NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Time Series Forecasting Applied to an Energy Management System

A Comparison Between Deep Learning Models and Other Machine Learning Models

João Henrique Leal Arienti

Project Work presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

TIME SERIES FORECASTING APPLIED TO AN ENERGY MANAGEMENT SYSTEM

A Comparison Between Deep Learning Models and Other Machine Learning Models

João Henrique Leal Arienti

Project Work presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

Advisor: Mauro Castelli, Ph.D.

August 2020

DEDICATION

I would like to dedicate this thesis to my family. To my father and mother, both former university professors, for always encouraging me and supporting me in all my academic years. To my wife who made it possible for me to work, to study, and to take care of our wonderful daughters, one of them being born in the middle of the master program. To my sister who was doing a master thesis in the medical field at the same time as me and gave me important tips and also motivated me. For all of you, my most sincere gratitude.

ACKNOWLEDGEMENTS

I would like to deeply thank my thesis advisor, Dr. Mauro Castelli. Thank you for being available for advising me and for doing this incredibly fast. I always had a prompt answer which helped a lot in developing the thesis on the right path from the beginning.

I would like to thank my friend and master's colleague Biazi Bayer for helping me find an inspiring theme and project to support my thesis. And for always motivating me and being available to help.

I would also like to thank my colleagues from the Ambiosensing project for their collaboration and teamwork.

ABSTRACT

A large amount of energy used by the world comes from buildings' energy consumption. HVAC (Heat, Ventilation, and Air Conditioning) systems are the biggest offenders when it comes to buildings' energy consumption. It is important to provide environmental comfort in buildings but indoor well-being is directly related to an increase in energy consumption. This dilemma creates a huge opportunity for a solution that balances occupant comfort and energy consumption. Within this context, the Ambiosensing project was launched to develop a complete energy management system that differentiates itself from other existing commercial solutions by being an inexpensive and intelligent system. The Ambiosensing project focused on the topic of Time Series Forecasting to achieve the goal of creating predictive models to help the energy management system to anticipate indoor environmental scenarios. A good approach for Time Series Forecasting problems is to apply Machine Learning, more specifically Deep Learning. This work project intends to investigate and develop Deep Learning and other Machine Learning models that can deal with multivariate Time Series Forecasting, to assess how well can a Deep Learning approach perform on a Time Series Forecasting problem, especially, LSTM (Long Short-Term Memory) Recurrent Neural Networks (RNN) and to establish a comparison between Deep Learning and other Machine Learning models like Linear Regression, Decision Trees, Random Forest, Gradient Boosting Machines and others within this context.

KEYWORDS

Artificial Intelligence; Big Data; Data Science; Deep Learning; Energy Management System; Heat, Ventilation, and Air Conditioning; Internet of Things; Long Short-Term Memory; Machine Learning; Neural Network; Recurrent Neural Networks; Time Series Forecasting

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
BPTT	Backpropagation Through Time
CART	Classification and Regression Trees
CNN	Convolutional Neural Networks
CoAP	Constrained Application Protocol
CSV	Comma Separated Value
DL	Deep Learning
GBM	Gradient Boosting Machines
GHCN	Global Historical Climatology Network
HTTP	Hypertext Transfer Protocol
HVAC	Heat, Ventilation, and Air Conditioning
IBM	Internet Business Machine
IoT	Internet of Things
IQR	Interquartile Range
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbors
LASSO	Least Absolute Shrinkage and Selection Operator
LR	Linear Regression
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptrons
MQTT	Message Queuing Telemetry Transport
MSE	Mean Squared Error
MSL	Mean Sea-Level
NN	Neural Network
NOAA	National Oceanic and Atmospheric Administration
NoSQL	Not Only SQL
OS	Operating System
PTFE	Polytetrafluoroethylene
RF	Random Forest
RFE	Recursive Feature Selection
RMSE	Root Mean Squared Error
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Networks
RPC	Remote Procedure Calls
RTRL	Real-Time Recurrent Learning

SD	Secure Digital
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SSE	Sum of Squared Errors
SVR	Support-Vector Regression

1. INTRODUCTION

1.1. MOTIVATION

A large amount of energy used by the world comes from buildings' energy consumption. Global wise, buildings are responsible for the consumption of 40% of the produced energy. In Portugal, buildings are responsible for the use of approximately 30% of the energy available for consumption (Bernardo, 2015) (Direcção Geral de Energia e Geologia, 2017). Within this context, any potential increase in buildings' energy efficiency will contribute significantly to a reduction in the global energy requirements (Pisello, Bobker, & Cotana, 2012).

Since a building has a considerable lifespan, it is critical to improve its energy efficiency. One possibility to reduce substantially the energetic requirements of a building is to optimize its HVAC (Heat, Ventilation, and Air Conditioning) system. HVAC systems are the biggest offenders when it comes to buildings' energy consumption. In European countries, HVAC systems could achieve 76% of the total energy usage of a building (Oldewurtel et al., 2012).

Even though most buildings are intended for human habitation, there is a huge amount of buildings that are designated to commerce, industry, and services. The excessive energy consumption verified in commercial buildings, for instance, is related to the lack of an energy management system, but also to a variety of inefficient users' behaviors, to isolation problems, and intensive equipment usage.

As people spend almost 90% of their time in buildings, it is important to provide environmental comfort. But, the fact is that the improvement of indoor well-being is directly related to an increase in energy consumption. This dilemma creates a huge opportunity for a solution that balances occupant comfort and energy consumption. This need could be answered by an energy management system that would be able to monitor, control, and optimize building services (Shaikh, Nor, Nallagownden, Elamvazuthi, & Ibrahim, 2013).

To be economically viable, an energy management system needs to be based on inexpensive technology and equipment. Other than that, it also needs to integrate intelligence into control and automation components of the system. With the use of Machine Learning techniques, it would be possible to create predictive models to help the system to anticipate environmental scenarios and to be prepared for that ahead of time, preventing unnecessary energy consumption.

1.2. OBJECTIVES

To develop a solution that balances environmental comfort and energy consumption, Ambiosfera ("Ambiosfera," n.d.), a Portuguese company specialized in energy, environment, and sustainability launched the Ambiosensing project. This initiative seeks to develop a complete energy management system that will differentiate itself from other existing commercial solutions by being an inexpensive and intelligent system.

The main goals of Ambiosensing project are the following:

1. To develop a low-cost and low-maintenance building energy management system;

2. To improve buildings occupant comfort through a system of adaptive user preference modes, keeping always in mind energy efficiency;
3. To develop a solution to monitor, control and manage energy efficiency adaptively to many existing buildings realities;
4. To make the system compatible with different equipment scenarios through the use of IoT's (Internet of Things) most established communication protocols;
5. To build a system that can handle huge amounts of telemetry data based on Big Data technology for data storage and processing;
6. To create a system that is scalable through public or private cloud computing.

Because of the nature of the Ambiosensing project, it deals with a lot of telemetry data, which can be represented as time series data points. For example, temperature and humidity readings from IoT sensors in a timestamp format. Therefore, to achieve the goal of creating predictive models to help the energy management system to anticipate environmental scenarios, it is needed to focus on the topic of Time Series Forecasting. This important field of Data Science involves fitting models on historical data to predict future observations taking into consideration the time component (Brownlee, 2018c).

A good approach for Time Series Forecasting problems is to apply Machine Learning (ML) algorithms because they can handle multiple input variables with noisy complex dependencies. More specifically inside the Machine Learning field, there is an interesting option of Deep Learning (DL). Unlike other Machine Learning algorithms, Deep Learning algorithms can learn features from time series data automatically, besides supporting multiple input variables.

In this context, from a Data Science perspective, the main objectives of the Ambiosensing project are:

1. To investigate and develop Deep Learning and other Machine Learning models that can deal with multivariate Time Series Forecasting;
2. To assess how well can a Deep Learning approach perform on a Time Series Forecasting problem, especially, LSTM (Long Short-Term Memory) Recurrent Neural Networks (RNN), that are capable of automatically learn features from a sequence of data;
3. To establish a comparison between Deep Learning and other Machine Learning models on a Time Series Forecasting problem.

2. THEORETICAL BACKGROUND

2.1. WHAT IS TIME SERIES FORECASTING?

Time Series Forecasting is the Machine Learning area responsible for prediction problems that involve a time component. It is important to distinguish Time Series Analysis from Time Series Forecasting since they have different goals. Time Series Analysis is related to describing, to understanding the dataset. Meanwhile, Time Series Forecasting is associated with predicting, with forecasting the future (Brownlee, 2018c, p. 9;10).

Times Series Analysis can be defined as the systematic approach of answering the mathematical and statistical questions posed by the time correlation introduced by the sampling of adjacent points in time. It is the analysis of experimental data that have been observed at different points in time (Shumway & Stoffer, 2017, p. 1). Time Series Analysis usually involves the study of the form of the data and of the components of a time series (Brownlee, 2018c, p. 11).

Time Series Forecasting can be defined as the process of taking historical data of a time series, sometimes with additional information, and fitting models to forecast future values. Unlike Time Series Analysis that can be done in retrospect and use "future" information, forecasting models don't have "future" information available. Everything must be done based only on what has already happened. A predictive model is evaluated by its predictive accuracy. Meanwhile, a descriptive model is assessed by its capability of providing correct causal explanations (Shmueli & Lichtendahl, 2016, p. 19).

2.2. DATA PREPARATION

In many cases, the dataset will require different types of data preparation. Whenever features present themselves in dissimilar scales, it is the case to consider standardization or normalization. Both techniques intend to make features comparable to one another. For example, input variables pressure and temperature could be very different in scale. Pressure could be measured in hPa (hectopascal), which at Mean Sea-Level (MSL) is 1013,25 hPa (or 1 atm) by the International Organization for Standardization (ISO) ("International Organization for Standardization," 1975). Still, temperature could be measured in Celsius degrees, which is usually between -40 and 40 degrees in its extremes. So, one variable has values in the range of the tens and the other in the thousands.

Some Machine Learning models will perform better if the dataset has consistent scales. That is the case of algorithms like Regression, and K-Nearest Neighbors. These algorithms take advantage of rescaled datasets for different reasons. Regression because it weights input variables and K-Nearest Neighbors because it uses distance measures (Brownlee, 2019, p. 48).

2.2.1. Standardization

Standardization (or z-score normalization) is the technique that rescales the data by transforming the mean of the distribution to zero and its standard deviation to one like a standard normal distribution (or standard Gaussian distribution) (Burkov, 2019, p. 39).

It is represented by the following formula:

$$y = (x - \text{mean}) / \text{standard deviation}$$

Where *mean* is calculated as:

$$\text{mean} = \text{sum}(x) / \text{count}(x)$$

And the *standard deviation* is calculated as:

$$\text{standard deviation} = \text{sqrt}(\text{sum}((x - \text{mean})^2) / \text{count}(x))$$

It is possible to standardize a dataset by using the class StandardScaler from Python's library scikit-learn ("sklearn.preprocessing.StandardScaler — scikit-learn 0.23.2 documentation," n.d.).

2.2.2. Normalization

Normalization (or min-max normalization) is the technique that rescales the data by converting the actual range of values to a standard range of values. Usually, it is done in the interval between 0 and 1 or -1 and 1 (Burkov, 2019, p. 39).

It is represented by the following formula:

$$y = (x - \text{min}) / (\text{max} - \text{min})$$

Where *min* and *max* stand for the minimum and the maximum values of *x* in the dataset.

It is possible to normalize a dataset by using the class MinMaxScaler from Python's library scikit-learn ("sklearn.preprocessing.MinMaxScaler — scikit-learn 0.23.2 documentation," n.d.).

2.3. Model Evaluation

Model evaluation is an estimate of how well the model will perform in practice. Usually, to perform a model evaluation, the dataset is split between training and test sets. Model evaluation cannot be done using the same data as in training. Otherwise, any estimate of performance would be optimistic. If training data is used on evaluation, the algorithm is likely to score perfectly. However, there is a very good chance that the predictions will be wrong (Brownlee, 2019, p. 57).

Therefore, the model evaluation must be done based on the test set. That is why the testing set is held back and it is an unseen data to the model. The algorithm never had contact with the testing set before model evaluation (Brownlee, 2018c, p. 145).

Sometimes, the dataset is split into three parts: training, validation, and test sets. In this case, training data is used by the model to learn, validation data is utilized to select the best model, and to select parameters from estimates of performance and test data is employed, exclusively, to assess performance at the end of the process (Igual & Seguí, 2017, p. 82).

2.3.1. Train-Test Split

Train-test split is the simplest method for model evaluation. And, if the dataset is large enough, both train and test splits could incorporate different patterns representing well the problem and the resulting estimate of performance could be consistent. Another positive aspect of the train-test split is that it is a fast method.

Train-test split consists of splitting the dataset into two parts: training and test sets. One for training the algorithm and the other to make predictions and compare them to the already known values. That way, it is possible to calculate a performance metric and to assess the accuracy of the model.

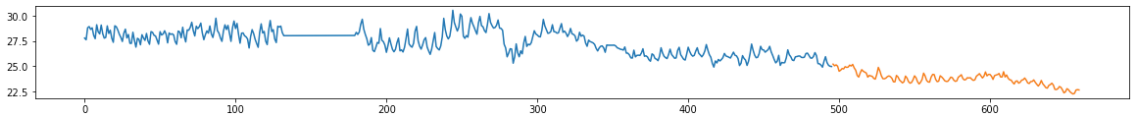


Figure 2-1 - Train and test splits in blue and orange respectively.

The negative aspect of the train-test split is that differences in training and test sets could end up in dissimilarities in the estimate of performance (Brownlee, 2019, p. 58).

It is possible to create a train-test split of the dataset by using the function train_test_split from Python's library scikit-learn ("sklearn.model_selection.train_test_split — scikit-learn 0.23.2 documentation," n.d.).

2.3.2. K-fold Cross-Validation

K-fold Cross-Validation is a more robust method of model evaluation than Train-Test Split. That is because instead of a single train-test split, in cross-validation, the model is trained and evaluated multiple times on different data (Brownlee, 2019, p. 59).

In K-fold Cross-Validation, the dataset is split into K-parts of equal size. These parts are known as folds. Then, the model is trained K times. Each time, the model is evaluated based on only one fold and it is trained using the remaining folds. This process is repeated in such a way that, by the end, every fold has performed the hole of validation data once. In the end, the K-fold Cross-Validation method provides K different accuracy values, that can be summarized using mean and standard deviation (Müller & Guido, 2016, p. 252).

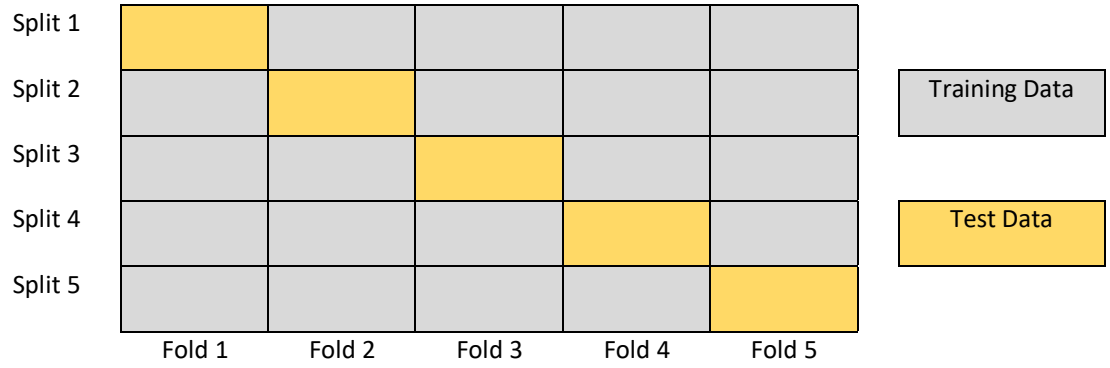


Figure 2-2 – Split and fold scheme of a 5-fold cross-validation example.

One of the positive aspects of K-fold Cross-Validation is that differences in the training and test sets are not as harmful as in Train-Test Split. For example, suppose that Fold 1 from Split 1 has data with a very distinct pattern when compared to Folds 2 to 5 also from Split 1. This would affect Split 1 performance score. But, by the time the model is trained for the second time with Split 2, the differences in Fold 1 are incorporated in the training data and represent a minor part of the training data. That is something that repeats through all the remaining splits until the end. Therefore, the impact of differences in training and test sets is minimized, resulting in a much more stable performance score. The main negative aspect of K-fold Cross-Validation is the computational cost since the model is trained K-times instead of once (Müller & Guido, 2016, p. 254).

It is possible to apply K-fold cross-validation by using the class KFold and the function cross_val_score from Python's library scikit-learn ("sklearn.model_selection.KFold — scikit-learn 0.23.2 documentation," n.d.) ("sklearn.model_selection.cross_val_score — scikit-learn 0.23.2 documentation," n.d.).

2.3.3. Multiple Train-Test Splits (applied to Time Series)

K-fold Cross-Validation is not really suited for model evaluation of time series because it is a method that ignores the temporal components inherent to the problem (Brownlee, 2018c, p. 145). Train-Test Split, on the other hand, has the downside of high variance when training and test sets have differences in data (Brownlee, 2019, p. 58). So, for a Time Series Forecasting problem, a more appropriate method of model evaluation would be Multiple Train-Test Splits.

Multiple Train-Test Splits consists of repeating the process of splitting the dataset into training and test sets several times. In each iteration, the training set gets larger and the test set remains the same for the sake of comparability among the performance scores obtained (Brownlee, 2018c, pp. 148, 149).

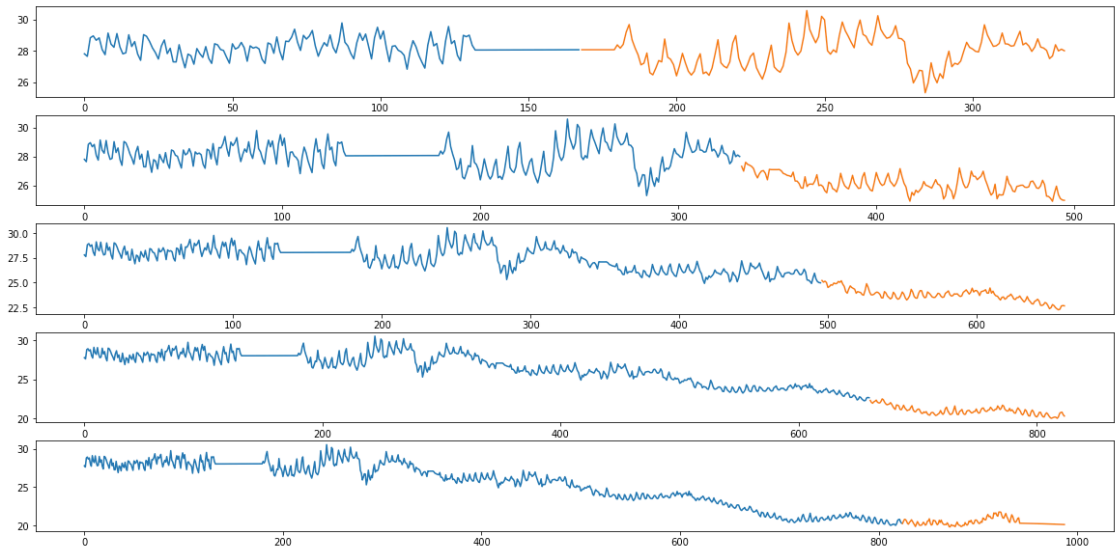


Figure 2-3 - Multiple Train-Test Splits created by TimeSeriesSplit configured for 5 splits.

It is possible to create multiple splits of a times series by using the class TimeSeriesSplit from Python's library scikit-learn. The training set size is calculated by the following formula ("sklearn.model_selection.TimeSeriesSplit — scikit-learn 0.23.2 documentation," n.d.):

$$\text{training set size} = (i * (\text{samples} / (\text{splits} + 1))) + \text{samples mod } (\text{splits} + 1)$$

Where *i* is the split iteration, *samples* is the number of samples, and *splits* is the number of splits desired. The part *samples mod (splits + 1)* is the remainder of the division of the number of samples by the number of splits plus one.

Meanwhile, the test set size is calculated as follows ("sklearn.model_selection.TimeSeriesSplit — scikit-learn 0.23.2 documentation," n.d.):

test index = samples / (splits + 1)

Where *samples* is the number of samples and *splits* is the number of splits wanted.

Below, an example of Multiple Train-Test Splits set to create 5 splits out of a dataset with 988 observations:

```
Split #1:
Total obs.: 332
Train shape: (168, 1)
Valid shape: (164, 1)

Split #2:
Total obs.: 496
Train shape: (332, 1)
Valid shape: (164, 1)

Split #3:
Total obs.: 660
Train shape: (496, 1)
Valid shape: (164, 1)

Split #4:
Total obs.: 824
Train shape: (660, 1)
Valid shape: (164, 1)

Split #5:
Total obs.: 988
Train shape: (824, 1)
Valid shape: (164, 1)
```

Figure 2-4 - Example of 5 splits created by TimeSeriesSplit.

2.4. PERFORMANCE METRICS

A Time Series Forecasting problem is a regression problem. And a regression problem is focused on the prediction of real values. A direct way to evaluate time series forecasts is based on the difference between the predicted values and the expected values.

Three common performance metrics for Time Series Forecasting problems are: Mean Absolute Error, Mean Squared Error, and Root Mean Squared Error.

2.4.1. Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a performance metric calculated as the average of the absolute differences between the predicted values and the expected values. This measure has as a positive aspect the fact that is simple and that it indicates the magnitude of the error. However, it has as its negative aspect the fact that it does not inform the direction of the error, because the differences are being forced as positive values (Brownlee, 2019, p. 67) (Skiena, 2017, p. 221).

The Mean Absolute Error is represented by the following formula:

$$mae = mean(abs(expected\ values - predicted\ values))$$

The Mean Absolute Error can be calculated by the function mean_absolute_error from Python's library scikit-learn ("sklearn.metrics.mean_absolute_error — scikit-learn 0.23.2 documentation," n.d.).

2.4.2. Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a performance metric calculated as the average of the squared differences between predicted values and the expected values. As in MAE, the differences are being forced to become positive values but, this time, by squaring and not by making them absolute. Squaring has the potential side effect of outliers dominating the error statistics. The MSE metric has the benefit of large error values contributing more to worsening the performance score. Therefore, it is an informative measure, especially, for noisy instances (Brownlee, 2019, p. 68) (Skiena, 2017, p. 222).

The Mean Squared Error is represented by the following formula:

$$mse = mean((expected\ values - predicted\ values)^2)$$

The Mean Squared Error can be calculated by the function mean_squared_error from Python's library scikit-learn ("sklearn.metrics.mean_squared_error — scikit-learn 0.23.2 documentation," n.d.).

2.4.3. Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is a performance metric that is simply the square root of the Mean Squared Error. The positive aspect of the RMSE is that it is on the same scale as the original values and, therefore, its magnitude is much more interpretable (Skiena, 2017, p. 223).

The Root Mean Squared Error is represented by the following formula:

$$rmse = sqrt(mean((expected\ values - predicted\ values)^2))$$

The Root Mean Squared Error can be calculated by the function mean_squared_error from Python's library scikit-learn. In recent versions of the scikit-learn library, in v0.23.1 for sure, function mean_squared_error has a parameter called *squared* that, when set to False, returns RMSE value. The default value is True, though. ("sklearn.metrics.mean_squared_error — scikit-learn 0.23.2 documentation," n.d.).

2.5. Machine Learning Models

2.5.1. Linear Algorithms

2.5.1.1. Linear Regression

The Linear Regression algorithm uses a linear function of the input variables to make predictions. Linear Regression is also known as Ordinary Least Squares. In Linear Regression, the predicted value is thought as the weighted sum of the input variables. Linear Regression can represent predictions as a line for a single feature, as a plane for two features or as a hyper-plane for more features (Müller & Guido, 2016, pp. 45–47). The objective in Linear Regression is to find the plane that minimizes the Sum of Squared Errors (SSE) between predictions and true values (Kuhn & Johnson, 2013, p. 105).

Linear regression has as positive aspects the fact that is easy to compute and that the coefficients, the weights in the sum of the input variables, are directly interpretable. Linear Regression has a couple of restrictions, though. An input variable cannot be determined from a combination of one or more of the other input variables (collinearity). And the number of observations must be greater than the number of input variables. Otherwise, it is impossible to reach a unique linear combination of the features to represent the target (Kuhn & Johnson, 2013, p. 108). Since Linear Regression has no parameters, it has no way to control model complexity (Müller & Guido, 2016, p. 47).

2.5.1.2. LASSO Regression

LASSO (Least Absolute Shrinkage and Selection Operator) Regression is another linear model for regression. The prediction is also thought of as the weighted sum of the input variables like in Linear Regression. However, the weights, which are the coefficients, are not only chosen to predict well on training data but also to respect a constraint. The idea of having a constraint to control model complexity and to avoid overfitting is known as regularization (Müller & Guido, 2016, p. 53).

In LASSO regression the constraint restricts the magnitude of the coefficients to be close or equal to zero. This type of regularization, where the sum of the absolute values of the regression coefficients is penalized, is known as L1 regularization. When the L1 regularization forces coefficients to be zero, in practice, the LASSO regression algorithm is performing a feature selection. Hence, the Selection Operator in the name of the method. LASSO regression achieves both improving the model quality and conducting variable selection by applying L1 regularization (Dinov, 2018, p. 579).

2.5.2. Nonlinear Algorithms

2.5.2.1. K-Nearest Neighbors

The K-Nearest Neighbors algorithm can be applied to a classification (K-Neighbors Classification) or a regression (K-Neighbors Regression) problem. It is a straightforward algorithm, easy to interpret, and the main logic is the same for both cases. The training phase consists only of storing the data points. Later, to predict, the algorithm just finds the k-closest data points in the training data, which are considered the nearest neighbors. In the simplest version of the K-Neighbors Regression, where K is defined as one, the prediction is just the target value of the nearest neighbor. When K is larger than one, the prediction is the mean of the K-nearest neighbors (Müller & Guido, 2016, pp. 35, 40, 41).

K-Nearest Neighbors algorithm can be applied through the KNeighborsRegressor class of Python's scikit-learn library. The distance metric used as default is the Minkowski distance, which is a generalization of the Euclidean distance and the Manhattan distance (Brownlee, 2019, p. 80). Since the power parameter for the Minkowski distance is set as 2 by default, the Euclidean distance is applied if not otherwise configured. When the power parameter is set as 1, Manhattan distance is employed ("sklearn.neighbors.KNeighborsRegressor — scikit-learn 0.23.2 documentation," n.d.).

The positive aspects of the K-Nearest Neighbors algorithm are that it is easy to understand and that it achieves reasonable performance scores without many adjustments. These characteristics make it a good baseline method. The downside of the K-Nearest Neighbors algorithm is that it is not fast when the training dataset is very large, even more, because it is an algorithm that requires pre-processing. Other than that, it is an algorithm that doesn't deal well with a dataset where most values are zeros either (Müller & Guido, 2016, p. 44).

2.5.2.2. Classification and Regression Trees

Decision Trees or Classification and Regression Trees (CART) is another algorithm that also can be used to classify or to predict. Classification or numeric prediction trees are built based on the same logic. Multiple if-else logical decisions are used to partition the data via a divide-and-conquer strategy based on features (Dinov, 2018, p. 373).

The Decision Trees algorithm builds a tree through the recursive process of searching over all possible splits within the dataset the one that is most informative about the target variable. The recursive partitioning of the data results in a tree of decisions that has nodes and leaves. In Regression Trees, nodes contain tests like "is feature a larger than value b?". Leaves are terminal nodes that contain answers. In Regression Tress, for instance, a regression value (Müller & Guido, 2016, pp. 71–73).

The Decision Trees algorithm has the upside of being easy to implement and highly interpretable. Besides, decision trees can handle many different types of input variables without pre-processing. Decision trees can also handle missing values and, because of the logic to build the trees, can perform feature selection either. However, decision trees do have negative aspects. One of them is model instability because minor changes in the data can affect the tree structure, compromising interpretability. Another one is predictive performance. Decision trees will have larger prediction errors if the relationship between input variables and target variables cannot be represented as rectangular subspaces of the input variables (Kuhn & Johnson, 2013, p. 174).

2.5.3. Ensemble Algorithms

2.5.3.1. Random Forest

Random Forest is an Ensemble method of the type Bagging. A Bagging or a Bootstrap Aggregation algorithm trains the model multiple times based on subsets of the dataset. By the end, the prediction is obtained by averaging all the results collected from the multiple trained models. Through this approach, Bagging tries to reduce the variance of the prediction (Brownlee, 2019, p. 92).

The Random Forest algorithm is a generalization of the Bagging method applied to Decision Trees. In Random Forest, randomness is inserted in the process of building the decision tree. Instead of choosing the best split point from the entire dataset, only a random sample from the original data, with replacement, is considered. The main idea is to use random decision trees and, by doing so, provide less correlation among different ensemble components (Aggarwal, 2015, p. 380).

2.5.3.2. Gradient Boosting Machines

Gradient Boosting Machines are an Ensemble method of the type Boosting. A Boosting algorithm creates a sequence of models. The main idea is that each subsequent model tries to correct the mistakes from the previous model (Brownlee, 2019, p. 94).

The basic principle of Gradient Boosting Machines is based on two things: a loss function and a weak learner. A loss function is a measure that indicates how good a combination of the coefficients fits the data (Müller & Guido, 2016, p. 56). For instance, squared error for regression. A weak learner is a model that can only provide good predictions on part of the data (Müller & Guido, 2016, p. 89). For instance, a shallow tree. The main goal of the algorithm is to search for an additive model out of the weak learners that can minimize the loss function (Kuhn & Johnson, 2013, pp. 204–206).

Gradient Boosting Machines are also referred to as Stochastic Gradient Boosting but, actually, Stochastic Gradient Boosting brings a minor tweak. It borrows the idea of random sampling the training data from Bagging to reduce prediction variance (Kuhn & Johnson, 2013, pp. 204–206).

2.6. DEEP LEARNING

2.6.1. Neural Networks

Artificial Neural Networks or just Neural Networks are a family of Machine Learning algorithms that encompasses different methods like Multilayer Perceptrons (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). Deep Learning and Neural Networks are considered to be the same field. Deep Learning is a kind of learning based on deep neural networks, networks that have several stacked layers to improve their predictive capability (Skansi, 2018, p. preface V).

Neural Networks models mimic the human nervous system response to external stimuli. Neural Networks simulate the brain using a network of interconnected nodes, known as neurons, just like nervous system cells. A neuron from a Neural Network can receive input data, compute on this data, and send the results to another neuron in the network. Each neuron is accompanied by a weight that defines its computation function. The learning in Neural Networks happens by changing theses weights accordingly. The main idea of this stage of the Neural Network learning is to modify the weights incrementally whenever a wrong prediction is made (Aggarwal, 2015, p. 326).

The following formula represents the mathematical relationship between the output and the inputs in a Neural Network:

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i + \theta\right)$$

Where *y* is the output and *f* is the activation function applied to the sum of the weighted inputs and θ is the threshold. The activation function is a key aspect of Neural Networks. It is a nonlinear function like the *rectifying nonlinearity* (or relu) or the *tangens hyperbolicus* (or tanh) that allows the Neural Network to learn much more complicated problems than a linear model (Müller & Guido, 2016, p. 106).

Below, the representation of a single neuron Perceptron. The Perceptron is a simple Neural Network with one or more neurons positioned in just one layer (Aggarwal, 2015, pp. 326–328).

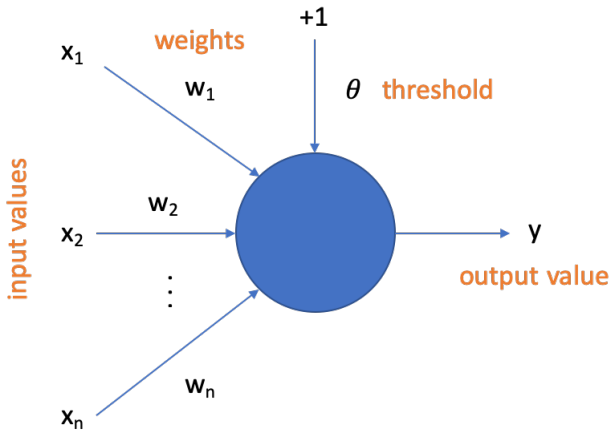


Figure 2-5 – An example of a Perceptron.

Another important aspect of Neural Networks is the network topology. It describes the number of layers and the number of neurons in each layer. Next, a Neural Network with a simple network topology:

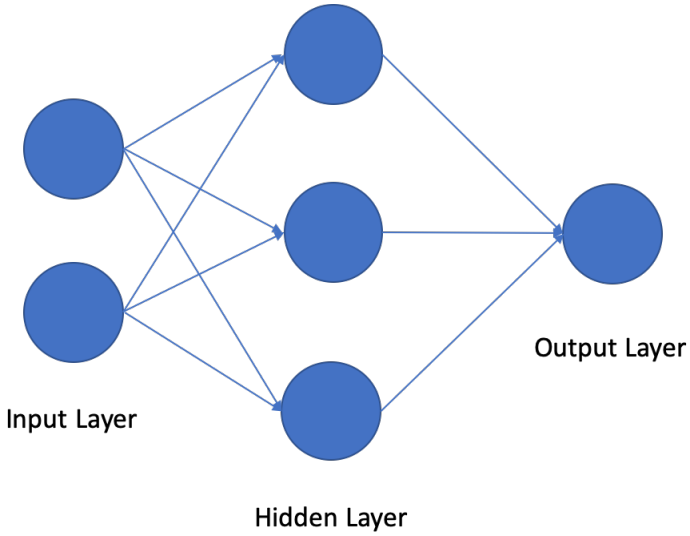


Figure 2-6 - An example of a Neural Network with a simple topology.

The input layer is composed of input nodes that are not really neurons. These units are responsible for taking input data and passing it to the next layer without performing any kind of computation like a neuron would do. Since the input layer is the exposed part of the network it is also known as the visible layer. Hidden layers are the layers of the Neural Network that come after the input layer. The single neuron Perceptron has the most basic network topology possible having only one neuron in the hidden layer. Deep Learning Neural Networks, on the other hand, can have many hidden layers, which can make them very time consuming when it comes to training. The output layer is the last hidden layer and is responsible for outputting the predicted value (Brownlee, 2018b, p. 38).

The training of a Neural Network occurs iteratively through all the instances of the training data and for each one of them it happens in two phases: the forward phase and the backward phase. These phases are part of the algorithm known as Backpropagation. The Backpropagation algorithm is essential to train Neural Networks because it helps to determine the error in the hidden layers. Since there are no expected values in the hidden layers to compare with the computed values, some kind of feedback is required from the forward layers (Aggarwal, 2015, p. 329). First, in the forward phase, each row from the training data is fed to the network. After that, the network forwards computations across the following layers using the current weights. This process ends-up producing an output value for each training instance. It is good to remember that the training data has the expected value to be compared with the predicted value from the output layer. So the error can be calculated (Brownlee, 2018b, p. 39). Next, in the backward phase, the error is backpropagated one layer at a time and the error estimates of the hidden neurons are computed. Lastly, the weights are updated proportionally to how much they contributed to the error. This process is repeated until the last row of the training data (Aggarwal, 2015, p. 329).

2.6.2. Recurrent Neural Networks

Recurrent Neural Networks are a specific type of Neural Networks that is characterized by having a connection not only to the neuron in the next layer but also to itself forming a loop. In some varieties, RNN has a connection to the neuron at the side in the same layer (Graves, 2012, p. 22). This

distinguished architecture gives feedback and memory to the network, two important features that make Recurrent Neural Networks promising to handle problems that involve sequence inputs, like Time Series Forecasting (Brownlee, 2018b, p. 170).

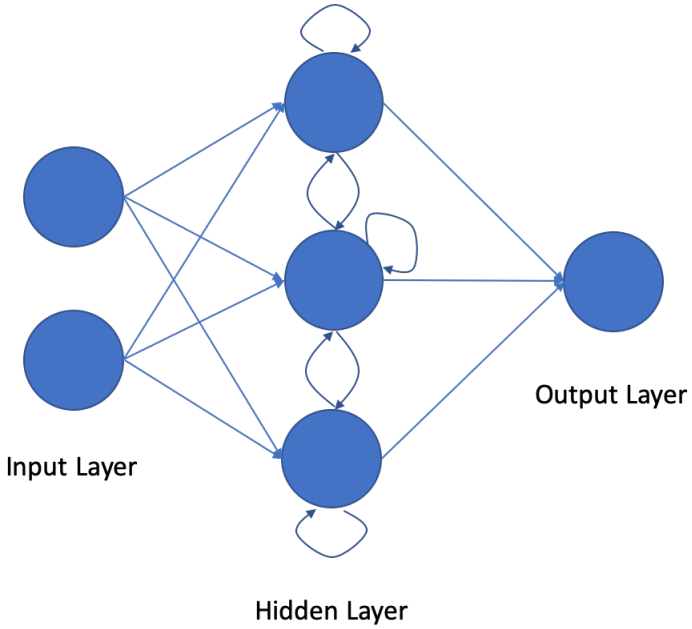


Figure 2-7 - An example of a Recurrent Neural Network (RNN) with cyclical and sideways connections.

The unique loop architecture of RNNs presents some challenges like, for instance, the training of the network. The classic training algorithm Backpropagation cannot be applied to neurons that have connections to itself. It is impossible to backpropagate the error and to update the weights proportionally. So, instead of applying the Backpropagation algorithm, Recurrent Neural Networks employ a variation of this algorithm called Backpropagation Through Time (BPTT). In this new version of the Backpropagation algorithm, the main idea remains. But, the neurons with loops are transformed into two neurons in sequence with the same weights, unrolling the network architecture. The result is a Neural Network very much like any other, one in which the classic Backpropagation algorithm can be applied (Brownlee, 2018b, p. 171). There is another algorithm that also can be applied to calculate weight derivatives efficiently for RNNs. It is called Real-Time Recurrent Learning (RTRL). But, it is more complex and not as efficient in terms of computational time as BPTT (Graves, 2012, p. 23).

2.6.2.1. Long Short-Term Memory Networks

Long Short-Term Memory networks are a specific type of Recurrent Neural Networks. LSTM networks solve the problem of very deep Neural Networks not having stable gradients to updated the weights. In RNN, where the network architecture is unrolled to promote training through BPTT, it is even a bigger problem. LSTM networks have a new type of architecture that addresses this issue (Brownlee, 2018b, p. 172). By having a mechanism that controls the flow of data, it allows relevant information

to pass down a much bigger sequence of nodes without stopping to learn because of very small values of gradients (Skansi, 2018, p. 143).

LSTM networks are distinguished by not having classic neurons. Instead, its computational unit is called a memory block. A memory block differentiates itself from a neuron by having memory and gates. These gates control the input and output of data from the memory block and also the state of the block, if it is activated or not. Each gate has a weight associated with it that are updated during the learning phase and also a sigmoid activation function to control its triggering. The activation function is important because it adds the conditional factor to the gates and, therefore, to its state and to the input and output of data from the memory block. There are three types of gates: input gate, output gate, and forget gate. While the input and the output gates are responsible for conditionally control the input and output of data from the memory, the forget gate is in charge of discarding data, also conditionally (Brownlee, 2018b, p. 172).

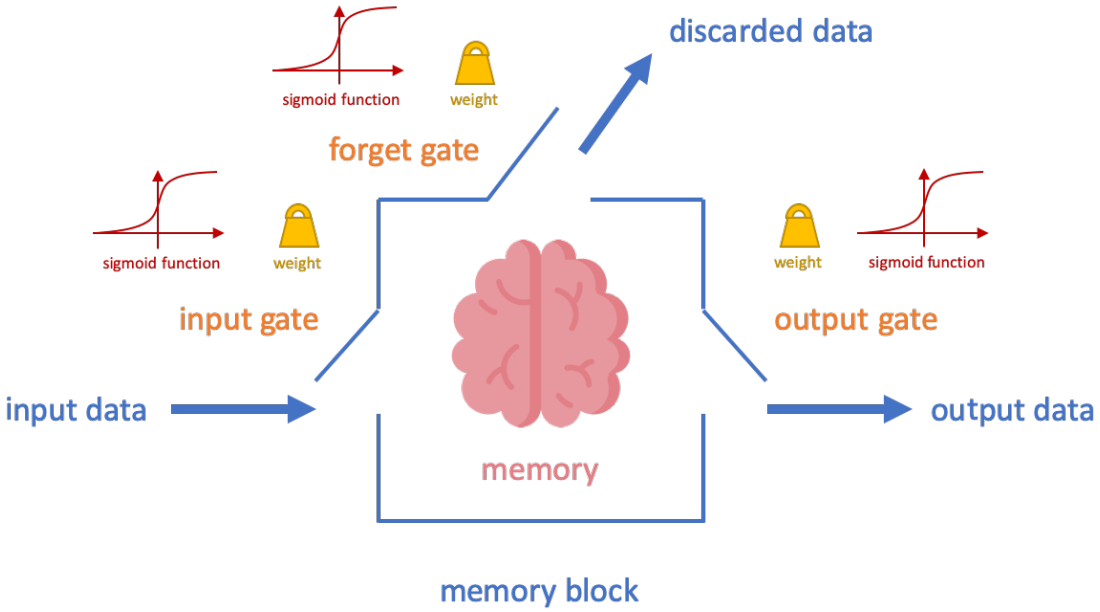


Figure 2-8 - LSTM memory block schema.

Concerning activation functions for LSTM networks, it is also possible to use the hyperbolic tangent function (tanh). But, while the sigmoid function returns values between 0 and 1 and, therefore, provides “yes” or “no” decisions, the hyperbolic tangent function returns values between -1 and 1 and, for that reason, makes “negative”, “neutral” and “positive” decisions (Skansi, 2018, p. 143). One other option as an activation function for LSTM networks is the rectifier function (relu) that has shown better results and also helps with the problem of unstable gradients (Brownlee, 2018b, pp. 37, 172).

3. CONCEPTUAL MODEL

3.1. CONTEXTUALIZATION

HVAC systems are accounted for most of the energy consumed in buildings. Because of inefficient operation and, sometimes, poor design, HVAC systems lead to energy waste. Although it is a possibility to reequip a building with a more efficient HVAC system, it requires a substantial amount of investments. On the other hand, improving control methods that operate existing HVAC systems to reduce energy consumption is much more cost-effective. Therefore, a lot of effort in being made on developing intelligent computational methods with the purpose of better controlling HVAC systems and, in that way, reducing energy usage in buildings (Patent No. US 10,047,968 B2, 2018).

Even though there is a concrete need for a solution that balances occupant comfort and energy consumption and that integrates computational intelligence, there are not a lot of options in the market concerning HVAC systems. In fact, in that aspect, the market is very limited or nearly existent. It is important to highlight that, in that context, computational intelligence comes from a mix of different recent technologies like IoT, Big Data, and Data Science.

3.2. INTERNET OF THINGS

The Internet of Things (IoT) is a concept that enables the instrumentation of all the *things* with sensors. It answers in meaningful ways the need to deal with data coming in from the sensors through the Internet. The Internet of Things makes it possible to measure, analyze, visualize, predict, and react to the environment around those *things* (Spann et al., 2018). A more technical definition of the Internet of Things would be that it is a network of objects, such as sensors and actuators, that can autonomously capture data and intelligently self-configure based on physical-world events, allowing these systems to become active participants in various public, commercial, scientific and personal processes (Spann et al., 2018).

The greatest benefit from IoT is bringing a huge amount of data so it is possible to make better decisions. There are a lot of use cases to apply IoT such as recognizing common trends in machinery imminent breakdown, offering special deals to store customers, and creating cars, buildings, and cities of the future (Spann et al., 2018). One of the most common use cases for IoT is fast and effective data analysis by combining with Artificial Intelligence / Machine Learning techniques. A survey done in 2019 by DZone.com with 575 tech professionals established that 44 percent of respondents said that plan to adopt new IoT-related technology within six months. From that, 65 percent respondents intend to adopt AI (Artificial Intelligence) / ML (Machine Learning) for IoT (Spann, Deschamps-Sonsino, Lawrence, Churilo, & Azzola, 2019).

Although IoT has huge potential and real interest from developers and tech and engineering companies, it still suffers a great deal from being a recent concept. Data security, data privacy and the lack of hardware and software standards to connect physical devices are top concerns and top barriers to wider adoption. In DZone.com 2019 IoT survey respondents declared, not exclusively, that security (78 percent), privacy (60 percent), and lack of standards (41 percent) are major issues of

concern in IoT. Besides all that, there is also a talent shortage that impacts the growth of IoT (Spann et al., 2018) (Spann et al., 2019).

3.3. BIG DATA

The term Big Data can be deceiving because it can make an impression of being just some more data. But, in fact, the Big Data concept is much more comprehensive. Big Data encompasses, not only a huge amount of data but also different types of data (structured and unstructured data) and different timings of data (fast-moving data or historical data). A possible definition for Big Data would be that it is a common term for data that must be coming into the system at a high velocity or with a large variation or at high volumes. Thus, Big Data is characterized by three V's: Volume, or how much data it is; Velocity, or how fast data is collected; and Variety, or how heterogeneous the dataset is (Bonér et al., 2018).

As the Internet of Things provides constant streams of data from hardware and Artificial Intelligence requires massive datasets to teach machines to think, Big Data is the in-between technology that bonds IoT and AI. Without Big Data distributed, fault-tolerant, and scalable storage solutions it would not be possible to handle all IoT collected data. Without Big Data parallelized and powerful processing engines, it would be unfeasible to train and to apply Machine Learning models.

In a sense, Big Data made the recent boom in AI adoption possible because it puts more computational power and huge amounts of data at the disposal of everyone in a cost-effective way. Data, not algorithms, is the key limiting factor to AI development (Groth, 2017). A lot of Neural Network algorithms, for instance, were developed in the late 50s and are known for a long time. As stated in the Google paper *The Unreasonable Effectiveness of Data*, "simple models and a lot of data trump more elaborate models based on less data" (Halevy, Norvig, & Pereira, 2009).

Even though Big Data has moved from hype to trend and from trend to the de-facto data processing and storage solution for advanced analytics projects, Big Data has challenges on its own. One of them is data quality. As the number of data sources increases, maintaining high data quality becomes a bigger task (Chala et al., 2019). Big Data adopters have to avoid the "garbage in, garbage out" concept (Zhang, Lipton, Li, & Smola, 2020, p. 21). Another challenge for Big Data initiatives is the shortage of skilled professionals. Developers, architects, and data scientists need to know an ecosystem of tools that exist in Big Data and how to apply best practices so the implemented solution can endure. Last, but not least, there is the challenge of dealing with real-time data. It is always important to assimilate new data as soon as possible, but streaming data analytics is notoriously difficult (Chala et al., 2019).

3.4. DATA SCIENCE

Data Science can be defined as a set of principles that support and guide the process of extracting information and knowledge from data (Igual & Seguí, 2017, p. 2,3). It also can be said that Data Science is a field of study that combines statistics, computer science, and business acumen to analyze data. The fact is that Data Science encompasses the whole spectrum of data processing and

analyzing, not just the algorithmic and statistical aspects. Data Science covers data integration, data engineering, data visualization, Data Mining, Business Intelligence, etc. (Granville, 2017). A successful data scientist is compelled to see a business problem from a data perspective.

Although Data Science is a comprehensive field, most of the attention of the scientific community and organizations are focused on Machine Learning, particularly on predictive analytics. Machine Learning is a kind of Artificial Intelligence that uses algorithms to extract patterns out of the data. These algorithms can learn from data as they are being trained. In this process of learning, they can improve their performance based on experience. In the end, a refined model is achieved that can be used to predict outcomes from unseen data based on the previous learning (Kirk, 2014, p. 2) (Bell, 2015, p. 2).

There are many use cases for Machine Learning in the energy and environment sectors, like using predictive analytics to pick the best location for wind farms (Hardesty, 2015) or to analyze pollution data and make predictions about air quality (IBM Research Editorial Staff, 2016). In reality, the convergence of Machine Learning and the Internet of Things leads to possibilities that both technologies alone could never achieve. For instance, Machine Learning allows IoT to provide AI-powered analytics platforms capable of continuous analytics, besides predictive and prescriptive analytics (Ruzicka, Lawrence, Chaudhri, Jacquet, & Smith, 2018).

Since Data Science popularity is a recent fact, some concerns are surrounding the matter. The most common issue is that some companies haven't yet realized the benefits of Data Science. There are a couple of reasons for that, like the lack of data quality, data scattered in silos, failure to define a problem to solve, and insufficient skilled professionals (Ruzicka et al., 2018). In fact, there is a shortage of talent because good professionals must be skilled in math, statistics, programming, database, and business knowledge.

3.5. Market Analysis and Big Players Initiatives

Although an advanced energy management system is based on computational intelligence from new technology, this is far from being the only reason why the market does not present many options. One other reason might be the difference in investments between Data Science projects oriented to HVAC systems and industrial systems. There are already applications in different industry sectors, such as turbine control, preventive maintenance, or other production processes. However, in the case of HVAC systems, besides relevant expectations, motivation, and scientific development on how to apply Machine Learning techniques (Smith & Lasch, 2016), there are not significant products available in the market.

The reason why Data Science projects for industrial systems are preferred when it comes to investments relies on the fact that much of the research is concentrated in intensive applications of capital and energy, like manufacturing and energy production. When HVAC systems are compared with this kind of application, it does not present the same amount of consuming nor the same potential return on investment. On the other side, the general dissemination of HVAC systems is much greater, which means that the sum of consumption of all HVAC systems reaches representative

marks. This makes valid any improvement in this field and is starting to catch the attention of big players from IT (Information Technology) and engineering.

Some companies are already aware of the benefits of applying computational intelligence to energy management systems through IoT, Big Data, and Data Science initiatives. One good example is Google that through an artificial intelligence research project called DeepMind uses an ensemble of deep Neural Networks to reduce the energy bill of its data centers (Evans & Gao, 2016). Another example comes from a partnership between IBM (Internet Business Machines) and international researchers to develop a solution to operate HVAC systems in three major commercial buildings located in Hong Kong through Machine Learning techniques (Vishwanath, 2018).

3.6. AMBIOSENSING CONCEPTUAL ARCHITECTURE

Since the market is absent in terms of energy management systems for buildings that integrate computational intelligence at competitive cost ready to work with HVAC or other systems, the Ambiosesing system is being developed to fill this gap.

Ambiosensing conceptual architecture is an initial reference to show the main components and functionalities of the system regardless of any technological specification. The proposed architecture is divided into layers to better designate responsibilities and identify dependencies. Each layer has a specific purpose and upper layers can use lower-layer services, but not the other way around.

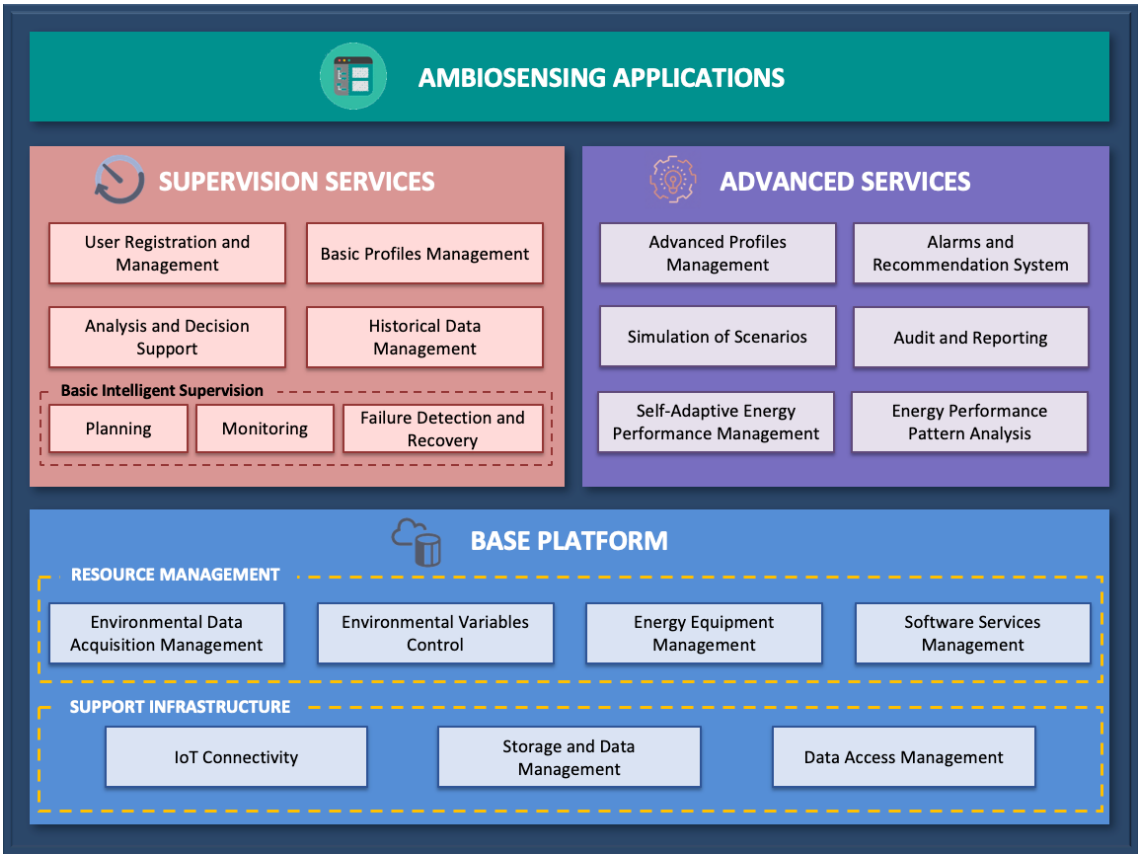


Figure 3-1 Ambiosensing Conceptual Architecture

3.6.1. Base Platform

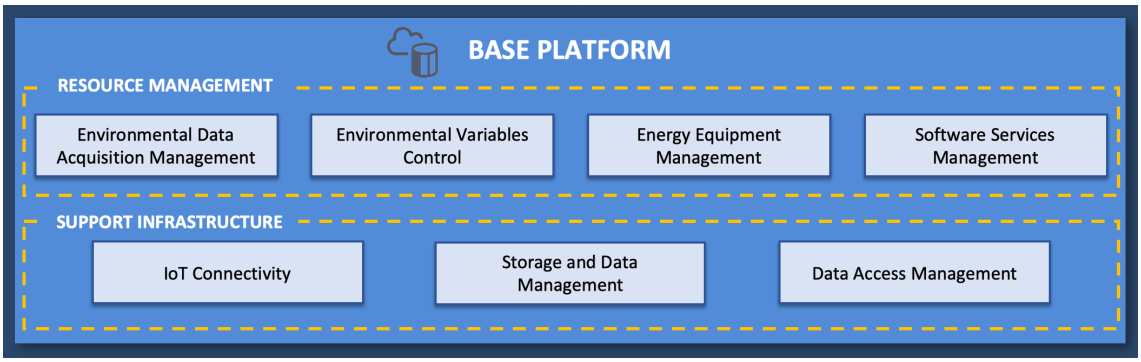


Figure 3-2 - Base Platform Layer

3.6.1.1. IoT Connectivity

The IoT Connectivity component allows devices to connect to the Ambiosensing system via Internet through IoT's most common communication protocols, like HTTP (Hypertext Transfer Protocol),

MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol). These devices could be sensors, actuators, microcontrollers, or single-board computers like Raspberry Pis.

3.6.1.2. Storage and Data Management

The Storage and Data Management module is not only responsible to support data storage, but also to enforce suitable access rights and proper privacy levels. It has to keep records of all telemetry data from environmental variables and all attribute data from energy profiles, customers, assets, devices, etc.

3.6.1.3. Data Access Management

The Data Access Management component provides control over user and data services accesses. Through the configuration of access policies, it grants correct use rights to the database.

3.6.1.4. Environmental Data Acquisition Management

The Environmental Data Acquisition Management module is responsible for configuring data collection for environment variables. Each environment variable reading can have a different requirement in terms of periodicity or measuring unit.

3.6.1.5. Environmental Variables Control

The Environmental Variables Control component is responsible for maintaining the relationship between inputs (environment variables) and devices. A device can have different types of sensors attached to it and even more than one sensor at a time. So, a single device can be able to register different environment variables.

3.6.1.6. Energy Equipment Management

The Energy Equipment Management unit is accountable for facilitating equipment configuration and administration. It provides an equipment catalog to register and organize all connected devices. It is important to notice that some devices, such as sensors and Raspberry Pi's, can be put together and can have an abstracted representation as a single device for the Ambiosensing system. The capability to manage all these kinds of equipment relationships is the responsibility of the Energy Equipment Management component.

3.6.1.7. Software Services Management

The Ambiosensing system provides a catalog of web services to make functionalities supported by lower-layer components available to upper layer modules. These web services also grant to Ambiosensing the ability to externalize functionalities to other systems. The Software Services Management module provides not only administration capabilities over these web services but also documentation like web services specification (request syntax and parameters, response messages, etc.).

3.6.2. Supervision Services

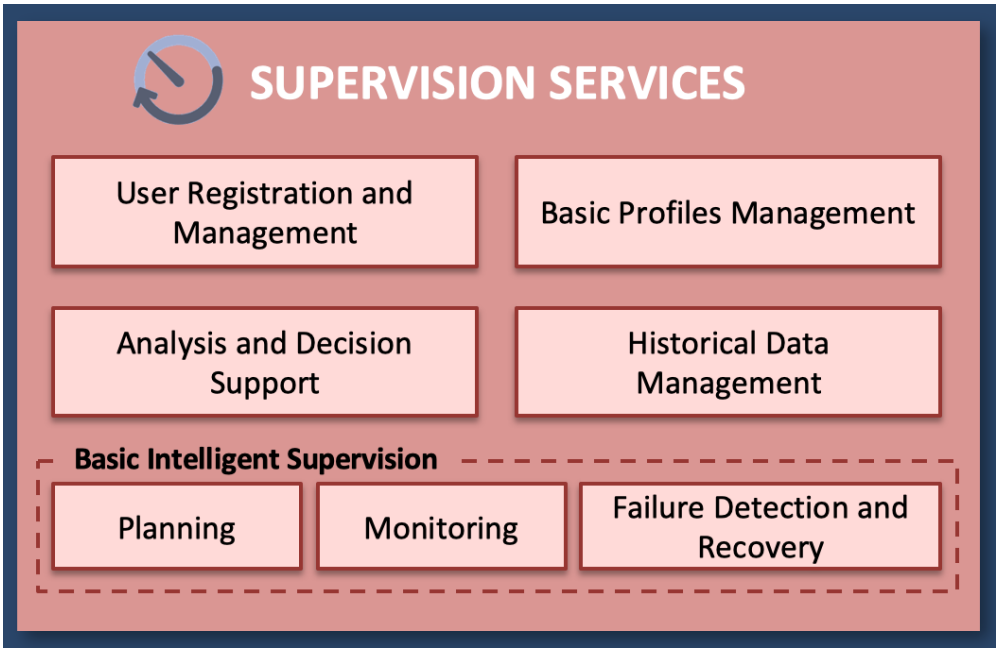


Figure 3-3 - Supervision Services Layer

3.6.2.1. User Registration and Management

The User Registration and Management component is responsible for user administration. It provides the Ambiosensing system with the capability to manage user profiles and preferences and also to group users accordingly with common aspects.

3.6.2.2. Basic Profiles Management

The Basic Profiles Management module allows administration over operating profiles of spaces. These profiles are created by an administrator user and can be based on specifics such as type of use

(e.g. office, store, warehouse, etc.), number of occupants, schedules, and other important information that might help to adjust the controlled environmental variables.

3.6.2.3. Analysis and Decision Support

The Analysis and Decision Support unit provides the needed mechanisms so the user can analyze the operation of a space and, in this way, can be able to make wise decisions that contribute to less energy consumption. All this is achieved through cross-referencing environmental variable measures and data of equipment in operation. The Analysis and Decision Support module also allows the user to keep track of the energy-saving provided by the system throughout its operation.

3.6.2.4. Historical Data Management

The Historical Data Management component is responsible for the administration of telemetry data collected and stored in the system. This time-series data can be from environmental variables readings or events that occurred in the system. No matter what source, historical data tends to be a large amount of data with time and needs to be managed accordingly to a lifecycle. This means that, when historical data gets huge, the system can "retire" the data by one of the two following strategies: data aggregation (e.g. old data stored by the minute is aggregated by the hour) or changing data storage type (e.g. withdraw data from database and store it on files).

3.6.2.5. Basic Intelligent Supervision

The Basic Intelligent Supervision unit is composed of three modules: Planning, Monitoring, and Failure Detection and Recovery. The Planning component allows the setting of an operational plan based on system parameterizations done by an administrator user. This plan must be put in place and have direct interaction with environmental variables, both in acquisition and intervention. The Monitoring module enables operational plan monitoring based on initially defined parameters, making it possible to detect deviations from the correct operation. The Failure Detection and Recovery component is responsible not only for detecting operations beyond a pre-defined acceptance threshold by also for providing recovery capabilities. Failure recoveries could be achieved by following a recovery plan initially set on the system along with the operational plan or by the intervention of a human operator with the appropriate user rights to do so.

3.6.3. Advanced Services

Figure 3-4 - Advanced Services Layer

3.6.3.1. Advanced Profiles Management

The Advanced Profiles Management module provides suggestions of the best operating profile of spaces considering the current condition in which the Ambiosensing system is working. The Advanced Profiles Management differentiates itself from Basic Profiles Management from Supervision Services in the sense that its suggestions are generated automatically by the system taking into consideration several parameters that may interfere with controlled environmental variables. In Basic Profiles Management it is up to an administrator user to create the operating profile of spaces. The Advanced Profiles Management can be set to run automatically or by user confirmation.

3.6.3.2. Alarms and Recommendation System

The Advanced and Recommendation System component is responsible for detecting situations that might raise an alarm based on data analysis from telemetry and event data. The Monitoring module from the Basic Intelligent Supervision unit works closely with the Alarms and Recommendation System component. Every improper operation is flagged and an alarm is sent to the most suitable user informing what happened and the recommended measures.

3.6.3.3. Simulation of Scenarios

The Simulation of Scenarios module provides the Ambiosensing system with the capacity of doing simulations on possible utilization scenarios of a space. This way it is possible to anticipate and elaborate operating strategies that are most suitable for the desired conditions. The Simulation of

Scenarios allows to assay the use of existing and desired equipment along with diverse operation conditions.

3.6.3.4. Audit and Reporting

The Audit and Reporting component allows administrator users to follow system operations and to track actions and configurations done by users in the system. Through the use of dashboards, it is possible to analyze the system's operational status. And, using reporting, it is possible to review the footprints that each user has left in the system, granting audit capabilities to Ambiosensing system. This auditing competence of the Ambiosensing system is configurable accordingly to the local legal regulations from where the system is being implanted.

3.6.3.5. Self-Adaptive Energy and Performance Management

The Self-Adaptive Energy and Performance Management module allows the system to identify specific circumstances in which its operation in no longer the most suitable one and it is still possible to adapt its operational profile to optimize performance. It is only possible to identify these situations through the use of intelligent services monitoring continuously current telemetry data. The system also uses historical telemetry data to define a performance baseline and to evaluate whether an operational profile adaptation will have a real advantage or not. In cases where a change in the operational profile is beneficial, it may occur automatically or by the consent of a properly granted user.

3.6.3.6. Energy Performance Pattern Analysis

The Energy Performance Pattern Analysis component is responsible for analyzing system operation and detect operation patterns directly correlated with energy performance. To be able to do so, the Ambiosensing system evaluates environmental variables, equipment setups, spaces attributes, and other important variables that may interfere in the balance of occupant well-being and energy consumption. Based on this assessment, the system identifies and classifies energy performance patterns and corresponding operational profiles.

3.6.4. Ambiosensing Applications

The top layer in Ambiosensing Conceptual Architecture is Ambiosensing Applications. This component is composed of interaction applications designed for end-users that allow execution of Ambiosensing system use cases such as:

- Set up environmental variables;
- Set up sensors;
- Set up actuators;
- Set up building plants (spaces);

- Create energy profiles;
- Manage energy profiles;
- Monitor environmental variables;
- Monitor sensors;
- Monitor actuators;
- Analyze energy behavior;
- Visualize alarms;
- Query historical data;
- Simulate scenarios;
- Predict scenarios;
- Generate audit reports;
- Generate energy consumption dashboards;
- Register user preferences;
- Manage users.

3.7. DATA COLLECTION AND AMBIOSENSING OVERVIEW

One of the greatest challenges of the Ambiosensing project is data collection. Hopefully, recent technological developments in IoT and Big Data contributed to advances in multiple computational tasks needed to device communication and coordination and data processing and storage. Definitely, the emergence of IoT platforms and Big Data frameworks for dealing with massive amounts of data and computation are among these recent technological breakthroughs.

The Ambiosesing system consists, basically, of two types of hardware: the Ambiosensing devices and the Ambiosensing central unit. Ambiosensing devices are scattered around the environment in which they are operating and can have up to two roles: sensor and actuator. Ambiosensing central unit is a component of the greatest importance. As the name implies, it is a fundamental module where all telemetry data is consolidated for storage, processing, and analysis.

Ambiosensing devices have as their main goal the capturing of electric signals from sensors attached to it and the conversion of these signals to digital values accordingly to pre-defined measurement units. For instance, in case of temperature to Celsius degree, in case of relative humidity to percentage and in case of luminosity to lux (lux is the unit from the International System of Units to measure luminous flux per unit area) ("IEC 60050 - International Electrotechnical Vocabulary - Details for IEV number 845-01-52: 'lux,'" n.d.). Ambiosensing devices have some data aggregation and storage capacity especially useful in case of network failure. However, its processing power is not suitable for the amount of data to be handled and, as such, it is expected that Ambiosensing devices send telemetry data to Ambiosensing central unit. Figure 3-5 shows a schema of an Ambiosensing device with three sensors attached to it: luminosity, humidity, and temperature sensors. An Ambiosensing device is composed also of a controller element, which is, usually, a Raspberry Pi single-board computer.

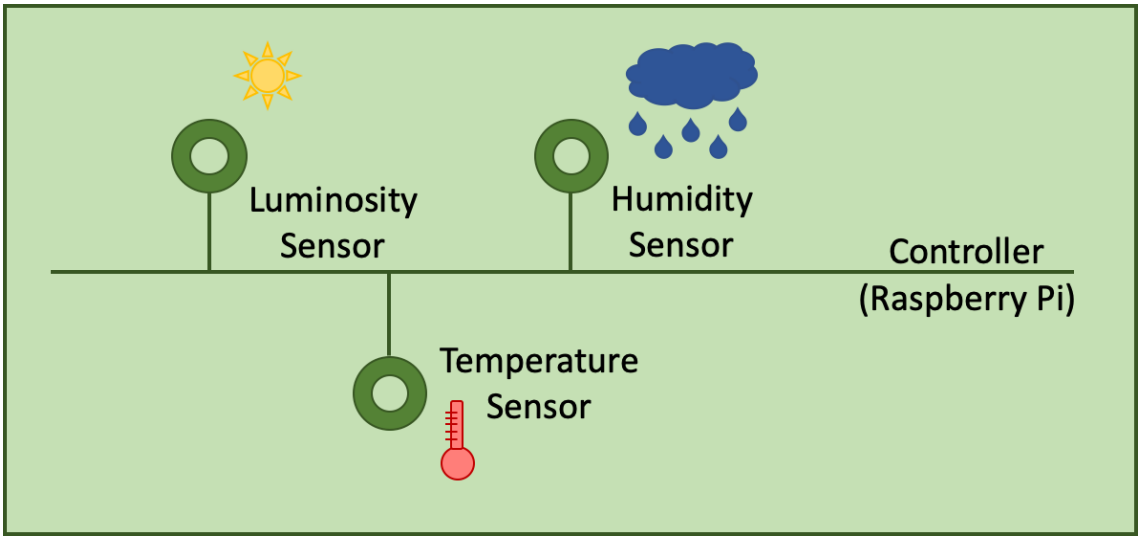


Figure 3-5 - Ambiosensing Device

Ambiosensing central unit is not only responsible for collecting telemetry data from all devices that are present in a monitored space. Ambiosensing central unit is also in charge of dispatching commands to control equipment that affects the controlled environment, such as HVAC systems. Since Ambiosensing central unit must perform many IoT functions, such as collect and store telemetry data and provision, monitor and control IoT devices, it makes sense that part of the Ambiosensing central unit is composed of an IoT platform. However, as the Ambiosensing central unit has goals that exceed the IoT platform's capabilities, it cannot be completely based on an IoT platform. An IoT platform may cover all modules from the Base Platform layer of the Conceptual Architecture and even a couple of components from Supervision Services and Advanced Services layers, such as User Registration and Management and Audit and Reporting, respectively. Figure 3-6 presents Ambiosensing central unit and Base Platform services and some top layers services as part of an IoT Platform.

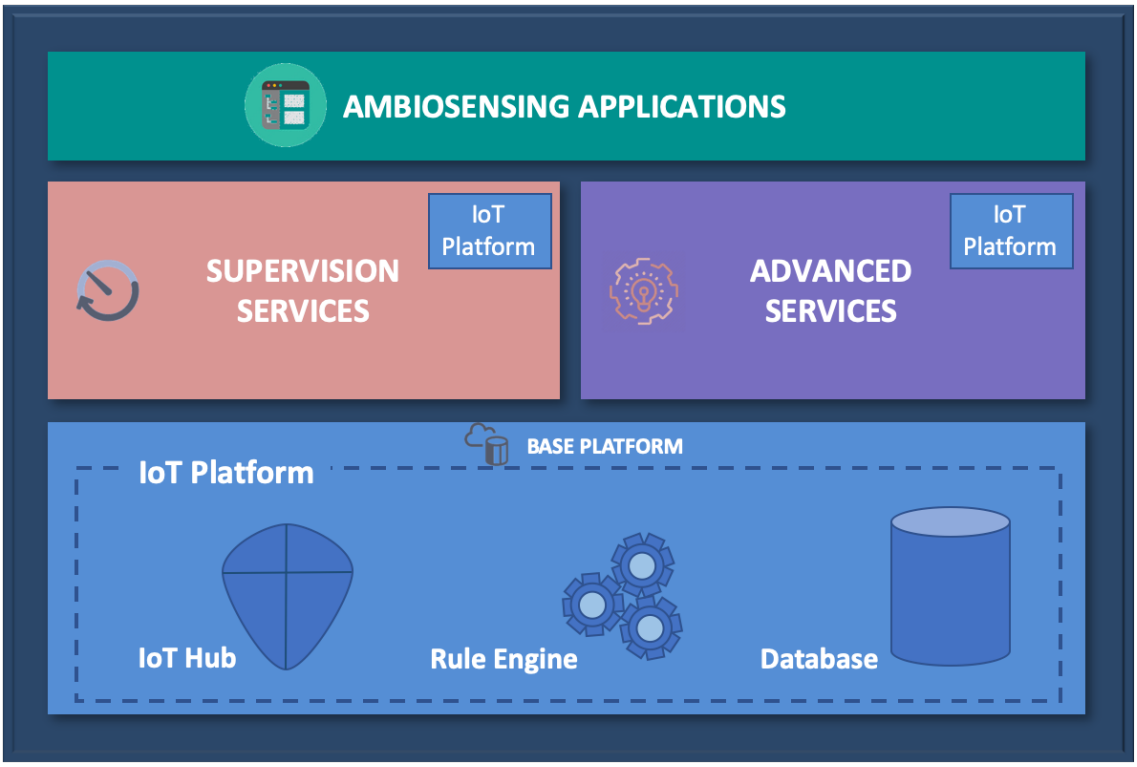


Figure 3-6 - Ambiosensing Central Unit

Ambiosensing central unit may communicate with equipment directly if they have this capacity or through an Ambiosensing device, which, in this case, would be performing as an actuator. The Ambiosensing device in communication with the equipment may be the same one that is doing sensor readings or an entirely different Ambiosensing device deployed only for that specific task. Figure 3-7 presents an Ambiosensing overview and demonstrates how these two scenarios work. Option 1 shows Ambiosensing Device 01 performing as a sensor and an actuator, being responsible not only for telemetry data collection but also for commanding the HVAC system through RPCs (Remote Procedure Calls) received from Ambiosensing central unit. Option 2 presents Ambiosensing Device 01 operating only as a sensor, doing environmental variables readings. Meanwhile, Ambiosensing Device 02 is responsible for receiving commands from Ambiosensing central unit and for passing these operational instructions through RPCs to the HVAC system.

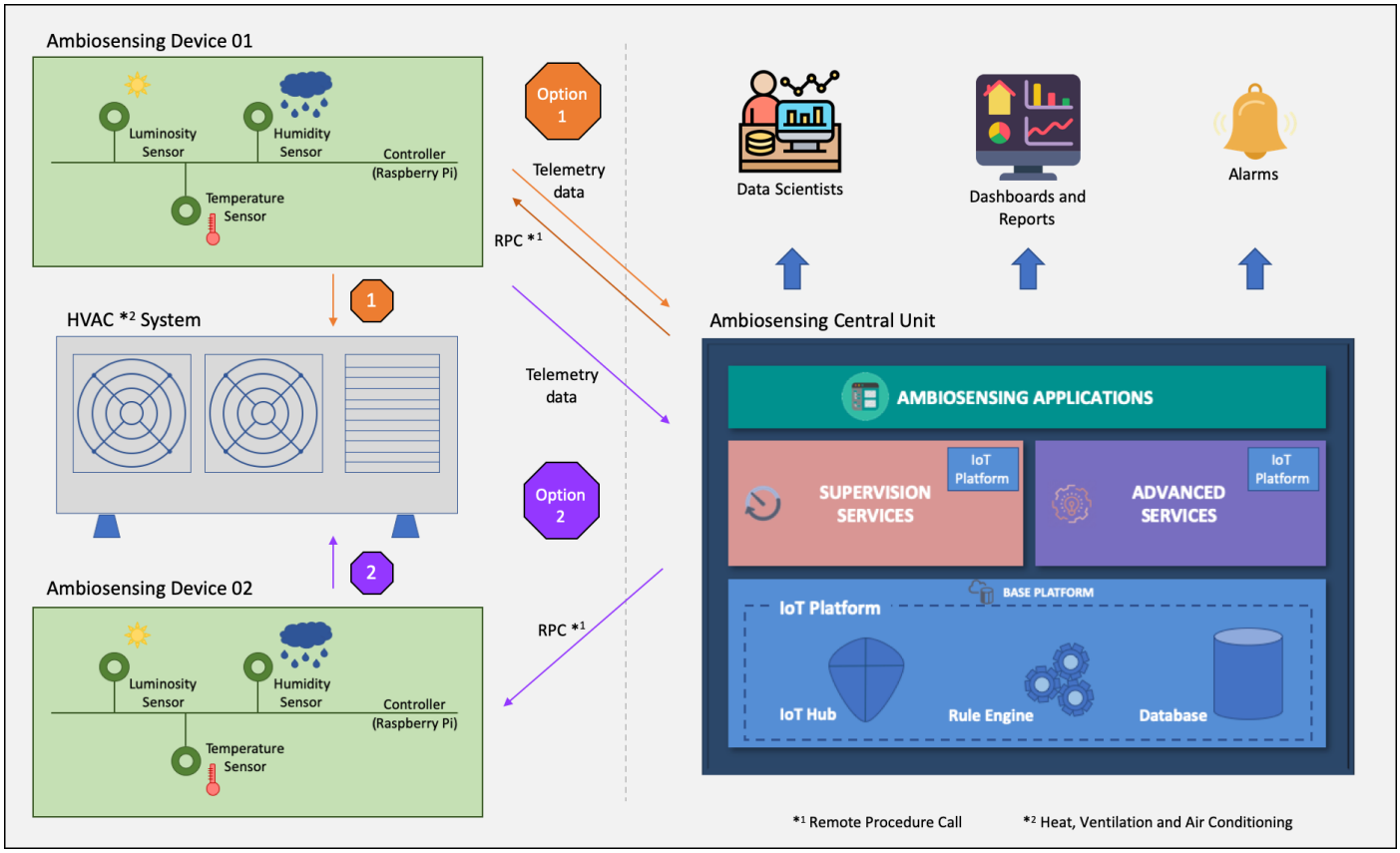


Figure 3-7 - Ambiosensing Overview

4. EXPERIMENTAL SETTINGS

4.1. DEVICE AND SENSORS

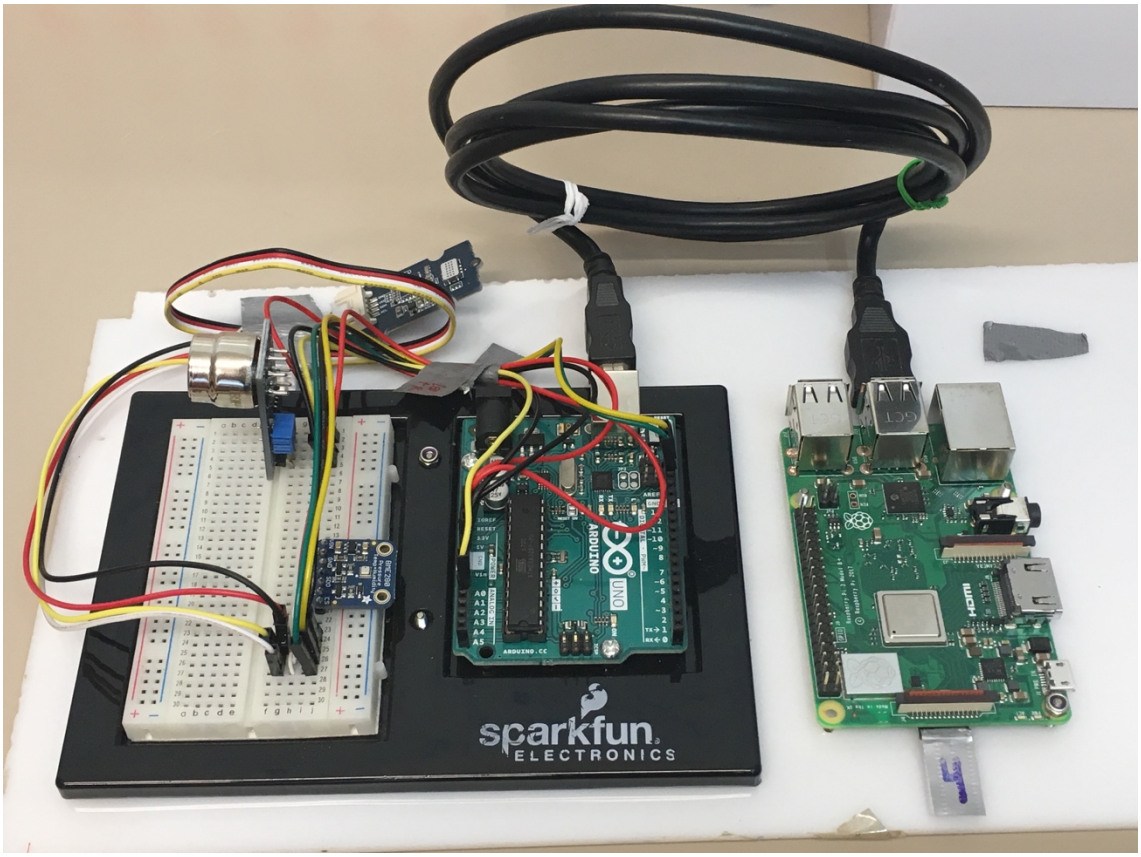


Figure 4-1 - Ambiosensing device proof of concept.

Ambiosensing device components:

- Raspberry Pi 3 B+
- aLaMode (an Arduino compatible board for Raspberry Pis)
- microSD memory card 64 GB
- Breadboard
- 1 SHT31-D sensor (temperature, and humidity)
- 1 BME-280 sensor (pressure, temperature, and humidity)
- 1 Grove Multichannel Gas Sensor

The SHT31-D is a temperature and humidity sensor. It has an I2C (Inter-Integrated Circuit) interface and is very accurate with ±2% relative humidity and ±0.3°C accuracy for most use cases. It also has a PTFE (Polytetrafluoroethylene) filter that helps it to stay clean allowing humidity measurements to work longer ("Adafruit Sensirion SHT31-D (Temperature and Humidity Sensor)," n.d.).

The BME-280 is an environmental sensor with temperature, barometric pressure, and humidity readings. It has an I2C and an SPI (Serial Peripheral Interface) interface. It is very precise measuring humidity with ±3% accuracy, barometric pressure with ±1 hPa absolute accuracy, and temperature

with ±1.0°C accuracy. Because pressure changes with altitude and pressure measurements are so exact, it is possible to use it as an altimeter with ±1 meter of accuracy ("Humidity Sensor BME280," n.d.).

The Grove Multichannel Gas Sensor can detect many dangerous gases like Ethanol (C2H5OH), Propane (C3H8), Iso-butane (C4H10), Methane (CH4), Carbon Monoxide (CO), Carbon Dioxide (CO2), Hydrogen (H2), Ammonia (NH3), and Nitrogen Dioxide (NO2). Because it is multi-channel, it can measure different gases simultaneously. It has an I2C interface for easier connectivity.

4.2. Data Acquisition Mechanism

In the Ambiosensing system, the data collected by the device sensors, the telemetry data, is sent to the chosen open-source IoT platform ThingsBoard using communication protocols such as MQTT, CoAP, and HTTP. To send telemetry data to ThingsBoard, a program running on the devices mainly do two things:

1. Invoke the telemetry data upload API (Application Programming Interface) provided by ThingsBoard for one of the available protocols (MQTT, CoAP or HTTP);
2. Submit data upload requests through the API to ThingsBoard containing messages formatted in a predefined timestamp/key/value scheme.

In a more detailed way, the data acquisition mechanism proceeds as follows:

- A device has, for instance, temperature and humidity sensors attached to it. A device could be a stand-alone Raspberry Pi or a composition between a Raspberry Pie and an Arduino depending on the sensors interface requirements;
- The Raspberry Pi has a Linux-based Operating System (OS) capable of running Python written programs;
- A data collector program, written in Python, runs at OS startup and, after that, periodically, for example at every minute, according to a scheduler;
- The data collector program captures temperature and humidity sensor data at the predefined interval;
- The data collector program, then, mounts messages in a timestamp/key/value format to send the moment, the metric (temperature or humidity, in this example) and the value of the metric to the IoT platform;
- Later, the data collector program establishes communication to the IoT platform through its data upload API for the chosen protocol, e.g. MQTT;
- The IoT platform authorizes the device communication after validation of the access key provided through data upload API. The device must have been previously configured in the IoT platform for the authorization work without problems;
- Finally, the data collector program sends the telemetry data to the IoT platform through an API post command.

At first, the data collector program was only capturing sensor data and sending it straight to the IoT platform. Later, some problems happened creating some data gaps. The main problem was connectivity failure, specifically, Internet or WiFi network issues. Because of that, the data collector program was improved and became fault-tolerant to connectivity issues, at least, for a given period.

The idea implemented is simple. Between capturing and sending sensor data to the IoT platform, data is now stored locally in the device in a light version of a database running on the Linux-based OS. This database uses the Raspberry Pi SD (Secure Digital) card as storage and saves data in JSON (JavaScript Object Notation) files. After some tests, it proved capable of backing up a reasonable amount of data, more than the equivalent of one month of data for a device with multiple sensors and measures. Only after confirmation that data has been saved locally, the data collector program tries to send data to the IoT platform. If everything is alright, the data is sent to the IoT platform and the local backup is deleted to free space inside the device. If not, the data is accumulated locally until the connection is reestablished. Once the connection is up again, the very next attempt to send data to the IoT platform is responsible for uploading as soon as possible all accumulated data to the IoT platform.

Below, a diagram of the data collector program. In this diagram, the activities described above are organized by layers. This diagram also shows the JSON files used as local storage and the parameter file used to configure each device based on its sensor setup.

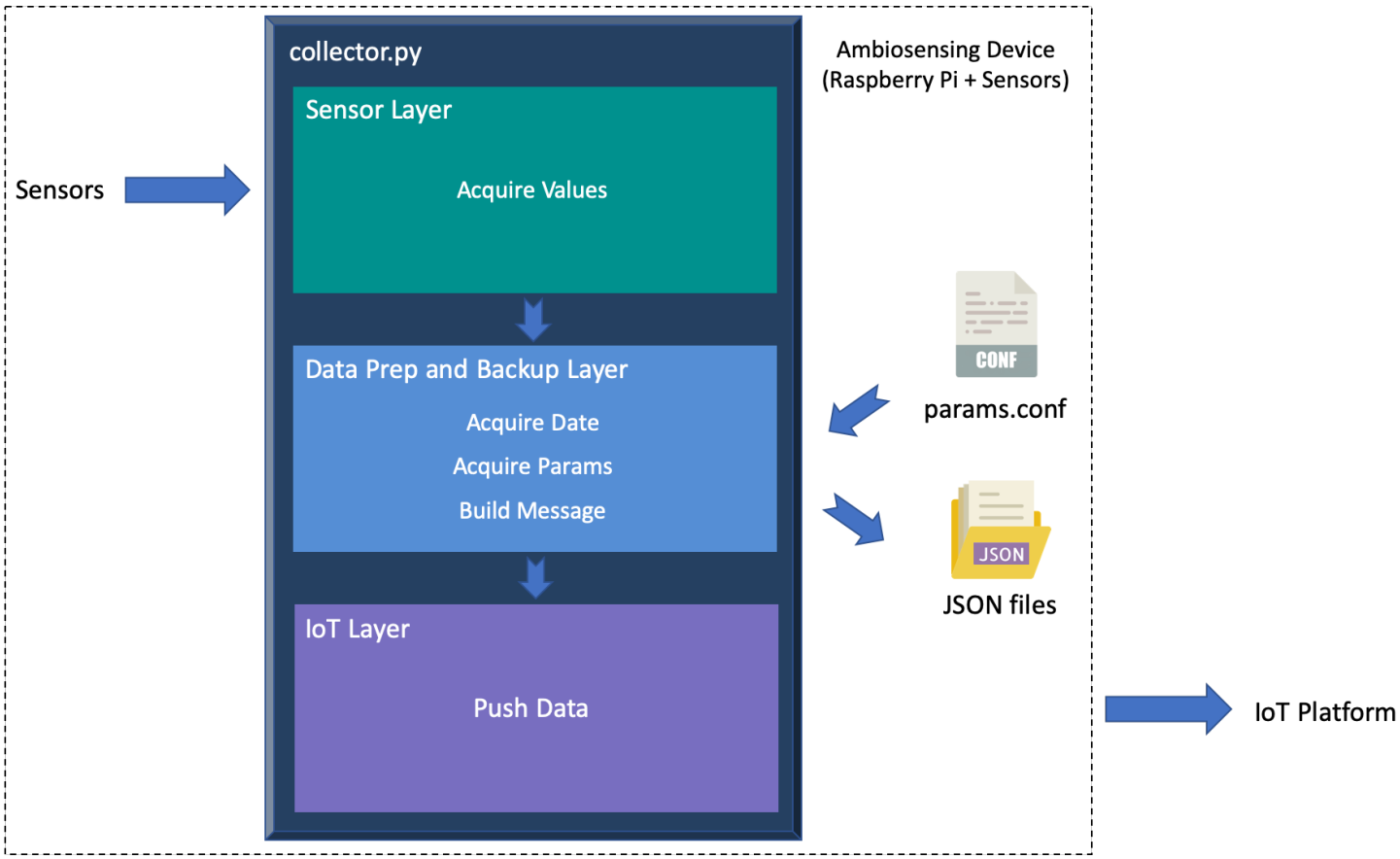


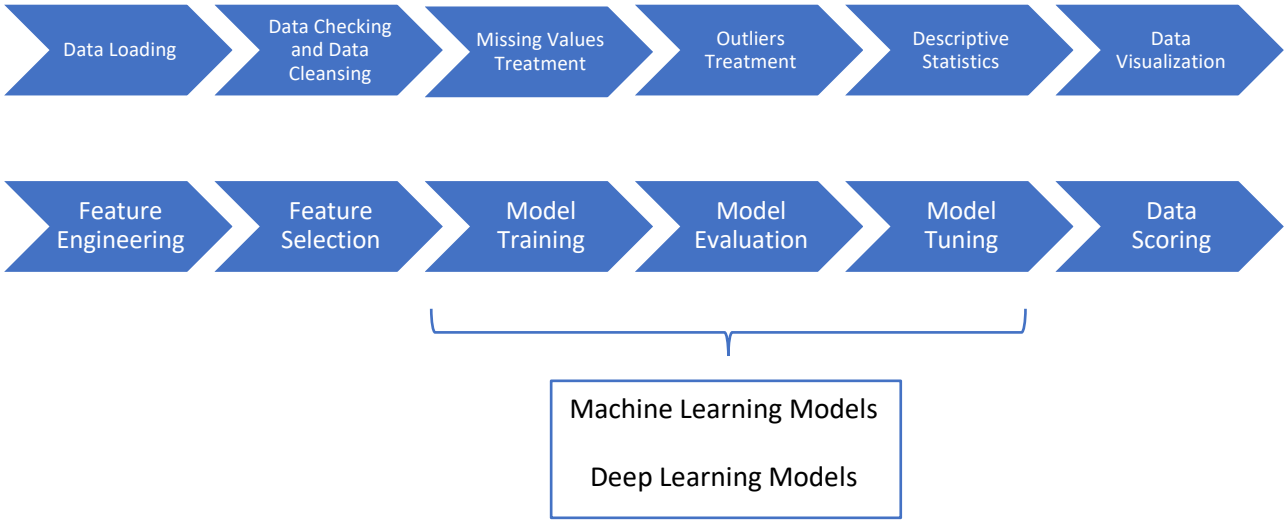
Figure 4-2 - Data Collector Program Diagram.

After receiving the telemetry data sent by the devices, the IoT platform ThingsBoard should store the data in a database capable of persisting time series data. ThingsBoard has two options as a production database: Cassandra as a NoSQL (Not Only SQL) choice and PostgreSQL as a SQL

(Structured Query Language) alternative. Since Cassandra database can handle better a large volume of data, it was chosen to be the IoT platform internal database.

4.3. Experimental Framework

After collecting raw data from sensors with the help of the IoT platform ThingsBoard and storing it on a Cassandra database, the following framework was used to apply Data Science techniques:



This framework was used with Deep Learning models and with other Machine Learning models to establish a comparison between them. LSTM RNNs were chosen as Deep Learning models because of their good performance with sequence problems like time series forecasting. Different types of LSTMs were used in an attempt to get the best model possible: univariate LSTM (to set a baseline), multivariate LSTM with multiple inputs, and multivariate LSTM with parallel series. About other Machine Learning models, Linear Regression, LASSO, K-Nearest Neighbors, Decision Trees, Random Forest and Gradient Boosting Machines were chosen to be compared with previously cited Deep Learning models.

4.3.1. Data Loading

The Data Loading step has the objective of capturing raw data from the IoT platform database and initiate data preparation for Data Science practice.

Initially, the dataset was exported from the Cassandra database into a CSV (Comma Separated Value) file. Later, this file was uploaded to a Databricks/Spark environment, where all Data Science process was conducted. The Databricks/Spark environment was set based on Python 3 and additional libraries such as Holidays, Keras, and Tensorflow. Python's library Pandas ("Pandas Documentation," n.d.) was extensively used to represent data structures, manipulate and analyze data, Python's

library Numpy (“NumPy Manual,” n.d.) was applied to do calculations on the dataset and Python’s library scikit-learn (“Scikit-learn Documentation,” n.d.) provided most of the Machine Learning algorithms.

Originally, the raw data were presented as the following dataset:

	entity_type	entity_id	key	partition	ts	bool_v	dbl_v	long_v	str_v
0	DEVICE	507a5780-d281-11e9-8d16-d75afdb8a7ce	Humidity	1569888000000	1571753587000	None	48.364238	NaN	None
1	DEVICE	507a5780-d281-11e9-8d16-d75afdb8a7ce	Humidity	1569888000000	1571753850000	None	48.098830	NaN	None
2	DEVICE	507a5780-d281-11e9-8d16-d75afdb8a7ce	Humidity	1569888000000	1571754303000	None	48.160375	NaN	None
3	DEVICE	507a5780-d281-11e9-8d16-d75afdb8a7ce	Humidity	1569888000000	1571754363000	None	48.024900	NaN	None
4	DEVICE	507a5780-d281-11e9-8d16-d75afdb8a7ce	Humidity	1569888000000	1571754423000	None	48.310874	NaN	None

Figure 4-3 - Original dataset.

The following data transformations were done in data loading as an initial data preparation:

- Date converted from Linux timestamp to date;
- Key and Value columns were pivoted to denormalize the dataset.

After initial data transformations, the dataset was presented as follows:

key ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure
2019-07-23 23:10:05	2.40	42.92	44.27	1890.64	5.90	399.0	23.6	74.48	1.15	67.3	0.12	0.58	1004.0
2019-07-23 23:16:11	1.74	33.10	35.52	762.61	4.62	399.0	23.8	74.84	0.79	66.8	0.10	0.64	1004.0
2019-07-23 23:17:09	1.81	31.63	34.17	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.64	1004.0
2019-07-23 23:18:09	1.81	30.19	32.85	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.65	1004.0
2019-07-23 23:19:09	1.85	30.19	32.85	915.53	4.85	399.0	23.8	74.84	0.85	67.5	0.10	0.66	1004.0

Figure 4-4 - Dataset after initial data preparation.

Below, a brief explanation of each dataset variable:

- C2H5OH – Ethanol gas measure in ppm;
- C3H8 – Propane gas measure in ppm;
- C4H10 – Iso-butane gas measure in ppm;
- CH4 – Methane gas measure in ppm;
- CO – Carbon Monoxide gas measure in ppm;
- CO2 – Carbon Dioxide gas measure in ppm;
- Celsius – Temperature in Celsius degree;
- Fahrenheit – Temperature in Fahrenheit degree;
- H2 – Hydrogen gas measure in ppm;
- Humidity – Relative humidity in % (percentage);
- NH3 – Ammonia gas measure in ppm;
- NO2 – Nitrogen Dioxide gas measure in ppm;
- Pressure – Atmospheric pressure in the equivalent units hPa (hectopascal) or mbar (millibar).

The dataset has data from July 23rd of 2019 to Jan 23rd of 2020, resulting in a total of 228.307 observations.

4.3.2. Data Checking and Data Cleansing

The Data Checking and Data Cleansing step has the objective to inspect the dataset and remove data collected inaccurately.

By simply checking the average and the count by day of some features, like for instance Celsius, that represents indoor ambient temperature, it was possible to find some possible gaps in data collection. In the first months of the Ambiosensing project, the collector program developed in Python that goes inside the devices with all the logic for collecting and sending data to the IoT platform Thingsboard lacked local storing capability and, because of that, it didn't have a fault-tolerant mechanism implemented. Later in the project, it was corrected, but, by then, the dataset already had two gaps of data for some days that had to be treated in the Missing Values Treatment step of the Data Science framework.

There were two gaps in the dataset. One of seven days from August 16th to August 23rd of 2019 and another one of seventeen days from December 29th of 2019 to January 15th of 2020.

Another issue also contributed to inaccurate data collection. The Grove Multichannel Gas sensor from the Ambiosensing Device has a unique characteristic. It needs some time to warm up. Until then, readings are very erratic. Because of that initial readings of the dataset and readings right after the two data gaps were inspected carefully. July 23rd, the initial date of data collection, had very few observations since the device was turned on late at night. July 24th had a lot of discrepancies because the sensor was still warming up. Because of that, data from July 23rd and 24th were removed from the dataset. Since data capture was resumed from data gaps on August 23rd for the first one and on January 15th for the second one, those days were also removed from the dataset to avoid completely inaccurate data.

4.3.3. Missing Values Treatment

The Missing Values Treatment step aims to handle missing values by filling them with data. Because of the unexpected two data collection interruptions explained in the Data Checking and Data Cleansing step, the dataset required a consistent data imputation. The Missing Values Treatment step was divided into three parts: dataset preparation, a study of the most appropriate data imputation method for each feature, and data imputation itself.

First, the dataset needed to be prepared for data imputation. New records were created for the two missing time frames with null values by the minute, almost as it should be if data were captured correctly. Besides that, all other variables, like the date-time features further explained in the Feature Engineering step, were also created. In the end, the observations generated were just like the ones captured from sensors, with the same data types, in the right order, but with null values.

	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period
0	2019-07-25 00:00:09	1.55	12.64	15.68	553.83	4.23	399.0	28.3	82.94	0.69	57.4	0.05	1.06	1002.0	2019	7	25	0	0	9	Thursday	2019-07-25	00:00:09	206	30	3	P1	2019-07-25-P1
1	2019-07-25 00:01:09	1.24	9.66	12.48	294.25	3.57	399.0	28.2	82.76	0.53	57.6	0.05	1.14	1001.0	2019	7	25	0	1	9	Thursday	2019-07-25	00:01:09	206	30	3	P1	2019-07-25-P1
2	2019-07-25 00:02:08	3.31	34.63	36.90	4692.64	7.54	399.0	28.2	82.76	1.67	58.7	0.11	0.78	1002.0	2019	7	25	0	2	8	Thursday	2019-07-25	00:02:08	206	30	3	P1	2019-07-25-P1
3	2019-07-25 00:03:08	2.44	23.65	26.70	1991.27	5.98	399.0	28.2	82.76	1.17	58.4	0.08	0.87	1001.0	2019	7	25	0	3	8	Thursday	2019-07-25	00:03:08	206	30	3	P1	2019-07-25-P1
4	2019-07-25 00:04:09	2.22	22.46	25.55	1528.47	5.57	399.0	28.2	82.76	1.05	58.9	0.08	0.88	1001.0	2019	7	25	0	4	9	Thursday	2019-07-25	00:04:09	206	30	3	P1	2019-07-25-P1

Figure 4-5 - Data captured from the sensors.

	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period
0	2019-08-16 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019	8	16	0	0	0	Friday	2019-08-16	00:00:00	228	33	3	P1	2019-08-16-P1
1	2019-08-16 00:01:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019	8	16	0	1	0	Friday	2019-08-16	00:01:00	228	33	3	P1	2019-08-16-P1
2	2019-08-16 00:02:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019	8	16	0	2	0	Friday	2019-08-16	00:02:00	228	33	3	P1	2019-08-16-P1
3	2019-08-16 00:03:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019	8	16	0	3	0	Friday	2019-08-16	00:03:00	228	33	3	P1	2019-08-16-P1
4	2019-08-16 00:04:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019	8	16	0	4	0	Friday	2019-08-16	00:04:00	228	33	3	P1	2019-08-16-P1

Figure 4-6 - Data generated for data imputation.

After dataset preparation, some data imputation methods for time series were tested. A section of the dataset without missing values was extracted and transformed to perform the tests. The idea behind this evaluation is to hide the known values, execute the data imputation methods on fake missing values, and then compare the results of the different methods with the original non-null values. In the end, the most effective method for data imputation in time series should be determined for each feature.

Steps taken to test different data imputation methods:

1. The dataset was filtered. Only a sample of the dataset was selected to be tested. A sample of 50.000 rows was taken;
2. The dataset sample was split into three parts to create the missing values observations in the middle one (out of the 50.000 rows, the first and the last 20.000 were not transformed, only the 10.000 in between were transformed to null values in every feature);

	index	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	C2H5OH_test
141520	130000	2019-10-31 07:42:21	0.63	2.23	3.59	44.07	2.14	396.0	23.6	74.48	0.24	60.8	0.02	8.4	1015.0	2019	10	31	7	42	21	Thursday	2019-10-31	07:42:21	304	44	4	P2	2019-10-31-P2	NaN
141521	130001	2019-10-31 07:43:21	0.63	2.23	3.59	44.07	2.14	404.0	23.6	74.48	0.24	60.6	0.02	8.5	1016.0	2019	10	31	7	43	21	Thursday	2019-10-31	07:43:21	304	44	4	P2	2019-10-31-P2	NaN
141522	130002	2019-10-31 07:44:18	0.63	2.23	3.59	44.07	2.14	399.0	23.6	74.48	0.24	61.0	0.02	8.5	1016.0	2019	10	31	7	44	18	Thursday	2019-10-31	07:44:18	304	44	4	P2	2019-10-31-P2	NaN
141523	130003	2019-10-31 07:45:22	0.63	2.23	3.59	44.07	2.14	404.0	23.6	74.48	0.24	60.5	0.02	8.5	1016.0	2019	10	31	7	45	22	Thursday	2019-10-31	07:45:22	304	44	4	P2	2019-10-31-P2	NaN
141524	130004	2019-10-31 07:46:22	0.63	2.23	3.59	44.07	2.14	404.0	23.6	74.48	0.24	60.7	0.02	8.5	1016.0	2019	10	31	7	46	22	Thursday	2019-10-31	07:46:22	304	44	4	P2	2019-10-31-P2	NaN

C2H5OH_test	C3H8_test	C4H10_test	CH4_test	CO_test	CO2_test	Celsius_test	Fahrenheit_test	H2_test	Humidity_test	NH3_test	NO2_test	Pressure_test
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 4-7 - Dataset with original columns and null columns. Five rows out of the 10.000 transformed.

	index	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	C2H5OH_test
121520	110000	2019-10-17 09:53:08	0.44	0.79	1.49	16.5	1.64	396.0	24.9	76.82	0.16	62.5	0.01	10.96	1009.0	2019	10	17	9	53	8	Thursday	2019-10-17	09:53:08	290	42	4	P3	2019-10-17-P3	0.44
121521	110001	2019-10-17 09:54:08	0.44	0.79	1.49	16.5	1.64	404.0	24.9	76.82	0.16	62.7	0.01	10.96	1009.0	2019	10	17	9	54	8	Thursday	2019-10-17	09:54:08	290	42	4	P3	2019-10-17-P3	0.44
121522	110002	2019-10-17 09:55:08	0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	63.0	0.01	10.96	1009.0	2019	10	17	9	55	8	Thursday	2019-10-17	09:55:08	290	42	4	P3	2019-10-17-P3	0.44
121523	110003	2019-10-17 09:56:09	0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	63.0	0.01	10.79	1009.0	2019	10	17	9	56	9	Thursday	2019-10-17	09:56:09	290	42	4	P3	2019-10-17-P3	0.44
121524	110004	2019-10-17 09:57:08	0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	62.7	0.01	10.79	1009.0	2019	10	17	9	57	8	Thursday	2019-10-17	09:57:08	290	42	4	P3	2019-10-17-P3	0.44

C2H5OH_test	C3H8_test	C4H10_test	CH4_test	CO_test	CO2_test	Celsius_test	Fahrenheit_test	H2_test	Humidity_test	NH3_test	NO2_test	Pressure_test
0.44	0.79	1.49	16.5	1.64	396.0	24.9	76.82	0.16	62.5	0.01	10.96	1009.0
0.44	0.79	1.49	16.5	1.64	404.0	24.9	76.82	0.16	62.7	0.01	10.96	1009.0
0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	63.0	0.01	10.96	1009.0
0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	63.0	0.01	10.79	1009.0
0.44	0.79	1.49	16.5	1.64	399.0	24.9	76.82	0.16	62.7	0.01	10.79	1009.0

Figure 4-8 - Dataset with original columns and replicated columns unaltered. Five rows out of the 40.000 not transformed.

3. The dataset sample was joined back together to create the whole test sample;

   By this point, for every feature, there were two columns: one with the original value and a replica "_test" column with missing value if previously transformed or with the original value if not.

4. For every feature, the original column was taken as reference and the replica "_test" column was set as the target;
5. Data filled using mean and median;
6. Data imputed using the rolling average;
7. Data imputed using the rolling median;
8. Data imputed using interpolation with different methods: linear, time, quadratic, cubic, slinear, akima, polynomial of order 5 and 7, spline of order 3, 4 and 5;
9. The results were scored using scikit-learn metric R2 score and a table was created with the top result by feature.

Below, an example of a feature before and after data imputation (sample dataset):

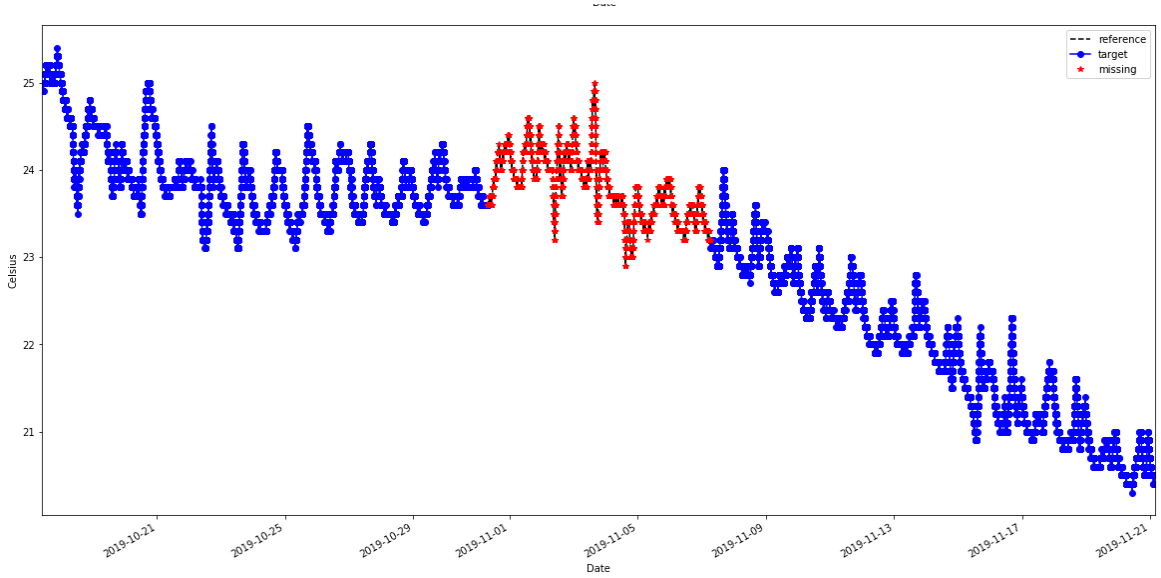


Figure 4-9- Sample dataset of Celsius feature before data imputation.

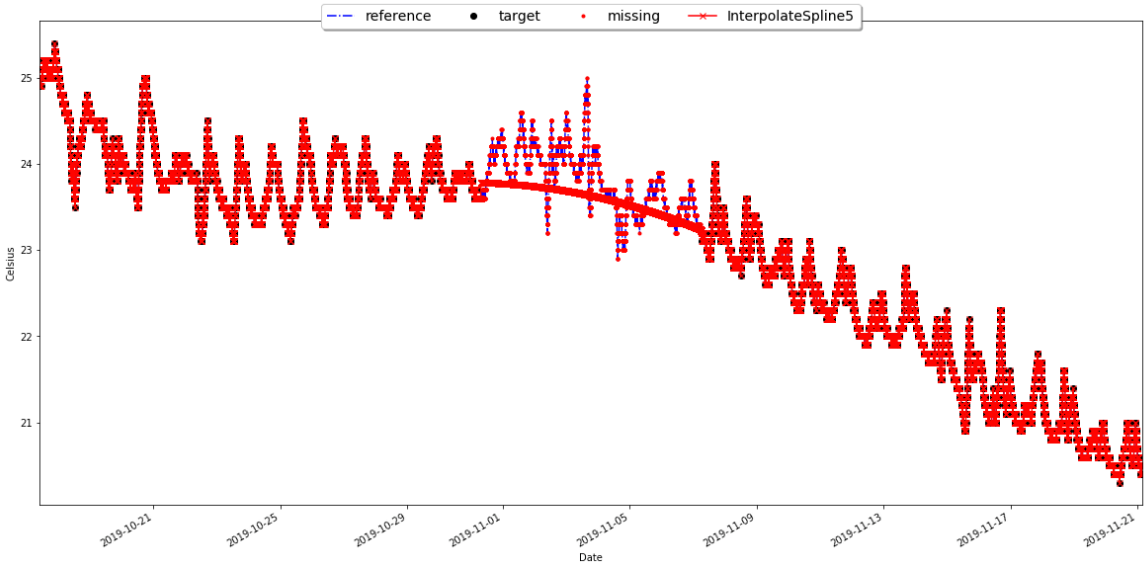


Figure 4-10 - Sample dataset of Celsius feature after data imputation.

In the last part of the Missing Values Treatment step, data imputation was applied to the whole dataset following the table created with the method with the highest score by feature.

	Metric	Method	R_squared
0	C2H5OH	InterpolateLinear	0.7661558571030871
1	C3H8	InterpolateSpline5	0.5911667899565282
2	C4H10	InterpolateSpline5	0.6310795921581845
3	CH4	InterpolateTime	0.756378662422326
4	CO	InterpolateLinear	0.761215629553004
5	CO2	FillMedian	0.8374440598367406
6	Celsius	InterpolateSpline5	0.9806422239252416
7	Fahrenheit	InterpolateSpline5	0.9806422239255362
8	H2	InterpolateLinear	0.7644016699015792
9	Humidity	InterpolateSpline3	0.9282727914664706
10	NH3	InterpolateSpline3	0.6676663036288546
11	NO2	FillMedian	0.9171628339915856
12	Pressure	InterpolateSpline3	0.8344865848036843

Table 1 - The best method for each feature by R2 score.

Below, an example of a feature before and after data imputation (whole dataset):

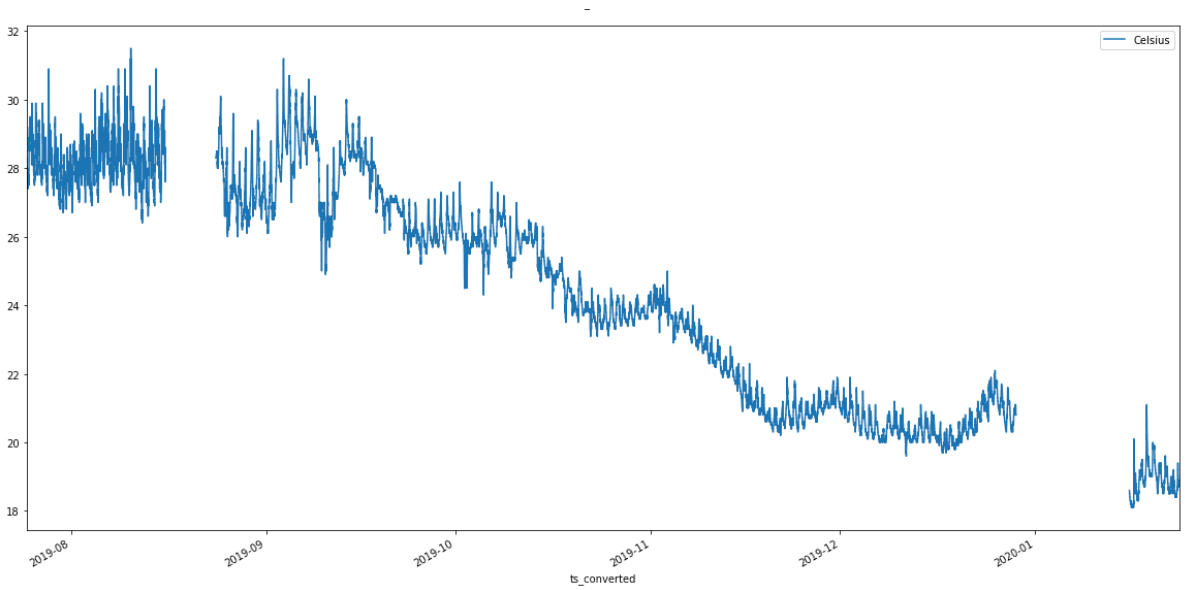


Figure 4-11- Whole dataset of Celsius feature before data imputation.

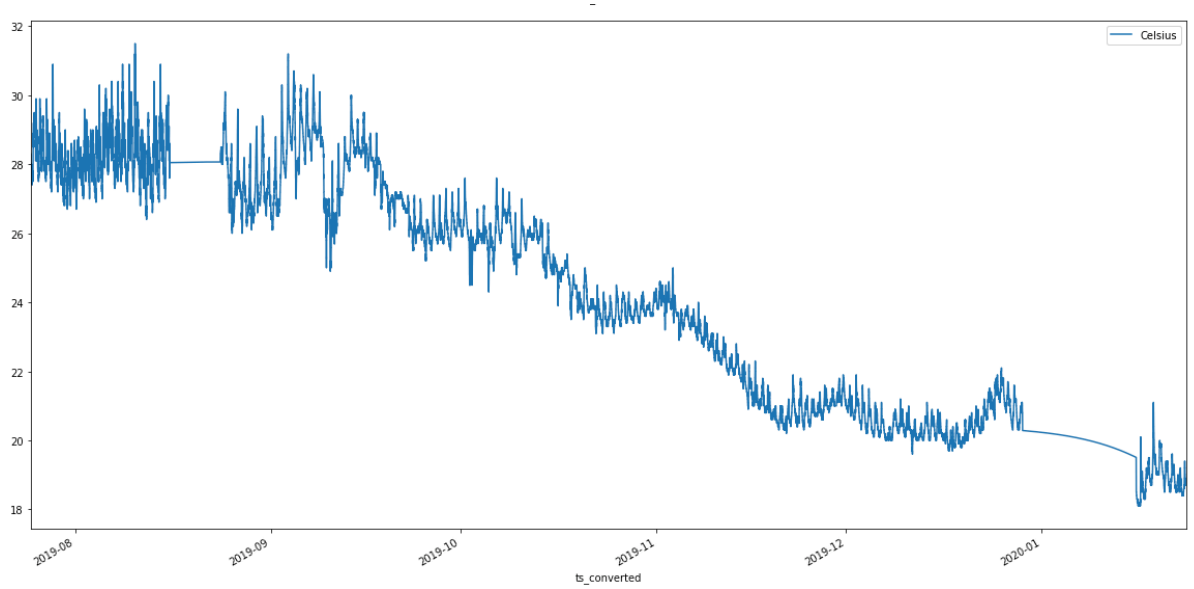


Figure 4-12 - Whole dataset of Celsius feature after data imputation.

After the removal of initial readings and of the first readings right after the identified data gaps, the dataset was reduced. After the first part of data imputation, the dataset preparation to receive imputed values, the dataset was incremented. Originally with 228.307 rows, after these two steps, the dataset had 263.357 observations. Out of these 263.357 records, 37.440 had missing values. So, the missing values accounted for 14,22% of the dataset. The Missing Values Treatment step eliminated all null values from the dataset.

4.3.4. Outliers Treatment

The Outliers Treatment step has the objective to detect and replace data points that differ significantly from other observations. The Outliers Treatment step was divided into three parts: outliers identification, outliers removal, and data imputation post outliers treatment. The main idea behind dealing with outliers is to pinpoint outliers through the application of a statistic method called IQR (Interquartile Range). After that, to delete outliers, they were simply replaced with null values. As the last step in the process of treating outliers, null values were replaced through data imputation following the same top recommended method for each feature as identified in the Missing Values Treatment step.

Steps taken to detect and replace outliers:

1. Since the Data Science process was conducted using Python, the dataset was converted from a Pandas dataframe to three Numpy arrays. The first and last Numpy arrays contained variables without the possibility of having outliers;
2. A function to identify outliers and replace them with null values was developed;
    a. The IQR was calculated as the difference from percentile 75^{th} and 25^{th};

$$IQR = Q75 - Q25$$

b. The outlier cutoff was calculated as the multiplication of IQR by a factor K and the lower and upper boundaries were established like percentile 25^{th} minus the cutoff and percentile 75^{th} plus the cutoff, respectively;

$$cutoff = IQR * k$$

$$lower = Q25 - cutoff$$

$$upper = Q75 + cutoff$$

Obs.: Usually, the K factor is set as 1.5 for outliers and 3 for extreme outliers (Tukey, 1977, p. 44). In this case, a K factor of 3 was used.

c. Values lesser than the lower cutoff or larger than the upper boundary were identified as outliers and replace by null values.

3. Variables likely to have outliers from the Numpy array in the middle were iterated and the function was applied;
4. After detecting and removing the outliers, the Numpy arrays were concatenated back together;
5. The resulting Numpy array was converted back to a Pandas dataframe.
6. As a final step, data and data types were double-checked to make sure that the final dataset was equivalent to the initial one, but without outliers.

The last part of Outliers Treatment step is to impute data where once were the outliers and now missing values can be found. By applying, for each feature, the top-recommended method for data imputation, almost all missing values created by outliers removing have been replaced by a consistent value. An interesting fact must be highlighted. Because three features, C3H8, C4H10, and CH4 gases' readings presented outliers in the initial of the dataset, the null values that replaced the outliers at the beginning of the dataset were not changed by the top-recommended method for data imputation. That occurred because the top method in all three cases was the slinear method, a kind of interpolation. And interpolation methods create a bridge between the two edges data points in the data gap. Without the first edge data point from the data gap, it is not feasible to use interpolation. For that reason and because outliers were not in place anymore, to fill the null values from the beginning of the dataset for features C3H8, C4H10, and CH4, the median of each series was applied as a data imputation method.

index	0
ts_converted	0
C2H5OH	0
C3H8	298
C4H10	153
CH4	1
CO	0
CO2	0
Celsius	0
Fahrenheit	0
H2	0
Humidity	0
NH3	0
NO2	0
Pressure	0
Year	0
Month	0
Day	0
Hour	0
Minute	0
Second	0
Weekday_Name	0
Date	0
Time	0
Day_Of_Year	0
Week_Of_Year	0
Quarter	0
Period	0
Date_Period	0

index	0
ts_converted	0
C2H5OH	0
C3H8	0
C4H10	0
CH4	0
CO	0
CO2	0
Celsius	0
Fahrenheit	0
H2	0
Humidity	0
NH3	0
NO2	0
Pressure	0
Year	0
Month	0
Day	0
Hour	0
Minute	0
Second	0
Weekday_Name	0
Date	0
Time	0
Day_Of_Year	0
Week_Of_Year	0
Quarter	0
Period	0
Date_Period	0

Table 2 - Features and the number of null values after data imputation first and second iterations.

4.3.5. Descriptive Statistics

The Descriptive Statistics step has the objective to present a statistical summary of each time-series feature. Eight statistical properties are showed: count, mean, standard deviation, minimum value, 25th percentile, median, 75th percentile, maximum value.

	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	PRCP	TAVG
count	263357.000	263357.000	263357.000	263357.000	263357.000	263357.0	263357.000	263357.000	263357.000	263357.000	263357.000	263357.000	263357.000	263357.000	263357.000
mean	0.606	1.788	2.969	74.584	1.966	399.0	24.084	75.399	0.244	54.931	0.014	10.017	1007.536	1.370	17.577
std	0.465	1.870	2.887	102.172	1.215	0.0	3.384	6.111	0.209	10.678	0.013	5.405	5.851	4.238	4.585
min	0.000	0.000	0.000	0.000	0.000	399.0	18.100	64.580	0.000	25.400	0.000	0.000	985.000	0.000	8.900
25%	0.280	0.430	0.880	4.740	1.170	399.0	20.600	69.295	0.100	47.300	0.010	7.420	1004.000	0.000	13.900
50%	0.530	1.110	1.990	28.708	1.880	399.0	24.000	75.200	0.200	54.900	0.010	10.190	1008.000	0.000	17.100
75%	0.839	1.970	3.230	99.400	2.646	399.0	27.300	81.140	0.340	64.300	0.020	11.460	1011.000	0.000	21.200
max	2.490	6.289	9.709	366.580	7.010	399.0	31.500	88.700	1.050	81.000	0.054	32.530	1024.000	32.000	27.300

Figure 4-13- Time-series features summary statistics.

4.3.6. Data Visualization

The Data Visualization step has as objective to better understand the dataset by analyzing graphically one feature at a time and to identify temporal structures like trends, cycles, and seasonality.

Five different types of plots were employed: line plot, histogram, box and whisker plot (by month), lag scatter plot, and autocorrelation plot. Line plots were used to analyze all the observations of each feature by time to detect trend and seasonality. Histograms were utilized to evaluate, without the temporal ordering, the distribution of observations for each input variable. On the other hand, box and whisker plots were applied to examine the distribution of values by time interval. In this case, by month. Lag scatter plots were used to explore the relationship between each observation and itself from a previous timestep. Finally, autocorrelation plots were employed to quantify the strength and type of relationship between observations and their lags (Brownlee, 2018c, p. 50).

As an example, below all five plots from feature Celsius:

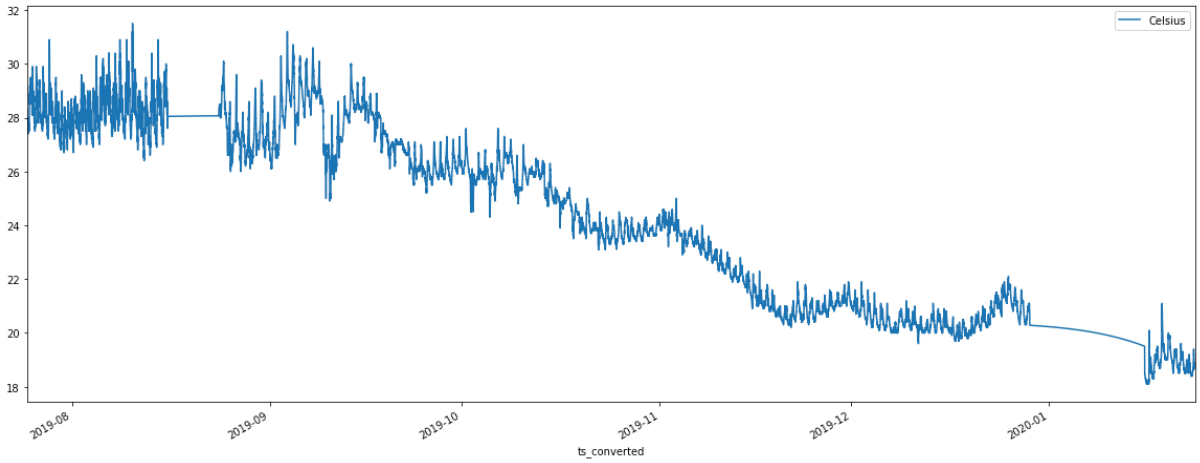


Figure 4-14 - Feature Celsius line plot.

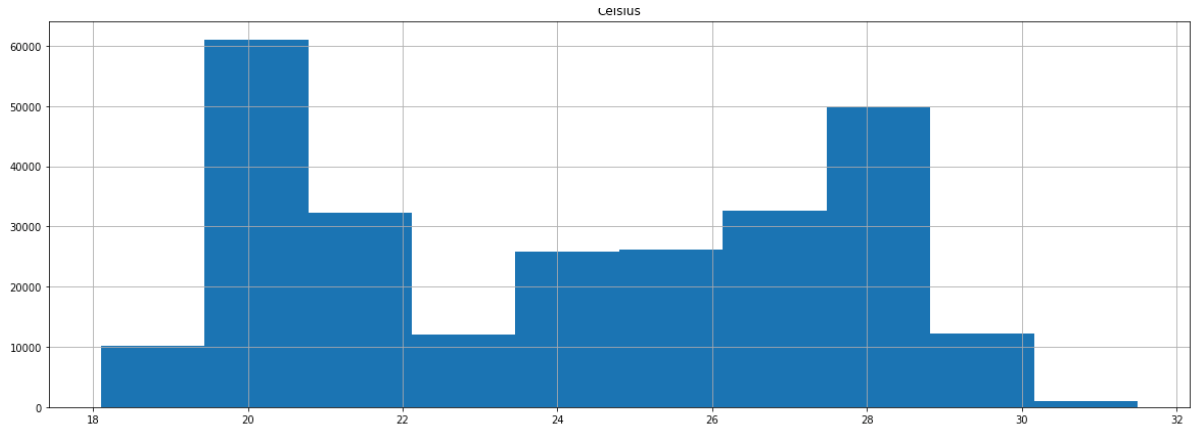


Figure 4-15 - Feature Celsius histogram.

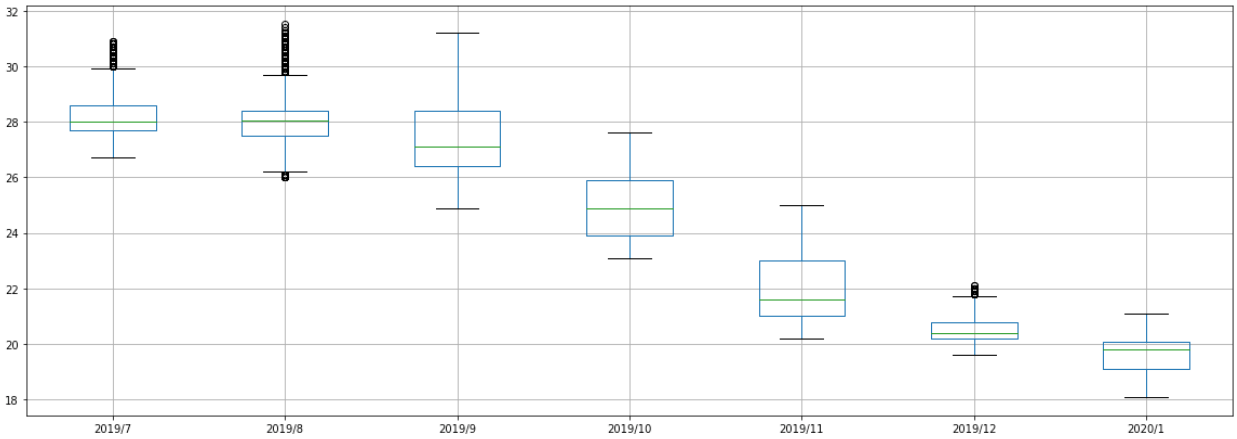


Figure 4-16 - Feature Celsius box and whisker plot.

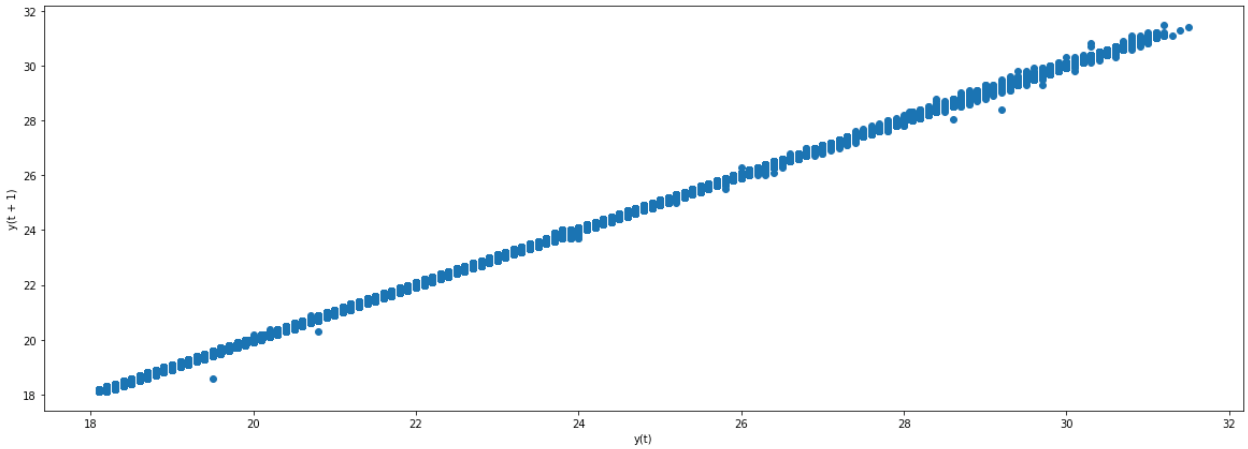


Figure 4-17 - Feature Celsius lag scatter plot.

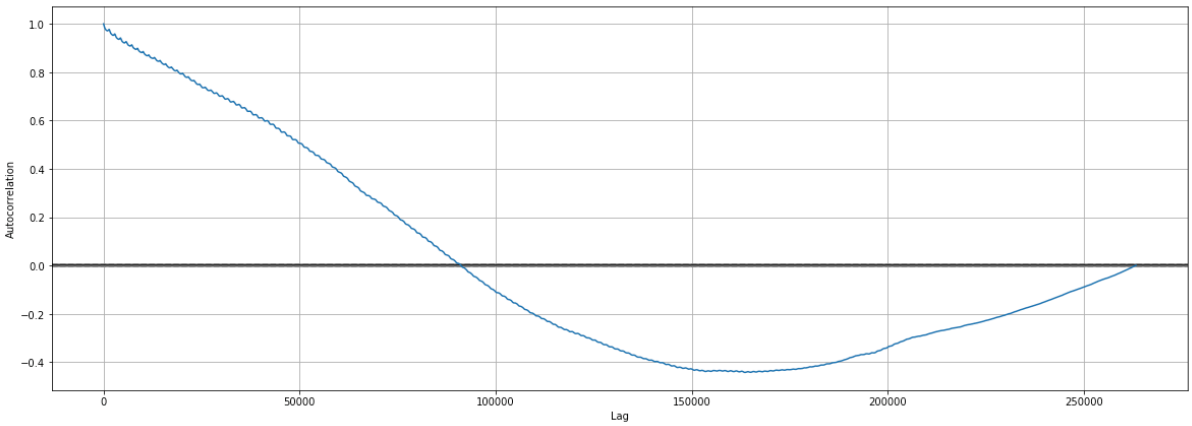


Figure 4-18 - Feature Celsius autocorrelation plot.

4.3.7. Feature Engineering

To help both Deep Learning and other Machine Learning models to obtain a more assertive prediction, a couple of new features were created out of the original inputs found in the dataset. These new features can be divided into three classes: date-time features, lag features, and climate features.

4.3.7.1. Date-Time Features

Since data is captured by sensors every minute, all measures are registered with a complete date field. This is important because allowed the creation of the following new features: Year, Month, Day, Hour, Minute, Second, Weekday_Name, Date (yyyy/mm/dd), Time, Day_Of_Year, Week_Of_Year and Quarter.

key ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter
2019-07-23 23:10:05	2.40	42.92	44.27	1890.64	5.90	399.0	23.6	74.48	1.15	67.3	0.12	0.58	1004.0	2019	7	23	23	10	5	Tuesday	2019-07-23	23:10:05	204	30	3
2019-07-23 23:16:11	1.74	33.10	35.52	762.61	4.62	399.0	23.8	74.84	0.79	66.8	0.10	0.64	1004.0	2019	7	23	23	16	11	Tuesday	2019-07-23	23:16:11	204	30	3
2019-07-23 23:17:09	1.81	31.63	34.17	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.64	1004.0	2019	7	23	23	17	9	Tuesday	2019-07-23	23:17:09	204	30	3
2019-07-23 23:18:09	1.81	30.19	32.85	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.65	1004.0	2019	7	23	23	18	9	Tuesday	2019-07-23	23:18:09	204	30	3
2019-07-23 23:19:09	1.85	30.19	32.85	915.53	4.85	399.0	23.8	74.84	0.85	67.5	0.10	0.66	1004.0	2019	7	23	23	19	9	Tuesday	2019-07-23	23:19:09	204	30	3

Figure 4-19 - Dataset with new date-time features.

Although there were some new features, a date-time feature more representative of the problem was still missing. The main idea is that from time to time the Ambiosensing system will be capable of predict the temperature or a combination of temperature and humidity so it can adjust its settings to more efficient energy use. But, to do so, a time window of a minute is too short and of a day is too long. Therefore, a new date-time feature was created named Date_Period. The Date_Period feature divided the day into six periods of four hours each and presents itself, for instance, as 2019-07-23-P1 (from 00:00 to 03:59), 2019-07-23-P2 (from 04:00 to 07:59), etc.

	key	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period
0	0	2019-07-23 23:10:05	2.40	42.92	44.27	1890.64	5.90	399.0	23.6	74.48	1.15	67.3	0.12	0.58	1004.0	2019	7	23	23	10	5	Tuesday	2019-07-23	23:10:05	204	30	3	P6	2019-07-23-P6
1	1	2019-07-23 23:16:11	1.74	33.10	35.52	762.61	4.62	399.0	23.8	74.84	0.79	66.8	0.10	0.64	1004.0	2019	7	23	23	16	11	Tuesday	2019-07-23	23:16:11	204	30	3	P6	2019-07-23-P6
2	2	2019-07-23 23:17:09	1.81	31.63	34.17	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.64	1004.0	2019	7	23	23	17	9	Tuesday	2019-07-23	23:17:09	204	30	3	P6	2019-07-23-P6
3	3	2019-07-23 23:18:09	1.81	30.19	32.85	862.04	4.77	399.0	23.8	74.84	0.83	67.2	0.10	0.65	1004.0	2019	7	23	23	18	9	Tuesday	2019-07-23	23:18:09	204	30	3	P6	2019-07-23-P6
4	4	2019-07-23 23:19:09	1.85	30.19	32.85	915.53	4.85	399.0	23.8	74.84	0.85	67.5	0.10	0.66	1004.0	2019	7	23	23	19	9	Tuesday	2019-07-23	23:19:09	204	30	3	P6	2019-07-23-P6

Figure 4-20 - Dataset with new Date_Period feature.

One last date-time feature was created to try to help the models. This new feature is an external feature because it is not engineered out of the original inputs. It is a holiday flag feature called Holiday_Flag. Since the Ambiosensing system could be applied to office spaces or even to shopping mall stores, this kind of information might be useful and help to improve model accuracy. To obtain the holidays, a Python's library named Holidays was used. This Python's library contains all the major

holidays known for different countries and, in some cases like the United States and Brazil, it gets to the state level holidays ("Holidays Library Documentation," n.d.). Since Ambiosensing Device is placed in Lisbon and the Holidays library doesn't get to state-level within Portugal, PortugalExt setting was used to get every Portuguese national holiday and also extended days that most people have off.

O2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	PRCP	TAVG	Holiday_Flag
.19	1011.458889	2020	1	1	0	0	0	Wednesday	2020-01-01	00:00:00	1	1	1	P1	2020-01-01-P1	0.0	9.7	1.0
.19	1011.457271	2020	1	1	0	1	0	Wednesday	2020-01-01	00:01:00	1	1	1	P1	2020-01-01-P1	0.0	9.7	1.0
.19	1011.455652	2020	1	1	0	2	0	Wednesday	2020-01-01	00:02:00	1	1	1	P1	2020-01-01-P1	0.0	9.7	1.0
.19	1011.454034	2020	1	1	0	3	0	Wednesday	2020-01-01	00:03:00	1	1	1	P1	2020-01-01-P1	0.0	9.7	1.0
.19	1011.452415	2020	1	1	0	4	0	Wednesday	2020-01-01	00:04:00	1	1	1	P1	2020-01-01-P1	0.0	9.7	1.0

Figure 4-21 - Dataset with new Holiday_Flag feature when it is a holiday.

O2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	PRCP	TAVG	Holiday_Flag
19	1009.017115	2020	1	2	0	0	0	Thursday	2020-01-02	00:00:00	2	1	1	P1	2020-01-02-P1	0.0	10.2	0.0
19	1009.015355	2020	1	2	0	1	0	Thursday	2020-01-02	00:01:00	2	1	1	P1	2020-01-02-P1	0.0	10.2	0.0
19	1009.013595	2020	1	2	0	2	0	Thursday	2020-01-02	00:02:00	2	1	1	P1	2020-01-02-P1	0.0	10.2	0.0
19	1009.011835	2020	1	2	0	3	0	Thursday	2020-01-02	00:03:00	2	1	1	P1	2020-01-02-P1	0.0	10.2	0.0
19	1009.010075	2020	1	2	0	4	0	Thursday	2020-01-02	00:04:00	2	1	1	P1	2020-01-02-P1	0.0	10.2	0.0

Figure 4-22 - Dataset with new feature Holiday_Flag when it is not a holiday.

4.3.7.2. Lag Features

In a time series forecasting problem, it is very important to the model if a feature can contribute with time-dependent knowledge. One major advantage of Deep Learning models like LSTMs over other Machine Learning models is exactly that it knows how to extract a temporal relationship from a sequence of numbers (Brownlee, 2018a, p. 7). On the other hand, to obtain some temporal awareness, other Machine Learning models require lag features, which are input variables created based on values at prior time steps. With some lag features, Machine Learning models improve in this matter but still cannot extract the knowledge that each different lag feature is actually the same input variable in another time step. They will treat each lag feature as a completely different input.

Because of the positive contribution of lag features to other Machine Learning models to be compared with Deep Learnings models, lag features were created. These lag features were also evaluated on the feature selection step explained further to narrow down how much lag features would be delivered to the models.

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius_t-18	Celsius_t-12	Celsius_t-6	Celsius_t-3	Celsius_t-2	Celsius_t-1	Celsius
0	201907281	28	209	6	5.082473	3.413125	1.541917	55.525833	999.770833	21.9	27.807500	28.084583	28.247500	29.022917	28.272500	28.589583	27.791667
1	201907282	28	209	6	5.317046	3.471208	1.548208	55.203750	999.208333	21.9	27.643750	27.750833	27.804167	28.272500	28.589583	27.791667	27.389167
2	201907283	28	209	6	4.183874	3.458500	4.491792	55.701250	1000.066667	21.9	28.833333	29.134167	27.870417	28.589583	27.791667	27.389167	29.032917
3	201907284	28	209	6	5.852645	4.210250	1.996042	52.837500	999.620833	21.9	28.952917	28.427083	29.022917	27.791667	27.389167	29.032917	28.927917
4	201907285	28	209	6	5.976042	3.271208	1.743875	48.847083	999.404167	21.9	28.672917	28.226250	28.272500	27.389167	29.032917	28.927917	28.604167

Figure 4-23 – Celsius lag features.

4.3.7.3. Climate Features

The Holiday_Flag explained in Date-Time Features is not the only external feature added to the original inputs. Since climate data plays a major role in the Ambiosensing system because it affects indoor conditions, external temperature and precipitation data were also included in the dataset. Climate data were extracted from a public dataset from NOAA (National Oceanic and Atmospheric Administration) GHCN (Global Historical Climatology Network) Daily (Menne et al., 2012). This public dataset is available through Google Cloud Public Datasets as part of Google Cloud Platform ("Google Cloud Public Datasets," n.d.).

D2	Pressure	Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	PRCP	TAVG
06	1002.0	2019	7	25	0	0	9	Thursday	2019-07-25	00:00:09	206	30	3	P1	2019-07-25-P1	0.0	22.6
14	1001.0	2019	7	25	0	1	9	Thursday	2019-07-25	00:01:09	206	30	3	P1	2019-07-25-P1	0.0	22.6
78	1002.0	2019	7	25	0	2	8	Thursday	2019-07-25	00:02:08	206	30	3	P1	2019-07-25-P1	0.0	22.6
87	1001.0	2019	7	25	0	3	8	Thursday	2019-07-25	00:03:08	206	30	3	P1	2019-07-25-P1	0.0	22.6
88	1001.0	2019	7	25	0	4	9	Thursday	2019-07-25	00:04:09	206	30	3	P1	2019-07-25-P1	0.0	22.6

Figure 4-24 - Dataset with new climate features.

NOAA GHCN Daily dataset has data from all around the world. Since Ambiosensing Device is in Lisbon, data were extracted, particularly, from the Lisbon meteorological station from the climatology network. The external temperature feature, called TAVG, is an average of hourly values for a day in tenths of Celsius degree. Later these data were transformed to Celsius degree to better comparability with existing features in the dataset. The precipitation feature, named PRCP, is the precipitation in tenths of millimeters. This feature was also transformed, but for millimeters, for better understandability.

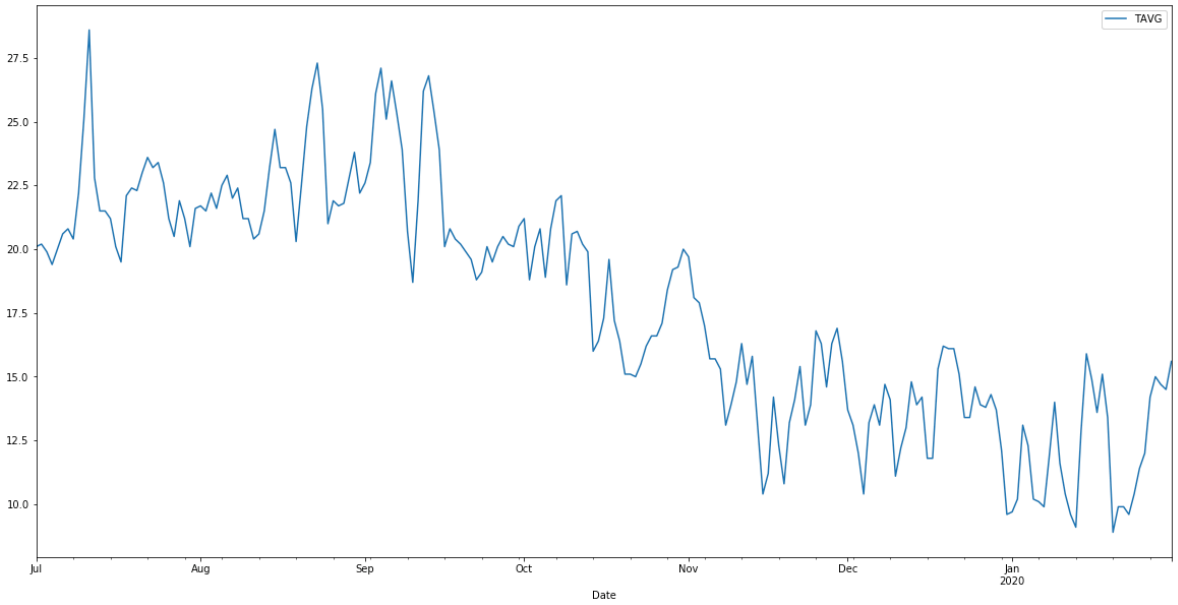


Figure 4-25- External temperature feature for the whole dataset period.

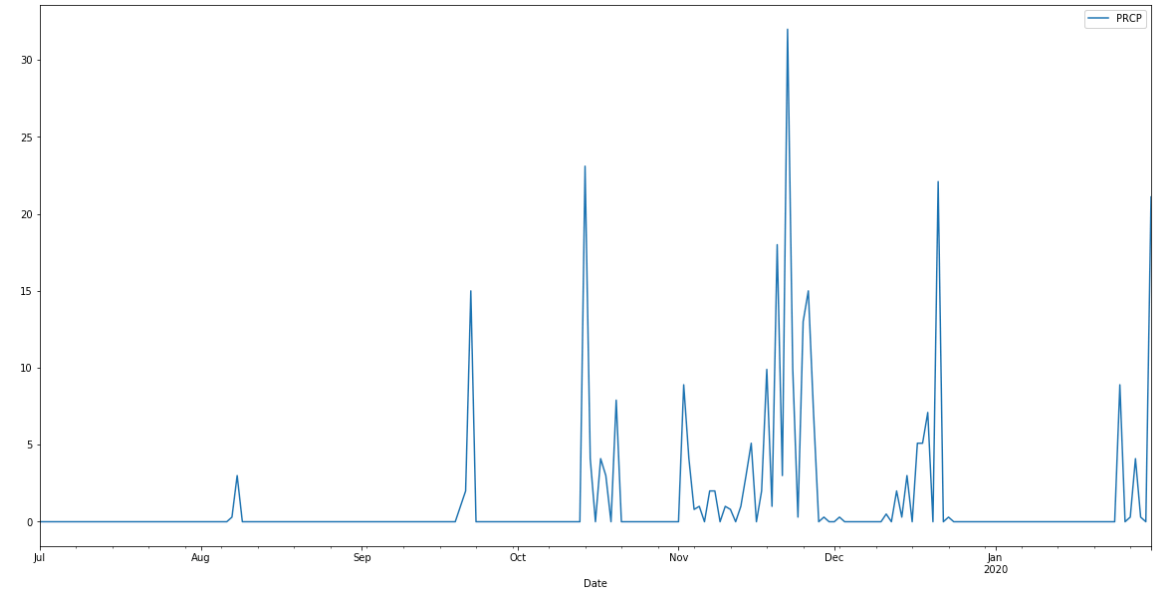


Figure 4-26 - Precipitation feature for the whole dataset period.

4.3.8. Feature Selection

The objective of Feature Selection step is to select the set of features that contributes the most to the prediction variable. The Feature Selection step was divided into two parts: a feature selection of all variables to be used on Deep Learning models and other Machine Learning models and a complementary feature selection of lag variables specific to be used on other Machine Learning models.

After all preparation steps so far in the Data Science framework, the dataset was presented as follows (the dataset is split in two because of the large number of variables):

	index	ts_converted	C2H5OH	C3H8	C4H10	CH4	CO	CO2	Celsius	Fahrenheit	H2	Humidity	NH3	NO2	Pressure	Year
0	0	2019-07-25 00:00:09	1.55	1.788279	2.968946	2.968946	4.230	399.0	28.3	82.94	0.690000	57.4	0.050000	1.06	1002.0	2019
1	1	2019-07-25 00:01:09	1.24	1.788279	2.968946	294.250000	3.570	399.0	28.2	82.76	0.530000	57.6	0.050000	1.14	1001.0	2019
2	2	2019-07-25 00:02:08	1.84	1.788279	2.968946	295.764000	4.775	399.0	28.2	82.76	0.703333	58.7	0.053675	0.78	1002.0	2019
3	3	2019-07-25 00:03:08	2.44	1.788279	2.968946	297.278000	5.980	399.0	28.2	82.76	0.876667	58.4	0.053674	0.87	1001.0	2019
4	4	2019-07-25 00:04:09	2.22	1.788279	2.968946	298.792000	5.570	399.0	28.2	82.76	1.050000	58.9	0.053673	0.88	1001.0	2019

Figure 4-27 - Variables from the beginning of the dataset.

Year	Month	Day	Hour	Minute	Second	Weekday_Name	Date	Time	Day_Of_Year	Week_Of_Year	Quarter	Period	Date_Period	PRCP	TAVG	Holiday_Flag
2019	7	25	0	0	9	Thursday	2019-07-25	00:00:09	206	30	3	P1	2019-07-25-P1	0.0	22.6	0.0
2019	7	25	0	1	9	Thursday	2019-07-25	00:01:09	206	30	3	P1	2019-07-25-P1	0.0	22.6	0.0
2019	7	25	0	2	8	Thursday	2019-07-25	00:02:08	206	30	3	P1	2019-07-25-P1	0.0	22.6	0.0
2019	7	25	0	3	8	Thursday	2019-07-25	00:03:08	206	30	3	P1	2019-07-25-P1	0.0	22.6	0.0
2019	7	25	0	4	9	Thursday	2019-07-25	00:04:09	206	30	3	P1	2019-07-25-P1	0.0	22.6	0.0

Figure 4-28 - Variables from the end of the dataset.

Two methods were used to decide which features should continue on the dataset to feed the models: a feature importance scoring and a feature selection algorithm. Because to calculate feature importance a RandomForestRegressor from Python's library scikit-learn was used, the dataset needed to be transformed to have only numerical features ("sklearn.ensemble.RandomForestRegressor — scikit-learn 0.23.2 documentation," n.d.). Timestamp and string features were converted to numerical variables whenever possible and, where not, they were replaced. That was the case with the Weekday_Name feature, which content was like 'Sunday', 'Monday',… , 'Saturday', that was replaced by Day_Of_Week feature, which contained 1, 2,… , 7 as weekdays.

One more important data transformation took place before feature importance and feature selection. Since variables were about to get compared against the target, it was important to set the correct data granularity. As explained in the Feature Engineering step, a new date-time feature was created to be more representative of the problem of the energy management system Ambiosensing: the Date_Period feature. So, the dataset was aggregated by Date_Period.

After all data transformations, the dataset was presented as follows (the dataset is split in two because of the large number of variables):

	Date_Period	Date	Year	Month	Day	Period	Quarter	Week_Of_Year	Day_Of_Year	Day_Of_Week	C2H5OH	C3H8
0	201907251	20190725	2019	7	25	1	3	30	206	3	1.618375	1.788279
1	201907252	20190725	2019	7	25	2	3	30	206	3	1.206750	5.026167
2	201907253	20190725	2019	7	25	3	3	30	206	3	1.082750	6.090000
3	201907254	20190725	2019	7	25	4	3	30	206	3	1.333083	5.844917
4	201907255	20190725	2019	7	25	5	3	30	206	3	1.230375	5.372125

Figure 4-29- Dataset after data transformations for feature selection. Variables from the beginning of the dataset.

C3H8	C4H10	CH4	CO	CO2	H2	NH3	NO2	Celsius	Humidity	Pressure	PRCP	TAVG	Holiday_Flag
1.788279	5.379789	265.036141	4.369229	399.0	0.716583	0.052860	0.942625	27.807500	58.362500	1001.112500	0.0	22.6	0.0
5.026167	9.191572	249.943933	3.493708	399.0	0.518125	0.050288	1.172792	27.643750	58.569583	1001.637500	0.0	22.6	0.0
6.090000	9.222554	205.424442	3.213125	399.0	0.457167	0.051422	1.177625	28.833333	55.735000	1002.016667	0.0	22.6	0.0
5.844917	9.040784	266.700958	3.753125	399.0	0.583708	0.048378	1.610792	28.952917	56.799583	1002.345833	0.0	22.6	0.0
5.372125	7.786337	256.103337	3.532375	399.0	0.532542	0.045167	2.511958	28.672917	57.453750	1002.900000	0.0	22.6	0.0

Figure 4-30- Dataset after data transformations for feature selection. Variables from the end of the dataset.

4.3.8.1. Feature Selection of All Variables

After getting the dataset ready, all variables were fed to the Random Forest model and the feature importance scores were obtained.

Below, a table with all the features and their respective feature importance scores.

	feature_name	feature_importance
1	Day_Of_Year	0.2517441321527238
2	Day	0.22823264933943027
3	CO	0.07879469367740699
4	C2H5OH	0.056645996224850136
5	Day_Of_Week	0.05246441909464633
6	Date	0.04943160698005341
7	C3H8	0.03903202883862637
8	Celsius	0.036606124128159355
9	Date_Period	0.03553813114425665
10	TAVG	0.034185699378384976
11	Pressure	0.024199811767758875
12	Humidity	0.023250075310881593
13	NO2	0.022891888413039383
14	H2	0.018494435164277373
15	C4H10	0.017510536233578886
16	CH4	0.017127482601905466
17	Period	0.005453015129521543
18	Week_Of_Year	0.0035868767746025016
19	Month	0.002429569009960483
20	NH3	0.001967597099127309
21	Quarter	0.0002491271384136982 3
22	PRCP	0.00011034201052172654
23	Year	0.00005376238787261544 6
24	CO2	0

Table 3 - Features and feature importance scores.

Next, two plots with feature importance scores, the first one with all the features and the second one with the top two features by importance taken out to a clear perspective of other variables.

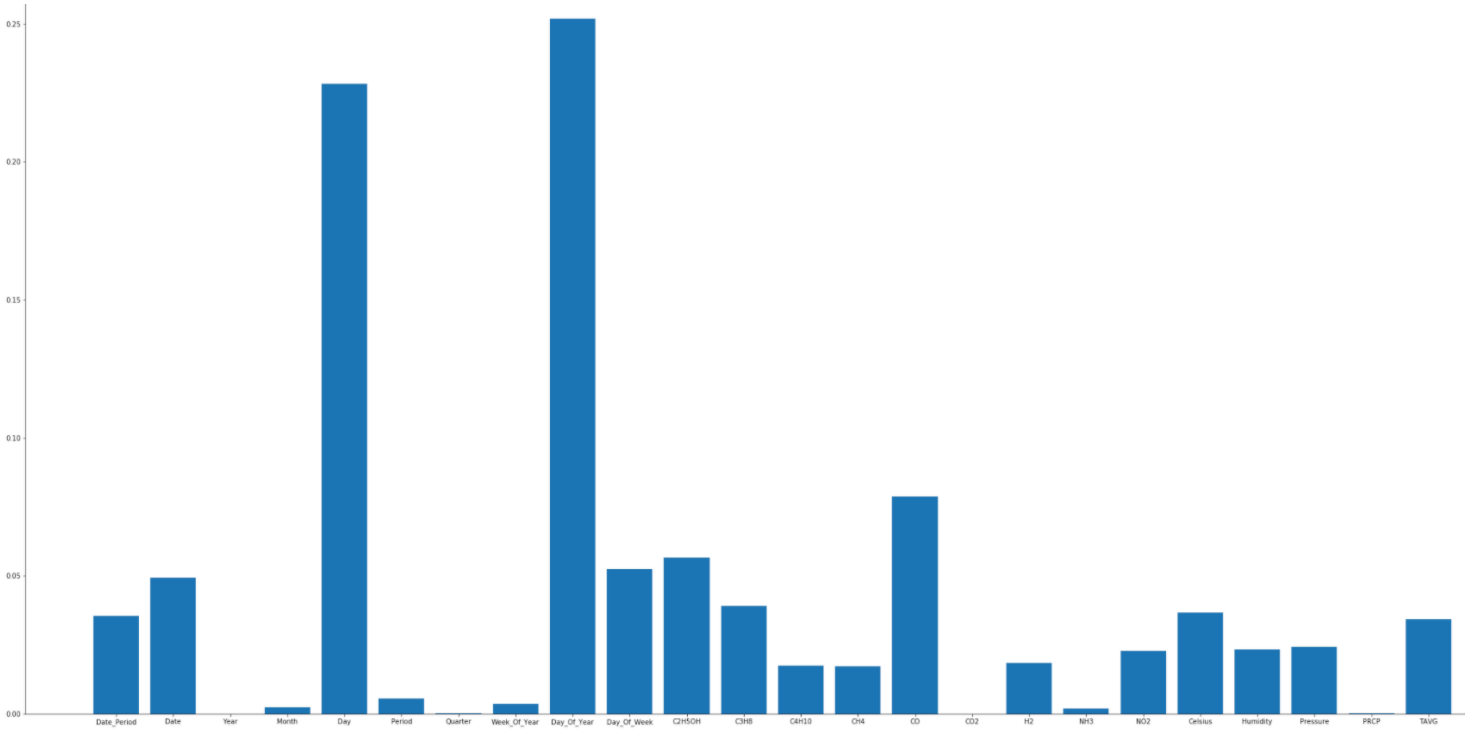


Figure 4-31- Features x Feature Importance Scores.

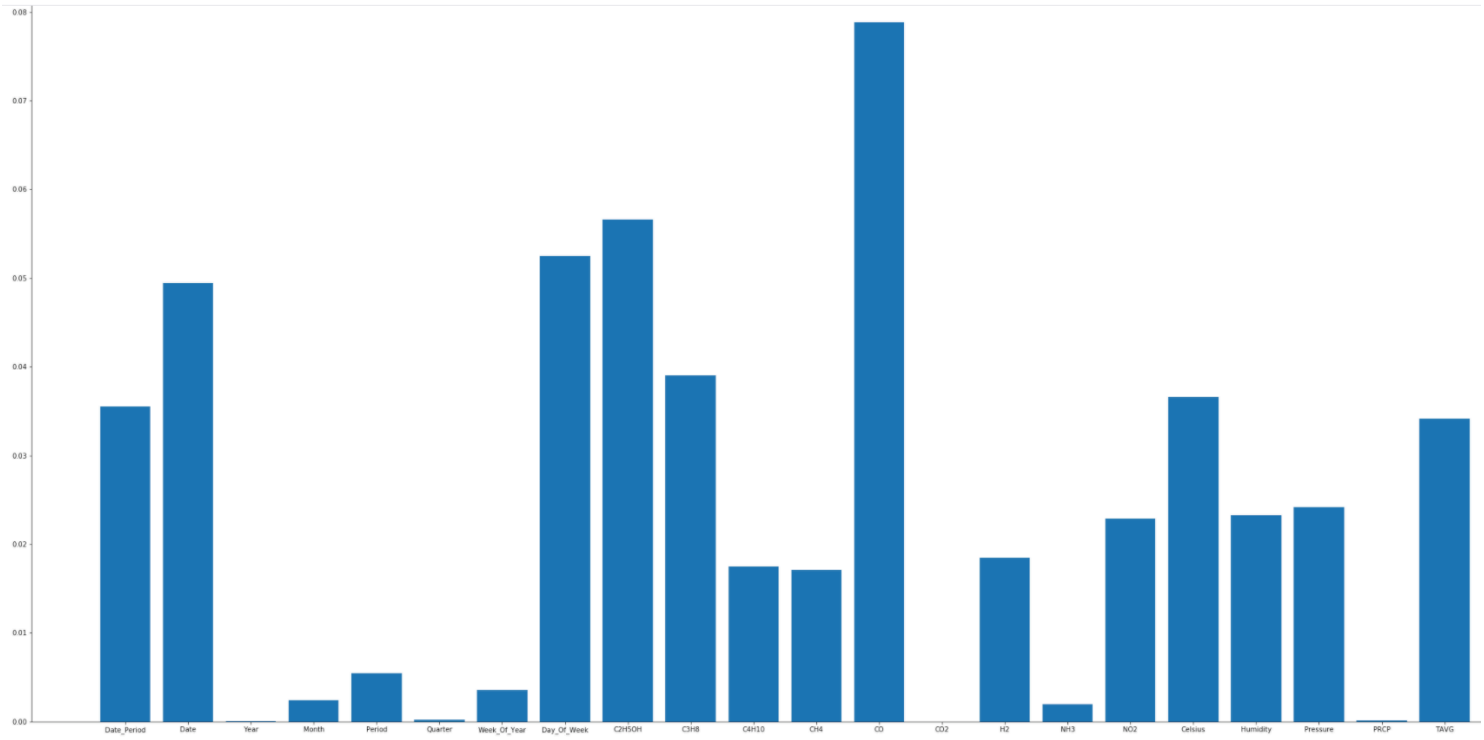


Figure 4-32 - Features x Feature Importance Scores (without top two features).

After feature importance analyzes, feature selection was performed. An RFE (Recursive Feature Selection) algorithm from Python's library scikit-learn was used. RFE algorithm demands a predictive model and again a Random Forest model was employed (Brownlee, 2019, p. 53).

Below, the list of selected features and a plot with their ranks.

```
Selected Features:
Date_Period
Date
Day
Day_Of_Year
Day_Of_Week
C2H5OH
C3H8
C4H10
CO
H2
NO2
Celsius
Humidity
Pressure
TAVG
```

Figure 4-33- List of selected features by the RFE algorithm.

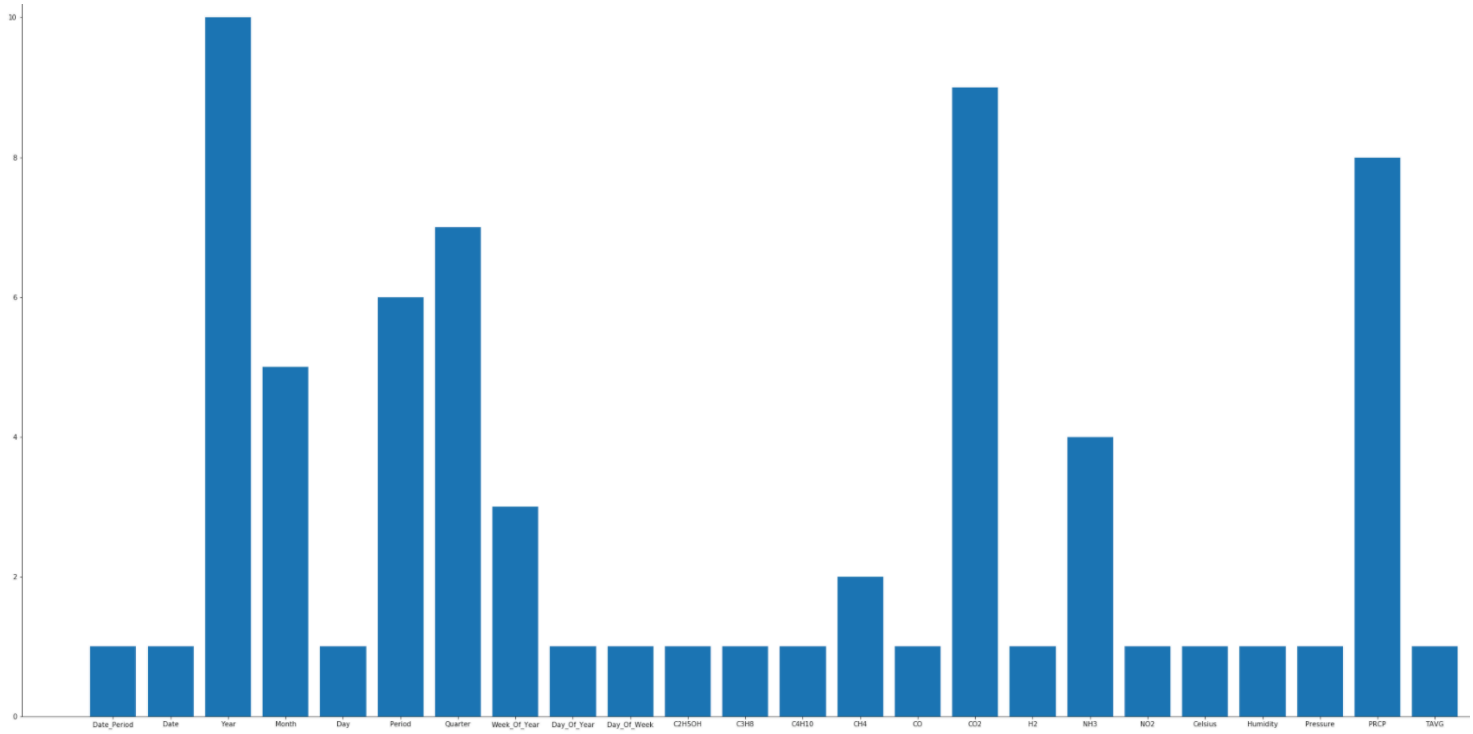


Figure 4-34 - Features x Feature Ranks by RFE algorithm (lower is better).

By the nature of some selected features, they seem to be highly correlated. So, before the final set of features, a correlation analysis was carried out. Since Pearson's correlation coefficient assumes a normal distribution of the attributes (Brownlee, 2019, p. 35) and since the selected features didn't present this characteristic, as can be seen in the histograms below, Kendall's correlation coefficient was applied (Dalgaard, n.d., p. 124).

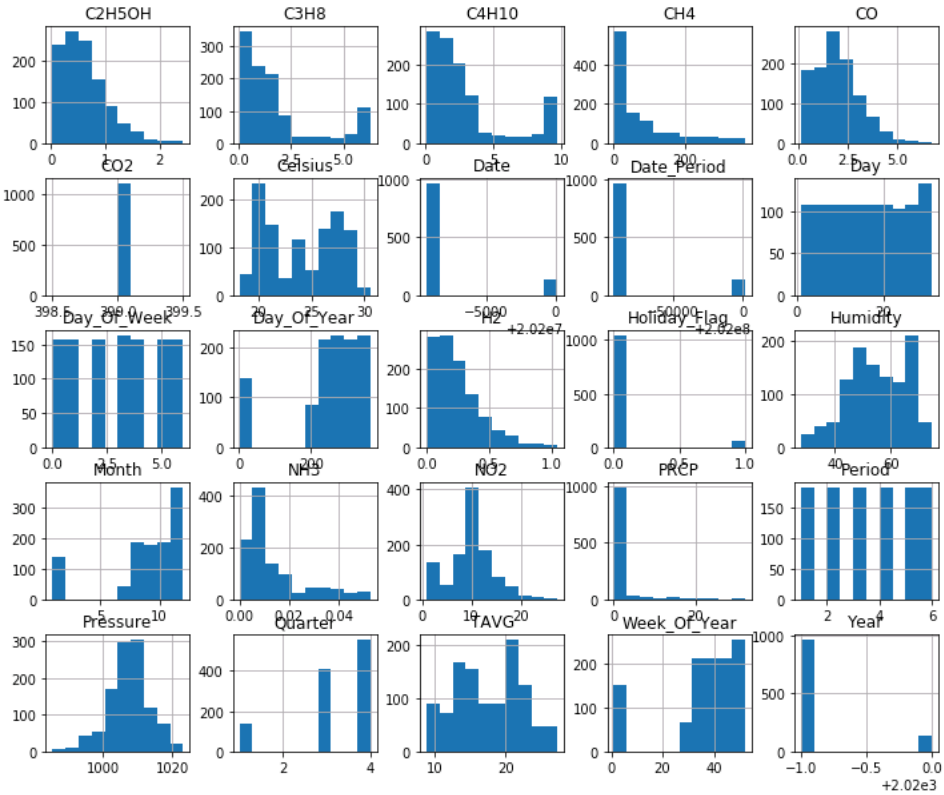


Figure 4-35 - Histogram of selected features.

Below, the correlation matrix created using Kendall's correlation coefficient.

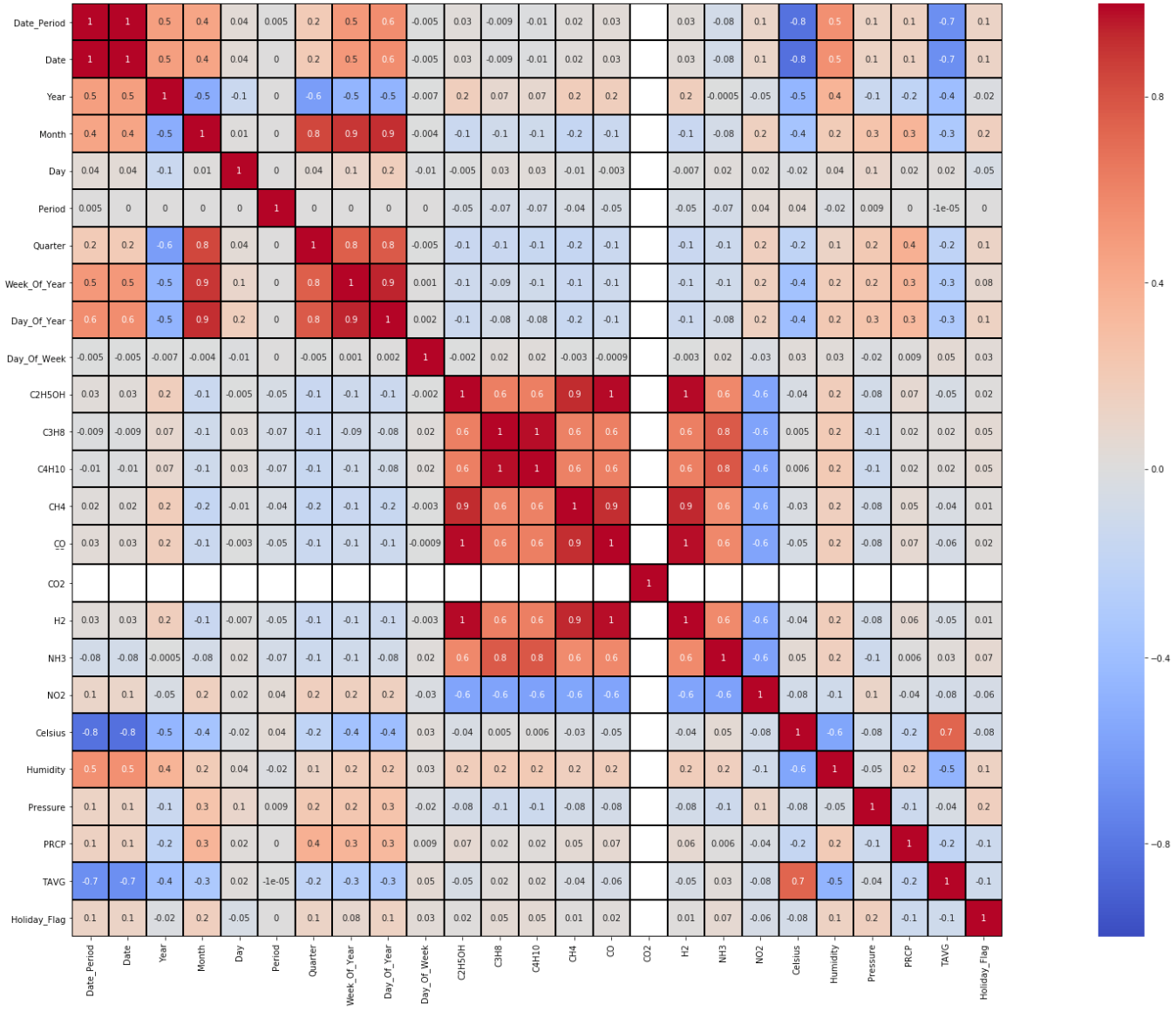


Figure 4-36 - Correlation matrix.

After analyzing the correlation matrix, some insights were extracted:

1. Date is highly correlated to Date_Period and it should be removed from the selected features;
2. C2H5OH is highly correlated to CO and CO has a bigger importance score. So, C2H5OH should also be removed from the selected features;
3. C4H10 is highly correlated to C3H8 and C3H8 has a bigger importance score. So, C4H10 should be removed too from the selected features;
4. Lastly, H2 is highly correlated to CO and CO has a bigger importance score. So, H2 should be removed as well from the selected features.

After finishing the Feature Selection for All Variables part, the dataset was presented as follows:

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius
0	201907251	25	206	3	1.788279	4.369229	0.942625	58.362500	1001.112500	22.6	27.807500
1	201907252	25	206	3	5.026167	3.493708	1.172792	58.569583	1001.637500	22.6	27.643750
2	201907253	25	206	3	6.090000	3.213125	1.177625	55.735000	1002.016667	22.6	28.833333
3	201907254	25	206	3	5.844917	3.753125	1.610792	56.799583	1002.345833	22.6	28.952917
4	201907255	25	206	3	5.372125	3.532375	2.511958	57.453750	1002.900000	22.6	28.672917

Figure 4-37 - Final dataset for Deep Learning models.

In the end, the final dataset had 1.098 observations, ten features, and the target variable. This dataset was later used with the Deep Learning models.

4.3.8.2. Complementary Feature Selection of Lag Features

The complementary feature selection was focused only on the lag features for the Machine Learning models other than Deep Learning. As explained in more detail in the Feature Engineering step, these models have to harvest temporal context out of lag features. The complementary feature selection objective is to define which lag feature should be added to the previously selected features.

Differently from the feature selection conducted previously for all variables, feature importance scores, and feature selection ranks were not calculated immediately. Since the complementary feature selection is about lag features, first, lag variables were evaluated with the help of an autocorrelation plot. Also, before exploring lag features through an autocorrelation plot, time-series components like trend or seasonality were removed if present (Brownlee, 2018c, p. 131).

Below, a plot of the complete series of the target variable Celsius.

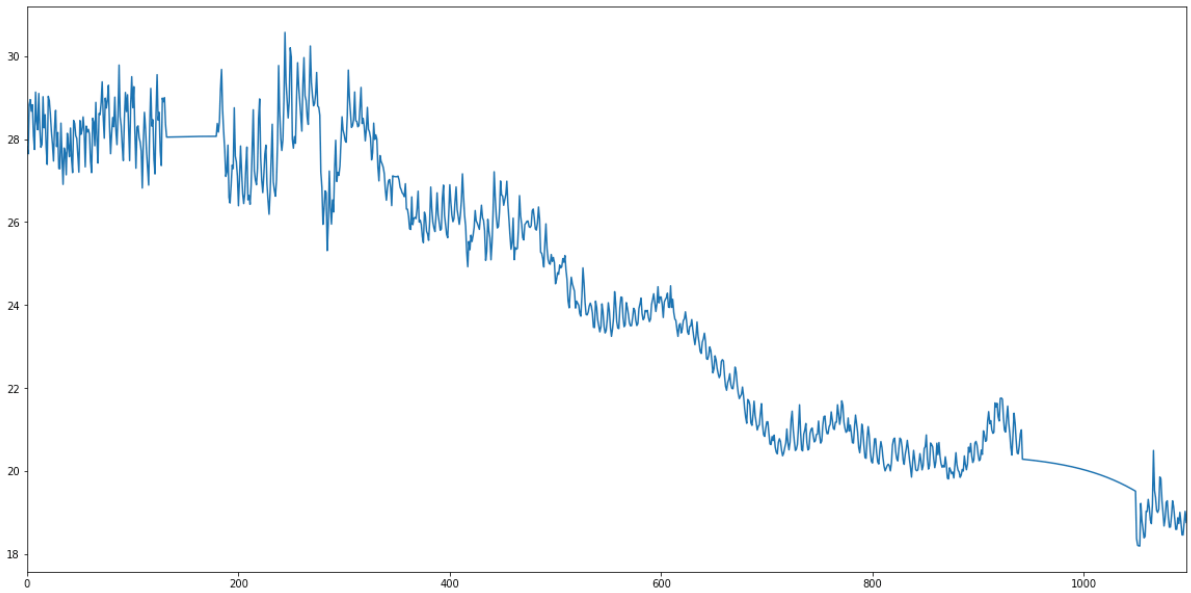


Figure 4-38 - Celsius feature series.

As can be seen from the plot, the Celsius series presents a decreasing trend. Because of that, to remove the trend from the Celsius series the differencing method was applied (Brownlee, 2018c, p. 114).

After removing the trend component, the Celsius series was presented as follows:

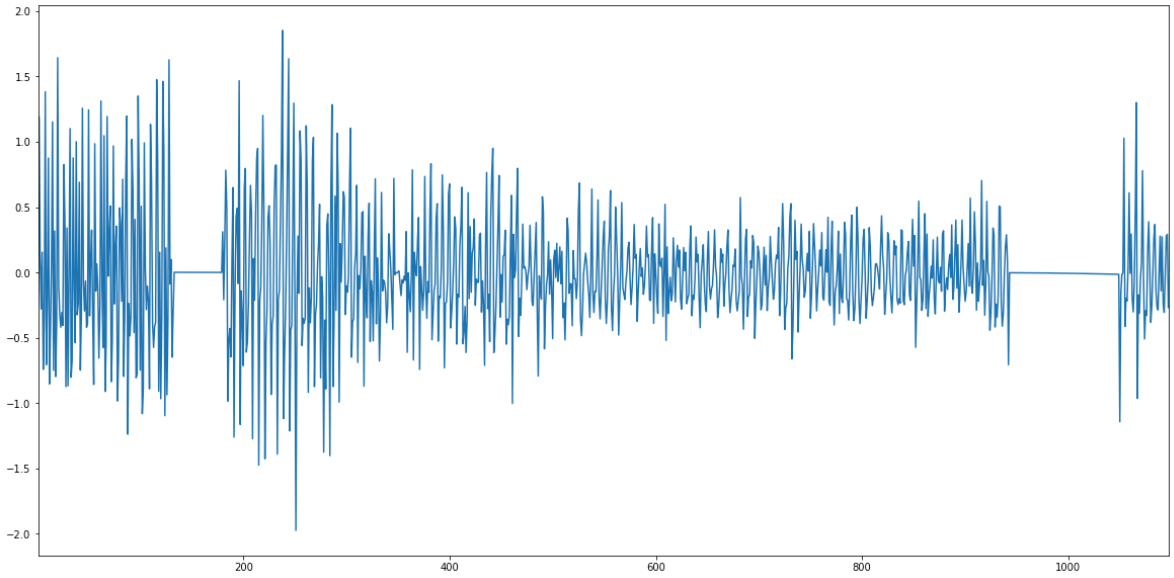


Figure 4-39- Celsius features series after trend removal.

Once the Celsius series didn't present a trend anymore, it was possible to create the autocorrelation plot.

Next, the autocorrelation plots of the two hundred first data points of the Celsius series.

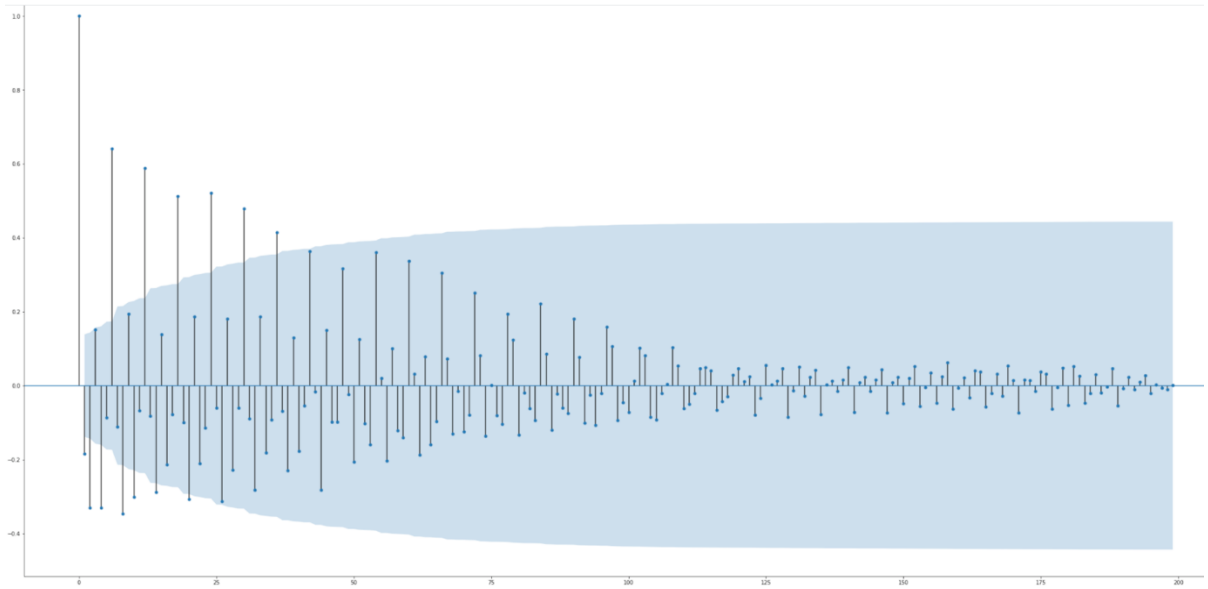


Figure 4-40 - Autocorrelation plot for the 200 first data points of the Celsius feature.

Since the dots below or above the blue area indicate statistical significance, the correlated data point that lasts the longest is data point 36. It represents a positive correlation. As each six data points represent one day, this means that the temperature from 6 days before still has some degree of correlation. After this important conclusion, the Celsius series was transformed into the dataset showed below with timesteps t-36, t-35,… , t-2, t-1, and t. This dataset was used to perform both feature importance scoring and feature selection ranking afterward.

	t-36	t-35	t-34	t-33	t-32	t-31	t-30	t-29	t-28	t-27	t-26
37	-0.163750	1.189583	0.119583	-0.280000	0.154702	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417
38	1.189583	0.119583	-0.280000	0.154702	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167
39	0.119583	-0.280000	0.154702	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333
40	-0.280000	0.154702	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250
41	0.154702	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500
42	-0.743036	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417
43	-0.333750	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417	0.317083
44	1.383333	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417	0.317083	-0.797917
45	-0.707083	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417	0.317083	-0.797917	-0.402500
46	-0.200833	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417	0.317083	-0.797917	-0.402500	1.643750
47	0.875417	-0.854167	-0.443333	0.066250	1.152500	-0.750417	0.317083	-0.797917	-0.402500	1.643750	-0.105000

Figure 4-41 - Celsius series with the oldest lag features.

t-10	t-9	t-8	t-7	t-6	t-5	t-4	t-3	t-2	t-1	t
0.397917	-0.875000	0.340833	-0.868750	-0.006667	1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750
-0.875000	0.340833	-0.868750	-0.006667	1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250
0.340833	-0.868750	-0.006667	1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000
-0.868750	-0.006667	1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167
-0.006667	1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417
1.101250	-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417	-0.747917
-0.802500	-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417	-0.747917	-0.327500
-0.674167	0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417	-0.747917	-0.327500	1.257917
0.876667	-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417	-0.747917	-0.327500	1.257917	-0.071667
-0.107500	-0.538750	1.001250	-0.325000	-0.239167	0.690417	-0.747917	-0.327500	1.257917	-0.071667	-0.295000

Figure 4-42 - Celsius series with the most recent lag features.

After getting the dataset ready, the whole process to calculate feature importance scores and feature selection ranks was repeated just like it has been done in the feature selection of all variables.

Below, a table with all the features and their respective feature importance scores.

	feature_name	feature_importance
1	t-6	0.4419282600796429
2	t-12	0.10108826220735137
3	t-1	0.05713078305908402
4	t-18	0.03564816920277953
5	t-24	0.027242607066767042
6	t-3	0.022843294690187515
7	t-2	0.022450899754609097
8	t-8	0.021118882030018257
9	t-5	0.021089677294109474
10	t-4	0.019849658853152298
11	t-17	0.017902561755024957
12	t-30	0.014561126246959209
13	t-10	0.013471641322147704
14	t-7	0.012190946848037263
15	t-20	0.011878780215151366
16	t-25	0.011037964641928046
17	t-32	0.010307646439970154
18	t-9	0.010167439904924885
19	t-22	0.009695718088021242
20	t-19	0.009584014333189084
21	t-26	0.009046256137855434
22	t-23	0.0089523432244175
23	t-29	0.008459942621087383
24	t-11	0.00795209594331485
25	t-35	0.007456117572190383
26	t-21	0.007274051160624732
27	t-13	0.00693056737168117
28	t-16	0.006784707965575773
29	t-27	0.00641357982371849
30	t-15	0.006106302796444232
31	t-33	0.0060997904192802216
32	t-28	0.006078136759434626
33	t-31	0.005835577951411555
34	t-36	0.005489345527437685
35	t-14	0.00536038431368334
36	t-34	0.004572466378786873

Table 4 - Features and feature importance scores.

Next, two plots with feature importance scores, the first one with all the features and the second one with the top lag feature by importance (t-6) removed.

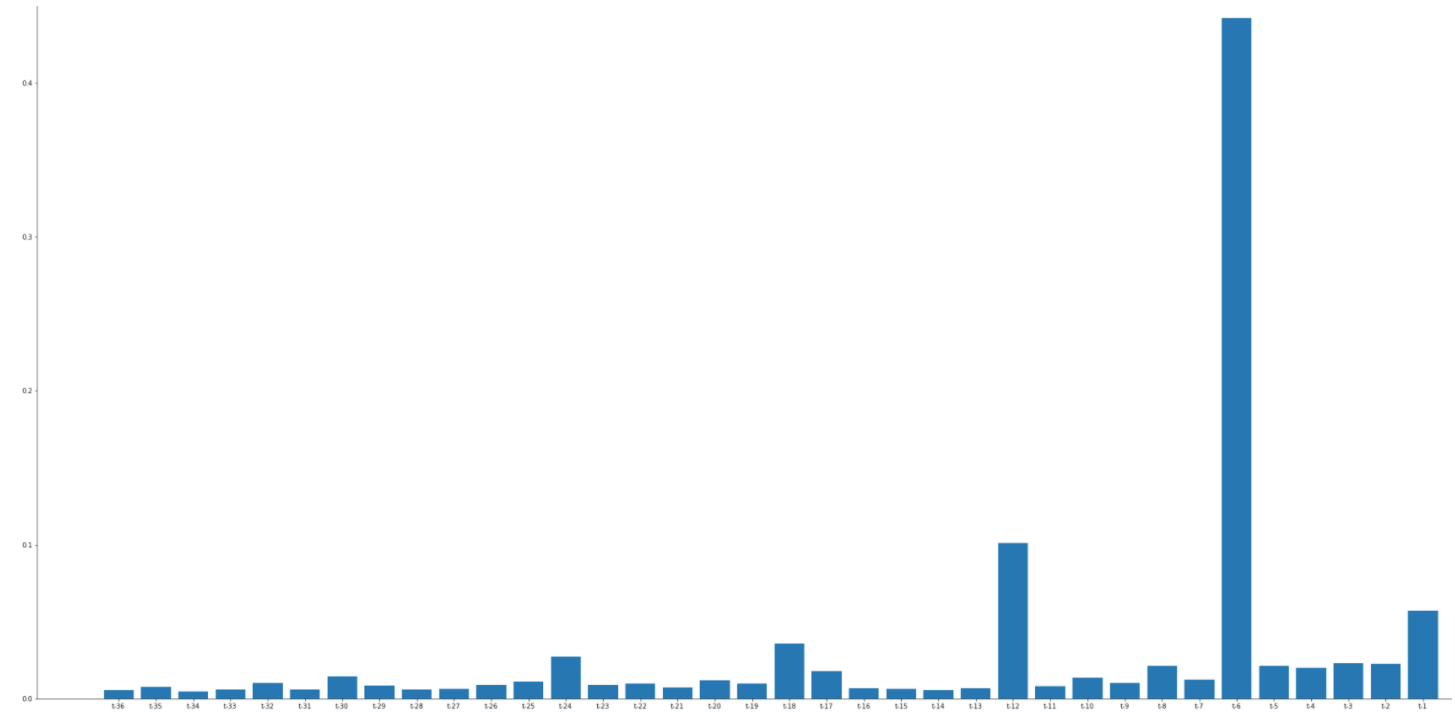


Figure 4-43 - Features x Feature Importance Scores.

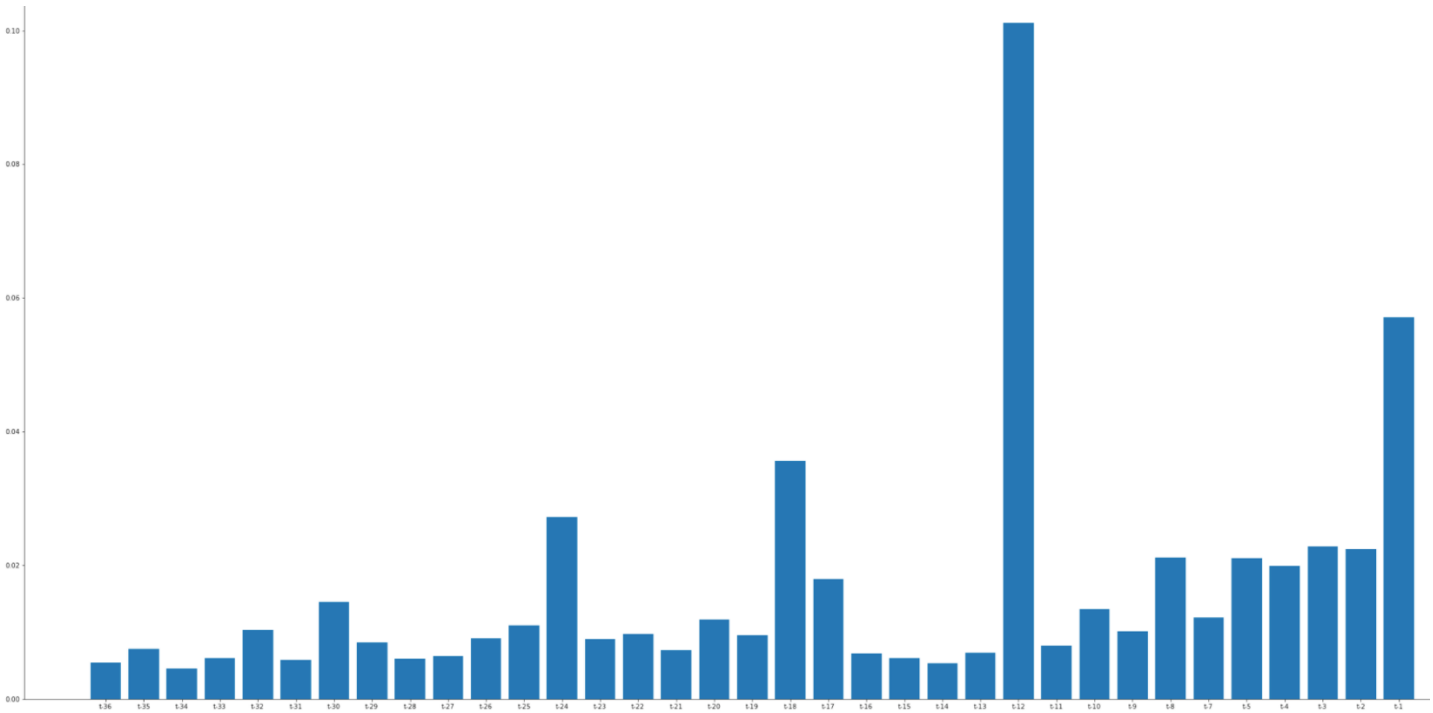


Figure 4-44 - Features x Features Importance Scores (without top feature t-6).

Below, the list of selected features and a plot with their ranks.

```
Selected Features:
t-18
t-12
t-6
t-3
t-2
t-1
```

Figure 4-45 - List of selected features by the RFE algorithm.

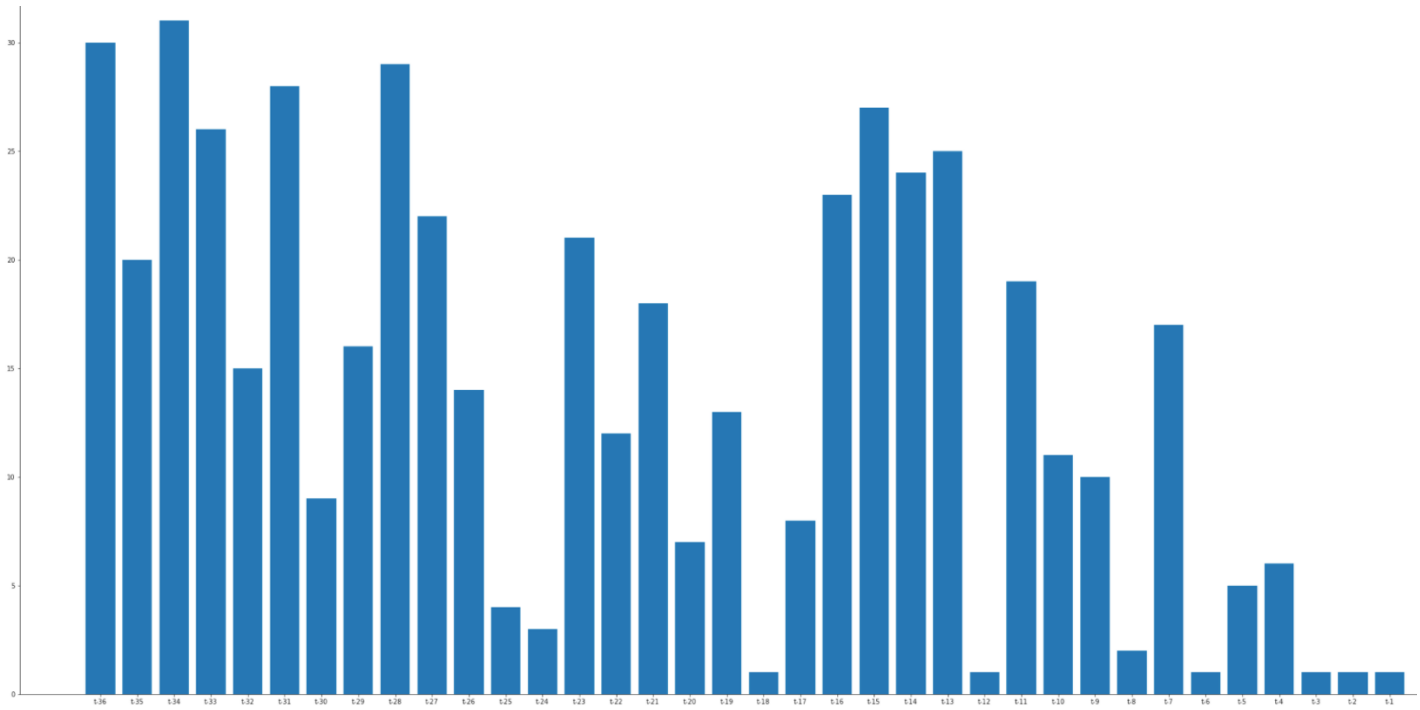


Figure 4-46 - Features x Feature Ranks by RFE algorithm (lower is better).

After finishing the Complementary Feature Selection of Lag Features part, the dataset was presented as follows:

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius_t-18	Celsius_t-12	Celsius_t-6	Celsius_t-3	Celsius_t-2	Celsius_t-1	Celsius
0	201907281	28	209	6	5.082473	3.413125	1.541917	55.525833	999.770833	21.9	27.807500	28.084583	28.247500	29.022917	28.272500	28.589583	27.791667
1	201907282	28	209	6	5.317046	3.471208	1.548208	55.203750	999.208333	21.9	27.643750	27.750833	27.804167	28.272500	28.589583	27.791667	27.389167
2	201907283	28	209	6	4.183874	3.458500	4.491792	55.701250	1000.066667	21.9	28.833333	29.134167	27.870417	28.589583	27.791667	27.389167	29.032917
3	201907284	28	209	6	5.852645	4.210250	1.996042	52.837500	999.620833	21.9	28.952917	28.427083	29.022917	27.791667	27.389167	29.032917	28.927917
4	201907285	28	209	6	5.976042	3.271208	1.743875	48.847083	999.404167	21.9	28.672917	28.226250	28.272500	27.389167	29.032917	28.927917	28.604167

Figure 4-47 - Final dataset for Machine Learning models.

In the end, the final dataset had 1.098 observations, ten features, six lag features, and the target variable. This dataset was later used with the Machine Learning models other than Deep Learning.

4.3.9. Deep Learning Models

Different types of LSTMs were used in an attempt to get the best model possible: univariate LSTM (to set a baseline), multivariate LSTM with multiple input series, and multivariate LSTM with multiple parallel series.

The steps of data preparation, model training, and model evaluation were repeated in the same way for the three different types of LSTMs.

4.3.9.1. DL Model Types

Univariate Model

Univariate LSTMs are characterized by having only one feature. In this case, the dataset was reduced from the eleven features chosen in the Feature Selection step for Deep Learning models to only one, the feature Celsius. The feature Celsius performed both holes of input variable and predict variable, the target.

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius
0	201907251	25	206	3	1.788279	4.369229	0.942625	58.362500	1001.112500	22.6	27.807500
1	201907252	25	206	3	5.026167	3.493708	1.172792	58.569583	1001.637500	22.6	27.643750
2	201907253	25	206	3	6.090000	3.213125	1.177625	55.735000	1002.016667	22.6	28.833333
3	201907254	25	206	3	5.844917	3.753125	1.610792	56.799583	1002.345833	22.6	28.952917
4	201907255	25	206	3	5.372125	3.532375	2.511958	57.453750	1002.900000	22.6	28.672917

Figure 4-48 - Final dataset for DL models. Celsius feature highlighted for the univariate model.

The following table presents the shapes of the different datasets for the univariate LSTM case:

Datasets		Shape (rows, columns)
Whole set		(1098, 1)
Training / Validation sets		(988, 1)
Test set		(110, 1)
Split 1	Training	(168, 1)
	Validation	(164, 1)
Split 2	Training	(332, 1)
	Validation	(164, 1)
Split 3	Training	(496, 1)
	Validation	(164, 1)
Split 4	Training	(660, 1)

	Validation	(164, 1)
Split 5	Training	(824, 1)
	Validation	(164, 1)

Table 5 - Shapes of univariate LSTM datasets.

The input sets were all in the shape (n, 1). The output sets were also in the same format, as expected in the case of a univariate problem. So, the output sets had only the predicted variable, the feature Celsius.

Multivariate Model (Multiple Inputs Series)

Multivariate LSTMs are characterized by having more than one feature for each time step. There are two main models of multivariate time series data. One of them is the multiple input series. In this case, the dataset has two or more time series and just one output time series. Therefore, the seven time-series features chosen in the Feature Selection step for Deep Learning models were used, not only the feature Celsius as in the univariate case. The feature Celsius performed both holes again of input variable and predict variable, the target. But, this time, when it came to be an input variable, it was in the shape of lag features.

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius_Input	Celsius
0	201907251	25	206	3	1.788279	4.369229	0.942625	58.362500	1001.112500	22.6	27.807500	27.807500
1	201907252	25	206	3	5.026167	3.493708	1.172792	58.569583	1001.637500	22.6	27.643750	27.643750
2	201907253	25	206	3	6.090000	3.213125	1.177625	55.735000	1002.016667	22.6	28.833333	28.833333
3	201907254	25	206	3	5.844917	3.753125	1.610792	56.799583	1002.345833	22.6	28.952917	28.952917
4	201907255	25	206	3	5.372125	3.532375	2.511958	57.453750	1002.900000	22.6	28.672917	28.672917

Figure 4-49 - Final dataset for DL models. Features highlighted for the multivariate multiple input series model.

The following table presents the shapes of the different datasets for this multivariate LSTM case:

Datasets		Shape (rows, columns)
Whole set		(1098, 8)
Training / Validation sets		(988, 8)
Test set		(110, 8)
Split 1	Training	(168, 8)
	Validation	(164, 8)
Split 2	Training	(332, 8)
	Validation	(164, 8)
Split 3	Training	(496, 8)

	Validation	(164, 8)
Split 4	Training	(660, 8)
	Validation	(164, 8)
Split 5	Training	(824, 8)
	Validation	(164, 8)

Table 6 - Shapes of multivariate multiple input series LSTM datasets.

The input sets were all in the shape (n, 8). The output sets were not in the same format, though. In this case, there are multiple input series, but only one output series. So, the output sets had only the predicted variable, the feature Celsius.

Multivariate (Multiple Parallel Series)

The other model of multivariate time series data is the multiple parallel series. In this case, the dataset has two or more time series and the output has the same number of time series. Because of that, again, the seven time-series features chosen in the Feature Selection step for Deep Learning models were used. Now, not only the feature Celsius but all time-series features performed both holes of input variable and predict variable, the targets.

	Date_Period	Day	Day_Of_Year	Day_Of_Week	C3H8	CO	NO2	Humidity	Pressure	TAVG	Celsius
0	201907251	25	206	3	1.788279	4.369229	0.942625	58.362500	1001.112500	22.6	27.807500
1	201907252	25	206	3	5.026167	3.493708	1.172792	58.569583	1001.637500	22.6	27.643750
2	201907253	25	206	3	6.090000	3.213125	1.177625	55.735000	1002.016667	22.6	28.833333
3	201907254	25	206	3	5.844917	3.753125	1.610792	56.799583	1002.345833	22.6	28.952917
4	201907255	25	206	3	5.372125	3.532375	2.511958	57.453750	1002.900000	22.6	28.672917

Figure 4-50 - Final dataset for DL models. Features highlighted for the multivariate multiple parallel series model.

The following table presents the shapes of the different datasets for this multivariate LSTM case:

Datasets		Shape (rows, columns)
Whole set		(1098, 7)
Training / Validation sets		(988, 7)
Test set		(110, 7)
Split 1	Training	(168, 7)
	Validation	(164, 7)

Split 2	Training	(332, 7)
	Validation	(164, 7)
Split 3	Training	(496, 7)
	Validation	(164, 7)
Split 4	Training	(660, 7)
	Validation	(164, 7)
Split 5	Training	(824, 7)
	Validation	(164, 7)

Table 7 - Shapes of multivariate multiple parallel series LSTM datasets.

The input sets were all in the shape (n, 7). The output sets were also in the same format. In this case, there are multiple input series and multiple output series. So, the output sets had not only the feature Celsius but all time series variables.

4.3.9.2. Model Evaluation

Largely used on different types of problems, k-fold cross-validation is not really suited for time series forecasting. That is because it ignores the temporal component. For instance, by the time a fold from the middle of the dataset is taken as the validation dataset, the training dataset remains with a considerable data gap. So, to evaluate time series forecasting models consistently, a TimeSeriesSplit object from Python's library scikit-learn was used to create multiple train-test splits.

The TimeSeriesSplit object was applied to create five splits that differ in the number of observations used to train the model. Each following split has a larger training dataset to work with as can be seen in the following picture. The validation dataset remains with the same amount of data on every split:

```
Split #1:
Total obs.: 332
Train shape: (168, 1)
Valid shape: (164, 1)

Split #2:
Total obs.: 496
Train shape: (332, 1)
Valid shape: (164, 1)

Split #3:
Total obs.: 660
Train shape: (496, 1)
Valid shape: (164, 1)

Split #4:
Total obs.: 824
Train shape: (660, 1)
Valid shape: (164, 1)

Split #5:
Total obs.: 988
Train shape: (824, 1)
Valid shape: (164, 1)
```

Figure 4-51 - Five splits of the dataset for a univariate LSTM NN.

As an example, below, the five splits from the univariate LSTM neural networks:

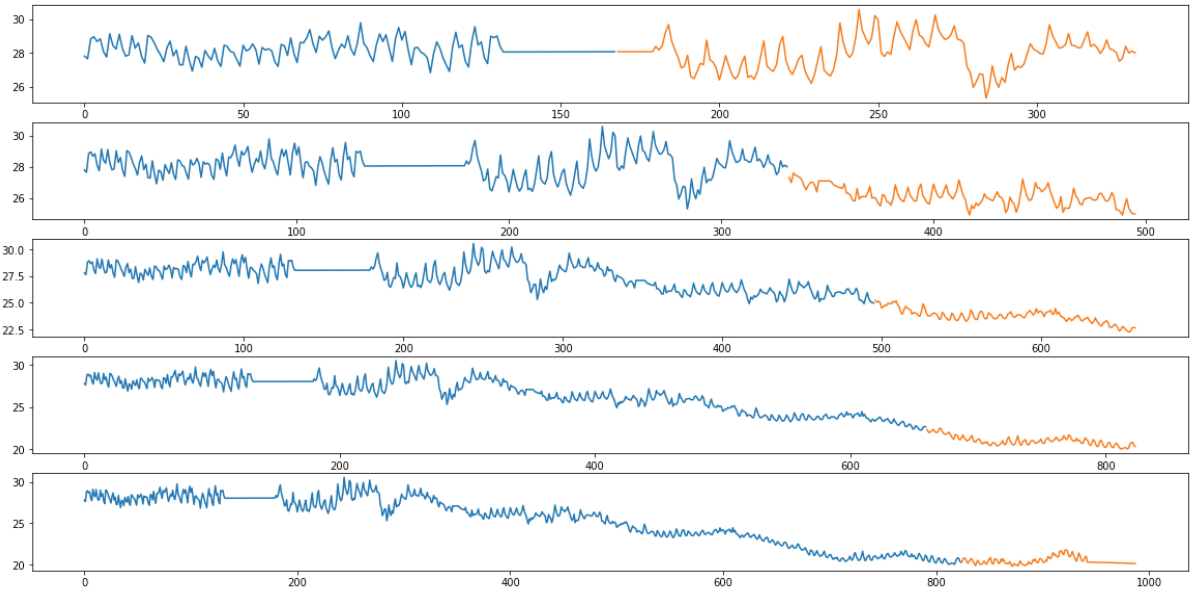


Figure 4-52 - Plots of five splits of the dataset for a univariate LSTM NN. Each following split with a little more data for training than the previous.

Before TimeSeriesSplit object be used to created multiple training and validation sets, a test set was split out of the dataset as a holdout. The idea is to utilize training and validation sets as a kind of time-series cross-validation and to have the test set as a totally impartial set of unseen data to do a final model evaluation. So, the dataset was first split into 90% to training/validation set and 10% to

test set and, later, among the 90% of the dataset designated to training and validation, five splits were created using TimeSeriesSplit.

4.3.9.3. Data Transformation

The Data Transformation step objective is to apply some transformations to the dataset that can maximize the capacity of the model to extract knowledge from the dataset. Since the dataset has variables with very different scales, mainly, the Pressure feature, a normalization object called MinMaxScaler from Python's library scikit-learn was used to rescale consistently the time series.

Below, a line plot of all the input variables from the multivariate LSTM neural network. Feature Pressure, in the range of thousands, is clearly on a different scale than all other variables.

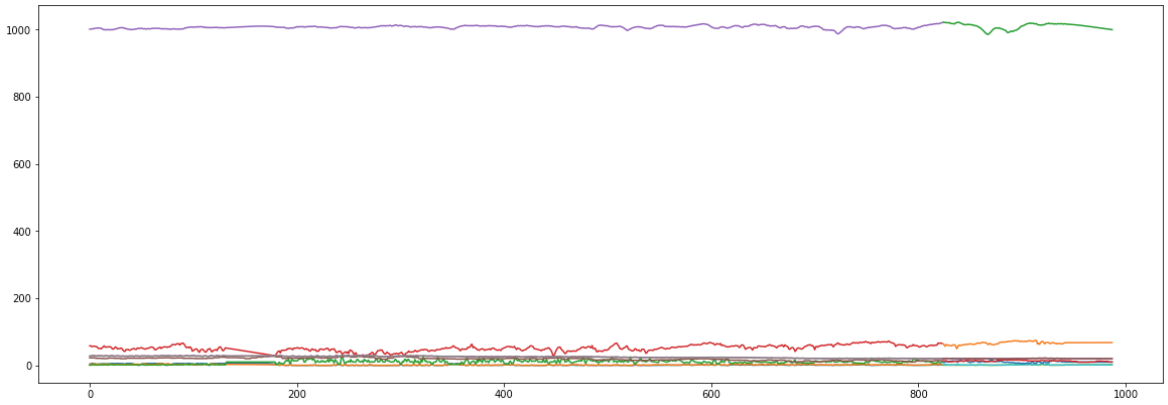


Figure 4-53 - Line plot of all the input variables with feature Pressure on a different scale.

When applying a normalization technique to a dataset, it has to be done carefully. Otherwise, the process could have data leakage. Data leakage occurs when data from outside the training data is used to train the model resulting in overfitting and an optimistic evaluation of the model (Brownlee, 2016). If the entire dataset is normalized at once and then splits for training and validation are created, scale factors, like min and max values of the series, will be from the whole dataset. Therefore, information from the validation set will be used in the training phase and vice-versa, resulting in data leakage. To avoid this situation, normalization was done within each split separately for training and validation sets. The same approach was done, later, with the test set.

4.3.9.4. Data Preparation

The goal of the Data Preparation step is to get the dataset ready to be used for training an LSTM neural network. To do so, the data must be presented in a three-dimensional array with the following components: samples, time steps, and features (Brownlee, 2018a, p. 46). Samples are the observations, the rows in the dataset. Features are the input and output variables, the columns in the dataset. A time step is one point of observation in the sample.

Below, an example of the data prepared for a univariate LSTM neural network as a three-dimension array [samples, time steps, features]. In this case, the training set has 684 samples, 18 time steps, and only one feature, since it is a univariate problem. For 684 times, only one feature (one column) is presented with its 18 lag features.

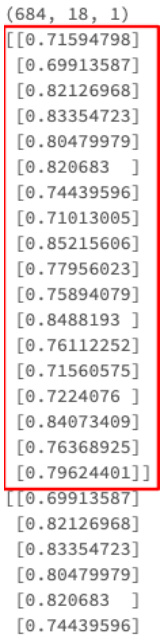


Figure 4-54 – Data prepared for a univariate LSTM NN. Highlighted the first of 684 samples with its 18 time steps and one feature.

To put in perspective, below, another example, but for a multivariate LSTM neural network. This time, the training set has 685 samples, 18 time steps, and 7 features, since it is a multivariate problem. For 685 times, 7 features (columns) are presented with its 18 lag features.

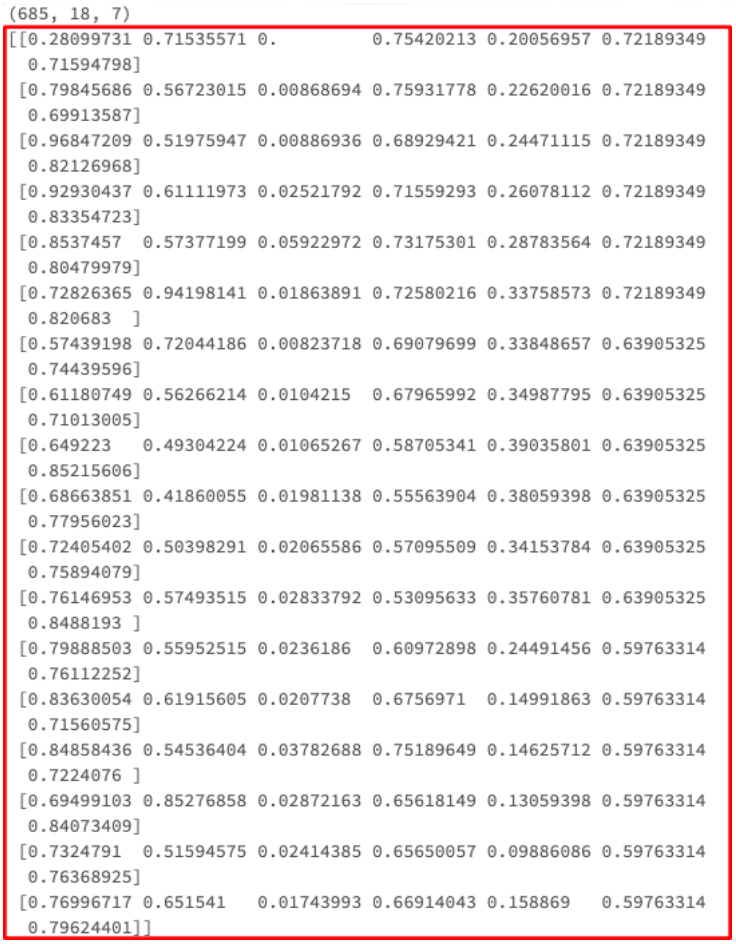


Figure 4-55 - Data prepared for a multivariate LSTM NN. Highlighted the first of 685 samples with its 18 time steps and seven features.

4.3.9.5. Define, Compile, Fit, Evaluation and Predict

After setting a time series cross-validation like evaluation process, transforming the dataset through normalization, and preparing the dataset to feed an LSTM neural network, the deep learning models were trained. But first, the number of steps was set to 18, following the indication from the Feature Selection step that lag features until the 18th time step still affect the target variable considerably.

The process of training the Deep Learning models was divided into five steps: define the neural network architecture, compile the neural network, fit the training data, evaluate with the validation data, and predict.

To define the neural networks different architectures were used. Below, a brief explanation of each one of them:

- Baseline
  - 1 hidden layer with 12 neurons.

- Wider NN (Neural Network)

        - 1 hidden layer with 25 neurons.

- Wider 2x NN
    - 1 hidden layer with 50 neurons.

- Deeper NN
    - 2 hidden layers with 25 neurons.

- Deeper 2x NN
    - 4 hidden layers;
    - First and last hidden layers with 25 neurons;
    - Second and third hidden layers with 35 neurons.

- Deeper 4x NN
    - 8 hidden layers;
    - First and last hidden layers with 25 neurons;
    - All the in-between layers with 35 neurons.

- Deeper 4x NN with Dropout Regularization
    - 8 hidden layers;
    - First and last hidden layers with 25 neurons;
    - All the in-between layers with 35 neurons;
    - 20% of dropout between every hidden layer.

The steps of compiling the neural network, fitting the training data, evaluating with the validation data, and predicting were repeated in the same way for all different neural network topologies.

Below, as an example from the univariate LSTM NN, the initial output of the model training:

```
Split #1:
TrainX shape: (150, 18, 1) / TrainY shape: (150, 1)
ValidX shape: (146, 18, 1) / ValidY shape: (146, 1)
Epoch 1/100
150/150 - 2s - loss: 0.0418 - val_loss: 0.0268
Epoch 2/100
150/150 - 2s - loss: 0.0342 - val_loss: 0.0263
Epoch 3/100
150/150 - 2s - loss: 0.0343 - val_loss: 0.0281
Epoch 4/100
150/150 - 2s - loss: 0.0329 - val_loss: 0.0257
Epoch 5/100
150/150 - 2s - loss: 0.0333 - val_loss: 0.0291
Epoch 6/100
150/150 - 2s - loss: 0.0351 - val_loss: 0.0287
Epoch 7/100
150/150 - 2s - loss: 0.0329 - val_loss: 0.0262
Epoch 8/100
150/150 - 2s - loss: 0.0336 - val_loss: 0.0256
Epoch 9/100
150/150 - 2s - loss: 0.0319 - val_loss: 0.0296
Epoch 10/100
150/150 - 2s - loss: 0.0355 - val_loss: 0.0258
Epoch 11/100
150/150 - 2s - loss: 0.0323 - val_loss: 0.0272
Epoch 12/100
150/150 - 2s - loss: 0.0326 - val_loss: 0.0301
Epoch 13/100
150/150 - 2s - loss: 0.0315 - val_loss: 0.0325
Epoch 14/100
150/150 - 2s - loss: 0.0315 - val_loss: 0.0295
Epoch 15/100
150/150 - 2s - loss: 0.0304 - val_loss: 0.0300
```

Figure 4-56 - Initial output of a univariate LSTM NN training.

Next, also as an example from the univariate LSTM NN training, charts of model loss from model training of the five splits (training and validation losses by epoch):

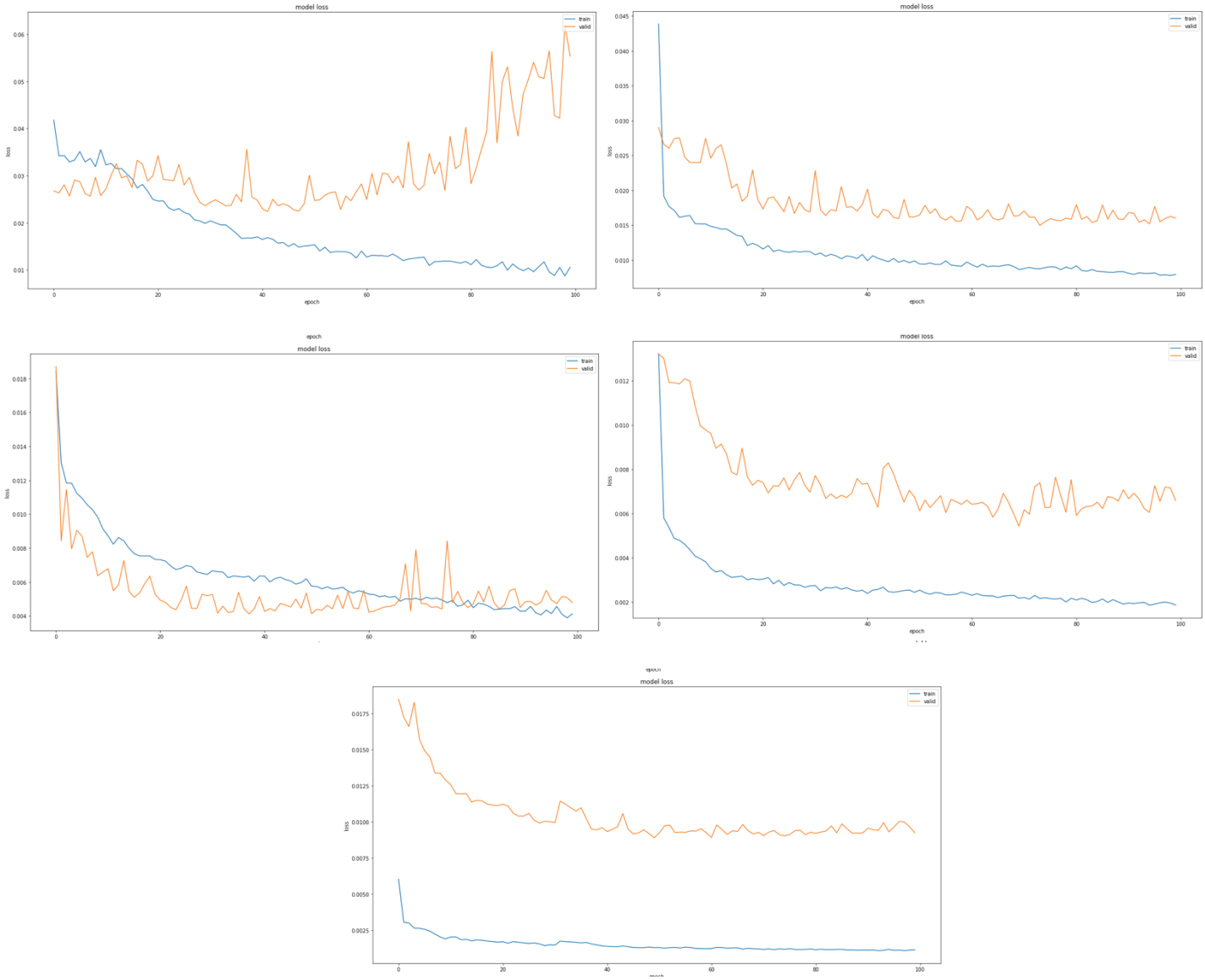


Figure 4-57 - Model Losses (Split #1 to #5).

4.3.9.6. Performance Metric

After training and evaluation of the Deep Learning models, predictions were made using the training and the validation datasets. The objective of this is to calculate the performance metric of each dataset by comparing the respective predictions to the already known values of the target variables. The performance metric used for that was the Root Mean Squared Error.

Before calculating the RMSE, a minor transformation step was required. Since the training and the validation datasets were normalized by the MinMax technique, the predictions needed to be transformed back to the scale of the original values. Thus, and as RMSE was being used as a performance metric, it would be more noticeable if the error was of 1 Celsius degree, more or even less. By using the inverse_transform function of the MinMaxScaler object from Python's library scikit-learn, it was relatively easy to invert the predictions. The only exception was in the case of the multivariate multiple input series LSTMs. Because input and output dataset shapes were different in

this case, input dataset shapes were (n, 8) and output dataset shapes were (n, 1), the inverse_transform function demanded some data transformation before be applied.

Below, as an example from the univariate LSTM NN, the training and the validation score calculated from the five splits:

```
Train Score: 0.343 RMSE (+/- 0.063)
Valid Score: 0.435 RMSE (+/- 0.405)
```

Figure 4-58 - Training and Validation Scores.

4.3.9.7. Prediction on Test Dataset

As the last step, to be sure of the generalization capability of the model, predictions were made out of the test set (holdout). The entire process was repeated, but with some small changes.

In this case, the training dataset was composed of what before was the training and the validation dataset. The new training dataset needed to have its data transformed and prepared just like before. The dataset was normalized and converted to the three-dimensional array of samples, time steps, and features that an LSTM expects. After that, the Deep Learning model was trained again, but, this time, with much more data.

The test dataset was also transformed and prepared. The test dataset was used to make predictions based on the recently trained model. Predictions were inverted following the same logic as before and the RMSE was calculated to obtain a test score.

The image below shows the results achieved, as an example with the univariate NN, with the dataset in blue, the predictions from the new training dataset in orange and the predictions from the test dataset in green:

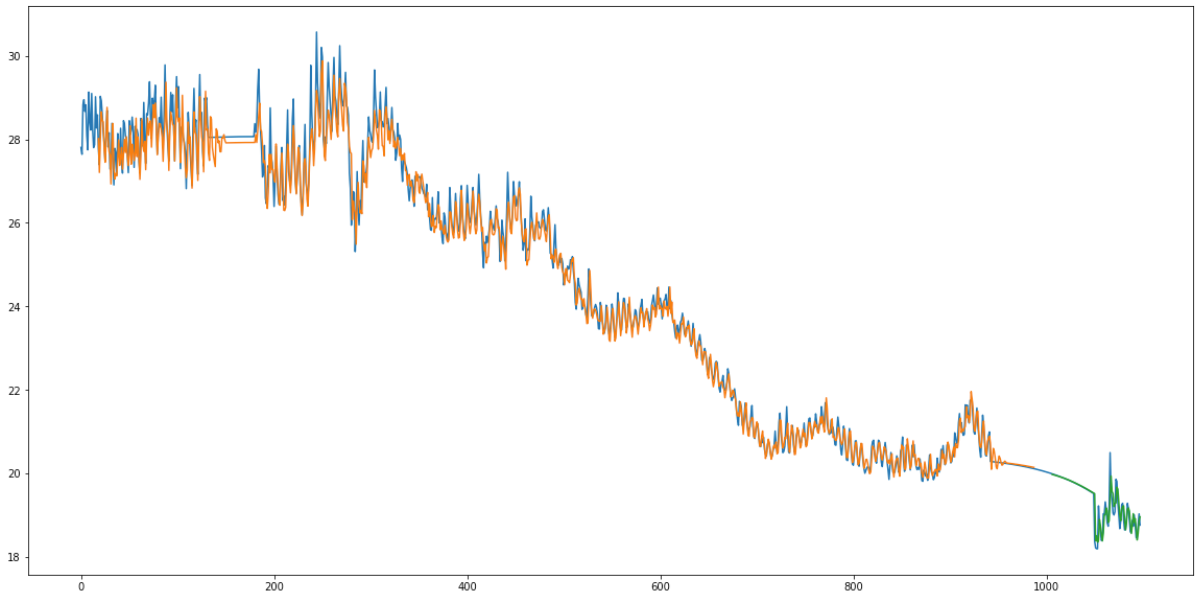


Figure 4-59 - Complete dataset overlaid with predictions.

Interesting to notice that there are no predictions for the first eighteen observations in the chart above. That is due to the eighteen time steps configured in the LSTMs.

4.3.10. Machine Learning Models

Different models were used in an attempt to get the best Machine Learning model possible to compare with Deep Learning models. Linear Regression, Decision Trees, and ensemble models like Gradient Boosting were among them. The idea was to test if Deep Learning models, more specifically, LSTM neural networks, have a great advantage or not for time series forecasting problems. Because of that, all the same criteria used with Deep Learning models were applied to Machine Learning models to have a fair comparison. It is important to remind that Machine Learning models didn't receive any special treatment to deal with a times series forecasting case, though.

Below, a table with all Machine Learning models that have been employed:

Model Acronyms	Model Type	Model Name
LR	Standard	Linear Regression
LASSO		LASSO Linear Regression
KNN		K-Nearest Neighbors
CART		Classification and Regression Trees
GBM	Ensemble	Gradient Boosting Machines (or Stochastic Gradient Boosting)

RF		Random Forest

Table 8 - Standard and Ensemble Machine Learning Models.

The steps of data transformation and model evaluation were repeated in the same way for all ML models.

4.3.10.1. Data Transformation

Just as with Deep Learning models, the dataset was also normalized using MinMaxScaler object from Python's scikit-learn library. However, it was applied differently. As a good practice and to avoid data leakage, Pipeline objects, also from the scikit-learn library, were employed to encapsulate data transformations, like the dataset normalization, within each model training and validation.

Below, pipelines being used on two different occasions, with standard Machine Learning models, and with ensemble models:

```
# Normalize the dataset.
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', MinMaxScaler()),('LR', LinearRegression())])))
pipelines.append(('ScaledLASSO', Pipeline([('Scaler', MinMaxScaler()),('LASSO', Lasso())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', MinMaxScaler()),('KNN', KNeighborsRegressor())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', MinMaxScaler()),('CART', DecisionTreeRegressor())])))
```

Figure 4-60 - Pipelines with standard machine learning models.

```
# Ensembles.
ensembles = []
ensembles.append(('ScaledGBM', Pipeline([('Scaler', MinMaxScaler()),('GBM', GradientBoostingRegressor())])))
ensembles.append(('ScaledRF', Pipeline([('Scaler', MinMaxScaler()),('RF', RandomForestRegressor(n_estimators=10))])))
```

Figure 4-61 - Pipelines with ensemble machine learning models.

The application of Pipelines was important to prevent the training set to be influenced by the scales of the validation set and vice-versa. Pipelines are capable of avoiding data leakage because they ensure that the applied data transformation is constrained to each fold of the cross-validation evaluation process (Müller & Guido, 2016, p. 310).

4.3.10.2. Model Evaluation

Similarly to what was done with Deep Learning models, the dataset was divided first between a training set and a test set with 90% and 10% as the respective percentages. So, a holdout was created for later confirmation of the generalization capability of the best ML model by a completely impartial dataset. Afterward, the training dataset was further split into training and validation sets. But now, with Machine Learning models, this last split was done differently. It was done through a K-fold cross-validation procedure.

```
results = []
names = []
for name, model in pipelines:
  kfold = KFold(n_splits=num_folds, random_state=seed)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f) --> RMSE: %f" % (name, cv_results.mean(), cv_results.std(), sqrt(abs(cv_results.mean())))
  print(msg)
```

Figure 4-62 - K-fold cross-validation procedure.

By using a KFold object and cross_val_score function from Python's scikit-learn library, the training and the validation sets were split with the number of folds set to 10. The scoring method was configured to be the Mean Squared Error, and, later, the square root of the result was calculated to obtain the Root Mean Squared Error as in the Deep Learning models.

Next, the results of model training and evaluation for standard and ensemble models:

```
ScaledLR: -0.107300 (0.016694) --> RMSE: 0.327567
ScaledLASSO: -11.485918 (0.819133) --> RMSE: 3.389088
ScaledKNN: -0.157995 (0.048525) --> RMSE: 0.397486
ScaledCART: -0.173268 (0.025285) --> RMSE: 0.416255
```

Figure 4-63 - MSE, standard deviation, and RMSE for 10-fold cross-validation of standard models.

```
ScaledGBM: -0.101337 (0.026927) --> RMSE: 0.318334
ScaledRF: -0.118631 (0.028558) --> RMSE: 0.344428
```

Figure 4-64 - MSE, standard deviation, and RMSE for 10-fold cross-validation of ensemble models.

As can be noticed from the results above, from all Machine Learning models, the ScaledGBM, which is the boosting algorithm Gradient Boosting Machines with normalized data, had the best result. Overall, ensemble models achieved better results than standard models.

Below, a comparison between the ensemble models in the form of a box and whisker plot of the MSE results from the 10-fold cross-validation:

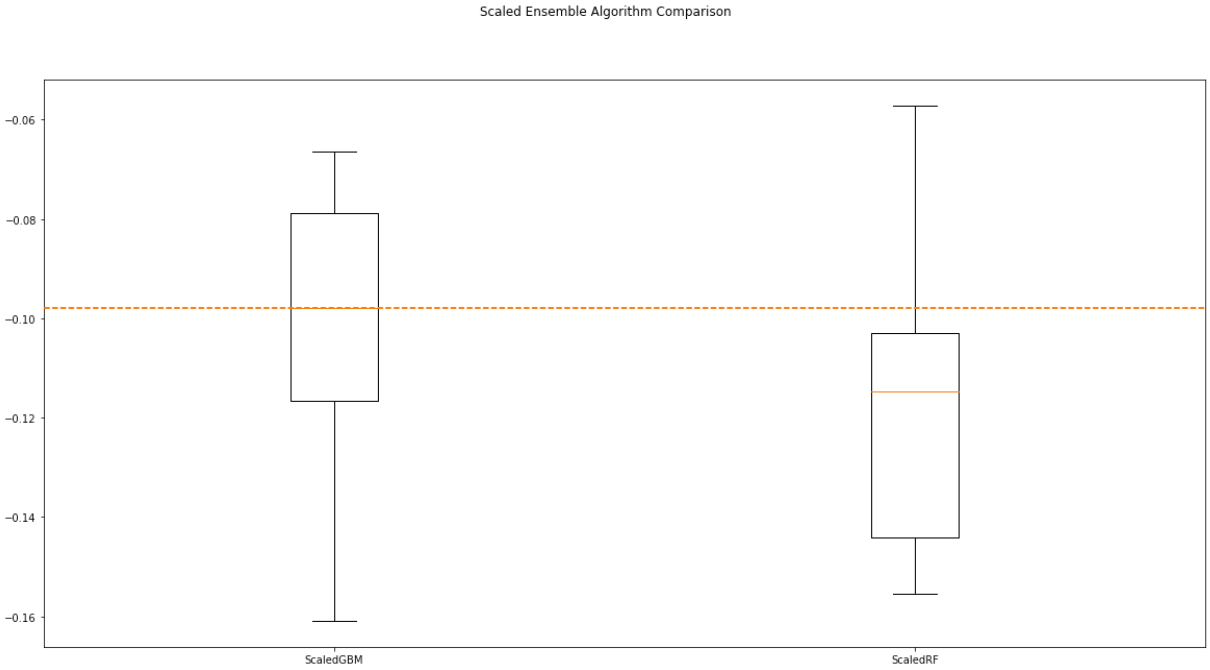


Figure 4-65 - Box and whisker plot of MSE performance metric for ensemble models.

As can be seen from the chart above, ScaledGBM had slightly better results than ScaledRF, the bagging algorithm Random Forest with normalized data, concerning both the median and the entire MSE distribution. As a whole, the Gradient Boosting Machines box and whiskers are a little bit closer to zero (top of the chart) than Random Forest, so it was selected as the best Machine Learning model to move forward to hyperparameter tuning.

4.3.10.3. Hyperparameter Tuning

Once Gradient Boosting Machines were chosen as the best ML model, its main parameter, the number of trees, was subject to hyperparameter tuning through a grid search strategy. Different numbers of trees were set in a grid to evaluate the model a couple of times more, one for each value passed. The number of trees tested was according to the following array [50, 100, 200, 300, 400, 500, 600].

```
# Tune scaled ensemble model
scaler = MinMaxScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = dict(n_estimators=numpy.array([50, 100, 200, 300, 400, 500, 600]))
model = GradientBoostingRegressor(random_state=seed)
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold, iid=True)
grid_result = grid.fit(rescaledX, Y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Figure 4-66 - Hyperparameter tuning through grid search of Extra Trees model.

Next, the results of the grid search that pointed 600 trees as the optimal number of trees for the Gradient Boosting Machines models:

```
Best: -0.094874 using {'n_estimators': 600}
-0.105931 (0.028195) with: {'n_estimators': 50}
-0.101225 (0.026700) with: {'n_estimators': 100}
-0.097626 (0.025636) with: {'n_estimators': 200}
-0.095999 (0.025116) with: {'n_estimators': 300}
-0.095395 (0.024642) with: {'n_estimators': 400}
-0.095358 (0.024645) with: {'n_estimators': 500}
-0.094874 (0.024625) with: {'n_estimators': 600}
```

Figure 4-67 - Grid search results as MSE, standard deviation, and the number of estimators.

With the cross-validation model evaluation method having selected Gradient Boosting Machines as the best ML model and with the grid search tuning strategy having indicated 600 as an optimal number of trees, predictions based on the test dataset could be calculated with this setup.

4.3.10.4. Prediction on Test Dataset

Ultimately, predictions were made out of the test set (holdout) to evaluate the generalization capability of the model. The whole process of data transformation, model training, and model evaluation was performed again. But, this time, using only the selected best model (Gradient Boosting Machines) with its optimal configuration (600 trees).

```
# Prepare the model.
scaler = MinMaxScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = GradientBoostingRegressor(random_state=seed, n_estimators=600)
model.fit(rescaledX, Y_train)
# Transform the validation dataset.
scaler = MinMaxScaler().fit(X_validation)
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
print('RMSE: %f' % sqrt(mean_squared_error(Y_validation, predictions)))
```

Figure 4-68 - Predictions based on the test dataset.

Below, the result of the best Machine Learning model and setup in RMSE:

```
RMSE: 0.569068
```

Figure 4-69 - The best result in RMSE for Machine Learning models.

5. RESULTS

After preparing the data and training and evaluating Deep Learning models and other Machine Learning models, it was possible to compare the results to understand if Deep Learning models would outperform or not other Machine Learning models in a time series forecasting practical problem. But first, before comparing Deep Learning models with other Machine Learning models, let focus on how different types of Deep Learning models compared to each other.

Below, the table of results of different types of Deep Learning models:

Model Type	Version	Name	Description	Train Score (RMSE)	Train Std.	Valid Score (RMSE)	Valid Std.
Univariate	2.1	Baseline	1 layer / 12 nodes	0,433	0,065	0,379	0,288
Univariate	2.2	Wider NN	1 layer / 25 nodes	0,387	0,033	0,375	0,271
Univariate	2.3	Wider 2x NN	1 layer / 50 nodes	0,343	0,063	0,435	0,405
Univariate	2.4	Deeper NN	2 layers / 25 nodes	0,345	0,052	0,536	0,571
Univariate	2.5	Deeper 2x NN	4 layers / 25 or 35 nodes	0,346	0,021	0,398	0,267
Univariate	2.6	Deeper 4x NN	8 layers / 25 or 35 nodes	0,461	0,2	0,479	0,306
Univariate	2.7	Dropout	8 layers / 25 or 35 nodes / dropout hidden layers	0,598	0,114	0,52	0,285
Multivar (Multiple Inputs)	2.1	Baseline	1 layer / 12 nodes	0,026	0,018	0,083	0,127
Multivar (Multiple Inputs)	2.2	Wider NN	1 layer / 25 nodes	0,028	0,018	0,029	0,029
Multivar (Multiple Inputs)	2.3	Wider 2x NN	1 layer / 50 nodes	0,013	0,005	0,028	0,038
Multivar (Multiple Inputs)	2.4	Deeper NN	2 layers / 25 nodes	0,021	0,003	0,041	0,038
Multivar (Multiple Inputs)	2.5	Deeper 2x NN	4 layers / 25 or 35 nodes	0,066	0,033	0,269	0,452
Multivar (Multiple Inputs)	2.6	Deeper 4x NN	8 layers / 25 or 35 nodes	0,241	0,211	0,379	0,415
Multivar (Multiple Inputs)	2.7	Dropout	8 layers / 25 or 35 nodes / dropout hidden layers	0,529	0,025	0,572	0,318
Multivar (Parallel Series)	2.1	Baseline	1 layer / 12 nodes	0,458	0,015	0,486	0,307
Multivar (Parallel Series)	2.2	Wider NN	1 layer / 25 nodes	0,416	0,031	0,577	0,468
Multivar (Parallel Series)	2.3	Wider 2x NN	1 layer / 50 nodes	0,303	0,049	0,718	0,677
Multivar (Parallel Series)	2.4	Deeper NN	2 layers / 25 nodes	0,388	0,032	0,86	0,883
Multivar (Parallel Series)	2.5	Deeper 2x NN	4 layers / 25 or 35 nodes	0,409	0,041	0,569	0,258
Multivar (Parallel Series)	2.6	Deeper 4x NN	8 layers / 25 or 35 nodes	0,517	0,012	0,624	0,298
Multivar (Parallel Series)	2.7	Dropout	8 layers / 25 or 35 nodes / dropout hidden layers	0,722	0,125	0,579	0,246

Table 9 - Results for Deep Learning models.

As can be seen from the validation scores from the table above (column Valid Score (RMSE)), the LSTM networks with multivariate multiple inputs series have lower values of RMSE overall, demonstrating that it performed better than the univariate or the multivariate multiple parallel series models. It is important to remark that the train and the validation scores were obtained through the average of five different executions done by the time series dataset split. This allowed the evaluation of the models in different segments of the dataset representing as much as possible different scenarios.

Focusing on the multivariate multiple inputs series models alone, the LSTM networks with just one hidden layer with 50 nodes in its network topology performed better than all the others. But, not by much if compared to the similar network configuration of one hidden layer with 25 nodes. Analyzing the validation scores and the validation standard deviations (column Valid. Std.), it is possible to say that both models were tied as the best Deep Learning model.

Now that we have the best results from Deep Learning models, let compare Deep Learning models with other Machine Learning models. As was just shown, the best results from Deep Learning models came from LSTM networks with multivariate multiple input series with one hidden layer and 25 or 50 nodes. From the Machine Learning experiments, the best result was obtained by the ensemble boosting method Gradient Boosting Machines with 600 trees in its configuration.

Below, the table presenting the final results:

Model Type	Model	Configuration	RMSE
Boosting Ensemble Method	Gradient Boosting Machines	600 trees	0,569
Deep Learning Neural Network	Long Short-Term Memory Network	1 hidden layer with 50 nodes	0,028

Figure 5-1 - Best results for Deep Learning and other Machine Learning models.

As can be noticed by comparing RMSE performance metrics, Deep Learning models indeed have a better performance when it comes to time series forecasting problems. However, it is important to remark that other Deep Learning models like the univariate and the multivariate multiple parallel series variations besides achieving overall better performance than other Machine Learning models did not accomplish that by such large margin as was the case of multivariate multiple input series models.

6. CONCLUSIONS

Several algorithms from different areas of Machine Learning were tested in an attempt to find the best possible model to solve the problem of time series forecasting for the Ambiosensing system. After many experiments, it became clear that the multivariate LSTM networks in a configuration of multiple variables as input and one variable as the target would be the best choice for the operational use case problem of the Ambiosensing system. By applying multivariate LSTM networks with a single hidden layer with 50 nodes the Ambiosensing system would predict with very high precision (RMSE of 0,028 Celsius degrees) the average temperature for the next four hours allowing the system to set its configurations adequately.

Taking into consideration only Deep Learning models, it was shown that, by a large margin, multiple input series models outperformed univariate and multivariate parallel series models (RMSE of 0,028 versus RMSE of 0,375 and 0,486 respectively). The reason to make better predictions than the univariate variations is, probably, due to the extra features that the multiple input series models take into consideration. With more information available, the model can learn more and, hence, make better predictions. Still, multiple parallel series models have the same features available as multiple input series models, but the fact that these models intend to predict not only one but several time series make it underperform when compared to multiple input series models that have only one target to predict.

It is important to remark, still concerning only Deep Learning models, that models that achieved the best performances were shallow networks with a single hidden layer. Deeper networks did not make as assertive predictions even when they were combined with regularization techniques to avoid overfitting like dropout. This is an indication that, probably, the problem of predicting indoor temperature for the Ambiosensing system is not as complex as initially thought. At least, as long as the right features are available with a consistent data quality to train the models.

Concerning other Machine Learning models other than Deep Learning, even with a set of features that mixed lag features among the selected features, they did not perform as good as multivariate LSTM networks (RMSE of 0,569 versus RMSE of 0,028 respectively). The lag features did bring time context to the Machine Learning models, but these models did not treat them as the same variable in different time steps like the LSTM networks. The Machine Learning models treated the lag features as completely different input variables. This is an important advantage for LSTM networks when compared to other Machine Learning models that are not as suited as LSTM networks to deal with large sequences of data. Ensemble methods did improve the performance between Machine Learning models, but not enough to beat LSTM networks in any configuration. But, the difference between the best Ensemble method, the Gradient Boosting Machines (RMSE of 0,569), and the second-best LSTM networks configuration, the univariate LSTM network with a single hidden layer with 25 nodes (RMSE of 0,375), was not that big.

If only the best result for each tested algorithm is considered, even the worst model still would be satisfactory to be applied in the use case of predicting indoor temperature for the Ambiosensing system. A 0,5 Celsius degree error would still be acceptable and would not affect drastically the ability of the Ambiosensing system to set its configurations autonomously.

An important final remark is that in an attempt to improve the performance of the Deep Learning models with LSTM networks, the dataset was transformed through differencing into a stationary time series. By analyzing the complete dataset line plot, it is possible to observe a decreasing trend in indoor temperature from July to January, corresponding with the weather changes from summer to winter in the northern hemisphere. However, removing the trend from the time series did not result as expected. The results were worse than the ones with the original dataset. A possible explanation is that the Neural Network learned the trend by the context provided in the input sequence eliminating the need to remove it from the time series.

7. PRACTICAL APPLICATIONS AND RECOMMENDATIONS FOR FUTURE WORK

The use case chosen to test and compare different models was the operational scenario of predicting the indoor temperature from historical data to support the Ambiosensing system to automatically set the best possible base configuration to prevent high energy consumption from drastic changes. However, the results showed that Deep Learning models like LSTM networks can be used for much more complex problems using the same dataset and the context of the Ambiosensing system.

Another possible application that can take advantage of the several input features related to gas concentrations and also from the unique feature of LSTM networks of handling well large sequences of data is to apply the model predictive capability to anomaly detection. A more specific variation of this use case could be a fire or explosion or danger alert module in the Ambiosensing system based on the dangerous concentration of certain gases like the flammable Methane or the toxic Carbon Monoxide.

As future work, although LSTM networks did achieve great results, it is possible to invest in some techniques to improve it even more. One of them would be lifting the performance by adapting the learning rate. Decreasing the learning rate over time during training could increase the performance and also reduce the training time. In the experiments, the Adam (Adaptive Moment Estimation) optimization algorithm was used. The Adam optimization algorithm in its default Keras configuration provides learning rate adaptation automatically. However, it would be interesting to explore further the parameter learning rate for the Adam optimization algorithm and also other optimization algorithms like the classic Stochastic Gradient Descent or the RMSProp (Root Mean Square Propagation) with different learning rate configurations.

Another possible future work also in the context of improving LSTM networks would be to experiment with different types of LSTM networks. For instance, bidirectional LSTMs or even combinations of LSTM and CNN networks. Bidirectional LSTM networks learn both from forward and backward passes. And LSTM and CNN networks can work together in a hybrid model joining the strengths of each algorithm. In a specific LSTM-CNN configuration, a CNN model could extract and learn features from the sequence of data and an LSTM model could work as a backend that would receive the data initially processed by the CNN model.

Deep Learning is a vast field with lots of possible practical applications. Because of the way that Neural Networks learn, with enough computational resources available, they should outperform classical linear forecasting methods in general. Mostly, due to its capability of mapping complex relationships between inputs and outputs. This characteristic together with the support for input sequences in RNN with stable gradients in LSTM networks granted efficiency in learning temporal dependencies from the input to the output. All these aspects together make LSTM networks a good fit for times series forecasting.

8. BIBLIOGRAPHY

Adafruit Sensirion SHT31-D (Temperature and Humidity Sensor). (n.d.). Retrieved August 6, 2020, from Adafruit Industries website: https://www.adafruit.com/product/2857

Aggarwal, C. (2015). *Data Mining: The Textbook*. Springer.

Ambiosfera. (n.d.). Retrieved June 8, 2019, from Ambiosfera - Energia e Ambiente website: http://www.ambiosfera.pt/

Barooah, P., Subramany, R., & Siddarth, G. (2018). *Patent No. US 10,047,968 B2*. United States of America: United States Patent.

Bell, J. (2015). *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley & Sons, Inc.

Bernardo, J. (2015). Estratégia para a Eficiência Energética nos Edifícios Públicos. In *Workshop: RePublic_ZEB*. Retrieved from http://www.lneg.pt/download/10887/DGEG_REPublic_ZEB 11dez2015.pdf

Bonér, J., Chala, A., Ruzicka, W., Bennet, L., Kappal, S., & Smith, T. (2018). Big Data: Stream Processing, Statistics, and Scalability. *DZone Devada Media*, 40. Retrieved from https://dzone.com/guides/big-data-stream-processing-statistics-and-scalabil

Brownlee, J. (2016). Data Leakage in Machine Learning. Retrieved August 7, 2020, from https://machinelearningmastery.com/data-leakage-machine-learning/

Brownlee, J. (2018a). *Deep Learning for Time Series Forecasting Predict the Future with MLPs, CNNs and LSTMs in Python* (v1.4).

Brownlee, J. (2018b). *Deep Learning with Python Develop Deep Learning Models on Theano and TensorFlow Using Keras* (v1.15).

Brownlee, J. (2018c). *Time Series Forecasting with Python How to Prepare Data and Develop Models to Predict the Future* (v1.6).

Brownlee, J. (2019). *Machine Learning Mastery with Python Understand Your Data, Create Accurate Models and Work Projects End-to-End* (v1.14).

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*.

Chala, A., Polak, A., Laird, C., Lamb, C., Stichbury, J., Babu, S., & Smith, T. (2019). Big Data: Volume, Variety, and Velocity. *DZone Devada Media*, 38. Retrieved from https://dzone.com/guides/big-data-volume-variety-and-velocity

Dalgaard, P. (n.d.). *Introductory Statistics with R* (Second Edition). Springer.

Dinov, I. (2018). *Data Science and Predictive Analytics Biomedical and Health Applications using R*. Springer Nature.

Direcção Geral de Energia e Geologia. (2017). *Balanços Energético Sintético 2016*. Retrieved from http://www.dgeg.gov.pt/wwwbase/wwwinclude/ficheiro.aspx?access=1&id=15901

Evans, R., & Gao, J. (2016). DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. Retrieved August 6, 2020, from https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/

Google Cloud Public Datasets. (n.d.). Retrieved August 7, 2020, from Google Cloud website: https://cloud.google.com/public-datasets

Granville, V. (2017). Difference between Machine Learning, Data Science, AI, Deep Learning, and Statistics. Retrieved August 6, 2020, from Data Science Central website: https://www.datasciencecentral.com/profiles/blogs/difference-between-machine-learning-data-science-ai-deep-learning

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer.

Groth, P. (2017). 5 reasons data is a key ingredient for AI applications. Retrieved August 6, 2020, from Elsevier website: https://www.elsevier.com/connect/5-reasons-data-is-a-key-ingredient-for-ai-applications

Halevy, A., Norvig, P., & Pereira, F. (2009). *The Unreasonable Effectiveness of Data*. Retrieved from www.computer.org/intelligent

Hardesty, L. (2015). Siting wind farms more quickly, cheaply. Retrieved August 6, 2020, from MIT News Office website: http://news.mit.edu/2015/siting-wind-farms-quickly-cheaply-0717

Holidays Library Documentation. (n.d.). Retrieved August 6, 2020, from https://pypi.org/project/holidays/

Humidity Sensor BME280. (n.d.). Retrieved August 6, 2020, from Bosch Sensortec website: https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/

IBM Research Editorial Staff. (2016). Forecasting the Future: Air Quality in South Africa with the IoT. Retrieved August 6, 2020, from IBM Research Blog website: https://www.ibm.com/blogs/research/2016/10/forecasting-air-quality-south-africa-iot/

IEC 60050 - International Electrotechnical Vocabulary - Details for IEV number 845-01-52: “lux.” (n.d.). Retrieved August 7, 2020, from International Electrotechnical Commission website: http://www.electropedia.org/iev/iev.nsf/display?openform&ievref=845-01-52

Igual, L., & Seguí, S. (2017). *Introduction to Data Science A Python Approach to Concepts, Techniques and Applications*. Retrieved from http://www.springer.com/series/7592

International Organization for Standardization. (1975). Retrieved August 6, 2020, from Standard Atmosphere (ISO Standard Nº 2533:1975) website: https://www.iso.org/standard/7472.html

Kirk, M. (2014). *Thoughtful Machine Learning: A Test-Driven Approach* (First Edition). O’Reilly Media, Inc.

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling* (5th ed.). Springer Nature.

Menne, M. J., Durre, I., Korzeniewski, B., McNeal, S., Thomas, K., Yin, X., … Houston, T. G. (2012). *Global Historical Climatology Network - Daily (GHCN-Daily), Version 3.26*. https://doi.org/http://doi.org/10.7289/V5D21VHZ

Müller, A., & Guido, S. (2016). *Introduction to Machine Learning with Python A Guide for Data Scientists* (First Edition). O'Reilly Media Inc.

NumPy Manual. (n.d.). Retrieved August 6, 2020, from https://numpy.org/doc/stable/

Oldewurtel, F., Parisio, A., Jones, C. N., Gyalistras, D., Gwerder, M., Stauch, V., … Morari, M. (2012). Use of model predictive control and weather forecasts for energy efficient building climate control. *Energy and Buildings*, *45*, 15–27. https://doi.org/10.1016/j.enbuild.2011.09.022

Pandas Documentation. (n.d.). Retrieved August 6, 2020, from https://pandas.pydata.org/docs/

Pisello, A. L., Bobker, M., & Cotana, F. (2012). A Building Energy Efficiency Optimization Method by Evaluating the Effective Thermal Zones Occupancy. *Energies*, *5*(12), 5257–5278. https://doi.org/10.3390/en5125257

Ruzicka, W., Lawrence, C., Chaudhri, A., Jacquet, F., & Smith, T. (2018). Artificial Intelligence: Automating Decision-Making. *DZone Devada Media*, 29.

Scikit-learn Documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/user_guide.html

Shaikh, P. H., Nor, N. B. M. ., Nallagownden, P., Elamvazuthi, I., & Ibrahim, T. (2013). Robust Stochastic Control Model for Energy and Comfort Management of Buildings. *Australian Journal of Basic and Applied Sciences*, *7*(10), 137–144.

Shmueli, G., & Lichtendahl, K. (2016). *Practical Time Series Forecasting with R: A Hands-on Guide* (Second Edition). Axelrod Schnall Publishers.

Shumway, R., & Stoffer, D. (2017). *Time Series Analysis and Its Applications with R Examples* (4th ed.). Retrieved from http://www.springer.com/series/417

Skansi, S. (2018). *Introduction to Deep Learning From Logical Calculus to Artificial Intelligence*. https://doi.org/10.1007/978-3-319-73004-2

Skiena, S. (2017). *The Data Science Design Manual*. Retrieved from http://www.springer.com/series/3191

sklearn.ensemble.RandomForestRegressor — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 7, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

sklearn.metrics.mean_absolute_error — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html

sklearn.metrics.mean_squared_error — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

sklearn.model_selection.cross_val_score — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

sklearn.model_selection.KFold — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

sklearn.model_selection.TimeSeriesSplit — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

sklearn.model_selection.train_test_split — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

sklearn.neighbors.KNeighborsRegressor — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

sklearn.preprocessing.MinMaxScaler — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

sklearn.preprocessing.StandardScaler — scikit-learn 0.23.2 documentation. (n.d.). Retrieved August 6, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Smith, S. I., & Lasch, C. (2016). Machine Learning Integration for Adaptive Building Envelopes. *Posthuman Frontiers*, 98–105. Retrieved from http://papers.cumincad.org/data/works/att/acadia16_98.pdf

Spann, T., Deschamps-Sonsino, A., Lawrence, C., Churilo, C., & Azzola, F. (2019). The Internet of Things: Connecting Devices and Data. *DZone Devada Media*, 36. Retrieved from https://dzone.com/guides/internet-of-things-connecting-devices-and-data

Spann, T., Lawrence, C., Azzola, F., Simmons, D., Styger, E., & Smith, T. (2018). Internet of Things: Harnessing Device Data. *DZone Devada Media*, *V*, 44. Retrieved from https://dzone.com/guides/iot-harnessing-device-data

Tukey, J. (1977). *Exploratory Data Analysis*. Addison-Wesley.

Vishwanath, A. (2018). IoT and Machine Learning to Reduce Energy Use in Cooling Systems. *IBM Research-Australia, Internet of Things*. Retrieved from https://www.ibm.com/blogs/research/2018/07/reduce-energy-cooling/

Zhang, A., Lipton, Z., Li, M., & Smola, A. (2020). *Dive into Deep Learning* (Release 0.7.1). Retrieved

from https://d2l.ai/index.html