




Article

TSxtend: A Tool for Batch Analysis of Temporal Sensor Data

Roberto Morcillo-Jimenez , Karel Gutiérrez-Batista  and Juan Gómez-Romero * 

Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain

* Correspondence: jgomez@decsai.ugr.es

Abstract: Pre-processing and analysis of sensor data present several challenges due to their increasingly complex structure and lack of consistency. In this paper, we present TSxtend, a software tool that allows non-programmers to transform, clean, and analyze temporal sensor data by defining and executing process workflows in a declarative language. TSxtend integrates several existing techniques for temporal data partitioning, cleaning, and imputation, along with state-of-the-art machine learning algorithms for prediction and tools for experiment definition and tracking. Moreover, the modular architecture of the tool facilitates the incorporation of additional methods. The examples presented in this paper using the ASHRAE Great Energy Predictor dataset show that TSxtend is particularly effective to analyze energy data.

Keywords: time series; pre-processing; prediction; machine learning; deep learning



Citation: Morcillo-Jimenez, R.; Gutiérrez-Batista, K.; Gómez-Romero, J. TSxtend: A Tool for Batch Analysis of Temporal Sensor Data. *Energies* **2023**, *16*, 1581. <https://doi.org/10.3390/en16041581>

Academic Editor: Fernando Morgado-Dias

Received: 24 December 2022

Revised: 23 January 2023

Accepted: 1 February 2023

Published: 4 February 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The development and growth of information and communication technologies have propitiated the daily generation of massive data. Today, much of the generated data come from sensor data. Temporal sensor data have become of great interest to the academic and private sectors, as studying this sort of data allows for studies of the evolution of data over time, providing end-users with robust algorithms and tools for decision-making.

There are several applications that use sensor data for various purposes [1,2], such as handling and managing measures such as temperature, humidity, pressure, gas, optical, and many others. Many companies in different industries are becoming increasingly aware of the great potential that the research and exploitation of temporal sensor data can offer [3,4].

There are many challenges in dealing with temporal sensor data. In the following, we mention the main challenges concerning this sort of data:

1. One of the main challenges is dealing with the large volumes of data that are generated by sensors each day. This can make storing, managing, and processing the data difficult, requiring specialized tools and techniques.
2. Another challenge is the variability and noise in the data, which makes it difficult to identify trends and patterns. This may require advanced data filtering and cleaning methods to remove irrelevant or inaccurate information.
3. Additionally, the lack of consistency and complex structure inherent in temporal sensor data make its processing and analysis even more difficult. Sophisticated algorithms and techniques may be required to analyze and interpret the data appropriately.
4. Furthermore, temporal sensor data are often harvested from heterogeneous sources. This can be challenging, requiring efficient integration approaches to handle the data in a timely manner.

All the aforementioned challenges hinder the definition of a workflow that allows for promising results to be obtained. Furthermore, selecting the most suitable algorithm to solve the problem constitutes another burdensome task. Solving this problem is hard work due to the nature and variability of each related problem.

One of the most significant challenges in prediction problems for time series and any other data type is the time needed to design the correct strategy for the experiment. Different studies focus on the application of specific algorithms to a data series using a static configuration [5–7]; in other words, without offering the possibility of parameterising the different experiments. This consumes a great deal of time for each experiment definition, thus slowing down the researcher’s main objective.

As stated before, time series data have gained the attention of the entire research community, as its analysis allows for an understanding of changes in the data over time. By analyzing the trends and patterns in the data, insights into how the data are evolving can be acquired, and predictions about future results can be made. This is particularly useful in fields where changes over time can significantly impact decision-making and strategy. Additionally, temporal sensor data analysis can help to identify potential issues or anomalies, allowing for timely interventions and corrective actions.

This paper proposes a tool called TSxtend for time series analysis. TSxtend presents a modular architecture and standardises the different stages of experimentation, creating a workflow through a simple configuration file. The proposed tool allows for the end-users to work on time series data without programming knowledge using the available techniques and focusing on developing the research. The tool enables data filtering and cleaning to remove incorrect or nonessential information. Finally, TSxtend has the ability to execute deep prediction and machine learning algorithms that are commonly utilized in various domains, such as energy [8], medicine [9], or geoscience [10,11]. This allows for a wide range of applications and flexibility in its usage. The results are presented through the visualization module, enabling end users to analyze the results at different workflow stages. A summary of the main contributions of this paper is as follows:

- The paper’s main contribution is the development of a tool called TSxtend for time series analysis, which has a modular architecture and standardizes different stages of experimentation.
- The tool allows for end-users to work on time series data without programming knowledge, simplifying the process of data filtering and cleaning to remove incorrect or nonessential information.
- TSxtend offers the possibility of executing prediction algorithms and visualizing the results through a visualisation module, enabling end-users to analyse the results at different workflow stages.

It is important to note that, to the best of the authors’ knowledge, there is no existing tool in the literature that has the same abilities as TSxtend. This tool is unique in its ability to standardize the experimentation process, simplify time series analysis for end-users without programming knowledge, and provide a visualisation module for a better analysis of results.

In this paper, we apply the proposed tool in an energy consumption problem. We obtained the data from the prediction features contest on the Kaggle platform called ASHRAE—Great Energy Predictor III [12]. The database comprised three years of hourly meter readings from more than a thousand buildings in different locations worldwide. It should be noted that the database belongs to ASHRAE, a large building technology association [13].

The rest of the paper is structured as follows: Section 2 reviews previous work on this topic. Section 3 presents the design and functionalities details of the presented tool (Tsxtend). Section 4 extends the description of the different modules that comprise the tool. In order to showcase the feasibility of the proposed tool, Section 5 presents a real-world use-case using TSxtend and discusses the obtained results. Finally, in Section 6, the conclusions and future research are presented.

2. Related Works

Many applications aim to make the work of researchers working with time series data easier [14–18]. This is driven by the need to abstract programming knowledge to a higher level, allowing for researchers who are not experts in data science to focus on studying the data rather than the technical details of the algorithm. Some tools that can be used

to achieve this abstraction, isolating the researcher from the technical complexities, are described in the following.

To make the literature review easier to understand and better highlight the novelty of this research, a comparative study of libraries was conducted. Various aspects of the libraries were analysed, such as their ability to collect and use heterogeneous data sources, the capacity of the processing tools available within the library, and the REST API abilities of the libraries. Furthermore, we also evaluated whether the libraries can implement Artificial Intelligence (AI) algorithms, if they have been used in real-world use-cases and if they are user-friendly. The ease of use is particularly important when the libraries are used by non-expert data-mining users. Through this analysis, we aim to provide a comprehensive comparison of the different libraries and their features, to aid in the understanding of the significance of this research.

One example of such a tool is Enlopy [19], which is an open-source tool developed in Python that offers a variety of methods for processing, analyzing, and plotting time series data. This tool has modules that are primarily focused on studying time series data, with capabilities such as analysis techniques, graphing, data augmentation, and feature extraction, mainly in the energy domain. However, it should be noted that this tool requires a significant amount of programming knowledge, making it inaccessible to researchers who are not experts in data science.

The TSSA tool [14] primarily focuses on the preprocessing stage, to obtain the correlation between the resistance variables of memories in different devices. This is not suitable for work with heterogeneous data and requires extensive knowledge in data science. Additionally, it does not have a REST API for data retrieval, which limits its functionality and accessibility for users.

Another tool that addresses the work with time series is tsfresh [20]. This tool enables the study of time series data by extracting features and training simple classification or regression models. However, it should be noted that this tool requires a significant amount of programming knowledge to be used to its full extent. Additionally, it does not offer a workflow feature to guide users through their work process. In [18], a Visual Warning Tool for Financial Time Series (VWSTFTS) is presented based on scaling analysis. The proposed method uses the time-dependent Generalised Hurst Exponent (GHE) method to analyse financial time series and identify temporal patterns in GHE profiles. By applying this methodology and using a visual tool, the researchers can analyse significant and peripheral stock market indices. The proposed method offers a new way of identifying patterns in financial time series data and can be used to provide early warning signals for market fluctuations. However, the tool is limited in its capabilities and does not support AI techniques. Additionally, it is primarily used in the financial domain and might not be suitable for other fields.

Darts [21] is one of the most comprehensive tools available, providing a wide range of algorithms for working with time series. Darts supports both univariate and multivariate time series and models. Acycle [17] is a specialized software for paleoclimate research and education that focuses on signal processing, particularly for cyclostratigraphy and astrochronology. It includes various models, such as sedimentation noise and sedimentation rate, which are specific to sedimentary research. Acycle's fully implemented graphical user interface makes it easy for users to operate and navigate the software, making it user-friendly for both researchers and educators. It should be noted that neither Darts nor Acycle includes a preprocessing phase, meaning that the data need to be cleaned, transformed and prepared before being used with these tools. This could mean that additional steps and resources are required for the effective use of these tools.

Another state-of-the-art tool for time series analysis is Kats [22]. This tool aims to make time series analysis more accessible to researchers with a solid background in data science. Kats offers a variety of forecasting algorithms, such as individual forecasting models, ensemble models, a self-supervised learning model (meta-learning), backtesting, hyperparameter fitting and empirical prediction intervals. This tool is a comprehensive and powerful option for researchers looking to perform advanced time series analysis.

TSFEL [23] is a tool that primarily focuses on the preprocessing stage of time series analysis, providing a range of options for exploratory analysis and feature extraction. One of its notable features is the inclusion of a set of unit tests to verify the runs. However, like many other tools, TSFEL can be challenging to use for researchers without strong programming skills.

SSTS [16] is a tool for searching patterns in time series data. It utilises a syntactic approach, analysing the structure and relationships within the data. SSTS allows for users to specify complex patterns using formal grammar and search for instances of a pattern. The tool can identify patterns that are difficult to detect using traditional methods such as statistical analysis or machine learning. Additionally, SSTS can extract features from time series data for further analysis or to train predictive models. It is a powerful tool for searching patterns in time series data that can be used in various fields, such as finance, healthcare, and transportation.

cleanTS [15] is an automated tool for cleaning univariate time series data. It utilises machine learning techniques to remove noise, outliers and missing values, helping to improve the accuracy and reliability of time series analysis. It also makes it easier to identify patterns and trends in the data. cleanTS can also be used to interpolate missing data and resample the data at different time scales. This tool can be applied in various domains, such as financial time series, sensor data and other time series data. It is a useful tool for the preprocessing of time series data, making it more reliable and accurate for further analysis.

Another library that allows for working with time series data is GreyKite [24]. This tool provides preprocessing capabilities as well as prediction models for heterogeneous data. One of its unique features is its own algorithm, which can be used for creating prediction models, called *Silverkite*. GreyKite is also aimed at data scientists. Another complete tool is AutoTS [25], which utilizes other libraries for the generation of prediction models with the help of a framework. As with other tools, it requires a good understanding of programming. Table 1 provides a comparison of the libraries mentioned above with the tool proposed in this paper.

Table 1. Comparison of the proposed tool with the tools mentioned earlier for analysing temporal sensor data. Legend: ✓—Feature supported, ✗—Feature not supported, ~—Feature not fully supported or with explicit limitations, ?—Unknown; information not available about the feature.

Name	Heterogeneous Data Collection	Preprocessing Data	REST API	AI Tools	Applications Wide Range of Areas	User Friendly
Enlopy	✓	✓	✗	✗	✗	✗
TSfresh	✗	✓	✗	✓	?	✗
Kats	✗	✗	✗	✓	✓	~
Darts	✓	✓	✗	✓	✓	✗
SSTS	?	✓	✗	✓	✓	✗
TSSA	✗	✓	✗	✗	?	✗
cleanTS	✗	✓	✗	✓	✓	✗
GreyKite	✓	✓	✗	✓	✓	✗
TSfel	✓	✓	✗	✗	✗	~
AutoTs	✓	✗	✗	✓	✓	✗
Acycle	✓	✗	✗	✓	✗	✓
VWSTFTS	✓	✓	✗	✗	✗	~
TSxtend	✓	✓	✓	✓	✗	✓

The proposed tool is designed to facilitate data processing for non-expert users with a simple, user-oriented interface. The tool is primarily aimed at the processing of time series data such as energy consumption, but can also process and analyse various other types of data. Additionally, the library includes commonly used data processing, cleaning, and transformation tools. Its design allows for the easy integration of new methods and algorithms. A wide range of data analysis algorithms is also available. One of the key strengths of TSxtend is its user-oriented design and communication through its REST API, which makes the tool dynamic and easy to use.

All the presented tools are comprehensive libraries for analysing temporal sensor data, helping researchers to study these types of data without the need for extensive knowledge of data science. However, almost all of these tools require programming knowledge, which can be a hindrance to researchers who want to focus solely on data analysis. In contrast, TSxtend not only provides a tool for working with temporal sensor data, but also includes an architecture that allows for researchers to create a workflow for their research by editing a configuration file. This enables researchers to conduct experiments step by step, analysing the data at each stage, without the need for programming knowledge.

3. Design and Functionalities of TSxtend

TSxtend comprises a series of modules for specific tasks. The modules that comprise this tool are grouped into different data science paradigms, each of which offers the possibility of executing different algorithms. In the following section, the software design of our tool will be introduced, and a brief explanation of its functionalities will be provided.

3.1. Design

The main idea behind designing this tool is to create a series of dynamic and flexible modules, allowing for a wide range of operations on the input data and providing a final result. One of the primary advantages of this design approach is that it provides a robust tool, where each component can be executed separately. This allows for each step of the pipeline to be run separately, without necessarily having to run the whole process every time the experiment is performed.

As shown in Figure 1, TSxtend is organised into five modules: one module to store the loggers of the tool, a configuration module, a module to process data, a module to execute machine learning algorithms and a module to execute deep learning algorithms. In the following, we briefly introduce each of these modules.

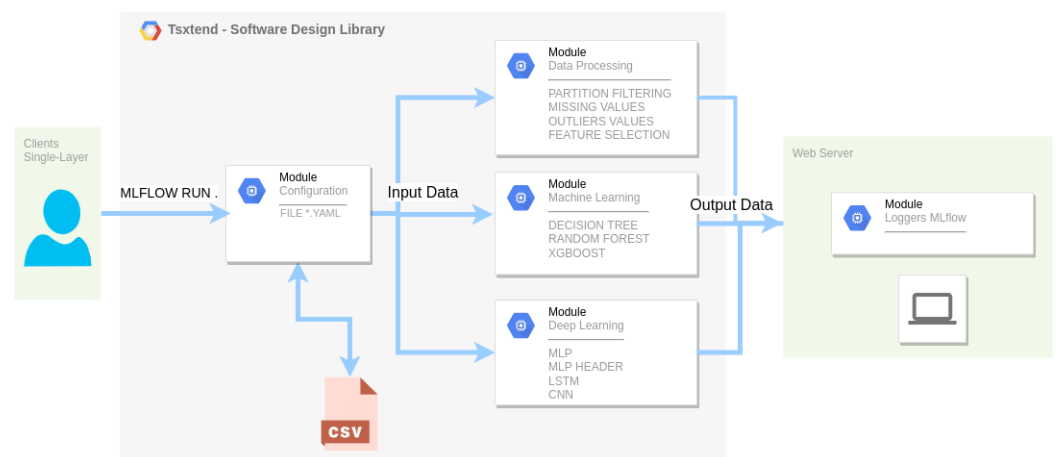


Figure 1. Design of the proposed tool.

3.2. Functionalities

Firstly, through a plain text file, the configuration module allows for the insertion of a series of parameters to configure the experimentation and automate the whole process without programming knowledge.

The data processing module includes a series of data preprocessing algorithms, such as eliminating missing values, eliminating out-of-range data in our input data, and a series of algorithms aiming to obtain the different relationships between the different characteristics of our input data. These algorithms were chosen because they cause one of the most common problems when working with temporal series data.

The machine learning module includes the standard algorithms used in this paradigm, such as XGBoost [26], Decision [27] and Regression Trees [28]. These algorithms were

chosen because they are among the most widely used in the state-of-the-art to solve prediction problems based on structured time series data.

In the deep learning module, the same selection policy for the integrated algorithms was followed as in the previous module. The most widely used algorithms for time series focused on the deep learning paradigm were selected, such as Convolutional Neural Network (CNN) [29], Long-Short Term Memory (LSTM) [30] and Multilayer Perceptron [31].

Finally, the module recorders represent the results and can track them. For this, we integrated Mlflow [32] because it is a tool that is used to keep track of the experiments performed by any user. This tool helps to show the status of the experiments with an accessible, user-friendly interface. In the following sections, we will explain the functionalities of each module in more detail.

4. Module Descriptions

The proposed tool presents a modular architecture that facilitates the effortless incorporation of future functionalities. In the subsequent sections, the characteristics and functionalities of each module will be described in detail.

4.1. Coordination Module

Our tool is mainly focused on carrying out exploratory data analyses in a fast and agile way. In this way, the end-users gain a better understanding of the data, and can identify potential issues or problems with the data and quickly develop more refined analyses. The tool offers a set of prediction algorithms that are mainly used to solve time series problems, such as neural networks. Although the algorithms provided by the tool can be applied to any data type, the proposed library specialises in temporal sensor data regarding energy.

Several works address energy-related problems using data science techniques to identify opportunities for energy-efficiency improvements [33–35]. However, only a few can automate the entire experimentation process. Furthermore, end-users require programming skills to use the library.

The configuration module is one of the most important. It allows for different executions of the defined experiment to be planned. The objective of this module is to configure a road map so that users can schedule the whole experimentation strategy. The main advantage of this approach is that it can endow end-users with a robust tool, which can be handled without programming knowledge.

The controller file, called *MLproject*, contains a definition of the algorithms that are to be used and the configuration parameters. The file reads the data in the YAML files and executes the user-defined workflow automatically.

The entire roadmap of the experimentation is configured through a series of YAML files [36], whose primary function is to define and serialize the different processes for all kinds of programming languages. Each file has a series of parameters that the researcher can edit. The parameters can be consulted in [37], where all the different options accepted by the tool have been detailed. The nomenclature used for this file is based on dictionaries (key:value pair).

Each module of the tool has one of these configuration files, and the controller file controls its execution. This allows for the configuration of some of the functions that are explicitly offered by the tool's sub-modules. It is possible to choose the algorithms that will run during the experimentation for each module, as well as the input and output data for the execution of these algorithms, and the number of rows in the dataset, which is of great help when simplifying the analysis of large datasets and reduce experimentation times. Once this module is configured, TSxtent can automatically execute each step configured in the file.

4.2. Data Pre-Processing Module

In this module, the researcher begins to prepare the data that will be used in the experimentation. The module is paramount for temporal sensor data (especially energy data) since the data are collected from heterogeneous data sources (sensors, databases, etc.) [38,39].

Generally, the raw data need to be pre-processed to feed the different learning algorithms, so the data require a cleaning and preparation process to improve the algorithms' performance. This module comprises different submodules, which are described below.

4.2.1. Partitioning

Data partition techniques transform the current datasets to generate new ones. These techniques are essential in time series on energy data because they generate new, hidden knowledge that can improve the results offered by learning algorithms.

Numerous works include this sort of technique [40–42]. The data-partitioning module takes care of partitioning, grouping, and extracting the necessary information so the user can experiment, using a given time interval. It helps to study the dataset defined in a specific time interval, which can be far smaller than the original dataset.

Another important feature is the possibility of grouping the dataset fields at different hierarchical levels, generating a new dataset for analysis in the experiment. For example, it is possible to group according to specific dataset field and obtain several subsets of data to analyse, generating new, hidden knowledge.

For this purpose, dynamic tree generation was implemented to create a new dataset following the parent–child–grandchild hierarchy. It should be noted that, deeper into the tree, the execution becomes less efficient. This permits the generation of new, specific datasets and focuses the experimentation on data with a more well-defined feature set. Finally, the module offers the possibility of defining where the generated dataset is stored and where the executions of the experiments are performed. The structure of this tree can be seen in Figure 2.

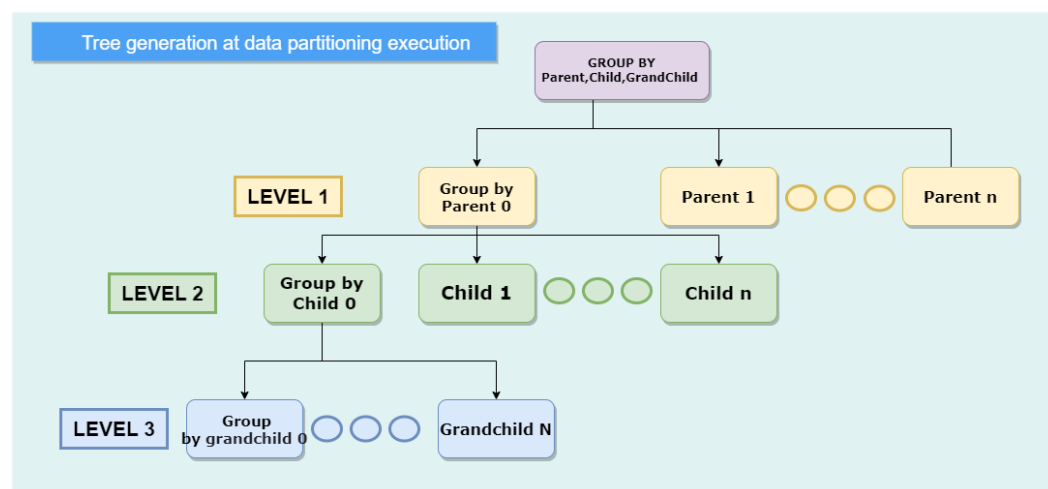


Figure 2. Example of tree generation.

4.2.2. Missing Values

Dealing with incomplete data is a common challenge when working with datasets, especially regarding energy efficiency [43,44]. Incomplete data can arise for various reasons, including faulty sensors, lost data during transmission, or missing data due to human error. This type of data can cause malfunctions in machine learning prediction algorithms, so it is essential to apply pre-processing techniques to deal with these errors and achieve better results. To address this issue, a submodule was created within the data processing module and the focus was placed on removing missing values from the dataset.

This submodule offers the option of treating the missing values by configuring the file and the possibility of indicating the path in which the dataset is located in order to apply the missing value techniques. The currently implemented algorithms are the *interpolation algorithm*, which performs an interpolation between the different rows of the dataset, and another, more aggressive one, which eliminates the rows with missing values.

The *interpolation algorithm* [45] allows for the generation of new data and enhances the quality of the original dataset. It involves estimating the value of a field at intermediate

points between known datapoints. The *row elimination algorithm* is more aggressive and, unlike the previous one, eliminates the row. This algorithm is necessary when the number of missing values within a row of the dataset is too high.

Once either algorithm is executed, the system generates a graphical resource to verify that the missing values have been correctly removed, showing the number of records in each field. The resources are stored in the log module so that the results are saved and can be consulted at any time. It should be highlighted that this process transforms the original dataset, thus improving the quality of the dataset and, therefore, the performance of the learning algorithms.

4.2.3. Outliers

Outliers are another common problem to be faced when solving data science problems [46–48]. These types of values are all too frequent in energy problems because the devices that are responsible for collecting the information and sending it to our source of knowledge are prone to manipulation by external sources from the environment in which they are installed.

There are cases in which the sensors are exposed to temperatures that do not correspond to the actual temperature of the environment that exists at that moment, either due to exposure to a foreign heat source or to a cold source. These actions lead to a series of outliers that are outside of the typical pattern of measurements.

To solve this type of problem, this preprocessing submodule was created, which focuses on the elimination of these outliers from the dataset with which the researcher is going to work. This includes the possibility of selecting the fields to be analyzed in the configuration file, as well as indicating the path on which the dataset to which we are going to apply the outlier preprocessing techniques is located.

In this first layer of our architecture, we included the “z score mean” algorithm. This algorithm detects the values that are within a range defined by us as missing values, and performs an average over its nearest neighbours to modify that value and obtain a prediction of what value should exist at that moment. The system generates a graphical resource to check that the outliers have been correctly eliminated by means of a simple box plot. The resources are stored in the loggers module, so the results are stored and can be queried later in a more effective way.

This process also transforms our dataset, i.e., the application of this algorithm modifies the dataset with which the research continues to more effectively apply our learning algorithms.

4.2.4. Feature Selection

Feature selection is a preprocessing technique that allows for us to obtain valuable information to learn what type of features are most relevant and how the different variables that make up our dataset relate to each other and the problem we are analysing. Like the previous ones, it is one of the most widely used techniques in data processing.

In problems such as those posed by energy efficiency in buildings [49,50], this is advantageous because, in most cases, there are a large number of variables, for which, depending on the problem we are trying to solve, we will need a series of variables or others.

The creation of this module manages to show the relationship between the different variables of the dataset on which we are working. For this to work correctly, it is advisable there are no missing values.

In our tool, it is possible to choose the fields that are to be analysed, as well as the route from which the data to be analysed are obtained. One of the algorithms that is implemented is a correlation algorithm, which generates an image-type resource, with a heat map displaying the correlation between the different selected fields. This algorithm shows the degree, between 0 and 1, of correspondence between one variable and another, with 0 being the lowest degree and 1 the highest degree of correspondence. Another of the implemented algorithms is a proprietary algorithm called FSMeasure that manages to

obtain the measures of mean, standard deviation, entropy, chi-square and dispersion of the data of the different variables of the dataset with which we are working.

This type of analysis helps us in the selection of the input fields of our time series, which we are going to insert into our learning algorithms. The resources generated by this type of algorithm are stored in the loggers module so that they can be consulted later in a convenient way.

4.3. Machine Learning Module

The machine learning paradigm has been widely applied to solve energy efficiency problems [51–54]. This type of paradigm allows for the input of a series of data and the execution of processes that result in an output. Depending on the problem being solved, this output can predict a result with a range of success or classify data into a specific category.

In the case of energy problems, most of these problems are regression prediction problems. In this library, which is based on time series analysis, the input dataset obtained from preprocessing and exploratory analysis is used as the input. A series of algorithms are run, which produce predictions about the building's consumption as output. This helps to optimize consumption and plan a series of energy-saving strategies for the end user.

This machine learning module contains several sub-modules, which include algorithms based on solving time series problems to make predictions using our datasets. As a time-series-focused tool, these algorithms can all solve regression problems.

These algorithms generate a series of resources that help the researcher to draw conclusions about the model generated in our experimentation. The models generated in our experimentation are stored in the logger module. Each of the algorithms integrated in the tool is explained below.

4.3.1. Random Forest Regression

Random Forest [55] averages many models with noise and impartial, reducing the variance. Trees are ideal candidates for bagging, as they can record complex interaction structures in the data.

For the prediction of a new element, a tree leaf is delved through the tree leaves. Then, it is assigned the label at the final node. This process is iteratively railed through all the trees to be assembled in the run and the node that obtains the highest coefficient is reported as the prediction.

The advantages of random forests is that they are highly adaptable to large amounts of data, run efficiently and handle a large number of features, and many energy-efficiency-related jobs are solved using this type of algorithm [56–58].

For these reasons, we implemented this submodule, which runs the random forest regression algorithm included in the sklearn library [59]. The main configuration allowed in this submodule includes the number of Kfolds to be performed within the algorithm, the measurement criteria for each selected Kfold, and the inputs and outputs for the model.

In addition, other parameters are required to further fine-tune the execution, such as the depth of the trees, the estimation and the minimum number of children. The submodule, as in the previous one, displays the scores of the different runs, as well as the mean and standard deviation of these results.

The resources generate a graph showing the tree generated with the various calculations performed by the algorithm, and a report with their respective scores, as well as the mean and standard deviation of these results. This includes the model generated by the algorithm. These resources can also be visualized in the logger module, and the models generated by the execution of these algorithms can be reused.

4.3.2. Decision Tree Regression

A decision tree [27] is a prediction model that has been used in countless fields, especially in the energy field [34,60,61], and is one of the most widely used algorithms in this field.

Given a set of data, a series of logical diagrams are created, similar to rule-based prediction systems, which represent and categorise a series of conditions that occur in succession to solve a problem.

Due to its widespread use in the energy field, the decision was made to implement this submodule, which executes this regression algorithm with the decision trees included in the sklearn library [59]. The main configuration submodule includes the number of Kfolds to be performed in the module, the measurement for each chosen Kfold, and the inputs and outputs for the model.

In addition, other parameters are included to fine-tune further the execution, such as the depth of the generated decision trees, the estimation, and the minimum number of children. In this submodule, as in the previous ones explained above, Random Forest Regression and XGBoost show the scores of the different runs and the mean and standard deviation of these results.

The generated resources consist of a graph showing the tree generated with the different calculations executed by the algorithm and a report with the scores generated in each Kfold, as well as the mean and the standard deviation of these results. These results are recorded in our logger's module and allow for the researcher to reuse them.

4.3.3. XGBoost

XgBoost [26] is quite efficient when running large amounts of data and is flexible regardless of the nature of the data. As a tree running in parallel, it can solve many problems quickly and accurately. Using this type of decision tree in our architecture is essential, as it has been shown in numerous works to work well [45,62–64].

XgBoost is one of the most widely used frameworks in energy efficiency for solving time series regression problems. This submodule runs an XgBoost regressor algorithm. The configuration mainly allowed for covers the number of Kfolds to be performed within the algorithm run, the "target" measure for each chosen Kfold, and the inputs and outputs for the model. The scores of the different runs are shown, as well as the mean and standard deviation of these results.

Once all these calculations have been performed, the generated resources are stored in the logger's module. The generated resources include a graph with the different "scores" of the chosen splits, reports with these "scores", the mean and standard deviation of these results, the parameters used in a run, and the model generated by the algorithm run.

The resulting models stored can be used at any time by the researcher to make predictions with other data, and even to analyse the different results between the runs performed in the experiment.

4.4. Deep Learning

The deep learning paradigm represented a different jump forward in the energy prediction [65,66]. Neural networks underwent a significant evolution, and can be used to solve the different problems that arise during optimisation in the field of energy consumption optimisation. Algorithms that simulate the neural behaviour of the brain, such as convolution networks or recursive neural networks, have been shown to give much better results than decision trees in problems dealing with time series.

For these reasons, it was proposed to create a module involving some of the most efficient neural networks for working with time series [67–69]. Initially, a multilayer perceptron network, a convolutional network and an extended short-term network were chosen. This module was configured through the configuration file, where the algorithms were indicated. Each of the implemented submodules will be explained in the following sections.

4.4.1. Multi-Layer Perceptron

A multilayer perceptron network [70] consists of an input layer, a hidden layer, and an output layer [71]. The most straightforward neural network uses a supervised learning technique for training and a non-linear activation function. This has been used in several

papers in the field of energy efficiency, but there is little work. Therefore, it is interesting to continue working with this type of network and to have our architecture ready to continue studying the behaviour of this algorithm in different problems [72–74].

For this reason, this algorithm was chosen to create a submodule within Deep Learning. A multilayer perceptron (MLP) neural network for univariate time series forecasting models was included. The configuration file allows for us to configure both the inputs and outputs of the model.

An exciting aspect of configuration is the n-steps parameter. This allows for the input/output sequence to be split into multiple patterns, which, being univariate, result in the prediction of an output. In addition, the configuration of the hidden layers and the number of epochs that the neural network executes are included. This neural network is composed of two density layers.

Another interesting aspect is the grouping of the data with the batch-size parameter. We can group these into more or fewer data. This sub-module generates a report with the obtained results and stores the model in the logger's module so that the researcher can make any prediction later and include the configuration of the hidden layers.

4.4.2. Convolutional Neuronal Network

A convolutional neural network [75] consists of an input layer, n-hidden layers and an output layer. In any feed-forward neural network, the intermediate layers are called hidden because the activation function and the final convolution mask their inputs and outputs. In a convolutional neural network, the hidden layers include layers that perform convolutions. Their activation function is usually ReLU. The convolution operation generates a feature map, which contributes to the next layer's input. This is followed by other layers, such as grouping, fully connected, and normalisation layers.

This was rarely used in the energy field [76–78] because it is a network that works very well with images. This is why it has been little tested in the energy domain, which tends to use time series. Thus, by carrying out a series of transformations, it can be adapted for use in time series, and we can study this type of network in problems in different domains. This neural network comprises a one-dimensional convolutional layer, another max-pooling subsampling layer, another flattening layer to reduce the matrix into a flat matrix, and two dense layers.

In the submodule in which we have included the Convolutional Network (CNN) [71], we can configure where the inputs and outputs of our model will be stored in the file. As in the previous submodule, n-steps are configured. We can configure the number of features considered for the model, the number of hidden layers, and the number of other parameters highlighted in this algorithm.

This submodule generates a report with the obtained results, and stores the module so that the researcher can make any prediction at a later date.

4.4.3. Long Short Term Memory

Short-term memory (LSTM) [30] is an artificial neural network in artificial intelligence and deep learning. Unlike standard neural networks, LSTM has feedback connections. This recurrent neural network (RNN) can process images and time series. In the energy field, it is currently one of the most widely used neural networks to date [39,79–82].

The submodule is configured in the file and allows for us to see both the inputs and outputs of the model. As in the previous modules, we can configure the n-steps. We can select a large number of features that are considered when generating the model and the number of hidden layers.

This module generates a report with the obtained results and stores the module so that the researcher can later make any type of prediction that can be consulted in the logger module. This neural network comprises an lstm layer and a dense layer.

4.5. Loggers

It is worth highlighting the module in which all the loggers of our tool are stored due to the importance given to the interpretability of data in machine learning problems. This type of solution, which is far from the realm of experts, can be understood and comprehended with little data science knowledge. This is one of the motivations for integrating MLflow into TSxtend.

MLFlow is a tool developed by [32], in which all the results produced during the execution of an experiment are recorded. With this tool, it is possible to keep track of all executions, as well as the creation of the different processes that are used to run our different modules. Another important reason that it has been integrated is that it shares the same language in its implementation. MLflow developed the same programming language that we developed in our tool, such as Python, so this integration is easier. As an open-source tool used by data engineers, we can follow the different executions of our work. At the same time, we can take advantage of the knowledge that is generated to adapt the tool to our needs.

Finally, this allows for us to visualize all the files generated by our tool when it finishes executing the experimentation we dictated in the configuration file. This way, all the logger generated by our tool is stored graphically and through a user-friendly interface. This will help us to better understand our results and to make decisions by planning a strategy to develop new experiments.

For installation and start-up, TSxtend automatically installs MLflow on the server on which the software is deployed, with the help of the Python package manager called Conda [83]. Once our library is executed, a web browser will be launched, on which each of the experiments carried out by the user can be accessed.

5. Use Case: Energy Data Analysis

In this section, a use case is presented to demonstrate the feasibility of the proposed tool. For this purpose, experimentation with real data is carried out. The workflow defined for the use case is depicted in Figure 3.

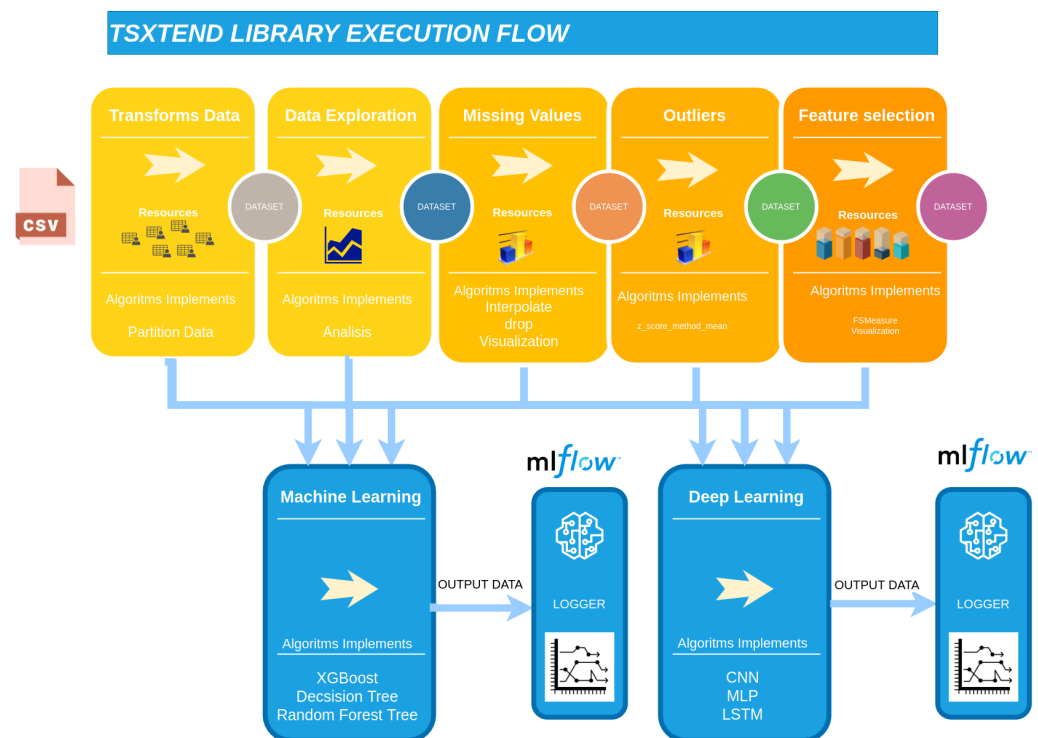


Figure 3. TSxtend execution flow.

5.1. Methodology

As previously outlined in Section 4, the proposed workflow includes stages for data mining, prediction using Machine Learning and deep learning techniques, and the storage and consultation of results on the server launched by the tool, which has been integrated with MLflow.

The data mining component enables the transformation of data through the execution of algorithms, partitioning it into smaller datasets for more efficient experimentation. The following stage, as shown in Figure 3, is data exploration which allows for the analysis of data through various visual representations such as tables, text files, and graphs.

Another stage of the methodology is the removal of missing values, followed by the detection and removal of values that fall outside the range of the data sampling and can be considered noise. The final stage of the data mining phase is feature selection, which allows for the identification of correlations between variables and the determination of the most important features within the dataset. These steps can be performed independently and in any order, allowing for any stage to be executed and prediction algorithms to be used at any point.

The final stages of the methodology allow for the implementation of machine learning and deep learning algorithms to make predictions on the specified output variable (as seen in the Appendices A.6 and A.7). The following sections demonstrate the application of this methodology within a specific use case in the energy domain.

5.2. Dataset Description

To conduct the experimentation, the dataset was obtained from the Kaggle Predictive Features Competition called ASHRAE - Great Energy Predictor III, which focuses on energy consumption analysis. To ease the experimentation, the original dataset was transformed and consolidated into a single training file. Additionally, the fields that were deemed most relevant for the analysis were selected. Table 2 showcases the selected fields. The dataset includes three years of hourly readings from counters of over a thousand buildings in different parts of the world, with 1,048,000 records. The data were taken from the contest posed on the Kaggle platform. For more details regarding the dataset, refer to [12].

Table 2. Dataset description.

Name Field	Descriptions
building_id	Foreign key for the building metadata.
meter	The meter ID code.
timestamp	When the measurement was taken
meter_reading	The target variable. Energy consumption in kWh (or equivalent).
site_id	Foreign key for the weather files.
air_temperature	Degrees Celsius
cloud_coverage	Portion of the sky covered in clouds, in oktas
dew_temperature	Degrees Celsius
precip_depth_1_hr	Millimeters
sea_level_pressure	Millibar/hectopascals
wind_direction	Compass direction (0–360)
wind_speed	Meters per second

5.3. Data Partitioning

As explained in Section 4.2.1, this phase aims to show how the tool works with energy data when pooling data from an original dataset. In this way, the disparate energy data are further disaggregated in the first capture phase. In the following sections, the use of the module with our tool is demonstrated.

In our use case, the data were grouped by *site_id* and *meter* to associate electricity consumption with specific areas. This process allows for the generation of new, hidden insights from the dataset. A specific period was selected, in this case, all of 2016, so that the algorithm could partition the energy data accordingly. The grouping and partitioning of energy data was carried out in an attempt to extract less biased information. An example of this configuration can be found in Appendix A.1. Once these parameters were selected, our tool generated more than thirty files containing energy consumption data for all the buildings that comprise ASHRAE. A diagram illustrating this process can be found in Figure 4.

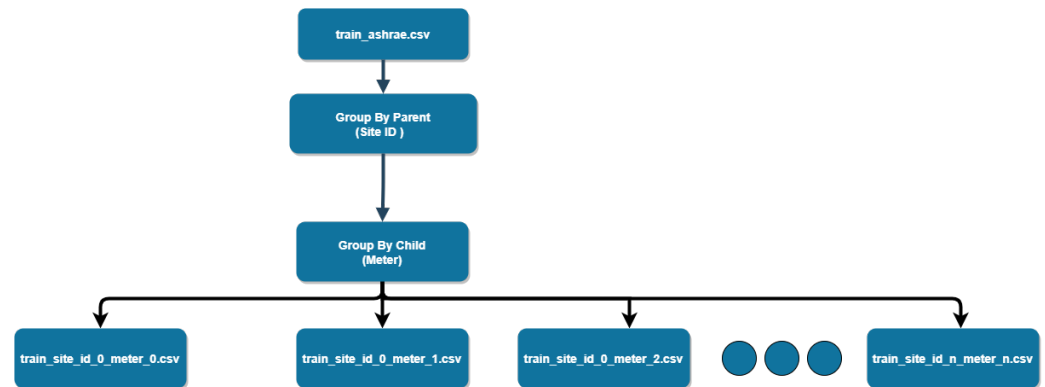


Figure 4. Generating CSVs through experimentation.

5.4. Data Exploration

The researcher typically examines the data using various statistical and visual techniques during the exploratory data analysis. TSxtend allows for this process to be defined through the *configuration file*. Appendix A.2 showcases an example of the *configuration file*. For the current use-case, we selected the fields *meter reading*, *air temperature*, *dew point temperature*, *precipitation depth of one h*, *pressure at sea level*, and *wind speed* because these variables are the most relevant for the use-case (energy consumption analysis). The exploratory data analysis generates a series of charts that can help to understand the data distribution, identify any outliers or anomalies, and identify trends and patterns.

In Figures 5 and 6, it can also be observed that *wind speed* is the variable that has the least correlation with the other ones. Finally, Figure 7 depicts a strong inverse correlation between the variables *air temperature* and *dew point temperature* and the buildings' energy consumption.

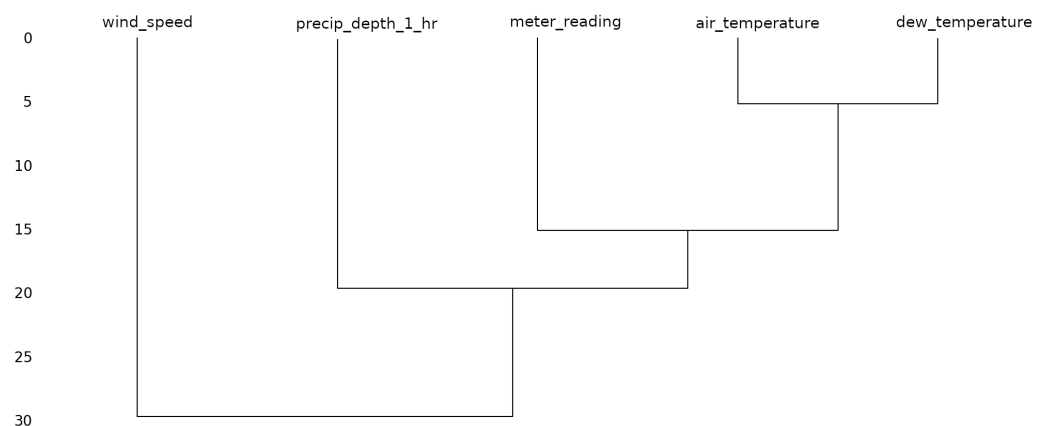


Figure 5. Dendrogram generated using selected variables.

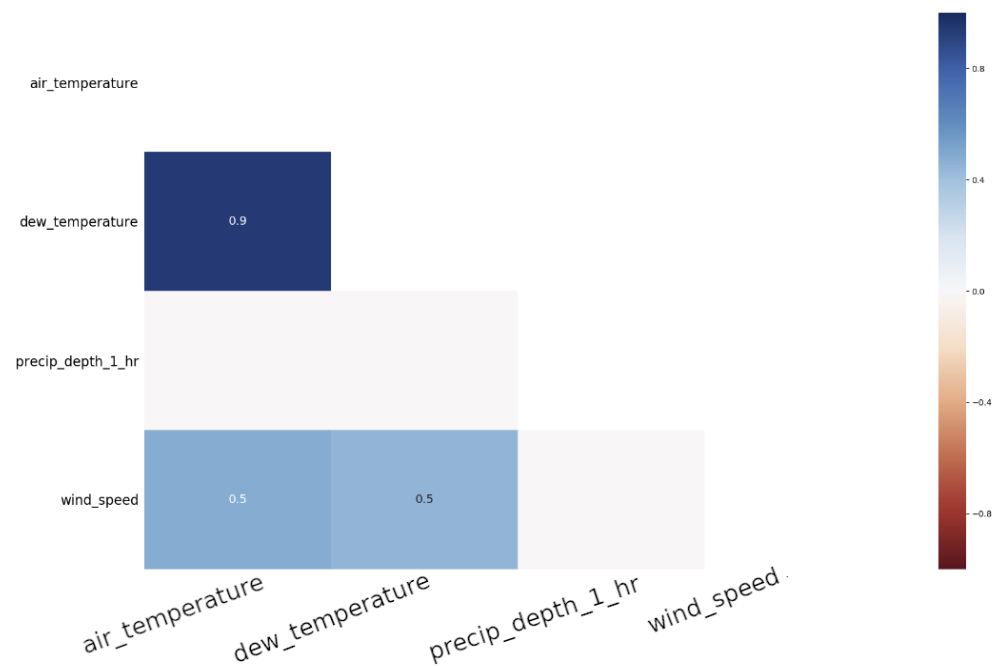


Figure 6. Correlation for selected variables.

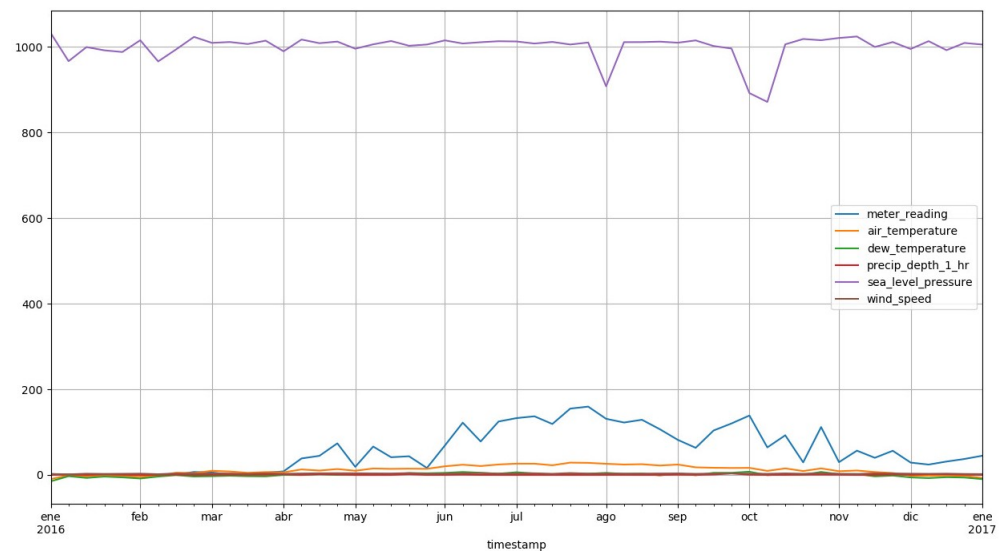


Figure 7. Evolution of time series over time for selected variables.

5.5. Removing Missing Values

During this experimentation phase, missing values were removed from the dataset. The removal of missing values is crucial to obtaining accurate, reliable, and valid results from any data analysis. The preceding section demonstrated the process of visualizing the data using TSxtend. From the charts, it can be concluded that missing values are present in all fields except for the *meter-reading field* variable.

Once the fields with missing values are detected, algorithms are applied to remove them. Only the *interpolation* and *omission* algorithms are implemented. For a more detailed explanation of the above algorithms, our repository can be consulted [37].

In the use-case developed in this paper, the *interpolation* algorithm based on its nearest neighbours was used. Missing values for fields such as *wind speed*, *precip depth*, *dew temperature* and *air temperature* were removed. It is important to note that this type of algorithm transforms the original dataset. The final results are shown in Figure 8, verifying that all the missing values were removed. Appendix A.3 illustrates how to remove missing values through the *alg_missing* parameter.

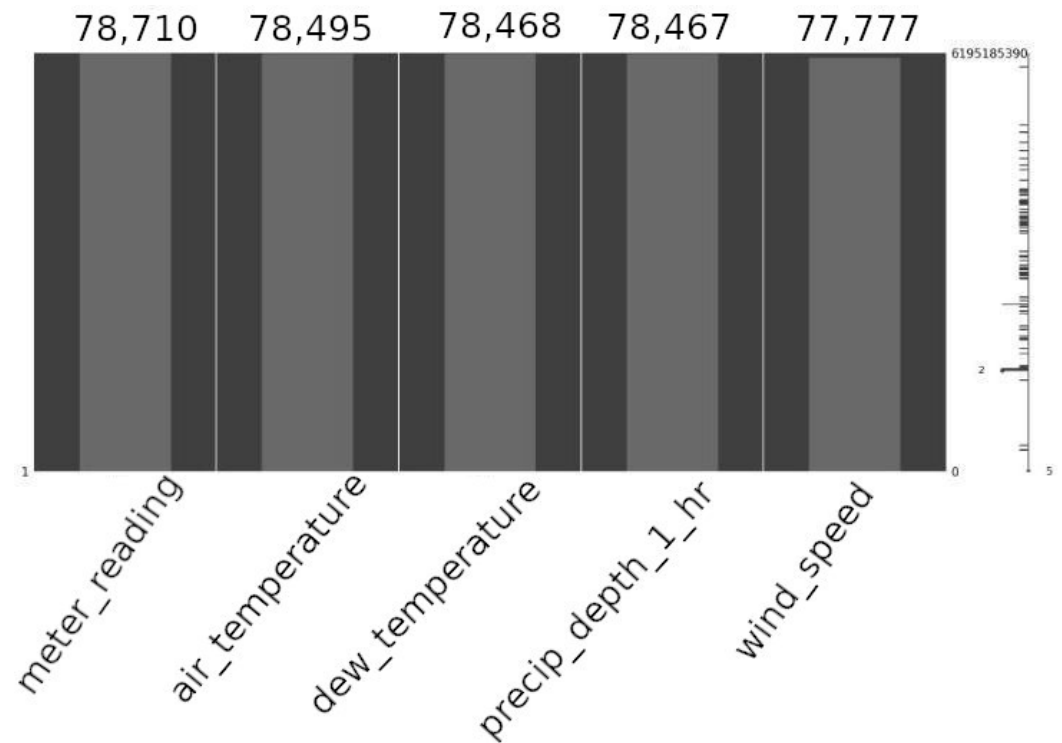


Figure 8. Removing missing values.

5.6. Noise Reduction

The next step comprises noise reduction. From the previous exploratory data analysis, we know there are outliers in our dataset, which is reasonable when dealing with temporal sensor data about energy. To achieve this, the *z-score-mean* algorithm will be applied (see Appendix A.4). This method identifies and removes outliers from a dataset by calculating the *z-score* for each datapoint and comparing it to a threshold. If the *z-score* of a datapoint is greater than the threshold, it is considered an outlier and removed from the dataset.

To use the *z-score-mean* algorithm, it is necessary to specify the threshold value, which specifies how many standard deviations a datapoint needs to be from the mean to be considered an outlier. A common threshold value is 3, meaning that a datapoint needs more than 3 standard deviations from the mean to be considered an outlier. However, the appropriate threshold value will depend on the specific characteristics of the data and the goals of the analysis. Figure 9 shows all the outliers for the use-case. It is important to note that the variables with the most variability in their values are *dew temperature*, *air temperature*, and *wind direction*, which are also the most influential variables in the energy dataset.

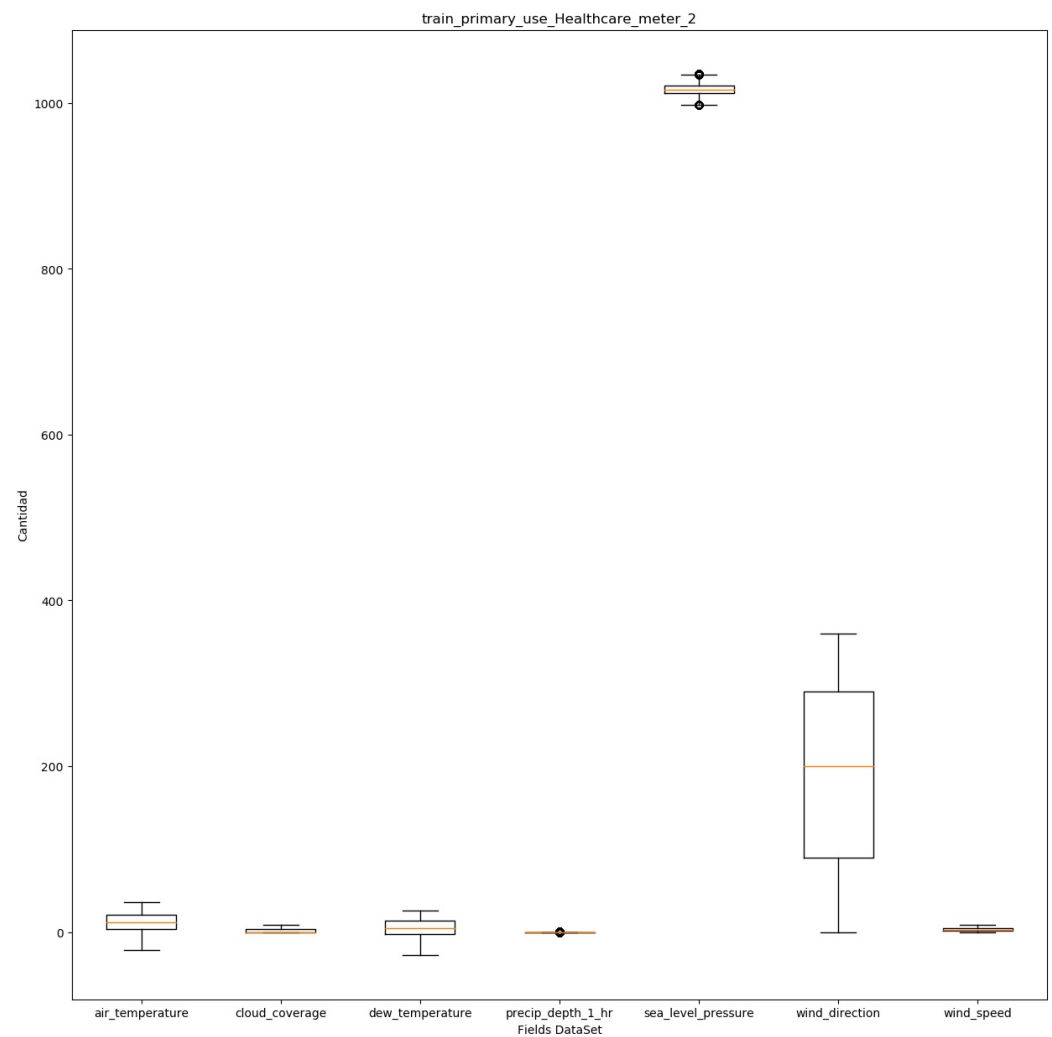


Figure 9. Outliers detection.

5.7. Feature Extraction

This process, one of the most critical in preprocessing, helps us to obtain the most relevant variables from our dataset. Our algorithms are shown in [37].

In this process, we ran a variable correlation algorithm that shows the relationship between our variables. Figures 10 and 11 showcases the correlation between variables *dew temperature* and *air temperature*. In this case, the variables are highly correlated, with a value of 0.9. Wind direction and wind speed have a correlation, but this is not very strong; the value we obtained was 0.5.

By executing the algorithm that measures the characteristics of the different variables in our dataset, the reliability of the different variables can be determined. In our use-case, it can be seen that the most reliable variables are *dew temperature*, *air temperature* and *wind speed*. Therefore, it can be verified that these are the most crucial variables for predicting the energy consumption of our buildings.

In this manner, the effectiveness of the algorithms was verified with the energy dataset and assisted us, together with the exploratory analysis, in conducting a comprehensive study of the different variables that comprise our experimentation.

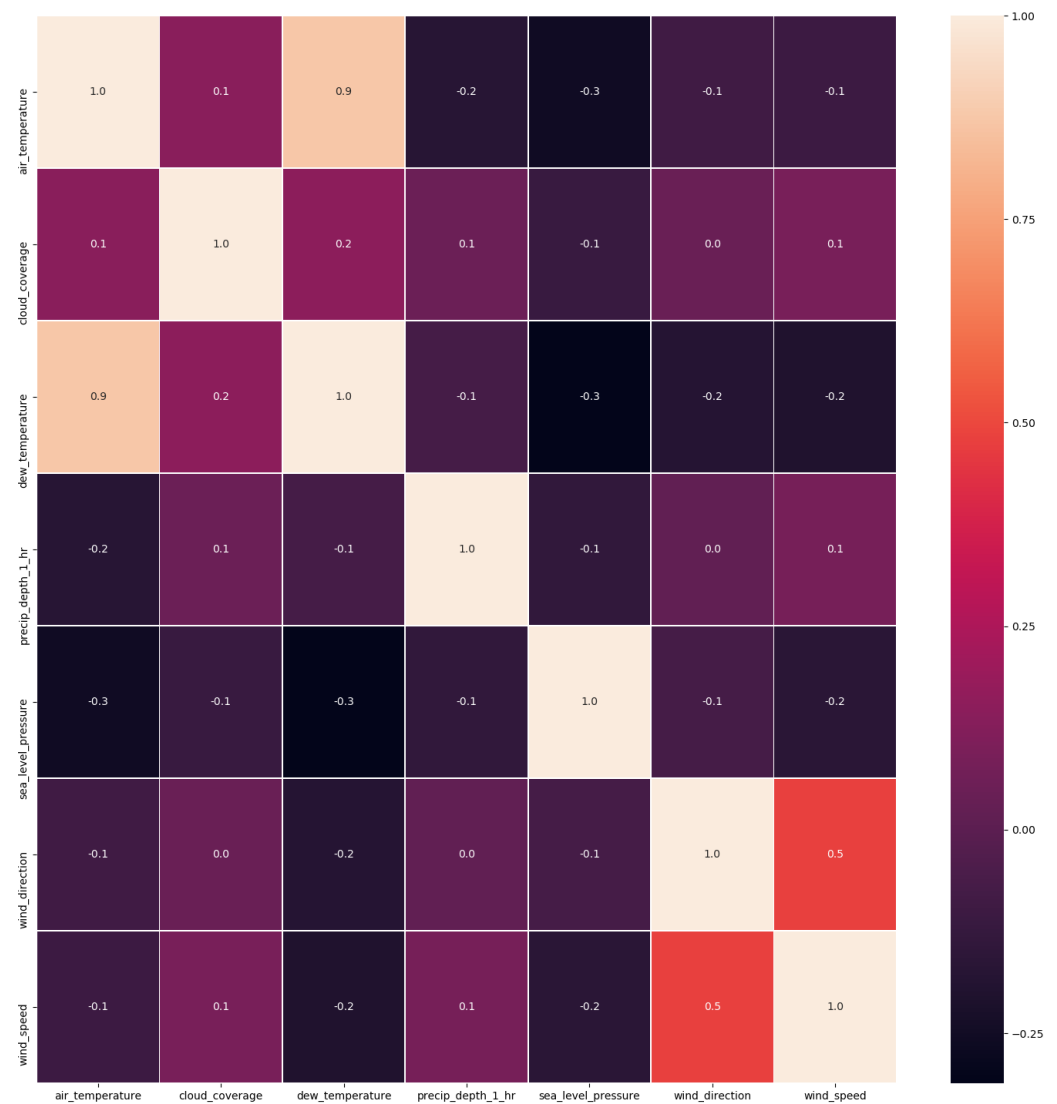


Figure 10. Correlation between selected variables.

	mean	Std.Dev	Var	entropy	chi	dispersion
meter_reading	226.25	376.54	141,788.92	6.44	5.67×10^8	89.44
air_temperature	22.86	6.01	36.11	3.76	1.43×10^6	1.12
dew_temperature	16.85	6.48	42.01	3.64	2.25×10^6	1.81
precip_depth_1_hr	1.38	12.97	168.24	0.52	1.10×10^8	2.29
sea_level_pressure	1,017.95	4.03	16.24	5.15	1.44×10^4	1.03
wind_speed	3.37	2.15	4.64	2.59	1.24×10^6	1.42

Figure 11. Statistical measures for selected variables.

5.8. Employing Machine Learning Techniques to Generate Predictions

One of the key points towards which energy-based information processing is directed is the execution of algorithms for predicting energy consumption. For this reason, the decision was made to implement these types of algorithms in TSxtend. The tool can make a prediction by running classical machine learning algorithms to see how previously

preprocessed data behave. In this use-case, the execution of a single algorithm, XGBoost, is demonstrated. The rest of the implementations can be viewed in [37].

This algorithm uses a K-fold number of models with XGBOOST Regressor algorithms. It makes predictions and stores each result in the array called scores to obtain the mean squared error of the energy consumption prediction. An important variable in the configuration is *n_splits* (see Appendix A.6), because it indicates the number of times the received dataset will be divided. Finally, metrics such as *mean squared error*, *standard deviation* and *number of partitions* of our data are computed.

As discussed in Section 4.5, the outputs will be stored in the logger module integrated by MLflow. These results are stored as graphs, plain text or files that can be consulted by users at any time, helping the user to interpret the experiment.

Tables 3 and 4 show the scores and statistical measures, respectively, for the Xgboost algorithm, while Figure 12 depicts the performance of the Xgboost algorithm. In our use-case, the three most crucial variables mentioned in previous sections, *dew temperature*, *air temperature*, and *wind speed*, were selected. Based on the selected variables that were used as input, the Xgboost algorithm was executed, and ten splits were made to generate the mean error when making the prediction.

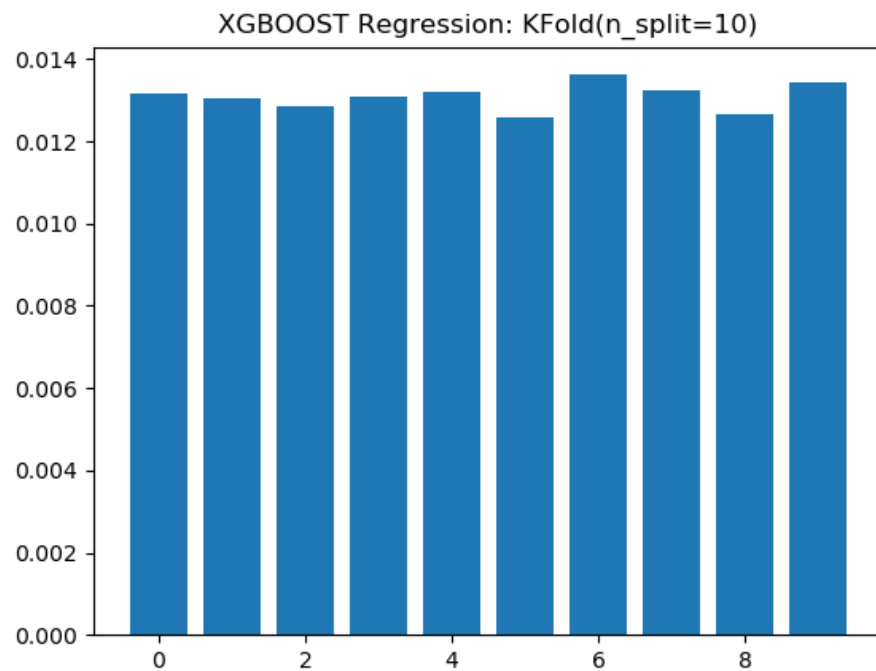


Figure 12. Performance of XGBoost algorithm.

Table 3. Scores for Xgboost algorithm.

Display Scores	Scores
0	0.0132
1	0.0130
2	0.0129
3	0.0131
4	0.0132
5	0.0126
6	0.0136
7	0.0132
8	0.0126
9	0.0134

Table 4. Mean, standard deviation, and number for the Xgboost algorithm.

MAE	STD	Display Scores
0.0131	0.001	10

The result shows us a squared error of 0.0131, which is quite low, with a standard deviation of 0.001, indicating that the execution with the passed input data is correct. This suggests that the input variables are well-correlated with the target variable (energy consumption), and that the algorithm is effectively learning the relationship between them. A test could be run to verify the results, but the goal of this work was to run the tool and see that it works correctly.

5.9. Using Deep Learning Algorithms to Make Predictions

As we mentioned in the previous module, the trend in the study of energy data is predicting consumption using machine learning algorithms.

With this, the tool can perform a prediction executed on classical neural network algorithms to see how the previously preprocessed data behave. In this use-case, the execution of a single algorithm, the long-term memory (LSTM) algorithm, is demonstrated. The rest of the implementations can be seen in [37].

Next, the data were sequenced, and the intervals were divided according to the n_steps (see Appendix A.7). Once the data were obtained, we created a model using, in this case, an LSTM network. The inputs were the same as those used in the XGboost algorithm executed in the previous section. The variables *dew temperature*, *air temperature*, and *wind speed* were used to predict the building's energy consumption.

Finally, we obtained the metrics we measured in our model (rmse, mae, mse). The LSTM model is stored in the system, and the evolution of the model over the execution of different periods is shown a graph. The results of the experimentation are presented in Table 5 and Figure 13.

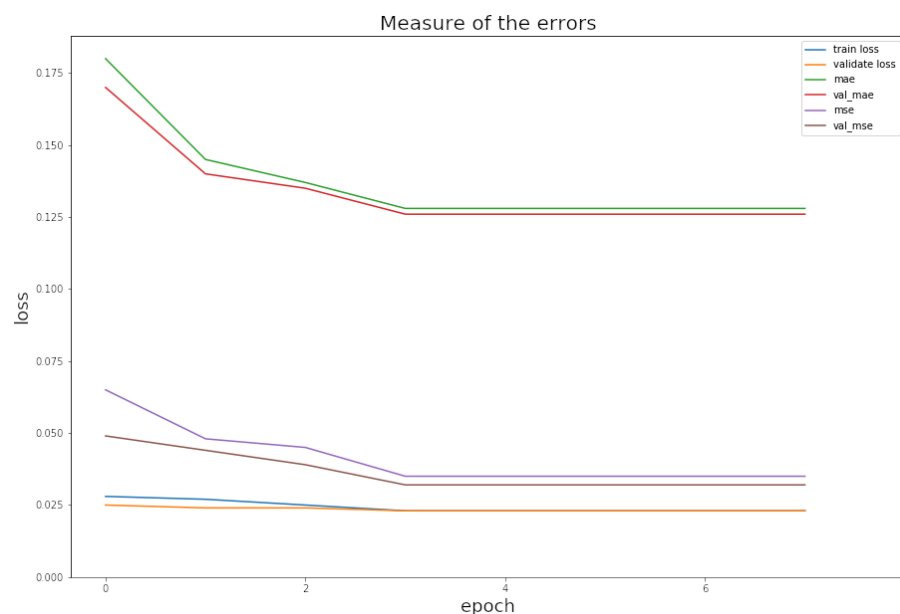
**Figure 13.** Graphical representation of the results of LSTM model execution.

Table 5. Evaluating the performance of an LSTM model in our case study.

Measure Error	Display Scores
rmse	0.1292
mse	0.1055
mae	0.0167

These results can be used by users, as mentioned in Section 4.5, the outputs generated by the different executions within our system will be stored in the loggers module, where, with the help of MLflow, they can be consulted at any time by the user. This will allow for them to perform an analysis of their experimentation and create reports, and also help when making a decision about the experimentation.

In this case, the RMSE, MSE, and MAE values were all relatively low, which suggests that the model is making relatively accurate predictions. The RMSE value of 0.1292 indicates that, on average, the model's predictions are off by about 0.1292 units. The MSE value of 0.1055 and the MAE value of 0.0167 are both lower than the RMSE value, which further supports the conclusion that the model is making accurate predictions.

6. Conclusions and Future Work

This paper introduces TSxtend, a new software tool designed to assist non-programming users in analysing temporal sensor data. TSxtend simplifies the process of transforming, cleaning, and analysing temporal sensor data through the use of a declarative language for defining and executing workflows. This tool aims to empower users to take control of their data, enabling them to make timely and well-informed decisions during the research process. With TSxtend, non-programming users can quickly analyse their data, allowing for them to focus on developing their research and understanding their results rather than struggling with programming.

The paper's main contribution is the development of the tool itself and the features that make it unique. The tool was implemented using a modular architecture, which allows for the standardisation of different stages of experimentation. TSxtend allows for data transformation, cleaning, and imputation, as well as the execution of prediction algorithms and the visualisation of results at different workflow stages. The tool also allows for the use of different techniques for time series analysis and makes experimentation more accessible for non-programmers. Finally, can obtain the first set of results for the analysed dataset in a fast and agile way. In this sense, this helps end-users to establish timely and proper strategies during the decision-making process. The main difference between the proposed tool and the others that were implemented is the possibility of defining a workflow quickly and readily, without prior programming knowledge.

TSxtend was implemented using an easy-to-use approach. This led to a considerable improvement in the user experience when executing algorithms and obtaining initial results, which could be used to analyse the data. Furthermore, the modular architecture of the tool enables the incorporation of additional techniques. The results obtained using the ASHRAE Great Energy Predictor dataset demonstrate the effectiveness of TSxtend in analysing energy data. This tool is a valuable addition to the field of time series analysis and can facilitate the work of researchers and practitioners in various domains.

This tool constitutes a starting point, opening up the possibility for future works. Next, we will implement and integrate more efficient algorithms to provide end-users with more powerful techniques. To make the user experience friendlier, we will develop a more intuitive user interface. Finally, it would be interesting to consider the problems with a multi-variable, as well as developing a complete tool in a distributed environment.

Author Contributions: R.M.-J.: Conceptualization, Software, Data curation, Validation, Writing Original Draft, Writing, Writing Review and Editing. K.G.-B.: Methodology, Writing Original Draft, Writing Review and Editing, Formal Analysis. J.G.-R.: Conceptualization, Writing Review and Editing, Supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by FEDER/Junta de Andalucía (DEEPSIM project, A-TIC-244-UGR20), Spanish Ministry of Science (SINERGY project, PID2021-125537NA-I00) and the NextGenerationEU funds (IA4TES project, MIA.2021.M04.0008).

Data Availability Statement: Data is publicly available at <https://www.kaggle.com/competitions/ashrae-energy-prediction/data> (accessed on 23 December 2022). See more details in [12].

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

Nomenclature and Formula

REST API
user-friendly
pipeline
end-user
roadmap
serializacion
dictionary
measures of mean
standard deviation
entropy
chi-square
heat map
bagging
fine-tune
RMSE
MSE
MAE

Description

Representational State Transfer (REST) *Application Programming Interface (API)*.
Describe applications, websites, and other digital products that are easy to use and understand.
Series of steps or stages that a set of data go through in order to be processed, analyzed, or otherwise transformed
The people who use the software that was developed by the programmers.
Plan or strategy that outlines the steps to be taken to achieve a specific research goal or set of goals.
Process of converting an object or data structure into a format that can be stored or transmitted over a network.
Data structure that is used to store a collection of key-value pairs.
Describe the average value of a dataset.
Measure of the spread or dispersion of a dataset.
Quantity that represents the amount of disorder or randomness in a system.
Statistical test used to determine whether there is a significant difference between an observed frequency distribution and a theoretical distribution.
Graphical representation of data, where individual values are represented as colors.
Technique used in ensemble learning to improve the stability and accuracy of predictive models.
Technique used in deep learning to adjust the parameters of a pre-trained model on a new dataset.
The square root of the mean of the squared differences between the predicted and actual values.
The mean of the absolute differences between the predicted and actual values.
The mean of the absolute differences between the predicted and actual values.

Appendix A

Appendix A.1. Partition Data

Listing A1: Config/main.yaml.

```
1 etl:      partition-data
2 deepl:    ""
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae
```

Listing A2: Config/partition-data.yaml.

```
1 date_init: 2016-01-01 00:00:00
2 date_end:  2016-12-31 00:00:00
3 fields_include: None
4 path_data: train.csv
5 group_by_parent: site_id, meter
6 output_dir: Data/train_ashrae
```

Listing A3: header function.

```

1 def PartitionDF(date_init, date_end, path_data, n_rows, fields_include,
    group_by_parent, output_dir)
2
3 ## Parameters ##
4 date_init:[ (string) Correct Date ] . Transform in datetime date init, in
    order to get data from dataset.
5 date_end: [ (string) Correct Date ]. Transform in data timedata end, in
    order to get data from dataset.
6 path_data: [ (string) path ] Path origin dataset.
7 n_rows: [ (int) ] Number rows dataset.
8 fields_include: [ (list string) name field dataset ]. Filter by dataset
    fields.
9 group_by_parent: [ (list string) name field dataset ]. Group by dataset
    levels.
10 output_dir: [ ( string ) name directory ]. Ouput directory, to save data.

```

*Appendix A.2. Exploratory Analysis***Listing A4:** Config/main.yaml.

```

1 etl:      exploratory-analysis
2 deepl:    ""
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A5: Config/exploratory.yaml.

```

1 field_x:    timestamp
2 field_y:    site_id
3 graph:      line
4 _resample:  W
5 measures:   meter_reading,air_temperature,dew_temperature,precip_depth_1_hr
    ,sea_level_pressure,wind_speed
6 input_dir:  Data/test_icpe_v2

```

Listing A6: header function.

```

1 def Visualization(n_rows,field_x, field_y, graph, measures, _resample,
    input_dir,elements)
2
3
4 ## Parameters ##
5 n_rows: [ (int) ] Numbers rows DataSet. This params get from Config/main.
    yaml
6 elements:[ (string) name elements ] Filter by elements. This params get
    from Config/main.yaml
7 field_x: [ (string) name field] Field X graphs.Usually this timestamp.
8 field_y: [ (string) name field] Field Y graphs.
9 graph:   [ (line | missing) ] Line Graphs time series or show missing values
    graph.
10 measures: [ (list string) measures ] Determinate fields. Example: site_id,
    meter
11 resample: [ (string) W, M, Y ] Resamples Week(W), Month(M), Y(Year). Only
    show data graph.
12 input_dir: [ (string) name directory ] Input directory to get data.

```

*Appendix A.3. Elimination of Missing Values***Listing A7:** Config/main.yaml.

```

1 etl:      missing-values
2 deepl:    ""
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A8: Config/missing-values.yaml.

```

1 fields_include: None
2 input_dir: Data/train_ashrae
3 alg_missing: interpolate

```

Listing A9: header function.

```

1 def missing_values(n_rows, fields_include, input_dir, elements, alg_missing)
2
3
4 ## Parameters ##
5 n_rows: [ (int) ] Numbers rows DataSet. This params get from Config/main.
      yaml
6 elements:[ (string) name elements ] Filter by elements. This params get
      from main.yaml
7 field_include: [ (list string) name field DataSet ] Filter by DataSet
      fields.
8 input_dir: [ (string) name directory ] Input directory to get data.
9 alg_missing:  [ (string) name algorithms] Name Algorithms missing values.
      [interpolate, drop]

```

*Appendix A.4. Elimination of Noise***Listing A10:** Config/main.yaml.

```

1 etl:      outliers
2 deepl:    ""
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A11: Config/outliers.

```

1 fields_include: None
2 input_dir: Data/train_ashrae
3 alg_outliers: z-score-mean

```

Listing A12: header function.

```

1 def outliers(input_dir, n_rows, q1, q3, fields_include, alg_outliers):
2
3 ## Parameters ##
4 n_rows: [int] number of rows to be extracted, 0 extracts all. This params
      get from main.yaml
5 field_include: [ (list string) name field DataSet ] Filter by DataSet
      fields.
6 input_dir: [ (string) name directory ] Input directory to get data.
7 alg_outliers:  [ (string) name algorithms] Name Algorithms outliers. [
      z_score_method_mean]
8 q1:  [ (int) ] \% outliers remove.
9 q3:  [ (int) ] \% outliers remove.

```

*Appendix A.5. Feature Extraction***Listing A13:** Config/main.yaml.

```

1 etl:      feature-selection
2 deepl:    ""
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A14: Config/feature-selection.yaml.

```

1 fields_include: meter_reading, air_temperature, dew_temperature,
                  precip_depth_1_hr, sea_level_pressure, wind_speed
2 input_dir: Data/train_ashrae
3 alg_fs: FSMeasures

```

Listing A15: header function.

```

1 def feature_selection( n_rows,fields_include,input_dir, elements,alg_fs)
2
3 ## Parameters ##
4 n_rows: [ (int) ] Numbers rows DataSet. This params get from Config/main.
          yaml
5 elements:[ (string) name elements ] Filter by elements. This params get
          from main.yaml
6 field_include: [ (list string) name field DataSet ] Filter by DataSet
          fields.
7 input_dir: [ (string) name directory ] Input directory to get data.
8 alg_fs:     [ (string) name algorithms] Name Algorithms feature selection. [
          FSMeasure, correlation]

```

*Appendix A.6. Prediction by Running Machine Learning Algorithms***Listing A16:** Config/main.yaml.

```

1 etl:      ""
2 deepl:    ""
3 mlearn:   xgb
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A17: Config/xgb.yaml.

```

1 model_input: air_temperature, cloud_coverage, dew_temperature,
              precip_depth_1_hr, sea_level_pressure, meter_reading
2 model_output: meter_reading
3 n_splits:     5
4 objective:    reg:squarederror
5 input_dir:    Data/train_ashrae
6

```

Listing A18: header function.

```

1 def xgboost(file_analysis,artifact_uri,experiment_id, run_id, input_dir,
    n_rows,model_input,model_output,n_splits, objective )
2
3 ## Parameters ##
4
5 file_analysis: File analyse. This param is generate from main.py
6 artifact_uri: URL artifact mlflow. This param is generate from main.py
7 experiment_id: Experiment id mlflow. This params is generate from main.py.
8 run_id: Run id mlflow. This param is generate from main.py
9 input_dir: [ (string) name\_directory ] Directory get Data.
10 n_rows: [ (int) ] Numbers rows DataSet. This params get from main.yaml
11 model_input: [ (list string) fields ] Fields input for run algorithms.
12 model_output: [ (list string) fields ] Fields output for run algorithms.
13 n_splits: [ (int) ] Number trees
14 objective: [ (string) ] Params algorithms XGBRegressor

```

*Appendix A.7. Prediction by Running Deep Learning Algorithms***Listing A19:** Config/main.yaml.

```

1 etl:      ""
2 deepl:    lstm
3 mlearn:   ""
4 n_rows:   0.0
5 elements: ""
6 output_dir: Data/train_ashrae

```

Listing A20: Config/lstm.yaml.

```

1 model_input: air_temperature, cloud_coverage, dew_temperature,
    precip_depth_1_hr,sea_level_pressure, meter_reading
2 model_output: meter_reading
3 input_dir:    Data/train_ashrae
4 n_steps:      3
5 n_features:   3
6 conv_filters: 64
7 conv_kernel_size: 2
8 pool_size:    2
9 hidden_units: 50
10 epochs:      10
11 batch_size:   72
12 verbose:     1

```

Listing A21: header function.

```

1 def lstm(file_analysis,artifact_uri,experiment_id, run_id, input_dir,
    model_input,model_output,n_rows,n_steps,epochs,hidden_units,batch_size,
    verbose)
2
3 ## Parameters ##
4
5 file_analysis: File analyse. This param is generate from main.py
6 artifact_uri: URL artifact mlflow. This param is generate from main.py
7 experiment_id: Experiment id mlflow. This params is generate
    from main.py.
8 run_id: Run id mlflow. This param is generate from main.py
9 input_dir: [ (string) name\_directory ] Directory get Data.
10 n_rows: [ (int) ] Numbers rows DataSet. This params get from main.yaml
11 model_input: [ (list string) fields ] Fields input for run algorithms.
12 model_output: [ (list string) fields ] Fields output for run algorithms.
13 n_splits: [ (int) ] Number trees
14 n_steps: [ (string) ] Params Split Sequences DataSet.
15 epochs: [ (string) ] Epochs Neuronal Network.
16 hidden_units: [ (string) ] Hidden Neuronals.
17 batch_size: [ (string) ] Batch Size every DataSet.
18 verbose: [ (string) ] Verbose algorithms.

```


References

1. Hang, L.; Kim, D.H. Design and implementation of an integrated iot blockchain platform for sensing data integrity. *Sensors* **2019**, *19*, 2228. [CrossRef] [PubMed]
2. Tushar, W.; Wijerathne, N.; Li, W.T.; Yuen, C.; Poor, H.V.; Saha, T.K.; Wood, K.L. Internet of things for green building management: Disruptive innovations through low-cost sensor technology and artificial intelligence. *IEEE Signal Process. Mag.* **2018**, *35*, 100–110. [CrossRef]
3. Kiran, M.S.; Özceylan, E.; Gündüz, M.; Paksoy, T. Swarm intelligence approaches to estimate electricity energy demand in Turkey. *Knowl.-Based Syst.* **2012**, *36*, 93–103. [CrossRef]
4. Nalcaci, G.; Özmen, A.; Weber, G.W. Long-term load forecasting: Models based on MARS, ANN and LR methods. *Cent. Eur. J. Oper. Res.* **2019**, *27*, 1033–1049. [CrossRef]
5. Salgotra, R.; Gandomi, M.; Gandomi, A.H. Time series analysis and forecast of the COVID-19 pandemic in India using genetic programming. *Chaos Solitons Fractals* **2020**, *138*, 109945. [CrossRef]
6. Tandon, H.; Ranjan, P.; Chakraborty, T.; Suhag, V. Coronavirus (COVID-19): ARIMA-based Time-series Analysis to Forecast near Future and the Effect of School Reopening in India. *J. Health Manag.* **2022**, *24*, 373–388. [CrossRef]
7. Chou, J.S.; Tran, D.S. Forecasting energy consumption time series using machine learning techniques based on usage patterns of residential householders. *Energy* **2018**, *165*, 709–726. [CrossRef]
8. Ruiz, M.D.; Gómez-Romero, J.; Fernandez-Basso, C.; Martin-Bautista, M.J. Big Data Architecture for Building Energy Management Systems. *IEEE Trans. Ind. Inform.* **2022**, *18*, 5738–5747. [CrossRef]
9. Fernandez-Basso, C.; Gutiérrez-Batista, K.; Morcillo-Jiménez, R.; Vila, M.A.; Martin-Bautista, M.J. A fuzzy-based medical system for pattern mining in a distributed environment: Application to diagnostic and co-morbidity. *Appl. Soft Comput.* **2022**, *122*, 108870. [CrossRef]
10. Zhang, W.; Li, H.; Li, Y.; Liu, H.; Chen, Y.; Ding, X. Application of deep learning algorithms in geotechnical engineering: A short critical review. *Artif. Intell. Rev.* **2021**, *54*, 5633–5673. [CrossRef]
11. Zhang, W.; Gu, X.; Tang, L.; Yin, Y.; Liu, D.; Zhang, Y. Application of machine learning, deep learning and optimization algorithms in geoenvironment and geoscience: Comprehensive review and future challenge. *Gondwana Res.* **2022**, *109*, 1–17. [CrossRef]
12. ASHRAE—Great Energy Predictor III. Available online: <https://www.kaggle.com/competitions/ashrae-energy-prediction/data> (accessed on 1 December 2022).
13. ASHRAE. Available online: <https://www.ashrae.org/> (accessed on 15 December 2022).
14. Roldán, J.; Alonso, F.; Aguilera, A.; Maldonado, D.; Lanza, M. Time series statistical analysis: A powerful tool to evaluate the variability of resistive switching memories. *J. Appl. Phys.* **2019**, *125*, 174504. [CrossRef]
15. Shende, M.K.; Feijoo-Lorenzo, A.E.; Bokde, N.D. cleanTS: Automated (AutoML) Tool to Clean Univariate Time Series at Microscales. *Neurocomputing* **2022**, *500*, 155–176. [CrossRef]
16. Rodrigues, J.; Folgado, D.; Belo, D.; Gamboa, H. SSTS: A syntactic tool for pattern search on time series. *Inf. Process. Manag.* **2019**, *56*, 61–76. [CrossRef]
17. Li, M.; Hinnov, L.; Kump, L. Acycle: Time-series analysis software for paleoclimate research and education. *Comput. Geosci.* **2019**, *127*, 12–22. [CrossRef]
18. Antoniadis, I.P.; Brandi, G.; Magafas, L.; Di Matteo, T. The use of scaling properties to detect relevant changes in financial time series: A new visual warning tool. *Phys. A Stat. Mech. Its Appl.* **2021**, *565*, 125561. [CrossRef]
19. Quoilin, S.; Kavvadias, K.; Mercier, A.; Pappone, I.; Zucker, A. Quantifying self-consumption linked to solar home battery systems: Statistical analysis and economic assessment. *Appl. Energy* **2016**, *182*, 58–67. [CrossRef]
20. Christ, M.; Braun, N.; Neuffer, J.; Kempa-Liehr, A.W. Time Series Feature Extraction on basis of Scalable Hypothesis tests (tsfresh—A Python package). *Neurocomputing* **2018**, *307*, 72–77. [CrossRef]
21. Herzen, J.; Læssig, F.; Piazzetta, S.G.; Neuer, T.; Tafti, L.; Raille, G.; Pottelbergh, T.V.; Pasieka, M.; Skrodzki, A.; Huguenin, N.; et al. Darts: User-Friendly Modern Machine Learning for Time Series. *J. Mach. Learn. Res.* **2022**, *23*, 1–6.
22. Jiang, X. KATS. Available online: <https://github.com/facebookresearch/Kats> (accessed on 1 December 2022).
23. Barandas, M.; Folgado, D.; Fernandes, L.; Santos, S.; Abreu, M.; Bota, P.; Liu, H.; Schultz, T.; Gamboa, H. TSFEL: Time Series Feature Extraction Library. *SoftwareX* **2020**, *11*, 100456. [CrossRef]
24. Hosseini, R.; Chen, A.; Yang, K.; Patra, S.; Su, Y.; Al Orjany, S.E.; Tang, S.; Ahammad, P. Greykite: Deploying Flexible Forecasting at Scale at LinkedIn. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), Washington, DC, USA, 14–18 August 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 3007–3017. [CrossRef]
25. Winedarksea, P. AutoTS. Available online: <https://github.com/winedarksea/AutoTS> (accessed on 1 December 2022).
26. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), San Francisco, CA, USA, 13–17 August 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 785–794. [CrossRef]
27. Quinlan, J. Induction of Decision Trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
28. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
29. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef]

30. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [\[CrossRef\]](#) [\[PubMed\]](#)
31. Bradley, A.P. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognit.* **1997**, *30*, 1145–1159. [\[CrossRef\]](#)
32. Zaharia, M.; Chen, A.; Davidson, A.; Ghodsi, A.; Hong, S.A.; Konwinski, A.; Murching, S.; Nykodym, T.; Ogilvie, P.; Parkhe, M.; et al. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.* **2018**, *41*, 39–45.
33. Bavithra, K.; Siva Kumar, R. Energy Efficient and Reliable K Best Detection Approach with Hybrid Decomposition for WiMAX Applications. *Int. J. Commun. Syst.* **2022**, *35*, e5043. [\[CrossRef\]](#)
34. Tarek, Z.; Shams, M.Y.; Elshewey, A.M.; El-Kenawy, E.S.M.; Ibrahim, A.; Abdelhamid, A.A.; El-Dosuky, M.A. Wind Power Prediction Based on Machine Learning and Deep Learning Models. *Comput. Mater. Contin.* **2023**, *74*, 715–732. [\[CrossRef\]](#)
35. Jeong, S.; Kwon, Y. Energy Efficient Text Spotting Technique for Mobile Edge Computing. In Proceedings of the 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Incheon, Republic of Korea, 13–15 June 2022; pp. 106–109. [\[CrossRef\]](#)
36. Ben-Kiki, O.; Evans, C.; Ingerson, B. Yaml ain't markup language (yaml™) version 1.1. *Work. Draft* **2009**, *5*, 11.
37. Roberto, M.J. TSxtend. 2022. Available online: <https://github.com/robermorjiUgr/Tsxtend> (accessed on 1 December 2022).
38. Ángeles Simarro, M.; García-Mollá, V.M.; Martínez-Zaldívar, F.; Gonzalez, A. Low-complexity soft ML detection for generalized spatial modulation. *Signal Process.* **2022**, *196*, 108509. [\[CrossRef\]](#)
39. Alsharekh, M.F.; Habib, S.; Dewi, D.A.; Albattah, W.; Islam, M.; Albahli, S. Improving the Efficiency of Multistep Short-Term Electricity Load Forecasting via R-CNN with ML-LSTM. *Sensors* **2022**, *22*, 6913. [\[CrossRef\]](#) [\[PubMed\]](#)
40. Tian, T.; Zhao, L.; Wang, X.; Wu, Q.; Yuan, W.; Jin, X. FP-GNN: Adaptive FPGA accelerator for Graph Neural Networks. *Future Gener. Comput. Syst.* **2022**, *136*, 294–310. [\[CrossRef\]](#)
41. Li, X.; Wang, S.; Zhu, G.; Zhou, Z.; Huang, K.; Gong, Y. Data Partition and Rate Control for Learning and Energy Efficient Edge Intelligence. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 9127–9142. [\[CrossRef\]](#)
42. Prasannababu, D.; Amgoth, T. Joint mobile wireless energy transmitter and data collector for rechargeable wireless sensor networks. *Wirel. Netw.* **2022**, *28*, 3563–3576. [\[CrossRef\]](#)
43. Jung, S.; Moon, J.; Park, S.; Rho, S.; Baik, S.W.; Hwang, E. Bagging ensemble of multilayer perceptrons for missing electricity consumption data imputation. *Sensors* **2020**, *20*, 1772. [\[CrossRef\]](#) [\[PubMed\]](#)
44. Pan, J.; Li, C.; Tang, Y.; Li, W.; Li, X. Energy Consumption Prediction of a CNC Machining Process with Incomplete Data. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 987–1000. [\[CrossRef\]](#)
45. Alachiotis, N.; Skrimponis, P.; Pissadakis, M.; Pnevmatikatos, D. Scalable Phylogeny Reconstruction with Disaggregated Near-memory Processing. *Acm Trans. Reconfig. Technol. Syst.* **2022**, *15*, 1–32. [\[CrossRef\]](#)
46. Rahmani, M.K.I.; Khan, F.; Muzaffar, A.W.; Jan, M.A. Internet of Things-Enabled Optimal Data Aggregation Approach for the Intelligent Surveillance Systems. *Mob. Inf. Syst.* **2022**, *2022*, 4681583. [\[CrossRef\]](#)
47. Soga, N.; Sato, S.; Nakahara, H. Energy-efficient ECG signals outlier detection hardware using a sparse robust deep autoencoder. *IEICE Trans. Inf. Syst.* **2021**, *104*, 1121–1129. [\[CrossRef\]](#)
48. Sanyal, S.; Zhang, P. Improving quality of data: IoT data aggregation using device to device communications. *IEEE Access* **2018**, *6*, 67830–67840. [\[CrossRef\]](#)
49. Reddy, K.H.K.; Luhach, A.K.; Kumar, V.V.; Pratihari, S.; Kumar, D.; Roy, D.S. Towards energy efficient Smart city services: A software defined resource management scheme for data centers. *Sustain. Comput. Inform. Syst.* **2022**, *35*, 100776. [\[CrossRef\]](#)
50. Feng, C.; Huang, Y.; Wu, Y.; Zhang, J. Feature-based optimization method integrating sequencing and cutting parameters for minimizing energy consumption of CNC machine tools. *Int. J. Adv. Manuf. Technol.* **2022**, *121*, 503–515. [\[CrossRef\]](#)
51. Li, X.; Zhong, K.; Feng, L. Machine learning-based metaheuristic optimization of an integrated biomass gasification cycle for fuel and cooling production. *Fuel* **2023**, *332*, 125969. [\[CrossRef\]](#)
52. Munawar, U.; Wang, Z. Coordinated integration of distributed energy resources in unit commitment. *Int. J. Electr. Power Energy Syst.* **2023**, *145*, 108671. [\[CrossRef\]](#)
53. Chi, L.; Su, H.; Zio, E.; Qadrdan, M.; Zhou, J.; Zhang, L.; Fan, L.; Yang, Z.; Xie, F.; Zuo, L.; et al. A systematic framework for the assessment of the reliability of energy supply in Integrated Energy Systems based on a quasi-steady-state model. *Energy* **2023**, *263*, 125740. [\[CrossRef\]](#)
54. Hai, T.; Hikmat Hama Aziz, K.; Zhou, J.; Dhahad, H.A.; Sharma, K.; Fahad Almojil, S.; Ibrahim Almohana, A.; Fahmi Alali, A.; Ismail Kh, T.; Mehrez, S.; et al. -Neural network-based optimization of hydrogen fuel production energy system with proton exchange electrolyzer supported nanomaterial. *Fuel* **2023**, *332*, 125827. [\[CrossRef\]](#)
55. Cutler, A.; Cutler, D.; Stevens, J. Random Forests. *Mach. Learn.* **2011**, *45*, 157–176. [\[CrossRef\]](#)
56. Zekić-Sušac, M.; Mitrović, S.; Has, A. Machine learning based system for managing energy efficiency of public sector as an approach towards smart cities. *Int. J. Inf. Manag.* **2021**, *58*, 102074. [\[CrossRef\]](#)
57. Pérez-Cutiño, M.; Rodríguez, F.; Pascual, L.; Díaz-Báñez, J. Ornithopter Trajectory Optimization with Neural Networks and Random Forest. *J. Intell. Robot. Syst. Theory Appl.* **2022**, *105*, 17. [\[CrossRef\]](#)
58. Senagi, K.; Jouandeau, N. Parallel construction of Random Forest on GPU. *J. Supercomput.* **2022**, *78*, 10480–10500. [\[CrossRef\]](#)
59. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

60. Aqduş, A.; Amin, R.; Ramzan, S.; Alshamrani, S.S.; Alshehri, A.; El-Kenawy, E.S.M. Detection Collision Flows in SDN Based 5G Using Machine Learning Algorithms. *Comput. Mater. Contin.* **2023**, *74*, 1413–1435. [\[CrossRef\]](#)
61. Ortiz, D.; Migueis, V.; Leal, V.; Knox-Hayes, J.; Chun, J. Analysis of Renewable Energy Policies through Decision Trees. *Sustainability* **2022**, *14*, 7720. [\[CrossRef\]](#)
62. Sakshi, T.; Singh, P. Short Term and Long term Building Electricity Consumption Prediction Using Extreme Gradient Boosting. *Recent Adv. Comput. Sci. Commun.* **2022**, *15*, 1082–1095. [\[CrossRef\]](#)
63. Sauer, J.; Mariani, V.C.; dos Santos Coelho, L.; Ribeiro, M.H.D.M.; Rampazzo, M. Extreme gradient boosting model based on improved Jaya optimizer applied to forecasting energy consumption in residential buildings. *Evol. Syst.* **2022**, *13*, 577–588. [\[CrossRef\]](#)
64. Nayakwadi, N.; Fatima, R. Automatic handover execution technique using machine learning algorithm for heterogeneous wireless networks. *Int. J. Inf. Technol.* **2021**, *13*, 1431–1439. [\[CrossRef\]](#)
65. Mariano-Hernández, D.; Hernández-Callejo, L.; Solís, M.; Zorita-Lamadrid, A.; Duque-Pérez, O.; Gonzalez-Morales, L.; García, F.S.; Jaramillo-Duque, A.; Ospino-Castro, A.; Alonso-Gómez, V.; et al. Analysis of the Integration of Drift Detection Methods in Learning Algorithms for Electrical Consumption Forecasting in Smart Buildings. *Sustainability* **2022**, *14*, 5857. [\[CrossRef\]](#)
66. Himeur, Y.; Alsalemi, A.; Bensaali, F.; Amira, A.; Al-Kababji, A. Recent trends of smart nonintrusive load monitoring in buildings: A review, open challenges, and future directions. *Int. J. Intell. Syst.* **2022**, *37*, 7124–7179. [\[CrossRef\]](#)
67. Zhou, M.; Shao, S.; Wang, X.; Zhu, Z.; Hu, F. Deep Learning-Based Non-Intrusive Commercial Load Monitoring. *Sensors* **2022**, *22*, 5250. [\[CrossRef\]](#)
68. Kalapothas, S.; Flamis, G.; Kitsos, P. Efficient Edge-AI Application Deployment for FPGAs. *Information* **2022**, *13*, 279. . [\[CrossRef\]](#)
69. Bouhamed, O.; Amayri, M.; Bouguila, N. Weakly Supervised Occupancy Prediction Using Training Data Collected via Interactive Learning. *Sensors* **2022**, *22*, 3186. . [\[CrossRef\]](#)
70. Hagan, M.T.; Menhaj, M.B. Training Feedforward Networks with the Marquardt Algorithm. *IEEE Trans. Neural Netw.* **1994**, *5*, 989–993. [\[CrossRef\]](#) [\[PubMed\]](#)
71. BrownLee, J. *Deep Learning for Time Series Forecasting*; Machine Learning Mastery: San Francisco, CA, USA, 2020.
72. Zhou, G.; Moayed, H.; Foong, L.K. Teaching-learning-based metaheuristic scheme for modifying neural computing in appraising energy performance of building. *Eng. Comput.* **2021**, *37*, 3037–3048. [\[CrossRef\]](#)
73. Irfan, M.; Ramlie, F.; Widiyanto, W.; Lestandy, M.; Faruq, A. Prediction of Residential Building Energy Efficiency Performance using Deep Neural Network. *IAENG Int. J. Comput. Sci.* **2021**, *48*, 731–737.
74. Ibrahim, D.M.; Almhafdy, A.; Al-Shargabi, A.A.; Alghieth, M.; Elragi, A.; Chiclana, F. The use of statistical and machine learning tools to accurately quantify the energy performance of residential buildings. *PeerJ Comput. Sci.* **2022**, *8*, e856. [\[CrossRef\]](#) [\[PubMed\]](#)
75. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2012**, *2*, 1097–1105. [\[CrossRef\]](#)
76. Mozafari, M.; Kheradpisheh, S.R.; Masquelier, T.; Nowzari-Dalini, A.; Ganjtabesh, M. First-spike-based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 6178–6190. . [\[CrossRef\]](#) [\[PubMed\]](#)
77. Wu, Z.; Zhang, H.; Lin, Y.; Li, G.; Wang, M.; Tang, Y. LIAF-Net: Leaky Integrate and Analog Fire Network for Lightweight and Efficient Spatiotemporal Information Processing. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 6249–6262. . [\[CrossRef\]](#)
78. Chakraborty, I.; Roy, D.; Roy, K. Technology Aware Training in Memristive Neuromorphic Systems for Nonideal Synaptic Crossbars. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 335–344. . [\[CrossRef\]](#)
79. Xu, Y.H.; Sun, Q.M.; Zhou, W.; Yu, G. Resource allocation for UAV-aided energy harvesting-powered D2D communications: A reinforcement learning-based scheme. *Ad Hoc Netw.* **2022**, *136*, 102973. [\[CrossRef\]](#)
80. Jayanthi, E.; Vallikannu, R. Enhancing the performance of asymmetric architectures and workload characterization using LSTM learning algorithm. *Adv. Eng. Softw.* **2022**, *173*, 103266. [\[CrossRef\]](#)
81. Wu, B.; Wang, Z.; Chen, K.; Yan, C.; Liu, W. GBC: An Energy-Efficient LSTM Accelerator With Gating Units Level Balanced Compression Strategy. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, *69*, 3655–3665. [\[CrossRef\]](#)
82. Zeng, J.; Ding, D.; Kang, K.; Xie, H.; Yin, Q. Adaptive DRL-Based Virtual Machine Consolidation in Energy-Efficient Cloud Data Center. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2991–3002. [\[CrossRef\]](#)
83. Gressling, T. 11 Python standard libraries and Conda. In *Data Science in Chemistry*; De Gruyter: Berlin, Germany, 2020; pp. 45–54.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.