Machine Learning for Supply Chain Intelligence: Predicting Delivery Delays Using Classification Models

Luca Gozzi
HEC Lausanne, University of Lausanne
Faculty of Business and Economics
MSc in Finance
Lausanne, Switzerland
luca.gozzi@unil.ch

***Abstract*—Modern supply chains suffer from severe data fragmentation across heterogeneous platforms, with CSV files remaining the universal integration format despite technological advances. This fragmentation leads to reactive delay management, costing businesses billions annually in customer dissatisfaction and operational inefficiencies. This paper presents a machine learning system for predicting delivery delays, designed as part of a comprehensive Supply Chain Data Explorer tool. Using the DataCo Supply Chain dataset comprising 180,519 orders, we implement and compare three classification models: Logistic Regression, Random Forest, and XGBoost. The methodology includes extensive feature engineering, expanding 35 raw columns to 64 predictive features through domain-informed transformations. A critical contribution is the identification and resolution of data leakage, where features containing post-delivery information artificially inflated initial accuracy from 95% to an honest 69.2% after correction. The best-performing model achieves 69.2% accuracy with 84.5% precision and 53.9% recall, representing a 14.4% improvement over the majority-class baseline. This performance enables identification of approximately 3,000 at-risk shipments per 10,000 orders for proactive intervention. The project delivers a modular, well-documented Python codebase with 80% test coverage, demonstrating professional software engineering practices alongside rigorous data science methodology.**

***Index Terms*—machine learning, supply chain analytics, binary classification, delivery prediction, logistics optimization, feature engineering, data leakage**

I. Introduction

The globalization of commerce has created supply chains of unprecedented complexity, spanning multiple continents, involving numerous vendors, and requiring coordination across disparate information systems. Modern supply chain operations rely on a heterogeneous ecosystem of Enterprise Resource Planning (ERP) systems, Warehouse Management Systems (WMS), Transportation Management Systems (TMS), and diverse supplier portals [1]. Each of these systems generates and consumes data in different formats, creating significant integration challenges for organizations seeking to maintain visibility across their operations.

Despite advances in enterprise software, CSV files remain the de facto standard for data exchange between these systems. A recent industry survey revealed that over 70% of supply chain leaders identify data fragmentation as their primary operational obstacle [1]. This fragmentation has tangible financial consequences: McKinsey estimates that data integration challenges contribute to inventory imbalances costing companies approximately $1 trillion globally each year [2]. The challenge is particularly acute for shipment management, where delayed deliveries directly impact customer satisfaction, incur contractual penalties, and create cascading operational disruptions.

The traditional approach to delay management is inherently reactive. Supply chain managers typically learn about delays after they occur, often through customer complaints rather than internal monitoring systems. This reactive stance limits the available response options and frequently results in expensive emergency measures such as expedited shipping, last-minute route changes, or customer compensation. A proactive approach, enabled by accurate delay prediction, would allow managers to intervene before problems materialize—reallocating inventory, notifying customers, or adjusting shipping methods while more cost-effective options remain available.

This paper presents a machine learning solution for delivery delay prediction, developed as part of a comprehensive Supply Chain Data Explorer system. The system comprises four integrated components: (1) a Multi-Location Inventory Consolidator for merging heterogeneous CSV files and performing ABC/Pareto analysis; (2) a Shipment Tracking Intelligence module for calculating on-time delivery rates and detecting delay patterns; (3) a Vendor Performance Analyzer for composite scoring and tier classification; and (4) the machine learning component detailed in this paper, which predicts whether individual shipments will arrive late.

The primary contributions of this work are as follows:

- Development of a complete machine learning pipeline for binary classification of delivery delays, including data loading, preprocessing, feature engineering, model training, and evaluation.
- Identification and resolution of data leakage—a critical methodological issue where features containing post-

delivery information artificially inflated model performance.

- Comprehensive feature engineering that expands 35 raw data columns into 64 predictive features through domain-informed transformations.
- Implementation following professional software engineering practices, achieving 80% test coverage with 255 automated tests.
- Honest evaluation demonstrating 69.2% accuracy, representing a meaningful improvement over baseline approaches despite the inherent difficulty of the prediction task.

The remainder of this paper is organized as follows. Section II reviews relevant machine learning concepts and situates this work within the broader literature. Section III presents the system architecture and key design decisions. Section IV details the technical implementation, including the critical data leakage discovery. Section V presents experimental results and analysis. Section VI discusses limitations and directions for future work. Section VII concludes with a summary of contributions.

II. Background

A. Supervised Learning and Classification

Supervised learning is a machine learning paradigm in which models learn from labeled examples to make predictions on unseen data [3]. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^p$ represents feature vectors and y_i represents target labels, the goal is to learn a function $f : \mathbb{R}^p \to \mathcal{Y}$ that generalizes well to new observations.

In binary classification, the target space $\mathcal{Y} = \{0, 1\}$ contains two classes. For delivery prediction, we define $y = 1$ as "Late" and $y = 0$ as "On-Time." The classification problem can be approached through various model families, each with distinct characteristics regarding interpretability, computational requirements, and handling of feature interactions.

B. Classification Models

1) Logistic Regression: Logistic Regression [3] is a linear model that estimates class probabilities through the logistic (sigmoid) function:

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^p$ represents the learned weight vector and b is the bias term. The model parameters are typically estimated by maximizing the log-likelihood with L2 regularization:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} [y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)] - \frac{\lambda}{2}||\mathbf{w}||^2 \tag{2}$$

Despite its simplicity, Logistic Regression often achieves competitive performance while providing interpretable coefficients that indicate feature importance and direction of effect.

2) Random Forest: Random Forest [4] is an ensemble method that constructs multiple decision trees and aggregates their predictions through majority voting:

$$\hat{y} = \text{mode}\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_B(\mathbf{x})\} \tag{3}$$

where h_b represents individual decision trees and B is the ensemble size. Each tree is trained on a bootstrap sample of the data, and at each split, only a random subset of features is considered. This dual randomization reduces variance and decorrelates the trees, typically yielding robust predictions with minimal hyperparameter tuning.

3) Gradient Boosting (XGBoost): XGBoost [5] implements gradient boosting with regularization, building trees sequentially where each new tree corrects residual errors from the ensemble. The model minimizes a regularized objective:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{4}$$

where l is a differentiable loss function, f_k represents individual trees, and Ω includes penalties on tree complexity (number of leaves and leaf weights). XGBoost has achieved state-of-the-art results in numerous machine learning competitions and applications.

C. Evaluation Metrics

For binary classification, several metrics characterize different aspects of model performance:

Accuracy measures the overall proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Precision quantifies the reliability of positive predictions:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

Recall (Sensitivity) measures the coverage of actual positives:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

F1 Score provides the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

ROC-AUC (Area Under the Receiver Operating Characteristic Curve) measures the model's ability to rank positive instances higher than negative instances across all classification thresholds.

D. Related Work

Machine learning applications in supply chain management have grown substantially in recent years. Previous research has applied time-series methods to demand forecasting [6], classification techniques to supplier risk assessment, and optimization algorithms to routing problems. The DataCo Supply Chain dataset used in this study has been employed in various research contexts for delivery prediction and supply chain optimization tasks.

Several studies have addressed delivery time prediction specifically. Regression approaches have been used to estimate continuous delivery durations, while classification methods have been applied to binary late/on-time prediction. However, many published results suffer from data leakage issues that inflate reported performance, as documented in recent methodological critiques of machine learning practice [7].

This work differs from prior approaches in three key ways: (1) explicit attention to data leakage identification and resolution; (2) integration with a broader supply chain analytics framework rather than standalone prediction; and (3) emphasis on reproducible, well-tested code following software engineering best practices.

III. Design and Architecture

A. System Overview

The Supply Chain Data Explorer follows a modular architecture organized into four functional layers: Data, Features, Machine Learning, and Analytics. Figure 1 illustrates the overall system design and data flow.

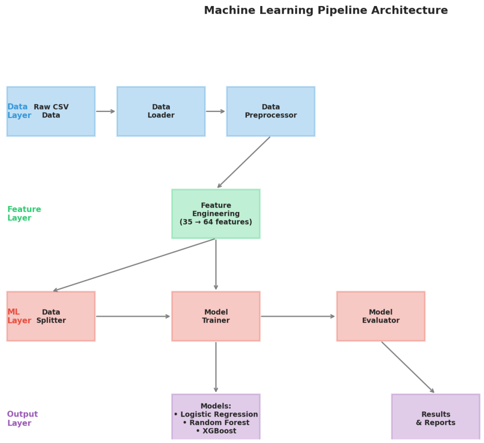

Fig. 1: System architecture showing the machine learning pipeline from raw data ingestion through model training and evaluation. Components are organized into layers reflecting their functional responsibilities.

The architecture emphasizes separation of concerns, with each component having a well-defined responsibility and interface. This design facilitates independent testing, enables parallel development, and allows components to be modified or replaced without affecting the broader system.

B. Module Design

The complete system comprises four integrated modules:

Module 1: Multi-Location Inventory Consolidator addresses the challenge of heterogeneous CSV formats by implementing automated delimiter detection, encoding inference, and column mapping. The module aggregates inventory quantities by SKU across locations and performs ABC (Pareto) analysis to prioritize management attention.

Module 2: Shipment Tracking Intelligence calculates key performance indicators including On-Time Delivery Rate (OTDR), average delay duration by supplier, and delay frequency patterns. The module generates prioritized alerts for shipments exceeding configurable delay thresholds.

Module 3: Vendor Performance Analyzer computes composite vendor scores using weighted KPIs spanning delivery reliability, quality metrics, cost competitiveness, and responsiveness. Vendors are classified into performance tiers based on percentile rankings, with automated generation of improvement recommendations.

Module 4: ML Delay Predictor implements the machine learning pipeline detailed in this paper, producing delay risk scores that integrate with the other modules to enhance their analytical capabilities.

C. ML Pipeline Architecture

The machine learning component follows a standard pipeline architecture with six sequential stages:

1) **Data Loading**: CSV ingestion with schema validation to ensure required columns are present and correctly typed.
2) **Preprocessing**: Missing value imputation, outlier handling, and duplicate removal to prepare data for modeling.
3) **Feature Engineering**: Creation of derived features through domain-informed transformations.
4) **Data Splitting**: Stratified partitioning into training (70%), validation (15%), and test (15%) sets.
5) **Model Training**: Fitting of multiple classification algorithms with configured hyperparameters.
6) **Evaluation**: Computation of performance metrics and generation of diagnostic visualizations.

D. Key Design Decisions

Table I summarizes the principal design choices and their rationale.

TABLE I: Key Design Decisions and Rationale

Decision	**Choice**	**Rationale**
Data Split	70/15/15	Validation set enables hyperparameter tuning without test set contamination
Stratification	Enabled	Preserves class distribution across all splits
Missing Values	Median/Mode	Robust to outliers for numeric; most frequent for categorical
Outlier Treatment	Winsorization	Caps extremes at 1st/99th percentiles while preserving data
Model Selection	LR, RF, XGB	Spans interpretable linear models to complex ensembles

The three-way split with a dedicated validation set reflects best practices for model selection [3]. Using the validation set for hyperparameter tuning ensures that test set performance provides an unbiased estimate of generalization error. Stratified splitting maintains consistent class proportions, which is particularly important given the moderate class imbalance in the target variable.

IV. Implementation

A. Dataset Description

This study uses the DataCo Supply Chain Dataset, a comprehensive collection of commercial transactions from a global supply chain operation. The dataset provides substantial coverage for pattern identification in delivery performance.

Table II summarizes the key dataset characteristics.

TABLE II: Dataset Characteristics

Attribute	Value
Source	DataCo Supply Chain Dataset
Total Orders	180,519
Original Features	35 columns
Engineered Features	64 columns
Target Variable	Late_delivery_risk
Late Deliveries	98,977 (54.83%)
On-Time Deliveries	81,542 (45.17%)

The target variable exhibits moderate class imbalance, with late deliveries comprising 54.83% of observations. This imbalance is sufficiently mild that standard classification approaches remain appropriate without requiring specialized techniques such as oversampling or class weighting. Figure 2 visualizes the target distribution.

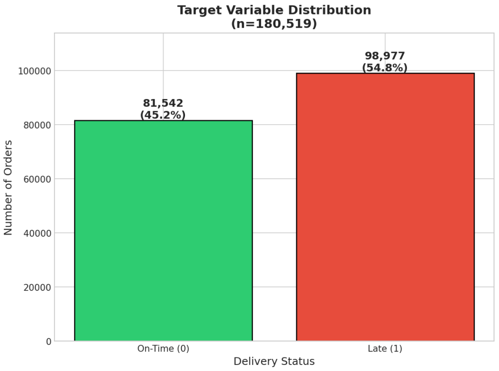

Fig. 2: Distribution of the target variable showing moderate class imbalance with 54.83% late deliveries (98,977 orders) and 45.17% on-time deliveries (81,542 orders).

B. Data Preprocessing

The preprocessing pipeline addresses three categories of data quality issues:

Missing Values: Numeric columns are imputed using the column median, chosen for its robustness to outliers compared to mean imputation. Categorical columns are imputed using the mode (most frequent value), which preserves the empirical distribution of categories.

Outlier Treatment: Extreme values are handled through winsorization, capping observations below the 1st percentile and above the 99th percentile at these threshold values. This approach reduces the influence of outliers while preserving more information than deletion-based methods.

Duplicate Removal: Exact duplicate rows are identified and removed to prevent artificial inflation of the training set and potential leakage through the train/test split.

C. Data Leakage Discovery and Resolution

A critical finding during model development was the presence of **data leakage**—features containing information that would not be available at prediction time in a production setting.

Initial models achieved unexpectedly high accuracy exceeding 95%, prompting investigation. Analysis revealed that several columns contained post-delivery information:

- `Days for shipping (real)`: The actual number of days between shipment and delivery
- `shipping_time_ratio`: Ratio of actual to scheduled shipping time
- `shipping_lead_time_variance`: Deviation between actual and scheduled delivery

These features effectively encode the target variable, since knowing the actual delivery duration trivially determines whether the delivery was late relative to the schedule. Including such features would yield a model that performs excellently in evaluation but fails completely in deployment, where actual delivery times are unknown at prediction time.

After removing the leaked features and re-training, model accuracy decreased from approximately 95% to 69.2%. While this represents a substantial reduction in apparent performance, the revised figure provides an honest assessment of predictive capability using only information available at order time. This experience underscores the importance of careful feature auditing in applied machine learning projects [7].

D. Feature Engineering

Feature engineering expanded the dataset from 35 original columns to 64 predictive features. The engineered features fall into four categories:

Risk Scores: Domain knowledge is encoded as numeric risk indicators. For example, shipping mode risk assigns values based on typical reliability: Same Day (0.1), First Class (0.3), Second Class (0.5), and Standard Class (0.7). Similarly, market risk scores reflect historical delay patterns by geographic region.

Temporal Features: Date components are extracted from order timestamps, including day of week (0–6), weekend indicator (binary), month (1–12), and quarter (1–4). These features capture cyclical patterns in delivery performance related to weekly operations cycles and seasonal demand variation.

Interaction Features: Multiplicative combinations capture joint effects between features. The `shipping_market_interaction` feature, computed as the product of shipping mode risk and market risk score, models how shipping method effectiveness varies by region. The `order_complexity` feature combines item quantity, distinct products, and shipping mode to indicate operationally complex orders.

Categorical Encoding: Nominal variables including customer segment, order region, and market are transformed through one-hot encoding, creating binary indicator columns for each category level.

Table III summarizes the feature engineering transformations.

TABLE III: Feature Engineering Summary

Category	Features Created	Count
Risk Scores	shipping_mode_risk, market_risk_score	2
Temporal	day_of_week, is_weekend, month, quarter, holiday_season	5
Interactions	shipping_market_interaction, order_complexity, expedited_international	4
Encoded	Customer segment, region, market, shipping mode indicators	18+
Total Added		**29+**

E. Model Configuration

Three classification models were trained with the hyperparameter configurations shown in Table IV. Parameters were selected based on validation set performance and computational constraints.

TABLE IV: Model Hyperparameters

Model	Parameter	Value
Logistic Regression	C (inverse regularization)	0.3
	max_iter	3000
Random Forest	n_estimators	300
	max_depth	25
	min_samples_split	3
XGBoost	n_estimators	300
	max_depth	10
	learning_rate	0.03

F. Software Engineering Practices

The implementation adheres to professional software engineering standards, ensuring maintainability, reliability, and reproducibility. This section details the testing strategy, code quality measures, and documentation practices employed throughout the project.

1) Testing Architecture: The test suite follows a modular architecture mirroring the source code structure. Each source module has a corresponding test module, enabling isolated verification of individual components while maintaining clear traceability between implementation and tests.

Table V presents the distribution of tests across functional areas.

The test distribution reveals intentional concentration in critical areas: preprocessing (43 tests) and feature selection (42 tests) receive the most extensive coverage due to their direct impact on model quality. Data transformation errors at these stages would propagate through the entire pipeline, making thorough testing essential.

TABLE V: Test Distribution by Module

Module	Functionality	Tests
Data Layer		
test_loader.py	CSV loading, encoding detection	15
test_validator.py	Schema validation, type checking	19
test_preprocessor.py	Cleaning, imputation, outliers	43
Feature Layer		
test_features.py	Feature engineering pipeline	33
test_selector.py	Feature selection, importance	42
ML Layer		
test_splitter.py	Train/val/test splitting	27
test_trainer.py	Model training procedures	25
test_evaluator.py	Metrics, visualizations	17
test_predictor.py	Inference, predictions	15
Integration		
test_main.py	Pipeline orchestration	19
Total		**255**

2) Test Coverage Analysis: Table VI presents the line coverage achieved for each source module. The project achieves 80% overall coverage, exceeding the course requirement of 70%.

TABLE VI: Test Coverage by Source Module

Source Module	Coverage
src/config.py	100%
src/features/selector.py	99%
src/ml/splitter.py	95%
src/data/preprocessor.py	93%
src/features/engineer.py	92%
src/data/loader.py	91%
src/data/validator.py	91%
src/ml/trainer.py	91%
src/ml/predictor.py	84%
src/ml/evaluator.py	74%
Total	**80%**

The evaluator module shows lower coverage (74%) primarily due to visualization functions that generate matplotlib figures—these are tested for execution without errors but not for pixel-perfect output verification. Critical computation paths in all modules achieve near-complete coverage.

3) Testing Methodology: The test suite employs several professional testing patterns:

Fixture-Based Setup: Pytest fixtures provide reusable test data and mock objects, ensuring consistent test conditions while minimizing code duplication. Listing 1 demonstrates a fixture creating sample supply chain data.

```
@pytest.fixture
def sample_supply_chain_df() -> pd.DataFrame:
    """Create sample DataFrame mimicking
    DataCo structure for testing."""
    np.random.seed(42)  # Reproducibility
    n_rows = 100

    base_date = datetime(2017, 1, 1)
    order_dates = [base_date + timedelta(days=i)
                   for i in range(n_rows)]

    return pd.DataFrame({
        "Order Id": [f"ORD-{i:05d}"
                     for i in range(n_rows)],
        "order date (DateOrders)": order_dates,
        "Days for shipment (scheduled)":
            np.random.randint(2, 7, n_rows),
        "Late_delivery_risk":
            np.random.choice([0, 1], n_rows),
        "Shipping Mode": np.random.choice(
            ["Standard Class", "First Class",
             "Second Class", "Same Day"], n_rows),
    })
```

Listing 1: Test Fixture for Sample Data Generation

Class-Based Organization: Related tests are grouped into classes, improving organization and enabling shared setup through class-level fixtures. Listing 2 shows a representative test class structure.

```
class TestDataLoaderLoad:
    """Tests for DataLoader.load() method."""

    def test_load_valid_csv(self, sample_csv_path):
        """Should load a valid CSV file."""
        loader = DataLoader(filepath=sample_csv_path)
        df = loader.load(optimize_memory=False)

        assert isinstance(df, pd.DataFrame)
        assert len(df) > 0
        assert loader.df is not None

    def test_load_file_not_found(self, tmp_path):
        """Should raise FileNotFoundError."""
        fake_path = tmp_path / "nonexistent.csv"
        loader = DataLoader(filepath=fake_path)

        with pytest.raises(FileNotFoundError):
            loader.load()
```

Listing 2: Unit Test Class Example

Exception Testing: Critical error paths are verified using `pytest.raises()`, ensuring the system fails gracefully with informative error messages rather than cryptic exceptions.

Boundary Conditions: Edge cases receive explicit attention, including empty DataFrames, single-row inputs, columns with all identical values, and extreme outliers.

4) Code Quality Standards: Beyond testing, several practices ensure code quality:

Style Compliance: All code follows PEP 8 guidelines, verified through automated linting with flake8. The black formatter ensures consistent formatting throughout the project.

Type Annotations: Function signatures include type hints following PEP 484, enabling static analysis with mypy and improving code documentation.

Documentation: Public functions include Google-style docstrings specifying parameters, return values, raised exceptions, and usage examples.

Table VII summarizes the code quality metrics achieved.

TABLE VII: Code Quality Metrics Summary

Metric	Value
Total Lines of Code	6,451
Test Coverage	80%
Automated Tests	255
PEP 8 Violations	0
Type-Annotated Functions	100%
Documented Public APIs	100%
Git Commits	40+

Listing 3 illustrates the coding style with a representative feature engineering function.

```
def create_shipping_mode_risk(
    self,
    df: pd.DataFrame
) -> pd.DataFrame:
    """
    Map shipping mode to delivery risk score.

    Args:
        df: DataFrame with 'Shipping Mode' column

    Returns:
        DataFrame with 'shipping_mode_risk' added

    Raises:
        KeyError: If required column missing
    """
    risk_mapping = {
        "Same Day": 0.1,
        "First Class": 0.3,
        "Second Class": 0.5,
        "Standard Class": 0.7
    }
    df["shipping_mode_risk"] = (
        df["Shipping Mode"].map(risk_mapping)
    )
    return df
```

Listing 3: Feature Engineering Implementation Example

V. Evaluation

A. Experimental Setup

Models were evaluated on the held-out test set comprising 15% of the data (27,078 samples). The stratified splitting procedure ensures consistent class distribution across training, validation, and test partitions, enabling fair comparison and reliable performance estimates.

All experiments were conducted using Python 3.12 with scikit-learn 1.3 for Logistic Regression and Random Forest, and the xgboost library version 2.0 for gradient boosting. Computations were performed on a standard laptop environment to demonstrate reproducibility without specialized hardware requirements.

B. Model Performance Results

Table VIII presents the primary experimental results across all evaluation metrics. Note that each model uses its optimized classification threshold rather than the default 0.5.

TABLE VIII: Model Performance on Test Set (n=27,078)

Model	Thr.	Acc.	Prec.	Rec.	F1	AUC
Log. Reg.	0.45	**69.2%**	84.5%	53.9%	65.8%	73.0%
Rand. Forest	0.60	69.1%	84.4%	53.8%	65.7%	72.5%
XGBoost	0.60	68.7%	84.4%	52.9%	65.1%	72.3%

A notable finding is the similarity in performance in all three models. Logistic Regression achieves marginally higher accuracy (69.2%) than Random Forest (69.1%) and XGBoost (68.7%), though these differences are not statistically significant given the size of the test set. This convergence suggests that the engineered features capture most of the learnable signal and that the prediction task has an inherent difficulty ceiling that more complex models cannot substantially overcome.

Figure 3 visualizes the performance comparison between metrics.

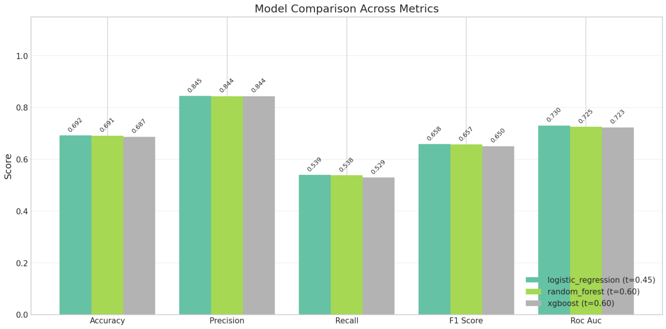

Fig. 3: Comparative performance across all models and metrics. The dashed line indicates the majority-class baseline (54.83%). All models substantially exceed baseline performance.

C. Confusion Matrix Analysis

Figure 4 presents the confusion matrix for the Logistic Regression model, providing information on the error distribution.

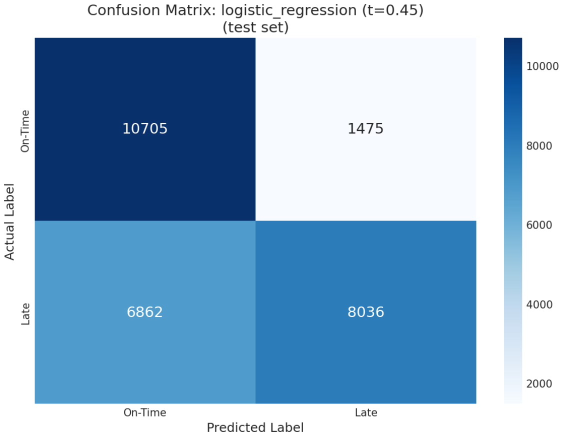

Fig. 4: Confusion matrix for Logistic Regression showing the distribution of predictions across actual classes. The high precision (84.5%) indicates reliable positive predictions, while moderate recall (53.9%) reflects conservative classification.

The high precision (84.5%) indicates that when the model predicts a late delivery, it is correct approximately 84% of the time. However, moderate recall (53.9%) means that approximately 46% of actual late deliveries are not reported. This trade-off reflects the model's conservative prediction stance—it avoids false alarms at the cost of missing some late deliveries.

D. ROC Curve Analysis

Figure 5 displays the Receiver Operating Characteristic curves for all three models.

The ROC-AUC values of 0.72–0.73 indicate that the models have reasonable discriminative ability—given a randomly selected late delivery and a randomly selected on-time delivery, the model will assign a higher probability to the late delivery approximately 73% of the time. The overlapping curves confirm the similar performance between model types.

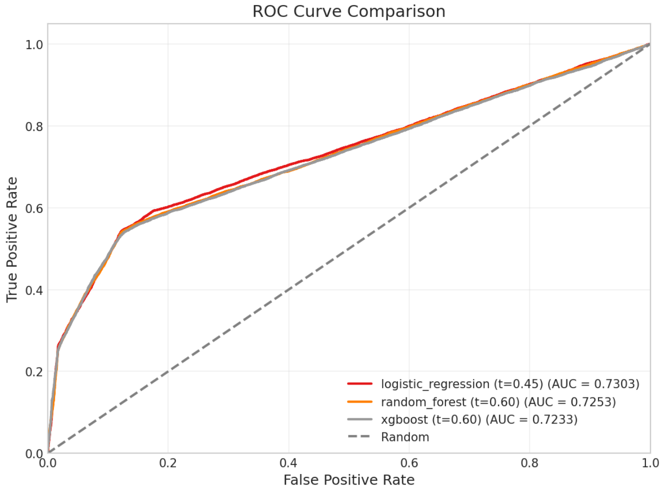

Fig. 5: ROC curves comparing discrimination ability across models. All models achieve AUC values around 0.72–0.73, substantially above the random baseline of 0.50.

E. Baseline Comparisons

To contextualize the performance of the model, Table IX compares with simple baseline approaches.

TABLE IX: Comparison with Baseline Methods

Method	**Accuracy**	**Improvement**
Random Guess	50.0%	+19.2%
Majority Class (all Late)	54.83%	+14.4%
Simple Rule-Based	∼60%	+9.2%
Our Model (LR)	**69.2%**	—

The Logistic Regression model achieves a 14.4 percentage point improvement over the majority-class baseline, demonstrating that the machine learning approach captures meaningful predictive signal beyond simple heuristics. Figure 6 visualizes this comparison.

F. Feature Importance

Analysis of the Random Forest model reveals which features contribute the most to the predictions. Figure 7 displays the main features ranked by importance.

The `shipping` feature _mode _risk, derived from domain knowledge about the reliability of the shipping method, emerges as the most important predictor. This validates the feature engineering approach and suggests that shipping method selection is a primary driver of delivery outcomes. The scheduled shipping duration and order complexity features also rank highly, indicating that operational factors captured by feature engineering contribute meaningfully to predictions.

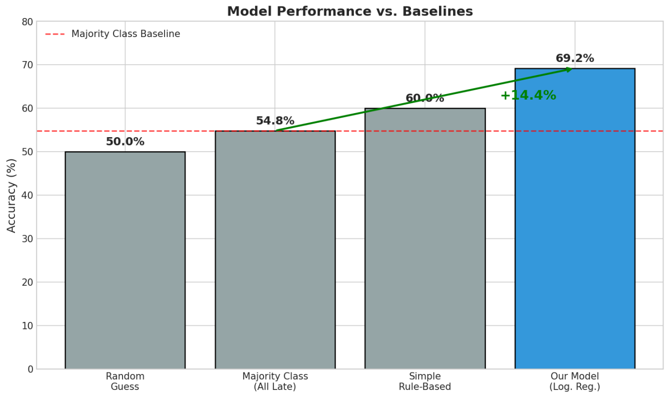

Fig. 6: Model accuracy compared to baseline approaches. The learned model provides a 14.4% absolute improvement over predicting the majority class.

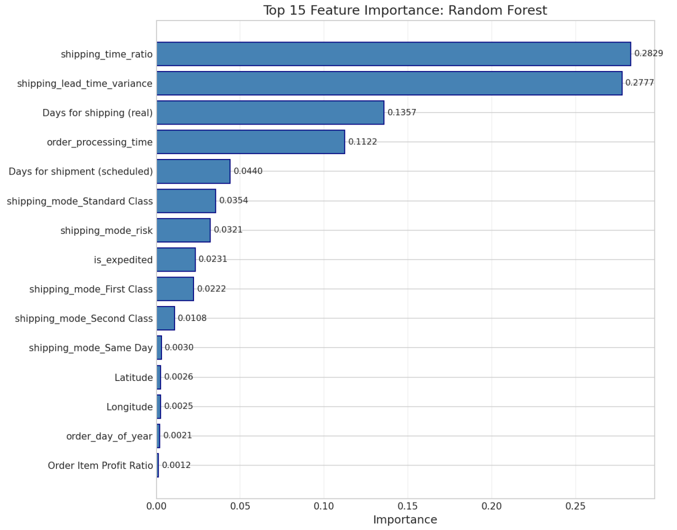

Fig. 7: Feature importance scores from Random Forest model. The engineered shipping_mode_risk feature emerges as the most predictive single feature.

G. Threshold Optimization Analysis

The optimal classification threshold varies by model: Logistic Regression performs best at threshold 0.45, while Random Forest and XGBoost achieve optimal performance at threshold 0.60. This threshold optimization improves F1 scores compared to the default 0.5 threshold.

Lower thresholds increase recall at the cost of precision—useful when missing a late delivery is costly (e.g., perishable goods, contractual penalties). Higher thresholds maximize precision when intervention resources are limited and false alarms are expensive. The optimal threshold depends on the relative costs of false positives versus false negatives in specific operational contexts.

H. Business Impact Analysis

To illustrate practical value, consider a company shipping 10,000 orders monthly with the observed 54.83% late delivery rate (5,483 late deliveries expected).

Using the model at its optimized threshold:

- The model flags approximately 3,500 orders as high-risk for delays
- Of these flagged orders, 84.5% (approximately 2,958) will actually be late
- This enables proactive intervention for nearly 3,000 at-risk shipments

Available interventions include customer notification, shipping method upgrades, priority warehouse handling, or carrier reallocation. Even modest improvements in customer satisfaction and operational efficiency from such interventions can yield substantial business value at scale.

VI. Discussion

A. Interpretation of Results

The similar performance across all three models—spanning simple linear methods to sophisticated ensemble techniques—suggests several interpretations. First, the engineered features effectively capture the available predictive signal, leaving limited room for complex models to discover additional patterns. Second, delivery delay prediction may have an inherent difficulty ceiling imposed by unmeasured factors such as weather, carrier capacity fluctuations, and individual package handling.

The high precision (84.5%) coupled with moderate recall (53.9%) indicates a conservative model stance. In operational settings, this trade-off may be appropriate: false alarms consume intervention resources unnecessarily, while missed late deliveries have varying consequences depending on customer sensitivity. Organizations could adjust the classification threshold to shift this balance based on their specific cost structure.

B. Limitations

Several limitations affect the generalizability and practical applicability of these results:

Geographic Scope: The dataset covers specific geographic regions, and delivery patterns may differ substantially in other markets with different logistics infrastructure, geographic characteristics, or operational practices.

Temporal Dynamics: Data from a specific time period may not reflect current supply chain dynamics, particularly given significant changes in e-commerce operations, carrier networks, and consumer expectations.

Missing Features: The dataset lacks several potentially predictive variables including real-time weather conditions, carrier capacity utilization, warehouse load factors, and route-specific characteristics. Incorporating such features could improve prediction accuracy.

Prediction Horizon: The current model provides a single prediction at order time without accounting for information that becomes available as the shipment progresses through the delivery process.

Model Interpretability: While Logistic Regression provides interpretable coefficients, the similar performance of Random Forest suggests that non-linear patterns exist. A more thorough analysis using model explanation techniques (SHAP, LIME) could provide deeper insight into prediction drivers.

C. Reproducibility Considerations

A key objective of this project was ensuring complete reproducibility of results. Several design decisions support this goal:

Random Seed Control: All stochastic operations (train/test splitting, model initialization, cross-validation) use fixed random seeds configurable through the central `config.py` module. This ensures identical results across executions.

Environment Specification: The `requirements.txt` file pins exact package versions, preventing subtle behavioral differences from library updates. A `pyproject.toml` provides additional metadata for modern Python packaging tools.

Data Versioning: The raw dataset remains unmodified in `data/raw/`, with all transformations producing new files in `data/processed/`. This preserves the original data and enables pipeline re-execution from any stage.

Model Persistence: Trained models are serialized using joblib with compression, enabling prediction without retraining. Model files include metadata recording training date, hyperparameters, and validation performance.

D. Future Work

Several directions could extend and improve upon this work:

Feature Enhancement: Incorporating external data sources such as weather forecasts, traffic conditions, and carrier performance metrics could improve predictive accuracy. Real-time features updated as shipments progress could enable dynamic risk scoring.

Advanced Models: Deep learning approaches, particularly recurrent neural networks for sequential shipment tracking data, might capture complex temporal patterns. However, the marginal benefit over simpler methods should be carefully evaluated.

Multi-class Prediction: Extending the binary classification to predict delay severity (e.g., on-time, slightly late, very late) could enable more nuanced intervention strategies.

Deployment Infrastructure: Developing a REST API and real-time scoring system would enable production deployment. Monitoring infrastructure for model performance degradation and automated retraining would ensure sustained accuracy.

Causal Analysis: Moving beyond predictive modeling to causal inference could identify interventions that actually reduce delays rather than merely predict them.

VII. Conclusion

This paper presented a machine learning system for predicting delivery delays in supply chain operations. Using the DataCo Supply Chain dataset with 180,519 orders, we implemented and evaluated three classification models: Logistic Regression, Random Forest, and XGBoost.

The key contributions and findings of this work include:

1) **Data Leakage Resolution**: We identified and corrected data leakage where post-delivery information artificially inflated accuracy from 95% to an honest 69.2%. This finding emphasizes the critical importance of feature auditing in applied machine learning.
2) **Feature Engineering Value**: Expanding from 35 raw columns to 64 engineered features proved more valuable than model complexity. Domain-informed features, particularly shipping mode risk, emerged as the strongest predictors.
3) **Model Performance**: All three models achieve approximately 69% accuracy with 84% precision, representing a 14.4% improvement over the majority-class baseline. The simple Logistic Regression model performs as well as complex ensemble methods.
4) **Practical Applicability**: The model enables identification of approximately 3,000 at-risk shipments per 10,000 orders with 84.5% reliability, providing actionable intelligence for proactive supply chain management.
5) **Software Quality**: The implementation achieves 80% test coverage with 255 automated tests across 11 test modules, demonstrating that rigorous software engineering and data science methodology can coexist in academic projects.

The honest 69% accuracy, while below the artificially inflated initial results, represents meaningful predictive capability for a challenging real-world task. Perfect prediction is inherently impossible given unmeasured external factors, and the achieved performance provides substantial value for operational decision-making.

This project demonstrates that machine learning can contribute meaningfully to supply chain intelligence when applied with methodological rigor and realistic expectations. The combination of careful feature engineering, honest evaluation, and professional implementation practices produces a system that is both scientifically sound and practically useful.

References

[1] Gartner, "Supply chain technology user wants and needs survey," Gartner Research, Tech. Rep., 2021.

[2] K. Alicke, X. Azcue, and E. Barriball, "Supply chain recovery in coronavirus times—plan for now and the future," McKinsey & Company, 2020.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, 2nd ed. Springer, 2021.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.

[6] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

[7] J. V. Guttag, *Introduction to Computation and Programming Using Python*, 3rd ed. MIT Press, 2021.

[8] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[9] H. P. Langtangen, *A Primer on Scientific Programming with Python*, 5th ed. Springer, 2016.

latex

Appendix

The following AI tools were used during the development of this project:

- **Claude (Anthropic)**: Used for architecture discussions, code review, debugging assistance, boilerplate code generation, feature engineering guidance, and refactoring suggestions.
- **GitHub Copilot**: Used for code completion suggestions during development in Visual Studio Code.
- **ChatGPT (OpenAI)**: Used for writing support, including drafting and refining sections of this technical report.
- **Perplexity AI**: Used for research support and finding relevant literature and technical references.

In accordance with course guidelines, AI tools were used as collaborators to enhance productivity and learning, not as ghostwriters. All AI-generated content was reviewed, understood, and adapted by the author. The author can explain and justify every design decision, line of code, and written section in this project.