Data-Driven Energy Flexibility Prediction

ISEP – Instituto Superior de Engenharia do Porto

2024 / 2025

1221063 – Pedro Manuel da Rocha Pires

Data-Driven Energy Flexibility Prediction

ISEP – Instituto Superior de Engenharia do Porto

2024 / 2025

1221063 – Pedro Manuel da Rocha Pires

DEPARTAMENTO DE ENGENHARIA
INFORMÁTICA

Degree in Informatics Engineering

June 2025

ISEP Advisor: **Tiago Carlos Caló Fonseca**

Co-Advisor: **Luís Miguel Moreira Lino Ferreira**

« The biggest risk is not taking any risks. »

Mark Zuckerberg

Acknowledgements

I want to thank my mother for everything she has done for me to this day. Her willpower and effort on ensuring both my and my sister's future is something I will be forever grateful for.

Secondly, I want to thank Tiago Carlos Caló Fonseca, my advisor in this project. During the development of this project, he was always available and made various suggestions which helped shape the final solution for the best. Furthermore, his sense of leadership really motivated me to keep on improving my work and I'm very thankful for that.

Finally, I would like to thank the team behind the *EnergAIze* project, which this solution is a part of. They provided me with valuable tips and advice during both the development of the application and the construction of this report.

Resumo

A integração de energias de fontes renováveis no setor elétrico tem encontrado vários desafios. Os utilizadores da rede têm investido em painéis solares, cujo preço tem sido cada vez menor, aumentando excessivamente a produção energética. Para além disso, esta produção ocorre maioritariamente durante o dia, sendo mínima após o pôr do sol. Em contrapartida, o consumo é mínimo durante o dia e máximo depois do pôr do sol. Este desfasamento compromete as vantagens económicas e ambientais da utilização de energia renovável, prejudicando a sua disseminação.

Uma das soluções mais promissoras para este problema é a flexibilidade energética, que mede a capacidade de um dispositivo ajustar dinamicamente o seu consumo energético. Um dos dispositivos mais flexíveis e, por isso, valiosos para o balanceamento da carga da rede, são os carros elétricos. Este projeto endereça diretamente o perfil de carregamento de carros elétricos numa rede elétrica, deduzindo a flexibilidade (Flex Offer) a partir de dados históricos dos seus utilizadores e inferência estatística, requerendo mínima participação ativa da sua parte. Isto permite gerir o equilíbrio da rede elétrica com mínima intrusão aos seus utilizadores.

No geral, a solução desenvolvida oferece resultados satisfatórios perante a sua principal premissa. Contudo, a apropriação dos modelos estatísticos utilizados deve ser revista no futuro, com o intuito de refinar os resultados perante os dados disponíveis de cada utilizador.

Palavras-chave (Tema): Flexibilidade Energética, Comunidades de Energias Renováveis, Smart Grids

Palavras-chave (Tecnologias): Kernel Density Estimation, Python, Inferência Estatística, Pré-Processamento de Dados, Flex-Offer

Abstract

The electricity sector is facing many challenges when it comes to integrating renewable energy in electric grids. The biggest challenge is the phase shift between peak energy demand and consumption, that ends up negating the advantages that come with using renewable energy sources.

Energy flexibility proves to be an effective solution for this problem, given it defines the ability of an appliance to dynamically adjust their energy consumption to the energy production in the grid. This project focuses on predicting the flexibility for electric vehicles because of their high flexibility and value to grid load balancing. This is done by applying statistical models to charging data and, through statistical inference, estimate intervals for the flexibility.

This solution provides satisfactory results for predicting the energy flexibility of electric vehicles. However, the statistical models that were used may be reviewed in the future, in order to refine the results and get better approximations for the flexibility predictions.

Key words (Theme):	Energy Flexibility, Renewable Energy Communities, Smart Grids
Key words (Technologies):	Kernel Density Estimation, Python, Statistic Inference, Data Pre-processing, Flex-Offer

Table of Contents

Table of Figures

List of Tables

Table of Algorithms

Table of Code Snippets

Notation and Glossary

AI	Artificial Intelligence
API	Application Programming Interface. Set of methods for the communication of different software components.
EnergAIze	User-centric energy management framework
EV	Electric Vehicle
Flex Offer	Concept that describes the flexibility of a device
Flexibility	The ability of changing a device’s energy consumption dynamically
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science. Private non-profit technology research association
ISEP	Instituto Superior de Engenharia do Porto
KDE	Kernel Density Estimation. Non-parametric method for PDF estimation
OPEVA	OPtimization of Electric Vehicle Autonomy. European project that aims to optimize the autonomy of electric vehicles.
PDF	Probability Density Function. Function that describes the probability of a random variable
Python	High-level general-purpose programming language
RE	Renewable Energy
RL	Reinforcement Learning. Type of machine learning based on receiving feedback in the form of rewards or penalties.
RGPD	General Data Protection Regulation, European Union legislation on data protection and privacy
SoC	State of charge of a device
SoftCPS	Research group at ISEP
Statistical Inference	The process of using sample data to deduce conclusions about the population it is derived from
UML	Unified Modelling Language, a software modelling language

1 Introduction

This chapter begins with a broad overview of the thesis, highlighting the context and motivations that inspired this work. Then, it refines the scope, offering an insight into the problem description and the proposed solution. Finally, it describes the project objectives, approach and planning.

1.1 Context

In recent years, there has been a growing global preference for the adoption of Renewable Energy (RE) sources. As global energy demand continues to rise, fossil fuels have increasingly proven to be an unsustainable and environmentally detrimental solution to meeting these needs. In contrast, renewable energy offers a viable alternative, capable of mitigating the adverse environmental impacts associated with fossil fuel consumption. Moreover, due to their naturally replenishing nature, renewable sources present long-term economic advantages by reducing dependence on finite resources and enhancing energy security [1].

However, RE integration in electric grids is challenging. RE generation is highly variable since it depends on weather conditions, such as the wind and the sun, which makes its availability fluctuate throughout the day [1], [2], [3]. For example, there is a high RE production during the day, through photovoltaic panels, (reaching overgeneration levels), which gets minimal by sunset (Figure 1). This clashes with peak energy demand which happens after sunset [4], resulting in time imbalance between peak demand and ER production, hindering the advantages of RE usage and even exacerbating some of the disadvantages of using fossil sources [1]. This means that overgeneration needs to be mitigated, otherwise there is no point in using RE sources.

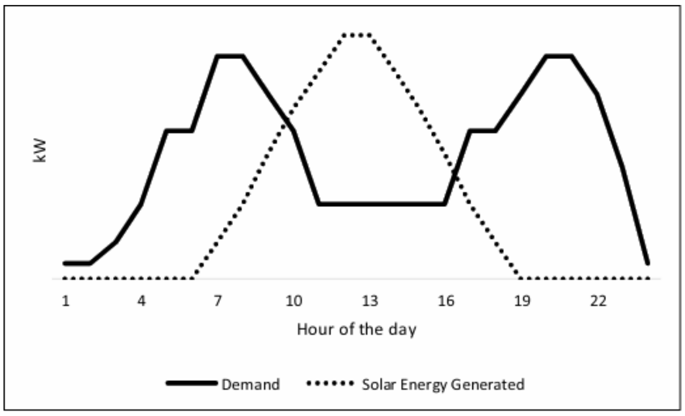


Figure 1 – Peak demand and energy production phase shift [5]

One way of coping with this problem is through adding dump loads. Water heating systems can use the surplus electricity to provide hot water for common household usage.

However, these systems have a limited capacity which is rapidly reached, requiring system shutdown [6]. Load bank resistors can also be used to absorb excess energy, but this proves to be highly wasteful since the energy isn't being given any productive value [6]. And even though battery storage is capable of storing the generated electricity surplus when peak is at its highest, it requires expensive ongoing maintenance [6].

Another better solution is to integrate energy flexibility management in the electric grid system. Energy flexibility is the ability of changing energy consumption in time and/or amount [3]. The use of flexibility allows energy consumption scheduling, resulting in a maximized usage of renewable sources [7] while minimizing energy cost [8] and CO2 emissions [9]. Peak supply and demand can be balanced given how many of household appliances, like EVs or washing machines, can have their energy demand shifted to the time when production from RE sources is available [2]. This solution avoids both unnecessary costs and waste, since it just operates with the usual provided energy amount, only in different time stamps.

This project was developed for the Software for Cyber Physical Systems Lab (SoftCPS), a research group of the School of Engineering of the Polytechnic Institute of Porto (ISEP), integrated in the INESC TEC [10]. This project was also developed in the scope of the European Union Optimization of Electric Vehicle Autonomy (OPEVA) project, whose purpose is to accelerate the deployment of sustainable EVs by means of optimizing their support in environments where mobility has been predominantly powered on fossil fuel [11]. The OPEVA project is partnered with many European organisations [12], and this solution highly benefits from the partnerships with i-charging and Cleanwatts, who provided real EV charging data, essential to probe the estimated flexibility.

The motivation for working on this project comes from the opportunity to work with a team whose objective is to implement solutions that promote a sustainable future as well as apply the knowledge gathered throughout the Informatic Engineering degree, such as good software development practices, problem solving and teamwork.

1.2 Problem description

1.2.1 Approach

The approach taken in this project started by performing research and analysing the current state of the art of the problem at hand. Then each dataset available in [13] was studied for further contextualisation and adaptation to the goal of the project. When the partners' i-charging and Cleanwatts data was available, all gathered knowledge was used, starting the implementation phase. Furthermore, every research result was discussed with the team so the solution could be refined regarding every component of the system it belongs to. The development was iterative, i.e. whenever a technique was applied to solving the problem it came with its set of challenges that were surpassed, in order to make the final solution as sophisticated as possible.

This project is part of the *EnergAIze* framework [14], whose architectural design was done using Visual Paradigm. This tool was also used for designing this project's solution (Section 3.3), ensuring the established workflow and avoiding design incompatibilities.

The project did not specify what development tools or technologies were to be used and, after careful analysis, it was decided to implement the solution using the Python programming language, given how it involves large data analysis for building the statistical inference models.

This project's objectives revolve around software design that complies with other components of the *EnergAIze* system. The objectives are as follows:

Objective 1. Predict user flexibility based on historical use.

Objective 2. Store user defined flexibility**.**

Objective 3. Receive user defined flexibility and prioritize it over generated flexibility.

1.2.2 Project Contribution

The effective operation of Renewable Energy Communities (RECs) relies heavily on the ability to schedule and control energy consumption and storage based on the flexibility of individual members. Flexibility information, such as when an electric vehicle (EV) will be available for charging or discharging, or when a building can shift heating, ventilation, and air conditioning (HVAC) loads, is essential for optimizing community-level objectives like cost minimisation, self-consumption maximisation, or peak shaving (some of the problems highlighted by the context Section 1.1).

In frameworks such as *EnergAIze* [14], to which this project actively contributes, predicted flexibility is used as input for multi-agent control algorithms that coordinate distributed energy resources across the community. These controllers depend on accurate forecasts of user availability and willingness to adjust consumption in order to make timely and effective energy management decisions. For further reading, the paper in [15] explains how the

EnergAIze project aims to adapt promising new technologies to real-world deployments, as well as the *Flexibility Prediction* component's role in it.

However, obtaining such flexibility information often requires user input, such as EV departure times or current state of charge (SoC), which can be perceived as intrusive and lead to low participation or engagement. To address this challenge, statistical and machine learning models can be developed using historical data from individual users. These models enable the prediction of flexibility without requiring explicit user input, thereby preserving user privacy while still supporting intelligent scheduling and control. This report explores such predictive approaches (refer to 3.1), aiming to balance user autonomy with the operational needs of REC energy management systems.

Although this project works in the absence of user input data, it doesn't mean it is incompatible with it. This means that for systems which already implement energy flexibility management, it is still possible to integrate this solution as a fail-safe option, making user flexibility measuring still possible despite the lack of specific user input. Ensuring flexibility in renewable energy communities helps accelerate the adoption of renewable energy sources globally, thereby contributing to the planet's environmental sustainability. Besides that, implementing energy flexibility strategies within renewable energy communities enables both consumers and utility providers to optimize energy use and reduce operational costs.

For consumers, this means tangible savings on energy bills by adjusting usage based on price signals or availability of renewable energy. For utilities, enhanced flexibility reduces the need for expensive grid upgrades or reliance on fossil energy source to handle demand surges. It also allows for more efficient energy distribution and planning, ultimately leading to a more cost-effective and sustainable energy system.

This project also proves useful to organisations such as i-charging [16] and Cleanwatts [17] whose objectives are to innovate in the electric mobility segment and to provide for people who wish to play an active role in energy transition, ensuring economic and environmental benefits.

1.2.3 Planning

An Agile approach was used, in which the work was developed in an iterative and incremental manner. This way, the solution evolved progressively, quickly adapting to new requirements [18].

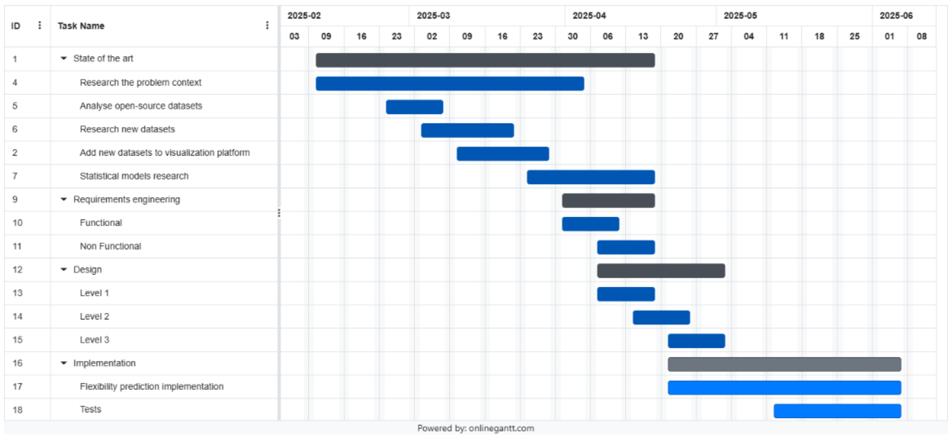


Figure 2 – Gantt Diagram for the project planning

As evidence in Figure 2, in the initial phase, a state-of-the-art review was conducted to ensure integration within the context of the problem to be solved. Next, an analysis of open-source data on the topic was conducted with the aim of recognizing patterns and assisting in the design of the solution. Further research was also endorsed with the objective of finding more datasets than the ones provided by the team members, deepening knowledge in the scope of the project. Found datasets were included in a website [13] for easy visualisation. Finally, the requirements were defined, the solution designed, and the implementation phase began.

Communication among team members was conducted via email and Microsoft Teams, with status updates presented in person at ISEP. In addition, meetings were held with organisations affiliated with the OPEVA project, which served to present the conclusions drawn regarding the resources they provided.

1.3 Report Structure

The following section explains the report structure as well as a brief description of each chapter.

- **Chapter 2 – State of the Art:** Explains the problem context, as well as related projects in the scope of the problem and what technologies are of interest.
- **Chapter 3 – Requirements Engineering Analysis and Design:** Defines the problem domain analysis and the adopted design on the problem solution, alongside all related documentation.
- **Chapter 4 – Solution Implementation:** Describes in high detail the most important parts of the solution implementation. Also points out used testing modules.
- **Chapter 5 – Conclusions:** Summarizes previous chapters, presenting strong aspects of the developed solution and what can be improved in the future.

2 State of the Art

This chapter serves as a mean for the reader to understand the scope of the project. First, there will be a brief contextualisation on the problem that originated this required solution. Then, there will be an overview of related projects that have already tackled the same problem, which ended up having an influence on this project. Finally, there will be a summary of already existing technologies and how they helped shape this project's solution.

2.1 Context

Over the years, we have seen a steady global increase in energy demand [2], and it is projected to keep rising [19]. Because of this, the downsides of using fossil fuels, particularly the greenhouse gas emissions, are exacerbated and start to represent a more imposing problem [2]. The alternative resides in renewable energy sources, such as the sun or the wind, to fulfil the population's energy demands while preventing climate changing gas emissions. In recent years, solar panels are becoming a trend, given how cheap solar energy is [20], which implies both economic and environmental advantages in using renewable energy sources instead of fossil fuels.

However, building a society based on renewable energy sources comes with its own set of challenges. Energy production from renewable sources is unpredictable since it depends on weather conditions [2]. This unpredictability and the increase in solar energy adoption creates a problem pointed by the "duck curve"[4]. This duck-shaped curve shows the difference between electricity demand and the amount of available solar energy throughout the day (Figure 3).

Throughout the day, there is higher solar production around midday which coincides with the moment when demand is at its lowest. On the other hand, solar production is at its lowest after the sun sets, when demand peaks. This overgeneration forces grid operators to either curtail renewable energy sources or establish negative prices to bring the energy demand upwards [1]. Both options go against the economic and environmental advantages previously mentioned, respectively.

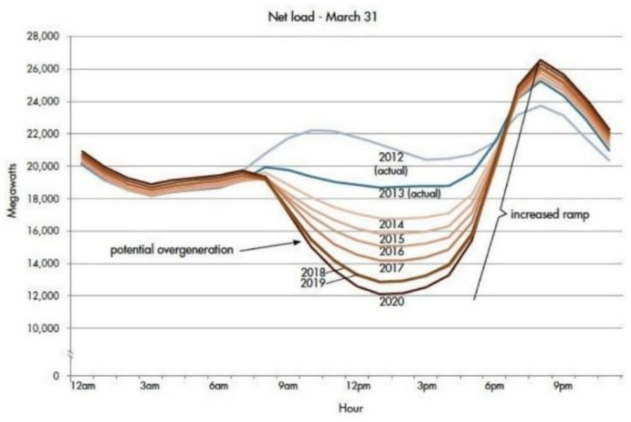


Figure 3 – CAISO's duck curve over the years [4]

Although not applicable to all home appliances, it is possible to shift certain devices' electric demand to coincide with times of higher energy production, mitigating the duck curve. This is called Energy Flexibility [1], and it is vital to the grid-load balancing of renewable energy communities. It can be represented through the Flex-Offer (FO) model introduced in [2] which is a way to define a generic abstraction of what energy flexibility is [1]. An FO may represent many flexibility attributes, namely start and end time (of energy influx) and total energy used. In Figure 4 is represented an FO example, where each x axis unit corresponds to an arbitrary time slice, which for this hypothetical case can be of 15 minutes. This representation is characteristic of an EV charging pattern.

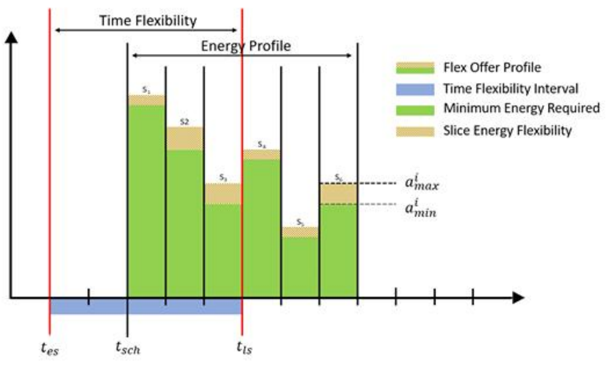


Figure 4 – Flex Offer example [1]

Considering an EV user who arrives home at 7 PM. They establish that by 7 AM the next day the EV must be charged with at least 70% battery. In Figure 4, the t_{es}, earliest start time, would be 7 PM, since the EV can start charging right away upon arrival. However, the definition of flexibility allows for later start times, the latest being t_{ls}, while still ensuring the pretended amount of energy by the end of the charging session. This is crucial since it allows for mitigating the phase shift between peak demand and production. These time stamps define the interval when the FO can be scheduled (t_{sch}) and satisfy the user's energy needs. Besides time flexibility, Figure 4 also shows energy flexibility, where a_{min} is minimum energy and a_{max} is maximum energy.

The MIRABEL (Micro Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply, and Distribution) system [2] also defines entities called prosumers as users that produce and/or consumer energy on the electric grid. The FO concept not only refers to demand, but also energy production. Users can thereby offer flexible energy production on their ends, e.g. using photovoltaic panels (PV), to support the total energy supply available on the grid [2]. These types of FOs are put on an energy market, where energy supply and demand on the electric network are evaluated, resulting in optimal scheduling of the FOs and grid load balancing [2]. This project aims to partake in this process, resorting to user historical data to generate energy flexibility automatically, rather than asking consistently for users to input their preferences, which can lead to user disengagement and affect the overall system action.

Flexibility types

In terms of flexibility, devices can be categorized according to two factors: slice energy flexibility and time flexibility [1]. This categorisation leads to 3 different home appliance types:

I. **Fixed Devices –** Devices whose consumption period and amount of energy consumed can't be changed, like televisions or lights.
II. **Shiftable Devices –** Time-flexible devices, i.e. devices whose consumption period can be modified within certain limits without modifying the load profile, allowing for grid load management optimisation. Examples of this device type include washing machines and dishwashers.
III. **Elastic Devices –** Adjustable devices both in terms of usage time and power consumption. Devices of this type are highly valuable in grid load management optimisation. Electric vehicles are categorized as being elastic devices.

Given how important elastic devices are for managing the electric grid load, this project focuses on analysing the data around electric vehicle charging profiles for that same purpose.

Flex-Offer Life Cycle

Before being approved and scheduled, an FO passes through many phases [21]. Figure 5 gives a general view of the Flex-Offer cycle.

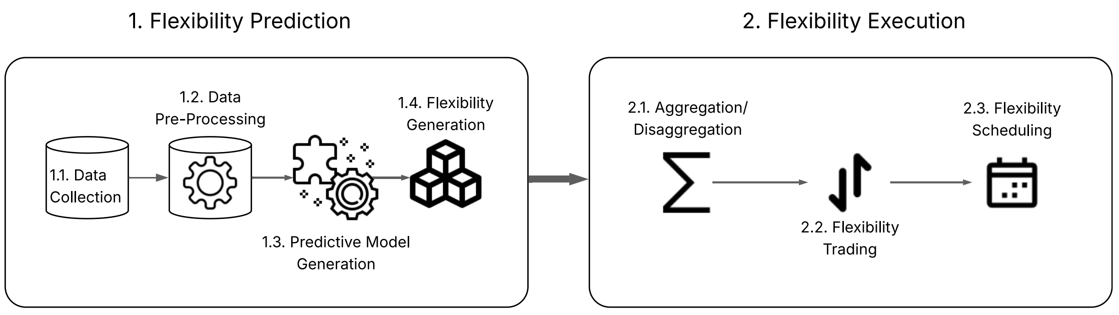

1. Flexibility Prediction
2. Flexibility Execution
1.1. Data Collection
1.2. Data Pre-Processing
1.3. Predictive Model Generation
1.4. Flexibility Generation
2.1. Aggregation/ Disaggregation
2.2. Flexibility Trading
2.3. Flexibility Scheduling

Figure 5 – Flex-Offer life cycle

This project's goal is Flexibility Prediction, which focuses on:

- **1.1. Data Collection** – The first phase involves data collection on energy consumption data from flexible devices. This is a very important part for this project given how predicting models get more sophisticated the bigger the sample size is (refer to Section 3.1).
- **1.2. Data Pre-Processing** – Collected energy is often riddled with outliers and missing values, which are removed in this phase. Furthermore, given how this project focuses on energy usage patterns, data may prove invalid under certain conditions, such as holidays or even different days of the week.
- **1.3. Predictive Model Generation** – Pre-processed data is used to build the predictive model.
- **1.4. Flexibility Generation** – The energy flexibility is deduced based on historical data.

2.2 Related Projects

The following section presents some related projects. Their analysis provided insight into the current state of energy flexibility integration in RECs as well as highlighted certain characteristics that this project aims to improve.

2.2.1 MIRABEL Project

MIRABEL is a research initiative funded by the European Union, designed to enhance the integration of renewable energy sources (RES) into the power grid [2].

In order to balance supply and demand effectively, the MIRABEL system is based on energy flexibility given by the participants in the energy market. This project established a new entity called prosumer, given how participants can act as both consumers and producers of energy [2]. The RES usage is of difficult integration given the unpredictable nature of energy production that comes with it. This system tackles this issue [2] by integrating forecasting techniques, providing a more sophisticated solution when compared to earlier approaches. However, this system still relies solely on active user input for certain aspects of their flexibility [2], which in the long term can lead to less cooperation on their part, due to it being slightly intrusive. This project's solution addresses this problem by providing an automated alternative based on user energy consumption data, requiring no input whatsoever.

2.2.2 Universal Smart Energy Network (USEF)

The Universal Smart Energy Framework (USEF) is a project developed by the Smart Energy Collective (SEC). This framework follows a set of implementation guidelines whose main purpose is to allow for establishing a completely functional smart energy system [22].

The USEF developed was motivated by the global necessity to reduce CO2 emissions and reduce the fossil fuels dependence in favour of increasing the use of RES, like the wind and sun, as well as satisfying the growing demand for energy in the electric transport sector [22]. The traditional energy system design, based on a top-down approach, does not allow for bidirectional traffic or new market roles like prosumers, energy service companies, and aggregators, leading to impractical solutions when trying to integrate new services and technologies [22]. USEF envisioned the transformation of this system as possible provided the close cooperation between all active participants and industries [22].

This framework manages this transition by integrating energy flexibility management services, such as EV charging and a market-based system where gained flexibility can be offered [22]. As stated by the USEF, EV are a core component in controlling the flexibility of the grid system and as such it is the primary focus of this project. However, the framework did not picture any automation regarding energy flexibility, which means the grid users will be often prompted to input their energy preferences, which can lead to user disengagement and overall hinderance of the grid's performance.

2.2.3 GOFLEX

GOFLEX is a European project that aims to transform the traditional energy system by integrating distributed flexibilities and dynamic pricing as key components for enabling better RES usage [23]. This project is motivated by the main focus in the European energy landscape, which is the need to shift from a centralised to a decentralised energy production and management [23].

GOFLEX focuses on delivering scalable, commercially viable, and easy-to-deploy technological solutions that can be integrated into existing infrastructures and ecosystems. Its main purpose is to make distributed flexibility and dynamic pricing market-ready by developing an integrated modular technology platform [23]. This platform enables actors such as prosumers, generators, energy suppliers, microgrid operators, and energy communities to aggregate and trade energy flexibilities in a seamless and efficient way [23]. GOFLEX already improves on previously mentioned aspects by estimating flexibility, utilising machine-learning models with a probabilistic nature [24], which helped shape this project’s solution.

3 Requirements engineering and Design

This chapter will define the project's requirements and its design.

3.1 Solution Approach

This solution revolves around predicting the users' flexibility based on their historical energy use, which is possible through the application of statistical models to the sample, with which the flexibility is estimated via statistical inference.

The following sub-sections focus on explaining technical details about the statistical inference procedure, as well as the different models that were studied during development.

3.1.1 Probability Density Function

Figure 6 and Figure 7 show the flexibility patterns for a user's EV from the open-source dataset in [25]. It is easy to see the random nature of these values, which is a common characteristic of real data. Because of this, it is very difficult to predict the exact values for these attributes. For example, traffic fluctuations directly impact the user's time of arrival and other variables like driving speed can influence the battery's SoC, affecting the total energy needed for charging the car. However, there are patterns to flexibility, since people have schedules and routines and often use the same itinerary when moving from their workplace to home. Additionally, different days of the week may impose different schedules and even holidays can interfere with these patterns.

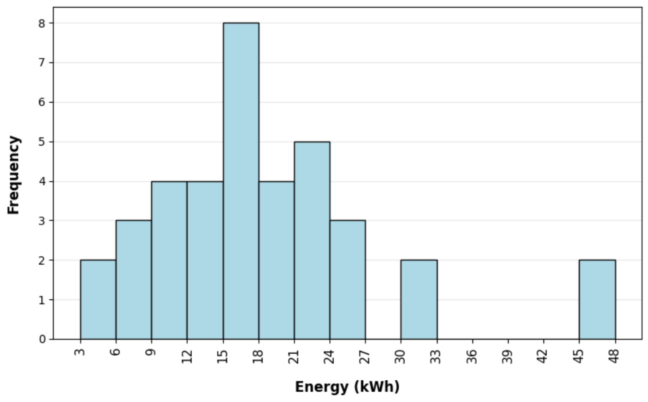


Figure 6 – Energy charged per charging session

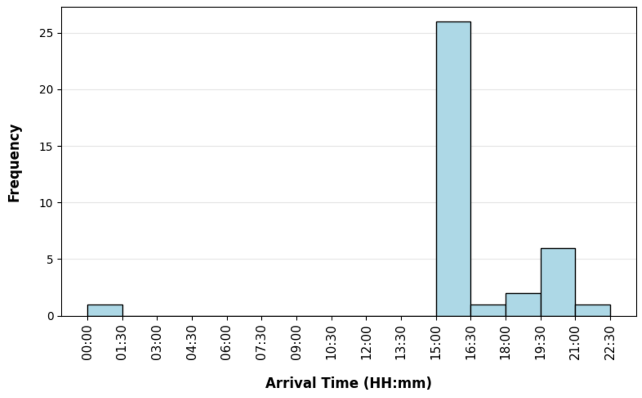


Figure 7 – Arrival time for each charging session

The many possible outcomes seen in charging sessions for a particular user can be represented using random variables [26]. Random variables can be discrete, when they assume only a finite number or an infinite sequence of values, or continuous, when they assume any value in an interval on the real number line [27]. The pretended flexibility parameters are of a continuous nature because both time and energy can take any value within a specific interval of real numbers. Furthermore, the visualisation of samples using appropriate graphs highlights possible patterns in the charging sessions of a specific user. For example, Figure 6 shows a higher number of cases where this user charges between 15 kWh and 18 kWh, and Figure 7 shows a preference for arrival time between 15:00 and 16:30.

The probability of these occurrences is described by the probability distribution of the continuous random variable, defined by the probability density function (PDF) [27]. This function is built on the sample of the random variable, from which the probability of that variable assuming a value within a certain interval is extracted [28]. For example, this can be applied by defining functions that represent the duration of a session, or the energy charged, allowing these flexibility attributes estimation.

If X is a continuous random variable and *f(x)* (Figure 8) is the corresponding PDF, the probability of X assuming within the [a, b] interval (where b > a) is given by the area bellow the curve defined by *f(x)* [28]. One important aspect of the PDF is that it follows two important conditions. First of which is that, for all values of X, *f(x)* is positive [28], for all probabilities are assigned on a scale from 0 to 1 [27]. The other condition is that, since the PDF is built on the sample, the total area bellow the function must equal one [28], because it encompasses all possible values for X.

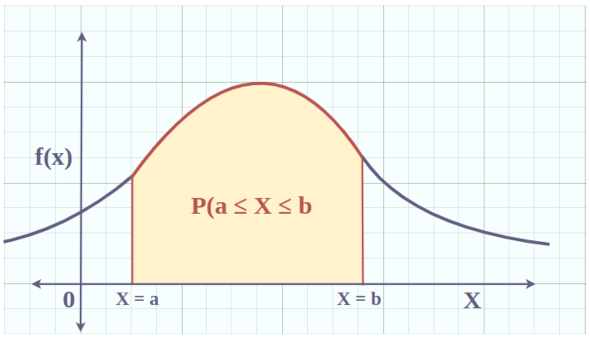


Figure 8 – PDF example with highlighted area between “a” and “b” [29]

One last remark about using the PDF for probability calculation, since it relies on integral calculus, the probability of X being equal to an exact value is zero as shown in Eq (1) [30].

$$\int_a^a f(x)dx = F(a) - F(a) = 0 \qquad \text{Eq (1)}$$

The PDF is used to learn about the characteristics of a large group of elements called the population [27], which in this project's context is all charging sessions of a user. However, most of the time it is not possible to have all data relative to one user. For example, data collection might have started sometime after the user started charging their EVs or even the system holding that information might implement a data expiration mechanism where old data is deleted. Besides, data can become corrupted and unusable for this matter. Nonetheless, a sample can reasonably represent the population it is taken from [31], allowing for deducing the random variable's characteristics, such as population mean and variance, through statistical inference [27].

In statistics, this is achieved through interval estimation, which is typically accompanied by a confidence level indicating the probability that the interval, called confidence interval (CI), contains the specified population parameter [27].

3.1.2 Normal Distribution

The normal distribution is the most widely used PDF due to its many applications and the fact that it approximates many natural phenomena very well, such as population height or birth weight [32]. Figure 9 demonstrates its characteristic symmetric bell shape as well the fact that most scores lie close to the centre of the distribution, reducing their frequency as they drift away from the centre [33].

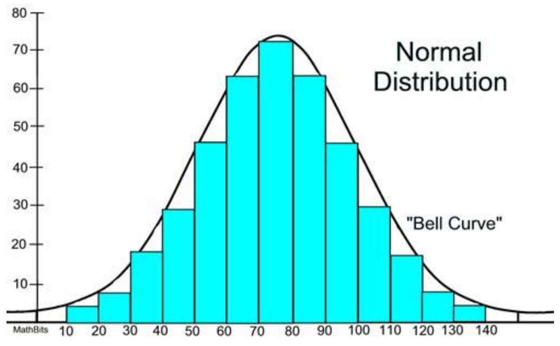


Figure 9 - Normal distribution example [34]

Given its common presence in nature, there are many statistical tests based on the normal distribution, called parametric tests [35]. To evaluate the normality of a sample it is possible to perform Kolmogorov-Smirnov test or the Shapiro-Wilk test [33]. However, there is an interesting proof in probability theory that states that large enough samples tend to follow a normal distribution [36]. This is known as the central limit theorem (CLT) and a good rule to follow is that samples with size $n \geq 30$ follow a normal distribution [36].

The normal distribution is easy to work with mathematically [37] and it makes up for simple CI calculation. By defining, e.g., a 95% confidence level, it is implied, by the PDF definition in Section 3.1.1, that the area between the interval defined by *a* and *b* (Figure 10) is 0.95 units. Since the curve is symmetric around the mean, we can deduce the CI by splitting the 0.95 area in half and calculating the corresponding values to the left and right of the mean [38]. The remaining areas, referred to as *Area* in Figure 10, add up to 5% which represents the probability of the population mean not falling inside the calculated CI.

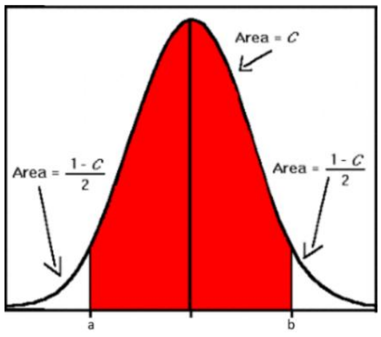


Figure 10 – CI calculation, where C = 0.95 [38]

3.1.3 Kernel Density Estimation (KDE)

Despite normal distribution being common in real world scenarios, it is not always the case. Normal distribution can be tested, but for smaller samples neither the CLT nor statistical tests [39], like Shapiro-Wilk, can be appropriately applied to define a sample as normally distributed. In the absence of enough evidence to assume data normality, parametric tests are no longer a viable option for statistical inference, and non-parametric tests should be used instead, given how little assumptions they make about the sample.

When data is too irregular, KDE provides a way of estimating the PDF based on local density, i.e., for a value present in the sample there is a high probability that in the population there will be values equal to or in the near vicinity of that same value [40]. Via KDE, the PDF can be deduced using (Eq (2) [41].

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} K_h\,(x - x_i) \qquad \text{Eq (2)}$$

To deduce *f(a)*, the probability of the random variable assuming a value *a*, the KDE uses kernel functions defined by *K*. A kernel function is a tiny PDF (for example a normal distribution) built around each data value present in the sample. In Figure 11 it is shown how these kernels (dashed lines) are centred in specific values and, being PDFs, they can also be used to estimate the probability of values in the vicinity. This property allows for probability calculation on behalf of multiple PDFs, which could determine how likely it is for a value in the population being lower than 3.0, for example. By taking the average of the resulting probabilities, the KDE defines a general idea of the likelihood of an event, while taking into consideration each sample data-based kernel. For the previous example, Figure 11 shows that the resulting KDE overlaps the left-most kernel, which makes sense because it is the only with the data to support occurrences in the vicinity of $x = 3.0$.

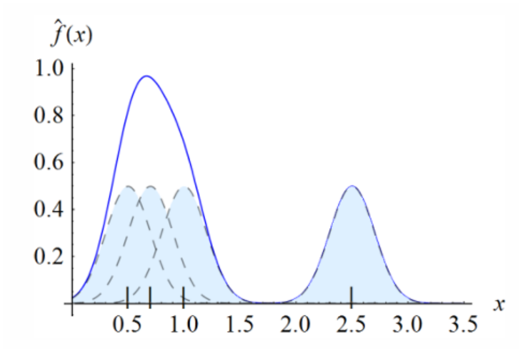


Figure 11 – KDE visualisation [41]

One remark about KDE is that it allows for statistical inference for small samples. Samples as small as 4 data values result in a relative mean square error of just 0.1 [42].

Finally, the *h* parameter represents the bandwidth chosen for the model. The bandwidth is a smoothing parameter, meaning the higher the smoother the PDF curve will be (Figure 12), and it is a crucial aspect of the kernel function because over smoothing leads to loss of important data and, on the other hand, under smoothing prevents taking any reasonable conclusion about the population, given how heavily the model is adapted to the sample [40].

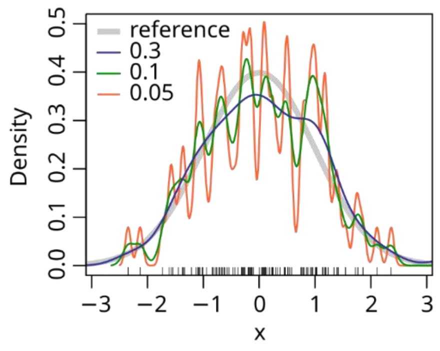


Figure 12 – How different smoothing factors affect the curve shape [43]

Defining a good bandwidth determines the quality of the model and it can be chosen by finger or, more optimally, using numerical models based on the sample's characteristics [40].

The easiest and most used models are called Rules of Thumb. Examples of these are the Scott and the Silverman methods, that rely on sample size and variance to get a reasonable value for the *h* factor [40]. Despite being easy to use and computationally fast, they make assumptions on the population's distribution, assuming its normality, which can be misleading [44].

Other models involve iterative procedures that test the available data many times to obtain progressively more refined smoothing factors [40]. Plug-in and Cross Validation are some examples, where the first uses the sample to test its characteristics multiple times and the later focuses on trying different *h* factors. However more accurate than Rules of Thumb, these procedures end up being computationally heavy [40].

The next section focuses on establishing the engineering requirements for the solution. The most appropriate statistical model will be deduced after carefully evaluating the system's requirements.

3.2 Requirements Engineering

Whenever a stakeholder places an order for a software product, it is necessary to understand, record and manage their demands. The process of defining the requirements that such product must have to match the client's expectations is called requirements engineering [45].

This is an important phase of the development of a project since it helps identify potential issues early, allowing for quick adjustments. Besides, it can help communication between the developers and the stakeholders, which ends up contributing for a better understanding of the client's needs and their implementation in the final product [45].

3.2.1 Functional Requirements

This section states the project's features that need to be implemented. These features are also referenced as being the functional requirements of the system [46].

After discussing with the project consortium, the functional requirements were defined as follows:

FR 1. The system must be able to predict user energy flexibility and create flexibility objects using statistical inference.

  a. This includes the following flexibility parameters:

    i. Starting time of a charging session;

    ii. Energy charged (kWh) of a charging session;

iii. Duration of a charging session.

FR 2. The system must be able of persisting the calculated flexibility objects. These flexibility objects are used by other components of the *EnergAIze* system and must be stored.

FR 3. The system must log all components actions. Upon system failure this helps pinpoint what went wrong and improve the maintainability of the system.

FR 4. The system should allow user to define their flexibility manually.

FR 5. The statistical models used by the system should be able to include new data that becomes available over time. Statistical models become more sophisticated the more data they have access to, so including new available samples helps refine flexibility predictions.

3.2.2 Non-functional Requirements

Non-functional encompass the criteria that define how a system should behave. Instead of describing system functions, as in functional requirements, these focus on different aspects like performance, scalability or security [47].

The non-functional requirements established for this project are the following:

Security

System security defines the constraints that ensure its robustness, as well as its access management [48].

Ensuring data privacy is a significant aspect of a system when it comes to guaranteeing lawful and ethical handling of personal matters [49]. This project revolves around using user information regarding their schedules. This is seen as sensitive information and must be treated carefully to prevent external actions with malicious intent.

NFR 1. The persistence of the generated flexibility objects should be done in safe channels where the access is managed.

NFR 2. User data must be treated anonymously, to ensure RGPD compliance.

Usability

System usability defines how simple and intuitive it is to use while still meeting the users' needs [48].

This project aims for predicting flexibility the day before it is needed, doing so automatically.

NFR 3. Upon request, the system must perform its tasks each day automatically, requiring minimal user intervention.

Performance and Scalability

The system performance defines its capabilities on returning the results that were requested. This requirement is paired with scalability, which determines the upper limit for workload until performance starts being affected noticeably [50].

The main objective of this project is to predict the energy flexibility patterns of energy communities with up to hundreds of users, which involves managing large amounts of data. This imposes precautions on choosing efficient algorithms for achieving the project's purpose without compromising the workflow of the system it is part of.

NFR 4. Queries to user database should request just the needed information.

NFR 5. The chosen algorithms should be lightweight and perform the assigned tasks within a time limit of 1 hour.

Reliability

Establishing a reliable system ensures its operation is consistent under both normal and exceptional conditions [51].

This project's focuses on providing a way of obtaining energy flexibility for a better electric grid management, hence the need to ensure its operation even under suboptimal circumstances.

NFR 6. The system should log error messages clearly to facilitate troubleshooting and bugfixes.

NFR 7. The system needs to be able to automatically detect failures and immediately utilise failover mechanisms, to ensure uninterrupted service in case of other components failure. Further detail on how the system manages this can be found in Sections 3.4.2 and 3.4.3.

3.2.3 Chosen Statistical Approach

The API used for data retrieval is recent, meaning there was little information about the charging profiles of the system's users. Because of this, the procedures involved in predicting flexibility relied solely on KDE since, as stated in Section 3.1.3, it provides reasonable results for small samples. In the future, other models may be implemented so, under different circumstances, the system may opt dynamically between different approaches, based on the sample's characteristics, to ensure the best results possible.

3.3 Design

Software design is critical in translating the stakeholder's needs into "blueprints" for building the software, basically linking the problem and solution spaces [52]. These blueprints describe the components and relationships between them that the system must encompass for meeting the functional and non-functional requirements [52].

This chapter defines, both the approach taken on modelling the system, and the system's architecture.

3.3.1 C4 Model

The C4 model was created to help communication about software architecture between software development team members, by representing the system's structure at different levels of detail [53]. This is often compared to the zoom functionality of Google Maps (Figure 13), where an area can be zoomed out to get a more general idea of the region, or zoomed in to view the local streets and alleys, for example.

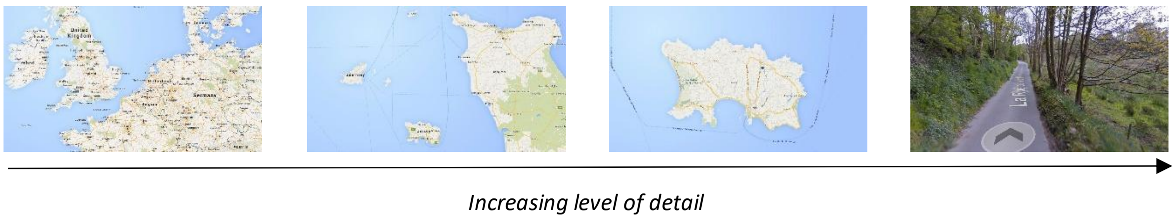


Figure 13 – Levels of detail of the C4 model [53]

This model specifies the following levels of detail [54]:

Level 1. **System Context** – This is the starting point and the less detailed representation of the model. It represents the software scope as well as its interaction with other systems and users.

Level 2. **Container** – The core components of the system are "zoomed in" and higher level of detail is attained, revealing the dynamics between the containers.

Level 3. **Component** – This level offers a closer look at individual containers and its components, showing how they relate to each other to fulfil the containers tasks.

Level 4. **Code** – Highest level of detail in the model, which is represented by the implementation of the component itself.

3.3.2 4 + 1 View Model

The 4 + 1 view model describes software architecture using a set of concurrent views of the system [55]. These views represent the perspective that end-users, developers and system engineers have simultaneously on the system. Addressing each distinct stakeholder's concerns provides a comprehensive understanding of the system [56], culminating in better results overall. The model proposes the following 5 views [56]:

- **Logical** – Focuses on the static structure of the system, representing its essential components.
- **Process** – Approaches the dynamic aspects of the system, like its behaviour between different activities/processes.
- **Physical** – Reveals how the components are deployed in physical infrastructure.
- **Development** – Establishes the mapping between the software architecture and its implementation.
- **Use Case/Scenarios** – This view differs from the others by emphasising the system's functionality from a specific role's standpoint instead of its internal aspects.

Figure 14 represents the 4 + 1 model and the relationships between each view.

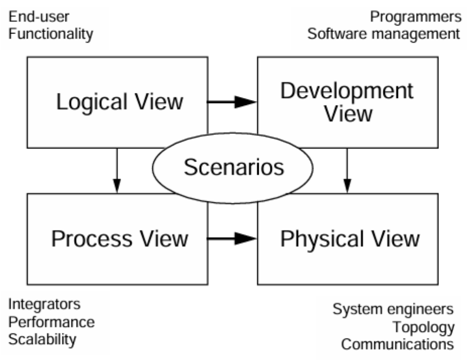


Figure 14 – The "4 + 1" view model [57]

3.3.3 C4 and 4 + 1 Models Representation

Establishing a hybrid model that combines both previous models allows the static modelling of the system, under various levels of detail, while taking into consideration different perspectives of the system [58].

For representing this hybrid model, the Unified Modelling Language (UML) can be used, some of the most used UML diagrams being:

- **Class** – Describes the types of objects in the system and their different relationships.

- **Component** – Depicts how components are wired together to form larger components.
- **Deployment** – Helps to model the physical aspect of a software system. Puts into perspective where the software is deployed in the real world.
- **Use Case** – It models the system's functional requirements in terms of use cases.
- **Sequence** – Models the collaboration of the software's objects based on a time sequence.

3.4 Architectural Design

The following section describes the structure of the system according to the hybrid model mentioned in Section 3.3.3.

3.4.1 Level 1

This section documents a panoramic view of the system.

Logical View

This system is encompassed by the *EnergAIze* project [14] and Figure 15 depicts it and its components.

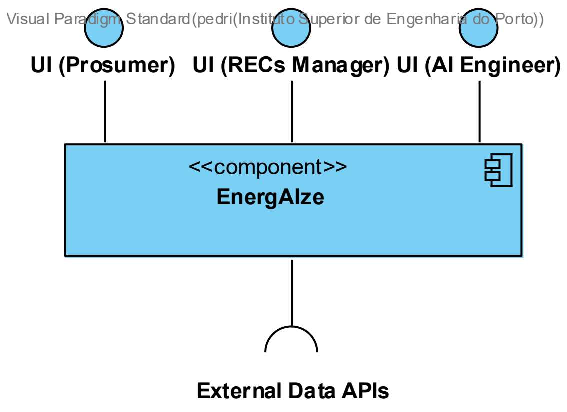


Figure 15 – Level 1 Logic View of the EnergAIze system

The EnergAIze system allows user-centric energy management [14] and encompasses tools that allow energy and flexibility scheduling. The *EnergAIze* system's purpose is to provide intelligent energy flexibility management within RECs [15]. The system focuses on energy demand peak shaving, energy sharing within the grid and improved renewable energy utilisation.

The *Flexibility Prediction* component complements the *EnergAIze* system by providing essential flexibility estimation that optimizes the energy flux within RECs, helping to balance the grid energy load.

The interfaces illustrated in Figure 15 are:

- **UI (Prosumer)** – This is an end point for the Prosumers so they can, along many other aspects of the system, provide user defined information on their energy flexibility. This information comes into play when meeting the FR 4 (Section 3.2.1), because the user input has priority over the estimated flexibility by this project's component – the *Flexibility Prediction* component.
- **UI (RECs Manager)** and the **UI (AI Engineer)** – Both these UIs are part of the broader *EnergAIze* system. They allow system REC managers to define what RECs the system will act on, otherwise needed data acquisition wouldn't be available for flexibility prediction. It also serves for the AI engineer to use past predictions of this project's component to retrain the algorithms for scheduling and optimizing energy flexibility.
- **External Data APIs** – This is an external interface through which the system can obtain necessary information for its tasks. This will be discussed in later sections when approaching the holiday filtering involved in data pre-processing.

3.4.2 Level 2

This section delves into the system, revealing its parts and its processes.

Scenario View

As mentioned in Section 3.3.3, this view emphasises on how a specific role interacts with the *EnergAIze* system. The functional requirements established in Section 3.2.1 define the following use cases:

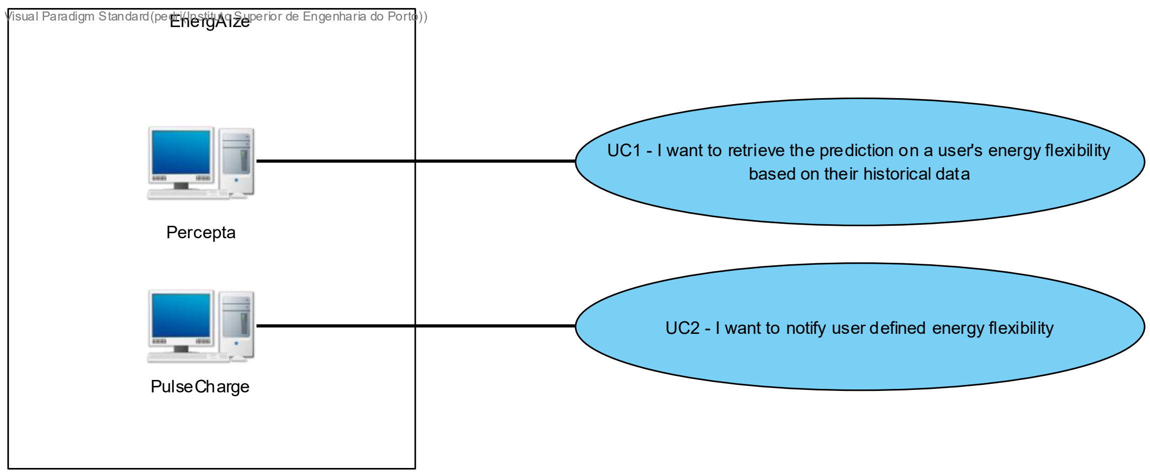

Visual Paradigm Standard(pedro(Instituto Superior de Engenharia do Porto))
EnergAIze
Percepta
PulseCharge
UC1 - I want to retrieve the prediction on a user's energy flexibility based on their historical data
UC2 - I want to notify user defined energy flexibility

Figure 16 – Use Cases for flexibility prediction

Use Case 1 refers mainly to FR 1 and FR 2. The predicting flexibility procedure depends on the access to user data and, to ensure the predicting models are accurate, on data pre-processing. Both these features are further detailed in Section 3.4.3. Historical data is constantly updated, and the solution should be able to adapt the models, as established by FR 5. Finally, all occurring actions should be logged (FR 3). The predicted flexibility is then stored in a database from where the *Percepta* component can claim it. The sequence diagram in Figure 21 explains the sequence of events of this Use Case.

Use Case 2 is simpler, because upon a user defined flexibility, it should store it (FR 4) and log whether it was successfully saved in the database or not (FR 3). The sequence diagram in Figure 22 explains the sequence of events of this Use Case.

Logical View

The Level 2 Logical view, illustrated in Figure 17, goes into the *EnergAIze* system that is represented in Figure 15. This higher level of detail reveals how this project, *Flexibility Prediction* component, interacts with the other containers.

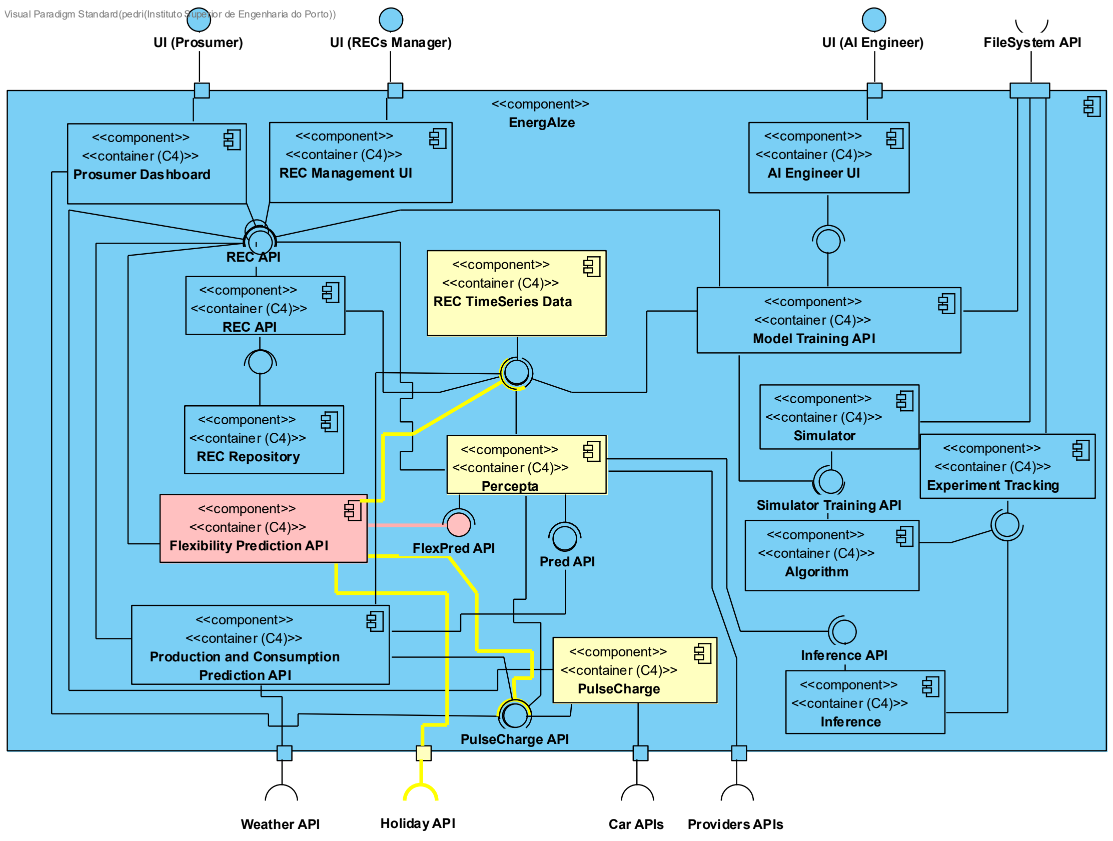

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
UI (Prosumer)
UI (RECs Manager)
UI (AI Engineer)
FileSystem API
<<component>> EnergAIze
<<component>> <<container (C4)>> Prosumer Dashboard
<<component>> <<container (C4)>> REC Management UI
<<component>> <<container (C4)>> AI Engineer UI
REC API
<<component>> <<container (C4)>> REC API
<<component>> <<container (C4)>> REC TimeSeries Data
<<component>> <<container (C4)>> Model Training API
<<component>> <<container (C4)>> REC Repository
<<component>> <<container (C4)>> Simulator
<<component>> <<container (C4)>> Percepta
<<component>> <<container (C4)>> Experiment Tracking
Simulator Training API
<<component>> <<container (C4)>> Flexibility Prediction API
FlexPred API
Pred API
<<component>> <<container (C4)>> Algorithm
<<component>> <<container (C4)>> Production and Consumption Prediction API
<<component>> <<container (C4)>> PulseCharge
Inference API
<<component>> <<container (C4)>> Inference
PulseCharge API
Weather API
Holiday API
Car APIs
Providers APIs

Figure 17 – Level 2 Logical view of the EnergAIze system (Flexibility Prediction component is highlighted in red and other relevant components/interfaces in yellow)

The Level 2 Logical view, in Figure 17, shows all the main components in the *EnergAIze* system. It also expands the *External APIs* interface in Figure 15, detailing each specific function. This project's contribution is the *Flexibility Prediction* component, whose tasks are dependent on other indirectly related components, like the *Algorithm* and the *Simulator* [14], and directly associated ones:

- **Flexibility Prediction API** – This is the main focus of this project, and it is the component responsible for energy flexibility deduction. It is responsible for providing user defined energy flexibility and, in its absence, to predict it based on user historical data through statistical inference.
- **REC Time Series Data** – This component provides an API that the *Flexibility Prediction* component can use to obtain the REC's users' historical data.
- **PulseCharge API** – This component is fundamental for meeting FR 4, established in Section 3.2.1, because it is responsible for notifying on user defined data.
- **Percepta** – The *Percepta* component acts as middleware, consuming and processing all the data circulating in the *EnergAIze* system, which includes the *Flexibility*

Component's output. The *EnergAIze*'s purpose is to manage flexibility within a REC resorting to Artificial Intelligence (AI) algorithms based on Reinforcement Learning (RL). The *Percepta* plays a role into this by providing the AI algorithm with a structured input that allows for a comprehensive view of the energy environment in the REC, resulting in a more sophisticated training phase for the AI model and better outcomes.

- **Holiday API** – This is an external Web API that is used for data pre-processing, which will be further discussed in higher level views (refer to Section 3.4.3).

Physical View

The Level 2 Physical view (Figure 18) shows how the physical deployment of the *EnergAIze* was conducted, and from where the *Flexibility Prediction* container operates.

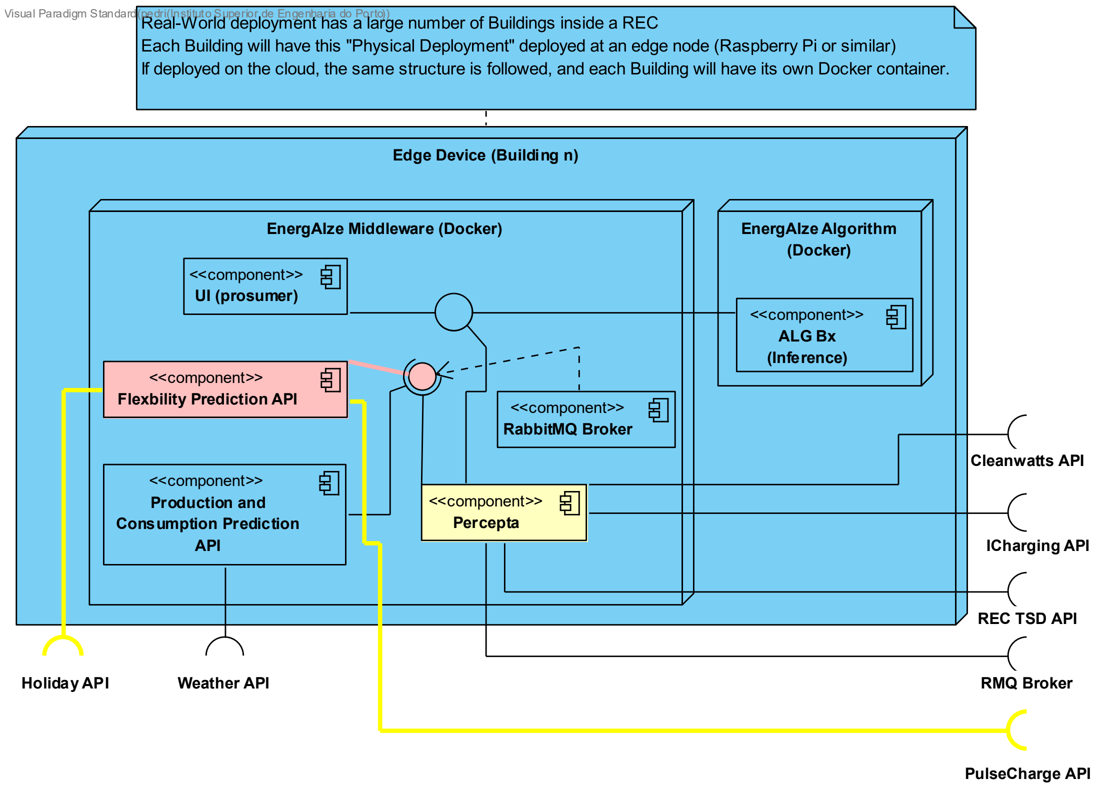

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
Real-World deployment has a large number of Buildings inside a REC
Each Building will have this "Physical Deployment" deployed at an edge node (Raspberry Pi or similar)
If deployed on the cloud, the same structure is followed, and each Building will have its own Docker container.
Edge Device (Building n)
EnergAIze Middleware (Docker)
EnergAIze Algorithm (Docker)
<<component>> UI (prosumer)
<<component>> ALG Bx (Inference)
<<component>> Flexbility Prediction API
<<component>> RabbitMQ Broker
<<component>> Production and Consumption Prediction API
<<component>> Percepta
Cleanwatts API
ICharging API
REC TSD API
RMQ Broker
Holiday API
Weather API
PulseCharge API

Figure 18 – Level 2 Physical view of the EnergAIze system (Flexibility Prediction component is highlighted in red)

Each component that runs on the EnergAIze middleware is deployed on a Docker container. Dockers implement a health check feature, that continuously checks its availability. If the container detects services failures, it is tagged as "not healthy" and restarts automatically. This allows for services to recover from failures, ensuring the system's reliability in compliance with NFR 7 (refer to Section 3.2.2).

RabbitMQ Broker paves the way to intercommunication between components, acting as a centralized entity that provides a scalable manner of communication. RabbitMQ is the implementation of AMQP protocol, a traditional centralized Publish Subscribe (Pub/Sub) paradigm [59]. Pub/Sub assigns two types of communicators: publishers that want to produce data, and subscribers that want to consume data. There is an intrinsic idea that publishers/subscribers are unconcerned about who is consuming/producing the data, resulting in high scalability. Thus, PulseCharge acts as a publisher, and redirect manual input from the user to Flexibility Prediction, that acts as a subscriber (refer to Figure 16).

These two Level 2 views, Figure 17 and Figure 18, are, as prescribed by the 4 + 1 model [57], connected and in Figure 19 shows the mapping between the Logical and Physical views.

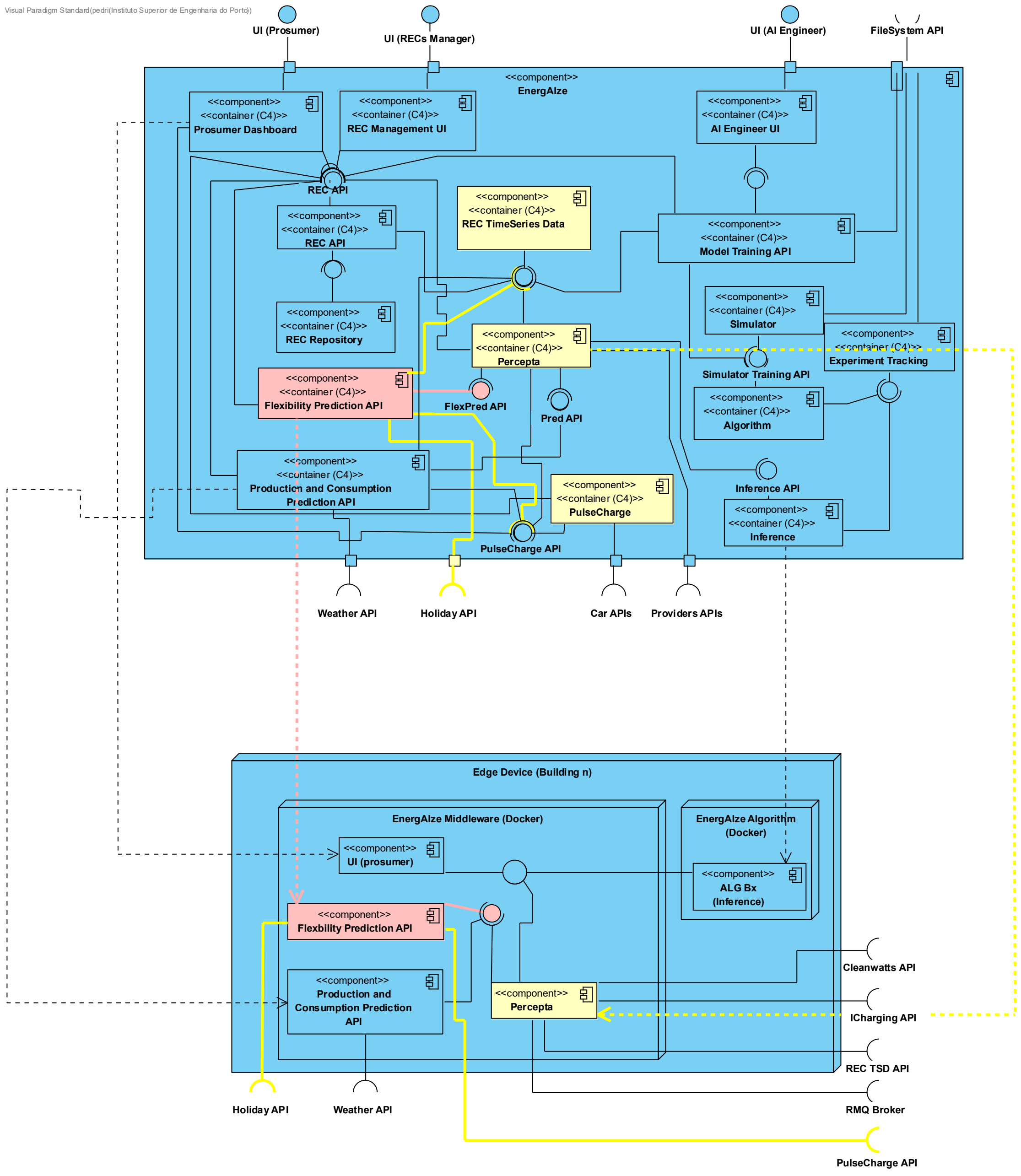

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
UI (Prosumer)
UI (RECs Manager)
UI (AI Engineer)
FileSystem API
<<component>> EnergAIze
<<component>> <<container (C4)>> Prosumer Dashboard
<<component>> <<container (C4)>> REC Management UI
<<component>> <<container (C4)>> AI Engineer UI
REC API
<<component>> <<container (C4)>> REC API
<<component>> <<container (C4)>> REC TimeSeries Data
<<component>> <<container (C4)>> Model Training API
<<component>> <<container (C4)>> REC Repository
<<component>> <<container (C4)>> Simulator
<<component>> <<container (C4)>> Percepta
<<component>> <<container (C4)>> Experiment Tracking
Simulator Training API
<<component>> <<container (C4)>> Flexibility Prediction API
FlexPred API
Pred API
<<component>> <<container (C4)>> Algorithm
<<component>> <<container (C4)>> Production and Consumption Prediction API
<<component>> <<container (C4)>> PulseCharge
Inference API
<<component>> <<container (C4)>> Inference
PulseCharge API
Weather API
Holiday API
Car APIs
Providers APIs
Edge Device (Building n)
EnergAIze Middleware (Docker)
EnergAIze Algorithm (Docker)
<<component>> UI (prosumer)
<<component>> ALG Bx (Inference)
<<component>> Flexbility Prediction API
<<component>> Production and Consumption Prediction API
<<component>> Percepta
Cleanwatts API
ICharging API
REC TSD API
RMQ Broker
Holiday API
Weather API
PulseCharge API

Figure 19 – Mapping between Level 2 Logical and Physical views. The Flexibility Prediction component is highlighted in red and other important components of the EnergAIze system in yellow.

3.4.3 Level 3

This section will approach the *Flexibility Prediction* container itself, along with its components and their associations, and with a deeper dive into the processes described in Section 3.4.2.

Logical View

The following section tackles the *Flexibility Prediction* component directly, revealing its parts and the interactions between them.

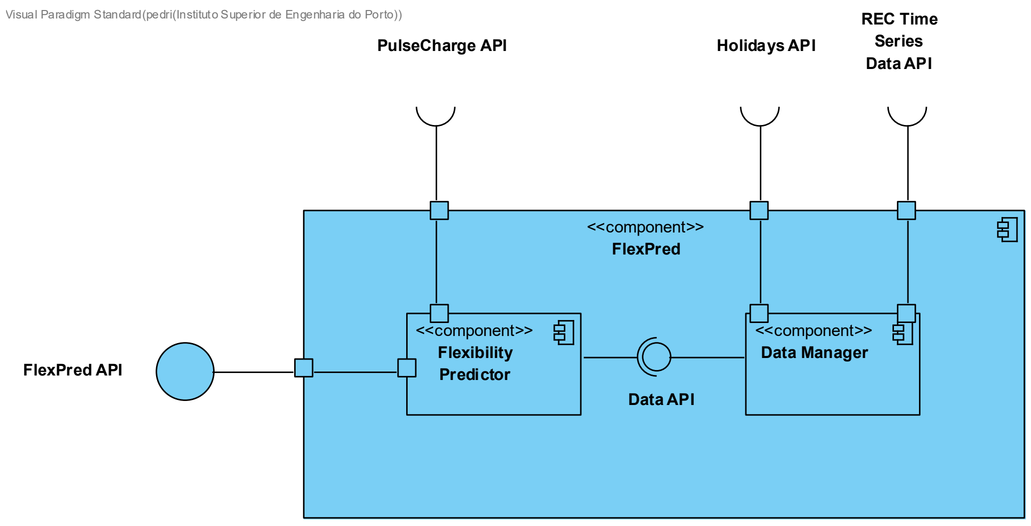

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
PulseCharge API
Holidays API
REC Time Series Data API
<<component>> FlexPred
<<component>> Flexibility Predictor
<<component>> Data Manager
FlexPred API
Data API

Figure 20 – Level 3 Logic View of Flexibility Prediction component

The components are each given a specific task, and they can be described as follows:

- **FlexPred API** – This is the interface provided by the *Flexibility Prediction* component that allows the system to make requests on the users' flexibility. It is through this interface that the *Percepta* component will obtain both the estimated flexibility based on historical data and user defined flexibility provided by the *PulseCharge* component.
- **Flexibility Predictor** – This component is responsible for applying the flexibility algorithms to the users' historical data. The data provided by the *Data Manager* is used to construct statistical models for inferring user flexibility.
- **Data Manager** – This component wraps data management for an easier data manipulation for the *Flexibility Predictor*. It encompasses data pre-processing and query functions that facilitate *Flexibility Predictor*'s accessibility to the users' data.
- **Data API** – This is the interface that the *Flexibility Predictor* uses to access the *Data Manager* functionalities.
- **Holidays API** – An external Web API for retrieving an updated list of the dates of holidays, for data pre-processing purposes. It plays a role in data pre-processing because it allows for dropping data that refer to holidays' patterns, which don't represent the users' usual energy usage behaviour.
- **PulseCharge API** – *EnergAIze* interface that communicates with the *Flexibility Prediction* component to notify on user-defined flexibility. Upon receiving a request

on the behalf of the *PulseCharge* component, the *Flexibility Prediction* component stores the user-defined flexibility, prioritising it over previous estimations.

Process View

The Level 3 Process View goes into more detail on the use cases for the system (Figure 16). It uses UML Sequence Diagrams to represent the interaction based on a time sequence. The following sections also give insight on different aspects taken into consideration when implementing the use case functionality.

UC1 – Retrieve users' predicted energy flexibility based on historical data

Table 1 – Use Case 1: Retrieve users' energy flexibility

Description	**The user's energy flexibility is requested**
Actor(s)	Percepta
Preconditions	1. The system must contain historical data on the energy charging profile of that user. 2. The system must have energy flexibility stored.
Postconditions	1. The user's flexibility is successfully retrieved from the system.
Necessary data	• User id; • Day of the week for which flexibility will be predicted.
Alternate flows	1. During data pre-processing, the holidays API may be unavailable, and a backup list is used, predicting a less optimal flexibility. 2. In case of insufficient data, the system logs the error. 3. In case of invalid user id or day of the week, the system logs the error.

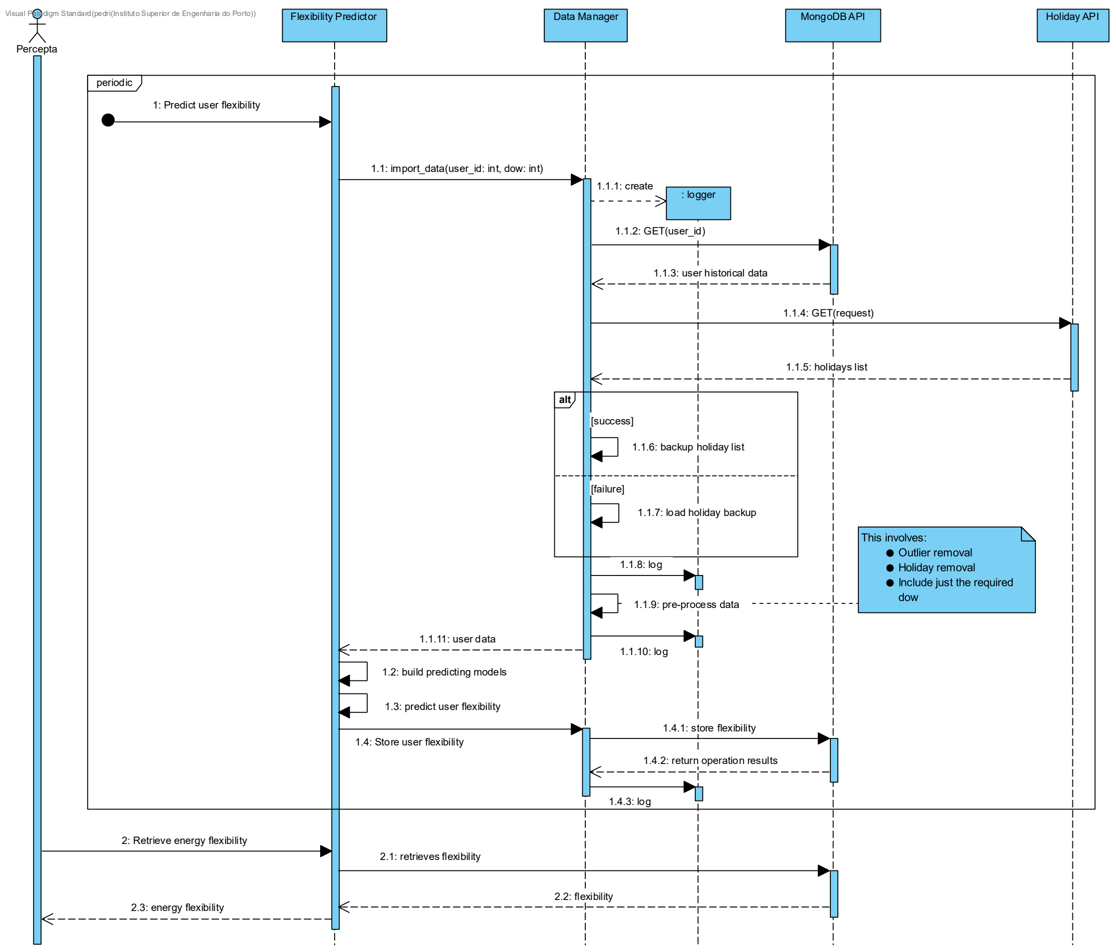

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
Percepta
Flexibility Predictor
Data Manager
MongoDB API
Holiday API
periodic
1: Predict user flexibility
1.1: import_data(user_id: int, dow: int)
1.1.1: create
: logger
1.1.2: GET(user_id)
1.1.3: user historical data
1.1.4: GET(request)
1.1.5: holidays list
alt
[success]
1.1.6: backup holiday list
[failure]
1.1.7: load holiday backup
This involves:
● Outlier removal
● Holiday removal
● Include just the required dow
1.1.8: log
1.1.9: pre-process data
1.1.10: log
1.1.11: user data
1.2: build predicting models
1.3: predict user flexibility
1.4: Store user flexibility
1.4.1: store flexibility
1.4.2: return operation results
1.4.3: log
2: Retrieve energy flexibility
2.1: retrieves flexibility
2.2: flexibility
2.3: energy flexibility

Figure 21 – Level 3 SDD for Use Case 1

Use Case 1 implementation requires algorithmic approaches for data pre-processing and statistical model generation. Statistical inference is highly sensitive to the data that is used to build the models it is based on. Because of this, the algorithm for data pre-processing, as seen in step 1.1.9 in Figure 21 (Algorithm 1), takes into consideration holidays, whose retrieval is done in step 1.1.5, particular user data, particular day of the week data and outliers to ensure the results are as reliable as possible. Then, in step 1.2, the predicting model is generated and used to predict user flexibility (step 1.3), accounting most frequent values in the sample and the desired confidence level.

The following section contains pseudocode representations of such algorithms and a description of how they complete their tasks.

```
Algorithm 1: filter_data

   Input:
      data - Users' historical data
      user_id - Pretended user
      dow - Day of the week: an integer value starting at 0 for Sunday
   Output:
      flex_object - A structure holding the user's flexibility for the specified day
 of the week

   1    Function filter_data(data, user_id, dow)
   2       wo_holidays <- remove_holidays(data)
   3       user_data <- get_user_data(wo_holidays, user_id)
   4       dow_only <- get_dow_data(user_data, dow)
   5       filtered_data <- remove_outliers(dow_only, th: def = 2.5)
   6       If length(filtered_data) < 4 then
   7          warning_log("Insufficient data")
   8          Return NULL
   9       End If
  10       Return filtered_data
  11    End
```

Algorithm 1 – Data filtering algorithm

Algorithm 1 is a simple data pre-processing procedure, and it refers to step 1.1.9 in Figure 21. It goes into different types of data filtering:

- → Line 2: Removes data referring to energy charging patterns for holidays. These do not represent normal user behaviour for their day-to-day activities, thereby it is removed.
- → Line 3: Selects just the data of the specified user, to avoid working with unnecessary and private data of other users.
- → Line 4: Selects the energy charging data for a specific day of the week, given how different days often impose different schedules.
- → Line 5: Removes data outliers. This is further discussed in Algorithm 2.

In the end, the algorithm checks for how much data exists, establishing a sample size minimum of 4, according to the KDE properties explained in Section 3.1.3. Upon insufficient data, the system logs a warning and returns an empty list.

Algorithm 2: remove_outliers

```
Input:
   data - Data with outliers
   th - Standard deviation threshold
Output:
   no_outliers - Data without outliers
```

```
 1  Function remove_outliers(data, th: def = 2.5)
 2     List abs_devs <- []
 3     For each value In data Do
 4        dev <- abs(value - median(data))
 5        abs_devs.append(dev)
 6     End for
 7     mdev <- median(abs_devs)
 8     List norm_devs <- []
 9     If mdev == 0 then
10        For i = 0 To length(abs_devs) Do
11           norm_devs.append(0)
12        End for
13     Else
14        For i = 0 To length(abs_devs) Do
15           norm_devs.append(abs_devs[i]/mdev)
16        End For
17     End If
18     List no_outliers <- []
19     For i = 0 To length(data) Do
20        If norm_devs[i] < th then
21           no_outliers.append(data[i])
22        End If
23     End For
24     Return no_outliers
25  End
```

Algorithm 2 – Outlier removal algorithm

Algorithm 2 dives further into the step 1.1.9 in Figure 21, focusing on the logic behind outlier removal. Outliers are observations that are very distant from the most common values in a sample [60]. Removing these outliers involve using sample statistical measures, such as mean or variance. However, using the median of a sample proves to be a more robust solution, given how little it is affected by both outliers and data dispersion [61]. This is more advantageous because outliers won't affect their own detection, but it imposes a limitation for the same reason: even if there are outliers in a sample, if they are very few when compared to the rest of the values, they might go undetected.

The procedure proposed by [61] was adapted in Algorithm 2, and its steps are as follows:

→ Line 3 to 6: A For each loop is used to calculate how each value in the sample deviate from the median, called Absolute Deviations (ADs). This is done by calculating the absolutes differences between the sample median and each sample value, storing each one in the *abs_dev* list.

→ Line 7: The median of the *abs_dev* list is taken, resulting in the Median Absolute Deviation (MAD). This metric is used as a reference to evaluate how much sample values deviate from what is commonly seen in the dataset.

→ Line 9 to 17: The ADs are normalized, i.e., each is divided by MAD and the ratio is used to evaluate if they are close or distant. Lower values imply that a specific AD doesn't fall very far from other values in the sample and vice-versa. Line 9 outlines a very specific case where MAD is 0 and, to prevent division by 0, the normalisation results in a list with all values at 0 because most data values don't deviate much from the median.

→ Line 19 to 23: Each normalized AD is confronted with a threshold (*th*) for considering the corresponding data value an outlier. The higher the threshold, the more permissive to outliers and vice-versa. A good general rule is establishing the threshold as 2.5 [61].

This outlier detection and removal method (Algorithm 2) is favourable for this project's purpose because it is ideal for cases where the population distribution is unknown and/or the sample size is small [61].

```
Algorithm 3: predict_user_flexibility

    Input:
      filtered_data - Pre-processed data
    Output:
      fo - Flexibility object

 1   Function predict_user_flexibility(filtered_data)
 2      If length(filtered_data) <= 0 then
 3         Return NULL
 4      End If
 5      fo <- new FlexObject()
 6      For each flex_attr In filtered_data Do
 7         kde_distr <- KDE(flex_attr)
 8         (peaks, density, grid) <- find_kde_peaks(kde_distr)
 9         all_intervals <- find_peak_intervals(peaks, density, grid)
10         bt_intervals <- better_intervals(all_intervals)
11         For each interval In bt_intervals Do
12            fo.add_flex_attr(type=flex_attr, interval)
13         End For
14      End For
15      Return fo
16   End
```

Algorithm 3 – Algorithm for predicting energy flexibility

Algorithm 3 is specific of step 1.3 of the sequence diagram in Figure 21, being responsible for predicting the energy flexibility. It starts by checking if there is data regarding energy use for the user (Line 2 to 4) and, if not, the process stops due to lack of information for flexibility estimation (based on historical data).

At Line 6 a every type of flexibility attribute present in the *filtered_data* is iterated over to determine their corresponding flexibility:

→ Line 7: Builds the KDE model around the historical data for the currently selected flexibility attribute.

→ Line 8: Finds the peaks of the built KDE distribution. As seen in Section 3.1.3, a KDE PDF can result in more than one peak (Figure 11). These outline the most probable events, and it is around them that confidence intervals can be built with a reasonable amount of supporting data. This is further discussed in Algorithm 4.

→ Line 9: All possible confidence intervals are calculated based on the KDE PDF peaks. This is further discussed in Algorithm 5.

→ Line 10: From all the intervals, the better ones are selected. This is further discussed in Algorithm 6.

→ Lines 11 to 13: The better intervals are stored.

Algorithm 4: find_kde_peaks

```
Input:
   kde - KDE distribution
   x_range - Range of values in kde
Output:
   peaks - The indices of the KDE peaks
   density - The estimated probability density values for each value in x_grid
   x_grid - Evenly spaced x values in the KDE x range
```

```
 1   Function find_kde_peaks(kde, x_range)
 2      num_points <- 1000
 3      x_grid <- segment_xrange(x_range, num_points)
 4      density <- find_likelihood(x_grid, kde)
 5      peak_indices <- find_peaks(density)
 6      List peaks <- []
 7      For each index In peak_indices Do
 8         peaks.append(x_grid[index])
 9      End For
10      Return (peaks, density, x_grid)
11   End
```

Algorithm 4 – Algorithm for finding the peaks of a KDE distribution

Algorithm 4 explains the step 1.3 in Figure 21 in further detail, focusing on segmenting the *kde* x range into multiple evenly distanced points, calculating their probability density and finding out all maximum values of the PDF. For better understanding, the steps taken in this algorithm are as follows:

- → Line 2: A default value for the number of points that will be evaluated. Higher values impose more accurate analysis of the PDF, but it ends up more computationally demanding.
- → Line 3: This creates a list with evenly spaced values for the distribution's x range. This comes into play in Line 4 and even in Algorithm 5.
- → Line 4: For each value in *x_grid*, the value of *f(x)* will be calculated and stored in the *density* list, where *f* is the PDF function described by the KDE method. This is an important step because later, when estimating confidence intervals, the resulting densities are input for integrating methods that will estimate the area under the PDF (refer to Section 3.1.1).
- → Line 5 to 9: All *x_grid* values that correspond to maximum values of the PDF, are stored in the *peaks* list.

This algorithm is an intermediate step because it defines the peaks of the PDF defined by the KDE, which are the starting points for confidence intervals estimation.

Algorithm 5: find_peak_intervals

Input:

peaks - KDE distribution peaks indices

density - Estimated probability density values for each value in x_values

x_vals - Evenly spaced x values in the KDE x range

Output:

peak_intervals - Confidence intervals built around KDE peaks

```
 1  Function find_peak_intervals(peaks, density, x_vals)
 2     confidence = 0.95
 3     List intervals <- []
 4     total_area <- simpson_integrate(density, x_vals)
 5     For each peak In peaks Do
 6        peak_idx <- index of x_vals closest to peak
 7        left_idx <- peak_idx
 8        right_idx <- peak_idx
 9        area <- 0.0
10        While area < confidence × total_area Do
11           left_idx <- max(0, left_idx - 1)
12           right_idx <- min(length(x_vals) - 1, right_idx + 1)
13           interval_x <- x_vals from left_idx to right_idx (inclusive)
14           interval_y <- density from left_idx to right_idx (inclusive)
15           area <- simpson_integrate(interval_y, interval_x)
16           If left_idx == 0 And right_idx == length(x_vals) - 1 Then
17              Break
18           End If
19        End While
20        lower_bound <- x_vals[left_idx]
21        upper_bound <- x_vals[right_idx]
22        intervals.append(new Interval(lower_bound, upper_bound))
23     End For
24     Return intervals
25  End
```

Algorithm 5 – Algorithm for finding all the confidence intervals around the peaks of the KDE distribution

Algorithm 5 still takes place in the step 2.3 procedure of Figure 21, taking the Algorithm 4 output as input to estimate the confidence intervals. In Line 2 it establishes the level of confidence and in Line 4 the total area described by the *density* values is calculated using the **Simpson's rule**. In theory, this total area beneath the PDF function should equal 1 (refer Section 3.1.1), however the *density* list describes discrete x values (for computational purposes) instead of the whole x range defined by the KDE, resulting in slight variations on the total area.

The Simpson's rule is a way of approximating integrals, with reasonable accuracy and less computation [62]. For applying this rule, it is necessary to divide the x range of the function into several evenly spaced values. Then, those values, along with their corresponding images, are used to estimate the area beneath the function. Both the segmented x range and their images are output of Algorithm 4, which serve as input for Algorithm 5.

The confidence interval will be calculated around each peak in the PDF. Line 5 starts by iterating over each peak found by Algorithm 4 and performs the following steps:

- → For the set of evenly spaced x values, it finds the closest to the currently selected peak (Line 6).
- → It defines the starting area as 0. The objective is to establish the lower and upper limits for which this area equals the level of confidence chosen.
- → In Line 10 begins loop with the following steps:
    - o The loop ends when the area between the two limits is more or equal than the level of confidence.
    - o Find new lower and upper limits based on the previous ones and the segmented x values (*x_vals*). Since they are ordered it can be done by iterating on the previous lower and upper limits forwards and backwards by one (Lines 11 and 12), respectively (ensuring the new ones aren't out of the *x_vals* bounds).
    - o In Line 13 to 15, the attributes for applying the Simpson's rule are calculated, and then the new area is found and stored. Depending on the newly found area, the loop continues or stops, adding the interval to the return list.

The resulting confidence intervals need to be evaluated to determine which represent better the energy flexibility, which is the role of Algorithm 6.

```
Algorithm 6: better_intervals
  Input:
     all_intervals - All possible intervals for flexibility
  Output:
     better_intervals - The better intervals from all the intervals

 1   Function better_intervals(all_intervals)
 2      List better_intervals <- []
 3      For each interval In all_intervals Do
 4         If length(better_intervals) == 0 then
 5            better_intervals.append(interval)
 6            Continue
 7         End If
 8         For i = 0 To length(better_intervals) Do
 9            If matching_intervals(better_intervals[i], interval) then
10               If interval.amplitude < better_intervals[i].amplitude then
11                  better_intervals[i] = interval
12               End If
13            Else
14               better_intervals.append(interval)
15            End If
16         End For
17      End For
18      Return better_intervals
19   End
```

Algorithm 6 – Algorithm for selecting the better confidence intervals

Algorithm 6 is the final part of the step 1.3 in Figure 21 and it focuses on selecting the better confidence intervals of all that were calculated by Algorithm 5. Line 2 instantiates the returning list that will contain the better intervals found. Then, in Line 3, each calculated interval that was calculated by Algorithm 5 is iterated over, to select the better ones. The loop is structured as follows:

→ Line 4 to 7: If statement responsible for initiating the returning list. Simply put, if the returning list is empty, then the currently selected interval is automatically the better one, so it is appended, and the loop moves to the next iteration.

→ Line 8 to 16: This for loop will confront the currently selected interval with every interval that was previously deemed as better, replacing it according to the following criteria:

- If the currently selected interval encompasses different values than any of the better intervals, it is appended to the *better_intervals* list (Line 14).
- Otherwise, the currently selected interval replaces the better interval if, and only if, the first's amplitude is smaller than the latter's.

The logic for choosing one interval in favour of another is as follows: for two different intervals, built under the same level of confidence and comprising similar values, the one with a smaller amplitude imposes more data supporting that the real value lies within that range. This results in more accurate confidence intervals and more reliable flexibility estimations.

UC2 – Notify user defined flexibility

Table 2 – Use Case 2: Notify user defined energy flexibility

Description	**Notification on user defined energy flexibility.**
Actor(s)	PulseCharge
Preconditions	User inputs their personalized energy flexibility.
Postconditions	User defined flexibility is stored in the system
Necessary data	Flexibility on: • Start time for the charging session; • Charging session duration; • Total energy charged.
Alternate flows	The system informs of an error upon communication failure with the database.

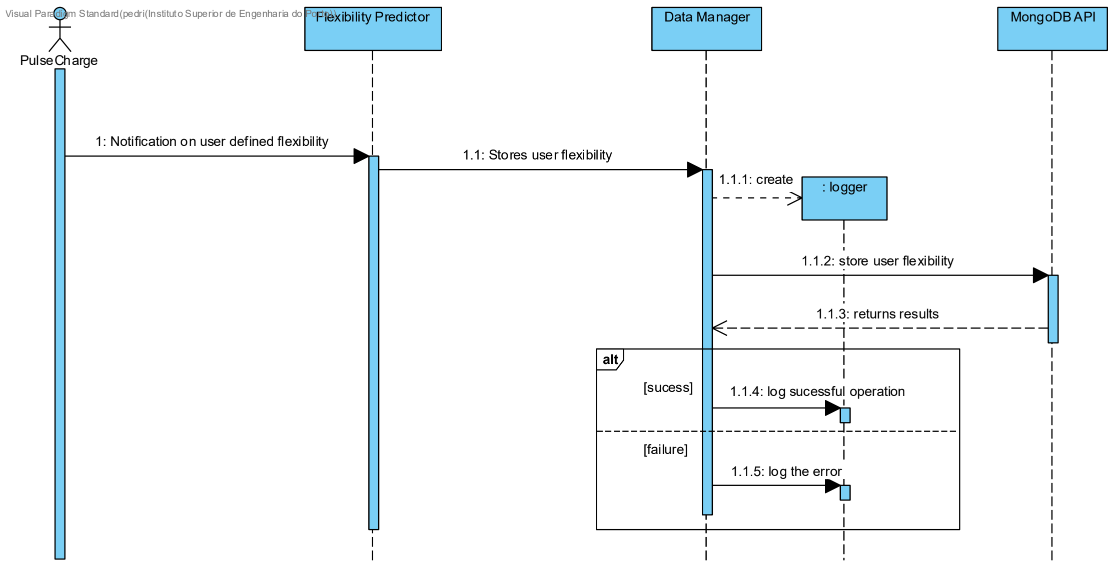

Visual Paradigm Standard(pedri(Instituto Superior de Engenharia do Porto))
PulseCharge
Flexibility Predictor
Data Manager
MongoDB API
1: Notification on user defined flexibility
1.1: Stores user flexibility
1.1.1: create
: logger
1.1.2: store user flexibility
1.1.3: returns results
alt
[sucess]
1.1.4: log sucessful operation
[failure]
1.1.5: log the error

Figure 22 – Level 3 SDD for Use Case 2.

The procedure in Figure 22 is simpler than the flexibility prediction (Figure 21). Upon receiving the energy preferences from the user (step 1), through a notification made by the *PulseCharge* component, the system communicates with the database to persist them (step 1.1.2), logging the results in steps 1.1.4 and 1.1.5.

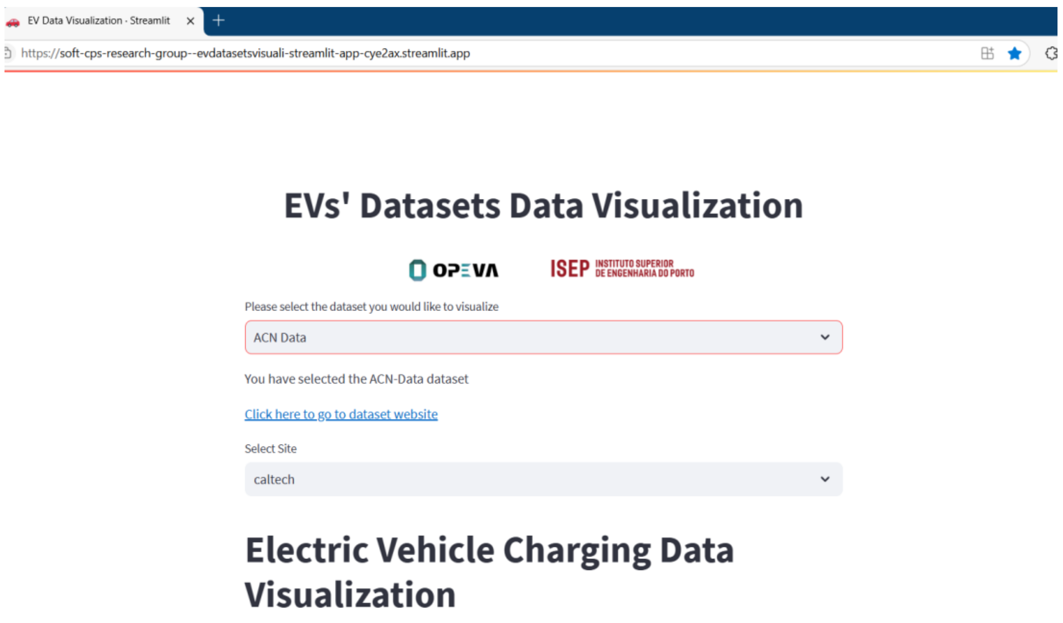
4 Implementation

This section focuses on describing the approach taken during the implementation of the solution. Utilised resources, such as external APIs or libraries, are addressed and then the implementation of the previous algorithm showcased, according to the problem's context. Then all tests used to evaluate the solution are presented and explained, ending the chapter with an overall conclusion on the final version of the developed solution.

4.1 Description

This solution was developed using Visual Code Studio, given its versatility and good support for the Python programming language. Furthermore, GitHub was used as a version control tool, providing a way to track the changes done to the project over time.

Besides algorithm implementation, there are other non-core implementations, such as wrapper structures and utilities module. Further detail on these functionalities can be found from Appendix A to Appendix F. In addition, the data analysis phase (refer to Section 1.2.3) involved the development of a web application, using the Streamlit framework, that allows for quick reports on various aspects regarding user energy use. This website comprises various open-source datasets and is currently hosted online thanks to the Streamlit cloud service. Figure 23 shows a preview of the developed website. Further detail can be found in Appendix K.


EV Data Visualization · Streamlit

https://soft-cps-research-group--evdatasetsvisuali-streamlit-app-cye2ax.streamlit.app

EVs' Datasets Data Visualization

Please select the dataset you would like to visualize

ACN Data

You have selected the ACN-Data dataset

Click here to go to dataset website

Select Site

caltech

Electric Vehicle Charging Data Visualization

Figure 23 – Web application for open-source data analysis screenshot.

In order to facilitate user understanding and usage of the application, the project's repository includes a README file that overviews the project's structure and scripts for running and testing the program. Appendix G and Appendix H present snapshots of said file.

For maintainability purposes, each functionality of this solution is well documented using Python's docstrings, which were then used to generate API documentation (Appendix I) for easy understanding of the project's ins and outs.

This solution makes use of external libraries. Data management was dependent on the NumPy and the Pandas frameworks, while statistics was Scikit-Learn's and SciPy's role. Further information on all dependencies and corresponding versions used are found in Appendix J.

4.2 Component Implementation

As stated during the design phase (refer to Section 3.3), the solution is divided into two main containers: *Data Manager*, responsible for data requesting and processing, and the *Flexibility Predictor*, responsible for building the statistical models and using it for statistical inference.

The following sub-sections focus on the implementation of the algorithms in Section 3.4.3, explaining how they were adapted to the problem's context, using code snippets.

4.2.1 Data Manager

Code Snippet 1 shows how Algorithm 1 was adapted to the project's context. The *Data Manager* component includes the *Data Navigator* (DN) and the *Filterer* functionalities. The *Filterer* works in tandem with the DN to process the data. While the DN's responsibility is to retrieve specific data only, such as specific user or day of the week tuples, the *Filterer* focuses on the outlier and holiday tuples removal.

```python
import pandas as pd
import numpy as np
import data_manager.data_navigator as dn
import data_manager.holidays as hd
from utils.logger import Logger
from utils import time_conv as tc

LOG = Logger("logs/filterer.log")

def filter_data(data: pd.DataFrame, user_id: int, dow: int):

    no_holidays = remove_holidays(data)
    user_data = dn.get_user_data(no_holidays, user_id)
    spec_dow_data = dn.get_spec_dow_data(user_data, dow)
    clean_data = dn.convert_date_to_min(spec_dow_data)
    final_data = remove_outliers(clean_data)

    if len(final_data) < 4:
        LOG.WARN(f"Insuficcient data to estimate flexibility for user [{user_id}] on [{tc.int_to_dow(dow)}]")
        return []

    return final_data
```

Code Snippet 1 – Data pre-processing

The Algorithm 2 was adapted to perform outlier removal for 3 different columns: start, duration and energy. In the original data, these correspond to time of arrival, charging duration and charged energy, respectively. Code Snippet 2 shows the implementation of outlier removal which in this case instead of directly dropping those values it is done using a boolean mask (*combined_mask* in line 5). This mask is changed iteratively to pinpoint which tuples have outlier values (lines 12 and 13), omitting them in the final return statement, i.e., if a tuple has an outlier value, then the whole tuple is removed.

```python
import pandas as pd
import numpy as np

def remove_outliers(df: pd.DataFrame, columns=["start",
"duration", "energy"], out_threshold=2.5):
    combined_mask = pd.Series(True, index=df.index)

    for col in columns:
        data = df[col].values
        d = np.abs(data - np.median(data))
        mdev = np.median(d)
        s = d / mdev if mdev else np.zeros(len(d))
        mask = s < out_threshold
        combined_mask &= mask

    return df[combined_mask]
```

Code Snippet 2 – Outlier removal

4.2.2 Flexibility Generator

The *Flexibility Prediction* function, represented by Code Snippet 3, makes use of the DN to retrieve the flexibility attributes data and, for each column, applies the procedure shown in Algorithm 3.

```
 1  from utils import visualization as v
 2  import pandas as pd
 3  from structs.fo import FlexObject
 4
 5  def predict_flexibility(data: pd.DataFrame, debug=False):
 6
 7      if len(data) <= 0:
 8          return None
 9
10      fo = FlexObject()
11      flex_cols = dn.get_flex_columns(data)
12
13      for e in flex_cols:
14          kde_s = kde_model(e)
15          peaks, den, grid = find_kde_peaks(kde_s)
16          all_intervals = find_peak_intervals(peaks, den, grid)
17          bt_intervals = better_intervals(all_intervals)
18
19          if debug:
20              v.plot_kde_density(kde_s, e.name)
21          for i in bt_intervals:
22              fo.add_flexibility(e.name, i)
23
24      return fo
```

Code Snippet 3 – Flexibility prediction

One important note is the use of the *Visualisation* functionality (lines 19 and 20). This can be activated through the *debug* parameter to obtain a visual representation of the resulting statistical model (Figure 24). Further detail can be found in Appendix D.

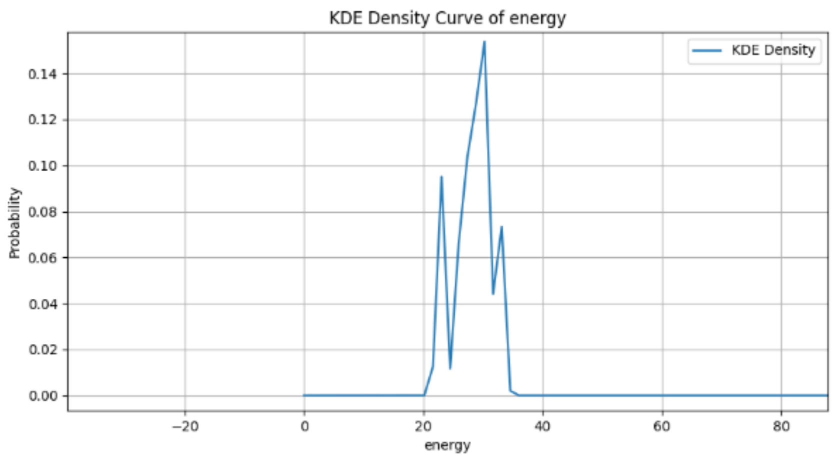


Figure 24 – Debug mode example for charged energy

Code Snippet 4 is a straightforward implementation of Algorithm 4. The difference is in adapting the data to the parameter format of functions of different libraries. Namely, the functions used for calculating the probability density (lines 7 and 8) and where they peak (line 9) use a 2D array. However, the returning values were all converted into one-dimensional arrays to facilitate data handling between programmed functions.

```
1    import numpy as np
2    from sklearn.neighbors import KernelDensity
3    from scipy.signal import find_peaks
4
5    def find_kde_peaks(kde: KernelDensity, x_range=(0, 1440),
     num_points=1000):
6        x_grid = np.linspace(x_range[0], x_range[1],
     num_points).reshape(-1, 1)
7        log_dens = kde.score_samples(x_grid)
8        density = np.exp(log_dens)
9        peak_indices, _ = find_peaks(density)
10       peak_positions = x_grid[peak_indices].flatten()
11       return peak_positions, density, x_grid.flatten()
```

Code Snippet 4 – Find KDE distribution function peaks

Code Snippet 5 is responsible for finding all confidence intervals for the flexibility attributes. Interval functions (Appendix F) and the Simpson function provided by the SciPy library (lines 7 and 20) allowed for a direct adaptation of Algorithm 5.

```
1   import numpy as np
2   from scipy.integrate import simpson
3   from structs.interval import Interval
4
5   def find_peak_intervals(peaks, density, x_vals,
    confidence=0.95):
6       intervals = []
7       total_area = simpson(density, x_vals)
8
9       for peak in peaks:
10          peak_idx = np.argmin(np.abs(x_vals - peak))
11          left_idx = peak_idx
12          right_idx = peak_idx
13          area = 0.0
14
15          while area < confidence * total_area:
16              left_idx = max(0, left_idx - 1)
17              right_idx = min(len(x_vals) - 1, right_idx + 1)
18              interval_x = x_vals[left_idx:right_idx + 1]
19              interval_y = density[left_idx:right_idx + 1]
20              area = simpson(interval_y, interval_x)
21
22              if left_idx == 0 and right_idx == len(x_vals) - 1:
23                  break
24
25          lower_bound = x_vals[left_idx]
26          upper_bound = x_vals[right_idx]
27          intervals.append(Interval(lower_bound, upper_bound))
28
29      return intervals
```

Code Snippet 5 – Find KDE peak confidence intervals

Code Snippet 6 makes use of interval functionalities (*interval* in line 13) found in Appendix F. This allows for a direct translation of Algorithm 6 into code.

```
1   from structs.interval import Interval, matching_intervals
2
3   def better_intervals(all_intervals: list[Interval]):
4       better_intervals = []
5
6       for i in all_intervals:
7
8           if not better_intervals:
9               better_intervals.append(i)
10              continue
11
12          for itr in range(len(better_intervals)):
13              if matching_intervals(better_intervals[itr], i):
14                  if i.amplitude <
    better_intervals[itr].amplitude:
15                      better_intervals[itr] = i
16              else:
17                  better_intervals.append(i)
18
19      return better_intervals
```

Code Snippet 6 – Find better confidence intervals

4.3 Tests

This project is based different functions and their interactions. The algorithmic nature of the solution imposes the need for unit testing to ensure each component is performing their tasks correctly.

Unit testing is a fundamental practice of software development. It focuses on testing the smallest units of functionality in the code, ensuring they are performing their tasks correctly, according to the programmer’s established logic. In the event of a bug or runtime error, unit tests provide an easy way of isolating the source of the problem, helping to fix it quickly [63].

Some unit testing covering involve using the following methods:

- **Logic Validations** – Checks if the system executes the right procedures for an expected input by evaluating its corresponding expected output.
- **Limit Validations** – Checks how the system responds to the input it is given, in the occasion of unexpected and/or expected values.
- **Error Treatment** – Checks how the system responds when the input has errors: if it proceeds, by implementing fail-safe methods, or completely shuts down.

Unit testing was performed for the simpler parts, such as utilities and interval logic, and for the more algorithm heavy parts, e.g. data pre-processing.

The developed unit tests developed did not cover any statistic inference related functionalities. This is due to the probabilistic nature of applying such models, which conflicts with the deterministic way in which the testing process assures the code is working as expected.

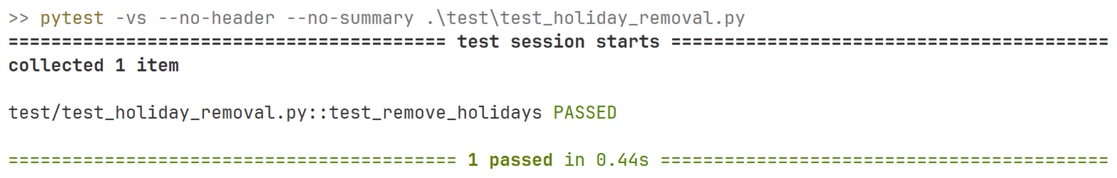
The unit tests were covered using Pytest, a simple Python framework for writing tests [64], and the following sub-sections show them in finer detail.

4.3.1 Data pre-processing

This sub-section details the unit tests that ensure that data pre-processing related functionalities are performing their tasks successfully.

Holiday Removal

This unit testing focuses on evaluating the performance of the holiday removal functionality. A list containing dates served as an input to the procedure, many of which were holidays. Finally, the result is compared to what resulting list it is expected. Figure 25 shows a successful execution of this test.

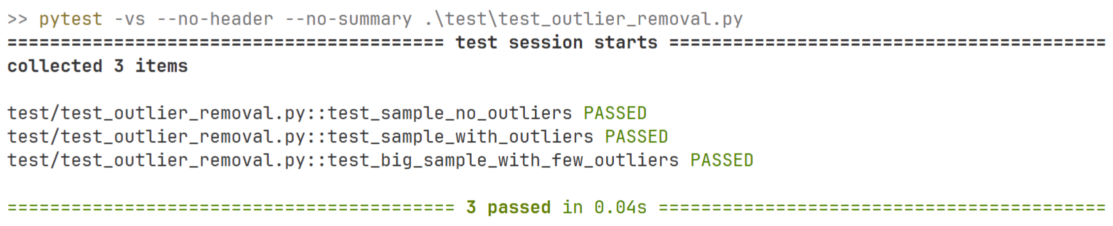


```
>> pytest -vs --no-header --no-summary .\test\test_holiday_removal.py
======================================== test session starts =========================================
collected 1 item

test/test_holiday_removal.py::test_remove_holidays PASSED

========================================= 1 passed in 0.44s ==========================================
```

Figure 25 – Holiday removal test execution

Outlier Removal

This unit testing focuses on the outlier detection and removal logic. This is one fundamental step of data pre-processing and ensuring that it performs as expected under different circumstances is important attaining the best possible results. For example, the outlier removal should leave data without outliers intact, otherwise important information could be lost. Also, data sample size can influence outlier detection. Both these scenarios were covered, and Figure 26 shows a successful execution for these tests.

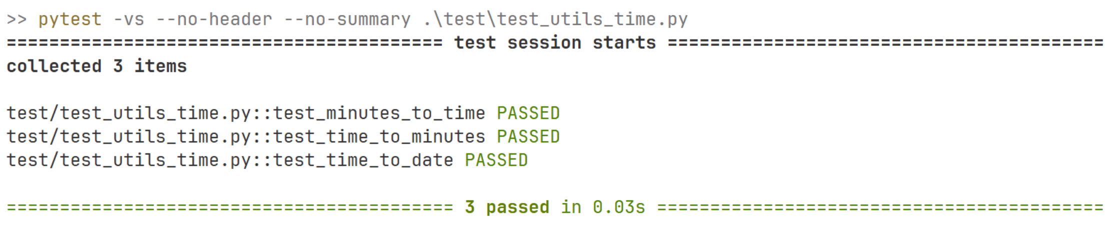


```
>> pytest -vs --no-header --no-summary .\test\test_outlier_removal.py
========================================= test session starts =========================================
collected 3 items

test/test_outlier_removal.py::test_sample_no_outliers PASSED
test/test_outlier_removal.py::test_sample_with_outliers PASSED
test/test_outlier_removal.py::test_big_sample_with_few_outliers PASSED

========================================== 3 passed in 0.04s ==========================================
```

Figure 26 – Outlier removal tests execution

4.3.2 Time conversions

This unit testing focuses on ensuring that the logic behind time conversions (Code Snippet 9), needed for transforming the original data to a more suitable format for statistical models, is working properly. For example, converting time in the format HH:mm to integers was tested to 1) ensure that the input is of the expected type and 2) the returning value corresponds correctly to what it is expected (for example 10:00 is equivalent to 600 minutes). Figure 27 shows the successful execution of these tests.

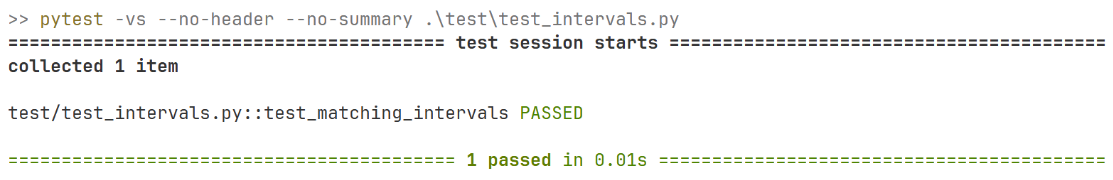


```
>> pytest -vs --no-header --no-summary .\test\test_utils_time.py
========================================= test session starts =========================================
collected 3 items

test/test_utils_time.py::test_minutes_to_time PASSED
test/test_utils_time.py::test_time_to_minutes PASSED
test/test_utils_time.py::test_time_to_date PASSED

========================================== 3 passed in 0.03s ==========================================
```

Figure 27 – Time utilities tests execution

4.3.3 Interval Logic

This unit testing focuses on evaluating if the *matching_intervals* algorithm, shown in Code Snippet 12, responsible for determining if two intervals encompass common values, is performing its task according to different possible scenarios. Figure 28 shows the successful execution of these tests.



```
>> pytest -vs --no-header --no-summary .\test\test_intervals.py
========================================= test session starts =========================================
collected 1 item

test/test_intervals.py::test_matching_intervals PASSED

========================================== 1 passed in 0.01s ==========================================
```

Figure 28 – Interval logic tests execution

4.4 Solution Evaluation

Starting off with Use Case 2, the design solution was projected alongside the other team members, in order to ensure compliance with the overall *EnergAIze* system. Overall, the *Flexibility Prediction* component design was a success because it proved little intrusive to the already existing components, requiring just the establishment of a new communication interface to work. However, the shared platform for uploading the flexibility related data is still being discussed in consortium with the members of the *EnergAIze* project, thereby preventing its persistence.

Regarding Use Case 1 implementation, the flexibility prediction results proved to be satisfactory considering the data that was available. The evaluation of the developed solution followed the approach bellow:

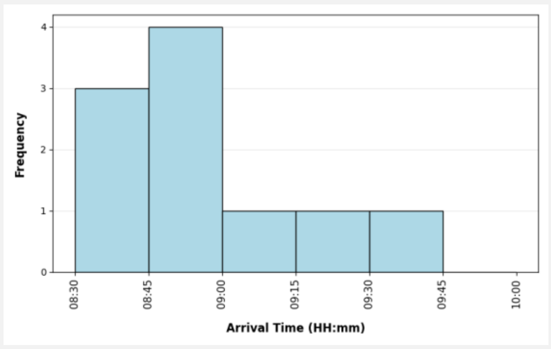
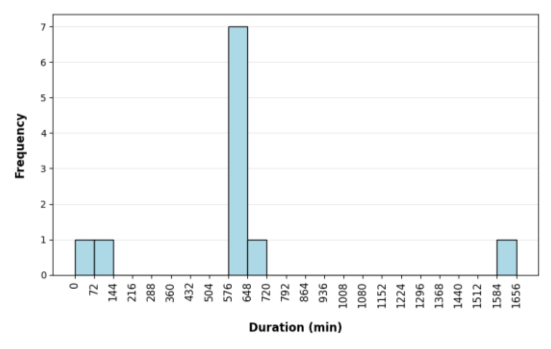
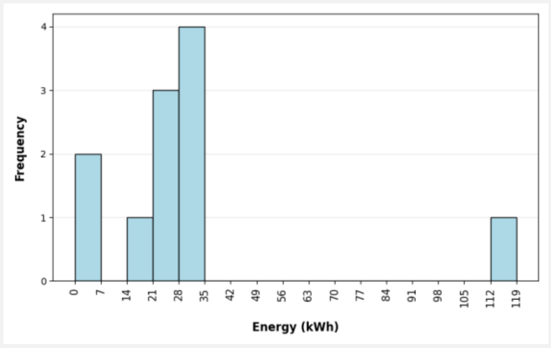
1) **Data visualisation of the sample** – Using a histogram representation, for example.
2) **Flexibility prediction** – The *Flexibility Prediction* component uses the sample to generate the statistical models and creates flexibility objects.
3) **Result Evaluation** – The returned flexibility is confronted with the visualisation.

For this, 3 different user samples were used: User 1 history on Tuesdays, User 2 history on Wednesday and User 3 history on Thursday.

4.4.1 User 1

Table 3 shows the energy use patterns for User 1 alongside the flexibility predicted by the *Flexibility Prediction* component.

Table 3 – Comparison between sample data and predicted flexibility for User 1

Flexibility Attribute	***Sample Visualisation***	***Predicted Flexibility***
Arrival Time		**] 08:40; 09:06 [,** at 95% confidence
Duration (min)		**] 605.41; 657.3 [,** at 95% confidence
Energy (kWh)		**] 21.62; 38.92 [,** at 95% confidence

A quick analysis on Table 3 reveals that:

I. **Regarding arrival time**, the resulting confidence interval encompasses the most frequent values in the sample, revealing it as being appropriate and proving the statistical model efficacy.

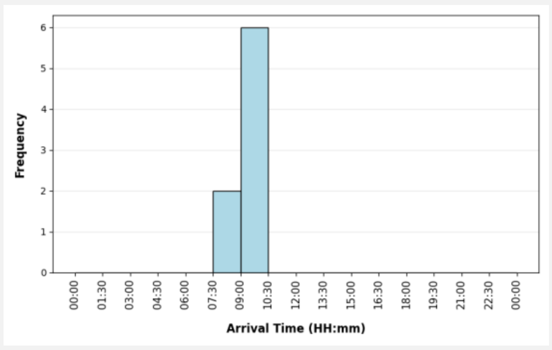
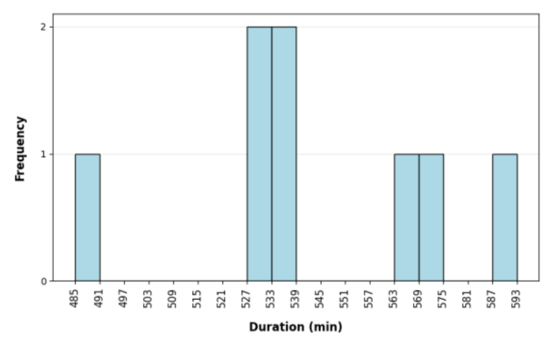
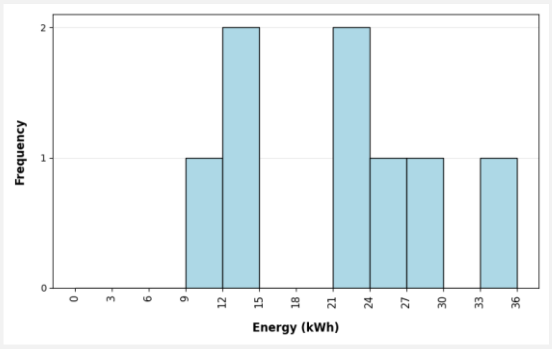
II. **Regarding Duration**, there is a pretty noticeable preference for the user 1 to charge between 576 and 648. This greatly impacted the statistical model, making the resulting confidence interval adapt to this user's patterns successfully.

III. **Regarding Energy**, the model was capable of successfully identifying the most frequent values for charged energy, which reflects on a reasonable confidence interval.

4.4.2 User 2

Table 4 shows the energy use patterns for User 2 alongside the flexibility predicted by the *Flexibility Prediction* component.

Table 4 – Comparison between sample data and predicted flexibility for User 2

Flexibility Attribute	***Sample Visualisation***	***Predicted Flexibility***
Arrival Time		**] 09:36; 10:02 [,** at 95% confidence
Duration (min)		**] 510.27; 565.05 [,** at 95% confidence
Energy (kWh)		**] 5.77; 37.48 [,** at 95% confidence

A quick analysis on Table 4 reveals that:

I. **Regarding arrival time**, the statistical model provided a reasonable confidence interval that reflects the most frequent arrival times for user 2.

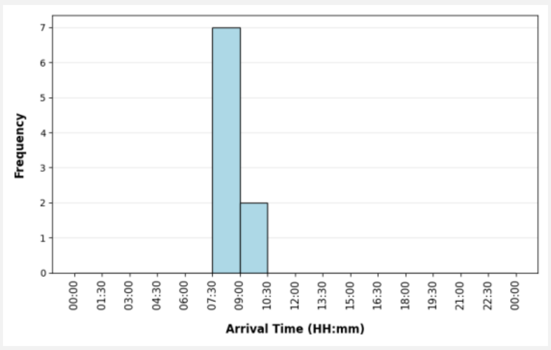
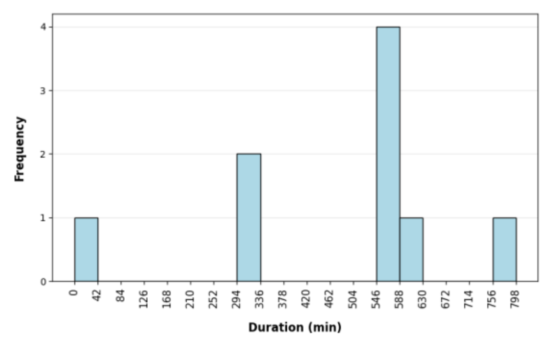
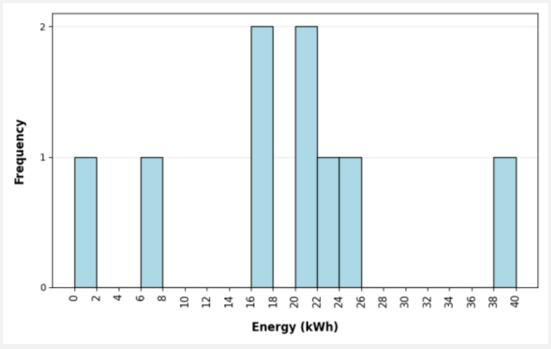
II. **Regarding Duration**, even though the data was sparse, the model returned a confidence interval that encompasses the most frequent values for the duration while also accounting for possibly lower values.

III. **Regarding Energy**, the data was sparse but nonetheless the procedures involved in the flexibility prediction treated all provided data as important, resulting in a larger confidence interval.

4.4.3 User 3

Table 5 shows the energy use patterns for User 3 alongside the flexibility predicted by the *Flexibility Prediction* component.

Table 5 – Comparison between sample data and predicted flexibility for User 3

Flexibility Attribute	***Sample Visualisation***	***Predicted Flexibility***
Arrival Time		**] 07:28; 08:02 [,** at 95% confidence
Duration (min)		**] 537.66; 627.03 [,** at 95% confidence
Energy (kWh)		**] 14.41; 28.83 [,** at 95% confidence

A quick analysis on Table 5 reveals that:

I. **Regarding arrival time**, the model was capable of easily detecting user 3 patterns, resulting in a reasonable confidence interval.

II. **Regarding Duration**, although the data was sparser, the statistical model was capable of identifying the higher number of cases around the 550 to 630 minutes marks of the sample and presenting a satisfactory confidence interval.

III. **Regarding Energy**, the data was, once again, sparse but the model focused on the values that were more frequent within the sample, building a good confidence interval around them.

5 Conclusion

This chapter is a summary of the development process of this project. It presents the results obtained and how they matched the project’s objectives. Then, the limitations and future improvements are explained, ending on an overall final appreciation of the solution.

5.1 Accomplishments

This project achieved important accomplishments, contributing in energy flexibility in a non user intrusive manner, an often overlooked aspect.

The main objective of this project was successfully attained by developing a system capable of predicting energy flexibility using statistical inference, mainly through the adaptation of KDE to problem's context. It allows for a seamless integration into other frameworks, such as the *EnergAIze* system, by allowing both user defined and statistically inferred energy flexibility. Table 6 shows a list of the objectives and their degree of accomplishment.

Table 6 – Project’s objectives and respective degrees of accomplishment

Objective	Degree of Accomplishment
Objective 1	Complete
Objective 2	Complete
Objective 3	Complete

In spite of all design related objectives being completely achieved, there were some limitations to the implementation phase itself. Flexibility prediction was accomplished but, due to impasses during development (refer to Section 5.2) and lack of time for meeting the deadlines, user defined flexibility wasn’t implemented. Nonetheless, the design choices that support its future implementation comply very well with the entire system, making for a swift upcoming integration.

5.2 Limitations and Future Work

Although the project did manage to achieve its main goal (flexibility prediction), there are some aspects that could be improved and complementary features that should be added in the future:

- As it was stated, the focus of this project was EV charging patterns, given how flexible they are and, thereby, influential when it comes to grid load balancing. However, in the future this solution could be adapted to other elastic type devices (refer to Section 2.1).
- This project was developed using collaborator provided data. However, the APIs for data retrieval were fairly recent, making available data scarce. Because of this, after evaluating the options for flexibility estimation, the focus of this project was to build statistical models using the KDE approach, given its reasonable results even with smaller samples (refer to Section 3.1.3). As more data is available, adding new statistical models that are more appropriate for larger samples is a priority for future implementation. Furthermore, the *Flexibility Component* should be able to dynamically opt between different models given the sample it is given. This is because new users could adhere to the electric grid, making the KDE a good approach in the beginning and later on, as more and more data is available, a different model could be used.
- Currently, the system does not store the flexibility objects in a database. This is because the platform for data uploading is still being discussed in consortium with the members of the *EnergAIze* project. This makes it so Use Case 2 is currently not implemented; however it is fully designed, and successfully so. It is a high priority to ensure the persistence of the flexibility objects in future implementations.
- Finally, the energy charging data had gaps due to hardware and software problems on the provider’s behalf. This caused a loss of precision on the final results for flexibility prediction. However, this problem has already been reported, and it is projected to not affect the solution’s efficiency in the future.

5.3 Final Appreciation

Considering this solution's engineering process and how accurate statistical inference was able to accomplish this project's main objective, of predicting energy flexibility with minimal user intrusion, the *Flexibility Component* can be considered a success.

Despite achieving its main objective, there are still some further improvements that should be carried out to ensure that it can be deployed in a real case scenario and still perform with reasonable results, as stated in Section 5.2.

From a more personal point of view, working on this research project was a great learning opportunity. It provided a deeper understanding on the importance of design and implementation concepts during development of more complex systems. Researching, discussing and adapting new concepts to the problem's context was a valuable experience that helped develop both technical and social skills.

Bibliography

[1] T. Fonseca, L. L. Ferreira, J. Landeck, L. Klein, P. Sousa, and F. Ahmed, "Flexible Loads Scheduling Algorithms for Renewable Energy Communities," *Energies (Basel)*, vol. 15, no. 23, Dec. 2022, doi: 10.3390/en15238875.

[2] M. Boehm *et al.*, "Data management in the MIRABEL smart grid system," in *ACM International Conference Proceeding Series*, 2012, pp. 95–102. doi: 10.1145/2320765.2320797.

[3] F. Lilliu, T. B. Pedersen, and L. Siksnys, "Heat FlexOffers: a device-independent and scalable representation of electricity-heat flexibility.," in *e-Energy 2023 - Proceedings of the 2023 14th ACM International Conference on Future Energy Systems*, Association for Computing Machinery, Inc, Jun. 2023, pp. 374–385. doi: 10.1145/3575813.3597347.

[4] "Confronting the Duck Curve: How to Address Over-Generation of Solar Energy | Department of Energy." Accessed: May 20, 2025. [Online]. Available: https://www.energy.gov/eere/articles/confronting-duck-curve-how-address-over-generation-solar-energy

[5] J. Ortiz-Ulloa and J. Alejandro Ortiz Ulloa, "The Role of the Industrial Engineer in an Energy System Development El Papel del Ingeniero Industrial en el Desarrollo de un Sistema Energético," 2017. [Online]. Available: https://www.researchgate.net/publication/323202976

[6] "Managing Excess Renewable Energy Generation in Businesses." Accessed: May 21, 2025. [Online]. Available: https://www.linkedin.com/pulse/managing-excess-renewable-energy-generation-businesses-ftmue

[7] F. Scheller, R. Burkhardt, R. Schwarzeit, R. McKenna, and T. Bruckner, "Competition between simultaneous demand-side flexibility options: the case of community electricity storage systems," *Appl Energy*, vol. 269, Jul. 2020, doi: 10.1016/j.apenergy.2020.114969.

[8] A. Mugnini, G. Coccia, F. Polonara, and A. Arteconi, "Energy flexibility as additional energy source in multi-energy systems with district cooling," *Energies (Basel)*, vol. 14, no. 2, Jan. 2021, doi: 10.3390/en14020519.

[9] I. Harris, C. L. Mumford, and M. M. Naim, “A hybrid multi-objective approach to capacitated facility location with flexible store allocation for green logistics modeling,” *Transp Res E Logist Transp Rev*, vol. 66, pp. 1–22, 2014, doi: 10.1016/j.tre.2014.01.010.

[10] “SoftCPS Laboratory – SoftCPS.” Accessed: May 22, 2025. [Online]. Available: https://www2.isep.ipp.pt/softcps/?page_id=577

[11] “OPEVA – OPtimization of Electric Vehicle Autonomy.” Accessed: May 22, 2025. [Online]. Available: https://opeva.eu/#

[12] “Partners – OPEVA.” Accessed: May 22, 2025. [Online]. Available: https://opeva.eu/partners/

[13] “EV Data Visualization · Streamlit.” Accessed: May 21, 2025. [Online]. Available: https://soft-cps-research-group--evdatasetsvisuali-streamlit-app-cye2ax.streamlit.app/

[14] T. Fonseca, L. Ferreira, B. Cabral, R. Severino, and I. Praça, “EnergAIze: Multi Agent Deep Deterministic Policy Gradient for Vehicle-to-Grid Energy Management,” 2024.

[15] T. Fonseca *et al.*, “Control of Renewable Energy Communities using AI and Real-World Data,” May 2025, Accessed: Jun. 03, 2025. [Online]. Available: https://arxiv.org/pdf/2505.17321

[16] “home - i-charging.” Accessed: May 22, 2025. [Online]. Available: https://i-charging.tech/

[17] “About us - Cleanwatts.” Accessed: May 22, 2025. [Online]. Available: https://cleanwatts.energy/about-us/

[18] S. Sharma, D. Sarkar, and D. Gupta, “Agile Processes and Methodologies: A Conceptual Study.”

[19] Iea, “Electricity 2025,” 2025. [Online]. Available: www.iea.org

[20] T. Nath, “The Economics of Solar Power.” Accessed: May 20, 2025. [Online]. Available: https://www.investopedia.com/articles/investing/061115/economics-solar-power.asp

[21] T. B. Pedersen, L. Siksnys, and B. Neupane, "Modeling and Managing Energy Flexibility Using FlexOffers," in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm 2018*, Institute of Electrical and Electronics Engineers Inc., Dec. 2018. doi: 10.1109/SmartGridComm.2018.8587605.

[22] "An introduction to the Universal Smart Energy Framework", Accessed: May 28, 2025. [Online]. Available: https://energy.ec.europa.eu/system/files/2014-11/xpert_group3_summary_0.pdf

[23] "The Project - GOFLEX." Accessed: May 22, 2025. [Online]. Available: https://goflex-project.eu/Project.html

[24] B. Neupane *et al.*, "GOFLEX: Extracting, Aggregating and Trading Flexibility based on FlexOffers for 500+ Prosumers in 3 European cities [Operational Systems Paper]," vol. 22, doi: 10.1145/3538637.3538865.

[25] Z. J. Lee, T. Li, and S. H. Low, "ACN-Data: Analysis and applications of an open EV charging dataset," in *e-Energy 2019 - Proceedings of the 10th ACM International Conference on Future Energy Systems*, Association for Computing Machinery, Inc, Jun. 2019, pp. 139–149. doi: 10.1145/3307772.3328313.

[26] "Random Variable: Definition, Types, How It's Used, and Example." Accessed: May 21, 2025. [Online]. Available: https://www.investopedia.com/terms/r/random-variable.asp

[27] "Statistics - Probability, Random Variables, Distributions | Britannica." Accessed: May 21, 2025. [Online]. Available: https://www.britannica.com/science/statistics/Probability

[28] "Numeracy, Maths and Statistics - Academic Skills Kit." Accessed: May 21, 2025. [Online]. Available: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/distribution-functions/probability-density-function.html

[29] "Probability Density Function | GeeksforGeeks." Accessed: May 21, 2025. [Online]. Available: https://www.geeksforgeeks.org/probability-density-function/

[30] "Calculus I - Computing Definite Integrals." Accessed: May 21, 2025. [Online]. Available:

https://tutorial.math.lamar.edu/Classes/CalcI/ComputingDefiniteIntegrals.aspx

[31] “Representative Sample: Definition, Importance, and Examples.” Accessed: May 23, 2025. [Online]. Available: https://www.investopedia.com/terms/r/representative-sample.asp

[32] “Normal Distribution | Examples, Formulas, & Uses.” Accessed: May 28, 2025. [Online]. Available: https://www.scribbr.com/statistics/normal-distribution/

[33] B. B. Frey, “Normal Distribution,” in *The SAGE Encyclopedia of Educational Research, Measurement, and Evaluation*, SAGE Publications, Inc., 2018. doi: 10.4135/9781506326139.n476.

[34] “Normal Distribution - MathBitsNotebook(A2).” Accessed: May 27, 2025. [Online]. Available: https://mathbitsnotebook.com/Algebra2/Statistics/STnormalDistribution.html

[35] D. R. Hodge and D. F. Gillespie, “Phrase Completion Scales,” *Encyclopedia of Social Measurement, Three-Volume Set*, vol. 3, pp. V3-53-V3-62, Jan. 2004, doi: 10.1016/B0-12-369398-5/00124-9.

[36] “Central Limit Theorem.” Accessed: May 23, 2025. [Online]. Available: https://www.probabilitycourse.com/chapter7/7_1_2_central_limit_theorem.php

[37] “Normal distribution.” Accessed: May 28, 2025. [Online]. Available: https://www3.nd.edu/~rwilliam/stats1/x21.pdf

[38] “Confidence Intervals.” Accessed: May 23, 2025. [Online]. Available: http://www.stat.yale.edu/Courses/1997-98/101/confint.htm

[39] Y. Cao, R. C. Chen, and A. J. Katz, “Why is a small sample size not enough?,” *Oncologist*, vol. 29, pp. 761–763, 2024, doi: 10.1093/oncolo/oyae162.

[40] “Kernel Density Estimation : Data Science Concepts - YouTube.” Accessed: May 23, 2025. [Online]. Available: https://www.youtube.com/watch?v=t1PEhjyzxLA&ab_channel=ritvikmath

[41] S. Węglarczyk, “Kernel density estimation and its application,” 2018, doi: 10.1051/itmconf/20182300037.

[42] Bernard. W. Silverman, *Density Estimation for Statistics and Data Analysis*. 1986.

[43] “Kernel density estimation - Wikipedia.” Accessed: May 23, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Kernel_density_estimation

[44] “The importance of kernel density estimation bandwidth.” Accessed: May 30, 2025. [Online]. Available: https://aakinshin.net/posts/kde-bw/

[45] “Requirements Engineering Process in Software Engineering | GeeksforGeeks.” Accessed: May 27, 2025. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/

[46] “A Guide to Functional Requirements (with Examples).” Accessed: May 27, 2025. [Online]. Available: https://www.nuclino.com/articles/functional-requirements

[47] “Non-functional Requirements: What They Do, Examples, and Best Practices | Perforce Software.” Accessed: May 27, 2025. [Online]. Available: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples#what-are-non-functional-requirements

[48] “Architecture 101: Top 10 Non-Functional Requirements (NFRs) you Should be Aware of | by Anji… | Medium.” Accessed: May 27, 2025. [Online]. Available: https://anjireddy-kata.medium.com/architecture-101-top-10-non-functional-requirements-nfrs-you-should-be-aware-of-c6e874bd57e0

[49] “Non-functional requirements. Previous… | by Kostiantyn Ivanov | Medium.” Accessed: Jun. 06, 2025. [Online]. Available: https://medium.com/@svosh2/non-functional-requirements-ce6e55305900

[50] “Nonfunctional Requirements: Examples, Types and Approaches.” Accessed: May 27, 2025. [Online]. Available: https://www.altexsoft.com/blog/non-functional-requirements/

[51] “Understanding Non-Functional Requirements: Insights & Examples.” Accessed: Jun. 13, 2025. [Online]. Available: https://qat.com/guide-understanding-non-functional-requirements/

[52] "Why Software Design Is Important." Accessed: May 28, 2025. [Online]. Available: https://www.computer.org/resources/importance-of-software-design-is-important

[53] "Introduction | C4 model." Accessed: May 28, 2025. [Online]. Available: https://c4model.com/introduction

[54] "What is C4 Model? Complete Guide for Software Architecture." Accessed: May 28, 2025. [Online]. Available: https://miro.com/diagramming/c4-model-for-software-architecture/

[55] "4+1 Architectural view model in Software | by Pasan Devin Jayawardene | Javarevisited | Medium." Accessed: May 28, 2025. [Online]. Available: https://medium.com/javarevisited/4-1-architectural-view-model-in-software-ec407bf27258

[56] "4 + 1 Views in Modeling System Architecture with UML - Visual Paradigm Guides." Accessed: May 28, 2025. [Online]. Available: https://guides.visual-paradigm.com/4-1-views-in-modeling-system-architecture-with-uml/

[57] P. Kruchten, "Architectural Blueprints-The '4+1' View Model of Software Architecture," *IEEE Softw*, vol. 12, no. 6, pp. 42–50, 1995.

[58] "Modelling and Documenting the Software Architecture | by Vijay Moorthy (https://vijaytechpings.blog) | Medium." Accessed: May 28, 2025. [Online]. Available: https://medium.com/@vijmoorthy/modelling-and-documenting-the-software-architecture-4a1a46ab0cd0

[59] "AMQP 0-9-1 Model Explained | RabbitMQ." Accessed: Jun. 14, 2025. [Online]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts

[60] "7.1.6. What are outliers in the data?" Accessed: May 31, 2025. [Online]. Available: https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm

[61] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *J Exp Soc Psychol*, vol. 49, no. 4, pp. 764–766, Jul. 2013, doi: 10.1016/J.JESP.2013.03.013.

[62] "Simpson's Rule and Integration", Accessed: May 31, 2025. [Online]. Available: https://web.stanford.edu/group/sisl/k12/optimization/MO-unit4-pdfs/4.2simpsonintegrals.pdf

[63] “O que são Testes de unidade? — Explicação sobre Testes de unidade — AWS.” Accessed: Jun. 13, 2025. [Online]. Available: https://aws.amazon.com/what-is/unit-testing/

[64] “pytest documentation.” Accessed: Jun. 06, 2025. [Online]. Available: https://docs.pytest.org/en/stable/

Appendix A   KDE Model Generation

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KernelDensity

def kde_model(data: pd.Series, bw="scott"):

    kde = KernelDensity(kernel="gaussian", bandwidth=bw)
    start_data = data.copy()
    kde.fit(np.array(start_data).reshape(-1, 1))
    return kde
```

Code Snippet 7 – KDE model generation (flex.py)

Appendix B   Logger Class

```
from datetime import datetime

class Logger():
    def __init__(self, path):
        self.log_path = path
        self.ERROR = "ERROR"
        self.WARNING = "WARNING"
        self.OK = "OK"

    def log(self, message: str, message_type: str):
        with open(self.log_path, 'a') as log:
            log.write(f"[{datetime.now().strftime("%Y-%m-%d
%H:%M")}]: {message_type.upper()} - {message}\n")

    def ERR(self, message:str):
        self.log(message, self.ERROR)

    def WARN(self, message:str):
        self.log(message, self.WARNING)

    def OK(self, message:str):
        self.log(message, self.OK)
```

Code Snippet 8 – Logger class (logger.py)

Appendix C Time Conversion Utilities

```python
from datetime import datetime
DOW_DICT = {
    0: "Sunday",
    1: "Monday",
    2: "Tuesday",
    3: "Wednesday",
    4: "Thursday",
    5: "Friday",
    6: "Saturday"
}

def minutes_to_time(min: float):
    min = int(min)
    return f"{min // 60:02d}:{min % 60:02d}"

def time_to_minutes(datetime_str: str):
    time_part = datetime_str.split()[1]
    hours, minutes = map(int, time_part.split(':'))
    return hours * 60 + minutes

def time_to_date(datetime_str: str, format="%Y-%m-%d"):
    date_part = datetime_str.split()[0]
    date_obj = datetime.strptime(date_part, format)
    return date_obj.strftime("%d/%m/%Y")

def int_to_dow(int: int):
    return DOW_DICT.get(int)
```

Code Snippet 9 – Time conversion utilities (time_conv.py)

Appendix D KDE Model Visualisation

```
import numpy as np
import matplotlib.pyplot as plt

def plot_kde_density(kde, name, x_range=(0, 1440), label='KDE
Density'):

    x = np.linspace(x_range[0], x_range[1], 1000).reshape(-1, 1)

    log_density = kde.score_samples(x)
    density = np.exp(log_density)

    plt.figure(figsize=(10, 5))
    plt.plot(x, density, label=label)
    plt.title(f"KDE Density Curve of {name}")
    plt.xlabel(name)
    plt.ylabel("Probability")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Code Snippet 10 – Debug visualisation module (visualization.py)

Appendix E Flexibility Wrapper Class

```
from structs.interval import Interval

class FlexObject:
    def __init__(self):
        self.start = []
        self.duration = []
        self.energy = []

    def add_flexibility(self, flex_type: str, flex_attr:
Interval):
        if flex_type == "start":
            self.start.append(flex_attr)
        elif flex_type == "duration":
             self.duration.append(flex_attr)
        else:
             self.energy.append(flex_attr)

    def __str__(self):
        start_lines = "\n\t".join(i.time_intervals__str__() for i
in self.start)
        duration_lines = "\n\t".join(str(i) for i in
self.duration)
        energy_lines = "\n\t".join(str(i) for i in self.energy)

        return (
            f"Start time flexibility:\n\t{start_lines}\n"
            f"Duration flexibility:\n\t{duration_lines}\n"
            f"Energy flexibility:\n\t{energy_lines}"
        )
```

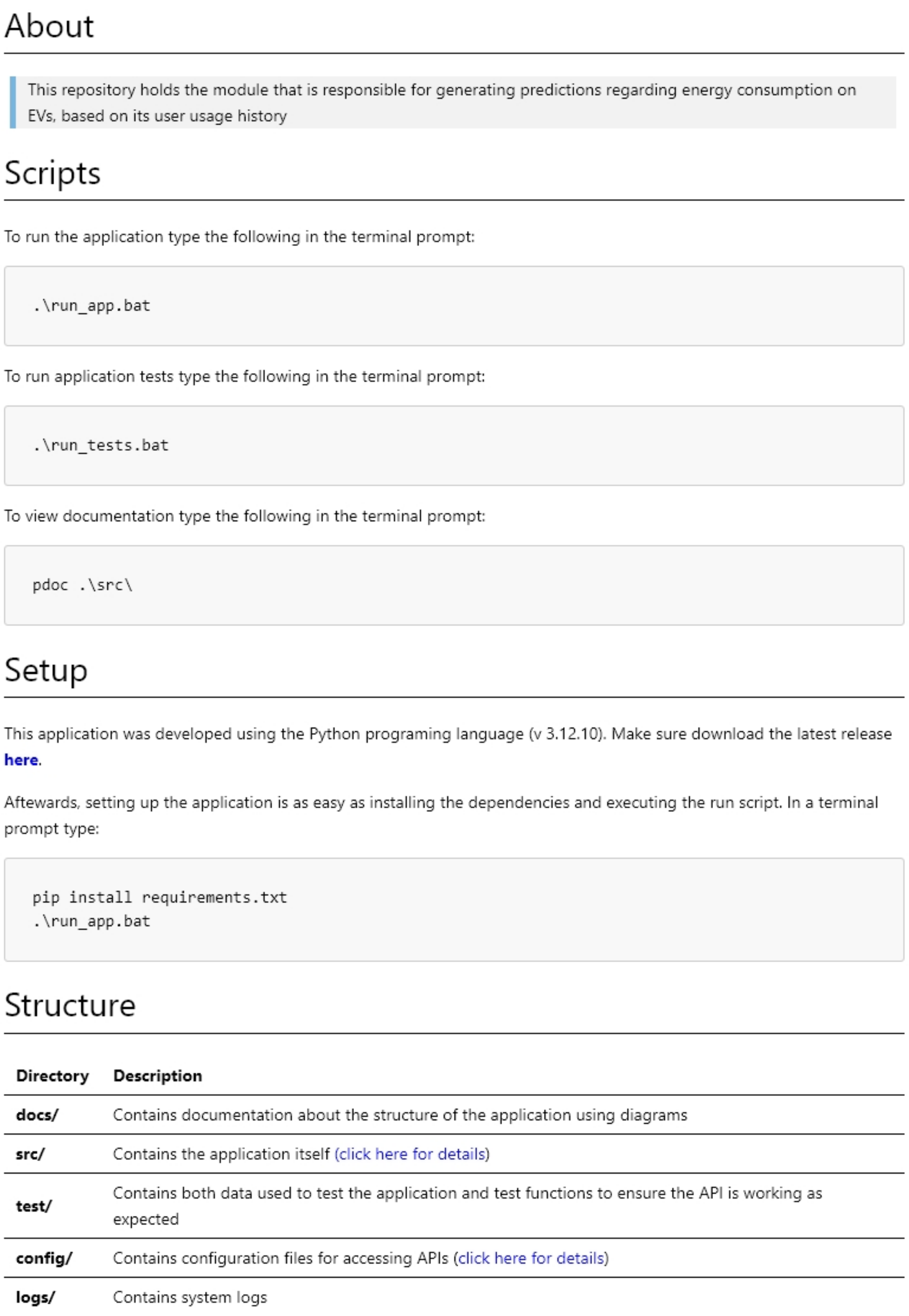
Code Snippet 11 – Flexibility class (fo.py)

Appendix F Interval Functionalities

```
from utils import time_conv as tc

class Interval:
    def __init__(self, lower: float, upper: float):
        self.lower = round(lower, 2)
        self.upper = round(upper, 2)
        self.amplitude = self.upper - self.lower
    def __str__(self):
        return f"]{round(self.lower, 2)}; {round(self.upper, 2)}["
    def time_intervals__str__(self):
        return f"]{tc.minutes_to_time(self.lower)};
{tc.minutes_to_time(self.upper)}["

def matching_intervals(int1: Interval, int2: Interval):

    if int1.lower <= int2.lower:
        return int1.upper >= int2.lower
    elif int2.lower <= int1.lower:
        return int2.upper >= int1.lower

    return False
```

Code Snippet 12 – Interval functionalities (interval.py)

Appendix G Guide README (page 1)



About

> This repository holds the module that is responsible for generating predictions regarding energy consumption on EVs, based on its user usage history

Scripts

To run the application type the following in the terminal prompt:

```
.\run_app.bat
```

To run application tests type the following in the terminal prompt:

```
.\run_tests.bat
```

To view documentation type the following in the terminal prompt:

```
pdoc .\src\
```

Setup

This application was developed using the Python programing language (v 3.12.10). Make sure download the latest release **here**.

Aftewards, setting up the application is as easy as installing the dependencies and executing the run script. In a terminal prompt type:

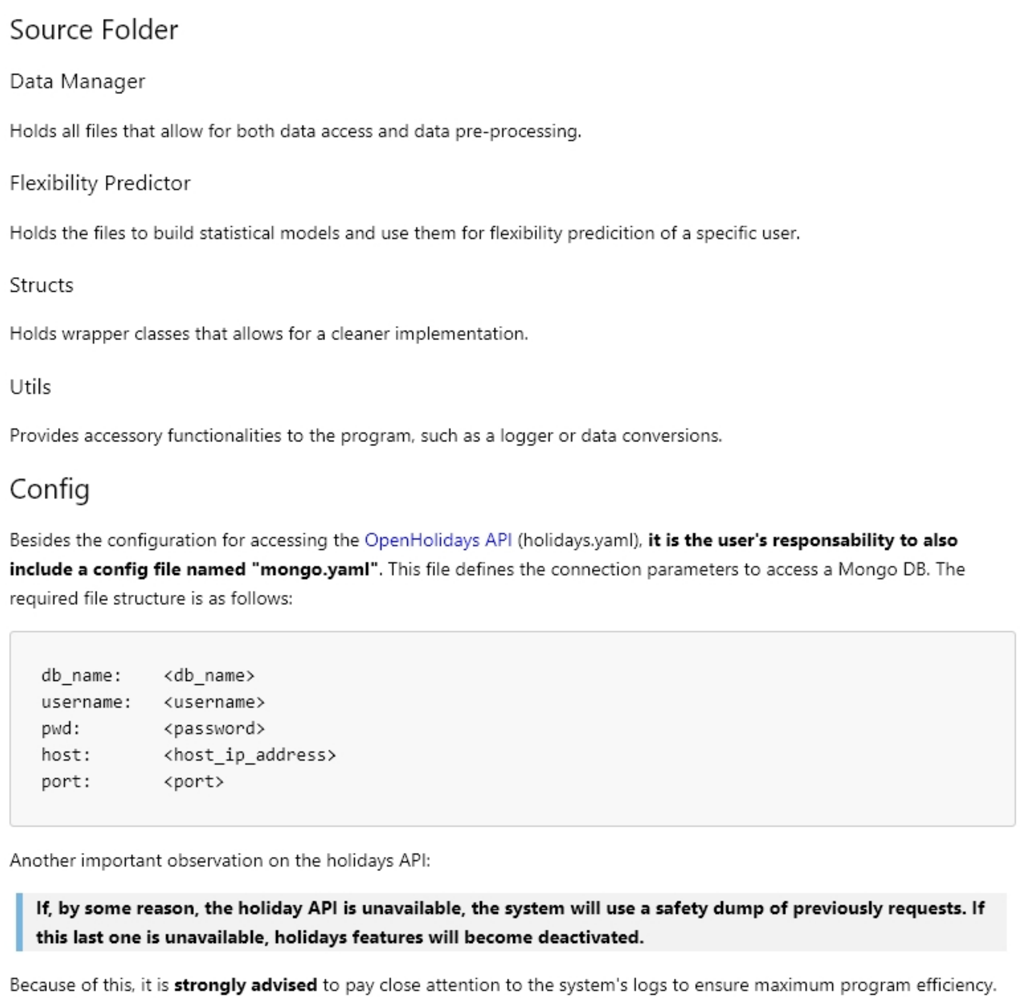
```
pip install requirements.txt
.\run_app.bat
```

Structure

Directory	Description
docs/	Contains documentation about the structure of the application using diagrams
src/	Contains the application itself (click here for details)
test/	Contains both data used to test the application and test functions to ensure the API is working as expected
config/	Contains configuration files for accessing APIs (click here for details)
logs/	Contains system logs

Figure 29 – Project's repository README (page 1)

Appendix H Guide README (page 2)



Source Folder

Data Manager

Holds all files that allow for both data access and data pre-processing.

Flexibility Predictor

Holds the files to build statistical models and use them for flexibility predicition of a specific user.

Structs

Holds wrapper classes that allows for a cleaner implementation.

Utils

Provides accessory functionalities to the program, such as a logger or data conversions.

Config

Besides the configuration for accessing the OpenHolidays API (holidays.yaml), **it is the user's responsability to also include a config file named "mongo.yaml"**. This file defines the connection parameters to access a Mongo DB. The required file structure is as follows:

```
db_name:    <db_name>
username:   <username>
pwd:        <password>
host:       <host_ip_address>
port:       <port>
```

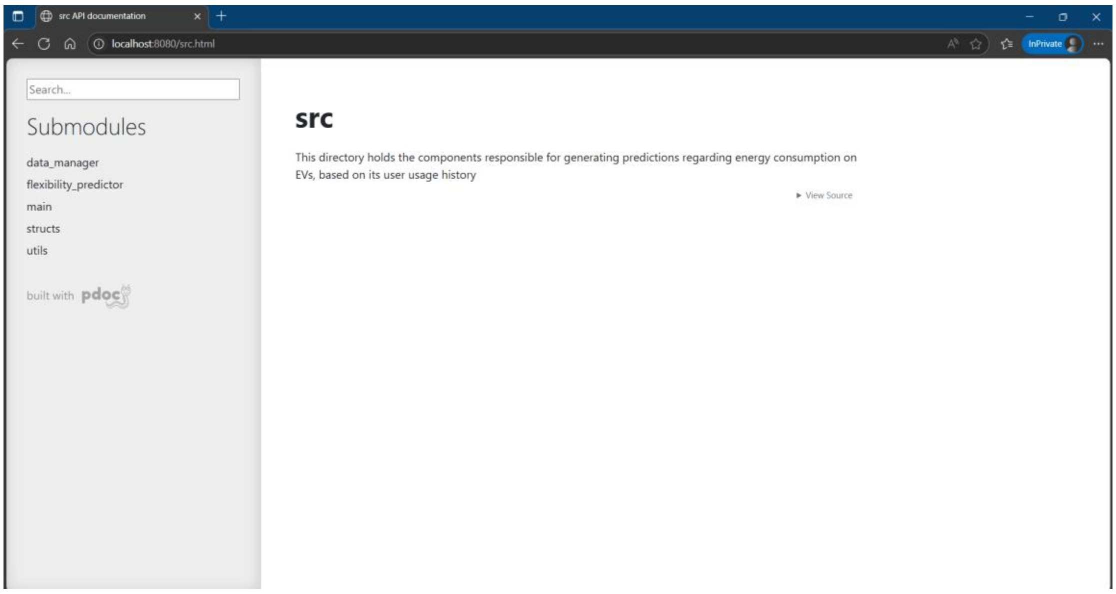
Another important observation on the holidays API:

> **If, by some reason, the holiday API is unavailable, the system will use a safety dump of previously requests. If this last one is unavailable, holidays features will become deactivated.**

Because of this, it is **strongly advised** to pay close attention to the system's logs to ensure maximum program efficiency.

Figure 30 – Project's repository README (page 2)

Appendix I Documentation



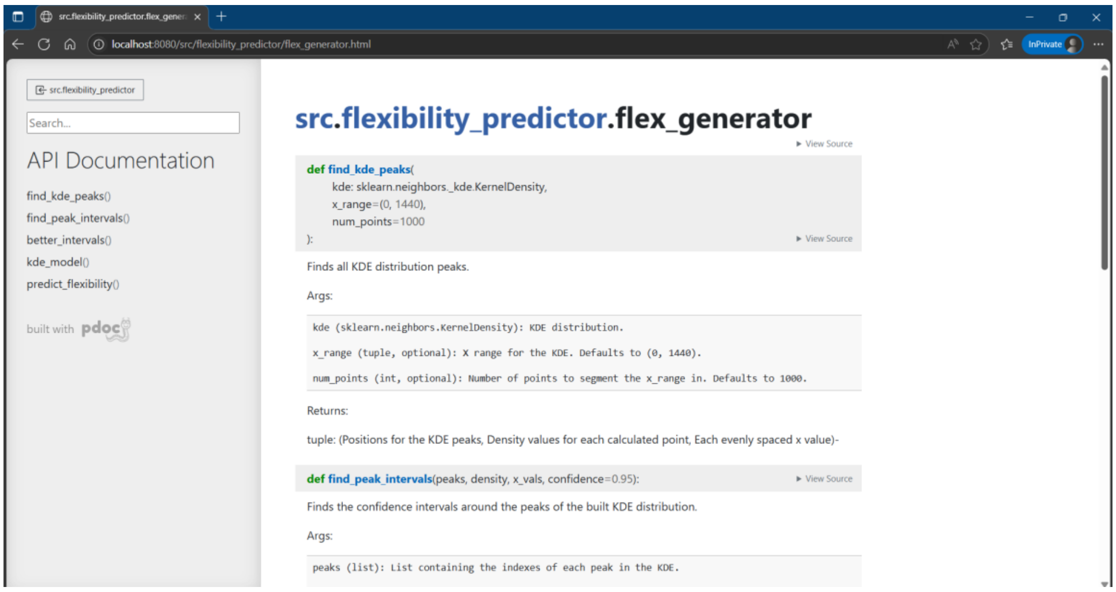
Submodules

- data_manager
- flexibility_predictor
- main
- structs
- utils

src

This directory holds the components responsible for generating predictions regarding energy consumption on EVs, based on its user usage history

Figure 31 – Page for the flexibility predictor component documentation


src.flexibility_predictor.flex_gener
localhost:8080/src/flexibility_predictor/flex_generator.html
src.flexibility_predictor
Search...
InPrivate
built with pdoc
View Source
View Source
View Source

API Documentation

- find_kde_peaks()
- find_peak_intervals()
- better_intervals()
- kde_model()
- predict_flexibility()

src.flexibility_predictor.flex_generator

```
def find_kde_peaks(
        kde: sklearn.neighbors._kde.KernelDensity,
        x_range=(0, 1440),
        num_points=1000
):
```

Finds all KDE distribution peaks.

Args:

```
kde (sklearn.neighbors.KernelDensity): KDE distribution.
x_range (tuple, optional): X range for the KDE. Defaults to (0, 1440).
num_points (int, optional): Number of points to segment the x_range in. Defaults to 1000.
```

Returns:

tuple: (Positions for the KDE peaks, Density values for each calculated point, Each evenly spaced x value)-

```
def find_peak_intervals(peaks, density, x_vals, confidence=0.95):
```

Finds the confidence intervals around the peaks of the built KDE distribution.

Args:

```
peaks (list): List containing the indexes of each peak in the KDE.
```

Figure 32 – Main page for the documentation

Appendix J Project’s Dependencies

Table 7 – Project’s dependencies list

Dependecy	Version
certifi	2025.4.26
charset-normalizer	3.4.1
colorama	0.4.6
contourpy	1.3.2
cycler	0.12.1
dnspython	2.7.0
fonttools	4.57.0
idna	3.1
iniconfig	2.1.0
joblib	1.4.2
kiwisolver	1.4.8
matplotlib	3.10.1
numpy	2.2.5
packaging	25
pandas	2.2.3
pdoc	15.0.4
pillow	11.2.1
pluggy	1.6.0
pymongo	4.12.1
pyparsing	3.2.3
pytest	8.3.5
python-dateutil	2.9.0.post0
pytz	2025.2
PyYAML	6.0.2
requests	2.32.3
scikit-learn	1.6.1
scipy	1.15.2
six	1.17.0
threadpoolctl	3.6.0
tzdata	2025.2

Appendix K Datasets visualisation tool

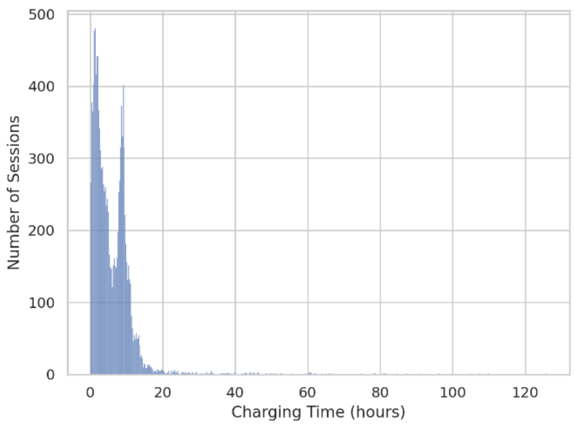


Figure 33 – Website report on the frequency of different charging hours

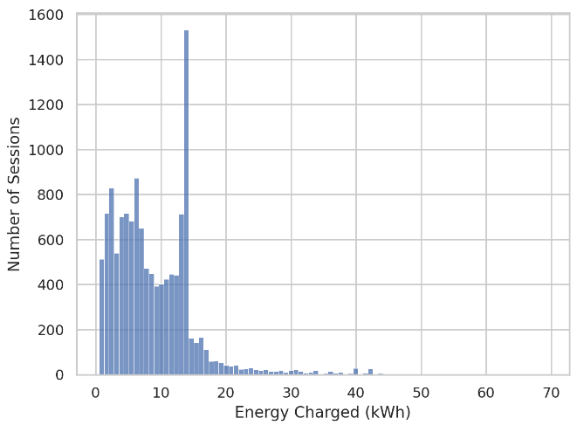


Figure 34 – Website report on the frequency of different amounts of energy charged

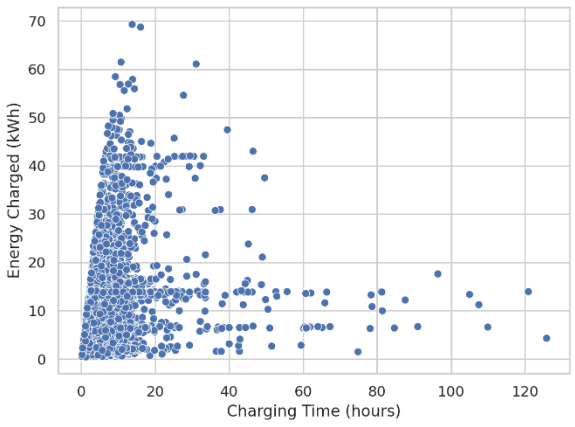


Figure 35 – Website report on the correlation between charging time and energy charged