University of South-Eastern Norway

www.usn.no

FM4017 Project 2021

Machine Learning for Predictive Maintenance of pumps at “Den Magiske Fabrikken”

MP-05-21

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FM4017 Project, 2021

Title: Machine Learning for Predictive Maintenance of pumps at "Den Magiske Fabrikken"

This report forms part of the basis for assessing the student's performance in the course.

Project group: MP-05-21

Group participants:	162696 Martin Holm
	238769 Ozgur Yalcin
	220160 Ronnie André Horne Moe
Supervisor:	Carlos Pfeiffer, Håkon Viumdal
Project partner:	Lindum AS

Summary:

This project has an external partner called Lindum AS which processes animal manure, food, and human waste for biogas production. In the process of the treatment, there is a need for changing for pumps due to failures. The project investigates and evaluates progressive cavity pump failures used in a waste processing plant by applying the machine learning methodology approach. The main steps are data gathering, pre-processing and implementing selected machine learning algorithms. Speed, ampere, vibration, torque, flow, control signal and pressure values of the pumps have been used. Furthermore, it has been studied in three different time frames 1s, 30s and 50ms data. The pre-processing phase consists of data averaging and outlier removal. The following machine learning algorithms were investigated: Support vector machine, naïve Bayes, and long-short term memory algorithms. Additionally, PCA analysis was conducted for examining variations while frequency components were analysed by Fourier Transform Spectrogram. It is concluded that 30s data produces better outcomes regarding detecting failure states. Scatter plots, correlation matrixes and confusing matrixes has been used to represent the data and results. As a result of the implementation of ML algorithms, accuracy results have been obtained and reported. The NB model trained on 30s data from May 2020 to August 2021 achieved a precision of 70.6% on the test set from the same timeframe as the training set. The SVM trained model on 30s achieved a much higher accuracy, 95.5%. LSTM-model trained on 30-second data were 78.5%.

Preface

This project constitutes the masters programme at University of South-Eastern Norway. The project was completed during the autumn semester in 2021.

Two students from Industrial IT and Automation master's programme and one student from Energy and Environmental Technology programme were participants of the project. There are several persons associated to this project whom we would like to thank. Firstly, we would like to thank the supervisors at the department, Professor Carlos Pfeiffer and Associate Professor Håkon Viumdal for providing valuable assistance. Secondly, we would like to thank Bertil Johansen and Frode Steen from Lindum AS for providing the data and their support.

Porsgrunn, November 18, 2021

Ronnie Moe

Martin Holm

Ozgur Yalcin

Contents

1 Introduction 7

2 Theory and Methods 8

2.1 Progressive cavity pumps 8
2.2 Predictive maintenance 9
2.2.1 Prognostics and health management 10
2.2.2 Diagnosis versus prognosis 11
2.3 Machine learning 12
2.3.1 Pre-Processing Methods 13
2.3.2 Principal Component Analysis 14
2.3.3 Classical pattern recognition 15
2.3.4 Naïve Bayes 16
2.3.5 Support Vector Machines 16
2.3.6 Neural Networks 17
2.3.7 Other algorithms 18
2.3.8 Understanding the confusion matrix 19
2.3.9 Built-in libraries are used in this study: Scikit Learn and Tensor Flow 19
2.4 Process Description 19
2.4.1 P&ID 19
2.4.2 Sampling intervals 21
2.5 Introduction to the data 22
2.5.1 Output classes 22
2.5.2 30 second interval 22
2.5.3 1 second interval 24
2.5.4 50 millisecond intervals 27
2.5.5 Time-series 50ms data 29
2.6 Outlier removal 30
2.7 Using averages and quantiles to remove noise and outliers 32
2.7.1 30 second averaged to 10 minutes 33
2.7.2 1 second data to 20 second 36
2.7.3 50ms to 1 second 39

3 Result 43

3.1 Analysis 43
3.1.1 PCA 43
3.1.2 Fourier transform spectrogram 52
3.2 Machine learning training and tests 55
3.2.1 Naïve Bayes 55
3.2.2 Support Vector Machine 58
3.2.3 Long Short-Term Memory 61

4 Discussion 68

5 Conclusion 69

5.1 Future work 69
5.1.1 Reporting of replacement 69
5.1.2 Retrain model over time 69
5.1.3 Live implementation 69

6 Bibliography 70

Nomenclature

Symbols

Symbol	Unit	Description
MO_PV	%	Control Signal
SF_PV	%	Speed
PW_PV	A	Ampere
TQ_PV	%	Torque
FT	m/s	Flow
PT	bar	Pressure
V_O/L/I	mm/s	Vibration
PeakVue	--	Emerson Peak Value

Abbreviations

Symbol	Description
AMS	Asset Monitor System
ARMA	Autoregressive Moving Average Model
CNNs	Convolutional Neural Networks
DBSCAN	Density-based Spatial Clustering of Applications with Noise
FFT	Fast Fourier transform
g	Gravity
GPU	Graphics Processing Unit
IQR	Interquartile range
IT	Information Technology
k-NNs	k-nearest Neighbours
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory Network
ML	Machine Learning
MO_PV	Control Signal
NB	Naïve Bayes
PCA	Principal Component Analysis
PHM	Prognostics and health management
PLC	Programmable Logic Controller
PW_PV	Power
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristics
RUL	Remaining Useful Life
SCADA	Supervisory Control and Data Acquisition
SF_PW	Speed
STFT	Short-Time Fourier Transform
SVMs	Support Vector Machines
TQ_PV	Torque
W-SVM	Wavelet Support Vector Machines

1 Introduction

In the industry, cost optimization pushes the engineers hard to decrease maintenance costs while maintaining high production rates. Since the control and process optimization systems have become advanced, predictive maintenance methods and their tools have changed the way how to approach problems. For example, it is now possible to detect whether the pump fails soon by some predictive maintenance methods. But more importantly, machine learning algorithms take the analysing phase to a step further and can even provide the time when it may stop functioning. As a result, machine learning tools have gained immense popularity in control systems.

This project investigates and evaluates progressive cavity pump failures used in a waste processing plant by applying the machine learning methodology approach.

Lindum operates a waste management company that produces biogas as a result of processing animal manure, food, and human waste. During the processing phase, the highly corrosive and acidic flow goes into pumps and causes severe effects on the pumps. In order to prevent any production losses and increase the pump lifetime, pumps are going under maintenance periodically. Obviously, excessive controls may result in increased labour force demand and changing costly equipment parts every time. Increased controls also do not prevent unexpected failures. Consequently, the objective of this project is to determine the parameters that have more impact on the pump failures by analysing pump parameters with ML algorithms and proposing a model to detect faults.

Firstly, the process has been tried to understand by the project participants by analysing PI&D diagrams and process descriptions. Later, a literature survey was conducted on progressive cavity pumps and machine learning approaches for predictive maintenance. Hence, the main steps have been identified as accessing data, pre-processing, deciding on the pump that needs to be focused on, machine learning algorithms, and model development.

The required data was provided by Lindum and by accessing the database, flow and pressure of the fluid and vibration, torque, speed, power, amper, PeakVue, and control signal of the pump parameters were obtained. Python and its packages have been used during the programming. Scatter plots do an excellent job of observing patterns. That is why values were plotted against each other to analyse the data. On a practical level, implementing PCA analysis offers unique insights into understanding patterns and variations. Furthermore, to compute and visualize the changing frequency components in the torque signal over time, Fourier transform spectrogram graphs were plotted. After trying some pre-processing methods, it was decided which method suits the best to data. As a result of the literature survey on machine learning algorithms that use in predictive maintenance, Support vector machine, naïve Bayes, and long-short term memory algorithms have been applied to the data. Results have shown under the analysis for data section with detailed information, and findings were criticised in the discussion section. Some statics results and programme code were given under the appendix section.

2 Theory and Methods

This chapter contains theoretical knowledge which may be used in the remaining chapter for the goal of assessing the progressive cavity pumps health.

2.1 Progressive cavity pumps

Progressive Cavity pumps are a type of positive displacement pump that can handle solids and high viscosity fluids. The pump technology differs from centrifugal pumps where efficiency decreases with increasing viscosity. In addition, pump efficiency decreases at both higher and lower pressures in centrifugal pumps whereas pump efficiency increases with increasing pressure in positive displacement pumps.

In general, positive displacement pumps transfer the fluid through the action of gears, screws, pistons or diaphragms. There is one specific type of positive displacement pump that is well known for pushing discrete quantities of liquids and solids through the pump and that is the progressive cavity pump.

Progressive cavity pumps are used in various industries such as food, chemical and wastewater. A progressive cavity pump has the following components: rotor, stator, drive train, shaft sealing, and suction housing [1]. The stator is the most important part as it faces the fluid directly. That is why the selection of stator plays a vital role.

The selection of high elastomer resilience will provide high continuous production and equipment lifetime; however, some elastomers have much higher swell rates in applications above 50°C [2]. High mechanical properties, wide working temperature range, economical options are highly preferable. The temperature has a big impact on the resiliency of the stator elastomer. Elastomer expansion occurs due to temperature increase; however, this expansion is restricted by the shell/metal casting [3].

In the case of high viscous fluid pumping, the motor speed should be adjusted to avoid volumetric flow efficiency loss.

Furthermore, operational problems may occur when pumping fluid exceeds the rated torque limit of the rod string where happens in the case of rapid influx.

There are some points to avoid pump failure specifically in progressive cavity pumps. These are:

- Choosing the right elastomer type with taking into account temperature and fluid physical properties.
- Avoiding dry running conditions.
- Selecting suitable rotor material to stay away from abrasive wear on the rotor.
- Applying predictive and preventive maintenance methods.

In this project, the analysed pump type is ‘Nemo’ brand progressive cavity pumps produced by Netzsch Pumpen & Systeme GmbH. These types of pumps provide a large capacity and pressure range with up to 1000 m^3/h flow rates and with several stages ranging from 1 to 8 for pressures from 6 to 48 bar or up to 240 bar for high-pressure applications. Furthermore, the pumps are able to pump solid size up to 150 mm and viscous materials from 1 mPas to 3 million mPas.

2.2 Predictive maintenance

Continuity in production has never been important as before in today's world as competitive prices dominate the market. In addition to that, although the cost reduction analysis is an old phenomenon, it is still worth taking into consideration. The advances in technology have created new ways for engineers to analyse the production systems better. Meanwhile, a couple of decades ago, the industry has started to flood with process data. Not only did automatic control systems improve processes but also did provide unique insights into equipment control. And now, the data has been shaping the maintenance methods.
Different types of maintenance methods can be applied for different scenarios. While some methods depend on inspector experience, other methods use process variables. Furthermore, planned maintenance is highly preferred as it gives chance to the maintenance team to check each detail of the equipment. There are general types of maintenance strategies commonly used in the industry [4]. These are;
Reactive Maintenance
Preventive Maintenance
Condition Based Maintenance
Predictive Maintenance

Reactive Maintenance refers to repairing the equipment after a breakdown or observed failure. It might cause a great number of production losses and requires immediate maintenance team action. It does create uncertainty about root cause of problem-solving. Due to maintenance taking place under pressure and against time, the problem might trigger again.

Preventive Maintenance is a systematic approach in order to predict and prevent equipment failures before it occurs. There are several types of preventive maintenance methods that are used in the industry. These are inspection based, equipment working hour based and time scheduled based methods. The method requires a planning phase and organisational capability.

Condition Based Maintenance is, however, considers the actual use of equipment instead of relying on pre-scheduled intervals. By doing that, maintenance costs can be optimized.
Nevertheless, these methods cause production losses. In order to minimize the production losses and increase equipment lifespan, predictive maintenance methods are used.

Predictive Maintenance aims to transform advanced analytical and process data into valued outcomes. Hence, equipment failure or breakdown can be prevented just before it occurs. The method uses advanced process control systems and algorithms in addition to process condition monitoring. But more importantly, predictive maintenance takes advantage of machine learning algorithms to build a systematic approach. Machine learning has dominated predictive maintenance and changed the way how to evaluate the impact of process data on equipment. Besides, predictive maintenance minimizes the cost of maintenance and improve the equipment life without causing unpredicted production losses. The process works as long as possible without interruption.
According to an analysis conducted by Deloitte, predictive maintenance increases equipment uptime by 10 to 20% while reducing overall maintenance costs by 5 to 10% and maintenance planning time by 20 to 50% [4].
In order to conduct a predictive maintenance system, some resources are required such as reliable process data, failure logs with exact time, and trained personnel.

Reliable process data consists of temperature, pressure, flow, voltages, torque, vibration and so on. Also, a systematic documentation on equipment failures with exact date and time is necessary. Trained personnel are pivotal to productively using the algorithm outcomes. Figure 2-1 represents a schematic drawing on predictive maintenance and ML algorithms implemented in the process systems.
Predictive maintenance by big data not only helps to reduce costs and extend the lifetime of equipment but also cut risks associated with health and safety.
Like any other big transforms in the industry, before implementing predictive maintenance it is important to integrate it into company policy and be supported by top executives. Because there might be additional investments in equipment to measure process data or IT infrastructure. One study conducted by PWC reveals that the companies who do not have plans to use predictive maintenance, state lack of budget as the main reason [5].

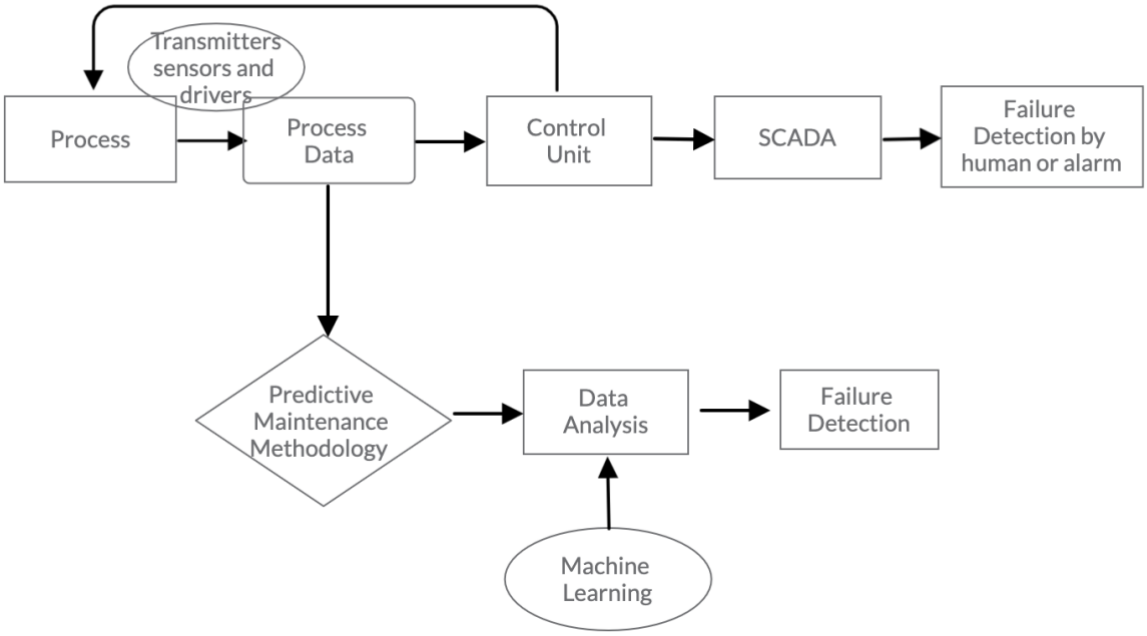


Figure 2-1: *Process flow diagram where predictive maintenance and machine learning implement*

2.2.1 Prognostics and health management

To decide what type of monitoring system should be made, a distinguishing of the type of system to monitor must also be made. A meta study of prognostic and health management [6] indicates that the difference in systems to monitor can be categorized as static and dynamic as seen in Figure 2-2. A static system is considered a system which stays the same if nothing goes wrong, as in a part is being deteriorated. A dynamic system may change such that even if the equipment is fine, the measurements may vary due to the environment it runs in. These distinctions will invariably have a role in picking the right type of monitoring system. Here,

the focus will be on the upper right square, Prognostics & health management, resilient systems, self-maintenance and to some degree engineering immune systems.
Prognostics and health management (PHM) can be used in systems where there are many measurements and some probabilistic elements, while the system being monitored is still static. Typical of these systems are that they utilize vibration, oil, temperature, acoustic or ultrasonic analysis to estimate the performance of the machine. Varying types of equipment has multiple commonly used algorithms.
Self-maintenance, resilient systems and engineering immune systems will all depend on the prognostic and health management system and are thus derivatives of PHM systems. Self-maintenance will use the PHM's health and remaining useful life (RUL) estimations to carry out repairs with spare parts when needed.
Resilient systems can handle dynamic systems which can have varying states and still differentiate between a healthy and deteriorated system.
Engineering immune systems should be able to self-train on new anomalies and adapt such that minimal human intervention is needed.

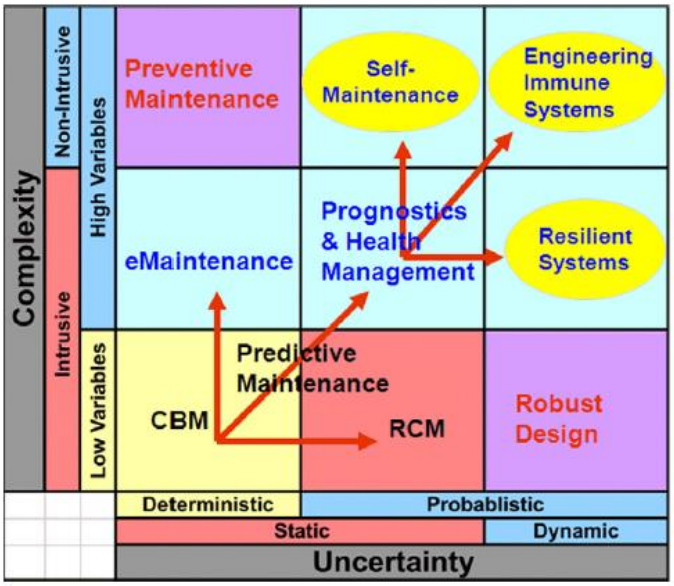


Figure 2-2: *Matrix for deciding type of monitoring approach should be used based on the system which should be monitored* [6].

2.2.2 Diagnosis versus prognosis

These similar words, diagnosis and prognosis, carry a big difference in what type of system that will be made. One word, diagnosis, refer to the act of finding what is wrong at this point by looking at symptoms. A prognosis will look at the current state or symptoms and say something about the future state [6]. This leads to a significant difference which is that time must be modelled in the prognosis, while it is not an issue in diagnosis models.

2.3 Machine learning

With the digital transformation in the industry, large data sets have become available. Meanwhile, recent developments in maintenance methods with data-based approaches have led to being heavily reliant on process data. This transformation brings enormous opportunity into equipment monitoring. There are certain types of machine learning algorithms for predictive maintenance that are commercially in use by companies such as the classification approach and regression approach. Before that, the data has to be handled and prepared for the algorithm and this process is called pre-processing the data.
The following steps represent a data-driven methodology in machine learning applications.
Stored process data is being accessed by PLC or SCADA systems and retrieved data to files.
Before implementing machine learning methods, the data have to be pre-processed. This step includes data visualising and data cleaning besides data transformation. The next step is creating an algorithm model and structure. Figure 2-3 represents a ML method used in predictive maintenance. The algorithm outputs are processed by validation applications. If the results are meaningful, accurate and meet the expectations of the system designer, process integration and monitoring applications are built within the company. However, if the model becomes too complex, then there is a possibility that each data point in the training set will be focused too much, and the model is not generalized well to the new dataset [7].

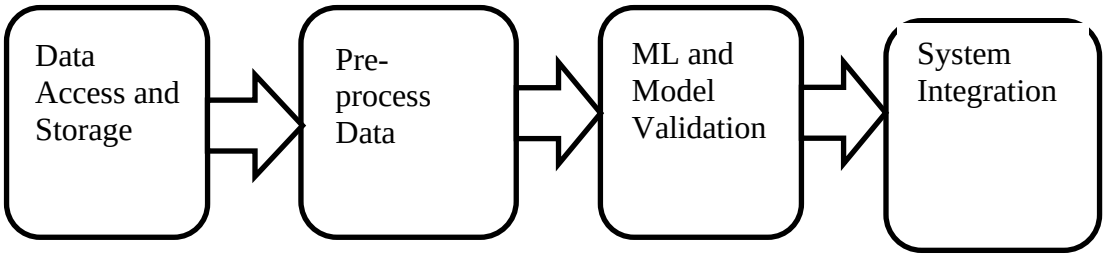


Figure 2-3: *Machine learning methods used in predictive maintenance*

One of the supervised learning methods is the classification approach which predicts whether there is a possibility of failure in the next steps. Classification exploits labelled data to learn a sketch from inputs to learn decision function by class labelling. The most common classifiers are k-nearest neighbours (k-NN), naïve Bayes classifiers, support vector machines (SVMs), decision trees, and random forest [8].
In order to predict how much time is left before the failure, another supervised learning method called regression model is used and this methodology is identified as remaining useful time. Regression use pairs of input and outputs to learn to predict continuous outputs for the new variables.
In unsupervised learning, output data is basically unknown and only input data is given to the algorithm. Even though they are usually harder to understand and evaluate, unsupervised learning methods are also investigated by researchers as labelling singular points and scoring systems may not be enough to predict anomalies in the system [9].
There are more advanced methods in use such as and Long Short-Term Memory Network (LSTM) [10].
In order to analyse the prediction quality of predictive maintenance ML algorithms, Receiver Operating Characteristics (ROC) curve allows users to optimise and compare failure prediction algorithms [11]. ROC curves are well known for evaluation metrics for checking any

classification model's performance. Furthermore, the threshold within the curve provides insights into the model to analyse false positives and false negatives.
Documentation is the key to a predictive maintenance method and machine learning. This is simply because error and maintenance history are pivotal to distinguishing healthy and faulty states when it comes to supervised and unsupervised learning methods.

A practical approach

In practice, noise, outlier, and missing value removal techniques are applied to data because the raw data is messy. However, these techniques may not be enough in the most situation even though they help to reduce overfitting and simplify datasets. In this case, condition indicators are used. During the pre-processing phase, condition indicators transform the raw the data into meaningful and comparable data sets. Condition indicators can be divided into 3 groups. These are time-domain features, frequency domain features and time-frequency domain features. Time-domain features are also divided into subgroups such as mean, standard deviation, root-mean-square, skewness and kurtosis algorithms. In some cases, time series may not be enough to generate classified data. At this point, frequency domain features may be useful distinguish health and faulty states. Some frequency domain features are power bandwidth, mean frequency, peak values, peak frequency and harmonics. More advanced pre-processing models use time-frequency domain features such as spectral entropy or spectral kurtosis [12]. In the next step, it is time to feed the machine learning model with processed datasets. By using labelled data from examples, classification algorithms are a good way to start to determine fault detection.
The remaining useful time allows users to predict the fault from the current condition to the failure condition. When each failure is known from similar machines then, the probability density function may be used with the survival model. If the threshold of fault status is determined by condition indicators, then the degradation model may be run to estimate the remaining useful time [12].

2.3.1 Pre-Processing Methods

Data Wrangling

Data Wrangling is an important step in pre-processing data by transforming raw data to a clean and organised format. Data frames can be both intuitive and versatile and based on rows and columns in a spreadsheet. In particular, each column represents one feature and has an index number. Thus, pulling samples to view small slices provides some insights about overview. Data wrangling also includes renaming columns, replacing values, handling missing values and merging data frames.

Outliers and Detection

An outlier is a variable that differs from known behaviour of the data or far away from the average values.
The incoming data needs to be filtered out of noise, outliers, and errors as well as tied to the real-time scale. Especially, in order to build a regression model, it is vital to exclude outliers from the data set. These outliers need to be classified into data obtained as a result of unreliability or anomalies. Therefore, it is possible to come across some common algorithms for outlier and anomaly detection in the literature [13]. Supervised learning algorithms would

be more effective because the characteristics of known examples could be used to sharpen the search process through more relevant outliers [14]. On the other hand, outlier detection might be beneficial where lack of awareness on the kind of outliers presents in the data.
There are various algorithms and methods which include iterative, metric, clustering, density and distance-based models. Some traditional approaches are box plot and inter-quartile range. Distance-based models do not require prior knowledge of data distribution as statistical methods. Nevertheless, this algorithm needs a threshold value or a parameter for the k nearest neighbour calculation. When input data has varying local densities, then density-based model approach is used. Density-based methods use local outlier factor (LOF) as the density measure. The possibility of whether the data is a local outlier or not determining by LOF method. If some points have a lower density than their neighbours, those are categorized as an outlier [14]. Another type of algorithm is the Isolation Forest which does not implement any distance or density measure.
If data is normally distributed, then the Gaussian distribution algorithm might be an effective method. Multivariate gaussian automatically captures correlations between features and formulated as equation 2.1.

$$p(x;\ \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \quad (2.1)$$

The algorithm finds its distance from every distribution and thus the probability of that point belonging to each cluster. So, if the probability is very low, this implements the data point as an outlier. The constructed method is then applied to the training set.
Some deep learning techniques are also in use such as Autoencoders and Long Short-Term Memory (LSTM) for effective outlier detection [15]. The reason is that many distance-based techniques such as KNNs do not give sufficient results when feature space has a high dimension. That is why high dimensionality has to be reduced.

2.3.2 Principal Component Analysis

Principal Component Analysis (PCA) is a method of reducing the complexity of data without losing information. The method reduces the dimensionality of the data to facilitate data analysis. The first principal component describes the largest part of the variability in the data, while the succeeding components describe a decreasing part of the variability in the data. PCA is a valuable method in describing similarities, differences and patterns in the data. [16]
PCA is a statistical method that can process large datasets, describe and summarize the data in a way that is easier to visualize and present. PCA can help identify correlations in the data, see trends, and identify clusters and outliers. PCA is a great method of extracting important information from the data. [17]

Outputs from a PCA is a score plot and loading plot. Studying the score plot of principal component (PC) 1 and PC2, which are the components describing the highest variability in the data, you will find data rows with similarities grouped together. In the loading plot, we find that variables that correlate are grouped together while variables that have a negative correlation will be placed on opposite sides of the plot.

2.3.3 Classical pattern recognition

Pattern recognition is basically the automated recognition of patterns and regularities in data. In other words, pattern recognition analysis data and tries to identify patterns. This technique makes pattern recognition applicable in many applications such as engineering and scientific disciplines. As it stated in the machine learning chapter, recognition and classification can consist of supervised or unsupervised classification. Typically, the recognition problem is usually presented as either classification or categorization task. Traditionally, there are three major approaches to pattern recognition such as statistical pattern recognition, structural pattern recognition, and neural pattern recognition [18]. Figure 2-5 shows oversimplified procedure of pattern classification procedure.

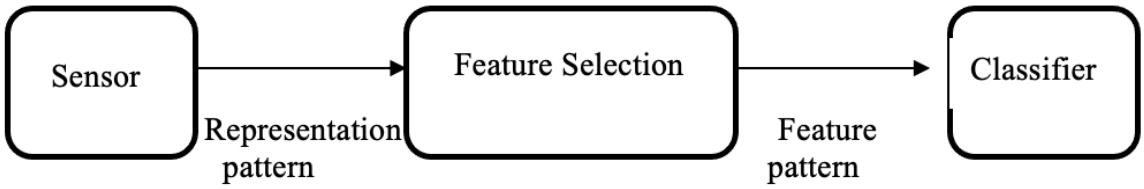


Figure 2-4: *Pattern classification procedure*

Data undergoes several transformations and the number of features is reduced so as to be transformed into a form more appropriate for classification [19]. Having a minimum number of variables are required to capture the structure within the data.
In general, pattern recognition aims to assign classes to objects according to some similarity problems.
One of the statistical pattern recognition techniques is nearest neighbour classification. This method makes predictions for a new data point by considering its closest neighbours in the training set.
Bayes' Theorem takes into account both probability and distribution in the training data sets.
For example; equation 2.2 is for a single feature,

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} \qquad (2.2)$$

The notation $p(x|C_i)$ is being defined as the probability that the class is C_i when the feature is known to have the value of *x*. Basically, the Bayes' rule represents that in order to find the class of an object it is needed to know two sets of information about the objects that might be the basic probability $P(C_i)$ and distribution of values of the feature *x* for every class [19].

Decision Trees are another widely used model for classification and regression. Basically, decision trees learn a hierarchy of if/else questions and conclude a decision. In machine learning, to build a tree, the algorithm looks for all possible tests and finds the one that is most informative about the target variable. Every time the algorithm answers questions, it also creates branches and segments the feature space into disjoint regions. The aim is to continue splitting the feature space and applying rules until there are no more rules to apply or data points left. One of the biggest advantages of decision trees is the resulting model can easily be visualized and understood. Especially, decision trees work well when features are on completely different scales or continuous features. However, they tend to overfit and provide

poor generalization performance [7]. In this case, random forests may be the solution. A random forest is a collection of decision trees, each of it is somewhat different from others. In particular, while each tree may do a decent job of predicting, it will certainly overfit some data. This overfitting can be limited by averaging the results. This strategy is eligible when there are many decision trees. By its nature, different random states can drastically change the model.

2.3.4 Naïve Bayes

The Naïve Bayes' Classifier minimizes computation while keeping adequate classification accuracy. Thus, less amount of storage capacity is needed. It is a simple and effective algorithm for classification. [20] The method relies on the "Naive" assumption of conditional independence between features given the class variable value. [21] There are several different NB classifiers which differ mainly in the assumption about the probability distribution. NB is considered to be a decent classifier method, but performs poorly as an estimator.
Particularly, Naïve Bayes' classifier is used where individual features can be selected. While reducing the number of parameters makes naïve Bayes' classifier less powerful, the right combination of features may result in higher accuracy [19].

2.3.5 Support Vector Machines

Support Vector Machines (SVM) is another statistical method used for classification. SVM creates a line or hyperplane that separates the data into classes. [22] The margin between the classes datapoints and the separator is maximized using supporting vectors nearest the separator at the edge of the class clusters. In most cases classes are not easily separated by a line but can be transformed to separable data by adding dimensions to the data using kernel functions [23]. Possible kernel functions used in SVM is linear, polynomial and Gaussian RBF. The selection of kernel function is crucial to the resulting classification since the kernel selections define in what feature space the data will be classified. Yang and Widodo's study on pattern recognition for machine health diagnosis using SVM showed great potential and showed that Wavelet SVM (W-SVM) reached high accuracy on detection of faults on induction motors using vibration sensors [23].

Support Vector Machine Techniques in Predictive Maintenance

There are some studies in the literature on using SVMs for fault detection. When data only represents normal operating conditions and fault data is not easy to classify, one-class SVM handles the training of the classifier [24]. The generalization performance is high in SVM regardless of prior knowledge about the data and even in high dimension data [25]. A study was reported on using SVM for classification after grouping data to train and test. With the linear Kernel, the SVM classifier trained for the data by using raw data. By using transformed data, another SVM classifier is trained and lower classification errors are obtained [26]. One study found that SVM creates better results than ANN for fault detections [27]. If data has a low dimension feature and the possibility of separating the data linearly is low, then, SVM produces linearly separable data with high dimension spaces which fit better our data sets.

There are 3 built-in kernel functions in scikit learn. Linear, polynomial and radial basis function kernel methods create different level of dimensions and can deliver distinct accuracy [28].

2.3.6 Neural Networks

Recurrent Neural Network

A recurrent neural network (RNN) is a form of artificial neural network that uses sequential data or time-series data. This deep learning algorithm is widely used for problems such as language translation, speech recognition, image captioning, and time series forecasting. Unlike convolutional neural networks (CNNs), recurrent neural networks utilize the training data to learn. Thus, it makes the recurrent neural networks more suitable than CNNs in time series problems [29]. Specifically, while traditional neural network methods consider that inputs and outputs are independent of each other, the output of recurrent neural networks depends on the prior elements within the sequence [30]. Figure 2-5 represents a recurrent neural network model.

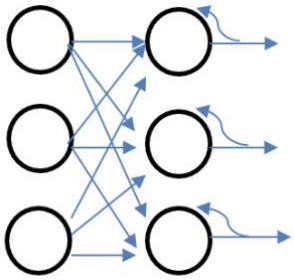


Figure 2-5: *A Recurrent Neural Network Model* [30]

These networks also have additional stored states or hidden layers showed in Figure 2-7. Hidden layers have a loop mechanism that allows information to flow from one step to the next step. In other words, a hidden layer acts as a representation of previous inputs.

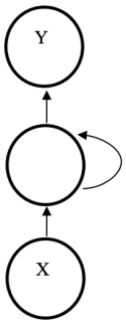


Figure 2-6 *A Recurrent Neural Network with a Hidden Layer*

A distinguishing specific of the recurrent neural network supports backpropagation through time algorithm to determine the gradients. During this process, RNNs tend to create two problems, known as exploding gradients and vanishing gradients. These problems are defined by the size of the slope of the loss function along the error curve. If the gradient becomes too

small, the algorithm is no longer learning due to the risk of becoming insignificant values of updating weigh parameters.
The advantages are the possibility of processing input of any length, model size not increasing with the size of input and weights are shared across time. However, drawbacks are computation being slow, cannot consider any future input for the current state [31].

Long short – term memory (LSTM)

In order to solve the gradient problem that mentioned above, Sepp Hochreiter and Juergen Schmidhuber introduced a new solution called long short -term memory. LSTM can learn to bridge minimal time lags in large time steps by imposing constant error flow through a method called constant error carrousels within special units [32]. Simply, the function is to store a value and determine how long it should be stored. It makes long short - term memory one of the most common models when working with time-dependent data [33]. Proposed multiplicative input gate units protect the memory content stored from perturbation by irrelevant inputs and output gate units protect other units from perturbation by irrelevant memory content which is stored.
In predictive maintenance analysis, the data can be processed by a min-max normalization model to scale attributes to values between 0 and 1 and LSTM may create better results with scaled data. In addition, due to increases in characteristics, Principal Component Analysis (PCA) may help to reduce attributes [33].

2.3.7 Other algorithms

Fourier transform

The Fourier transform is a transformation of the coordinate system from the time-amplitude domain to the frequency-amplitude domain. This allows for an entirely different view of the information in time-series data. The Fourier transform generalizes the Fourier series which allows any function to be transformed into only sines and cosines with varying amplitudes [34]. When extending the time over which the Fourier series looks to infinity, the Fourier transform is born. The Fourier transform can be calculated as equation 2.3.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \quad (2.3)$$

where ω is the frequency.

The real power of the Fourier transform for data analysis comes from the discrete Fourier transform (DFT), which can be calculated as equation 2.4.

$$F(\omega) \approx \sum_{n=1}^{N} f(t_n) e^{-j\omega t} \quad (2.4)$$

and further on from the Fast Fourier transform (FFT) which reduces the computational time from N^2 for DFT to $N\log_2 N$ for FFT. FFT does have a drawback in that it requires there to be a power of 2 samples used.
The Fourier transform can take yet another form, which is the Short-Time Fourier Transform (STFT). The STFT takes small chunks of a signal and transforms each of the chunks to add a temporal value to the Fourier-transformed signal. That is, e.g., instead of transforming one

minute of the signal, one minute may be split into 60 to see the changes in frequencies over the minute. This is especially useful in dynamic signals as implied in dynamic, it changes [6].

2.3.8 Understanding the confusion matrix

Confusion matrixes in other words error matrixes are used for evaluate the performance of the classification model and represent it in a fancy way. Predicted and actual values for each class are represented in Figure 2-8. There are some terms in this matrix and explained below. As an example of 2x2 confusion matrix where rows and columns represent each class.

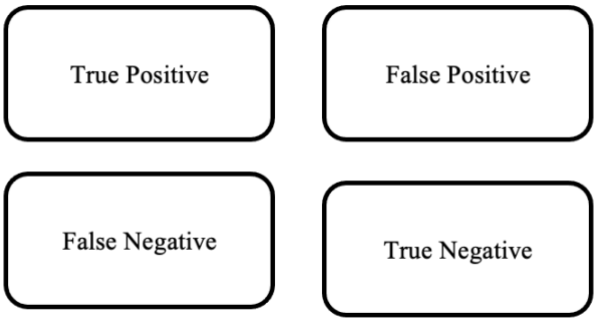


Figure 2-7: *Confusion matrix example*

True Positive: Correctly predicted positive results.
True Negative: Correctly predicted negative results.
False Positive: Incorrectly predicted positive results, those values are actually negative.
False Negative: Incorrectly predicted negative results, those values are actually positive.
For calculations and display, sklearn.metrics confusion_matrix library is used.

2.3.9 Built-in libraries are used in this study: Scikit Learn and Tensor Flow

One of the well-known and open-source libraries is Scikit Learn for machine learning applications. NumPy and SciPy data objects are also supported in this library [35]. It is more convenient to implement supervised and unsupervised network models. Splitting the data into training and test sets comes in handy. On the other hand, TensorFlow is also a widely used library in deep learning algorithms. One of the main advantages is allowing to work on large datasets with relatively less GPU.

2.4 Process Description

This chapter will showcase the process around the pump of interest using P&IDs. Then an introduction to various intervals of data collection before moving to the data itself

2.4.1 P&ID

In Figure 2-8, starting from the upper left there is a buffer tank for the slurry which is being pumped, here there is a flow sensor, FT01, before the pump starts. In the first area, there are 4

pumps to move the slurry between heat exchangers and one backup which can be used by turning the correct valves, these pumps have 5 signals, on and off signal, the control signal in %, speed signal in %, ampere measurement and torque measurement where the first signal comes from the PLC, while the others come from a frequency converter. Most pumps have pressure measurements before and after, except for the first one in this area.
After the slurry has been heated, it is moved into one of three storage tanks, here depicted by E-11 which is a placeholder for all three vessels. These are not believed to have much impact on the pump's health and is thus not explained in detail.
When the slurry is finished from the vessels, it is moved on to the next set of pumps where the slurry should be cooled down before going to the rot tank to produce methane. Here there is another flow transmitter FT02 to measure the flow after the storage tank. Then there are two pumps, with one backup pump in the event that one breaks down. As in the previous pump area, the first pump only has pressure measurements after, while the remaining pumps have pressure measurements before and after.
PU19 has additional measurements, here there are three accelerometers, one is on the bearing house, one on the inlet and one at the outlet. These measurements are processed by an AMS Asset Monitor [36]. The outputs from these measurements give multidirectional velocity and a calculated value called PeakVue which in simple terms, measures knocking inside the pump, for the three points on the pump.

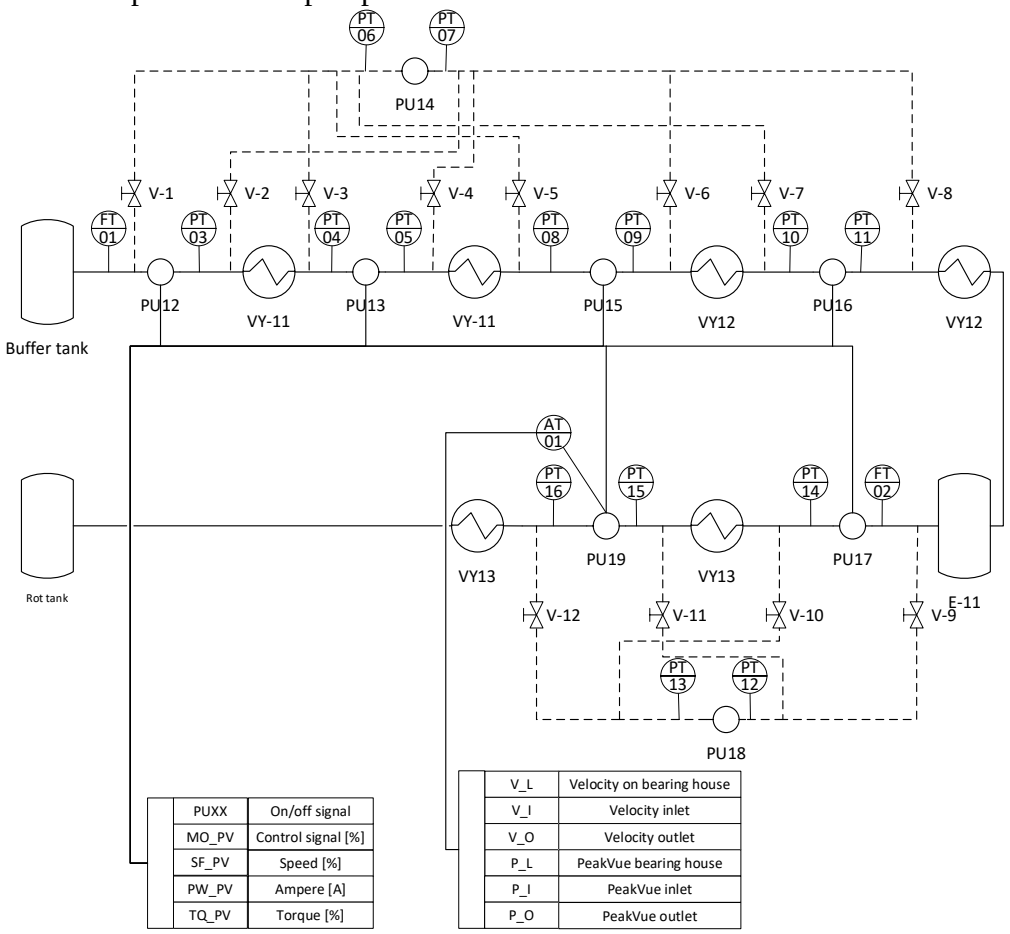


Figure 2-8: *P&ID of the process for around the pumps in question*

The pumps in Figure 2-8 are all of approximately the same type, the only difference being that the first five are made for slightly lower temperature. The main focus will be on the last three pumps which are most often replaced, a larger figure of them can be seen below in Figure 2-9.

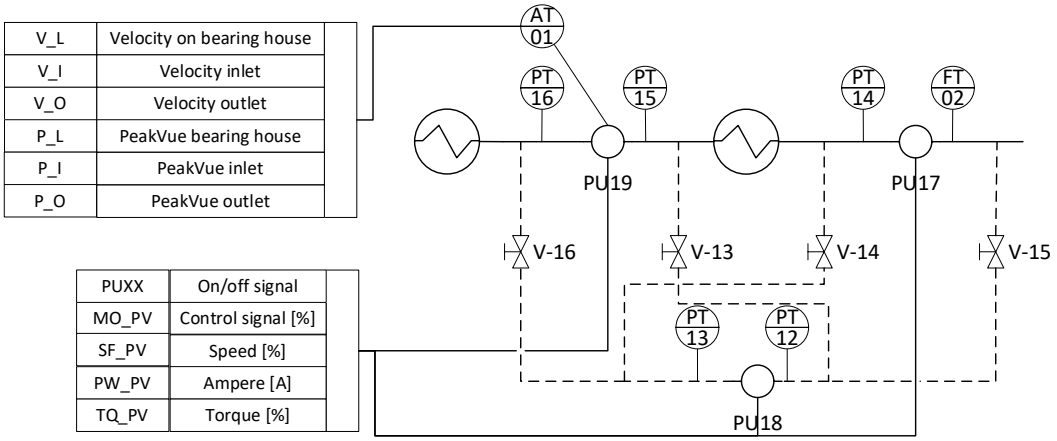


Figure 2-9 *A closer look at the P&ID drawing for the three last pumps*

2.4.2 Sampling intervals

There are three sampling intervals, 30 seconds, 1second and 10 ms. At 30 seconds all the values presented here are sampled except for AT01, the AMS Asset Monitor, the AT01 measurements are sampled every second, and the on/off signal for the pumps which are logged whenever they change.

At one second the focus is on PU19 to get the best sampling rate for the AMS values which are updated every second, in addition to the AMS values, pump control signal, speed, ampere and torque are stored, as well as PT15 and PT16.

At the lowest level 10ms, all control signals, speeds, ampere, torque, flow and pressure are stored.

To reduce the amount of data to be processed (unnecessarily) the 10ms data has been reduced to 50ms interval by means of average. While the data is taken from the PLC at a 10ms interval, the PLC does not update the data as often. *Table 2-1* shows the average update time from an arbitrary hour of the 10ms dataset. This shows that it would be unnecessary to process all this data as there would be at least 7 redundant values for each update.

Table 2-1 Update rates for the 10ms dataset for PU19

ft02	0:00:00.288012
pt15	0:00:00.496314
pu19_mo	0:00:00.432347
pu19_pw_pv	0:00:00.072106
pu19_tq_pv	0:00:00.072564
pu19_sf_pv	0:00:00.518926
pt16	0:00:00.482217

2.5 Introduction to the data

To start off, the data will be described by simple statistics, as histograms and in form of correlation plots. As mentioned, the focus will be on PU19 as this is the pump with most measurement signals. The code for the database connection and queries can be seen in Appendix B, code for putting the data together in Appendix C, date list manipulation in Appendix D, plotting functions in Appendix E, and finally Appendix F which calls the methods needed for the plots in chapter 2.5.2, 2.5.3 and 2.5.4 the plots for chapter 2.5.5 can be seen in Appendix H.

2.5.1 Output classes

The data will be introduced in the various sampling intervals. The following plots are colour coded. As there are no distinctive data to tell the state of the pump, the data has been marked for times less than 1h before replacement stop, less than 24h before replacement, less than 1 week before replacement, anything else is considered normal operations except for when it's stopped these states will be used as output classes for the machine learning models and are further described in *Table 2-2*. It is here assumed that the degradation is not instantaneous, but rather that it degrades over time due to bearing faults, seal faults or other mechanical parts.

Table 2-2: *Description for color-coding in scatter plots and PCA*

State	Color	Description
0	Dark blue	Stopped
1	Purple	Normal running
2	Light red	Less than 1 week before failure
3	Orange	Less than 24h before failure
4	Yellow	Less than 1h before failure

As there is too much data to fit into one plot (especially for the correlation plots with higher granularity) with the computers available, whole datasets are not considered here. Instead, a subset of each hour is taken, this is specified for each interval level where applicable. In addition, the data is from here on split into training and test sets where 70% of the data is used for training, and thus inspected.

2.5.2 30 second interval

This data is sampled at a rate of 120 samples per hour, this is a frequency which doesn't take very much storage, and all the data from May 2020 to September 2021 is fitted into the plots. This means that it incorporates the greatest number of faulty pumps which is an advantage compared to the datasets with higher sampling frequency.
Table 2-3 shows some basic statistics on the dataset, it consists of just above 900 000 samples after taking out the test set of 30%. Some of the maximums appear to be quite high comparing to their means and quartiles. This is especially noticeable at PU19_TQ_PV which is at 296% momentum and thus close to three times larger than the intended value.

Figure 2-10 show histograms for each measurement around HYG_PU19 which is measured at 30 second interval. The 1st subplot, meas_time, shows the occurrence of measurements. This shows where data is sparse in a timeline. The 2nd subplot shows the ampere usage of the pump (PU19_PW_PV) there seems to be two peaks in the distribution except for the zero peak. The 3rd shows the momentum (PU19_TQ_PV), here there are three peaks. These two measurements may point to variations in pump types; however, it could also be due to variations in the fluid going through the pump. The 4th plot shows the control signal to the pump (PU19_MO), it appears to be close to normally distributed, but with high peaks. This can in at least some cases be due to the operators taking manual control of the controller and fixes the control signal. The 5th plot shows the speed of the pump (PU19_SF_PV) it appears to be similar in shape to the control signal, this makes sense as the speed is a function of the control signal. The 6th plot shows the flow before the pump, the peaks lie around 17m³/h with a semi smooth distribution. The 7th subplot shows the inlet pressure which is controlled to 1 bar, in a period the pump before (PU17) was faulty and was not able to deliver enough pressure, and thus it stayed around zero for a long time. The 8th subplot shows the outlet pressure distribution, it is for the most part smoothly distributed around 5 bars. The final subplot shows the occurrence of the states from *Table 2-2*.

Table 2-3 Descriptive statistics

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16
count	918645	918645	918645	918645	918645	918645	918645
mean	11.80410561	29.28553451	71.35552905	58.94030983	16.03986763	0.822674394	4.636202852
std	4.907181148	17.64759575	9.141768376	20.50914369	3.875615368	0.594410695	1.512648989
min	0	-36.2	0	0	0	-1	-1
25%	9.75	16.2	67.9052	61.1375	16.38	0.830329	4.04688
50%	13.22	30.2	71.4673	64.6952	16.9374	0.98758	4.87188
75%	15.2	43.8	76.1372	68.502	17.4449	1.01863	5.57188
max	52.47	296	100	119.997	50	13.152	21.9802

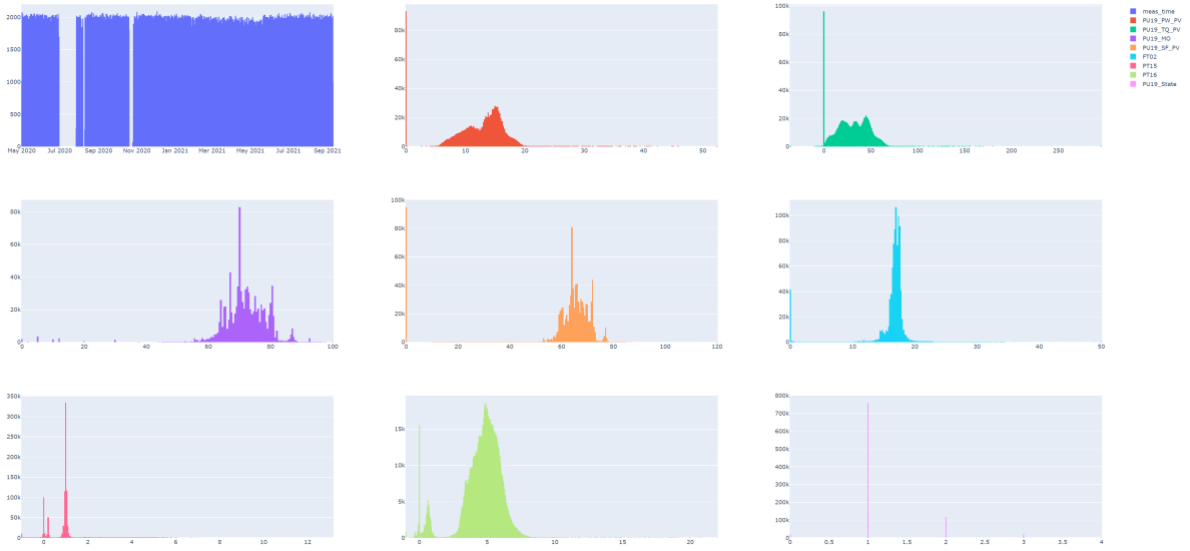


Figure 2-10 Histogram for period 2020-05-01 to 2021-09-15

Table *2-4* indicates that only the power usage (PU19_PW_PV), the torque (PU19_TQ_PV) and the speed of the pump (PU19_SF_PV) with the power usage significantly correlate. This is further shown in Figure 2-11 where it isn't visible that any of the measurements correlate. It is neither visible that the colours form any significant groupings. There are some groupings, such as in in PT15 and PT16, however they are not consistent.

Table 2-4 Correlation matrix

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16
PU19_PW_PV	1	0.801511987	0.372170514	0.772910386	0.175444502	0.082542924	0.440451029
PU19_TQ_PV	0.801511987	1	0.329338636	0.53178375	0.109797981	0.192542352	0.425004956
PU19_MO	0.372170514	0.329338636	1	0.462219847	0.196120819	0.186187391	0.311669285
PU19_SF_PV	0.772910386	0.53178375	0.462219847	1	0.290547346	-0.059409088	0.402644323
FT02	0.175444502	0.109797981	0.196120819	0.290547346	1	0.223743784	0.370509707
PT15	0.082542924	0.192542352	0.186187391	-0.059409088	0.223743784	1	0.393270882
PT16	0.440451029	0.425004956	0.311669285	0.402644323	0.370509707	0.393270882	1

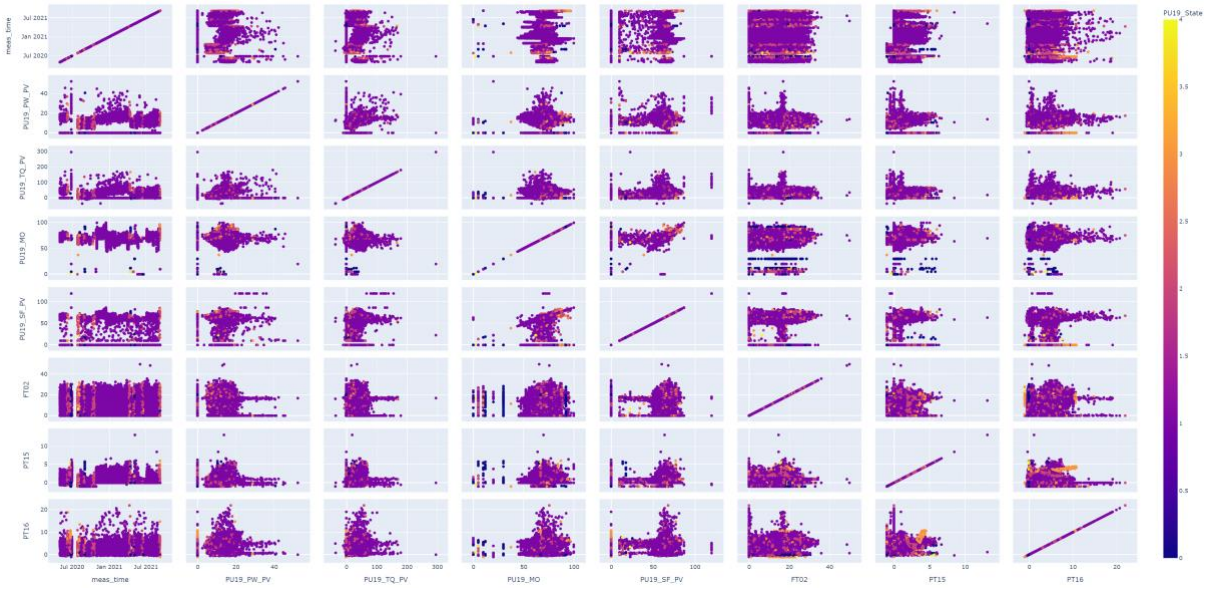


Figure 2-11 Scatter matrix plot for period 2021-05-01 to 2021-09-15

2.5.3 1 second interval

Here the data which is sampled every second will be investigated. This will be done through the use of histograms, scatter matrix plots and with some statistics.
This dataset was started in May 2021, and thus has fewer faulty pumps than the preceding dataset. It may however hold important data. It covers all the sensors earlier mentioned and in addition three vibration sensors which give out calculated values every second. These represent 3-dimensional vibration on the bearing house (PU19_V_L), at the inlet (PU19_V_I) and at the outlet (PU19_V_O) in mm/s. They also give out a value which measures impacts in g (gravitational pull) at the bearing house (PU19_P_L), inlet (PU19_P_I) and outlet (PU19_P_O). These may give new insights into the state of the pump.
The data presented here shows 10minutes of every hour from 1st of June to 9th of September. Table *2-5* shows some statistics of the dataset for the vibration data, the sample set consists of 990297 samples for each variable after taking out 30% for test set, the means are very low

comparing to the max value, which is further made visible by the quartiles. The statistics for all the sensors can be found in Appendix H.

Table 2-5 Statistics for vibration (1s interval) data

	PU19_V_L	PU19_V_I	PU19_V_O	PU19_P_L	PU19_P_I	PU19_P_O
count	990297	990297	990297	990297	990297	990297
mean	1.595377552	1.541178036	1.440449513	0.668973969	7.926183562	0.733070915
std	18.02307949	15.76758003	13.36994915	0.506189405	5.728567057	0.605381431
min	0	0	0	0	0	0
25%	0.777806	0.880416	0.898132	0.525112	5.6409	0.535751
50%	1.11986	1.17746	1.16671	0.632766	6.83131	0.627473
75%	1.72832	1.49572	1.46378	0.766576	9.02625	0.819968
max	2598.51	2830.47	2607.29	93.75	820.962	93.75

The time index meas_time in Figure 2-12 shows that there is a gap in sampling from 16th of June 09:00 to 17th of June 21:00. The power usage is quite similar to the previous 30 second data, the torque differs quite significantly, as it has become bell shaped and doesn't have the three peaks as in the previous histogram. The control signal and speed are quite different as well, here they are more spread and with high peaks, this is due to a smaller time frame and high sampling frequency. The inlet pressure is quite the same, while the outlet pressure is narrower and is likely due to the smaller timeframe. The new vibration measurements show that normally, the vibration in mm/s for all three sensors (PU19_V_L, PU19_V_I, PU19_V_O) are typically in the range of 0 to 2 and the PeakVue (impact) values (PU19_P_L, PU19_P_I, PU19_P_O) are typically in the range from 0 to 8. From the plots it is clear that the measurements vary a lot and some of them might be considered outliers as it is simply not realistic with movement of 2830mm/s as in the case of PU19_V_I. The last subplot indicates the created values from *Table 2-2*, this is the assumed state of the pump based on time before replacement, as expected there are fewer and fewer values when it increases as it is less samples in the last hour compared to the last 24hours before replacement etc.

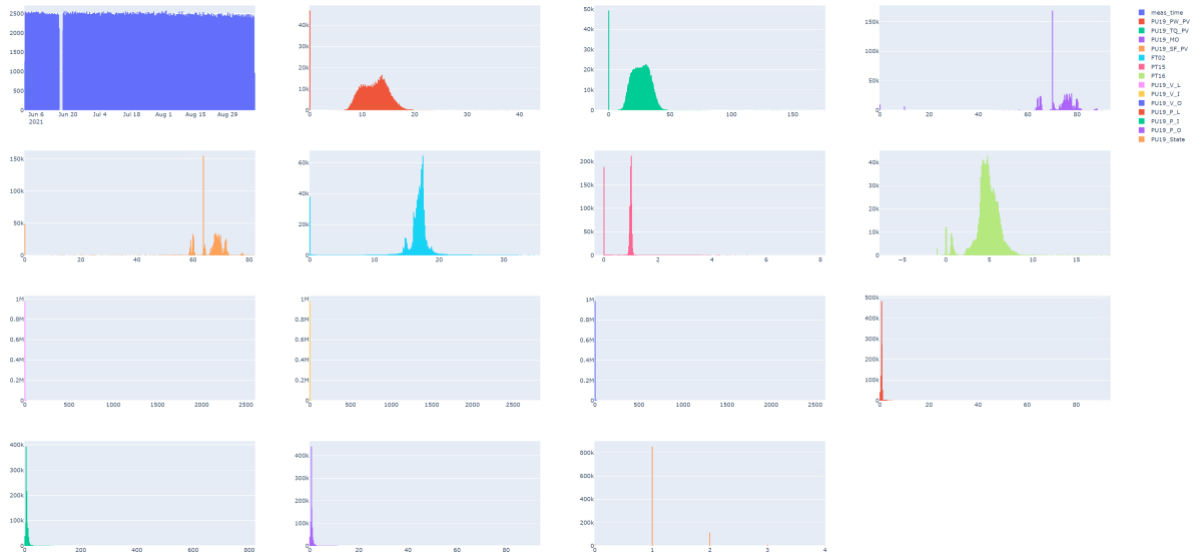


Figure 2-12 Histogram from 1s interval data

The scatter matrix plot in Figure 2-13 shows all the variables in the 1 second interval data plotted against each other. In the leftmost column, the measurements are plotted against time, from this one can take out that there have been two replacements in the period as there are two lines of light red, the colour of 1 week ahead of replacement in the time series. It is also noticeable that the vibration sensors go to high values at times long before replacements, which leads to the question of why this is happening and are these useful for estimating the state? In addition, the inlet pressure PT15 is very low at some points, this has at least in the period of 21st June to 5th July been shown to be caused by the preceding pump PU17 not being able to give enough pressure and is not a cause for alarm to PU19 (other than possible wear and tear). While it is not useless, this feature will be saved for the expansion of this system where all pumps should be surveyed. Figure 2-13 can be seen in increased size in Appendix G.

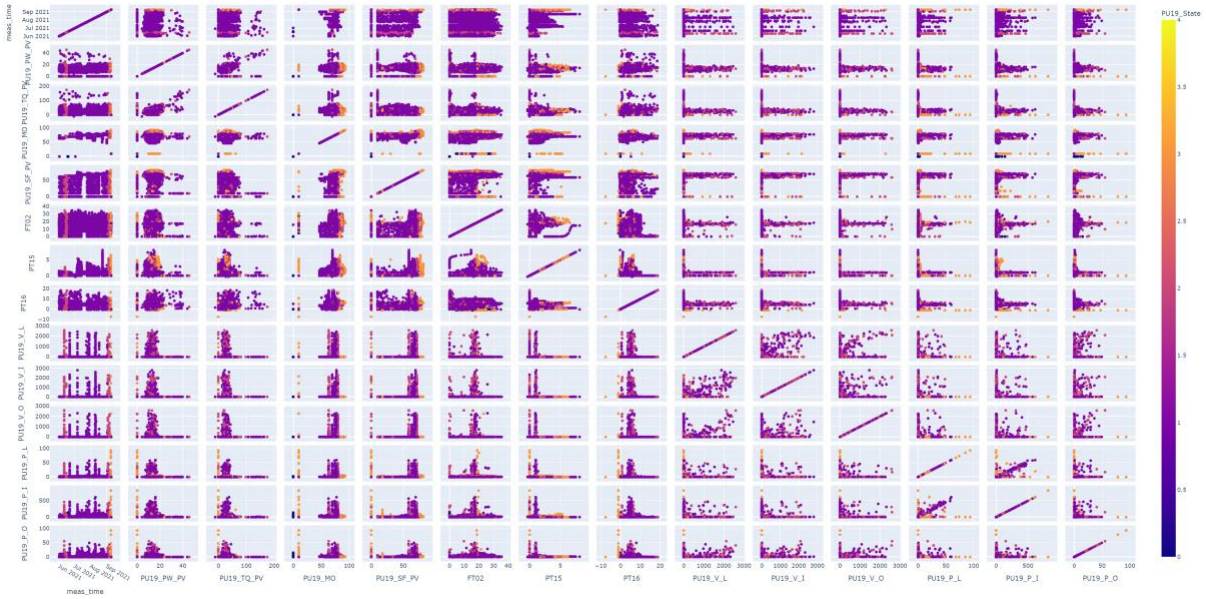


Figure 2-13 Scatter matrix plot for all the values of 1 second data set

From the scatter plots it may be useful to investigate the scatters which seem to have concentrations of red, orange and yellow further as this is clusters where the pump is ending its lifetime as explained in *Table 2-2*. This appears to be the case for PU19_PW_PV, PU19_TQ_PV, PU19_MO, PU19_SF_PV, FT02 and PT15. A correlation matrix of the variables can be seen in Table 2-*6*. Here, the highest correlation is at 0.72 between the power consumption and the pump speed, PU19_PW_PV and PU19_SF_PV respectively. These should be kept in mind in further investigations.

Table 2-6 Correlation matrix

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16	PU19_V_L	PU19_V_I	PU19_V_O	PU19_P_L	PU19_P_I	PU19_P_O
PU19_PW_PV	1.000	0.567	0.459	0.725	0.491	0.314	0.471	0.010	0.014	0.015	0.215	0.248	0.270
PU19_TQ_PV	0.567	1.000	0.396	0.643	0.422	0.283	0.431	0.009	0.012	0.013	0.195	0.247	0.268
PU19_MO	0.459	0.396	1.000	0.627	0.411	0.355	0.358	-0.001	0.005	0.006	0.263	0.196	0.169
PU19_SF_PV	0.725	0.643	0.627	1.000	0.685	0.400	0.548	0.009	0.015	0.016	0.265	0.238	0.258
FT02	0.491	0.422	0.411	0.685	1.000	0.394	0.545	0.005	0.009	0.010	0.246	0.210	0.134
PT15	0.314	0.283	0.355	0.400	0.394	1.000	0.332	-0.024	-0.023	-0.021	0.203	0.276	0.252
PT16	0.471	0.431	0.358	0.548	0.545	0.332	1.000	0.010	0.011	0.012	0.185	0.216	0.166
PU19_V_L	0.010	0.009	-0.001	0.009	0.005	-0.024	0.010	1.000	0.708	0.455	0.094	0.082	0.147
PU19_V_I	0.014	0.012	0.005	0.015	0.009	-0.023	0.011	0.708	1.000	0.402	0.105	0.088	0.144
PU19_V_O	0.015	0.013	0.006	0.016	0.010	-0.021	0.012	0.455	0.402	1.000	0.099	0.087	0.151
PU19_P_L	0.215	0.195	0.263	0.265	0.246	0.203	0.185	0.094	0.105	0.099	1.000	0.646	0.253
PU19_P_I	0.248	0.247	0.196	0.238	0.210	0.276	0.216	0.082	0.088	0.087	0.646	1.000	0.479
PU19_P_O	0.270	0.268	0.169	0.258	0.134	0.252	0.166	0.147	0.144	0.151	0.253	0.479	1.000

2.5.4 50 millisecond intervals

Here the aggregated data from the 10ms interval is represented. The data is reduced to 1minute per hour in the period of 1st of July to 9th of September to allow it to fit in memory for the following plots. This may seem to be little representative with a first glance, especially as there is a stop each hour for the pump, however, this stop is not perfectly at one hour and variables such as other planned or unplanned stops in the process does not follow this scheme, it is believed that this is sufficient to demonstrate the data. Table *2-7* shows some descriptive statistics of the data set; count of samples, mean, standard deviation, min, max and the quartiles for each variable.

Table 2-7 Statistics for 50ms data

	pu19_pw_pv	pu19_tq_pv	pu19_sf_pv	pu19_mo	pt15	pt16
count	1374553	1374553	1374553	1374553	1374553	1374553
mean	12.23684988	26.27462568	65.47286829	74.78337372	0.919407702	4.634274193
std	3.277832416	8.25088209	13.65883321	7.892697231	0.306410377	1.286166084
min	0	0	0	10	-0.073117	-1
25%	10.77	21.9	67.1719	74.3274	0.973558	4.16042
50%	12.64	27	68.4832	75.984	0.997596	4.68125
75%	14.2725	31.7667	69.8555	77.7512	1.01562	5.38229
max	40.84	158.2	80.129	91.0786	5.28546	16.2083

Figure 2-14 shows the histograms for each of the sensors for PU19, from the time histogram it can be noted that, some time frames are missing some values now and then and especially at the end of August, there are over one day missing. It should also be noted that this data is not in the same time frame as the 1s data as there are no data for 50ms before July.
As has been seen in the preceding subchapters on 30s interval and 1s interval, there are still lots of measurements in all the plots where the measurements are zero and that this is expected. The granularity has increased again and is visible over the spectrum of measurements as smoother curves, especially in PU19_PW_PV and PU19_TQ_PV. The high concentration around 70% in PU19_MO has become more distributed comparing to the 1s data this also goes for the speed (PU19_SF_PV) where the bulk used to be at 62 and now is more spread. There

also appears new details for PT16, e.g., where the dip just above 5 Bar has become larger. The flow of FT02 has also become smoother around 18m³/h.

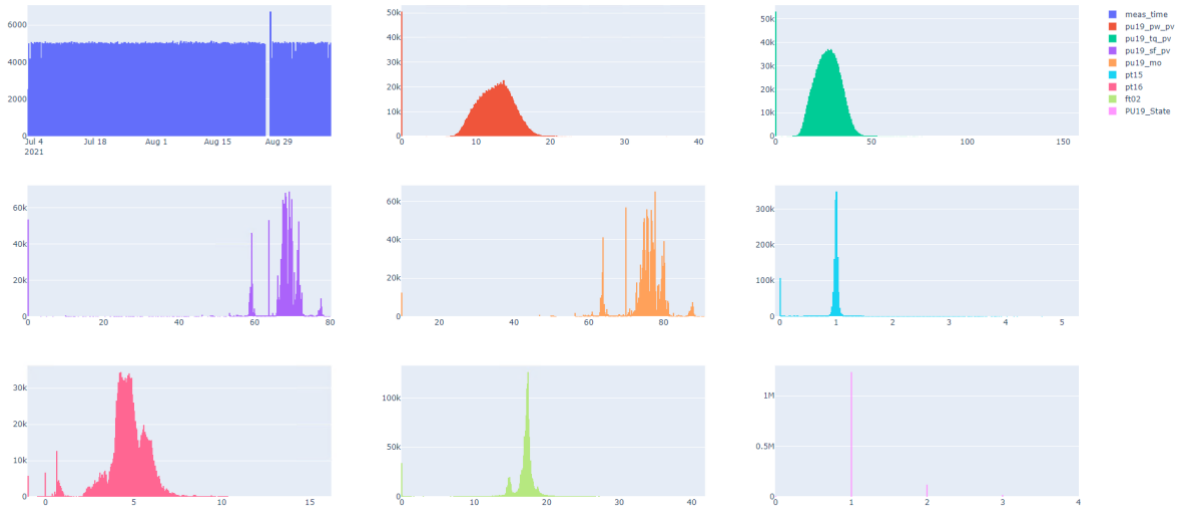


Figure 2-14 Density plots of the data around PU19

Table 2-8 shows the correlation factors between all the variables in the 50ms data. From this it appears that the highest correlation is between speed, PU19_SF_PV, and power usage, PU19_PW_PV, although it is not very high. This seems counterintuitive to what can be seen in Figure 2-15 where the power usage and torque seem to align best.

Table 2-8 Correlation matrix for 50ms unprocessed data

	pu19_pw_pv	pu19_tq_pv	pu19_sf_pv	pu19_mo	pt15	pt16
pu19_pw_pv	1	0.613685979	0.710494337	0.402248993	0.419452844	0.53502777
pu19_tq_pv	0.613685979	1	0.637054218	0.341468416	0.371964338	0.498969541
pu19_sf_pv	0.710494337	0.637054218	1	0.588121173	0.614416499	0.673703768
pu19_mo	0.402248993	0.341468416	0.588121173	1	0.396279628	0.441913858
pt15	0.419452844	0.371964338	0.614416499	0.396279628	1	0.390872582
pt16	0.53502777	0.498969541	0.673703768	0.441913858	0.390872582	1

In the figure below there are some clusters at the end of a lifetime in some of the subplots. It can clearly be seen in PU19_MO and PU19_SF as these increased at the end of the lifetime. There can be at least two reasons either, it is required due to the pump degrading, or the flow increases. From the flow (ft02) in the leftmost column, it does not appear to increase significantly which may indicate that the control signal has to increase to keep the same pressure.

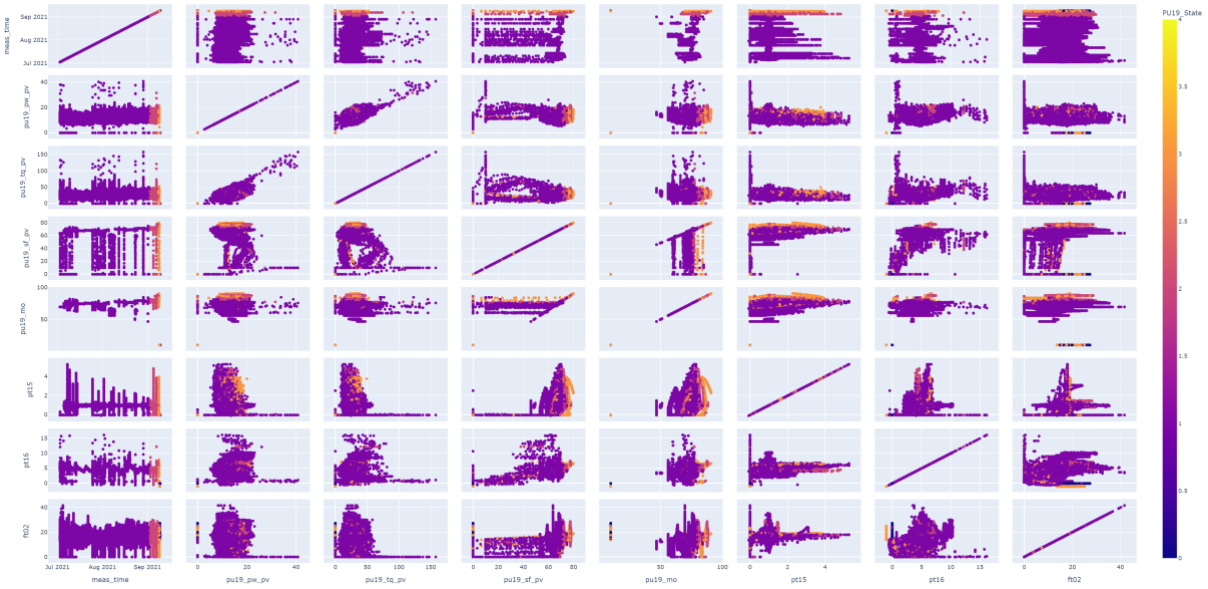


Figure 2-15 Scatter matrix plot for data around PU19

2.5.5 Time-series 50ms data

Some information has been gathered from time-series data, when inspecting the torque at the end of a lifetime and at the beginning, it is clear that it is increasing, which may give a good indication of the state of the pump which can be seen in Figure 2-16. The left plot shows the torque just before the pump is shut down for replacement. The amplitude of the signal is approximately ±12.5 of its mean on the leftmost plot, while in the rightmost plot it varies approximately ±3 from the mean.

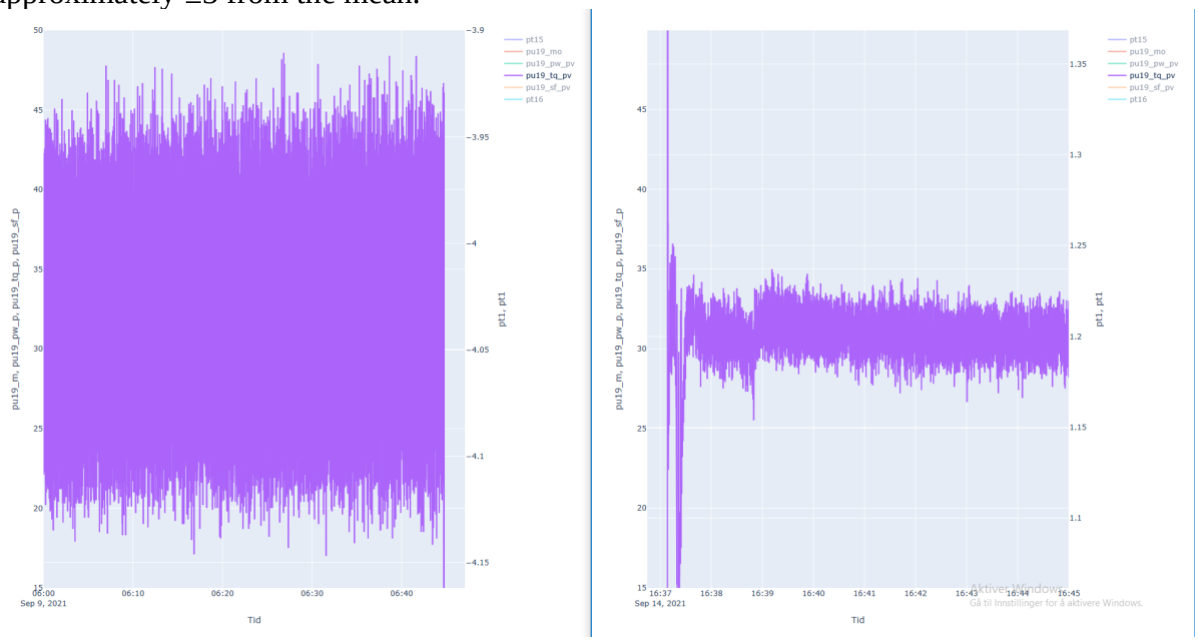


Figure 2-16 Decrease in torque after replacement of pump

Figure 2-17 shows another plot zoomed more in to give a feeling of how the torque signal is varying. It is here visible that the signal changes it's frequency, it is although without doubt

aliasing included here as the equipment used for sampling doesn't allow for high enough sampling rate. Although aliasing occurs, it may be useful to investigate this signal using a Fourier transform to see how it changes over time using the Short-Time Fourier Transform. While the exact frequencies would be the best, an aliased signal found to be associated with a fault can in the future too be associated with a given fault.

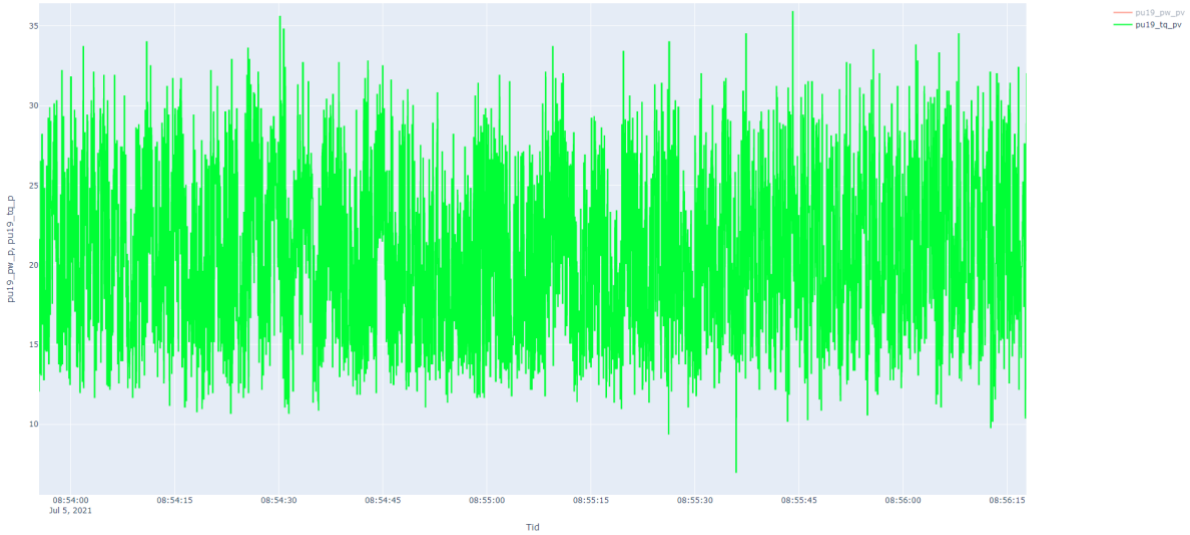


Figure 2-17 Time-series of 50ms data

2.6 Outlier removal

Before diving deeper into the data, the data may be affected by noise as is typical in general for electric analogue signals, there may be more correlation than it appears at first glance. There are two ways out of this, through removing outliers, or using average on short timespans.
This brings up the question of what should be considered an outlier? As has been discussed already, a value of zero in the power consumption (PU19_PW_PV) is not an outlier, it is a part of the process, high values in power and torque appear at start-up and are not anomalies which "shouldn't" happen. Using a Z-score outlier removal would likely remove vital data for these measurements.
The other methods were investigated such as distance and density-based models. In the pre-processing chapter, detailed information was given on these methods and concluded that studying KNN for distance-based models, and DBSCAN for density-based models. In KNN method, the distance between each point and the k-centre is calculated automatically. In this method, setting the number of clusters and determining the cut-off value is the challenging one. Standardization must be done as data should be equally scaled. In order to determine the method efficacy for our data, small data set is studied as a representation and the code was given in Appendix I.
Figure 2-18 shows PU_19_PW_PV and PU19_SF_PV were investigated with KNN method result. During trials, different cut-off numbers were set out, however, there are some values in the outlier class that might be useful for pump health. Even though, it appears that KNN catches some points, these red values might represent anomalies which are highly possible.

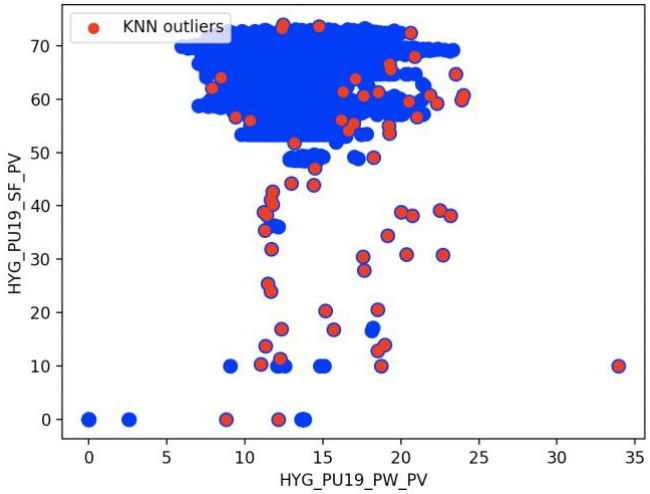


Figure 2-18: KNN outlier results

Likewise, distance-based methods, the data is caught by density-based algorithms might be anomalies which we want to keep in our dataset.
Removing some highest or lowest numbers from the dataset may be beneficial for excluding pump peak data. This idea was also investigated in Figure *2-19*. It is evident that the data distribution is crucial to apply this method. For instance, for the same dataset removing top %1 of each column keeps much of the possible anomaly data.

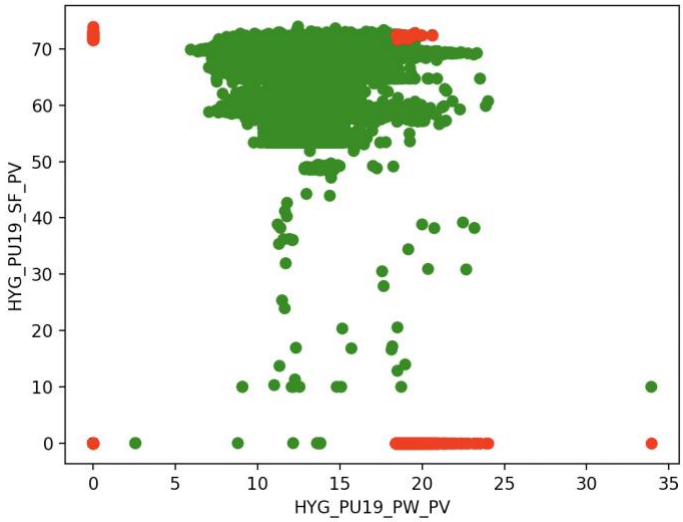


Figure 2-19:Pump 19 Speed and Power data

If data has normal distribution and some values are far from the densely data, then, this method might work, however, for all data in our datasets we found this method unsustainable except vibration data which has the most peak values. Possible working condition for this method shown in one specific example in Figure 2-20. Red values were determined when the top %0.001 of values was chosen.

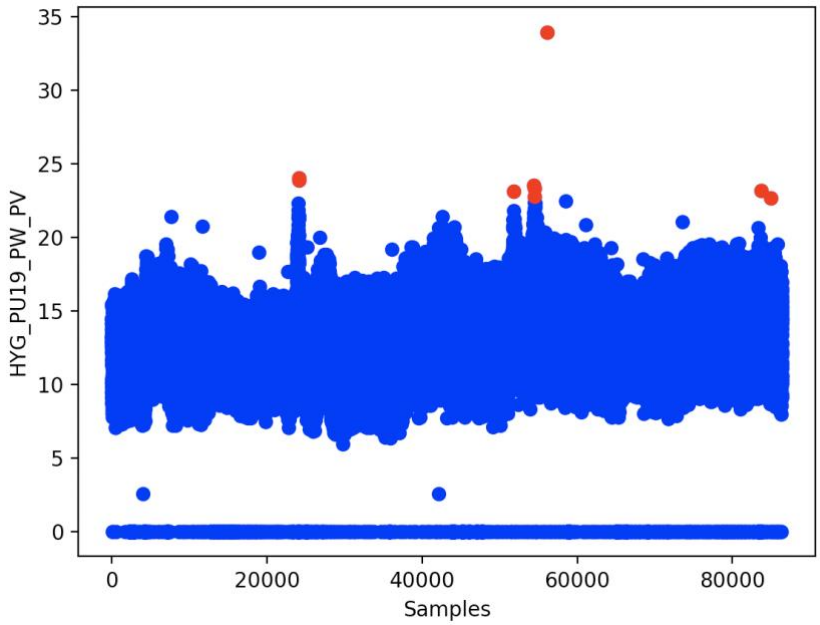


Figure 2-20: Top %0.001 values detected

The very high measurements from the vibration data may be fitting for inter-quartile outlier removal as these are known to be unrealistic high and occur very few times as can be seen in Figure 2-12 and Table *2-5*.

2.7 Using averages and quantiles to remove noise and outliers

Average can be a very powerful tool to remove noisy measurements and may give smoother curves. It allows for less computation in a model as there will be fewer samples to run through it. Here TimescaleDB's time_bucket has been used to average the datasets [37].
This project has decided to use averaging on 20 samples at a time, which means that the 30 second data will have one sample every 10min, the one second data has one sample every 20s and the 50ms data has one sample every second. This also allows for more data per hour to be included for plots and may show a different picture.
Using the average to show correlations may be perilous as it may skew the correlations as explained by the Simpson's paradox [38]. The issue often occurs when analysing datasets with lurking variables, there is at least one lurking variable here, namely the density of the fluid going through the pumps. However, it is expected that the power usage and momentum are related as the momentum is a calculated value of power. It does make sense that the power usage increases as the control signal to the pump increases and as the speed increases. It would also make sense that the outlet pressure (PT16) correlates to some degree with the control signal and thus, the other pump measurements. As PT15 is controlled by this pump (set point at 1 Bar), it makes sense that this doesn't correlate very well.
This chapter also shows the difference after removing high values for the vibration data as explained in chapter 2.5.5 where the Python library Pandas is used [39].
In addition, datasets where the data which is related to the pump being off will be shown, to do this, the Boolean pump signal is used and is done in Pandas. This will have the effect of

removing samples where the backup pump is running which would give e.g., flow, pressure before and after the pump values when the pump is off. The code for this can be seen in Appendix K.
The code for the plots for chapter 2.7.1, 2.7.2 and 2.7.3 can be seen in Appendix L, this code also covers the PCA in chapter 3.1.1.

2.7.1 30 second averaged to 10 minutes

In this subchapter the data is taken in the time range 01.05.2020 to 15.10.2021, the dataset is described with simple statistics in *Table 2-9*. The number of datapoints have been decreased comparing with the previous run due to the averaging of the dataset, as the time-range is longer in this dataset, it is not quite 20 times less samples. The means, standard deviation and maximum values have generally been reduced.

Table 2-9 Descriptive statistics of 30 second dataset

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16
count	49738	49738	49738	49738	49738	49738	49738
mean	11.558	28.553	70.635	58.283	16.113	0.814	4.524
std	4.478	16.104	12.515	19.697	2.547	0.548	1.470
min	0.000	-0.780	0.000	0.000	0.000	-1.000	-1.325
25 %	9.688	16.290	67.864	59.357	16.070	0.700	3.919
50 %	12.195	27.115	71.732	64.098	16.828	0.993	4.724
75 %	14.730	42.050	76.957	68.381	17.344	1.002	5.434
max	26.417	96.035	100.000	101.398	21.851	4.953	10.502

After an average filter of 20 samples per new datapoint, many of the variables seem to correlate in a higher degree as seen in Table *2-10*.

Table 2-10 Correlation matrix after averaging

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16
PU19_PW_PV	1.000	0.918	0.532	0.842	0.309	0.192	0.679
PU19_TQ_PV	0.918	1.000	0.408	0.598	0.193	0.259	0.564
PU19_MO	0.532	0.408	1.000	0.654	0.181	0.322	0.536
PU19_SF_PV	0.842	0.598	0.654	1.000	0.449	0.081	0.648
FT02	0.309	0.193	0.181	0.449	1.000	0.212	0.355
PT15	0.192	0.259	0.322	0.081	0.212	1.000	0.403
PT16	0.679	0.564	0.536	0.648	0.355	0.403	1.000

The histograms in Figure 2-21 shows the averaged dataset, here no data is removed, just averaged. Comparing to the earlier histogram in Figure 2-10, generally the highest values has been reduced and the distributions look wider. This is mostly due to the lower range of the variables. The power (PW_PV) and torque (TQ_PV) seems to have several normal distributions next to each other. The control signal (MO) is close to normal distribution around 70, but also has spikes around 10, 20 and 100, these are now more visible than earlier. This is likely because they are set manually instead of the controller being in automatic and have been at manual for some time causing the averaged values to stay in the vicinity instead of being

smoothed out as other samples have been. The speed (SF_PV), flow (FT02), inlet pressure (PT15) and outlet pressure (PT16) look very similar to earlier, somewhat gaussian.

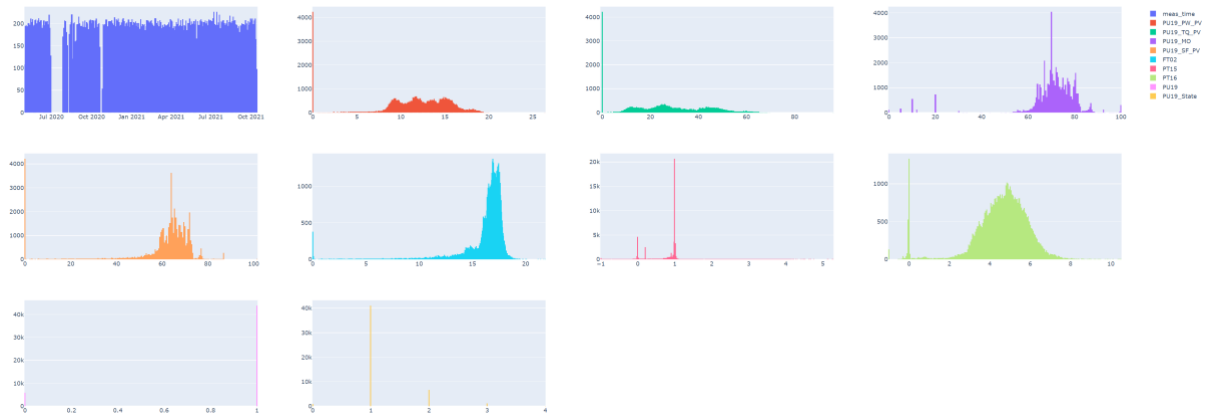


Figure 2-21 Histograms for 30 second dataset before removal of samples where the pump is off

The correlations have increased, and the maximum values has decreased as previously seen, how does this change the scatter plots? The scatter plots can be seen in Figure 2-22, while it may seem cleaner, some of the plots such as between torque (TQ_PV) and speed (SF_PV) show that the later stages of a pump may be seen with higher speed signals, however this is still not consistent. It can be noted that both the speed and control signals typically are high at the end of a lifetime. This is likely caused by the pump having to increase the control signal to keep the inlet pressure at 1 bar.

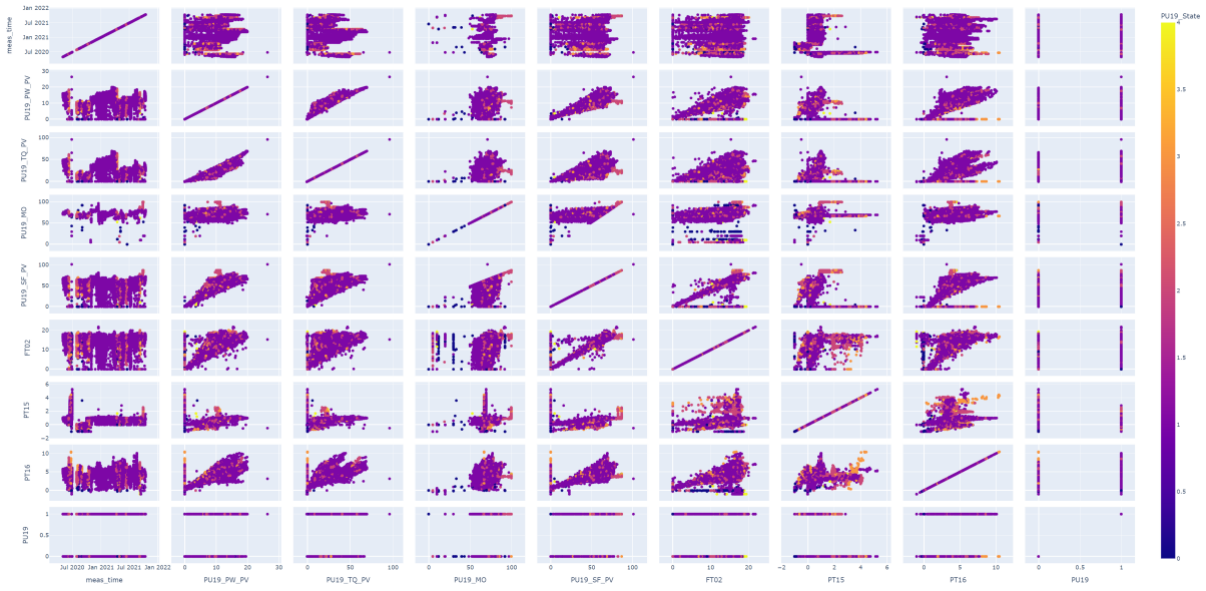


Figure 2-22 Scatter matrix plot for 30 second data after averaging

When the samples where the pump is off has been removed there are 43849 samples left, the correlation rises as can be seen in *Table 2-11*.

Table 2-11 Correlation matrix for 30 second data after removing samples where the pump is off

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16
PU19_PW_PV	1.000	0.949	0.317	0.505	0.380	0.563	0.517
PU19_TQ_PV	0.949	1.000	0.251	0.316	0.169	0.518	0.435
PU19_MO	0.317	0.251	1.000	0.705	0.458	0.343	0.149
PU19_SF_PV	0.505	0.316	0.705	1.000	0.892	0.422	0.395
FT02	0.380	0.169	0.458	0.892	1.000	0.269	0.423
PT15	0.563	0.518	0.343	0.422	0.269	1.000	0.378
PT16	0.517	0.435	0.149	0.395	0.423	0.378	1.000

After the samples where the pump is off have been removed, what is left is shown in Figure 2-23, the number of values around zero has been considerably lowered from Figure 2-10 and there are more holes in the time subplot. The distributions for power and torque are considerably more visible, yet there are still values where they are set to zero.

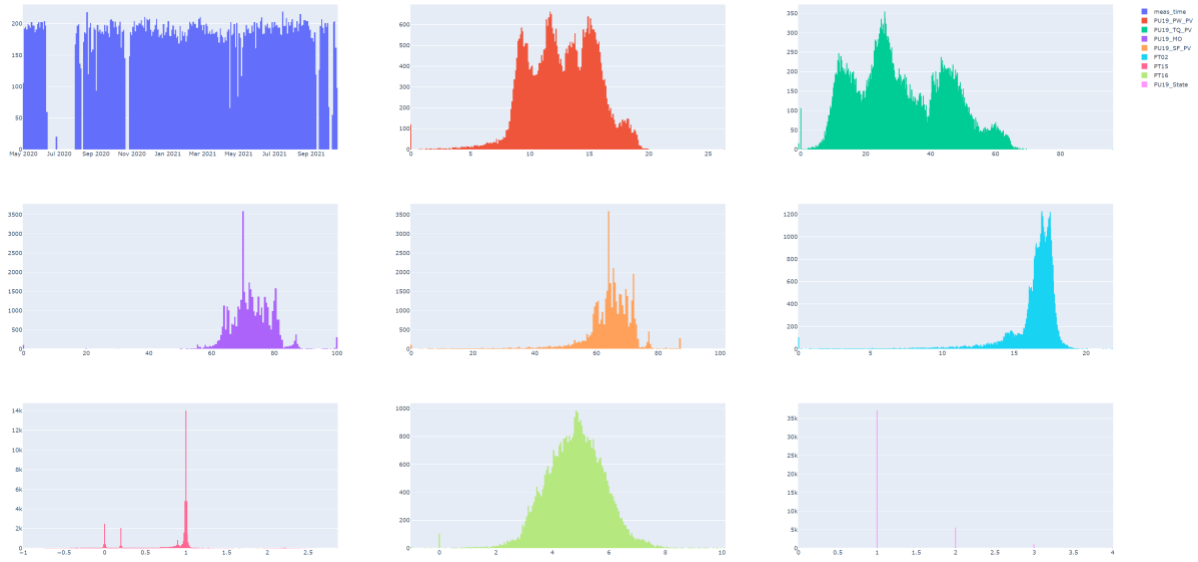


Figure 2-23 Histogram for 30 second data

A scatter matrix plot can be seen in Figure 2-24, comparing to what was seen in Figure 2-22 there isn't much change except for the confusing lines around zero which has been removed.

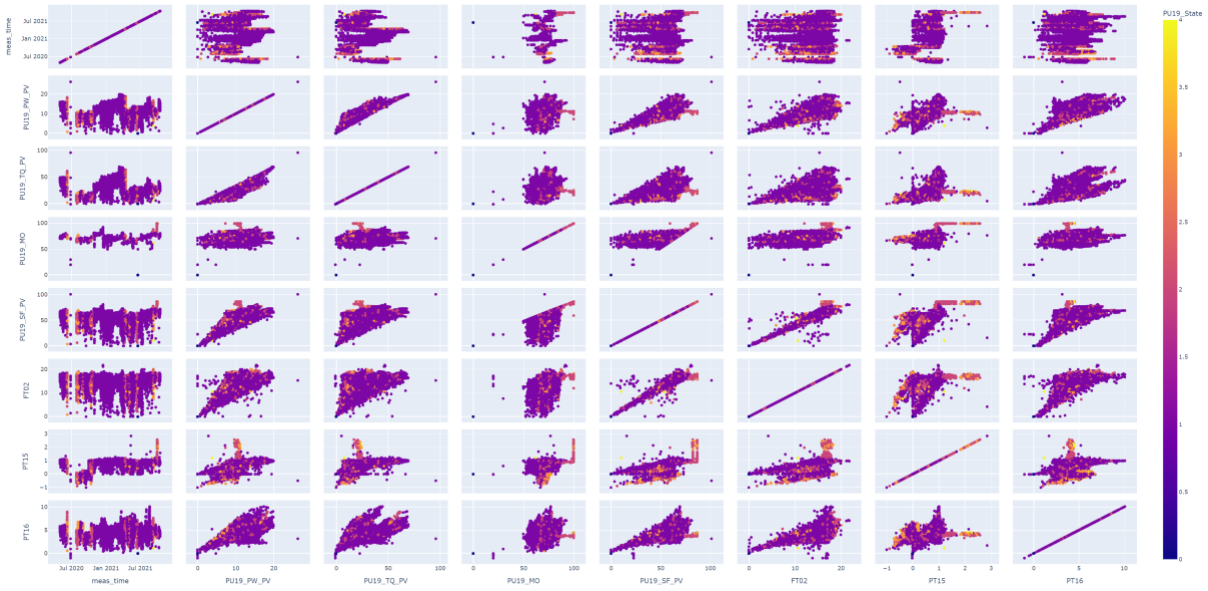


Figure 2-24 Scatter matrix plot

2.7.2 1 second data to 20 second

This dataset consists of pressure before and after the pump, flow before the pump, power, momentum, control signal, speed, the three vibration sensors which gives PeakVue values and vibrational velocity. The data used here is averaged per 20 second and it consists of 10 minutes of every hour, 21st May 2021 to 15th October 2021. Here IQR has been used to remove the top 0.005% of the ordered values of the vibration data, parts of the new dataset are described in *Table 2-12*. The maximum values have been decreased dramatically for all the vibration sensors comparing to *Table 2-5*.

Table 2-12 Descriptive statistics of 1 second dataset

PU19_V_L	PU19_V_I	PU19_V_O	PU19_P_L	PU19_P_I	PU19_P_O
434639.000	434639.000	434639.000	434639.000	434639.000	434639.000
1.025	1.013	0.970	0.490	5.822	0.533
0.890	0.858	0.761	0.359	4.484	0.429
0.000	0.000	0.000	0.000	0.000	0.000
0.016	0.016	0.025	0.020	0.213	0.018
0.907	0.986	1.001	0.538	6.109	0.573
1.470	1.351	1.309	0.682	7.729	0.694
5.137	4.134	3.495	2.736	24.782	2.367

As in the 30 second data, correlations have increased significantly in *Table 2-13* between the pump measurements and outlet pressure. The vibration measurements also have increased in correlation within the same type of measurements (differing between velocity (V_X) and PeakVue (P_X).

Table 2-13 Correlation matrix for 1 second dataset

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16	PU19_V_L	PU19_V_I	PU19_V_O	PU19_P_L	PU19_P_I	PU19_P_O	PU19
PU19_PW_PV	1.000	0.953	0.787	0.930	0.341	0.572	0.800	0.368	0.409	0.432	0.471	0.464	0.484	0.885
PU19_TQ_PV	0.953	1.000	0.763	0.908	0.337	0.553	0.774	0.327	0.371	0.390	0.421	0.433	0.459	0.838
PU19_MO	0.787	0.763	1.000	0.815	0.165	0.621	0.678	0.155	0.214	0.224	0.334	0.286	0.274	0.731
PU19_SF_PV	0.930	0.908	0.815	1.000	0.387	0.564	0.758	0.307	0.354	0.377	0.415	0.380	0.406	0.878
FT02	0.341	0.337	0.165	0.387	1.000	0.322	0.326	0.051	0.089	0.096	0.187	0.146	0.095	0.245
PT15	0.572	0.553	0.621	0.564	0.322	1.000	0.553	-0.175	-0.178	-0.131	0.257	0.289	0.284	0.560
PT16	0.800	0.774	0.678	0.758	0.326	0.553	1.000	0.418	0.397	0.431	0.447	0.447	0.395	0.761
PU19_V_L	0.368	0.327	0.155	0.307	0.051	-0.175	0.418	1.000	0.918	0.930	0.550	0.539	0.525	0.367
PU19_V_I	0.409	0.371	0.214	0.354	0.089	-0.178	0.397	0.918	1.000	0.982	0.627	0.594	0.593	0.375
PU19_V_O	0.432	0.390	0.224	0.377	0.096	-0.131	0.431	0.930	0.982	1.000	0.659	0.628	0.627	0.409
PU19_P_L	0.471	0.421	0.334	0.415	0.187	0.257	0.447	0.550	0.627	0.659	1.000	0.867	0.770	0.424
PU19_P_I	0.464	0.433	0.286	0.380	0.146	0.289	0.447	0.539	0.594	0.628	0.867	1.000	0.903	0.392
PU19_P_O	0.484	0.459	0.274	0.406	0.095	0.284	0.395	0.525	0.593	0.627	0.770	0.903	1.000	0.407
PU19	0.885	0.838	0.731	0.878	0.245	0.560	0.761	0.367	0.375	0.409	0.424	0.392	0.407	1.000

The histograms for each measurement can be seen in Figure 2-25. Here the zero values are very dominant in terms of number of samples. It is also noticeable that the vibration measurements have dropped dramatically.

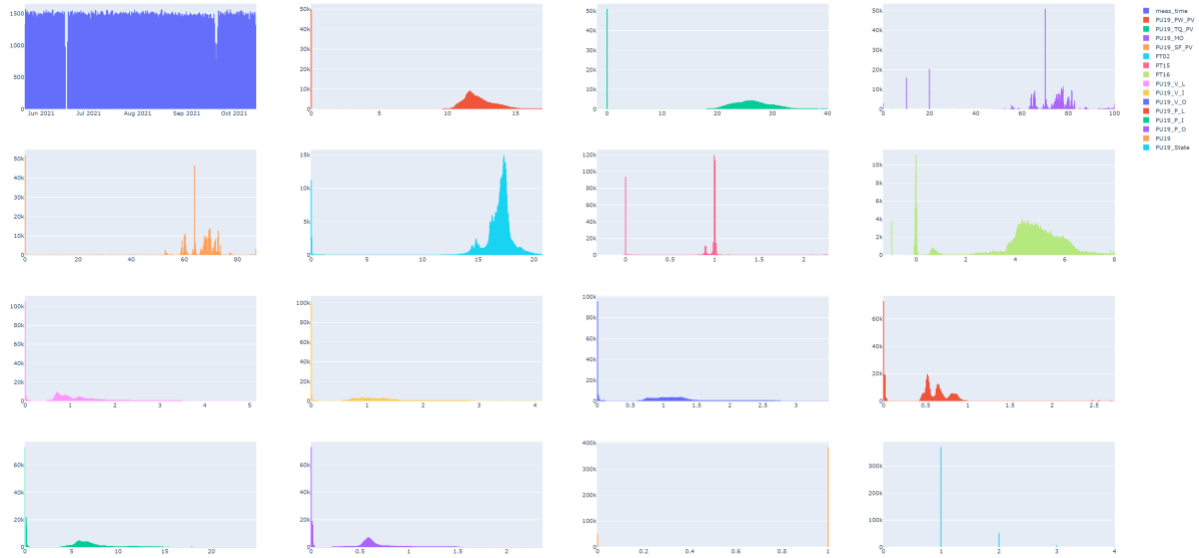


Figure 2-25 Histograms for 1 second dataset where samples where the pump is off has not been removed

Now that the data has been averaged and the highest values (0.005%) of the vibrational data are removed, it is much clearer where the endings of a lifetime is for the vibrational data as can be seen in Figure 2-26. However, this does not give a good indication of the end for a given pump, as the purple (normal operation) points seems to also be in these areas. It should also be noted that after September 9th the vibration data is constantly very close to zero. This will likely have a great impact on the PCA analysis. Figure 2-26 can be seen in increased size in Appendix M.

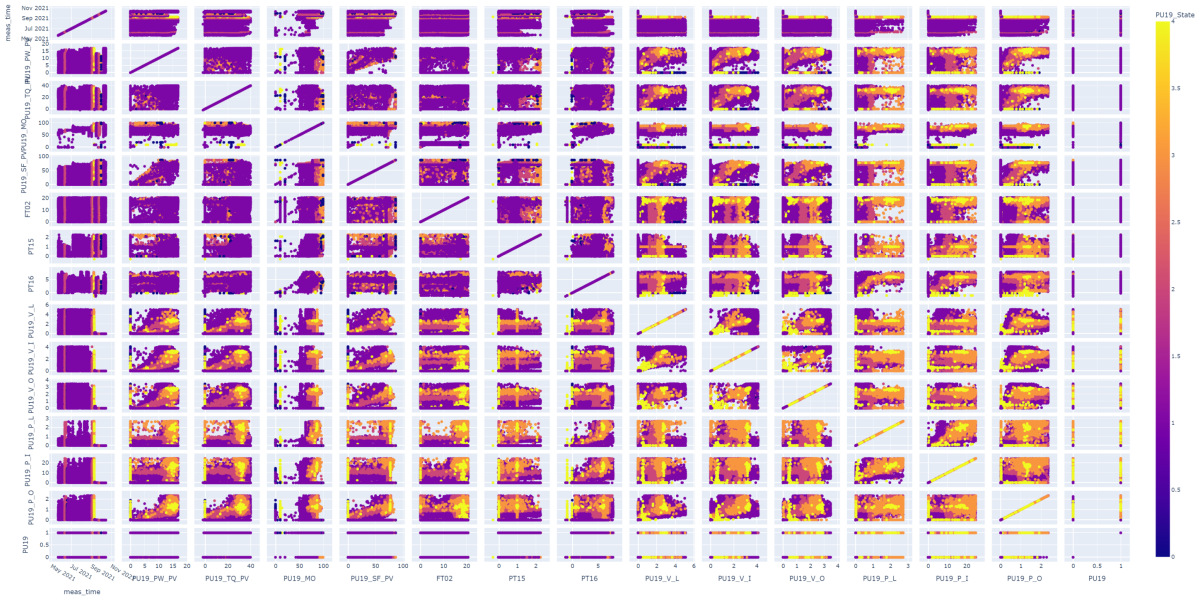


Figure 2-26 Scatter matrix plot for 1s data averaged before removing samples related to the pump being off

As in Figure 2-23, many of the zero values has been removed from the dataset. Other than this, only the visibility of the curves is more concrete in Figure 2-27.
There is not much change in Figure 2-28 from Figure 2-26 other than some of the zeros has been removed. Figure 2-28 can be seen in increased size in Appendix N.

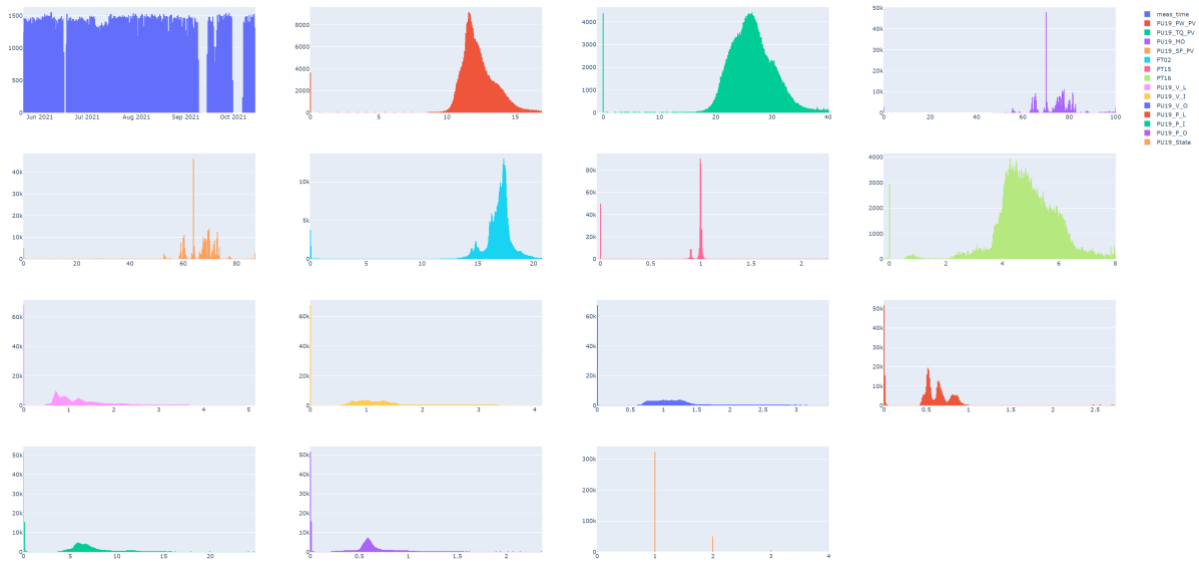


Figure 2-27 Histogram for averaged 1 second dataset where the pump on/off signal has been used to remove samples where the pump is off

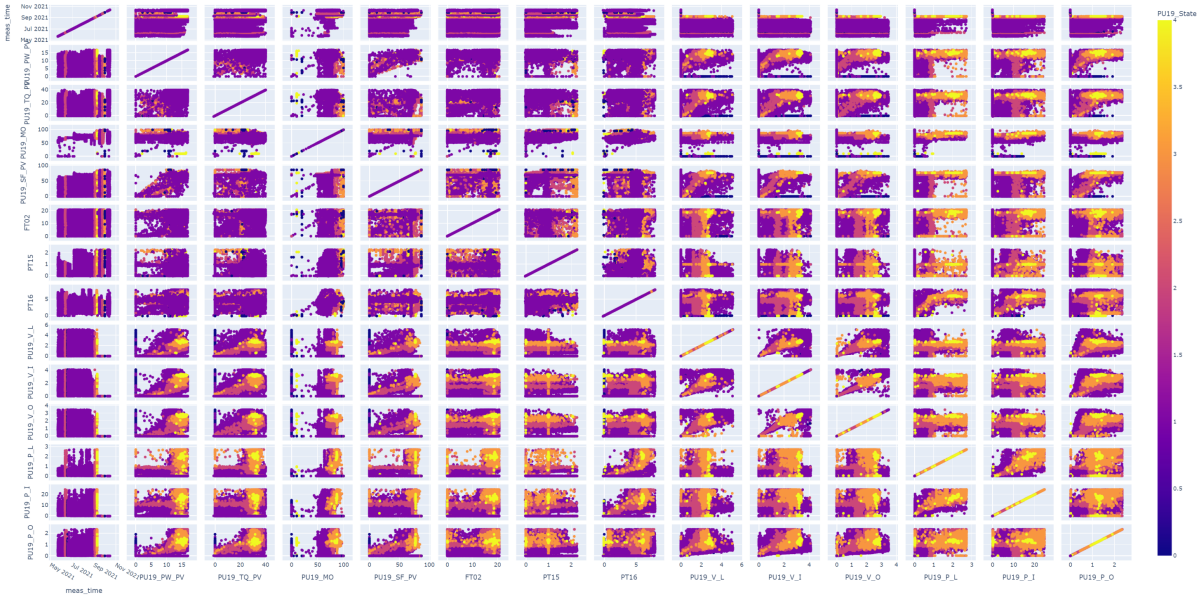


Figure 2-28 Scatter matrix plot of averaged 1 second data where sample matched with pump off signals are removed

2.7.3 50ms to 1 second

This dataset consists of 5 minutes of every hour from July 1st to October 15th and the variables are pressure before and after the pump, flow before the pump, ampere, momentum, control signal and speed of the pump. The data is averaged over 1 second such that each datapoint consists of 20 points.

Some simple statistics of this dataset can be seen in *Table 2-14*, as with the previous cases, the means, standard deviation and maximum values has decreased.

Table 2-14 Descriptive statistics of the dataset before removing samples where pump is off

	pu19_pw_pv	pu19_tq_pv	pu19_sf_pv	pu19_mo	pt15	pt16	ft02
count	519396	519396	519396	519396	519396	519396	519396
mean	10.875	23.657	59.789	70.116	0.844	4.075	16.710
std	4.533	10.225	24.672	20.810	0.437	1.837	3.154
min	0.000	-1.158	0.000	10.000	-0.104	-1.146	0.000
0.250	11.271	23.102	66.385	73.717	0.902	3.952	16.880
0.500	12.079	26.489	68.990	76.656	0.994	4.499	17.283
0.750	13.354	29.634	71.209	79.523	1.006	5.085	17.548
max	24.893	88.925	86.995	100.000	8.020	12.120	38.160

Correlations here as well has increased significantly as seen in Table 2-15, interestingly the inlet pressure correlations have increased very much towards the pump measurements as well.

Table 2-15 Correlation matrix

	pu19_pw_pv	pu19_tq_pv	pu19_sf_pv	pu19_mo	pt15	pt16	ft02
pu19_pw_pv	1.000	0.966	0.929	0.831	0.711	0.881	0.284
pu19_tq_pv	0.966	1.000	0.911	0.802	0.684	0.871	0.290
pu19_sf_pv	0.929	0.911	1.000	0.906	0.818	0.876	0.349
pu19_mo	0.831	0.802	0.906	1.000	0.769	0.804	0.041
pt15	0.711	0.684	0.818	0.769	1.000	0.678	0.280
pt16	0.881	0.871	0.876	0.804	0.678	1.000	0.266
ft02	0.284	0.290	0.349	0.041	0.280	0.266	1.000

The shapes of the data for variables such as ampere (PW_PV) and torque (TQ_PV) have changed significantly. This may indicate that the 1 minute of every hour used earlier was in fact not representative, yet this dataset may be misrepresenting the full dataset. From the one second dataset in Figure 2-25 it looks rather similar in ampere and torque.

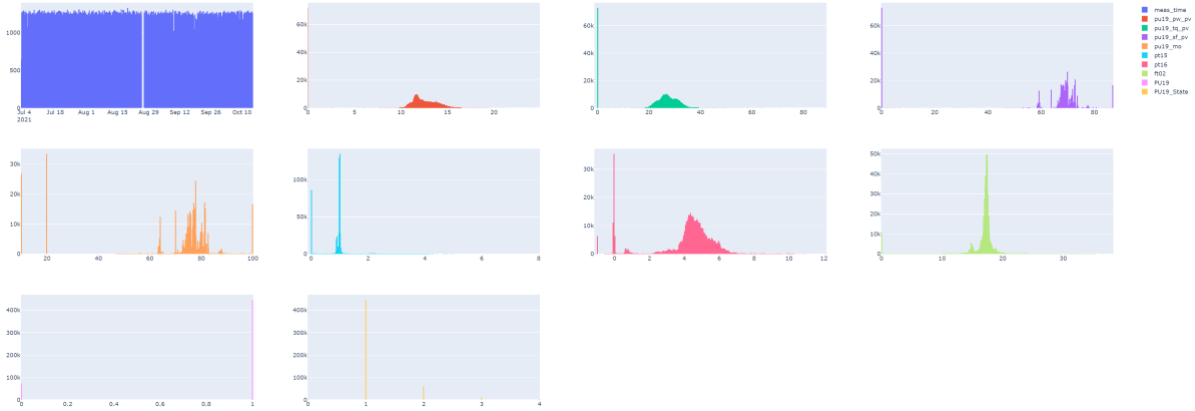


Figure 2-29 Histograms for averaged 50ms data before removal of samples where the pump is off

From the columns for control signal and speed in Figure 2-30, it once again appears that these increase at the end of a lifetime.

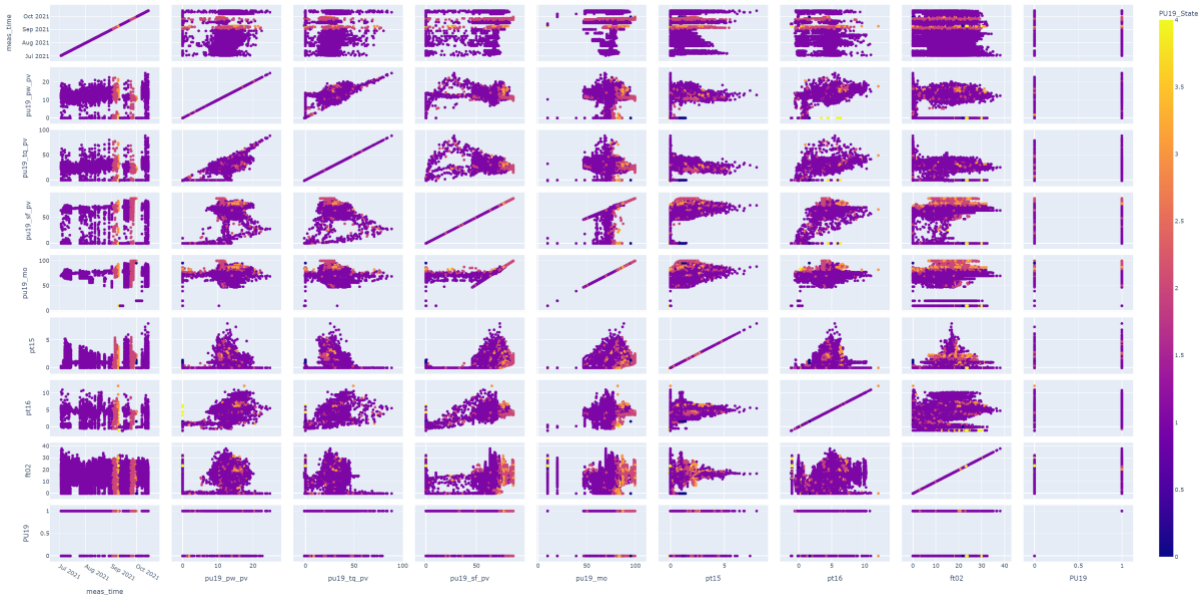


Figure 2-30 Scatter matrix plot of averaged 50ms data before removing samples where pump is off

As seen in the previous histograms where samples which is associated with the pump being off has been removed, zero values have been greatly reduced and the shapes of the data is more visible in Figure 2-31.Figure 2-31. The effect is also visible in control signal (MO) where there were many measurements on 20% which has been removed due to the pump being off.

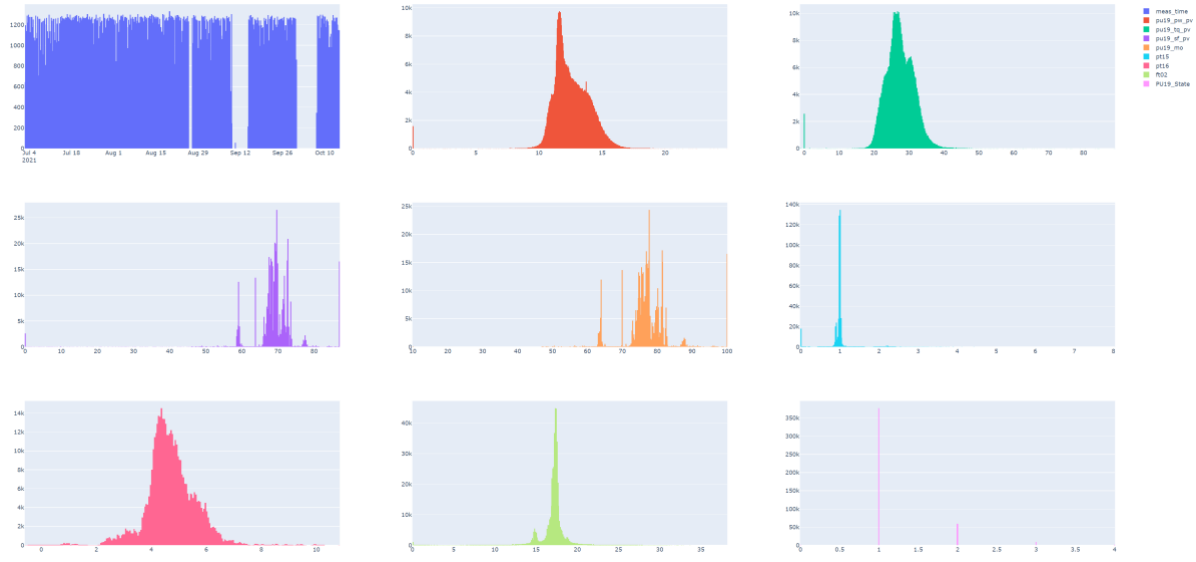


Figure 2-31 Histograms for averaged 50ms data where samples that has pump off associated with them removed

Figure 2-32 shows scatter plots between each of the variables after processing. The speed (SF_PV) and control signal (MO) seems to give the largest indication to the end of a lifetime where these increase towards the end when comparing to flow, inlet and outlet pressure, torque and power usage.

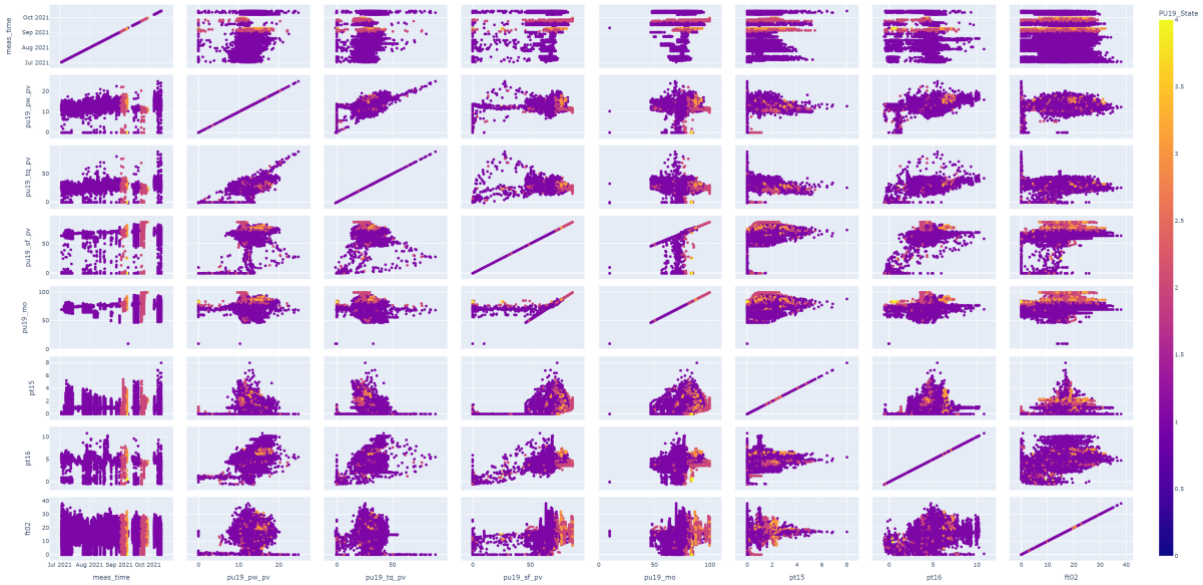


Figure 2-32 Scatter matrix plot for averaged 50ms data

3 Result

This chapter will focus on achieving the goals set earlier; finding the most important variables and creating a machine learning model for indicate the state of the pump. This will be done using PCA, STFT, Naïve Bayes, SVM and LSTM.

3.1 Analysis

This subchapter will use PCA and STFT to investigate important variables for the degradation of the pump.

3.1.1 PCA

In this subchapter, four PCA analyses will be investigated, one for each data set and an additional for the 30 second dataset where samples which is associated with the pump being off is included. The data is used as described in the previous subchapter. That is; average of 20 samples, and outliers from vibration data which accounts for 0.005% of the top measurements. The same amounts of data per hour will be used as previously described, however for convenience it is repeated for each sub chapter.

30s interval

This dataset consists of pressure before and after the pump, flow before the pump, power, momentum, control signal in percent, speed and on/off signal for the pump. The data used here is averaged per 10minute and it is all the data from May 2020 to 15th October 2021.
At first, the Boolean on/off signal for the pump will be included to see if it is useful for the analysis. The danger here lies is in that it may make the components focus overly much on this one Boolean signal and that signals where the backup pump is running will interfere with the analysis of pump 19 (PU19). Then the measurements which is taken when the pump is off are removed. Afterall, the interest lies not in when the pump is off, but rather when it's running.
The loading plot in Figure 3-1 show that PT15 seems to have the highest impact on PC2. Other than this, the plot says that most of the variables correlate with PC1 where the speed has the highest impact. Figure 3-2 shows that PC3 primarily show the level of flow in the system.

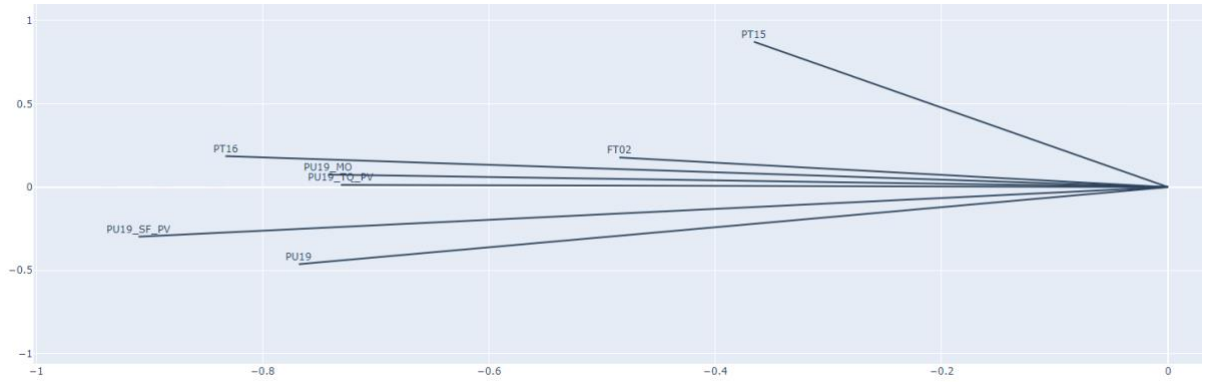


Figure 3-1 Loading plot PC1 along X-axis, PC2 along y axis

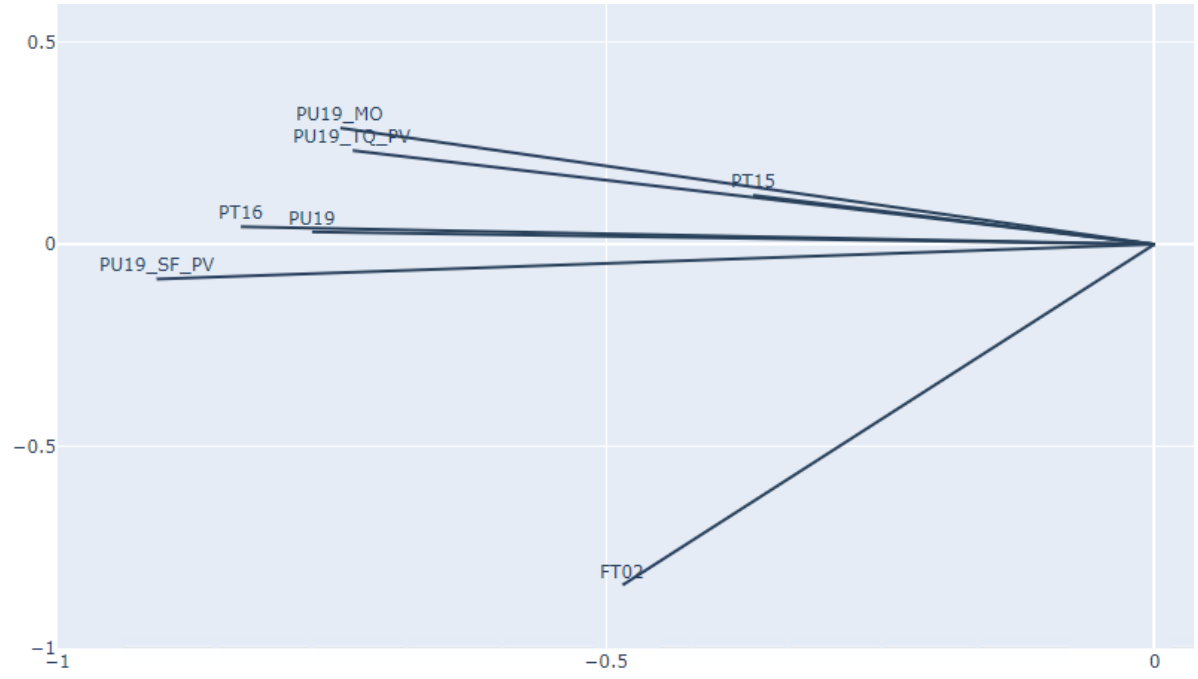


Figure 3-2 Loading plot PC1 along X-axis, PC3 along y axis

From the score plot in Figure 3-3 some horizontal clusters may be seen, however what separates them appear from Figure 3-1 to be the inlet pressure and the on/off signal. There is a small separate cluster with the colour of ready to be replaced, this cluster is not distinct however, the light red, orange and yellow samples are all over the map.

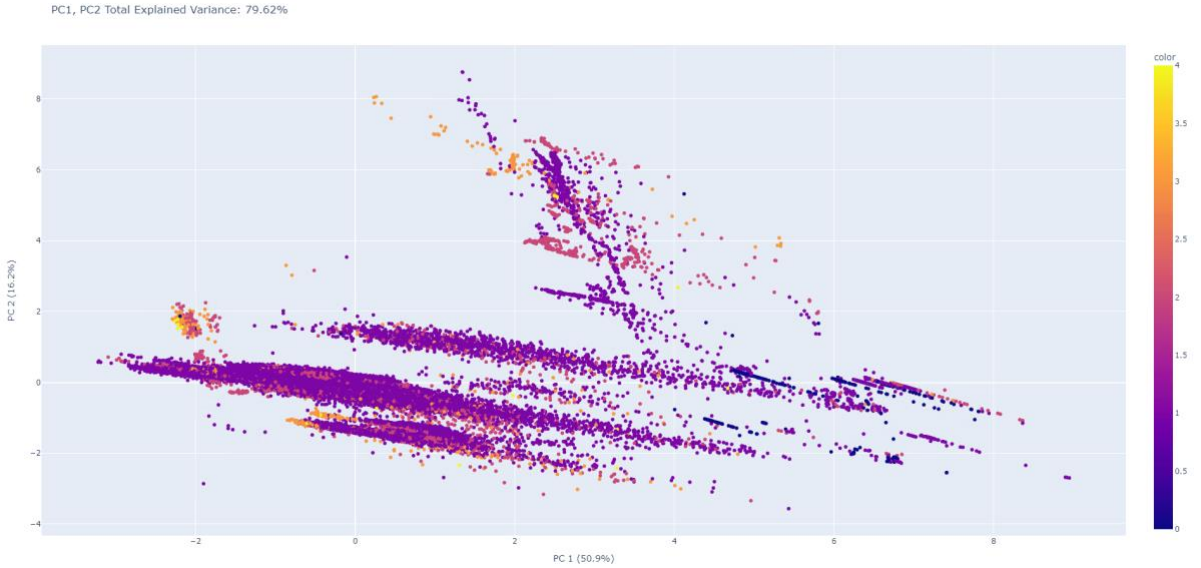


Figure 3-3 Score plot PC1 PC2

In Figure 3-4 the bad state of the pump seems to be distributed among the good states, there are some small clusters, the ones furthest to the lower left may hold important information, however from Figure 3-2 this indicates the change in flow.

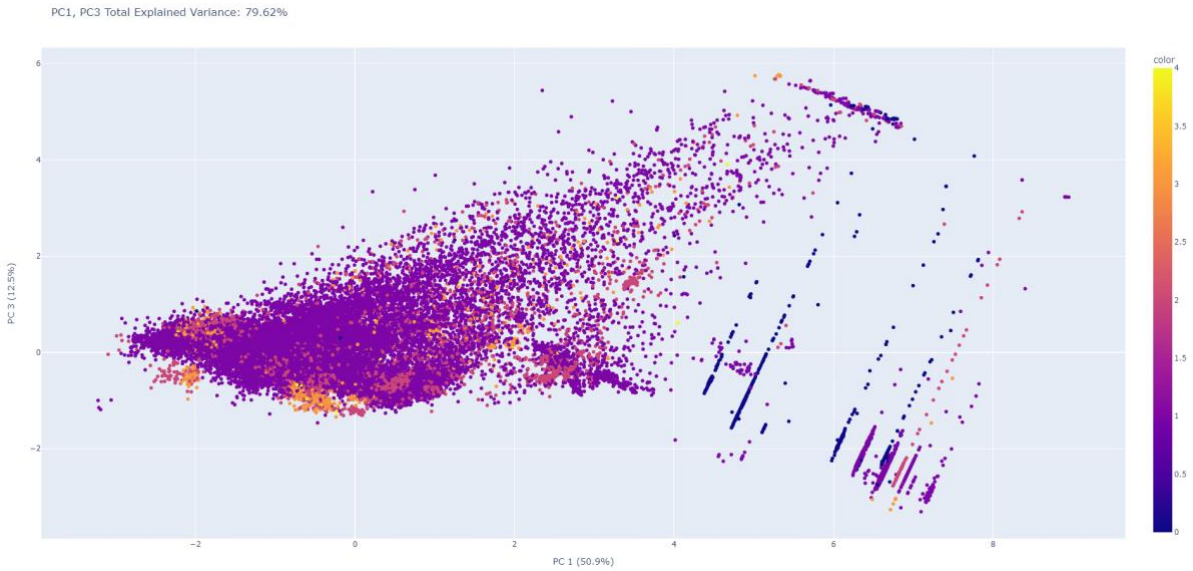


Figure 3-4 Score plot PC1 PC3

From these preceding plots, there is not much to distinguish in the sense of the failing pumps, removing the data from when the pump is inactive may refine the variations of the data as shown in chapter 2.7.1.

Figure 3-5 shows the loading plot for PC1 and PC2 where still all the variables keep to one side of PC1, this may be due to the averaging of the data, as seen in the difference between Table *2-4* and Table *2-10*. There is a slight variation in PC3 compared to PC2 where primarily PT16 have switched places with MO.

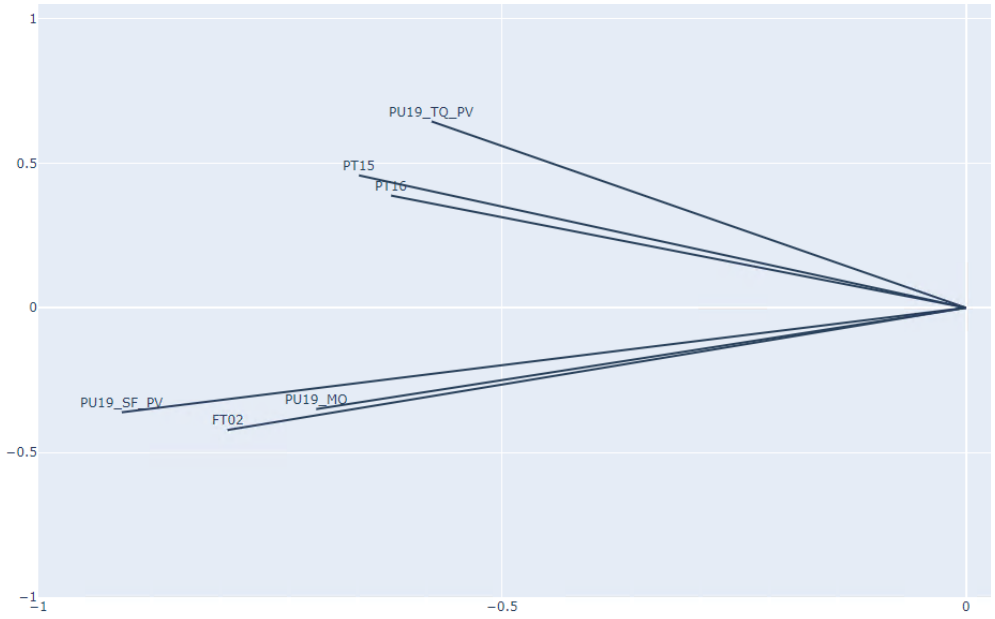


Figure 3-5 Loading plot, PC1 and PC2, after removing samples where pump is off

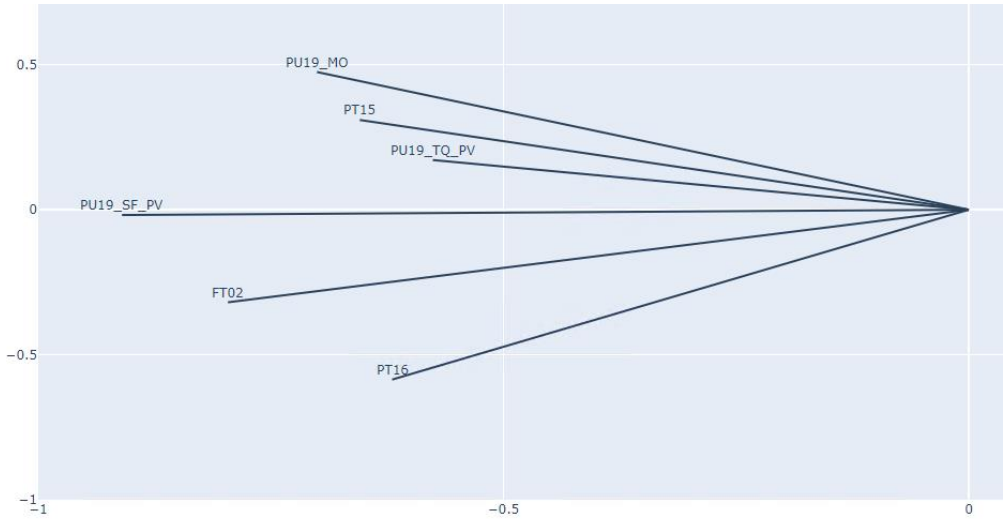


Figure 3-6 Loading plot, PC1 and PC3, after removing samples where pump is off

In Figure 3-7 some clusters of the orange and light red can be seen on the outskirts of the massive cluster around the origin. However, there are larger clusters of light red within the purple cluster, thus there aren't any clear indication of the difference between the pump state. PC1 and PC3 in Figure 3-8 show very much the same case where most of the samples are meshed. This may be caused by outliers or the fact that the pump is replaced with a new one with small differences.

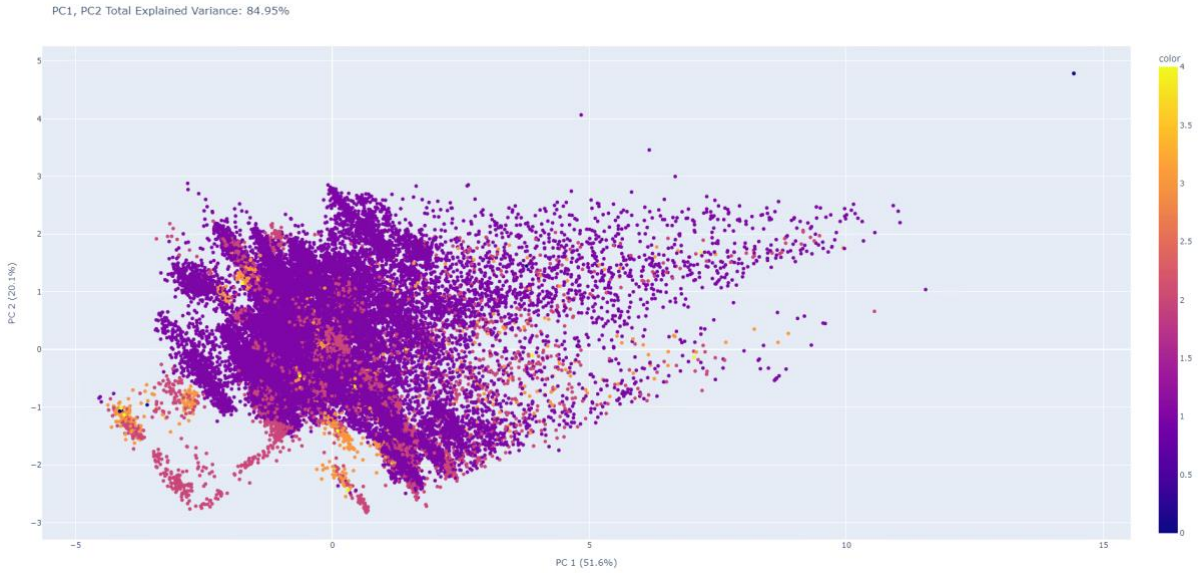


Figure 3-7 Score plot, PC1 and PC2, after removing samples where pump is off

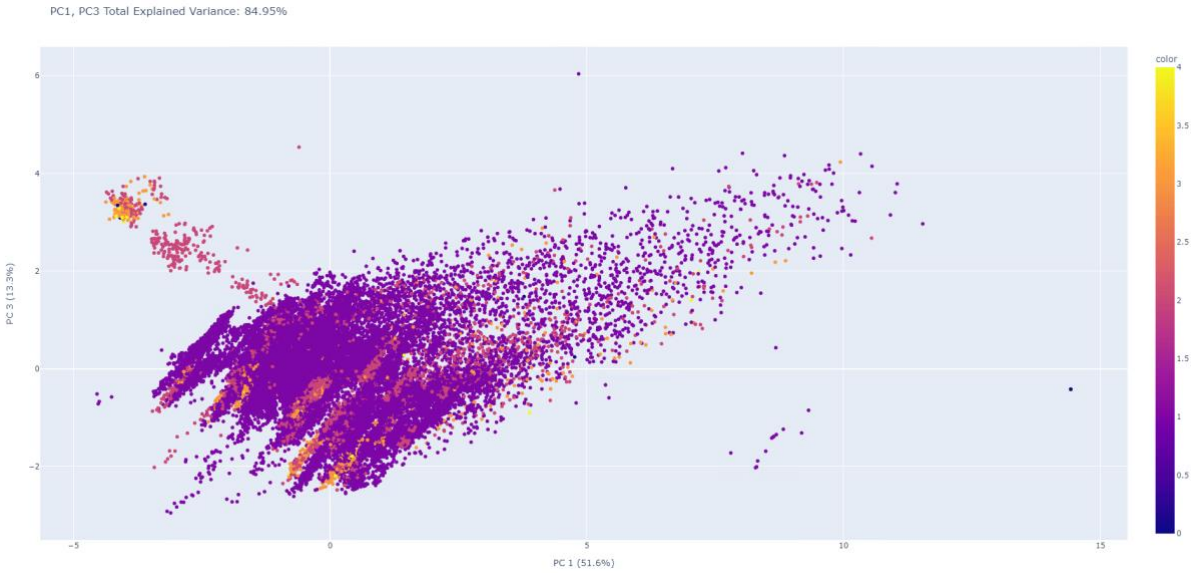


Figure 3-8 Score plot, PC1 and PC3, after removing samples where pump is off

1s interval

Here the data from chapter 2.7.2 is used, the samples where the pump is off has been removed. Comparing to the previous PCA analyses, here the loading plot has some orthogonality between the variables in Figure 3-9. The vibrational measurements, both in velocity and PeakVue dominate PC1. While pump measurements (PU19_XX) and flow are the main influence on the PC2 axis. The pressure measurements do not have a high influence on neither of PC1 and PC2.

The PC1, PC3 loading plot in Figure 3-10 indicate that the vibrational velocity measurements are negatively correlated to the inlet pressure while the vibrational PeakVue values are more or less uncorrelated to the inlet pressure. This is interesting as there has been a period of low inlet pressure, and from this, the low inlet pressure should not impact the PeakVue values considerably, however it should affect the vibrational velocity.

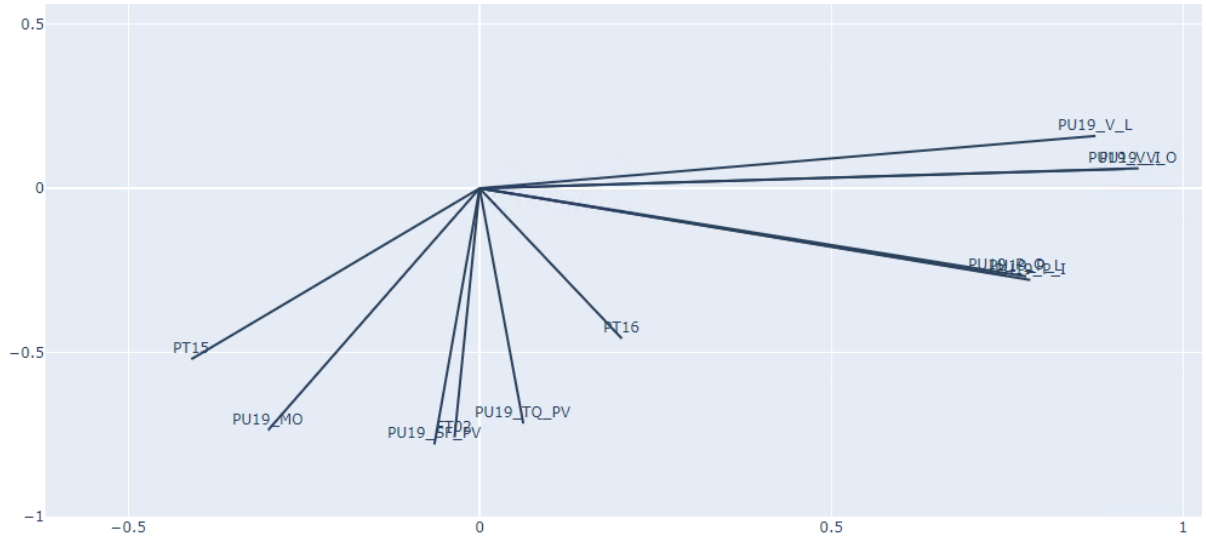


Figure 3-9 Loading plot for vibration data PC1 and PC2

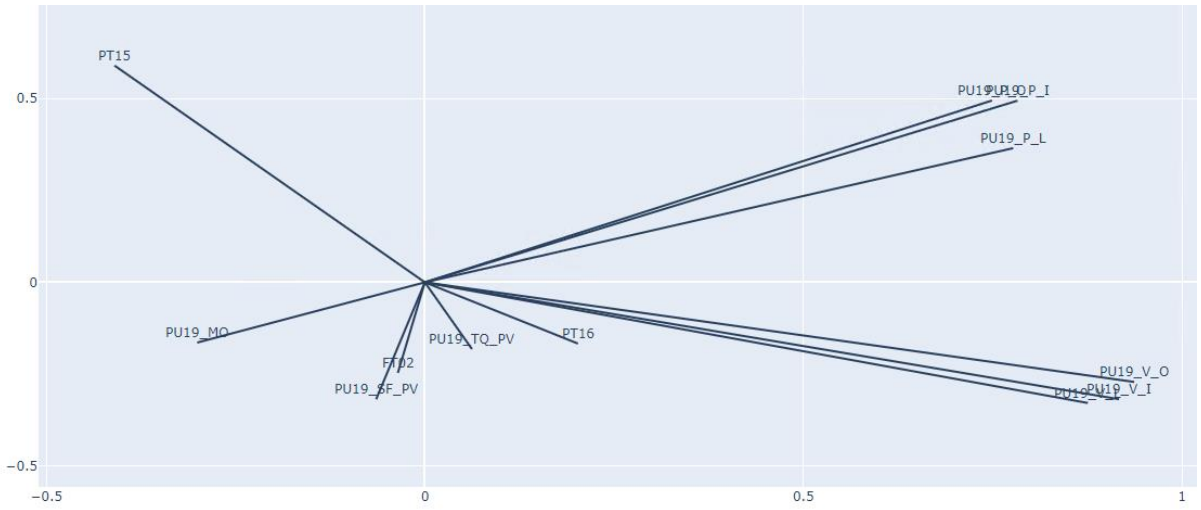


Figure 3-10 Loading plot for vibration data PC1 PC3

The score plot in Figure 3-11 show that 75% of the variance has been explained where PC1 is 38% and PC2 is 24.8%. The dataset contains three failures, which can to some degree be seen in the coloured cluster of orange and yellow. One of these clusters appear to be very different from the others with much smaller variations, this, as mentioned in chapter 2.7.2 is likely the cause of the low vibration measurement values from 9th of September. It has been found that the pump in the timespan 9th to 30th September is of another brand than Netzsch which has primarily been used in this tag position. This may reduce the nuances of the plots for the Netzsch pump. On the right end of the score plot there appears to be a grouping with the later stages of a pump and yet another little grouping around 2 on PC1 and just above the large purple grouping. This indicates that there are differences in the measurements from fault to fault, it is however expected as one fault would be different from another, yet they seem to share more or less the same base (purple dots in the large cluster).
PC3 covers 12.5% of the variance, the same anomaly can be seen in Figure 3-12, where, on the left side a cluster appears which is assumed to be due to the low vibration measurements. The other cluster appears to be rather continuous in the sense of colouring, most of the orange dots are a long trail, however it is still mixed into the light red and purple clusters, which make it hard to distinguish them from each other.

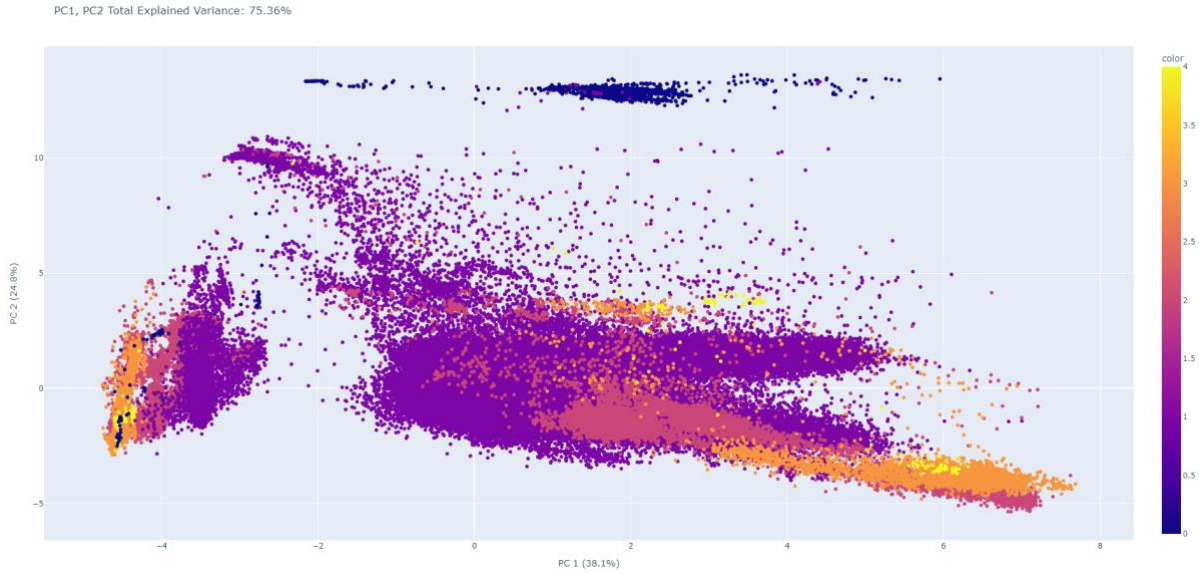


Figure 3-11 Score plot for PC1 and PC2

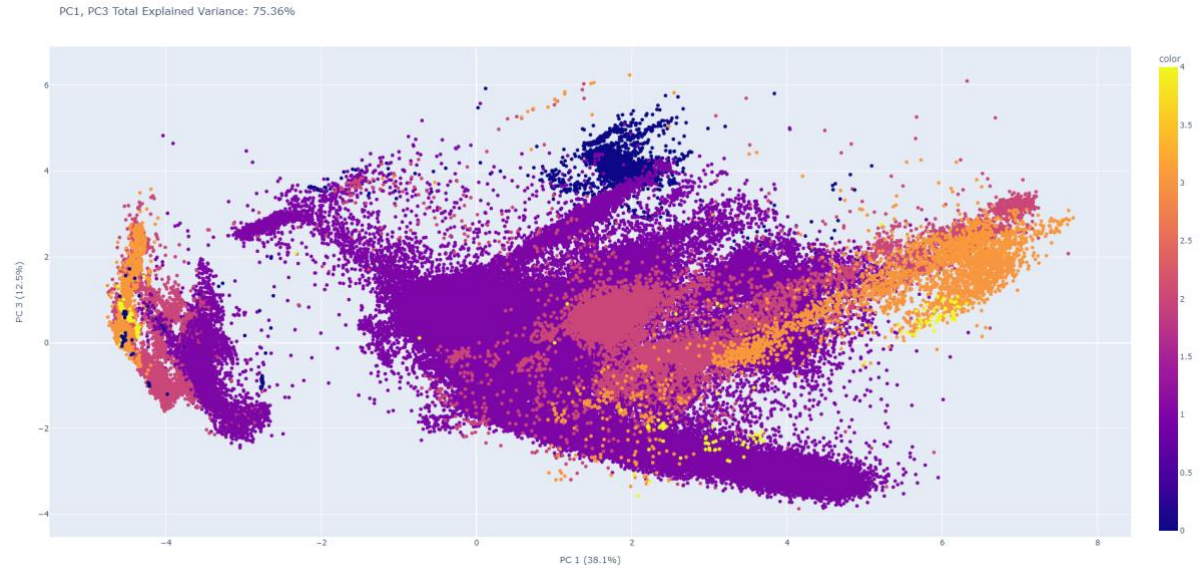


Figure 3-12 Score plot for PC1 and PC3

50ms interval

Here the 50ms data explained in chapter 2.7.3 is used where the measurements where the pump is off has been removed. The loading plot in Figure 3-13 shows PC1 along the x-axis and PC2 along the y-axis, all of the variables are on the left-hand side of PC1 where the speed (PU19_SF_PV) is the most important variable. While for PC2 the torque (TQ_PV), ampere (PW_PV), control signal (MO) and inlet pressure (PT15) are the most important variables. Control signal and inlet pressure are close to orthogonal with the torque and ampere signals and are thus close to uncorrelated along PC2.

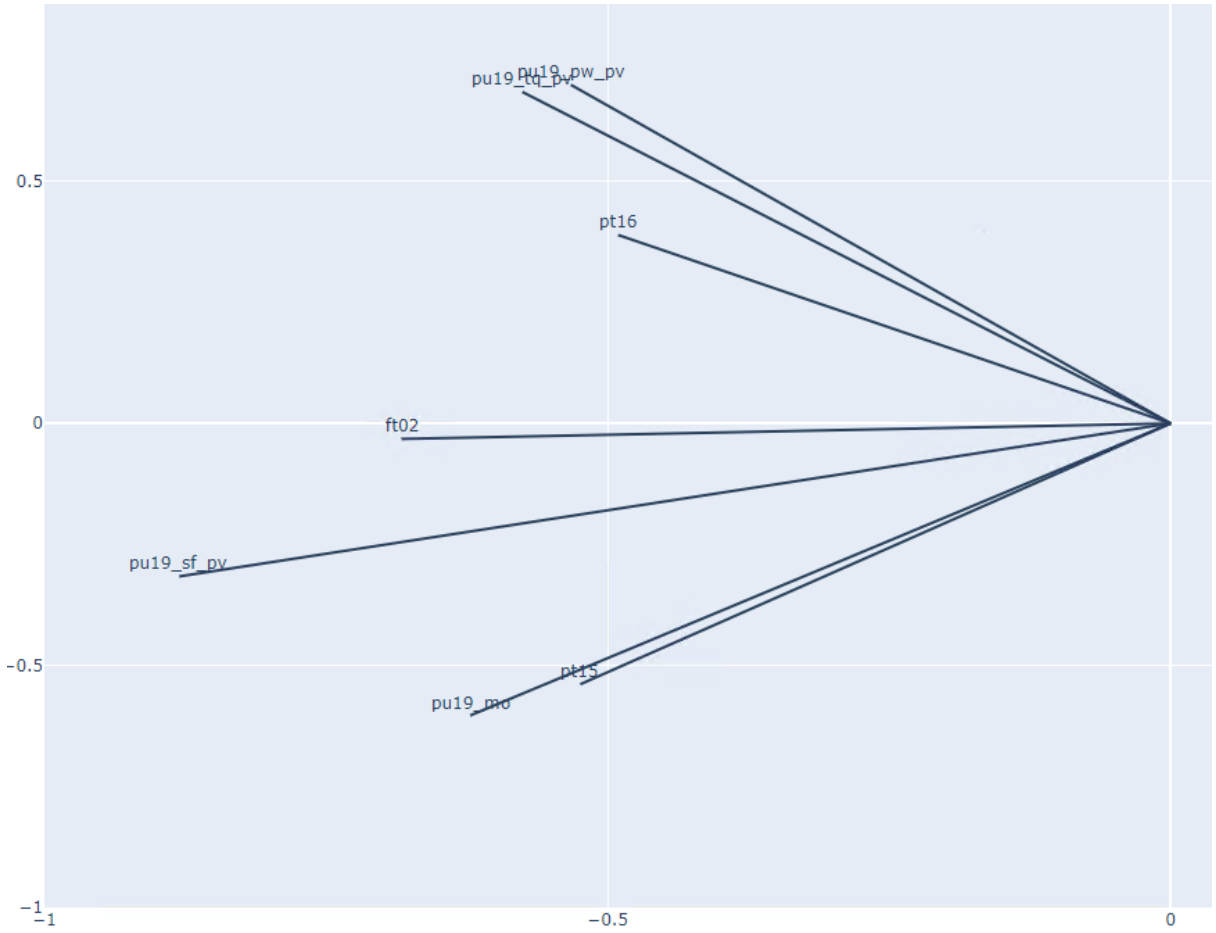


Figure 3-13 Loading plot for PC1 and PC2

In Figure 3-14 PC1 is shown along the x-axis and PC3 along the y-axis. As earlier seen, the speed is the most important variable of PC1 and all variables are on the left side of PC1. In PC3 the outlet pressure (PT16) and the inlet flow (FT02) are the most important variables.
The score plot in Figure 3-15 shows the distribution of samples along PC1 and PC2 as well as the total explained variance, 77.3%. This is somewhat low to estimate the dataset on.
The later stages of the pump seem to be separated to some degree in the lower yellow, orange and light red clusters on the left-hand side. As the states are somewhat arbitrarily set in time it isn't expected that these will indicate perfectly well the wear of the pump. The tail on the right-hand side is likely points where the speed is low (and consequently most other measurements) as the speed is the dominating factor of PC1.

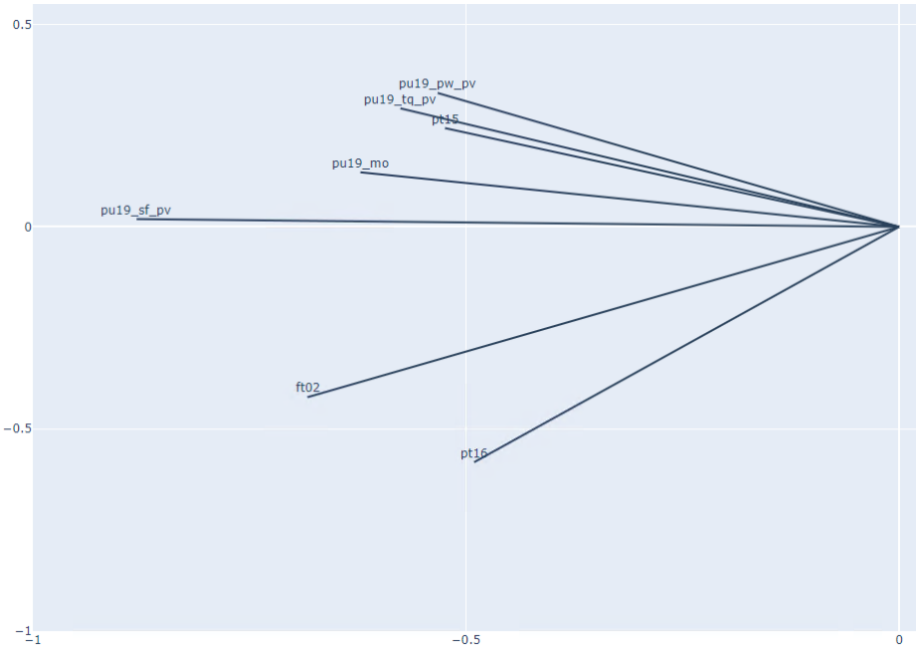


Figure 3-14 Loading plot for PC1 and PC3

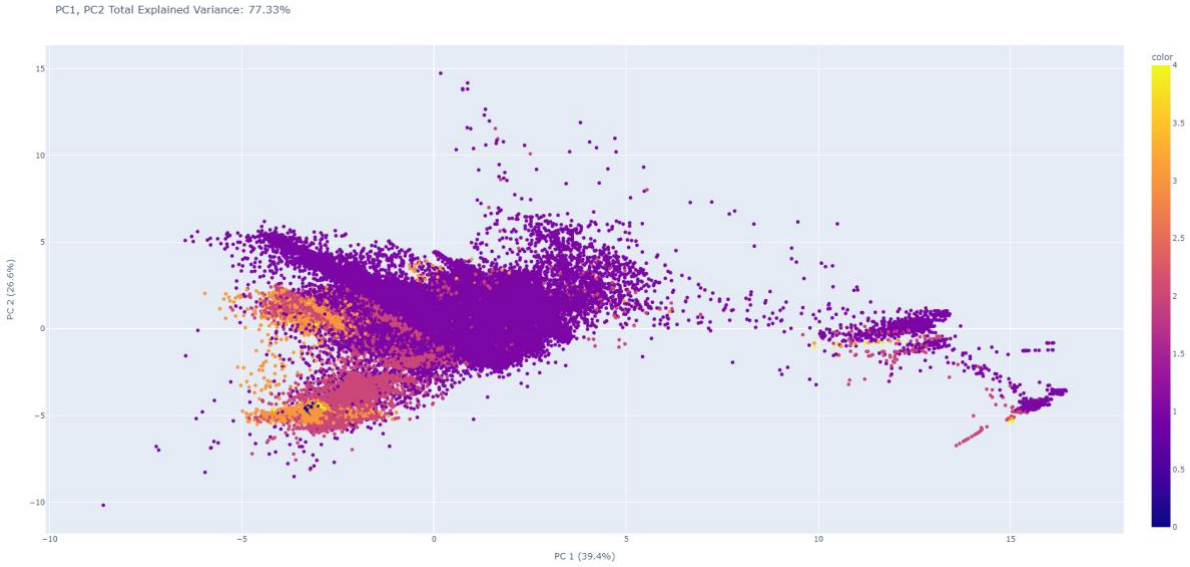


Figure 3-15 Score plot for PC1 and PC2

Figure 3-16 shows PC1 and PC3 of the analysis, the clusters of yellow, orange and light red seem only to be extensions of the purple cluster without much difference from the other tendrils of the purple cluster. Again, the tail on the right-hand side is the measurements of low values.

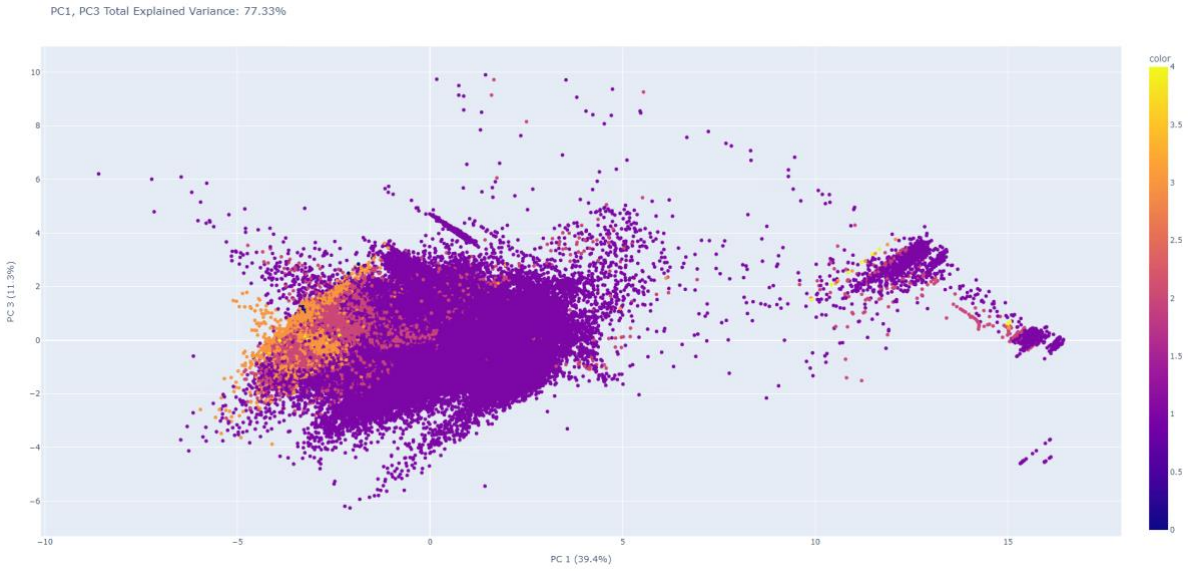


Figure 3-16 Score plot for PC1 and PC3

3.1.2 Fourier transform spectrogram

To compute and visualize the changing frequency components in the torque signal over time, a spectrogram with consecutive Fourier transforms has been calculated using the scikit-learn signal processing method spectrogram. The code for this can be seen in Appendix O. for the last 120 samples of the 30s data (1 hour) every 10 minutes. The Hanning/Hann window was used since it has good frequency resolution and reduce some of the effect of spectral leakage and is considered satisfactory in 95% of cases [40].
When plotting the results from the signal processing using a heatmap-plot the colours represent the resulting value for the frequency. The y-axis represents the frequencies and the x-axis is time. In Figure 3-17 below a heatmap plot for 5 days before a pump failure. It's not very easy to see a clear change right before the failure, but if we zoom in on the lower frequencies of the spectrogram as seen in Figure 3-18. We see that the distribution of frequencies is changing the day before pump failure.

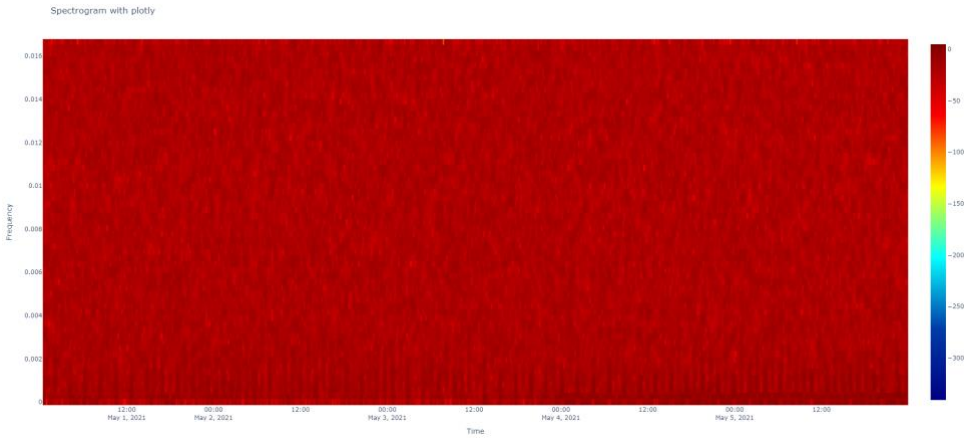


Figure 3-17: Consecutive Fourier transform visualized using heatmap plot

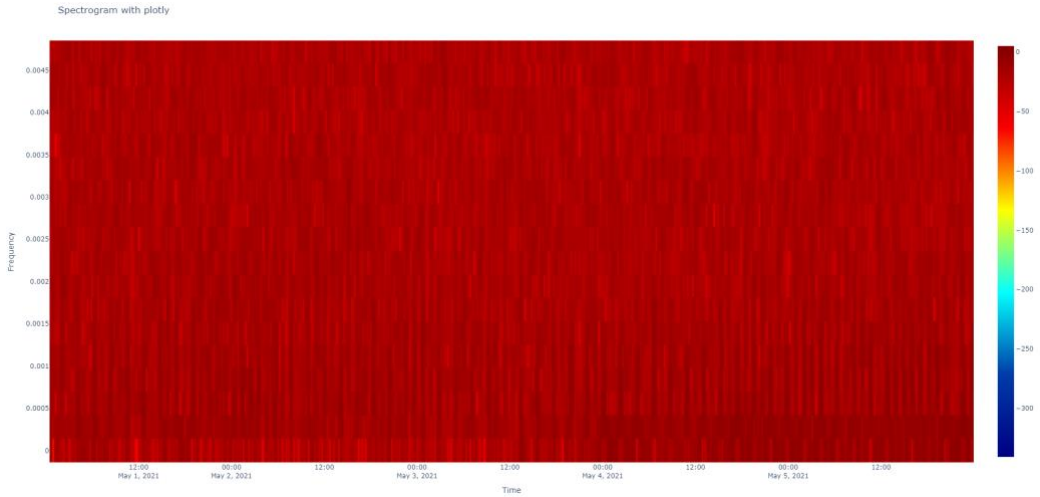


Figure 3-18: Zoomed in portion of spectrogram heatmap

When inspecting the Fourier transform spectrograms in different ways the changes can be more clearly seen. In Figure 3-19, Figure 3-20, Figure 3-21 and Figure 3-22 the spectrograms have been plotted as a heatmap focused on the lower frequencies as well as line plots, for two timeframes leading up to a pump failure at the right end of the plots. As can be seen, the frequencies in the torque measurements are clearly changing near the end of the pump's life. We can also see from the plots that the change is not the same, which can be due to differences in the pump's behaviour or possibly due to this being different failure-types.

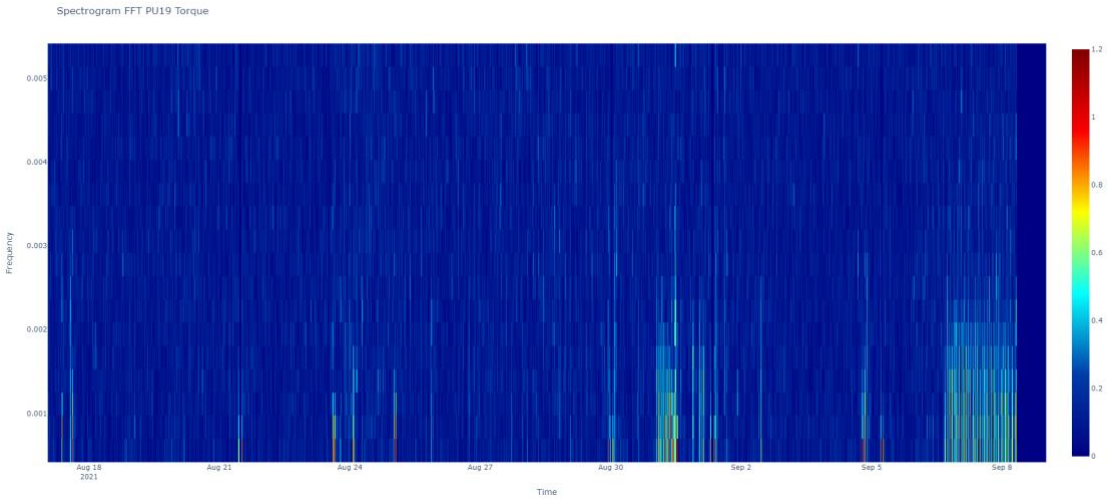


Figure 3-19: Spectrogram heatmap focused on lower frequencies, before failure 9th of September

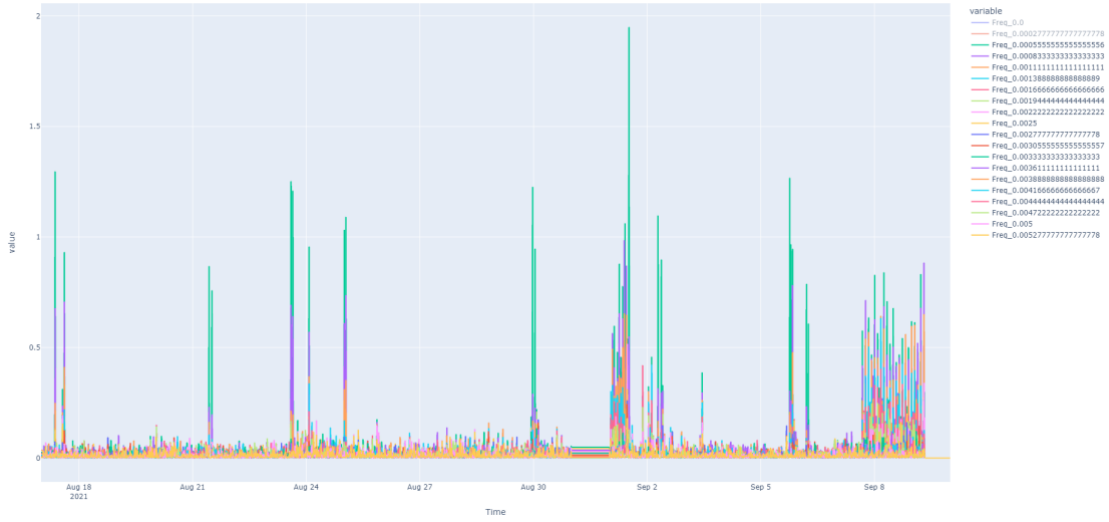


Figure 3-20: Spectrogram line plots of lower frequencies, before failure 9th of September

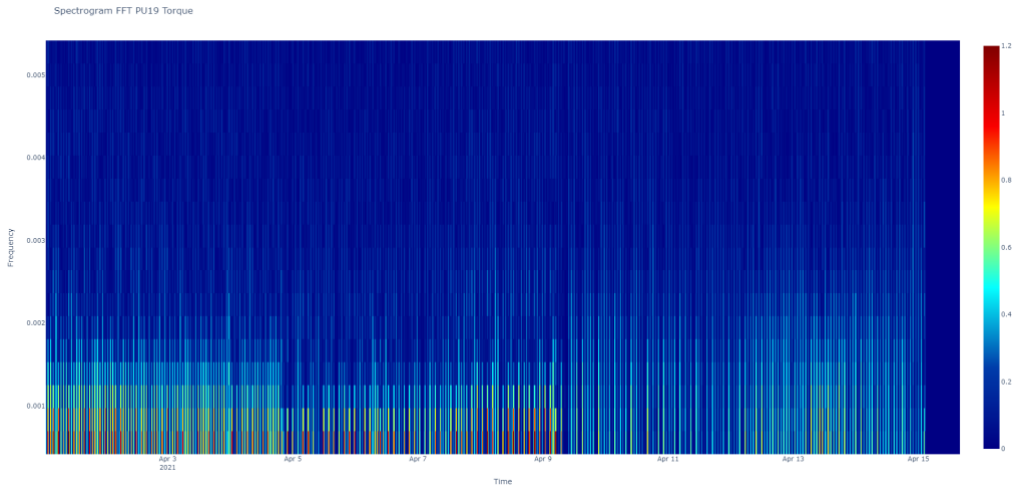


Figure 3-21: Spectrogram heatmap of lower frequencies, before failure 15th of April

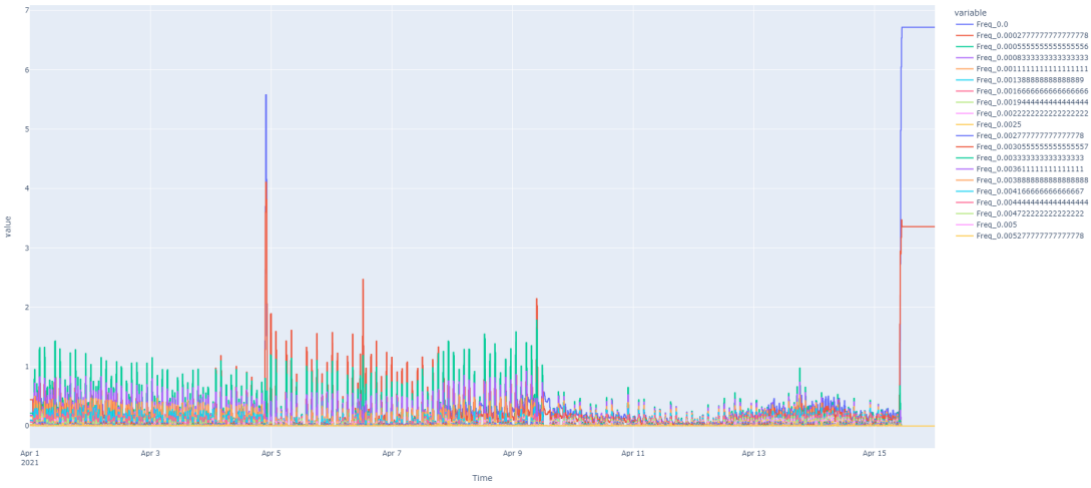


Figure 3-22: Spectrogram line plots of lower frequencies, before failure 15th of April

3.2 Machine learning training and tests

The machine learning methods that have been tested here are Naïve Bayes Classification, Support Vector Machine (SVM) classification and classification using the Long Short-Term Memory (LSTM) artificial recurrent neural network architecture. The code for the model training and testing is seen in Appendix P and Appendix Q.

3.2.1 Naïve Bayes

For these tests the Gaussian Naïve Bayes classifier in the scikit-learn python library was used.

30-second data with Fourier Transform values

In addition to the 30s data measurements the torque values were pre-processed calculating the Short Time Fourier Transform (STFT) for 1-hour windows of measurements (120 samples) adding the values as features for the training.
The values were scaled and centred using a Standard Scaling function provided by the Scikit learn API. The scaler was fitted to the data from the training set timeframe data.
The resulting model trained on 30s data from May 2020 to August 2021 achieved a precision of 70.6% on the test set from the same timeframe as the training set. It fails a lot at predicting near to failure states (labels 2-4). It also predicts a lot of false positive failures for the normal operating state (label 1) as can be seen in the confusion matrix plot in Figure 3-23.

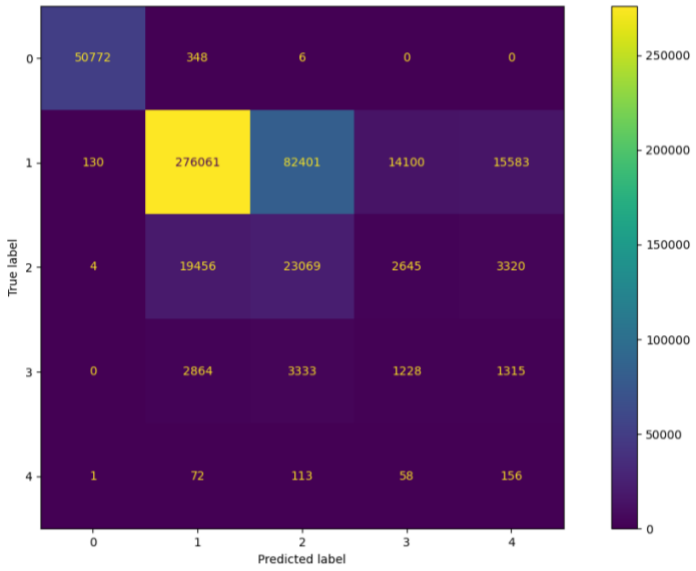


Figure 3-23: Confusion matrix of Naïve Bayes-model prediction results compared true label

When testing the model with data from September 2021, with data outside the training timeframe range the results of the model is worse. The model achieves an accuracy of 54.9%, The accuracy of predicting stopped and normal running is high (99.7% and 98.6%), but the model predicts most failure states as normal running with a near to 0.0% total accuracy of the failure modes (Label 2-4) as can be seen in Figure 3-24.

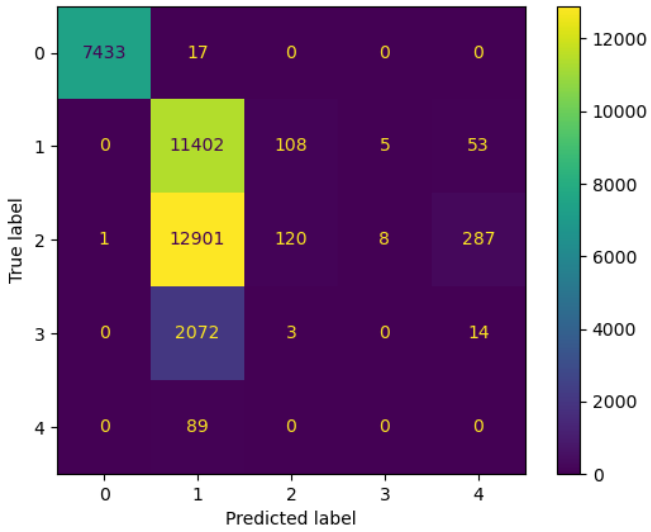


Figure 3-24: NB-prediction results performed on September 2021 30s data

50-millisecond data

The model tested here was on 50ms data averaged over 1s. The Short Time Fourier Transform interval was done using windows of 3 minutes (180 samples). The training data spans from beginning of July until 10th of September 2021. In this timespan there are few failures. The total accuracy of the trained model achieved a total accuracy of 33.1%, but as can be seen in the confusion plot in Figure 3-25 that it fails a lot on separating "stopped" and "normal running", and all failure states are predicted to be "less than 1 week from failing (Label 2) and no correct predictions of "less than 24 hours from failure" (Label 3) and "less than 1 hour from failure" (Label 4).

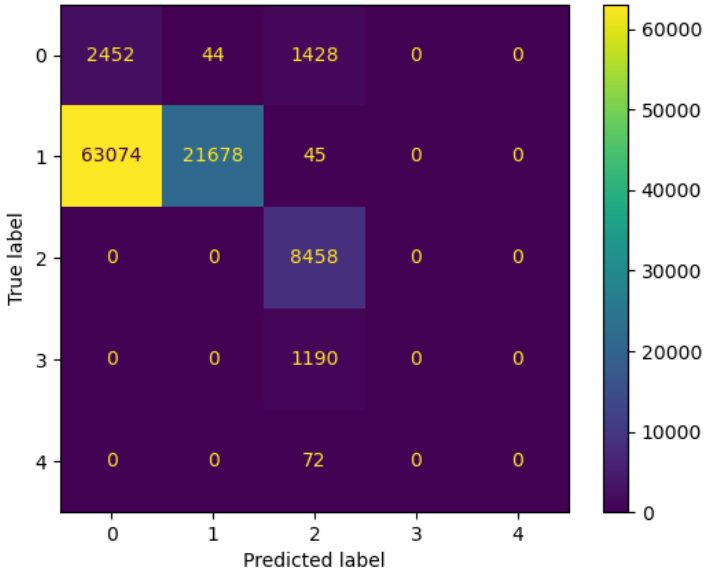


Figure 3-25: NB-prediction results from test set in training timeframe, July 2021 - September 2021

When tested with data from 10th of September – 15th of October the model predictions, seen in Figure 3-26, are all "less than 1 week from failure" which is mostly wrong giving a total accuracy of 16.6% all from the 2nd label.

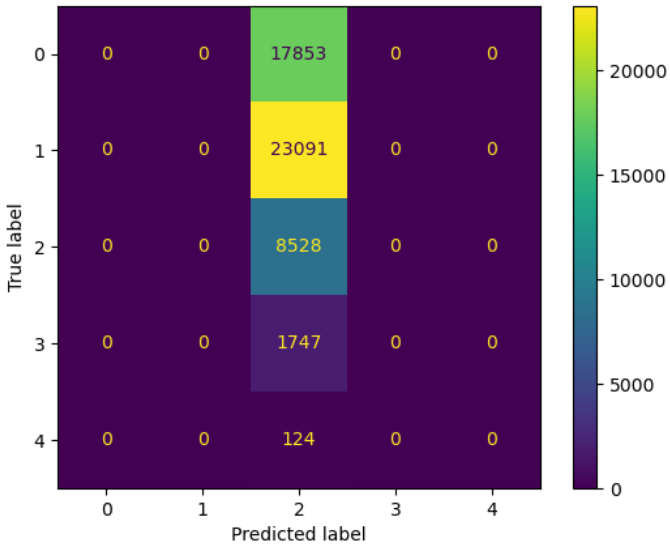


Figure 3-26: Results from testing on 10th of September – 15th of October 2021

1-second data

The model was then trained using the 1-second data, averaged over 20-second intervals, including vibration and PeakVue measurements. Short Time Fourier Transforms were calculated for the torque measurements with a window size of 1 hour (180 samples). The model was trained with data from May to 10th of September 2021 and tested with a separate test-set from the same timeframe as well as a test-set from 10th of September – 15th of October 2021. The results of the test from the same timeframe as the training set can be seen in Figure 3-27, the total accuracy is 27.5%, with a lot of false failure predictions for stopped and normal running states. As with the model trained with 50ms data also this model predicts «less than 1 week from failure» for all predictions on the dataset from September – October as seen in Figure 3-28.

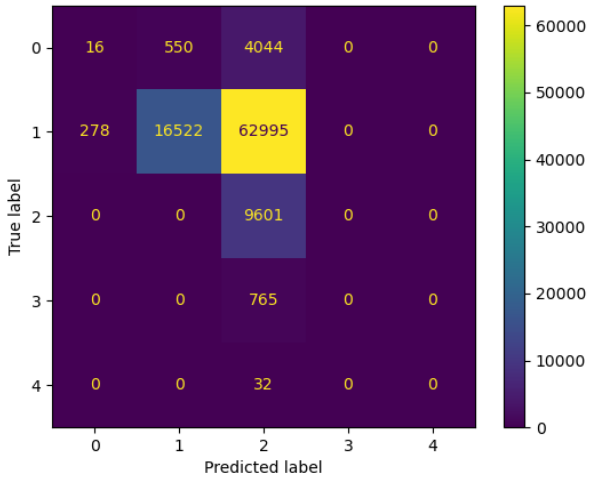


Figure 3-27: Naïve Bayes prediction results from May 2021 - September 2021

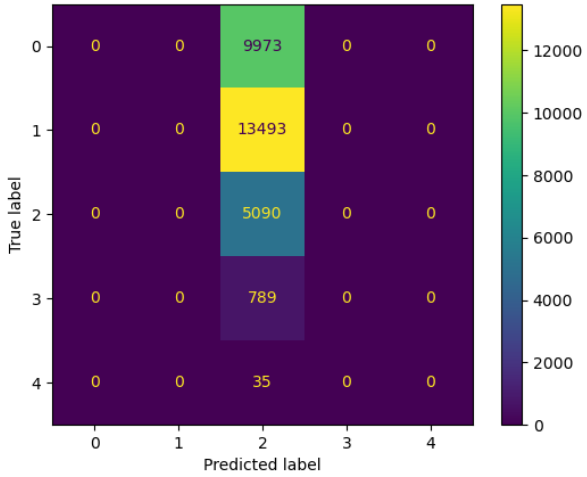


Figure 3-28: Naïve Bayes prediction results from September 2021- October 2021

3.2.2 Support Vector Machine

For this test the C-Support Vector Classification implementation of SVM in scikit-learn was used with a Radial Basis Function Kernel.

30-second data with Fourier Transform values

As for the Naïve Bayes-model training the 30s data was used with addition of Fourier transform values. The same timeframes were used for training and test sets as for the Naïve Bayes-model. But the model does predict “normal running” (Label 1) for a large part of the “less than 1 week from failing” states and some of the “less than 24 hours from failing” states. But the results look a lot more centred at the diagonal of the confusion matrix as seen in Figure 3-29.

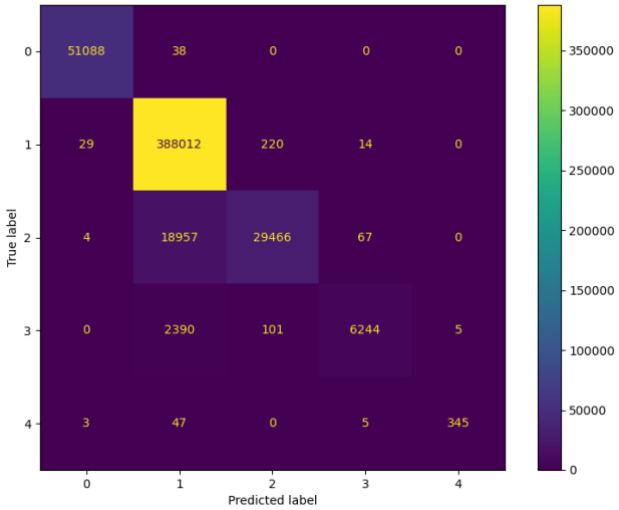


Figure 3-29: Confusion Matrix of SVM-model prediction results compared to true labels

When testing the SVM-model with data outside the timeframe of the training data, the results are not very satisfactory as seen in Figure 3-30, the model seems to predict almost every row of the dataset to be “Normally running” with just a few correctly predicted failing labels.

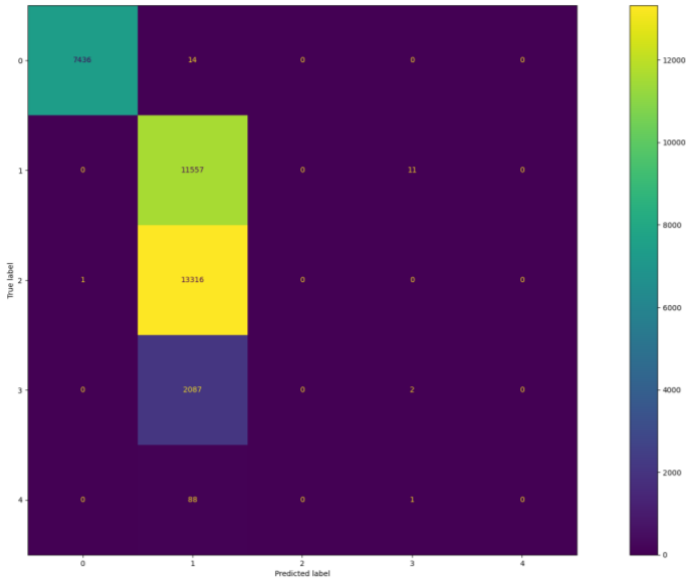


Figure 3-30: SVM-model prediction results for September 2021 30s data

50-millisecond data

Training of the model with the same 50-millisecond / 1-second averaged data as for the Naïve Bayes model, we get test results with a high 94.9% accuracy, when testing against the separate test set from the same timeframe as the training set. The test shows very high accuracy for "normal running" (Label 1, 99.6%) and "less than 1 week from failing" (Label 2, 100%), but poor results for the 3 other labels, but all "failing" labels are predicted as "less than 1 week from failing".

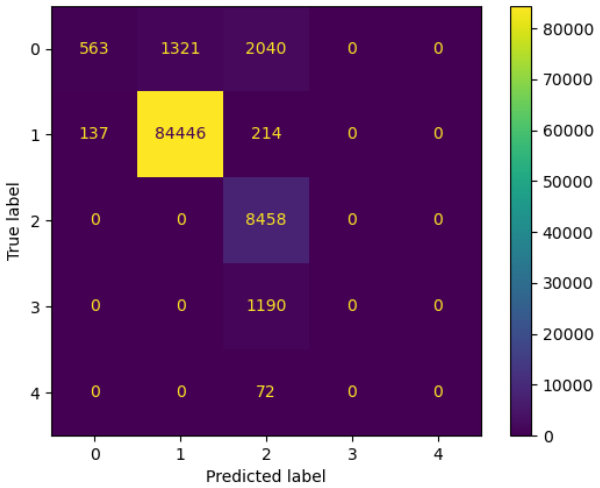


Figure 3-31: SVM-model prediction on May 2021 - September 2021

When the model is tested with data from September – October outside the timeframe of the first training and test sets, the model predicts all labels to be "less than 1 week from failing" (Label 2) as seen in Figure 3-32.

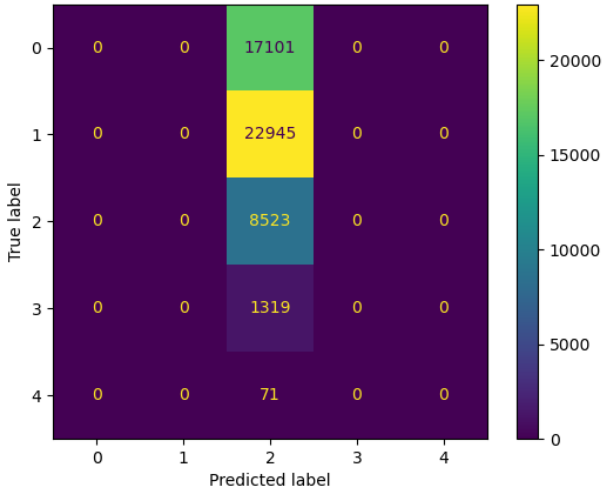


Figure 3-32: SVM-model prediction on September 2021 - October 2021

1-second data

Training of a SVM-model for the 1-second data with 20-second averages as for the Naïve Bayes model ended up with results for test set in the same timeframe as the training set as seen in Figure 3-33. The prediction results are all labelled “normal running” (Label 1) both for the test set within the same timeframe as the training set and the test set from outside the timeframe seen in resulting confusion matrix in Figure 3-34.

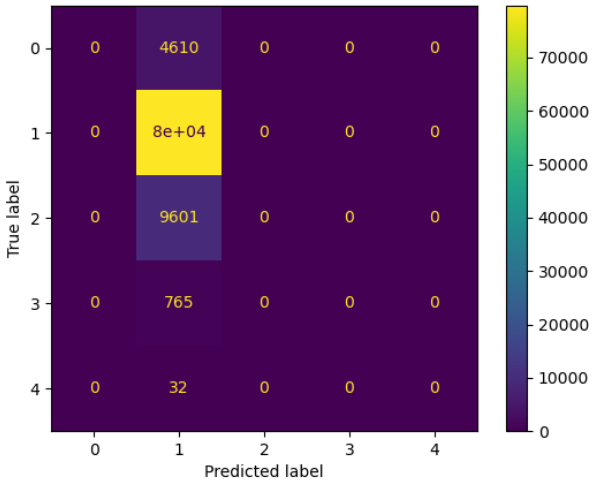


Figure 3-33: SVM-model prediction on May 2021 - September 2021 data

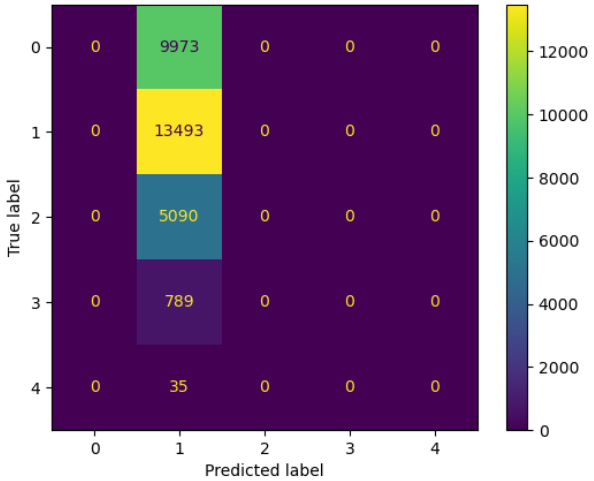


Figure 3-34: SVM-model prediction on September 2021- October 2021 data

3.2.3 Long Short-Term Memory

The LSTM-model architecture used for testing is set up as shown in Figure 3-35 for the 30s data trained model, with an input layer of 7 features with 120 timeseries rows. First hidden layer is a LSTM layer of 32 nodes with sequence return. Second hidden layer is also LSTM of 16 nodes, and lastly a Dense layer with "softmax" activation so that the 5 outputs sum is 1. Output is 5 nodes representing the 5 possible labels (0-4) one hot encoded.

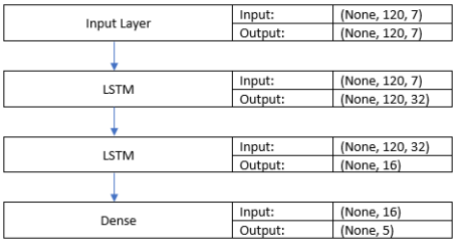


Figure 3-35: LSTM-model architecture

The model is compiled using a loss function of "categorical crossentropy" which computes the loss between the labels and the predictions, optimizer used is the "adam optimizer" which is computational efficient, and a good choice when training on large data sets or with many features [41].
The architecture for the models trained with 50-millisecond and 1-second data differs only in the input layer. LSTM-model trained with 1-second data (20 sec average) has 6 more features (total 13) and 180 timeseries rows (1 hour), while for 50-millisecond data (1 second average) the input layer has 7 features like 30-second model, and 180 timeseries rows (3 minutes).

30-second data 1-hour timeseries

The LSTM Neural network was trained using the 30-second data set from May 2020 to August 2021. 80% for training and 20% for test set. Out of the training set 10% was used for validation. The values were scaled and centred using a Standard Scaling function in this case as well as for the NB and SVM models. The scaler was fitted to the data from the training set timeframe data.

The model was set to train for 50 epochs, but set to stop if the “validation loss” rises 3 consecutive epochs. In the plot in Figure 3-36 the training losses are shown, with the training loss in blue and validation loss in red. The training ended at epoch 18 after validation loss was raised.

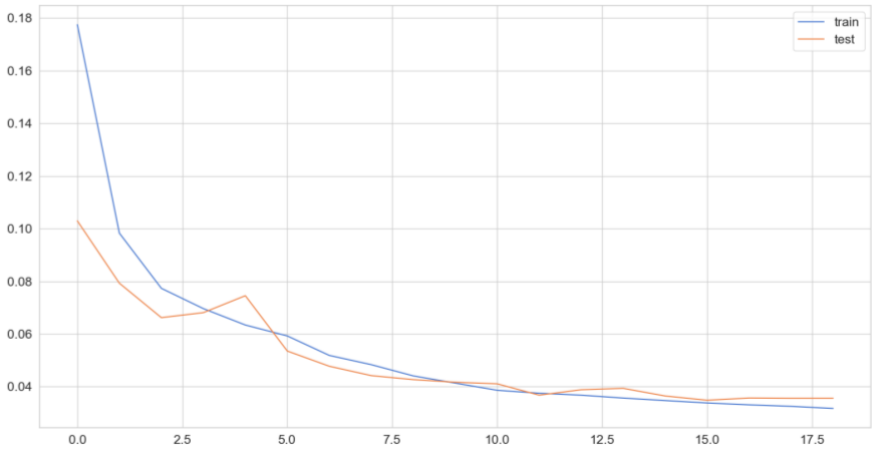


Figure 3-36: Training of LSTM NN model 18 epochs

The LSTM-model seems to be better at predicting the failing states (labels 2-4), but worse at predicting stopped and normally running states (labels 0-1) if we compare the results from the test set predictions in Figure 3-29 and Figure 3-37.

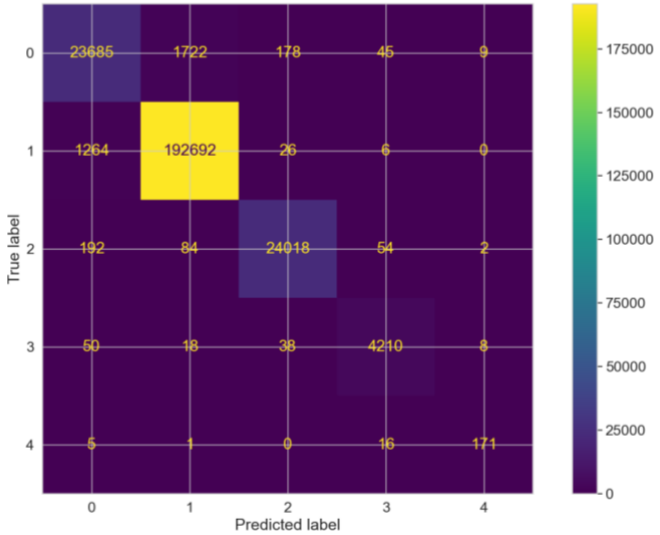


Figure 3-37: Confusion Matrix LSTM-model predictions vs true labels May 2020 – August 2021 data

Testing the model with data from September 2021 the results show a heavy bias at predicting “normal running” state as seen in Figure 3-38.

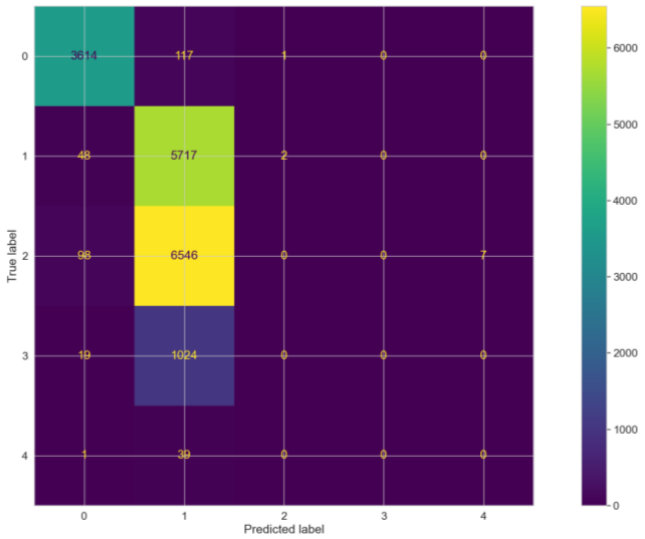


Figure 3-38: LSTM-model prediction September 2021 data

Change in standard scaling of values

After the prediction results from the models showed poor results for data from timeframes outside the timeframe of the training set data a decision was made to scale and centre the values from each individual pump based on the first hour run of normal operation after a pump had been replaced. And in this way the "normal" state of each pump might be easier to train for and predict.

Redoing the training of the LSTM-model using the new scaling method the results in Figure 3-39 was made with a total accuracy of 98.2%, 93.6% for "stopped" label, 98.9% for "normal running", 98.8% for "less than 1 week from failure", 90.6% for "less than 24 hours from failure" and 47% for "less than 1 hour from failure". Looking at the failure modes (Label 2-4) combined, by combining columns and rows 2-4 into one, a combined failure mode accuracy of 99.2% is calculated.

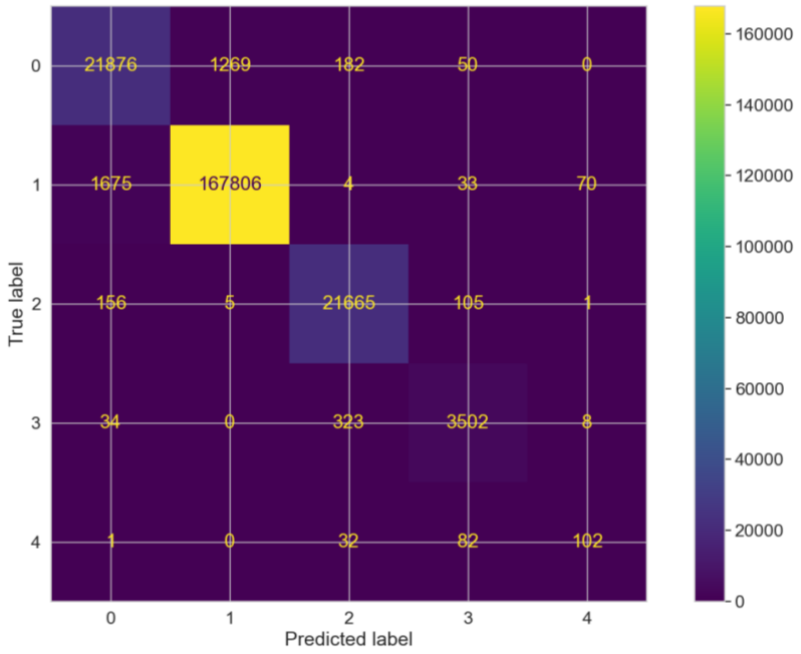


Figure 3-39: LSTM-model prediction after per pump value scaling/centering May 2020 - September 2021

The same model tested with data outside the training data timeframe shows a lot higher accuracy than the model with the original scaling method as can be seen in Figure 3-40.
The total accuracy for data outside the training set timeframe were 78.5%. Prediction accuracy for “stopped” 97.1%, “normal running” 99.2%, “Less than 1 week from failure” 55.2%, “Less than 24 hours from failure” 53.9% and “Less than 1 hour from failure” 0%.
If looking at the failure states combined a failure mode accuracy of 98.8% is calculated.

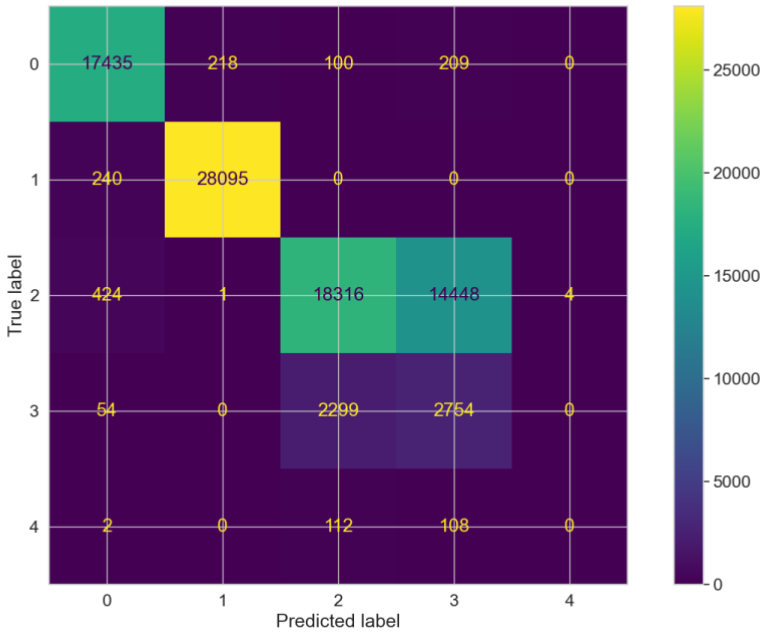


Figure 3-40: LSTM-model after per pump value scaling/centering September 2021

50-millisecond data

The LSTM-model was trained using 50-millisecond data averaged over 1-second intervals. The same number of features as the 30-second model, but with 180 timeseries samples, representing 3 minutes of data. The data was scaled and centred using the individual pump standard scaler fitting method finally used with the 30-second data. The LSTM-model was trained with data from the start of July 2021 – 10th of September with a training-/test-set split of 80%/20%, and a 10% validation-set picked from the training-set used during training. The resulting test prediction is shown in Figure 3-41, with a total accuracy of 97.6%. The label accuracies were 88% for «stopped», 99.8% for «normal running», 99.9% for «less than one week from failure», 11.4% for «less than 24 hours from failure» and 0% for «less than 1 hour from failing». The total failure mode accuracy when combined was 99.9%, because most false predictions of the failure states (label 2-4) predict one of the other failure states.

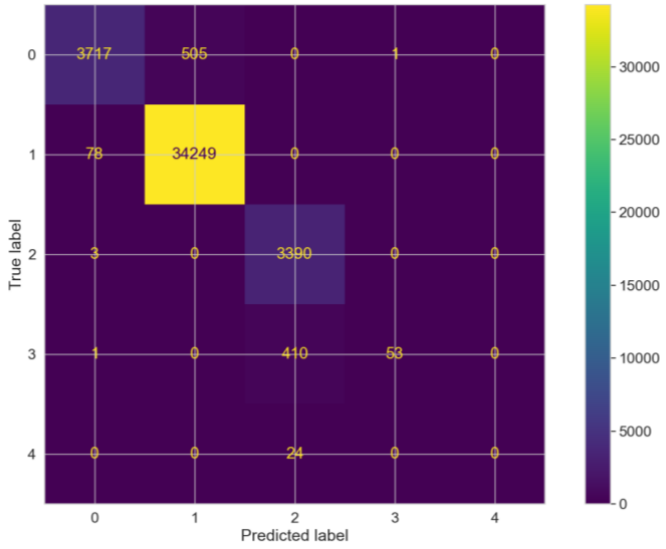


Figure 3-41: LSTM-model prediction May 2021 - September 2021

When the model was tested with data outside the training-set timeframe the results in Figure 3-42 was achieved. Total accuracy of 59.5%, “stopped” prediction accuracy of 51.2%, “normal running” accuracy of 53.6%, “less than 1 week from failure” accuracy of 99.96%, “less than 24 hours from failure” and “less than 1 hour from failure” accuracy of 0%. Combined failure model prediction accuracy is 99.97%. Compared to the 30-second LSTM-model this model gives more false predictions of eminent failures when “stopped” and “normally running”.

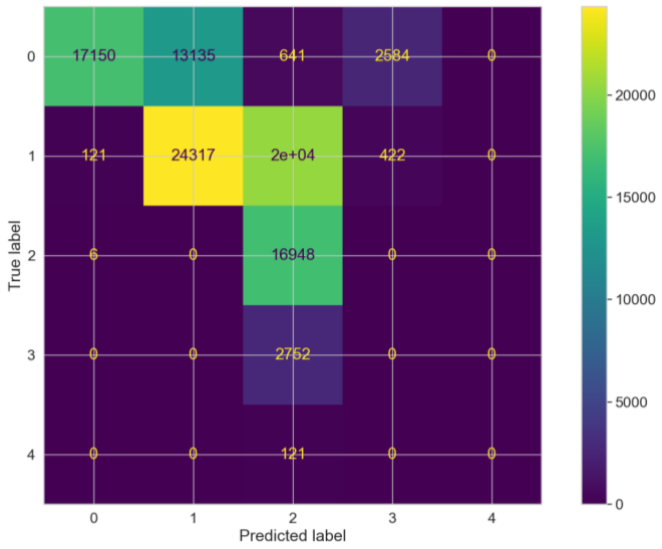


Figure 3-42: LSTM-model prediction September 2021 - October 2021

1-second data

Finally, a LSTM-model was trained using 1-second data, averaged over 20-second intervals. A total of 13 features and 180 timeseries samples (1 hour) was the input to the model. Training data and initial test set were from the time span from 21st of May 2021 to 10th of September. The randomly selected train/test split was 80/20%, with a validation set of 10% selected from the training set. The training of the model ended after 20 epochs with a training loss of 0.0132, training accuracy of 0.9956 and validation loss of 0.0159 and accuracy of 0.9948 as seen in Figure 3-43.

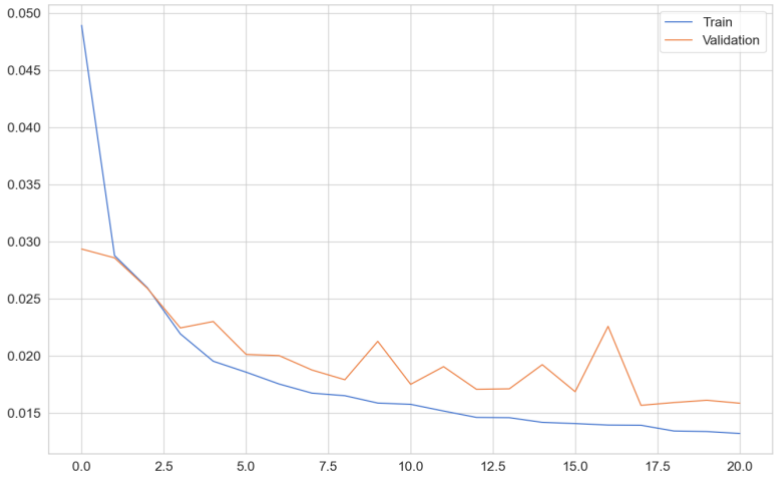


Figure 3-43: Loss plot for training of 1-second LSTM-model

Testing with initial test-set from the same time period as the training set achieved a total accuracy of 99.6%, a "stopped" label prediction accuracy of 98.3%, "normal running" accuracy of 99.7%, "less than 1 week from failure" accuracy of 99.6%, "less than 24 hours from failure" accuracy of 99.2% and a 100% accuracy for "less than 1 hour from failure", the total failure

states prediction accuracy combined were 99.6%. Details of the test results are seen in Figure 3-44.

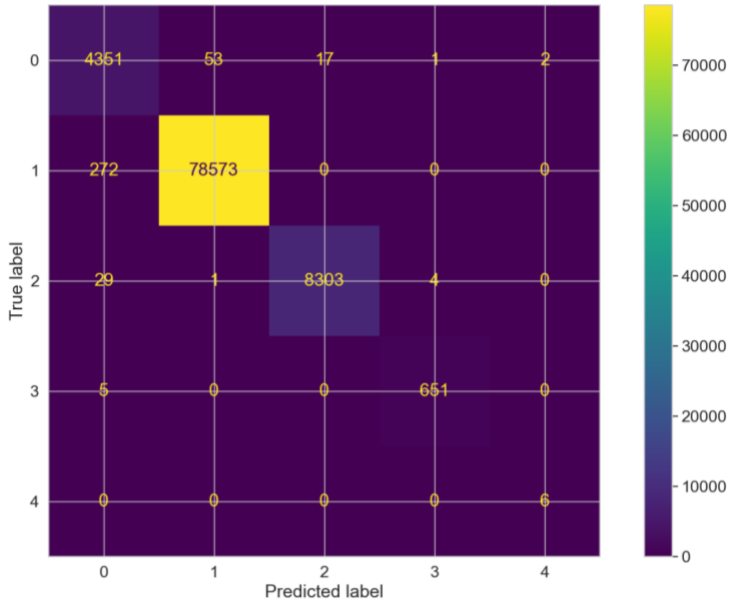


Figure 3-44: LSTM-model (1s data) prediction on May 2021 – 10th of September 2021 data

Testing with test-set from outside the time frame of the training set (10th of September – 15th of October) achieves a total accuracy of 71.8%, "stopped" label prediction accuracy of 37.5%, "normal running" accuracy of 93.1%, "less than 1 week from failure" accuracy of 99.7%, "less than 24 hours from failure" accuracy of 0% and 0% accuracy for "less than 1 hour from failure", the two last labels are mainly predicted as "less than 1 week from failure" giving a total combined failure labels prediction of 99.6%.

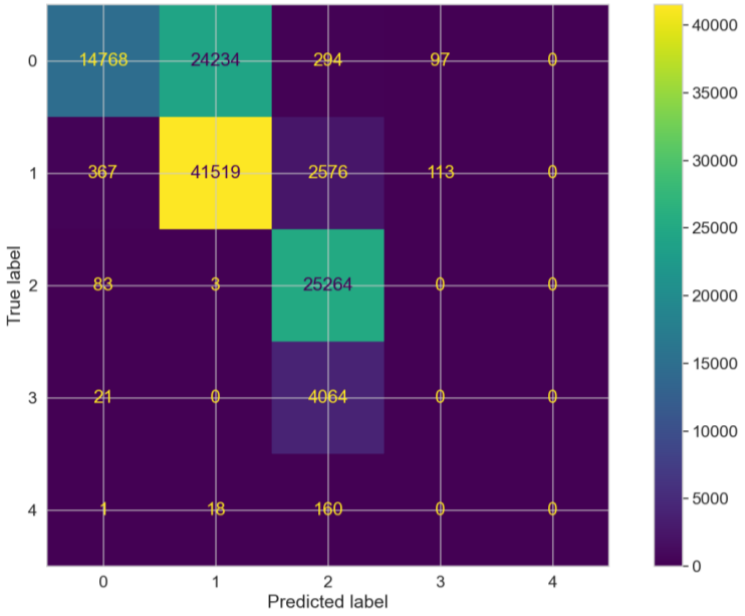


Figure 3-45: LSTM-model (1s data) prediction on 10th of September – 15th of October 2021 data

4 Discussion

Some pump replacements have been assumed as the data leads to such conclusions, one occurrence is that the maintenance log does not state anything, however the maintenance manager says in the period between 5th of May and 9th of September, there has been a replacement, in a period where there lacks process (16th June to 17th June) it has been assumed a pump replacement.

The code for arranging the downloaded data into a Pandas data frame aligns the length of each measurement for the 30 second and 1 second data into equally long lists, these should have been merged to a data frame instead such that each timestamp was aligned. What may happen as it has been done is that the end is cut off to make the lists equally long, which may shift which datapoints that should form a sample set of the variables.

Outlier detection in predictive maintenance is utterly different from general case applications since anomalies and outliers are actually the data which represents possible fault status. Distance and density-based models basically detect other clusters which cannot be removed. As it shown in the data analysis section, KNN catches the data just around the densely and far off data. Playing with outlier_index in the KNN method does not provide sufficient results. It is evident that distance and density-based methods are not in line with expectations. In this case each data set have to be investigated by using process and engineering knowledge. There is no single approach that could be applicable for all data. It is concluded that focusing some peak values and understanding the values before removing them is the best approach since zero values of the pump are not the outliers. Apparently, vibration data has the most peak values so removing top values approach fits well. While specifying a threshold and removing values above the threshold is one of the solutions, averaging the data within a specific time frame is another solution to deal with the outliers. IQR method is used for removing outliers by determining specific threshold. Furthermore, taking 20 data in 30s dataset, averaging them and producing 1 data which represents 20 data is the best suitable approach for out dataset. Consequently, we do not only avoid data loss but also the correlation is increased within the dataset.

The vibration sensors appear promising, however due to there only being three faults recorded with the vibration it cannot be concluded that they work nor that they do not work for the objective of identifying faulty pumps. For these three failures, the faults were not the same or the data was constantly close to zero.

From the PCA analyses it seems like the speed of the pump is the most important factor of the system in the 30 second data set and 50ms dataset, while this may explain the system's behaviour well, it will not explain the mechanical state very well due to the simple reason that the operators can at will set this to an arbitrary number between 0 and 100. For the 1s data set, the vibration measurements were the most important.

For the SVM and Naïve Bayes models the Fourier transforms were done on the averaged data, this would likely have given quite another effect on the 50ms data if done on the unprocessed dataset.

5 Conclusion

The most important variables found during PCA analysis for the system was for the 30 second data, and the 50ms data the speed (PU19_SF_PV). For the 1 second data set the vibration data was the most important, specifically the velocity on the inlet and outlet (PU19_V_I and PU19_V_O respectively).
The performance of the models on data outside the timeframe of the data the model was trained on is the most important criteria to evaluate the models, and on this criterion the LSTM-models trained using 30-second data and 1-second data are the most accurate. The total accuracy of the LSTM-model trained on 30-second data were 78.5%, with high accuracy on predicting "stopped", 99.2%, and "normal running" states, 98.2%, which means the model does not falsely predict a lot of eminent failures for these states. The model trained on 1-second data, with data from vibration sensors, had a total accuracy of 71.8%, with only 37.5% accuracy for "stopped" states, slightly less accuracy for "normal running" state at 93.1%, and a bit higher combined accuracy for the failure states combined, which is impressive considering that there are a lot less failures in the training data for the model, than what is present in the training data for the 30-second LSTM-model.

5.1 Future work

5.1.1 Reporting of replacement

From the work in this report, it is clear that information regarding failures must be improved for these types of projects to succeed in a reasonable time. Firstly, that all of the equipment needs to be logged when it is replaced, secondly more details regarding what was wrong with the equipment such that each type of fault can be modelled and discriminated between. Thirdly it should be logged structurally to reduce time spent on finding the faults and the relating timestamp.

5.1.2 Retrain model over time

To really discern whether the vibration data can be used further on it should be tested with a larger data set with more faults, this also goes for the 50ms data. As both these data sets have higher granularity it would make sense that these would have more information.
Retraining the model when more data is available should improve the model over time, gaining higher accuracy. The models trained on the 50ms- and 1-second data, would most likely benefit highly from this due to low number of faults recorded in the data used for this project. To reduce computational workload reinforced training may be applied.

5.1.3 Live implementation

To use the findings at the factory, a system must be implemented to continually monitor the pumps and run the measurements through the final model(s). In addition, this system needs to communicate with other systems to give messages to the operators.

6 Bibliography

[1] "NEMO® Progressing Cavity Pumps," 7 September 2021. [Online]. Available: https://d2brmtk65c6tyc.cloudfront.net/media/pumps/NPS/Brochures/NEMO/NEMO-Exzenterschneckenpumpe_0721_EN.pdf?1627291535&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6XC9cL2QyYnJtdGs2NWM2dHljLmNsb3VkZnJvbnQubmV0XC9tZWRpYVwvcHVtcHNcL05QU1wvQnJvY2h1cmVz.

[2] J. Lea, H. Nickens and W. Michael, "Chapter 13 - Progressive Cavity Pumps," in *Gas Well Deliquification,* Gulf Professional Publishing, 2003, pp. 251-26.

[3] J. Elsey, "Pump&Systems," 12 06 2017. [Online]. Available: https://www.pumpsandsystems.com/progressive-cavity/beginners-guide-progressive-cavity-pumps.

[4] Deloitte, "Predictive Maintanence," 2017. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/de/Documents/deloitte-analytics/Deloitte_Predictive-Maintenance_PositionPaper.pdf.

[5] Mainnovation and PWC, "Predictive Maintenance 4.0," PWC, The Netherlands, 2018.

[6] F. W. W. Z. M. G. L. L. D. S. Jay Lee, "Prognostics and health management design for rotary machinery systems - Reviews, methodology and applications," 2014. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0888327013002860.

[7] A. Muller and S. Guido, Introduction to Machine Learning with Python, California: O'Reilly Media Inc., 2017.

[8] J. P.-V. M. K. G. E. Andreas Theissler, "Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry," *Reliability Engineering & System Safety,* vol. Volume 215, 2021.

[9] D. L. I. A.-M. D. G.-G. I. M. M. G.-B. M. A.-R. J. L. F. H. Jacinto Carrasco, "Anomaly detection in predictive maintenance: A new evaluation framework for temporal unsupervised anomaly detection algorithms," *Neurocomputing,* pp. 440-452, 2021.

[10] P. Peeling, "Matlab Expo 2017," 08 September 2021. [Online]. Available: https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/uk/2017/big-data-machine-learning-for-predictive-maintenance.pdf.

[11] E. Florian, F. Sgarbossa and I. Zennaro, "Machine learning-based predictive maintenance: A cost-oriented model for implementation," *International Journal of Production Economics,* vol. 236, 2021.

[12] Mathworks, "Introduction to Predictive Maintenance with MATLAB," 11 September 2021. [Online]. Available:

https://se.mathworks.com/content/dam/mathworks/ebook/predictive-maintenance-ebook-part1.pdf.

[13] A. Andryushin, I. Shcherbatov, N. Dolbikova, A. Kuznetsova and G. Tsurikov, "Outlier Detection in Predictive Analytics for Energy Equipment," in *Cyber-Physical Systems: Advances in Design & Modelling*, Springer Nature Switzerland, 2020, pp. 193-203.

[14] R. Suri, N. Murthy and A. G., Outlier Detection: Techniques and Applications, India: Springer Nature Switzerland, 2019.

[15] P. Kamal and R. Sugandhi, "Anomaly Detection for Predictive Maintenance in Industry 4.0- A survey," *E3S Web Conferences,* vol. 170, p. 8, 2020.

[16] N. Bhushan and A. S. Rathore, "Use of multivariate data analysis (MVDA) for generating process understanding from manufacturing data of biotech processes," Proceedings of the Indian national Science Academy, 2011.

[17] Sartorius, "What Is Principal Component Analysis (PCA) and How It Is Used?," Sartorius, 18 August 2020. [Online]. Available: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186. [Accessed 26 September 2021].

[18] J. Beyerer, R. M. and N. M., Pattern Recognition, Berlin: De Gruyter, 2018.

[19] E. Davies, Computer and Machine Vision: Theory, Algorithms, Practicalities, Oxford: Elsevier, 2012.

[20] N. Zhang, L. Wu, J. Yang and Y. Guan, "Naive Bayes Bearing Fault Diagnosis Based on Enhanced Independence of Data," *Sensors 2018,* p. 463, February 2018.

[21] Scikit learn, "1.9. Naive Bayes," Scikit learn, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed September 2021].

[22] R. Pupale, "Support Vector Machines(SVM) — An Overview," Towards data science, 16 June 2018. [Online]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. [Accessed 25 September 2021].

[23] B.-S. Yang and A. Widodo, "Pattern Recognition for machine fault diagnosis using Support Vector Machine," in *Support Vector Machines*, Nova Science Publishers, Incorporated, 2011.

[24] S. J. H. a. G. S. M. T. Sarmiento, "Fault detection in reactive ion etching systems using one-class support vector machines," in *IEEE/SEMI Conference and Workshop on Advanced Semiconductor Manufacturing*, 2005.

[25] C.-L. H. Te-Sheng Li, "Defect spatial pattern recognition using a hybrid SOM–SVM approach in semiconductor manufacturing," *Expert Systems with Applications,* vol. 36, no. 1, pp. 374-385, 2009.

[26] S. Sharifi, A. Tivay, M. Rezaei and M. Zareinejad, “Leakage fault detection in Electro-Hydraulic Servo Systems using a nonlinear representation learning approach,” *ISA Transactions,* vol. 73, pp. 154-164, 2018.

[27] R. Jegadeeshwaran and V. Sugumaran, “Fault diagnosis of automobile hydraulic brake system using statistical features and support vector machines,” *Mechanical Systems and Signal Processing,* Vols. 52-53, pp. 436-446, 2015.

[28] A. Navlani, “Support Vector Machines with Scikit-learn,” Datacamp, 27 December 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python. [Accessed 25 October 2021].

[29] X. Bampoula, G. Siaterlis and N. Nikolakis, “A Deep Learning Model for Predictive Maintenance in Cyber-Physical Production Systems Using LSTM Autoencoders,” *Sensors,* vol. 21, p. 972, 2021.

[30] IBM, “Recurrent Neural Networks,” IBM, 14 September 2020. [Online]. Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks. [Accessed 28 September 2021].

[31] A. Afshine, “Recurrent Neural Networks,” [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks. [Accessed 28 September 2021].

[32] S. Hochreiter and J. Schmidhuber, “LONG SHORT-TERM MEMORY,” *Neural Computation,* vol. 9, no. 8, pp. 1735 - 1780, 1997.

[33] A. Rivas, J. Fraile and P. Chamso, “A Predictive Maintenance Model Using Recurrent Neural Networks,” *Springer Nature,* pp. 261 - 270, 2019.

[34] A. R. G. A. J. Wheeler, “Discrete Sampling and Analysis of Time-Varying Signals,” in *Introduction to Engineering Experimentation, 3rd Edition,* New York City, Pearson, 2009, pp. 102-127.

[35] F. a. V. G. a. G. A. a. M. V. a. T. B. a. G. O. a. B. M. a. P. P. a. W. R. a. D. V. a. V. J. a. P. A. a. C. D. a. B. Pedregosa, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[36] Emerson, “AMS Asset Monitor Online Prediction, Protection and Process Monitor,” [Online]. Available: https://www.emerson.com/no-no/automation/asset-management/machinery-health-management/ams-asset-monitor. [Accessed 27 09 2021].

[37] TimescaleDB, “Advanced analytic queries,” 11 11 2021. [Online]. Available: https://docs.timescale.com/timescaledb/latest/how-to-guides/query-data/advanced-analytic-queries/#first-and-last.

[38] L. D. M. G. &. J. L. Steve Berman, “Simpson’s Paradox: a cautionary tale in advanced analytics,” Significance, 25 09 2021. [Online]. Available:

https://www.significancemagazine.com/14-the-statistics-dictionary/106-simpson-s-paradox-a-cautionary-tale-in-advanced-analytics. [Accessed 03 11 2021].

[39] Pandas, “pandas.DataFrame.quantile,” [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.quantile.html. [Accessed 12 11 2021].

[40] N. Instruments, “Understanding FFTs and Windowing,” [Online]. Available: https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf. [Accessed 05 10 2021].

[41] J. Brownlee, “Machine Learning Mastery,” 13 01 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed 05 11 2021].

[42] A. D. L. M. f. P. M. i. C.-P. P. S. U. L. Autoencoders, “A Deep Learning Model for Predictive Maintenance in Cyber-Physical Production Systems Using LSTM Autoencoders”.

Appendices

Appendix A Task Description
Appendix B Database interface
Appendix C Data wrangling and calls to database
Appendix D Date list manipulation
Appendix E Plot histogram, scatter matrix plot, PCA
Appendix F Call to get data, wrangling and plotting
Appendix G Increased size of Figure 2-13
Appendix H Call for time-series plots
Appendix I Outlier Removal
Appendix J Statistics in Chapter 2.5.3
Appendix K IQR and removing samples where pump is off
Appendix L Call to get, wrangle and plot average data
Appendix M Increased size of Figure 2-26
Appendix N Increased size of Figure 2-28
Appendix O Code for analysis and Fourier Transform plotting
Appendix P Code for training and testing Naïve Bayes and SVM models
Appendix Q Code for training and testing LSTM-models

Appendix A Task Description

Title: Machine Learning for Predictive Maintenance of pumps at "Den Magiske Fabrikken"
USN supervisor: Carlos Pfeiffer
Co-supervisor: Håkon Viumdal
External partner: Lindum AS by Bertil Johansen

Task background:

Lindum owns several factories treating animal manure, food and human waste for biogas production. The process heavily relies on pumping acidic and hazardous substrate, which cause corrosion and abrasion on the pumps. Most pumps are changed on a frequency of two to three months, either when the pumps are no longer functional, or when a worker decide to change them if they believe the pump will fail soon. The decision to overhaul or replace a pump is usually based on the sound, the feel of vibrations by touch, and other pump's characteristics that according to the workers experience and intuition indicate that the pump will fail soon. It is uncertain how many spare parts are needed on site and when work on the pumps is needed. The pumps in question are of type progressive cavity pumps.

Task description:

The aim of this project is to evaluate different Machine Learning (ML) techniques to predict when pumps need maintenance. The data available (provided by Lindum) are registers from vibrations sensors, pressure, momentum, speed and electric signals from various pumps.

The following activities describe the main tasks:

a) Literature survey on progressive cavity pumps, predictive maintenance, and varying methods of machine learning (ML) that will be feasible for this approach.
b) Pre-process the data, to be prepared for machine learning algorithms.
c) Select the most appropriate ML algorithms to estimate the condition of the pumps (good or in need of replacement), and if possible, estimate the remaining lifetime.
d) Analysis of which sensors and electric measurements are useful to determine the condition of the pumps.
e) If possible, test the algorithm live on one or several pumps. This activity heavily relies on the pumps on the factory and the time it takes to create the programs needed.

Some comments on suggested Machine Learning Techniques suggested as starting points: Classic Pattern Recognition techniques involve the explicit selection and calculation of signal characteristics and feature extraction, for example frequency components based on Fourier analysis, decomposition of signal using wavelets, specially designed filter banks, etc. It would be interesting to test and evaluate some of these techniques.

For the actual classification part, some suggested techniques that can be explored are Support Vector Machines, Naïve Bayes, Principal Components Analysis, and classic three layers Back Propagation Neural Networks, where the calculated characteristics are supplied.

Matlab has very good toolboxes for these techniques. The use of Deep Learning Neural Networks could also be explored and evaluated, since now there are several packages to do

it.

Student category:
IIA-students. This task is connected to one Industry master student at IIA. In addition both online and campus based IIA-students are able to select this task. The number of students recommended for this project are three or maximum four.

Practical arrangements:
The students will have access to the necessary dataset from the factory. However, it is recommended to visit the physical factory for all the group members during the first month. The students should arrange for their own transportation. (Lindum address: Taranrødveien 97, 3171 Sem).

The Industry master student, Martin Holm, will participate as a member of the master project team, and he will have a special responsibility to provide the necessary data sets.

The main recommended programming language will be Python. Other tools like Matlab, R, etc. can also be used for the analysis of data and the testing of the different methods.

Signatures:
Supervisor (date and signature):

_____
Co-supervisor (date and signature):

_____
Students (date and signature):

24.09.2021 Martin Holm
Martin Holm

23/09/21
Ozgur Yalcin

22/9-21 Ronnie K. Moe
_____
Ronnie Moe

Appendix B Database interface

```
import psycopg2 as pg
import numpy as np
import xml.etree.ElementTree as ET

class XML:
    xmlFile = None
    xmlObject = None
    root = None

    def __init__(self, xmlFile):
        """
        initialize an XML object
        :param xmlFile: string, name of XML file
        """
        self.xmlFile = xmlFile
        self.xmlObject = ET.parse(xmlFile)
        self.root = self.xmlObject.getroot()


    def SearchForTag(self, searchitem):
        """
        Searches the xml file for searchitem tag

        :param searchitem: string, XML tag in XML file
        :return: string, value of searchitem
        """

        for item in self.root.iter(searchitem):
            text = item.text

        return text

def connect(xml_file='./DBConnection.xml'):
    """
    Connects to a database using an xml file
    :return: connected psql object
    """

    xml = XML(xml_file)

    database = xml.SearchForTag('database')
    host = xml.SearchForTag('host')
    user = xml.SearchForTag('user')
    pw = xml.SearchForTag('pw')
    sql = psql(database, host, user)
    sql.connect(pw)
    return sql


class psql:
    database = ""
    server = ""
    username = ""
```

```
conn = pg
cur = conn
pw = None
connected = False
error = None

def __init__(self, database, server, username):
    """
    Creates a psql object

    :param database: databasename
    :param server: hostaddress
    :param username: username for the database
    """
    self.database = database
    self.server = server
    self.username = username


def connect(self, pw):
    """
    connects the psql object to a database and creates a cursor within the psql object

    :param pw: password
    :return:
    """
    self.pw = pw
    try:
        self.conn = self.conn.connect("dbname = '{0}' user = '{1}' host = '{2}' password = '{3}'".format(
            self.database, self.username, self.server, pw))

        self.cur = self.conn.cursor()

        self.connected = True
    except:
        print("Failed to connect")
        self.error = "Failed to connect"
        self.connected = False

def reconnect(self):
    try:
        self.conn = self.conn.connect("dbname = '{0}' user = '{1}' host = '{2}' password = '{3}'".format(
            self.database, self.username, self.server, self.pw))

        self.cur = self.conn.cursor()
        print("connected to {0}".format(self.database))
    except:
        print("Failed to connect")

def disconnect(self):
    """
    disconnects from the database

    :return:
    """
```

```
    self.conn.close()
    print("disconnected from {}".format(self.database))

def insert(self, query):
    self.cur.execute(query)

def q_select(self, query, params=None):
    """
    Sends a query to the connected database and returns the data

    :param query: string, PostgreSQL query
    :return: object of rows
    """
    rows = []
    if params == None:
        try:
            self.cur.execute(query)
            rows = self.cur.fetchall()
        except pg.errors.InFailedSqlTransaction:
            rows = False

    else:
        try:
            self.cur.execute(query, params)
            rows = self.cur.fetchall()
        except pg.errors.InFailedSqlTransaction:
            rows = False

    return rows

def send_q(self, query, data=None):
    try:
        self.cur.execute(query, data)
        self.conn.commit()
    except pg.errors.UniqueViolation as e:
        print(e)
        self.sql.conn.rollback()
    except pg.errors.SyntaxError as e:
        print(e)
    except pg.errors.InvalidTextRepresentation as e:
        print(e)


def transpose_sql_query(self, rows):
    """
    Extracts the 0th and 1st columns from rows and transposes them

    :param rows: cursor object, data returned from a query
    :return: list, 0th and 1st column of rows
    """
    time = []
    val = []
    for row in rows:
        time.append(row[0])
        val.append(row[1])
```

```
        data = [time, val]
        return data

    def get_tagnames_from_station(self, station_name):
        """
        Selects all tags related to a station name

        :param station_name: string, name of the station
        :return: list of string, tagnames related to station_name
        """
        query = """select tag from tag where station_id =
        (select station_id from station where station_name = '{}')
        order by tag_id desc""".format(station_name)
        self.cur.execute(query)
        tagnames = self.cur.fetchall()
        tagnames2 = []
        for row in tagnames:
            a = row[0]
            b = "(',)"
            for char in b:
                a = a.replace(char, "")
            tagnames2.append(a)
        return tagnames2



class Tag:
    """
    Class to hold a process tags information
    """
    tagID = None
    connected_tag = None
    station_id = None
    unit_type = None
    tag = None
    tag_desc = None
    interval = None
    interval_unit = None
    pw = None
    timestamp = None
    measurements = None

    sql = None

    def __init__(self, tagname):
        """
        Object of a tag

        :param tagname: string, name of tag
        """

        """Connects to database"""
        self.sql = connect()

        """Gets metainformation of the given tag"""
```

```
        self.get_tag(tagname)

    def get_tag(self, tagname):
        """
        Gets meta information of a tag from table tag in database and sets properties in the Tag object

        :param tagname: string, name of tag
        :return:
        """

        query = """select * from tag where tag = '{}'""".format(tagname)
        rows = self.sql.q_select(query)

        self.tagID = rows[0][0]
        self.connected_tag = rows[0][1]
        self.station_id = rows[0][2]
        self.unit_type = rows[0][3]
        self.tag = tagname
        self.tag_desc = rows[0][5]
        self.interval = rows[0][6]

    def get_measurement(self, time_from, time_to, table='measurement'):
        """
        Gets measurement for the given time period and tag

        :param time_from: string or datetime, Start time for dataset
        :param time_to: string or datetime, Stop time for dataset
        :param table: string, name of table in database
        :return:
        """
        query = """select distinct meas_time, meas_value
              from {}
              where tag_id = '{}' and meas_time >= '{}' and meas_time < '{}'
              order by meas_time asc;""".format(table, self.tagID, time_from, time_to)

        rows = self.sql.q_select(query)
        self.timestamp = []
        self.measurements = []
        for row in rows:
            self.timestamp.append(row[0])
            self.measurements.append(float(row[1]))

    def append_measurement(self, time_from, time_to, table='measurement'):
        """
        Gets measurement for the given time period and tag

        :param time_from: string or datetime, Start time for dataset
        :param time_to: string or datetime, Stop time for dataset
        :param table: string, name of table in database
        :return:
        """
        query = """select distinct meas_time, meas_value
              from {}
              where tag_id = '{}' and meas_time >= '{}' and meas_time < '{}'
              order by meas_time asc;""".format(table, self.tagID, time_from, time_to)
```

```
        rows = self.sql.q_select(query)

        for row in rows:
            self.timestamp.append(row[0])
            self.measurements.append(float(row[1]))

    def get_avg_measurement(self, time_from, time_to, agg_time, aggregation, table='measurement'):
        """
        Gets measurement for the given time period and tag

        :param time_from: string or datetime, Start time for dataset
        :param time_to: string or datetime, Stop time for dataset
        :param agg_time: int or float, minutes to average
        :param aggregation: string, type of aggregation
        :param table: string, name of table in database
        :return:
        """

        query = """
        SELECT time_bucket('{} minutes', meas_time) AS avg_min, {}(meas_value)
        FROM {}
        where tag_id = %s and meas_time between %s and %s
        GROUP BY avg_min
        ORDER BY avg_min asc;""".format(agg_time, aggregation, table)

        rows = self.sql.q_select(query, (self.tagID, time_from, time_to))
        self.timestamp = []
        self.measurements = []
        for row in rows:
            self.timestamp.append(row[0])
            self.measurements.append(float(row[1]))


def select_high_hyg(start, stop, tagname='*', table='high_speed_big', q_tag_list=False, numpy_arr=True):
    sql = connect('./DBconnection.xml')
    if q_tag_list == True:
        rows = sql.q_select("SELECT column_name FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME = 'high_speed_big';")
        tags = []

        for row in rows:
            tags.append(row[0])

    taglist = ''

    if len(tagname) > 1:
        for tag in tagname:
            taglist += tag + ', '
        taglist = taglist[:-2]


    query = """select {} from {} where meas_time >= '{}' and meas_time <= '{}' order by meas_time asc;"""\
        .format(taglist, table, start, stop)
```

```
    rows = sql.q_select(query)
    data = []

    if numpy_arr == True:
        data = np.array(rows).T
    else:
        for row in rows:
            data.append(row)

    del rows
    sql.disconnect()
    del sql
    if q_tag_list == True:
        return tags, data
    else:
        return data

def select_high_hyg_agg(start, stop, agg_time, tagname='*', q_tag_list=False, numpy_arr=True,
                        aggregation=''):
    sql = connect('./DBconnection.xml')
    if q_tag_list == True:
        rows = sql.q_select("SELECT column_name FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME = 'high_speed_big';")
        tags = []

        for row in rows:
            tags.append(row[0])

    taglist = ''

    if len(tagname) > 1:
        #Skip meas_time
        if aggregation == '':
            for tag in tagname[1:]:
                taglist += tag + ', '
            taglist = taglist[:-2]
        else:
            for tag in tagname[1:]:
                taglist += aggregation + '(' + tag + '), '
            taglist = taglist[:-2]

    query = """  --High speed 1min
    SELECT time_bucket('{} second', meas_time) AS five_min, {}
    FROM high_speed_big_reduced
    where meas_time between %s and %s
    GROUP BY five_min
    ORDER BY five_min asc;"""\
        .format(agg_time, taglist)

    rows = sql.q_select(query, (start, stop))
    data = []

    if numpy_arr == True:
        data = np.array(rows).T
    else:
```

```
    for row in rows:
        data.append(row)

del rows
sql.disconnect()
del sql
if q_tag_list == True:
    return tags, data
else:
    return data
```

Appendix C Data wrangling and calls to database

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split

from stat_fun import dual_date_list_interval_per_unit, date_list
from postgres import *


def equalize_tag_length(tags):
    """Takes a list of tags and makes sure they have measurement lists of equal length

    Args:
        tags (list[Tag]): List of tags

    Returns:
        list[Tag]: List of same-length tags
    """
    # double amount of measurements of the first tag
    # Instead dont x2 beause we want the shortest
    shortest_meas_length = len(tags[0].measurements) * 2

    # for each tag
    for i in range(len(tags)):
        print(len(tags[i].measurements))
        #
        amount_of_measurements = len(tags[i].measurements)
        if amount_of_measurements < shortest_meas_length:
            shortest_meas_length = amount_of_measurements
            print('new shortest_meas_length =  {}'.format(shortest_meas_length))

    print('final shortest_meas_length = {}'.format(shortest_meas_length))

    b = 0
    # for each tag
    for i in range(len(tags)):
        current_meas_length = len(tags[i].measurements)
        print(tags[i].tag)
        print(current_meas_length > shortest_meas_length)
        if current_meas_length > shortest_meas_length:
            b = shortest_meas_length - current_meas_length
            # If this measurement length is longer than the shortest
            # Perhaps this should be b<0
            if b < -1:
                # Replaces the measurements of a tag with a sliced version of itself
                # lets say b = -20
                # -> measurements[1:b+1] will make a copy that does not include the first value and stops at len-20-
1+1
                tags[i].measurements = tags[i].measurements[1:b + 1]
                tags[i].timestamp = tags[i].timestamp[1:b + 1]
            else:
                # Deletes the last element
                del tags[i].measurements[-1]
                del tags[i].timestamp[-1]
            # Looks like there might be something wrong here
```

```
            # My best guess is that at least the first if should be b<0, and slice from 0 to b-1

    # trimmed?
    print("Fjernet")

    # for each tag
    for i in range(len(tags)):
        # Prints length of measurements for each tag to check if they are the same.
        print(len(tags[i].measurements))
    return tags


def merge_df(df1, df2):
    print(df1.columns)

    #Remove duplicates of both dataframes
    df1.drop_duplicates(keep='first')
    df2.drop_duplicates(keep='first')


    #Reindex short array
    B = df2.set_index('meas_time').reindex(df1.set_index('meas_time').index, method='ffill').reset_index()

    #Merge dataset
    df = pd.merge_asof(df1, B, on='meas_time')
    df = df.dropna(axis=0)
    return df


def get_narrow_table_extract_over_time(date_start, date_stop, tagnames, table, min_per_h=None,
agg_time=None,
                                       avg=False, std_dev=False):
    """
    Connects to DB, gets data for given tagnames in the time range between date_start and date_stop with
    min_per_h minutes per hour in the range
    :param date_start: string, start time
    :param date_stop: string, stop time
    :param tagnames: list of strings, names of the tags
    :param min_per_h: int, number of minutes per hour
    :param table: string, name of table, either 'high_speed_big', 'high_speed_big'
    :return: list of tagnames, matrix of data
    """
    # Split time frame up
    if min_per_h != None:
        date_1, date_2 = dual_date_list_interval_per_unit(date_start, date_stop, min_per_h)
    else:
        if table == 'measurement':
            date_1 = date_list(date_start, date_stop, 1, day=True)
        else:
            date_1 = date_list(date_start, date_stop, 12, hour=True)
    tags = []
    # Narrow tables
    if not avg and not std_dev:
        for i in range(len(tagnames)):
            tags.append(Tag(tagnames[i]))
```

```
            tags[i].get_measurement(date_1[0], date_2[0], table=table)
            print(tagnames[i])

            for j in range(1, len(date_1)):
                tags[i].append_measurement(date_1[j], date_2[j], table=table)


    else:
        if avg:
            agg = 'avg'
        elif std_dev:
            agg = 'stddev_samp'
        for i in range(len(tagnames)):
            print(tagnames[i])
            tags.append(Tag(tagnames[i]))
            tags[i].get_avg_measurement(date_1[0], date_1[1], agg_time, agg, table=table)

            for j in range(1, len(date_1) - 1):
                tags[i].append_agg_measurement(date_1[j], date_1[j + 1], agg_time, agg, table=table)

    tags = equalize_tag_length(tags)
    data = [tags[0].timestamp]
    for x in tags:
        data.append(x.measurements)

    data = np.array(data, dtype=object)
    data = np.transpose(data)
    tag = ['meas_time']
    for x in tagnames:
        tag.append(x[4:])

    # Convert to pandas dataframe
    df = pd.DataFrame(data)

    df.columns = tag
    df['meas_time'] = pd.DatetimeIndex(df['meas_time'])

    # Set datetime datatype
    df['meas_time'] = df['meas_time'].astype('datetime64[ns]')
    # Set float datatype for numeric values
    for name in tag[1:]:
        df[name] = df[name].astype('float')

    print(df.columns)

    return df


def get_wide_table_extract_over_time(date_start, date_stop, tagnames, table, min_per_h=None,
agg_time=None,
                                     avg=False, std_dev=False):
    """
    Connects to DB, gets data for given tagnames in the time range between date_start and date_stop with
    min_per_h minutes per hour in the range
    :param date_start: string, start time
```

```
    :param date_stop: string, stop time
    :param tagnames: list of strings, names of the tags
    :param min_per_h: int, number of minutes per hour
    :param table: string, name of table, either 'high_speed_big', 'high_speed_big'
    :return: matrix of data
    """
    # Split time frame up
    date_1, date_2 = dual_date_list_interval_per_unit(date_start, date_stop, min_per_h)

    if not avg and not std_dev:
        data = select_high_hyg(date_1[0], date_2[0], tagnames, table=table, numpy_arr=False)
        for j in range(1, len(date_1)):
            print(date_1[j])
            rows = select_high_hyg(date_1[j], date_2[j], tagnames, table=table, numpy_arr=False)
            for row in rows:
                data.append(row)

    else:
        if std_dev:
            agg = 'stddev_samp'
        elif avg:
            agg = 'avg'

        data = select_high_hyg_agg(date_1[0], date_2[0], agg_time, tagnames, table=table, numpy_arr=False,
                                   aggregation=agg)
        for j in range(1, len(date_1)):
            print(date_1[j])
            rows = select_high_hyg_agg(date_1[j], date_2[j], agg_time, tagnames, table=table, numpy_arr=False,
                                       aggregation=agg)
            for row in rows:
                data.append(row)

    data = np.array(data, dtype=object)

    # Convert to pandas dataframe
    df = pd.DataFrame(data)

    df.columns = tagnames
    df['meas_time'] = pd.DatetimeIndex(df['meas_time'])

    # Set datetime datatype
    df['meas_time'] = df['meas_time'].astype('datetime64[ns]')
    # Set float datatype for numeric values
    for name in tagnames[1:]:
        df[name] = df[name].astype('float')

    return df


def get_df(date_start, date_stop, tagnames, table, min_per_h=None, agg_time=None, avg=False,
std_dev=False):
    """
    Connects to DB, gets data for given tagnames in the time range between date_start and date_stop with
    min_per_h minutes per hour in the range
    formats data to fit for pca
```

```
    executes pca
    :param date_start: string, start time
    :param date_stop: string, stop time
    :param tagnames: list of strings, names of the tags
    :param min_per_h: int, number of minutes per hour
    :param table: string, name of table, either 'measurement', 'vibration', 'high_speed_big', 'high_speed_big'
    :return:
    """
    state = None
    on_sig = None
    tagnames_ = tagnames.copy()
    if "HYG_PU19_State" in tagnames:
        state = ["HYG_PU19_State"]
        tagnames_.remove("HYG_PU19_State")
        print('state defined')

    if "HYG_PU19" in tagnames_:
        on_sig = ["HYG_PU19"]
        tagnames_.remove("HYG_PU19")

    write = True
    # Narrow tables
    if table == 'measurement' or table == 'vibration':
        if os.path.exists('./' + 'raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' + str(
                std_dev) + 'min_per_h=' +
                    str(min_per_h) + table + ' ' + str(date_start)[:10] + ' to ' + str(date_stop)[:10] + '.csv'):
            try:
                df = pd.read_csv('./' + 'raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' +
                                 str(std_dev) + 'min_per_h=' + str(min_per_h) + table + ' ' + str(date_start)[:10] +
                                 ' to ' + str(date_stop)[:10] + '.csv',
                                 usecols=['meas_time', 'PU19_PW_PV', 'PU19_TQ_PV', 'PU19_MO', 'PU19_SF_PV', 'FT02',
                                          'PT15',
                                          'PT16', 'PU19_V_L', 'PU19_V_I', 'PU19_V_O', 'PU19_P_L', 'PU19_P_I',
                                          'PU19_P_O'])
            except ValueError as e:
                df = pd.read_csv('./' + 'raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' +
                                 str(std_dev) + 'min_per_h=' + str(min_per_h) + table + ' ' + str(date_start)[:10] +
                                 ' to ' + str(date_stop)[:10] + '.csv',
                                 usecols=['meas_time', 'PU19_PW_PV', 'PU19_TQ_PV', 'PU19_MO', 'PU19_SF_PV', 'FT02',
                                          'PT15', 'PT16'])

            df['meas_time'] = df['meas_time'].apply(lambda x: x[0:18])
            df['meas_time'] = pd.to_datetime(df['meas_time'])
            # Sort df2
            df['meas_time'] = pd.DatetimeIndex(df['meas_time'])

            write = False
        else:
            df = get_narrow_table_extract_over_time(date_start, date_stop, tagnames_, table,
                                                    min_per_h=min_per_h, agg_time=agg_time, avg=avg, std_dev=std_dev)

    # Wide tables
    elif table == 'high_speed_big' or table == 'high_speed_big_reduced':
        if os.path.exists('./' + 'raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' + str(
                std_dev) + 'min_per_h=' +
```

```
                str(min_per_h) + table + ' ' + str(date_start)[:10] + ' to ' + str(date_stop)[:10] + '.csv'):
        df = pd.read_csv('./' + 'raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' + str(
            std_dev) + 'min_per_h=' +
                 str(min_per_h) + table + ' ' + str(date_start)[:10] + ' to ' + str(date_stop)[
                                                                                :10] + '.csv',
                 usecols=tagnames_)

        df['meas_time'] = df['meas_time'].apply(lambda x: x[0:18])
        df['meas_time'] = pd.to_datetime(df['meas_time'])
        # Sort df2
        df['meas_time'] = pd.DatetimeIndex(df['meas_time'])

        write = False
      else:
        df = get_wide_table_extract_over_time(date_start, date_stop, tagnames_, min_per_h=min_per_h,
table=table,
                                              agg_time=agg_time, avg=avg, std_dev=std_dev)

    if write:
      df.to_csv('raw ' + 'agg_time=' + str(agg_time) + 'avg=' + str(avg) + 'std_dev=' + str(std_dev) + 'min_per_h='
+
           str(min_per_h) + table + ' ' + str(date_start)[:10] + ' to ' + str(date_stop)[:10] + '.csv')
    print(on_sig)
    if on_sig != None:
      if os.path.exists('./HYG_PU19 ' + str(date_start)[:10] + ' to ' + str(date_stop[:10]) + '.csv'):
        df2 = pd.read_csv('./HYG_PU19 ' + str(date_start)[:10] + ' to ' + str(date_stop[:10]) + '.csv',
                 usecols=['meas_time', 'PU19'])

        df2['meas_time'] = df2['meas_time']  # .apply(lambda x: x[0:18])
        # df2['meas_time'] = pd.to_datetime(df2['meas_time'])
        df2['meas_time'] = df2['meas_time'].astype('datetime64[ns]')
        # Sort df22
        df2['meas_time'] = pd.DatetimeIndex(df2['meas_time'])



      else:
        df2 = get_narrow_table_extract_over_time(date_start, date_stop, on_sig, table='measurement',
min_per_h=60)

        df2.to_csv('HYG_PU19 ' + str(date_start)[:10] + ' to ' + str(date_stop)[:10] + '.csv')

      df = merge_df(df, df2)

    if state != None:
      print('Get state values')
      # df2 = get_narrow_table_extract_over_time(date_start, date_stop, state, table='measurement',
min_per_h=60)
      df2 = pd.read_csv('HYG_PU19 Status.csv', usecols=['meas_time', 'PU19_State'])

      df2['meas_time'] = df2['meas_time'].apply(lambda x: x[0:18])
      df2['meas_time'] = pd.to_datetime(df2['meas_time'])
      # Sort df2
      df2['meas_time'] = pd.DatetimeIndex(df2['meas_time'])
      df2.sort_index(axis=1, ascending=True)
```

```
    df = merge_df(df, df2)

# Define parts of tagnames which are on right hand side axis of line plot

df['meas_time'] = pd.DatetimeIndex(df['meas_time'])
for x in df.columns[1:]:
    df[x] = df[x].astype('float')
train, test = train_test_split(df.copy(deep=True), test_size=0.3, random_state=42)

df = None

df = train.copy(deep=True)

df.dropna()

return df, test
```

Appendix D Date list manipulation

```
import datetime
import pandas as pd


def date_list(date_start, date_stop, interval, ms=True, second=False, min=False,
              hour=False, day=False):
    """
    Creates a list of datetime objects from date_start to date_stop with interval defined by optional flags and
    interval, ms is standard

    :param date_start: string, Time start
    :param date_stop: string, time stop
    :param interval: int, time of interval in ms to create date list of
    :param ms: bool, if only this is true, give list of ms interval
    :param second: bool, if only this is true, give list of ms interval
    :param min: bool, if only this is true, give list of ms interval
    :param hour: bool, if only this is true, give list of ms interval
    :param day: bool, if only this is true, give list of ms interval
    :return:
    """
    if type(date_start) is not datetime.datetime or not pd.Timestamp:

        try:
            date_stop = datetime.datetime.strptime(date_stop, '%Y-%m-%d')
        except ValueError as e:
            try:
                date_stop = datetime.datetime.strptime(date_stop, '%Y-%m-%d %H:%M:%S')
            except ValueError as e:
                print(e)
                date_stop = datetime.datetime.strptime(date_stop, '%Y-%m-%d %H:%M:%S.%f')
        try:
            date_start = datetime.datetime.strptime(date_start, '%Y-%m-%d')

        except ValueError as e:
            try:
                date_start = datetime.datetime.strptime(date_start, '%Y-%m-%d %H:%M:%S')

            except ValueError as e:
                print(e)
                date_start = datetime.datetime.strptime(date_start, '%Y-%m-%d %H:%M:%S.%f')


    list_ms = []
    list_ms.append(date_start)

    # Makes list of days interval
    if day:
        while list_ms[-1] < date_stop:
            list_ms.append(list_ms[-1] + datetime.timedelta(days=interval))

    # Makes list of hours interval
    elif hour:
        while list_ms[-1] < date_stop:
```

```
            list_ms.append(list_ms[-1] + datetime.timedelta(hours=interval))

    # Makes list of minutes interval
    elif min:
        while list_ms[-1] < date_stop:
            list_ms.append(list_ms[-1] + datetime.timedelta(minutes=interval))

    # makes list of seconds interval
    elif second:
        while list_ms[-1] < date_stop:
            list_ms.append(list_ms[-1] + datetime.timedelta(seconds=interval))

    # Makes lists of ms intervals
    elif ms:
        while list_ms[-1] < date_stop:
            list_ms.append(list_ms[-1] + datetime.timedelta(milliseconds=interval))

    return list_ms

def dual_date_list_interval_per_unit(date_start, date_stop, time_per_unit):
    """
    makes two lists of datetimes with a difference of time per unit
    :param date_start: string, start time
    :param date_stop: string, stop time
    :param time_per_unit: int, number of minutes per hour
    :return: date_1 is a list of intervals with 1h from initial time to initial end,
    date_2 is a list of intervals with 1h from initial time + time_per_unit
    """
    date_start = str(date_start)
    date_stop = str(date_stop)

    date_1 = date_list(date_start, date_stop, interval=1, hour=True)

    try:
        date_2 = date_list(str(datetime.datetime.strptime(date_start, '%Y-%m-%d %H:%M:%S') +
                               datetime.timedelta(minutes=time_per_unit)),
                           str(datetime.datetime.strptime(date_stop, '%Y-%m-%d %H:%M:%S') +
                               datetime.timedelta(minutes=time_per_unit)),
                           interval=1, hour=True)
    except ValueError:
        try:
            date_2 = date_list(str(datetime.datetime.strptime(date_start, '%Y-%m-%d') +
                                   datetime.timedelta(minutes=time_per_unit)),
                               str(datetime.datetime.strptime(date_stop, '%Y-%m-%d') +
                                   datetime.timedelta(minutes=time_per_unit)),
                               interval=1, hour=True)
        except ValueError:
            date_2 = date_list(str(datetime.datetime.strptime(date_start, '%Y-%m-%d %H:%M:%S.%f') +
                                   datetime.timedelta(minutes=time_per_unit)),
                               str(datetime.datetime.strptime(date_stop, '%Y-%m-%d %H:%M:%S.%f') +
                                   datetime.timedelta(minutes=time_per_unit)),
                               interval=1, hour=True)
```

```
except TypeError:
   date_2 = date_list((date_start +
                       datetime.timedelta(minutes=time_per_unit)),
                      (date_stop +
                       datetime.timedelta(minutes=time_per_unit)),
                      interval=1, hour=True)

date_1 = date_1[:-1]
date_2 = date_2[:-1]
return date_1, date_2
```

Appendix E Plot histogram, scatter matrix plot, PCA

```
import math
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px
import plotly
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

def create_colors():
    """
    Changes the color scheme of plotly

    :return: object of colors
    """
    color1 = plotly.colors.qualitative.Light24
    color2 = plotly.colors.qualitative.Dark24
    color3 = plotly.colors.qualitative.Alphabet

    colors = []
    for i in range(len(color1)):
        colors.append(color1[i])
    for i in range(len(color2)):
        colors.append(color2[i])
    for i in range(len(color3)):
        colors.append(color3[i])
    return colors

def remove_tags_from_taglist(calculation_suffixes, tagname):
    """
    Removes tags in calculation suffixes from the tagname list

    :param calculation_suffixes: List of string, entries in tagname which consist of any of these will not be part of
output
    :param tagname: List of strings, tagnames which should be filtered
    :return: list of filtered strings
    """
    num_label_included = 0
    result = ''
    for k in range(len(calculation_suffixes)):
        if calculation_suffixes[k] in tagname:
            pass
        else:
            num_label_included += 1

        if num_label_included == len(calculation_suffixes):
            unit_index = tagname.find("[")
            label_tag = tagname[:unit_index]
            result = label_tag

    if result != '':
        return result
```

```
def plot(filename, tagname, data, right_hand_axis=None, single_time=False):
    """
    Plots data with tagnames and stores in filename.html

    :param filename: name of html file output
    :param tagname: List of tagnames to be plotted
    :param data: Matrix of data to be plotted
    :param right_hand_axis: List of tags which should be plotted on right hand y-axis, can contain part of the
names.
    e.g. to plot all tags from HYG insert 'HYG' into list
    :return: No return
    """

    """Creates a subplot for the second hand y-axis"""
    fig = make_subplots(specs=[[{"secondary_y": True}]])
    plotted = False
    first_axis = ''
    second_axis = ''

    colors = create_colors()

    calculation_suffixes = ['_MA' '_LPF', '_sum', '_RoT'] #Used to remove suffixes for the axis labels
    tag_length = len(tagname)

    for i in range(tag_length):
        print(i)
        plotted = False          #Variable to check if the tag has been plotted on the right hand axis
        """Sets up specified data on right hand y-axis"""
        if right_hand_axis is not None:
            for j in range(len(right_hand_axis)):
                if right_hand_axis[j] in tagname[i]:

                    if single_time == False:
                        #Adds the tagdata to the plot
                        fig.add_trace(go.Scatter(x=data[i * 2], y=data[i * 2 + 1], name=tagname[i],
                                      marker=dict(color=colors[i], size=15)), secondary_y=True)
                        plotted = True

                    elif single_time == True:
                        # Adds the tagdata to the plot
                        fig.add_trace(go.Scatter(x=data[0], y=data[i+1], name=tagname[i],
                                      marker=dict(color=colors[i], size=15)), secondary_y=True)
                        plotted = True

                    """Removes unwanted tagnames from the right y-axis"""
                    if remove_tags_from_taglist(calculation_suffixes, tagname[i]) != None:
                        second_axis += remove_tags_from_taglist(calculation_suffixes, tagname[i]) + ", "
                        print("RHA")


        """If the data is not specified to be on right hand y-axis, put on left hand y-axis"""
        if plotted is False:
            if not single_time:
                # Adds the tagdata to the plot
```

```
            fig.add_trace(go.Scatter(x=data[i * 2], y=data[i * 2 + 1],
                                     name=tagname[i],
                                    marker=dict(color=colors[i],
                                    size=15)),
                                    secondary_y=False)

        elif single_time:
            # Adds the tagdata to the plot
            fig.add_trace(go.Scatter(x=data[0], y=data[i+1],
                                     name=tagname[i],
                                     marker=dict(color=colors[i],
                                    size=15)),
                                    secondary_y=False)


        """Removes unwanted tagnames from the left y-axis"""
        if remove_tags_from_taglist(calculation_suffixes, tagname[i]) != None:
            first_axis += remove_tags_from_taglist(calculation_suffixes, tagname[i]) + ", "
            print("LHA")

    """Removes the last space and comma from the axis text"""
    first_axis = first_axis[:-2]
    second_axis = second_axis[:-2]
    """Updates axes"""
    fig.update_xaxes(title_text='Tid')
    fig.update_yaxes(title_text=first_axis, secondary_y=False)
    fig.update_yaxes(title_text=second_axis, secondary_y=True)

    """Creates a html file of the plot"""
    plotly.offline.plot(fig, filename=filename+'.html')

def histogram_ly(df, filename):

    size = len(df.columns)
    rows = math.ceil(math.sqrt(size))
    col = rows
    #print(rows, col, size)
    fig = make_subplots(rows,col)
    trace = []
    j = 1
    k = 1
    for i in range(size):
        trace.append(go.Histogram(x=df[df.columns[i]], nbinsx=500, name=df.columns[i]))
        print(j, k)
        fig.append_trace(trace[i], j, k)
        #Move along columns, if end at row, increment row reset column
        k += 1
        if k == col+1:
            j += 1
            k = 1

    plotly.offline.plot(fig, filename='histogram ' + filename + '.html')


def correlation_plots(df, filename):
```

```
    recolor = True
    stats = df.describe()
    stats.to_csv('./Statistics value ' + filename + '.csv', sep=',')


    corr_matrix = df.corr()
    corr_matrix.to_csv('./Correlation matrix ' + filename + '.csv', sep=',')

    print(stats)
    if recolor:
        print('State plot')
        fig = px.scatter_matrix(df, dimensions=df.columns[:-1], color="PU19_State")
    else:
        fig = px.scatter_matrix(df[1:])

    plotly.offline.plot(fig, filename='Correlation ' + filename + '.html')


def pca(df, no_component, filename):

    print(df.columns)
    recolor = False
    timeIndex = pd.to_datetime(df['meas_time']).astype(np.int64)
    if "PU19_State" in df.columns:
        colorIndex = df["PU19_State"]
        df.drop(columns=["PU19_State"], axis=1, inplace=True)
        recolor = True


    features = df.columns[1:]

    scaler = StandardScaler()
    df[features] = scaler.fit_transform(df[features])

    pca = PCA(n_components=no_component)
    components = pca.fit_transform(df[features])
    labels = {
        str(i): f"PC {i + 1} ({var:.1f}%)"
        for i, var in enumerate(pca.explained_variance_ratio_ * 100)
    }

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

    total_var = pca.explained_variance_ratio_.sum() * 100

    if recolor:
        fig = px.scatter_3d(
            components, x=0, y=1, z=2, color=colorIndex,
            title=f'Total Explained Variance: {total_var:.2f}%',
            labels=labels
        )

        plotly.offline.plot(fig, filename='PCA score ' + filename + '.html')
        fig2 = px.scatter(
            components, x=0, y=1, color=colorIndex,
```

```
        title=f'PC1, PC2 Total Explained Variance: {total_var:.2f}%',
        labels=labels
    )

    plotly.offline.plot(fig2, filename='PCA score PC1 PC2' + filename + '.html')

    fig3 = px.scatter(
        components, x=0, y=2, color=colorIndex,
        title=f'PC1, PC3 Total Explained Variance: {total_var:.2f}%',
        labels=labels
    )

    plotly.offline.plot(fig3, filename='PCA score PC1 PC3' + filename + '.html')
    try:
        fig4 = px.scatter(
            components, x=0, y=3, color=colorIndex,
            title=f'PC1, PC3 Total Explained Variance: {total_var:.2f}%',
            labels=labels
        )

        plotly.offline.plot(fig4, filename='PCA score PC1 PC4' + filename + '.html')
    except:
        pass


else:
    fig = px.scatter_3d(
        components, x=0, y=1, z=2, color=timeIndex,
        title=f'Total Explained Variance: {total_var:.2f}%',
        labels=labels
    )

    plotly.offline.plot(fig, filename='PCA score ' + filename + '.html')

fig5 = px.scatter()
for i, feature in enumerate(features):
    fig5.add_shape(
        type='line',
        x0=0, y0=0,
        x1=loadings[i, 0],
        y1=loadings[i, 1],

    )
    fig5.add_annotation(
        x=loadings[i, 0],
        y=loadings[i, 1],

        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",

        text=feature
    )
plotly.offline.plot(fig5, filename='PCA loading PC1 PC2' + filename + '.html')
```

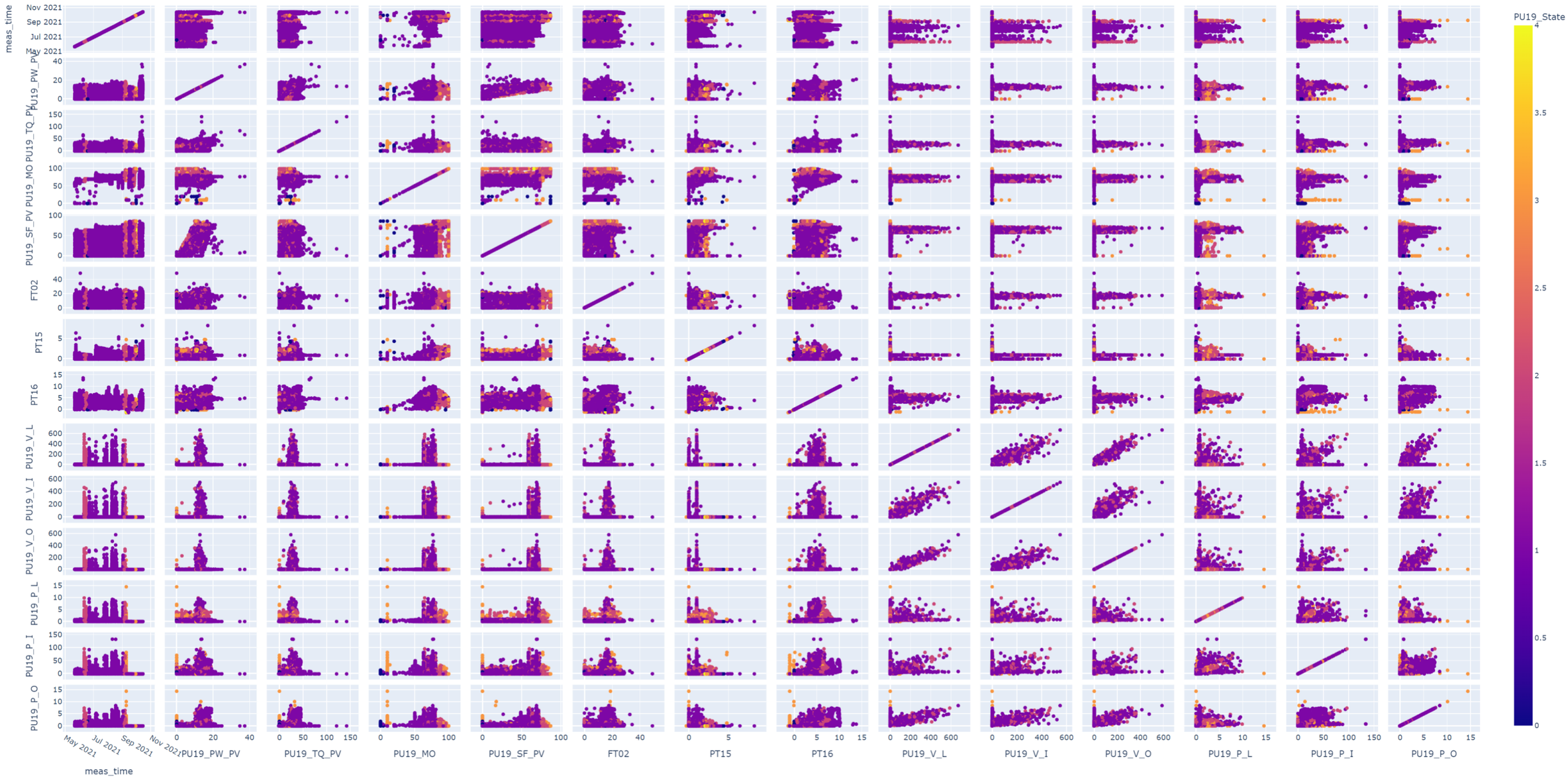
```
fig6 = px.scatter()
for i, feature in enumerate(features):
    fig6.add_shape(
        type='line',
        x0=0, y0=0,
        x1=loadings[i, 0],
        y1=loadings[i, 2],

    )
    fig6.add_annotation(
        x=loadings[i, 0],
        y=loadings[i, 2],

        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",

        text=feature
    )
plotly.offline.plot(fig6, filename='PCA loading PC1 PC3' + filename + '.html')

try:
    fig7 = px.scatter()
    for i, feature in enumerate(features):
        fig7.add_shape(
            type='line',
            x0=0, y0=0,
            x1=loadings[i, 0],
            y1=loadings[i, 3],

        )
        fig7.add_annotation(
            x=loadings[i, 0],
            y=loadings[i, 3],

            ax=0, ay=0,
            xanchor="center",
            yanchor="bottom",

            text=feature
        )
    plotly.offline.plot(fig7, filename='PCA loading PC1 PC4' + filename + '.html')

except:
    pass
```

Appendix F Call to get data, wrangling and plotting

```
from main import *

if __name__ == '__main__':

    # 1s data
    vib = False
    vib = True
    if vib:
        tagnames = ["HYG_PU19_PW_PV", "HYG_PU19_TQ_PV", "HYG_PU19_MO", "HYG_PU19_SF_PV",
"HYG_FT02", "HYG_PT15",
                "HYG_PT16",
                "HYG_PU19_V_L", "HYG_PU19_V_I", "HYG_PU19_V_O", "HYG_PU19_P_L", "HYG_PU19_P_I",
"HYG_PU19_P_O",
                "HYG_PU19_State"]

        date_start = '2021-05-01'
        date_stop = '2021-09-09'
        table = 'vibration'
        std_dev = False
        min_per_h = 10
        avg = False
        agg_time = 1/3

        df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                    min_per_h=min_per_h, avg=avg, agg_time=agg_time)


        histogram_ly(df, '1' + date_start + ' to ' + date_stop + table +
                ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                ' agg_time=' + str(agg_time))
        correlation_plots(df, '1' + date_start + ' to ' + date_stop + table +
                    ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                    ' agg_time=' + str(agg_time))
        pca(df, 4, date_start + ' to ' + date_stop + table +
            ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
            ' agg_time=' + str(agg_time*60))


    # 50ms data
    hsb = False
    hsb = True
    if hsb:
        tagnames = ['meas_time', 'pu19_pw_pv', 'pu19_tq_pv', 'pu19_sf_pv', 'pu19_mo', 'pt15', 'pt16', 'ft02',
                "HYG_PU19_State"]

        date_start = '2021-07-01'
        date_stop = '2021-09-09'
        table = 'high_speed_big_reduced'
        std_dev = False
        min_per_h = 1
        avg = False
        agg_time = 1
```

```
    df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                      min_per_h=min_per_h, avg=avg, agg_time=agg_time)

    histogram_ly(df, date_start + ' to ' + date_stop + table +
                 ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                 ' agg_time=' + str(agg_time))
    correlation_plots(df, date_start + ' to ' + date_stop + table +
                      ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                      ' agg_time=' + str(agg_time))
    pca(df, 4, date_start + ' to ' + date_stop + table +
        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
        ' agg_time=' + str(agg_time))
  # 30sec
  meas = False
  meas = True
  if meas:
    tagnames = ["HYG_PU19_PW_PV", "HYG_PU19_TQ_PV", "HYG_PU19_MO", "HYG_PU19_SF_PV",
"HYG_FT02", "HYG_PT15", "HYG_PT16",
                "HYG_PU19_State"]

    date_start = '2020-05-01'
    date_stop = '2021-09-09'
    table= 'measurement'
    std_dev = False
    min_per_h = 60
    avg = False
    agg_time = 10

    df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                      min_per_h=min_per_h, avg=avg, agg_time=agg_time)


    histogram_ly(df, date_start + ' to ' + date_stop + table +
                 ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                 ' agg_time=' + str(agg_time))
    correlation_plots(df, date_start + ' to ' + date_stop + table +
                 ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                 ' agg_time=' + str(agg_time))
    pca(df, 3, date_start + ' to ' + date_stop + table +
                 ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                 ' agg_time=' + str(agg_time))
```

Appendix G Increased size of Figure 2-13



Appendix H Call for time-series plots

```
from postgres import select_high_hyg, Tag
from plots import *
import datetime
import csv


def lag_matrise_tags(tagnames, date_start, date_stop, Raw=False, table='measurement',
one_time_axis=False):
    """
    Creates a matrix of raw data and/or calculated values, such as moving average, derivative, integration,
filtering

    :param tagnames: list of strings, Tagnames from the process which should be put into a matrix for plotting
    :param date_start: string or datetime
    :param date_stop: string or datetime
    :param Raw: bool, raw data will be part of output
    :param table: string, name of table
    :return: list of lists, data as described in option on form [[time], [values], [time], [values]...], list of strings
    name of value lists
    """

    """Initializing lists"""
    matrix = []
    tags = []
    new_tagnames = []

    """Iterates through tags in the taglist"""
    for i in range(len(tagnames)):
        """Gets the tag data from database"""
        print(tagnames[i])
        tags.append(Tag(tagnames[i]))
        tags[i].get_measurement(date_start, date_stop, table=table)

        """Adding raw, or calculated values to matrix for plotting, also creates a tagname list depending on which
              values are added"""
        if not one_time_axis:
            # Raw val
            if Raw == True:
                new_tagnames.append(tagnames[i] + '[' + tags[i].unit_type + '] ' + tags[i].tag_desc)
                """Takes the tag timestamp and data and appends it to the data matrix"""
                matrix.append(tags[i].timestamp)
                matrix.append(tags[i].measurements)

        tags[i].sql.disconnect()

    return matrix, new_tagnames

def vibrasjon_freq():
    date_start = input('When to start? YYYY-MM-DD HH:MM:SS')
    date_stop = input('When to stop?')
    names, data = select_high_hyg(date_start, date_stop)
    freq_arr = []
```

```
    stats = []

    for j in range(1, len(data)):
        temp_time = []
        for i in range(len(data[j]) - 1):
            if data[j][i] != data[j][i + 1]:
                temp_time.append(data[0][i + 1])
        temp2_time = []

        for i in range(len(temp_time) - 1):
            temp2_time.append(temp_time[i + 1] - temp_time[i])
        print(temp2_time)
        freq_arr.append(temp2_time)
        try:
            average_timedelta = sum(temp2_time, datetime.timedelta(0)) / len(temp2_time)
            min1 = min(temp2_time)
            max1 = max(temp2_time)
        except ZeroDivisionError as e:
            print(e)
            average_timedelta = 0
            min1 = 0
            max1 = 0
        stats.append([names[j], average_timedelta, min1, max1])

    # print(freq_arr)
    freq_arr = transpose_undefined_matrix(freq_arr)
    ArrayTocsv(['ft', 'pt15', 'pt16', 'mo', 'pw', 'tq', 'sf'], freq_arr, 'measurement frequencies.csv')

    ArrayTocsv([' ', 'avg', 'min', 'max'], stats, 'freq stats.csv')


def vibration_high_wide(date_start=None, date_stop=None, tags=None):
    if date_stop == None:
        # Start and stopping point
        date_start = input('When to start? YYYY-MM-DD HH:MM:SS')
        date_stop = input('When to stop?')
    # list of tags
    if tags == None:
        tags = ['meas_time', 'pu19_pw_pv', 'pu19_tq_pv']
    data = select_high_hyg(date_start, date_stop, tagname=tags)
    length = len(tags)
    for x in tags:
        print(x)
    data2 = transpose_undefined_matrix(data)
    ArrayTocsv(tags, data2, 'HYG high.csv')
    print(tags[1:length])
    tagnames = tags[1:length]
    for x in tagnames:
        print(x)
    rha = ['pt']
    plot("hsb høy ", tagnames, data, rha, single_time=True)


def ArrayTocsv(header_array,ArrayToWrite,file):
    """
```

```
    Writes a 2D array or list to csv file
    :param header_array: list, headers for the csv file
    :param ArrayToWrite: list of list or 2D array, data which should be written
    :param file: string, name of the file
    :return:
    """
    ArrayToWrite = zip(*ArrayToWrite)
    with open(file, 'w', newline='') as csvFile:

        writer = csv.writer(csvFile, delimiter=',')
        if header_array is not None:
            print('skriver overskrift')
            writer.writerow(header_array)
        print('Skriver verdier')
        for row in ArrayToWrite:
            writer.writerow(row)

    csvFile.close()

def find_longest_subarray(array):
    """
    Finds longest array in a matrix

    :param array: Matrix of any data
    :return: integer number of the longest subarray
    """
    max = 0
    for i in range(len(array)):
        if len(array[i]) > max:
            max = len(array[i])
    print(max)
    return max

def transpose_undefined_matrix(matrix):
    """
    Transposes a matrix

    :param matrix: Matrix to be transposed
    :return: Transposed matrix
    """
    max = find_longest_subarray(matrix)
    data = []
    temp = []
    for i in range(max):
        temp.clear()
        for j in range(len(matrix)):

            try:
                temp.append(matrix[j][i])
            except IndexError:
                temp.append('')

        data.append(temp.copy())
    return data
```

```
if __name__ == '__main__':
    #Time-series
    vibration_high_wide()
    #Update rates for 10ms data
    vibrasjon_freq()
```

Appendix I Outlier Removal

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

features = ['HYG_PU19_PW_PV', 'HYG_PU19_SF_PV']
df = pd.read_csv('HYG_2021_08.csv', usecols=features, sep=';', encoding='ISO-8859-1')

X = df.values

nbrs = NearestNeighbors(n_neighbors = 2)
nbrs.fit(X)

distances, indexes = nbrs.kneighbors(X)

plt.scatter(df['HYG_PU19_PW_PV'], df['HYG_PU19_SF_PV'], color="b", s=65)
outlier_index = np.where(distances.mean(axis = 1) > 0.3)
outlier_values = df.iloc[outlier_index]
plt.scatter(outlier_values['HYG_PU19_PW_PV'], outlier_values['HYG_PU19_SF_PV'], color = "r", label = 'KNN
outliers')
plt.xlabel("HYG_PU19_PW_PV")
plt.ylabel("HYG_PU19_SF_PV")
plt.legend(loc='upper left')
plt.show()
```

Appendix J Statistics in Chapter 2.5.3

	PU19_PW_PV	PU19_TQ_PV	PU19_MO	PU19_SF_PV	FT02	PT15	PT16	PU19_V_L	PU19_V_I	PU19_V_O	PU19_P_L	PU19_P_I	PU19_P_O	PU19_State
count	990297	990297	990297	990297	990297	990297	990297	990297	990297	990297	990297	990297	990297	990297
mean	11.840	25.207	72.032	63.093	16.210	0.798	4.755	1.595	1.541	1.440	0.669	7.926	0.733	1.125
std	3.547	9.024	10.430	15.027	3.777	0.413	1.431	18.023	15.768	13.370	0.506	5.729	0.605	0.391
min	0.000	-11.400	0.000	0.000	0.000	-0.331	-7.506	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.250	10.150	20.000	69.917	63.830	16.260	0.940	4.199	0.778	0.880	0.898	0.525	5.641	0.536	1.000
0.500	12.350	26.000	74.614	67.402	17.018	0.991	4.816	1.120	1.177	1.167	0.633	6.831	0.627	1.000
0.750	14.130	31.700	77.019	69.277	17.491	1.014	5.585	1.728	1.496	1.464	0.767	9.026	0.820	1.000
max	45.650	176.100	93.462	81.924	35.562	8.163	18.857	2598.510	2830.470	2607.290	93.750	820.962	93.750	4.000

Appendix K IQR and removing samples where pump is off

```
import pandas as pd


def quantile_remove(df, quantile, except_col):
    """
    Removes quantile highest values
    :param df: dataframe to remove values from
    :param quantile: decimal number equal to a percentage e.g. 0.995 for vibration
    :param except_col: columns which should not be quantile clipped
    :return: dataframe ordered as in inserted_cols
    """
    cols = df.columns.to_list()
    use_cols = [x for x in cols if x not in except_col]
    tmp = df.loc[:, use_cols]

    df.drop(columns=use_cols, inplace=True)
    outliers = tmp.quantile(quantile, numeric_only=True)

    filt_df = tmp.apply(lambda x: x[(x < outliers.loc[x.name])], axis=0)
    filt_df.to_csv('filt_df.csv')

    for x in except_col:
        try:
            filt_df = pd.concat([df.loc[:, [x]], filt_df], axis=1)

        except KeyError as e:
            print(e)

    filt_df = filt_df.fillna(method='ffill')

    inserted_cols = [ 'meas_time', 'PU19_PW_PV', 'PU19_TQ_PV',
      'PU19_MO', 'PU19_SF_PV', 'FT02', 'PT15', 'PT16', 'PU19_V_L', 'PU19_V_I',
      'PU19_V_O', 'PU19_P_L', 'PU19_P_I', 'PU19_P_O', 'PU19', 'PU19_State']

    cols = ([col for col in inserted_cols if col in filt_df]
            + [col for col in filt_df if col not in inserted_cols])
    filt_df = filt_df[cols]

    return filt_df


def remove_on_off(df):
    df = df[df['PU19'] != 0]
    df = df.drop('PU19', 1)
    return df
```

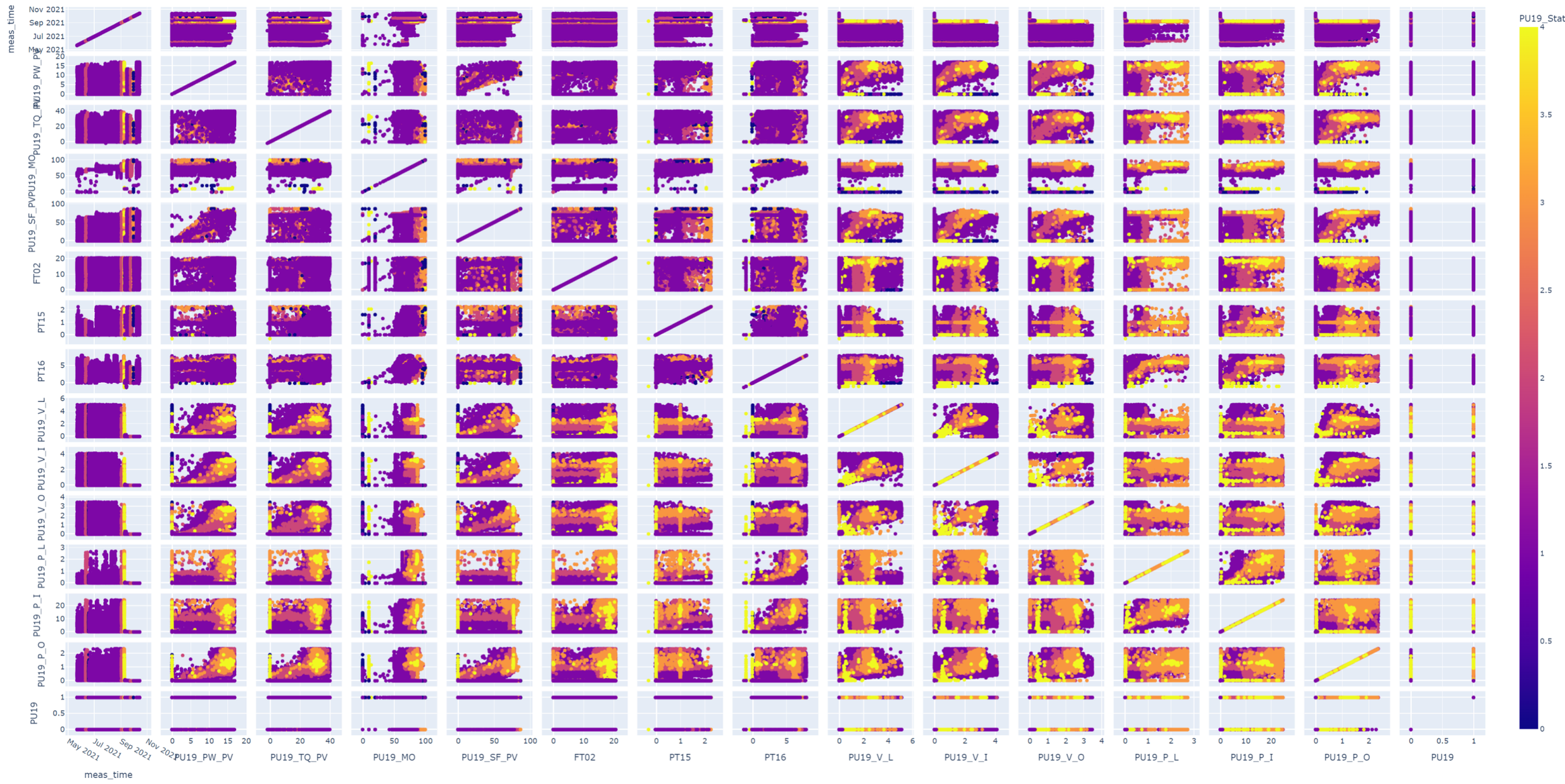
Appendix L Call to get, wrangle and plot average data

```
from main import *
from plots import *
from remove_samples import quantile_remove, remove_on_off


if __name__ == '__main__':

    # 1s data
    vib = False
    vib = True
    if vib:
        tagnames = ["HYG_PU19_PW_PV", "HYG_PU19_TQ_PV", "HYG_PU19_MO", "HYG_PU19_SF_PV",
"HYG_FT02", "HYG_PT15",
                "HYG_PT16",
                "HYG_PU19_V_L", "HYG_PU19_V_I", "HYG_PU19_V_O", "HYG_PU19_P_L", "HYG_PU19_P_I",
"HYG_PU19_P_O",
                "HYG_PU19_State", "HYG_PU19"]

        date_start = '2021-05-01'
        date_stop = '2021-10-15'
        table = 'vibration'
        std_dev = False
        min_per_h = 10
        avg = True
        agg_time = 1/3

        df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                  min_per_h=min_per_h, avg=avg, agg_time=agg_time)


        df = quantile_remove(df, 0.995, ['meas_time', 'PU19_State', 'PU19', "HYG_PU19_PW_PV",
"HYG_PU19_TQ_PV",
                                 "HYG_PU19_MO", "HYG_PU19_SF_PV", "HYG_FT02", "HYG_PT15", "HYG_PT16"])
        histogram_ly(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
                ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                ' agg_time=' + str(agg_time))
        correlation_plots(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
                  ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                  ' agg_time=' + str(agg_time))

        df = remove_on_off(df)

        histogram_ly(df, '1' + date_start + ' to ' + date_stop + table +
                ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                ' agg_time=' + str(agg_time))
        correlation_plots(df, '1' + date_start + ' to ' + date_stop + table +
                  ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                  ' agg_time=' + str(agg_time))
        pca(df, 4, date_start + ' to ' + date_stop + table +
          ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
          ' agg_time=' + str(agg_time*60))
```

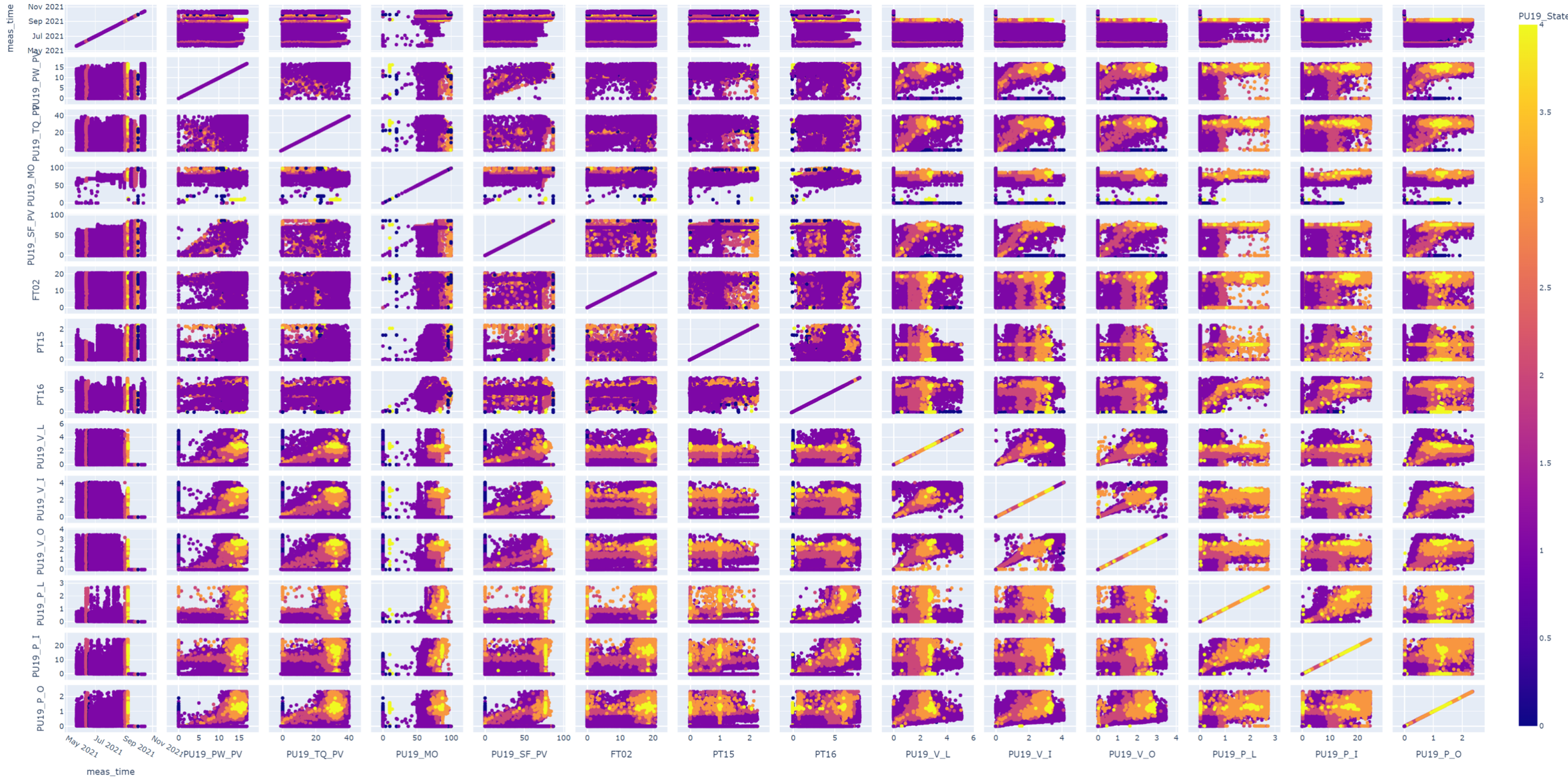
```
    # 50ms data
    hsb = False
    hsb = True
    if hsb:
        tagnames = ['meas_time', 'pu19_pw_pv', 'pu19_tq_pv', 'pu19_sf_pv', 'pu19_mo', 'pt15', 'pt16', 'ft02',
                "HYG_PU19_State", "HYG_PU19"]

        date_start = '2021-07-01'
        date_stop = '2021-10-15'
        table = 'high_speed_big_reduced'
        std_dev = False
        min_per_h = 5
        avg = True
        agg_time = 1

        df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                          min_per_h=min_per_h, avg=avg, agg_time=agg_time)
        histogram_ly(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
                ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                ' agg_time=' + str(agg_time))
        correlation_plots(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
                        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                        ' agg_time=' + str(agg_time))
        df = remove_on_off(df)

        histogram_ly(df, date_start + ' to ' + date_stop + table +
                ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                ' agg_time=' + str(agg_time))
        correlation_plots(df, date_start + ' to ' + date_stop + table +
                        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
                        ' agg_time=' + str(agg_time))
        pca(df, 4, date_start + ' to ' + date_stop + table +
            ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
            ' agg_time=' + str(agg_time))
    # 30sec
    meas = False
    meas = True
    if meas:
        tagnames = ["HYG_PU19_PW_PV", "HYG_PU19_TQ_PV", "HYG_PU19_MO", "HYG_PU19_SF_PV",
"HYG_FT02", "HYG_PT15", "HYG_PT16",
                "HYG_PU19_State", "HYG_PU19"
                # "HYG_PU19_V_L", "HYG_PU19_V_I", "HYG_PU19_V_O", "HYG_PU19_P_L", "HYG_PU19_P_I",
"HYG_PU19_P_O"
                ]
        date_start = '2020-05-01'
        date_stop = '2021-10-15'
        table= 'measurement'
        std_dev = False
        min_per_h = 60
        avg = True
        agg_time = 10

        df, test = get_df(date_start, date_stop, tagnames, table, std_dev=std_dev,
                          min_per_h=min_per_h, avg=avg, agg_time=agg_time)
        histogram_ly(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
```

```
        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
        ' agg_time=' + str(agg_time))
correlation_plots(df, 'before on_off removal' + date_start + ' to ' + date_stop + table +
            ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
            ' agg_time=' + str(agg_time))
df = remove_on_off(df)

histogram_ly(df, date_start + ' to ' + date_stop + table +
        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
        ' agg_time=' + str(agg_time))
correlation_plots(df, date_start + ' to ' + date_stop + table +
        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
        ' agg_time=' + str(agg_time))
pca(df, 3, date_start + ' to ' + date_stop + table +
        ' min_per_h=' + str(min_per_h) + ' std_dev=' + str(std_dev) + 'avg=' + str(avg) +
        ' agg_time=' + str(agg_time))
```

Appendix M Increased size of Figure 2-26



Appendix N Increased size of Figure 2-28



Appendix O Code for analysis and Fourier Transform plotting

```
import glob
import math
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import datetime
from scipy import signal
from scipy.fft import fftfreq
from scipy.signal.filter_design import normalize
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pickle

scale_features = True
plot_features = False
print_skew_kurt = False
perform_correlation_lag_test = False
plot_value_distribution = False
calculate_fft = True
plot_fft_heatmap = True
perform_pca = False
plot_3d_pca = False
plot_2d_pca = False

# Reading of CSV to Dataframe
extension = 'csv'
all_datafiles = [i for i in glob.glob('data/*.{}'.format(extension))]
df = pd.concat([pd.read_csv(f, sep=';', usecols= ['Time', 'HYG_FT02',
'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV', 'HYG_PU19_SF_PV',
'HYG_PU19_MO', 'HYG_PT16'], decimal='.', encoding="ISO-8859-1") for f in
all_datafiles ])
df2 = pd.concat([pd.read_csv('data/temp/HYG_2021_09.csv', sep=';', usecols=
['Time', 'HYG_FT02', 'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV',
'HYG_PU19_SF_PV', 'HYG_PU19_MO', 'HYG_PT16'], decimal='.', encoding="UTF-8")])

df = df.append(df2)
df['Time'] = pd.DatetimeIndex(df['Time'])
```

```
dfStatus = pd.concat([pd.read_csv('HYG_PU19Status.csv', sep=';', usecols=
['Time', 'HYG_PU19_DEV_ID', 'HYG_PU19_STATUS'], decimal='.', encoding="ISO-
8859-1")])
dfStatus['Time'] = pd.DatetimeIndex(dfStatus['Time'])

df = pd.merge_asof(df.sort_values('Time'), dfStatus.sort_values('Time'),
on='Time')

# Start and end time of work data
start = datetime.datetime(2021,4,1, tzinfo=datetime.timezone.utc)
end = datetime.datetime(2021,4,16, tzinfo=datetime.timezone.utc)

mask = (df['Time'] >= start) & (df['Time'] < end)
df = df.loc[mask]

features  = ['HYG_FT02', 'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV',
'HYG_PU19_SF_PV', 'HYG_PU19_MO', 'HYG_PT16']

stoppedMask = df['HYG_PU19_SF_PV'] <= 0.0
statusColumn = 'HYG_PU19_STATUS'
df.loc[stoppedMask, statusColumn] = 0

if(plot_features):
    # Plot features over time
    fig = px.line(df, x='Time', y=features)
    fig.show()

if(scale_features):
    # Standard scaling of feature variables
    scaler = StandardScaler()
    df[features] = scaler.fit_transform(df[features])

def crosscorr(seriesA, seriesB, lag=0):
    return seriesA.corr(seriesB.shift(lag))

if(perform_correlation_lag_test):
    lagCorrelation = [(i, crosscorr(df['HYG_PU19_SF_PV'], df['HYG_PU19_MO'],
lag=i)) for i in range(-10, 10)]

    fig = px.line(lagCorrelation, x=0, y=1)
    fig.show()

    print(df[features].corr(method='pearson'))

if(print_skew_kurt):
    print("Feature skewedness: ")
    print(df[features].skew())
```

```
    print("Feature kurtosis: ")
    print(df[features].kurtosis())

if(plot_value_distribution):

    fig = make_subplots(
        rows=math.ceil(len(features) / 3), cols=3,
        shared_xaxes=True,
        vertical_spacing=0.03
    )
    plotindex = 0
    for feature in features:
        fig.add_trace(
            go.Histogram(x=df[feature], histnorm='probability'),
            col=(plotindex % 3) + 1,
            row=math.floor(plotindex / 3) + 1
        )
        plotindex += 1
    fig.show()

if(calculate_fft):
    # FFT 1 hour segments (120 samples) every 10 min
    freqs, times, Sx = signal.spectrogram(df['HYG_PU19_TQ_PV'].values,
fs=(1/30), window='hanning',
                                          nperseg=120, noverlap=100,
                                          detrend=False, scaling='spectrum')

    # Time as int (for color indication in plot)
    timeIndex = pd.to_datetime(df['Time']).astype(np.int64)

    FFTtimes = (times + 1800.0) * 1e9
    FFTtimes = np.array(FFTtimes, dtype=np.int64)
    startTimeInt = timeIndex.values[0]
    FFTtimes = FFTtimes + startTimeInt
    newFFTtimes = []
    for i in range(len(times)):
        newFFTtimes.append(df['Time'].values[119 + (i*20)])
    newFFTtimes = pd.to_datetime(newFFTtimes, unit='ns')
    freqsStrings = []
    for freq in freqs:
        freqsStrings.append("Freq_" + str(freq))

    fftDf = pd.DataFrame(columns=freqsStrings)
    fftDf['Time'] = newFFTtimes
    fftDf[freqsStrings] = np.transpose(Sx)
    fftDf['Time'] = pd.to_datetime(fftDf['Time'],
unit='ns').dt.tz_localize('UTC')
```

```
    fftDf['Time'] = pd.DatetimeIndex(fftDf['Time'])

    df = pd.merge_asof(df.sort_values('Time'), fftDf.sort_values('Time'),
on='Time')
    df = df.fillna(method='bfill')

    if(plot_fft_heatmap):
        # FFT Heatmap plot
        trace = [go.Heatmap(
            x= newFFTtimes[2:20],
            y= freqs[2:20],
            z= np.sqrt(Sx[2:20]), #10*np.log10(Sx)
            zmin=0,
            zmax=1.2,
            colorscale='Jet',
            )]
        layout = go.Layout(
            title = 'Spectrogram FFT PU19 Torque',
            yaxis = dict(title = 'Frequency'), # x-axis label
            xaxis = dict(title = 'Time'), # y-axis label
            )
        fig = go.Figure(data=trace, layout=layout)
        fig.show()

        fig = px.line(df, x='Time', y=freqsStrings[:20])
        fig.show()

    if(scale_features):
        df[freqsStrings] = scaler.fit_transform(df[freqsStrings])

if(perform_pca):
    # PC Analysis
    pcaFeatures = features + freqsStrings
    pca = PCA(n_components=3)
    components = pca.fit_transform(df[pcaFeatures])
    labels = {
        str(i): f"PC {i+1} ({var:.1f}%)"
        for i, var in enumerate(pca.explained_variance_ratio_ * 100)
    }

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

    total_var = pca.explained_variance_ratio_.sum() * 100

if(plot_3d_pca):
    # 3D scatter plot of 3 PC
    fig = px.scatter_3d(
```

```
        components, x=0, y=1, z=2, color=df['HYG_PU19_STATUS'],
        title=f'Total Explained Variance: {total_var:.2f}%',
        labels=labels
    )
    fig.show()

if(plot_2d_pca):
    # 2D scatter plot PC1 and 2 with loadings
    fig = px.scatter(
        components, x=0, y=1, color=df['HYG_PU19_STATUS'],
        title=f'Total Explained Variance: {total_var:.2f}%',
        labels=labels
    )

    for i, feature in enumerate(pcaFeatures):
        fig.add_shape(
            type='line',
            x0=0, y0=0,
            x1=loadings[i, 0],
            y1=loadings[i, 1]
        )
        fig.add_annotation(
            x=loadings[i, 0],
            y=loadings[i, 1],
            ax=0, ay=0,
            xanchor="center",
            yanchor="bottom",
            text=feature,
        )

    fig.show()
```

Appendix P Code for training and testing Naïve Bayes and SVM models

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from scipy import signal
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pickle

nb_svm_selector = False

train_model = False
load_and_test_model = True

nb_filename = 'training_nb_svm_vib/nb_model.sav'
svm_filename = 'training_nb_svm_vib/svm_model.sav'
scaler_filename = 'training_nb_svm_vib/scaler.pkl'

# Reading of CSV to Dataframe
df = pd.concat([pd.read_csv('data/temp/1s_vibration.csv', sep=',', usecols=
['meas_time','PU19_PW_PV','PU19_TQ_PV','PU19_MO','PU19_SF_PV','FT02','PT15','P
T16','PU19_V_L','PU19_V_I','PU19_V_O','PU19_P_L','PU19_P_I','PU19_P_O'],
decimal='.', encoding="UTF-8")])
df['meas_time'] = df['meas_time'] + '.000000+00:00'
df['Time'] = pd.DatetimeIndex(df['meas_time'])
print(df.head())

dfStatus = pd.concat([pd.read_csv('HYG_PU19Status.csv', sep=';', usecols=
['Time', 'HYG_PU19_DEV_ID', 'HYG_PU19_STATUS'], decimal='.', encoding="ISO-
8859-1")])
dfStatus['Time'] = pd.DatetimeIndex(dfStatus['Time'])

df = pd.merge_asof(df.sort_values('Time'), dfStatus.sort_values('Time'),
on='Time')
print(df.head())

print(df['HYG_PU19_STATUS'].unique())
# Start and end time of work data
start = datetime.datetime(2021,5,1, tzinfo=datetime.timezone.utc)
end = datetime.datetime(2021,9,10, tzinfo=datetime.timezone.utc)

mask = (df['Time'] >= start) & (df['Time'] < end)
```

```
df = df.loc[mask]

features  =
['PU19_PW_PV','PU19_TQ_PV','PU19_MO','PU19_SF_PV','FT02','PT15','PT16','PU19_V
_L','PU19_V_I','PU19_V_O','PU19_P_L','PU19_P_I','PU19_P_O']

stoppedMask = df['PU19_SF_PV'] <= 0.0
statusColumn = 'HYG_PU19_STATUS'
df.loc[stoppedMask, statusColumn] = 0

print(df['HYG_PU19_STATUS'].unique())
# FFT 1 hour segments (120 samples) every 10 min
freqs, times, Sx = signal.spectrogram(df['PU19_TQ_PV'].values, fs=(1/20),
window='hanning',
                                      nperseg=180, noverlap=160,
                                      detrend=False, scaling='spectrum')

# Time as int (for color indication in plot)
timeIndex = pd.to_datetime(df['Time']).astype(np.int64)

FFTtimes = (times + 1800.0) * 1e9
FFTtimes = np.array(FFTtimes, dtype=np.int64)
startTimeInt = timeIndex.values[0]
FFTtimes = FFTtimes + startTimeInt
#newFFTtimes = pd.to_datetime(FFTtimes, unit='ns')
newFFTtimes = []
for i in range(len(times)):
    newFFTtimes.append(df['Time'].values[179 + (i*20)])
newFFTtimes = pd.to_datetime(newFFTtimes, unit='ns')
freqsStrings = []
for freq in freqs:
    freqsStrings.append("Freq_" + str(freq))

fftDf = pd.DataFrame(columns=freqsStrings)
fftDf['Time'] = newFFTtimes
fftDf[freqsStrings] = np.transpose(Sx)
fftDf['Time'] = pd.to_datetime(fftDf['Time'], unit='ns').dt.tz_localize('UTC')
fftDf['Time'] = pd.DatetimeIndex(fftDf['Time'])

df = pd.merge_asof(df.sort_values('Time'), fftDf.sort_values('Time'),
on='Time')
df = df.fillna(method='bfill')

uniquePumpIDs = df['HYG_PU19_DEV_ID'].unique()

print(len(df))
```

```
time_steps = 3*60

if(True):
    for pumpID in uniquePumpIDs:
        if len(df.loc[(df['HYG_PU19_DEV_ID'] == pumpID) &
(df['HYG_PU19_STATUS'] <= 1)]) >= time_steps:
            scaler = StandardScaler()
            scaler.fit(df.loc[(df['HYG_PU19_DEV_ID'] == pumpID) &
(df['HYG_PU19_STATUS'] <= 1), features + freqsStrings].head(time_steps))
            df.loc[df['HYG_PU19_DEV_ID'] == pumpID, features + freqsStrings] =
scaler.transform(df.loc[df['HYG_PU19_DEV_ID'] == pumpID, features +
freqsStrings])
        else:
            df = df.drop(df[(df['HYG_PU19_DEV_ID'] == pumpID)].index)

print(len(df))

X = df[features + freqsStrings].values
print(df['HYG_PU19_STATUS'].unique())
y = df['HYG_PU19_STATUS'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

if(train_model and not nb_svm_selector):
    nb_clf = GaussianNB()
    nb_clf.fit(X_train,y_train)
    pickle.dump(nb_clf, open(nb_filename, 'wb'))

if(load_and_test_model and not nb_svm_selector):
    loaded_nb_model = pickle.load(open(nb_filename, 'rb'))
    y_pred_nb = loaded_nb_model.predict(X_test)
    print("Accuracy Naive Bayes: " + str(accuracy_score(y_test, y_pred_nb)))
    nb_cm = confusion_matrix(y_test, y_pred_nb,
labels=loaded_nb_model.classes_)
    nb_cm_norm = nb_cm.astype('float') / nb_cm.sum(axis=1)[:, np.newaxis]
    nb_cm_acc = nb_cm_norm.diagonal()
    print("Accuracy Label 0 (Stopped): ", nb_cm_acc[0])
    print("Accuracy Label 1 (Normal run): ", nb_cm_acc[1])
    print("Accuracy Label 2 (< week from fail): ", nb_cm_acc[2])
    print("Accuracy Label 3 (< 24h from fail): ", nb_cm_acc[3])
    print("Accuracy Label 4 (< 1h from fail): ", nb_cm_acc[4])

    failmatrix1 = np.zeros((3,5))
    for i in range(len(nb_cm[0])):
        failmatrix1[0,i] = nb_cm[0,i]
        failmatrix1[1,i] = nb_cm[1,i]
        failmatrix1[2,i] = nb_cm[2,i] + nb_cm[3,i] + nb_cm[4,i]
```

```
    failmatrix2 = np.zeros((3,3))
    for i in range(3):
        failmatrix2[i,0] = failmatrix1[i,0]
        failmatrix2[i,1] = failmatrix1[i,1]
        failmatrix2[i,2] = failmatrix1[i,2] + failmatrix1[i,3] +
failmatrix1[i,4]

    nb_fail_norm = failmatrix2.astype('float') / failmatrix2.sum(axis=1)[:,
np.newaxis]
    nb_fail_acc = nb_fail_norm.diagonal()

    print("Fail mode accuracy: ", nb_fail_acc[2])
    nb_disp = ConfusionMatrixDisplay(confusion_matrix=nb_cm,
display_labels=loaded_nb_model.classes_)
    nb_disp.plot()
    plt.savefig("training_nb_svm_vib/cfsmatplot_nb.png")
    plt.show(block=True)

if(train_model and nb_svm_selector):
    svm_clf = svm.SVC()
    svm_clf.fit(X_train,y_train)
    pickle.dump(svm_clf, open(svm_filename, 'wb'))

if(load_and_test_model and nb_svm_selector):
    loaded_svm_model = pickle.load(open(svm_filename, 'rb'))
    y_pred_svm = loaded_svm_model.predict(X_test)
    print("Accuracy SVM: " + str(accuracy_score(y_test, y_pred_svm)))
    svm_cm = confusion_matrix(y_test, y_pred_svm,
labels=loaded_svm_model.classes_)
    svm_cm_norm = svm_cm.astype('float') / svm_cm.sum(axis=1)[:, np.newaxis]
    svm_cm_acc = svm_cm_norm.diagonal()
    print("Accuracy Label 0 (Stopped): ", svm_cm_acc[0])
    print("Accuracy Label 1 (Normal run): ", svm_cm_acc[1])
    print("Accuracy Label 2 (< week from fail): ", svm_cm_acc[2])
    print("Accuracy Label 3 (< 24h from fail): ", svm_cm_acc[3])
    print("Accuracy Label 4 (< 1h from fail): ", svm_cm_acc[4])

    failmatrix1 = np.zeros((3,5))
    for i in range(len(svm_cm[0])):
        failmatrix1[0,i] = svm_cm[0,i]
        failmatrix1[1,i] = svm_cm[1,i]
        failmatrix1[2,i] = svm_cm[2,i] + svm_cm[3,i] + svm_cm[4,i]

    failmatrix2 = np.zeros((3,3))
    for i in range(3):
        failmatrix2[i,0] = failmatrix1[i,0]
```

```
        failmatrix2[i,1] = failmatrix1[i,1]
        failmatrix2[i,2] = failmatrix1[i,2] + failmatrix1[i,3] +
failmatrix1[i,4]

    svm_fail_norm = failmatrix2.astype('float') / failmatrix2.sum(axis=1)[:,
np.newaxis]
    svm_fail_acc = svm_fail_norm.diagonal()

    print("Fail mode accuracy: ", svm_fail_acc[2])

    svm_disp = ConfusionMatrixDisplay(confusion_matrix=svm_cm,
display_labels=loaded_svm_model.classes_)
    svm_disp.plot()
    plt.savefig("training_nb_svm_vib/cfsmatplot_svm.png")
    plt.show(block=True)
```

Appendix Q Code for training and testing LSTM-models

```
import os
import glob
import numpy as np
import pandas as pd
from pylab import rcParams
from matplotlib import rc
import matplotlib.pyplot as plt
import datetime
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
import seaborn as sns

print(tf.__version__)

rcParams['figure.figsize'] = 16, 10

checkpoint_path = "training_PU19_individualscale/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
scaler_path = "training_PU19_individualscale/scaler.pkl"

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 16, 10

train_model = False
load_test_model = True

def one_hot_encode(sequence, n_unique=5):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_unique)]
        vector[value] = 1
        encoding.append(vector)
    return np.array(encoding)

def one_hot_decode(encoded_seq):
    return [np.argmax(vector) for vector in encoded_seq]

def create_dataset(Time, X, y, time_steps=1):
    Xs, ys = [], []

    skipped_series = 0
```

```
    for i in range(len(X) - time_steps):
        # Start and end time of work data
        start = pd.to_datetime(Time.iloc[i])
        end = pd.to_datetime(Time.iloc[i+time_steps])

        if end <= start + pd.DateOffset(minutes=61):
            v = X.iloc[i:(i + time_steps)].values
            Xs.append(v)
            ys.append(y.iloc[i + time_steps])
        else:
            skipped_series += 1
    print("Skipped series: ", skipped_series)
    return np.array(Xs), np.array(ys)

# Reading of CSV to Dataframe
extension = 'csv'
all_datafiles = [i for i in glob.glob('data/*.{}'.format(extension))]
df = pd.concat([pd.read_csv(f, sep=';', usecols= ['Time', 'HYG_FT02',
'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV', 'HYG_PU19_SF_PV',
'HYG_PU19_MO', 'HYG_PT16'], decimal='.', encoding="ISO-8859-1") for f in
all_datafiles ])
df2 = pd.concat([pd.read_csv('data/temp/HYG_2021_09.csv', sep=';', usecols=
['Time', 'HYG_FT02', 'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV',
'HYG_PU19_SF_PV', 'HYG_PU19_MO', 'HYG_PT16'], decimal='.', encoding="UTF-8")])

df = df.append(df2)
df['Time'] = pd.DatetimeIndex(df['Time'])

dfStatus = pd.concat([pd.read_csv('HYG_PU19Status.csv', sep=';', usecols=
['Time', 'HYG_PU19_DEV_ID', 'HYG_PU19_STATUS'], decimal='.', encoding="ISO-
8859-1")])
dfStatus['Time'] = pd.DatetimeIndex(dfStatus['Time'])

df = pd.merge_asof(df.sort_values('Time'), dfStatus.sort_values('Time'),
on='Time')

# Start and end time of work data
start = datetime.datetime(2021,9,1, tzinfo=datetime.timezone.utc)
end = datetime.datetime(2021,10,1, tzinfo=datetime.timezone.utc)

mask = (df['Time'] >= start) & (df['Time'] < end)
df = df.loc[mask]

features  = ['HYG_FT02', 'HYG_PT15', 'HYG_PU19_PW_PV', 'HYG_PU19_TQ_PV',
'HYG_PU19_SF_PV', 'HYG_PU19_MO', 'HYG_PT16']
```

```
stoppedMask = df['HYG_PU19_SF_PV'] <= 0.0
statusColumn = 'HYG_PU19_STATUS'
df.loc[stoppedMask, statusColumn] = 0

uniquePumpIDs = df['HYG_PU19_DEV_ID'].unique()

if(True):
    for pumpID in uniquePumpIDs:
        if len(df.loc[(df['HYG_PU19_DEV_ID'] == pumpID) &
(df['HYG_PU19_STATUS'] <= 1)]) >= 120:
            scaler = StandardScaler()
            scaler.fit(df.loc[(df['HYG_PU19_DEV_ID'] == pumpID) &
(df['HYG_PU19_STATUS'] <= 1), features].head(120))
            df.loc[df['HYG_PU19_DEV_ID'] == pumpID, features] =
scaler.transform(df.loc[df['HYG_PU19_DEV_ID'] == pumpID, features])
            print("Scaler values for pump" + str(pumpID))
            for i in range(scaler.n_features_in_):
                print("Feature " + str(i))
                print("Mean: " + str(scaler.mean_[i]) + " Scale: " +
str(scaler.scale_[i]))
        else:
            df = df.drop(df[(df['HYG_PU19_DEV_ID'] == pumpID)].index)

time_steps = 120

X_tot, y_tot = create_dataset(df['Time'], df[features], df[statusColumn],
time_steps)
y_tot = one_hot_encode(y_tot, 5)

X_train, X_test, y_train, y_test = train_test_split(X_tot, y_tot,
test_size=0.99, random_state=42)

def create_model():
    model = keras.Sequential()
    model.add(keras.layers.LSTM(32, input_shape=(X_train.shape[1],
X_train.shape[2]), return_sequences=True))
    model.add(keras.layers.LSTM(16))
    model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    return model

model = create_model()
model.summary()

if(train_model):
```

```
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
    cp_callback = keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
save_weights_only=True, verbose=1)

    history = model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=32,
        callbacks=[callback, cp_callback],
        verbose=1,
        shuffle=True,
        validation_split=0.2
    )

    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Validation')
    plt.legend()
    plt.savefig('training_PU19_individualscale/lossplot.png')
    plt.show()

if(load_test_model):
    model.load_weights(checkpoint_path)

    y_pred = model.predict(X_test)

    y_pred_dec = one_hot_decode(y_pred)
    y_test_dec = one_hot_decode(y_test)

    print("Total Accuracy: " + str(accuracy_score(y_test_dec, y_pred_dec)))
    lstm_cm = confusion_matrix(y_test_dec, y_pred_dec)
    lstm_cm_norm = lstm_cm.astype('float') / lstm_cm.sum(axis=1)[:,
np.newaxis]
    lstm_cm_acc = lstm_cm_norm.diagonal()
    print("Accuracy Label 0 (Stopped): ", lstm_cm_acc[0])
    print("Accuracy Label 1 (Normal run): ", lstm_cm_acc[1])
    print("Accuracy Label 2 (< week from fail): ", lstm_cm_acc[2])
    print("Accuracy Label 3 (< 24h from fail): ", lstm_cm_acc[3])
    print("Accuracy Label 4 (< 1h from fail): ", lstm_cm_acc[4])

    failmatrix1 = np.zeros((3,5))
    for i in range(len(lstm_cm[0])):
        failmatrix1[0,i] = lstm_cm[0,i]
        failmatrix1[1,i] = lstm_cm[1,i]
        failmatrix1[2,i] = lstm_cm[2,i] + lstm_cm[3,i] + lstm_cm[4,i]

    failmatrix2 = np.zeros((3,3))
    for i in range(3):
```

```
        failmatrix2[i,0] = failmatrix1[i,0]
        failmatrix2[i,1] = failmatrix1[i,1]
        failmatrix2[i,2] = failmatrix1[i,2] + failmatrix1[i,3] +
failmatrix1[i,4]

    lstm_fail_norm = failmatrix2.astype('float') / failmatrix2.sum(axis=1)[:,
np.newaxis]
    lstm_fail_acc = lstm_fail_norm.diagonal()

    print("Fail mode accuracy: ", lstm_fail_acc[2])

    lstm_disp = ConfusionMatrixDisplay(confusion_matrix=lstm_cm)
    lstm_disp.plot()
    plt.savefig('training_PU19_individualscale/cfsmatplot.png')
    plt.show(block=True)
```