The work was submitted to the

Institut für Textiltechnik der RWTH Aachen University

Univ.-Prof. Prof. h.c. (MGU)
Dr.-Ing. Dipl.-Wirt. Ing. Thomas Gries

Development of a methodology for implementing Predictive Maintenance

Presented as:	Bachelor Thesis
by:	Jeremy Theocharis
Matr.-No.	345607

1st examiner: Univ.-Prof. Prof. h.c. (MGU) Dr.-Ing. Dipl.-Wirt. Ing. Thomas Gries

2nd examiner: Dr.-Ing. Dieter Veit

Contact: Inga Gehrke, M.Sc.

Aachen, February 2019

Three steps for implementing Predictive Maintenance (PdM)

1. Find critical machines and components with random failures and high revenue loss due to breakdown
2. Evaluate the financial impact of PdM on critical components
3. Select, train and optimize PdM models for the highest financial impact

2018
Bachelor Thesis
Development of a methodology for implementing Predictive Maintenance
Jeremy Theocharis

Deutsche Kurzfassung

Abstract in German

Ziel der Arbeit: Ziel dieser Arbeit ist eine Methodik zu entwickeln, um Predictive Maintenance (Abkürzung: PdM, Englisch für "vorausschauende Wartung") in einem Unternehmen betriebswirtschaftlich sinnvoll zu implementieren. Die entwickelte Methodik wird anschließend im Digital Capability Center (DCC) Aachen, einer Lernfabrik zum Thema Industrie 4.0, validiert.

Lösungsweg: Wartungsstrategien und Algorithmen für maschinelles Lernen werden zusammen mit Optimierungsmethoden von Produktionslinien recherchiert. Das Wissen wird anschließend in eine Methodik zusammengefasst und im DCC Aachen an einer Produktionslinie validiert.

Zentrale Ergebnisse: Aufgrund von sehr hohen Kosten und Aufwand lohnt sich PdM nur an Maschinen und Bauteilen, bei denen sehr hohe Gewinnausfälle bei einem Produktionsausfall entstehen. Im DCC Aachen wird das Lager der Schärmaschine als für PdM geeignet identifiziert. Die Herangehensweise des maschinellen Lernens in Kombination mit den bereits existierenden Sensoren reicht jedoch nicht für eine betriebswirtschaftlich sinnvolle Implementierung aus.

Schlagwörter: Predictive Maintenance, vorausschauende Wartung, Wartungsstrategien, maschinelles Lernen

English Abstract

Objective of this thesis: The goal of this thesis is to develop a methodology to implement Predictive Maintenance (PdM) economically viable into a company. The methodology is then validated in the Digital Capability Center (DCC) Aachen.

Solution process: Maintenance strategies and machine learning algorithms are researched together with methods for optimizing productions lines. This knowledge is then summarized and validated in the DCC Aachen.

Key results: Because of high costs and effort PdM is only economically viable on machines and components with high revenue losses due to breakdown and where the failure is almost independent from uptime and wear. In the DCC Aachen the wind up bearing at the warping machine is identified as a component for a PdM implementation, but a combination of machine learning and existing sensors is not enough for a economically viable implementation.

Key word: Predictive Maintenance, maintenance strategies, machine learning

Table of contents

List of Figures

List of Tables

1 Introduction

Predictive Maintenance (PdM) is a maintenance type, in which a forecast about the remaining useful lifetime (RUL) of a machine component is derived from an analysis of historical data.

According to a market study by PwC among 200 executives from industrial companies in Germany the importance of Predictive Maintenance will increase till 2022 to over 38% (see Fig. 1.1). Nevertheless, companies are struggling with PdM according (see Fig. 1.2) as only 25% of the 74 experts in the market study by BearingPoint said, that they have implemented a single project or exploited the potential.

A successful PdM implementation does not only need a functioning algorithm but also needs to generate a measurable financial impact. This includes economically selecting machine and components for PdM as well as choosing and optimizing the algorithm to generate the least costs based on the lead time or unit costs of the component.

There is enough scientific literature available about machine learning algorithms, PdM techniques, investment theory and maintenance topics in general, but very few on the topic of implementing PdM while covering business and technical topics at the same time.

This bachelor thesis is a guide for companies to economically implement PdM and covers both business and technical topics. At first, the state of the art is explained including an overview of different maintenance strategies and types, different machine learning types and algorithms, and analysis of PdM in practice. Then the economically viable selection of critical machines and components is explained using bottleneck detection methods and Failure mode, effects and criticality analysis (FMECA). The third part of the guide is about selecting the type of PdM model, evaluating the financial impact and then creating, optimizing and automating the machine learning algorithm. The finished guide is then validated in the Digital Capability Center Aachen (DCC Aachen), an Industry 4.0 model factory. The focus in the validation chapter is more on the economically viable implementation than on the mathematical optimization of the algorithms.

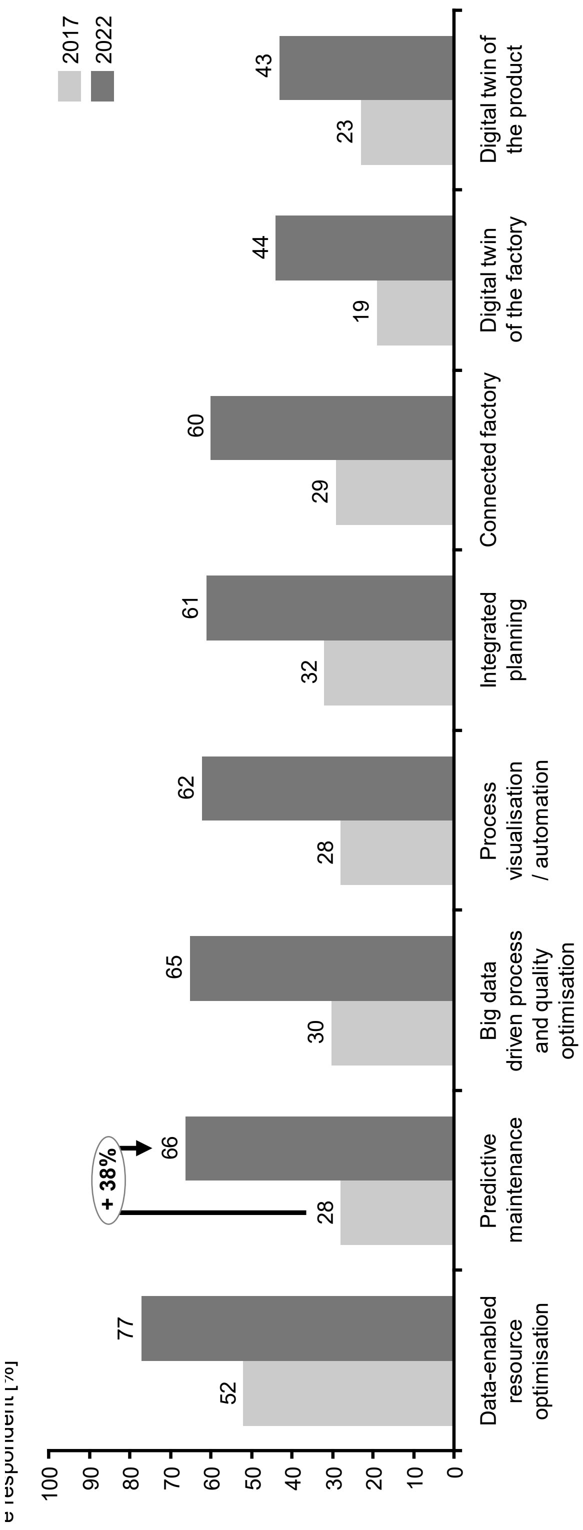


Fig. 1.1 Importance of PdM will increase over the next five years [PwC17]

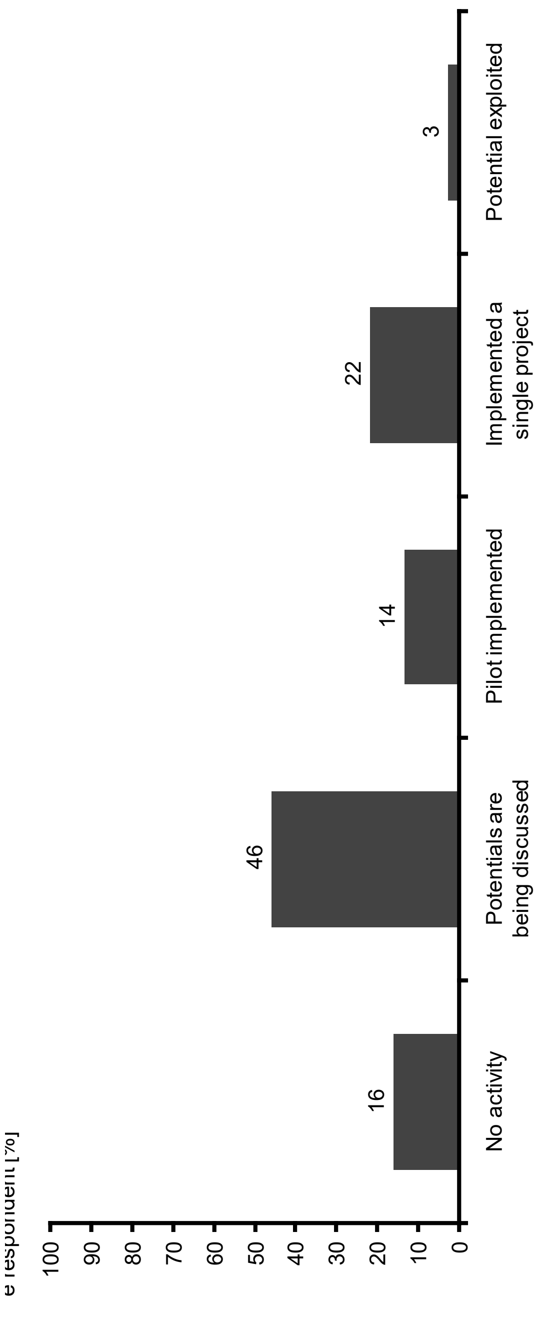

e respondent [%]
100
90
80
70
60
50
40
30
20
10
0
16
No activity
46
Potentials are being discussed
14
Pilot implemented
22
Implemented a single project
3
Potential exploited

Fig. 1.2 Companies are struggling with implementing PdM [DBG17]

2 State of the art

This chapter includes an overview of different maintenance strategies and types, different machine learning types and algorithms, and analysis of the use of PdM in practice

2.1 Maintenance strategies

Several maintenance strategies are practiced across different industries as shown in Tab. 2.1. This chapter will explain CBM and TPM. A-RCM will be left out as it is a short version of RCM and RCM is already explained in detail. CBM will be handled in detail in chapter 3.

Tab. 2.1 Maintenance strategies [DJ14]

Maintenance strategy	**Goal**
Condition-Based-Monitoring (CBM)	Detection of failure
Total Productive Maintenance (TPM)	Cultural change
Reliability-Centered Maintenance (RCM)	Failure prevention
Accelerated Reliability Centered Maintenance (A-RCM)	Failure prevention

2.1.1 Total Productive Maintenance (TPM)

The key objective of TPM is to eliminate or minimize sixteen losses impeding manufacturing performance and can be categorized as seen in Fig. 2.1. It is a more general approach to increase efficiency than a decision tool to choose the right maintenance type for each machine component.

Overall production effectiveness
- Equipment failure
- Set-up and adjustment loss
- Reduced speed loss
- Idling and minor stoppage loss
- Defect and rework loss
- Start-up loss
- Tool changeover loss

Equipment loading time
- Planned shutdown loss

Worker efficiency
- Distribution/logistic loss
- Measurement and adjustment loss
- Management loss
- Motion-related loss
- Line organization loss

Efficient use of production resources
- Yield loss
- Consumables (jig, tool, die) loss
- Energy loss

Fig. 2.1 The sixteen losses impeding manufacturing performance [AK08]

TPM recommends using "OEE […] as an indicator of the reliability of the production system.". OEE stands for Overall Equipment Efficiency and can be calculated by multiplying Availability, Performance efficiency and Rate of Quality Products as seen in Fig. 2.2. It is important to know that different OEE definitions are existing, especially in industry. In this definition, loading time is calculated by taking the full calendar time (24h) and subtracting time for planned maintenance before and then subtracting the idling time according to the production plan (e.g. if the machine is not operated during night shifts). Overall Plant Efficiency (OPE) is the name of the KPI that includes planned maintenance and idling time.

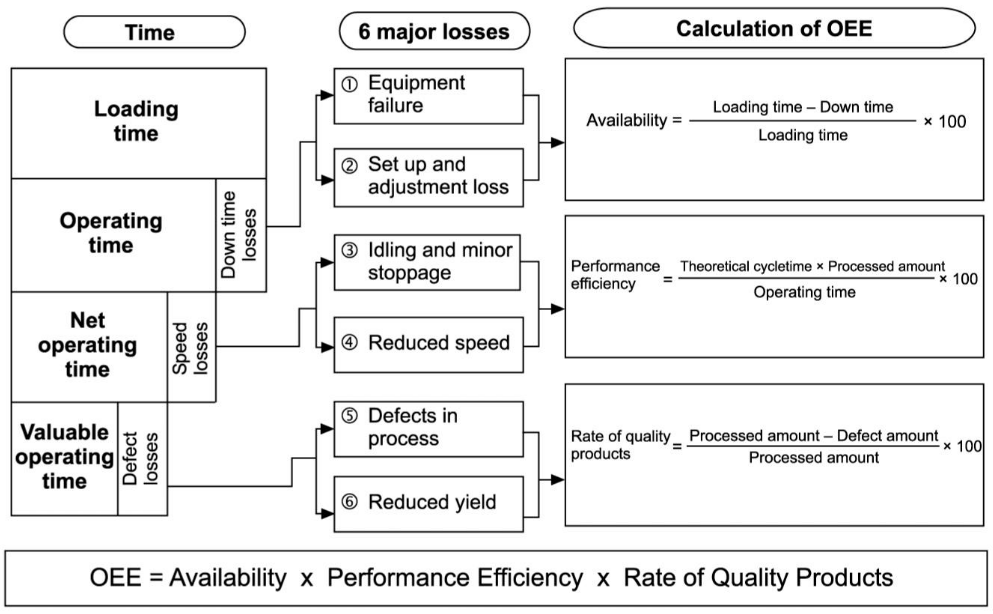

Time
6 major losses
Calculation of OEE
Loading time
Operating time
Down time losses
Net operating time
Speed losses
Valuable operating time
Defect losses
① Equipment failure
② Set up and adjustment loss
③ Idling and minor stoppage
④ Reduced speed
⑤ Defects in process
⑥ Reduced yield
Availability = (Loading time – Down time) / Loading time × 100
Performance efficiency = (Theoretical cycletime × Processed amount) / Operating time × 100
Rate of quality products = (Processed amount – Defect amount) / Processed amount × 100
OEE = Availability x Performance Efficiency x Rate of Quality Products

Fig. 2.2 Calculating the Overall Equipment Efficiency (OEE)

2.1.2 Reliability-Centered Maintenance (RCM)

The key objective of RCM is to "determine the maintenance requirements of any physical asset" by using seven steps [AK08]:

1. Selecting plant areas that are significant
2. Determining key functions and performance standards
3. Determining possible function failures
4. Determining likely failure modes and their effects
5. Selecting feasible and effective maintenance tactics
6. Scheduling and implementing selected tactics
7. Optimizing tactics and programs

Typical tools used in RCM and described in [AK08] are:

- Failure mode and effect analysis (FMEA)
- Failure mode effect and criticality analysis (FMECA)
- Physical Hazard Analysis (PHA)
- Fault Tree Analysis (FTA)
- Optimizing Maintenance Function (OMF)
- Hazard and Operability (HAZOP) Analysis

FMEA / FMECA will be handled in chapter 3 in detail. More information on the other tools can be found in [AK08]. RCM is often criticized for being "unreliable" [Mob02] and too "time-consuming" and "expensive" [Idh18].

2.2 Maintenance types

According to DIN EN 13306:2017 [DIN 13306] maintenance types can be divided into *corrective* and *preventive maintenance* and their sub-types as seen in Fig. 2.3.

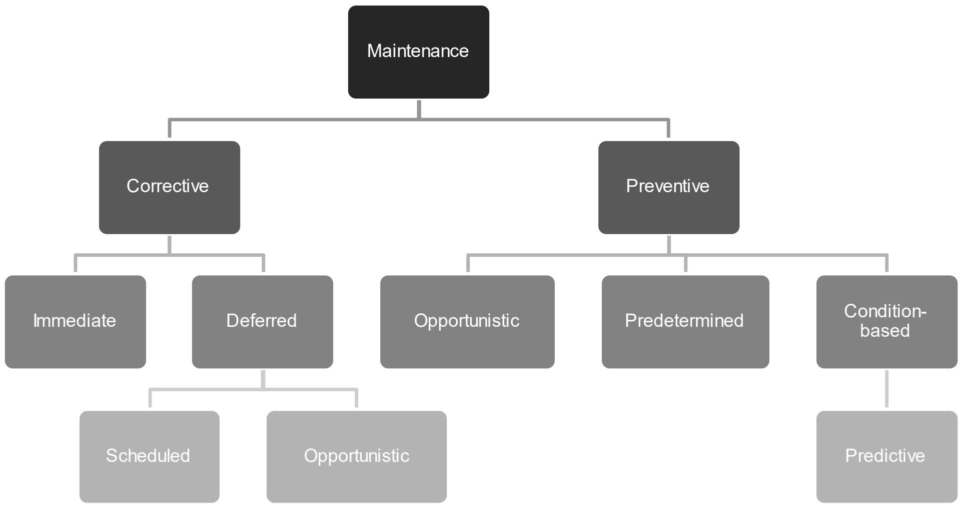


Fig. 2.3 Maintenance types as described in DIN EN 13306 [DIN 13306]

Corrective maintenance is carried out after a breakdown to restore the failed component back into a functioning state.

It is divided into

- *immediate maintenance*, which is carried out directly after a breakdown
- *deferred maintenance*, which is carried out not directly after a breakdown and can either be
    - o “carried out in accordance with a specified time schedule or specified number of units of use” (*scheduled maintenance*) or
    - o “at the same time as other maintenance actions or particular event to reduce costs, unavailability, etc.” (*opportunistic maintenance*).

Preventive maintenance is carried out before a breakdown and intended to reduce the probability of failure of a component.

It is divided into

- *Opportunistic maintenance* (see also deferred maintenance)
- *Predetermined maintenance*, which is regularly carried out based on intervals of either time or number of units
- *Condition-based maintenance*, which triggers maintenance action based on analysis of the physical condition of the component. One sub-type is
    - o *Predictive Maintenance*, in which a forecast about the remaining useful lifetime (RUL) of a machine component is derived from an analysis of historical data.

2.3 Machine learning algorithms

Machine learning is a “field of study that gives computer the ability to learn without being explicitly programmed" [Sim13]. It is used when “the application is too complex for people to manually design the algorithm” or “the application requires that the software customize to its operational environment after it is fielded” [Mit06].

Example applications besides Predictive Maintenance can be found in Fig. 2.4.

“Machine learning algorithms are organized into taxonomy, based on the desired outcome of the algorithm” [GBC16] as seen in Fig. 2.5. On the next pages, several supervised and unsupervised algorithms are explained in detail and then shown together with examples for a better understanding. The other categories of machine learning are not in focus of this bachelor thesis as they are less relevant to PdM than supervised and unsupervised algorithms. More details on them can be found in [Ayo10].

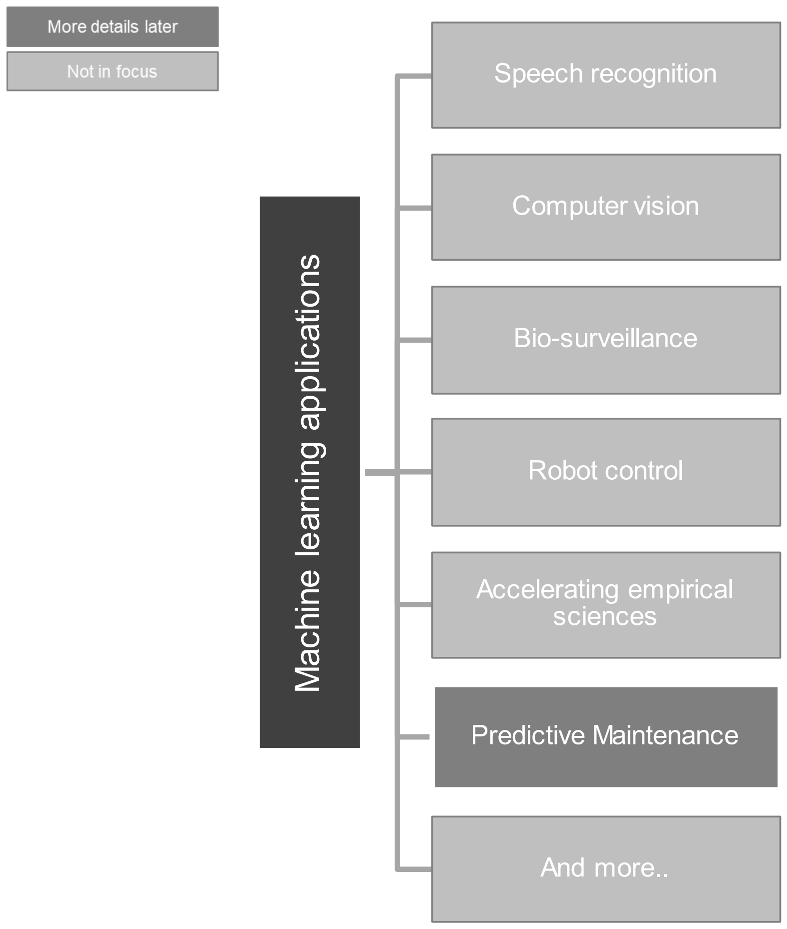


Fig. 2.4 Applications of machine learning [Mit06]

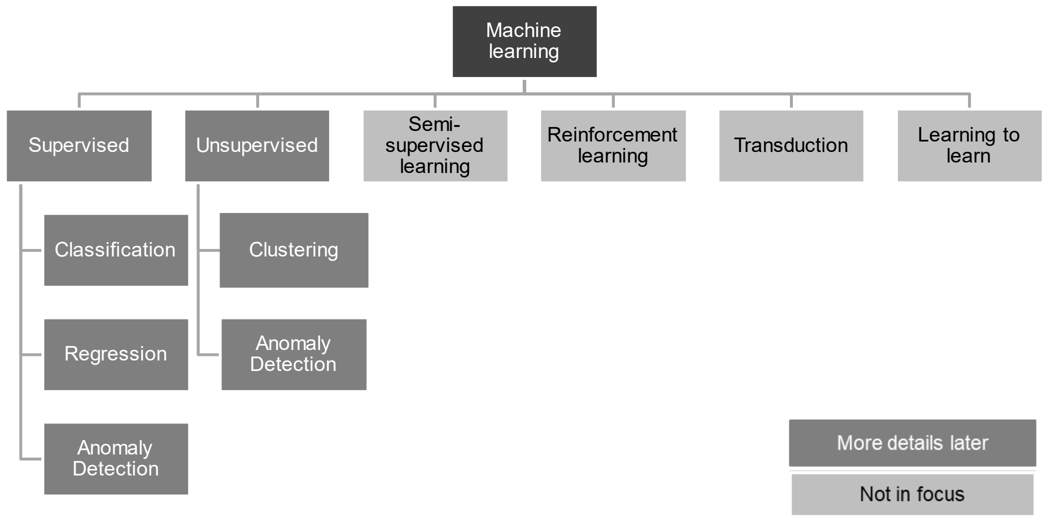


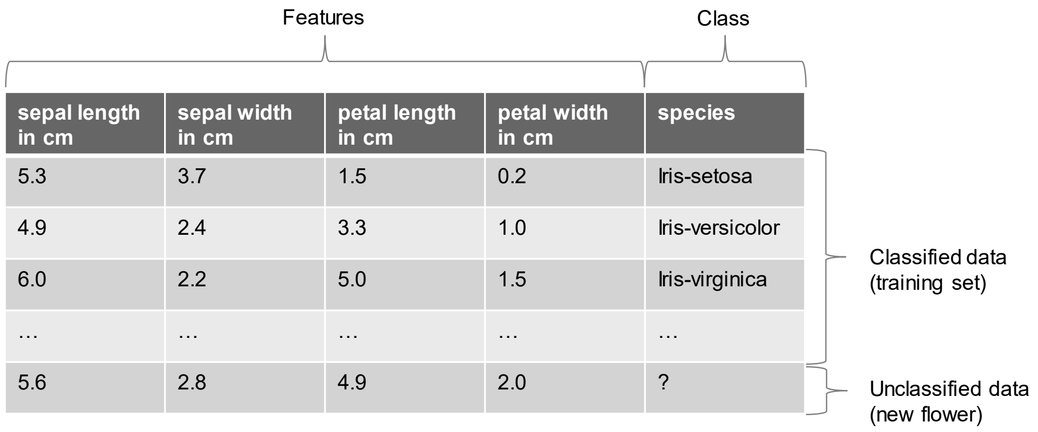
Fig. 2.5 Types of machine learning algorithms based on [CBK09; GBC16; Ayo10]

2.3.1 Supervised machine learning algorithms

„Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or target” [GBC16].

A classic example of a dataset suited for supervised learning can be seen in Fig. 2.6. In this dataset sepal and petal length and width (so-called features) have been measured for several iris species (so-called label or class). The historical and known data is called a training set. Now the sepal and petal length and width are measured for an unknown species (unclassified data) and the machine learning algorithm needs to estimate the species based on these features. As the features for each iris species scatter, which can be seen in the boxplots for the iris dataset in Fig. 2.7, a simple prediction is impossible.

In maintenance, the features would be for example machine data (energy consumption, speed, quality, etc.) and possible classes could be “machine okay” or “machine fails in under a week”.


Features
Class
Classified data (training set)
Unclassified data (new flower)

sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	species
5.3	3.7	1.5	0.2	Iris-setosa
4.9	2.4	3.3	1.0	Iris-versicolor
6.0	2.2	5.0	1.5	Iris-virginica
…	…	…	…	…
5.6	2.8	4.9	2.0	?

Fig. 2.6 The iris dataset (here: excerpt) is a classic example of a dataset suited for classification in supervised learning. [DK17]

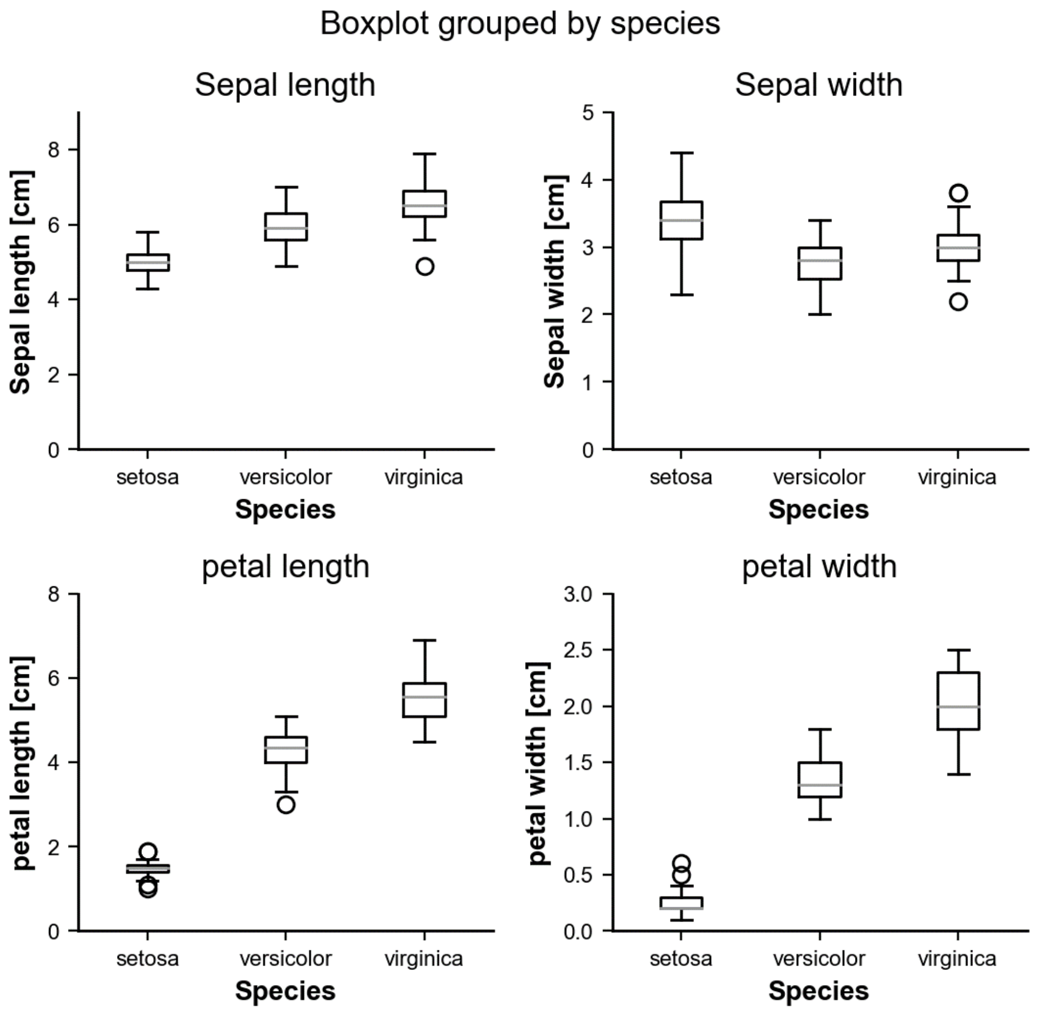
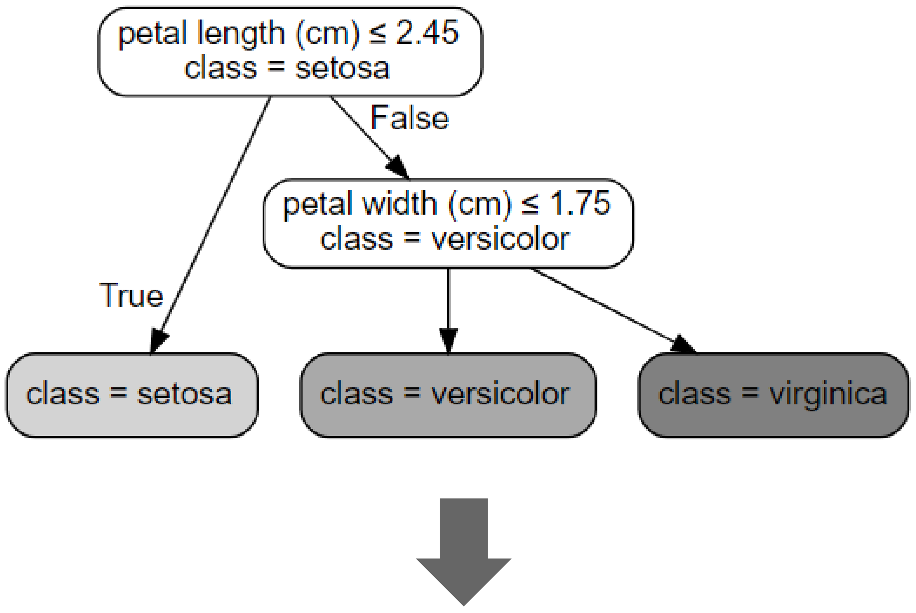


Fig. 2.7 The features sepal and petal length and width scatter for each iris species making a simple prediction impossible. Image generated with python [Ped11] based on the data from [DK17]

Because the label is discrete, the challenge is called **classification**. Algorithms suited for classification are for example decision trees, naïve Bayes or support vector machines [Mic17].

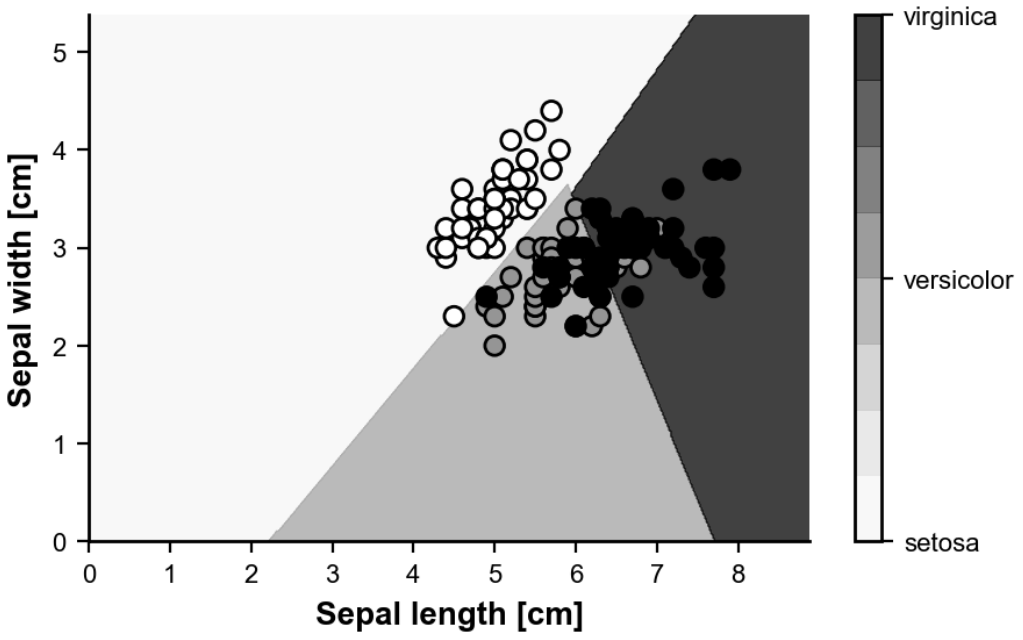
An example of the **decision tree** algorithm based on the iris dataset can be seen in Fig. 2.8. At first, a decision tree with a specified depth is created based on the training dataset. In every step, an expression for one feature is created, e.g. “petal length (cm) < 2.45”. Based on the result of that expression either the left or the right sub-tree gets processed. In the last rows, the final predicted class can be seen. This decision tree is then applied to the unclassified dataset and then resulting in the prediction of “Iris-virginica”.



sepal length in cm	**sepal width in cm**	**petal length in cm**	**petal width in cm**	**species**
5.3	3.7	1.5	0.2	Iris-setosa
4.9	2.4	3.3	1.0	Iris-versicolor
6.0	2.2	5.0	1.5	Iris-virginica
…	…	…	…	…
5.6	2.8	4.9	2.0	**Iris-virginica**

Fig. 2.8 Decision tree generated with python [Ped11] based on the dataset from [DK17] results in the prediction of „Iris-virginica“ for the unclassified dataset.

An example of the **support vector machines** algorithm based on the iris dataset can be seen in Fig. 2.9. For an easier understanding, the figure shows only two features and therefore a two-dimensional diagram. The full algorithm works in a four-dimensional space. The training data is shown as a colored scatter plot with the legend right to the image. The algorithm tries to create classification areas so that most of the points are in the correct area. Several methods to generate these areas are possible. Here a linear kernel is used, which means that the borders are straight (linear). Unclassified data is then plotted into the diagram and based on the location of the unclassified data in the diagram the classification is made.

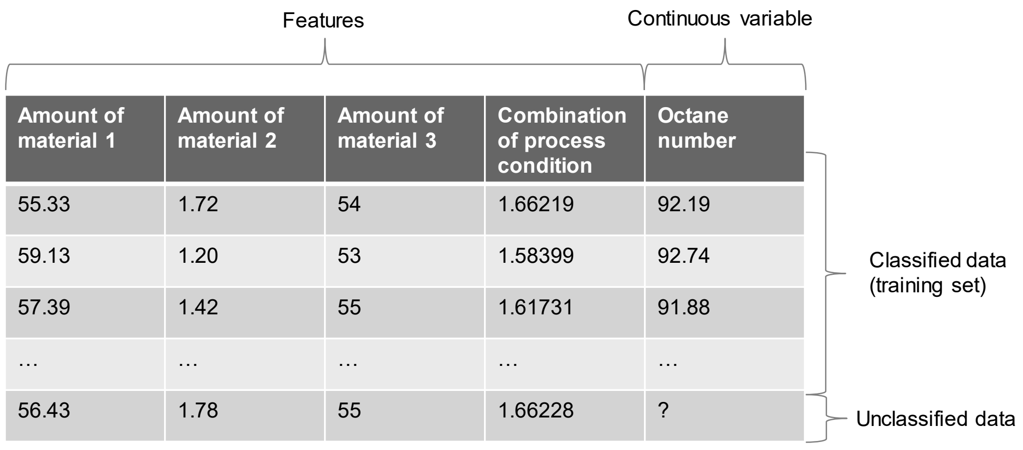


sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	species
5.3	3.7	1.5	0.2	Iris-setosa
4.9	2.4	3.3	1.0	Iris-versicolor
6.0	2.2	5.0	1.5	Iris-virginica
…	…	…	…	…
5.6	2.8	4.9	2.0	**Iris-virginica**

Fig. 2.9 Support vector machines algorithm generated with python based on an SVM with linear kernel [Ped11] with the dataset from [DK17]

On the other side, **regression** is used to predict a continuous variable [Zha10]. Example algorithms are linear regression, Poisson regression or decision trees [Mic17].

One example application is the prediction of the octane number of gasoline based on certain process conditions and the varying amount of three materials, as shown in Fig. 2.10. The regression line for linear regression can be seen in Fig. 2.11.


Features
Continuous variable
Classified data (training set)
Unclassified data

Amount of material 1	Amount of material 2	Amount of material 3	Combination of process condition	Octane number
55.33	1.72	54	1.66219	92.19
59.13	1.20	53	1.58399	92.74
57.39	1.42	55	1.61731	91.88
…	…	…	…	…
56.43	1.78	55	1.66228	?

Fig. 2.10 Example of a dataset suited for regression. Data from [Woo73]. The units are not mentioned in the source.

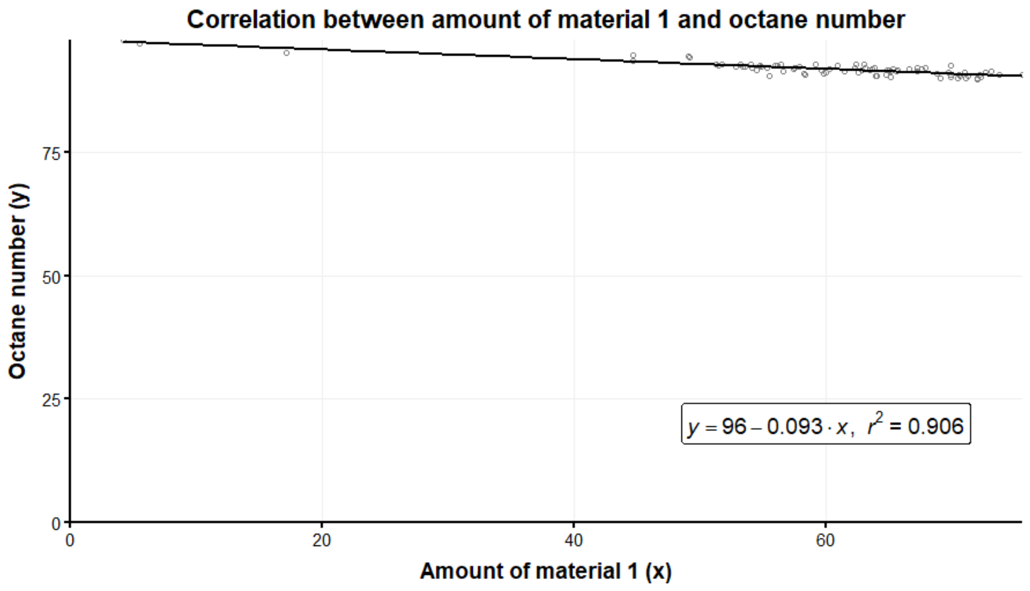


Fig. 2.11 Linear regression line for the dataset from [Woo73]. Image generated with R

2.3.2 Unsupervised machine learning algorithms

„Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset” [GBC16]. It can be divided into clustering and anomaly detection.

Clustering is used to organize and explore unknown data by splitting them into so-called clusters [JD88].

One example application can be found in Fig. 2.12. The dataset contains a list of orders during the year 2010 for an anonymous online retailer selling unique gifts. Using k-means the data can be clustered and needs to be interpreted afterwards. Cluster 2 (light grey on the right) shows, for example, big transactions from wholesalers and cluster 5 (black) transactions from end customers.

InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	Customer ID	Country
536365	85123A	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
536365	71053	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
…	…	…	…	…	…	…

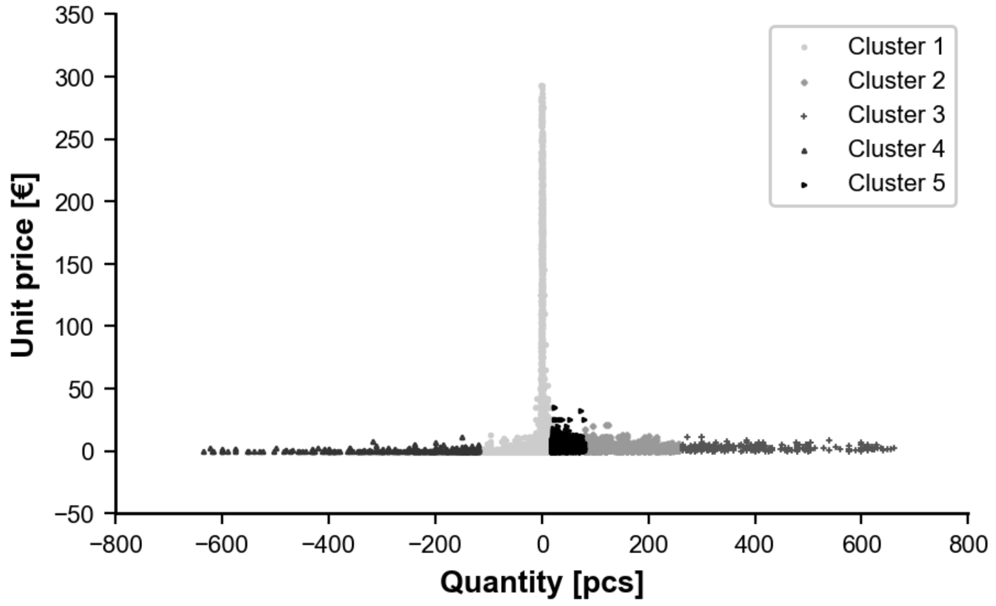


Fig. 2.12 Online retail example for clustering. Image generated with python (sci-kit-learn k-means clustering [Ped11]) with data from [CSG12]. Outliers were removed from the original data based on 95th and 5th percentile

The same dataset can be used to explain **anomaly detection**, which “refers to the problem of finding patterns in data that do not conform to expected behavior” [CBK09]. Example algorithms are statistical profiling using histograms, neural networks or support vector machines.

The online retail dataset without outliers can be seen in Fig. 2.13 on top. By creating a histogram, outliers can be found (Fig. 2.13 bottom). One possible interpretation for the outliers of quantity is typing errors during the processing of the transaction as they have been reversed.

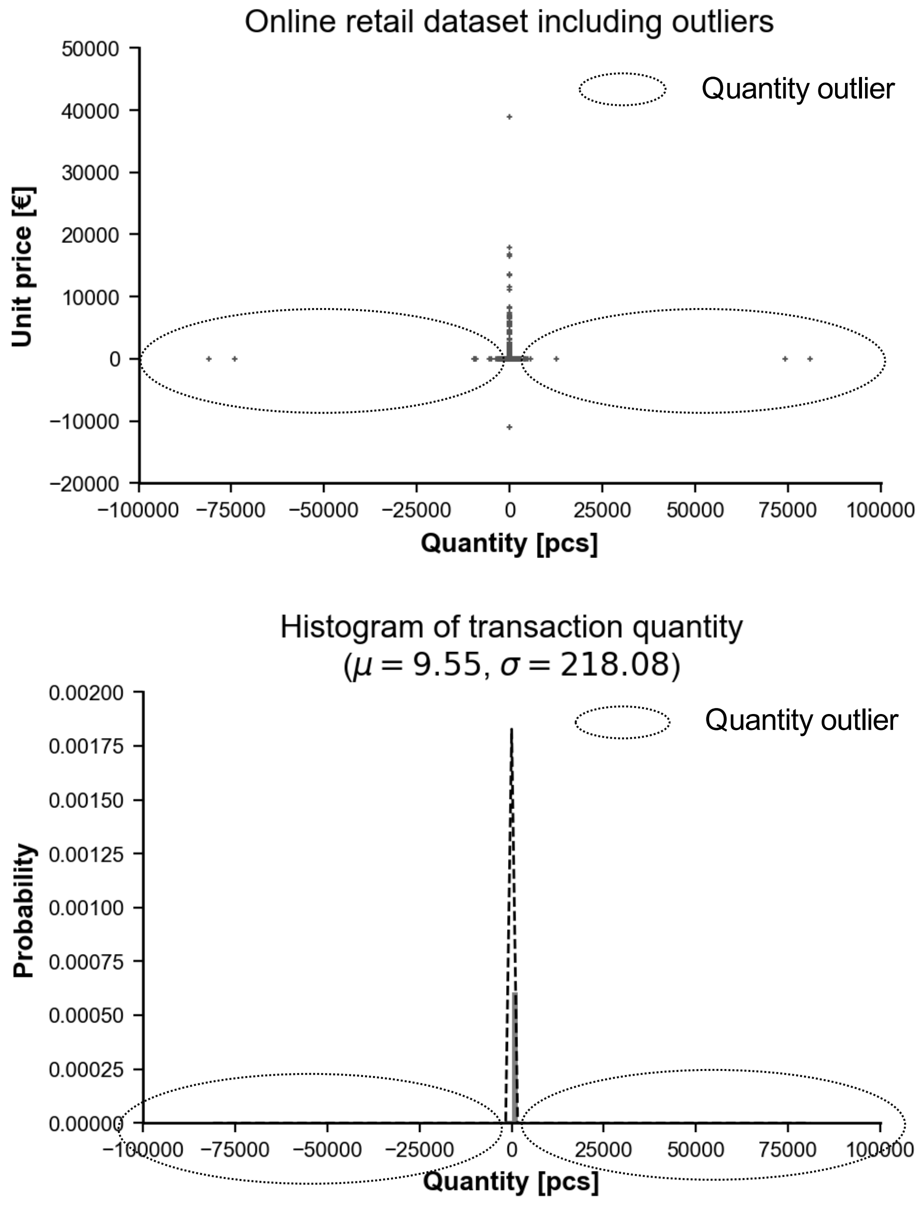


Fig. 2.13 Anomaly detection example using histograms based on [CSG12]. Image generated with python and sci-kit-learn [Ped11]

2.4 Predictive Maintenance in practice

Although PdM has shown its effectiveness (see 2.4.1) it is barely implemented in the industry (see 2.4.2) due to a variety of reasons (see 2.4.3).

2.4.1 Effectiveness of PdM

According to several studies evaluated by [Sul10] for the United States Department of Energy a company in the energy industry can benefit from a successful PdM implementation on average by:

- Return on investment: 10 times
- Reduction in maintenance costs: 25% to 30%
- Elimination of breakdowns: 70% to 75%
- Reduction in downtime: 35% to 45%
- Increase in production: 20% to 25%

Similar in-depth studies have not been conducted for other industries. A possible reason will be explained in 2.4.3.

2.4.2 Frequency of implementation

75% of the questioned companies in [DBG17] have not implemented and scaled a single PdM project.

2.4.3 Example challenges in implementing PdM

- Measuring maintenance performance [PK06]. This might be a possible reason that there are few numbers about the effectiveness of PdM available (see also 2.4.1)
- Many of the implemented programs “have failed to generate measurable benefits” due to missing changes in the workplace [Mob02]
- Not enough data [Gil17]
- Failure to justify the program [N.N18a]

2.5 Conclusion

The existing literature about implementing PdM is not enough as companies are still not able to implement PdM.

Most companies use contract personnel for their PdM program [Mob02], which is still up to date as a Google search for “predictive maintenance contractor” gave back 312.000 and “predictive maintenance consulting” around 2.320.000 entries (10.03.2018).

It cannot be due to missing literature about machine learning as there are several machine learning libraries for various programming languages available (R, Python / sci-kit-learn, C++ / TensorFlow, …) for free including extensive tutorials (for example there are 29.200 videos for “sci-kit-learn tutorial” on YouTube).

The author concludes that companies are not implementing PdM, because they do not know how to economically utilize machine learning algorithms for PdM. This guide is designed to build a bridge over the gap between science and industry and should help companies in mastering daily challenges, e.g. the challenges mentioned in 2.4.3.

3 Selection of critical machines and components

In this chapter, the economically viable selection of critical machines and components is explained. Two lead questions will be answered in this chapter:

1. Which steps does a company need to take to increase the efficiency of production?
2. How can a company choose the right maintenance type for each machine & component?

3.1 Selection of critical machines

At first, it will be explained how machines suited for PdM can be found by identifying bottleneck machines. After analyzing the OEE and finding out that the asset reliability is the main reason for the bottleneck, several methods to check the current maintenance system are explained. If this does not improve asset reliability, then a critical machine is found, and further investigation is needed.

3.1.1 Identifying bottleneck machines

"A bottleneck is defined as a machine that has a negative impact on the output of other machines by a disturbance of the throughput" [Wed15]. One example of a bottleneck station can be seen in Fig. 3.1, in which station B limits the total output, although all other machines have a higher capacity.

A bottleneck may change over time, so two types of bottlenecks exist [Wed15]:

1. **Short-term bottleneck**, which usually lasts for a day at most. Example: a single breakdown
2. **Long-term bottleneck**, which usually lasts longer than a day. Example: repeated breakdowns or a machine with insufficient maximum capacity

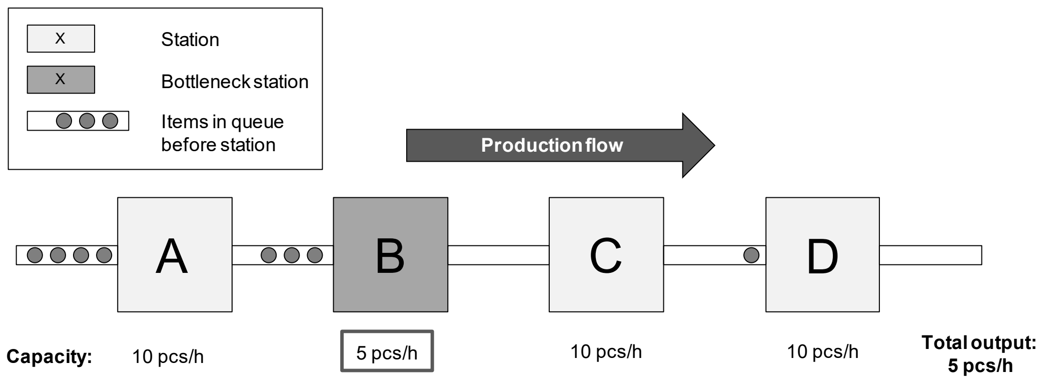


Fig. 3.1 One example of a bottleneck machine.

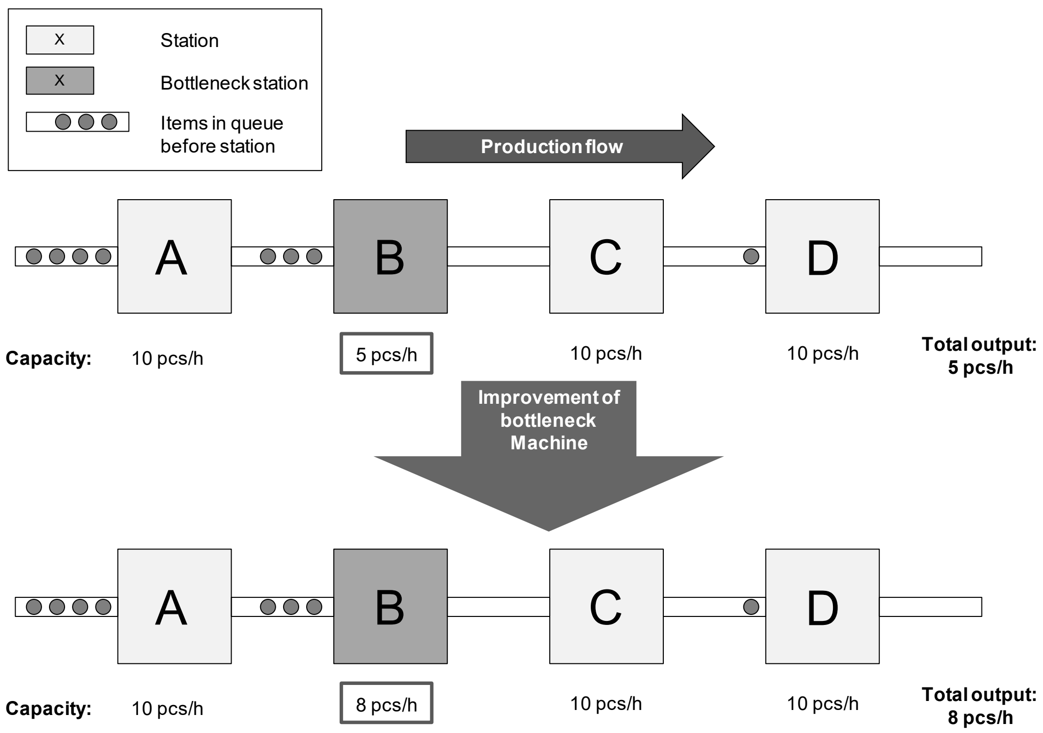


Fig. 3.2 Improving the capacity of the bottleneck machine results in a higher total output for the line.

Improving the KPIs on the long-term bottleneck machine is, therefore, improving the production output, see also Fig. 3.2.

There are several bottleneck identification methods available, see also Fig. 3.3. The focus is on the bottleneck identification method based on queue length in front of each machine as it is simple and easily implemented. One must be careful using this method, as it is not always correct (e.g. if several machines with the same measure index exist) [Zha10].

There are three main steps to identify a bottleneck using the queue length in front of each machine [Ros14; RLD14]:

1. Draw for each buffer in the production line the bottleneck direction arrow based on following rules:
   - If the buffer between two processes is full or rather full, the bottleneck is probably downstream. Therefore, draw an arrow against the flow of production
   - If the buffer is empty or rather empty, the bottleneck is probably upstream. Therefore, draw an arrow in direction of the production
   - If the inventory is half full, the bottleneck may be in either direction. Therefore, draw nothing.
2. The bottleneck then must be between the arrows pointing toward each other (see also Fig. 3.4).
3. Repeat and choose the bottleneck with the primary frequency (long-term bottleneck)

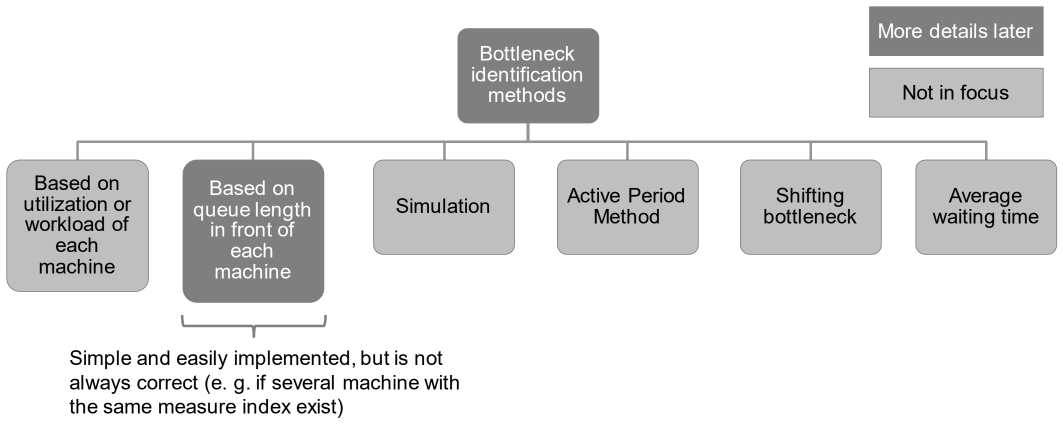

More details later
Not in focus
Bottleneck identification methods
Based on utilization or workload of each machine
Based on queue length in front of each machine
Simulation
Active Period Method
Shifting bottleneck
Average waiting time
Simple and easily implemented, but is not always correct (e. g. if several machine with the same measure index exist)

Fig. 3.3 There are several bottleneck identification methods available [Zha10] [RLD14] [Axe17]

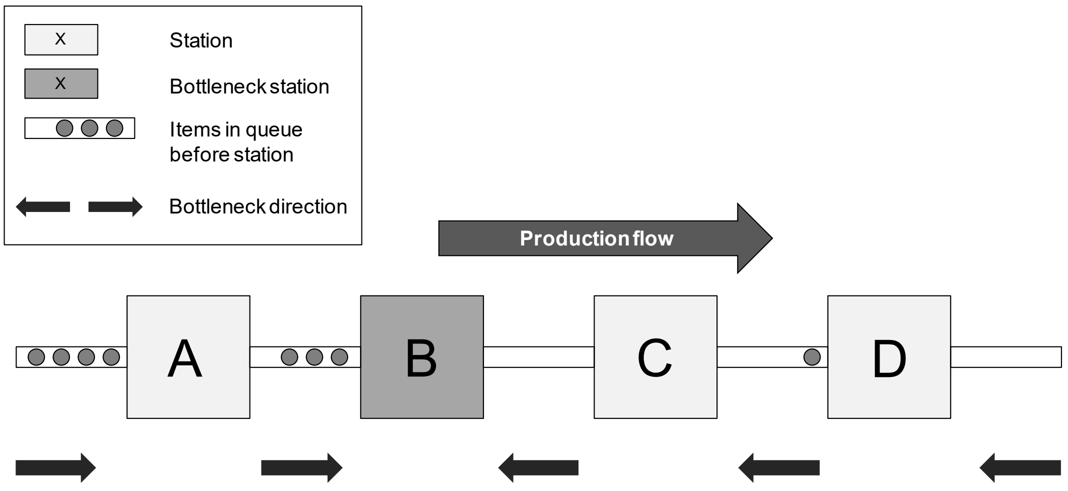


Fig. 3.4 The bottleneck identification method using the queue length in front of each machine

3.1.2 Identifying bottleneck causes

After identifying the bottleneck machine, the cause for the bottleneck must be further analyzed.

There are two main causes of a bottleneck machine [Axe17] (see also Fig. 3.5):

1. Inefficient process
2. Insufficient capacity

An **inefficient process** usually results in a low OEE (availability, quality and/or performance) and can be fixed by removing the underlying issues (see also Fig. 3.5 for more examples).

Insufficient capacity usually results in high inventory before the station despite high OEE. To fix this the entire line needs to be re-balanced (moving work steps from the bottleneck machine to other machines), new staff has to be set or an additional machine must be bought.

Bottleneck cause	**Results in**	**Fix it by**
Inefficient process	Low availability (e.g. unplanned breakdowns)	Improving maintenance
	Low quality (e. g. high scrap)	incoming quality checks, trainings, new SOPs, etc.
	Low performance (e. g. high minor stoppages like cleaning or idle time due to waiting for operator)	increase machine speed, improve men/machine collaboration
Insufficient capacity	High inventory before the station despite high OEE	→ Re-balance the line → Add a new machine / more personnel

Fig. 3.5 Two main causes for a bottleneck

This means, that in all further chapters an availability issue is automatically assumed as otherwise improving maintenance would not be economically viable. Low availability can be seen in the OEE waterfall, for example in the OEE waterfall in Fig. 3.6.

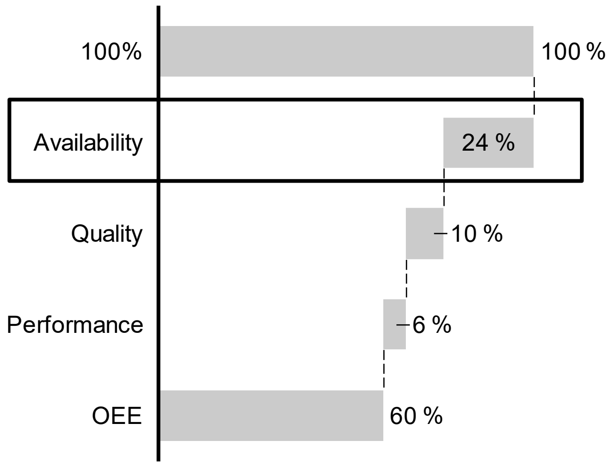


Fig. 3.6 Example OEE waterfall with availability as the main issue

3.1.3 Analyzing existing maintenance strategies and -costs

The first step to improve the availability, and therefore removing the bottleneck, is to check the efficiency of the current maintenance processes. The current maintenance process can be improved by [SH04]:

- defining work requests in a standard way to reduce checking and rework waste

- selecting and prioritizing tasks based on criticality to cancel non-value adding work and leveling the remaining work
- planning the task and securing the necessary resources, tools and material to eliminate execution waste (e.g. waiting for permits, searching for parts)
- executing the task according to the plan (derived from [SH04]) to prevent waste by disruption of maintenance work, canceling nonvalue-adding work (e.g. “Can you help me shortly with this machine?”)
- measuring maintenance performance & performance management using KPIs for continuous improvement

3.2 Selection of critical components

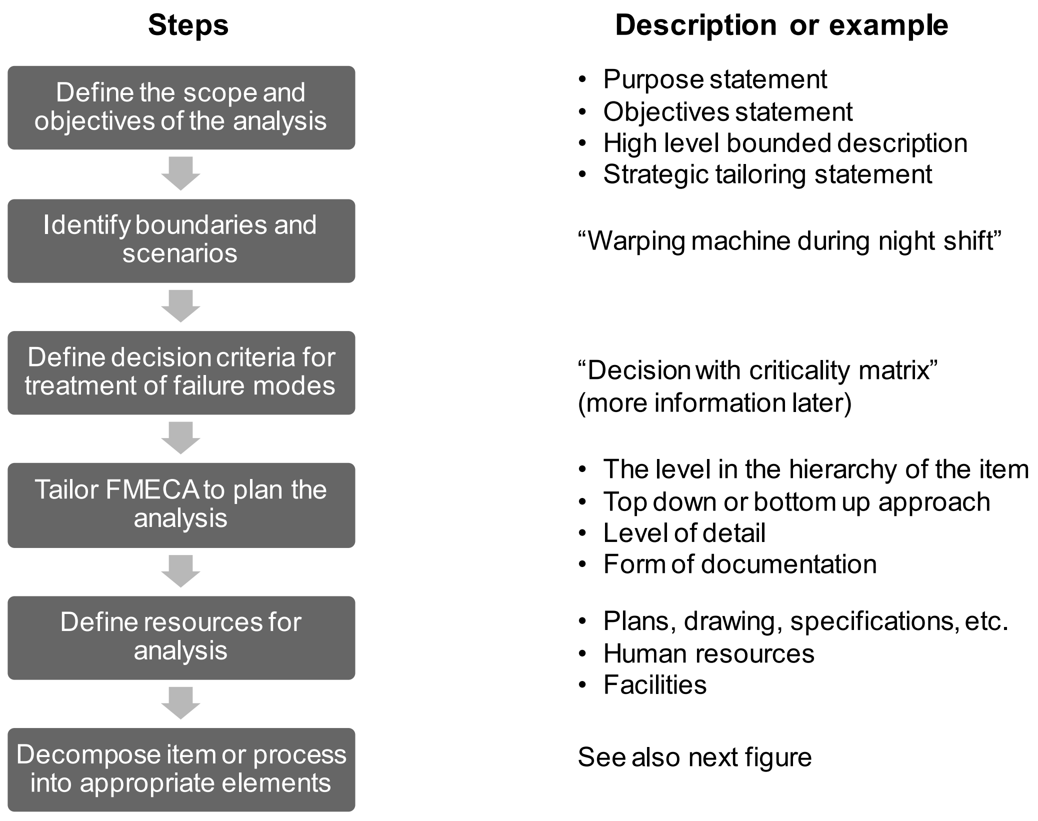
If asset reliability is still an issue a Failure mode, effects and criticality analysis (FMECA) is conducted to find the critical components.

3.2.1 Failure Mode, Effects and Criticality Analysis (FMECA)

Failure mode, effects analysis (FMEA) is “a systematic method for evaluating an item or process to identify the ways in which it can potentially fail” [DIN 60812]. An FMECA is an FMEA plus an assessment of the criticality of each failure mode to prioritize them for action and consists out of three main steps [DIN 60812]:

1. plan the FMECA
2. perform the analysis
3. document the analysis

Planning the FMECA starts with defining the scope and objectives of the analysis and identifying boundaries and scenarios. Then the decision criteria for treatment of failure modes are defined and the FMECA is tailored to plan the analysis. Finally, the resources are defined, and the item or process decomposed into appropriate elements. A description and/or example for each of these steps can be seen in Fig. 3.7, a decomposed item is shown in Fig. 3.8 [DIN 60812].



Steps	Description or example
Define the scope and objectives of the analysis	• Purpose statement • Objectives statement • High level bounded description • Strategic tailoring statement
Identify boundaries and scenarios	“Warping machine during night shift”
Define decision criteria for treatment of failure modes	“Decision with criticality matrix” (more information later)
Tailor FMECA to plan the analysis	• The level in the hierarchy of the item • Top down or bottom up approach • Level of detail • Form of documentation
Define resources for analysis	• Plans, drawing, specifications, etc. • Human resources • Facilities
Decompose item or process into appropriate elements	See also next figure

Fig. 3.7 Steps to plan an FMECA [DIN 60812]

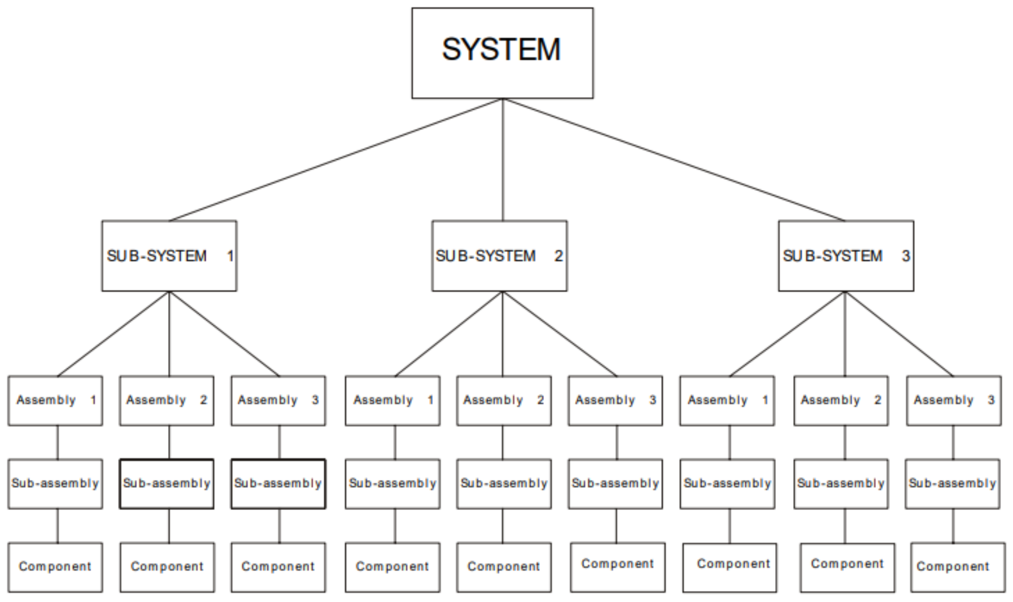


Fig. 3.8 Decompose item or process into appropriate elements [Rob01]

During the step **Perform the analysis** the functions and performance standards of each sub-component are identified together with the failure modes, their effects, and their cause (see also Fig. 3.9). For every failure mode, the detection method is discovered, and the severity of the failure determined. After estimating the likelihood of the failure mode, the criticality is evaluated using either the Risk priority number (RPN) or the criticality matrix. The RPN is calculated by multiplying severity, occurrence, and detectability, in which all these three factors are determined on a scale e.g. from 1 to 10. One example of a criticality matrix can be seen in Fig. 3.11 in which component failures are rated based on their consequence (e.g. life endangering) and frequency to prioritize them for action.

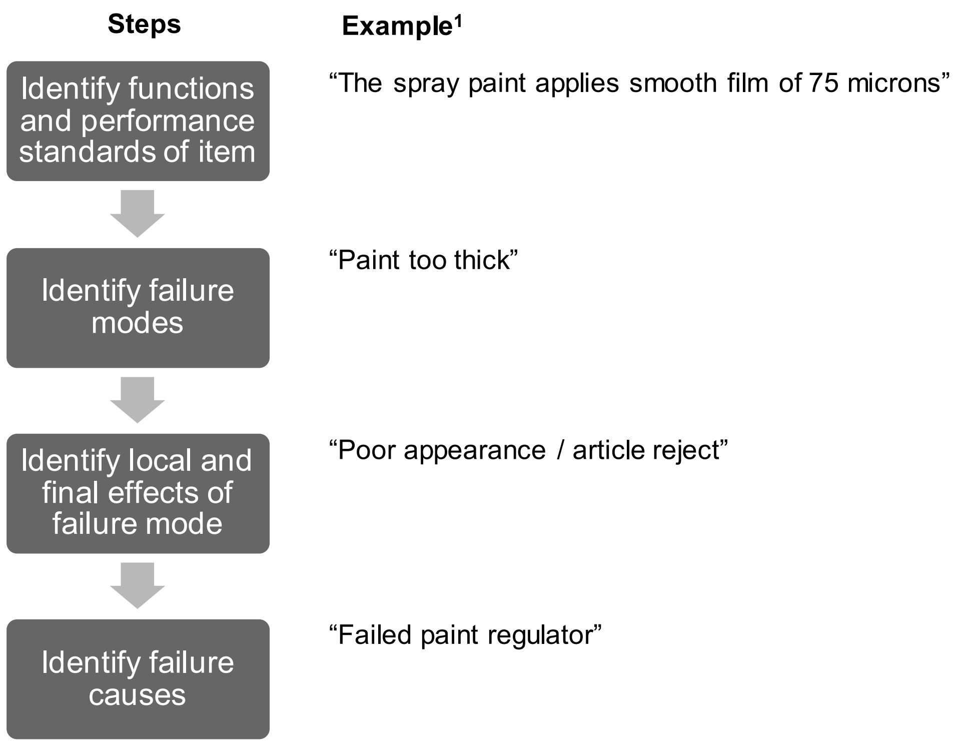

Steps
Example[1]
Identify functions and performance standards of item
"The spray paint applies smooth film of 75 microns"
Identify failure modes
"Paint too thick"
Identify local and final effects of failure mode
"Poor appearance / article reject"
Identify failure causes
"Failed paint regulator"

Fig. 3.9 Step 2 of an FMECA: perform the analysis (1 / 2) with examples from an automobile production line assuming the spray paint is a bottleneck machine. [DIN 60812]

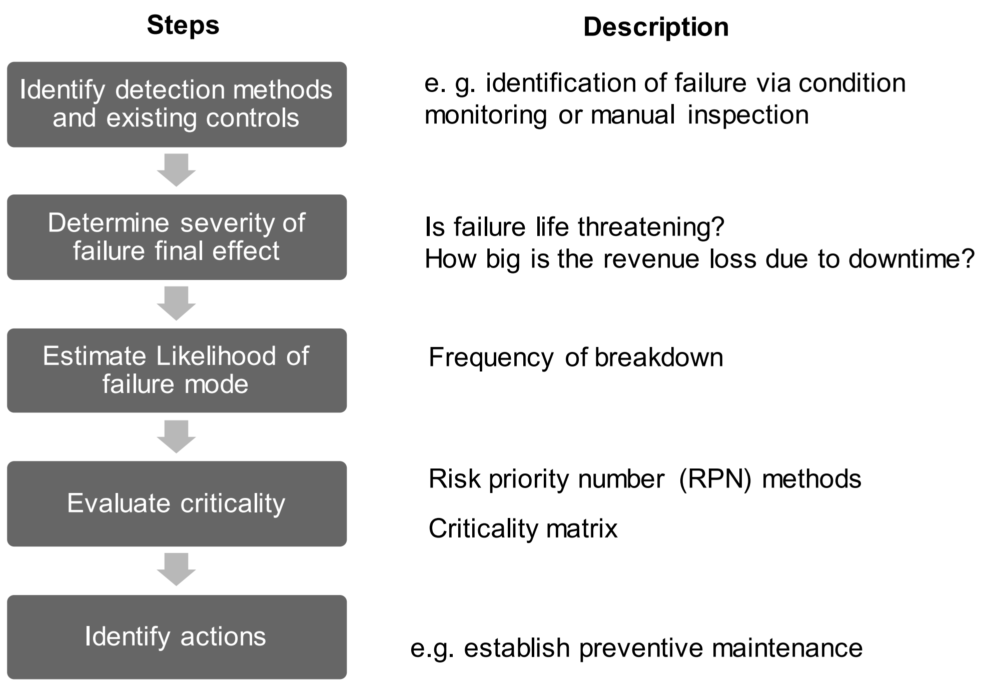
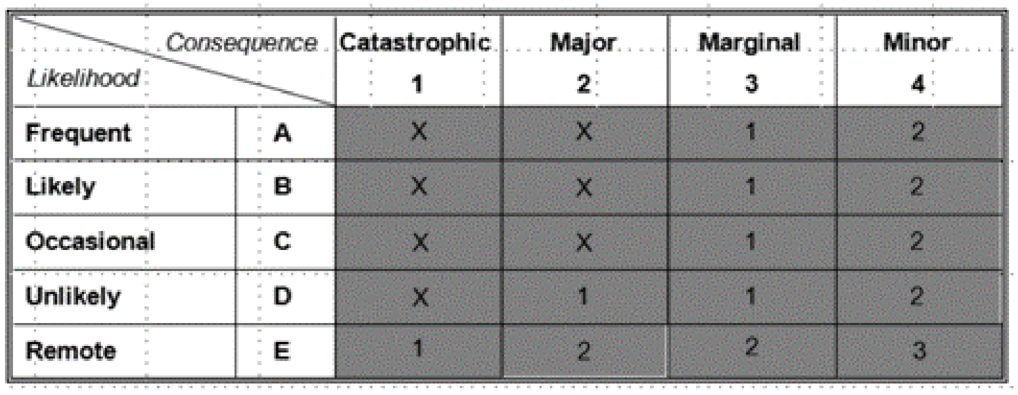

Steps
Description
Identify detection methods and existing controls
e. g. identification of failure via condition monitoring or manual inspection
Determine severity of failure final effect
Is failure life threatening?
How big is the revenue loss due to downtime?
Estimate Likelihood of failure mode
Frequency of breakdown
Evaluate criticality
Risk priority number (RPN) methods
Criticality matrix
Identify actions
e.g. establish preventive maintenance

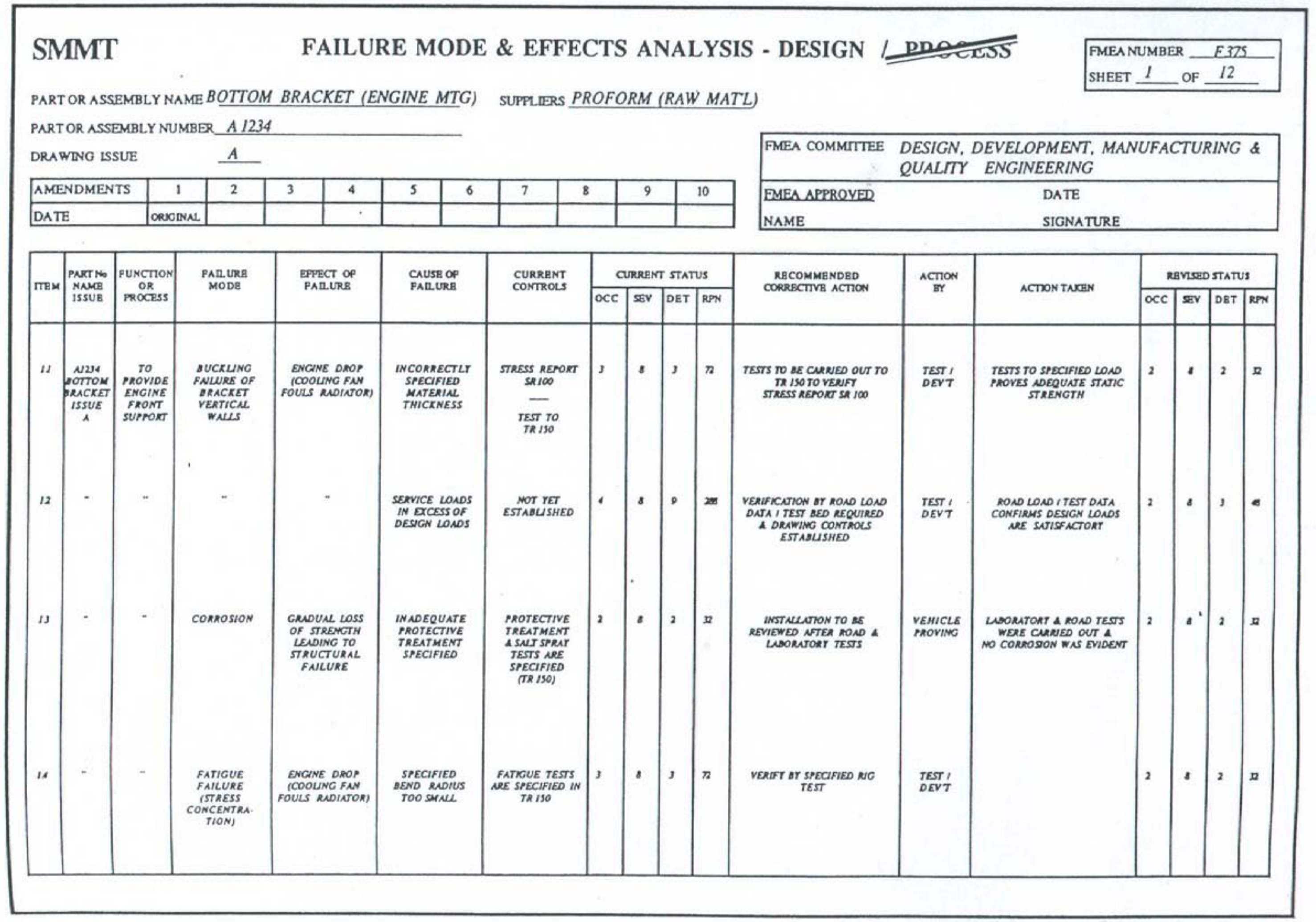
Fig. 3.10 Step 2 of an FMECA: perform the analysis (2 / 2) [DIN 60812]



Likelihood \ *Consequence*		Catastrophic 1	Major 2	Marginal 3	Minor 4
Frequent	A	X	X	1	2
Likely	B	X	X	1	2
Occasional	C	X	X	1	2
Unlikely	D	X	1	1	2
Remote	E	1	2	2	3

Fig. 3.11 Example of a criticality matrix [DIN 60812]

Finally, the **results of the FMECA are documented**. Every company has its own documentation templates. One example can be seen in Fig. 3.12.



SMMT

FAILURE MODE & EFFECTS ANALYSIS - DESIGN / ~~PROCESS~~

FMEA NUMBER E375
SHEET 1 OF 12

PART OR ASSEMBLY NAME BOTTOM BRACKET (ENGINE MTG) SUPPLIERS PROFORM (RAW MAT'L)

PART OR ASSEMBLY NUMBER A 1234

DRAWING ISSUE A

FMEA COMMITTEE DESIGN, DEVELOPMENT, MANUFACTURING & QUALITY ENGINEERING

FMEA APPROVED DATE

NAME SIGNATURE

AMENDMENTS	1	2	3	4	5	6	7	8	9	10
DATE	ORIGINAL									

ITEM	PART No NAME ISSUE	FUNCTION OR PROCESS	FAILURE MODE	EFFECT OF FAILURE	CAUSE OF FAILURE	CURRENT CONTROLS	CURRENT STATUS				RECOMMENDED CORRECTIVE ACTION	ACTION BY	ACTION TAKEN	REVISED STATUS			
							OCC	SEV	DET	RPN				OCC	SEV	DET	RPN
11	A1234 BOTTOM BRACKET ISSUE A	TO PROVIDE ENGINE FRONT SUPPORT	BUCKLING FAILURE OF BRACKET VERTICAL WALLS	ENGINE DROP (COOLING FAN FOULS RADIATOR)	INCORRECTLY SPECIFIED MATERIAL THICKNESS	STRESS REPORT SR100 — TEST TO TR 150	3	8	3	72	TESTS TO BE CARRIED OUT TO TR 150 TO VERIFY STRESS REPORT SR 100	TEST / DEV'T	TESTS TO SPECIFIED LOAD PROVES ADEQUATE STATIC STRENGTH	2	8	2	32
12	-	-	-	-	SERVICE LOADS IN EXCESS OF DESIGN LOADS	NOT YET ESTABLISHED	4	8	9	288	VERIFICATION BY ROAD LOAD DATA / TEST BED REQUIRED & DRAWING CONTROLS ESTABLISHED	TEST / DEV'T	ROAD LOAD / TEST DATA CONFIRMS DESIGN LOADS ARE SATISFACTORY	2	8	3	48
13	-	-	CORROSION	GRADUAL LOSS OF STRENGTH LEADING TO STRUCTURAL FAILURE	INADEQUATE PROTECTIVE TREATMENT SPECIFIED	PROTECTIVE TREATMENT & SALT SPRAY TESTS ARE SPECIFIED (TR 150)	2	8	2	32	INSTALLATION TO BE REVIEWED AFTER ROAD & LABORATORY TESTS	VEHICLE PROVING	LABORATORY & ROAD TESTS WERE CARRIED OUT & NO CORROSION WAS EVIDENT	2	8	2	32
14	-	-	FATIGUE FAILURE (STRESS CONCENTRATION)	ENGINE DROP (COOLING FAN FOULS RADIATOR)	SPECIFIED BEND RADIUS TOO SMALL	FATIGUE TESTS ARE SPECIFIED IN TR 150	3	8	3	72	VERIFY BY SPECIFIED RIG TEST	TEST / DEV'T		2	8	2	32

Fig. 3.12 Example of an FMECA documentation sheet [Rob01]

3.3 Selection of the right maintenance type for critical components

After finding out the critical components we analyze the failure models of the components and define for each of them a cost-efficient maintenance type.

There are six failure models for industrial equipment in describing the course of failure probability of a component over time (see also Fig. 3.13) [HB11]:

1. Infant Mortality
2. Wear-Out
3. Bathtub
4. Fatigue
5. Initial Break-In Period
6. Random

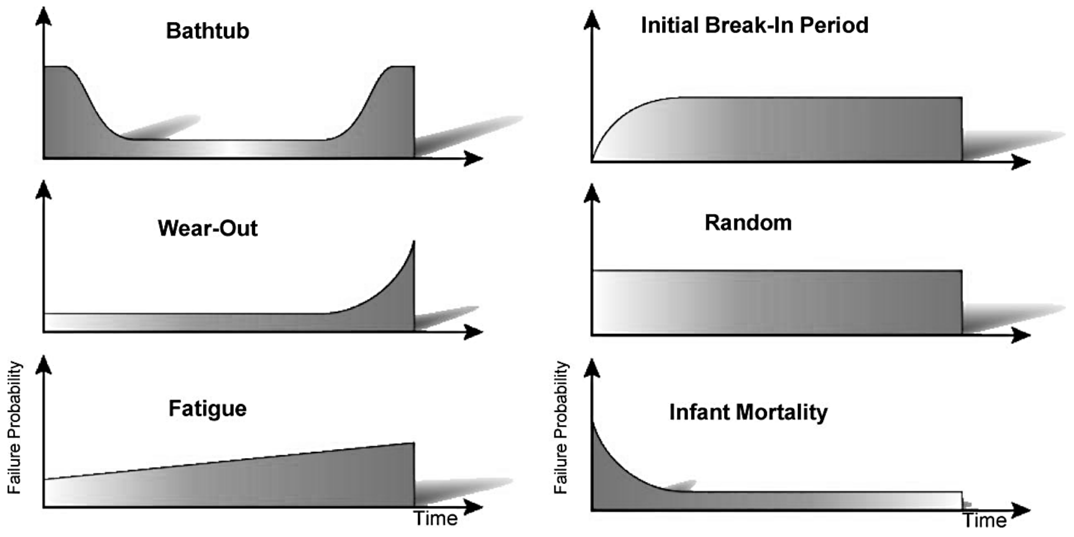
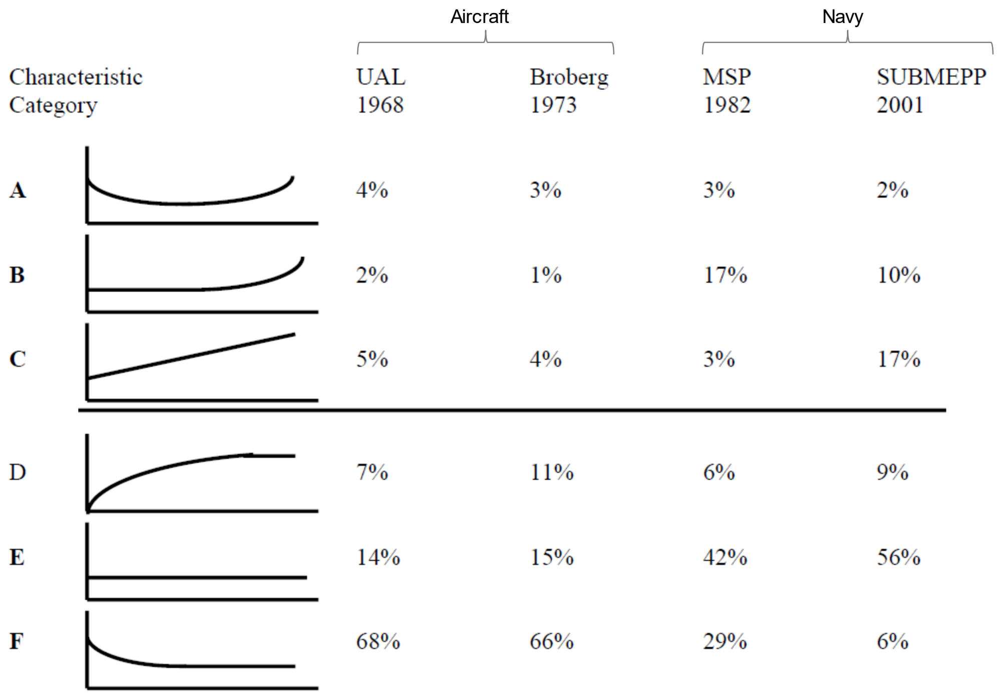


Fig. 3.13 Six failure models for industrial equipment based on [HB11]

The frequency of each failure model varies heavily on the industry and cannot be generalized (see also Fig. 3.14) [All05], e.g. the frequency of infant mortality ranges from 68% in aircraft (UAL) to 6% in navy/submarines (SUBMEPP). Other extensive studies about the failure models as mentioned in the figure are not available. Therefore, the failure model needs to be analyzed for each component and no shortcuts can be made.



	Aircraft		Navy	
Characteristic Category	UAL 1968	Broberg 1973	MSP 1982	SUBMEPP 2001
A	4%	3%	3%	2%
B	2%	1%	17%	10%
C	5%	4%	3%	17%
D	7%	11%	6%	9%
E	14%	15%	42%	56%
F	68%	66%	29%	6%

Fig. 3.14 The frequency of each failure model varies heavily on the industry and cannot be generalized. Based on [All05]

In chapter 2.2 three sub-categories of preventive maintenance have been described: predetermined, opportunistic and condition-based maintenance. Opportunistic maintenance can be conducted on all components but depends on the failure modes of other components, so we will not look further into it.

It is obvious that predetermined maintenance, which means exchanging parts after a fixed amount of time or units, can only apply to components that have an increasing failure probability (Bathtub, Wear-Out and Fatigue) while condition-based maintenance can be theoretically applied on all failure models. An overview of the applicability can be found in Fig. 3.15.

Predictive maintenance offers the most effective maintenance planning but should be used on critical equipment only due to high costs (see also Fig. 3.16).

It can be applied to all failure models like CBM but it also shows the remaining useful lifetime (RUL) like predetermined maintenance. Knowing the RUL is useful when

1. Spare parts are very expensive and/or have long delivery times. With the RUL inventory costs due to just-in-time spare parts delivery can be reduced as well as avoiding machine breakdown due to spare parts delivery
2. The machine has almost to none planned stops and it is business critical that it runs through. With the RUL maintenance tasks can be bundled and executed

during unavoidable stops (e.g. changeovers, security checks, etc.) or the stops can be planned during a time with low productivity (holiday season, etc.)

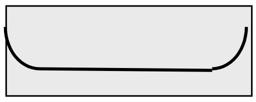
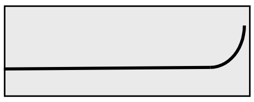
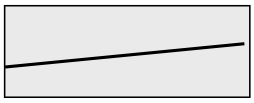
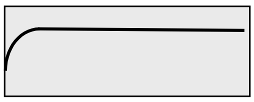
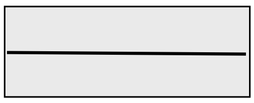
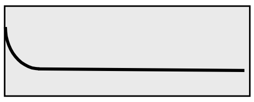
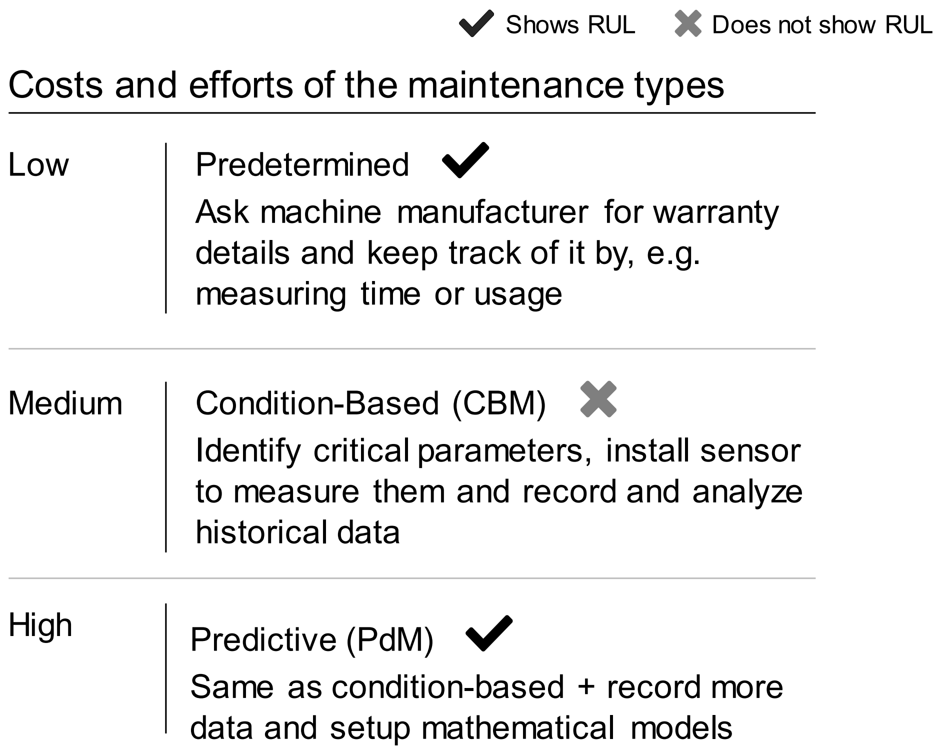
Conditional probability of failure	Failure mode	Applicability of maintenance type		
		Predetermined	Condition-based	Predictive
	Bathtub	✓	✓	✓
	Wear-out	✓	✓	✓
	Fatigue	✓	✓	✓
	Initial break-in period	✗	✓	✓
	Random	✗	✓	✓
	Infant mortality	✗	✓	✓

Fig. 3.15 Overview over the applicability of maintenance types



✓ Shows RUL ✖ Does not show RUL

Costs and efforts of the maintenance types

Low	Predetermined ✓ Ask machine manufacturer for warranty details and keep track of it by, e.g. measuring time or usage
Medium	Condition-Based (CBM) ✖ Identify critical parameters, install sensor to measure them and record and analyze historical data
High	Predictive (PdM) ✓ Same as condition-based + record more data and setup mathematical models

Fig. 3.16 Costs and effort of the maintenance types

3.4 Conclusion

The two lead question can be answered as follows:

1. Which steps does a company need to take to increase the efficiency of the production?
    a. Find the bottleneck machine
    b. Eliminate bottleneck cause by improving the OEE or increasing capacity
2. How can a company choose the right maintenance type for each machine & component?
    a. Conduct FMECA on bottleneck machine if availability is an issue
    b. Choose for each critical component the correct maintenance type (predetermined, condition-based or PdM) based on the failure mode and the need to know the RUL

In the next chapter, the implementation of PdM is described.

4 Systematic implementation of Predictive Maintenance on critical components

In this chapter, the economically viable implementation of PdM is explained. It is assumed that it is applied on a bottleneck machine with a component suited for PdM (see also chapter 3). Three lead questions will be answered in this chapter:

1. How to assess whether the existing data is sufficient for PdM or if additional sensors are required?
2. How to select, train and optimize machine learning algorithms when doing PdM, so it generates a measurable impact?
3. What needs to be done after a functional algorithm is developed?

4.1 Deciding the type of PdM model

There are four main types of models to estimate approximate time of a failure (see also Fig. 4.1) [Jah15]:

1. Physical Model-Based Methodology
2. Knowledge-Based Models
3. Data-Driven Models
4. Combinations

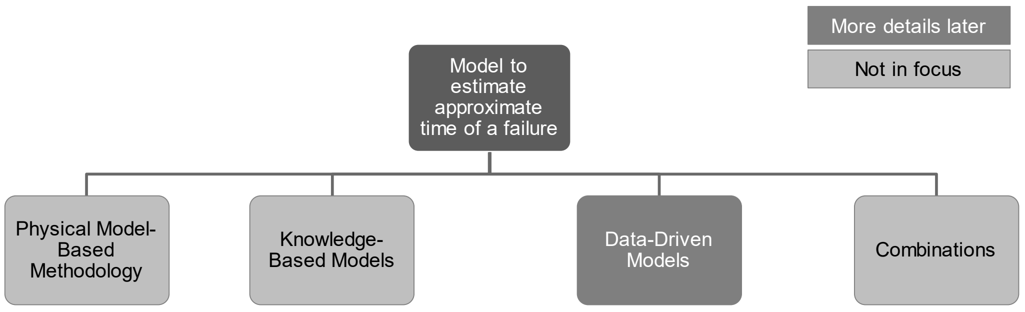


Fig. 4.1 Four main types of models to estimate approximate time of a failure [Jah15]

Physical models in **Physical Model-Based Methodology** are realized based on statistically validated mathematical models and are created by domain experts[1]. Advantage is the good accuracy and the disadvantages are high costs and component specialty.

[1] A person who is an expert in his domain, e.g. the machine or the specific component

A **Knowledge-Based Model** is created by domain experts, but unlike the previous method it is not based on mathematical models, but purely on the knowledge of the domain expert.

Data-driven models are based on statistical and learning techniques and will be in focus on the next pages as they are the easiest to create, as almost no domain knowledge is needed to create them[2]. Finally, there are **combinations** of the models stated before.

Data-driven models can be separated into classification and regression algorithms (see also Fig. 4.2).

The classification algorithms provide a low level of detail, but their financial impact can be easily calculated, which makes them a great tool if a discrete approximation of the RUL is enough. The calculation can be done by researching the average costs for an over- and undercast and then applying the algorithm on historical data.

The regression algorithms provide a high level of detail, but their financial impact is hard to calculate, which makes them only viable if a precise and continuous RUL is really needed. The calculation can be done by creating a cost function based on the exchange policy and then applying the algorithm on historical data.

Level of detail of the RUL		Effort	Used when
Low / discrete	**Classification** Calculates discrete RUL, e.g. • Green (>50 days) • Yellow (50 – 10 days) • Red (<10 days)	**Low** Financial impact easily estimated using false-positives / false negatives and ROC-curves	A discrete approximation of the RUL is enough
High / continuous	**Regression** Calculate a continues RUL, e. g. 156h till failure	**High** Financial impact needs to be estimated using cost functions	Precise and continuous RUL is needed.

Fig. 4.2 Two types of data-driven models

For a successful data-driven model, historical data of component failures including machine and external sensor data need to be available. The historical data needs to be available for the entire wear process (installation till failure) to calculate the remaining useful lifetime or categorize wear states properly (depending on the algorithm). The features must show a correlation and have causality. Several algorithms like the random forests can find the most correlated features themselves but cannot check for causality.

[2] Domain knowledge is not needed to during the model creation process, but at the end of the process to ensure that the model is not based on random correlations but on actual physical relationships

4.2 Evaluating the financial impact

A PdM implementation is an investment over a long period, therefore its net present value needs to be higher than the alternatives to be worthwhile (see also Fig. 4.3). Other methods of comparing investments exist (e.g. internal rate of return), but are also based on the principle of the net present value and will show the same result (invest or not invest). [Bre12]

Net present value: $k_{PdM} = \sum_{t=0}^{T} \frac{z_t}{(1+i)^t} > k_{Alternative}$ for a worthwhile investment

Fig. 4.3 The net present value must be higher than the alternative to be worthwhile. t = time; T = end time; z_t = payment in t[3], i= interest rate (assumed to be constant)

On the following pages, predetermined maintenance is used as the alternative investment, as it is the simplest preventive maintenance available with also the smallest implementation costs. Two types of payments are relevant for assessing maintenance investments:

1. implementation costs (negative payment in t=0) and
2. running costs (negative payments in other time periods)

Positive payments exist but are not used in maintenance as maintenance is about reducing costs and not creating revenue. Below an example for running and implementation costs for PdM and predetermined maintenance.

Implementation costs for predetermined maintenance are, for example, for

- Defining the exchange policy (personnel or external contractors)
- Labor cost (maintenance engineer)
- Software costs to keep track of the time (e.g. maintenance software)

Running costs for predetermined maintenance are mainly average costs for model inaccuracy. It is calculated by defining the exchange rate, e.g. time after 10% of the equipment has failed in history or experiments, and then creating and applying a cost function based on depreciation (because of unnecessary exchange) or labor costs and revenue loss (because of breakdown)

Implementation costs for PdM are, for example, for

[3] A negative payment in t=0 is called one-time costs and all other negative payments are called running costs

- Defining the exchange policy (personnel or external contractors)
- Labor cost (data scientist, IT integration, maintenance engineers, etc.)
- Hardware costs for additional sensors
- Software costs (if commercial software for the model generation is used)

Running costs for PdM maintenance are mainly average costs for model inaccuracy, costs for refining the model due to external factors (e.g. lost accuracy after moving the machine to another place) and maintaining the sensors. Average costs because of model inaccuracy are calculated by creating a mathematical model to estimate the RUL and then creating and applying a cost function based on depreciation (because of unnecessary exchange) or labor costs and revenue loss (because of breakdown).

As already stated in chapter 4.1 evaluating the financial impact is different for regression and classification algorithms.

4.2.1 Evaluating the financial impact for regression algorithms

In the beginning, an exchange policy needs to be defined to create a cost function for regression algorithms and therefore estimate average costs because of model inaccuracy. Average costs because of model inaccuracy per year are calculated by multiplying the amount of components failures per year with the average costs due to model inaccuracy per component.

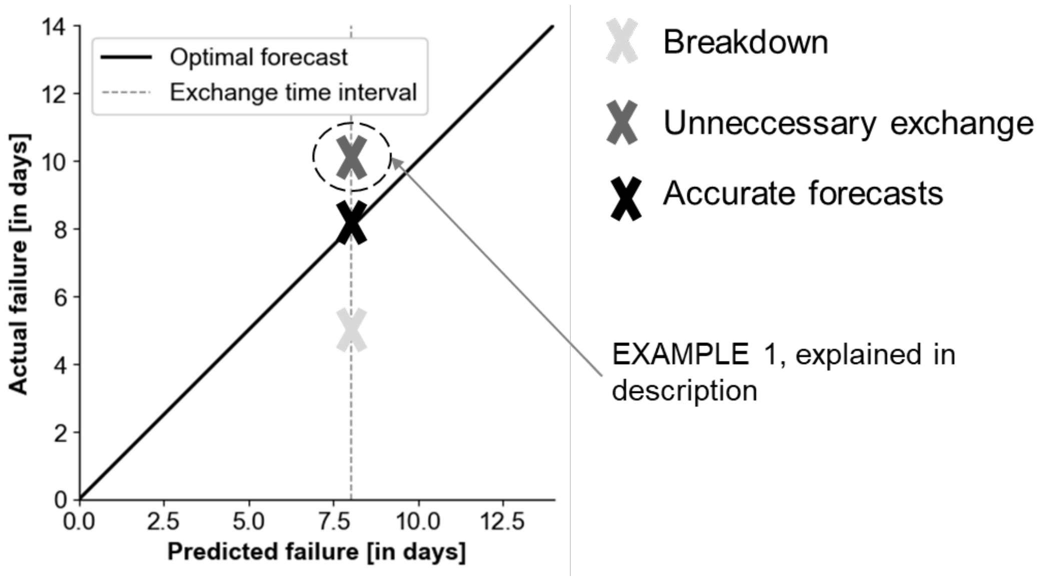


Fig. 4.4 Exchanging after a fixed amount of time (predetermined maintenance). In example 1 the component will be exchanged after 8 days, but it would fail after 12 days. This results in 4 days wasted

The exchange policy defines at which conditions the maintenance order gets executed. The condition mainly depends on the chosen maintenance type. Predicted RUL is shown over the actual RUL for predetermined maintenance in Fig. 4.4, where an exchange always happens after a fixed time or usage. The algorithm involved can either predict the failure correctly (accurate forecast), trigger too early causing an unnecessary exchange (undercast) or trigger too late and cause a breakdown (overcast). This will be used later as the starting point to establish the cost function. Another example of the predicted RUL shown over the actual RUL can be found in Fig. 4.5 for predictive maintenance.

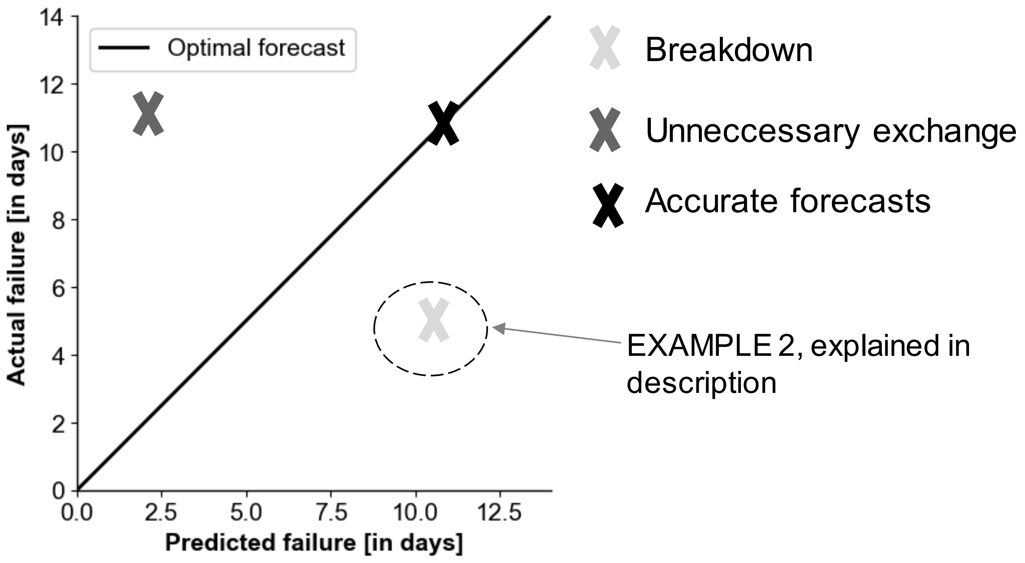


Fig. 4.5 Exchanging based on the forecast. In example 2 the component will be exchanged after 12 days, but it would fail after 6 days. This results in 6 days breakdown (assuming the lead time is higher than 6 days)

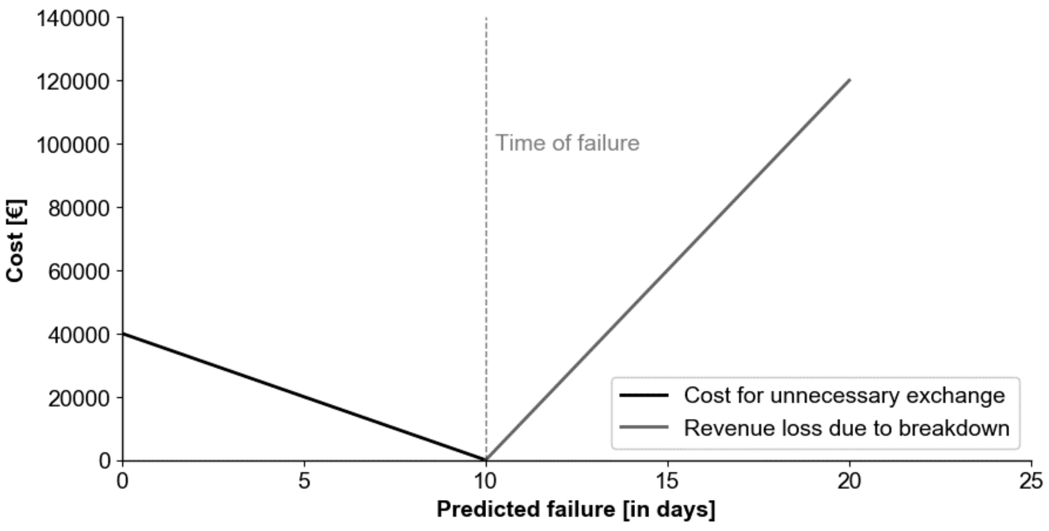


Fig. 4.6 Costs for over- and undercast assuming day 10 is the actual time of failure, 500€ breakdown costs per hour, 30 000 € spare part costs and 10 000 € exchange labor cost

The next step to establish the cost function is to research the costs for unnecessary exchanges and revenue loss due to breakdowns as seen in Fig. 4.6. The cost for unnecessary exchanges can be calculated via depreciation, the revenue loss by the opportunity costs for missing out production. A detailed calculation can be found in chapter 5.3. Depending on the business context, the calculation approach might require adjustments.

Only if the remaining useful lifetime is under the lead time of a component a company will act, resulting in an area of no cost (see Fig. 4.7). Lead time is the time needed to prepare a maintenance action. It is different for every component and company and is based e.g. on whether the spare part is already in the warehouse or needs to be ordered and how long it will take to allocate a maintenance technician.

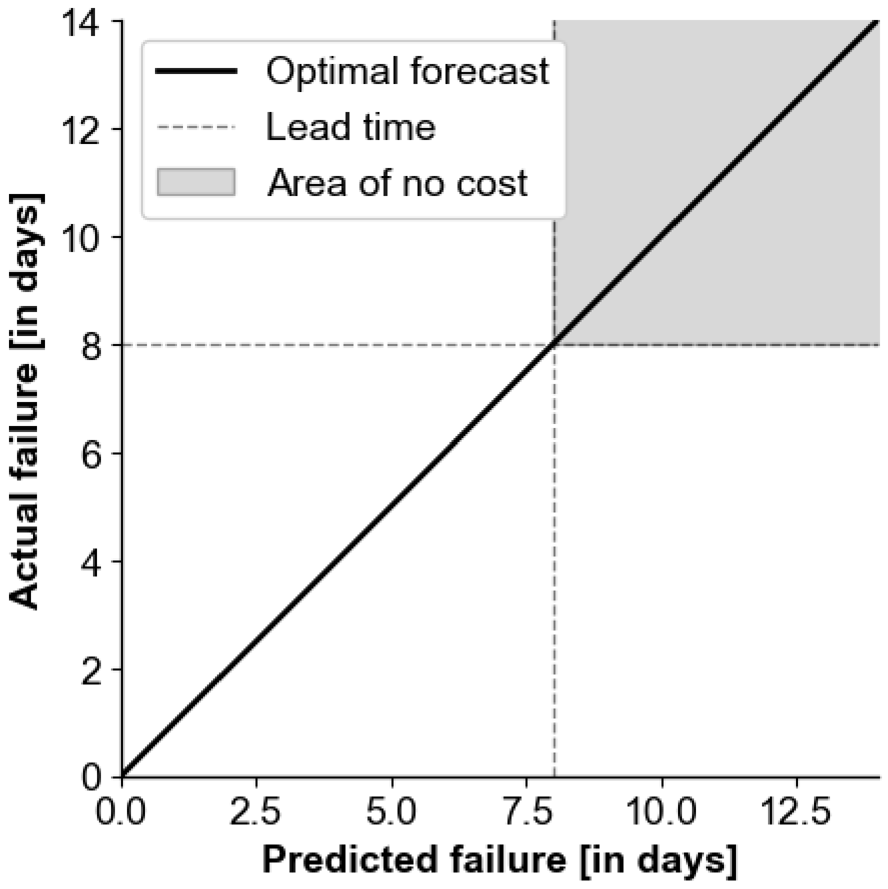


Fig. 4.7 Area of no cost highlighted in light grey. In the figure, a lead time of 8 days is assumed.

Combining Fig. 4.6 and Fig. 4.7 results in the cost function (see Fig. 4.8). The cost function combined with historical data on component failures can be used to calculate the average costs per component due to model inaccuracy. As the breakdown costs are usually higher than the costs for unnecessary exchange the company can add a buffer to their exchange policy and therefore provoking an undercast to prevent high costs due to a breakdown (see Fig. 4.9).

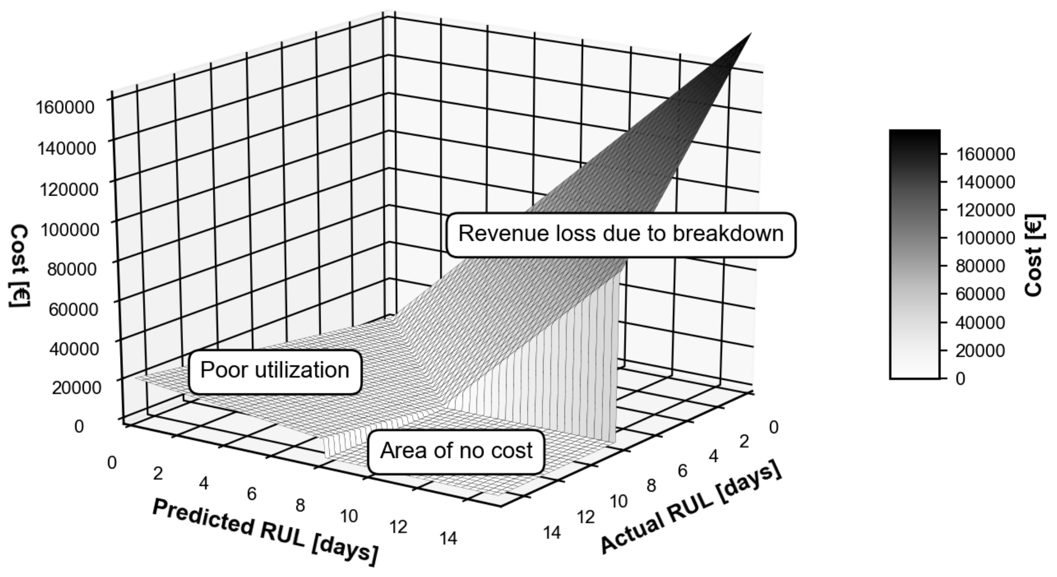


Fig. 4.8 Resulting cost function

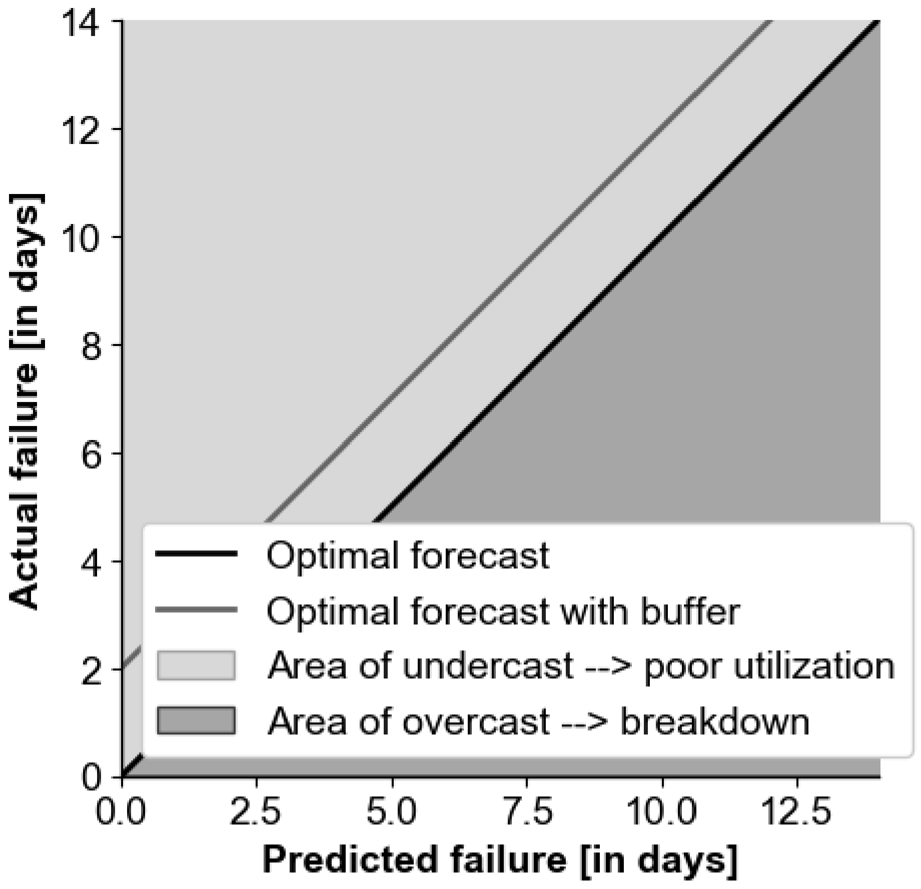


Fig. 4.9 Exchange policy with a buffer of 2 days. This buffer can vary based on the companies needs

4.2.2 Evaluating the financial impact for classification models

The financial impact of a classification model can be estimated using a confusion matrix [Faw06] (see Fig. 4.10).

The confusion matrix is an example of a bivariate PdM model (classes: „exchange“, „not exchange“). A correct prediction results in no cost, but an incorrect prediction results either in suboptimal utilization or breakdown costs.

The average costs per component can then be estimated using historical data and the average costs for a breakdown and suboptimal utilization.

Example:

- Historical data shows 5% false positives, 8% false negatives
- One false positive results in average costs of 5.000€, a false negative in average costs of 50.000€

- The average cost due to model inaccuracy per component is then 0.05 * 5.000 € + 0.08 * 50.000 € = 4250 €

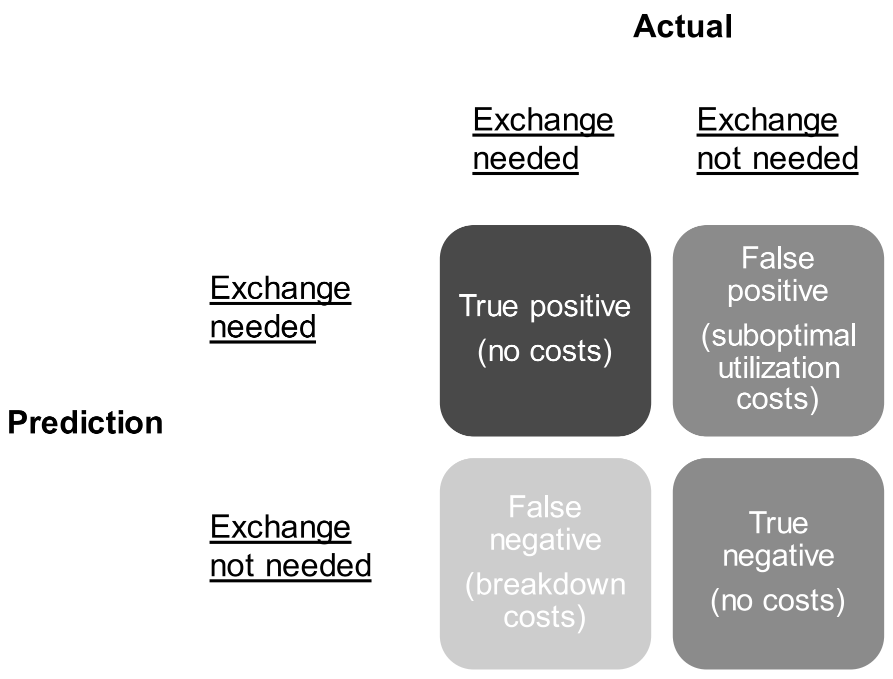


Fig. 4.10 Confusion matrix in maintenance context

A classification model can be optimized using the ROC-curve[4] and making a trade-off between false positive and false negative errors. ROC-curves is a technique for selecting models based on their performance by visualizing the components of the confusion matrix in a graph. More information can be found in [Faw06].

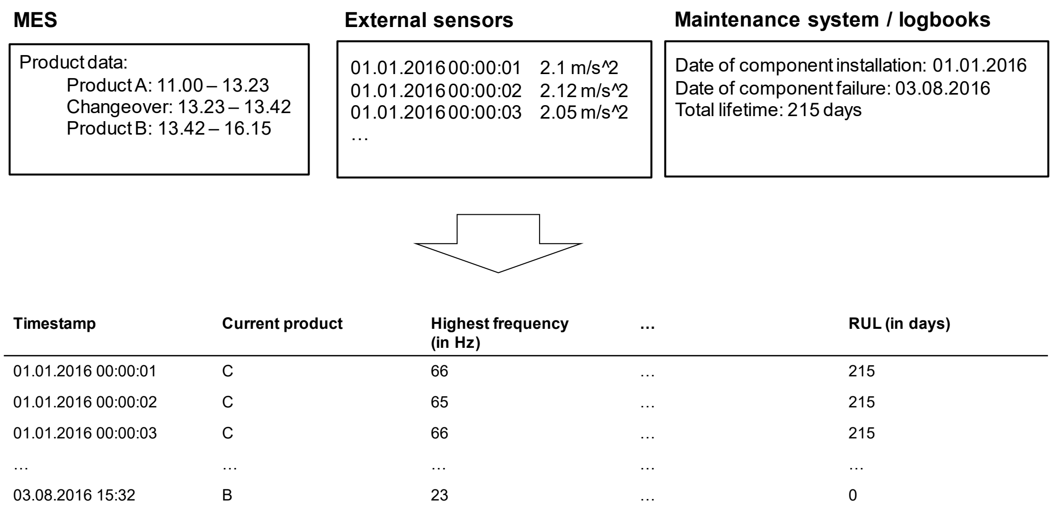
4.3 Creating, selecting, optimizing and automating the PdM model

After deciding the type of the PdM model and understanding how to calculate the average costs due to model inaccuracy, the company needs to test different algorithms and optimize them for the lowest costs.

[4] ROC = Receiver operating characteristics.

4.3.1 Creating the PdM model

Before the algorithms can be created all data needs to be transformed into one time-based data table (see Fig. 4.11) [WFH11]. In the figure, the time, current product and the highest frequency are the features and the RUL is the target. Later one of the processes to transform the data called feature extraction is further explained.



MES

Product data:
Product A: 11.00 – 13.23
Changeover: 13.23 – 13.42
Product B: 13.42 – 16.15

External sensors

01.01.2016 00:00:01 2.1 m/s^2
01.01.2016 00:00:02 2.12 m/s^2
01.01.2016 00:00:03 2.05 m/s^2
…

Maintenance system / logbooks

Date of component installation: 01.01.2016
Date of component failure: 03.08.2016
Total lifetime: 215 days

Timestamp	Current product	Highest frequency (in Hz)	…	RUL (in days)
01.01.2016 00:00:01	C	66	…	215
01.01.2016 00:00:02	C	65	…	215
01.01.2016 00:00:03	C	66	…	215
…	…	…	…	…
03.08.2016 15:32	B	23	…	0

Fig. 4.11 Data from various sources need to be transformed into one data table. The number of sources will depend on the use case.

4.3.2 Selecting the PdM model

Several software (-extensions) exist to compare machine learning algorithms. Three of them are illustrated in Fig. 4.12 and then further explained.

Fig. 4.12 Three popular software (-extensions) to compare machine learning algorithms. From left to right: weka [Mac18], scikit-learn [N.N18b] and R [N.N18c]

The Waikato Environment for Knowledge Analysis, **Weka** for short, is mainly a toolset and provides spare implementations of simple techniques to aid understanding as well as industrial-strength implementation of many popular algorithms. With a framework, in the form of a Java class library, it can be used in real-time environments. [WFH11]

Scikit-learn is a Python module used to integrate state-of-the-art machine learning algorithms for medium scale problems. [Ped11] Because it is a module to the general-purpose language Python it is excellently suited for integrating machine learning algorithms in real-time environments, as external data sources can easily be imported, and results easily exported.

R is a programming language for statistical computing and graphics and provides a wide variety of statistical and graphical techniques including machine learning algorithms in one environment. [N.N18c] As it is mainly a statistical programming language it is not recommend using it in a real-time environment, as it is poor at importing and exporting data in real-time in my opinion. Theoretically, it is possible as seen in [Rya09].

With these tools, a time-based data table can be imported (see Fig. 4.11) to execute, validate and optimize algorithms. Independent from the software two methods exist to validate machine learning models according to [Ron95]:

In the **Holdout method,** the data gets split into training and test set. It is common to use 2/3 of the total data as training and 1/3 as test set. The model is then trained using the training set and then validated with the test set using the cost function we defined before.

The other method is called **Cross-Fold-Validation**, which has different variants. In the k-fold cross-validation, the dataset is randomly split into k mutually exclusive subsets (the folds) of approximately the same size. The model is then trained k-times using the k^{th}- subset as validation and the rest of the data as a training set. The result is the average of the cost function applied to the validation datasets.

4.3.3 Optimizing the model

The machine learning algorithms are then tuned (optimized) by systematically trying out different so-called hyperparameters. Hyperparameters are the configuration parameters that need to be fixed before a machine learning algorithm can be started and changing these can have a huge impact on the resulting model. In the next chapter grid-search, parameter tuning is used, which is trying out different combinations and choosing the ones with the best results. More so-called tuning strategies can be found in [CM15] and [PBB18].

If the result is not satisfactory after optimizing new features can be created using basically three methods:

The first method is an **improved feature extraction**. This means creating new features by extracting more information from existing data, for example using statistical analysis (minimum, maximum, average, variance, etc.), Windowing or Fast-Fourier-Transformations (based on [GS04] and [OAC06]).

The second method is **connecting more existing data sources**. New features are created by adding already recorded data into the mode, for example, digitizing manually recorded data, an MES[5] system for product information and cycle time or PLCs[6] for machine speed, warnings or temperature.

The third and last method is **adding new sensors** that are important in detecting the component failure. For example, vibration sensors for bearings or three phase energy consumption monitor to identify failures in electrical engines. The disadvantage is that new historical data needs to be recorded.

4.3.4 Automating the PdM model

To automate and continuously calculate the RUL and feed it into the maintenance system additional software needs to be implemented around the machine learning algorithm as seen in Fig. 4.13.

4.4 Conclusion

The three lead questions can be answered as follows:

1. How to assess whether the existing data is sufficient for PdM or if additional sensors are required?
    - Historical data of several component failures is needed
    - If no correlation and causality can be found additional features need to be created e.g. by adding additional sensors
2. How to select, train and optimize machine learning algorithms when doing PdM, so it generates a measurable impact?
    - Create a cost function/maintenance policy
    - Train and optimize different machine learning algorithms
    - Choose the algorithm with the lowest costs
3. What needs to be done after a functional algorithm is developed?
    - Automated data import and feeding the maintenance system with the calculated RUL

In the next chapter, the guide is validated in the Digital Capability Center Aachen.

[5] MES = Manufacturing Execution System

[6] PLC = Programmable Logic Controler

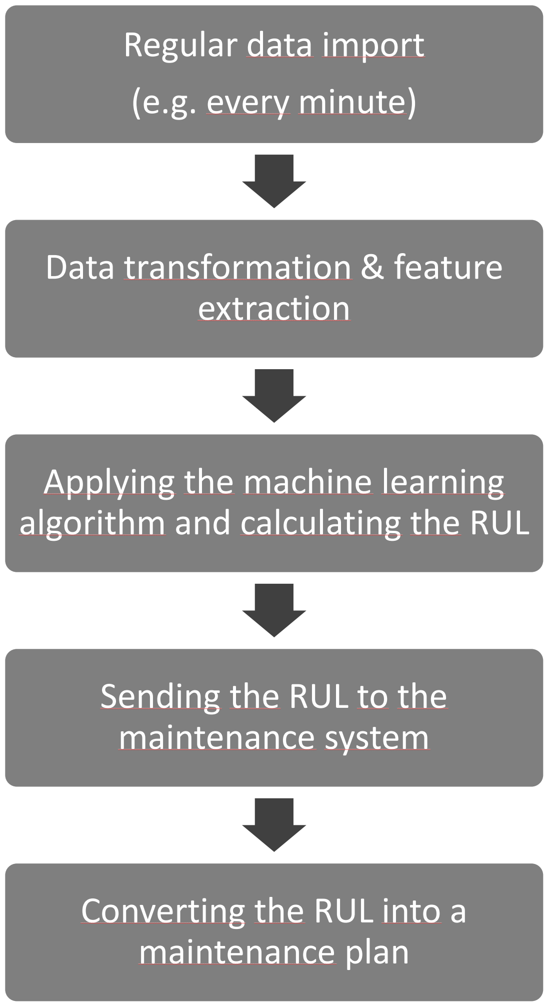

Regular data import
(e.g. every minute)
Data transformation & feature extraction
Applying the machine learning algorithm and calculating the RUL
Sending the RUL to the maintenance system
Converting the RUL into a maintenance plan

Fig. 4.13 Abstract process flow of a PdM implementation

5 Validation of the guide in the Digital Capability Center Aachen

In this chapter, the Digital Capability Center Aachen including its factory line is shortly presented. Then the textile process is analyzed regarding critical machines and components. Finally, PdM is implemented on one component and financially evaluated.

5.1 The Digital Capability Center Aachen (DCC)

To evaluate the financial impact of Industry 4.0 solutions an imaginary company was created by McKinsey and the “Institut für Textiltechnik der RWTH Aachen University”. This company is called GoSmart and is a global high-end textile manufacturer for sports goods, work clothes, and automotive textiles. In the storyline, it is a publicly traded international company with 1.4 billion € revenue and a net profit margin of 6%. Around 3900 FTE are working in 19 factories around the globe and the DCC Aachen is one of these factories in Germany (see Fig. 5.1).

The GoSmart Aachen factory serves the sports goods and automotive textiles product lines and has around 400 FTE and total revenue of 100 million €. In all use cases, the focus is on the wristband production, which includes 1 warping machine, 6 weaving machines, 2 coating machines and 12 assembly lines with printing, sewing, quality check, and packing. On the real shopfloor is only one of these machine types including one assembly line.

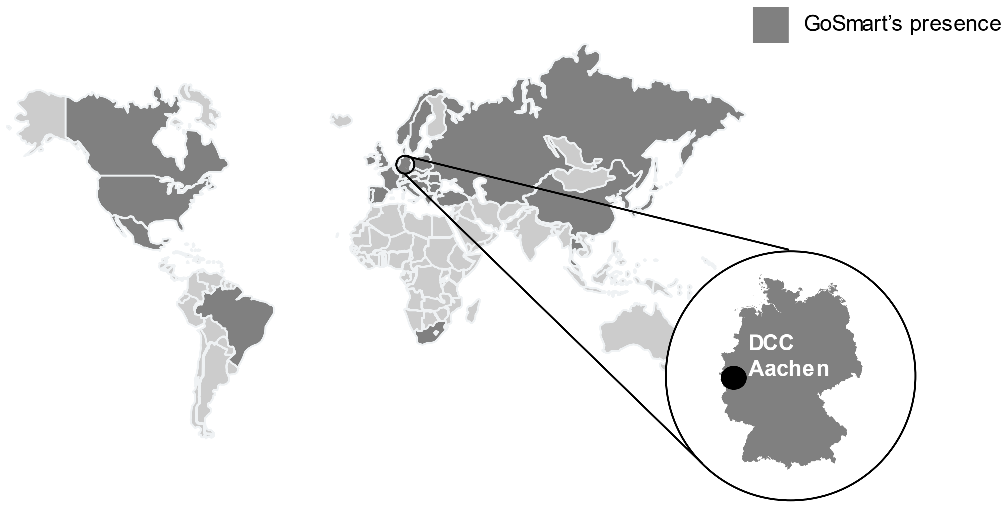


Fig. 5.1 Geographic presence of GoSmart

5.2 Selection of critical machines and components

The first step to integrate PdM is to find the bottleneck. Then it must be analyzed whether the availability of the bottleneck machine is an issue. If this is the case, a FMECA should be conducted for each of the machine components and a cost-efficient maintenance type should be selected. If not, the next bottleneck machine needs to be checked.

Based on the provided business case the warping machine is the bottleneck of the line. The theoretical bottlenecks are the sewing stations, but because of a low availability, the practical bottleneck is the warping machine (see Fig. 5.2).

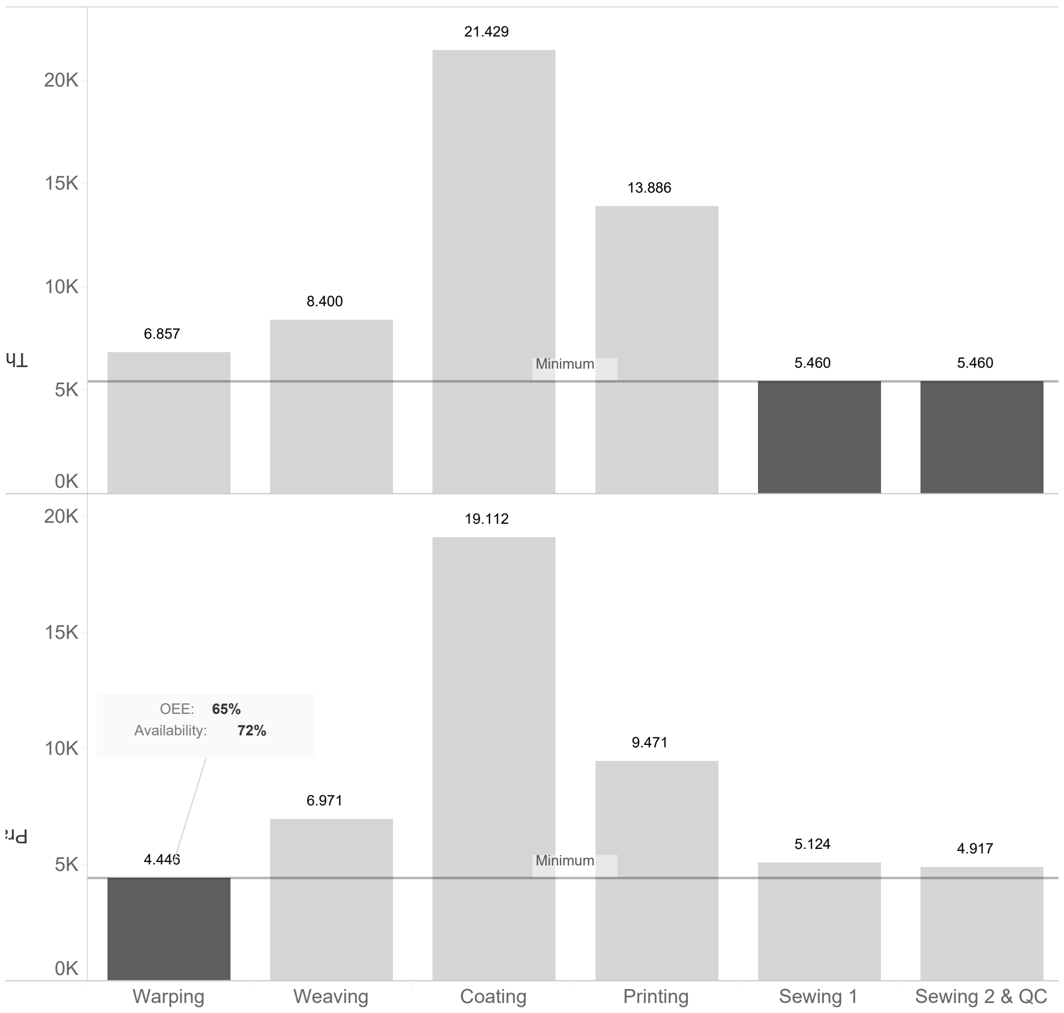


Fig. 5.2 GoSmart Aachen wristband line business case. More information can be found in Attachment A.

A FMECA is conducted on the warping machine to find the most critical components. In total the warping machine can be divided into five functional components and 10 sub-components in total. The results of the first step can be seen in Fig. 5.3 and Fig. 5.4.

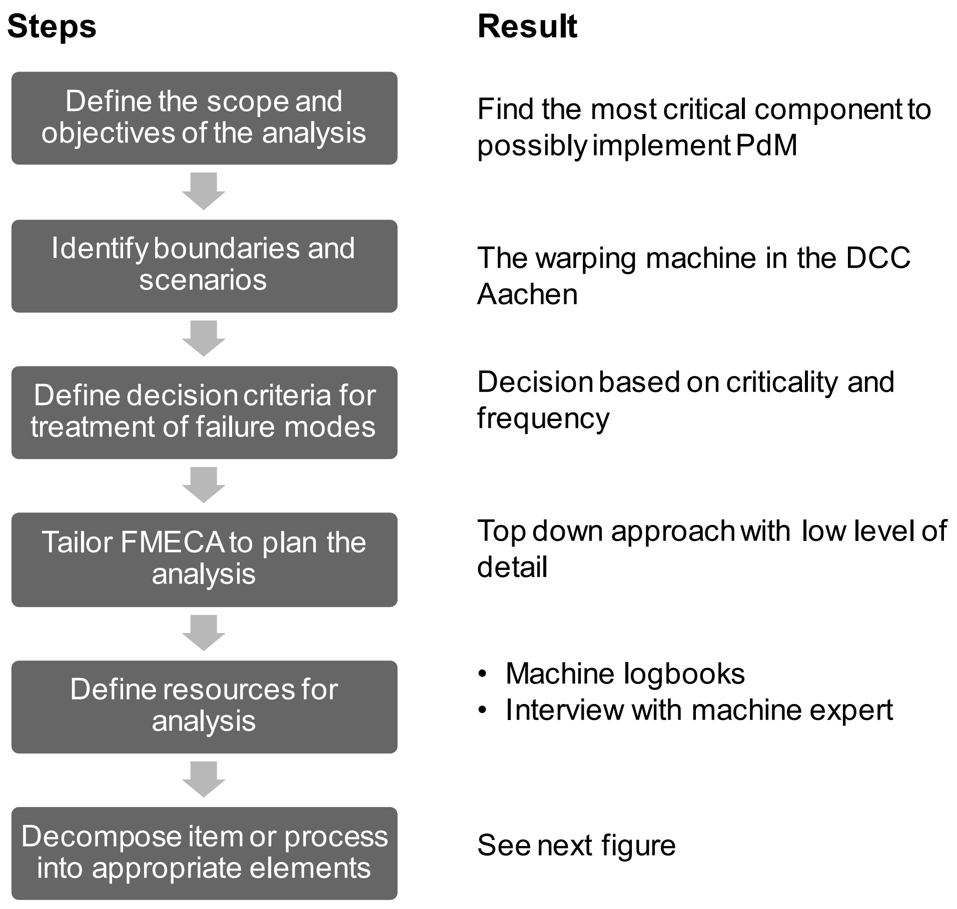

Steps
Result
Define the scope and objectives of the analysis
Find the most critical component to possibly implement PdM
Identify boundaries and scenarios
The warping machine in the DCC Aachen
Define decision criteria for treatment of failure modes
Decision based on criticality and frequency
Tailor FMECA to plan the analysis
Top down approach with low level of detail
Define resources for analysis
• Machine logbooks
• Interview with machine expert
Decompose item or process into appropriate elements
See next figure

Fig. 5.3 Planning the FMECA on the warping machine

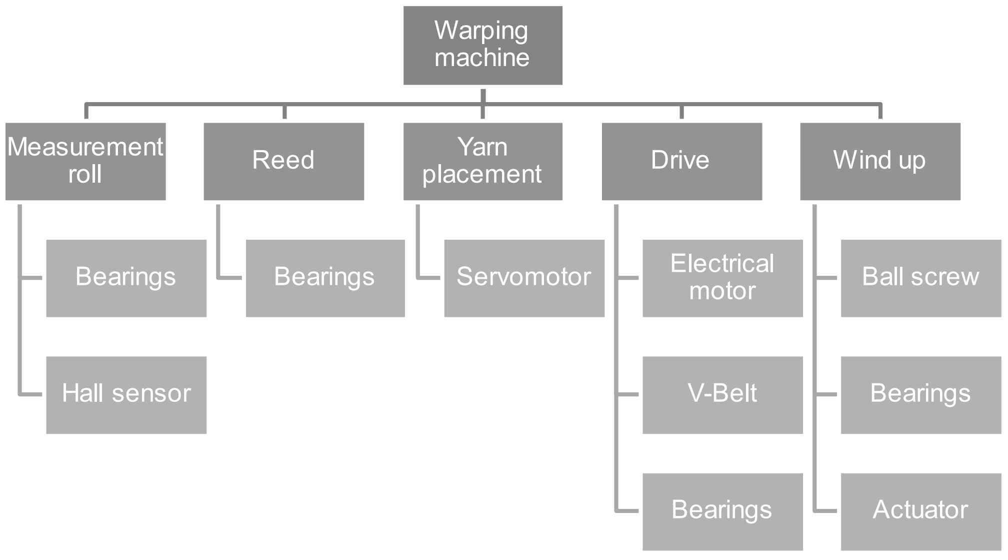
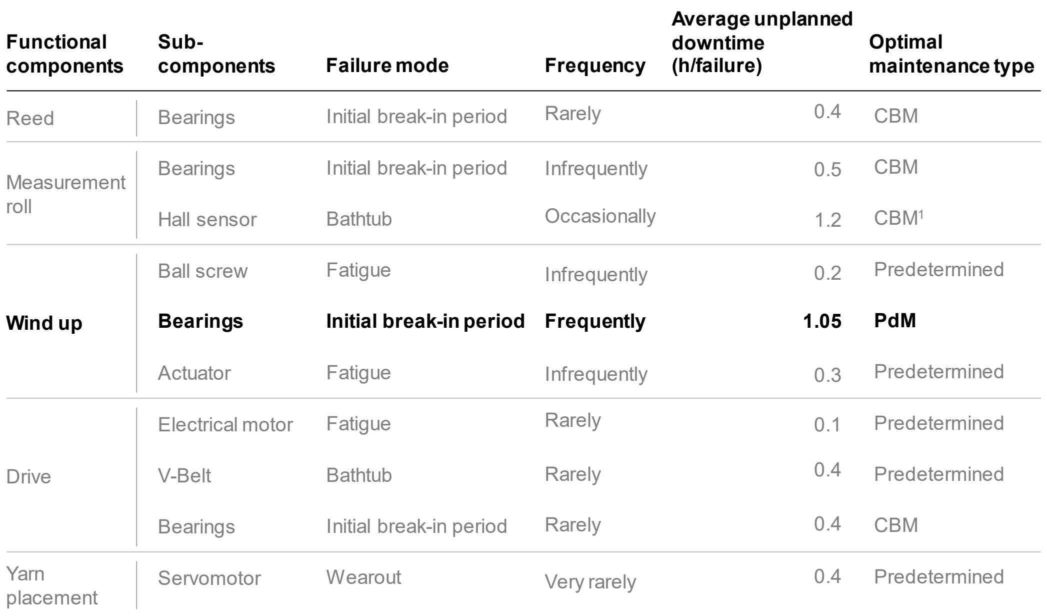


Fig. 5.4 The warping machine can be divided into five functional components and 10 sub-components in total



Functional components	Sub-components	Failure mode	Frequency	Average unplanned downtime (h/failure)	Optimal maintenance type
Reed	Bearings	Initial break-in period	Rarely	0.4	CBM
Measurement roll	Bearings	Initial break-in period	Infrequently	0.5	CBM
	Hall sensor	Bathtub	Occasionally	1.2	CBM[1]
Wind up	Ball screw	Fatigue	Infrequently	0.2	Predetermined
	Bearings	**Initial break-in period**	**Frequently**	**1.05**	**PdM**
	Actuator	Fatigue	Infrequently	0.3	Predetermined
Drive	Electrical motor	Fatigue	Rarely	0.1	Predetermined
	V-Belt	Bathtub	Rarely	0.4	Predetermined
	Bearings	Initial break-in period	Rarely	0.4	CBM
Yarn placement	Servomotor	Wearout	Very rarely	0.4	Predetermined

Fig. 5.5 Analysis of the sub-components and resulting optimal maintenance type. Explanation of frequencies can be found in the footnote[7]

In Fig. 5.5 the sub-components are analyzed and the optimal maintenance type is selected. The average unplanned downtime is based on historical data and used as criticality, as there are no life-threatening failures and the biggest financial impact is the time due to breakdown. For the Hall sensor, CBM is used instead of predetermined maintenance to refine the time interval (assumption: time interval provided by the manufacturer is too low). Because the bearings of the wind up are failing frequently and have the highest unplanned downtime, PdM is implemented there.

5.3 Systematic implementation of Predictive Maintenance on critical components

In this chapter, PdM is implemented on the bearings of the wind up. As approach, a data-driven model is chosen, as it requires no domain knowledge compared to physical

[7] Frequently: between once per week and once per month; Occasionally: between once per month and once every 6 months; Infrequently: between once every 6 months and once every year; Rarely: between once every year and once every 2 years; Very rarely: between once every 2 years and once every 4 years; Nearly never: between once every 4 years and once every 8 years

model-based methodology and knowledge-based models. To achieve the highest precision the regression algorithms are chosen over the classification approach.

The final cost function for the wind-up bearing can be seen in Fig. 5.6 with the following input parameters derived from the GoSmart business case in Attachment A:

- Breakdown costs: $10.36\ €/band \times 5460\ bands/shift\ /\ 8h/shift = 7070{,}7\ €/h$
- Spare part costs: 300 € (estimation)
- Exchange labor: $165\ €\ /\ day \times 0.5\ day = 82.5\ €$
- Lead time: 4 days
- Buffer time: 2 days

The Python code to generate the cost function can be found in Attachment B.

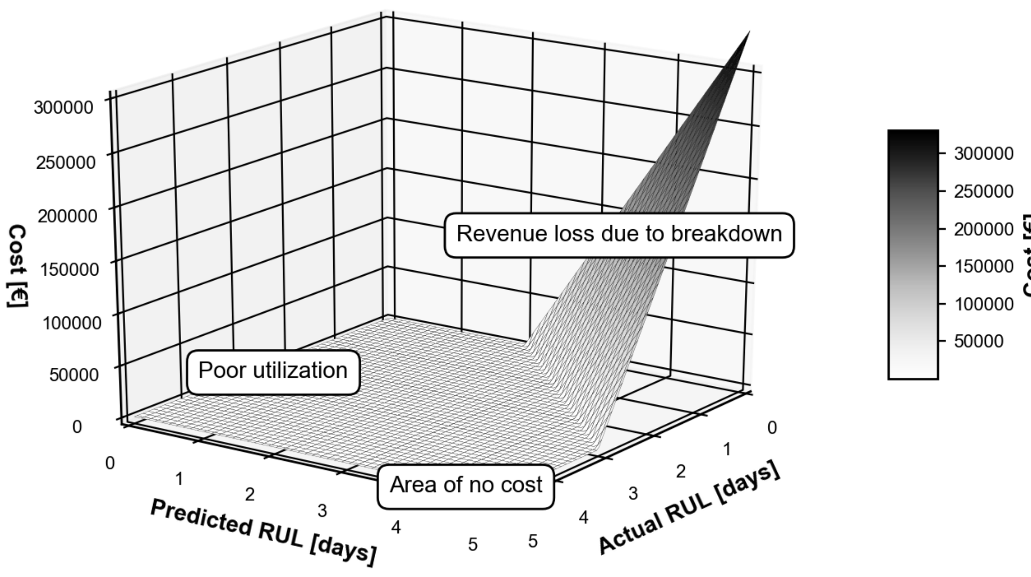


Fig. 5.6 Final cost function for the wind-up bearing

To calculate the net present value of the alternative investment, predetermined maintenance, it is assumed that the bearings fail accordingly to Fig. 5.7. Therefore, the 10% exchange rate is at 6.44×10^6 revolutions[8], which results in 14.23 component failures per year[9].

[8] Extracted out of Fig. 5.7

[9] Assuming 16 hours per day and 300 days per year, average machine speed of 400 m/min and average diameter of warp beam of 0.4m

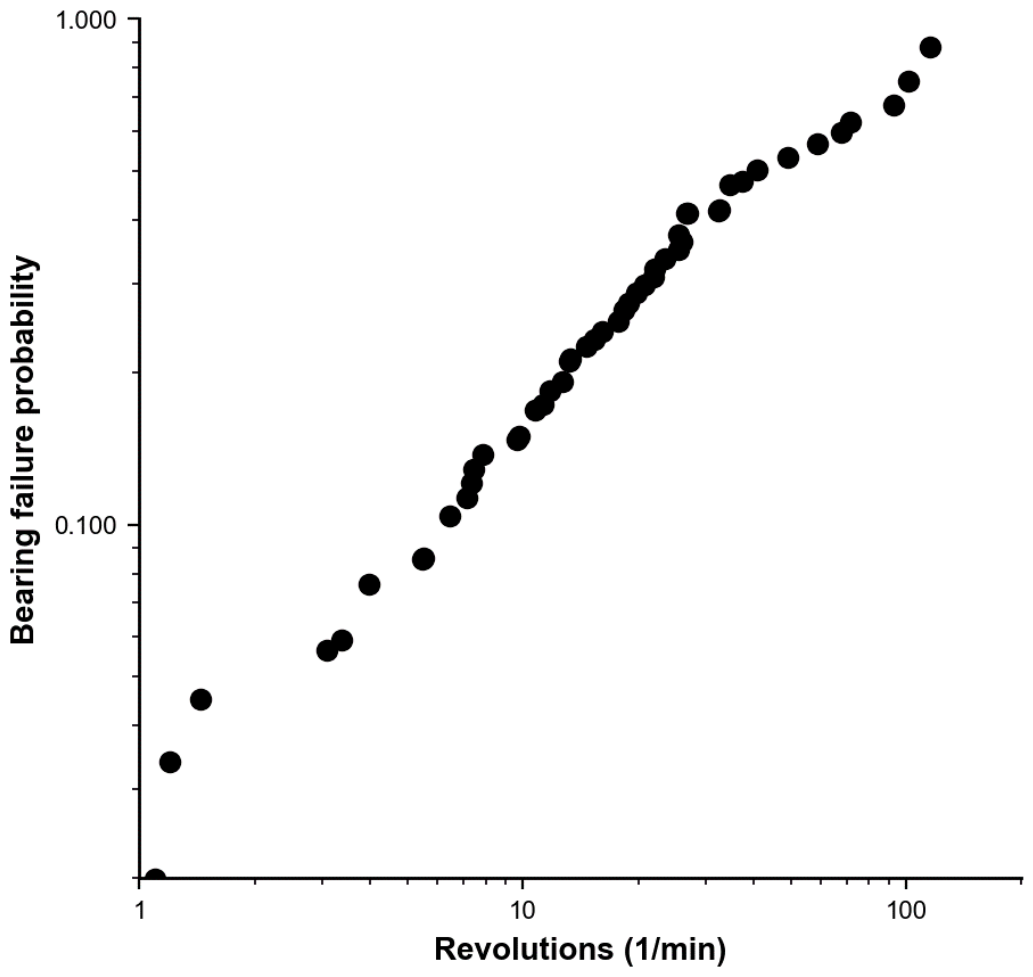


Fig. 5.7 Failure rate of 50 bearings, type unknown, in double-logarithmic diagram based on [Esc64]

The implementation costs of 10330€ for predetermined maintenance consists out of:

- 10000€ for a maintenance system to keep track of the RUL (assumption)
- 330€ labor costs (assuming 2 days 1 FTE of 165€/day)

Based on the historical data for in Fig. 5.7, the cost function in Fig. 5.6 and the implementation costs of 10330€, this results in net present value over three years of -3,012,090.67€[10].

The net present value of a PdM implementation over three years must then be higher than the net present value of predetermined maintenance to be viable.

[10] Assuming constant interest rate of 0.32%. Annual costs consist out of $14.23\, failures\ per\ year\ \times\ 70741.07€\, per\ failure$. More information can be found in attachment E

The implementation costs for PdM are with 81800€ higher than for predetermined maintenance:

- 15000€ for 15 days 1 FTE data scientist for model generation a 1000€/day
- 1300€ for 2 days 1 FTE installation technician a 650€/day
- 14000€ for 7 days 2 FTE software developer a 1000€/day
- 50000€ for an IoT / analytics platform (estimation)
- 1500€ hardware costs (sensor and PLC)

Historical data of the wind-up bearing was not available and would go beyond the scope of this bachelor thesis, so a bearing failure was simulated instead by adding different weights to the warp beam to generate an unbalance (see also Fig. 5.8).

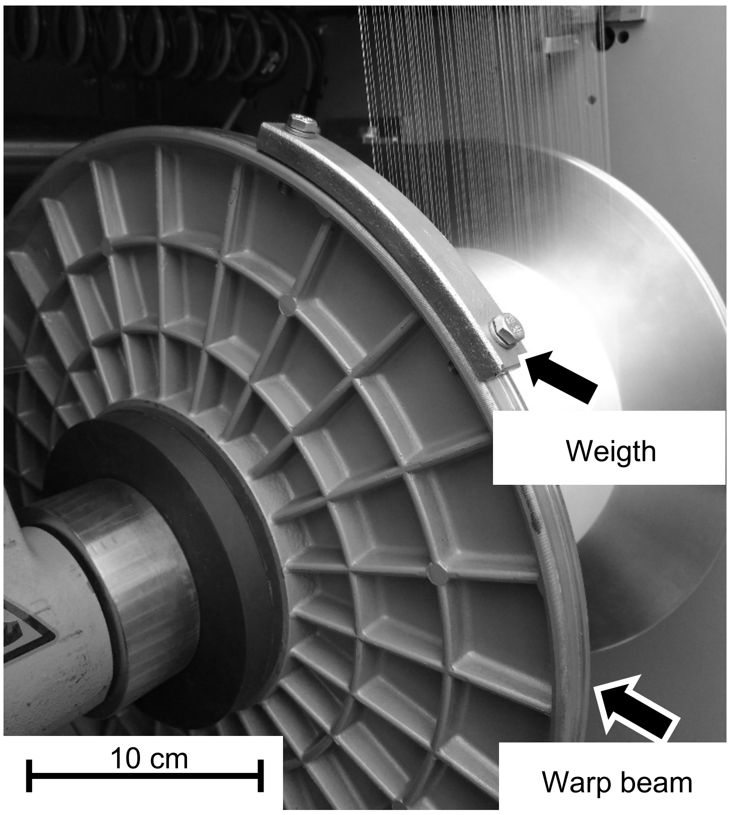


Fig. 5.8 Additional mounted weights on the warp beam

Before the experiment, three different data sources were added to the factory database (OSIsoft PI system) and exported as a .csv (see also Fig. 5.9).

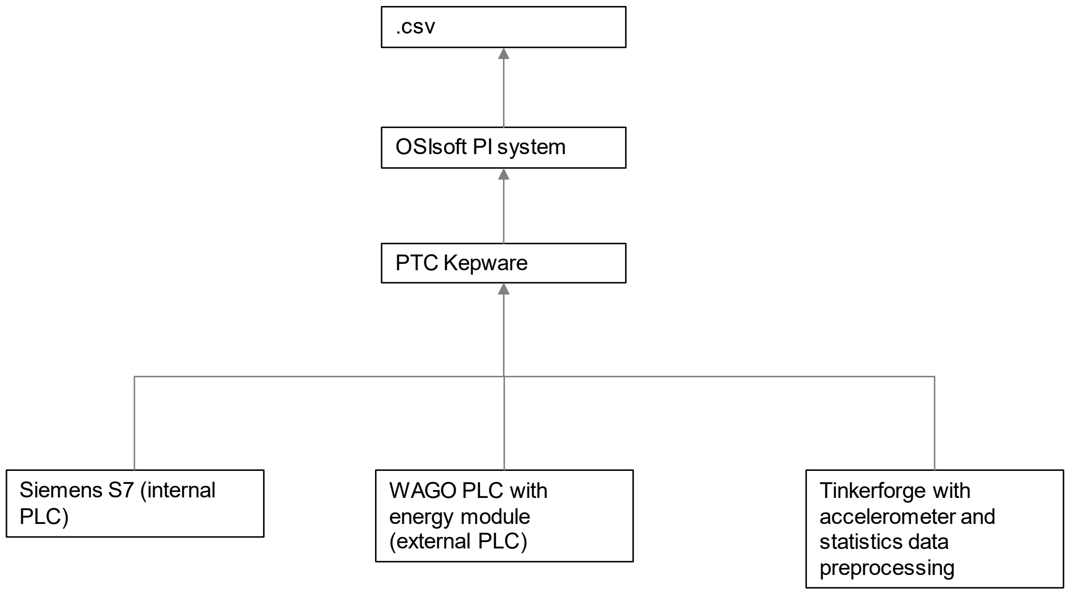

.csv
OSIsoft PI system
PTC Kepware
Siemens S7 (internal PLC)
WAGO PLC with energy module (external PLC)
Tinkerforge with accelerometer and statistics data preprocessing

Fig. 5.9 Data sources connected to the factory database (OSIsoft PI system) and exported into .csv format

The Siemens S7 is the internal PLC of the warping machine and provides amongst others the following machine parameters:

- Machine speed (m/min)
- Amount of produced material (m)

The Wago PLC with an energy module is an external PLC and connected to the warping machine's power supply to provide e.g.

- three phase energy consumption (kWh)
- the voltage of each phase (V)
- the current of each phase (A)

The main idea was that an unbalance in the bearing would cause different vibrations compared to a fully working bearing [OAC06]. Therefore, the Tinkerforge pre-processes the data coming from the accelerometer on the wind-up bearing to generate in seconds intervals features like:

- Statistical analysis for each axis (average over the last second, min/max, etc.)
- Top five frequencies including amplitude (via Fast-Fourier-Transformation).

A Fourier Transformation (FT) is a mathematical method to split a signal into a series of sinus and cosine functions. The Discrete-Fourier-Transformation (DFT) is using the FT to convert a signal from the time to the frequency domain. The Fast-Fourier-Transformation (FFT) is a DFT with improved execution speed. [But12]

PTC Kepware is a middleware and extracts the data by “translating” the various industry protocols of PLCs (e.g. Siemens S7) to a more general protocol called OPC UA[11].

The OSIsoft PI system then accesses the data provided via OPC UA and stores it into its time series database[12]. The data is then exported in minute intervals into .csv[13] files.

During the first experiment, different weights and machine speeds were used to generate data for the holdout method (see Fig. 5.10).

Parameters adjusted: / Data type	Weigth	Machine speed for each weigth	Summary
Training data:	1. 0g (without any weigths) 2. 100g 3. 200g 4. 300g 5. 400g 6. 500g	1. 100 m/min 2. 200 m/min	The machine was operated for each weigth and machine speed for around 800m
Validation data:	1. 0g (without any weigths) 2. 200g	Randomly choosen between 100 and 200 m/min	The machine was operated 5 minutes in total for the validation data

Fig. 5.10 Table of weights and machine speeds used in the first experiment

The recorded data was transferred from the PI system into a Python script and cleaned, transformed and scaled (see Fig. 5.11)

During the **cleaning step,** unnecessary data was removed (see also attachment C), e.g.

- Dropping of irrelevant or misleading machine parameters with domain knowledge (e.g. temperature inside of the machine)
- Dropping values where the machine was not running
- Dropping values with negative machine speed (caused by resetting the length at the machine)

[11] OPC UA = Open Platform Communications United Architecture

[12] A time series database is a database optimized to store over a long period of time time series data.

[13] CSV = Comma-Separated Values

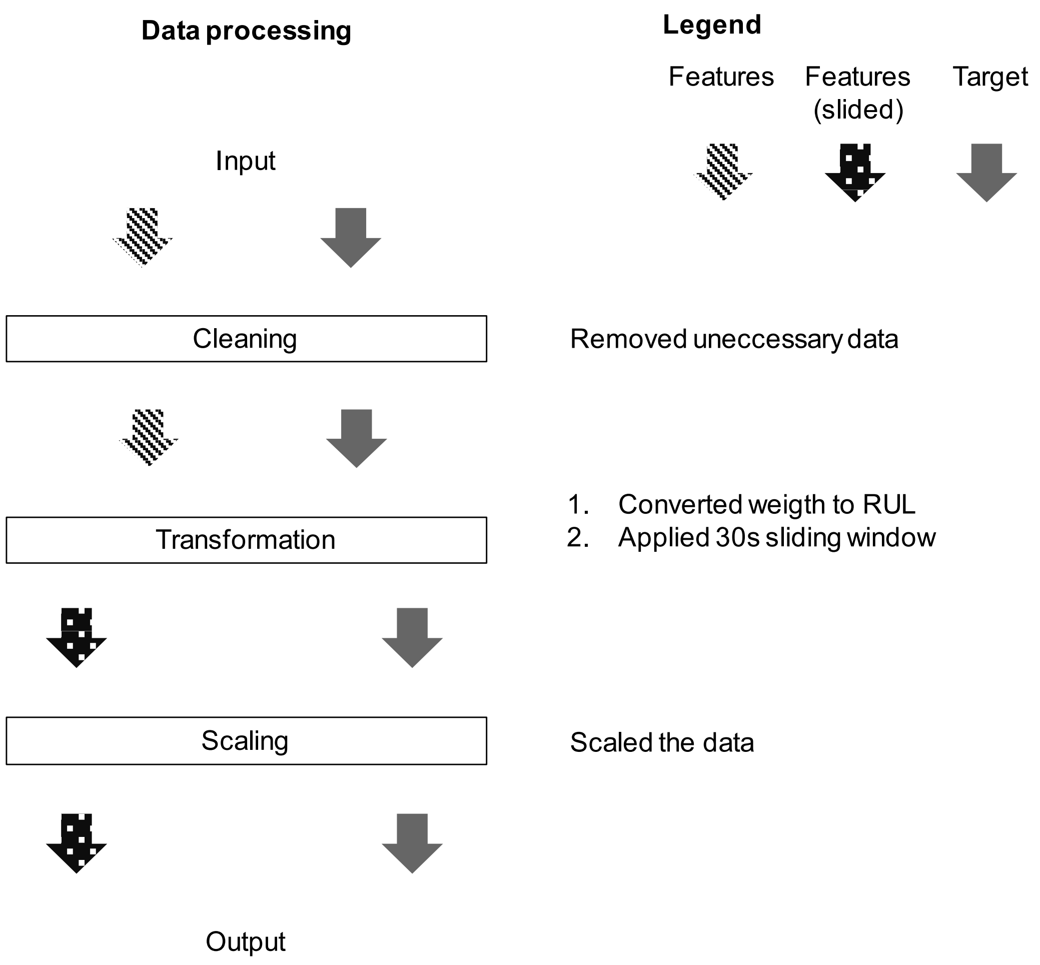


Fig. 5.11 Data pre-processing

During the **transformation step** (see also attachment D) the weight was converted into a RUL with 0g being 50 days, 500g being 0 days and everything in between interpolated with a root function. The features were transformed using 30 seconds sliding window.

In the **scaling step** (see also attachment D) the features and targets were normalized, which is required for a lot of machine learning algorithms.

In the Python script, several algorithms were tested after pre-processing the data. On the training set, the algorithms perform well and all net current values are positive, so all algorithms are better than predetermined maintenance (see Fig. 5.12). But on the validation dataset the algorithms fail (see Fig. 5.13). Although the net present value is positive the algorithms seem to be overfit to the training data and neglecting the real correlation. Even after generating and adding additional training data the algorithms still fail (see Fig. 5.14).

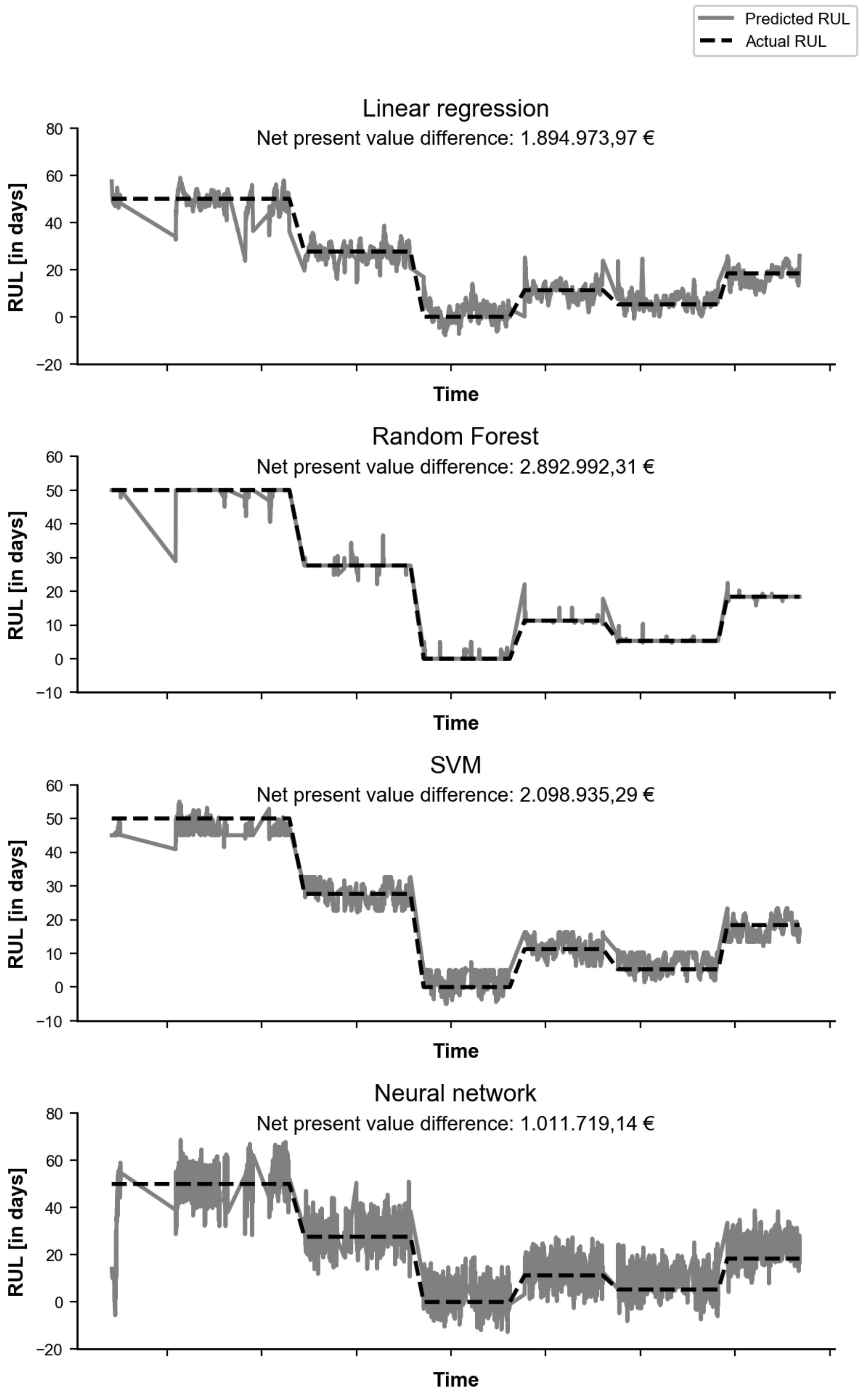


Fig. 5.12 Machine learning algorithms executed on the training dataset. The predicted RUL roughly matches the actual RUL

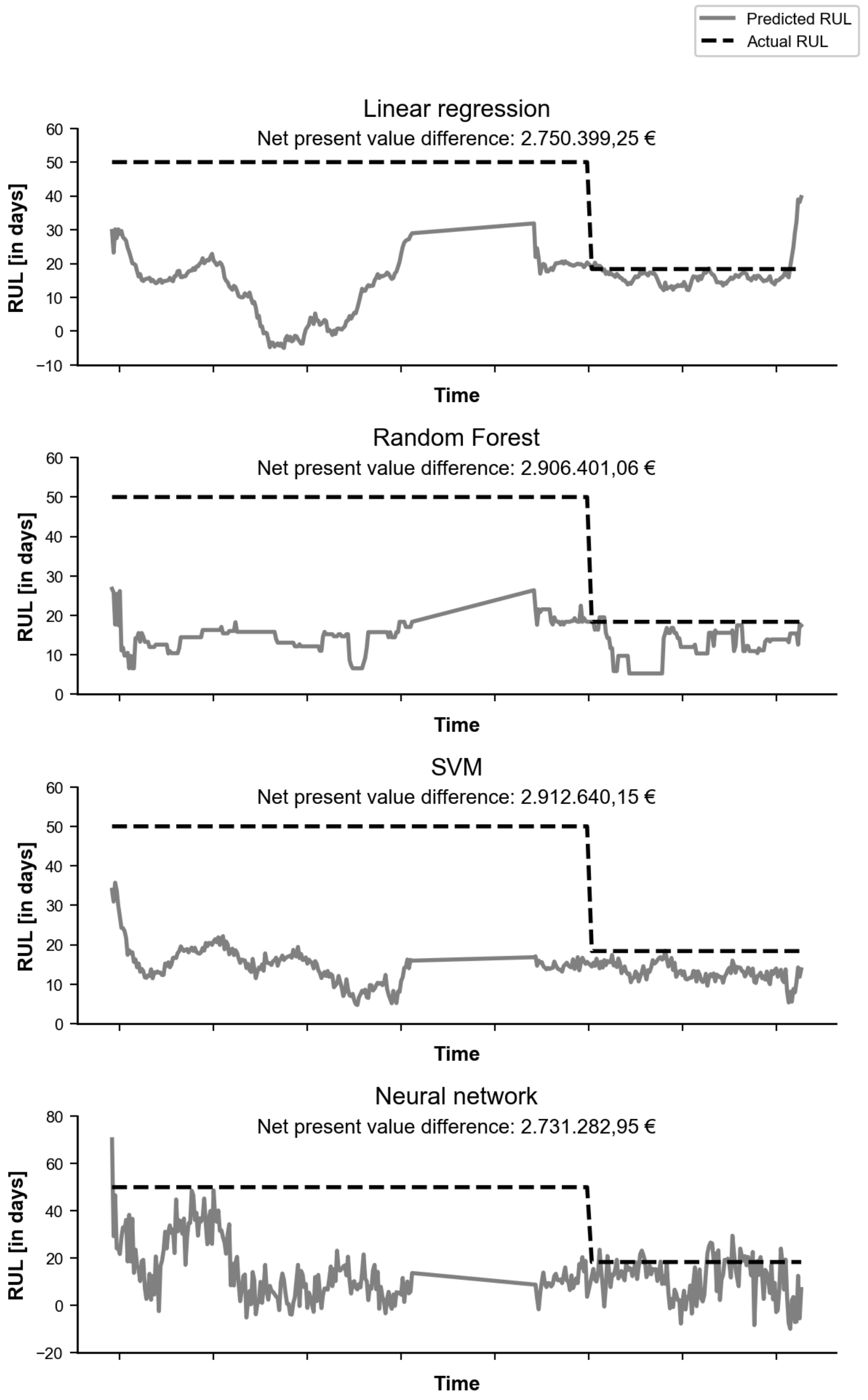


Fig. 5.13 Machine learning algorithms executed on the validation data set. The predicted RUL does not match the actual RUL

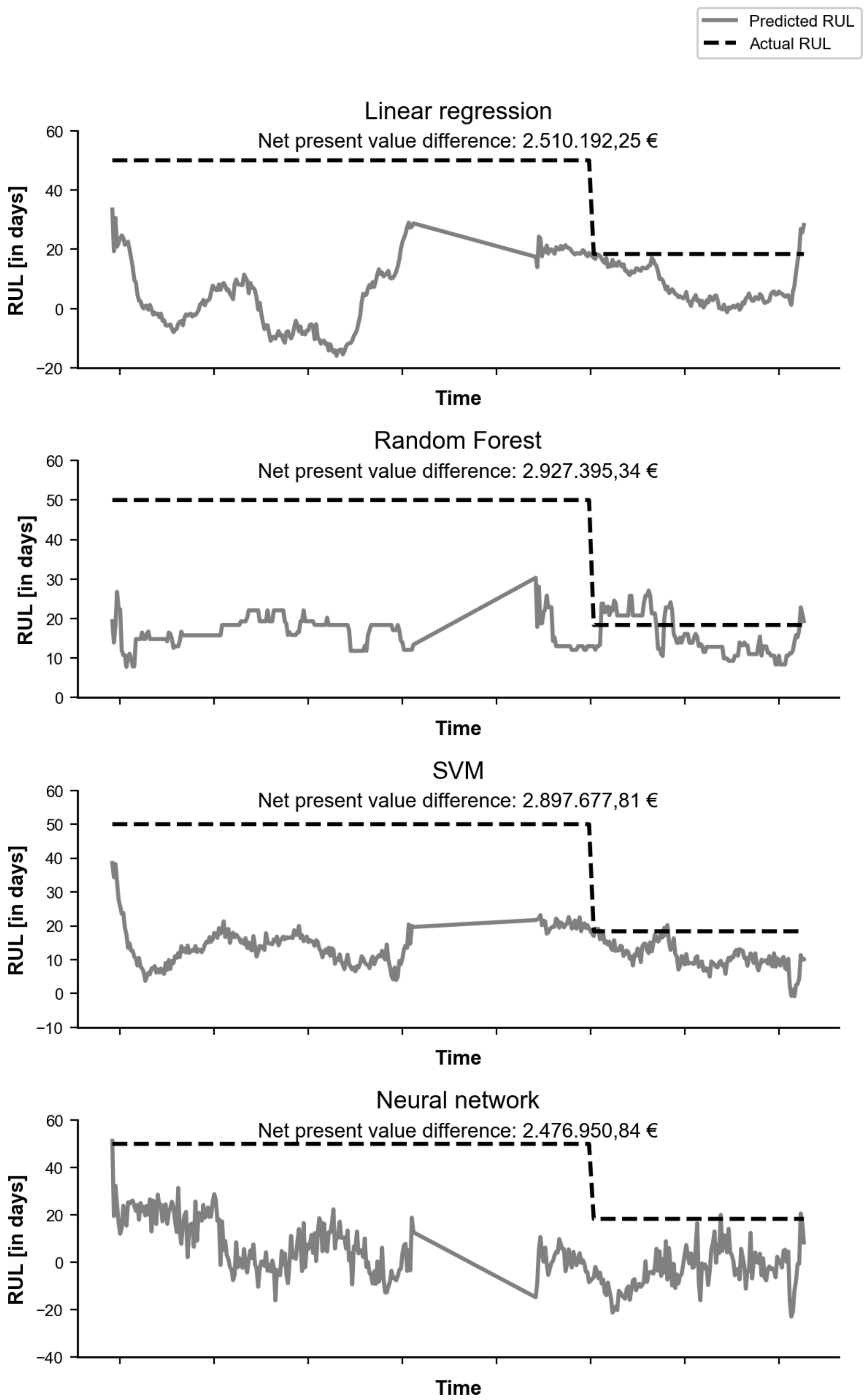
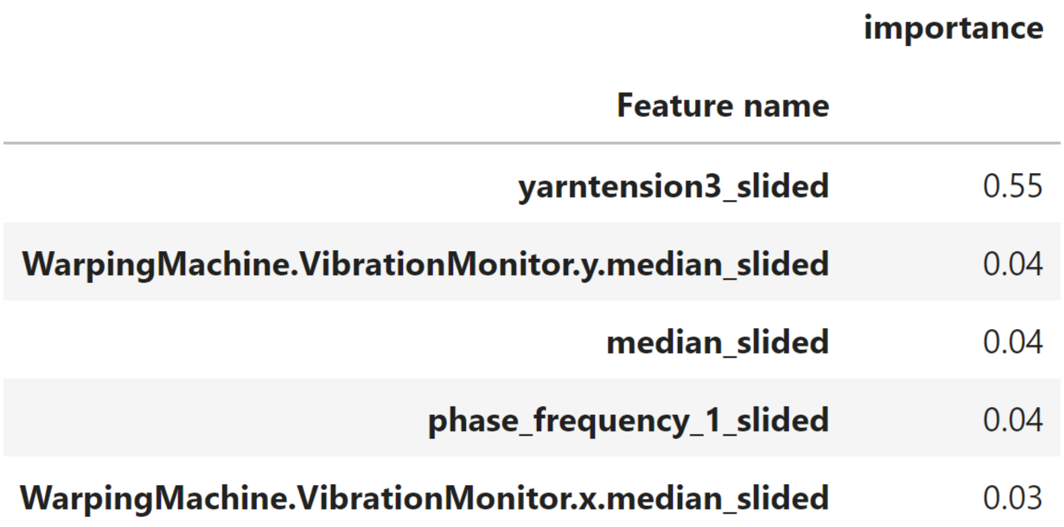


Fig. 5.14 Machine learning algorithms trained with additional data executed on the validation data set. The predicted RUL does not match the actual RUL.

After this additional analysis of the data it is clear that with the current sensors a successful data-driven model cannot be created. Several algorithms, for example, random forests or linear regression, can show their most important parameters and these parameters do not correlate with the wear state.

The top 5 most important features of the random forest algorithm can be seen in Fig. 5.15. The boxplot for yarntension3_slided, the most important feature, can be seen in Fig. 5.16 (see also attachment D) and yarntension3_slided changes in the training dataset based on the weight but stays almost constant in the validation dataset. Possible reasons for this could be for example not measured external factors, that caused the yarn tension to change, or that yarntension3_slided just correlated accidentally with the RUL without a cause.



	importance
Feature name	
yarntension3_slided	0.55
WarpingMachine.VibrationMonitor.y.median_slided	0.04
median_slided	0.04
phase_frequency_1_slided	0.04
WarpingMachine.VibrationMonitor.x.median_slided	0.03

Fig. 5.15 The "importance" of features in the random forest algorithm

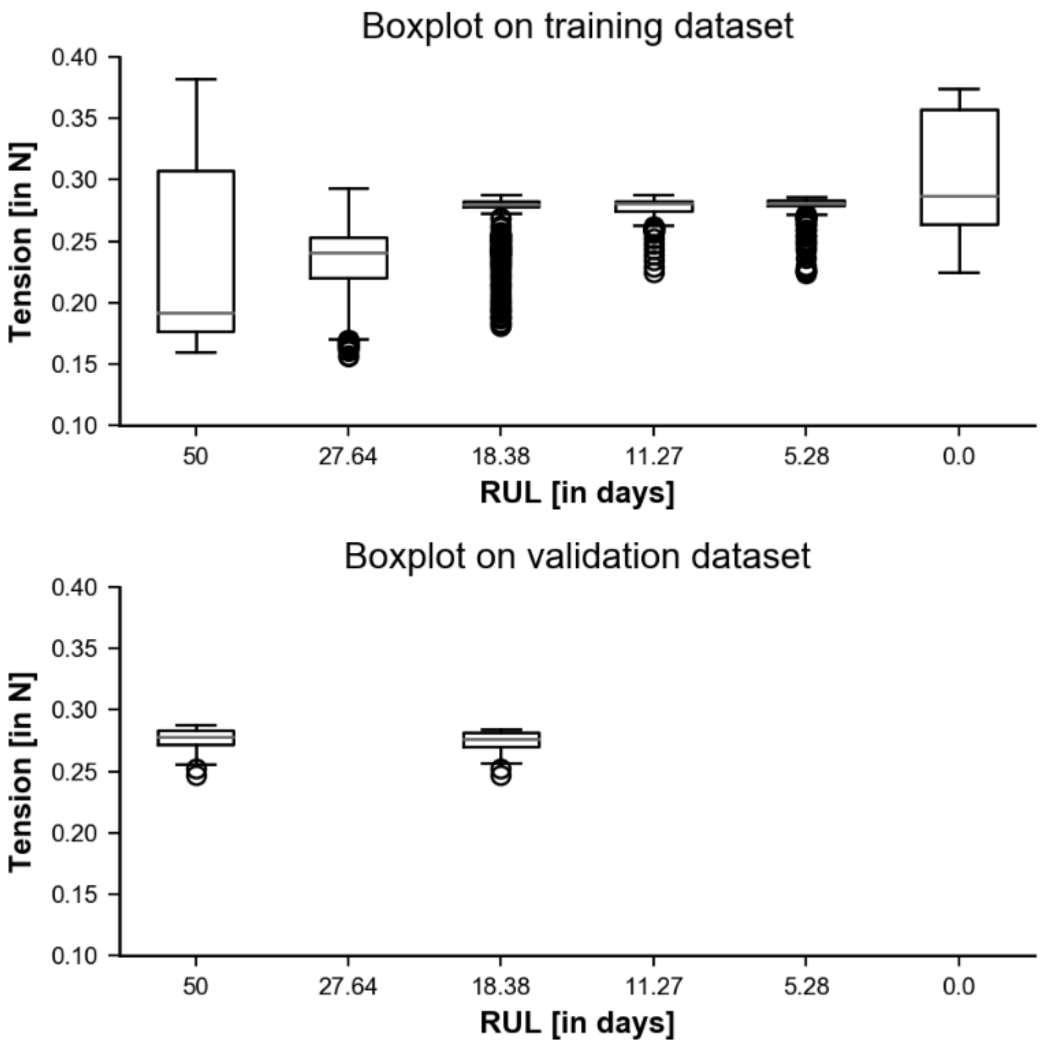
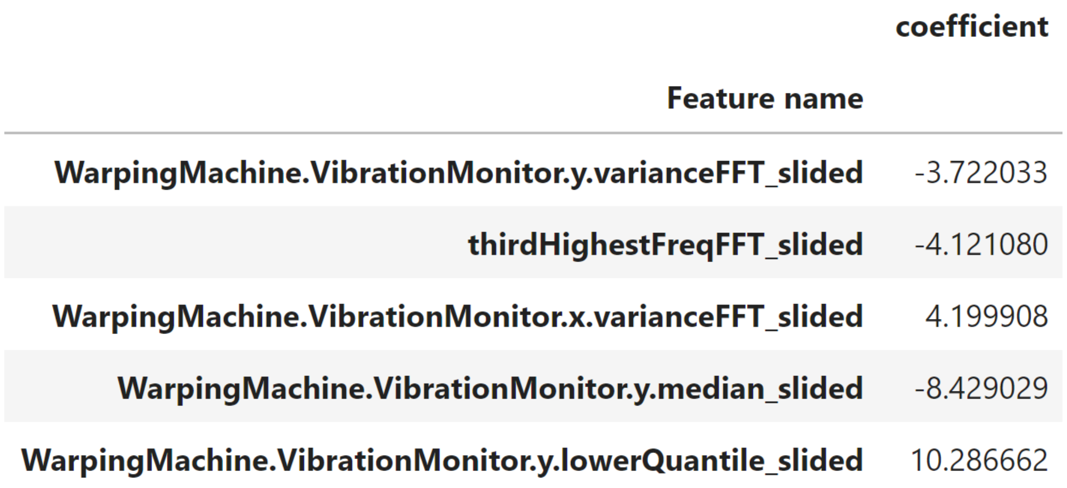


Fig. 5.16 No correlation between training and validation dataset for yarntension3_slided

The same reasons can be the cause that the features with the highest absolute coefficient for the Linear Regression algorithm (for most important one see Fig. 5.17) do not correlate with the RUL. The average acceleration stays almost entirely constant at around -1 m/s^2 except for "RUL 50 days" in the training data set, which is scattered between 0 and -1 m/s^2. Therefore, the correlation is very likely to be random without causality.



	coefficient
Feature name	
WarpingMachine.VibrationMonitor.y.varianceFFT_slided	-3.722033
thirdHighestFreqFFT_slided	-4.121080
WarpingMachine.VibrationMonitor.x.varianceFFT_slided	4.199908
WarpingMachine.VibrationMonitor.y.median_slided	-8.429029
WarpingMachine.VibrationMonitor.y.lowerQuantile_slided	10.286662

Fig. 5.17 The most "important" features according to the linear regression algorithm based on the absolute value of the coefficient

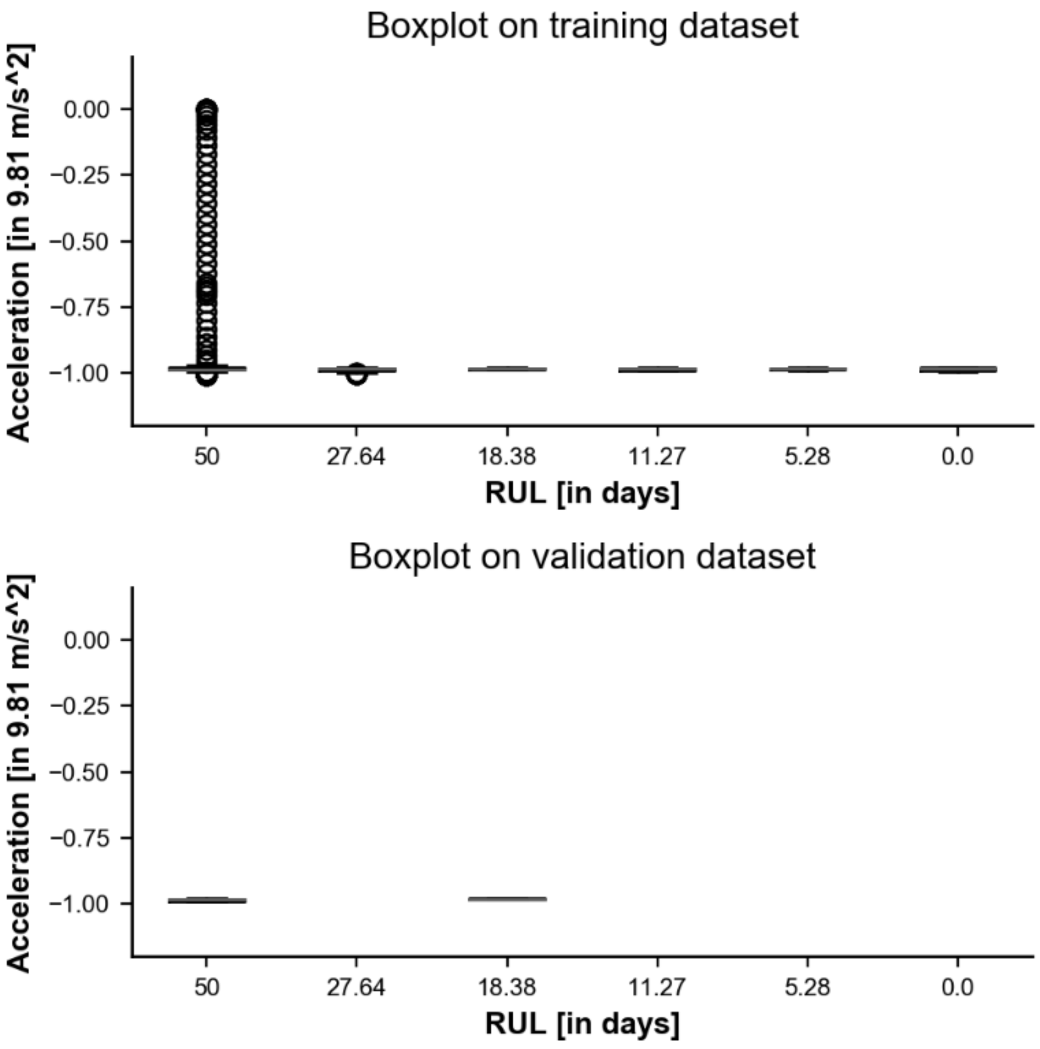


Fig. 5.18 No correlation between RUL and WarpingMachine.VibrationMonitor.y.lower-Quantile_slided

The next steps are to try out Physical-Based Models or Knowledge-Based Models or adding more sensors / features. Because of limited time of the bachelor thesis, the additional methods need to be investigated in future scientific works.

If a functioning and financial impactful algorithm can be created, the implementation could look like this:

The .csv files can be read, processed and deleted automatically every minute in one Python script. The deletion process is necessary to save disk storage (data remains stored in the PI system). The Python script needs to be restarted during a system restart or a failure/bug inside the program. In UNIX-like operating systems (e.g. Linux or macOS) this can be done by using supervisord [AG18].

The remaining useful lifetime can then be sent via REST API to Thingworx, where the maintenance planner's dashboard as seen in Fig. 5.19 is created. Thingworx is an IoT platform suited for rapid application development and offers a wide variety of APIs and connectors[14]. In the DCC maintenance case, it is also connected with the maintenance system and forwards to RUL to the maintenance system and gets the maintenance plan in return. This has already been done. On top of that platform, a role-based application exists to give the maintenance planner all the information he needs in a single dashboard and in this dashboard the RUL can be seen together with the maintenance plan.

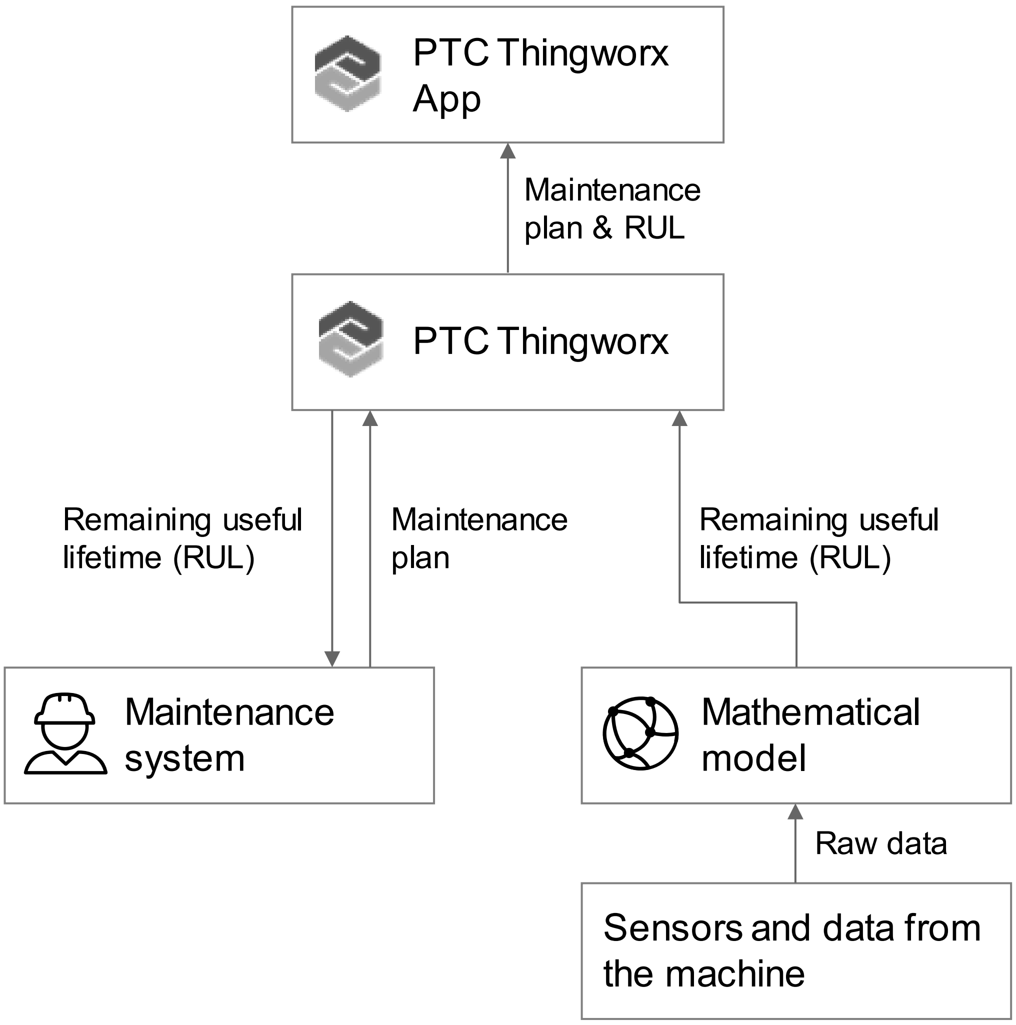
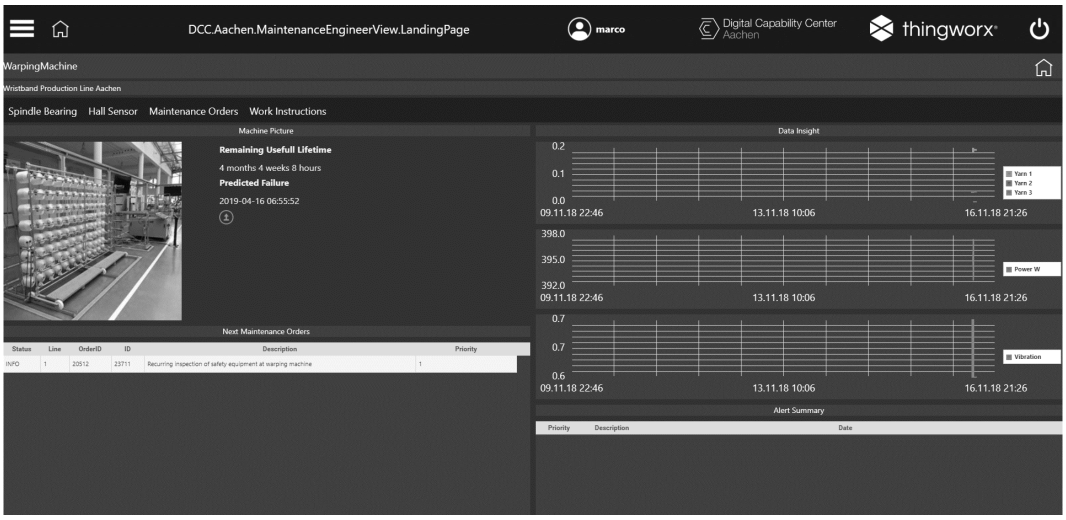


Fig. 5.19 Integration of the created algorithm into the maintenance system

[14] Based on the authors experience with working with Thingworx. Detailed information can be found under thingworx.com



Remaining Usefull Lifetime

4 months 4 weeks 8 hours

Predicted Failure

2019-04-16 06:55:52

Next Maintenance Orders

Status	Line	OrderID	ID	Description	Priority
INFO	1	20512	23711	Recurring inspection of safety equipment at warping machine	1

Fig. 5.20 PTC Thingworx dashboard for the maintenance planner

6 Summary and outlook

At the beginning of this bachelor thesis, the state of the art is explained giving an overview of different maintenance strategies and types, different machine learning types and algorithms, and the use of PdM in practice is analyzed. As companies are still struggling with implementing PdM it is clear, that there is a need for a guide that covers the entire implementation process from finding the machines and components with the highest financial impact to the technical implementation.

The economically viable selection of critical machines and components is explained in the third chapter of this thesis using first bottleneck detection methods to find the critical machines and then a Failure mode, effects and criticality analysis (FMECA) to identify critical components. Furthermore, the applicability of different maintenance types is discussed.

The fourth part of the guide is about selecting the type of PdM model (data-driven models, knowledge-based models, etc.) and correct level of detail (classification or regression) and evaluating the financial impact for classification as well as regression algorithms by using a dynamic investment approach and comparing it with predetermined maintenance. Finally, several methods for selecting and optimizing algorithms for the highest impact are explained.

The finished guide is then validated in the Digital Capability Center Aachen (DCC Aachen), an Industry 4.0 model factory. The textile process is analyzed, and the warping machine identified as a bottleneck. For all components, a suitable maintenance type is chosen. From this analysis, the wind-up bearing is best suited for a PdM implementation. A data-driven PdM model is chosen as an in-depth technical bearing analysis is not in the scope of this bachelor thesis (and would be required if choosing other types of models). After choosing regression algorithms and creating a cost function data failure data are generated by adding additional weights to the warp beam, as historical data is not available. Unfortunately, a failure could not be detected using a data-driven model and the currently existing sensors and their features. Therefore, if a company wants to implement PdM on the wind-up bearings it would need to try a different approach, use more (or better) sensors or additional methods to generate features, e. g. wavelet transformation.

Not included in this methodology is the required change management, so that the operators or maintenance engineers use the PdM technology, or a validation of the methodology (and especially the assumptions in the economic analysis) in a company. Furthermore, the methodology needs to be very likely adjusted in practice due to missing data or very complex processes. Therefore, additional scientific work is required to validate and further develop the methodology in practice.

7 Publication bibliography

[AG18]	Amruthnath, Nagdev; Gupta, Tarun A Research Study on Unsupervised Machine Learning Algorithms for Early Fault Detection in Predictive Maintenance: 5th International Conference on Industrial Engineering and Applications, 2018
[AK08]	Ahuja, I.P.S.; Khamba, J. S. Total productive maintenance: literature review and directions International Journal of Quality & Reliability Management. Vol.25 (2008) 7, p. 709–756
[All05]	Allen, Timothy U.S. Navy Analysis of Submarine Maintenance Data and the Development of Age and Reliability Profiles, 2005, https://pdfs.semanticscholar.org/27c8/71b845f0c05fee98d018da6346452a081dd9.pdf, Accessed on 25.03.2018
[Axe17]	Axel, Ericson Bottleneck detection in manufacturing - Gothenburg, Sweden Chalmers University of Technology, Master thesis
[Ayo10]	Ayodele, Taiwo Oladipupo Types of Machine Learning Algorithms. In Zhang, J.: New Advances in Machine Learning: In-Tech, 2010
[Bre12]	Breuer, Wolfgang Investition I 4. ed. - Wiesbaden: Gabler Verlag / Springer Fachmedien Wiesbaden GmbH Wiesbaden, 2012
[But12]	Butz, Tilman Fouriertransformation für Fußgänger 7. ed. - Wiesbaden: Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH Wiesbaden, 2012
[CBK09]	Chandola, Varun; Banerjee, Arindam; Kumar, Vipin Anomaly detection ACM Computing Surveys. Vol.41 (2009) 3, p. 1–58

[CM15]	Claesen, Marc; Moor, Bart De Hyperparameter Search in Machine Learning, 07.02.2015, http://arxiv.org/pdf/1502.02127v2
[CSG12]	Chen, Daqing; Sain, Sai Laing; Guo, Kun Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining Journal of Database Marketing & Customer Strategy Management. Vol.19 (2012) 3, p. 197–208
[DBG17]	Duschek, Frank; Blameuser, Ralf; Gehrmann, Sven Maschinenverügbarkeit rauf, Wartungs- und Servicekosten runter, 2017, https://www.bearingpoint.com/de-de/downloadformular/?item=9467&mo-dule=476573, Accessed on 27.02.2018 [Company publication]
[DIN 13306]	DIN 13306:2017 Maintenance – Maintenance terminology.
[DIN 60812]	DIN 60812:2015-08 Fehlzustandsart- und -auswirkungsanalyse (FMEA).
[DK17]	Dheeru, Dua; Karra Taniskidou, Efi: UCI Machine Learning Repository, 2017, http://archive.ics.uci.edu/ml
[Esc64]	Eschmann, Paul Das Leistungsvermögen der Wälzlager - Berlin, Heidelberg: Springer Berlin Heidelberg, 1964
[Faw06]	Fawcett, Tom An introduction to ROC analysis Pattern Recognition Letters. Vol.27 (2006) 8, p. 861–874
[GBC16]	Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron Deep Learning: MIT Press, 2016
[Gil17]	Gilabert, Eduardo; Fernandez, Santiago; Arnaiz, Aitor; Konde, Egoitz Simulation of predictive maintenance strategies for cost-effectiveness analysis Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. Vol.231 (2017) 13, p. 2242–2250
[GS04]	Girdhar, Paresh; Scheffer, Cornelius Practical Machinery Vibration Analysis and Predictive Maintenance: Elsevier, 2004

[HB11] Hashemian, H. M.; Bean, Wendell C.
State-of-the-Art Predictive Maintenance Techniques
IEEE Transactions on Instrumentation and Measurement. Vol.60 (2011) 10, p. 3480–3492

[Idh18] Idhammar, Christer: Can you really Justify Reliability Centered Maintenance (RCM) - Part II, 2018, https://www.idcon.com/resource-library/articles/reliability-centered-maintenance/458-can-you-really-justify-rcm-2.html, Accessed on 08.03.2018

[Jah15] Jahnke, Patrick
Machine Learning Approaches for Failure Type Detection and Predictive Maintenance - Darmstadt Technische Universität Darmstadt, Master Thesis

[JD88] Jain, Anil; Dubes, Richard
Algorithms For Clustering Data - New Jersey: Prentice Hall, 1988

[Mac18] Machine Learning Group at the University of Waikato: Weka 3 - Data Mining with Open Source Machine Learning Software in Java, 2018, https://www.cs.waikato.ac.nz/ml/images/Weka%20(software)%20logo.png, Accessed on 25.07.2018

[Mic17] Microsoft (Publisher):
How to choose algorithms for Microsoft Azure Machine Learning, 2017, https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice, Accessed on 04.03.2018

[Mit06] Mitchell, Tom
The Discipline of Machine Learning - Pittsburgh Carnegie Mellon University

[Mob02] Mobley, Keith R.
An Introduction to Predictive Maintenance
2. ed.: Butterworth-Heinemann, 2002

[N.N18a] N.N.: Why Do Predictive Maintenance Programs Fail? - Reliabilityweb, 2018, https://reliabilityweb.com/articles/entry/why_do_predictive_maintenance_programs_fail, Accessed on 11.03.2018

[N.N18b] N.N.: scikit-learn: machine learning in Python, http://scikit-learn.org/stable/about.html, Accessed on 25.07.2018

[N.N18c] N.N.: R: The R Project for Statistical Computing, https://www.r-project.org/about.html, Accessed on 25.07.2018

[OAC06] Orhan, Sadettin; Aktürk, Nizami; Celik, Veli
Vibration monitoring for defect diagnosis of rolling element bearings as a predictive maintenance tool: Comprehensive case studies
NDT & E International. Vol.39 (2006) 4, p. 293–298

[PBB18] Probst, Philipp; Bischl, Bernd; Boulesteix, Anne-Laure
Tunability: Importance of Hyperparameters of Machine Learning Algorithms, 26.02.2018, http://arxiv.org/pdf/1802.09596v1

[Ped11] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.
Scikit-learn: Machine Learning in Python
Journal of Machine Learning Research. Vol.12 (2011) 12, p. 2825–2830

[PK06] Parida, Aditya; Kumar, Uday
Maintenance performance measurement (MPM): issues and challenges
Journal of Quality in Maintenance Engineering. Vol.12 (2006) 3, p. 239–251

[PwC17] PwC (Publisher):
Digital Factories 2020, April 2017, https://www.pwc.de/de/digitale-transformation/digitalfactories-2020-inkl-international%20contacts-screen.pdf, Accessed on 27. Februar 2018
[Company publication]

[RLD14] Roser, C.; Lorentzen, K.; Deuse, J.
Reliable Shop Floor Bottleneck Detection for Flow Lines through Process and Inventory Observations
Procedia CIRP. Vol.19 (2014) 19, p. 63–68

[Rob01] Roberts, Paul
Failure Modes, Effects & Criticality Analysis, February 2001, https://warwick.ac.uk/fac/sci/wmg/ftmsc/modules/modulelist/peuss/slides/section_12a_fmeca_notes.pdf, Accessed on 07.04.2018

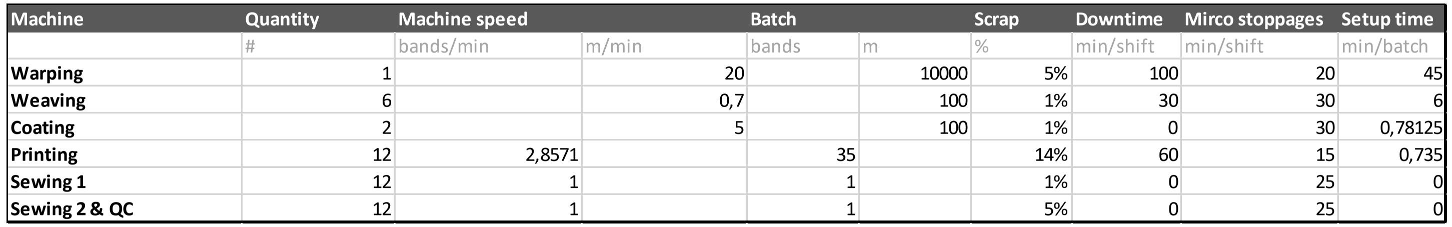
[Ron95] Ron Kohavi
A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection
International Joint Conference on Arti (1995) 1, p. 1137-1143

[Ros14] Roser, Christoph: The Bottleneck Walk – Practical Bottleneck Detection Part 1, 2014, https://www.allaboutlean.com/bottleneck-walk1/, Accessed on 25.03.2018

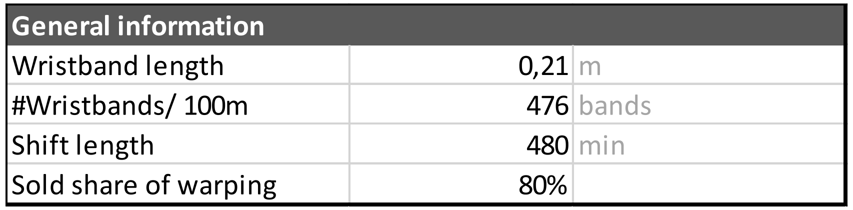
[Rya09]	Ryan, Jeffrey A.: Real Time Market Data and Trade Execution with R, https://cran.r-project.org/web/packages/IBrokers/vignettes/RealTime.pdf
[SH04]	Smith, Ricky; Hawkins, Bruce Lean maintenance - Amsterdam: Elsevier/Butterworth-Heinemann, 2004
[Sim13]	Simon, P. Too Big to Ignore: The Business Case for Big Data: Wiley, 2013
[Sul10]	Sullivan, Greg; Pugh, Ray; Melendez, Aldo P.; Hunt, W. D. Operations & Maintenance Best Practices - A Guide to Achieving Operational Efficiency (Release 3) United States, 2010
[Wed15]	Wedel, Michael; von Hacht, Michael; Hieber, Ralf; Metternich, Joachim; Abele, Eberhard Real-time Bottleneck Detection and Prediction to Prioritize Fault Repair in Interlinked Production Lines Procedia CIRP (2015) 37, p. 140–145
[WFH11]	Witten, Ian H.; Frank, Eibe; Hall, Mark A. Data mining 3. ed. - Amsterdam: Elsevier/Morgan Kaufmann, 2011
[Woo73]	Wood, Fred S. The Use of Individual Effects and Residuals in Fitting Equations to Data Technometrics. Vol.15 (1973) 4, p. 677–695
[Zha10]	Zhai; Sun; Wang (Publisher) An Effective Bottleneck Detection Method for Job Shop. Vol.2: IEEE, 2010a
[Zha10]	Zhang, J. (Publisher) New Advances in Machine Learning: In-Tech, 2010b

8 Attachment

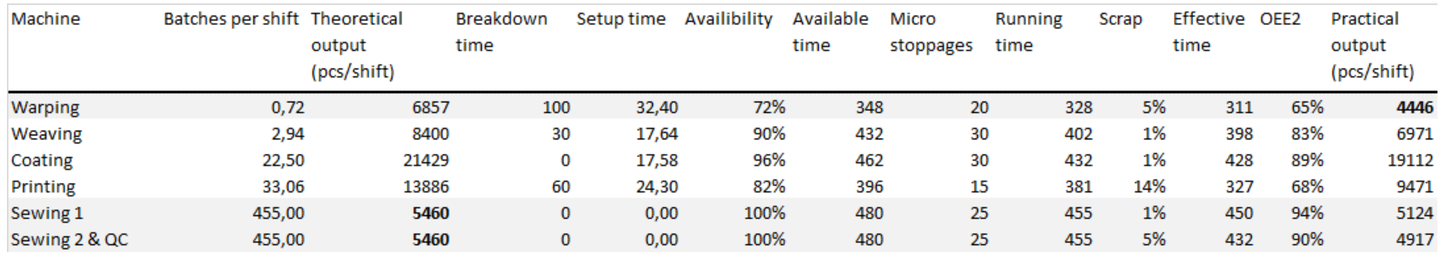
8.1 Attachment A: GoSmart business case



Machine	Quantity	Machine speed		Batch		Scrap	Downtime	Mirco stoppages	Setup time
	#	bands/min	m/min	bands	m	%	min/shift	min/shift	min/batch
Warping	1		20		10000	5%	100	20	45
Weaving	6		0,7		100	1%	30	30	6
Coating	2		5		100	1%	0	30	0,78125
Printing	12	2,8571		35		14%	60	15	0,735
Sewing 1	12	1		1		1%	0	25	0
Sewing 2 & QC	12	1		1		5%	0	25	0



General information		
Wristband length	0,21	m
#Wristbands/ 100m	476	bands
Shift length	480	min
Sold share of warping	80%	



Machine	Batches per shift	Theoretical output (pcs/shift)	Breakdown time	Setup time	Availibility	Available time	Micro stoppages	Running time	Scrap	Effective time	OEE2	Practical output (pcs/shift)
Warping	0,72	6857	100	32,40	72%	348	20	328	5%	311	65%	**4446**
Weaving	2,94	8400	30	17,64	90%	432	30	402	1%	398	83%	6971
Coating	22,50	21429	0	17,58	96%	462	30	432	1%	428	89%	19112
Printing	33,06	13886	60	24,30	82%	396	15	381	14%	327	68%	9471
Sewing 1	455,00	**5460**	0	0,00	100%	480	25	455	1%	450	94%	5124
Sewing 2 & QC	455,00	**5460**	0	0,00	100%	480	25	455	5%	432	90%	4917

8.2 Attachment B: Cost function

Note: original file needs to be opened with JupyterLab or JupyterNotebook (recommendation: anaconda)

```
# importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib as mpl

from mpl_toolkits.mplot3d import Axes3D, proj3d

from matplotlib import cm

from matplotlib.ticker import LinearLocator, FormatStrFormatter

%matplotlib inline

# setting up matplotlib (plotting library)

plt.style.use('grayscale')

plt.rcParams['font.family'] = 'Arial'

plt.rcParams['font.size'] = 10

plt.rcParams['axes.labelsize'] = 10

plt.rcParams['axes.labelweight'] = 'bold'

plt.rcParams['xtick.labelsize'] = 8

plt.rcParams['ytick.labelsize'] = 8

plt.rcParams['legend.fontsize'] = 8

plt.rcParams['figure.titlesize'] = 12

plt.rcParams['lines.linewidth'] = 2

plt.rcParams['image.cmap'] = 'Greys'

plt.rcParams['figure.figsize'] = [5.0, 3.0]

plt.rcParams['figure.dpi'] = 200

plt.rcParams['axes.autolimit_mode'] = 'round_numbers'

plt.rcParams['axes.linewidth'] = 1

#plt.rcParams['axes.axisbelow'] = True

plt.rcParams['axes.spines.top'] = False

plt.rcParams['axes.spines.right'] = False

plt.rcParams['legend.frameon'] = 'True'
plt.rcParams["legend.fancybox"] = 'True'

plt.rcParams["legend.framealpha"] = "1"

plt.rcParams["axes.facecolor"] = "white"

plt.rcParams["savefig.facecolor"] = "white"

plt.rcParams["figure.facecolor"] = "white"

interest_rate = 0.0032 #assuming constant interest rate. Equals 0.32
percent

hoursWorkingPerDay = 16 # Amount of hours in which the line runs per
day

lead_time_in_days = 4 #Lead time. Explaination see chapter 5.3

buffer_time_in_days = 2 #Buffer time. Explaination see chapter 5.3

breakdown_cost_per_hour_in_euro = 7070.7 #Breakdown costs per hour.
Explaination see chapter 5.3

spare_part_cost_in_euro = 300 #Spare part costs for the bearing
(estimation)

exchange_labor_cost_in_euro = 165/2 #Labor costs for exchanging.
Explaination see chapter 5.3

#This function calculates the revenue loss due to breakdown for day
"x" assuming the failure happened at day "time_of_failure" and with
breakdown costs "breakdown_cost_per_hour"

def RevenueLossDueToBreakdown_func(x, time_of_failure,
breakdown_cost_per_hour): # x in days

            y = breakdown_cost_per_hour*hoursWorkingPerDay * (x-
time_of_failure)

            return y # y in euro

# This function calculates the cost for over- and undercasting and has
as parameters the predicted RUL and the actual RUL

def CostFunction(actual_RUL_in_days, predicted_RUL_in_days): # (see
addition for predetermined maintenance!)

    predicted_RUL_in_days -= buffer_time_in_days #subtracting the
buffer time

    if (predicted_RUL_in_days < 0):

        predicted_RUL_in_days = 0

    maximum_RUL = actual_RUL_in_days # only for predetermined
maintenance
```

```
    if (actual_RUL_in_days >= lead_time_in_days and
predicted_RUL_in_days >=lead_time_in_days): # area of no cost

        return 0

    if (actual_RUL_in_days == predicted_RUL_in_days): # optimal
forecast

        return 0

    elif (actual_RUL_in_days > predicted_RUL_in_days): # algorithm
triggered to early -> unneccessary exchange

        total_cost = spare_part_cost_in_euro +
exchange_labor_cost_in_euro

        y = (total_cost / maximum_RUL) * (actual_RUL_in_days -
predicted_RUL_in_days) # hits y=0 at time of failure

        return y # y in euro

    else: # algorithm not detecting failure early enough --> revenue
loss due to breakdown

        return RevenueLossDueToBreakdown_func(predicted_RUL_in_days,
actual_RUL_in_days, breakdown_cost_per_hour_in_euro)

def CostFunctionArray(actual_RUL_array, predicted_RUL_array): #
Calculates the total costs for an array of actual and predicted RUL

    totalCost = 0

    for i in range(0, len(actual_RUL_array)):

        totalCost +=
CostFunction(convertRevolutionsIntoRemainingDays(actual_RUL_array[i]),
convertRevolutionsIntoRemainingDays(predicted_RUL_array[i]))

    return totalCost

time_range_in_days = 5 # time range to be plotted

#Creating the figure

fig = plt.figure(figsize=(8, 5))

ax = fig.add_subplot(111, projection='3d')

# Annotation

plt.annotate(

    "Poor utilization",

    xy = (0, 0), xytext = (-40, -60),

    textcoords = 'offset points', ha = 'right', va = 'bottom',

    bbox = dict(boxstyle = 'round,pad=0.5', fc = 'white', alpha = 1))
```

```
plt.annotate(
    "Area of no cost",
    xy = (0, 0), xytext = (45, -110),
    textcoords = 'offset points', ha = 'right', va = 'bottom',
    bbox = dict(boxstyle = 'round,pad=0.5', fc = 'white', alpha = 1))
plt.annotate(
    "Revenue loss due to breakdown",
    xy = (0, 0), xytext = (150, 0),
    textcoords = 'offset points', ha = 'right', va = 'bottom',
    bbox = dict(boxstyle = 'round,pad=0.5', fc = 'white', alpha = 1))
#Annotation end

actual_RUL = np.linspace(0, time_range_in_days, time_range_in_days*20)
predicted_RUL = np.linspace(0, time_range_in_days,
time_range_in_days*20)
X, Y = np.meshgrid(actual_RUL, predicted_RUL)
zs = np.array([CostFunction(x,y) for x,y in zip(np.ravel(X),
np.ravel(Y))])
Z = zs.reshape(X.shape)
surf = ax.plot_surface(X, Y, Z, cmap=cm.Greys,
                       linewidth=0.1, antialiased=True,
edgecolor="black")
ax.set_xlim(0, time_range_in_days)
ax.set_ylim(0, time_range_in_days)
ax.set_zlim(0, 300000)
ax.set_xlabel('Actual RUL [days]')
ax.set_ylabel('Predicted RUL [days]')
ax.set_zlabel('Cost [€]')
# Add a color bar which maps values to colors.
colorbar_graph = fig.colorbar(surf, shrink=0.4, aspect=5)
colorbar_graph.set_label("Cost [€]")
ax.view_init(20, 35)
plt.show()
```

8.3 Attachment C: Data pre-processing

Note: original file needs to be opened with JupyterLab or JupyterNotebook (recommendation: anaconda). Date and times are incorrect due to a misconfigured database.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib as mpl

from sklearn import datasets, linear_model

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

%matplotlib inline

df = pd.read_csv("WarpingMachineNewV1_20180425113902.csv", sep=";")

df_new = pd.read_csv("WarpingMachineNewV2_20180508132659.csv",
sep=";")

df_new_2 = pd.read_csv("WarpingMachineNewV3_20180525152521.csv",
sep=";")

df = pd.concat([df, df_new])

df = pd.concat([df, df_new_2])

df.machineCurrentlyRunning = df.machineCurrentlyRunning.astype(int)
#convert machineCurrentlyRunning from boolean to integer

df.TimeStamp = pd.to_datetime(df.TimeStamp, format='%d/%m/%Y %I:%M:%S
%p') # convert timestamp to pandas format

df.set_index('TimeStamp', inplace=True, drop=False)

df = df.drop(columns=['machineCurrentlyRunning', 'Id', 'yarntension',
'phase_quadrant_1', "phase_quadrant_2", "phase_quadrant_3",
"Total_Apparent_Power_VA", "temperature1_C"]) # remove as they have to
purpose

df = df[~(df['machineSpeed'] < 20)] # drop values with negative
machine speed (it is caused by resetting the total length at the
machine) & drop values where machine is not running as weigth cannot
be determined during machine downtime

#df = df[~(df['Total_Active_Power_W'] < 500)]

df['material_on_warp_beam_m'] = df.machineSpeed.cumsum()/60

df.tail()
```

```
def classifier(row):

    #print(df.index.get_values()[0] > np.datetime64('2018-04-25
09:13:23'))

    if row['TimeStamp'] > np.datetime64('2018-04-25T09:00:00') and
row['TimeStamp'] < np.datetime64('2018-04-25T09:52:00'):

        return 0

    elif row['TimeStamp'] > np.datetime64('2018-04-25T09:52:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T10:12:00'):

        return 100

    elif row['TimeStamp'] > np.datetime64('2018-04-25T10:12:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T10:28:00'):

        return 500

    elif row['TimeStamp'] > np.datetime64('2018-04-25T10:28:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T10:43:00'):

        return 300

    elif row['TimeStamp'] > np.datetime64('2018-04-25T10:43:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T11:00:00'):

        return 400

    elif row['TimeStamp'] > np.datetime64('2018-04-25T11:00:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T11:13:00'):

        return 200

    elif row['TimeStamp'] > np.datetime64('2018-04-25T11:13:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T11:19:00'):

        return 0

    elif row['TimeStamp'] > np.datetime64('2018-04-25T11:19:01') and
row['TimeStamp'] < np.datetime64('2018-04-25T11:22:00'):

        return 200

    elif row['TimeStamp'] > np.datetime64('2018-05-08T13:00:00') and
row['TimeStamp'] < np.datetime64('2018-05-08T13:14:00'):

        return 0

    elif row['TimeStamp'] > np.datetime64('2018-05-08T13:14:01') and
row['TimeStamp'] < np.datetime64('2018-05-08T13:21:00'):

        return 500

    elif row['TimeStamp'] > np.datetime64('2018-05-25T15:13:00') and
row['TimeStamp'] < np.datetime64('2018-05-25T15:17:00'):
        return 100
    elif row['TimeStamp'] > np.datetime64('2018-05-25T15:19:00') and
row['TimeStamp'] < np.datetime64('2018-05-25T15:22:00'):
        return 200
    else:
        return -1
df["weigth"] = df.apply(classifier, axis=1)
df = df[~(df['weigth'] == -1)] # drop data that is not classified
df = df.drop(columns=['TimeStamp'])
df.to_csv("export_v2.csv")
df.tail()
```

8.4 Attachment D: Algorithm comparison

Note: original file needs to be opened with JupyterLab or JupyterNotebook (recommendation: anaconda)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from matplotlib.ticker import FormatStrFormatter
%matplotlib inline
plt.style.use('grayscale')
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 10
plt.rcParams['axes.labelsize'] = 10
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['xtick.labelsize'] = 8
plt.rcParams['ytick.labelsize'] = 8
plt.rcParams['legend.fontsize'] = 8
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['image.cmap'] = 'Greys'
plt.rcParams['figure.figsize'] = [5.0, 3.0]
plt.rcParams['figure.dpi'] = 200
plt.rcParams['axes.autolimit_mode'] = 'round_numbers'
plt.rcParams['axes.linewidth'] = 1
plt.rcParams['axes.spines.top'] = False
plt.rcParams['axes.spines.right'] = False
plt.rcParams['legend.frameon'] = 'True'
plt.rcParams["legend.fancybox"] = 'True'
plt.rcParams["legend.framealpha"] = "1"
plt.rcParams["axes.facecolor"] = "white"
plt.rcParams["savefig.facecolor"] = "white"
plt.rcParams["figure.facecolor"] = "white"
pd.set_option('display.max_columns', 100)
# Sliding window function
from sklearn.base import BaseEstimator, TransformerMixin
# Using the sliding window technique as a custom transformer
class SlidingWindow(BaseEstimator, TransformerMixin):
    def __init__(self, window_samples):
        self.window_width = window_samples
    def transform(self, X,y=None):
        condensed_X = pd.merge(X,
                               X.rolling(self.window_width).mean(),
                               how='left',
                               suffixes=('_normal', '_slided'),
                               left_index=True,
                               right_index=True)

        return condensed_X
    def fit(self,X,y=None):
        # This class does not neet fitting (yet)
        return self
# function to convert wear state into RUL

def convertWearStateIntoRUL(wear_state):
    # wear_state between 0 (OK), and 1 (failure)
    #0 equals 50 days
    #1 equals 0 days
    return (1-wear_state)*50
# Cost function. See also previous definition of cost function
hoursWorkingPerDay = 16
averageFailuresPerYear = 14.23
workingDaysPerYear = 300

maximum_RUL = workingDaysPerYear / averageFailuresPerYear # average
RUL

interest_rate = 0.0032

lead_time_in_days = 4
buffer_time_in_days = 2

breakdown_cost_per_hour_in_euro = 7063.88
spare_part_cost_in_euro = 10000
exchange_labor_cost_in_euro = 165/2

predeterminedMaintenance_netcurrentvalue = -3009195.34 # net current
value of predetermined maintenance
time_area_before_failure_in_days = lead_time_in_days
def RevenueLossDueToBreakdown_func(x, time_of_failure,
breakdown_cost_per_hour): # x in days
```

```
            y = breakdown_cost_per_hour*hoursWorkingPerDay * x -
(breakdown_cost_per_hour*hoursWorkingPerDay * time_of_failure)

            return y # y in euro

def CostFunction(actual_RUL_in_days, predicted_RUL_in_days):

    global averageMachineDowntimePerYear

    predicted_RUL_in_days -= buffer_time_in_days #subtracting the
buffer

    if (predicted_RUL_in_days < 0):

        predicted_RUL_in_days = 0

    if (actual_RUL_in_days >= lead_time_in_days and
predicted_RUL_in_days >=lead_time_in_days): # area of no cost

        return 0

    if (actual_RUL_in_days == predicted_RUL_in_days): # optimal
forecast

        return 0

    elif (actual_RUL_in_days > predicted_RUL_in_days): # algorithm
triggered to early -> unneccessary exchange

        total_cost = spare_part_cost_in_euro +
exchange_labor_cost_in_euro

        y = (total_cost / maximum_RUL) * (actual_RUL_in_days -
predicted_RUL_in_days) # hits y=0 at time of failure

        return y # y in euro

    else: # algorithm not detecting failure early enough --> revenue
loss due to breakdown

        return RevenueLossDueToBreakdown_func(predicted_RUL_in_days,
actual_RUL_in_days, breakdown_cost_per_hour_in_euro)

def CostFunctionArray(actual_RUL_array, predicted_RUL_array):

    totalCost = 0

    for i in range(0, len(actual_RUL_array)):

        totalCost +=
CostFunction(convertWearStateIntoRUL(actual_RUL_array[i]),
convertWearStateIntoRUL(predicted_RUL_array[i]))

    return totalCost

#import pre-processed .csv

df = pd.read_csv("export_v2.csv", sep=",")
```

```
df.TimeStamp = pd.to_datetime(df.TimeStamp)
df.set_index('TimeStamp', inplace=True)
#removing material_on_warp_beam_m as it is not important
df = df.drop(columns=['material_on_warp_beam_m'])
train = df['2018-04-25T09:00:00':'2018-04-25T11:13:00']
train = train.append(df['2018-05-08T13:00:00':'2018-05-25T15:22:00'])
#add line above for additional data
test = df['2018-04-25T11:13:00':'2018-04-25T11:22:00']
#use line above for validation data
#test = train
#use line above to use the same dataset for training and testing
# convert weight into RUL
train_features = train[train.columns.difference(['weigth'])]
train_target = np.sqrt(train["weigth"]/500).to_frame()
test_features = test[test.columns.difference(['weigth'])]
test_target = np.sqrt(test["weigth"]/500).to_frame()
total_features = df[df.columns.difference(['weigth'])]
total_target = np.sqrt(df["weigth"]/500).to_frame()
#algorithm and plot function

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.dates as mdates

#used for currency display
import locale
locale.setlocale(locale.LC_ALL, 'de_DE')

rf_importance_df = pd.DataFrame()
regression_importance_df = pd.DataFrame()

def returnPlotForRegressor(regressor, subplot, regressorName):
    global rf_importance_df

    global regression_importance_df

    global feature_list


    # Sliding window init

    slider = SlidingWindow("30s")

    feature_list = slider.transform(total_features)


    #Pipeline init. Add 30s sliding window, scale and then try out the
algorithm

    pipeline = Pipeline([

    ("slidingWindow", slider),

    ("scaler",StandardScaler()),

    ("regressor", regressor)

    ])


    # Run pipeline

    pipeline.fit(train_features, train_target)

    prediction = pipeline.predict(test_features)# test_features


    #BEGIN Print detailled information about algorithms #


    if (regressorName == "Random Forest"):

        # Get numerical feature importances

        importances =
list(pipeline.named_steps['regressor'].feature_importances_)


        feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list.columns, importances)]

        # Sort the feature importances by most important first

        feature_importances = sorted(feature_importances, key = lambda
x: x[1], reverse = True)
```

```
        rf_importance_df = pd.DataFrame(feature_importances)

        rf_importance_df.set_index(0, inplace=True)

        rf_importance_df.columns = ['importance']

        rf_importance_df.index.names = ['Feature name']

    elif (regressorName == "Linear regression"):

        regression_importance_df =
pd.DataFrame(list(zip(feature_list.columns,
pipeline.named_steps['regressor'].coef_[0])))

        regression_importance_df.set_index(0, inplace=True)

        regression_importance_df.columns = ['coefficient']

        regression_importance_df =
regression_importance_df.reindex(regression_importance_df.coefficient.
abs().sort_values().index)

        regression_importance_df.index.names = ['Feature name']

    #END Print detailled information about algorithms #

    #Score and calculate net current values

    score = CostFunctionArray(test_target.values, prediction)

    averageCostPerBearing = (score/len(test_target.index))

    netCurrentValue = np.npv(interest_rate,[-81800,
averageFailuresPerYear * -averageCostPerBearing,
averageFailuresPerYear * -averageCostPerBearing,
averageFailuresPerYear * -averageCostPerBearing])

    netCurrentValueDifference = netCurrentValue-
predeterminedMaintenance_netcurrentvalue

    target_df = test_target # test_target

    prediction_df = pd.DataFrame(data=prediction,
index=test_target.index) # test_target

    # convert wear state into RUL

    converted_target_df = target_df.apply(convertWearStateIntoRUL)

    converted_prediction_df =
prediction_df.apply(convertWearStateIntoRUL)

    predictionGraph, = subplot.plot(converted_prediction_df,
label="Predicted RUL", color="grey")

    targetGraph, = subplot.plot(converted_target_df, label="Actual
RUL", linestyle='dashed', color="black")
```

```
    """targetGraph, = subplot.plot(target_df, label="Actual RUL")

    predictionGraph, = subplot.plot(prediction_df, label="Predicted 
RUL")"""

    #subplot.legend(handles=[targetGraph, predictionGraph])

    subplot.set_xlabel('Time')

    subplot.set_ylabel('RUL [in days]')

    subplot.set_title('%s' % (regressorName,))

    subplot.text(.5,.93,'Net present value difference: %s' 
%(locale.currency(netCurrentValueDifference, grouping=True),), 
fontsize=10, ha='center', clip_on=True, transform=subplot.transAxes)

    subplot.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))

    subplot.set_xticklabels([]) #remove this line to have time shown

    return subplot

# print importance of random forest

rf_importance_df.head()

# print importance of linear regression

regression_importance_df.tail()

#print boxplots for different wear states

def seperateDfIntoWearStates(dataframe_to_seperate_features, 
dataframe_to_seperate_target):

    features_0 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(0/500)]

    features_100 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(100/500)]

    features_200 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(200/500)]

    features_300 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(300/500)]

    features_400 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(400/500)]

    features_500 = dataframe_to_seperate_features[ 
dataframe_to_seperate_target.weigth == np.sqrt(500/500)]

    return (features_0, features_100, features_200, features_300, 
features_400, features_500)

def plotBoxplotForDifferentWearStates(features, target):
    labels = (np.sqrt(0/500), np.sqrt(100/500), np.sqrt(200/500),
np.sqrt(300/500), np.sqrt(400/500), np.sqrt(500/500))

    features_0 = features[ target.weigth == np.sqrt(0/500)]

    features_100 = features[ target.weigth == np.sqrt(100/500)]

    features_200 = features[ target.weigth == np.sqrt(200/500)]

    features_300 = features[ target.weigth == np.sqrt(300/500)]

    features_400 = features[ target.weigth == np.sqrt(400/500)]

    features_500 = features[ target.weigth == np.sqrt(500/500)]

    fig, (ax1, ax2) = plt.subplots(2,1, figsize=(5,5), dpi=200,
sharey=True)

    train_feat = features['2018-04-25T09:00:00':'2018-04-25T11:13:00']

    train_feat = train_feat.append(features['2018-05-
08T13:00:00':'2018-05-25T15:22:00'])

    train_target = target['2018-04-25T09:00:00':'2018-04-25T11:13:00']

    train_target = train_target.append(target['2018-05-
08T13:00:00':'2018-05-25T15:22:00'])

    valid_feat = features['2018-04-25T11:13:00':'2018-04-25T11:22:00']

    valid_target = target['2018-04-25T11:13:00':'2018-04-25T11:22:00']

    ax1.boxplot(seperateDfIntoWearStates(train_feat, train_target) ,
labels=labels)

    ax1.set_title("Boxplot on training dataset")

    ax1.set_xlabel('RUL [in days]')

    ax1.set_ylabel('Tension [in N]')

    #ax2.boxplot(seperateDfIntoWearStates(features['2018-05-
08T13:00:00':'2018-05-08T13:21:00'], target['2018-05-
08T13:00:00':'2018-05-08T13:21:00']) , labels=labels)

    ax2.boxplot(seperateDfIntoWearStates(valid_feat, valid_target) ,
labels=labels)

    ax2.set_title("Boxplot on validation dataset")

    ax2.set_xlabel('RUL [in days]')

    ax2.set_ylabel('Tension [in N]')

    ax1.set_xticklabels([convertWearStateIntoRUL(0),
round(convertWearStateIntoRUL(np.sqrt(100/500)),2),
round(convertWearStateIntoRUL(np.sqrt(200/500)),2),
round(convertWearStateIntoRUL(np.sqrt(300/500)),2),
round(convertWearStateIntoRUL(np.sqrt(400/500)),2),
round(convertWearStateIntoRUL(np.sqrt(1)),2)])

    ax2.set_xticklabels([convertWearStateIntoRUL(0),
round(convertWearStateIntoRUL(np.sqrt(100/500)),2),
round(convertWearStateIntoRUL(np.sqrt(200/500)),2),
round(convertWearStateIntoRUL(np.sqrt(300/500)),2),
round(convertWearStateIntoRUL(np.sqrt(400/500)),2),
round(convertWearStateIntoRUL(np.sqrt(1)),2)])

    plt.tight_layout()

    plt.show()

#Remove material on warp beam and only work with machine speed ~100 m
/ min

adjusted_df = total_features

adjusted_df = adjusted_df[~ (adjusted_df['IST_SPEED_WICKLER_M_Min'] >
110)]

adjusted_df = adjusted_df[~ (adjusted_df['IST_SPEED_WICKLER_M_Min'] <
95)]

plotBoxplotForDifferentWearStates(feature_list['yarntension3_slided'],
total_target)

def seperateDfIntoWearStates(dataframe_to_seperate_features,
dataframe_to_seperate_target):

    features_0 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(0/500)]

    features_100 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(100/500)]

    features_200 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(200/500)]

    features_300 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(300/500)]

    features_400 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(400/500)]

    features_500 = dataframe_to_seperate_features[
dataframe_to_seperate_target.weigth == np.sqrt(500/500)]

    return (features_0, features_100, features_200, features_300,
features_400, features_500)

def plotBoxplotForDifferentWearStates(features, target):
    labels = (np.sqrt(0/500), np.sqrt(100/500), np.sqrt(200/500),
np.sqrt(300/500), np.sqrt(400/500), np.sqrt(500/500))

    features_0 = features[ target.weigth == np.sqrt(0/500)]

    features_100 = features[ target.weigth == np.sqrt(100/500)]

    features_200 = features[ target.weigth == np.sqrt(200/500)]

    features_300 = features[ target.weigth == np.sqrt(300/500)]

    features_400 = features[ target.weigth == np.sqrt(400/500)]

    features_500 = features[ target.weigth == np.sqrt(500/500)]

    fig, (ax1, ax2) = plt.subplots(2,1, figsize=(5,5), dpi=200,
sharey=True)

    train_feat = features['2018-04-25T09:00:00':'2018-04-25T11:13:00']

    train_feat = train_feat.append(features['2018-05-
08T13:00:00':'2018-05-25T15:22:00'])

    train_target = target['2018-04-25T09:00:00':'2018-04-25T11:13:00']

    train_target = train_target.append(target['2018-05-
08T13:00:00':'2018-05-25T15:22:00'])

    valid_feat = features['2018-04-25T11:13:00':'2018-04-25T11:22:00']

    valid_target = target['2018-04-25T11:13:00':'2018-04-25T11:22:00']

    ax1.boxplot(seperateDfIntoWearStates(train_feat, train_target) ,
labels=labels)

    ax1.set_title("Boxplot on training dataset")

    ax1.set_xlabel('RUL [in days]')

    ax1.set_ylabel('Acceleration [in 9.81 m/s^2]')

    #ax2.boxplot(seperateDfIntoWearStates(features['2018-05-
08T13:00:00':'2018-05-08T13:21:00'], target['2018-05-
08T13:00:00':'2018-05-08T13:21:00']) , labels=labels)

    ax2.boxplot(seperateDfIntoWearStates(valid_feat, valid_target) ,
labels=labels)

    ax2.set_title("Boxplot on validation dataset")

    ax2.set_xlabel('RUL [in days]')

    ax2.set_ylabel('Acceleration [in 9.81 m/s^2]')

    ax1.set_xticklabels([convertWearStateIntoRUL(0),
round(convertWearStateIntoRUL(np.sqrt(100/500)),2),
round(convertWearStateIntoRUL(np.sqrt(200/500)),2),
round(convertWearStateIntoRUL(np.sqrt(300/500)),2),
round(convertWearStateIntoRUL(np.sqrt(400/500)),2),
round(convertWearStateIntoRUL(np.sqrt(1)),2)])

    ax2.set_xticklabels([convertWearStateIntoRUL(0),
round(convertWearStateIntoRUL(np.sqrt(100/500)),2),
round(convertWearStateIntoRUL(np.sqrt(200/500)),2),
round(convertWearStateIntoRUL(np.sqrt(300/500)),2),
round(convertWearStateIntoRUL(np.sqrt(400/500)),2),
round(convertWearStateIntoRUL(np.sqrt(1)),2)])


    plt.tight_layout()

    plt.show()

#Remove material on warp beam and only work with machine speed ~100 m
/ min

adjusted_df = total_features

adjusted_df = adjusted_df[~ (adjusted_df['IST_SPEED_WICKLER_M_Min'] >
110)]

adjusted_df = adjusted_df[~ (adjusted_df['IST_SPEED_WICKLER_M_Min'] <
95)]

plotBoxplotForDifferentWearStates(feature_list['WarpingMachine.Vibrati
onMonitor.y.lowerQuantile_slided'], total_target)
```

8.5 Attachment E: Costs for bearing exchange using predetermined maintenance

Note: original file needs to be opened with JupyterLab or JupyterNotebook (recommendation: anaconda)

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib as mpl

import math

from sklearn import datasets, linear_model

%matplotlib inline

hoursWorkingPerDay = 16

machineSpeed = 400 #m/min
```

```
averageDiameter = 0.4 #m

revolutionsPerMinute = machineSpeed/(math.pi * averageDiameter)

revolutionsPerDay = revolutionsPerMinute * 60 * hoursWorkingPerDay

workingDaysPerYear = 300

def convertRevolutionsIntoRemainingDays(revolutions):

    remainingRevolutions = revolutions * 10**6 / revolutionsPerDay

    return remainingRevolutions

print(convertRevolutionsIntoRemainingDays(1.4))

averageFailuresPerYear =
   workingDaysPerYear/convertRevolutionsIntoRemainingDays(6.44)

print("Average exchanges per year:
   {0}".format(averageFailuresPerYear))

averageMachineDowntimePerYear = 0

tenPercentFailureInRevolutions = 6.44

interest_rate = 0.0032


lead_time_in_days = 4 # the time amount the company needs to know
   before a failure will happen, e. g. spare time delivery time,
   allocation of labor, etc.

buffer_time_in_days = 2

breakdown_cost_per_hour_in_euro = 7070.7

spare_part_cost_in_euro = 300

exchange_labor_cost_in_euro = 165/2


def RevenueLossDueToBreakdown_func(x, time_of_failure,
   breakdown_cost_per_hour): # x in days

            y = breakdown_cost_per_hour*hoursWorkingPerDay * x -
   (breakdown_cost_per_hour*hoursWorkingPerDay * time_of_failure)

            return y # y in euro

def CostFunction(actual_RUL_in_days, predicted_RUL_in_days): # see
   addition for predetermined maintenance

    global averageMachineDowntimePerYear

    predicted_RUL_in_days -= buffer_time_in_days #subtracting the
   buffer
```

```
    if (predicted_RUL_in_days < 0):

        predicted_RUL_in_days = 0

    maximum_RUL = actual_RUL_in_days # only for predetermined
  maintenance

    if (actual_RUL_in_days >= lead_time_in_days and
  predicted_RUL_in_days >=lead_time_in_days): # area of no cost

        return 0

    if (actual_RUL_in_days == predicted_RUL_in_days): # optimal
  forecast

        return 0

    elif (actual_RUL_in_days > predicted_RUL_in_days): # algorithm
  triggered to early -> unneccessary exchange

        total_cost = spare_part_cost_in_euro +
  exchange_labor_cost_in_euro

        y = (total_cost / maximum_RUL) * (actual_RUL_in_days -
  predicted_RUL_in_days) # hits y=0 at time of failure

        return y # y in euro

    else: # algorithm not detecting failure early enough --> revenue
  loss due to breakdown

        averageMachineDowntimePerYear += -
  actual_RUL_in_days+predicted_RUL_in_days

        return RevenueLossDueToBreakdown_func(predicted_RUL_in_days,
  actual_RUL_in_days, breakdown_cost_per_hour_in_euro)

def CostFunctionArray(actual_RUL_array, predicted_RUL_array):

    totalCost = 0

    for i in range(0, len(actual_RUL_array)):

        totalCost +=
  CostFunction(convertRevolutionsIntoRemainingDays(actual_RUL_array[i]
  ), convertRevolutionsIntoRemainingDays(predicted_RUL_array[i]))

    return totalCost
```

9 Statement of academic honesty

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

I hereby declare to the best of my knowledge that this thesis contains no material previously published or written by any other person. The work submitted in this thesis is the product of my own original research, except where I have duly acknowledged the work of others.

Ort, Datum	Unterschrift
City, Date	Signature