IT mIA 25 010
Degree project 30 credits
July, 2025

UPPSALA UNIVERSITET

Reducing Supply Chain Uncertainty with Machine Learning: Forecasting Supplier Lead Times

Oskar Granlund

Master's Programme in Industrial Analytics

UPPSALA UNIVERSITET

Reducing Supply Chain Uncertainty with Machine Learning: Forecasting Supplier Lead Times

Oskar Granlund

Abstract

With supply chains becomes more globalized and complex, accurate lead time forecasting is becoming increasingly more critical for maintaining operational efficiency and company competitiveness. This thesis investigates how Machine Learning (ML) techniques can increase the predictability of external supplier lead times using historical and contextual data. Three models were chosen based on previous research done in similar studies: Random Forest (RF), EXtreme Gradient Boosting (XGBoost) and Neural Network (NN). The data, obtained from a leading manufacturing company in the mining sector, was preprocessed and engineered to improve the model forecasting accuracy. Each model was optimized using hyperparameter tuning and evaluated using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the Coefficient of Determination (R^2).

All three models demonstrated considerable improvements over the company's baseline lead time estimations. The NN outperformed the other models with an improvement of 63.7% MAE in comparison to the baseline estimates. XGBoost and RF did also perform well, with error reductions of 59.6% and 54.5% respectively.

The results confirm that ML can effectively improve external lead time forecasting in complex supply chain environments. The models developed during this thesis offer a practical use for improving planning reliability, operational efficiency improvements, and supporting decision-making.

Faculty of Science and Technology
Uppsala University, Place of publication Uppsala

Supervisor: Toni Sigmundson Subject reader: Stefanos Kaxiras
Examiner: Olle Gällmo & Hans Karlsson

Sammanfattning

I takt med att leveranskedjor blir allt mer globaliserade och komplexa, blir noggrann prognostisering av ledtider allt viktigare för att upprätthålla operativ effektivitet och konkurrenskraft. Denna uppsats undersöker hur maskininlärningstekniker kan öka förutsägbarheten för externa leverantörers ledtider genom att använda historisk och kontextuell data. Tre modeller valdes ut baserat på tidigare forskning inom området: RF, XGBoost och NN. Den data som erhölls från ett ledande tillverkningsföretag inom gruvsektorn, förbehandlades och anpassades för att förbättra modellernas prognosnoggrannhet. Varje modell optimerades genom hyperparameterjustering och utvärderades med hjälp av metoderna genomsnittligt absolut fel MAE, kvadratroten av medelkvadratfelet RMSE samt determinationskoefficienten R^2.

Samtliga modeller visade tydliga förbättringar jämfört med företagets befintliga uppskattningar av ledtider. NN presterade bäst med en förbättring på 63,7% i MAE jämfört med baslinjeuppskattningarna. Även XGBoost och RF uppvisade goda resultat med felreduktioner på 59,6% respektive 54,5%.

Resultaten bekräftar att ML effektivt kan förbättra prognostiseringen av externa ledtider i komplexa leveranskedjemiljöer. De modeller som utvecklats inom ramen för denna uppsats utgör ett praktiskt verktyg för att förbättra planeringssäkerhet, öka den operativa effektiviteten och stödja beslutsfattande.

Acknowledgement First and foremost, I would like to express my deepest gratitude to my supervisors at Frontit, Daniel Rödjemyr and Toni Sigmundson, for their invaluable support and encouragement throughout this thesis. I am especially thankful for their efforts in coordinating and maintaining communication with Epiroc, which proved vital to the success of this thesis. I would also like to show gratitude towards Frontit for helping me find this thesis opportunity, and for their warm welcome and supportive environment.

I would also like to extend my sincere appreciation to Epiroc for serving as the case company in this study. A special thanks to Stina Bouvin for being supportive throughout the project, as well as providing data and practical insights into Epiroc's operations.

Furthermore, I am grateful to my subject reviewer, Stefanos Kaxiras, for his constructive feedback and guidance on academic writing.

Contents

List of Figures

List of Tables

Acronyms

R^2	Coefficient of Determination.
Adam	Adaptive Moment Estimation.
AI	Artificial Intelligence.
CV	Cross-Validation.
FNN	Feedforwad Neural Network.
GridSearchCV	Grid Search with Cross-Validation.
MAE	Mean Absolute Error.
ML	Machine Learning.
MSE	Mean Squared Error.
NN	Neural Network.
RF	Random Forest.
RMSE	Root Mean Squared Error.
SCM	Supply Chain Management.
SGD	Stochastic Gradient Descent.
XGBoost	EXtreme Gradient Boosting.

1 Introduction

This master’s thesis was conducted at Uppsala University in collaboration with Frontit AB and Epiroc. Frontit is a consulting company that specializes in project- and change management, providing expert guidance to organizations. Epiroc is an industry leader in innovative solutions for mining and infrastructure operations.

1.1 Background

In modern times, supply chains are increasingly globalized, complex, and sensitive to external disruptions. To attain an efficient Supply Chain Management (SCM), the ability to accurately forecast supplier lead times has become a crucial component. Not having accurate lead time forecasts may lead to major operational inefficiencies, such as unnecessarily high inventory costs, production disruptions, or idle resources.

The COVID-19 pandemic exposed underlying vulnerabilities in the global supply chains [1]. As just-in-time strategies have become widespread, production disruptions in certain areas have had major downstream impacts and uncertainties throughout entire supply networks [2]. To address this, many companies started to overcompensate by inflating their promised lead times, to reduce the risk of missing delivery deadlines [3]. While this may prevent late deliveries, it introduces new inefficiencies, such as early arrivals and increasing of stock capacity demands.

As companies attempt to balance a lean methodology with supply chain robustness, the need for more accurate, data-driven lead time predictions has grown. In recent years, ML has become an increasingly promising method to forecast lead times more reliably [4], by identifying the underlying patterns in historical and contextual data that traditional forecasting methods fail to uncover.

1.2 Problem Description

To illustrate this industry-wide issue in a real-world setting, this thesis examines a case study involving Epiroc, a global manufacturing company in the mining and infrastructure industry. Epiroc collaborates with a large network of external suppliers, all of whom provide fixed annual lead time estimates that are often inaccurate. In a significant number of cases, deliveries arrive either earlier or later than originally promised by the supplier. This results in planning disruptions, stock inefficiencies, production delays, and increased operational costs.

These deviations in delivery timing, complicate scheduling and inventory control. while overestimating lead times may result in a reduction in late deliveries, they can lead to early deliveries which in turn lead to excess inventory levels. This brings issues to the supply chain further downstream in the production system, as it introduces inefficiencies that ultimately result in worse operational performance [2].

To address this issue, this thesis proposes the utilization of a ML-based approach to predict the actual supplier lead times. By analyzing historical order data and incorporating contextual variables, the aim is to develop ML models that offer greater accuracy than the existing supplier lead time estimates. Three models is selected based on prior research in the subject and the structure of the data, they are compared against each other and the baseline supplier lead time estimates. Such models can enable more reliable planning, improved production up-time, and reduced inventory and delay-related costs.

1.3 Explanation of Key Concepts

In this section, the key concepts which are specific to this project will be explained.

- **Lead Time:** In the context of this thesis, lead time refers to the entire duration of the supplier's process. From the moment an order is released to the point when the goods leave the supplier's possession.
- **Actual Lead Time:** The real-time interval between when the order is released and when it is fully completed in the supplier's process. Delivery or transportation is not included in the scope of this thesis.
- **Predicted Lead Time:** The estimated value of the actual lead time, as predicted by the ML-models based on available input features.
- **Baseline:** In this thesis, the baseline refers to the lead time estimates currently provided by suppliers and used by Epiroc in their operations. It serves as a reference point against which the performance of the ML models is evaluated.
- **External Supplier:** Refers to a third-party company that delivers goods or components to Epiroc from outside the organization.

1.4 Purpose

The purpose of this thesis is to investigate the feasibility and effectiveness of using ML to improve accuracy of external lead time predictions. Reliable lead time forecasting is essential in SCM, since inaccurate estimations will lead to increased operational costs and production issues. To address this, this thesis will implement and evaluate a data-driven method for forecasting supplier lead times. The performance of these models will be evaluated against actual lead times and compared to the accuracy of the supplier-provided estimates.

1.5 Goal

This thesis explores the use of state-of-the-art machine learning algorithms to predict actual lead times based on historical and situational data. Instead of focusing on one ML model and optimizing its performance metric, three distinct models will be implemented to address the challenge. Which of the models are most suited for this type of regression task, will also be evaluated. The nature of this report will be exploratory rather than exploitative, primarily due to the absence of related research on the subject.

1.6 Research Questions

- Can a machine learning model accurately predict the actual lead time of a company's suppliers using historical lead-time data?
- Which of the selected machine learning models performs best in predicting supplier lead times, based on various metrics?

1.7 Methodology

In this thesis, a standard ML-pipeline [5] has been followed. It includes data acquisition, preprocessing, feature engineering, model development, and evaluation. Historical order and product data were used to train and test the models. The dataset were cleaned and filtered to ensure quality, and relevant features were engineered to further improve the models performances. Grid Search with Cross-Validation (GridSearchCV) was used to tune the hyperparameters, and the models were evaluated with MAE, RMSE, and R^2-score. For the NN, Cross-Validation (CV) was also utilized to assess how well the models generalized.

1.8 Delimitations

In this report there are several delimitations applied to narrow the scope of the project;

- Only the external supply chain will be incorporated in the research.
- Only materials ordered to the Parts and Service division at Epiroc Örebro will be considered.
- Only data from 2024 will be used in order to make sure the data is still relevant.
- This thesis will only consider three different models, to make the scope more manageable.

2 Literature Review

Recent research has begun to address the application of ML to predict lead times in external supply chains. During the literature review a few relevant research articles and theses were found. The research explained below uses similar methodologies when building ML models, even though they are utilized in different fields.

A study conducted in the German machinery industry investigates the utilization of ML regression models to predict delivery delays for low-volume, high-variety products [4]. The authors developed and evaluated models using real-world data, comparing various ML approaches. The results show that regression models can predict not only whether a delivery is going to be late or not, but how long that delay will be in number of days. The study highlights the usefulness of having accurate predictions early in the procurement process.

Another relevant study was conducted at a semiconductor manufacturing company [6]. This research compares several ML algorithms, such as Support Vector Regression, k-Nearest Neighbors, RF, and NN to forecast lead times of a complex production system. The research highlights the importance of combining both static features (product and customer information) and dynamic features (work-in-progress and equipment status) to improve prediction quality.

The use of ML to solve regression issues in real-life scenarios is widely used in various fields. The medical field has seen a significant increase in the use of ML algorithms to improve the efficiency of its processes. One particularly interesting study [7] relevant to this thesis employs a RF algorithm to predict the length of stay of patients based on a number of variables. It focuses on a regression problem to predict how many days a patient will spend in the hospital, meaning the output is a continuous numerical value as in this thesis.

Another interesting study [8] concerns using ML to make data-driven delay predictions in order to optimize supplier selection. In this research, ML algorithms such as RF and XGBoost are used to predict whether a supplier will deliver on time or not.

This thesis will explore the use of ML for predicting external lead times of the company's suppliers, using historical and relevant contextual data. The output of the model will be a continuous numerical value, which means it is a regression task. The literature has shown that for this kind of regression task, either XGBoost or RF are most suitable. Other prominent research has made it evident that NNs are also relevant to this kind task, which is why this will be tested as well.

Using NNs to predict continuous values has shown promising results, as shown in 'Estimating Real Estate Selling Prices using Multimodal Neural Networks' [9]. This thesis investigates the use of NNs to predict real-estate selling prices based on both tabular data and image data. The results show that NNs using tabular data can be highly effective at making accurate predictions regarding numerical values. This research is directly relevant to this thesis, as predicting external lead times produces a continuous numerical value, just as real-estate selling prices. The success of using NNs for estimating the prices, supports the notion that it could also be applied in a supply chain context.

3 Theory

This chapter will give context to the principles used in this ML project. Theory about ML models, methods, and other relevant subjects will be described to fully understand the content of the thesis.

3.1 Supply Chain Lead Times and Their Impact

The term lead time generally refers to the time between the start of a process and its completion [2][10]. However, its exact definition varies across disciplines such as manufacturing, software development, and supply chains. In the context of SCM, lead time typically denotes the duration between when an order is placed and when the goods are received [2]. This thesis specifically focuses on supplier lead time, defined as the time from when a purchase order is released until the production of the order is completed and ready for shipment.

Accurate estimation of supplier lead times plays a crucial role in several core supply chain functions:

- **Inventory Planning:** Safety stock levels are often calculated based on the expected lead time and its variability. Inaccurate lead time forecasts can lead to either material shortages which might lead to production stops, or excess inventory levels, which will generate costs [2]. Accurate inventory planning enables the avoidance of unnecessary costs and the reduction of operational inefficiencies.

- **Production Scheduling:** Supplier lead time deviations can disrupt production planning by delaying the arrival of critical components or materials. In companies that rely on just-in-time or lean production systems, even minor lead time delays can create bottlenecks that halt production [2]. These disruptions may lead to planners having to reschedule jobs, shift workloads, or idle machines and labor, all of which increase operational costs. Such uncertainties make it difficult for a company's ability to reliably fulfill their obligations [10]. In complex production systems, the disruptions may have effects downstream, turning initially minor issues into significantly larger problems.

- **Customer Service Levels:** Reliable customer service in supply chain operations is closely tied to the ability to meet promised delivery dates [10]. When a company has a hard time estimating its production lead times, downstream delivery commitment becomes hard to uphold. This can harm customer trust and satisfaction. In a business-to-business context, delayed deliveries may result in contractual penalties or cause disruptions in the customer's production schedules. Variability in supplier lead times may also reduce the accuracy of promised delivery schedules, making it harder for sales or planning systems to provide reliable order estimations [10]. To tackle this issue, many companies compensate by inflating promised lead times [3] to avoid late deliveries, but this creates inefficiencies like early deliveries and excessive inventory levels.

Traditional lead time estimate methods often rely on static supplier-provided values or historical averages [10]. However, these estimates fail to capture the complex dynamic patterns which influences the delivery accuracy, such as order size variability, geopolitical situation, or global supply chain disruptions. ML offers more sophisticated methods to predict lead times [11].

Modern supply chain strategies focus on flow by prioritizing speed, flexibility, and responsiveness to thrive in the increasingly complex global market [12]. Lead time variability is widely recognized as a major contributor to operational uncertainty and costs [13].

To reduce these uncertainties and maintain lean operations, it is essential to improve the accuracy of lead time predictions [11]. The traditional lead time estimate methods lack the ability to capture underlying patterns in complex relationships and often fail to adjust to contextual changes or supplier reliability [14]. ML is a powerful alternative to these static traditional methods.

3.2 Machine Learning

ML is a subcategory of Artificial Intelligence (AI) that focuses on the development of algorithms that are able to automatically identify patterns in data and make predictions without being explicitly programmed for each individual task [15]. By incorporating more data into the learning process, the model's ability to make accurate predictions increases. Usually, ML is divided into three categories:

- **Supervised Learning:** Involves training a model utilizing a labeled dataset, where both input features and output targets are known [16]. Commonly utilized for classification and regression tasks.
- **Unsupervised Learning:** Utilizes unlabeled data to identify underlying patterns, groupings, and structures within the dataset [17].
- **Reinforcement Learning:** Utilizes a learner that improves its decision-making by interacting with an environment and receiving feedback in the form of rewards or penalties [18].

This thesis uses supervised learning methods because the dataset available contains both input features and known outcomes.

One of the central challenges in ML is to achieve good generalization, meaning that the model performs well not just on the training data, but on new unseen data as well.

This is known as the bias-variance trade-off [16]. If a model has high bias, it is typically too simple and may not capture the underlying patterns in the data, leading to underfitting [17]. A model with high variance is overly complex and sensitive to noise, often resulting in overfitting. Balancing these two characteristics is crucial for building reliable and robust models.

3.3 Supervised Machine Learning

Supervised ML is a category of algorithms with the purpose of learning patterns from labeled data, unlike unsupervised ML where the data is not labeled [15]. The training data used by the model consists of examples that illustrate the relationship between input variable x and output variable y. A mathematical model will be used to fit the training data and then predict the output y of an unknown set of data, from which only x is known.

In many cases, providing arbitrary explanations of the relationship between the input and output may prove challenging. In these cases a supervised ML approach is to be advised. Supervised learning is distinguished by a dataset having labeled data, in other words the training data consists of an input-output pair [19]. The objective of the model is to learn a function which provides the correct output to the input, enabling accurate predictions on new, unseen data.

Mathematically, the training set is represented as:

$$\mathcal{T} = (x_i, y_i)_{i=1}^{n} \tag{1}$$

x_i stands for the input vector, y_i represents the corresponding output label, and n stands for the number of data points. The purpose of the dataset is to obtain as much information as possible from the training data. There are two types of supervised learning, classification and regression, which are distinguished by the type of output y. In a ML model which uses regression, the output is always numerical, while for classification the output is categorical [19]. The primary goal for a regression model is to learn to approximate the true function $f(x)$, such that:

$$y = f(x) + \varepsilon \tag{2}$$

where ε is an error term that visualizes the noise in the data. The output y_i is assumed to be a scalar value. For classification tasks, the goal is to predict categorical labels. Instead of getting a numerical value as in a regression task, the model is estimating the probability distribution:

$$P(y|x) = f(x) \tag{3}$$

$f(x)$ represents a function that outputs the probability of each class.

3.4 Tree-Based Methods

In this section different kinds of tree-based algorithms will be described. These different methods can be used for either regression or classification problems. The models are hierarchical, which creates a tree-like structure [15]. Each node represents a decision based on a specific variable, and when a node has no further splits its assigned value becomes the prediction. When making predictions, the model assigns a value based on the mean of the observations in that prediction region for regression, or the mode for classification tasks. Tree-based methods are generally used in situations when simplicity and interpretability are important.

3.4.1 Decision Trees

A decision tree is a supervised ML model which involves defining a set of rules that segments the predictor space into different regions [17]. There are two different types of decision trees, regression trees and classification trees. In this thesis, only regression trees are relevant, and therefore, they will only be described.

The term *decision tree* originates from its visual appearance of an inverted tree-like structure. It begins with one root node that is connected via branches to internal nodes, and when an internal node has no more branches, it is called a leaf, which indicates the end of that specific path. The initial step in building a decision tree involves splitting the predictor space at the root node, followed by additional splits at each internal node. As more nodes are added, the predictor space will be split into smaller non-overlapping regions, which can result in more precise predictions [17].

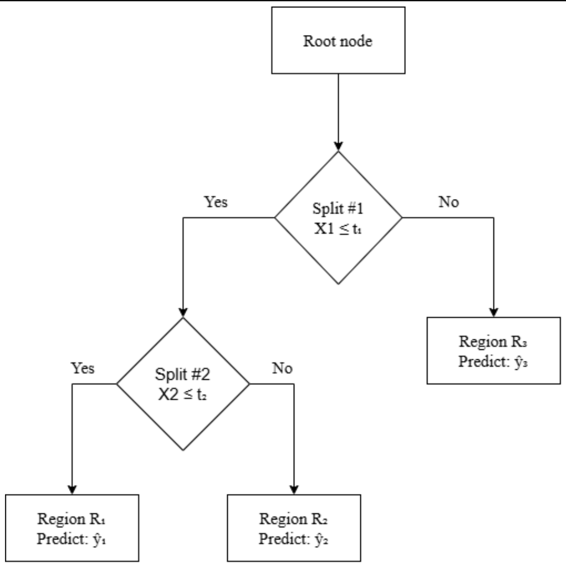

Figure 1 Example of a Regression tree. The tree's base is the root node and splits the predictor space using based on variables X_1 and X_2. Every decision sequence results in a region R_j, where a prediction is made using the average of the observations in that region.

In a regression tree, the predictor space is split into a set of specific regions. x_1, x_2,..., x_n represents the predictor variables [16]. Each x_i represents a specific value of a predictor variable that defines a region within the tree. The algorithm divides the predictor space into j regions, each of which will be referred to as R_1, R_2,..., R_j.

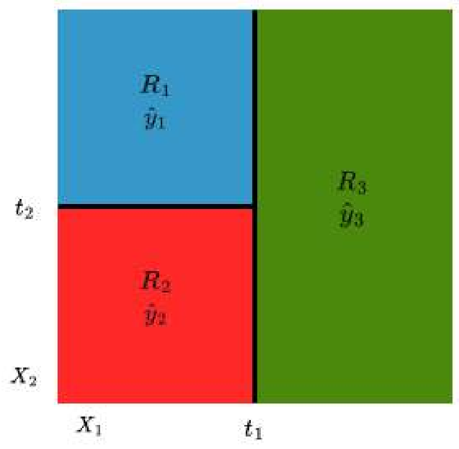

Figure 2 This simple predictor space reflects the regression tree in Figure 1. Every split of the predictor space is represented by a boundary in the two-dimensional space. It is divided into regions R_1, R_2, R_3, by the binary decision rules, where each region represents a leaf in the regression tree.

In each region R_j, the predicted output is given by the mean value of each observation that is within that region [17]. As an example, our predictor space consists of three regions, R_1, R_2, R_3. The mean response for the region R_1 is 55, for R_2 it is 50, and for R_3 it is 75. If a new observation $X \in R_1$, then X will be predicted to be 55, however, if $X \in R_2$ then X will be predicted to be 50, and if $X \in R_3$, the model will predict 75.

To simplify the interpretation of a regression tree, the different regions R_1, R_2,..., R_j can be divided into a set of high-dimensional rectangles. The goal is to find regions which minimize the Residual Sum of Squares (RSS), which can be defined as:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{4}$$

Here, $\hat{y}_{R_j}$ represents the mean response for the data points in R_j. Since it is not computationally feasible to evaluate all possible regions of the predictor space, one has to use a top-down, greedy approach known as recursive binary splitting. Initially, the predictor space is treated as a single region containing all data points. The first split occurs at the root node, dividing the predictor space into two regions. This process is repeated iteratively, making the regions smaller and more refined [15]. The splitting process consists of choosing a predictor variable X_j and a threshold s which results in the optimal split of the predictor space. The goal is to split the data into two separate regions that have a minimal residual sum of squares. The two regions which are a result from the split can be defined as:

$$R_1(j,s) = \{X \mid X_j < s\}, \quad R_2(j,s) = \{X \mid X_j \geq s\} \tag{5}$$

When applying the algorithm, the goal is to find the values of j and s that minimize the equation, and this is defined as:

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \tag{6}$$

$\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ represents the mean response value of the training observations in the regions R_1 and R_2. By choosing the predictor and split point that minimize the sum, one ensures that the variance in each region is as low as possible [16].

This process is iterated, continuously splitting the resulting regions to minimize the residual sum of squares. Instead of splitting the entire prediction space at every iteration, only one of the previously split regions is chosen for additional division. This continues until a stopping criterion is met, which has been decided on beforehand [17].

Once the stopping criterion is met, the final regions R_1, R_2,..., R_j are established. The training of the model is complete, and predictions can be made. To make predictions for new data, the model determines which region the observation falls into based on the learned splitting rules, and then assigns it the mean value, $\hat{y}_{R_j}$, of that region [15].

3.4.2 Random Forests

RFs are an improvement of Bagging, which combines a set of decision trees that have been trained on bootstrapped samples. The purpose of Bagging, also known as bootstrap aggregate, is to reduce variance, which is common when using decision trees. Bagging creates several trees, each trained on randomly selected subsets of the training data, and then combine their predictions into one final decision [16]. One common issue with bagged trees is that strong correlations between trees might occur, since they often select similar splits in each tree based on the same criterion, such as MAE in regression tasks. This is what RFs are aimed to solve. By introducing random elements when creating the decision trees, the correlations between the trees will be reduced [20].

When training a RF-model, instead of being able to choose from all possible input-variables x_1, x_2,...,x_p as in Bagging, each split is made using a random subset of variables, containing $q \leq p$ variables. This will be done at each splitting point, having a different subset of possible variables at each node. These random subsets of variables are done independently for the various trees in the ensemble. The trees in the ensemble are then averaged to produce a final prediction. By doing these steps, randomness is introduced into the model and in turn should reduce both overall model variance and correlation between the trees [16].

3.4.3 Gradient Boosting

Gradient Boosting is an improved version of traditional boosting algorithms, which train multiple weak ML models in sequence by iteratively improving the errors of their predecessors [21]. While the foundation of Gradient Boosting lies in the generic concept of Boosting, which revolves around the idea that even weak, high-bias models can capture some patterns in the data [17]. In contrast to Bagging, Boosting aims to reduce bias by building models in sequence rather than in parallel.

To intuitively grasp the concept of Gradient Boosting think of it as a method that builds models in sequence [21], where each one is trained to learn from the previous one's mistakes just as in generic Boosting. Each new model is then trained to correct the errors of the previous one by moving in the direction of the loss function's steepest descent, in other words the negative gradient [17]. By iteratively applying this process, the model becomes more accurate, and the error effectively decreases over time.

A widely used and more advanced implementation of this approach is XGBoost, which is discussed in the following section.

3.4.4 eXtreme Gradient Boosting

Much like the RF algorithm, gradient boosting trees are based on decision trees. XGBoost is a scalable and efficient implementation of gradient boosting. While traditional gradient boosting builds an ensemble of weak models sequentially, XGBoost implements several engineering and algorithmic optimizations [22], like regularization, parallel tree construction, and efficient handling of sparse data.

From a mathematical perspective, XGBoost is a powerful implementation of gradient boosting designed for efficiency and scalability [22]. Much like other gradient boosting algorithms it builds an additive model [23]. The final prediction is the sum of base learners;

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f(x_i), f_k \in (F), \tag{7}$$

In this context, $\mathcal{F} = \left\{ f(x) = w_{q(x)} \mid q : \mathbb{R}^m \to \mathcal{T},\ w \in \mathbb{R}^T \right\}$, where each $f(x)$ represents an individual regression tree. $q(x)$ defines the structure of the tree by assigning an x to a leaf index, and w determines what output value the model will produce. Each tree contains $\mathcal{T}$ leaves. The output of $f(x)$ which is a weighted sum of the basis functions, can be used as a prediction. The main objective is to find the $\phi(x_i)$ which minimizes the loss function:

$$J(f(X)) = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, f(\mathbf{x}_i)\right) \tag{8}$$

Where L represents some differentiable loss function. For regression tasks, the most popular choice would be Mean Squared Error (MSE), given by $L(y, \hat{y}) = (y - \hat{y})^2$.

However, when using XGBoost a regularization term is implemented into the objective function to avoid overfitting and improve generalization [23]. This term penalizes model complexity by punishing deep trees and extreme leaf weights. The final objective value represents the total loss of all trees, along with a penalty term for each tree to prevent overfitting [22]. To minimize the regularized objective function one uses the formula:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{9}$$

The regularization term promotes simpler trees and thereby reducing overfitting by implementing the following penalty term:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{10}$$

As in formula (A.1), $\mathcal{T}$ represents the number of leaves in the tree, w is the vector of leaf scores, γ is a parameter which regulates the number of leaves, while λ punishes the amount of leaf weights.

Since it is computationally unfeasible to directly optimize the space of trees, XGBoost uses the same additive training method as Gradient Boosting, where the trees are added sequentially to improve the errors of the previous tree [23].

3.5 Feedforward Neural Networks

A Feedforwad Neural Network (FNN) is the most fundamental type of NN. It may also be called a Multilayer Perceptron [24]. The name FNN originates from how the information moves in one direction through connected layers, from the input layer to the output layer. The concept of NNs lies in how the human brain operates. It consists of neurons and layers, connected via weighted links that transport signals from one layer to the next.

The goal of a FNN, like other ML algorithms, is to approximate some unknown target function $f^*(x)$ as accurately as possible [16]. The model learns by optimizing the parameters in order to achieve the best possible approximation of the target function. The target function can represent either a classification or regression task, however, within the scope of this thesis only a regression variant of FNN will be described.

A regression FNN can be seen as an extension of a linear regression model, capable of capturing more complex nonlinear patterns [25]. The output layer will typically consist of a single neuron with an activation function, which results in a continuous numerical value.

The network is fed with an input-vector $X = [x_1, x_2, ..., x_n]$, which is passed to the first hidden layer. Each neuron in the first layer computes the weighted sum of all the inputs [16], which is followed by passing the sum into a non-linear activation function.

The number of layers, L, in a FNN can be defined as $l \in [1, 2, ..., L]$. The computation done at each layer is [26]:

$$q^{(l)} = h(W^{(l)}q^{(l-1)} + b^{(l)} \tag{11}$$

Where $W^{(l)}$ represents the weight matrix linking the current layer to the next layer, $b^{(l)}$ is the layer's bias vector, $h^{(l)}$ is the activation function, and $q^{(l)}$ is the output vector of the layer.

To make it more concrete, a FNN containing one input-layer, two hidden layers, and one output layer will be used to illustrate the concept. The network can be expressed as follows;

$$q^{(1)} = h^{(1)}(W^{(1)}X + b^{(1)}) \tag{12}$$

The output of hidden layer 1 is subsequently passed to the next hidden layer as input-vector [26].

$$q^{(2)} = h^{(2)}(W^{(2)}q^{(1)} + b^{(2)}) \tag{13}$$

Similar computations are done with the weights, biases, and activation function for the second layer. Once the computations are done in the second layer, the output-vector is sent to the output-layer, which can be described as [26];

$$\hat{y} = W^{(3)}q^{(2)} + b^{(3)} \tag{14}$$

Here $b^{(3)}$ is a scalar bias term. It can be seen in this function that there is no $h^{(l)}$ present, this is because in regression tasks the output layer produces a continuous value which should remain untouched [16]. This represents a simple FNN, however, in theory the number of layers is arbitrary.

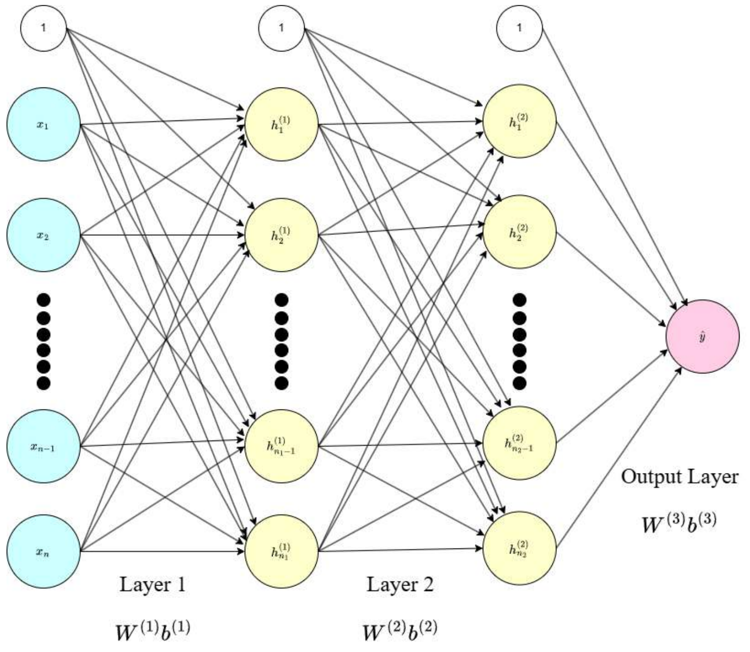

Figure 3 This image represents the network explained in equations 12 - 14. It contains one input layer, two hidden layers, and one output layer. This figure was inspired by the diagram in [16]. Out of each neuron in the first and second hidden layer, an output value will be gained. This is as previously mentioned, denoted as $q^{(l)}$.

3.5.1 Activation Functions

In a FNN, each neuron applies an activation function to its input value. This introduces non-linearity, which is essential in order to capture complex patterns in the data [27]. If no activation functions are present, the FNN would behave as a single-layer linear model. There are several activation functions available, Sigmoid, Tanh, and Rectified Linear Unit (ReLU) [27]. Sigmoid and Tanh functions are most commonly used in classification tasks, particularly in output layers, while ReLU serves as the most regularly used activation function in the hidden layers, regardless of the type of task.

3.5.2 Rectified Linear Unit

The Rectified Linear Unit, also known as ReLU, is one of the most common activation functions utilized in FNNs [24]. It is defined as;

$$ReLU(z) = max(0, z) \tag{15}$$

where z denotes the input value. In the context of a FNN, it translates to the following equation;

$$ReLU(z) = max(0, W \cdot x + b) \tag{16}$$

here W is the weight, x represents the input, and b is the bias. The ReLU function returns all negative values as zero, while positive values stay unchanged [27]. The simplicity of this activation function makes it more computationally efficient, and minimizes the vanishing gradient problem, which is a common issue in FNN models [24].

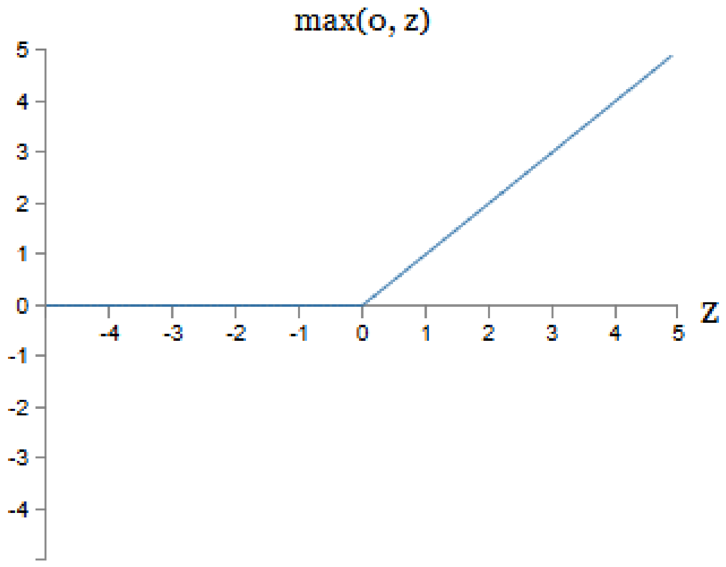

Figure 4 Visualization of the Rectified Linear Unit (ReLU) activation function. Source: [24].

3.5.3 Training a Neural Network

Training a FNN revolves around adjusting the model's parameters to minimize the prediction error. This process uses an objective function, also known as loss function, that measures the difference between the predictions compared to the true values [28].

When facing regression problems, the objective is typically defined by a loss function that measures the difference between output $\hat{y}_i$ and the true value y_i.

One of the most frequently used loss functions is MSE [29], described in equation 18. By squaring the error, the larger values will be more penalized, making the function more sensitive to outliers. However, when a problem does not require the same level of sensitivity to outliers [29], MAE is often utilized. MAE calculates the average of the absolute values of errors that a model produces, as explained in equation 20.

In FNNs, the minimization of the objective function is typically achieved by implementing gradient-based optimization methods [28]. The most common methods are Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam). A gradient based optimization algorithm, utilizes the gradient of the objective function to iteratively improve the model's parameters to reduce the error.

Before computing gradients, the network performs a forward pass, which is explained as sending the input data through the FNN to make a prediction [24]. The error calculated from this prediction is later used in the backpropagation method to update the model's parameters.

3.5.4 Backpropagation

To be able to implement the gradient-based methods mentioned in section 3.5.3, it is essential to compute the gradients of the loss function with respect to each parameter in the FNN. The backpropagation algorithm enables this [28].

Backpropagation is a method for efficiently computing these gradients by utilizing the chain rule [30]. The algorithm propagates backwards in the network, starting at the output layer where the error is computed. As the error is passed backwards in the network, the gradients of all weights and biases are computed, providing insight into how much each of the parameters is contributing to the overall loss [30].

During the backward pass, the network utilizes what has been learned during the forward pass, such as how strongly each neuron is activated or what inputs it received. These stored values will aid in helping the network understand how much each parameter influenced the final output and the required adjustments needed to reduce the error [28]. By utilizing this information, the model becomes more efficient as it avoids making redundant computations, making the training process significantly more efficient.

Backpropagation is not to be mistaken for an optimization method in itself, however, it serves as an important mechanism in computing gradients efficiently in algorithms like SGD or Adam. The optimization and regularization methods utilized in this thesis are described under appendix sections B to E.

3.6 Early Stopping

Early stopping is a widely used regularization technique in deep learning that helps prevent overfitting and improves generalization. This method works by monitoring the model's performance on a validation set during training and stopping the training process if the validation error begins to increase, which is an indication of overfitting [26].

This technique helps limit the complexity of the model, without changing the model architecture. The number of training steps can be interpreted as a regularization hyperparameter. By stopping the training process early, the model avoids fitting noise in the training data, therefore reducing the variance. In practice, this involves saving the optimal model parameters which results in the lowest validation error and restoring them at the end of the training process [26].

3.7 K-fold Cross-Validation

K-fold CV is a resampling method used to measure the performance of the model. The data is randomly split into k-number of equally sized subsets, also called folds [31]. In each iteration of the method, one of the folds is used as a validation set, also known as the held-out fold, while the model is trained on the remaining folds. In each iteration, the validation error is recorded and after all k-folds have been utilized as the held-out fold, the average of these errors is computed. This error is denoted as E_{k-fold}, which is an approximation of the model's test error on unseen data. The equation for the k-fold CV error can be defined as [16]:

$$E_{k-fold} = \frac{1}{k} \sum_{l=1}^{k} E_{hold-out}^{(l)} \tag{17}$$

where l denotes the specific iteration, $E_{hold-out}$ is the error, k is the number of folds the dataset is split into. The image below visualizes the process:

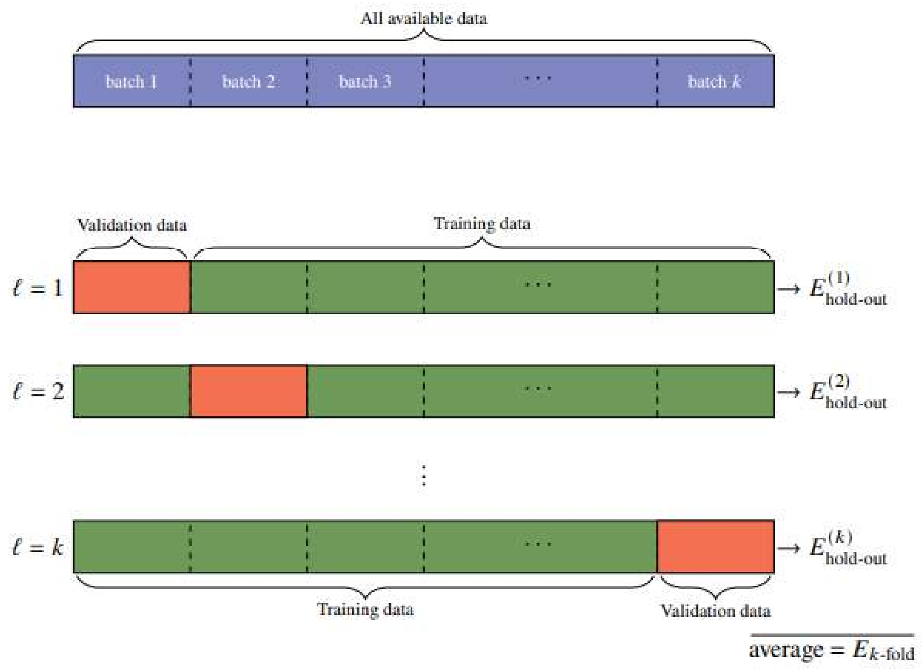
All available data
batch 1
batch 2
batch 3
· · ·
batch k
Validation data
Training data
$\ell = 1$
$\rightarrow E^{(1)}_{\text{hold-out}}$
$\ell = 2$
$\rightarrow E^{(2)}_{\text{hold-out}}$
$\ell = k$
$\rightarrow E^{(k)}_{\text{hold-out}}$
Training data
Validation data
average = $E_{k\text{-fold}}$

Figure 5 This figure visualizes how the dataset is split into k-folds for CV. The mean of each $E_{hold-out}$ summed together gives the estimated test-error on new unseen data. Image source: [16].

K-fold CV is commonly used since it eliminates the need to have a separate validation set. Instead, it allows for the model to train on the complete dataset, resulting in better utilization of the provided data [31]. Most commonly, the dataset is divided into 5-folds or 10-folds.

3.8 Model Evaluation

Once the model training is complete, it is essential to evaluate the model's performance using previously unseen and unbiased data. The evaluation metrics quantify how well the model has generalized outside the training data [29]. A wide range of metrics for ML are available, each suitable for different types of tasks. As this project is based on a regression problem, appropriate metrics for continuous predictions are used. For this project, it is important that the evaluation metric is measuring the distance between predicted and actual values, in other words, measuring the residuals produced by the model [29]. The metrics used in this project are MSE, RMSE, MAE, and R^2.

3.8.1 Mean Squared Error

MSE quantifies the average squared differences between the predicted and the true values in a dataset [31]. Squaring the residuals and then taking their mean of each one in a dataset results in the MSE. Once the MSE has been computed, it is used as an indicator of the model's predictive accuracy. Lower values indicate better model performance.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{18}$$

y_i represents the true value, $\hat{y}_i$ is the predicted value and N is the number of data points in the dataset. The summation calculates the residual for each data point, divided by the number of data points results in the MSE.

3.8.2 Root Mean Squared Error

RMSE is another commonly used metric when evaluating the performance of regression models. It is computed by taking the square root of MSE [29]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{19}$$

The notations are the same as those used in the definition of MSE, (18). Typically, RMSE is used when a more interpretable result is desired. It uses the same unit as the target variable, which makes the error easy to interpret. RMSE penalizes large errors more heavily [29].

3.8.3 Mean Absolute Error

MAE is a metric used to evaluate the accuracy of regression models. Similar to MSE and RMSE, it measures the average error in predictions [16]. However, instead of penalizing larger errors more heavily like MSE and RMSE, MAE computes the average of the absolute differences between predicted and actual values. It can be defined as [29]:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{20}$$

MAE is particularly valued for its interpretability, as it represents the average absolute error of predictions compared to the actual values in the same units as the target variable [29]. This makes it an accessible and intuitive choice for evaluating model performance.

3.8.4 Coefficient of Determination

The R^2 is an evaluation metric which assesses how well a model explains the variability present in the target variable [31]. It represents how well the variance in the target variable is explained by the model. The result ranges from $[-\infty, 1]$, where 1 indicates perfect predictions, a value of 0 indicates the model does not predict any better than a simple prediction based on the mean of the target variable [29]. If the value is below 0, it indicates the model performs worse than predictions made on the sample mean.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{21}$$

Here, the same notations in are utilized as in equation (18), $\bar{y}$ represents the mean of the actual values. The numerator represents the unexplained variance, also referred to as the residual sum of squares. The denominator is the total variance in the data, also known as total sum of squares [29].

4 Method

Initially, the scope of the project was determined by a basic understanding of the problem. However, as the problem was explored and discussed with all involved parties, a clearer understanding emerged. This led to iterative changes to the project's scope until a final problem description could be agreed upon.

This iterative process continued throughout the project to achieve the best possible outcome. To ensure a scientific approach, a widely accepted methodology from the field of ML was adopted [32]. Specifically, the methodology followed was a general, well-established ML-pipeline framework [5]. The following eight steps formed the foundation of the research:

1. **Data Collection:** The initial step in the ML-pipeline, involves gathering the data required to train the model. This data can be collected from various sources, such as a company's internal database, data warehouses, or external APIs. At this stage, the data is typically in its raw and unstructured format, and may contain inconsistencies, missing values, or noise. Therefore, preprocessing is required to ensure that the data is suitable for analysis and model training [5].

2. **Data Preprocessing:** Usually, the raw data obtained are not directly usable, which is why it requires cleaning, transforming, and preparing for modeling [33]. This step consists of handling missing values, encoding categorical features, scaling numerical values and splitting the dataset into training, validation and testing sets.

3. **Feature Engineering:** In this step, new features are created based on the original features in the dataset [32]. These new features are aimed to improve the model's predictive ability.

4. **Model Selection:** Based on the problem description, appropriate ML algorithms are chosen. Which algorithm is chosen depends on whether it is a classification or regression task, data characteristics, and performance requirement [32].

5. **Model Training:** In this phase, the chosen ML algorithm learns patterns using the training dataset. The model learns the patterns and relationships by optimizing its parameters to minimize the errors [16]. Different algorithms have different ways of learning the patterns, it can involve techniques such as decision trees, gradient descent or backpropagation. A validation set is used to evaluate the model's performance during training.

6. **Model Evaluation:** Once the models are trained, the performance will be evaluated using the test dataset and k-fold CV. It may involve using metrics such as

RMSE, accuracy and precision [29]. It uses the test dataset to test the model on unknown data to see how well it performs.

7. **Model Deployment (out of scope):** Once the model achieves a satisfactory performance, it is implemented into the real-life setting to make predictions [32]. This is outside of the scope of this thesis.

8. **Monitoring and Maintenance (out of scope):** When the model has been deployed, it must be monitored regularly to ensure the quality of the predictions is up to par. Over time, the data used to train the model may become obsolete, which will reduce the performance of the model [32]. This is, however, outside of the scope of the project.

As previously mentioned, an iterative approach was adopted during the development of the ML models, and it was concluded once the performance metrics reached a level the customer deemed acceptable for company utilization.

As illustrated in figure 6, the scope of the thesis is confined within the bounds of the ML-pipeline. The two last phases; Model Deployment and Monitoring and Maintenance fall outside of the scope and are therefore not addressed.

The process of building a ML model is inherently iterative. At first, the ML-pipeline follows a sequential workflow. While the pipeline initially follows a sequential flow, the results of model evaluation might reveal discrepancies in its performance [32]. As a counter-measure, the process may loop back to earlier stages, specifically: data preprocessing, feature engineering, and model training. By adjusting the input data, creating new features, or tuning hyperparameters, the performance might be improved. These specific steps were chosen for iteration because they can directly increase the results of the models, without changing the scope of the thesis.

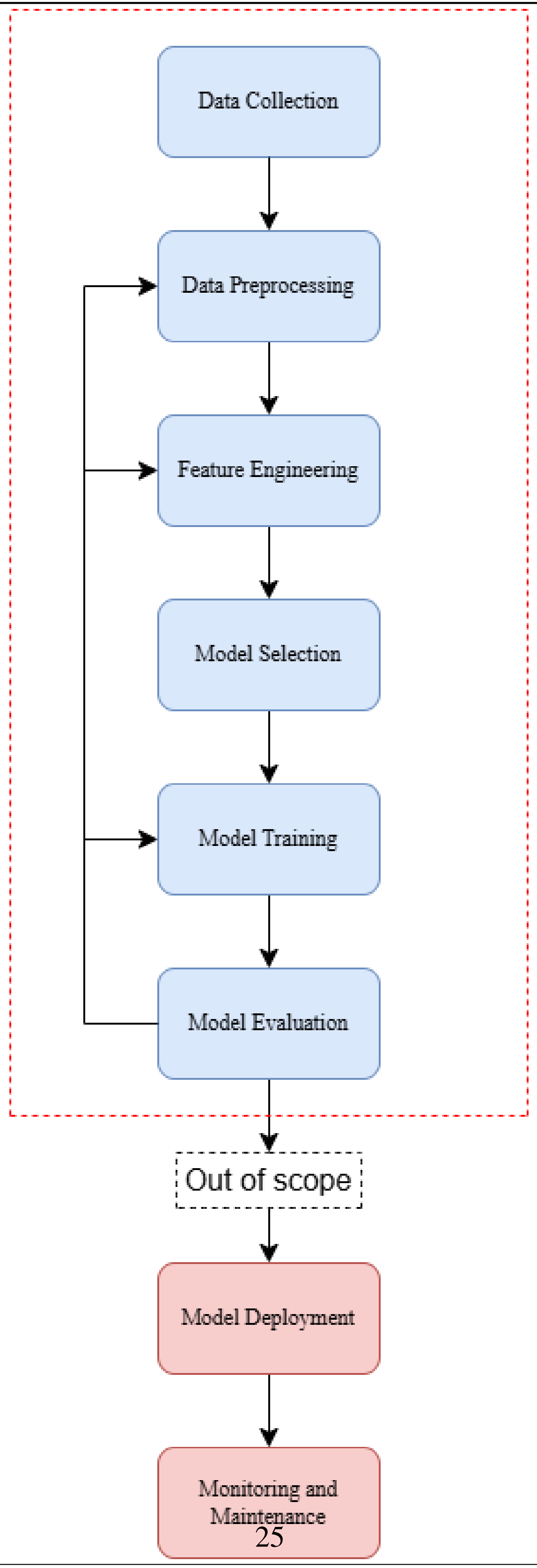

Figure 6 Visualization of the full pipeline used to build and train the models in this thesis.

The programming language used to develop the ML models was Python, with development conducted in Visual Studio Code (VSC). Several libraries and frameworks were used in model design, such as PyTorch, Scikit-learn, Pandas, Numpy.

4.1 Acquisition and Exploration of Data

To obtain the data used for developing the ML models, close collaboration with Epiroc was required. The data was not publicly available, thus it had to be extracted directly from the company's internal information system. Initially, Epiroc exported an order dataset which contained 29 unique features, including order numbers, agreed lead times, actual lead times, order dates, quantities, and other related attributes. A second dataset was acquired, containing product-specific information, holding features such as product IDs, product names, product categories, weight, among others. Both datasets were provided in tabular format.

Merging these two datasets into one was necessary to make it suitable for ML use. The order dataset was used as the base, and the product dataset was merged by using the product ID as identifier. This integration ensured that each row in the dataset contained relevant product information, providing the model with additional features to learn from.

Following the merge, the number of features amounted to 39 features. The order data consisted of approximately 400 000 distinct orders spanning from 2022 to early 2025. Each row represents a unique order and includes information such as whether the order was completed on the agreed delivery date, the actual lead time, and the quantity ordered.

4.2 Preprocessing

A significant part of developing ML models involves data preprocessing. At the beginning of most projects, datasets are typically unstructured, noisy, or incomplete, and therefore require cleaning and formatting before they can be used for ML purposes [32].

It is important to note that this phase is highly dependent on the nature of the problem and the structure of the available data. Each ML task may present unique challenges, requiring specific preprocessing steps based on the specific structure of the dataset.

4.2.1 Handling Inconsistencies

The first step in the preprocessing process involved cleaning the dataset by converting dates to standardized date-time format and handling inconsistencies, like varying decimal formats in numeric fields. It also involved standardizing column names by converting all headers to lowercase for consistency.

4.2.2 Time-frame Filtering

The dataset acquired from Epiroc spanned the years 2022 to 2025. These years differed greatly in a supply chain perspective. In 2022, global supply chains were significantly disrupted due to the Covid-19 pandemic, which led to widespread lockdowns and disruptions in logistics and production networks [1]. In more recent years, specifically 2024, supply chains have begun to stabilize, resulting in shorter and more consistent lead times.

Given this context, the data acquired from the years 2022 to 2023 are less representative of the current logistics environment. Therefore, a filter was applied to limit the training data to only include data from 2024.

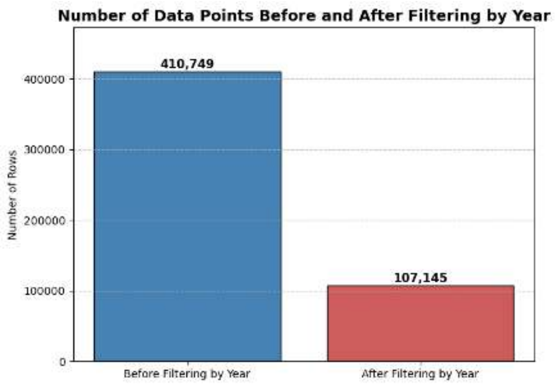

Figure 7 Visualization of the number of data points before and after applying the year-based filter. The dataset was reduced from 410,749 to 107,145 inputs after removing inputs outside the target year range.

4.2.3 Missing Values

When working with ML algorithms, it is generally impossible to utilize rows which contain cells with missing values. Either the missing values needs to be imputed, or removed from the dataset [34]. In this study, imputation was avoided since the missing values in the dataset were found in features such as product weight and supplier number, which are specific and unique to each row. These characteristics made imputation impractical and potentially misleading.

Rows that contained missing values in the product weight field were therefore dropped. Additionally, there were several companies lacking specific supplier numbers. In these cases, missing values were resolved by mapping them based on their specific supplier name.

4.2.4 Frequency Threshold

To improve the robustness and the ability to generalize of the ML model, suppliers with a very low number of historical orders were excluded from the dataset. Suppliers with limited data could introduce noise and outlier behavior, which might cause the model to overfit and worsen its prediction capabilities. Therefore, a threshold of 100 orders was put in place, to minimize the variance introduced by these low frequency suppliers.

4.2.5 Outlier Detection

In large datasets, as the one used in this thesis, it is highly probable that there are outliers in the data. These outliers consist of extreme values that do not represent the patterns within the data. To improve model reliability, Z-score outlier detection was applied to the target variable, *Actual Lead Time*.

Observations with a Z-score value larger than 2 were deemed as outliers, and subsequently removed from the dataset. The value 2 was chosen as a threshold since it results in approximately 95% of the data, with the assumption that the data follows a normal distribution. By removing the extremities, the model will be trained on data which better represents the underlying patterns, and reducing the risk of the model being skewed by anomalies.

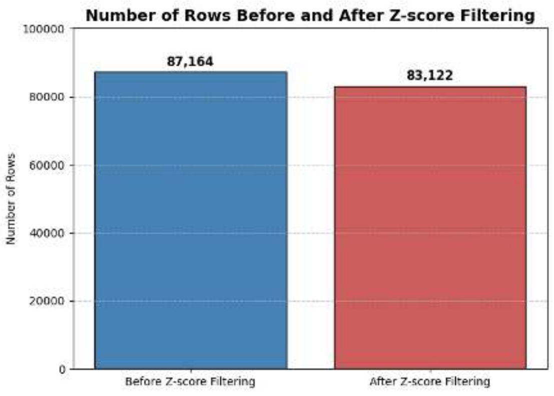

Figure 8 Number of rows before and after applying Z-score filtering. The dataset were reduced from 87,144 to 83,122 rows by removing statistical outliers.

4.2.6 Feature Engineering

In the field of ML, enhancing the dataset through the creation of new features by using existing ones is a valuable strategy for improving model performance. The first engineered feature was time-based and named 'Planned Delivery Month'. This variable converted the order date into a monthly format to capture underlying seasonal or monthly patterns in lead times.

To further enhance the model's ability to predict the actual lead time, features were introduced to quantify the historical variability of both suppliers and product groups. These new features indicate how consistent a specific supplier and product category has been historically.

4.2.7 Categorical and Textual Features

ML models require numerical input values, therefore making it necessary to transform the categorical and textual features into a numerical format prior to training. For categorical variables, label encoding was employed. This technique assigns each unique category a specific integer, allowing the feature to be used effectively by the model. Label encoding was applied to the following features: *supplier country, supplier number, item group, and planned delivery month*.

For textual features such as *stock availability code* and *dispatch advice*, binary mapping was used. These variables were converted into binary values (0 or 1) based on their intended meaning, enabling the model to interpret them during training.

4.2.8 Feature Reduction

The large number of features in the acquired dataset required strict filtering. Many of the original features were found to be redundant, irrelevant, or correlated with others, which could lead to overfitting or reduced model performance [31].

To determine which features should be retained in the preprocessed dataset, both correlation analysis and feature importance scoring were applied. A correlation matrix was used to identify heavily correlated features, giving justification to their removal. To analyze whether a feature contributes with usable information when training the model, feature importance scores were used.

These were obtained during the training of preliminary models. In the final model, various combinations of features were evaluated to identify the most relevant subset. The use of both statistical and model-driven techniques provided sufficient evidence for selecting the most informative features.

By reducing the number of features, the training of the model becomes more efficient, less noise is implemented, and improves generalization.

4.3 Selected Features for Model Training

After preprocessing and feature engineering, a set of features were carefully selected. These features were chosen based on domain knowledge, data availability, and their potential to improve predictive performance. The final set includes both original and feature engineered variables:

- **Agreed lead time:** The annually fixed lead time estimated by the supplier.

- **Dangerous goods binary:** A binary feature which indicates if the product is classified as dangerous goods.
- **Stock availability code binary:** Indicates whether an item is available for immediate delivery or if it has to be ordered.
- **Weight:** The weight of the individual product in the order.
- **Quantity:** The number of units ordered.
- **Supplier country (encoded):** Displays which country a supplier is located in. It is encoded numerically.
- **Item group (encoded):** Encoded label representing the item's product group.
- **Supplier number (encoded):** A unique identifier for each supplier. It is numerically encoded.
- **Supplier mean lead time:** Historical average lead time per supplier, representing variability in past deliveries.
- **Item group mean lead time:** The historical average for each item group.
- **Planned delivery month (encoded):** Extracted from the order date to capture seasonal patterns.
- **Supplier lead time standard deviation:** The historical variability of lead times for each supplier. It captures how reliable and consistent the specific supplier has been.
- **Item group lead time standard deviation:** The historical variability of lead times within each product group.

The target variable for all models was the actual lead time, which is measured in days between the order date and when the supplier has finished production.

4.4 Model Selection

The models selected for this thesis are RF, XGBoost, and FNN. These models were chosen based on the characteristics of the dataset and their demonstrated performance in similar regression tasks, mentioned in section 2. The data consists of tabular, structured features including order date, quantity, weight, agreed lead time, supplier number, and historical lead time variance. To predict the actual lead time as accurately as possible, appropriate models had to be chosen.

RF was selected due to its ability to handle a mix of categorical and numerical data, while also being robust to noise and capable of capturing non-linear patterns [20]. It may also provide the ability to see the importance of specific features, which is useful for interpretability. XGBoost was included as it typically shows high accuracy in structured data problems, and its use of boosting to correct errors from earlier iterations. While RF is typically easier to utilize, XGBoost allows more control and often better performance on complex problems when properly tuned [22].

FNNs were used to evaluate whether a more flexible and high-capacity model could find patterns in the data that tree-based models might miss. FNNs can model complex, non-linear relationships and are effective when sufficient amount of data is available, which was the case in this problem [24]. Techniques such as dropout and early stopping were utilized to prevent the model from overfitting during training.

Other models such as linear regression or time series models were not selected. Linear models assume a linear relationship between features and the target [16], which is highly unlikely in lead time scenarios. Time series models like ARIMA are designed for continuous time-dependent data, which does not apply here since each order is treated as an individual event [35].

4.5 Model Training

This section describes the methodology used to train the three ML models developed during this thesis. RF, XGBoost and a FNN were trained to predict supplier lead times using various features. To ensure that the models would perform well, the following steps were carefully considered; Data splitting, model creation, and hyperparameter optimization.

4.5.1 Data Splitting

The final dataset, after completing the preprocessing, consisted of 83077 data-points. In ML, the size of the dataset plays a crucial role in the model's ability to recognize underlying patterns effectively [36]. To avoid overfitting and to ensure model reliability, the dataset was divided into three separate subsets: training set, validation set, and test set. The validation set prevents overfitting by controlling the error during training, while the test set gives an unbiased evaluation about the model's performance.

For XGBoost and RF, the dataset was split into three partitions, the training set (70%), the validation set (15%), and the test set (15%). In this thesis, it was determined that 83077 data-points, split into the three subsets, combined with early stopping was sufficient to prevent the model from overfitting but still recognize patterns in the data. To avoid unnecessarily high computational costs, k-fold CV was not utilized, since the dataset's size and separate validation set were considered sufficient to make sure the model was reliable.

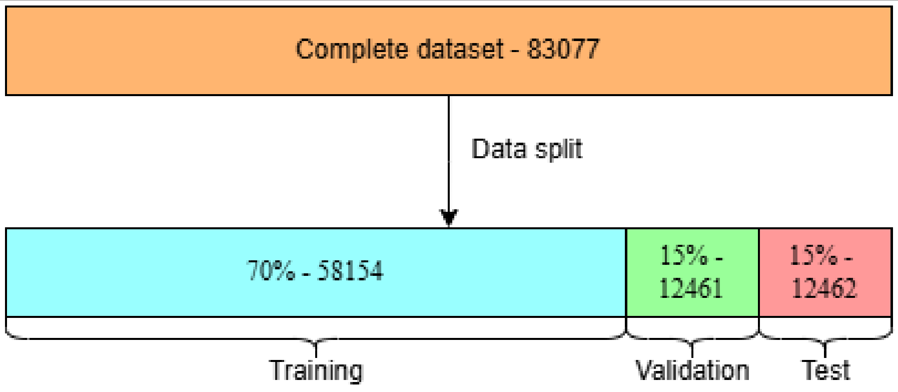

Figure 9 Represents the splitting of the complete dataset, into subsets for training, validation and testing of the model. [9].

FNN typically demand large quantities[37] of data to efficiently achieve a generalized model. Therefore, to further introduce reliability into the model, 5-fold CV was utilized, as described in section 3.7. Initially, the dataset was split into a training set (85%) and a test set (15%). During training, the training set was further divided into 5 folds, resulting in approximately 17% of the full dataset in each fold.

In each iteration, one fold served as the validation set while the remaining folds were used for training. In the next run, a new fold was used for validation. This was done iteratively until each fold had been used once as validation set.

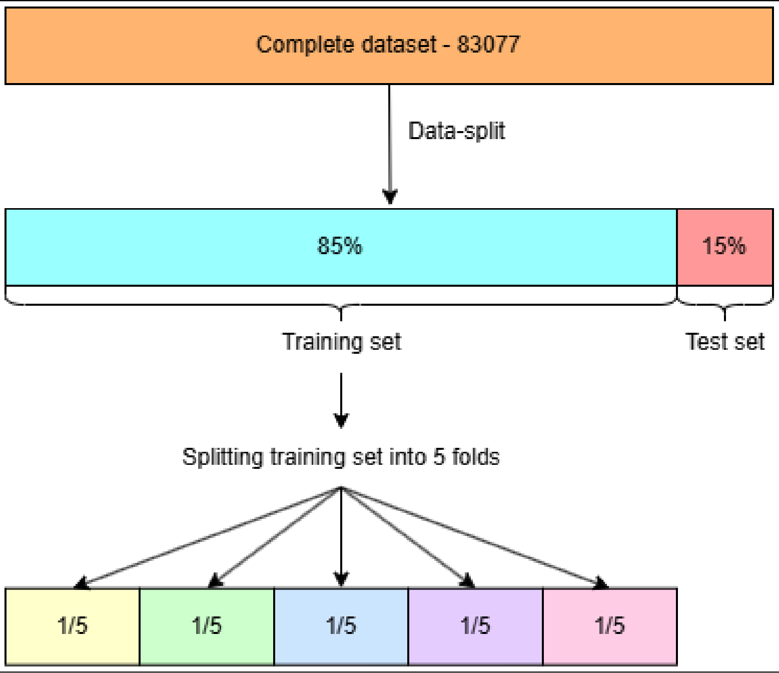

Figure 10 Represents the splitting of the dataset into 5 folds, preparing for k-fold CV. Inspired by [9].

4.5.2 Model Creation

After preprocessing and splitting the data, multiple models were created and evaluated. Each model was designed to predict the actual lead time based on a carefully selected set of input features.

The RF model employed out-of-bag estimation to evaluate how well the model was performing during training. The hyperparameter tuning process for this model is described in section 4.5.4. The evaluation was done using metrics, such as MAE, RMSE and R^2.

For XGBoost, early stopping was implemented to prevent overfitting by monitoring the validation error during training. If the validation error began to increase, the training was automatically stopped, as it indicated that the model was starting to overfit the training data [26].

The FNN had a two hidden layer architecture, each layer utilizing ReLU activation functions. To avoid overfitting, dropout regularization and early stopping were implemented [26]. The model was trained to minimize the MAE (L1-loss), and Adam optimizer was used to optimize the parameters during backpropagation.

4.5.3 Hyperparameter Tuning

Before training a ML model, an initial set of hyperparameters must be selected. These settings determine how the model is trained and can significantly influence the model's performance. Therefore, hyperparameter tuning is a crucial step in the modeling process. Hyperparameters can vary depending on the algorithm, for example, in tree-based algorithms, they may include the tree depth or learning rate, while in FNNs, they may refer to the number of layers or neurons per layer [16].

To optimize the models' hyperparameters for each model used in this thesis, GridSearchCV method was employed. This approach involves defining a grid of hyperparameter values and evaluating all possible combinations. GridSearchCV then applies CV to provide a more reliable performance estimate of each combination. Finally, the set of hyperparameters that yields the best performance, based on the chosen error metric, is selected.

4.5.4 Hyperparameters - Random Forest

For every method, there are specific hyperparameters that can be chosen. In this section the chosen hyperparameters for RF will be described [38]. Four hyperparameters were chosen, n_estimators, max_depth, min_samples_split, and min_samples_leaf.

- **N_estimators:** This hyperparameter decides the number of trees in the model. Typically, a higher number will generate better performance, but the model then becomes more computationally expensive.
- **Max Depth:** Determines the maximum number of levels in each tree. This prevents each individual decision tree from overfitting.
- **Min samples split:** The minimum number of samples needed to divide a node into two child nodes.
- **Min samples leaf:** This describes the least number of samples that are required in a leaf node. For a split to be considered at a node, it has to have at least the amount of samples as this value.

Parameter	**Values**
`n_estimators`	600, 800, **1000**
`max_depth`	20, **40**, None
`min_samples_split`	**2**, 5, 10, 15
`min_samples_leaf`	**1**, 2, 5, 10

Table 1 The various hyperparameters tested in the GridSearchCV for RF.

4.5.5 Hyperparameters - XGBoost

The following hyperparameters were selected for tuning the XGBoost [39]: n_estimators, max_depth, learning_rate, subsample, and colsample_bytree.

- **n_estimators:** This hyperparameter has the same function as n_estimators in section 4.5.4.
- **max_depth:** Max_depth in XGBoost is the same as in RF, described in section 4.5.4.
- **learning_rate:** Determines how much each tree contributes to the prediction. A low learning rate will result in a slow learning process, and might lead to the model requiring more trees for good performance.
- **subsample:** This determines how large a part of the training data that is randomly sampled for building each tree. By lowering the value below 1, randomness is introduced which might result in reduced risk of overfitting.
- **colsample_bytree:** The proportion of features randomly selected in the construction of each tree. If the value is 1, each tree uses all features, however, if the value is 0.8, it utilizes 80% of the features. Reducing this hyperparameter prevents the model from relying on strong features, which in turn improves generalization.

Parameter	**Values**
`n_estimators`	600, 800, **1000**
`max_depth`	6, 8, **10**
`learning_rate`	0.03, **0.036**, 0.04, 0.05
`subsample`	**0.8**, 0.9, 1.0
`colsample_bytree`	0.8, **0.9**, 1.0

Table 2 Grid settings for hyperparameter tuning of a XGBoost model.

4.5.6 Neural Network

In this section, the various hyperparameters for FNN selected for GridSearchCV will be described [40]. They consist of lr (learning rate), batch size, hidden1 (hidden layer 1), hidden2 (hidden layer 2), and dropout.

- **Learning Rate:** The learning rate describes how much the weights of the model are updated during training. If a lower value is chosen, the model will converge at a slower pace but the training will be more stable.

- **Batch Size:** Determines how many training samples are used in each batch.
- **Hidden1:** This hyperparameter decides how many neurons the first layer consists of. More neurons in each layer results in a more complex model, which can make it recognize more advanced patterns, but is also more prone to overfitting.
- **Hidden2:** Determines the amount of neurons in the second layer.
- **Dropout:** Randomly disables a subset of neurons when training the FNN. Utilized to prevent overfitting and increase generalization.

Parameter	**Values**
`lr`	**0.001**, 0.0005, 0.0001
`batch_size`	128, **256**, 512
`hidden1`	64, **128**, 256
`hidden2`	64, **128**, 256
`dropout`	0.0, **0.1**, 0.3, 0.5

Table 3 Expanded parameter grid for FNN hyperparameter tuning.

5 Results

In this chapter, the performance of the three models will be visualized. The models' predictions will be compared to the baseline predictions currently used by the company, in order to evaluate if the models offer improvements. The optimal hyperparameters will be added, identified through GridSearchCV. The tables demonstrate the MAE, RMSE, and R^2 score, and the respective improvements compared to the baseline. The figures, 11 to 17, illustrate how the models perform compared to the baseline over time, where the y-axis represents the MAE and the x-axis is each month of 2024.

5.1 Random Forest

The following section shows how the RF model performs compared to the baseline values.

Parameter	**Value**
`n_estimators`	1000
`max_depth`	40
`min_samples_split`	2
`min_samples_leaf`	1

Table 4 The optimal set of hyperparameters for RF, determined by GridSearchCV.

Metric	**Value**	**Baseline Value**	**Improvement (%)**
MAE	5.70 days	12.53 days	54.5%
RMSE	9.41 days	22.81 days	58.7%
R^2-score	0.75	-0.48	–

Table 5 This table shows how the RF model performed, using MAE, RMSE, and R^2 as evaluation metrics. Percentage improvement is included only for MAE and RMSE, since R^2 measures model fit and not error, making percentage improvement inappropriate.

The RF model achieved a MAE of 5.70 days, which is a 54.5% reduction compared to the baseline value of 12.53 days. RMSE decreased by 58.7%, indicating a substantial overall reduction in prediction error. The R^2-score of 0.75 suggests that the model fits the data well. While its performance is strong, the RF model shows higher errors than both XGBoost and the FNN.

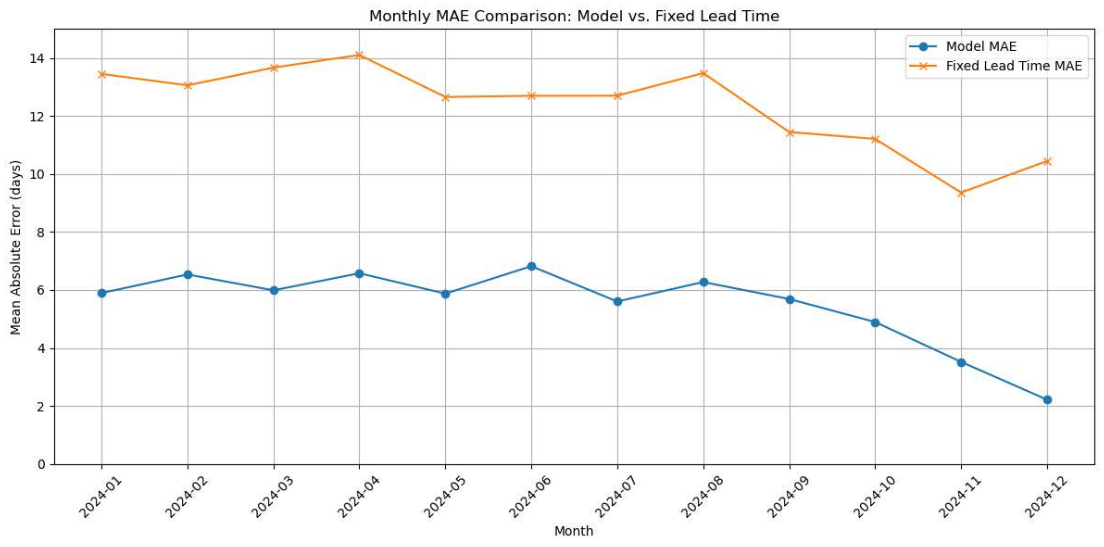

Figure 11 Model prediction error compared to the baseline error. MAE was used to compare the model predictions to the current lead time estimations. A lower MAE indicates better model performance.

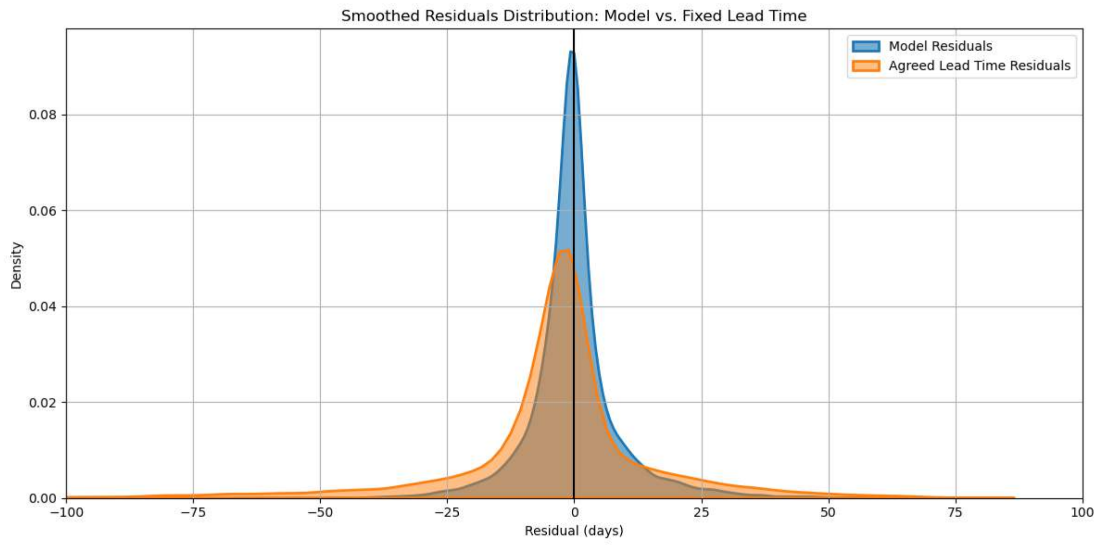

Figure 12 The figure presents the results from the RF in comparison to the agreed lead time. The model residuals (blue) are more centered around zero, than the agreed lead time residuals (orange). Indicating the model outperforms the baseline in terms of accuracy and reliability.

5.2 XGBoost

This section will visualize and explore the results gained from the conducted experiments.

Parameter	Value
`n_estimators`	1000
`max_depth`	10
`learning_rate`	0.036
`subsample`	0.8
`colsample_bytree`	0.9

Table 6 The optimal set of hyperparameters for XGBoost according to GridSearchCV.

Metric	Value	Baseline Value	Improvement (%)
MAE	5.07 days	12.53 days	59.6%
RMSE	8.69 days	22.81 days	61.9%
R^2-score	0.79	-0.48	–

Table 7 Metric values when using XGBoost to make predictions. Percentage improvement is shown for error metrics only.

The XGBoost model achieved a MAE of 5.07 days, reducing the baseline error by 59.6%. It also obtained the lowest RMSE among all models at 8.69 days, which is an improvement of 61.9% compared to the baseline. The R^2-score resulted in 0.79, indicating a strong correlation between predicted and actual lead times. The results gained from the experiment demonstrated that XGBoost provides a significant performance increase over the current fixed lead time estimations provided by the suppliers.

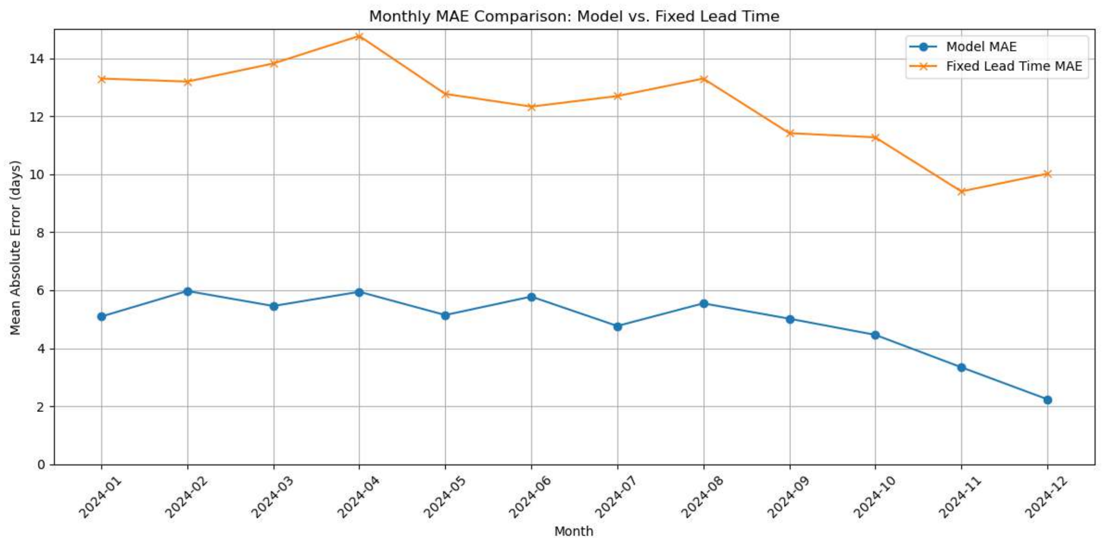

Figure 13 Visualization of the performance of XGBoost compared to the baseline performance.

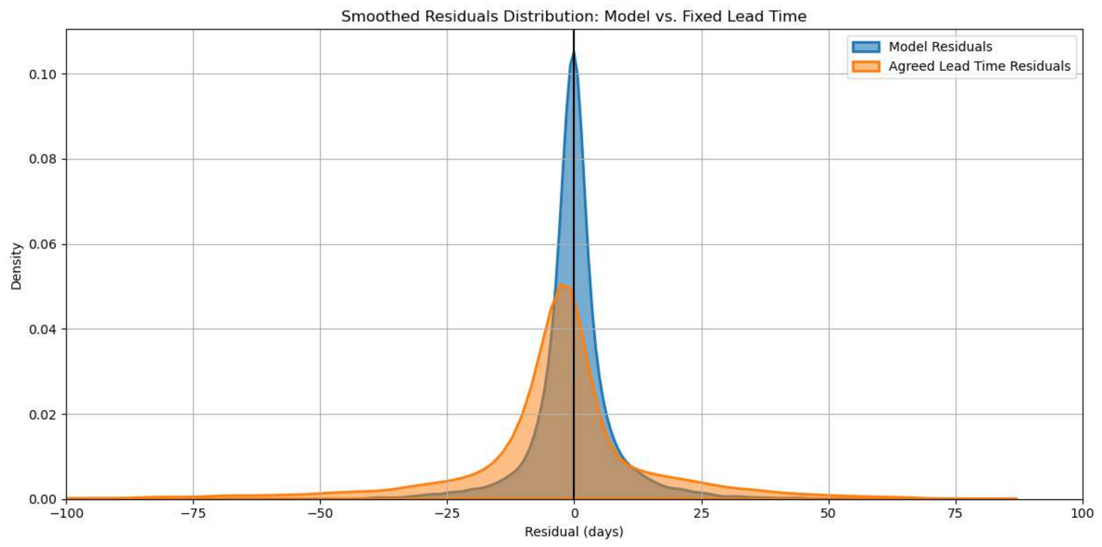

Figure 14 Smoothed residual distribution for the XGBoost model compared to the agreed lead time estimates. The model outperforms the baseline estimates, as the distribution is more tightly centered around zero. This indicates the model provide more accurate and consistent predictions.

5.3 Neural Network

In this section the results from the FNN model will be explored.

Parameter	Value
`Learning Rate`	0.001
`Batch Size`	256
`Hidden Layer 1`	128
`Hidden Layer 2`	128
`Dropout`	0.1

Table 8 The optimal set of hyperparameters for FNN according to GridSearchCV.

Metric	Value	Baseline Value	Improvement (%)
MAE	4.52 days	12.53 days	63.7%
RMSE	9.00 days	22.81 days	59.8%
R^2-score	0.76	-0.48	–

Table 9 Metric values when using FNN to make predictions. Percentage improvement is shown for error metrics only.

The FNN achieved the lowest MAE among the evaluated models, with a value of 4.52 days compared to the baseline of 12.53 days, resulted in a 63.7% reduction. The RMSE was also significantly improved, decreasing from 22.81 to 9.00 days, while the R^2 score of 0.76 indicates a strong fit between the model predictions and the actual values. These results suggest that the FNN consistently outperforms the baseline approach in predicting lead times across the dataset.

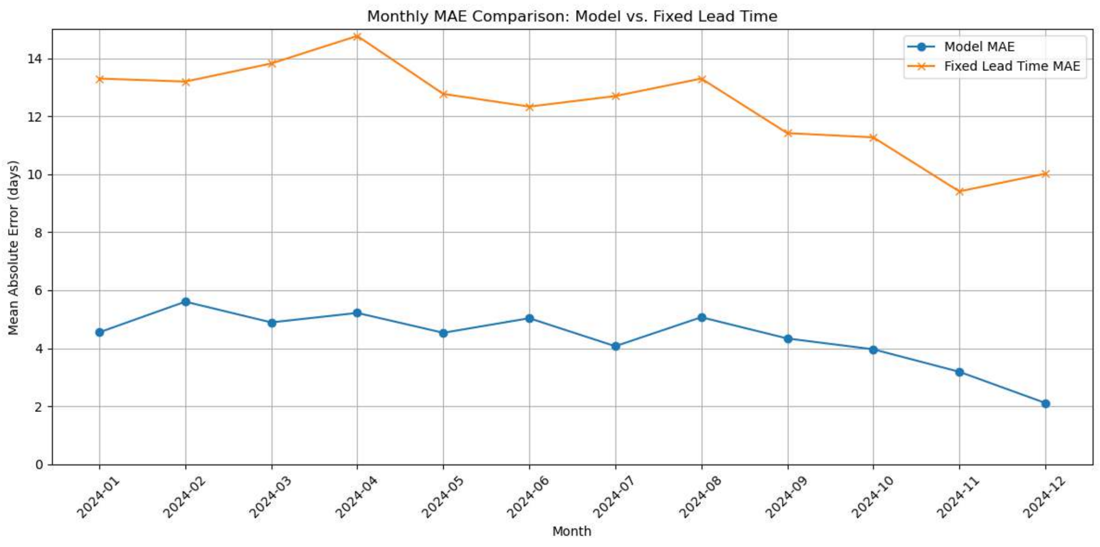

Figure 15 Visualization of the performance of FNN compared to the baseline performance.

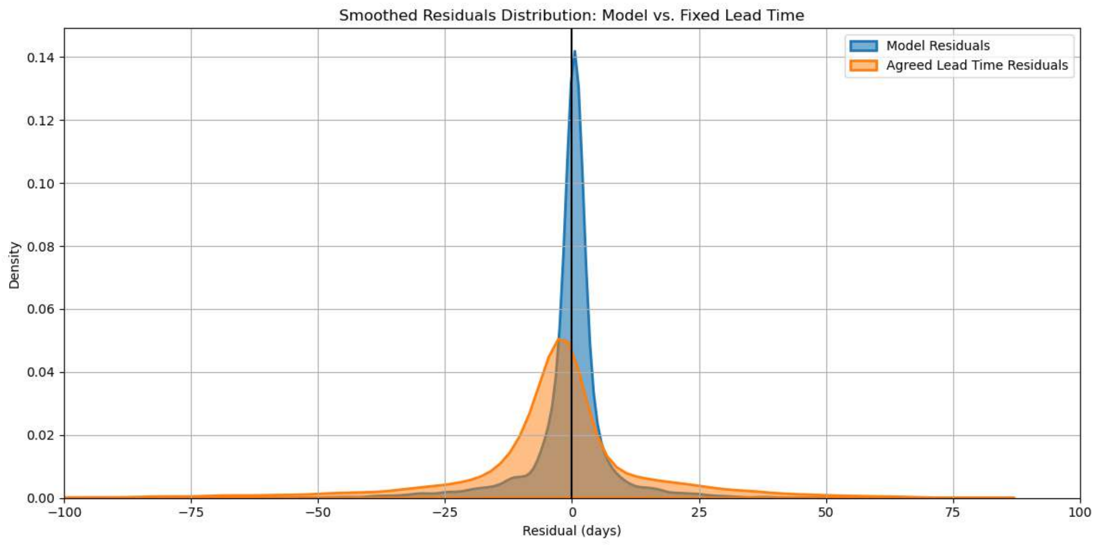

Figure 16 Smoothed distribution of residuals for the FNN model and the supplier provided lead time estimates. As is visualized, the model residuals (blue) are closely centered around zero, indicating a high prediction accuracy and lower variance compared to the broader distribution of the agreed lead time residuals (orange).

5.4 Overall Performance

This section summarizes the overall performance of the three models. The MAE is the primary metric of comparison, while RMSE and R^2 are also investigated to provide additional insights.

Model	**MAE**	**MAE Improv. (%)**	**RMSE**	**RMSE Improv. (%)**	R^2**-score**
FNN	4.52 days	63.7%	9.00 days	59.8%	0.76
XGBoost	5.07 days	59.6%	8.69 days	61.9%	0.79
RF	5.70 days	54.5%	9.41 days	58.7%	0.75
Baseline	12.53 days	–	22.81 days	–	-0.48

Table 10 Comparison of model performance metrics. Baseline values are shown for reference. Percentage improvement is shown only for error metrics.

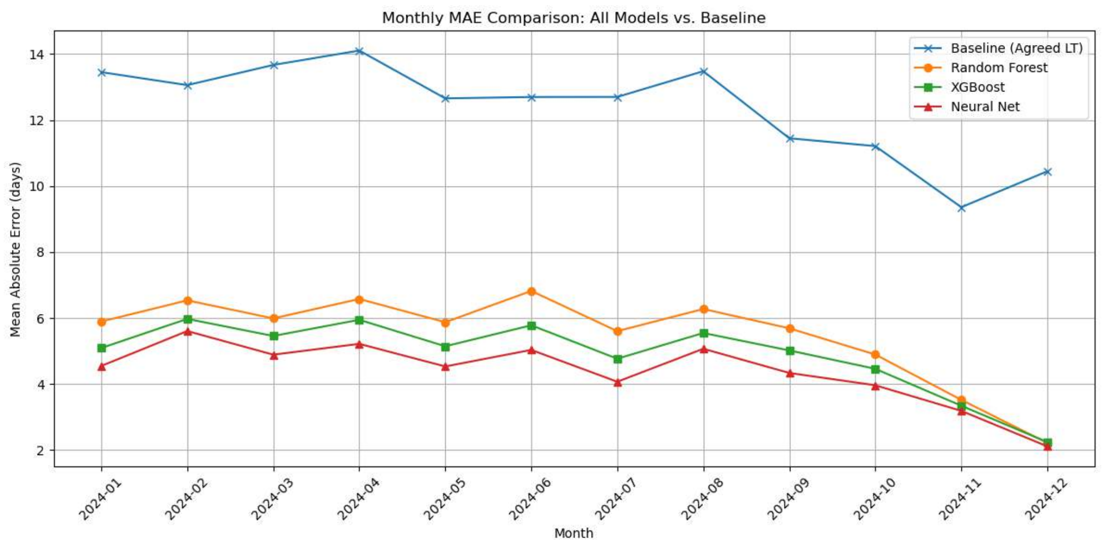

Figure 17 This image illustrates the overall performance of the models compared to baseline predictions. As can be seen, FNN is the best performing model.

6 Discussion

The objective of this thesis is to explore the feasibility and effectiveness of implementing ML to predict supplier lead times, with the end goal of making the supply chain more predictable and reliable. Three models are chosen, RF, XGBoost, and FNN, based on previous research done within the subject. They are developed and evaluated against baseline lead time estimations currently used by the company.

The results demonstrate that all ML models utilized within the bounds of this thesis significantly outperform the baseline estimations across all tested metrics. With regard to MAE, the FNN performs the best with a MAE-score of 4.52 days, marking an improvement of 63.7% over the baseline estimations. RF and XGBoost outperform the baseline by 54.5% and 59.6% respectively.

The performance improvements gained from the three models confirm the first research question: ML models can predict actual lead times with a high accuracy using historical and contextual data. These insights provide a clearer understanding of how the supply chain is likely to behave as global supply chains recover from the pandemic, where early or late deliveries can significantly affect the efficiency and cost of production systems. By minimizing forecasting errors, companies may balance inventory levels with demand to reduce the risk of material shortages, while avoiding excess inventory levels.

Regarding the second research question, the evaluation of the models reveal that all three of them are viable options for regression tasks in supply chain forecasting. The FNN model outperforms both tree-based models, suggesting the underlying nonlinear relationships in the data are better captured by the FNN. This is likely due to FNNs ability to capture complex, nonlinear relationships in the data that tree-based models may struggle to find. Supplier lead times are affected by a wide range of factors [3], such as order quantity, product type, supplier reliability, and global contexts, which may act in ways which are hard to capture with hierarchical or additive models. The flexibility of FNNs enables it to learn subtle patterns and relationships in variables [24], resulting in more accurate predictions, especially with a sufficiently large dataset.

XGBoost performs better than RF, likely due to its advanced gradient boosting algorithm and built-in regularization [39]. The difference between the FNN model and XGBoost model is 10.85% in terms of MAE, so in this case the FNN's result is better. However, in terms of RMSE, XGBoost performs 3.44% better than the FNN. In this thesis, MAE is chosen as the primary evaluation metric due to the easily understood results. While RMSE penalizes larger errors more heavily, it does not offer the same level of interpretability in this case as MAE, which represents the average prediction error in the same units as the target variable [29]. This makes it easier for stakeholders without a technical background to understand the performance of the models. Using MAE allows the results of the models to become directly actionable as the company uses the same unit, days, when analyzing lead times. As a result of focusing on the MAE, the recommended model, purely based on the evaluation, is using a FNN for predicting supplier lead times.

However, using a FNN can be computationally expensive [41], and requires careful hyperparameter tuning and large datasets to perform well. This makes it less practical in environments where data or computational resources are limited [37]. However, RFs and XGBoost models are less complex to interpret and easier to deploy. They also offer the ability to provide feature importance, which can aid in decision-making. Therefore, although the FNN out-performed the rest, XGBoost and RF remain strong alternatives for situations that require easy implementation and feature transparency.

6.1 Future Work

While the models in this thesis showed strong performance, there are several opportunities for further research and improvements. This thesis limited the number of ML algorithms to three, FNN, RF, and XGBoost, to keep the scope more manageable. Future work might involve ML algorithms such as LightGBM [42], CatBoost [43], or more advanced FNN architectures, which could offer improved results and efficiency. Second, while the provided dataset contains a sufficient amount of data [37], additional features would be necessary to train a more powerful and accurate model. These features may include global logistics indicators, global financial status, and more detailed supplier production data. Finally, although data from 2022 to 2024 is available, this thesis focused exclusively on 2024 due to the unpredictability in lead times during the years affected by the global pandemic. However, future work could explore the use of historical data from earlier years to identify long-term patterns.

7 Conclusion

This thesis explored the possibility of using ML models to predict the actual lead times of external suppliers. The historical data, consisting exclusively of order and product data, was used to train three models: RF, XGBoost and a FNN. Each model significantly outperforms the existing baseline estimations, with the FNN achieving the lowest MAE of 4.52 days, representing a 63.7% improvement. This result underscores the potential of using deep learning techniques to capture the complex patterns in supply chains.

The results obtained in this thesis reinforce the findings presented in section 2, Literature Review. Using ML models can significantly enhance a company's ability to forecast lead times, leading to more informed decision-making and reducing operational inefficiencies. The results highlight the value of historical and contextual data, while also emphasizing the importance of model selection, feature selection, and hyperparameter tuning. This work serves as a foundation for future studies that may incorporate larger datasets, alternative features, and additional ML models which might result in a more robust and accurate supply chain forecasting model.

References

[1] Queiroz MM, Ivanov D, Dolgui A, Fosso Wamba S. Impacts of epidemic outbreaks on supply chains: mapping a research agenda amid the COVID-19 pandemic through a structured literature review. Springer Nature Link. 2020;319. Available from: https://link.springer.com/article/10.1007/s10479-020-03685-7.

[2] Olhager J. Produktionsekonomi: principer och metoder för utformning, styrning och utveckling av industriell produktion. 2nd ed. Lund, Sverige: Studentlitteratur; 2013.

[3] Raj A, Mukherjee AA, Lopes de Sousa Jabbour AB, Srivastava SK. Supply chain management during and post-COVID-19 pandemic: Mitigation strategies and practical lessons learned. Journal of business research. 2022;142. Available from: https://doi.org/10.1016/j.jbusres.2022.01.037.

[4] Steinberg F, Burggräf P, Wagner J, Heinbach B, Sassmannshausen T, Brintrup A. A novel machine learning model for predicting late supplier deliveries of low-volume-high-variety products with application in a German machinery industry. Supply Chain Analytics. 2023;1. Available from: https://www.sciencedirect.com/science/article/pii/S294986352300002X.

[5] Struder S, Bui TB, Drescher C, Hanuschkin A, Winkler L, Peters S, et al. Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. Machine Learning and Knowledge Extraction. 2021;3:392-413. Available from: https://arxiv.org/abs/2003.05155.

[6] Lingitz L, Gallina V, Ansari F, Gyulai D, Pfeiffer A, Sihn W, et al. Lead time prediction using machine learning algorithms: A case study by a semiconductor manufacturer. Procedia CIRP. 2018;72. Available from: https://www.sciencedirect.com/science/article/pii/S2212827118303056.

[7] Mekhaldi RN, Caulier P, Chaabane S, Chraibi A, Piechowiak S. Using Machine Learning Models to Predict the Length of Stay in a Hospital Setting. In: Advances in Intelligent Systems and Computing. Springer Nature Switzerland; 2020. p. 202-11. Available from: https://www.researchgate.net/publication/341449887.

[8] Zaghdoudi MA, Hajri-Gabouj S, Ghezail F, Darmoul S, Varnier C, Zerhouni N. Collaborative and Integrated Data-Driven Delay Prediction and Supplier Selection Optimization: A Case Study in a Furniture Industry. Computers Industrial Engineering. 2024;197:21. Available from: https://www.sciencedirect.com/science/article/pii/S0360835224007113.

[9] Öijar Jansson A. Estimating Real Estate Selling Prices using Multimodal Neural Networks. KTH Royal Institute of Technology; 2023.

[10] Chopra S, Meindl P. Supply Chain Management: Strategy, Planning, and Operation. 6th ed. Pearson Education Limited; 2019. Available from: https://www.pearson.com/en-us/subject-catalog/p/supply-chain-management-strategy-planning-and-operation.

[11] Vandeput N. Data Science for Supply Chain Forecasting. 2nd ed. De Gruyter; 2021. Available from: https://www.degruyterbrill.com/document/doi/10.1515/9783110671124/.

[12] Handfield R, Linton T. Flow: How the Best Supply Chains Thrive. University of Toronto Press; 2022.

[13] Tang CS, Sodhi MS. Managing Supply Chain Risk. 2012th ed. International Series in Opeartions. Springer; 2012. Available from: https://www.statlearning.com.

[14] Burggräf P, Wagner J, Heinbach B, Steinberg F. Machine Learning-Based Prediction of Missing Components for Assembly – a Case Study at an Engineer-to-Order Manufacturer. IEEE Access. 2021;9. Available from: https://ieeexplore.ieee.org/document/9416418.

[15] Müller AC, Guido S. Introduction to Machine Learning with Python. 1st ed. Springer Texts in Statistics. O'Reilly Media, inc.; 2017.

[16] Lindholm A, Wahlström N, Lindsten F, Schön TB. Machine Learning - A First Course for Engineers and Scientists. Cambridge University Press; 2022. Available from: https://smlbook.org.

[17] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning. Springer; 2017. Available from: https://hastie.su.domains/ElemStatLearn/.

[18] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. 1st ed. MIT Press; 2018.

[19] Jiang T, Gradus JL, Rosellini AJ. Supervised machine learning: A brief primer. Behavior Therapy. 2020;51. Available from: https://doi.org/10.1016/j.beth.2020.05.002.

[20] Ibrahim M. Evolution of Random Forest from Decision Tree and Bagging: A Bias-Variance Perspective. Dhaka University Journal of Applied Science and Engineering. 2023;7. Available from: https://www.sciencedirect.com/science/article/pii/S2212827118303056.

[21] Natekin A, Knoll A. Gradient boosting machines, a tutorial. Frontiers in Neurorobotics. 2013;7. Available from: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2013.00021/full.

[22] Bentéjac C, Csörgo A, Martínez-Muñoz G. A Comparative Analysis of XGBoost. CoRR. 2019;abs/1911.01914. Available from: https://arxiv.org/abs/1911.01914.

[23] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. arXiv preprint arXiv:160302754. 2016. Available from: https://arxiv.org/abs/1603.02754.

[24] Nielsen MA. Neural Networks and Deep Learning. Determination Press; 2015. Available from: http://neuralnetworksanddeeplearning.com/.

[25] McInerney A, Burke K. A Statistical-Modelling Approach to Feedforward Neural Network Model Selection. Statistical Modelling. 2024;5. Available from: https://doi.org/10.1177/1471082X241258261. Available from: https://arxiv.org/abs/2207.04248.

[26] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. Available from: http://www.deeplearningbook.org.

[27] Dubey SR, Singh SK, Chaudhuri BB. Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing. 2022;503. Available from: https://www.sciencedirect.com/science/article/pii/S0925231222008426.

[28] Hagan MT, Demuth HB, Beale MH, De Jesús O. Neural Network Design; 2017. Available from: https://hagan.okstate.edu/nnd.html.

[29] Terven J, Cordova-Esparza DM, Romero-González JA, Ramírez-Pedraza A, Chávez-Urbiola EA. A comprehensive survey of loss functions and metrics in deep learning. Springer Nature. 2025;58. Available from: https://link.springer.com/article/10.1007/s10462-025-11198-7.

[30] Torralba A, Isola P, Freeman WT. Foundations of Computer Vision. Adaptive Computation and Machine Learning series. MIT Press; 2024.

[31] James G, Witten D, Hastie T, Tibshirani R. An Introduction to Statistical Learning: with Applications in R. Seventh printing ed. Springer Texts in Statistics. Springer; 2013. Available from: https://www.statlearning.com.

[32] Géron A. Hands-on Machine Learning with Scikit-Learn, Keras TensorFlow. O'Reilly Media, Inc; 2019.

[33] Kotsiantis SB, Kanellopoulos D, Pintelas PE. Data Preprocessing for Supervised Learning. International Journal of Computer Science. 2006;1. Available from: https://www.researchgate.net/publication/228084519.

[34] Emmanuel T, Maupong T, Mpoeleng D, Semong T, Mphago B, Tabona O. A survey on missing data in machine learning. Journal of Big Data. 2021;8. Available from: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00516-9.

[35] Shumway RH, Stoffer DS. Time Series Analysis and Its Applications. Springer; 2006. Available from: https://link.springer.com/book/10.1007/978-3-031-70584-7.

[36] Bishop CM. Pattern Recognition and Machine Learning. Springer; 2006. Available from: https://link.springer.com/book/9780387310732.

[37] Alwosheel A, van Cranenburgh S, Chorus CG. Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis. Journal of Choice Modelling. 2018;28:167-82. Available from: https://www.sciencedirect.com/science/article/pii/S1755534518300058.

[38] Scikit-learn Developers. RandomForestRegressor — scikit-learn documentation; 2025. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[39] XGBoost Developers. XGBoost Parameters Documentation; 2025. Available from: https://xgboost.readthedocs.io/en/stable/parameter.html.

[40] PyTorch Developers. torch.optim — PyTorch 2.1 Documentation; 2025. Available from: https://pytorch.org/docs/stable/optim.html.

[41] Thompson NC, Greenewald K, Lee K, Manso GF. Deep Learning's Diminishing Returns. IEEE Spectrum. 2020. Available from: https://spectrum.ieee.org/deep-learning-computational-cost.

[42] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. 2017;30. Available from: https://dl.acm.org/doi/10.5555/3294996.3295074.

[43] Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. CoRR. 2017;abs/1706.09516. Available from: http://arxiv.org/abs/1706.09516.

[44] Ajagekar A. Adam; 2021. Available from: https://optimization.cbe.cornell.edu/index.php?title=Adam.

[45] Kingma DP, Ba JL. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. Springer Nature. 2017;9. Available from: https://arxiv.org/abs/1412.6980v8?hl=pl.

[46] Salehin I, Kang DK. A Review on Dropout Regularization Approaches for Deep Neural Networks within the Scholarly Domain. MDPI. 2023;12. Available from: https://www.mdpi.com/2079-9292/12/14/3106.

A EXtreme Gradient Boosting

Much like the RF algorithm, gradient boosting algorithms are based on decision trees. XGBoost is a scalable and efficient implementation of gradient boosting. While traditional gradient boosting builds an ensemble of weak models sequentially, XGBoost implements several engineering and algorithmic optimizations [22], like regularization, parallel tree construction, and efficient handling of sparse data.

From a mathematical perspective, XGBoost is a powerful implementation of gradient boosting designed for efficiency and scalability [22]. Much like other gradient boosting algorithms it builds an additive model [23]. The final prediction is the sum of base learners;

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f(x_i), f_k \in (F), \tag{A.1}$$

In this context, $\mathcal{F} = \{f(x) = w_{q(x)} \mid q : \mathbb{R}^m \to \mathcal{T},\ w \in \mathbb{R}^T\}$, and each $f(x)$ represents an individual regression tree. $q(x)$ defines the structure of the tree by assigning an x to a leaf index, and w determines what output value the model will produce. Each tree contains $\mathcal{T}$ leaves. The output of $f(x)$ which is a weighted sum of the basis functions, can be used as a prediction. The main objective is to find the $\phi(x_i)$ which minimizes the loss function:

$$J(f(X)) = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, f(\mathbf{x}_i)\right) \tag{A.2}$$

Where L represents some differentiable loss function. For regression tasks, the most popular choice would be MSE, given by $L(y, \hat{y} = (y - \hat{y})^2$.

However, when using XGBoost a regularization term is implemented into the objective function to avoid overfitting and improve generalization [23]. This term penalizes model complexity by punishing deep trees and extreme leaf weights. The final objective value represents the total loss of all trees, along with a penalty term for each tree to prevent overfitting [22]. To minimize the regularized objective function one uses the formula:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{A.3}$$

The regularization term promotes simpler trees and thereby reducing overfitting by implementing the follow penalty term:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{A.4}$$

As in formula (A.1), $\mathcal{T}$ represents the number of leaves in the tree, w is the vector of leaf scores, γ is a parameter which regulates the number of leaves, while λ penalizes large leaf weights.

Since it is computationally unfeasible to directly optimize the space of trees, XG-Boost uses the same additive training method as Gradient Boosting, where the trees are added sequentially to improve the errors of the previous tree[23]. At iteration k, a new function f_k is greedily added, that minimizes the penalized objective the most:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)\right) + \Omega(f_k) \tag{A.5}$$

To efficiently approximate the loss function, a second-order Taylor approximation is used:

$$\mathcal{L}^{(k)} \approx \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(k-1)}\right) + g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i) \right] + \Omega(f_k) \tag{A.6}$$

Here, formula (A.7) represents the gradient (first derivative)

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(k-1)}} l\left(y_i, \hat{y}_i^{(k-1)}\right) \tag{A.7}$$

and the following formula, (A.8) is the Hessian (second derivative)

$$h_i = \frac{\partial^2}{\partial \hat{y}_i^{(k-1)\,2}} l\left(y_i, \hat{y}_i^{(k-1)}\right) \tag{A.8}$$

The goal here is to optimize f_k. Since the constant term, $l\left(y_i, \hat{y}_i^{(k-1)}\right)$ does not depend on f_k, it can simply be ignored. This simplifies the objective to:

$$\tilde{\mathcal{L}}^{(k)} = \sum_{i=1}^{n} \left[g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i) \right] + \Omega(f_k) \tag{A.9}$$

After simplifying the objective function, the various instances can be summed based on the leaf node they fall into. By grouping them together, the optimization becomes more computationally efficient [23]. The set $I_j = \{i | q(x_i) = j\}$ gathers every sample that results in leaf j, and $q(x_i)$ represents the function which assigns each input to its respective leaf. This grouping results in:

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \tag{A.10}$$

To find the optimal weight w_j^* for every leaf j can be calculated with the following formula:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{A.11}$$

Once the optimal value of w_j^* for each leaf has been found by using formula (A.11), they can be plugged into the objective function. This makes the value of the objective function only dependent on:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{A.12}$$

Formula (A.12) gives the tree a score which represents the quality of the tree structure, in other words it evaluates the tree splits. To help the algorithm decide whether a node should be split, and where the split should be done, the following formula is used:

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{A.13}$$

Here, I_L and I_R are the instance sets assigned to the left and right children after the split, and $I = I_L \cup I_R$ represents the parent node [23].

B Gradient Descent

Gradient Descent is an optimization method that iteratively improves a model's parameters by minimizing the loss function. At each step, the algorithm improves the parameters in the opposite direction of the loss function's gradient [30]. In the full-batch variant of gradient descent, the algorithm computes the gradients for the complete dataset before updating the model parameters. It is defined as [30]:

$$\theta_{i+1} = \theta_i - \eta \nabla_{\theta_i} J \tag{B.1}$$

where $\eta > 0$ represents the learning rate, and $\nabla_{\theta_{i-1}} J$ is the gradient with respect to the model parameters. By moving in the opposite direction of the gradient, or in other words, moving in the steepest descent, the algorithm decreases the loss function value [24]. Calculating the gradients over the full dataset is computationally expensive, and results in a model which will scale poorly. To tackle the scalability issue, SGD is commonly used.

C Stochastic Gradient Descent

As previously mentioned in section 3.5.3, SGD is a widely used optimization algorithm which is used to train ML models, particularly FNNs. It is a variant of the gradient descent algorithm, which introduces stochastic elements by estimating the gradient using a subset of the dataset at each step [30].

In supervised ML, a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ contains pairs of input data and respective label, where x_i is a specific input, and y_i is the corresponding label [24]. The overall loss function $J(\theta)$, which represents the average loss over the complete dataset, can be expressed as:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J_i(\theta) \tag{C.1}$$

where $J(\theta)$ is the loss for iteration i.

In standard gradient descent, the gradient of the loss function over the complete dataset is computed. This is computationally expensive for large datasets. To make the algorithm more efficient, SGD utilizes mini-batches, which can be described as small randomly selected subsets of the complete dataset, to estimate the gradient [30]. Once the estimate of a mini-batch has been computed, the parameters are updated. Therefore, the convergence of SGD is typically faster than that of gradient descent. The update equation for a mini-batch with m-number of samples is:

$$\theta_{i+1} = \theta_i - \eta \cdot \frac{1}{m} \sum_{j=1}^{m} \nabla_\theta J_{b_j}(\theta) \tag{C.2}$$

where $\{b_1, b_2, \ldots, b_m\}$ refers to a mini-batch sampled from D, and η is the learning rate.

SGD adds some noise into the parameter updates, which might help the model avoid getting stuck in local minima and saddle points [30]. It also makes computation time more sensitive to the chosen learning rate.

D Adaptive Moment Estimation

Adam is an optimization algorithm widely used for FNNs. It's a highly popular algorithm due to its robustness and efficiency. It combines the advantages of two gradient-based optimization methods [44]: Root Mean Square Propagation (RMSP) and Momentum. Adam keeps track of the average of past gradients and the average of their squared values, gradually the importance of past values are decreasing [45]. This gives the algorithm a way of adjusting the learning rate for each parameter automatically.

At each iteration, the first and second moment are computed using the following equations [44]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta_t) \tag{D.1}$$

where m_t is computing the first moment, and where:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2 \tag{D.2}$$

v_t is calculating the second moment. β_1 and β_2 are hyperparameters that controls at what rate the importance of past values are decreasing for the two moments. Typically m_t and v_t are biased towards zero in the initial steps of the algorithm as the hyperparameters are typically set to a value very close to 1. Therefore, Adam uses bias-corrected estimates [45]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{D.3}$$

The parameter update rule is defined as:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{D.4}$$

where η represents the learning rate, ϵ is a small constant added to prevent division by zero [45]. Adam is strong when it comes to problems which involves noisy and sparse gradients, and is known to have a quick convergence while needing minimal hyperparameter tuning [44].

E Dropout

Dropout is a widely used regularization technique in FNNs that helps reduce overfitting and improve generalization. The method works by randomly deactivating a subset of neurons during each training iteration, which prevents the model from depending too heavily on specific nodes [46]. Instead, the model is forced to learn more robust and redundant

This technique can be interpreted as training an ensemble of smaller sub-networks within the same overall architecture. At each training iteration, a different subset of neurons is deactivated, leading to small variations in the network structure [46]. When making predictions on new data, all neurons are active, and the outputs are appropriately scaled to reflect the expected values from training.

Dropout is commonly applied in the fully connected layers of FNNs and has been shown to significantly reduce overfitting, especially in models with a large number of parameters [46].