TECHNISCHE UNIVERSITÄT CHEMNITZ

Prediction of Tool Recipe Runtimes in Semiconductor Manufacturing

Master Thesis

Submitted in Fulfilment of the

Requirements for the Academic Degree

M.Sc. Automotive Software Engineering

Dept. of Computer Science

Chair of Computer Engineering

Submitted by: Karim Sadek

Student ID: 580632

Date: 04.10.2021

Supervising tutors: Prof. Dr. Dr. h. c. Wolfram Hardt

Dr. Holger Brandl

Reda Harradi

Abstract

To improve throughput, due date adherence, or tool usage in semiconductor manufacturing, it is crucial to model the duration of individual processes such as coating, diffusion, or etching. Equipped with such data, production planning can develop dispatch schemes and schedules for optimized material routing. However, just a few tools indicate how long a process will take. Many variables affect the runtime of *tool recipes* that are used to realize processes. These variables include wafer processing mode, historical context, batch size, and job handling. In this thesis, a model that allows inferring tool recipe runtimes with adequate accuracy shall be developed.

Firstly, predictive models shall be built for selected tools with known runtime behavior to establish a baseline for the methodology. Tools will be selected to cover a broad spectrum of processing modalities. The main predictors will be revealed using variable importance analysis. Furthermore, the analysis shall reveal under which conditions recipe runtime modeling is most accurate.

Secondly, a generic approach shall be created to model recipe runtime. By accounting for tool, process, and material context, methods would be investigated from feature selection and automatic model selection. Finally, a pipeline for data cleansing, feature engineering, model building, and metrics will be developed using historical data from a wide range of factory data sources.

Finally, a scheme to operationalize the findings shall be outlined. In particular, this requires establishing model serving to enable consumption in applications such as dispatching or operator interfaces.

Keywords: Machine Learning, Data-driven models, Regression

Content

List of Figures

List of Tables

List of Abbreviations

AUTOML	Automated Machine Learning	**AMHS**	Automated Material Handling System
ANN	Artificial Neural Network	**API**	Application Programming Interface
DRF	Distributed Random Forests	**EDA**	Exploratory Data Analysis
ERP	Enterprise Resource Planning	**GBM**	Gradient Boosting Machine
KNN	K-nearest neighbor	**ML**	Machine Learning
MES	Manufacturing Execution System	**REST**	Representational state transfer
RMSE	Root-mean-square error		

1 Introduction

Semiconductor manufacturing is composed of a set of Process Steps. A Process Recipe performs a set of Process Steps to manifest an end product. Prediction of Recipe Runtimes in Semiconductor Manufacturing is a complex task due to the various factors that could be affecting the manufacturing process of semiconductors, which is a high technology complex process. In addition, it is a competitive industry, utilizing costs, quality, and manufacturing time are crucial. Traditionally, factory in-house specialists rely on custom-built methods such as custom calculations, rule of thumb, statistical methods, or simulation to predict recipe runtimes. However, such approaches are static and do not take into consideration the changing behavior and patterns. In this thesis, a different approach to predict tool recipe runtimes is studied, the possibility of predicting future recipe runtimes based on historical data by applying machine learning algorithms.

Prediction of manufacturing times has been recently an attractive area of research due to the value it could bring if the accuracy is adequate, which will ultimately utilize costs and manufacturing times such as Recipe Runtime, Lead Time, or Cycle time. First, prediction of such times is currently possible due to the data gathered by MESs at customer's plants. Secondly, due to the high computation power available now.

Prediction of recipe runtimes requires first understanding the processes involved when producing Semiconductors. Following that, studying the properties of recipe runtimes was essential to understand the production workflow and the processes timeline. In this chapter, the steps involved in this initial understanding are discussed and the use cases for such prediction and the objectives for this thesis.

1.1 Semiconductor Manufacturing

Manufacturing is *"The process through which raw materials are converted into a finished product"* [1]. In such a process, the raw materials are inputs, and the finished products are the output. The input could be semiconductor materials, insulators, dopants, or metals in the semiconductor manufacturing domain. The output ranges from integrated circuits to commercial electronic products such as personal computers, mobile phones, and servers.

Semiconductors are the foundation of every integrated circuit. The manufacturing process is composed of two stages. Firstly, wafer processing, which is the stage of the formation of the silicon wafer, illustrated in Figure 1.1, where the electronic circuits are produced on the surface of a silicon wafer. Followed, the front end of line processing, which is the operation of wafer processing, and the back end of line processing, which is the assembly process, as shown in Figure 1.2, a set of steps to complete the semiconductor assembly.

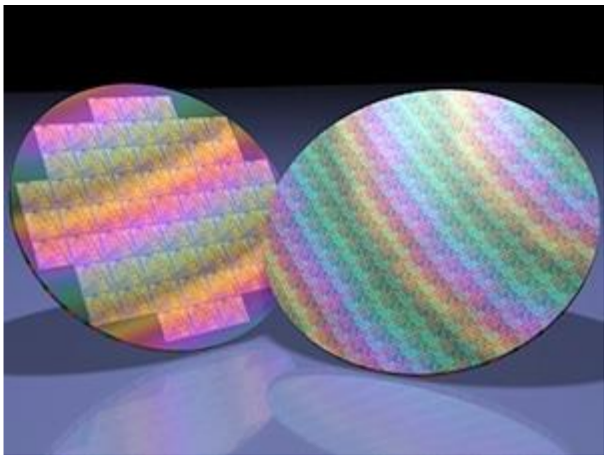


Figure 1.1: Silicon Wafer

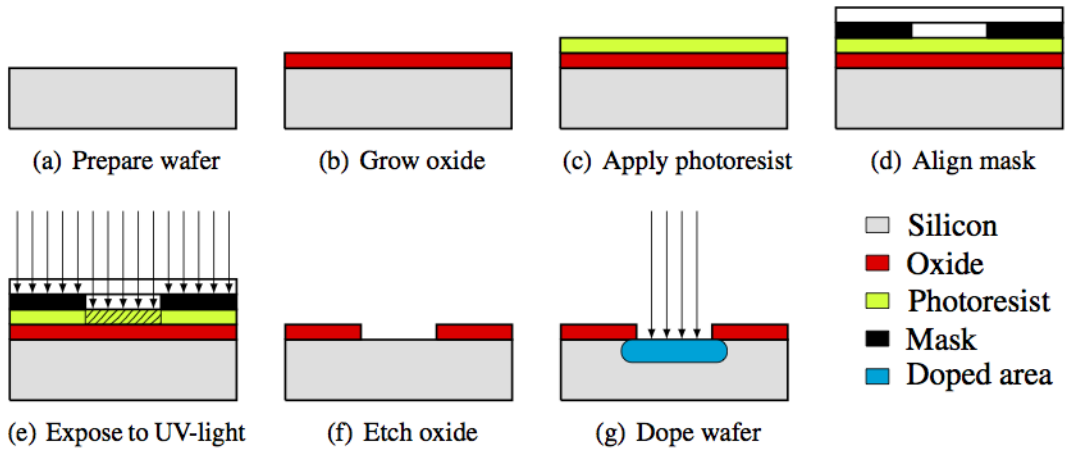


Figure 1.2: Basics of Front End of Line Processing

Understanding Semiconductor manufacturing processes such as coating, diffusion, and etching is the first step towards predicting Recipe Runtimes. Semiconductor manufacturing is a complex process that is composed of several phases. Each phase could consist of more than one hundred steps. [2] Each step is one of four phases such as the following:

- *Layering*: where the wafer surface is covered by a thin layer of conducting, semiconducting, or insulating material
- *Pattering*: where some parts are of the layers inserted in the layering phase are removed
- *Doping:* where specific amounts of dopants are added to the water surface
- *Heat treatments:* where the wafers are heated or cooled according to goals.

The mentioned phases could be repeated multiple times to reach the end product.

1.2 Wafer Processing and Lot Size

One of the most significant and predictable influences on the recipe runtime is how many wafers are processed.

In principle, there are three different cases:

- Complete Batch Processing

- Single Wafer Processing
- Interleaving Wafer Processing

1.2.1 Complete Batch Processing

An example of that is a furnace where all the wafers are put together into a tube. In such a case, the batch is usually filled up to a certain extent, even with dummy wafers. Therefore, the size of the Lot should not be of any significance for the processing but could be for the loading and unloading.

Complete batch processing is the mode of processing where all wafers are put together in a tube. The batch is usually filled up to a certain extent, including dummy wafers if needed. Therefore, the size of the Lot should not be of significance for the processing but could be for the loading and unloading processes.

1.2.2 Single Wafer Processing

Metrology equipment where wafers are measured one after the other is an example of single wafer processing. In this case, the runtime comprises of the following components:

- Loading including the scanning of the slot map
- Loading into, processing, and unloading from the metrology chamber
- Unloading

In some cases, it can take longer for the 1st wafer if it has to go to an aligner or laser mark reader before it is loaded into the metrology chamber. Subsequent wafers might already be aligned or read while the preceding wafer is measured.

1.2.3 Interleaving Wafer Processing

The standard example for this case is multi-chamber equipment. In this case, wafers from one cassette are processed simultaneously in different chambers.

Interleaving Wafer Processing is performed at multi-chamber equipment. In this case, wafers from one cassette are processed simultaneously in different chambers. Each chamber performs a custom function and has its customized configuration, and wafers are transferred from one chamber to another. Figure 1.3 shows multi-chamber equipment with individual chamber configurations. The flow of a wafer visiting the chambers in sequences A, B, C, and D is also shown.

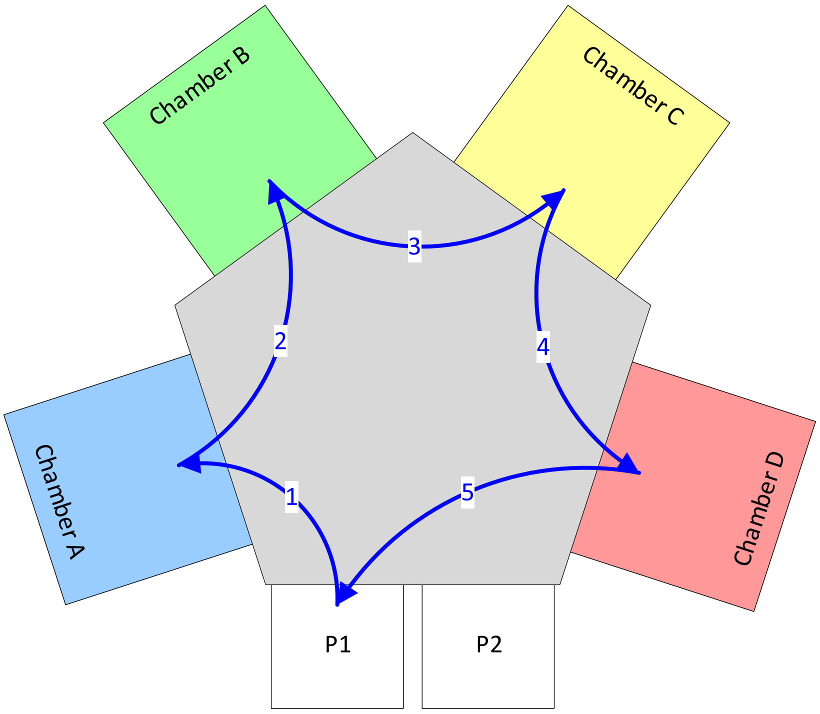


Figure 1.3: Individual chamber functions

Depending on the processing time for each chamber, the entire processing time for each wafer will vary. Figure 1.4 shows an example.

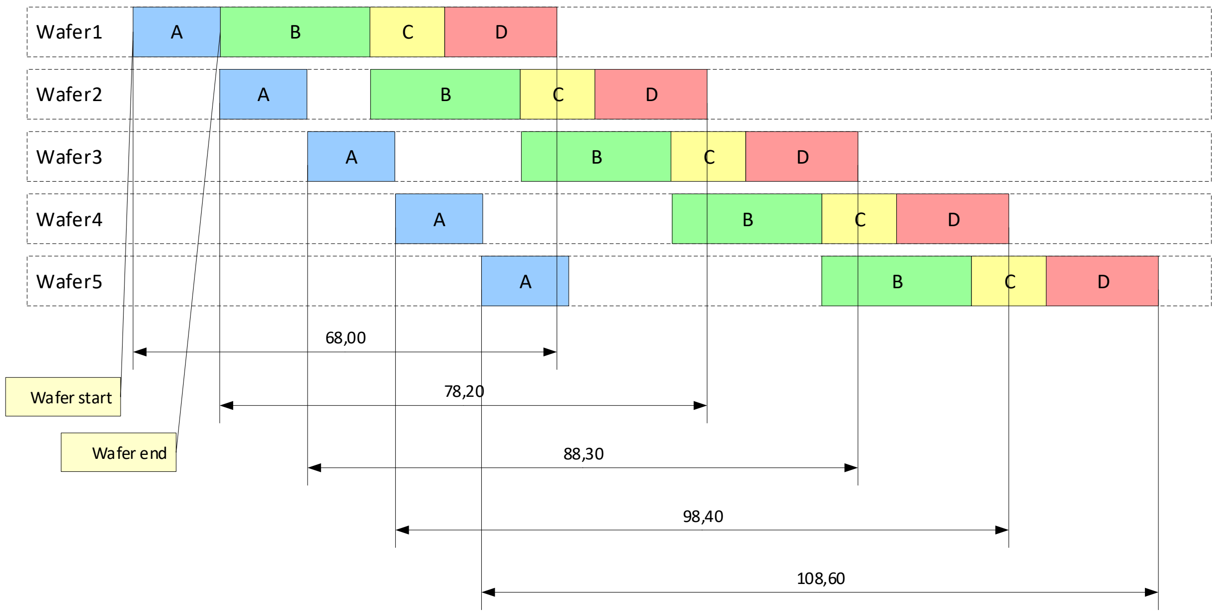


Figure 1.4: Individual chamber functions, timeline

The following Figure 1.5 shows a configuration where more than one chamber can be used for the same process. In this case, the sequence is A → (B or C) → D. The two possible flows of a wafer 1.x and 2.x are shown too.

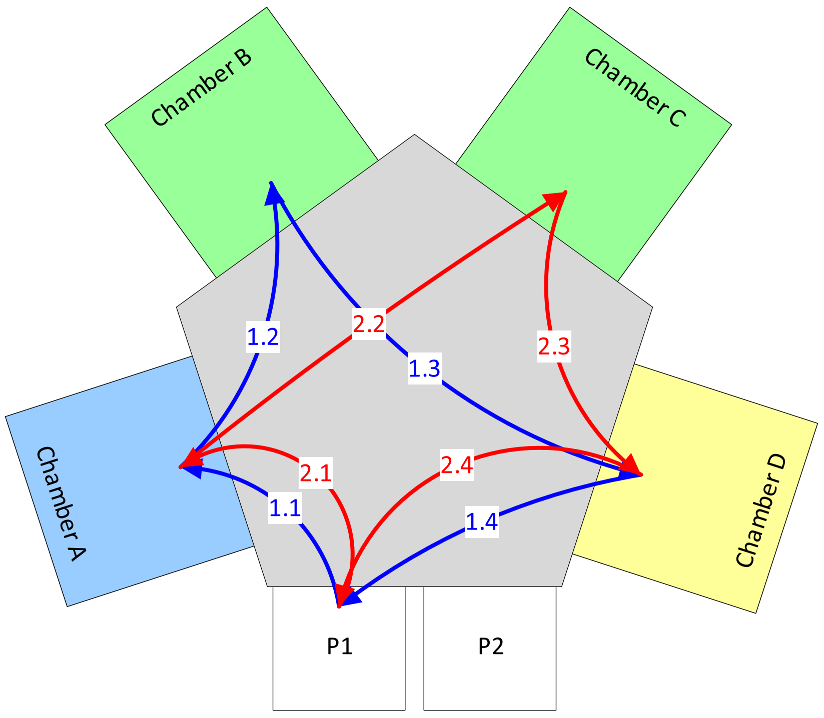


Figure 1.5: Chambers with same functions

Figure 1.6 shows a timeline example assuming that the two chambers B and C having the same functions have the longest processing time. In the example, it is more than two times longer than all other processes.

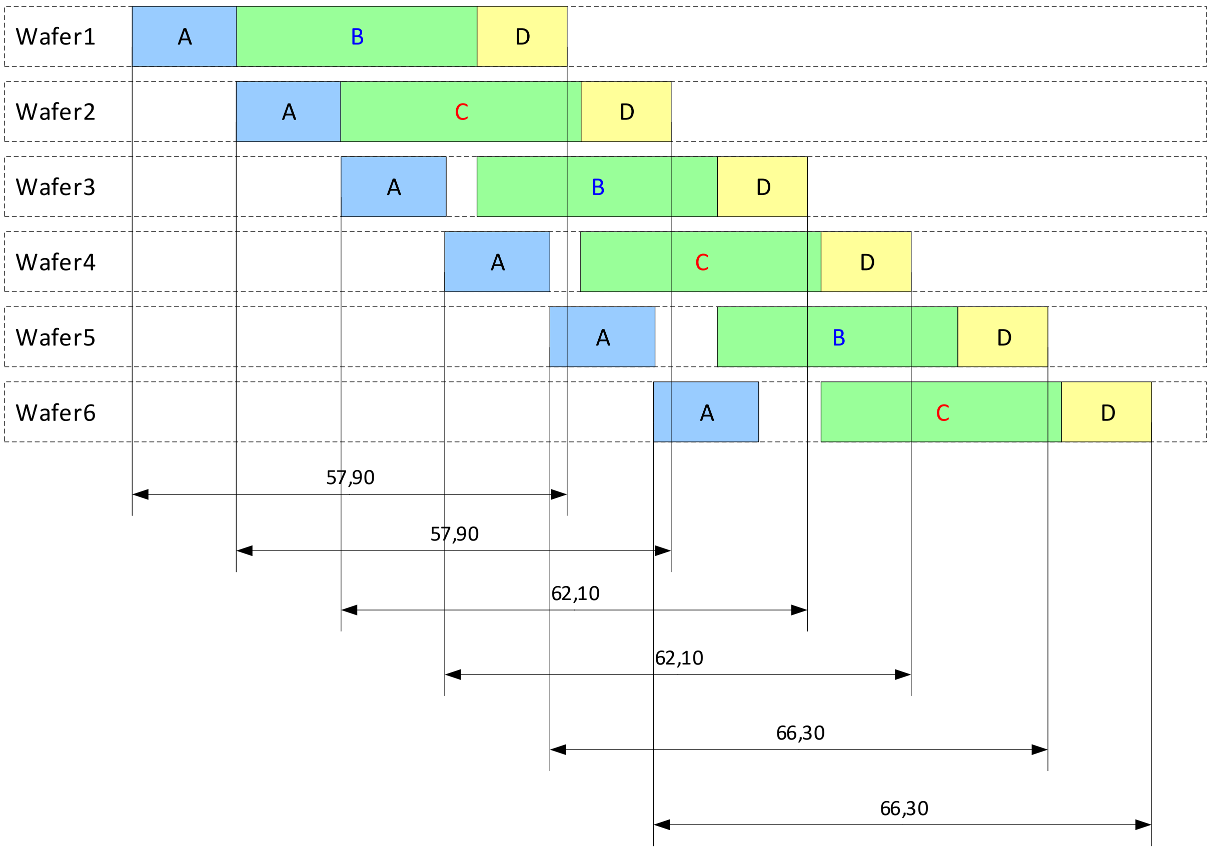


Figure 1.6: Chambers with identical functions, timeline

The configuration shown in Figure 1.5 adds another complexity to the forecasting of runtimes. For example, if chamber B or C is down, the equipment could still run the process, but it will be slower given the assumption that chambers B and C are the longest.

1.3 Process Job Handling

Process jobs can essentially be executed in two different ways:

- Serialized: a subsequent process job only starts after the preceding one was finished

- Interleaving: a subsequent process job starts while preceding one(s) are still running

Process Job Handling depends on the equipment type built, and on the other hand, the equipment internal job scheduler.

1.3.1 Serial Jobs

Examples for serial jobs are

- Furnaces, where multiple lots are loaded all together into a tube
- Metrology equipment with one metrology chamber

Figure 1.7 below shows two jobs executed serially. It also shows some events and information from the systems, the equipment, and the AMHS (Automated Material Handling System) associated with the different points in time.

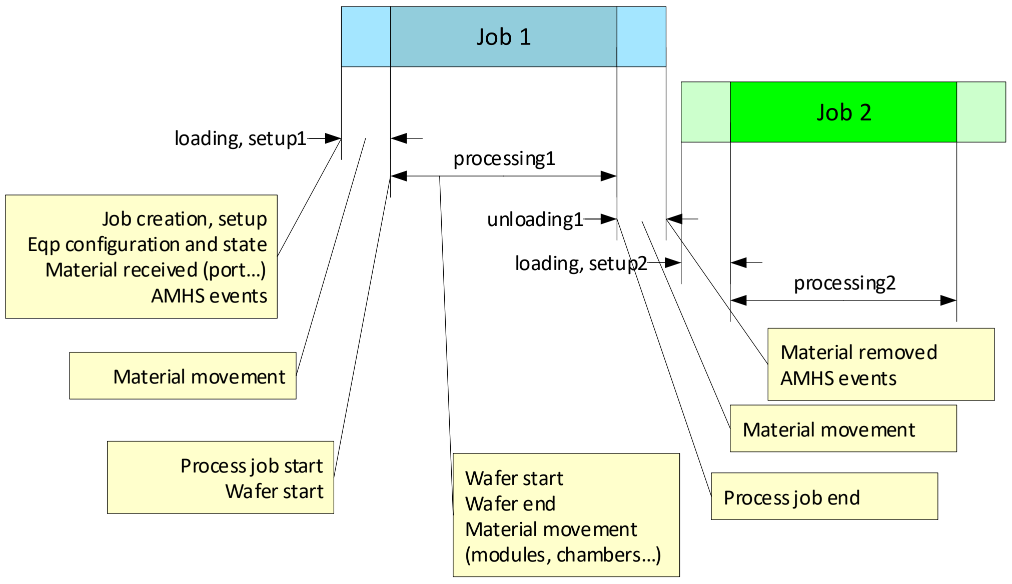


Figure 1.7: Serial Jobs

1.3.2 Interleaving Jobs

Examples for interleaving jobs are

- Wet benches, where the batches are processed through a series of different tanks

- Multi-chamber equipment, as shown in Figure 1.8, is capable of running wafers from a 2nd (or nth) job in chambers not used by the preceding job

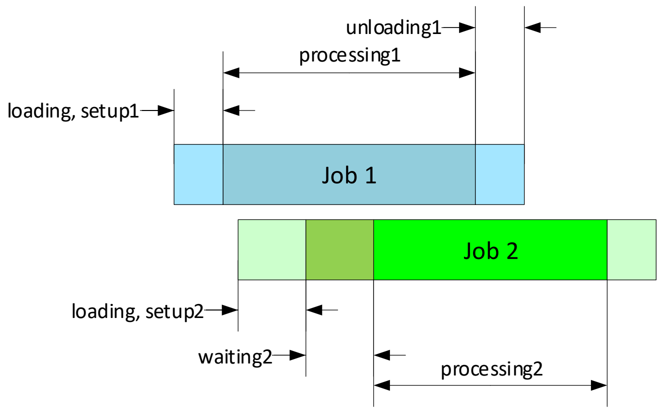


Figure 1.8: Interleaving Jobs

Predicting interleaving job runtimes is usually more complicated than for serialized jobs.

Figure 1.9 shows multi-chamber equipment with four chambers and two load ports. On load port 1, job one was started. Wafers 1 to 4 already have been processed; 5 to 8 are currently being processed. In this example, the Recipe for that job sends all wafers through all chambers (A → B → C → D). On load port 2, job 2 with the same chamber flow is prepared and waiting.

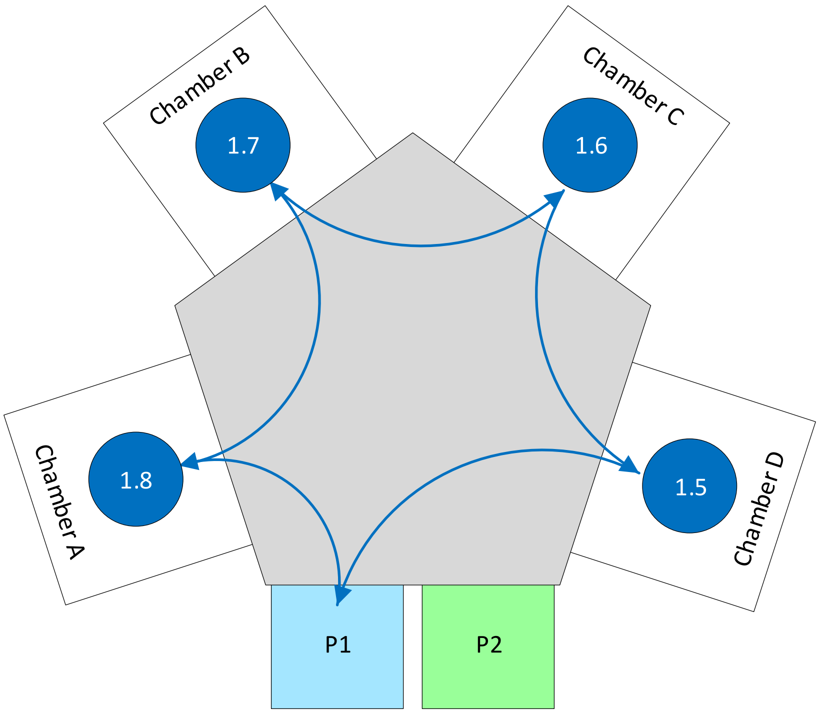


Figure 1.9: Multi-chamber equipment, 1st job

While Figure 1.10 shows the situation where the last (25th) wafer of job 1 is still in chamber C while job 2 was already started.

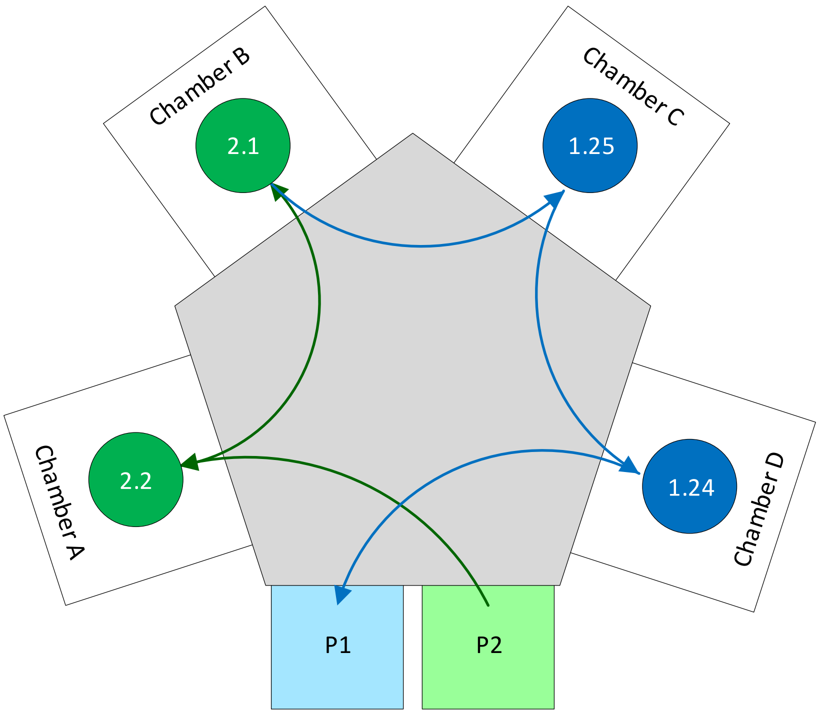


Figure 1.10: Multi-chamber equipment, interleaving jobs, 2nd job started

The situation is getting different when job 1 and job 2 are using different chamber sequences. The equipment-internal scheduler that controls the parallel usage of chambers will be considered the overlaps of used chambers if it is a capable scheduler. The effect could be that the overlap of two process jobs could be minimal, or even zero, if the same chambers are used, or very high if there is no or minimal overlap.
Figure 1.11 shows the situation that job 1 only uses chambers A and B, job 2 only chambers C and D. In this case, there is no overlap.

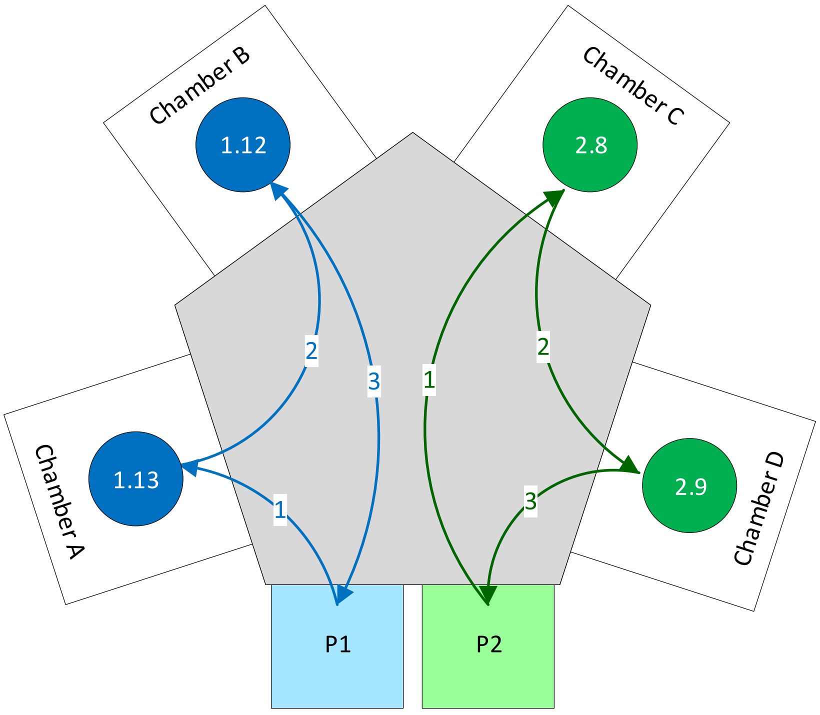


Figure 1.11: Multi-chamber equipment, parallel jobs

The situation gets even more complicated when for example, the sequence for job 1 is A → B → C, while the sequence for job 2 is still C → D. In this case, the scheduler has to decide whether it can start both jobs and let the jobs share chamber C.
Even in the more straightforward example of a wet bench where the whole Lot is moved from one tank to the next, the preceding job can influence by blocking a tank. Nevertheless, also here, it depends on the process and the equipment scheduler. For example, the process will not be started if job 1 blocks tank B since job 2 has to be moved immediately from tank A to B and is not allowed to wait while still in contact with the chemicals of tank A.

1.4 Recipe Runtime

Recipe Runtime is the time taken for a particular recipe to be completed. It represents the difference between the job completion time and job start time. There exist different recipes for different purposes. According to the Recipe, each phase consists of multiple sub-steps operated by different machinery in a specific order according to the desired output device. The mentioned operations are applied to lots consisting of up to 25 wafers in one batch to optimize costs and reduce the manufacturing time [2].

Recipe Runtimes are influenced directly by the type of wafer processing and the number of wafers in a lot being processed. The time taken for each recipe is recorded, and historical data are available at manufacturers.

It is evident that semiconductor manufacturing is composed of a complex set of processes and predicting the recipe runtimes requires an understanding of the processes involved. In addition, it is vital to examine job parameters to decide on the predictors. Therefore, there could be various job parameters.

1.5 Use cases for Recipe Runtime

A recipe runtime prediction system could have different use cases once developed, such as the following:

i. Operator Guidance:

Once the duration per second for a specific recipe is predicted with adequate accuracy, the operator shall leverage such information. Such information could help the operator plan his or her time and be alerted when needed.

ii. Planning:

A use case that serves the manufacturing process further is Planning. Predicting the time needed for each Recipe to be completed could help factories plan their production overhead and thus utilize their resources fully and ultimately achieve the maximum throughput. With the correct planning, optimized processes could be achieved, reducing the work in progress and ultimately reducing costs.

iii. Engineering Perspective:

A third use case that serves the engineering perspective in a factory is to understand the reasons for having variability in tool recipe runtimes in the first place.

1.6 Thesis Objective

In this thesis, ML models shall be built to predict Tool Recipe Runtimes in Semiconductor manufacturing. Having such models could contribute to building a system that would ultimately predict future tool recipe runtimes for a given recipe in semiconductor manufacturing with minimum input and adequate prediction accuracy. Thus, a user would ultimately choose a recipe name as an input and obtain the predicted runtime as an output to serve the mentioned potential use cases for such technology.

The thesis studies the possibility and adequacy of different ML models such as baseline models, specific models, and Automated Machine Learning. The errors obtained from each model are analyzed in-depth, and the possibility of operationalizing such models is studied.

2 State of the Art

This thesis studies the applicability, effectiveness, and possibility of predicting recipe runtimes by leveraging historical data by building machine learning, predictive models. Prediction of tool recipe runtimes in semiconductor manufacturing is mainly predicted by in-house tools, simulation, or rule of thumb. [3] These tools are developed internally to calculate an estimate of recipe runtimes. However, this approach could be limiting as it does not consider the historical behavior in production according to the data generated by manufacturing execution systems.

2.1 Literature Review

There has been ongoing research and development in predicting the cycle time of semiconductor manufacturing. [3] However, the literature review did not come across research in predicting specific recipes based on historical data and its features. Since machine learning could theoretically provide predictions in any domain or application shall this data be available, we decided to apply it to semiconductors recipe runtime data available to us, study the results obtained, and analyze the approach's effectiveness.

A big part of the prediction is identifying patterns. A similar anology is when humans learn a new hobby or a new language. We being to look for patterns. Applying statistics on data is a traditional approach to derive conclusions for specific data. Statistical models could help us in identifying patterns in data. However, statics methods could only provide us with insights such as mean, median and to determine the type of distribution, if there is any. However, it is crucial to derive a model to predict recipe runtimes.

Similar research has been developed in predicting the Lead Time (LT) in semiconductor manufacturing through predictive data analytics, which uses historical data to build models able to predict according to historical data [4]. Cross Industry-standard Proces for Data Mining sets 6 phases for the lifecycle of predictive data analysis projects. The phases are as the following:

i. Business understanding
ii. Data Understanding
iii. Data preparation
iv. Modeling
v. Evaluation
vi. Deployment

The modeling phase is where such predictive models are built by applying machine learning algorithms. Researchers applied different ML algorithms on semiconductor manufacturing data to predict lead time. The top-performing model was selected for evaluation and later for deployment.

Machine learning automates the process of extracting a pattern from historical data. There are two main types of ML, Supervised learning, where the training dataset is labeled, and Unsupervised learning, where the dataset is not labeled. There exist other types of ML, such as semi-supervised learning and reinforcement learning. In predicting the lead time, the data was labeled, and so it was supervised learning.

As the prediction process of lead time in manufacturing is ongoing, it is a regression problem.

The first mention of statistical learning, knowledge discovery, or data mining as a possible practical approach to extract insights and derive conclusions from various datasets, was in 1989 [5]. The research was focused on the industries of medicine, biotechnology, finance, and marketing. However, there was less interest in the manufacturing domain [6]. Recently, there has been a growing research interest and more publications concerning leveraging production management data through analytical methods and techniques and recorded investment returns.

There are possible outcomes for data mining, descriptive statistical analytics, and predictive data mining. The first is used in finding patterns in data to provide a better understanding. Like classification or regression, the latter analyzes actual data to predict future data points for specific variables. Several applications rely on this principle, such as scheduling, defect analysis, and fault diagnosis. One more potential application could be predicting cycle time, waiting time, recipe runtimes in semiconductor manufacturing.

The literature survey reveals research in predicting lead time, cycle time, waiting time by applying different ML algorithms. However, no prediction of Recipe specific runtime research has been found in the review. Results show that applying ML methods to predict different manufacturing times showed the effectiveness and higher accuracy than traditional prediction methods. The traditional methods are tailored for table production environments to make a data-driven tool recipe runtime model more dynamic [4], [7].

2.2 Machine Learning

Machine learning is the science of teaching machines through historical data to eventually predict further data points in the future. Machine learning can be defined as "*Computational methods using experience to improve performance or to make accurate predictions."* [8].

Machine learning utilizes the sciences of statistics, mathematics, and computing to solve a problem, gather insights, or bring business value. The significant rise in popularity of machine learning is owed to the increase in computation power, which is currently sufficient to perform high computation tasks to achieve the required results in an adequate time.

The issue of how to create computer programs that automatically develop through experience is answered by machine learning. It is one of today's fastest-growing technological areas at the intersection of computer science and mathematics and the heart of artificial intelligence and data science. The emergence of new learning algorithms and hypotheses and the continuing boom in data availability and high computing have driven recent progress in machine learning.

Machine learning works through learning correlations and dependencies from historical data. There are two main modes for learning, either supervised learning or unsupervised learning. Supervised learning refers to classification, While unsupervised learning refers to clustering. Through the machine, learning programs could improve their outcome through an ongoing learning experience. There are different use cases for machine learning, such as identifying trends, abnormalities, prediction, planning, and diagnosing. With the current state-of-the-art in machine learning, prediction of any future data point is possible. Thus, it is possible to build predictive models that could learn from historical data. In recipe runtime prediction, such a model will be supervised, as shown below in Figure 2.1 [9]–[14].

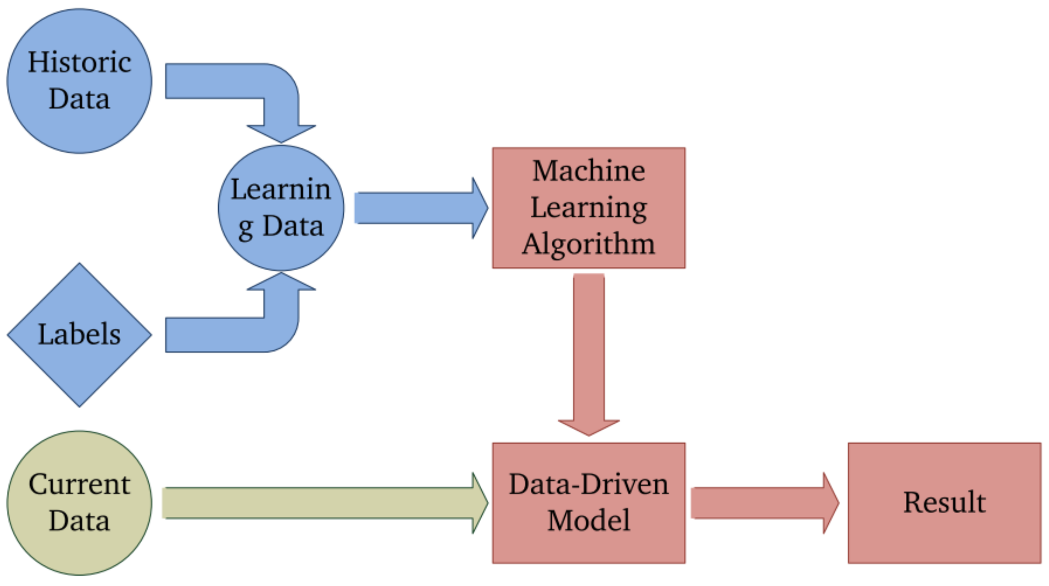


Figure 2.1: Predictive Model

Several machine learning algorithms could be applied to build a desired predictive models. In this subchapter, the models used in this thesis are reviewed.

2.2.1 Random Forests

Random forests regression is a supervised machine learning algorithm that utilizes the ensemble learning function for regression. Ensemble learning function combines predictions provided by several machine learning algorithms to produce higher accuracy prediction than a sole model. Random forests algorithm is based on decision trees where each decision tree predicts its best score, as shown below in Figure 2.2. Afterward, the predictions are averaged, and the random forests algorithm provides its predicted value.

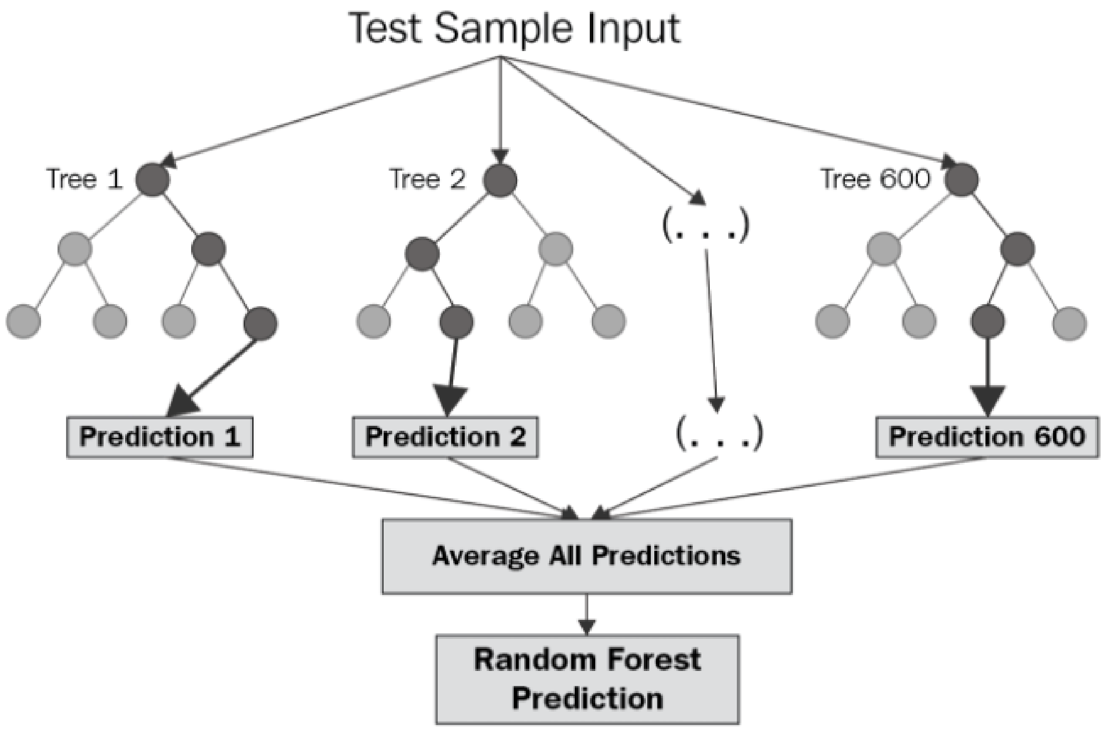


Figure 2.2: Random Forests prediction

Random forests are a collection of tree predictors in which the values of a random vector are collected independently and with the same distribution for all trees in the forest are used to predict the activity of each tree. Following the generation of a large number of trees, the top-performing score is voted. This procedure is defined as the generation of Random Forests [15].

Random forests regression provides conformal prediction. It utilizes all given data as a training set. Compared to other renowned algorithms such as the KNN and ANN on both standardized and normalized setup, the generated prediction results provided by Random Forests are of higher accuracy on the majority of confidence levels compared to the mentioned alternative algorithms. As the number of trees in a forest grows, the generalization error converges to a limit. The strength of individual trees in the forest and their correlation determine the generalization error of a forest of tree classifiers. When a random selection of features is used to split each node, the error rates are comparable to Adaboost, but they are more noise-resistant in comparison. Random Forests can be used for either a categorical response variable, classification, or a continuous response, regression [15], [16].

The random forests regressor firstly picks a random number of data points from a training set. Following that, it builds decision trees associated with the specified data points. Afterward, as specified, several trees are built, and thus the preceding steps are repeated accordingly. Finally, the new data points are predicted through each tree, and then the average of all values is calculated.

Distributed Random Forest is a sophisticated classification and regression algorithm. When given a data set, DRF constructs a forest of classification or regression trees rather than a single classification or regression tree. Each of these trees is based on a subset of rows and columns and is a weak learner. The variance will be reduced when more trees are planted. Whether predicting for a class or a numeric value, both classification and regression use the average forecast across all of their trees to generate a final prediction [17].

2.2.2 Gradient Boosting

Building a non-parametric regression or classification model from data is a common task in several machine learning applications. One technique for creating a model in domain-specific domains is to start with theory and then alter the parameters based on the evidence. Unfortunately, such models are not available in most real-life circumstances. In most cases, researchers do not reference even initial expert-driven assumptions about the possible correlations between input variables. The lack of a model can be overcome by using non-parametric machine learning techniques such as neural networks, support vector machines, or any other algorithm to create a model straight from the data [18].

Building a single solid predictive model is the most common method of data-driven modeling. However, building a bucket, or an ensemble of models, for a learning objective is a different technique. Instead, consider creating a collection of robust models, like neural networks, that may be integrated to produce a superior prediction.

In practice, however, the ensemble method involves merging a large number of relatively simple models to produce a more robust ensemble forecast. Random forests and neural network ensembles are two of the most well-known machine-learning ensemble approaches [19], [20].

Random forests and other joint ensemble approaches rely on simple averaging of models in the ensemble. The boosting approaches are based on a distinct, constructive ensemble-building strategy. The primary idea behind boosting is to add additional models to the ensemble incrementally. An additional base-learner model is trained concerning the error of the entire ensemble learned so far at each iteration.

2.2.3 Automated Machine Learning

An outbreak of machine learning research and applications has occurred in the last decade; in particular, deep learning techniques have enabled significant advances in many fields of application, such as computer vision, speech recognition, and gameplay. However, the performance of many machine learning approaches is very susceptible to many design decisions, which is a significant barrier for new users. It is especially true in the booming field of deep learning, where human engineers need to choose the correct neural architectures, training procedures, methods of regularization, and hyperparameters of all of these components to make their networks perform with satisfactory performance. For every application, this step has to be replicated. Also, scientists are often left with tedious trial and error runs before finding an adequate solution for a specific dataset [21].

Machine Learning often is an experimental incremental process without fixed limits for improvement. Different algorithms are encouraged to be applied to test hypotheses and results. However, not everyone is an ML expert and knows how to control every model, and as well there is no way to apply different algorithms in one training process. This is where Automated Machine Learning comes to fulfill the gap as it applies, tune, and test a library of different ML algorithms until the best possible score is achieved using efficient Bayesian optimization methods [21], [22].

Automated Machine Learning aims to make these decisions in a data-driven, objective, and automated manner: the user simply provides data, and the AutoML framework automatically decides the best performing approach for this specific application. AutoML thus makes state-of-the-art approaches to machine learning available to domain scientists interested in implementing machine learning but cannot learn in-depth about the technologies behind it. This leads to the democratization of Machine Learning, making the technology appliable to a larger pool of users [21].

The recent significant progress in ML has led to rising demand for hands-free ML systems that can help developers and ML novices build new ML applications effectively. Since different datasets require different ML pipelines, this demand has given rise to standard AutoML systems such as Auto- WEKA, hyperopt-sklearn, Auto-sklearn, TPOT, and Auto-Keras perform a combined optimization across different preprocessors, classifiers, hyperparameter settings, and others, thereby reducing the effort for users substantially [22].

AutoML systems have been effective in numerous applications, implementing their new hyper-hyperparameters, including the option of the evaluation strategy used in the loss function, the time estimate used, and the hyper-hyperparameter optimization strategy [23].

AutoML methods are classified as “lifelong machine learning” systems. ML systems learn several tasks from multiple domains, possibly in their lifetime. An ideal AutoML system would be a method to learn from all tasks and adapt itself during its lifetime. As a result, it would ideally be the most competitive and robust machine learning solution for all problem statements [24].

The protocol of how AutoML is performed is as shown in Figure 2.3. Firstly, the machine learning problem is identified. Then, the dataset to be used is made available, and the labeled data is identified. Then the target metrics are set, the constraints, if any, and the hyperparameters. Afterward, the training is of the different algorithms is executed. Finally, the results are observed [25].

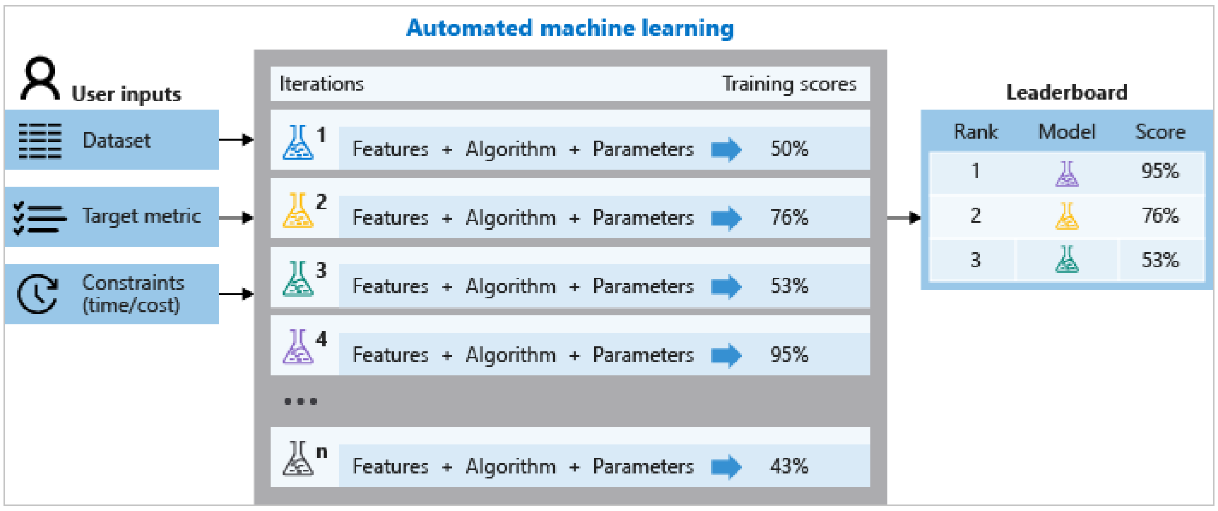

Automated machine learning
User inputs
Dataset
Target metric
Constraints (time/cost)
Iterations
Training scores
1 Features + Algorithm + Parameters 50%
2 Features + Algorithm + Parameters 76%
3 Features + Algorithm + Parameters 53%
4 Features + Algorithm + Parameters 95%
n Features + Algorithm + Parameters 43%
Leaderboard
Rank Model Score
1 95%
2 76%
3 53%

Figure 2.3: AutoML Workflow

Due to the numerous benefits, AutoML brings such as high accuracy results and time-saving, it was selected for development in several projects.

Auto-sklearn is a robust AutoML engine based on the renowned most commonly used Scikit-learn Python ML package. Scikit-learn is a widely used ML library used by both experts and beginners to implement ML algorithms on datasets in Python. Auto-sklearn is an AutoML methodology that uses meta-learning, Bayesian optimization, and ensemble selection to determine promising ML pipelines comprising preprocessing techniques and ML classifiers. Auto-sklearn is the “State of the Art of AutoML.” [24] Bayesian optimizations are used to establish a high-performance optimized ML pipeline on any data dump. Auto-sklearn uses 15 built-in classifiers, 14 preprocessing methods, and 4 data preprocessing methods, as illustrated in Figure 2.4 comparing the different outcomes and thus selecting the best result [21], [26].

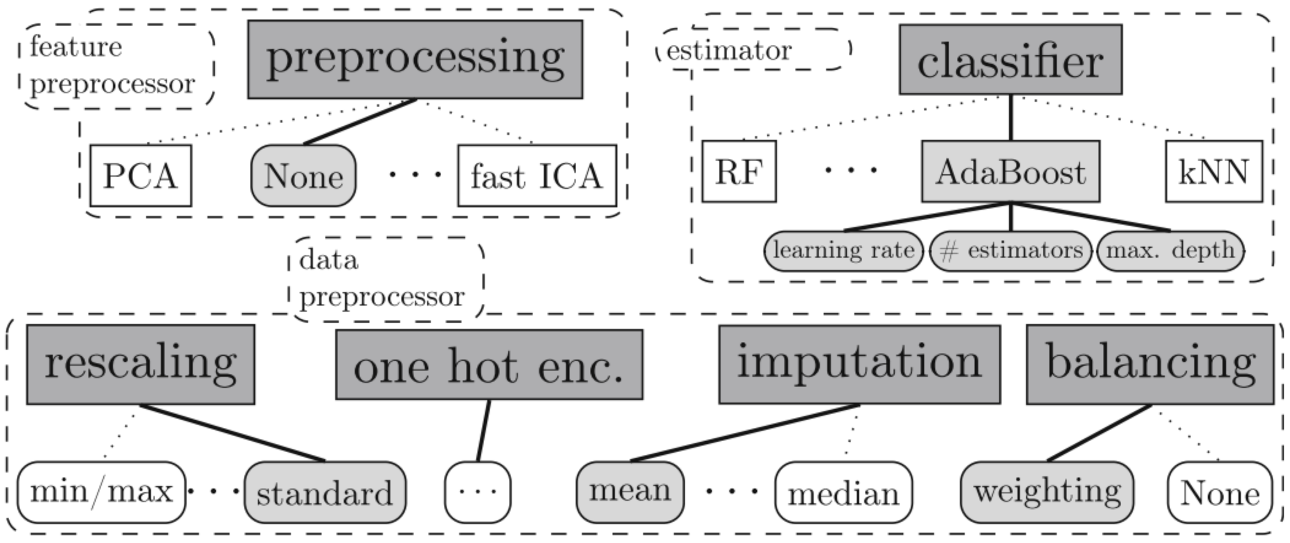


Figure 2.4: Autosklearn Structure

Two main elements make Auto-Sklearn effective. The first is built on meta-learning, complementary to Optimization techniques, which is used to accelerate the optimization by rapidly recommending instantiations of a framework that is forecasted to perform well. The meta-learning was accomplished from an off process where 38 meta-features were learned from 140 OpenML datasets. The second function is the automatic ensemble construction of models evaluated during optimization. Instead of discarding the remainder of the models found in the Bayesian optimization process, it stores them and then constructs and assembles them using a greedy ensemble selection algorithm to select the best model [24].

In the final step of Autosklearn's workflow, the best pipelines identified during the Bayesian search process are used to construct an ensemble. This automated ensemble construction avoids committing itself to a single hyper-parameter setting, and it is more robust than only using the best pipeline found with the optimization component.

Another provider of user-friendly AutoML software that non-experts could use is H2O AutoML. H2O simplifies automated machine learning by providing a unified interface to several machine learning algorithms. Although H2O AutoML simplifies AutoML training, it requires machine learning knowledge to produce and understand high-performing ML models. AutoML automates the process of training a variety of potential models. In addition, it automates data preprocessing, feature engineer and provides helpful insights that explain the performance of the selected models [27].

H2O AutoML can automate the ML workflow by automating training and tuning several models within a specified time limit. Stacked ensembles are automatically trained through the combination of several models to build the highest predictive ensemble models. Commonly, such models are the top-performing models based on the AutoML score leaderboard [27].

2.3 Chapter Summary

A significant part of the prediction is identifying patterns. Machine learning automates the process of identifying patterns in historical data to predict future data points. This process could be applied to different applications and domains, such as the semiconductor manufacturing domain. Literature survey reveals past research in applying ML algorithms to predict cycle time, waiting time, and lead time. Since predicting recipe runtimes is a regression problem, different algorithms such as Random Forests, Gradient Boosting, and AutoML have been studied to be selected and used to serve such applications.

3 Data Preparation

To predict tool recipe runtimes in semiconductor manufacturing by building ML models and predicting future points, the data must first be available and then prepared. For the data to be ready, there are a set of procedures that has to be applied. This chapter reviews the following data preparation phases: data gathering, parameter selection, exploratory data analysis (EDA), and data engineering.

3.1 Data Gathering

The requirements for advanced manufacturing are many, including optimized resource planning, quality management, utilizing shop floor tools, production and personnel timeline utilization, and the ability to stay dynamic and respond to changes.

For over three decades already, corporations worldwide have been allocating an investment budget in information systems to increase productivity and realize the added return on investment values. Enterprise Resource Planning Systems serve as the backbone of the operation of corporations, and even for small and medium enterprises to manage resources, orchestrate scheduling, and manage the supply chain. When it comes to manufacturing, a solution has been needed for the shop floor as ERP systems do not offer the level of detail needed to add the suitable materials at the correct time with the matching tool in a highly complex and dynamic product line. The layer was needed to achieve production goals by delivering the desired material on time, safely, reliability, and predictably.

Companies have been relying on custom made systems to fulfill specific needs. In highly automated factories, such data is commonly collected by a variety of sensors. Shop floors traditionally required data gathering systems such as databases and spreadsheets to collect production data and monitor real-time execution processes. However, this approach could be complex to manage. Manufacturing execution systems overcome the challenge of integrating several point systems by integrating multiple execution components into a standalone solution. MES offers a unified user interface for data management. The demand for utilized manufacturing has led to the development of MES. Together with ERP, MES provides contributes to the various functionalities described in the below Figure 3.1 [28].

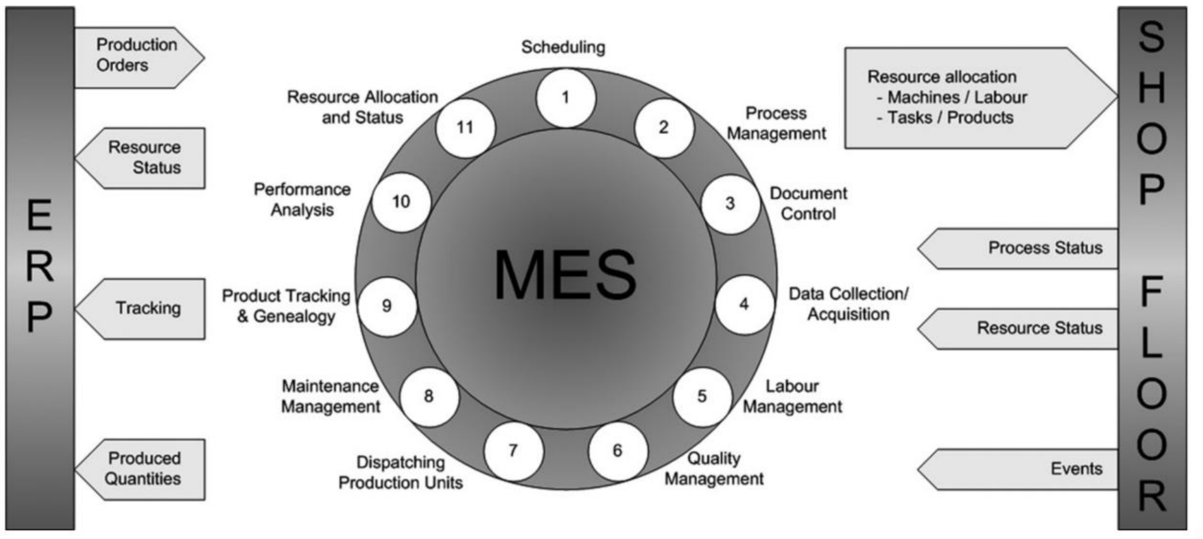


Figure 3.1: Manufacturing Execution Systems

The data used in the master thesis is an industrial encoded data provided by an anonymous industrial partner, and the data was derived from their manufacturing execution system.

3.2 Parameter Selection

This thesis aims to develop a system to predict tool recipe runtimes and study if it is effective. Such a system would depend heavily on historical data. This stresses the importance of defining the predictors from the list of input. In this thesis, the data I used initially had a large set of variables.

After careful consideration, only a subset of those variables has been decided on as predictors. The list of predictors is grouped into the following categories:

- Lot details
- Process attributes
- Operation attributes
- Equipment attributes
- Operating personnel
- Other categorical data

Lot details contain the Lot Item Count, Lot Group Mapped, the Product, Product Type, and Technology used. In contrast, Process attributes are concerned with the Process Step and the Recipe being executed. Operation Attributes are the operation name, type, and group. Equipment attributes specify the equipment name, type, type mapped, group, group mapped, and area. In comparison, the operating personnel category includes the operator's name at the beginning of the shift and the end of the shift. The final list of predictors is listed in below Table 1.

Table 1: List of predictors

Variable	Variable Class	Description
LotIemCount	Numeric	Count of items in the Lot
LotGroupMapped	Character	Mapped grouping of lots
Product	Character	Product manufactured
ProductType	Character	Grouping of products
Technology	Character	Semiconductor technology
ProcessStep	Character	Each operation is part of one or more production schedules and has a subset of definedProcesStep's
ProcessRecipe	Character	The recipe used for the ProcessStep
Operation	Numeric	The identifier for a specified production target. Each operation is part of one or more production schedules and has a subset of a defined ProcessStep
OperationType	Character	Grouping of Operations
OperationGroup	Character	Grouping of OperationTypes
Equipment	Character	Name of the tool
EquipmentType	Character	Grouping of Equipment
EqupmentType Mapped	Character	Mapped grouping of EquipmentTypes
EquipmentGroup	Character	Grouping of EquipmentTypes
EquipmentGroup Mapped	Character	Mapped grouping of Equipment
Area	Character	Production location inside a factory

Sequence1	Character	Route (work plan), lithography layer (mask layer), logical location
Sequence2	Character	Route (work plan), lithography layer (mask layer), logical location
Sequence3	Character	Route (work plan), lithography layer (mask layer), logical location
Sequence4	Character	Route (work plan), lithography layer (mask layer), logical location
Operator	Character	Staff member identification
StartShiftName	Character	Team name of the Shift.
EndShiftName	Character	Team name of the Shift.

3.3 Exploratory Data Analysis

A good practice for any machine learning project is to begin by performing exploratory data analysis. EDA is a powerful approach to provide insights and methods to help identify patterns in data and understand data typically through graphs and Tables for visualization. Through EDA, data can be looked at from different points of view, thus contributing to further understanding of the data and deriving conclusions [29], [30].

The data used in this thesis is actual production data from a semiconductor manufacturer. The data has been anonymized for discretion. The specific attributes that have been anonymized are LotGroupMapped, Product, ProductType, Technology, ProcessStep, Equipment, Operator, StartShiftName, and EndShiftName. The initial description of the numeric columns of the dataset is plotted in below Figure 3.2. It shows that the minimum recipe duration time is 2629.57 seconds. The minimum recipe time elapsed 1 second only, which is a failed recipe or inaccurate data. The maximum recipe time was 368287 seconds. Concerning the LotItemCount, it is observed that the mean LotItemCount is 24.

	DurationSeconds	LotItemCount
count	188026.000000	188026.000000
mean	2629.571628	24.079095
std	5153.162132	4.169733
min	1.000000	0.000000
25%	486.000000	25.000000
50%	1439.000000	25.000000
75%	2562.000000	25.000000
max	368287.000000	25.000000

Figure 3.2: Numeric variables described

The first plot in Figure 3.3 below illustrates the distribution of average process duration by plotting the durations in seconds taken by each Recipe to be executed on a specific tool against the Start Timestamp. The Start timestamp ranges between 2020-07-19 00:00:00 and 2020-09-19 00. Thus, the most significant density for the recipe's duration is between 0 seconds and 2500 seconds. A lower density follows them for recipes requiring 2500 seconds to 5000 seconds. Finally, they were followed by the minor population of some scattered recipes that consumed more than 5000 seconds.

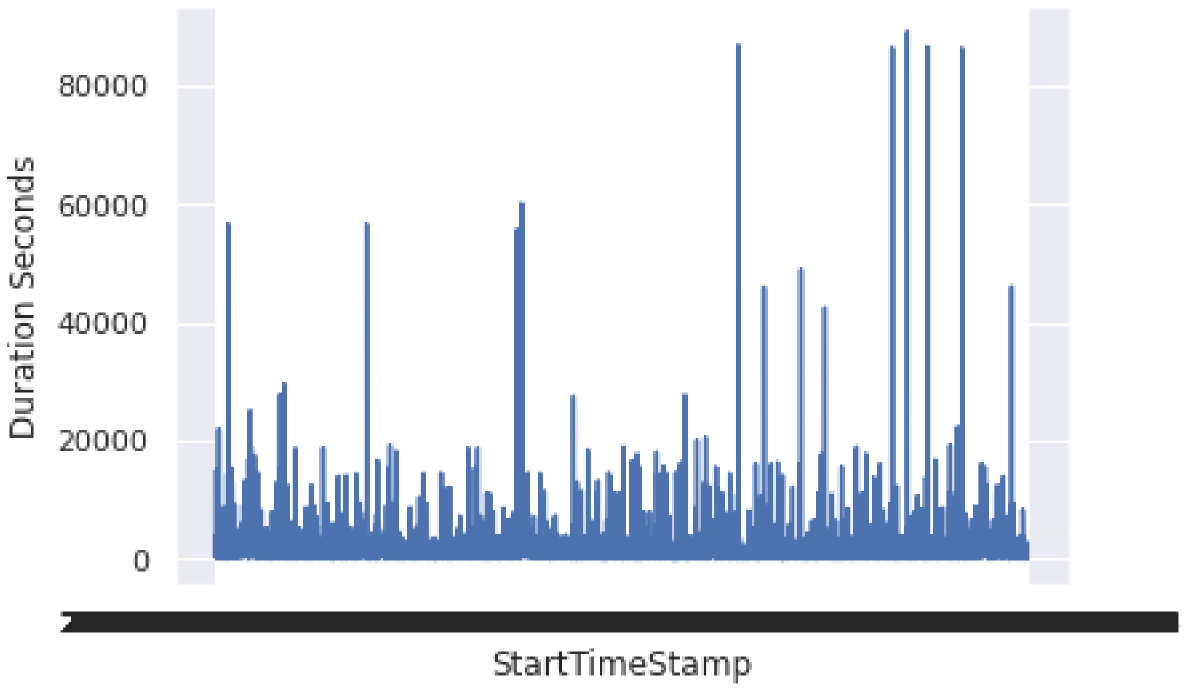


Figure 3.3: Distribution of average process duration

Moreover, The distribution of EquipmentType is plotted below in Figure 3.4. The Figure shows different classifications of Equipment Types such as Single Chamber Tool, Not Mapped, nan_other, MultiChamber Time, Cluster MultiPath Tool, and Batch Tool. The most significant concentration of data is of Single Chamber Tool Equipment Type Mapped with a value of over 100000 records and followed by significantly less Batch Tool with a value a little higher than 40000, Cluster MultiPath Tool with just a little over 20000 records and followed lastly, by the least frequent occurrences of Not Mapped, Multi-Chamber Tool, and the nan_other group.

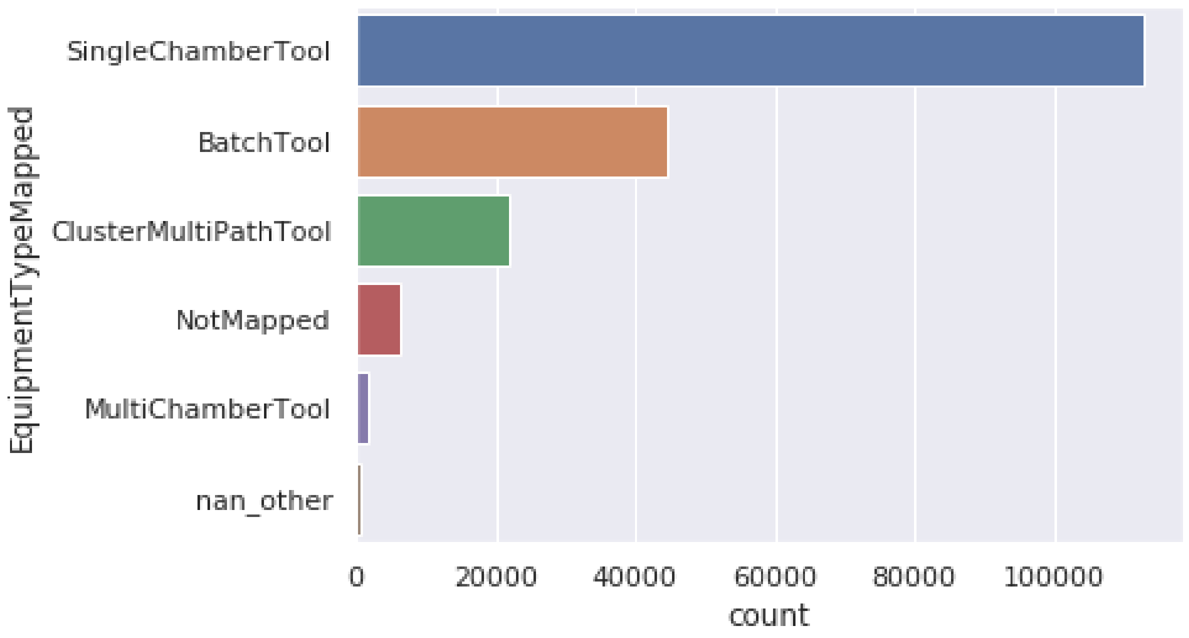


Figure 3.4: Equipment Type Mapped Distribution

Concerning the equipment group distribution, the following is observed in Figure 3.5. The ICON Equipment Group is the major equipment group in the dataset with over 60000 records, followed by the LITH Equipment Group with more than 40000 records, followed by the WPRO equipment group with a little over 30000 records. The equipment groups with values under 20000 records are the ETCH, HCVD. While IMET and nan_other are the least redundant with values lower than 10000 records.

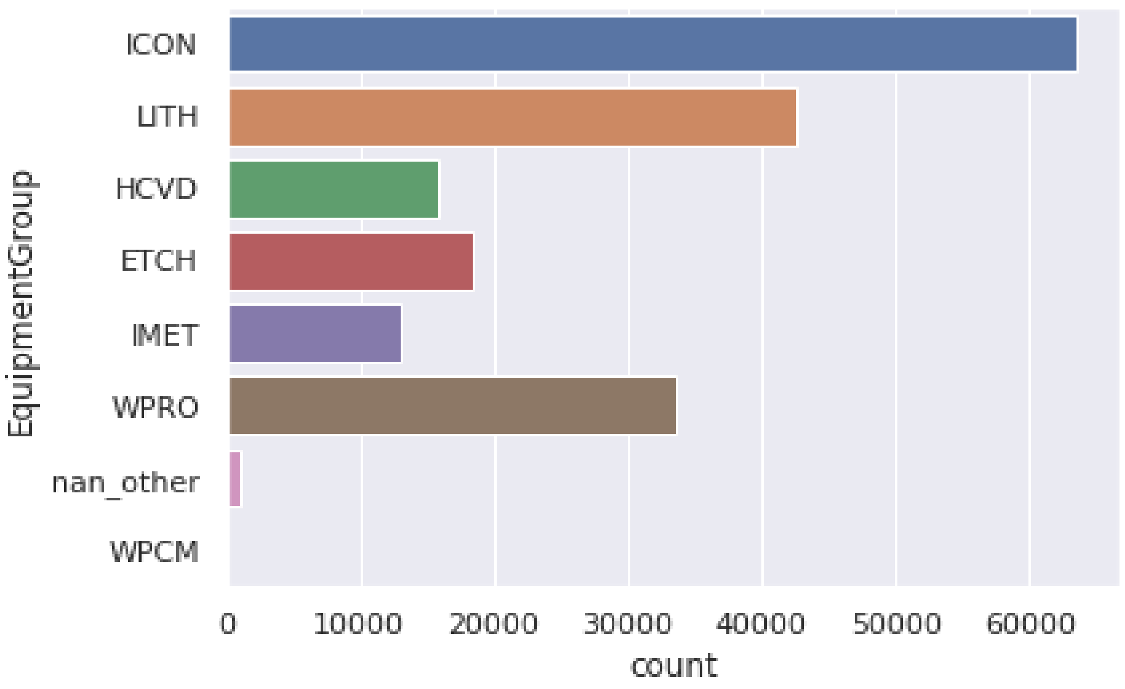


Figure 3.5: Equipment Group Distribution

Regarding the distribution of the different technologies used, the below Figure 3.6 plots the count of each Technology in the dataset. Naturally, there exist some technologies that are more redundant than others. The most frequent technologies used descendingly are Technology 6 and Technology 1, with over 20000 records. While Technology 10, Technology 20, and Technology 0 come after with over 10000 records. The rest of the technologies are of less than 10000 records.

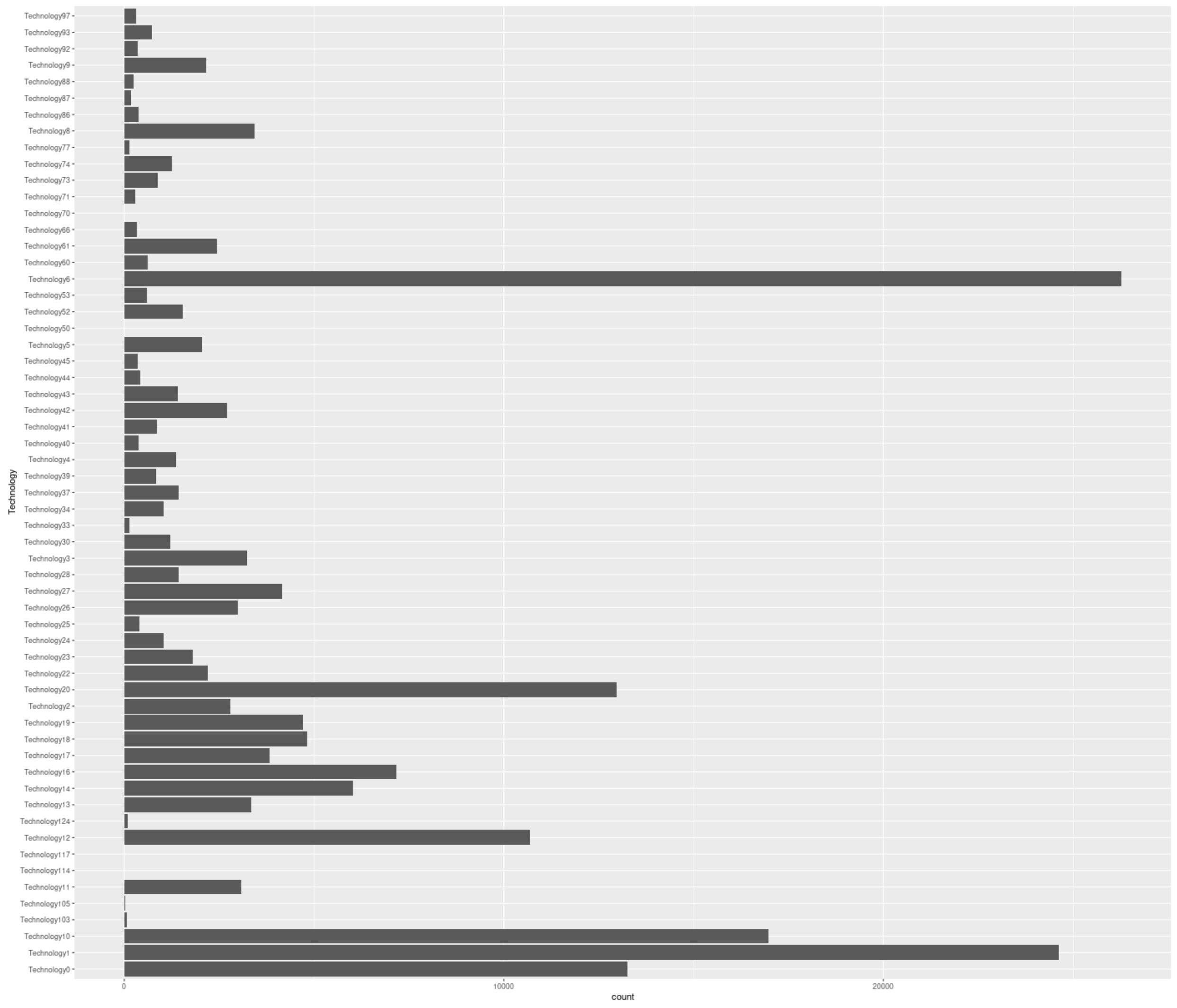


Figure 3.6: Technology Distribution

Moreover, it was helpful to plot process recipes against duration seconds to understand the variability of process recipes. In the below Figure 3.7, the duration in seconds for three process recipes was plotted.

Process Recipe 1 has a considerably large variability between its minimum and maximum values. However, most of the data averaged between 100 and 1000 seconds, with the most frequency around 500 seconds. “Process Recipe 112” has significantly less variability between its maximum and minimum values. However, the most significant portion of occurrence was still around the 500 seconds threshold. Process Recipe 4 had even higher limits' variability. Its distribution mainly occurred around 1000 seconds.

Process Recipe 4 shows the most considerable difference between its maximum and minimum values. However, most values are slightly above the 1000 seconds threshold. However, most values remain dense slightly above the 1000 seconds threshold. This Figure shows that some recipes could have large variability in their data due to large ranges, affecting the prediction process for specific recipes.

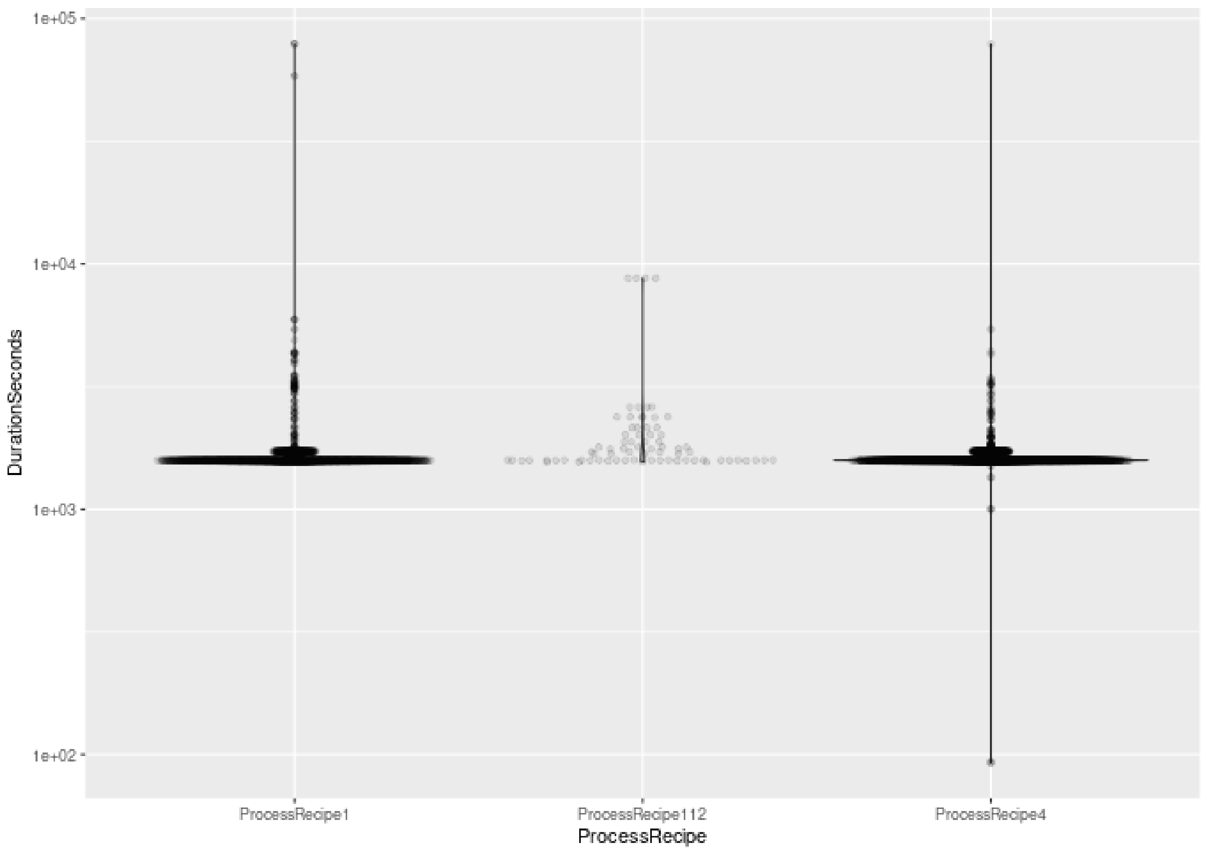


Figure 3.7: Process duration by Recipe

When observing the trends in process time by Recipe as in the below Figure 3.8 for Process Recipe 1 and Process Recipe 4, it is observed that the duration seconds for the individual process recipe are distributed mostly below the 10000 seconds threshold. However, in both process recipes, outliers overs the observed StartTimeStamp are observed.

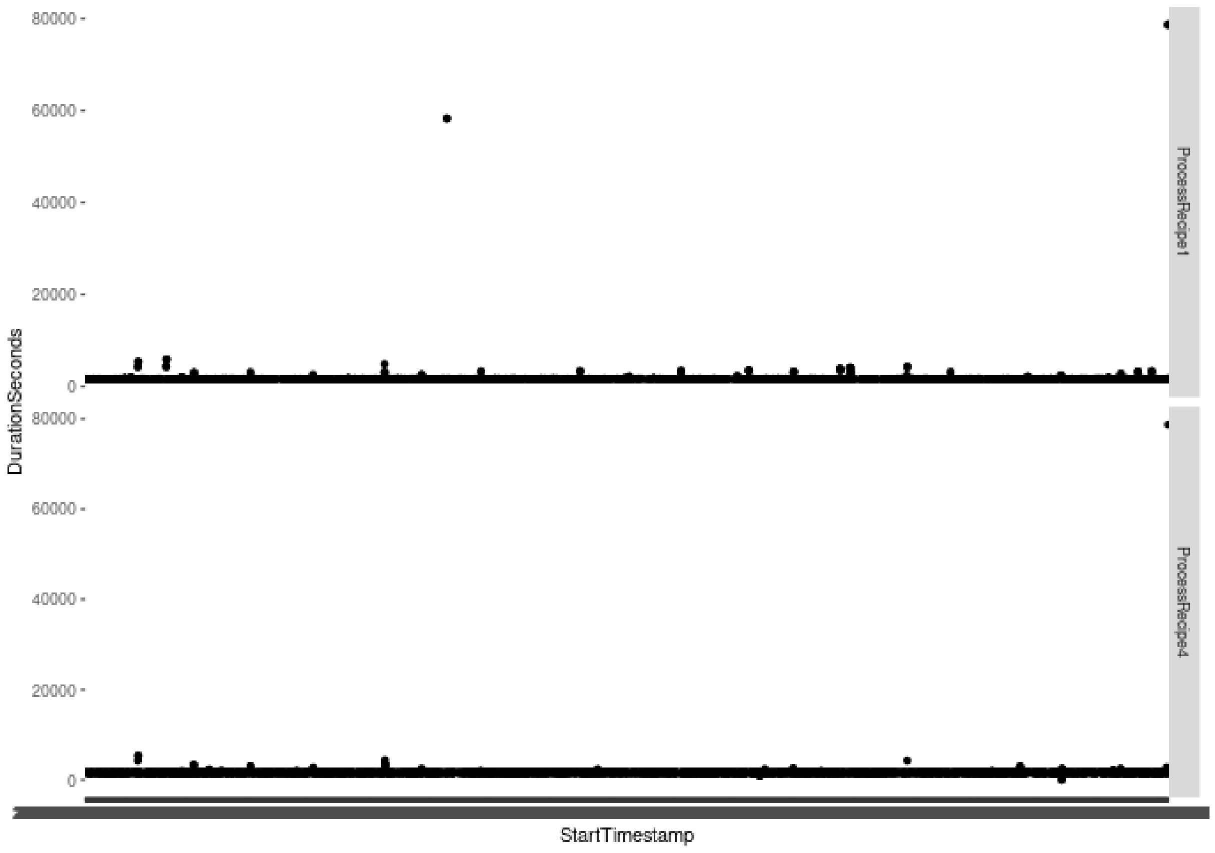


Figure 3.8: Trends in process time by Recipe

Moreover, in the EDA, the LotItemCount shows that the most used LotItemCount is 25, as shown in Figure 3.9. Therefore, a LotItemCount of 25 is the ideal count for having as this is the maximum number of lots possible to be processed later.

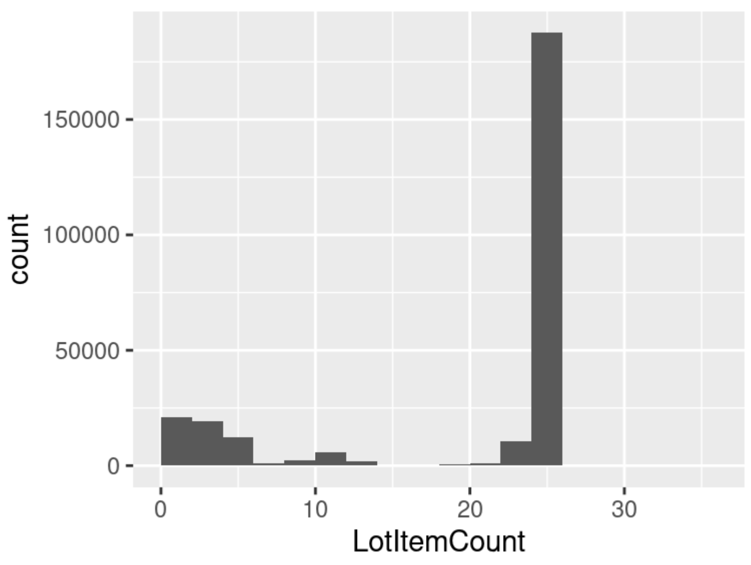


Figure 3.9: LotItemCount

It is also essential to view the correlation between the variables of the data, if any. The below heatmap in Figure 3.10 shows the correlation between the variables. However, unfortunately, there is a very low correlation between the variables.

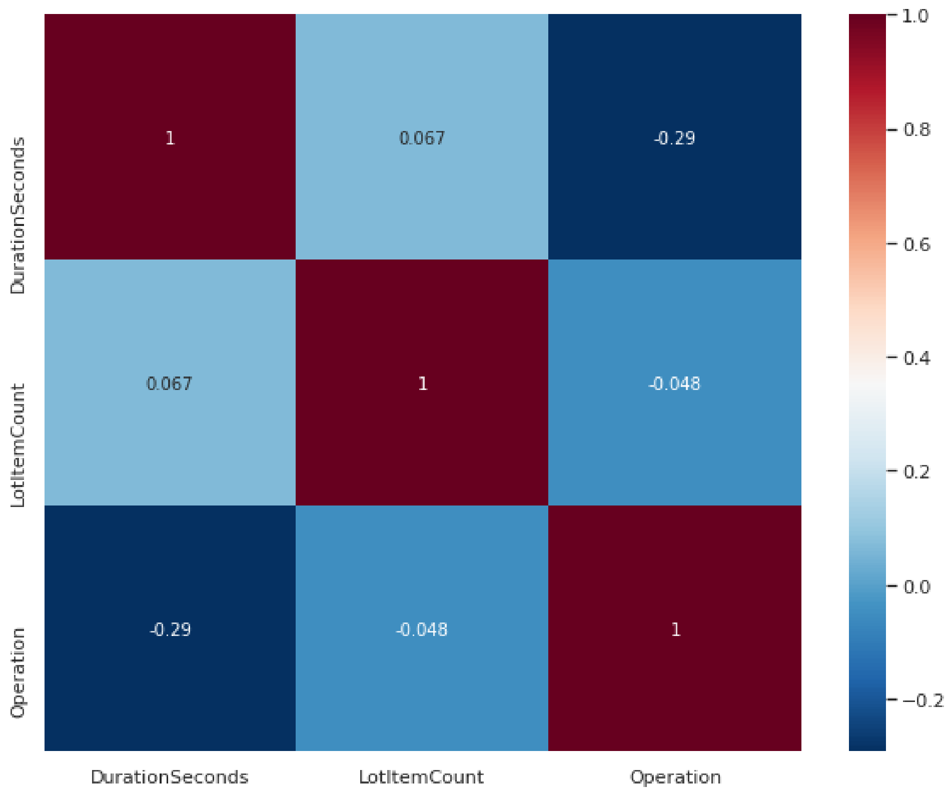
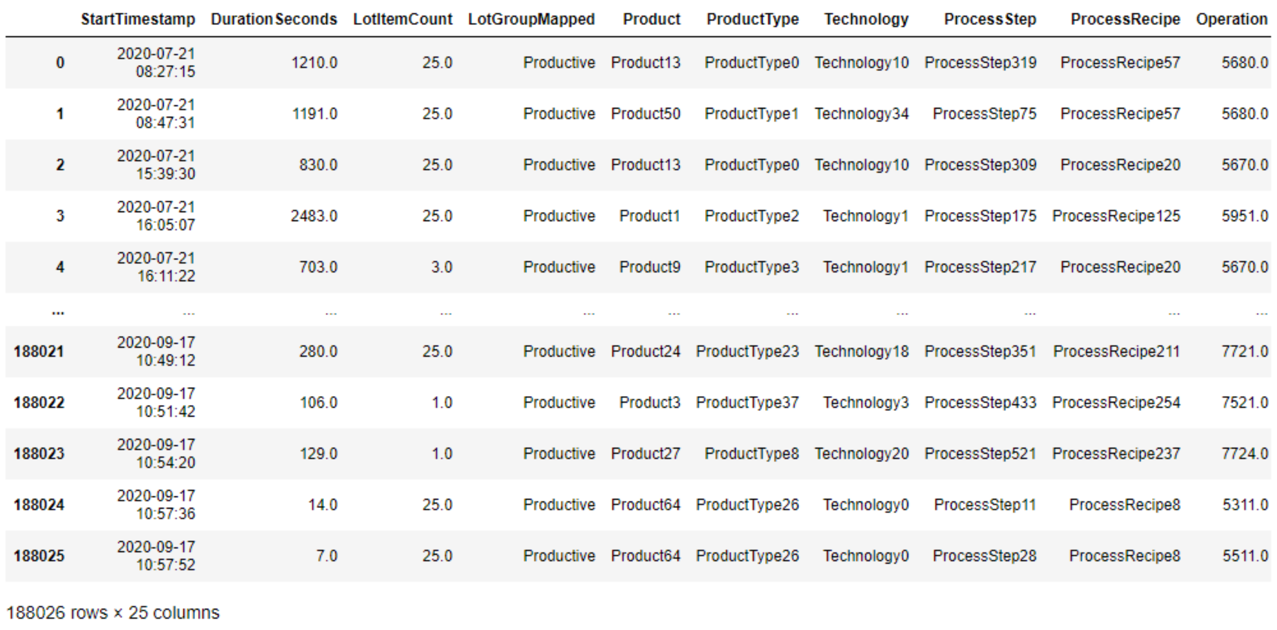


Figure 3.10: Heatmap Correlation Map

After selecting the predictors, the final shape of the dataset is 188026 and has 25 columns, as shown in Figure 3.11.



	StartTimestamp	DurationSeconds	LotItemCount	LotGroupMapped	Product	ProductType	Technology	ProcessStep	ProcessRecipe	Operation
0	2020-07-21 08:27:15	1210.0	25.0	Productive	Product13	ProductType0	Technology10	ProcessStep319	ProcessRecipe57	5680.0
1	2020-07-21 08:47:31	1191.0	25.0	Productive	Product50	ProductType1	Technology34	ProcessStep75	ProcessRecipe57	5680.0
2	2020-07-21 15:39:30	830.0	25.0	Productive	Product13	ProductType0	Technology10	ProcessStep309	ProcessRecipe20	5670.0
3	2020-07-21 16:05:07	2483.0	25.0	Productive	Product1	ProductType2	Technology1	ProcessStep175	ProcessRecipe125	5951.0
4	2020-07-21 16:11:22	703.0	3.0	Productive	Product9	ProductType3	Technology1	ProcessStep217	ProcessRecipe20	5670.0
...	...	...	...	...	...	...	...	...	...	...
188021	2020-09-17 10:49:12	280.0	25.0	Productive	Product24	ProductType23	Technology18	ProcessStep351	ProcessRecipe211	7721.0
188022	2020-09-17 10:51:42	106.0	1.0	Productive	Product3	ProductType37	Technology3	ProcessStep433	ProcessRecipe254	7521.0
188023	2020-09-17 10:54:20	129.0	1.0	Productive	Product27	ProductType8	Technology20	ProcessStep521	ProcessRecipe237	7724.0
188024	2020-09-17 10:57:36	14.0	25.0	Productive	Product64	ProductType26	Technology0	ProcessStep11	ProcessRecipe8	5311.0
188025	2020-09-17 10:57:52	7.0	25.0	Productive	Product64	ProductType26	Technology0	ProcessStep28	ProcessRecipe8	5511.0

188026 rows × 25 columns

Figure 3.11: Tool Recipe Runtimes Prediction Dataset

3.4 Data Engineering

Before and following the machine learning process usually, some data engineering is required. Data engineering is the science of data collection and performing analysis on data. Data scientists aim to retrieve valuable insights from datasets and answer questions to bring value to an organization, usually a business value.

To work with the provided data and apply the developed methods, certain data engineering operations must make the data usable. In this subchapter, a few of the operations that were needed are discussed.

3.4.1 Removing Missing Values

As such extensive data collected from sensors could have some missing values, commonly referred to as Non-Available Values or NaNs. There are several types of missing values, such as Missing entirely at random, missing at random, and not missing at random. Thus, a primitive step in the EDA was to check the presence of NaN values, if any, and their count. Initial analysis showed the presence of such values, and they had to be removed to provide predictions based on the presence of all predictors. Thus, any row that contains any missing column value had to be removed.

There are several ways of dealing with non-available data, including not performing any action, dropping the missing values, or imputing the missing data with a value that could be the mean, median, k-nearest neighbors, or other means.

3.4.2 Encoding Categorical Data

Categorical data are the type of data that contain a certain number of possibilities, groups, or categories, unlike continuous data. Every value of categorical data is a substitute for one of the categories. Categorical data are either nominal or ordinal. To apply statistical methods or machine learning, it is essential to encode categorical data by converting it to numerical data as the algorithms require numbers as input rather than strings [31].

The predictors mostly were categorical data, and it was crucial to encode them. There are several methods to encode categorical data. One-hot-encoding and dummy variables creation were the two methods analyzed, and dummy variables creation was selected for encoding the categorical variables.

Finally, the dataset was split into a training set (60%), a test (20%), and a validation test (20%) for the next steps.

3.5 Chapter Summary

Prior to method development, data must be gathered and prepared for use to be ready for prediction. Firstly, data was gathered from industrial sources and annonmized, a list of predictors was defined. Later, EDA revealed an initial understanding of the data. Finally, data engineering was performed to make the data ready for implementation.

4 Results

Once the data had been prepared, it was possible to proceed with method development. In the method development, several models were built and trained to predict recipe runtimes. These models include different implementation and techniques.

This chapter examines the methods used to study the hypothesis, and the output results are recorded. First, the following flow of events chart was collected to view an abstract, as shown below in Figure 4.1. Then, as discussed in earlier chapters, the process begins by collecting the data, defining the parameters, performing the EDA to understand the data, and performing data engineering to make the dataset ready for implementation.

The implementation phases begin by applying the selected machine learning algorithms on the dataset, training the models, analyzing them, and finally choosing the appropriate model for deployment. In the next subchapters, the results from each model are recorded and analyzed.

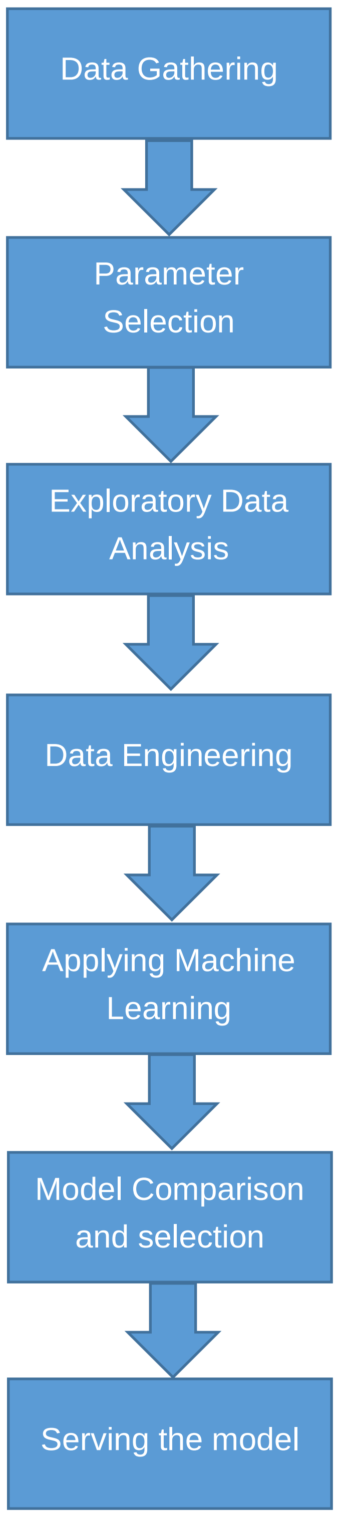

Data Gathering
Parameter Selection
Exploratory Data Analysis
Data Engineering
Applying Machine Learning
Model Comparison and selection
Serving the model

Figure 4.1: The flow of events

4.1 Error Analysis

In this subchapter, the errors from the implementation of each applied ML algorithm are analyzed.

4.1.1 Sklearn GradientBoosting

Boosting is the procedure of improving the accuracy of a model. It is based on the idea that finding and averaging many rules of thumb is more accessible than finding a single, high-precision prediction rule. Gradient Boosting Machine is a straightforward learning ensemble technique. Satisfactory predictive results could be generated by increasing refined approximations.

The following parameters were set for the training of the model. Random state: 42, number of estimators: 90, minimum sample split: 50, maximum features: 60, and maximum depth: 40. The rmse value achieved for Sklearn's Gradient boosting regression on the test data is 2725 seconds and 2133 seconds on the validation data. Figures 4.2 and 4.3 below plot the difference between the predicted values and the actuals after training the model on the validation data. The values up to 3000 seconds were predicted with adequate accuracy. The values from 30000 to 50000 seconds had less accuracy in prediction. The values larger than 50000 seconds were predicted with even greater error.

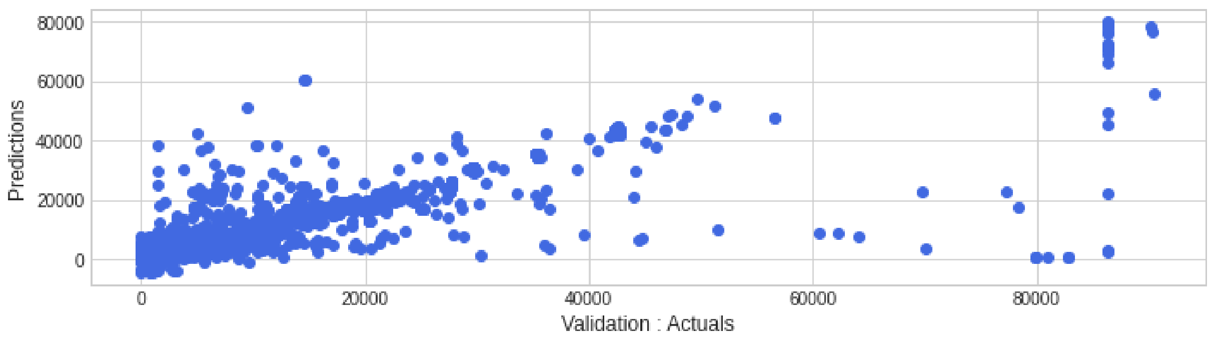


Figure 4.2: Sklean GradientBoosting Baseline Model, Validation Data, Actuals Vs. Predictions

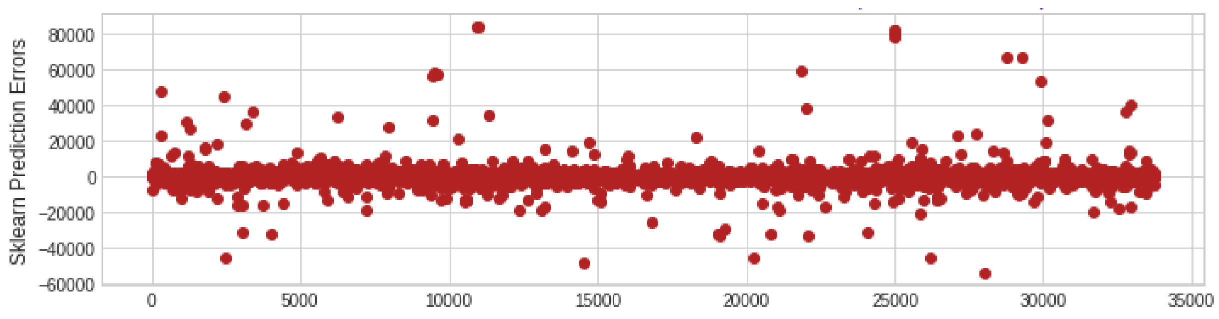


Figure 4.3: Sklean GradientBoosting Baseline Model, Validation data, Prediction Errors

While Figures 4.4 and 4.5 demonstrated the prediction error when training the model on the test data. The predicted values were close to the actuals for values up to 30000 seconds. However, much lower accuracy was observed for larger values.

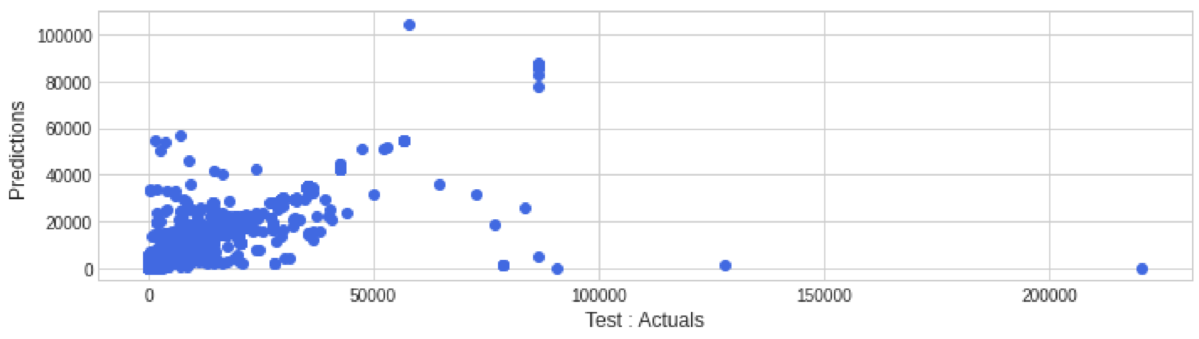


Figure 4.4: Sklean GradientBoosting Baseline Model, Test Data, Actuals Vs. Predictions

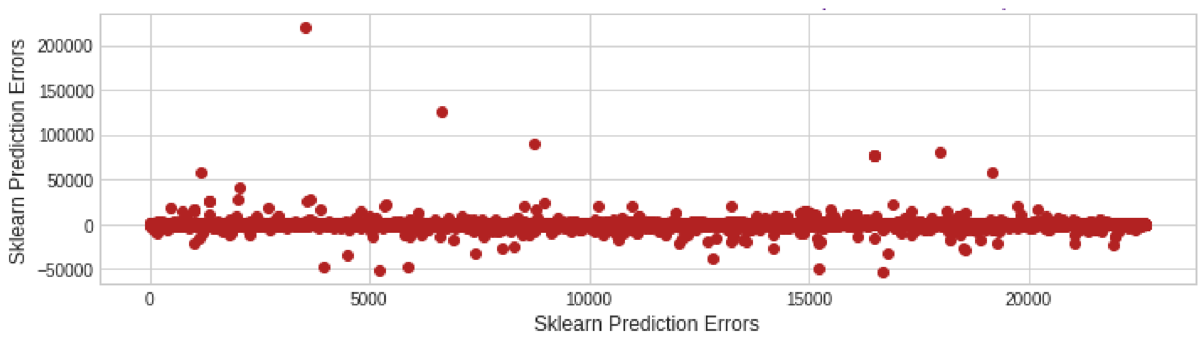


Figure 4.5: Sklean GradientBoosting Baseline Model, Test data, Prediction Errors

When applying sklearn's gradient boosting, a model was created, and the error was measured again in different categories, as shown below in Figure 4.6—first analyzing the rmse against EquipmentGroup on the test data as shown below Figure. The error was the highest for the nan_other group. For the rest of the groups, WPCM Equipment group predictions were the most accurate with a rmse of 641 seconds, followed by

LITH with a rmse value of 919 seconds, followed by ETCH with a value of 1140 seconds. IMET and HCVD showed slightly higher values of 1723 and 1735 seconds, respectively. Finally, the highest error was from the ICON group, with a value of 2571 seconds.

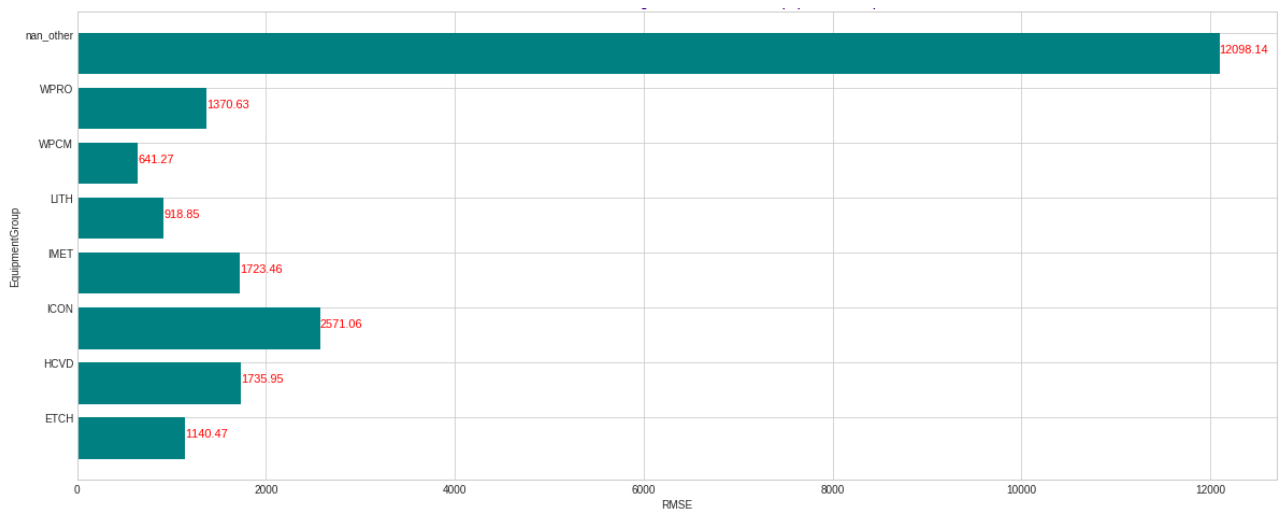


Figure 4.6: Sklearn GBM RMSE per EquipmentGroup – Validation data

However, on the test data, Figure 4.7 demonstrates somewhat different results. The group with the lowest error prediction is the nan_other group with a rmse value of only 305.59 seconds. For the rest of the groups, LITH Equipment Group has the lowest error of 903.1 seconds, followed by IMET with a higher error of 1028.71 seconds. HCVD and WPRO show a higher error of 1844.49 and 2881.98 seconds, respectively. Finally, the highest error was of the ICON equipment group, with a value of 3923.82.

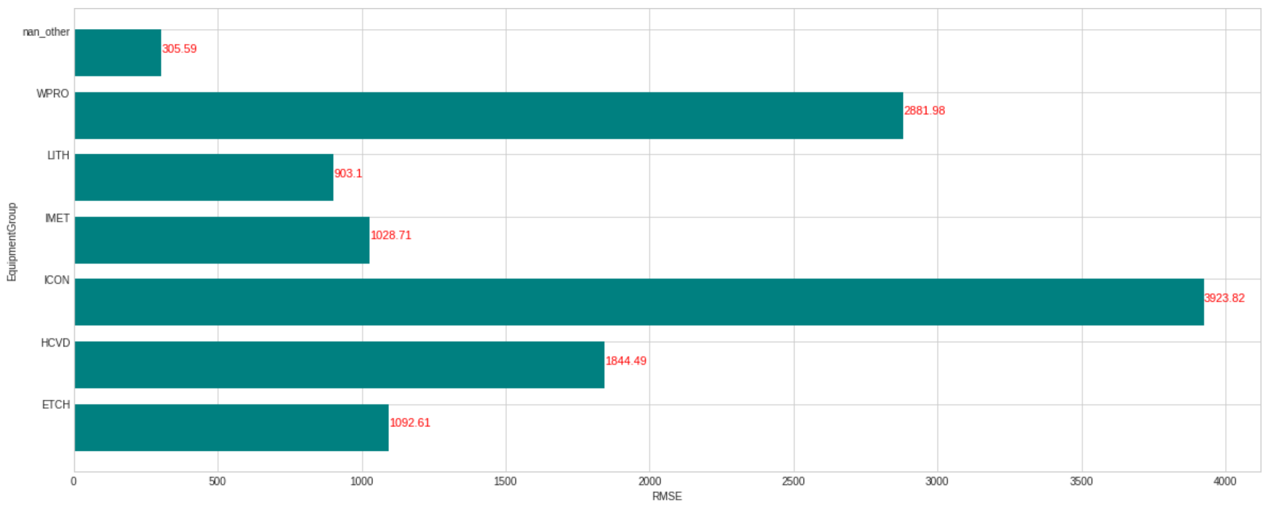


Figure 4.7: Figure 4.6: Sklearn GBM RMSE per EquipmentGroup – Test data

The error was also compared against EquipmentTypeMapped, as shown in Figure 4.8 when training the validation data. EquipmentTypeMapped is the information about the equipment type used. The group with the lowest prediction error was the not_mapped group. The nan_other equipment type had the lowest prediction error of 33.92 seconds for the rest of the groups. MultiChamberTool error was slightly higher with 810.68 seconds, followed by ClusterMultiPathTool with a value of 1365.15 seconds and BatchTool with 1497.9 seconds. The most significant error was of the NotMapped equipment type.

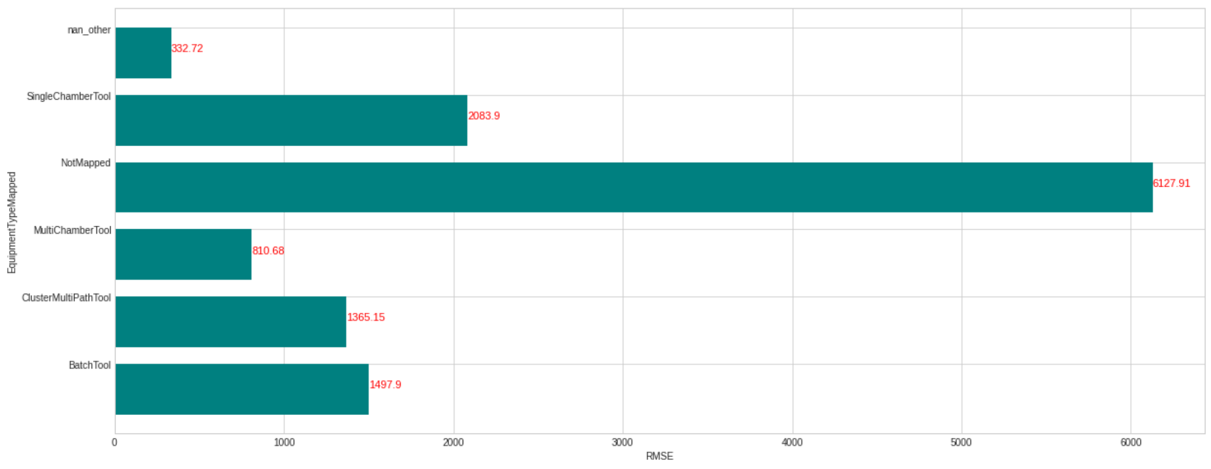


Figure 4.8: Sklearn GBM RMSE per Equipment Type Mapped – Validation data

Different results were observed when training the model on the test data, as shown in Figure 4.9. The lowest prediction error was similar for the nan_other Equipment type. For the rest of the tools, MultiChamberTool shows the lowest error with a value of

1073.41 seconds, followed by ClusterMultiPathTool with a value of 1427.6 seconds, followed by the NotMapped group with a value of 1541seconds. The highest errors were of BtachTool and SingleChamberTool with values of 2357.68 and 3106.24 seconds, respectively.

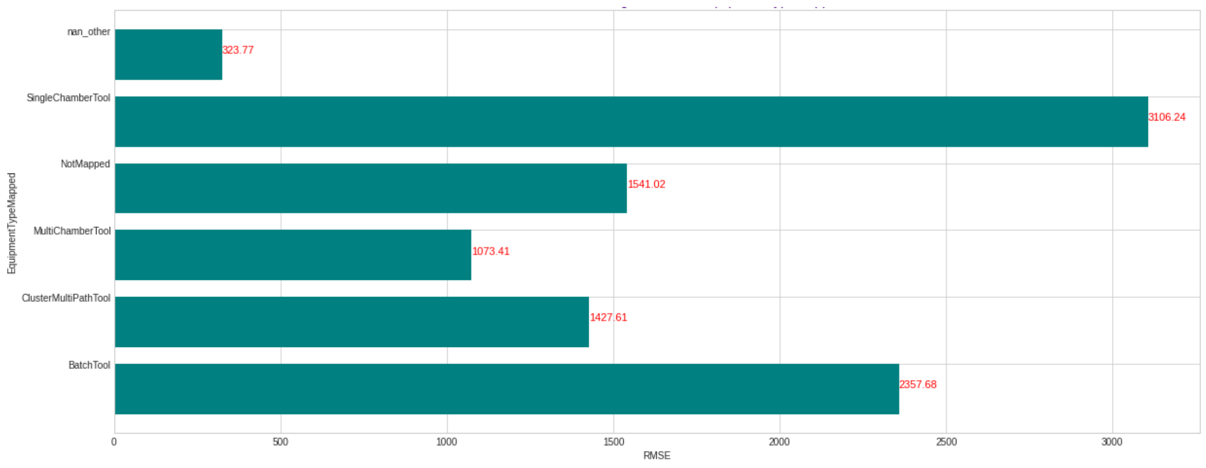


Figure 4.9: Sklearn GBM RMSE per Equipment Type Mapped – Test data

There are various semiconductor manufacturing technologies recorded in the data. Therefore, it was also essential to know if predictions were accurate for specific technology and not for others. As plotted below in Figure 4.10, the results show that Technology 19 had the lowest prediction RMSE of 572 seconds, followed by Technology 5 with an error of 672 seconds. The highest errors were of Technology 17 and Other_level of 3390 and 3786 seconds, respectively. The rest of the technologies demonstrated varying errors. It is unclear the reasons that lead to such differences in errors for different technologies.

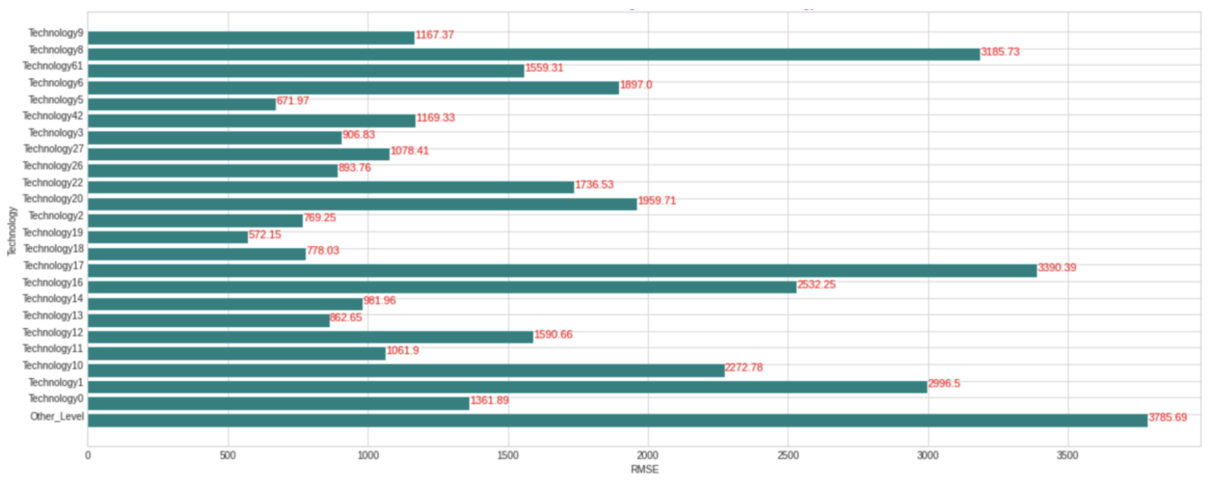


Figure 4.10: Sklearn Gradient Boosting RMSE against Technology – Validation data

As for the test data, the below results are observed in Figure 4.11. Technology 26 had the lowest prediction RMSE of 656 seconds, followed by Technology 9 with an error of 695 seconds. The highest errors were of Technology 8 and Technology 12 of 5244 and 6105 seconds, respectively. The rest of the technologies demonstrated varying errors.

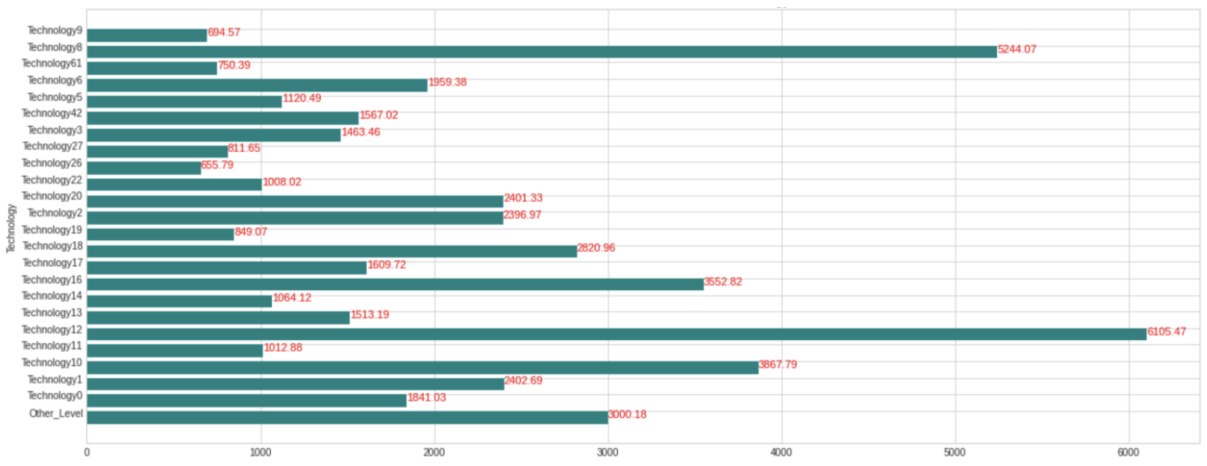


Figure 4.11: Sklearn Gradient Boosting RMSE against Technology – Test data

As explained earlier, recipe runtimes in semiconductor manufacturing are composed of many steps and substeps. Analyzing the root mean squared error for process recipes showed the following as illustrated below Figure 4.12, ProccessStep8 had the lowest rmse value of only 76.33 seconds, followed by ProcessStep9 with a rmse of 163.42 seconds. For unclear reasons, Other_Level process recipes show the highest error of 2522.12. seconds, followed by ProcessStep3 with an error of 1708.15 seconds. The remaining of the rmse values ProcessSteps varied.

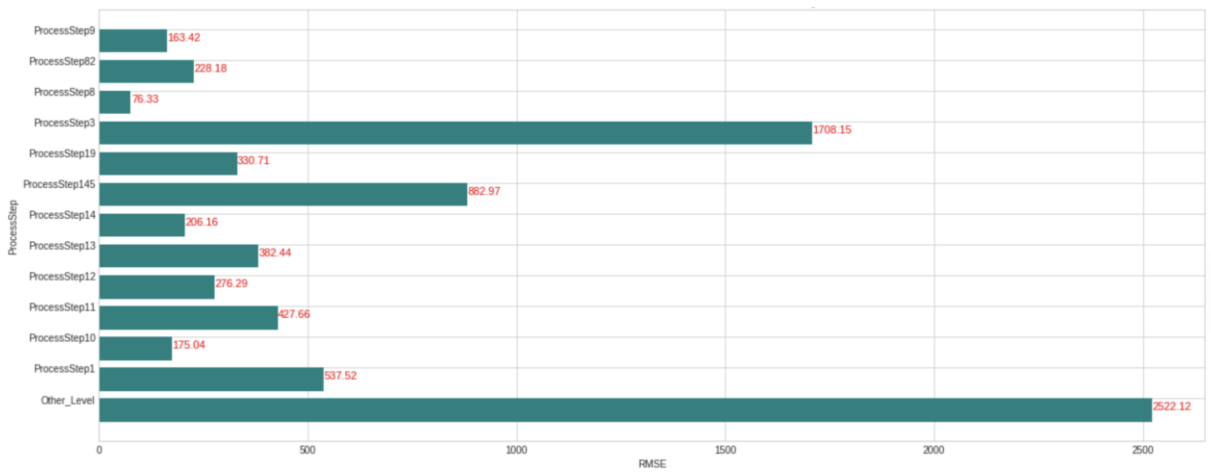


Figure 4.12: Sklearn GBM RMSE vs ProcessStep - Validation data

Figure 4.13 shows the results achieved on the test data. Similar to the validation dataset ProccessStep8 had the lowest rmse value of only 64.59 seconds. However, it was followed by ProcessStep82 with a rmse of 187.7 seconds, ProcessStep3 with 222.24 seconds, ProcessStep12 with 226.5 seconds. ProcesStep9 obtained a rmse of 512.46 seconds. For unclear reasons, ProcessStep13 shows the highest error of 7618 seconds, followed by ProcessStep1 with 6611.97 seconds. The remaining rmse values of ProcessSteps varied.

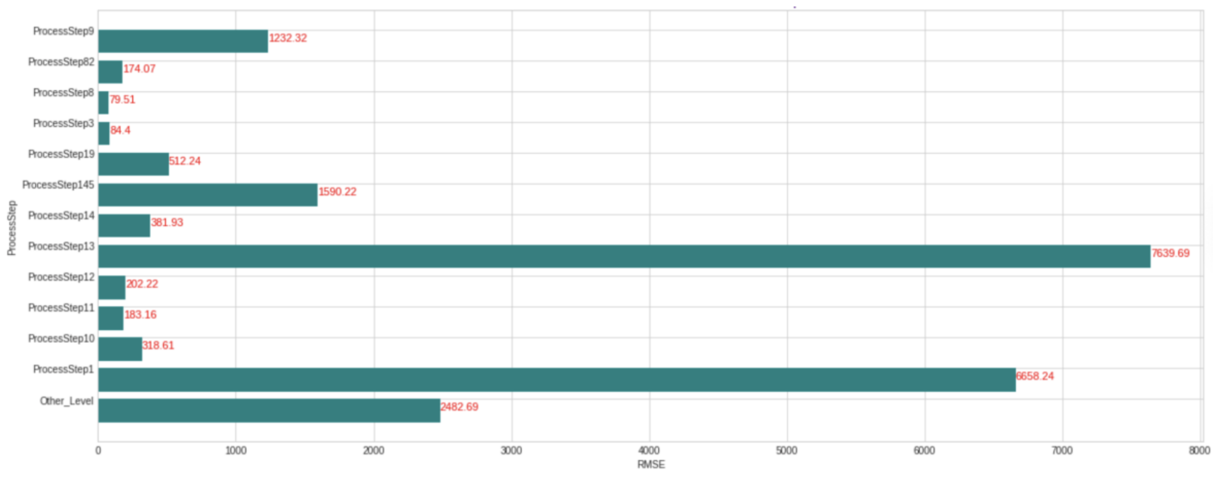


Figure 4.13: Sklearn GBM RMSE vs ProcessStep - Test data

4.1.2 *Sklearn Random forests*

In this section, the results from applying sklearn's Random forests implementation are reviewed. In the implementation, sklearn's Random Forests regressor was first chosen to build a random forests regression model. Several parameters could be changed; otherwise, they are set to the default values of sklearn. In this model, two parameters were adjusted. The number of estimators is the number of trees in the forest, and it was set to 90 estimators [32]. The random state controls the randomness of the bootstrapping of the samples used when building trees, and the sampling of the features to consider when looking for the best split at each node was set to 4.6.

The rmse value achieved for Sklearn's Random Forests on the test data is 2759.72 seconds and 2165.13 seconds on the validation data.

Analyzing the error for specific equipment groups as illustrated in Figure 4.14, When applying sklearn's random forests to the validation data, the following errors appear per Equipment Group. The lowest error is for the WPCM Equipment Group, with an error of 272.96 seconds. Regarding the rest of the equipment groups, the LITH equipment group had the second. The lowest rmse value of 980.29 seconds, followed slightly by the ETCH equipment group with a value of 1339.02 seconds, followed by a higher value for the WPRO equipment group with a value of 1487.09, followed by the HCVD equipment group with a higher value of 1693.01 seconds, followed by IMET equipment group with a rmse value of 1912.91 seconds. The highest error in prediction was off the ICON equipment group and the nan_other group having an error of 2560.56 seconds and 12034.83 seconds, respectively.

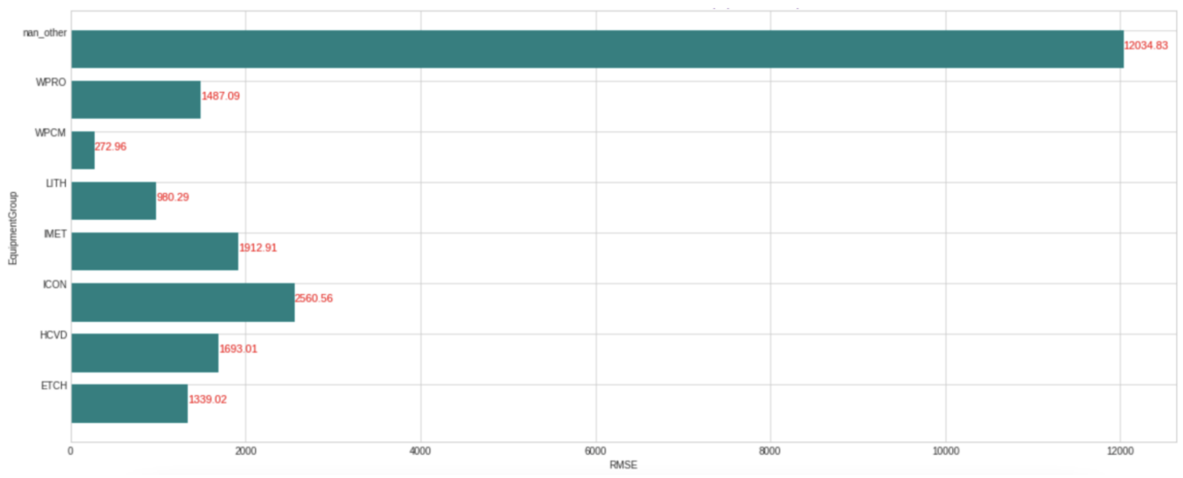


Figure 4.14: Sklearn RandomForest RMSE per EquipmentGroup – Validation data

The results from the test data are plotted in below Figure 4.15. In contrast with the results from the validation data, the nan_other group has the lowest rmse from the test data with an error of 356.69 seconds. For the rest of the groups, LITH Equipment Group has the lowest rmse with an error of 922.3 seconds, and similarly to the validation data followed in rank by the ETCH Equipment Group with an error of 1151.44 seconds, followed by HCVD with an error of 1825 seconds, followed by IMET with an error of 2132.32 seconds, followed by WPRO with an error of 2931.74 seconds. Like the validation data, besides the nan_other group, ICON comes last in rank with the highest prediction error with a rmse of 3885.83 seconds.

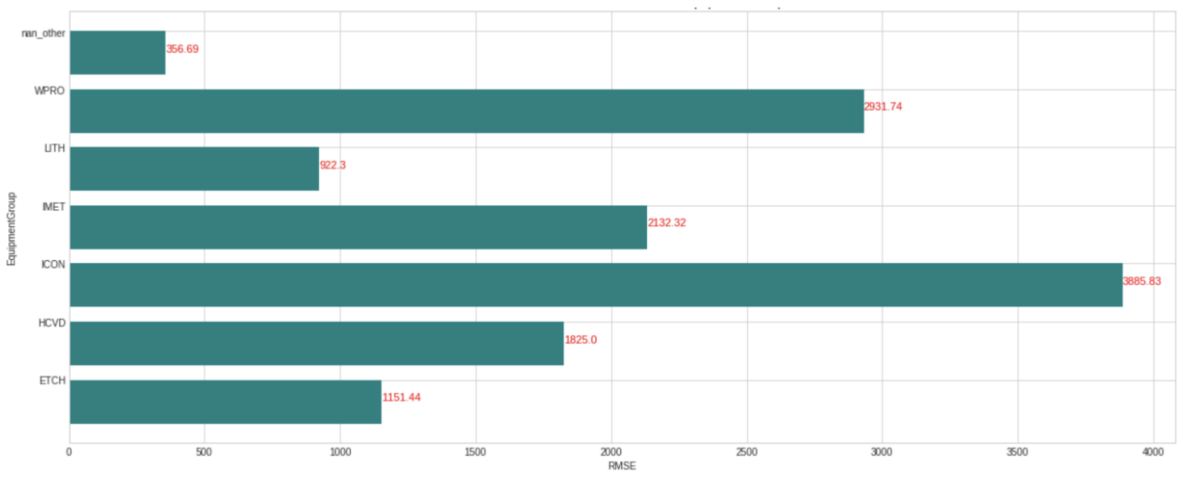


Figure 4.15: Sklearn RandomForest RMSE per EquipmentGroup – Test data

Analyzing the error when applying Sklearn’s Random Forest per EquipmentTypeMapped shows the below results plotted in Figure 4.16. The lowest error is of the nan_other category, with an error of 262.07 seconds. For the rest of the categories, MultiChamberTool shows the lowest error of 675.84 seconds, followed by ClusterMultiPathTool with an error of 1458.72, followed by a higher error of 1577.72 for the BatchTool category, followed by an error of 2082.75 seconds for the SingleChamberTool Equipment Type. The highest error in prediction was off the NotMapped category with an error of 6247.04 seconds.

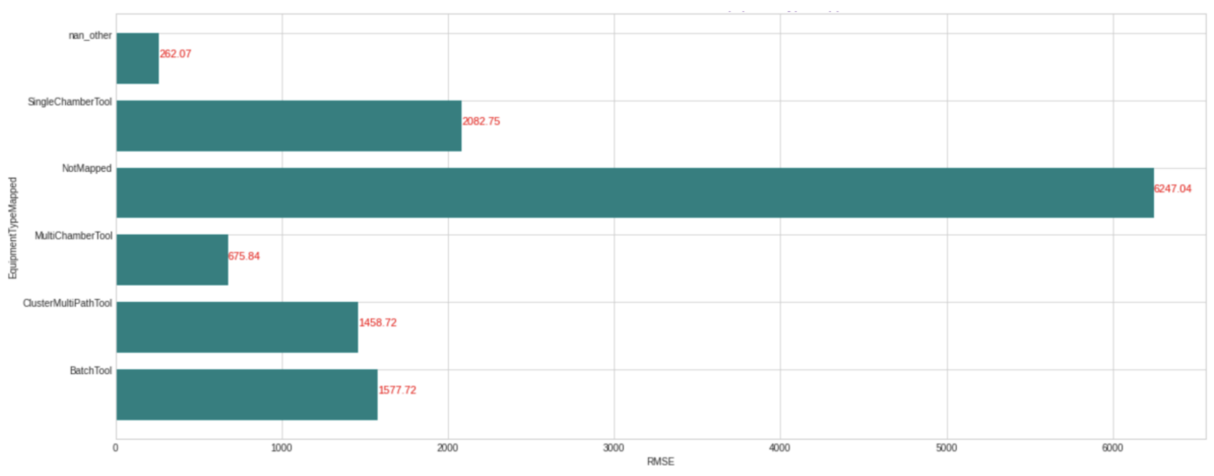


Figure 4.16: Sklearn RandomForest RMSE per EquipmentTypeMapped - Validation data

Applying sklearn's Random forests algorithm on the test data also showed the following rmse errors against EquipmentTypeMapped as plotted in Figure 4.17. The lowest error is of the nan_other category with an error of 333.25 seconds, similarly to the validation error. For the rest of the categories, MultiChamberTool shows the lowest error of 962.6 seconds, followed slightly by ClusterMultiPathTool with an error of 1475.29, followed by a higher error for the NotMapped category of 2035.6 seconds, followed by an error of 2393.9 for the BatchTool Equipment Type. The highest error in prediction was off the SingleChamberTool with an error of 3127.27 seconds. NotSpecified operation group with values of 1456 seconds and 3529.1 seconds, respectively. Nan_other, MultiChamberTool, and ClusterMultiPathTool show the same rank in errors across both the test and validation data.

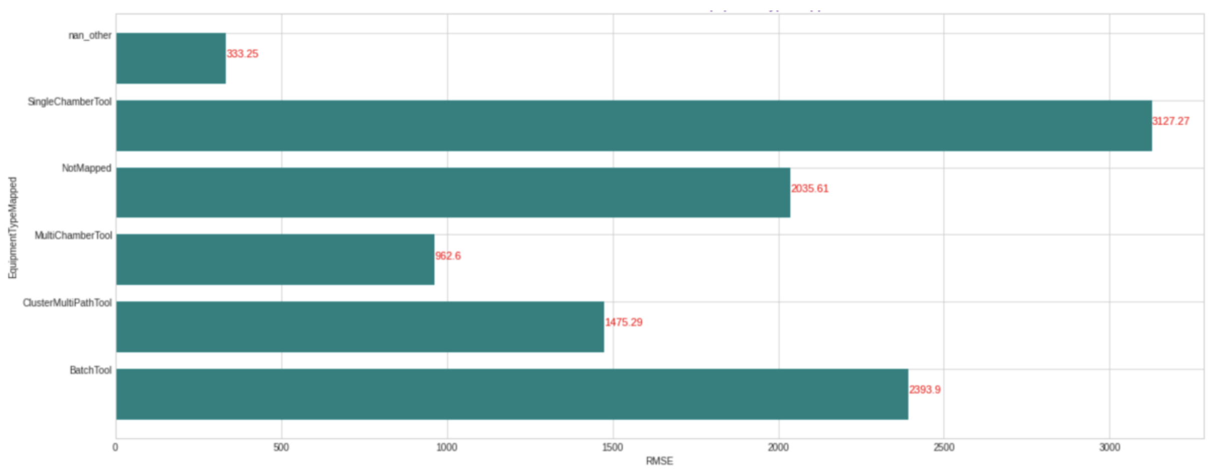


Figure 4.17:Sklearn RandomForest RMSE per EquipmentTypeMapped - Test data

Comparing the rmse values aginst OperationGroup shows the following errors of the validation data as plotted below in Figure 4.18. The lowest error is of the MaskLayerOperation operation group with an error value of 411.58 seconds, followed by the PE_SiN-SiO operation group with an error of 736.41, followed by the Developer operation group with a value of 633.69 seconds, followed by the PE-SiN-SiO operation group with a value of 772.12 seconds, followed by the Coater operation group with a value of 1036.45 seconds. The highest errors are for the Other_Level operation group and NotSpecified groups, with errors of 2811.19 and 2243.9 seconds.

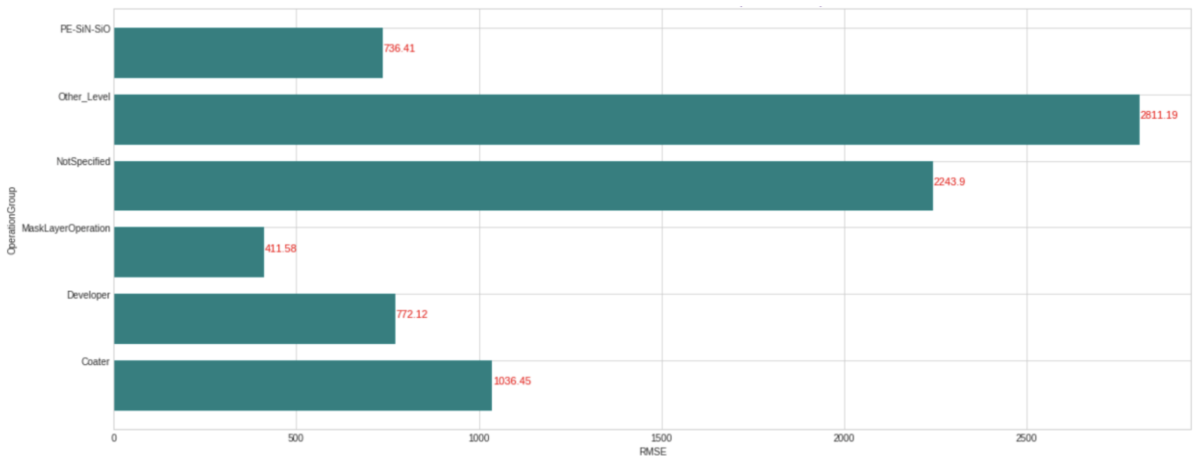


Figure 4.18: Sklearn RandomForest RMSE per OperationGroup – Validation data

Comparing the rmse values aginst OperationGroup shows the following errors of the test data as plotted below in Figure 4.19. The lowest error is of the MaskLayerOperation operation group with an error value of 175.64 seconds, followed by the PE-SiN-SiO operation group with a value of 816.64 seconds, followed by the Developer operation group with a value of 1106.65, followed by the Coater operation group with a value of 1291.78 seconds, followed by the Other_level operation group with a value of 1497.98 seconds. The highest error is of the NotSpecified Operation Group, with a value of 2982.23 seconds.

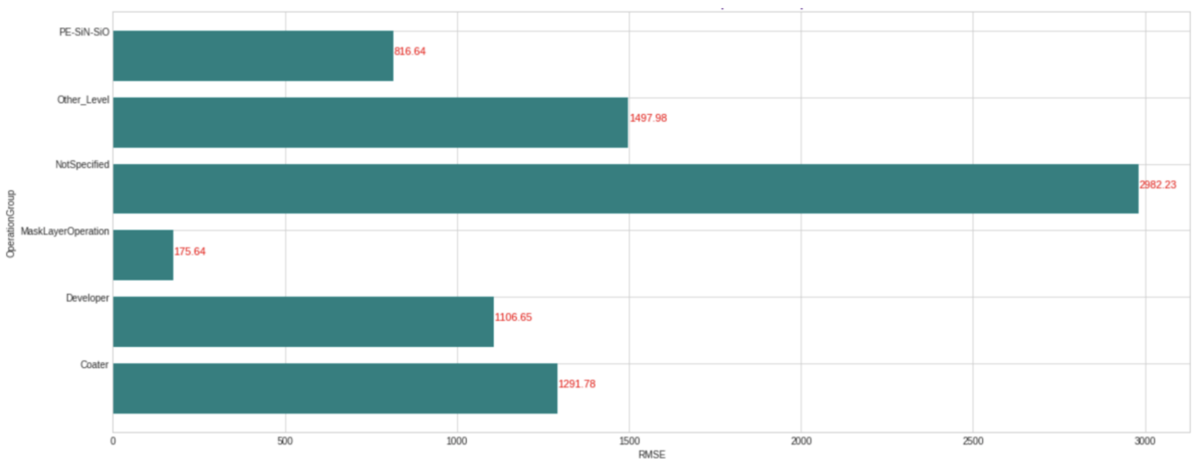


Figure 4.19: Sklearn RandomForest RMSE per OperationGroup – Test data

For comparing RMSE values of the validation data per technology, the following errors in below Figure 4.20 are shown. The lowest prediction error is for Technology 19 with a value of 578.54 seconds, followed slightly by Technology 5 with a value of 737.77 seconds. The highest error is for Other_level with a value of 3756.11 seconds. The remaining values varied from 775.37 seconds (Technology 61) to 3395.6 seconds (Technology 17).

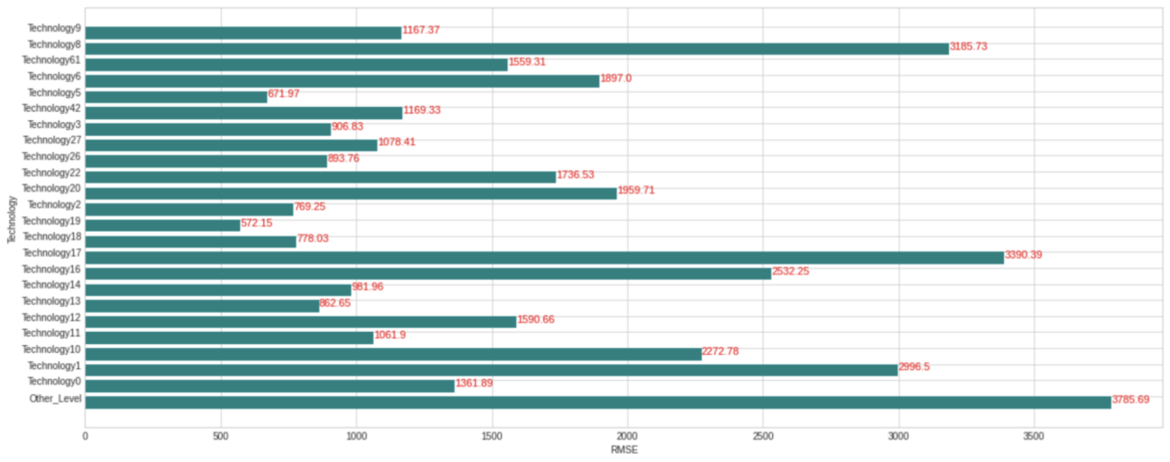


Figure 4.20: Sklearn RandomForest RMSE per Technology – Validation data

For comparing RMSE values per technology on the test data, the following errors in below Figure 4.21 are shown. The lowest prediction error is for Technology 26 with 655.79 seconds, followed slightly by Technology9 with 694.57 seconds. The highest error is for Technology 12, with a value of 6105.47 seconds. The remaining values varied from 750.39 seconds (Technology 61) to 5244.07 seconds (Technology 8).

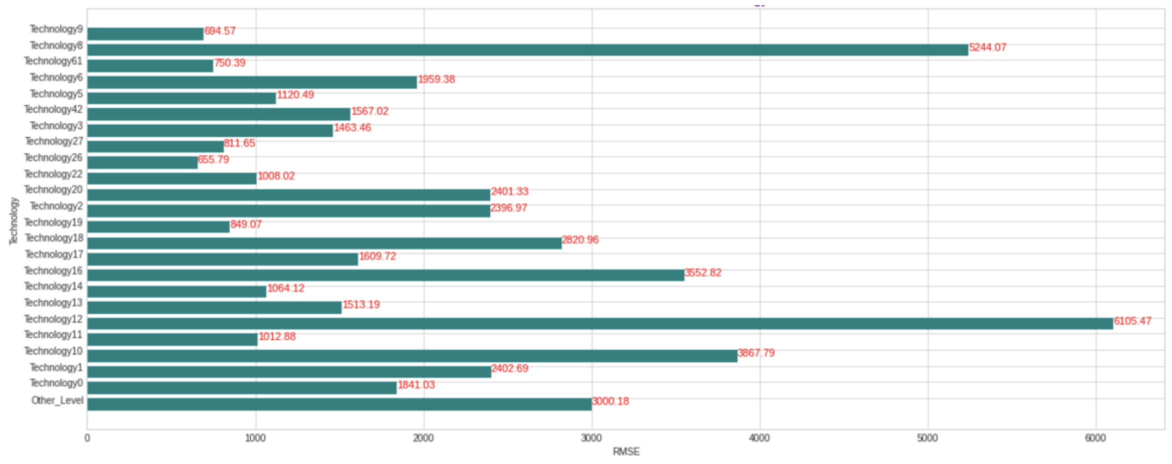


Figure 4.21: Sklearn RandomForest RMSE per Technology – Test data

Lastly, Sklearn's random forests show the following errors per Process steps as plotted in Figure 4.22. The lowest error is 76.33 for ProcessStep8, followed by a higher error of 163.42 seconds for ProcessStep9. The highest error in prediction is of 2522.12 seconds for Other_level. The rest of the errors varied between 176.04 seconds for ProcessStep10 and 1708.15 seconds for ProcessStep3.

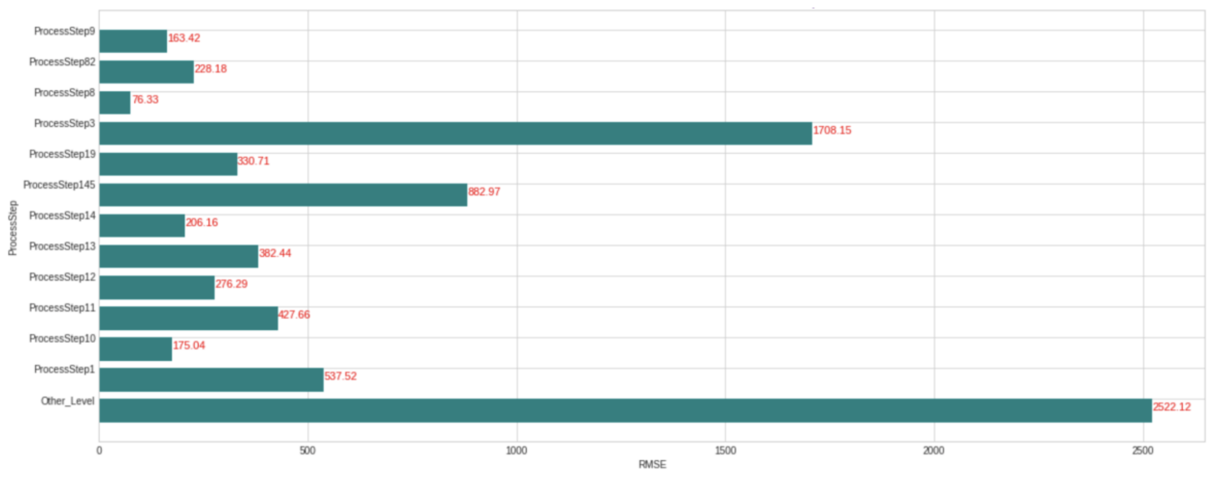


Figure 4.22: Sklearn RandomForest RMSE per ProcessStep – Validation data

LSklearn's random forests show the following errors for the test data per Process steps as plotted in Figure 4.23. The lowest error is also for ProcessStep8 with 79.51 seconds, followed by 84.4 seconds rsme for ProcessStep3. The highest error in prediction is of

7639.69 seconds for ProcessStep13. The rest of the errors varied between 174.07 seconds for ProcessStep82 and 6658.24 seconds for ProcessStep1.

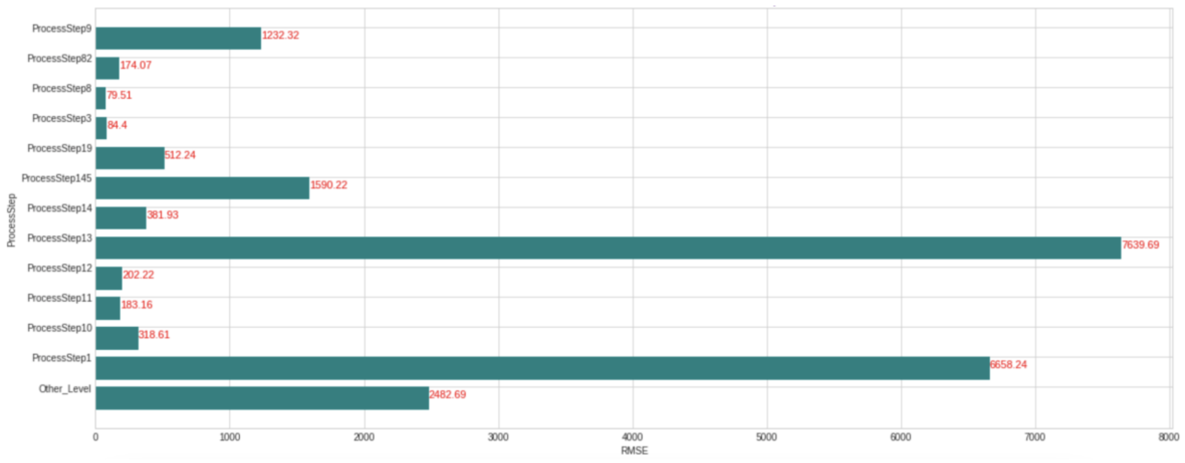


Figure 4.23: Sklearn RandomForest RMSE per ProcessStep – Test data

4.1.3 H2O Gradient Boosting

Gradient Boosting Machine (for Regression and Classification) is an ensemble method for learning. The guiding principle is that with increasingly more satisfactory approximations, good forecasting outcomes can be produced. H2O's GBM creates regression trees progressively on all of the dataset's characteristics in a fully distributed manner. Each tree is generated in parallel.

A further step in building predictive models is applying H2O’s Gradient boosting machines on the data. The rmse value achieved for H2O Gradient boosting regression on the test data is 2894.1 seconds and 2330.55 seconds on the validation data.

The error per EquipmentGroup is as shown below in Figure 4.24 for the validation data. The highest error in prediction is for the nan_other equipment group with a rmse value of 12190.49. For the rest of the equipment groups, the lowest value is of 828.55 seconds for the WPCM equipment group, followed by LITH with a value of 913.16 seconds, followed by ECTH with a value of 1157.71 seconds, followed by WPRO with a value of 1458.68 seconds, followed by IMET with an error of 1750.25. The highest errors are of HCVD and ICON equipment groups with 1840.51 and 29474.07 seconds, respectively.

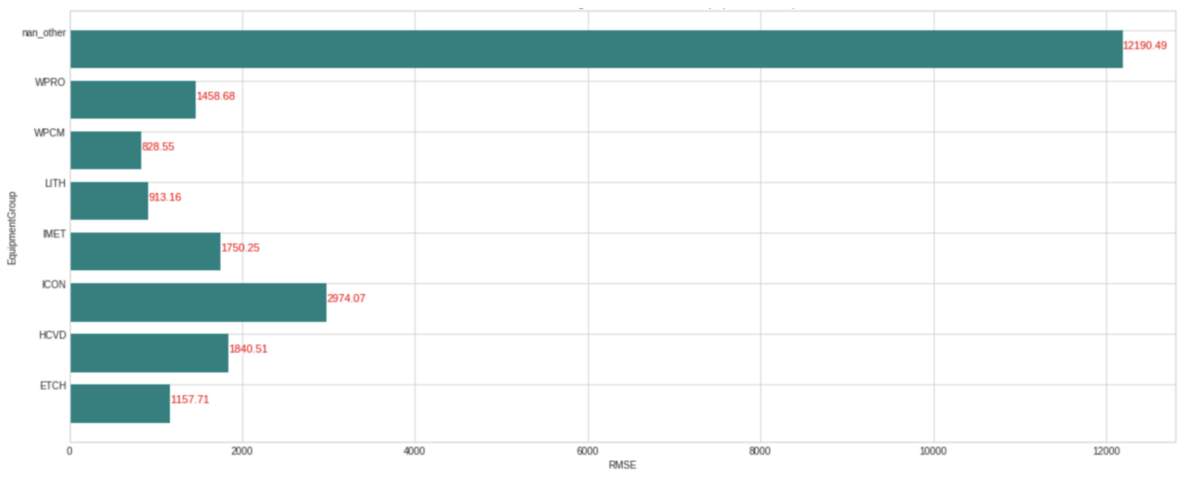


Figure 4.24: H2O GradientBoosting RMSE per EquipmentGroup – Validation data

The error per EquipmentGroup for the test data is shown below in Figure 4.25. The highest error in prediction is for the ICON equipment group with a rmse value of 4263.98 seconds. For the rest of the equipment groups, the lowest value is of 347.42 seconds for the nan_other equipment group, followed by LITH with a value of 887.46 seconds, followed by ECTH with a value of 1070.96 seconds, followed by IMET with a value of 1014.58 seconds, followed by HCVD with 1792.74 seconds, lastly followed by WPRO with a value of 2903.86 seconds.

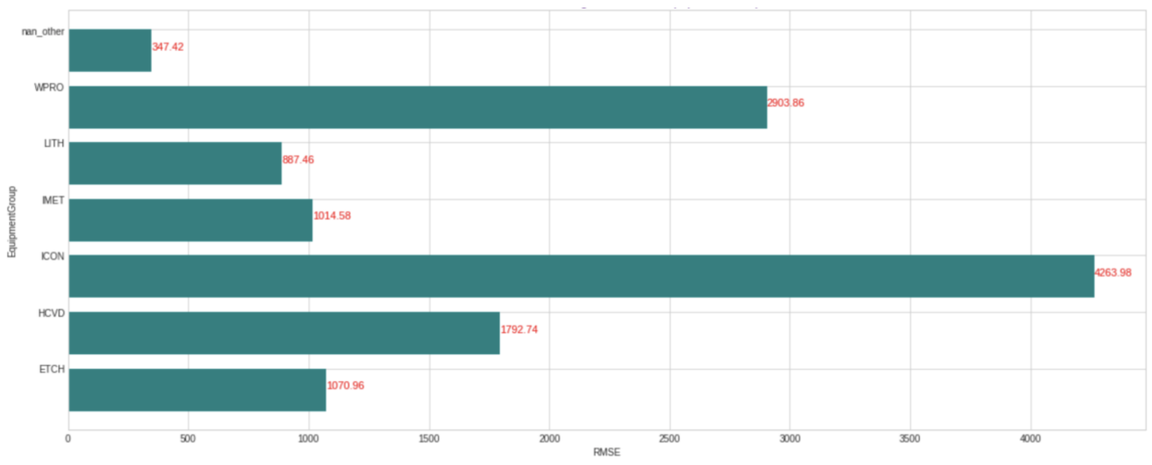


Figure 4.25: H2O GradientBoosting RMSE per EquipmentGroup – Test data

The subsequent analysis is comparing the rmse value for each EquipmentTypeMapped as shown in Figure 4.26. The lowest error is for the nan_other equipment type mapped with an error of 447.89 seconds. For the rest of the categories, MultiChamberTool has the lowest error of 986.7 seconds, followed by

ClusterMultiPathTool with a value of 1472.83 seconds, followed by BatchTool with a value of 1555.52 seconds, followed by SingleChamberTool with a value of 2368.95 seconds. The highest error is for the NotMapped equipment type mapped with an error of 6232.12 seconds.

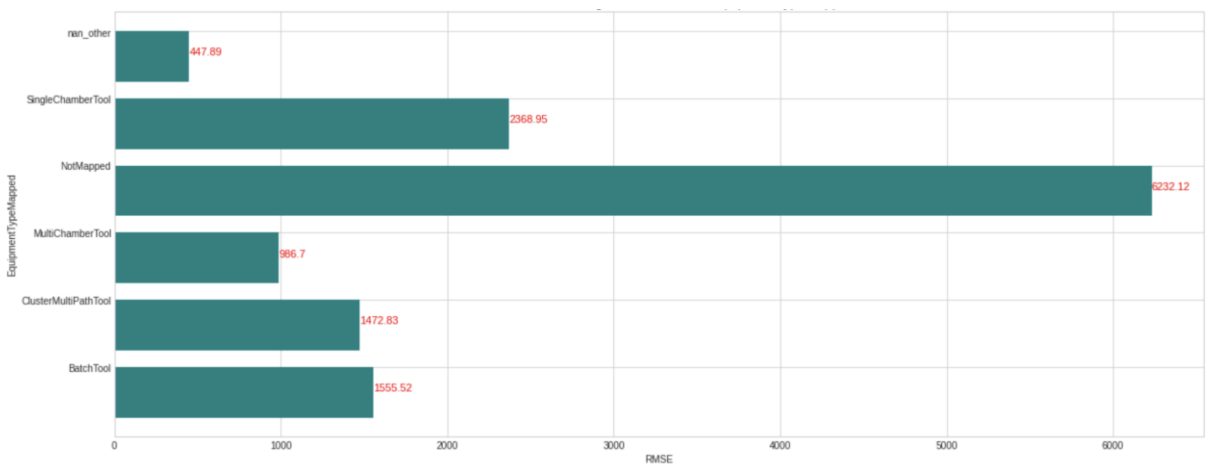


Figure 4.26: H2O GradientBoosting RMSE per EquipmentTypeMapped – Validation data

The following analysis compares the rmse value for each EquipmentTypeMapped as shown in Figure 4.27. The lowest error is for the nan_other equipment type mapped with a rmse of 421.13 seconds. For the rest of the categories, MultiChamberTool has the lowest error of 1101 seconds, followed by NotMapped with 1461.33 seconds, followed by ClusterMultiPathTool with a value of 1470 seconds, followed by BatchTool with a value of 2327.69 seconds. The highest error is for the SingleChamberTool equipment type mapped with an error of 3358.39. seconds.

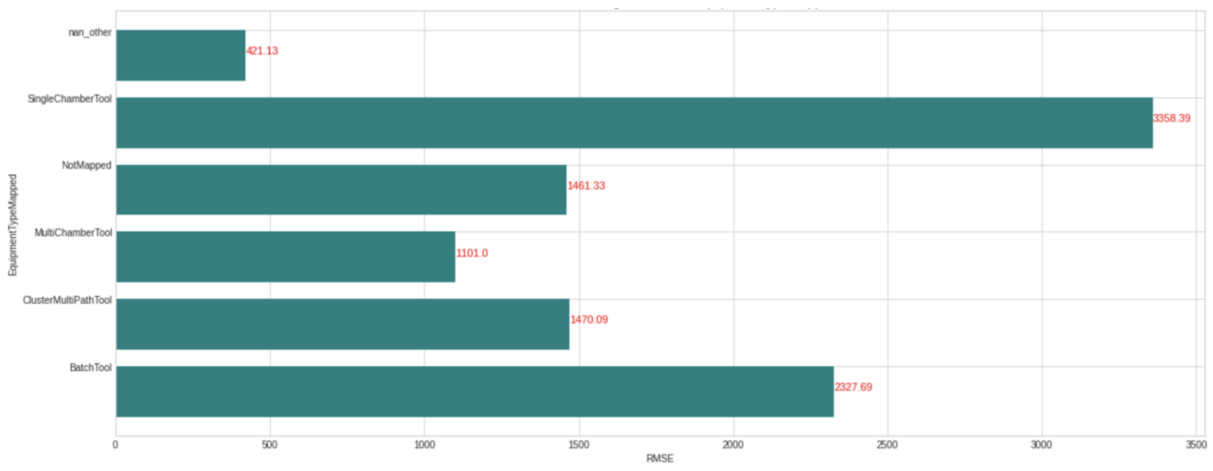


Figure 4.27: H2O GradientBoosting RMSE per EquipmentTypeMapped – Test data

Regarding the comparison of errors per OperationGroup, the following errors are shown as plotted in Figure 4.28. The lowest error is of MaskLayerOperation operation group with a value of 441.55 seconds, followed by the Developer operation group with a value of 788.14 seconds, followed by the Coater operation group with a value of 1035.85 seconds, followed by the PE-SiN-SiO operation group with a value of 1738.2 seconds, followed by the NotSpecified group with rmse of 2426 seconds. The highest error is of the Other_level operation group with a rmse of 2764.48. seconds.

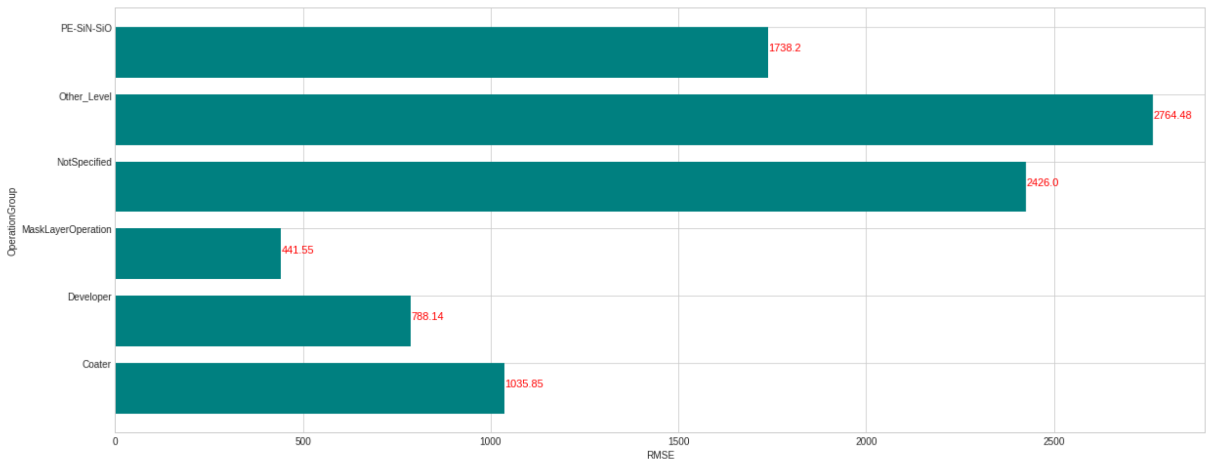


Figure 4.28: H2O GradientBoosting RMSE per OperationGroup – Validation data

Regarding the comparison of errors for the test data per OperationGroup, the following errors are shown as plotted in Figure 4.29. The lower error is of the MaskLayerOperation group similar to the validation data with a value of 271 seconds, followed by the Developer operation group with a value of 1100 seconds, followed by the Coater operation group with a value of 1326.54 seconds, followed by the Other_level operation group with a value of 1418.9 seconds. The highest errors are 1571.4 and 3126.88 for PE-SIN-SiO and NotSpecified, respectively.

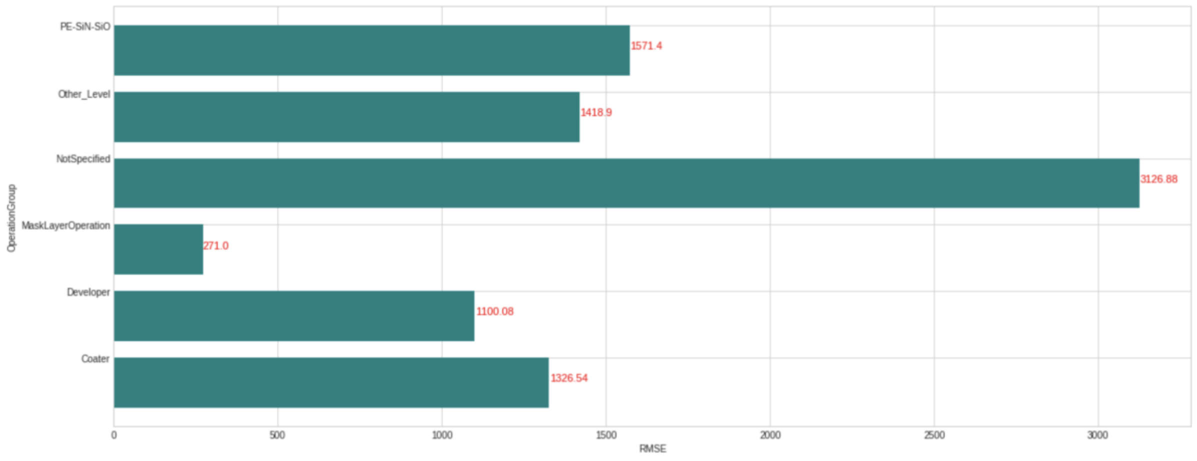


Figure 4.29: H2O GradientBoosting RMSE per OperationGroup – Test data

Comparing rmse values per technology for the validation data shows the below results in Figure 4.30. The lowest prediction error is Technology19's with 591.25 seconds, followed by a slightly higher error of Technology5's 627.2 seconds. The highest error is for Technology26 and Other_level, with errors of 3741.8 and 3871.13 seconds, respectively. The rest of the errors varied for the different technologies.

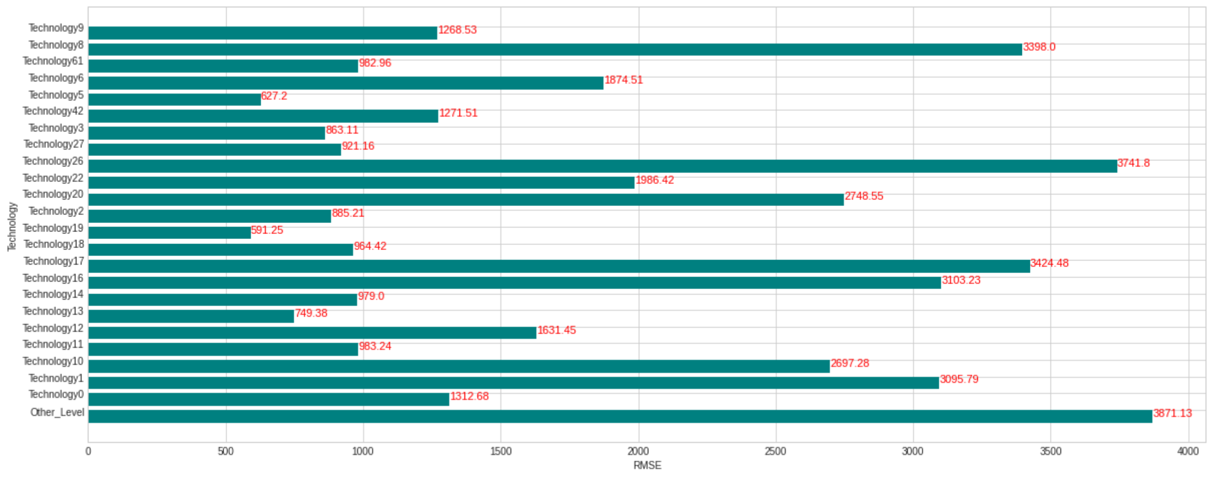


Figure 4.30: H2O GradientBoosting RMSE per Technology – Validation data

While companies rmse values per technology for the test data shows, the below results in Figure 4.31. The lowest prediction error is Technology9's with 546.83 seconds, followed by a slightly higher error of Technology26's 674.8 seconds. The highest error is for Technology12 and Technology8, with errors of 6062.87 and 7979 seconds, respectively. The rest of the errors varied for the different technologies.

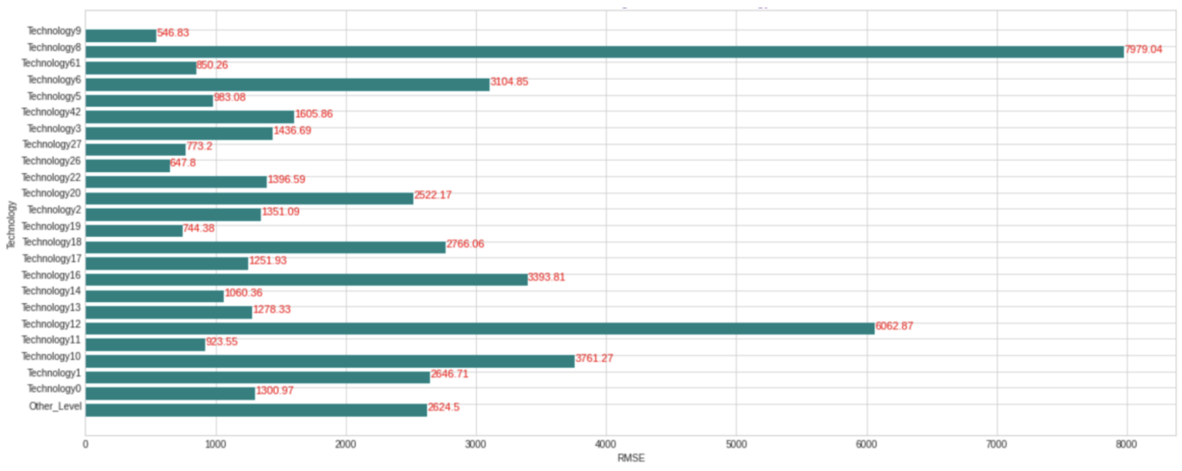


Figure 4.31: H2O GradientBoosting RMSE per Technology – Test data

Finally, for ProcessStep errors for the validation data, the following in observed as plotted in Figure 4.32. The lowest prediction error is 121.84 seconds for ProcessStep8, followed by a slightly higher error for ProcessStep82 with 159.22 seconds. The highest errors are of PorcessStep3 and Other_level with values of 1118.9 seconds and 2724.68 seconds, respectively. The remaining errors varied for the different process steps, with values ranging from 172.56 for ProcessStep9 to 804.85 seconds for ProcessStep145.

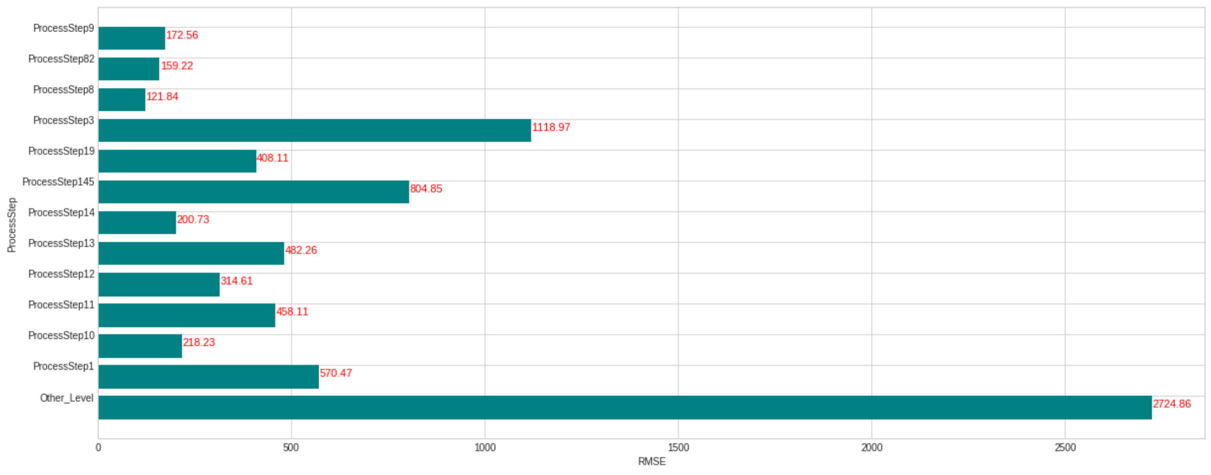


Figure 4.32: H2O GradientBoosting RMSE per ProcessStep – Validation data

The following ProcessStep errors for the test data are observed in Figure 4.33. The lowest prediction error is 87.16 seconds for ProcessStep8, similar to the validation data, followed by a slightly higher error for ProcessStep3 with 120.1 seconds. However, the highest errors are of PorcessStep1 and ProcessStep13 with 6645.94 seconds and 7635.74 seconds, respectively. The remaining errors varied for the different process steps, with values ranging from 144.16 for ProcessStep82 to 2699.64 seconds for Other_level process steps.

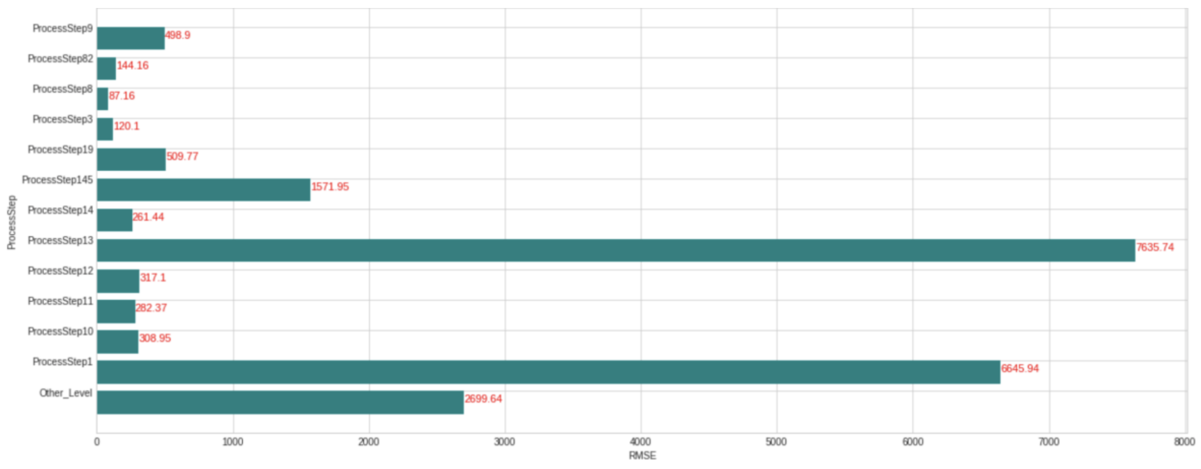


Figure 4.33: H2O GradientBoosting RMSE per ProcessStep – Test data

4.1.4 H2O Random Forests Model

Applying H2O GBM was one further step to view the equity of predicting tool recipe runtimes in semiconductor manufacturing. The rmse value achieved for H2O distributed random forests on the test data is 26439 seconds and 2178 seconds on the validation data.

When comparing the rmse values for each OperationGroup, the following results are shown for the validation in Figure 4.34. The lowest error is for the WPCM operation group, with an error of 748 seconds. For the rest of the groups, LITH equipment groups observed 909.78 seconds, followed by a higher error of 1159.3 seconds for the ETCH group, followed by the WPRO group with a slightly higher error of 1431.87 seconds, followed by the HCVD group with an error of 1656.73 seconds, followed. By IMET operation group with a value of 1677.78 seconds. The highest errors are for the groups' ICON and nan_other, with 2511.52 and 12145.26 seconds, respectively.

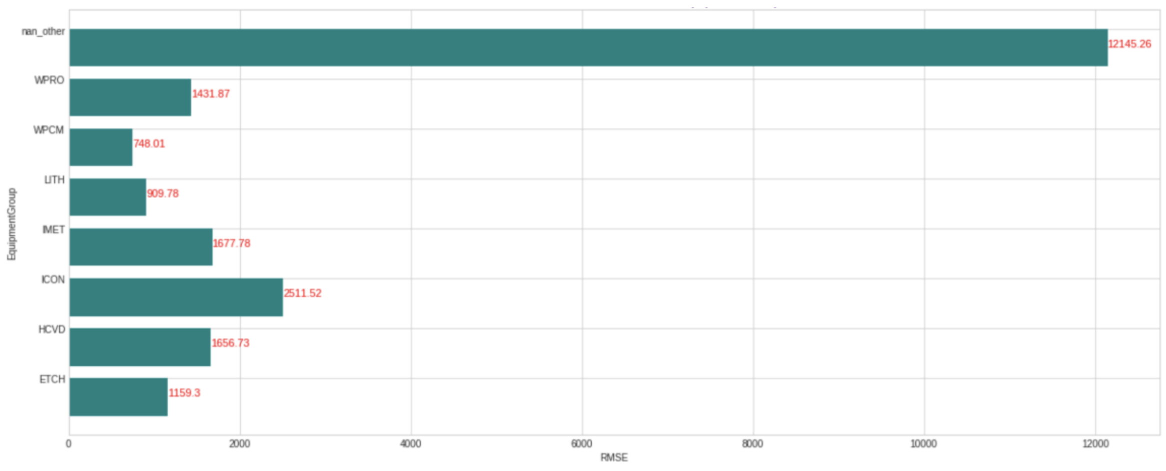


Figure 4.34: H2O RandomForest RMSE per Equipment Group – Validation data

While comparing the rmse values for each EquipmentGroup for the test data, the following results are shown in Figure 4.35. The lowest error is for the nan_other equipment group, with an error of 274.68 seconds. For the rest of the groups, LITH equipment groups demonstrate 875.32 seconds rmse, followed by a higher error of 972.77 seconds for the IMET group, followed by the WPRO group with a slightly higher error of 1431.87 seconds, followed by the ETCH group with an error of 1047.86 seconds, followed by HCVD equipment group with a value of 1655.39 seconds. Finally, the highest errors are for the groups WPRO and ICON with 2892.29 and 3690.55 seconds, respectively.

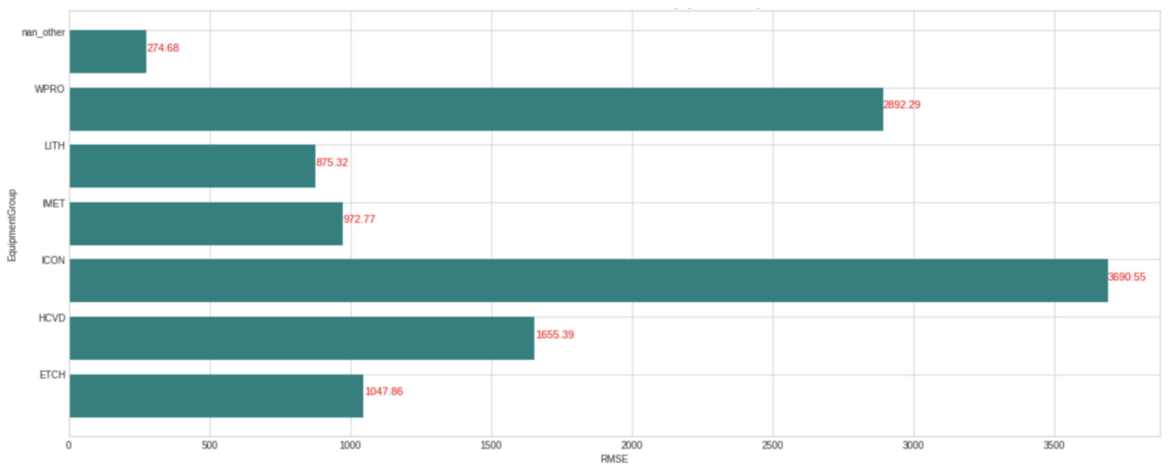


Figure 4.35: H2O RandomForest RMSE per Equipment Group – Test data

When analyzing the rmse values per EquipmentTypeMapped on the validation data, the following results in Figure 4.36 are shown. The lowest error observer is 280.59 for the nan_other equipment type. For the rest of the equipment types, the lowest error is 1034.37 seconds for the MultiChamberTool equipment type, followed by ClusterMultiPathTool's error of 1322.05 seconds, followed by BatchTool error of 1539.59 seconds, followed by SingleChamberTool with an error of 2027.73 seconds, and lastly by NotMapped group with an error of 6183.34 seconds.

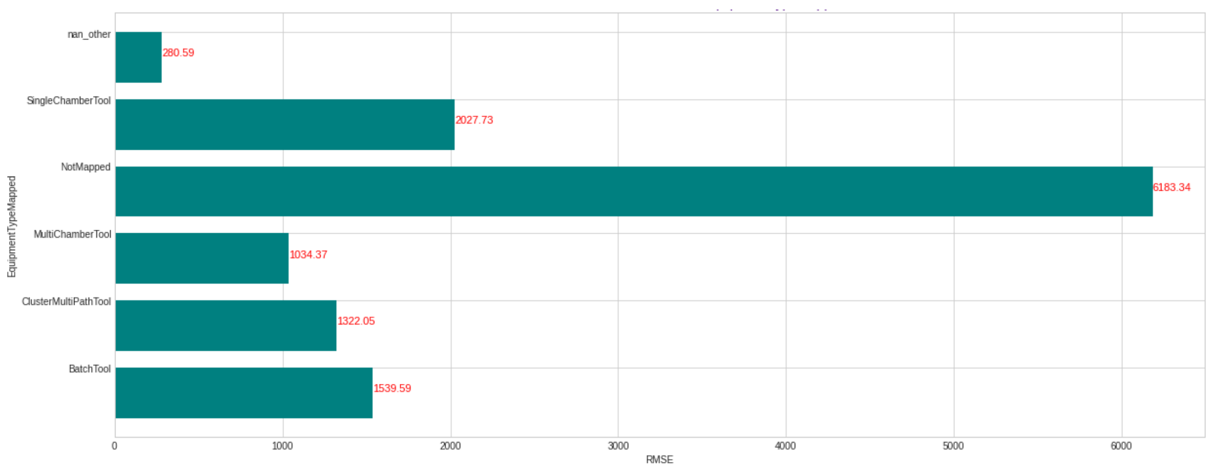


Figure 4.36: RMSE values for H2O Random Forests per EquipmentTypeMapped - Validation data

While when analyzing the rmse values per EquipmentTypeMapped on the test data, the following results are observed. The lowest error observer is 305.32 for the nan_other equipment type. For the rest of the equipment types, the lowest error is 1106.81 seconds for the MultiChamberTool equipment type, followed by

ClusterMultiPathTool's error of 1386.86 seconds, followed by NotMapped error of 1737.38 seconds, followed by BatchTool group with an error of 2317.46 seconds, and lastly by SingleChamberTool group with an error of 2926.91 seconds.

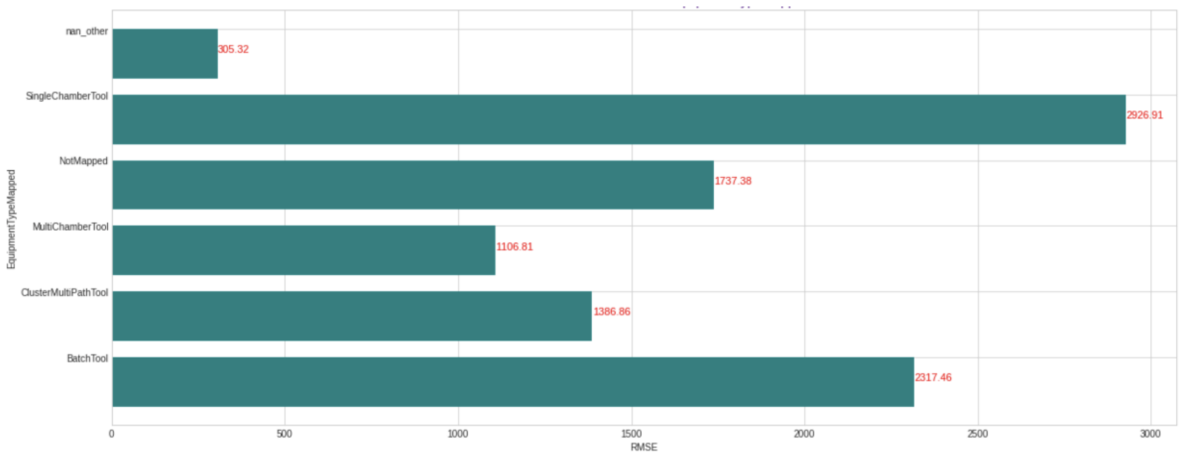


Figure 4.37: RMSE values for H2O Random Forests per EquipmentTypeMapped - Test data

Comparing the rmse values per OperationGroup shows the following errors for the validation data as observed in Figure 4.38. The lowest error is 485.32 seconds of the MaskLayerOperation group, 741.57 of the Developer group, and 866.5 seconds of the PE-SiN-SiO 1035.64 seconds of the Coater group, followed by 1737.54 of the NotSpecified group, lastly followed by 2723.56 of the Other_Level group.

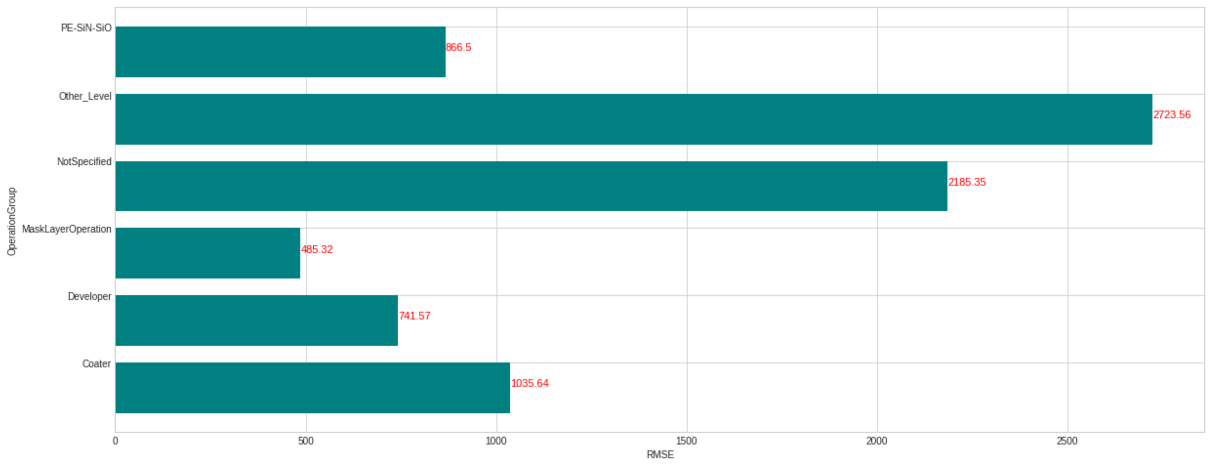


Figure 4.38: RMSE values for H2O Random Forests per OperationGroup - Validation data

While comparing the rmse values per OperationGroup shows the following errors for the test data as observed in Figure 4.39. The lowest error is 228.57 seconds of the MaskLayerOperation group, 872.66 of the PE-SiN-SiO group, and 1114.49 seconds of the Developer followed by 1326.71 seconds of the Coater group, followed by 1361.85 of the Other_level group, lastly followed by 2804.61 of the NotSpecified group.

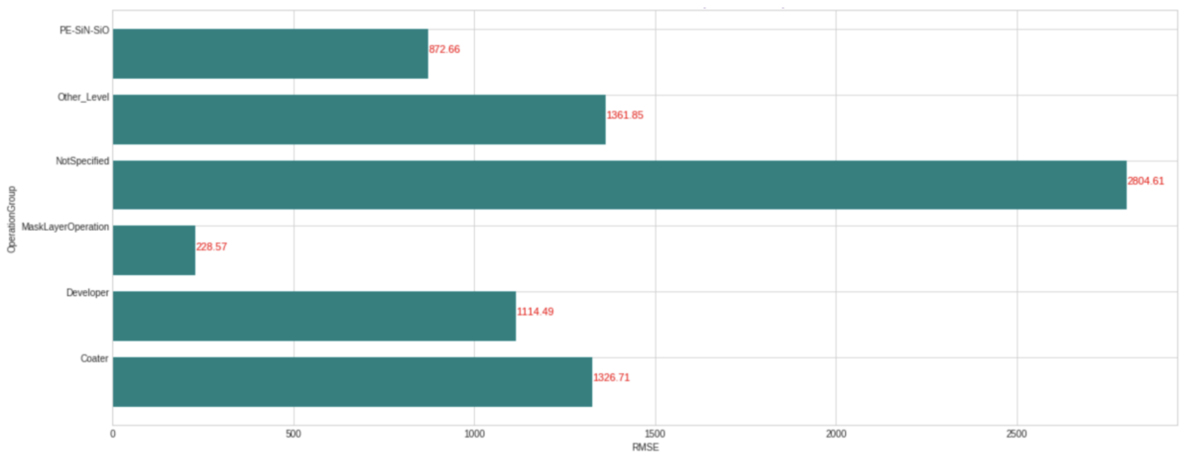


Figure 4.39: RMSE values for H2O Random Forests per OperationGroup - Test data

As for the technologies, the following errors for the validation data in Figure 4.40 are observed per Technology. The lowest error recorded is 507.98 seconds for Technology19, followed by a slightly higher 556.96 seconds for Technology61. The highest error is of 3789 seconds for Other_Level. The remaining errors varied from 687.73 for Technology13 to 3380.95 seconds for Technology8.

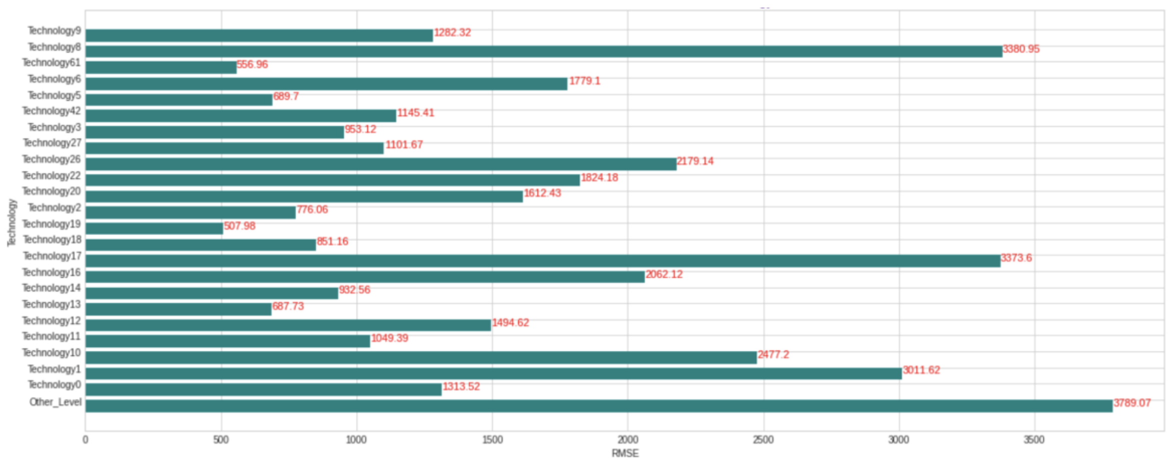


Figure 4.40: H2O Random Forests RMSE per Technology – Validation data

For the test data, the below errors in Figure 4.41 are observed per Technology. The lowest error recorded is 538.48 seconds for Technology9, followed by a slightly higher error of 627.85 seconds for Technology26. The highest error is of 6804.96 seconds for Technology8. The remaining errors varied from 733.96 for Technology27 to 6067.82 seconds for Technology12.

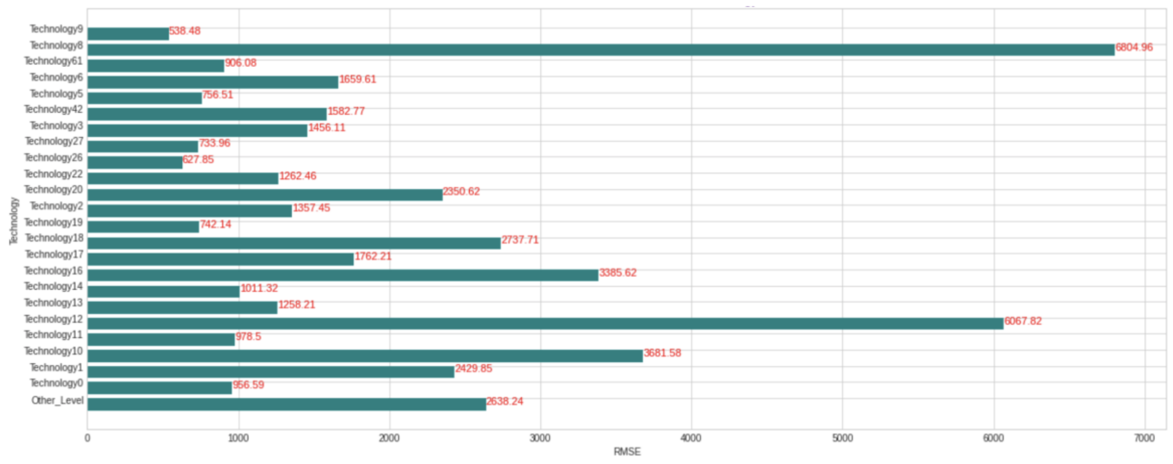


Figure 4.41: H2O Random Forests RMSE per Technology – Test data

Finally, the rmse values for ProcessStep on the validation data are observed in Figure 4.42. The lowest error is 69.07 seconds for ProcessStep8, followed by a slightly higher error of 125.2 seconds for ProcessStep82. The highest errors are of 1373.89 and 2461.76 seconds for ProcessStep3 and Other_Level, respectively. The remaining values varied from 157.72 seconds for ProcessStep9 to 788.51 seconds for ProcesStep145.

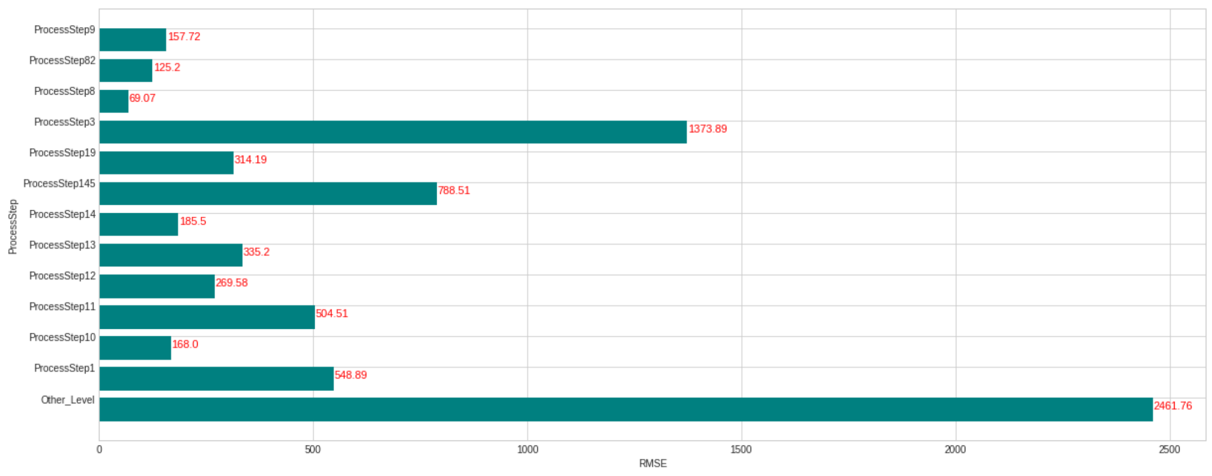


Figure 4.42: H2O Random Forests RMSE per ProcessStep – Validation data

For the errors observed on the test data, the below result in Figure 4.43 are recorded. The lowest error is 69.3 seconds for ProcessStep8, similarly to the validation data, followed by a slightly higher error of 84.41 seconds for ProcessStep82 with a similar rank as observed on the validation data. The highest errors are of 6648.8 and 7626.66

seconds for ProcessStep1 and ProcessStep13, respectively. The remaining values varied from 162.66 seconds for ProcessStep3 to 2225.86 seconds for Other_Level.

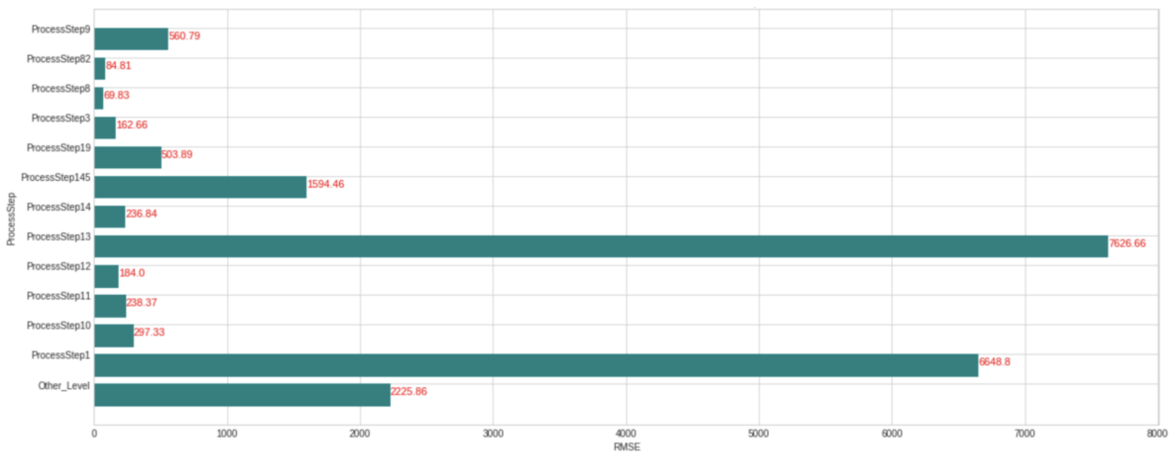


Figure 4.43: H2O Random Forests RMSE per ProcessStep – Test data

4.1.5 EquipmentGroup Specific Models

Applying Sklearn’s RandomForests algorithm to validate the ETCH EquipmentGroup specific model resulted in an RMSE of 1064.53 seconds. While when applying Sklearn’s GadientBoosting, a time of 974.82 seconds was recorded. To further analyze the errors across the different models. Specific models were built to compare their prediction errors with that of the baseline models. Such models are various EquipmentGroup specific models. In this subchapter, these models are reviewed.

Figure 4.44 and Figure 4.45 show the difference between the actuals and the ETCH EquipmentGroup specific model predictions. The predicted values are close to the predicted values for actuals of values up to 10000 seconds. From 10000 to 15000 seconds, Predicted values show a higher difference when compared to the actuals. The values higher than 15000 show significantly less accurate predicted values. Similar results are observed when Sklearn Gradient boosting machines for ETCH models, as observed in Figure 4.46.

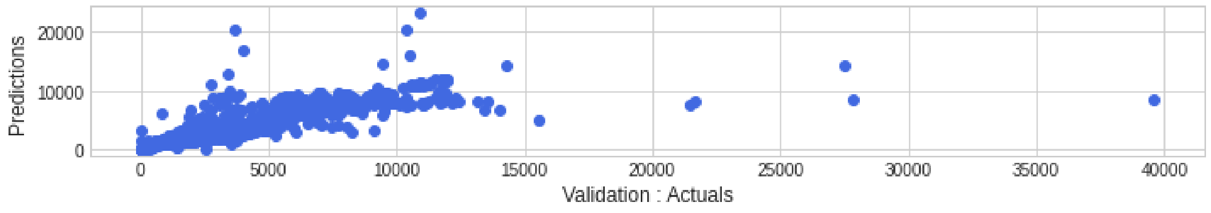


Figure 4.44: Actuals vs. Predictions, Sklearn RandomForest on ETCH specific model

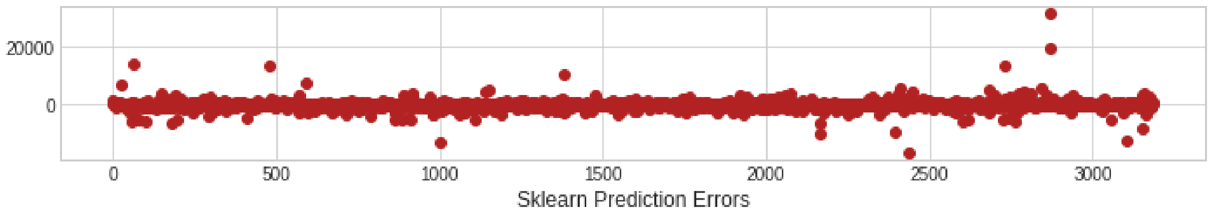


Figure 4.45: Prediction Errors, Sklearn RandomForest on ETCH specific model

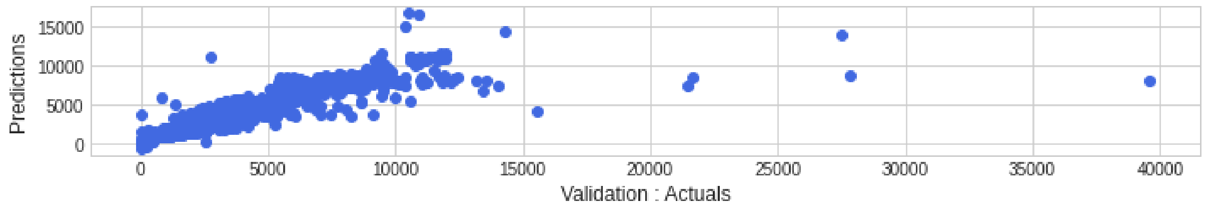


Figure 4.46: Actuals vs. Predictions, Sklearn GBM on ETCH specific model

The RMSE value for the H2O RandomForests model is 1036.89 and 1007.02 for the H2O GBM model. The results are similar with some improvement when building the H2O model, as shown in Figures 50 and 51.

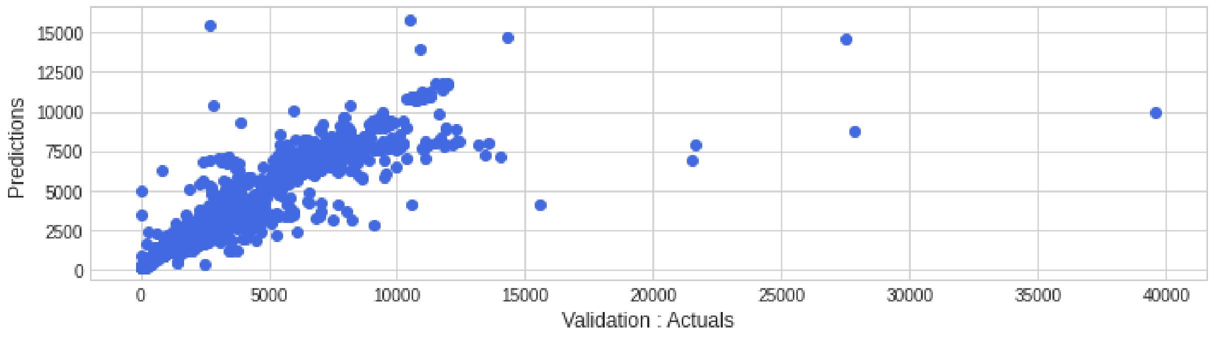


Figure 4.47: Actuals vs. Predictions, H2O RandomForest on ETCH specific model

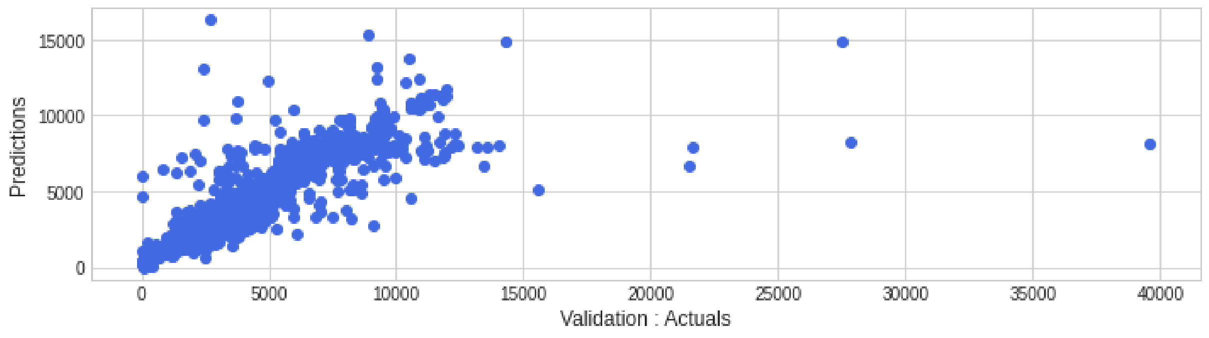


Figure 4.48: Actuals vs. Predictions, H2O GBM on ETCH specific model

As for the HCVD EquipmentGroup Model, applying Sklearn's RandomForests algorithm on the test data for the HCVD EquipmentGroup specific model resulted in an RMSE of 1953.51 seconds. While when using Sklearn's GadientBoosting, a rmse of 1812.76 seconds was recorded.

Figure 4.49 and Figure 4.50 show the difference between the actuals and the HCVD EquipmentGroup specific model predictions. The predicted values are close to the predicted values for actuals of values up to 25000 seconds. From 25000 to 35000 seconds, Predicted values show a higher difference when compared to the actuals. The values higher than 35000 show significantly less accurate predicted values. Similar results are observed when Sklearn Gradient boosting machines for HCVD models, as observed in Figure 4.51.

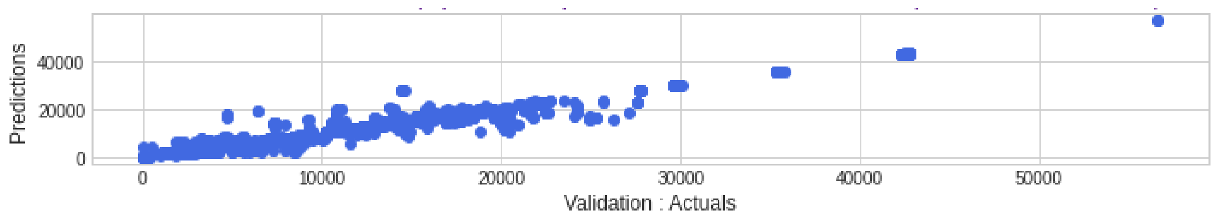


Figure 4.49, Actuals vs. Predictions, Sklearn RandomForest on HCVD specific model

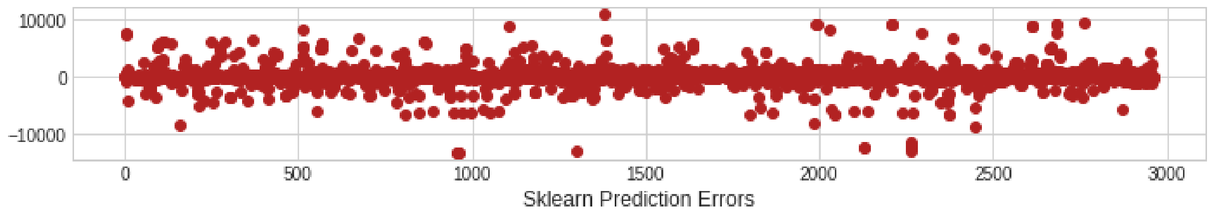


Figure 4.50: Prediction Errors, Sklearn RandomForest on HCVD specific model

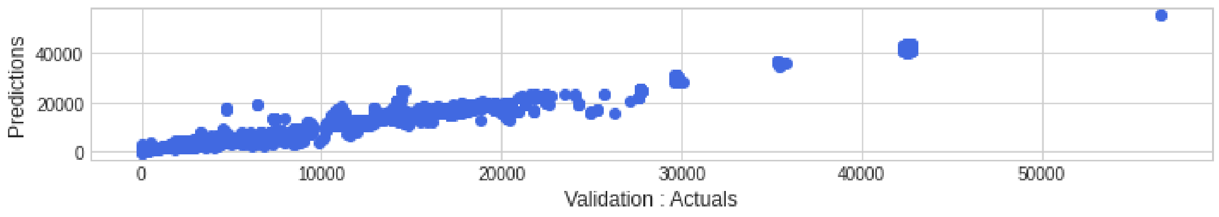


Figure 4.51: Actuals vs. Predictions, Sklearn GBM on HCVD specific model

The RMSE value for the H2O RandomForests model is 1694.01 and 1250.0 for the H2O GBM model. The results are similar with slight improvement when building H2O models, as Figure 4.52 and Figure 4.53.

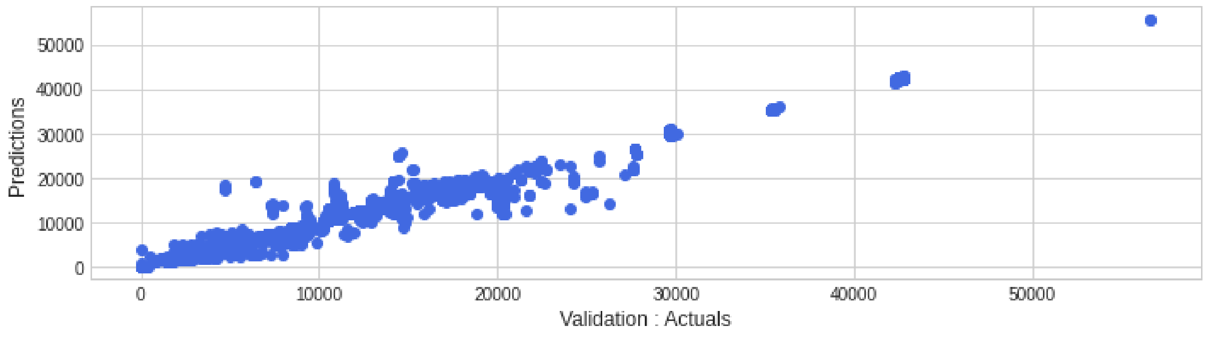


Figure 4.52: Actuals vs. Predictions, H2O RandomForest on HCVD specific model

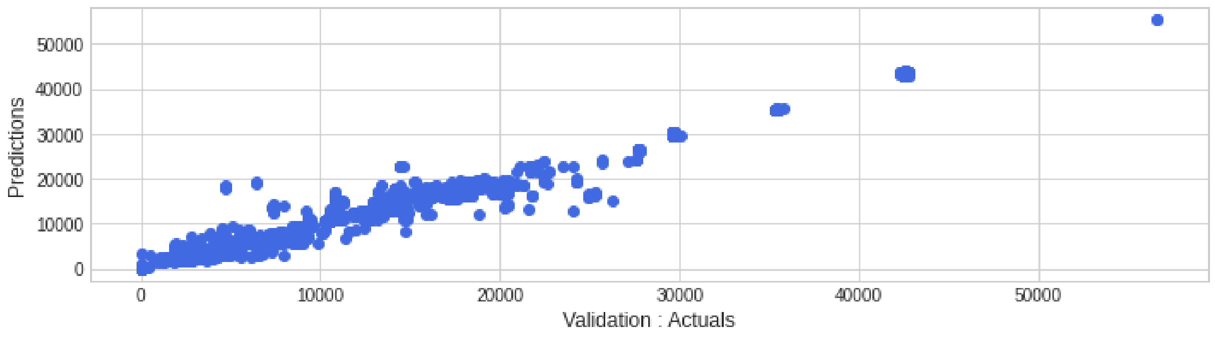


Figure 4.53: Actuals vs. Predictions, H2O GBM on HCVD specific model

For the ICON EquipmentGroup Model, Applying Sklearn's RandomForests algorithm on the test data for the ICON EquipmentGroup specific model resulted in an RMSE of 4625.2 seconds. While when using Sklearn's GadientBoosting, a rsme of 3991.47 seconds was recorded.

Figure 4.54 and Figure 4.55 show the difference between the actuals and the ICON EquipmentGroup specific model predictions. The predicted values are close to the predicted values for actual values up to 50000 seconds with few outliers. From 50000 to 90000 seconds, Predicted values show a higher difference when compared to the actuals. The values were higher than were falsy predicted. Similar results are observed when Sklearn Gradient boosting machines for ICON models, as observed in Figure 4.56.

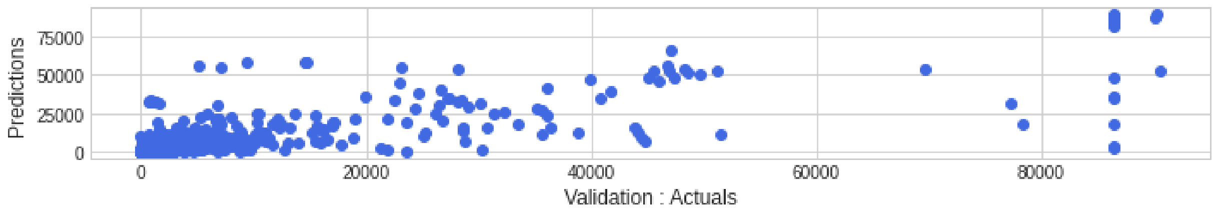


Figure 4.54: Actuals vs. Predictions, Sklearn RandomForest on ICON specific model

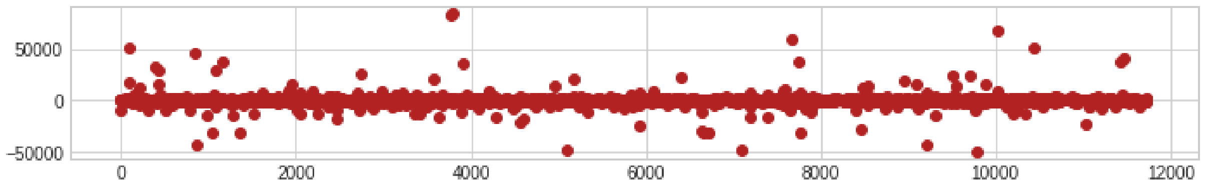


Figure 4.55: Prediction Errors, Sklearn RandomForest on ICON specific model

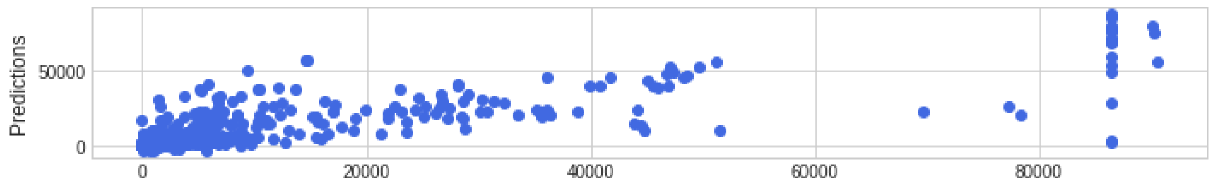


Figure 4.56: Actuals vs. Predictions, Sklearn GBM on ICON specific model

The RMSE value for the H2O RandomForests model is 3754.93and 3784.69 for the H2O GBM model. The results are similar, with some improvement when building H2O models, as shown in Figure 4.57 and Figure 4.58.

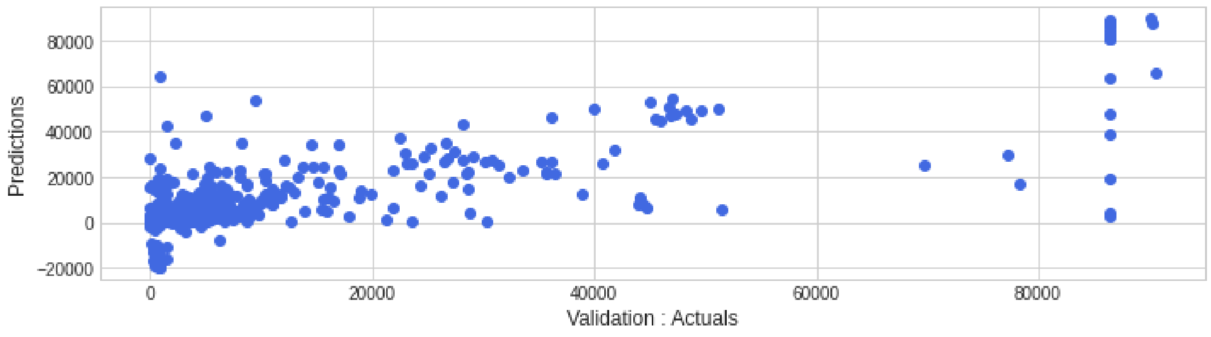


Figure 4.57: Actuals vs. Predictions, H2O RandomForest on ICON specific model

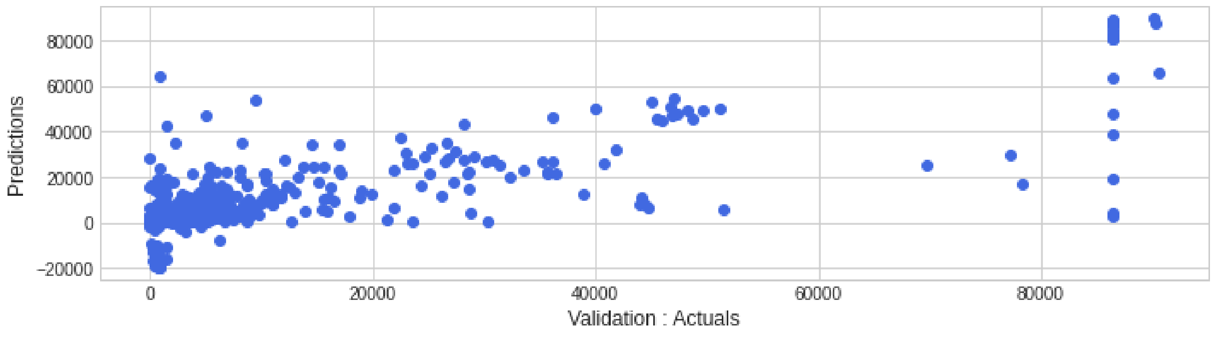


Figure 4.58: Actuals vs. Predictions, H2O GBM on ICON specific model

4.1.6 Process Recipe Specific Models

In order to further analyze the errors and understand if Recipe Runtime Prediction using ML-built data-driven models can predict specific recipes with high accuracies and fail to predict others, Process Recipe Specific Models were built. A Radom Forest model was built for each Process Recipe, and the prediction errors were analyzed for both the validation and test data. Figure 4.59 and Figure 4.60 plot these errors. Process Recipe 1 has a very high error of over 2500 seconds o both datasets. In contrast, the rest of the recipes, excluding ProcessRecipe4 and Other_level, had a much lower error of smaller than 500 seconds on both datasets.

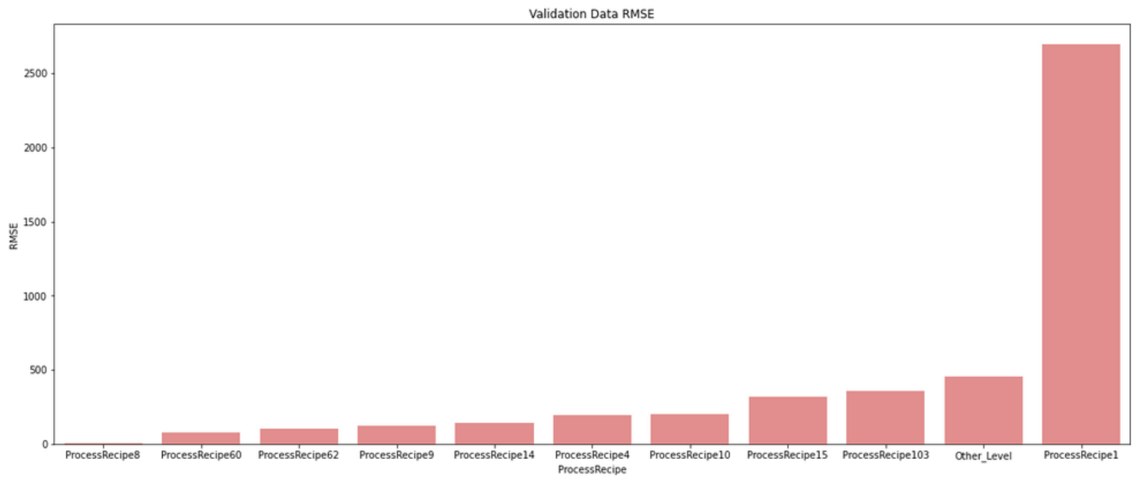


Figure 4.59: RMSE value per every ProcessRecipe model - Validation data

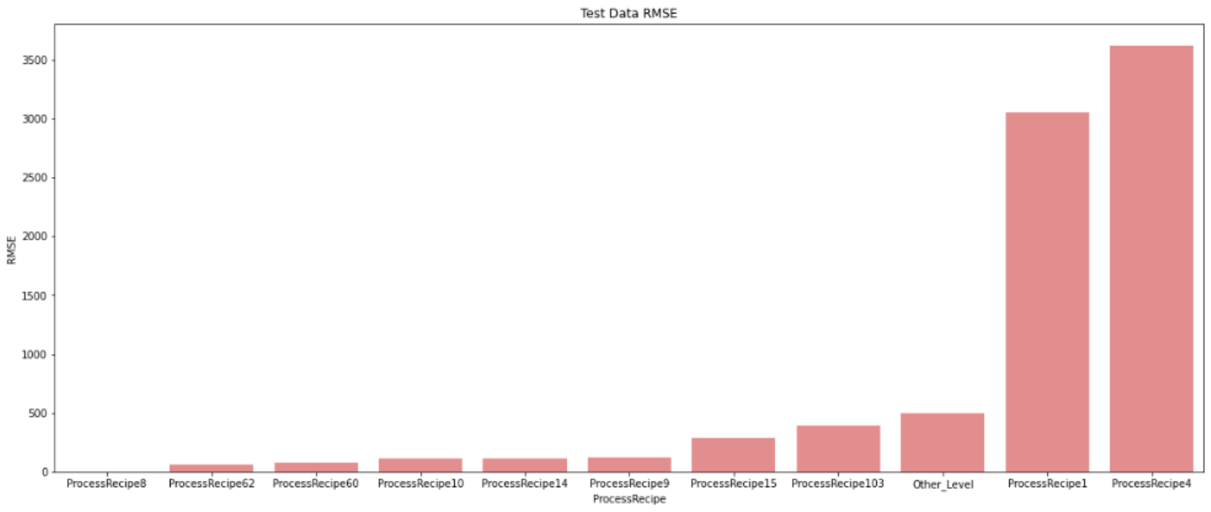


Figure 4.60: RMSE value per every ProcessRecipe model - Test data

To further investigate the possible reasons why ProcessRecipe1 had the highest error consistently, a sample of ProcessRecipe1 values were analyzed on both datasets, as shown in Figure 61. It can be observed that ProcessRecipe1 is not consistent over time in both the validation test dataset. On several occurrences, it has sudden high peaks in DurationSeconds. The frequent change in values caused the model to predict falsely with high RMSE. On the other hand, ProcessRecipe8 shows one of the lowest errors, and when a sample of the data was analyzed, as shown in Figure 4.62, such hikes in values in DuationSeconds did not exist.

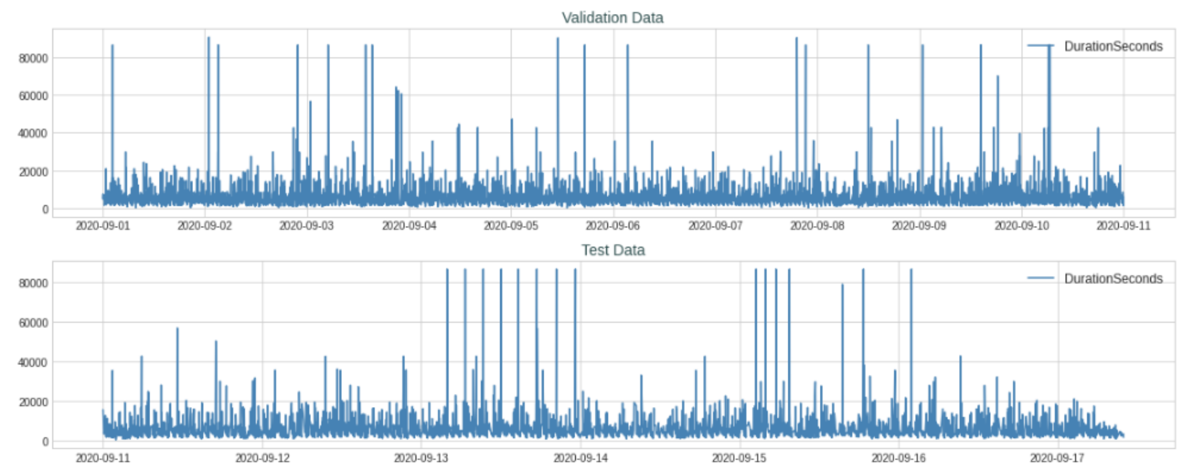


Figure 4.61: ProcessRecipe1 - DurationSeconds Analyzed

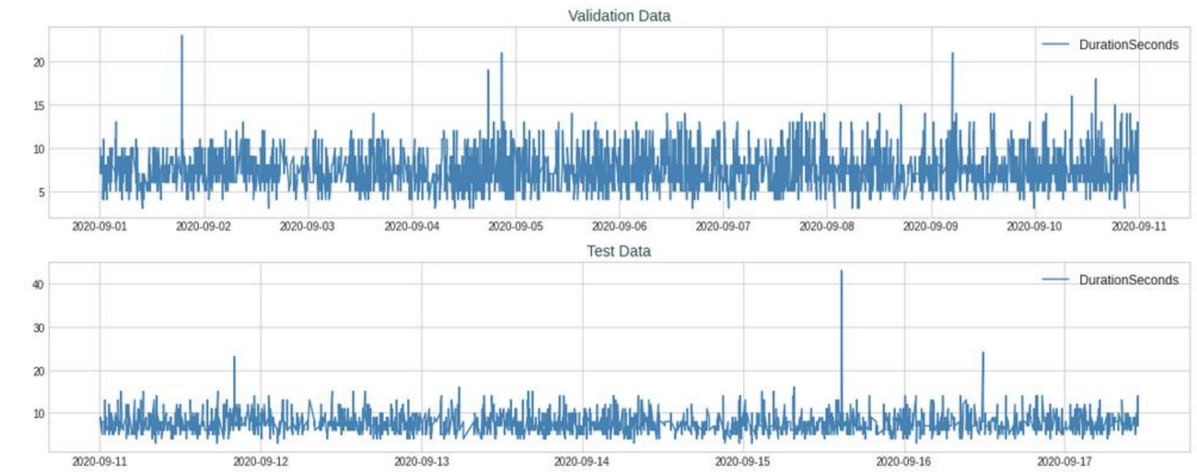


Figure 4.62: ProcessRecipe8 - DurationSeconds Analyzed

PorcessRecipe4 demonstrated similar behavior. Figure 4.63 demonstrates consistent values of Duration seconds around 400 seconds on the validation test. However, in the test dataset, there was a spike in DurationSeconds. Such steep changes in values drastically change model behavior.

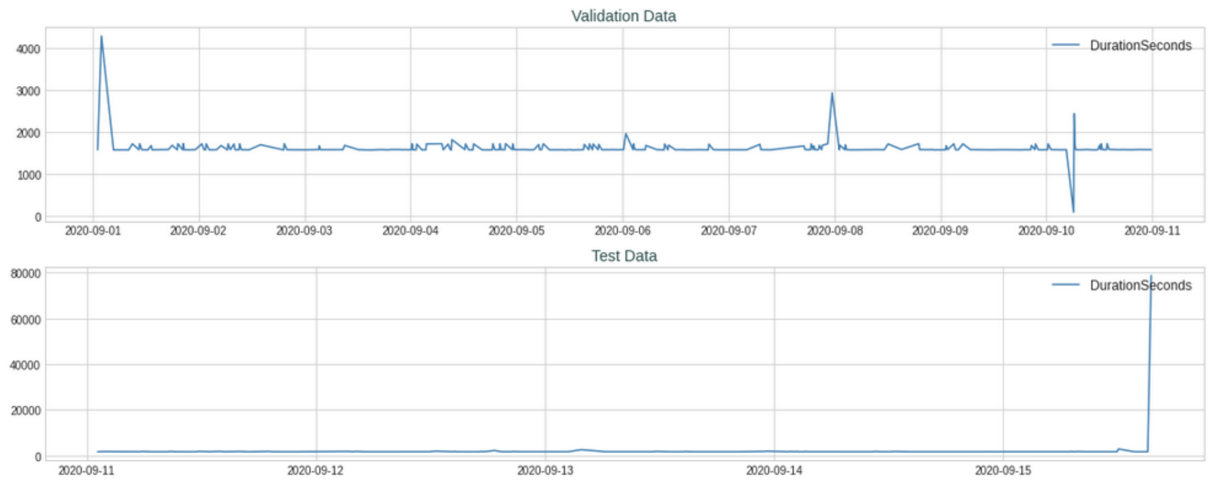


Figure 4.63: ProcessRecipe4 - DurationSeconds Analyzed across the Validation and Test datasets

4.1.7 H2O AutoML

H2O AutoML interface requires minimum parameters such as the label, the training frame, and, if required, stopping parameters. Values were kept to the default value in the implementations, not specifying a time limit to run AutoML. The output leaderboard is shown in below Table 2. The leaderboard shows models with their metrics. When provided with the H2OAutoML object, the leaderboard shows 5-fold cross-validated metrics by default. At most, 20 models are shown by default.

The lowest rmse value is 2056 seconds for the model StackedEnsemble_AllModels_AutoML_20210820_135536, followed by a slighly higher error of 2059.6 seconds for StackedEnsemble_BestOfFamily_AutoML_20210820_135536. The highest rmse values are 2213.7 and 2395.4 seconds for the models XGBoost_grid__1_AutoML_20210820_135536_model_1, and XRT_1_AutoML_20210820_135536 respectievely.

Table 2: H2O AutoML Leaderboard

model_id	mean_residual_deviance	rmse	mse	mae	rmsle	training_time_ms	predict_time_per_row_ms
StackedEnsemble_AllModels_AutoML_20210820_135536	4.23E+06	2056	4.23E+06	380.67	nan	1615	0.102582
StackedEnsemble_BestOfFamily_AutoML_20210820_135536	4.24E+06	2059.6	4.24E+06	381.43	nan	1438	0.096246

DeepLearning_grid__1_AutoML_20210820_135536_model_1	4.41E+06	2100.7	4.41E+06	439.49	nan	588990	0.100572
DRF_1_AutoML_20210820_135536	4.42E+06	2102	4.42E+06	342.03	0.50721	197363	0.006981
GBM_3_AutoML_20210820_135536	4.51E+06	2124.4	4.51E+06	413	nan	3947	0.003532
DeepLearning_1_AutoML_20210820_135536	4.54E+06	2130.9	4.54E+06	449.69	nan	20505	0.007489
GBM_4_AutoML_20210820_135536	4.54E+06	2131.7	4.54E+06	398.38	nan	6245	0.003757
GBM_2_AutoML_20210820_135536	4.55E+06	2132.5	4.55E+06	423.96	nan	3014	0.003411
XGBoost_2_AutoML_20210820_135536	4.56E+06	2136.1	4.56E+06	440.37	nan	55191	0.000813
XGBoost_grid__1_AutoML_20210820_135536_model_4	4.57E+06	2137.5	4.57E+06	498.56	nan	22250	0.000586
GBM_grid__1_AutoML_20210820_135536_model_2	4.58E+06	2139.9	4.58E+06	432.23	nan	27941	0.008769
XGBoost_grid__1_AutoML_20210820_135536_model_2	4.61E+06	2147.3	4.61E+06	443.31	nan	42300	0.000971
GBM_1_AutoML_20210820_135536	4.62E+06	2148.3	4.62E+06	425.59	nan	2374	0.003046
GBM_grid__1_AutoML_20210820_135536_model_1	4.64E+06	2154.6	4.64E+06	467.95	nan	1893	0.004474
XGBoost_3_AutoML_20210820_135536	4.67E+06	2160.5	4.67E+06	553.52	nan	16262	0.000459
XGBoost_grid__1_AutoML_20210820_135536_model_3	4.74E+06	2177.8	4.74E+06	573.36	nan	18059	0.00056
XGBoost_1_AutoML_20210820_135536	4.78E+06	2186.6	4.78E+06	502.12	nan	21688	0.000465
GBM_5_AutoML_20210820_135536	4.79E+06	2188.4	4.79E+06	424.46	nan	23012	0.01352

XGBoost_grid__1_AutoML_20210820_135536_model_1	4.90E+06	2213.7	4.90E+06	585.7	nan	19101	0.000589
XRT_1_AutoML_20210820_135536	5.74E+06	2395.4	5.74E+06	785.63	1.6854	39803	0.006308

The residual analysis below in Figure 4.64 plots the fittest values against the residual on the dataset. Ideally, the residuals should have a normal distribution. Most values are normally distributed. However, there exist several outliers.

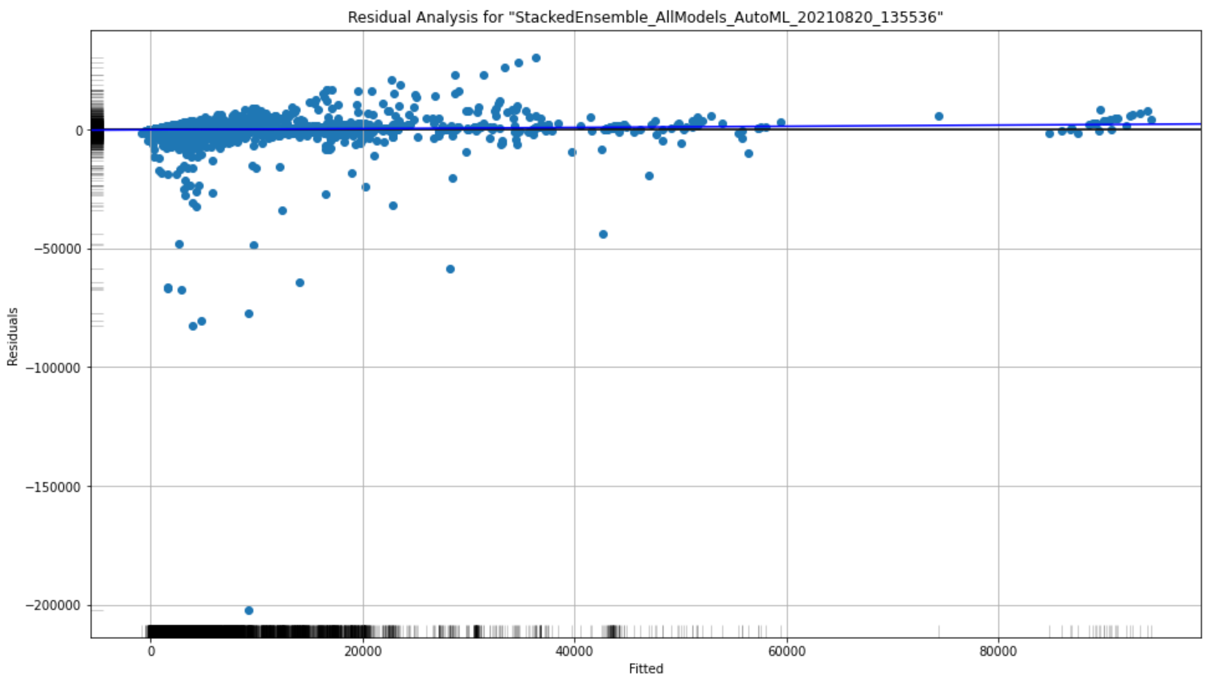


Figure 4.64: H2O AutoML Residual Analysis

As for the variable importance, the below Figure 4.65 show the relative importance of the most critical variables in the H2O AutoML model. On the top of the list is ProcessStep446 attributing to significant importance in prediction, followed by Area.LITH, ProcessRecipe 364, ProcessRecipe 480, Oprator1634, and several other variables.

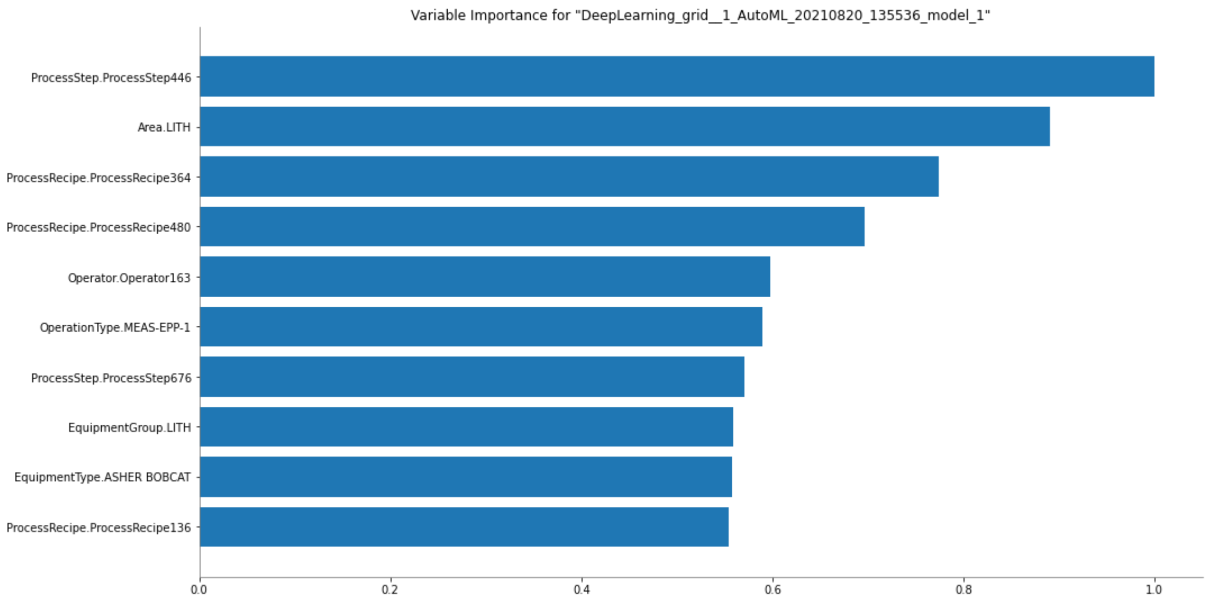


Figure 4.65: H2O AutoML Variable Importance

4.2 Model Comparison and Selection

The results from the applied baseline algorithms of gradient boosting and random forests from both libraries of sklearn and H2O provided similar results. However, no one model performs better than the rest, including the H2O AutoML. It is also unclear whether the EquipmentGroup specific models are a better choice compared to using baseline models.

Some Equipment Group-specific models, such as the ETCH Equipment Group model, provided very competitive results compared to others. At the same time, the ICON Equipment Group-specific model provided the highest error of prediction across all models. The top-performing model is the H2O AutoML model providing an adequate accuracy of prediction in comparison with the rest of the models.

Process Recipe Specific models provide a great indication of which Process Recipes a data-driven model would provide an adequate accuracy when predicting. Generally, the best performing models would be of recipes where the DurationSeconds were inconsistent limits and without many spikes in values.

4.3 Chapter Summary

Once the data was made ready and split into train, test, and validation datasets, the different ML algorithms were trained to build predictive models. Firstly, Gradient Boosting and Random Forests models were built using two distinctive libraries, and then the errors achieved were analyzed per specific predictors on both the validation and test datasets. A further step was building equipment-group-specific models and process recipe-specific models, and analyze their errors. AutoML was then applied to the test data, and the errors from the different models created were compared. Finally, all errors produced by the different models were compared for selection for deployment.

5 Operationalization

The advancement of machine learning involves addressing new issues that are not part of the traditional lifecycle of software development. While conventional software, for example, has a well-defined collection of product features to be created.

The growth of ML appears to revolve around experimentation: Machine Learning Developers will continuously experiment to optimize a business metric such as the model, new datasets, models, software libraries, tuning parameters, and more since the model output is heavily dependent on the data, and training processes. Reproducibility and experiment tracking are crucial in ML development [33].

Finally, ML systems need to be deployed to production to have a business effect, which means both are deploying a model in a way that can be used for inference, such as REST API Serving and deploying scheduled jobs to update the model periodically. This is particularly difficult when deployment involves cooperation with another team, such as application developers or factory users who are not ML experts. Several cloud vendors offer machine learning operations tools. However, with significant limitations to the usage of libraries from other sources, this limits organizations' capabilities to reach their applied machine learning goals, which is often composed of several libraries and is not necessarily cloud-deployed [33].

5.1 Model Management

MLflow offers APIs, functional in Python, Java, and R, for experiment monitoring, reproducible runs, and model packaging and deployment to facilitate life cycle management. Enhanced machine learning requires various tools, experiment tracking, reproducibility, production deployment, and collaborative model management. MLflow fulfills these requirements through its modules of MLflow Tracking, MLflow Projects, MLflow Models, and MLflow Models Registry [33], [34].

A model stored in MLflow, for instance, can simply be a Python function that MLflow then knows how to deploy in different environments such as batch or real-time processing. Other MLflow abstractions, such as REST APIs and Docker containers, are likewise based on generic interfaces. This open interface design allows users versatility and control while maintaining the advantages of lifecycle management compared to existing ML systems such as FBLearner, Michelangelo, and TFX [33].

5.2 Real-time Inference

The top-performing model shall be selected later for deployment through an ML lifecycle solution such as MLflow. The below Figure 5.1 shows the proposed architecture. The architecture involves converting the desired model's code into deployment at MLflow's interface, passing through different phases such as hyperparameter tuning, training, and storing the results. The proposed architecture would capture the production data, build models, evaluate models daily, keep track of training metrics, and deploy the selected model to predict the future points and serve use cases such as operator guidance.

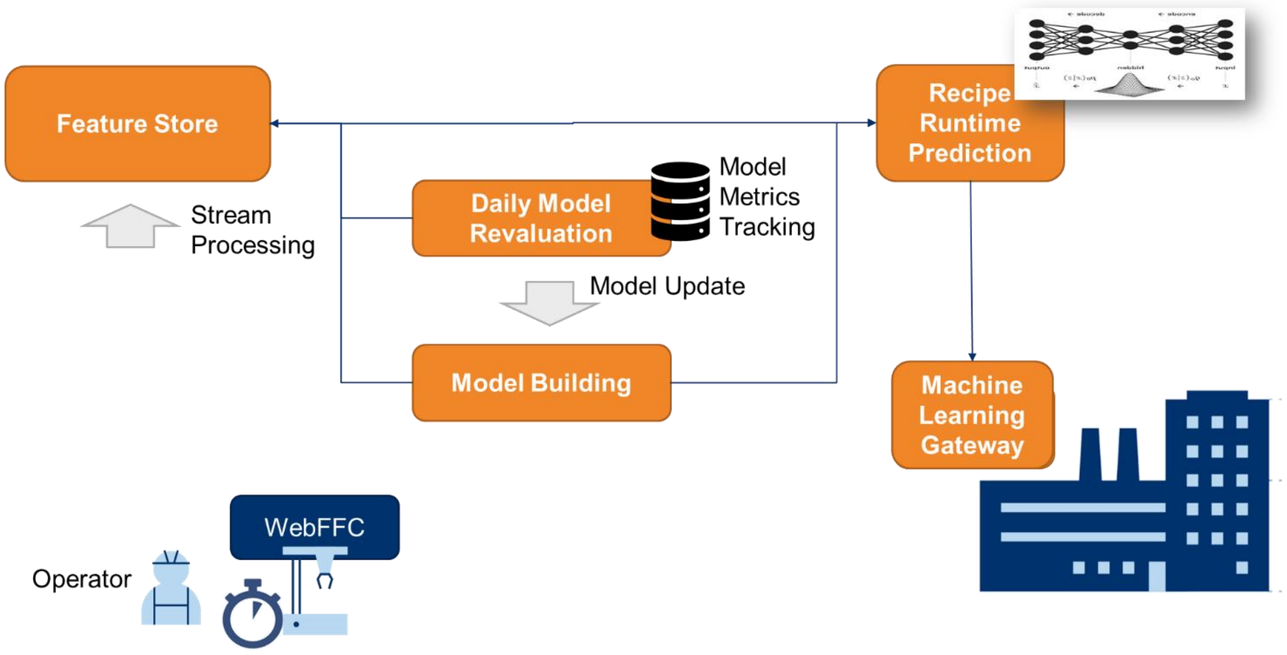


Figure 5.1: Proposed deployment architecture

5.3 Chapter Summary

Machine learning operations lifecycle platforms such as MLflow enable ML systems to be deployed to production. Through its modules, it also enables metrics tracking and management of models. Moreover, MLflow could be used to operationalize recipe runtime prediction to serve the proposed use cases.

6 Conclusion

Semiconductor manufacturing is a complex process that consists of a large number of substeps. There are different modes of wafer processing and job handling. Wafers are produced according to recipes. Recipe runtime is the time taken for each recipe to be completed. Predicting recipe runtime could serve different use cases, such as operator guidance. The proposed approach for predicting is by building ML models.

Predicting tool recipe runtimes in semiconductor manufacturing is possible by building machine learning models that leverage historical data to predict future data points. This approach initially requires understanding the semiconductor manufacturing processes and data and applying suitable machine learning algorithms, including AutoML, to build the data-driven predictive models.

The results achieved in this prediction approach can be considered high compared to the mean duration of recipes. However, due to the presence of outliers in data, the prediction process is affected. Perhaps, if the data contains as minimum outliers as possible, the prediction accuracy could be achieved. The error analysis shows that a particular tool can provide adequate accuracy with the current data predicting recipe runtimes in semiconductor manufacturing by training machine learning algorithms and building data-driven models. For other tools, machine learning prediction did not seem feasible due to the high error generated.

Achieving better prediction scores would lead to better prediction of tool recipe runtimes in semiconductor manufacturing, serving the potential use cases for such a system such as guiding the operators by the estimated duration remaining, planning within the industrial plant, or as insights for engineering, perspective to optimize the tool recipe runtimes.

Further Work

Further work to this thesis could build and analyze models that provide higher predictive accuracy than the analyzed models. First, recipe Specific models could be built for selected recipes that seem a good candidate for high accuracy prediction through the discussed methodology. Secondly, once an accepted prediction score is achieved, the potential model could be served and operationalized for deployment to serve the proposed use cases in the industry of semiconductor manufacturing.

Bibliography

[1] G. S. May and C. J. Spanos, *Fundamentals of Semiconductor Manufacturing and Process Control*. 2006.

[2] E. Bertino, B. Catania, and E. Caglio, "LNAI 1704 - Applying Data Mining Techniques to Wafer Manufacturing."

[3] Y. Meidan, B. Lerner, M. Hassoun, and G. Rabinowitz, *Data mining for cycle time key factor identification and prediction in semiconductor manufacturing*, vol. 42, no. 4 PART 1. IFAC, 2009.

[4] L. Lingitz, V. Gallina, F. Ansari, D. Gyulai, A. Pfeiffer, and W. Sihn, "Lead time prediction using machine learning algorithms: A case study by a semiconductor manufacturer," *Procedia CIRP*, vol. 72, pp. 1051–1056, 2018, doi: 10.1016/j.procir.2018.03.148.

[5] Y. Cheng, K. Chen, H. Sun, Y. Zhang, and F. Tao, "Data and knowledge mining with big data towards smart production," *J. Ind. Inf. Integr.*, vol. 9, pp. 1–13, Mar. 2018, doi: 10.1016/J.JII.2017.08.001.

[6] C. Rainer, "Data Mining as Technique to Generate Planning Rules for Manufacturing Control in a Complex Production System BT - Robust Manufacturing Control," 2013, pp. 203–214.

[7] T. Chen, Y. C. Wang, and H. R. Tsai, "Lot cycle time prediction in a ramping-up semiconductor manufacturing factory with a SOM-FBPN-ensemble approach with multiple buckets and partial normalization," *Int. J. Adv. Manuf. Technol.*, vol. 42, no. 11–12, pp. 1206–1216, 2009, doi: 10.1007/s00170-008-1665-4.

[8] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning second edition*, vol. №3, no. 1. 2018.

[9] D. Zhang and J. J. P. Tsai, "Machine learning and software engineering," *Softw. Qual. J.*, vol. 11, no. 2, pp. 87–119, 2003, doi: 10.1023/A:1023760326768.

[10] B. Hayes-roth, "Many complex steps, with few chances t o measure the process or the products. o u r architecture augments classical control methods with," no. December, 1991.

[11] H. A. Al-Jamimi and M. Ahmed, "Machine learning-based software quality prediction models: State of the art," *2013 Int. Conf. Inf. Sci. Appl. ICISA 2013*, pp. 13–16, 2013, doi: 10.1109/ICISA.2013.6579473.

[12] S. Sejdovic, Y. Hegenbarth, G. H. Ristow, and R. Schmidt, "Industry paper: Proactive disruption management system: How not to be surprised by upcoming situations," *DEBS 2016 - Proc. 10th ACM Int. Conf. Distrib. Event-Based Syst.*, pp. 281–288, 2016, doi: 10.1145/2933267.2933271.

[13] M. Belhaj Mohamed, A. Meddeb-Makhlouf, A. Fakhfakh, and O. Kanoun, "Wireless Body Sensor Networks with Enhanced Reliability by Data Aggregation Based on Machine Learning Algorithms," *Smart Sensors, Meas. Instrum.*, vol. 38, pp. 67–81, 2021, doi: 10.1007/978-3-030-71225-9_4.

[14] D. Koska and C. Maiwald, "A time-series based framework for exploring the unknown effects of shoe comfort-induced biomechanical adaptations," *https://doi.org/10.1080/19424280.2020.1734866*, vol. 12, no. 2, pp. 113–122, May 2020, doi: 10.1080/19424280.2020.1734866.

[15] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[16] U. Johansson, H. Boström, T. Löfström, and H. Linusson, “Regression conformal prediction with random forests,” *Mach. Learn.*, vol. 97, no. 1–2, pp. 155–176, 2014, doi: 10.1007/s10994-014-5453-0.

[17] H2O.ai, “Distributed Random Forest (DRF) — H2O 3.30.0.5 documentation,” 2020. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/drf.html.

[18] S. Shadi, S. Hadi, M. A. Nazari, and W. Hardt, “Outdoor navigation for visually impaired based on deep learning,” *CEUR Workshop Proc.*, vol. 2514, pp. 397–406, 2019.

[19] N. Asadi, J. Lin, and A. P. De Vries, “Runtime Optimizations for Tree-Based Machine Learning Models,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2281–2292, 2014, doi: 10.1109/TKDE.2013.73.

[20] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Front. Neurorobot.*, vol. 0, no. DEC, p. 21, 2013, doi: 10.3389/FNBOT.2013.00021.

[21] J. Hutter, Frank; Kotthoff, Lars; Vaschoren, *Automated Machine Learning Mathods, Systems, Challenges*, vol. 498. 2014.

[22] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, “Auto-Sklearn 2.0: The Next Generation,” *arXiv*, Jul. 2020, Accessed: Dec. 02, 2020. [Online]. Available: http://arxiv.org/abs/2007.04074.

[23] M. Feurer and F. Hutter, “Towards Further Automation in AutoML,” 2018.

[24] J. G. Madrid *et al.*, “Towards AutoML in the presence of Drift: first results,” 2018.

[25] Microsoft, “What is automated ML? AutoML - Azure Machine Learning | Microsoft Docs,” 2020. https://docs.microsoft.com/en-us/azure/machine-learning/concept-automated-ml.

[26] J. S. Angarita-Zapata, A. D. Masegosa, and I. Triguero, “General-purpose automated machine learning for transportation: a case study of auto-sklearn for traffic forecasting,” in *Communications in Computer and Information Science*, Jun. 2020, vol. 1238 CCIS, pp. 728–744, doi: 10.1007/978-3-030-50143-3_57.

[27] H2O.ai, “AutoML: Automatic Machine Learning,” *H2O.ai docs 3.26.0.11*, 2019. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/AutoML.html (accessed Aug. 24, 2021).

[28] B. Saenz De Ugarte, A. Artiba, and R. Pellerin, “Production Planning and Control Manufacturing execution system-a literature review Manufacturing execution system-a literature review,” *Prod. Plan. Control*, vol. 20, no. 6, pp. 525–539, 2009, doi: 10.1080/09537280902938613.

[29] J. T. Behrens, “Principles and Procedures of Exploratory Data Analysis,” *Psychol. Methods*, vol. 2, no. 2, pp. 131–160, 1997, doi: 10.1037/1082-989X.2.2.131.

[30] S. Morgenthaler, “Exploratory data analysis,” *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 1, no. 1, pp. 33–44, 2009, doi: 10.1002/wics.2.

[31] W. D. McGinnis, C. Siu, and H. Huang, “Category Encoders: a scikit-learn-contrib package of transformers for encoding categorical data Software • Review • Repository • Archive,” doi: 10.21105/joss.00501.

[32] “sklearn.ensemble.RandomForestRegressor — scikit-learn 0.24.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (accessed Aug. 23, 2021).

[33] M. Zaharia *et al.*, “Accelerating the Machine Learning Lifecycle with MLflow,” *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, pp. 39–45, 2018.

[34] A. Chen *et al.*, “Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle a Model Registry for collaborative model management and

review, tools for simplifying ML code instrumentation, and experiment analytics functions for extracting insights from mi,” 2020, doi: 10.1145/3399579.3399867.

TU Chemnitz References

[13] M. Belhaj Mohamed, A. Meddeb-Makhlouf, A. Fakhfakh, and O. Kanoun, “Wireless Body Sensor Networks with Enhanced Reliability by Data Aggregation Based on Machine Learning Algorithms,” *Smart Sensors, Meas. Instrum.*, vol. 38, pp. 67–81, 2021, doi: 10.1007/978-3-030-71225-9_4.

[14] D. Koska and C. Maiwald, “A time-series based framework for exploring the unknown effects of shoe comfort-induced biomechanical adaptations,” *https://doi.org/10.1080/19424280.2020.1734866*, vol. 12, no. 2, pp. 113–122, May 2020, doi: 10.1080/19424280.2020.1734866.

[18] S. Shadi, S. Hadi, M. A. Nazari, and W. Hardt, “Outdoor navigation for visually impaired based on deep learning,” *CEUR Workshop Proc.*, vol. 2514, pp. 397–406, 2019.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt
Selbstständigkeitserklärung

Name:	Sadek	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.
Vorname:	Karim	
geb. am:	15.07.1992	
Matr.-Nr.:	580632	

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 01.10.2021 Unterschrift: [illegible]

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.