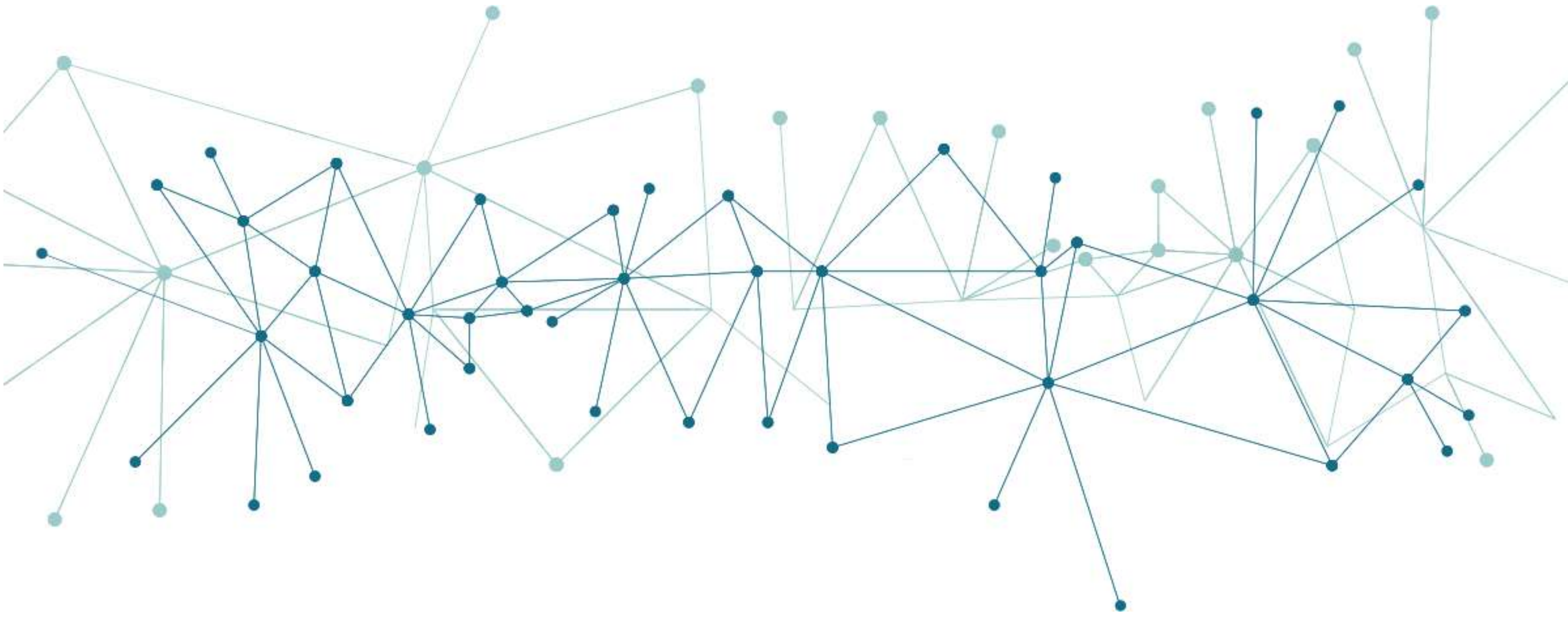
The eDREAM project is co-founded by the EU's Horizon 2020 innovation programme under grant agreement No 774478

enabling new Demand REsponse Advanced, Market oriented and secure technologies, solutions and business models

DELIVERABLE: D4.6 Load Profiles and customer clusters V2

Authors: Lourdes Gallego, Ugo Stecchi, Javier Gómez



Imprint

LOAD PROFILES AND CUSTOMER CLUSTERS V2	May 2020
Contractual Date of Delivery to the EC:	31.05.2020
Actual Date of Delivery to the EC:	31.05.2020
Author(s):	Lourdes Gallego (ATOS), Ugo Stecchi (ATOS), Javier Gomez (ATOS), Napoleon Bezas (CERTH), Paraskevas Koukaras (CERTH), Christos Tjortjis (CERTH), Dimos Ioannidis (CERTH), Giuseppe Mastandrea (E@W), Luigi D'Oriano (E@W), Giuseppe Rocco Rana (E@W), Tommaso Bragatto (ASM), Victoria Murcia (KIWI), Carolina Fernandes (KIWI),
Participant(s):	ATOS, E@W, ASM, KIWI, CERTH
Project:	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models (eDREAM)
Work package:	WP4 – Next generation DR Services for Aggregators and Customers
Task:	4.2 – Big Data clustering Techniques for load profiling and customer segmentation
Confidentiality:	public
Version:	0.9 for internal peer review

Legal Disclaimer

The project enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models (eDREAM) has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 774478. The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the Innovation and Networks Executive Agency (INEA) or the European Commission (EC). INEA or the EC are not responsible for any use that may be made of the information contained therein.

Copyright

Executive Summary

The deliverable D4.6 "*Load Profile and Customer Clusters V2*" is the second deliverable associated wich task 4.2 "*Big Data Clustering techniques for load profiling and customer segmentation*", which main objective is to propose techniques and methodologies for clustering the profiles of prosumers. This deliverable describes the architecture, integration, data-flows, techniques and methodologies adopted for the modules that compose the so-called Big Data Layer: "*Load Profiling*", "*Big Data Clustering at Multiple Scale*" and "*Customer Segmentation*"; a relevant portion of the architecture layer "*Next Generation Services for Aggregators and Customers*".

This document provides the description of the architecture, integration, methodologies and techniques of the Big Data Layer components as follows.

Chapter 1: describes the objectives of task 4.6, derived from the work in the previous deliverable 4.2 of the same task 4.2. Later, a description of the dataset from the Italian Pilot site (ASM) is provided, which will be used throughout the document. The potential application of the described tools in the other pilot site in U.K (Kiwi Power) is described as well.

Chapter 2 includes all the elements useful for the integration of T4.2 components into the eDREAM platform; the development of these components has followed the concepts of the stable and scalable solution described in deliverable 4.2. The use cases in which the tools will be tested, have been analysed in section 2.1 considering the interaction of three components with the other modules. The Big Data Layer architecture is described in section 2.2 and 2.3, with comprehensive information of all sub-components and technologies adopted. An important preliminary activity in big data analysis is the pre-processing, that deals with the quality of data ingested. Section 2.4 provides details of the pre-processing module that oversees filtering and cleaning the data to be processed. Several techniques have been applied to ASM dataset and for each one of them, graphic results are provided.

The first one of the three modules, the Load Profiling is described in **Chapter 3**. This component deals with the organization of data according to different timing. Basically, when a large dataset of energy profiles is considered, timeseries need to be reconfigured in order to extract information from them. In this case eleven different options for profiling have been identified, that are generally considered for the customer portfolio characterization (i.e. from ASM dataset). In this case, the different profiles represent the features for the next clustering module. It is important to remark options 10 and 11 deal with the organization of data from the "Electricity consumption/production forecasting" module which calculates baseline and flexibility profiles for the same ASM dataset.

In **Chapter 4** the clustering algorithm is described. It is indeed a combination of three different algorithm that could work or independently. As a matter of fact, plenty of clustering techniques are today available with different specifications and performances depending on the dataset and the feature to cluster. In eDREAM and in particular in Task 4.2 there is the possibility to cluster large datasets for market analysis, that could be required by a service operator (DSO, Aggregator, Retailer, etc.). This clustering analysis generally considers a high number of heterogeneous users with several years of data (even with poor granularity). In this case the overall architecture should be able to treat large volumes of data and extract as many insights as possible. On the other hand, it is necessary to cluster a different dataset composed by fewer prosumers (sometimes more homogeneous data), but with higher granularity and in a shorter time. The first clustering is useful for portfolio characterization, market analysis, offering proper tariffs and planning activities in general, while the second one could be adopted for intraday or fast response market applications. In this deliverable two different clustering techniques have been developed for both purposes: Autoencoder (a deep learning technique with high performance in big data applications) and K-means (a more generic technique for the second case that also provides good scalability). A third technique DB-Scan has been considered for its complementary features with respect of the K-means. For

each one of the algorithms, graphic results are depicted; Autoencoder have been used only in one option (the daily profiles of the whole prosumers portfolio was the only one with enough data), while K-means and DB Scan with multiple options.

Finally, in **Chapter 5** the Customer Segmentation module is described. Based on the clusters previously calculated by the Big Data Clustering at multiple scale module, this component can be fed with a generic energy user profile and is able to assign it to the proper cluster, with an accuracy value that confirm the matching between the customer profile and the cluster overall profile.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

eDREAM	enabling new Demand Response Advanced, Market oriented and secure technologies, solutions and business models
DBSCan	Density Based Scan
DSO	Distribution System Operator
ESD	Extreme Studentized Deviate test
HP	Potential Hired
PCA	Principal Component Analysis
POD	Point Of Delivery
S-H-ESD	Seasonal Hybrid Extreme Studentized Deviate

1 Introduction

1.1 Objective

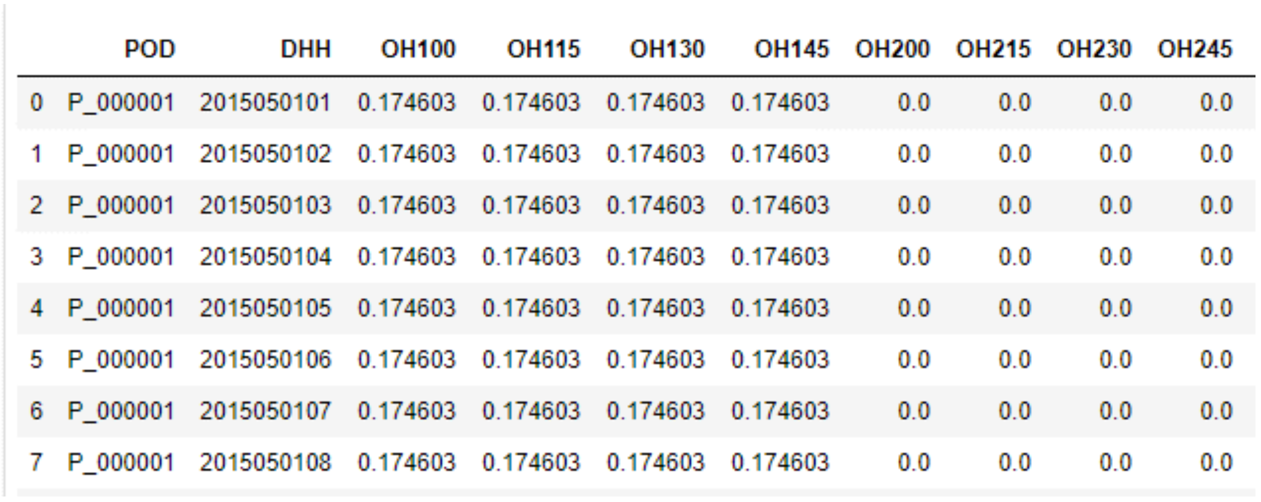
The objective of this deliverable is to describe the development of the three modules of the eDREAM architecture "*Load Profiling*", "*Big Data Clustering at multiple scale*", and "*Customer Segmentation*". Starting from the information included in Deliverable 4.2, where methodology, procedures, and technologies had been identified, in this second version of the document the actual development of the three modules is described. In particular, this document describes the algorithms composing the modules, all the technical details and the choices taken by developers and the complete operation of the considered tools, taking into account the overall objectives of the eDREAM platform and the use cases where they are going to be validated. The three software components are part of the so-called Big Data Layer, a set of tools adopting big data approach for providing added value services for the eDREAM platform users.

1.2 Pilot Sites Application

This chapter aims to describe the provided historical datasets: where they come from, collected period time, dataset quality and requirements that must be addressed for being used in order to succeed the objectives of this deliverable:

1.2.1 Italian Pilot

ASM Terni provides two datasets, collected from smart meters of prosumers and final users in KW/h, coded and anonymized by alphanumeric unique code as "P_XXXXX". Datasets are composed by 2 folders named: "*Scambio*" and "*Prelievo*" (Italian words respectively for "Exchange" and "Withdrawal"). Both files are organized with the following fields represented in Figure 1:



	POD	DHH	OH100	OH115	OH130	OH145	OH200	OH215	OH230	OH245
0	P_000001	2015050101	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
1	P_000001	2015050102	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
2	P_000001	2015050103	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
3	P_000001	2015050104	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
4	P_000001	2015050105	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
5	P_000001	2015050106	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
6	P_000001	2015050107	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0
7	P_000001	2015050108	0.174603	0.174603	0.174603	0.174603	0.0	0.0	0.0	0.0

Figure 1: ASM dataset fields

The name of the fields are acronyms hereby described:

- POD: it is the Point of Delivery, the unique code of the supply or exchange assigned by the DSO to a user
- DHH: datetime interval of the data included in the OH** fields of the same line. The data is written in the format: yyyymmddhh

- OH100, OH115, OH130, OH145: refer to the active energy consumed or produced if the POD is exchanging energy , in the first, second, third and fourth 15 minutes of the time indicated in the DHH field respectively.
- OH200, OH215, OH230, OH245: refer to the reactive energy consumed or produced if the POD is exchanging energy , in the first, second, third and fourth 15 minutes of the time indicated in the DHH field respectively.

As for reactive values, both dataset present null values so this variable will not be taken into account for further use or analysis.

1.2.1.1 “*Scambio*” dataset

This dataset consists of exchanged electricity energy records every 15 min for almost 4 years (from May 2015 to February 2019) from 1229 prosumers. The dataset should record two measures (consumption and generation by user), but actually, only consumption data are available in the end, , so it is not possible to understand the total amount of energy generated/consumed for each record.

Looking deeper into the dataset, it seems that most of the records are synthetic measurements, as depicted in Figure 2. It means that with few real signals from smart meters, it is possible to recreate approximations of the missing values. This technique could originate recurring patterns in profiles that make the dataset useless for flexibility or clustering analysis purposes (for the flatness of the curve).

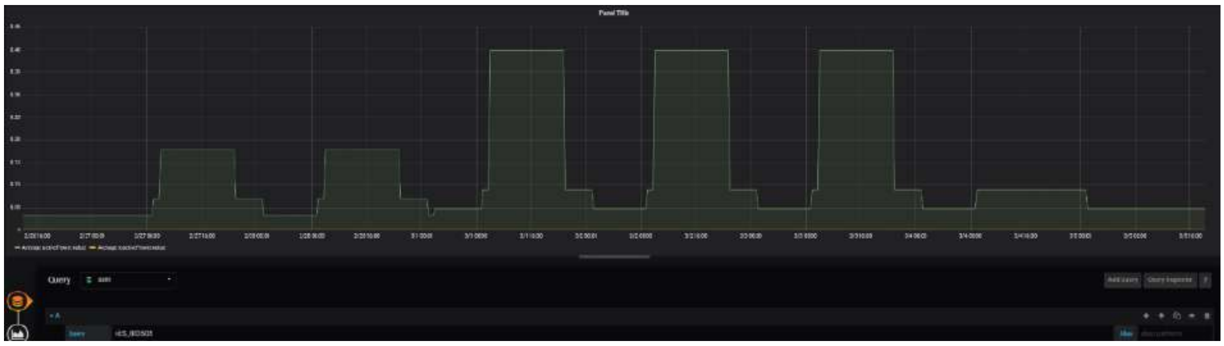


Figure 2: Sample of the “scambio” profile for a random prosumer

1.2.1.2 “Prelievo” dataset

This dataset includes the values of the energy consumed by final users. The data are collected every 15 minutes, for more than 4 years (from January 2015 to February 2019) from 561 users. As shown in Figure 3, the values of this dataset present a pointed and seasonal curve with real measurements.

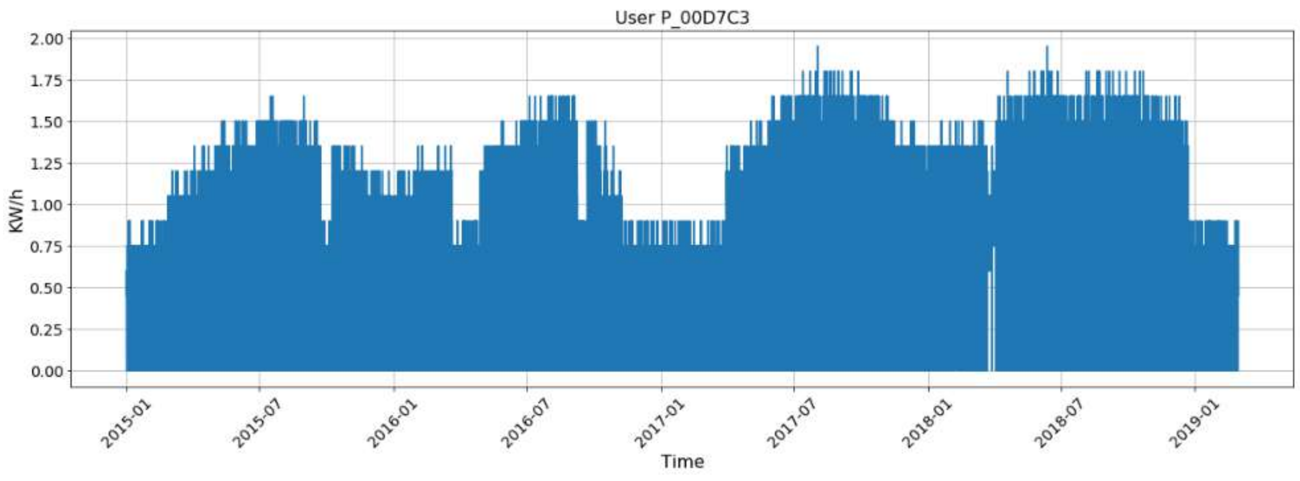


Figure 3: Sample of the "prelievo" profile for a random final user

1.2.2 UK Pilot

Whilst not directly involved in the testing and validation phase of this tool, the U.K. pilot, coordinated by Kiwi Power, can be a potential user for further applications. Thus, some aspects of this pilot have been considered during T4.2 activities to allow for replicability and maximization of the impact.

Although the mentioned site (see section 7 of Deliverable D4.2) is not able to provide data aligned with the requirements for a clustering application, Kiwi Power, as an aggregator, would have access to large volumes of data with time-wise granularity in the range of minutes. On the other hand, in order to ensure access to different service markets, it is important to create coalitions of users (i.e. clusters) rapidly. This means the considered application should be able to detect and gather details in customers profiles patterns with a certain degree of accuracy while being flexible enough to rapidly respond to large volumes of data. Due to the computational and timing specific nature of each problem, no single technique could fit all and as so, attempting a unique solution would be misguided.

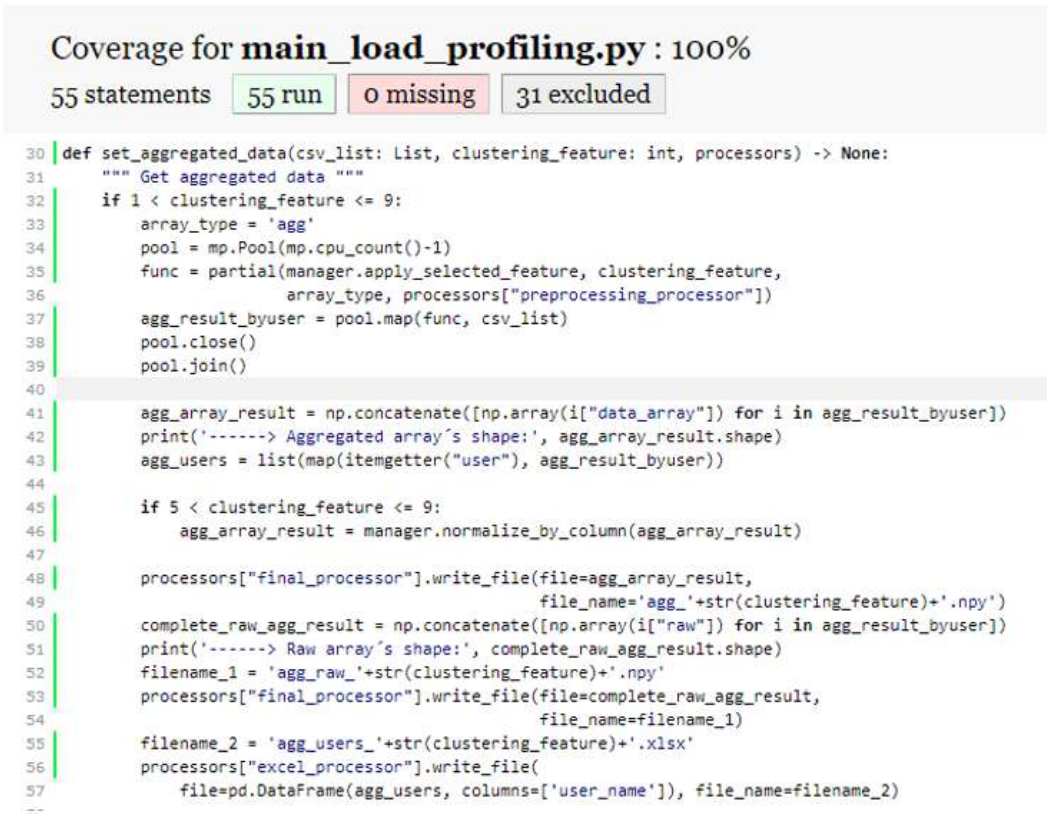
For this purpose, it has been decided to have a tool based on a selection of algorithms that jointly have the necessary characteristics to tackle individual problems and cover as many scenarios as possible. The scalability constraint has been addressed in D4.2 and in the following section in this document, while the description of the different adopted algorithms and their activation is included in section 4.

2 Implementation of a stable and scalable solution

This section deals with the implementation strategy of the big data layer and its integration into the eDREAM platform. According to the methodology defined in the first version of this deliverable (Ref. to D4.2), the big data layer should accomplish generic requirements in terms of stability and scalability. To achieve these two features, it was proposed a design strategy based on the so-called 5 V's: Volume, Velocity, Variety, Veracity and Value (use the same refs. of D4.2 page 14). So basically, the 5V's have been adopted during the during the design phase in order to obtain a scalable and stable solution during the development and integration phase.

Starting from the definition of stability requirements described in the Deliverable 4.2, the operational stability is somehow ensured by design due to intrinsic characteristic of the eDREAM platform based on microservices architecture (more information about this in Deliverable 6.1 and Deliverable 6.2). On the other hand, some tests have been conducted to check the proper operation of the modules; unit tests, functional tests and integration tests.

Unit tests have been planned in the Deliverable D6.1 "*Concept for Joint Integration and Interconnection*", where the plan for the integration of the project's modules into the eDREAM platform is described. In unit test, every module and every portion of module's code is tested in isolated way for checking if it works or not. The unit tests are uploaded into the GitLab repository in accordance to the Continuous Integration (CI) methodology proposed in Deliverable D6.2. For instance, in following Figure 4 the screenshot of the unit test of a script sample of the load profiling module (described in section 3) is presented. While in Figure 5 the unit test of the entire pipeline of the Customer Segmentation module is shown.



Coverage for **main_load_profiling.py** : 100%

55 statements | 55 run | 0 missing | 31 excluded

```
def set_aggregated_data(csv_list: List, clustering_feature: int, processors) -> None:
    """ Get aggregated data """
    if 1 < clustering_feature <= 9:
        array_type = 'agg'
        pool = mp.Pool(mp.cpu_count()-1)
        func = partial(manager.apply_selected_feature, clustering_feature,
                       array_type, processors["preprocessing_processor"])
        agg_result_byuser = pool.map(func, csv_list)
        pool.close()
        pool.join()

        agg_array_result = np.concatenate([np.array(i["data_array"]) for i in agg_result_byuser])
        print('------> Aggregated array´s shape:', agg_array_result.shape)
        agg_users = list(map(itemgetter("user"), agg_result_byuser))

        if 5 < clustering_feature <= 9:
            agg_array_result = manager.normalize_by_column(agg_array_result)

        processors["final_processor"].write_file(file=agg_array_result,
                                                  file_name='agg_'+str(clustering_feature)+'.npy')
        complete_raw_agg_result = np.concatenate([np.array(i["raw"]) for i in agg_result_byuser])
        print('------> Raw array´s shape:', complete_raw_agg_result.shape)
        filename_1 = 'agg_raw_'+str(clustering_feature)+'.npy'
        processors["final_processor"].write_file(file=complete_raw_agg_result,
                                                  file_name=filename_1)
        filename_2 = 'agg_users_'+str(clustering_feature)+'.xlsx'
        processors["excel_processor"].write_file(
            file=pd.DataFrame(agg_users, columns=['user_name']), file_name=filename_2)
```

Figure 4: Example of unit test of .py script in pre-processing tool

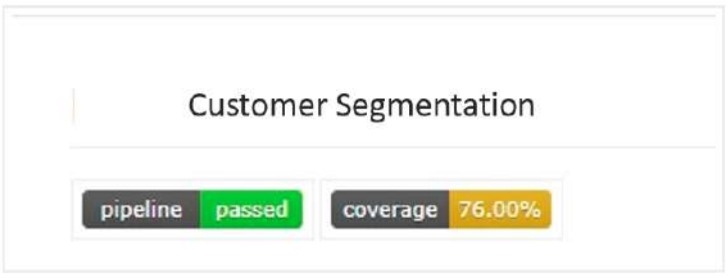


Figure 5: Result of unit test for the whole Customer Segmentation pipeline

Apart from unit tests, functional tests will be conducted during the validation phase (WP7) and Integration tests are considered in Deliverable 6.2.

The microservice architecture is also advantageous for scalability requirements, because it is possible to scale it out in several machines, duplicating only specific services (Ref. D4.2). Along with this intrinsic feature, a specific algorithm for big data clustering have been selected. The Big Data Clustering at multiple scale, the tool devoted to clustering data from customers, have been designed with the option to select different algorithms according to the volume of input data. It means that a specific deep learning algorithm with artificial neural network (see section 4.3.1) is activated when a threshold in data volume is overpassed.

2.1 Integration in eDREAM Platform

The integration of the three modules developed in this document into the eDREAM architecture is described in the deliverable D2.5 (eDREAM H2020 Project) with the latest architectural diagram and the use cases describing the interaction of these modules with the other ones. In the Use Case 1.6 "*Flexibility offering*" is involved the Big Data Clustering at Multiple Scale module, and in Use Case 3.1 "*Prosumers Profiling and Clusterization*" are involved Load Profiling, Big Data Clustering at Multiple Scale and Customer Segmentation.

In the Use Case 1.6 (Figure 6) the Big Data Clustering at Multiple scale module, receives a request from the DSS & DR Strategies Optimization User Interface about a specific subset of prosumers to calculate. This module will send back the resulting subset of prosumers according to the received request.

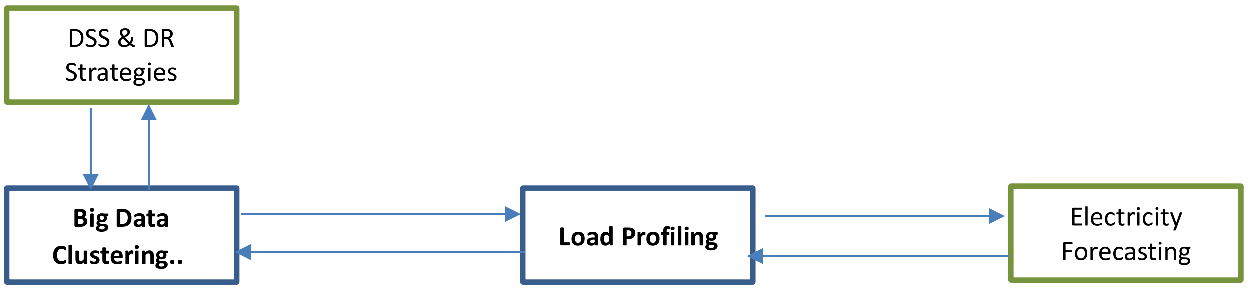


Figure 6: Modules interactions described in Use Case 1.6

In the Use Case 3.1 (Figure 7) the Graph Based analytics module sends a request to the Big Data Clustering at Multiple scale with specific criteria of prosumers' flexibility categorization. The request is forwarded to the Load Profiling module that asks flexibility profiles to Electricity Consumption and Production Forecasting module and extract the flexibility profiles in accordance to the original petition. Those profiles are sent back to the Big Data Clustering module that calculates the clusters of flexibility and sends the results to the Graph based analytics. In case of new prosumers to be assessed, it is possible to gather them into the cluster with the most similar profile through the Customer segmentation module, directly connected to the Big Data Clustering Module.

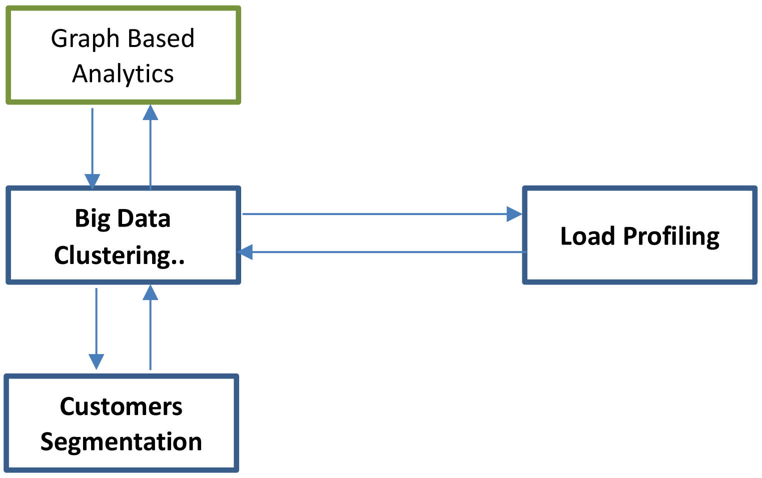


Figure 7: Modules interactions described in Use Case 3.1

2.2 Big Data Layer Architecture

The three modules of task T4.2 are supposed to work and operate with a strong mutual interaction, and since they also share some common objectives, they constitute de-facto, a unique element that has been informally called Big Data Layer. Apart from Load Profiling, Big Data Clustering at Multiple Scale and Customer Segmentation, a pre-processing tool is also part of the Big Data Layer, since it is necessary for the data processing of the three modules. Big Data Layer has been already widely described in Deliverable D4.2 and in this document the high-level architectural scheme is provided as a reminder in Figure 8, for a better comprehension of the tools described in the following sections.

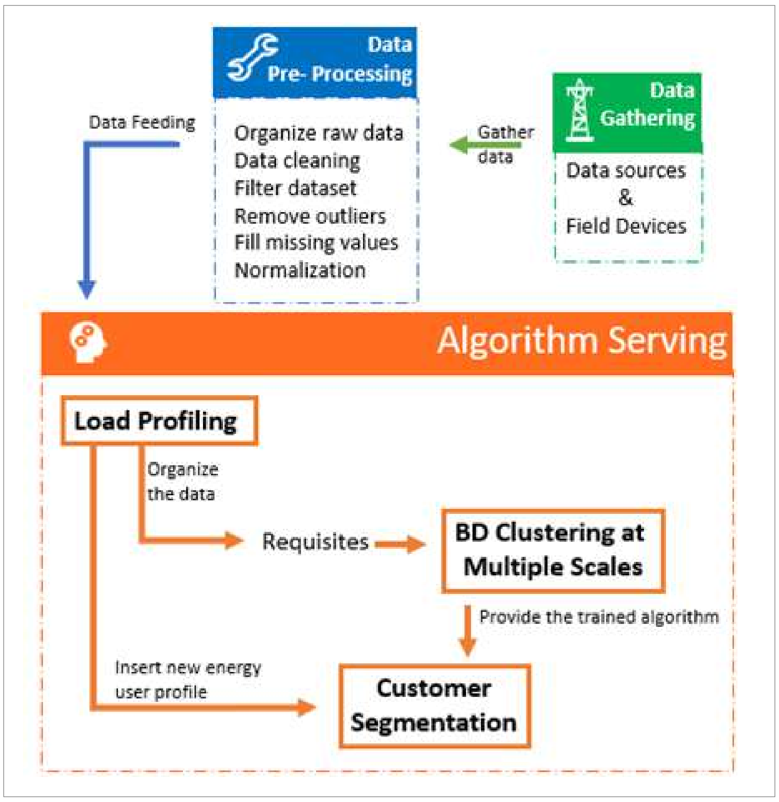
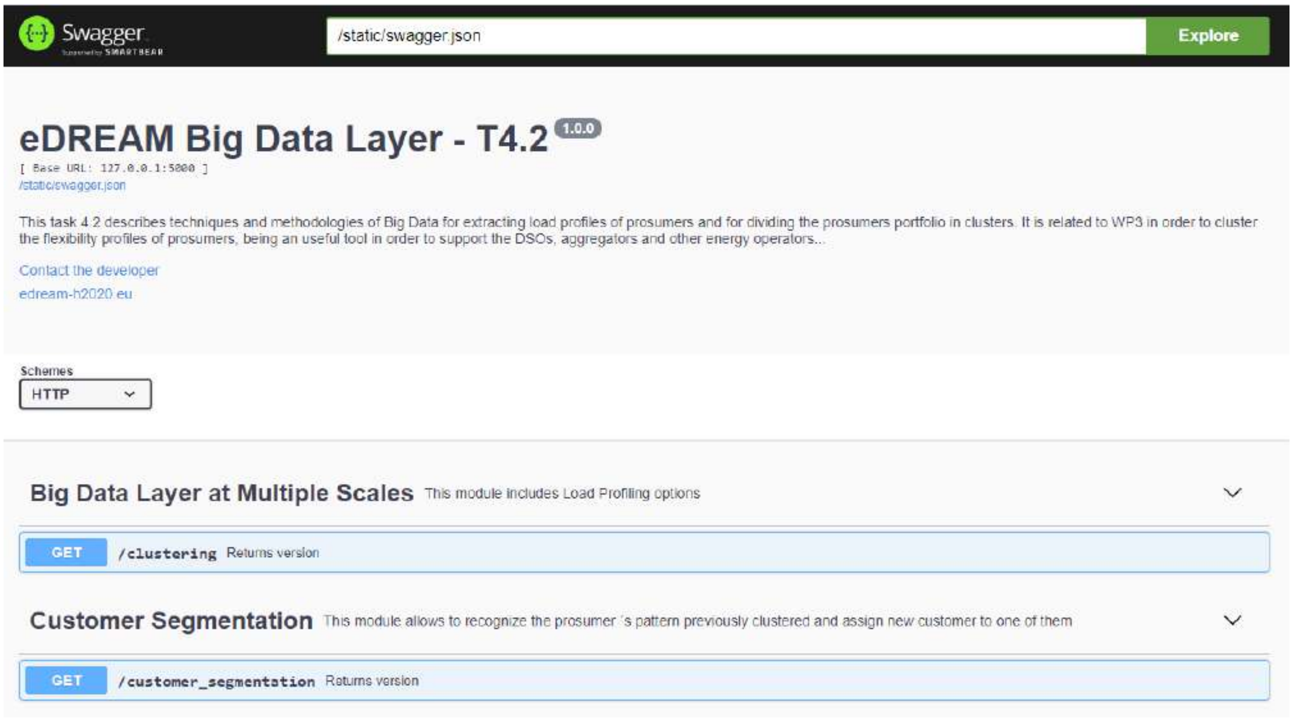


Figure 8: Big Data Layer Architecture

2.3 API for Big Data Layer

Big Data Layer provides a useful and handy web service designed with an API-Rest, enabling access to the algorithm serving outcome. API's resources are associated with : Big Data Clustering at Multiple Scales and Customer Segmentation modules.

In addition, a swagger interface has been built as an easy way to visualize the main API functionality and test the resources, as illustrated in Figure 9.



/static/swagger.json

eDREAM Big Data Layer - T4.2 1.0.0

[Base URL: 127.0.0.1:5000]
/static/swagger.json

This task 4.2 describes techniques and methodologies of Big Data for extracting load profiles of prosumers and for dividing the prosumers portfolio in clusters. It is related to WP3 in order to cluster the flexibility profiles of prosumers, being an useful tool in order to support the DSOs, aggregators and other energy operators...

Contact the developer
edream-h2020.eu

Schemes
HTTP

Big Data Layer at Multiple Scales This module includes Load Profiling options

GET /clustering Returns version

Customer Segmentation This module allows to recognize the prosumer´s pattern previously clustered and assign new customer to one of them

GET /customer_segmentation Returns version

Figure 9: User REST API interface of Big Data Layer

Big Data Clustering at Multiple Scales resource provides the corresponding clusters according to one of the available options in the Load Profiling module, detailed in section 3. Customer segmentation resource requires entering the day and username to obtain the belonging cluster. Both End-point URL and parameters are detailed in the following Table 1, as well as the response obtained.

Big Data Layer – REST API	
Big Data Layer at Multiple Scales	
Description	Through this interface, end users or other modules of eDREAM project are able to access to the clusters and results of Big Data Layer according the option selected
End-point URL	/clustering?option={}&date={}
Parameters	option (see section 3, Table 1) format_time = %Y-%m-%d-%H:%M:%S (only available for option 10)
Allowed HTTP Methods	GET

Class Type of GET response	{
	    "option":{
	        "option_id":"7",
	        "details": { "id": "fall",
	            "granularity_id": "months",
	          }

```
        "algorithm_result":[{ algorith_name: "kmeans",
        "clusters": {
                "num_clusters":"4",
                "details": {
                        "num_cluster_0": "304",
                        "num_cluster_1": "8",
                        "num_cluster_2": "10",
                        "num_cluster_3": "26",
                        },
                "results": [
                        {"cluster": "0",
                         "users": [
                                "P_008104",
                                "P_00E9DF",
                                "P_00B401",
                                ....,
                                "P_00D8C9"
                                ]
                        },
                        {"cluster": "0",
                         "users": [
                               "P_0092C3",
                               "P_00EC6F",
                               ...,
                               "P_00B211"
                               ]
                        },
                        {"cluster": "0",
                         "users": [
                               "P_00A130",
                               "P_011BF8",
                               ...,
                               "P_00B211"
                               ]
                        },
                        },
                        {"cluster": "0",
                         "users": [
                               "P_00AB21",
                               "P_00DEE7",
                               ....,
                               "P_00D7FE"
                               ]
                        ]}
                }
        ]}
}
```

Customer Segmentation	
Description	Through this interface, final users or other modules of eDREAM project are been able to obtain the cluster to which the user provided belongs
End-point URL	/customer_segmentation?user_id={}&date={}
Parameters	user_id: the user identification of which you want to know the cluster to which it belongs date: values of the selected day
Allowed HTTP Methods	GET

Class Type of GET response	{
	"user_id": "P_055FE3"
	"date": "2015-08-17",
	"cluster_belong": 3
	}

Table 1: REST API details of Big Data Layer

2.4 Software and libraries

In Deliverable D4.2 a list of libraries has been identified and proposed for tools development; among the technologies identified this section describes the adopted ones. The main scope of the libraries used in this module is framed in the machine learning and deep learning fields. In this way we can achieve the goal of deploying and developing the Big Data Layer as a fully connected and functional platform.

SOFTWARE:

- **Python** (Python, 2020): is the programming language used to develop all modules that are part of the Big Data Layer. As reference language for machine learning and deep learning, it also counts with advantages such as readable and maintainable code, compatible with major platform and systems, robust standard library, many open source frameworks and tools, fast and easy prototyping algorithms. (Solutions, 2017)

LIBRARIES

- **NumPy** (NumPy, 2020): is a numerical python library, aimed at scientific computing to perform mathematical operations on arrays.
- **Pandas** (Pandas, 2020): is a package that principally works with datasets for processing and analysis, allowing: load, analyse, manipulate and write.
- **Dask** (DASK, 2019): Per its official page, “Dask is a flexible library for parallel computing in Python”. It enables parallelize task of Pandas, Numpy or Scikit-Learn packages. Thus, it can help to optimize and scale algorithms.
- **Scikit-learn** (scikit-learn, 2020): is one of the most famous libraries to use for machine learning. It includes a prebuilt algorithm for classification, regression, clustering, dimensionality reduction, model selection or preprocessing. This package is used in all Big Data Layer modules providing the algorithms of k-means, DBSCAN or Isolation Forest among others.
- **Tensorflow** (TensorFlow, 2020): is the most important library used to create deep learning models. It is created by Google for fast numerical computing and maintained and released under the Apache 2.0 open

source license. One of the great advantages is that it can run on single CPU systems, GPUs or mobile devices as well as in large-scale distributed systems of hundreds of machines. (Brownlee, 2016).

- **Pyculiarity** (Miller, 2018): is a python library originally developed in R for Twitter's Anomaly Detection. This package detects anomalies both in time series and in a vector of numerical values from a statistical standpoint, in the presence of seasonality and an underlying trend.
- **Yellowbrick** (Yellowbrick, 2019): is a set of diagnostic and visual analysis tools designed to facilitate machine learning with scikit-learn. It implements a Visualizer to scikit-learn, through an API, enabling to choose the models, the hyperparameters tuning and creating intuition around feature engineering.

2.5 Pre-processing

This chapter aims to describe all those tasks which cover the pre-processing step, being this a mandatory process, in order to filter and clean the raw data acquiring quality and consistency for future clustering.

This pre-processing step is a pipeline divided into five subtasks:

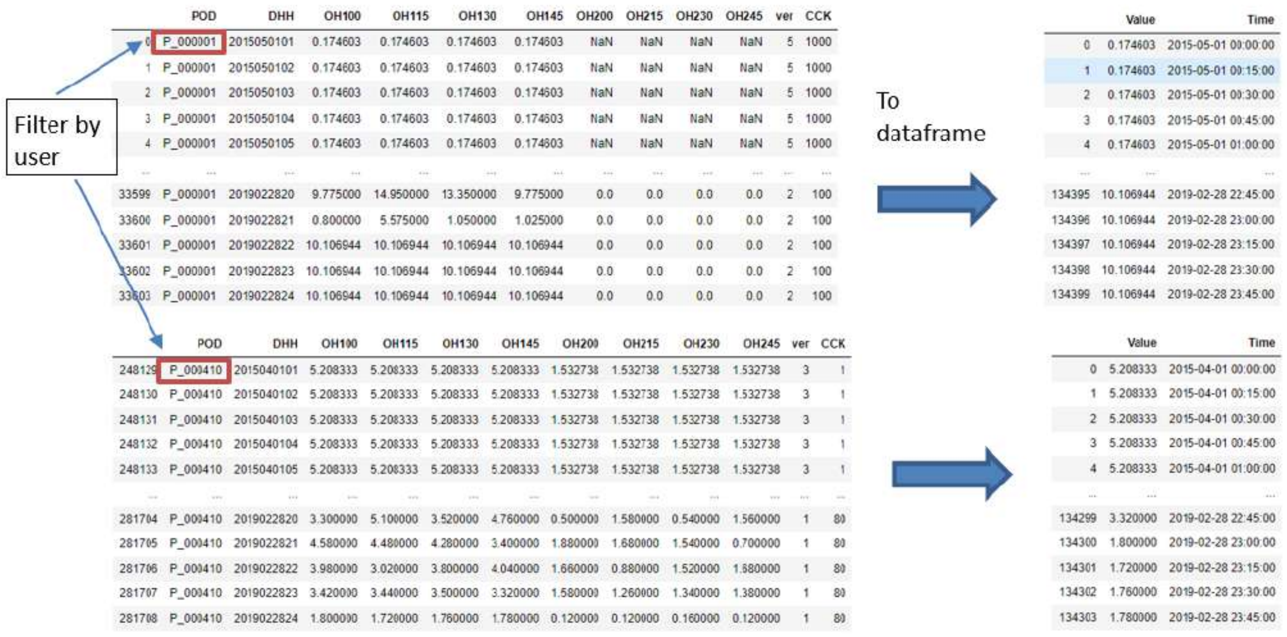
- Organize raw data: due to the datasets coming from different resources (prosumers and final users), these must be organized with a unique structure in order to be homogeneous and manageable
- Data cleaning: this subtask, in essence, deals with the quality of the gathered data carrying out different techniques (e.g. remove duplicated or delete inconsistent data), in order to obtain consistency and usability.
- Filter dataset: once the datasets accomplish the quality and consistency objectives, these should be filtered (i.e. accept only the dataset which counts with a minimum number of points or have a certain number of null values among others) to obtain good results in the analysis.
- Remove outliers: the presence of outliers in the dataset can distort the analysis, reason because of they must be removed.
- Fill missing values: the goal of this subtask is to prevent the shortage of data. Firstly, it analyses the cases where it is required or if otherwise, the missing values can be omitted due to its size.
- Normalization: the variety of scales among the initial data sets require the normalization in order to obtain their consumption pattern.

As aforementioned, the module developed for this task can receive the input dataset from the modules of "*Electricity Consumption/Production Forecasting*", "*Baseline Flexibility Estimation*" and historical data (ASM "*prelievo*" dataset) depending on the user's choice. According to the needs of each input source there are different workflows of subtask. For ASM "*prelievo*" dataset all the subtasks are performed due to the historical data had not been pre-processed previously, contrary to Forecasting and Flexibility modules where only required to organize the data to have the same feeding structure for clustering module.

In the next subsections, all subtasks are described in detail applying examples of historical data.

2.5.1 Organize raw data

This task aims to organize the raw data from ASM "*prelievo*" dataset, improving readability and homogenizing the data. To this objective, the initial txt file, presented in Figure 1 is filtered by user and the hour values of each row are harmonized into a data-frame as time series (2-dimensional labelled data structure with columns, (Pandas, 2014) as shown in Figure 10.



	POD	DHH	OH100	OH115	OH130	OH145	OH200	OH215	OH230	OH245	ver	CCK
0	P_000001	2015050101	0.174603	0.174603	0.174603	0.174603	NaN	NaN	NaN	NaN	5	1000
1	P_000001	2015050102	0.174603	0.174603	0.174603	0.174603	NaN	NaN	NaN	NaN	5	1000
2	P_000001	2015050103	0.174603	0.174603	0.174603	0.174603	NaN	NaN	NaN	NaN	5	1000
3	P_000001	2015050104	0.174603	0.174603	0.174603	0.174603	NaN	NaN	NaN	NaN	5	1000
4	P_000001	2015050105	0.174603	0.174603	0.174603	0.174603	NaN	NaN	NaN	NaN	5	1000
...	...	...	...	...	...	...	...	...	...	...	...	...
33599	P_000001	2019022820	9.775000	14.950000	13.350000	9.775000	0.0	0.0	0.0	0.0	2	100
33600	P_000001	2019022821	0.800000	5.575000	1.050000	1.025000	0.0	0.0	0.0	0.0	2	100
33601	P_000001	2019022822	10.106944	10.106944	10.106944	10.106944	0.0	0.0	0.0	0.0	2	100
33602	P_000001	2019022823	10.106944	10.106944	10.106944	10.106944	0.0	0.0	0.0	0.0	2	100
33603	P_000001	2019022824	10.106944	10.106944	10.106944	10.106944	0.0	0.0	0.0	0.0	2	100

	Value	Time
0	0.174603	2015-05-01 00:00:00
1	0.174603	2015-05-01 00:15:00
2	0.174603	2015-05-01 00:30:00
3	0.174603	2015-05-01 00:45:00
4	0.174603	2015-05-01 01:00:00
...	...	...
134395	10.106944	2019-02-28 22:45:00
134396	10.106944	2019-02-28 23:00:00
134397	10.106944	2019-02-28 23:15:00
134398	10.106944	2019-02-28 23:30:00
134399	10.106944	2019-02-28 23:45:00

	POD	DHH	OH100	OH115	OH130	OH145	OH200	OH215	OH230	OH245	ver	CCK
248129	P_000410	2015040101	5.208333	5.208333	5.208333	5.208333	1.532738	1.532738	1.532738	1.532738	3	1
248130	P_000410	2015040102	5.208333	5.208333	5.208333	5.208333	1.532738	1.532738	1.532738	1.532738	3	1
248131	P_000410	2015040103	5.208333	5.208333	5.208333	5.208333	1.532738	1.532738	1.532738	1.532738	3	1
248132	P_000410	2015040104	5.208333	5.208333	5.208333	5.208333	1.532738	1.532738	1.532738	1.532738	3	1
248133	P_000410	2015040105	5.208333	5.208333	5.208333	5.208333	1.532738	1.532738	1.532738	1.532738	3	1
...	...	...	...	...	...	...	...	...	...	...	...	...
281704	P_000410	2019022820	3.300000	5.100000	3.520000	4.760000	0.500000	1.580000	0.540000	1.560000	1	80
281705	P_000410	2019022821	4.580000	4.480000	4.280000	3.400000	1.880000	1.680000	1.540000	0.700000	1	80
281706	P_000410	2019022822	3.980000	3.020000	3.800000	4.040000	1.660000	0.880000	1.520000	1.680000	1	80
281707	P_000410	2019022823	3.420000	3.440000	3.500000	3.320000	1.580000	1.260000	1.340000	1.380000	1	80
281708	P_000410	2019022824	1.800000	1.720000	1.760000	1.780000	0.120000	0.120000	0.160000	0.120000	1	80

	Value	Time
0	5.208333	2015-04-01 00:00:00
1	5.208333	2015-04-01 00:15:00
2	5.208333	2015-04-01 00:30:00
3	5.208333	2015-04-01 00:45:00
4	5.208333	2015-04-01 01:00:00
...	...	...
134299	3.320000	2019-02-28 22:45:00
134300	1.800000	2019-02-28 23:00:00
134301	1.720000	2019-02-28 23:15:00
134302	1.760000	2019-02-28 23:30:00
134303	1.780000	2019-02-28 23:45:00

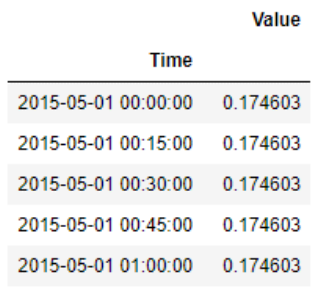
Figure 10: Module for raw data organization

2.5.2 Data cleaning

Once the dataset has been organized, different techniques have been applied to obtain consistency and usability:

1. Set index datetime: because the Pandas´ library is used, the set data-frame index with timestamp allows an agile manipulation;
2. Remove duplicates: having duplicates can lead to wrong analysis when the data pass through the algorithm;
3. Resample by 15 min: this technique allows to fill any gap that there might be, introducing a timestamp if it lacks and applying NaN (Not a Number) to the value;
4. Delete spring clock change: remove the values that belongs to the clock change hour (02:00:00 -> 03:00:00)

Finally, this module saves each user data-frame into csv files as shown in Figure 11.



	Value
Time	
2015-05-01 00:00:00	0.174603
2015-05-01 00:15:00	0.174603
2015-05-01 00:30:00	0.174603
2015-05-01 00:45:00	0.174603
2015-05-01 01:00:00	0.174603

Figure 11: Data cleaning step output

Initially txt file counts with 561 users, but after applying organize raw data and data cleaning, four users are deleted due to the impossibility of reorganizing their data. So, the final users to use are 557.

- Domestic users: 105
- Other users: 452

Since 6kW is the upper limit of power capacity (with +10% of tolerance) for domestic users, we use this threshold to separate domestic customers from other ones. Basically, we assume as domestic user any profile with an hourly consumption lower than 6,6 kWh.

Below, Figure 12 and Figure 13 illustrate both domestic and other monthly consumption.

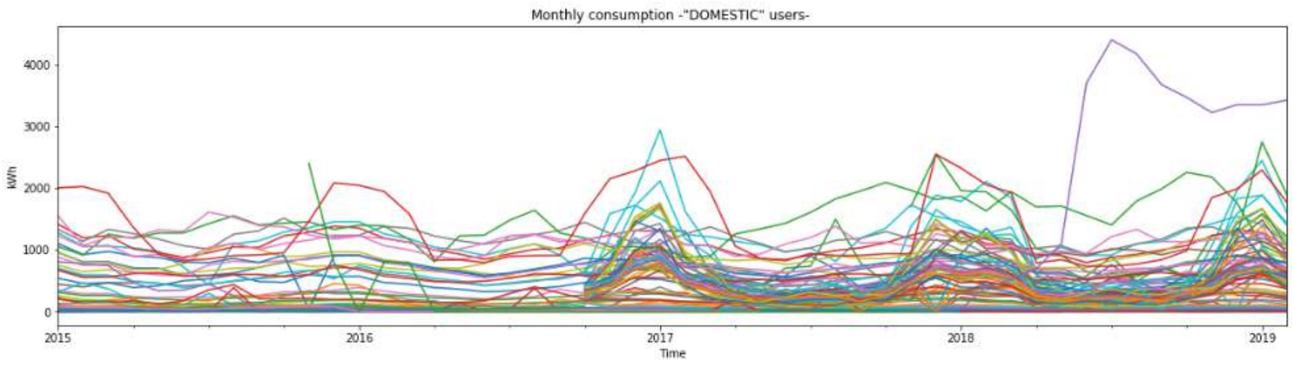


Figure 12: Monthly consumption of domestic users

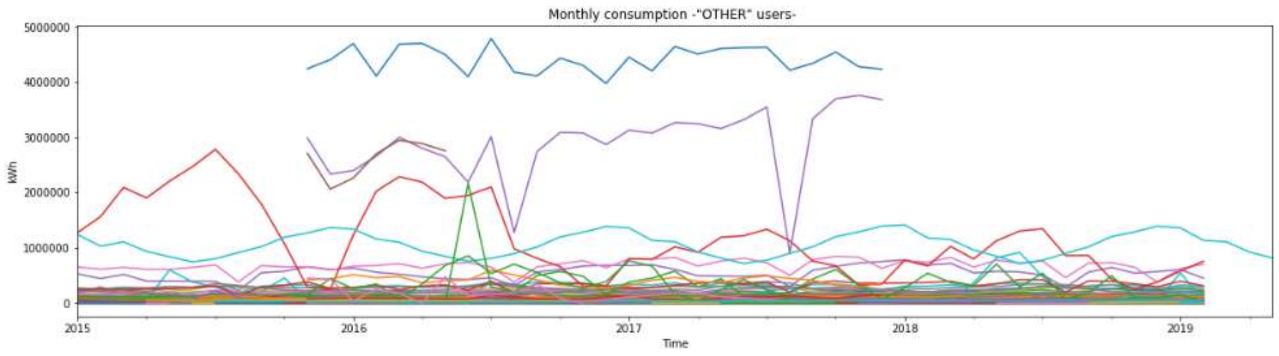
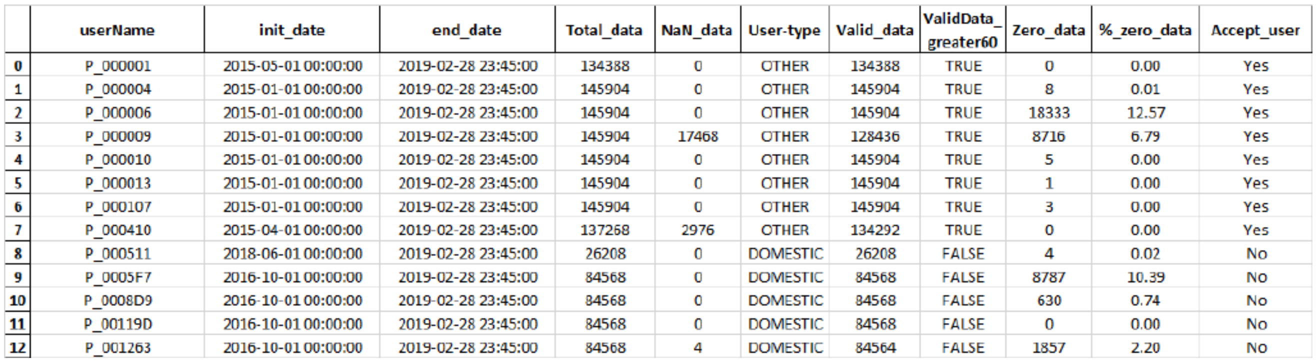


Figure 13: Monthly consumption of other users

2.5.3 Filter dataset

This subtask aims to filter the users, (i.e. accept or exclude), according to several criteria, to obtain good results in the clustering.

At this end, subtask provides an excel file, which looksss as shown in Figure 14, where each row belongs to one user describing : the initial and end datetime period of its records, the total records that it has, number of NaN values, which type of user is, (i.e. domestic or other), number of valid data, valid data greater than 60, which refers to the first criterion, number of zero data and its percentage, second criterion and the final column, that describes if the user finally is accepted or excluded.



	userName	init_date	end_date	Total_data	NaN_data	User-type	Valid_data	ValidData_greater60	Zero_data	%_zero_data	Accept_user
0	P_000001	2015-05-01 00:00:00	2019-02-28 23:45:00	134388	0	OTHER	134388	TRUE	0	0.00	Yes
1	P_000004	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	0	OTHER	145904	TRUE	8	0.01	Yes
2	P_000006	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	0	OTHER	145904	TRUE	18333	12.57	Yes
3	P_000009	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	17468	OTHER	128436	TRUE	8716	6.79	Yes
4	P_000010	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	0	OTHER	145904	TRUE	5	0.00	Yes
5	P_000013	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	0	OTHER	145904	TRUE	1	0.00	Yes
6	P_000107	2015-01-01 00:00:00	2019-02-28 23:45:00	145904	0	OTHER	145904	TRUE	3	0.00	Yes
7	P_000410	2015-04-01 00:00:00	2019-02-28 23:45:00	137268	2976	OTHER	134292	TRUE	0	0.00	Yes
8	P_000511	2018-06-01 00:00:00	2019-02-28 23:45:00	26208	0	DOMESTIC	26208	FALSE	4	0.02	No
9	P_0005F7	2016-10-01 00:00:00	2019-02-28 23:45:00	84568	0	DOMESTIC	84568	FALSE	8787	10.39	No
10	P_0008D9	2016-10-01 00:00:00	2019-02-28 23:45:00	84568	0	DOMESTIC	84568	FALSE	630	0.74	No
11	P_00119D	2016-10-01 00:00:00	2019-02-28 23:45:00	84568	0	DOMESTIC	84568	FALSE	0	0.00	No
12	P_001263	2016-10-01 00:00:00	2019-02-28 23:45:00	84568	4	DOMESTIC	84564	FALSE	1857	2.20	No

Figure 14: Filtered table of users

Analysing in more details, the table, the following conclusions can be drawn:

- 452 users belong to "other"
- 105 users are "domestic"
- 355 users count with all records, values between 01-January to 28-February (145904 values, this includes NaN, zero and valid data) as depicted in Figure 15.

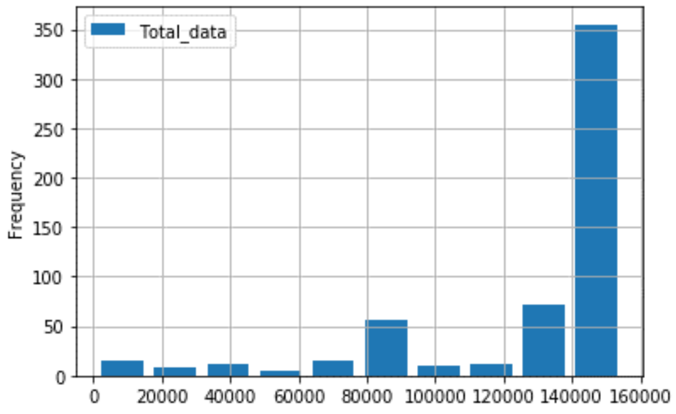


Figure 15: Histogram of total data for all users

- The following histogram of NaN values, Figure 16, shows a relevant quantity of null data (record = NaN, i.e. empty or none), but only 15 users have more than 5000 nan values so in this case, it is decided not to set a filter.

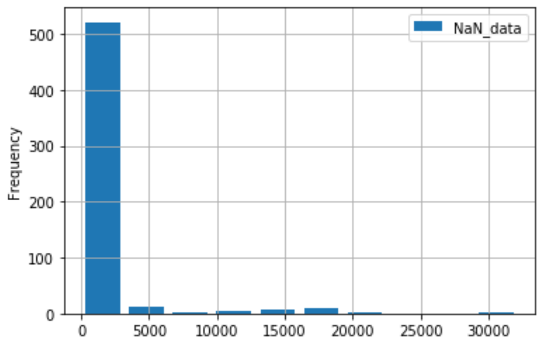


Figure 16: Histogram of Nan data for all users

- As in the previous case, through the analysis of the percentage of zero values in all the dataset (Figure 17), it seems that the quantity is also relevant, so it is decided to set a filter.

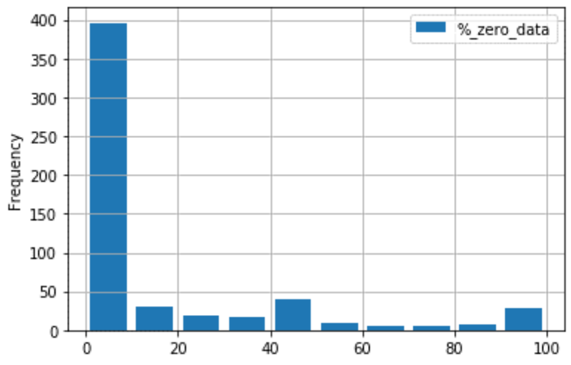


Figure 17: Percentage of zero data for all users

The choice to set the following filters, is based mostly on the previous results and on the fact that for the characterization of flexibility in the customer portfolio the value should be as lower as possible; allowing the acceptance of the largest number of users. That said, to accept one user, it must accomplish the following criteria:

1. $valid\ data\ \geq 87542\ values$: understanding as valid data, those values different than NaN (i.e. empty or none), and being at least the 60% of the total records per user (from January 2015 to February 2019 = 145904 * 0,6 = 87542 records)
2. $zero\ data\ \leq 30\%$: if its valid data: being zero data those records equal to 0

After filtering all the users with those criteria:

- 2 “domestic” users are accepted
- 347 “other” users are being accepted

So that 349 users are accepted, representing 62.65% of the total users (557). For these users the remove outliers and fill missing values modules are applied. Hereafter, Figure 18 and Figure 19 depict the monthly consumption of domestic and other users respectively.

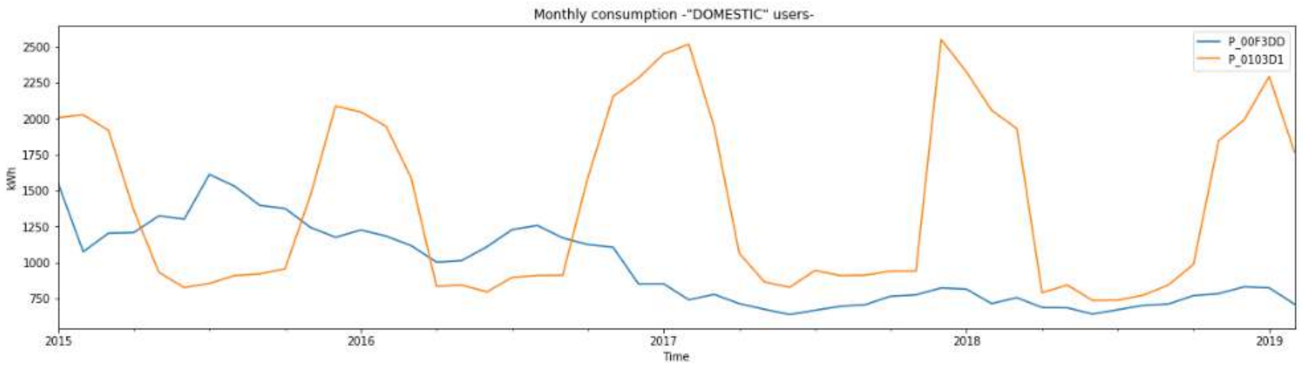


Figure 18: Monthly consumption of domestic accepted users

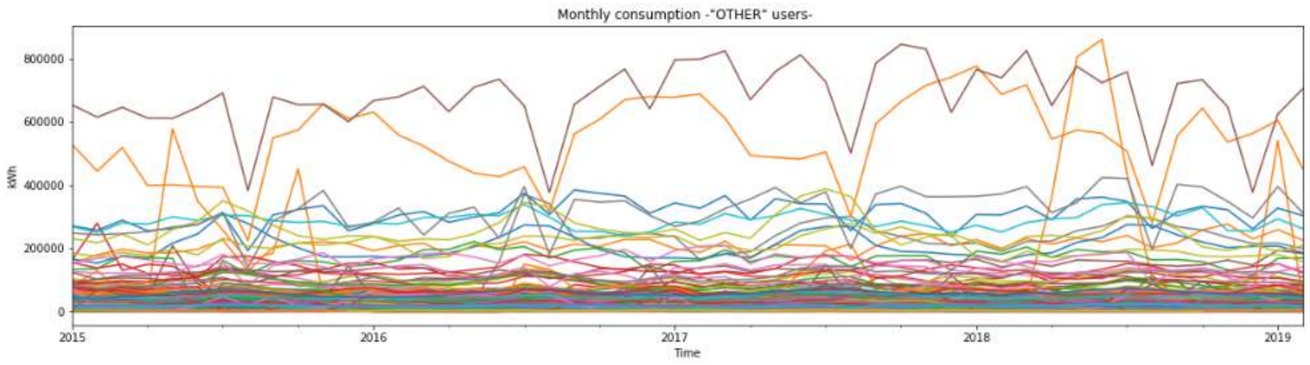


Figure 19: Monthly consumption other accepted users

2.5.4 Removing outliers

The presence of anomalies in the dataset can affect the analysis and further clustering; so that they must be removed or replaced. Three techniques are applied: threshold, Isolation Forest and Pyculiarity algorithms.

1. **Threshold:** this technique replaces all records which exceed the contractual power, by the user (maximum hourly consumption hired) to its value contracted value;

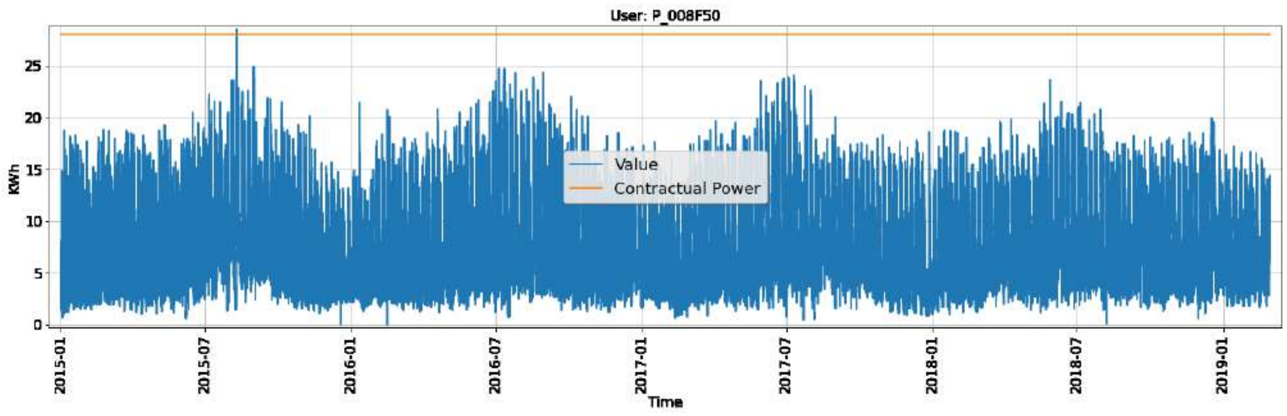


Figure 20: Sample of four years energy consumption data (blue) and consumption capacity threshold (yellow)

2. **Isolation Forest:** is an unsupervised learning algorithm based on Decision Tree for anomaly detection that uses an ensemble of Isolation Trees for the given data points.

   Isolation Forest algorithm (K, 2020) selects randomly a characteristic from the available ones from the given data set and isolates the outliers. Subsequently, it selects a value, also randomly, from the maximum and minimum values of that previously selected characteristic. The anomalous points when producing shorter routes allow them to be distinguished from the rest of the data, which generate longer routes.

   To use this technique two parameters must be set (Scikit-learn, 2019):

   - max_samples: the number used to train the algorithm. Because all the users having more than 87542 records, it is set as 90000.
   - contamination: the number of outliers in the dataset. This parameter has been set as 0.1 which represents that the maximum number of outliers are being 10% of the total dataset

   The output is an array with the same length as the dataset, with two values: "-1" corresponding to outliers, depicted with red circles in Figure 21, and "1" corresponding to normal data, blue line.

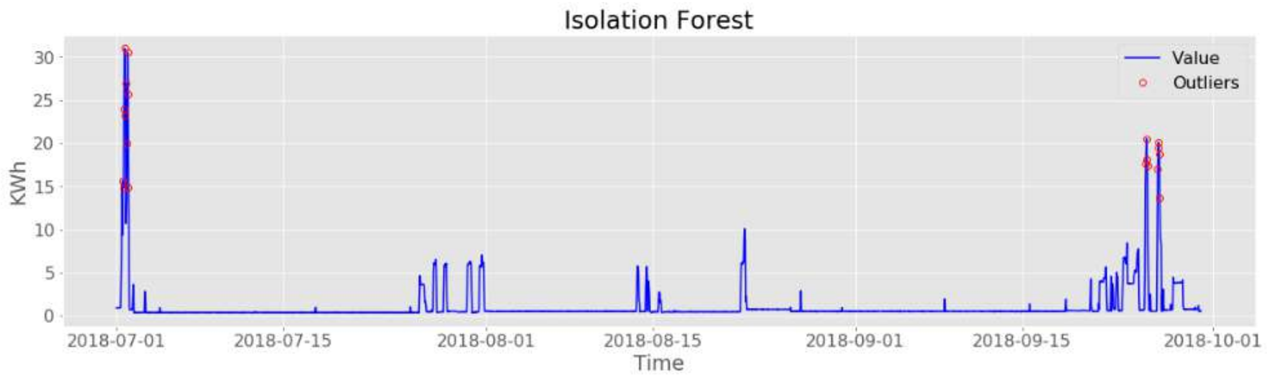


Figure 21: Isolation Forest outliers

3. **Pyculiarity:** (Miller, 2018) is a technique for anomaly detection which uses the seasonal hybrid ESD Test in order to detect anomalies in seasonal univariate time series, where the input is a series of <timestamp, value> pairs. Its advantage is not only to detect the anomalies but also estimate the expected value. As well as in Isolation forest algorithm, two parameters must be set
   - **alpha:** set to 0.05 is the level of statistical significance with which to accept or reject anomalies.
   - **max_anoms**: maximum number of anomalies that S-H-ESD will detect as a percentage of the data. This parameter has been set as 0.1 which represents the maximum number of outliers are 10% of the total dataset.

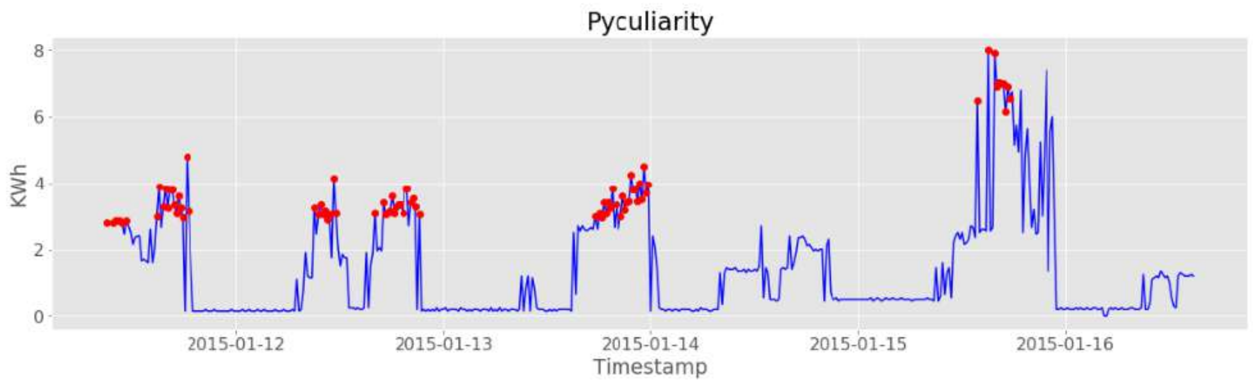


Figure 22: Pyculiarity outliers

Both Isolation Forest and Pyculiarity search up to a maximum of 10% of outliers in the dataset. Therefore, only those outliers (red points in Figure 21 and Figure 22) that are presented in both methods will to be turned into the offered Pyculiarity expected value.

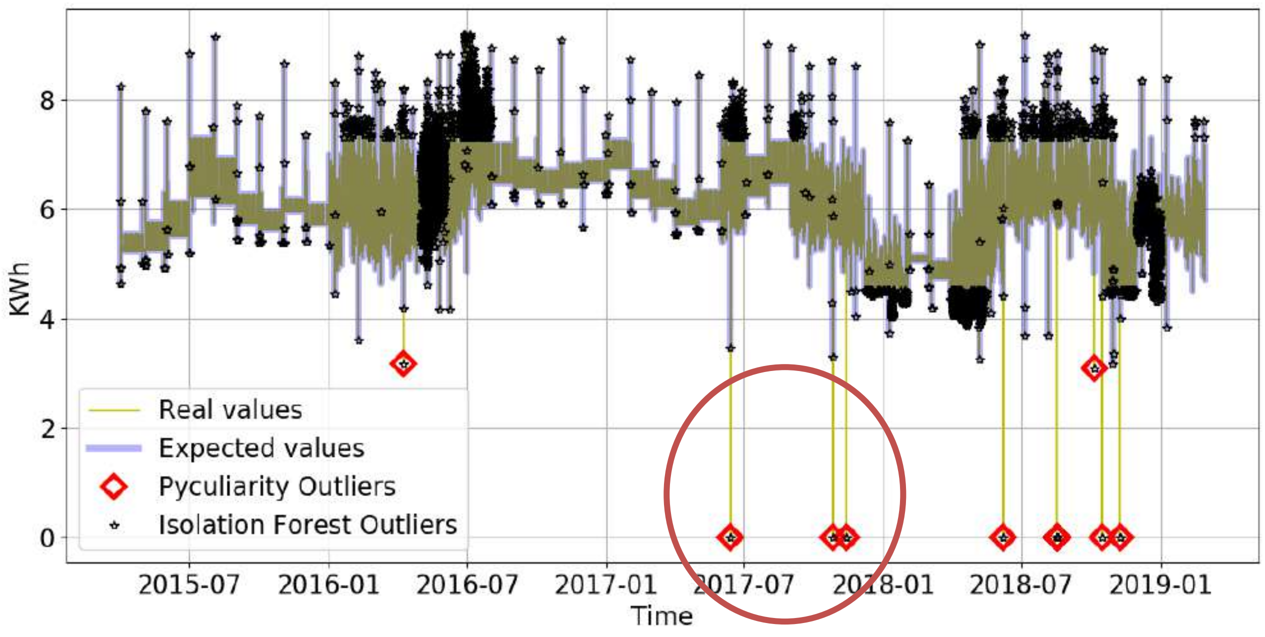
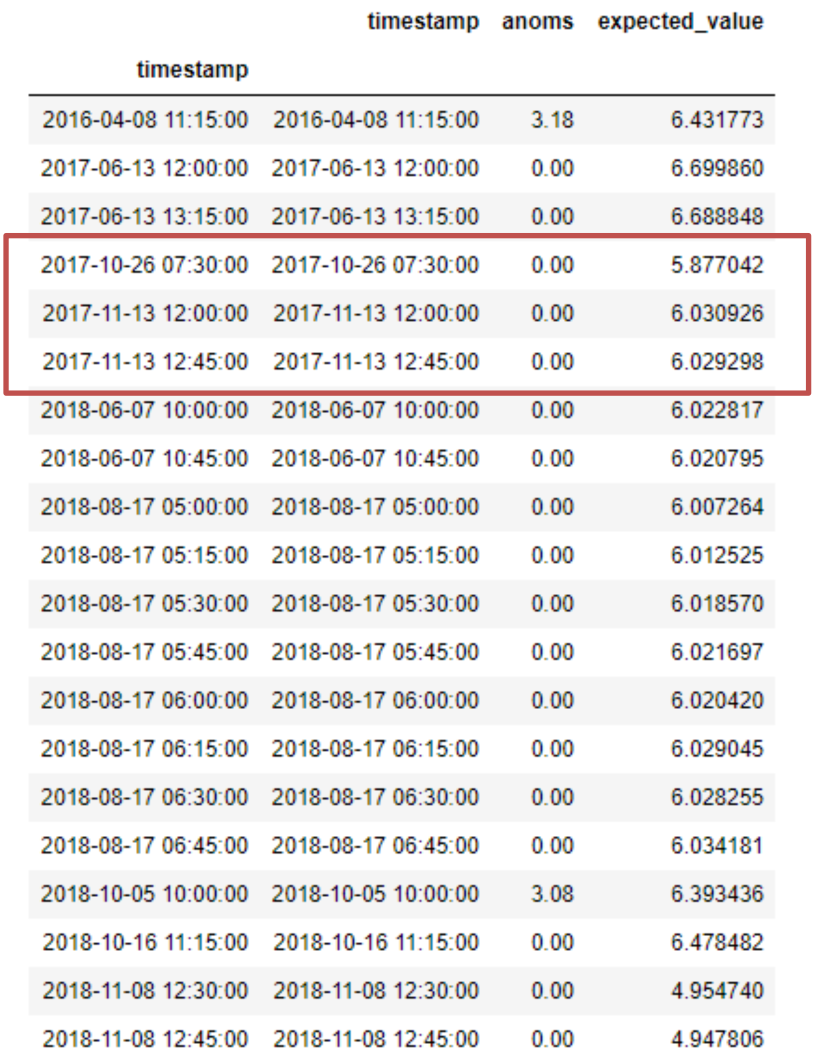


Figure 23: Outliers results

Figure 23 depicts the two mentioned methods, Pyculiarity in red squares and Isolation Forest in black points. Pyculiarity obtains less points because it considers the stationarity and seasonality, while Isolation Forest does not consider those two parameters; thus it provides many outliers. The proposed method for adjusting the outlier's values is illustrated in detail inside the red circle: these three points (yellow line) are considered as outliers by Pyculiarity and Isolation Forest so the values turned into the expected Pyculiarity values (light purple lines) and numerically represented in Figure 24.



timestamp	timestamp	anoms	expected_value
2016-04-08 11:15:00	2016-04-08 11:15:00	3.18	6.431773
2017-06-13 12:00:00	2017-06-13 12:00:00	0.00	6.699860
2017-06-13 13:15:00	2017-06-13 13:15:00	0.00	6.688848
2017-10-26 07:30:00	2017-10-26 07:30:00	0.00	5.877042
2017-11-13 12:00:00	2017-11-13 12:00:00	0.00	6.030926
2017-11-13 12:45:00	2017-11-13 12:45:00	0.00	6.029298
2018-06-07 10:00:00	2018-06-07 10:00:00	0.00	6.022817
2018-06-07 10:45:00	2018-06-07 10:45:00	0.00	6.020795
2018-08-17 05:00:00	2018-08-17 05:00:00	0.00	6.007264
2018-08-17 05:15:00	2018-08-17 05:15:00	0.00	6.012525
2018-08-17 05:30:00	2018-08-17 05:30:00	0.00	6.018570
2018-08-17 05:45:00	2018-08-17 05:45:00	0.00	6.021697
2018-08-17 06:00:00	2018-08-17 06:00:00	0.00	6.020420
2018-08-17 06:15:00	2018-08-17 06:15:00	0.00	6.029045
2018-08-17 06:30:00	2018-08-17 06:30:00	0.00	6.028255
2018-08-17 06:45:00	2018-08-17 06:45:00	0.00	6.034181
2018-10-05 10:00:00	2018-10-05 10:00:00	3.08	6.393436
2018-10-16 11:15:00	2018-10-16 11:15:00	0.00	6.478482
2018-11-08 12:30:00	2018-11-08 12:30:00	0.00	4.954740
2018-11-08 12:45:00	2018-11-08 12:45:00	0.00	4.947806

Figure 24: Screenshot of Pyculiarity expected values

2.5.5 Fill missing values

Sometimes the gathered data has gaps or is incomplete, as shown in Figure 25, due to multiple factors e.g. no connection with smart meter during a period, sensor malfunctions among others, which can lead to errors in estimations, distortion of the analysis or invalid conclusions. Therefore, the goal of this subtask is to fill these gaps carrying out a process of reconstruction.

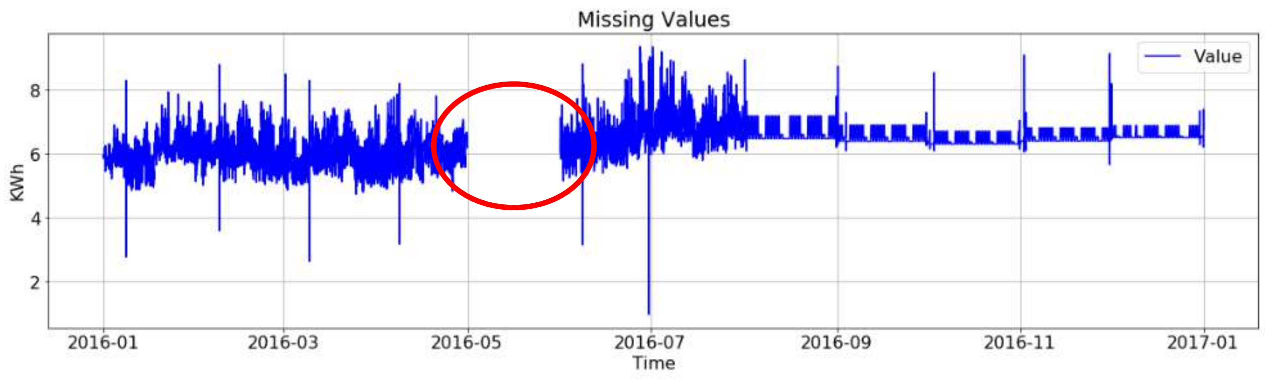
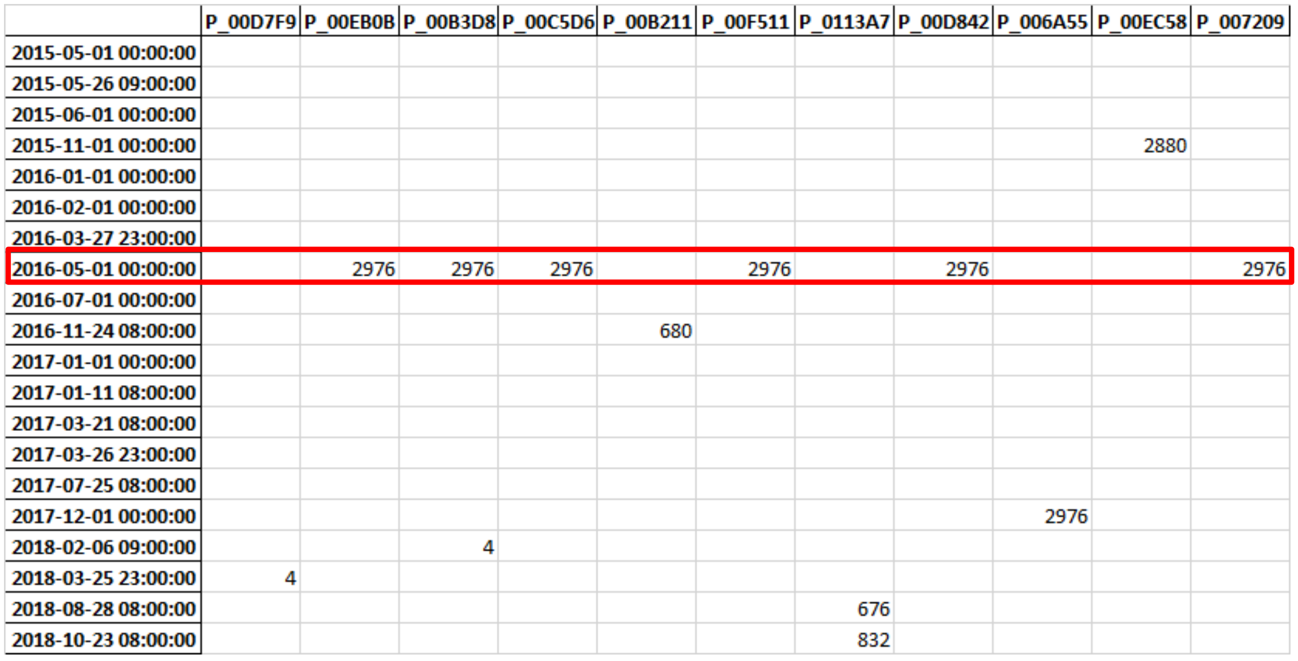


Figure 25: Sample of missing values in user dataset

The documentation in the field is so wide, with some techniques like replace missing data with an impossible value, data imputation (e.g. mean, median, mode, interpolation or imputation) or rolling. Considering the data is a date time, methods like mean or median are not applicable because they don´t take into account the seasonality or stationarity (El-Nesr, 2018).

Considering the previous paragraph, the procedure for filling the missing values is:

1. Create an excel file that collects the prosumers with missing values. This file provides, by columns, the prosumer name; by index, when its missing period starts, and the values represent how many consecutive missing values it has. Figure 26 shows a sample extract of this excel file.



	P_00D7F9	P_00EB0B	P_00B3D8	P_00C5D6	P_00B211	P_00F511	P_0113A7	P_00D842	P_006A55	P_00EC58	P_007209
2015-05-01 00:00:00											
2015-05-26 09:00:00											
2015-06-01 00:00:00											
2015-11-01 00:00:00										2880	
2016-01-01 00:00:00											
2016-02-01 00:00:00											
2016-03-27 23:00:00											
2016-05-01 00:00:00		2976	2976	2976		2976		2976			2976
2016-07-01 00:00:00											
2016-11-24 08:00:00					680						
2017-01-01 00:00:00											
2017-01-11 08:00:00											
2017-03-21 08:00:00											
2017-03-26 23:00:00											
2017-07-25 08:00:00											
2017-12-01 00:00:00									2976		
2018-02-06 09:00:00			4								
2018-03-25 23:00:00	4										
2018-08-28 08:00:00							676				
2018-10-23 08:00:00							832				

Figure 26: Part of the excel with the missing values in ASM "prelievo" dataset

2. Two techniques are applied depending on the number of missing values. Interpolation methods for missing values´ gaps equal or less than 96 points and the reconstruct method for gaps greater than 96 points (24 hours * 15minutes sampling = 96 records).

- Interpolation methods (between 0-96 missing values): for gaps up to 4 points the nearest interpolation method is used (pandas, 2014), while for more points (4 to 96 missing gaps) slinear (a first order spline linear interpolation) method is applied (pandas, 2014).

  Both methods have been selected after testing other ones as: cubic, spline, akima or polynomial (orders 3, 4 and 5) obtaining lower accuracy.

- Reconstruct method (more than 96 missing values): this method applies when these two conditions occur as shown in Figure 27:
  - 1) the prosumer should have a gap with more than 96 missing values and
  - 2) there must be equal or more than 15 prosumers with the same number of missing values for the same date time;

  As it can be seen in the partial excel, Figure 26, the two conditions appear in the remarked box, for all the prosumers with 2976 missing values at datetime 2016-05-01 00:00:00. This method works as follow for each prosumer:

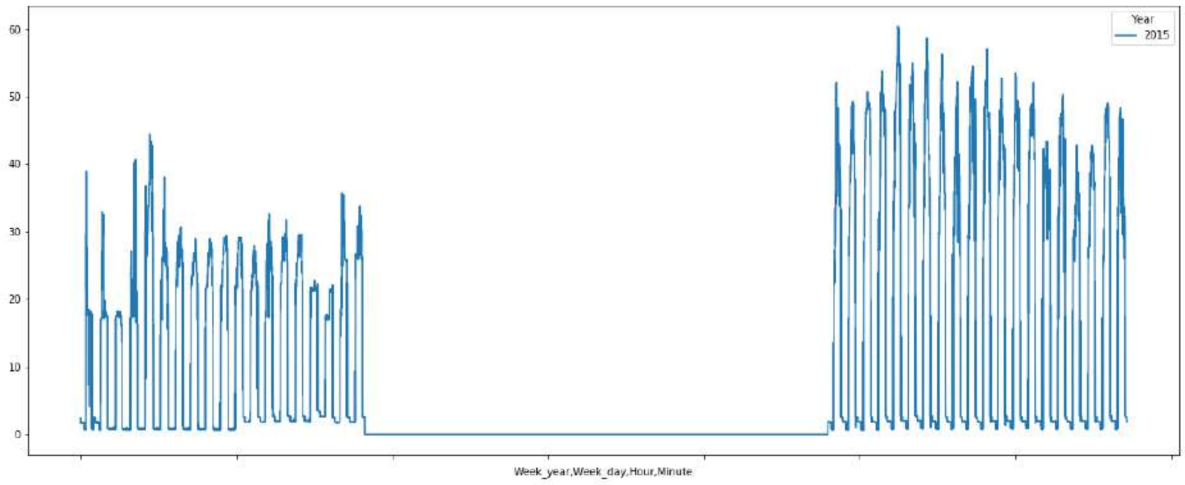


Figure 27: Dataset with one month missing values

  - First it calculates the trend of the missing values. The trend is determined by drawing a line (ascending or descending) between the average of the last full day and the next available full day (Figure 28). Understanding the average of the last full day as the calculation of the mean of the full day (24-hour average * 4 points / hour = 96 points) before the missing value period. If this day is not complete (96 points), the algorithm searches in previous days, until finding a complete one. The last full day of the trend should contain 96 points after missing values period.

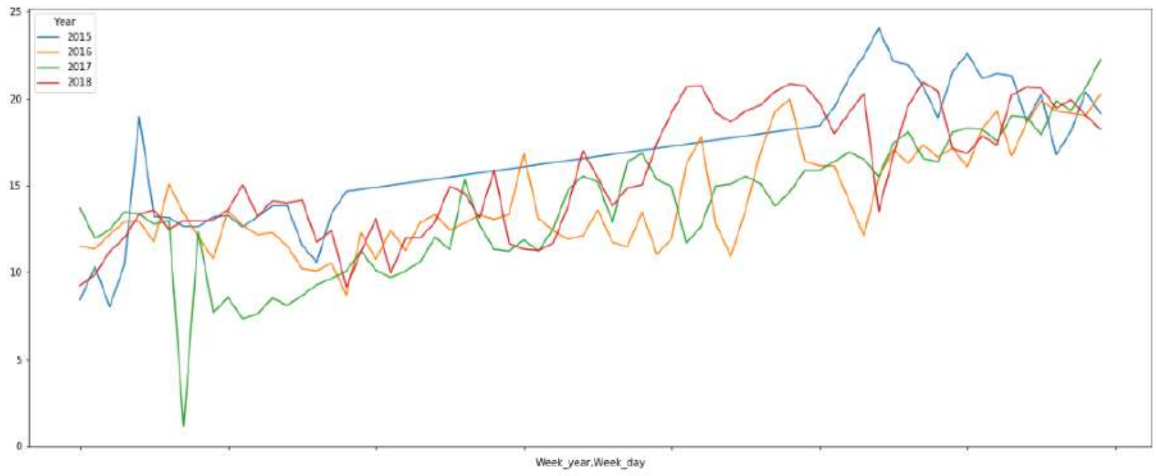


Figure 28: Reconstruct method

- Then, it collects same missing values period with the two previous and subsequent weeks in the remaining available years. Afterwards, the difference between the weeks available and the drawn line is calculated. The week with the smallest result, that is, the value closest to the line, will be selected. Finally, the difference between the selected week and the line is completed until reaching the latter. Results of the filled gap are shown in Figure 29.

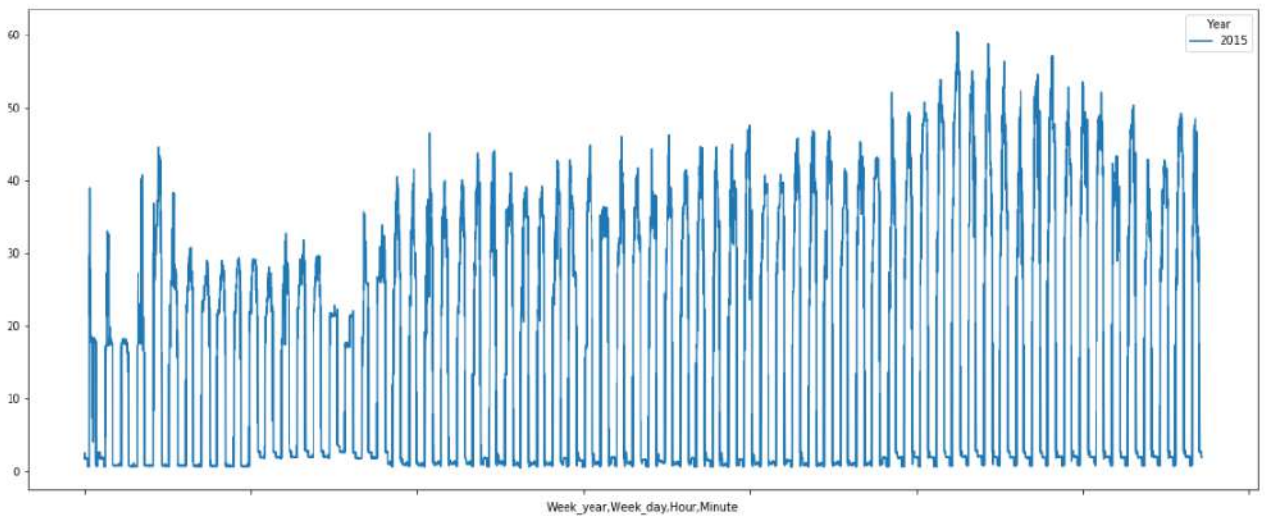


Figure 29: Dataset after applied reconstruct method

2.5.6 Normalization

One of the objectives of this deliverable is to cluster the users based on their consumption profiles and consumption patterns. Understanding, as consumption profiles, how much energy the users consume and as consumption patterns when users consume more (ups and downs along short periods of time).

To do so, the datasets collected (raw data) represent the user's consumption, so they must be treated, applying a normalization to obtain the consumption patterns and consumption profiles.

Minmax Scaler (Scikit learn, 2019) is the method which allows to make this treatment. It takes the dataset and scales the values in range [0-1]. This means, that the highest measurement of the dataset is transformed to 1 and the lowest to 0; the transformation is given by Equation 1 and calculated as Equation 2.

$$X_{std} = \frac{(X - X_{\min(axis=0)})}{(X_{\max(axis=0)} - X_{\min(axis=0)})}; \quad X_{scaled} = X_{std} * (max - min) + min$$

Equation 1: MinMaxScaler transformation

$$X_{scaled} = scale * X + \min - X_{\min(axis=0)} * scale$$

$$where\ \ scale = \frac{((\max - \min)}{(X_{\max(axis=0)} - X_{\min(axis=0)})}$$

Equation 2: MinMaxScaler calculations

Although standardization is considered as a pre-processing method, in this case, it will be applied as a previous step to the clustering algorithms since the pattern or profile portfolio of the dataset is given by the selected option in Load Profiling module.

3 Load Profiling

Load Profiling is the first module of Big Data Layer which aims to categorize the prosumers according to a series of features. It ingests the data coming from the pre-processing module or Electricity Consumption and Production Forecasting module, afterwards categorize the dataset and finally it ingests them in Big Data Clustering at Multiple Scale module.

The considered features have been selected from biography (Yassine, 2018), (Torabi, Hashemi, Saybani, Shamshirband, & Mosavi, 2018), previous deliverable 4.2 and the knowledge field.

Following

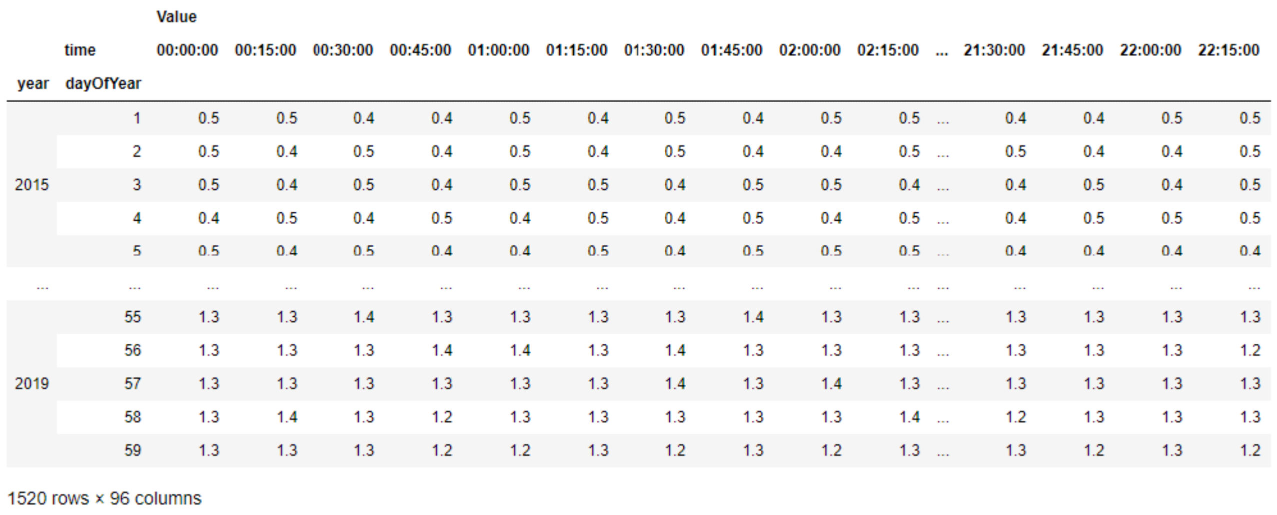
Table 2 exposes the selected features:

Option	Feature description	Granularity data	Length of data (number of values)
1	Daily	15 min	96
2	Daily working days (Mon – Fri)	15 min (mean)	480
3	Daily weekend days (Sat – Sun)	15 min (mean)	192
4	Weekly on hourly base (Mon – Sun)	1h (mean)	168
5	Weekly on daily base (Mon – Sun)	Daily (mean)	7
6	Summer months on daily base (Jun-Jul-Aug)	Daily (mean)	92
7	Fall months on daily base (Sep-Oct-Nov)	Daily (mean)	91
8	Winter months on daily basis (Dec-Jan-Feb)	Daily (mean)	90
9	Spring months on daily basis (Mar-Apr-May)	Daily (mean)	92
10	Flexibility working days on daily base	1h	672
11	Monthly Flexibility	1h	1

Table 2: Features of load profiling module

This module allows selecting an option, among those available in Table 2, according to which the clustering will be carried out. It works as follows:

1. If the selected option is among 1-9, Load Profiling receives the pre-processed data from the historical dataset of ASM, but when option 10 or 11 are selected, data are provided by "Electricity Consumption and Production Forecasting" module. This module calculates the baseline and flexibility of the ASM customers; thus, it is possible to profile the day-ahead flexibility with a 1-hour granularity.
2. Below paragraphs describe how the dataset is treated based on the option:
    a. Option 1: the final dataset is the concatenation of all days of every user, that allows to obtain a dataset with more than 450.000 rows. Figure 30 depicts partially one user dataset when option 1 is selected.

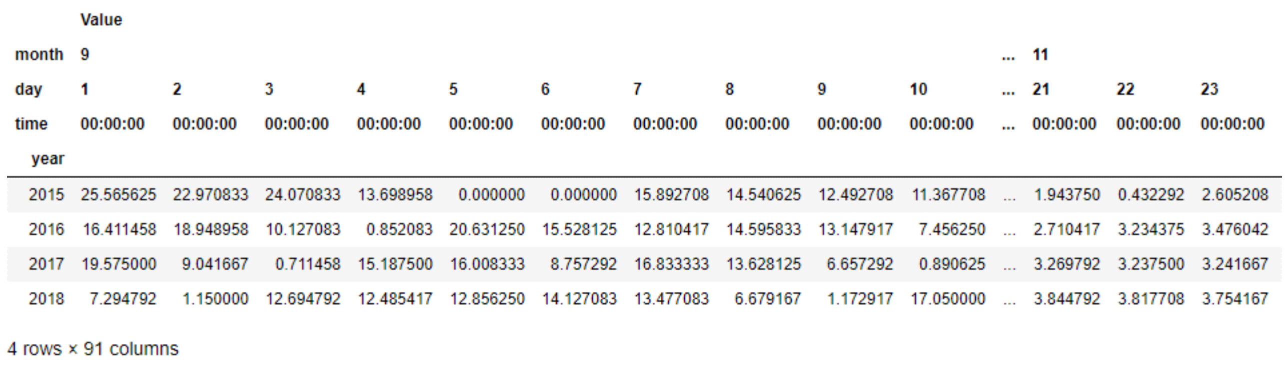


		Value														
	time	00:00:00	00:15:00	00:30:00	00:45:00	01:00:00	01:15:00	01:30:00	01:45:00	02:00:00	02:15:00	...	21:30:00	21:45:00	22:00:00	22:15:00
year	dayOfYear															
2015	1	0.5	0.5	0.4	0.4	0.5	0.4	0.5	0.4	0.5	0.5	...	0.4	0.4	0.5	0.5
	2	0.5	0.4	0.5	0.4	0.5	0.4	0.5	0.4	0.4	0.5	...	0.5	0.4	0.4	0.5
	3	0.5	0.4	0.5	0.4	0.5	0.5	0.4	0.5	0.5	0.4	...	0.4	0.5	0.4	0.5
	4	0.4	0.5	0.4	0.5	0.4	0.5	0.4	0.5	0.4	0.5	...	0.4	0.5	0.5	0.5
	5	0.5	0.4	0.5	0.4	0.4	0.5	0.4	0.5	0.5	0.5	...	0.4	0.4	0.4	0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2019	55	1.3	1.3	1.4	1.3	1.3	1.3	1.3	1.4	1.3	1.3	...	1.3	1.3	1.3	1.3
	56	1.3	1.3	1.3	1.4	1.4	1.3	1.4	1.3	1.3	1.3	...	1.3	1.3	1.3	1.2
	57	1.3	1.3	1.3	1.3	1.3	1.3	1.4	1.3	1.4	1.3	...	1.3	1.3	1.3	1.3
	58	1.3	1.4	1.3	1.2	1.3	1.3	1.3	1.3	1.3	1.4	...	1.2	1.3	1.3	1.3
	59	1.3	1.3	1.3	1.2	1.2	1.3	1.2	1.3	1.2	1.3	...	1.3	1.2	1.3	1.2

1520 rows × 96 columns

Figure 30: Dataset obtained from option 1 for one user

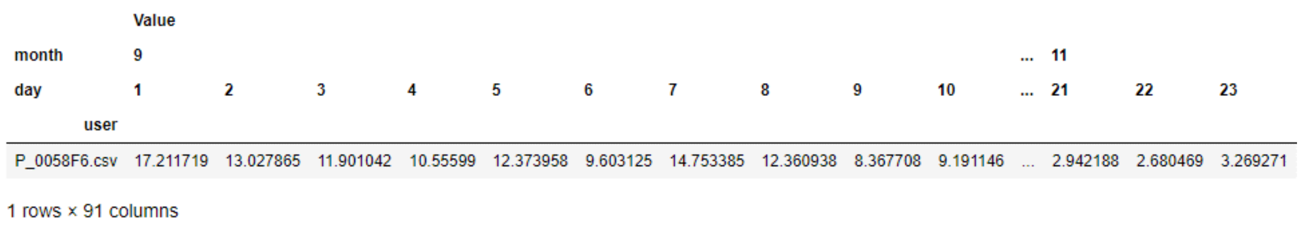
b. <u>In options 2 to 9</u>: these files calculate the mean in accordance with the option description. This enables to get a single row per user, with a total dataset of 348 rows, the same number of users obtained from the pre-processed module. Figure 31 shows how the data per user is organized based on the selected option; while Figure 32 illustrates the mean used to form the final dataset file.



	Value										...			
month	9										...	11		
day	1	2	3	4	5	6	7	8	9	10	...	21	22	23
time	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	...	00:00:00	00:00:00	00:00:00
year														
2015	25.565625	22.970833	24.070833	13.698958	0.000000	0.000000	15.892708	14.540625	12.492708	11.367708	...	1.943750	0.432292	2.605208
2016	16.411458	18.948958	10.127083	0.852083	20.631250	15.528125	12.810417	14.595833	13.147917	7.456250	...	2.710417	3.234375	3.476042
2017	19.575000	9.041667	0.711458	15.187500	16.008333	8.757292	16.833333	13.628125	6.657292	0.890625	...	3.269792	3.237500	3.241667
2018	7.294792	1.150000	12.694792	12.485417	12.856250	14.127083	13.477083	6.679167	1.172917	17.050000	...	3.844792	3.817708	3.754167

4 rows × 91 columns

Figure 31: User´s dataset organizes based on option 7



	Value										...			
month	9										...	11		
day	1	2	3	4	5	6	7	8	9	10	...	21	22	23
user														
P_0058F6.csv	17.211719	13.027865	11.901042	10.55599	12.373958	9.603125	14.753385	12.360938	8.367708	9.191146	...	2.942188	2.680469	3.269271

1 rows × 91 columns

Figure 32: Final row of the mean option 7

c. <u>Options 10 and 11</u>: these options validate the use case 3.1, where the Load Profiling module asks the flexibility profiles to Electricity Consumption and Production Forecasting module. Thanks to these two options it is possible to extract the profiles of flexibility from customers, therefore the clustering tool will be able to group the users not only for the energy they consume, but also for the flexibility they can potentially provide. This way allows to achieve a more granular and accurate response, than adopting the features from section 5.1 of Deliverable D4.2, which is estimating the flexibility indirectly from common consumption data. The "Electricity Consumption and Production Forecasting module calculates and exposes the values of flexibility (thanks to external tool for Baseline Estimation) for each one of the customers of the Italian pilot site. The requests are done through an API Rest (See D3.1 "Electricity production/consumption forecasting techniques and tool V1, section 4.3), as shown in Figure 33, customizing the parameters in accordance to the option.

$$"http://95.211.2.240{:}1777/db-api-mysql-0.0.1/predictions/\{flexibility\_type\ or\ baseline\}/$$

$$dayahead/\{user\_id\}/\{init\_time\}/\{end\_time\}"¶$$

Figure 33: Screenshot of the API Rest for flexibility request

The output of the module is, in all cases, a dataset with a different number of rows and columns; an element that directly affects the selection of the clustering algorithm and the treatment required, as will be seen in the next chapter.

As mentioned in paragraph 2.5.6, once the final dataset is obtained a normalization step is mandatory in order to obtain the consumption patterns and consumption profiles:

- Options 1-5 (daily and weekly options) are normalized with minmax scaled by row in order to cluster the pattern consumption of the profiles. The resulted clusters are independently of how much energy they consume, visualizing when they consume along the period time (ups and downs).
- Options 6-9 (monthly options) are scaled by row. In these cases, the clustering considers how much energy the users consume to face the flexibility and demand response modules.

4 Big Data Clustering at Multiple Scale

Big Data Clustering at Multiple Scale is the core module of Big Data Layer and therefore of task 4.2. This analytic tool offers the valuable information on the clustering process to the operators enabling them to characterize their portfolio; since it helps in decision-making for flexibility management and assesses the participation of prosumers in the electricity market.

According to deliverable 4.2 chapter 5 (p. 33) “The overall clusterization process is divided into five steps, as Figure 34 illustrates. The pre-processing step, filtering and cleaning the initially collected data, becomes essential for clustering algorithms (revise chapter 2.4 for more details). The second step deals with attribute selections, where the proper feature must be defined to extract valuable information from clusterization. In the third step, one out of the three developed clustering algorithms must be chosen according to pre-requisites. After the calculation, obtained results must be validated through different metrics. Finally, when information is already available and validated, the operators can interpret the results and extract the knowledge of the portfolio”.

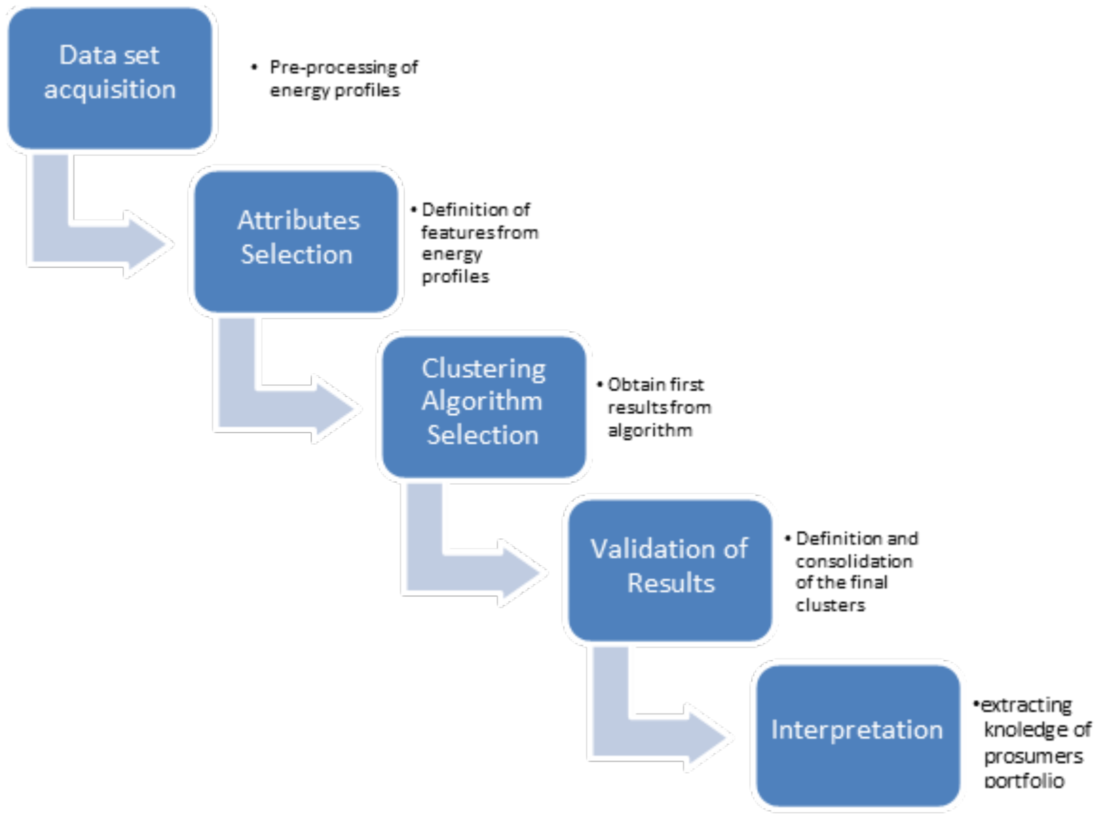

Data set acquisition
• Pre-processing of energy profiles
Attributes Selection
• Definition of features from energy profiles
Clustering Algorithm Selection
• Obtain first results from algorithm
Validation of Results
• Definition and consolidation of the final clusters
Interpretation
•extracting knoledge of prosumers portfolio

Figure 34: Steps of clusterization procedure (source Atos based on (Halkidi, 2001))

4.1 Selected clustering algorithms

The algorithms, which finally have been selected, considering those describes in Deliverable 4.2, section 5.2, are K-means, DBSCAN and Deep Embedded Clustering (Autoencoder+ K-means) due to the following reasons:

- Both K-means and DBSCAN need only set one parameter to work. Both are scalable and perform well for a large number of examples and a medium number of clusters and employ a (metric) geometry of point distances (Scikit-learn, 2019).
- Deep Embedded Clustering (Autoencoder+ K-means): this algorithm, although must set many parameters as hidden layers, batch size, epochs or number of clusters, allows to reduce the dimensionality and noise, keeping most of the information into the decoder layers. Another reason why this algorithm has been selected is that its training is automatic with sample data, maintaining good performance in similar types of input, without the need to generalize.

The following paragraphs describe the algorithms, how they work, and which parameters are required.

4.2 Pre-requisites

The goal of this section is to define in detail the initial parameters required by the three algorithms in the tool. K-means and Autoencoder need to be initialized with the number of clusters, so different indices are considered to find the optimal number.

DBScan needs no number of clusters, but a couple of parameters for the initialization hereby described.

4.2.1 Pre-requisites for K-Means and Autoencoder

Both K-means and Deep Embedded Clustering (Autoencoder+ K-means) techniques requires the following two actions before being executed:

- **Size of data:** unlike K-means or DBSCAN algorithms that can achieve good results with not many records, Autoencoder requires a large amount of records to train. Considering scalability requirements described in Section 2, Autoencoder provides the necessary capacity to manage big data, but, on the other hand, it is not precise with a small dataset.

  In particular, after tests conducted for ASM pilot site, with a limited number of customers in the dataset and the amount of data per customer, this algorithm can only be used when "option 1" in Table 2 is selected; since it offers this large amount of data set, being more than half a million. This value, which has been empirically selected, has a threshold for the activation of autoencoder; for dataset lesser than half million points K-means and DBScan show better performances.

- **Set the optimal number of clusters (*k*):** set off this parameter is one crucial issue for these clustering algorithms, since it depends on the method used. Two types of methods are presented:
  - **Elbow method:** (Kassambara, 2018) that tries to optimize the sum of squared distances between each observation and its closest centroid. This method is performed with the Euclidean distance.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Equation 3: Euclidean distance

  - **Gap statistic**: (Kassambara, 2018) which compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data. The optimal value is the one that maximized the gap statistic.

$$Gap(k) \geq Gap(k+1) - s_{k+1}$$

Equation 4: Gap Statistic metric

In case above methods give different results, two more scores are applied to achieve the best optimal number of clusters. (Scikit learn, 2019)

  - **Silhouette coefficient**: calculates the mean ratio of intra-cluster and nearest-cluster distance. It presents an advantage, since the separation distance between the resulting clusters is in a range of [-1,1]; being "-1" when the distances between clusters are near or even overlapped and "1" when the distances between centroids are so distance.

$$Silhouette\ coefficient = \frac{(b-a)}{\max(a,b)}$$
$$a = mean\ intra - cluster\ distance; \quad b = mean\ nearest - cluster\ distance$$

Equation 5: Silhouette coefficient

- **Calinski-Harabasz:** or variance ratio criterion, is the ratio between the within-cluster dispersion and the between-cluster dispersion. This score presents a better grouping the higher value obtained (Liu, 2015)

$$CH = \frac{\sum_{i=0}^{K} |C_i| \times \frac{\|\mu_i - \mu\|^2}{K-1}}{\sum_{k=1}^{K} \frac{\sum_{i=0}^{|C_i|} \|\mu_i - \mu\|^2}{N-1}} \rightarrow \frac{SS_B}{SS_W} \times \frac{N-k}{k-1}$$

Equation 6: Calinski Harabasz distance

k: represents the number of clusters; N means the total number of observations (data points); $\mathbf{SS_W}$ is the overall within-cluster variance (equivalent to the total within sum of squares); $\mathbf{SS_B}$ is the overall between-cluster variance.

Considering above requisites, the actions keep up to achieve them are the following:

- **Elbow method**

Yellowbrick library (Yellowbrick, 2019) provides one module for clustering which not only offers the visualization, but also provides the optimal number of clusters automatically for the elbow method.

Elbow method has to set a tuple of the possible optimal number of clusters. This range is set as 2-10 and the algorithm where they are tried is K-means. So, iteratively it calculates the metric for each number in the range and provide their best scores, numerically and visually.

- **Gap statistic**

Gap statistic metric (Robert Tibshirani, 2000) is also tried using K-means algorithm and requires setting a range (2-10), but unlike the above metric, it does not allow obtaining the optimal number of clusters automatically.

In this case, Figure 35 is represented with the obtained values and the elbow must be located visually.

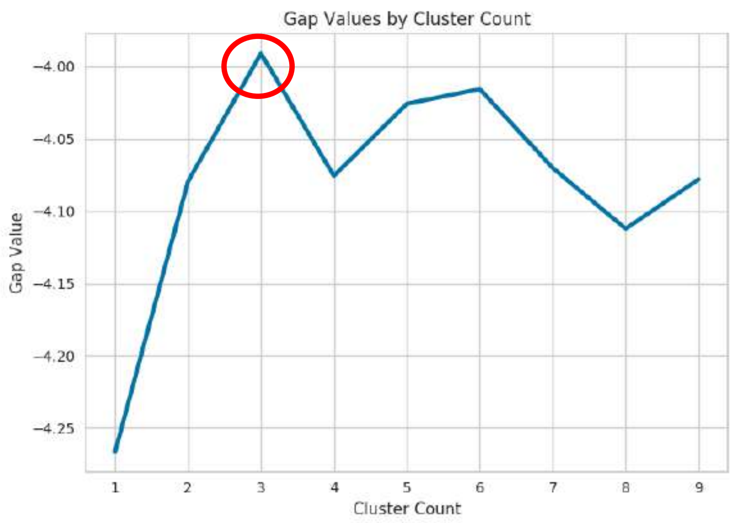


Figure 35: Gap Statistic for option 3 “Daily weekend days (Sat – Sun)”

Following Table 3 gathers the results from the above methods for all the option of Load Profiling module. Each option presents two types of dataset from which the number of clusters is obtained:

- Normalized dataset: comes directly from Load Profiling module where the data has been organized according to the selected option and applied the normalization described in paragraph 2.5.6 to obtain consumption patterns (option 1-5) and consumption profiles (option 6-9). These datasets get theirs number of clusters depicted in the first line of every option. K-means algorithm, used to calculate the number of clusters, has the disadvantage that from a certain number of dimensions (length of data) the number of obtained clusters is constant. A possible solution is to reduce the dimensionality applying the PCA (Principal Component Analysis) algorithm; which converts each row of dataset with many columns into a given number of principal components previously defined; here PCA is set with two components
- PCA dataset: this type of dataset is the result of applying the PCA algorithm to the normalized dataset; it allows to check if the disadvantage of K-means influences the calculation of the number of clusters.

As presented in Table 3, both types of data set are applied to Elbow and Gap Statistic methods. This last one exposes the dimensionality problem, since it does not converge for normalized dataset. This problem is also shown in option 1 where a large number of data does not accomplish any result. In options 1, 6, 7, 8, and 9 the dimensionality problem does not appear since they accomplish the same number of clusters both methods and data types. Additionally, Silhouette metric is performed in order how well the clustering is working on each dataset type.

On the contrary, in the rest of the options where the clusters are different, Silhouette method is applied first (greater number, better result) and in case of the same results, Calinski-Harabasz method is calculated (greater number, better number of clusters).

For instance, option 4 achieves five and four clusters for Elbow method with normalized and PCA data respectively, and three clusters for PCA data with Gap statistic. In this case, the calculation of Silhouette metric does not enable the selection of the best number due to Elbow and Gap Statistic method for PCA data get the same value, 0.46. So that Calinski-Harabasz metric is calculated, obtaining a better result for four number of clusters. These steps allow getting the optimal number of clusters for all option.

Option	Data Type	Elbow	Gap Statistic	Silhouette Coefficient		Calinski-Harabasz		Optimal number of clusters
				Elbow	Gap Statistic	Elbow	Gap Statistic	
1	Normalized	4	-	0.2				4
	PCA	4	-	0.4				
2	Normalized	5	-	0.22				5
	PCA	5	4	0.402	0.397	447.37	430.025	
3	Normalized	4	-	0.27	-		-	3
	PCA	4	3	0.44	0.46	429.498	442.587	
4	Normalized	5	-	0.25	-		-	4
	PCA	4	3	0.46	0.46	504.268	466.079	
5	Normalized	4		0.40				3

	PCA	4	3	0.5	0.5	577.18	585.704	
6	Normalized	3	-	0.88				3
	PCA	3	3					
7	Normalized	4		0.67				4
	PCA	4	4	0.69				
8	Normalized	4	-	0.67				4
	PCA	4	4	0.7				
9	Normalized	4	-	0.85	-			4
	PCA	4	4	0.86				
10	Normalized	4	-	0.756				3
	PCA	4	3	0.756	0.836			
11	Normalized	5	-	0.774				4
	PCA	4	4	0.786				

Table 3: Optimal number of clusters for k-means and autoencoder algorithm

4.2.2 Pre-requisites for DBScan

On the other hand, DBSCAN requires these two parameters to set (Prado, 2017):

- **minPoints:** the minimum number of neighbours that a given point should have in order to be identifies as core point. The general rule is to set:

$$minPoints \geq D + 1 \quad where\ D = number\ of\ dimensions$$

Equation 7: minPoints parameter for DBSCAN algorithm

As was mentioned in D4.2, "considering the parameter minPoints, a general rule is that it can be derived from the number of dimensions (D) in the dataset, as minPoints ≥ D+1. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for minPoints must be 3, but as larger the dataset is, the larger the value of minPoints should be" (Prado, 2017).

- **eps:** two points are considered as neighbours, if the distance between them is below the threshold epsilon. The idea of the epsilon parameter is to calculate the average distances of every point to its k nearest neighbors. Eps is chosen based on the distance using a k-distance graph. (Ivan, 2020)

$$N(p) = \{q\ e\ D | dist(p, q) \leq Eps\}$$

Equation 8: Epsilon neighbourhood of a point

If the epsilon value is too small, the largest part of the dataset will be not clustered. On the other hand, if the value is too high, clusters will merge, and most of the data points will end up being appointed to the same class. The decision of eps value should be based on the distance of dataset (k-distance graph could be used), but in general small eps values are preferable. For that, the k-nearest neighbors´ algorithm is used to calculate the distance from each point to its k closest neighbors. The value of k will be specified by the user and corresponds to MinPoints.

These k-distances are plotted in ascending order in order to determine the 'knee', which corresponds to the optimal eps parameter. A knee corresponds to a threshold, where a sharp change occurs along the k-distance curve. Figure 36 below shows an example of k-distance curve for option 9, where the optimal value of the epsilon parameter is close to 2.

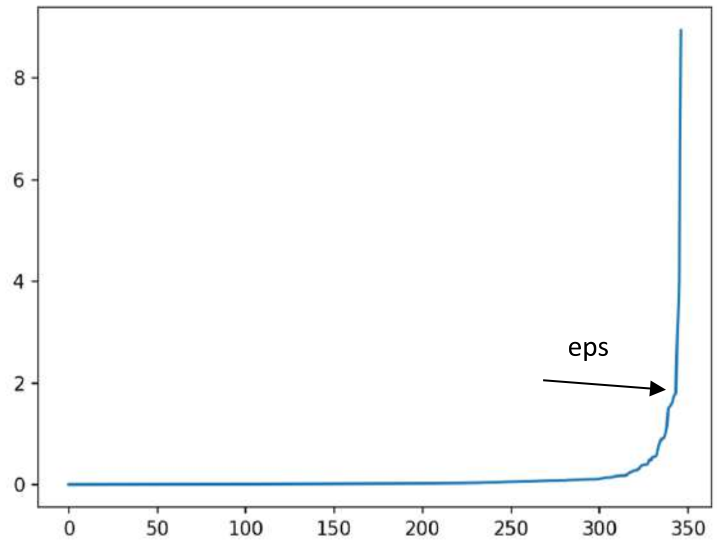


Figure 36: k-distance graph

Table 4 shows the optimal number of clusters for every option of Load Profiling module, besides the parameters required for the DBSCAN algorithm: epsilon and the minimum number of neighbours, minPoints, according the Equation 7 and Silhouette score. On this occasion, only option 2 applies the reduction of dimensionality, with value two, to the normalized data set.

Option	Eps	minPoints	Optimal number of clusters	Silhouette Coefficient
2	3.5	6	5	0.327
3	5	193	2	0.345
4	4.5	169	2	0.400
5	0.6	8	3	0.547
6	2	93	2	0.948
7	2	92	2	0.934
8	1.5	91	2	0.907
9	1.4	93	2	0.948

Table 4: Optimal number of clusters for DBSCAN algorithm

The objective of this Table 5 is to select the best number of clusters for each option of Load Profiling (apart from option#1, that will be clustered through Autoencoder), based on the results accomplish with K-Means and DBScan algorithms. It shows the results with the Silhouette index commonly used for comparing both algorithms.

	K-Means		DBScan	
Option	Optimal number of clusters	Silhouette Coefficient	Optimal number of clusters	Silhouette Coefficient
2	5	0.402	5	0.327
3	3	0.44	2	0.345
4	4	0.46	2	0.400
5	3	0.5	3	0.547
6	3	0.88	2	0.948
7	4	0.69	2	0.934
8	4	0.7	2	0.907
9	4	0.86	2	0.948

Table 5: Comparison of Silhouette coefficient between the number of clusters of K-Means and DBScan

From this table it is possible to see the different performances of both K-means and DBScan for the optimal number of clusters. Options 2 and 5 calculate the same number of clusters, while the rest show different numbers; the Silhouette index can be used for evaluating the quality of this pre-requisite. It seems that K-means shows better results for options 3 and 6, while DBScan for the others, but it is important to remark that K-means is always showing a higher level of partitioning compared with DBScan, that is less sensitive to outliers (e.g. it tends to aggregate more) and it implicitly depends from Eps.

4.3 Clustering algorithms and results

This section provides a deeper description of chosen algorithms; describing deeper how they perform; which parameters must be set to their proper operation and the architecture or dataflow. Also, it is presented and discussed the different results of clustering obtained in section 4.2.2 above.

4.3.1 Autoencoder

Autoencoder is a deep learning technique based on artificial neural network. It has been adopted in Big Data Clustering at Multiple Scales as an alternative to a generic-purpose clustering algorithm (K-means) and providing scalability to the service.

Improved Deep Embedded Clustering

As stated in (Yin, 2017), the traditional approach to apply deep learning techniques in the clustering tasks is to create a clustering loss layer. Despite the good results obtained by this method further analysis have revealed that there is an important drawback on measuring this clustering loss, and it is that the latent space variable which contains the essential structure of the original dataset. The solution proposed is to keep training to preserve the latent space variable structure while the clustering layer is trained to reduce the clustering loss.

Neural Network Architecture and Topology

The core of this clustering solution is based on an autoencoder. Autoencoders are neural networks specialized in extracting the latent space variable for an input and, afterwards, reconstructing the expected output. They are widely used and present the state-of-the-art results for very different projects: denoising, super resolution, dimensionality reduction, anomaly detection etc. In the environment of the eDREAM's project the autoencoder will be pretrained with the load profiles of different users, it shall compress the input data to extract the latent variables and reconstruct from those latent variables the original input. Consequently, it produces two outputs the encoded vector (Latent space array resulting from encoding the original input) and a decoded vector (Reconstructed array that should be similar to the original input).

After this pretraining phase, the autoencoder has already adjusted their weights to perform a compression and decompression on the input vector minimizing the loss and keeping the latent space structure. At this point the custom clustering layer is added to the model as it is shown in the scheme from Figure 37:

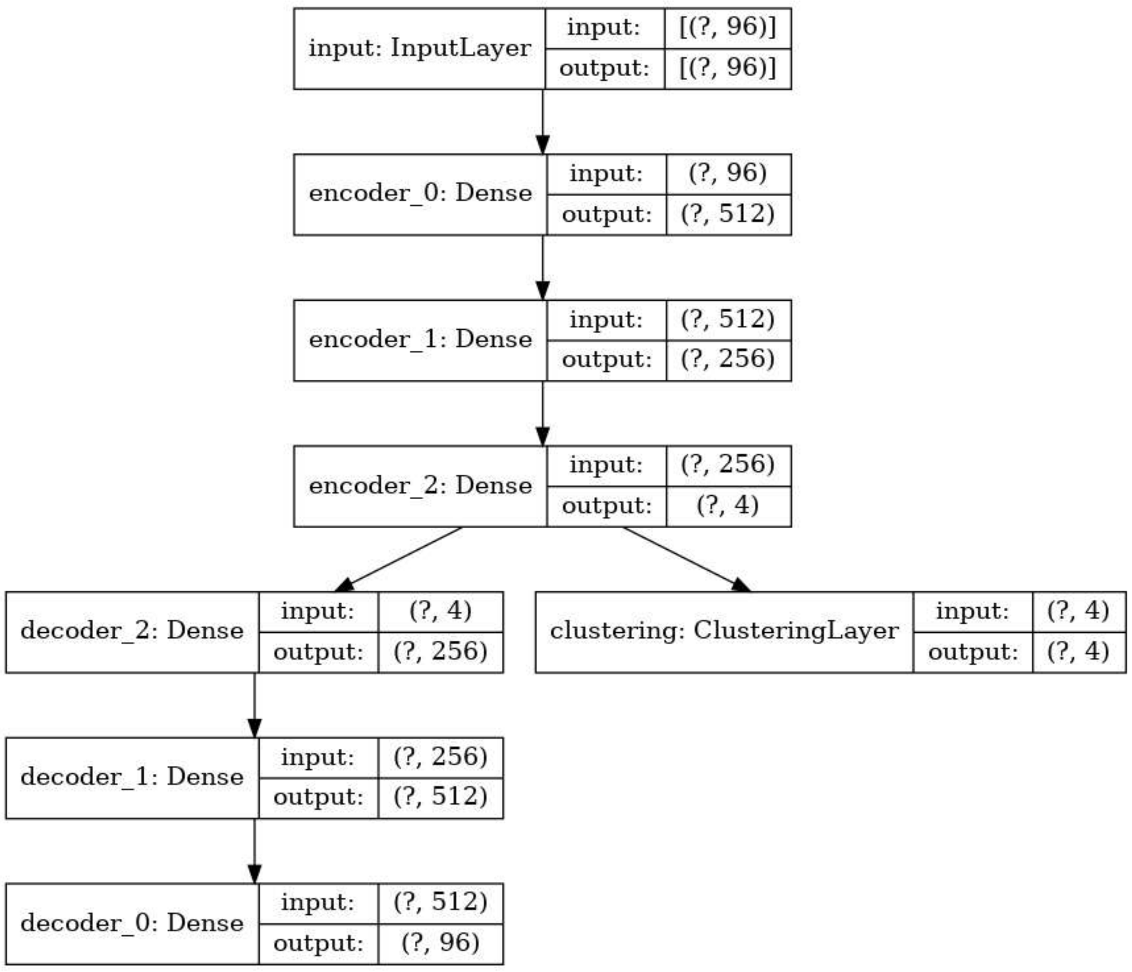


Figure 37: Autoencoder topology

As both, the clustering layer and autoencoder, are trained in parallel the latent space structure shall be preserved while the clustering loss shall periodically decrease.

This algorithm has outperformed k-means or other for high dimensional arrays as it can reduce the dimensionality of the problem without losing as much information as the traditional algorithms usually do.

Setting parameters

Autoencoder algorithm is performed with Tensorflow 2.0 (TensorFlow, 2020) and the parameters which must be set, are described the in following Table 6

Table 6

Parameter	Set with	Description
dims [1*, 2*, 3*, 4*]	[length_data, 512, 256, n_clusters]	Dims is an array which contains the required parameter for the autoencoder (encoder-decoder) and K-means algorithm
1*= input data shape	length_data	Length of data (number of values) per row according to the selected option (See section 3, Table 1)
2*= first hidden layer	512	Number of nodes in the first hidden layer
3*= second hidden layer	300	Number of nodes in the second hidden layer
4*= number of clusters	1e-4	Number of clusters for K-Means algorithm

Table 6: Setting parameters in Autoencoder algorithm

Results

This option 1, "Daily every 15 min", shows four optimal numbers of clusters (see Table 2) as shown in Figure 38. Clusters 0, 2, and 3 have consumption throughout the day, in different ranges; while cluster 1 maintains a constant consumption, decreasing slightly in the middle of the day and subsequently increasing after 19:00.

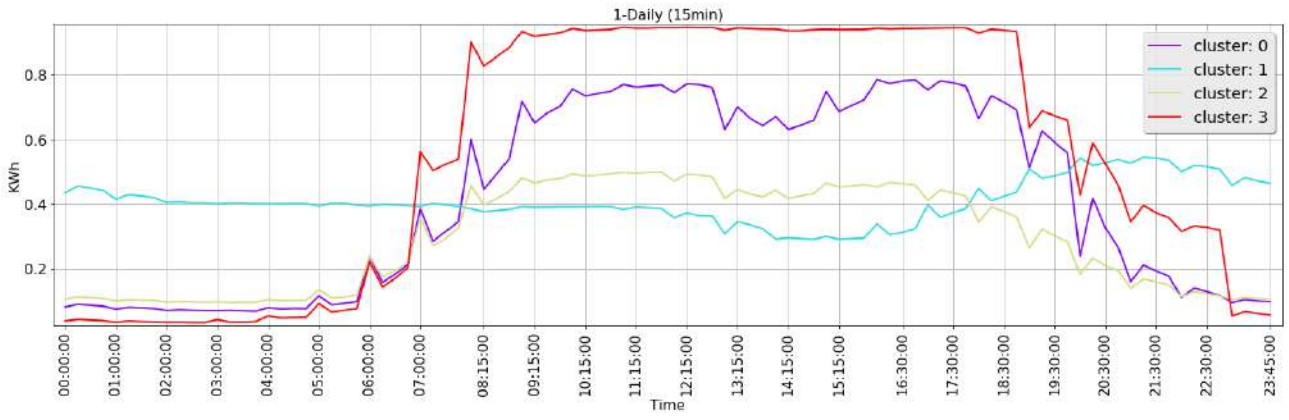


Figure 38: Result of option 1 "Daily"

4.3.2 K-means

K-means clustering (Garbade, 2018) (Munnelly, 2017) is an unsupervised learning algorithm, which makes inferences from the given dataset, using only input vectors. Its objective is to group elements that share similar characteristics and separate them from those that do not have these characteristics.

Without entering much detail, K-means works as follows: prefixed the number of clusters (k) it randomly selects k centroids, as starting centroids, in the datasets. Every point, considering point as each user time series array, is allocated to a cluster based on its nearest centroid. Iteratively, the centroids are recomputed as the mean of all points assigned to their cluster until the recomputed performance reaches the optimal (i.e. centroids have stabilized and points no longer switch to different clusters). (Munnelly, 2017)

Some advantage of using K-means algorithm are (Google Developers, 2020):

- Scikit learn (Scikit learn, 2019) library includes k-means++ as method for initialization which tries to choose good starting clusters, theoretically yielding better results;
- Scales to large data sets;
- Guarantees convergence;

However, K-means presents disadvantages (Google Developers, 2020):

- due to the initial centroids are randomly selected they can provide different results each time it is executed because its operation is probabilistic;
- number of clusters (k) must be given;
- clustering outliers: centroids can be dragged by outliers, or outliers might get their own cluster instead of being ignored; reason for the importance of pre-processing;

- Scaling with the number of dimensions: As the number of dimensions rises, a distance-based similarity measure converges to a constant value between any given examples. PCA can be a good algorithm for reducing dimensionality (Google Developers, 2020)

Setting parameters

K-means algorithm is provided by Scikit-learn library (Scikit-learn, 2019) which parameters are set as shown in Table 7.

Parameter	Set with	Description
clusters	options of section 4.2.2	The number of clusters to form as well as the number of centroids to generate.
Init	'k-means++'	Method for initialization. 'k-means++': selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
n_init	10	Number of time the k-means algorithm will be run with different centroid seeds. The results will be the best output of n_init consecutive runs in terms of inertia.
max_iter	300	Maximum number of iterations of the k-means algorithm for a single run
tol	1e-4	Relative tolerance with regards to inertia to declare convergence.
precompute_distances	'auto'.	Precompute distances (faster but takes more memory).
random_state	None	Determines random number generation for centroid initialization. Use the global random state from numpy.random

Table 7: Setting parameters in K-means algorithm

Results

The results of the optimal number of cluster with K-means algorithm for options from #2 to #9 of load profiling (Table 2) are represented in the images below. Figures 39 and 40 correspond respectively to options #2 and #3 for daily based profiles, figures 41 and 42 correspond to options #4 and #5 for weekly-based profiles and finally figures from 43 to 45 correspond respectively to option from #6 to #9 for seasonal profiles. Due to the representation of all the users by cluster would be difficult to visualize, it has been chosen to illustrate the average of every cluster, which basically corresponds to its cluster centroid in the considered timespan for each option.

Results for option 10 cannot be represented by a graphic since it is only a 1-hour time slot, thus the value of the energy flexibility for each centroids' clusters is provided. Finally, option 11 (the flexibility profiles clustered on a monthly base) is represented in Figure 47.

Option 2: Daily working days (Mon – Fri)

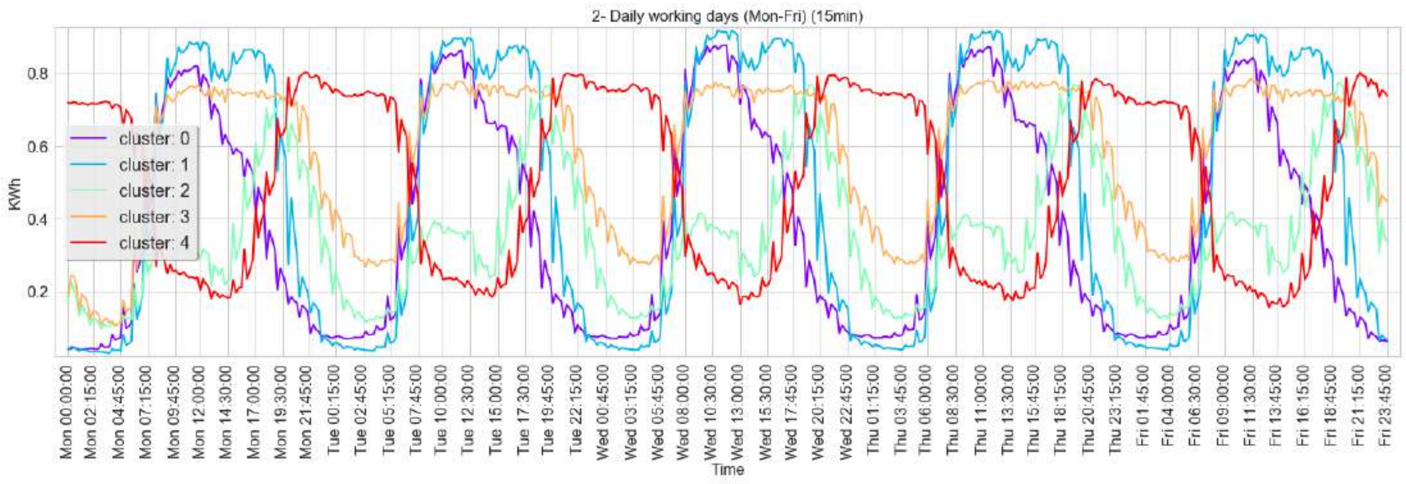


Figure 39: Results for K-means optimal clusters of option 2 “Daily working days”

This option displays five optimal clusters that are cyclically repeated along the week, all working days have the same consumption pattern:

- **Cluster 0:** shows a peak in consumption during the morning, around 8: 00-12: 00 am, and a gradual decrease in the rest of the day;
- **Cluster 1:** the consumption is divided into two strips, in the morning between 8:00 a.m. and 12:30 p.m., and in the afternoon between 3:00 p.m. and 5:45 p.m.;
- **Cluster 2:** shows a slight consumption in the morning that increases during the afternoon where there is a peak from 18:00 to 20:00;
- **Cluster 3:** maintains a constant consumption throughout the day;
- **Cluster 4:** unlike the previous ones, this cluster shows consumption only at night and constantly.

- **Option 3: Daily weekend days (Sat – Sun)**

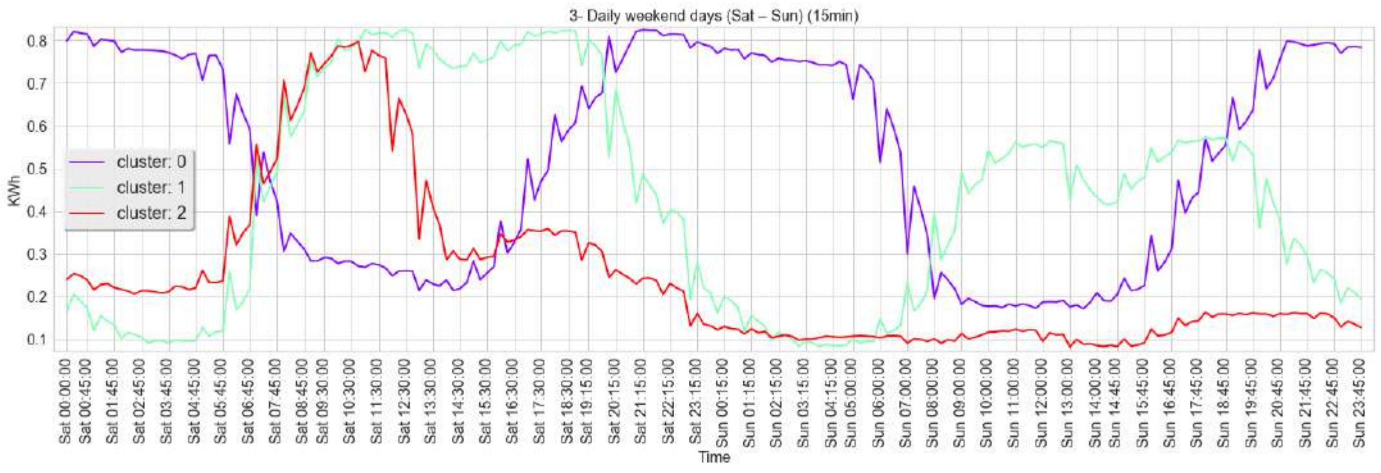


Figure 40: Results for K-means optimal clusters of option 3 “Daily weekend days”

Weekend´s consumption is divided into three clusters.

- **<u>Cluster 0:</u>** illustrates a consumption that begins in the afternoon and lasts overnight until early in the morning
- **<u>Cluster 1:</u>** shows more consumption on Saturdays than Sundays, reaching almost the double.
- **<u>Cluster 2:</u>** shows a slight consumption in the morning that increases during the afternoon where there is a peak from 18:00 to 20:00

- **<u>Option 4: Weekly on hourly base (Mon – Sun)</u>**

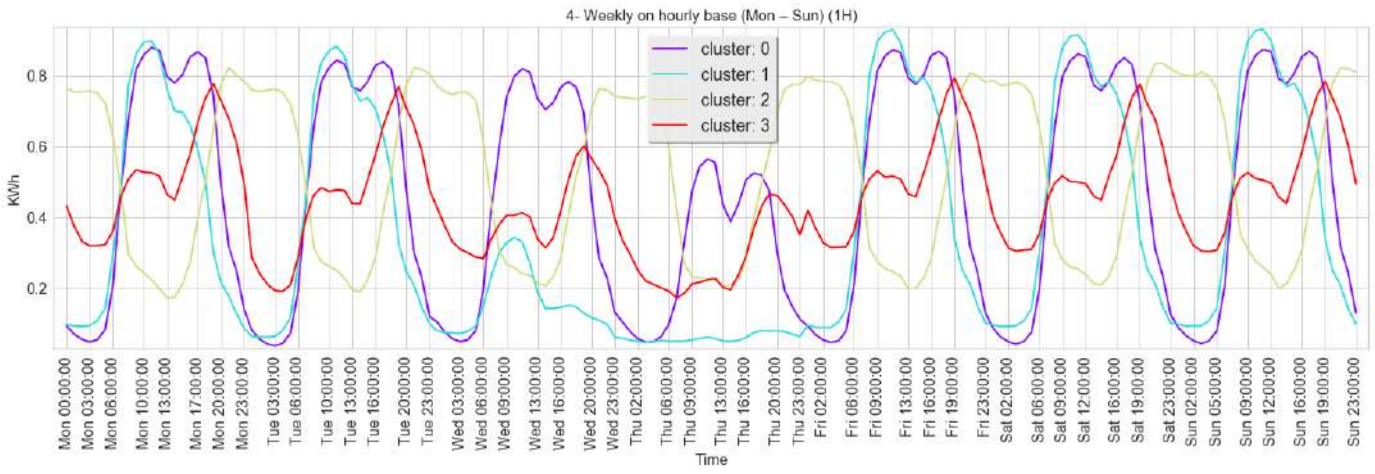


Figure 41: Results for K-means optimal clusters of option 4 "Weekly on hourly base"

This option presents an optimal number of clusters equal to four.

- **<u>Cluster 0:</u>** maintains a similar consumption except on Thursday, which decreases about a third
- **<u>Cluster 1:</u>** follows a consumption´s pattern similar to cluster 0 but its decrease occurs on Wednesday and does not show consumption on Thursday
- **<u>Cluster 2:</u>** presents a constant pattern throughout the week with consumption peaks during the night
- **<u>Cluster 3:</u>** follows a similar pattern of weekly consumption similar to cluster 0, Thursday has an off-peak

- **<u>Option 5: Weekly on daily base (Mon – Sun)</u>**

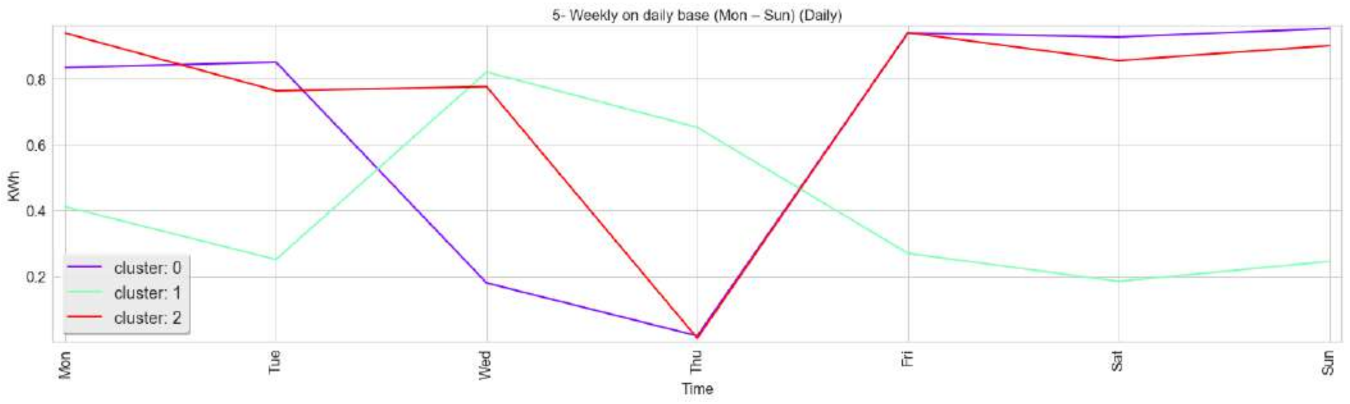


Figure 42: Results for K-means optimal clusters of option 5 "Weekly on daily base"

This weekly option is the same as option 4 but on a daily basis. The choice of three optimal clusters agrees somewhat with the analysis carried out in option 4.

- **Cluster 0:** presents a decrease in consumption for Wednesday reaching its off-peak on Thursday
- **Cluster 1:** maintains a similar consumption during the week except for a peak on Wednesday and its gradual decrease during Thursday
- **Cluster 2:** shows an inverse pattern to cluster 1, with a consumption off-peak on Thursday

The following options depict the optimal number of clusters based on the consumption profile; how much energy is consumed by the prosumer.

- **Option 6: Summer months on daily base (Jun-Jul-Aug)**

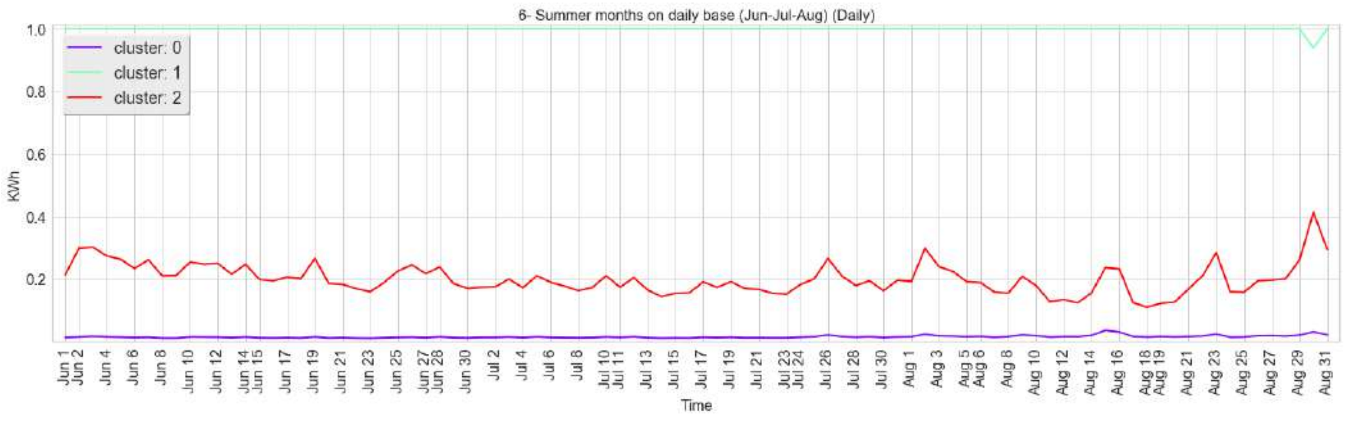


Figure 43: Results for K-means optimal clusters of option 6 "Summer months on daily base"

- **Option 7: Fall months on daily base (Sep-Oct-Nov)**

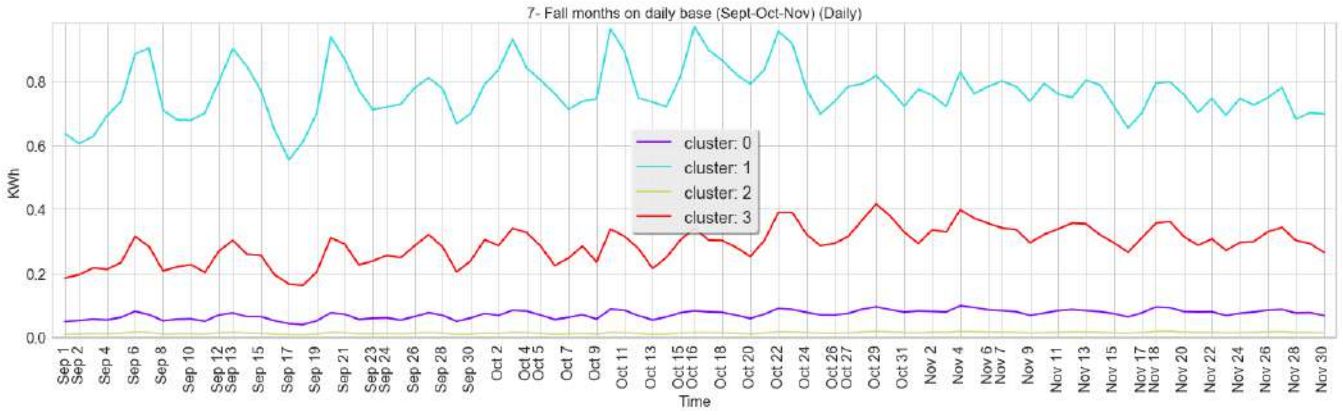


Figure 44: Results for K-means optimal clusters of option 7 "Fall months on daily base"

- **<u>Option 8: Winter months on daily basis (Dec-Jan-Feb)</u>**

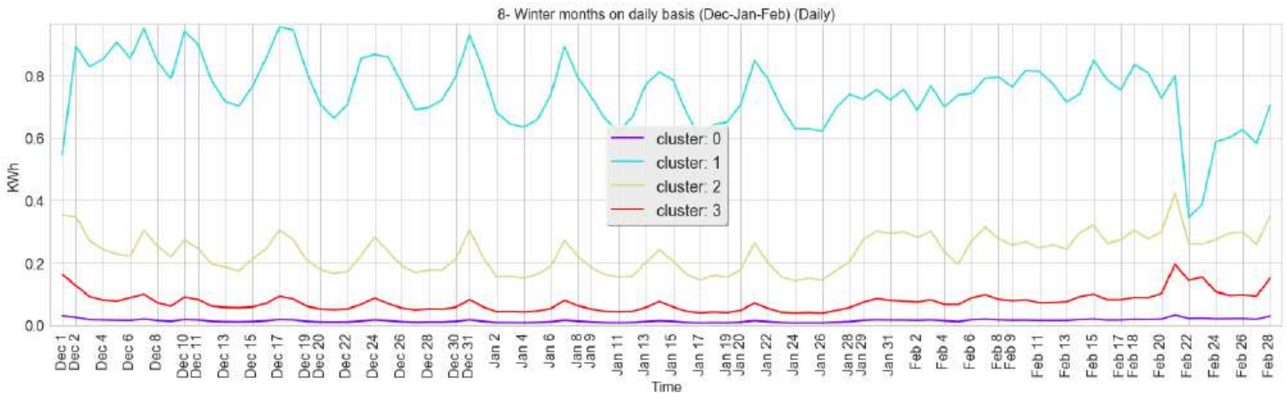


Figure 45: Results for K-means optimal clusters of option 8 "Winter months on daily base"

- **<u>Option 9: Spring months on daily basis (Mar-Apr-May)</u>**

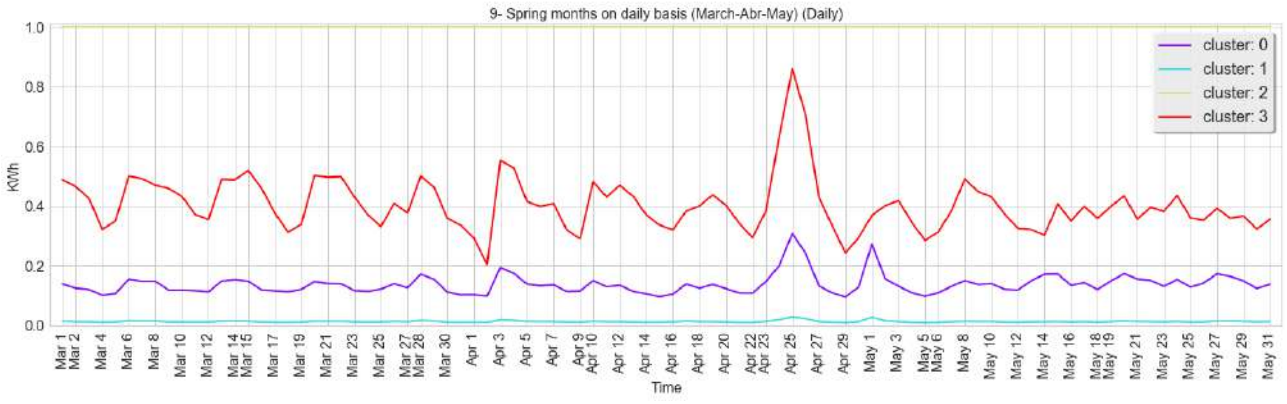


Figure 46: Results for K-means optimal clusters of option 9 "spring months on daily base"

Summer is the only season that presents an optimal number of clusters equal to three; the rest obtain four clusters. Regardless of the number of clusters, a differentiation between prosumers is displayed in all figures; with prosumers who consume little, others with medium consumption, and finally by large consumers.

Furthermore, Summer and Fall are the seasons with the most stable consumption; while Spring and Winter have outstanding peaks on specific days.

- **Option 10: Flexibility working days on daily base**

Option 10 represents the clustering of a specific date and hour of the flexibility profiles. As described in section 3, the "Electricity Consumption / Production Forecasting" module sends the flexibility values to be clustered. By choosing this option in the "Load Profiling" module it is possible to select a specific date and time for clustering the customers portfolio flexibility. Moreover, the "Load Profiling" can separate the positive flexibility (the capacity to increase the energy consumption with respect to the baseline) and the negative flexibility (the capacity to reduce the energy profile below the baseline) and cluster them separately. This scenario reflects the events where an aggregator needs to identify within its pool, a group of prosumers offering a given amount of flexibility for a specific timeslot. It is basically a simulation of a demand response request, where the aggregator must evaluate the flexibility capacity of its portfolio. As an example, it has been simulated a request to be received by the following date and hour: 2019-02-01 14:00:00.

Since the simulated request is for a specific hour, it is not possible to depict the results in a two-dimensional graphic, therefore the obtained clusters are characterized only for the energy flexibility value. For this specific time the down boundary of the flexibility (load reduction capacity) is null, so the positive one is the only available.

Three Clusters have been obtained:

Cluster 0 = 0,012 kWh

Cluster 1 = 0,1496 kWh

Cluster 2 = 0,8605 kWh

- **Option 11: Monthly Flexibility**

Option 11 clusters the flexibility profiles for an entire month. As in the previous options the "Load Profiling" module will provide the dataset organized ready to be clustered and it also gives the possibility to select which month to cluster. In the example shown in Figure 47 the month of February 2019 has been clustered and four different categories of profiles have been identified.

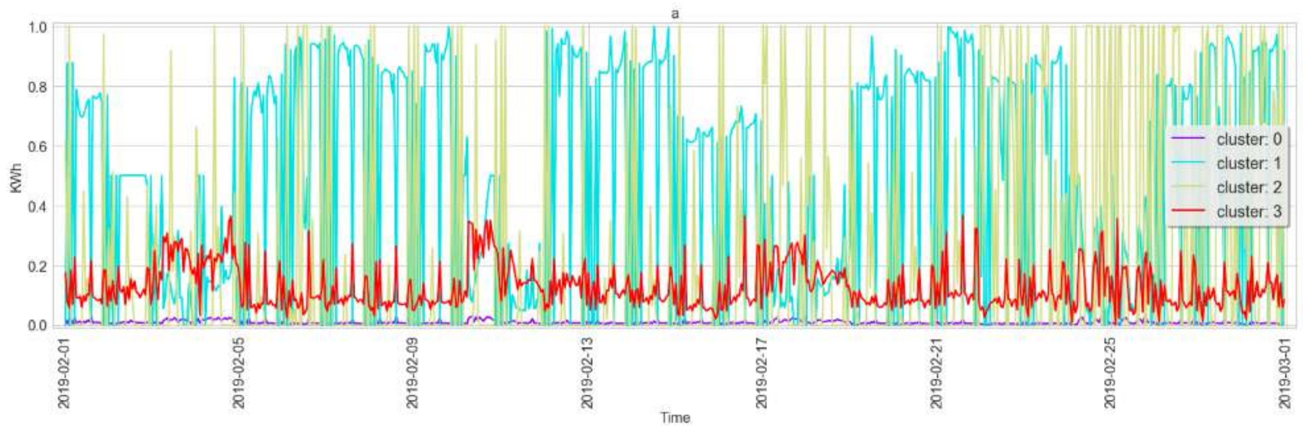


Figure 47: Results of K-means optimal clusters for option 11 "monthly flexibility"

As in option 10, no negative flexibility values have been calculated, so the customers profiles only have positive flexibility for increasing their loads. Cluster 0 shows almost no flexibility, Cluster 1 has regular flexibility capacity in the daytime of working days, Cluster 2 shows irregular spikes along the whole month with a peak concentration in the final days, and Cluster 3 includes customers with poor flexibility capacity.

4.3.3 DBSCAN

Density-based Spatial Clustering of Applications with noise (DBSCAN) Algorithm

DBSCAN (Ester, Kriegel, Sander, & Xu, 1996) belongs to the category of unsupervised data mining techniques and more particulary to the category of density-based clustering, hence the name of the algorithm. Essentially, the algorithm groups various points together, that are close to each other (regions with high density), marking as outlier those points that appear alone in low-density regions. DBSCAN defines as clusters, groups of dense points. (eDREAM H2020 Project, 2019)

Algorithm workflow

As described in D4.2, given eps and minPoints categorize the objects into 3 exclusive groups.

- A point is considered as a **core point** if the corresponded number of points within eps is greater than a predefined number of points (**minPoints**)-These are points that are at the interior of a cluster.
- A **border point** has fewer than **minPoints** within **eps** but belongs to the neighborhood of a core point.
- In addition, if there is a path $p_1, \dots, p_n$, where $p_1 = core\ point$ and p_{i+1} is directly reachable from p_n results that p_n is a border point.
- A **noise point** is any point that is not a core point nor a border point.

As was described in Deliverable 4.2, each core point forms a cluster together with the points that are reachable within its eps radius. Two points are considered **"directly density-reachable"** if one of the points is a core point and the other point is within its ε radius. Larger clusters are formed when directly density-reachable points are chained together. (Wikipedia, 2020)

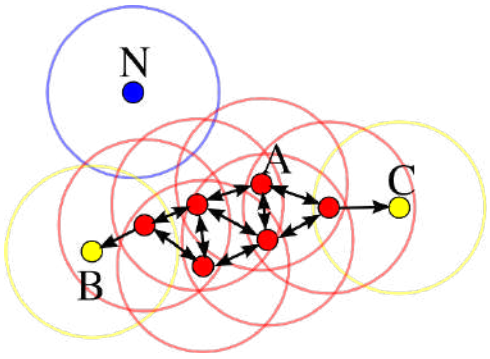


Figure 48: Core, border points and outliers (Wikipedia, 2020)

In Figure 48, the number of minPoints is set 4. Thus, point A is considered as a core point as within its area there are 4 points. B belongs to the neighborhood of core Point A and is considered as a border point, despite that fewer points are within its area. Last but not least, point N is an outlier. Neither is a neighbor of a core point, nor points are enough to form a core point.

Unlike some other clustering techniques, DBSCAN does not require all data points to be assigned to a cluster. The DBSCAN algorithm repeats the following process shown in Figure 49Figure 49 until all points have been assigned to a cluster or are labeled as visited. Some advantages of DBSCAN are:

- The ability to discover clusters of arbitrary shapes (spherical, elongated, linear) and noise.
- Working with spatial datasets.
- There is no need to predefine the number of clusters.

The minor disadvantage of DBSCAN is that it is sensitive to parameters.

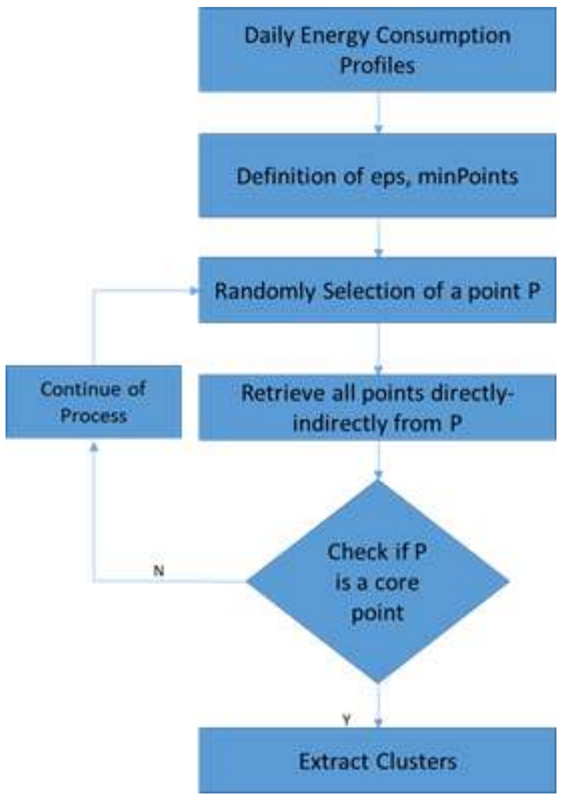

Daily Energy Consumption Profiles
Definition of eps, minPoints
Randomly Selection of a point P
Continue of Process
Retrieve all points directly-indirectly from P
Check if P is a core point
N
Y
Extract Clusters

Figure 49: Process of clusterization with DBSCAN

- **<u>Option 2: Daily working days (Mon – Fri)</u>**

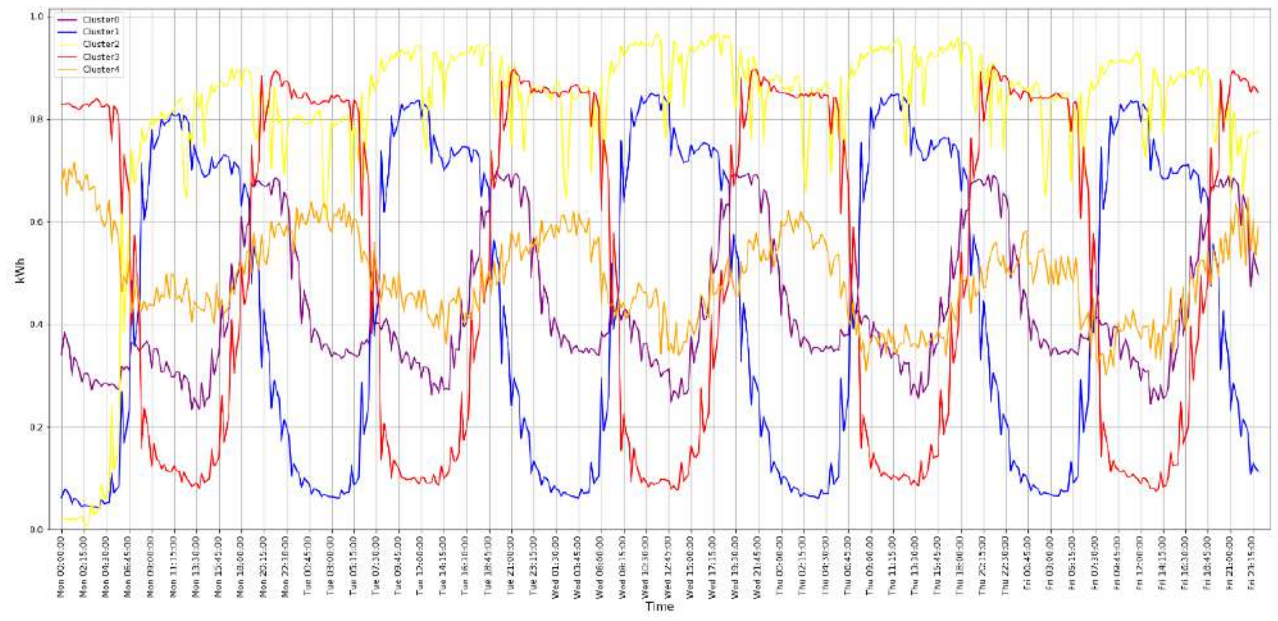


Figure 50: Results for DBSCAN optimal clusters of option 2 “Daily working days”

Figure 50 illustrates the results of the clustering process based on the energy consumption of working days (Monday-Friday). Each color line represents a different cluster showing the aggregated mean values of 15-minute time intervals of the prosumers that were classified into this cluster (different colors). It is evident that the prosumer dataset is separated into 5 clusters.

- **<u>Option 3: Daily weekend days (Sat – Sun)</u>**

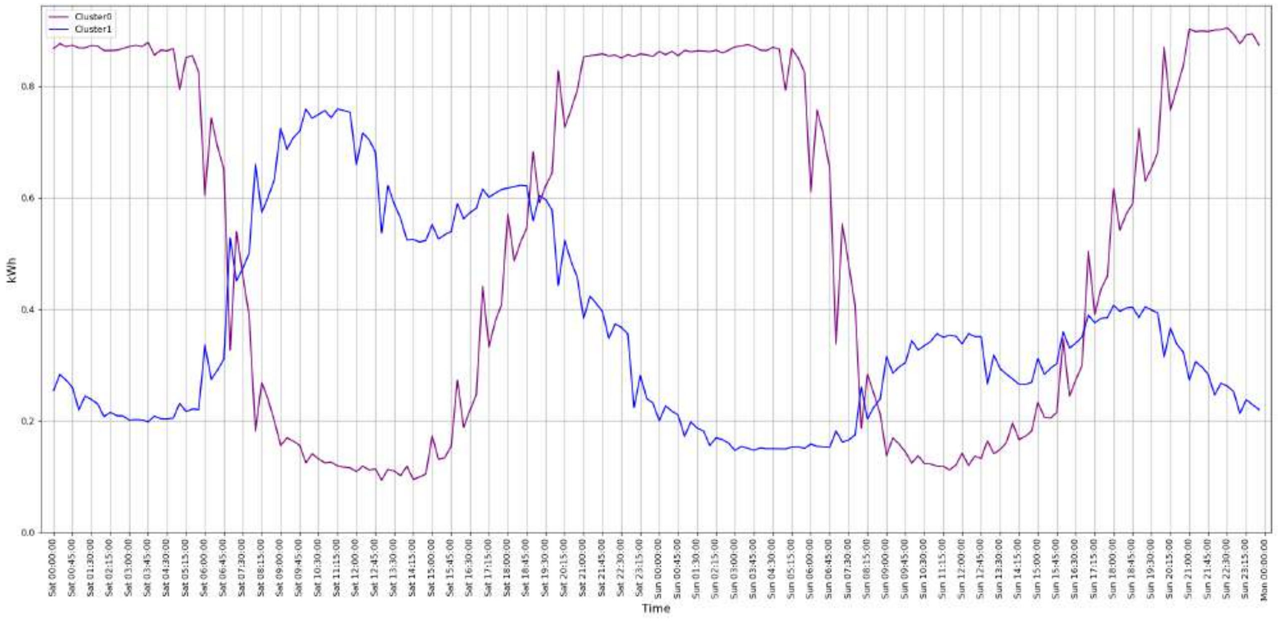


Figure 51: Results for DBSCAN optimal clusters of option 3 “Daily weekend days”

In Figure 51 two clusters are formed representing different behavior regarding the mean energy consumption of 15-minute time intervals during weekends.

- **Option 4: Weekly on hourly base (Mon – Sun)**

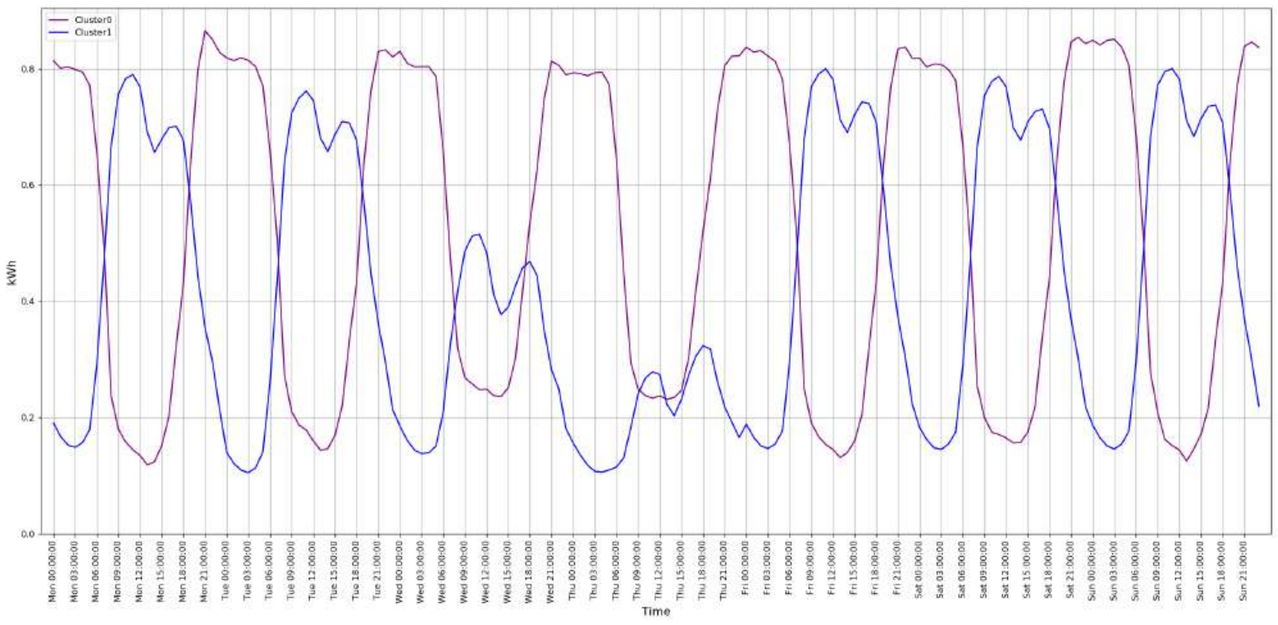


Figure 52: Results for DBSCAN optimal clusters of option 4 "Weekly on hourly base"

Figure 52 shows the results of the clustering process in hourly time intervals representing the mean energy consumption throughout the whole week. Two clusters are formed and as it becomes clear, the second cluster (blue line) shows a significantly lower energy consumption during Wednesday and Thursday compared to the first cluster (purple line).

- **Option 5: Weekly on daily base (Mon – Sun)**

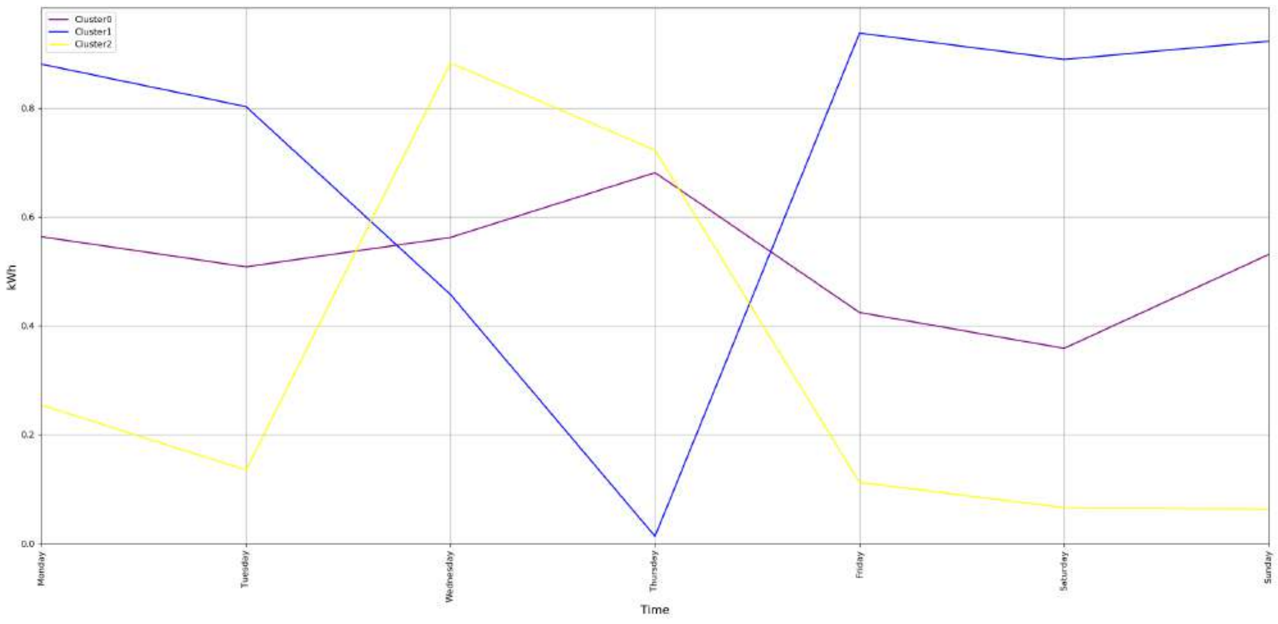


Figure 53: Results for DBSCAN optimal clusters of option 5 "Weekly on daily base"

Figure 53 shows the daily average consumption during the week. Prosumers are grouped into 3 clusters. Cluster 1 (yellow line) and cluster 2 (blue line) have an opposite energy behavior, while cluster 0 (purple line) presents a steady consumption throughout the week.

- **<u>Option 6: Summer months on daily base (Jun-Jul-Aug)</u>**

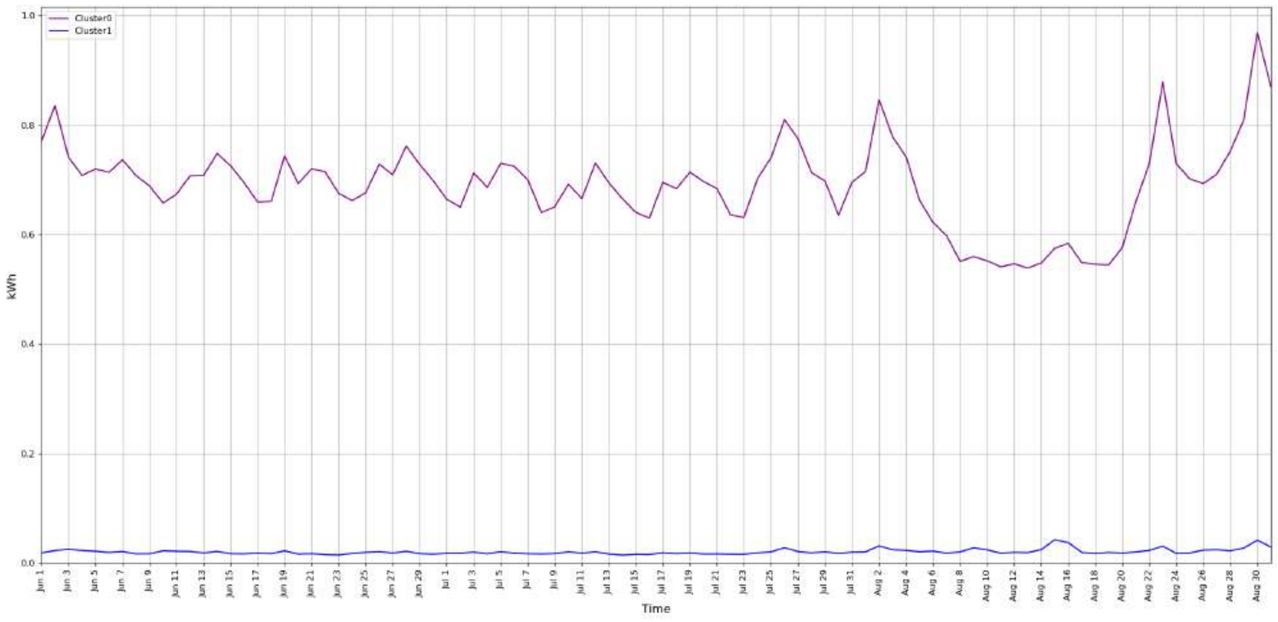


Figure 54: Results for DBSCAN optimal clusters of option 6 “Summer months on daily base”

- **<u>Option 7: Fall months on daily base (Sep-Oct-Nov)</u>**

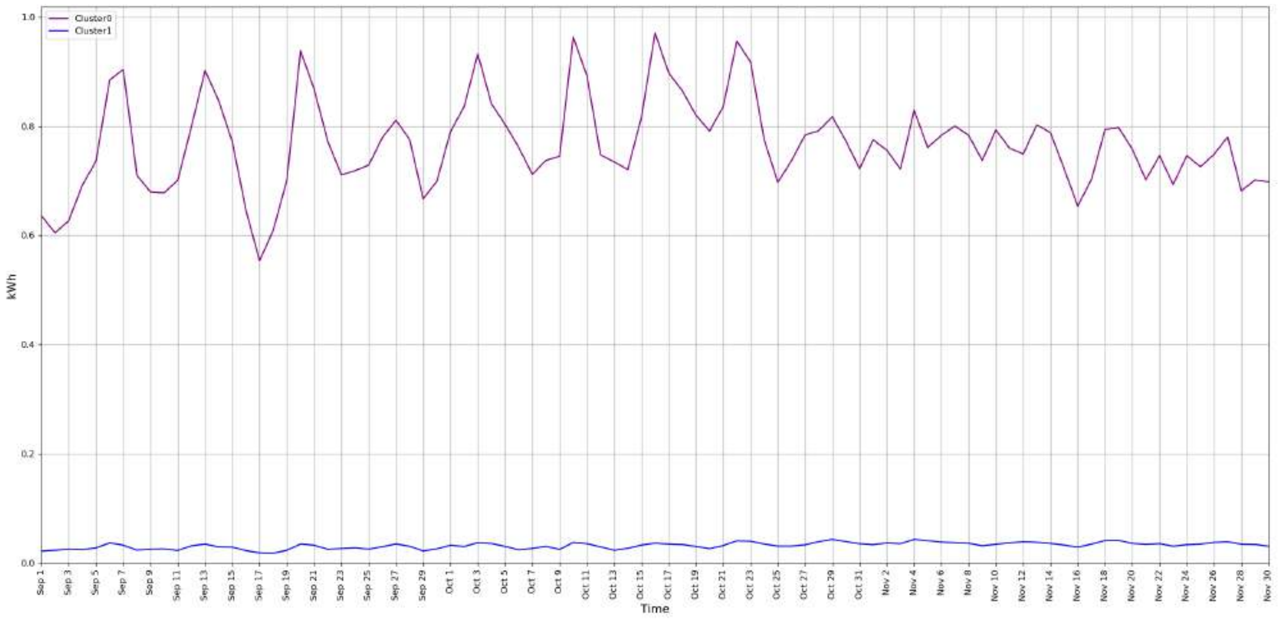


Figure 55: Results for DBSCAN optimal clusters of option 7 “Fall months on daily base”

- **<u>Option 8: Winter months on daily basis (Dec-Jan-Feb)</u>**

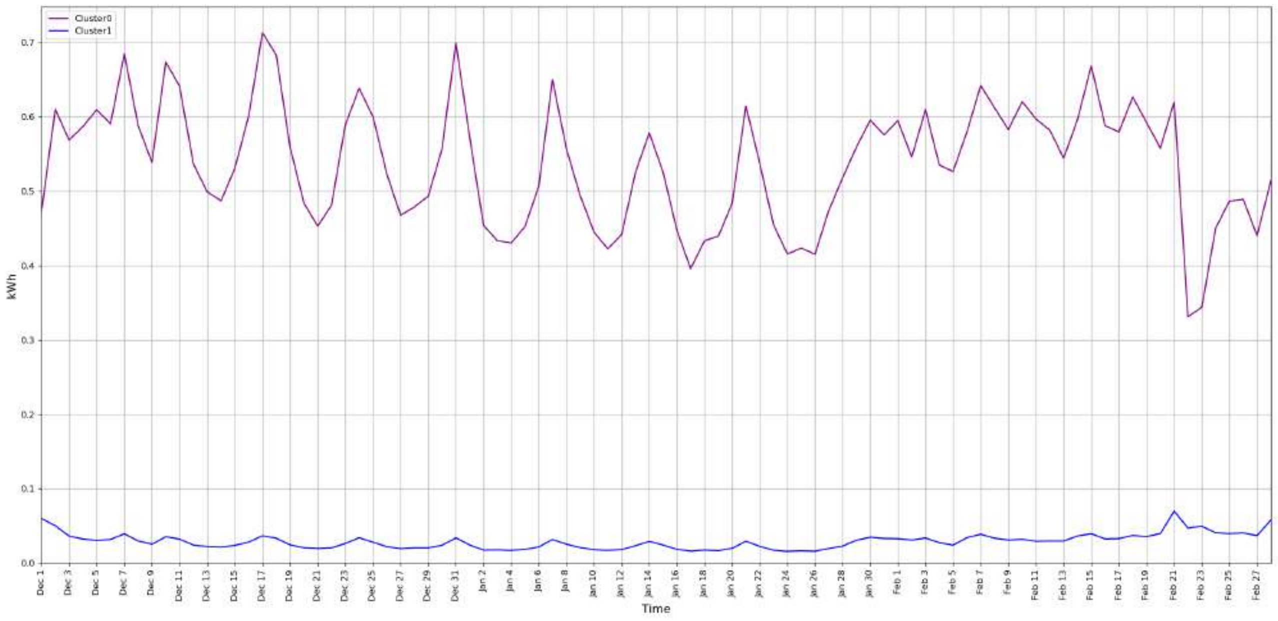


Figure 56: Results for DBSCAN optimal clusters of option 8 "Winter months on daily base"

- **<u>Option 9: Spring months on daily basis (Mar-Apr-May)</u>**

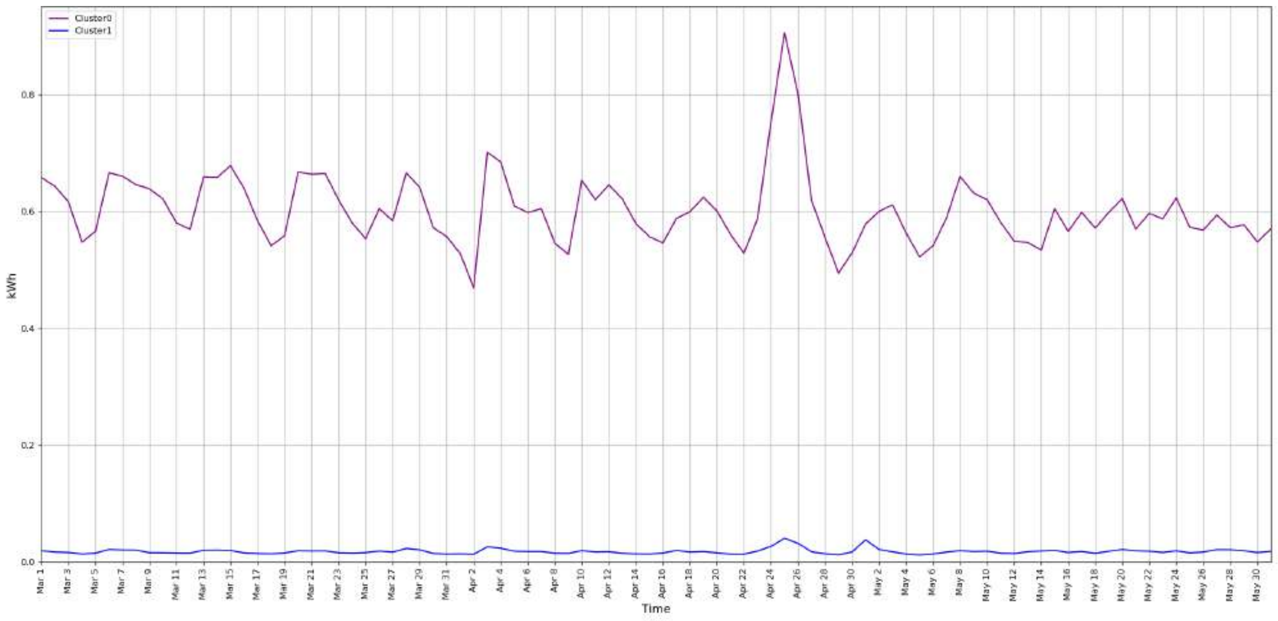


Figure 57: Results for DBSCAN optimal clusters of option 9 "spring months on daily base"

Figure 54, Figure 55, Figure 56 and Figure 57 illustrate the results of the clustering process on the daily average energy consumption in different seasonal periods (summer, winter, fall, spring). As it is evident in each of the 4 figures, two clusters are formed with a significantly different characteristics in their energy consumption behavior.

5 Customer Segmentation

Customer Segmentation module is the last module of the Big Data Layer, which objective is to identify the consumption pattern of a new prosumer and allocate it, with high accuracy, to the proper cluster previously obtained by the "*Big Data Clustering at Multiple Scale*" module.

The first step in order to achieve the goal is to feed the neural network used by this module. A neural network is a type of deep learning algorithm, like a graph, that enables to recognize the underlying patterns in a given data set. This network is characterized by multiple layers of neurons connected to each other, where every layer extract characteristic from an increasingly higher level, combining relationships until reaching the answer. Deliverable D4.2 describes in details the neural network technologies chosen for the Customer Segmentation module.

Algorithm workflow

Two main advantages of this module are its scalability (similarly to the Autoencoder, the more data the better) and its versatility. Training this type of algorithm requires much data, thus only the option 1 data set of load profiling provides this characteristic. This implies that the pattern of the new prosumer must fit the same length as the dataset used.

The procedure for running this tool is described in the following steps:

- The initial data set enters the neural network, through the input layer. This layer must be set with a number of neurons equal to the length of the data set, in this case 96 neurons, the length data of option 1 (see Table 1).
- The information of the input layer neurons feeds the rest of the hidden layers, calculating the weights and processing in the activation function.
- This process returns values that are sent to the output layer where they are compared to a target. The target values are the clusters obtained in the big data clusters module.
- Batch size and epoch parameters determine how many times this process is repeated. Both parameters are critical to the performance of the algorithm. A misallocation can lead to an overfit or underfit.
- Finalizing the iterations, an accuracy is achieved, being able to determine how precise will be the assignation a new prosumer to one cluster.

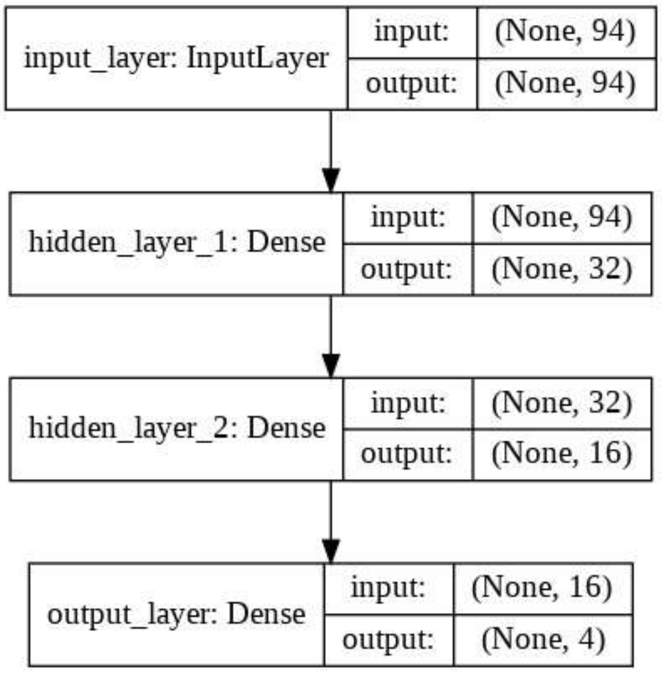


Figure 58: Neural networks topology

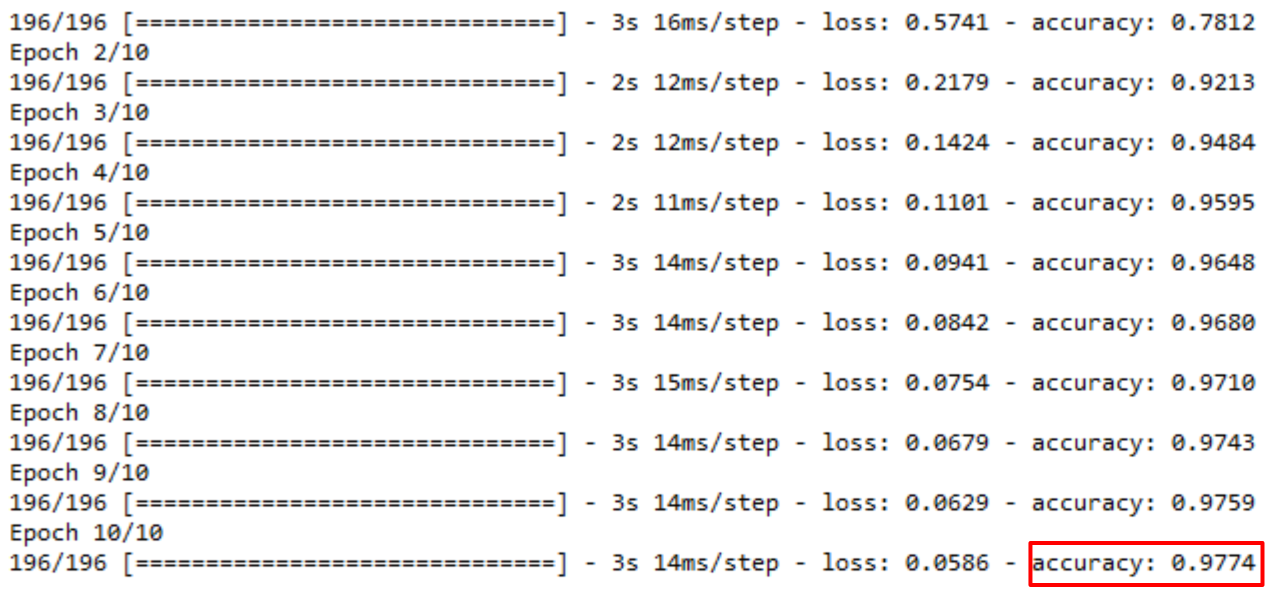
Setting parameters

As in the previous algorithms, Table 8 shows the parameters set for running the neural network

Parameter	Set with	Description
dims [1*, 2*, 3*, 4*]	[length_data, 32, 16, 4]	Dims is an array which contains the required parameter for the autoencoder (encoder-decoder) and k-means algorithm
1*= input data shape	96	Length of data of option 1
2*= first hidden layer	32	Number of nodes in the first hidden layer
3*= second hidden layer	16	Number of nodes in the second hidden layer
4*= number of clusters	4	Number of clusters for K-Means algorithm
epoch	10	One forward pass and one backward pass of all the training examples
batch size	2048	Number of training examples in one forward/backward pass.

Table 8: Setting parameters in Neural Networks algorithm

The algorithm training, for option 1, achieves an accuracy close to 0.98, as illustrated in Figure 59; allowing the inference of the consumption pattern of a new prosumer to be assigned to the correct cluster with a confidence of 98%.



```
196/196 [==============================] - 3s 16ms/step - loss: 0.5741 - accuracy: 0.7812
Epoch 2/10
196/196 [==============================] - 2s 12ms/step - loss: 0.2179 - accuracy: 0.9213
Epoch 3/10
196/196 [==============================] - 2s 12ms/step - loss: 0.1424 - accuracy: 0.9484
Epoch 4/10
196/196 [==============================] - 2s 11ms/step - loss: 0.1101 - accuracy: 0.9595
Epoch 5/10
196/196 [==============================] - 3s 14ms/step - loss: 0.0941 - accuracy: 0.9648
Epoch 6/10
196/196 [==============================] - 3s 14ms/step - loss: 0.0842 - accuracy: 0.9680
Epoch 7/10
196/196 [==============================] - 3s 15ms/step - loss: 0.0754 - accuracy: 0.9710
Epoch 8/10
196/196 [==============================] - 3s 14ms/step - loss: 0.0679 - accuracy: 0.9743
Epoch 9/10
196/196 [==============================] - 3s 14ms/step - loss: 0.0629 - accuracy: 0.9759
Epoch 10/10
196/196 [==============================] - 3s 14ms/step - loss: 0.0586 - accuracy: 0.9774
```

Figure 59: Neural Networks training

6 Conclusions

This document is the second deliverable of task T4.2 "*Big data clustering techniques for load profiling and customer segmentation*" and describes the development of three of the modules of the eDREAM architecture: Load Profiling, Big Data Clustering at Multiple Scale and Customer Segmentation. These modules, together with the pre-processing tool compose the Big Data Layer, a portion of the eDREAM architecture devoted to the analysis of large volume of data. All the development activities and the results presented in this deliverable are based on the methodology and the technologies previously described in deliverable D4.2, the first output of task T4.2.

The Big Data Layer modules will be tested in the Italian pilot (at the ASM site) and the dataset coming from the pilot is described. This dataset includes data from the smart meters of 542 final users for approximately 4 years (2015-2019) with a frequency sampling of 15 minutes. In the first part of this deliverable the Big Data layer architecture is presented in its final version, with references to its stability and scalability properties. The connections among all modules have been reported in dedicated diagrams and explained even considering the following validation activities based on the use cases (use case 1.6 and 3.1 are the ones where these modules are going to be tested and validated). The results of unit tests have been also reported, with some screenshot showing the percentage of code coverage (higher than the 80%). Later on, the full documentation of the API has been included, in order to ensure full accessibility to other modules in the platform. In the last section of this first part, all the adopted technologies (starting from the ones selected in D4.2) have been listed and described.

In the second part of the deliverable the development of the three modules together with the pre-processing tool is described. The pre-processing tool is necessary for preparing the data to be injected in Load Profiling. In most cases raw data from the field need to be cleaned and filtered because in their sampling and acquisition process many external factors can affect their integrity and veracity. The pilot dataset is characterized by many gaps, nulls, nans, etc, so it is mandatory to clean the data before extracting any information. All the different steps of the pre-processing bring to a new dataset with 350 final users. Load Profiling is a tool aimed at organizing the data according to different options. Data received from pre-processing are basically timeseries organized on a daily base (each day of the four years includes a sequence of 96 measures of all prosumers). So, it is essential to re-structure the data matrix in something more useful for extracting insights. Eleven different options have been identified: each one of them organize the data of each customer in a different timing (daily, working days, season, etc.). The first nine options are devoted to the management of historical data, while option 10 and 11 consider the flexibility data from the "Electricity Consumption/Generation Forecasting". In this case a potential user of the eDREAM platform (aggregator, DSO, retailer, etc) can extract information of its portfolio for real time applications (intraday and day-ahead services as described in other eDREAM documents) and for portfolio characterization (planing, tariff analysis and in general long-term and off-line operation). In addition to the organization of data with different eleven options, Load Profiling feeds the Big Data Clustering at multiple Scale module, that is able to cluster the ASM portfolio according to these eleven features. Three different algorithms have been included in this module: Autoencoder, K-means and DBScan. Since the eleven features of Load Profiling can produce dataset with different dimensions and characteristics, each one of the three algorithms can be activated in different conditions. Autoencoder is a deep learning technique that works better with large volumes of data. Tests on the Italian pilot have shown a good level of performance with a dataset greater than half million points, which is choosing feature#1 for ASM dataset). K-means and DBScan can be activated with the rest of the features, but they have different peculiarities. K-means is a general purpose technique that show good scalability, it is fast enough (good applicability to short-response applications like intraday market services) , but it needs an optimal value of clusters previously calculated (see Table 3). On the other hand DBScan needs no pre-calculated optimal number of clusters, it is pretty numb to outliers, but it requires a couple of parameters to be set (see Table 4). With the three algorithms combined, the whole tool features great scalability for the most of Big Data applications and enough flexibility to be applied to different portfolios. Even the optimal number of clusters has been figured out with a script combining several evaluation indexes (see section 4.2). Graphic results have been depicted for the three algorithms.

Finally, the Customer Segmentation tool features a neural network algorithm able to identify a new or random customer profile with the cluster previously calculated. Therefore, new customers and prosumers can be added to an existing portfolio and can be characterized to pre-existent clusters. The new customer profile should be by 96 points data string (according to the structure of the Italian pilot time series). The tool is able to match the customer profile with the profile of a cluster that is the best fit.; The accuracy of this matching is provided in percentage (maximum accuracy of 98% has been obtained). Therefore, it can be very helpful when new customers and prosumers are added to an existing portfolio and they are segmented according to pre-existent clusters. On the other hand, some customers already included in the clustering process, can be moved from a cluster to another and check how they fit with the new segment.

References

Brownlee, J. (2016, May 5). Introduction to the Python Deep Learning Library TensorFlow. Retrieved from https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/

DASK. (2019). Retrieved from https://dask.org/

eDREAM H2020 Project. (2019, May 31). DELIVERABLE D4.2: LOAD PROFILES AND CUSTOMER CLUSTERS V1. Retrieved from https://edream-h2020.eu/wp-content/uploads/2019/07/eDREAM.D4.2.ATOS_.WP4_.V1.0-compressed.pdf

eDREAM H2020 Project. (n.d.). Deliverable D2.5 Requirements-Driven SystemDevelopment V2.

El-Nesr, D. M. (2018, December 31). Imputing the time-series using python. Retrieved from https://medium.com/@drnesr/filling-gaps-of-a-time-series-using-python-d4bfddd8c460

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters. Institute for Computer Science, University of Munich, 226-231. Retrieved from https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf?source=post_page

Garbade, D. M. (2018, September 12). Understanding K-means Clustering in Machine Learning. Retrieved from https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

Google Developers. (2020, February 10). k-Means Advantages and Disadvantages. Retrieved from https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages

Halkidi, M. B. (2001). On clustering Validation Techniques. Journal of Intelligent Information Systems, 104-145.

Ivan, M. (2020). Density-based clustering in R. Retrieved from https://en.proft.me/2017/02/3/density-based-clustering-r/

K, D. (2020, March 2). Anomaly Detection Using Isolation Forest in Python. Retrieved from https://blog.paperspace.com/anomaly-detection-isolation-forest/

Kassambara, A. (2018). CLUSTER VALIDATION ESSENTIALS. Retrieved from https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/

Lewinson, E. (2018, July 2). Outlier Detection with Isolation Forest. Retrieved from https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

Ling J, L. D. (2017). Comparison of Clustering Techniques for Residential Energy Behavior using Smart Merter Data. AAAI-17 (pp. 260-266). San Francisco, CA USA: AAAI Workshops - Artificial Intelligence for Smart Grids and Buildings,. Retrieved from https://www.aaai.org/ocs/index.php/WS/AAAIW17/paper/view/15166/14673

Liu, E. (2015, November 6). Calinski-Harabasz Index and Boostrap Evaluation with Clustering Methods. Retrieved from https://ethen8181.github.io/machine-learning/clustering_old/clustering/clustering.html

Miller, N. S. (2018, August 29). pyculiarity 0.0.7, 2015. Retrieved from https://pypi.org/project/pyculiarity/

Munnelly, G. (2017). K-Means. Retrieved from https://www.scss.tcd.ie/~munnellg/projects/kmeans.html

NumPy. (2020). NumPy. Retrieved from https://numpy.org/devdocs/

Pandas. (2014). Intro to data structures. Retrieved from https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html

pandas. (2014). pandas.DataFrame.interpolate. Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/frame.html

Pandas. (2020). Retrieved from https://pandas.pydata.org/index.html

Prado, K. S. (2017). How DBSCAN works and why should we use it? Retrieved from https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80

Python. (2020). Retrieved from https://www.python.org/: https://www.python.org

Robert Tibshirani, G. W. (2000, November). Estimating the number of clusters in a data set via the gap statistic. 411-423. Retrieved from https://statweb.stanford.edu/~gwalther/gap

Scikit learn. (2019). sklearn. metrics. Retrieved from https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

Scikit learn. (2019). sklearn.preprocessing.MinMaxScaler. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Scikit-learn. (2019). 2.3. Clustering. Retrieved from https://scikit-learn.org/stable/modules/clustering.html

Scikit-learn. (2019). sklearn.cluster.KMeans. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Scikit-learn. (2019). sklearn.ensemble.IsolationForest. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

scikit-learn. (2020, March). scikit-learn. Retrieved from https://scikit-learn.org/stable/

Solutions, M. (2017, October 3). Python: 7 Important Reasons Why You Should Use Python. Retrieved from https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b

TensorFlow. (2020). TensorFlow. Retrieved from https://www.tensorflow.org/

Torabi, M., Hashemi, S., Saybani, M. R., Shamshirband, S., & Mosavi, A. (2018, June 27). A Hybrid clustering and Classification Technique for Forecasting Short-Term Energy Consumptio. Wiley Online Library, 38(1), 11. doi:10.1002/12934

Wikipedia. (2020, 3 30). DBSCAN. Retrieved from https://en.wikipedia.org/wiki/DBSCAN

Yassine, S. S. (2018, February 20). Big Data Mining of Energy Time Series for Behavoiral Analytics and Energy Consumption Forecasting. MDPI, 26. Retrieved from www.mdpi.co/journal/energies

Yellowbrick. (2019). Source code for yellowbrick.cluster.elbow, Revision 682b3528. Retrieved from https://www.scikit-yb.org/en/latest/_modules/yellowbrick/cluster/elbow.html

Yellowbrick. (2019). Yellowbrick: Machine Learning Visualization, Revision 682b3528. Retrieved from https://www.scikit-yb.org/en/latest/

Yin, X. G. (2017). Improved Deep Embedded Clustering with Local Structure Preservation. Proceedings of the Twenty-Sixth International Joint Conference on, 1753-1759.