ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΤΕ

Electricity use profiling and forecasting at microgrid level

MSc Thesis

by

Enea Mele

Supervisors:
Charalambos Elias, Assistant Professor, TEI of Sterea Ellada
Aphrodite Ktena, Professor, TEI of Sterea Ellada

2018

This page is intentionally left blank

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΤΕ

Electricity use profiling and forecasting at microgrid level

Διπλωματική Εργασία

του

Ενέα Μέλε

Επιβλέποντες: Χαράλαμπος Ηλίας, Επίκουρος Καθηγητής, ΤΕΙ Στερεάς Ελλάδας
Αφροδίτη Κτενά, Καθηγήτρια, ΤΕΙ Στερεάς Ελλάδας

Μεταπτυχιακή Διατριβή που υποβάλλεται για την μερική εκπλήρωση των υποχρεώσεων απόκτησης του τίτλου του Προγράμματος Μεταπτυχιακών Σπουδών «Ευφυής Διαχείριση Ανανεώσιμων Ενεργειακών Συστημάτων» του Τμήματος Ηλεκτρολόγων Μηχανικών ΤΕ του ΤΕΙ Στερεάς Ελλάδας

MSc Thesis submitted in partial fulfillment of the requirements for the degree of Master in "Intelligent Management of Renewable Energy Systems"

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31 Μαΐου 2018

.....	.....	.....
Χαράλαμπος Ηλίας	Αφροδίτη Κτενά	Χρήστος Μανασής
Επίκουρος Καθηγητής	Καθηγήτρια	Καθηγητής

2018

.....

ΕΝΕΑ ΜΕΛΕ, Πτυχιούχος Ηλεκτρολόγος Μηχανικός Τ.Ε., M.Sc.

Abstract

The aim of this thesis is to create a flexible and easily customized tool applicable in microgrids to carry out electricity use profiling and forecasting. This modular tool is called Divinus and its architecture consists of several interconnected well-defined components where each one interacts directly with the other. The first three structural pillars of the platform are its database where all the information is stored, the Django framework in which the code exists and finally the website where all the results are displayed. The next set of components are not as structural as they are functional. Upon them is based the collection of data that will be saved in the database, the use profile that will be performed on the collected data and the load forecasting for which use profiling data will be used.

Through the Self-Organizing Map, that are competing networks that provide topological mapping to the imported data, we perform the use profiling based on the collected data of Technological Institute of Sterea Ellada, Psachna campus from 2010 till 2017. As soon as the use profiling is complete and these data are placed in clusters based on their characteristics the forecasting process is able to begin. The forecasting is performed based on the machine learning methodology and more specifically with the k-neighbours algorithm.

From the tests that have been carried out so far, we observed that Divinus has a high accuracy and low mean errors. More specifically based on forecasts made for the next five days, the next month and the next year the average error does not exceed 5% for the next five days, 12% for next month and 16% for the next year.

Therefore, at the current stage of the tools is we are able to say that it is quite promising tool and that is likely to be used for both short-term and medium-term forecasts.

Key Words: Use Profiling, Self-Organizing Maps, Load Forecasting

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι η δημιουργία ενός ευέλικτου και εύκολα προσαρμόσιμου εργαλείου που θα εφαρμοστεί σε microgrids για την δημιουργία ενεργιακών προφίλ χρήσης ηλεκτρικής ενέργειας και για την πρόβλεψη φορτίου. Το αρθρωτό αυτό εργαλείο ονομάζεται Divinus και η αρχιτεκτονική του αποτελείται από πολλά διασυνδεδεμένα και καλά καθορισμένα στοιχεία, όπου το καθένα αλληλεπιδρά άμεσα με το άλλο. Οι τρεις πρώτοι δομικοί πυλώνες της πλατφόρμας είναι η βάση δεδομένων, στην οποία αποθηκεύονται όλες οι πληροφορίες, το Django framework στο οποίο υπάρχει ο πηγαίος κώδικας και τέλος ο ιστότοπος όπου εμφανίζονται όλα τα αποτελέσματα. Το επόμενο σύνολο στοιχείων δεν αφορά τόσο την δομική όσο την λειτουργική πλευρά του Divinus. Στα στοιχεία αυτά εμπεριέχονται διαδικασίες όπως είναι η συλλογή δεδομένων που θα αποθηκευτούν στη βάση, η δημιουργία ενεργειακών προφίλ χρήση που θα εκτελεστεί πάνω στα δεδομένα που συλλέγονται καθώς και η πρόβλεψη φορτίου για την οποία θα χρησιμοποιηθούν δεδομένα από τα ενεργειακά προφίλ χρήσης.

Μέσω τον αυτοοργανωτικών χαρτών, που είναι ανταγωνιστικά δίκτυα που παρέχουν τοπολογική χαρτογράφηση στα εισαγόμενα δεδομένα, πραγματοποιούμε τη δημιουργία ενεργιακών προφίλ χρήσης ηλεκτρικής ενέργειας με βάση τα συλλεχθέντα δεδομένα από το 2010 έως το 2017 της περιοχής των Ψαχνών Ευβοίας του Τεχνολογικού Εκπαιδευτικού Ινστιτούτου Στερεάς Ελλάδας. Μόλις η χαρτογράφηση των δεδομένων αυτών είναι πλήρης τοποθετηθούν σε ομάδες βάσει των χαρακτηριστικών τους, η διαδικασία πρόβλεψης είναι σε θέση να ξεκινήσει. Η πρόβλεψη πραγματοποιείται με βάση τη μεθοδολογία machine learning και πιο συγκεκριμένα μέσω του αλγόριθμο k-neighbours.

Από τις δοκιμές που έχουν πραγματοποιηθεί μέχρι τώρα, παρατηρούμαι ότι το Divinus έχει υψηλή ακρίβεια και μικρά σφάλματα. Πιο συγκεκριμένα, με βάση τις προβλέψεις που πραγματοποιήθηκαν για τις επόμενες πέντε ημέρες, τον επόμενο μήνα και τον επόμενο χρόνο, το μέσο σφάλμα δεν υπερβεί το 5% για τις επόμενες πέντε ημέρες, το 12% για τον επόμενο μήνα και το 16% για το επόμενο έτος.

Ως εκ τούτου, στο στάδιο που βρίσκεται αυτήν την στιγμή το Divinus μπορούμε να πούμε ότι αποτελεί ένα πολύ ελπιδοφόρο εργαλείο που είναι πιθανό να χρησιμοποιηθεί τόσο για βραχυπρόθεσμες όσο και για μεσοπρόθεσμες προβλέψεις.

Λέξεις – Κλειδιά: Ενεργειακό Προφίλ, Αυτοοργανωτικοί Χάρτες, Πρόγνωση φορτίου

Table of Contents

Index of Tables

Index of Figures

Introduction

Until recently, the electricity production and distribution systems were located far away from end-use points. This caused a lot of losses during the energy transport. Moreover, it also hindered the decentralization of power generation that made the dependence on large generation plants even higher. However, the efforts made in order to cope with the increased energy demands as well as transport costs lead to the creation of various techniques such as the energy forecasting. The knowledge of future load behaviour in electrical distribution systems was of fundamental importance in many electrical systems, being one of the main subjects discussed in the operational areas of electricity utilities. Moreover, load forecasting was also used for possible energy interchange with other utilities as well as to make the system more stable and secure [1].

However, a conceptual change has been proposed so as to make the current supply system more sustainable in economic and environmental terms, as reflected for instance in the Lisbon Treaty [2]. As a result, in order to increase sustainability and optimize resource consumption, electric utilities should constantly try to adjust their power supply to the energy demands. Moreover, taking into account that it is extremely difficult to store energy at a large scale, power generation has to be adjusted with the real time demand [1]. Therefore, it is of crucial importance that the electric load forecasting to be as accurate as possible.

In order to succeed a high accuracy in load forecasting access is required to a wild variety of electric power demand factors such as the day of the week, the month of the year as well as the corresponding data at the respective days and months of past years along with past and future environmental data such as humidity, temperature etc. However, as the data gathered in the smart grid increases, the importance of clustering techniques that will classify those data increases because huge amounts of data will need to be reduced in a reasonable way.

Decades ago, the clustering that was performed in electricity customers was performed only based on pre-assigned contract types such as household, manufacturer, and school. The clustering of customers is, however, now possible based on real-time energy consumption patterns because of the richness of data in smart grids. Therefore, clustering is considered to be a pre-processing stage in many data analysis scenarios [3].

Nowadays, the need for renewable energy resources led to the emergence of microgrids that are environments of small electric power generation and demand. However, traditional clustering and forecasting methods cannot have direct application to microgrids for two main reasons. In microgrids the aggregated consumption figure is not only several times smaller than in region-wide areas, but also the load curve presents a much higher variability [1].

Based on the aforementioned the purpose of this master thesis is the creation of a Short-Term Load Forecasting two stage prediction methodology, called Divinus, which is based on the Self Organised Map (SOM) clustering technique and on a custom made forecasting technique using machine learning which can be applied in microgrids enviroments.

Due to the fact that there is a wide variety of clustering techniques, databases used for storing data and programming languages used in machine learning techniques, in the Introduction Section we analyse the reasons that led us to use these systems. Section 2 presents how and from where the required data for the clustering phase were acquired. It is a crucial chapter due to the fact those data consist the backbone of our methodology and therefore have to be as accurate as possible. Section 3 describes how those data were implemented into the PostgreSQL database. Section 4 describes how the data stored in the database were used by the Self Organized Map for the clustering process. Section 5 presents the machine-learning forecasting technique based on the data that were clustered previously and last but not least Section 6

analyses the results obtained, summarizes the conclusions of this study and proposes future improvements on the proposed tool.

1.1. Electrical Load Curves Clustering Methods

Nowadays, the collection of scientific data is performed much easier and faster due to the advances in modern mining techniques. Scientists are able to unearth implicit information from huge databases and use them much easier and faster than it was done in the past. However, these data mining techniques besides the benefits that they brought, they also resulted in a large scale accumulation of data pertaining to diverse fields. It is practically impossible to extract useful information from a huge load of data whose attributes might be totally different. Therefore, an essential and effective method had to be found in order to deal with these issues. Cluster analysis is such a method that was introduced to deal with these issues. The main aim of cluster analysis is to find and associate patterns by forming groups of patterns that contain similar attributes. In this way the pattern groups that will be formed will include objects that have similar attributes compared to different clusters that differ considerably, with respect to their attributes [4].

Many clustering approaches and algorithms have been proposed from time to time in literature to suit various requirements [4], [5]. Many of them are based on conventional approaches are such as the numerical clustering approach which assumes that patterns are points in a dimensional space and perform clustering by defining a (dis)similarity measure. Another conventional approach is the symbolic clustering approach which is suitable for clustering patterns or objects that are often represented by qualitative or symbolic features. On the other hand, knowledge-based clustering approaches use high-level knowledge pertaining to a set of problems to perform the clustering task. In these approaches, knowledge is embedded into the approach for solving a class of problems [5].

Recently, clustering analysis methods and techniques have been used in the field of electrical engineering in order to cluster load curves [4]- [13]. These techniques are suitable in defining typical load profile (TLP) of customers. Different applications are available for classification of the load curve of customers. Accurate knowledge of the customers' consumption patterns represents a worthwhile asset for electricity providers in the competitive electricity markets [6]. Classification of loads in terms of their time-varying power consuming behavior is an important task for load forecasting, load data processing, locational customer services, power system analysis and pricing [7]. With the electricity market liberalization, the electricity distribution business looks for better market strategies based on adequate information about the consumption patterns of the electricity customers. A fair insight into the customer's consumption behavior allows the distribution utilities to better address the operation of the distribution infrastructure and its future enhancement, not to mention the ability to design specific tariff options for the various classes of customers in tune with real operation costs [8]. In order to cope with the ever-increasing demands of the market that arise, different methods are used in clustering load curves. Some of the most popular methods used are the K Means, the Modified Follow the Leader, the Self-Organizing Maps, etc.

1.1.1. Classical K-means

K-means algorithm was first introduced by J.B. MacQueen in 1967 [12], [14]. K-means is a type of unsupervised learning clustering algorithm, which means that it uses data without having previously defined the categories or the groups that these data will be inserted. Data are clustered based on feature similarities and the process consist of two separate phases. The first phase is to define k centroids, one for each cluster. During this phase each data point based on the Euclidean distance, which is considered to determine the distance between data points and

the centroids, is assigned to its nearest centroid. More specifically, if ci is the collection of centroids in set C, then each data point x is assigned to a cluster based on the following formula:

$$\underset{C_i\ EC}{arg\ min}\ dist(c_i, x)^2 \qquad (1.1)$$

Where dist(-) is the standard (L_2) Euclidean distance. Let the set of data point assignments for each i^{th} cluster centroid be Si [16]. The first step is completed when all the points are included in one of the cluster groups and an early grouping is performed.
In the second phase the centroids need to be recalculated as the inclusion of new points may lead to a change in the cluster centroids. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|}\Sigma_{x^i ES_i}{}^{x_i} \qquad (1.2)$$

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster. The algorithm iterates between steps one and two until a situation will be reached when the centroids do not move anymore [16]. This signifies the convergence criterion for clustering. In Table 1 is presented a pseudocode for the k-means clustering algorithm [15].

Table 1 - Pseudocode for the k-means clustering algorithm *[15]*

Input:

D = {d1, d2,.....,dn} //set of n data items.
k // Number of desired clusters

Output:

A set of k clusters.

Steps:

1. Arbitrarily choose k data-items from D as initial centroids;
2. Repeat

   Assign each item di to the cluster which has the closest centroid;
   Calculate new mean for each cluster;

   Until convergence criteria is met.

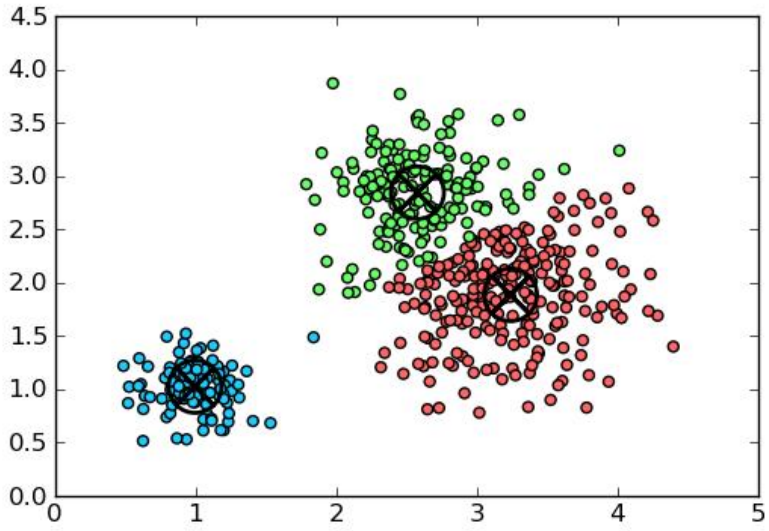


Figure 1 - K-means Clustering Visualization *[17]*

The k-means algorithm is the most extensively studied clustering algorithm. The major drawback of this algorithm is that it produces different clusters for different sets of values of the initial centroids. Quality of the final clusters heavily depends on the selection of the initial centroids. The k-means algorithm is computationally expensive and requires time proportional to the product of the number of data items, number of clusters and the number of iterations [15].

1.1.2. Weighted Fuzzy Average (WFA) K-means

Fuzzy logic is based in an intuitive theory based on human reason of approximation. It differs from traditional logic methods due to the fact that each data point has a probability of belonging to each cluster, while in traditional methods exact and solid results are expected. Zadelh was the first that put forth the concept of fuzzy logic [18] and since 1975 it is used in problems where the solution tends to be more approximate rather than exact. Therefore, due to its principles fuzzy logic quickly became an integral part of solving clustering problems in which their results were determined by some degree of closeness to true or to false.

Weighted Fuzzy Average k-means was proposed as a new method which could be used to overcome the drawback of the k-means algorithm in the computation of the distance between each vector and cluster center. The benefit of this new method over the previous one was that it introduced a fuzzy averaging that puts the center prototype among more situated points [13], [14]. The weighted fuzzy average (WFA) of the vectors in a cluster is done component-wise. Let $\{x_1,\ldots, x_P\}$ be a set of P real numbers. To find its weighted fuzzy average, this algorithm initially takes the sample mean $\mu^{(0)}$ and variance σ^2 to start the process. A Gaussian is centred over the current approximate WFA $\mu(r)$ and iterates as follows [19]:

$$w_p^{(r)} = \frac{\exp\left[-\frac{\left(x_p-\mu^{(r)}\right)}{2\sigma^2}\right]}{\sum_{(m=1,P)}\exp[-\frac{x_m-\mu^{(r)}}{2\sigma^2}]} \tag{1.2}$$

$$\mu^{(r+1)} = \sum_{(\rho=1,P)} w_p^{(r)}\, x_p, r = 0,1,2,\ldots \tag{1.3}$$

The denominator in Equation (1.2) standardizes the weights so they all sum to unity. We compute σ^2 on each of three or four iterations and then leave it fixed. After about five iterations the approximate WFA is sufficiently close to the true WFA. Schneider and Craig [1992] used

a weighted fuzzy expected value for histogram adjustment, but it was based on a decaying exponential. Figure 2 below shows an example of five points (circles) that compares the mean, median, and the WFA [19].

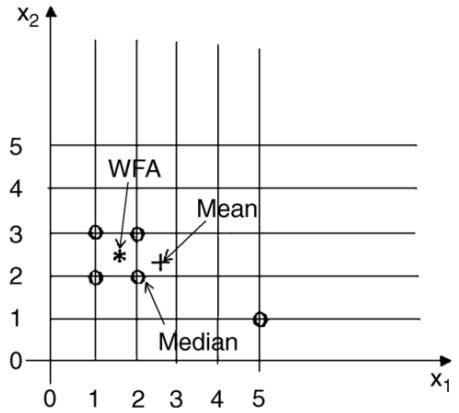


Figure 2 - A mean, median, and WFA of five points *[19]*.

Even though this method was an improvement on the simple K-means, it still lacked the ability of finding better centers, since mean does not always represent the center of a given data.

1.1.3. Modified Follow the Leader (MFTL)

In many communities such as social networks datasets there are usually some members who play a key role. The reason why some members have a higher role within social network analysis is their centrality. Members that have a high centrality have a greater structural importance in the network and as a result can be named also as leaders. In the follow-the-leader procedure a group is formed starting from the leader and new members are added based on the relationship they have with the group. To put it in simple words, this algorithm process requires to choose the vertex (Leader) with the highest centrality score that is not included in any existing groups. Then after the new group containing a leader member has been created, a repetitive process is required so as to add new vertexes. The new vertexes will be added only if the new density of the newly extended group is above a given threshold [20].

As we understand from the aforementioned description, the Follow-the-leader algorithm does not require cluster numbers initialization and uses an iterative process to compute the cluster centroids. The first cycle of the algorithm, using a follow-the-leader approach that depends on a distance threshold ρ, sets the K numbers of clusters and the number $n^{(k)}$ of patterns belonging to each cluster $k = 1, \dots, K$. The subsequent cycles refine the clusters, by possibly reassigning the patterns to closest clusters. The procedure stops when the number of patterns changing clusters in a single cycle is zero. The process is essentially controlled by the distance threshold ρ, which has to be chosen by a trial-and-error approach. This procedure has been modified to fit the needs of the proposed classification, by taking into account the data dispersion in the input vector [6], [22]. For this purpose, the Euclidean metric used in the original algorithm has been modified by introducing for each index a weighting factor, where is the variance of the hth feature computed from all the load patterns in the initial population, and $\bar{\sigma}^2$ is the average value of the variance σ_h^2 for $h = 1, \dots, H$. As such, the impact of the indexes having a high variance is amplified in the computation of the weighted Euclidean distance [21].

1.1.4. Hierarchical algorithm

In hierarchical clustering, there are initially M singleton clusters, as much as the number of representative load patterns (RLTs) [21], [23]. At first, a MxM similarity matrix is built using the Euclidean norm distance criterion. Then the value expressing the similarity between the clusters $X^{(q)}$ and $X^{(s)}$, which is the $\gamma^{(q,s)}$, needs to be called. Afterwards, with the use of a linkage criterion which is based on the similarity matrix, the M RLPs are grouped into binary

clusters. The process is iteratively repeated by merging the clusters of each level into bigger ones at the upper level, until all RLPs are grouped in a single cluster. The history of the process is kept in order to form a binary tree structure, whose root is the cluster that contains the whole data set [21].
The similarities between clusters at each level are measured by the linkage criterion which is also responsible for determining the cluster formation at the upper level. The extreme cases for these criteria include the single linkage, for which the similarity between two clusters depends on the closest pair of members in the two clusters, and the complete linkage, for which the similarity between two clusters depends on the farthest pair of members in the two clusters [24]. As a result, the single linkage criterion may lead to the formation of few large clusters, whereas the complete linkage criterion may form too many clusters. In order to prevent these effects, other linkage criteria, such as average distance and Ward [25], have been defined [21].
With the average distance criterion, grouping two clusters $X^{(s)}$and $X^{(t)}$ depends on the average distance as it shown in the following equation:

$$\gamma_A^{(s,t)} = d(X^{(s)}, X^{(t)}) \tag{1.3}$$

Once two clusters $X^{(s)}$and $X^{(t)}$ have been merged to form $X^{(w)}$, the similarity between the new cluster and another cluster $X^{(g)}$ becomes as it shown in the following equation:

$$\gamma_A^{(w,g)} = \frac{1}{2}(\gamma_A^{(s,g)}, \gamma_A^{(t,g)}) \tag{1.4}$$

The hierarchical tree (or dendrogram) of Figure 3 is obtained by grouping the RLPs of the data set by this method. The horizontal axis contains the RLP identifiers, whereas the height of each vertical branch represents the similarity between each pair of merged clusters. The final clusters are then constructed by choosing in the binary tree the maximum distance admissible or by directly selecting the distance corresponding to the desired number of clusters [21].

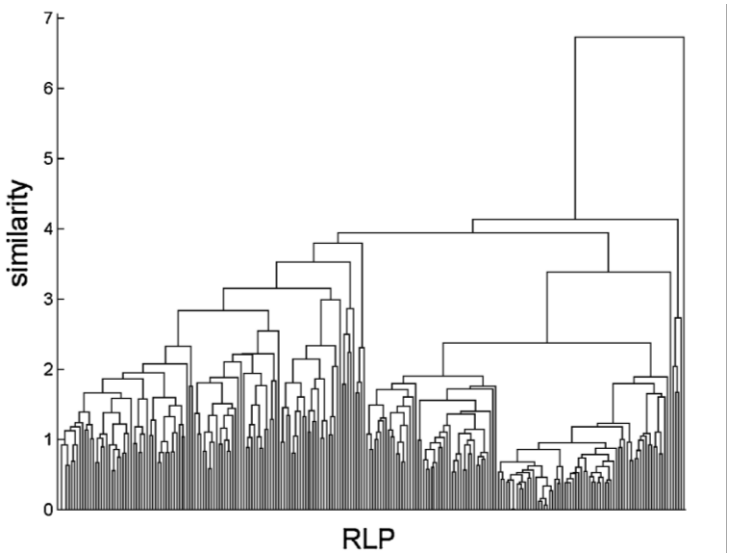


Figure 3 - Dendrogram of the hierarchical clustering with average distance criterion. Horizontal axis: RLP identifier. Vertical axis: similarity measure (5) between clusters *[21]*.

In the Ward linkage criterion, the clusters are formed in order to minimize the increase of the within-cluster sums of squares. The similarity between the two clusters $X^{(s)}$ and $X^{(t)}$ is measured as the increase of these squares sums if the two clusters were merged as it shown in the following equation:

$$\gamma_W^{(s,t)} = \frac{n^{(s)}n^{(t)}}{n^{(s)}+n^{(t)}} d^2(c^{(s)}, c^{(t)}) \quad (1.5)$$

Where $c^{(s)}$ and $c^{(t)}$ are the centroids of the two clusters. Once two clusters $X^{(s)}$ and $X^{(t)}$ have been merged to form $X^{(w)}$, the similarity between the new cluster $X^{(w)}$ and another cluster $X^{(g)}$ becomes as it shown in the following equation:

$$\gamma_W^{(w,g)} = \frac{(n^{(s)}+n^{(g)})\gamma_W^{(s,g)}+(n^{(t)}+n^{(g)})\gamma_W^{(t,g)}-n^{(g)}\gamma_W^{(s,t)}}{n^{(s)}+n^{(t)}+n^{(g)}} \quad (1.6)$$

Figure 4 shows the dendrogram obtained by using the Ward linkage criterion. The comparison between the two hierarchical trees shows that the average distance criterion forms large clusters of similar RLPs and rejects the very dissimilar ones in small or singleton clusters, whereas the Ward criterion prevents the formation of large clusters [21].

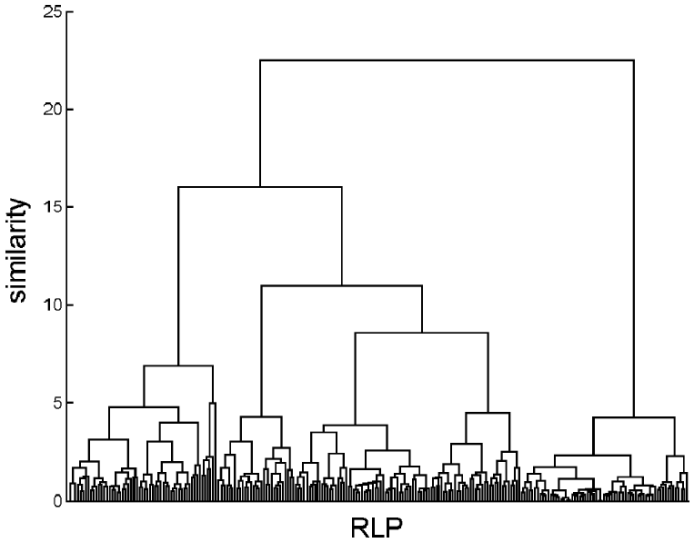


Figure 4 - Dendrogram of the hierarchical clustering with Ward linkage criterion.
Horizontal axis: RLP identifier. Vertical axis: similarity measure (7) between clusters *[21]*.

1.1.5. Self-Organized Map

The Self-Organizing has been developed by professor Kohonen [26], is one of the most popular artificial neural networks and has been proven useful in many applications [27].

To get a better understanding on what a SOM is we need to mention a few things regarding Artificial Neural Networks (ANNs.) ANNs are based on the functions of the human brain and, therefore, they consist powerful tools for modelling, especially when the underlying data relationship is unknown. Moreover, they can identify and learn correlated patterns between input data sets and corresponding target values [28], [29], [30]. They have been successfully applied in a variety of scientific fields such as mathematics, engineering, medicine, economics, meteorology, psychology, neurology and many other [28], [29]. The reason that they have been successfully applied in so many scientific fields lies in the fact that they operate in accordance with the four operating principles that are displayed below:

1. The fairly large database that is required, i.e. known inputs should be compared with their corresponding outputs in order to "educate" the network.
2. The comparison of the output value that is produced with the real one and the amendment of the weights in accordance with the "education rule".
3. The produced error that works as a guide, which decreases as the repetition is increased. It is considered that the network has been educated when the error becomes smaller than the threshold.
4. The certification that the system is adequately trained when it responds correctly to

new samples. The broad spectrum of the learning set is considered a criterion [30].

Due to the aforementioned, it is obvious that ANNs are educated through previous load patterns also taking into account other influencing factors such as weather conditions and the day of the week, as a result predicting new load patterns using recent load data [28]- [30].
The Self-Organizing Map as we mentioned is a type of ANN, however, it differs a lot from them as it applies competitive learning as opposed to the methods used for training the classical ANNs such as error-correction learning and in the sense that they use a neighbourhood function to preserve the topological properties of the input space.
The Self-Organizing Map is based on unsupervised learning, which means that no human intervention is needed during the learning and that little needs to be known about the characteristics of the input data. It provides a topology preserving mapping from the high dimensional space to map units. Map units, or neurons, usually form a two-dimensional lattice and thus the mapping is a mapping from high dimensional space onto a plane. The property of topology preserving means that the mapping preserves the relative distance between the points. Points that are near each other in the input space are mapped to nearby map units in the SOM. The SOM can thus serve as a cluster analyzing tool of high-dimensional data. Also, the SOM has the capability to generalize. Generalization capability means that the network can recognize or characterize inputs it has never encountered before. A new input is assimilated with the map unit it is mapped to [31]. A description of the basic SOM training algorithm is presented below:

Table 2 - Pseudocode for the SOM clustering algorithm [32].

Let

X be the set of n training patterns $x1, x2, .. xn$

W be a $p \times q$ grid of units w_{ij} where i and j are their coordinates on that grid

α be the learning rate, assuming values in]0,1[, initialized to a given initial learning rate

r be the radius of the neighborhood function $h(w_{ij}, w_{mn}, r)$, initialized to a given initial radius

1. Repeat
2. For $k = 1$ to n
3. For all $w_{ij} \in W$, calculate $d_{ij} = \|x_k - w_{ij}\|$
4. Select the unit that minimizes d_{ij} as the winner w_{winner}
5. Update each unit $w_{ij} \in W$: $w_{ij} = w_{ij} + \alpha\, h\,(w_{winner}, w_{ij}, r)\|x_k - w_{ij}\|$:
6. Decrease the value of α and r
7. Until α reaches 0

The neighborhood function h is responsible for the interactions between different SOM units and usually is a function that decreases with the distance (in the output space) to the winning unit. During training, each unit will become more isolated from the effects of its neighbors and as a result the radius of this function usually decreases. However, it should be noted that some SOM implementations decrease this radius to one, while others decrease it to zero. This means that the implementations with a reduction level of one will have even in the final stages of training their units affected by their nearest neighbors, while the rest that have a reduction level of zero will not have no affection at all from their neighbors [32].
The Self-Organizing Map as it is displayed in Figure 5 is a two-dimensional array of neurons. One neuron is a vector called the codebook vector.

$$M = \{m_{i1}, \dots, m_{in}\}$$

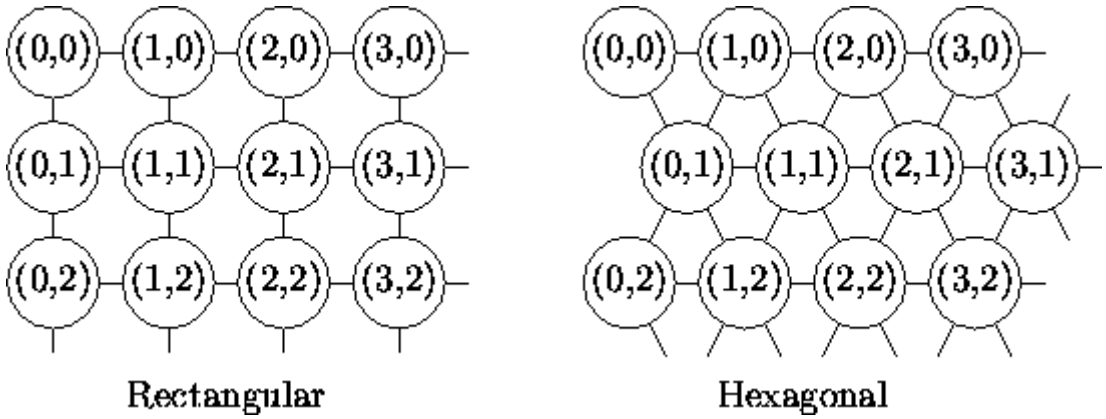


Figure 5 - Different topologies *[31]*

Moreover, the distance between the map units and the topology relations can be defined. One can also define a distance between the map units according to their topology relations. By using the concept of immediate neighbors we refer to the neurons that are adjacent. As a result the immediate neighbors belong to the neighborhood N_c of the neuron m_c. The neighborhood function should be a decreasing function of time: $N_c = N_c(t)$. Neighborhoods of different sizes in a hexagonal lattice are illustrated in Figure 6.

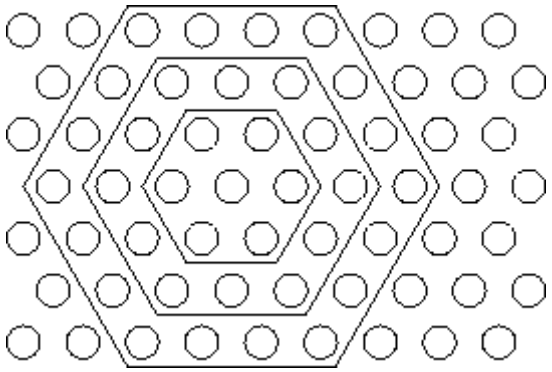


Figure 6 - Neighborhood of a given winner unit *[31]*

In the smallest hexagon, there are all the neighbors belonging to the smallest neighborhood of the neuron in the middle belonging to a hexagonal lattice. The topological relations between the neurons are left out for clarity.
In the basic SOM algorithm, the topological relations and the number of neurons are fixed from the beginning. This number of neurons determines the scale or the granularity of the resulting model. Scale selection affects the accuracy and the generalization capability of the model. It must be taken into account that the generalization and accuracy are contradictory goals. By improving the first, we lose on the second, and vice versa.

1.2. Benefits of Self Organised Map (SOM) among other Clustering Methods

The clustering process is the first process that will be always run in Divinus and the forecast that will run next will be based on the clusters that were created. As a result, the clusters must be as accurate as possible so that the forecast that will run to have as little mean error as possible. For this reason Self Organizing Maps where chosen as they can be applied in many areas including the area that we are interested which is data clustering with great precision and success.
The advantage of using this type of artificial neural network to cluster power loads is that they group the loads in terms of the uniformity of the characteristics that define them, reducing the

size of the problem to a two-dimensional map while maintaining all the information about the n features valued. In this way reducing the dimensionality and the grid clustering the data are easier to observe.
Moreover, SOM is not sensitive to initialization, as k-means, which provides a more robust learning. It preserves the topology of input data by assigning each datum to a neuron having the highest similarity, and maps into adjacent neurons the data that contains similar attributes.
However, despite the positive or the negative aspects that a clustering algorithm may have, there is no rule for the best matching clustering algorithm. In our tool based on the survey that we conducted we chose to use SOM as our clustering algorithm on whose clusters the forecast will be based.

1.3. Relational Database Management Systems

Relational Database Management System (RDMS) is responsible for defining a set of relation schema that will allow information to be stored and retrieved without unnecessary redundancy. RDMS consists a subset of Database Management System (DBMS) which in turn is a database program. From a technical point of view, a database program is a software system that through a standard method catalogs, retrieves and runs queries on data. Furthermore, a DBMS also manages, organizes and provides ways for the incoming data to be modified and/or to be extracted by other programs or users. However, databases in the early days were relatively "flat," which means they were limited to simple rows and columns, like a spreadsheet. With the passing of the time, the majority of the databases used in application to store or retrieve data were made relational. In addition, relational databases allow users to access, update, and search information based on the relationship of data stored in different tables but also allow them to run queries that involve multiple databases. As a result, because nowadays almost all the databases used are relational, the terms "database" and "relational database" are used most of the times synonymously [33].

Nowadays however, users are given more options regarding the databases they use. Depending solely on how the data will be used, they can store them in SQL or NoSQL databases. To start with, SQL database was created back in 1975 by IBM, the initial letters stand for "Structured Query Language" and it is a query language used for accessing and modifying information in a database. The most common commands that can be found in SQL include "Insert", "Update" and “Delete" and it is mostly used for Web database development and management. Moreover, by using scripting languages such as PHP we are given the opportunity to execute SQL commands from a web page. Therefore, because of the possibilities SQL has given it is possible to display different information on each webpage [33].

On the other hand, NoSQL which originally meant "non SQL" or "non relational" has existed since the late 1960s, but gained popularity and necessity in the early twenty-first century triggered by the needs of Web companies such as Google, Amazon and Facebook [34]. NoSQL is a non-relational database that stores and accesses data using key-values. This means that NoSQL databases store data without using the classical means such as rows and columns to which the data are stored but rather identify each data individually with the use of a unique key. Furthermore, **NoSQL is a more flexible database compared to relational databases as it does not require a structured schema that defines each table separately** [33].

Moreover, while relational databases (like SQL) are ideal for storing structured data, their rigid structure makes it difficult to add new fields and quickly scale the database. NoSQL provides an unstructured or "semi-structured" approach that is ideal for capturing and storing user generated content (UGC). This may include text, images, audio files, videos, click streams, tweets, or other data. While relational databases often become slower and more inefficient as they grow, NoSQL databases are highly scalable. In fact, you can add thousands or hundreds of thousands of new records to a NoSQL database with a minimal decrease in performance [33]. Therefore, NoSQL flexibity and scalability has led many large businesses and organizations to start using NoSQL databases for the storage of their data. NoSQL databases are especially common in applications such as cloud computing and are becoming even more popular as storing solutions for big data applications.

Due to the aforementioned, we understand that there are many database management systems available and it is very important for them to be able to communicate with each other. The solution to this problem comes with the name of **Open Database Connectivity (ODB**C) which is a driver that allows databases to integrate to others. In order to give a description of how the ODBC we should have a look at the common SQL statements such as "Insert", "Select", "Update" and “Delete". These statements through ODBC are translated from a program's proprietary syntax into a syntax that other databases can understand.

In the tool that we developed the use of a Relational Database Management System was necessary. For this reason, we present through a brief description the most widespread and known databases such as Oracle Database 12c, Microsoft SQL, MySQL, IBM DB2, SQLite, MariaDB, Teradata as well as PostgreSQL which is the one that we chose to use.

1.3.1.Oracle Database 12c

Figure 7 - Oracle Database 12c Logo [35]

Oracle began its journey in 1979 as the first commercially available relational database management system (RDBMS) and today it supports a wide range of operating systems multiple versions of Windows and multiple Unix and Linux variations. Oracle's name is synonymous with enterprise database systems, unbreakable data delivery and fierce corporate competition from CEO Larry Ellison. Powerful but complex database solutions are the mainstay of this Fortune 500 Company [35].

The current release of Oracle's RDBMS is Oracle 12c. The "c" stands for cloud and is reflective of Oracle's work in extending its enterprise RDBMS to enable firms to consolidate and manage databases as cloud services when needed via Oracle's multitenant architecture and in-memory data processing capabilities. Furthermore, there is an abundance of tools for Oracle database administration, application development and data movement/management. In terms of functionality, Oracle keeps pace with many new and advanced features such as JavaScript Object Notation (JSON) support, temporal capabilities, multi-tenancy and new database options such as Oracle Database that uses in-memory columnar technology to enable enterprises to easily and transparently accelerate the performance of their business analytics [36].

Oracle heavily promotes its database appliance, Exadata, which combines software and hardware engineered in order to provide a high-performance and high-availability platform for running Oracle Database. Its architecture features a scale-out design with industry-standard servers and intelligent storage, including flash technology and a high-speed InfiniBand internal fabric. Elastic configurations enable systems to be tailored to specific database workloads, including online transaction processing (OLTP), data warehousing, in-memory analytics and mixed workloads. The key selling point of a database appliance is that it's easy to deploy and includes all of the needed components to run the DBMS [36].

Oracle 12c Release 1 will be fully supported by Oracle through the end of July 2018, and a newer update, Oracle Database 12c Release 2 (12.2), became available in early March 2017. From a cost perspective, Oracle has a reputation as being expensive to license and support. Additionally, according to surveys conducted at Gartner's annual IT Financial Procurement & Asset Management summits in North America and Europe, Oracle ranked lowest in terms of ease of doing business [36].

1.3.2.IBM DB2

Figure 8 - IBM DB2 [35]

DB2 is Oracle's biggest competitor on Unix and Linux operating systems. DB2 11.1 The latest release of DB2, runs on Linux, UNIX, Windows, the IBM iSeries and mainframes. IBM has pitted its DB2 system squarely in competition with Oracle's, via the International Technology Group, and the results showed significant cost savings for those that migrate to DB2 from Oracle which is translated into 34 percent to 39 percent for comparative installations over a three-year period. In addition to these two platforms, DB2 is available on Windows, z/OS mainframe and iSeries midrange servers. The latest versions of DB2 are DB2 Version 11 for Linux, Unix, Windows (LUW), DB2 11 for z/OS and DB2 for i v7.2. DB2 SQL is almost identical between the z/OS and LUW platforms, but administratively there are significant differences. Likewise, many development, data movement and DBA tools are available for DB2, both from IBM and other independent software vendors (ISVs) [36].
In terms of functionality, DB2 is regularly revised and updated with market-leading features, including JSON support, temporal capabilities, shadow tables and advanced compression being among the recent advances. With the DB2 SQL compatibility feature, IBM delivers the ability to run Oracle applications in DB2 for LUW with no changes to business logic in the client code, triggers or stored procedures. Feature-wise, it would be remiss not to mention IBM's next-generation database technology for DB2 called BLU Acceleration. It provides a combination of in-memory performance techniques, compression capabilities and column store capabilities. As is the case with Oracle, IBM regularly publishes benchmark results for DB2. As with any benchmark, it's always advisable to perform your own performance benchmarks on your own systems and workload if possible. IBM offers a database appliance called the PureData System, which provides single part procurement including pre-installed and configured DB2. The system is ready to load data in hours and provides open integration with third-party software. PureData comes with an integrated management console for the entire system, a single line of support, integrated system upgrades and maintenance. The PureData System is available in different models that have been designed, integrated and optimized for analytics, operational analytics and transaction processing [36].

1.3.3.Microsoft SQL

Figure 9 - Microsoft SQL Server Logo [35]

Microsoft is the most profitable technology company and the SQL server helped a lot to put it there. It is almost certain that, Microsoft's desktop operating system is everywhere, but if you're running a Microsoft Windows-based server, you're likely running SQL Server on it. SQL Server's ease of use, availability and tight Windows operating system integration makes it an easy choice for firms that choose Microsoft products for their enterprises. Microsoft promotes the latest release, SQL Server 2016, as the platform for both on-premises and cloud databases and business intelligence solutions.
Microsoft promotes SQL Server 2016 in helping enterprises build mission-critical applications with high-performance, in-memory security technology across OLTP (online transaction processing), data warehousing, business intelligence and analytics.
The most recent release of Microsoft SQL Server is Microsoft SQL Server 2016 SP1 (v13.0.4001.0), which debuted on November 15th, 2016. Microsoft is currently developing SQL Server 2017, codenamed SQL Server vNext, but no release date has been announced for the upcoming version at this time.
From a technology and functionality standpoint, Microsoft keeps abreast with the market. Features added to the latest version include stretch database capabilities for integrating on-premises with cloud, strong encryption capabilities, integration of Hadoop with relational data using the Polybase feature and improved in-database analytics capabilities. With Azure, Microsoft's cloud-integration vision for SQL Server is the strongest of the big three DBMS vendors, including simplified backup to Azure and the ability to set up an Azure virtual machine as an always-on secondary. Microsoft boasts strong performance benchmark results for SQL

Server 2016, including TPC-E, which measures modern OLTP workloads and TPC-H, which measures data warehousing workloads.
However, Microsoft lacks a database appliance like Oracle's Exadata and IBM's PureData System. Therefore, if a user is looking for a pure plug-and-play database appliance, Microsoft isn't a realistic option. However, there are third-party appliances that embed SQL Server, and Microsoft also offers the Microsoft Analytics Platform System, an analytics appliance that integrates SQL Server with data from Hadoop.

1.3.4.Teradata

Figure 10 - Teradata Logo *[35]*

Teradata was founded as early as the late 1970s, and it laid the groundwork for the first data warehouse before the term even existed. Teradata is known mostly for its analytics and data warehousing capabilities. For organizations looking to run analytical processes, the Teradata Database and the company's Active Enterprise Data Warehouse offers a gateway to organizational knowledge based on advanced in-database analytics, intelligent in-memory processing, parallel in-database execution of scripting languages, native JSON support and transparent single query, multi-system processing. Teradata created the first terabyte database for Wal-Mart in 1992. Since that time, data warehousing experts almost always say Teradata in the same sentence as enterprise data warehouse. The version 15.10 of its RDBMS was released by Teradata in early 2015 [35] ,[36].

1.3.5.MySQL

Figure 11 - MySQL Logo *[35]*

MySQL began as a niche database system for developers but grew into a major contender in the enterprise database market and was sold to Sun Microsystems in 2008. Since then MySQL has since become part of the Oracle empire and being more than just a niche database now, MySQL powers commercial websites by the hundreds of thousands, and it also serves as the backend for a huge number of internal enterprise applications. Today MySQL remains a very popular option for use in Web applications and continues to serve as a central component of the LAMP open-source Web application software stack, along with Linux, Apache and PHP (or Python or Perl). At the same time, MySQL has seen support from users and developers erode over the last few years following the acquisition by Oracle [35].
MySQL's decline has helped fuel the adoption of other open-source database options and forks of MySQL like the fully-open source MariaDB, which doesn't feature closed-source modules like some of those found in newer versions of MySQL Enterprise Edition, as well as Percona and the cloud-optimized Drizzle database system. MySQL Community Server 5.7.x is the most current release of the MySQL database system, with v5.7.19 having made its debut in July 2017 [35].

1.3.6.MariaDB

Figure 12 - MariaDB Logo *[35]*

MariaDB was created in 2009 by the original developers of MySQL, who created the fork following concerns over MySQL's acquisition by Oracle. It is used by tech giants like Wikipedia, Facebook, and even Google. MariaDB is a database server that offers drop-in replacement functionality for MySQL. MariaDB has seen its popularity explode recently at the expense of MySQL, particularly in its support by popular Linux distributions. In 2013 alone, Red Hat Enterprise Linux (RHEL) ditched MySQL for MariaDB, Fedora opted for MariaDB over MySQL in its Fedora 19 release, and both openSUSE and Slackware Linux made similar switches to MariaDB over MySQL. Wikipedia also adopted MariaDB over MySQL as its backend database in 2013.

Another key factor in moving MariaDB ahead of MySQL is its enhanced query optimizer and other performance-related improvements, which give the database system a noticeable edge in overall performance compared to MySQL. Last but not least, security is a top concern and priority for MariaDB. Therefore, in each solution release, the developers also merge in all of MySQL's security patches and enhance them if need be.

The most recent "stable" release of MariaDB Enterprise Server is version 10.2 (v10.2.6 debuted May 23, 2017), also known as the MariaDB Server 2017 release. The 10.x releases add better protection for data against application and network-level attacks and also enables fast delivery of new, high-performance applications.

1.3.7.PostgreSQL

Figure 13 - PostgreSQL Logo [35]

POSTGRES, now known as PostgreSQL, is considered to be the most advanced open-source database available today. **PostgreSQL, is an open-source object-relational database management system (ORDBMS)** that hides in such interesting places as online gaming applications, data center automation suites and domain registries. PostgreSQL also enjoys some high-profile duties at Skype and Yahoo! PostgreSQL is in so many strange and obscure places that it might deserve the moniker, "Best Kept Enterprise Database Secret." PostgreSQL's current stable release is PostgreSQL 9.6.3, which was released in late May 2017, and PostgreSQL 10 is expected to debut in the second half of 2017, with PostgreSQL 10 Beta 2 available now. PostgreSQL runs on a wide variety of operating systems, including Linux, Windows, FreeBSD and Solaris. And as of OS X 10.7 Lion, Mac OS X features PostgreSQL as its standard default database in the server edition. PostgreSQL benefits from more than 25 years of development as a free, open-source database system, and it includes enterprise-grade features comparable to Oracle and DB2 such as full ACID compliance for transaction reliability and Multi-Version Concurrency Control for supporting high concurrent loads.

1.4. Benefits of PostgreSQL among other Relational Database Management Systems

To choose the ideal database was to choose the one that would best fit for the needs of our tool. Therefore, we relied on criteria that would help us delimit the options we had in order to make the most accurate database choice for our tool.
The criteria that we relied on our decision were two. The first criterion, based on the type of data we had, stated that the database should be a relational one. That meant that only Structured Query Language (SQL) databases were accepted and as a result the NoSQL databases automatically withdraw from competition. The second criterion stated that the relational database that would be used should be low cost. This criterion restricted our choices even further as now we had only the option of open-source relational databases. Therefore, we had to find the most appropriate open-source relational database solution for our tool and that why PostgreSQL was chosen.
After 15 years of active development and having a reliable architecture that ensures data integrity and correctness, PostgreSQL is not just a relational database but rather a powerful object-relational database. To start with, there is an extensive list of data types that PostgreSQL supports such as Integer, Numeric, Boolean, Char, Varchar, Date, Interval and Timestamp [37]. Besides those date types, PostgreSQL boasts uuid, monetary, enumerated, geometric, binary, network address, bit string, text search, xml, json, array, composite and range types, as well as some internal types for object identification and log location. To be fair, open databases such as MySQL and MariaDB each have some of these to varying degrees, but only PostgreSQL supports them all [38].
Furthermore, PostgreSQL is highly scalable both in the sheer quantity of data it can manage as well as in the number of concurrent users it can accommodate. There are active PostgreSQL instances in production environments that are able to manage many terabytes of data, as well as clusters managing petabytes [37]. However, open databases such as MySQL and MariaDB are notorious for their 65,535 byte row size limit. Typically the data size is limited by the operating system file size limit. Because PostgreSQL can store table data in multiple smaller files, it can get around this limitation - though, it is important to note that too many files may negatively impact performance. MySQL and MariaDB do, however, support more columns per table (up to 4,096 depending on the data type) and larger individual table sizes than PostgreSQL, but it is in rare conditions that the existing PostgreSQL limits would need to be exceeded [38]. The PostgreSQL limits are displayed in the Table 3 below.

Table 3 - PostgreSQL Limits and Values *[37]*

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Due to the aforementioned and to the tolerable limitations it contains PostgreSQL has won praise from its users and industry recognition, including the "Linux New Media Award for Best Database System" and five time winner of the "The Linux Journal Editors' Choice Award" for best DBMS [37].

PostgreSQL is both a standard compliant and a highly customizable database that offers a wide range of features. It prides itself in standards compliance as its SQL implementation strongly conforms to the ANSI-SQL:2008 standard. Moreover, it has full support for subqueries (including subselects in the FROM clause), read-committed and serializable transaction isolation levels. And while PostgreSQL has a fully relational system catalog which itself supports multiple schemas per database, its catalog is also accessible through the Information Schema as defined in the SQL standard. Another standard compliance feature is its data integrity features that include (compound) primary keys, foreign keys with restricting and cascading updates/deletes, check constraints, unique constraints, and not null constraints [37].
Other open-source databases such as MySQL and MariaDB are doing a lot to be SQL standard compliant with the InnoDB/XtraDB storage engines. They now offer a STRICT option using SQL modes, which determines the data validation checks that get used; however, depending on the mode we use, invalid and sometimes silently-truncated data can be inserted or created on update. Neither of these databases currently supports check constraints and there are also a host of caveats for foreign key constraints. Additionally, data integrity may suffer significantly depending on the storage engine selected. MySQL (and the MariaDB fork) has made no secret that they have long made tradeoffs for speed and efficiency over integrity and compliance [38].
Last but not least, PostgreSQL has customizable features through which it is able to run stored procedures in more than a dozen programming languages, including Java, Perl, Python, Ruby, Tcl, C/C++, and its own PL/pgSQL, which is similar to Oracle's PL/SQL. Moreover, besides the standard function library that is included, there are the hundreds of built-in functions that range from basic math and string operations to cryptography and Oracle compatibility. PostgreSQL also includes a framework that allows developers to define and create their own custom data types along with supporting functions and operators that define their behavior [37].
Best of all PostgreSQL features though is that its source code is available under a liberal open source license: the PostgreSQL License. This license provides the users with the freedom to use, modify and distribute PostgreSQL in any form they like, open or closed source. As such, PostgreSQL is not only a powerful database system capable of running the enterprise, it is a development platform upon which to develop in-house, web, or commercial software products that require a capable RDBMS [37].
In the tool that we built it is more likely that most of the aforementioned advanced features will not be used, but since data needs can evolve quickly, there is an undoubtedly clear benefit to having them as our database capabilities. Therefore, due to the wide variety of capabilities, the extensive data capacity, the data integrity and its exceptional documentation that can guide experienced or fresh users in its use, PostgreSQL was chosen to be the relational database of our tool.

1.5. Programming Languages for the Developement Unsupervised Clustering and Forecasting Tools through Machine Learning

As the integration of internet in our lives rises, whether this is a good or a bad outcome, the integration of information technology in mores areas of our lives also rises. Along with the rise of internet and information technologies there is also an increase in the amounts of data retrieved and as a result an increase in the importance of processing those data in large scales. Based on recent estimates, 2.5 quintillion (10^{18}) bytes of data are generated on a daily basis. In order to get an understanding of the amount of data available nowadays we only need to realize that 90 percent of the information that we store nowadays was generated in the past decade alone. It is made obvious that this amount of data is beyond the means of standard analytical methods or it is simply too vast for humans limited minds to even comprehend. In order to cope with this infinite amount of data machine learning was developed. Through Machine Learning, we enable computers to process, learn from, and draw actionable insights out of the otherwise impenetrable walls of big data [39]. The goal of this section is to deliver a comparison of five programming languages, which are C/C++, Java, R, JavaScript and Python in order to determine the most appropriate of them in order to be used for electricity clustering and forecasting in a microgrid level using machine learning technology.

1.5.1.C/C++

C is a general-purpose, imperative computer language and was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs and used to re-implement the Unix operating system. Since then it has become one of the most widely used programming languages of all time, with C compilers from various vendors available for the majority of existing computer architectures and operating systems [40]. C++ on the other hand, is a middle-level programming language that was also developed at Bell Labs by Bjarne Stroustrup in 1979. The purpose of its creation was to bypass the difficulties of analyzing UNIX kernel for distributed systems that arose using other available programming languages that were either too slow or low level. The development of C++ was based on C because it was a general purpose language, very efficient as well as fast in its operations. Nowadays, C++ is ranked 4th in popularity according to 2017 IEEE spectrum Top Programming Language ranking [41].
C/C++ is ideal for low-level software such as operating system components and networking protocols where computational speed and memory efficiency are extremely critical. For these same reasons, it is also a popular choice for implementing the guts of Machine Learning procedures. However, its lack of idiomatic abstractions for data processing and added overhead for memory-management can make it unsuitable for beginners, and burdensome for developing complete end-to-end systems. In either case, there is no dearth of Machine Learning libraries available in C/C++, e.g. LibSVM, Shark and mlpack [42].

1.5.2.JAVA

Java was developed by James Gosling at Sun Microsystems as a general-purpose computer-programming language that was concurrent, class-based and object-oriented. It was released in 1995 as a core component of Sun Microsystems' Java platform and derived much of its syntax from C and C++, but it had fewer low-level facilities than either of them [43]. Java became the software engineer's language of choice because of its clean and consistent implementation of object-oriented programming, and platform-independence using JVMs. **It sacrifices brevity and flexibility for clarity and reliability, which makes it popular for implementing critical enterprise software systems.** In order to maintain that same level of reliability and to avoid writing messy interfaces, companies that have been using Java may prefer to stick to it for their Machine Learning needs [42].

1.5.3.R

R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. It was named partly after the first names of the first two R authors and partly as a play

on the name of S as it started as an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme. S was created by John Chambers in 1976, while at Bell Labs. The R project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000 [44].
R is a GNU package. The source code for the R software environment is written primarily in C, Fortran, and R. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available [44].
R is used for statistical computing and is a clear winner for large-scale data-mining, visualization and reporting. It provides an easy access to a huge collection of packages that enable the users to apply almost all kinds of Machine Learning algorithms, statistical tests and analysis procedures. The language itself has an elegant—albeit esoteric—syntax for expressing relationships, transforming data and performing parallelized operations [42].

1.5.4.JavaScript

JavaScript was deployed for the first time in 1995 in the Netscape Navigator 2.0 beta. Until it came to the name we know it today it had changed quite a bit. It started during its development as Mocha, then it was officially named as Livescript when it first shipped in beta releases of Netscape Navigator 2.0 and then changed again to the one that we know today [45].
JavaScript often abbreviated as JS, is a high-level, interpreted programming language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm. Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content engineering. It is used to make webpages interactive and provide online programs, including video games. The majority of websites employ it, and all modern web browsers support it without the need for plug-ins by means of a built-in JavaScript engine. Nowadays JavaScript is evolving with a rapid speed as it can be found on mobile **devices, desktop applications, embedded systems and backend applications. Therefore, due to the wild range of its usage it can be used even in machine learning applications. Perhaps it's not the best idea to train machine learning models in the browser but using pre-trained models in the browser might be a promising field in the future and** it can be used as the bridge for the web developers to enter the field of machine learning [45].

1.5.5.Python

Python got a definite seat among the modern high-languages as a general purpose programming language. It was invented in the early 90s in CWI Netherlands by Guido Van Rossum in an effort to find an alternative for the ABC language [46]. Python is one of the most popular programming languages for machine learning and data science and therefore enjoys a large number of useful add-on libraries developed by its great community. Although the performance of interpreted languages, such as Python, for computation-intensive tasks is inferior to lower-level programming languages, extension libraries such as NumPy and SciPy have been developed that build upon lower layer Fortran and C implementations for fast and vectorized operations on multidimensional arrays. For machine learning programming tasks, we will mostly refer to the scikit-learn library, which is one of the most popular and accessible open source machine learning libraries as of today [39].

1.6. Benefits of Python among other Programming Languages regarding Clustering Methods

The most decisive factor when selecting a language for machine learning is the type of project that it will be used. In a survey, the results of which are displayed in Figure 14, developers where asked would their choice be for machine learning languages in 17 different application

areas. Python was the programming language with the highest popularity, for machine learning developers and data scientists, among the other languages regarding machine learning [47].

Figure 14 - Popularity of Machine Learning Languages [48]

Python leads the pack, with 57% of data scientists and machine learning developers using it and 33% prioritizing it for development. This fact should not surprise us based on the fact that there is a huge evolution in deep learning Python frameworks over the past 2 years, including the release of TensorFlow and a wide selection of other libraries. Python ratio of usage is 57% which is the highest ratio making it a primary choice for machine learning language among the other five languages [47].

In addition, given all the evolution, Python is often compared to R, but they are nowhere near comparable in terms of popularity: R comes fourth in overall usage (31%) and fifth in prioritization (5%). R is in fact the language with the lowest prioritization-to-usage ratio among the five, with only 17% of developers who use it prioritizing it. This means that in most cases R is a complementary language, not a first choice in opposition to Python. Furthermore, Python is ahead of many highly preferred languages such as C/C++ which is found second, both in usage (44%) and prioritization (19%). Java follows C/C++ very closely, while JavaScript comes fifth in usage, although with a slightly better prioritization performance than R (7%) [47]. Moreover, those who responded to the survey about other programming languages used in machine learning application also suggested the usual suspects of Julia, Scala, Ruby, Octave, MATLAB and SAS, but they all fall below the 5% mark of prioritization and below 26% of usage [47].

Machine learning scientists working on sentiment analysis prioritise Python (44%) and R (11%) more and JavaScript (2%) and Java (15%) less than developers working on other areas. In contrast, Java is prioritised more by those working on network security / cyber-attacks and fraud detection, the two areas where Python is the least prioritised. Network security and fraud

detection algorithms are built or consumed mostly in large organisations—and especially in financial institutions—where Java is a favourite of most internal development teams. In areas that are less enterprise-focused, such as natural language processing (NLP) and sentiment analysis, developers opt for Python which offers an easier and faster way to build highly performing algorithms, due to the extensive collection of specialised libraries that come with it [47]. C/C++ is mostly favoured for Artificial Intelligence (AI) in games (29%) and robot locomotion (27%) which are two areas where the level of control, high performance and efficiency are required. Therefore, a lower level programming language such as C/C++ that comes with highly sophisticated AI libraries is a natural choice, while R, designed for statistical analysis and visualizations, is deemed mostly irrelevant and is therefore prioritized in the lower position in AI followed by speech recognition where the case is similar [47].

Although surveys can indicate a programming language being more appropriate for an application than another, there is no rule for the best machine learning language. In our application based on the survey and on the fact that our effort of creating a predictive tool is our maiden journey in machine learning, **Python will be used as the best option, given its wealth of libraries and ease of use.**

2. Electrical Load Data

Nowadays, data are the most important source in each program and they are the basis of its success. This means that if the data we import are good then the system we are making is likely to work properly and to have a high precision. On the other hand if the data we put into the system are incomplete, or lack the precision required than the most likely scenario is that the system that is implemented will have limited precision and will not function properly.

2.1. Data Retrieval from the Administrator of the Greek Electricity Distribution Network

Taking the aforementioned under consideration and after searching the best possible ways to retrieve the data we came to the decision to get our data from the Administrator of the Greek Electricity Distribution Network (Greek: Διαχειριστής Ελληνικού Δικτύου Διανομής Ηλεκτρικής Ενέργειας, or ΔΕΔΔΗΕ) which was formed by the separation of the Distribution Department of Greece's Public Power Corporation in order to comply with the 2009/72/EC EU Directive relative to the electricity market organization. Its mission is to assume the responsibilities of the Distributor for the Network Operation of Greece. It is a 100% subsidiary of the public power corporation (Greek: ΔΕΗ), however, it is independent, maintaining all the independence requirements embodied in the above legislative framework. Therefore, through the smart metering system that it contains we were able to obtain past data of both active and reactive loads of the Technological Institute of Sterea Ellada for the Chalkis location only by entering in the following site:

https://meteringnet.deddie.gr/login.aspx?ReturnUrl=%2fbilling.aspx

Figure 15 - DEDDIE Login Page

After we have successfully logged in the website we were able to choose the time periods of the loads on which we would rely our solution. In order to have a sufficient amount of data we draw data from January 1, 2010 to January 31, 2018. From the data retrieved 95% will be used for training and the rest of them for testing as it will be explained in chapter 5.
We are given the ability to retrieve the data per hour or per quarter. Based on the Divinus requirements we decide to retrieve the data per hour.

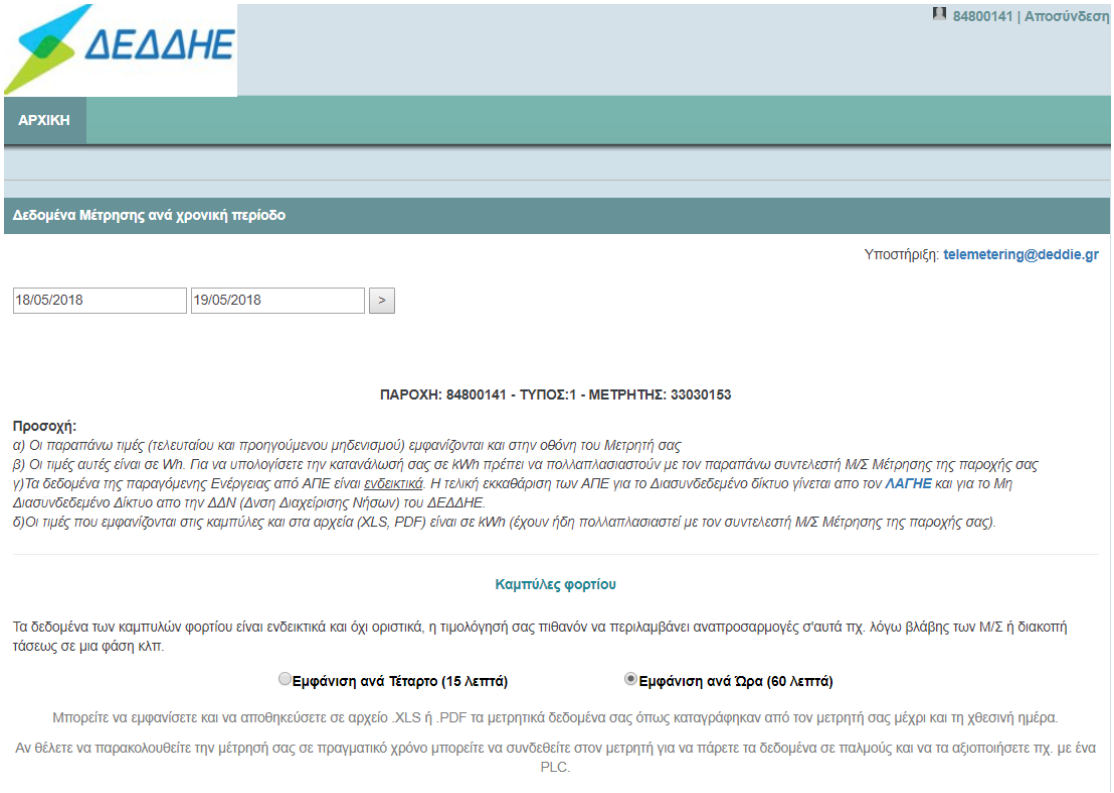


Figure 16 - DEDDIE Site where we choose the data we want to retrieve and the time unit to which the data will be retrieved

As soon as the data are ready to be downloaded we are able to view two graphs one for the active and one for the reactive load as it is shown in Figure 16.

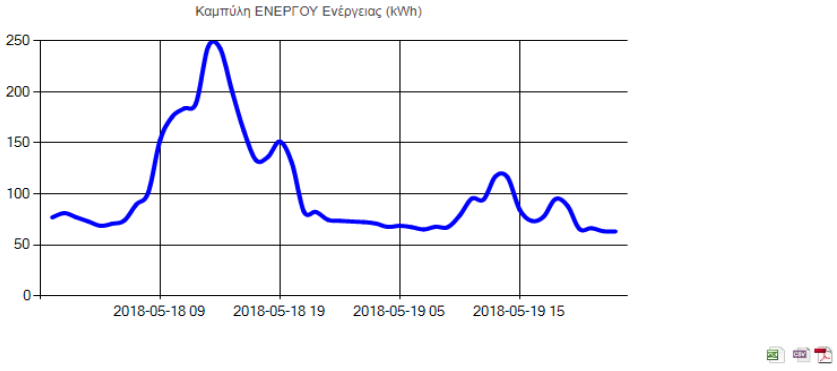


Figure 17 - Active Load Graph Displaying the Selected Time Period

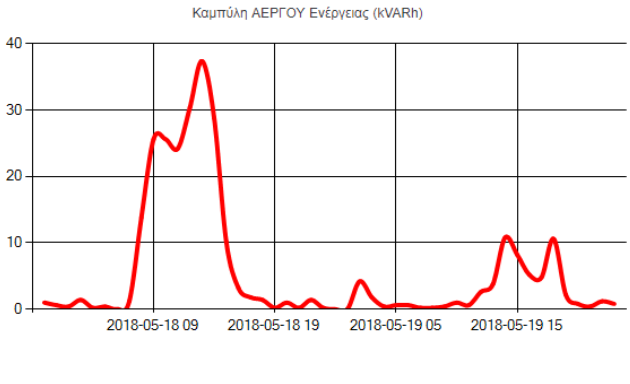


Figure 18 - Reactive Load Graph Displaying the Selected Time Period

These data can be exported from the site in three possible formats: a) excel b) csv and c) pdf. The pdf format was automatically excluded due to the fact that it is a non-manageable format type. Therefore, the excel and the csv formats are the only that we could handle. We will proceed with the excel format.

Ημερομηνία	Ενέργεια
2018-05-18 00	77
2018-05-18 01	81,2
2018-05-18 02	77
2018-05-18 03	73
2018-05-18 04	68,8
2018-05-18 05	70,8
2018-05-18 06	73,8
2018-05-18 07	89,6
2018-05-18 08	101,2
2018-05-18 09	153
2018-05-18 10	175,6
2018-05-18 11	183,8
2018-05-18 12	189,2
2018-05-18 13	244,2
2018-05-18 14	242,8
2018-05-18 15	200,6
2018-05-18 16	161,2
2018-05-18 17	133
2018-05-18 18	136,4
2018-05-18 19	151,4
2018-05-18 20	130
2018-05-18 21	82,6
2018-05-18 22	82,4
2018-05-18 23	74,6

Figure 19 - DEDDIE Excel Format

```
Ημερομηνία;Ενέργεια(kWh)
2018-05-17 23;21,2
2018-05-18 00;76
2018-05-18 01;80,4
2018-05-18 02;77,4
2018-05-18 03;69,4
2018-05-18 04;68,2
2018-05-18 05;74,6
2018-05-18 06;73,2
2018-05-18 07;92,8
2018-05-18 08;117,4
2018-05-18 09;153,4
2018-05-18 10;183,8
2018-05-18 11;180,8
2018-05-18 12;197,6
2018-05-18 13;253,2
2018-05-18 14;237,4
```

Figure 20 - DEDDIE CSV Format

As soon as we have downloaded the data required for Divinus we are able to proceed to the next step which is the insertion of these data to our program's database.

2.2. Data Insertion in Divinus PostgreSQL Database

The next thing that needs to be done after the data have been successfully downloaded is the insertion to the database and as a result to Divinus itself. This action is performed with the use of a python library called pandas.
Pandas is an open source, software library written for Python and is used for data manipulation and analysis. Its name derives from the term "panel data" and as it is expected it offers solutions regarding data structures and operations for manipulating numerical tables and time series. It is easy to use and a very useful tool when it comes to the management of a wide variety of data.
The first use of pandas is to enter the data in Divinus in order to check whether they are properly structured. In case they are not, corrective actions are performed on the data and then they are store in the database. Table 4 contains the code that performs the aforementioned actions.

Table 4 - DEDDIE Power Loads Code

```
'''
Created on Mar 4, 2018

@author: dimitris mele
'''

import os
import threading
import pandas as pd
from glob import glob
from Database.Insert import insert_to_ap, insert_to_rp
from Core.XLS_Removal import XLS_Removal_Energeia, XLS_Removal_Aerga

def Deddie_active_power_data():

    for file in glob(r'C:\Users\dimit\Downloads\Loads\Energeia___*.*'):
        directory = (os.path.abspath(file))
        print("-------------------------------------------------------------
-------------------------")
```

```
        print ('A file found: {}'.format(directory))
        print ("")

        get_ap, = pd.read_html(directory, thousands='.', decimal=',',
header=0)

        get_ap.rename(columns={'Î–Î¼ÎµÏ•Î¿Î¼Î·Î¯Î±':'date_time',
'Î•Î¼Î-Ï•Î³ÎµÎ¹Î±':'active_power_kwh'}, inplace=True)
        get_ap['date_time'] = get_ap['date_time'].apply(pd.Timestamp)

        for i in range(len(get_ap)):
            insert_to_ap(get_ap['date_time'][i],
get_ap['active_power_kwh'][i])
        print("Data were successfully inserted in the database")
        print("----------------------------------------------------------
-------------------------")

        XLS_Removal_Energeia()

    seconds=1.0
    minutes=seconds*60
    hour=minutes*60

    threading.Timer(hour, Deddie_active_power_data).start()

Deddie_active_power_data()
```

The aforementioned code runs every hour. This means that every hour it will search at (C:\Users\dimit\Downloads\Loads\) which is the location where the DEDDIE files are saved. Once it identifies files whose names start with "Energeia" or "Aerga" it will try to integrate them through pandas in Divinus, perform corrections wherever they are needed and store them in the database. The integration into the database is performed through the SQL insert functions.

Table 5 - SQL Insertion Command for the Active Power Loads Implemented through Python

```
def insert_to_ap(date_time, active_power_kwh):
    conn=psycopg2.connect ("host='localhost' dbname='postgres'
user='postgres' password='123456q!'")
    cur=conn.cursor()
    cur.execute ("INSERT INTO active_power VALUES (%s,%s) ON CONFLICT
(date_time) DO NOTHING",(date_time, active_power_kwh))
    conn.commit()
    conn.close()
```

The insertion functions are implemented by inserting and executing the SQL commands through python. In order for the SQL commands to work in python the first thing that needs to be done is to set the information regarding the database in which they will be saved. The information required are the host, the database name, the user and the password. By giving these information we are able to log into the database and define the command we want to execute. In our case the command we want to execute is the insertion command and it will be executed as follows:

INSERT INTO **TABLE** VALUES (%s,%s) ON CONFLICT (**VALUE**) DO NOTHING

The bold words in the SQL command should be replaced by the corresponding table and the corresponding table column.

INSERT INTO **active_power** VALUES (%s,%s) ON CONFLICT (**date_time**) DO NOTHING

The last step in the data retrieval process is the deletion of the files after the data have been successfully inserted in the database. Therefore, as soon as the files are successfully inserted into the database the functions XLS_Removal_Energeia() starts running. Its purpose is to delete the downloaded xlsx files from the directory that they are stored in order to release computing resources.

Table 6 - XLSX Removal Code

```
'''
Created on 8 Μαρ 2018

@author: d.mele
'''

import os
from glob import glob

def XLSX_Removal_Energeia():

    for file in glob(r'C:\Users\dimit\Downloads\Loads\Energeia___*.*'):
        os.remove(file)
```

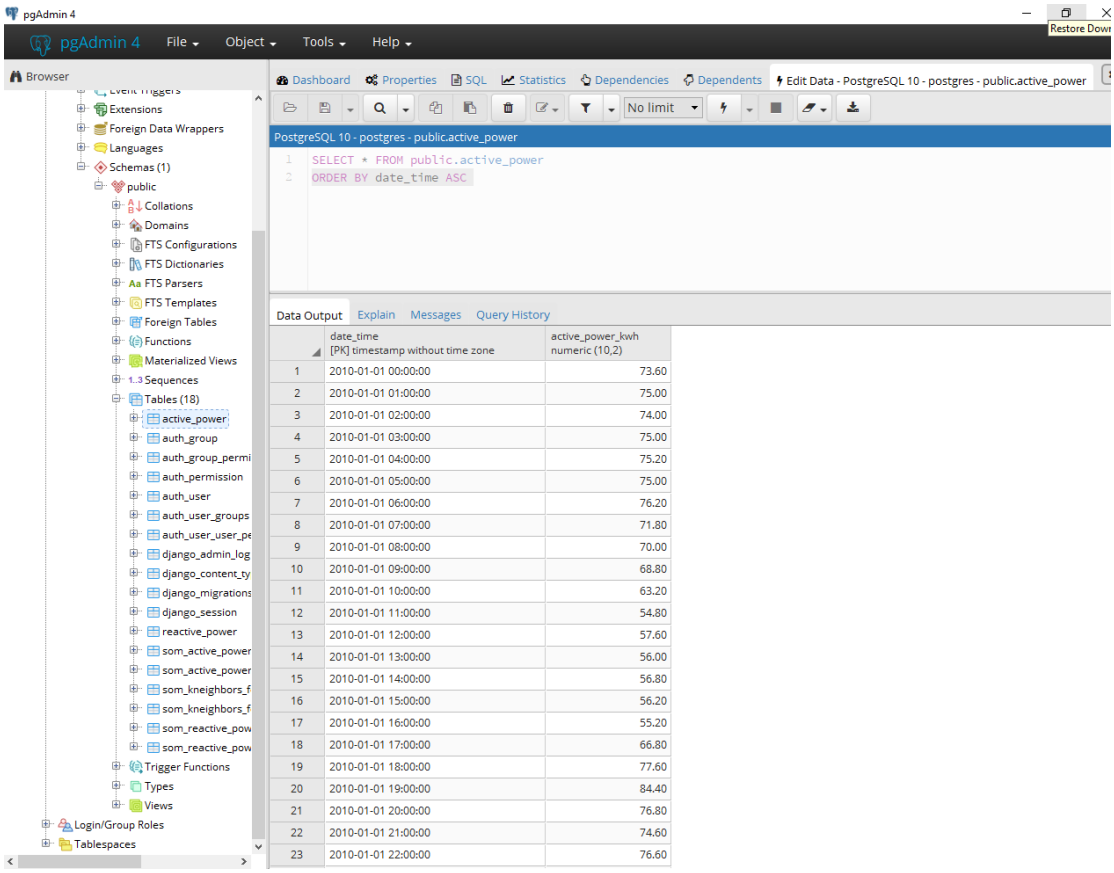
After all the functions and the processes have been successfully completed we are able to enter the database and check that the data required for Divinus to start working are inserted in the corresponding tables. To do that pgadmin package is required to be downloaded. pgAdmin is a free and open source graphical user interface administration tool for PostgreSQL, which is supported on many computer platforms.
Here end the data insertion process. The same process is also followed for the reactive power loads.

3. Clustering Electricity User Profiles Data through Self Organised Map (SOM)

Having all the data ready allow us to move to the next step which is to implement the first of the two algorithms through which the use profiling goal is achieved. The algorithm chosen for this goal as it is already mentioned in Chapter 1 is the Self Organizing Map which is an unsupervised learning algorithm. SOM is a type of Artificial Neural Networks able to convert complex, nonlinear statistical relationships between high-dimensional data items into simple geometric relationships on a low-dimensional display [49].

3.1. Data Pre-Processing

In order to implement the SOM algorithm, data pre-processing is required. The data that are loaded in our system are hourly values which means that they contain a timestamp and the hourly consumption. A depiction of how these data are stored in the database can viewed in Figure 21. This format however is not the desired one because although it can be clustered by SOM the clusters will not make any sense. The data need to be reorganized in a format that will be more logical and the clusters created afterwards could be easily used.



	date_time [PK] timestamp without time zone	active_power_kwh numeric (10,2)
1	2010-01-01 00:00:00	73.60
2	2010-01-01 01:00:00	75.00
3	2010-01-01 02:00:00	74.00
4	2010-01-01 03:00:00	75.00
5	2010-01-01 04:00:00	75.20
6	2010-01-01 05:00:00	75.00
7	2010-01-01 06:00:00	76.20
8	2010-01-01 07:00:00	71.80
9	2010-01-01 08:00:00	70.00
10	2010-01-01 09:00:00	68.80
11	2010-01-01 10:00:00	63.20
12	2010-01-01 11:00:00	54.80
13	2010-01-01 12:00:00	57.60
14	2010-01-01 13:00:00	56.00
15	2010-01-01 14:00:00	56.80
16	2010-01-01 15:00:00	56.20
17	2010-01-01 16:00:00	55.20
18	2010-01-01 17:00:00	66.80
19	2010-01-01 18:00:00	77.60
20	2010-01-01 19:00:00	84.40
21	2010-01-01 20:00:00	76.80
22	2010-01-01 21:00:00	74.60
23	2010-01-01 22:00:00	76.60

Figure 21 - pgAdmin Active Loads View

As a result, the first pre-processing step in SOM's implementation is to reorganize the data in the appropriate format. In order to do that we make use of some real helpful data structures and analysis libraries such as pandas, minisom [49], sklearn preprocessing, and sqlalchemy. The first step as it shown in Table 8 is to retrieve the data from the database with the use of

sqlalchemy. Through the sqlalchemy we are given the ability to choose the data we want and set specific rules. For instance, as it shown in Table 8 we choose to retrieve the active power data where the consumption field is not null. In this way we get all the required data avoiding to have information that are incomplete (e.g. date without consumption). The next thing that should be done as soon as we retrieve the data required is to put them in pandas dataset with the required format. In order to do that we need to create a unique day that will contain 24 empty slots, one for each of the hourly consumptions of that day. As it is shown in Table 8 by running a for loop we are able to insert the hourly consumptions to each of the empty slots.

Table 7 – SOM Data Preprocessing

```
'''
Created on 26 Μαρ 2018

@author: d.mele
'''

import numpy as np
import pandas as pd
from minisom import MiniSom
from sqlalchemy import create_engine
from sklearn.preprocessing import MinMaxScaler
from Database.Truncate import truncate_som_ap, truncate_som_day_ap


def som_active_power_day_clusters():

    # Truncate the data that exist in SOM from previous runs
    truncate_som_ap()
    truncate_som_day_ap()

    # Importing the dataset
    engine                                                                  =
create_engine('postgresql://postgres:123456q!@localhost:5432/postgres')
    dataset = pd.read_sql_query("SELECT date_time, active_power_kwh FROM
active_power WHERE active_power_kwh IS NOT NULL", con=engine)
    #dataset["date_time"] = dataset["date_time"].astype(np.int64)

    print("------------------------------------------------------------------
----------------------")
    print ("Getting Data ready for training and clustering...")

    # We first need to create a dataset that has a unique date and 24 empty
slots for each date in order to enter the consumptions of that date
    date_clusters = pd.DataFrame(columns=['date', 'Hour 0', 'Hour 1', 'Hour
2', 'Hour 3', 'Hour 4', 'Hour 5', 'Hour 6', 'Hour 7',
                                          'Hour 8', 'Hour 9', 'Hour 10',
'Hour 11', 'Hour 12', 'Hour 13', 'Hour 14', 'Hour 15',
                                          'Hour    16','Hour    17','Hour
18','Hour 19','Hour 20','Hour 21','Hour 22','Hour 23'])
    date_clusters['date'] = dataset['date_time'].dt.date.unique()

    data_check = pd.DataFrame(dataset['date_time'].dt.date)
    for i in range(len(date_clusters)):
        get_index    =    data_check.index[data_check['date_time']    ==
date_clusters['date'][i]].tolist()

        for j in get_index:
```

```
            hour = int(dataset['date_time'].loc[j].hour)
            if hour == int(0):
                date_clusters['Hour                0'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(1):
                date_clusters['Hour                1'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(2):
                date_clusters['Hour                2'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(3):
                date_clusters['Hour                3'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(4):
                date_clusters['Hour                4'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(5):
                date_clusters['Hour                5'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(6):
                date_clusters['Hour                6'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(7):
                date_clusters['Hour                7'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(8):
                date_clusters['Hour                8'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(9):
                date_clusters['Hour                9'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(10):
                date_clusters['Hour               10'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(11):
                date_clusters['Hour               11'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(12):
                date_clusters['Hour               12'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(13):
                date_clusters['Hour               13'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(14):
                date_clusters['Hour               14'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(15):
                date_clusters['Hour               15'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(16):
                date_clusters['Hour               16'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(17):
                date_clusters['Hour               17'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(18):
                date_clusters['Hour               18'][i]                =
dataset['active_power_kwh'][j]
```

```
            elif hour == int(19):
                date_clusters['Hour                19'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(20):
                date_clusters['Hour                20'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(21):
                date_clusters['Hour                21'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(22):
                date_clusters['Hour                22'][i]                =
dataset['active_power_kwh'][j]
            elif hour == int(23):
                date_clusters['Hour                23'][i]                =
dataset['active_power_kwh'][j]
```

As soon as all data are reorganized we will be able to see in the console that the pandas dataset is filled with hourly consumptions in the corresponding days. Table 9 shows the format of the pandas dataset after the pre-processing is completed.

Table 8 - Pandas Dataset after the preprocessing is complete. It can now be used to cluster the dates with SOM based on their daily consumptions

	date	Hour 0	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6
0	1262304000000000000	73.6	75	74	75	75.2	75	76.2
1	1262390400000000000	74.8	78.4	76.8	76.6	74.8	76	73.4
2	1262476800000000000	74.4	75.6	74.4	70.8	71.4	70	69.4
3	1262563200000000000	81.2	79.2	78.4	76.2	77.2	75.2	74.6
4	1262649600000000000	84.6	85	84.6	85	85.4	83.2	83.2
5	1262736000000000000	112.6	114.2	107.2	112.8	108.8	108	99.2
6	1262822400000000000	104.6	105.2	99	100.2	95.8	98.2	95
7	1262908800000000000	111	110	108	107.8	106.4	106	104.8
8	1262995200000000000	102.8	103	96.4	100.8	99.6	102.6	98.8
9	1263081600000000000	94.2	95.8	94.8	95.2	95	94.6	93.4
10	1263168000000000000	157.8	160.2	155.6	170.4	154.8	147.4	162.8
11	1263254400000000000	207.6	195	196.4	195	184.2	186.2	193.4
12	1263340800000000000	184.2	180.4	172	189	190.2	178.2	177.8
13	1263427200000000000	192.6	185.8	192.4	180	168	172.6	176.4
14	1263513600000000000	167.2	164.2	168	159	165	169	168.4
15	1263600000000000000	128	122.6	118.8	113.4	115.2	109.2	112.2

```
16    1263686400000000000   111.6  113.6  111.2  113.6  106.8  110.2
107.6
17    1263772800000000000     152  146.8    157  145.4    136    145
156.2
18    1263859200000000000   209.4  203.4  194.8  196.2    197    197
194.8
19    1263945600000000000     195  194.8  197.8  200.8  191.2  211.8
212
20    1264032000000000000   211.4  214.6    219  222.8  200.4  205.4
206.8
21    1264118400000000000   174.8  170.6    167  160.2    173  172.4
168.6
22    1264204800000000000   147.4  143.2  129.4  125.2  121.6  120.4
117.8
23    1264291200000000000   143.4  139.8  143.4    136  132.4    133
128.6
24    1264377600000000000   174.6  172.4  185.6  174.4  179.8  187.4
187
25    1264464000000000000   231.2    230  222.4  205.6  225.2  221.4
219.4
26    1264550400000000000   222.6  205.2  198.8  205.4    208  203.6
201
27    1264636800000000000   195.2  186.8  187.4  184.4  182.4  195.6
184.4
28    1264723200000000000   162.8  160.2  157.2  157.2  170.2  171.2
155.8
29    1264809600000000000   138.2    143  140.2  132.6  128.2  126.6
123.6
...                   ...     ...    ...    ...    ...    ...    ...
...
2951  1517270400000000000   120.4  117.2  117.2  113.4  109.4  103.2
101.4
2952  1517356800000000000   135.6  146.2  134.4    133  125.4    137
133.2
2953  1517443200000000000  132.23  122.8  120.2  117.6  116.8  124.8
119.8
2954  1517529600000000000   112.6  109.6  110.2    106  107.2  112.8
115.2
2955  1517616000000000000    95.6   93.4   88.6   87.4   88.2   87.2
86
2956  1517702400000000000    90.2   77.4   80.2   82.2   80.4   78.2
76.2
2957  1517788800000000000   121.2  122.6    111  103.8   96.8  112.2
107.2
2958  1517875200000000000   120.6  118.6  114.4  112.2  112.8    125
119
2959  1517961600000000000   120.8  119.6  101.8  103.4    102    120
120.2
2960  1518048000000000000      99   99.8   98.4   91.6   90.4  104.8
112.6
2961  1518134400000000000    86.6   94.2   95.2   96.8     98  104.6
110.2
2962  1518220800000000000    89.6   89.2   84.4   79.8   80.8   78.4
78
2963  1518307200000000000    85.6   88.8   86.4   80.6   80.6     77
81
2964  1518393600000000000    88.6     91   89.4     92   88.8  103.8
100.4
```

```
2965  1518480000000000000   112.8  116.2  116.6  113.8    120    125
129.6
2966  1518566400000000000   104.6  101.8  109.2  108.8  103.2  107.8
115.2
2967  1518652800000000000   114.4  110.4  115.8  110.4    112    117
123.2
2968  1518739200000000000    97.4  100.2   98.4   88.6   85.4   97.6
105
2969  1518825600000000000    89.4   85.2   87.4   81.2   77.8   76.8
75
2970  1518912000000000000      93   91.6   92.2   88.2     90     92
88
2971  1518998400000000000    88.2   83.4     88   83.6   75.6   78.4
76.8
2972  1519084800000000000   101.2   95.6  102.4   96.4   92.6  105.6
112.4
2973  1519171200000000000   111.4  107.4    108    101   99.8  103.8
113
2974  1519257600000000000   114.8  111.8    111  105.2   99.6   98.8
105
2975  1519344000000000000     114  106.6  102.6   95.8     86   89.6
94.6
2976  1519430400000000000    82.6   78.4     78   78.6   77.6   75.4
73.4
2977  1519516800000000000    78.2   78.2   82.6   78.2   76.8   76.6
76.6
2978  1519603200000000000    93.8   88.8   88.4   82.2   82.2   95.4
100.4
2979  1519689600000000000   104.2  107.4  103.2   99.4     99    108
103
2980  1519776000000000000     107    104   97.4   98.6   90.8  103.4
111.8

     Hour 7 Hour 8   ...   Hour 14 Hour 15 Hour 16 Hour 17 Hour 18 Hour
19  \
0      71.8     70   ...      56.8    56.2    55.2    66.8    77.6
84.4
1      70.4   66.4   ...      59.2    59.8    59.8    66.2    76.8
82.8
2      66.4   63.8   ...      59.4    58.6    59.8      72    85.4
91.2
3      74.4   80.8   ...     112.4    83.4      88    92.2   100.8
105.8
4      83.8   92.6   ...       114   100.8   104.8   108.4   120.6
123.8
5      96.4     90   ...      80.4    79.4      81    96.6   108.6
113
6      97.8  111.2   ...       109   107.4   104.4   113.4   128.2
126.4
7     108.6    121   ...     175.2   157.4   147.2   142.2   143.8
134.6
8      97.6   92.2   ...      87.4    90.4    89.8   101.4   111.2
113.6
9      92.2   87.4   ...      88.6    88.6    91.2   102.6   115.4
123.8
10    168.8  211.6   ...       399   396.8     399   399.6     344
297
11      200    249   ...     450.6   425.2   409.8   366.4   366.4
362
```

```
12    185.2  223.4   ...     421.2     407   366.2   384.4     382
362.8
13      183  232.2   ...     415.2   381.8   350.6   338.2   324.8
296.6
14      185  226.2   ...       345     303     273   279.8   290.2
263.2
15    106.8    108   ...     121.8   100.4   112.6     129   154.8
140.8
16    107.4  105.2   ...     115.2    98.6    98.6   120.2   136.6
154
17    151.8  221.4   ...       456     427     423   399.8   392.6
361
18      201  258.4   ...     452.8   421.2   398.8   411.6     394
361.4
19    224.6  256.4   ...     449.8   416.4   405.6   397.2   396.4
405.6
20    219.4  258.6   ...     371.6     320   300.2   338.6   331.8
304.6
21    181.2  216.6   ...     402.6     356     347   314.2     323
303.2
22      109  115.4   ...     148.2   124.8   127.6   148.4   172.2
170.6
23      124  121.8   ...     153.4     149   148.2   163.8   193.6
185
24    194.4  267.2   ...       495   434.2   413.6   398.2   392.4
363.8
25    225.8  276.8   ...     496.6   464.8   440.8     426   425.6
386.4
26      207  256.4   ...     453.8   423.8     407   382.6   357.4
343.8
27    199.6    248   ...     392.2   317.4     289   287.8   319.2
309.8
28    169.6  201.4   ...     236.8   229.2   236.4   230.4   256.4
239.6
29      119  119.2   ...     132.8   113.2   107.6   134.4   160.4
168.2
...     ...    ...   ...       ...     ...     ...     ...     ...
...
2951    115  130.8   ...       151   120.2   119.4   123.4   150.8
168.4
2952    162  224.6   ...     319.2   261.8   242.2   229.4   225.6
230.8
2953  148.4  179.4   ...     306.8     245   206.4   187.6   194.4
197.6
2954  135.2    179   ...     343.8   282.4   225.8     170   182.6
181.4
2955   83.8   79.8   ...     139.6   105.4    88.2    91.4   109.4
116
2956   75.8   73.6   ...     112.8    92.6    94.8   102.4   128.8
127.8
2957    140  183.2   ...     269.6   249.6   209.8     184   199.2
195.2
2958  152.2  193.6   ...     240.2   213.6     187   167.4   191.8
196.8
2959  165.4  192.6   ...     251.8     222   183.4   142.2     141
161.4
2960  143.4  179.4   ...       240   200.8   186.2   163.2     187
201.2
```

```
2961  130.4    170   ...     234.6     162   148.4   132.8   120.6
129.2
2962   78.4   76.4   ...     141.6     115   105.2   114.2   120.4
129.2
2963   81.8   80.2   ...     122.4    96.6      95    91.4   112.2
115.2
2964    136  169.6   ...     241.2   185.2   170.6     155   165.6
168.4
2965    165  194.6   ...     306.2   226.8   193.2     164   166.8
181.4
2966  158.8  177.2   ...       286   224.4     196     164   181.8
198
2967  145.2    195   ...     310.6   226.6     187   169.8   165.6
155
2968  140.8  162.4   ...     271.6   222.8   181.8   159.6   148.8
137.8
2969   73.2   77.4   ...     123.4    93.6    89.2    90.4   107.4
112.8
2970   84.6   86.4   ...     101.6    76.2    71.6    85.4    93.6
92
2971   91.6  105.2   ...       124    96.2   102.2   100.6   103.2
115.4
2972  142.4  179.2   ...     292.8   232.8   193.2   165.2   151.8
166.8
2973  148.8  175.2   ...     272.4   201.6   182.6   159.4   159.8
174.2
2974  137.6  161.6   ...     216.2     184   170.4   138.8   143.4
167.4
2975  124.4  156.8   ...     237.2   184.4   149.8   131.2   133.6
143
2976   75.4   76.8   ...     130.2      94    90.4    93.6    96.6
103.4
2977   71.8   73.8   ...     115.4    92.8    89.6   108.2     116
117.8
2978    138  165.8   ...       256   183.8   177.8     181     168
191.4
2979  147.2    189   ...       299   245.2   218.2     192     171
184.6
2980  143.8  173.6   ...       260     212   177.6   153.6   140.4
159.2

     Hour 20 Hour 21 Hour 22 Hour 23
0       76.8    74.6    76.6      73
1       74.8    75.2    75.6    78.4
2       80.2      79    79.6    78.2
3       90.2    88.2    83.8    84.8
4      117.2   114.6   116.8   114.2
5      104.2   103.8   101.8   103.8
6      112.4     110   114.4   111.2
7        106    99.4      97    98.8
8       98.4   100.8    93.6    93.8
9      121.4   124.2   133.2   145.4
10     222.6   209.2     205     209
11     268.8   204.6   188.6   191.6
12     276.4   223.6   202.2   199.4
13     211.4   185.8   178.8   176.8
14     174.2   136.2   135.8   130.8
15     124.2   123.6   121.2     119
16     129.2     137   137.2   147.2
```

```
17     270.8   236.4   235.4   225.4
18     278.4   239.8   220.6   205.6
19     278.6   233.6   221.4     214
20       232     201     178   165.8
21     208.2   164.2   152.8   147.8
22     157.2   157.4   153.2     151
23       172   177.2   173.8   178.8
24     282.4   250.8     238   229.6
25     285.4   231.6   227.8   230.4
26     257.4   221.8   213.4   198.4
27     257.2     200     189   181.6
28     184.4   148.2   144.2   142.4
29     152.4     143   146.8   135.6
...      ...     ...     ...     ...
2951   146.4   140.2     139   131.6
2952   200.4   148.2     139   145.4
2953   184.2   137.4   118.2   113.6
2954   154.6      97     106    97.4
2955    89.2    94.6      93      92
2956     105   112.2     115   117.2
2957   154.4   120.4     125   122.8
2958   161.6   121.2   132.8   125.8
2959     138   103.6    96.2     100
2960   168.4     118    98.2    89.6
2961     117    96.4    94.8    95.2
2962    92.8    92.2      89    89.2
2963    91.2    89.4      93    88.2
2964   158.6     120     121   114.4
2965     167     125     123     121
2966   172.8     143   121.4   121.2
2967   131.2   110.8      96    99.4
2968   129.8    88.4      91    92.2
2969    95.4    93.4      91    92.6
2970    80.2    75.6    82.6      85
2971    98.6    93.8    95.4      95
2972   154.6   128.4   124.8   119.2
2973   147.2   118.2   114.2   118.8
2974   158.4   126.2   118.2     117
2975     132    88.6    80.8    89.2
2976    85.2    84.6      82    75.8
2977    97.4    99.8    98.2     102
2978   164.8   119.6     117   109.4
2979     163   123.4     118   114.8
2980   158.2   120.6     112   108.8
```

Furthermore, besides the data reshape, in Table 9 we are able to see that the date column does not show dates but epochs. This happens not because of a mistake but rather of the need to convert dates into integers so that they can be read by the SOM algorithm. This change is accomplished with the use of the code in Table 10:

Table 9 - Code that converts date to epoch in order to be used in SOM algorithm

```
date_clusters['date'] = pd.to_datetime(date_clusters['date'])
date_clusters['date'] = date_clusters['date'].astype(np.int64)
print(date_clusters)
```

3.2. Implementation of Self Organizing Map

Having successfully reshaped that data in a logical format we are able to progress to the SOM implementation. The SOM implementation is based on the minisom python library which is a minimalistic and Numpy based implementation of the Self Organizing Maps [49]. The first thing that needs to be done is to split the data into the data that will be clustered and the data on which the clusters are to be created.

Table 10 - Creation of two numpy data arrays X and y. The X numpy array holds the data that will be clustered while the y numpy array holds the data based on which the clustering of X will take place

```
    X = date_clusters.iloc[:, 0:25].values

    #Feature Scaling
    sc = MinMaxScaler(feature_range= (0,1))
    X = sc.fit_transform(X)
    print(X)
```

As soon as this task is complete we have to specify the size of the SOM which in turn determines the accuracy of the clusters. By reducing or increasing the size we can define a larger or a smaller number of clusters. The actual question is not how many clusters can the SOM algorithm produce but rather how many clusters are really needed. The answer to this questions can be given relatively easy if we know the functionality of the building. For instance, the use profile of a home is quite different from the use profile of a factory of a university. Therefore, crucial information such as the aforementioned should be taken under consideration. In our case, having data from several past years, we are able to observe repeated patterns of consumption and therefore comprehend the functionality of the building we want to create its use profile. After carrying out a small research based on past loads and having tested a variety of clusters we came to the conclusion that for the Technological Education Institute facilities in Psachna Euboeas the required number of clusters sums up to 4.
Having found the required number of clusters, we are able to start the implementation of the SOM algorithm.

Table 11 – Initialization of the SOM algorithm with a 2x2 matrix

```
    x = 2
    y = 2
    # Training the SOM
    som = MiniSom(x = x, y = y, input_len = 25, sigma=1.0,
learning_rate=0.5) # initialization of a 2x2 SOM
    som.random_weights_init(X)
    print("------------------------------------------------------------------
----------------------")
    print ("Training active power loads...")
    som.train_random(data = X, num_iteration = 100) # trains the SOM with
100 iterations
    print ("...ready!")
    #Showing the Clusters
    clusters = som.win_map(X)

    xx = x + 1
    yy = y + 1
    for z in range(xx):
        for w in range(yy):
            cluster = clusters[(z,w)]
            if cluster != []:
                cluster_norm = sc.inverse_transform(cluster)
```

```
                som_predataset = pd.DataFrame(cluster_norm)
                som_predataset['som_column'] = w
                cols = som_predataset.columns.tolist()
                som_predataset = som_predataset[[cols[-1]] + cols[:-1]]
                som_predataset['som_row'] = z
                cols = som_predataset.columns.tolist()
                som_predataset = som_predataset[[cols[-1]] + cols[:-1]]

                som_predataset.rename(columns={0:'date_active_power',
1:'hour_0', 2:'hour_1', 3:'hour_2',
                                               4:'hour_3', 5:'hour_4',
6:'hour_5', 7:'hour_6',
                                               8:'hour_7', 9:'hour_8',
10:'hour_9', 11:'hour_10',
                                               12:'hour_11', 13:'hour_12',
14:'hour_13', 15:'hour_14',
                                               16:'hour_15', 17:'hour_16',
18:'hour_17', 19:'hour_18',
                                               20:'hour_19', 21:'hour_20',
22:'hour_21', 23:'hour_22',
                                               24:'hour_23'}, inplace=True)

                som_predataset['date_active_power'] =
pd.to_datetime(som_predataset['date_active_power'])
                som_predataset['date_active_power'] =
som_predataset['date_active_power'].dt.round('1s')
                som_predataset['date_active_power'] =
pd.DataFrame(som_predataset['date_active_power'].dt.date)
                print(som_predataset)
```

SOM training starts as soon as we have specified the clusters and have run the SOM algorithm. A few seconds later we are given print outs of pandas datasets as it shown in Table 13 in which the reshaped data contain two more columns which are the som_row and the som_column that act as the identifies of the cluster in which the dataset belongs.

Table 12 – Pandas Dataset that belongs to the cluster [0,0]

```
--------------------------------------------------------------------------
-----------
Training active power loads...
...ready!
     som_row  som_column date_active_power  hour_0  hour_1  hour_2
hour_3  \
0          0           0        2010-01-11   157.8   160.2   155.6
170.4
1          0           0        2010-01-12   207.6   195.0   196.4
195.0
2          0           0        2010-01-13   184.2   180.4   172.0
189.0
3          0           0        2010-01-14   192.6   185.8   192.4
180.0
4          0           0        2010-01-15   167.2   164.2   168.0
159.0
5          0           0        2010-01-18   152.0   146.8   157.0
145.4
6          0           0        2010-01-19   209.4   203.4   194.8
196.2
```

```
7           0           0        2010-01-20   195.0   194.8   197.8
200.8
8           0           0        2010-01-21   211.4   214.6   219.0
222.8
9           0           0        2010-01-22   174.8   170.6   167.0
160.2
10          0           0        2010-01-25   174.6   172.4   185.6
174.4
11          0           0        2010-01-26   231.2   230.0   222.4
205.6
12          0           0        2010-01-27   222.6   205.2   198.8
205.4
13          0           0        2010-01-28   195.2   186.8   187.4
184.4
14          0           0        2010-02-01   155.4   162.4   154.0
158.6
15          0           0        2010-02-02   176.0   170.2   169.0
164.0
16          0           0        2010-02-03   213.8   210.6   217.6
199.4
17          0           0        2010-02-05   164.4   164.0   167.6
165.6
18          0           0        2010-02-08   166.4   163.2   166.4
165.4
19          0           0        2010-02-09   195.0   187.0   183.4
175.6
20          0           0        2010-02-11   154.0   147.4   146.4
145.8
21          0           0        2010-02-12   160.6   154.8   150.6
154.8
22          0           0        2010-02-17   142.4   140.6   143.2
135.2
23          0           0        2010-02-23   150.2   145.0   140.4
144.0
24          0           0        2010-03-01   138.6   136.6   136.4
142.2
25          0           0        2010-03-08   157.2   153.2   160.0
152.0
26          0           0        2010-03-09   179.6   175.0   179.0
174.0
27          0           0        2010-03-10   173.0   163.6   153.6
149.2
28          0           0        2010-03-15   161.8   158.2   150.2
162.6
29          0           0        2010-03-16   200.0   196.2   185.6
192.0
..        ...         ...               ...     ...     ...     ...
...
570         0           0        2014-03-21   110.4   107.4   108.2
100.4
571         0           0        2014-03-26   106.6   111.4   106.0
100.8
572         0           0        2014-03-31   111.6   115.6   116.4
124.2
573         0           0        2014-04-08    98.8    95.8    98.4
100.6
574         0           0        2014-12-16   120.2   119.6   112.4
127.0
```

```
575        0           0        2014-12-17   124.4   126.8   116.2
108.8
576        0           0        2014-12-18   117.0   106.8   112.4
116.0
577        0           0        2015-01-07   138.0   136.6   138.4
138.8
578        0           0        2015-01-08   158.0   162.6   150.0
140.0
579        0           0        2015-01-13   152.0   151.6   151.2
162.0
580        0           0        2015-01-14   134.8   129.0   128.8
138.6
581        0           0        2015-01-15   134.8   132.8   125.6
143.2
582        0           0        2015-01-16   124.2   124.6   119.0
123.0
583        0           0        2015-01-19   119.0   112.4   105.8
120.8
584        0           0        2015-01-21   129.6   116.8   118.6
122.6
585        0           0        2015-01-27   132.4   132.4   129.0
134.0
586        0           0        2015-01-28   136.8   132.0   132.0
136.6
587        0           0        2015-01-29   134.2   126.0   127.4
130.2
588        0           0        2015-02-03   118.2   108.4   113.8
122.2
589        0           0        2015-02-04   144.8   127.4   119.0
117.2
590        0           0        2015-02-18   157.0   142.4   131.4
124.2
591        0           0        2015-02-19   125.0   126.2   125.4
123.6
592        0           0        2015-02-25   115.0   115.2   110.4
130.0
593        0           0        2015-12-15   156.0   152.8   143.8
135.2
594        0           0        2015-12-16   131.8   134.6   124.6
120.2
595        0           0        2016-01-19   126.6   129.8   126.4
128.2
596        0           0        2016-01-22   139.6   140.2   139.2
132.8
597        0           0        2016-01-25   135.2   137.8   137.0
132.8
598        0           0        2016-01-26   145.6   149.6   139.2
130.8
599        0           0        2016-01-27   150.2   133.0   129.8
133.0

     hour_4  hour_5  hour_6   ...     hour_14  hour_15  hour_16  hour_17
\
0     154.8   147.4   162.8   ...       399.0    396.8    399.0    399.6
1     184.2   186.2   193.4   ...       450.6    425.2    409.8    366.4
2     190.2   178.2   177.8   ...       421.2    407.0    366.2    384.4
3     168.0   172.6   176.4   ...       415.2    381.8    350.6    338.2
4     165.0   169.0   168.4   ...       345.0    303.0    273.0    279.8
5     136.0   145.0   156.2   ...       456.0    427.0    423.0    399.8
```

```
6     197.0   197.0   194.8   ...       452.8    421.2    398.8    411.6
7     191.2   211.8   212.0   ...       449.8    416.4    405.6    397.2
8     200.4   205.4   206.8   ...       371.6    320.0    300.2    338.6
9     173.0   172.4   168.6   ...       402.6    356.0    347.0    314.2
10    179.8   187.4   187.0   ...       495.0    434.2    413.6    398.2
11    225.2   221.4   219.4   ...       496.6    464.8    440.8    426.0
12    208.0   203.6   201.0   ...       453.8    423.8    407.0    382.6
13    182.4   195.6   184.4   ...       392.2    317.4    289.0    287.8
14    145.6   158.0   160.4   ...       318.6    289.8    297.2    299.6
15    151.2   162.8   160.8   ...       429.8    370.2    334.2    318.2
16    184.2   192.6   186.8   ...       367.6    317.8    313.0    309.6
17    173.4   183.4   174.4   ...       294.2    259.8    251.8    253.8
18    153.4   163.8   165.0   ...       364.4    293.6    284.4    292.4
19    173.0   187.6   175.6   ...       322.4    277.8    278.0    283.6
20    136.2   144.0   150.0   ...       291.0    238.4    233.0    242.2
21    142.8   148.4   147.4   ...       279.2    243.8    214.8    210.6
22    123.6   123.0   126.2   ...       238.6    233.4    193.0    204.2
23    143.6   157.4   155.4   ...       283.8    244.0    242.2    242.4
24    135.6   139.2   147.0   ...       260.2    228.6    197.4    197.0
25    146.4   141.2   134.0   ...       335.6    309.4    296.4    281.2
26    191.0   176.2   167.8   ...       310.6    263.6    236.6    225.6
27    168.2   175.2   170.4   ...       351.4    309.6    288.4    276.2
28    154.2   159.2   155.8   ...       382.2    337.8    313.8    295.0
29    180.2   184.6   179.8   ...       436.2    354.6    335.8    309.8
..      ...     ...     ...   ...         ...      ...      ...      ...
570   105.2   119.6   113.2   ...       251.0    197.0    177.4    142.4
571    99.2   103.0    99.0   ...       267.0    229.2    212.0    170.4
572   109.8   119.0   134.2   ...       315.8    280.8    247.4    211.6
573    99.4   102.0   106.4   ...       247.6    212.8    185.8    137.0
574   125.4   121.4   109.8   ...       330.8    304.6    287.0    249.0
575   130.8   130.0   112.4   ...       272.8    252.6    241.2    215.2
576   123.4   120.6   109.2   ...       256.8    231.4    204.0    174.4
577   138.6   142.0   152.4   ...       330.4    301.0    284.2    259.0
578   133.2   159.6   166.6   ...       426.2    357.4    324.4    300.0
579   157.0   158.8   167.2   ...       495.4    457.2    398.2    332.6
580   137.6   146.2   137.4   ...       404.4    329.2    283.0    243.4
581   143.4   152.2   138.8   ...       341.4    285.2    240.4    213.8
582   136.4   132.0   118.8   ...       317.4    255.0    201.4    191.2
583   117.4   123.4   111.2   ...       342.6    298.8    275.0    276.6
584   109.6   115.2   107.4   ...       298.0    246.6    215.8    187.8
585   126.8   129.8   130.4   ...       369.2    324.6    276.2    259.8
586   131.6   134.6   123.8   ...       341.6    292.0    272.2    228.0
587   128.0   130.6   119.8   ...       296.0    231.0    207.6    190.6
588   115.0   118.8   119.4   ...       283.0    236.0    211.6    197.8
589   116.4   116.4   115.4   ...       321.6    271.8    225.6    201.0
590   122.4   137.6   149.6   ...       364.2    288.4    241.4    217.6
591   129.6   159.0   150.6   ...       330.8    264.4    219.4    183.0
592   118.2   135.6   123.2   ...       305.2    255.0    204.8    187.2
593   126.4   151.0   142.4   ...       293.6    275.4    259.2    236.4
594   122.0   156.4   152.8   ...       323.0    296.2    285.2    251.8
595   132.2   150.4   151.0   ...       328.0    281.4    258.2    226.0
596   130.6   153.2   146.2   ...       321.0    277.4    253.4    245.2
597   129.2   159.8   150.4   ...       403.4    349.2    308.8    268.2
598   131.6   171.4   159.0   ...       392.8    336.0    289.0    261.4
599   140.6   163.8   151.2   ...       343.2    300.0    248.2    215.4

     hour_18  hour_19  hour_20  hour_21  hour_22  hour_23
0      344.0    297.0    222.6    209.2    205.0    209.0
1      366.4    362.0    268.8    204.6    188.6    191.6
```

2	382.0	362.8	276.4	223.6	202.2	199.4
3	324.8	296.6	211.4	185.8	178.8	176.8
4	290.2	263.2	174.2	136.2	135.8	130.8
5	392.6	361.0	270.8	236.4	235.4	225.4
6	394.0	361.4	278.4	239.8	220.6	205.6
7	396.4	405.6	278.6	233.6	221.4	214.0
8	331.8	304.6	232.0	201.0	178.0	165.8
9	323.0	303.2	208.2	164.2	152.8	147.8
10	392.4	363.8	282.4	250.8	238.0	229.6
11	425.6	386.4	285.4	231.6	227.8	230.4
12	357.4	343.8	257.4	221.8	213.4	198.4
13	319.2	309.8	257.2	200.0	189.0	181.6
14	321.4	322.2	270.8	190.4	178.4	179.4
15	353.0	363.8	313.8	225.8	215.2	207.6
16	346.2	362.8	289.0	228.0	216.0	226.0
17	275.6	281.8	222.8	176.8	147.2	140.8
18	321.2	325.2	263.2	200.2	204.8	205.8
19	310.6	297.8	258.4	193.4	174.0	174.2
20	272.0	277.8	246.0	180.6	157.4	155.4
21	238.0	222.2	185.6	138.0	126.8	114.6
22	207.8	219.2	179.8	144.6	127.2	123.2
23	257.0	262.6	189.8	152.6	142.6	148.0
24	210.8	217.2	183.6	151.2	149.0	152.4
25	292.4	295.8	226.8	180.4	187.2	188.0
26	244.8	260.2	193.8	175.0	181.4	181.4
27	264.8	261.4	209.6	181.0	173.0	162.4
28	300.0	301.6	237.4	201.6	200.2	204.2
29	322.4	329.6	245.4	194.8	187.0	188.4
..	...	...	...	...	...	...
570	146.6	159.4	116.2	82.2	86.2	81.4
571	173.0	195.0	162.0	136.4	128.0	124.8
572	203.0	192.6	158.0	141.2	139.8	147.0
573	135.4	149.0	118.2	99.6	101.2	104.0
574	233.6	221.6	179.0	147.4	141.0	134.2
575	212.4	210.6	167.4	156.6	146.4	136.0
576	188.8	197.4	162.0	145.0	125.2	112.6
577	236.0	239.8	196.2	188.0	186.2	166.6
578	297.2	298.0	237.4	203.8	193.4	175.4
579	309.0	277.6	205.0	168.4	160.2	151.6
580	248.2	261.2	215.0	178.2	153.0	145.2
581	233.6	245.8	206.2	181.4	155.0	142.2
582	197.4	208.4	164.2	138.2	132.0	122.6
583	273.6	283.6	226.2	187.2	173.8	168.2
584	224.0	262.8	209.0	162.6	153.6	138.4
585	271.8	282.2	228.2	183.8	168.8	155.0
586	239.0	250.4	199.4	177.4	161.2	150.4
587	190.0	200.2	161.4	137.6	131.0	125.4
588	201.6	196.2	163.4	144.8	147.4	149.8
589	209.0	243.4	200.8	148.4	139.4	136.0
590	209.0	218.8	165.4	132.2	119.6	116.8
591	193.4	215.2	153.6	135.0	125.0	112.0
592	183.6	197.6	140.6	112.0	109.6	95.8
593	238.0	218.2	176.0	153.2	135.0	133.2
594	244.8	233.2	182.2	159.6	142.2	133.6
595	233.0	237.2	198.8	169.2	168.8	149.8
596	209.8	218.2	173.8	145.2	121.0	115.4
597	254.8	265.2	204.8	190.2	162.4	159.2
598	235.4	256.8	206.0	168.6	159.6	158.4
599	216.4	228.2	183.2	148.0	132.6	120.2

```
[600 rows x 27 columns]
------------------------------------------------------------------------
-----------
```

As soon as a pandas dataset is displayed in the console it is also automatically inserted in the database. The insertion is completed with the use of the sqlalchemy library through which the database insertion is performed instantly. The command with which this action is performed is displayed in Table 14.

Table 13 - SQLAlchemy Insertion Command for reshaped data

```
                som_predataset.to_sql(name =
'som_active_power_day_clusters', con = engine, if_exists = 'append',
index =False)
```

3.3. Recreation of the Initial data to the created clusters

After the insertion of the modified data in the database we should also import the data in their initial state in order to be ready to be used by the next algorithm which predicts future loads. To make that possible we created a for loop which based on the previous matrix where we have all the reshaped data it performs an index search to both the initial values and the reshaped one and starts to create a new pandas dataset where it will contain all the initial values with two additional columns. The additional columns are again the som_row and the som_column that act as the clusters identifies.

Table 14 - Process performed in order to link the intial data with the created clusters

```
                # Inserting SOM Clusters to database
                print("-----------------------------------------------------
-----------------------------------")
                print ("Inserting SOM Clusters to database...")
                print(" ")
                som_dataset = pd.DataFrame(columns=['som_row',
'som_column', 'date_time', 'som_active_power_kwh'])

                for n in range(len(som_predataset)):
                    get_index = data_check.index[data_check['date_time']
== som_predataset['date_active_power'][n]].tolist()
                    k = 0
                    for m in get_index:
                        som_dataset.at[k, 'som_row'] =
int(som_predataset.at[n, 'som_row'])
                        som_dataset.at[k, 'som_column'] =
int(som_predataset.at[n, 'som_column'])
                        som_dataset.at[k, 'date_time'] = dataset.at[m,
'date_time']
                        som_dataset.at[k, 'som_active_power_kwh'] =
int(dataset.at[m, 'active_power_kwh'])
                        k= k + 1
                    print(som_dataset)
```

For each loop that will be completed a pandas dataset will be printed and will be automatically inserted into the database again using the sqlalchemy library for instant import. The command used to insert the new pandas set into the database is depicted in Table

Table 15 - SQLAlchemy Insertion Command for intial data and the clusters that they now belong

```
                    som_dataset.to_sql(name = 'som_active_power', con =
engine, if_exists = 'append', index =False)

    print("")
    print("All active loads data were successfully inserted into the
database")
    print("-------------------------------------------------------------
----------------------")
    print("")
```

This is where the SOM execution ends. The time required for the SOM implementation to reshape approximately 71544 data, cluster them and then insert into the created clusters the initial data is approximately 4 minutes. The data of the SOM clusters are presented in Divinus website which will be shown in Chapter 5. The same process is followed to cluster the reactive power data.

4. Forecasting Future Electricity User Profiles

Short Term Load Forecasting (STLF) is a very important aspect in the formulation of economic, reliable, and secure operating strategies for the power system. To perform STLF it is usually required to have a lot of past data based on which our forecast mechanism will be trained and tested. In the forecast that is performed by Divinus program we used a different approach. We are not interested in using as much past data as possible rather than using past data that are qualitatively close to each other in the sense that they present relative common consumption. We achieve to retrieve the required qualitatively closeness by retrieving the clusters that were implemented with the use of SOM algorithm.
Furthermore, Divinus forecasting is a three step process. The first stage of this process is to retrieve all the required past data needed. The second stage is to use these data for training and testing of the prediction algorithm and the final stage is to perform the forecasting of the days that we want to predict.

4.1. First Stage of the forecasting process

The first thing that is set to run in the prediction code is the SOM algorithm. We have set the SOM algorithm to always be triggered by the forecast prediction as is it depicted in Table 16. This happens because the implementation of the forecasting process heavily relies on SOM clusters and cannot happen without them.

Table 16 - SOM algorithm trigged to run by the Forecasting Code

```
'''
Created on 29 Μαρ 2018

@author: d.mele
'''

# Importing the libraries
import calendar
import threading
import pandas as pd
from sqlalchemy import create_engine
from SOM.SOM_Active_Load import som_active_power_day_clusters
from Database.Update import update_som_KNeighbors_forecasted_ap

def SOM_KNeighbors_forecasting_ap():

    som_active_power_day_clusters() #We trigger the function that contains
the SOM implementation
```

As soon as all the clusters have been created and the data are inserted to all or some of the clusters we retrieve the data located in Divinus database with the use of the sqlachemy library. When all the data are retrieve we perform two checks to make sure that both data for forecast and data on which the forecast will rely on exist. If one of the required datasets does not exist than the forecasting process will be finalized here.

Table 17 - Checking the required dataset to make sure that all the required data exist

```
    # Importing the Dataset
    engine =
create_engine('postgresql://postgres:123456q!@localhost:5432/postgres')
```

```
    dataset = pd.read_sql_query("SELECT som_active_power_kwh, som_row,
som_column, date_time FROM som_active_power WHERE som_active_power_kwh IS
NOT NULL", con=engine)

    som_KNeighbors_ap_dataset = pd.read_sql_query("SELECT
som_KNeighbors_forecasted_active_power_kwh, date_time FROM
som_KNeighbors_forecasted_active_power WHERE
som_KNeighbors_forecasted_active_power_kwh IS NULL", con=engine)
    som_KNeighbors_ap_dataset =
som_KNeighbors_ap_dataset.sort_values(['date_time']).reset_index(drop=Tru
e)

    # Fixing the date to be used for the update command
    date_time =
pd.to_datetime(som_KNeighbors_ap_dataset['date_time']).sort_values().rese
t_index(drop=True)

    if som_KNeighbors_ap_dataset.empty:
        print("-------------------------------------------------------------
--------------------------")
        print ("Training active power loads with KNeighbors
Algorithm...")
        print("There are no new data to perform forecasting")
        print("-------------------------------------------------------------
--------------------------")

    elif dataset.empty:

        print("-------------------------------------------------------------
--------------------------")
        print ("Training active power loads with KNeighbors
Algorithm...")
        print("There are no no data to forecast")
        print("-------------------------------------------------------------
--------------------------")
```

Having checked that all the data exist, Divinus enters the first stage of the forecasting process. In this stage the action that Divinus is required to do is to find out whether the day that it is going to predict exists in past data. To put it in simple terms, Divinus tries to match this date of the year 2018 with the same day if it exists of the past years. This action is performed for a very specific reason. In case this day is not a fixed holiday or a movable holiday we are going to use the same past days including all the other days that are stored in the same cluster that they belong. On the other hand if this day is a holiday fixed or not we are going to make use of the same day in the past years without including the other values contained in their cluster.
To make that happen and to retrieve the correct days from the past data we use a python library called calendar. Through this library we are able to retrieve the year, the month, the day and the hour from a timestamp. As a result, having all this valuable data in or hands we are able to compare the data of day we want to forecast with past data and retrieve the corresponding dates and the cluster where they belong as it shown in Table 18.

Table 18 – Checking data such as year, month, day, hour and retrieving past data based on these criteria.

```
    else:
```

```
        dataset_year = dataset['date_time'].dt.year.unique()
        dataset_year_length = len(dataset_year)

        mean_test_error = pd.DataFrame([])
        error_check_dataset = pd.DataFrame([])

        for i in range(len(som_KNeighbors_ap_dataset)):

            som_KNeighbors_ap_dataset['date_time'] =
pd.to_datetime(som_KNeighbors_ap_dataset['date_time'])
            som_KNeighbors_ap_dataset['date_time'] =
som_KNeighbors_ap_dataset['date_time'].dt.round('1s')

            year =
int(som_KNeighbors_ap_dataset['date_time'].loc[i].year)
            month =
int(som_KNeighbors_ap_dataset['date_time'].loc[i].month)
            hour =
int(som_KNeighbors_ap_dataset['date_time'].loc[i].hour)

            print("------------------------------------------------------
------------------------------")
            print("")
            day_of_the_week_number =
(som_KNeighbors_ap_dataset['date_time'].loc[i].weekday())
            print("Day Number of the week: ", day_of_the_week_number)
            day_of_the_week_name =
(calendar.day_name[som_KNeighbors_ap_dataset['date_time'].loc[i].weekday(
)])
            print("Day of the week: ", day_of_the_week_name)
            number_of_the_week =
(som_KNeighbors_ap_dataset['date_time'].loc[i].week)
            print("Week Number: ", number_of_the_week)

            clusters_finder = []
            datasets_to_train = pd.DataFrame()
            print("------------------------------------------------------
------------------------------")

            for z in range(dataset_year_length + 1):
                clusters_finder =
dataset.loc[(dataset['date_time'].dt.year == year-z) &
(dataset['date_time'].dt.month == month) & (dataset['date_time'].dt.week
== number_of_the_week) & (dataset['date_time'].dt.weekday ==
day_of_the_week_number) & (dataset['date_time'].dt.hour == hour)]
                if not clusters_finder.empty :
                    print(clusters_finder['date_time'])
                    previous_year =
int(clusters_finder['date_time'].dt.year)
                    print("Year from which we get the corresponding day:
", previous_year)

                    yearly_day_of_the_week =
pd.to_datetime(clusters_finder['date_time'])
                    yearly_day_number =
int(yearly_day_of_the_week.dt.weekday)
                    print("Number of the Day of the week: ",
yearly_day_number)
```

```
                    yearly_day_name =
yearly_day_of_the_week.dt.weekday_name
                    print("Day of the week: ", yearly_day_name)
                    yearly_week_number =
int(yearly_day_of_the_week.dt.week)
                    print("Week Number: ", yearly_week_number)
                    print("")
                    row = int(clusters_finder['som_row'])
                    column = int(clusters_finder['som_column'])
```

4.2. Second Stage of the forecasting process

Having retrieve the required past dates and the clusters in which they belong we are able to enter the second stage of the prediction process. In this stage based on the cluster that the past data belong we are able to retrieve all the data required in order to start training the algorithm that makes the predictions and then test these prediction on test data.
An easy way to retrieve the clusters is by using a sql in which is specified the row and the column of the clusters we want to retrieve.

```
sql = ("SELECT som_active_power_kwh, som_row, som_column, date_time FROM
som_active_power WHERE som_row = '{0}' AND som_column =
'{1}'".format(row, column))
```

Furthermore, besides the data of the clusters we retrieve we perform a modification on both the cluster data and the data on which we want to perform the forecast. The modification is nothing more than breaking the timestamp and creating separate columns containing the date, month, day and hour time. This modification takes place because the algorithm has a better performance if the data on which the forecasting is based are separately and not all combined in a timestamp

Table 19 - Retrieving the data required for train and test and for the real forecast and performing the modifications required

```
  sql = ("SELECT som_active_power_kwh, som_row, som_column, date_time
FROM som_active_power WHERE som_row = '{0}' AND som_column =
'{1}'".format(row, column))
                    datasets_for_train = pd.read_sql_query(sql,
con=engine)
                    datasets_for_train['year'] =
(datasets_for_train['date_time'].dt.year)
                    datasets_for_train['month'] =
(datasets_for_train['date_time'].dt.month)
                    datasets_for_train['day'] =
(datasets_for_train['date_time'].dt.weekday)
                    datasets_for_train['hour'] =
(datasets_for_train['date_time'].dt.hour)
                    datasets_to_train =
datasets_to_train.append(datasets_for_train)

            # Making the date an integer to be used for the prediction
            som_KNeighbors_ap_dataset['year'] =
(som_KNeighbors_ap_dataset['date_time'].dt.year)
            som_KNeighbors_ap_dataset['month'] =
(som_KNeighbors_ap_dataset['date_time'].dt.month)
            som_KNeighbors_ap_dataset['day'] =
(som_KNeighbors_ap_dataset['date_time'].dt.weekday)
```

```
            som_KNeighbors_ap_dataset['hour'] =
(som_KNeighbors_ap_dataset['date_time'].dt.hour)
            som_KNeighbors_ap_dataset =
som_KNeighbors_ap_dataset.sort_values(['date_time']).reset_index(drop=Tru
e)
```

As soon as this step is complete Divinus splits the data in two groups X and y. X is the group that contains the data required for the prediction while y is the group that contains the values that should be predicted.

Table 20 - Splitting the dataset in X and y

```
# Splitting the variables to the desired columns
            X = datasets_to_train.iloc[:, 4:9].values
            y = datasets_to_train.iloc[:, 0].values

            # Splitting the Dataset into Training and Testing set
            from sklearn.model_selection import train_test_split
            preferable_test_size = (240/len(X))
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = preferable_test_size, random_state = 0)

            # Fitting Forest Algorithm to the training set
            from sklearn.neighbors import KNeighborsRegressor
            regressor = KNeighborsRegressor(algorithm='auto',
leaf_size=30, metric='minkowski',
                                            metric_params=None, n_jobs=1,
n_neighbors=8, p=2, weights='distance')

            print("-------------------------------------------------------
------------------------------")
            print ("Training active power loads with KNeighbors
Algorithm...")
            regressor.fit(X_train, y_train)
            print(regressor)
            print("")
            print("... Data Training Completed with KNeighbors
Algorithm")
```

As soon as the first split is over another one takes place. This one takes the already spitted datasets and splits them even more. Now we have four datasets which are the a) X_train, b) y_train, c) X_test and d) Y_test. This new split is performed in order to create the groups from which the algorithm will be trained and then tested. We use all the data of the cluster to train the algorithm and 240 hourly values that practically are translated in 10 days to test its results. The algorithm that is used is the k neighbors algorithm. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure however standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply “remember” all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree.) [50].

Table 21 - Training through sklearn and testing its predictions

```
            # Splitting the Dataset into Training and Testing set
            from sklearn.model_selection import train_test_split
            preferable_test_size = (240/len(X))
            X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = preferable_test_size, random_state = 0)

            # Fitting Forest Algorithm to the training set
            from sklearn.neighbors import KNeighborsRegressor
            regressor = KNeighborsRegressor(algorithm='auto',
leaf_size=30, metric='minkowski',
                                            metric_params=None, n_jobs=1,
n_neighbors=8, p=2, weights='distance')

            print("-----------------------------------------------------
-----------------------------")
            print ("Training active power loads with KNeighbors
Algorithm...")
            regressor.fit(X_train, y_train)
            print(regressor)
            print("")
            print("... Data Training Completed with KNeighbors
Algorithm")

            for j in range(len(X_test)):

                #Perform prediction on Test Data to check the mean
prediction error
                X_test_data = X_test[j]
                X_test_data = X_test_data.reshape(1,-1)

                y_pred = regressor.predict(X_test_data)
```

Having performed the algorithm training and having predicted 240 test values the question that reasonably arises is what will happen if the test values are not close to the real ones.
The answer is that the forecast made on the test values acts as a control for the forecast. As soon as we retrieve the predictions that are performed on the test data we use them in order to calculate the mean error that will occur.

Table 22 - Calculation of the test data overall mean prediction error

```
      error_check = abs(100-((y_pred/y_test[j])*100))
                error_check_dataset =
error_check_dataset.append(pd.DataFrame({'error_check': error_check},
index=[j]), ignore_index=False)
            print(error_check_dataset)

            if len(error_check_dataset)>=240:
                overall_error = error_check_dataset['error_check'].sum()
                mean_prediction_error =
float(overall_error/len(error_check_dataset))
                print("")
                print("The mean prediction error is: ",
mean_prediction_error)

            mean_test_error =
mean_test_error.append(pd.DataFrame({'mean_test_error':
mean_prediction_error}, index=[j]), ignore_index=True)
            print(mean_test_error)
```

```
if mean_prediction_error < float(20):

                # Predicting the Active Load Results
                som_KNeighbors_ap = som_KNeighbors_ap_dataset.iloc[:,
2:6].values
                som_KNeighbors_ap = som_KNeighbors_ap[i]
                som_KNeighbors_ap = som_KNeighbors_ap.reshape(1,-1)

                som_KNeighbors_ap_pred =
float(regressor.predict(som_KNeighbors_ap))

                # Inserting the data into the database

update_som_KNeighbors_forecasted_ap(som_KNeighbors_ap_pred, date_time[i])
                print("Forecasting process completed and Data being
inserted into the database")
                print("----------------------------------------------------
----------------------------------")
            else:
                print("The mean prediction error was higher than the
limit. SOM Clusters are going to re-run")
                SOM_KNeighbors_forecasting_ap()
```

If the mean error is higher than a limit that is specified by the user (in our case the overall max mean error is set to 20%) than all the process until this step is going to rerun. By saying that it is going to rerun we mean that all the forecasting process will be reset and it will start over from the creation of clusters so that the data to be reassigned and the forecast to rerun and retrieve new clusters that will reduce the mean error. Therefore, it is obvious that the clusters directly affect the outcome of the forecasting.

4.3. Final Stage of the forecasting process

If the control is successful than the forecasting process reaches in last stage which is the forecasting of the real dates. For each one of the dates that will be forecasted this process will start from the beginning. This means that for each one of the days that should be forecasted Divinus will get their corresponding past days and the cluster that they belong to and rerun all the aforementioned. Again, if the overall mean error that will occur will be higher than 20% the clusters will be deleted and recreated. This process will be followed for each day that will be forecasted. **So far the Divinus mean forecast error is 12%.**

Last but not least the forecast process has contains a timer. This timer is trigger the first time that the programs runs and then it automatically activates itself based one the time that we have set it. The same way is used to perfume the reactive load forecast.

Table 23 - The timer contained in Divinus Forecasting process

```
    seconds=1.0
    minutes=seconds*60
    hour=minutes*60
    day = (hour*24)*5
    threading.Timer(day, SOM_KNeighbors_forecasting_ap).start()
```

5. Divinus Website

Being able to perform data clustering and to forecast future loads, the only step that is left was to display all these information somewhere that users would be able to visit and get informed. For this step we choose to use the Django framework. Django is an open source high-level Python Web framework that encourages rapid development and clean, pragmatic design. It was built by experienced developers in order to take care much of the hassle of the Web development [51]. Django was designed to:

1. Help developers take applications from concept to completion as quickly as possible.
2. Contain dozens of extras that developers can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks.
3. Take security seriously and to help developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.
4. Be exceedingly scalable. Some of the busiest sites on the planet use Django's ability to quickly and flexibly scale to meet the heaviest traffic demands.
5. Be incredibly versatile. Companies, organizations and governments have used Django to build all sorts of things — from content management systems to social networks to scientific computing platforms [51].

Based on the aforementioned it was very easy for us to decide to cope with Django in designing our tool. Having set the project as Django based we were able to writte all the required code without worrying about how it will be transferred to the Internet or what modification we should make. As soon as we had a functional code the thing that was left to do was to set this code in Django in order to be viewable in the web.

5.1. Object-relational mapper

The first thing that was needed was to define our databases in Django. This was perfomed easily through Django's Model. A model is the single, definitive source of information about data. It contains the essential fields and behaviors of the data that are stored. Generally, each model maps to a single database table [52]. Our Model is shown in Table

Table 24 - Divinus Model.py file showing all the database information

```
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#   * Rearrange models' order
#   * Make sure each model has one field with primary_key=True
#   * Make sure each ForeignKey has `on_delete` set to the desired
behavior.
#   * Remove `managed = False` lines if you wish to allow Django to
create, modify, and delete the table
# Feel free to rename the models, but don't rename db_table values or
field names.
from django.db import models


class ActivePower(models.Model):
    date_time = models.DateTimeField(primary_key=True)
```

```
    active_power_kwh = models.DecimalField(max_digits=10,
decimal_places=2, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'active_power'


class AuthGroup(models.Model):
    name = models.CharField(unique=True, max_length=80)

    class Meta:
        managed = False
        db_table = 'auth_group'


class AuthGroupPermissions(models.Model):
    group = models.ForeignKey(AuthGroup, models.DO_NOTHING)
    permission = models.ForeignKey('AuthPermission', models.DO_NOTHING)

    class Meta:
        managed = False
        db_table = 'auth_group_permissions'
        unique_together = (('group', 'permission'),)


class AuthPermission(models.Model):
    name = models.CharField(max_length=255)
    content_type = models.ForeignKey('DjangoContentType',
models.DO_NOTHING)
    codename = models.CharField(max_length=100)

    class Meta:
        managed = False
        db_table = 'auth_permission'
        unique_together = (('content_type', 'codename'),)


class AuthUser(models.Model):
    password = models.CharField(max_length=128)
    last_login = models.DateTimeField(blank=True, null=True)
    is_superuser = models.BooleanField()
    username = models.CharField(unique=True, max_length=150)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=150)
    email = models.CharField(max_length=254)
    is_staff = models.BooleanField()
    is_active = models.BooleanField()
    date_joined = models.DateTimeField()

    class Meta:
        managed = False
        db_table = 'auth_user'


class AuthUserGroups(models.Model):
    user = models.ForeignKey(AuthUser, models.DO_NOTHING)
    group = models.ForeignKey(AuthGroup, models.DO_NOTHING)
```

```
    class Meta:
        managed = False
        db_table = 'auth_user_groups'
        unique_together = (('user', 'group'),)


class AuthUserUserPermissions(models.Model):
    user = models.ForeignKey(AuthUser, models.DO_NOTHING)
    permission = models.ForeignKey(AuthPermission, models.DO_NOTHING)

    class Meta:
        managed = False
        db_table = 'auth_user_user_permissions'
        unique_together = (('user', 'permission'),)


class DjangoAdminLog(models.Model):
    action_time = models.DateTimeField()
    object_id = models.TextField(blank=True, null=True)
    object_repr = models.CharField(max_length=200)
    action_flag = models.SmallIntegerField()
    change_message = models.TextField()
    content_type = models.ForeignKey('DjangoContentType',
models.DO_NOTHING, blank=True, null=True)
    user = models.ForeignKey(AuthUser, models.DO_NOTHING)

    class Meta:
        managed = False
        db_table = 'django_admin_log'


class DjangoContentType(models.Model):
    app_label = models.CharField(max_length=100)
    model = models.CharField(max_length=100)

    class Meta:
        managed = False
        db_table = 'django_content_type'
        unique_together = (('app_label', 'model'),)


class DjangoMigrations(models.Model):
    app = models.CharField(max_length=255)
    name = models.CharField(max_length=255)
    applied = models.DateTimeField()

    class Meta:
        managed = False
        db_table = 'django_migrations'


class DjangoSession(models.Model):
    session_key = models.CharField(primary_key=True, max_length=40)
    session_data = models.TextField()
    expire_date = models.DateTimeField()

    class Meta:
        managed = False
        db_table = 'django_session'
```

```
class ReactivePower(models.Model):
    date_time = models.DateTimeField(primary_key=True)
    reactive_power_kvar = models.DecimalField(max_digits=10,
decimal_places=2, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'reactive_power'


class SomActivePower(models.Model):
    som_row = models.IntegerField()
    som_column = models.IntegerField()
    date_time = models.DateTimeField(primary_key=True)
    som_active_power_kwh = models.DecimalField(max_digits=10,
decimal_places=2, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'som_active_power'


class SomActivePowerDayClusters(models.Model):
    som_row = models.IntegerField()
    som_column = models.IntegerField()
    date_active_power = models.DateField(primary_key=True)
    hour_0 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_1 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_2 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_3 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_4 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_5 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_6 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_7 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_8 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_9 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_10 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_11 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_12 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_13 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_14 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_15 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_16 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_17 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_18 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_19 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_20 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_21 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_22 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_23 = models.DecimalField(max_digits=10, decimal_places=2)

    class Meta:
        managed = False
        db_table = 'som_active_power_day_clusters'


class SomKneighborsForecastedActivePower(models.Model):
```

```
    date_time = models.DateTimeField(primary_key=True)
    som_kneighbors_forecasted_active_power_kwh =
models.DecimalField(max_digits=10, decimal_places=2, blank=True,
null=True)

    class Meta:
        managed = False
        db_table = 'som_kneighbors_forecasted_active_power'


class SomKneighborsForecastedReactivePower(models.Model):
    date_time = models.DateTimeField(primary_key=True)
    som_kneighbors_forecasted_reactive_power_kvar =
models.DecimalField(max_digits=10, decimal_places=2, blank=True,
null=True)

    class Meta:
        managed = False
        db_table = 'som_kneighbors_forecasted_reactive_power'


class SomReactivePower(models.Model):
    som_row = models.IntegerField()
    som_column = models.IntegerField()
    date_time = models.DateTimeField(primary_key=True)
    som_reactive_power_kvar = models.DecimalField(max_digits=10,
decimal_places=2, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'som_reactive_power'


class SomReactivePowerDayClusters(models.Model):
    som_row = models.IntegerField()
    som_column = models.IntegerField()
    date_reactive_power = models.DateField(primary_key=True)
    hour_0 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_1 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_2 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_3 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_4 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_5 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_6 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_7 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_8 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_9 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_10 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_11 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_12 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_13 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_14 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_15 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_16 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_17 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_18 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_19 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_20 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_21 = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    hour_22 = models.DecimalField(max_digits=10, decimal_places=2)
    hour_23 = models.DecimalField(max_digits=10, decimal_places=2)

    class Meta:
        managed = False
        db_table = 'som_reactive_power_day_clusters'
```

5.2. Template

Being a web framework, Django needs a convenient way to generate HTML dynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. A Django project can be configured with one or several template engines (or even zero if no templates are required). Django defines a standard API for loading and rendering templates regardless of the backend. Loading consists of finding the template for a given identifier and preprocessing it, usually compiling it to an in-memory representation. Rendering means interpolating the template with context data and returning the resulting string. The Django template language is Django's own template system. Until Django 1.8 it was the only built-in option available. It's a good template library even though it's fairly opinionated and sports a few idiosyncrasies [53].
Our template [54] is built on HTML, CSS and Javascript. Through our model we pass the required data to the python Views page which is the file that links the data that we want infuse with the template.

Table 25 - Divinus Views.py file

```
'''
Created on 7 Μαρ 2018

@author: d.mele
'''

#import json
from django.shortcuts import render
from django.core import serializers
#from django.http import JsonResponse
#from django.shortcuts import loader
#from django.http import HttpResponse

from Core.models import ActivePower, ReactivePower
from Core.models import SomActivePower, SomReactivePower
from Core.models import SomKneighborsForecastedActivePower,
SomKneighborsForecastedReactivePower

# Create your views here.

def  index(request):
    Active_Power_queryset =
ActivePower.objects.exclude(active_power_kwh__isnull=True).order_by('date
_time')
    Active_Power_json = serializers.serialize('json',
Active_Power_queryset, fields=('active_power_kwh'))

    Limited_Active_Power_queryset =
ActivePower.objects.exclude(active_power_kwh__isnull=True).order_by('-
date_time')[:1000][::-1]
```

```
    Limited_Active_Power_json = serializers.serialize('json',
Limited_Active_Power_queryset, fields=('active_power_kwh'))

    Reactive_Power_queryset =
ReactivePower.objects.exclude(reactive_power_kvar__isnull=True)
    Reactive_Power_json = serializers.serialize('json',
Reactive_Power_queryset, fields=('reactive_power_kvar'))

    Limited_Reactive_Power_queryset =
ReactivePower.objects.exclude(reactive_power_kvar__isnull=True).order_by(
'-date_time')[:1000][::-1]
    Limited_Reactive_Power_json = serializers.serialize('json',
Limited_Reactive_Power_queryset, fields=('reactive_power_kvar'))

    SOM_KNeighbors_Forecasted_AP_queryset =
SomKneighborsForecastedActivePower.objects.all().order_by('date_time')
    SOM_KNeighbors_Forecasted_AP_json = serializers.serialize('json',
SOM_KNeighbors_Forecasted_AP_queryset,
fields=('som_kneighbors_forecasted_active_power_kwh'))

    SOM_KNeighbors_Forecasted_RP_queryset =
SomKneighborsForecastedReactivePower.objects.all().order_by('date_time')
    SOM_KNeighbors_Forecasted_RP_json = serializers.serialize('json',
SOM_KNeighbors_Forecasted_RP_queryset,
fields=('som_kneighbors_forecasted_reactive_power_kvar'))

    SOM_AP_clusters_queryset = SomActivePower.objects.all()
    SOM_AP_clusters_json = serializers.serialize('json',
SOM_AP_clusters_queryset)

    SOM_RP_clusters_queryset = SomReactivePower.objects.all()
    SOM_RP_clusters_json = serializers.serialize('json',
SOM_RP_clusters_queryset)

    return render(request,'Core/test5.html',
{'Active_Power_json':Active_Power_json,

'Limited_Active_Power_json':Limited_Active_Power_json,
                                            'Reactive_Power_json':
Reactive_Power_json,

'Limited_Reactive_Power_json': Limited_Reactive_Power_json,

'SOM_KNeighbors_Forecasted_AP_json': SOM_KNeighbors_Forecasted_AP_json,

'SOM_KNeighbors_Forecasted_RP_json': SOM_KNeighbors_Forecasted_RP_json,
                                            'SOM_AP_clusters_json':
SOM_AP_clusters_json,
                                            'SOM_RP_clusters_json':
SOM_RP_clusters_json})
```

As soon as we have set the views.py we are ready to modify our template [54] in the best possible way in order to display the required information. Figures 22, 23, 24, 25, 26 show some pages of the Divinus website.

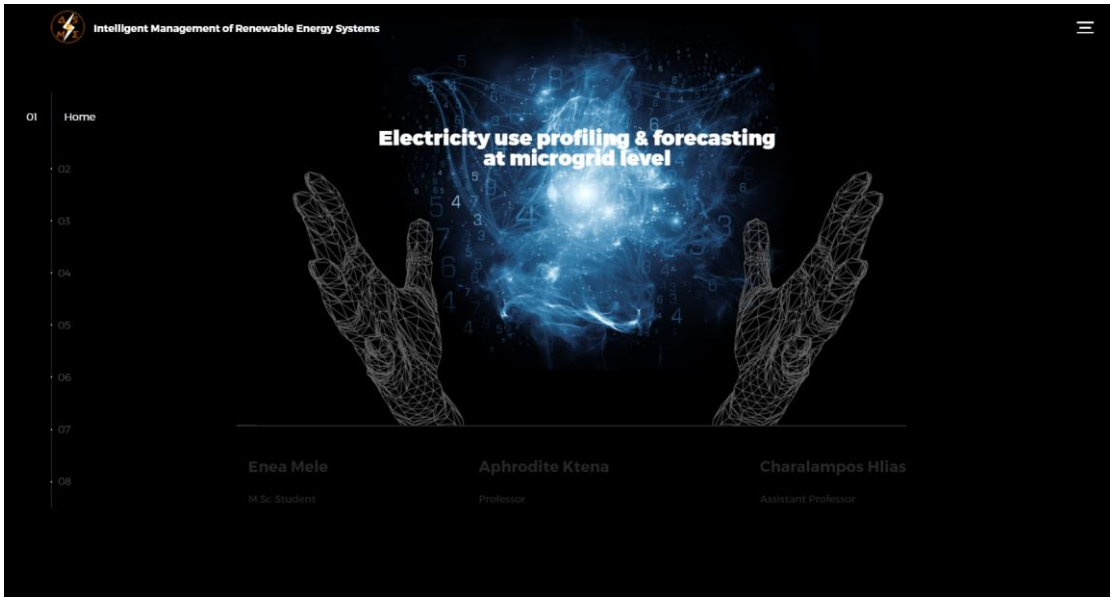


Figure 22 - Divinus Front Page

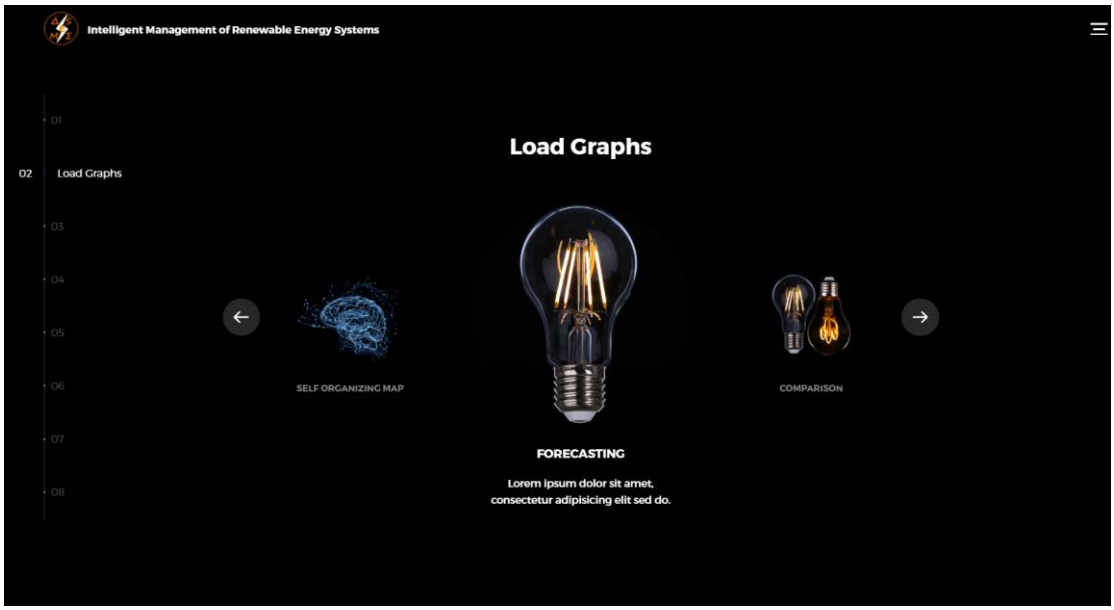


Figure 23 - Divinus Menu Selection

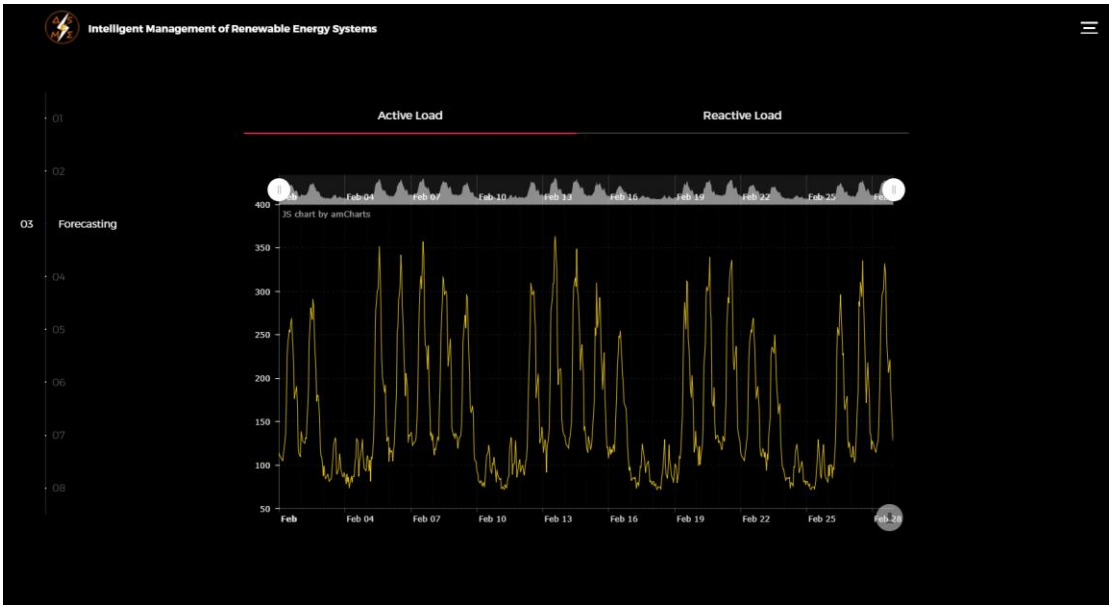


Figure 24 - Divinus Forecasting Page

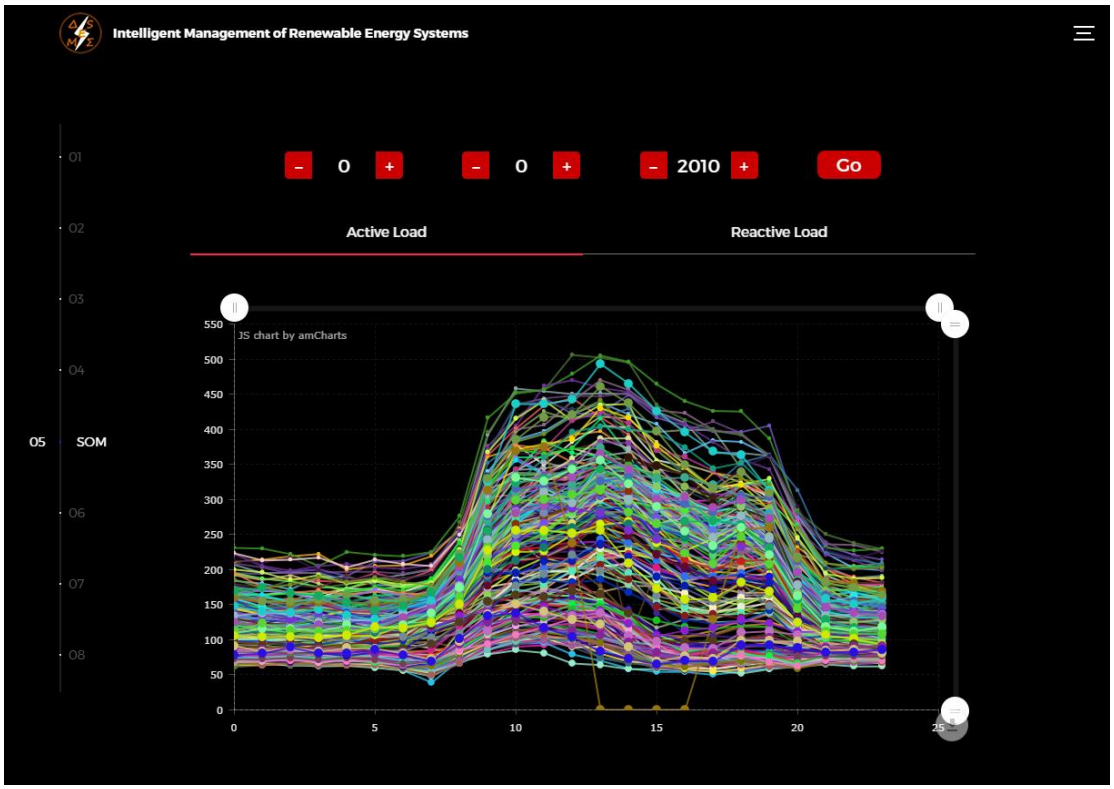


Figure 25 - Divinus SOM Clusters Page

Figure 26 - Divinus Comparison Page between Real & Forecasted Load

6. Conclusion

This master thesis had two goals. The first one was to create a tool that could be perform both use profiling and load forecasting. Regarding the use profiling, after having done a lot of research we arrived to the decision that for the tool that we wanted to build the most suitable algorithm to be used was the Self Organizing Map. As for the forecasting algorithm is was decided in a second phase due to the fact that we had to have the results of the SOM algorithm first in order to proceed with the forecast. As soon as we had the SOM results in our hands we started experimenting with machine learning libraries containing several forecasting algorithms, but none of them was as good as the k-neighbors algorithm through which we managed to perform predictions with a forecast error of only 12%.

The second goal basically was interrelated with the first one, as we wanted to see if forecasts could occur based on the data of the clustering algorithm. If that could happen then we would be able to create a methodology based on which we could forecast the consumptions of various consumers within a microgrid based on their user profiles.

At the end of this thesis we are able to say that we successfully fulfilled the first goal and made the first basic and promising steps towards the completion of the second. Many steps still need to be taken in terms of creating a methodology through which we can handle the consumption of different consumers within a microgrid.

7. Bibliography

[1] Hernández L, Baladrón C, Aguiar JM, Carro B, Sánchez-Esguevillas A, Lloret J, "Short-Term Load Forecasting for Microgrids Based on Artificial Neural Networks.," *Energies,* vol. 6, no. 3, pp. 1385-1408, 2013.

[2] Treaty Establishing A Constitution For Europe, "EU treaties," [Online]. Available: https://europa.eu/european-union/sites/europaeu/files/docs/body/treaty_establishing_a_constitution_for_europe_en.pdf. [Accessed 4 February 2018].

[3] Jimyung, K., Jee-Hyong, L., "Electricity Customer Clustering Following Experts' Principle for Demand Response Applications," *Energies 2015,* vol. 8, no. 10, pp. 12242-12265, 2015.

[4] S. M. Bidoki, N. Mahmoudi-Kohan, S. Gerami, "Comparison of several clustering methods in the case of electrical load curves classification," in *16th Conference on Electrical Power Distribution Networks (EPDC),* IEEE (2011).

[5] G. Phanendra Babu, M. Narasimha Murty and S. Sathiya Keerthi, "A Stochastic Connectionist Approach for Global Optimization with Application to Pattern Clustering," *IEEE Trans. Systems, Man, And Cybernetics-Part B: Cybernetics,* vol. 30, no. 1, pp. 10-24, Feb 2000.

[6] G. Chicco, R. Napoli, F. Piglione, P. Postolache,M. Scutariu and C. Toader, "Load Pattern-Based Classification of Electricity Customers," *IEEE Trans. Power Systems,* vol. 19, no. 2, pp. 1232-1239, May 2004.

[7] W. Li, J. Zhou, X. Xiong and J. Lu, "A Statistic-Fuzzy Technique for Clustering Load Curves," *IEEE Trans. Power Systems,* vol. 22, no. 2, pp. 890-891, May 2007.

[8] G. Chicco, R. Napoli, P. Postolache, M. Scutariu, and C. Toader, "Electric energy customer characterisation for developing dedicated market strategies," *Power Tech Proceedings,* vol. 1, 2001.

[9] V. Figueiredo, F. Rodrigues, Z. Vale and J. B. Gouveia, "An Electric Energy Consumer Characterization Framework Based on Data Mining Techniques," *IEEE Trans. Power Syst.,* vol. 20, no. 2, pp. 596-602, May 2005.

[10] G. J. Tsekouras, N. D. Hatziargyriou and E. N. Dialynas, "Two-Stage Pattern Recognition of Load Curves for Classification of Electricity Customers," *IEEE Trans. Power Syst.,* vol. 22, no. 3, pp. 1120-1128, Aug 2007.

[11] G. Chicco, R. Napoli and F. Piglione, "Application of Clustering Algorithms and Self Organizing Maps to Classify Electricity Customers," *Proc. 2003,* vol. 1.

[12] S. Chunhua, B. Feng, Z. Jianying. T. Tsuyoshi and S. Kouichi, “Privacy-Preserving Two-Party K-Means Clustering via Secure Approximation,” *in Proc. 2007 Advanced Information Networking and Applications Workshops 21st International Conf.,* vol. 1, pp. 385 - 391, 2007.

[13] N. Mahmoudi-Kohan, M. P. Moghaddam, M. K. Sheikh-EI-Eslami and S. M. Bidaki, “Improving WFA K-means Technique for Demand Response Programs Applications,” in *accepted for presentation, IEEE, General Meeting 2009*.

[14] S. Nasser, R. Alkhaldi and G. Vert, “A Modified Fuzzy K-means Clustering using Expectation Maximization,” in *Proc. 2006 IEEE International Conf, pp. 231-235*, 2006.

[15] K. A. Abdul Nazeer, M. P. Sebastian, “Improving the Accuracy and Efficiency of the k-means Clustering Algorithm,” in *Proceedings of the World Congress on Engineering 2009*, London, U.K., 2009.

[16] Andrea Trevino, “DataScience.com,” [Online]. Available: https://www.datascience.com/blog/k-means-clustering. [Accessed 05 02 2018].

[17] Maciej Pacula, “Maciej Pacula,” [Online]. Available: http://blog.mpacula.com/2011/04/27/k-means-clustering-example-python/. [Accessed 08 02 2018].

[18] L. A. Zadeh, “Fuzzy logic and approximate reasoning,” *Synthese,* vol. 30, pp. 407-428, 1975.

[19] Carl G. Looney, “Pattern Recognition,” [Online]. Available: www.cse.unr.edu/~looney/cs773b/1162_C09.pdf. [Accessed 07 02 2018].

[20] Wu Q, Qi X, Fuller E, Zhang C., “'Follow the Leader': a centrality guided clustering and its application to social network analysis,” *The Scientific World Journal,* vol. 2019, 2013.

[21] G. Chicco, R. Napoli, F. Piglione, “Comparisons among clustering techniques for electricity customer classification,” *IEEE Transactions on Power Systems,* vol. 21, no. 2, pp. 933 - 940, May 2006.

[22] G. Chicco, R. Napoli, P. Postolache, M. Scutariu, C. Toader, “Customer characterization options for improving the tariff offer,” *IEEE Transactions on Power Systems,* vol. 18, no. 1, pp. 381-387, Feb. 2003.

[23] M. R. Anderberg, Cluster Analysis for Applications, New York: Academic Press, 1973.

[24] B. S. Everitt, Cluster Analysis 3rd edition, London, U.K: Arnold and Halsted, 1993.

[25] J. H. Ward, “Hierarchical grouping to optimize an objective function,” *J. Amer. Stat. Assoc.,* vol. 58, pp. 236-244, 1963.

[26] Teuvo Kohonen, Self-Organizing Maps, Berlin, Heidelberg: Springer, 1995.

[27] Teuvo Kohonen, Erkki Oja, Olli Simula, Ari Visa, Jari Kangas, "Engineering applications of the self-organizing map. Manuscript submitted to a journal.," *Proceedings of the IEEE,* vol. 84, no. 10, pp. 1358-1384 , 1996 .

[28] D.C. Park, M.A. El-Sharkawi, R.J. Marks II, L.E. Atlas and M.J. Damborg, "Electric Load Forecasting Using An Artificial Neural Networks," *IEEE Transactions on Power Engineering,* vol. 6, pp. 442-449, May. 1991.

[29] K. Y. Lee and J. H. Park, "Short-Term Load Forecasting Using an Artificial Neural Network," *IEEE Transactions on Power Systems,* vol. 7, pp. 127-132, Feb. 1992.

[30] Alireza Khotanzad, Rey-Chue Hwang, Alireza Abaye and Dominic Maratukulam, "An Adaptive Modular Artificial Neural Network Hourly Load Forecaster and its Implementation at Electric Utilities," *IEEE Transactions on Power Systems,* vol. 10, pp. 1716-1721, Aug. 1995.

[31] Jaakko Hollmen , "Self-Organizing Map (SOM)," [Online]. Available: http://users.ics.aalto.fi/jhollmen/dippa/node9.html. [Accessed 08 07 2018].

[32] Fernando Bação, Victor Lobo, Marco Painho, "Self-organizing Maps as Substitutes for K-Means Clustering," *Fifth International Conference on Computational Science (ICCS 2005),* vol. 3, pp. 476 - 483, 22-25 May 2005.

[33] "TechTerms," Sharpened Productions, [Online]. Available: https://techterms.com/. [Accessed 12 02 2018].

[34] C. Mohan, "History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla," *Proc. 16th ACM Int'l Conference Extending Database Technology (EDBT 13),* p. 11–16, 2013.

[35] Forrest Stroud, "ServerWatch," IT Business Edge Network, [Online]. Available: https://www.serverwatch.com/server-trends/slideshows/top-10-enterprise-database-systems-to-consider-2015.html. [Accessed 12 02 2018].

[36] Craig S. Mullins, "TechTarget," SearchDataManagement.com, [Online]. Available: http://searchdatamanagement.techtarget.com/feature/Which-relational-DBMS-is-best-for-your-company. [Accessed 12 02 2018].

[37] The PostgreSQL Global Development Group, "PostgreSQL," [Online]. Available: https://www.postgresql.org/about/. [Accessed 20 02 2018].

[38] Lisa Smith, "What PostgreSQL has over other open source SQL databases: Part I," Compose, [Online]. Available: https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/. [Accessed 21 02 2018].

[39] Sebastian Raschka, Python Machine Learning, Birmingham, UK: Packt Publishing, 2015.

[40] Kernighan, Brian W.; Ritchie, Dennis M. (). , The C Programming Language (1st ed.), Englewood Cliffs, NJ: Prentice Hall, Feb. 1978.

[41] "Learn C++," Programiz, [Online]. Available: https://www.programiz.com/cpp-programming. [Accessed 14 02 2018].

[42] Arpan Chakraborty, "Languages and Libraries for Machine Learning," Udacity, [Online]. Available: https://blog.udacity.com/2016/04/languages-and-libraries-for-machine-learning.html. [Accessed 20 02 2018].

[43] Ken Arnold, James Gosling, David Holmes, The Java Programming Language (The Java Series), Boston, MA, USA: Addison-Wesley Longman Publishing Co., 1996.

[44] Ross Ihaka and Robert Gentleman, "R: A Language for Data Analysis and Graphics," *Journal of Computational and Graphical Statistics*, vol. 5, no. 3, pp. 299-314, Sep. 1996.

[45] Nicholas C. Zakas, Professional JavaScript for Web Developers, Crosspoint Boulevard, Indianapolis: Wiley E-Text, Jan 2012.

[46] Masoud Nosrati, "Python: An appropriate language for real world programming," *World Applied Programming*, vol. 1, no. 2, pp. 110-117, June 2011.

[47] Christina Voskoglou, "What is the best programming language for Machine Learning?," [Online]. Available: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7. [Accessed 20 02 2018].

[48] Vsion Mobile, "State of the Developer Nation Q1 2017," [Online]. Available: http://www.mwc.gr/presentations/2017/konstantinou.pdf. [Accessed 20 02 2018].

[49] Giuseppe Vettigli, "MiniSom: minimalistic and NumPy-based implementation of the Self Organizing Map," 15 September 2013. [Online]. Available: https://github.com/JustGlowing/minisom. [Accessed 24 May 2018].

[50] "scikit-learn," scikit-learn developers, [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed 2018 May 26].

[51] Django Software Foundation, "django," Django Software Foundation, [Online]. Available: https://www.djangoproject.com/start/overview/. [Accessed 26 May 2018].

[52] Django Software Foundation, “django,” Django Software Foundation, [Online]. Available: https://docs.djangoproject.com/en/2.0/topics/db/models/. [Accessed 26 May 2018].

[53] Django Software Foundation, “django,” Django Software Foundation, [Online]. Available: https://docs.djangoproject.com/en/2.0/topics/templates/. [Accessed 26 May 2018].

[54] Bucky Maler, “Global,” [Online]. Available: http://buckymaler.com/global/#0. [Accessed 26 May 2018].