Smart System

Projet Smart Home:

SEM-3A

Presenté par:

Fournier Camille

Kaddour Jana

07/02/2023

Tables des matières

Tables des figures

Tables des tables

Abstract

The goal of this project is to make the most of solar energy by scheduling the operation of electrical appliances during times of maximum solar production. This is achieved through the use of a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The LSTM model can be used to model the temporal dependencies between the solar energy production and the scheduling of the electrical appliances, especially the one who consumes energy the most. In this paper, it has been proposed that a hybrid deep learning model can enhance previous models and use a combined CNN-LSTM network to achieve more accurate short-term load forecasting. The performance of the proposed model is evaluated on real-world data sets and compared with existing models. The results indicate that the proposed method outperforms existing models, making it a valuable tool for short-time load prediction applications. The efficiency of the approach can be measured through detailed energy loss calculations before and after implementation, over a period of one year.

1. Introduction

The Expe-smart house project was initiated in 2018 to provide real-time data from a 5-person household occupying a 120 m^2 space. Through a Grafana portal with an Influxdb database, 340 measuring points are accessible to scientists. This smart home is constructed with Open Source Hardware and Software, making it adaptable to new technologies and sensors. The data collected includes consumption of electricity, gas, and water for each device, temperature, humidity, and brightness of each room, door, and window positions, motion sensor readings, light state, air analysis for each room, and outdoor weather conditions (see Figure 1).

Figure 1: Expe-smarthouse project

The objective of this project is to make the most of solar energy by scheduling the operation of electrical appliances during times of maximum solar production. By doing so, the aim is to minimize the energy losses that occur during the charging and discharge of batteries. To measure the success of this strategy, detailed calculations will be carried out before and after implementation over a certain period of one year. These calculations will allow a comparison of the energy losses that occurred before and after this scheduling strategy. The results of these calculations will provide valuable insights into the efficiency of the approach and will inform any necessary improvements that need to be made in the future.

To implement the scheduling, predicting the PV production is necessary to know when solar energy can be used directly. To implement this objective, the first needed step is to extract the data that can highly affect the prediction's results. The data is related to a sensor that measures it. The following sensor data were chosen to be investigated:

- Outdoor Temperature
- Irradiation (pyranometer data)
- Solar Panels Production

The data is extracted through a specified period of the year, starting from 27/05/2022 until 24/09/2022, which is 4 months of training data. After this step, to create the prediction, two algorithms that can be used for time series prediction will be implemented: LTSM and CNN-LSTM.

2. Machine learning algorithm presentation

State of Art:

The paper "A Hybrid Deep Learning Model for Short-Term Load Forecasting" by Behnam , Manar et al. (2021) presents a state-of-the-art approach for short-term load forecasting by combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. The proposed hybrid deep learning model, called PLCNet, is evaluated on two real-world data sets: "hourly load consumption of Malaysia" and "daily power electric consumption of Germany". This article highlights the potential of combining CNN and LSTM networks for accurate predictions in the context of smart grids, where accurate load forecasting is crucial for minimizing the gap between electricity supply and demand.
The article also highlights the importance of considering time-series attributes, such as trend, seasonality, and noise, when making load forecasting predictions. In summary, the study provides evidence that the combination of CNN and LSTM networks is a promising approach for short-term load forecasting and has the potential to make a significant contribution to the field of energy forecasting. (see bibliography 1)

The paper "An Enhanced Hybrid Model Based on Convolutional Neural Network and Long Short-Term Memory for Short-Term Load Forecasting" by Musaed et al. (2020) presents the state of the art of combining Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) for short-term load forecasting. In recent years, there has been a growing interest in this combination due to its superior performance in terms of accuracy and efficiency compared to individual models.
The authors propose a hybrid model that enhances previous models and uses a combination of CNN and LSTM to achieve more accurate short-term load forecasting. The performance of the proposed model is evaluated on real-world data sets and compared with existing models. The results indicate that the proposed model outperforms existing models, making it a valuable tool for short-term load forecasting applications (see bibliography 2).

The objective of this project can be achieved through the use of a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTMs) models. The CNN model can be utilized to analyze and process the data of solar production to predict the future production. An LTSM can be used then to model the temporal dependencies between the solar production and the scheduling of the energy appliances, especially the one who consumes energy the most. LTSM can be trained on the predicted solar energy production and the data of electrical appliances usage to schedule the appliances in a manner that minimizes energy losses. This combination of models allows for a more accurate and efficient solution, taking advantage of the strengths of both CNNs and LSTMs. The success of the approach can be measured through detailed energy loss calculations before and after implementation, over a period of one year.

3. Data recovery

In the Smart Home, as mentioned before, there are sensors installed to measure a wide range of variables. We have selected the following: Outdoor Temperature, Irradiation (pyranometer data) and Solar Panels Production. The data is measured with a frequency of seconds which means that the size of the data is important making the analysis harder. For that reason, data resampling will be applied (explained more thoroughly in part 4) to assure different purposes such as:

- Smooth out the noise or fluctuations in the data and make patterns or trends more visible.
- Built models with aggregated data rather than raw data, better for machine learning approaches.
- Less amount of data, easier to be analyzed and visualized.

After that, it was identified that there were periods where no values had been measured. For example, there was a period from 25/09/2022 until 03/10/2022 where no measures were registered. One solution could be to try replacing the value of the missing days 25/09 with ones of some weeks before.

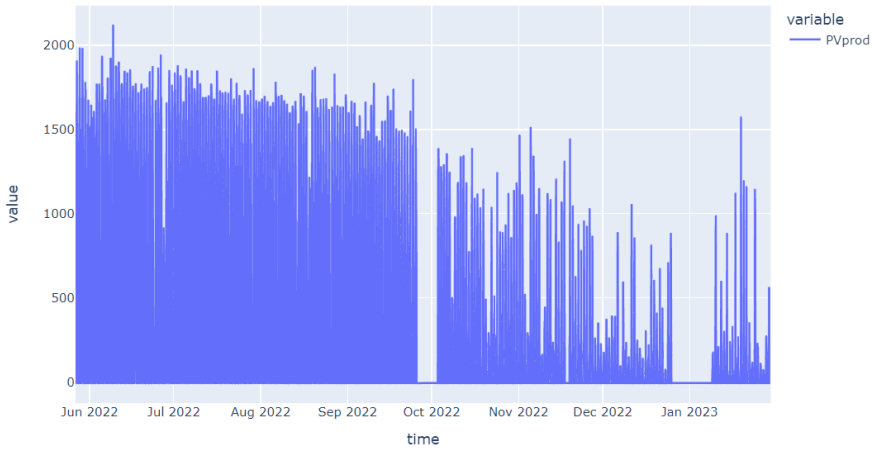


Figure 2: PV production (W)

However, this method might not be very accurate since the values can be changed with a large proportion in respect to the previous week's values. Instead, it was decided to separate the data into 2 divisions : Winter and Summer and only exploit the "summer' values for our prediction. In fact, since in Winter the solar panels will produce the minimal amount to feed the home, it was better to select the Summer part only and study the models on it because in that time of the season the sun will give the highest and maximum production to feed the home. So, removing the data from the winter months was applied since there were missing values problems and these months can degrade the performance of the model. Here, it was the idea to build a model with high performance with a high production where all data are presented. The selected period was from 27/05/2022 until 24/09/2022.

4. Data processing

4.1. First observations and pre-processing

Once the data is recovered in different csv files, it is now possible to make observations about its quality. To do so and for ease of exploitation later on in the project, the panda library will be used.

Concerning the quality of the measurements, what can be observed from a quick overview of the csv file is that there are a lot of duplicates, with often two data given at the same time (see Figure 3). To be able to exploit the data later on in the predictive algorithms, these duplicates need to be taken out. This is easily done using the panda function drop_duplicate (see Appendix 1).

Filter 710,579 records	−100 1.3k
2021-01-30T13:45:46.684979Z	31,74
2021-01-30T13:45:46Z	32,26
2021-01-30T13:46:46.770057Z	30,69
2021-01-30T13:46:46Z	34,73
2021-01-30T13:47:46.645663Z	30,54
2021-01-30T13:47:46Z	36,9

Figure 3: Duplicates in the measurement, here for pyranometer data

After, the time range selected for this application is extracted and a resampling process is applied to get the frequency of measurement from 1s to 1 hour. After the resampling, it can be observed that missing values have appeared in the dataframe, they are replaced with the previous existing values in the dataframe using interpolation with the "ffil" method (see Appendix 2).

4.2. Visualization

Data are displayed in graphs to identify possible outliers in the measurement (abnormal measures made by the sensor). Below, the time series of the PV production, irradiance and external temperature can be observed:

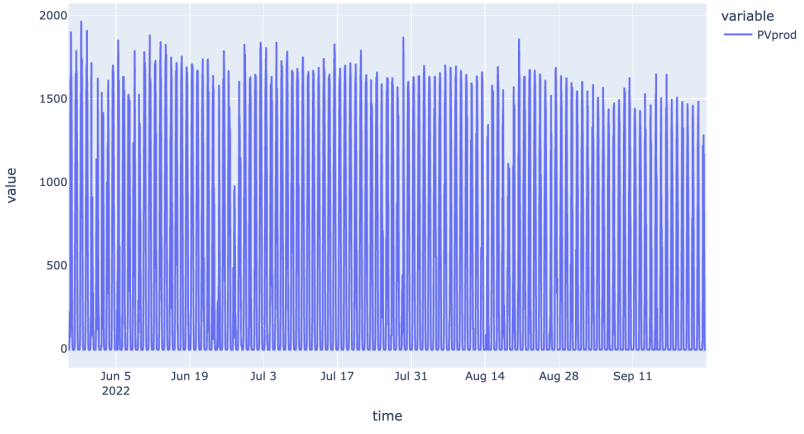


Figure 4: PV production (W) data to be used in the predictive algorithms

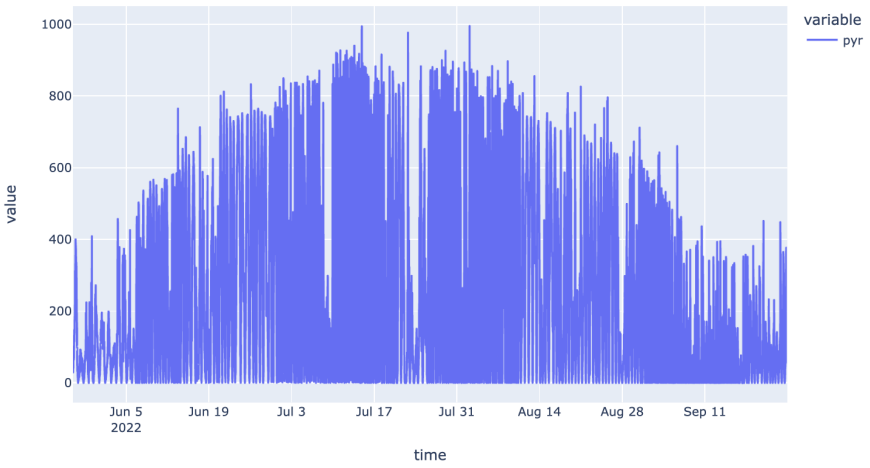


Figure 5: Pyranometer (W/m^2) data to be used in the predictive algorithms

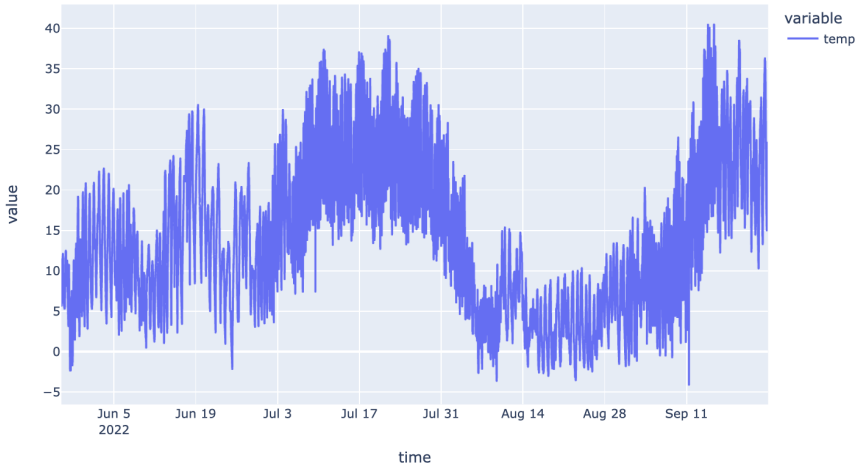


Figure 6: Temperature (°C) data to be used in the predictive algorithms

From the data visualization, no outliers can be identified. Thus, no further adjustments are made to the data. It can now be exploited in predictive algorithms.

5. Algorithms implementation

5.1. LSTM implementation

To implement the LSTM model, the data is first normalized (see Appendix 3) to make sure it will be in an appropriate range for the model.

Then, we need to window the data to create the features and labels for the model. To do so, we use the function split_sequence presented in Figure 7 below. The function takes as inputs the normalized data (=sequence) and how many data points we will be using for the next prediction(=nsteps). It returns the array X, the features splitted into sequences of the specified number of steps, and the array y, the label we want to predict.

```
from numpy import array
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence

n_steps = 24
# split into samples
X, y = split_sequence(scaled_data, n_steps)
# summarize the data
#for i in range(len(X)):
  #print(X[i], y[i]) #--> if you want to visualize remove #
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

Figure 7: Creation of the features and labels for the model

The training and testing datasets are created using approximately 80% (num_div in the code) of the data for the training and 20% for the testing (see Figure8).

```
num_div = 14000

X_train=X[0:num_div]
y_train=y[0:num_div]
X_test=X[num_div:]
y_test=y[num_div:]
y_train=y_train.reshape(y_train.shape[0])
y_test=y_test.reshape(y_test.shape[0])
```

Figure 8: Definition of the training and testing sets

Finally, we define the LSTM model architecture:

- By setting the number of hidden units in each LSTM layer, here 1.
- By setting the input shape that contains the numbers of time steps and the number of features used for the input data.
- By choosing the dropout rate (0.2) for our model. It is the fractions of neurons that are randomly dropped during the training phase to prevent the model from overfitting.
- By choosing the optimization algorithm that will be used to train the model, here 'adam'.
- By selecting the loss function(here mean squared error) that will be used to evaluate the LSTM model's performance during training .
- By setting the evaluation metrics to be used for evaluating the model's performance for training and testing, here 'accuracy'.

```
model = Sequential()
model.add(LSTM(1,input_shape=(n_steps, n_features),dropout=0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse',metrics=['accuracy'])
```

Figure 9: Hyperparameters settings

The optimizer, loss and metrics chosen and dropout rate are standard for a LSTM algorithm. The number of hidden layers used can start from 1. We tested 1 and 2 hidden layers to see which was giving the best results. After computing the R2 score, it was found that 1 hidden layer had a R2 score of 0.926 while two layers gave 0.911. Thus, only one hidden layer was selected.

It can be noted that for this project only experimentation was used to fine tune the hidden layers of the LSTM model. However, it is possible to find an optimization of all the hyperparameters to be used by implementing a grid search algorithm as described in (Fuentes, 2018). This method was not implemented for our project because of the long computation time of the model.

Finally, we finished implementing the model by training it and making a prediction to compare with our y_test dataset (see Appendix 6) .

5.2. CNN- LSTM implementation

The CNN-LSTM being a hybrid version of LSTM, its implementation is made in the same manner of the LSTM:

- Sequencing the data into features and labels of the desired number of steps
- Dividing the data into train and test subset
- Defining the model architecture (here again we choose to use standard value and not to pursue a finer tuning of the hyperparameters).

6. Results

Using the LSTM model and CNN-LSTM model, predictions were made for a time horizon of 1, 12 and 24 hours. As a reminder, our objective for this project would be to have the best prediction possible for the next 24 hours of the PV production to advise the homeowner on when to schedule the running of high consumption appliances such as the dishwasher, washing machine or dryer.

As a first step it was decided to run the LSTM and CNN-LSTM algorithms for a prediction of 1 hour to investigate which was the most performing algorithm. According to the result Table 1 below. CNN-LSTM proved to have better performances than LSTM. It was thus decided to investigate if adding other features to it could improve its prediction performances. The impact of adding data of irradiance, temperature and both were tested however none improved the quality of the model.

Average R2 score	**Time predicted**		
Model	**1H**	**12H**	**24H**
LSTM	0.765	-	-
CNN LSTM with only PV prod	0.908	0.81879	0.826882
CNN LSTM PVprod+irradiance future	0.899	0.72284	0.80858
CNN LSTM PVprod+temp future	0.90202	0.75743	0.697771
CNN LSTM temp +irrad future	0.8862624	0.74514	0.75278

Table 1: Average R2 score measured for the different predictions

7. Discussion

Our prediction performance for the 24h prediction has a best R2 score of 0.82 by using the CNN LSTM algorithm with only the PV production data. The results are good but maybe other pathways could be explored to find improvements. For example, looking at the best way to make the prediction here we use the 24h last hour of data to make our prediction, maybe using more or less data would be more effective. We could also look into fine tuning our hyperparameters. It was not done in this work due to time constraints but it could be another improvement pathway.

Should we reach a higher accuracy, we can hope to make good energy savings by directly using the PV produced energy because using the battery means that each time we use the inverter only 95% (ref Victron 3kW) of the energy is converted. Thus, if we look only at the consumption of the dishwasher (one of the heavy load appliances that can be programmed at a certain time to run) which is approximately 1 kW. The dishwasher runs for 2 hours approximately every two days. For the summer months, the production can peak 1,6kW which is more than enough to cover the load of the dishwasher. Thus, scheduling the running of the dishwasher could save:

- For a day (2 hour of running): Energy lost to have 1kWh from the battery :(1000/0,95*0,95 -1000)*2= 216 Wh
- For 15 days: Energy lost: 3,2 kWh
- For 6 summer months (April-Septembre): Energy lost: 19,4 kWh

8. Conclusion

During this project, we looked at ways to implement a prediction of the PV production for the next day. For that, we first had to retrieve and process the data we were interested in. After,, we researched LSTM and CNN LSTM algorithms to make our prediction. After testing both algorithms, it appeared that CNN LSTM was most promising. We looked at adding other features to enhance the CNN LSTM model's prediction but none of the features we tried were able to increase the model. The best prediction was thus with CNN LSTM algorithm using only the historical PV production data with a R2 score of 0,82.

9. Appendix

Appendix 1: drop_duplicate function to remove duplicates in the csv files

```
#removing duplicates
df['time'] =pd.to_datetime(df['time'])
df['time'] = df['time'].dt.floor('s')
df.drop_duplicates(inplace=True)

pyr['time'] =pd.to_datetime(df['time'])
pyr['time'] = pyr['time'].dt.floor('s')
pyr.drop_duplicates(inplace=True)

temp['time'] =pd.to_datetime(df['time'])
temp['time'] = df['time'].dt.floor('s')
temp.drop_duplicates(inplace=True)

hum['time'] =pd.to_datetime(df['time'])
hum['time'] = df['time'].dt.floor('s')
hum.drop_duplicates(inplace=True)

helio['time'] =pd.to_datetime(df['time'])
helio['time'] = df['time'].dt.floor('s')
helio.drop_duplicates(inplace=True)
```

Appendix 2: example of resampling of the pyranometer data to a frequency of 10mn and use of interpolation

```
pyr=pyr.resample('10T',on='time').mean()
pyr.interpolate(method="ffill", inplace=True)
df['pyr']=pyr['value']
df
```

Appendix 3: Implementation of the LSTM model normalization

```
#Normalizing the data to be used in the LSTM model
data_to_use=df.PVprod.values
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_data=scaler.fit_transform(data_to_use.reshape(-1,1))
```

Appendix 4: Windowing of the data to create the features and label of the data set

```
# creating the input and features for the model by sequencing the data
from numpy import array
def split_sequence(sequence, n_steps):
   X, y = list(), list()
   for i in range(len(sequence)):
```

```
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
# define input sequence
n_steps = 24
# split into samples
X, y = split_sequence(scaled_data, n_steps)
# summarize the data
#for i in range(len(X)):
 #print(X[i], y[i]) #--> if you want to visualize remove #
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
#for i in range(len(X))
```

Appendix 5: Training and test set

```
num_div = 14000

X_train=X[0:num_div]
y_train=y[0:num_div]
X_test=X[num_div:]
y_test=y[num_div:]
y_train=y_train.reshape(y_train.shape[0])
y_test=y_test.reshape(y_test.shape[0])
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

Appendix 6: Training and prediction of the LSTM model

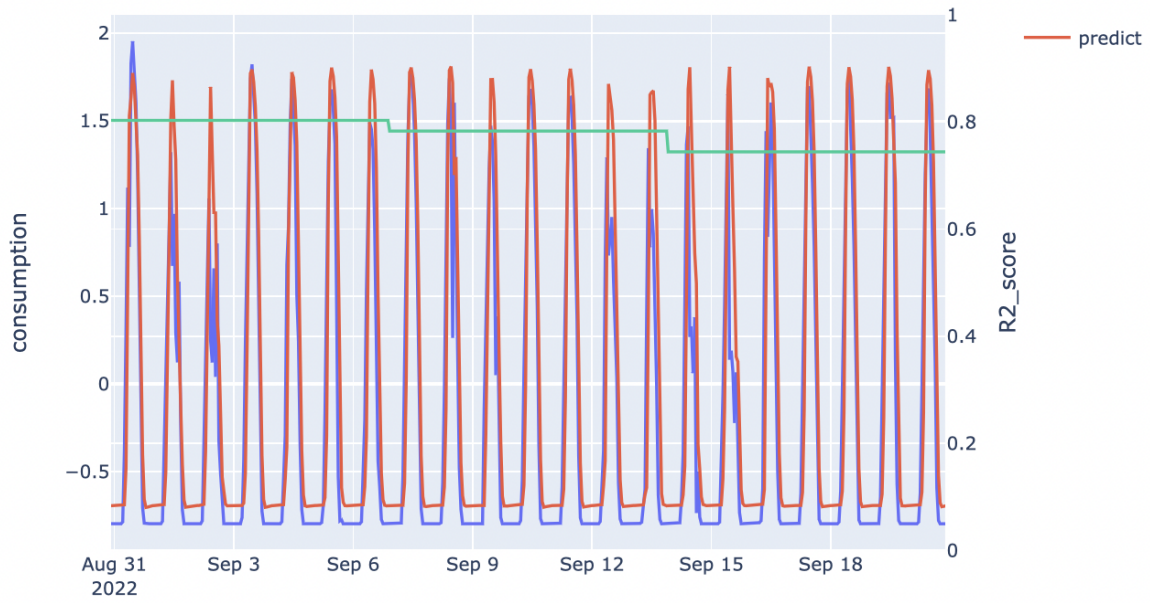
```
%%time

network = model.fit(X_train,y_train, epochs=20, validation_data=(X_test,
y_test), shuffle=False,verbose=1)
```

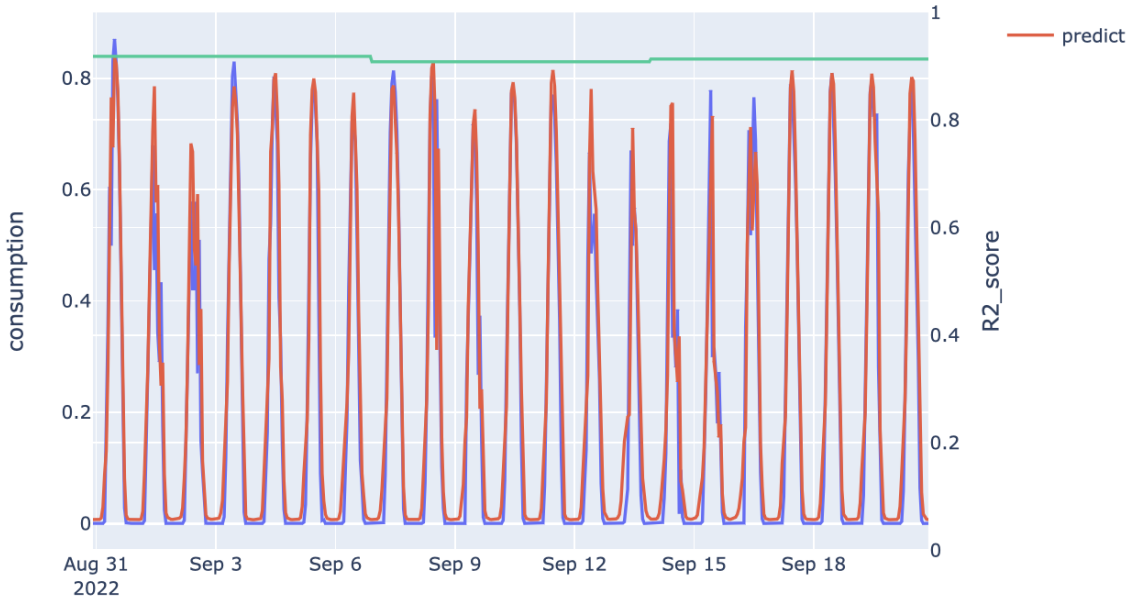
```
# plot train and validation loss
plt.plot(network.history['loss'])
plt.plot(network.history['val_loss'])
plt.title('model train vs validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()

yhat = model.predict(X_test, verbose=0) #verbose is to see the epochs or
not (0,1,2)
```

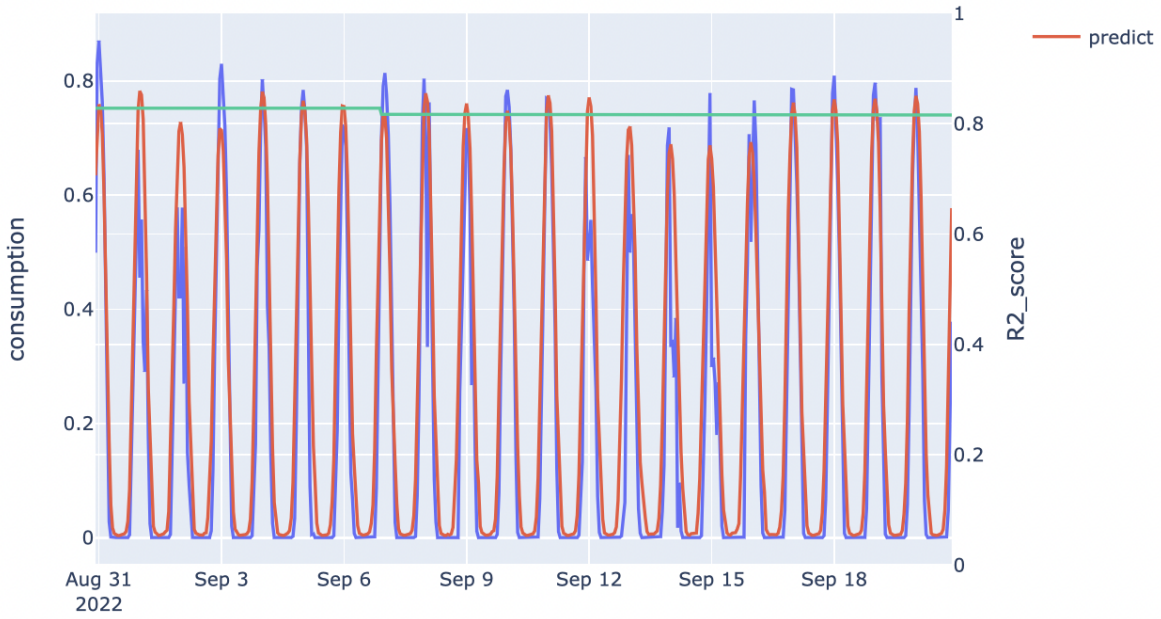
Appendix 7: Variation of the R2 score between y_test and the prediction of 1H for the LSTM algorithm



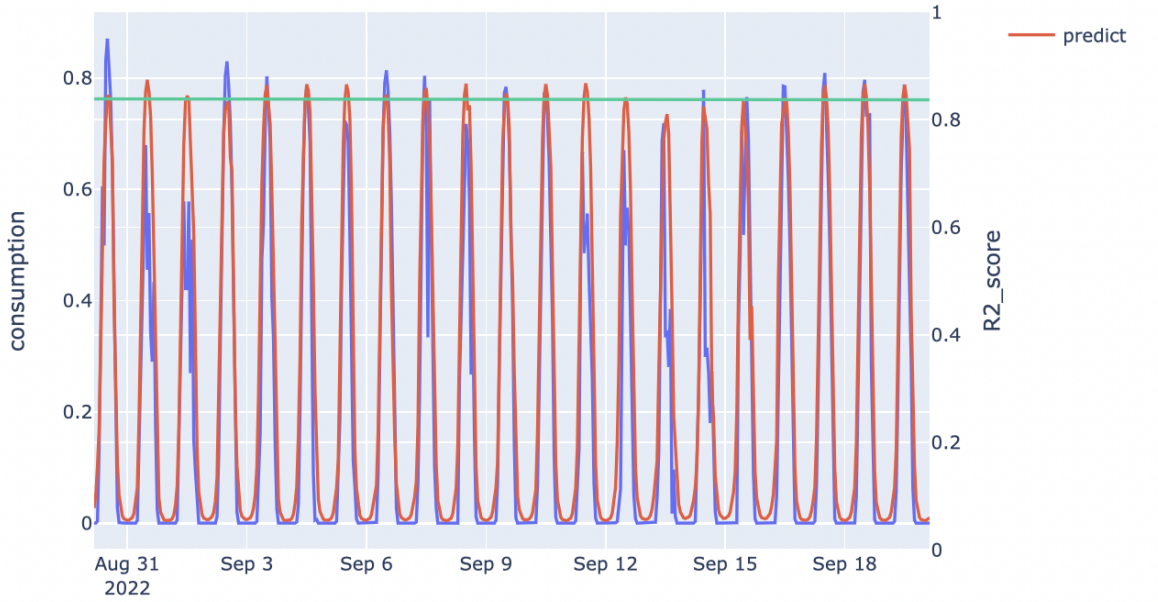
Appendix 8: Variation of the R2 score between y_test and the prediction of 1H for the CNN-LSTM algorithm



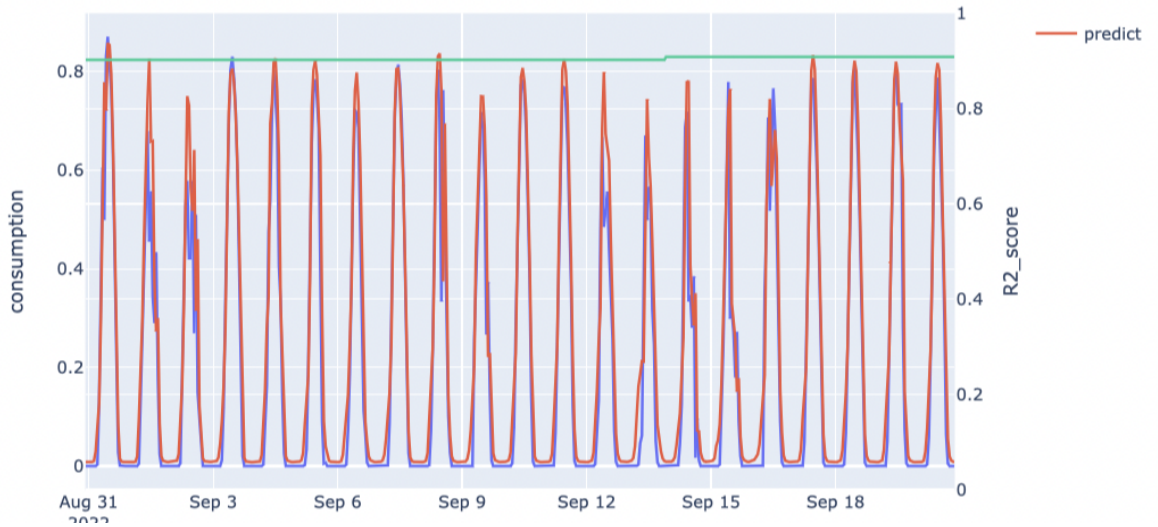
Appendix 9: Variation of the R2 score between y_test and the prediction of 12H for the CNN-LSTM algorithm



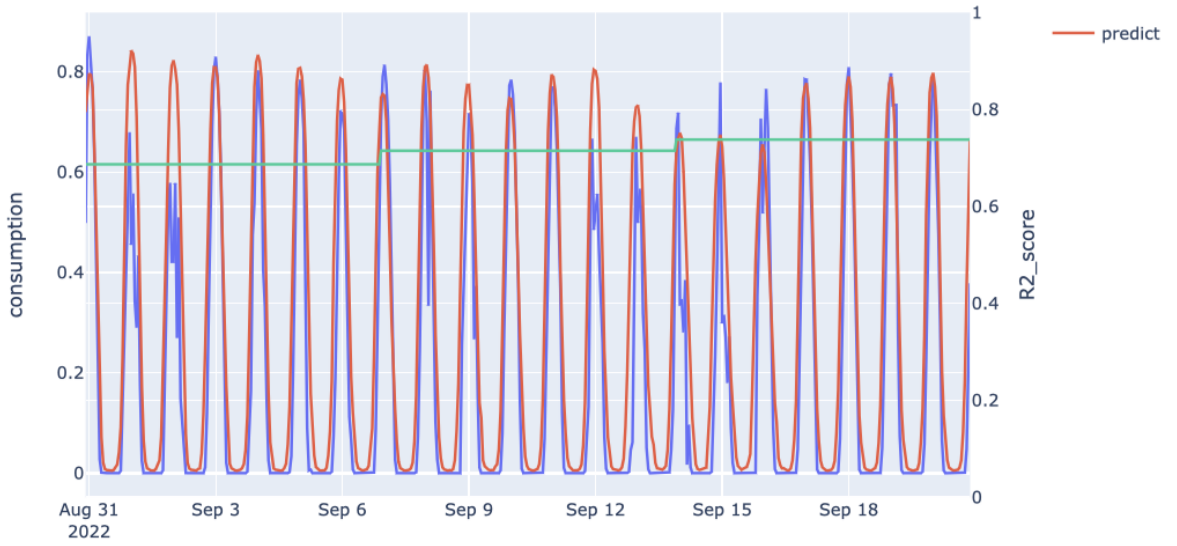
Appendix 10: Variation of the R2 score between y_test and the prediction of 24H for the CNN-LSTM algorithm



Appendix 11: Variation of the R2 score between y_test and the prediction of 1H for the CNN-LSTM algorithm adding the irradiance feature

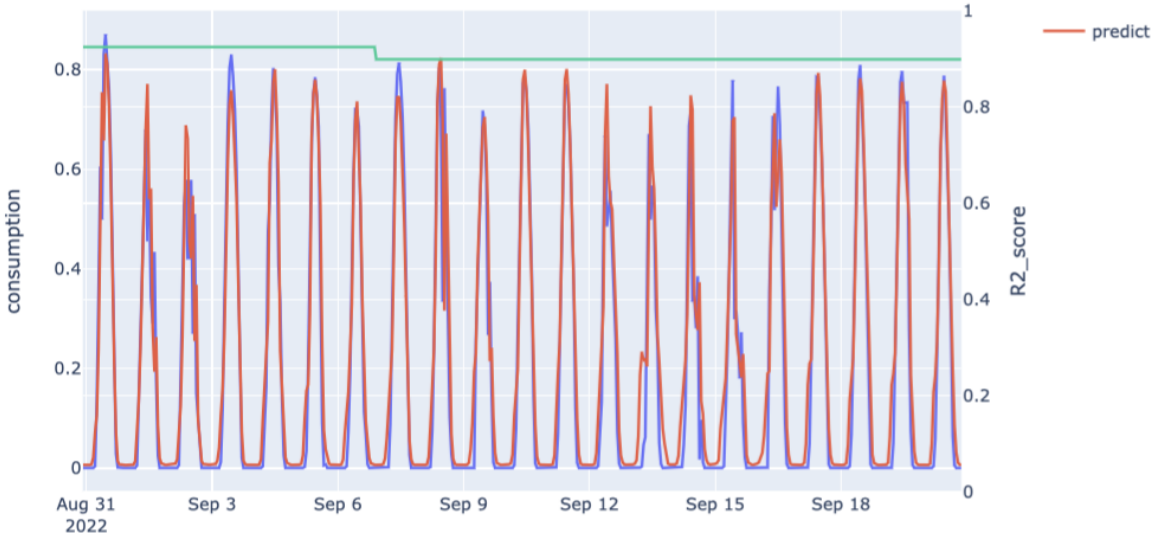


Appendix 12: Variation of the R2 score between y_test and the prediction of 12H for the CNN-LSTM algorithm adding the irradiance feature

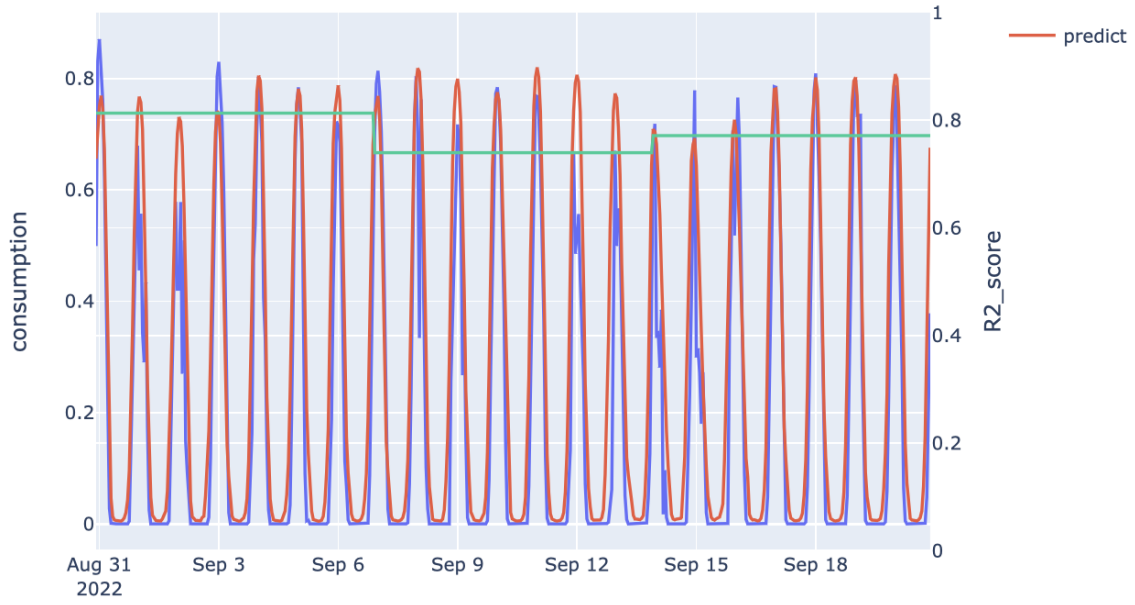


Appendix 13: Variation of the R2 score between y_test and the prediction of 24H for the CNN-LSTM algorithm adding the irradiance feature

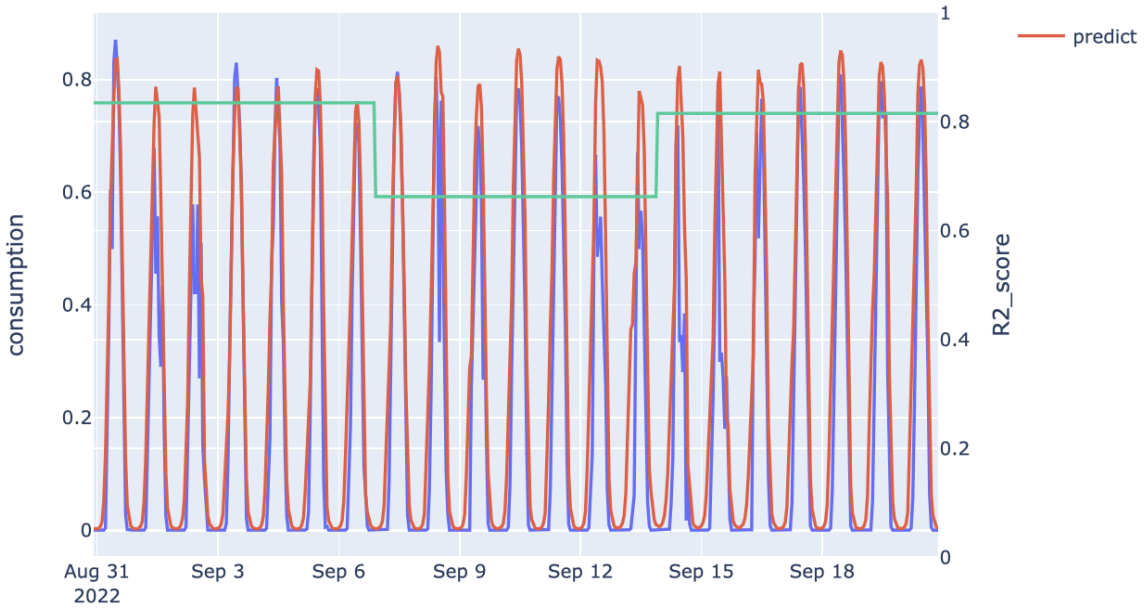
Appendix 14: Variation of the R2 score between y_test and the prediction of 1H for the CNN-LSTM algorithm adding the temperature feature



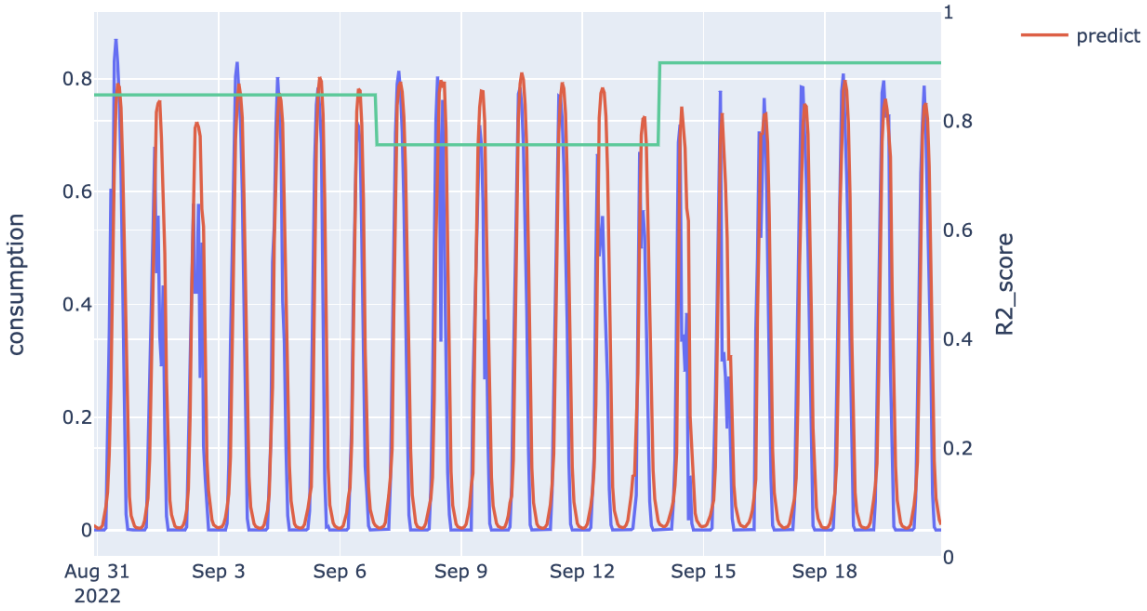
Appendix 15: Variation of the R2 score between y_test and the prediction of 12H for the CNN-LSTM algorithm adding the temperature feature



Appendix 16: Variation of the R2 score between y_test and the prediction of 24H for the CNN-LSTM algorithm adding the temperature feature



-Appendix 17: Variation of the R2 score between y_test and the prediction of 24H for the CNN-LSTM algorithm adding the temperature and irradiance feature



Bibliography

1- Behnam,Manar et al. (2021). A Hybrid Deep Learning Model for Short-Term Load Forecasting.[IEEE Transactions on Smart Grid, 10(1), 1-10]. https://ieeexplore.ieee.org/abstract/document/9356582

2- Musaed et al. (2020).An enhanced hybrid model based on convolutional neural network and long short-term memory for short-term load forecasting.[IEEE Xplore]. https://ieeexplore.ieee.org/document/9210478

3-Fuentes, A. (2018). *Hands-On Predictive Analytics with Python: Master the complete predictive analytics process, from problem definition to model deployment*. Packt Publishing Ltd.