Report on ShipmentSure: Predicting On-Time Delivery Using Supplier Data

Name	Batch
Arpita Mishra	03

ShipmentSure: Predicting On-Time Delivery Using Supplier Data

1. Introduction

Our project, "ShipmentSure: Predicting On-Time Delivery Using Supplier Data", aims to build a machine learning model that can predict whether a shipment will reach on time or not based on supplier and product-related factors. As the first step, we performed Exploratory Data Analysis (EDA) to understand the dataset, explore feature distributions, identify relationships between variables, and check for class imbalance in the target variable (Reached.on.Time_Y.N).

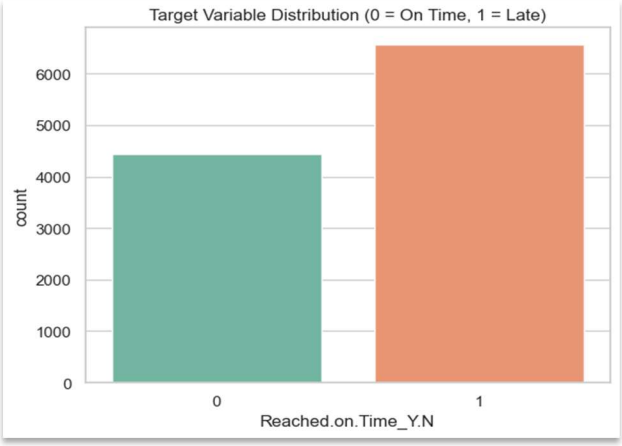
Dataset size: 10,999 records with 12 features (8 numerical, 4 categorical).
Target variable: Reached.on.Time_Y.N (0 = On time, 1 = Late).

2. Univariate Analysis

We examined individual features to understand their distributions:

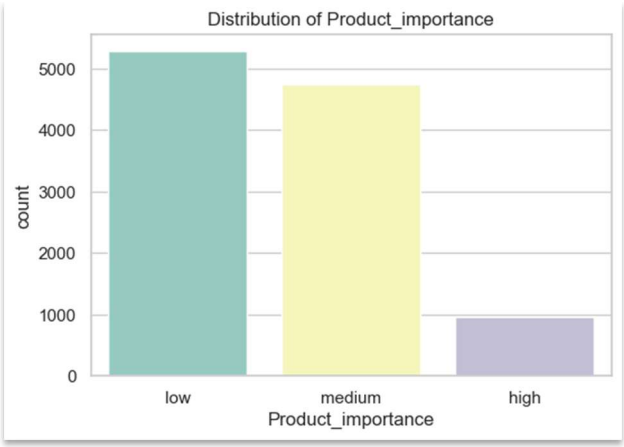
i. Target Variable:

- Class 1 (Late): 6563 (59.7%)
- Class 0 (On Time): 4436 (40.3%)
- Shows mild imbalance toward delayed deliveries.



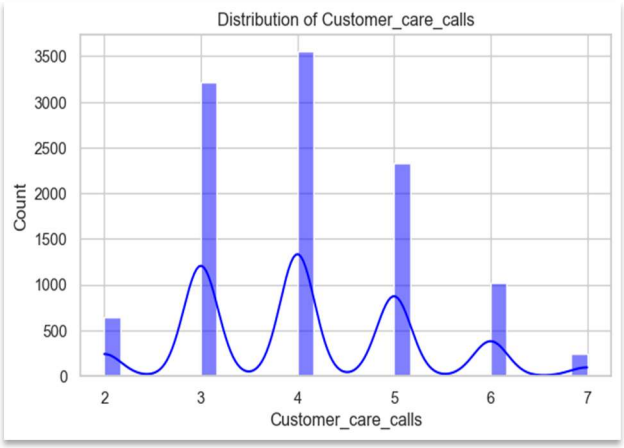
ii. Categorical Features:

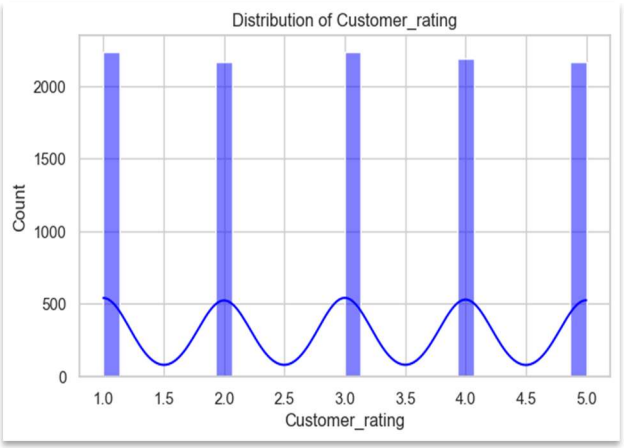
- Warehouse_block: Balanced distribution across blocks A–F.
- Mode_of_Shipment: Mostly "Ship", followed by "Flight" and "Road".
- Product_importance: Majority are "low" importance, fewer "medium" and very few "high".
- Gender: Roughly balanced between Male and Female.

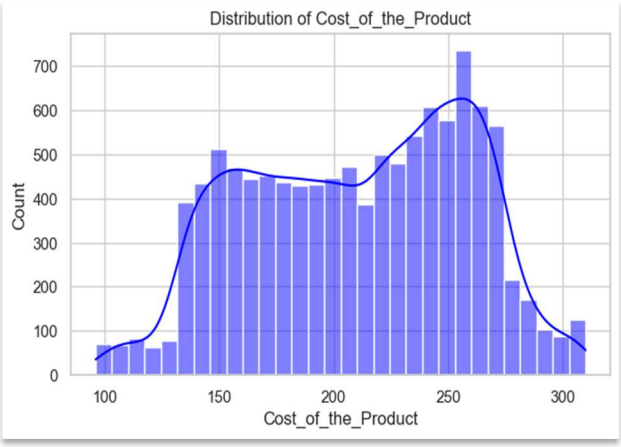


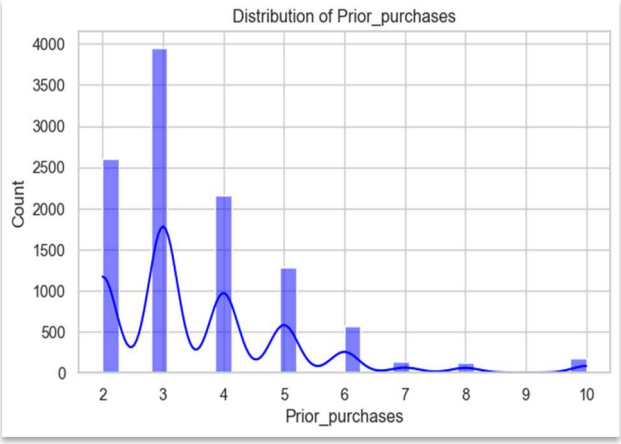
iii. Numerical Features:

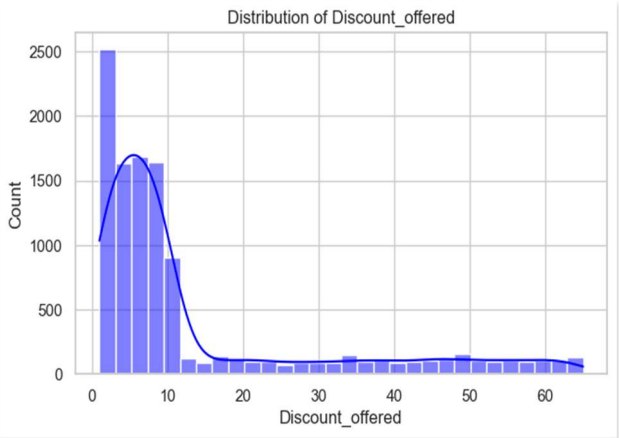
- Customer_care_calls: Most values between 2–5 calls.
- Customer_rating: Fairly uniform from 1–5.
- Cost_of_the_Product: Wide spread with most values clustered in lower ranges.
- Prior_purchases: Majority have 2–5 past purchases.
- Discount_offered: Skewed toward lower discounts.
- Weight_in_gms: Normally distributed with a long tail.

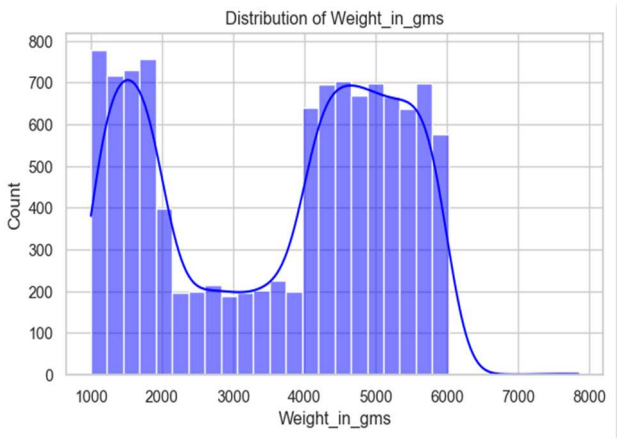










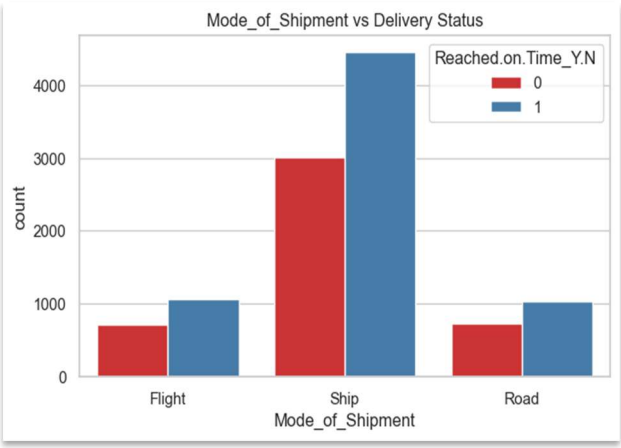


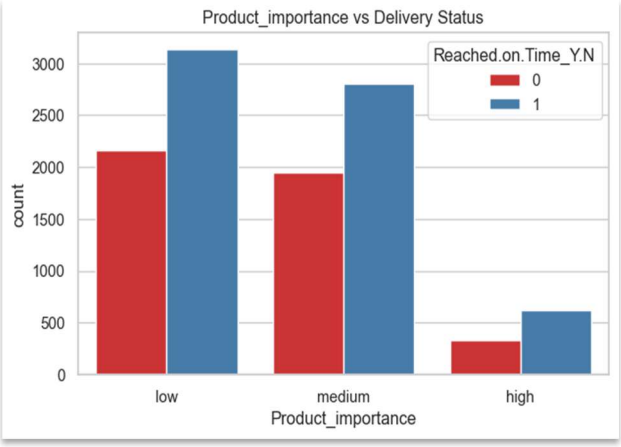
3. Bivariate Analysis

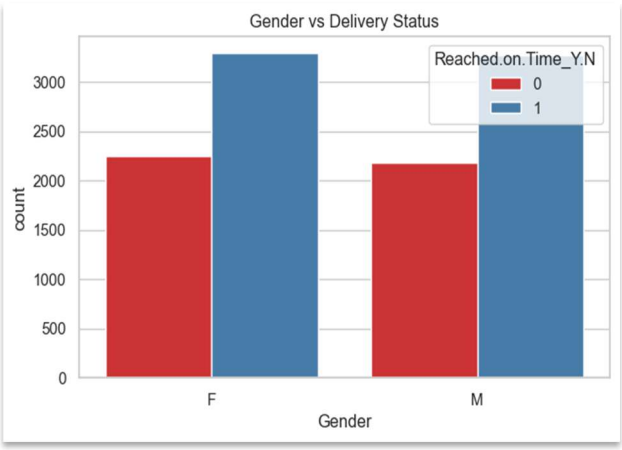
We explored how independent variables relate to the target variable:

i. Categorical vs Target:

- Mode_of_Shipment: Flights tend to have better on-time rates compared to shipping by road.
- Product_importance: High importance products tend to arrive on time more often.
- Warehouse_block: Slight variations, but not strongly predictive on their own.

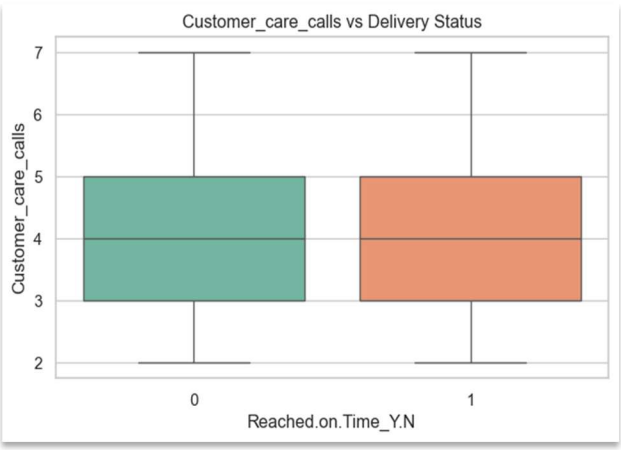


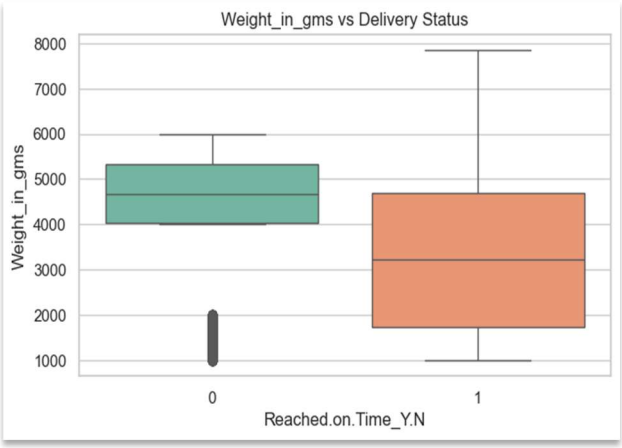




ii. Numerical vs Target:

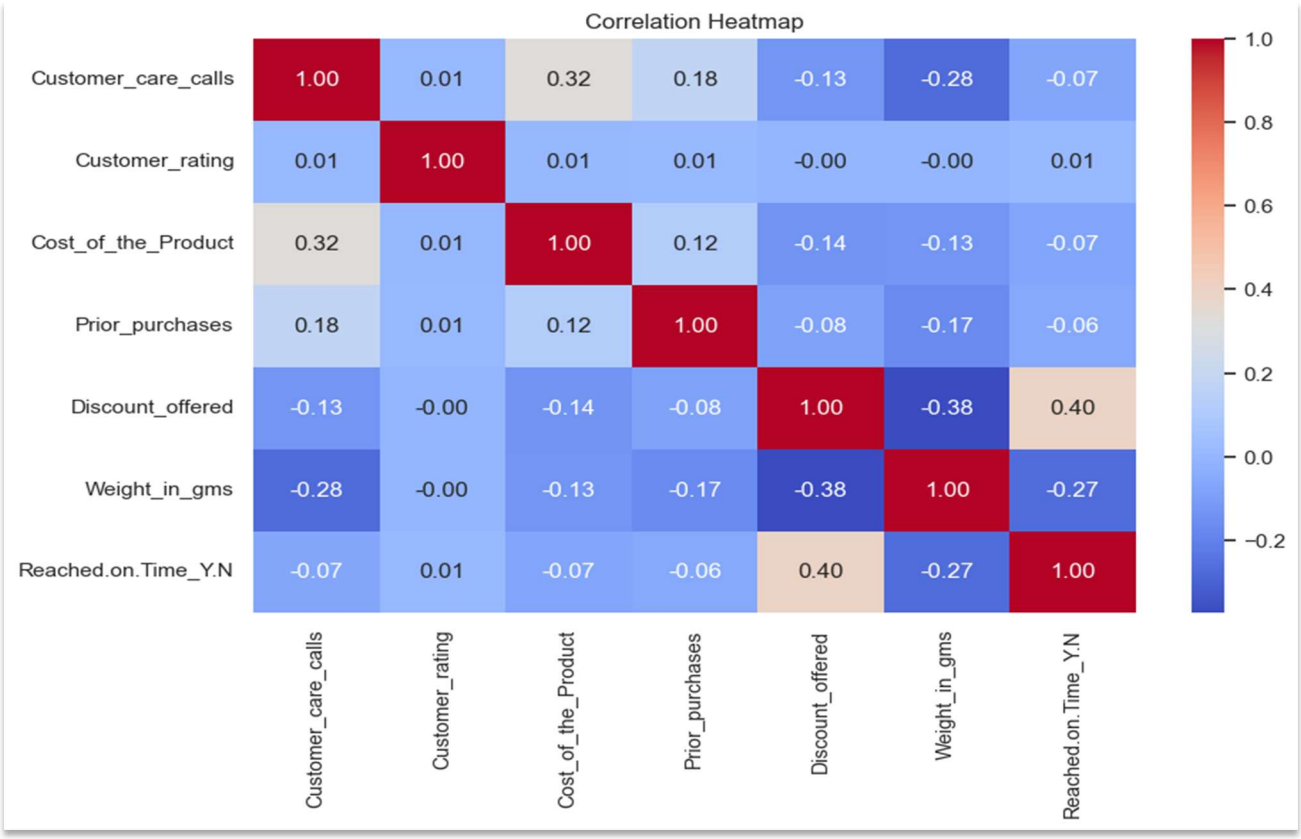
- Discount_offered: Higher discounts are associated with late deliveries.
- Customer_care_calls: Higher calls are linked to late deliveries.
- Weight_in_gms: Heavier products tend to have more delays.
- Customer_rating: Shows little variation with delivery status.





iii. Correlation Heatmap:

- Strong correlation between Weight_in_gms and Cost_of_the_Product.
- Negative relationship between Discount_offered and delivery status (late deliveries linked with higher discounts).



4. Class Imbalance Analysis

We analyzed whether the dataset is balanced for classification:

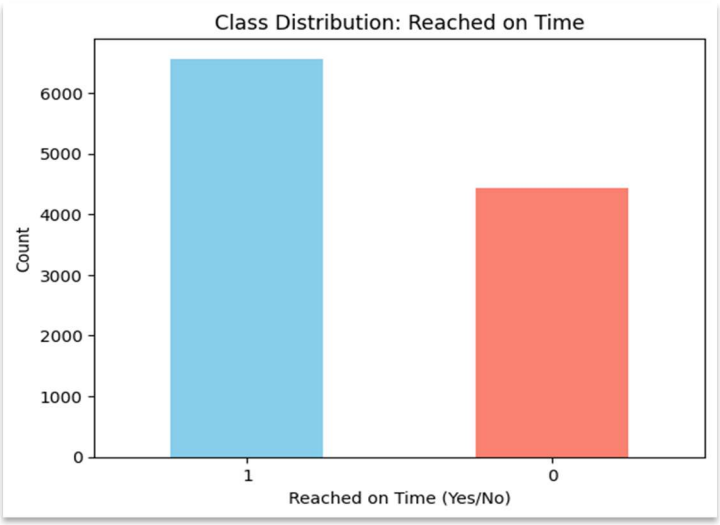
To examine class imbalance in the Reached.on.Time_Yes/No column, means we are checking whether the dataset has a roughly equal number of "Yes" and "No" values—or if one class dominates.

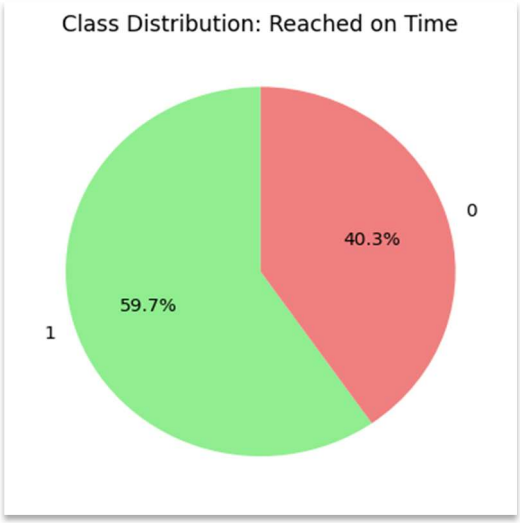
Distribution:

- Class 1 (Late): ~60%
- Class 0 (On Time): ~40%

Visualization:

- Bar plot and pie chart confirm imbalance.





- **Conclusion:**
  Not heavily skewed, but imbalance-handling strategies (e.g., SMOTE, class weights) may be useful for modeling.

Data Preprocessing and Dataset Splitting

1. Handling Missing Values

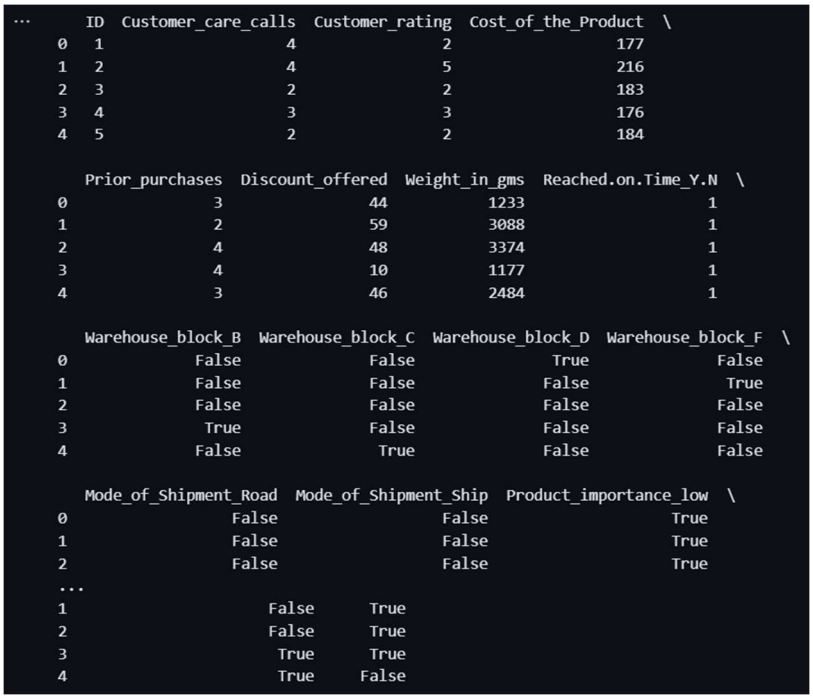
Before applying any machine learning algorithms, it's essential to ensure the dataset is complete and free from inconsistencies. Missing values can lead to biased models or runtime errors. In this project, we performed a thorough check using **df.isnull().sum()** and confirmed that **no columns contained null values**. As a result, no imputation or deletion was necessary, and the dataset was deemed clean and ready for further processing.

2. Encoding Categorical Features

Machine learning models require numerical input, so categorical variables must be transformed appropriately:

- **Label Encoding** was applied to binary or ordinal features such as Gender and Product_importance. This technique assigns a unique integer to each category.
- **One-Hot Encoding** was used for nominal features like Warehouse_block and Mode_of_Shipment. This creates separate binary columns for each category, ensuring no artificial ordering is introduced.

These encoding techniques allow the model to interpret categorical data without misrepresenting relationships between categories.



```
...     ID  Customer_care_calls  Customer_rating  Cost_of_the_Product  \
0    1                   4                2                  177
1    2                   4                5                  216
2    3                   2                2                  183
3    4                   3                3                  176
4    5                   2                2                  184

   Prior_purchases  Discount_offered  Weight_in_gms  Reached.on.Time_Y.N  \
0                3                44           1233                    1
1                2                59           3088                    1
2                4                48           3374                    1
3                4                10           1177                    1
4                3                46           2484                    1

   Warehouse_block_B  Warehouse_block_C  Warehouse_block_D  Warehouse_block_F  \
0              False              False               True              False
1              False              False              False               True
2              False              False              False              False
3               True              False              False              False
4              False               True              False              False

   Mode_of_Shipment_Road  Mode_of_Shipment_Ship  Product_importance_low  \
0                  False                  False                    True
1                  False                  False                    True
2                  False                  False                    True
...
1                      False      True
2                      False      True
3                       True      True
4                       True     False
```

3. Normalizing Numerical Features

Numerical features often vary widely in scale. For example, **Cost_of_the_Product** may range from **100 to 300**, while **Customer_rating ranges** from **1 to 5**. To prevent features with larger values from dominating the learning process, we applied **StandardScaler**, which transforms each feature to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model's learning process.

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N	Warehouse_block_B
0	1	-0.047711	-0.700755	-0.690722	-0.372735	1.889983	-1.468240	1	False
1	2	-0.047711	1.421578	0.120746	-1.029424	2.815636	-0.333893	1	False
2	3	-1.799887	-0.700755	-0.565881	0.283954	2.136824	-0.159002	1	False
3	4	-0.923799	0.006689	-0.711529	0.283954	-0.208162	-1.502484	1	True
4	5	-1.799887	-0.700755	-0.545074	-0.372735	2.013404	-0.703244	1	False

Warehouse_block_B	Warehouse_block_C	Warehouse_block_D	Warehouse_block_F	Mode_of_Shipment_Road	Mode_of_Shipment_Ship	Product_importance_low	Product_importance_medium	Gender_M
False	False	True	False	False	False	True	False	False
False	False	False	True	False	False	True	False	True
False	False	False	False	False	False	True	False	True
True	False	False	False	False	False	False	True	True
False	True	False	False	False	False	False	True	False

4. Splitting the Dataset

To evaluate model performance effectively, the dataset was split into training and testing subsets using train_test_split from sklearn.model_selection. We used a **80:20 split ratio** with stratification to maintain the class distribution of the target variable (Reached.on.Time_Y.N) in both sets.

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42, stratify=y
)
```

```
Training set shape: (8799, 17)
Testing set shape: (2200, 17)
```

5. Saving the Split Data

To preserve the processed data for future use and avoid repeating preprocessing steps, all four subsets — x_train, x_test, y_train, and y_test — were saved as separate CSV files using to_csv():

```
# Save the training and testing datasets
X_train.to_csv('X_train.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
X_test.to_csv('X_test.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
```

✓ 0.2s

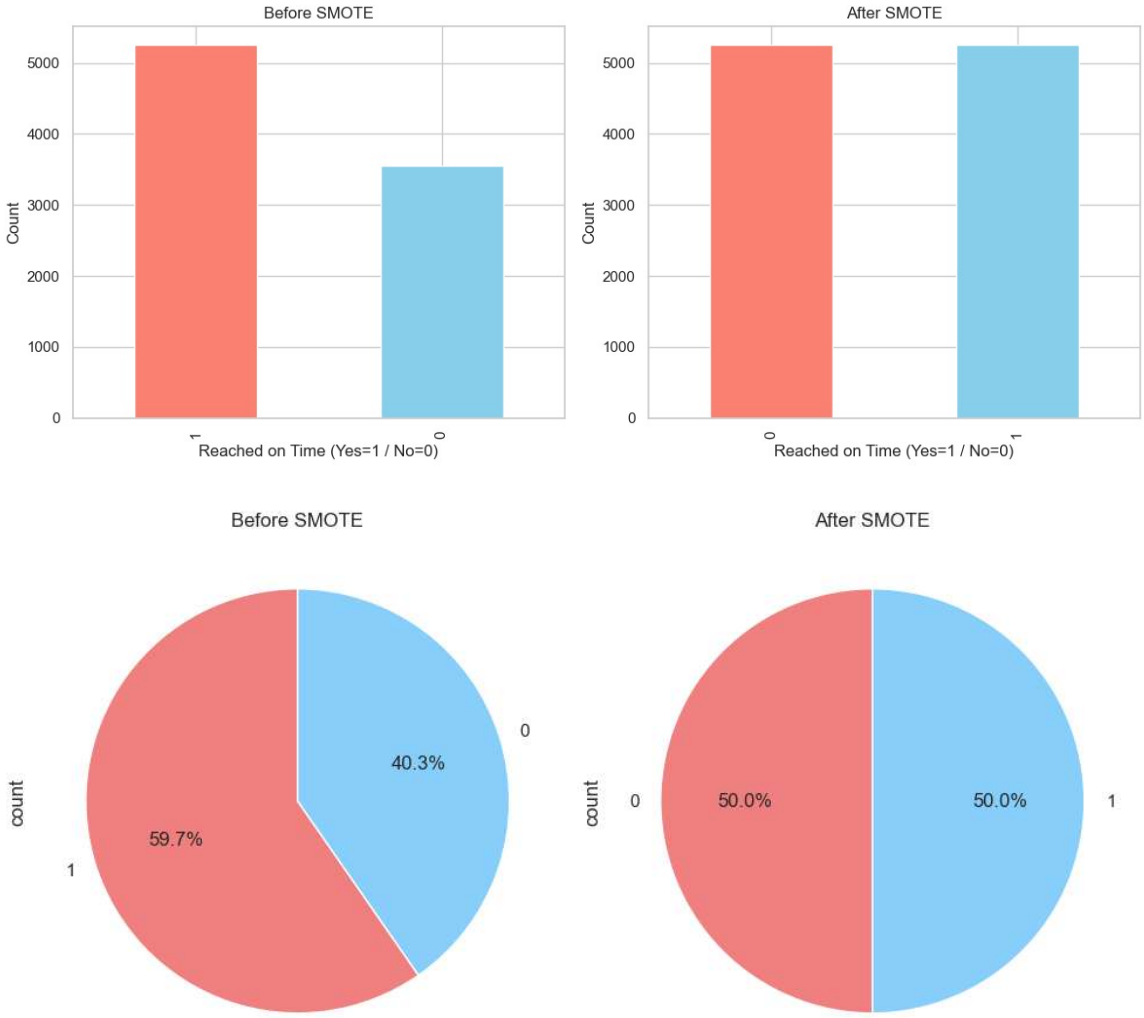
6. Class Imbalance Handling Using SMOTE

Why SMOTE?

In classification tasks, having a balanced target variable is crucial for fair and accurate predictions. In our dataset, the target column Reached.on.Time_Y.N was **imbalanced**, with more

late deliveries (class 1) than on-time deliveries (class 0). This imbalance can cause the model to favour the majority class, leading to poor generalization and biased predictions.

To address this, we used **SMOTE (Synthetic Minority Over-sampling Technique)**, which generates synthetic examples of the minority class to balance the training data.



7. Model Training, Evaluation and Analysis

1. Objective

The goal of this phase is to train multiple classification models on the SMOTE-balanced dataset and evaluate their performance using standard metrics such as accuracy, ROC-AUC score, and classification reports. This helps identify the most effective model for predicting shipment delivery status.

2. Data Preparation

The training and testing datasets were loaded from previously saved CSV files. The training data (X_train_resampled.csv, y_train_resampled.csv) had been balanced using SMOTE to ensure equal representation of both classes. The test data (X_test.csv, y_test.csv) remained untouched to provide an unbiased evaluation.

To ensure compatibility with scikit-learn models, the target variables y_res and y_test were flattened using .ravel().

3. Feature Selection

A subset of relevant features was selected based on prior analysis and domain knowledge. These features included numerical variables (e.g., Discount_offered, Weight_in_gms) and encoded categorical indicators (e.g., Mode_of_Shipment_Ship, Gender_F). This step ensures that only informative and non-redundant features are used for model training.

4. Model Definition

Four classification models were defined with appropriate hyperparameters:

- **XGBoost Classifier**: A gradient boosting model optimized with regularization and histogram-based tree construction.
- **Random Forest Classifier**: An ensemble of decision trees with controlled depth and split criteria.
- **Gradient Boosting Classifier**: A boosting model with tuned learning rate and subsampling.
- **Logistic Regression**: A linear model with regularization, used as a baseline.

Each model was configured with a fixed random seed (random_state=42) to ensure reproducibility.

5. Training and Evaluation Procedure

Each model was trained on the selected features from the SMOTE-balanced training set. For Logistic Regression, feature scaling was applied using StandardScaler due to its sensitivity to feature magnitude. Tree-based models were trained on unscaled data.

After training, predictions were made on both the training and test sets. The following metrics were computed for each model:

- **Training Accuracy**: Measures how well the model fits the training data.
- **Testing Accuracy**: Measures generalization performance on unseen data.
- **ROC-AUC Score**: Evaluates the model's ability to distinguish between classes.

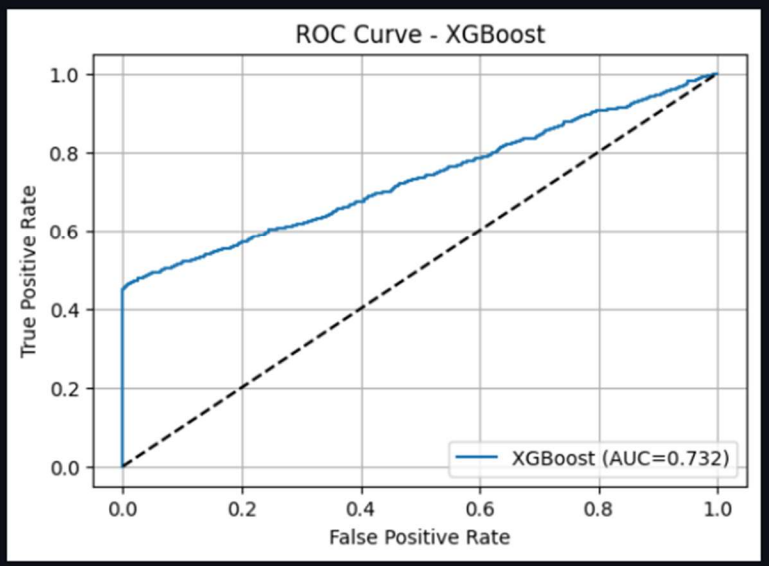
- **Classification Report**: Provides precision, recall, F1-score, and support for each class.

6. ROC Curve Visualization

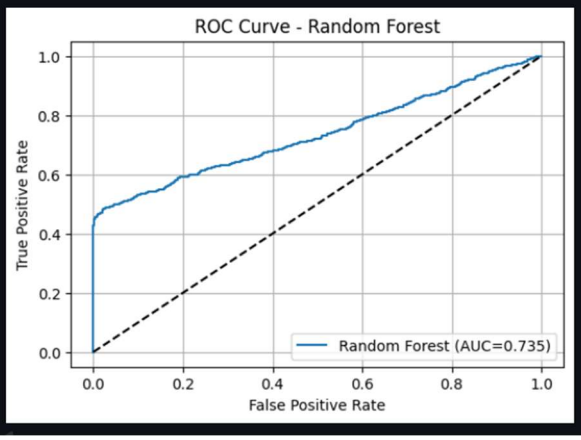
For each model, the Receiver Operating Characteristic (ROC) curve was plotted using the predicted probabilities on the test set. The ROC curve illustrates the trade-off between the true positive rate and false positive rate across different threshold values. A diagonal line was included to represent random classification performance.

The Area Under the Curve (AUC) was annotated in the legend to quantify the model's discriminative power. A higher AUC indicates better performance.

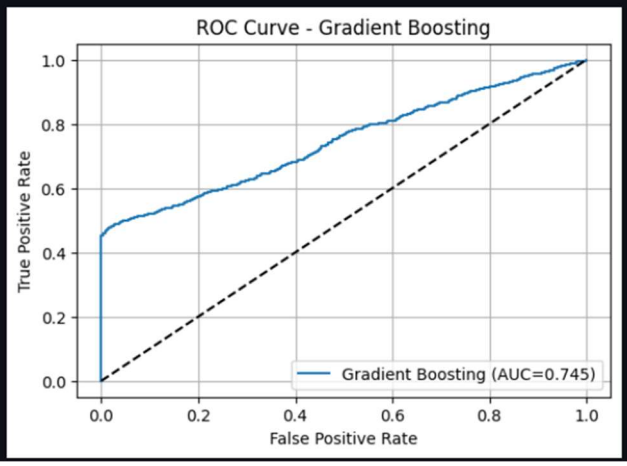
```
================================================================================
🚀 Training XGBoost...
================================================================================
✅ XGBoost Train Accuracy: 0.9372
✅ XGBoost Test Accuracy:  0.6509
✅ XGBoost ROC-AUC Score:  0.7324

📄 XGBoost Classification Report:

              precision    recall  f1-score   support

           0       0.55      0.70      0.62       887
           1       0.75      0.62      0.68      1313

    accuracy                           0.65      2200
   macro avg       0.65      0.66      0.65      2200
weighted avg       0.67      0.65      0.65      2200
```

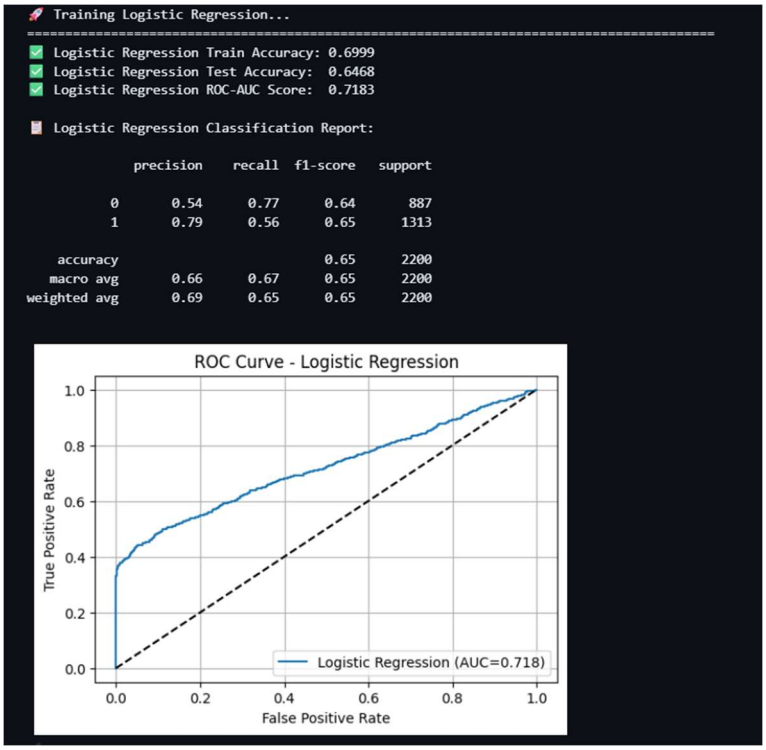


```
================================================================================
🚀 Training Random Forest...
================================================================================
✅ Random Forest Train Accuracy: 0.8449
✅ Random Forest Test Accuracy:  0.6773
✅ Random Forest ROC-AUC Score:  0.7349

📋 Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.56      0.92      0.70       887
           1       0.90      0.52      0.66      1313

    accuracy                           0.68      2200
   macro avg       0.73      0.72      0.68      2200
weighted avg       0.76      0.68      0.67      2200
```




```
================================================================================
🚀 Training Gradient Boosting...
================================================================================
✅ Gradient Boosting Train Accuracy: 0.9173
✅ Gradient Boosting Test Accuracy:  0.6595
✅ Gradient Boosting ROC-AUC Score:  0.7451

📋 Gradient Boosting Classification Report:

              precision    recall  f1-score   support

           0       0.56      0.76      0.64       887
           1       0.78      0.59      0.67      1313

    accuracy                           0.66      2200
   macro avg       0.67      0.68      0.66      2200
weighted avg       0.69      0.66      0.66      2200
```





Model Deployment Using Streamlit:

8. Model Deployment

To make the ShipmentSure prediction system accessible to users, the trained machine learning model was deployed as a **web application** using **Streamlit Cloud**. The deployment enables real-time prediction of shipment delivery status based on user-entered shipment details.

8.1 Deployment Platform: Streamlit Cloud

Streamlit Cloud provides a simple and efficient way to deploy Python-based machine learning applications.
Key advantages:

- No server setup required
- Automatic dependency installation
- Direct integration with GitHub
- Fast and interactive UI
- Free hosting for public apps

8.2 Files Required for Deployment

The following project files were uploaded to GitHub for deployment:

- **app.py** → The Streamlit application script
- **best_model.pkl** → Trained Random Forest classifier
- **train_columns.pkl** → Feature column order used during training
- **requirements.txt** → Dependencies to install on the server

Additionally, dataset files and the notebook were uploaded for documentation and reproducibility.

8.3 Steps Followed for Deployment

Step 1: Preparing `requirements.txt`

A requirements file was created to ensure Streamlit installs all necessary libraries.

Example:

```
streamlit
pandas
numpy
scikit-learn
pickle-mixin
```

Step 2: Uploading Project to GitHub

A new repository was created and all required files were manually uploaded:

```
ShipmentSure/
 ├── app.py
 ├── best_model.pkl
 ├── train_columns.pkl
 ├── requirements.txt
 ├── assets/ (EDA images)
 ├── Train.csv
 └── Jupyter notebook (analysis)
```

Step 3: Configuring Streamlit Cloud

1. Log in at: https://share.streamlit.io
2. Click **Deploy an App**
3. Select GitHub repository
4. Set:

- **Branch:** main
- **Main file:** app.py
- **App URL:** shipmentsure.streamlit.app

Step 4: Fixing File Path Issues

Initially, the model was loaded using a **local file path**:

```
C:\Users\DELL\Downloads\ShipmentSure\best_model.pkl
```

This caused an error on the cloud server.

It was corrected to:

```
model = pickle.load(open("best_model.pkl", "rb")) train_columns =
pickle.load(open("train_columns.pkl", "rb"))
```

Step 5: Successful Deployment

After fixing dependencies and file paths, the application deployed successfully.

Final App Link:
https://shipmentsure.streamlit.app/

8.4 User Interface of the Deployed App

The Streamlit web application provides:

- Numeric input fields
- Dropdown selectors
- Clean and modern UI styling
- A "Predict Delivery Status" button
- Color-coded results (Green → On Time, Red → Late)
- Prediction probability display

8.5 Output of the Deployed Model

The app outputs:

- **Prediction:** On Time / Late
- **Probability Score**
- **Interpretation using color-coded messages**

This allows users to quickly understand the likelihood of delayed shipments based on real-world data.

8.6 Deployment Summary

Step	Description
1	Trained model exported as `.pkl`
2	Project uploaded to GitHub
3	Dependencies added via `requirements.txt`
4	Streamlit configured to point to `app.py`
5	Local paths corrected
6	App successfully deployed

Conclusion:

The *ShipmentSure – Predicting On-Time Delivery Using Supplier Data* project successfully demonstrates the complete end-to-end development of a machine learning system, starting from data exploration and preprocessing to model training, evaluation, and final deployment.

Using a dataset of 10,999 shipment records, the project conducted detailed exploratory data analysis, handled class imbalance, engineered essential features, and prepared high-quality inputs for model training. Multiple algorithms were evaluated, and the **best-performing model** was selected based on accuracy, stability, and overall predictive performance.

A significant achievement of the project is the deployment of the best model as an interactive web application using **Streamlit Cloud**. This allows users to easily input shipment details and instantly obtain predictions regarding the likelihood of on-time or late delivery. The deployed system is user-friendly, visually clear, and practically useful for logistics planning and decision-making.

Overall, this project demonstrates how data-driven insights can improve supply chain reliability and optimize delivery operations. The successful deployment completes the full machine learning pipeline, transforming raw data into a functional and accessible real-world application.

Github repository: https://github.com/Arps09/ShipmentSure-Predicting-On-Time-Delivery-Using-Supplier-Data

Deployement link: https://shipmentsure.streamlit.app/

Submitted by:

Name: Arpita Mishra

Branch: 03