Comparative Analysis of Deep Learning Techniques on Electricity Time Series Data

Introduction

Abstract

Management and efficient operations in critical infrastructure such as Smart Grids take huge advantage of accurate power load forecasting which, due to its nonlinear nature, remains a challenging task. Recently, deep learning has emerged in the machine learning field achieving impressive performance in a vast range of tasks, from image classification to machine translation. Applications of deep learning models to the electric load forecasting problem are gaining interest among researchers as well as the industry, but a comprehensive and sound comparison among different architectures is not yet available in the literature. This work aims at filling the gap by reviewing and experimentally evaluating a real-world dataset, by contrasting deep learning architectures on the short-term forecast (one day ahead prediction) and long-term forecast (multi-step predictions). Specifically, the project focuses on comparing traditional tree-based models against the recurrent neural networks to study the feature importances extracted from the dataset.

Motivation

This project would be a foundation for my MRP which is investigating the role of various fundamentals such as oil or other commodities in predicting the USD - CAD exchange rate volatility, using time series analysis.

Literature review

Augmented Out-of-Sample Comparison Method for Time Series Forecasting Techniques[3]

- The study compares well-known auto-regressive integrated moving average (ARIMA) models with recurrent neural network (RNN), based models, using Turkish electricity data.
- It concludes that RNN-based models outperform ARIMA models, and as the length of forecast interval increases, the performance gap widens between these two approaches

Time Series Forecasting for Patient Arrivals in Online Health Services[4]

- The study uses patient arrival data and applies both statistical forecasting techniques (e.g., exponential smoothing and ARIMA) and machine-learning models (e.g., random forest and long short-term memory networks) to predict the patient volume.
- It evaluates the performances of alternative prediction methods by using MAE, NRMSE, and ND metrics.
- The results show that the long short-term memory networks and ARIMA outperform other algorithms, however, the performances of the prediction models are highly impacted by the dataset size.

Explainable boosted linear regression for time series forecasting[5]

- The study proposes an explainable boosted linear regression (EBLR) algorithm for time series forecasting, which is an iterative method that starts with a base model and explains the model's errors through regression trees.
- It observes that EBLR substantially improves the base model performance through extracted features, and provides comparable performance to other well-established approaches.

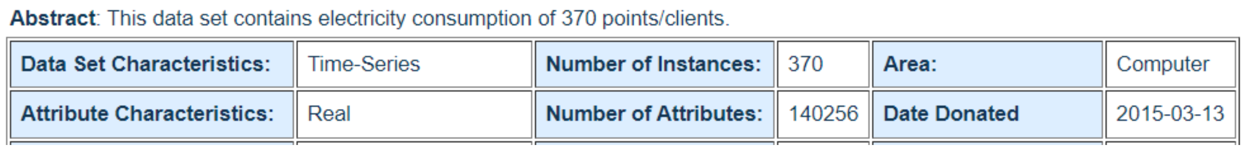
Predicting the Number of Reported Bugs in a Software Repository[6]

- In this study, they examine eight different time series forecasting models, including LSTM, ARIMA, and Random Forest Regressor, and assess the impact of exogenous variables such as software release dates by incorporating those into the prediction models.
- We analyze the quality of long-term prediction for each model based on different performance metrics.

Methodology

Data Description

Link: http://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014



Abstract: This data set contains electricity consumption of 370 points/clients.

Data Set Characteristics:	Time-Series	**Number of Instances:**	370	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	140256	**Date Donated**	2015-03-13

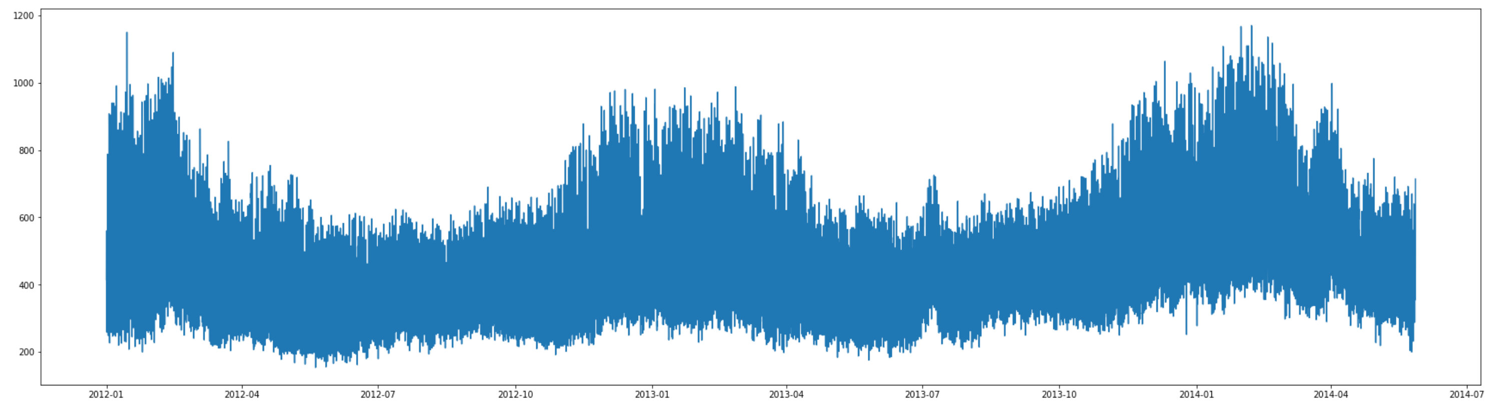
Summary:

- Each column represents one client. Some clients were created after 2011. In these cases, consumption was considered zero.
- All-time labels report to Portuguese hour.
- However, all days present 96 measures (24*4). Every year on March time change day (which has only 23 hours) the values between 1:00 am and 2:00 am are zero for all points. Every year in October time change day (which has 25 hours) the values between 1:00 am and 2:00 am aggregate the consumption of two hours.

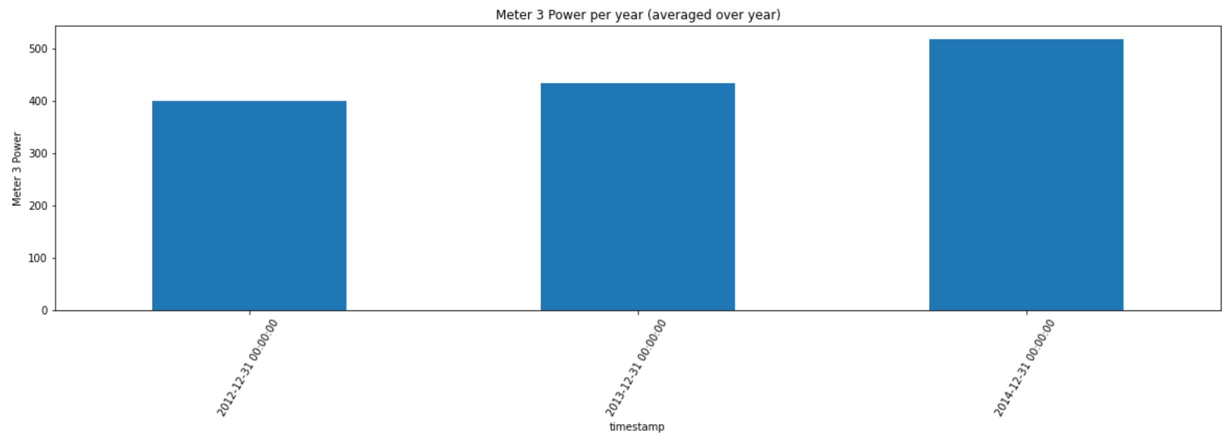
In this project, I have selected one meter/ client attribute (MT_3) and performed time-series forecasting on that.

EDA

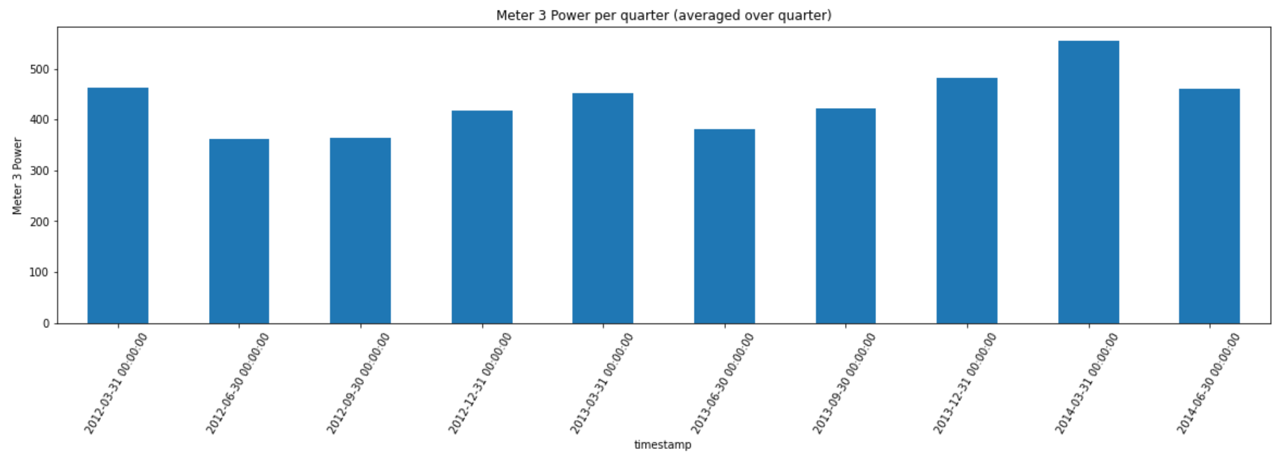
The below plot shows the overall distribution of the MT_3 meter reading values:



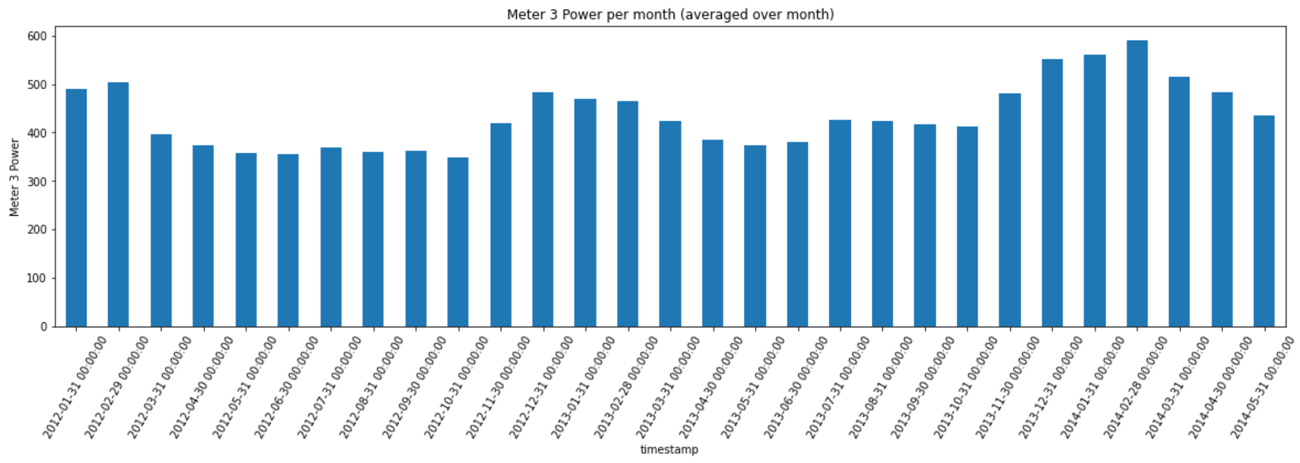
The below plot shows the Power consumption averaged yearly:



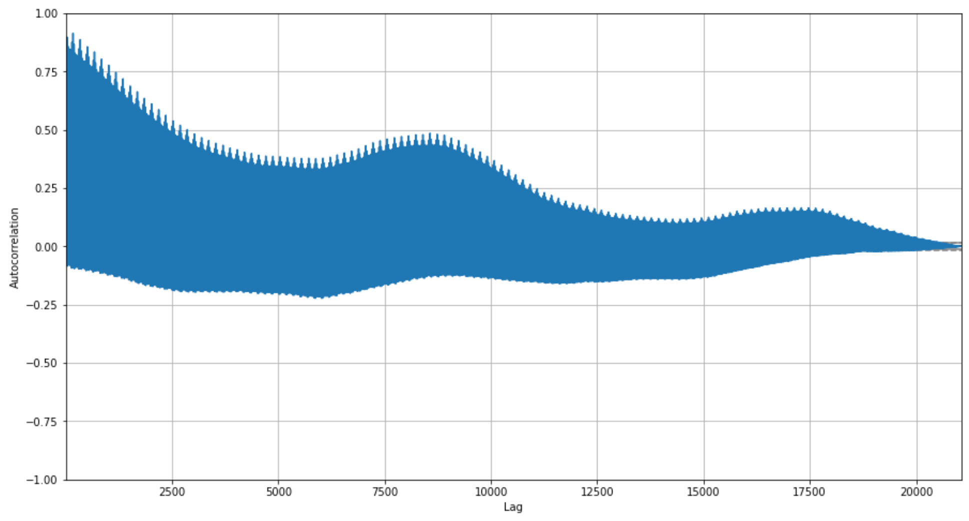
The below plot shows the Power consumption averaged quarterly:



The below plot shows the Power consumption averaged monthly:



The below graph shows the Autocorrelation plot of the data:



ADF Test:

The Adder-Fuller test gave an output p-value of **0.000003**. Since the value is <= 0.05 the data does not have a unit root and is stationary, so we can continue our forecasting.

Feature Engineering - No Lag

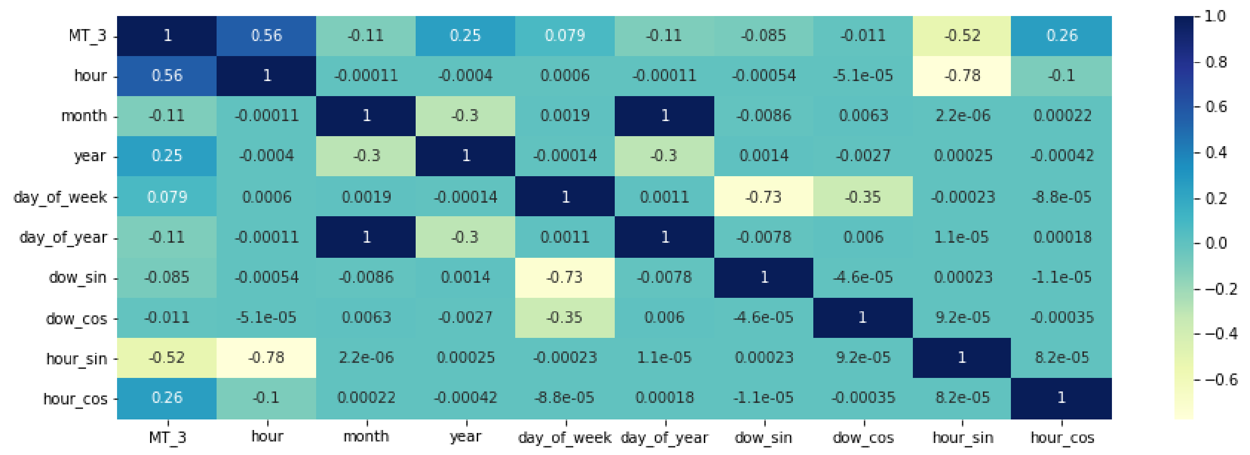
The first step is to extract time-series features from the hourly timestamp index provided in the dataset. Features such as hour, month, year, day of the week, etc are extracted from the timestamp. Furthermore, I add cyclic features for the day of week and hour features respectively.

Dataset Snapshot

	MT_3	hour	month	year	day_of_week	day_of_year	dow_sin	dow_cos	hour_sin	hour_cos
0	415.0	0	1	2012	6	1	-0.781831	0.62349	0.000000	1.000000e+00
1	556.0	1	1	2012	6	1	-0.781831	0.62349	0.258819	9.659258e-01
2	560.0	2	1	2012	6	1	-0.781831	0.62349	0.500000	8.660254e-01
3	443.0	3	1	2012	6	1	-0.781831	0.62349	0.707107	7.071068e-01
4	346.0	4	1	2012	6	1	-0.781831	0.62349	0.866025	5.000000e-01
...	...	...	...	...	...	...	...	...	...	...
21063	376.0	15	5	2014	1	147	0.781831	0.62349	-0.707107	-7.071068e-01
21064	368.0	16	5	2014	1	147	0.781831	0.62349	-0.866025	-5.000000e-01
21065	394.0	17	5	2014	1	147	0.781831	0.62349	-0.965926	-2.588190e-01
21066	469.0	18	5	2014	1	147	0.781831	0.62349	-1.000000	-1.836970e-16
21067	714.0	19	5	2014	1	147	0.781831	0.62349	-0.965926	2.588190e-01

21065 rows × 10 columns

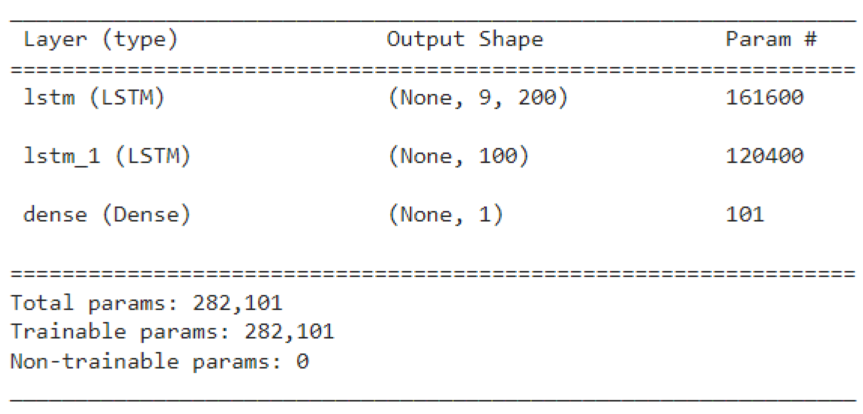
Correlation Plot



Data Modeling

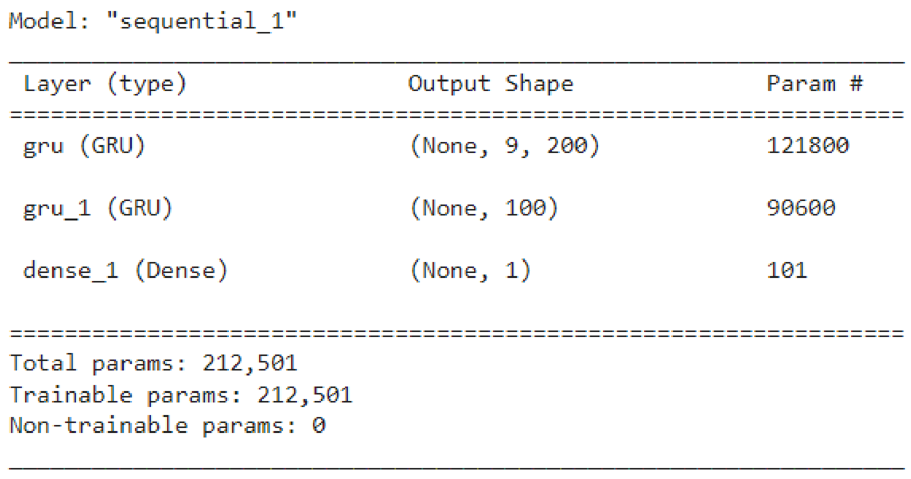
We start by modeling the above dataset using two techniques, tree-based ensemble models and RNNs namely LSTM and GRU.

LSTM Architecture:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 9, 200)            161600

 lstm_1 (LSTM)               (None, 100)               120400

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 282,101
Trainable params: 282,101
Non-trainable params: 0
_________________________________________________________________
```

GRU Architecture:



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 9, 200)            121800

 gru_1 (GRU)                 (None, 100)               90600

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 212,501
Trainable params: 212,501
Non-trainable params: 0
_________________________________________________________________
```

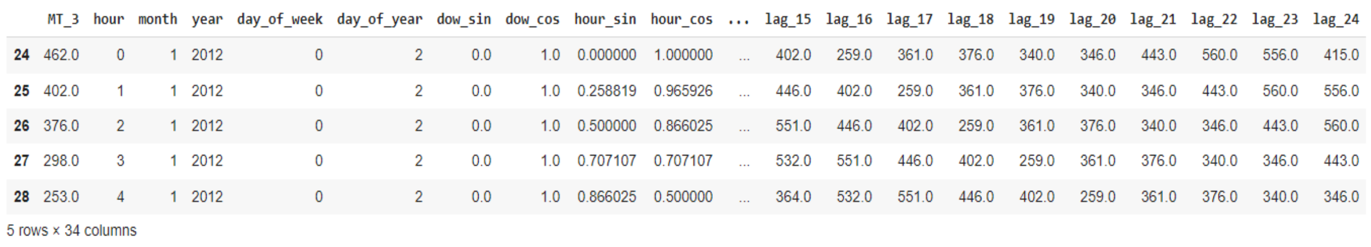
The ensemble models include Random Forest Regressor, Gradient Boosted Regressor, and XGBoost Regressor.

Random Forest: n_estimators=1000, max_features=1
GBR: n_estimators=3000, max_features='auto', criterion='squared_error'
XGBoost: n_estimators=1000, max_features=1

Feature Engineering - 24 Hour Lag

Apart from the above features, we add the 24-hour lag features.

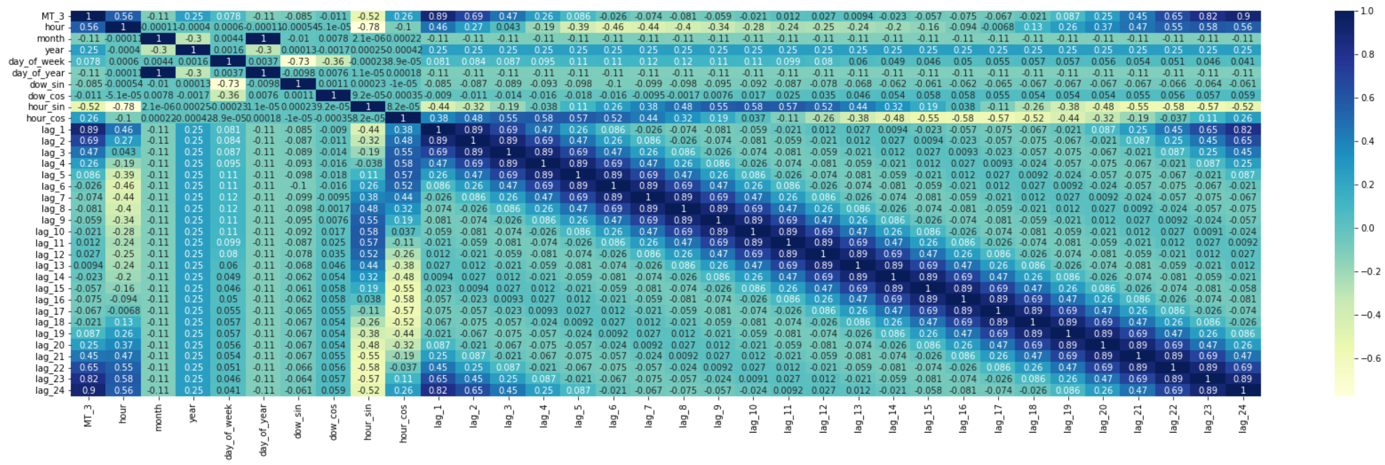
Dataset Snapshot



	MT_3	hour	month	year	day_of_week	day_of_year	dow_sin	dow_cos	hour_sin	hour_cos	...	lag_15	lag_16	lag_17	lag_18	lag_19	lag_20	lag_21	lag_22	lag_23	lag_24
24	462.0	0	1	2012	0	2	0.0	1.0	0.000000	1.000000	...	402.0	259.0	361.0	376.0	340.0	346.0	443.0	560.0	556.0	415.0
25	402.0	1	1	2012	0	2	0.0	1.0	0.258819	0.965926	...	446.0	402.0	259.0	361.0	376.0	340.0	346.0	443.0	560.0	556.0
26	376.0	2	1	2012	0	2	0.0	1.0	0.500000	0.866025	...	551.0	446.0	402.0	259.0	361.0	376.0	340.0	346.0	443.0	560.0
27	298.0	3	1	2012	0	2	0.0	1.0	0.707107	0.707107	...	532.0	551.0	446.0	402.0	259.0	361.0	376.0	340.0	346.0	443.0
28	253.0	4	1	2012	0	2	0.0	1.0	0.866025	0.500000	...	364.0	532.0	551.0	446.0	402.0	259.0	361.0	376.0	340.0	346.0

5 rows × 34 columns

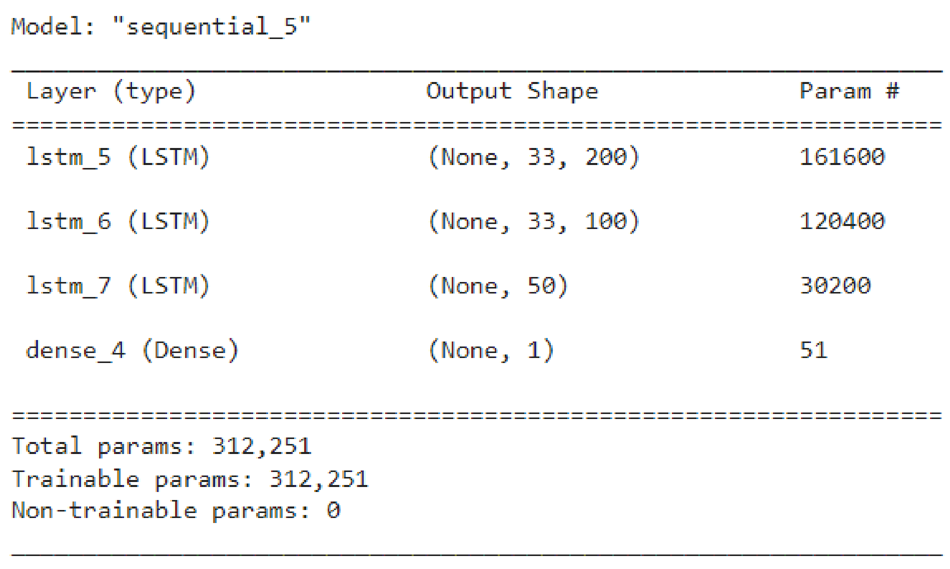
Correlation Plot



Data Modeling

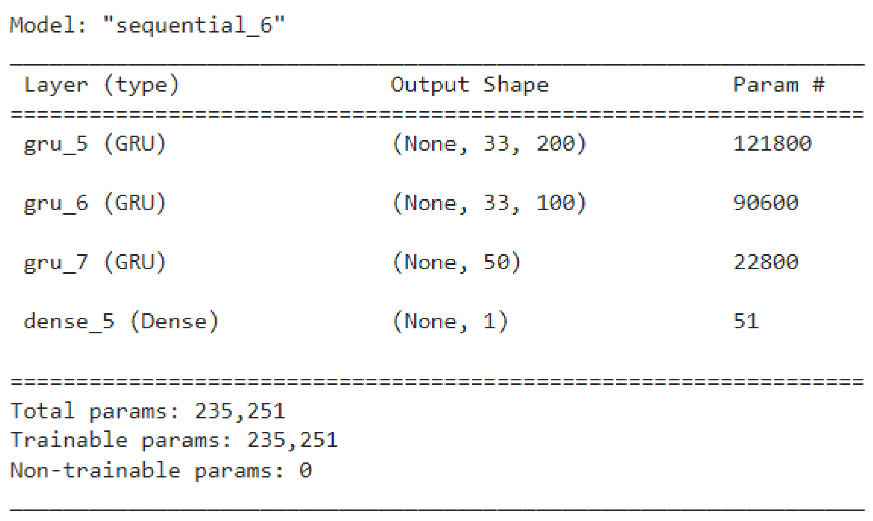
Here we model the above dataset using RNNs namely LSTM and GRU.

LSTM Architecture:



```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_5 (LSTM)               (None, 33, 200)           161600

 lstm_6 (LSTM)               (None, 33, 100)           120400

 lstm_7 (LSTM)               (None, 50)                30200

 dense_4 (Dense)             (None, 1)                 51

=================================================================
Total params: 312,251
Trainable params: 312,251
Non-trainable params: 0
_________________________________________________________________
```

GRU Architecture:



```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_5 (GRU)                 (None, 33, 200)           121800

 gru_6 (GRU)                 (None, 33, 100)           90600

 gru_7 (GRU)                 (None, 50)                22800

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 235,251
Trainable params: 235,251
Non-trainable params: 0
_________________________________________________________________
```

Once the modeling is done, we evaluate the models by running 1-step and multi-step predictions in windows of 6,12,24, and 48 hours. We do this by creating samples of these hourly prediction windows and then averaging the scores.

Evaluation Metrics

MAPE: Mean Absolute Percentage Error

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - F_i|}{A_i}$$

A_i is the actual value

F_i is the forecast value

n is total number of observations

MAE: Mean Absolute Error

$$MAE = \overset{\text{Divide by the total number of data points}}{\boxed{\frac{1}{n}}} \underset{\text{Sum of}}{\Sigma} \underbrace{\left| \overset{\text{Actual output value}}{\boxed{y}} - \overset{\text{Predicted output value}}{\boxed{\hat{y}}} \right|}_{\text{The absolute value of the residual}}$$

RMSE: Root Mean Squared Error

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

NRMSE: Normalized Root Mean Squared Error

$$\mathrm{NRMSE}(y, \hat{y}) = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}}{\frac{1}{N}\sum_{i=1}^{N}|y_i|},$$

Model Interpretation

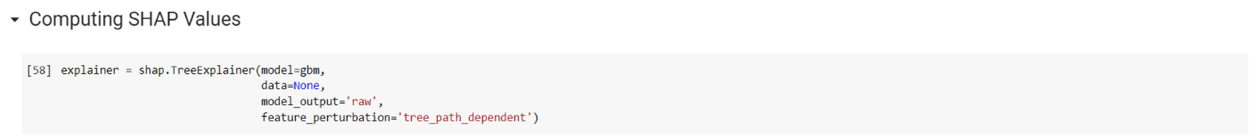
SHAP

“Interpretability is the degree to which a human can consistently predict the model’s result [7]” can be found in machine learning literature. The higher the interpretability of a machine learning model, the easier it is for someone to comprehend why a certain prediction was made by the model.

SHAP (SHapley Additive exPlanation) [8] is a game-theoretic approach to explain the output of any machine learning model. The goal of SHAP is to explain the prediction for any instance x_i as a sum of contributions from its individual feature values. Individual feature values are assumed to be in a cooperative game whose payout is the prediction. In this setting, Shapley values provide a means to fairly distribute the payout among the feature values.

Tree SHAP

Tree SHAP is an algorithm to compute exact SHAP values for Decision Trees based models [10].



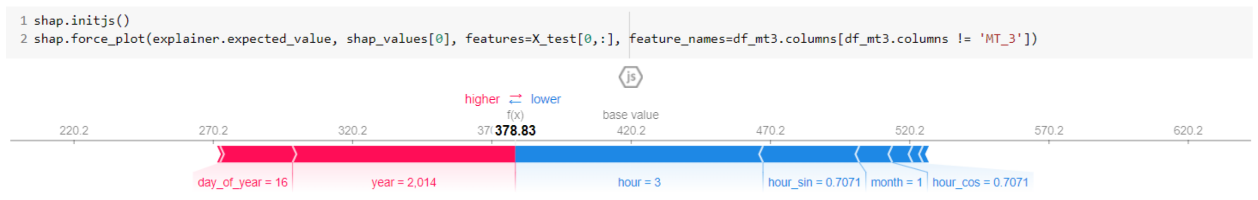
▾ Computing SHAP Values

```
[58] explainer = shap.TreeExplainer(model=gbm,
                                    data=None,
                                    model_output='raw',
                                    feature_perturbation='tree_path_dependent')
```

- model: A tree-based model.
- data: Unless provided, the training dataset will be used as explained in the algorithm section above.
- feature_perturbation: Can take two values. ‘tree_path_dependent’ is the default value when the data argument is None. ‘interventional’ is the default value if a data argument is provided. The interventional approach uses the shared dataset to compute conditional expectations in the presence of correlated input features. “tree_path_dependent” approach on the other hand uses the training dataset and the number of samples that fall into each node as explained in the algorithm section.
- model_output: With model_output=’raw’ (default), SHAP values explain the raw predictions from the leaf nodes of the trees. Since most gradient boosting classification models predict logit (log-odds) in their leaf nodes, SHAP values

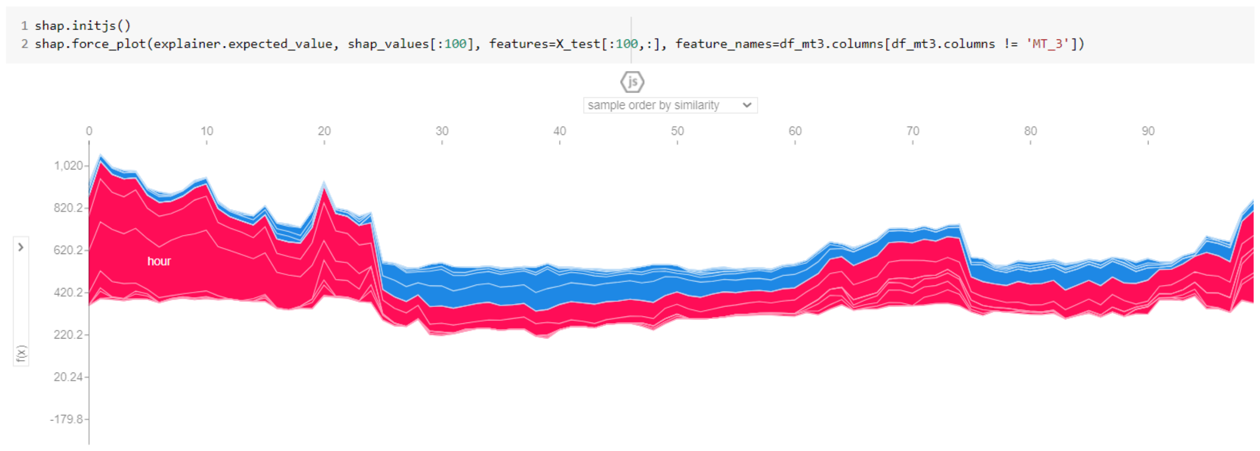
explain the logit prediction for GBM models by default. Other possible values are "probability", "log_loss", or any model method name. With "log_loss" SHAP values explain the log of the model's loss function.

Explaining a Single Prediction:



```
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[0], features=X_test[0,:], feature_names=df_mt3.columns[df_mt3.columns != 'MT_3'])
```

Explaining Predictions for a More Than One Sample:



```
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[:100], features=X_test[:100,:], feature_names=df_mt3.columns[df_mt3.columns != 'MT_3'])
```

Summary Plots:
shap.summary_plot() creates a density scatter plot of SHAP values for each feature to identify how much impact each feature has on the model output. Features are sorted by the sum of the SHAP value magnitudes across all samples. The summary plots are shown in the Results section here.

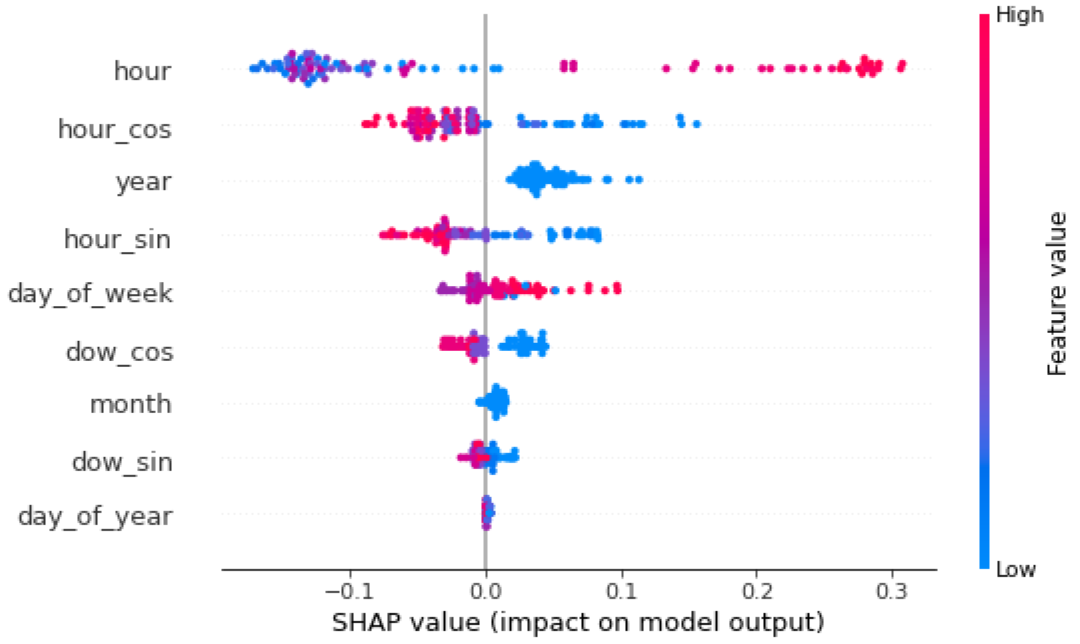
Deep SHAP

Unlike other black-box machine learning explainers in python, SHAP can take 3D data as an input. In this case, the output will be 3D data containing importance for each sample, time steps, and features in that order. If we sum up the output on the 0th dimension, (i.e. sample axis) we get 2D data containing feature importance as columns and time steps as rows. This way SHAP gives time step-wise feature importance. For instance, this can be extremely useful for economic researchers who can use this knowledge to analyze the relationship between macroeconomic variables and stock prices of the share market.

To compute the SHAP values, we use the DeepExplainer shown below:

```
explainer = shap.DeepExplainer(model, X_train[:1000])
shap_values = explainer.shap_values(X_test[:100], check_additivity=False)
```

Summary Plots:



Detailed plots and their analysis is explained in the Results section here and here.

Experimental Setup

Environment

- The environment used was Google Colab GPU [1].
- Numpy and Pandas libraries were used for data reading, processing, and filtering.
- Scikit-learn and xgboost libraries were used for ensemble models while Keras and TensorFlow were used for creating RNN models.
- Shap was used for interpreting the model results.
- Plotly, seaborn, and matplotlib were used for generating the time-series plots.

Ensemble Models

Different hyperparameters were applied to the ensemble models. Below is the list along with the values that were tried:

- Number of estimators (trees): 100-1500
- Max depth: None-10
- Min sample split: 2-6
- Max features: auto, 5-10

RNN Models

For LSTM and GRU models, the hyperparameters tried were:

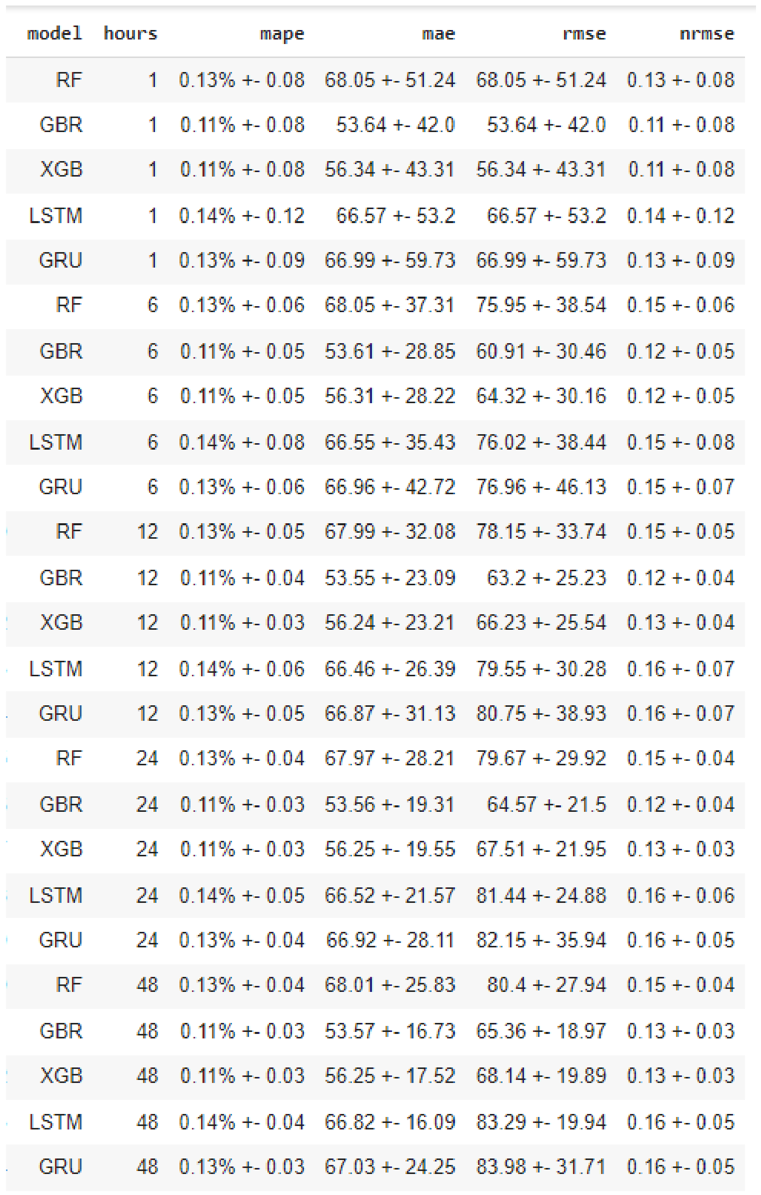
- Different number of layers: 1-4
- Different number of hidden units in each layer: 50-200
- Activation functions: relu, gelu, tanh
- Vanilla LSTM/ GRU
- Stacked LSTM/ GRU
- Optimizer: RMSProp, Adam
- Early Stopping and Checkpoint Callbacks
- Batch Size and Buffer Size

Results

No Lag Results

All the predictions were stored in a csv (attached in the zip in the *sample_results* folder). Depending on the prediction window, the metrics described here were calculated. We can see that GBR gives the best results for the given prediction horizons.

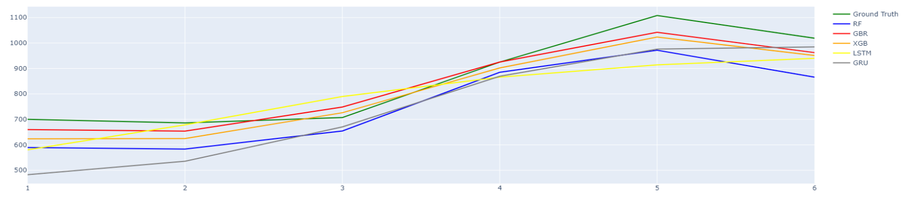
Metrics



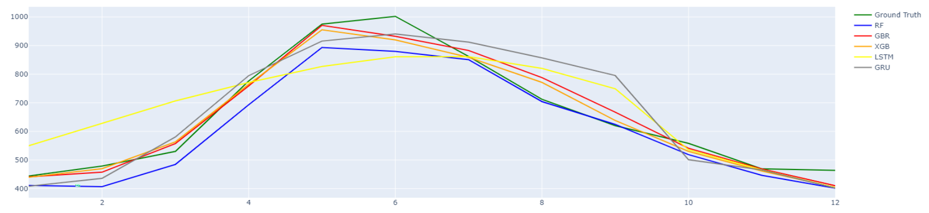
model	hours	mape	mae	rmse	nrmse
RF	1	0.13% +- 0.08	68.05 +- 51.24	68.05 +- 51.24	0.13 +- 0.08
GBR	1	0.11% +- 0.08	53.64 +- 42.0	53.64 +- 42.0	0.11 +- 0.08
XGB	1	0.11% +- 0.08	56.34 +- 43.31	56.34 +- 43.31	0.11 +- 0.08
LSTM	1	0.14% +- 0.12	66.57 +- 53.2	66.57 +- 53.2	0.14 +- 0.12
GRU	1	0.13% +- 0.09	66.99 +- 59.73	66.99 +- 59.73	0.13 +- 0.09
RF	6	0.13% +- 0.06	68.05 +- 37.31	75.95 +- 38.54	0.15 +- 0.06
GBR	6	0.11% +- 0.05	53.61 +- 28.85	60.91 +- 30.46	0.12 +- 0.05
XGB	6	0.11% +- 0.05	56.31 +- 28.22	64.32 +- 30.16	0.12 +- 0.05
LSTM	6	0.14% +- 0.08	66.55 +- 35.43	76.02 +- 38.44	0.15 +- 0.08
GRU	6	0.13% +- 0.06	66.96 +- 42.72	76.96 +- 46.13	0.15 +- 0.07
RF	12	0.13% +- 0.05	67.99 +- 32.08	78.15 +- 33.74	0.15 +- 0.05
GBR	12	0.11% +- 0.04	53.55 +- 23.09	63.2 +- 25.23	0.12 +- 0.04
XGB	12	0.11% +- 0.03	56.24 +- 23.21	66.23 +- 25.54	0.13 +- 0.04
LSTM	12	0.14% +- 0.06	66.46 +- 26.39	79.55 +- 30.28	0.16 +- 0.07
GRU	12	0.13% +- 0.05	66.87 +- 31.13	80.75 +- 38.93	0.16 +- 0.07
RF	24	0.13% +- 0.04	67.97 +- 28.21	79.67 +- 29.92	0.15 +- 0.04
GBR	24	0.11% +- 0.03	53.56 +- 19.31	64.57 +- 21.5	0.12 +- 0.04
XGB	24	0.11% +- 0.03	56.25 +- 19.55	67.51 +- 21.95	0.13 +- 0.03
LSTM	24	0.14% +- 0.05	66.52 +- 21.57	81.44 +- 24.88	0.16 +- 0.06
GRU	24	0.13% +- 0.04	66.92 +- 28.11	82.15 +- 35.94	0.16 +- 0.05
RF	48	0.13% +- 0.04	68.01 +- 25.83	80.4 +- 27.94	0.15 +- 0.04
GBR	48	0.11% +- 0.03	53.57 +- 16.73	65.36 +- 18.97	0.13 +- 0.03
XGB	48	0.11% +- 0.03	56.25 +- 17.52	68.14 +- 19.89	0.13 +- 0.03
LSTM	48	0.14% +- 0.04	66.82 +- 16.09	83.29 +- 19.94	0.16 +- 0.05
GRU	48	0.13% +- 0.03	67.03 +- 24.25	83.98 +- 31.71	0.16 +- 0.05

Prediction Horizon Plots

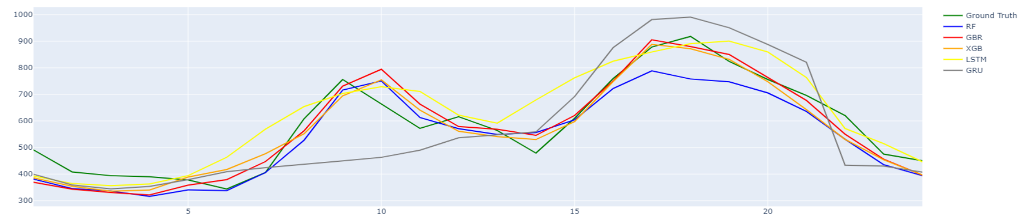
6 Hour Window



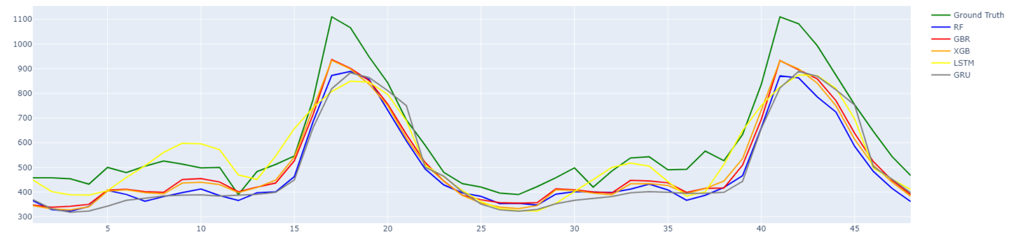
12 Hour Window



24 Hour Window



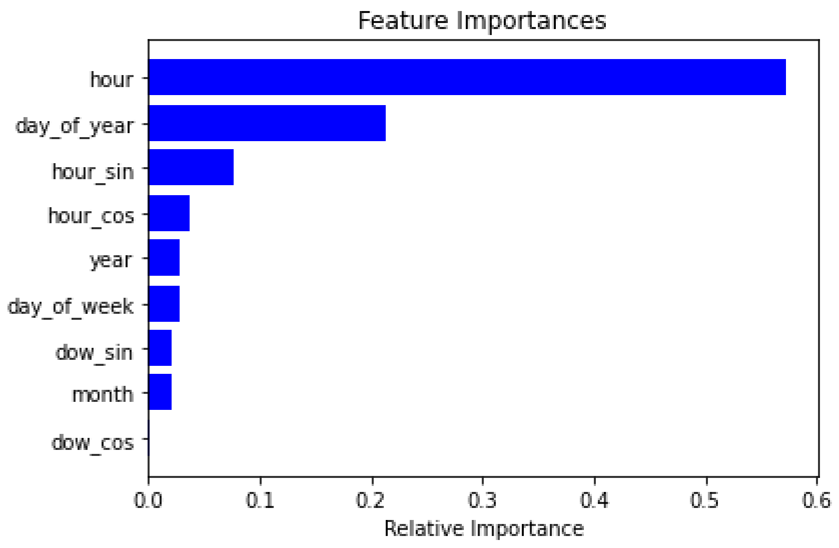
48 Hour Window



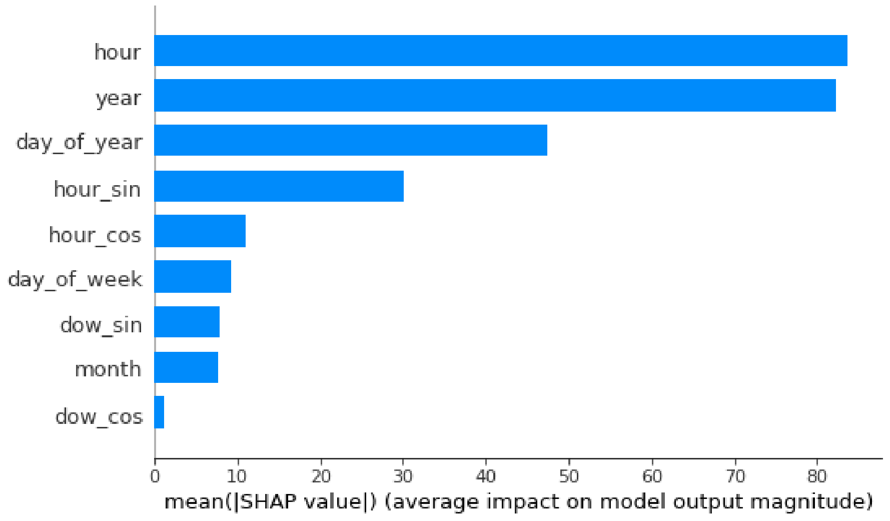
Tree - SHAP

Since GBR was the best of the ensemble models, we select that for interpretation. Below is the Feature Importance vs Shap summary plot (for a test sample of 100 records):

In-built Feature Importance



SHAP Summary Plot

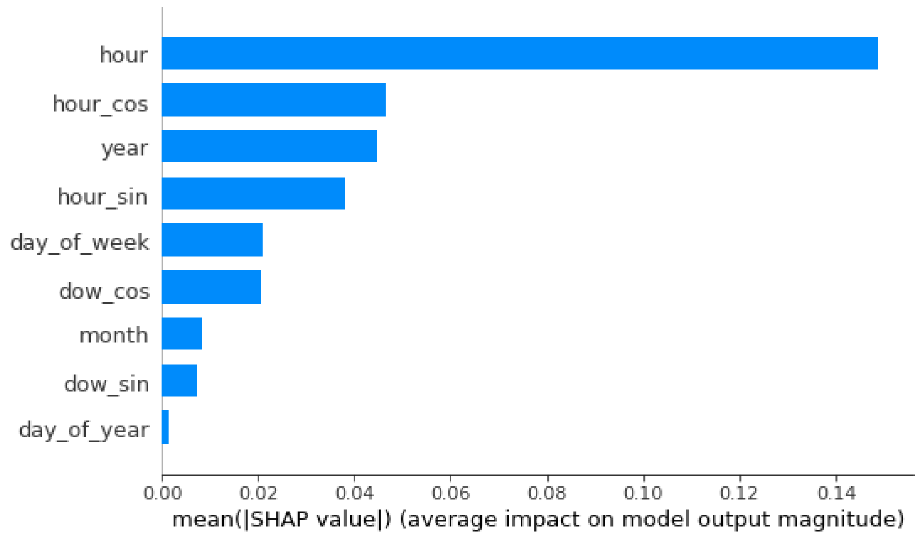


We can clearly see that ***hour*** has the highest importance as expected given the data is sampled hourly, with *year*, *day of the year,* and *cyclic hour* features as the next significant. Hence, we can conclude that both the techniques compare the relative feature significance in a similar manner.

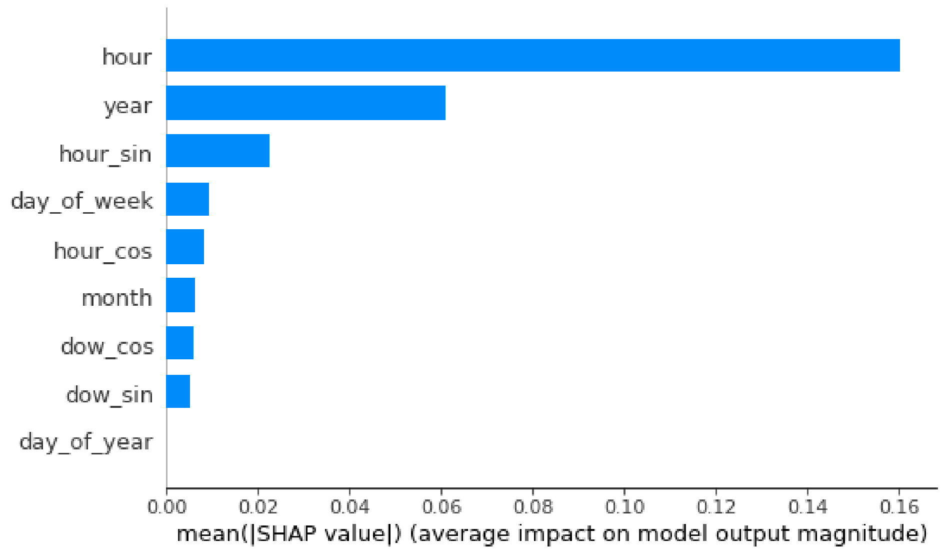
Deep - SHAP

Next, we try to interpret the RNN models using Deep SHAP on the same 100 sample records.

LSTM



GRU

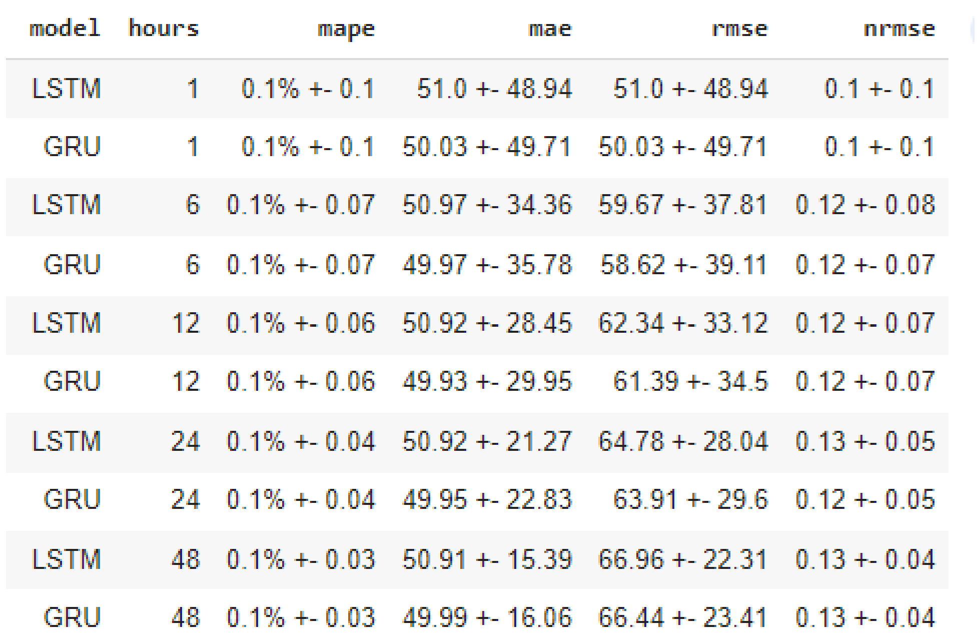


The above summary plots illustrate that while the ***hour*** remains the most important feature, LSTM discovers that the *cyclic hour* features are more important than others. Conversely, GRU attributes a higher value to the *year* feature same as GBR but ignores the *day of year* feature from the predictions completely.

Lag Results

Using the same prediction horizons, metrics were calculated for the LSTM and GRU predictions.

Metrics

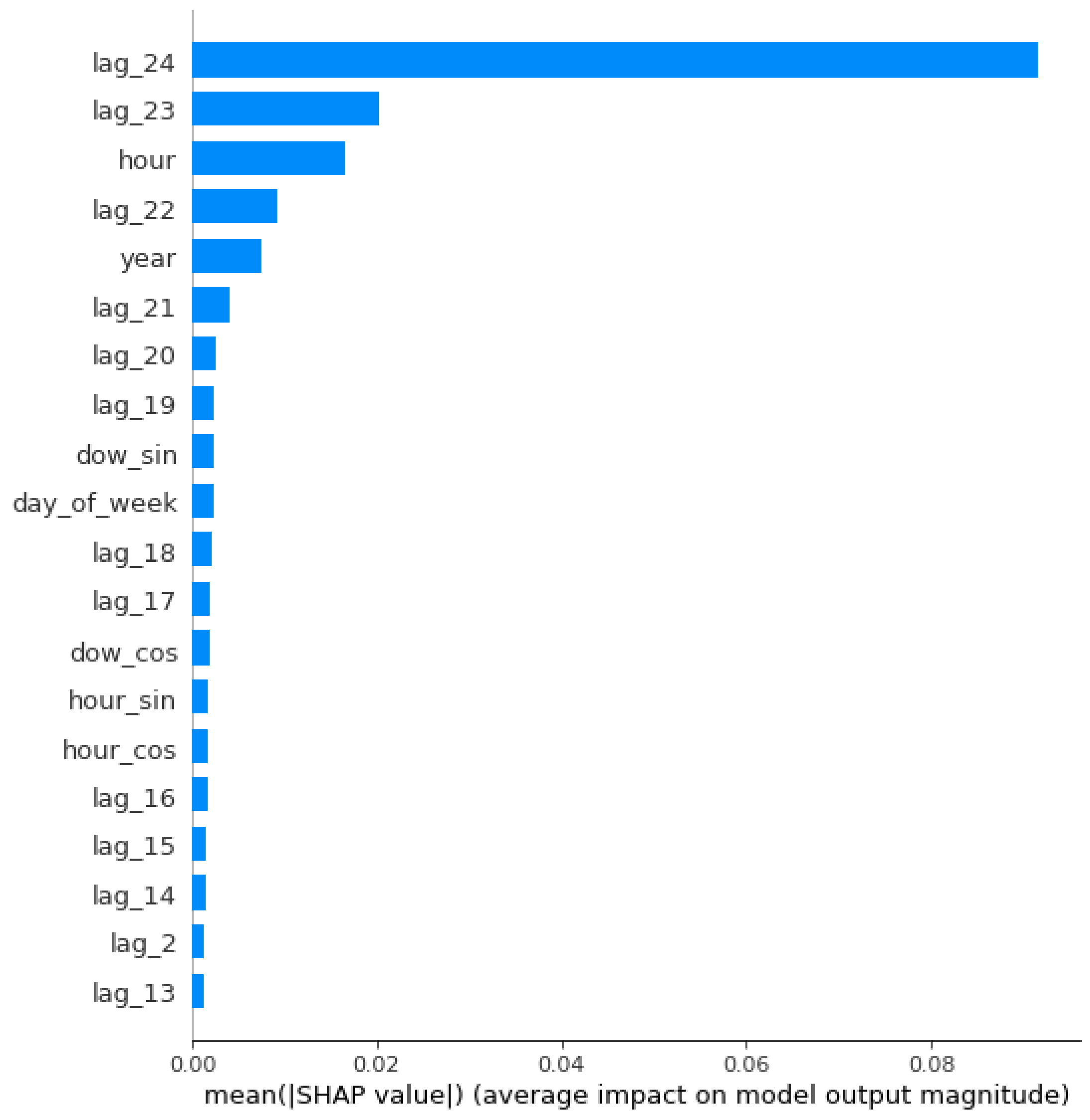


model	hours	mape	mae	rmse	nrmse
LSTM	1	0.1% +- 0.1	51.0 +- 48.94	51.0 +- 48.94	0.1 +- 0.1
GRU	1	0.1% +- 0.1	50.03 +- 49.71	50.03 +- 49.71	0.1 +- 0.1
LSTM	6	0.1% +- 0.07	50.97 +- 34.36	59.67 +- 37.81	0.12 +- 0.08
GRU	6	0.1% +- 0.07	49.97 +- 35.78	58.62 +- 39.11	0.12 +- 0.07
LSTM	12	0.1% +- 0.06	50.92 +- 28.45	62.34 +- 33.12	0.12 +- 0.07
GRU	12	0.1% +- 0.06	49.93 +- 29.95	61.39 +- 34.5	0.12 +- 0.07
LSTM	24	0.1% +- 0.04	50.92 +- 21.27	64.78 +- 28.04	0.13 +- 0.05
GRU	24	0.1% +- 0.04	49.95 +- 22.83	63.91 +- 29.6	0.12 +- 0.05
LSTM	48	0.1% +- 0.03	50.91 +- 15.39	66.96 +- 22.31	0.13 +- 0.04
GRU	48	0.1% +- 0.03	49.99 +- 16.06	66.44 +- 23.41	0.13 +- 0.04

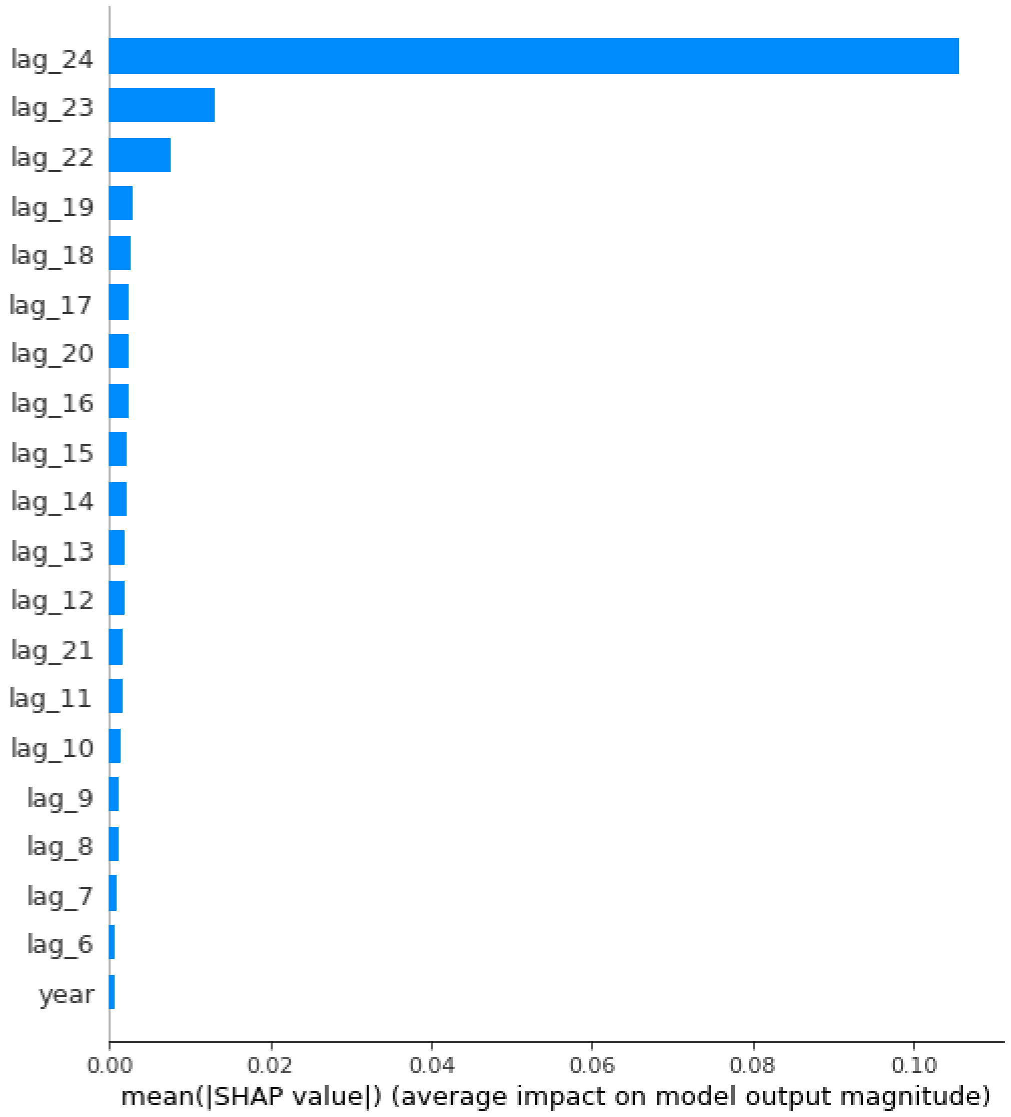
Here we can see that both models have significant improvements over the previous models due to the lag features. So let’s see the impact of the new lag features on the predictions on a sample of 100 records.

Deep - SHAP

LSTM



GRU



Looking at the above two summary plots, we can see that the ***lag_24*** and ***lag_23*** values greatly impact the prediction, representing that the meter readings for any time T are deduced from the T-24 (1 day) and T-23 hour readings. Furthermore, while LSTM still had the ***hour*** as a key feature, GRU did not use that for any predictions and only used the lag values for any prediction.

Threats to Validity

The project was carefully designed to study various techniques on the dataset to ensure valid research outcomes. The models were evaluated on multiple prediction horizons of 6,12,24 and 48 hours. Each batch was evaluated using the metrics defined here and they provide the mean and standard deviation of the results across the batches. This allows to capture the variance in the test set and truly evaluate the models. The ML models were allotted random states to avoid stochasticity, and all deep learning models were tuned based on batch size, learning rate, and the number of epochs. Besides, further parameter tuning for the forecasting models might improve the predictive accuracy. In addition, while we explored many different models in our preliminary analysis, custom deep learning architectures and hybrid models might provide improved prediction performance for our forecasting task.

Conclusion and Future Work

The purpose of this study is to provide an empirical analysis to compare commonly used forecasting methods for the task of predicting the electricity meter readings. By comparing the traditional ensemble tree-based models and the RNNs, it was found that the ensemble model GBR outperformed the neural networks i.e. LSTM and GRU. On the contrary, the RNN models outperformed them when lag values were added as they represented the sequential dataset better. In terms of feature importances, SHAP values provide great insights into what features attribute the most at the time of predictions.

The work presented can be further extended by collecting more data as well as performing a more detailed comparative analysis by including other forecasting models such as Poisson regression and convolutional neural networks. In addition, some external covariates such as weather information can be explored. The problem can also be treated as a multi-step time series classification problem, where the meter readings can be classified into high, medium, and low categories. However, such an approach would require the dataset to be balanced in a manner that the time aspect of the class labels is preserved, as the class ‘low’ would have significantly high instances.

References

1. Setup Google Colab GPU
2. Deep Learning For Time Series Forecasting: The Electric Load Case
3. Augmented Out-of-Sample Comparison Method for Time Series Forecasting Techniques
4. Time Series Forecasting for Patient Arrivals in Online Health Services
5. Explainable boosted linear regression for time series forecasting
6. Predicting the Number of Reported Bugs in a Software Repository
7. Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. “Examples are not enough, learn to criticize! Criticism for interpretability.” Advances in Neural Information Processing Systems (2016)
8. SHAP: A Unified Approach to Interpreting Model Predictions. arXiv:1705.07874
9. SHAP
10. Tree SHAP
11. Deep SHAP