YRKESHÖGSKOLAN
NOVIA

Exploring Anomaly Detection in Building Automation Through Machine Learning Techniques

Robin Lyttbacka

Master's Thesis in Automation Technology

The Degree Programme in Automation Technology – Intelligent systems

Vaasa, 2023

MASTER THESIS

Author: Robin Lyttbacka
Degree Program and place of study: Automation Technology, Vaasa
Specialization: Intelligent systems
Supervisor(s): Ray Pörn, Lauri Westerholm

Title: Exploring Anomaly Detection in Building Automation Through Machine Learning Techniques

---

Date: 12.1.2024 Number of pages: 65 Appendices: 2

Abstract

The purpose of this thesis was to analyze and improve anomaly detection for ventilation processes in building automation. The final goal of the thesis was to develop an anomaly detection method that was more accurate than the one previously used for the same purpose.

The anomaly detection method was developed at Fidelix. Fidelix is a company that provides building automation solutions. Fidelix has a service called Flow_how and data collected from the Fidelix Flow_how service was used to develop the anomaly detection method.

The theory part describes how the ventilation process is structured and how it is implemented in a building automation system. Furthermore, different types of anomaly detection methods and different machine learning methods are discussed. The anomaly detection methods that have been used and analyzed are LSTM (Long Short Term Memory), Kmeans, Isolation forest and a Python machine learning library called Pycaret.

The result of the thesis was an anomaly detection method that was more accurate than the one previously used. The new method is faster than the one previously used. At the time of writing, the previous version is still used. The method developed in this thesis is available for future use.

Language: English
Key Words: Anomaly detection, building automation, machine learning, Python, unsupervised learning

EXAMENSARBETE

Författare: Robin Lyttbacka
Utbildning och ort: Automation, Vasa
Inriktning: Intelligenta system
Handledare: Ray Pörn, Lauri Westerholm

Titel: Exploring Anomaly Detection in Building Automation Through Machine Learning Techniques

Datum: 12.1.2024 Sidantal: 65 Bilagor: 2

Abstrakt

Syftet med detta examensarbete var att analysera och förbättra avvikelse detektering för ventilations processer inom fastighetsautomatik. Avsikten var att få fram en avvikelsedetekteringsmetod som var mera exakt än den tidigare använda för samma ändamål.

Avvikelsedetekteringsmetoden utvecklades vid Fidelix. Fidelix är ett företag som tillhandahåller fastighetsautomatiklösningar. Insamlad data från Fidelix Flow_how tjänst användes för att utveckla avvikelsedetekteringsmetoden.

I teoridelen beskrivs det hur ventilations processen är strukturerad och hur denna implementeras i ett fastighetsautomationsystem. Vidare behandlas olika typer av avvikelse detekterings metoder och olika maskininlärningsmetoder. Avvikelsedetekteringsmetoderna som har använts och analyserats är LSTM (Long Short Term Memory), Kmeans, Isolation forest och Python maskininlärningsbibiliotek kallat Pycaret.

Resultatet blev en avvikelsedetekteringsmetod som är mera exakt än den som tidigare användes. Den nya metoden är också snabbare än den som tidigare användes. För tillfället används fortfarande den tidigare versionen. Det nya programmet finns till förfogande för framtida behov.

Språk: Engelska
Nyckelord: Anomali detektering, fastighetsautomatik, maskininlärning, Python, oövervakad inlärning

OPINNÄYTETYÖ

Tekijä: Robin Lyttbacka
Koulutus ja paikkakunta: Automaatiotekniikka, Vaasa
Suuntautumisvaihtoehto: Tekoäly
Ohjaaja(t): Ray Pörn, Lauri Westerholm

Nimike: Exploring Anomaly Detection in Building Automation Through Machine Learning Techniques

Päivämäärä: 12.1.2024 Sivumäärä: 65 Liitteet: 2

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli analysoida ja parantaa ilmanvaihtoprosessien poikkeamien havaitsemista rakennusautomaatiossa. Opinnäytetyön lopullisena tavoitteena oli kehittää poikkeamien havaitsemismenetelmä, joka on tarkempi kuin aiemmin samaan tarkoitukseen käytetty menetelmä.

Anomalian havaitsemismenetelmä kehitettiin Fidelixissä. Fidelix on yritys, joka tarjoaa kiinteistöautomaatioratkaisuja. Fidelixillä on Flow_how-niminen palvelu, Fidelixin Flow_how-palvelusta kerättyjä tietoja analysoitiin poikkeamien havaitsemismenetelmän kehittämiseksi.

Teoriaosuudessa kuvataan, miten ilmanvaihtoprosessi rakentuu ja miten se toteutetaan rakennusautomaatiojärjestelmässä. Lisäksi käsitellään erityyppisiä poikkeamien havaitsemismenetelmiä ja erilaisia koneoppimismenetelmiä. Käytetyt ja analysoidut poikkeamien havaitsemismenetelmä ovat LSTM (Long Short Term Memory), Kmeans, Isolation forest ja Python-koneoppimiskirjasto Pycaret.

Opinnäytetyön tuloksena saatiin aikaisempaa tarkempi poikkeamien havaitsemismenetelmä. Menetelmä oli myös nopeampi kuin aiemmin käytetty menetelmä. Kirjoitushetkellä käytetään edelleen edellistä versiota. Tässä opinnäytetyössä käytetty ohjelma on käytettävissä myöhempää käyttöä varten.

Kieli: Englanti
Avainsanat: Poikkeaminen havaitsemista, rakennusautomaatio, kone-oppiminen, Python, valvomaton oppiminen

Table of Contents

1 Introduction

What if a building could talk and tell you how to best optimize the climate and temperature inside? Or tell you if something that controls these processes breaks down? The building can not tell you that the heat exchanger on the fifth floor has a malfunctioning sensor. This could be done by having a machine learning system monitor the processes and react to any abnormal events.

Over the years buildings have become more and more complex, with different types of systems that control different house technologies. Different systems can control the lights, heating and ventilation. It is becoming increasingly more common in smart homes that these systems are connected so that the user can have a clear and easy overview of all their "gadgets". In larger properties, it is a challenge to keep these smart systems connected to only one overhead system. From a user perspective, it would be appreciated if one could control all the different properties systems from a single overhead system.

Within building automation, it is becoming more common to monitor other systems and even control other systems. One example of this is a lightning system that is activated by presence. This presence control can also set the air ventilation at a specific speed in a specific room. If the overhead system could learn from itself when the lights need to come on and could identify individual temperature needs for different people, it could save a lot of energy. If we go one step further and analyze and compare the data of multiple buildings, there could be a possibility to detect anomalies.

1.1 Background

A basic automation system can be built using some relays and sensors. These sensors control the relays which then can, for example, start and stop an air handling unit. Another type of automation system can be light control, using presence sensors, lux sensors and control relays.

A modern automation system consists of I/O modules. There are four commonly used I/O modules: Analog Input (AI), Analog Output (AO), Digital Input (DI) and Digital Output (DO).

The data from and to these I/O modules are controlled by a CPU. The CPU runs a program that has been made specifically for a process or processes.

Depending on which type of CPU is used there are possibilities to access different types of communication protocols. A gateway is often used to access these protocols. An example of communication protocols are Modbus, M-bus, BACnet and KNX.

There are different ways to access and control the data gathered by the CPU. One commonly used approach is that there is a screen connected to the CPU. From the screen, all the data is readable and controllable by the user. Another commonly used approach is to access the CPU over the internet or through a VPN.

If a property has more than one CPU, a common approach is to have some sort of control system to gather the data from all the CPUs in one place. These systems are often cloud-based. From the cloud service, it is possible to share data between CPUs and to control I/O modules from one CPU to another.

There is a lot of data accumulated when gathering data from many properties with many CPUs. Using this data it is possible to compare properties to one another. Another possibility with all the gathered data is to analyze it to check for errors in the processes. This can be done by machine learning and anomaly detection.

1.2 The purpose of the study

The purpose of this study is to collect and use data gathered from different sensors in buildings and compare several anomaly detection methods. The main goal is to create an anomaly detector prototype that would be more sensitive and more accurate than the one previously used. The anomaly detection system is meant to be used by energy experts. The anomaly detection system can also be used to create reports for service personal.

Another aspect of this study is to compare the new system with the older one, which uses a method called Long Short Term Memory or LSTM to detect anomalies. To make the anomalies detection more precise and sensitive it will be compared with other algorithms and how they detect anomalies.

2 Theoretical part

This chapter includes the theory behind the thesis. The chapter starts with a short description of building automation systems, how air handling units work and some information about the company that the anomaly detection system is created for. At the end of this chapter machine learning theory is presented.

2.1 Building automation systems

Building automation is often abbreviated HVAC (Heating, Ventilation and Air Conditioning). An automation system is all about controlling, regulating and monitoring the heating, ventilation and air condition processes. Often multiple systems can be controlled by one system, like systems for lightning, fire alarms, doors, etc. (Mossberg Sonnek & Lindgren, 2015)

These building automation systems have progressed a lot through the years. Earlier on, one system could be used to control a process and another one to supervise. Nowadays everything is integrated into one system. (Mossberg Sonnek & Lindgren, 2015)

An automation system can be divided into three different levels. The first level or the highest level includes the software that gathers all the data from one or many buildings. From this level, it is possible to analyze and manually set values in the systems. This level is called the information level. The second level controls and maneuvers the equipment automatically by given values in the system. The equipment is often installed in cabinets in a technical space or in smaller buildings combined with the distribution board. This level is called the automation level. The third level includes the sensors for detection and measurement and the maneuvering of actuators. This level is called the field level. (Mossberg Sonnek & Lindgren, 2015)

An important aspect to consider in operating a building is the cost required for ventilation, heating/cooling and illumination. With the possibility to control and program a start/stop - function, speed settings, heating duty cycling, on/off controlling and more, it’s possible to reduce the cost of operating a building. (Wang, 2010)

2.1.1 AHU Processes

The basic process of an Air Handling Unit is to extract fresh air from the outside, pass it through a machine and deposit it inside. The Air Handling Unit extracts the inside air from the room or the building and passes it through the air handling unit to the outside. (Venus, 2020)

The key components of an Air Handling Unit (AHU) are:

- Inlet and outlet dampers: these open when the AHU is running and close when it's not.
- Filters: both on the fresh air side and on the extract air side.
- Cooling and heating coils: Depending on the type of AHU, there can be a combination of both or only heating coils.
- Fans: Fans are controlled through a modern process by an EC-engine or can be controlled by a frequency converter.
- Heat recovery: There are a few types of heat recovery systems for AHU processes. The two commonly used are thermal wheels Figure 1 and plate heat exchangers Figure 2. (Evans, Air Handling Units Explained, 2018)

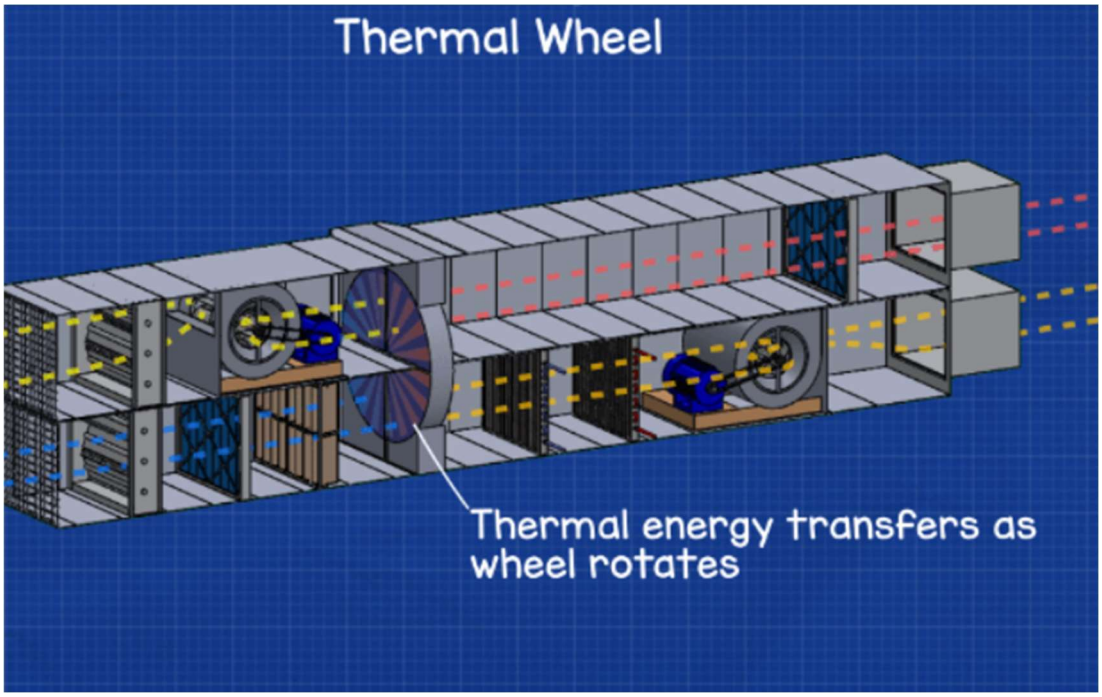


Figure 1 Thermal wheel heat recovery. (Evans, Air *Handling* Units Explained, 2018)

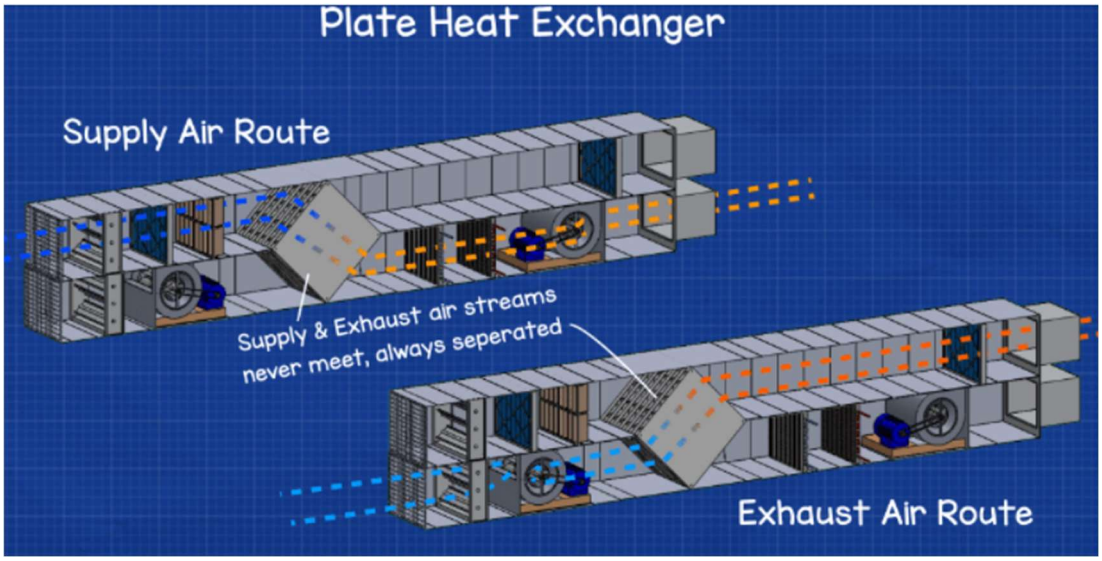


Figure 2 Plate heat exchanger. (Evans, Air Handling Units Explained, 2018)

Air handling units are then filled with sensors and controlled by the building automation system. The inlet and outlet dampers are controlled by the building automation system so that the dampers are open when the fan starts. The filters have pressure sensors that measure the pressure and give an alarm if it surpasses a given value. The heating and cooling coils are controlled by measuring the temperature that leaves the AHU. The temperature is given a set value and if the actual value is below the set value, the heating coil will produce more heat into the air. The energy recovery is also controlled by the set value for the temperature. The speed of the fans is controlled by pressure or flow in the duct. There can also be humidifiers in the AHU process. The humidifiers are controlled by steam or water mist or by the cooling coil to dehumidify. (Evans, Air Handling Units Explained, 2018).

2.1.2 Fidelix

This anomaly detection system was created for a company called Fidelix. Fidelix is one of the leading companies in Finland that deals with building automation. Fidelix was founded in 2002 and has its headquarters in Vantaa, Finland. (Fidelix, 2022)

The company develops and test their own products. Their product range includes their own PLC, I/O modules, sensors and more. Apart from this they also offer different kinds of web services. (Fidelix, 2022)

2.2 Machine learning

In the modern world, everything and everyone generates and consumes data. This data can be as simple as what type of ice cream a person eats to a program checking for spam emails. These are types of data that a computer algorithm can analyze.

A computer algorithm can be useful for many different purposes, from analyzing and sorting data about people's ice-cream preferences to sorting numbers according to specific requirements. For certain tasks, it can be hard to find an already known algorithm, when there are only a known input and a known output. For example, the input can be a text document sent by email and the output is spam or not spam. In this scenario, the computer can be given a lot of different emails sorted as spam and not spam and then the computer can learn from this data and be able to sort out new incoming emails accordingly. (Alpaydin, 2014)

2.2.1 Supervised learning

In supervised learning, a training set is used to instruct supervised learning models to produce desired results. The algorithm learns by itself by comparing the predicted output to the known output and measuring the error rate through a loss function. The algorithm adjusts itself until the error is sufficiently reduced. (IBM, n.d.).

A supervised learning method uses labeled data, this is called training data. In Figure 3, the training data consists of pictures of cats and dogs, these are labeled "Cats" and "Dogs". This machine learning method then predicts a new prediction and compares the new prediction to the training data. In the picture, the unpredicted model is a picture of a cat. The machine learning method compares the unpredicted model against the training data and by this, it can predict the label of the unpredicted model. (Serrano, 2021).

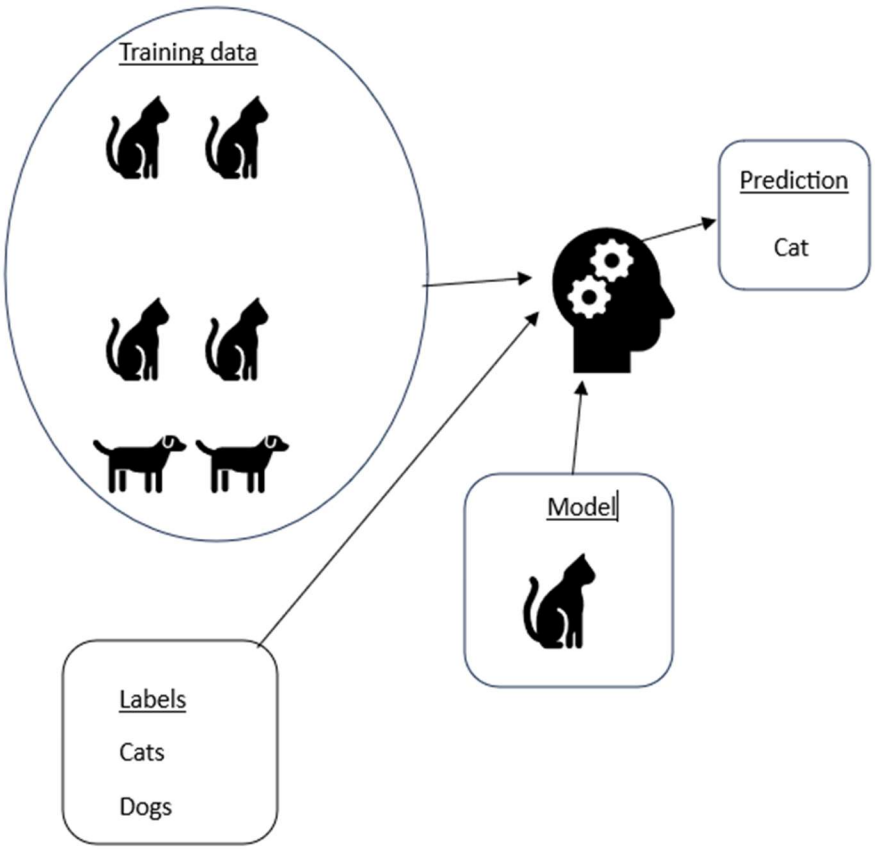


Figure 3 Supervised learning

2.2.2 Unsupervised learning

Compared to supervised learning where the goal is to learn what the output should look like given the input with the help of a supervisor (label), in unsupervised learning there is only an input and no supervisor. The goal of unsupervised learning is for the model to find consistencies or inconsistencies in the input. One method in unsupervised learning is clustering. This is a type of density estimation. Using density estimation in statistics the unsupervised model recognizes patterns in the input data, where some patterns occur more often than others. (Alpaydin, 2014). See Figure 4 for a graphic explanation of the unsupervised learning model.

Another task of unsupervised learning is outlier detection. In outlier mining detection the model tries to find a smaller group of objects that are significantly different from the rest of the data. (Bhattacharyya & Kalita, 2014)

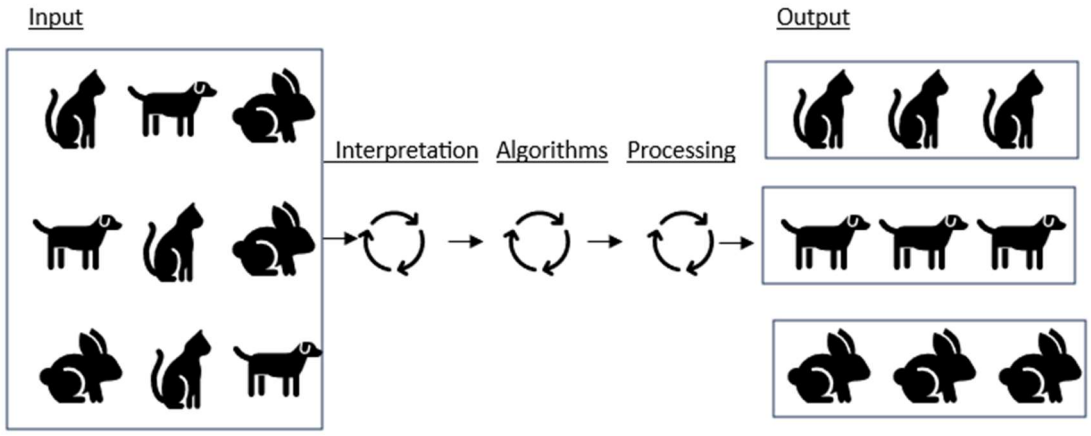


Figure 4. Unsupervised learning.

2.2.3 Semi-supervised learning

Semi-supervised learning is a combination of supervised learning and unsupervised learning. In semi-supervised learning, a small portion of the data is labeled and a big portion of the data is unlabeled. The small portion of labeled data is used as training data for the larger dataset. (Potrimba, 2022)

2.2.4 Scaling

Scaling is a method to make data more suitable for processing or modeling. Scaling is needed when the gathered data includes data with different ranges and units. There are a few different types of scaling methods. One of these methods is normalization. With normalization, the values of the data will be set between 0 and 1. The minimum value of the gathered data will equal 0 and the maximum value will be equal to 1. (Bhandari, 2023). The formula for normalization is this:

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

Another type of scaling is standardization. In scaling the gathered data is centered around the mean value and divided by a standard deviation. In the formula for standardization, as seen below, the μ is the mean value of input data and σ is the standard deviation. (Bhandari, 2023)

$$X' = \frac{X - \mu}{\sigma}$$

2.2.5 PCA

PCA stands for Principal Component Analysis. PCA can be used as a dimensionality reduction method, which means it transforms a large dataset into a smaller dataset without losing accuracy. PCA captures the variance of the data with a straight line as seen in Figure 5. The goal of PCA is to transform the original dataset into new datasets, these are called components. The PCA components will get a magnitude and direction. The first component includes the most variation in the original data and the second component the second most variation and so on. (Biswal, 2023)

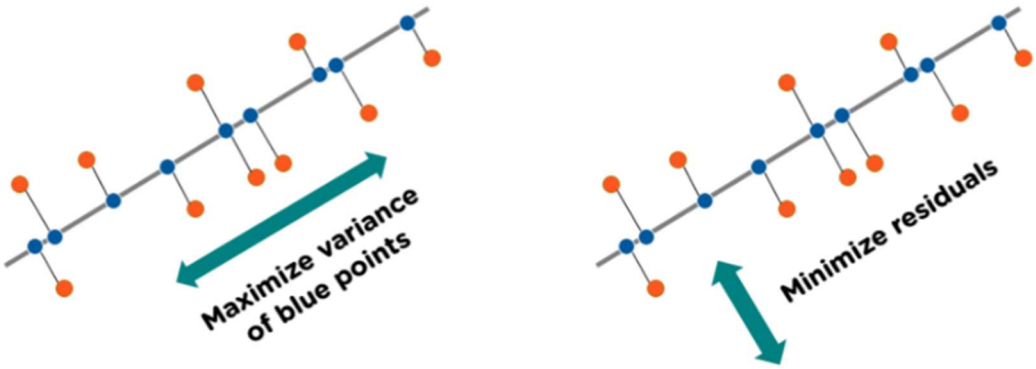


Figure 5. Principal Component Analysis. (Biswal, 2023)

3 Software

This chapter discusses the software used in the thesis work. The three main software that have been used are the Microsoft Azures web applications, Visplore and Python with different Python libraries.

3.1 Microsoft Azure

Microsoft Azure was first announced as a Community Technical Preview back in 2008, at this time it was called Windows Azure. In 2010 it was commercially available for use (Webber-Cross, 2014).

IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) are two services provided by Microsoft cloud computing services, Microsoft Azure. There is also a third cloud service called SaaS (Software as a Service). The lowest tier service is IaaS which offers storage, server and networking infrastructure. The next tier is PaaS and in this tier, users can build their applications. The highest tier of SaaS includes software products (Webber-Cross, 2014).

Public, private, hybrid and community clouds are the four primary cloud deployment options. Microsoft Azure is classed as public. It is possible to install Azure products as a private cloud model if it is installed on a private data center. A public cloud service is available for the public to use and hosted by a vendor. A private cloud service is only available within a company's domain. When a private cloud is shared between several users it is called a community cloud. A mixture of all these three services is called a hybrid cloud. (Webber-Cross, 2014)

The main reason for selecting any type of cloud service is if a company lacks its own infrastructure to run solutions on or if the infrastructure they have does not have sufficient capacity. It can be expensive to have your own infrastructure and it can also be challenging to migrate from an older infrastructure to a newer one. Choosing a cloud service can also be more cost-efficient than hosting your own infrastructure. (Webber-Cross, 2014)

3.2 Python libraries

A few Python libraries have been used in this thesis. Some of them where needed for analyzing the data, others to calculate results and to show graphs of the results. The library for analyzing was Tensorflow, Pandas was used for the different data frames, matplotlib for graphs, scikit-learn for result calculations and machine learning models and Pycaret for anomaly detection methods.

3.2.1 Tensorflow

TensorFlow was originally developed by Google for internal use only. In 2015 it was released as open source. Google still uses Tensorflow for research and product development and it is also used for the development of machine learning and deep neural network models. (Fandango, 2018)

Tensorflow can be split into three different elements called models: Data model, Programming model and Execution model.

- The data model includes the basic elements of computation, in Tensorflow this is called a tensor. This can be an n-dimensional collection of data.
- The programming model includes graphs. Creating a program in Tensorflow involves some sort of computational graphs.
- The execution model includes the computation of graph nodes in a series of dependent orders. (Fandango, 2018)

3.2.2 Pandas

Pandas is an open-source library available in Python. Pandas was initially created and designed for financial data analysis. Pandas was developed in 2008 by Wes McKinney, in 2009 it was open-sourced. The main goal with Pandas is to discover information easily and quickly in data. Pandas has gained wide adoption in many areas because of its extensive feature set and smooth interaction with Python and other tools. (Heydt, 2017)

Some features of Pandas are:

- Manipulation of data series and data frame objects.
- Data alignment with indexes and labels.
- Missing data correction.
- Tidying messy data.
- Possibility to read and store data in various file formats such as CSV, HDF5 and JSON.

3.2.3 Matplotlib

Matplotlib is a data visualization library for Python. Matplotlib is easy to use for a variety of different plots, such as line, scatter, box, bar and radial plots. For easy plotting, Matplotlib provides a straightforward oriented interface called Pyplot module. (Yim, Chung, & Yu, 2018)

3.2.4 Scikit-learn

Scikit-learn is an open-source software that is used to solve unsupervised and supervised problems in machine learning. The possibility to implement some of the most popular machine learning methods combined with Scikit-learn being easy to use makes Scikit-learn a popular library for Python. Scikit-learn is built completely in Python and also includes Python libraries such as Numpy and SciPy (Jolly, 2018).

3.2.5 Pycaret

One of many machine learning libraries that can be found for Python is Pycaret. Pycaret is inspired by the Caret machine learning package that can be found in the R coding language. The library includes many functions for different machine learning techniques such as

classification, comparing models, and evaluating models and predictions. (Brownlee, A Gentle Introduction to PyCaret for Machine Learning, 2020)

3.3 Visplore

Visplore is a software for analyzing time series data. The main features of this software are analytics, visualization, workflow and integration. These features include outlier detection, time series plot, data exploration, Python integration, etc. Visplore is a free software with some limitations of the professional or discovery versions. (Visplore, n.d.)

4 Anomaly detection

Anomalies are outliers, exceptions, surprises, or aberrations from a known pattern within a large set of data. Anomalies can occur due to malfunctioning devices/sensors, overload in the data reading, or any other type of error readings from a process. (Bhattacharyya & Kalita, 2014)

There are a few different types of anomalies. Point anomalies are singular data points that differ from the rest of the data points as seen in Figure 6. Point anomalies are used to detect sensor faults. (Rizwana, 2022)

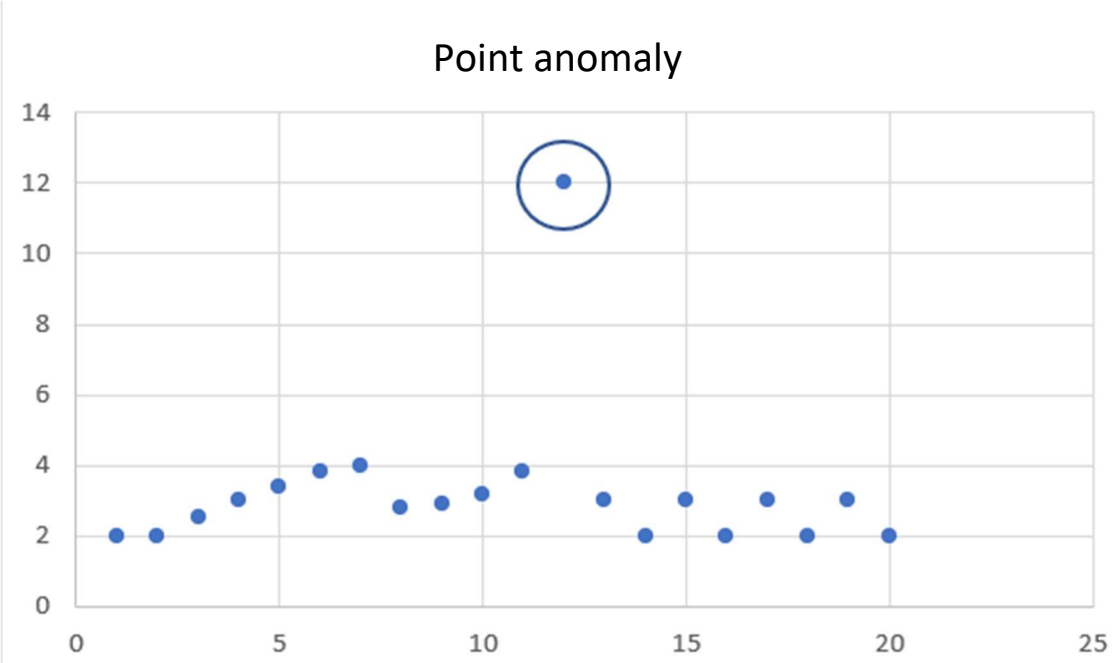


Figure 6. Point anomalies

Contextual anomalies are single data points that differ from the rest of the data points in a closed context. Contextual anomalies can be seen in time series data. Contextual anomaly only works if there is a specific context. For example, money spent on food is normal on weekdays and weekends but abnormal during holidays. (Rizwana, 2022). See a visual representation of the contextual anomalies in Figure 7.

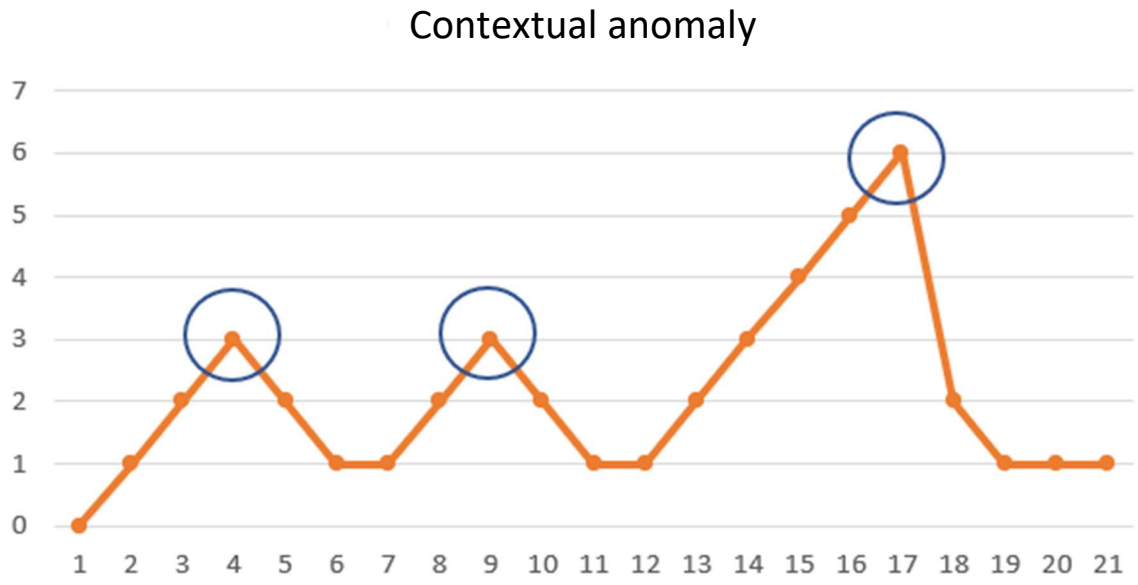


Figure 7. Contextual anomalies

Collective anomalies are a group of coinciding data points that when compared to the rest of the data points are anomalous. Collective anomaly detection can be used to detect for example data copying from a server to a local computer unexpectedly. This works by putting together instances of data. (Rizwana, 2022)

There are several challenges to anomaly detection. For example, it can be difficult to define a known interval when a sensor reading is correct but outside of the known pattern. If different systems or sensors, or even the same sensors in different systems, are used it can create another known pattern, which could cause issues. The availability of trained data and labeled data can cause issues if there isn't enough of either data available. If there is not enough data available it's naturally also difficult to recognize anomalies.

Because of these challenges, there is not one general system that works for everything. Depending on the scenario, type of data and data availability there are anomaly detection models that will work better or worse. Different scenarios that can influence what anomaly detection is best suited are for example machine learning, spectral theory, statistics, data mining, or information technology. (Chandola, Banerjee, & Kumar, 2007)

4.1 Supervised anomaly detection

Supervised anomaly detection needs to have training data with two labels. One label for normal data and the other for anomaly data. The more training data available the better the model will become. Data that is fed into the anomaly detection software will be compared with the previously labeled data. (Bhattacharyya & Kalita, 2014)

4.2 Unsupervised anomaly detection

Unsupervised anomaly detection does not require any training data, but it is difficult to attain the results at hand. If the data has a lot of anomalies, the detection method will give a lot of false alarms. That is because unsupervised anomaly detection requires that the normal data occur more often than the anomaly data. (Bhattacharyya & Kalita, 2014)

4.3 Different types of data

Real-world objects have a few different attributes to describe them. According to Han, Micheline & Pei (2000), these are the commonly used attributes:

- Nominal Variables: This variable doesn't have a numeric value, such as employee ID or gender.
- Ordinal Variables: This variable is similar to categorical with the difference that these are sorted in a fixed sequence.
- Categorical Variables: This variable can have more than two states, but as the name mentions they are categorical. Example colors and sensor type.
- Binary Variables: This type of variable can only have two states, true/false or 0/1.
- Interval-Scaled Variables: This variable describes the differences or intervals between values.
- Ratio-Scaled Variables. This variable describes a positive measurement on a nonlinear scale.
- Mixed-type Variables. This variable can be a mix of the variables listed above.

4.4 Machine learning methods for anomaly detection

The general structure of an anomaly detection system consists of some basic stages, parametrization, training and anomaly detection. First, the raw data is collected and prepared for the anomaly detection system. This stage is called preprocessing. The second stage is where the model gets trained according to training data. The training data can be manually selected or selected by some automatic method. The last stage is the detection stage. In the detection stage, the data in the preprocessing stage is compared to the data in the training stage. A threshold criterion is configurable to detect anomalous data. (Omar, Ngadi, & Jebur, 2013)

4.4.1 KNN

KNN is an abbreviation of K-Nearest Neighbor. It is a simple machine-learning method for classifying a new data point. (Yehoshua, n.d.) New data points in KNN get classified by calculating the Euclidian distance between the new data point and the previously labeled data points. A value is set for KNN for how many “neighbors” the new data point has. The data point will be classified into the same class as the majority of neighbors. (Ahmed)

In Figure 7 the new data point is classified as an anomaly, here the number of neighbors has been set to five. The new data point is in orange and the closest “neighbors” are in gray.

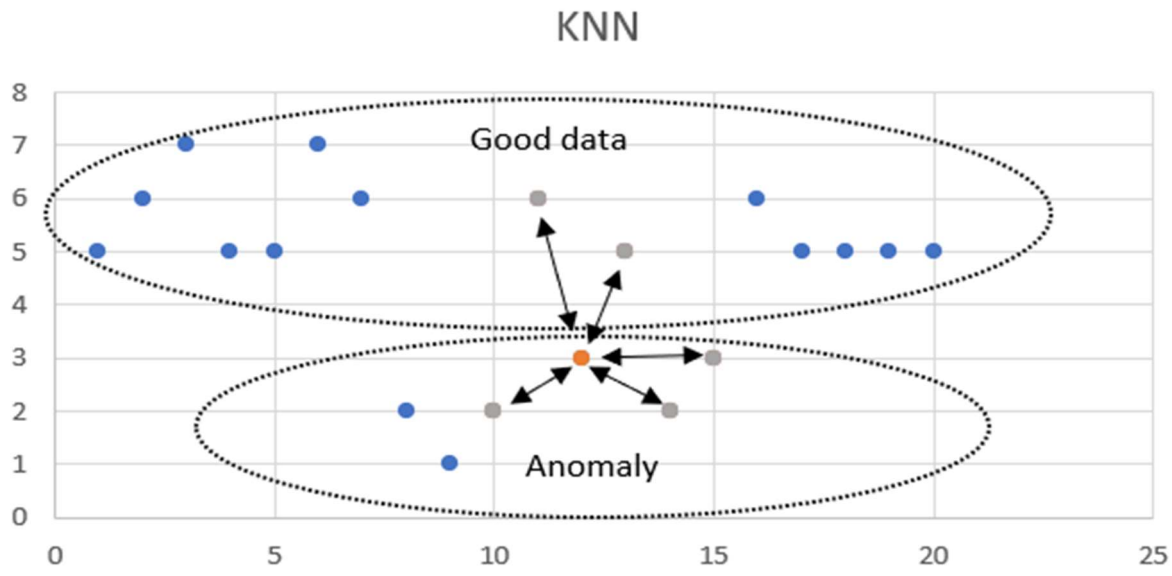


Figure 8. KNN

4.4.2 Clustering

One method in machine learning and more specifically in unsupervised learning is clustering. The task of clustering is to find similar characteristics in the data by dividing the data set into numbers of clusters. The data points gathered in the same clusters should have similar features, values, or characteristics. Clustering can be seen as grouping data points so that the data between data points is held at a minimum, as in Figure 9. (Rohit, 2022).

There are two main ways of clustering. The first is hard clustering, where a single data point belongs to a single cluster. In the second category, soft clustering, a data point can belong to a certain number of clusters. (Rohit, 2022)

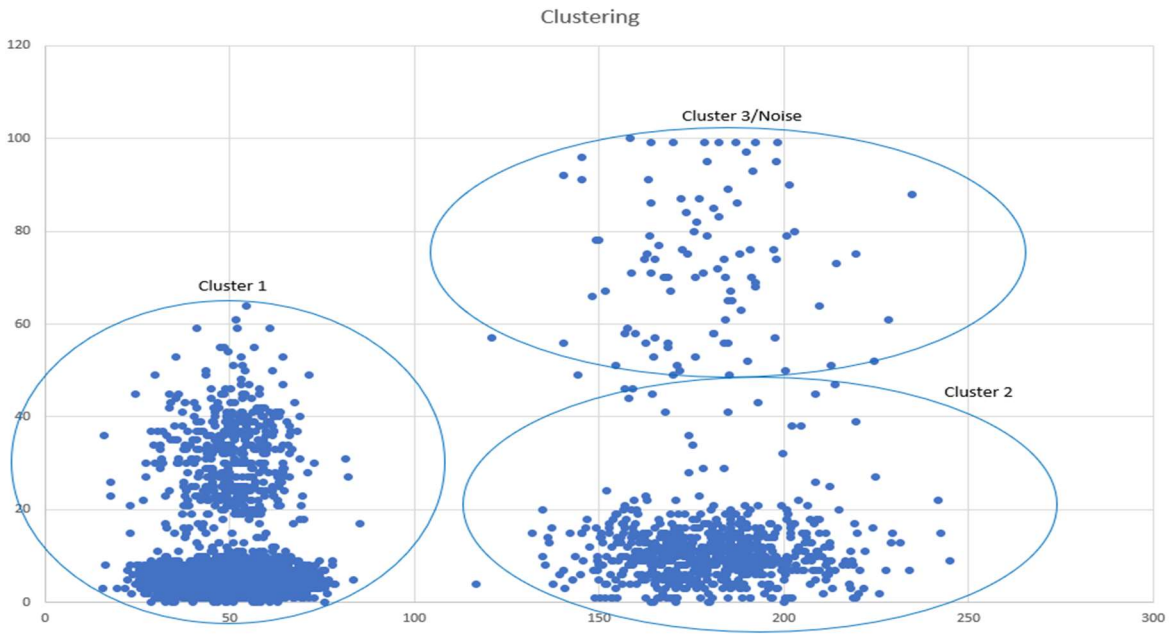


Figure 9 Clustering

4.4.3 K-means

K-means is a type of clustering method. The first step with K-means is to set the number of clusters the data should be split into. The next step in K-means is to initialize the center point, called centroids, of the cluster. This is often set at random value because it is unknown. In figure 10 the number of clusters is set to three and the centroids are marked with an x. The next step in K-means is to calculate distances from data points to centroid points, where the shortest distance between data point and centroid makes a cluster, as seen in Figure 11. After this, a new value for the centroid point is calculated from the mean value of data points in the cluster. These steps are then repeated until the value of the centroid point doesn't change, as seen in Figure 12. (Sharma, 2023)

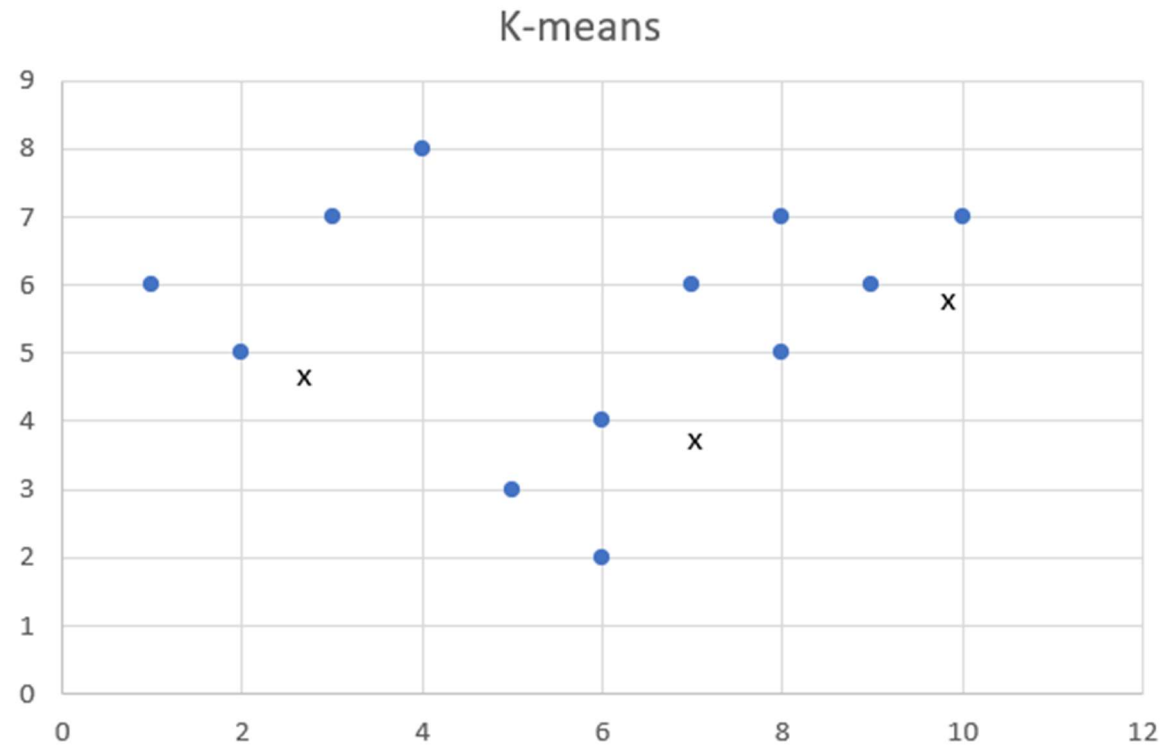


Figure 10. K-means centroids.

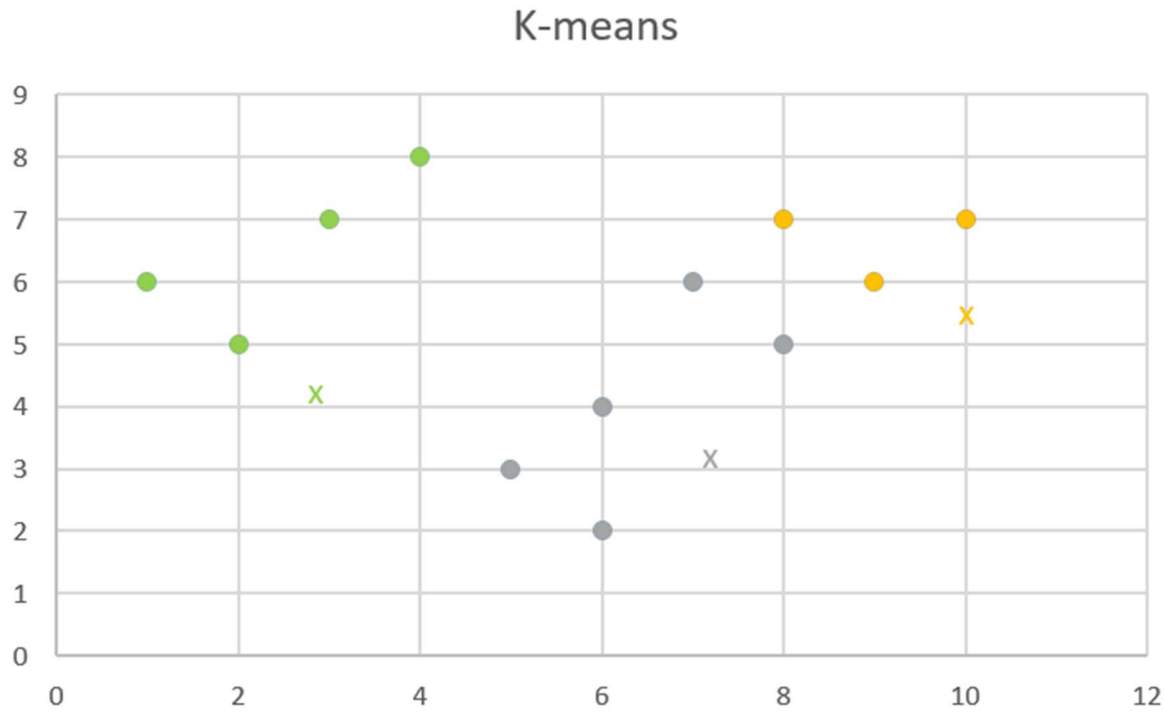


Figure 11. K-means centroid cluster.

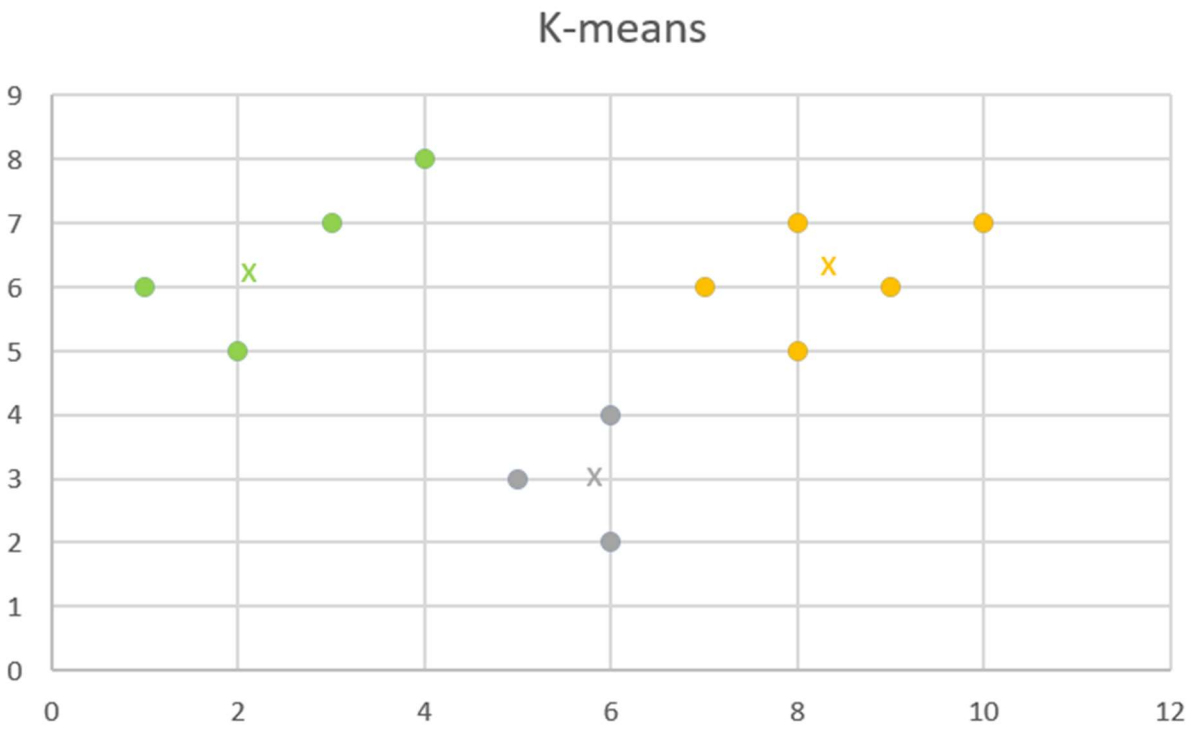


Figure 12. K-means centroid value.

4.4.4 Isolation forest

Normally in anomaly detection, some known data is set as the normal data, this data can be set by the user. All data that is not considered normal data is then flagged as anomalous. Isolation forest does not work with a normal/anomalous data setup, instead, it works by isolating anomalous points in the dataset (Dhiraj, 2020). By isolating all data points in a dataset, anomalies are easier to isolate than normal data. As seen in Figure 13, data point X requires less partition from the rest of the data compared to data point Y. (Liu & Ting, 2009)

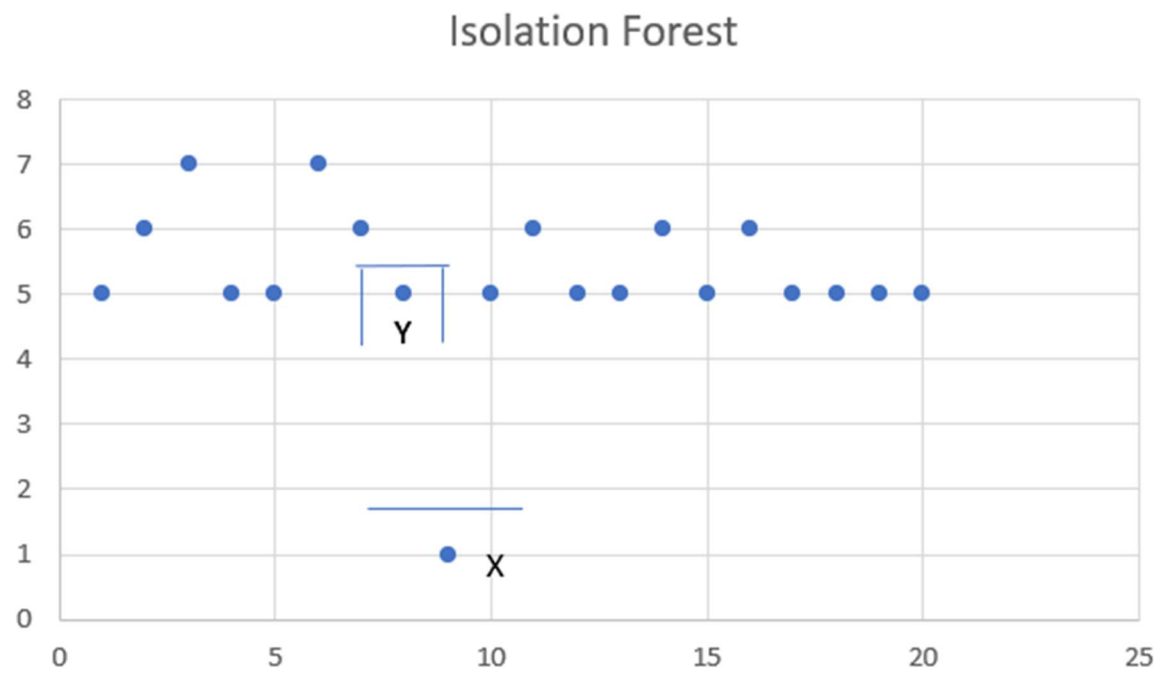


Figure 13. Isolation Forest

4.4.5 LSTM

At the beginning of this thesis work, LSTM (Long Short-Term Memory) was the model used at Fidelix. LSTM is used at Fidelix because it should learn time-series relations.

LSTM is an enhanced recurrent neural network (RNN). RNN has a gradient vanishing problem, but this is not a problem with LSTM. RNNs have the drawback of being unable to recall long-term dependencies. These issues are specifically avoided by using LSTM. (Saxena, 2021)

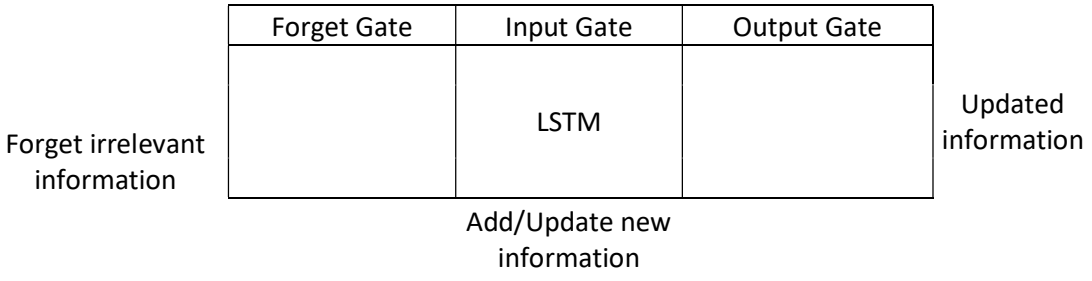


Figure 14. LSTM Gates.

An LSTM cell is divided into three gates and a memory cell. The three cells are a forget gate, an input gate and an output gate, as seen in Figure 14. The forget gate takes care of deciding what information is important and what information to forget. A sigmoid function is used to decide how important the information is. This sigmoid function has a value between 0 and 1. A value closer to 0 means that it isn't important and a value closer to 1 means that it is important. (Saxena, 2021)

The input gate adds information to a neuron cell. The input gate uses an activation function, typically a Tanh function. This function generates a value between -1 and 1. A filter is used to control what information that will be stored in the cell. The filter uses a sigmoid function. By combining the output from the input and the forget gate the memory cell is updated. (Saxena, 2021)

The output gate uses the Tanh function to create a vector of values. A sigmoid function is used to filter out output values from previous output and current input. (Choubey, 2020). See figure 15 for a graphic explanation of the LSTM model.

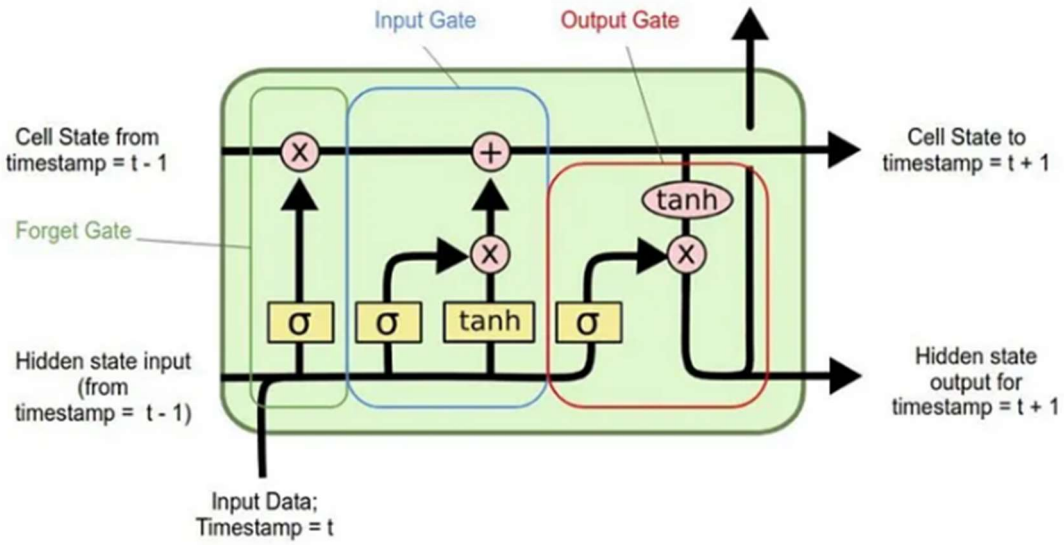


Figure 15. LSTM (Saxena, 2021)

5 Data analysis

The data in this thesis is gathered using Microsoft Azure Databricks. Using Databricks the data is first read and then filtered according to buildings, processes and timestamps. With the assistance of the supervisor at Fidelix, it was decided, that for the dataset to be used for the analysis I would need to take data from three different buildings, with about 50 different processes and about two-year-old data. All the data from different processes are stored in Microsoft Azure Datalake through a Datafactory. The data can then be accessed and processed with Azure Databricks. In Figure 16 it is described how the system works.

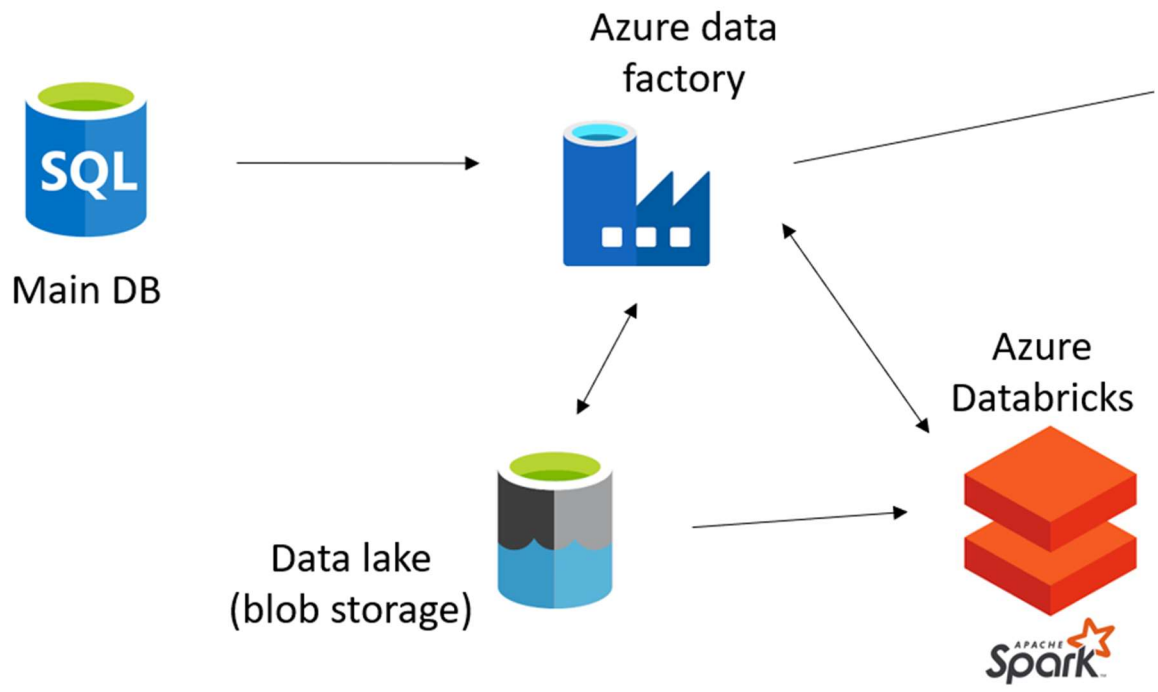


Figure 16. Data access.

5.1 Data gathering using Databricks

A fixed dataset is needed for this thesis work because the data changes periodically so it would be difficult to compare different methods if the data changes every time the code runs. The prepared dataset consists of data from three different buildings. Each building

consists of a variety of different air handling unit processes. In total, there were data from 50 different processes.

The data used in this thesis work is taken from air handling unit processes (AHU processes). There are a few other processes, like cooling- and heating processes, but these processes were filtered out in the fixed dataset. The timespan for the fixed dataset was about two years.

5.1.1 Data filtrering using Databricks

The data stored in a Datalake is accessed from Microsoft Databricks. To be able to access the data stored in a storage account a secret key is needed. This key was provided by Fidelix at the beginning of the thesis work.

A file named processTagMeasurment contains all the data needed for data filtering. The execution of the command to read this file was slow because the file includes a lot of buildings, processes and data. The file processTagMeasurment includes entries according to Code example 1.

Code example 1. ProcessTagMeasurment entries.

```
{
 "id": "78a0152c-0b8d-eb11-b566-000d3a229b96",
 "tag":{"id": 11923, "name": "TK303 D-osa 2013 QE27_FH_LOI"},
 "timestamp": "2021-03-25T03:40:00.0000000Z",
 "value": 0.0,
 "unit": "",
 "process": {"id": "204_QE27", "name":"QE27"},
 "building":{"id": 204, "name": "RakennusA"}
}
```

“id”: This is the id used in FlowHow.

“tag”: This tells what sensor/point from the process, in this example: “TK303 D-osa 2013 QE27_FH_LOI"}. This is an alarm point when the air quality is too low. QE for air quality, FH for alarm point and LO for low. “TK303 D-osa 2013” tells from which process and where it is.

“timestamp”: This is when the data is saved.

“value”: What value does the sensor/point have at the given timestamp.

“unit”: Tells what unit the data is saved in. In the example when this is an alarm point there are no units.

“process”: Tells what process it is from.

“building”: From what building the process is taken from.

The data that is collected is called “processTagMeasurment_nested and contains the following attributes: ID, Tag, Timestamp, Value, Unit, Process, and Building.

The entries used in this thesis to filter the data are:

- Timestamp: To filter the data accordingly to the desired period
- Process: To filter out other than AHU processes.
- Building: Filter the data to get the desired buildings.

The timestamp filtering was done by checking manually on the Web-service (Flow_how). On this service, the energy experts can create a report when something doesn’t work as it should. These reports where used to check for known anomalies on a known timestamp. As seen in Figure 17, an energy expert has made a report that something is wrong. In the figure, we can see the timestamp from when the report was submitted, 06.03.2022. The observation is that the temperature sensor TE05 shows a bad value.

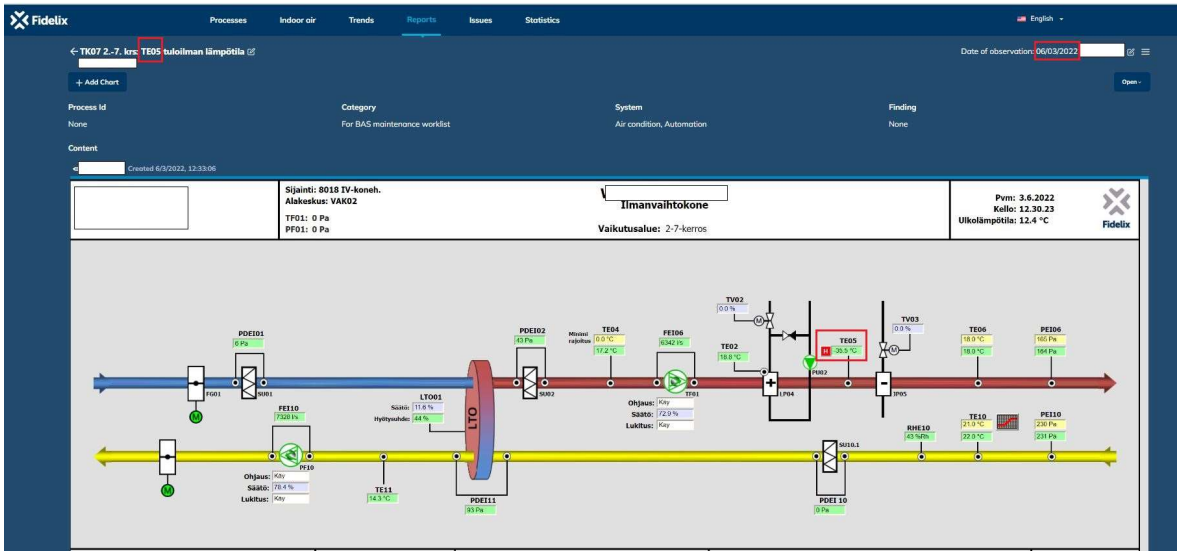


Figure 17. Flow How report

The data in processTagMeasurments included different types of processes, such as air handling unit processes, cooling processes and heating processes. In this thesis work the main goal was to analyze processes from air handling units. To be able to filter out other processes I had to search for names of air-handling unit processes which was successfully completed.

It was agreed upon to use processes from bigger buildings, like bigger malls and supermarkets. In these buildings, there are often quite many AHU-processes and when they are in the same building they are probably from the same manufacturer and include the same type of sensors.

When the data was filtered according to the desired timestamp, process and building the datasets were downloaded so it was possible to look at the data graphically. The datasets were saved separately for each process and stored in CSV-format. In total about 50 processes were downloaded locally. What the data from one process looks like can be seen in Table 1.

Table 1. Data in CSV-format

timestamp	value	tagName	tagId	processId
01-01-2021 02:10	0.0	TK01_POISTOSUODATIN_PAINE-ERO_FHP	12588	200_TK01
01-01-2021 02:10	0.0	TK01_TULOSUODATIN_PAINE-ERO_FHP	12585	200_TK01
01-01-2021 02:10	23.0	BUILDING_AK01.2197_TK01_TE45_M	12589	200_TK01
01-01-2021 02:10	0.0	BUILDING_AK01.2197_TK01_PF01_EC_FI	12579	200_TK01
01-01-2021 02:10	97.5	TK01_TULOPAINE_SP_FHP	12560	200_TK01
01-01-2021 02:10	10.599	TK01_LTO_SP_FHP	12566	200_TK01
01-01-2021 02:10	0.0	BUILDING_AK01.2197_TK01_PE30_M	12562	200_TK01
01-01-2021 02:10	0.0	BUILDING_AK01.2197_TK01_PE10_M	12559	200_TK01
01-01-2021 02:10	0.0	BUILDING_AK01.2197_TK01_FI	12577	200_TK01
01-01-2021 02:10	125.0	BUILDING_AK01.2197_TK01_PE30_C	12561	200_TK01

This is just the ten first rows of one data process. This process has 2 944 659 rows of data. The data from this process is filtered according to “200_TK01” and a timestamp from “01-01-2021” to “31-08-2022”. Graphically this process can be seen in Figure 18. It is difficult to distinguish specific sensor values in Figure 18. In Figure 19 the data has been zoomed in to show only data for one week. Here it is possible to see that there are many data values per timestamp.

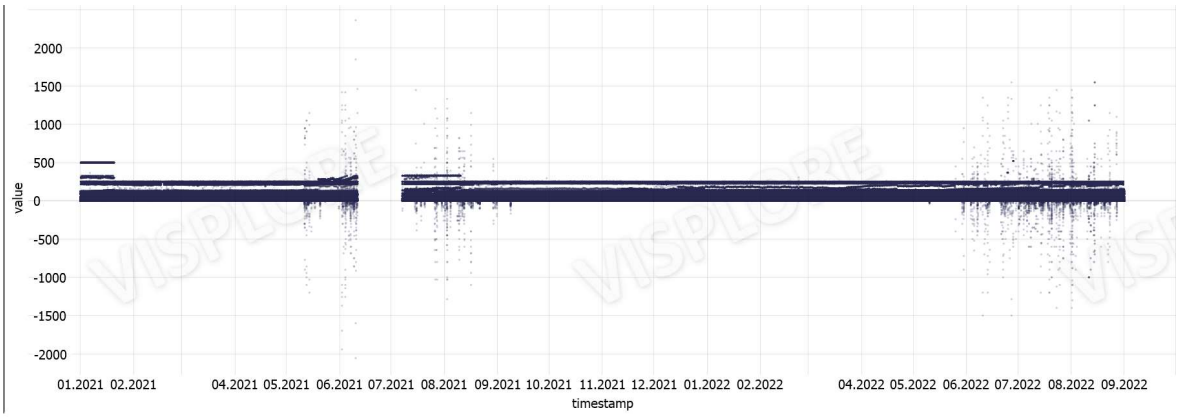


Figure 18. Data graphically

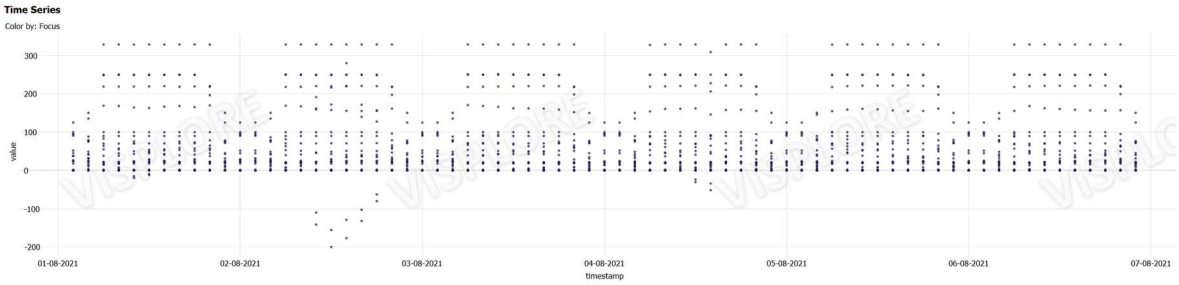


Figure 19. Data graphically zoomed in at one week.

5.2 Labeling

To be able to have some sort of feedback from the anomaly detection methods, the data had to be manually labeled. The data was labeled “Rest” and “Process malfunction”. The labeling was done by a software called Visplore. With Visplore it was possible to import the CSV-process files. In Visplore the data was marked as process malfunction where it was possible to see anomalous data as seen in Figure 20. The data from Visplore was exported with a new column called Label, as seen in Table 2.

Table 2. New column "Label".

timestamp	value	Label	tagId	processId
10-05-2021 03:50	1	Rest	12583	200_TK01
10-05-2021 03:50	0	Rest	12586	200_TK01
10-05-2021 03:50	0,962	Rest	12566	200_TK01
10-05-2021 03:50	0	Rest	12573	200_TK01
10-05-2021 03:50	45	Rest	12558	200_TK01
10-05-2021 03:50	0	Rest	12559	200_TK01
10-05-2021 03:50	0	Rest	12574	200_TK01
10-05-2021 04:00	93,985001	Process malfunction	12569	200_TK01
10-05-2021 04:00	100	Process malfunction	12563	200_TK01
10-05-2021 04:00	20,799999	Process malfunction	12565	200_TK01
10-05-2021 04:00	0	Process malfunction	12585	200_TK01

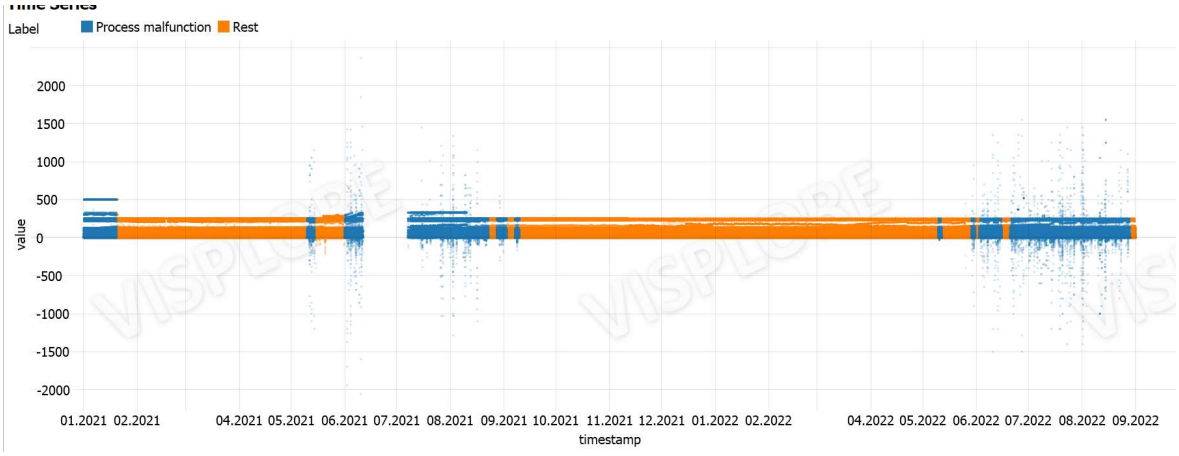


Figure 20. Labeling in Visplore

5.3 Preprocessing

The data from Visplore needed to be preprocessed to fit in the anomaly detection program. The timestamp data had to be set to date/time. The data points were also decreased by taking the mean values over time. The data from the CSV-files was updated every ten minutes. The processed data was down-sampled to a point where no anomalies were lost and so it could analyze at a good speed. The preprocessing was done in a function in Python. The function can be seen in Code example 2.

Code example 2. Function to preprocess data.

```
def preprocess_data(df):
    ''' Function to preprocess the data'''
    # Preprocess the data
    df.value =pd.to_numeric(df.value)
    df['timestamp'] =pd.to_datetime(df['timestamp'])
    df = df.groupby([pd.Grouper(freq='60min',
key='timestamp'),'tagId','Label']).mean()
    df = df.reset_index()
    return df
```

5.4 Pivot dataframe

After the data had been preprocessed, the data frame was pivoted so it would fit different types of anomaly detection methods. To pivot the data frame, the preprocessed data was read into another function. This function reconstructs the dataframes with different measurements from one process in each column and sorted according to the timestamp. The tags become columns and timestamps are used as index. The function can be seen in Code example 3. The differences between the dataframes after preprocessing and pivoting can be seen in Table 3 and Table 4.

Code example 3. Function to Pivot data.

```
def pivotGroupDataframe(df):
    subframe = df[['value', 'timestamp', 'tagId',]]
    # Pivot
    subframe = pd.pivot_table(subframe, index='timestamp', columns='tagId',
values='value', aggfunc=np.mean)
    return subframe
```

Table 3. Data preprocessed.

	timestamp	tagId	Label	value
0	2021-01-01 00:00:00	12558	Process malfunction	45.000000
1	2021-01-01 00:00:00	12559	Process malfunction	0.272727
2	2021-01-01 00:00:00	12560	Process malfunction	96.894000
3	2021-01-01 00:00:00	12561	Process malfunction	125.000000
4	2021-01-01 00:00:00	12562	Process malfunction	0.000000
...	...	...	...	...
248493	2022-09-01 02:00:00	12589	Rest	24.900000
248494	2022-09-01 02:00:00	12590	Rest	0.000000
248495	2022-09-01 02:00:00	12591	Rest	2.900000
248496	2022-09-01 02:00:00	12592	Rest	0.000000
248497	2022-09-01 02:00:00	12593	Rest	0.000000

Table 4. Data pivoted.

tagId	12558	12559	12560	12561	12562	\
timestamp						
2021-01-01 00:00:00	45.0	0.272727	96.894000	125.000000	0.000000	
2021-01-01 02:00:00	45.0	0.666667	96.018667	125.000000	0.000000	
2021-01-01 04:00:00	60.0	17.250000	74.189833	166.666667	93.500000	
2021-01-01 06:00:00	90.0	39.583333	53.518917	250.000000	249.750000	
2021-01-01 08:00:00	90.0	39.833333	53.241167	250.000000	250.333333	
...	...	...	...	...	...	
2022-08-31 18:00:00	45.0	44.916667	0.000000	125.000000	124.666667	
2022-08-31 20:00:00	45.0	44.833333	0.000000	125.000000	123.166667	
2022-08-31 22:00:00	45.0	7.500000	80.879667	125.000000	20.666667	
2022-09-01 00:00:00	45.0	0.666667	96.018667	125.000000	0.000000	
2022-09-01 02:00:00	45.0	1.000000	95.278000	125.000000	0.000000	

For example, in Tables 3 and 4: tagId 12559 from the process is the measured value from the supply air duct pressure sensor.

With the tagIds sorted column vise and sorted according to timestamp, I had to check for infinity and NaN values. These were replaced with 0, as seen in Code example 4.

Code example 4. NaN values replaced.

```
        # Replace infinite and nan values with 0
        df.replace([np.inf, -np.inf],np.nan, inplace=True)
        df.fillna(0,inplace=True)
```

5.5 Different types of anomaly detection methods

For every process and anomaly detection method, the same preprocessing and pivoting were used.

5.5.1 LSTM

This was the first method I used. This method had been used previously for anomaly detection at Fidelix. I had to change the code a bit so it would fit my type of data. The anomaly detection model previously made was analyzing all processes and not just one process at a time.

For the LSTM method, the Keras library is used. In Keras, there are a few different types of autoencoder models. For timeseries data, the LSTM autoencoder model should work quite well.

The LSTM autoencoder code was already made when I started this thesis work. The type of activation function, number of layers and layer settings have been chosen since before.

5.5.2 K-means

This was the first other method I tried to use on the dataset. Here the same preprocessing and pivoting were used as in the LSTM method. With the K-means method the StandardScaler from sklearn.preprocessing was used. The formula for this is:

$$z = (x - u)/s$$

The value of the sample x is calculated from this formula with u is the mean value of the dataset and s is the standard deviation. PCA is also applied on the scaled dataset, as seen in Code example 5.

Code example 5. Standardize/scaling of data.

```
        # Standardize/scale the dataset and apply PCA
        pca = PCA(n_components=2)
        names = df.columns
        x = df[names]
        pipeline = make_pipeline(StandardScaler(), pca)
        pipeline.fit(x)

        # Calculate PCA with 2 components
        principalComponents = pca.fit_transform(x)
        principalDf = pd.DataFrame(data = principalComponents, columns =
['pc1', 'pc2'])
        df['pc1']=pd.Series(principalDf['pc1'].values, index=df.index)
        df['pc2']=pd.Series(principalDf['pc2'].values, index=df.index)
        df.head()
```

The Kmeans method that is used is found in the tslearn clustering library and is called TimeSeriesKMeans. In Code example 6 it is assumed that 20% of the entire dataset are outliers. The program then calculates the distances from the Kmeans centroid to the values of principalDf. Then the program calculates the number of outliers from the assumed percentage of outliers. With these variables, a minimum value is calculated and all the distanced are then compared against this variable. In the example below this variable is called threshold. If any of the distances are above or equal to the threshold they are marked as an anomaly. In this example, the threshold had a value of 514.

Code example 6. Kmeans clustering.

```
        # K-means clustering
        kmeans = TimeSeriesKMeans(n_clusters=3,
                                  max_iter=50, random_state=42)
        kmeans.fit(principalDf.values)
        # Assume that 15% of the entire data set are anomalies
        outliers_fraction = 0.15
        # get the distance between each point and its nearest centroid. The
biggest distances are considered as anomaly
        distance = getDistanceByPoint(principalDf, kmeans)
        # number of observations that equate to 15% of the entire data set
        number_of_outliers = int(outliers_fraction*len(distance))
        # Take the minimum of the largest 15% of the distances as the
threshold
        threshold = distance.nlargest(number_of_outliers).min()
        print(threshold)
        # anomaly1 contains the anomaly result of the above method Cluster
(0:normal, 1:anomaly)
        principalDf['anomaly1'] = (distance >= threshold).astype(int)
```

5.5.3 Isolation forest

The Isolation Forest method was also used from the tslearn library. It used the same scaler and PCA as in the Kmeans method. In the Isolation Forest method, it is also assumed the percentage of the data that contained anomalies as seen in Code example 7. The Isolation Forest method then predicts if a data point is an anomaly or not.

Code example 7. Isolation forest.

```
        # Isolation forest
        # Assume that 5% of the entire data set are anomalies
        outliers_fraction = 0.05
        model =  IsolationForest(contamination=outliers_fraction)
        model.fit(principalDf.values)
        principalDf['anomaly2'] =
pd.Series(model.predict(principalDf.values))
```

5.5.4 Pycaret

The Pycaret library was used to try different types of methods easily and with simplicity. Different types of anomaly detection methods were tried with Pycaret for example: clustering, isolation forest, knn and SVM. With Pycaret it is also possible to use different types of data normalizing.

With the Pycaret method, the same preprocessing and pivoting of the data were used as in the other methods. Instead of using a separate scaling library the ones built-in in Pycaret were used. In Code example 8, “K nearest neighbor” has been used in the Pycaret setup with z-score as the normalizing method and knn set with a fraction of 0,025. Normalize method z-score is the same as the standard scaler.

Code example 8. Pycaret setup.

```
        setup(df, normalize = True, normalize_method = 'zscore')

        # train model
        knn = create_model('knn', fraction = 0.025)
        knn_results = assign_model(knn)
        knn_results.head()
```

5.5.5 Supervised learning

A Supervised Learning method was also used on the dataset. This was a bit different from the previous methods. The processed dataset was used instead of the pivoted dataset as in the other methods. This meant that the data used in the supervised learning looked like the data in Table 5.

Table 5. Data supervised learning.

	timestamp	tagId	Label	value
0	2021-01-01 00:00:00	12630	Rest	220.000000
1	2021-01-01 00:00:00	12631	Rest	220.545455
2	2021-01-01 00:00:00	12632	Rest	0.000000
3	2021-01-01 00:00:00	12633	Rest	300.000000
4	2021-01-01 00:00:00	12634	Rest	300.636364
...	...	...	...	...
249030	2022-09-01 02:00:00	12661	Process malfunction	23.400000
249031	2022-09-01 02:00:00	12662	Process malfunction	8.200000
249032	2022-09-01 02:00:00	12663	Process malfunction	3.800000
249033	2022-09-01 02:00:00	12664	Process malfunction	0.200000
249034	2022-09-01 02:00:00	12665	Process malfunction	0.210000

The train_test_split method was used from, this method can be found in the sklearn library for Python. The train_test_split method uses features and labels as inputs. Timestamp features were extracted and saved under different columns, as seen in Code example 9. This was done because the supervised learning method that was used couldn't handle the datetime format. The label columns were also converted from Rest and Process malfunction to 0 and 1. With the data then sorted correctly for the supervised learning method, the data were split into training and test data. In this case, 20% of the data were set as test and 80% as trained, as seen in Code example 10.

Code example 9. Preprocess data for supervised learning.

```
        # Preprocess the data
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        label_encoder = LabelEncoder()
        df['label_encoded'] = label_encoder.fit_transform(df['Label'])

        # Extract timestamp features
        df['year'] = df['timestamp'].dt.year
        df['month'] = df['timestamp'].dt.month
        df['day'] = df['timestamp'].dt.day
        df['hour'] = df['timestamp'].dt.hour
        df['minute'] = df['timestamp'].dt.minute


        # Define features (X) and target (y)
        X = df[['year', 'month', 'day', 'hour', 'minute', 'value']]
        y = df['label_encoded']
```

Code example 10. Train_test_split function.

```
        # Split the data into training and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

Then the data is run through a RandomForestClassifier, where it predicts the outcome of the test data, the code for RandomForestClassifier can be seen in Code example 11. The data is then sorted so it is possible to compare the classifier predicted with the manually labeled data.

Code example 11. Random forest classifier.

```
# Train a Random Forest classifier
        clf = RandomForestClassifier(n_estimators=10)
        clf.fit(X_train, y_train)

        # Make predictions on the test data
        y_pred = clf.predict(X_test)

        # Create a DataFrame for test data with 'Label' and 'y_pred' columns
        test_df = df.iloc[X_test.index].copy()
        test_df['y_pred'] = y_pred

        # Sort the DataFrame by 'timestamp'
        test_df.sort_values(by='timestamp', inplace=True)
```

5.6 Comparing methods and method settings

The same type of comparing was used in all of the methods above. The predicted/calculated anomalies were added to the original dataframe and then compared column-wise to see if they were equal. For example, as seen in Figure 36, a column that states true or false depends if the column “Label” is “Rest” or “Process malfunction” was made. The same was done with the column “y_pred” but here if the value is “0” or “1” it equals true or false, as seen in Code example 12. Then these two columns are compared and if they are equal it will give a result true, as seen in Table 6.

Code example 12. Check for anomalies.

```
        # Add another column that checks if the Label value is Process
malfunction
        test_df['result'] = test_df['Label'].apply(lambda x: True if x ==
'Process malfunction' else False)
        # Add another column that checks if anomaly1 value is 1 = Anomaly
        test_df['Anomalie1'] = test_df['y_pred'].apply(lambda x: True if x
== 0 else False)
        test_df['checked'] =
np.where((test_df['Anomalie1']==test_df['result']),True,False)
```

Table 6. Checked for anomalies.

	timestamp	tagId	Label	value	label_encoded	year	...	hour	minute	y_pred	result	Anomalie1	checked
21	2021-01-01 00:00:00	12651	Rest	1.000000	1	2021	...	0	0	1	False	False	True
22	2021-01-01 00:00:00	12652	Rest	51.781818	1	2021	...	0	0	1	False	False	True
15	2021-01-01 00:00:00	12645	Rest	100.000000	1	2021	...	0	0	1	False	False	True
26	2021-01-01 00:00:00	12656	Rest	157.000000	1	2021	...	0	0	1	False	False	True
33	2021-01-01 00:00:00	12663	Rest	3.754545	1	2021	...	0	0	1	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...
249020	2022-09-01 02:00:00	12651	Process malfunction	0.000000	0	2022	...	2	0	0	True	True	True
249021	2022-09-01 02:00:00	12652	Process malfunction	0.000000	0	2022	...	2	0	0	True	True	True
249031	2022-09-01 02:00:00	12662	Process malfunction	8.200000	0	2022	...	2	0	0	True	True	True
249025	2022-09-01 02:00:00	12656	Process malfunction	157.000000	0	2022	...	2	0	0	True	True	True
249012	2022-09-01 02:00:00	12643	Process malfunction	0.000000	0	2022	...	2	0	0	True	True	True

Accuracy, recall and precision were used to compare the results. Graphs were printed to compare the original labeled anomaly data with the calculated/predicted anomalies. Accuracy is calculated by comparing the length of how many values equal True in the “Checked” column with the length of column “result”. Recall and precision are calculated with sklearn.recall_score and sklearn.precision_score. The code for results can be found in Code example 13. The formulas for recall and precision:

$$Accuracy = \frac{Number\ of\ datapoints\ correct\ predicted}{Number\ of\ datapoints\ in\ dataset}$$

$$Precision = \frac{True\ positives}{True\ positives + false\ positives}$$

$$Recall = \frac{True\ positives}{True\ positives + false\ negatives}$$

Code example 13. Result calculations.

```
        correct = test_df.loc[test_df['checked']==True]
        Accuracy = 100*(len(correct)/len(test_df['result']))
        recall = 100.0 *
sklearn.metrics.recall_score(test_df['Anomalie1'].to_numpy(),
test_df['result'].to_numpy())
        precision = 100.0 *
sklearn.metrics.precision_score(test_df['Anomalie1'].to_numpy(),
test_df['result'].to_numpy())

      # Plot the data
        Fig.axs = plt.subplots(2)
        fig.suptitle('Anomaly Detection')
        axs[0].plot(Rest['timestamp'],Rest['value'])
        axs[0].plot(Malfunction['timestamp'],Malfunction['value'])
        axs[1].plot(test_df['timestamp'],test_df['Anomalie1'])
        pyplot.show()
```

5.7 Running the code on Databricks

All of the methods described above have only predicted one process at a time, depending on the process selected to read as csv. All of these methods were copied to run on Databricks as well. The labeled data was uploaded to Azure explorer and from here it was accessible from Datalake. The code on Databricks was made to run the same method through all processes on the same execution. The results from each process were saved to a CSV-file and a graph comparing labeled and predicted/calculated data was saved from each process. The header from the CSV-file can be seen in Table 7.

Table 7. Results from Databricks header.

Column1	Process	Accuracy	Recall	Precision

6 Results

In this chapter results from the different methods are presented. It will also compare different settings for a few different methods. For the three first chapters when comparing point grouping, preprocessing and threshold, LSTM and Pycarets KNN methods are compared to show how the results differ. In the fourth chapter, all methods are compared. At the end of the result chapter, a conclusion on how the methods compared to each other when analyzing all 36 processes at the same execution and also how long each method took.

As seen in all of the tables below, recall will have a much better result than precision. This is because the predicted results have more false positives than false negatives compared to the manually labeled data. As the number of false positives increases, precision gets worse and as the false negatives decrease the recall result gets better.

6.1 Process point grouping

The original data had a sample time of 10 minutes. A few different sample times were used to compare the results, these were: 10 minutes as the original data, 30 min, 60 min and 120 min. As seen in Table 8, accuracy is almost the same, recall gets worse with LSTM when putting a longer sample time. With Pycaret the results are much more stable. The shorter the sample time the slower the method becomes.

Table 8. Comparing sample time.

Method	Sample time	Accuracy	Recall	Precision
LSTM	10 min	78,96 %	49,51 %	2,35 %
Pycaret KNN	10 min	81,15 %	93,56 %	11,13 %
LSTM	30 min	78,69 %	40,29 %	1,92 %
Pycaret KNN	30 min	81,05 %	93,06 %	11,06 %
LSTM	60 min	79,40 %	79,86 %	3,78 %
Pycaret KNN	60 min	80,97 %	93,10 %	11,04 %
LSTM	120 min	78,30 %	30,00 %	14,10 %
Pycaret KNN	120 min	80,73 %	90,29 %	10,67 %

6.2 Preprocessing/data scaling

For LSTM, MinMaxScaler and StandarScaler were compared, both from sklearn. With the Pycaret KNN, z-score, minmax and robust were compared. With LSTM, the scaling method didn't seem to affect the results. Pycarets KNN works best without any scaling method when predicting this specific model, as seen in Table 9. The method also becomes slower when a scaling method is applied.

Table 9. Comparing scaling methods.

Method	Scaling method	Accuracy	Recall	Precision
LSTM	MinMaxScaler	79,40 %	79,86 %	3,78 %
LSTM	StandarScaler	79,45 %	82,14 %	3,92 %
Pycaret KNN	z-score	80,02 %	83,67 %	7,15 %
Pycaret KNN	minmax	78,99 %	55,35 %	4,73 %
Pycaret KNN	robust	80,22 %	88,69 %	7,69 %
Pycaret KNN	without	80,97 %	93,10 %	11,04 %

6.3 Method settings

With LSTM the settings that could be changed were the number of epochs and batch size. Batch size doesn't affect the results, it is more a setting for computer processing. With Pycaret KNN the setting that could be changed was the fraction. For LSTM the results don't change that much with the number of epochs, as seen in the result in Table 10. When analyzing this process the LSTM doesn't run more than 56 epochs, this is because the LSTM function stops looping when the loss from the function goes under a specific set value.

Table 10. Comparing LSTM settings.

Method	Number epochs	Batch size	Accuracy	Recall	Precision
LSTM	25	10	78,91 %	55,40 %	2,62 %
LSTM	50	10	78,94 %	57,14 %	2,72 %
LSTM	100	10	79,40 %	79,86 %	3,78 %
LSTM	150	10	79,40 %	79,86 %	3,78 %

Table 11. Comparing Pycaret settings.

Method	Fraction	Accuracy	Recall	Precision
Pycaret KNN	0,025	80,97 %	93,10 %	11,04 %
Pycaret KNN	0,05	82,74 %	89,22 %	21,15 %
Pycaret KNN	0,1	86,30 %	87,39 %	41,31 %

When changing the setting for pycaret as seen in Table 11, It looks like the results are getting better the higher the fraction is but when looking at the graph, one can see the results get more sensitive. The first graph has with fraction size of 0.1 and the second with 0.025. In Figure 21 the graphs show the result with a fraction size of 0.1 and in Figure 22 with a fraction size of 0.025. In both figures, the manually labeled results are presented in the graph above and the predicted results are presented in the graph below. In the graph above, the data in blue are the data that has been labeled manually as “Rest” and the data in green is “Process malfunction”.

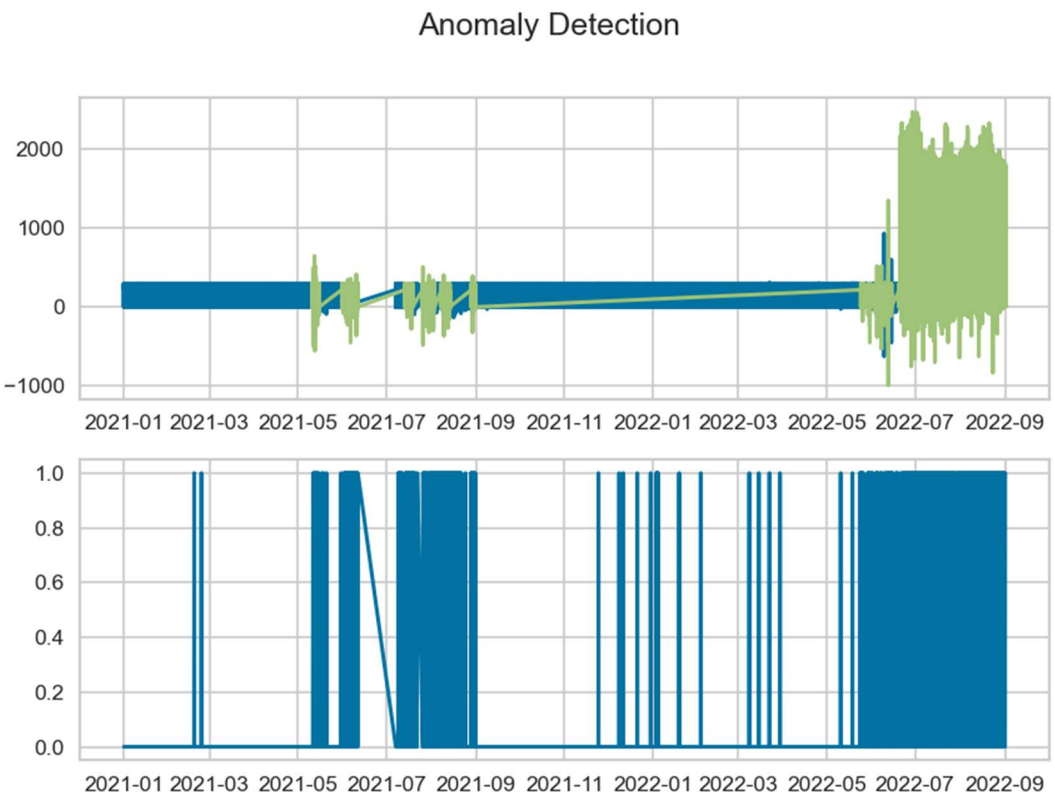


Figure 21. Pycaret KNN with fraction size 0.1

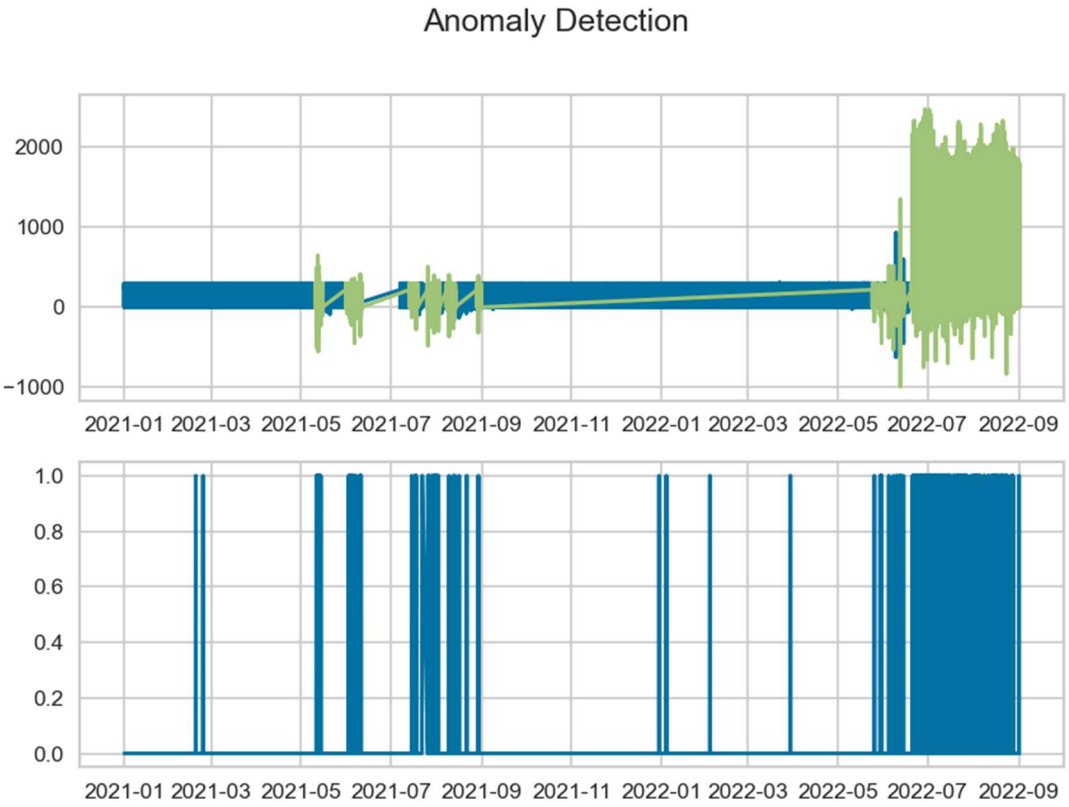


Figure 22. Pycaret KNN with fraction size 0.025

6.4 Comparing machine learning models

The supervised method gave the best result on the specific process that was used in all the previously mentioned examples. A supervised method could be used in these sorts of anomaly detection methods, but somebody has to manually give the model train and test data.

When comparing different unsupervised methods, Pycaret gave very good results. On the specific process shown in Table 12, the SVM- and cluster method gave the best results. The graphic results for every process can be found under attachment 1.

Table 12. Comparing machine learning models.

Method		Accuracy	Recall	Precision
LSTM		79,40 %	79,86 %	3,78 %
Pycaret KNN		80,97 %	93,10 %	11,04 %
Pycaret Iforest		80,85 %	91,04 %	10,73 %
Pycaret cluster		81,29 %	99,71 %	11,78 %
Pycaret SVM		81,03 %	94,27 %	11,21 %
Kmeans		78,99 %	59,42 %	2,79 %
Iforest		79,05 %	62,69 %	2,86 %
Randomforest	Supervised	99,79 %	99,50 %	99,53 %

6.5 Summary of results

When analyzing all manually labeled processes, Pycarets KNN worked the best. It was also the fastest method. Comparing the time it took to run through all processes, Pycaret took three minutes, LSTM 34 minutes and supervised learning one and a half hours. With different fraction sizes and different scaling methods, it was concluded that fraction size 0,025 and no scaling provided the desired result. As seen in the results presented in Table 12, the precision is quite low while the accuracy and recall are higher. It was possible to get better precision but this affected recall. The accuracy is done first by checking if the labeled data equals the predicted data, this is done on every timestamp. Another column is called “checked”. The “checked” column has the values true and false, true when the predicted result and labeled data are equal and false when they are not equal. The program then calculates how many rows that are “True” and this value is divided by the number of data points in the dataset, the column headers can be seen in Table 13. Table 14 shows the mean value from all processes with different prediction methods and settings.

Table 13. Column headers accuracy calculation

ManLabeled	Predicted	checked
False	False	True
False	False	True
False	False	True
False	False	True
False	False	True
...	...	...
False	True	False

Table 14. Comparing mean values from machine learning models.

Method	Accuracy	Recall	Precision
Pycaret KNN (F=0,025, no scaling)	84,68 %	69,98 %	11,06 %
Pycaret KNN (F=0,1, Robust)	83,87 %	50,94 %	29,48 %
Pycaret KNN (F=0,15, no scaling)	83,84 %	50,53 %	44,49 %
K-Means (F=0,0082, PCA)	83,44 %	1,63 %	35,36 %
LSTM (n_epochs = 100)	83,73 %	3,20 %	45,00 %
Supervised learning	100,00 %	97,29 %	97,29 %

Graphic results for a few of these processes can be found under attachment 1 and results for each process can be found under attachment 2.

7 Conclusion

The goal of this thesis was to analyze the anomaly detection model that is used in Fidelix Flow_how and to see how it affects results depending on the models settings. The model currently used is LSTM. Another goal of this thesis was to compare the LSTM model to other anomaly detection methods and to see if other methods will perform better on this type of data.

As concluded in the results chapter, changing the scaling method or the settings of the LSTM method does not improve the results. It is possible to make the LSTM method faster by reducing the sampling time while still maintaining a good result.

When comparing the LSTM method to other anomaly detection methods, it can be concluded that other methods give better results than the LSTM method. Other methods seem to be more accurate and precise in finding anomalies in the data.

Some other methods researched in this thesis, for example, Pycaret, are also faster than the LSTM method. Currently, there is no need to make the LSTM model faster, so having a slow but accurate LSTM method is not a problem that needs solving. In the LSTM model, the data is gathered once a week and the model runs at nighttime. If the model would have to run once or even twice a day, another faster anomaly detection model would be preferred.

Comparing recall and precision results with Pycarets' different fraction sizes. With a higher fraction, the recall gets lower but precision is higher and with a lower fraction the other way around. This can be explained by the fact that with a higher fraction size, there are fewer false positives because the model gets more sensitive and will predict more data points to be anomalies. This in turn leads to a better precision result, but with more data points predicted to be anomalies, the more false negatives as well which leads to a worse recall result.

When considering this thesis's results, it is important to note that the manual labeling of the data was necessary for being able to compare the different anomaly detection methods researched. Human error may have occurred when manually labeling the data and that should be noticed when considering the results. There are about two million values on the X-axis of the data and to manually and accurately label such vast quantities of data is impossible.

One possibility to get a more accurate result could have been done by having a shorter dataset when manually labeling the data. For example, to have a time period of a month where there were known anomalies. This would have led to more precise labeling but also a loss in data points.

The LSTM model will continue to run on the Flow_how service. However, at some point in the future, the Flow_how service will be updated. When this happens, it is a possibility that one of the other anomaly detection methods researched in this thesis will be applied instead. Anomaly detection is only a small part of the Flow_how services and for the time being, other updates are prioritized. Because the anomaly detection method was not updated in the Flow_how service, it is not possible to know how the different methods works on live data.

8 References

Ahmed, R. (Producer). (n.d.). *K Nearest Neighbors (KNN) in 10 Minutes (Beginner Friendly)* [Motion Picture].

Alpaydin, E. (2014). *Introduction to machine learning (Third edition.).* Massachusetts: The MIT Press.

Assemblin kasvattaa rakennusautomaatio- ja BMS-osaamistaan saattamalla päätökseen Fidelix-yrityskaupan. (2021, September 24).

Bhandari, A. (2023, October 27). Feature Engineering: Scaling, Normalization, and Standardization.

Bhattacharyya, D., & Kalita, J. (2014). *Network anomaly detection: a machine learning perspective.* CRC Press.

Biswal, A. (2023, November 7). What is Principal Component Analysis?

Brownlee, J. (2020, November 15). A Gentle Introduction to PyCaret for Machine Learning. *Python Machine Learning*.

Brownlee, J. (2020, August 20). A Gentle Introduction to the Rectified Linear Unit (ReLU). *Deep Learning Performance*.

Chandola, V., Banerjee, A., & Kumar, V. (2007). *Anomaly Detection: A Survey.* Minnesota: University of Minnesota Digital Conservancy.

Choubey, V. (2020). Understanding Recurrent Neural Network (RNN) and Long Short Term Memory(LSTM). *Analytics Vidhya*.

Dhiraj, K. (2020). *Anomaly Detection Using Isolation Forest in Python.* Retrieved from PaperspaceBlog: www.blog.paperspace.com

Evans, P. (2018, September 26). Air Handling Units Explained. Retrieved from https://theengineeringmindset.com/air-handling-units-explained/

Fandango, A. (2018). *Mastering Tensorflow 1.x (1st edition).* Packt Publishing.

Fidelix. (2022). Retrieved from Fidelix: https://www.fidelix.fi/

Han, J., Micheline, K., & Pei, J. (2000). *DATA MINING Concepts and Techniques.* Morgan Kaufmann Publishers.

Heydt, M. (2017). *earning Pandas: High-performance Data Manipulation and Analysis in Python. Second edition.* Birmingham: Packt.

Hidden Markov Models, Theory and Applications. (u.d.). InTech.

IBM. (n.d.). *What Is Supervised Learning*. Retrieved from IBM: www.ibm.com

Jolly, K. (2018). *Machine Learning with scikit-learn Quick Start Guide.* Packt Publishing.

Liu, F. T., & Ting, K. M. (2009, January). Isolation Forest. Victoria, Australia.

Mossberg Sonnek, K., & Lindgren, F. (2015). NCS3 - Industriella informations- och styrsystem inom fastighetsautomation.

Omar, S., Ngadi, A., & Jebur, H. (2013). Machine learning techniques for anomaly detection: an overview. *Internation Journal of Computer Applications 79.2*.

Potrimba, P. (2022, December 16). *roboflow*. Retrieved from What is Semi-Supervised Learning? A Guide for Beginners: https://blog.roboflow.com/

R. S. (2022, July 27). What is Clustering and Different Types of Clustering Methods.

Rizwana, K. (2022). An End-to-end Guide on Anomaly Detection. *Analytics Vidhya*.

Saxena, S. (2021). Introduction to Long Short Term Memory (LSTM). *Analytics Vidhya*.

Sericola, B. (2013). *Markov Chains: Theory, alorithms and applications.* London: ISTE.

Serrano, L. (2021). *Grokking Machine Learning.*

Sharma, N. (2023, August 8). K-Means Clustering Explained.

Venus, L. (2020). Heat Pumps for Sustainable Heating and Cooling. *Air Handling Unit*, 51-64.

Visplore. (n.d.). Retrieved Mars 28, 2023, from Visplore: www.visplore.com

Wang, S. (2010). *Intelligent Buildings and Building Automation.* Abingdon: Spon Press.

Webber-Cross, G. (2014). *Learning microsoft azure : a comprehensive guide to cloud application development using microsoft azure.* Packt Publishing.

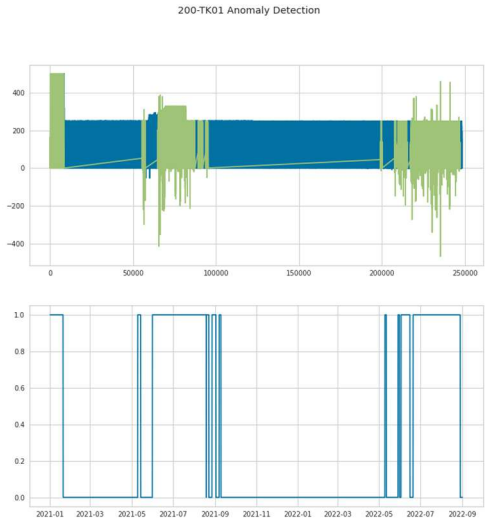
Yehoshua, R. (n.d.). *K-Nearest Neighbors (KNN): A Comprehensive Guide*. Retrieved from medium.com: https://medium.com/ai-made-simple/k-nearest-neighbors-knn-a-comprehensive-guide-7add717806ad

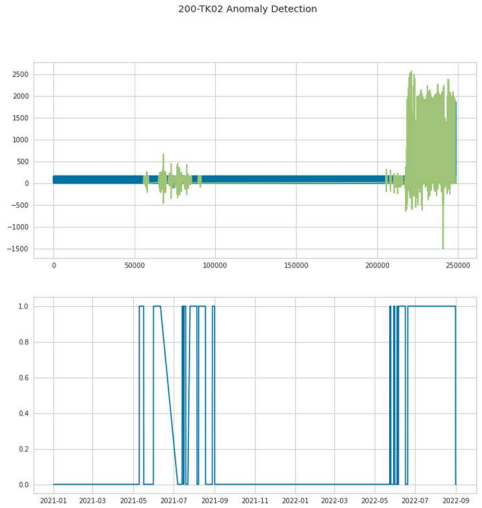
Yim, A., Chung, C., & Yu, A. (2018). *Matplotlib for Python Developers (2nd Edition).* Packt Publishing.

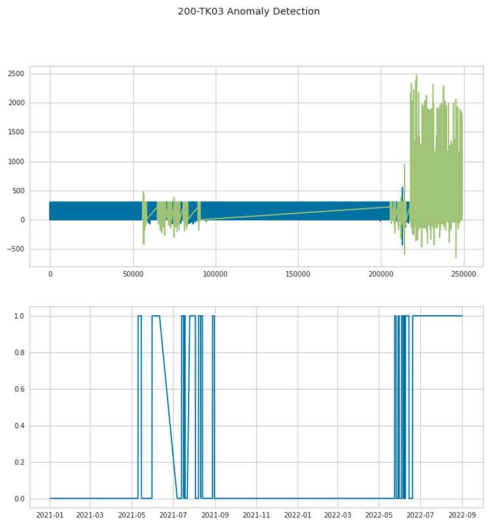
9 Appendix

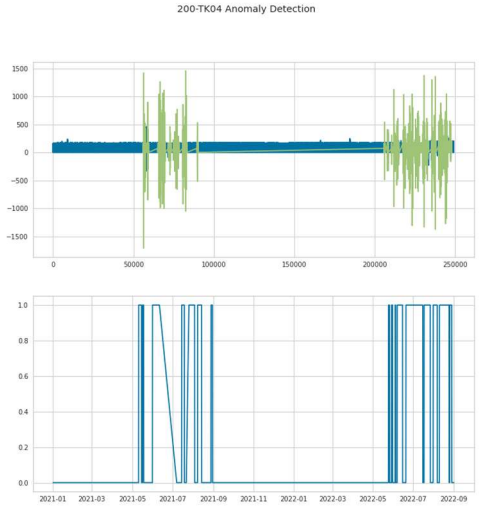
9.1 Appendix 1 (Graph results)

Pycaret KNN Fraction 0,025. Process time 3,14 min. Above manually labeled, below predicted.

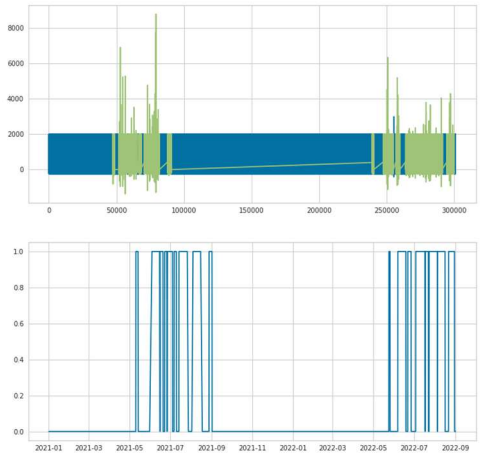




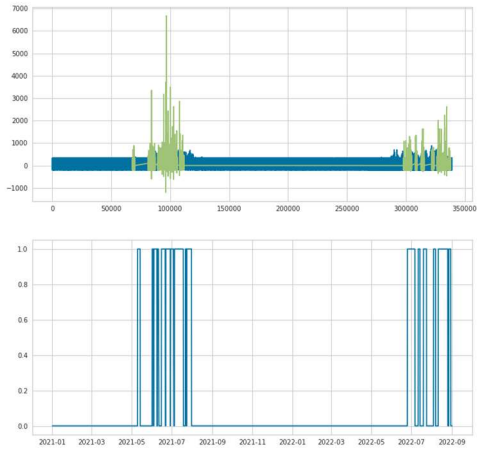




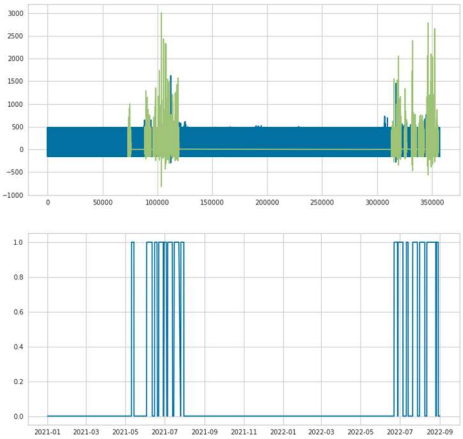
226-K330 Anomaly Detection



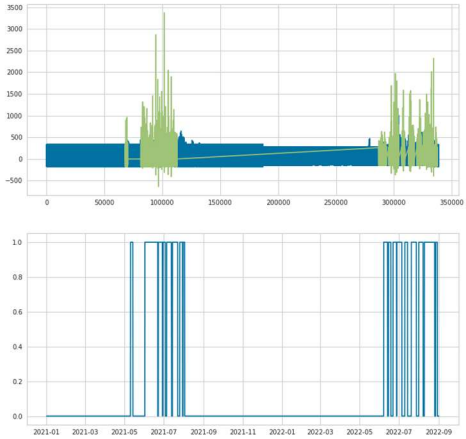
226-K331 Anomaly Detection



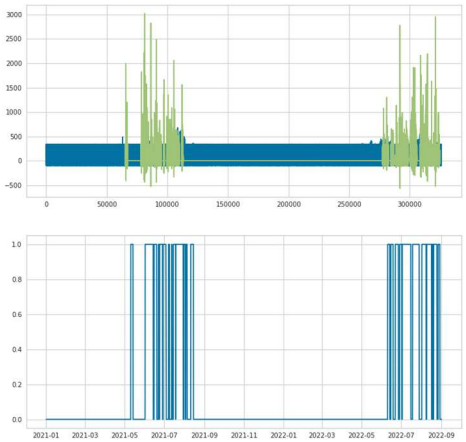
226-K339 Anomaly Detection



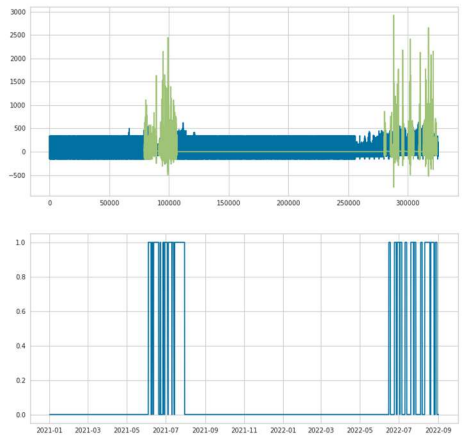
226-K340 Anomaly Detection



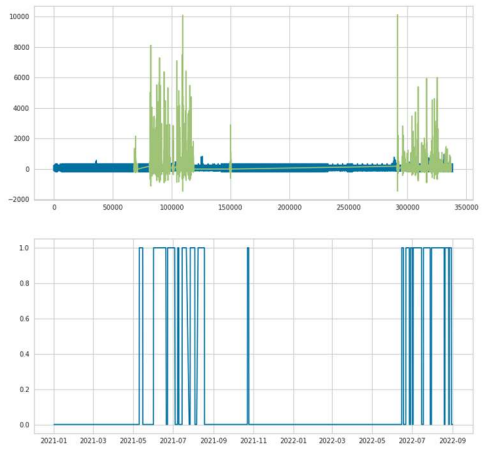
226-K346 Anomaly Detection



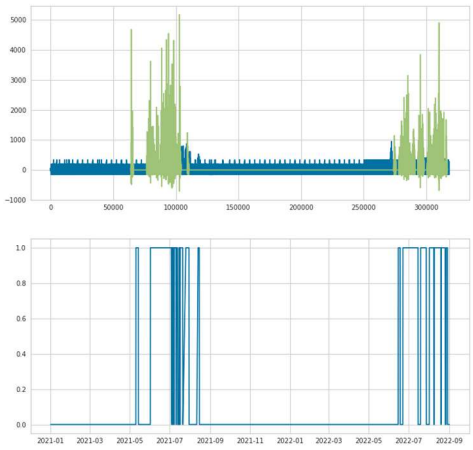
226-K347 Anomaly Detection



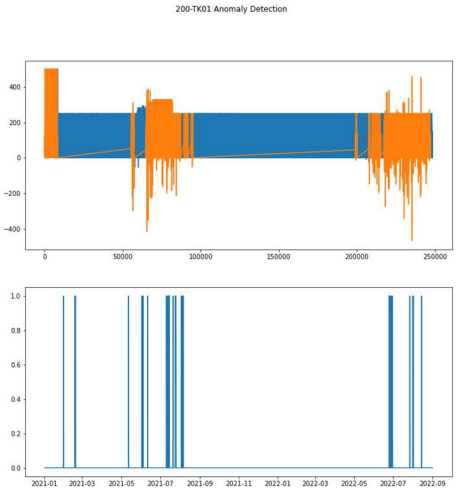
226-T330 Anomaly Detection

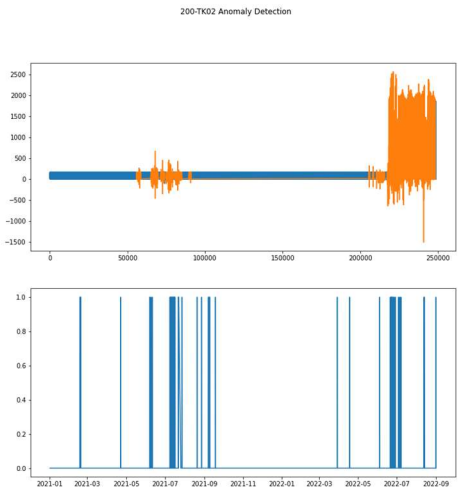


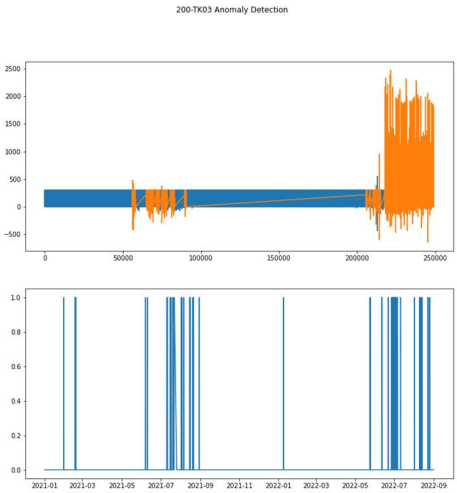
226-T338 Anomaly Detection

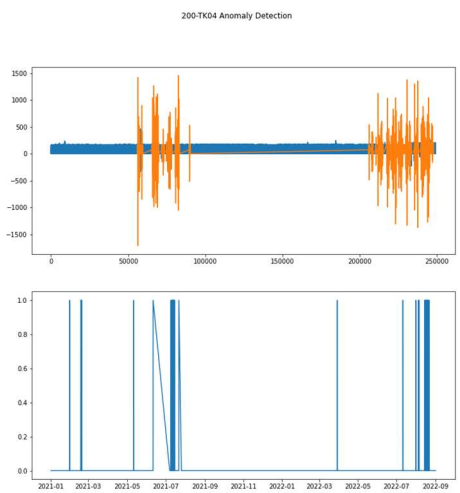


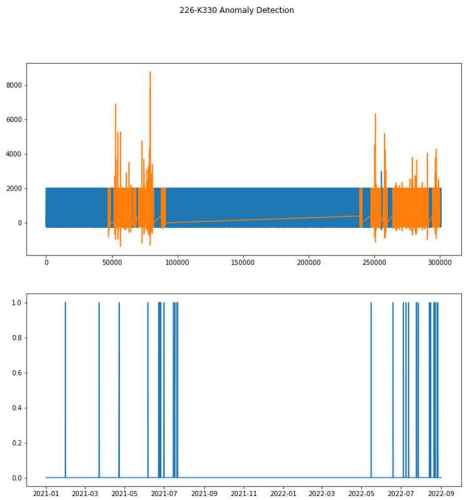
LSTM, number of epochs 100, batch size 10 and minmax scaler:

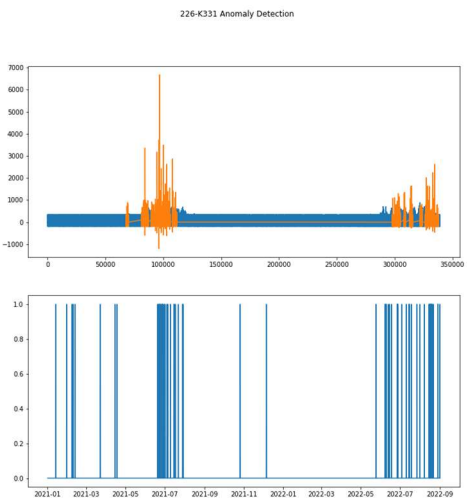




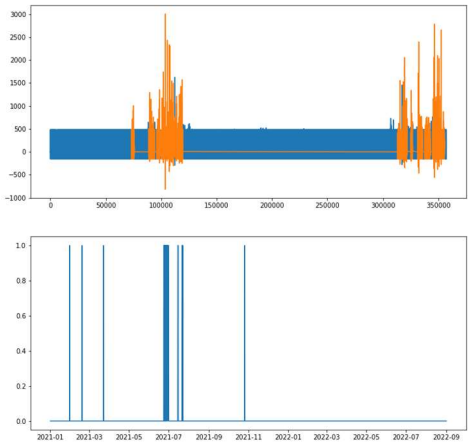




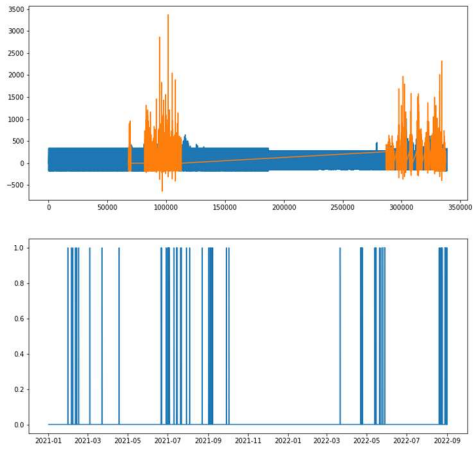




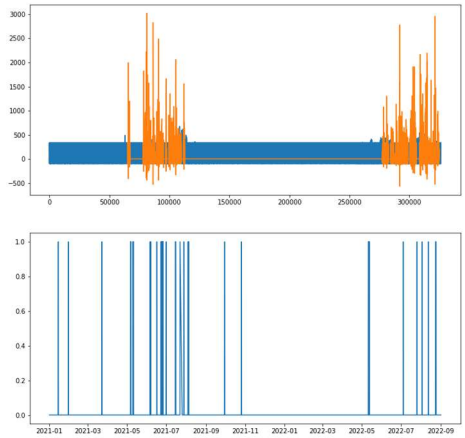
226-K339 Anomaly Detection



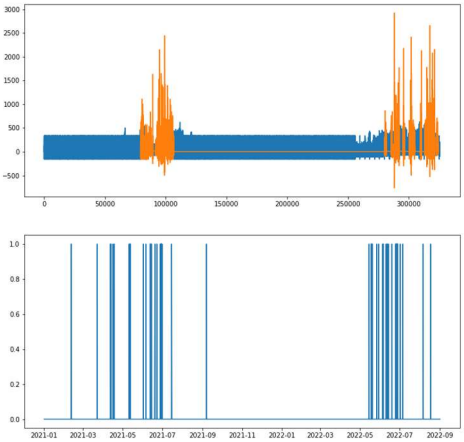
226-K340 Anomaly Detection



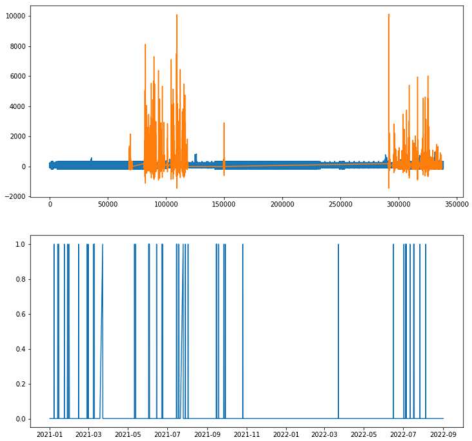
226-K346 Anomaly Detection



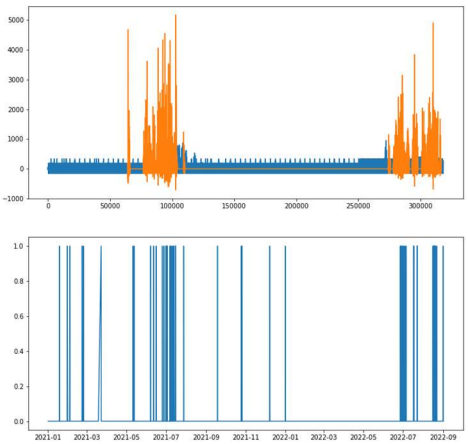
226-K347 Anomaly Detection



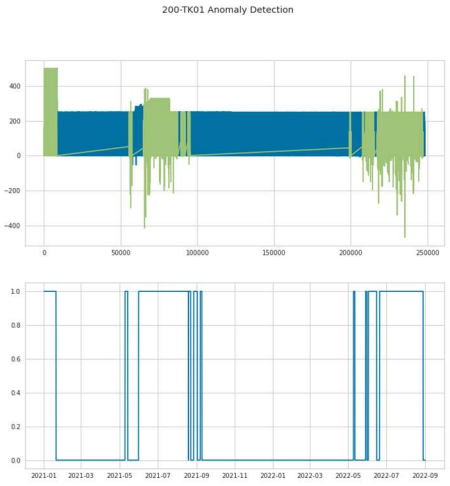
226-T330 Anomaly Detection

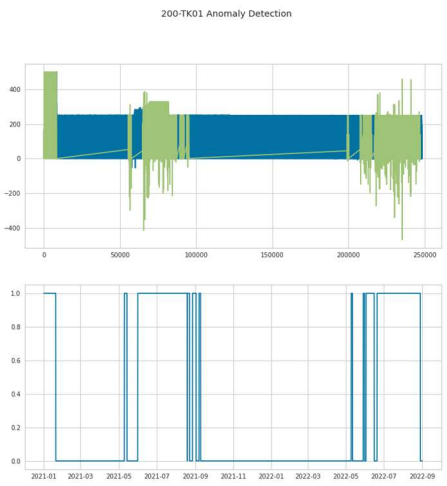


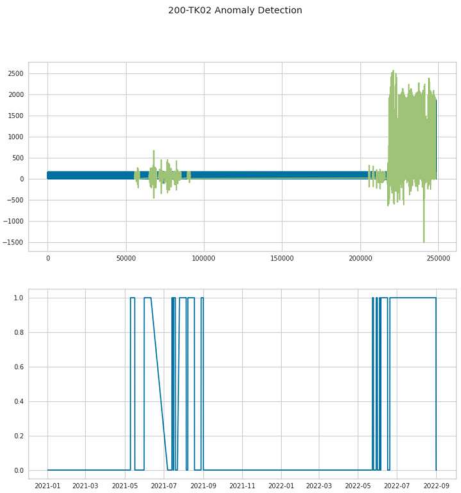
226-T338 Anomaly Detection

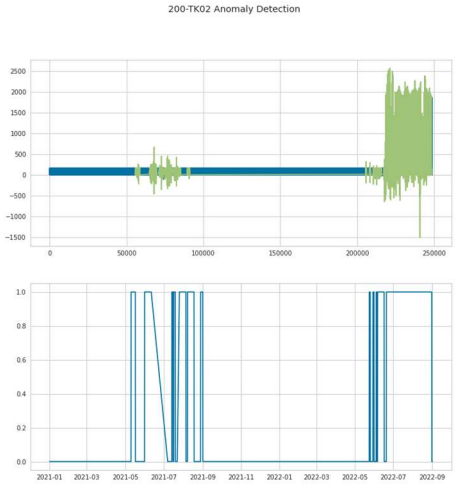


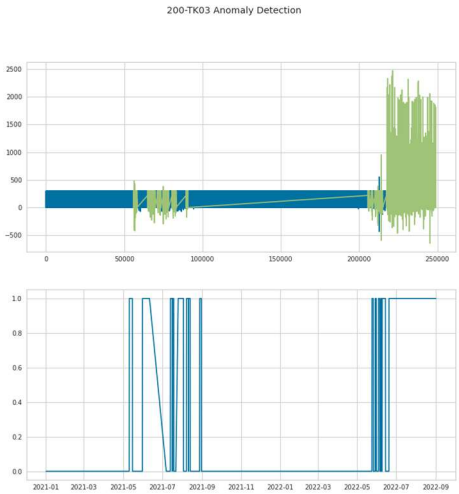
To the left: Pycaret KNN fraction 0.1. Scaling method Robust. To the right: Pycaret KNN fraction 0,15. No scaling.

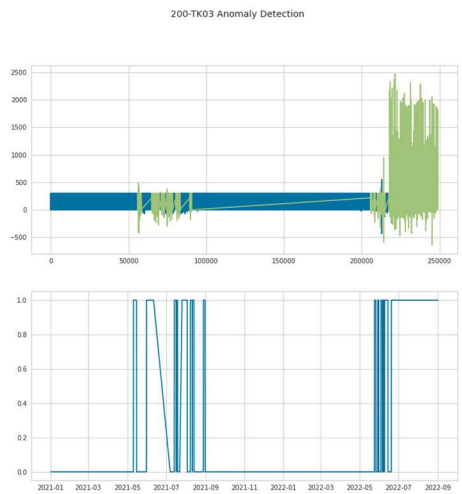




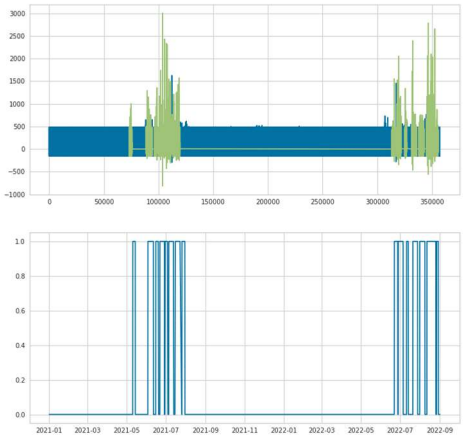




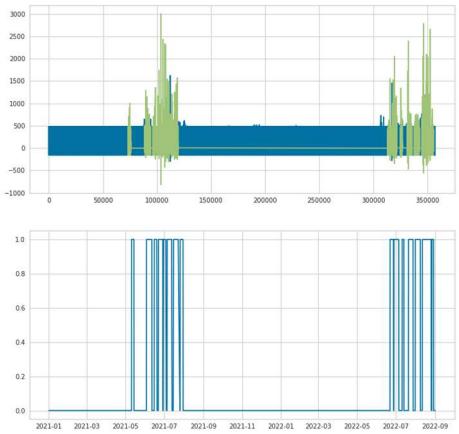




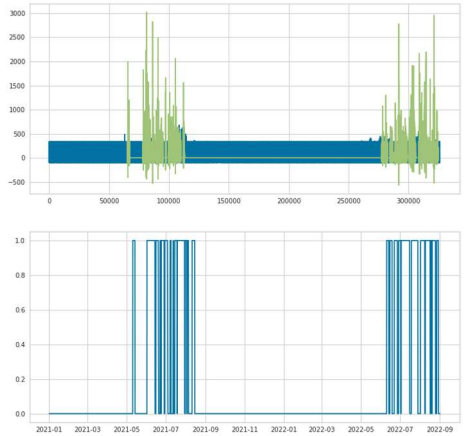
226-K339 Anomaly Detection



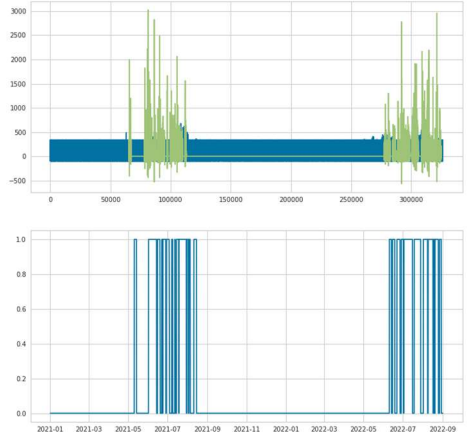
226-K339 Anomaly Detection



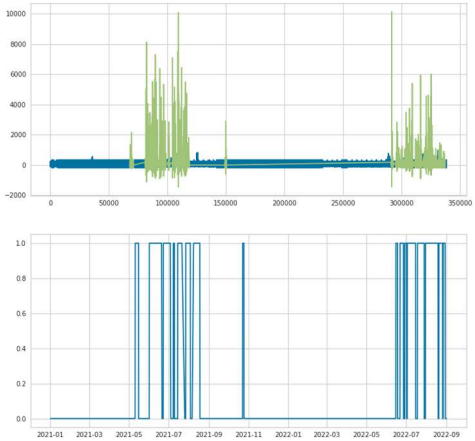
226-K346 Anomaly Detection



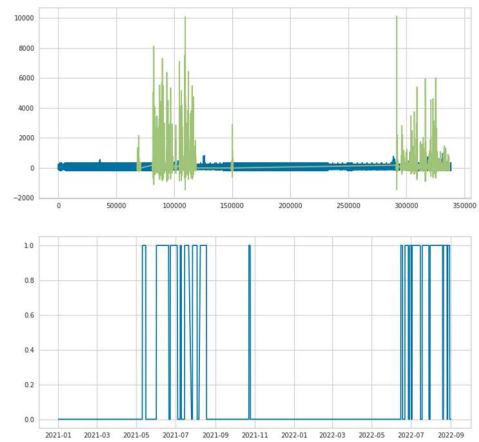
226-K346 Anomaly Detection



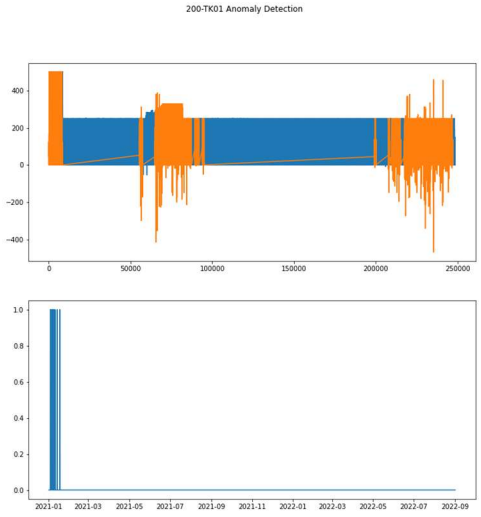
226-T330 Anomaly Detection

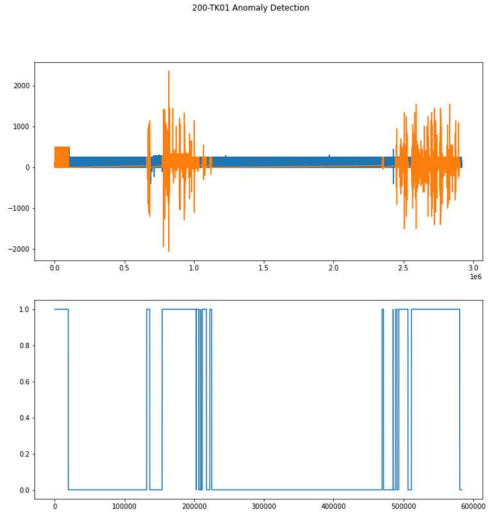


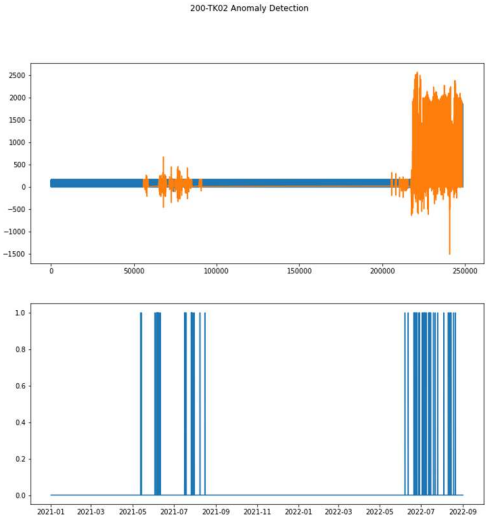
226-T330 Anomaly Detection

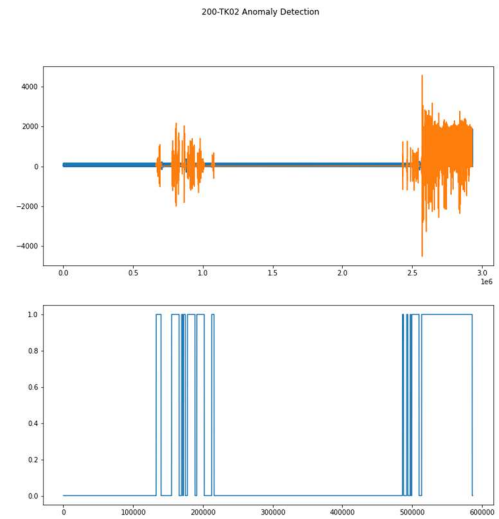


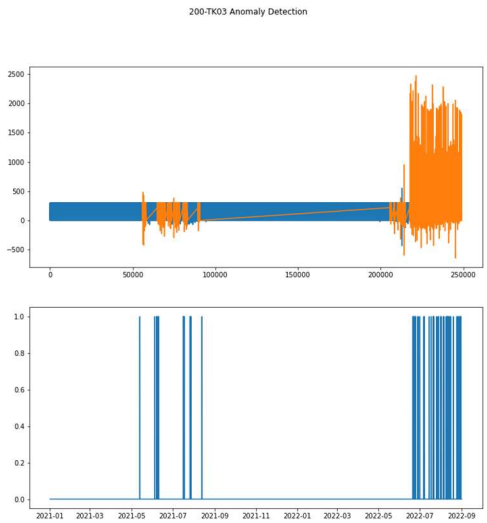
To the left: K-Means, fraction 0,0082 Scaling method PCA. To the right: Supervised learning

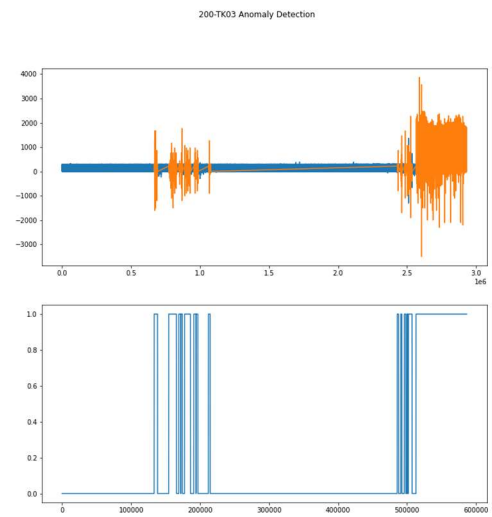




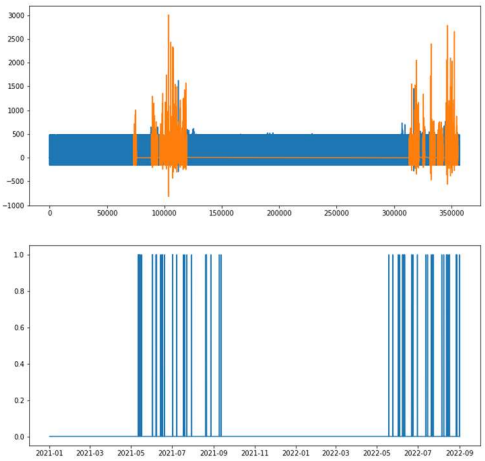




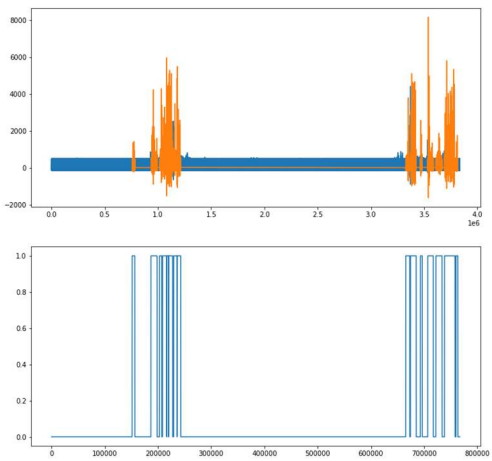




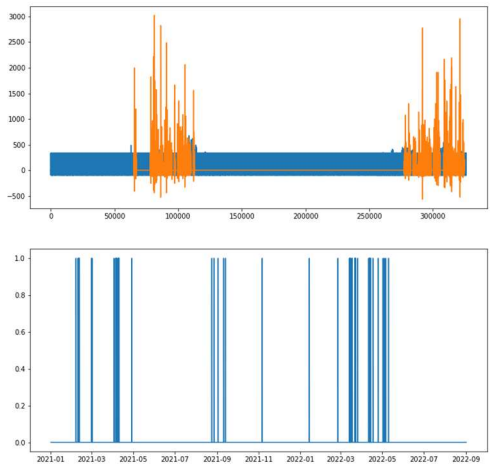
226-K339 Anomaly Detection



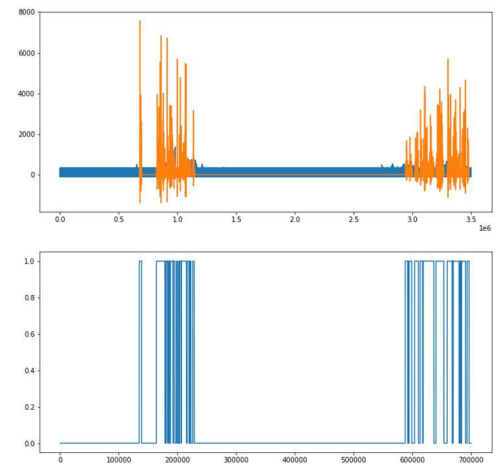
226-K339 Anomaly Detection



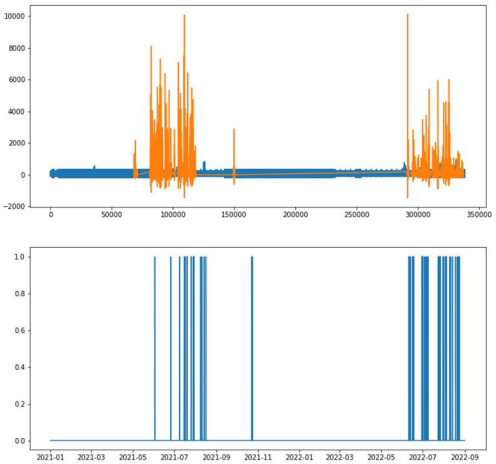
226-K346 Anomaly Detection



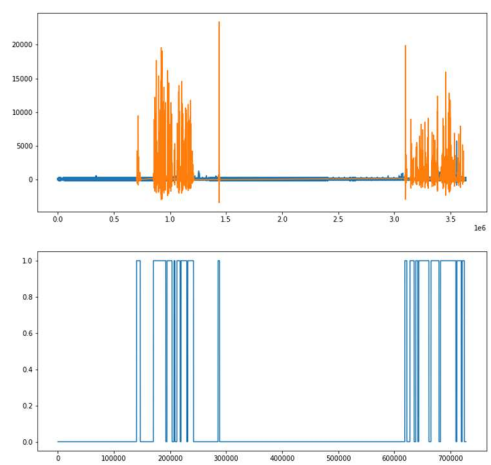
226-K346 Anomaly Detection



226-T330 Anomaly Detection



226-T330 Anomaly Detection



9.2 Appendix 2 (Worksheet results)

Pycaret KNN Fraction 0,025 Time: 3,14 min

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	71,85302139	88	7,35434575
1	200-TK02	79,27603115	97,07602339	10,38798498
2	200-TK03	80,67821068	90,8045977	10,66846725
3	200-TK04	83,29483107	94,25287356	12,50953471
4	226-K330	77,82851344	94,59459459	9,702009702
5	226-K331	86,39476035	83,33333333	13,64522417
6	226-K332	92,08761734	22,02380952	8,466819222
7	226-K339	85,34908058	82,73809524	12,75229358
8	226-K340	81,8426563	86,30952381	10,85329341
9	226-K344	90,96079018	42,85714286	12,4137931
10	226-K346	82,56249065	85,62874251	11,13707165
11	226-K347	87,66282378	80,23952096	14,48648649
12	226-T301	79,32223722	81,43712575	9,164420485
13	226-T302	84,88529015	85,62874251	12,68855368
14	226-T303	82,32118758	88,0239521	11,25574273
15	226-T306	83,29584645	89,82035928	12,02886929
16	226-T313	80,94167042	91,01796407	10,79545455
17	226-T314	90,53831159	74,8502994	17,5070028
18	226-T316	84,07557355	81,43712575	11,6538132
19	226-T319	87,734293	86,22754491	15,33546326
20	226-T320	86,20482831	77,84431138	12,83316881
21	226-T321	85,63502774	88,0239521	13,5483871
22	226-T322	83,41331734	85,62874251	11,67346939
23	226-T323	81,19376125	85,02994012	10,35740336
24	226-T324	85,46790642	79,04191617	12,38273921
25	226-T330	81,10377924	95,20958084	11,26860383
26	226-T338	82,73845231	95,20958084	12,21198157
27	542-TC1055	78,91523499	60,22727273	6,969099277
28	542-TC3044	82,06670312	2,173913043	0,352112676
29	542-TC3045	90,28613085	52,17391304	13,3580705
30	542-TC3046	90,19500638	41,30434783	11,08949416
31	542-TK01	70,88535755	57,62711864	4,906204906
32	542-TK02	97,48864926	0	0
33	542-TK07	96,74992904	6,214689266	14,86486486
34	542-TK10	80,38325053	92,65536723	10,69797782
35	542-TK11	91,92334989	14,12429379	5,656108597
36	542-TK14	95,59971611	30,50847458	22,406639
Mean		**84,67988209**	**69,98104825**	**11,06440457**

Pycaret KNN Fraction 0,1 Normalize-method = Robust Time: 3 min

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	77,80967418	89,36781609	29,70391595
1	200-TK02	85,5494664	92,93948127	40,36295369
2	200-TK03	84,35786436	78,64357864	36,79945982
3	200-TK04	88,18943113	85,56998557	45,23264683
4	226-K330	82,15795704	82,77027027	33,95703396
5	226-K331	86,39476035	58,33333333	38,20662768
6	226-K332	85,63552377	10,73025335	16,47597254
7	226-K339	89,9088055	81,01644245	49,72477064
8	226-K340	79,28507329	46,33781764	23,20359281
9	226-K344	87,77312182	32,28699552	37,24137931
10	226-K346	85,70573267	74,66266867	38,78504673
11	226-K347	89,1600539	65,06746627	46,91891892
12	226-T301	81,03163893	66,4167916	29,85175202
13	226-T302	82,39616134	46,47676162	27,50665484
14	226-T303	83,19088319	63,86806597	32,618683
15	226-T306	85,36512221	70,31484258	37,61026464
16	226-T313	81,78137652	64,46776612	30,53977273
17	226-T314	86,99955016	38,53073463	35,99439776
18	226-T316	80,89668616	41,97901049	23,99314482
19	226-T319	85,44009597	47,5975976	33,75931842
20	226-T320	88,5739991	68,8155922	45,31095755
21	226-T321	89,62363173	79,46026987	48,84792627
22	226-T322	79,33413317	38,53073463	20,97959184
23	226-T323	76,27474505	34,18290855	16,63019694
24	226-T324	79,54409118	27,62762763	17,26078799
25	226-T330	79,73905219	54,5045045	25,72643515
26	226-T338	87,4775045	85,00749625	43,5483871
27	542-TC1055	87,10776658	93,4751773	43,32675871
28	542-TC3044	76,69035903	11,11111111	7,159624413
29	542-TC3045	84,07144159	19,48998179	19,85157699
30	542-TC3046	84,67286313	20,21857923	21,59533074
31	542-TK01	67,87741203	36,87943262	12,50601251
32	542-TK02	90,01135074	0	0
33	542-TK07	89,21373829	1,278409091	12,16216216
34	542-TK10	80,44002839	60,9929078	28,04957599
35	542-TK11	84,74095103	5,106382979	8,14479638
36	542-TK14	88,70120653	10,63829787	31,12033195
Mean		**83,86819603**	**50,93775927**	**29,47856108**

Pycaret KNN Fraction 0,15 Time: 2,87 min

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	79,1732453	80,76555024	40,30563515
1	200-TK02	86,93394866	83,26923077	54,19274093
2	200-TK03	88,2972583	82,21153846	57,73126266
3	200-TK04	85,0996246	63,42637151	50,26697178
4	226-K330	84,77929985	80,60879369	49,54954955
5	226-K331	87,43673712	59,02777778	57,99220273
6	226-K332	81,96990016	11,61866931	26,77345538
7	226-K339	86,90387203	60,65737052	55,87155963
8	226-K340	82,72509722	59,02293121	44,31137725
9	226-K344	84,80993714	28,31505484	48,96551724
10	226-K346	85,39140847	65,36926148	51,01246106
11	226-K347	86,48001198	51,09780439	55,35135135
12	226-T301	84,8702954	73,72627373	49,73045822
13	226-T302	84,4654371	54,54545455	48,44720497
14	226-T303	85,65002249	67,43256743	51,68453292
15	226-T306	85,00524816	62,33766234	50,04009623
16	226-T313	87,3294347	78,12187812	55,53977273
17	226-T314	89,60863698	51,04895105	71,56862745
18	226-T316	85,06522717	58,54145854	50,21422451
19	226-T319	89,27875244	61,2	65,17571885
20	226-T320	86,74463938	56,44355644	55,77492596
21	226-T321	87,47938222	62,5	57,60368664
22	226-T322	84,49310138	59,54045954	48,65306122
23	226-T323	85,78284343	71,12887113	51,9328957
24	226-T324	86,00779844	56,64335664	53,18949343
25	226-T330	83,80323935	66,53346653	47,20056697
26	226-T338	88,31733653	76,12387612	58,52534562
27	542-TC1055	76,57248332	43,88625592	30,44049967
28	542-TC3044	70,23874613	2,55164034	2,464788732
29	542-TC3045	81,62930563	21,50668287	32,83858998
30	542-TC3046	81,10078367	18,22600243	29,18287938
31	542-TK01	67,83484677	41,11531191	20,92352092
32	542-TK02	84,98864926	0	0
33	542-TK07	84,62957707	2,270577105	32,43243243
34	542-TK10	76,0397445	42,6679281	29,41943901
35	542-TK11	80,28388928	5,203405866	12,44343891
36	542-TK14	84,89709013	11,06906339	48,54771784
Mean		**83,84099599**	**50,53392039**	**44,49454062**

K-Means outliers_fraction = 0,0082 Time: 5,17 min

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	70,76216449	2,722063037	100
1	200-TK02	77,7329103	3,441802253	98,21428571
2	200-TK03	79,43722944	3,781228899	100
3	200-TK04	81,24458562	2,593440122	60,71428571
4	226-K330	76,37409099	3,257103257	97,91666667
5	226-K331	84,44477523	1,754385965	32,72727273
6	226-K332	93,08597824	3,203661327	25,45454545
7	226-K339	83,73448946	2,568807339	51,85185185
8	226-K340	80,07777445	2,170658683	53,7037037
9	226-K344	90,51182281	0	0
10	226-K346	79,97305793	0	0
11	226-K347	85,46189549	0,432432432	7,407407407
12	226-T301	76,99805068	0,134770889	3,703703704
13	226-T302	82,56110361	0,798580302	16,66666667
14	226-T303	80,4468436	2,143950995	51,85185185
15	226-T306	81,72139751	3,287890938	75,92592593
16	226-T313	78,7974209	1,704545455	44,44444444
17	226-T314	88,72394662	1,120448179	14,81481481
18	226-T316	83,10091468	4,027420737	87,03703704
19	226-T319	85,47008547	1,277955272	22,22222222
20	226-T320	84,27050532	0,888450148	16,66666667
21	226-T321	83,04093567	0,368663594	7,407407407
22	226-T322	81,68866227	2,367346939	53,7037037
23	226-T323	78,62927415	0	0
24	226-T324	84,16316737	3,001876173	59,25925926
25	226-T330	79,52909418	3,543586109	92,59259259
26	226-T338	81,10377924	3,686635945	88,88888889
27	542-TC1055	77,67996592	0,197238659	5,263157895
28	542-TC3044	83,67049389	0	0
29	542-TC3045	89,55713505	0,927643785	11,36363636
30	542-TC3046	89,86695826	0,194552529	2,272727273
31	542-TK01	69,69353008	0	0
32	542-TK02	99,19125993	0	0
33	542-TK07	98,1407891	0	0
34	542-TK10	77,51596877	0,195694716	5,263157895
35	542-TK11	92,97374024	0,452488688	3,50877193
36	542-TK14	96,05393896	4,149377593	17,54385965
Mean		**83,44404692**	**1,632289215**	**35,36190582**

Supervised learning Time: 1,57 hours

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	100	100	99,99708397
1	200-TK02	100	100	100
2	200-TK03	100	99,99838569	100
3	200-TK04	100	100	99,99908479
4	226-K330	100	100	100
5	226-K331	100	100	99,9990367
6	226-K332	100	99,99296072	100
7	226-K339	100	100	99,99832567
8	226-K340	100	100	99,99929596
9	226-K344	100	99,99138392	100
10	226-K346	100	100	99,99171874
11	226-K347	100	99,99782209	99,99673316
12	226-T301	100	99,99935404	99,99806214
13	226-T302	100	100	99,99835253
14	226-T303	100	100	99,99850387
15	226-T306	100	100	99,99705349
16	226-T313	100	100	99,99853617
17	226-T314	100	100	100
18	226-T316	100	99,99913427	99,99480587
19	226-T319	100	100	99,99265107
20	226-T320	100	100	99,99810959
21	226-T321	100	100	99,99439645
22	226-T322	100	100	99,99922869
23	226-T323	100	100	99,99797393
24	226-T324	100	100	99,99911011
25	226-T330	100	100	100
26	226-T338	100	100	99,9931416
27	542-TC1055	100	100	100
28	542-TC3044	100	99,96527606	99,96031746
29	542-TC3045	100	99,89577905	99,96089167
30	542-TC3046	100	99,88269433	99,96910338
31	542-TK01	100	100	100
32	542-TK02	100	0	0
33	542-TK07	100	100	100
34	542-TK10	100	100	100
35	542-TK11	100	100	100
36	542-TK14	100	100	100
Mean		**100**	**97,28980514**	**97,2927437**

LSTM n_epochs = 100, batch_size = 10

Column1	Process	Accuracy	Recall	Precision
0	200-TK01	70,34591646	2,340019102	70
1	200-TK02	76,63686184	1,501877347	34,28571429
2	200-TK03	78,83116883	2,835921675	60
3	200-TK04	80,98469535	2,440884821	45,71428571
4	226-K330	76,27261965	3,465003465	83,33333333
5	226-K331	84,93599286	3,996101365	60,29411765
6	226-K332	92,74325734	2,059496568	13,23529412
7	226-K339	84,46703543	5,412844037	88,05970149
8	226-K340	79,91325157	2,245508982	44,7761194
9	226-K344	90,49685723	1,034482759	8,955223881
10	226-K346	81,03577309	3,271028037	62,68656716
11	226-K347	86,01587064	3,135135135	43,28358209
12	226-T301	77,582846	1,886792453	41,79104478
13	226-T302	82,45614035	1,064773736	17,91044776
14	226-T303	80,34188034	2,373660031	46,26865672
15	226-T306	80,92667566	1,6840417	31,34328358
16	226-T313	79,23226871	3,196022727	67,1641791
17	226-T314	89,45868946	5,462184874	58,20895522
18	226-T316	82,54610886	2,999143102	52,23880597
19	226-T319	85,96491228	3,727369542	52,23880597
20	226-T320	84,25551057	1,480750247	22,3880597
21	226-T321	83,83565752	3,410138249	55,2238806
22	226-T322	80,95380924	0,897959184	16,41791045
23	226-T323	79,48410318	2,552881109	52,23880597
24	226-T324	83,96820636	3,001876173	47,76119403
25	226-T330	78,49430114	1,559177888	32,8358209
26	226-T338	80,69886023	3,149001536	61,19402985
27	542-TC1055	78,90103649	3,484549638	74,64788732
28	542-TC3044	83,98031711	1,643192488	25,45454545
29	542-TC3045	90,1221068	4,823747681	47,27272727
30	542-TC3046	90,17678148	2,918287938	27,27272727
31	542-TK01	71,1123723	2,741702742	80,28169014
32	542-TK02	98,99262202	0	0
33	542-TK07	98,0272495	4,054054054	4,225352113
34	542-TK10	78,8502484	3,718199609	80,28169014
Mean		**82,94405727**	**2,730508857**	**45,97955541**