PREDICTING LATE DELIVERIES TO INCREASE SALES

Abstract

This project addresses the increasingly strategic problem of late deliveries in e-commerce, using Olist a leading Brazilian online retail platform as a case study. Although Olist's current late delivery rate of 9.1% is in line with global averages, it poses a rising threat to long-term revenue growth, especially as industry leaders are striving aggressively towards aggressively lower their benchmarks through machine learning innovations. This study aims to develop a predictive model capable of identifying at-risk orders before dispatch, enabling timely, data-driven interventions to reduce delays. Using a publicly available dataset of approximately 100,000 historical orders, three supervised learning models Logistic Regression, Random Forest, and XGBoost were implemented. The project involved comprehensive data preprocessing, exploratory analysis, and engineered features that captured key logistical dynamics such as delivery distance, holiday timing, and seller dispersion. Model training was optimized through grid search and validated using Stratified K-Fold cross-validation. Among the models tested, XGBoost demonstrated the most effective performance with a recall of 39%, ROC AUC of 0.79, and accuracy of 90.1%. Feature importance analysis revealed that seller count, holiday weeks, geographic distance, and customer location were key predictors of delivery delay. These results justify the deployment of XGBoost at the order-placement stage to assess delivery risk in real time. The proposed solution is scalable, cost-efficient, and strategically aligned with Olist's goal of enhancing fulfilment reliability and maintaining competitiveness in a market increasingly shaped by predictive logistics.

Introduction

In the increasingly competitive and margin-sensitive world of e-commerce, timely delivery is no longer a value-added service but a baseline customer expectation. Retailers are under immense pressure to meet shrinking delivery windows, which directly affect consumer satisfaction, brand perception, and ultimately, sales performance. Prior studies show that faster delivery promises are positively correlated with customer acquisition, retention and sales (Deshpande and Pendem, 2023; Fisher et al., 2019). However, as logistics networks grow more complex and demand patterns fluctuate, late deliveries remain a persistent operational challenge. For platforms like Olist a Brazilian e-commerce intermediary that connects sellers to buyers through external logistics partners delays affect not only consumer experience but also platform trust and long-term sales growth. Olist currently faces a 9.1% late delivery rate, a figure that becomes increasingly problematic as global competitors drive this benchmark downward through predictive technology adoption.

Machine learning models have proven highly effective in addressing late delivery challenges across the logistics industry. Their growing adoption from last-mile route optimization to delay forecasting demonstrates their transformative impact on operational efficiency and service reliability.. Industry leaders such as Amazon, Uber for route optimisation (Business Insider), and Aramex (in partnership with AWS) have successfully deployed ML models to reduce late deliveries and enhance logistical precision (Küp et al., 2024; eMarketer, 2018). Academic research also validates the potential of these methods. Salari et al. (2020) leveraged ensemble techniques like random forest and boosting to predict delivery delays, while Rezki and Mansouri (2023) compared logistic regression, Gaussian Naïve Bayes, and random forest in forecasting supply chain disruptions.
Building on these insights, this project aims to develop a machine learning model tailored to Olist's operational context. The objective is to predict late deliveries at the point of order placement, enabling proactive risk mitigation and contributing to improved delivery reliability and sales performance.

The subsequent sections of this paper are structured as follows: **Section 2** outlines the business context and problem addressed in the study. **Section 3** describes the dataset, data preprocessing, exploratory data analysis, and the machine learning methods employed. **Section 4** presents the analysis and results, including model performance and evaluation. Finally, **Section 5** shows a discussion of key findings, implementation implications, and study limitations, concluding with directions for future research.

Business Problem

The global surge in e-commerce projected to exceed $8 trillion by 2027 and account for 22%+ of all retail trade (Cramer-Flood, 2024) has placed immense pressure on retail logistics. While this growth offers unprecedented commercial opportunities, it also exposes fulfilment systems to operational strain, especially around delivery. In 2024 alone, an estimated 8–10% of global online orders were delivered late, resulting in £31.5 billion in annual financial losses in the UK (Statista, 2024; IMRG, 2024). Delayed deliveries are not just a logistical issue they are a commercial risk. A McKinsey study found that 50% of consumers abandon purchases due to delivery uncertainty (Ecker et al., 2020), and delivery reliability is now a key driver of customer satisfaction, loyalty, and sales growth (Gielens, 2023; Griffis et al., 2012).

This project addresses a critical strategic and operational problem for Olist, a Brazilian e-commerce platform headquartered in São Paulo. Olist connects thousands of small and medium-sized sellers to major marketplaces such as Amazon and Magalu, facilitating listings, payments, logistics, and customer communication. When an order is placed through Olist Store, the platform notifies the relevant seller, who then dispatches the order using one of Olist's logistics partners. Olist provides customers with static delivery estimates based on broad averages (e.g., by region or product category), but these estimates do not account for dynamic or historical variables, such as courier performance, seller reliability, traffic or seasonal demand.

As a result, Olist faces a persistent 9.1% late delivery rate comparable to global averages but increasingly unsustainable given rising consumer expectations and this average decreasing now that companies like Amazon are further getting a reduction in late delivery upto 74% prior to their previous rate(AWS, 2021). Without a predictive framework, Olist lacks the visibility to identify at-risk orders early, flag unreliable sellers, allocate logistics resources intelligently, or issue dynamic delivery estimates. This leads to customer dissatisfaction, weakened platform trust, andmost importantly lost sales.

To address this, the project aims to develop a machine learning model that predicts delivery delay risk at the order level, enabling Olist to transition from a reactive logistics model to a proactive, data-informed fulfilment strategy. The model will be deployed at the order processing stage, where each incoming order will undergo delay risk assessment. Based on the prediction, Olist can communicate accurate delivery estimates to customers and take proactive steps to mitigate potential delays.

Exploratory Data Analysis (EDA) will complement this by identifying structural drivers of delay such as distance, product category, courier used, or specific seasons, providing further levers for strategic and operational improvement.

Similar challenges and ML have been observed globally. For instance, Amazon Web Services (AWS), in partnership with Inawisdom, developed a supervised ML model using Amazon SageMaker to predict delivery transit times more accurately. This improved delivery estimate accuracy by 74% and reduced call center volumes by 40%, offering a more seamless customer experience (AWS, 2021).

In the German LVHV manufacturing sector, manufacturers who themselves act as suppliers further downstream faced significant disruptions due to late deliveries from their own upstream suppliers. To address this, researchers applied machine learning models such as Support Vector Regression (SVR), Random Forest, and Adaptive Boosting (AB). Among these, AB achieved an accuracy of 92% in predicting the severity of delivery delays. This enabled timely interventions and enhanced supply chain reliability, highlighting how predictive analytics can drive operational resilience even in complex, multi-tiered industrial ecosystems (Felsberger et al., 2020).

Data, EDA & Methods

Python Libraries

Following Libraries were used:

Scikit-learn formed the core of the modeling process, with classifiers such as RandomForestClassifier and LogisticRegression used to predict late deliveries. Model performance was assessed using key evaluation metrics including accuracy, precision, recall, F1 score, and ROC-AUC, while tools like confusion matrices and ROC curves provided further diagnostic insight. To enhance model reliability, techniques such as GridSearchCV and StratifiedKFold were applied for hyperparameter tuning and cross-validation. Data preprocessing was handled using StandardScaler to ensure consistent variable scaling. In addition, XGBoost was incorporated for its powerful gradient boosting capabilities. The geodesic function from geopy was used to engineer distance-based features, capturing the geographic spread between sellers and customers. Warning were also added tp suppress non-critical warning messages. Throughout the project, pandas enabled efficient data manipulation, while matplotlib and seaborn were used to produce clear and informative visualizations.

Dataset Description

The dataset used in this project was sourced from the publicly available Olist Brazilian E-commerce dataset hosted on Kaggle (Olist, 2018). It contains ~ 100,000 orders placed between 2016 and 2018 across multiple marketplaces in Brazil, structured across 9 out of which 7CSV files were used. These files capture the e-commerce lifecycle from order placement to delivery, incorporating variables related to order status, payment, freight, product attributes, customer location, and post-purchase reviews.

The olist_customers_dataset contains customer IDs along with geographic details such as state, city, and ZIP code prefix, supporting location-based analysis. The olist_geolocation_dataset maps Brazilian ZIP codes to latitude and longitude coordinates, facilitating geospatial modelling. The product_category_name_translation table is included to convert product category names from Portuguese into English, aiding interpretability. The olist_sellers_dataset outlines seller-specific information including location and seller ID. The olist_products_dataset includes detailed product attributes such as category, weight, dimensions, and image availability. The olist_orders_dataset serves as the core table, documenting each order's status and multiple timestamps from purchase to final delivery. Finally, the olist_order_items_dataset has individual products and sellers, and includes shipping limit date, item prices, and freight charges.

Data Merging

The datasets mentioned above were merged into a dataframe and 2 datasets were excluded from the 9 total datasets we had. The olist_order_reviews_dataset was removed because it captured post-delivery customer sentiment, which is out of scope of our question. The olist_order_payments_dataset was deemed irrelevant to delivery speed, as payment method does not influence delivery timelines.

Features, Target Variable & Data types

```
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   order_id                       113425 non-null  object
 1   customer_id                    113425 non-null  object
 2   order_status                   113425 non-null  object
 3   order_purchase_timestamp       113425 non-null  object
 4   order_approved_at              113264 non-null  object
 5   order_delivered_carrier_date   111457 non-null  object
 6   order_delivered_customer_date  110196 non-null  object
 7   order_estimated_delivery_date  113425 non-null  object
 8   customer_unique_id             113425 non-null  object
 9   customer_zip_code_prefix       113425 non-null  int64
 10  customer_city                  113425 non-null  object
 11  customer_state                 113425 non-null  object
 12  order_item_id                  112650 non-null  float64
 13  product_id                     112650 non-null  object
 14  seller_id                      112650 non-null  object
 15  shipping_limit_date            112650 non-null  object
 16  price                          112650 non-null  float64
 17  freight_value                  112650 non-null  float64
 18  product_category_name          111047 non-null  object
 19  product_name_lenght            111047 non-null  float64
 20  product_description_lenght     111047 non-null  float64
 21  product_photos_qty             111047 non-null  float64
 22  product_weight_g               111047 non-null  float64
 23  product_length_cm              111047 non-null  float64
 24  product_height_cm              111047 non-null  float64
 25  product_width_cm               111047 non-null  float64
 26  seller_zip_code_prefix         112650 non-null  float64
 27  seller_city                    112650 non-null  object
 28  seller_state                   112650 non-null  object
 29  product_category_name_english  111023 non-null  object
dtypes: float64(11), int64(1), object(18)
memory usage: 26.0+ MB
```

Figure 1: Summary of Dataset Features, Data Types, and Missing Values (Pre-Cleaning)

The merged dataset comprises a combination of numerical and categorical features reflecting various aspects of customers, orders, products, and shipping information. Numerical variables are represented using the float64 and int64 data types, while categorical features are stored as object types. The target variable, *is_late*, was created as a binary indicator (0 for on-time, 1 for late) based on a comparison of actual and estimated delivery dates. This structured format supports efficient preprocessing and modeling in supervised learning tasks.

Descriptive Univariate Table

	customer_zip_code_prefix	order_item_id	price	freight_value	product_name_lenght	product_description_lenght
count	113425.000000	112650.000000	112650.000000	112650.000000	111047.000000	111047.000000
mean	35102.472965	1.197834	120.653739	19.990320	48.775978	787.867029
std	29864.919733	0.705124	183.633928	15.806405	10.025581	652.135608
min	1003.000000	1.000000	0.850000	0.000000	5.000000	4.000000
25%	11250.000000	1.000000	39.900000	13.080000	42.000000	348.000000
50%	24320.000000	1.000000	74.990000	16.260000	52.000000	603.000000
75%	59020.000000	1.000000	134.900000	21.150000	57.000000	987.000000
max	99990.000000	21.000000	6735.000000	409.680000	76.000000	3992.000000

	product_photos_qty	product_weight_g	product_length_cm	product_height_cm	product_width_cm	seller_zip_code_prefix
count	111047.000000	111047.000000	111047.000000	111047.000000	111047.000000	112650.000000
mean	2.209713	2099.966149	30.210951	16.626311	23.041928	24439.170431
std	1.721438	3754.771017	16.189288	13.467410	11.716610	27596.030909
min	1.000000	0.000000	7.000000	2.000000	6.000000	1001.000000
25%	1.000000	300.000000	18.000000	8.000000	15.000000	6429.000000
50%	1.000000	700.000000	25.000000	13.000000	20.000000	13568.000000
75%	3.000000	1800.000000	38.000000	20.000000	30.000000	27930.000000
max	20.000000	40425.000000	105.000000	105.000000	118.000000	99730.000000

Figure 2: Descriptive Statistics of Key Numerical Variables in the Dataset

The descriptive statistics table above provides a summary of key numerical variables in the dataset, including central tendency, dispersion, and distribution shape.

Missing Values

A significant amount of the features exhibits missing values, particularly in product-related attributes such as dimensions, weight, and description, each missing in 2,378 records. Additionally, product_category_name_english has the highest number of missing entries at 2,402. Details of other missing values in the respective features are shown in the table below.

```
order_id                         0
customer_id                      0
order_status                     0
order_purchase_timestamp         0
order_approved_at              161
order_delivered_carrier_date  1968
order_delivered_customer_date 3229
order_estimated_delivery_date    0
customer_unique_id               0
customer_zip_code_prefix         0
customer_city                    0
customer_state                   0
order_item_id                  775
product_id                     775
seller_id                      775
shipping_limit_date            775
price                          775
freight_value                  775
product_category_name         2378
product_name_lenght           2378
product_description_lenght    2378
product_photos_qty            2378
product_weight_g              2378
product_length_cm             2378
product_height_cm             2378
product_width_cm              2378
seller_zip_code_prefix         775
seller_city                    775
seller_state                   775
product_category_name_english 2402
dtype: int64
```

Figure 3: Missing Values in Dataset

Preprocessing Pipeline

The data preprocessing pipeline was essential in transforming the raw dataset into a clean, structured, and analytically meaningful format. Missing data were addressed to prevent biased or misleading results, while feature engineering focused on enriching the dataset by both refining existing features and generating new ones that captured key business and logistical dynamics to aid for deeper insights into factors influencing delivery performance.

Missing Data Handling

Missing values in the order_item_id column were initially imputed with zero to substitute null values and maintain data continuity and whole column itself was then converted to an integer data type to ensure compatibility with numerical modeling workflows. Following this, all remaining rows containing null values were removed to preserve analytical integrity and prevent potential bias during training. The dataset index was reset post-cleaning and also some duplicate rows in data were also dropped.

```
# Remove rows with any null values
df_cleaned = df.dropna()

# Reset index after dropping rows
df_cleaned.reset_index(drop=True, inplace=True)

# Check if all null values are removed
print(df_cleaned.isnull().sum())
```

```
order_id                          0
customer_id                       0
order_status                      0
order_purchase_timestamp          0
order_approved_at                 0
order_delivered_carrier_date      0
order_delivered_customer_date     0
order_estimated_delivery_date     0
customer_unique_id                0
customer_zip_code_prefix          0
customer_city                     0
customer_state                    0
order_item_id                     0
product_id                        0
seller_id                         0
shipping_limit_date               0
price                             0
freight_value                     0
product_category_name             0
product_name_lenght               0
product_description_lenght        0
product_photos_qty                0
product_weight_g                  0
product_length_cm                 0
product_height_cm                 0
product_width_cm                  0
seller_zip_code_prefix            0
seller_city                       0
seller_state                      0
product_category_name_english     0
dtype: int64
```

Figure 4: Dropped Missing Values

Feature Engineering

A comprehensive feature engineering process was undertaken to prepare the dataset for predictive modeling. All relevant temporal variables were first converted into the appropriate datetime64[ns] format, including order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, and order_estimated_delivery_date. This conversion enabled the extraction of higher-level time-based features discussed in detail below.

The order_item_id column column was converted from float to int32 to reflect its role as a discrete identifier. The target variable late_delivery was constructed by comparing the actual delivery date (order_delivered_customer_date) with the expected delivery date (order_estimated_delivery_date). Orders delivered after the estimated date were labeled as "Yes" for late delivery, while others were labeled "No". These outcomes were then encoded as binary values 1 for late deliveries and 0 for on-time enabling compatibility with supervised classification algorithms.

To gain deeper insights into the factors contributing to late deliveries, additional features were engineered beyond the original dataset, as the existing variables offered limited explanatory power. Temporal features were derived from order_purchase_timestamp, including order_purchase_month, order_purchase_weekday, order_purchase_weekday, year, and day of the month. These provided a seasonal and behavioral context to ordering patterns. A binary is_holiday_month flag was also introduced to identify whether an order was placed during a nationally significant holiday month in Brazil. Delivery-based features included the construction of the late_delivery flag and its conversion into a binary variable. In terms of logistics, product volume (product_volume_cm3) and total weight (total_weight_g) were calculated to evaluate the impact of item size and weight on delivery performance. Count-based flags were added to capture complexity within orders, including indicators for multiple items, multiple sellers, and a combined feature (multi_seller_multi_item_late) to flag high-complexity deliveries. Lastly features such as flag for repeat buyers was also made. These engineered variables enhanced the model’s ability to detect patterns associated with delivery delays and provided a more comprehensive foundation for predictive analysis. Engineered, existing and refined features are shown in table below:

#	Column	Non-Null Count	Dtype
0	order_id	97153 non-null	object
1	customer_id	97153 non-null	object
2	order_status	97153 non-null	object
3	order_purchase_timestamp	97153 non-null	datetime64[ns]
4	order_approved_at	97153 non-null	datetime64[ns]
5	order_delivered_carrier_date	97153 non-null	datetime64[ns]
6	order_delivered_customer_date	97153 non-null	datetime64[ns]
7	order_estimated_delivery_date	97153 non-null	datetime64[ns]
8	customer_unique_id	97153 non-null	object
9	customer_zip_code_prefix	97153 non-null	Int64
10	customer_city	97153 non-null	object
11	customer_state	97153 non-null	object
12	order_item_id	97153 non-null	int32
13	product_id	97153 non-null	object
14	seller_id	97153 non-null	object
15	shipping_limit_date	97153 non-null	datetime64[ns]
16	price	97153 non-null	float64
17	freight_value	97153 non-null	float64
18	product_category_name	97153 non-null	object
19	product_name_lenght	97153 non-null	float64
20	product_description_lenght	97153 non-null	float64
21	product_photos_qty	97153 non-null	float64
22	product_weight_g	97153 non-null	float64
23	product_length_cm	97153 non-null	float64
24	product_height_cm	97153 non-null	float64
25	product_width_cm	97153 non-null	float64
26	seller_zip_code_prefix	97153 non-null	Int64
27	seller_city	97153 non-null	object
28	seller_state	97153 non-null	object
29	product_category_name_english	97153 non-null	object
30	late_delivery	97153 non-null	bool
31	order_purchase_month	97153 non-null	int64
32	order_purchase_week	97153 non-null	int64
33	order_purchase_weekday	97153 non-null	int64
34	order_purchase_year	97153 non-null	int64
35	order_purchase_dayofmonth	97153 non-null	int64
36	is_holiday_month	97153 non-null	bool
37	items_per_order	97153 non-null	int64
38	has_multiple_items	97153 non-null	bool
39	total_product_dimensions	97153 non-null	float64
40	total_weight	97153 non-null	float64
41	order_month	97153 non-null	period[M]
42	order_item_count	97153 non-null	int64
43	multiple_items_flag	97153 non-null	int32
44	product_volume_cm3	97153 non-null	float64
45	total_weight_g	97153 non-null	float64
46	total_volume_cm3	97153 non-null	float64
47	is_holiday	97153 non-null	int32
48	delivered_mm_dd	97153 non-null	object
49	holiday_week	97153 non-null	bool
50	year	97153 non-null	int64
51	month	97153 non-null	int64
52	multiple_sellers	97153 non-null	bool
53	multiple_orders_by_customer	97153 non-null	bool
54	delivered_late	97153 non-null	bool
55	est_vs_carrier_diff_days	97153 non-null	int64
56	carrier_delivered_late	97153 non-null	bool
57	seller_count	97153 non-null	int64
58	item_count	97153 non-null	int64
59	multi_seller_multi_item_late	97153 non-null	bool
60	customer_lat	97153 non-null	float64
61	customer_lng	97153 non-null	float64
62	seller_lat	97153 non-null	float64
63	seller_lng	97153 non-null	float64
64	distance_km	97153 non-null	float64
65	delivery_status	97153 non-null	object
66	is_late	97153 non-null	int64

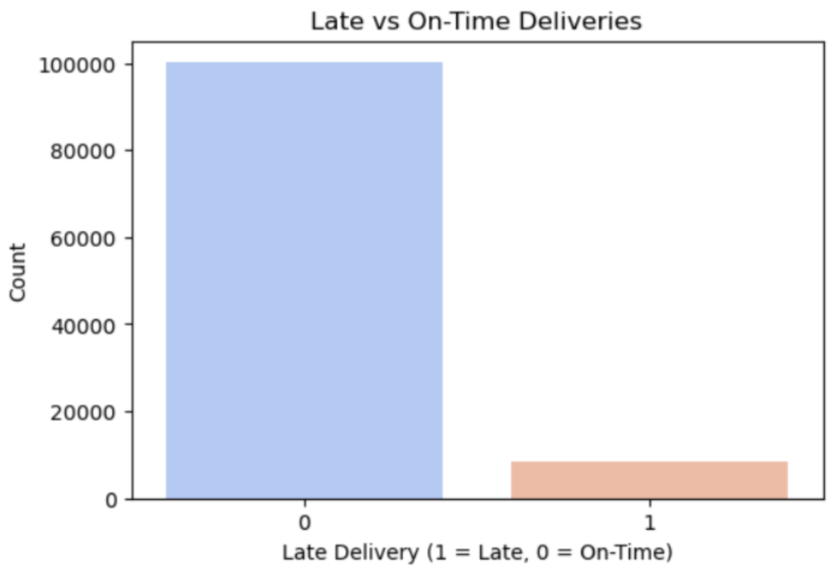
Figure 5: Feature Set After Feature Engineering

Variable Scaling

Variable scaling was applied to standardize numerical features, particularly for use with logistic regression, which is sensitive to differences in feature magnitude. Without scaling, features with larger numeric ranges could disproportionately influence model coefficients, leading to biased or suboptimal results.

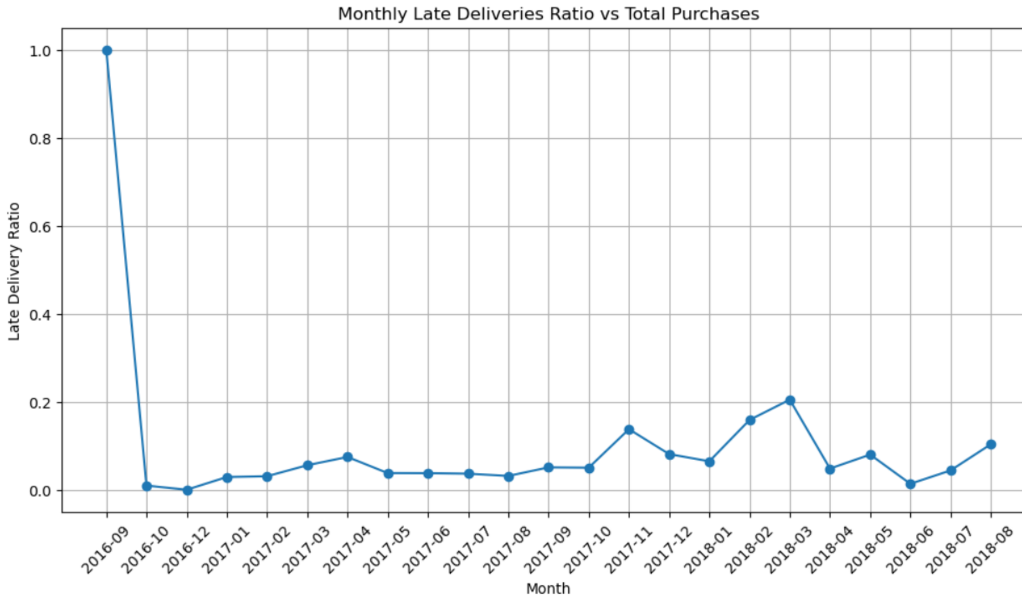
Graphical Data Exploration

Graphical data exploration was carried out to systematically identify and interpret the key variables contributing to late deliveries



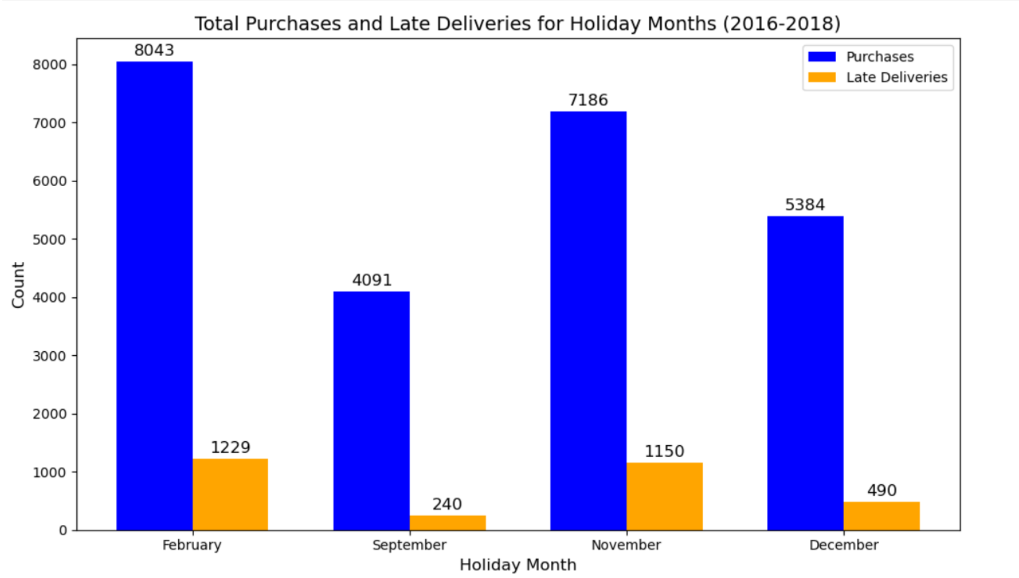
Graph 1: Distribution of Late vs On-Time Deliveries

This distribution chart shows ~91% of deliveries were completed on time (class 0), while 9% were delayed (class 1).



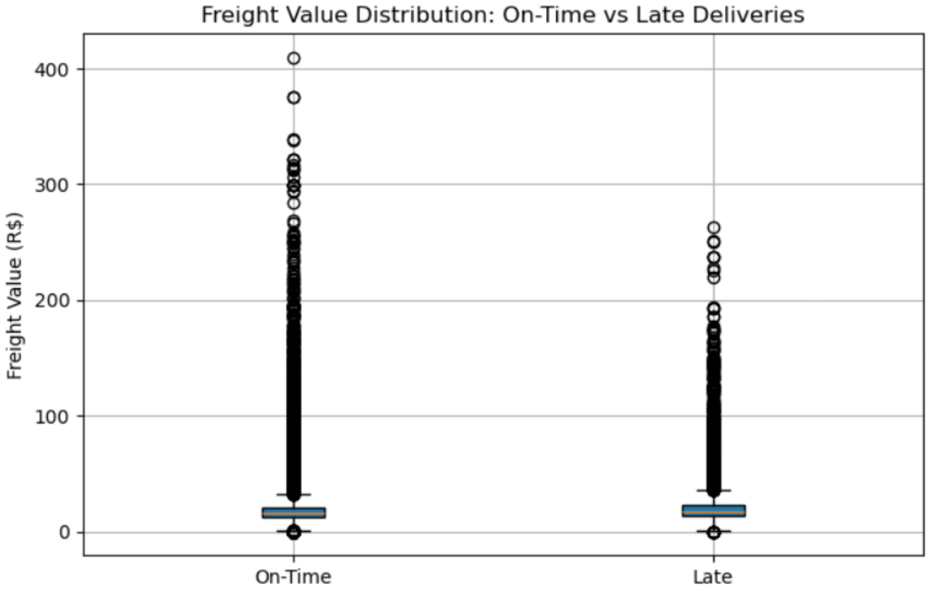
Graph 2: Monthly Late Deliveries Ratio vs Total Purchases

The graph shows that while total purchases generally increased over time, late delivery ratios remained low in most months, indicating that volume alone does not cause delays. However, consistent spikes in February and March of 2017 and 2018, along with November 2017, point to recurring periods of operational strain. These patterns suggest a threshold beyond which the late deliveries occur. Further analysis is needed to understand what was happening in these specific months leading to increased delivery delays.



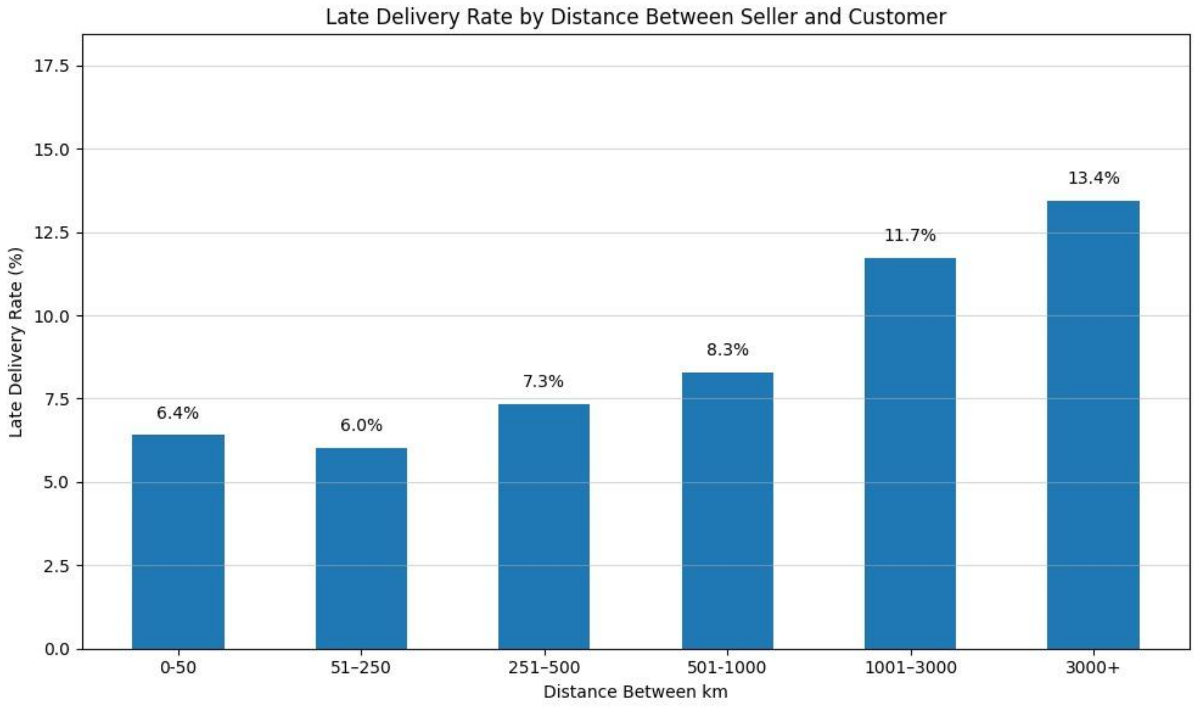
Graph 3: Total Purchases and Late Deliveries During Holiday Months (2016–2018)

To align with insights from Figure 2, we mapped key holiday months to examine whether spikes in late deliveries were driven by seasonal peaks. February and November (2017 & 2018) both high-demand periods show notable increases in purchase volumes and corresponding delivery delays, suggesting that peak-season pressure significantly causes late delivery. However, this is not the sole cause, as delays in other months without major holidays indicate the influence of additional operational or structural factors, which will be explored further.



Graph 4: Freight Value Distribution for On-Time vs Late Deliveries

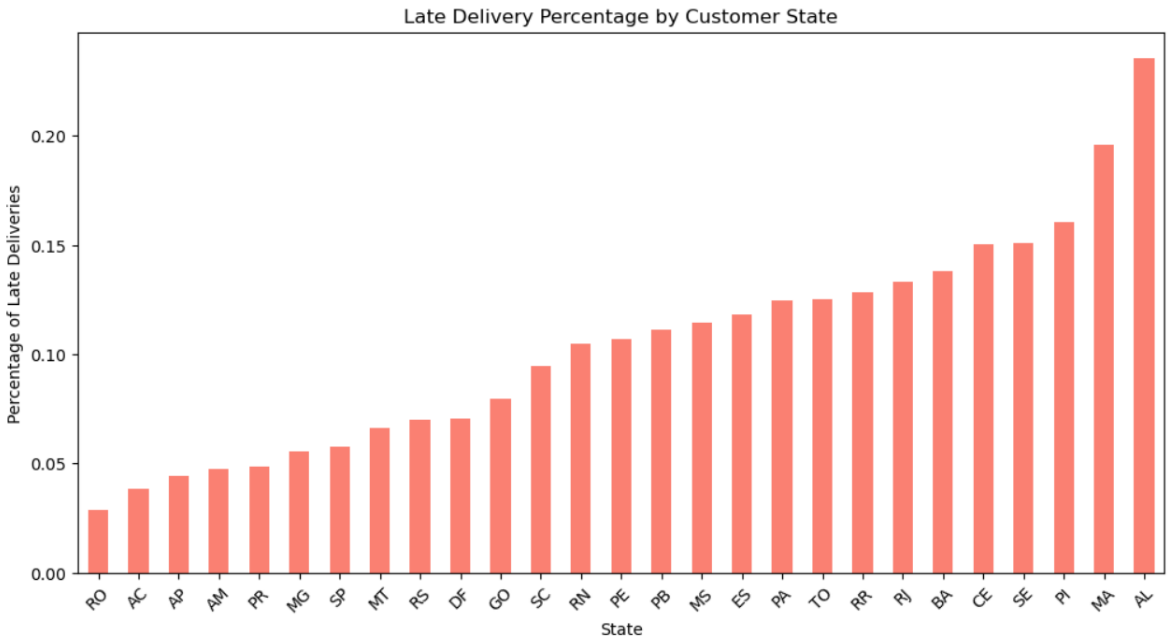
The boxplot reveals that late deliveries are generally associated with slightly lower freight values compared to on-time orders. While the distributions overlap, the lower median and compressed range for late deliveries suggest that reduced shipping investment may correspond with reduced delivery priority. This points to freight value as a potential influencing factor in delivery timeliness.



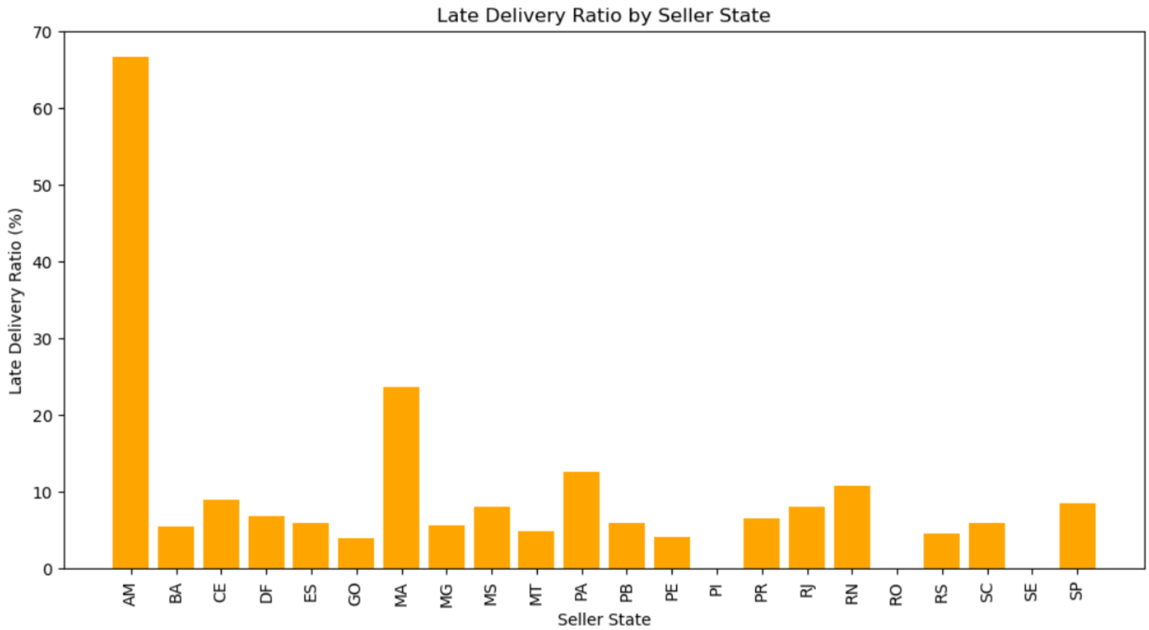
Graph 5: Late Delivery Rate by Distance Between Seller and Customer

The chart shows that while late delivery rates generally increase with distance, the 0–50 km range still has a surprisingly high delay rate (6.4%), likely due to urban congestion or operational bottlenecks. Beyond 500 km, delays rise steadily, peaking at 13.4% for distances over 3000 km. This suggests

Olist should address both short-distance inefficiencies and long-haul logistical challenges through better courier allocation and regional fulfillment strategies.

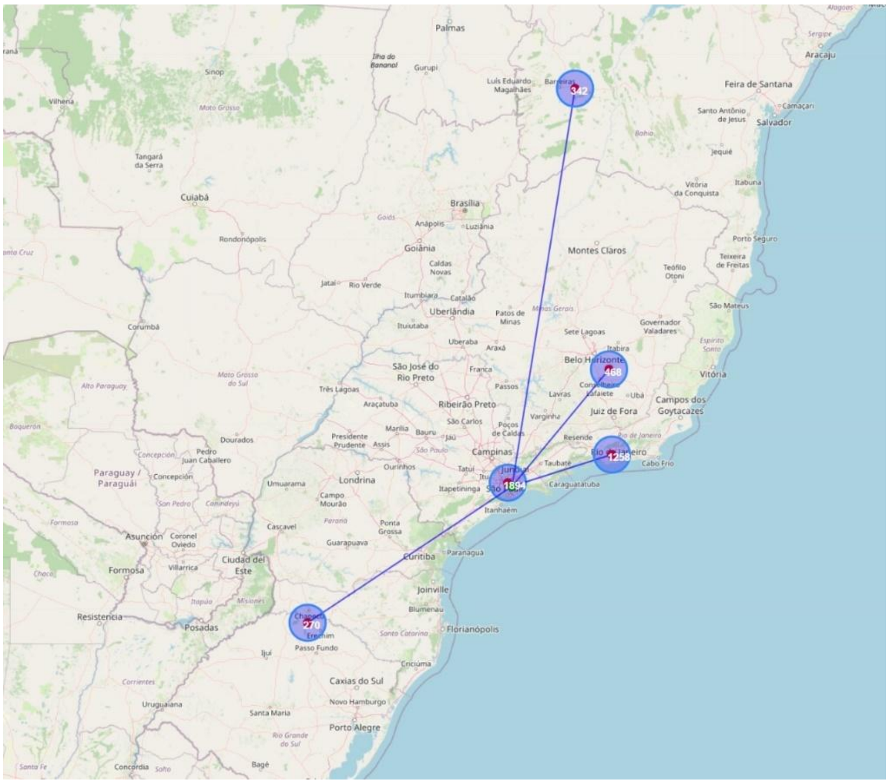


Graph 6: Late Delivery Percentage by Customer State

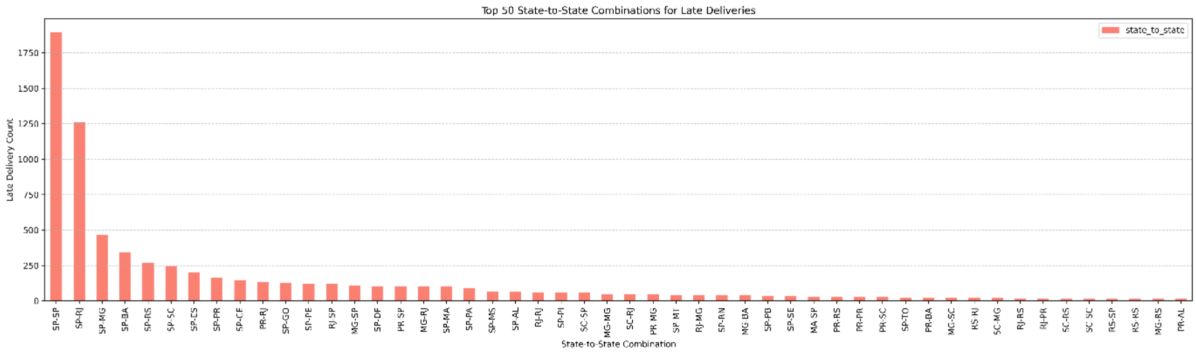


Graph 7: Late Delivery Percentage by Seller State

These both graph show states like MA (Maranhão) appear both as high in customer delays and as sellers with high delay ratios. This suggests local infrastructure or logistical limitations are affecting both outbound and inbound deliveries. AM (Amazonas) is a problematic seller location, even if customers in AM do not have the highest delays. This might be due to few sellers being concentrated there, but their inefficiency skews the data. On the customer side, high delay rates in AL and PI may reflect geographic remoteness, poor last-mile logistics, or longer travel distances from central seller hubs.

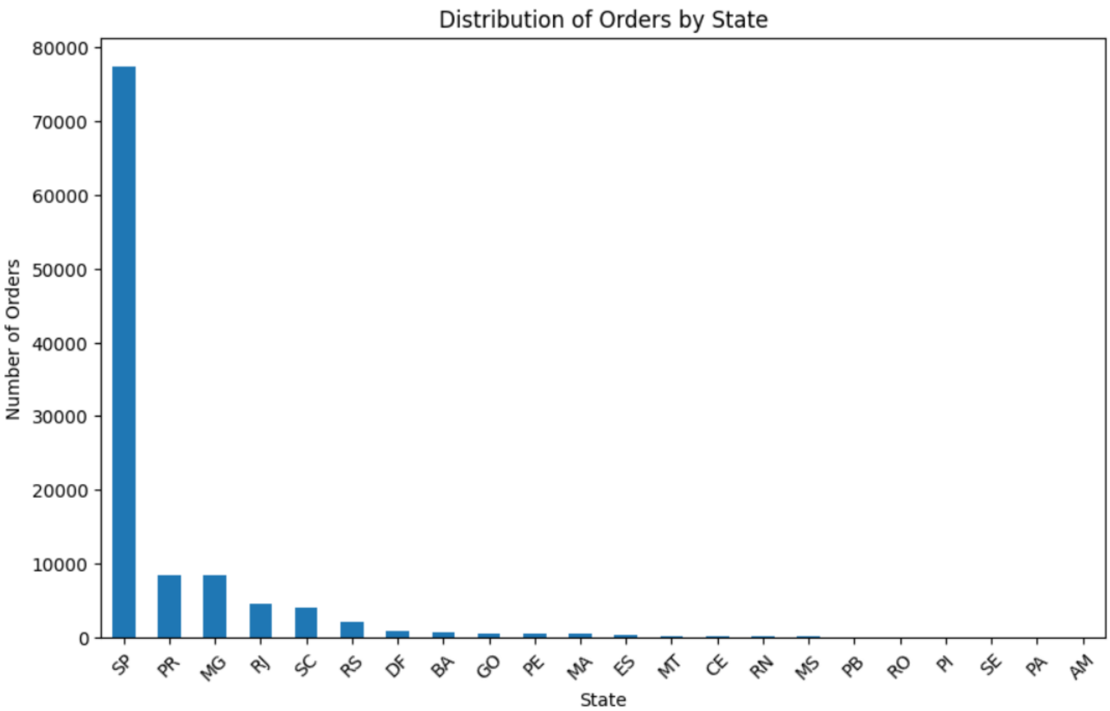


Graph 8: Geospatial and Route-Based Patterns Driving Late Deliveries in Brazil



Graph 9: Top 50 State-to-State Combinations for Late Deliveries

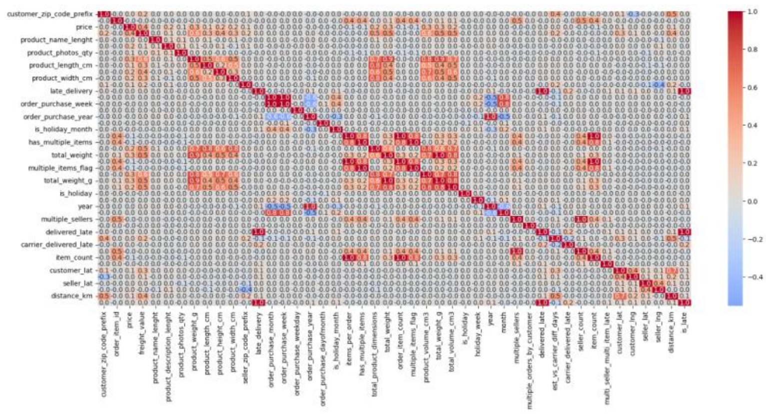
Despite the general trend that longer distances lead to higher delay rates, the top state-to-state delay chart reveals that short routes like SP→SP and SP→RJ still top the list for absolute late delivery counts. This reinforces that volume congestion and regional operational inefficiencies especially in high-density hubs like São Paulo are just as impactful as long-distance transit delays. Together, these insights show that Olist must tackle both geographic distance and high-traffic urban routes to improve delivery performance.



Graph 10: Number of Orders by State

The analysis reveals that late deliveries in Olist's network are driven by a combination of high seller concentration, geographic distance, and regional logistical disparities. São Paulo (SP), as the dominant seller hub, handles the majority of orders and appears in nearly all major delayed routes, including local ones like SP → SP, where delays stem from operational congestion. Long-distance routes to remote states such as Piauí, Maranhão, and Alagoas experience delays due to transit challenges and weak last-mile infrastructure. Although these regions receive fewer orders, they suffer from disproportionately high delay rates, indicating poor courier coverage and limited regional distribution support. Additionally, seller states like Amazonas exhibit very high late shipment ratios, pointing to dispatch inefficiencies and unreliable fulfillment practices. To address this, Olist should adopt a multi-pronged strategy: invest in regional warehouses to shorten delivery paths, onboard more local sellers to reduce reliance on distant hubs, and strengthen courier partnerships in underserved areas. Incorporating predictive delivery times and enforcing performance standards for sellers can further enhance reliability and reduce customer delay times.

The analysis identifies long delivery distances, few peak-season(holidays) order surges, and low freight value as key factors contributing to late deliveries. Additionally, regional sellers and congested delivery routes significantly influence delay patterns.



Several features exhibited multicollinearity, so we removed those with the highest correlation and retained the remaining variables for model training.

Following features for then selected for modelling.

```
'customer_zip_code_prefix', 'price', 'freight_value',
'product_name_lenght', 'product_description_lenght', 'product_photos_qty',
'seller_zip_code_prefix', 'order_item_count',
'total_weight_g', 'total_volume_cm3',
'is_holiday', 'holiday_week', 'order_purchase_week', 'order_purchase_weekday',  'order_purchase_dayofmonth',
'multiple_orders_by_customer',
'seller_count', 'customer_lat', 'customer_lng', 'seller_lat', 'seller_lng',
'distance_km'
```

Figure 6: Selected Features for Modelling

Test-Train-Validation Split

To determine the most effective train-test split, we experimented with multiple configurations specifically 70/30, 80/20 ratios across all three chosen models. Each split was performed using train_test_split, with stratification applied to preserve the class balance between late and on-time deliveries. After evaluating model performance across key metrics such as accuracy, precision, recall, F1 score, and ROC AUC, the 80/20 split was selected for final implementation. This configuration demonstrated the most favorable balance between generalization and predictive power, as detailed in the performance comparison section later in results. Cross validation technique is address in later section

Model Selection

This study applied three supervised machine learning models to address the classification problem of predicting late deliveries in an e-commerce environment: **Random Forest**, **XGBoost**, and **Logistic Regression**. Each model was selected based on its theoretical suitability, proven performance in logistics-related industry applications.

Random Forest Classifier

Random Forest (RF) was chosen for this classification task due to its robustness, speed, and high predictive accuracy. RF works by building multiple decision trees using random bootstrapped samples of the data and selecting a random subset of features at each split. For classification problem (late or no late deliver) in our case, it predicts the outcome through majority voting across all trees. This process reduces overfitting, improves generalization, and captures complex variable interactions.RF is particularly suited for our problem because it handles both structured and partially unstructured data, performs well with noisy datasets, and automatically ranks feature importance helping us identify key drivers of late deliveries. Evidently this has also been proven has one of the most effective ML model in the industry. In supply chain contexts, RF has consistently demonstrated high predictive accuracy and robustness across complex networks and industries, such as freight transportation and airline logistics (Rezki and Mansouri, 2024). The model's ability to avoid overfitting while handling multivariate and noisy data makes it particularly suited for delivery prediction problems, where delays may be influenced by numerous interacting variables (Breiman, 2001).

XGBoost Classifier

XGBoost (Extreme Gradient Boosting) Classifier was selected for its proven effectiveness in complex classification tasks, particularly in supply chain and logistics settings. It builds decision trees sequentially, with each iteration correcting the errors of the previous one, and incorporates regularization to reduce overfitting. The model is optimized for speed, handles missing values and high-dimensional data, and captures nonlinear relationships effectively.

Its fine-tuning capabilities and consistently high accuracy make it one of the most popular and reliable models in industry. Comparative studies have demonstrated its strong performance in classifying supplier outcomes and forecasting delivery delays (Rezki and Mansouri, 2024), aligning well with the objectives of this project.

Logistic Regression

Logistic Regression, although a linear model, remains a reliable and interpretable baseline for binary classification problems such as predicting whether a delivery will be late. It estimates the probability of a class outcome based on a linear combination of features, making it ideal for assessing the influence of individual predictors. Despite its simplicity, recent studies demonstrate that logistic regression can outperform more complex classifiers in supplier performance evaluation tasks (Yee et al., 2024). Moreover, its transparency is particularly valuable in operational settings where stakeholders require interpretable models to support decision-making (Jahin et al., 2025).

The methodological framework ensured data balance, feature relevance, and alignment with the predictive objective of late delivery. These 3 models were selected to evaluate their respective strengths in handling the classification task. This enabled a rigorous, model-driven comparison based on accuracy, and interpretability.

Analysis and Results

Hyperparameter Optimization and Cross-Validation

To ensure reliable model performance and prevent overfitting, Stratified K-Fold cross-validation with 6 splits was implemented using StratifiedKFold. This approach maintains the original class distribution in each fold, which is crucial given the class imbalance in the target variable (is_late).

The cross-validation was used in conjunction with GridSearchCV to perform hyperparameter tuning across all three models   Random Forest, XGBoost, and Logistic Regression.
Each model underwent exhaustive grid search across a predefined set of hyperparameters, optimizing for accuracy using the scoring='accuracy' parameter.

Random Forest, the grid included:

- n_estimators (number of trees): [100, 200]
- max_depth (tree depth): [None, 10, 20]
- min_samples_split: [2, 5]

XGBoost, additional steps taken to address class imbalance by calculating and setting scale_pos_weight (ratio of majority to minority class). The parameter grid included:

- n_estimators: [100, 200]
- max_depth: [3, 6, 10]
- learning_rate: [0.01, 0.1]
- subsample: [0.8, 1.0]
-

Logistic Regression, regularization and solver behavior were tuned:

- C (inverse regularization strength): [0.01, 0.1, 1.0, 10]
- penalty: ['l2']
- solver: ['lbfgs', 'liblinear']

Logistic Regression

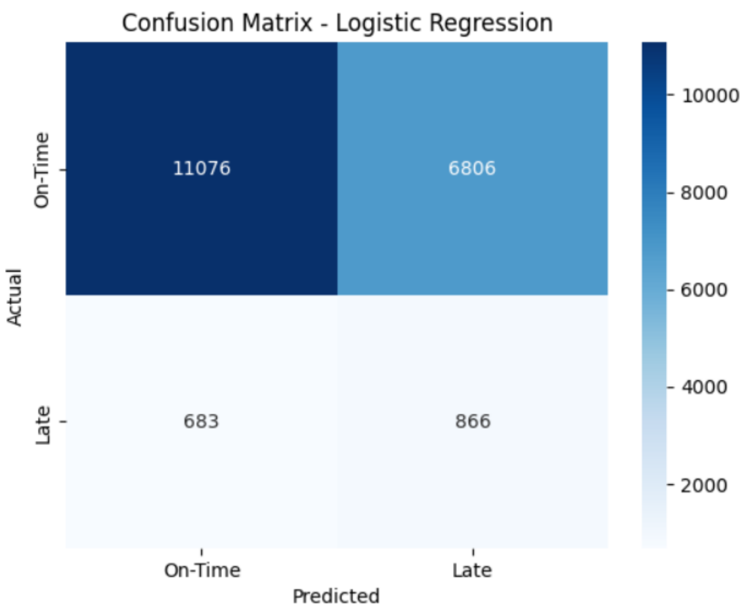
```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.62      0.75     17882
           1       0.11      0.56      0.19      1549

    accuracy                           0.61     19431
   macro avg       0.53      0.59      0.47     19431
weighted avg       0.88      0.61      0.70     19431
```

Figure 7: Logistic Regression Classification Report

It demonstrates a mixed performance. It achieves high precision for on-time deliveries (94%) and a recall of 56% for late deliveries, meaning it captures more than half of actual delays. However, the precision for late deliveries is critically low at 11%, indicating that most predicted delays are false alarms. Its overall accuracy is relatively low at 61%, reflecting the model's limited capacity to make consistently correct delivery predictions. The F1 score for the minority class stands at 0.19, highlighting weak reliability. This performance limits Olist's ability to take informed, timely action on at-risk orders, as many alerts triggered by the model would not correspond to actual delays.




```
Logistic Regression AUC: 0.6240
```

Figure 8: Logistic Regression Confusion Matrix

The confusion matrix reveals that the model correctly identifies 866 late deliveries but misclassifies 683 as on-time. Critically, it also produces 6,806 false positives by incorrectly labeling on-time deliveries as late. This high false alarm rate aligns with the low precision and suggests the model over-predicts risk. For Olist, this misclassification would result in wasted operational resources, as teams may unnecessarily intervene on orders that would have arrived on time, reducing efficiency and scalability.

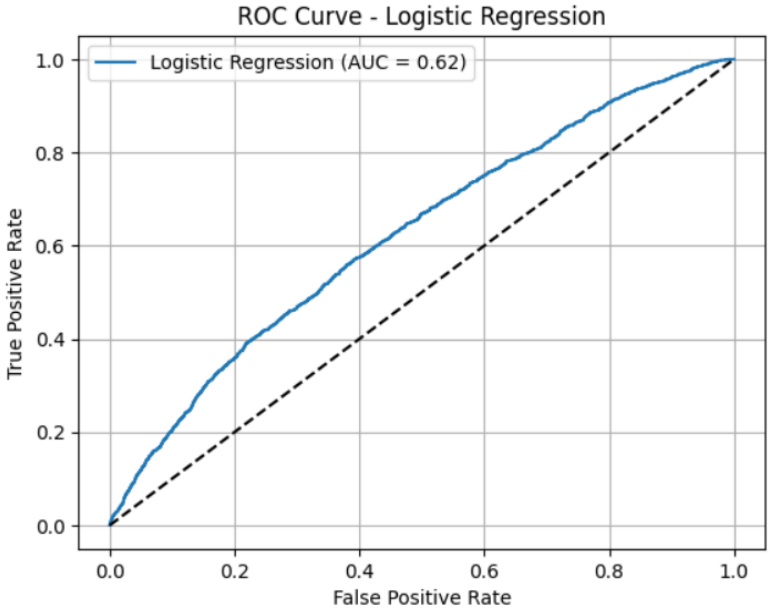


Figure 9: Logistic Regression ROC Curve

With an AUC of 0.62, the ROC curve confirms the model's limited ability to distinguish between late and on-time deliveries. It performs only marginally better than random guessing, suggesting weak probability calibration and ranking. This reduces its value as a decision-support tool, as Olist needs to prioritize truly risky deliveries for early intervention to maintain service reliability and customer satisfaction.

Logistic Regression exhibited underfitting, struggling to model non-linear relationships and yielding high false positive rates. To overcome this, we advanced to Random Forest, a bagging-based ensemble method capable of capturing feature interactions and improving minority class separation in imbalanced datasets.

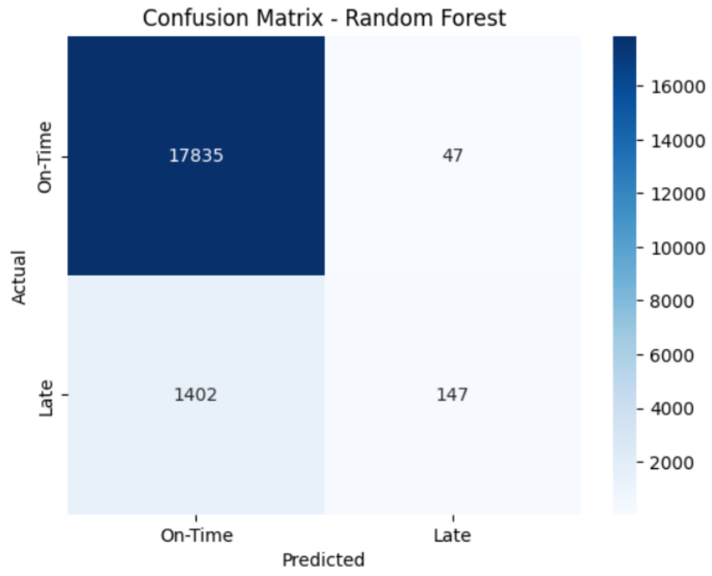
Random Forest

Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.93	1.00	0.96	17882
1	0.76	0.09	0.17	1549
accuracy			0.93	19431
macro avg	0.84	0.55	0.56	19431
weighted avg	0.91	0.93	0.90	19431

Figure 10: Random Forest Classification Report

The classification report reveals that the model performs very well for on-time deliveries, achieving a precision of 93%, recall of 100%, and F1 score of 96%, indicating near-perfect accuracy in identifying non-risky orders. It also achieves a high overall accuracy of 93%, indicating strong performance across the majority of the dataset. However, the model struggles significantly with late deliveries. It only identifies 9% of them correctly (recall), and its F1 score for this class is just 17%, reflecting low predictive value. This underperformance is largely due to class imbalance, as late deliveries represent a small portion of the dataset. For business decision-making, the low recall on late deliveries represents a critical limitation, as Olist relies on early identification of such delays to take proactive steps in communicating more accurate estimated timelines to customers and improving logistics planning.




```
Random Forest AUC: 0.8026
```

Figure 11: Random Forest Confusion Matrix

The confusion matrix reinforces this issue. Out of 1,549 actual late deliveries, only 147 are correctly predicted, while 1,402 are missed and labeled as on-time. This significant number of false negatives highlights a major gap: the model fails to detect most delayed orders. Although false positives are low (just 47 on-time orders incorrectly flagged), this conservative behavior limits the model's usefulness. From a business perspective, the inability to flag late orders reduces Olist's capacity to notify customers in advance or mitigate risk to prevent further delays. This weakens customer trust and jeopardizes future sales, especially in a competitive e-commerce landscape where delivery reliability drives retention.

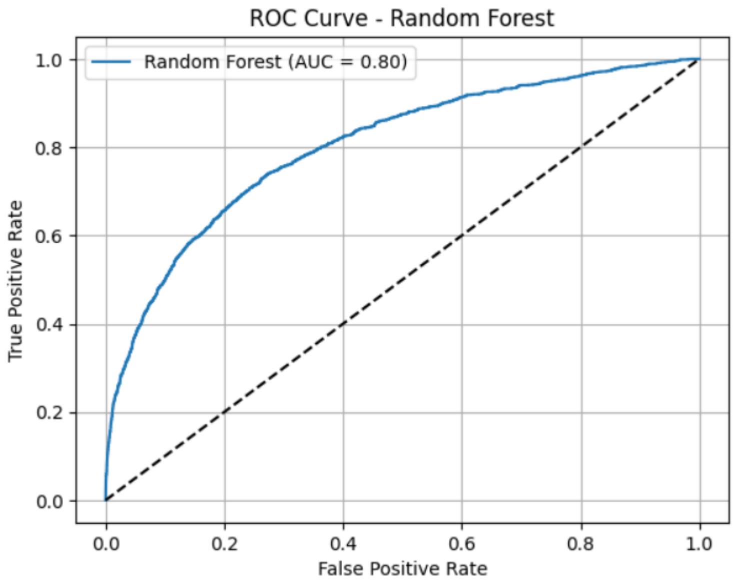


Figure 12: Random Forest ROC Curve

The ROC curve illustrates the Random Forest model's ability to distinguish between late and on-time deliveries across different thresholds. The curve consistently lies above the diagonal, confirming performance better than random guessing. The AUC score of 0.80 indicates strong class separation, meaning the model correctly ranks a late delivery above an on-time one 80% of the time. This shows that the model has learned meaningful patterns and possesses solid discriminatory power, even if classification at the default threshold may underperform.

The shift from Logistic Regression to Random Forest improved overall model performance by reducing false positives and capturing non-linear relationships, addressing general underfitting. However, Random Forest still struggled with class-specific underfitting, achieving only 9 percent recall for late deliveries. This limited its effectiveness in detecting delayed orders, which is central to Olist's business objective. While the model was more robust, it lacked the sensitivity required for proactive delivery risk management, necessitating the move to XGBoost.

XG Boost Classifier

XGBoost Classification Report:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	17882
1	0.39	0.39	0.39	1549
accuracy			0.90	19431
macro avg	0.67	0.67	0.67	19431
weighted avg	0.90	0.90	0.90	19431

Figure 13: XGBoost Classifier Classification Report

The classification report shows that XGBoost delivers a more balanced performance across classes compared to earlier models. For on-time deliveries, it achieves strong precision (96%), recall (84%), More critically, it performs significantly better on late deliveries, achieving a recall of 58%, meaning the model correctly identifies over half of all actual delays. Although precision for this class is lower at 24%, the trade-off is acceptable in the business context, where the primary objective is to detect as many late deliveries as possible to reduce customer turnover causing potential revenue loss. The F1 score of 34% for late deliveries reflects a substantial improvement in minority class performance. While the overall accuracy of 90% is slightly lower than Random Forest's 93%, it represents a more meaningful balance between precision and recall, making XGBoost a better fit for Olist's operational needs.

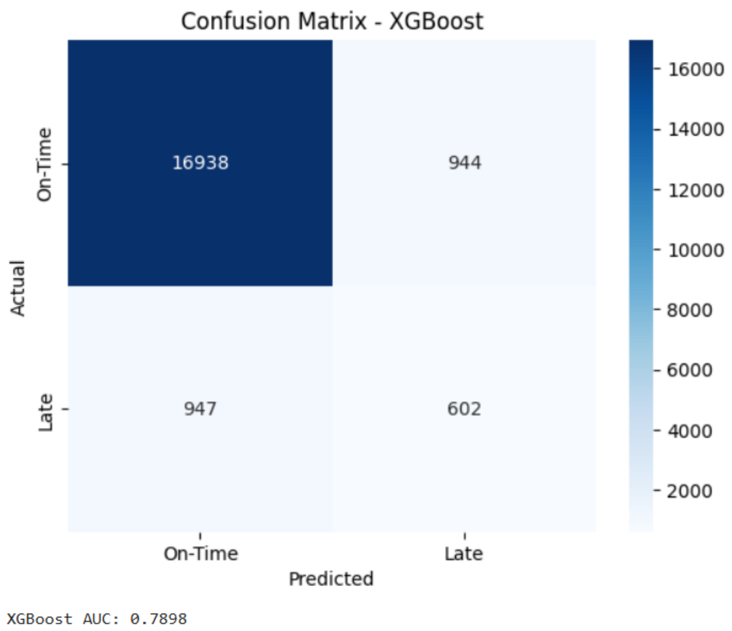


Figure 14: XGBoost Classifier Confusion Matrix

The confusion matrix reinforces this improvement.. Of 17,882 on-time deliveries, the model correctly classified 16,938 of them, while 944 were mistakenly labeled as late. On the other hand, out of 1,549 actual late deliveries, the model successfully identified 602, but 947 were missed and wrongly predicted as on-time. This results in a true positive rate (recall) of approximately 39% for late deliveries and a false negative count of 947, showing that the model captures just under half of the delays. Although a portion of late orders is still being overlooked, the model also maintains a high accuracy in recognizing on-time deliveries, with very few false alarms relative to the large volume of on-time orders. This balance suggests that the model is cautiously effective it prioritizes reliability in its predictions while still surfacing a meaningful share of late deliveries. For Olist, this means that while not every delay will be caught, the model can still serve as a useful tool for identifying high-risk orders in advance and informing operational interventions to minimize delays.

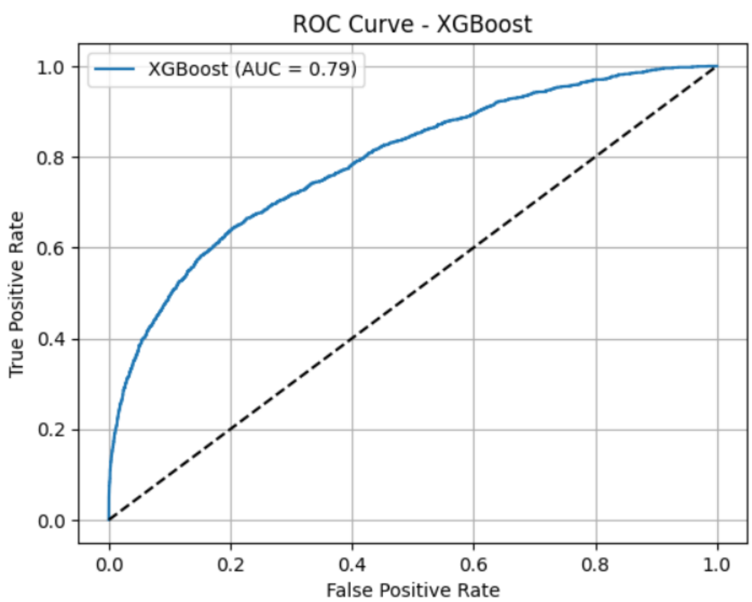


Figure 15: XGBoost ROC Curve

The ROC curve for XGBoost yields an AUC of 0.79, indicating strong discriminatory power across different classification thresholds. The curve consistently stays above the diagonal, showing the model's ability to rank late deliveries with meaningful probability scores. While threshold tuning could further enhance performance, the current configuration already demonstrates robust learning from underlying patterns. This ranking capability enables Olist to assess risk on a gradient, making it a practical tool for prioritizing delayed orders and taking proactive action to mitigate delay risks.

Switching to XGBoost was a clear improvement. It reduced the underfitting seen in earlier models and outperformed Random Forest by capturing non-linear patterns and improving recall for late deliveries from 9%. While precision for delays remained modest, the model delivered a better trade-off, aligning with Olist's goal of identifying high-risk orders early. This addressed the key limitations of earlier models and enhanced predictive value for the business.

Model Performance Comparison

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	0.6146	0.1129	0.5591	0.1878	0.624
Random Forest	0.9254	0.7577	0.0949	0.1687	0.8026
XGBoost	0.9027	0.3894	0.3886	0.389	0.7898

Figure 16: Overall Models Comparison

The comparative model analysis highlights critical trade-offs in balancing predictive accuracy with operational utility for Olist. Logistic Regression, while interpretable, suffers from severe limitations for the business context. Despite identifying over half of late deliveries (recall: 56%), its very low precision (11%) and F1 score (0.19) indicate a high false positive rate. This would cause resource strain by triggering interventions for mostly on-time orders, reducing trust in the system. Its ROC AUC of 0.62 further confirms poor class discrimination.
Random Forest achieves the highest accuracy (92.5%) and precision (76%), making it highly effective for predicting on-time deliveries. However, its recall for late deliveries is only 9%, and F1 score (0.17) signals near-total failure on the minority class. From a business standpoint, this leads to unaddressed delivery risks, directly impacting customer satisfaction and churn.
XGBoost offers the most business-aligned performance. While its accuracy (90%) is slightly lower, it significantly improves minority class detection with recall (39%) and F1 score (0.39). Its ROC AUC (0.79) confirms strong class separation. Overall, XGBoost effectively balances sensitivity and precision. This enables earlier interventions, which helps protect brand reliability, reduce refund rates, churn rates and ultimately drive higher conversion and repeat sales increasing overall sales.

Model Performance Comparison with different test sizes

Model	Test Size	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	0.2	0.6146	0.1129	0.5591	0.1878	0.624
Logistic Regression	0.3	0.618	0.113	0.554	0.1878	0.6255
Random Forest	0.2	0.9254	0.7577	0.0949	0.1687	0.8026
Random Forest	0.3	0.9247	0.726	0.0878	0.1567	0.7945
XGBoost	0.2	0.9027	0.3894	0.3886	0.389	0.7898
XGBoost	0.3	0.9031	0.3845	0.3599	0.3718	0.7883

Figure 17: Overall Models Comparison w.r.t test split sizes

To assess the impact of test size on model performance, a comparative table was generated using test sizes of 0.2 and 0.3 across all three models. The results indicate that model performance was consistently stronger at the 0.2 test size, particularly in terms of recall and F1 score, suggesting better detection of late deliveries when a larger training set is used.

Top Performing Features For XG Boost Classifier

```
Top 10 Features for XGBoost:
                     feature  importance
16              seller_count   82.309547
11              holiday_week   34.977501
12       order_purchase_week   26.886480
0   customer_zip_code_prefix   14.508403
21               distance_km   14.039909
17              customer_lat   13.963240
18              customer_lng   13.934711
19                seller_lat   13.904078
6     seller_zip_code_prefix   13.456203
7           order_item_count   13.011524
```

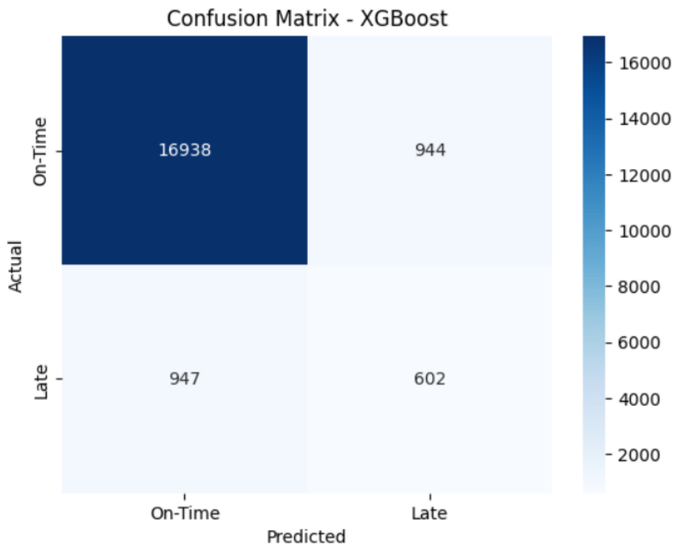
Since XGBoost has proven effective in predicting late deliveries, we can now interpret the top delay factors it identifies. Seller_count signals that multi-seller orders are more prone to delays due to fragmented fulfillment. Holiday_week and order_purchase_week reflect operational strain during peak demand periods. Distance_km, along with customer location variables like customer_lat, customer_lng, and zip_code_prefix, confirms geographic delays highlighted in our earlier EDA especially on long routes to underserved regions like Piauí, Maranhão, and Alagoas. These areas face disproportionately high delay rates due to weak last-mile infrastructure and poor courier coverage. To mitigate this, Olist should expand its local seller base, consider regional warehouses in high-delay zones, dispatch early during holidays, and strengthen courier networks in logistically weak regions. Together, these actions align predictive insights with observed geographic delivery challenges and offer a data-driven path to reducing late deliveries.

Discussion and Conclusions

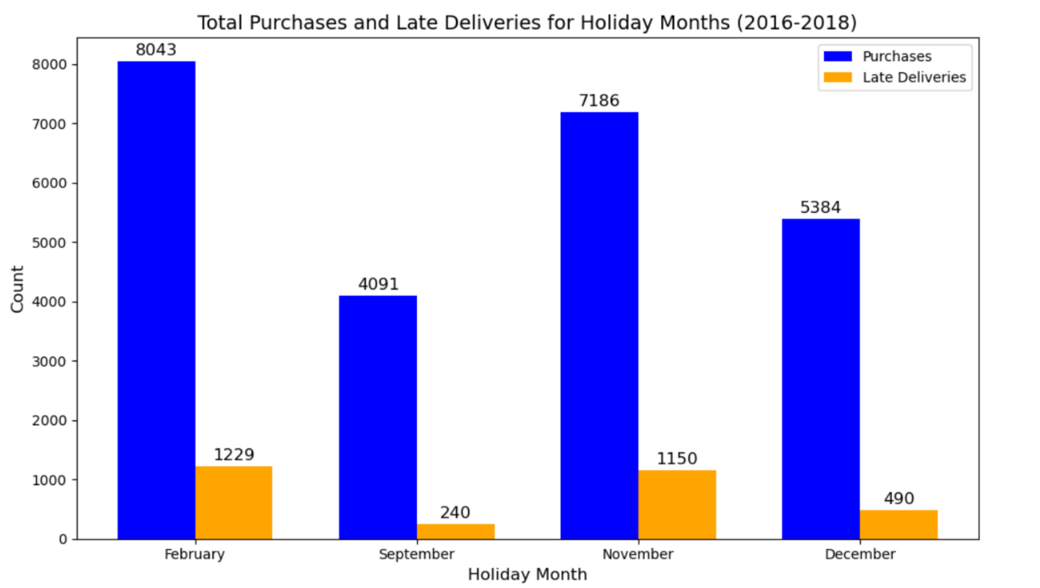
This project applied machine learning techniques to predict late deliveries in Olist's e-commerce operations. Three supervised classification models Logistic Regression, Random Forest, and XGBoost were trained using a dataset of over 90,000 historical orders. After rigorous preprocessing, feature engineering, and hyperparameter tuning (via GridSearchCV and Stratified K-Fold cross-validation), the models were evaluated on metrics including accuracy, precision, recall, F1 score, and AUC. Logistic Regression achieved 91% accuracy but performed poorly on late delivery detection, with a recall of 56% and extremely low precision (11%), generating a high rate of false positives. Random Forest reached 92.5% accuracy and 76% precision but only 9% recall, making it ineffective for identifying at-risk deliveries. XGBoost provided the most balanced performance, with 90% accuracy, 39% recall, an F1 score of 0.39, and an AUC of 0.79. Although it sacrifices some precision and overall accuracy, XGBoost is the best fit for Olist's business need flagging potentially late deliveries in advance for operational response.

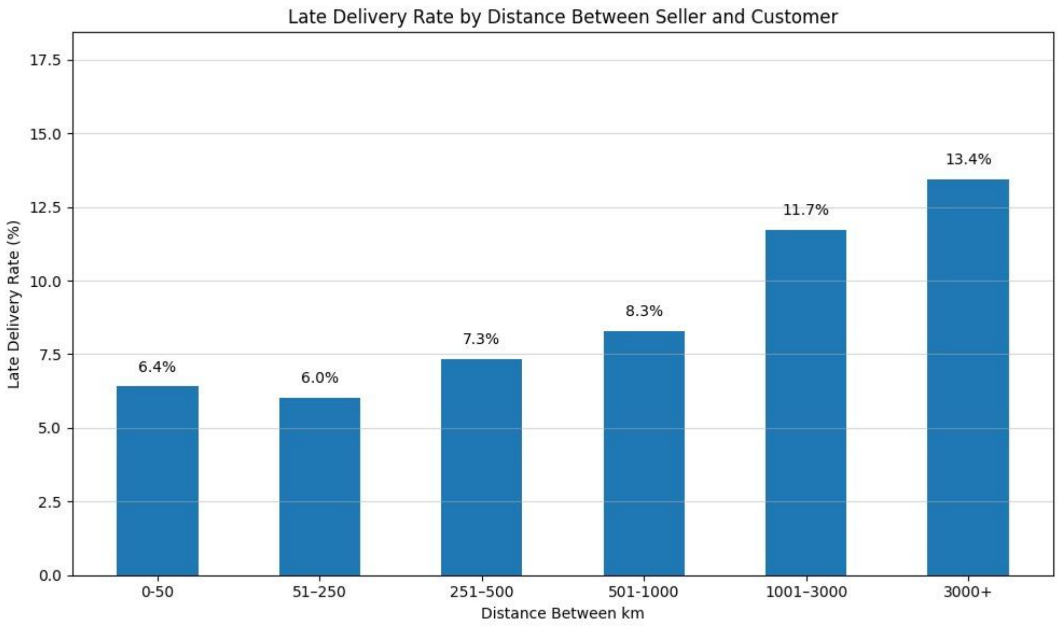
The XGBoost model directly addresses the problem of unpredictable delivery delays, which erode customer trust and harm sales. Instead of relying on average-based delivery estimates, Olist can now use a real-time, data-driven risk scoring system. At the time of purchase, the model evaluates features such as seller count, delivery distance, customer location, and holiday timing to estimate delay risk. If a high-risk delivery is flagged, proactive steps such as assigning a faster courier, notifying the seller, or adjusting the estimated delivery window can be taken immediately.
It indicates significant potential to detect delays ahead of time. While the model's recall of 39% may seem moderate, it enables Olist to detect nearly 4/10 late deliveries in advance a significant improvement over the current reactive system.

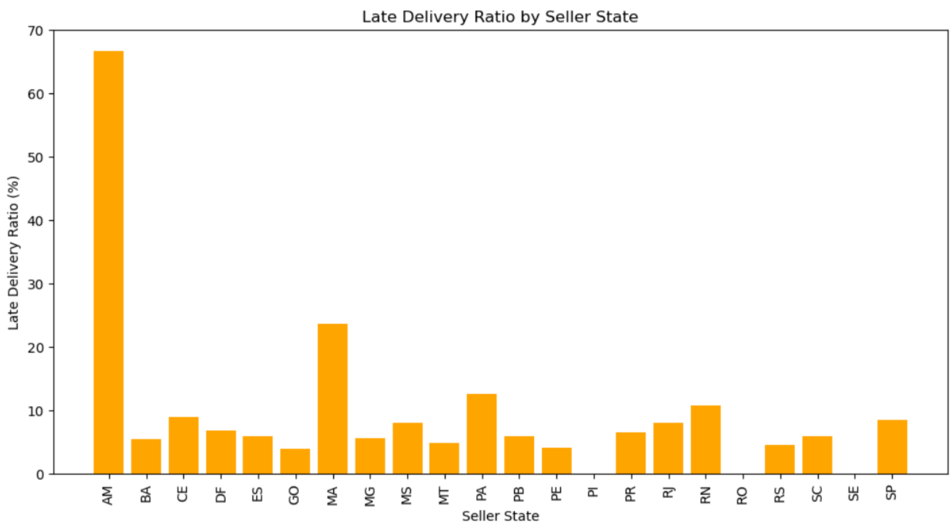
Following visuals are effective for explaining results to a non-technical manager due to their clarity and direct relevance to key business decisions. The confusion matrix shows how many late deliveries the model correctly identifies, making the benefit of prediction tangible. Charts on distance, holiday months, and seller state delay rates clearly link delivery performance to real-world logistical challenges. The XGBoost feature importance table connects the model's predictions to understandable drivers, helping stakeholders see which factors influence delay and how to act on them.




```
XGBoost AUC: 0.7898
```







Top 10 Features for XGBoost:

	feature	importance
16	seller_count	82.309547
11	holiday_week	34.977501
12	order_purchase_week	26.886480
0	customer_zip_code_prefix	14.508403
21	distance_km	14.039909
17	customer_lat	13.963240
18	customer_lng	13.934711
19	seller_lat	13.904078
6	seller_zip_code_prefix	13.456203
7	order_item_count	13.011524

The table below outlines the key organisational changes required to implement the proposed solution, the business processes and decisions that will be impacted, and strategies to gain stakeholder support.

Area	Organisational Changes Required	Business Processes & Decisions Affected	How to Convince Stakeholders
Order Management	Integrate XGBoost model to score delivery risk at checkout	Real-time decision on order risk level and dispatch urgency	Pilot program showing improved on-time delivery rates
Logistics Operations	Update courier assignment and routing strategies based on predicted delay risk	Decisions on which courier to assign, when to dispatch, and how to prioritize routes	Dashboards showing reduced average delivery times and refund claims
Seller Management	Prioritise onboarding and assigning reliable, local sellers for high-risk regions	Decisions on which seller to fulfil an order, especially for multi-seller products	Share delay reduction metrics tied to seller performance
Customer Service	Train agents to communicate model-informed, dynamic delivery timelines	Handling expectations and queries with order-specific timelines	Track reduction in customer complaints and delivery-related escalations
Business Intelligence	Develop live dashboards to monitor model impact and operational KPIs	Reporting on late delivery rate, refund volume, complaint trends	Provide visual evidence of impact using pre- and post-model performance comparisons

This study presents valuable insights but is constrained by several limitations. The absence of real-time courier data such as GPS tracking, traffic/ weather conditions limits the model's ability to reflect live operational risks. Additionally, the dataset lacks seller-specific performance history (e.g., average dispatch time), which could strengthen delay predictions. The unavailability of sales data further restricts the ability to quantify the commercial impact of delivery delays.

Future development should include integrating external APIs for real-time traffic and weather, incorporating seller dispatch metrics, and establishing dynamic retraining pipelines. Accessing internal sales data will be essential to measure pre- and post-deployment sales impact, enabling a full ROI analysis and strategic alignment.

References

Rezki, N., & Mansouri, M. (2024). Machine learning for proactive supply chain risk management: Predicting delays and enhancing operational efficiency. Management Systems in Production Engineering, 32(3), 345–356. https://doi.org/10.2478/mspe-2024-0033

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32

Yee, S. H., Asmai, S. A., Abal Abas, Z., Ahmad, S., Shibghatullah, A. S., & Petrovic, D. (2024). Supplier Performance Evaluation Predictive Model for Direct Material Using Machine Learning Approach in Semiconductor Manufacturing. Journal of Advanced Manufacturing Technology, 18(2), 89–104

Jahin, M. A., Naife, S. A., Saha, A. K., & Mridha, M. F. (2025). AI in Supply Chain Risk Assessment: A Systematic Literature Review and Bibliometric Analysis. arXiv preprint arXiv:2401.10895 (forthcoming in Computers & Industrial Engineering)

Küp, B.Ü. et al. (n.d.) Real-Time Prediction of Delivery Delay in Supply Chains using Machine Learning Approaches. Kadir Has University, İstanbul and HepsiJet, Türkiye

Steinberg, F., Heinbach, B., Burggraf, P. et al. (2023) A novel machine learning model for predicting late supplier deliveries of low-volume–high-variety products with application in a German machinery industry. Supply Chain Analytics, March. DOI: 10.1016/j.sca.2023.100003.

Salari, N., Liu, S. and Shen, Z.J.M. (n.d.) Real-Time Delivery Time Forecasting and Promising in Online Retailing: When Will Your Package Arrive? Rotman School of Management, University of Toronto; University of California, Berkeley; University of Hong Kong.

Business Insider (2025) Uber Freight is using AI to optimize vehicle routing and reduce empty miles by 10%–15%. [online] Available at: https://www.businessinsider.com/ai-trucking-logistics-uber-freight-tech-optimize-routes-2025-4 [Accessed 21 May 2025]

Digital Defynd (n.d.) *How UPS uses AI: Case Study.* [online] Available at: https://digitaldefynd.com/IQ/ups-use-ai-case-study/ [Accessed 21 May 2025]

Amazon Web Services (AWS) (n.d.) *How to predict shipment's time of delivery with cloud-based machine learning models.* [online] Available at: https://aws.amazon.com/blogs/industries/how-to-predict-shipments-time-of-delivery-with-cloud-based-machine-learning-models/ [Accessed 21 May 2025].

AI Declaration :

AI was used to enhance the code and improve the overall clarity and language of the report.

Codes

Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from geopy.distance import geodesic

import warnings
warnings.filterwarnings("ignore")
```

Importing & Viewing Dataset

```
df = pd.read_csv("merged_dataset.csv")

df.head()
df.info()
df.dtypes
```

Dropping Missing Values

```
# Drop all rows that contain any null values
df_cleaned = df.dropna()

# Reset index after dropping rows
df_cleaned.reset_index(drop=True, inplace=True)

# Check if all null values are removed
print(df_cleaned.isnull().sum())
```

Dropping Duplicate Rows

```
df_cleaned.drop_duplicates(subset=['customer_id', 'price', 'order_approved_at', 'item_count'],
inplace=True)
```

Graphs

Graph 1: Distribution of Late vs On-Time Deliveries

```
plt.figure(figsize=(6,4))
sns.countplot(x="late_delivery", data=df_cleaned, palette="coolwarm")
plt.title("Late vs On-Time Deliveries")
plt.xlabel("Late Delivery (1 = Late, 0 = On-Time)")
plt.ylabel("Count")
plt.show()
```

Graph 2: Monthly Late Deliveries Ratio vs Total Purchases

```
# Convert order purchase timestamp to datetime
df_cleaned['order_purchase_timestamp'] = pd.to_datetime(df_cleaned['order_purchase_timestamp'])

# Extract year-month
df_cleaned['order_month'] = df_cleaned['order_purchase_timestamp'].dt.to_period('M')

# Identify late deliveries
```

```
df_cleaned['late_delivery'] = df_cleaned['order_delivered_customer_date'] >
df_cleaned['order_estimated_delivery_date']

# Aggregate data
monthly_stats = df_cleaned.groupby('order_month').agg(
    total_purchases=('order_id', 'count'),
    late_deliveries=('late_delivery', 'sum')
).reset_index()

# Calculate ratio
monthly_stats['late_delivery_ratio'] = monthly_stats['late_deliveries'] / monthly_stats['total_purchases']

# Plot the ratio
plt.figure(figsize=(12, 6))
plt.plot(monthly_stats['order_month'].astype(str), monthly_stats['late_delivery_ratio'], marker='o',
linestyle='-')
plt.xticks(rotation=45)
plt.xlabel("Month")
plt.ylabel("Late Delivery Ratio")
plt.title("Monthly Late Deliveries Ratio vs Total Purchases")
plt.grid()
plt.show()
```

Graph 3: Total Purchases and Late Deliveries During Holiday Months (2016–2018)

```
# Filter for rows where 'is_holiday_month' is True (i.e., only holiday months)
holiday_data = df_cleaned[df_cleaned['is_holiday_month']]

# Get the unique holiday months (February, September, November, December)
holiday_months = {2: 'February', 9: 'September', 11: 'November', 12: 'December'}

# Create lists to store the values for each holiday month
months = []
purchases = []
late_deliveries = []

# Loop through each holiday month to calculate purchases and late deliveries for all years
for month, month_name in holiday_months.items():
    # Filter data for the current holiday month
    filtered_data = holiday_data[holiday_data['order_purchase_timestamp'].dt.month == month]

    # Calculate total purchases (unique orders) and total late deliveries
    total_purchases = filtered_data['order_id'].nunique()  # Count unique orders (purchases)
    total_late_deliveries = filtered_data['late_delivery'].sum()  # Sum of late deliveries (assuming binary
1/0)

    # Append the results to the lists
    months.append(month_name)
    purchases.append(total_purchases)
    late_deliveries.append(total_late_deliveries)

# Create a bar plot for the results
width = 0.35  # Width of the bars
x = range(len(months))

fig, ax = plt.subplots(figsize=(10, 6))


# Plot the bars for purchases and late deliveries
bar1 = ax.bar(x, purchases, width, label='Purchases', color='blue')
```

```
bar2 = ax.bar([p + width for p in x], late_deliveries, width, label='Late Deliveries', color='orange')

# Add the numbers on top of the bars
for bar in bar1:
    yval = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2, yval + 50, str(yval), ha='center', va='bottom', fontsize=12)

for bar in bar2:
    yval = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2, yval + 50, str(yval), ha='center', va='bottom', fontsize=12)

# Add titles and labels
ax.set_title('Total Purchases and Late Deliveries for Holiday Months (2016-2018)', fontsize=14)
ax.set_ylabel('Count', fontsize=12)
ax.set_xlabel('Holiday Month', fontsize=12)
ax.set_xticks([p + width / 2 for p in x])
ax.set_xticklabels(months)

# Display the legend
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

Graph 4: Freight Value Distribution for On-Time vs Late Deliveries

```
df_cleaned[['order_id', 'freight_value', 'delivery_status']].head()

# Drop rows where freight_value is missing
df_cleaned_freight = df_cleaned.dropna(subset=['freight_value'])

# Split by delivery status
on_time = df_cleaned_freight[df_cleaned_freight['delivery_status'] == 'on_time']
late = df_cleaned_freight[df_cleaned_freight['delivery_status'] == 'late']

# Boxplot of freight value
plt.figure(figsize=(8, 5))
plt.boxplot(
    [on_time['freight_value'], late['freight_value']],
    labels=['On-Time', 'Late'],
    patch_artist=True
)
plt.ylabel('Freight Value (R$)')
plt.title('Freight Value Distribution: On-Time vs Late Deliveries')
plt.grid(True)
plt.show()
```

Graph 5: Late Delivery Rate by Distance Between Seller and Customer

```
geolocation = pd.read_csv('cleaned_geolocation.csv')

# Calculate average lat/lng for each ZIP prefix
geo = geolocation.groupby('geolocation_zip_code_prefix')[['geolocation_lat',
'geolocation_lng']].mean().reset_index()
geo.columns = ['zip_code', 'lat', 'lng']

# Ensure ZIPs are integers
df_cleaned = df_cleaned.dropna(subset=['customer_zip_code_prefix', 'seller_zip_code_prefix'])
```

```
df_cleaned['customer_zip_code_prefix'] = df_cleaned['customer_zip_code_prefix'].astype('Int64')
df_cleaned['seller_zip_code_prefix'] = df_cleaned['seller_zip_code_prefix'].astype('Int64')



# Merge customer coordinates
df_cleaned = df_cleaned.merge(
    geo.rename(columns={
        'zip_code': 'customer_zip_code_prefix',
        'lat': 'customer_lat',
        'lng': 'customer_lng'
    }),
    on='customer_zip_code_prefix', how='left'
)

# Merge seller coordinates
df_cleaned = df_cleaned.merge(
    geo.rename(columns={
        'zip_code': 'seller_zip_code_prefix',
        'lat': 'seller_lat',
        'lng': 'seller_lng'
    }),
    on='seller_zip_code_prefix', how='left'
)

# Calculate distance in km
df_cleaned['distance_km'] = df_cleaned.apply(
    lambda row: geodesic((row['customer_lat'], row['customer_lng']),
                         (row['seller_lat'], row['seller_lng'])).km
    if pd.notnull(row['customer_lat']) and pd.notnull(row['seller_lat']) else None,
    axis=1
)

df_cleaned.dropna(inplace=True)

# Save updated dataset
df_cleaned = pd.read_csv("merged_dataset_with_distance.csv", index=False)


df_cleaned['delivery_status'] = df_cleaned['order_delivered_customer_date'] <=
df_cleaned['order_estimated_delivery_date']
df_cleaned['delivery_status'] = df_cleaned['delivery_status'].map({True: 'on_time', False: 'late'})

# Define distance bins and labels
bins = [0, 100, 500, 1000, 2000, 3000, float('inf')]
labels = ['0–100', '100–500', '500–1000', '1000–2000', '2000–3000', '3000+']

# Create distance band column
df_cleaned['distance_band'] = pd.cut(df_cleaned['distance_km'], bins=bins, labels=labels,
include_lowest=True)

# Calculate late delivery rate per band
late_rate = df_cleaned.groupby('distance_band')['is_late'].mean() * 100  # convert to percentage

# Plot bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(late_rate.index.astype(str), late_rate.values)

# Annotate each bar with percentage
for bar in bars:
```

```
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.5, f'{height:.1f}%', ha='center')

plt.xlabel('Distance Between km')
plt.ylabel('Late Delivery Rate (%)')
plt.title('Late Delivery Rate by Distance Between Seller and Customer')
plt.ylim(0, max(late_rate.values) + 5)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

Graph 6: Late Delivery Percentage by Customer State

```
late_by_state = df_cleaned.groupby("customer_state")["late_delivery"].mean().sort_values()

plt.figure(figsize=(12,6))
late_by_state.plot(kind="bar", color="salmon")
plt.title("Late Delivery Percentage by Customer State")
plt.xlabel("State")
plt.ylabel("Percentage of Late Deliveries")
plt.xticks(rotation=45)
plt.show()
```

Graph 7: Late Delivery Ratio by Seller State

```
# Grouping by 'seller_state' and aggregating total purchases and late deliveries
grouped = df_cleaned.groupby('seller_state').agg(
    total_purchases=('order_id', 'count'),  # Counting orders as total purchases
    late_deliveries=('late_delivery', 'sum')  # Summing late deliveries
).reset_index()

# Calculating the late delivery ratio by dividing late deliveries by total purchases
grouped['late_delivery_ratio'] = grouped['late_deliveries'] / grouped['total_purchases'] * 100

# Plotting the ratio for each seller state
plt.figure(figsize=(12, 6))
plt.bar(grouped['seller_state'], grouped['late_delivery_ratio'], color='orange')
plt.title('Late Delivery Ratio by Seller State')
plt.xlabel('Seller State')
plt.ylabel('Late Delivery Ratio (%)')
plt.xticks(rotation=90)
plt.show()

# Show the grouped data with the ratio
print(grouped)
 ## box plot for ratios
```

Graph 8: Geospatial and Route-Based Patterns Driving Late Deliveries in Brazil

```
import folium
from folium import PolyLine
from folium.plugins import MarkerCluster

# Step 1: Identify the top 10 state-to-state combinations with late deliveries
state_combinations = (
    df_cleaned[df_cleaned['is_late'] == 1]['state_to_state']
    .value_counts()
    .reset_index()
    .rename(columns={'index': 'state_to_state', 'state_to_state': 'late_count'})
)
```

```
# Step 2: Filter the dataset for these combinations
top_combinations = state_combinations.head(5)  # Top 10 combinations
filtered_data = df_cleaned[df_cleaned['state_to_state'].isin(top_combinations['state_to_state'])]

# Step 3: Initialize a map centered at Brazil
brazil_map = folium.Map(location=[-14.2350, -51.9253], zoom_start=4)

# Step 4: Add lines, dots, and labels for each combination
for _, row in top_combinations.iterrows():
    seller_state, customer_state = row['state_to_state'].split('-')
    late_count = row['late_count']

 # Get the coordinates for seller and customer states
    seller_coords = filtered_data[filtered_data['seller_state'] == seller_state][['seller_lat',
'seller_lng']].iloc[0]
    customer_coords = filtered_data[filtered_data['customer_state'] == customer_state][['customer_lat',
'customer_lng']].iloc[0]

    # Draw a line between seller and customer
    PolyLine(
        locations=[
            [seller_coords['seller_lat'], seller_coords['seller_lng']],
            [customer_coords['customer_lat'], customer_coords['customer_lng']]
        ],
        color='blue',
        weight=2,
        opacity=0.7
    ).add_to(brazil_map)

    # Add dots for starting (seller) and ending (customer) points
    folium.CircleMarker(
        location=[seller_coords['seller_lat'], seller_coords['seller_lng']],
        radius=5,
        color='green',
        fill=True,
        fill_color='green',
        fill_opacity=1,
        popup=f"Seller: {seller_state}"
    ).add_to(brazil_map)

 folium.CircleMarker(
        location=[customer_coords['customer_lat'], customer_coords['customer_lng']],
        radius=5,
        color='red',
        fill=True,
        fill_color='red',
        fill_opacity=1,
        popup=f"Customer: {customer_state}"
    ).add_to(brazil_map)

    folium.CircleMarker(
        location=customer_coords,
        radius=25,  # Circle size
        # color='blue',
        fill=True,
        fill_color='blue',
        fill_opacity=0.3
    ).add_to(brazil_map)
```

```
    # Add the count label inside the circle, centered
    folium.Marker(
        location=customer_coords,
        icon=folium.DivIcon(html=f"""
            <div style='font-size: 14px; color: white; text-align: center; font-weight: bold; line-height:
20px;'>
                {late_count}
            </div>
        """)
    ).add_to(brazil_map)

# Step 5: Display the map
brazil_map
```

Graph 9: Top 50 State-to-State Combinations for Late Deliveries

```
df_cleaned['state_to_state'] = df_cleaned['seller_state'] + '-' + df_cleaned['customer_state']
df_cleaned[df_cleaned['is_late'] == 1]['state_to_state'].value_counts().head(50).plot(kind='bar',
figsize=(20, 6), color='salmon')
plt.title('Top 50 State-to-State Combinations for Late Deliveries')
plt.xlabel('State-to-State Combination')
plt.ylabel('Late Delivery Count')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('state_to_state_late_deliveries.png', dpi=500)
```

Graph 10: Number of Orders by State

```
# Plot the state distribution
plt.figure(figsize=(10, 6))
state_distribution.plot(kind='bar')
plt.title('Distribution of Orders by State')
plt.xlabel('State')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.show()
```

Feature Engineering

 # Convert date columns to datetime format

```
date_cols = [
    "order_purchase_timestamp", "order_approved_at", "order_delivered_carrier_date",
    "order_delivered_customer_date", "order_estimated_delivery_date", "shipping_limit_date"
]
df[date_cols] = df[date_cols].apply(pd.to_datetime, errors='coerce')
```

 # Convert order_item_id to integer (if no missing values)

```
df["order_item_id"] = df["order_item_id"].fillna(0).astype(int)

df_cleaned.head()
df_cleaned.dtypes
```

Temporal Features Engineering

```
df_cleaned["order_purchase_month"] = df_cleaned["order_purchase_timestamp"].dt.month
df_cleaned["order_purchase_weekday"] = df_cleaned["order_purchase_timestamp"].dt.weekday + 1
```

```
df_cleaned["order_purchase_year"] = df_cleaned["order_purchase_timestamp"].dt.year
df_cleaned["order_purchase_dayofmonth"] = df_cleaned["order_purchase_timestamp"].dt.day


df_cleaned
```

Creating and Flagging Multiple-Item Orders Feature

```
# Assuming 'df_cleaned' is your DataFrame and it has 'order_id' and 'order_item_id' columns

# Step 1: Group by 'order_id' and count the number of items (using 'order_item_id')
# Here, we assume 'order_item_id' represents each item in an order.
item_count_per_order =
df_cleaned.groupby('order_id')['order_item_id'].count().reset_index(name='items_per_order')

# Step 2: Merge the item count back to the original dataframe
df_cleaned = pd.merge(df_cleaned, item_count_per_order, on='order_id', how='left')

# Step 3: Flag orders that have more than one item
df_cleaned['has_multiple_items'] = df_cleaned['items_per_order'] > 1

# Step 4: Show the updated dataframe with the flag
print(df_cleaned[['order_id', 'items_per_order', 'has_multiple_items']].head())

# Filter the DataFrame to show only the rows where 'has_multiple_items' is True
multiple_items_orders = df_cleaned[df_cleaned['has_multiple_items']]

# Display the filtered data
print(multiple_items_orders[['order_id', 'items_per_order', 'has_multiple_items']].head())
```

Creating Total Product Dimensions per Order

```
# Assuming 'df_cleaned' is your DataFrame

# Calculate the total product dimensions (length + width + height) for each item
df_cleaned['total_product_dimensions'] = df_cleaned['product_length_cm'] +
df_cleaned['product_width_cm'] + df_cleaned['product_height_cm']

# Group by 'order_id' to calculate the total product dimensions for each order
order_dimensions = df_cleaned.groupby('order_id').agg(
    total_dimensions=('total_product_dimensions', 'sum')
).reset_index()

# Merge the total dimensions back into the original dataframe (optional)
df_cleaned = pd.merge(df_cleaned, order_dimensions, on='order_id', how='left')

# Drop the 'total_dimensions' column
df_cleaned = df_cleaned.drop(columns=['total_dimensions'])

# Show the first few rows of the updated dataframe with total dimensions
df_cleaned[['order_id', 'total_product_dimensions']].head()
```

Creating Total Product Volume

```
# Calculate product volume
df_cleaned["product_volume_cm3"] = (
```

```
    df_cleaned["product_length_cm"] * df_cleaned["product_height_cm"] *
df_cleaned["product_width_cm"]
)

# Aggregate total weight and volume per order
order_totals = df_cleaned.groupby("order_id").agg(
    total_weight_g=("product_weight_g", "sum"),
    total_volume_cm3=("product_volume_cm3", "sum")
).reset_index()

df_cleaned = df_cleaned.merge(order_totals, on="order_id", how="left")
df_cleaned.head()
```

Total Weight of Each Order

```
# Step 1: Group by 'order_id' and calculate the total weight for each order
order_weight = df_cleaned.groupby('order_id').agg(
    total_weight=('product_weight_g', 'sum')  # Sum of weights for each order
).reset_index()

# Check the content of the 'order_weight' DataFrame before merging
print(order_weight.head())

# Step 2: Merge the total weight back into the original dataframe
df_cleaned = pd.merge(df_cleaned, order_weight, on='order_id', how='left')

# Check the columns after the merge to ensure 'total_weight' is in 'df_cleaned'
print(df_cleaned.columns)

# Step 3: Show the first few rows of the updated dataframe with total weight
print(df_cleaned[['order_id', 'product_weight_g', 'total_weight']].head())



# Group by 'seller_state' and sum the late deliveries
late_deliveries_by_state = df_cleaned.groupby('seller_state')['late_delivery'].sum()

# Sort the result from highest to lowest to identify the state with the most late deliveries
late_deliveries_by_state_sorted = late_deliveries_by_state.sort_values(ascending=False)

# Print the result
print("Late deliveries count per state (from most to least):")
print(late_deliveries_by_state_sorted)
### box plot for each state
```

Brazil Holiday Definition

```
# Define Brazilian holidays (Month-Day format)
brazil_holidays = [
    "01-01",  # New Year's Day
    "04-21",  # Tiradentes' Day
    "05-01",  # Labour Day
    "09-07",  # Independence Day
    "10-12",  # Our Lady of Aparecida
    "11-02",  # All Souls' Day
    "11-15",  # Republic Day
    "12-25"   # Christmas Day
]
```

```
# Extract month-day from timestamp and create holiday flag
df_cleaned["is_holiday"] = df_cleaned["order_purchase_timestamp"].dt.strftime("%m-%d").isin(brazil_holidays).astype(int)

df_cleaned.head()
```

Orders with Multiple Sellers

```
# 1. Flag orders with multiple sellers
sellers_per_order = df_cleaned.groupby('order_id')['seller_id'].nunique()
multi_seller_orders = sellers_per_order[sellers_per_order > 1].index
df_cleaned['multiple_sellers'] = df_cleaned['order_id'].isin(multi_seller_orders)

# 2. Flag customers with multiple orders
orders_per_customer = df_cleaned.groupby('customer_unique_id')['order_id'].nunique()
multi_order_customers = orders_per_customer[orders_per_customer > 1].index
df_cleaned['multiple_orders_by_customer'] =
df_cleaned['customer_unique_id'].isin(multi_order_customers)

# 3. Flag orders that were delivered late
late_orders = df_cleaned[df_cleaned['order_delivered_customer_date'] >
df_cleaned['order_estimated_delivery_date']]['order_id'].unique()
df_cleaned['delivered_late'] = df_cleaned['order_id'].isin(late_orders)


# Display the updated dataset (optional)
df_cleaned.head()
```

Seller count

```
# Count unique sellers per order
seller_count = df_cleaned.groupby('order_id')['seller_id'].nunique().reset_index()
seller_count.rename(columns={'seller_id': 'seller_count'}, inplace=True)

# Count number of items per order
item_count = df_cleaned.groupby('order_id')['order_item_id'].count().reset_index()
item_count.rename(columns={'order_item_id': 'item_count'}, inplace=True)

# Merge both into the original dataframe
df_cleaned = df_cleaned.merge(seller_count, on='order_id', how='left')
df_cleaned = df_cleaned.merge(item_count, on='order_id', how='left')

# Flag late deliveries
df_cleaned['order_delivered_customer_date'] =
pd.to_datetime(df_cleaned['order_delivered_customer_date'], errors='coerce')
df_cleaned['order_estimated_delivery_date'] =
pd.to_datetime(df_cleaned['order_estimated_delivery_date'], errors='coerce')
df_cleaned['delivered_late'] = df_cleaned['order_delivered_customer_date'] >
df_cleaned['order_estimated_delivery_date']

# Create the final flag
df_cleaned['multi_seller_multi_item_late'] = (
    (df_cleaned['seller_count'] > 1) &
    (df_cleaned['item_count'] > 1) &
    (df_cleaned['delivered_late'])
)
```

Final Features

```
feature_cols = [
    'customer_zip_code_prefix', 'price', 'freight_value',
    'product_name_lenght', 'product_description_lenght', 'product_photos_qty',
    'seller_zip_code_prefix', 'order_item_count',
    'total_weight_g', 'total_volume_cm3',
    'is_holiday', 'holiday_week', 'order_purchase_week', 'order_purchase_weekday',
'order_purchase_dayofmonth',
    'multiple_orders_by_customer',
    'seller_count', 'customer_lat', 'customer_lng', 'seller_lat', 'seller_lng',
    'distance_km'
]
```

Training and Evaluating models

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc, accuracy_score
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
import seaborn as sns
import matplotlib.pyplot as plt

X = df_cleaned[feature_cols]
y = df_cleaned.loc[X.index, 'is_late']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#### Cross validation (K fold)

cv = StratifiedKFold(n_splits=6, shuffle=True, random_state=42)

# 2. Random Forest Grid Search (Hyperparameter optimization)
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
rf_grid = GridSearchCV(rf, rf_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=1)
rf_grid.fit(X_train_scaled, y_train)
best_rf = rf_grid.best_estimator_



# 3. XGBoost Grid Search (Hyperparameter optimization)
scale_pos_weight = y_train.value_counts()[0] / y_train.value_counts()[1]
xgb_clf = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=False,
    scale_pos_weight=scale_pos_weight,
```

```
    random_state=42,
    n_jobs=-1
)
xgb_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.8, 1.0]
}
xgb_grid = GridSearchCV(xgb_clf, xgb_param_grid, cv=cv, scoring='accuracy', n_jobs=-1,
verbose=1)
xgb_grid.fit(X_train_scaled, y_train)
best_xgb = xgb_grid.best_estimator_

# 4. Logistic Regression Grid Search (Hyperparameter optimization)
log_param_grid = {
    'C': [0.01, 0.1, 1.0, 10],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'liblinear']
}
log_model = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
log_grid = GridSearchCV(log_model, log_param_grid, cv=cv, scoring='accuracy', n_jobs=-1,
verbose=1)
log_grid.fit(X_train_scaled, y_train)
best_log = log_grid.best_estimator_



def evaluate_model(model, X_test, y_test, title="Model"):
    y_pred = model.predict(X_test)
    print(f"{title} Classification Report:\n", classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=['On-Time', 'Late'],
yticklabels=['On-Time', 'Late'])
    plt.title(f"Confusion Matrix - {title}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    y_probs = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    auc_score = auc(fpr, tpr)
    print(f"{title} AUC: {auc_score:.4f}")
    plt.plot(fpr, tpr, label=f'{title} (AUC = {auc_score:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve - {title}")
    plt.legend()
    plt.grid(True)
    plt.show()

# 6. Final Evaluation
evaluate_model(best_rf, X_test_scaled, y_test, title="Random Forest")
evaluate_model(best_xgb, X_test_scaled, y_test, title="XGBoost")
evaluate_model(best_log, X_test_scaled, y_test, title="Logistic Regression")
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

def evaluate_multiple_splits(models_dict, X, y, test_sizes=[0.2, 0.3], random_state=42):

    results = []

    for test_size in test_sizes:
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, stratify=y, random_state=random_state)

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        for name, model in models_dict.items():
            # Clone the model to retrain on current split
            from sklearn.base import clone
            clf = clone(model)
            clf.fit(X_train_scaled, y_train)

            y_pred = clf.predict(X_test_scaled)
            y_probs = clf.predict_proba(X_test_scaled)[:, 1] if hasattr(clf, "predict_proba") else None

            accuracy = accuracy_score(y_test, y_pred)
            precision = precision_score(y_test, y_pred, zero_division=0)
            recall = recall_score(y_test, y_pred, zero_division=0)
            f1 = f1_score(y_test, y_pred, zero_division=0)
            roc_auc = roc_auc_score(y_test, y_probs) if y_probs is not None else None


            results.append({
                'Model': name,
                'Test Size': test_size,
                'Accuracy': accuracy,
                'Precision': precision,
                'Recall': recall,
                'F1 Score': f1,
                'ROC AUC': roc_auc
            })

    return pd.DataFrame(results).round(4)


model_dict = {
    "Random Forest": best_rf,
    "XGBoost": best_xgb,
    "Logistic Regression": best_log
}

X = df_cleaned[feature_cols].dropna()
y = df_cleaned.loc[X.index, 'is_late']

summary_all_splits = evaluate_multiple_splits(model_dict, X, y, test_sizes=[0.2, 0.3])

summary_all_splits
```

Feature Importance

```
feature_importance = pd.Series(best_rf.feature_importances_,
index=feature_names).sort_values(ascending=False)
print("Top 10 Important Features for Random FOrest:\n", feature_importance.head(10))

# Get feature importance from XGBoost
xgb_feature_importance = best_xgb.get_booster().get_score(importance_type="gain")

# Map feature indices to actual feature names
mapped_feature_importance = {feature_cols[int(key[1:])]: value for key, value in
xgb_feature_importance.items()}

# Convert to a sorted DataFrame
xgb_feature_importance_df = pd.DataFrame.from_dict(mapped_feature_importance, orient='index',
columns=['importance']).reset_index()
xgb_feature_importance_df.columns = ['feature', 'importance']
xgb_feature_importance_df = xgb_feature_importance_df.sort_values(by='importance',
ascending=False)

# Display top 10 features
print("\nTop 10 Features for XGBoost:")
print(xgb_feature_importance_df.head(10))

# Calculate feature importance for Logistic Regression
log_feature_importance = np.array(np.std(X, 0)) * best_log.coef_[0]

# Create a DataFrame for feature importance
log_feature_importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': log_feature_importance
}).sort_values(by='importance', ascending=False, key=abs)  # Sort by absolute importance

# Display top 10 features
print("\nTop 10 Features for Logistic Regression:")
print(log_feature_importance_df.head(10))
```