analyze a quantitative variable, and the Logistic Regression model is best used for determining the impact of categorical variables. This model did provide a baseline that captured linear relationships between variables.

Decision Trees Model:

Train-Test Split:

Similar to logistic regression, the dataset is split into training and test sets using the same method (createDataPartition). The division remains consistent, allocating 80% for training (xtrain_dt, ytrain_dt) and 20% for testing (xtest_dt, ytest_dt) the decision tree model.

We trained the Decision tree model using the training data and evaluated it. Then we computed the confusion matrix, ROC curve and AUC.



```
> cm = confusionMatrix(ypred_dt_factor, ytest_dt_factor)
> print(cm)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 8063 5223
         1 1937 2989

               Accuracy : 0.6069
                 95% CI : (0.5997, 0.614)
    No Information Rate : 0.5491
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.1766

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8063
            Specificity : 0.3640
         Pos Pred Value : 0.6069
         Neg Pred Value : 0.6068
             Prevalence : 0.5491
         Detection Rate : 0.4427
   Detection Prevalence : 0.7295
      Balanced Accuracy : 0.5851

       'Positive' Class : 0
```

Figure 8: *Confusion matrix for decision trees.*

The decision tree model demonstrates a 60.69% accuracy, surpassing the 54.91% No Information Rate. With a Kappa value of 0.1766, the model shows a fair level of agreement between predicted and actual classes. Sensitivity at 80.63% indicates effective identification of class 0 instances, while specificity at 36.40% suggests room for improvement in classifying class 1.

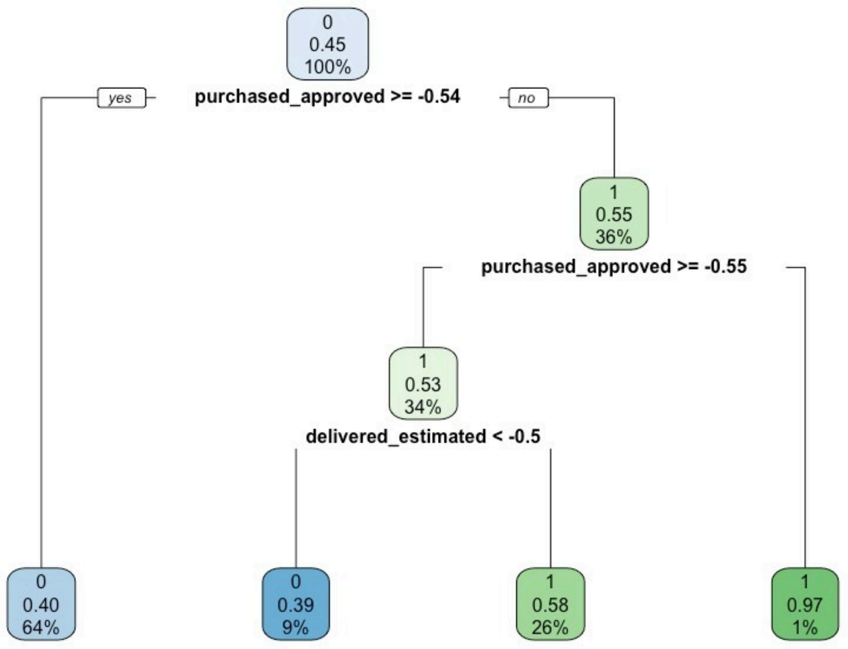

0
0.45
100%
yes
purchased_approved >= -0.54
no
1
0.55
36%
purchased_approved >= -0.55
1
0.53
34%
delivered_estimated < -0.5
0
0.40
64%
0
0.39
9%
1
0.58
26%
1
0.97
1%

Figure 9: *Decision tree model*

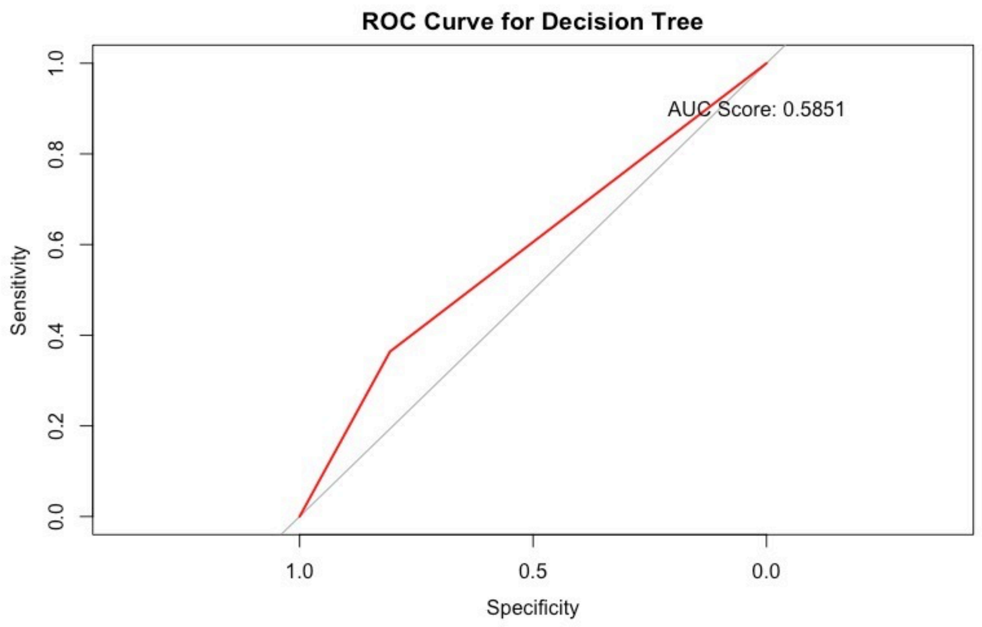


Figure 10: *ROC Curve for Decision Tree Model*

The Decision Tree model was similar to the Logistic Regression model in that it did not provide strong enough accuracy or an AUC score to draw any conclusions from the model. What the Decision Tree model was able to provide that was not seen within the Logistic Regression were clear decision paths and interpretability.

Random Forest Model

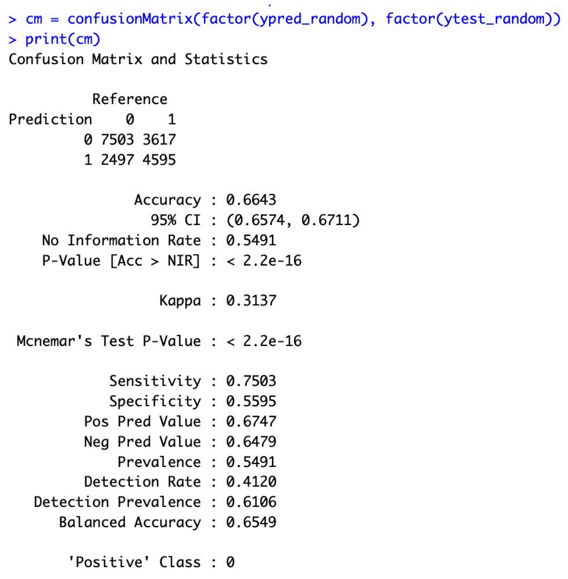
Train-Test Split:

Again, the dataset is split into training and test sets using createDataPartition.

Similar to the previous models, it divides the data into 80% for training (xtrain_random, ytrain_random) and 20% for testing (xtest_random, ytest_random) the random forest model.

We trained the Decision tree model using the training data and evaluated it. Then we computed the confusion matrix, ROC curve and AUC.

Confusion matrix:



```
> cm = confusionMatrix(factor(ypred_random), factor(ytest_random))
> print(cm)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7503 3617
         1 2497 4595

               Accuracy : 0.6643
                 95% CI : (0.6574, 0.6711)
    No Information Rate : 0.5491
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3137

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.7503
            Specificity : 0.5595
         Pos Pred Value : 0.6747
         Neg Pred Value : 0.6479
             Prevalence : 0.5491
         Detection Rate : 0.4120
   Detection Prevalence : 0.6106
      Balanced Accuracy : 0.6549

       'Positive' Class : 0
```

Figure 11: *The confusion matrix for random forest*

The random forest model exhibits a commendable performance with an accuracy of 66.43%, significantly surpassing the 54.91% No Information Rate. The Kappa value of 0.3137 indicates substantial agreement between predicted and actual classes. Sensitivity is strong at 75.03%, reflecting accurate identification of class 0 instances, while specificity at 55.95% indicates effective classifying of class 1. Positive Predictive Value (PPV) and Negative Predictive Value (NPV) stand at 67.47% and 64.79%, respectively. The model outperforms the baseline, demonstrating robust predictive capabilities across both classes.

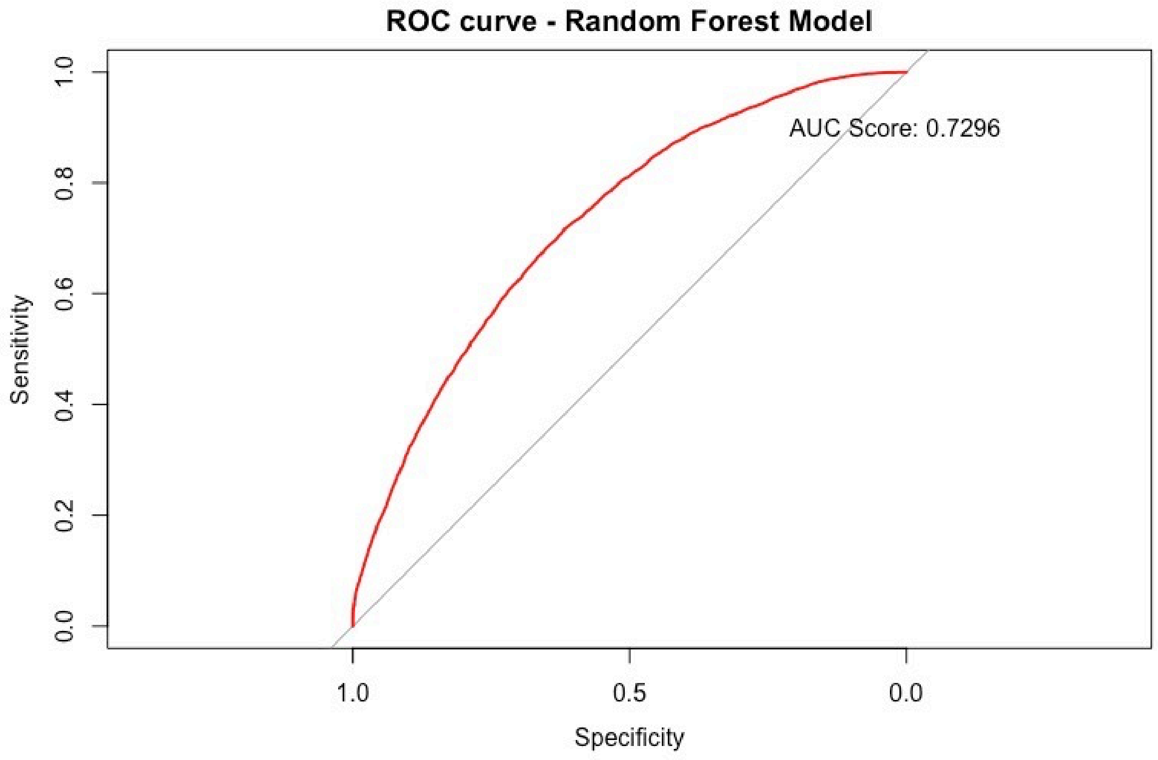


Figure 12: *The ROC Curve for our Random Forest Model*

The random forest model provided us with the best results from the various models we tested through our project. This AUC score of 0.7296 provides enough accuracy to confirm our tests data compared to what has been observed. This score was significantly better than the previous two models and reassured our thoughts that this is what should be used in the future to project how sales will be impacted throughout the market. This model further indicates that it can identify intricate patterns in the data, which will allow for those who analyze the output to make based decisions with analytically backing.

Classification report (Random forest)

1. Precision = 67.47% (high - Number of False positives are low)
2. Recall = 75.03%(high - Number of False negatives are low)
3. F1 Score = 67.47% (high - imbalance in recall and precision is low)
4. Accuracy = 66.43%

The Random forest is the best model for predicting Churn classification.

Discussion and Recommendations

The first major recommendation moving forward would be separating the analysis per major market. If we split our analysis based on states, or even cities depending on sales volume, we will be able to more accurately predict future sales and where resources may best be utilized for marketing or infrastructure development. For example, although São Paulo has the highest sales volume (4365.4k sold), the average payment per order (114) is the lowest. This points to possible areas for targeted marketing campaigns or price optimization.

Other group insights suggest focusing on enhancing features contributing to predictive power. Any future models should target consumer behavior, not just the items sold or what is delivered. There are plenty of market opportunities to be found within what items were browsed but not purchased, but focusing on finishing sales the network could boost overall profitability and return consumers. Another example of this observation would be the items that were canceled or lost in shipping, this could expose any weak points in the fulfillment network that could be improved to lock in any future sales that may have been lost without trimming the data.

Another recommendation is to improve ongoing model monitoring for adaptability. Aligning with the initial problem statement, the results reinforce the importance of leveraging robust predictive models for improved decision-making, guiding future actions towards a more refined, feature-rich approach to better address the specified problem.

Appendix

R Script

```
# Install the necessary packages
install.packages("dplyr")
install.packages("skimr")
install.packages("randomForest")

# Load necessary libraries in R
library(dplyr)
library(ggplot2)
library(skimr)
library(lubridate)
library (grid)
library(tidyverse)
library(moments)
library(corrplot)
library(caret)
library(readr)
library(tidyr)
library(modelr)
library(stats)
library(reshape2)
library(irr)
library(pROC)
library(randomForest)
library(rpart)
```

```
#Data Preprocessing


#Reading the data

df_customer = read.csv('olist_customers_dataset.csv')

df_geolocation = read.csv('olist_geolocation_dataset.csv')  # Replace with the correct file name

head(df_geolocation)

df_items = read.csv('olist_order_items_dataset.csv')

df_payments = read.csv('olist_order_payments_dataset.csv')

df_reviews = read.csv('olist_order_reviews_dataset.csv')

df_orders = read.csv('olist_orders_dataset.csv')

df_product = read.csv('olist_products_dataset.csv')

df_seller = read.csv('olist_sellers_dataset.csv')

df_category = read.csv('product_category_name_translation.csv')


#Joining the datasets

head(df_geolocation)

head(df_orders)

inner_join_result = inner_join(df_orders, df_payments, by = "order_id")

print(inner_join_result)

df_orderpay = inner_join_result # Joined Orders and Payments datset

nrow(df_orderpay)

ncol(df_orderpay)

orders_com = left_join(df_orderpay, df_items, by = 'order_id')

head(orders_com)

df_final1 = left_join(orders_com, df_product, by = 'product_id')

head(df_category)


df_final2 = left_join(df_final1 , df_category, by = 'product_category_name' )
```

```
df_final3 = left_join(df_final2, df_customer , by = 'customer_id')
df_final4 = left_join(df_final3 , df_seller, by = 'seller_id')
colnames(df_final4)

#Checking the missing values
missing_values = sapply(df_final4, function(x) sum(is.na(x)))
print(missing_values)



df_final = left_join(df_final4, df_geolocation, by = c('customer_zip_code_prefix'='geolocation_zip_code_prefix'))
head(df_final)

#Cleaning Dataset
colnames(df_final)

#Checking the missing values
missing_values = sapply(df_final, function(x) sum(is.na(x)))
print(missing_values)

rows_with_missing_values = df_final[rowSums(is.na(df_final)) > 0, ]
print(rows_with_missing_values)

#Dropping the rows with missing values
df_cleaned = na.omit(df_final)
print(summary(df_cleaned))
```

```
#Dealing with the missing value-
missing_values = colSums(is.na(df_cleaned))
print(missing_values)
#data frame "df_cleaned" has no missing values


# Convert relevant columns to Date type
# Convert relevant date columns to POSIXct format
df_cleaned$order_approved_at = as.POSIXct(df_cleaned$order_approved_at, format="%Y-%m-%d %H:%M:%S", tz="UTC")
df_cleaned$order_delivered_customer_date = as.POSIXct(df_cleaned$order_delivered_customer_date, format="%Y-%m-%d %H:%M:%S", tz="UTC")
df_cleaned$order_estimated_delivery_date = as.POSIXct(df_cleaned$order_estimated_delivery_date, format="%Y-%m-%d %H:%M:%S", tz="UTC")
df_cleaned$order_purchase_timestamp = as.POSIXct(df_cleaned$order_purchase_timestamp, format="%Y-%m-%d %H:%M:%S", tz="UTC")


# Create new columns
df_cleaned = df_cleaned %>%
  mutate(purchased_approved = as.numeric(difftime(order_approved_at, order_purchase_timestamp, units = "secs")),
     approved_carrier = as.numeric(as.Date(order_delivered_carrier_date) - as.Date(order_approved_at)),
         carrier_delivered = as.numeric(as.Date(order_delivered_customer_date) - as.Date(order_delivered_carrier_date)),
         delivered_estimated = as.numeric(as.Date(order_estimated_delivery_date) - as.Date(order_delivered_customer_date)),
         purchased_delivered = as.numeric(as.Date(order_delivered_customer_date) - as.Date(order_purchase_timestamp)))
# New columns are created using the available datetime columns for easy analysis of the available data.
```

# Purchased_approved represents the seconds taken for an order to get approved after the customer purchases it.

# approved_carrier represents the days taken for the order to go to the delivery carrier after it being approved.

# carrier_delivered represents the days taken for the order to be delivered to the customer from the date it reaches the delivery carrier.

# delivered_estimated represents the date difference between the estimated delivery date and the actual delivery date.

# purchased_delivered represents the days taken for the order to be delivered to the customer from the date the customer made the purchase.

```
missing_values = colSums(is.na(df_cleaned))
print(missing_values)

# Filter rows where 'approved_carrier' is less than 0
falsifiedData = which(df_cleaned$approved_carrier < 0)
df_cleaned = df_cleaned[-falsifiedData, ]

# Filter rows where 'carrier_delivered' is less than 0
falsifiedData = which(df_cleaned$carrier_delivered < 0)
df_cleaned = df_cleaned[-falsifiedData, ]

# Find indices of rows with 'order_status' as 'canceled' after dropping NA values
canceledIndex = which(!is.na(df_cleaned$order_status) & df_cleaned$order_status == 'canceled')
df_cleaned = df_cleaned[-canceledIndex, ]
```

```
# Define the date-time format
date_format = "%Y-%m-%d %H:%M:%S"

# Convert columns to POSIXct type specifying the format
date_columns = c("order_approved_at", "order_purchase_timestamp", "order_delivered_carrier_date", "order_delivered_customer_date", "order_estimated_delivery_date")
df_cleaned[date_columns] = lapply(df_cleaned[date_columns], function(x) as.POSIXct(x, format = date_format))

# Calculate the time differences and create new columns
df_cleaned$purchased_approved = as.numeric(difftime(df_cleaned$order_approved_at, df_cleaned$order_purchase_timestamp, units = "secs"))
df_cleaned$approved_carrier = as.numeric(difftime(df_cleaned$order_delivered_carrier_date, df_cleaned$order_approved_at, units = "days"))
df_cleaned$carrier_delivered = as.numeric(difftime(df_cleaned$order_delivered_customer_date, df_cleaned$order_delivered_carrier_date, units = "days"))
df_cleaned$delivered_estimated = as.numeric(difftime(df_cleaned$order_estimated_delivery_date, df_cleaned$order_delivered_customer_date, units = "days"))
df_cleaned$purchased_delivered = as.numeric(difftime(df_cleaned$order_delivered_customer_date, df_cleaned$order_purchase_timestamp, units = "days"))

head(df_cleaned)

# Get summary statistics for non-numeric columns
non_numeric_summary = summary(df_cleaned[, sapply(df_cleaned, function(x) !is.numeric(x))])

# Display the summary statistics for non-numeric columns
print(non_numeric_summary)
```

```
#summary(df_cleaned)

#rfm
df_merged = df_cleaned
colnames(df_merged)

# Dropping multiple columns
columns_to_drop = c('order_status', 'order_item_id', 'order_approved_at', 'order_delivered_carrier_date',
          'order_delivered_customer_date', 'order_estimated_delivery_date', 'approved_carrier',
          'carrier_delivered', 'seller_id', 'shipping_limit_date', 'product_category_name',
          'product_name_lenght', 'product_description_lenght', 'product_photos_qty',
          'payment_sequential', 'seller_zip_code_prefix')

merged = df_merged%>% select(-one_of(columns_to_drop))

# Printing the first few rows of the modified dataframe
head(merged)

# Assuming 'merged' is your dataframe in R and 'purchased_approved', 'delivered_estimated', 'purchased_delivered' are columns with boolean values
#summary(merged)

final <- merged %>%
  group_by(customer_unique_id) %>%
  summarise(customer_zip_code_prefix = max(customer_zip_code_prefix),
       customer_city = first(customer_city),
       customer_state = first(customer_state),
       order_id = n_distinct(order_id),

       purchased_approved = mean(purchased_approved),

       delivered_estimated = min(delivered_estimated),

       purchased_delivered = mean(purchased_delivered),

       product_id = n_distinct(product_id),

       price = sum(price),

       freight_value = sum(freight_value),

       product_weight_g = sum(product_weight_g),

       product_length_cm = sum(product_length_cm),

       product_height_cm = sum(product_height_cm),

       product_width_cm = sum(product_width_cm),

       geolocation_lat = mean(geolocation_lat),

       geolocation_lng = mean(geolocation_lng),

       payment_type = last(payment_type),

       payment_installments = max(payment_installments),

       payment_value = sum(payment_value),

  ) %>%

  ungroup()


# Printing the first few rows of the aggregated dataframe

head(final)



# Recency

# Assuming 'merged' is your dataframe in R

recency = merged %>%

  group_by(customer_unique_id) %>%

  summarise(LastPurchaseDate = max(order_purchase_timestamp)) %>%
         mutate(Recency = as.integer(as.Date(max(merged$order_purchase_timestamp)) -
as.Date(LastPurchaseDate)))


# Display the head of the recency dataframe
head(recency)


# Print the last recent date in the available dataset
recent_date = as.Date(max(merged$order_purchase_timestamp))
print(paste("The last recent date in the available dataset is:", recent_date))


# Convert 0 in recency to 1
recency$Recency = ifelse(recency$Recency == 0, 1, recency$Recency)


# Frequency
frequency = merged %>%
  group_by(customer_unique_id) %>%
  summarise(Frequency = n_distinct(order_id))


# Monetary
monetary = merged %>%
  group_by(customer_unique_id) %>%
  summarise(Monetary = sum(payment_value))



# Install and load necessary packages
if (!require("tidyverse")) {
  install.packages("tidyverse")
}
```

```
if (!require("lubridate")) {
  install.packages("lubridate")
}

# Calculate Recency
recent_date = max(merged$order_purchase_timestamp %>% as.Date)
recency = merged %>%
  group_by(customer_unique_id) %>%
  summarize(LastPurchaseDate = max(order_purchase_timestamp)) %>%
  mutate(Recency = as.integer(difftime(recent_date, LastPurchaseDate, units = "days")))

# Calculate Frequency
frequency = merged %>%
  group_by(customer_unique_id) %>%
  summarize(Frequency = n_distinct(order_id))

# Calculate Monetary
monetary = merged %>%
  group_by(customer_unique_id) %>%
  summarize(Monetary = sum(payment_value))

# Merge Recency, Frequency, and Monetary
rfm = recency %>%
  left_join(frequency, by = "customer_unique_id") %>%
  left_join(monetary, by = "customer_unique_id")

# Remove 0s from Recency
recent_date = as.Date(recent_date)
```

```
rfm$Recency = ifelse(rfm$Recency == 0, 1, rfm$Recency)


# Create a target variable for Churn
rfm$Churn = ifelse(rfm$Recency > mean(rfm$Recency, na.rm = TRUE), 1, 0)
# The 'Churn' column is derived based on the 'Recency' feature.
# It assesses customer behavior by determining if their last interaction or purchase occurred more recently than the average.


# Using a lambda function with apply to categorize customers:
# If the customer's 'Recency' is greater than the mean 'Recency', they are marked as 'Churn' (1),
# indicating potential disengagement or a higher likelihood to stop doing business.
# Otherwise, they are labeled as 'Not Churn' (0),
# suggesting ongoing engagement or a lower risk of discontinuation.


# Check the structure of 'final' and 'rfm' dataframes
str(final)
str(rfm)


# Make sure 'customer_unique_id' exists in both dataframes and has consistent values
# If not, adjust the column names or perform necessary data cleaning steps to align them


# Assuming the column names are consistent, and both dataframes have 'customer_unique_id'
# Merge the dataframes based on 'customer_unique_id' column
#Merging the target variable with our final dataframe


final = merge(final, rfm[c("customer_unique_id", "Recency", "Monetary", "Frequency", "Churn")], by = "customer_unique_id", all.x = TRUE)
```

```
# Check the merged 'final' dataframe
str(final)
head(final)

# Check the head of the merged RFM dataframe
#head(final)

#summary(final)

final$Churn = as.factor(final$Churn)


#Checking the skewness

# Select numeric columns
numeric_cols = sapply(final, is.numeric)
numeric_data = final[, numeric_cols]

# Loop through numeric columns
for (col in names(numeric_data)) {
# Calculate skewness and standard deviation
  skew = skewness(numeric_data[[col]])
  std_dev = sd(numeric_data[[col]])

# Print skewness and standard deviation
  cat(paste("Skewness of", col, ":", skew, "\n"))
  cat(paste("Standard deviation of", col, ":", std_dev, "\n"))
}
```

```
# Dropping the columns which has close to 0 standard deviation
columns_to_drop = c('customer_zip_code_prefix', 'no_of_orders', 'no_of_products', 'Frequency')

final = final[, !names(final) %in% columns_to_drop]



final_c = final
missing_percentages = colSums(is.na(final_c)) / nrow(final_c) * 100
colnames(final_c)



# Multivariate Analysis
percentage_distribution = prop.table(table(final_c$Churn)) * 100
print(percentage_distribution)

#dealing with the outliers
replace_outliers = function(x) {
  z_scores = scale(x)
  outliers = abs(z_scores) > 3  # Define threshold for outliers (e.g., 3)
  x[outliers] = median(x, na.rm = TRUE)  # Replace outliers with median (or mean)
  return(x)
}

# Apply outlier handling to numeric columns in the dataframe
numeric_columns = final_c %>% select_if(is.numeric)

final_c[ , names(numeric_columns)] = lapply(numeric_columns, function(x) {
  replace_outliers(x)
})


#correlation matrix


head(final_c)


numeric_cols = sapply(final_c, is.numeric)
numeric_data = final_c[, numeric_cols]
corr_matrix = cor(numeric_data)
skewness_values = sapply(numeric_data, skewness)
std_deviation = sapply(numeric_columns, sd, na.rm = TRUE)


# Print standard deviation of each column
print(std_deviation)


print(skewness_values)
corrplot(corr_matrix, method = 'circle')



# Define the function for state encoding
state_encoding = function(state) {
  if (state %in% c('RS', 'SC', 'PR')) {
    return('southern')
  } else if (state %in% c('SP', 'RJ', 'MG', 'ES')) {
    return('southeastern')
  } else if (state %in% c('MT', 'MS', 'GO', 'DF')) {
    return('centralwestern')
  } else if (state %in% c('MA', 'PI', 'CE', 'RN', 'PB', 'PE', 'AL', 'SE', 'BA')) {
    return('northeastern')
  } else {
    return('northern')
  }
}

# Apply state encoding to 'customer_state' column in 'final_c'
final_c$customer_state = sapply(final_c$customer_state, state_encoding)

# Creating 'features' dataframe by copying 'final_c'
features = final_c

# Dropping unnecessary columns
features = subset(features, select = -c(customer_unique_id, customer_city, payment_value))

# Dropping 'Recency' column from 'features'
features$Recency = NULL

# Creating 'independent' dataframe without 'Churn' column
independent = subset(features, select = -Churn)

# Separating numeric and categorical columns
df_numeric = independent[, sapply(independent, is.numeric)]
df_categorical = independent[, !sapply(independent, is.numeric)]

# Converting 'Churn' to integer type for 'df_target'
df_target = as.integer(features$Churn)
```

```
# Encoding categorical variables

encoded_data = model.matrix(~ . - 1, data = df_categorical)


# Combining numeric and encoded categorical variables

X = cbind(df_numeric, encoded_data)


# Dropping 'Recency' column from 'X'

X$Recency = NULL


# Displaying the resulting 'X' dataframe

head(X)

colnames(X)


# Set seed for reproducibility

set.seed(500)

head(final_c)

####################### Model building ##########################################################


# Load and preprocess data

#final_c = read_csv('final_c.csv')


# Selecting columns

columns_to_keep = c("customer_state", "purchased_approved", "delivered_estimated",

                "purchased_delivered", "price", "freight_value", "product_weight_g",

                "product_length_cm", "product_height_cm", "product_width_cm",

                "geolocation_lat", "geolocation_lng", "payment_type",
                "payment_installments", "Monetary", "Churn")


final_c = final_c %>% select(all_of(columns_to_keep)) %>% drop_na()


# Check missing values

missing_values = sapply(final_c, function(x) sum(is.na(x)))

missing_columns = missing_values[missing_values > 0]

print(missing_columns)


# Using only 10% of the data

set.seed(500)

final_c_sampled = final_c %>% sample_frac(0.9)


# Separate the data into independent and dependent variables

final_independent = final_c %>% select(-Churn)

final_df_target = final_c$Churn


# Convert 'Churn' column to integer type

#final_c_sampled$Churn <- as.integer(final_c_sampled$Churn)

#head(final_c_sampled)


# Create dummy variables for categorical data

final_encoded_data = dummyVars("~ .", data = final_independent)

final_independent_encoded = predict(final_encoded_data, newdata = final_independent)

head(final_independent_encoded)

# Scale the features

preProcValues = preProcess(final_independent_encoded, method = c("center", "scale"))

final_X_scaled = predict(preProcValues, final_independent_encoded)
```

```
# Train-test split
set.seed(500)
trainIndex = createDataPartition(final_df_target, p = .8,
                                 list = FALSE,
                                 times = 1)
X_train = final_X_scaled[trainIndex, ]
X_test  = final_X_scaled[-trainIndex, ]
y_train = final_df_target[trainIndex]
y_test  = final_df_target[-trainIndex]

# Print the shapes of the resulting sets
cat('xtrain:', dim(X_train), '\n')
cat('ytrain:', length(y_train), '\n')
cat('xtest:', dim(X_test), '\n')
cat('ytest:', length(y_test), '\n')

summary(y_train)

####################################################

# Build the logistic regression model
logit_model = glm(y_train ~ ., data = data.frame(X_train, y_train), family = "binomial")
summary(logit_model)



logit_model_updated = glm(
  y_train ~ . - customer_statecentralwestern - customer_statenortheastern -
    customer_statenorthern - customer_statesoutheastern - customer_statesouthern -
    price - geolocation_lat - geolocation_lng - payment_typeboleto -
    product_weight_g,
  family = "binomial",
  data = data.frame(X_train, y_train)
)
summary(logit_model_updated)

#aic remained the same
# AIC
AIC(logit_model)

# Predictions
y_pred_prob_train = predict(logit_model, newdata = data.frame(X_train), type = "response")
y_pred_train = ifelse(y_pred_prob_train < 0.5, 0, 1)

y_pred_prob = predict(logit_model, newdata = data.frame(X_test), type = "response")
y_pred = ifelse(y_pred_prob < 0.5, 0, 1)
head(y_pred)

# Create a Confusion Matrix
cm = confusionMatrix(factor(y_pred), factor(y_test))
print(cm)

# ROC Curve and AUC
roc_obj = roc(y_test, y_pred_prob)
plot(roc_obj, main = "ROC curve - Logistic Regression",col='red')
```

```
text(x = 0.02, y = 0.9, labels = paste("AUC Score:", round(auc(roc_obj), 4)))


auc_score = auc(roc_obj)

print(paste('AUC Score:', auc_score))


#########################################


# Building the Decision Tree Model




# Scale the features

preProcValues = preProcess(final_independent_encoded, method = c("center", "scale"))

final_X_scaled_df = predict(preProcValues, final_independent_encoded)




# Assuming final_X_scaled_df and final_df_target are available

set.seed(500)

trainIndex = createDataPartition(final_df_target, p = .8, list = FALSE, times = 1)

xtrain_dt = final_X_scaled_df[trainIndex, , drop = FALSE]  # Ensure it remains a dataframe

xtest_dt = final_X_scaled_df[-trainIndex, , drop = FALSE]  # Ensure it remains a dataframe

ytrain_dt = final_df_target[trainIndex]

ytest_dt = final_df_target[-trainIndex]


# Print the shapes of the resulting sets

cat('xtrain:', dim(xtrain_dt), '\n')

cat('ytrain:', length(ytrain_dt), '\n')

cat('xtest:', dim(xtest_dt), '\n')

cat('ytest:', length(ytest_dt), '\n')
```

```
# Fit the Decision Tree

decisionTree = rpart(ytrain_dt ~ ., data = data.frame(xtrain_dt, ytrain_dt), method = "class")



rpart.plot(decisionTree)


# Predictions

ypred_dt = predict(decisionTree, newdata = data.frame(xtest_dt), type = "class")


# Convert to factors ensuring they have the same levels

levels = union(levels(factor(ypred_dt)), levels(factor(ytest_dt)))

ypred_dt_factor = factor(ypred_dt, levels = levels)

ytest_dt_factor = factor(ytest_dt, levels = levels)



# Convert factor/character to numeric

ypred_numeric = as.numeric(as.character(ypred_dt_factor))

ytest_numeric = as.numeric(as.character(ytest_dt_factor))


# Create Confusion Matrix

cm = confusionMatrix(ypred_dt_factor, ytest_dt_factor)

print(cm)


# Calculate ROC curve

roc_obj = roc(ytest_numeric, ypred_numeric)
```

```
# Plot ROC curve
plot(roc_obj, main = "ROC Curve for Decision Tree", col = "red")
text(x = 0.02, y = 0.9, labels = paste("AUC Score:", round(auc(roc_obj), 4)))

# Calculate AUC
auc_val = auc(roc_obj)
cat("AUC Score:", auc_val, "\n")

#########################################

#Building the Random forest Model



# Splitting the dataset
set.seed(500)
head(final_df_target)
splitIndex = createDataPartition(final_df_target, p = 0.8, list = FALSE)
xtrain_random = final_X_scaled_df[splitIndex, ]
xtest_random = final_X_scaled_df[-splitIndex, ]
ytrain_random = final_df_target[splitIndex]
ytest_random = final_df_target[-splitIndex]

# Print the shapes of the resulting sets
cat('xtrain:', dim(xtrain_random), '\n')
cat('ytrain:', length(ytrain_random), '\n')
cat('xtest:', dim(xtest_random), '\n')
cat('ytest:', length(ytest_random), '\n')
```

```
# Assuming xtrain_random, ytrain_random, xtest_random, and ytest_random are available

# Train the Random Forest model
rand_model = randomForest(xtrain_random, ytrain_random)

# Predict probabilities and classes
ypred_proba_random = predict(rand_model, xtest_random, type = "prob")
ypred_random = ifelse(ypred_proba_random[,2] > 0.5, 1, 0) # Assuming class '1' is the positive class

# Predict on the training set for evaluation
ypred_proba_random_train = predict(rand_model, xtrain_random, type = "prob")
ypred_random_train = ifelse(ypred_proba_random_train[,2] > 0.5, 1, 0)

# Confusion Matrix and heatmap
cm = confusionMatrix(factor(ypred_random), factor(ytest_random))
print(cm)

# ROC Curve
roc_obj = roc(ytest_random, ypred_proba_random[,2])
plot(roc_obj, main = "ROC curve - Random Forest Model",col='red')

text(x = 0.02, y = 0.9, labels = paste("AUC Score:", round(auc(roc_obj), 4)))


#######################Exploratory Data Analysis ############################################
```

```
#Distribution of Order Status
# create a count table of order_status
status_count <- df_orders %>%
  count(order_status) %>%
  mutate(pct = n/sum(n))
# reorder the order_status by the count of each status in descending order
status_count$order_status <- factor(status_count$order_status, levels =
status_count$order_status[order(status_count$n, decreasing = TRUE)])
# plot the count table
ggplot(status_count, aes(order_status, n)) +
  geom_bar(stat="identity", fill = "slateblue") +
  labs(title= "Distribution of Order Status", x="Order Status", y="Count") +
  theme_minimal() +
  geom_text(aes(label = paste0(sprintf("%.1f", pct*100), "%")), size = 3.5, vjust = -0.2) +
  theme(plot.title = element_text(size = 13)) +
  theme(panel.background = element_rect(fill='white'),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())

#combine df_orders and df_customer
df_orders <- df_orders %>%
  left_join(df_customer, by = "customer_id")

#Define a function that can convert latitude and longitude into decimal degrees
angle2dec <- function(angle) {
  angle <- as.character(angle)
  x <- do.call(rbind, strsplit(angle, split=' '))
  x <- apply(x, 1L, function(y) {
    y <- as.numeric(y)
    y[1] + y[2]/60 + y[3]/3600
  })
  return(x)
}

lat_north <- "5 16 27.8"
lat_south <- "33 45 04.21"
long_west <- "73 58 58.19"
long_east <- "34 47 35.33"

decimal_degrees_north <- angle2dec(lat_north)
decimal_degrees_south <- -angle2dec(lat_south)#the negative sign and multiplication by -1 to get south longitude
decimal_degrees_west <- -angle2dec(long_west)#the negative sign and multiplication by -1 to get West longitude
decimal_degrees_east <- -angle2dec(long_east)#the negative sign and multiplication by -1 to get West longitude

#Brazils territory
geo <- df_geolocation %>% filter(geolocation_lat <= decimal_degrees_north &
                                  geolocation_lat >= decimal_degrees_south &
                                  geolocation_lng <= decimal_degrees_east &
                                  geolocation_lng >= decimal_degrees_west)
geo <- geo %>%
  group_by(geolocation_zip_code_prefix) %>%
  summarise_all(min) %>%
  ungroup()
```

```
library(forcats)
# Top cities with more customers orders in Brazil
df_orders %>%
  group_by(geolocation_city) %>%
  count() %>%
  arrange(desc(n)) %>%
  head(10) %>%
  ggplot(aes(fct_reorder(geolocation_city,n), n, fill = fct_reorder(geolocation_city,n))) +
  geom_bar(stat="identity", show.legend = F) +
  coord_flip() +
  labs(title = "Top 10 Brazilian Cities with More Orders",
      x = "City", y = "Total Orders")+
  theme_minimal() +
  theme(plot.title = element_text(size = 13),
       axis.title=element_text(size=8)) +
  theme(panel.background = element_rect(fill='white', colour = "white"),
       panel.grid.major = element_blank(),
       panel.grid.minor = element_blank()) +
  geom_text(aes(label = n), size = 3, hjust = 1.2, color = "white")+
  scale_fill_brewer(palette = "Spectral")


#Total orders by state
df_orders %>%
  group_by(geolocation_state) %>%
  count() %>%
  arrange(desc(n)) %>%
  head(10) %>%
  ggplot(aes(fct_reorder(geolocation_state,n), n, fill = fct_reorder(geolocation_state,n))) +
  geom_bar(stat="identity", show.legend = F) +
  coord_flip() +
  labs(title = "Top 10 Brazilian State with More Orders",
       x = "State", y = "Total Orders")+
  theme_minimal() +
  theme(plot.title = element_text(size = 13),
        axis.title=element_text(size=8)) +
  theme(panel.background = element_rect(fill='white', colour = "white"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  geom_text(aes(label = n), size = 3, hjust = 1.2, color = "white")+
  scale_fill_brewer(palette = "Spectral")


###Growth of Sales Analyis
# Growth of sales
df_order_items2 <- df_items %>%
  group_by(order_id) %>%
  mutate(price_2 = mean(price),
         freight_value2 = mean(freight_value)) %>%
  distinct(order_id, .keep_all = T) %>%
  dplyr:: select(-price, -freight_value)

df_orders_items <- df_orders  %>%
  left_join(df_order_items2, by = "order_id")
```

```
#combine olist_orders and olist_customer
df_orders <- df_orders %>%
  left_join(df_customer, by = "customer_id")

# Changing the data type for datetime format
timestamp_cols <- c('order_purchase_timestamp', 'order_approved_at', 'order_delivered_carrier_date',
                  'order_estimated_delivery_date')
df_orders[,timestamp_cols] <- lapply(df_orders[,timestamp_cols], as.POSIXct, format = "%Y-%m-%d
%H:%M:%S")

# Extracting attributes for purchase date - Year and Month
df_orders <- df_orders %>%
  mutate(order_purchase_year = year(order_purchase_timestamp),
        order_purchase_month = month(order_purchase_timestamp, label = T),
        order_purchase_year_month = format(df_orders$order_purchase_timestamp, "%Y%m"),
        order_purchase_year_month_day = format(df_orders$order_purchase_timestamp, "%Y%m%d"))

df_orders_items <- df_orders  %>%
  left_join(df_items, by = "order_id")

df_orders<- df_orders  %>%
  left_join(df_orders_items, by = "order_id")

sales_plot <- df_orders %>%
  group_by(order_purchase_year_month.y) %>%
  summarise(price_sum = sum(price, na.rm = T),
          count = n()) %>%
  ggplot(aes(order_purchase_year_month.y, price_sum, group = 1)) +
  geom_line(color = "slateblue") +
  geom_point(color = "slateblue") +
  labs(title = "Evolution of Brazilian E-Commerce : Total Orders and Total Amount Sold(R$)",
      x = "Year-Month", y = "Total Amount Sold (R$)") +
  theme_minimal() +
  theme(plot.title = element_text(size = 13),
       axis.text.x = element_blank(),
       axis.title.x =element_blank(),
       axis.title.y = element_text(size = 10),
       axis.text.y=element_blank()) +
  theme(panel.background = element_rect(fill='white'),
       panel.grid.major = element_blank(),
       panel.grid.minor = element_blank())+
   geom_text(aes(label = paste0(round(price_sum/1000, 1), "k")), size = 3, vjust = 1.3, color = "black",
position =  position_dodge(width = 1))


amount_plot <- df_orders %>%
  group_by(order_purchase_year_month.y) %>%
  count() %>%
  ggplot(aes(order_purchase_year_month.y, n)) +
  geom_bar(stat = "identity", fill = "slateblue") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10),
       axis.text.x = element_text(angle = 45, hjust = 1),
       axis.title=element_text(size=8)) +
  theme(panel.background = element_rect(fill='white'),
       panel.grid.major = element_blank(),
       panel.grid.minor = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_text(size = 10)) +
  labs(y = "Total Orders")


library (grid)
grid.newpage()
grid.draw(rbind(ggplotGrob(sales_plot), ggplotGrob(amount_plot), size = "last"))



### Product Analysis


df_orders_items <- df_orders_items %>%
  left_join(df_product, by = "product_id")


df_orders_items <- df_orders_items %>%
  left_join(df_reviews, by = "order_id")


category_counts <- df_orders_items %>%
  group_by(product_category_name) %>%
  summarize(count = n())


# Sort the categories by count in descending order
category_counts <- category_counts %>%
  arrange(desc(count))


# Create a bar plot
ggplot(head(category_counts,10), aes(x = reorder(product_category_name, -count), y = count)) +
  geom_bar(stat = "identity", fill = "#bcbddc") +
labs(x = "Product Category", y = "Frequency") +
coord_flip() +  # Horizontal bars for better readability
theme_minimal() +
ggtitle("Distribution of Product Categories")
```