UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Optimization of the replenishment order deliveries at the Febelco warehouses

Racheal Natumanya

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
Prof. dr. Inneke VAN NIEUWENHUYSE

SUPERVISOR :
Niels DUQUET

2024
2025

Transnational University Limburg is a unique collaboration of two universities in two countries: the University of Hasselt and Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Optimization of the replenishment order deliveries at the Febelco warehouses

Racheal Natumanya

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
Prof. dr. Inneke VAN NIEUWENHUYSE

SUPERVISOR :
Niels DUQUET

Acknowledgements

First, I express my sincere gratitude to my supervisor, Prof. Dr. Inneke Van Nieuwenhuyse, for their invaluable guidance, encouragement, and critical feedback throughout the development of this thesis. Their expertise and support have been instrumental in shaping the direction and depth of this work.

I also thank Febelco for providing access to the data and operational context that made this study possible. Special thanks to Niels Duquet and Ann-Sofie Decaigny for their collaboration and insight into supply chain processes.

A heartfelt thanks goes to my family and friends, whose unwavering encouragement, patience, and belief in me have sustained me through the challenges of this work. Your support has been my foundation.

Finally, I am grateful to all those who contributed, directly or indirectly, to the completion of this thesis. Your participation, no matter how small, is greatly appreciated.

Abstract

In pharmaceutical supply chains, inaccurate estimate of lead time can lead to stock shortages, overstocking, and service-level failures, risks that are particularly critical in health-sensitive environments. Febelco, Belgium's largest pharmaceutical wholesaler, faces such challenges due to the variability in both external supplier delivery and internal warehouse processing times. Traditional planning methods, which often rely on static average lead times, fall short in capturing the operational complexity and uncertainty inherent in real-world logistics.

This study pursues two main objectives: (1) to perform an analysis of Febelco's historical data to uncover patterns and inefficiencies in historical lead-time behavior and (2) to develop predictive models that estimate both supplier delivery lead time and warehouse cover time using machine learning. The goal is to improve the accuracy of the planning while also identifying areas of uncertainty and risk that affect the warehouse planning.

The analysis of the data revealed a substantial deviation between theoretical expectations and actual outcomes. Although most orders were delivered within 5–10 working days of the total leadtime, outliers extended beyond 20 days for suppliers and more than 10 days for warehouse processing, underscoring the limitations of assumptions based on average. These findings justified the shift to data-driven predictive approaches.

Tree-based machine learning models—Random Forest, XGBoost, and LightGBM—were developed using historical order-level features such as supplier ID, order volume, urgency indicators, and order timing. Among these, Random Forest delivered the most balanced and interpretable performance. For the supplier lead time model, it achieved a Root Mean Square Error (RMSE) of 6.410, a mean absolute error (MAE) of 1.963 working days and a mean absolute percentage error (MAPE) of 20. 068%. For the warehouse lead time model, performance was stronger, with an RMSE of 0.972, MAE of 0.204 working days, and MAPE of 8.529% , reflecting greater consistency of internal operations.

This study combines machine learning and uncertainty quantification to improve lead time prediction in pharmaceutical supply chains. Using Random Forest models, we found that the accuracy and uncertainty of the prediction vary significantly between suppliers. Some suppliers exhibited high uncertainty, and this could be due to inconsistent delivery patterns, while others showed stable, well-predicted behavior. In particular, there were also some suppliers for whom fluctuations in prediction levels were observed according to the different suppliers.

Contents

1 Introduction

In an increasingly interconnected and dynamic global market, the effectiveness of supply chain operations plays a vital role in determining the competitiveness and long-term viability of an organization. To meet the demands of today's volatile business environment, supply chains must strike a balance between cost effectiveness and flexibility to adapt to changing market conditions and consumer needs. A key barrier to achieving such adaptability is the reliability of deliveries, with particular emphasis on the management of lead times, the interval between when a purchase order is made and when those items are available for sale or use[1]. This duration may include multiple stages such as order processing, supplier production or picking, transportation, customs handling, warehouse receiving, and internal operations like inspection or put-away. The ability to forecast and control lead times accurately is essential to ensure operational continuity, minimize excess inventory, and meet customer expectations [2].

Even minor fluctuations in delivery performance can impact the supply chain, disrupt production schedules, cause inventory shortages or surpluses, and degrade service quality. As companies increasingly adopt lean inventory strategies and Just-In-Time (JIT) systems, the consequences of lead-time variability become even more pronounced[1]. These challenges are amplified in the pharmaceutical sector, where the stakes of disruption are especially high. Pharmaceutical supply chains are marked by strict regulatory requirements, product shelf-life constraints, and the critical nature of demand. In this setting, stockouts can affect supply, while overstocking can lead to product expiration and financial loss. Ensuring that medicines and medical supplies arrive consistently and on time is essential not only for maintaining service levels but also for safeguarding public health. Pharmaceutical wholesalers, as intermediaries between manufacturers and pharmacies or hospitals, bear the operational burden of managing inbound variability while ensuring timely deliveries. As service expectations increase and demand patterns shift, the reliability of replenishment lead times becomes central to operational performance.

Lead-time variability occurs when the actual time taken fluctuates from the expected or average duration. This unpredictability makes inventory management more complex and

increases the risk of overstocking or stock shortages. Therefore, accurate forecasting and understanding of its patterns are essential to reduce rational disruptions and improve warehouse planning. According to [3], fluctuations in delivery performance erode supply chain agility and increase operational costs by requiring emergency orders or maintaining excess safety stock. This highlights the importance of improving supply chain visibility and adopting proactive supplier management practices to counteract these inefficiencies. Supply chains operate under rigid Just-In-Time frameworks that are particularly susceptible to disruptions caused by lead-time uncertainty. Even small deviations in delivery timing can adversely affect inventory planning and diminish service level performance.

Warehouses play a pivotal role in absorbing supply-side variability and ensuring that customer orders are fulfilled accurately and on time. Warehouse operations encompass functions such as receiving, reserve storage, picking, sorting, and dispatch [4]. Among these, the receiving function is especially vulnerable to unpredictable lead times. If deliveries arrive earlier or later than planned, downstream operations, such as take-away, replenishment, and order picking, are disrupted. Irregular delivery flows can overwhelm warehouse labor on certain days and leave it underutilized on others. Early arrivals may also cause storage congestion and disrupt scheduled receiving plans, while late deliveries risk product unavailability and missed dispatch windows.

These challenges are particularly pronounced in the case of Febelco, Belgium's leading pharmaceutical wholesaler. Operating a nationwide distribution network comprising eight warehouses and supplying more than 2,500 pharmacies and hospitals, the company holds an estimated 43% share of the country's pharmaceutical wholesale market, making it the largest holder of market share [5]. To support continuous product availability, Febelco follows a structured replenishment strategy, placing biweekly or monthly orders with a diverse range of suppliers.

However, despite the structured nature of this ordering system, variability in lead times persists as a major operational issue. Deliveries often deviate from scheduled timelines, arriving prematurely or with significant delays, creating unpredictability in inbound flows. This inconsistency complicates labor scheduling, inventory management, and warehouse receiving activities, ultimately affecting the company's ability to consistently meet the high

service level demands of the healthcare sector. The problem addressed in this research is the unpredictability of lead times in Febelco's warehouses. This unpredictability negatively affects warehouse operations, inventory control, and order fulfillment capabilities. Although the company adheres to fixed reorder schedules, it lacks a predictive mechanism to anticipate when stock will actually arrive and be available for sale.

To address these challenges, this study focuses on analyzing and predicting lead time variability within Febelco's warehouse operations. The specific objectives of the study are as follows.

- Analyze historical lead time data across different supplier-item combinations to identify patterns and anomalies.
- Develop a predictive model to estimate the time between order placement and stock availability, accounting for both delivery performance and internal processing times.

The insights generated by this research are intended to support more accurate inventory planning, improve warehouse efficiency, and enhance service levels within the pharmaceutical supply chain.

2 Methodology

2.1 Data Description and Preprocessing

The dataset used for this research originates from Febelco's procurement and warehouse operations and contains detailed historical records of purchase orders and product receptions for a period of one year (from January 2024 to January 2025). It comprises 37 columns and 45,476 rows which capture various dimensions of inbound logistics, including supplier information, product details, order quantities, and delivery timelines.

As evident from Table 9, the dataset includes operational timestamps, product metadata, order information, and process-related flags used for prioritization and planning. Each reception is modeled as a technical delivery event, uniquely identified by a reception number. The header-level fields describe attributes that apply to the entire delivery, such as the reception creation date, delivery timestamps, warehouse identification, and flags like urgency(products given higher priority on reception) or processing times(from the point of delivery of products up to when they are available for sale) at the warehouse. Line-level data, on the other hand, correspond to individual item types within a reception. It includes fields such as product identifiers, order quantities, scanned quantities, delivery lead times, storage conditions, and flags for product categorization (e.g., refrigerated, narcotic, or quota products). No missing or invalid values were present in the dataset, eliminating the need for data cleaning or imputation steps. All variables are consistent and within expected ranges. The timestamps were used in their original format, and time intervals;order to delivered time(supplier lead time) and delivered to product available for sale time (cover time) were computed directly from these raw dates. Since the data covers only a single warehouse, no warehouse-specific indicator features are used.

A key variable in our analysis is the lead time of an order, which we define as the total duration from placing a replenishment order to the point where the items are available as stock in the warehouse. For clarity, this total lead time is conceptually decomposed into two sequential stages: (1) Supplier Lead Time – the time from order placement until delivery at the warehouse, and (2) Cover Time – the time from delivery at the warehouse until the goods are available in the inventory and ready to be sold. This decomposition aligns with

how operational delays can occur in different segments of the supply chain. Separating lead time into these components provided additional insight into where delays might occur (external supplier delays vs. internal processing delays).

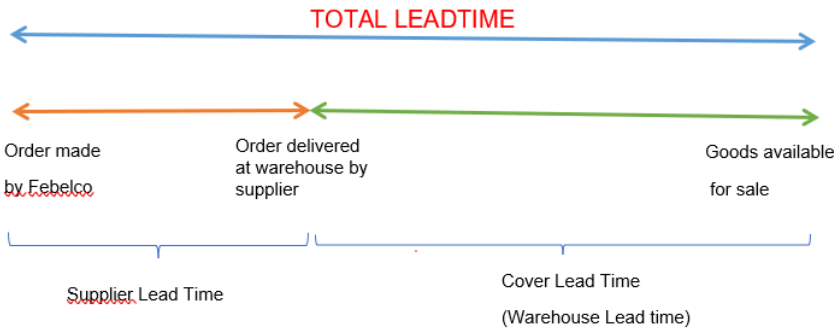

Figure 1: Lead Time Components: From Order Placement to Warehouse Availability

2.2 Predictive Modelling Approach

To accurately predict lead times at different stages of the supply chain—specifically supplier lead time (order to delivery) and warehouse cover time (delivery to availability), we employ three tree-based ensemble regression models: Random Forest, Extreme Gradient Boosting (XGBoost), and LightGBM with the use of Python libraries. These models are well-suited for lead time prediction in a multilayer supply chain because they can model complex interactions between features and have demonstrated superior predictive performance in similar contexts [6].Tree-based ensemble methods are widely used in predictive analytics due to their high accuracy and relative interpretability; they can map nonlinear associations that simpler linear models miss, an important capability given that supply chain lead times depend on numerous interrelated factors (supplier behavior, logistics, demand fluctuations).

The implementation of these models utilized several well-established Python libraries such as Scikit-learn and other libraries that are tailored for XGBoost and LightGBM models. To fine-tune model performance, Optuna was employed as the hyperparameter optimization framework. Optuna uses Bayesian optimization with a Tree-structured Parzen Estimator

to efficiently search the hyperparameter space[7]. For each model, an objective function was defined to minimize the Root Mean Squared Error (RMSE) using 5-fold cross-validation. A total of 100 optimization trials were conducted per model, targeting key hyperparameters such as the number of estimators, maximum tree depth, learning rate, and others. This approach enabled systematic and efficient tuning, resulting in improved predictive accuracy and model robustness.

2.2.1 Modelling Approaches

(i) **Random Forest Regression**

Random Forest (RF) is an ensemble learning method that constructs a "forest" of decision trees using bootstrap samples and random feature selection, then averages their predictions for regression tasks [6]. This bagging approach improves generalization by reducing variance, enabling Random Forests to handle large datasets and high-dimensional feature spaces without severe overfitting, and to model complex nonlinear relationships in both classification and regression problems. These properties make Random Forests particularly suitable for lead time forecasting, where outcomes are influenced by multiple interacting features. According to [8] Random Forest models were applied to predict lead times as part of an inventory optimization framework, demonstrating the model's effectiveness in improving operational decision-making in supply chain contexts.

(ii) **LightGBM (Light Gradient Boosting Machine)**

LightGBM is a gradient boosting framework that, like XGBoost, builds an ensemble of decision trees in sequence. However, LightGBM was specifically engineered to be highly efficient and scalable on large, high-dimensional data sets [9]. LightGBM employs Gradient-Based One-Side Sampling to retain informative instances with large gradients while down-sampling those with small gradients, improving the accuracy of information gain estimation for splits. It also uses Exclusive Feature Bundling to combine sparse, mutually exclusive features, thereby reducing the number of features and accelerating tree-building. From a supply chain perspective, such efficiency is valuable: lead time prediction models often must be retrained or updated as new data arrives, and LightGBM can handle this rapidly, even with millions of records or dozens of features. We include LightGBM not only for

its computational benefits but also for its proven effectiveness in supply chain risk and lead time prediction [9][10]. Given these advantages, LightGBM is an appropriate choice to model lead times, as it can quickly learn complex patterns from large supply chain datasets and potentially yield more accurate predictions without prohibitive computational cost. Like other tree-based models, it also provides feature importance metrics, helping analysts interpret which factors contribute most to lead time variability. By leveraging LightGBM alongside Random Forest and XGBoost, we aim to harness the strengths of advanced ensemble methods for lead time prediction.

(iii) **XGBoost (Extreme Gradient Boosting)**

In contrast to the Random Forest parallel ensemble approach, XGBoost builds trees sequentially: each new tree corrects the residual errors of the previous group using a gradient descent optimization process. By iteratively "boosting" weak learners, XGBoost builds trees sequentially, with each tree correcting the mistakes of its predecessor. Specifically, it employs gradient descent optimization to reduce a chosen loss function, such as mean squared error, by fitting new trees to the residuals of previous ones. In the words of an AnalyticsVidhya guide, the algorithm 'minimizes a loss function by adding weak learners using gradient descent', thus refining the ensemble stage by stage [11]. We selected XGBoost for our lead time prediction task to leverage these strengths: It can capture complex non-linear interactions among supply chain characteristics (e.g., interactions between order size and product flag) more finely than a single-stage model [12]. While XGBoost models are somewhat less interpretable than a standalone decision tree, they still allow for extraction of feature importance and partial dependence plots, giving analysts insights into which factors most strongly influence lead time predictions. Their consistent performance in related forecasting scenarios and the ability to reduce prediction error through iterative refinement make them a compelling choice for improving lead time predictions in a complex system [13].

2.2.2 Model Evaluation Metrics

To evaluate the accuracy of lead time predictions, we employ three error metrics: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square

Error (RMSE). These metrics are widely used in the forecasting literature, including supply chain demand and lead time studies[8] to assess model performance. Each metric captures a different aspect of prediction error, and together they provide a comprehensive view of forecast quality. In the context of supply chain forecasts, it is common to report multiple error measures to ensure robust evaluation of models. The formulas for the chosen metrics are given below, where y_i is the actual lead time, $\hat{y}_i$ is the predicted lead time, and n is the number of predictions.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

MAE is the average of the absolute errors between predicted and actual values. Measures the typical magnitude of prediction errors in the same units as the target variable (in this case, working days of lead time). MAE is straightforward to interpret: it tells us, on average, How many working days do the forecasted lead times deviate from the actual values. One advantage of MAE is that it treats all errors equally (linearly), making it less sensitive to outliers than squared error metrics. For example, an MAE of 2 days means that, on average, the model's lead time predictions differ from actual outcomes by 2 days. MAE is a standard metric for evaluating prediction models. By minimizing MAE, we aim to ensure that the model performs well in terms of overall day-to-day prediction accuracy. MAPE represents the mean absolute error as a percentage of the actual values. It is scale-independent, allowing us to gauge the error relative to the size of the actual lead time. This is particularly meaningful for practitioners: a 5% forecasting error has intuitive significance regardless of whether the lead time is 10 days or 100 days. MAPE is widely used in supply chain forecasting studies as a key performance indicator for accuracy. Many organizations and researchers prefer MAPE because it directly answers the question, 'On average, what is the percentage error present in our predictions?' In our context, a low MAPE indicates that the predicted lead times are very close to the actual lead times in relative terms. However, MAPE can be sensitive when the actual values y_i are very small (since it involves

$|y_i - \hat{y}_i|/y_i$). In general, MAPE provides an intuitive measure of forecast precision in relative terms, complementing absolute error metrics. A smaller MAPE implies higher predictive precision, which is crucial for demand planning and inventory control decisions in the supply chain.

The root mean square error (RMSE) is a widely used metric to evaluate predictive accuracy, particularly in regression problems. RMSE measures the average magnitude of the prediction errors, capturing how closely the predictions match the observed data. Specifically, RMSE calculates the square root of the average of squared differences between the actual observed values and their corresponding predicted values.

The squaring of errors emphasizes larger discrepancies, making RMSE particularly sensitive to substantial errors compared to metrics like Mean Absolute Error (MAE). This characteristic makes RMSE especially suitable for situations where large prediction errors are particularly undesirable and should be heavily penalized. For example, an RMSE value can be interpreted as the typical magnitude by which the predicted values deviate from the actual values on average, measured in the same units as the data.

RMSE is widely regarded as the most appropriate metric when the goal is to penalize large prediction errors more severely. Unlike MAE or MAPE, which treat all deviations equally or proportionally, RMSE squares each error term before averaging. This mathematical property makes it more sensitive to larger deviations, which are typically the most disruptive in real-world supply chain contexts[14].Additionally, RMSE retains the same unit as the predicted variable (in this case, lead time), which allows for more intuitive interpretation by planners and stakeholders. It is highly sensitive to outliers, which can disproportionately inflate its value, potentially misrepresenting the performance of the model. Furthermore, RMSE does not provide an indication of the direction of errors, which means that it does not differentiate between under-predictions and over-predictions. Given its intuitive nature and practical utility, RMSE is commonly applied in various fields, including forecasting, econometrics, machine learning, and environmental modeling[15]. It provides a straightforward measure of predictive accuracy, allowing researchers and practitioners to evaluate and refine their predictive models effectively. This explains the need to compare it with other metrics.

2.2.3 Feature Engineering

The final dataset used for both exploratory data analysis and predictive modeling is composed of engineered variables derived from raw inbound delivery records. These variables capture key aspects of operational performance, such as lead times, quantity discrepancies, urgency flags, and product-specific classifications. The data set integrates temporal and categorical features, enabling a comprehensive understanding of factors that influence inventory flow and delivery effectiveness.

Table 1: Summary of Variables per purchase order in the dataset

Column	**Data Type**	**Description**
ReceptionNbr	Integer	Reception identifier
DimProductKey	Categorical	Product identifier
OrderNbr	Integer	Purchase order number
DimVendorKey	Categorical	Vendor identifier
OrderDateTime	Integer	Order date and time
DeliveredDateTime	Integer	Timestamp on which the products are delivered
DelInStockDateTime	Integer	Timestamp at which the products are made available for sale
DelProcStartDateTime	Integer	Processing start timestamp at the warehouse
OrderQuantity	Integer	Quantity ordered
ReceptionQuantity	Integer	Quantity received
Quantity difference	Integer	Difference between ordered and received quantity
TheoreticalLeadTime	Integer	Theoretical/agreed delivery lead time
Order to available wkdays	Integer	Working days between order placement and availability of products for sale
Order to delivered wkdays	Integer	Working days from order placement to products delivered at warehouse
Cover time	Integer	Working days between order delivery and availability of products for sale
Delivered to processing wkdays	Integer	Working days from delivery time to start of processing at the warehouse
Processing to available wkdays	Integer	Working days from processing to products availability at the warehouse
UrgentPreReceptionFlg	Binary	Urgency flag at reception level
UrgentLineFlg	Binary	Urgency flag at line level
Refrigerated	Binary	Cold chain flag
Narcotic	Binary	Narcotics classification
Productcategory	Categorical	Product category
QuotaProduct	Binary	Limited supply product flag

Feature engineering is performed to enhance the predictive capabilities of the lead time models by transforming raw operational data into meaningful inputs. Several time-based attributes were derived from the original timestamp fields. From the order timestamps, features such as order hour, day, day of the week, month, and year were extracted to capture temporal dynamics and potential seasonality in order patterns. Similarly, delivery timestamps were used to derive the hour and weekday of delivery, which can influence delivery timelines due to operational cycles.

Multiple columns containing text representations of binary attributes—such as refrigerated, narcotic, or quota-restricted—were standardized and converted into binary format. This conversion accounts for a wide range of inconsistent string formats, including abbreviations, mixed-case entries, and missing values. Additionally, the product category column was processed to distinguish between critical pharmaceutical categories and others, contributing further to the classification process.

To model interactions between operational conditions, several composite features were created. These include the product of order quantity and theoretical lead time, interactions between urgency flags and refrigeration requirements, and combinations of urgency and quantity. Such interaction terms were introduced to allow the models to learn from compound operational factors that may jointly influence lead times. Categorical keys representing suppliers were encoded numerically using one hot encoding, enabling them to be used effectively with tree-based models without implying any ordinal relationships.

Regarding outlier handling, no data points were removed from the dataset. Although the dataset exhibited some extreme values, the decision was made to preserve the data's integrity and reflect real-world operational variability. Overall, the feature engineering process incorporated temporal, categorical, and interaction-based transformations rooted in both domain knowledge and statistical insight. This approach was essential in providing the machine learning models with a rich and informative feature set capable of capturing complex real-world behaviors.

2.2.4 Hyperparameter Tuning

To find optimal predictive accuracy, hyperparameter tuning was performed for all machine learning models using Optuna, an automatic hyperparameter optimization framework based

on Bayesian optimization [16]. The tuning process employed the Tree-structured Parzen Estimator (TPE) as the search algorithm to efficiently navigate the high-dimensional hyperparameter space. For each model—Random Forest, XGBoost, and LightGBM—100 optimization trials were conducted to identify configurations that minimized the Root Mean Squared Error (RMSE) during 5-fold cross-validation.Key hyperparameters tuned included the number of estimators, maximum depth, learning rate, subsample ratios, and regularization parameters. The use of Optuna allowed for dynamic trial pruning, which significantly reduced computational overhead by terminating underperforming trials early. This efficiency is particularly valuable in industrial-scale supply chain datasets, where training and validation can be resource-intensive.

Integrating Optuna into the modeling pipeline contributed directly to performance improvements, particularly for boosting models where tuning has a pronounced impact on convergence and generalization. The automated tuning framework enhanced model robustness and mitigated the risk of manual bias in hyperparameter selection, aligning with best practices in machine learning-based supply chain applications [17].Each model was optimized separately for both the supplier and warehouse lead time prediction tasks. The objective function used in tuning was the minimization of the Root Mean Squared Error (RMSE) via k-fold cross-validation (with k=5) on the training set. For Random Forest, parameters such as `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_features`, and `bootstrap` were tuned. LightGBM and XGBoost involved additional control over regularization (`reg_alpha`, `reg_lambda`), learning rate, number of leaves or maximum depth, and sampling ratios (`colsample_bytree`, `subsample`).

2.2.5 Uncertainity Estimation

Uncertainty estimation plays a critical role in machine learning, particularly in domains where decision making depends not only on predictive accuracy but also on the reliability of predictions. In the context of Random Forest models, uncertainty is typically quantified by examining the variation in predictions produced by individual trees within the ensemble. Random forests aggregate the outputs of multiple decision trees trained on bootstrapped data samples, offering a natural mechanism to capture uncertainty[18]. In regression problems, the standard deviation of the predictions across trees can be used to estimate the

degree of uncertainty associated with a given prediction. In classification, uncertainty can be derived from the distribution of class probabilities, often measured through entropy or confidence in the predicted class. Two key forms of uncertainty are relevant here: aleatoric uncertainty, which reflects noise inherent in the data (e.g., randomness in outcomes), and epistemic uncertainty, which arises due to limited knowledge or data scarcity. Random forests are particularly well-suited to capturing epistemic uncertainty, since their ensemble nature inherently reflects disagreement among learners, a useful proxy for where the model lacks confidence[19]. Deep learning models often require Bayesian approximations or ensemble techniques to capture uncertainty, Random Forests offer a more computationally accessible path to the same goal. Incorporating uncertainty helps move beyond simple point predictions, offering a richer understanding of where models perform reliably and where caution is warranted [20]. This is especially valuable in operations and planning, where inaccurate predictions can lead to costly errors. Random forests strike a balance between predictive power and interpretability, making them a practical choice for scenarios where understanding both the outcome and its reliability is essential

3 Results

3.1 Exploratory Data Analysis

The analysis focuses on the two main components of Febelco's replenishment order lead time, shown in Figure 10. In the Slim4 inventory system, each component has a target value – a theoretical lead time for supplier delivery (specific to each product type) and a default processing time (3 working days) for the warehouse, which together define the expected coverage period. These target values directly influence the calculations of safety stock: a longer actual coverage period requires more safety stock to maintain the desired cycle service level. The central question in this exploratory analysis is whether these targets are met in reality and how the actual lead times vary between different combinations of supplier – item.

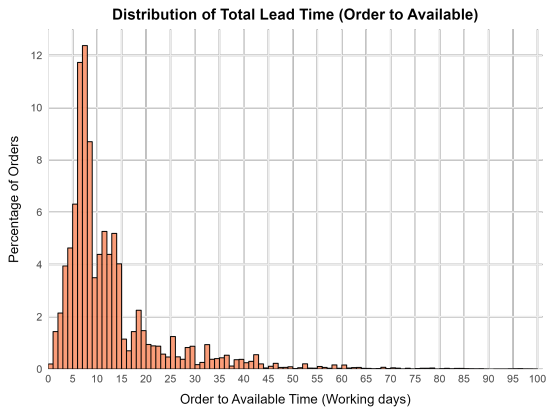

Figure 2: Distribution of Total Lead Time

Percentile	Working Days
5%	3.00
25%	5.00
50%	7.00
75%	10.00
95%	18.00
100%	60.00

Table 2: Empirical percentiles of Total Lead Time

Figure 2 shows the distribution of total lead time in working days, measured from the time an order is made to the time when the products are available for sale.The distribution is asymmetric, with a clear concentration of orders between 5 and 10 working days, peaking near day 7. Beyond 15 working days, the frequency of orders steadily declines, though a small number of orders take considerably longer, extending to more than 60 working days.This pattern indicates that some of the orders are fulfilled promptly, with experiencing longer fulfillment durations. These extended lead times may be associated with specific operational constraints or external factors.

Table 2 summarizes key empirical percentiles that provide further insight into the distribution of lead times. The 5th percentile indicates that 5% of orders are fulfilled within 3 working days, while the 25th percentile corresponds to a lead time of 5 days. These percentile values highlight that the majority of orders are processed efficiently within a 10-day period, although a small portion experience significantly longer fulfillment times.

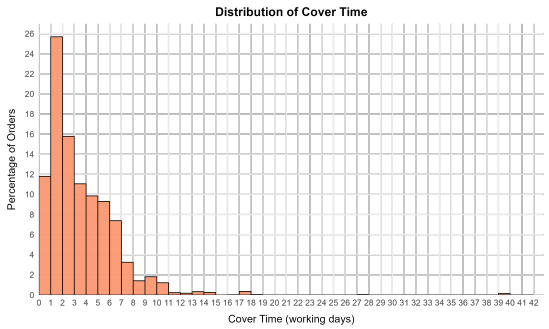

Figure 3: Distribution of Cover Time

Percentile	Working Days
5%	2.00
25%	4.00
50%	6.00
75%	8.00
95%	12.00
100%	21.00

Table 3: Empirical percentiles of Cover Time

Figure 3 displays the distribution of cover time in working days, defined as the number of working days between when a product is delivered and when it becomes available. The histogram shows the percentage of orders that fall into each one-day interval, across a range of 0 to 42 days. The majority of the cover times fall within a relatively narrow window, with the highest concentration of orders occurring between 4 and 8 working days, indicating that a good number of the products are processed within the 3-day planned period. There is an observed gradual decline in frequency beyond 10 days, and only a few instances extend to the 40-day period. This pattern suggests that most deliveries are made available for use within a predictable time frame, although there are occasional longer delays that may be influenced by storage, inspection, or administrative procedures. According to 3, only a small proportion of orders, fewer than 10% are made available within 3 working days. This is evident from the 5th percentile (2 days) and the 25th percentile (4 days). Since the 25th percentile exceeds the planned timeframe, this implies that at least 75% of orders do not meet the 3-day target. The median cover time is 6 days, implying that half of all orders take twice as long as the planned standard. This substantial deviation from the planned benchmark suggests a gap between operational expectations and actual

system performance. The above observations provide a clear, data-driven assessment of this misalignment. Addressing the causes of delay—whether they stem from internal processes, resource constraints, or external dependencies, could help improve compliance with the planned 3-day window and improve overall supply chain responsiveness.

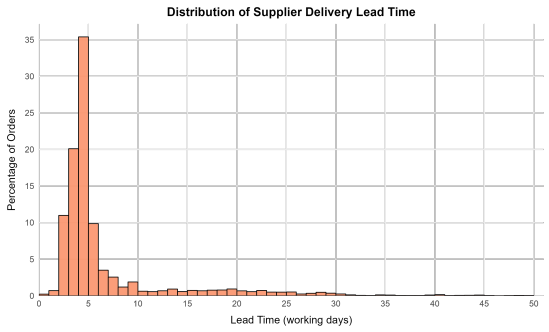

Figure 4: Distribution of Supplier Delivery Lead Time

Percentile	Working Days
5%	3
25%	4
50%	5
75%	6
95%	23
100%	132

Table 4: Empirical Percentiles of Supplier Delivery Lead Time

Figure 4 illustrates the distribution of supplier delivery lead time, calculated in working days. The histogram shows the percentage of orders across one-day intervals. Most deliveries are made in a short time frame, with a clear concentration of orders around 4 to 6 working days, reaching a peak of 5 days, where more than 35% of all orders are fulfilled. The distribution also exhibits a long rightward extension, indicating the presence of outliers. These less frequent but significantly longer lead times stretch beyond 20 days. Table **??**ab:supplier$_leadtime_percentilesshowsthatatleasthalfofallordersaredeliveredin5workingdays.The75thperc$
$quartersofdeliveriesoccurwithinafairlynarrowrange(between3and6days).However,the95thpercentilejump$

Figure 5 compares two key dimensions of supplier delivery performance: the median percentage of late orders (x-axis) and the percentage of late orders (y-axis). The median percentage of late delivery refers to the typical severity of the delay caused by late deliveries by a supplier, expressed as a percentage of the original delivery time. For example, a value of 50% means that the delivery occurred 50% later than expected. The percentage of late orders, on the other hand, indicates how often a supplier fails to meet the expected delivery date. Each point in the plot represents a supplier. Suppliers positioned in the upper left area tend to deliver a high proportion of orders late (24.662.5665), but delays

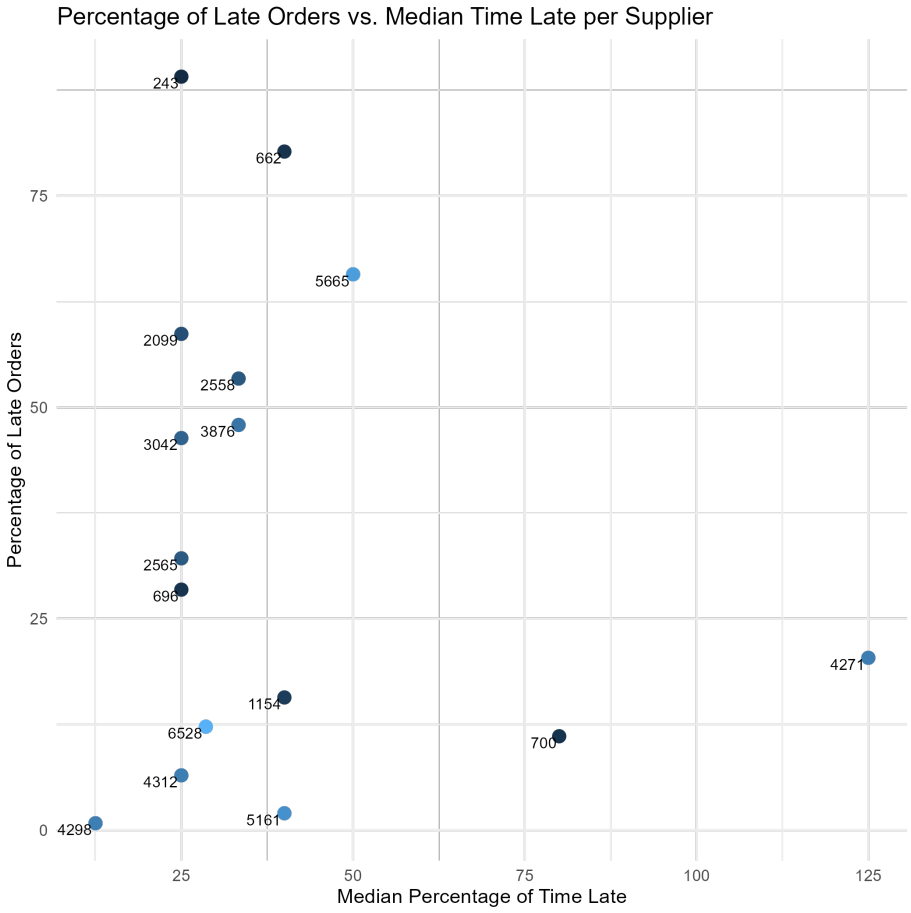

Figure 5: Late orders by supplier

are generally modest in duration. Suppliers such as 700 and 427 stand out due to their extreme median lateness values, above 75%, despite having a lower proportion of late deliveries. This suggests that while they are not frequently late, the delays they do incur are substantial and potentially disruptive. Meanwhile, suppliers located in the lower left quadrant (4296, 4312) demonstrate both a low frequency of lateness and relatively small delays, indicating consistently reliable performance. Overall, this graph provides a nuanced view of supplier reliability, highlighting the importance of considering not just how often deliveries are late but also how late they are.

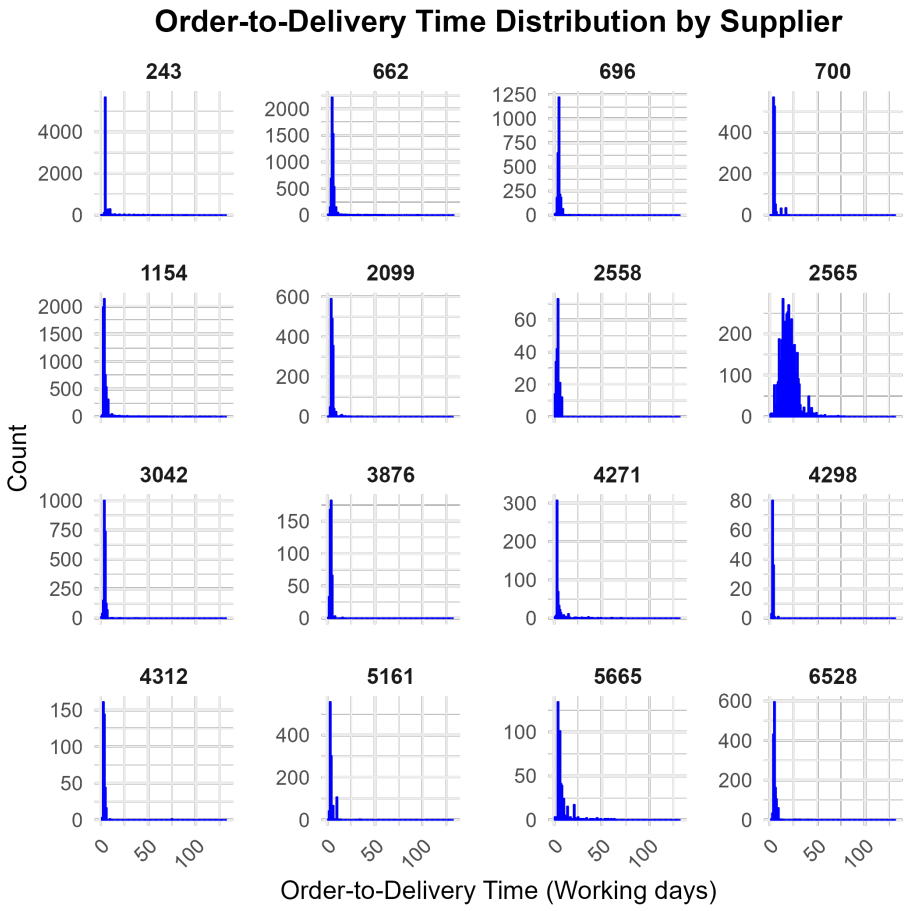

Figure 6: Order to delivery time distribution by supplier

Figure 6 displays the distribution of order-to-delivery times in working days, for individual suppliers. Majority of suppliers demonstrate a highly concentrated delivery pattern, with most orders being fulfilled in fewer than 10 working days. This is evident from the sharp peaks near the lower end of the x-axis in most panels for suppliers 243, 1154, 3042, and 6528. These distributions suggest consistent and predictable delivery performance. In contrast, some suppliers, like 2565 and 5665, show a broader spread of delivery times, indicating greater variability and a longer average lead time. Suppliers— 5665,2565 and 6528 show some outliers, with delivery times. Although these long delays are rare, they could represent risks to supply continuity if not managed proactively. Compared side-by-side, this graph provides a clear and granular view of supplier performance, enabling the identification of high-performing and potentially problematic vendors .

Supplier Delivery Lead Time vs. Theoretical Expectations

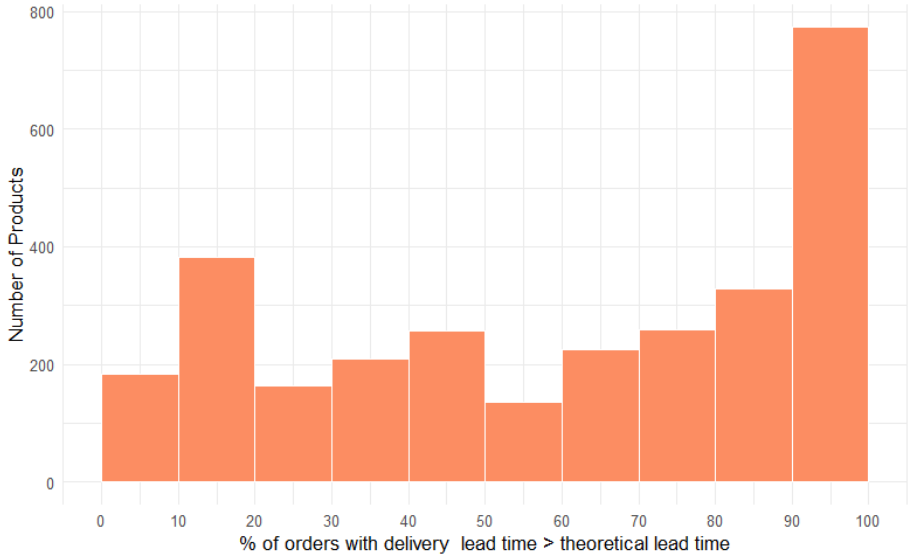

Figure 7: Distribution of Products by Frequency of Late Supplier Deliveries (Exceeding Theoretical Lead Time)

Figure 7 compares the supplier delivery lead time and the theoretical lead time,only about 20% of product types (773 out of 3690 products) have no late deliveries at all, consistently meeting their theoretical lead time on every order.These punctual items were excluded from the plot, allowing us to focus on the remaining 80% of products that experienced at least occasional delays.A significant cluster of products are late on nearly every order: the histogram shows a pronounced peak in the 90–100% bin, indicating there are numerous products(almost 800 products) for which almost every delivery arrives later than promised. This finding clearly indicates that the theoretical lead times recorded in Slim4 are overly optimistic for most products. In most cases, suppliers do not adhere to the contracted delivery lead times, resulting in actual delivery times that regularly exceed the targets.

Warehouse Cover Time vs. Standard 3-Day Target

Figure 8 shows the internal processing interval, from the receipt of the goods to when they are available for sale, is evaluated against a standard target of 3 working days. A similar pattern of widespread target violation emerged: only about 15% of the product types (558

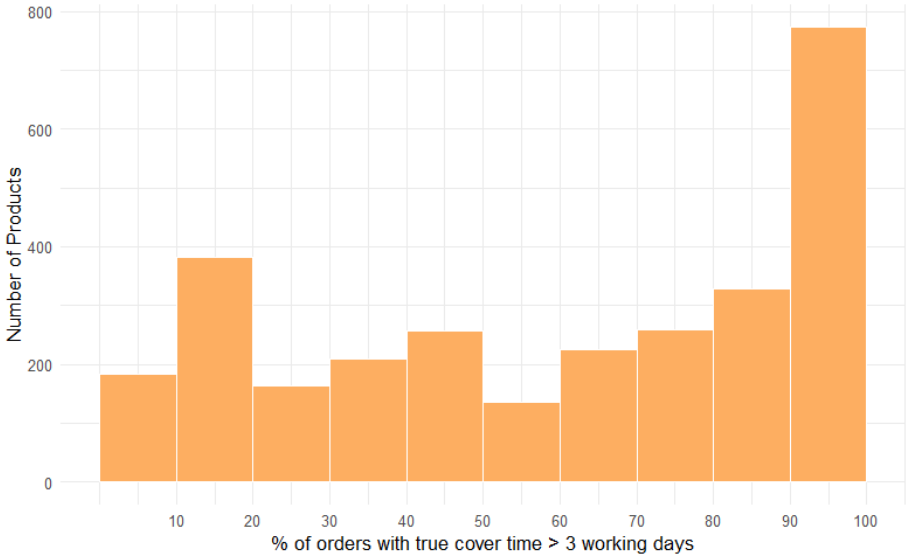

Figure 8: Probability that the real cover time exceeds the standard 3 days

products of 3690 products) had all their orders processed within the 3-day standard, which means they never exceeded the target. These items that were consistently on time were removed from the histogram and further analysis focused on those that did not meet the target.

Among the remaining 85% products, many show substantial probabilities of delay in the warehouse stage. The plot indicates that a large number of product types exceed the 3-day target in more than 30% . For many items, delays in internal handling are not rare exceptions, but rather a frequent occurrence. Some product types exceed the 3-day cover time for almost every order they undergo. This means that almost all deliveries of these products take longer than the supposed 3-day processing period once in the warehouse. The implication is again clear: the fixed 3-day cover-time assumption in the system is not realistic for a large portion of the product types.

Total Lead Time vs. Expected Coverage Period

Combining the supplier and warehouse stages, the total coverage period (from order placement to goods sellable) against the sum of the theoretical lead time plus 3 days is evaluated as shown in Figure 9. This gives a full view of whether the overall replenishment process

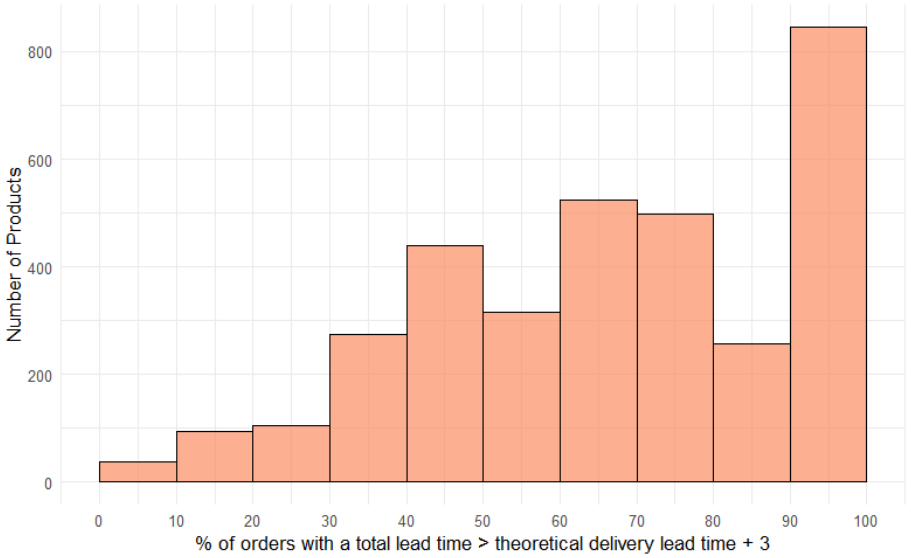

Figure 9: Probability that the true total lead time exceeds (theoretical delivery lead time + 3 days)

meets the expected timeline. The results show an even more pronounced deviation from the plan. Only 8% of the types of products (301 in total) always had their actual total lead time within or equal to the expected time frame (theoretical leadtime and 3 days plan). Only a small fraction of products consistently achieve the combined target without delay in any order. Most of the products (over 90% of product types) encountered some orders in which the total lead time was longer than expected.

The distribution of these probabilities shows that exceeding the total coverage target is the norm, rather than the exception. Many products have a high likelihood of delay: a considerable number of product types show a higher probability than 30% that an order's total lead time will exceed the theoretical + 3-day benchmark. There are spikes in the 90 to 100% range, reflecting that there is a set of products for which every order takes longer than the nominal total lead time. This aligns with the earlier findings: since delays can originate from either the supplier or the warehouse (or both), the combined process is even more likely to suffer delays.

Analysis of Supplier and Warehouse Delays

To explore whether supplier and warehouse delays show systematic patterns across prod-

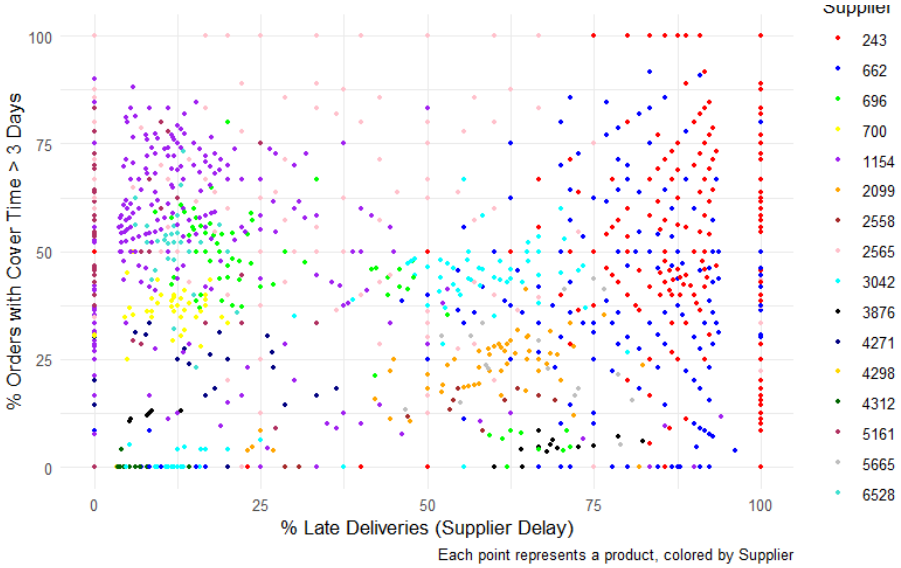

Figure 10: Supplier Vs Cover time delays

ucts, we created a bivariate plot to assess the distribution of product types across the four quadrants of delay behavior as seen in figure Figure 10. Specifically, our goal was to identify whether most products clustered in one quadrant (e.g., supplier delay only, warehouse delay only), whether certain quadrants were sparsely populated, or whether there was evidence of a clear linear relationship (positive or negative) between the two types of delays. However, no such dominant patterns or alignments were observed in the data.

This observation is further supported by an analysis of product types that fall into extreme delay categories across both supplier and warehouse performance. In the extreme point (0%, 0%) of the plot, representing product types that are consistently on time in both supplier delivery and warehouse processing, there are more than 200 product types associated with a diverse set of suppliers, including 243, 662, 700, 1154, and 2565. These combinations suggest a strong reliability of end-to-end supply. On the opposite end, the types of products located on extreme point (100%, 100%),indicating consistent delays from both the supplier and the warehouse, also involve suppliers such as 243, 662, 1154, 2099, and 2565, highlighting that some suppliers appear in the best and worst categories. This suggests that delay performance may be influenced not only by the supplier, but also by product-specific characteristics or internal handling complexity. Furthermore, some prod-

uct types show delays in only one component: those near (100%, 0%) experience consistent supplier delays but timely warehouse processing, often associated with suppliers like 3876, 4271, and 5665; while those near (0%, 100%) face the reverse pattern, timely supplier delivery but frequent warehouse delays, with suppliers like 243, 1154, and 2565 again recurring. These observations suggest that the variability in lead time arises from both external and internal factors.

However, most products are between these extremes, indicating that delays are due to both supplier and internal factors. The bulk of the points are dispersed in the middle of the plot rather than hugging the axes, which means that, for most types of products, neither the supplier nor the warehouse is solely responsible for delays. Instead, a product that occasionally sees late deliveries from the supplier often also experiences some protracted handling times in the warehouse, and vice versa. It is also worth noting that some points may lie near the horizontal or vertical edges (e.g., a high supplier delay percentage but low warehouse delay, or the reverse), indicating cases where one component is consistently on schedule while the other is frequently delayed. These cases are less common than the mixed delay cases, but they do exist (for instance, some products have nearly 0% warehouse delays despite substantial supplier lateness, and a few vice versa). In general, the joint analysis reinforces that both sources of delay contribute to extended coverage periods in the supply chain. Identifying products in extreme categories (always on time, always late in either or both dimensions) helps to flag specific supplier-product relationships that either perform exceptionally well or may require urgent attention. Meanwhile, the broad middle cloud of points underscores that improvement efforts cannot focus on just one facet (supplier or warehouse) in isolation; both aspects have inherent variability that needs to be accounted for.

3.2 Model Results

To evaluate the performance of different machine learning models in predicting lead times for suppliers and warehouses, the Random Forest, XGBoost, and LightGBM algorithms are used. Each model is assessed using 5-fold cross-validation on the training dataset(80% of the data) and subsequently evaluated on a temporally segregated holdout test set (20% of the data). Performance was measured using the root mean squared error (RMSE), mean

absolute error (MAE), and mean absolute percentage error (MAPE), which collectively capture the magnitude, consistency, and scale-independent accuracy of the error.

Supplier Model Results

The evaluation of the supplier lead time prediction models revealed distinct differences in performance across the three algorithms. As shown in Table 6, the LightGBM model achieved the lowest Root Mean Squared Error (RMSE) of 6.142 on the test set, suggesting its superior ability to minimize large prediction errors. However, in terms of Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), the Random Forest model outperformed the others, with values of 1.963 and 20.068%, respectively. These lower error values indicate that Random Forest was more accurate consistently and less prone to large percentage deviations. Although LightGBM performed best in RMSE, Random Forest demonstrated more balanced performance across all three metrics. Therefore, when considering robustness and overall accuracy in test data, Random Forest is identified as the most effective model for predicting supplier lead time in this context.

Table 5: Supplier Model Performance – Training Set Results

Model	**RMSE**	**MAE**	**MAPE (%)**
Random Forest	6.0831	**1.8803**	**20.3720**
XGBoost	5.9639	2.1407	26.8645
LightGBM	**5.7873**	2.1492	26.8393

Table 6: Supplier Model Performance – Test Set Results

Model	**RMSE**	**MAE**	**MAPE (%)**
Random Forest	6.410	**1.963**	**20.068**
XGBoost	6.455	2.196	25.761
LightGBM	**6.142**	2.199	25.780

Interpreting model performance metrics provides valuable insight into the practical reliability of supplier lead-time predictions. A Mean Absolute Error (MAE) of approximately

1.96 working days indicates that, on average, the predicted delivery times deviate from the actual delivery dates by about two days. This level of deviation offers a concrete and interpretable estimate of the uncertainty of the forecast, helping supply chain planners to set reasonable expectations for the precision of delivery.

The Mean Absolute Percentage Error (MAPE) complements this by expressing the average error as a proportion of the actual value. With an MAPE of 20%, the model's lead time predictions are, on average, 20% above or below the true delivery durations. The root mean squared error (RMSE), recorded at 6.41 working days, reflects the average magnitude of the larger prediction errors made by the model, with a stronger penalty applied to extreme deviations. Given that the target variable is the target variable, this value suggests that while the model generally performs well, there are occasional instances where the predicted lead times deviate significantly, by more than six working days, from the actual delivery durations. Such discrepancies may not be frequent but can have substantial operational implications.

Warehouse Model Results

From the evaluated models for warehouse lead time prediction, the Random Forest model demonstrated the most balanced and superior performance across all key evaluation metrics in the test set. From Table 8, the Random Forest model achieved the best performance across all error metrics. With a Root Mean Squared Error (RMSE) of 0.972 working days, a Mean Absolute Error (MAE) of 0.204 working days, and a mean absolute percentage error (MAPE) of 8.53%, it outperformed both XGBoost and LightGBM in terms of predictive precision and consistency.

Table 7: Warehouse Model Performance – Training Set Results

Model	**RMSE**	**MAE**	**MAPE (%)**
Random Forest	**0.8256**	**0.2402**	**10.6591**
XGBoost	1.0513	0.4942	20.5353
LightGBM	1.2695	0.7214	27.8454

Table 8: Warehouse Model Performance – Test Set Results

Model	**RMSE**	**MAE**	**MAPE (%)**
Random Forest	**0.972**	**0.204**	**8.529**
XGBoost	0.963	0.383	16.511
LightGBM	1.216	0.652	25.480

The MAE value indicates that predictions deviate from actual cover times by approximately 0.2 working days on average, which translates to around 4.8 to 5 hours. The MAPE of 8.53% shows that the prediction error remains proportionally small, even as the cover times vary between different scenarios. The root mean square error (RMSE) of 0.972 working days indicates that, on average, the prediction errors tend to vary around one weekday, with larger errors penalized more heavily due to the squaring of differences.

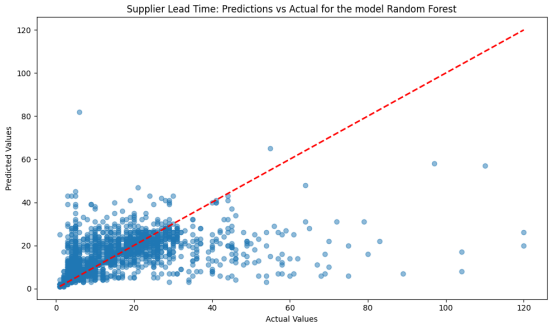

Figure 11: Supplier leadtime (Prediction vs Actual)

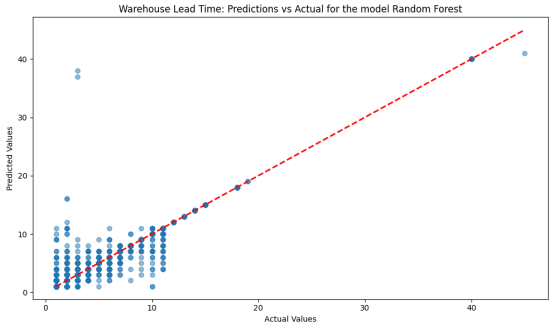

Figure 12: Warehouse leadtime (Prediction vs Actual)

The two scatter plots compare the predicted versus actual lead times for the supplier and warehouse models using Random Forest. The supplier lead time plot shows a concentration of points at lower values but with noticeable under-prediction for longer lead times, reflecting the variability and unpredictability of supplier performance. In contrast, the warehouse lead time plot shows tighter alignment along the diagonal, indicating higher predictive accuracy and more consistent performance. In general, the model performs better in predicting warehouse lead times than supplier lead times. This improved performance may be attributed to the better signal available for prediction of characteristics, such as urgency

signs, delivery time, or refrigeration requirements.

Feature Importance

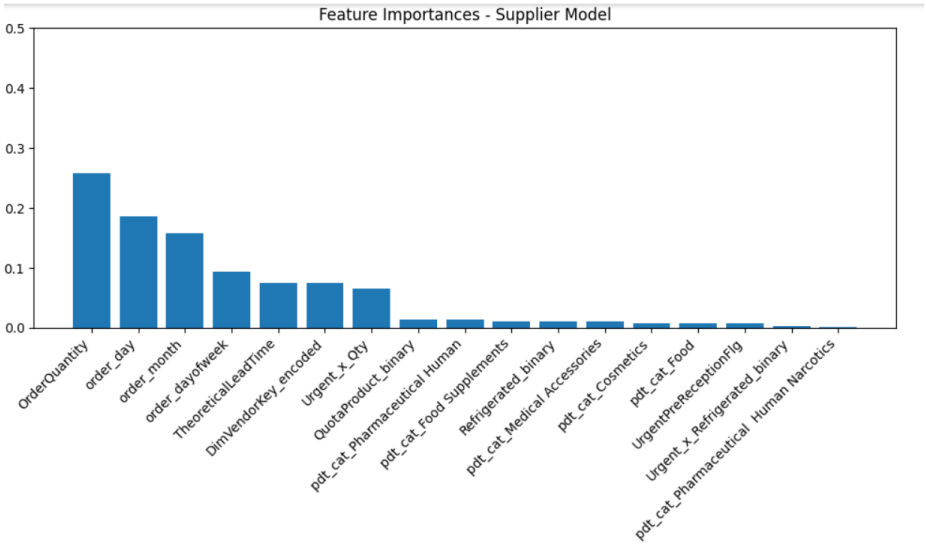

Figure 13: Feature Importance (Supplier)

From Figure 13, the supplier model's feature importance distribution reveals that OrderQuantity is by far the most significant predictor of the lead time of the supplier. This suggests that larger order volumes may be more prone to delays or require more time for procurement, possibly due to batching, supplier capacity constraints, or negotiation overhead. Temporal features such as order day, order month, and order day of week also ranked highly, indicating the presence of temporal patterns in supplier responsiveness, for instance, slower fulfillment near month-ends or on specific weekdays. TheoreticalLeadTime, while included as a baseline indicator, was moderately important, suggesting that the model learned from it but did not rely exclusively on it. Other relevant features include DimVendorKey encoded, Urgent X OrderQuantity, and QuotaProduct, although their influence is relatively lower. Interestingly, product-level flags such as Food Supplements and Pharmaceutical Human had minimal influence, implying that supplier lead times are more responsive to operational and temporal characteristics than to broad product categories.

As shown in Figure 14, the warehouse model placed the greatest importance on UrgentPreReceptionFlg, indicating that warehouse lead times are highly sensitive to the urgency

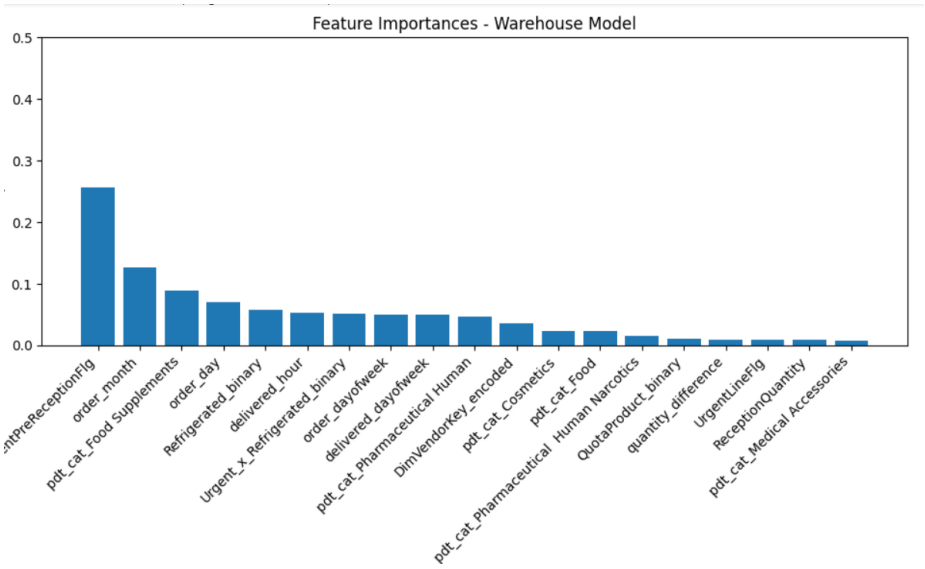

Figure 14: Feature Importance (Warehouse)

classification of incoming orders. This is mainly due to the fact that orders flagged as urgent follow prioritized internal workflows, leading to faster or more variable processing times. Temporal features such as order month, order day, and delivery hour also featured prominently, revealing time-based fluctuations in warehouse throughput, possibly due to labor shifts, batching windows, or peak load periods. Product categories such as food supplements and refrigeration products appeared in the top five characteristics, suggesting that warehouse handling time varies based on the requirements and sensitivity of product storage. Features such as delivery day of the week and interaction between urgent flag and Refrigerated reinforce this idea, showing that product handling characteristics interact with temporal factors to influence actual lead time performance.

Model Performance Across Suppliers

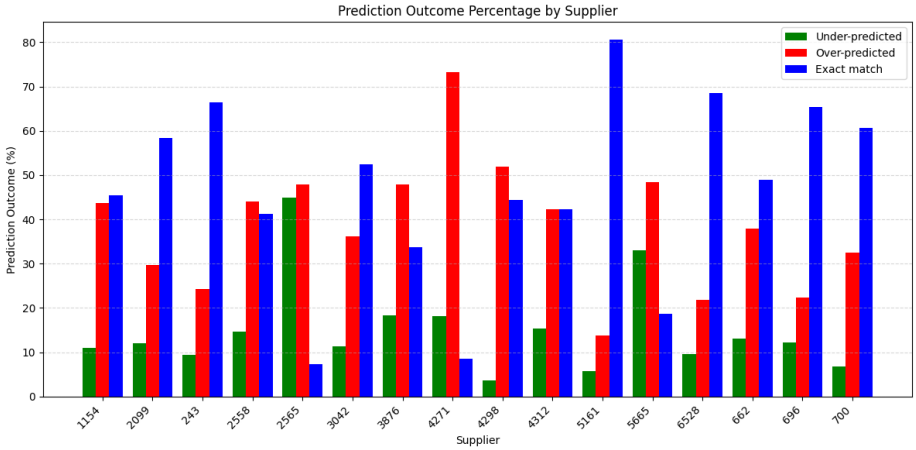

Figure 15: Prediction Percentage (Supplier Model)

From Figure 15, model performance across different suppliers was assessed, and the prediction outcomes were categorized into three groups: under-predicted, over-predicted, and exact match, expressed as percentages. The plot below illustrates the distribution of these outcomes for each supplier in the test set. It reveals a substantial variation in prediction accuracy between suppliers. Some suppliers, such as 243, 5161, and 6528, exhibit a high rate of exact matches, with over 60% of their lead time predictions. This indicates that the model has effectively learned and generalized their lead time for those suppliers.
In contrast, suppliers like 4271 and 4298 show a predominance of over-predicted outcomes, exceeding 70% and 50% respectively. Over-prediction in this context suggests that the model anticipated longer lead times than actually occurred. Although this may not lead to stockouts, it could result in excess safety stock or conservative planning, potentially increasing holding costs.

Suppliers such as 5665 and 2565 display a higher rate of underprediction, indicating that the model systematically underestimated actual lead times. This is a more concerning scenario, as underestimation can lead to stock shortages or late replenishment, especially if predictions are used directly for reorder timing. The variation in predictive performance likely stems from differences in supplier behavior or orders. Suppliers with more stable

and frequent interactions (e.g. Suppliers 243 and 6528) are better modeled, while those with irregular patterns or fewer data points may lead to higher prediction error, especially under- or overestimation.

Uncertainity Estimation Results

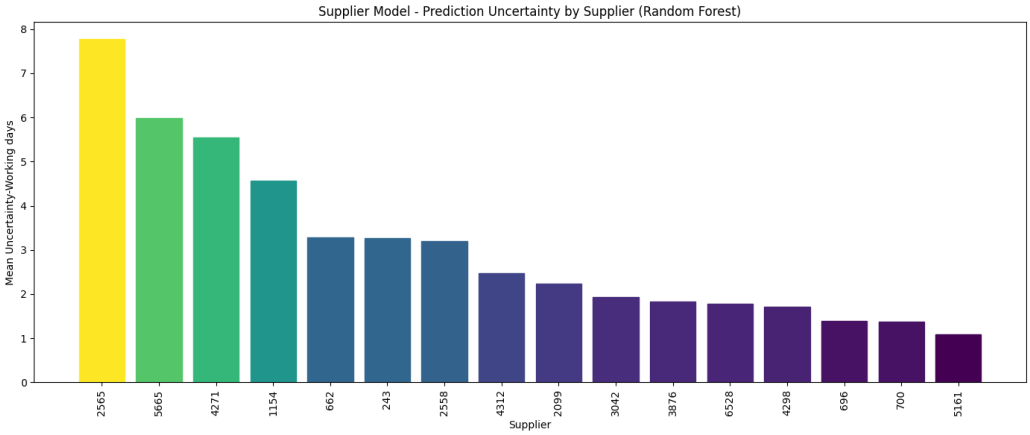

Figure 16: Supplier Model – Prediction Uncertainty by Supplier (Random Forest)

Figure 16 illustrates the prediction uncertainty associated with the Random Forest models related to supplier model characteristics. Each bar represents a supplier, with the height corresponding to the mean prediction uncertainty expressed in working days. The random forest model estimates uncertainty based on the standard deviation of the predictions in all trees in the set, reflecting the confidence of the model in its output for each supplier.
The supplier model shows the trend of uncertainty across the suppliers. Supplier 2565 ranks highest in terms of uncertainty, with an average of nearly 8 working days. Suppliers 5665 and 4271 also rank high, while 5161, 700, and 696 show low prediction variance, indicating stable and learnable delivery patterns. The higher overall magnitude of uncertainty in this model suggests that supplier-level features may introduce more variability or that suppliers themselves may behave less consistently over time. These uncertainties provide valuable operational information on the level of confidence of the model between different suppliers, especially those with high levels of uncertainty, and this could be evidenced by the distributions of these suppliers having outliers, which could be the cause of the high variability.

4 Discussion

The discussion focuses on insights derived from Febelco's lead time data through exploratory analysis, including trends, supplier variability, and deviations from planning expectations. The performance and interpretability of machine learning models applied to predict lead times, assess the importance of features, and discuss the role of uncertainty estimation in improving operational visibility. Together, these analyses offer a multifaceted understanding of lead time dynamics and support the development of data-driven approaches to improve supply chain responsiveness and reliability.

The exploratory analysis revealed substantial variability in both the supplier and internal warehouse lead times, challenging the assumption that average lead times are sufficient for effective inventory planning. Although most orders were fulfilled within 5 to 10 working days in relation to the total lead time, significant outliers, some extending beyond 60 days, highlight inconsistencies that static planning systems may overlook. Approximately 25% of the warehouse orders exceeded the internal 3 day processing standard, and supplier lead times reached 23 days at the 95th percentile. These results point to external and internal sources of unreliability that can erode service levels, disrupt replenishment schedules, and compromise patient-facing availability. According to [1], supply chains must be designed to account for variability and uncertainty, as deterministic models often do not reflect the operational realities of modern logistics. These findings support the shift toward adaptive, data-driven planning methods that incorporate variability into lead time forecasting and inventory control strategies.

Tree-based machine learning models particularly Random Forest, XGBoost, and LightGBM proved to be effective in predicting both supplier and warehouse lead times. Among these, Random Forest delivered the most balanced and interpretable performance, making it the most suitable model for operational implementation. For supplier lead time prediction, the Random Forest model achieved an RMSE of 6.410, an MAE of 1.963 working days, and an MAPE of 20.068% on the test set. In the case of the prediction of the warehouse lead time (cover time), the model yielded an RMSE of 0.972, an MAE of 0.204, and an MAPE of 8.529 %. These results indicate that, on average, supplier delivery times were

predicted with an error of around two working days, while warehouse processing times were predicted with much greater precision, within a fraction of a working day. The results of these models are counterintuitive in comparison to the performance of the training set. The better performance on the test data set could be attributed to the disproportionate representation of a well-predicted variable (e.g., supplier) that can potentially skew aggregate evaluation metrics, leading to unexpected interpretations of model generalization. According to these findings, Random Forest consistently achieved the lowest MAPE compared to other methods in a chemical industry case study [10].

Feature importance analysis provided further insight into the key drivers of lead time variability. Operational factors, particularly the order quantity and urgency classification, were found to be much more influential than fixed product attributes such as category or storage requirements. In the supplier model, the number of orders emerged as the most significant predictor, likely reflecting batching effects or capacity limitations on the supplier side. For the warehouse model, urgency flags were the dominant factor, indicating that internal prioritization protocols greatly influence processing times. Temporal variables such as day of the week and month also ranked highly, suggesting the presence of seasonality or cyclical workload patterns. Surprisingly, product-level classifications such as "Refrigerated" or "Narcotic" played a relatively minor role in determining lead time. This challenges the traditional inventory planning assumption that product characteristics primarily drive fulfillment timelines and emphasizes the need for a more dynamic, context-aware approach to lead time management[18]

The supplier-level prediction analysis revealed considerable variability in model performance across different suppliers. Certain suppliers (243, 6528, 5161) exhibited high exact match rates, indicating stable and predictable delivery behavior. In contrast, others (4271, 4298) showed significant over-prediction tendencies, which may reflect more erratic delivery patterns or other supplier-related factors. This variability shows the diverse operational characteristics present within the supplier network and highlights the complexity involved in achieving uniformly accurate lead time predictions.

The uncertainty plots provide a detailed view of the supplier-level variability in the supplier model. Suppliers such as 2565 and 5665 demonstrated high uncertainty prediction, indi-

cating irregular or inconsistent external delivery behavior. In contrast, suppliers such as 700 and 6528 showed low variance, suggesting stable supply patterns that the model could predict with confidence.

In conclusion, the integration of machine learning models into supply chain forecasting represents a significant advance over static planning systems. The findings demonstrate that predictive analytics can capture real-world complexity and variability more effectively than theoretical assumptions alone. This approach enables organizations to move from reactive to proactive inventory management, improving service levels, reducing waste, and enhancing overall operational resilience. Given the high-stakes nature of pharmaceutical supply chains, such data-driven forecasting tools offer both economic and societal value by supporting timely and reliable product availability.

5 Limitations

Despite the promising results obtained from the application of machine learning models to the prediction of lead time in the pharmaceutical supply chain, several limitations should be acknowledged. The models developed in this study did not incorporate external or macro-level variables that may influence lead times. Factors such as supplier capacity disruptions, transportation delays, regulatory interventions, or macroeconomic events were not represented in the structured data used for model training. In addition, there is an absence of detailed information related to the warehouse workload. Variables such as staffing levels or shift schedules, which are known to influence internal lead times, were not available in the data set.

Another important limitation is the temporal scope of the data used for model training and evaluation. The study relied on a single year of historical lead-time data, which may not be sufficient to capture seasonal effects, year-to-year variability, or infrequent but operationally significant events. A longer data horizon could have improved model robustness and allowed for a deeper understanding of trends and anomalies in both supplier and warehouse processes. The restricted time frame may, therefore, limit the generalizability of the findings and the reliability of the predictions under changing operational conditions.

6 Ethical Thinking, Societal Relevance, and Stakeholder Awareness

6.1 Ethical Thinking

The analysis and modeling of order lead times involve the collection, processing, and interpretation of operational data. Ethical thinking in this context requires ensuring that all data used for analysis are handled responsibly and confidentially. Although the data set used does not contain personal information, it involves sensitive business intelligence such as supplier performance and internal warehouse processing efficiency. As such, ethical data usage mandates securing data, preventing unauthorized access, and avoiding any misuse that could lead to reputational or contractual risks to suppliers or the company.

In addition, the development of a predictive model must adhere to the principles of transparency and fairness. Model output should not lead to unjustified penalization of specific suppliers or internal teams without contextual understanding. Ethical forecasting involves clearly communicating model limitations, avoiding overreliance on predictions without human oversight, and ensuring that the model serves to support, not replace, responsible decision making by supply chain professionals.

6.2 Societal Relevance

Efficient inventory management has broader societal relevance, especially in sectors such as healthcare, pharmaceuticals, and essential goods, areas in which companies like Febelco operate. Timely replenishment and accurate stock availability are critical to ensuring the continuous supply of essential products to pharmacies and hospitals. Lead time variability that results in stockouts or delivery delays can directly affect public health and patient outcomes.

By identifying systemic inefficiencies and proposing data-driven improvements, this research contributes to building more resilient supply chains. A reliable and empirically informed forecasting model not only enhances service levels but also reduces waste, avoids overstocking, and supports more sustainable logistics. In a wider context, such improvements contribute to economic efficiency and resource conservation, both of which are vital to societal well-being.

6.3 Stakeholder Awareness

This research is directly relevant to multiple stakeholder groups within and beyond Febelco. Internally, supply chain managers, inventory planners and IT teams benefit from improved visibility into lead time behavior and the opportunity to recalibrate Slim4 parameters with empirical support. Understanding where and why delays occur allows these stakeholders to make targeted process improvements, renegotiate supplier contracts, or review warehouse procedures.

Externally, suppliers are important stakeholders whose performance is scrutinized. Ethical stakeholder participation requires that such insights be communicated constructively,

focusing on collaborative improvement. Furthermore, end customers (e.g., pharmacies and patients) are indirect but critical stakeholders. Their expectations of consistent product availability depend on the system's ability to anticipate and adapt to variability in supply chain operations.

In summary, the research maintains awareness of the multi-stakeholder environment in which inventory management operates and seeks to provide actionable insights that support transparency, accountability, and continuous improvement across the supply chain.

7 Conclusions and Future Research

This study was aimed at improving lead time visibility and prediction accuracy in Febelco's pharmaceutical supply chain by leveraging machine learning techniques. Exploratory data analysis confirmed a critical issue: The theoretical lead times used in Febelco planning systems (e.g. Slim4) were frequently inaccurate. A significant portion of supplier and warehouse lead times deviated from their expected values, especially for product categories such as refrigerated items, narcotics, and those with quota restrictions. These inconsistencies have direct implications for service levels, stockouts, and excess inventory.

To address this gap, separate predictive models were developed for supplier and warehouse lead times. The modeling results revealed meaningful patterns in lead-time variability. For suppliers, key influencing factors included order quantity and order timing, particularly the day and month of ordering. These findings suggest that supplier performance at Febelco is shaped by ordering behaviors and temporal cycles, indicating that targeted adjustments to procurement timing or batch sizing could reduce delays.

Warehouse lead times, on the other hand, were found to be driven primarily by urgency classifications and product handling requirements, such as refrigeration. This highlights the role of internal operational processes in shaping delivery speed and reinforces the importance of efficient warehouse workflow management. The insights suggest that prioritization protocols and resource allocation for urgent or sensitive products could be further optimized to improve turnaround times.

Future research should explore the integration of additional operational data, including staffing levels, shift patterns, and transport schedules, to improve the accuracy and interpretability of the model. A longer historical data window would enable seasonal modeling and provide a more robust validation base. Segmenting suppliers based on behavioral traits or reliability profiles may also support the development of more specialized forecasting models. Finally, deploying predictive models into Febelco's planning systems would allow dynamic lead time management and real-time decision support, ultimately improving inventory control, order planning, and customer service outcomes.

References

[1] David Simchi-Levi, Philip Kaminsky, and Edith Simchi-Levi. *Designing and Managing the Supply Chain: Concepts, Strategies and Case Studies*. 3rd ed. New York: McGraw-Hill, 2008. ISBN: 9780073341521.

[2] Irshadullah Asim Mohammed and Joydeb Mandal. "The impact of lead time variability on supply chain management". In: *International Journal of Supply Chain Management* 8.2 (2023), pp. 41–55.

[3] Asiye Moosivand, Ali Rajabzadeh Ghatari, and Hamid Reza Rasekh. "Supply chain challenges in pharmaceutical manufacturing companies: Using qualitative system dynamics methodology". In: *Iranian Journal of Pharmaceutical Research* 18.2 (2019), pp. 1103–1116. DOI: 10.22037/ijpr.2019.2389. URL: https://doi.org/10.22037/ijpr.2019.2389.

[4] Alan Rushton, Phil Croucher, and Peter Baker. *The Handbook of Logistics and Distribution Management*. 5th ed. Kogan Page, 2014.

[5] Febelco. *About Febelco*. Accessed: 2025-05-07. 2025. URL: https://www.febelco.be/en/about-us.

[6] S. Islam and S.H. Amin. "Prediction of probable backorder scenarios in the supply chain using Distributed Random Forest and Gradient Boosting Machine learning techniques". In: *Journal of Big Data* 7.1 (2020), p. 65. DOI: 10.1186/s40537-020-00345-2. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00345-2.

[7] Arman Rezasoltani, Ahmad Jafarnejad, and Amir Mohammad Khani. "A voting-based hybrid machine learning model for predicting backorders in the supply chain". In: *Journal of Decisions and Operations Research* (2025). Available online.

[8] Robin Reiners, Christiane B Haubitz, and Ulrich W Thonemann. "Lead Time Prediction for Inventory Optimization With Machine Learning". In: *Production and Operations Management* (2025), pp. 1–16. DOI: 10.1177/10591478251328630. URL: https://journals.sagepub.com/home/pao.

[9] Shehu Sani et al. "Supply Chain 4.0: A Machine Learning-Based Bayesian-Optimized LightGBM Model for Predicting Supply Chain Risk". In: *Machines* 11.9 (2023), p. 888. DOI: 10.3390/machines11090888. URL: https://www.mdpi.com/2075-1702/11/9/888.

[10] Mustafa Can Camur, Sandipp Krishnan Ravi, and Shadi Saleh. *Enhancing Supply Chain Resilience: A Machine Learning Approach for Predicting Product Availability Dates Under Disruption*. arXiv preprint. 2023. arXiv: 2304.14902. URL: https://arxiv.org/abs/2304.14902.

[11] Dariusz Woźniak et al. "Predictive Algorithms for Supply Chain Management: A Comprehensive Approach to Forecasting Delivery Times and Managing Risk". In: *Journal of Modern Science* 57.3 (2024), pp. 498–512. DOI: 10.13166/jms/191218. URL: https://www.jomswsge.com/pdf-191218-113223?filename=Predictive%20algorithms%20for.pdf.

[12] Pankaj Goyal, Shivam Kumar Pandey, and Anjali Bansal. "Understanding and interpreting feature interactions in gradient boosting models". In: *arXiv preprint arXiv:2012.13971* (2020). URL: https://arxiv.org/abs/2012.13971.

[13] Ali Akbar ForouzeshNejad, Farzad Arabikhan, and Shohin Aheleroff. "Optimizing project time and cost prediction using a hybrid XGBoost and simulated annealing algorithm". In: *Machines* 12.12 (2024), p. 867. DOI: 10.3390/machines12120867. URL: https://www.mdpi.com/2075-1702/12/12/867.

[14] Rob J Hyndman and Anne B Koehler. "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22.4 (2006), pp. 679–688.

[15] Tianfeng Chai and Roland R Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature". In: *Geoscientific Model Development* 7.3 (2014), pp. 1247–1250.

[16] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 2623–2631.

[17] Sakir Camur, Orhan Yilmaz, and Volkan Kaplanoglu. "Supply chain forecasting using ensemble learning models: A hybrid approach". In: *Expert Systems with Applications* 212 (2023), p. 118595.

[18] Farzana Mahbuba, Md Nurul Huda, and Md Mahmudur Rahman. "Dynamic lead-time forecasting using machine learning in a make-to-order supply chain". In: *Applied Sciences* 11.21 (2021), p. 10105. DOI: 10.3390/app112110105. URL: https://www.mdpi.com/2076-3417/11/21/10105.

[19] Alex Kendall and Yarin Gal. "What uncertainties do we need in Bayesian deep learning for computer vision?" In: *Advances in neural information processing systems*. Vol. 30. Curran Associates, Inc., 2017.

[20] Sanjay Kumar and Manisha Goswami. "Machine learning-based lead time prediction for supply chain risk mitigation". In: *International Journal of Production Research* 59.7 (2021), pp. 2145–2161.

Appendix

A Inbound Delivery Table Description

ColumnName	**Header/Line**	**Description**
ReceptionNbr	Header	A reception is a technical concept in our module where we register inbound shipments. It models mostly as a 'delivery'. This is the number that indicates a unique reception.
OrderNbr	Line	This is the Purchase Order number. A purchase order is a unique purchase done at one vendor for one warehouse.
OrderLineNbr	Line	This is the number that indicates a unique line on a purchase order.
DimProductKey	Line	This is the key that references to the product.
DimVendorKey	Line	This is the key that references to the vendor.
DimwarehouseKey	Header	This is the key that references to the warehouse.
Type	Line	This is a parameter that categorizes the inbound delivery line in 2 categories, which should have significantly different lead times.
OrderQuantity	Line	The original ordered quantity from the purchase order header. As this is info from the PO header, that means that the same quantity will be added to different inbound delivery lines. Do not sum these up!
ReceptionQuantity	Line	The reception (scanned) quantity for this inbound delivery quantity. These quantities need to be summed to know how much was delivered/receptioned in total.
OrderDate	Line	Order date PO (=OrderDate).
RequestedDate	Line	This is the requested delivery date at the time of order creation.
DeliveredDate	Header	DateTimestamp of the delivery (=DeliveredDateTime). This field is preset at the creation time of a reception. This field can be corrected by a worker if necessary.
DeliveredTime	Header	DateTimestamp of the delivery (=DeliveredDateTime). This field is preset at the creation time of a reception. This field can be corrected by a worker if necessary.
RecCreatedDate	Header	The timestamp when the reception (header) was created/saved.
RecCreatedTime	Header	The timestamp when the reception (header) was created/saved.
DelProcDate (start)	Header	The timestamp when a worker started processing this specific reception, after it was in a queue between the delivery and the unpacking.

DelProcTime (start)	Header	The timestamp when a worker started processing this specific reception, after it was in a queue between the delivery and the unpacking.
ReceptionDate (line)	Line	The timestamp when a worker scanned the detailed line and entered the information like productNbr, Quantity, etc.
ReceptionTime (line)	Line	The timestamp when a worker scanned the detailed line and entered the information like productNbr, Quantity, etc.
DelClosedDate	Header	The timestamp when an administrative worker closed the reception.
DelClosedTime	Header	The timestamp when an administrative worker closed the reception.
DelInStockDate	Line	The timestamp when the quantity on the line was made sellable.
DelInStockTime	Line	The timestamp when the quantity on the line was made sellable.
UrgentPreReceptionFlg	Header	A flag that indicates if the reception is a mixed pallet, with a lot of small orders. We flag this as urgent, because it has a higher priority in the reception. This flag is entered manually by a worker.
UrgentLineFlg	Line	A flag that indicates if the reception line is urgent.
TheoreticalLeadTime	Line	Theoretical or agreed lead time.
RequestedDeliveryDateDifference	Line	Difference between theoretical and effective lead time.
EffectiveLeadTime	Line	Calculated time between 'Order confirmation' and 'in stock'. (Should be the same as the ActualDeliveryPeriod calculation).
Refrigerated	Line	This parameter indicates if the product is in the cold chain or not. This parameter impacts our prioritization process.
storageconditionCode	Line	This parameter is broader than refrigerated and indicates the storage conditions. However, this is of less importance.
QuotaProduct	Line	Indicates if the product has a difficult supply. These are high-priority products with limited delivery.
productcategory	Line	A categorization of the product. This impacts our prioritization process.
Narcotic	Line	Indicates if the product is a narcotics product. This impacts our prioritization process.
buyerGroup	Line	A planning parameter that groups products on a Purchase Order. It has no other function than grouping products.
CoverageGroupCode	Line	A planning parameter that groups products with similar order planning. It dictates the planning date based on frequency, week, day, and buyer.

Table 9: Inbound Delivery: Column Description

R code

```
#********************************** Loading Data ************************************


# Step 1: Read only headers to get column names
col_names <- names(read_excel(file_path, n_max = 0,sheet = 2))

# Step 2: Set all to "guess" by default
col_types <- rep("guess", length(col_names))

# Step 3: Force specific columns to be "text"
col_types[col_names == "OrderNbr"] <- "text"

# Step 4: Read the full data with column types
df <- read_excel(file_path, col_types = col_types, sheet = 2)


## Data preprocessing
# Rename columns for clarity
df_convert3 <- df %>% rename(DelProcStartDate = `DelProcDate (start)`, DelProcStartTime = `DelProcTime (start)`

df_convert3 <- df_convert3 %>%
  group_by(DimProductKey, OrderNbr) %>%
  summarize(
    OrderDate = first(OrderDate),  # Take the first occurrence
    DeliveredDate = first(DeliveredDate),
    DelInStockDate = first(DelInStockDate),
    DelProcStartDate = first(DelProcStartDate),
    DeliveredTime = first(DeliveredTime),
    DelInStockTime = first(DelInStockTime),
    DelProcStartTime = first(DelProcStartTime),
    TheoreticalLeadTime = first(TheoreticalLeadTime),
    EffectiveLeadTime = first(EffectiveLeadTime),
    ReceptionQuantity = sum(ReceptionQuantity),  # Sum reception quantity
    OrderQuantity = first(unique(OrderQuantity)), # Take the first unique order quantity
    DimVendorKey = first(DimVendorKey),
    ReceptionNbr = first(ReceptionNbr),
    DimwarehouseKey = first(DimwarehouseKey),
    UrgentPreReceptionFlg = first(UrgentPreReceptionFlg),
    UrgentLineFlg = first(UrgentLineFlg),
    Refrigerated = first(Refrigerated),
    Narcotic = first(Narcotic),
    productcategory = first(productcategory),
```

```
    QuotaProduct = first(QuotaProduct)


  ) %>%
  ungroup()

# Convert date and time columns to character format

df_convert3 <- df_convert3 %>%
  mutate(
    OrderDate = as.character(OrderDate),
    DeliveredDate = as.character(DeliveredDate),
    DelInStockDate = as.character(DelInStockDate),
    DelProcStartDate = as.character(DelProcStartDate),
    DeliveredTime = as.character(DeliveredTime),
    DelInStockTime = as.character(DelInStockTime),
    DelProcStartTime = as.character(DelProcStartTime)
  )
# Remove fractional seconds from time columns if present
df_convert3 <- df_convert3 %>%
  mutate(
    DeliveredTime = sub("\\.\\d+$", "", DeliveredTime),
    DelInStockTime = sub("\\.\\d+$", "", DelInStockTime),
    DelProcStartTime = sub("\\.\\d+$", "", DelProcStartTime)
  )

df_convert3 <- df_convert3 %>%
  mutate(
    unique_product_order = paste(DimProductKey,OrderNbr)
  )
print(df_convert3)

# Check the format of the date columns before merging
print("Unique values in DelProcStartDate:")
print(unique(df_convert3$DelProcStartDate))

print("Unique values in DelProcStartTime:")
print(unique(df_convert3$DelProcStartTime))

# Merge date and time columns into single datetime columns
df_convert3 <- df_convert3 %>%
  mutate(
    OrderDateTime = as.POSIXct(paste(OrderDate, " ","00:00:00"), format="%Y-%m-%d %H:%M:%S"),
    DeliveredDateTime = as.POSIXct(paste(DeliveredDate," ", DeliveredTime), format="%Y-%m-%d %H:%M:%S"),
    DelInStockDateTime = as.POSIXct(paste(DelInStockDate," ", DelInStockTime), format="%Y-%m-%d %H:%M:%S"),
```

```
    DelProcStartDateTime = as.POSIXct(paste(DelProcStartDate," ", DelProcStartTime), format="%Y-%m-%d %H:%M:%S
    TheoreticalLeadTime = (as.numeric(TheoreticalLeadTime)+1),  # Convert days to hours
    EffectiveLeadTime = (as.numeric(EffectiveLeadTime)+1)        # Convert days to hours
  )

print(df_convert3)

calculate_weekday <- function(start_time, end_time) {
  # Handle incorrect input
  if (start_time > end_time) return(1)

  # Generate sequence of all days between start and end time
  days_seq <- seq(as.Date(start_time), as.Date(end_time), by = "day")

  # Remove weekends
  weekdays_seq <- days_seq[!weekdays(days_seq) %in% c("Saturday", "Sunday")]
  return(length(weekdays_seq))

}



# Compute time differences
df_convert3 <- df_convert3 %>%
  mutate(
    order_to_available = as.numeric(difftime(DelInStockDateTime, OrderDateTime, units = "days")),
    order_to_delivered = as.numeric(difftime( DeliveredDateTime,OrderDateTime, units = "days")),
    delivered_to_processing = as.numeric(difftime(DelProcStartDateTime, DeliveredDateTime, units = "days")),
    processing_to_available = as.numeric(difftime(DelInStockDateTime, DelProcStartDateTime, units = "days")),
    quantity_difference = ReceptionQuantity - OrderQuantity,
    delivered_available_wkdays = mapply(calculate_weekday, DeliveredDateTime, DelInStockDateTime),
    order_available_wkdays = mapply(calculate_weekday, OrderDateTime, DelInStockDateTime),
    order_to_delivered_wkdays = mapply(calculate_weekday, OrderDateTime,DeliveredDateTime),
    delivered_to_processing_wkdays = mapply(calculate_weekday,DeliveredDateTime, DelProcStartDateTime),
    processing_to_available_wkdays = mapply(calculate_weekday,DelProcStartDateTime, DelInStockDateTime),
    cover_time = mapply(calculate_weekday,DeliveredDateTime,DelInStockDateTime)



  )

# Select only the columns of interest
df_convert3 <- df_convert3 %>%
  select(ReceptionNbr,DimProductKey,DimwarehouseKey,OrderNbr,DimVendorKey, OrderDateTime, DeliveredDateTime, D
         OrderQuantity,ReceptionQuantity,quantity_difference,TheoreticalLeadTime, EffectiveLeadTime, order_to_a
```

```
        delivered_to_processing, processing_to_available,unique_product_order,order_available_wkdays,order_to_
        ,UrgentPreReceptionFlg,
        UrgentLineFlg,Refrigerated,Narcotic,productcategory,
        QuotaProduct)

print(df_convert3)

## Exploratory data analysis

# total lead time by percentage

lead_time_plot <- ggplot(df_convert3, aes(x = order_to_available)) +
  geom_histogram(
    aes(y = after_stat((count / sum(count)) * 100)),  # Convert to percentage
    binwidth = 1,
    fill = "#fc8d62",
    color = "black",
    alpha = 0.85,
    boundary = 0
  ) +
  scale_x_continuous(
    limits = c(0, 100),
    breaks = seq(0, 100, by = 5),
    expand = expansion(mult = c(0, 0.01))
  ) +
  scale_y_continuous(
    breaks = seq(0, 18, by = 2),  # Even number tick marks in percentage
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    title = "Distribution of Total Lead Time (Order to Available)",
    x = "Order to Available Time (Working days)",
    y = "Percentage of Orders"
  ) +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold", size = 16),
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10)),
    panel.grid.minor = element_blank()
  )

print(lead_time_plot)

## distribution of supplier lead time
```

```
supplier_leadtime_plot <- ggplot(df_convert3, aes(x = order_to_delivered_wkdays)) +
  geom_histogram(
    aes(y = after_stat(count / sum(count) * 100)),  # Convert count to percentage
    binwidth = 1,
    fill = "#fc8d62",
    color = "black",
    alpha = 0.85,
    boundary = 0
  ) +
  scale_x_continuous(
    limits = c(0, 50),  # Adjust based on your data range
    breaks = seq(0, 50, by = 5),
    expand = expansion(mult = c(0, 0.02))
  ) +
  scale_y_continuous(
    breaks = seq(0, 40, by = 5),  # Adjust as needed based on max %
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    title = "Distribution of Supplier Delivery Lead Time",
    x = "Lead Time (working days)",
    y = "Percentage of Orders"
  ) +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold", size = 16),
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10)),
    panel.grid.minor = element_blank()
  )

plot(supplier_leadtime_plot)

## distribution of cover time

cover_time_plot <- ggplot(df_convert3, aes(x = delivered_available_wkdays)) +
  geom_histogram(
    aes(y = after_stat(count / sum(count) * 100)),  # Convert to percentage
    binwidth = 1,
    fill = "#fc8d62",
    color = "black",
    alpha = 0.85,
    boundary = 0
  ) +
  scale_x_continuous(
```

```
    limits = c(0, 42),
    breaks = seq(0, 50, by = 1),
    expand = expansion(mult = c(0, 0.02))
  ) +
  scale_y_continuous(
    breaks = seq(0, 30, by = 2),  # adjust as needed
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    title = "Distribution of Cover Time",
    x = "Cover Time (working days)",
    y = "Percentage of Orders"
  ) +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold", size = 16),
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10)),
    panel.grid.minor = element_blank()
  )

plot(cover_time_plot)

## distribution of order to delivered time(supplier-product combination)

order_delivery_plot <- ggplot(df_convert3, aes(x = order_to_delivered_wkdays)) +
  geom_histogram(binwidth = 1, fill = "black", color = "blue", alpha = 0.7) +
  facet_wrap(~ DimVendorKey, scales = "free_y", ncol = 4) +
  labs(
    title = "Order-to-Delivery Time Distribution by Supplier",
    x = "Order-to-Delivery Time (Working days)",
    y = "Count"
  ) +
  theme_minimal(base_size = 14) +
  theme(
    strip.text = element_text(size = 12, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1),
    panel.spacing = unit(1.2, "lines"),
    plot.title = element_text(hjust = 0.5, face = "bold")
  )

# View the plot
order_delivery_plot

## Distribution of late orders on suplier side
```

```
#  Create late flag based on supplier delay
df_late2 <- df_convert3 %>%
  mutate(late_flag = order_to_delivered_wkdays > TheoreticalLeadTime)

# Summarise by Product
product_late_pct <- df_late2 %>%
  group_by(DimProductKey) %>%
  summarise(
    total = n(),
    late = sum(late_flag, na.rm = TRUE),
    late_pct = (late / total) * 100
  )

#  Plot Late Deliveries per Product
ggplot(product_late_pct, aes(x = late_pct)) +
  geom_histogram(binwidth = 10,boundary = 0, fill = "#fc8d62", color = "white") +
  scale_x_continuous(breaks = seq(0, 100, by = 10)) +  # Optional: Clean x-axis ticks
  labs(
    #title = "Distribution of Late Orders (Supplier side)",
    x = "% of orders with delivery  lead time > theoretical lead time",
    y = "Number of Products"
  ) +
  theme_minimal()

  ## Supplier vs. Cover Time Delay per Product

  # Step 1: Mark Late Flag for Supplier Delay
df_late2 <- df_convert3 %>%
  mutate(late_flag = order_to_delivered_wkdays > TheoreticalLeadTime)

# Step 2: Summarize Supplier lateness per Product
product_late_pct <- df_late2 %>%
  group_by(DimProductKey, DimVendorKey) %>%
  summarise(
    total = n(),
    late = sum(late_flag, na.rm = TRUE),
    late_pct = (late / total) * 100,
    .groups = 'drop'
  )

# Step 3: Mark Late Flag for Cover Time Delay
df_late <- df_convert3 %>%
  mutate(late_flag = cover_time > 3)
```

```r
# Step 4: Summarize Cover Time lateness per Product
product_covertime_late_pct <- df_late %>%
  group_by(DimProductKey) %>%
  summarise(
    total = n(),
    late = sum(late_flag, na.rm = TRUE),
    late_pct = (late / total) * 100,
    .groups = 'drop'
  )

# Step 5: Combine Both Datasets
product_combined <- product_late_pct %>%
  select(DimProductKey, DimVendorKey, late_pct) %>%
  rename(late_pct_supplier = late_pct) %>%
  inner_join(
    product_covertime_late_pct %>%
      select(DimProductKey, late_pct) %>%
      rename(late_pct_covertime = late_pct),
    by = "DimProductKey"
  )

# Check
print(product_combined)

# Define your custom color palette
my_colors <- c(
  "red", "blue", "green", "yellow", "purple", "orange",
  "brown", "pink", "cyan", "black", "navy", "gold",
  "darkgreen", "maroon", "gray", "turquoise"
)

# Check if enough colors are available
num_suppliers <- length(unique(product_combined$DimVendorKey))

if (num_suppliers > length(my_colors)) {
  stop(paste0(
    "You have ", num_suppliers,
    " suppliers but only ", length(my_colors),
    " colors defined! Add more colors to 'my_colors'."
  ))
}

# Plot
ggplot(product_combined, aes(x = late_pct_supplier, y = late_pct_covertime, color = as.factor(DimVendorKey))) +
  geom_point(size = 1) +
```

```
  scale_color_manual(values = my_colors) +
  labs(
    #title = "Supplier vs. Cover Time Delay per Product",
    x = "% Late Deliveries (Supplier Delay)",
    y = "% Orders with Cover Time > 3 Days",
    color = "Supplier",
    caption = "Each point represents a product, colored by Supplier"
  ) +
  theme_minimal()

## Percentage of Late Orders (Total Lead Time > Theoretical + 3 Days

# Step 1: Flag orders where total lead time exceeds theoretical + 3
df_lead_excess <- df_convert3 %>%
  mutate(late_leadtime_flag = order_to_available > (TheoreticalLeadTime + 3))

# Step 2: Summarize per product
product_leadtime_late_pct <- df_lead_excess %>%
  group_by(DimProductKey) %>%
  summarise(
    total = n(),
    late = sum(late_leadtime_flag, na.rm = TRUE),
    late_pct = (late / total) * 100,
    .groups = 'drop'
  )

ggplot(product_leadtime_late_pct, aes(x = late_pct)) +
  geom_histogram(binwidth = 10,boundary = 0, fill = "#fc8d62", color = "black", alpha = 0.7) +
  scale_x_continuous(breaks = seq(0, 100, 10)) +
  labs(
    title = "Percentage of Late Orders (Total Lead Time > Theoretical + 3 Days)",
    x = "Percentage of Late Orders",
    y = "Number of Products"
  ) +
  theme_minimal()

  ## Distribution of Late Orders

  # Create late flag based on supplier delay
df_late2 <- df_convert3 %>%
  mutate(late_flag = order_to_delivered_wkdays > TheoreticalLeadTime)

# Summarise by Product
product_late_pct <- df_late2 %>%
  group_by(DimProductKey) %>%
```

```
  summarise(
    total = n(),
    late = sum(late_flag, na.rm = TRUE),
    late_pct = (late / total) * 100
  ) %>%
  filter(late_pct > 0)  # <-- Exclude 0% late deliveries

# Plot Late Deliveries per Product
ggplot(product_late_pct, aes(x = late_pct)) +
  geom_histogram(binwidth = 10, boundary = 0, fill = "#fc8d62", color = "white") +
  scale_x_continuous(breaks = seq(0, 100, by = 10)) +
  labs(
    #title = "Distribution of Late Orders (Supplier Delay, Excluding 0%)",
    x = "% of orders with delivery  lead time > theoretical lead time",
    y = "Number of Products"
  ) +
  theme_minimal()

  ## Total lead time versus the Planned time

  # Step 1: Flag orders where total lead time exceeds theoretical + 3
df_lead_excess <- df_convert3 %>%
  mutate(late_leadtime_flag = order_to_available > (TheoreticalLeadTime + 3))

# Step 2: Summarize per product
product_leadtime_late_pct <- df_lead_excess %>%
  group_by(DimProductKey) %>%
  summarise(
    total = n(),
    late = sum(late_leadtime_flag, na.rm = TRUE),
    late_pct = (late / total) * 100,
    .groups = 'drop'
  )

ggplot(product_leadtime_late_pct, aes(x = late_pct)) +
  geom_histogram(binwidth = 10,boundary = 0, fill = "#fc8d62", color = "black", alpha = 0.7) +
  scale_x_continuous(breaks = seq(0, 100, 10)) +
  labs(
    title = "Percentage of Late Orders (Total Lead Time > Theoretical + 3 Days)",
    x = "Percentage of Late Orders",
    y = "Number of Products"
  ) +
  theme_minimal()
```

```
  ### PYTHON CODE

  # Custom evaluation metrics


def mean_absolute_percentage_error(y_true, y_pred):
    """Calculate MAPE"""
    mask = y_true != 0
    return np.mean(np.abs((y_true[mask] - y_pred[mask]) / y_true[mask])) * 100

def evaluate_model(y_true, y_pred, model_name):
    """Evaluate model performance"""
    mae = mean_absolute_error(y_true, y_pred)
    mape = mean_absolute_percentage_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))



    print(f"\n{model_name} Performance:")
    print(f"MAE: {mae:.3f}")
    print(f"MAPE: {mape:.3f}%")
    print(f"RMSE: {rmse:.3f}")


    return {'MAE': mae, 'MAPE': mape, 'RMSE': rmse}


class ModelType(Enum):
    RANDOM_FOREST = 'Random Forest'
    LIGHTGBM = 'LightGBM'
    XGBOOST = 'XGBoost'


class WarehouseLeadTimePredictor:
    def __init__(self, data_path):
        """Initialize the predictor with data"""
        self.df = pd.read_csv(data_path)
        self.supplier_features = []
        self.warehouse_features = []
        self.categorical_columns = ['DimProductKey', 'DimwarehouseKey', 'DimVendorKey']
        self.label_encoders = {}
        self.handle_outliers = True
        self.remove_outliers = False
        self.outlier_threshold = 40
        self.use_optuna = True
```

```
        self.n_trials = 50
        self.cv_folds = 5
        self.plot_grouped_analysis = False
        self.round_results = True

        self.models_to_use = [
            ModelType.RANDOM_FOREST,
            ModelType.LIGHTGBM,
            ModelType.XGBOOST
        ]

    def convert_to_binary(self, value, column_name=None):
        """Convert various string representations to binary values"""
        if pd.isna(value):
            return 0

        value_str = str(value).strip().lower()

        # Special handling for 'Narcotic' column
        if column_name == 'Narcotic':
            # Positive values for narcotic products
            narcotic_positive = ['narcotic', 'yes', 'y', 'true', '1', 't']
            # Negative values for non-narcotic products
            narcotic_negative = ['non narcotic', 'non-narcotic', 'non_narcotic', 'no', 'n', '0', '_n/a', 'n/a'

            if value_str in narcotic_positive:
                return 1
            elif value_str in narcotic_negative:
                return 0
            else:
                print(f"Warning: Unknown value '{value}' in Narcotic column. Converting to 0.")
                return 0

        # Special handling for 'Refrigerated' column
        elif column_name == 'Refrigerated':
            # Positive values for refrigerated products
            refrigerated_positive = ['refrigerated', 'yes', 'y', 'true', '1', 't', 'cold', 'frozen']
            # Negative values for non-refrigerated products
            refrigerated_negative = ['non refrigerated', 'non-refrigerated', 'non_refrigerated', 'no', 'n', 'fa

            if value_str in refrigerated_positive:
                return 1
            elif value_str in refrigerated_negative:
                return 0
            else:
```

```
            print(f"Warning: Unknown value '{value}' in Refrigerated column. Converting to 0.")
            return 0

    # Special handling for 'QuotaProduct' column
    elif column_name == 'QuotaProduct':
        # Positive values for quota products
        QuotaProduct_positive = ['quota']
        # Negative values for non-quota products
        QuotaProduct_negative = ['non quota', '_n/a']

        if value_str in QuotaProduct_positive:
            return 1
        elif value_str in QuotaProduct_negative:
            return 0
        else:
            print(f"Warning: Unknown value '{value}' in QuotaProduct column. Converting to 0.")
            return 0

    # Special handling for 'productcategory' column
    elif column_name == 'productcategory':
        pharmaceutical_human_categories = [
        'Pharmaceutical Human',
        'Pharmaceutical Human Narcotics'
             ]

        if value_str in pharmaceutical_human_categories:
           return 1
        else:
           return 0

    # General case for other columns
    else:
        # Check for positive values
        positive_values = ['yes', 'y', 'true', '1', 't', 'on', 'active', 'enabled']
        if value_str in positive_values:
            return 1

        # Check for negative values
        negative_values = ['no', 'n', 'false', '0', 'f', 'off', 'inactive', 'disabled', '', '_n/a', 'n/a',
        if value_str in negative_values:
            return 0

        # If value doesn't match known patterns, print warning and return 0
        print(f"Warning: Unknown value '{value}' encountered. Converting to 0.")
        return 0
```

```
def preprocess_data(self):
    """Preprocess the data and create features"""
    # Convert datetime columns
    datetime_columns = ['OrderDateTime', 'DeliveredDateTime', 'DelInStockDateTime', 'DelProcStartDateTime']
    for col in datetime_columns:
        if col in self.df.columns:
            self.df[col] = pd.to_datetime(self.df[col], errors='coerce')

    # Create time-based features
    if 'OrderDateTime' in self.df.columns:
        self.df['order_hour'] = self.df['OrderDateTime'].dt.hour
        self.df['order_dayofweek'] = self.df['OrderDateTime'].dt.dayofweek
        self.df['order_month'] = self.df['OrderDateTime'].dt.month
        self.df['order_day'] = self.df['OrderDateTime'].dt.day
        self.df['order_year'] = self.df['OrderDateTime'].dt.year

    if 'DeliveredDateTime' in self.df.columns:
        self.df['delivered_hour'] = self.df['DeliveredDateTime'].dt.hour
        self.df['delivered_dayofweek'] = self.df['DeliveredDateTime'].dt.dayofweek

    # Handle string categorical columns (Refrigerated, Narcotic)
    if 'Refrigerated' in self.df.columns:
        # Convert Refrigerated to binary using the helper function
        self.df['Refrigerated_binary'] = self.df['Refrigerated'].apply(lambda x: self.convert_to_binary(x,

    if 'Narcotic' in self.df.columns:
        # Convert Narcotic to binary using the helper function
        self.df['Narcotic_binary'] = self.df['Narcotic'].apply(lambda x: self.convert_to_binary(x, 'Narcoti

    if 'QuotaProduct' in self.df.columns:
        # Convert QuotaProduct to binary using the helper function
        self.df['QuotaProduct_binary'] = self.df['QuotaProduct'].apply(lambda x: self.convert_to_binary(x,

    if 'productcategory' in self.df.columns:
        # Convert productcategory to binary using the helper function
        # self.df['productcategory_binary'] = self.df['productcategory'].apply(lambda x: self.convert_to_b
        self.df = pd.get_dummies(self.df, columns=['productcategory'], prefix='pdt_cat')

    if self.handle_outliers:
        self.treat_outliers()

    # Encode categorical variables
    for col in self.categorical_columns:
        if col in self.df.columns:
```

```
                self.label_encoders[col] = LabelEncoder()
                self.df[f'{col}_encoded'] = self.label_encoders[col].fit_transform(self.df[col].astype(str).fi

        # Interaction Features
        self.df['Qty_x_LeadTime'] = self.df['OrderQuantity'] * self.df['TheoreticalLeadTime']
        self.df['Urgent_x_LeadTime'] = self.df['UrgentPreReceptionFlg'] * self.df['TheoreticalLeadTime']
        self.df['Refrigerated_x_LeadTime'] = self.df['Refrigerated_binary'] * self.df['TheoreticalLeadTime']
        self.df['Urgent_x_Qty'] = self.df['UrgentPreReceptionFlg'] * self.df['OrderQuantity']
        self.df['Urgent_x_Refrigerated_binary'] = self.df['UrgentPreReceptionFlg'] * self.df['Refrigerated_bina



        # Define features for each model
        self.supplier_features = [
            'DimVendorKey_encoded',
            'UrgentPreReceptionFlg',
            'Refrigerated_binary',
            'order_dayofweek', 'order_month', 'order_day',
            'QuotaProduct_binary',
            'OrderQuantity',
            'TheoreticalLeadTime',
            'Urgent_x_Qty',
            'Urgent_x_Refrigerated_binary',
        ]

        self.warehouse_features = [
            'DimVendorKey_encoded',
            'UrgentPreReceptionFlg', 'UrgentLineFlg',
            'Refrigerated_binary',
            'order_dayofweek', 'order_month', 'order_day',
            'delivered_hour', 'delivered_dayofweek',
            'QuotaProduct_binary',
            'ReceptionQuantity', 'quantity_difference',
            'Urgent_x_Refrigerated_binary'

        ]

        # Adding the product categories 1-hot encoded
        for col in self.df.columns:
            if col.startswith('pdt_cat_'):
                self.warehouse_features.append(col)
                self.supplier_features.append(col)

        # Remove features that don't exist in the dataset
        self.supplier_features = [f for f in self.supplier_features if f in self.df.columns]
```

```
        self.warehouse_features = [f for f in self.warehouse_features if f in self.df.columns]

        # Define target variables
        self.supplier_target = 'order_to_delivered_wkdays'
        self.warehouse_target = 'cover_time'

        # Handle missing values for numeric columns
        numeric_columns = self.df.select_dtypes(include=[np.number]).columns
        for col in numeric_columns:
            self.df[col] = self.df[col].fillna(self.df[col].mean())

        # Print unique values to check string formats (helpful for debugging)
        if 'Refrigerated' in self.df.columns:
            print("Unique values in Refrigerated column:", self.df['Refrigerated'].unique())
        if 'Narcotic' in self.df.columns:
            print("Unique values in Narcotic column:", self.df['Narcotic'].unique())

        return self.df

    def split_data(self, test_size=0.2):
        """Split data into train and test sets (validation handled by CV)"""
        # For supplier model
        X_supplier = self.df[self.supplier_features]
        y_supplier = self.df[self.supplier_target]

        # For warehouse model
        X_warehouse = self.df[self.warehouse_features]
        y_warehouse = self.df[self.warehouse_target]

        # Split: train and test (validation handled by cross-validation)
        X_supplier_train, X_supplier_test, y_supplier_train, y_supplier_test = train_test_split(
            X_supplier, y_supplier, test_size=test_size, random_state=42
        )
        X_warehouse_train, X_warehouse_test, y_warehouse_train, y_warehouse_test = train_test_split(
            X_warehouse, y_warehouse, test_size=test_size, random_state=42
        )

        return {
            'supplier': {
                'X_train': X_supplier_train, 'X_test': X_supplier_test,
                'y_train': y_supplier_train, 'y_test': y_supplier_test
            },
            'warehouse': {
                'X_train': X_warehouse_train, 'X_test': X_warehouse_test,
                'y_train': y_warehouse_train, 'y_test': y_warehouse_test
```

```
            }
        }

    def objective_random_forest(self, trial, X_train, y_train, model_type='supplier'):
        """Optuna objective function for Random Forest"""
        # Suggest hyperparameters
        n_estimators = trial.suggest_int('n_estimators', 50, 300, step=50)
        max_depth = trial.suggest_categorical('max_depth', [5, 10, 15, 20, None])
        min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
        min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 5)
        max_features = trial.suggest_categorical('max_features', ['sqrt', 'log2', None])
        bootstrap = trial.suggest_categorical('bootstrap', [True, False])

        # Create model
        model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            min_samples_split=min_samples_split,
            min_samples_leaf=min_samples_leaf,
            max_features=max_features,
            bootstrap=bootstrap,
            random_state=42,
            n_jobs=-1
        )

        # Cross-validation
        cv_scores = cross_val_score(
            model, X_train, y_train,
            cv=self.cv_folds,
            scoring='neg_root_mean_squared_error',
            n_jobs=-1
        )

        return -cv_scores.mean()  # Return positive RMSE

    def objective_xgboost(self, trial, X_train, y_train, model_type='supplier'):
        """Optuna objective function for XGBoost"""
        # Suggest hyperparameters
        n_estimators = trial.suggest_int('n_estimators', 50, 300, step=50)
        max_depth = trial.suggest_int('max_depth', 3, 10)
        learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3, log=True)
        subsample = trial.suggest_float('subsample', 0.6, 1.0)
        colsample_bytree = trial.suggest_float('colsample_bytree', 0.6, 1.0)
        reg_alpha = trial.suggest_float('reg_alpha', 1e-8, 1.0, log=True)
        reg_lambda = trial.suggest_float('reg_lambda', 1e-8, 1.0, log=True)
```

```
        # Create model
        model = xgb.XGBRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            learning_rate=learning_rate,
            subsample=subsample,
            colsample_bytree=colsample_bytree,
            reg_alpha=reg_alpha,
            reg_lambda=reg_lambda,
            random_state=42,
            verbosity=0,
            enable_categorical=True
        )

        # Cross-validation
        cv_scores = cross_val_score(
            model, X_train, y_train,
            cv=self.cv_folds,
            scoring='neg_root_mean_squared_error',
            n_jobs=-1
        )

        return -cv_scores.mean()

    def objective_lightgbm(self, trial, X_train, y_train, model_type='supplier'):
        """Optuna objective function for LightGBM"""
        # Suggest hyperparameters
        n_estimators = trial.suggest_int('n_estimators', 50, 300, step=50)
        max_depth = trial.suggest_int('max_depth', 3, 15)
        learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3, log=True)
        num_leaves = trial.suggest_int('num_leaves', 10, 300)
        subsample = trial.suggest_float('subsample', 0.6, 1.0)
        colsample_bytree = trial.suggest_float('colsample_bytree', 0.6, 1.0)
        reg_alpha = trial.suggest_float('reg_alpha', 1e-8, 1.0, log=True)
        reg_lambda = trial.suggest_float('reg_lambda', 1e-8, 1.0, log=True)

        # Create model
        model = lgb.LGBMRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            learning_rate=learning_rate,
            num_leaves=num_leaves,
            subsample=subsample,
            colsample_bytree=colsample_bytree,
```

```
            reg_alpha=reg_alpha,
            reg_lambda=reg_lambda,
            random_state=42,
            verbose=-1
        )

        # Cross-validation
        cv_scores = cross_val_score(
            model, X_train, y_train,
            cv=self.cv_folds,
            scoring='neg_root_mean_squared_error',
            n_jobs=-1
        )

        return -cv_scores.mean()

    def optimize_hyperparameters(self, X_train, y_train, model_type, target_name):
        """Optimize hyperparameters using Optuna"""
        best_models = {}
        best_params = {}

        for model_name in self.models_to_use:
            print(f"\nOptimizing {model_name.value} for {target_name}...")

            # Create study
            study = optuna.create_study(direction='minimize', study_name=f"{model_name.value}_{target_name}")

            # Define and optimize based on model type
            if model_name == ModelType.RANDOM_FOREST:
                def objective(trial):
                    return self.objective_random_forest(trial, X_train, y_train, model_type)
                study.optimize(objective, n_trials=self.n_trials, show_progress_bar=True)
            elif model_name == ModelType.XGBOOST:
                def objective(trial):
                    return self.objective_xgboost(trial, X_train, y_train, model_type)
                study.optimize(objective, n_trials=self.n_trials, show_progress_bar=True)
            elif model_name == ModelType.LIGHTGBM:
                def objective(trial):
                    return self.objective_lightgbm(trial, X_train, y_train, model_type)
                study.optimize(objective, n_trials=self.n_trials, show_progress_bar=True)

            # Get best parameters
            best_params[model_name] = study.best_params
            print(f"Best parameters for {model_name.value}: {study.best_params}")
            print(f"Best CV RMSE: {study.best_value:.4f}")
```

```
            # Train final model with best parameters
            if model_name == ModelType.RANDOM_FOREST:
                best_model = RandomForestRegressor(**study.best_params, random_state=42, n_jobs=-1)
            elif model_name == ModelType.XGBOOST:
                best_model = xgb.XGBRegressor(**study.best_params, random_state=42, verbosity=0, enable_categor
            elif model_name == ModelType.LIGHTGBM:
                best_model = lgb.LGBMRegressor(**study.best_params, random_state=42, verbose=-1)

            best_model.fit(X_train, y_train)
            best_models[model_name] = best_model

        return best_models, best_params

    def train_supplier_models(self, data_splits):
        """Train models for supplier lead time prediction with Optuna optimization"""
        X_train = data_splits['supplier']['X_train']
        y_train = data_splits['supplier']['y_train']

        if self.use_optuna:
            models, best_params = self.optimize_hyperparameters(X_train, y_train, 'supplier', 'supplier_leadti
        else:
            # Fallback to default parameters
            models = {}
            if ModelType.RANDOM_FOREST in self.models_to_use:
                models[ModelType.RANDOM_FOREST] = RandomForestRegressor(n_estimators=100, random_state=42, n_j
                models[ModelType.RANDOM_FOREST].fit(X_train, y_train)
            if ModelType.XGBOOST in self.models_to_use:
                models[ModelType.XGBOOST] = xgb.XGBRegressor(n_estimators=100, random_state=42, verbosity=0)
                models[ModelType.XGBOOST].fit(X_train, y_train)
            if ModelType.LIGHTGBM in self.models_to_use:
                models[ModelType.LIGHTGBM] = lgb.LGBMRegressor(n_estimators=100, random_state=42, verbose=-1)
                models[ModelType.LIGHTGBM].fit(X_train, y_train)

        # Evaluate models using cross-validation
        from sklearn.model_selection import cross_validate
        results = {}
        print("\n=== Supplier Model Cross-Validation Results ===")

        # Custom MAPE scorer
        from sklearn.metrics import make_scorer
        mape_scorer = make_scorer(lambda y_true, y_pred: mean_absolute_percentage_error(y_true, y_pred), greate

        for model_name, model in models.items():
            cv_results = cross_validate(
```

```
                model, X_train, y_train,
                cv=self.cv_folds,
                scoring={
                    'rmse': 'neg_root_mean_squared_error',
                    'mae': 'neg_mean_absolute_error',
                    'mape': mape_scorer
                },
                return_train_score=False
            )

            results[model_name] = {
                'CV_RMSE_mean': -cv_results['test_rmse'].mean(),
                'CV_RMSE_std': cv_results['test_rmse'].std(),
                'CV_MAE_mean': -cv_results['test_mae'].mean(),
                'CV_MAE_std': cv_results['test_mae'].std(),
                'CV_MAPE_mean': -cv_results['test_mape'].mean(),
                'CV_MAPE_std': cv_results['test_mape'].std()
            }

            print(f"{model_name.value}:")
            print(f"   RMSE: {-cv_results['test_rmse'].mean():.4f} (+/- {cv_results['test_rmse'].std() * 2:.4f}
            print(f"   MAE: {-cv_results['test_mae'].mean():.4f} (+/- {cv_results['test_mae'].std() * 2:.4f})")
            print(f"   MAPE: {-cv_results['test_mape'].mean():.4f}% (+/- {cv_results['test_mape'].std() * 2:.4

        return models, results

    def train_warehouse_models(self, data_splits):
        """Train models for warehouse lead time prediction with Optuna optimization"""
        X_train = data_splits['warehouse']['X_train']
        y_train = data_splits['warehouse']['y_train']

        if self.use_optuna:
            models, best_params = self.optimize_hyperparameters(X_train, y_train, 'warehouse', 'warehouse_leadt
        else:
            # Fallback to default parameters
            models = {}
            if ModelType.RANDOM_FOREST in self.models_to_use:
                models[ModelType.RANDOM_FOREST] = RandomForestRegressor(n_estimators=100, random_state=42, n_jo
                models[ModelType.RANDOM_FOREST].fit(X_train, y_train)
            if ModelType.XGBOOST in self.models_to_use:
                models[ModelType.XGBOOST] = xgb.XGBRegressor(n_estimators=100, random_state=42, verbosity=0)
                models[ModelType.XGBOOST].fit(X_train, y_train)
            if ModelType.LIGHTGBM in self.models_to_use:
                models[ModelType.LIGHTGBM] = lgb.LGBMRegressor(n_estimators=100, random_state=42, verbose=-1)
                models[ModelType.LIGHTGBM].fit(X_train, y_train)
```

```
        # Evaluate models using cross-validation
        from sklearn.model_selection import cross_validate
        results = {}
        print("\n=== Warehouse Model Cross-Validation Results ===")

        # Custom MAPE scorer
        from sklearn.metrics import make_scorer
        mape_scorer = make_scorer(lambda y_true, y_pred: mean_absolute_percentage_error(y_true, y_pred), greate

        for model_name, model in models.items():
            cv_results = cross_validate(
                model, X_train, y_train,
                cv=self.cv_folds,
                scoring={
                    'rmse': 'neg_root_mean_squared_error',
                    'mae': 'neg_mean_absolute_error',
                    'mape': mape_scorer
                },
                return_train_score=False
            )

            results[model_name] = {
                'CV_RMSE_mean': -cv_results['test_rmse'].mean(),
                'CV_RMSE_std': cv_results['test_rmse'].std(),
                'CV_MAE_mean': -cv_results['test_mae'].mean(),
                'CV_MAE_std': cv_results['test_mae'].std(),
                'CV_MAPE_mean': -cv_results['test_mape'].mean(),
                'CV_MAPE_std': cv_results['test_mape'].std()
            }

            print(f"{model_name.value}:")
            print(f"   RMSE: {-cv_results['test_rmse'].mean():.4f} (+/- {cv_results['test_rmse'].std() * 2:.4f
            print(f"   MAE: {-cv_results['test_mae'].mean():.4f} (+/- {cv_results['test_mae'].std() * 2:.4f})"
            print(f"   MAPE: {-cv_results['test_mape'].mean():.4f}% (+/- {cv_results['test_mape'].std() * 2:.4

        return models, results


    def feature_importance_analysis(self, model, feature_names, model_type='supplier'):
        """Analyze feature importance"""
        if hasattr(model, 'feature_importances_'):
            importances = model.feature_importances_

            # Normalize importances to 0-1 scale for consistent comparison
```

```
            normalized_importances = importances / importances.sum()

            indices = np.argsort(normalized_importances)[::-1]

            plt.figure(figsize=(10, 6))
            plt.title(f'Feature Importances - {model_type.capitalize()} Model')
            plt.bar(range(len(normalized_importances)), normalized_importances[indices])
            plt.xticks(range(len(normalized_importances)), [feature_names[i] for i in indices], rotation=45, ha
            plt.ylabel('Normalized Importance (0-1)')
            plt.ylim(0, max(0.5, normalized_importances.max() * 1.1))
            plt.tight_layout()
            plt.show()

            # Print feature importances (both original and normalized)
            print(f"\nTop 5 important features for {model_type} model:")
            for i in range(min(5, len(indices))):
                original_imp = importances[indices[i]]
                normalized_imp = normalized_importances[indices[i]]
                print(f"{feature_names[indices[i]]}: {normalized_imp:.4f} (original: {original_imp:.4f})")


    def plot_predictions_vs_actual(self, y_true, y_pred, title="Predictions vs Actual"):
        """Plot predictions against actual values"""
        plt.figure(figsize=(10, 6))
        plt.scatter(y_true, y_pred, alpha=0.5)
        plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], 'r--', lw=2)
        plt.xlabel('Actual Values')
        plt.ylabel('Predicted Values')
        plt.title(title)
        plt.tight_layout()
        plt.show()



    def plot_prediction_outcome_percentage_by_supplier(self, y_true, y_pred, supplier_ids, top_n=16):
        # Determine outcome type
        outcome = np.where(y_pred > y_true, 'Over-predicted',
                  np.where(y_pred < y_true, 'Under-predicted', 'Exact match'))

        # Create DataFrame
        df = pd.DataFrame({
            'supplier': supplier_ids,
            'outcome': outcome
        })
```

```
        # Count predictions per supplier and outcome
        grouped = df.groupby(['supplier', 'outcome']).size().unstack(fill_value=0)

        # Calculate percentage distribution
        grouped_percent = grouped.div(grouped.sum(axis=1), axis=0) * 100

        # Sort by % over-predicted (if it exists) and select top N
        #if 'Over-predicted' in grouped_percent.columns:
            #top_suppliers = grouped_percent.sort_values(by='Over-predicted', ascending=False).head(top_n)
        # else:
        #     top_suppliers = grouped_percent.head(top_n)

        top_suppliers = grouped_percent.head(top_n)

        # Plot
        fig, ax = plt.subplots(figsize=(max(12, len(top_suppliers) * 0.6), 6))

        # Get available columns and set up bar positions
        available_columns = [col for col in ['Under-predicted', 'Over-predicted', 'Exact match']
                             if col in top_suppliers.columns]

        colors = {'Under-predicted': 'green', 'Over-predicted': 'red', 'Exact match': 'blue'}

        n_bars = len(available_columns)
        width = 0.8 / n_bars if n_bars > 0 else 0.8
        x = np.arange(len(top_suppliers))

        # Plot each available column
        for i, col in enumerate(available_columns):
            offset = (i - (n_bars - 1) / 2) * width
            ax.bar(x + offset, top_suppliers[col], width, label=col, color=colors[col])

        ax.set_xlabel("Supplier")
        ax.set_ylabel("Prediction Outcome (%)")
        ax.set_title("Prediction Outcome Percentage by Supplier")
        ax.set_xticks(x)
        ax.set_xticklabels(top_suppliers.index, rotation=45, ha='right')
        ax.legend()
        ax.grid(axis='y', linestyle='--', alpha=0.5)

        plt.tight_layout()
        plt.show()
```

```
def run_full_pipeline(self):
    """Run the complete pipeline"""
    # Preprocess data
    print("Preprocessing data...")
    self.preprocess_data()

    if self.plot_grouped_analysis:
        self.grouped_analysis()

    # Split data
    print("Splitting data...")
    data_splits = self.split_data()

    # Train models
    print("\n=== Training Supplier Models ===")
    supplier_models, supplier_results = self.train_supplier_models(data_splits)

    print("\n=== Training Warehouse Models ===")
    warehouse_models, warehouse_results = self.train_warehouse_models(data_splits)

    # Find best models based on CV RMSE
    best_supplier_model_name = min(supplier_results.items(), key=lambda x: x[1]['CV_RMSE_mean'])[0]
    best_warehouse_model_name = min(warehouse_results.items(), key=lambda x: x[1]['CV_RMSE_mean'])[0]

    print(f"\nBest supplier model: {best_supplier_model_name.value}")
    print(f"Best warehouse model: {best_warehouse_model_name.value}")

    # Feature importance analysis
    if best_supplier_model_name in self.models_to_use:
        self.feature_importance_analysis(
            supplier_models[best_supplier_model_name],
            self.supplier_features,
            'supplier'
        )

    if best_warehouse_model_name in self.models_to_use:
        self.feature_importance_analysis(
            warehouse_models[best_warehouse_model_name],
            self.warehouse_features,
            'warehouse'
        )

    # Final evaluation on test set
```

```
        print("\n=== Final Test Set Evaluation ===")

        # Supplier model
        X_test_supplier = data_splits['supplier']['X_test']
        y_test_supplier = data_splits['supplier']['y_test']

        # Printing results for other supplier models except best model
        for supplier_model_name, supplier_model in supplier_models.items():
            if supplier_model_name.value == best_supplier_model_name.value:
                continue

            predictions = self.predict(supplier_models[supplier_model_name], X_test_supplier)

            print(f"\nSupplier Test Results for model: {supplier_model_name.value}")

            plot_title = f"Supplier Lead Time: Predictions vs Actual for the model {supplier_model_name.value}"
            self.evaluate_model_prediction(supplier_model_name.value, y_test_supplier, predictions, plot_title)


        # Printing results for best supplier model
        supplier_pred = self.predict(supplier_models[best_supplier_model_name], X_test_supplier)

        print(f"\nTest Results for BEST Supplier model: {best_supplier_model_name.value}")

        plot_title = f"Supplier Lead Time: Predictions vs Actual {best_supplier_model_name.value}"
        self.evaluate_model_prediction(best_supplier_model_name.value, y_test_supplier, supplier_pred, plot_title)


        # Use the test set indices to get original vendor keys
        test_indices = X_test_supplier.index

        # Retrieve the actual vendor keys using test indices
        vendor_ids = self.df.loc[test_indices, 'DimVendorKey']

        # Since we don't have vendor names, use the vendor keys as identifiers
        # Convert to string for better display in the plot
        vendor_identifiers = vendor_ids.astype(str)

        self.plot_prediction_outcome_percentage_by_supplier(
            y_test_supplier.to_numpy(),
            supplier_pred,
            vendor_identifiers.to_numpy(),
            top_n=16
        )
```

```
        # Compute and plot uncertainty if best model is Random Forest
        if best_supplier_model_name == ModelType.RANDOM_FOREST:
            rf_model = supplier_models[ModelType.RANDOM_FOREST]
            pred_matrix = get_rf_prediction_distribution(rf_model, X_test_supplier)
            uncertainty = compute_prediction_uncertainty(pred_matrix)
            plot_uncertainty_by_supplier(uncertainty, vendor_ids)


        # Warehouse model
        X_test_warehouse = data_splits['warehouse']['X_test']
        y_test_warehouse = data_splits['warehouse']['y_test']

        # Printing results for other warehouse models except best model
        for warehouse_model_name,warehouse_model in warehouse_models.items():
            if warehouse_model_name.value == best_warehouse_model_name.value:
                continue

            predictions = self.predict(warehouse_models[warehouse_model_name], X_test_warehouse)

            print(f"\nWarehouse Test Results for model: {warehouse_model_name.value}")
            plot_title = f"Warehouse Lead Time: Predictions vs Actual for the model {warehouse_model_name.valu
            self.evaluate_model_prediction(warehouse_model_name.value, y_test_warehouse, predictions, plot_titl


        # Printing results for best warehouse model
        print(f"\Test Results for BEST Warehouse model: {best_warehouse_model_name.value}")
        warehouse_pred = self.predict(warehouse_models[best_warehouse_model_name], X_test_warehouse)

        print(f"\nWarehouse Test Results for BEST model: {best_warehouse_model_name.value}")
        plot_title = f"Warehouse Lead Time: Predictions vs Actual for the model {best_warehouse_model_name.valu
        self.evaluate_model_prediction(best_warehouse_model_name.value, y_test_warehouse, warehouse_pred, plot_

        # Plot comparisons
        supplier_planned_leadtime = X_test_supplier['TheoreticalLeadTime']
        warehouse_planned_leadtime = [3.0] * y_test_warehouse.shape[0]
        self.plot_target_comparison(y_test_supplier, supplier_planned_leadtime, supplier_pred, plot_name="Suppl
        self.plot_target_comparison(y_test_warehouse, warehouse_planned_leadtime, warehouse_pred, plot_name="Wa

        return {
            'supplier_models': supplier_models,
            'warehouse_models': warehouse_models,
            'supplier_results': supplier_results,
            'warehouse_results': warehouse_results,
            'best_models': {
                'supplier': best_supplier_model_name,
```

```
                'warehouse': best_warehouse_model_name
            },
            'supplier_test_data': {
                'features': X_test_supplier,
                'target': y_test_supplier
            },
            'warehouse_test_data': {
                'features': X_test_warehouse,
                'target': y_test_warehouse
            }
        }


    def predict(self, model, X_test):
        if self.round_results:
            predictions = np.round(model.predict(X_test)).astype(int)
        else:
            predictions = model.predict(X_test)

        return predictions


    def evaluate_model_prediction(self, model_name, y_test, predictions, plot_title):
        evaluate_model(y_test, predictions, model_name)
        self.plot_predictions_vs_actual(y_test, predictions, plot_title)


    def remove_outliers_with_threshold(self):
        self.df = self.df[self.df['order_to_delivered_wkdays'] <= self.outlier_threshold]

    def cap_outliers(self, column):
        Q1 = self.df[column].quantile(0.25)
        Q3 = self.df[column].quantile(0.75)
        IQR = Q3 - Q1
        upper_bound = Q3 + 1.5 * IQR

        self.df[column] = self.df[column].clip(upper=upper_bound)

    def treat_outliers(self):
        if self.remove_outliers and ('order_to_delivered_wkdays' in self.df.columns):
            self.remove_outliers_with_threshold()
        else:
            variables_to_cap = [
                "EffectiveLeadTime",
                "order_to_available",
```

```
            "order_to_delivered",
            "processing_to_available",
            "order_available_wkdays",
            "order_to_delivered_wkdays",
            "processing_to_available_wkdays"
        ]

        for col in variables_to_cap:
            if col in self.df.columns:
                self.cap_outliers(col)

def identify_outlier_candidates(self, df):
    """Systematically identify which variables need outlier checking"""

    # Get only numerical columns
    numerical_cols = df.select_dtypes(include=[np.number]).columns

    outlier_candidates = {}

    for col in numerical_cols:
        # Calculate basic stats
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1

        # Outlier bounds
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

        # Count outliers
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
        outlier_count = len(outliers)
        outlier_percentage = (outlier_count / len(df)) * 100

        # Check for extreme skewness
        skewness = df[col].skew()

        # Store results
        outlier_candidates[col] = {
            'outlier_count': outlier_count,
            'outlier_percentage': outlier_percentage,
            'skewness': skewness,
            'min': df[col].min(),
            'max': df[col].max(),
            'mean': df[col].mean(),
```

```
                'median': df[col].median(),
                'std': df[col].std()
            }

        return outlier_candidates

    def print_candidates_that_need_attention(self):
        # Usage
        candidates = self.identify_outlier_candidates(self.df)

        # Print candidates that need attention
        for col, stats in candidates.items():
            if stats['outlier_percentage'] > 5 or abs(stats['skewness']) > 2:
                print(f"\n{col}:")
                print(f"  Outliers: {stats['outlier_count']} ({stats['outlier_percentage']:.1f}%)")
                print(f"  Skewness: {stats['skewness']:.2f}")
                print(f"  Range: {stats['min']:.2f} to {stats['max']:.2f}")

                self.visualize_outliers(self.df, col)

    def visualize_outliers(self, df, column, log_scale=True):
        # Basic stats
        q1 = df[column].quantile(0.25)
        q3 = df[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

        print(f"\n{column} stats:")
        print(f"  Q1: {q1:.2f}, Q3: {q3:.2f}")
        print(f"  IQR: {iqr:.2f}")
        print(f"  Lower Bound: {lower_bound:.2f}, Upper Bound: {upper_bound:.2f}")
        print(f"  Skewness: {df[column].skew():.2f}")

        fig, axes = plt.subplots(figsize=(8, 3))

        # Histogram
        sns.histplot(df[column], bins=100, ax=axes)

        axes.set_title("Histogram")
        axes.axvline(upper_bound, color='red', linestyle='--', label='Upper Bound')
        axes.axvline(lower_bound, color='green', linestyle='--', label='Lower Bound')
        axes.legend()

        plt.tight_layout()
```

```
        plt.show()

    def grouped_analysis(self):
        self.simple_box_plot(self.df, 'order_dayofweek', 'cover_time')
        self.simple_box_plot(self.df, 'order_month', 'cover_time')
        self.simple_box_plot(self.df, 'order_day', 'cover_time')

    def simple_box_plot(self, df, x_col, y_col):
        plt.figure(figsize=(10, 6))
        sns.boxplot(data=df, x=x_col, y=y_col)
        plt.title(f"{y_col} by {x_col}")
        plt.xlabel(x_col)
        plt.ylabel(y_col)
        plt.show()




    def plot_target_comparison(self, y_true, y_planned, y_pred, plot_name="Cover time"):
        #self.plot_target_comparison_line(y_true, y_planned, y_pred, plot_name)
        #self.plot_target_comparison_scatter(y_true, y_planned, y_pred, plot_name)
        self.plot_target_comparison_bar(y_true, y_planned, y_pred, plot_name)
        #self.plot_target_comparison_strip(y_true, y_planned, y_pred, plot_name)
        #self.plot_prediction_outcome_counts(y_true, y_planned, y_pred, plot_name)
        #self.plot_prediction_error_by_vendor(y_true, y_planned, y_pred, plot_name)
        #self.plot_prediction_outcome_by_supplier_grouped(y_true, y_pred, y_planned)
        if plot_name == "Cover time":
            self.plot_prediction_outcome_percentage_by_supplier(y_true, y_pred, y_planned)


    def plot_target_comparison_line(self, y_true, y_planned, y_pred, plot_name="Cover time"):
        plt.figure(figsize=(12, 6))
        plt.plot(y_true, label=f"True {plot_name}", marker='o')
        plt.plot(y_planned, label=f"Planned {plot_name}", linestyle='--')
        plt.plot(y_pred, label=f"Predicted {plot_name}", marker='x')
        plt.title(f"{plot_name} Comparison")
        plt.xlabel('Test set')
        plt.ylabel('Days')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    def plot_target_comparison_scatter(self, y_true, y_planned, y_pred, plot_name="Cover time"):
        n = y_true.shape[0]
```

```
        x = np.arange(n)

        plt.figure(figsize=(14, 6))

        # Add scatter points
        plt.scatter(x - 0.2, y_true, label=f"True {plot_name}", alpha=0.7)
        plt.scatter(x, y_planned, label=f"Planned {plot_name}", alpha=0.7)
        plt.scatter(x + 0.2, y_pred, label=f"Predicted {plot_name}", alpha=0.7)

        # Formatting
        plt.title(f"{plot_name} Comparison")
        plt.xlabel('Test set')
        plt.ylabel('Days')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()




def plot_prediction_outcome_counts(self, y_true, y_planned, y_pred, plot_name="Cover time"):
    """
    Plots the number of under-, over-, and exactly predicted values as a bar chart.

    Parameters:
    - y_true: array-like of true values
    - y_pred: array-like of predicted values
    """
    # Compute prediction error
    error = np.array(y_pred) - np.array(y_true)

    # Count outcomes
    over_pred_count = np.sum(error > 0)
    under_pred_count = np.sum(error < 0)
    exact_match_count = np.sum(error == 0)

    # Prepare data
    labels = ['Under-predicted', 'Over-predicted', 'Exact match']
    counts = [under_pred_count, over_pred_count, exact_match_count]
    colors = ['green', 'red', 'blue']

    # Plot
    plt.figure(figsize=(8, 5))
```

```
        plt.bar(labels, counts, color=colors)
        plt.ylabel("Number of Orders")
        plt.title(f"{plot_name}Prediction Outcome Comparison")
        plt.grid(axis='y', linestyle='--', alpha=0.5)
        plt.tight_layout()
        plt.show()

        # Optionally return the counts
        return {
            'under_predicted': under_pred_count,
            'over_predicted': over_pred_count,
            'exact_match': exact_match_count
        }



    def plot_target_comparison_bar(self, y_true, y_planned, y_pred, plot_name="Cover time"):
        # Use first 10 samples (to avoid overcrowding)
        n = min(10, y_true.shape[0])
        x = np.arange(n)  # positions for each sample

        # Prepare bar widths and positions
        width = 0.25

        # True, predicted, planned
        true_vals = y_true[:n]
        pred_vals = y_pred[:n]
        plan_vals = y_planned[:n]

        # Create plot
        plt.figure(figsize=(12, 6))

        plt.bar(x - width, true_vals, width=width, label='True', color='steelblue')
        plt.bar(x, plan_vals, width=width, label='Planned', color='gray')
        plt.bar(x + width, pred_vals, width=width, label='Predicted', color='seagreen')

        # Labels
        plt.xlabel('Test set')
        plt.ylabel("Days")
        plt.title(f"{plot_name} Comparison (First 10 samples)")
        plt.xticks(x, [f"#{i}" for i in range(n)])
        plt.legend()
        plt.grid(axis='y')
        plt.tight_layout()
        plt.show()
```

```
def plot_prediction_error_by_vendor(self, y_true, y_planned, y_pred, plot_name="Cover time"):
    """
    Plots average absolute prediction error (|predicted - true|) grouped by vendor.

    All bars are shown above zero (no distinction between over- and under-prediction).
    """

    # Prepare DataFrame
    results_df = pd.DataFrame({
        'vendor': y_planned,
        'true': y_true,
        'pred': y_pred
    })
    results_df['error'] = results_df['pred'] - results_df['true']
    results_df['abs_error'] = results_df['error'].abs()

    # Group by vendor (mean absolute error)
    vendor_summary = results_df.groupby('vendor')['abs_error'].mean().reset_index()

    # Sort vendors by magnitude of error (descending)
    vendor_summary = vendor_summary.sort_values('abs_error', ascending=False)

    # Plot
    plt.figure(figsize=(14, 6))
    plt.bar(
        x=vendor_summary['vendor'].astype(str),
        height=vendor_summary['abs_error'],
        color='orange'
    )
    plt.xlabel("Vendor")
    plt.ylabel("Mean Absolute Prediction Error")
    plt.title(f"Mean Absolute Error by Vendor - {plot_name}")
    plt.xticks(rotation=90)
    plt.grid(axis='y', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()


def plot_target_comparison_strip(self, y_true, y_planned, y_pred, plot_name="Cover time"):
    # Limit to first 20 samples for clarity
    n = min(20, y_true.shape[0])

    # Create a tidy dataframe
```

```
        df_plot = pd.DataFrame({
            'True': y_true[:n],
            'Predicted': y_pred[:n],
            'Planned': y_planned[:n]
        })
        df_plot['Sample'] = range(n)

        df_melted = df_plot.melt(id_vars='Sample', var_name='Type', value_name=plot_name)

        # Plot as strip
        plt.figure(figsize=(12, 6))
        sns.stripplot(data=df_melted, x='Sample', y=plot_name, hue='Type', jitter=True, dodge=True)
        plt.title(f"{plot_name} Comparison (First {n} samples)")
        plt.xlabel('Test set')
        plt.ylabel('Days')
        plt.legend()
        plt.tight_layout()
        plt.show()


    # === Helper Functions for Uncertainty Analysis ===

def get_rf_prediction_distribution(model, X):
    """Return a matrix of shape (n_samples, n_trees) with predictions from all trees."""
    X_np = X.values
    return np.stack([tree.predict(X_np) for tree in model.estimators_], axis=1)

def compute_prediction_uncertainty(pred_matrix):
    """Compute uncertainty as 90th percentile - 10th percentile for each row."""
    return np.percentile(pred_matrix, 90, axis=1) - np.percentile(pred_matrix, 10, axis=1)

def plot_uncertainty_by_supplier(uncertainties, supplier_ids, top_n=20, title="Prediction Uncertainty by Suppli
    df_uncertainty = pd.DataFrame({
        'Supplier': supplier_ids,
        'Uncertainty': uncertainties
    })

    # Group by supplier, take the mean
    grouped = df_uncertainty.groupby('Supplier')['Uncertainty'].mean().sort_values(ascending=False).head(top_n)

    # Plot
    plt.figure(figsize=(14, 6))
    bars = plt.bar(grouped.index.astype(str), grouped.values)
    sm = plt.cm.ScalarMappable(cmap='viridis', norm=plt.Normalize(grouped.min(), grouped.max()))
    colors = plt.cm.viridis((grouped.values - grouped.min()) / (grouped.max() - grouped.min()))
```

```
    for bar, color in zip(bars, colors):
        bar.set_color(color)

    plt.xticks(rotation=90)
    plt.ylabel("Mean Uncertainty (Working Days)")
    plt.xlabel("Supplier")
    plt.title(title)
    plt.tight_layout()
    plt.show()



# Initialize predictor
predictor = WarehouseLeadTimePredictor('df_convert3.csv')
predictor.models_to_use = [ModelType.RANDOM_FOREST, ModelType.LIGHTGBM, ModelType.XGBOOST]

# New Optuna parameters (replace the old grid search parameters)
predictor.use_optuna = True
predictor.n_trials = 100
predictor.cv_folds = 5
predictor.round_results = True

# Your existing parameters
predictor.handle_outliers = False


# Run the pipeline
results = predictor.run_full_pipeline()

# --- Supplier Model: Random Forest Prediction Uncertainty ---
supplier_rf_model = results['supplier_models'][ModelType.RANDOM_FOREST]
X_supplier_test = results['supplier_test_data']['features']

# Retrieve the original supplier IDs for test data
supplier_ids = predictor.df.loc[X_supplier_test.index, 'DimVendorKey']

# Get predictions from all trees
supplier_pred_matrix = get_rf_prediction_distribution(supplier_rf_model, X_supplier_test)

# Compute prediction uncertainty (90th - 10th percentile)
supplier_uncertainty = compute_prediction_uncertainty(supplier_pred_matrix)

# Plot uncertainty per supplier
plot_uncertainty_by_supplier(
    uncertainties=supplier_uncertainty,
```

```
    supplier_ids=supplier_ids,
    top_n=20,
    title="Supplier Model - Prediction Uncertainty by Supplier (Random Forest)"
)


# --- Warehouse Model: Random Forest Prediction Uncertainty ---
warehouse_rf_model = results['warehouse_models'][ModelType.RANDOM_FOREST]
X_warehouse_test = results['warehouse_test_data']['features']

# Retrieve the original supplier IDs (still 'DimVendorKey') for warehouse test data
warehouse_ids = predictor.df.loc[X_warehouse_test.index, 'DimVendorKey']

# Get predictions from all trees
warehouse_pred_matrix = get_rf_prediction_distribution(warehouse_rf_model, X_warehouse_test)

# Compute prediction uncertainty
warehouse_uncertainty = compute_prediction_uncertainty(warehouse_pred_matrix)

# Plot uncertainty per supplier
plot_uncertainty_by_supplier(
    uncertainties=warehouse_uncertainty,
    supplier_ids=warehouse_ids,
    top_n=20,
    title="Warehouse Model - Prediction Uncertainty by Supplier (Random Forest)"
)


# Save the best models
import joblib

# Save supplier model
best_supplier_model = results['supplier_models'][results['best_models']['supplier']]
joblib.dump(best_supplier_model, 'best_supplier_model.joblib')

# Save warehouse model
best_warehouse_model = results['warehouse_models'][results['best_models']['warehouse']]
joblib.dump(best_warehouse_model, 'best_warehouse_model.joblib')

# Save the predictor object for future use
joblib.dump(predictor, 'warehouse_predictor.joblib')

print("\nModels saved successfully!")

#********************************** END-CODE ****************************************
```