Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Hes·so

Degree Programme
Systems Engineering

Major Infotronics

Bachelor's thesis
Diploma 2023

Gaëlle Bitz

Eco: Energy Visualization & Machine Cycle Detection

Professor
Silvan Zahno

Expert
Benoît Hubert

Submission date of the report
18.08.2023

HES-SO Valais-Wallis • rue de l'Industrie 23 • 1950 Sion
+41 58 606 85 23 • hei@hevs.ch • www.hevs.ch/hei

Données du travail de diplôme
Aufgabenstellung der Bachelorarbeit

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2022-23**	No TB / *Nr. BA* **IT/2023/79**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☒ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Gaëlle Bitz** Professeur / *Dozent* **Silvan Zahno**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja ☒ non / nein	Expert / *Experte (données complètes)* **Benoît Hubert**, benoit.hubert@constellium.com Constellium Valais SA, Rue de l'industrie 15, 3965 Chippis	

Titre / *Titel*

Eco: Energy consumption visualization and machine cycles detection

Constellium Valais SA utilizes 57.7GWh of energy with a power peak of 11.6MW during 2021 for the casting of aluminium cylinders. To minimize production costs and improve efficiency, it is essential to limit the maximum power peak and distribute the consumption load effectively. Therefore, this bachelor thesis aims to develop a software application that analyzes real-time energy consumption data and detects machine cycle patterns. By utilizing time-series data analysis and machine learning techniques, the software application will suggest optimizations for energy usage during machine cycles, enabling organizations to make informed decisions and improve energy efficiency. The data will be collected from various sources and stored in a centralized database (historian), with communication between the database and software application occurring over an OPC-UA connection.

Objectives:

- Set up the dataserver to collect energy consumption data from various sources and store it in a database (historian).
- Collect data through an OPC-UA interface from the dataserver.
- Develop a user-friendly interface to visualize real-time energy consumption data and identify usage patterns.
- Implement algorithms to detect machine cycles and identify energy usage patterns.
- (Optional) Develop algorithms to suggest potential optimizations for energy usage during machine cycles.
- (Optional) Evaluate the software application's accuracy and effectiveness in detecting machine cycles and identifying energy usage patterns.

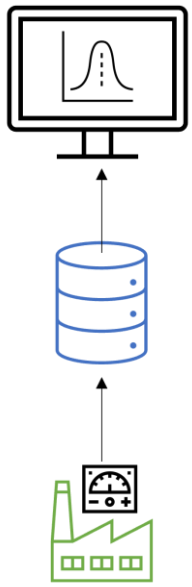
Signature ou visa / *Unterschrift oder Visum*	Délais / *Termine*
Responsable de l'orientation / *Leiter der Vertiefungsrichtung:*	Attribution du thème / *Ausgabe des Auftrags:* **15.05.2023** Présentation intermédiaire / *Zwischenpräsentation*: **19 - 20.06.2023** Remise du rapport final / *Abgabe des Schlussberichts:* **18.08.2023, 12:00** Expositions / *Ausstellungen der Diplomarbeiten:* **25.08.2023** – HEI **28.08.2023** – Monthey **31.08.2023** – Visp Défense orale / *Mündliche Verfechtung:* **Semaine/Woche 36 (04-07.09.2023)**
[1] Etudiant / Student:	

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* …….. Visa du secrétariat / *Visum des Sekretariats* ……..

Hes·so VALAIS WALLIS

Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften



Eco: Energy Visualization & Machine Cycle Detection

Graduate Gaëlle Bitz

Objectives

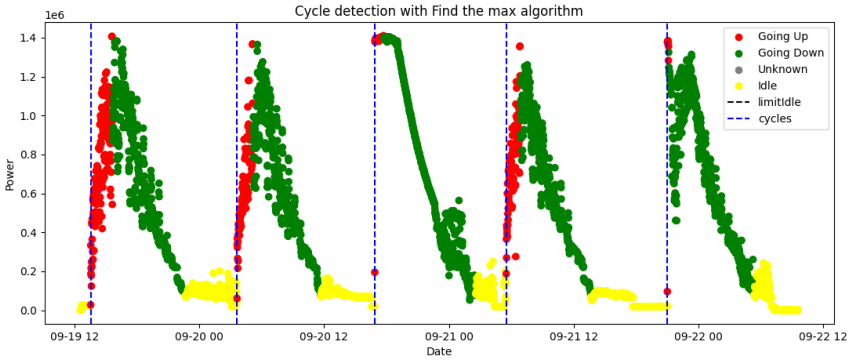
The first objective of this project is to develop a machine cycle detection algorithm for Constellium's homogenisation oven. The second one focuses on creating a user-friend dashboard to display real-time measurements and cycle of the oven.

Methods | Experiences | Results

Developing the machine cycle detection involves a systematic exploration of diverse algorithms. Through rigorous testing and comparison with machine cycle parameters, an algorithm with optimal accuracy and reliability is identified.

The Python-powered dashboard is crafted utilizing the 'plotly.dash' library. This dynamic solution offers a holistic view, presenting concurrent insights into power consumption trends and comprehensive statistical analyses of operational cycles for each oven. In addition, a vital component of the project involves efficient data collection. This is achieved by establishing a localized OPC-UA server. Given the constraints posed by limited available datasets, the inclusion of synthetically generated data enhances the depth and robustness of the analysis.

In summary, the project aims to optimize operational efficiency on two fronts: through the precise detection of machine cycles and the provision of a user-friendly dashboard offering real-time monitoring and data-driven insights into energy utilization and operational cycles of Constellium's homogenisation oven.



Result of the cycle detection

Bachelor's Thesis
| 2023 |

Degree programme
Systems Engineering

Field of application
Infotronics

Supervising professor
Silvan Zahno
silvan.zahno@hevs.ch

Partner
Constellium

Content

1 ACKNOWLEDGMENTS 3

2 INTRODUCTION 4

2.1 CONTEXT 4
2.2 OBJECTIVES 4
2.3 REPORT'S STRUCTURE 4

3 STATE OF THE ART 6

3.1 TIME SERIES 6
3.2 MACHINE CYCLE 7
3.3 MACHINE LEARNING 8
3.4 SYNTHETIC DATA GENERATION 9
3.4.1 Auto-regressive models 9
3.4.2 Gaussian Process 10
3.4.3 Markov chains 10
3.5 DATA VISUALIZATION ON A DASHBOARD 11
3.5.1 HTML/CSS and JavaScript 11
3.5.2 Data visualization platforms: Grafana 11

4 DATA ACQUISITION 13

4.1 ARCHITECTURE OF THE SYSTEM 13
4.2 OPC-UA COMMUNICATION 13
4.3 SIMULATION SERVER 14
4.4 DATA PREPARATION 15
4.4.1 Linearity and Seasonality 15
4.4.2 Filters 17
4.5 CUSTOM SYNTHETIC DATA GENERATION 21
4.5.1 Synthetic power data 21
4.5.2 Synthetic temperature data 22
4.6 LOCAL OPC-UA SERVER 23
4.7 DATA ACQUISITION FROM THE DASHBOARD 24

5 DASHBOARD 27

5.1 MAIN LIBRARIES 27
5.1.1 Plotly Dash 28
5.1.2 Pandas 28
5.2 GETTING TO GRIPS WITH THE TOOLS 29
5.3 MORE ADVANCED OPTIONS 30
5.3.1 Multiple ovens display 30
5.3.2 Summary tab for every oven 32
5.3.3 Data storage management 33

6 CYCLES DETECTION 35

6.1 CHANGE POINT DETECTION ALGORITHM 36
6.1.1 Concept 36
6.1.2 Results & Conclusion 37
6.2 CLUSTERING WITH K-MEANS 38
6.2.1 Concept 38
6.2.2 Results & Conclusion 39
6.3 CYDETS PACKAGE 40

6.3.1 Concept ... *40*
6.3.2 Results & Conclusion ... *40*
6.4 ISOLATION FOREST ... 41
6.4.1 Concept ... *41*
6.4.2 Results & Conclusion ... *42*
6.5 SLOPE DETECTION ... 46
6.5.1 Concept ... *46*
6.5.2 Results & Conclusion ... *46*
6.6 IDLE DETECTION ... 47
6.7 FIND THE MAXIMUM ... 47
6.7.1 Concept ... *47*
6.7.2 Results & Conclusion ... *48*

7 RESULTS ... 51

7.1 OPC-UA SERVER ... 51
7.2 DASHBOARD & CYCLE DETECTION ALGORITHM ... 51

8 CONCLUSION ... 53

8.1 SYNTHESIS ... 53
8.2 IMPROVEMENTS AND FURTHER DEVELOPMENTS ... 53
8.2.1 OPC-UA server ... *53*
8.2.2 Dashboard ... *53*
8.2.3 Storage ... *54*
8.2.4 Algorithm ... *54*

9 BIBLIOGRAPHY ... 55

10 FIGURE TABLE ... 58

11 CODE TABLE ... 59

12 CHART TABLE ... 59

13 ANNEXE ... 60

13.1 SUSTAINABLE DEVELOPMENT GOALS ... 60
13.1.1 What are the Sustainable Development Goals ... *60*
13.1.2 Goal 7: ensuring access to affordable, reliable, sustainable, and modern energy for all ... *60*
13.1.3 Goal 9: Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation ... *60*
13.2 DEPLOYMENT PROCESS ... 62

1 Acknowledgments

I would like to thank my family and my friends who have helped and supported me throughout the three months of this Bachelor's thesis. A special thank you to Apolline Aymon for her funny comments while proofreading this work, and to Axel Bonetti who made sure I did not go crazy. Finally, thank you to M. Silvan Zahno for his involvement in this project.

2 Introduction

2.1 Context

The aluminium casting industry is well-established in Valais. Initially operated by Alcan in 1989, and by Constellium and Novelis nowadays, the factories in Sierre, Chippis, and Steg focus on four types of production: billets and slab casting, as well as extrusion and plate making. The aluminium from these facilities is used in aerospace and transportation, packaging, automotive business, and in the industry [1]. This work obviously needs a lot of energy. As a matter of fact, Constellium Valais consumed 57.7 GWh of energy in the year 2021, with a power peak of 11.6 MW. Calculating electricity costs relies on two parameters: annual energy consumption and power consumption. Power consumption is defined as follows: the highest power peak is recorded every 15 minutes, and the monthly consumption is the mean of these recorded peaks [2]. However, this calculation outcome does not necessarily reflect the usual power usage pattern. Being more effective in the power load distribution is a method to decrease these peaks. For instance, reducing the number of oven heating at the same time by adjusting their activation. To achieve this, the operator needs to know the stage of each machine's cycle. This thesis aims to address this issue by developing a virtual interface that provides real-time consumption data and cycle information for various devices, thus assisting operators in optimizing power distribution effectively.

2.2 Objectives

The objectives of this work are split into three categories:

- Data acquisition
- Data visualization
- Data analysis

The data collection process employs an OPC-UA interface to establish a connection between the server and the data sources. Therefore, it is necessary to set up this server. Furthermore, if the collection of real data is not possible, the production of synthetic data would be necessary.

In parallel, the development of an intuitive user interface is undertaken. This interface is designed to provide users with real-time visualization of energy consumption data. Its user-friendly nature ensures easy interpretation of usage patterns and trends.

Ultimately, the focal point of this work revolves around the data analysis phase. This critical stage involves the deployment of diverse algorithms designed to detect machine cycles.

If time allows it, some additional objectives are provided, such as the development of an algorithm to suggest potential optimizations for energy usage during machine cycles and the evaluation of its accuracy and effectiveness.

2.3 Report's structure

The rest of this paper is organized as follows: Section 3 presents some key aspects used during the project and their current development; Section 4 explains the global architecture of the system and the data acquisition process. Section 5 addresses the visualization part and the development of the dashboard, Section 6 presents the algorithms tried to perform the cycle detection and the production of custom algorithms. Section 7 brings together the results of this work. Finally, the conclusion in

Section 8 opens up to further improvements that could be made and gives personal feedback on the project. In annexe, you will find a reflection on how this work fits in the sustainable development goals decreed by the United Nations Development Programme, as well as an explanation on how to deploy the dashboard.

The entire project is stored in the school's git: https://gitlab.hevs.ch/SPL/bachelorthesis/2023-eco/eco_tb . For any access rights, please contact Silvan Zahno (silvan.zahno@hevs.ch).

3 State of the art

3.1 Time Series

First of all, it seems essential to discuss what time series are and how to analyse them, as datasets for this project are presented under this form.

Time series are a way to represent the evolution of a value through time. They appear in multiple fields, such as economy, with the evolution of the price of gold through the years; meteorology, when recording temperature variations; healthcare, to monitor the evolution of a disease; or, as in this work, in the industry, where they can monitor the power consumption of an oven. In these examples, the time series is discrete, which means that measurements or observations are made at specific times, usually with a predefined frequency. This means that to each timestamp corresponds a unique value.

The first step when working with time series is processing the received data. This implies handling missing values, filtering the noise, and smoothing the data or even in some cases normalizing it to remove any scaling effect. Filters and smoothing options will be discussed in 4.4.2.

Once the data has been cleaned, analysis can begin. Indeed, time series can lead to numerous underlying information: they can provide a visual description and explanation of the considered sample or can be useful to predict future behaviour or to base control decisions on. The easiest operation when facing a time series is plotting it in the function of time. A simple time graphic can already provide a lot of information, as the time plot for the three first examples cited earlier can show:

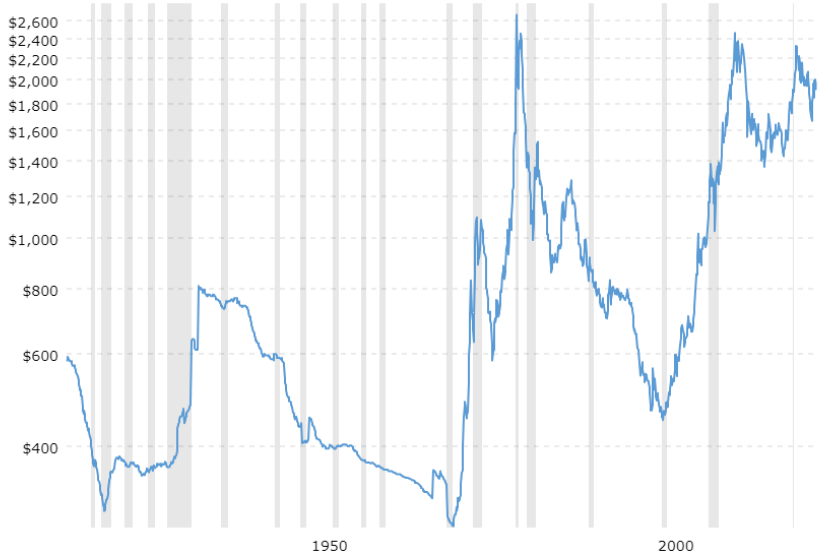
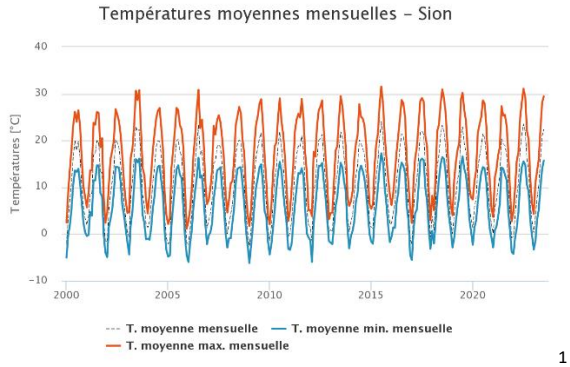
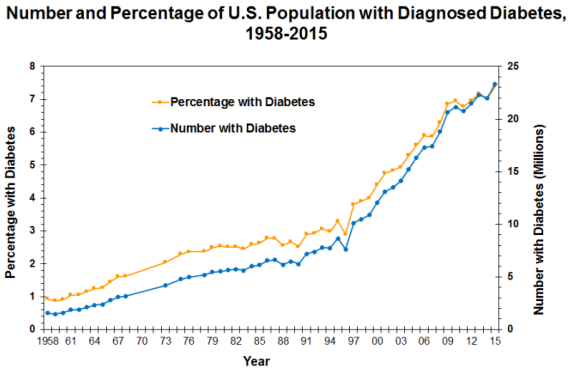


Figure 1: the price of gold per ounce, between 1915 and 2023 (inflation-adjusted)[3]





[1]

Figure 2: Average monthly temperatures – Sion [4]

Figure 3: Evolution of diabetes diagnosis [5]

Just by looking at the graphs, we can conclude a certain amount of information:

- Figure 1: some important features represented are the minimum and maximum points of the graph. It may be interesting to conduct more research to determine which event could have led to these extremum
- Figure 2: there is an evident seasonality in this graph. Indeed, temperatures are dependent on the period of the year
- Figure 3: the main element of this plot is the evident upward trend.

These three examples enlightened essential aspects of time series analysis: extremum, seasonality, and trend. This information is very valuable when performing data analysis and are critical parameter used for statistics or machine learning algorithms.

3.2 Machine cycle

Given that this thesis centres around the detection of machine cycles in industrial ovens, it becomes imperative to dedicate time to address the fundamental question: 'What constitutes a machine cycle?'. Curiously, there appears to be a dearth of documentation explicitly outlining what can be deemed as a machine cycle. In fact, there is a lack of a term that precisely encapsulates the scenario presented in this project. This underscores the significance of establishing a clear and precise definition for this term.

The first step in defining a machine cycle is to understand how the process happens in Constellium. A visit and presentation of the factory allowed to understand that every time the operator puts a new load in an oven, the action is registered. This registration protocol takes into account various factors such as the aluminium alloy type and the load volume, thereby estimating a specific duration for the load's tenure within the oven. Therefore, a machine cycle could be defined as the time between the registration of two different loads of aluminium. Upon further analysis, it has been found that the information about the end of the heating process exists, potentially making the machine cycle detection way easier. However, the reliability of this resource is tempered by the uncertainty of access, rendering it an unpredictable reference point to depend upon.

At this point in the global project that this thesis is part of, the only time series that can be relied on is the power consumption of each oven. Thus, the only way to define what a machine cycle is, consists

[1] In grey : average monthly temperatures; in blue : average minimum monthly temperatures; in orange : average maximum monthly temperatures

of detecting when the power values start to rise again, after a being low for a while. This will constitute the base to build a machine cycle detection algorithm, as presented in section 6.

3.3 Machine learning

As some algorithms presented in Section 6 involve machine learning, it seems important to provide a base ground knowledge about this field of computer science. Since their invention, there has been this underlying question of whether computers will one day be able to learn or not. This question has been a driving force behind technological advancements. However, in the contemporary landscape, this inquiry has transformed into a reality. Today, generative artificial intelligence like ChatGPT have seamlessly integrated into the fabric of our lives, revolutionizing how we interact with technology. But what underpins these remarkable feats? The answer lies in the domain of machine learning.

At its core, machine learning aims to emulate the process of human learning, in a highly automated and data-driven manner. Instead of explicit programming, where every possible scenario is predetermined by human developers, machine learning leverages algorithms that enable computers to adapt and evolve based on the information they encounter. This paradigm shift has led to monumental advancements in various fields, ranging from healthcare and finance to entertainment and beyond.

Tom Mitchell, in his book 'Machine Learning' [6] proposes the following definition to what machine learning is:

'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.'

This means that machine learning involves three features: a task, a performance measure, and some training experience. A great illustration would be teaching a computer to play chess: the task is playing chess; the performance measure is how many games are won; and the training experience is playing practice games against itself.

Machine learning algorithms can be sorted into three classes: supervised, unsupervised, and semi-supervised learning. Supervised learning is characterized by its reliance on labelled data and the task of mapping input data to corresponding output labels. The process involves creating a model that learns the relationships between input features and their corresponding labels from the provided training data. Supervised learning encompasses two main types of tasks: classification and regression. In classification, the goal is to assign input data to predefined categories or classes. Examples include email spam detection (spam or not spam) and image classification (identifying objects in images). In regression, the objective is to predict a continuous numerical value. This is used in scenarios like predicting housing prices based on features like square footage, number of bedrooms, etc [7].

On the other hand, unsupervised learning deals with unlabelled data. The algorithm's goal is to uncover patterns, relationships, and structures within the data without explicit guidance on the correct outputs. Some important applications of unsupervised machine learning consist of clustering and dimensionality reduction [8].

Finally, semi-supervised algorithms are a mix between supervised and unsupervised algorithms. It harnesses the power of both labelled and unlabelled data to build more robust models. In semi-supervised learning, the algorithm leverages a small amount of labelled data along with a larger pool

of unlabelled data to make predictions or decisions. This approach acknowledges the reality that obtaining labelled data can be expensive and time-consuming [9].

The number of data being very limited, the optimal choice for this project lies in the utilization of unsupervised machine learning algorithms.

3.4 Synthetic data generation

In a world where data seems to be omnipresent, finding the right dataset for a project can be a challenge for legal, privacy or other reasons. Regarding this thesis the problem is the lack of data, having only a couple of datasets, which is not enough to test algorithms.

In the next paragraphs, you will find several approaches often used to generate synthetic data.

3.4.1 Auto-regressive models

One way to generate data from an existing dataset is to use an auto-regressive model. Often used in statistics, this mechanism takes as input existing measurements to predict future values. It is based on linear regression, which determines the line that represents the trend of a time series, applied to a variable amount of previous data [10].

One of the most well-known models for analysing time series is the Auto-Regressive Integrated Moving Average (ARIMA). The “AR” part of ARIMA represents the autoregressive component, which involves using linear regression techniques as discussed earlier. The moving average component of the model calculates the difference between a variable number of past data points and the mean value of the time series. This information is then used to predict the next data point. ARIMA is characterized by three parameters: d, p, and q.

The first one, d, represents the number of differentiations needed to get a stationary time series. A stationary time series keeps a constant mean and variance over time. This step is necessary to get the best prediction possible, by feeding the linear regression part of the model data that is independent and non-correlated.

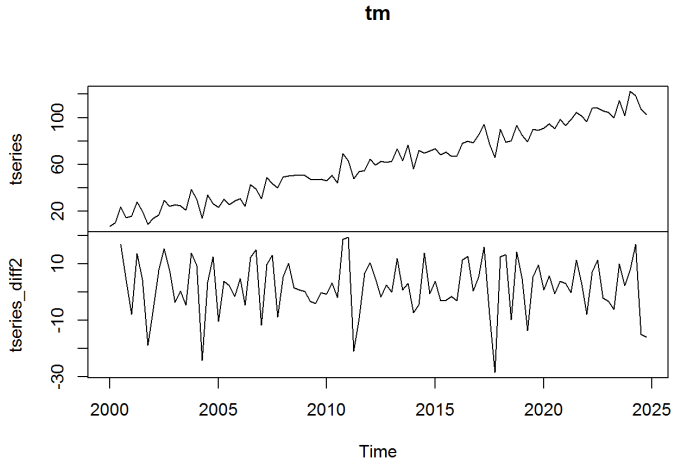


Figure 4: time series differentiated twice to become stationary [11]

The second one, p, indicates the order of the auto-regressive model. This order equals to the number of past values considered for the linear regression calculation. Finally, q determines the order of the moving average model [12].

These parameters are used to fit a model to an already existing data set. Once the model meets the requirements, it is used to predict future values and therefore grow the amount of data. The library 'statmodels' in Python is the reference for the ARIMA model [13].

This model is used in multiple fields, such as economy where S. Khan and H. Alghulaiakh explain how to forecast stocks time series [14], or the industry to forecast electricity consumption as enlightened by F. Mahia, A. R. Dey, M. A. Masud, and M. S. Mahmud [15].

3.4.2 Gaussian Process

Another popular technique that generates synthetic data based on existing data uses Gaussian Process (GP), as demonstrated by N. Rücker, L. Pflüger, and A. Maier to predict hardware failure [16]. This generative model is based on supervised machine learning and is commonly found in regression problem-solving. Like ARIMA, Gaussian processes output comes out of a regression model, this time based on Gaussian distributions.

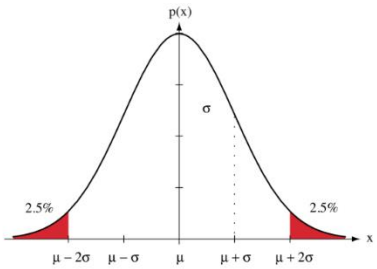


Figure 5: Gaussian distribution [17]

As seen in Figure 5, Gaussian distribution has a symmetric density around the mean, μ, and a standard deviation of σ. These characteristics enable the model to craft several functions that could fit with the current representation of the given data. With every new point analysed, the model adapts itself and gets more accurate in determining the right function.

A fundamental notion in GP is the function k, called the kernel of the Gaussian process. It defines the relationship between data points and quantifies how the output of a GP varies as a function of the input variables. Mathematically, a kernel $k(x_i, x_j)$ is a function that takes two input data points x_i and x_j, and produces a measure of similarity or correlation between them. The choice of kernel function shapes the characteristics of the GP model, influencing factors like smoothness, periodicity, and noise levels in the predictions[18].

Unlike many traditional regression models that rely on fixed parameters and assumptions about the underlying data distribution, GP is non-parametric, which means that it does not assume a specific functional form for the relationships between variables. Instead, it models the underlying data distribution directly from the observed data, allowing more flexibility in capturing complex patterns.

However, it is important to highlight that GP are not well-suited for handling large datasets. Indeed, GP has a cubic time complexity $\mathcal{O}(n^3)$, required to compute every calculation [19].

3.4.3 Markov chains

Another commonly used process involves the Markov chains, which takes into account an already existing dataset. A Markov chain is a system that relies on the probability to transition from a state to another. The main property of this concept, the Markov property, is that the probability to change state only depends on the current state and not on previous ones [20].

In practice, the considered time series is discretised into distinct states. Then, a transition probability matrix is built by determining the transition probability for each combination of state. This matrix signifies how likely the system is to move from one state to another. After that, the state transitions are simulated with a sample of uniformly distributed values between 0 and 1. The results are converted back to real values. This approach effectively captures not only the distribution and correlation attributes but also the autocorrelation of longer lags in a time series with a notably high level of precision [21].

This approach is often use to generate synthetic data for smart grid application, such as solar states[22] or modelling wind power [23]

3.5 Data visualization on a dashboard

Various methods exist for displaying data on a dashboard. This section aims to introduce two alternative approaches to the one discussed in Section 5. This examination offers an insight into diverse techniques for presenting data effectively to cater to different requirements. Exploring these alternatives provides a comprehensive understanding of available options and their respective strengths in data visualization for dashboards.

3.5.1 HTML/CSS and JavaScript

One of the foundational approaches to construct a data-displaying website involves using HTML and CSS for layout design and JavaScript to implement the logic. This classical combination forms the backbone of countless web applications, offering a robust foundation for presenting information in a structured and interactive way. The integration of these three technologies can create user-friendly and visually engaging data-driven platforms. In this project, the data is displayed as plots. A layout using HTML would use a Scalable Vector Graphics (SVG) element to create space for an interactive plot. It allows for a responsive and scalable chart, especially designed for graphical web implementations. Unlike the canvas element, SVG is XML-based, which means that its graphics are tied to the Document Object Model (DOM). Therefore, it can be easily manipulated using CSS and JavaScript. Furthermore, the shape displayed on the web page is remembered as an object, meaning that if an attribute is changed, the browser re-renders the shape only and not the entire scene. JavaScript on the other hand takes care of the logical part of the program, in this case data analysis. An important point to note is that JavaScript allows the SVG to be updated upon every change in the dataset, which is primordial in a real-time environment [24].

3.5.2 Data visualization platforms: Grafana

There are plenty of different platforms that offer real-time data visualization. This section takes Grafana as an example since this tool has already been used during projects throughout the school year.

Grafana is an open-source platform designed for data visualization and monitoring. It enables users to create, explore, and share interactive and customizable dashboards that display various metrics, analytics, and insights from a wide range of data sources. Grafana is commonly used to monitor systems, applications, services, and other data streams in real-time.

Grafana's main strength lies in its ability to create interactive dashboards. It offers a wide range of visualization options such as graphs, charts, gauges, tables, heatmaps, and more. These visualizations can be customized using various options to best represent the data being displayed. Dashboards created in Grafana can easily be shared and embedded in other applications, websites, or portals.

Collaboration features allow multiple users to collaborate on dashboard design and content. This tool also provides a powerful query editor that allows users to retrieve specific data points or perform aggregations from their chosen data sources. It supports query languages specific to the connected data sources, making it adaptable to different types of databases[25].

Grafana supports a wide range of plugins and extensions, offering additional data sources, visualizations, and functionalities. These extensions can be used to enhance the capabilities of Grafana based on specific use cases. There is even a plugin that allows to read data from an OPC-UA server [26].

4 Data Acquisition

This chapter outlines the data acquisition process. It commences with a comprehensive overview of the system's architecture and fundamental concepts for this project's data acquisition phase. Subsequently, an elucidation of the data preprocessing methodology is presented, along with insights into the generation of synthetic data. Lastly, the chapter delves into the specific approach adopted for data acquisition throughout the course of this project.

4.1 Architecture of the system

As stated in the introduction, the aim of this thesis is to monitor some of Constellium's infrastructure. This work focuses on homogenization oven. Homogenization is a thermal process that consists of heating the aluminium in order to evenly distribute its elements. This process allows for a uniform internal structure.

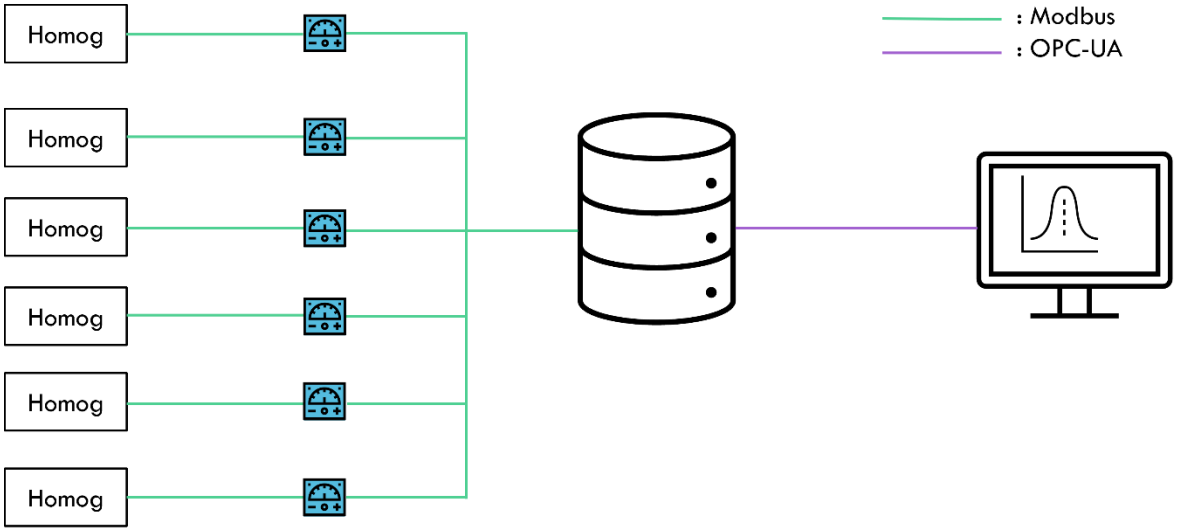


Figure 6 : Data acquisition architecture

As shown in the above Figure, measures are taken inside the oven by Schneider Electrics devices, thanks to a Modbus communication. They are then sent towards an OPC-UA server, hosted by a Beckhoff computer. The next section provides a description of OPC-UA. Finally, the program hosted by the final computer polls periodically the server, with the aim of displaying real-time information and their analysis.

4.2 OPC-UA communication

Until those last couple of years, data transmission in the world of industry was based on protocols that did not provide any information security. This system used to make sense since sharing data was not such a central point as it is now. With the development of the Industry 4.0, more and more measuring instruments, sensors, actuators and even machines are part of the Internet of Things (IoT), which means that data is shared on a global network that anybody can access. This, of course, is not a viable situation. Firstly, most companies want their data to be confidential. Secondly, it is an open window for attacks on an enterprise. If anyone can access the controller of any machines, this can lead to dangerous situations. There are plenty of other reasons that justify the need of a secure way to transmit data.

To overcome this problem, the Open Platform Communications Unified Architecture standard was developed by the OPC Foundation, an organization which include automation and programming industries. OPC-UA is an upgrade from OPC-DA, invented in 1996 [27]. This first version was already a revolution in the industrial world. As long as a software was implementing OPC-DA it could display real time data acquired by any device, for instance PLCs (Programmable Logic Controller). OPC-UA adds the information security layer. It provides a secure way to exchange data, based on a client/server or server/server communication infrastructure. The security layer includes authentication, authorization, encryption and checksum [28]. This technology is open-source and cross-platform, which allows its use on multiple different industrial equipment and a cloud-based infrastructure.

4.3 Simulation server

Since the system described on Figure 6 is not yet operational, to simulate the OPC-UA layer, a simulation server was used, the Prosys OPC-UA Simulation Server. It allows the user to test their OPC-UA client freely, and without the need for an actual server. The server runs on the user's computer. The simulated server includes predefined variables, a large amount of data types and permits the simulation of various data signals, such as sinusoids, random points, counters, etc. Additionally, the user can add their own variables, to get a more accurate representation of a given system. This simulation server also takes into consideration the information security aspect of OPC-UA. The user can easily define multiple access levels or generate authentication certificate.

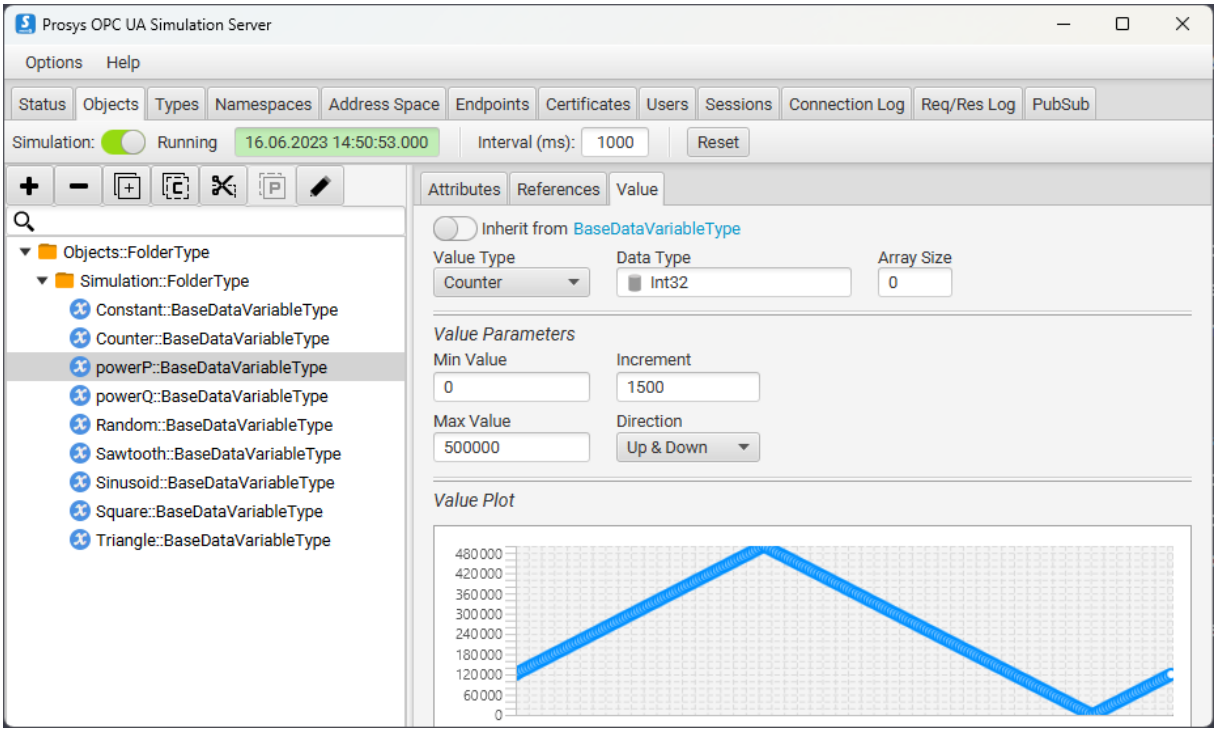


Figure 7 : Prosys OPC-UA Simulation Server interface

The selection of this simulation server was primarily based on its quick and uncomplicated installation process. Moreover, its certification by the OPC Foundation adds an extra layer of assurance, ensuring that any work performed on it translates to compliance when transitioning to the actual server[29].

4.4 Data preparation

Before proceeding with the further development of this project, it is crucial to take some time to comprehend the key characteristics of the two datasets provided by Constellium. They form the backbone of the entire development process, as they constitute the only data made available for most of the project. The first dataset spans from the 19th to the 22nd of September 2022, while the second one starts the 3rd of October 2022 and ends the 6th. These datasets present measurements related to various electrical components, including power consumption, tension, current and more for each phase. Understanding the underlying specificities of these datasets is of the greatest importance to justify choices made throughout the project. Since this thesis focuses mainly on power consumption, the focus is put on the '|P total| average' measurement. This measurement groups the three phases, providing a good overview of the power variation, in a single time series.

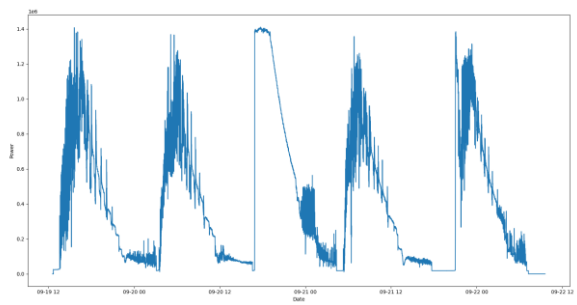


Figure 8 : |P total| average, 19/09/22 to 22/09/22

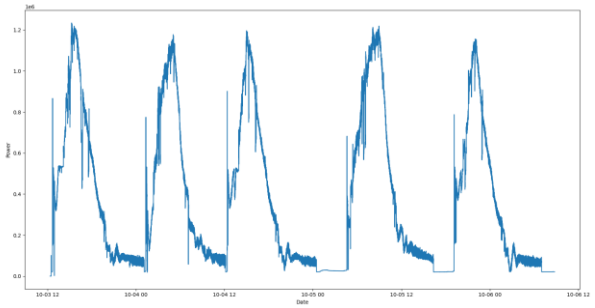


Figure 9 : |P total| average, 03/10/22 to 06/10/22

Later in the project, additional temperature data was received. However, before using this data, some preprocessing work was required to segregate it based on their respective oven number and load number. Unlike the power data, the temperature data's sampling frequency was not stable, requiring additional work to resample the data and obtain a fixed frequency of one data point every 3 seconds. The code developed for this part can be found in the 'tempCSV' project.

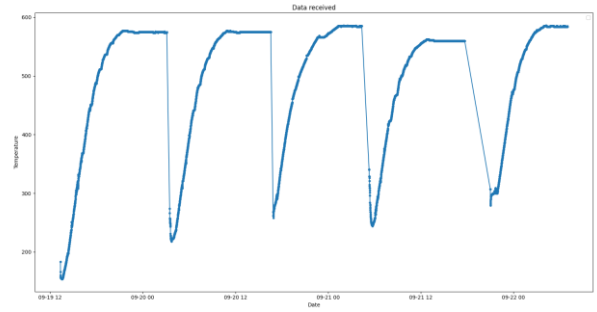


Figure 10 : Temperature data, 19/09/22 to 22/09/22

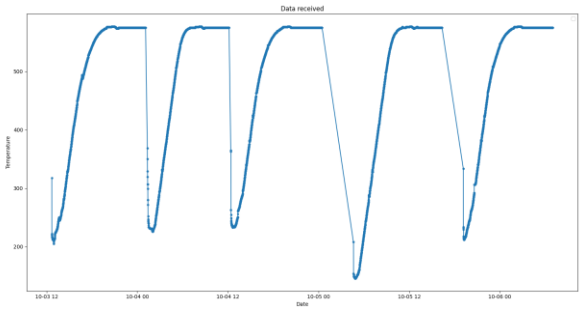


Figure 11 : Temperature data, 03/10/22 to 06/10/22

4.4.1 Linearity and Seasonality

As discussed in section 3.1, time series can be characterized by several features. In this section, the general trend, linearity, and seasonality aspects are especially addressed. These steps are crucial for selecting appropriate algorithms for the rest of the project.

Both signals exhibit clear non-linear behaviour, as no discernible trend or straight-line pattern is observed. Therefore, to formally confirm this observation, the analysis proceeds with the application of the Brock, Dechert, and Schinkman (BDS) test [30], specifically designed to detect non-linear characteristics in the data. The BDS test is based on the null hypothesis that the provided data is linear. The characteristics that determine whether a time series is linear or not is the p-value. This value indicates the likelihood of obtaining the calculated correlation dimension of the original time series, assuming that the data follows a linear pattern. A small p-value, typically under 0.05, suggests that the observed correlation dimension is unlikely under the assumption of linearity. The code used can be found in the 'testDataAnalysis' project, in the file 'testBehavior.py'.

The test has been performed on the four datasets with the following results:

BDS Test Results Power September: P-value: 3.559200975819734e-211 The time series is nonlinear	BDS Test Results Power October: P-value: 4.6929319116480127e-240 The time series is nonlinear
BDS Test Results Temperature September P-value: 2.468682649949159e-190 The time series is nonlinear	BDS Test Results Temperature October P-value: 8.163081265361446e-239 The time series is nonlinear

Figure 12 : p-value from the BDS test for each dataset

The p-value for every dataset is way under 0.05, proving their non-linearity. This means that no algorithm using linear regression or requiring data to be linear can be used later in the project.

Testing for seasonality in the datasets is considered an important step. The visual observations provide some insights; patterns are evident, but the question arises as to whether they can genuinely be attributed to seasonality. To address this question, the 'seasonal_decompose' function from 'statsmodels.tsa.seasonal' was applied to decompose each time series, resulting in the following outcomes:

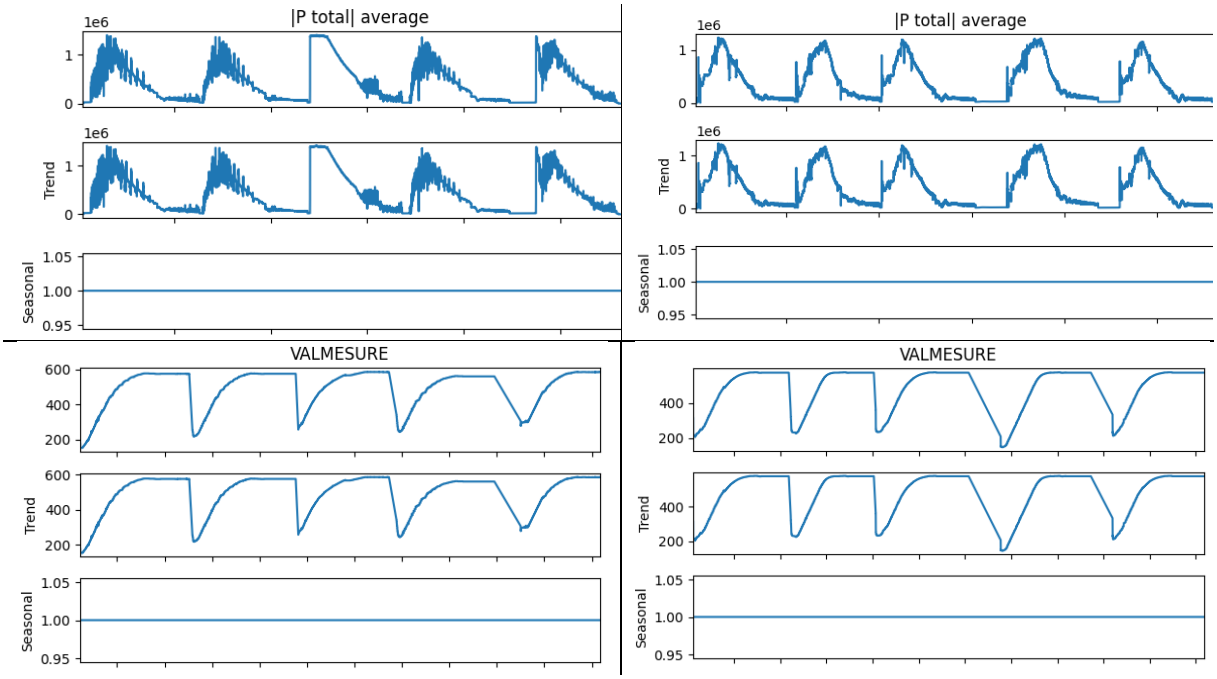


Figure 13 : seasonality decomposition for each dataset; top left: power in September, top right: power in October, bottom left: temperature in September, bottom right: temperature in October

The analysis of each dataset provides additional evidence of the absence of a clear trend, which in turn indicates a lack of seasonality. This observation is easily explained by the fact that seasonality typically occurs over a full year. However, in the case of Constellium's oven usage, it is evident that the patterns do not follow seasonal variations. While there are cycles present in the data, they cannot be considered as seasonality.

4.4.2 Filters

The power data exhibits noise due to the regulation process required to achieve the desired oven temperature. To obtain smoother and cleaner data to perform a better cycle detection, several filtering techniques have been experimented with.

4.4.2.1 Low-pass Butterworth filter

A Butterworth filter is a type of electronic filter, used in signal processing. It allows certain frequencies to pass through, while attenuating others. The cut-off frequency is determined by the filter order. The higher the order, the sharper the roll-off slope. However, it can introduce a notable phase shift, implying a time offset. Butterworth filters provide a smooth and flat response in the passband, not affecting much amplitude in the desired frequencies.

The 'scipy.signal' Python library offers the function 'butter' that returns the coefficients of a digital or analogue Butterworth filter, based on the desired order and the cutoff frequency [31]. According to the Nyquist principle, the sampling frequency must be at least twice the cutoff frequency. Since values are sampled every ten seconds, the sampling frequency is 0.1 Hz, making the cutoff frequency 0.05 Hz. Various orders have been tested and will be presented in the next paragraph. It is essential to strike a balance between a steep roll-off slope and the minimum phase shift possible.

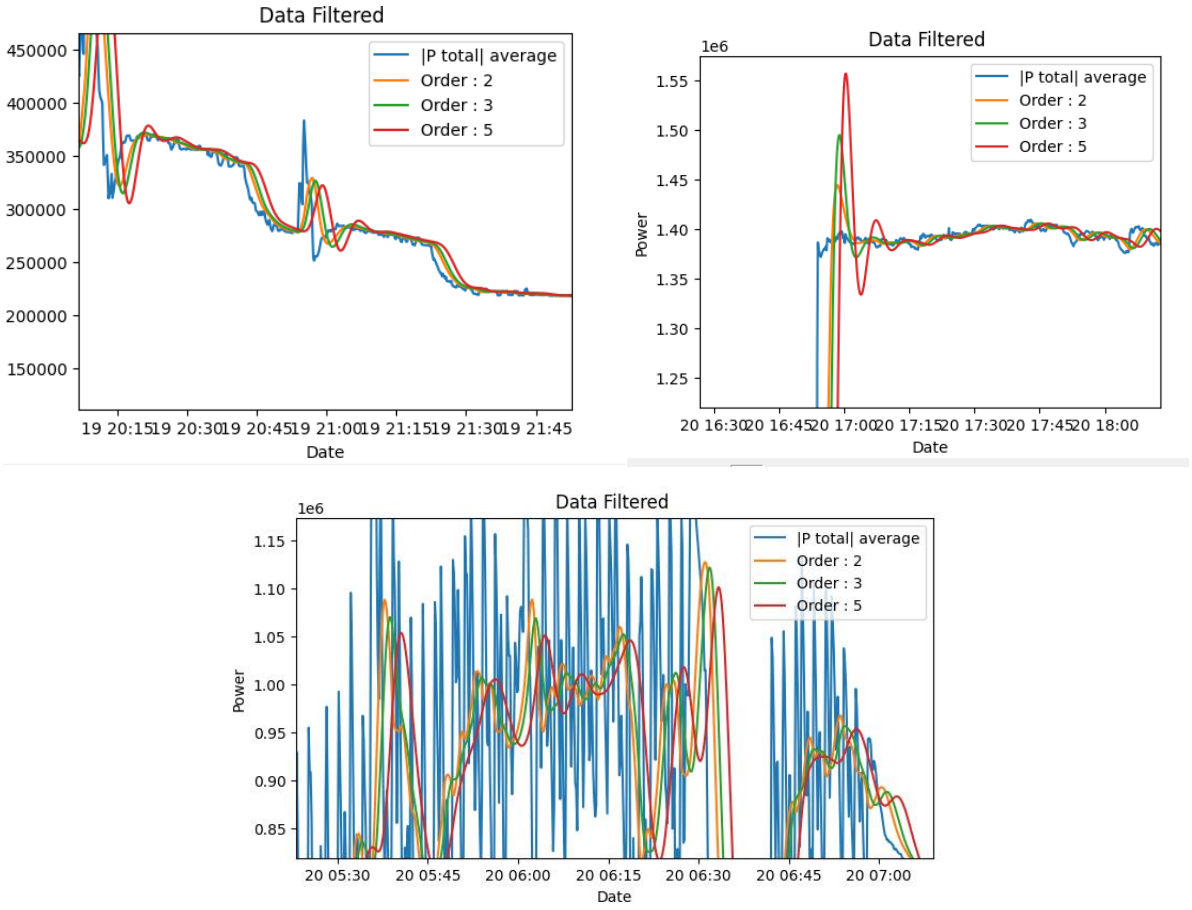


Figure 14 : |P total| average, 19/09/22 to 22/09/22, low-pass Butterworth filter of different orders, zoomed

Three segments of the filtered signal are taken into consideration to prove that a Butterworth filter is not appropriate for the received signal. In the top left picture, the phase shift generated by the filter can be clearly seen. It is quite clear that the fifth order implies way too much delay, nearing four minutes in this case. Even the second order filter produces a non-negligible delay. Since the project is set in a real time environment, introducing a delay is not feasible. Furthermore, the top right picture shows that the filter introduces even more perturbation. This peak is a characteristic of the step response of a low pass Butterworth filter. Unfortunately, the considered signal can sometimes take the form of a step response. However, the aim of filtering, in this case, is to remove peaks. Finally, still in the same idea of smoothing the signal, the third picture reveals that a low-order filter still leaves the signal with noticeable bumps.

These observations lead to the conclusion that the best-case scenario, as far as a low-pass Butterworth filter is concerned, would be the little phase-shift of a low order, but the attenuation of a higher order.

Fortunately, 'scipy.signal' offers a function called 'filtfilt', that applies a linear digital filter twice: once forward and once backward. The combined filter has zero phase, and an order twice that of the original filter[32]. The actual order of the filters is 4, 6 and 10.

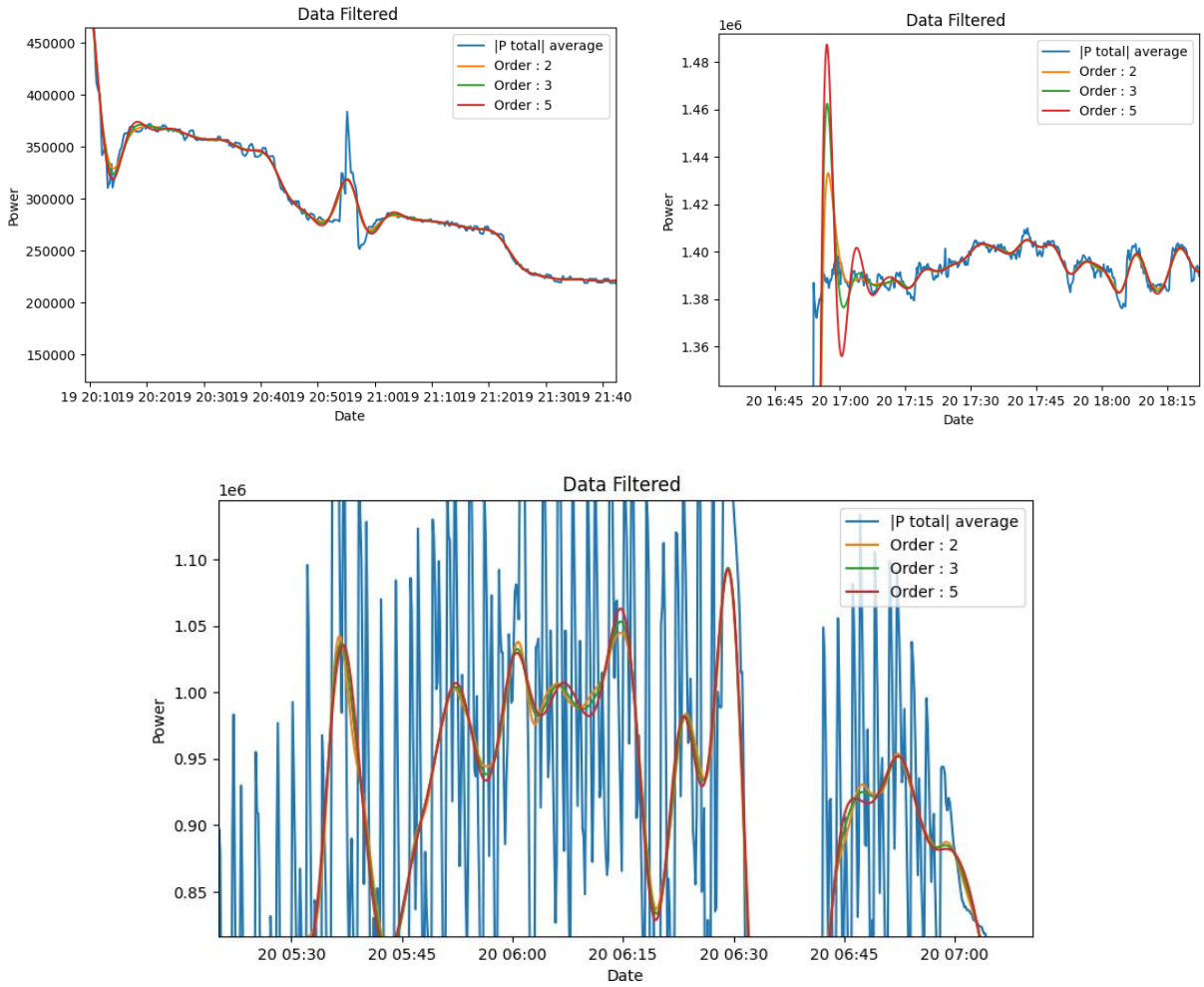


Figure 15 : |P total| average, 19/09/22 to 22/09/22, low-pass Butterworth filter of different orders, using filtfilt, zoomed

The pictures have been taken around the same areas as in Figure 14, to facilitate the comparison. As expected, the time shift has disappeared. However, the peaks of the top right picture are still present. The bottom picture shows that the difference between orders is small. In this case, an order 3 (actual order: 6) would be a good fit.

4.4.2.2 Upper envelope

However, in order to align with the fundamental concept of the project, a Butterworth filter is not an ideal long-term solution. The peaks in the power consumption data hold significance and should not be eliminated during signal preprocessing. An alternative approach to achieving a smoother signal without removing the peaks is to focus on considering only the outer envelope, particularly the upper one.

The idea is to locate the highest points of the signal, by differentiating it a couple of time [33]. The result is an array containing the index of the highest points. Then, in order to retrieve the full signal, linear interpolation was used. The results being very promising, the same process has been applied multiple times, to get a smoother signal, while keeping the general shape of the original signal.

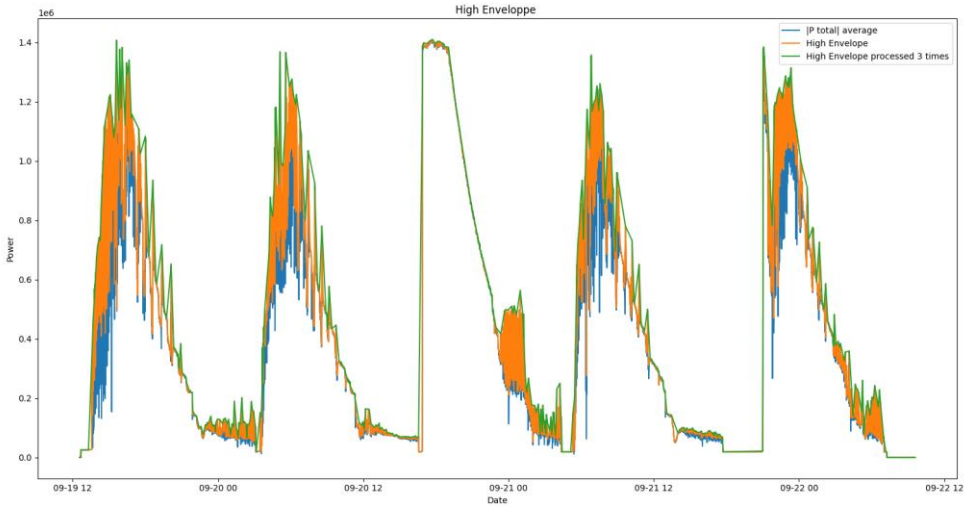


Figure 16 : |P total| average, 19/09/22 to 22/09/22, High envelope

As one can see the lower points of the data are lost, but it is not a problem, as nothing depends on them.

Another way to obtain the same kind of result is to use a rolling window. Each point of the window, which size is predefined, gets the value of the highest point. With this algorithm, there is no need of a linear interpolation, which can be interesting as far as time of execution is concerned. However, more iterations are needed to get a smooth signal. This can be turned as an advantage, as the envelope is closer to the actual signal. In the following Figure, the window size is set to 6.

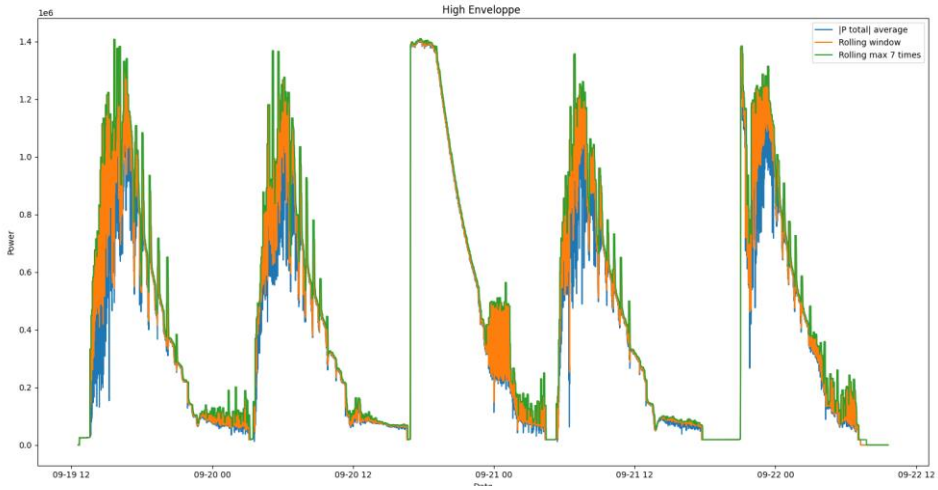


Figure 17 : |P total| average, 19/09/22 to 22/09/22, High envelope with rolling window

The two algorithms have similar results, and their positive and negative aspects balance each other. In the end, the rolling window algorithm is chosen. To be a bit more reasonable than in the Figure 17, the algorithm is repeated five times, with a window size of 4.

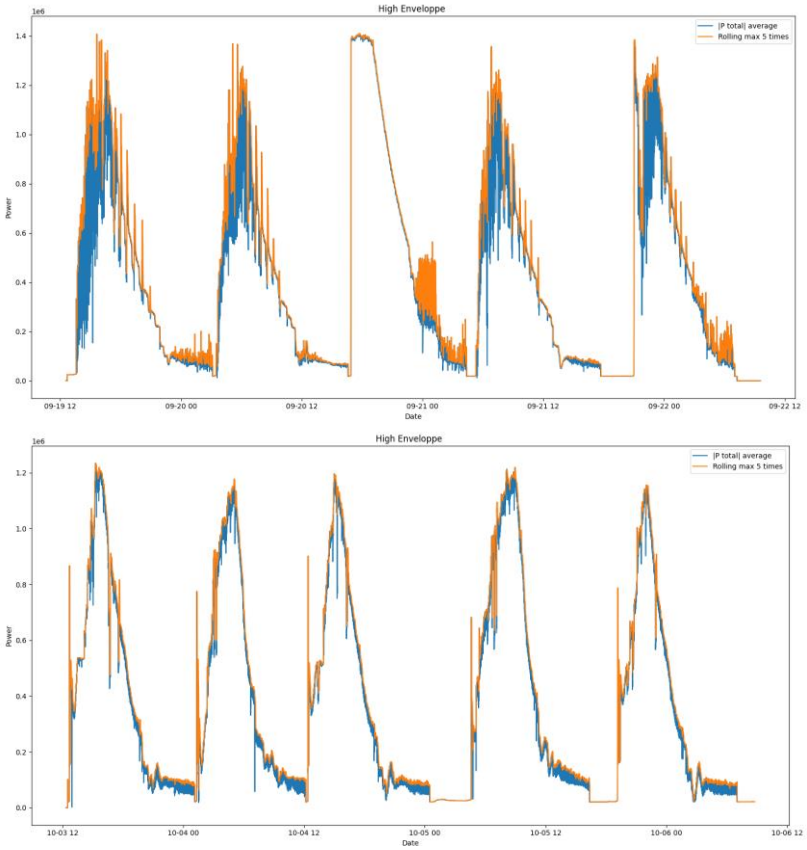


Figure 18 : Results of high envelope with rolling window on both datasets

4.5 Custom synthetic data generation

The Prosys Simulation Server was very useful for the OPC-UA part. However, the data generated allows only a minimal personalisation and cannot produce datasets that really fit the needs of this project. Since only a couple of measurements were provided, more data was needed to test cycle detection algorithms. Therefore, some time was taken in order to generate synthetic data, based on the received datasets.

As discussed in section 3.4, there are a lot of different ways to generate synthetic data. However, since the aim of this work is not data forecasting, but data analysing, it does not make sense to use these techniques. Furthermore, as proven in the last section, the data is not linear, which rules out most of the usual data forecasting algorithms. Therefore, a custom synthetic data generation has been performed.

4.5.1 Synthetic power data

In order to create valuable data, it is necessary to identify its main characteristics. In this case, the signal roughly looks like a half-wave rectified sine, but with a variable period and amplitude. The period depends on the weight of the load placed in the oven, and the amplitude on the alloy aluminium heated. After looking at the provided data, the conclusion was made that the amplitude has to be between 956,000 Watts and 1.56 kWatts. The length of a cycle usually falls between four and six hours. Since the generated data aims to be used as a basis to develop real time machine cycle detection, a cycle of this period is too long. Consequently, it has been decided to generate cycles with a period between one and three minutes. These parameters are then used to create a sinus with a variable amplitude and period. The idle part of the signal, representing the time between two loads, is

generated on the negative part of the sine. Whenever the sine has a negative value, it is randomly substituted by one between zero and 100,000. This last value was chosen as the threshold for the idle phase, based once more on the provided data. Finally, to simulate the noise, every other point is updated by adding a random variation within a range of a quarter of the original value. The code developed on this idea allows to generate as many cycles as wanted, with every cycle having a different amplitude and period. This allows to form a large dataset.

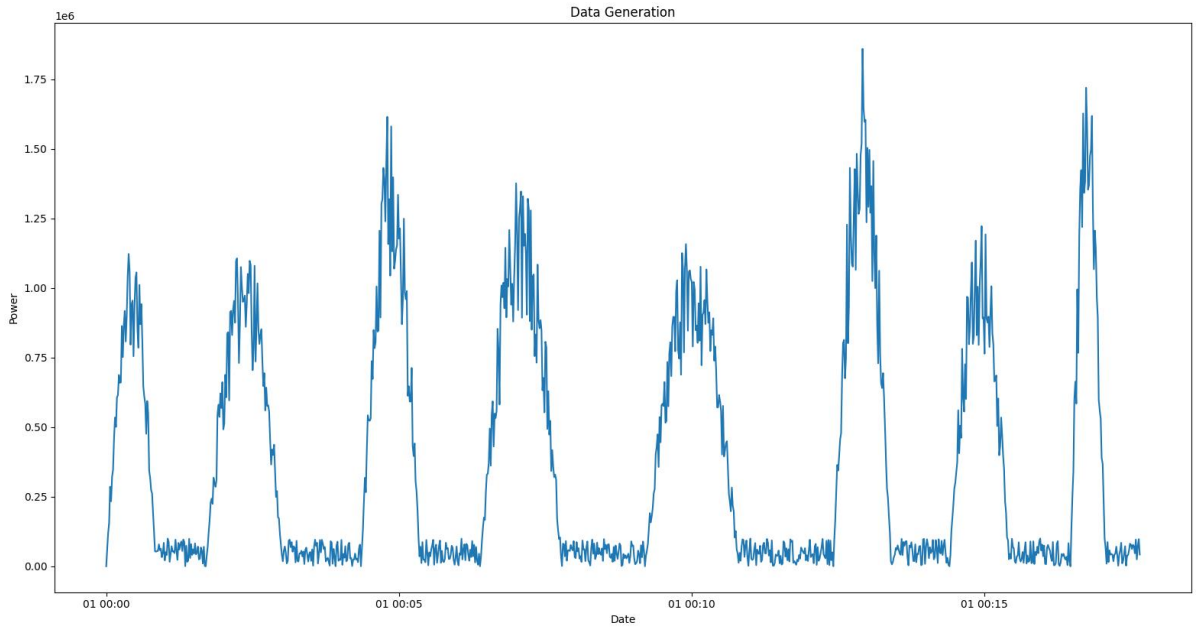


Figure 19 : Synthetic Power data

4.5.2 Synthetic temperature data

The same analysis was conducted for the temperature data. Even though more data was available for temperature, it was very quickly established that the power and temperature had to be linked in order to have a coherent visualisation. That is where the necessity of generating synthetic temperature data came from. Given that no datasets including temperature and power were supplied, it is challenging to describe their correlation precisely. In addition, the temperature data does not provide any information about the cooling of the oven. Therefore, some assumptions were made to generate the data. Again, the generation process is based on observations of the original signal's behaviour. Similar to the power, the signal can be thought as a sine. This time the sinusoidal behaviour can clearly be seen in the ascending phase of the signal. The temperature's amplitude is based on the alloy of aluminium. The correlation between the alloy and the temperature was given by Constellium. Based on this information, the amplitude of the generated data is randomly set between 360 and 580 degrees Celsius, while the period is the same as the power data. As soon as the sine has reached its peak, the temperature signal keeps this value until the power reaches its idle phase. It then drops, in a linear way, to reach zero degrees when the next cycle begins. Since the temperature data is inherently smooth, no noise was added.

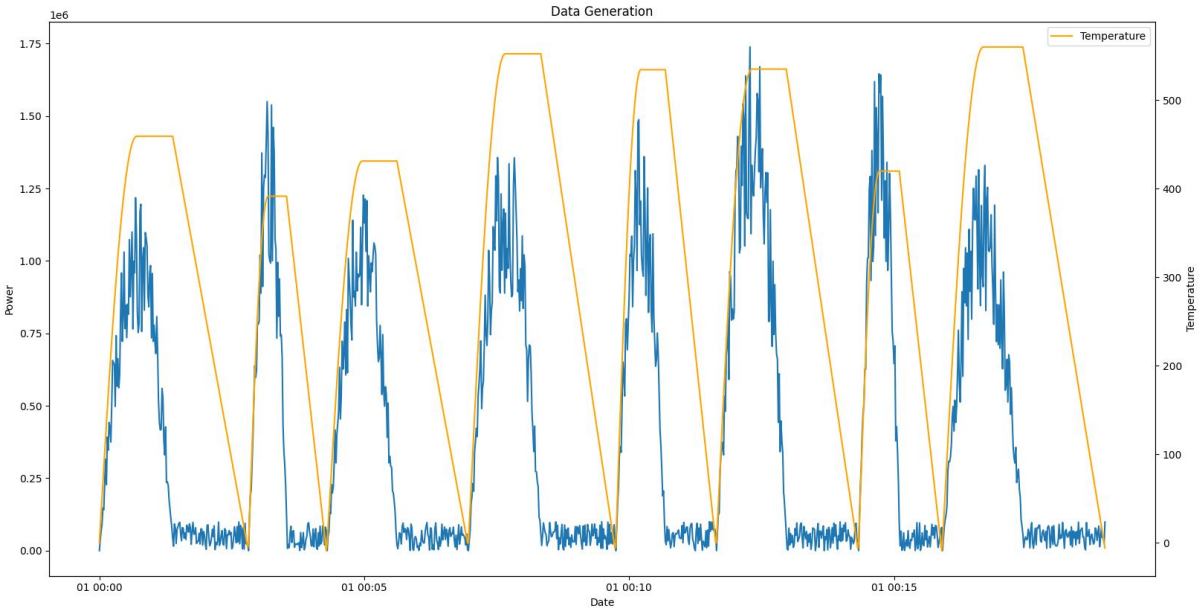


Figure 20 : Synthetic power and Temperature data

Although the datasets generated in this manner proved highly beneficial throughout the project, in particular to produce data, they are not entirely reflective of an optimal scenario. Notably, they lack an offset between power and temperature, which should mirror the oven's inertia – a power change does not immediately affect temperature. Moreover, while the synthetic power data visually resembles actual data, they do not precisely match in behaviour. Real power datasets exhibit quicker ascents than descents. Implementing these nuanced differences would have consumed excessive time and effort. As a result, algorithm results tested on these synthetic datasets may not necessarily align with real dataset outcomes. Hence, each algorithm is initially assessed using the limited received datasets.

4.6 Local OPC-UA Server

As mentioned earlier, the Prosys OPC-UA server does not support custom data usage. Therefore, it became a necessity to create a local OPC-UA server. To fulfil this requirement, the 'opcua' Python library [21] was employed to set up the local server. The implementation of the server proved to be straight forward and can be found in the ‘opcUAServ’ project.

First of all, the URL for the server is setup on ‘opc.tcp://localhost:4840’, with port 4840 being the default OPC UA port. Subsequently a namespace ‘Constellium’ is created to hold the root node containing all relevant variables, including power and temperature values for each oven. Every second, each variable is updated with a new value, which can originate from either the datasets provided by Constellium or synthetically generated datasets. Naturally, every oven has a different dataset. Furthermore, every variable’s historian mode is activated, allowing clients to read past values.

To verify that the proper functioning of the server, the UaExpert full-featured OPC UA Client is used [34]. This enables comprehensive testing of the server’s performance and ensured that it effectively served as a reliable data hub for the project, while the real server is not active.

4.7 Data acquisition from the dashboard

The final segment of the data acquisition process involves understanding how the information from the server arrives to the dashboard. The main classes used for this are 'Dataset' and 'opcClient', which were provided at the beginning of the project. Although some minor modifications have been implemented, including the addition of the 'getLastTime()' and 'readHistoryData()' functions within the 'Dataset' class.

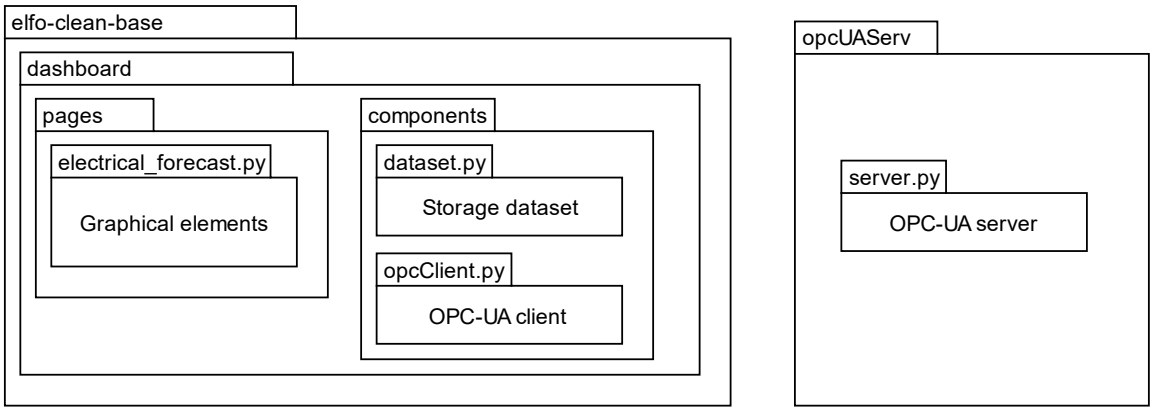


Figure 21: Global view of the packages for the data's acquisition

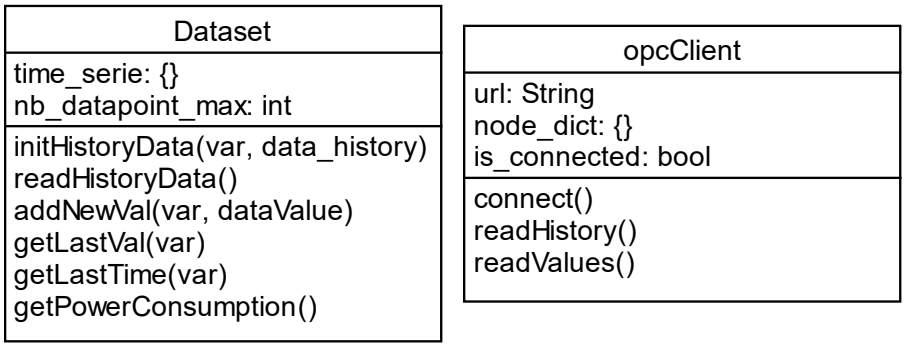


Figure 22 : Class diagram for Dataset and opcClient

Figure 21 provides a global view of the packages necessary for the data acquisition. It is important to understand that the 'elfo-clean-base' and 'opcUAServ' are two completely different program. They can run on two different computers.

The following sequence diagram presents how the system reads the historian values when the page is launched.

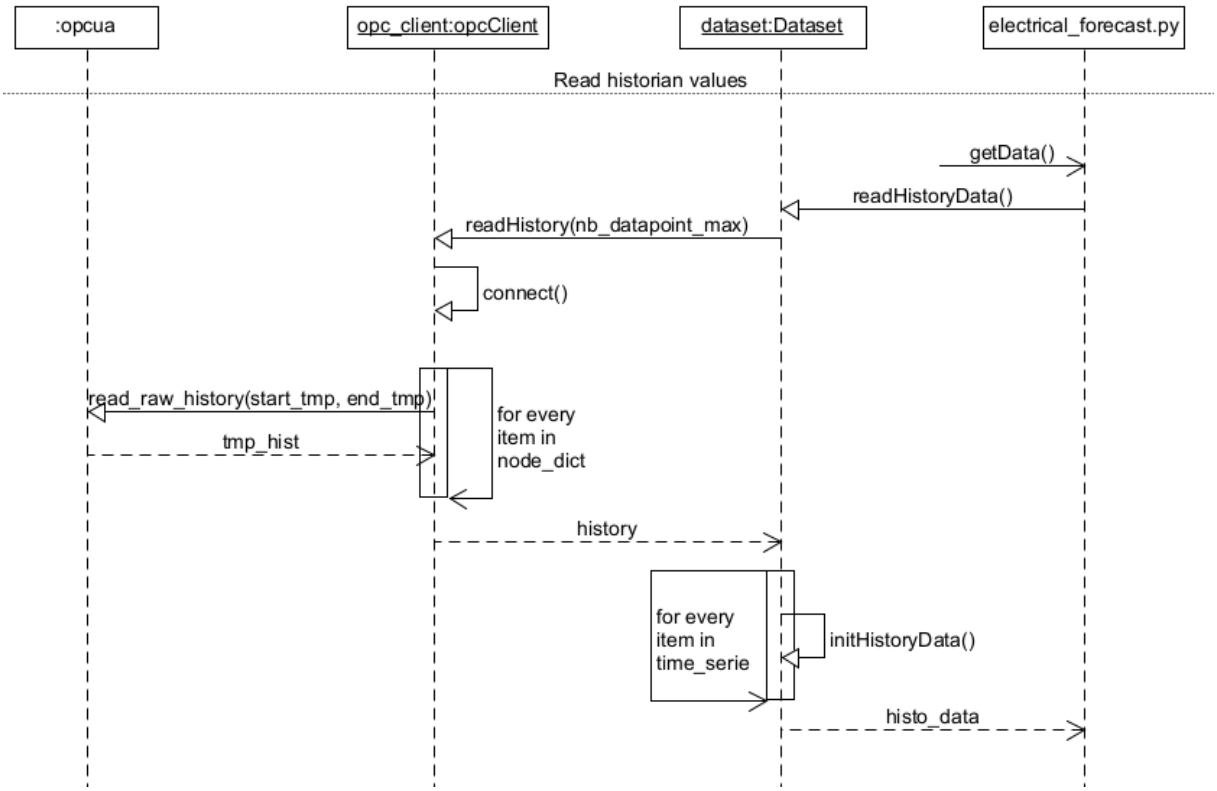


Figure 23 : Reading of historian values – sequence diagram

In summary, when the page starts, a request is issued to read a certain amount of historian points from the server. These datapoints fill a dictionary within the 'opcClient' class, that will be used to populate another dictionary in the 'Dataset' class. Finally, this information is then transmitted to the page.

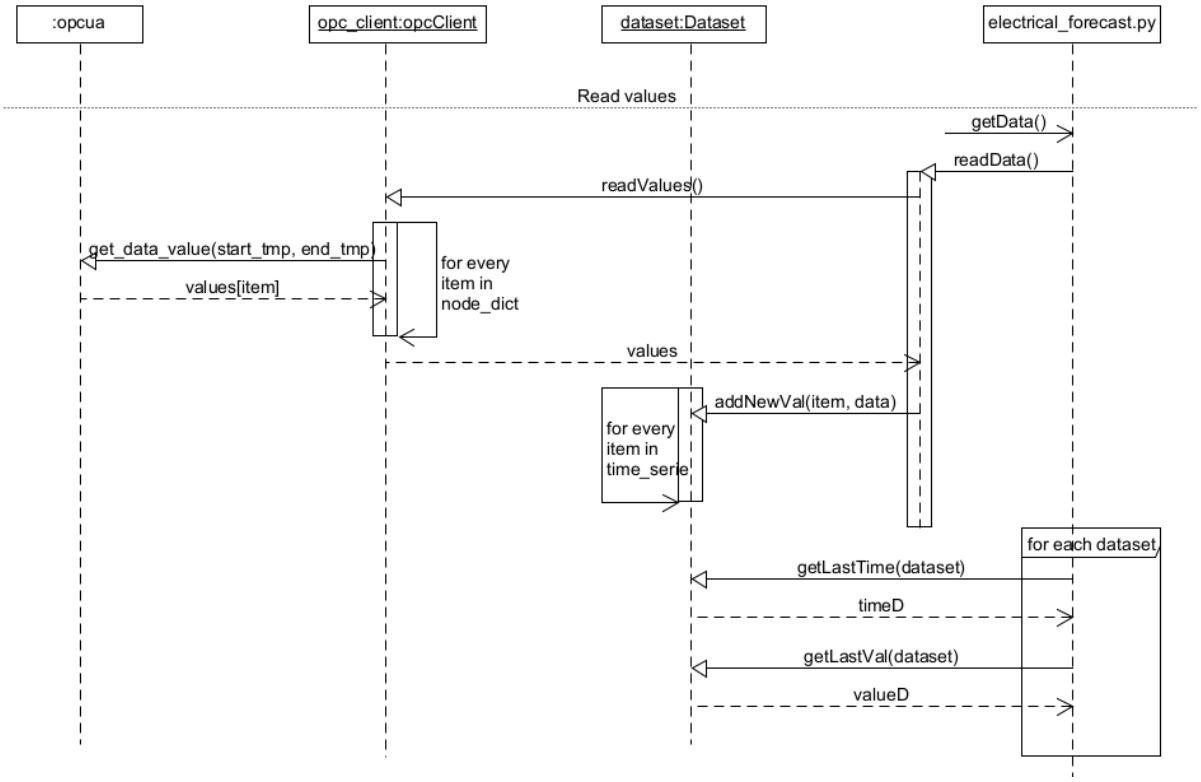


Figure 24 : getting data from the server – sequence diagram

The same kind of mechanism happens the rest of the time, except that only one value for each dataset is read.

Upon different tests, it has been remarqued that when the window is not on focus, the polling of the server is not regular. Indeed, the timer that trigger the reading of the server becomes lazy and does not count the time correctly. The only solution to get the missing datapoints is to reload the page. By doing this, every structure holding data points are reset. The historian for every dataset is read, allowing to get any missing value. This solution is implemented by the 'readHistoryData()' function in the Dataset class.

5 Dashboard

The dashboard constitutes the primary focus for users within this project. While users are not required to grasp the intricacies of data collection or analysis procedures, presenting the results of these operations in a clean, straightforward, and user-friendly manner holds significance. This section delves into the process of creating this interface and outlines the developmental steps required to achieve the final product.

But first to facilitate the comprehension of further development, here is a global representation of the software's architecture, and a global view of the organization of packages used during the project.

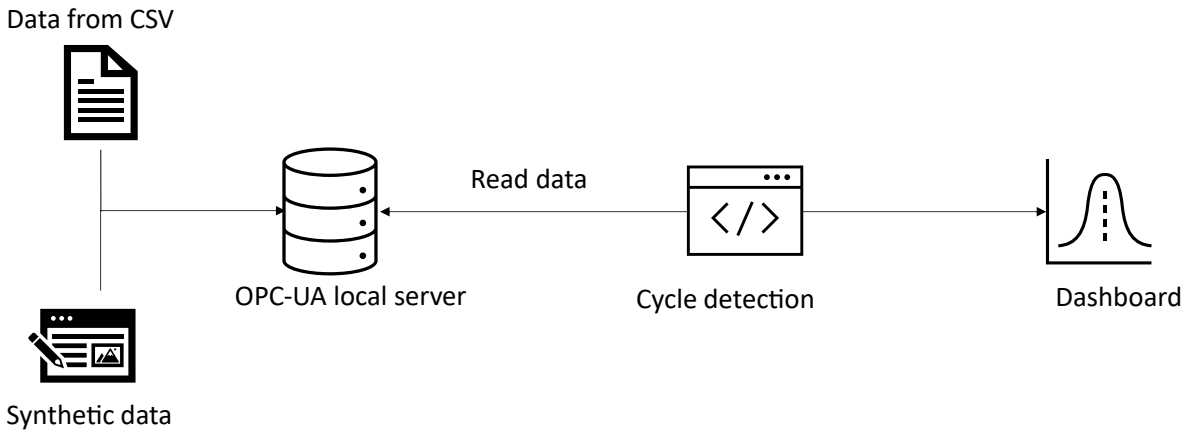


Figure 25: Global software architecture

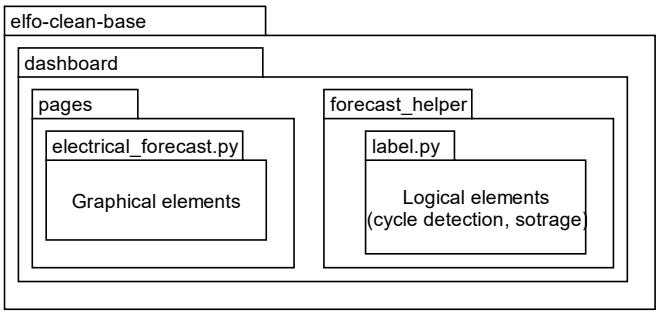


Figure 26: Global view of useful packages

5.1 Main libraries

The dashboard skeleton that is shown in Figure 27 was developed by Steve Devènes, using Python. It combines different libraries to achieve an efficient visualization.

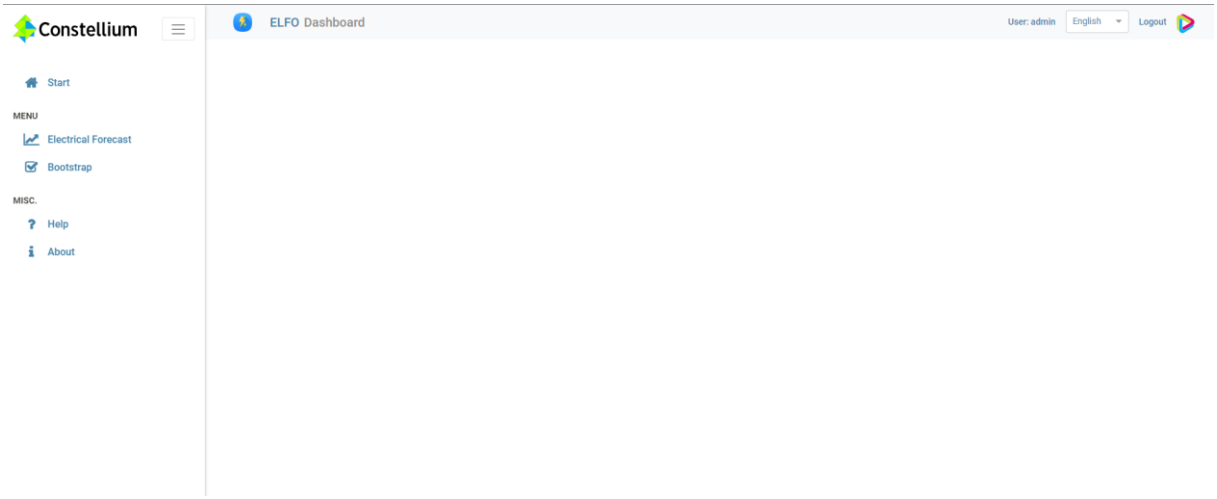


Figure 27 : Dashboard received at the beginning of the project

5.1.1 Plotly Dash

The primary visualization library used in this project is Plotly Dash. This framework is very popular for developers that need to create a website while displaying data. It is powered by Plotly, an open-source graphing library. The main difference between these two technologies, lies in the fact that Plotly is used to create standalone graphs, that can be exported as static files. On the other hand, Plotly Dash's graphs can be immediately exported as an HTML file.

One of the key points of Plotly Dash is its component-based architecture. The layout of a Dash app is based on multiple components, representing every element seen. These components are based on an HTML-like display. For instance, 'html.H1(children='Hello World') 'component will generate the following HTML tag in the final app: "<h1>Hello World<\h1>". Obviously, the developer can add styling effects, similar to classic HTML files. In order to generate higher-level components, such as interactive graphs or dropdowns for instance, there is the Dash Core Components module. Such elements are generated via JavaScript, CSS, and HTML, through the React.js library [35].

To complete the dynamic part of the application, Plotly Dash uses the concept of callbacks. A function that is accompanied by the '@callback' decorator will change the state of an element. When using the '@callback' decorator, the developer must provide a set of input and a set of output. These elements are properties of a component. Every time that an element changes, every function that has this element as an input will be executed. The result generated will modify the output listed [36].

```
@callback(Output("timer", "disabled"), Input("acquireServ", "n_clicks"))
def activeServ(n):
```

Code 1 : Example of the callback form

The example in Code 1 is taken from the project's code. We can see that every time the button with the id "acquireServ" is clicked, the function "activeServ" is executed. The result of this execution will affect the "disabled" characteristic of the component with the id "timer". In this case, a Boolean return value is expected, to determine if the timer is active or not.

5.1.2 Pandas

Pandas is a powerful and popular open-source library for data manipulation and analysis in Python. It provides easy-to-use data structures and data analysis tools, making it an essential tool for data

scientists, analysts, and researchers working with tabular and time-series data. Pandas is built on top of the NumPy library, which provides support for efficient numerical computations in Python [37].

The predominant data structure employed in this project is the DataFrame. This structure facilitates the presentation of data in a two-dimensional layout, like a spreadsheet. It organizes data into ordered columns with varying types. Furthermore, a DataFrame includes an index that aids in item access. Pandas offers an array of methods for managing missing data, converting data types, resampling, grouping, and more, enhancing its versatility and user-friendliness[38].

5.2 Getting to grips with the tools

Some time was taken to get to grips with these elements. To do so, the decision was made to display some data relevant to the project. Some measurements were taken in September and October of 2021 in the oven from Constellium. The aim of this first implementation was to display the data properly, with corresponding time for each data point. In order to start small, only the "|P total| average" measurement is considered.

This was done using two functions. The first one is "getData()". When the user clicks the "Get Data from file" button, the function will fetch the specified .csv file and store each column in a column of a DataFrame. To correctly display the date and time, these two columns are merged and converted to a datetime format. Finally, the DataFrame is converted to a dictionary. This way, it can be stored and used by other functions. Furthermore, the function produces a second output that activates a dropdown menu.

The second function is "displayData()", which is executed when the user selects the 'Power' option of the dropdown menu. The function fetches the dictionary stored earlier and converts it back to a DataFrame. A Butterworth filter is applied to the data to remove some of the noise. Then a figure is created, specifying which columns of the DataFrame are represented on the x and y-axis. This figure is used to generate a graph, which in this case plots the measured and filtered data according to the measuring time. The results are presented in the following figure.

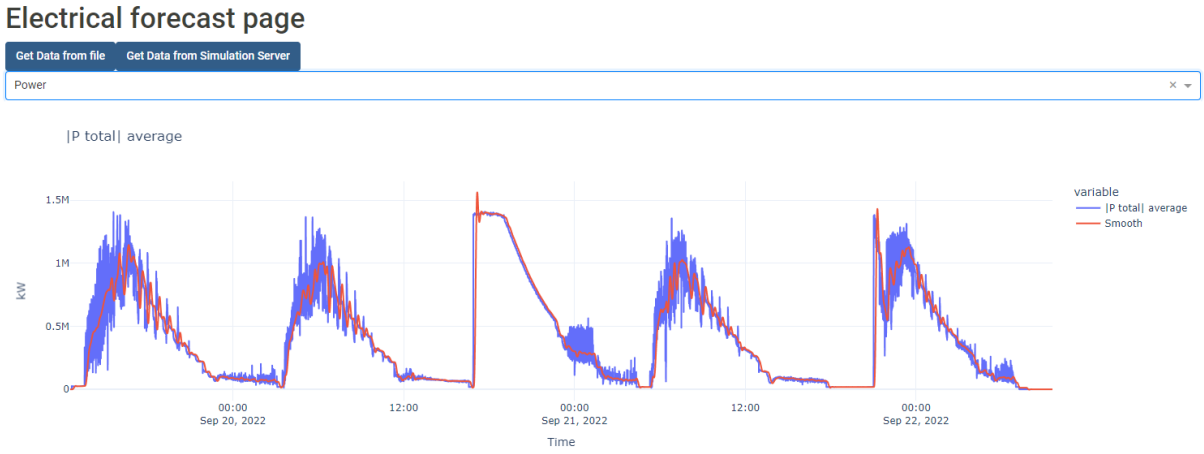


Figure 28 : Result of a .csv file display

The main source of problems in this part of the project comes from type conversion. First of all, since measurements were taken by a European device, the separator between the integer and decimal part of numbers is a comma. However, when programming, double and float use a point as a separator. Then, a significant amount of time was spent on finding the appropriate way to ensure proper data conversion. For instance, when using dcc.Store to save data, it is not possible to use DataFrames, as

only object that can be parsed as Json can be saved. However, most of the time it is way easier to handle pandas data structure when applying filters or using them as parameters to create a figure. Therefore, a lot of conversion from dictionary to DataFrames, and the other way around, was necessary.

The final implementation does not have the option to choose where the data comes from, as the OPC-UA server has been implemented. Furthermore, the dropdown menu has also disappeared since it has been decided to display only the power consumption and the temperature.

5.3 More advanced options

As the project progressed, the necessity for additional options arose in order to enhance the efficiency of the dashboard.

5.3.1 Multiple ovens display

When the project started, the priority was put on trying to understand how to use the libraries and the given code. Therefore, the dashboard displayed only one signal, such as depicted in Figure 28. However, for the purpose of achieving a more realistic representation, it became essential to visualize every oven within the homogenization sector. This endeavour presented certain challenges, the first among which was aesthetic in nature – necessitating the identification of an effective and visually pleasing approach for displaying all ovens. Secondly, a solution was required for adequately storing information specific to each oven.

The first challenge was addressed by employing the dbc.Tabs element. This component serves as a container for a series of dcc.Tab elements, with each tab corresponding to an individual oven. The dbc.Tabs component as an ‘active_tab’ property that can be leveraged as an input for a callback function. This feature enables the dashboard to dynamically adjust its display based on the oven selected by the user.

Unfortunately, this improvement led to the second problem. Until this point the code was structured as follows:

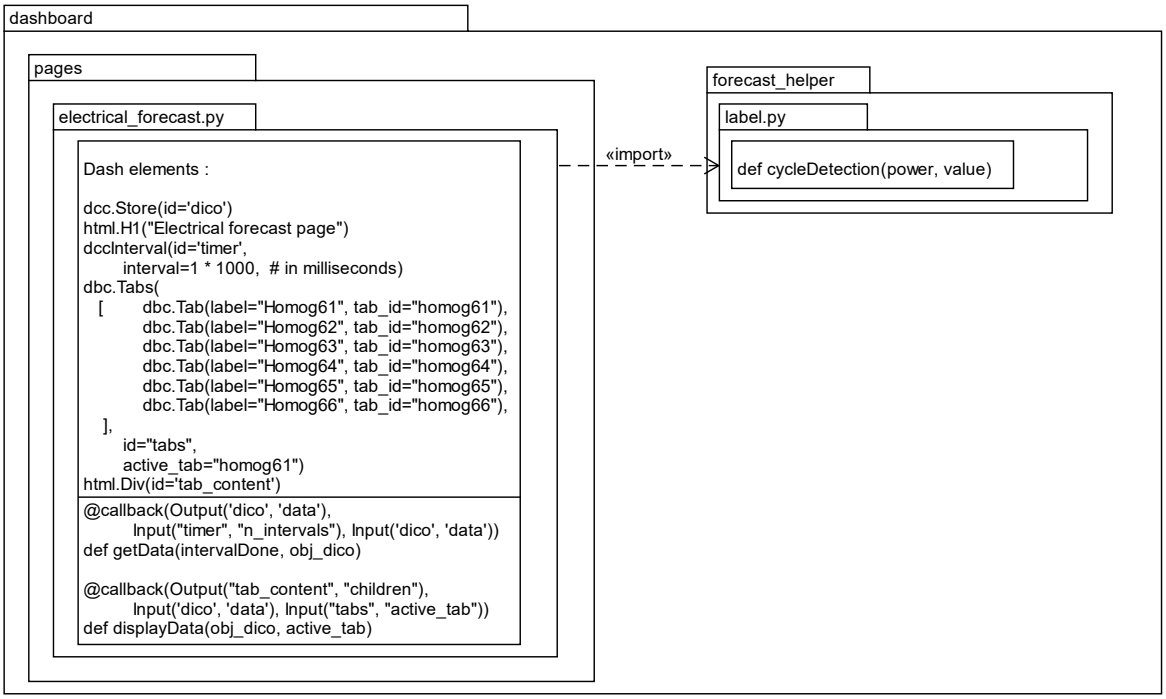


Figure 29 : package diagram before class (note that only the relevant files and folders are represented)

The page's structure is established in the 'electrical_forecast.py' file, where each component is defined. These components play a crucial role in influencing the operation of the two callback functions responsible for retrieving and displaying data. Upon the timer's completion, the 'getData' function is invoked. This function fetch data from the server and invokes the 'cycleDetection' function present in the 'label.py' file. After that, the data is stored in the dcc.Store component, thereby triggering the execution of the 'displayData' function.

However, this mechanism encounters a challenge when dealing with diverse datasets for each oven. Certain features essential for cycle detection are dataset-specific and require a connection to the respective dataset. To resolve this complication, an object-oriented approach was adopted. From this point forward, each oven is treated as an instance of the 'ComputeOven' class.

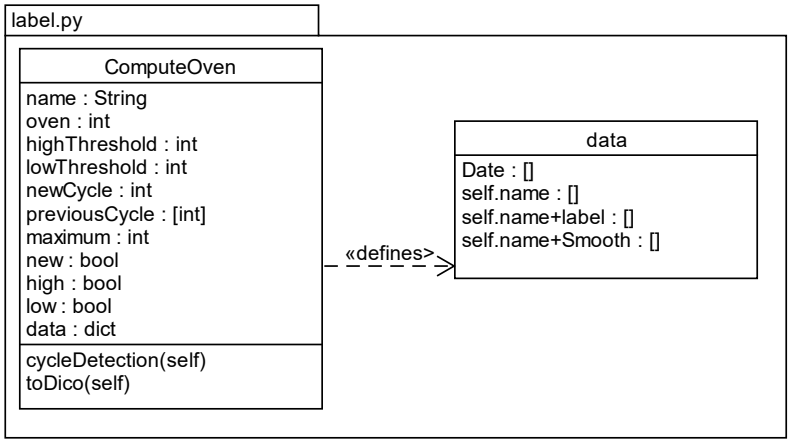


Figure 30 : ComputeOven class diagram V1

Unfortunately, storing objects in the dcc.Store component is not feasible, which necessitated a series of type conversions. Dcc.Store holds a dictionary with a dictionary for each oven. Each dictionary holds the information necessary to construct a 'ComputeOven' object.

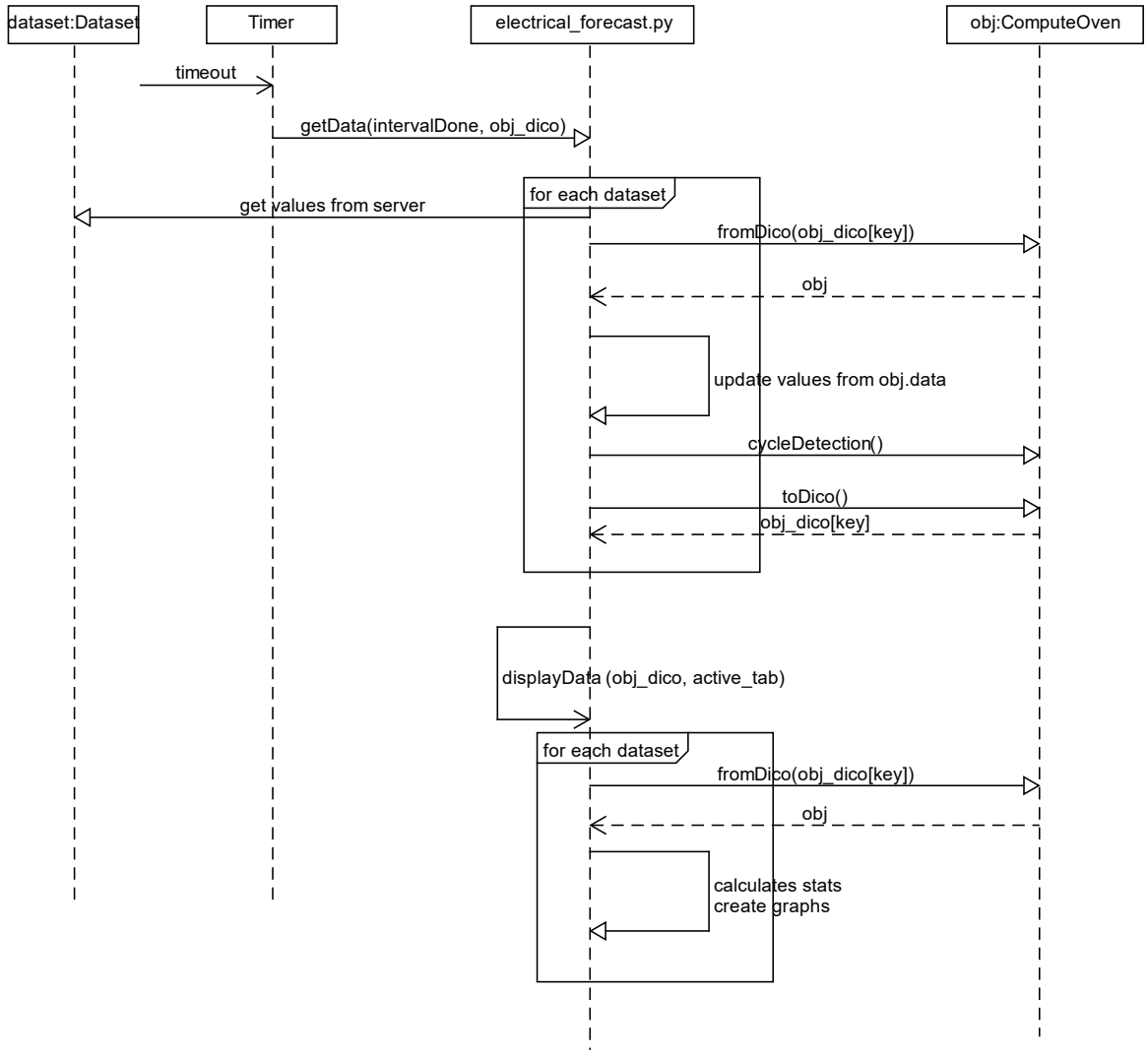


Figure 31 : Sequence diagram for that focuses on dictionary-object conversion

In Figure 31, the sequence of conversions can be observed. Initially, a dictionary is employed to instantiate an object of the ComputeOven class. Once the computations are completed, the object is reconverted into a dictionary format to be stored in the dcc.Store components. This transformation in the dcc.Store prompts the 'displayData' function, where a similar process takes place. The dictionary associated with each dataset is converted back into an object format to facilitate easier access to its components. This approach aids in calculating statistics and generating the graphs that are subsequently showcased on the dashboard.

5.3.2 Summary tab for every oven

To support the additional objective of offering energy usage optimization suggestions during machine cycles, an extra tab has been incorporated. This tab serves as a foundation for this optional aspect. To gain an overview of the overall consumption in the homogenization area, a display showcasing the

energy usage of the six ovens has been devised. Additionally, a bar chart illustrates the contribution of each individual oven.

Upon selecting the 'HomogAll' tab, all power datasets are merged to generate a new dataset encapsulating the collective consumption. At the same time, a ‘go.Bar’ figure is created for each dataset. By incorporating the 'barmode='stack'' parameter within the layout specifications, a consolidated bar is attained, representing the consumption of each oven for every time stamp.

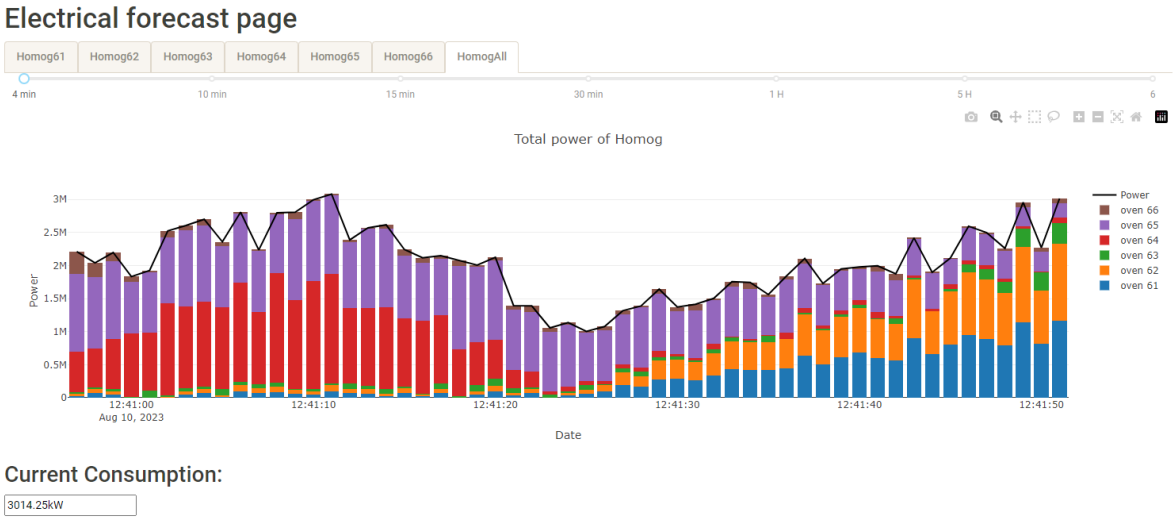


Figure 32 : Tab displaying the consumption of every oven

Even if no time was available to pursue any optional objectives, this inclusion can still provide valuable insights to the operator, highlighting the potential need to adjust the load of a new charge.

5.3.3 Data storage management

Another critical challenge that required attention was data storage management. Without proper storage or occasional data deletion, the data structures for each oven could accumulate endlessly, potentially hindering page performance. Furthermore, it was unfathomable to just let data disappear. Thus, an efficient data storage solution was imperative.

The initial approach to data storage revolved around the number of displayed cycles. When the dashboard exhibited more than four cycles, the earliest cycle was archived in a CSV file and removed both from the data structure and the dashboard. While this approach functioned, the complete removal of a cycle from the dashboard raised aesthetic concerns. Another critical point was the behaviour of the system if for any reasons no more cycles were detected.

The second iteration of data storage aims to address these challenges. The primary concern, which pertains to storage based on the number of cycles, has been the foremost focus. Rather than tracking the count of cycles, the approach now centres around the duration of time displayed, determining the optimal points for data storage.

The aesthetic concern was effectively addressed by implementing a buffer mechanism. When the displayed data points surpass the designated time limit, the initial data point is temporarily stored in a buffer. This approach enhances the overall smoothness of data presentation. Once the buffer accumulates a specified number of data points, it is then stored in a CSV file for archival purposes.

The user has the flexibility to determine the time threshold for data storage through the utilization of a dcc.Slider component. Each increment of the slider aligns with a specific time frame. Within each

dataset, a variable is integrated to represent the user's selected time threshold. Whenever a new datapoint is acquired, just before the cycle detection phase, the system cross-references this value with the previous datapoint. If the two values differ, the pertinent variable is updated accordingly. Upon completion of the cycle detection, the current displayed signal's duration is computed. If this duration surpasses the user-specified time threshold, the buffer begins to accumulate data.

While this version seems to resolve every problem, it unfortunately still relies on the number of identified cycles. Indeed, the trigger to empty the buffer in the CSV file must be the end of a cycle. As explained later, the detection cycle algorithm stores the index of the current and previous cycles. These indexes are pivotal for multiple statistical calculations and necessitate constant updates with each new buffer addition. Maintaining accurate values becomes more streamlined when the transfer to the CSV file aligns with the completion of a full cycle within the buffer. Given that data storage is not the primary focus of this thesis, no further effort was allocated to identifying a solution to this challenge.

6 Cycles detection

This section focuses on the different algorithms that were tested in order to perform a machine cycle detection. As in 3.2, where the definition of a machine cycle is detailed, only the power data will be considered in this chapter. The methodology used is reflected by the structure of the section. For each algorithm a background and explanation are provided, then it is tested on the CSV containing power data, especially September's dataset. In this configuration, no real time is involved. This setting allows to see how the algorithm performs and decide if it fits the requirements. If it does, it is then tested on the second power dataset and on a synthetic dataset. Finally, the algorithm is implemented on the dashboard to determine if it is suitable for a real time environment.

In order to ascertain the efficiency of an algorithm, a foundational body of work is essential for conducting a meaningful assessment. A graphical representation outlining the preliminary expectations has been devised to provide a visual framework for anticipating the algorithm's outcomes.

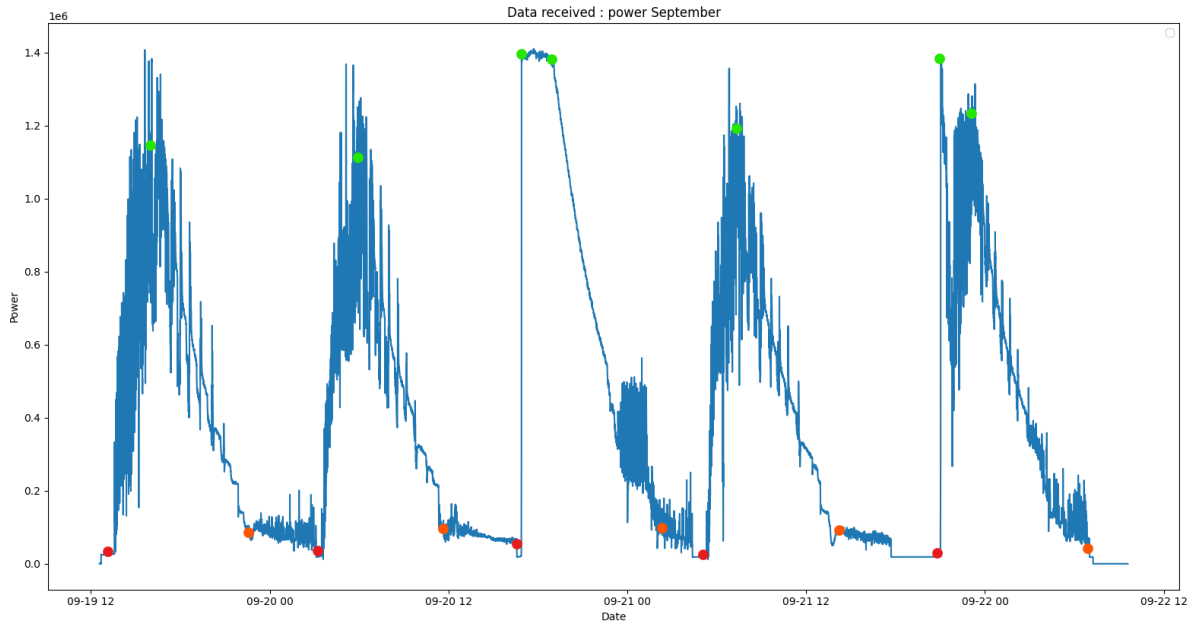


Figure 33 : Power data from September, expected output from algorithms

In Figure 28, the red dots symbolize the defined cycle criterion mentioned earlier. To accommodate a slightly more flexible approach, outcomes aligning with the orange dots will also be accepted as indicative of a cycle. Additionally, certain algorithms may decompose the signal into distinct phases, denoted by the inclusion of green dots, which would also be deemed as satisfactory outputs.

The same kind of reflection is applied the second dataset.

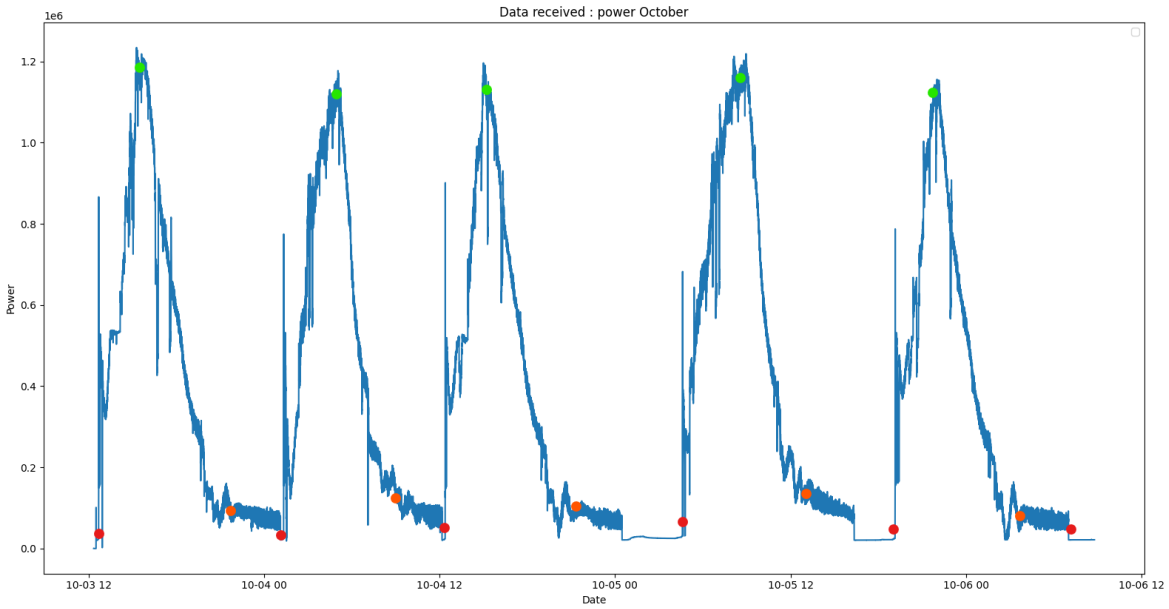


Figure 34 :Power data from October, expected output from algorithms

6.1 Change point detection algorithm

6.1.1 Concept

Change point detection (CPD) algorithms are powerful tools in the field of time series analysis that help identify points in a sequence of data where there is a significant change in the underlying pattern. These changes, also known as change points, mark shifts in the data behaviour, which can have various implications in different domains.

CPD can signify diverse events such as shifts in trends, abrupt transitions, outliers, or other structural changes in the data generating process. For example, in a power consumption time series, changepoints could represent shifts between different operational states of a machine.

CPD algorithms can be classified in multiple categories. Firstly, they can be categorised as either online or offline algorithms. The first category is suitable for applications needing real-time constraint, performing updates based on previous results. On the other hand, offline algorithms are more accurate but introduce delay due to their consideration of the entire dataset at once [39].

Originally, the aim of CPD was to find a single point in the entire time series. This approach is interesting when searching for one event or transition. However, when dealing with more complex datasets, it became evident that a signal could hold multiple change points. This can lead to the segmentation of the time series, in order to get one change point in each sub-time series [40].

Two main approaches can be considered when implementing a CPD algorithm, the binary segmentation, and the Bayesian approach. The first one is one of the most popular algorithms for offline detection. This algorithm, first introduced in the seventies [41], is based on the divide and conquer strategy. The time series is recursively split into segments, and change points are detected within each segment. If a segment is found to contain a change point, it is further divided into smaller segments for more refined analysis. This process continues until no more change points are detected. This strategy allows to extend a single change point detection to a multiple one [42].

Bayesian change point detection is a statistical methodology. Rooted in Bayesian principles, this widely used approach combines prior knowledge with observed data to infer the locations where these changes occur. It starts by formulating prior distributions that represent beliefs or expectations about possible change point locations. As new data is observed, the likelihood of the data under different change point scenarios is evaluated. By applying Bayes' theorem [43], that describes the relationship between conditional probabilities of events, the prior beliefs are updated to yield posterior distributions, which encapsulate refined insights into the change point positions. Bayesian change point detection allows for the quantification of uncertainty and the incorporation of prior information, enabling a nuanced understanding of when and where transitions in data behaviour take place [39] .

Both approaches can fit the requirements of this projects since they can be used in a real-time environment and perform well with multiple change points.

6.1.2 Results & Conclusion

In order to test both the binary segmentation and Bayesian's approach, the 'ruptures' Python's libraire is used [44]. The code producing the following results can be found in 'testDataAnalysis → testChangPoint.py → chgPT()'.

Main settings, apart from the chosen model, come in the 'predict()' function. This function returns the optimal number of breakpoints, representing the number of change points. The 'pen' argument, representing the penalty value, controls the trade-off between the number of detected change points and the quality of fit. The higher the penalty value is, the more selective the algorithm is. There also is the 'n_bkps' argument that allows the user to choose the number of desired change points. However, this option is only available for the binary segmentation model.

First of all, it is important to acknowledge that the Bayesian model is a very demanding algorithm that generates a lot of calculations. It was therefore necessary to perform a down sampling of the data to one point every minute instead of every 10 seconds, which could constitute a weakness. Furthermore, changing the penalty value did not seem to influence the result, making the algorithm impossible to tune.

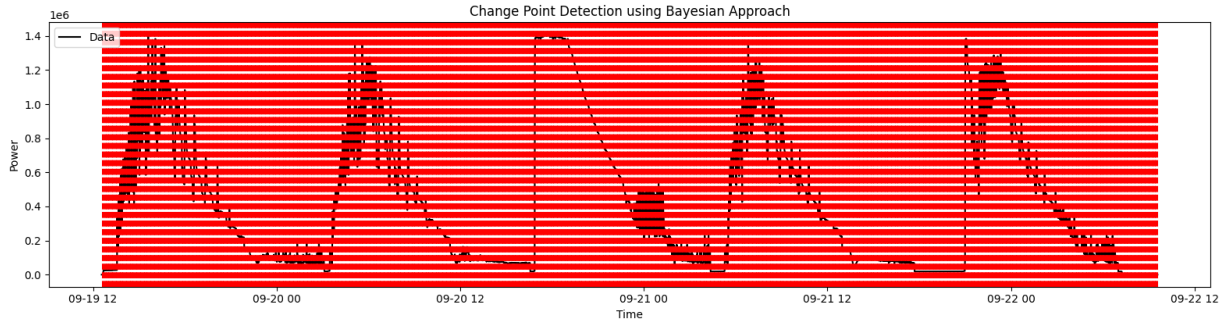


Figure 35 : Bayesian approach in change point detection, changing points represented by a vertical dashed line

As one can see in Figure 35, it seems that every point represents a change point, providing no insight on the data. Even applying filters or a bigger down sampling does not change anything.

On the other hand, the binary segmentation is a bit easier to work with. To get closer to the desired result, the penalty values is set to 2100. This setting seeming suspiciously high, it was replaced by the argument ruling the number of expected breakpoints.

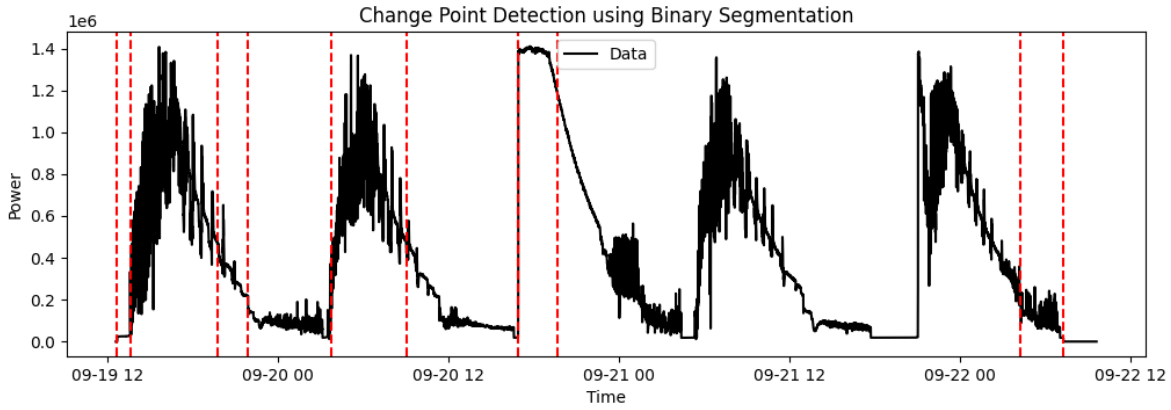


Figure 36 :Binary Segmentation model of change point detection, n_bkps=10, on raw data

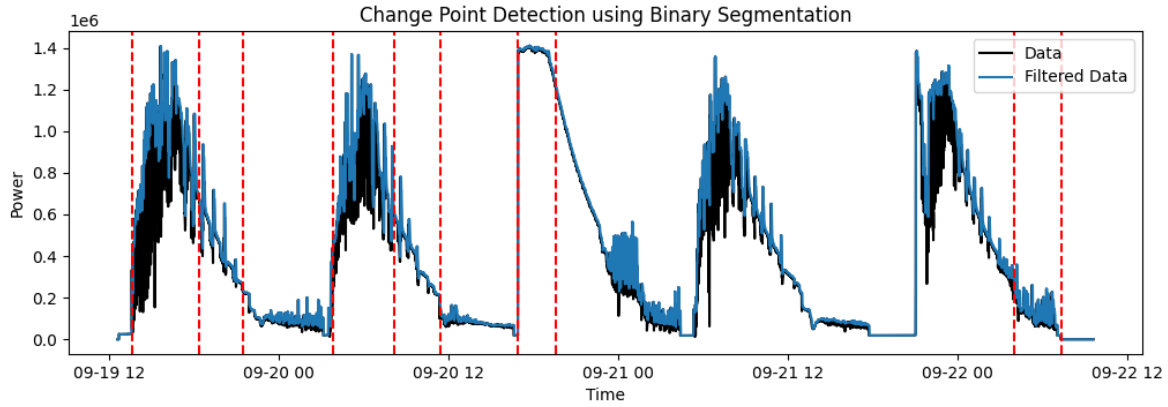


Figure 37 : Binary Segmentation model of change point detection, n_bkps=10, on filtered data (rolling window)

The algorithm has been tested on raw and filtered data, demonstrating a slightly better performance with filtered data. Unfortunately, results do not really align with the expected output. Since there does not appear to be a way to tune the algorithm to get the wanted results, the change point detection algorithm is not the right candidate for this project.

6.2 Clustering with K-means

6.2.1 Concept

The objective of this project is to segment a signal into distinct cycles. Given this goal, delving into clustering algorithms is a logical step. Clustering is a fundamental concept in the field of unsupervised machine learning. It involves grouping similar data points together based on some measure of similarity or distance, without the use of explicit labels or predefined categories. These groups are called clusters. The goal of clustering is to discover hidden patterns, structures, or relationships within a dataset.

As the aim of clustering is to form groups that contain similar data but are different from each other two measures are often used: the similarity or the distance measure. Similarity measures quantify the degree of likeness or resemblance between two data points. A high similarity score indicates that the points are very similar, while a low score suggests they are dissimilar. Similarity measures are often used in scenarios where higher values mean stronger similarity. On the other hand, a distance measure is the opposite. It quantifies the dissimilarity between two points, therefore the lower the results the stronger the similarity is [45].

One of the most commonly used algorithms is the k-means clustering algorithm. This algorithm offers a systematic approach to grouping data points into distinct clusters, with the number of clusters (denoted as 'k') determined by the user. Initially, k points are chosen randomly from the dataset to serve as the initial cluster centres. These centroids represent the heart of each cluster. Subsequently, each data point is assigned to the cluster whose centroid it is closest to. This assignment is often made based on the Euclidean distance between the data point and the centroid. As data points are allocated to clusters, the centroids of these clusters are updated to reflect the mean of the data points assigned to them. These two steps—data point assignment and centroid update—are iteratively repeated until a predefined stopping criterion is met. Common stopping criteria include reaching a maximum number of iterations or achieving a state where the centroids cease to change significantly [46].

6.2.2 Results & Conclusion

The code to test this algorithm is simple and can be found in 'testDataAnalysis → testClustering.py → testCl()'. The 'sklearn.cluster' library offers the 'KMeans' class, which facilitates the setup of a k-means clustering model [47]. The number of clusters is set to 5 as per the number of cycles expected.

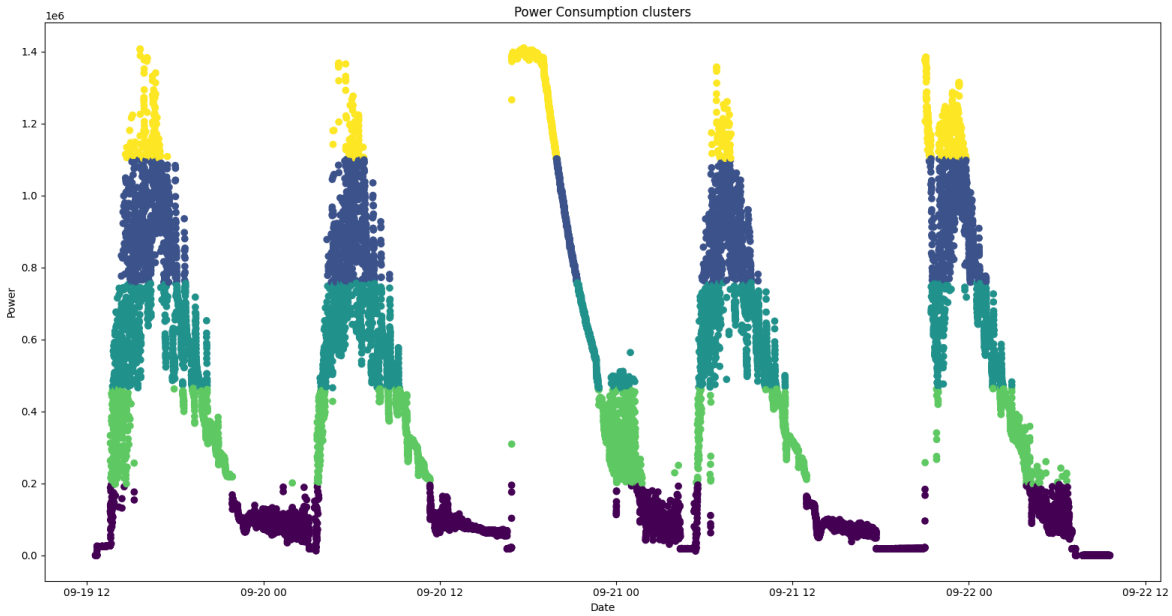


Figure 38 : k-means clustering, k=5, random_state = 666

The results are obviously not congruent, compared to the expected outcome from Figure 33. When thinking about the core definition of clustering, it is obvious that it cannot be applied to this situation. Machine cycles represents the opposite of clustering, the data within each group does not really have anything in common, however, every group has the same kind of structure.

Stepping back from the anticipated outcome, each cluster could potentially represent distinct stages within the heating or cooling process, derived from the cluster sequence. Upon completion of the sequence 'cluster 1(purple) → cluster 2(green) → cluster 3(turquoise) → cluster 4(blue) → cluster 5 (yellow)→ cluster 4 → cluster 3 → cluster 2 → cluster 1,' a cycle concludes. First and foremost, the assurance of this sequence cannot be guaranteed. This is evident from the values around '09-21 00', where data points are prematurely assigned to the first cluster, even though the signal continues to oscillate between clusters 2 and 3. Furthermore, variations in consumption levels between cycles might result in some cycles bypassing a phase if their peak consumption is modest. When a cycle manifests a higher peak, certain data points could fail to align with any cluster. While this scenario is

not depicted here, it is a certainty that such situations could arise, especially considering that the second dataset already exhibits peaks smaller than those in this dataset.

6.3 Cydets package

6.3.1 Concept

The next test comes from the 'cydets' package. The developers that worked on this project have crafted an algorithm to detect cycles in time series, along with their respective duration and depth-of-cycle. This last information represents the minimum height between the minimum value of the cycle and the bordering peaks [48].

The code is based on the 'detect_cycles()'. When called, it pinpoints peaks and valleys in the normalized series, given as an input. Next, the function searches for regions between peaks and valleys that could form cycles. These regions are called "precycles," and their start and end points are calculated. To identify actual cycles, the function removes overlapping precycles, keeping the narrowest ones that are free from overlap.

The depths of identified cycles are calculated, representing the minimum distance between the lowest point of the cycle and neighbouring peaks. The resulting cycle information, including start times, end times and depths, is organized into a DataFrame.

Users can choose to drop cycles with zero depth from the DataFrame. The function then returns the DataFrame, providing users with a clear overview of detected cycles in the input time series.

6.3.2 Results & Conclusion

The implementation of the 'cydets' package can be found in 'testDataAnalysis → testClustering.py → testCydets()'. The first iteration on raw data produced 4179 cycles, which is way too much. On filtered data, the number of cycles is smaller but still consequent, with 470 detections. To address this issue, cycles lasting less than four hours were removed from the results. This choice was made after careful examination of the received data, which clearly demonstrated that cycle durations were consistently longer than four hours.

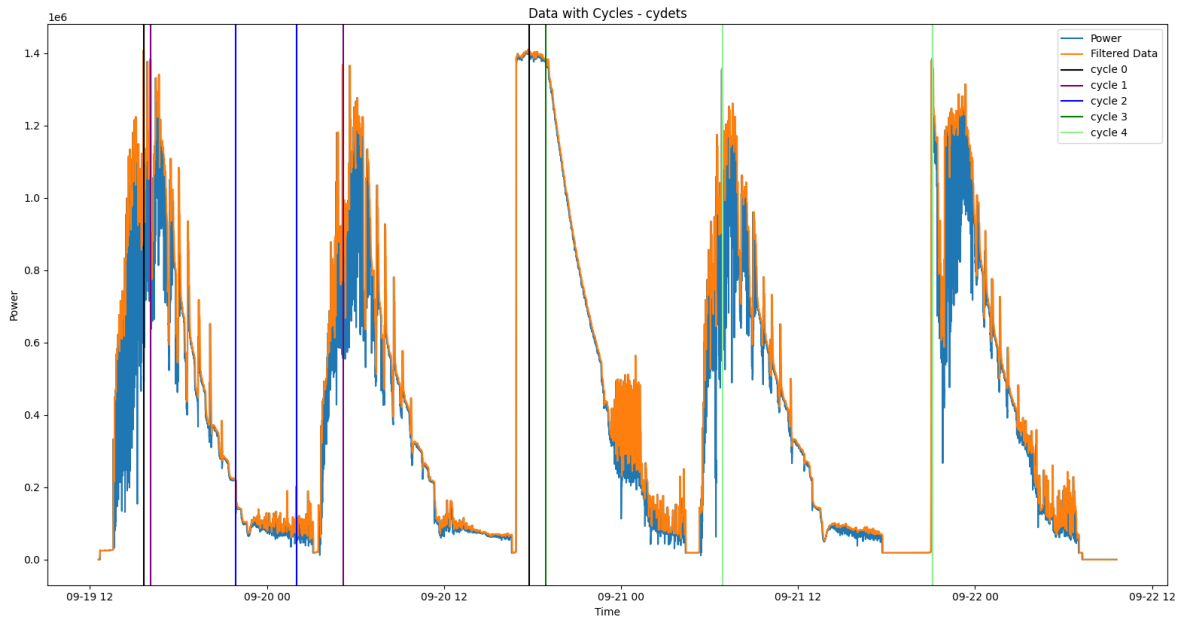


Figure 39 : cydets detection, cycles over 4 hours

The filtering process generated the five cycles shown in Figure 39. Although the results might not be entirely precise, some interesting observations can be made. First of all, five cycles are detected, which is the anticipated number of results. The repartition of cycles is the most interesting point. Cycles 1, 4, and even 0 correspond to the green dots illustrated in Figure 33, while cycle 2 corresponds to certain red and orange dots. However, not all conditions are met to precisely achieve the desired cycles detection, as no result accurately captures the desired output. Cycle 0 significantly surpasses the duration of any other detected cycle, while cycle 2 is notably shorter. Such variation in the results make the outcome in a real environment much less predictable. Unfortunately, there are no alternative methods to fine-tune the algorithm to achieve closer results.

6.4 Isolation Forest

6.4.1 Concept

In order to find an algorithm that could perform the desired cycle detection, other path have been explored. This chapter focuses on Isolation Forest, an unsupervised machine learning algorithm aiming to detect outlier and anomaly.

A significant portion of existing models employed for anomaly detection starts with the construction of a profile of normal patterns within the data. Then, any data point that deviates from this profile is labelled as an outlier. However, such algorithms are optimized to identify normal patterns, and they often exhibit inefficiency when dealing with large datasets due to their high computational complexity [49].

Isolation Forest, on the other hand, operates on a different principle. Anomalies tend to exhibit a lower occurrence and distinct characteristics compared to normal data points, rendering them easier to isolate. This algorithm takes advantage of this distinction, making it particularly effective for detecting anomalies in large datasets. Concretely, this algorithm is based on a collection of binary trees, called Isolation Trees or iTrees. These trees are used to partition the data and identify anomalies. For each tree, the algorithm randomly selects a feature and then selects a random value within the range of that feature. This value is used to split the data into two parts: one containing datapoints that are greater than the selected value, and the other containing data points that are less than or equal to the selected value. The algorithm repeats the random partitioning process on the subset of data that falls within the selected range. This process is repeated recursively until isolated points are identified or until a predefined stopping criterion is met. Each internal node of the tree represents a feature and a splitting value, while the terminal nodes represent isolated points [50].

The algorithm assigns an anomaly score to each data point based on the height of the isolation tree in which it is isolated. The idea is that genuine anomalies can be isolated more quickly than regular data points, so their average path length within the tree is shorter. Anomalies are identified by comparing the anomaly scores to a predefined threshold, set by the user. Data points with scores above this threshold are considered anomalies [51].

Figure 40 represents one of the iTrees, where each bubble corresponds to a randomly chosen feature and threshold feature. As mentioned earlier, shorter path is considered as outliers, as represented by the red bubble.

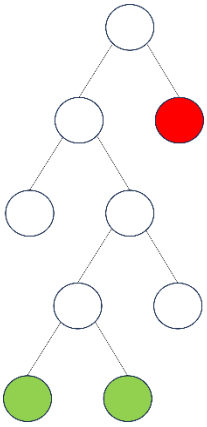


Figure 40 : Isolation Tree

While each individual tree might be constructed based on random feature selections and splitting values, the randomness introduced during the construction process of a single tree can lead to some variability in the results. By using an ensemble of trees, the algorithm reduces the impact of this randomness, resulting in more robust and consistent anomaly detection.

6.4.2 Results & Conclusion

Once more the sklearn library provides a class that implements the Isolation Forest algorithm [52]. The code using this library can be found in in 'testDataAnalysis → testClustering.py → testIsolationForest()'.

An interesting parameter of the implementation of the Isolation Forest algorithm is the contamination parameter, when creating an instance of the Isolation Forest model. This value represents the proportion of expected outliers and plays a crucial role in establishing the threshold for scoring anomalies. The 'auto' option is considered as a great starting point, particularly when the proportion of outliers in the dataset is unknown. This setting assigns the 'offset_' parameter a value of -0.5. This argument takes part in the decision function, defined as:

$$decision_{function} = score_{samples} - offset\_$$

Isolation Forest algorithm follows an approximate Gaussian distribution centred at 0. The algorithm assigns a positive score to inliers and a negative score to outliers. By setting the threshold at -0.5, it is creating a balance point where data points with a score less than -0.5 are classified as anomalies. In cases where the contamination parameter differs from 'auto', the offset is determined to achieve the expected count of outliers in the results. This enables the algorithm to adapt and accurately classify anomalies based on the provided parameter.

As it seemed to be a good place to start, the first test was made with the contamination parameter set to 'auto'.

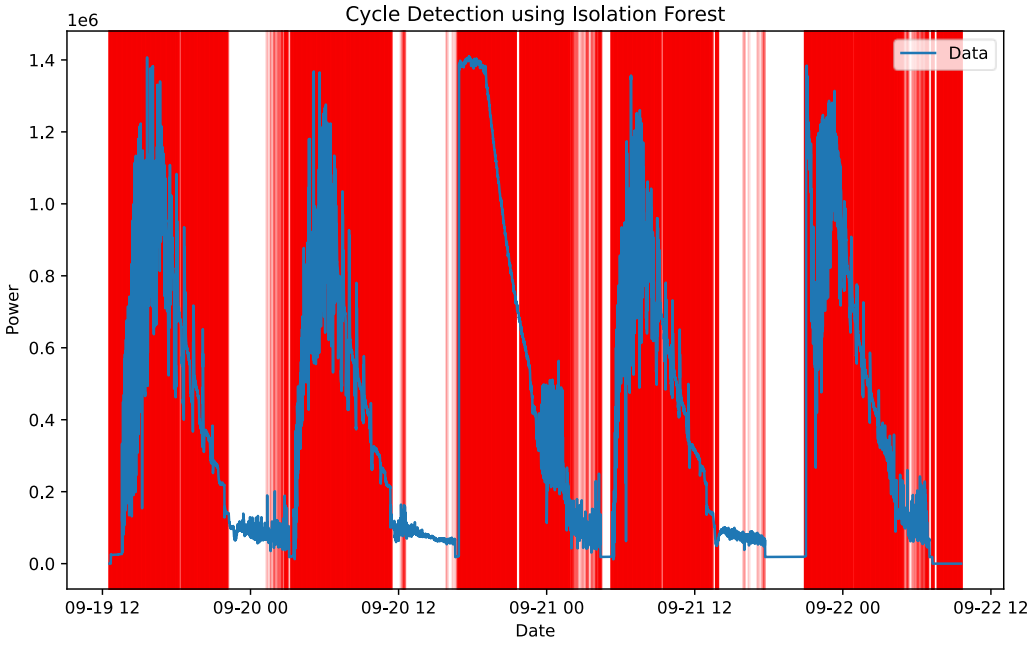
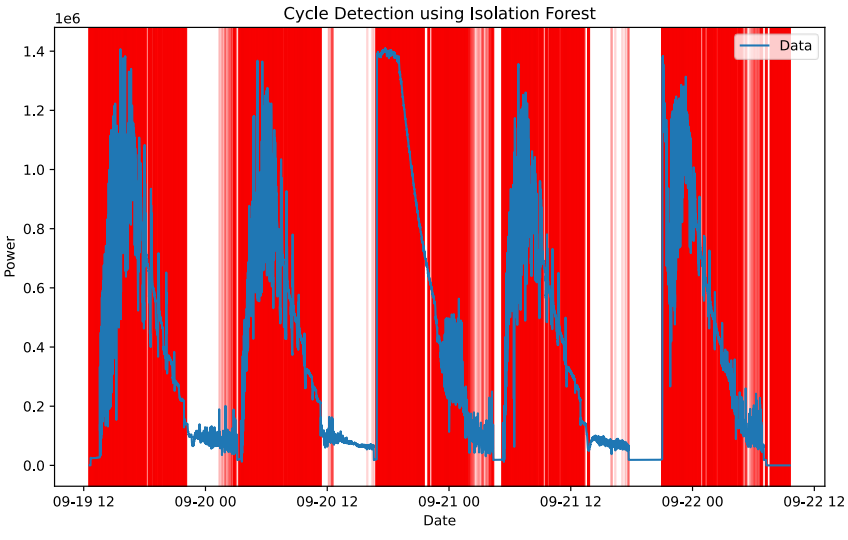


Figure 41 : Isolation Forest, contamination='auto', random_state=666

In this configuration, the algorithm detects 14035 outliers, out of 24699 points. Surprisingly, outliers, represented by a red line, gather around cycles starting by a red dot and finishing by an orange one in Figure 33. This result encouraged deeper involvement in this algorithm.

The following table presents a comparison of the results when changing the contamination value.

0.5 Number of outliers: 12346	

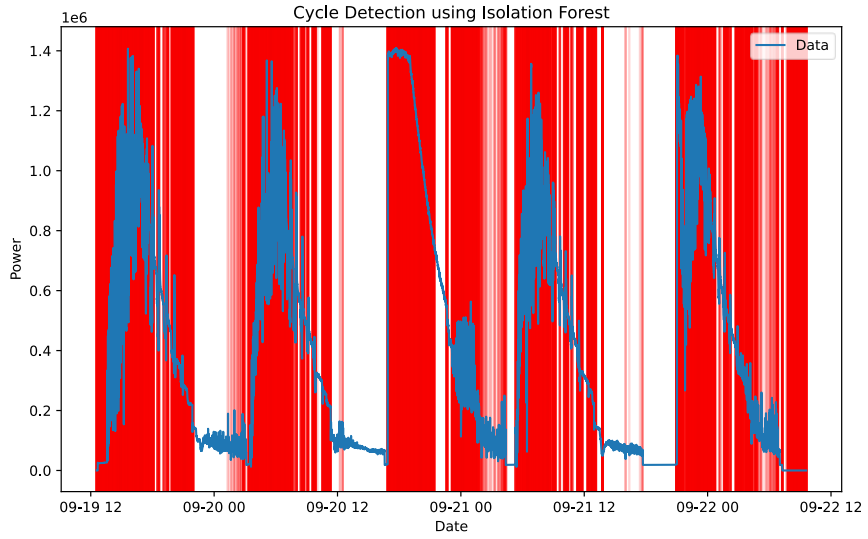
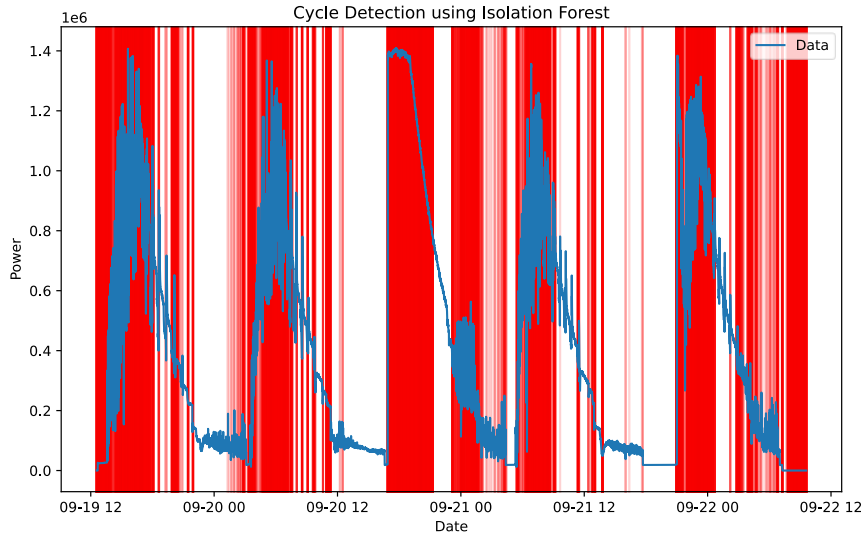
0.4 Number of outliers: 9877	
0.3 Number of outliers: 7409	

Chart 1: Isolation Forest with different contamination parameters, random_state=666

As the contamination parameter decreases, fewer outliers are identified, potentially resulting in gaps within the detected cycles. This is why, for the ongoing development of this algorithm, the contamination parameter is consistently set to 'auto'.

At this point, the aim is to organize the results in a manner that displays the desired cycles. The first amelioration has been to aggregate outliers following each other into a single window, which will eventually represent the cycles. If x_i is considered as an anomaly, and so is x_{i+1}, they are put in an array. This array is then striped of any value between the first and the last one. The aim is to keep only the beginning and end information of each group of outliers. From now on, these arrays of outliers will be called cycles. Additionally, if the time gap between two cycles is less than 30 minutes, they are merged together. These adjustments effectively reduce the number of detected cycles to 9.

The next improvement has been made by removing any cycle lasting less than four hours, for the same reasons as in 6.1.2. This curation results in 5 cycles that are positioned around the expected timeframes.

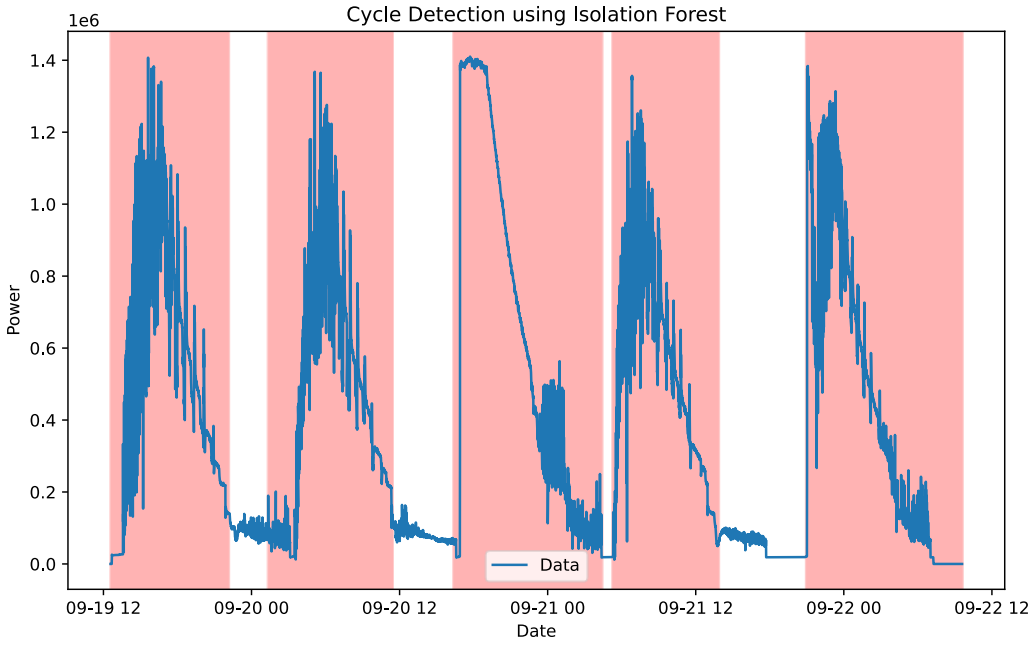


Figure 42 : Customed Isolation Forrest algorithm, contamination='auto', random_state=666

At this point it is also important to test the enhanced algorithm on the second dataset containing power consumption values.

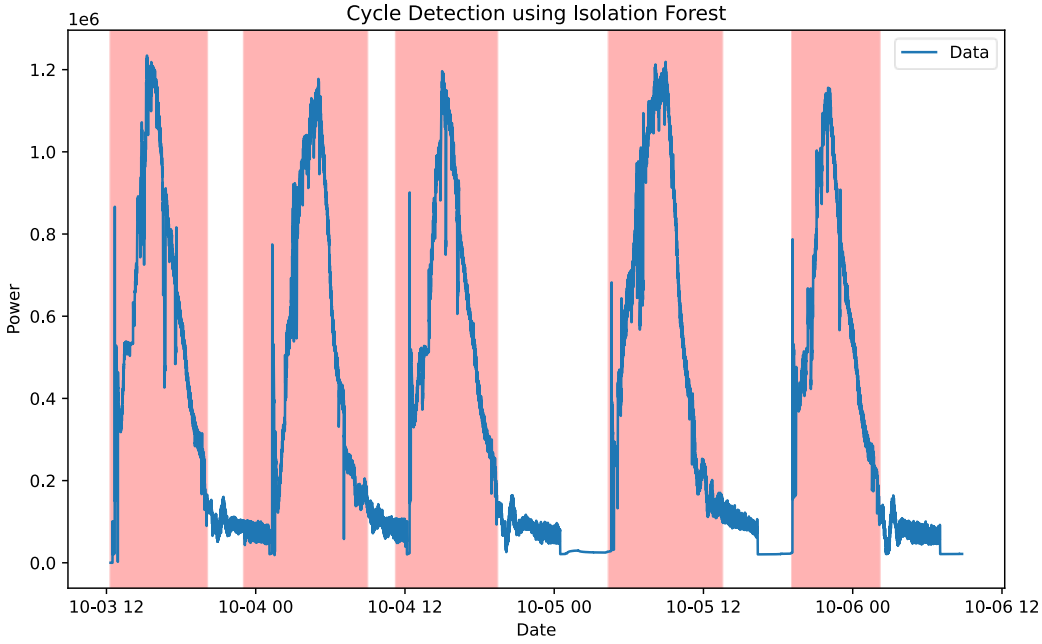


Figure 43 : Customed Isolation Forrest algorithm on October's dataset, contamination='auto', random_state=666

Once again, the results are satisfying. The end of each cycle is earlier than in Figure 34, but this variance is tolerable.

Even if the results are very promising, this algorithm will unfortunately not fit the real time operational context of this project. The choices made earlier to merge cycles that are less than 30 minutes apart and to delete ones that are longer than four hours, go against the real time application. Moreover, these choices are based on a subjective interpretation of only two datasets. The fitness of this algorithm on other datasets cannot be guaranteed.

6.5 Slope detection

6.5.1 Concept

This algorithm's development was primarily guided by the projected outcome showcased in Figure 29. Given the thesis' core objective of cycle detection, it is particularly interested in recognizing phases when the signal is ascending, descending, or stable. Essentially, this code assigns labels to individual data points.

The underlying concept hinges on differentiation. This involves subtracting each data point from its preceding one, resulting in the slope for each point. Subsequently, specific threshold values are established to distinguish and categorize these differentiation outcomes. This stage establishes a rudimentary form of labelling.

6.5.2 Results & Conclusion

The complete code for this algorithm can be found in 'testDataAnalysis → testClustering.py → testLab()'

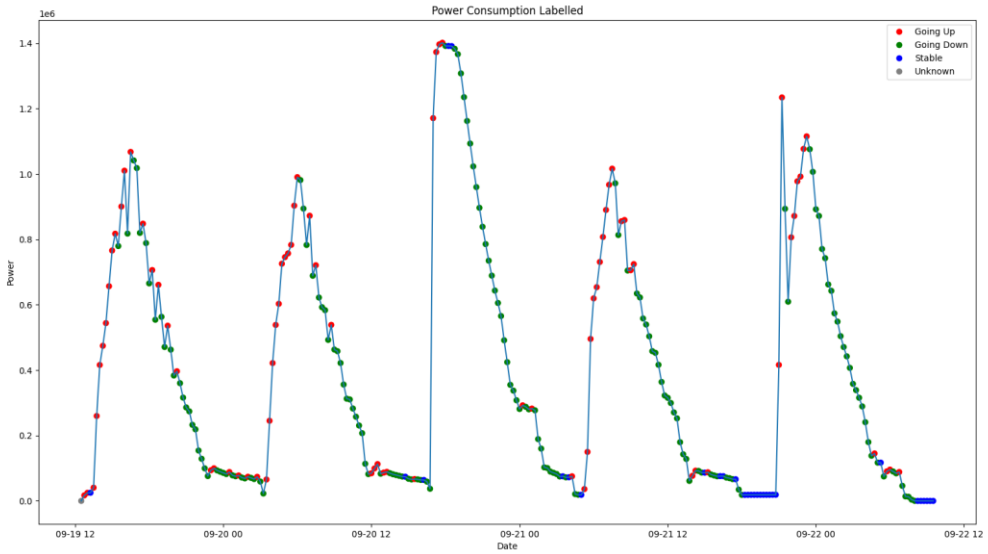


Figure 44 : Custom labelling V1 on filtered data

The obtained result is not particularly effective in detecting phases. For instance, it is evident that there are instances of red points within decreasing phases, which indicates an increase. To cancel this effect, an improvement has been tried. The concept involved considering the trend exhibited by the three most recent points. For each data point, its label is computed and then compared against the average label of the preceding three points. If they align, the new point retains its calculated label. If they diverge, the new point assumes the same label as the three preceding points. To prevent uniform

labelling across all data points, each measurement is endowed with two labels: the initially computed one and the one assigned after the comparison. It is important to note that the comparison is predicated on the initially calculated label.

This improvement producing only little changes in the results, it has been determined that this algorithm is not the most optimal for the desired outcome.

6.6 Idle detection

Even though the previous algorithm is not perfect, it sparked the idea to detect when the oven is in an idle phase, allowing to segment the signal in distinct cycles.

Given the potential noise present in the signal, even after filtering, a strategic choice was made to implement a Schmitt Trigger mechanism to detect periods when the oven is deemed idle. Frequently utilized in electronics, the Schmitt Trigger compares the signal's amplitude to two predetermined thresholds: one high and one low. If the signal falls below the low threshold, it signifies that the system is in an idle phase. To transition out of this state, the signal must surpass the high threshold. This mechanism accommodates signal fluctuations within these two thresholds, preventing unintended phase transitions. The values used as low and high threshold are once more based on the observation of the given datasets.

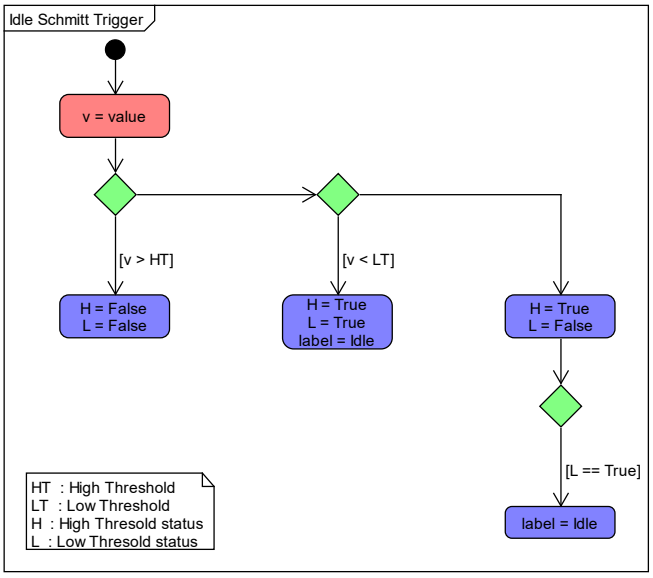


Figure 45 : Schmitt Trigger for Idle state diagram

Ultimately, another phase has been added. If the idle phase is too long, the signal switches to an ‘off’ state.

6.7 Find the maximum

6.7.1 Concept

The final algorithm discussed is another one that has been especially developed for this project. Building upon the approach outlined in Section 6.5, this algorithm aims to segregate the signal into ascending, descending, and idle phases. The underlying concept revolves around identifying the peak

point of the signal. Preceding the maximum point, each data point is labelled as ascending, while those following the maximum point are labelled as descending.

Every time a new cycle begins, the variable used to save the current highest point encountered, is set to the value of the high threshold discussed in Section 6.6. Whenever there is a new point, it is compared to this variable. If it is higher, the variable takes the value of the new point. To overcome the challenge of accurate labelling raised by the 'Slope Detection' algorithm due to inherent noise in the data, this algorithm adopts a unique solution: whenever the variable holding the maximum value is updated, each point from the beginning of the cycle, up to the new maximum value, undergoes re-labelling as ascending points.

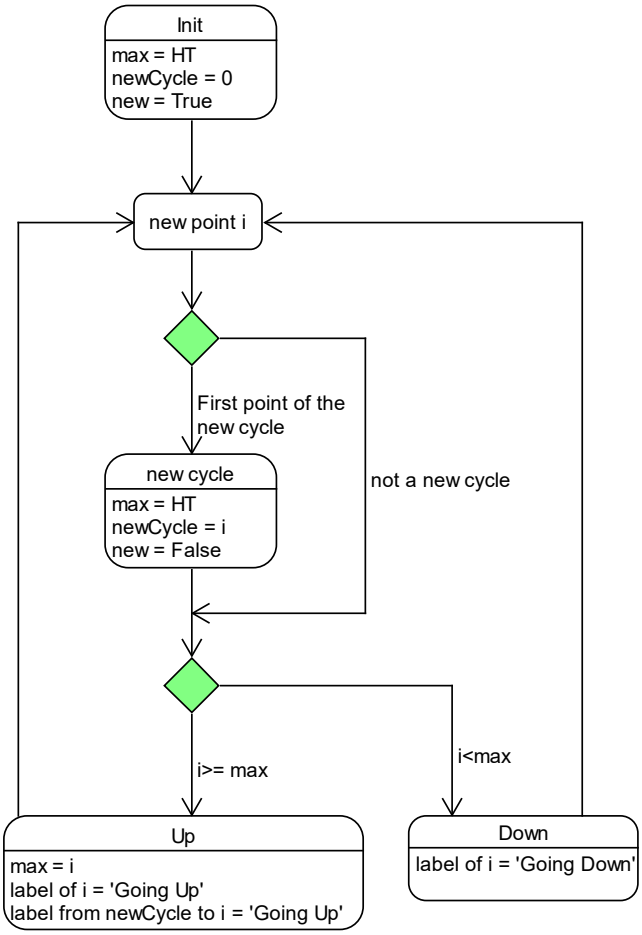


Figure 46: Find the maximum state diagram

The idle detection is not portrayed in Figure 46, to focus only on the new algorithm. The idle detection occurs right after the arriving of the new point. As established earlier, a new cycle is set to begin when the signal leaves the idle state. The index of every new cycle is stored in an array.

6.7.2 Results & Conclusion

The code for this algorithm can be found in 'testDataAnalysis → testClustering.py → testIdea()'.

As per usual, the first step of the implementation is to test the algorithm on the CSV file containing the dataset. For this case, the diagram in Figure 46 is implemented with a loop that traverses the entire structure containing the data.

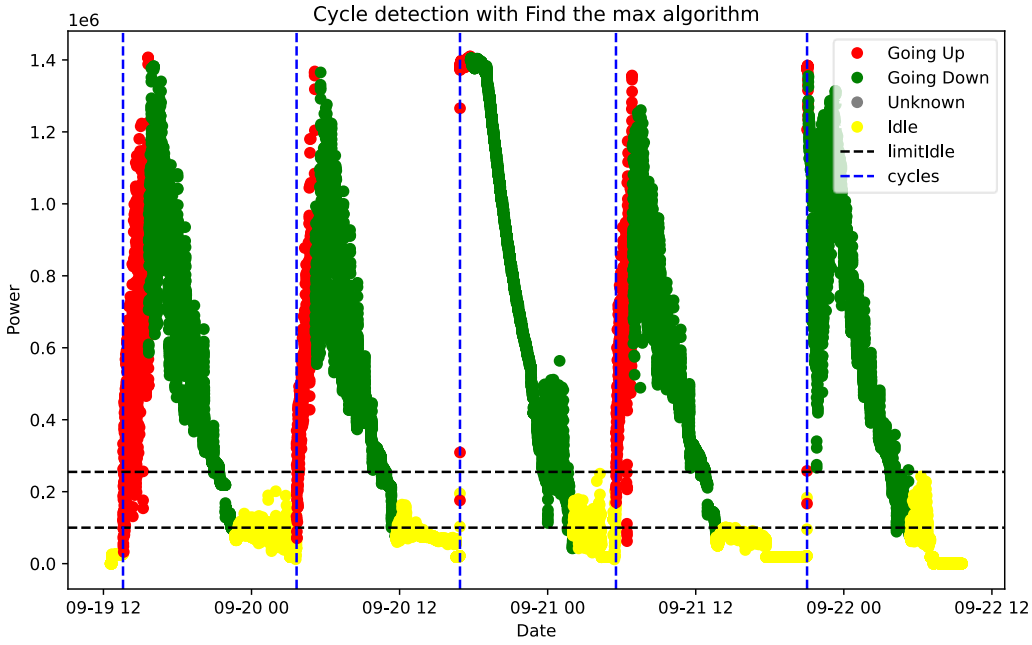


Figure 47: Find the maximum algorithm on September's power dataset, filter data with the rolling max algo

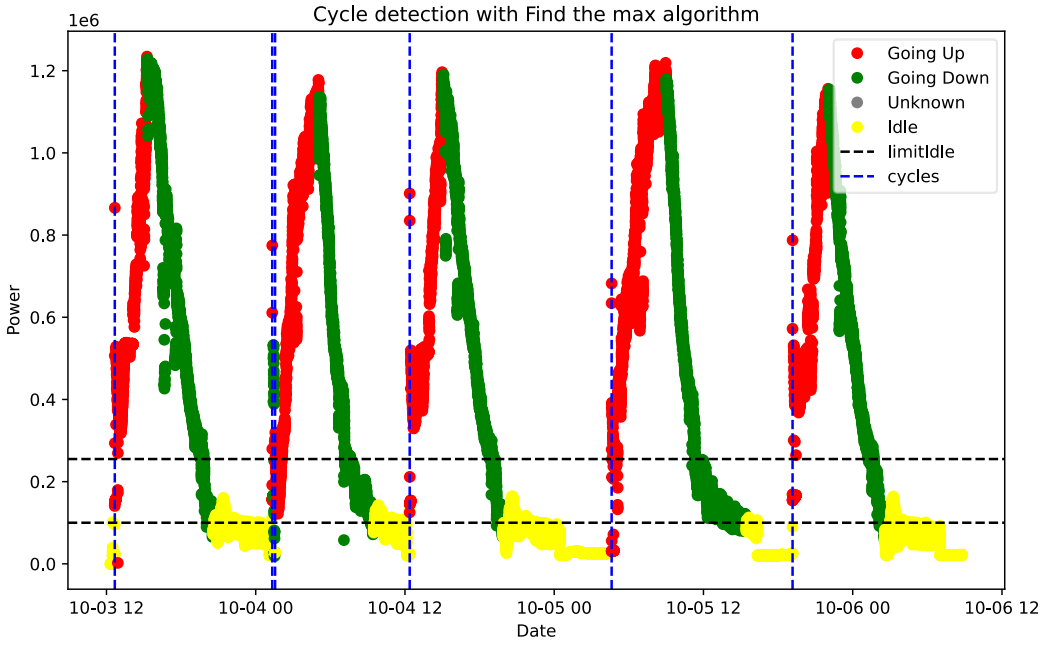


Figure 48: Find the maximum algorithm on October's power dataset, filter data with the rolling max algo

The results for both datasets are in line with the expected output presented in Figure 33 and Figure 34. The only difference comes from October's dataset, where two cycles are detected around the beginning of the fourth of October. This unexpected cycle disappears when the dataset is resampled, each point representing the maximum value of every measurement taken every minute. This change in the sampling frequency is considered as acceptable, as it is way under the critical threshold of 15 minutes relevant for the calculation of the electricity's price.

Since the deployment using the CSV files produced very good results, the algorithm has been implemented in the real-time environment of this project, through the dashboard. As data points arrive one after the other, the aim is to only compute the label for the incoming datapoint. Instead of going through the entire dataset at once, every time a new point arrives, its label is computed. The implementation for the dashboard can be found in 'elfo-clean-base → src → dash → dashboard → forecast_helper → label.py →cycleDetection()', in the ComputOven class.

This time results will be displayed on the synthetically generated datasets. As there is no way in this report to present a dynamic result, where one could see updates on labels in real-time, GIFs can be found in the presentation section of the git.

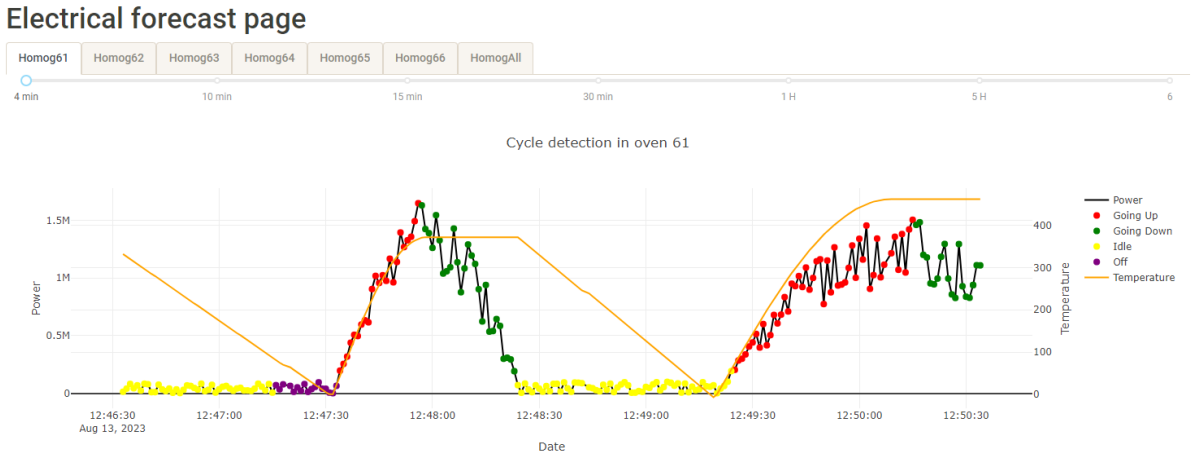


Figure 49: Find the maximum algorithm on synthetically generated dataset

Of course, this algorithm could also be applied on temperature dataset. However, displaying labels on both signals is not optimal, as too much information is present on the screen.

7 Results

This section aims to very briefly summarize every choice that have been made to produce the final result. For further details, please refer to the appropriate section.

7.1 OPC-UA server

A local OPC-UA server has been implemented, operating on 'opc.tcp://localhost:4840'. Concerning the data sent to this server, two options are available: users can either provide a CSV file containing the required data or opt for the generation of synthetic datasets. The server configuration encompasses six datasets denoting power consumption and an additional six for temperature, simulating data incoming from the six ovens. Notably, the historian mode has been enabled on each node, simulating data acquisition from the six ovens. Data points are transmitted to the server every second.

The implementation of this server is found within the 'opcUAServ' folder, which also includes the code for producing synthetic power and temperature data.

7.2 Dashboard & cycle detection algorithm

The dashboard serves as a comprehensive platform encompassing various functionalities. Its primary purpose is to provide real-time visualization of measurements for each oven. The dashboard's updates are orchestrated by two callback functions. The initial function, 'getData()', is triggered upon the expiration of a one-second timer. This function facilitates the polling of the server for new datapoints, followed by the application of the cycle detection algorithm. Subsequently, the second function, 'displayData()' is activated, facilitating the display of the freshly acquired information on the dashboard. Notably, the incorporation of the cycle detection algorithm has enabled the presentation of statistics concerning the ongoing and preceding cycles.

Figure 50: Statistics presented on the dashboard

Each oven is accessible through its own tab. A seventh tab has been added, summarizing the power consumption of every oven. To provide greater control over displayed timeframes, a sliding bar feature has been integrated into the dashboard. Any datapoint falling outside this specified timeframe is systematically stored in a CSV file.

After evaluating multiple algorithms, the 'Find the maximum' approach has emerged as the most promising solution, warranting its integration into the dashboard.

This decision finds its foundation in the algorithm's significantly favourable outcomes, particularly when juxtaposed with other tested methods. Notably, it closely aligns with the anticipated results initially presented in Section 6. Moreover, its successful implementation aptly accommodates the real-time operational demands set forth by this project.

8 Conclusion

8.1 Synthesis

The primary objective of this project was to devise an algorithm capable of detecting homogenization oven machine cycles at Constellium. This effort aimed to mitigate the influence of power consumption peaks on electricity costs. To realize this goal, the process began with the collection of oven measurements via an OPC-UA server. Subsequently, this data was channelled to a program facilitating machine cycle detection and the real-time display of consumption and temperature data.

Due to infrastructure constraints regarding data collection and storage, a local OPC-UA server was employed. In terms of available data, two datasets were furnished. To broaden the scope, synthetic datasets were also generated. The project's dashboard adeptly showcases information and statistics about six ovens. An additional interface offers a comprehensive overview of the homogenization sector's collective consumption.

In the quest to identify the optimal cycle detection method, a range of algorithms was subjected to testing. Ultimately, the most effective approach thus far involves segregating cycles based on their idle phases, while simultaneously furnishing insights into consumption fluctuations throughout each cycle.

This implementation effectively addresses all the mandatory objectives outlined for this Bachelor project. Regrettably, due to time constraints, the optional objectives could not be thoroughly pursued.

The main problem encountered throughout this project was the lack of data. At multiple points during the project I asked for more data, and was told it would be available soon, which eventually never happened. The temperature data were only available during the last couple of weeks of the implementation, which allowed to display a bit more information. However, the final implementation has not been tested on new real data.

8.2 Improvements and further developments

8.2.1 OPC-UA server

Currently, the operational framework involves a polling mechanism that solicits the server every second for data updates. However, it is worth noting that OPC-UA servers offer a notification service, designed to inform every subscribed client of a change in a value. At this stage, the client cannot process a large and frequent number of notifications. If it was more robust, the dashboard logic could be based on notifications, instead of polling. This new strategy could be implemented in two different ways. The first would imply displaying changes as soon as the notification is received. This solution would imply irregular changes, which might be a problem for the current implementation of the dashboard. Otherwise, the idea of displaying changes at a fix frequency is maintained. If there is no new data since the last display, the same value is kept for a couple of minutes and then set to zero. Since there is no way to know the cause of the lack of data (no new point yet, problem of the machine, problem of the server), it could be interesting to be able to read alert from the server.

8.2.2 Dashboard

Although Python is a very useful programming language due to its ease of learning, its performance is known not to be optimal. Unfortunately, the current implementation of the dashboard suffers from these performance issues, particularly when it is launched. As the historian for twelve datasets has to be read and the cycle detection performed on six of them, it implies a lot of computation. Furthermore,

the server's polling takes the priority over anything else. Therefore, the page unfortunately lags. There are multiple ways to improve the performance of the dashboard. First of all, the use of a thread for each oven could allow a parallelised computation of the machine cycle algorithm. Secondly, pandas provide different techniques to speed up data processing, such as 'padaral.lel' that allows a parallelisation of operations. Finally, Pola.rs, which is a Rust library that provides a Python API, centred around DataFrames.

8.2.3 Storage

As mentioned in the dedicated chapter, the current solution is not optimal. The question of how to store data still needs an answer. Is it really a good idea to store every datapoint, with their label? Or would a summary of the cycle, with information on its length, power peak, etc, would be enough ? As this subject is not part of the project's objectives, further considerations are left open to anyone interested.

8.2.4 Algorithm

Finally, the algorithm itself possesses latent opportunities for enhancement. A deeper understanding of machine learning principles might unveil more efficient avenues for executing cycle detection. The algorithm's potential could be further unlocked through the incorporation of additional inputs. The inclusion of an expanded dataset featuring a broader spectrum of measurements or comprehensive load-related data has the potential to considerably elevate the precision of detection. This could lead to substantial enhancements in the algorithm's overall efficiency.

9 Bibliography

[1] 'Le Valais, Switzerland - Locations | Constellium | Constellium'. https://www.constellium.com/locations/le-valais (accessed Jun. 14, 2023).
[2] HES-SO Valais-Wallis, 'Informations given by the HES-SO'.
[3] 'Gold Prices - 100 Year Historical Chart'. https://www.macrotrends.net/1333/historical-gold-prices-100-year-chart (accessed Jul. 14, 2023).
[4] 'Récapitulatif du climat pour Sion'. https://www.prevision-meteo.ch/climat/annuel/sion (accessed Jul. 14, 2023).
[5] CDC's Division of Diabetes Translation, 'Long-term Trends in Diabetes', p. 6.
[6] T. M. Mitchell, *Machine Learning*. in McGraw-Hill series in computer science. New York: McGraw-Hill, 1997.
[7] C. M. Bishop, *Pattern recognition and machine learning*. in Information science and statistics. New York: Springer, 2006.
[8] Z. Ghahramani, 'Unsupervised Learning', in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 72–112. doi: 10.1007/978-3-540-28650-9_5.
[9] Q. Bi, K. E. Goodman, J. Kaminsky, and J. Lessler, 'What is Machine Learning? A Primer for the Epidemiologist', *Am. J. Epidemiol.*, vol. 188, no. 12, pp. 2222–2239, Dec. 2019, doi: 10.1093/aje/kwz189.
[10] 'What Is an Autoregressive Model?', *365 Data Science*, Mar. 06, 2020. https://365datascience.com/tutorials/time-series-analysis-tutorials/autoregressive-model/ (accessed Jun. 30, 2023).
[11] T. Walters, 'Time Series Analysis in R Part 2: Time Series Transformations | R-bloggers', Sep. 25, 2017. https://www.r-bloggers.com/2017/09/time-series-analysis-in-r-part-2-time-series-transformations/ (accessed Aug. 15, 2023).
[12] S. Prabhakaran, 'ARIMA Model - Complete Guide to Time Series Forecasting in Python | ML+', *Machine Learning Plus*, Aug. 22, 2021. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/ (accessed Jun. 02, 2023).
[13] 'statsmodels: Statistical computations and models for Python'. Accessed: Jul. 04, 2023. [OS Independent]. Available: https://www.statsmodels.org/
[14] S. Khan and H. Alghulaiakh, 'ARIMA Model for Accurate Time Series Stocks Forecasting', *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 7, 2020, doi: 10.14569/IJACSA.2020.0110765.
[15] F. Mahia, A. R. Dey, M. A. Masud, and M. S. Mahmud, 'Forecasting Electricity Consumption using ARIMA Model', in *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*, Dec. 2019, pp. 1–6. doi: 10.1109/STI47673.2019.9068076.
[16] N. Rücker, L. Pflüger, and A. Maier, 'Hardware Failure Prediction on Imbalanced Times Series Data', *J. Digit. Imaging*, vol. 34, no. 1, pp. 182–189, Feb. 2021, doi: 10.1007/s10278-020-00411-4.
[17] TheBeard, 'The Gaussian Distribution', *The Beard Sage*, Jan. 28, 2020. http://thebeardsage.com/the-gaussian-distribution/ (accessed Jul. 04, 2023).
[18] E. Schulz, M. Speekenbrink, and A. Krause, 'A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions', *J. Math. Psychol.*, vol. 85, pp. 1–16, Aug. 2018, doi: 10.1016/j.jmp.2018.03.001.
[19] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, 'When Gaussian Process Meets Big Data: A Review of Scalable GPs', *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4405–4423, Nov. 2020, doi: 10.1109/TNNLS.2019.2957109.
[20] A. Tolver, 'An introduction to Markov chains'.
[21] L. Kegel, M. Hahmann, and W. Lehner, 'Feature-based comparison and generation of time series', in *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, Bozen-Bolzano Italy: ACM, Jul. 2018, pp. 1–12. doi: 10.1145/3221269.3221293.

[22] W. Tushar, S. Huang, C. Yuen, J. A. Zhang, and D. B. Smith, 'Synthetic generation of solar states for smart grid: A multiple segment Markov chain approach', in *IEEE PES Innovative Smart Grid Technologies, Europe*, Oct. 2014, pp. 1–6. doi: 10.1109/ISGTEurope.2014.7028832.
[23] K. Brokish and J. Kirtley, 'Pitfalls of modeling wind power using Markov chains', in *2009 IEEE/PES Power Systems Conference and Exposition*, Mar. 2009, pp. 1–6. doi: 10.1109/PSCE.2009.4840265.
[24] 'HTML SVG'. https://www.w3schools.com/html/html5_svg.asp (accessed Jul. 27, 2023).
[25] 'Grafana | Query, visualize, alerting observability platform', *Grafana Labs*. https://grafana.com/grafana/ (accessed Aug. 07, 2023).
[26] 'OPC UA Datasource plugin for Grafana'. Grafana Labs, Jul. 06, 2023. Accessed: Aug. 07, 2023. [Online]. Available: https://github.com/grafana/opcua-datasource
[27] 'Understanding the OPC Unified Architecture (OPC UA) Protocol - Technical Articles'. https://control.com/technical-articles/understanding-the-opc-ua-protocol/ (accessed Jun. 23, 2023).
[28] 'Unified Architecture', *OPC Foundation*. https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed Jun. 23, 2023).
[29] 'Prosys OPC UA Simulation Server - Prosys OPC'. https://www.prosysopc.com/products/opc-ua-simulation-server/ (accessed Jun. 16, 2023).
[30] 'nonlinear.pdf'. Accessed: Aug. 02, 2023. [Online]. Available: https://faculty.washington.edu/ezivot/econ584/notes/nonlinear.pdf
[31] 'scipy.signal.butter — SciPy v1.11.1 Manual'. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html (accessed Jul. 31, 2023).
[32] 'scipy.signal.filtfilt — SciPy v1.11.1 Manual'. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html#scipy.signal.filtfilt (accessed Jul. 31, 2023).
[33] Yacola, 'Answer to "How to get high and low envelope of a signal"', *Stack Overflow*, Feb. 25, 2020. https://stackoverflow.com/a/60402647 (accessed Jul. 31, 2023).
[34] 'UaExpert "UA Reference Client" - Unified Automation'. https://www.unified-automation.com/products/development-tools/uaexpert.html (accessed Aug. 03, 2023).
[35] 'Part 1. Layout | Dash for Python Documentation | Plotly'. https://dash.plotly.com/layout (accessed Jun. 16, 2023).
[36] 'Part 2. Basic Callbacks | Dash for Python Documentation | Plotly'. https://dash.plotly.com/basic-callbacks (accessed Jun. 16, 2023).
[37] W. McKinney, 'pandas: a Foundational Python Library for Data Analysis and Statistics', vol. 14, no. 9, pp. 1–9, 2011.
[38] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.
[39] A. B. Downey, 'A novel changepoint detection algorithm'. arXiv, Dec. 05, 2008. Accessed: Aug. 08, 2023. [Online]. Available: http://arxiv.org/abs/0812.1237
[40] P. Fearnhead and Z. Liu, 'On-Line Inference for Multiple Changepoint Problems', *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 69, no. 4, pp. 589–605, Sep. 2007, doi: 10.1111/j.1467-9868.2007.00601.x.
[41] A. J. Scott and M. Knott, 'A Cluster Analysis Method for Grouping Means in the Analysis of Variance', *Biometrics*, vol. 30, no. 3, p. 507, Sep. 1974, doi: 10.2307/2529204.
[42] R. Killick, 'Analysis of changepoint models.', 2011.
[43] S. R. Eddy, 'What is Bayesian statistics?', *Nat. Biotechnol.*, vol. 22, no. 9, Art. no. 9, Sep. 2004, doi: 10.1038/nbt0904-1177.
[44] 'ruptures: Change point detection for signals in Python.' Accessed: Aug. 08, 2023. [OS Independent]. Available: https://github.com/deepcharles/ruptures/
[45] R. Xu and D. Wunsch, 'Survey of clustering algorithms', *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005, doi: 10.1109/TNN.2005.845141.

[46] S. Na, L. Xumin, and G. Yong, 'Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm', in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, Apr. 2010, pp. 63–67. doi: 10.1109/IITSI.2010.74.
[47] '2.3. Clustering', *scikit-learn*. https://scikit-learn/stable/modules/clustering.html (accessed Aug. 09, 2023).
[48] 'oemof/cydets'. oemof community, May 15, 2023. Accessed: Aug. 10, 2023. [Online]. Available: https://github.com/oemof/cydets
[49] F. T. Liu, K. M. Ting, and Z.-H. Zhou, 'Isolation Forest', in *2008 Eighth IEEE International Conference on Data Mining*, Dec. 2008, pp. 413–422. doi: 10.1109/ICDM.2008.17.
[50] S. Hariri, M. C. Kind, and R. J. Brunner, 'Extended Isolation Forest', *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1479–1489, Apr. 2021, doi: 10.1109/TKDE.2019.2947676.
[51] C. Jung, Y.-G. Lee, J.-W. Lee, and S. Kim, 'Performance Evaluation of the Multiple Quantile Regression Model for Estimating Spatial Soil Moisture after Filtering Soil Moisture Outliers', *Remote Sens.*, vol. 12, p. 1678, May 2020, doi: 10.3390/rs12101678.
[52] 'sklearn.ensemble.IsolationForest', *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.ensemble.IsolationForest.html (accessed Aug. 12, 2023).
[53] 'Transforming our world: the 2030 Agenda for Sustainable Development | Department of Economic and Social Affairs'. https://sdgs.un.org/2030agenda (accessed Jun. 07, 2023).
[54] 'Goal 7: Ensure access to affordable, reliable, sustainable and modern energy for all'. https://www.eda.admin.ch/agenda2030/en/home/agenda-2030/die-17-ziele-fuer-eine-nachhaltige-entwicklung/ziel-7-zugang-zu-bezahlbarer-verlaesslicher-nachhaltiger-und.html (accessed Jun. 07, 2023).
[55] 'Goal 9: Industry, innovation and infrastructure | Sustainable Development Goals | United Nations Development Programme', *UNDP*. https://www.undp.org/sustainable-development-goals/industry-innovation-and-infrastructure (accessed Jun. 07, 2023).

10 Figure Table

FIGURE 1: THE PRICE OF GOLD PER OUNCE, BETWEEN 1915 AND 2023 (INFLATION-ADJUSTED)[3] 6
FIGURE 2: AVERAGE MONTHLY TEMPERATURES – SION [4] FIGURE 3: EVOLUTION OF DIABETES DIAGNOSIS [5] 7
FIGURE 4: TIME SERIES DIFFERENTIATED TWICE TO BECOME STATIONARY [11] 9
FIGURE 5: GAUSSIAN DISTRIBUTION [17] 10
FIGURE 6 : DATA ACQUISITION ARCHITECTURE 13
FIGURE 7 : PROSYS OPC-UA SIMULATION SERVER INTERFACE 14
FIGURE 8 : |P TOTAL| AVERAGE, 19/09/22 TO 22/09/22 FIGURE 9 : |P TOTAL| AVERAGE, 03/10/22 TO 06/10/22 15
FIGURE 10 : TEMPERATURE DATA, 19/09/22 TO 22/09/22 FIGURE 11 : TEMPERATURE DATA, 03/10/22 TO 06/10/22 15
FIGURE 12 : P-VALUE FROM THE BDS TEST FOR EACH DATASET 16
FIGURE 13 : SEASONALITY DECOMPOSITION FOR EACH DATASET; TOP LEFT: POWER IN SEPTEMBER, TOP RIGHT: POWER IN OCTOBER, BOTTOM LEFT: TEMPERATURE IN SEPTEMBER, BOTTOM RIGHT: TEMPERATURE IN OCTOBER 17
FIGURE 14 : |P TOTAL| AVERAGE, 19/09/22 TO 22/09/22, LOW-PASS BUTTERWORTH FILTER OF DIFFERENT ORDERS, ZOOMED .. 18
FIGURE 15 : |P TOTAL| AVERAGE, 19/09/22 TO 22/09/22, LOW-PASS BUTTERWORTH FILTER OF DIFFERENT ORDERS, USING FILTFILT, ZOOMED 19
FIGURE 16 : |P TOTAL| AVERAGE, 19/09/22 TO 22/09/22, HIGH ENVELOPE 20
FIGURE 17 : |P TOTAL| AVERAGE, 19/09/22 TO 22/09/22, HIGH ENVELOPE WITH ROLLING WINDOW 20
FIGURE 18 : RESULTS OF HIGH ENVELOPE WITH ROLLING WINDOW ON BOTH DATASETS 21
FIGURE 19 : SYNTHETIC POWER DATA 22
FIGURE 20 : SYNTHETIC POWER AND TEMPERATURE DATA 23
FIGURE 21: GLOBAL VIEW OF THE PACKAGES FOR THE DATA'S ACQUISITION 24
FIGURE 22 : CLASS DIAGRAM FOR DATASET AND OPCCLIENT 24
FIGURE 23 : READING OF HISTORIAN VALUES – SEQUENCE DIAGRAM 25
FIGURE 24 : GETTING DATA FROM THE SERVER – SEQUENCE DIAGRAM 26
FIGURE 25: GLOBAL SOFTWARE ARCHITECTURE 27
FIGURE 26: GLOBAL VIEW OF USEFUL PACKAGES 27
FIGURE 27 : DASHBOARD RECEIVED AT THE BEGINNING OF THE PROJECT 28
FIGURE 28 : RESULT OF A .CSV FILE DISPLAY 29
FIGURE 29 : PACKAGE DIAGRAM BEFORE CLASS (NOTE THAT ONLY THE RELEVANT FILES AND FOLDERS ARE REPRESENTED) 31
FIGURE 30 : COMPUTEOVEN CLASS DIAGRAM V1 31
FIGURE 31 : SEQUENCE DIAGRAM FOR THAT FOCUSES ON DICTIONARY-OBJECT CONVERSION 32
FIGURE 32 : TAB DISPLAYING THE CONSUMPTION OF EVERY OVEN 33
FIGURE 33 : POWER DATA FROM SEPTEMBER, EXPECTED OUTPUT FROM ALGORITHMS 35
FIGURE 34 :POWER DATA FROM OCTOBER, EXPECTED OUTPUT FROM ALGORITHMS 36
FIGURE 35 : BAYESIAN APPROACH IN CHANGE POINT DETECTION, CHANGING POINTS REPRESENTED BY A VERTICAL DASHED LINE 37
FIGURE 36 :BINARY SEGMENTATION MODEL OF CHANGE POINT DETECTION, N_BKPS=10, ON RAW DATA 38
FIGURE 37 : BINARY SEGMENTATION MODEL OF CHANGE POINT DETECTION, N_BKPS=10, ON FILTERED DATA (ROLLING WINDOW).. 38
FIGURE 38 : K-MEANS CLUSTERING, K=5, RANDOM_STATE = 666 39
FIGURE 39 : CYDETS DETECTION, CYCLES OVER 4 HOURS 41
FIGURE 40 : ISOLATION TREE 42
FIGURE 41 : ISOLATION FOREST, CONTAMINATION='AUTO', RANDOM_STATE=666 43
FIGURE 42 : CUSTOMED ISOLATION FORREST ALGORITHM, CONTAMINATION='AUTO', RANDOM_STATE=666 45
FIGURE 43 : CUSTOMED ISOLATION FORREST ALGORITHM ON OCTOBER'S DATASET, CONTAMINATION='AUTO', RANDOM_STATE=666 45
FIGURE 44 : CUSTOM LABELLING V1 ON FILTERED DATA 46
FIGURE 45 : SCHMITT TRIGGER FOR IDLE STATE DIAGRAM 47
FIGURE 46: FIND THE MAXIMUM STATE DIAGRAM 48
FIGURE 47: FIND THE MAXIMUM ALGORITHM ON SEPTEMBER'S POWER DATASET, FILTER DATA WITH THE ROLLING MAX ALGO 49
FIGURE 48: FIND THE MAXIMUM ALGORITHM ON OCTOBER'S POWER DATASET, FILTER DATA WITH THE ROLLING MAX ALGO 49
FIGURE 49: FIND THE MAXIMUM ALGORITHM ON SYNTHETICALLY GENERATED DATASET 50
FIGURE 50: STATISTICS PRESENTED ON THE DASHBOARD 51

11 Code table

CODE 1 : EXAMPLE OF THE CALLBACK FORM 28

12 Chart table

CHART 1: ISOLATION FOREST WITH DIFFERENT CONTAMINATION PARAMETERS, RANDOM_STATE=666 44

13 Annexe

13.1 Sustainable Development Goals

This section aims to place this thesis in the context of sustainable development. Therefore, you will find a brief explanation on what the sustainable development goals are and which ones are met in this work. To properly understand this section, the reader is advised to first read the Context Section of the report.

13.1.1 What are the Sustainable Development Goals

Established by the United Nations General Assembly in 2015, the Sustainable Development Goals are a set of 17 goals used as guidelines to provide a more sustainable future. They are part of the 2030 Agenda for Sustainable Development, which aims to protect the environment, ensuring prosperity, peace, and equity between everyone, independently of gender, religion, or race [53]. The UNDP counts on the collaboration of businesses, industries, governments, and society in general to reach those goals.

13.1.2 Goal 7: ensuring access to affordable, reliable, sustainable, and modern energy for all

There is no doubt about the fact that energy is indispensable in the current way of life and that its production has a huge impact on the planet. Most energy sources are not reusable, and their extraction causes great harm on the environment. Furthermore, energy prices are dependent on a few suppliers who have significant control over the market, leading to huge fluctuations [54].

As the aim of this thesis is to detect machine cycles to optimize energy consumption, it contributes to the seventh goal. Indeed, if the operator is able to make informed decisions on when to start a process, it can decrease the peaks in energy usage at Constellium. This energy can then be used by other consumers.

Knowing when and how much electricity is necessary is also a way to start a transition toward more sustainable sources of energy. The data can be used to determine the best time to use solar or wind energy over fossil fuels.

13.1.3 Goal 9: Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation

With the ninth goal, the UNDP encourages investments in research and innovation in order to develop more efficient and sustainable processes and infrastructure. These contributions are a great way to ensure economic growth, development, energy efficiency and constant problem-solving [55].

This project focuses on analysing time series data and *using machine learning algorithms* to optimize energy consumption. Using this kind of technology fits the innovation part of this goal.

Furthermore, the usage of the software leads to costs savings. This money can then be injected into the research and development sector to explore a more sustainable industrialization and support innovation.

Sustainable industrialization extends to data acquisition and storage practices. Potential storage solutions are explored in Section 5.3.3 of the report. A sustainable approach to data acquisition could

involve fostering communication between Constellium and the stakeholders involved in the overall project. This dialogue would aim to identify the essential measurements necessary for the project, thereby streamlining the data transmission process and minimizing unnecessary data transfer.

13.2 Deployment process

This annexe serves as a guide to launch the dashboard.

First, you need to download the code that can be found in https://gitlab.hevs.ch/SPL/bachelorthesis/2023-eco/eco_tb . For any access rights, please contact Silvan Zahno (silvan.zahno@hevs.ch). The code for the dashboard is in 'code → elfo-clean-base'. You will also need the 'opcUAServ' project.

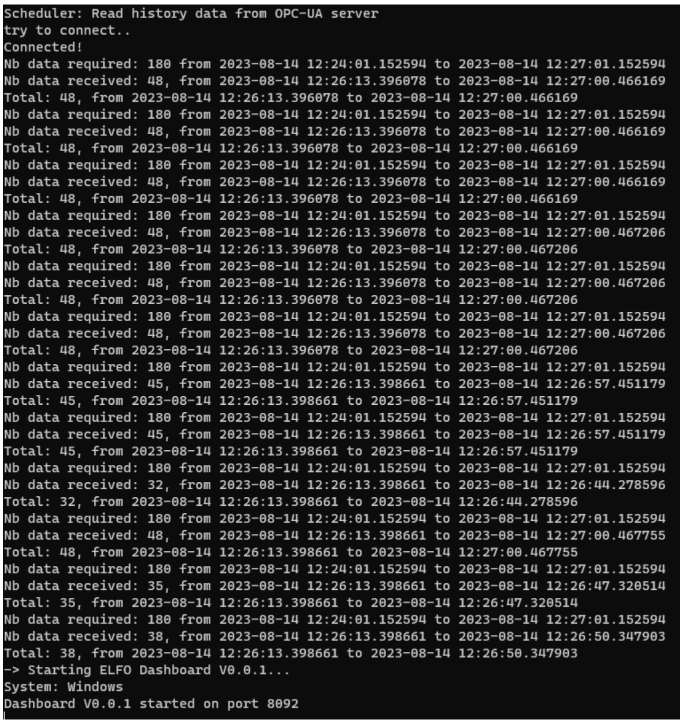
As this project is configured to run inside a conda environment, make sure to download 'Anaconda' or 'Miniconda'. Once done, you can run the 'make env' in the conda console, in the elfo-clean-base folder. This should install a new python virtual environment called 'ELFO' based on the file 'environment.yml'. To make sure that every library has been properly installed, check the top of 'electrical_forecast.py', 'label.py', 'sampling.py', 'dataset.py' and 'opcClient.py' files. If any library is underlined in red, right click on the name and install the package. At this point the environment is set up.

For the code to run smoothly, you need to first start the server. You have the option to either use real data from a CSV file or synthetically generated dataset. The code presents example for both implementations. You can then launch the server. The console displays' Server online' once it is running. Make sure that the server's URL matches in the opcUAServ project and in the opcClient.py file of the 'elfo-clean-base' project.

To launch the dashboard, open a conda command prompt, go into the 'elfo-clean-base' folder, and enter the following commands:

- conda activate ELFO
- python src/dash/index.py

The first command activates the environment, the second one launches the dashboard. The command prompt will then show something like the next image:



```
Scheduler: Read history data from OPC-UA server
try to connect..
Connected!
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.466169
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Total: 48, from 2023-08-14 12:26:13.396078 to 2023-08-14 12:27:00.467206
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 45, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:57.451179
Total: 45, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:57.451179
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 45, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:57.451179
Total: 45, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:57.451179
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 32, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:44.278596
Total: 32, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:44.278596
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 48, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:27:00.467755
Total: 48, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:27:00.467755
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 35, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:47.320514
Total: 35, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:47.320514
Nb data required: 180 from 2023-08-14 12:24:01.152594 to 2023-08-14 12:27:01.152594
Nb data received: 38, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:50.347903
Total: 38, from 2023-08-14 12:26:13.398661 to 2023-08-14 12:26:50.347903
-> Starting ELFO Dashboard V0.0.1...
System: Windows
Dashboard V0.0.1 started on port 8092
```

You can then open a browser and enter the following address: http://127.0.0.1:8092/start . This directs you toward a login page. The user name is: admin and the password: admin

You can then access the Electrical Forecast page that displays the dashboard.