UPPSALA
UNIVERSITET

IT mIA 25 015

Degree project 30 credits

August, 2025

Applying Machine Learning to Demand Forecasting: A Case Study in Industrial Supply Chain Manangement

Julia Henningsson

Master's Programme in Industrial Analytics

UPPSALA
UNIVERSITET

Applying Machine Learning to Demand Forecasting: A Case Study in Industrial Supply Chain Manangement

Julia Henningsson

Abstract

The application of time series forecasting spans a wide range of domains, including weather, finance, energy, and healthcare. Regardless of the field, forecasting plays a vital role in optimizing resources and preparing for future challenges. One domain where forecasting is particularly critical is supply chain management, where it is frequently used for demand forecasting. Demand forecasting serves as the foundation for all supply chain planning and helps balance inventory levels, reduce costs, and meet customer expectations.

This thesis explores the application of machine learning to demand forecasting at a case company that has recently experienced greater fluctuations in demand and increasing variations in customer behavior. The limitations of the current approach have prompted the company to explore alternative methods, one of which being machine learning.

The thesis involves two experiments forecasting demand for two product families, A and B, using historical demand data. The experimental setup was designed to reflect the company's current forecasting conditions. The process included data collection, preprocessing, analysis, feature engineering, model development, and model evaluation. Three machine learning models were developed: Light Gradient Boosting Machine (LightGBM), Extreme Gradient Boosting (XGBoost), and Long Short-Term Memory (LSTM).

Model performance was evaluated using Mean Square Error (MAE), Root Mean Square Error (RMSE), and Symmetric Mean Absolute Percentage Error (SMAPE), and compared against a 12-month moving average baseline. The results show that LightGBM and XGBoost outperformed the baseline for Product Family A, while none of the models outperformed the baseline for Product Family B. The thesis concludes that further investigation and refinements are needed before the models can be deployed into production.

Faculty of Science and Technology

Uppsala University, Place of publication Uppsala

Supervisor: Toni Sigmundsson Subject reader: Lars-Henrik Eriksson

Examiner: Hans Karlsson & Olle Gällmo

Acknowledgement Writing this acknowledgement marks the final words of my thesis. And writing the final words of my thesis also signifies that my academic journey is approaching its end. I began my studies at Uppsala University in the fall of 2020, and during my time here, I have gained valuable knowledge, both academically and personally, for which I am deeply grateful.

I would like to express my sincere thanks to Frontit AB for the opportunity to write my thesis in collaboration with them. In particular, I would like to thank my supervisor Toni Sigmundsson. Thank you for your continuous support and dedication throughout this process. Your commitment has been truly appreciated. I would also like to thank the Frontit Uppsala team for being so welcoming and for showing genuine curiosity in my work.

I am also thankful to Süleyman Dag and Pär Olsson at CrossControl AB for their participation and for trusting me with the company's sensitive data. The opportunity to apply theoretical knowledge in a real-world context has been both insightful and challenging.

Lastly, I want to express my heartfelt gratitude to my dear friend Niclas Svensson. Thank you for your unwavering and multifaceted support. Academically, emotionally, and mentally. Not only during this thesis but throughout the past several years. Thank you.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACF Autocorrelation Function.

AI Artificial Intelligence.

ANNs Articial Neural Networks.

EFB Exclusive Feature Bundling.

EWM Exponentially Weighted Mean.

GBDTs Gradient Boosting Decision Trees.

GOSS Gradient-based One-Side Sampling.

LightGBM Light Gradient Boosting Machine.

LSTM Long Short-Term Memory.

MA Moving Average.

MAE Mean Square Error.

ML Machine Learning.

PACF Partial Autocorrelation Function.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

SCM Supply Chain Management.

SD Standard Deviation.

SMAPE Symmetric Mean Absolute Percentage Error.

XGBoost Extreme Gradient Boosting.

1 Introduction

This chapter introduces the subject of the thesis and outlines its scope. It includes background information, a description of the problem, an overview of the case company, the objective, research questions, and delimitations. The chapter concludes with an outline of the thesis structure.

1.1 Background

The desire to predict the future has long been fundamental to human nature. Anticipating threats and opportunities enabled early humans to make informed decisions critical for survival [33]. What was once a necessity for survival has evolved into a driving force behind many scientific and technological advancements. Humans have continuously tried to develop methods to anticipate outcomes and minimize uncertainty, from weather forecasting to financial market predictions. In modern times, Machine Learning (ML) and Artificial Intelligence (AI) have significantly improved predictive accuracy, allowing organizations to make data-driven decisions. Regardless of the field, forecasting is crucial in optimizing resources and preparing for future challenges. With the rapid increase of computational power and data availability, more advanced predictive models have been developed, offering organizations the ability to leverage their data to anticipate future needs with high credibility [38]. Although the landscape in predictive modeling is constantly changing, the goal remains unchanged: The better predictions, the better preparations.

One area where accurate forecasting is particularly critical is Supply Chain Management (SCM). SCM is the management of the flow of goods and services and includes all processes that transform raw materials into final products [40]. However, effective coordination of supply chain processes requires extensive planning. Procurement decisions depend on production levels, which in turn rely on anticipated delivery volumes, both of which are ultimately driven by demand.

All processes within a supply chain can be divided into two categories, push processes and pull processes. Push processes are defined as processes executed in anticipation of

predicted demand, whereas pull processes are defined as processes executed in response to actual demand. For push processes, managers must plan activity levels for operations such as procurement, production, transportation, or other scheduled activities. For pull processes, the focus is on preparing the capacity and inventory levels to be available. In both scenarios, the first step for a manager is to forecast customer demand accurately [7].

As a result, demand forecasting serves as the foundation for all supply chain planning. Demand forecasting holds a crucial role in SCM as it directly influences key operational and strategic decisions. Accurate prediction helps balance inventory levels, minimize costs and meet customer expectations. In contrast, poor accuracy can result in overproduction, stockouts, and increased holding costs, ultimately yielding financial losses and diminished customer satisfaction [37].

1.2 Problem Description

Traditional statistical forecasting methods, such as *ARIMA* and *Exponential Smoothing*, have long been used to predict future demand based on historical data. While effective in simpler contexts, these models often fall short in capturing the growing complexity of modern supply chains, where demand is shaped by numerous dynamic factors [12].

With the increasing availability of data and computational power, ML methods have emerged as a powerful alternative. ML models can handle large datasets, uncover complex patterns, and adapt to changing demand conditions in ways that traditional models cannot. By leveraging ML-based forecasting techniques, businesses can improve predictive accuracy, improve decision making, and achieve greater resilience in supply chain operations [32]. This shift highlights the growing necessity of integrating data-driven approaches in demand forecasting.

Currently, the demand forecasting process in the case company is not data-driven. Demand planners do not utilize any established technical tools for forecasting but instead rely on a combination of intuition, experience, communication with customers, and Excel calculations. Recently, the company has experienced greater fluctuations in demand

and increasing variations in customer behavior. As demand patterns have become more complex, forecasting has become increasingly challenging. The limitations of the current approach have prompted the company to explore alternative methods, one of which is ML.

1.3 Case Company

This thesis is conducted in collaboration with the consultancy firm Frontit, with the case study situated at one of their partners, CrossControl. CrossControl is a Swedish business-to-business company that develops and manufacturers advanced electronic solutions for industrial machines and vehicles. The company supports Original Equipment Manufacturers and System Suppliers in making industrial vehicles smarter, safer, and more productive. CrossControl offers a comprehensive platform for machine intelligence, comprising a wide portfolio of hardware, software, and services. Its customers are operating in end markets such as agriculture, construction, forestry, material handling, mining, cargo transport, and utility vehicles. CrossControl specializes in products designed to operate under demanding environmental conditions and applications with high reliability and quality requirements.

Since CrossControl's customers operate across a diverse range of end markets, its business is sensitive to a broad spectrum of external economic and industry specific factors. For example, an increase in forestry activity may boost demand, while a downturn in mining can lead to reduced orders. Exposure to multiple and sometimes opposing market dynamics complicates demand forecasting. Moreover, the effects of such external factors are often delayed, meaning that their impact on CrossControl's operations may not be immediately observable. In addition, the company faces long procurement lead times for critical components, often requiring orders to be placed several months before their own product delivery. As a result, demand forecasting must be performed well in advance to inform purchasing decisions, which inherently increases uncertainty. The further into the future a forecast attempts to predict, the less certain and reliable it becomes.

1.4 Objective

The objective of the thesis is to explore how ML can be applied to demand forecasting in a case company. Various state of the art ML forecasting models will be developed and evaluated. Since there is no established forecasting method at the case company, historical forecasts are not available for comparison. Therefore, the performance of the developed ML models will be evaluated against a baseline model.

1.5 Research Questions

To achieve the objective, this thesis will focus on addressing the following research questions:

1. How does the performance of ML models compare to that of a baseline model in forecasting the demand for Product Family A?
2. How does the performance of ML models compare to that of a baseline model in forecasting the demand for Product Family B?

1.6 Delimitations

The raw data provided by CrossControl is confidential and will therefore not be disclosed in this thesis. Real names of product families will be anonymized, referred to as ”Product Family A” and ”Product Family B”. Furthermore, since all ML models will be trained exclusively on data from the case company, the generalizability of the findings will be limited to this context.

1.7 Outline

This thesis is structured to follow a logical progression from the initial problem formulation to the final conclusions. Chapter 1 introduces the reader to the topic, time series forecasting using ML, and defines the scope by outlining the problem statement, objective, research questions, and delimitations. Chapter 2 lays the theoretical foundation by introducing key concepts and terminology related to the topic. Additionally, this chapter presents the ML models developed in the thesis. Chapter 3 reviews related work, emphasizing the shift from traditional statistical methods to ML models in demand forecasting. Chapter 4 details the methodology, including the experimental design and the approach to address the research questions. Chapter 5 describes the experimental setup, and Chapter 6 presents the results. Chapter 7 provides an analysis and discussion of the findings. Finally, Chapter 8 concludes the thesis with a summary of key insights, and Chapter 9 offers recommendations for future research.

2 Theory

This chapter presents the theoretical foundation relevant to the thesis. It covers key concepts related to time series forecasting, as well as the application of ML in this context. The aim is to equip the reader with the necessary background to understand the methodological choices made in the thesis.

2.1 Time Series Forecasting

Time series forecasting is the practice of predicting future values of a time series using the past values and/or other related variables [27]. One example of time series forecasting is estimating next month's demand using data from the past five years.

2.2 Time Series Data

Most forecasting problems involve the use of time series data. Time series data refers to a chronologically ordered set of observations that track variations in a specific variable over time [34]. These observations are recorded at consistent, evenly spaced time intervals, a characteristic known as granularity. The granularity level defines the frequency at which data points are recorded and determines the time interval between consecutive observations in a time series [20]. The appropriate level of granularity is dictated by the nature of the forecasting problem and can vary across different time scales. Common examples of granularity levels include hourly, daily, weekly, and monthly intervals.

2.3 Univariate and Multivariate Time Series Data

Time series data can be either univariate or multivariate. The distinction between them lies in the number of variables being tracked over time. A univariate time series consists of a single time-dependent variable, whereas multivariate times series includes two or more time-dependent variables [18].

For this thesis, a univariate time series would only consider past demand to predict future demand. Here, the model relies on patterns like trend, seasonality, and cycles within the historical demand itself.

2.4 Forecast Horizon

The forecast horizon represents the time into the future for which predictions are required. Its length is determined by the nature of the problem and the operational constraints of the system. For example, in production planning, a demand forecast for a specific product may need to be generated on a monthly basis. Given the time required to adjust production schedules, secure raw materials and components from suppliers, and coordinate the distribution of finished products to customers or storage facilities, a forecast horizon of three months may be necessary [34].

2.5 Forecast Interval

The forecast interval refers to the frequency at which new forecasts are generated. Using the production planning example again, demand may be forecasted on a monthly basis for a forecast horizon extending up to three months. In this case, a new forecast is prepared each month, making the forecast interval one month. When the forecast horizon remains set at T time steps and the forecast is revised at each time step, an adaptive approach known as *Rolling* or *Moving Horizon* is employed. Using this method, forecasts for $T - 1$ time steps within the horizon are updated, and a new forecast is computed for the most recent time step, T. The rolling horizon approach ensures that the prediction remains dynamic and responsive to new data, making it widely used when forecast horizons extend over multiple time steps [34].

2.6 Single Step and Multistep Predictions

Time series forecasting can be performed using either a single step or a multistep approach. Single step forecasting predicts the value at the next time step immediately

following the last observed data point. In contrast, multistep forecasting estimates values n time steps into the future [20].

Several approaches exist for multistep forecasting. One method is *Direct Multistep Forecasting*, which involves training a separate model for each forecasting time step [20]. For example, when predicting demand for the next two months and assuming monthly granularity, one model would be trained to forecast demand for the first month, and a separate model would be trained to predict demand for the second month.

Another approach is *Recursive Multistep Forecasting*, where a single model is used iteratively. The model predicts the next time step, and that prediction is then fed back as input to generate subsequent forecasts [20]. For instance, a single step forecasting model would first predict demand for the next month, and this forecast would then serve as input to estimate demand for the second month.

A third approach, *Direct Recursive Hybrid Forecasting*, combines elements of both prior methods. Distinct models are trained for each future time step, but each model incorporates predictions from prior time steps as input [20]. For example, when forecasting demand for two months, two models are built, and the output from the first model is used as input for the second model.

Finally, *Multiple Output Forecast* uses a single model capable of predicting the entire sequence at once. Instead of generating forecasts step by step, the model is trained to predict multiple future time steps in a single computation [20]. For instance, a single model could be developed to estimate demand for both the first and second month simultaneously.

Approaches that rely on previous forecasts to generate subsequent predictions are prone to error accumulation. Small errors early in the process can accumulate and significantly impact the accuracy of later predictions, especially when n is large. In contrast, the multiple output forecasting method is not prone to error accumulation. However, models following this approach have greater difficulty learning complex patterns [20].

2.7 Time Series Cross Validation

Cross validation is a widely used model evaluation technique in ML for assessing how well a model generalizes to data it has not seen during training. The primary goal is to prevent overfitting and obtain a more reliable estimate of the model's performance [11]. Overfitting occurs when a ML model becomes too closely tailored to training data and loses its ability to generalize. The model learns not only the underlying patterns but also the noise. As a result, the model will achieve good performance on training data, but poor performance on new and unseen test data [17]. Cross validation simulates multiple rounds of training and testing using the same dataset. This is achieved by partitioning the dataset into several subsets, or folds, which is why the method is commonly referred to as *K-fold Cross Validation* [11].

"Cross-validation is a widely used model evaluation technique in ML for assessing how well a model generalizes to data it has not seen during training. Its primary goal is to prevent overfitting and provide a more reliable estimate of the model's performance. Overfitting occurs when a model becomes too closely tailored to the training data, learning not only the underlying patterns but also the noise. As a result, the model performs well on the training set but poorly on new, unseen data. Cross-validation simulates multiple rounds of training and testing using the same dataset. This is done by partitioning the data into several subsets, or folds, which is why the method is commonly referred to as K-fold Cross-Validation [11]."

Figure 1 illustrates the concept using 4-fold cross validation ($K = 4$). In this case, the data is randomly divided into four folds of equal size. Since there are four folds, the process involves four iterations. In each iteration, one fold is used as the test set while the remaining three serve as the training set. The model is trained and evaluated in each iteration, and the performance metric is recorded. After all four folds have been used as the test set once, the cross validation process ends, and the results are averaged to produce a reliable performance estimate.

However, standard K-fold cross validation is not suitable for time series data. Unlike independent and identically distributed data, time series observations are temporally ordered, which must be preserved during training and testing. K-fold cross validation

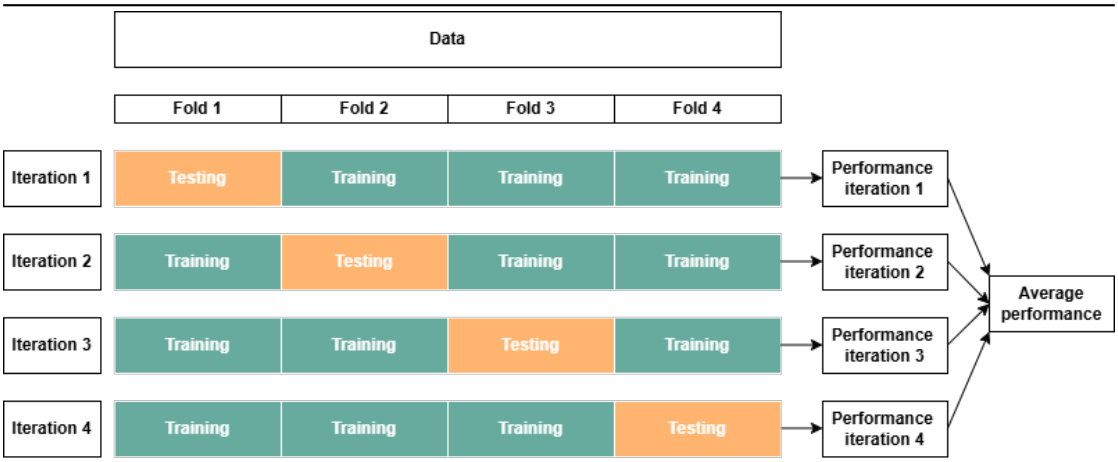


Figure 1 Illustration of K-fold cross validation ($K = 4$).

randomly shuffles the data, which breaks temporal dependencies and can lead to data leakage, where future values are used to predict past values [11].

Therefore, standard K-fold cross validation is incompatible with time series data. Instead, several alternative cross validation strategies have been developed specifically for time series. One such method is known as the *Rolling Forecast Origin*, also referred to as *Expanding Window Cross Validation*. This thesis adopts this approach, and from this point onward, it will be referred to as *Time Series Cross Validation*.

Figure 2 illustrates the concept using time series cross validation. As shown in the figure, this approach simulates multiple rounds of training and testing, similar to K-fold cross validation, but while preserving the temporal order of the data. The training set expands with each iteration, and the process continues until the entire dataset has been utilized.

2.8 Approaches to Time Series Forecasting

Time series forecasting can be performed using either a *Local* or a *Global* approach, depending on how models are trained across multiple time series. In the local approach, a separate model is trained for each individual time series and as a result, each model learns patterns specific to its own series. In contrast, the global approach involves train-

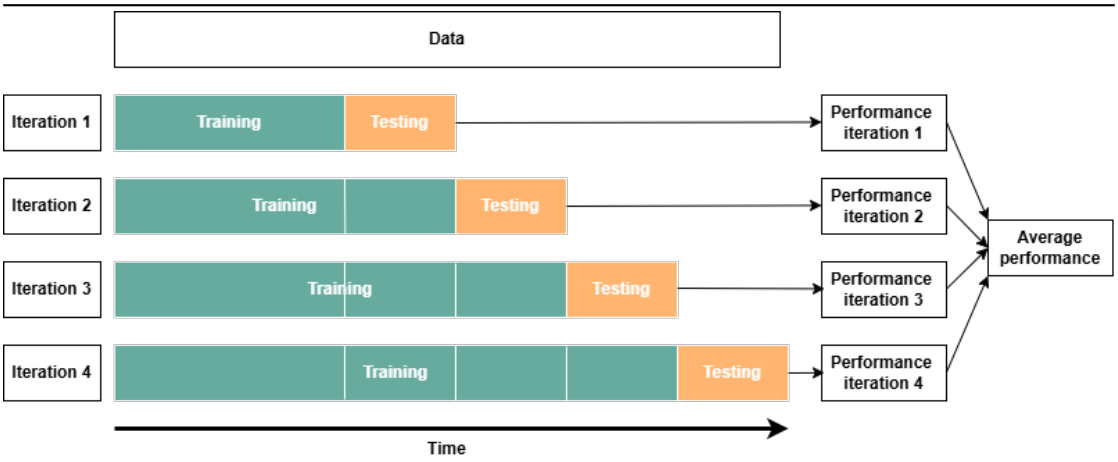


Figure 2 Illustration of 4-fold times series cross validation.

ing a single model on all time series simultaneously [39]. This allows the model to learn common patterns shared across series while also distinguishing between them using metadata such as unique time series identifiers. Consequently, a global model can utilize information from all available time series to generate forecasts for individual time series. This thesis exclusively adopts the local approach.

For example, consider the task of forecasting demand per product for a company. A local approach would involve training one model per product, while a global approach would train a single model using demand data from all products collectively. This example is illustrated in Figure 3.

Local Approach

Product A	
Date	Demand
2020-01-01	1
2020-02-01	2
2020-03-01	3

Product B	
Date	Demand
2020-01-01	2
2020-02-01	4
2020-03-01	3

Global Approach

Product A and B		
Series ID	Date	Demand
A	2020-01-01	1
B	2020-01-01	2
A	2020-02-01	2
B	2020-02-01	4
A	2020-03-01	3
B	2020-03-01	3

Figure 3 Illustration of a local and global approach to time series forecasting.

2.9 Decision Trees

The first two models developed in this thesis are Light Gradient Boosting Machine (LightGBM) and Extreme Gradient Boosting (XGBoost), both of which are implementations of Gradient Boosting Decision Trees (GBDTs). A decision tree is a supervised learning model used for both classification and regression tasks. Supervised learning is a type of ML in which the model is trained on labeled data, meaning that both the input features and the corresponding output values are known. The goal of the training is for the model to learn the relationship between inputs and outputs, allowing it to make accurate predictions on new, unseen data. A classification task involves predicting a discrete category, such as determining whether demand will increase, decrease, or remain stable. Regression tasks on the other hand, involves predicting continuous numerical values [21]. As in the context of this thesis, a regression task could involve predicting the expected quantity of product demand for the upcoming months. Before delving into the technical details of GBDTs it is useful to first understand how decision tree models make predictions and how they learn from data.

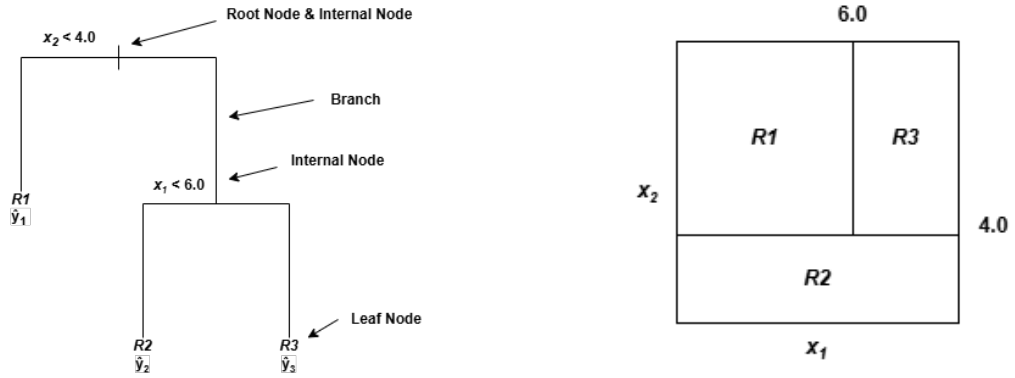
2.9.1 Making Predictions With Decision Trees

Decision trees are often referred to as rule-based models because they rely on a series of if-then conditions to define their structure. These rules are organized in a graph structure known as a binary decision tree (see Figure 4a). The decision tree recursively divides the input space into multiple non-overlapping regions (see Figure 4b), where each region corresponds to a leaf node. In each region, a constant value is used for the prediction [21].

Consider a case with two numerical input features $x = [x_1 \ x_2]^T$ and an output variable y. The task is to use an existing decision tree to predict $\hat{y}(x_*)$, where x_* represents a test input and $\hat{y}(x_*)$ denotes the predicted output corresponding to that test input.

The structure and rules defining the decision tree are illustrated in Figure 4. To generate a prediction for the test input $[x_{*1} \ x_{*2}]^T$, the process begins at the top of the tree, known as the root node. At each internal node, a condition is evaluated. If the condition is true,

meaning if $x_{*2} < 4.0$, the path continues down the left branch. Consequently, if the condition is false, the path follows the right branch. If the path reaches another internal node, the condition associated with that node is checked, and the left or right branch is selected accordingly. The procedure is repeated until the the path terminates at a leaf node, which assigns a constant prediction value. In classification tasks, the prediction is categorical and typically corresponds to the majority class among the samples in the leaf node. In regression tasks, the prediction is numerical and usually represents the average of the target values from the training data points assigned to the leaf node.



(a) A binary decision tree with two internal nodes (including the root) and three leaf nodes.

(b) Corresponding region partitioning of the two-dimensional input space *R*.

Figure 4 Illustration of a binary decision tree alongside its corresponding partitioning of input space. Each region (*R1, R2, R3*) corresponds to a leaf node, and the boundaries between regions reflect the split conditions applied at the internal nodes of the tree.

2.9.2 Learning With Decision Trees

The learning (or training) of a decision tree involves recursively dividing the input space into smaller, more homogeneous regions (denoted as R1, R2, and R3 in Figure 4b). The goal is to find splits that group similar outcomes together based on the the target variable. Recursive splitting means that decision rules are determined sequentially, one at a time. The process begins at the root node and proceeds from the top down, with each split selected without knowledge of how the full tree will eventually look. The learning process starts with the entire training dataset and the full input space considered as a

single region [21].

When determining the first split at the root node (e.g., $x_{*2} < 4.0$), the objective is to find the decision rule that best separates the training data based on the target variable. Once the split is selected, it is fixed, and the same procedure is applied recursively to each resulting subregion. At every step in the learning process, the algorithm evaluates all possible features and thresholds. Each candidate split is assessed according to how well it separates the training data according to a task-specific criterion defined by the user. These criteria differ between classification and regression, and multiple options exist within each category [21].

The final result is a tree structure where each internal node represents a decision rule, each branch corresponds to a path taken based on that rule, and each leaf node represents a final prediction. Conceptually, the decision tree has learned to approximate the relationship between inputs and outputs by dividing the input space into smaller regions and assigning each region a fixed output based on the training data it contains.

2.9.3 Gradient Boosting Decision Trees

As previously mentioned, LightGBM and XGBoost are implementations of GBDTs. Gradient boosting is an ensemble ML technique that combines a collection of weak learners into a single stronger learner. The weak learners are typically decision trees, and thus, models like LightGBM and XGBoost are commonly referred to as GBDTs. At its core, these models build an ensemble of models sequentially, where each new tree tries to correct the residuals (errors) of the previous tree. This process minimizes a loss function (defined by the user) using gradient descent, hence the name gradient boosting [19].

The boosting algorithm [30] starts by initializing a model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma) \tag{1}$$

Where $F_0(x)$ is a constant representing the initial model prediction before any trees are

added. $\arg\min_\gamma$ denotes the value of γ that minimizes the total loss. n is the number of training examples, and $\mathcal{L}(y_i, \gamma)$ is the loss function comparing the true target y_i with the prediction γ.

The following steps are repeated for M iterations, where M is the total number of trees in the model and m denotes the index of the current tree. After initializing the model with a constant value, the next step is to compute the residuals for each training sample i. This is done by taking a derivative of the loss $\mathcal{L}$ with respect to the previous prediction $F_{m-1}(x)$:

$$r_{im} = -\left[\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n \tag{2}$$

Multiplying the residuals r_{im} by -1 provides both the direction $(+/-)$ and the magnitude needed to minimize the loss function $\mathcal{L}$.

The next step is to train a regression tree using the features x as input and the residuals r as target. This results in a set of terminal node regions R_{jm} where $j = 1, ..., J_m$. Here, j denotes a terminal node (leaf) in the mth tree, and J represents the total number of terminal nodes in that tree.

The next step is to find the value γ_{jm} that minimizes the loss function $\mathcal{L}$ within each terminal node j:

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} \mathcal{L}(y_i, F_{m-1}(x_i) + \gamma) \quad \text{for } j = 1, \ldots, J_m \tag{3}$$

The term $\sum_{x_i \in R_{jm}} \mathcal{L}$ indicates that the loss is being aggregated over all samples x_i that fall within the terminal node region R_{jm}.

The final step updates the prediction of the combined model $F_m(x)$:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm}\, 1(x \in R_{jm}) \tag{4}$$

The term $\gamma_{jm}1(x \in R_{jm})$ indicates that the value γ_{jm} is selected if the input x falls within the terminal node region R_{jm}. Since all terminal nodes are exclusive, any given input x belongs to exactly one region R_{jm}. The corresponding value γ_{jm} is then added to the previous prediction $F_{m-1}(x)$, resulting in the updated prediction $F_m(x)$. The parameter $\nu \in [0, 1]$ is the learning rate. This parameter controls the extent to which the new tree's prediction γ should contribute to the update of the combined model $F_m(x)$.

After completing M iterations, the model is fully trained and can be used to make predictions on unseen data.

Implementations of GBDTs do not inherently capture temporal dependencies. Therefore, applying models like LightGBM or XGBoost to time series forecasting requires reframing the sequential data as a supervised learning problem. (Supervised learning is a type of ML in which the model is trained on labeled data, meaning that both the input features and the corresponding output values are known. The goal of the training is for the model to learn the relationship between inputs and outputs so it can make accurate predictions on new, unseen data.) Reframing time series data for supervised learning involves engineering features such as lags, date-based indicators, and rolling statistics. As the sequence of observations carries essential information, preserving temporal order when splitting the data into training and test sets is crucial. The ultimate goal of the transformation is to provide the model with features that encode time related patterns from the time series [8]

2.9.4 Light Gradient Boosting Machine (LightGBM)

LightGBM is an open-source implementation of GBDTs developed by Microsoft and released in 2017 [16]. LightGBM adheres to the fundamental principles of GBDTs, but distinguishes itself through a series of optimization focused design choices:

- **Gradient-based One-Side Sampling (GOSS)** Reduces the number of data training examples without sacrificing accuracy. Training examples with large gradients indicate poor model performance and contribute more to information gain. GOSS retain all such examples while randomly dropping examples with small gradients

[16].

- **Exclusive Feature Bundling (EFB):** Reduce the number of effective features without losing important information. Many datasets are sparse, with most feature values being zero, especially when one-hot encoding is used for categorical variables. For example, encoding weekdays results in only one nonzero feature per row. EFB bundles features that rarely have nonzero values at the same time into a single feature [16].
- **Histogram-based Decision Tree Learning:** While EFB groups mutually exclusive features into the same feature bundle, the histogram-based algorithm merges their values into one feature without losing track of their origin. This is achieved by assigning each original feature a distinct bin range within the combined feature. By adding an offset, overlapping value ranges can be prevented. For example, if feature A has values in $[0, 10]$ and feature B in $[0, 20]$, adding an offset of 10 to feature B shifts its range to $[10, 30]$. The two features can then be safely merged into one bundled feature with range $[0, 30]$, while still allowing the model to distinguish them during training [16].
- **Leaf-Wise Tree Growth:** Instead of growing trees level-wise as in many other implementations of GBDTs, LightGBM grows trees leaf-wise by choosing the split with the maximum loss reduction [16].
- **Empirical Performance:** The original paper reports that LightGBM speeds up the training process by more than 20 times compared to conventional implementations of GBDTs, without compromising accuracy. These findings are supported by experiments on multiple public datasets [16].

For a more detailed explanation of the optimization design choices in LightGBM, see the original paper [16].

2.9.5 Extreme Gradient Boosting (XGBoost)

XGBoost is another open-source implementation of GBDTs. It was developed by Tianqi Chen and Carlos Guestrin, and was formally introduced in 2016 [4]. Similarly to Light-

GBM, XGBoost follows the core principles of GBDTs, but emphasizes optimization and scalability. The original paper outlines the major contributions of their work as follows:

- **Scalable end-to-end boosting system:** XGBoost is a complete ML framework designed to handle a wide range of workloads. From small datasets on single laptops to massive datasets distributed across multiple machines [4].
- **Weighted Quantile Sketch:** Finding the best points to split features is key in building decision trees. XGBoost introduces a weighted quantile sketch algorithm that enables this process to be efficient, even when datasets are big or contain sample weights. These weights allow the model to treat some training examples as more important than others. This proposed algorithm estimates good split points without scanning every single individual value [4].
- **Sparsity-aware Algorithm:** Many datasets are sparse, meaning they contain many zero or missing values. XGBoost is designed to handle sparse data by learning the best default behavior for zeros entries while focusing on nonzero entries [4].
- **Cache-aware Block Structure for Out-of-Core learning:** Sometimes datasets are too big to fit into memory. To handle this, XGBoost uses a strategy that stores data in compressed blocks stored on disk. It also loads this type of dataset in a way that minimizes slow memory operations. These innovations enable training of powerful models on large datasets using standard machines with limited Random Access Memory [4].

For a more detailed explanation of each contribution, see the original paper introducing XGBoost [4].

2.10 Artificial Neural Networks (ANNs)

Articial Neural Networks (ANNs) are inspired by the structure and function of the human brain. In the brain, each biological neuron receives signals from many other neu-

rons through its dendrites. These signals are combined and, if they exceed a certain threshold, the neuron activates and sends an electrical impulse through its axon to other connected neurons. Similarly, in ANNs, each artificial neuron receives inputs, each associated with a weight representing the strength of the connection. The neuron computes a weighted sum of its inputs, applies a bias, and passes the result through an activation function, which determines whether and to what extent the neuron ”fires”. Just as in the brain, where neurons are organized into layers that process information hierarchically, artificial neurons are arranged in layers [41]. ANNs always have an input layer to receive raw data, one or more hidden layers to extract patterns and features, and an output layer to produce predictions, see Figure 5.

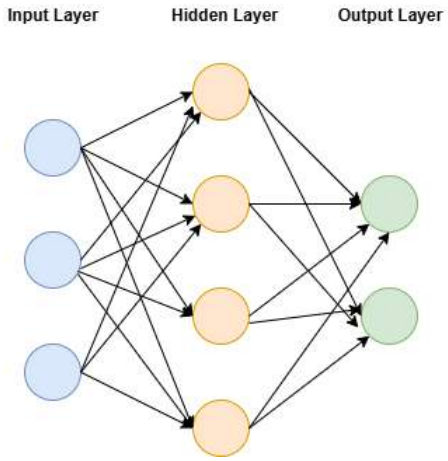


Figure 5 Conceptual architecture of an ANN with one input layer, one hidden layer, and one output layer.

2.11 Recurrent Neural Networks (RNNs)

Multiple types of ANNs exist, and most process each input independently, without retaining a state between inputs. This is suitable for many industrial tasks, such as image classification or object detection, where sequence memory is not required. However, some problems depend not only on the current input but also on previous inputs [41]. For example, when reading this sentence, the meaning emerges word by word while retaining memory of earlier words. Similarly, in demand forecasting, as in this thesis, past observations influence current predictions.

Biological intelligence processes information incrementally, maintaining an internal model built from past information and continuously updating it as new information arrives. A Recurrent Neural Network (RNN) follows the same principle in a simplified form. It processes sequences by iterating through their elements while maintaining a *state* that encodes what has been seen so far [6]. In effect, an RNN incorporates an internal loop that allows the network to retain memory, see Figure 6

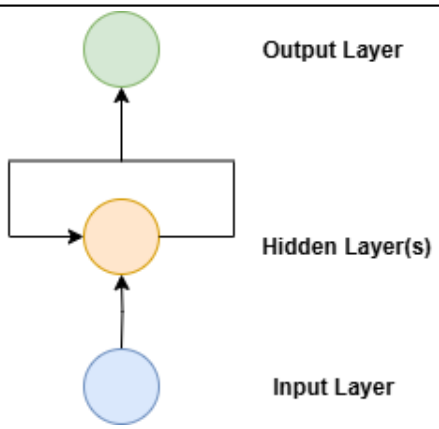


Figure 6 Conceptual architecture of an RNN showing the recurrent connection loop.

The state is reset between independent sequences, meaning each sequence is still treated as a single data point. However, during processing, the RNN is unfolded over time, handling the sequence element by element in multiple internal steps rather than in a single pass [6], see Figure 7.

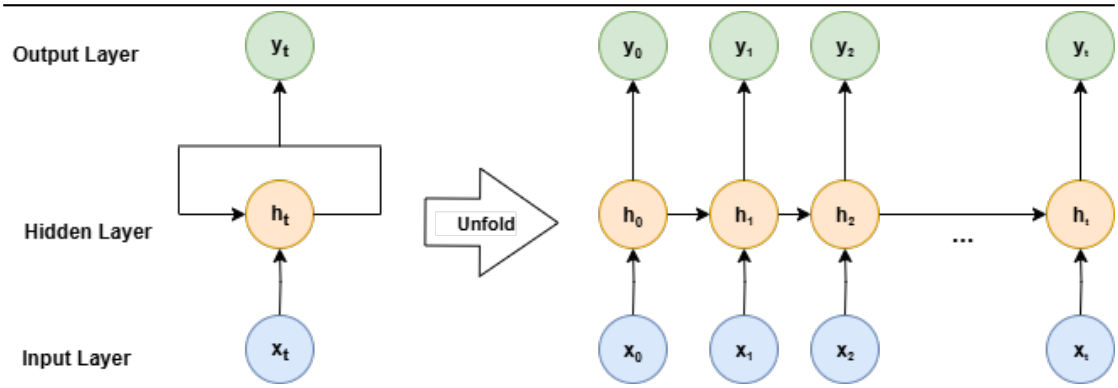


Figure 7 Illustration of how an RNN is unfolded over time. The network is unfolded into separate time steps t, with inputs x_t, hidden states h_t, and outputs y_t for each position in the sequence.

The unfolding process represents the RNN as a long chain of identical layers, one for each sequence element. *Backpropagation Through Time* is applied through this chain in reverse, from the last element back to the first. The algorithm computes how much each weight contributed to the total error and propagates this error signal backward through the unfolded network. Since the same weights are used every time step in the sequence, gradients from all steps are accumulated to update the shared parameters [6].

As gradients are multiplied repeatedly while moving backward through time, the signal can either diminish or grow uncontrollably. If it weakens at each step, it approaches zero by the time it reaches the earliest steps, causing the *vanishing gradient problem*. If it is amplified at each step, it can grow excessively large, leading to the *exploding gradient problem*, which causes unstable training and widely fluctuating weight updates.

2.11.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is the third and last model developed in this thesis. This method is a type of RNN and was originally introduced by Hochreiter and Schmidhuber in 1997 [13]. It was designed to overcome the vanishing and exploding gradient problems commonly encountered in standard RNNs during the training of long sequences. Hochreiter and Schmidhuber addressed this problem by introducing a new recurrent cell structure, called the LSTM cell. This cell incorporates an internal gating mechanism that enables the model to selectively retain and discard information over time [29].

The core idea behind LSTM is to separate and regulate two types of memory: the hidden state (short-term) and the cell state (long-term). LSTMs are structured around three gates: the forget gate, the input gate, and the output gate. These gates use matrix operations and activation functions to regulate the flow of information within the LSTM cell. By controlling this flow, each gate can retain, boost, or forget information from the sequence in both the hidden state and the cell state [29]. An LSTM cell is illustrated in Figure 8.

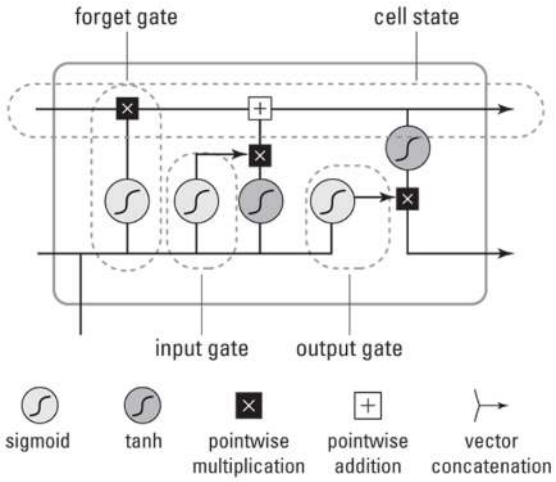


Figure 8 Visual representation of the internal structure of the LSTM cell. Illustrates the forget gate, input gate, output gate, and the cell state. Information flows from left to right [29].

Two types of activation functions are used within the LSTM cell: the *Sigmoid Function* (Equation 5) and the *Tanh Function* (Equation 6), each serving a distinct purpose. The tanh function normalizes values to the interval $[-1, 1]$, helping to keep them within a stable and manageable range. The sigmoid function scales values to the interval $[0, 1]$. Less important values are pushed toward zero and more important ones toward one. In essence, the sigmoid function allows the model to decide which information to retain and which to forget. These activation functions are defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6}$$

Each gate serves a specific role in controlling how the hidden state and the cell state are updated. Each gate includes two learnable weight matrices and a bias term. The resulting expression is then passed through a sigmoid activation function. The processing of information by each gate can be mathematically described by the following equations [15]:

First, let the input to the LSTM cell at time t be x_t and the hidden state from the previous time step be H_{t-1}.

The input gate determines how much information to retain from the current input x_t and the previous hidden state H_{t-1}. This is determined by the following equation:

$$F_t = \sigma(W_{xi} \cdot x_i + W_{hi} \cdot h_{t-1} + b_i) \tag{7}$$

The forget gate determines how much information to forget from the previous cell state C_{t-1}. This is determined by the following equation:

$$F_t = \sigma(W_{xf} \cdot x_i + W_{hf} \cdot h_{t-1} + b_f) \tag{8}$$

The output gate determines how much information to retain from the current cell state C_t to create the current hidden state H_t (the output of the cell). This is determined by the following equation:

$$O_t = \sigma(W_{xo} \cdot x_i + W_{ho} \cdot h_{t-1} + b_o) \tag{9}$$

In all gate equations, W_{xi}, W_{xf}, W_{xo}, W_{hi}, W_{hf}, and W_{ho} represent learnable weight parameters. Similarly, b_i, b_f, and b_o represent learnable bias parameters.

The three previously described gates work together to continuously update the current cell state C_t. The process begins by computing a candidate cell state $\tilde{C}_t$ using the tanh activation function:

$$\tilde{C}_t = \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c) \tag{10}$$

Where W_{xc} and W_{hc} are learnable weight parameters and b_c is a learnable bias parameter.

The current cell state is then updated by combining the candidate cell state $\tilde{C}_t$, the forget

gate F_t, and the input gate I_t:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{11}$$

Where $\odot$ denotes elementwise multiplication. The forget gate F_t determines how much information from the previous cell state C_{t-1} to retain. The input gate I_t determines how much information from the current candidate cell state $\tilde{C}_t$ to incorporate into the updated cell state C_t.

In the final step, the newly updated C_t from Equation 11 and the output gate O_t are used to update the current hidden state H_t.

$$H_t = O_t \odot tanh(C_t) \tag{12}$$

2.12 Hyperparameter Optimization

A ML model learns from data by adjusting its internal parameters during training. For example, in GBDTs such as LightGBM and XGBoost, internal parameters include the splits which determine which features are used for splitting and at what thresholds. In a LSTM network, internal parameters include the weights learned during training.

In contrast, hyperparameters are variables that control the learning process itself by affecting various aspects of the behavior of the algorithm. One of the most important hyperparameters in ML is the learning rate. The learning rate regulates how much the model's internal parameters are updated in response to the error during training. Hyperparameters are set before training begins, either manually or through an optimization process, commonly referred to as *tuning* [2].

Hyperparameter optimization is the process of systematically searching for the best combination of hyperparameter values to improve model performance. Hyperparameters have a significant impact on how well a model learns and generalizes to patterns from data, making their selection a crucial step in model development [2].

Multiple strategies exist for hyperparameter optimization. This thesis adopts the *GridSearchCV* approach. GrisSearchCV is a method that systematically searches through a predefined set of hyperparameter combinations to identify the configuration that yields the best performance.

3 Related work

This chapter examines how demand forecasting in SCM has evolved in response to technological advances and market uncertainty. It highlights the shift from traditional statistical models to ML, supported by findings from recent literature and forecasting competitions.

3.1 The Increasing Role of Demand Forecasting in SCM

Mediavillaa et. al [32] argues that the era of stable markets is history. Demand forecasting has long been a critical component of SCM. However, in today's fast-paced global economy, its importance has increased even more. Companies now operate in highly complex scenarios shaped by unpredictable factors such as monetary crisis, pandemics, climate change, and supply disruptions. In such unstable conditions, demand forecasting becomes both more challenging and important. Mediavilla et. al highlights that this scenario has driven the need for continuous improvement of forecasting methods in SCM. As a result, demand forecasting has emerged as an increasingly prominent research area in the field.

In another study, Aamer et.a al further underscore the rising importance of demand forecasting in SCM. Their paper presents a comprehensive literature review of ML application in demand forecasting. A total of 1870 papers published between 2010-2019 were initially retrieved from different academic databases. Following a systematic screening process, the number of papers was reduced to 77 for further analysis. The results showed that the number of publications fluctuated throughout the decade until 2017, after which a clear and strong upward trend emerged. Publication from 2018 and 2019 alone accounted for 44% of the total reviewed. Given the ongoing advancements in technology and increasing data availability, it is reasonable to assume that this trend has continued.

3.2 Insights From The Makidakis Competitions

Spyros Makridakis is one of the most influential figures in the field of time series forecasting. He is best known for organizing the renowned Makridakis Competitions (M1 [23], M2 [22], M3 [24], M4 [25], and M5 [26]), which began in 1979. These competitions benchmark forecasting methods in real-world settings and promote empirical evaluation over theoretical claims. Therefore, these competitions are widely regarded as milestones in forecasting research.

The M4 competition revealed that the forecasting accuracy of individual statistical or ML methods was relatively low. Instead, the best performing methods involved hybrid approaches and combinations of methods. Among the 17 most accurate submissions, 12 were combinations of primarily statistical models. Notably, the biggest surprise came from a hybrid method that integrated both statistical and ML features. This method achieved nearly 10% higher accuracy than the benchmark method [25].

The second most accurate submission was also a combination of statistical and ML methods. Moreover, six submitted methods were purely based on ML, and none of them outperformed the benchmark. However, M4 was still the first forecasting competition to show that two ML-based methods, one using RNNs and the other using XGBoost, were significantly more accurate than simple statistical approaches. These findings highlights a growing belief in the potential of ML in forecasting, although researchers also underline that ML remains in its infancy in this field [25].

By the time of the M5 competition in 2022, ML have made significant progress. All of the top performing methods were purely based on ML in this competition and outperformed all statistical benchmarks and their combinations. Most of these models were based on LightGBM. A key insight from the M5 competition was the observed critical importance of involving explanatory variables. In fact, all winning submissions leveraged external information (e.g., holidays, promotions, weather) to improve forecast accuracy.

3.3 Forecasting With Highly Fluctuating Demand Data

A study titled *An Optimized Model Using LSTM Network for Demand Forecasting* [1] investigates the application of ML techniques to demand forecasting in a furniture company, to develop an accurate forecasting model. The data used in the study shares similarities with the data used in this thesis, particularly in its highly fluctuating nature. The dataset in the study comprises the monthly sale quantity for a product (local approach) from 2007 to 2017 and forms a times series of 132 data points. The study explores recent deep learning methods available at the time (2020) and focuses on optimizing hyperparameters for an LSTM network. The proposed method, a stacked LSTM network, is compared to well-known forecasting techniques from both statistical and computational intelligence categories. The results showed that the proposed method outperformed all other evaluated methods. The findings underscore the potential of ML techniques to capture patterns in real-world contexts, while also highlighting the comparatively weaker performance of traditional statistical methods.

4 Methodology

This chapter outlines the methodology used to address the research questions in the thesis. It presents the research design and the pipeline for developing the ML models, including data collection, preprocessing, analysis, and feature engineering. The chapter then describes the model development process and evaluation methods. The chapter concludes with a discussion on validity and reliability.

4.1 Research Approach and Design

This thesis adopts a positivist epistemological approach. This philosophical approach to knowledge assumes that reality is objective and measurable. It relies on scientific methods to analyze data and aims to confirm or derive claims through observable and quantifiable results [3]. Moreover, given the focus on analyzing numerical data, the research design of this thesis is quantitative in nature.

This thesis focuses on time series forecasting, a scientific approach that uses historical data to make future predictions. The core idea of this practice is to build models that learn patterns from past data and apply them to forecast future values and support decision-making. The thesis aims to develop and compare the performance of three models: LightGBM, XGBoost, and LSTM, in the context of demand forecasting. Figure 9 shows the overall pipeline for developing these models. The problem formulation is detailed in Section 5.1 in the subsequent chapter.

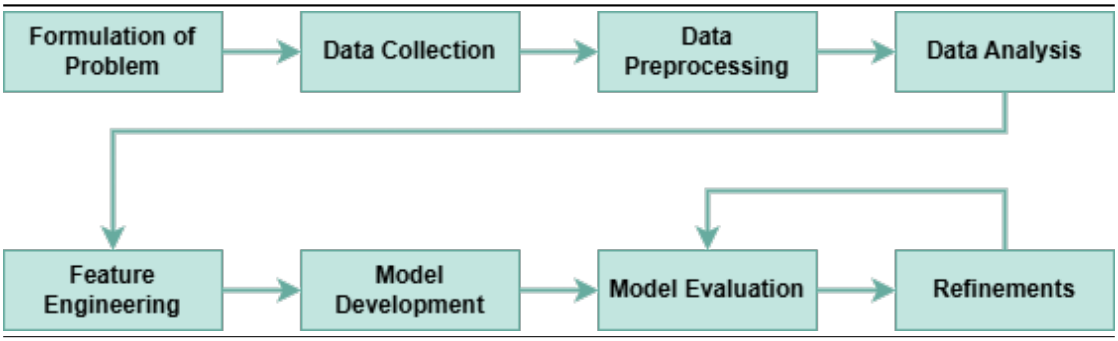


Figure 9 Pipeline for ML model development in this thesis.

4.2 Data Collection

Once the problem has been defined, the next step is to collect data. This process involves gathering raw data from one or more sources relevant to the problem. The raw data used in this thesis is sourced from the case company's billing log and is provided in Excel format. It is organized in tabular form, where each row represents a an order of a product, and each column contains a specific attribute related to that order. From this log, key information such as date, item number, and quantity is extracted to construct time series datasets for a Product Family A and for a Product Family B.

4.3 Data Preprocessing

Data preprocessing is a critical step in the ML pipeline illustrated in Figure 9. It is defined as the process of transforming raw data into into a clean, structured, and usable format for the preceding data analysis [9].

The goal of the data preprocessing in this thesis is to create two univariate time series datasets with monthly granularity: one for Product Family A and one for Product Family B. First, all rows corresponding to item numbers for families A and B, respectively, were selected. The rows in each dataset were then sorted chronologically and grouped by month. In some months, no data were recorded, indicating zero demand during those periods. To establish a continuous monthly time series dataset, missing months were explicitly added with their corresponding demand values set to zero.

In this thesis, the raw data is organized in tabular format, with each row representing a transaction and each column representing a specific attribute such as date, item number, or quantity.

4.4 Data Analysis

The data analysis consists of a general exploratory analysis and an autocorrelation analysis of the two established time series datasets. The overall aim of the data analysis is to

gain a deeper understanding of the data. The general analysis involves plotting the data to reveal its characteristics. Visualizing the data provides an overview of how demand fluctuates over time. Each time series is plotted alongside a 12-month Moving Average (MA) and a 12-month rolling Standard Deviation (SD). The 12-month MA smooths out short-term fluctuations and highlights the long-term trend (upward, downward, or stable) of demand. The 12-month rolling SD shows the stability of demand around the trend. A rising SD indicates that demand is becoming more unpredictable with larger fluctuations from month to month, while a declining SD suggests more predictable and stable behavior. Visualizations of the two datasets are shown in Figure 10 and Figure 13 in Chapter 5.

The second part of the analysis focuses on autocorrelation. Autocorrelation refers to the correlation between a time series and its lagged version over time. It captures the degree to which current observations are influenced by historical values. Identifying which past values are most predictive supports more informed and targeted feature engineering. To explore the autocorrelation within a time series dataset, Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots are commonly used as diagnostic tools. The ACF shows the overall correlation at multiple lags, including both direct and indirect effects, while the PACF isolates the direct influence of each lag by accounting for the effects of intervening lags. The conceptual difference can be illustrated with the following example. The ACF answers the question "How correlated is y_t with y_{t-3}, including any influence through y_{t-1} and y_{t-2}?", while PACF answers the question "How correlated is y_t with y_{t-3}, after removing the effects of y_{t-1} and y_{t-2}?". Please note that this example is intended solely to clarify the conceptual difference between ACF and PACF. It does not reflect the specific implementation used in the thesis.

4.5 Feature Engineering

The feature engineering process is guided by the findings of the data analysis and is defined as the process of creating new features from existing data [20]. This phase is especially important for models like LightGBM and XGBoost, which do not inherently capture temporal dependencies. In this thesis, feature engineering includes the creation

of the following types of features:

Time Based Features: Extracted from the date to introduce temporal context and capture seasonality patterns.

Lag Features: Demand values from previous time steps that provide the model with historical context. The selection of lag features is guided by the PACF plot and is chosen to ensure that the models has access to the most informative past values.

First Order Difference Features: The difference between the current and previous demand values. These features represent the rate of change in demand, enabling the model to detect whether demand is increasing, decreasing, or remaining stable. Their selection is informed by the PACF plot.

Second Order Difference Features: The change in the rate of change (i.e., acceleration or deceleration) of demand. These features help the model detect inflection points, such as the start of a trend or the end of a surge. Also guided by the PACF plot.

Minimum and Maximum Features: The minimum and maximum values observed within a rolling time window. A rolling time window is a fixed time interval that moves sequentially through the time series, one time step at a time. At each position, the calculations are applied to the data within that window. These features help the model identify recent peaks and valleys, incorporates range-awareness that can aid in mitigating overfitting. The choice of time interval is based on the ACF plot.

MA Features: The average demand over a rolling time window. These features highlight trends and the selection of time interval is guided by the ACF plot.

Exponentially Weighted Mean (EWM) Features: A smoothed representation of past demand with exponentially decreasing weights for older observations. Unlike simple MAs, which weigh all past observations equally, EWMs decays older values exponentially. The decay rate is controlled by a smoothing factor $\alpha \in [0, 1]$, where values close to 0 emphasize long-term trends and values near 1 emphasize short-term changes. The selection of time interval is guided by the ACF plot. In this thesis, EWM features are computed using α values of 0.5, 0.7, and 0.9.

Rolling SD Features: The SD in demand over a rolling time window. These features

capture variability in demand and offer insight into the volatility of recent demand patterns. The selection of time interval is guided by the ACF plot.

LSTM models inherently capture temporal dependencies, making many of the manually engineered features redundant. Therefore, all features except the time-based ones are excluded from the LSTM model. The input sequence for the LSTM model is guided by the ACF plots.

4.6 Model development

The LightGBM, XGBoost, and LSTM models were developed in a Python environment. Python is widely adopted for ML tasks due to its rich ecosystem of open-source libraries that support all stages in the ML pipeline. In this thesis, several Python libraries have been used, including NumPy [10], Pandas [31], Matplotlib [14], Scikit-learn [36], LightGBM [16], and XGBoost [4]. The LSTM model [13] was implemented using the Keras library [5], which is a high-level Application Programming Interface for building and training deep learning models. Keras was chosen for its ease of use, as it abstracts much of the underlying complexity while still allowing for customization. For example, when using Keras, manual implementation of backpropagation is not required, unlike in lower-level frameworks such as TensorFlow [28] or PyTorch [35].

After importing all necessary libraries, the structure of each model could be setup in accordance with the problem formulation outlined in Section 5.1. All models used GridSearchCV for hyperparameter optimization with time series cross validation. The pseudocode of LightGBM and XGBoost can be found in Appendix A and the pseudocode for LSTM can be found in Appendix B.

4.6.1 Hyperparameter Optimization LightGBM

LightGBM offers a wide range of hyperparameters, and the selection of which to tune depends on the use case. The hyperparameters of LightGBM deemed most important and relevant to this thesis are:

Number of Estimators: Controls the number of boosting rounds (trees) in the ensemble. If set too low, the model stops training before learning sufficient patterns, resulting in underfitting. If set too high, the model risks overfitting by continuing to train beyond the point of optimal generalization.

Number of Leaves: Controls the maximum number of leaf nodes per tree. If set too low, the trees may be too simple to capture complex patterns in the data, leading to underfitting. If set too high, the trees can become overly complex and risk memorizing noise from the training set, leading to overfitting.

Maximum Depth: Controls the maximum depth of each tree and limits how many splits can occur from root to leaf. If set too low, the trees may not grow deep enough to capture important splits, leading to underfitting. If set too high, the trees may fit the training data too closely, including noise, which increases the risk of overfitting.

Learning Rate: Controls how much each tree contributes to the final prediction. Higher values speeds up learning but increases the risk of overshooting the optimal solution. Lower values slow down training, but often lead to better generalization.

4.6.2 Hyperparameter Optimization XGBoost

Similarly to LightGBM, XGBoost also offers a wide range of hyperparameters, and the choice of which to tune depends on the use case. The hyperparameters of XGBoost deemed most important and relevant to this thesis are:

Number of Estimators: Same significance in XGBoost as in LightGBM.

Minimum Child Weight: Controls the minimum number of samples required in each leaf node. This hyperparameter ensures that splits are only made when there is sufficient data to support them. A lower value allows more frequent splitting, which can help capture fine-grained patterns but also increases the risk of overfitting. A higher value restricts splitting, resulting in shallower trees and a bigger risk of underfitting by missing important patterns.

Maximum Depth: Same significance in XGBoost as in LightGBM.

Learning Rate: Same significance in XGBoost as in LightGBM.

4.6.3 Hyperparameter Optimization LSTM

The hyperparameter optimization for the LSTM model revolves around the same hyperparameters as for a normal RNN. The hyperparameters most important and relevant to optimize in this thesis are:

Input Sequence: Controls the number of time steps fed into the model and corresponds to the lag features engineered for the LightGBM and XGBoost models. Its selection is informed by the ACF and PACF plots from the data analysis.

Hidden Layers: Controls the depth of the network and refers to the number of stacked LSTM layers within the model. More layers allow the model to learn more complex temporal patterns, but come with a greater risk of overfitting and longer training times.

Neurons per Layer: Controls the dimensionality of the hidden state vector in each LSTM layer. Each neuron in an LSTM layer captures some aspect of temporal dynamics in the input sequence. Thus, increasing the number of neurons increases the model's representational capacity. However, higher capacity also raises the risk of overfitting, as the model may start learning noise rather than meaningful patterns.

Dropout Rate: Controls the proportion of neurons that are randomly deactivated during each training iteration. This regularization technique helps prevent the model from relying too heavily on specific neurons. Instead, it encourages the model to learn more robust and generalizable patterns.

Learning Rate: Controls the size of weight updates during each iteration and determines how quickly the model learns from the data. A too high learning rate can cause the model to overshoot optimal weights and fail to occur, while a learning rate that is too small can lead to slow training and convergence to suboptimal solutions.

Batch Size: Controls the number of training sequences processed in parallel during each training iteration. The model computes gradients based on all sequences in the batch before updating the weights. With a small batch size, each weight update is based

on very few samples. In contrast, a larger batch size produces more stable updates.

Epoch Size: Controls the number of complete passes through the entire training dataset during model training. One epoch means the model has seen each training sample once. Training over multiple epochs allows the model to gradually improve by repeatedly updating its weights. However, using too many epochs increases the risk of overfitting, as the model may begin to memorize the training data rather than generalize to unseen data.

4.7 Model Evaluation

Model evaluation is crucial for estimating the performance of the developed forecasting models. As performance is typically assessed based on predictive accuracy, it is essential to use error metrics that quantify this. These metrics offer insight into model performance and guide the selection of the best performing model. Ideally, the developed models would be compared against an existing method used by the case company. However, since this is not possible, performance will instead be evaluated against a 12-month MA, which serves as a baseline model.

In this thesis, the performance of the developed forecasting models and the 12-month MA will be evaluated using three widely used error metrics in the field of time series forecasting and ML: Mean Square Error (MAE), Root Mean Square Error (RMSE), and Symmetric Mean Absolute Percentage Error (SMAPE).

4.7.1 Mean Absolute Error (MAE)

MAE calculates the average absolute difference between the forecasting values and the true values. It provides a clear measure of the magnitude of the errors without considering its direction. It is expressed in the same unit as the target variable, making it easy to interpret. MAE is defined by the following Equation:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{13}$$

Where y_t is the true value at time point t, $\hat{y}_t$ is the forecasting value at time point t, and n is the total number of observations.

4.7.2 Root Mean Square Error

RMSE calculates the square root of the average of the squared differences between the forecasting values and the true values. It is particularly useful when large errors are undesirable, as the squaring operation penalizes larger errors more heavily than smaller ones. Similar to MAE, RMSE is expressed in the same unit as the target variable, making it easy to interpret. RMSE is defined by the following Equation:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{14}$$

Where y_t is the true value at time point t, $\hat{y}_t$ is the forecasting value at time point t, and n is the total number of observations.

4.7.3 Symmetric Mean Absolute Percentage Error

SMAPE calculates the percentage difference between the forecasting values and the true values. This error metric treats under-forecasts and over-forecast equally, which is why it is referred to as symmetric. Expressing the error as a percentage aids interpretability in a business context, making SMAPE a valuable complement to MAE and RMSE. SMAPE is defined by the following Equation:

$$SMAPE = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{(|y_t| + |\hat{y}_t|)/2} \tag{15}$$

Where y_t is the true value at time point t, $\hat{y}_t$ is the forecasting value at time point t, and n is the total number of observations.

5 Experimental Setup

This chapter presents the experimental setup of the thesis. It includes the problem formulation, the results from the data analysis and feature engineering process, as well as the parameter grids for the hyperparameter optimization.

5.1 Formulation of Problem

As outlined in Section 1.3, the case company is subject to supply chain constraints due to long component lead times. Consequently, material procurement must be planned well in advance. To align this thesis with the company's operational realities, a simulated forecasting scenario was established in which product demand is predicted one year ahead with monthly granularity. The forecasting procedure will generate predictions for each month of the year 2024. Hence, the year 2024 serves as the test set.

For instance, when forecasting demand for January 2024, only data available up to and including January 2023 is used. This restriction necessitates the use of a multistep forecasting strategy with $n = 12$ time steps, yielding a forecast horizon of 12 months. As detailed in Section 2.6, multiple strategies exist for multistep forecasting. However, this thesis adopts the recursive multistep forecasting approach, wherein a one-step-ahead model is iteratively applied to generate forecasts for subsequent time steps. This procedure is then repeated for each subsequent month in 2024. Forecasting demand for February 2024 involves training on data up to and including February 2023 and applying the same recursive multistep process with $n = 12$ recursive time steps.

As described in Section 2.5, this setup results in a forecast interval of one month, as the model is retrained and re-evaluated monthly using an expanding training window. A rolling forecast horizon is also employed, wherein the forecast start point advances by one month for each iteration.

This thesis conducts one experiment for a Product Family A and one experiment for Product Family B using a local approach. As outlined in Section 2.8, this means separate models are developed for forecasting of each product family.

In summary, the forecasting framework employed in this thesis consists of a 12-month forecast horizon, a recursive multistep prediction strategy with $n = 12$ time steps, a one month forecast interval, an expanding training window, and a rolling forecast horizon. This forecasting framework is employed to each product family using a local approach.

5.2 General Data Analysis Product Family A

The result of plotting the demand of Product Family A, the 12-month MA, and the 12-month rolling SD is shown in Figure 10.

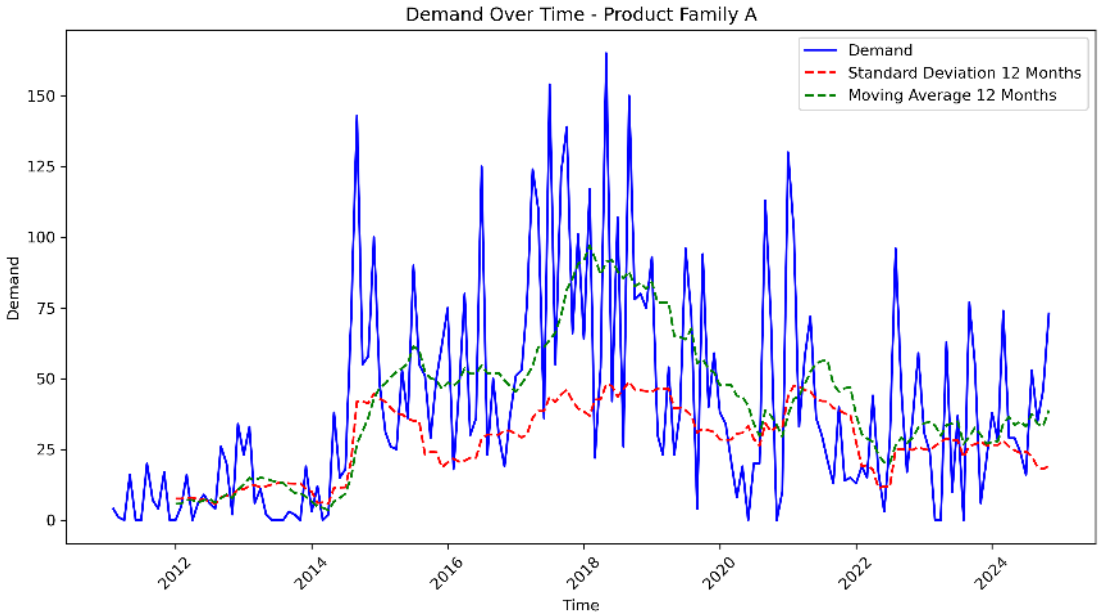


Figure 10 Demand over time of Product Family A

The demand (blue line) is characterized by high volatility with strong month-to-month fluctuations with several sharp peaks and valleys. These sudden shifts in trend make the forecasting task particularly challenging [1]. Notably, there are several major spikes around 2017-2019 during which demand surged sharply. After 2020, a decline in overall demand is evident, and from 2021 onward, both the magnitude and frequency of large demand spikes decrease noticeably.

Observing the 12-month MA (green line), a steady upward trend is visible until 2019, after which the long-term demand begins to decline. In more recent years, particularly from 2022 onward, the trend flattens out, suggesting that demand has stabilized at a lower level.

Observing the 12-month rolling SD (red line), there is a sharp spike around 2014, after which the SD fluctuates considerably. However, similarly to the MA, from around 2022 onward, the curve flattens out, and the level of predictability remains relatively stable.

5.3 Autocorrelation Analysis Product Family A

Figure 11 shows the ACF plot for Product Family A using a 95% confidence interval.

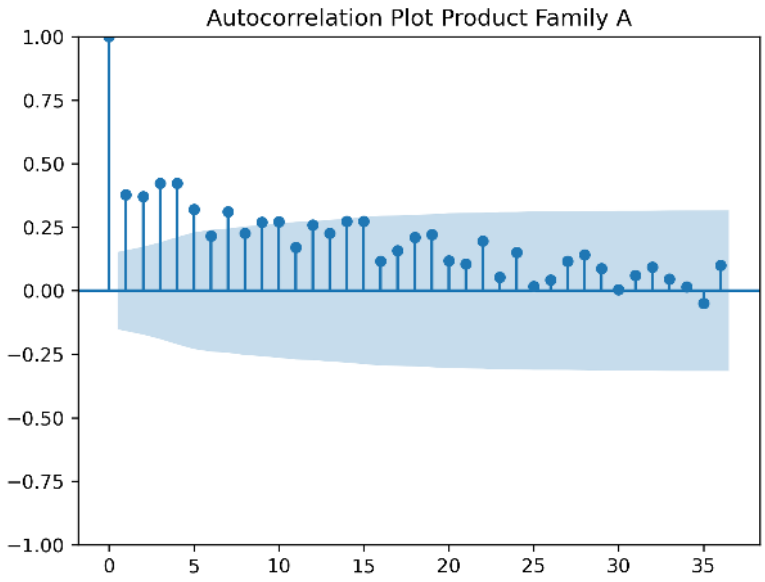


Figure 11 ACF plot of Product Family A

An autocorrelation of +1 represents a perfect positive relationship, and an autocorrelation of -1 represents a perfect negative relationship. A time series is always perfectly autocorrelated with itself at lag 0. Any spike outside the shaded region is statistically significant at the 5% level, whereas a spike inside the shaded region implies no meaningful autocorrelation. Looking at Figure 11, we can observe meaningful autocorrelation

at lag 1, 2, 3, 4, 5, 7, 9, 10, 12, 14, and 15.

Figure 12 shows the PACF plot for Product Family A using a 95% confidence interval.

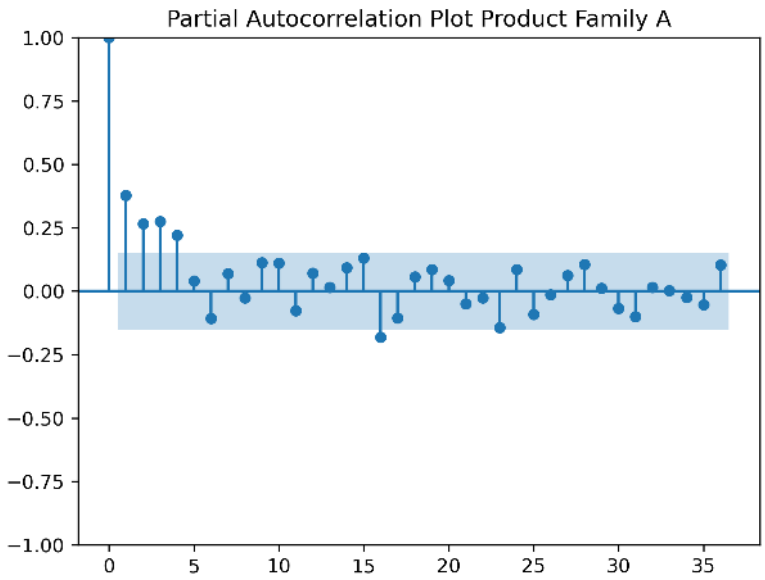


Figure 12 PACF plot of Product Family A

Looking at Figure 12, we can observe the lags with a meaningful direct autocorrelation. These lags are lag 1, 2, 3, and 4.

5.4 General Data Analysis Product Family B

The result of plotting the demand of Product Family B, the 12-month MA, and the 12-month rolling SD can be shown in Figure 13.

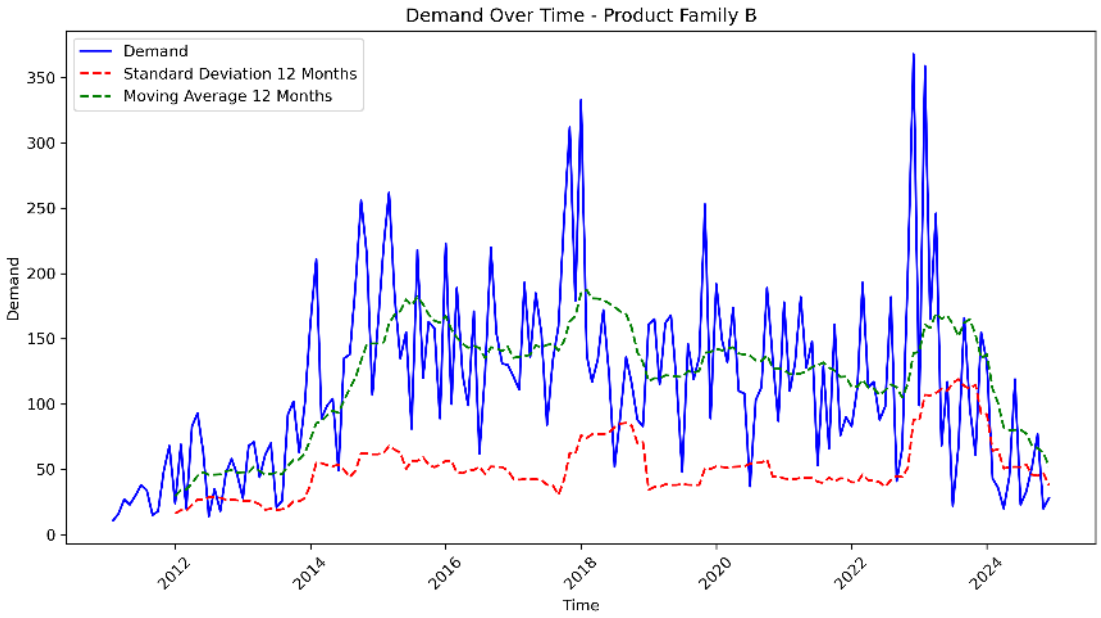


Figure 13 Demand over time of Product Family B

Similar to Product Family A, the demand (blue line) for Product Family B is characterized by high volatility, with strong month-to-month-fluctuations. Figure 13 reveals several extreme peaks, particularly around 2018 and 2023. Following the final peak, demand declines rapidly.

The 12-month MA (green line) shows a steady increase in long-term demand until 2016. Between 2016 and 2023, no clear long-term trend is evident, although fluctuations remain pronounced. From 2023 onward, a strong downward trend is observed.

The 12-month rolling SD (red line) remains relatively stable throughout most of the series. However, noticeable increases are observed in connection with the extreme peaks in demand, indicating that demand became more unpredictable during these periods.

5.5 Autocorrelation Analysis Product Family B

Figure 11 shows the ACF plot for Product Family A using a 95% confidence interval.

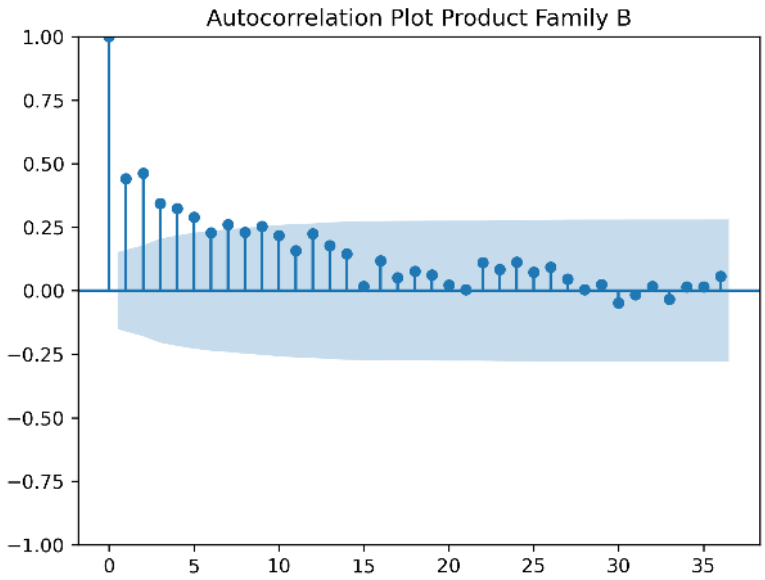


Figure 14 ACF plot of Product Family B

Looking at Figure 14, we can observe meaningful autocorrelation at lag 1, 2, 3, 4, 5, 7 and 9.

Figure 15 shows the PACF plot for Product Family B using a 95% confidence interval.

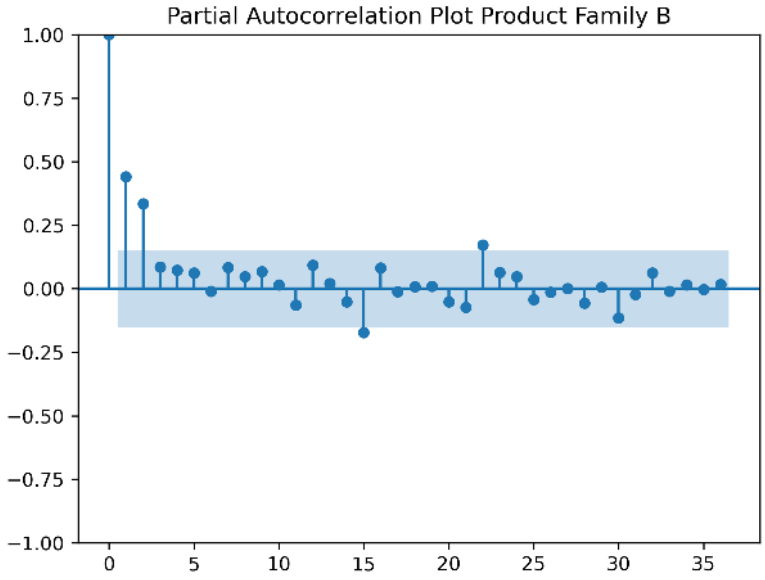


Figure 15 PACF plot of Product Family B

Looking at Figure 15, we can observe the lags with a meaningful direct autocorrelation. These lags are lag 1 and 2.

5.6 Result of Feature Engineering Process

As outlined in Section 4.5, the feature engineering process is guided by the findings of the data analysis. Table 1 summarizes the complete set of engineered features for each product family and each developed model. "Yes" indicates that the model uses the feature for learning, "No" indicates that the model does not use the feature for learning.

	Product Family A	**Product Family B**	**LightGBM**	**XGBoost**	**LSTM**
Time-Based Features	Month, Quarter, Year	Month, Quarter, Year	Yes	Yes	Yes
Lag Features (lags)	1, 2, 3, 4	1, 2	Yes	Yes	No
First Order Difference Features (lags)	1, 2, 3, 4	1, 2	Yes	Yes	No
Second Order Difference Features (lags)	1, 2, 3, 4	1, 2	Yes	Yes	No
Minimum and Maximum Features (window)	4, 10, 15	5, 9	Yes	Yes	No
MA features (window)	4, 10, 15, expanding	5, 9, expanding	Yes	Yes	No
EWM features (window)	4, 10, 15, expanding	4, 10, 15, expanding	Yes	Yes	No
Rolling SD Features (window)	4, 10, 15, expanding	5, 9, expanding	Yes	Yes	No

Table 1 Complete set of engineered features for LightGBM, XGBoost, and LSTM.

The selected features were informed by the analyses presented in Sections 5.2, 5.3, 5.4, and 5.5. Lag features were chosen based on statistically significant lags identified in the autocorrelation analysis, ensuring that the model had access to the most informative past values. The difference features (first and second order) were included to capture changes and inflection points in demand, which were observed for both Product Family A and Product Family B. Rolling statistics features (MA, Minimum and Maximum, Rolling SD) were motivated by the presence of local volatility. These features were chosen to help the model detect recent peaks, valleys, and periods of stability. EWM features were included to provide a smoothed representation of past demand, with adjustable sensitivity controlled by α. Using $\alpha = 0.5, 0.7, 0.9$ ensured a balance between short-term and long-term memory.

As earlier noted, for the LSTM model, only time-based features were retained, as the LSTM architecture is inherently capable of learning temporal dependencies.

5.7 Hyperparameter Optimization Parameter Grids

The hyperparameter values listed in Tables 2, 3, and 4 were selected to balance flexibility, training time, and the risk of overfitting. All ranges were determined based on prior literature, exploratory testing, and computational feasibility.

The hyperparameters of LightGBM deemed most important and relevant to this thesis are listed in Table 2.

Hyperparameter	**Values Product Family A**	**Values Product Family B**
Number of Estimators	[100, 150, 200, 250]	[100, 150, 200, 250]
Number of Leaves	[5, 10, 15]	[5, 10, 15]
Maximum Depth	[3, 5, 7, 9]	[3, 5, 7, 9]
Learning Rate	[0.05, 0.06, 0.07, 0.08]	[0.05, 0.06, 0.07, 0.08]

Table 2 Hyperparameter optimization parameter grid for LightGBM model.

The hyperparameters of XGBoost deemed most important and relevant to this thesis are listed in Table 3.

Hyperparameter	**Values Product Family A**	**Values Product Family B**
Number of Estimators	[100, 150, 200, 250]	[100, 150, 200, 250]
Minimum Child Weight	[5, 10, 15]	[5, 10, 15]
Maximum Depth	[3, 5, 7, 9]	[3, 5, 7, 9]
Learning Rate	[0.05, 0.06, 0.07, 0.08]	[0.05, 0.06, 0.07, 0.08]

Table 3 Hyperparameter optimization parameter grid for XGBoost model.

The number of estimators for both LightGBM and XGBoost was set in the range of 100-250 for both product families. This range allows the ensemble models to be expressive without incurring excessive training cost. Increasing the number of estimators beyond this range did not improve performance but instead led to overfitting. The values for number of leaves, minimum child weight, and maximum depth were tuned to control the tree complexity and were determined primarily through experimental testing. The learning rates for the GBDTs implementations were also chosen based on empirical experimentation. More conservative and aggressive rates were initially explored, however, the range that yielded the best and most consistent performance was found to be in the range of 0.05-0.08.

The hyperparameters of most important and relevant to optimize in this thesis are listed in Table 4.

Hyperparameter	**Values Product Family A**	**Values Product Family B**
Input Sequence	[4, 5, 7, 10, 15]	[5, 6, 7, 8, 9]
Hidden Layers	[1, 2]	[1, 2]
Neurons per Layer	[4, 8, 16, 32, 64, 128]	[4, 8, 16, 32, 64, 128]
Dropout Rate	[0.01, 0.05]	[0.01, 0.05]
Learning Rate	[0.0001, 0.001, 0.01]	[0.0001, 0.001, 0.01]
Batch Size	[1, 5, 10, 20]	[1, 5, 10, 20]
Epoch Size	[50, 100, 200, 300, 400, 500]	[50, 100, 200, 300, 400, 500]

Table 4 Hyperparameter optimization parameter grid for LSTM model.

The input sequence parameter for LSTM corresponds to the lag features used in LightGBM and XGBoost, as it determines the number of time steps fed into the model. Its values were selected based on lags showing meaningful autocorrelation. For example, Product Family A showed no meaningful autocorrelation beyond lag 15. Therefore, longer sequences than 15 were not considered.

The number of hidden layers was limited to one or two to avoid unnecessary complexity. This was consistent with findings in similar studies where deeper architectures provided no additional benefit. The number of neurons per layer was explored in the interval $[4, 8, 16, 32, 64]$, following a power of two progression. This choice is common in neural network tuning as it spans a wide range of representational model capacitites while maintaining efficiency. The dropout rate was restricted to relatively low values, reflecting the small dataset sizes. Higher rates would risk discarding too much information and hindering learning rather than improving generalization.

The learning rate range was determined after testing more aggressive and more conservative values outside this interval. A wider range was initially explored to identify whether faster learning or more gradual optimization was advantageous. However, the chosen interval $[0.0001, 0.001, 0.01]$ yielded the best and most consistent performance.

The batch size was explored in the interval $[1, 5, 10, 20]$ to assess the trade-off between sensitivity to data variability (smaller batches) and gradient stability (larger batches).

Finally, the number of epochs was explored in the interval $[50, 100, 200, 300, 400, 500]$

to capture both short and long training schedules. This allowed identification of the point where no additional training no longer improved performance, achieving a balance between convergence and generalization. Given the small datasets, it was computationally feasible to explore this wider range of epoch sizes.

6 Results

This chapter presents the results of the experiments conducted in the thesis and addresses the research questions. The performance of the developed forecasting models for both product families is evaluated using selected error metrics, and the results are compared against a baseline method.

6.1 Forecasting Results for Product Family A

Table 5 presents the performance of the developed forecasting models for Product Family A. Each model, including the baseline, is evaluated using the selected error metrics: MAE, RMSE, and SMAPE.

Error Metric	LightGBM	XGBoost	LSTM	Baseline
MAE (units)	6.4	14.8	22.4	15.6
RMSE (units)	9.4	18.1	27.9	20.5
SMAPE (%)	20.8	36.8	54.5	40.3

Table 5 Forecasting performance for Product Family A during the year 2024.

Table 5 shows that LightGBM achieved best performance in forecasting the demand of Product Family A, followed by XGBoost and then LSTM. When compared to the baseline model, both LightGBM and XGBoost outperformed the baseline, while LSTM did not.

Table 6 presents the monthly forecasts for the year 2024. The values in the parentheses indicate the absolute difference between the actual and forecasted demand.

Date	Actual Demand	LightGBM	XGBoost	LSTM	Baseline
2024-01-01	21.0	29.4 (8.4)	48.1 (27.1)	41.1 (20.1)	30.2 (9.2)
2024-02-01	38.0	28.8 (9.2)	52.8 (14.8)	21.4 (16.6)	27.0 (11.0)
2024-03-01	28.0	33.8 (5.8)	51.2 (23.2)	41.0 (13.0)	27.6 (0.4)
2024-04-01	74.0	74.8 (0.8)	51.2 (22.8)	41.9 (32.1)	27.9 (46.1)
2024-05-01	29.0	29.0 (0.0)	38.2 (9.2)	23.0 (6.0)	34.1 (5.1)
2024-06-01	29.0	25.7 (3.3)	32.2 (3.2)	64.8 (35.8)	36.5 (7.5)
2024-07-01	24.0	19.1 (4.9)	22.7 (1.3)	60.2 (36.2)	33.7 (9.7)
2024-08-01	16.0	41.9 (25.9)	34.2 (18.2)	60.9 (44.9)	34.8 (18.8)
2024-09-01	53.0	53.3 (0.3)	35.6 (17.4)	51.2 (1.8)	33.1 (19.9)
2024-10-01	35.0	46.4 (11.4)	28.8 (6.2)	31.8 (3.2)	27.5 (7.5)
2024-11-01	46.0	39.4 (6.6)	46.4 (0.4)	52.0 (6.0)	34.0 (12.0)
2024-12-01	73.0	62.0 (11.0)	39.3 (33.7)	20.4 (52,6)	33.3 (39.7)

Table 6 Forecasting results by month for Product Family A during 2024.

Table 6 shows that LightGBM outperformed the baseline model in 9 out of 12 months (75%), while XGBoost did so in 8 out of 12 months (67%). LSTM, by contrast, only surpassed the baseline in 3 out of 12 months (25%)

The research question was ”How does the performance of ML models compare to that of a baseline model in forecasting the demand for Product Family A?”. The results show that LightGBM performed best relative to the baseline, followed by XGBoost, and lastly LSTM. A visual representation of the forecasting results for Product Family A is provided in Appendix C

6.2 Forecasting Results for Product Family B

Table 7 presents the performance of the developed forecasting models for Product Family B. Each model, including the baseline, is evaluated using the selected error metrics: MAE, RMSE, and SMAPE.

Error Metric	LightGBM	XGBoost	LSTM	Baseline
MAE (units)	63.8	56.0	70.8	45.3
RMSE (units)	69.4	63.7	83.4	52.0
SMAPE (%)	87.1	81.2	86.9	72.4

Table 7 Forecasting performance for Product Family B during the year 2024.

Table 7 shows that XGBoost achieved the best performance in forecasting demand for Product Family B, followed by LightGBM and then LSTM. However, when compared to the baseline model, none och the developed forecasting models outperformed it.

Table 8 presents the monthly forecasts for the year 2024. The values in the parentheses indicate the absolute difference between the actual and forecasted demand.

Date	Actual Demand	LightGBM	XGBoost	LSTM	Baseline
2024-01-01	132.0	122.3 (9.7)	133.1 (1.1)	118.5 (13.5)	135.3 (3.3)
2024-02-01	43.0	89.5 (46.5)	101.2 (58.2)	91.6 (48.6)	130.0 (87.0)
2024-03-01	36.0	137.0 (101.0)	131.0 (95.0)	140.4 (104.4)	111.7 (75.7)
2024-04-01	20.0	113.8 (93.6)	122.6 (102.6)	160.4 (140.4)	100.9 (80.9)
2024-05-01	44.0	123.4 (79.4)	134.6 (92.6)	148.5 (104.5)	82.1 (38.1)
2024-06-01	119.0	82.3 (36.7)	71.9 (47.1)	143.7 (24.7)	80.1 (38.9)
2024-07-01	23.0	103.3 (80.1)	66.3 (43.3)	144.8 (121.8)	80.3 (57.1)
2024-08-01	33.0	75.4 (42.4)	91.0 (58)	133.5 (100.5)	80.3 (47.3)
2024-09-01	52.0	105.5 (53.5)	91.2 (20.8)	101.7 (49.7)	77.4 (25.4)
2024-10-01	77.0	123.6 (46.6)	64.4 (12.6)	72.7 (4.3)	67.9 (9.1)
2024-11-01	20.0	113.7 (93.7)	100.3 (80.3)	120.4 (100.4)	66.3 (46.3)
2024-12-01	28.0	110.3 (82.3)	71.5 (43.5)	64.8 (36.8)	62.8 (34.0)

Table 8 Forecasting results by month for Product Family B during 2024.

Table 8 shows that LightGBM outperformed the baseline in only 2 out of 12 months (17%), while XGBoost and LSTM outperformed the baseline in 4 months (33%) and 3 months (25%), respectively.

The research question was ”How does the performance of ML models compare to that of a baseline model in forecasting the demand for Product Family A?”. The results show that XGBoost performed best relative to the baseline, followed by LightGBM, and lastly LSTM. However, none of the developed forecasting models outperformed the baseline. A visual representation of the forecasting results for Product Family B is provided in Appendix D

7 Discussion

This chapter presents a discussion of the results and the methodological choices. It also includes general reflections and considerations of potential improvements.

From the previous chapter, it is evident that the forecasting results for Product Family A are significantly better than the results for Product Family B. The PACF plots indicate that in the time series for Product Family B, only the first two lags exhibit meaningful direct autocorrelation. In comparison, the time series for Product family A shows meaningful autocorrelation across the first four lags. This difference could explain the observed variation in forecasting performance. Another possible explanation for the poorer performance on Product Family B is the nature of the underlying dataset. Product Family B includes demand from 82 unique customers, whereas Product Family A's dataset involves only 8 customers, all associated with the same company. This uniformity could have contributed to the forecasting being more predictable, and thus the forecasting performance being better for Product Family A.

As seen in Figures 10 and 13, both time series are characterized by strong month-to-month fluctuations, making forecasting a difficult task as it is. But looking more into detail on Figure 13, depicting the demand of Product Family B, there is a sharp spike followed by a steep downward trend at the end of the observation period. This sudden change suggests that the year 2024, which the experiment was forecasting, have been an anomalous year. Since the model was not trained on patterns resembling this behavior, it struggled to produce accurate forecasts. It is important to remember that a machine learning model is only as good as the data it has been trained on.

The datasets for Product Family A and Product Family B are derived from historical demand, with all other features engineered from this base. Given the high month-to-month fluctuations in both time series, it would be beneficial to incorporate features that explain and influence demand, commonly referred to as demand drivers. These features could include indicators of a product family's position in its life cycle, such as whether it is peaking or approaching end of life, as these stages are likely to impact demand patterns. Additionally, as mentioned in the introduction chapter, the case company is sensitive to a wide range of external economic and industry specific factors. These

external factors could also serve as important demand drivers.

Initially, this thesis aimed to conduct two types of experiments: the first being the current study, and the second involving the inclusion of external factors to assess their impact on forecasting performance. Examples of such external factors could be inflation, GDP, and market trends. However, the case company did not have this type of data documented, and it proved difficult to obtain this type of data from external sources. As a result, the second planned experiment had to be discarded.

Another aspect worth discussing is the representativeness of the data. Does a billing log truly reflect actual demand? For instance, the billing log does not account for product availability. Theoretically, there could still be demand for a product even if it is out of stock and therefore cannot be sold or recorded in the billing log. Additionally, the billing log does not capture the impact of blocked customers. Blocked customers are customers temporarily restricted due to, for instance, unpaid invoices. As a result, a peak in the demand plots (Figures 10 and 13) could be explained by previously blocked customers being unblocked.

The methodological choices made in this thesis also merit reflection. In hindsight, it may have been better to adopt a global approach rather than a local one, or to conduct experiments using both to determine which yields better performance. A global approach would provide the model with more data and involve training on all product families simultaneously. This would enable the model to learn interdependencies between product families. For example, product families targeting the same end markets are likely to exhibit similar demand patterns. Moreover, a decline in demand in one end market could indicate growth in another. When any form of interdependency is present, a global approach may yield a more accurate and robust forecasting model.

Another methodological consideration concerns the approach used for multistep forecasting. As outlined in Section 2.6, several strategies exist for addressing this type of problem. Based on the findings from the data analysis presented in Chapter 5, particularly the characteristics of both time series and their autocorrelation patterns, this thesis adopted the recursive multistep forecasting approach. The alternative strategies involve developing models that forecast multiple future time steps directly. However, considering the strong month-to-month fluctuations and the limited autocorrelation, especially

in the latter part of the forecast horizon, direct forecasting appeared unstable. For these reasons, the recursive approach was deemed the most appropriate choice.

8 Conclusions

This chapter summarizes the key findings of the thesis and answers the research questions.

The objective of this thesis was to explore how ML can be applied to demand forecasting. This was achieved by conducting experiments on two product families using historical demand data provided by a case company. The experimental setup was designed to simulate the case company's current forecasting conditions. Since no existing method was available for comparison, a 12-month MA was introduced as a baseline model. The thesis evaluated three different ML models: LightGBM, XGBoost, and LSTM.

Two experiments were conducted, one for a Product Family A and one for a Product Family B. The results show that for Product Family A, LightGBM achieved the best performance, followed by XGBoost and LSTM. In this case, both LightGBM and XGBoost outperformed the baseline. For Product Family B, XGBoost performed best, followed by LightGBM and LSTM. However, none of the developed forecasting models outperformed the baseline for this product family.

Accurate demand forecasting can be highly valuable for businesses as it supports better planning and decision-making. However, forecasting remains a complex task, especially over longer horizons. The forecasting models developed in this thesis are not yet ready for deployment in a production environment as further refinements are necessary. The performance of ML models heavily depends on the preparatory work carried out before model training. It is essential to structure and model the data in a way that accurately reflects the context in which the forecasts will be applied.

In conclusion, this thesis has demonstrated how ML can be applied to demand forecasting in a real-world setting. While the results leave room for improvement, they suggest that ML holds strong potential in this domain. Therefore, the objective of this thesis is considered fulfilled.

9 Future Work

This chapter outlines potential directions for future research based on the findings and limitations of the thesis.

While this thesis has provided valuable insights into the application of ML for demand forecasting, several areas remain open for further exploration. One such area is the use of a global forecasting approach, which would account for interdependencies across product families. This is an aspect that is not captured by the local approach used in this thesis. Another area for improvement is data modeling. The observed demand patterns of the product families in this thesis are highly fluctuating and exhibit limited autocorrelation. This indicates a need for involving features that can explain the underlying behavior. Future research could therefore focus on identifying demand drivers and collecting such data if available, or establishing processes to collect it going forward. Finally, further investigation into the mentioned multistep forecasting approaches could serve as valuable direction for future research.

References

[1] H. Abbasimehr, M. Shabani, and M. Yousefi, "An optimized model using lstm network for demand forecasting," *Computers Industrial Engineering*, vol. 143, p. 106435, 2020.

[2] M. Agrawal and P. Agrawal, "A survey on hyperparameter optimization of machine learning models," *2024 2nd International Conference on Disruptive Technologies (ICDT)*, pp. 11–15, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269090334

[3] A. Bryman, E. Bell, and B. Harley, *Business Research Methods*, 5th ed. Oxford University Press, 2019.

[4] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[5] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[6] F. Chollet, T. Kalinowski, and J. J. Allaire, *Deep learning with R*, second edition. ed. Shelter Island, NY: Manning Publications Co., 2022.

[7] S. Chopra and P. Meindl, *Supply Chain Management: Strategy, Planning, and Operation*. Pearson, 2021.

[8] P. P. Deka, J. Weiner, and P. R. V. Zicari, *XGBoost for Regression Predictive Modeling and Time Series Analysis : Learn How to Build, Evaluate, and Deploy Predictive Models with Expert Guidance.*, 1st ed. Birmingham: Packt Publishing, Limited, 2024.

[9] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: methods and prospects," *Big data analytics*, vol. 1, pp. 1–22, 2016.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del

Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[11] W. He, P. Liu, and Q. Qian, *Machine Learning Contests: A guidebook*. Springer Nature Singapore, 2023.

[12] T. Ho, L. Tran, H. Tran, and S. Dao, "Machine learning in demand forecasting," *International Research Journal of Advanced Engineering and Science*, 01 2022.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[15] M. Joseph, *Modern Time Series Forecasting with Python : Explore Industry-Ready Time Series Forecasting Using Modern Machine Learning and Deep Learning*, first edition. ed. Birmingham, England: Packt Publishing, 2022.

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[17] M. Kubat, *An Introduction to Machine Learning*, 3rd ed. Cham: Springer Nature, 2015.

[18] A. R. Kulkarni, *Time Series Algorithms Recipes : Implement Machine Learning and Deep Learning Techniques with Python*, 1st ed. Berkeley, CA: Apress, 2023.

[19] G. Kunapuli, *Ensemble methods for machine learning*. Shelter Island: Manning Publications, 2023.

[20] F. Lazzeri, *Machine Learning for Time Series Forecasting with Python*. John Wiley & Sons, Incorporated, 2020.

[21] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. [Online]. Available: https://smlbook.org

[22] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler, "The accuracy of extrapolation (time series) methods: Results of a forecasting competition," *Journal of forecasting*, vol. 1, no. 2, pp. 111–153, 1982.

[23] S. Makridakis and M. Hibon, "Accuracy of forecasting: An empirical investigation," *Journal of the Royal Statistical Society: Series A (General)*, vol. 142, no. 2, pp. 97–125, 1979.

[24] S. Makridakis and M. Hibon, "The m3-competition: results, conclusions and implications," *International Journal of Forecasting*, vol. 16, no. 4, pp. 451–476, 2000, the M3- Competition. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207000000571

[25] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: Results, findings, conclusion and way forward," *International Journal of Forecasting*, vol. 34, no. 4, pp. 802–808, 2018.

[26] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1346–1364, 2022.

[27] J. Manu Joseph, *Modern Time Series Forecasting With Python: Master Industry-Ready Time Series Forecasting Using Modern Machine Learning and Deep Learning*. Packt Publishing, 2022.

[28] A. Martín, A. Ashish, B. Paul, B. Eugene, C. Zhifeng, C. Craig, S. C. Greg, D. Andy, D. Jeffrey, D. Matthieu *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," *Software available from tensorflow. org*, vol. 7, 2015.

[29] L. Massaron, *Deep learning*, 1st ed., ser. For dummies. Hoboken, N.J: J. Wiley, 2019.

[30] T. Masui, "All you need to know about gradient boosting algorithm — part 1 (regression)," https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502/, 2022, accessed: 2025-05-15.

[31] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference (SciPy)*, S. van der Walt and J. Millman, Eds. SciPy, 2010, pp. 51–56.

[32] M. A. Mediavilla, F. Dietrich, and D. Palm, "Review and analysis of artificial intelligence methods for demand forecasting in supply chain management," *Procedia CIRP*, vol. 107, pp. 1126–1131, 2022.

[33] D. Mobbs, C. C. Hagan, T. Dalgleish, B. Silston, and C. Prévost, "The ecology of human fear: Survival optimization and the nervous system," *Frontiers in Neuroscience*, vol. 9, no. 55, p. 121062, 2015. [Online]. Available: https://doi.org/10.3389/fnins.2015.00055

[34] D. C. Montgomery, *Introduction to Time Series Analysis and Forecasting*. John Wiley & Sons, Incorporated, 2015.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] F. Petropoulos *et al.*, "Forecasting: theory and practice," *International Journal of Forecasting*, vol. 38, no. 3, pp. 705–871, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021001758

[38] M. Pravin and V. Bhandari, "A systematic study on effective demand prediction using machine learning," *Journal of Integrated Science and Technology*, vol. 12, no. 1, p. 711, 2023. [Online]. Available: https://pubs.thesciencein.org/journal/index.php/jist/article/view/a711

[39] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, pp. 75–85, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207019301153

[40] D. Stanton, *Supply chain management*, third edition. ed., ser. for dummies. Hoboken, New Jersey: John Wiley Sons Inc., 2023.

[41] N. Yadav, A. Yadav, and M. Kumar, *An introduction to neural network methods for differential equations*, 2015th ed., ser. SpringerBriefs in Applied Sciences and Technology. Dordrecht: Springer, 2015.

A Pseudocode of LightGBM and XGBoost Models

Since LightGBM and XGBoost are both implementations of GBDTs, their model development follows the same procedure. The pseudocode for both models is presented in Algorithm 1.

```
Load preprocessed data
Split data into features and target using data up to 2023
Define hyperparameter grid
Run GridSearchCV to find best hyperparameters
for month = 1 to 12 do
    Select training data up to the same month in 2023
    Fit model using best hyperparameters
    Predict on training data
    Compute RMSE, MAE, and SMAPE of training
    Initialize recursive forecast dataset (copy of training data)
    for step = 1 to forecast horizon do
        Select last row as input
        Predict next value
        Append prediction to recursive forecast dataset
        Recalculate features
    end for
    Store final forecast for the month
end for
Compute RMSE, MAE, and SMAPE of forecast
Visualize results
```

Algorithm 1 Pseudo code for the LightGBM and XGBoost models.

B Pseudocode of LSTM Model

The pseudocode for the LSTM model is presented in Algorithm 2

function CREATESEQUENCES(data, lookback)
    Generate overlapping input-output pairs $(X_{\text{train}}, y_{\text{train}})$
end function
function RECURSIVEFORECAST(model, input_seq, steps)
    for each forecast horizon step **do**
        Predict next value
        Append prediction to sequence
    end for
    return last prediction
end function
Load preprocessed data
Normalize target using `MinMaxScaler`
Split data into features and training using all data up to 2023 using CREATESEQUENCES(train_series, lookback)
Reshape x_{train} for LSTM input
Run `GridSearchCV` to find best hyperparameters
for month = 1 to 12 **do**
    Split data into features and target using all data up to the same month in 2023 using CREATESEQUENCES(train_series, lookback)
    Reshape x_{train} for LSTM input
    Fit model using best hyperparameters
    Predict on training data
    Inverse scale predictions
    Compute RMSE, MAE, and SMAPE of training
    Extract last lookback window from training data
    RECURSIVEFORECAST(model, input_seq, 12)
    Inverse scale last prediction
    Store result
end for
Compute RMSE, MAE, and SMAPE of forecast
Visualize results

Algorithm 2 Pseudo code for the LSTM model.

C Visual Representation of Forecasting Results Product Family A

The following plots provide a visual representation of the forecasting results for Product Family A.

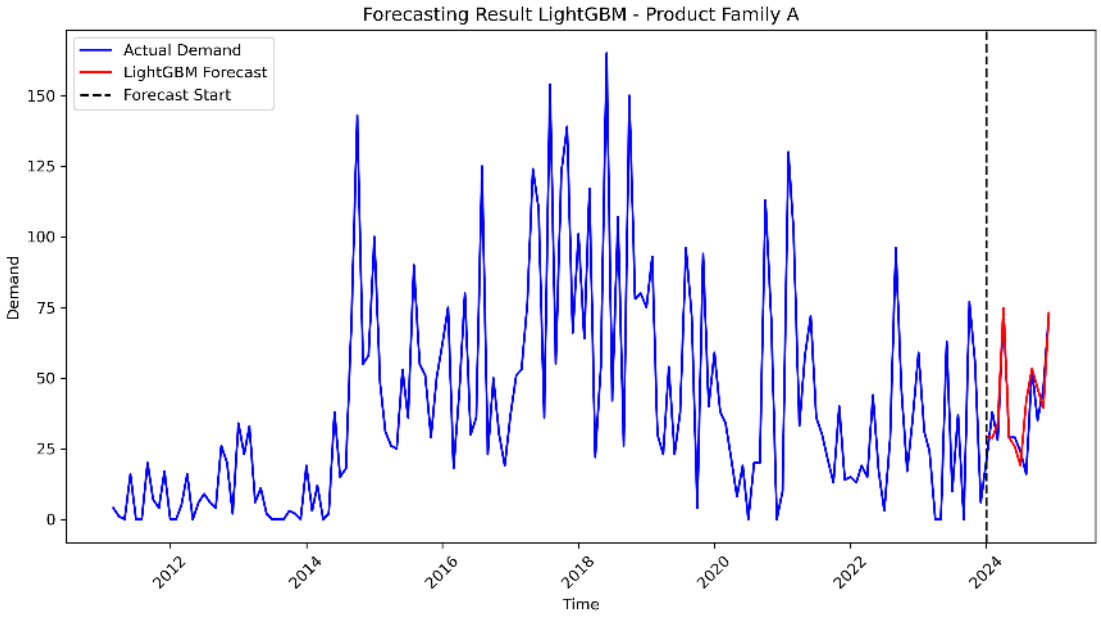


Figure 16 Forecasting results of LightGBM for Product Family A

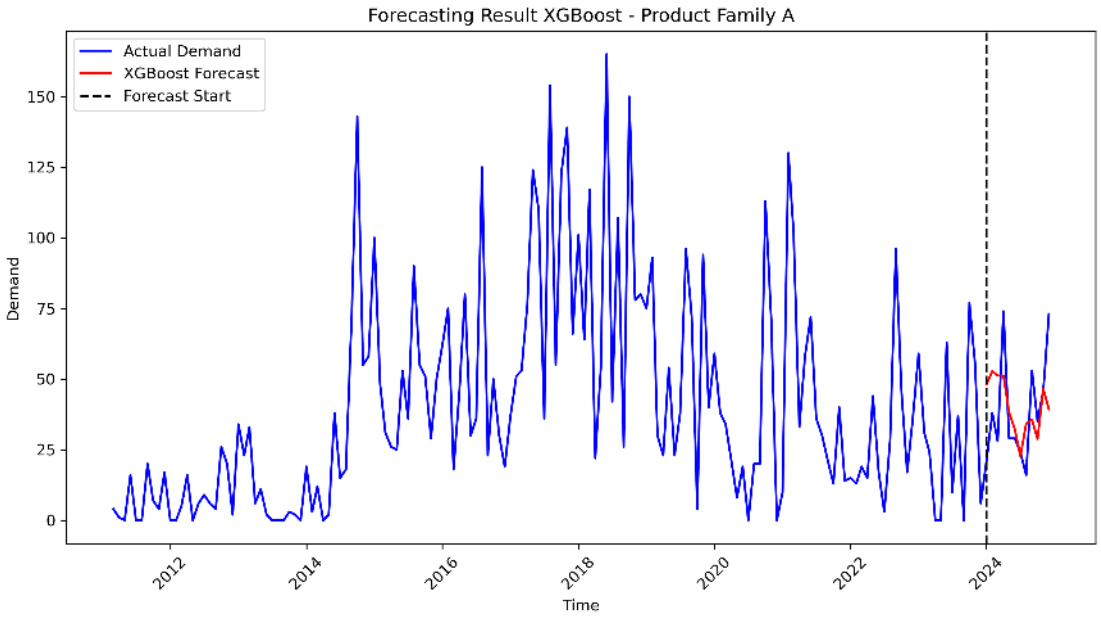


Figure 17 Forecasting results of XGBoost for Product Family A

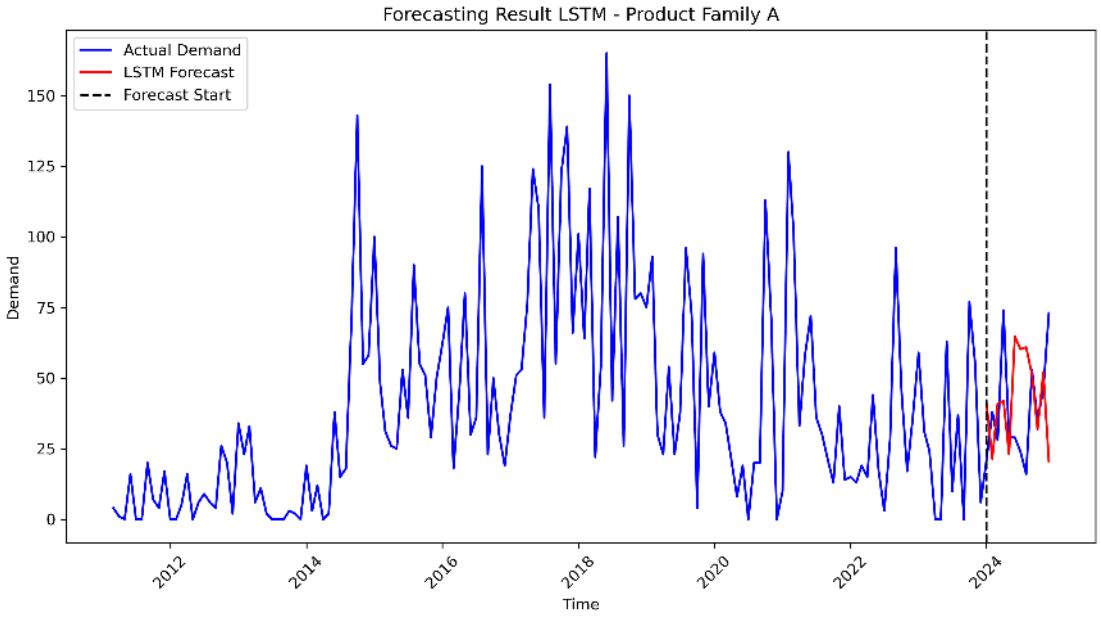


Figure 18 Forecasting results of LSTM for Product Family A

D Visual Representation of Forecasting Results Product Family B

The following plots provide a visual representation of the forecasting results for Product Family B.

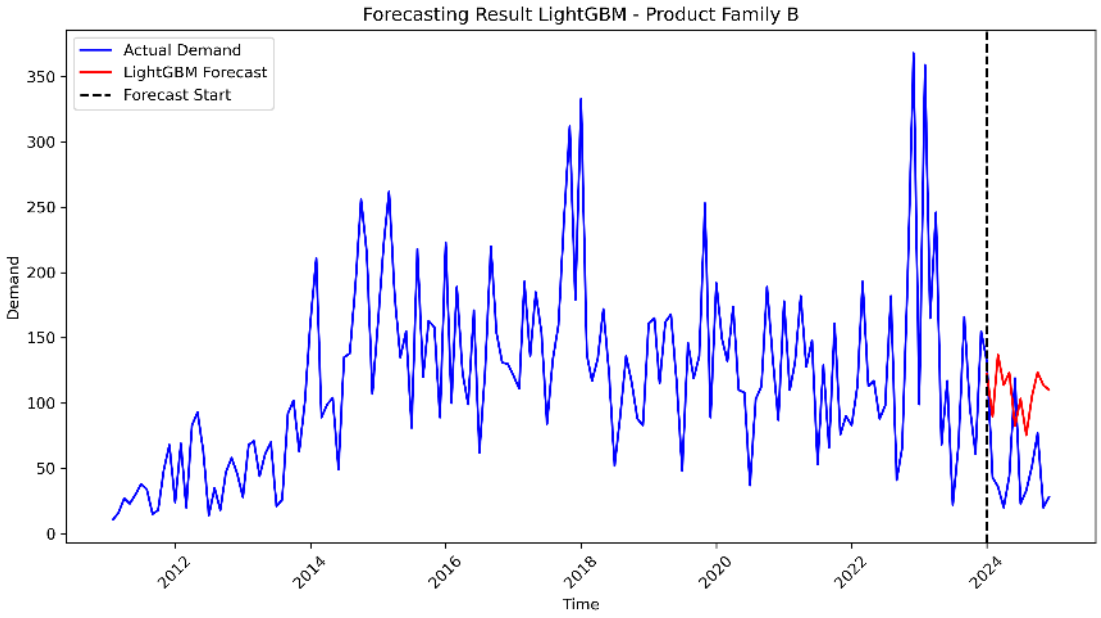


Figure 19 Forecasting results of LightGBM for Product Family B

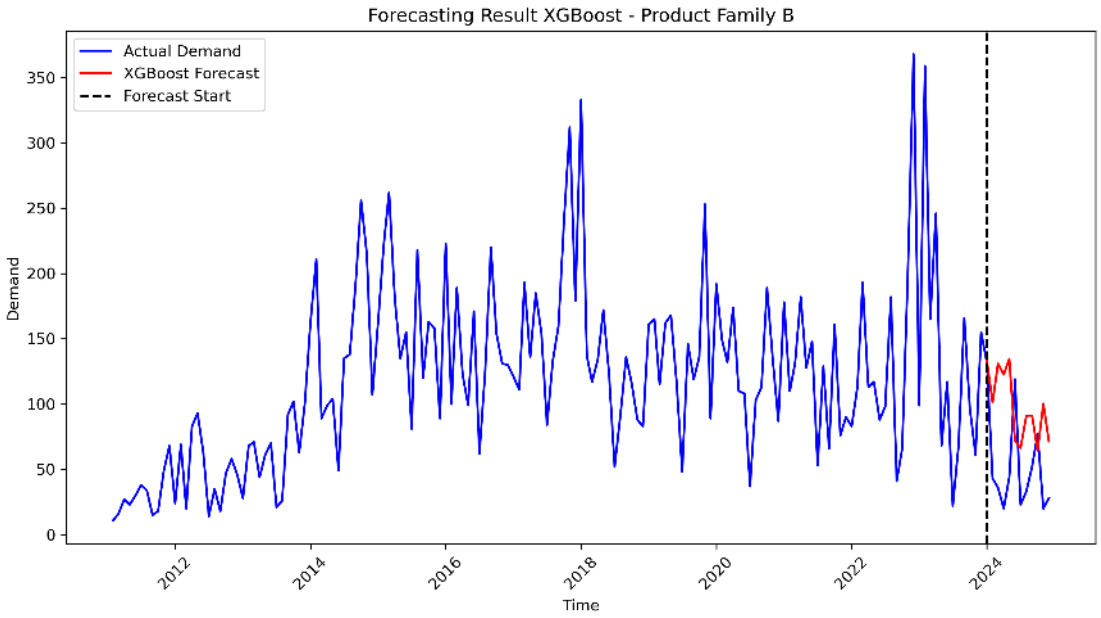


Figure 20 Forecasting results of XGBoost for Product Family B

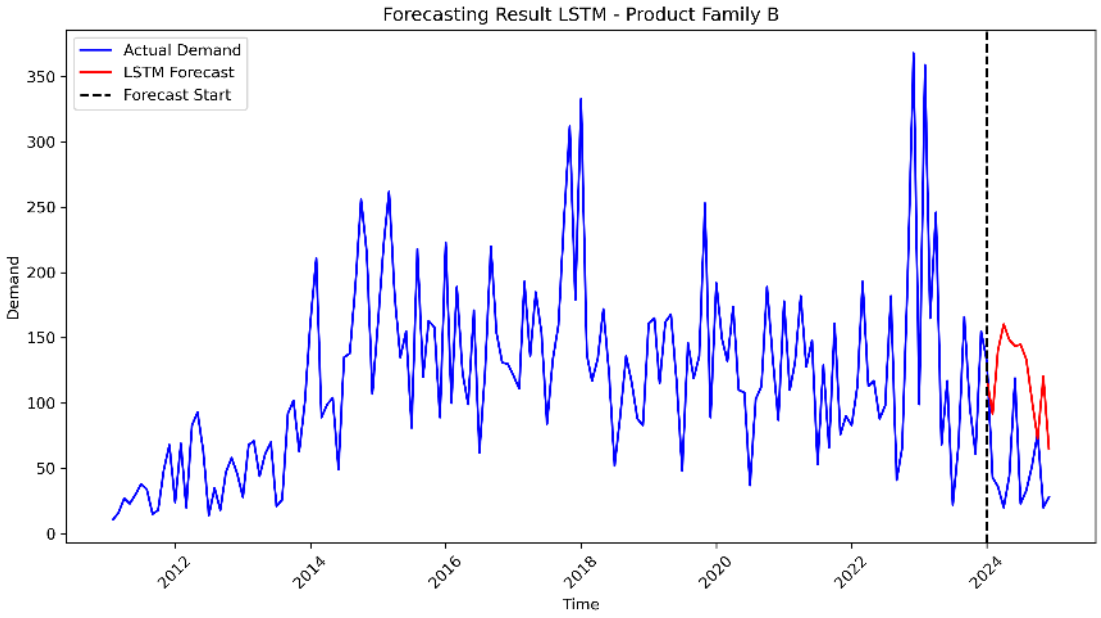


Figure 21 Forecasting results of LSTM for Product Family B