Predictive Modeling of Energy Use in Appliances

Final Project Report

Professor: E. Houstis

Armpounioti Maria-Eleni 03183
Kostarellou Panagiota 03357
Patakioutis Sofoklis 03310

Machine Learning for Data Science and Analytics 2024-2025

Department of Electrical & Computer Engineering
University of Thessaly, Volos

{marmpounioti, pkostarellou, spatakioutis}@e-ce.uth.gr

Table of Contents

Abstract In this project the goal was to explore various machine learning techniques to predict the energy use of appliances based on sensor data. The project was inspired by the paper ***"Data driven prediction models of energy use of appliances in a low-energy house"***. In our research we explored some of the same models as noted in the paper and added a few more to make comparative analysis. This report describes the whole process as well as our observations and results.

1 Introduction

1.1 Project Topic - Goals

The objective of this project was to develop predictive models for appliance energy consumption using sensor data and to evaluate their performance. Inspired by the paper ***"Data Driven Prediction Models of Energy Use of Appliances in a Low-Energy House"*** we experimented with different machine learning and deep learning models such as GBM, LSTM and transformers. Feature engineering is an equally crucial part in our project for improving the performance of the models.

1.2 Technical Details

The whole project was develop in **Python**, using the **Google Colab** and **Kaggle** runtime environments, thus, all our files are in the **Jupiter Notebook** (.ipynb) format. Since we performed our research on deep learning models (LSTM and Transformer), we utilized Colab's and Kaggle's **T4 GPU** accelerators to reduce the training and inference times.

We also utilized a number of Python libraries to perform our research. Here is a list:

- **pandas** for dealing with the dataset files and Exploratory Data Analysis.
- **tensorflow/keras** for multiple steps of the dataset preprocessing as well as defining, training and testing the models.
- **sklearn** for multiple steps of the dataset preprocessing and model trainings.
- **matplotlib** for dealing with plots.
- **seaborn** for dealing with specific plots.
- **os, joblib, json** for saving / loading model valuable information such as scalers, encoders, etc.
- **pytorch** for building and training the transformer model.

Finally, the dataset and some parameters are stored in Google Drive. You can access them as well as the source code **here**.

2 Brief Overview of the base paper

The goal of the paper ***"Data Driven Prediction Models of Energy Use of Appliances in a Low-Energy House"***, is to explore data-driven models for forecasting energy consumption. The models studied include multiple linear regression, support vector machine with radial kernel, random forest, and gradient boosting machines (GBM).

The data used in the research were collected from a low-energy house and an airport weather station nearest to the house. A feature analysis is performed focusing on the energy consumption of each appliance and the influence of temporal factors, such as time of day and day of the week, on energy use. The Boruta package is used to select the relevant variables,identifying 35 optimal predictors. Each model is trained and the metrics used included RMSE, R-squared/R2, MAE and MAPE. Among all models, GBM had the best performance, explaining 97% of the variance in the training data and 57% in the test data. The most significant predictors included time of day, atmospheric pressure, and indoor temperature and humidity. For all models, the time information (NSM) was ranked the most important for predicting the consumption of appliances.

These findings emphasize the value of incorporating environmental and temporal data into predictive models for residential energy management.

3 Exploratory Data Analysis

3.1 Studying the nature of the dataset

The dataset consists of 19735 data samples 27 columns. We use the pandas dataframe built-in functions to examine the dataset for column names, data types and missing values. We observe no null values and that all columns except 'date' have numerical values as shown in **Table 1**:

Table 1. Dataset Description - Check for missing values

#	Column Name	Non-Null Count	Data Type
0	date	19735	object
1	Appliances	19735	int64
2	lights	19735	int64
3	T1	19735	float64
4	RH_1	19735	float64
5	T2	19735	float64
6	RH_2	19735	float64
7	T3	19735	float64
8	RH_3	19735	float64
9	T4	19735	float64
10	RH_4	19735	float64
11	T5	19735	float64
12	RH_5	19735	float64
13	T6	19735	float64
14	RH_6	19735	float64
15	T7	19735	float64
16	RH_7	19735	float64
17	T8	19735	float64
18	RH_8	19735	float64
19	T9	19735	float64
20	RH_9	19735	float64
21	T_out	19735	float64
22	Press_mm_hg	19735	float64
23	RH_out	19735	float64
24	Windspeed	19735	float64
25	Visibility	19735	float64
26	Tdewpoint	19735	float64
27	rv1	19735	float64
28	rv2	19735	float64

3.2 Visualizing the Time Series

In this step we took a close look at the target column **'Appliances'** and the characteristics of it as a time series by plotting the whole series and also decomposing it into **trend, seasonality** and **residuals** as shown in **Figure 1**.

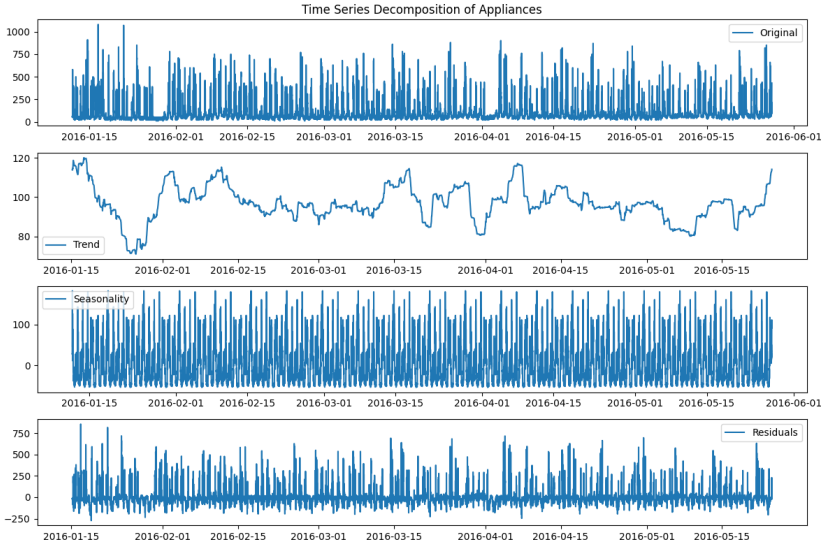

Fig. 1. Time series decomposition plots

We did not observe a specific trend throughout the series with more or less stable energy costs. We did observe 2-3 points with major drop in energy costs and assumed the people living in the house were on vacation. Seasonality has a daily pattern and weekly pattern which stays more or less the same throughout the entire dataset. We used this information to fine tune our models as described in the next sections.

We also plotted the distribution of values of the target column 'Appliances' as shown in **Figure 3**. As we observe, the dataset points do not follow a normal distribution so we will need some sort of normalization / standardization in the next steps.

Lastly, we plotted the autocorrelation of the 'Appliances' column for 1 week (1008 lags) to observe how the variable's past values affect its future ones. In **Figure 2** we can see that the same pattern occurs every day and every week. We used this information to test the models with lags of 144 and 1008 (1 day and 1 week of past data points).

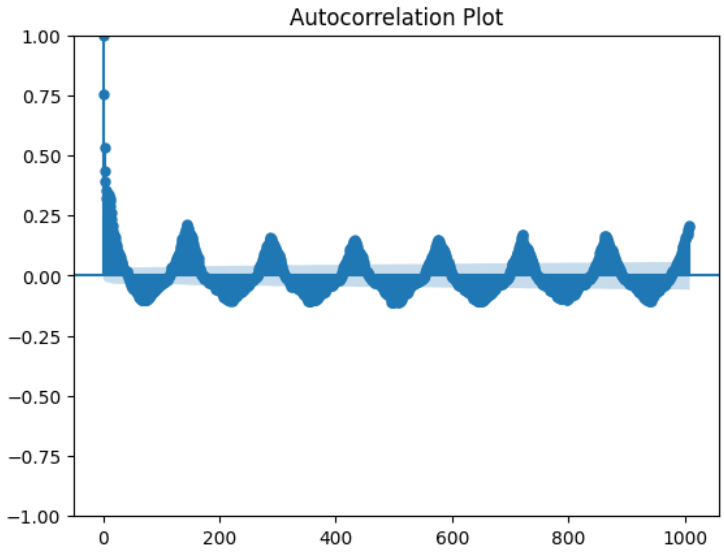

Fig. 2. Autocorrelation of 'Appliances'

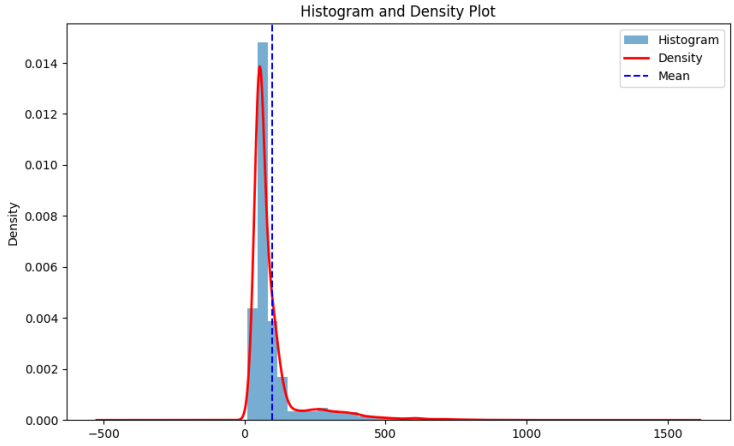

Fig. 3. Distribution of 'Appliances' Watt loads

3.3 Visualizing feature correlations

We then created some visualizations similar to those in the paper to observe feature correlations with each other and with the target variable. We used **seaborn** and **matplotlib** to create pairplots and correlation heatmaps of the features. The groups of features used are the same as described in the paper (we illustrate here only the figures from group 1 and the rest are in **Appendix A**):

Group 1: the energy consumption of appliances with lights, T1, RH1, T2, RH2, T3, RH3.

Group 2: the energy consumption of appliances with T4, RH4, T5, RH5, T6, RH6.

Group 3: the energy consumption of appliances with T7, RH7, T8, RH8, T9, RH9.

Group 4: the energy consumption of appliances with T out, Pressure, RH out, Windspeed, Visibility, TDewpoint, NSM and T6.

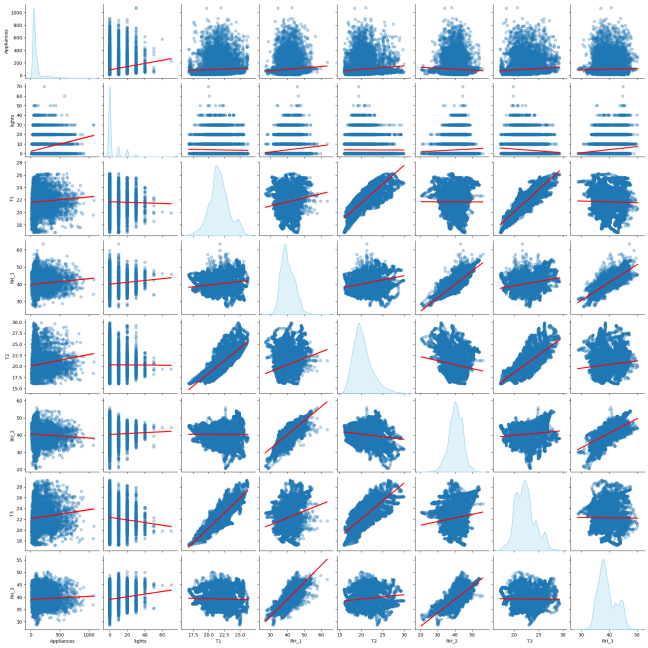

Fig. 4. Pairplot of group 1

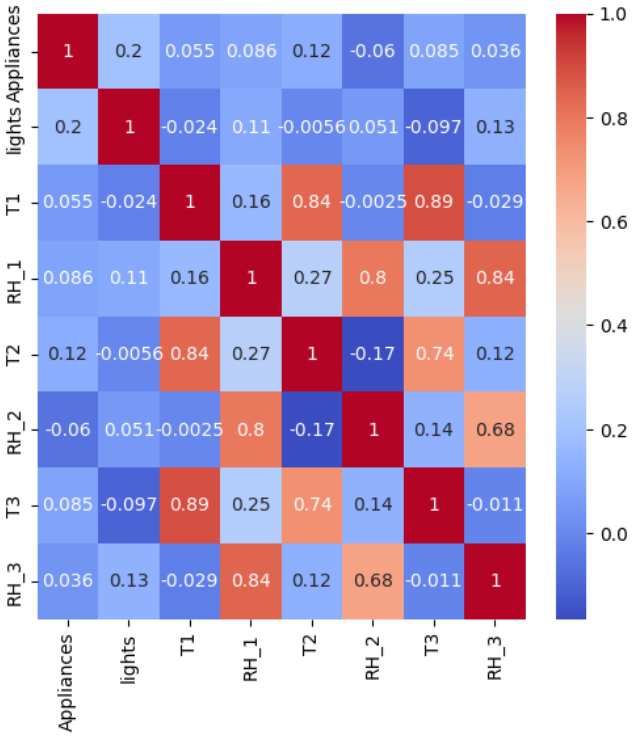

Fig. 5. Correlation heatmap of group 1

4 Dataset Preprocessing

Preprocessing is a procedure in which we clean, transform and integrate a dataset, so its format is suitable for the specific analysis.

After having the dataset loaded, we transform the 'date' column from string to date format. Then, following the paper's flow, we add 'NSM', 'week_status' and 'weekday' columns. 'NSM' column calculates the total number of seconds from midnight untill the time that the measurement was made. 'Week_status' column represents if a measurement is made during workdays or during weekend. 'Weekday' column refers to the day that the measurement was made.

Firstly, we scale the dataset in the range of [0,1] (Min-Max Scaling), in order to ensure that all features contribute equally to the model, so no single feature dominates the distance calculation over the others. LSTM and Transformer models are specifically are very sensitive to scaled data.

'Week_status' and 'weekday' columns are categorical, so we have to encode them. The 'week_status' column in encoded as binary variable, while we have

used one-hot encoding for 'weekday' column. After encoding procedure, we concatenate the encoded columns after having dropped the previous categorical ones.

Note that certain steps of this procedure were not applied on the transformer model because we followed the source code of another project and some pre-processing steps where pre defined there (more information about that on the **Section 9**.

5 Feature Selection

Feature selection is a vital part of every machine learning task. Filtering the features to discard highly correlated ones and ones that are not contributing important information to the prediction can decrease the need for computational resources (CPU, GPU, RAM) and increase the accuracy of the model since it will not train on redundant data.

The method we chose to perform the process is the **Boruta** algorithm (the same method was used in the paper). Boruta is a very popular feature selection method that uses an estimator (a machine learning model) and the dataset to determine the importance of all features. It then keeps the most relevant ones.

In our case, we chose to use an **Extreme Gradient Boost Regressor** (XGBoostRegressor) as Boruta's estimator and run the algorithm on the whole dataset after the preprocessing. The algorithm selected the features shown in **Table 2** as important:

Table 2. Features selected by Boruta

Feature
lights
T3
RH_3
T4
T5
RH_6
T7
T8
T9
T_out
NSM
weekday_Monday
weekday_Saturday
weekday_Tuesday
weekday_Wednesday

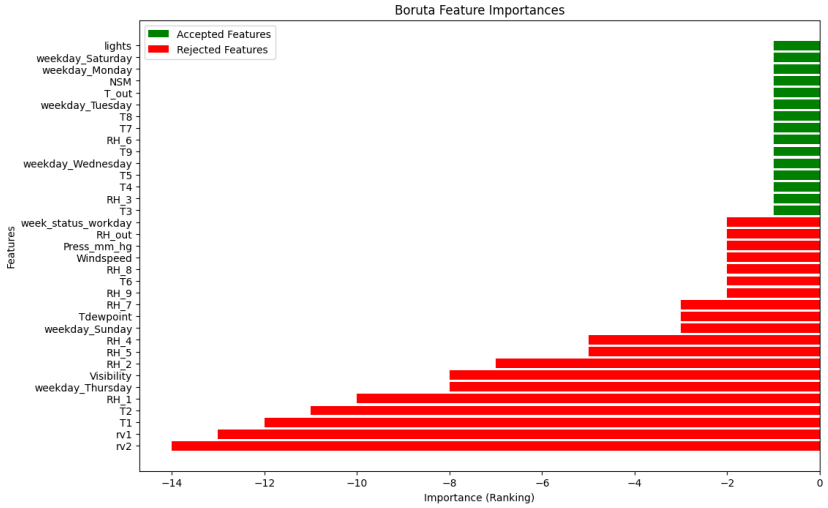

Fig. 6. Boruta Importance rankings (closer to 0 is the best)

However, by observing the selected features as well as the **Figure 6** we saw that some feature selections do not make logical sense. For example, the Boruta suggests that some weekdays are not important while others are but we cannot keep a portion of weekdays as features because of the one-hot encoding. For that reason, we decided to inly eliminate the dummy variables 'rv1' and 'rv2'.

6 Imlementing and testing a GMB model

This is the first model we experimented with and we decided to use an **Gradient Boosting Machine** (GBMRegressor) for our training. We used the model **GradientBoostingRegressor** from the **sklearn.ensemble** library. For this model we did not use a time series approach because we encountered compatibility and technical issues. We used simple regression instead.

We split the dataset into 20% for testing and 80% for training and performed the training of the model with the hyper-parameters described in **Table 6**. Before deciding these hyper-parameter values we experimented manually with 1000, 1200, 1500 and 2000 estimators, learning rates ranging from 0.1 to 0.2 and max depths 3 and 5.

The metrics on the test set, which are identical to the paper's performance on the GBM model, are shown on **Table 4** and the model's predictions are further visualized with **Figures 7 and 8**.

Parameter	Value
Loss Function	Mean Square Error (MSE)
Learning Rate	0.15
Maximum Depth	5
Number of Estimators	1500

Table 3. Parameter values of GBM model

Metric	Value
Mean Absolute Error (MAE)	32.30
Mean Squared Error (MSE)	4234.46
Root Mean Squared Error (RMSE)	65.07
R-squared (R^2)	0.577

Table 4. Performance Metrics of GBM model on test set

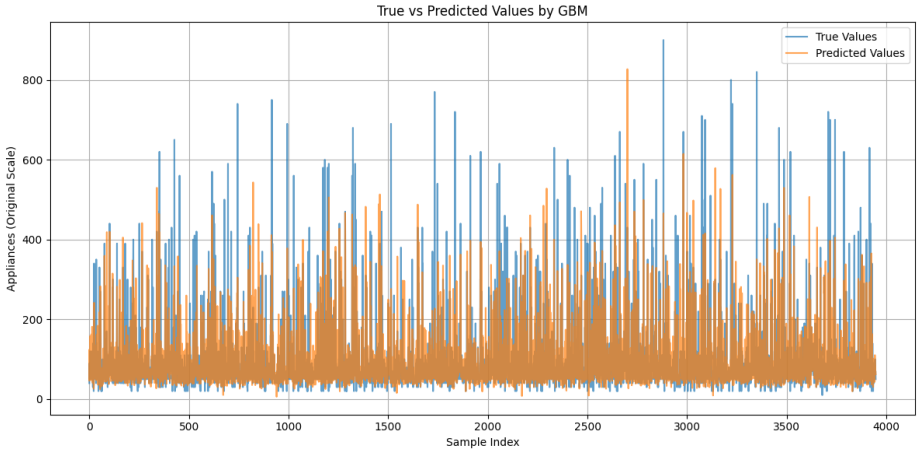

Fig. 7. GBM predictions vs True values

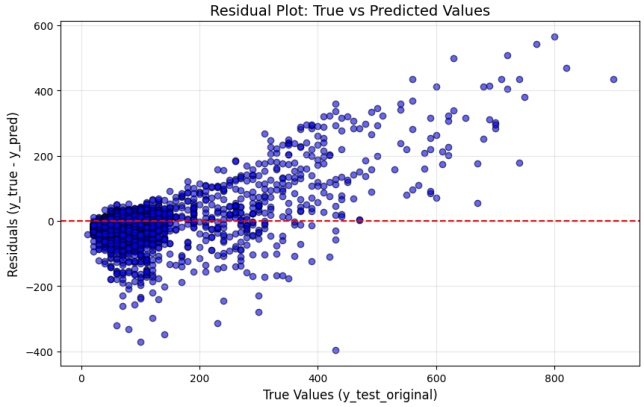

Fig. 8. GBM residuals

7 Implementing and testing an LSTM model

In this section we implemented an LSTM model. Long Short-Term Memory (LSTM) networks are a type of recurrent neural networks (RNN) specifically designed to capture long-term dependencies in sequential data. This quality makes them an ideal model for predicting the energy use of appliances which is required for this project.

7.1 Data Preparation

In this step we used the data already preprocessed from **Section 4** and **Section 5**. We split the data into training (65%), test (20%) and validation (15%) sets in order to evaluate the performance of the model. Time series data has a sequential nature meaning that each observation is dependent on previous ones therefore, before feeding the data into the LSTM it is crucial to transform them into sequences. To achieve that we used TimeseriesGenerator function from TensorFlow/Keras which creates sliding window sequences. We included 66 time steps into each sequence, which is equivalent to an entire week for each sequence.

7.2 Architecture

After having the dataset splitted, we use **TimeseriesGenerator** from **tensorflow.keras.preprocessing.sequence** library to transform the dataset into pairs of sequences and their corresponding target values. We selected 66 lags for every sequence and a batch size of 32. Then, we built a sequential model which consists of:

- LSTM layer of 256 units and 'LeakyReLU' activation function with negative slope equal to 0.3.
- Dropout (0.09)
- Dense layer of 128 units and 'ReLU' activation function.
- LSTM layer of 256 units and 'LeakyReLU' activation function with negative slope equal to 0.15.
- Dropout (0.08)
- LSTM layer of 128 units and 'LeakyReLU' activation function with negative slope equal to 0.07.
- Dropout (0.07)
- LSTM layer of 128 units and 'LeakyReLU' activation function with negative slope equal to 0.07.
- Dropout (0.06)
- Dense layer of 128 units and 'ReLU' activation function.
- LSTM layer of 64 units and 'LeakyReLU' activation function with negative slope equal to 0.04.
- Dropout (0.05)
- LSTM layer of 64 units and 'LeakyReLU' activation function with negative slope equal to 0.04.
- Dropout (0.04)
- Dense layer of 128 units and 'ReLU' activation function.
- Dense layer of 64 units and 'ReLU' activation function.
- Dense layer of 1 unit for the regression's output.

We used **Mean Square Error** (MSE) as the loss function, **Adam** optimizer with learning rate 0.00001 and **Mean Absolute Error** (MAE) for measuring the prediction accuracy. We performed 100 epochs but to prevent overfitting we used **Early stopping** with patience of 10 epochs and we also enabled restore_best_weights parameter, so the model keeps the best training.

7.3 Results

Firstly, we evaluated our model by plotting the training and validation loss (**Figure 9**) and mean absolute error (**Figure 10**) as well.

Afterwards, we test the model and produce the predicted values (**Figure 11**) and the residuals (**Figure 12**).

On the following table (**Table 5**) we sum up the metrics calculated on the test set:

Fig. 9. LSTM train and validation MSE loss

Metric	**Value**
Mean Absolute Error (MAE)	46.753
Mean Squared Error (MSE)	7376.872
Root Mean Squared Error (RMSE)	85.889
R-squared (R^2)	0.015

Table 5. Performance Metrics of LSTM model on test set

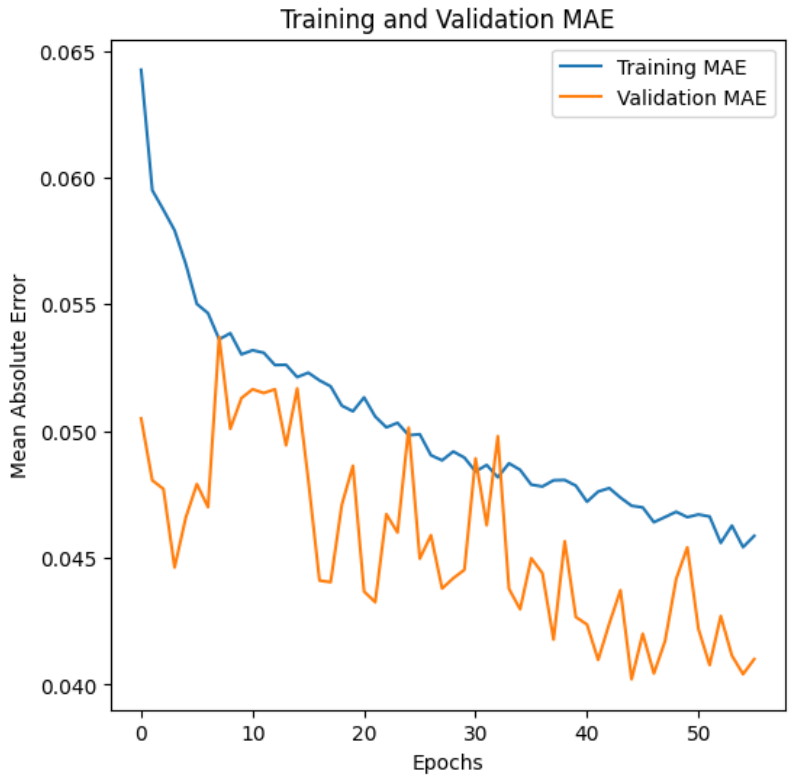

Fig. 10. LSTM train and validation MAE loss

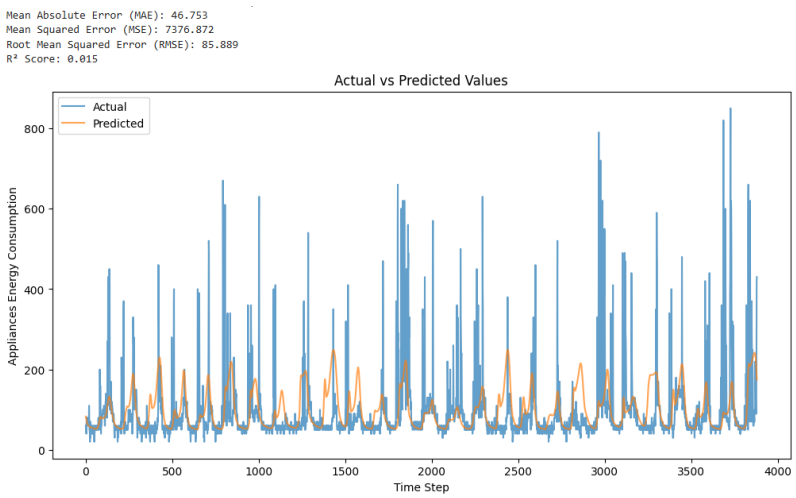

Fig. 11. LSTM predictions vs actual values

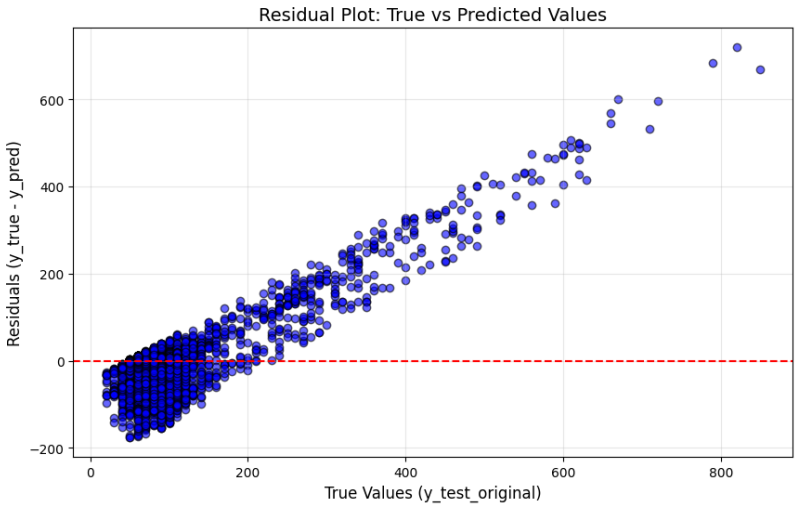

Fig. 12. LSTM prediction residuals

8 Application of Transformer Models for Time Series Prediction

8.1 Introduction to Transformer Models

Transformers are a brand new deep learning architecture that introduces the self-attention mechanism, which allows models to capture relationships between all elements in a sequence simultaneously. The core components of transformers include:

- **Self-Attention:** Computes attention scores for all pairs of tokens in a sequence.
- **Multi-Head Attention:** Combines multiple attention mechanisms in parallel to capture diverse relationships.
- **Positional Encoding:** Adds position-based information to the input embeddings, allowing the model to recognize sequence order.

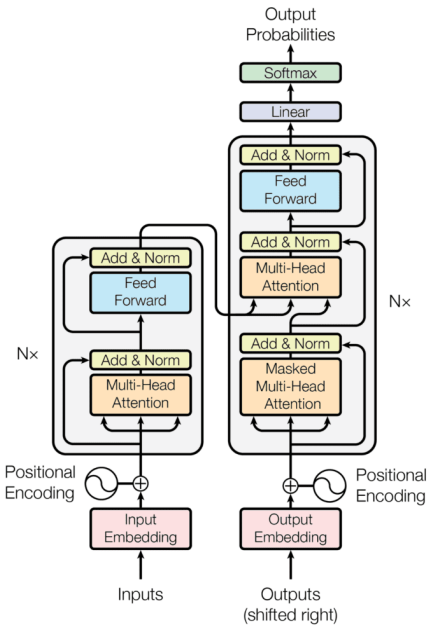

Fig. 13. Transformer Architecture

8.2 Issues in traditional Time Series Prediction models

Time series prediction involves forecasting future values based on historical data. This dependency creates a number of issues that traditional models like RNNs cannot handle well:

- **Sequential Dependencies:** Capturing long-term dependencies in sequential data is a real challenge for statistical models and RNNs.
- **Irregular Intervals:** Handling unevenly spaced time points is another issue that reduces the performance of traditional networks.
- **Feature Extraction:** Automatically extracting meaningful features from raw data is essential for time series and traditional models cannot perform this task efficiently.

8.3 Benefits of Transformers in Time Series

Transformers offer several new features that improve many of the issues faced by traditional methods such as RNNs:

- **Parallelization:** Unlike sequential models, transformers process entire sequences simultaneously which makes the process a lot faster.
- **Long-Range Dependencies:** Self-attention effectively captures relationships across distant time steps.
- **Scalability:** These models are suitable for large datasets and high-dimensional inputs.

8.4 Transformer-Based Architectures for Time Series

Several transformer variants have been developed specifically for time series tasks:

- The **Informer**: Incorporates sparse self-attention for efficiency in long sequences.
- The **TimeTransformer**: Adapts positional encodings to represent time intervals explicitly.

8.5 Applications

Transformers have been successfully applied in various time series tasks:

- **Anomaly Detection:** Identifying irregular patterns in sensor or network traffic data.
- **Forecasting:** Predicting stock prices, weather patterns, or energy demand.
- **Sensor Data Analysis:** Monitoring and predicting outcomes from IoT devices.

8.6 Challenges and Issues

Despite their many benefits compared to traditional models, transformers face several challenges in time series prediction:

- **Computational Cost:** High memory usage due to quadratic complexity of self-attention.
- **Data Requirements:** Transformers perform best with large, labeled datasets.

9 Implementing and testing a Transformer model

The transformer approach is a special case of our study because we did not create the model from scratch. Instead, we used and modified the given source file **'Transformers_for_timeseries.ipynb'** which contains a project by Alice Martin and adapted to pytorch by Charles Ollion. You can access the original project code **here**.

9.1 Overview of the source code

The project begins by importing the dataset and performing basic preprocessing. Then some functions are defined for splitting the dataset into training, validation and test sets, as well as creating the sequences with lags necessary for time series regression. The next step is defining the Multi-Head Attention layer class which is the core of the transformer. After that, positional encoding is defined to include positional information in the embedding of the sequences. Lastly, the transformer layer and transformer model are constructed as classes. Then we create a transformer mode and perform training testing and create plots.

9.2 Modifications on the source code

We made the following modifications to match the process of our other models and create comparable results:

1. The original code was built to use the original dataset but we added the aforementioned extra columns that were created in the original paper (NSM, week_status, weekday) and removed columns 'rv1' and 'rv2' as we did for the rest of the models. We also One-hot encoded 'week_status' and 'weekday' columns.
2. We noticed that the original code was built to make regression for 20 features of the time series, not just the 'Appliances' column (multivariate regression). Since we are performing univariate regression in our experiment, we adjusted the code parameters to make the transformer produce predictions only for the column 'Appliances'.

3. We adjusted some parameters in the time series sequence creation and positional encoding to try and create sequences with 144 lags (This window size corresponds to 1 day since each data point is 10 minutes apart) and 1008 lags (This window size corresponds to 1 week).

9.3 Transformer Architecture - Training

We explored a vast set of parameter values for the transformer architecture and training hyper-parameters and we decided to stick with the values as shown in **Table 6** since the model performed the highest scores with them.

Parameter	**Value**
Epochs	99
Batch Size	32
History Size (Lags)	1008
Optimizer	Adam
Learning Rate	0.000001
Dropout Rate	0.05
Attention Layers	1
Hidden Layer Size	64
Number of Heads	8
Attention Block Size	64

Table 6. Architecture and Hyper-parameters

We split the dataset into **70%** for training and the rest of **30%** was split into **15%** validation and **15%** test set. We performed training with **Mean Square Error** (MSE) as the loss function. We also added an **Early Stopping** feature that monitored the validation loss and triggered when it improve for less than 10^{-4} for 5 consecutive epochs. The 1008 lags is not random, it corresponds to 1 week of data , since the data points are every 10 minutes.

The model reached a loss of **0.43** and a validation loss of **0.40**. The training and validation loss over the epochs of the training can be observed in **Figure 14**. Note that the losses are on the standardized scale and not on the original scale.

For reference and comparison purposes, we also present an example of another training we did with 144 lags (which correspond to 1 day of data) with the parameters shown in **Table 7** and **Figure 15**.

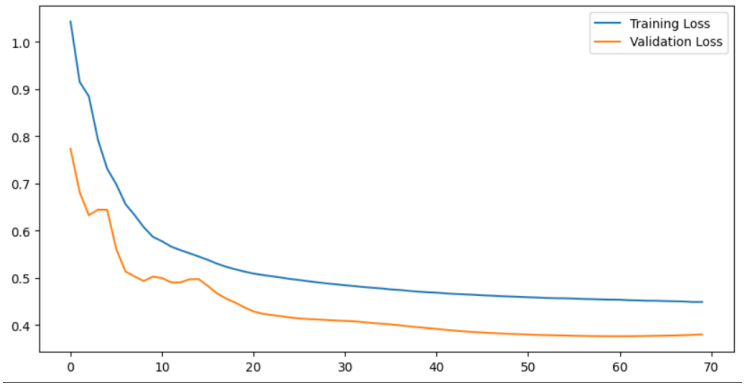

Fig. 14. Training and Validation loss for Transformer with 1008 lags

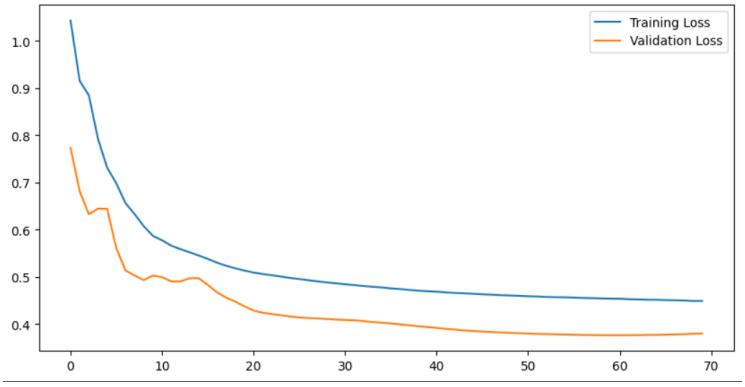

Fig. 15. Training and Validation loss for Transformer with 144 lags

Parameter	Value
Epochs	69
Batch Size	32
History Size (Lags)	144
Optimizer	Adam
Learning Rate	0.000001
Dropout Rate	0.05
Attention Layers	1
Hidden Layer Size	64
Number of Heads	8
Attention Block Size	64

Table 7. Architecture and Hyper-parameters Example 2

9.4 Evaluation on test set

We then evaluated the model on the test set and computed metrics on the original scale of the data to have a better view of the performance. The model reached a MSE of **3737.88** and an RMSE of **61.13**. The **Figure 17** displays a random example of the model's prediction on a specific sample of the test set.

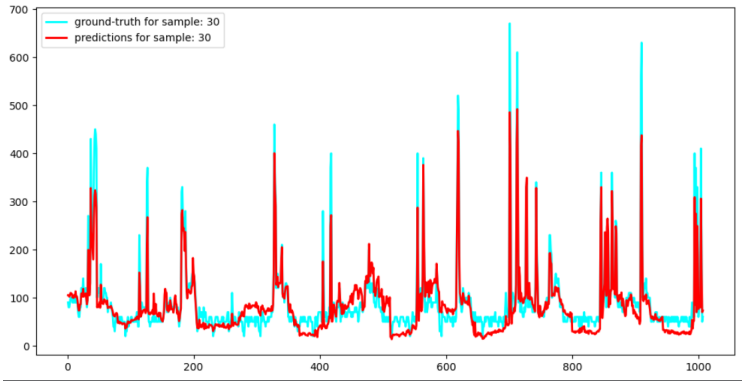

Fig. 16. Transformer prediction vs Ground Truth on random sample with 1008 lags

Similarly, the model with 144 lags reached a MSE of **3922.24** and an RMSE of **62.62**. The **Figure 17** again displays a random example of the model's prediction on a specific sample of the test set.

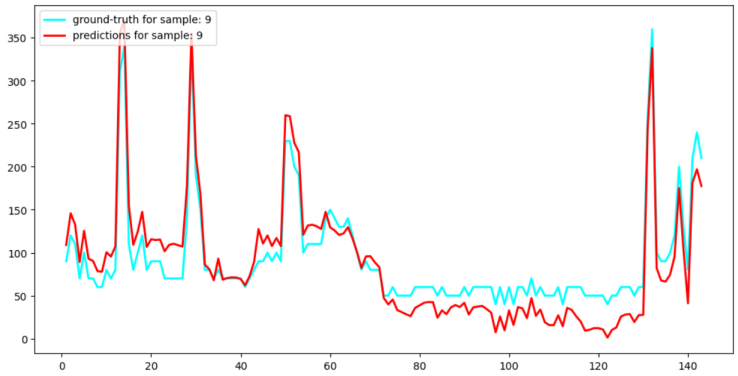

Fig. 17. Transformer prediction vs Ground Truth on random sample with 144 lags

10 Observations - Comparative Analysis

Comparing the MSE and RMSE plots for the three models (GBM, LSTM, Transformers), we can see that transformers have the least error, LSTM have the largest error while GBM is set in the middle and its error is closer to transformers errors'. So we can observe that the transformer model has the best performance between the three models because of its capability to learn long-term patterns. GBM regressor despite the fact that is not the most appropriate for data sequences and it does not take into account the lags performs pretty good. As we observe in this specific task, LSTM model struggles to capture long-term dependencies.

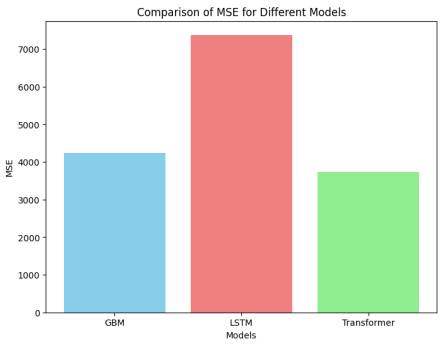

Fig. 18. Comparative MSE plot

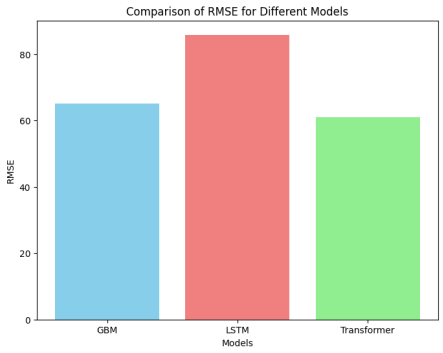

Fig. 19. Comparative RMSE plot

A Appendix - Pairplot and Correlation Heatmap figures

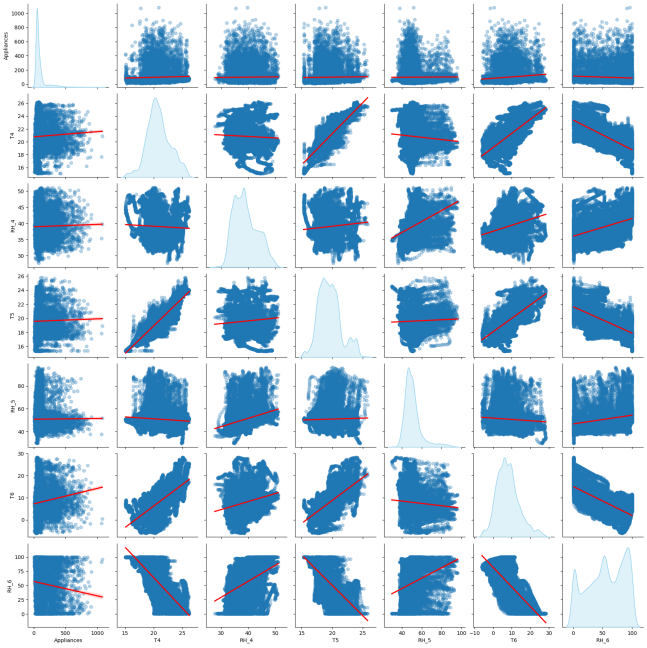

Fig. 20. Pairplot of group 2

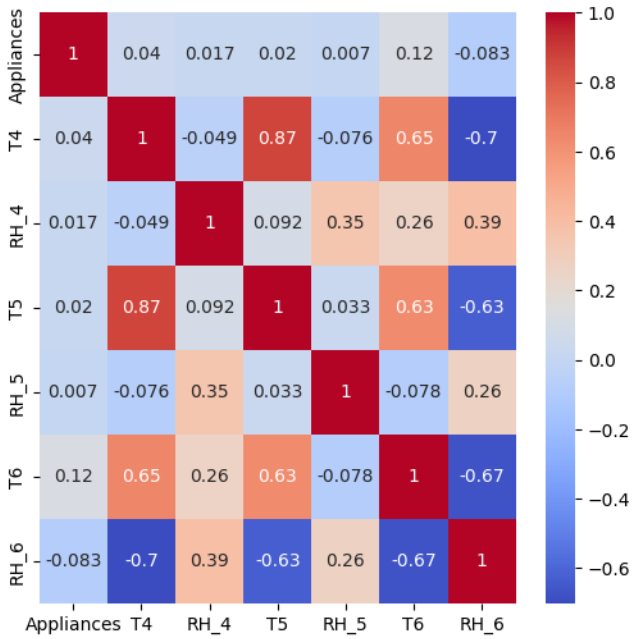

Fig. 21. Correlation heatmap of group 2

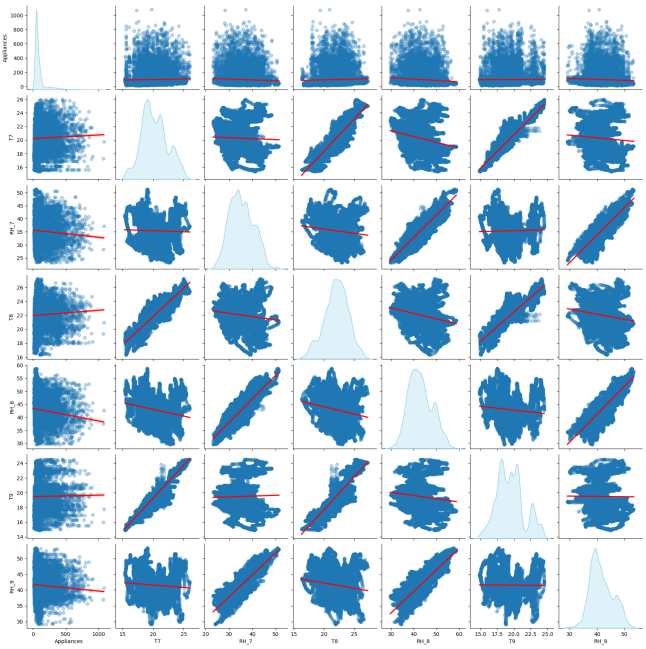

Fig. 22. Pairplot of group 3

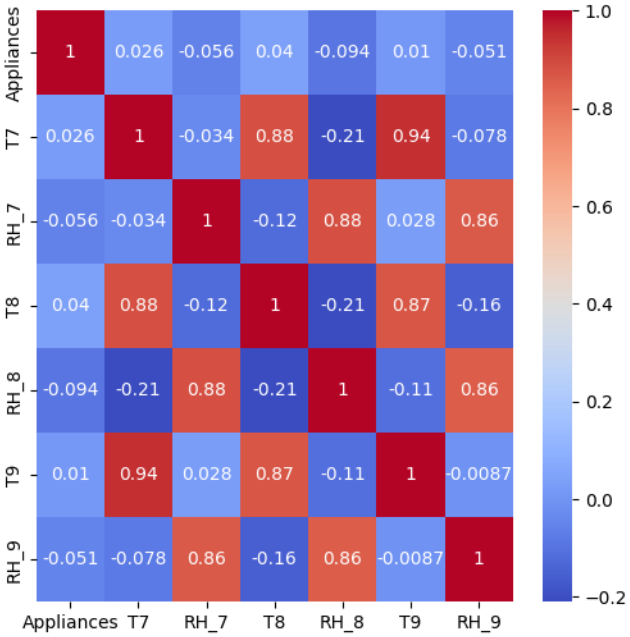

Fig. 23. Correlation heatmap of group 3

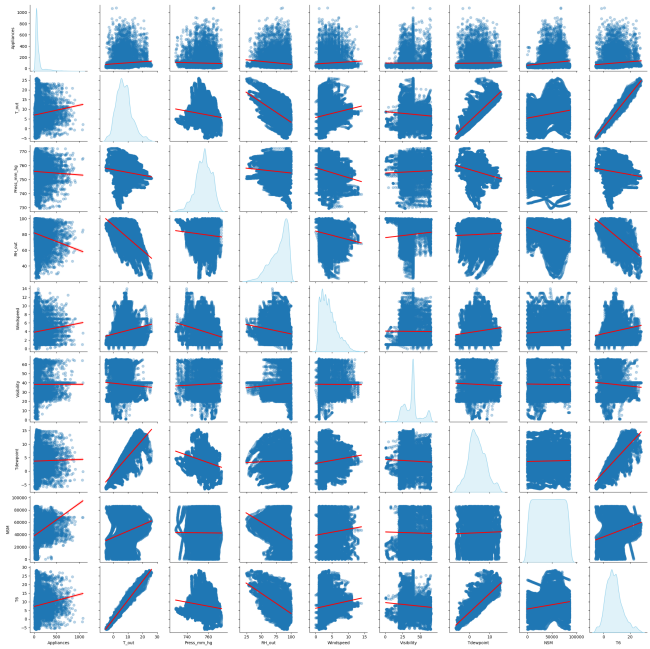

Fig. 24. Pairplot of group 4

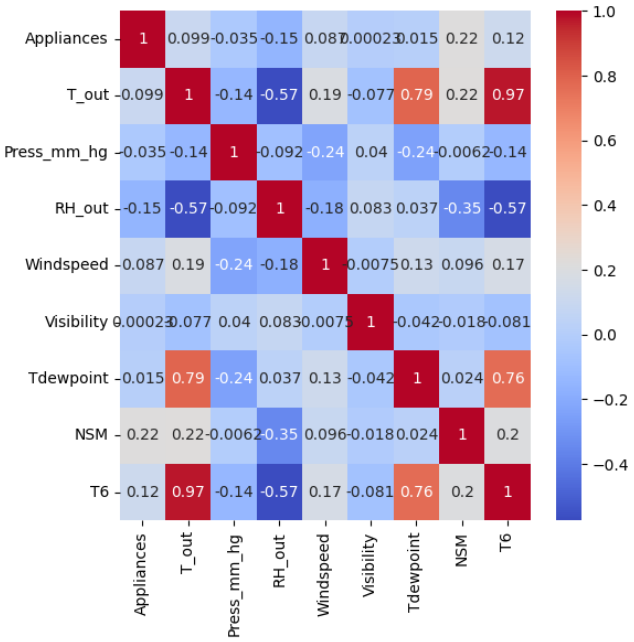

Fig. 25. Correlation heatmap of group 4

References

1. Candanedo, Luis M. Ibarra et al. “Data driven prediction models of energy use of appliances in a low-energy house.” Energy and Buildings 140 (2017): 81-97.
2. Nosek, G. (2022a) Time Series stock prediction with LSTM, Medium. Available at: https://gabenosek.medium.com/time-series-stock-prediction-with-lstm-eb04f2224c22.
3. Ollion, Charles. dlexperiments: Transformers for Time Series. GitHub, 2025, https://github.com/charlesollion/dlexperiments/tree/master/7-Transformers-Timeseries
4. Bishop, Christopher, and Hugh Bishop. Deep Learning: Foundations and Concepts. Springer, November 2023.
5. Bijit. ”Transformers Like Informer Are Revolutionizing Time Series Forecasting.” Medium, August 12, 2024, https://medium.com/@bijit211987/transformers-like-informer-arrevolutionizing-time-series-forecasting-f4e4ebd7db1b.