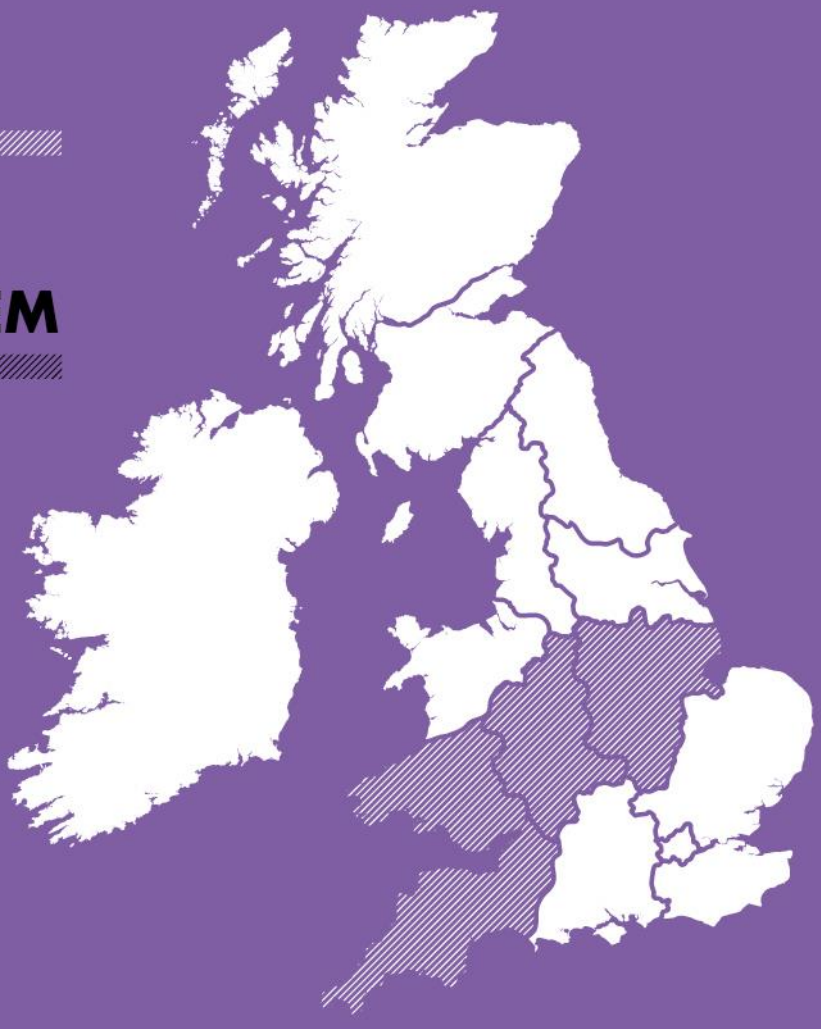
WESTERN POWER DISTRIBUTION

EFFS

ELECTRICITY FLEXIBILITY AND FORECASTING SYSTEM

EFFS
WPD_EN_NIC_003

NIC PROJECT
Forecasting Evaluation Report



RIIO NIC NETWORK INNOVATION COMPETITION

Report Title	:	Forecasting Evaluation Report
Report Status	:	FINAL
Project Reference:	:	WPD/EN/NIC/03
Date	:	06 June 2019

Document Control		
	Name	Date
Prepared by:	Mark Collins, Rachael Taljaard, Pedro Almeida	10/05/2019
Reviewed by:	Euan Davidson, Ivan Todorovic, George Marsh	31/05/2019
Approved by:	Jennifer Woodruff	06/06/2019

Revision History		
Date	Issue	Status
10/05/2019	A	Initial draft
31/05/2019	B	Final draft
06/06/2019	C	Final

Contents

Glossary

Term	Definition
ANM	Active Network Management
APT	Advanced Planning Tool
ARIMA	Auto Regressive Integrated Moving Average
BSP	Bulk Supply Point
DNO	Distribution Network Operator
DSO	Distribution System Operator
EFFS	Electricity Forecasting and Flexibility System
ENA	Energy Networks Association
GPU	Graphical Processor Unit
GSP	Grid Supply Point
I/O	Input / Output
KASM	Kent Active System Management
LSTM	Long Short Term Memory
NIC	Network Innovation Competition
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
SGS	Smarter Grid Solutions
TPW	Tree-structure Parzen Estimator
WPD	Western Power Distribution
XGBoost	Extreme Gradient Boosting

1. Executive Summary

Forecasting will play a key role in the Distribution Network Operator (DNO) to Distribution System Operator (DSO) transition. If DSOs are to manage their networks with the aid of flexibility services, the efficient use of these services is supported by the ability to assess where problems could occur and what level of service might be required to mitigate them ahead of need. The level of efficiency that can be achieved will be driven by the degree of certainty to which behaviour of demand and generation connected to distribution networks can be predicted.

In this report, we explore the forecasting of real and reactive power flows at Primary substations, Bulk Supply Points (BSP), and Grid Supply Points (GSP) over different time horizons: six months ahead; one month ahead; one week ahead; one day ahead; and one hour ahead.

The following methods were investigated using an agile approach that aimed to quickly identify promising methods rather than hand-tune a specific predetermined method:

- **Auto-Regressive Integrated Moving Average (ARIMA)**. A classic statistical modelling approach for building time-series forecasting models.
- **Long Short Term Memory (LSTM) Artificial Neural Networks**. A specific type of deep-learning neural network for learning patterns in time-series data.
- **Extreme Gradient Boosting (XGBoost)**. A machine-learning technique based on decision trees that has performed well in recent machine learning and forecasting competitions.

The key outcomes from the forecasting development delivered within this part of the EFFS project include the following:

- **Model performance.** For the majority of test cases, Extreme Gradient Boosting outperformed the other methods tested. Although, due to different data sets, a direct comparison with forecasting trials in Project KASM cannot be made, based on the same accuracy criteria, LTSM and XGBoost achieve in most cases, the performance requirements for EFFS. Details of the comparison can be found in section 7 but are summarised below.
- **Forecasting at different voltage levels and substation types**. EFFS applied a series of techniques to GSP, BSP, Primary, Load and Generation customers across multiple time horizons. The high-level results include:
    - Techniques based on historical data work best on short time horizons (hour ahead and day ahead). This result is seen across most of the voltage levels, including load and generation customers.
    - For the Primary and BSP cases with low penetration of wind and solar, relative to yearly demand, a feature set containing only temporal trends will provide predictions with acceptable levels of accuracy; for higher penetrations of

renewables, predictions benefit from the addition of weather features to meet accuracy requirements.

- For the GSP case, we selected a GSP including connected solar and wind generation capacities comparable to that of its total demand. The stochastic nature of the renewable generation made it more challenging to identify trends/patterns from historical data for the net real and reactive power flows at the GSP. By forecasting on an individual transformer basis and then aggregating the forecasts yielded better results. Although the results were only for a limited number of substations, this suggests to achieve the desired accuracy, DSOs may look build a large number of specific models to aggregate up to the GSP level.

- **The practicalities of using the techniques**. The results for the performance of the techniques themselves are difficult to decouple from the skill of the data scientist building the model. Although the results do appear to show a clear benefit of the machine learning techniques over ARIMA, this should be treated with caution. With adequate time and skill, one technique could outperform another in the hands of the right data scientist. However, in assessing the different techniques, we have metrics such as training time, tuning time and forecasting time to give an indication of what would be involved to use these techniques at scale. This hints at a potential trade-off between accuracy and – given the way the underlying methods work – what can be automated, reducing the need to have large teams of data scientists to maintain a large set of forecasting models. Understanding model creation and maintenance will be key in how the DSO approach forecasting.

The UK Power Networks (UKPN) Kent Active System Management (KASM) project assessed the accuracy of its proprietary ensemble forecasting method but using different metrics. While a direct comparison may be misleading as the data used was different, the EFFS results compare favourably when looking at the MAPE and RSME/Capacity figures achieved:

- MAPE for Load: KASM-9% day ahead approximation, EFFS-3.5% month ahead average as highlighted in Table 1;
- RMSE/Capacity for Solar: KASM-10% day ahead approximation, EFFS-8.4% day ahead average as highlighted in Table 47; and
- RMSE/Capacity for Wind: KASM-16% day ahead approximation, EFFS-12.5% day ahead average as highlighted in Table 44.

Furthermore, the results achieved in this project can be seen in the next table. Fields highlighted in green illustrate where the forecasters have been assessed to meet the criteria of greater than the target accuracy 80% of the time, which was the performance target set for the EFFS forecasting (see section 7.2 for details). It should be noted that KASM and EFFS used different data sets so the difference in performance may not purely be attributable to the underlying techniques used.

Summary of accuracy (%) results from forecasting methods described in this report.

		Time Horizon				
Use Case	Accuracy	Six Months Ahead	Month Ahead	Week Ahead	Day Ahead	Hour Ahead
UC1 – GSP	>50%	30.61	28.89	25.07	30.95	50.00
	>80%	11.91	11.69	9.42	13.39	25.00
UC2 – BSP	>50%	99.42	99.94	99.78	100.00	100.00
	>80%	79.23	83.50	92.11	97.32	100.00
UC3 – Primary	>50%	98.23	99.98	100.00	100.00	100.00
	>80%	96.05	98.59	99.33	99.70	100.00
UC4 – BSP	>50%	68.99	73.48	73.41	85.12	100.00
	>80%	29.88	33.75	34.10	45.54	52.08
UC5 – Primary	>50%	97.54	97.74	98.96	100.00	100.00
	>80%	87.36	86.97	91.39	98.51	100.00
UC6 – Wind Generation	>50%	37.33	40.35	48.91	87.20	87.50
	>80%	12.76	18.68	27.49	71.73	79.17
UC7 Solar Generation	>50%	72.28	73.08	77.38	76.19	89.58
	>80%	58.16	54.70	52.68	60.12	62.50
UC8 – Large	>50%	N/A	66.66	71.58	79.17	100.00
	>80%	N/A	27.43	29.41	47.32	93.75

Implementing Forecasting Methods: an Open and Reusable Approach.

A key part of the approach taken by the EFFS project team has been to aim for reproducibility of the methods by other parties. All of the forecasters detailed in this report were built using techniques that can be implemented using freely available open source libraries and implemented on a standard open source data science platform.

To allow others to use, reproduce or even improve on the results of the UK-customer funded work in this project, the underlying forecasting tool-chain used by the project's forecasting methods partner has been detailed in this report. This has been done at a suitable level to allow the TRANSITION and FUSION projects to implement specific forecasting models based on the same techniques for their licence areas. In these cases, the performance will be dependent on the quality and quantity of available data.

2. Introduction

Western Power Distribution (WPD) is currently in the middle of its Ofgem Network Innovation Competition (NIC) funded project Electricity Flexibility and Forecasting System (EFFS), with a projected end date of January 2021. This project is key to their transition from Distribution Network Operator (DNO) to Distribution System Operator (DSO) and has the following objectives:

- Enhance the output of the Energy Networks Association (ENA) Open Networks project, looking at the high-level functions a DSO must perform, provide a detailed specification of the new functions validated by stakeholders, and the inclusion of specifications for data exchange;
- Determine the optimum technical implementation to support those new functions;
- Create and test the technical implementation by developing software and integrating hardware as required;
- Use the testing of the technical implementation, which will involve modelling the impact of flexibility services to create learning relevant to forecasting, the likely benefits of flexibility services and the impact of changing network planning standards.

The EFFS project aims to design and implement a system which will allow the planning and dispatch of flexibility services in operational timescales. To do so, EFFS will use forecasts of generation and demand at specific network locations to drive the analysis of what those patterns mean for the distribution network.

Forecasting is not a new art; statistical methods such as Box-Jenkins (auto-regressive moving average) have been used to build demand models for decades. However, forecasting tends to be highly skilled and requires teams of people to craft and maintain forecasting models. A world with diverse small scale to medium scale distributed energy resources interacting with specific local demand patterns means hand-crafted models may not prove practical.

For EFFS, as well as assessing traditional methods, we have looked to recent advances in machine learning and assessed their practical application to forecasting for the timescales required by EFFS.

As part of the EFFS project, WPD is seeking the development of a forecasting system. The ability to forecast load and generation at a range of timescales from an hour ahead to several months ahead will be an essential input to power flow analysis of the network that will highlight possible future network constraints which, depending on the timescale, may result in dispatching services already procured, or procuring services to be used in the future. Generation and demand forecasting is often rudimentary and disconnected from an integrated system. The intention of this project is to provide reliable, repeatable forecasting methods and algorithms to support the development of forecasting capacity. It is WPD's intention that the learning and methods or algorithms will be transferable to the related NIC projects TRANSITION and FUSION, managed by Scottish and Southern Energy Network and Scottish Power Energy Networks respectively.

Smarter Grid Solutions (SGS) was selected as the forecasting partner in and this report outlines the selected and developed forecasting methods, with all necessary information and artefacts to allow for recreation.

2.1. Objectives and Deliverables

Forecasting plays a central role in the EFFS project, to serve as a high-quality input for power systems analysis, the output of which being used in flexibility service procurement decisions. The accuracy of forecasts and understanding likely variability is therefore paramount.

SGS was contracted to provide forecasting for methods for input data for use in the wider EFFS project. The underlying forecasts are then used to drive power systems analysis that determines the effect of load and generation on the network, i.e. circuit flows through load flow analysis.

The aim of the work described in this report was to:

- Use DNO data, along with additional data sources (e.g. weather data), to evaluate a set of different approaches to forecasting.
    - This includes the development of a database to store all the relevant data that is integrated with the forecasting methods.
    - Create a forecasting environment that uses a range of open source forecasting libraries to evaluate statistical methods, machine learning, and deep learning methods.
    - Apply these methods to the following forecasting applications:
        - Load, Power Factor, Generation, Generation Power Factor, Net Load / Generation, Maximum load and Maximum Generation at 33kV, 66kV and 132kV transformers; and
        - Load, Power Factor, Generation, Generation Power Factor, Net Load / Generation, Maximum load and Maximum Generation at 33kV, 66kV and 132kV connected customers.
    - Forecast the parameters above across for the following time horizons:
        - Intraday;
        - Day-ahead;
        - Weak ahead;
        - One month ahead; and
        - Six months ahead.
    - Apply the WPD-defined accuracy evaluation methods to calculate the efficacy of the forecasting methods.

The key deliverables for the work detailed in this report were:

- A Toolchain for building forecasting models (based on open source technology);
- Database schema, including data and test results;
- Scripts to allow replication of results by the EFFS partners; and
- This report gives details of the evaluation of methods and how to replicate the methods.

3. Forecasting Methods

3.1. Background to Forecasting

Forecasting is used when an estimate of uncertain future events is required, with the results primarily used to improve decision-making and planning activities. Forecasts that are produced almost always incorporate some degree of error, however, it is still beneficial to have the limited information provided by a forecast than to plan for the future in ignorance.

There are qualitative forecasting methods based on soliciting opinions that are:

- Focused on collecting opinions from industry stakeholders and experts, meaning they are subjective;
- Useful when past data is unavailable to help inform future trends; and
- Typically applied to medium and long range time horizons.

An example of a qualitative forecasting method is the Delphi Method. This method uses an iterative technique that relies on input from experts. It is based on the principle that forecasts from a structured group will outperform those from an unstructured group. The experts answer questionnaires in rounds, and after each round, the questions are re-asked but an anonymised summary of responses from the previous round is also supplied. It is expected that by providing the information from all the experts the range of answers provided reduces, thus converging on a "correct" answer. This is typically applied in long-range forecasting for technological advances[1,2].

Quantitative forecasting methods use explicit mathematical models to determine future trends as a function of past data. These methods are:

- Useful when historical data is available and can be used as a reliable predictor for the future; and
- Typically applied to shorter-term time horizons.

Time series forecasting is important as so many prediction problems involve some temporal component. It is assumed that patterns are due to time, and historical data patterns are projected into the future. The time series can be broken down into component parts: level, trend, seasonal, cyclical, and random.

The random component is unknown and unpredictable. The cyclical component is due to the longer term cycles and is difficult to identify, and so time series methods generally focus on the identification of all these components, for example, of the seasonal component – a cycle that repeats annually: the trend and level components. The trend component is the optional linear increasing or decreasing behaviour of the series over time, and the level component is the baseline value for the series if it were a straight line[3].

[1] https://personal.ashland.edu/dlifer/internal/omlectureforecasting.pdf
[2] https://www.gwern.net/docs/predictions/2001-armstrong-principlesforecasting.pdf
[3] https://machinelearningmastery.com/time-series-forecasting/

There are a number of questions that also impact the forecasting and the effectiveness of the employed method. These include:

- How much data is available and can it be gathered together in one place? More information can often be advantageous, allowing for greater opportunity to detect patterns.
- What time horizon is the prediction required for? In general, shorter-term predictions are easier to achieve and with greater accuracy or confidence. The farther into the future the prediction is, the more difficult it is to accurately predict what the patterns may be.
- Can forecasts be updated over time or must they remain static? If forecasts can be updated as more information becomes available, often the accuracy can be improved. However, too much information can reduce this accuracy. Therefore, the concept of over and underfitting explains this balance.
- At what time resolution is the forecast required? There is the potential to employ up/downsampling of data should a different resolution of forecast be required. Upsampling sees the creation of new data points when adapting low-resolution dataset (e.g. half hourly) to a high-resolution dataset (e.g. minutely). Downsampling is the opposite action.

The importance of data in the forecasting process links to another concern for time series prediction – the quality of the data. Quite often some degree of data cleansing will be required. This can be due to bad or missing data in the dataset, or simply due to the fact the data is in a format or resolution not suitable for forecasting purposes. It is always worthwhile to spend some time scrutinising the input data to identify if there are erroneous values, errors in data logging and if outliers are credible.

3.1.1. Underfitting/Overfitting

In statistical analysis, overfitting is the production of an analysis which corresponds too closely or exactly to a particular set of data, and may, therefore, fail to fit additional data or predict future observations reliably[4]. Likewise, underfitting occurs when the method cannot adequately extract or identify trends in the data. This can appear in machine learning and can sometimes be referred to as over or under training. Overfitting can occur due to there being a mismatch between the criteria used for selection of the model and that used to determine the suitability of the model. An example of this is a model being selected for maximising its performance on training data, but its suitability may be determined by its ability to perform well on unseen data. Overfitting occurs when the model memorises the training data rather than learning to generalise from a trend[5].

When training a machine learning method the performance progresses from underfitting, where it is training with too little data or too few features, and does not identify key elements of the trends, to overfitting where too much information is provided. The optimal

[4] https://en.oxforddictionaries.com/definition/overfitting
[5] https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765

point lies between these two points, and the investigation of the impact of different quantities of training data and features will help the user determine what results in this optimal point.

3.2. Investigation on Forecasting Techniques

A time series ready for forecasting should allow decomposition into four basic constituents:

- **Level** – baseline value that would correspond to the series if it were a straight line;
- **Trend** – the increasing or decreasing linear slope of the time series over time;
- **Seasonality** – the cyclical patterns of the curve over time;
- **Noise** – the variability of the curve that cannot be explained by the model.

Several questions condition what can be done with the data or how accurate the results will turn out to be:

- **Amount of data** – more data generally allows for better forecasts and analysis;
- **Forecast horizon** – shorter time horizons are easier to predict with greater confidence;
- **Frequency of historical data updates** – models can be retrained as frequently as there are updates of the historical data and therefore the accuracy of the forecasts can be improved over time;
- **Required granularity** – the frequency of the required output conditions down-sampling or up-sampling actions that can be made in modelling.

Before proceeding to forecasting it is also necessary to analyse the input historical data and oftentimes some data manipulation is required by cleaning, scaling or transforming the original dataset:

- **Frequency** – when frequency is too high or too low or data points are unevenly spaced there may be a requirement to resample the data;
- **Outliers** – wrong or extreme outlier values may need to be identified or handled;
- **Missing** – missing values or gaps in the dataset may need to be interpolated or complemented with additional sources.

In this analysis a spectrum of methods have been covered; both classical statistical methods and artificial intelligence based methods.

Classical Statistical Method

Classical statistical methods are rooted in inductive inference from data, where the likelihood principle drives the outcome from these methods.

The classical methods analysed included Holt-Winters, exponential smoothing, moving average, Autoregressive Moving Average and Autoregressive Integrative Moving Average (ARIMA).

Artificial Intelligence

Artificial intelligence is where the role of inductive inference is placed in the hands of a machine implementing various types of machine learning algorithms.

Machine learning attempts to achieve an objective without the aid of specific instructions but instead determines patterns and relationships, hidden in data, to instruct its actions on achieving the objective.

There are many machine learning algorithms, of varying approaches, since supervised learning approaches deal with building a mathematical model of a set of data that contains both the inputs and desired outputs, it provides a structurally sensible approach to the forecasting problem and was therefore selected as the algorithmic and modelling route.

The application of machine learning for forecasting is not new. However, since the turn of this decade, the machine learning community has made inroads into a number of different problems. Advances in neural networks and decision trees for what is sometimes termed "Deep Learning" has resulted in improvements in performance in key problems, such as image recognition where results are so strong the problem could be almost considered to be solved. The same family of techniques can be turned to forecasting.

Part of the way the machine learning community continues to make advances is through the use of benchmark problems and competitions to solve those problems. Forecasting problems feature in the machine learning community Kaggle[6]. Moreover, in the area of load forecasting, the IEEE's Global Energy Forecasting competition[7] (also run on Kaggle) has allowed a number of different techniques to compete against each other, including techniques which employ classical statistical models.

The motivation to investigate machine learning was two-fold; firstly there was the success of specific techniques in the competitions above. National Grid ESO has also recently produced interesting results for solar forecasting using deep learning techniques[8].

Of the AI-Machine Learning based options, different formulations for the supervised learning problems were tested.

- **Neural networks**: Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU) and Long-Short-Term Memory (LSTM).
- **Tree-based methods**: Prophet and XGBoost.

Initial probing funnelled these down to three relevant options: ARIMA as the best option among the conventional methodologies, Recurring Neural Networks with Long-Short-Term Memory as the alternative among Neural Network based models and XGBoost as the key reference in the tree-based approaches.

3.2.1. ARIMA

The Autoregressive Integrative Moving Average (ARIMA) model is a classical time series forecasting technique that results from a generalisation of different methods. ARIMA models

[6] https://www.kaggle.com/
[7] https://www.ieee-pes.org/ieee-pes-announces-the-winning-teams-for-the-global-energy-forecasting-competition-2017
[8] http://powerswarm.co.uk/wp-content/uploads/2018/10/2018.10.18-Bruce-National-Grid-ESO-Deep-Learning-Solar-PV-and-Carbon-Intensity.pdf

can be applied to cases where non-stationarity exists and the integrative part can be applied multiple times to eliminate the non-stationarity.

The autoregressive component of the model establishes a relation where the forecasted variable regresses from its own lagged or historical values. The moving average term links regression error with a linear combination of error terms over time. The integrative establishes that some values result from differentiation of previous values.

The ARIMA model is one of the most widely used forecasting techniques and has proven its value on many applications. Very often it is used in predicting load at a nationwide level. Accuracy levels are high when sufficient data scientist time is spent on modelling and tuning of the model. This means that for applications where a large number of forecasts are required, ARIMA, or other conventional methods, may not be the recommended option as they become impractical.

In terms of development, ARIMA lacked more complete libraries in Python, the language selected for the project. As it is commonly known, R is the reference language for data science, but many libraries were ported and new libraries have been built for Python, allowing data science work to be conducted with Python.

In the case of ARIMA, R is still more complete than Python and even though it was possible to produce results in Python with ARIMA, the quality of results was largely improved when deployed in R. So, for the particular case of ARIMA, R was used in the final testing.

The advantage of using R was the possibility of applying Fourier transforms to capture seasonality patterns, which improves significantly the quality of results. It is necessary to use one Fourier transform per seasonality pattern that is being captured and the frequency of that season is not captured automatically by R, given the complex shape of the input profile. So, when importing the data it is necessary to save multiple copies of it, once per frequency, so as to extract the Fourier transform. Frequency in data science corresponds to the more general knowledge of period in other science fields, so frequency will be defined by the number of steps that form the season, e.g. for half hourly data frequency is 48 for capturing the daily pattern or 336 for the weekly pattern.

The three parameters of the ARIMA model (p, q, d) are automatically computed by a built-in method in R when provided the input data and the external regressors. The external regressors are what in AI based methods are called features and in the case of R will consist of the Fourier transforms of the input time series and other relevant variables such as temperature. Dealing with external regressors for ARIMA requires more significant data preparation than in AI based techniques.

A full theoretical background and tutorials for ARIMA development can be found in the references[9,10,11].

[9] https://www.datascience.com/blog/introduction-to-forecasting-with-arima-in-r-learn-data-science-tutorials
[10] https://people.duke.edu/~rnau/411arim.htm
[11] https://www.kaggle.com/kailex/arima-with-fourier-terms

3.2.2. Long-Short-Term-Memory

Long Short Term Memory (LSTM) is a type of neural network machine learning algorithm. Like all neural networks, it creates a series of relationship connections in the hidden layer (artificial neurons) between the data set inputs to determine possible outputs. Since one of the largest influential relationships for time series data is time, a neural net variant was created, called recurrent neural networks (RNN), where memory is introduced to the algorithmic structure, shown in Figure 1, to temporally link predictions made in the hidden layer with input data to improve output data predictions.

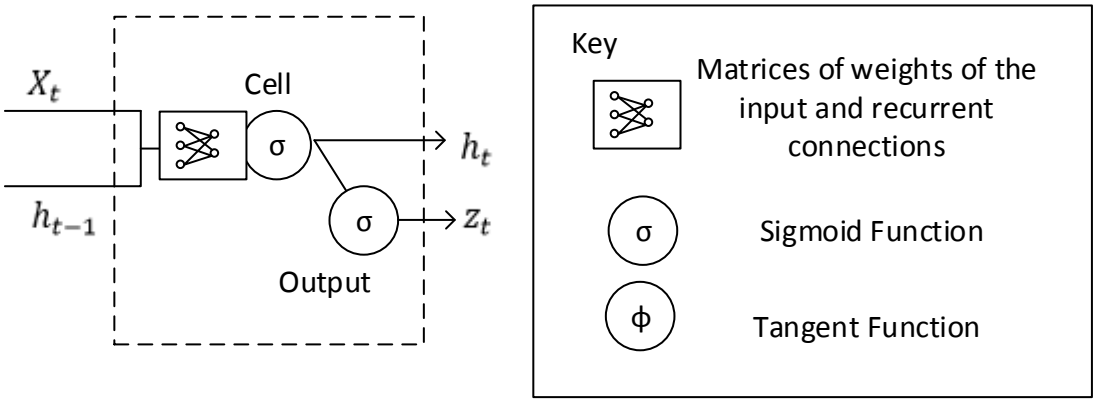


Figure 1: Recurrent Neural Network Architecture

The number of hidden layers, or neurons, can vary dependent on the amount of dependent or independent relationships that may or may not exist between the input data. This concept can be optimised, but in this section, it just needs to be understood that the hidden layer exists to hold these relationships.

The LSTM algorithm improves on the recurrent networks problems, namely the exploding and vanishing gradient problems which do not allow recurrent neural networks to recognise important time series events for unspecified durations. The introduction of long term memory is illustrated in Figure 2.

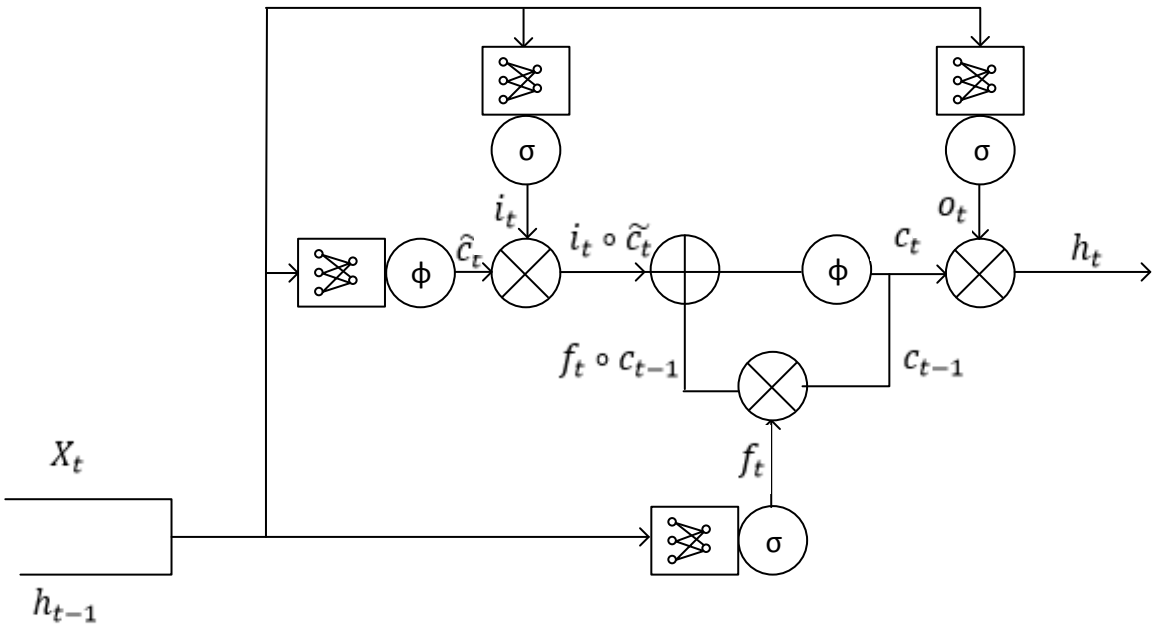


Figure 2: Long Short Term Memory Architecture

The architecture allows the LSTM to hold on to significant events (seasonal variations) while forgetting insignificant ones (erroneous data spikes). It is achieved by updating the cell state with a forget gate. Information, from previous intervals, can now be added or forgot by the cell state, where required, to improve the predictive ability of the cell:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Where:

$f_t \epsilon \mathbb{R}^h$: forget gates activation vector

$i_t \epsilon \mathbb{R}^h$: input gates activation vector

$c_t \epsilon \mathbb{R}^h$: cell state vector

$c_{t-1} \epsilon \mathbb{R}^h$: cell state feedback vector

$\tilde{c}_t \epsilon \mathbb{R}^h$: input modulation gate's activation vector

The superscripts d and h refer to the number of input features and the number of hidden units (neurons).

The output of the forget gate tells the cell state which information to forget:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

Where:

$W \epsilon \mathbb{R}^{hxd}$, $U \epsilon \mathbb{R}^{hxd}$, $b \epsilon \mathbb{R}^h$: are the weight matrices and bias vector parameters.

The input gate determines which information should enter the cells memory:

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

and allows the addition of memory, with the modulation gate:

$$\tilde{c_t} = \phi_c(W_c x_t + U_c h_{t-1} + b_c)$$

allowing for removing memory, creating a selectable input:

$i_t \circ \tilde{c}_t$.

Finally, the output gate determines which values should be moved on to the hidden layer:

$h_t = c_t \circ \phi_c$.

In all the previous operations, the activation functions enable the behaviour desired from the weights being produced in the matrices; the values themselves can be trained using an optimised algorithm to improve the outputs the cell makes to the hidden layer to improve predictions.

Further reading material on LSTM is available in the reference[12].

3.2.3. XGBoost

Extreme Gradient Boosting (XGBoost) is one of the most respected machine learning algorithms for supervised learning. It can tackle regression, classification and ranking problems. Gradient boosting techniques produce forecasts by creating an ensemble of weak prediction models, which in the case of XGBoost are decision trees.

XGBoost like other gradient boosting techniques builds the final model in a stage-wise manner. Yet, it builds a more generic framework by optimising an arbitrary differentiable loss function, which allows control of overfitting and improves performance.

XGBoost is being vastly adopted for its execution speed and the model performance. Existing libraries are widely supported in different platforms and allows parallelisation, distributed computing implementations, out-of-core computing for very large datasets and cache optimisation.

Further reading material on XGBoost is available in footnotes[13,14].

3.2.4. Model Execution Method

Both the machine learning techniques require the definition of a model that consists of several methods:

- **Creation of features** – this is the step where the data that will influence the forecast is defined and prepared in the right format;
- **Training of model** – in this step, the historical data and features are used in combination with the training model to fit the model to the data; the training model requires the definition of hyperparameters that condition the final performance of the prediction;

[12] Deep Learning with Python: Francois Chollet
[13] https://blog.exploratory.io/introduction-to-extreme-gradient-boosting-in-exploratory-7bbec554ac7
[14] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

- **Forecast model** – using the fitted model, the length of the future prediction required is provided and the forecast is produced.

In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins. These contain the data that govern the training process for the prediction. The forecasting model considers three types of data:

- Input data – this is the collection of individual time series profiles used to make predictions.
- Model parameters – these are the variables the chosen model uses to adjust the data.
- Hyperparameters – these variables are not directly related to the training data, but are used to configure the model.

Hyperparameters are user defined and can be optimised using different techniques addressed in the next section.

3.2.5. Hyperparamters for XGBoost

The development of our XGBoost models was done in Python and using the XGBoost library. This library is quite complete and allows for multiple levels of analysis and validation while building confidence in the model being developed. Establishing a first XGBoost model proved to be simple and there are many tutorials online that help a less experienced user to accomplish that task.

The following task of gaining confidence and improving the model takes some extra development work and testing. Given that XGBoost is very fast, testing becomes a streamlined process and many combinations of features and hyperparameters can be made.

The most relevant hyperparameters were found to be the number of decision trees and the size of the trees. The number of trees is controlled by the **n_estimators** hyperparameter and the tests conducted in this project showed there is a big negative impact if this parameter is not large enough. The advantage of keeping the number of trees small is that the resulting model can be more easily audited by a human as decision trees are not black boxes. However, when the team attempted to keep the number of trees contained performance degraded and the benefits of the model being fully tractable do not overweight the loss of accuracy. It was also observed that for optimising this parameter there is a loss of speed as it can vary from a few trees to thousands, increasing exponentially the number of hyperparameter combinations. Therefore, there is a firm recommendation to use 1000 decision trees for all models developed in the scope of this project.

As to the size of each tree, this is controlled by the **max_depth** hyperparameter. This parameter provides good performance improvements and it is recommended that it is optimised between 1 and 20 layers per tree.

Other relevant parameters that were being optimised, but where performance improvements were not so important were:

- **min_child_weight** – When the tree partitions, if a leaf node results in a weight less than min_child_weight, then the building process partitions no more. The recommended range is 1-30.

- **subsample** – subsample ratio is used to avoid overfitting. It defines the proportion of random sampling of the training data before growing a tree. Normal range is from 0-1, but to really avoid overfitting 0.7-1 was selected in the project.
- **gamma** – Minimum loss reduction required to make a further partition on a leaf node of the tree. Possible range from 0-∞, but in the course of the project the range 0.1-10 was used.
- **colsample_bytree** – subsample ratio of columns when constructing a tree. Used range was 0.7-1.
- **reg_lambda** – L2 regularisation term on weights. Defaults to 1 but can be changed, so range was set to 0-2.

3.2.6. Hyperparamters for LSTM

LSTM was implemented in Python via the Keras package library. It allows the abstract model present in Section **Error! Reference source not found.** to be implemented in software. The onstruction of the LSTM model consists of hyperparameters crucial to its ability to learn and predict outputs from a series of inputs. The most relevant hyperparameters for LSTM to improve upon the LSTM forecasting are:

Neurons (Hidden Layer)- number of neurons that hold the relationships between the data, input as a range typically 0-100.

Number of Hidden Layers – number of hidden layers to provide greater depth to relationships, typically one or two.

Activation- behavioural functions associated with weights and bias matrices, selected from a list of functions, such as the sigmoid function that creates a weight between 0 and 1.

Optimiser- optimises the weights and bias matrices to improved performance, there are many choices[15].

3.2.7. Hyperparameters Optimisation

Artificial intelligence based techniques, LSTM and XGBoost, require the input of user-defined hyperparameters. These hyperparameters may vary with the case being analysed and so a good set of hyperparameters can improve the results significantly. While the training of the model is required for every new forecast, the tuning of hyperparameters does not need to be as frequent.

The selection process is not trivial as there is a significant number of hyperparameters. Commonly applied methods are random search, matrix search or other heuristic based methods. These methods are either time consuming or have little guarantees of being near optimal options.

To tackle this problem, Bayesian optimisation can be applied. Bayesian optimisation is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric. The function being evaluated can be of any level of complexity.

[15] https://keras.io/optimizers/

Optimisation is finding the input values to an objective function that yield the lowest output value. In machine learning, the objective function is multi-dimensional because it takes in a set of model hyperparameters. For simpler functions, the minimum loss can be found by trying different input values and verifying which set of values yields the lowest objective function result. This works reasonably well, while evaluations of the objective function are computationally cheap. For complex objective functions the number of evaluations should be reduced to the bare minimum.

The case of LSTM is particularly benefitting of Bayesian optimisation, but XGBoost also sees large improvement in the tuning of hyperparameters.

In the project, a library available for Python (but also other platforms), HyperOpt, was used. Hyperparameter optimisation was achieved using Bayesian optimisation with a tree-structure Parzen estimator (TPE) search space approach.

The tree-structured Parzen estimator is a sequential model-based optimisation method that sequentially constructs models to approximate the performance of hyperparameters based on historical measurements, and then subsequently chooses new hyperparameters to test based on this model. The TPE approach models P(x|y) and P(y), where x represents hyperparameters and y the associated quality score. P(x|y) is modelled by transforming the generative process of hyperparameters, replacing the distributions of the configuration prior with non-parametric densities.

HyperOpt requires four major input methods to be defined and run:

- **Objective function** – the objective function method defines the fit function and the metric to monitor in the optimisation process.
- **Search space method** – this method defines the search space, including the hyperparameters to be optimised and their desired ranges.
- **Trials methods** – optional method that initiates the structures for advanced analysis of results and auditing of optimisation process. This is not required in deployment mode, but very useful in the first steps of tuning with any new dataset.
- **Optimisation algorithm** – defines the methods to be used in the optimisation process.

With these four inputs, HyperOpt conducts the optimisation process in an automated manner and the final results should be used as hyperparameters for that dataset.

The tuning process does not need to be run as frequently as the training process, due to hyperparameters adequacy to the dataset and not to the particular moment that is being forecasted, or the particular parameter being forecasted.

When optimising hyperparameters, if possible the tuning set should be different from the training set to avoid overfitting. In this case, given the limited length of the datasets, the tuning set was defined as a subset of the training set. Even though this was done there were no signs of overfitting.

The recommendation is that when deployment comes the tuning set is chosen as a separate time period from the training set.

In terms of recurrence, the tuning process does not require as frequent repetition as the training process. Training needs to be conducted every time there are data updates and a forecast for a subsequent period is sought. Tuning should be cyclically repeated but maybe once every month or quarter. Further research would be required to identify the best cycle for tuning. As long as the data trends remain similar the tuning process should be little more than a validation of the previous solution.

3.3. Investigation on Features

Having discussed the forecasting models and the process to optimise the hyperparameters there is only one additional key element required to build quality forecasts, the definition of a good feature set.

A feature is a known variable that is used to inform the forecast of a variable for which only historical information is known. In case the feature is not known, forecasts of the feature can be used as a proxy, naturally taking a toll in the final accuracy of the forecast. Certain variables such as air temperature are widely forecasted with very high levels of accuracy, particularly in the shorter-term horizons. These variables are very useful as features in the prediction of other variables that have some sort of dependency on them. Electrical load and air temperature are commonly correlated, especially when electric heating and cooling systems exist.

More generally, weather related data can improve the quality of a forecast in the electric power systems domain. Tests were conducted using more variables in addition to the air temperature, such as air pressure. Results were largely improved when multiple features were applied, but in practical terms, it does not seem reasonable to expect all of those features to be available and so that is not a recommendation of the project.

Another technique was successfully tested which uses data manipulation to facilitate the training process of the forecasting methods. One hot encoding is a technique that models qualitative variables as binary variables. One very important example of application is the days of the week. Instead of labelling the day of the week by a number from 1 to 7, seven variables are created (Sunday through to Saturday) and for each, a 0 or a 1 is assigned. The sum of these variables per data point must always be 1, as a certain day cannot be, e.g., both Monday and Tuesday at the same time. Holidays are also modelled with binary variables.

As final recommendations for feature selection, depending on the type of variable being forecasted some features might or might not be relevant. A good breakdown is as follows:

- **Load profiles** – the most relevant features for these profiles are day of the week, day of the year, season, hour of the day, bank holidays and in cases where a meaningful correlation exists with temperature, the air temperature (or other available weather data).
- **Renewable Generation profiles** – in this case, day of the week or bank holidays do not have an influence, but all other features may have as well as wind speed or solar irradiance.
- **Substation net flow profiles** – as a combination of the above, all of the load indicated features should be relevant, as well as the dominant generation features for each specific case.

- **Domain Knowledge**: Where domain knowledge can be applied to identify patterns not identified in the aforementioned, features can be appended.

Finally, when active and reactive power is being forecasted with XGBoost, a process of first forecasting the active power component and next using that forecast as a feature for the reactive power component was developed. This should be the order as the dominant variable is active power and not reactive power. In the case of LSTM, the cross impacts of the two variables can be withdrawn by the methods and therefore they can be forecasted at the same time, in which case this process does not apply.

4. Systemisation of Procurement & Development of Use Cases

The DSOs aim is to procure, arm and dispatch services from distributed customer assets to ensure continued operational security and stability of the network. To ensure this the DSO must procure, arm and dispatch services in a timely manner which will involve modelling the network across multiple time horizons and voltage levels. The timings of the analysis are likely to be driven by the gate closure timings for various business processes e.g. the cut-off time to submit information to a certain market, accept bids, provide arming notifications, provide dispatch notifications, etc.

To determine the requirements for flexibility services, credible outage conditions are assessed for a part of the network at a particular point in time. Power flow analysis is used to identify issues such as thermal overloads or voltages being out of the permissible range. Power flow analysis requires a model of the network that shows how the various transformers, switchgear, and cables are connected and also provides information on the impedances and ratings of these network components. To model the flow of power over the network, forecasted values for load and generation at all the relevant nodes within the network model must be provided.

The power flow analysis may require some adjustments to be made. For example where the networks for each voltage level are modelled separately, then the impact of adjustments to the load and generation at one voltage level, for example, to account for the use of flexibility services, may need to be reflected at other voltage levels. Other interactions between voltage levels, including exchanging and blending forecasts with National Grid, should be considered.

Where the contingency being modelled would result in generators being tripped off the network after a fault, then there is a need to use forecasts for the network loads that would occur without the contribution from embedded generation i.e. the Total Load rather than the Net Load.

The SCADA systems will normally record the net load on the healthy network rather than the total load on a post-fault network, therefore, the total load forecast must be generated by creating a net load forecast and then adjusting this using a forecast of the output of the embedded generation. The embedded generation that needs to be considered may also need to be aggregated across multiple voltage levels as the load at a 33kV Primary Transformer, say, will be reduced by embedded generation connected at 11kV or LV.

Examples of the procurement task classes, the time horizon, modelling and input data required are shown in Figure 3 below, mapped by voltage level and time frame. Each task is broken down into three requirements that need to be satisified to achieved procurement via forecast:

- The problem to be solved via procurement
- Network Model Representation
- Input Data for the model

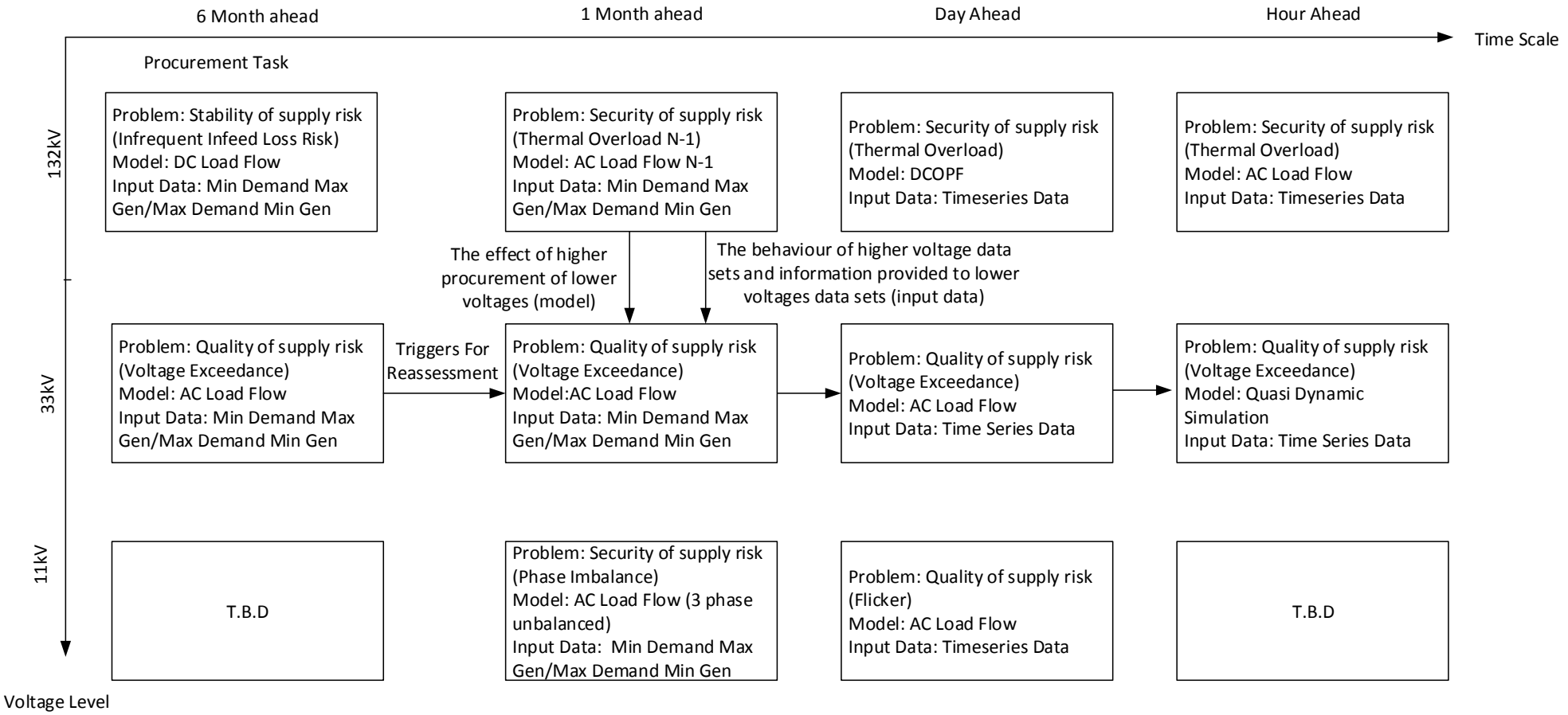


Figure 3: Systemisation of Procurement - Exampl

The figure presented is by no means exhaustive of the types of problems to be solved via procurement tasks, this will be developed during other parts of this project. Similarly, the physical modelling of the network to highlight the problems to be procured for will also require further investigation. What this document seeks to fufill is the task of forecasting for input data that will be associated with each procurement problem. To successfully achieve this, use case and test scenarios are developed to determine where forecasting for input data can be useful and where it is limited.

4.1. Use Cases and Test Scenarios

In order to develop forecasting models in a systematic way, it was necessary to establish a benchmark, therefore a sufficient number of Use Cases (UC) were developed. These use cases reflect the different needs for forecasting data and therefore consider different voltage levels and time frames as changing variables across the use cases.

The four use cases were:

1. **UC1** – 6 months ahead, GSP study – forecasts for the subsequent 6 months will be provided in 30min time steps.
2. **UC2** – 1 month ahead, BSP study – forecasts for the following month will be provided in 30min time steps.
3. **UC3** – Day ahead, primary study – forecasts for the next 24h in 30min time steps.
4. **UC4** – Hour ahead, BSP study – forecasts for the next 2 half hours.

Two forecast tests were proposed per use case and consist of different training sets for the same test set. Given that there are many different variables that can be changed to try to influence the quality of the results, the definition of the training periods for each of the use cases was decided in a way that allows drawing conclusions about the influence of the training set length on the results in each of the cases.

The tuning and validation datasets were used first in the hyperparameter optimisation, then the training and test sets are used for the prediction.

For each of the use cases, input data was provided in excess of the minimum set of data needed to forecast. When applying the different forecasting methods, the influence of using (or not) these extra time series was be analysed.

Use cases 1-4 have been used throughout the development of the forecasting methods, since these contained load and generators of differing import and export behaviours. For more single type testing, use cases 5-8 were developed and included. The results from these tests are outlined in Section 7. For the single type testing there was a selection of generator types to choose from:

- Solar;
- Wind;
- CHP;
- Biomass;
- Anaerobic Digestors;
- STOR; and

- Battery.

Each type was graphed and can be found in section 9: Appendix A. From the behaviour presented in the graphs and domain knowledge the generators were grouped by whether forecasting would provide a benefit. It was determined that where weather plays a key role for generation it would need to be forecasted due to the physical data that call be readily called upon to enable prediction.

Where a generators export was dictated indirectly from weather, fuel availability, controllable dispatch or market forces, they would not be considered at this stage. This resulted in solar and wind forecast models to be developed whereas no model was created for CHP, biomass, anaerobic digesters, batteries, and STOR. However, the techniques and tool described in this report could be used to look for patterns and correlations between data and the profiles of these generators; for example, the correlation between STOR running and predicted solar and wind output reduction.

4.1.1.UC1 – GSP, Six Months Ahead

Type	**GSP**
Name	Indian Queens
Site location	SW 93918 59012 (50° 23’ 41.20” N 004° 54’ 03.09” W)
Input data per transformer	Active power (MW) Reactive power (MVAR)
Full dataset filename	Indian_Queens_GSP_Full.csv
Dataset period	14/12/2014 – 15/02/2018
Forecast test 1	
Tuning set (where applicable)	14/12/2014 – 12/11/2016
Validation set (where applicable)	13/11/2016 – 13/12/2016
Training set	14/12/2014 – 13/12/2016
Test set	14/12/2016 – 13/05/2017
Forecast test 2	
Tuning set (where applicable)	14/12/2015 – 12/11/2016
Validation set (where applicable)	13/11/2016 – 13/12/2016
Training set	14/12/2015 – 13/12/2016
Test set	14/12/2016 – 13/05/2017

4.1.2.UC2 – BSP, Month Ahead

Type	**BSP**
Name	Cardiff South Grid
Site location	ST 19840 74680 (51° 27’ 55.41” N 003° 09’ 19.19” W)
Input data (For each transformer)	Active power (MW) Reactive power (MVAR)
Full dataset filename	Cardiff_South_Grid_BSP_Full.csv
Dataset period	01/01/2014 – 15/02/2018
Forecast test 1	
Tuning set (where applicable)	01/06/2014 – 31/05/2015

Validation set (where applicable)	01/06/2015 – 30/06/2015
Training set	01/06/2014 – 30/06/2015
Test set	01/07/2015 – 31/07/2015
Forecast test 2	
Tuning set (where applicable)	01/01/2015 – 31/05/2015
Validation set (where applicable)	01/06/2015 – 30/06/2015
Training set	01/01/2015 – 30/06/2015
Test set	01/07/2015 – 31/07/2015

4.1.3.UC3 – Primary, Day Ahead

Type	**Primary**
Name	Prince Rock
Site location	SX 49800 54100 (50° 22′ 03.29″ N 004° 06′ 47.98″ W)
Input data (for each transformer)	Active power (MW) Reactive power (MVAR)
Weather source	Metoffice – 9001, Mount Batten
Full dataset filename	Prince_Rock_full.csv
Dataset period	01/01/2014 – 15/02/2018
Forecast test 1	
Tuning set (where applicable)	01/06/2014 – 23/06/2015
Validation set (where applicable)	24/06/2015 – 30/06/2015
Training set	01/06/2014 – 30/06/2015
Test set	01/07/2015
Forecast test 2	
Tuning set (where applicable)	01/04/2015 – 23/06/2015
Validation set (where applicable)	24/06/2015 – 30/06/2015
Training set	01/04/2015 – 30/06/2015
Test set	01/07/2015

4.1.4.UC4 – BSP, Hour Ahead

Type	**BSP**
Name	Truro
Site location	SW 80210 46794 (50° 16′ 48.33″ N 005° 05′ 10.72″ W)
Input data (for each transformer)	Active power (MW) Reactive power (MVAR)
Weather source	Metoffice – 200324, Hendra, Truro
Full dataset filename	Truro_BSP_Full.csv
Dataset period	01/01/2014 – 15/02/2018
Forecast test 1	
Tuning set (where applicable)	01/04/2015 – 23/06/2015
Validation set (where applicable)	24/06/2015 – 30/06/2015
Training set	01/04/2015 – 30/06/2015
Test set	01/07/2015
Forecast test 2	
Tuning set (where applicable)	01/06/2015 – 23/06/2015

Validation set (where applicable)	24/06/2015 – 30/06/2015
Training set	01/06/2015 – 30/06/2015
Test set	01/07/2015

4.1.5.UC5 – Primary

Type	**Primary**
Name	Llynfi Valley
Site location	SS 8718 8876 (51° 35’ 11.04” N 003° 37’ 46.92” W)
Input data (for each transformer)	Active power (MW) Reactive power (MVAR)
Weather source	N/A
Full dataset filename	XGBoost_Input.csv
Dataset period	01/01/2014 - 16/02/2018
Forecast test	All time horizons

4.1.6.UC6 – Generator Customer-Wind Farm

Type	**Generator Customer**
Name	Goonhilly Wind Farm
Site location	SS 8718 8876 (51° 35’ 11.04” N 003° 37’ 46.92” W)
Input data from historian	Active power (MW) Reactive power (MVAR)
Weather source	N/A
Full dataset filename	XGBoost_Input.csv
Dataset period	01/01/2014 - 16/02/2018
Forecast test	All time horizons

4.1.7.UC7 – Generator Customer-Solar Farm

Type	**Generator Customer**
Name	AYSHFORD COURT FARM 33kV SOLAR PARK
Site location	ST 04850 15130
Input data from historian	Active power (MW) Reactive power (MVAR)
Weather source	N/A
Full dataset filename	XGBoost_Input.csv
Dataset period	01/01/2014 - 16/02/2018
Forecast test	All time horizons

4.1.8.UC8 – Large Load Customer

Type	**Primary**
Name	Load 3
Site location	SS 8718 8876 (51° 35’ 11.04” N 003° 37’ 46.92” W)
Input data	Active power (MW)

	Reactive power (MVAR)
Weather source	N/A
Full dataset filename	XGBoost_Input.csv
Dataset period	01/01/2014 - 16/02/2018
Forecast test	All time horizons

5. Development of Methods

This section shows how to set up the toolchain before using worked examples to show how the XGBoost and LSTM forecasts are configured and used. The method to set up forecasts using ARIMA has been omitted as this gives inferior results to the two machine learning methods, and requires considerable user-in-the-loop interactions for training and execution.

The toolchain is based on the Anaconda data science platform.

5.1. Toolchain Set-Up

There are a couple of conflicts and issues with the packages on the standard Anaconda install, this is due to quirks in the Tensorflow and Keras packages.

For example, the standard Anaconda 3 python executable is presently 3.7 and Tensorflow is only compatible up to 3.6.6. Therefore Miniconda is required, a product of Anaconda, which allows the user to bolt together environments from the previous versions to achieve a working environment.

To achieve a stable environment/Jupyter kernel the following steps in the following order must be undertaken.

1. Go to https://www.anaconda.com/distribution/
2. Download Anaconda for Windows 2018.12, Python 3.7 version, 64-Bit Graphical Installer
3. Install Anaconda 3 64 Bit -make sure to place it in a sensible location, Anaconda likes to install in hidden files locations.

The install should now be available in the Windows toolbar, Figure 4.

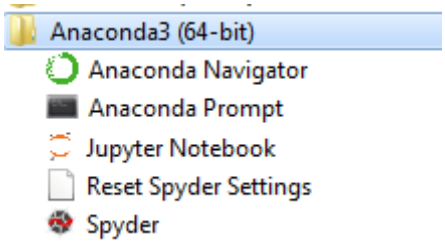


Figure 4: Anaconda install in toolbar

There is a requirement to update Anaconda with Miniconda to implement environment creation:

4. Go to https://conda.io/en/latest/miniconda.html
5. Download Miniconda 3 64 Bit
6. Install Miniconda 3 64 bit

Miniconda will supersede the Anaconda 3 prompt (figure above) so now when the Anaconda prompt is opened the following is visible, Figure 5.

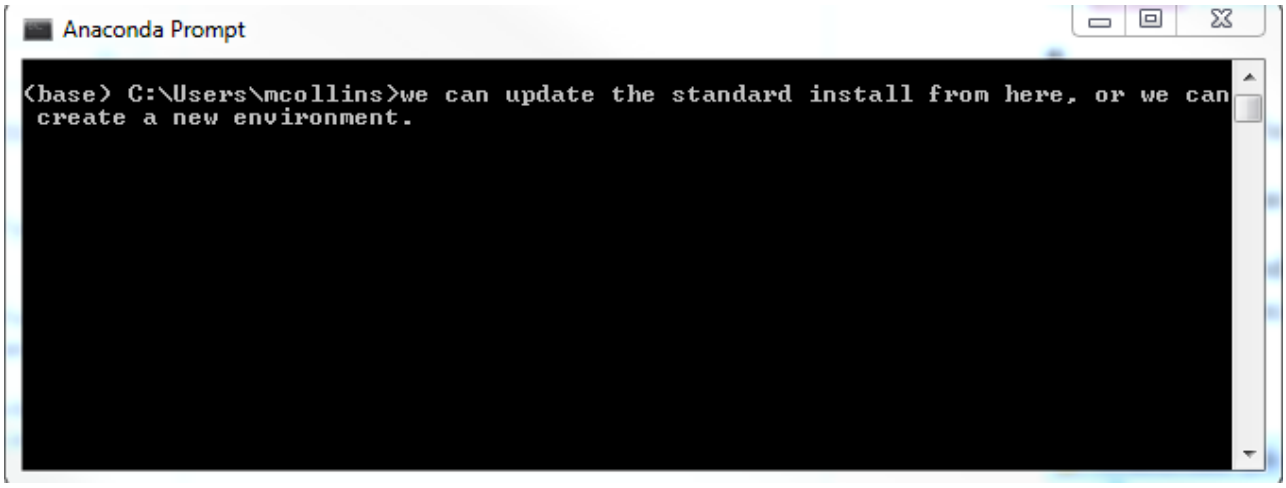


Figure 5: Anaconda Prompt

To create the majority of the environment needed copy and paste the following instruction into the prompt: *conda create --name effsEnv python=3.6.6 pandas matplotlib scipy scikit-learn jupyter sympy ipykernel pip openpyxl*

```
(effsEnv) C:\Users\mcollins>conda create --name effsEnv python=3.6.6 pandas matp
lotlib scipy scikit-learn jupyter sympy ipykernel pip openpyxl
```

Figure 6: Environment instruction

Pressing enter will create the environment; it is important to include ipykernal, to make the Python environment available as a Jupyter Notebook.

After the environment has been created, install the Statsmodels package independently.

```
(effsEnv) C:\Users\mcollins>conda install -c anaconda statsmodels
```

Figure 7: Statsmodels install

For hyperparameter exploration, the HyperOpt package is required.

```
(effsEnv) C:\Users\mcollins>conda install -c conda-forge hyperopt
```

Figure 8: HyperOpt install

Then the Tensorflow package must be installed. Problems with the Conda installs have been experienced and this approach fixed the problem, Figure 9.

```
(base) C:\Users\mcollins>activate effsEnv
(effsEnv) C:\Users\mcollins>pip install tensorflow
```

Figure 9: Tensorflow install

Finally, pip install keras in the same way.

```
(effsEnv) C:\Users\mcollins>pip install keras
```

Figure 10: Keras install

Again, for hyperparameter exploration for LSTM, the Hyperas version 0.4 must be installed; a pip install is recommended.

For XGBoost the following is required:

```
conda install -c anaconda py-xgboost
```

and

```
conda install -c anaconda seaborn
```

Figure 11: XGBoost packages

The environment and kernels are now ready for use. The Jupyter Notebook is now available for the desired kernel.

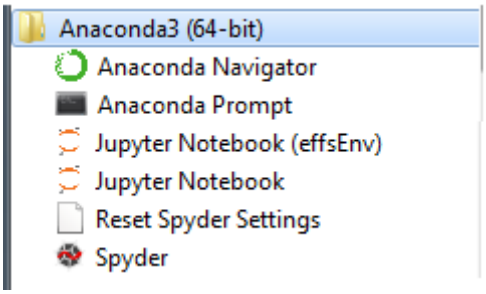


Figure 12: Jupyter Notebook

The methods can now be run. The high-level flow charts of the methods are provided in Section 10.

5.2. XGBoost Notebook Development

The structure of the developed notebooks follows the same pattern: importing packages, input data, actions, outputs. The process of note book construction is presented in section 10: Appendix B.

5.2.1. Hyperparameter Optimisation

Before running the prediction model, the hyperparameter optimisation is run. The hyperparameter optimisation notebook follows the same pattern as above. The packages required at this stage are shown in Figure 13.

```
import pandas as pd, numpy as np
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from math import sqrt
import time
import matplotlib.pyplot as plt
```

Figure 13: Hyperparameter optimisation import

The notebook then needs to pointed to an input file (csv) containing the data for tuning and validating the model, Figure 14. The filepath should be updated to the location of that file. The split data here refers to where the training set ends and the validation set begins. The features are then listed. If additional data is to be used in the tuning of the model, for

example solar irradiance, it should be added to the input file, and then the feature added to this below. The feature name should match the column heading in the input file. The parameter that the forecasting and hyperparameter optimisation is being run for also need to be specified. This is done in the bottom two lines in Figure 14; in this example, the parameter is “Load”.

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%d/%m/%Y %H:%M')
df = pd.read_csv('C:\\filepath\\input_data.csv',index_col=0,date_parser=dateparse)
#df.index=pd.to_datetime(df['Date'])
print(df.head(10))
print(df.dtypes)

split_date = '2015-06-23'
train = df.loc[df.index <= split_date].copy()
test = df.loc[df.index > split_date].copy()

def create_features(df, label=None):
    """
    Creates time series features from datetime index
    """
    df['date'] = df.index
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear
    #df['Temp'] = df['Temp']

    X = df[['hour','dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear']]
    if label:
        y = df[label]
        return X, y
    return X

train, y_train = create_features(train, label='Load')
valid, y_valid = create_features(test, label='Load')
```

Figure 14: Hyperparameter optimisation input and features

The objective function and search space need to be defined. These are shown in Figure 15 and described in Section **Error! Reference source not found.**. Then follows the trials method nd optimisation algorithm, Figure 16.

These then output an optimal set of hyperparameters for use in XGBoost. This part of the process is manual and these values are to be copied into the XGBoost notebook. Figure 17 shows an example of the hyperparameters that can be generated.

```
def rmse(true,pred):
    return sqrt(mean_squared_error(y_true=true,
                y_pred=pred))
```

```
def objective(space):

    clf = xgb.XGBRegressor(n_estimators = 1000,
                           max_depth = space['max_depth'],
                           min_child_weight = space['min_child_weight'],
                           subsample = space['subsample'],
                           gamma = space['gamma'],
                           colsample_bytree = space['colsample_bytree'],
                           reg_lambda = space['reg_lambda']
                          )

    eval_set  = [( train, y_train), ( valid, y_valid)]

    clf.fit(train, y_train,
            eval_set=eval_set, eval_metric="rmse",
            early_stopping_rounds=30,
            verbose=False)

    pred = clf.predict(valid)
    score=rmse(y_valid,pred)
    print("SCORE:", score)

    return{'loss':score, 'status': STATUS_OK }
```

```
space ={


        'max_depth':  hp.choice('max_depth', np.arange(1, 20, dtype=int)),
        'min_child_weight': hp.quniform ('x_min_child', 1, 30, 1),
        'subsample': hp.uniform ('x_subsample', 0.7, 1),
        'gamma' : hp.uniform ('x_gamma', 0.1,1.0),
        'colsample_bytree' : hp.uniform ('x_colsample_bytree', 0.7,1),
        'reg_lambda' : hp.uniform ('x_reg_lambda', 0,2),

        'nthread': 4,
#         'objective': 'binary:logistic',
        'booster': 'gbtree',
        'seed': time.time()
    }
```

Figure 15: Objective Function and Search Space

```
trials = Trials()
```

```
start_time = time.time()

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=1000,
            trials=trials)

print(best)

print("--- %s seconds ---" % (time.time() - start_time))
```

Figure 16: Trials Method and Optimisation Algorithm

```
{'max_depth': 11, 'x_colsample_bytree': 0.7233944410783489, 'x_gamma': 0.4422939713231595, 'x_min_child': 23.0,
'x_reg_lambda': 1.62928317876416, 'x_subsample': 0.8292149095594812}
```

Figure 17: Example hyperparameter output

5.2.2.XGBoost Notebook

As with the hyperparameter optimisation, the XGBoost notebook begins with importing the necessary packages, Figure 18, and importing the input data, Figure 19.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error
plt.style.use('fivethirtyeight')
import time
```

Figure 18: XGBoost package import

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%d/%m/%Y %H:%M')
df = pd.read_csv('C:\\filepath\\input_data.csv',index_col=0,date_parser=dateparse)
#df.index=pd.to_datetime(df['Date'])
print(df.head(10))
print(df.dtypes)
```

Figure 19: XGBoost input filepath

The date at which input data is split needs to be specified. This is by the Split Date which determines when the training data is separated from the test data. The End Date then specifies when the prediction ends. In this example, the prediction is for one day, 1st July 2015, Figure 20.

```
split_date = '2015-07-01'
end_date = '2015-07-02'
train = df.loc[df.index <= split_date].copy()
test = df.loc[df.index > split_date].copy()
```

Figure 20: Splitting the input data into training and test sets

As with the hyperparameter optimisation notebook, the features applied in the forecast need to be defined, Figure 21. These should match the column headings in the input file. In this example bank holiday in England and Wales are included, and one hot encoding on the day of the week.

```
def create_features(df, label=None):
    """
    Creates time series features from datetime index
    """
    df['date'] = df.index
    df['hour'] = df['date'].dt.hour
    df['dayofweek'] = df['date'].dt.dayofweek
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear
    df['Temperature'] = df['Temperature']
    #df['Wind_Power'] = df['Wind Power']
    df['Monday'] = df['Monday']
    df['Tuesday'] = df['Tuesday']
    df['Wednesday'] = df['Wednesday']
    df['Thursday'] = df['Thursday']
    df['Friday'] = df['Friday']
    df['Saturday'] = df['Saturday']
    df['Sunday'] = df['Sunday']
    df['Holidays'] = df['Holiday']

    X = df[['hour','dayofweek','month','year','quarter',
          'dayofyear','dayofmonth','weekofyear','Temperature',
          'Monday','Tuesday','Wednesday',
          'Thursday','Friday','Saturday','Sunday','Holiday']]
    if label:
        y = df[label]
        return X, y
    return X
```

Figure 21: Feature definition

XGBoost is then trained. The parameter the forecast is for is specified here, Figure 22. The training function is then updated with the optimal parameters from the hyperparameter optimisation, Figure 23.

```
X_train, y_train = create_features(train, label='Load')
X_test, y_test = create_features(test, label='Load')
```

Figure 22: Setting up training

```
start = time.time()

#tuning parameters from hyperopt
reg = xgb.XGBRegressor(n_estimators=1000,n_jobs=3, max_depth=9, x_colsample_bytree = 0.864149076142802,
      x_gamma = 0.1822602040528707, x_min_child=19.0, x_reg_lambda = 1.423996374313407,
      x_subsample = 0.8797950284970765)

reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        early_stopping_rounds=50,
       verbose=True) # Change verbose to True if you want to see it train

elapsed = time.time()
elapsed = elapsed - start
print("Time spent in (function name) is: ", elapsed)
```

Figure 23: Training function

The prediction can then be run.

```
start = time.time()

test['Prediction'] = reg.predict(X_test)
fullset = pd.concat([test, train], sort=False)

elapsed = time.time()
elapsed = elapsed - start
print("Time spent in (function name) is: ", elapsed)
```

Figure 24: Prediction function

There are then a number of options for what can be outputted. The prediction overlaid on the entire dataset is an option (Figure 25), as well as the prediction and actual data for the specified time horizon (Figure 26).

```
_ = fullset[['Load','Prediction']].plot(figsize=(15, 5))
```

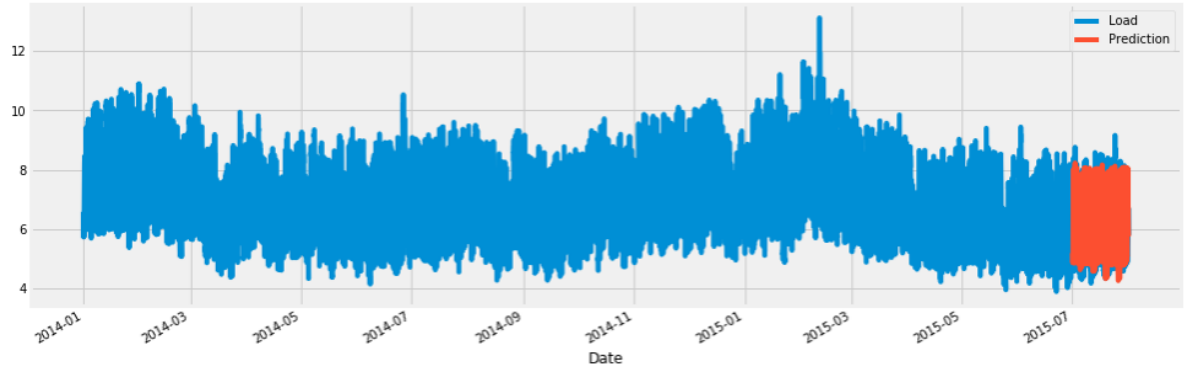


Figure 25: MW Prediction overlaid on dataset

```
# Plot the forecast with the actuals
f, ax = plt.subplots(1)
f.set_figheight(5)
f.set_figwidth(15)
_ = fullset[['Prediction','Load']].plot(ax=ax,
                                        style=['-','.'])
ax.set_xbound(lower='2015-07-01', upper='2015-07-02')
ax.set_ylim(0, 20)
plot = plt.suptitle('Forecast vs Actuals')
```

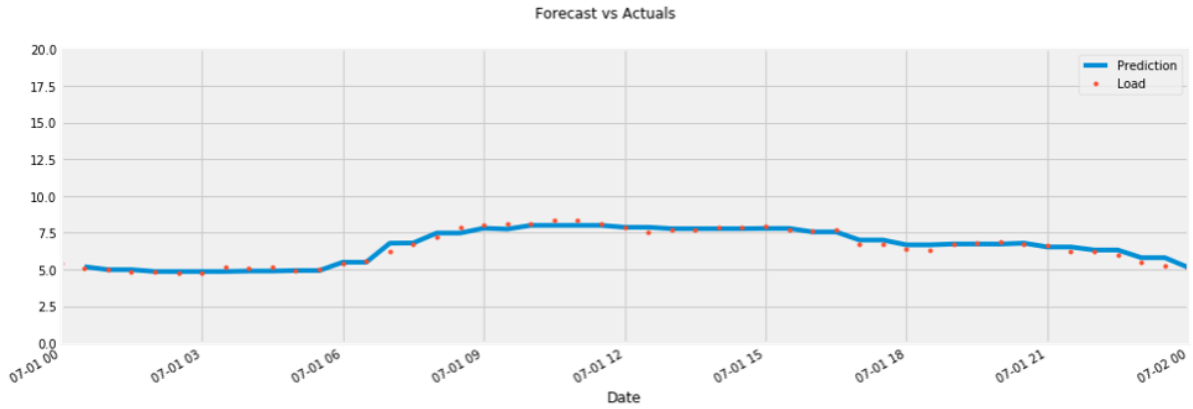


Figure 26: MW Prediction and actual values for specified time horizon

The Mean Squared Error, Mean Absolute Error, Root Mean Squared Error, and Mean Absolute Percentage Error can either be calculated within the notebook, or the calculations can be applied manually to the predicted data. The syntax for including these calculations in the notebook is shown below, Figure 27.

```
mean_squared_error(y_true=test['Load'],
                   y_pred=test['Prediction'])
```

```
mean_absolute_error(y_true=test['Load'],
                    y_pred=test['Prediction'])
```

```
from math import sqrt
rmse=sqrt(mean_squared_error(y_true=test['Load'],
                     y_pred=test['Prediction']))
print(rmse)
```

```
def mean_absolute_percentage_error(y_true, y_pred):
    """Calculates MAPE given y_true and y_pred"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
mean_absolute_percentage_error(y_true=test['Load'],
                   y_pred=test['Prediction'])
```

Figure 27: Error calculations

5.2.3.Influence of Features

Features and their importance have already been introduced in Section 3.3, and here results from initial Use Case testing are provided to show the importance of different features. The results shown are for UC2, Cardiff South BSP for a one month ahead prediction. The Use Case is set up as defined for Forecast Period 1 in Section 4 and three sets of features are run.

Each builds on the last by adding new features to the list, rather than by replacing previously tested features.

- Baseline: hour, day of week, quarter, month, year, day of year, day of month, week of year;
- Round 2: temperature, one hot encoding (Section 3.3) and holidays;
- Round 3: wind and solar output collected from weather sources and wind and solar physical models[16].

The importance of the features on the prediction for these three rounds of testing are shown in Figure 28. With the basic feature set in the Baseline test, the day of year is most important, followed by the hour. With the introduction of temperature, holidays and one hot encoding in Round 2, the day of the year and the hour remain the most important features, followed by temperature. With the introduction of wind and solar output in Round 3, the importance of the features changes, with temperature becoming the most important feature in the prediction. This change in importance reflects the methods ability to detect trends in the input data and predict based on those trends.

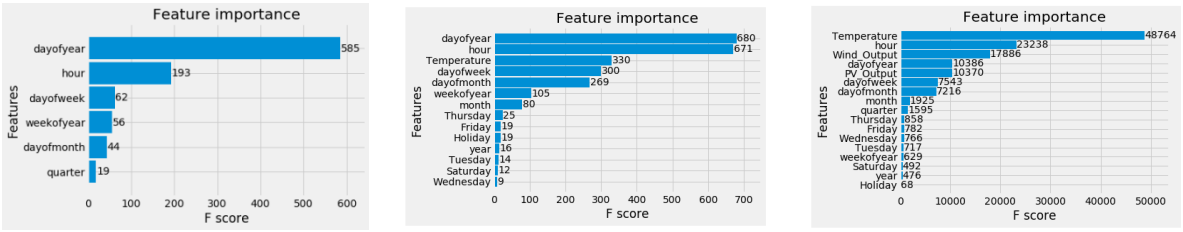


Figure 28: Left – baseline, Centre – Round 2, Right – Round 3

The prediction itself becomes much more in line with the actual values for the test period, Figure 29.

[16] https://www.renewables.ninja/

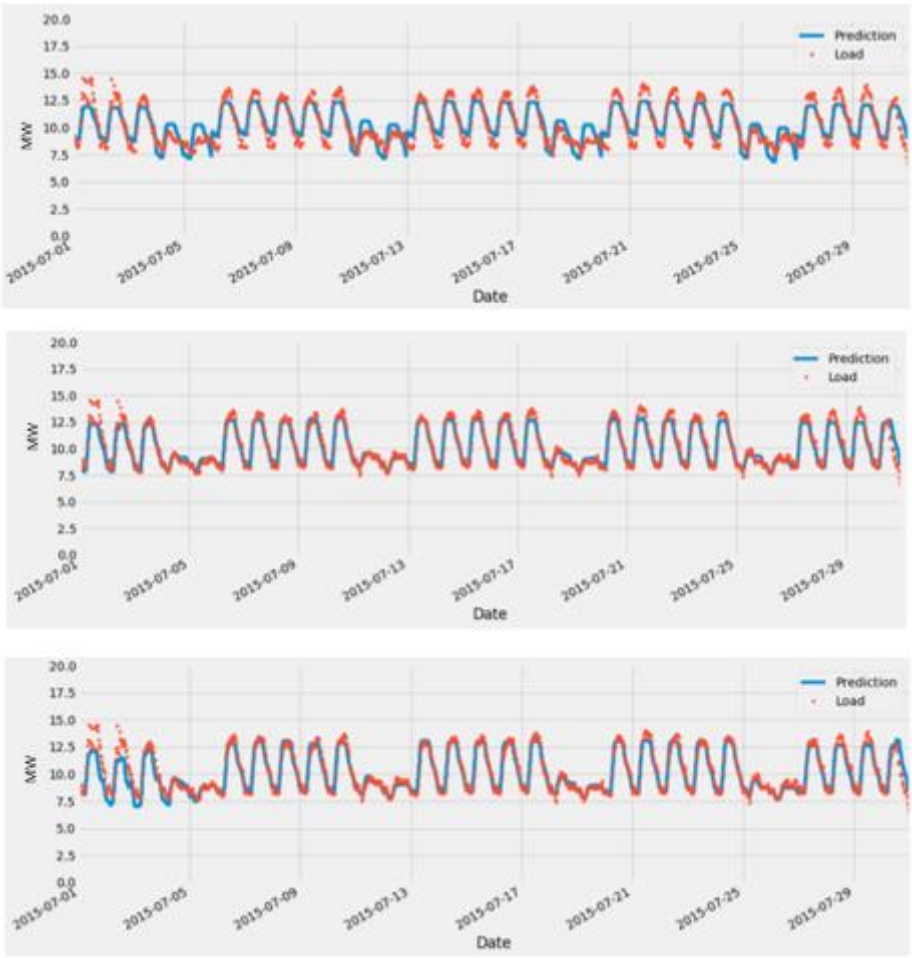


Figure 29: Actual MW versus Predicted MW. Top – Baseline, Middle – Round 2, Bottom – Round 3

Finally, there is an impact on the metrics. The Root Mean Squared Error (RMSE) values for each of the three tests are shown in Table 1. The RMSE is the standard deviation of the prediction errors. The equation used is:

$$RMSE = \sqrt{\overline{(f - o)^2}}$$

Where *f* represents the forecasted value and *o* represents the actual value.

Table 1: Feature impact on RMSE

Test	RMSE	MAPE (%)
Baseline	0.902	7.152
Round 2	0.569	3.517
Round 3	0.647	4.466

The impact on the RMSE by introducing temperature, one hot encoding and holidays shows a reduction of 37%. Interestingly, by providing more information in Round 3 the RMSE has increased by 14%. This shows that while introducing more data can be beneficial and

improve accuracy, it can also have a detrimental effect. In this instance, it could be that there is limited wind and solar generator connected behind the BSP meaning that trends found are not relevant for the prediction in the load. The temperature remains an important feature, however, as changes in temperature impact on load.

5.2.4. Influence of Training Dataset Length

The length of the training set can also impact the accuracy of the prediction. The Use Cases were designed with two forecasting sets, one with a longer training period, and one with a shorter training period. The point of which was to determine the impact the differing datasets had on a prediction for the same Use Case. Considering UC2, as in the previous section, the forecasts are defined in the Use Case description in Section 4. Forecast 1 uses 12 months of historical data in the training set, whereas Forecast 2 uses six months of historical data. The ranking of features does not change significantly for the different training sets, Figure 30.

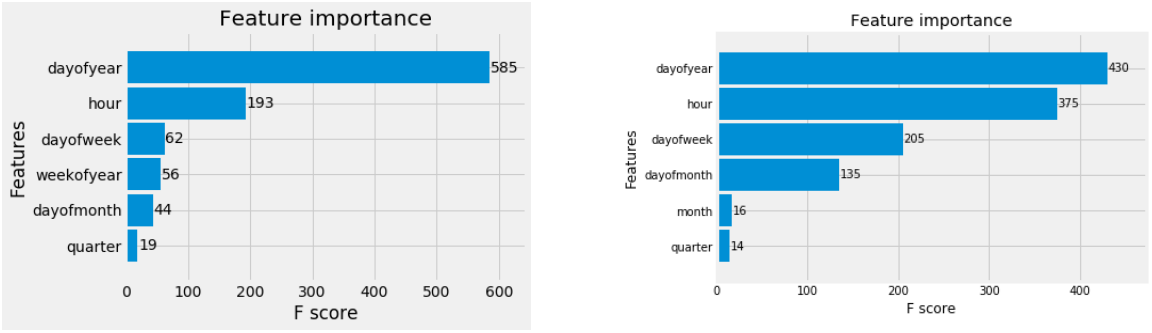


Figure 30: Feature Importance. Left – Forecast 1, Right – Forecast 2

The prediction improved with the shorter training set in Forecast 2, Figure 31. This is echoed in the RMSE values for the two forecasts, Table 2, where a reduction of 35% is seen in the RMSE for Forecast 2.

Table 2: Training set length impact on RMSE (Month ahead forecast)

Test	Training set length	RMSE
Forecast 1	12 months	0.902
Forecast 2	6 months	0.581

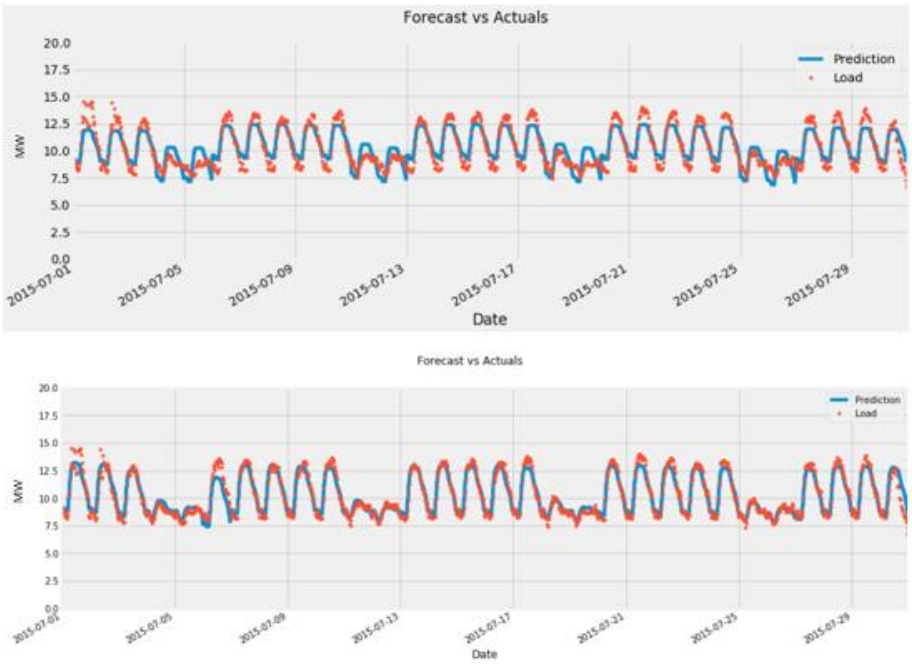


Figure 31: Forecast MW versus actual MW data. Top – Forecast 1. Bottom – Forecast 2.

While the conclusion can be drawn that the prediction may be better for shorter time horizons when using a shorter training set, the same cannot be said for longer time horizons. If we consider one of the transformers for Use Case 1 (discussion on analysing GSP transformers individually can be found in Section 7.11 and Section 8.4) the RMSE decreases when using the shorter training set. Given that UC1 initially investigates six months ahead, having more data available to facilitate that prediction helps to improve the accuracy. The RMSE is shown in Table 3, and the forecast versus the actual data is shown in Figure 32.

Table 3: UC1 – Impact of Training set length on RMSE (month ahead forecast)

Test	Training set length	RMSE
Forecast 1	12 months	12.50
Forecast 2	6 months	13.23

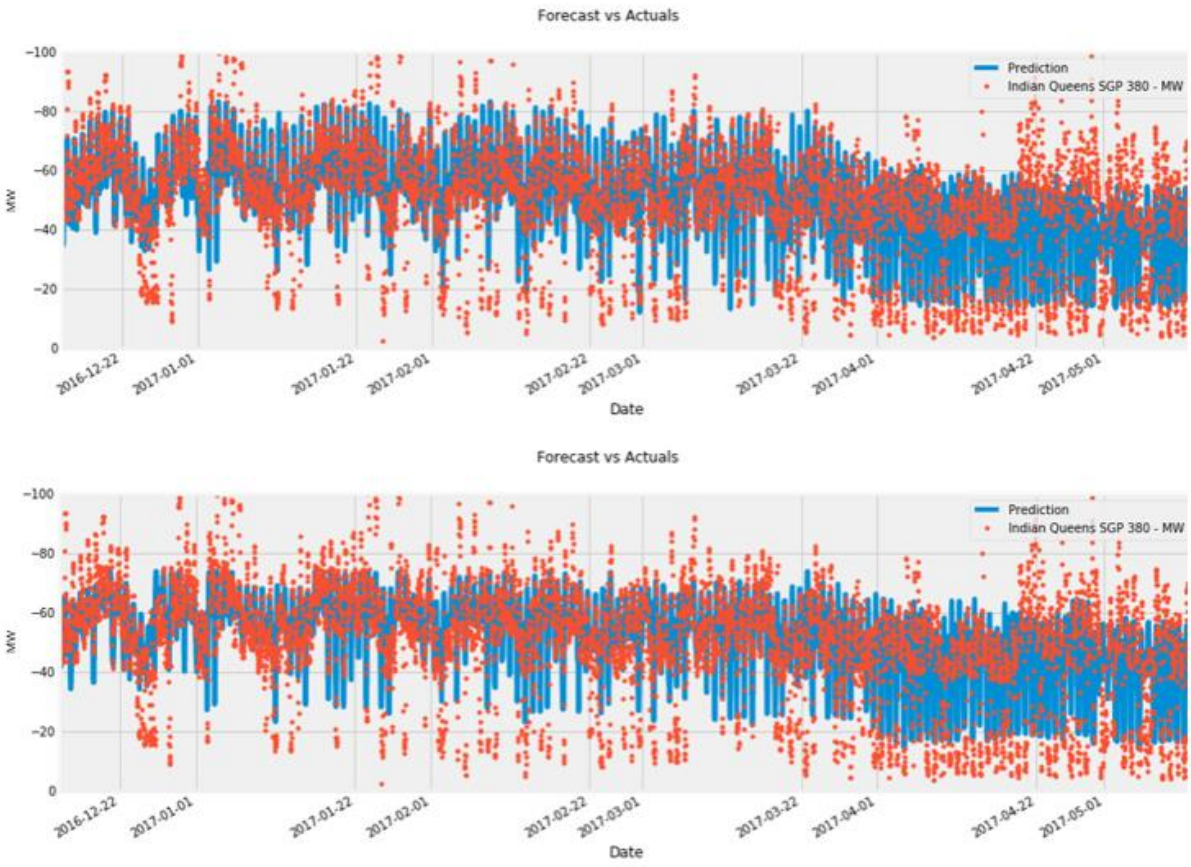


Figure 32: UC1 – Forecast MW versus actual MW data. Top – Forecast 1. Bottom – Forecast 2.

5.3. LSTM Notebook Development

5.3.1. Hyperparameter Optimisation

As with the XGBoost hyperparameter optimisation, it is first necessary to import the packages required, Figure 33.

```
from __future__ import print_function
from hyperopt import Trials, STATUS_OK, tpe, space_eval
import keras.optimizers, keras.initializers
from keras.regularizers import l2
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout, Activation
from keras.models import Sequential
from keras.utils import np_utils
from sklearn.preprocessing import MinMaxScaler
from pdb import set_trace as debug
from hyperas import optim
from hyperas.distributions import choice, uniform, conditional
import numpy, json
#import globalvars
import pandas as pd
import time
```

Figure 33: Hyperparameter optimisation package imports

It is then necessary to import the data for tuning, Figure 34. This should be in csv format. A full file path is not required, however, the name of the file is, and it should be stored in the same location as the notebook file. The parameter that the tuning is for should be specified, in this example, it is for “Load”, and this should match one of the columns in the input file.

The dates for which the split between the tuning dataset and the validation data set are also specified here.

```
def data():
        exchange_data = pd.read_csv('Prince_Rock_full_one_hot_temporal.csv')
        exchange_data["Date"] = pd.to_datetime(exchange_data["Date"], format='%d/%m/%Y %H:%M')

        # Data must be date ascending oldest to newest, may require sorting in some cases
        ind_exchange_data = exchange_data.set_index(["Date"])

        df = ind_exchange_data[["Load"]]

        # All input data training data frame
        # Known as Tuning Set in Hyper Opt
        train = df.loc['2014-06-01 00:00:00':'2015-06-24 23:30:00']
        print(train.size)

        # Known as Validation Set in Hyper Opt
        test = df.loc['2015-06-24 00:00:00':'2015-06-30 23:30:00']


        # transform train

        sc = MinMaxScaler(feature_range=(-1, 1))
        train_scaled = sc.fit_transform(train)
        x_train_s = train_scaled[:-1]
        x_train = x_train_s.reshape(x_train_s.shape[0], 1, x_train_s.shape[1])

        y_train = train_scaled[1:]
        Y_tr_o_t = y_train.reshape(y_train.shape[0], 1, y_train.shape[1])

        print("Size of true output training set", y_train.size, Y_tr_o_t.size)

        sc_test = MinMaxScaler(feature_range=(-1, 1))
        test_ouput_scaled = sc_test.fit_transform(test)
        x_test_i = test_ouput_scaled[1:]
        x_test = x_test_i.reshape(x_test_i.shape[0], 1, x_test_i.shape[1])
        y_test= test_ouput_scaled[:-1]


        return (x_train, y_train, x_test, y_test)
```

Figure 34: Data import

The hyperparameter optimisation function is then developed, shown in Figure 35. The Hyperopt function, implemented in Hyperas, allows for a Bayesian inference approach to finding the optimal features for the LSTM model. The choices presented are from experience and are by no means exhaustive, therefore expanding these selections could yield better metrics. In this case, metrics refer to the improvement of mean squared error.

```
def model(x_train, y_train, x_test, y_test):
    #Select Sequential Model
    model = Sequential()
    ## All optimisation variables are presented as dictionary of choices in the structure shown, they can be
    ## expanded for the
    ## choices highlighted, to add extra options please consult the Hyperas documentation.
    ## Choice highlighted: Number of Neurons in the hidden layer
    model.add(
        LSTM({{choice([10, 20, 50, 100])}}, input_shape=(1,x_train.shape[1]), return_sequences=False))
    model.add(Activation('relu'))
    ## Choice highlighted: Requirement for further hidden layers and subsequent activation
    if conditional({{choice(['one', 'two'])}}) == 'two':
        model.add(Dense({{choice([10, 20, 30, 100])}}))
        model.add(Activation('relu'))

    model.add(Dense(1))

    ## Choice highlighted: Optimisation bounds
    adam = keras.optimizers.Adam(lr={{choice([10 ** -6, 10 ** -5, 10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1])}},
                                 clipnorm=1.)
    rmsprop = keras.optimizers.RMSprop(lr={{choice([10 ** -6, 10 ** -5, 10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1])}},
                                       clipnorm=1.)
    sgd = keras.optimizers.SGD(lr={{choice([10 ** -6, 10 ** -5, 10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1])}},
                               clipnorm=1.)

    choiceval = {{choice(['adam', 'sgd', 'rmsprop'])}}

    ## Choice highlighted: Which optimisation technique to employ with chosen bounds

    if choiceval == 'adam':
        optim = adam
    elif choiceval == 'rmsprop':
        optim = rmsprop
    else:
        optim = sgd

    model.compile(loss='mean_squared_error', metrics=['accuracy'],
                  optimizer=optim)

    filepath = "weights_mlp_hyperas" + str(1) + ".hdf5"
    earlyStopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=20, verbose=0, mode='auto')
    checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
    callbacks_list = [earlyStopping, checkpoint]

    #Altering the epochs will improve performance but increase overhead

    hist = model.fit(x_train, y_train,batch_size=1, epochs=100, verbose=2, validation_data=(x_test, y_test),
                     callbacks=callbacks_list)

    h1 = hist.history
    acc_ = numpy.asarray(h1['acc'])
    loss_ = numpy.asarray(h1['loss'])
    val_loss_ = numpy.asarray(h1['val_loss'])
    val_acc_ = numpy.asarray(h1['val_acc'])

    acc_and_loss = numpy.column_stack((acc_, loss_, val_acc_, val_loss_))


    score, acc = model.evaluate(x_test, y_test, verbose=0)
    print('Final validation accuracy:', acc)

    return {'loss': -acc, 'status': STATUS_OK, 'model': model}
```

Figure 35: Hyperparameter optimisation function focusing on improvement of mean squared error

The main method is then developed, shown in Figure 36. This runs through the hyperparameter optimisation process to produce an output JSON file with the optimal LSTM hyperparameters. These are then used in the LSTM model notebook.

```
trials = Trials()
#data=data()

#Altering the trails will improve the optimsation process but increase the overall burden
start_time=time.time()
best_run, best_model = optim.minimize(model=model, data=data, algo=tpe.suggest, max_evals=1, trials=trials,
                                      notebook_name='HyperOpt_LSTM')
X_train, Y_train, X_test, Y_test = data()

print("X Train Size",X_train.size)
print("Y Train Size",X_train.size)
print("X test Size",X_test.size)
print("Y test Size",Y_test.size)

print(best_model.evaluate(X_test, Y_test))
print("Best performing model chosen hyper-parameters:")
print(best_run)
json.dump(best_run, open("best_run.txt", 'w'))

end_time=time.time()-start_time
print("Tuning Time",end_time, "S")
```

Figure 36: Hyperparameter optimisation main method function

The output is as shown in Figure 37. This should be interpreted as:

- Dense: This refers to the options for the Dense variable in Figure 35. The available options are 10, 20, 30, 100. The output of 2 means that 20 is the optimal choice for this variable.
- LSTM: This refers to the options for the LSTM variable in Figure 35. The available options are 10, 20, 50, 100. The output of 1 means that 10 is the optimal choice for this variable.
- Choiceval: This refers to the options for the Choiceval variable in Figure 35. The options are “adam”, “sgd”, and “rmsprop”. These are different optimisation algorithms. The output of 1 means that “adam” is the optimal choice for this variable.
- lr, lr_1, and lr_2 refer to options for the three choiceval options. By choosing “adam” for choiceval, the lr value applies. The third option for this is 10 ** -4. If the “sgd” option had been chosen then the lr_2 value applies.

These variables refer to the LSTM architecture that is described in Section **Error! Reference ource not found.**.

```
Best performing model chosen hyper-parameters:
{'Dense': 2, 'LSTM': 1, 'choiceval': 1, 'conditional': 1, 'lr': 3, 'lr_1': 0, 'lr_2': 1}
```

Figure 37: Optimal hyperparameter output

5.3.2.LSTM Notebook

As with the hyperparameter optimisation notebook, the LSTM notebook starts with the necessary package imports, Figure 38. The input data is then specified, Figure 39.

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from matplotlib import pyplot
import matplotlib.pyplot as plt
from pdb import set_trace as debug
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers, keras.initializers
import keras.backend as K
from keras.callbacks import EarlyStopping
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import Adam
from keras.models import load_model
from keras.layers import LSTM
from math import sqrt
import time
```

Figure 38: LSTM package import

```
#Data to be used
exchange_data = pd.read_csv('Prince_Rock_full_one_hot_just_mwmvar.csv')
exchange_data["Date"] = pd.to_datetime(exchange_data["Date"], format='%d/%m/%Y %H:%M')

# Data must be date ascending oldest to newest, may require sorting in some cases
ind_exchange_data = exchange_data.set_index(["Date"])

#Data to make predictions on (Regressor Variable)
regressor_df = ind_exchange_data
df = ind_exchange_data


#ind_exchange_data.head()
regressor_df.head()
```

Figure 39: LSTM input data

The training and test sets are then split by dates, and the predictors are defined, Figure 40. In this case, if the prediction is for the week ahead, then the previous seven days are identified as predictors. If the prediction was for a month ahead then the previous 31 days would be used as predictors.

The whole data train, the training dataset, and the test data set are then normalised. Figure 41 - Figure 43. The data is normalised to reduce the range that relationships are identified in.

```
#All input data training data frame
train = df.loc['2015-04-01 00:00:00':'2015-06-30 23:30:00']
print(train.size)
#Regressor variable data training data frame
r_train=regressor_df.loc['2015-04-01 00:00:00':'2015-06-30 23:30:00']
#print(r_train.size/5)
#Prediction example next 7 days hours based on last 7 days observations


#Select test outputs the actual observations
test = regressor_df.loc['2015-07-01 00:00:00':]

#Select predictors to regress model on
predictors = df.loc['2015-06-24 00:00:00':'2015-06-30 23:30:00']

r_predictors= regressor_df.loc['2015-06-24 00:00:00':'2015-06-30 23:30:00']
```

Figure 40: Splitting the dataset into training and test sets

```
# transform train

# scale train and test data to [-1, 1]
sc = MinMaxScaler(feature_range=(-1, 1))

#All Data Inputs
train_scaled = sc.fit_transform(train)

#Step training data one time step into the past
X_train = train_scaled[:-1]
X_tr_t = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])

#Single Inputs to test ouput predictions shift one time step
sc_1_o = MinMaxScaler(feature_range=(-1, 1))
r_train_scaled=sc_1_o.fit_transform(r_train)
y_train_o=r_train_scaled[1:]
Y_tr_o_t=y_train_o.reshape(y_train_o.shape[0], 1, y_train_o.shape[1])
print("Size of true output training set", y_train_o.size,Y_tr_o_t.size)
```

Figure 41: Normalising the data train

```
sc_test = MinMaxScaler(feature_range=(-1, 1))
test_ouput_scaled=sc_test.fit_transform(test)
y_test = test_ouput_scaled[1:]
y_test.size
```

Figure 42: Normalising test data

```
sc_pred = MinMaxScaler(feature_range=(-1, 1))
pred_input_scaled=sc_pred.fit_transform(predictors)
x_pred = pred_input_scaled[1:]
print(x_pred.size/5)
X_pred_t= x_pred.reshape(x_pred.shape[0], 1, x_pred.shape[1])

sc_r_pred = MinMaxScaler(feature_range=(-1, 1))
r_pred_input_scaled=sc_r_pred.fit_transform(r_predictors)
r_x_pred = r_pred_input_scaled[1:]
```

Figure 43: Normalising predictor data

The LSTM model is then developed, shown in Figure 44. Here the neurons variable relates to the LSTM value in the hyperparameter optimisation output and should be updated to match. The same for the Dense variable. This should match the Dense value in the hyperparameter optimisation. The optimisation algorithm should also be updated depending on the

hyperparameter optimisation output, with the appropriate “lr” value. These dictate how the model will train.

```
## LSTM Modelling

# Clear Keras Backend
K.clear_session()

# Design Network
model_lstm = Sequential()

# First input is number of neurons in the first hidden layer
neurons=10

# input_shape, batch input size, data you want to train on
model_lstm.add(LSTM(neurons, input_shape=(1, X_train.shape[1]), activation='relu',
                    kernel_initializer='lecun_uniform', return_sequences=False))
model_lstm.add(Dense(20))
model_lstm.add(Activation('relu'))
# Dense number is the output layer
model_lstm.add(Dense(2))
adam = keras.optimizers.Adam(lr=10 ** -4,clipnorm=1.)


#Compile Model
model_lstm.compile(loss='mean_squared_error', optimizer='adam')

#Summary
model_lstm.summary()

# Early Stopping- Stop training when a monitored quantity has stopped improving.
# Patience = how many epochs before loss is not improving
early_stop = EarlyStopping(monitor='loss', patience=5, verbose=1)

#Monitor Training Time
start_time=time.time()

#fit network
# Batch size number of predictions you with to make
history_model_lstm = model_lstm.fit(X_tr_t, y_train_o, epochs=1, batch_size=1, verbose=1,
                                    shuffle=False, callbacks=[early_stop])

#Training Time
end_time=time.time()-start_time
print("Time To Train",end_time)
```

Figure 44: Creating LSTM model

The forecast can then be developed, and an output can be generated either looking at the whole dataset, or just the time horizon for which the prediction is required, Figure 45 and Figure 46.

```
#Forecast Input Data dynamic
y_pred_lstm=model_lstm.predict(X_tr_t)
#y_pred_lstm_single_d=y_pred_lstm.reshape(y_pred_lstm.shape[0])
#X_tst_t_single_d=y_train_o.reshape(y_train_o.shape[0])

y_pred=sc_1_o.inverse_transform(y_pred_lstm)
y_actual=sc_1_o.inverse_transform(y_train_o)

x=list(y_pred)
y=list(y_actual)
pyplot.plot(y)
pyplot.plot(x)
plt.legend(['Actual', 'Predicted'])
pyplot.show()
```

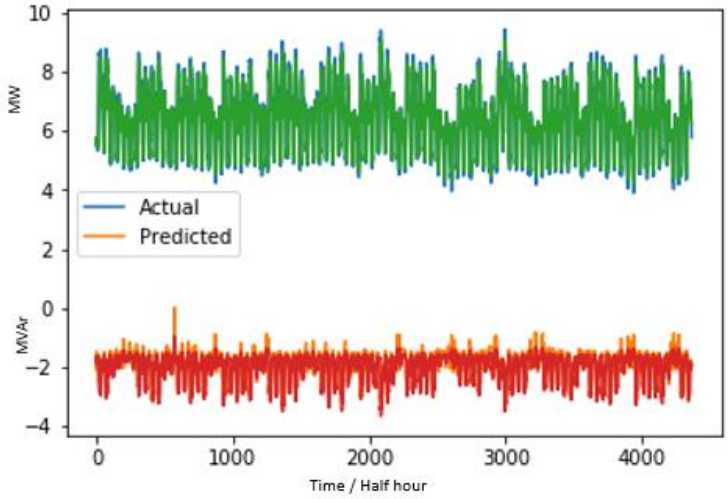


Figure 45: Prediction function with Forecast versus Actual output

```
#Forecast Input Data dynamic
y_pred_lstm=model_lstm.predict(X_pred_t)
#pred_lstm_single_d=y_pred_lstm.reshape(y_pred_lstm.shape[0])


y_pred=sc_r_pred.inverse_transform(y_pred_lstm[:335])
y_actual=sc_test.inverse_transform(y_test[:335])
#print(y_pred.size)
#print(y_actual.size)

x=list(y_pred)
y=list(y_actual)
pyplot.plot(y)
pyplot.plot(x)
plt.legend(['MW_Actual', 'Mvar_Actual','MW_Predicted','Mvar_predicted'])
pyplot.show()
```

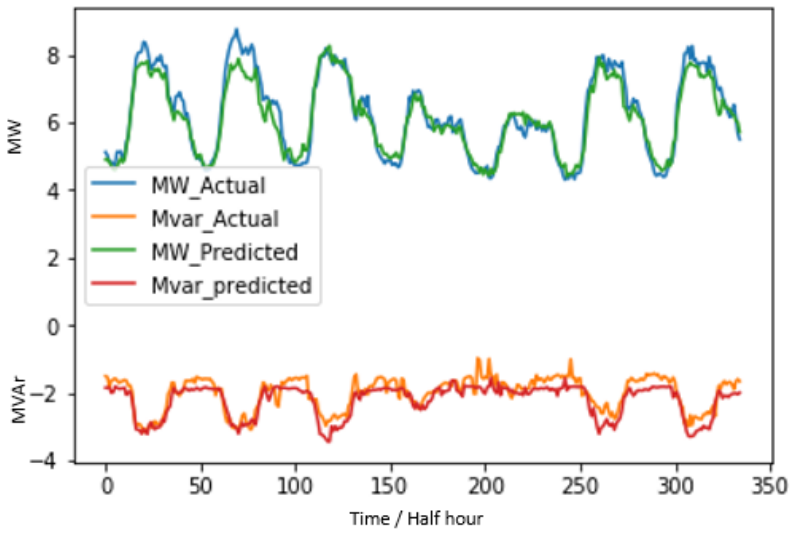


Figure 46: Output for specified time horizon

Finally, the error calculations are performed, Figure 47.

```
rmse = sqrt(mean_squared_error(y_actual, y_pred))
print('RMSE: %.3f' % rmse)
#MAPE
y_true, y_pred = np.array(y_actual), np.array(y_pred)
mape=np.mean(np.abs((y_true - y_pred) / y_true)) * 100
print('mape: %.3f' % mape)
mae=mean_absolute_error(y_true=y_actual, y_pred=y_pred)
print('MAE: %.3f' % mae)
```

Figure 47: Error calculation

5.3.3. Influence of Features

The results shown are for UC2, Cardiff South BSP for a one month ahead prediction. The Use Case is set up as defined for Forecast Period 1 in Section 4 and three sets of features are run. Each builds on the last by adding new features to the list, rather than by replacing previously tested features.

- Baseline: hour, day of week, quarter, month, year, day of year, day of month, week of year;

- Round 2: temperature, one hot encoding (Section 3.3) and holidays;
- Round 3: wind and solar output collected from weather sources and wind and solar physical models[16].

The predictions improve with the introduction of more data as shown in the graphs in Figure 48. The improvement can also be seen in the RMSE for each round of testing in Table 4.

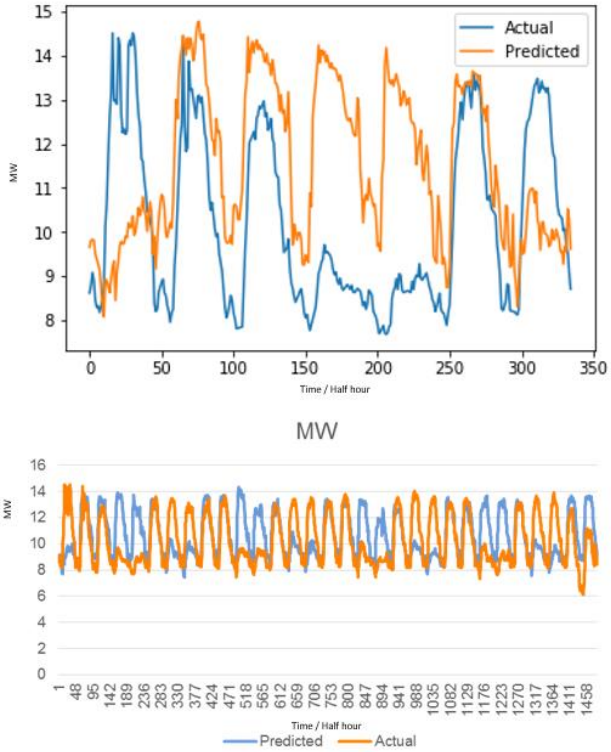


Figure 48: Actual versus Predicted. Top – Baseline, Middle – Round 2, Bottom – Round 3

Table 4: Feature impact on RMSE

Test	RMSE
Baseline	2.833
Round 2	2.644
Round 3	2.030

The impact on the RMSE by introducing temperature, one hot encoding and holidays shows a reduction of 7%. The introduction of further features shows another reduction of 23% in the RMSE. This shows that LSTM is able to identify trends in the additional data and improve on the errors calculated in the baseline testing. Where it has failed to identify trends, as the weekend days shows, more features specific to weekend rather than weekday could be used to enable this pattern to be understood by the LSTM's hidden layer.

5.3.4. Influence of Training Dataset Length

Considering UC2, as in the previous section, the forecasts are defined in the Use Case description in Section 4. Forecast 1 uses 12 months of historical data in the training set, whereas Forecast 2 uses six months of historical data. The prediction improves with the

shorter training set in Forecast 2, Figure 49. This is echoed in the RMSE values for the two forecasts, Table 5, where a reduction of 17% is seen in the RMSE for Forecast 2.

Table 5: Training set length impact on RMSE

Test	Training set length	RMSE
Forecast 1	12 months	2.833
Forecast 2	6 months	2.342

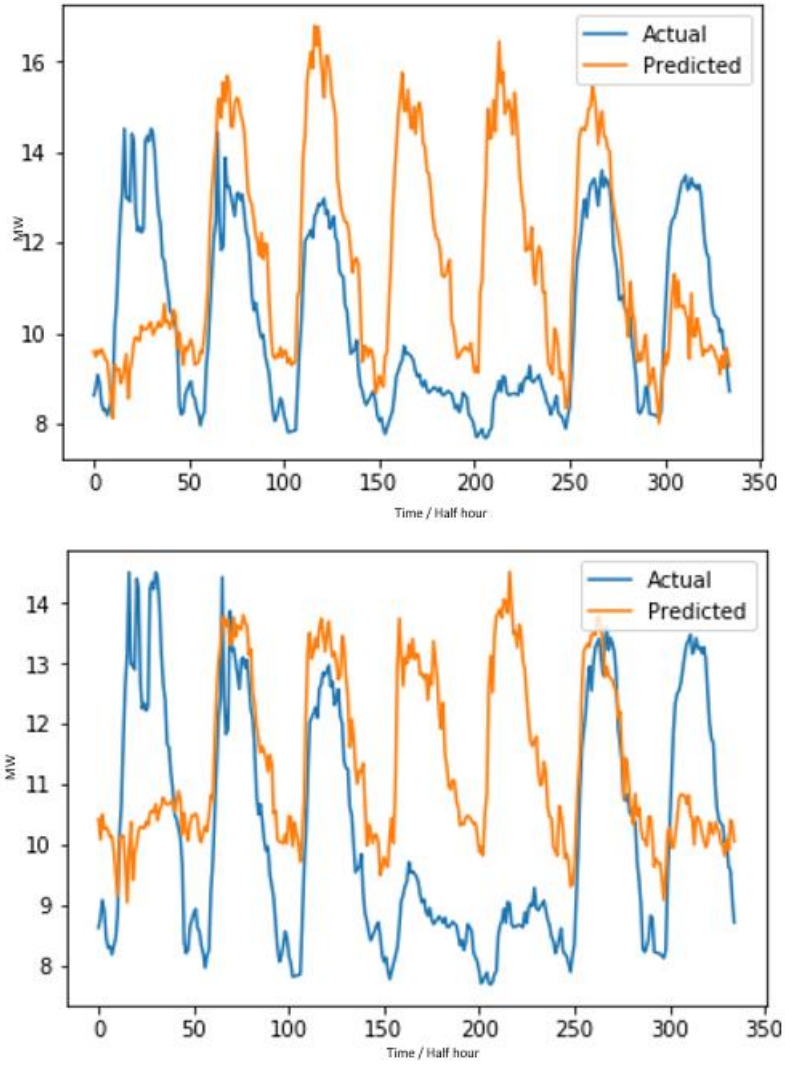


Figure 49: Forecast versus actual data. Top – Forecast 1. Bottom – Forecast 2.

6. Database

This section captures the database schema used for the historical data repository in the EFFS forecasting methods evaluation toolchain. The requirements for the database include that it can:

- Be replicated by third parties, in this case, AMT Sybex/Capita. This requires a shared schema and SQL scripts for creating the database.
- Be used as part of the EFFS forecasting evaluation toolchain, including scripts, in the form of Jupyter notebooks or Python scripts which populate the database from the raw input data.

6.1. Data Sources

Data sources used during the initial design stage include:

- WPD Load and Generation data for individual customers;
- GSP, BSP, and Primary substation power flows;
- Weather data.

6.2. Technologies

SGS has selected the following technologies for the EFFS forecasting methods evaluation toolchain:

- **PostgreSQL**. PostgreSQL is a mature open source database with commercial support. https://www.postgresql.org/
- **The TimescaleDB extensions to PostgreSQL.** The TimescaleDB extensions to PostgreSQL overcomes some of the traditional issues with storing large amounts of time-series data within a classical relational database. https://www.timescale.com/
- PG Admin. The standard database admin tool that comes with PostgreSQL. All scripts for creation of the EFFS forecasting methods toolchain databases are presented

6.3. Installation

To install the database on a MS Windows machine, simply following the instructions on the TimescaleDB website: https://docs.timescale.com/v1.2/getting-started

6.3.1.The first instruction deals with Installing.

The version that should be installed is version 10:

https://docs.timescale.com/v1.2/getting-started/installation/windows/installation-windows

Once this version is selected the site will provide details on how to install Timescale for Windows and to check that it has been installed correctly.

Prerequisites include:

- The installation of PostgreSQL. SGS is currently using the latest version 10 build for Windows.

6.3.2. The next instruction deals with Setting up TimescaleDB.

https://docs.timescale.com/v1.2/getting-started/setup

Follow the instructions on this page, creating a database with an appropriate name when the following instruction is given:

```
-- Extend the database with TimescaleDB
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

An error will occur if this is entered verbatim. The following must be used:

CREATE EXTENSION IF NOT EXISTS timescaledb WITH VERSION '1.2.1' CASCADE;

This should start timescale DB correctly and the screen shown in Figure 50 should be displayed:

Figure 50: TimescaleDB start screen

6.4. Database Schema

The database schema is presented using entity relationship diagrams, shown in Figure 51, Figure 52, and Figure 53. These diagrams capture the relationships between data and it has been used to derive the database schema. The schema itself is provided as an SQL script, which, when run in PG Admin, will create an instance of the EFFS database.

The database schema is based on a layered data architecture. Standard data architectures transform data from foundation layers though to a presentation layer. In the case of the work for EFFS, the following three layers are defined:

- The foundation layer (Figure 51): This layer contains the raw time-series data; configuration data from loggers; locational data; and classification of load groupings per relevant substation;
- The staging layer (Figure 52): This layer contains cleansed datasets, and training and test sets; and
- The results/presentation layer (Figure 53): This layer contains forecast results linked back to training and test sets.

TimescaleDB uses a special hypertable representation for time series data. It also support a series of special time series functions for the aggregation, up and down sampling of time series data.

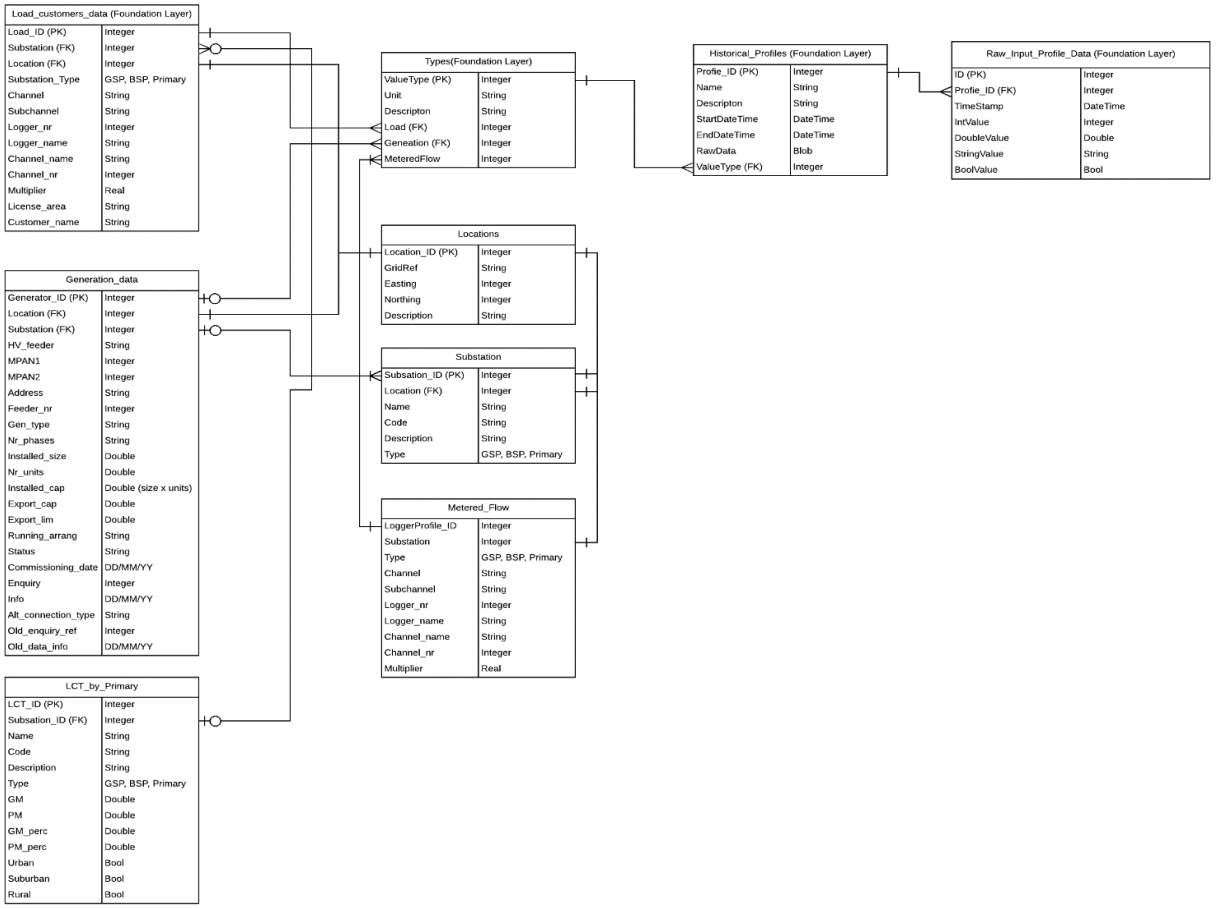


Figure 51: Foundation layer

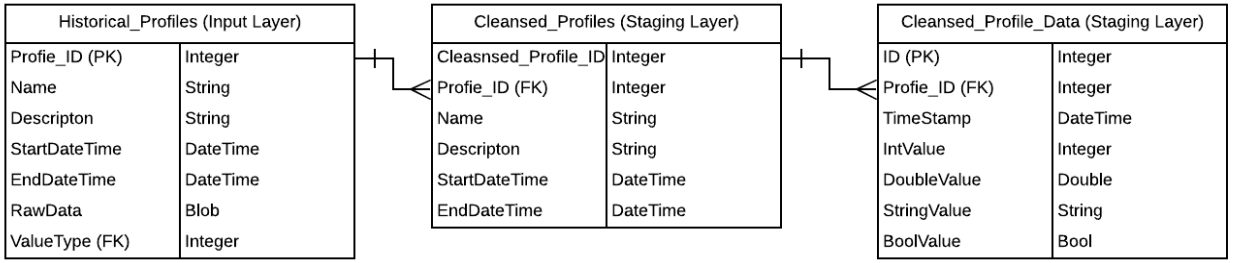


Figure 52: Staging layer

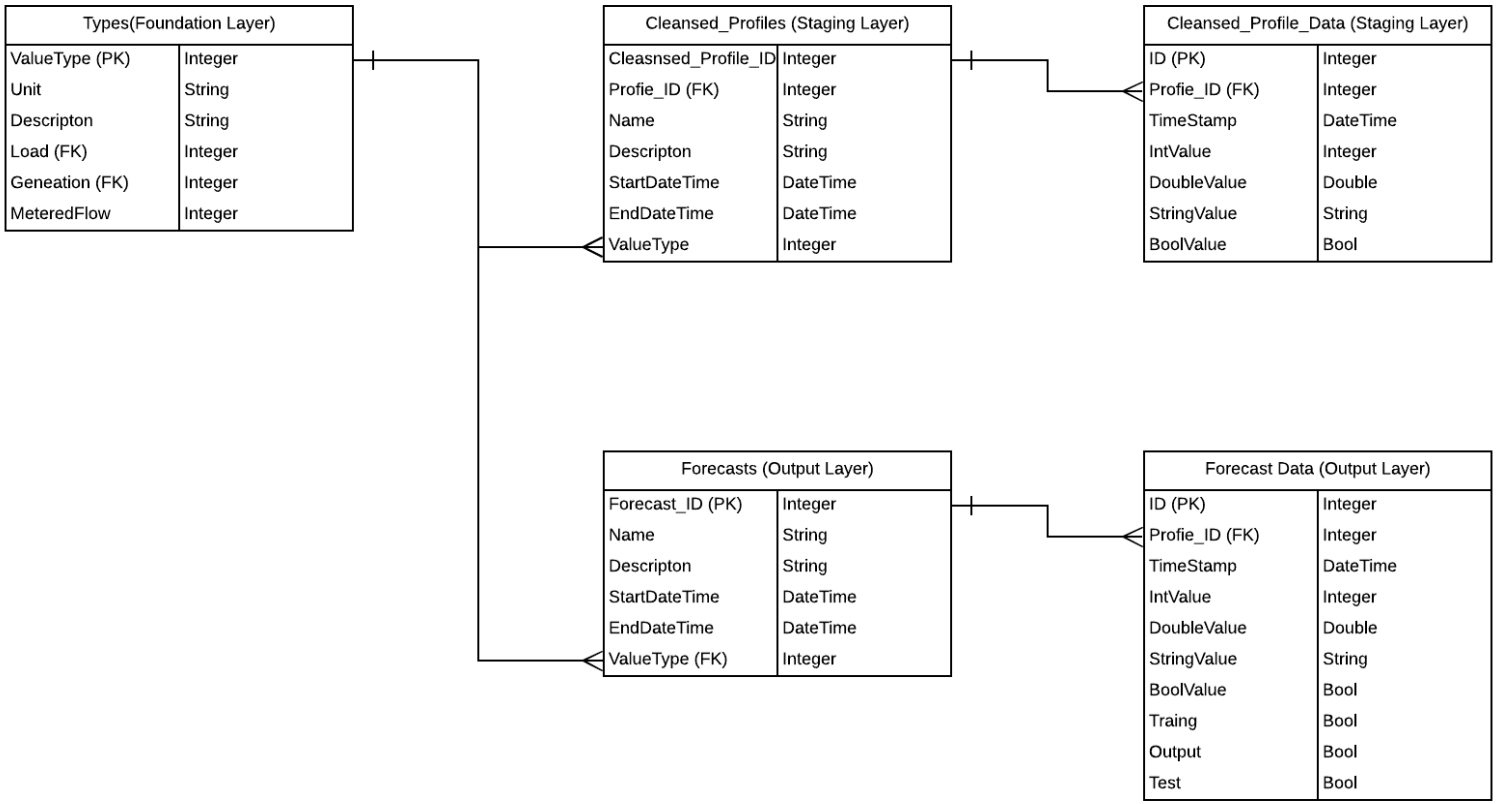


Figure 53: Output layer

6.5. Instruction for database creation:

Open PG Admin, the version used in this example is pgAdmin3 LTS by BigSQL, it is the default window 7 install. Other versions can be used, but the instructions may slightly vary:

1. Configure the local host, a timescaleDB database would have been created in Section 6.3 and shown in Figure 54. If a local host does not exist, one may need to be created - upon creation refresh the application and it should find any local PostgreSQL database;
2. Right click on the database, a series of options are presented. Select create script, as shown in Figure 55;
3. Copy and paste the script attached, into the new script and save with an appropriate name, click the green play button to run, as shown in Figure 56;
4. The database with the aforementioned schema has been updated and is shown in Figure 57.

The database is now ready to be populated with data and used as part of the forecasting for input data method.

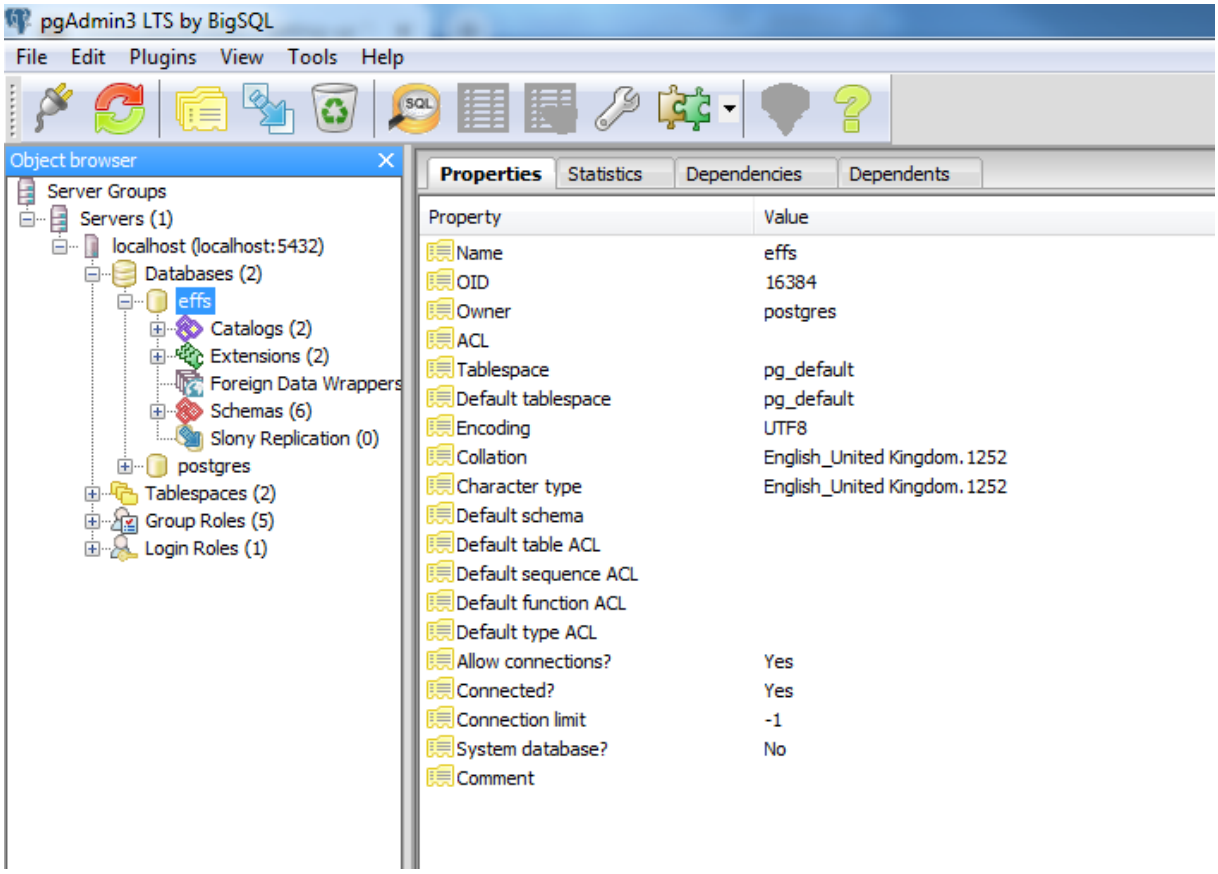


Figure 54: Configure localhost

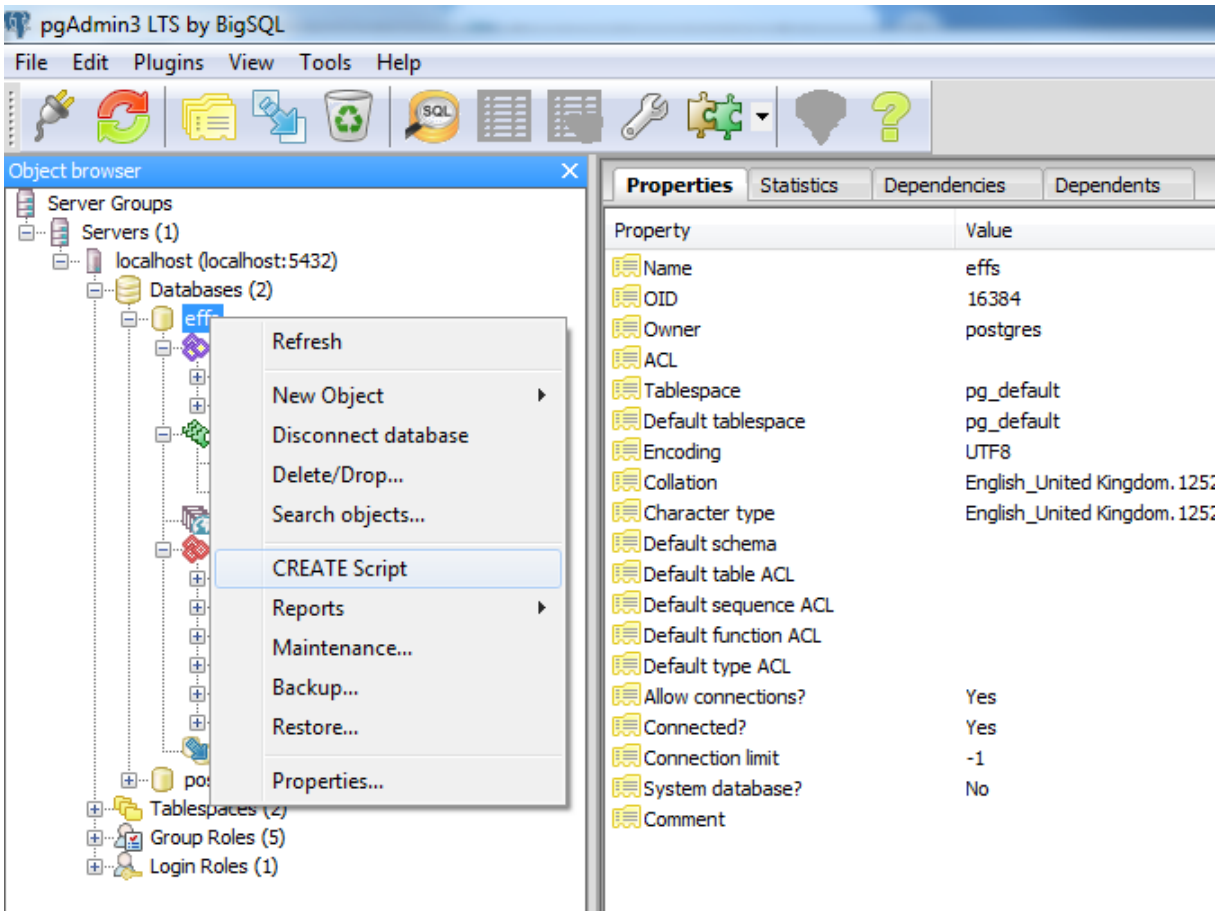


Figure 55: Create script

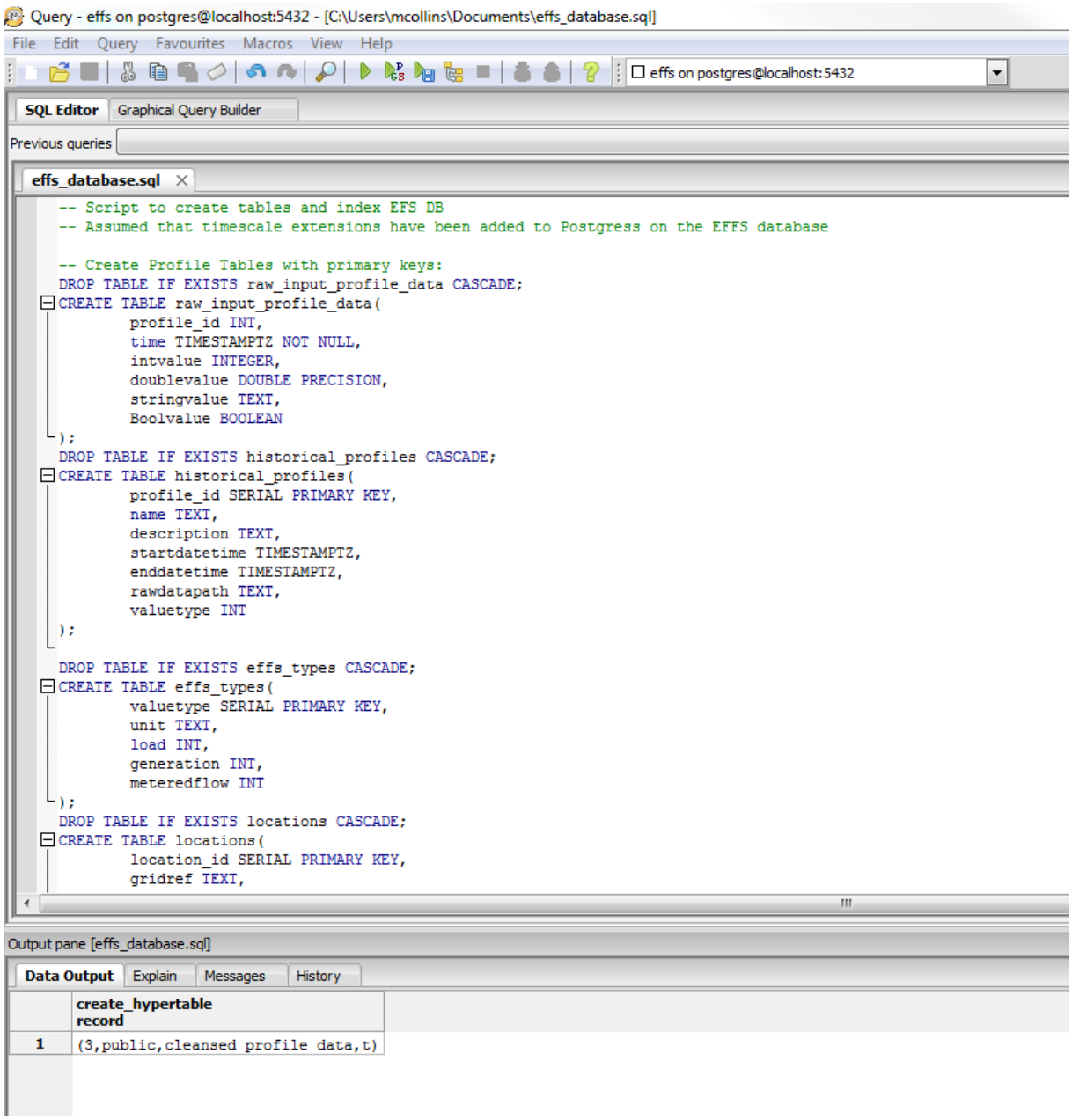


Figure 56: Update and save database creation script

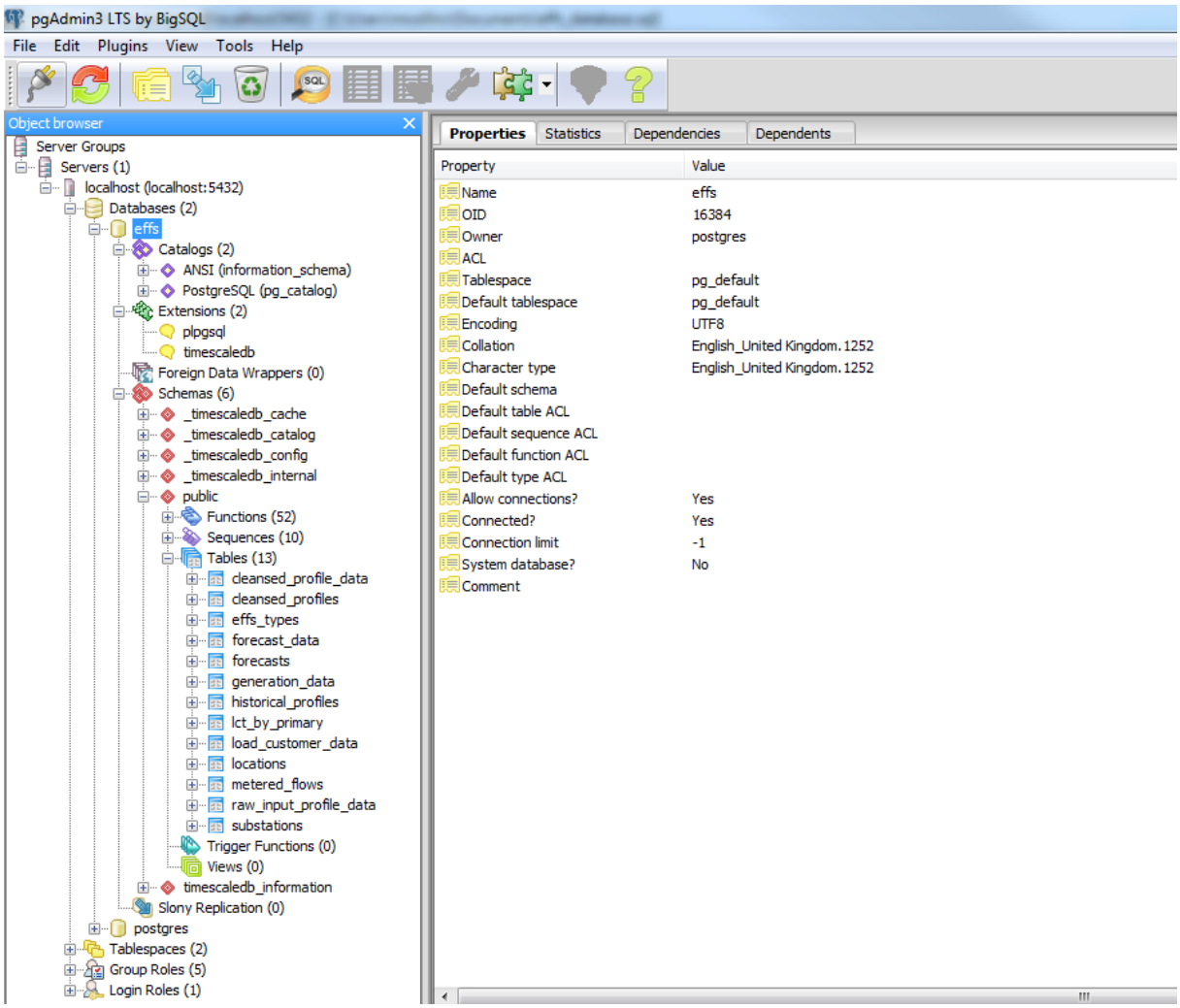


Figure 57: EFFS Database

6.6. Python SQL Interface

The database will interact with the forecasting methods by way of a python/SQL IO. It was explained in the toolchain. When data is to be extracted from the database the following instruction set.

6.6.1. Python SQL Input Instructions

```
try:
    connection = psycopg2.connect(user = "postgres",
                                  password = "",
                                  host = "127.0.0.1",
                                  port = "5432",
                                  database = "effs")
    cursor = connection.cursor()
    print ( connection.get_dsn_parameters(),"\n")
    ''' This Sets up the connection with the data base
    # The Example instruction allows for data to be submitted to the database
    for x in range forecast_data
        input="INSERT into forecast_data (profile_id,time) VALUES (forecast_data[0], forecast_data[1])"
        cursor.execute(input)
        connection.commit()
    '''


except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)
finally:
    #closing database connection.
        if(connection):
            cursor.close()
            connection.close()
            print("PostgreSQL connection is closed")
```

Figure 58: Python SQL Extract From Database

```
try:
    connection = psycopg2.connect(user = "postgres",
                                  password = "",
                                  host = "127.0.0.1",
                                  port = "5432",
                                  database = "effs")
    cursor = connection.cursor()
    print ( connection.get_dsn_parameters(),"\n")
    # Print PostgreSQL version
    ''' This Sets up the connection with the data base
    # The Example instruction allows for data to be extracted from the database
    #Once the database has been populated update the call for multiple records
    output="select * from raw_input_profile_data"
    cursor.execute(output)
    record = cursor.fetchall()
    for x in range (len(record)):
        if(record[1]==1):
            MW_in.append(record[4])
        else:
            temp_in.append(record[4])
    '''


except (Exception, psycopg2.Error) as error :
    print ("Error while connecting to PostgreSQL", error)
finally:
    #closing database connection.
        if(connection):
            cursor.close()
            connection.close()
            print("PostgreSQL connection is closed")
```

Figure 59: Python SQL Insert to Database

The I/O can now be used to pull information into the forecast methods and save data from the forecasts into the database.

6.7. Populating the Database

The structure of the database will be used to populate the database correctly, where primary keys will be used as foreign keys in the foundation layer schema to establish the correct links. Therefore, metadata is required to be populated first, as this will be used as database signposting to enable efficient extraction of data.

The ordering of this establishment is important, measurement data must be linked to an existing asset, therefore location data of all substation/transformers are established first, be it GSP, BSP, Primary, Load or Generator transformers.

Then for each substation/transformers their type is established and finally the associated information, measurements can be established.

After which, the raw time series data can be updated post metadata set up. Each table will be discussed with examples of how the shown in the database schema operate, and how metadata is established.

6.7.1. Location Data

The first table in the foundation layer to be populated is the Location table, presented in Table 6, with Prince Rock as an example:

Table 6: Location Table

	Location_id	area	gsp_name	bsp_name	primary_name	description	longitude	latitude
1	1	South West	Abham GSP	Plymouth BSP	Prince Rock	Prince Rock	50	-4

The parameters are as follows:

Location_id: This is the primary key, each unique location record will be assigned one of these.

Area: The licence area assets will be associated with and the geospatial location. For WPD these areas are:

- South West;
- South Wales;
- West Midlands;
- East Midlands.

gsp_name: This assigns the associated GSP name to the data.

bsp_name: This assigns the associate BSP name to the data, if there is not one, for example, a GSP entry, this will be left null.

primary_name: This assigns the associate primary name to the data, if there isn't one, for example, a BSP entry, this will be left null.

Description: This is the unique name identified for the entry, for GSP, BSP, and Primaries a good convention is to maintain these IDs. Where things may differ is for generators or load customers.

Longitude/Latitude: This is the data point's geographic reference represented in longitude and latitude by decimal degrees. It is suggested that all points are converted to this referencing before entry to ensure standardisation. However, if an alternative referencing was desired it could be achieved by updating these database entries.

6.7.2.Substation Data

The second table in the foundation layer to be populated is the substation data table presented in Table 7, using Prince Rock as an example.

Table 7: Substation Data

	substation_id	location_id	description	type
1	1	1	Prince_Rock	3

The parameters are as follows:

Substation _id: This is the primary key, each unique substation record will be assigned one of these.

Location_id: This is the foreign key, each substation that relates to the same location will be assigned the foreign key respective of this location and the description assigned. This allows for substations to be sorted by location.

Description: This is the unique name identified for this entry, although not entirely required as it can be inferred from the foreign key, it is used for reference to make sure of the correct assignment.

Type: This defines what type of substation it is. There are 5 types:

- GSP (1)
- BSP (2)
- Primary (3)
- Generator (4)

- Load (5)

6.7.3. Metered Flow

The third table is the metered flow table, the metered flow data establishes the metered flows at the substation, typically electrical parameters, however, this is extended to include weather data measurements for the site also. An example is shown in Table 8.

Table 8: Metered Flow Tables

	profile_id	substation	type	channel
1	1	1	3	PRINCE ROCKCB 27/19Power MW
2	2	1	3	PRINCE ROCKCB 27/19Power MVAr
3	3	1	3	MOUNT BATTEN Weather Station

The parameters are as follows:

Profile_id: This is the primary key and unique Identifier to the meter flow.

Substation_id: This is the foreign key identifying the metered flows with the substation it is associated with since there can be multiple measurements at a substation.

Type: This is the type of substation associated with the metered flow.

Channel: This is the defined channel that records the metered flow, this can be drawn from Data Logger, Met Office or any channel providing information for the metered flow.

6.7.4.Load Customers

The fourth table is for the Load Customers. These are typically large customers whose imports have a significant effect on network operations, due to being directly connected. Their data structure is presented in Table 9, using Bombardier 33 kV as an example.

Table 9: Load Customer Table

	load_id	substation	location	substation_type	channel	licence_area	customer_name
1	1	2	2	Primary	Customer Metering CB No1 MVA	East Midlands	Bombardier 33kV MW/MVAR/MVA

The parameters are as follows:

Load_id: The primary key uniquely identifying the site.

Substation_id: The foreign key assigning the customer to a substation.

Location_id: The foreign key assigning the customer to a location (this can vary from the substation and can, therefore, require a separate location reference).

Channel: The channel which measurement data is associated.

License Area: As titled.

Customer Name: As titled.

6.7.5.Generator Customers

The fifth table is for the Generator Customers. These are typically large customers whose exports have a significant effect on network operations, due to being directly connected. Their data structure is presented in, Table 10, using ALLER COURT 33kV SOLAR FARM as an example.

Table 10: Generator Customer Table

	generator_id	substation	location	gen_type	installed_cap
1	1	3	3	ALLER COURT 33kV SOLAR FARM	20

Load_id: The primary key uniquely identifying the site.

Substation_id: The foreign key assigning the customer to a substation.

Location_id: The foreign key assigning the customer to a location (this can vary from the substation and can, therefore, require a separate location reference).

Channel: The channel which measurement data is associated.

License Area: As titled.

Customer Name: As titled.

6.7.6.LCT By Primary

The sixth table is Load Customers By Primary. This holds data on the primaries for the number of customers associated with each primary. This was created to highlight how external information can be linked to the core forecasting objects. This was not populated during the project.

6.7.7.Effs_Types

The seventh table is the EFFS Types table, which determines the types of data that the EFFS forecasting methods will forecast for, however, as part of the forecasts, supplementary parameters are considered as a type. The full list is, for each GSP, BSP, Primary, Generator and Load customers:

- Uses and Forecasts
  - MW
  - MVAR
- Only uses:
  - Voltage
  - Amps
  - Temp
  - Solar Irradiance
  - Wind Speed
  - Wind Direction

An example from Prince Rock is shown in Table 11.

Table 11: Effs_Types Data

	ValueType	MeteredFlow	Unit	Description	Generation	Load
1	1	1	MW	PRINCE ROCKCB 27/19Power MW		

6.7.8.Historical Profiles

The eighth table, Table 12, assigns metadata associated with the measurement data. Given an overall summary of what it contains and signposting it to the relevant external tables.

Table 12: Historical Profiles Table

Profile_ID	ValueType	Name	Description	StartDateTime	EndDataTime

1	1	1	PRINCE ROCKCB 27/19Power MW	PRINCE ROCKCB 27/19Power MW	01/01/2014 00:00	15/02/2018 23:30

The parameters are as follows:

Profile_ID: The unique primary key for the historical profile entry.

ValueType: The foreign key to relate the data to other tables.

Description: Related to its measurement.

Start Time/End Time: The encompassing data period.

6.7.9. Raw Input Data

The ninth and final table is slightly different from the metadata tables and explained in Section 6.9.

6.8. Populating Metadata for Location and Substation Data

The metadata for the locations and substation tables can be found in various documents provided by WPD. All of which contain varying structures. In order to populate database automation scripts that interact with these external sources will need to be developed. An example in Figure 60 shows one way this can be achieved.

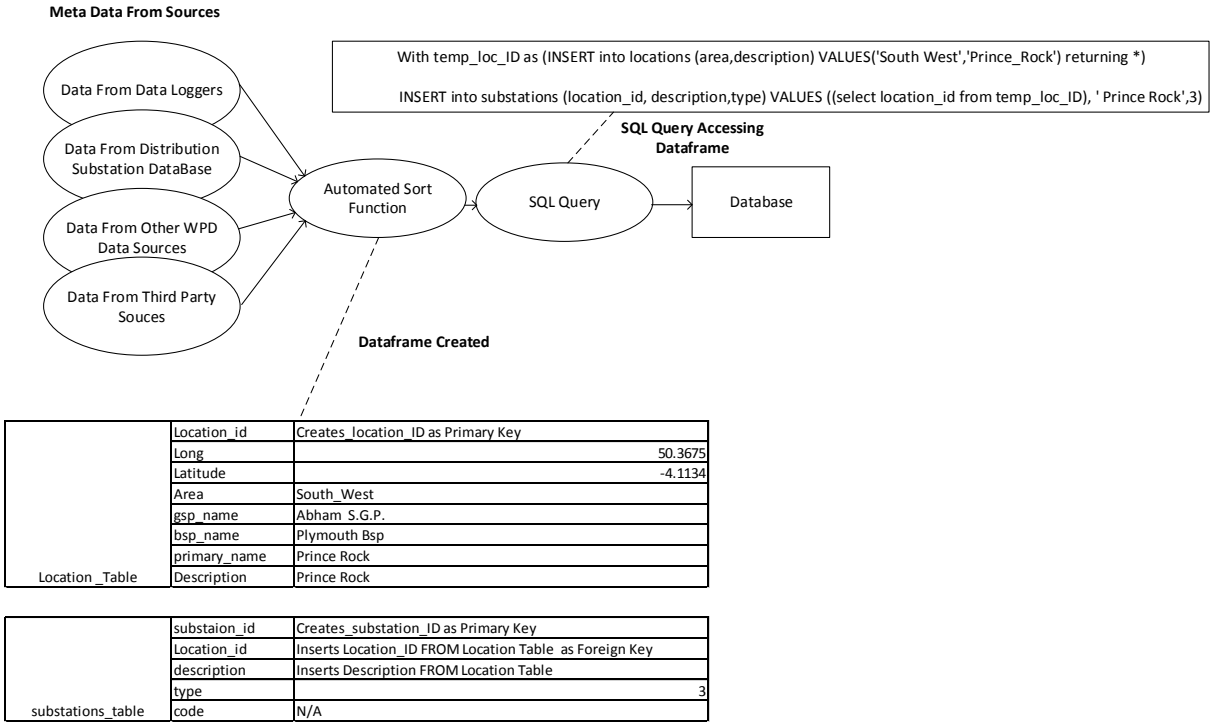


Figure 60: Populating Metadata Database proposed Scheme

After the initial data population, the SQL database can be updated routinely. This structure uses an implicit example where the network is structured as per the condition the data was provided. However, if primaries are switched to new B.S.Ps, or exist in a meshed state between two primaries, the substation table can be updated to consider a switch or appended to associate multiple foreign key entries. Therefore allowing the record to be extracted for the specific search query applied.

6.9. Updating Substations for time-series data

Once the metadata is assigned, the time series data can be input, this can be done by using the metadata signposts to update the foreign key assignments accordingly as part of a python SQL input/output (I/O) loop.

Where the SQL instruction has the form shown in Figure 61:

```
SELECT historical_profiles.profile_id FROM historical_profiles
where historical_profiles.valuetype = 1
```

Figure 61: SQL Update Query

Once the correct key is extracted new data can be looped into the raw_data_table as shown in Table 13.

Table 13: Raw Input Profile Table

Profile_ID	TimeStamp	Double Value
1	01/01/2014 00:00	6.58
1	01/01/2014 00:30	6.43
1	01/01/2014 01:00	6.32

6.10. Forecast and Cleansed Data

The forecast and cleansed layers are where the output results for external forecast methods (presented in this document) and external cleansing methods (not presented) can be placed. They have the same structure as the raw input data and should be looped into the foundation layer referencing in the exact same way. The allows the DNO to keep raw, forecasted and cleansed data separate so when configuring the forecasting and cleansing methods they can be assured on what the data source origins are.

6.11. Database Summary

The database section has shown how the SQL database is created using PostGresSQL with PGAdmin, alongside time series DB. The SQL script that creates the database schema and the database schema itself has also been defined.

How the tables can be populated with the metadata and updated with the subsequent time-series data has also been presented.

Once the database is populated Python I/O can be used to extract the data and input it into the forecast model, the I/O can then be used to update the forecast tables.

The main challenge surrounding the database is the automation of the population tasks. A process has been presented here with all the required SQL queries and I/O to make it work. However, the metadata and time series data will be in many uncommon structures and will have to be cleaned up using an automated sort process. This is outside the scope of this project, but an important task to consider whilst populating the database.

7. Results of Testing

7.1. Scenarios and Inputs

The testing scenarios and input data outlined in Table 14 build upon the Use Cases defined in Section 4. Each of the seven Use Cases are testing for all the time horizons (with the exception of UC7: Large Load Customer where the available input data of one calendar year does not allow for six months ahead forecasting). The forecasting method applied, XGBoost is tuned once for each Use Case. The method then automatically retrains itself for each forecast run, adjusting the amount of data used for each time horizon and period studied.

For each time horizon associated to the use case, the following is predicted:

- Six Months Ahead:
  - Six months for each quarter from the end of the validation period in the tuning dataset. There are six separate six month ahead predictions.
- Month Ahead:
  - One month for the calendar year following the end of the validation period in the tuning set. There are 12 separate month ahead predictions.
- Week Ahead:
  - The first week of every calendar month following the end of the validation period in the tuning set. There are 12 separate week ahead predictions.
- Day Ahead:
  - Every day in the first week of every calendar month following the end of the validation period in the tuning set. There is 84 separate day ahead predictions.
- Hour Ahead:
  - Every hour in the first week of each quarter following the end of the validation period in the tuning set. There are 672 separate hour ahead predictions.

In order to accommodate the testing outlined, the basic feature set, described in Table 14, is applied. This is done in order to achieve an efficient computation time for all testing. Weather data is however applied to the Generator forecasting as this is more relevant than other features.

Table 14: Scenarios and input data for testing

Location	Time Horizons	Features	Tuning, Validation, Forecasting Dates	Data Inputs & Sources
Indian Queens GSP **4x 240 MVA Transformers** **Forecasts are produced for each transformer, and an**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 14-12-2014 – 13-11-2015 Validation: 14-11-2015 – 14-12-2015 Forecasting: 01-01-2016 – 30-09-2017	Indian Queens SGP 180 – MW Indian Queens SGP 380 – MW Indian Queens SGP 480 – MW Indian Queens SGP 280 – MW Indian Queens SGP 180 – MVAR Indian Queens SGP 380 – MVAR Indian Queens SGP 480 – MVAR Indian Queens SGP 280 – MVAR Bank holidays for England and Wales[17]

[17] http://www.calendarpedia.co.uk

Location	Time Horizons	Features	Tuning, Validation, Forecasting Dates	Data Inputs & Sources
aggregate produced by summing individual transformers.				
Cardiff South BSP **2x 40 MVA Transformers** **Forecasts are produced for the aggregate BSP.**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 01-01-2015 – 30-11-2015 Validation: 01-12-2015 – 31-12-2015 Forecasting: 01-01-2016 – 30-09-2017	Cardiff SouthGRID 1Power MW Cardiff SouthGRID 2Power MW Cardiff SouthGRID 1Power MVAR Cardiff SouthGRID 2Power MVAr Bank holidays for England and Wales[17]
Prince Rock Primary **2x 17.25 MVA Transformers** **Forecasts are produced for the aggregate primary.**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 01-01-2015 – 30-11-2015 Validation: 01-12-2015 – 31-12-2015 Forecasting: 01-01-2016 – 30-09-2017	PRINCE ROCKCB 27/19Power MW PRINCE ROCKCB 27/21Power MW PRINCE ROCKCB 27/19Power MVAr PRINCE ROCKCB 27/21Power MVAr Bank holidays for England and Wales[17]
Truro BSP **2x 60 MVA Transformers** **Forecasts are produced for the aggregate BSP.**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 01-01-2015 – 30-11-2015 Validation: 01-12-2015 – 31-12-2015 Forecasting: 01-01-2016 – 30-09-2017	TRURO BSPCB 1T0Power MW TRURO BSPCB 2T0Power MW (inverted as measurement appears to be in the wrong direction) TRURO BSPCB 1T0Power MVAr TRURO BSPCB 2T0Power MVAr (inverted as measurement appears to be in the wrong direction) Bank holidays for England and Wales[17]
Llynfi Valley Primary **1x 12 MVA 1x 21 MVA Transformers** **Forecasts are produced for the aggregate primary.**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 01-01-2015 – 30-11-2015 Validation: 01-12-2015 – 31-12-2015 Forecasting: 01-01-2016 – 30-09-2017	LlynfiTrans 1Power MW LlynfiTrans 2Power MW LlynfiTrans 1Power MVAr LlynfiTrans 2Power MVAr Bank holidays for England and Wales[17]
Goonhilly Wind Farm, the Lizard, Cornwall **12 MVA Capacity**	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Quarter Month Year Day of Year Day of Month Week of Year Temperature	Tuning: 14-12-2014 – 13-11-2015 Validation: 14-11-2015 – 14-12-2015 Forecasting: 01-01-2016 – 30-	Goonhilly MW Goonhilly MVAr Temperature[18] Wind Output[18] Wind Speed[18]

[18] Renewables Ninja - https://www.renewables.ninja/ - for The Lizard, Cornwall.

Location	Time Horizons	Features	Tuning, Validation, Forecasting Dates	Data Inputs & Sources
		Wind Output Wind Speed	09-2017	
5MVA Capacity	Six Month Ahead Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Quarter Month Year Day of Year Day of Month Week of Year Temperature Wind Output Wind Speed	Tuning: 14-12-2014 – 13-11-2015 Validation: 14-11-2015 – 14-12-2015 Forecasting: 01-01-2016 – 30-09-2017	Ashercourt_Farm MW Ashercourt_Farm MVAr Direct Sunlight[18]
Large Load Customer **Maximum Demand 16 MVA**	Month Ahead Week Ahead Day Ahead Hour Ahead	Hour Day of Week Quarter Month Year Day of Year Day of Month Week of Year Holidays	Tuning: 01-01-2017 – 31-05-2017 Validation: 01-06-2017 – 30-06-2017 Forecasting: 01-07-2017 – 31-12-2017	Load Customer MW[19] Load Customer MVAr Bank holidays for England and Wales[17]

The results for each Use Case present the time taken for tuning and the minimum, average, and maximum times are taken for training and forecasting. The minimum, average, and maximum are given for the Root Mean Square Error (RMSE) and Mean Average Percentage Error(MAPE) which is a representation of performance accuracy. This is the standard deviation of the prediction errors. This is a measure of the size of the error that gives more weight to larger, or infrequent errors.

The minimum, average, and maximum are given for the accuracy of the prediction. This is based on the following calculation specified by WPD:

$$Accuracy\ (\%) = 100 - \left(\left|\frac{Actual - Prediction}{Actual}\right| \times 100\right)$$

The forecast is considered to be successful if:

- Forecasts are produced for all half hour timesteps in the prediction window;
- Accuracy is greater than 80% for 80% of the time; and
- Accuracy is greater than 50% for 80% of the time.

In some cases, the actual value is very close to zero, which can be a credible value for a substation. Dividing by such a small value results in the large negative numbers in some of the results.

Finally, the average over and underprediction, based on substation transformer capacity (or generation capacity or maximum demand as for UC6 and UC7) is given. This is to give an indication of the percentage exceedance of the transformer, and is calculated as:

$$Over\ or\ under\ prediction = \frac{(Predicted\ Capacity_{\%\ Tx\ Capacity} - Actual\ Capacity_{\%\ Tx\ Capacity})}{Predicted\ Capacity_{\%Tx\ Capacity}}$$

[19] Large Load 3

The following describes the results analysed to produce the tables shown in the following sections:

- Six Month Ahead
  - The six predictions performed.
- Month Ahead
  - The twelve predictions performed.
- Week Ahead
  - The twelve predictions performed.
- Day Ahead
  - The first day from each week prediction performed.
- Hour Ahead
  - The first 24 hour predictions performed.

The accuracy percentages show the average time greater than the specified accuracy calculated for each of the test cases for MW only. MVAr was also predicted, however, its Average Over/Under prediction was not presented as it is assumed MW accuracy it the key control for most procurement tasks. More comprehensive time-series, trend and histogram analysis of each result is presented in section 11: Appendix C.

7.2. UC1: Indian Queens GSP

7.2.1. Aggregated GSP (Sum of all Grid transformer loads)

Table 15: Aggregated UC1 Results - MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	5,166.7	5.6	0.1	55.8	-447,159.9	1,930.1	-1,461.7
	Average		9.3	0.2	62.2	-266.4		
	Max		14.0	0.2	72.0	100.0		
Month Ahead	Min	5,166.7	3.0	0.0	44.0	-650,887.5	420.8	-755.2
	Average		4.6	0.1	56.5	-788.5		
	Max		8.9	0.1	89.3	100.0		
Week Ahead	Min	5,166.7	2.7	0.0	38.8	-46,533.6	48.1	-98.8
	Average		4.9	0.0	52.6	-135.2		
	Max		8.9	0.1	75.6	99.9		
Day Ahead	Min	5,166.7	2.4	0.0	13.5	-16,038.4	26.0	-2.6
	Average		6.2	0.0	45.1	-77.2		
	Max		20.9	0.1	141.4	99.6		
Hour Ahead	Min	5,166.7	0.7	0.0	0.1	-196.9	0.3	-0.3
	Average		2.7	0.0	16.3	19.2		
	Max		20.6	0.2	112.3	96.8		

Table 16: Aggregated UC1 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	3,660.4	7.7	0.1	46.6
	Average		9.6	0.1	51.3
	Max		13.5	0.2	56.8
Month Ahead	Min	3,660.4	4.6	0.0	38.0
	Average		6.9	0.1	49.1
	Max		11.3	0.1	62.4
Week Ahead	Min	3,660.4	4.7	0.0	32.2
	Average		7.2	0.0	46.6
	Max		13.2	0.1	67.3
Day Ahead	Min	3,660.4	3.6	0.0	5.8
	Average		8.6	0.0	36.7
	Max		30.3	0.1	87.6
Hour Ahead	Min	3,660.4	1.0	0.0	0.0
	Average		3.1	0.0	15.5
	Max		18.2	0.2	165.8

Table 17: Aggregated UC1 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	30.61%	11.91%
Month Ahead	28.89%	11.69%
Week Ahead	25.07%	9.42%
Day Ahead	30.95%	13.39%
Hour Ahead	50.00%	25.00%

The aggregated UC1, Table 17, shows that the GSP accuracy criteria are never met. This is due to the patterns embedded in the data being too stochastic in nature to extract. Therefore, disaggregation is required at each transformer in an attempt to separate the underlying behavioral patterns.

7.2.2.Transformer 1

Table 18: UC1 Transformer 1 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,187.2	1.4	0.0	14.0	-28,733.3	204.6	-153.6
	Average		2.2	0.0	16.0	39.6		
	Max		2.6	0.0	19.9	100.0		
Month Ahead	Min	1,187.2	0.9	0.0	10.8	-25,876.9	54.5	-46.4
	Average		1.0	0.0	13.6	49.0		
	Max		1.3	0.0	19.1	100.0		
Week Ahead	Min	1,187.2	0.7	0.0	9.4	-9,883.0	8.7	-21.2
	Average		1.1	0.0	13.1	47.1		
	Max		1.7	0.0	17.6	100.0		
Day Ahead	Min	1,187.2	0.7	0.0	3.0	-486.5	0.6	-1.1
	Average		1.4	0.0	11.2	73.5		
	Max		4.7	0.0	37.6	99.6		
Hour Ahead	Min	1,187.2	0.2	0.0	0.0	76.7	0.0	-0.1
	Average		0.7	0.0	4.2	92.0		
	Max		7.0	0.0	26.6	100.0		

Table 19: UC1 Transformer 1 – MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	816.4	2.0	0.0	20.2
	Average		2.5	0.0	21.5
	Max		3.2	0.0	23.2
Month Ahead	Min	816.4	1.2	0.0	15.9
	Average		1.5	0.0	20.5
	Max		2.3	0.0	26.2
Week Ahead	Min	816.4	1.4	0.0	14.8
	Average		1.8	0.0	19.5
	Max		2.8	0.0	26.7
Day Ahead	Min	816.4	1.0	0.0	1.2
	Average		2.2	0.0	14.9
	Max		7.2	0.0	27.1
Hour Ahead	Min	816.4	0.3	0.0	0.0
	Average		1.0	0.0	6.4
	Max		4.2	0.1	43.8

Table 20: UC1 Transformer 1 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	81.51%	49.77%
Month Ahead	84.50%	56.88%
Week Ahead	83.90%	56.60%
Day Ahead	92.26%	61.90%
Hour Ahead	100.00%	97.92%

In Table 20 disaggregation has improved accuracy substantially, bringing it within an acceptable acceptance criteria. This conclusion is compounded by the next 3 disaggregated transformer results.

7.2.3.Transformer 2

Table 21: UC1 Transformer 2 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,854.0	0.8	0.0	14.0	-1,123.2	9.4	-0.6
	Average		2.7	0.1	14.6	50.8		
	Max		5.4	0.1	15.2	100.0		
Month Ahead	Min	1,854.0	0.4	0.0	11.9	-836.0	6.3	-0.5
	Average		1.2	0.0	14.1	57.5		
	Max		3.7	0.0	19.8	100.0		
Week Ahead	Min	1,854.0	0.4	0.0	10.4	-742.1	3.8	-0.5
	Average		1.1	0.0	12.8	60.0		
	Max		2.8	0.0	18.0	100.0		
Day Ahead	Min	1,854.0	0.3	0.0	3.5	-97.4	1.1	-0.3
	Average		1.3	0.0	11.3	78.8		
	Max		5.1	0.0	28.2	100.0		
Hour Ahead	Min	1,854.0	0.1	0.0	0.0	70.9	0.0	-0.1
	Average		0.3	0.0	4.1	92.6		
	Max		1.6	0.0	31.9	99.7		

Table 22: UC1 Transformer 2 Results - MVAr

MVAr		Tuning Time	Training Time	Forecasting Time	RMSE
Six Month Ahead	Min	1,396.9	1.2	0.0	4.8
	Average		1.5	0.0	5.9
	Max		1.7	0.0	6.9
Month Ahead	Min	1,396.9	0.7	0.0	4.2
	Average		1.0	0.0	5.6
	Max		1.5	0.0	6.7
Week Ahead	Min	1,396.9	0.7	0.0	3.1
	Average		1.1	0.0	5.4
	Max		2.9	0.0	7.0
Day Ahead	Min	1,396.9	0.5	0.0	1.4
	Average		1.1	0.0	5.1
	Max		6.9	0.0	14.3
Hour Ahead	Min	1,396.9	0.1	0.0	0.0
	Average		0.3	0.0	2.4
	Max		2.5	0.0	44.8

Table 23: UC1 Transformer 2 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	77.36%	51.05%
Month Ahead	80.86%	53.66%
Week Ahead	80.63%	55.83%
Day Ahead	92.26%	69.35%
Hour Ahead	100.00%	87.50%

7.2.4.Transformer 3

Table 24: UC1 Transformer 3 Results - MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,044.9	1.6	0.0	13.2	-189,504.5	260.5	-355.8
	Average		2.1	0.0	15.2	35.1		
	Max		2.9	0.0	17.6	100.0		
Month Ahead	Min	1,044.9	0.7	0.0	10.1	-83,725.5	104.0	-30.8
	Average		1.1	0.0	14.6	45.9		
	Max		1.8	0.0	30.0	100.0		
Week Ahead	Min	1,044.9	0.7	0.0	8.7	-56,549.0	53.5	-13.6
	Average		1.2	0.0	13.2	37.6		
	Max		1.6	0.0	20.7	100.0		
Day Ahead	Min	1,044.9	0.5	0.0	4.3	-54.0	0.5	-0.3
	Average		1.7	0.0	10.5	82.5		
	Max		6.6	0.0	34.8	100.0		
Hour Ahead	Min	1,044.9	0.2	0.0	0.0	74.5	0.0	-0.1
	Average		0.9	0.0	3.6	92.6		
	Max		7.8	0.1	24.2	99.6		

Table 25: UC1 Transformer 3 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	873.5	3.7	0.0	11.5
	Average		4.4	0.0	12.8
	Max		5.5	0.0	14.5
Month Ahead	Min	873.5	2.4	0.0	9.8
	Average		3.7	0.0	12.4
	Max		6.0	0.0	16.1
Week Ahead	Min	873.5	2.1	0.0	6.9
	Average		3.3	0.0	11.5
	Max		5.2	0.0	18.5
Day Ahead	Min	873.5	1.9	0.0	1.5
	Average		4.4	0.0	8.9
	Max		9.7	0.0	28.4
Hour Ahead	Min	873.5	0.5	0.0	0.0
	Average		1.4	0.0	3.7
	Max		6.9	0.0	45.3

Table 26: UC1 Transformer 3 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	78.75%	51.90%
Month Ahead	80.23%	55.52%
Week Ahead	83.83%	55.52%
Day Ahead	96.43%	69.35%
Hour Ahead	100.00%	95.83%

7.2.5.Transformer 4

Table 27: UC1 Transformer 4 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,080.6	1.7	0.0	14.6	-14827.7	10.1	-49.7
	Average		2.3	0.0	16.4	60.3		
	Max		3.1	0.0	19.3	100.0		
Month Ahead	Min	1,080.6	0.9	0.0	11.3	-13473.3	5.7	-14.7
	Average		1.3	0.0	14.2	65.8		
	Max		2.1	0.0	20.5	100.0		
Week Ahead	Min	1,080.6	0.9	0.0	10.4	-6017.0	3.9	-7.5
	Average		1.6	0.0	13.5	63.9		
	Max		2.8	0.0	19.4	100.0		
Day Ahead	Min	1,080.6	0.9	0.0	2.7	-25.4	0.4	-0.2
	Average		1.7	0.0	12.1	83.9		
	Max		4.5	0.0	40.7	99.8		
Hour Ahead	Min	1,080.6	0.2	0.0	0.0	-2.4	0.0	-0.1
	Average		0.8	0.0	4.5	88.5		
	Max		4.1	0.0	29.6	99.8		

Table 28: UC1 Transformer 4 Results - MVAr

MVAr		Tuning Time	Training Time	Forecasting Time	RMSE
Six Month Ahead	Min	573.6	0.7	0.0	10.1
	Average		1.3	0.0	11.1
	Max		3.1	0.0	12.2
Month Ahead	Min	573.6	0.4	0.0	8.0
	Average		0.8	0.0	10.7
	Max		1.4	0.0	13.4
Week Ahead	Min	573.6	0.5	0.0	7.5
	Average		1.0	0.0	10.2
	Max		2.2	0.0	15.0
Day Ahead	Min	573.6	0.3	0.0	1.7
	Average		0.9	0.0	7.8
	Max		6.6	0.0	17.8
Hour Ahead	Min	573.6	0.1	0.0	0.0
	Average		0.3	0.0	3.1
	Max		4.6	0.0	31.9

Table 29: UC1 Transformer 4 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	83.24%	52.10%
Month Ahead	86.70%	60.04%
Week Ahead	86.95%	59.04%
Day Ahead	98.51%	71.13%
Hour Ahead	95.83%	91.67%

7.3. UC2: Cardiff South BSP

Table 30: UC2 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,496.8	3.1	0.0	1.6	-57.0	0.6	-0.8
	Average		5.9	0.1	2.1	88.4		
	Max		12.1	0.1	2.5	100.0		
Month Ahead	Min	1,496.8	1.5	0.0	0.5	36.5	0.2	-0.3
	Average		2.7	0.0	1.6	89.4		
	Max		3.6	0.0	3.0	100.0		
Week Ahead	Min	1,496.8	1.4	0.0	0.3	36.9	0.1	-0.2
	Average		2.5	0.0	1.1	91.9		
	Max		3.6	0.0	2.2	100.0		
Day Ahead	Min	1,496.8	1.3	0.0	0.2	77.1	0.1	-0.1
	Average		2.8	0.0	0.9	94.9		
	Max		10.2	0.0	5.6	100.0		
Hour Ahead	Min	1,496.8	0.4	0.0	0.0	91.1	0.0	0.0
	Average		1.0	0.0	0.3	97.5		
	Max		4.9	0.1	8.2	100.0		

Table 31: UC2 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	1,157.7	3.6	0.0	0.5
	Average		5.6	0.1	1.0
	Max		7.6	0.1	2.1
Month Ahead	Min	1,157.7	2.4	0.0	0.4
	Average		3.6	0.0	0.5
	Max		4.2	0.0	0.7
Week Ahead	Min	1,157.7	1.7	0.0	0.3
	Average		3.9	0.0	0.5
	Max		6.7	0.0	0.8
Day Ahead	Min	1,157.7	1.6	0.0	0.1
	Average		3.1	0.0	0.4
	Max		4.8	0.0	1.5
Hour Ahead	Min	1,157.7	0.5	0.0	0.0
	Average		1.4	0.0	0.1
	Max		10.6	0.1	0.8

Table 32: UC2 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	99.42%	79.23%
Month Ahead	99.94%	83.50%
Week Ahead	99.78%	92.11%
Day Ahead	100.00%	97.32%
Hour Ahead	100.00%	100.00%

The BSP predictions have performed well in the near term, fall short of six month ahead predictions for >80% of the time, albeit fractional.

7.4. UC3: Prince Rock Primary

Table 33: UC3 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1481.2	2.8	0.0	0.5	4.2	0.4	-0.5
	Average		3.4	0.0	0.8	93.0		
	Max		3.9	0.1	1.7	100.0		
Month Ahead	Min	1481.2	1.3	0.0	0.3	16.9	0.3	-0.3
	Average		1.6	0.0	0.5	94.8		
	Max		2.2	0.0	1.4	100.0		
Week Ahead	Min	1481.2	1.3	0.0	0.3	50.6	0.2	-0.2
	Average		1.9	0.0	0.4	95.5		
	Max		3.0	0.0	0.7	100.0		
Day Ahead	Min	1481.2	1.2	0.0	0.2	79.8	0.1	-0.1
	Average		2.0	0.0	0.4	96.7		
	Max		6.4	0.0	2.6	100.0		
Hour Ahead	Min	1481.2	0.3	0.0	0.0	84.0	0.0	0.0
	Average		0.8	0.0	0.2	97.3		
	Max		4.0	0.0	5.2	100.0		

Table 34: UC3 Results - MVAr

MVAr		Tuning Time	Training Time	Forecasting Time	RMSE
Six Month Ahead	Min	1033.6	1.2	0.0	0.3
	Average		1.6	0.0	0.3
	Max		2.5	0.0	0.5
Month Ahead	Min	1033.6	0.7	0.0	0.2
	Average		1.0	0.0	0.3
	Max		1.6	0.1	0.4
Week Ahead	Min	1033.6	0.6	0.0	0.2
	Average		0.9	0.0	0.2
	Max		1.6	0.0	0.3
Day Ahead	Min	1033.6	0.5	0.0	0.1
	Average		0.9	0.0	0.2
	Max		2.8	0.0	0.9
Hour Ahead	Min	1033.6	0.2	0.0	0.0
	Average		0.6	0.0	0.1
	Max		7.3	0.1	0.9

Table 35: UC3 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	98.23%	96.05%
Month Ahead	99.98%	98.59%
Week Ahead	100.00%	99.33%
Day Ahead	100.00%	99.70%
Hour Ahead	100.00%	100.00%

Primary prediction provides the most consistent and accurate. This shows that there is deeply recognisable behaviour in the primary. Where this is exhibited in other primaries, it should give DSO confidence in these predictions.

7.5. UC4: Truro BSP

Table 36: UC4 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,263.1	1.8	0.0	9.8	-68,461.0	411.5	-308.5
	Average		2.3	0.0	10.3	10.9		
	Max		2.7	0.0	11.0	100.0		
Month Ahead	Min	1,263.1	1.0	0.0	9.0	-66,584.7	130.8	-47.8
	Average		1.2	0.0	10.1	24.7		
	Max		1.5	0.0	12.4	100.0		
Week Ahead	Min	1,263.1	0.8	0.0	7.1	-13,177.5	13.7	-26.7
	Average		1.2	0.0	9.5	39.2		
	Max		2.2	0.0	13.1	100.0		
Day Ahead	Min	1,263.1	0.6	0.0	3.5	-5,321.0	8.8	-1.4
	Average		2.0	0.0	8.0	46.1		
	Max		12.4	0.0	23.7	100.0		
Hour Ahead	Min	1,263.1	1.8	0.0	9.8	-68,461.0	-0.1	-0.2
	Average		2.3	0.0	10.3	10.9		
	Max		2.7	0.0	11.0	100.0		

Table 37: UC4 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	1,773.2	0.7	0.0	2.1
	Average		1.2	0.0	2.3
	Max		1.6	0.0	2.6
Month Ahead	Min	1,773.2	0.6	0.0	1.7
	Average		1.3	0.0	2.2
	Max		2.5	0.0	2.7
Week Ahead	Min	1,773.2	0.4	0.0	1.3
	Average		2.3	0.0	2.2
	Max		7.2	0.0	3.3
Day Ahead	Min	1,773.2	0.2	0.0	0.7
	Average		1.2	0.0	1.8
	Max		3.4	0.0	4.3
Hour Ahead	Min	1,773.2	0.0	0.0	0.0
	Average		0.3	0.0	1.1
	Max		4.1	0.1	6.8

Table 38: UC4 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	68.99%	29.88%
Month Ahead	73.48%	33.75%
Week Ahead	73.41%	34.10%
Day Ahead	85.12%	45.54%
Hour Ahead	100.00%	52.08%

The Truro BSP does not perform as well as Cardiff BSP, this is due to the patterns becoming harder to extract, similar to the proposal of disaggregating for a GSP to each transformer, a similar technique could be applied here to improved the accuracy criteria to acceptable levels.

7.6. UC5: Llynfi Valley Primary

Table 39: UC5 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,400.7	0.7	0.0	1.1	-1,860.2	9.6	-0.6
	Average		1.0	0.0	1.6	74.9		
	Max		1.2	0.0	2.4	100.0		
Month Ahead	Min	1,400.7	0.4	0.0	1.1	-502.2	2.4	-0.4
	Average		0.7	0.0	1.4	80.1		
	Max		1.2	0.0	2.3	100.0		
Week Ahead	Min	1,400.7	0.4	0.0	0.6	-50.0	0.7	-0.3
	Average		0.9	0.0	1.1	83.2		
	Max		2.7	0.0	1.6	100.0		
Day Ahead	Min	1,400.7	0.3	0.0	0.3	0.0	0.2	-0.2
	Average		0.9	0.0	0.9	82.4		
	Max		2.5	0.0	3.9	100.0		
Hour Ahead	Min	1,400.7	0.1	0.0	0.0	0.0	0.0	0.0
	Average		0.3	0.0	0.3	94.3		
	Max		1.3	0.0	3.1	100.0		

Table 40: UC 5 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	785.7	0.4	0.0	1.3
	Average		0.7	0.0	1.7
	Max		1.1	0.0	2.6
Month Ahead	Min	785.7	0.3	0.0	1.0
	Average		0.5	0.0	1.3
	Max		1.1	0.0	1.5
Week Ahead	Min	785.7	0.2	0.0	0.9
	Average		0.5	0.0	1.2
	Max		2.4	0.0	1.6
Day Ahead	Min	785.7	0.2	0.0	0.1
	Average		0.7	0.0	0.9
	Max		4.5	0.0	1.9
Hour Ahead	Min	785.7	0.1	0.0	0.0
	Average		0.3	0.0	0.4
	Max		1.7	0.1	2.7

Table 41: UC5 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	97.54%	87.36%
Month Ahead	97.74%	86.97%
Week Ahead	98.96%	91.39%
Day Ahead	100.00%	98.51%
Hour Ahead	100.00%	100.00%

Similar to the Prince Rock primary, this primary also performs strongly, adding more confidence that behaviour at more reduced voltage results in more accurate predictions. This is driven by behavioural patterns being easier to identify by machine learning techniques when there is less diversity of behaviours embedded in the profiles.

7.7. UC6: Generator Customer (Wind farm)

Results for the generator customers introduces a comparison of RMSE/Capacity, where the wind farm under test has a 12 MW capacity.

Table 42: UC6 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE (MW)	RMSE/ Capacity (%)	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	863.4	0.2	0.0	2.9	24.17	-164,436.8	743.8	-23.6
	Average		0.4	0.0	3.3	27.50	-212.2		
	Max		0.8	0.0	3.9	32.50	100.0		
Month Ahead	Min	863.4	0.1	0.0	0.8	6.67	-174,602.1	261.0	-6.4
	Average		0.2	0.0	2.4	20.00	-285.2		
	Max		0.5	0.0	4.0	33.33	100.0		
Week Ahead	Min	863.4	0.1	0.0	0.8	6.67	-27,606.3	40.6	-1.2
	Average		0.3	0.0	2.3	19.17	-116.0		
	Max		0.8	0.0	4.1	34.17	100.0		
Day Ahead	Min	863.4	0.1	0.0	0.3	2.50	-827.5	2.6	-0.3
	Average		0.2	0.0	1.5	12.50	71.9		
	Max		1.3	0.0	6.0	50.00	100.0		
Hour Ahead	Min	863.4	0.0	0.0	0.0	0.00	38.7	0.1	0.0
	Average		0.1	0.0	0.9	7.50	85.7		
	Max		1.2	0.1	9.5	79.17	99.5		

Table 43: UC6 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Six Month Ahead	Min	570.6	0.8	0.0	0.0
	Average		1.2	0.0	0.1
	Max		1.9	0.1	0.3
Month Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.5	0.0	0.0
	Max		0.8	0.0	0.0
Week Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.6	0.0	0.0
	Max		1.0	0.0	0.0
Day Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.6	0.0	0.0
	Max		1.1	0.1	0.0
Hour Ahead	Min	570.6	0.1	0.0	0.0
	Average		0.2	0.0	0.0
	Max		3.4	0.0	0.0

Table 44: UC6 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Six Months Ahead	37.33%	12.76%
Month Ahead	40.35%	18.68%
Week Ahead	48.91%	27.49%
Day Ahead	87.20%	71.73%
Hour Ahead	87.50%	79.17%

The wind farm prediction is acceptable in the near term but falls away in the future, this is mainly due to the increased variability of seasonal wind patterns versus day to day, a problem faced by weather forecaster with decades of experience and more complicated model. An acceptable criterion in the short term still allow the DSO to procure confidently in the near term horizon.

7.8. UC7: Generator Customer (Solar farm)

Table 45: UC7 Results MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time(s)	RMSE (MW)	RMSE/ Capacity (%)	Accuracy	Average Over Prediction (MW)	Average Under Prediction
Six Month Ahead	Min	863.4	0.2	0.0	0.54	10.8	-5905.2	60.1	-0.8
	Average		0.4	0.0	0.67	13.4	58.4		
	Max		0.8	0.0	0.73	14.6	100.0		
Month Ahead	Min	863.4	0.1	0.0	0.31	6.2	-1661.6	17.6	-0.9
	Average		0.2	0.0	0.63	12.6	53.2		
	Max		0.5	0.0	0.86	17.2	100.0		
Week Ahead	Min	863.4	0.1	0.0	0.1	2.0	-397.8	5.0	-0.8
	Average		0.3	0.0	0.68	13.6	66.6		
	Max		0.8	0.0	0.97	19.4	100.0		
Day Ahead	Min	863.4	0.1	0.0	0.07	1.4	-106.6	2.1	-0.7
	Average		0.2	0.0	0.42	8.4	73.5		
	Max		1.3	0.0	1.2	24.0	100.0		
Hour Ahead	Min	863.4	0.0	0.0	0.04	0.80	-32.0	0.1	-0.2
	Average		0.1	0.0	0.32	6.40	74.2		
	Max		1.2	0.1	0.72	14.40	99.6		

Table 46: UC7 Results MVAr

MVAr		Tuning Time	Training Time	Forecasting Time	RMSE
Six Month Ahead	Min	570.6	0.8	0.0	0.0
	Average		1.2	0.0	0.1
	Max		1.9	0.1	0.2
Month Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.5	0.0	0.0
	Max		0.8	0.0	0.0
Week Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.6	0.0	0.0
	Max		1.0	0.0	0.0
Day Ahead	Min	570.6	0.3	0.0	0.0
	Average		0.6	0.0	0.0
	Max		1.1	0.1	0.0
Hour Ahead	Min	570.6	0.1	0.0	0.0
	Average		0.2	0.0	0.0
	Max		3.4	0.0	0.0

Table 47 UC7 Accuracy Results

	Accuracy	
	>50%	>80%
Six Months Ahead	72.28%	58.16%
Month Ahead	73.08%	54.70%
Week Ahead	77.38%	52.68%
Day Ahead	76.19%	60.12%
Hour Ahead	89.58%	62.50%

The solar predictions are most accurate in the near time but drop away quite quickly from accepted accuracy. However, once the accuracy has dropped it stabilises. This is due to the very predictive nature of seasonal and diurnal irradiance embedded in the patterns. Where solar forecasts may suffer in the near term is cloud cover effects, or low pressure introducing changeable conditions. Adding these weather phenomena as features in the future may improve near terms accuracy.

7.9. UC8: Large Load Customer

Table 48: UC8 Results – MW

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Six Month Ahead	Min	1,159.2	0.3	0.0	0.1	-1,037.6	5.8	-0.6
	Average		0.4	0.0	3.3	32.4		
	Max		0.6	0.0	6.0	100.0		
Month	Min	1,159.2	0.3	0.0	0.1	-265.0	2.6	-0.6

MW		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE	Accuracy (%)	Average Over Prediction (%)	Average Under Prediction (%)
Ahead	Average		0.4	0.0	2.7	58.4		
	Max		0.9	0.0	4.4	100.0		
Week Ahead	Min	1,159.2	0.2	0.0	0.0	-126.7	1.3	-0.4
	Average		0.4	0.0	2.0	64.5		
	Max		1.6	0.0	6.3	99.8		
Day Ahead	Min	1,159.2	0.1	0.0	0.0	63.8	0.0	0.0
	Average		0.2	0.0	0.4	95.8		
	Max		1.2	0.0	4.9	100.0		
Hour Ahead	Min	1,159.2	0.3	0.0	0.1	-1037.6	5.8	-0.6
	Average		0.4	0.0	3.3	32.4		
	Max		0.6	0.0	6.0	100.0		

Table 49: UC8 Results - MVAr

MVAr		Tuning Time (s)	Training Time (s)	Forecasting Time (s)	RMSE
Month Ahead	Min	1,348.7	0.3	0.0	0.0
	Average		0.5	0.0	0.5
	Max		0.8	0.0	0.7
Week Ahead	Min	1,348.7	0.3	0.0	0.0
	Average		0.4	0.0	0.4
	Max		0.6	0.0	0.6
Day Ahead	Min	1,348.7	0.2	0.0	0.0
	Average		0.5	0.0	0.3
	Max		1.4	0.0	0.8
Hour Ahead	Min	1,348.7	0.1	0.0	0.0
	Average		0.3	0.0	0.1
	Max		3.4	0.0	1.2

Table 50: UC8 – Accuracy Calculations

	Accuracy	
	>50%	>80%
Month Ahead	66.66%	27.43%
Week Ahead	71.58%	29.41%
Day Ahead	79.17%	47.32%
Hour Ahead	100.00%	93.75%

7.10. Forecasting with Active Network Management Data

An initial example forecast was run for a case where a generator is subject to Active Network Management (ANM) control. This is real-time control of generator export with respect to a specific constraint location that the generator contributes to. ANM Data has been provided by a UK DNO. The data provided is for January 2017 and is for two measurement points at GSP transformers and for the ANM controlled wind farm output. The aim of this forecast is to predict the output of the wind farm.

The data is extracted from the ANM Historian and it is based on change. This has required data processing to organise the data into a per second profile from 1st January 2017 to 12th January 2017. A day ahead forecast is produced for the 12th of January 2017 based on the previous 11 days.

No temperature or other weather data has been included in this forecast given the requirement to sample it at a per second resolution, however, should this information be required there are upsampling techniques that can be employed. The features used in this forecast are:

- Hour;
- Day of Week;
- Quarter;
- Month;
- Year;
- Day of Year;
- Day of Month; and
- Week of Year.

The error metrics are shown in Table 51.

Table 51: ANM Data Forecast Error Metrics

Error Metric	Value
Mean Squared Error	171.4
Mean Absolute Error	10.5
Root Mean Squared Error	13.1
Mean Absolute Percentage Error	32.6

The prediction is shown in Figure 62.

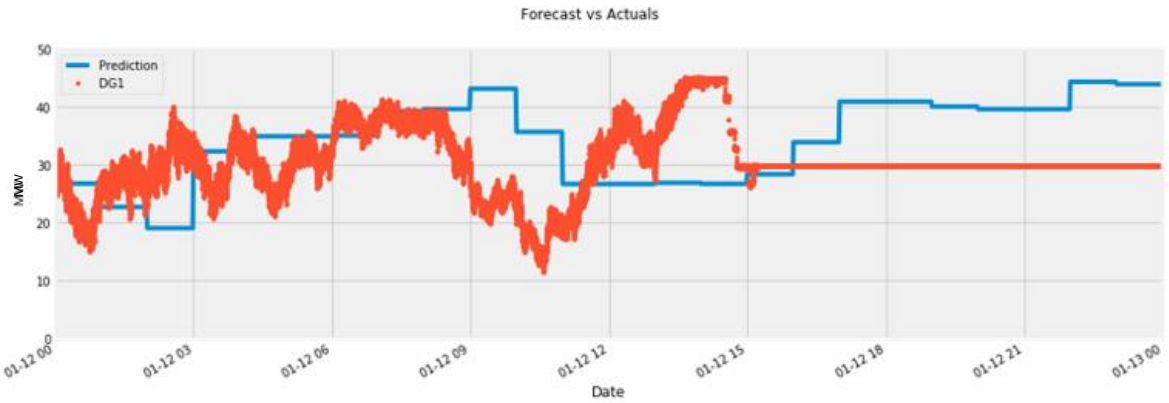


Figure 62: ANM Data forecast versus actual

The prediction appears to follow a more stepped profile, rather than the very fluid profile seen in the actual data, resulting in the smoothing of some peaks and troughs. There is potential for this to be smoothed by using a different time resolution, for example minutely.

The actual data is frozen after 15:00 on 12th January and so some scrutiny of the input data would be required, as recommended for all future forecasts.

Additionally, the use of secondly data significantly increases the tuning and training times. These are recorded as 4978 seconds to tune the model and 116 seconds to train it.

The result here show that second by second data would require additional inputs in order to produce usable ultra-short term forecasts, potentially including weather data, and the selection of inputs would depend on the intended application of the forecast.

An potential avenue of further investigation would be to look at how second-by-second data could be used to update hour ahead forecasts for when peaks or contraints are likely to occur. For example, a weather front moving from west to east will affect DER based on their geographical location. The windfarm to the west output picks up before the windfarm in the east. ANM data could be used to investigate how forecasts could be corrected based on learning the relationship of how behaviour is related between sites of the same type, e.g. wind or solar, where second-by-second or minute-by-minute data is available.

7.11. Summary of results

The key observations from the results generated show that:

- For a GSP where there is already a high level of aggregation in the power flow data, the prediction improves if the GSP is disaggregated down to its individual transformers, and predictions are performed for each in turn. The prediction for the whole GSP is a sum of the individual transformer tuning, training, forecasting times and errors.
- At the BSP level using a feature set that does not include temperature or wind and solar data still results in a suitable performance for Use Case 2. The accuracy metric shows that over 80% accuracy for more than 80% of the time for all time horizons except for the six months ahead prediction. For BSPs that have a significant penetration of embedded generation connected behind it, including these additional features will help to improve the prediction. It is expected that this will be the case for Use Case 4.
- At a Primary level, using a feature set that does not include temperature or wind and solar data still results in a suitable performance. As with the BSP case, if there is significant penetration of embedded generation behind the Primary, these additional features can help to improve the prediction. For demand dominated Primaries, including temperature in the feature set may also improve the prediction given the correlation between demand and temperature.
- Predicting for generator customers will require a different feature set. For renewable generation such as wind and solar pertinent weather data (temperature, wind speed, wind output, solar irradiance, and solar output) will be required. It is more favourable to apply wind speed/direction and solar irradiance data to models to produce a generator export in kW given the nonlinear relationship between renewable resource and generation. The impact of the day of the week will be less significant, however, seasonality across the year will still be important. A variety of models are in the public domain as referenced from the renewables ninja website.
- For load customers, the prediction is satisfactory using a feature list that does not include temperature. For individual large load sites, the performance of the method is satisfactory without this additional feature, however adding temperature if predicting for a number of aggregated load customers may improve the prediction.

- Tuning the model is not required every time a forecast is run. For the testing run in this section, the model was tuned once for each Use Case, and then a number of forecasts were run based on that single set of hyperparameters.
- The model was trained ahead of every forecast. This is required as it ensures the most recent and relevant historical data is being applied to the prediction. Training the model takes a significantly shorter length of time than tuning, meaning that training for every forecast is not temporally impractical.
- Predicting for longer time horizons requires more input data than predicting for shorter time horizons. Predicting for six months ahead has a requirement for sufficient training data that allows the method to determine the trends in order to predict so far ahead in the future. Conversely, predicting for an hour ahead will not require a full year of data, and providing that volume of information could result in overfitting. The six months ahead timeframe was chosen purely to see what kind of accuracy could be obtained that far in advance rather than because it was expected that a six month lead time would be required to remedy any shortfall in flexibility services. Given the low level of accuracy compared to month ahead forecasts, it would be useful to test forecasts at a three month ahead timeframe to determine whether they were accurate enough to support flexibility service planning. It is likely that a three month lead time would allow for remedial actions to take place if a shortfall in flexibility services was found.
- The accuracy of the prediction improves as the time horizon shortens. In every Use Case, the accuracy is improved in the day ahead and hour ahead predictions when compared with the longer time horizons. In all Use Cases, the accuracy is greater than 50% for more than 80% of the time for the hour ahead time horizon.
- For using ANM data for ultra-short term forecasting, more work is needed to understand the required input data set for producing usable forecasts.

8. Conclusions

This project focused on the analysis of methods and solutions for forecasting load and generation at the distribution level. It was structured in three major steps:

1. Build a conceptual framework for being able to produce forecasts for GSPs, BSPs, primaries, and significant loads or generators;
2. Explore the methods and solutions for building the necessary blocks for building the developed framework;
3. Perform thorough testing on the best candidate methods for forecasting at the distribution level.

To be able to operationalise forecasting at the distribution level, significant automation is required as the number of variables to be forecasted is too large. Within the DSO vision, there will be a need to forecast at different levels of aggregation and for multiple time horizons from very short-term to longer-term horizons.

Therefore, the creation of a database and methods that reduce user-in-loop requirements was proposed. It was also within the scope of the project to deliver a toolchain that relied as much as possible on open source libraries.

8.1. Database Solution

The database solution enables a common source of data for forecasting methods to interact with. It will allow for the data required for forecasts to be fed into Python based forecasting methods, and hold the outputs seamlessly. The structuring of the database will allow querying and appending to the database conveniently as new data sets are added as features or new assets added.
Furthermore, the database will allow for the extraction of the data as part of load flow forecasts to be extracted from the database as part of procurement tasks, as they become more defined in the other areas of the projects.

One major challenge for the database is the automation of the population tasks. A process has been presented here with all the required SQL queries and input/output to make it work. However, the metadata and time series data will be in many uncommon structures and will have to be cleaned up using an automated sort process. This is outside the scope of this project, but an important task to consider while populating the database.

Although a cleansed data table is provided in the database, it was assumed the external data science methods applied to cleanse raw input data would be undertaken by the DNO or a third party and exported to the cleansed table for use in forecasts. It is left to the DNO to manage this data quality, only raw data with the very minimal of fixes was used in this project.

8.2. Forecasting Methods

As to forecasting, three methods emerged as powerful candidates for becoming a method of reference for the DSO of the future: artificial intelligence based methods LSTM and XGBoost as well as the conventional ARIMA method. The latter was selected as a benchmark as it is one of the most widely used methods for forecasting. The former are trending as solutions that provide good accuracy results. Other conventional and AI based methods were tested, but with less interesting results.

It was found that conventional and AI based methods can perform equally if properly parameterised and trained. However, the conventional ARIMA method requires more user-

in-the-loop and deeper data science skills to master effectively. ARIMA as a general rule takes longer to train than the AI based counterparts and is significantly more complex to test different combinations of features (or regressors). Given that the expected volume and applications of forecasting at the distribution level is very large, these are significant setbacks to the usage of conventional methods. In addition to this, the most complete ARIMA libraries are only available in R language, whereas AI based methods are vastly supported both in R and Python.

Among the AI based solutions XGBoost, a tree based method, is a key reference. The libraries are user friendly and it allows understanding what the final method values within its decision trees. It is also fast to train and relatively easy to get to good accuracy results. XGBoost provided the best results of the three methods tested, closely followed by LSTM. LSTM, a recurring neural network method, could not be fully explored due to the lack of Graphical Processor Unit (GPU) machines that could improve training times. Thus, the number of feature combinations that could be tested with LSTM was inferior to the number of tests conducted with XGBoost. It is expected that if more tests would have been run, the performance of LSTM would have increased to levels comparable with XGBoost, but likely not improve beyond the XGBoost level.

XGBoost is, therefore, the recommended method, as it allows simplified testing of features and it can also be easily and effectively automated. Because of the wide community of users among the data scientists, it is supported in a number of open source formats and it is expected that there will be continued support for years to come.

8.3. Tuning Approach

In order to get the best results from AI based methods, it is necessary to tune the training parameters. To do so, it is recommended that an historical dataset different from the training set is used to avoid a phenomenon called overfitting, which occurs when the forecast corresponds to exactly to or very closely to the historical dataset, instead of drawing trends and extrapolating into the future.

Given that the datasets available were limited in extension and that there was an aim to build multiple forecasts across the available period, the selected tuning set was covering the same period as the first training set analysed. Results did not show any signs of overfitting, but still, it is recommended that tuning in a deployment scenario is done using older datasets.

The tuning action is, in fact, a search process that looks for the optimal combination of hyperparameters required to run the AI method training task. However, generic optimisation tools do not work in this case as there is no information on the derivatives of the function being optimised. Different methods are applicable, based generally in random search or involving some sort of heuristic process. Such a process can be a computational burden. It is a particular concern in the LSTM implementation as LSTM takes longer to train than XGBoost. The current state-of-the-art solution to improve the tuning process is to apply Bayesian optimisation processes combined with a tree-structured Parzen estimator. Bayesian optimisation is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric. The tree-structured Parzen estimator application structures the hyperparameter search space ordering the search for hyperparameters.

This combination of methods is available with another open source tool called HyperOpt. It was successfully applied in the tuning process of all the results presented in this report. Further testing is recommended to identify the most relevant hyperparameters to be

optimised, as by increasing the number of hyperparameters within the optimisation, finding near optimal solutions becomes increasingly challenging.

The conducted analysis showed that for the XGBoost case the number of estimators, the maximum depth of a tree and minimum child weight are critical parameters on the quality of final forecasts. Other hyperparameters seem to be impactful as well but it is hard to assess the trade-off of including them or not in the optimisation process. It was also observed that a conservative approach for the number of estimators is advisable and instead of optimising its value a sufficiently large number of trees should be imposed. In all presented results 1000 was the selected number of trees. It never presented worst results than cases with fewer trees and the computational burden does not increase significantly the training period. It also allows HyperOpt search to focus on other sensitive variables. The only identified drawback is that as the number of trees increases it becomes more complex for a human analysis of the tree contents, albeit still possible.

A possible action for the future is to perform a two level optimisation where first an optimisation of the most critical hyperparameters is performed and second the critical parameters are fixed and optimisation is applied to the second set of important parameters. This will penalise speed for an incremental gain in accuracy.

8.4. Results and Key Recommendations

To test the final implementation of the forecasting process, seven use cases were defined: one referring to a GSP, two for BSPs, two for primaries, one large load, and one wind generator. Initially, there was a general expectation that the greater the level of aggregation the easier it would become to build a meaningful forecast. Throughout the project, it was concluded that this is only valid up to a certain level and depends on several factors.

Primaries and BSPs presented the best accuracy levels, with a slight edge towards primaries, followed by load and generator individual profiles and finally the GSP. The reason behind this is that moving from individual loads / generators to the primary level there is a gain in aggregation that defines a more consistent pattern and makes it easier to forecast.

Moving one level higher in aggregation would, in theory, provide even better results as aggregating cancels behavioural dispersion of energy consumption or isolated impacts of generation fluctuation. However, there is an increasing importance of the physical grid and grid control aspects on the distribution of the power flows in the network, hence introducing a disturbance to the forecasting model. Actions such as load transfers, tapping of transformers or existing links to other substations alter the flow distribution in ways that are not trivial for the forecasting methods to pick up.

This effect becomes even more important at the GSP level as controllability and ties to other network areas proliferate. To get meaningful results at the GSP level there was a need to break the analysis down to the individual transformer level. In all other aggregated cases, the net flow at the substation, computed as the sum of the flows at the different transformers that compose that substation, provided good enough results. When aggregated at a country level the aggregation benefits should again be very important and the effects of the grid get cancelled out due to the fact that everything is being considered and not only parts of the network.

Having analysed the quality of the results by aggregation it is also necessary to analyse the impact of the forecast horizon on performance. In this case and as it was expected from the start, the greater the horizon the less accurate the results are on average. Yet, in terms of minimum and maximum accuracy ranges, it is verified that in the very short-term

forecasting, hour and day ahead, the results can fluctuate more than in week or month ahead. These present minimum and maximum values that are much closer to the average accuracy metric.

Six months ahead was, as expected, the case with smaller accuracy as it refers to a really long period. Perhaps with several more years of historical data, this could be improved. Month ahead, week ahead and day ahead generally provided good results that could likely be used with confidence and with increased levels of accuracy to procure services for the DSO.

The hour ahead presented good results, but the granularity of the data was not sufficient to provide greater resolution forecasts. Adding additional sources of data such as smart meter data or ANM data of the area of the GSP / BSP / primary would likely further improve the quality and the applications of such forecast outputs.

Finally, feature-wise, several were tested and these can be grouped in different categories. There are basic features that are recommended at all times: hour of the day, day of the week, quarter, month, year, day of the year, day of the month, week of the year. Some of these, such as day of the week can and should be one hot encoding, which is the action of instead of labelling weekdays from 1 to 7, creating seven binary variables to avoid misinterpretation by the forecasting method. In the forecast of generation, some of these might not be relevant, such as the day of the week, but it will always depend on the type of generator. If a combined heat and power generator is being forecasted there will most likely be a correlation of power output and day of the week.

The category of features is basic weather related features. Oftentimes, load is driven by temperature, which is fairly easy to get forecasts for, but also other variables, such as humidity or air pressure. The latter ought to be more difficult to get forecasts for that could be used as features and therefore temperature has been successfully used in many of the use cases.

The last category is also weather related, but influencing the generators instead of the loads. Solar irradiance, wind speed, and direction forecasts are important features when forecasting generation. It is advisable that these are converted into power values using a generic turbine or solar panel that is close to the ones on site. It improves the quality of the results as the relation between wind speed/solar irradiance and power is not linear.

8.5. Comparison with UKPN's KASM Project

To put the results of the EFFS forecasting methodology in context we have compared accuracy metrics against UK Power Networks' (UKPN) Kent Active System Management (KASM) project. This is a relevant comparison as it is relatively recent and also, like EFFS, the forecasting focussed on the 33kV and 132kV networks.

The aim of UK Power Networks (UKPN) Kent Active System Management (KASM) was to demonstrate Great Britain's first use of real-time power system modelling and short term forecasting on electricity distribution networks. It delivered enhanced visibility and analysis capabilities regarding the power flows and stability of the 132 kV network to control room engineers and outage and network planners.

Three specific capabilities were successfully developed and trialled:

- The sharing of real-time measurement data between the NG and UKPN control rooms via an Inter-Control Centre Communications Protocol (ICCP) link;

- Forecasting modules which uses advanced analytics and machine learning techniques to provide realistic load and generation forecasts for the KASM trial area; and
- Contingency analysis which provides state estimation and power flow calculations for contingent scenarios.

In terms of forecasting, this workstream developed the systems that are used in conjunction with the contingency analysis: generator and load modules; forecasting engine; historical generation and load patterns; historical weather patterns; optimisation and normalisation modules.

The EFFS forecasting evaluation has specifically outlined the forecasting methods explored and selected for development and provides enough information for them to be recreated. The details of the feature exploration and their impacts on the predictions are provided, along with the results of extensive testing on different customers, locations, voltage levels, and time horizons.

The KASM project assessed the accuracy of its proprietary ensemble forecasting method but using different metrics. The EFFS results compare favourably when looking at the MAPE and RSME/Capacity figures achieved:

The outputs of the forecasting from the KASM project are summarised in Table 52.

Table 52: KASM/EFFS comparison

	KASM	EFFS
MAPE for Load	9% day ahead	3.5% day ahead
RMSE/Capacity for Solar	10% day ahead	8.4% day ahead
RMSE/Capacity for Wind	39.95	17.37

Had the EFFS results been significantly worse than those achieved by KASM, it would have prompted further analysis and potentially a change of approach. However the favourable results suggest that while the risk that poor forecasts due to poor underlying data remains (as it would for any forecasting method), the risk of poor forecasts due to poor forecasting methodology is now low.

8.6. Transferability to other DNOs

One of the objectives of the project is to produce a forecasting evaluation report that can be published and shared with the industry that details the models, their efficacy and makes available the tools for other DNO stakeholders to use in their own licence areas. The forecasting methods investigated and implemented in this project have been developed using fully open source libraries and environments. The methods themselves are agnostic to the location, voltage level, and time horizon. The Use Cases described in this report cover GSP, BSP, and Primary substations and load and generation customers, in various locations across both WPD's and other DNO licence areas.

The performance of the forecasting methods will also depend on the input data made available. While this project has not employed data cleansing techniques on the input data, some scrutiny should be applied to ensure erroneous values are removed, identify if meters

are polling data with correct polarity, and identify when metering equipment is not functioning correctly.

8.7. Recommendations for Further Research

The limited time available for this portion of the EFFS project has meant that a limited number of Use Cases have been investigated. While the tests conducted can provide a general view of performance at different voltage levels and for different time horizons, there is some investigation that could be carried out at a later stage to further understand the operational benefits and limitations of the methods. This includes:

- Further testing with ANM system data to determine the benefit of forecasting a higher time resolution;
- Investigation of performance on lower voltage feeders;
- Further investigation into the disaggregated approach to predicting at a GSP level; and
- The implementation of LSTM on a GPU to determine if performance improves.

9. Appendix A: Generator Types

The generator types provided:

- Solar;
- Wind;
- CHP;
- Biomass;
- Anaerobic Digestors;
- STOR; and
- Battery.

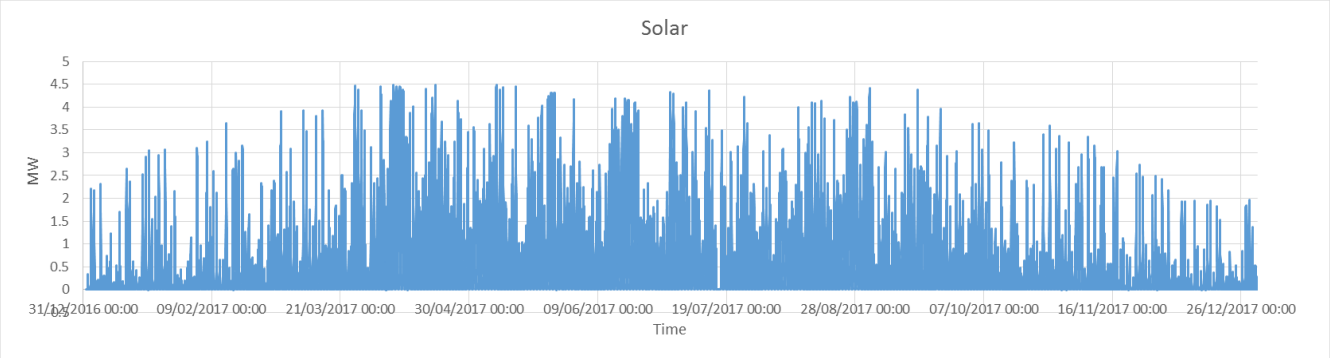


Figure 63: Solar Profile (MW)

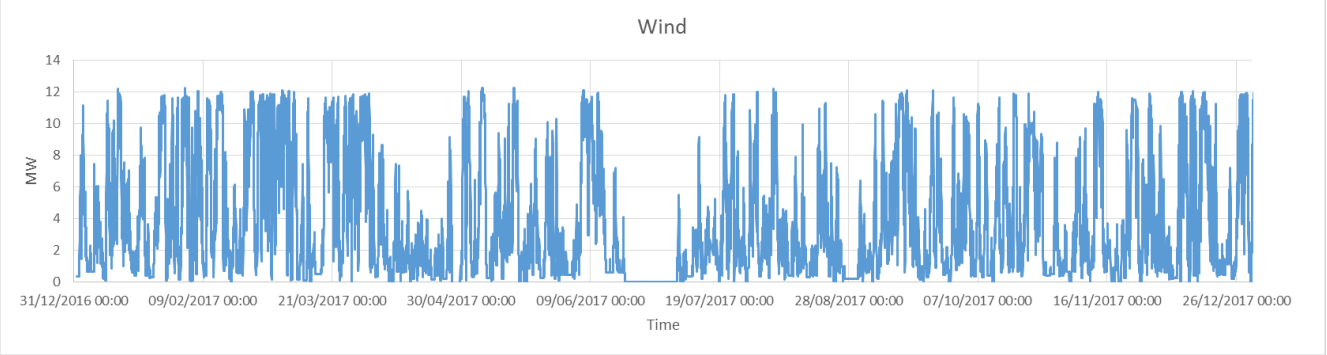


Figure 64: Wind Profile (MW)

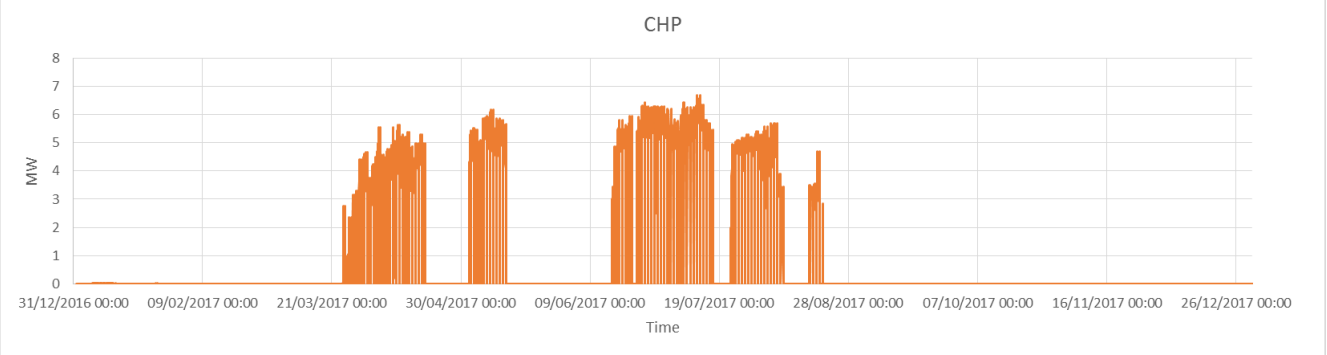


Figure 65: CHP (MW)

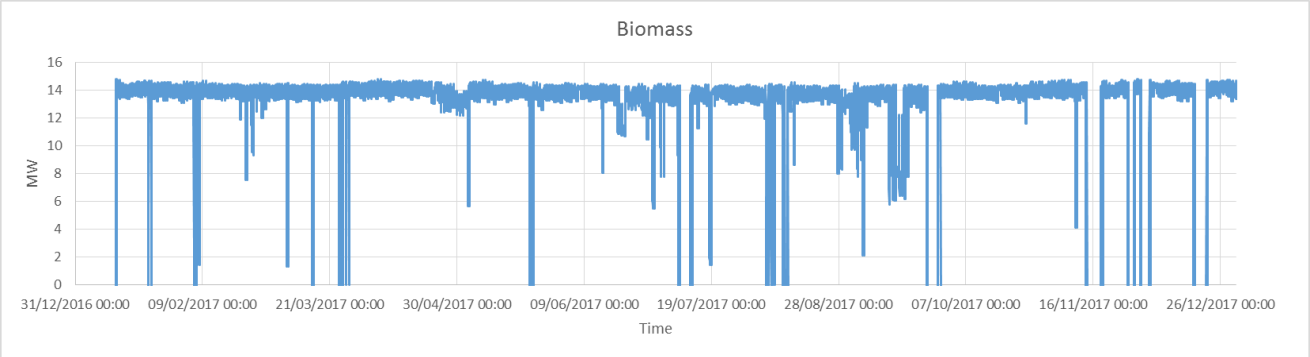


Figure 66: Biomass (MW)

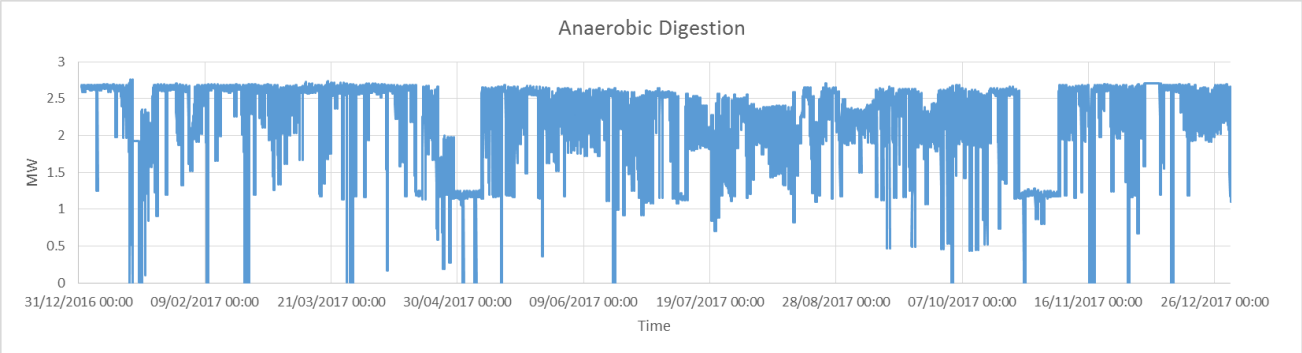


Figure 67: Anaerobic Digestion (MW)

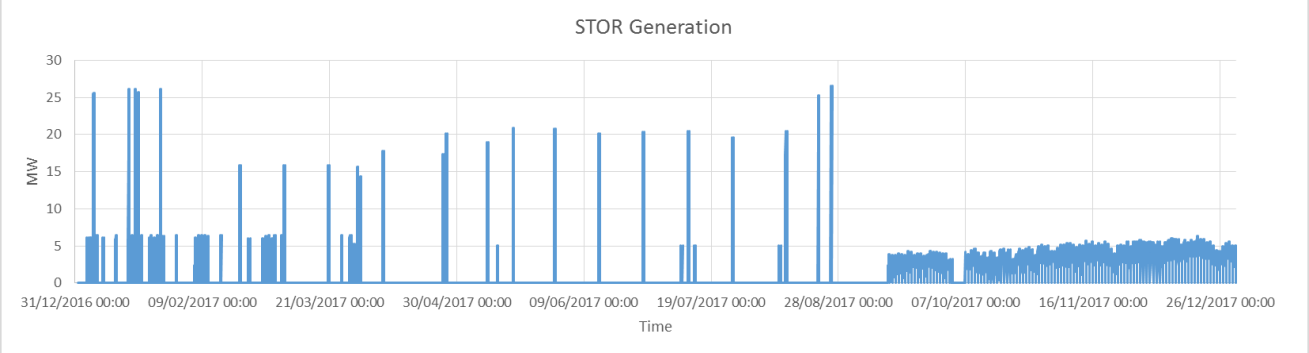


Figure 68: STOR Generation (MW)

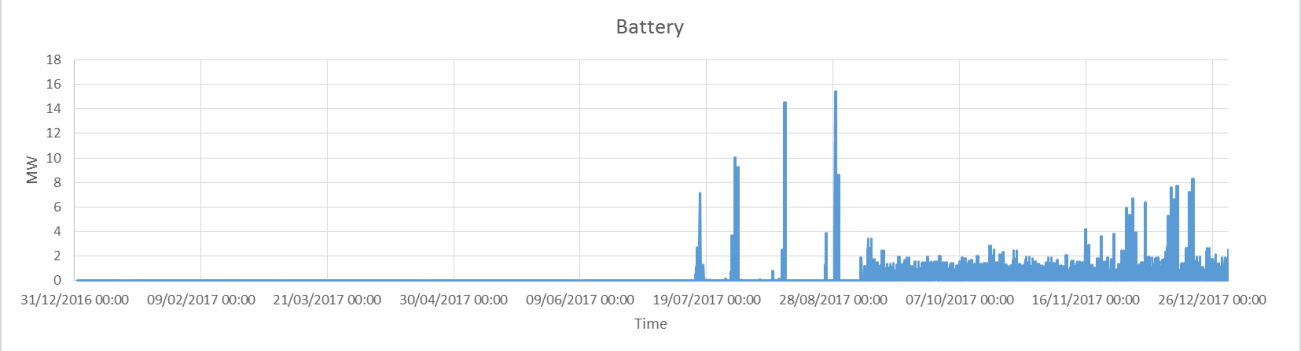


Figure 69: Battery (MW)

10. Appendix B: Example Forecasting Flow Charts

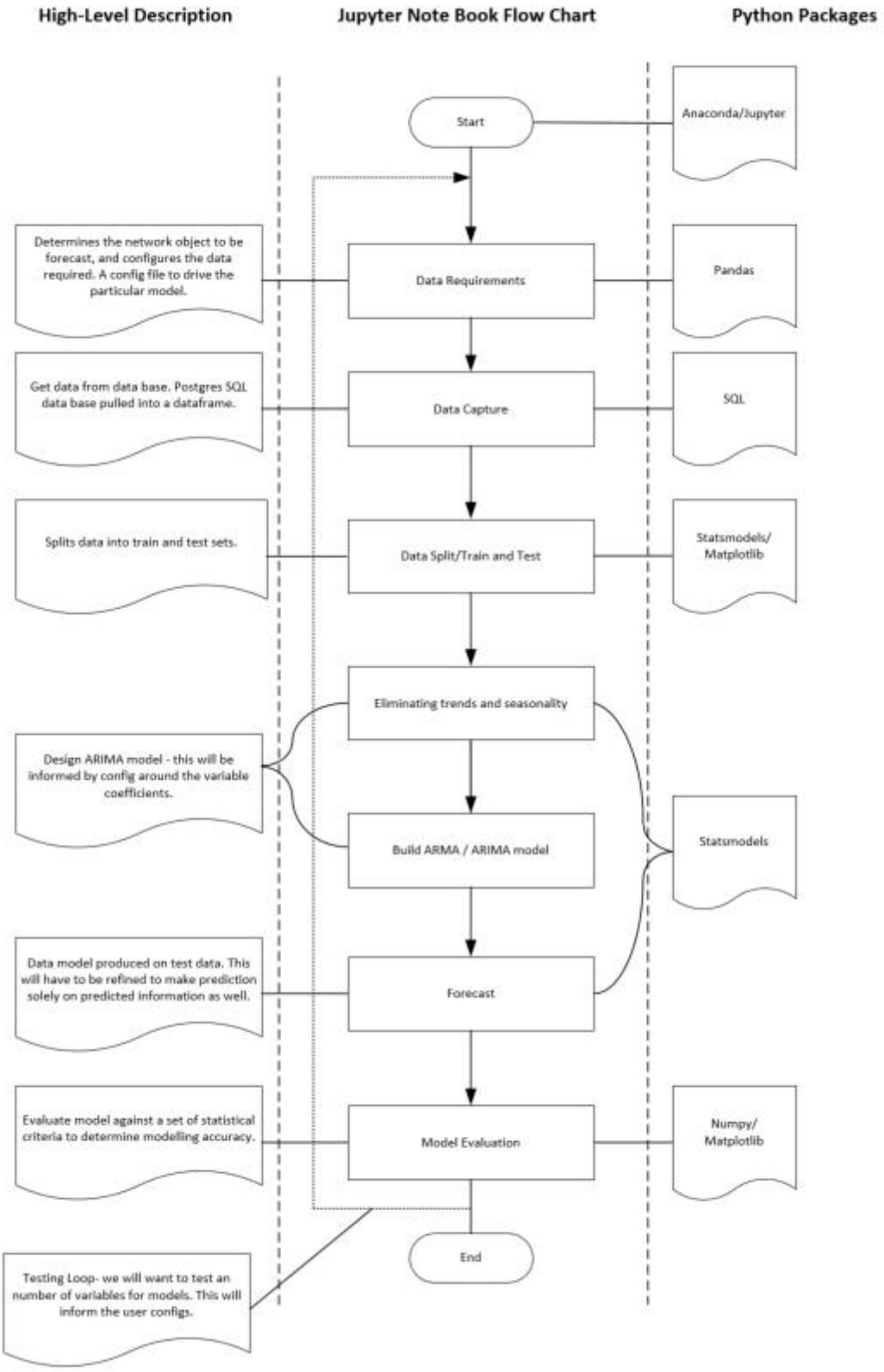


Figure 70: Forecasting for ARMA/ARIMA Model

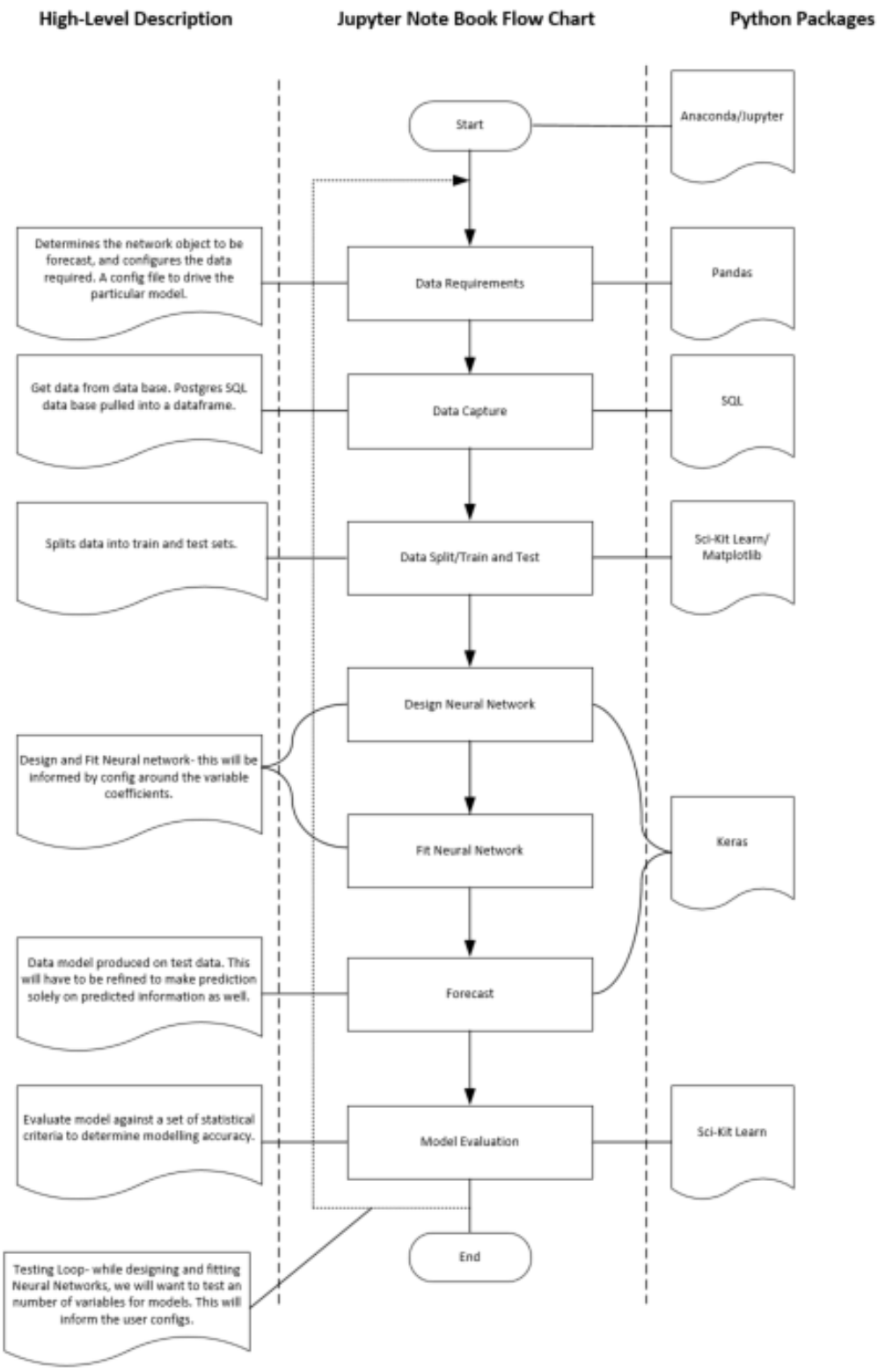


Figure 71: Forecasting for Neural Networks

11. Appendix C: Results Graphs

In following sections the graphical results are displayed for each time horizon and for each location/customer studied. The results show the dispersion of error (error in relation to the line of best fit), the histogram of error (frequency and magnitude of error), and the forecasted values against the actual values.

11.1. Indian Queens

11.1.1. Transformer 1

11.1.1.1. Six Months Ahead

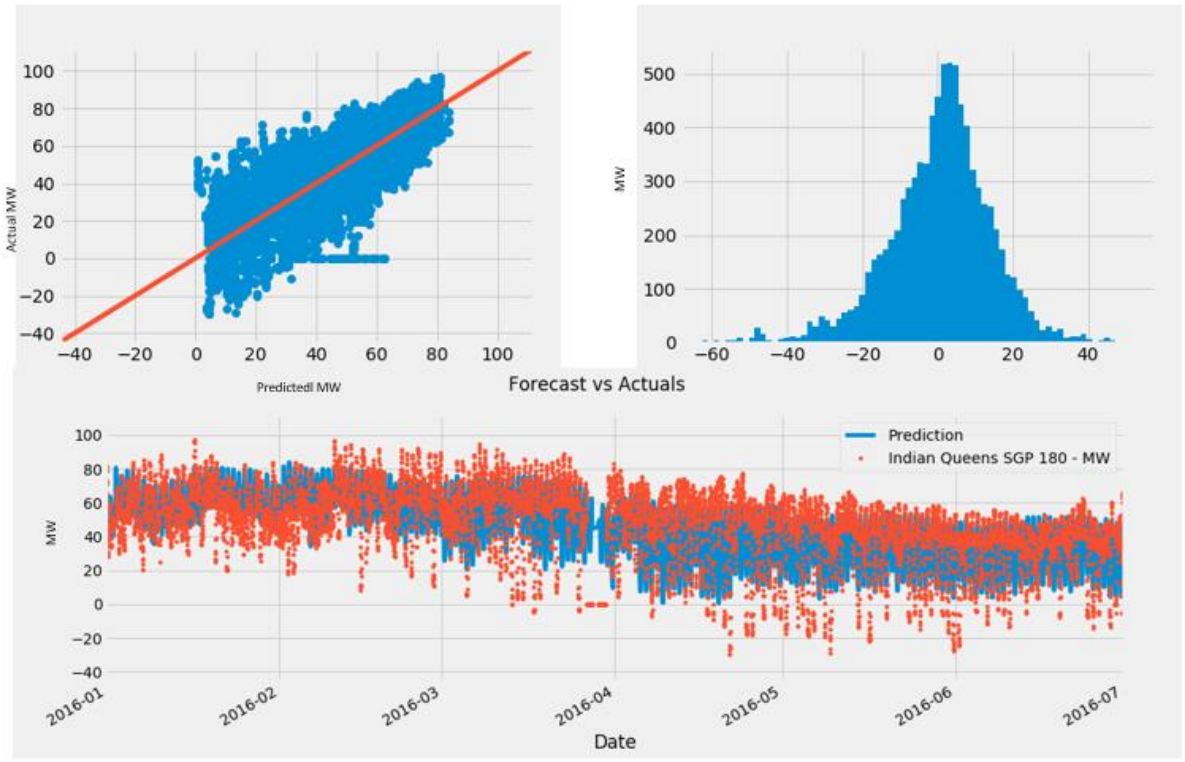


Figure 72: Six Month Ahead results for real power, January-June 2016.

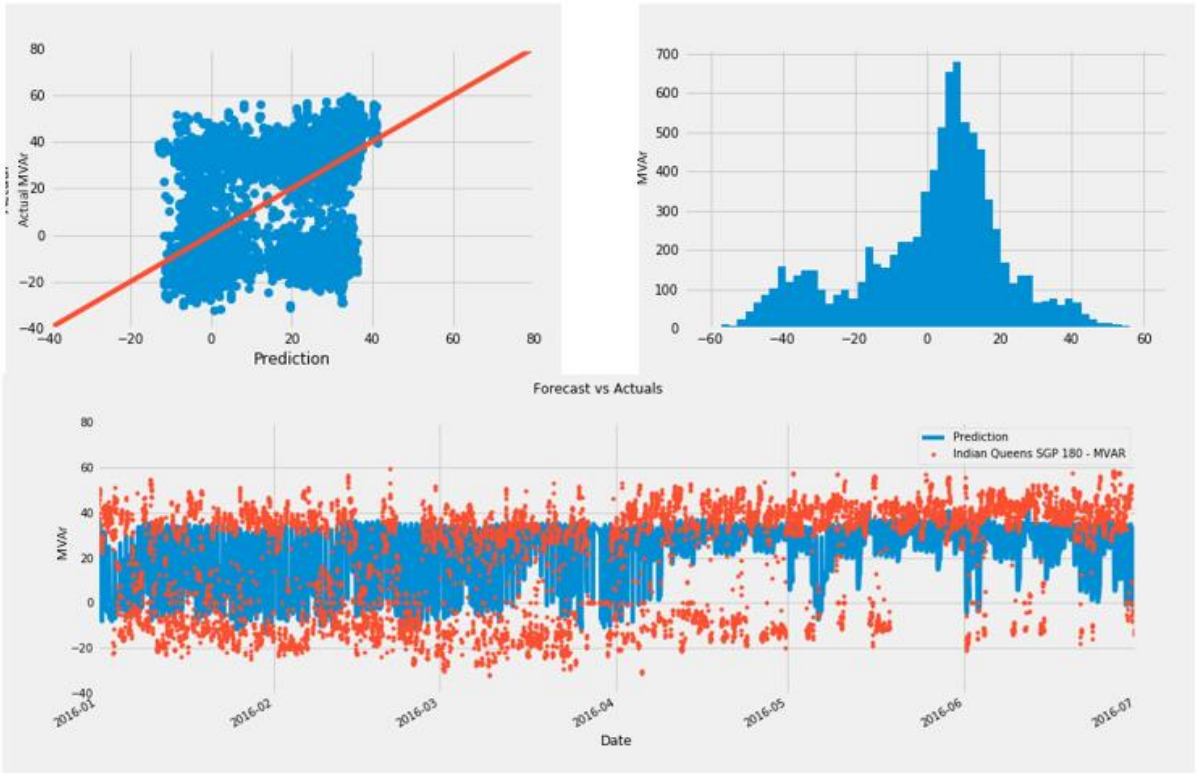


Figure 73: Six Month Ahead results for reactive power, January-June 2016.

11.1.1.2. Month Ahead

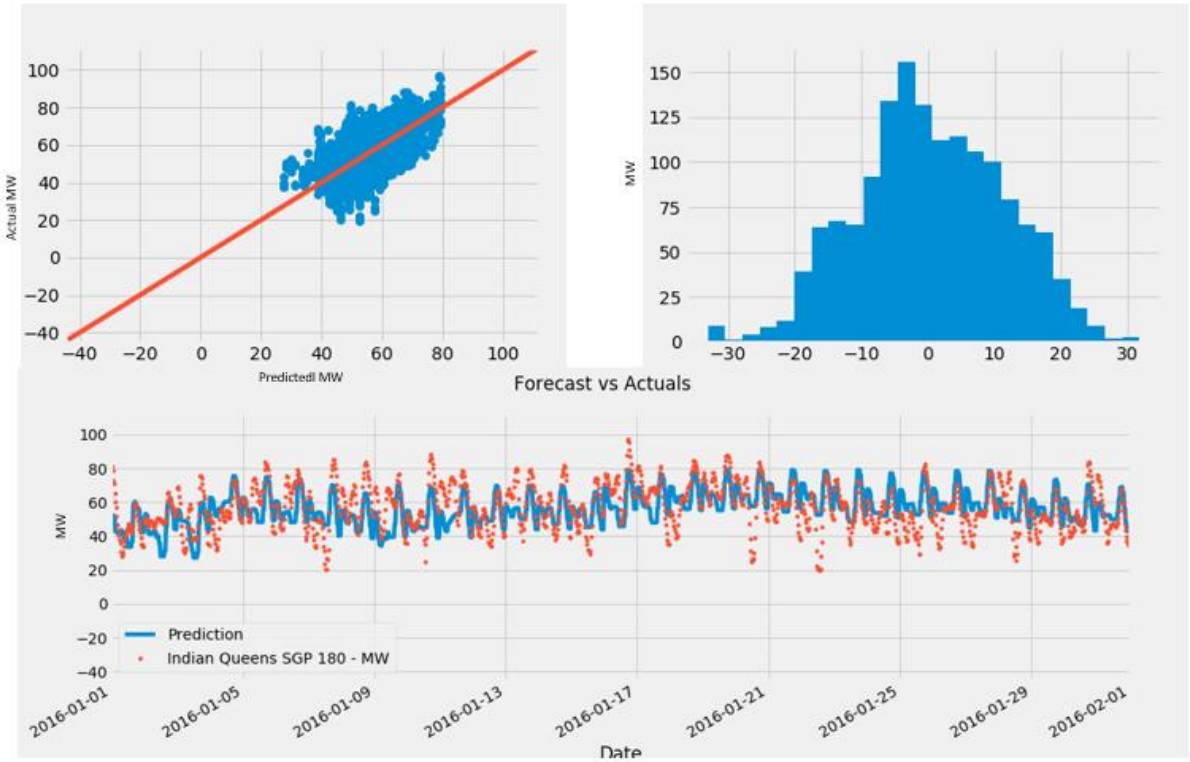


Figure 74: Month Ahead results for real power, January 2016.

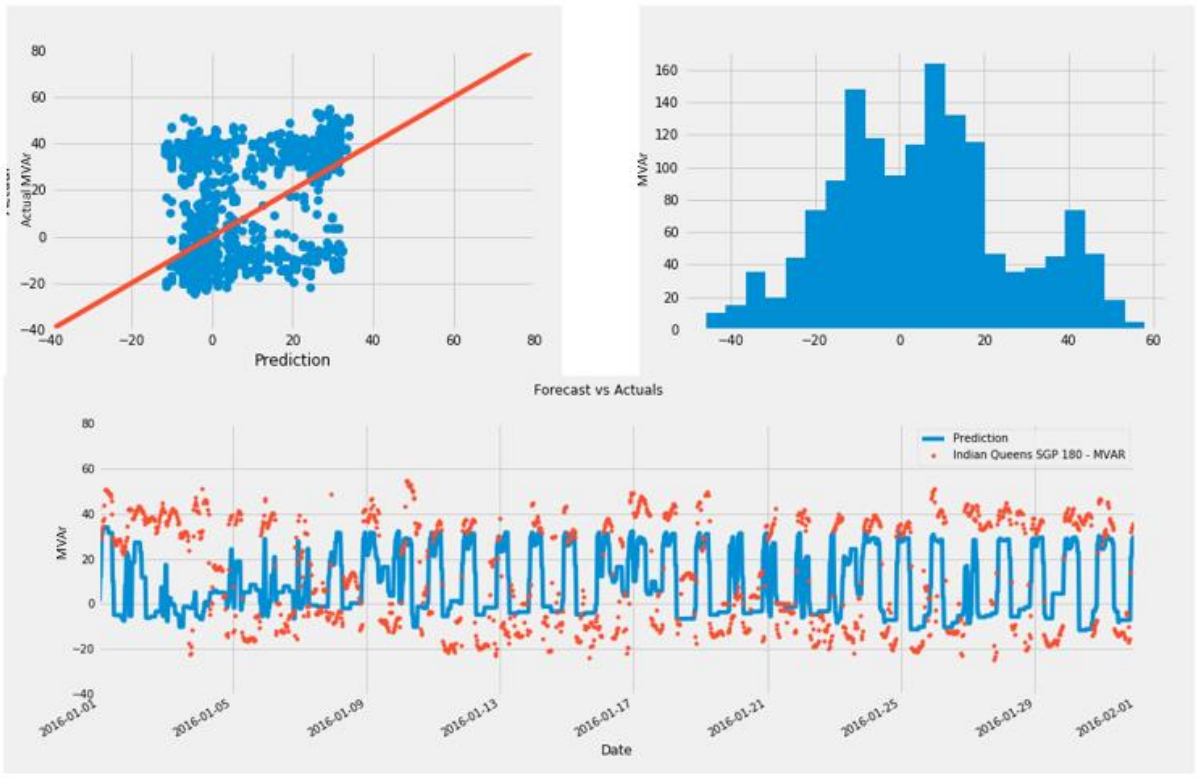


Figure 75: Month Ahead results for reactive power, January 2016.

11.1.1.3. Week Ahead

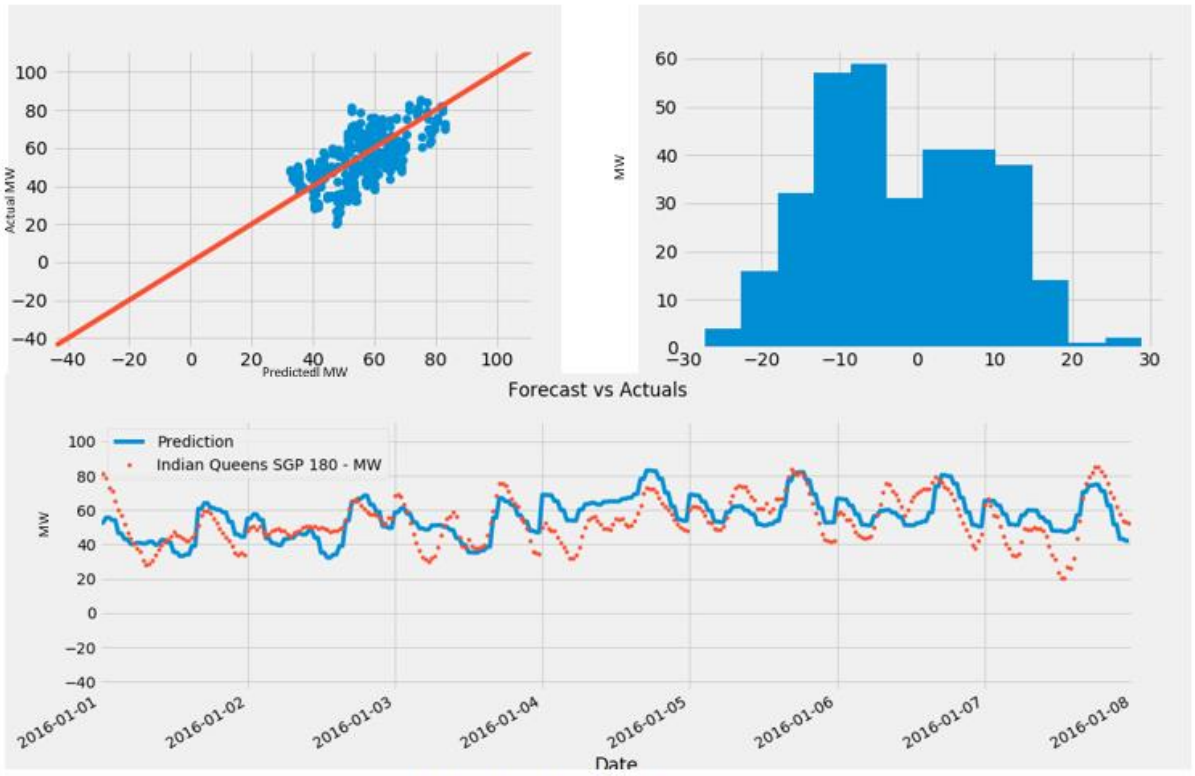


Figure 76: Week Ahead results for real power, 1-7 January 2016.

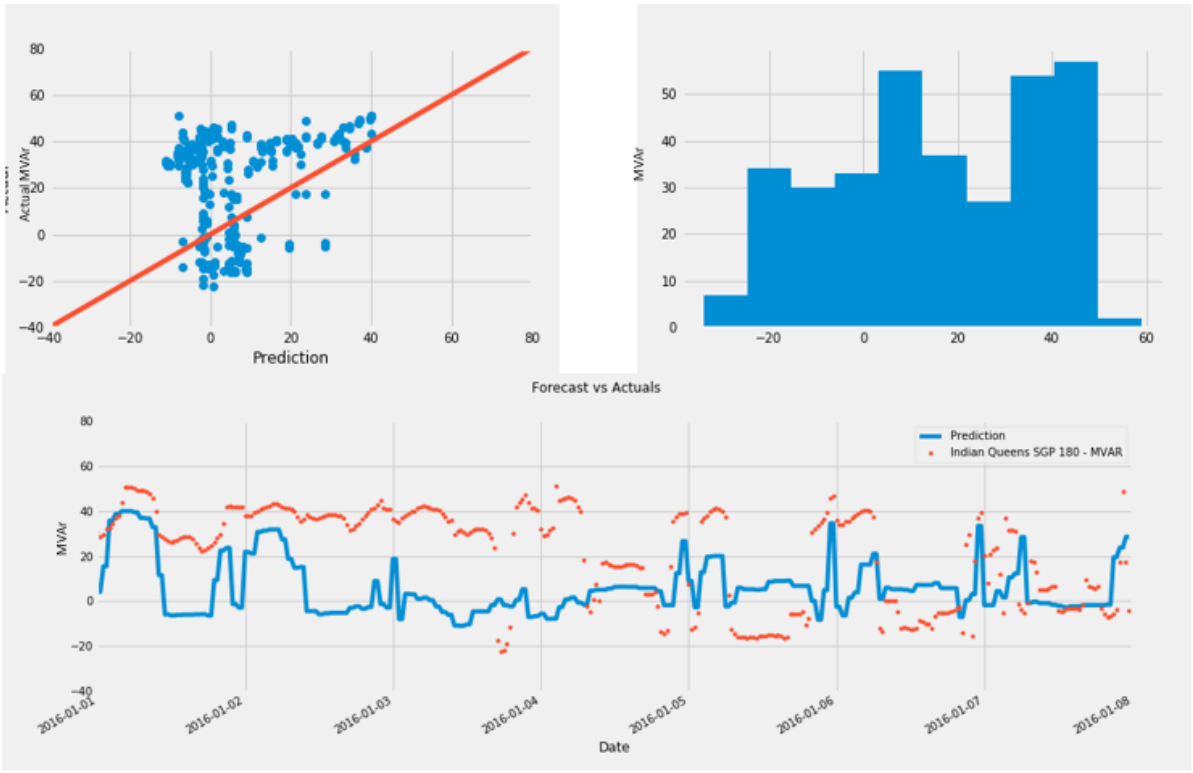


Figure 77: Week Ahead results for reactive power, 1-7 January 2016.

11.1.1.4. Day Ahead

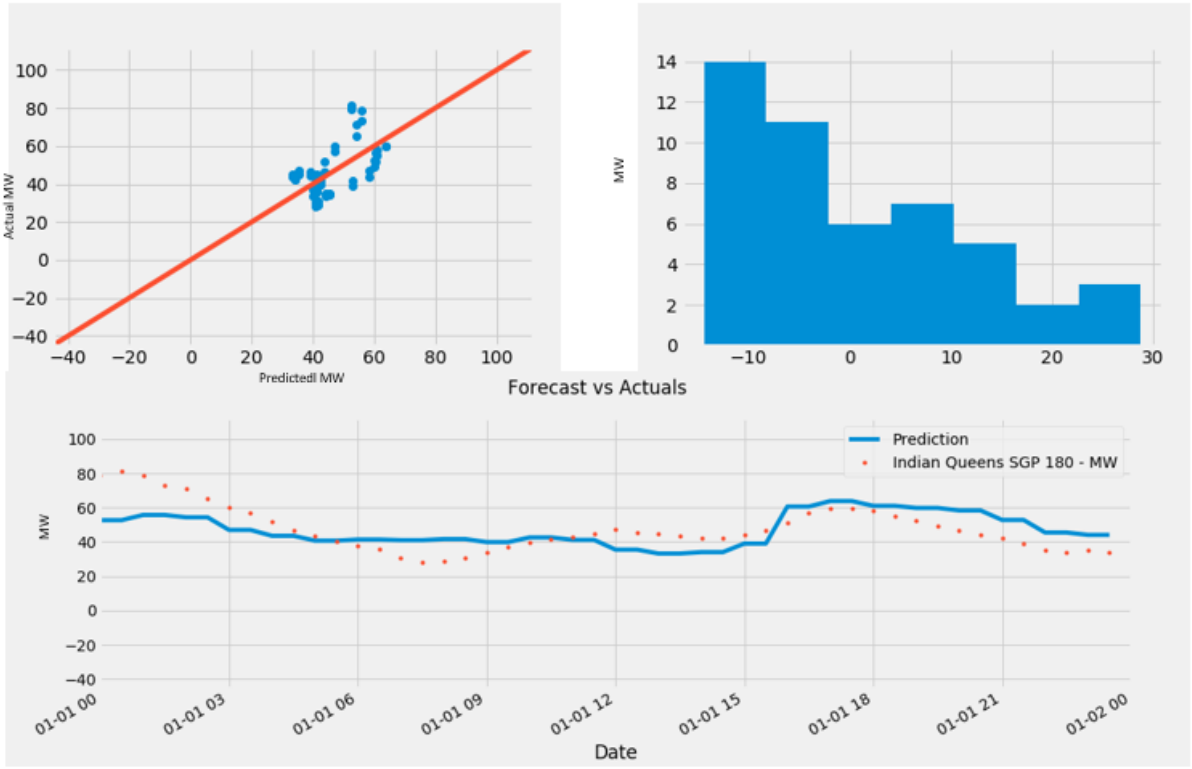


Figure 78: Day Ahead results for real power, 1st January 2016.

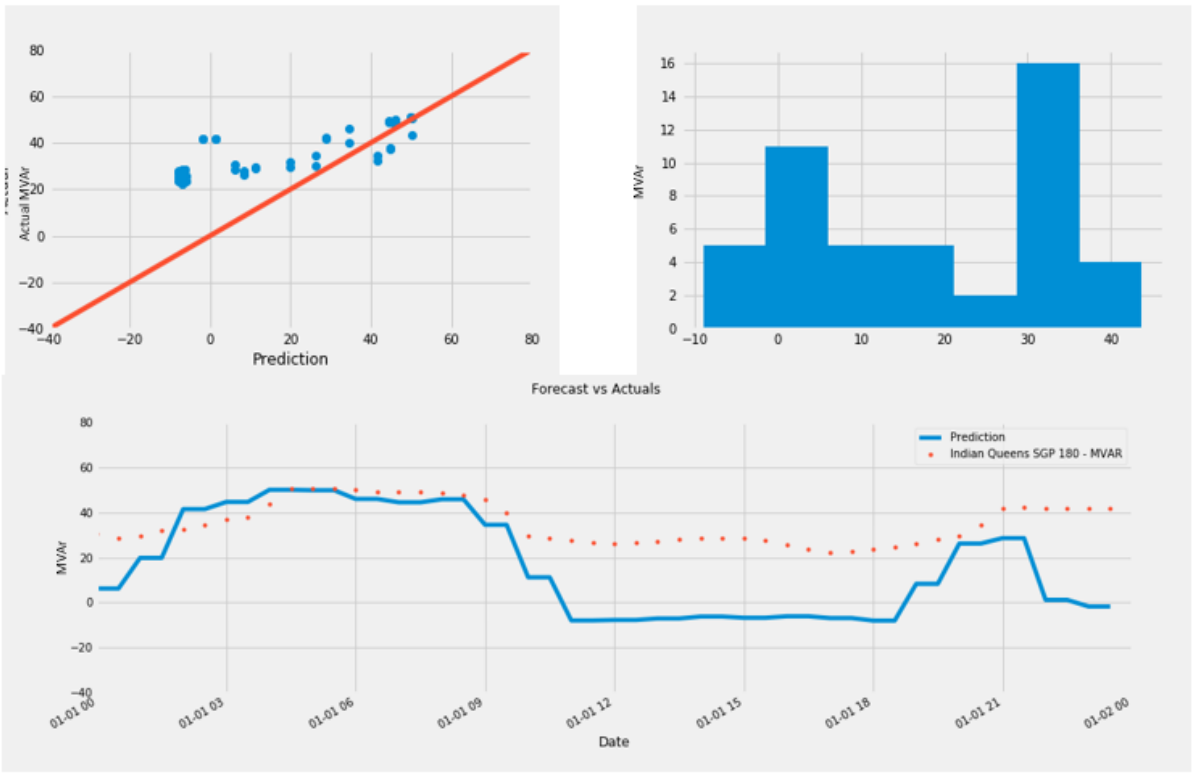


Figure 79: Day Ahead results for reactive power, 1st January 2016.

11.1.1.5. Hour Ahead

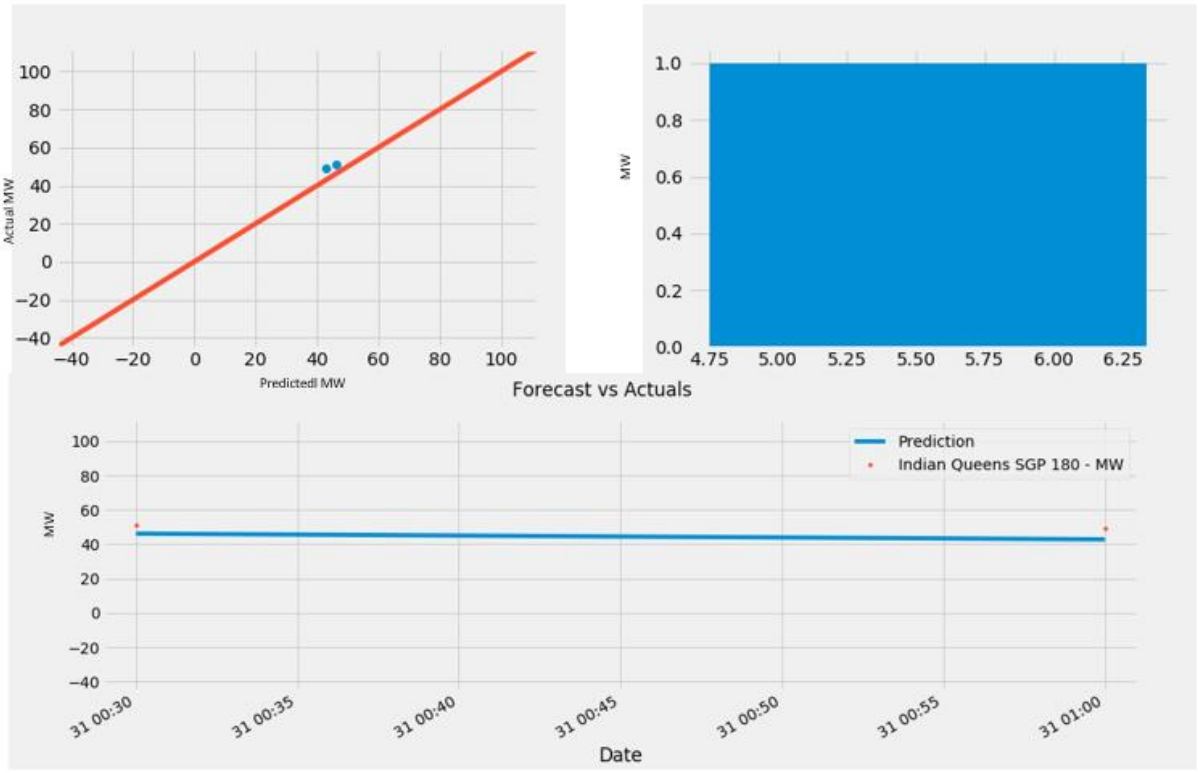


Figure 80: Hour Ahead results for real power, 31st December 2015.

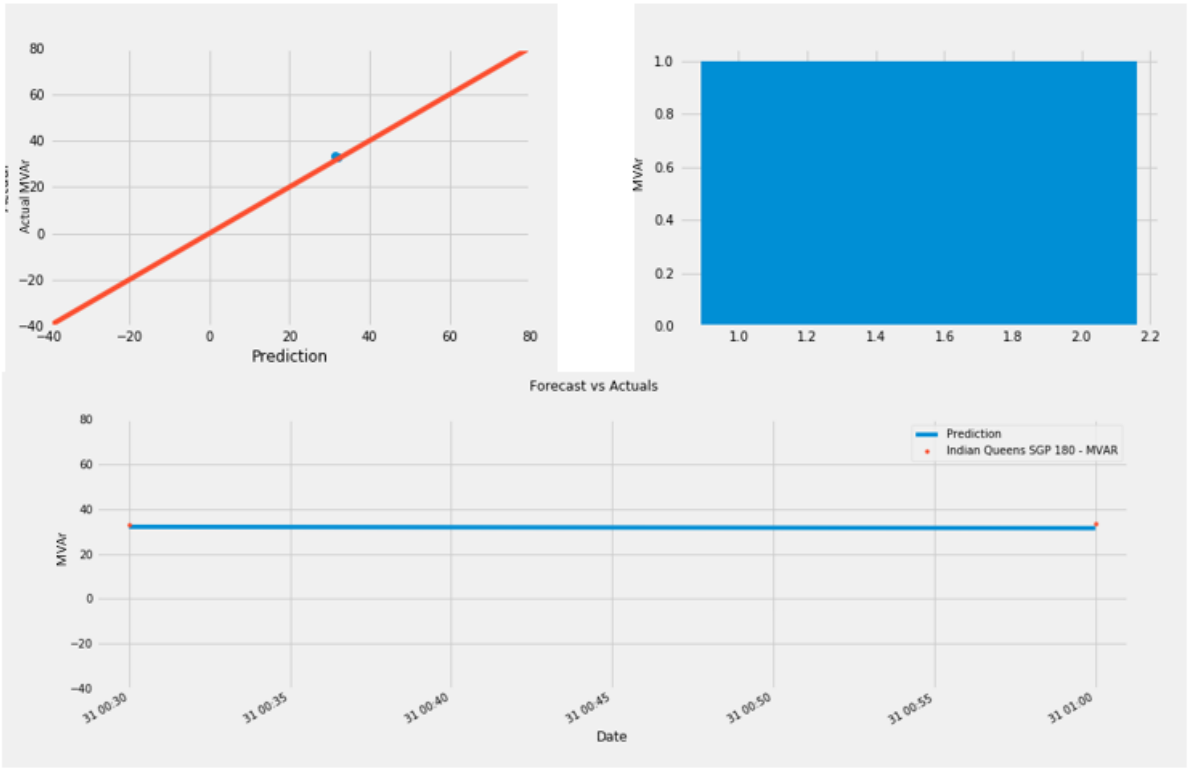


Figure 81: Hour Ahead results for reactive power, 31st December 2015.

11.1.2. Transformer 2

11.1.2.1. Six Months Ahead

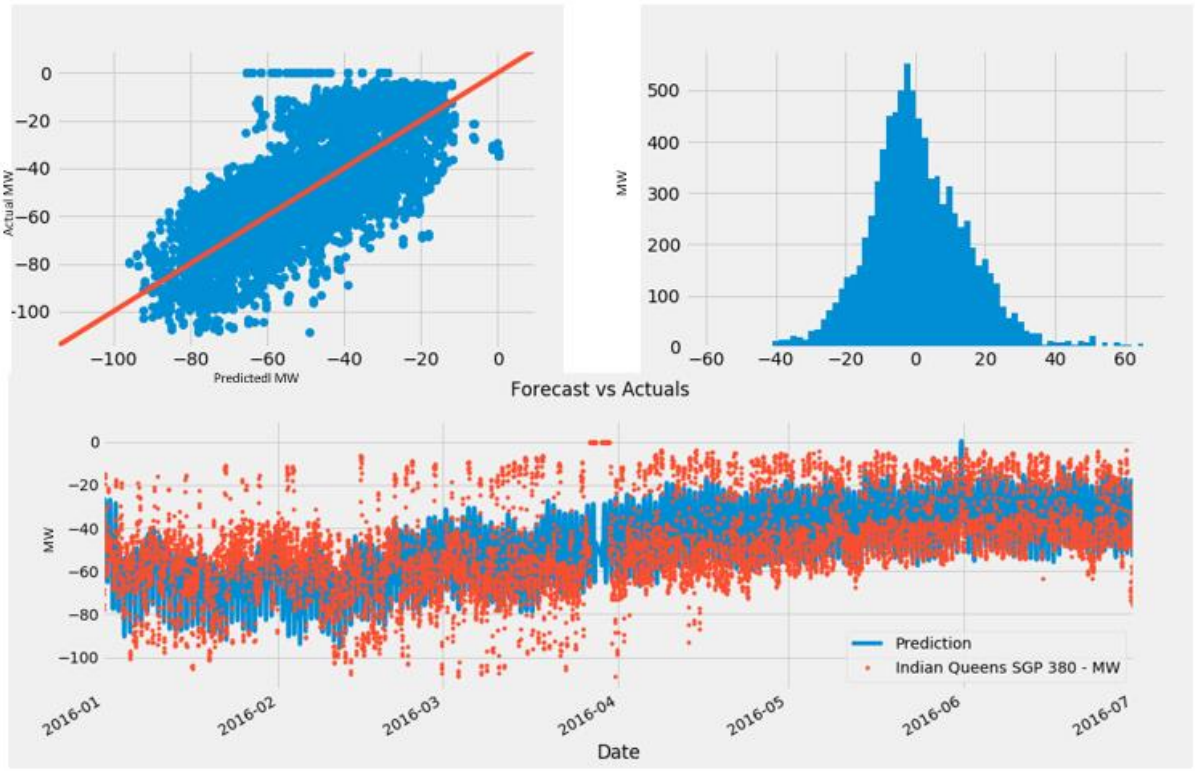


Figure 82: Six Month Ahead results for real power, January-June 2016.

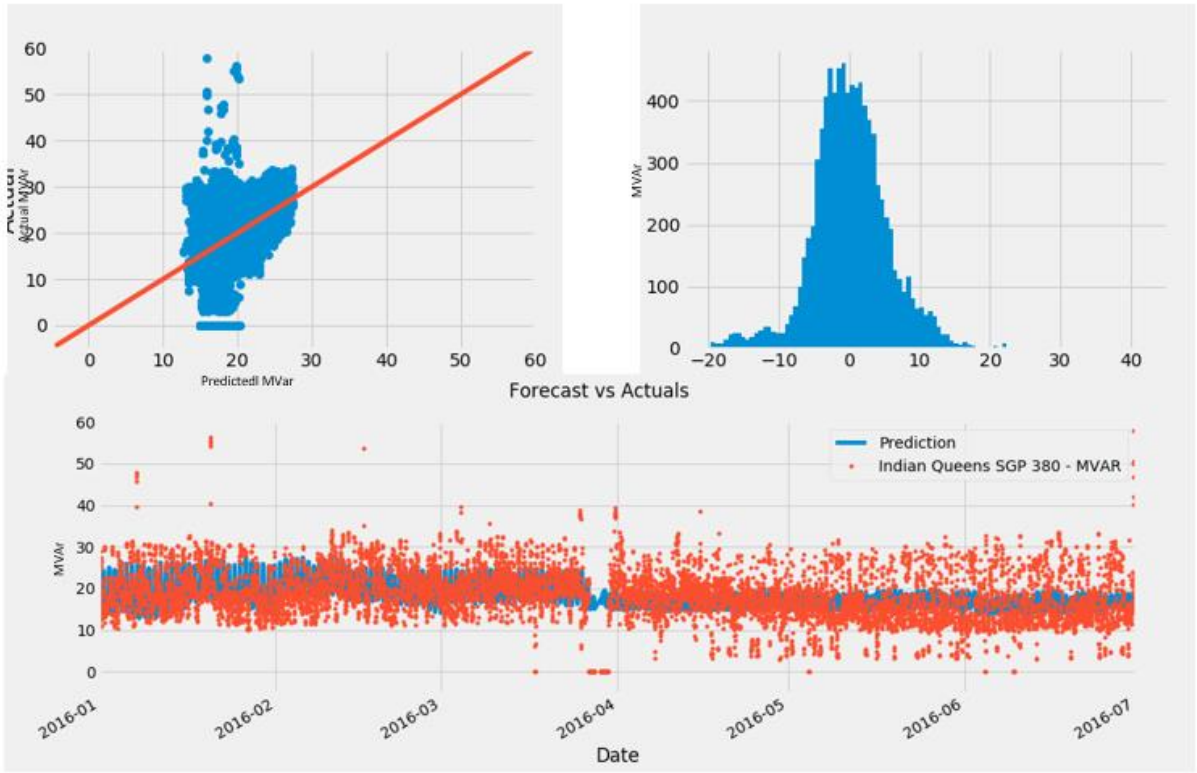


Figure 83: Six Month Ahead results for reactive power, January-June 2016.

11.1.2.2. Month Ahead

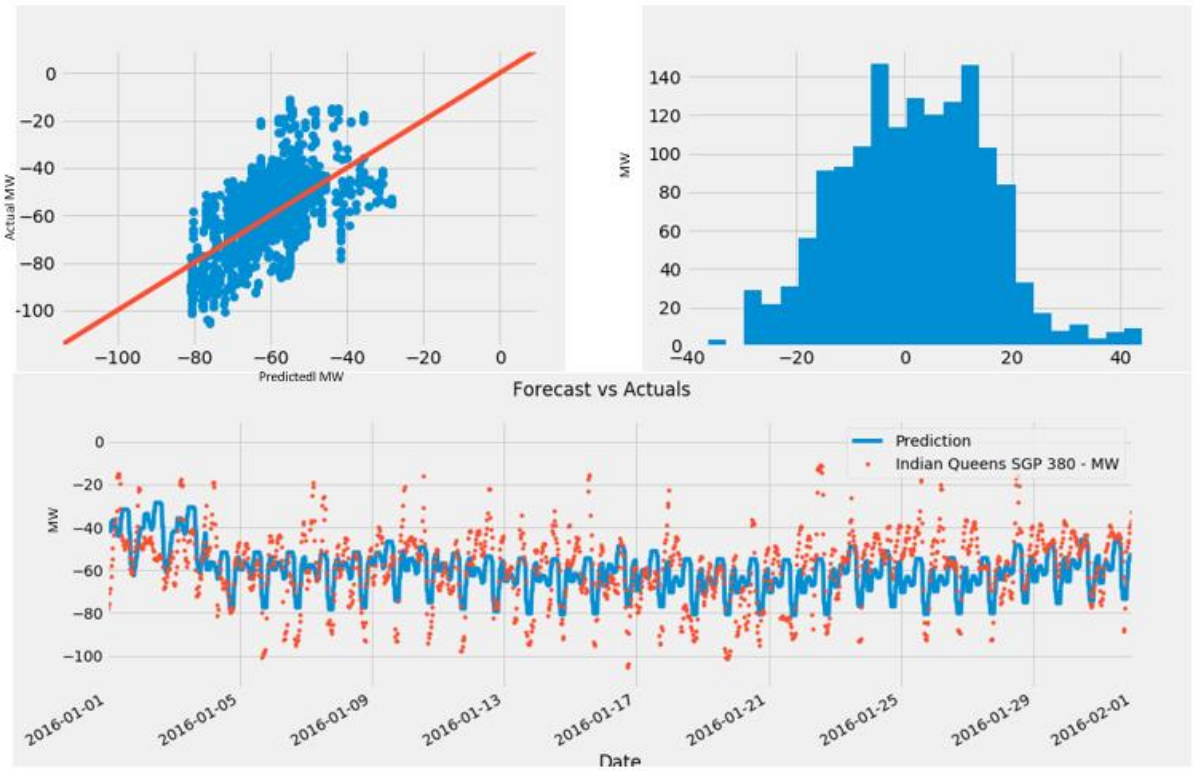


Figure 84: Month Ahead results for real power, January 2016.

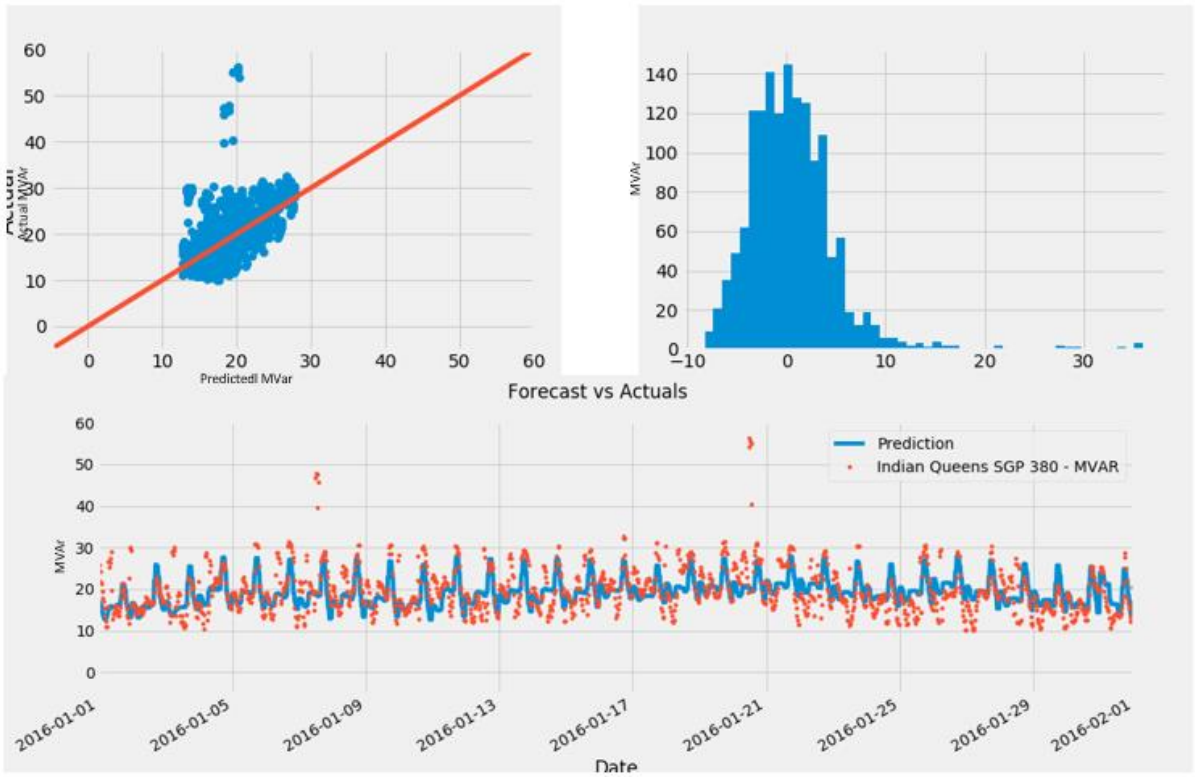


Figure 85: Month Ahead results for reactive power, January 2016.

11.1.2.3. Week Ahead

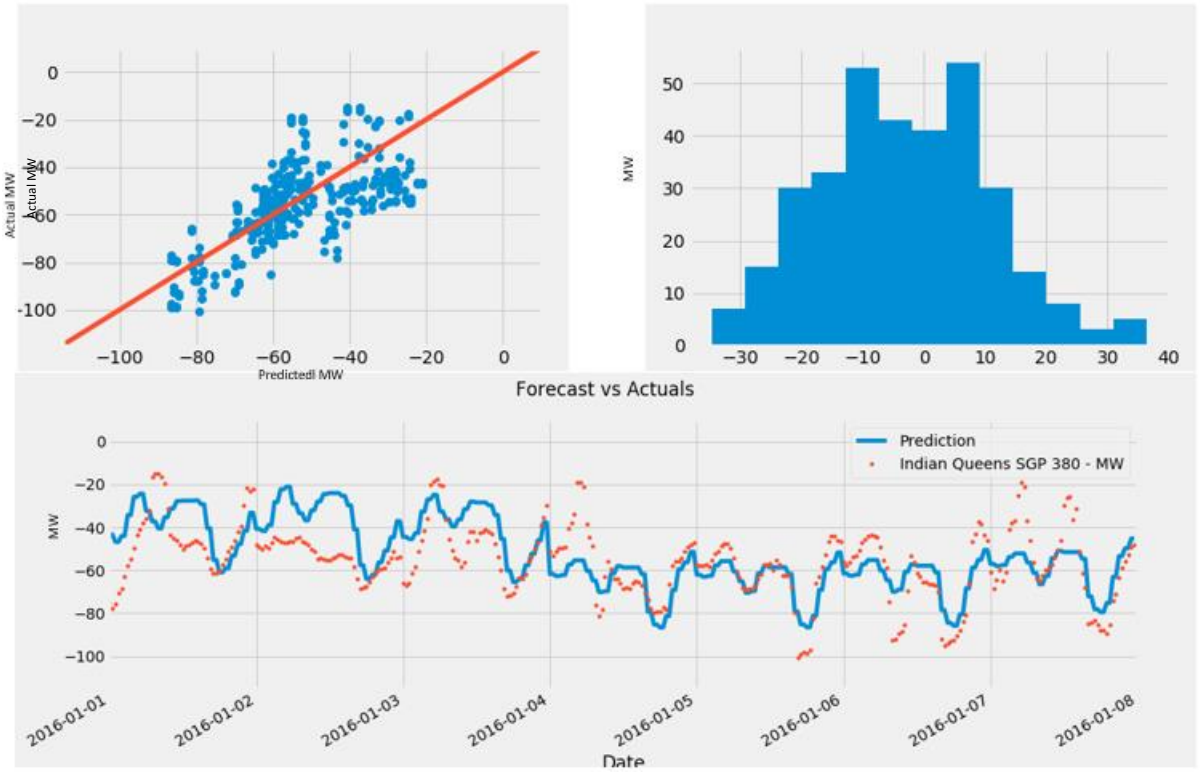


Figure 86: Week Ahead results for real power, 1-7 January 2016.

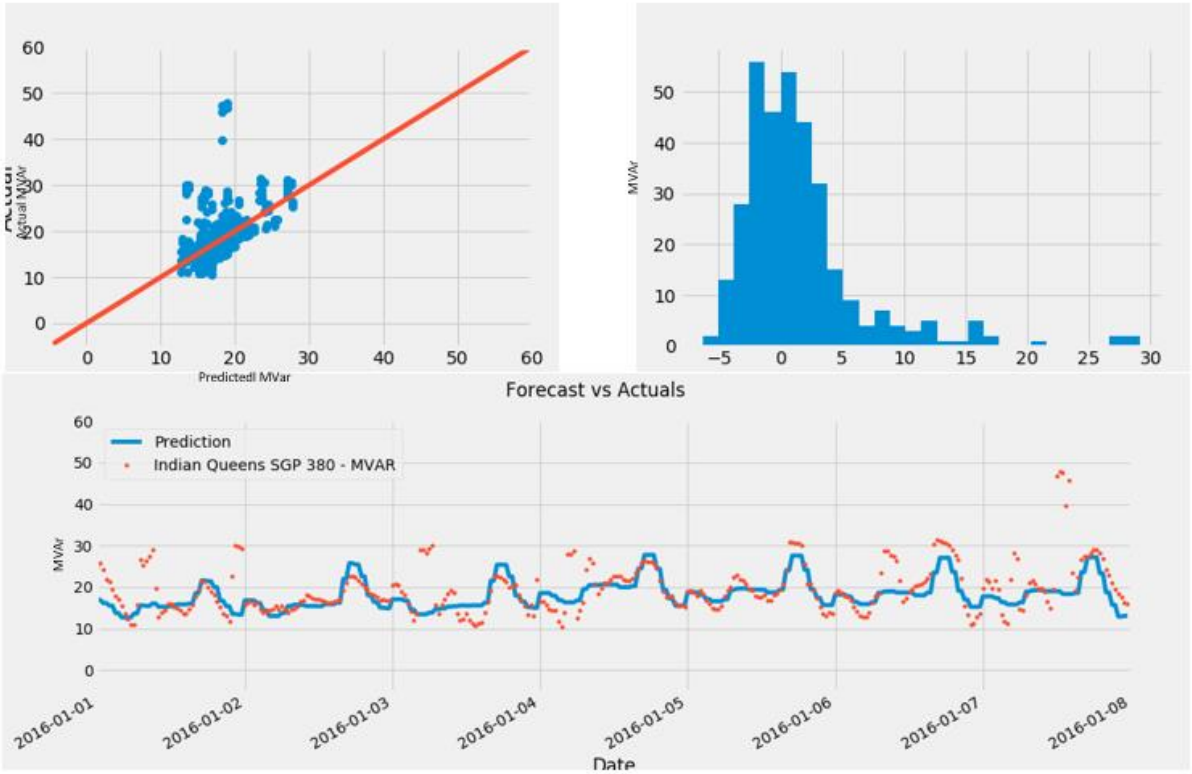


Figure 87: Week Ahead results for reactive power, 1-7 January 2016.

11.1.2.4. Day Ahead

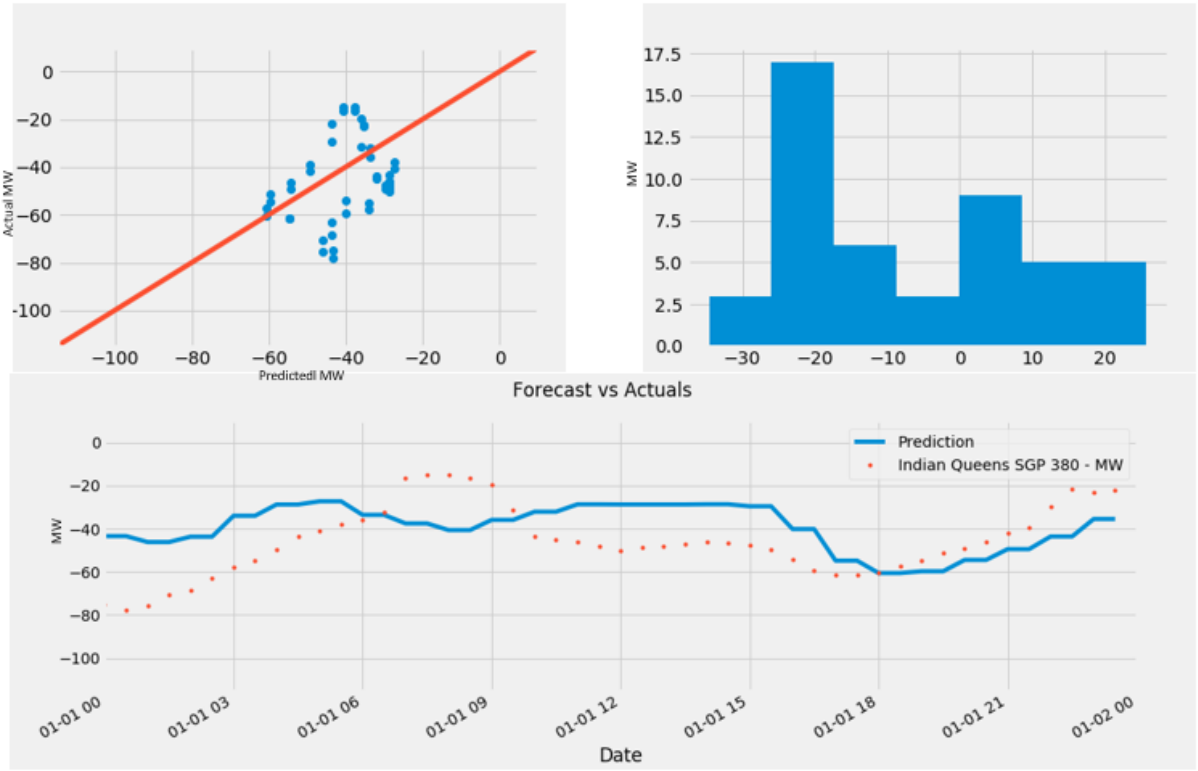


Figure 88: Day Ahead results for real power, 1st January 2016.

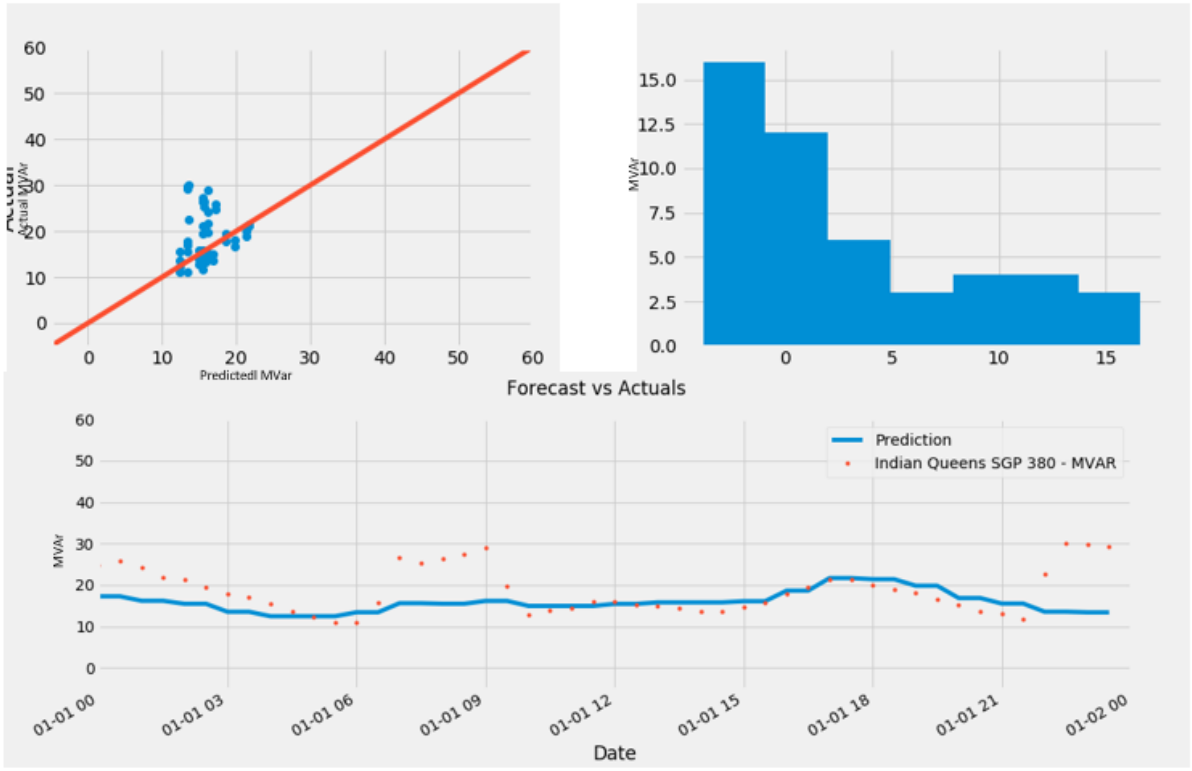


Figure 89: Day Ahead results for reactive power, 1st January 2016.

11.1.2.5. Hour Ahead

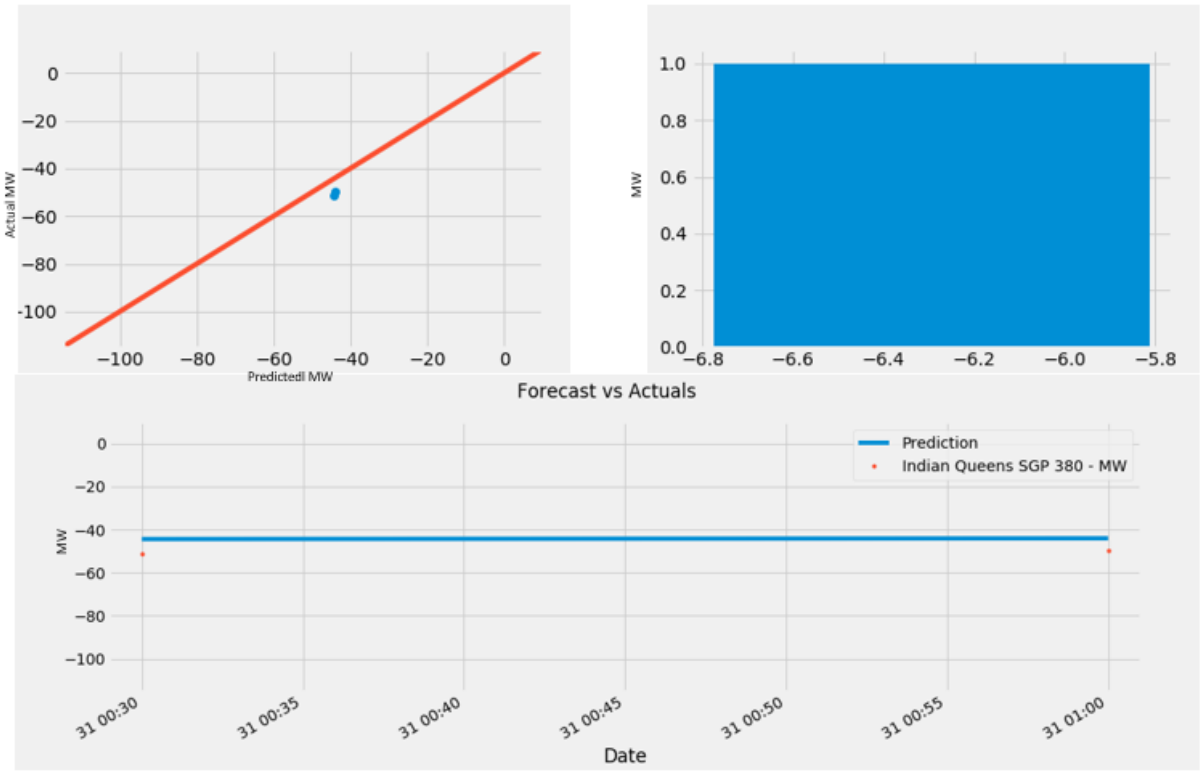


Figure 90: Hour Ahead results for real power, 31st December 2015.

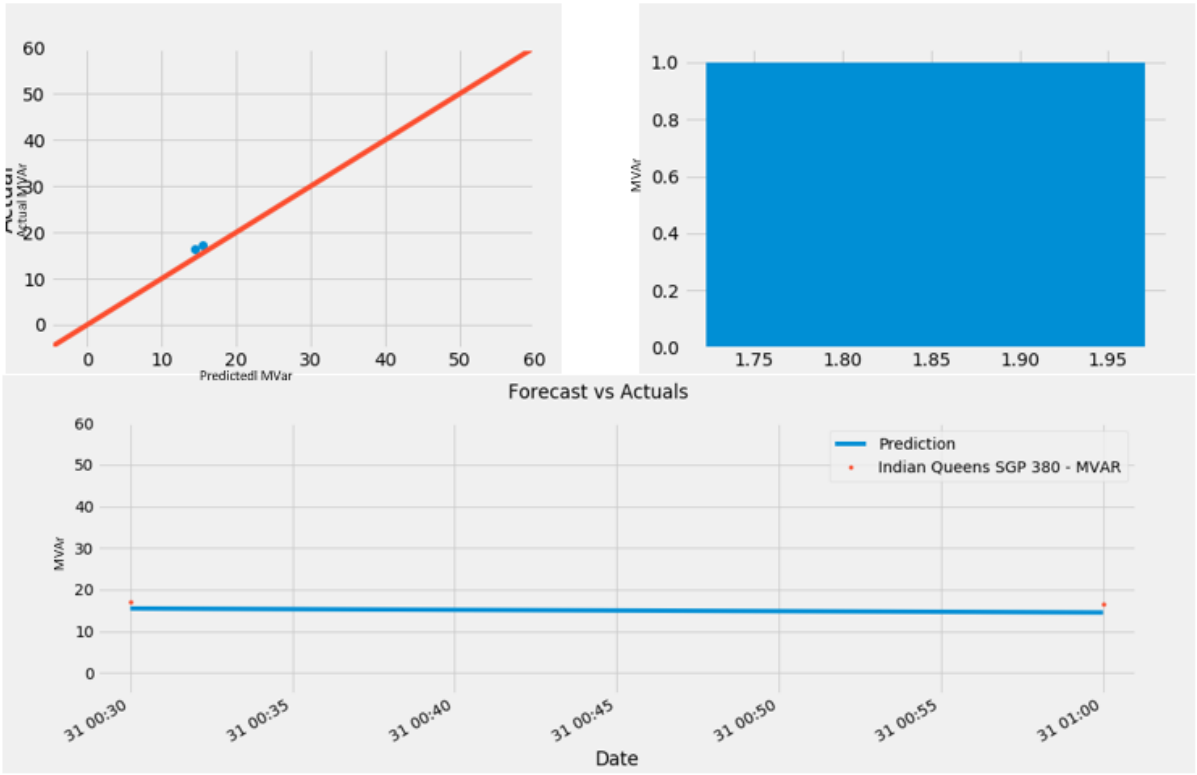


Figure 91: Hour Ahead results for reactive power, 31st December 2015.

11.1.3. Transformer 3

11.1.3.1. Six Months Ahead

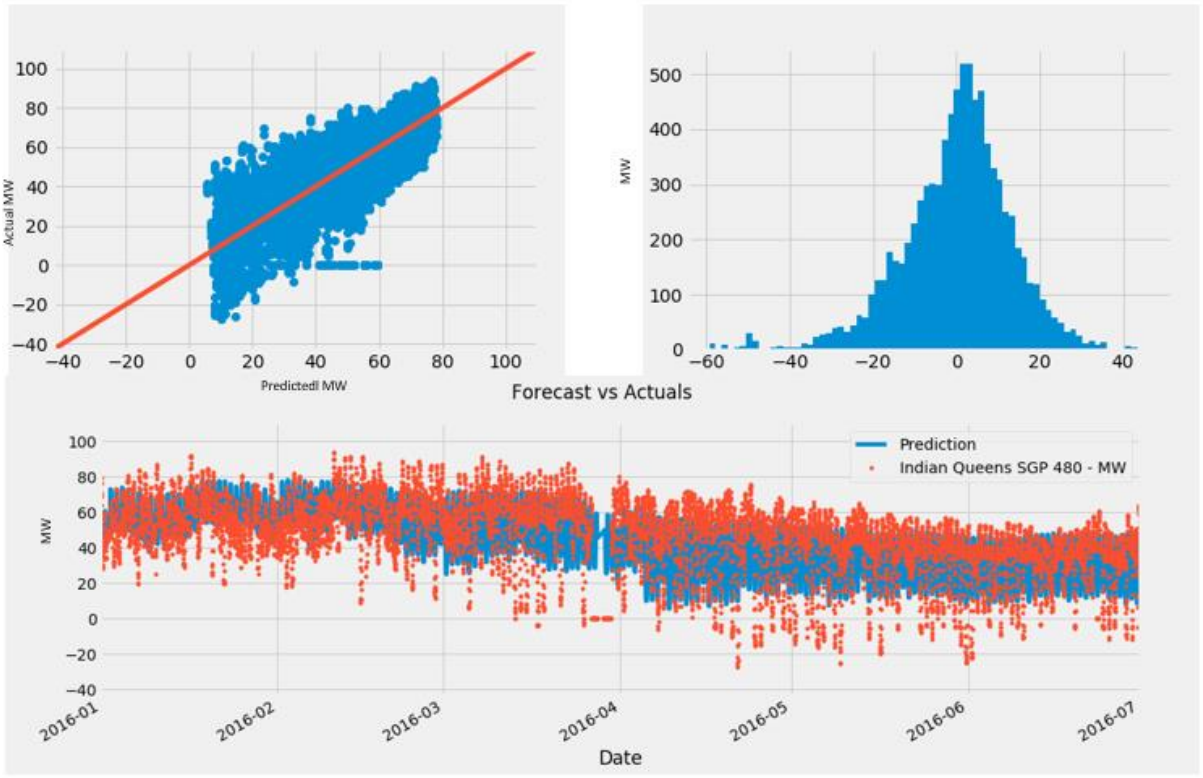


Figure 92: Six Month Ahead results for real power, January-June 2016.

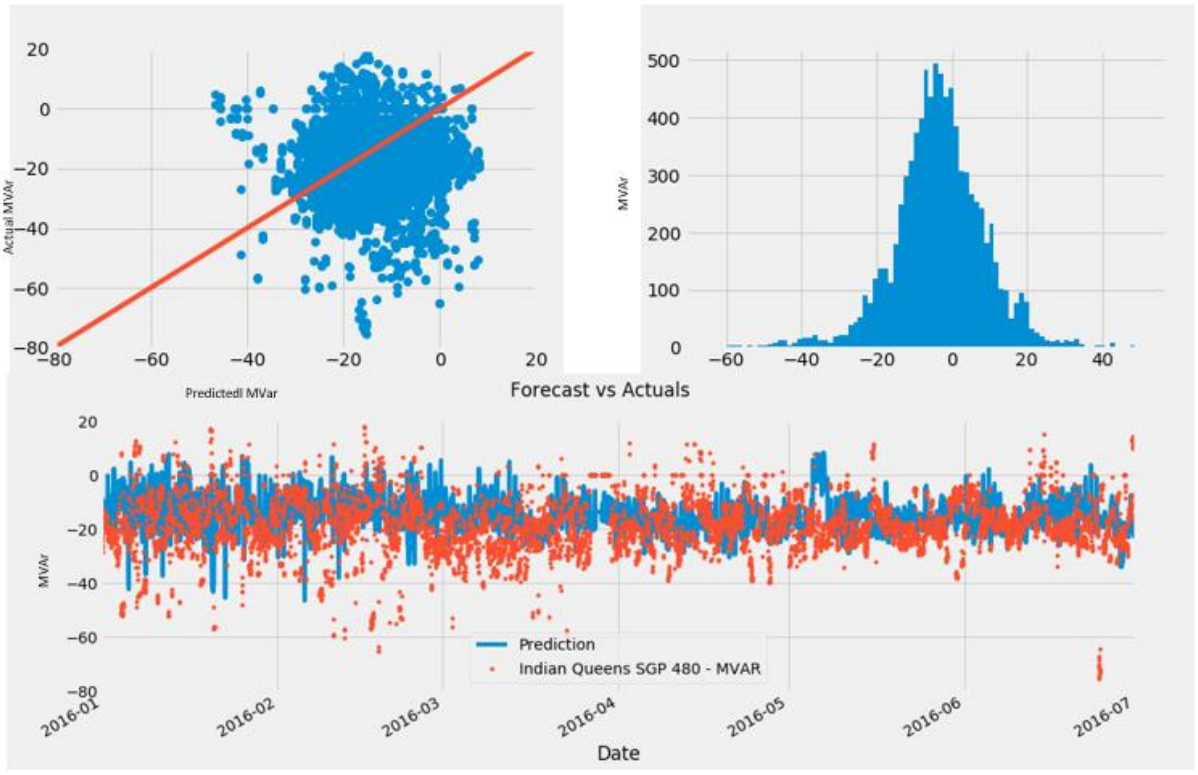


Figure 93: Six Month Ahead results for reactive power, January-June 2016.

11.1.3.2. Month Ahead

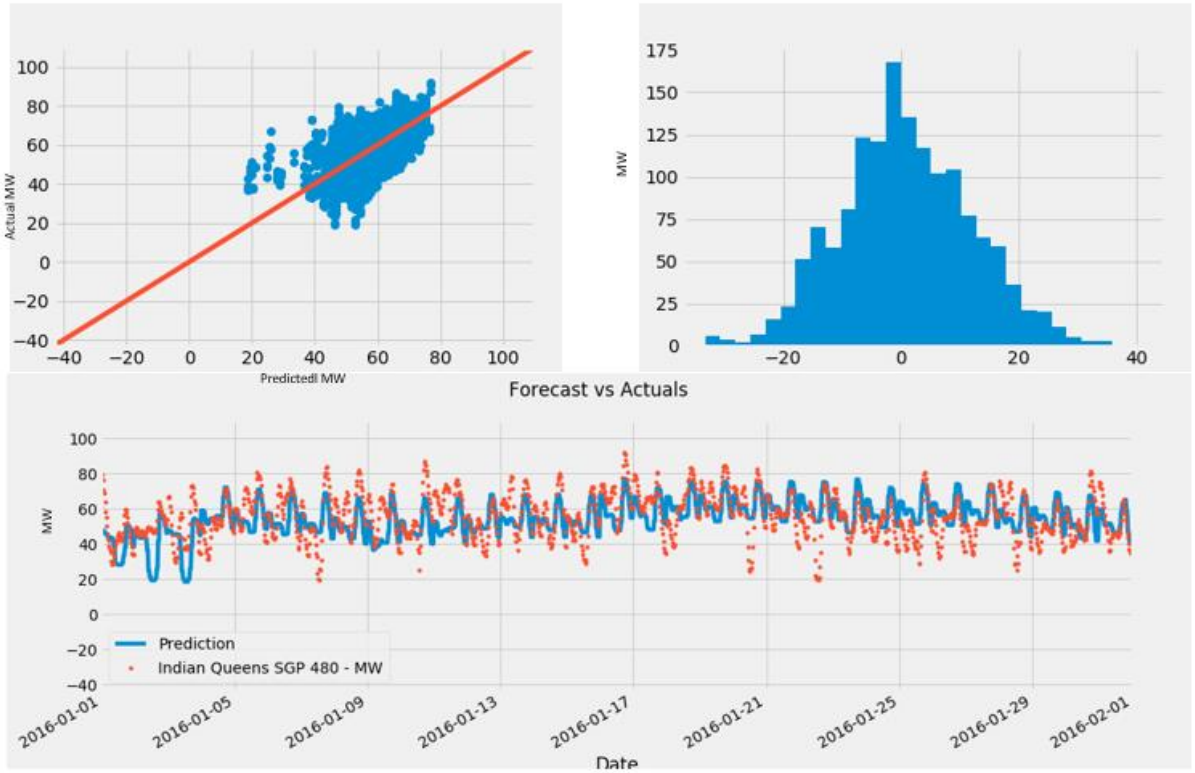


Figure 94: Month Ahead results for real power, January 2016.

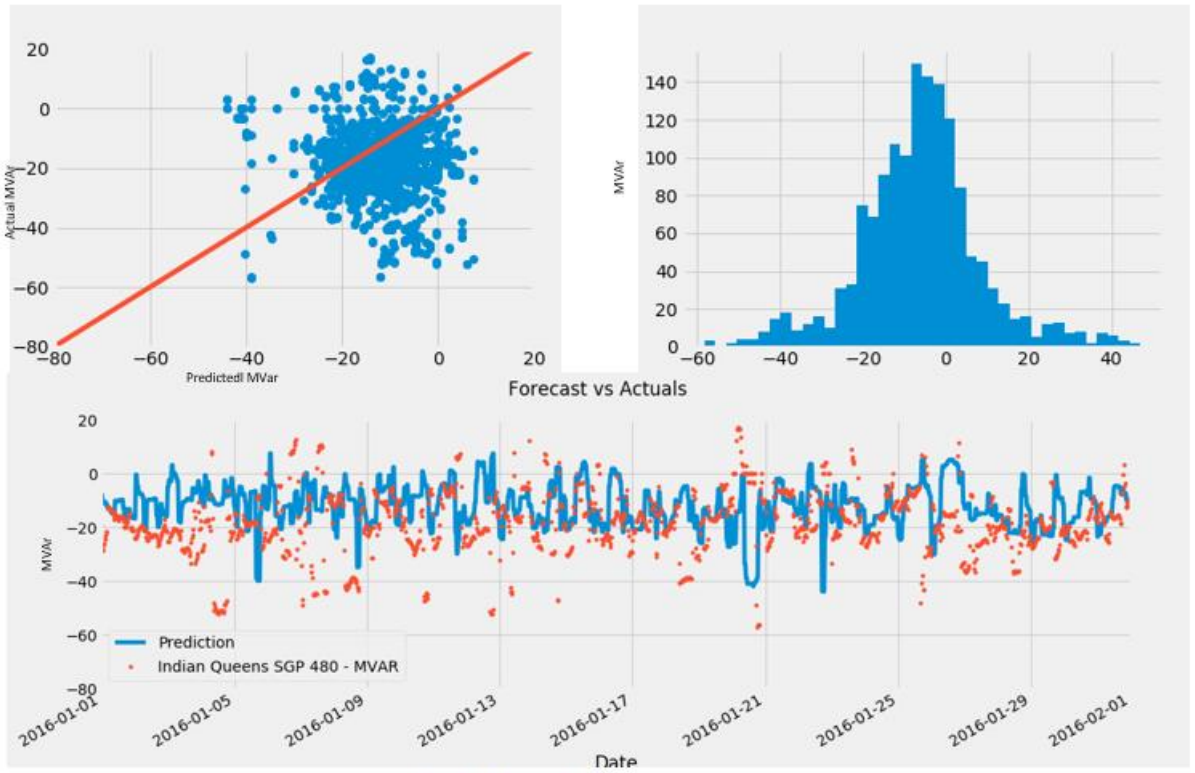


Figure 95: Month Ahead results for reactive power, January 2016.

11.1.3.3. Week Ahead

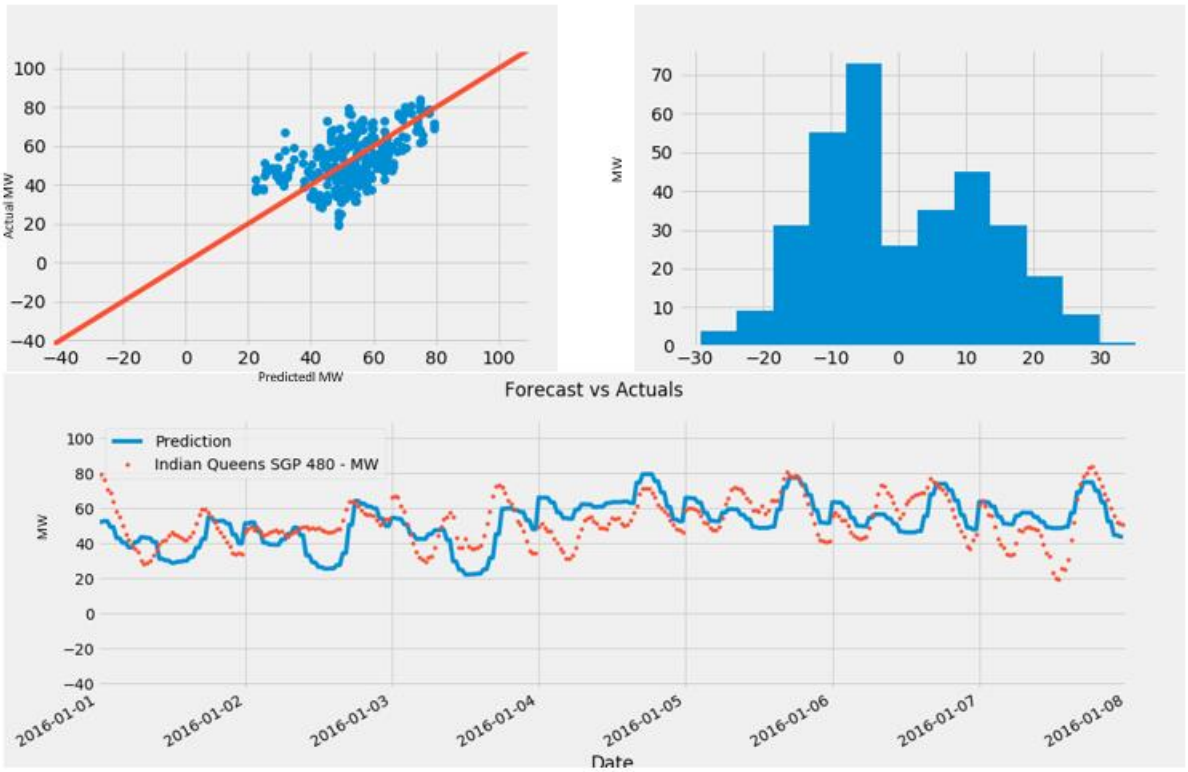


Figure 96: Week Ahead results for real power, 1-7 January 2016.

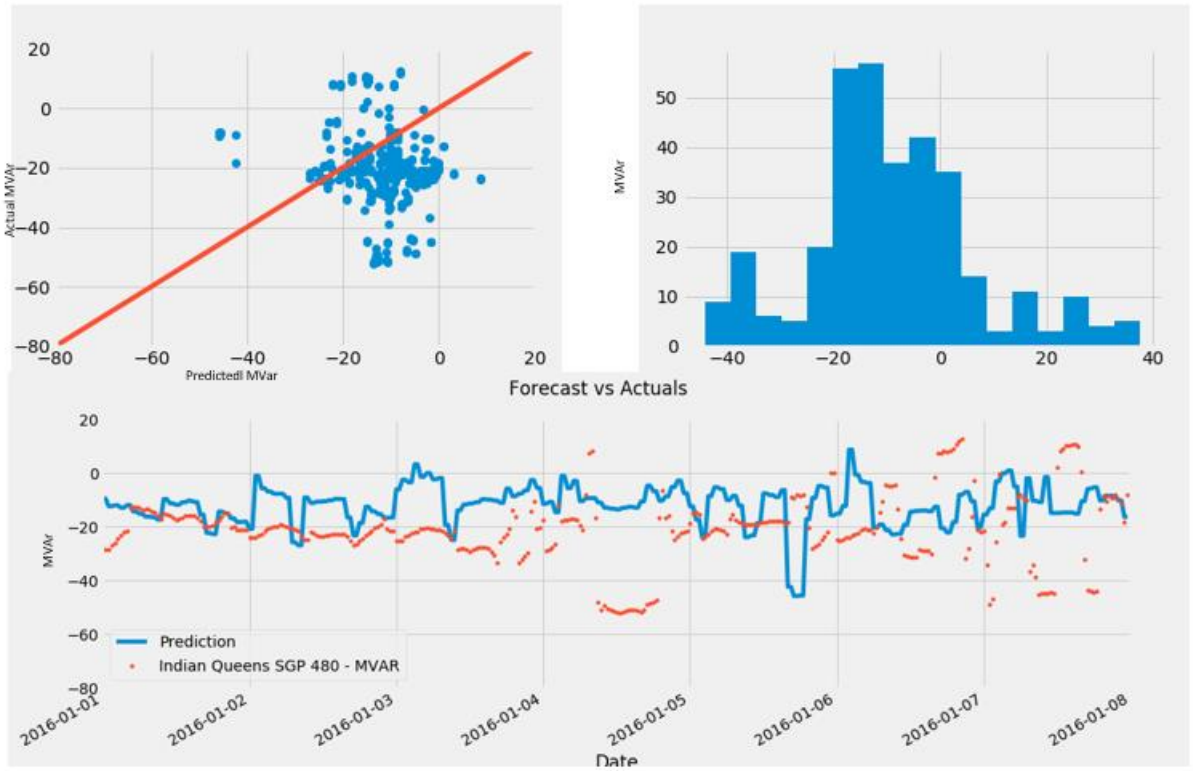


Figure 97: Week Ahead results for reactive power, 1-7 January 2016.

11.1.3.4. Day Ahead

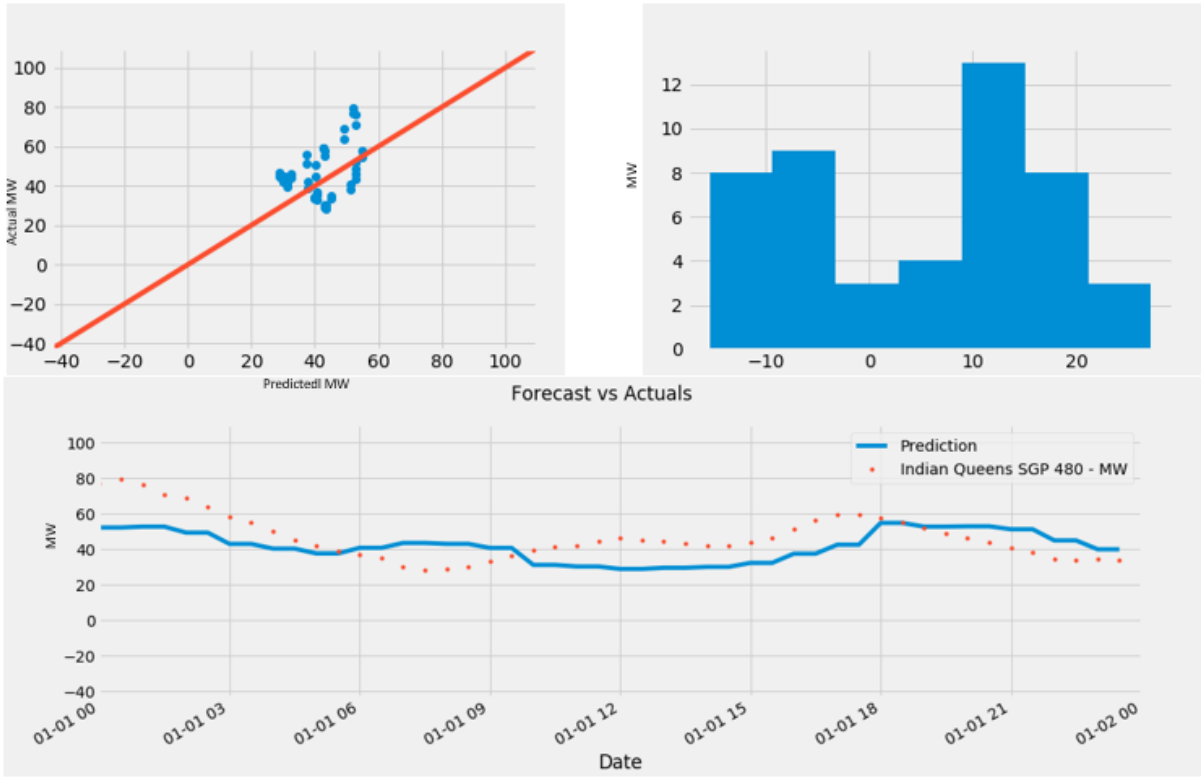


Figure 98: Day Ahead results for real power, 1st January 2016.

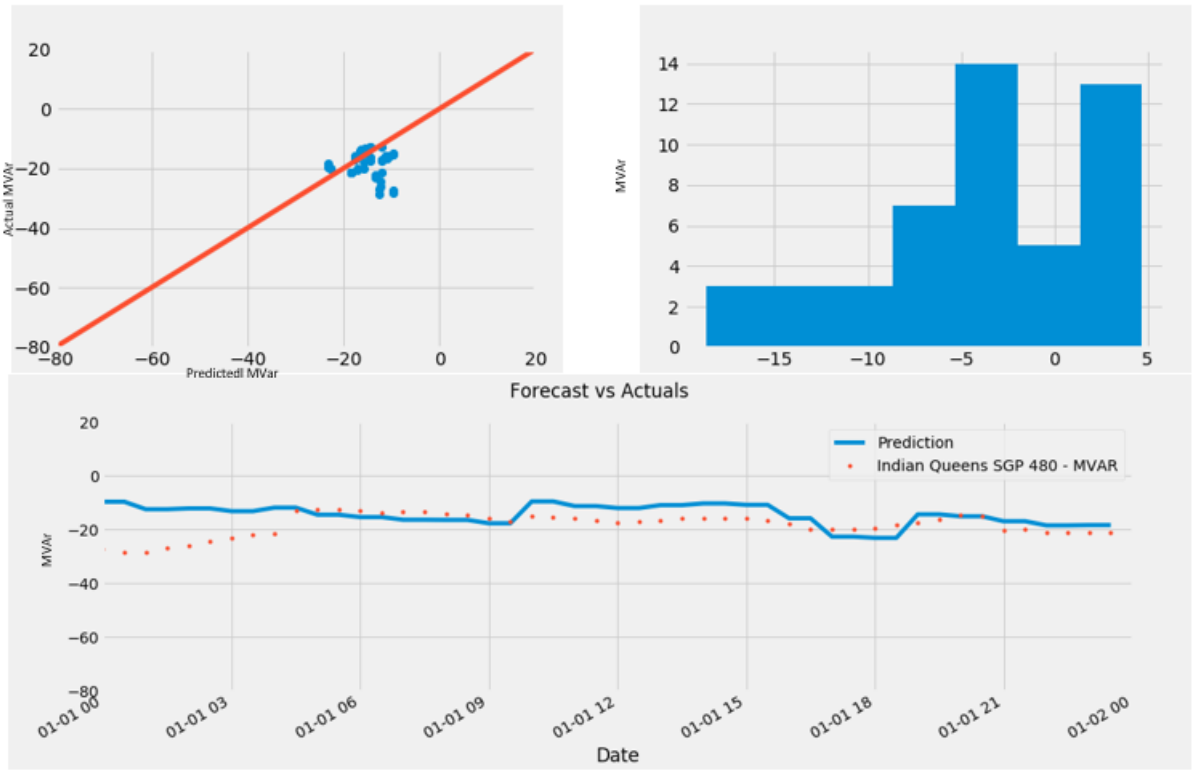


Figure 99: Day Ahead results for reactive power, 1st January 2016.

11.1.3.5. Hour Ahead

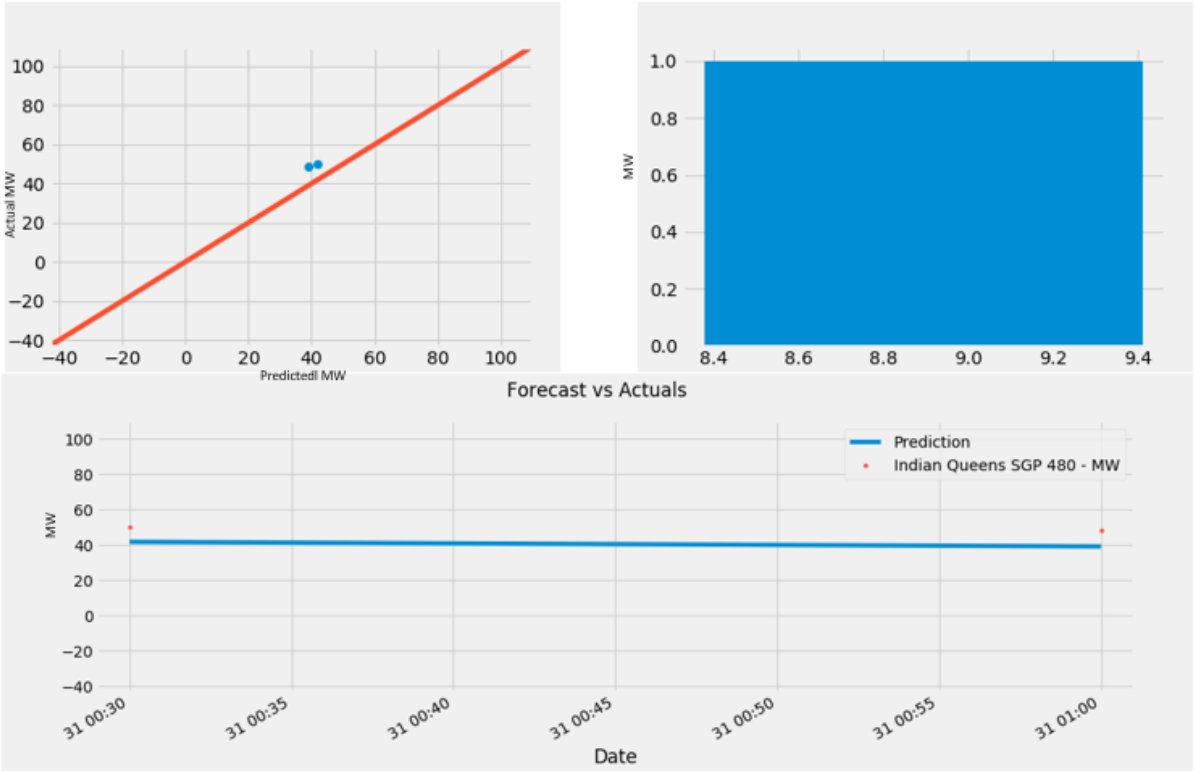


Figure 100: Hour Ahead results for real power, 31st December 2015.

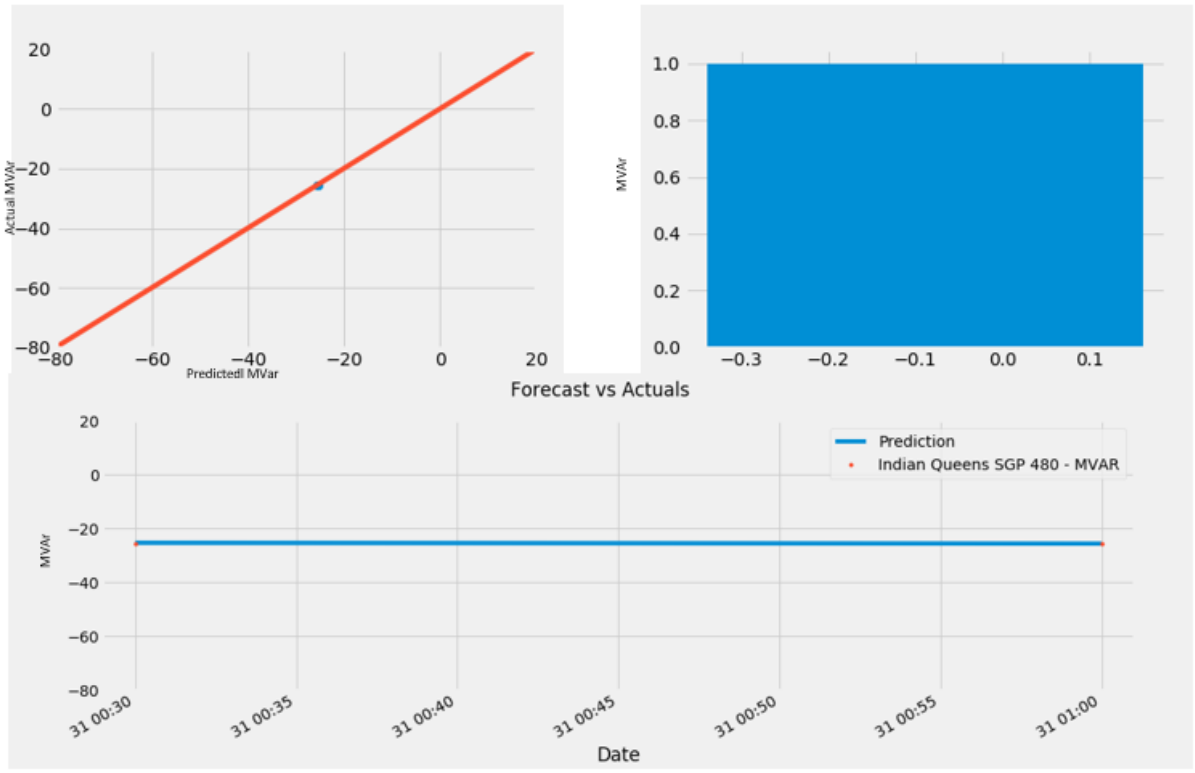


Figure 101: Hour Ahead results for reactive power, 31st December 2015.

11.1.4. Transformer 4

11.1.4.1. Six Months Ahead

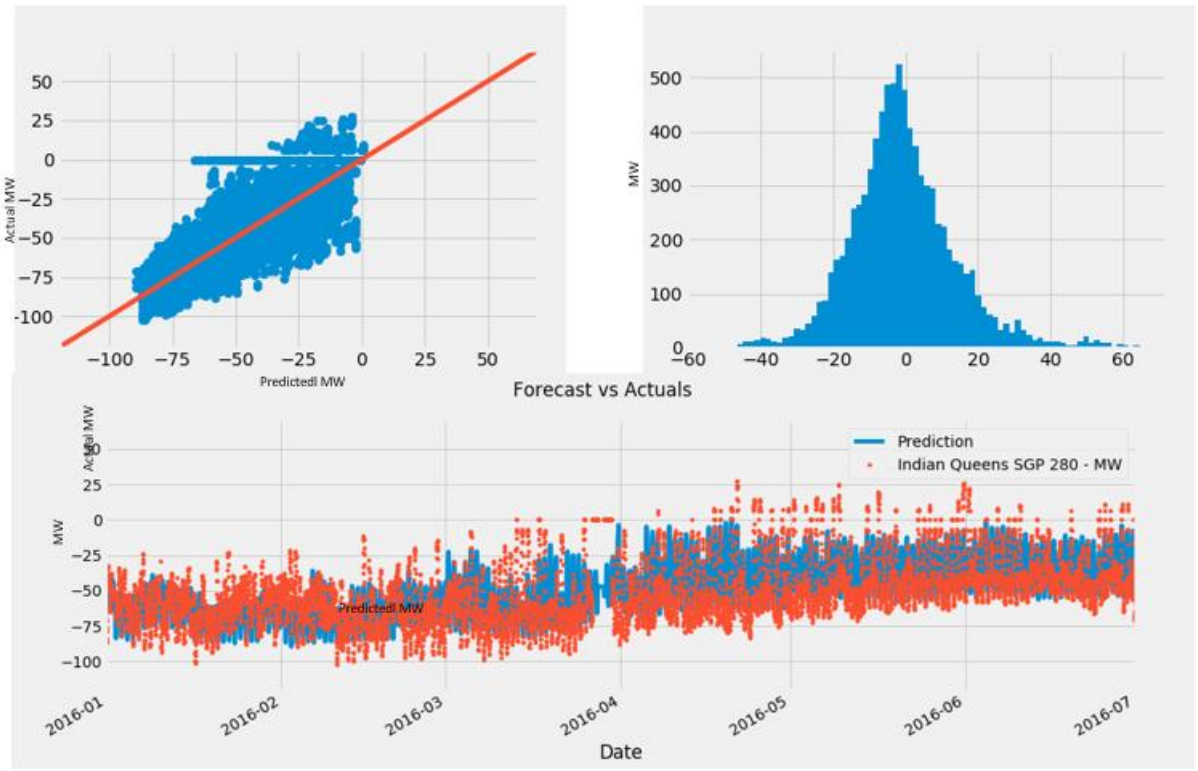


Figure 102: Six Month Ahead results for real power, January-June 2016.

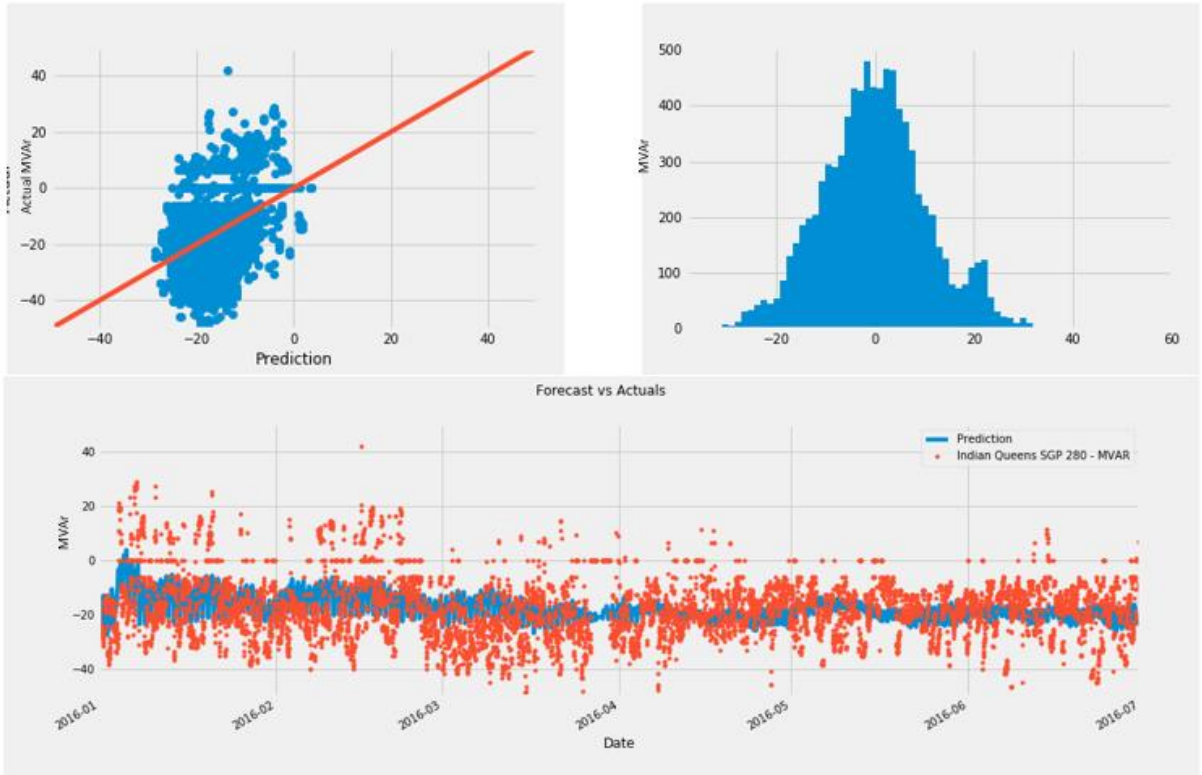


Figure 103: Six Month Ahead results for reactive power, January-June 2016.

11.1.4.2. Month Ahead

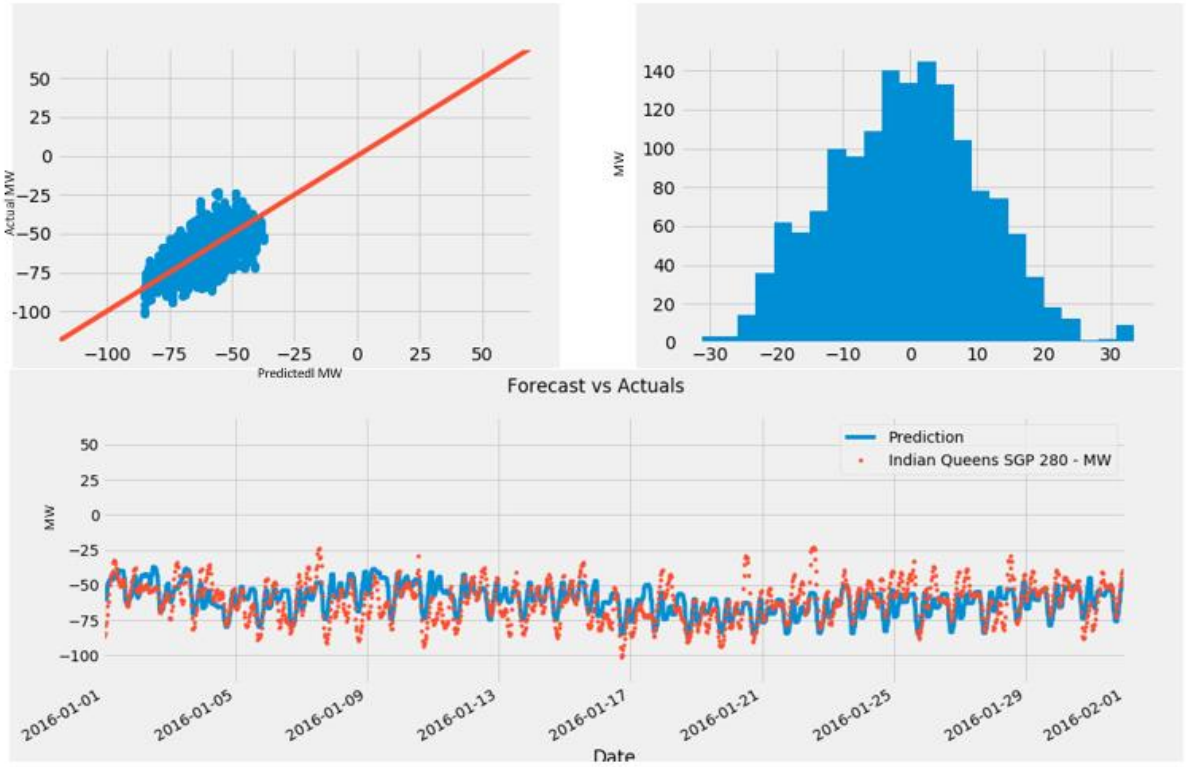


Figure 104: Month Ahead results for real power, January 2016.

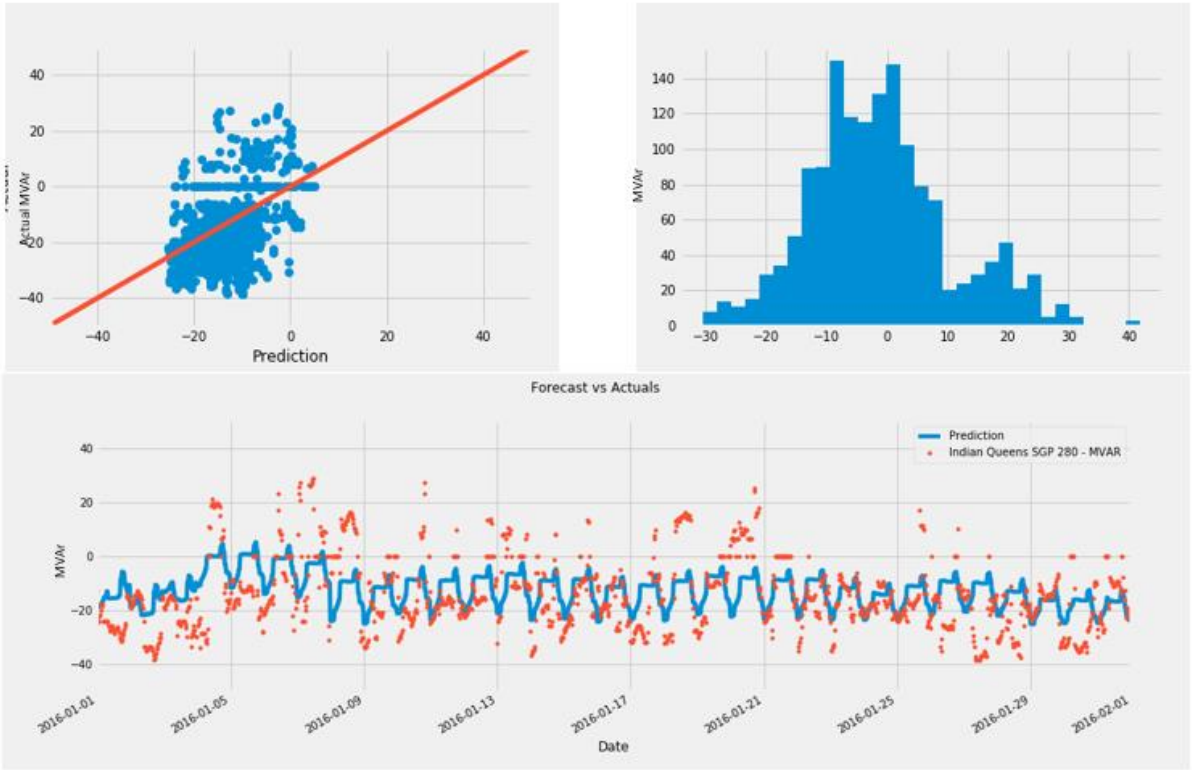


Figure 105: Month Ahead results for reactive power, January 2016.

11.1.4.3. Week Ahead

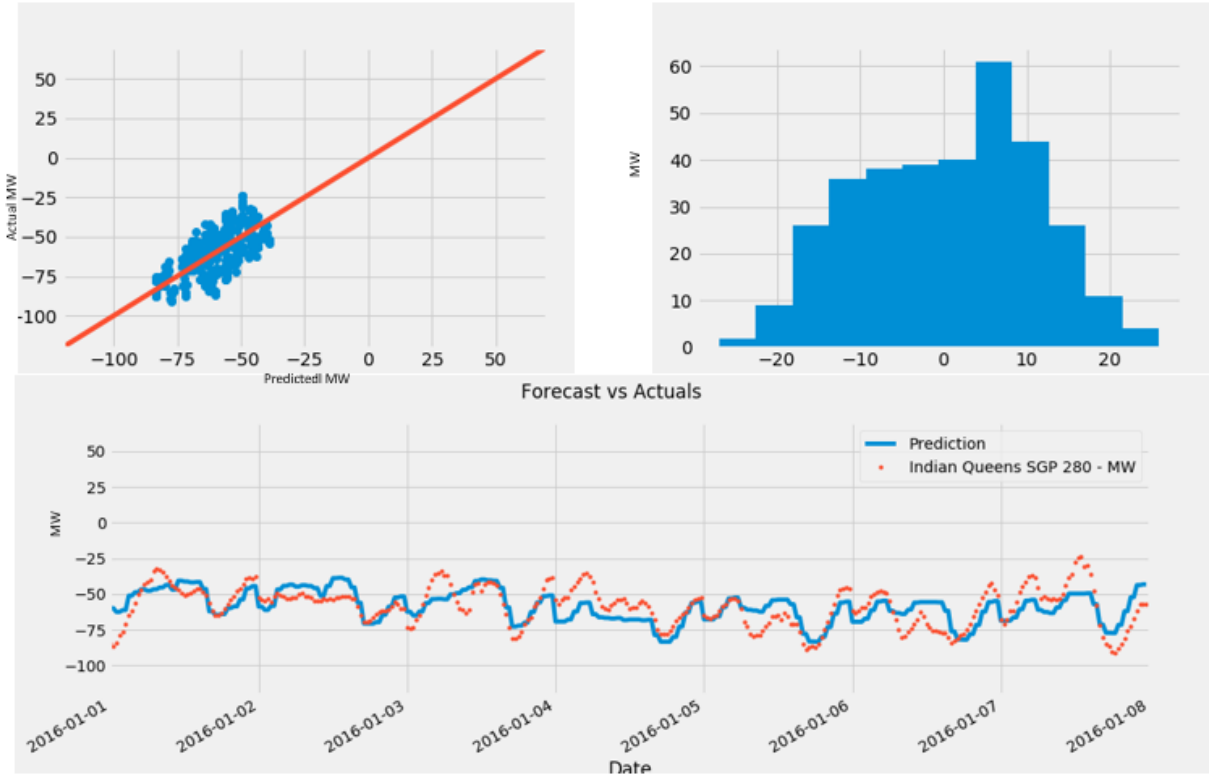


Figure 106: Week Ahead results for real power, 1-7 January 2016.

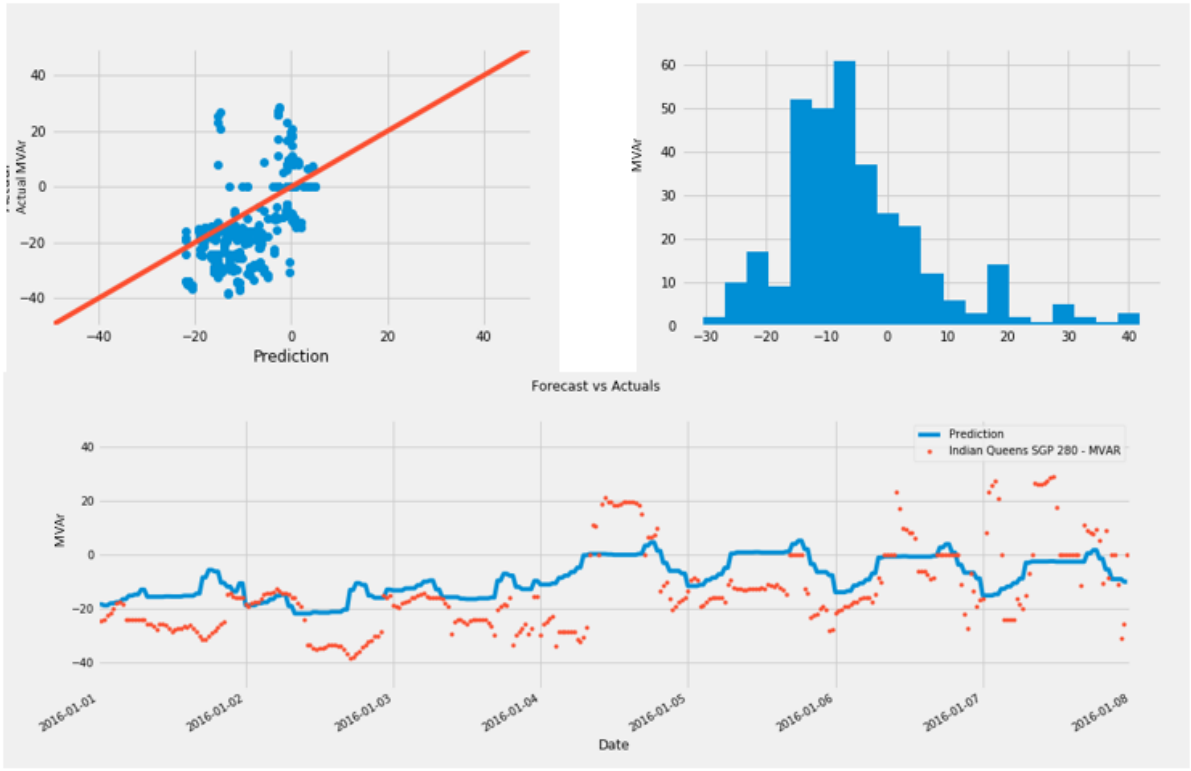


Figure 107: Week Ahead results for reactive power, 1-7 January 2016.

11.1.4.4. Day Ahead

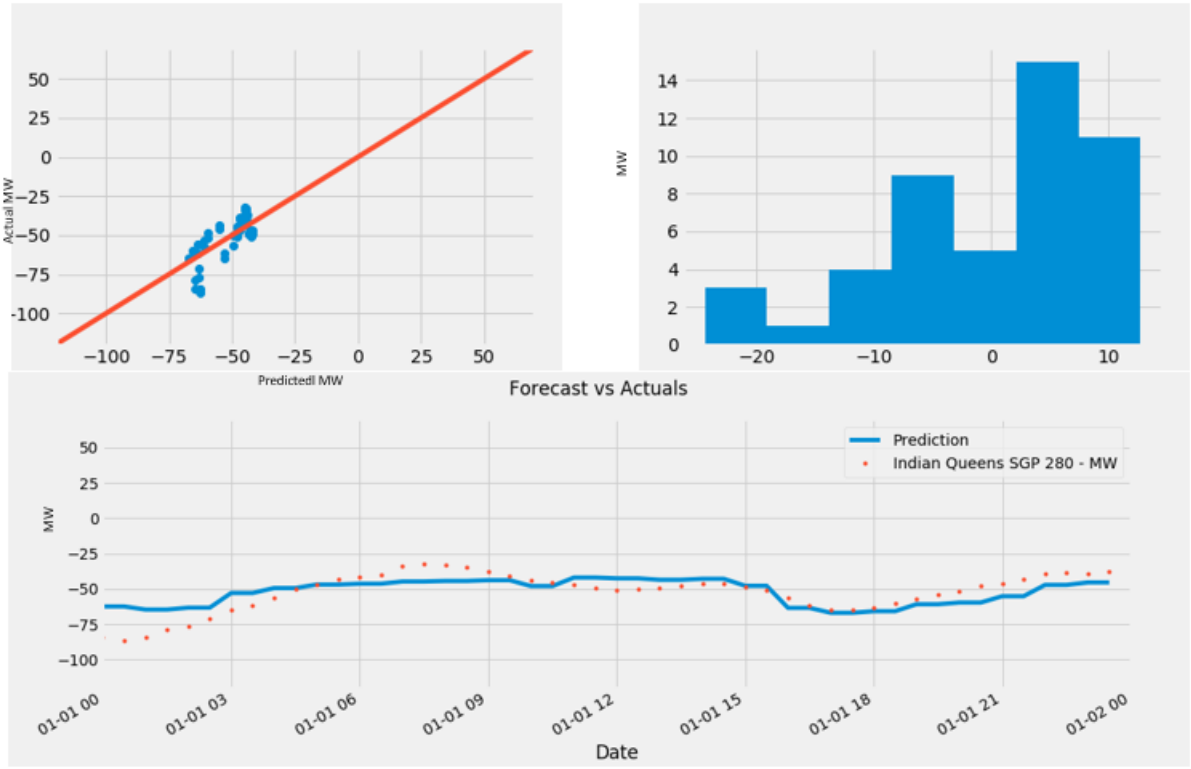


Figure 108: Day Ahead results for real power, 1st January 2016.

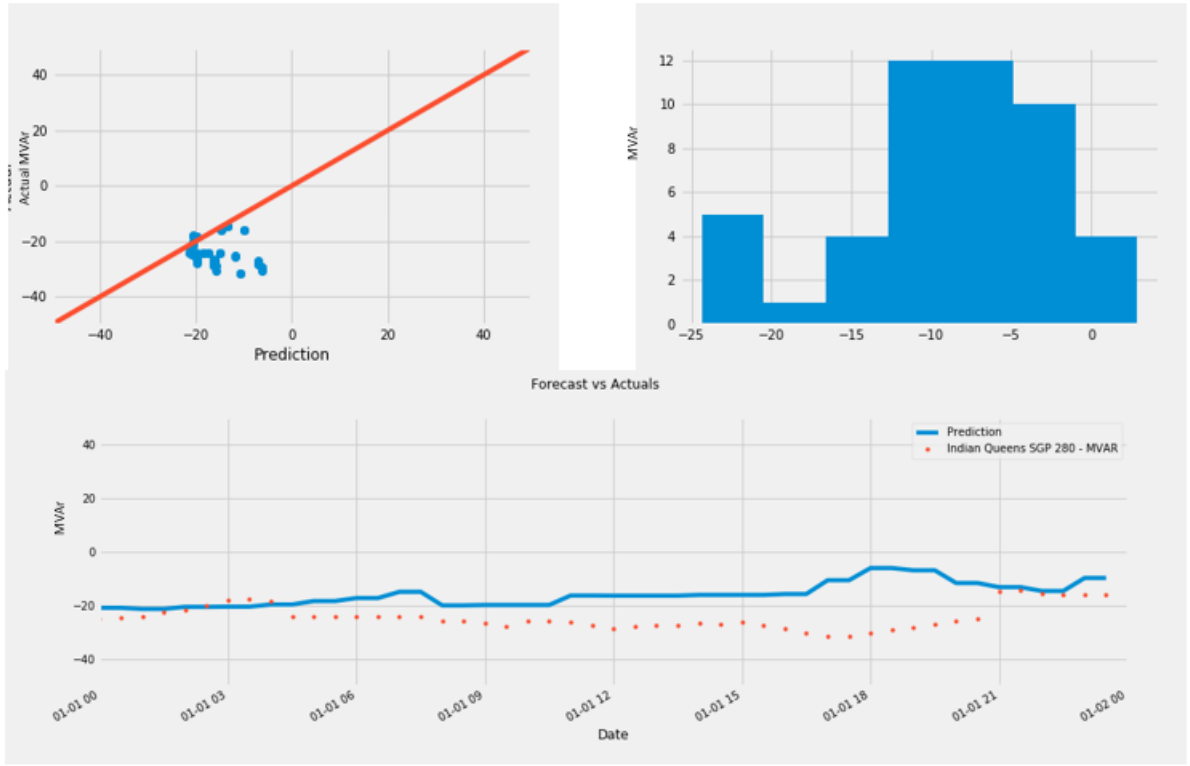


Figure 109: Day Ahead results for reactive power, 1st January 2016.

11.1.4.5. Hour Ahead

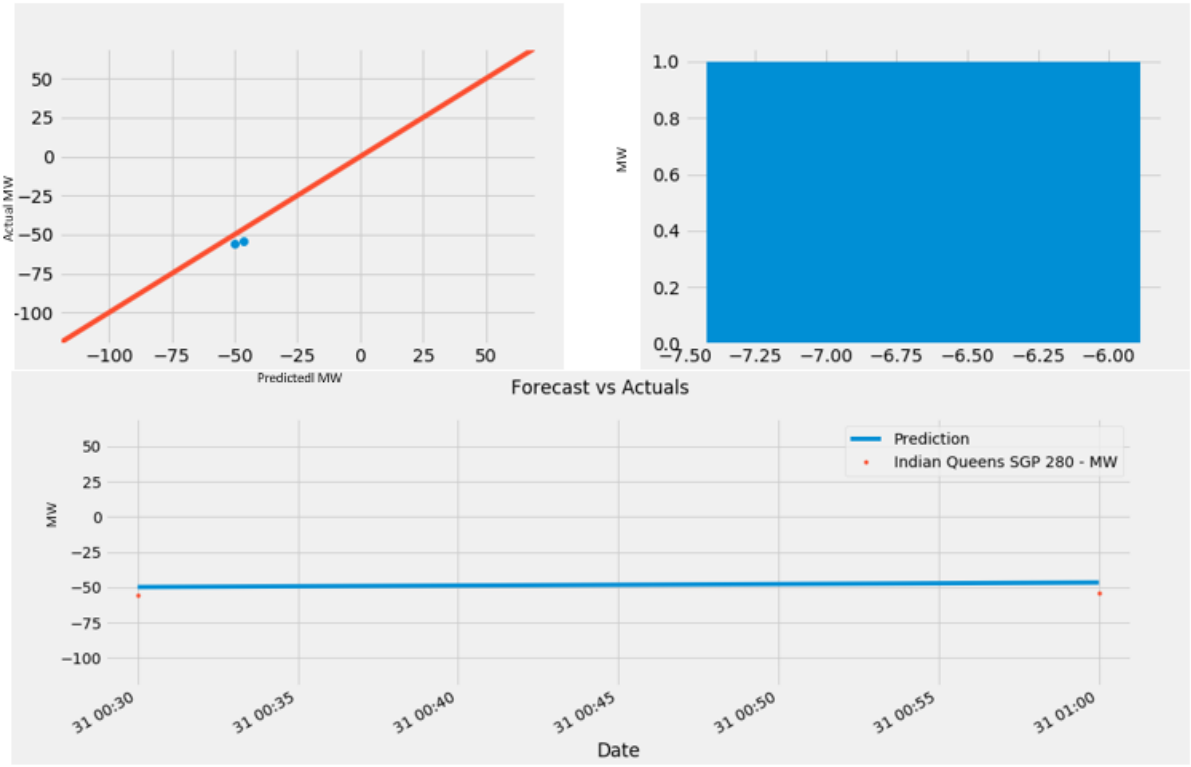


Figure 110: Hour Ahead results for real power, 31st December 2015.

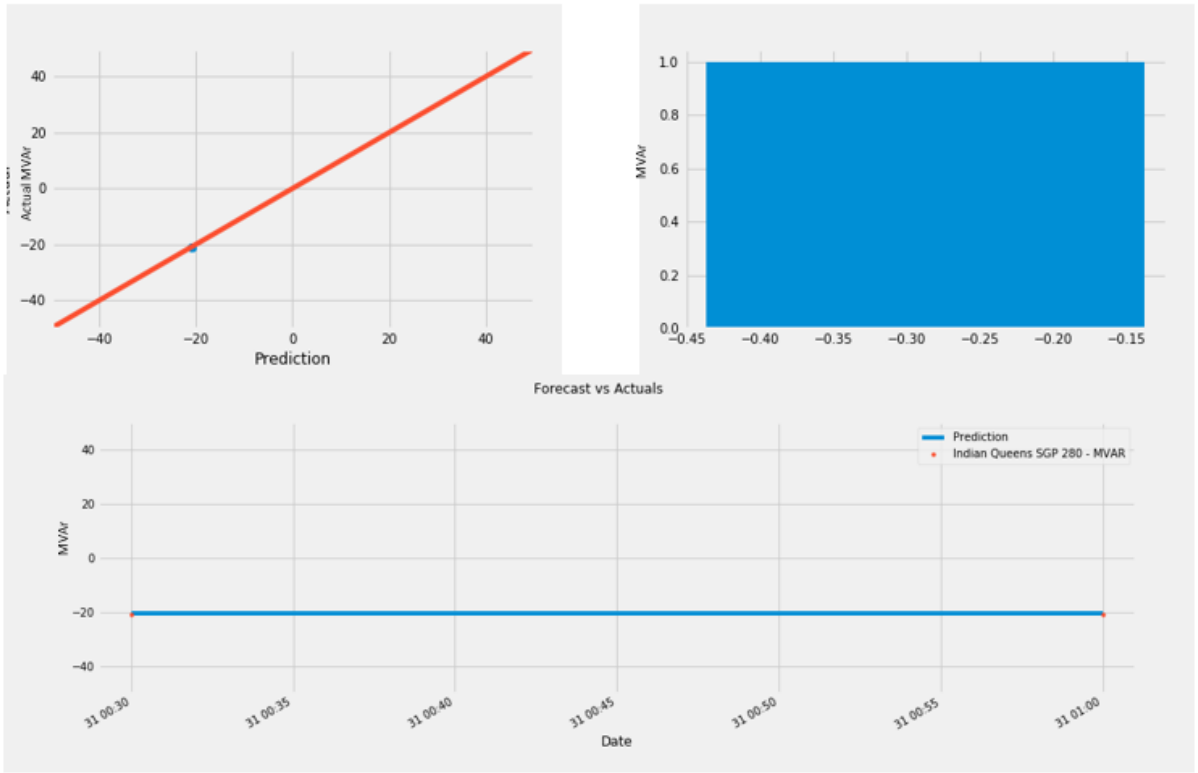


Figure 111: Hour Ahead results for reactive power, 31st December 2015.

11.2. Cardiff South

11.2.1. Six Months Ahead

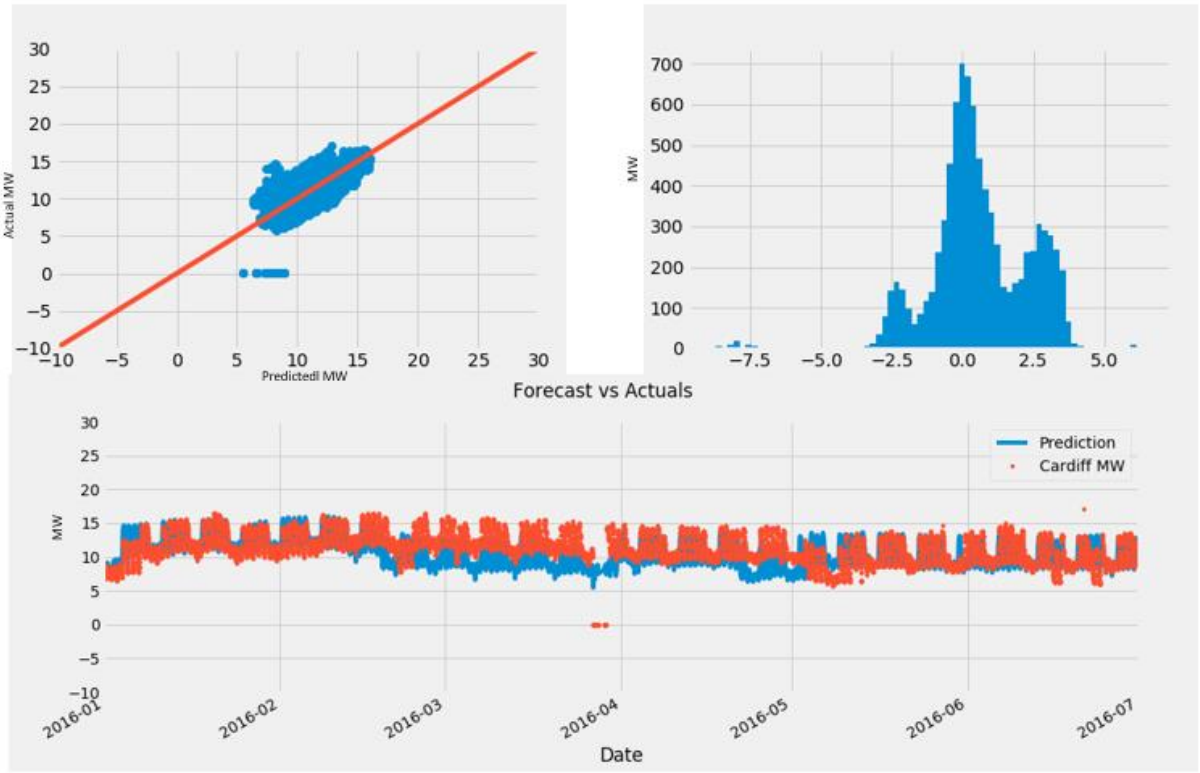


Figure 112: Six Month Ahead results for real power, January-June 2016.

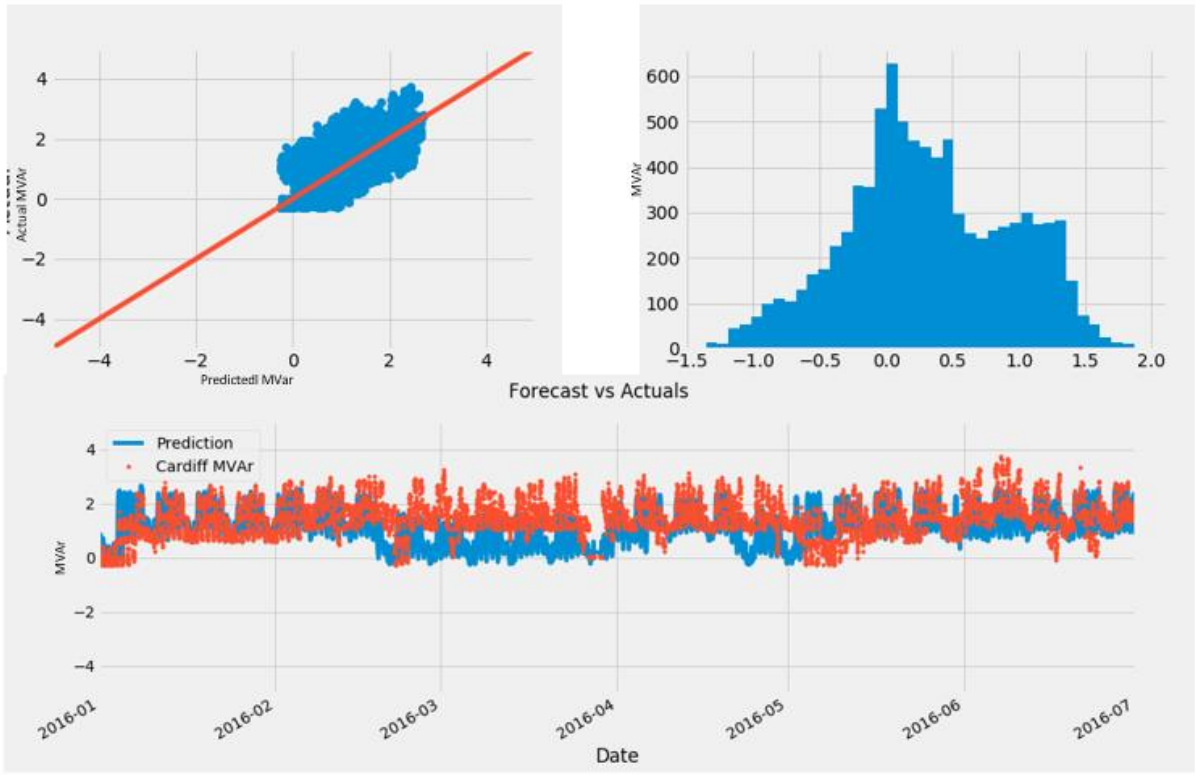


Figure 113: Six Month Ahead results for reactive power, January-June 2016.

11.2.2. Month Ahead

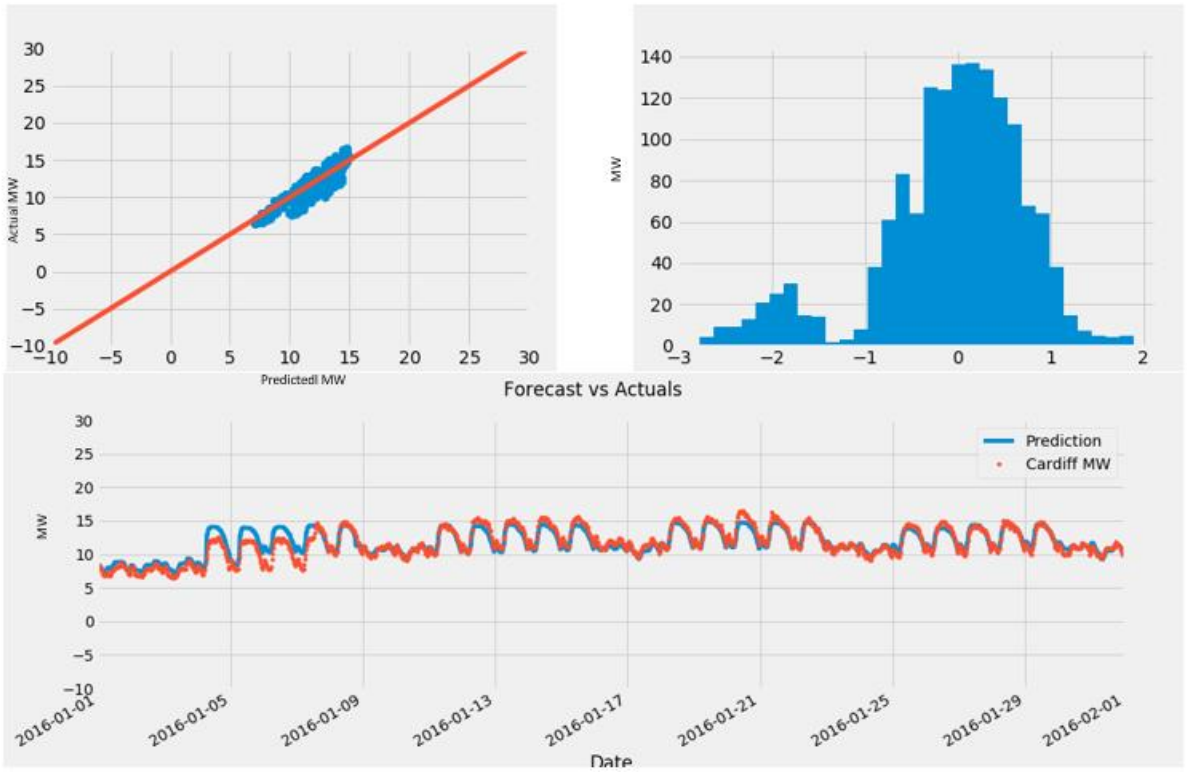


Figure 114: Month Ahead results for real power, January 2016.

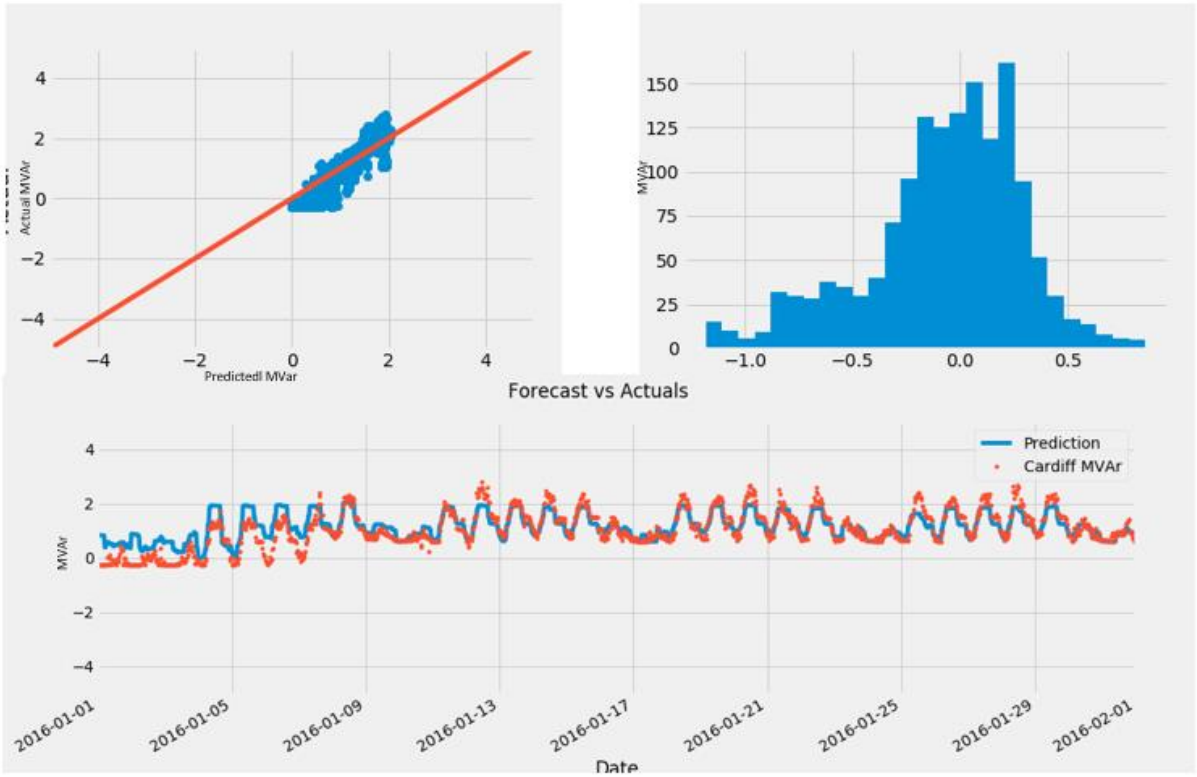


Figure 115: Month Ahead results for reactive power, January 2016.

11.2.3. Week Ahead

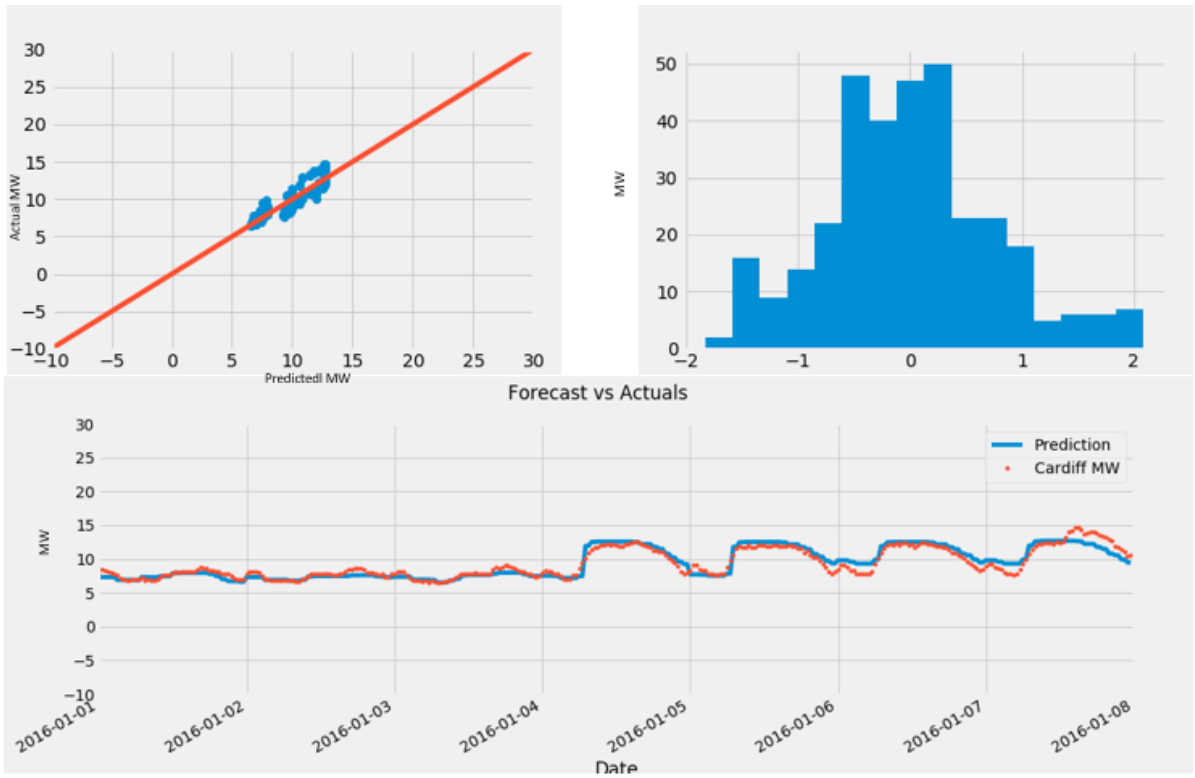


Figure 116: Week Ahead results for real power, 1-7 January 2016.

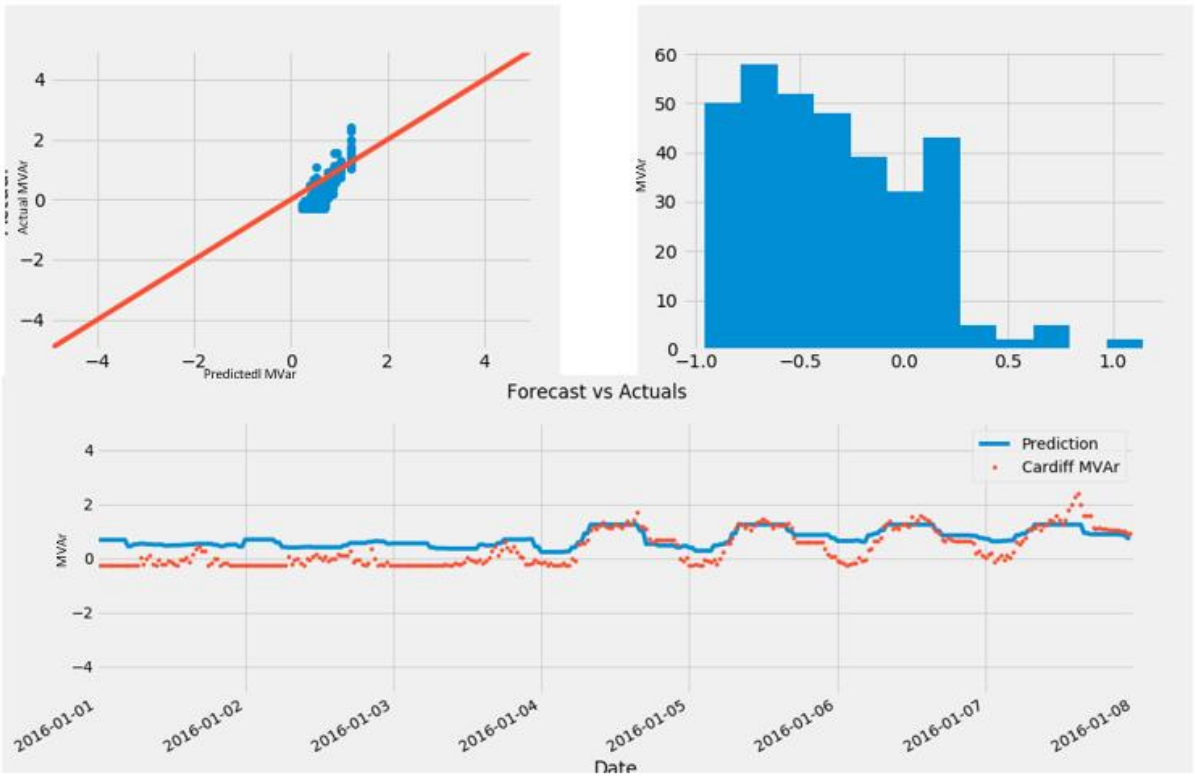


Figure 117: Week Ahead results for reactive power, 1-7 January 2016.

11.2.4. Day Ahead

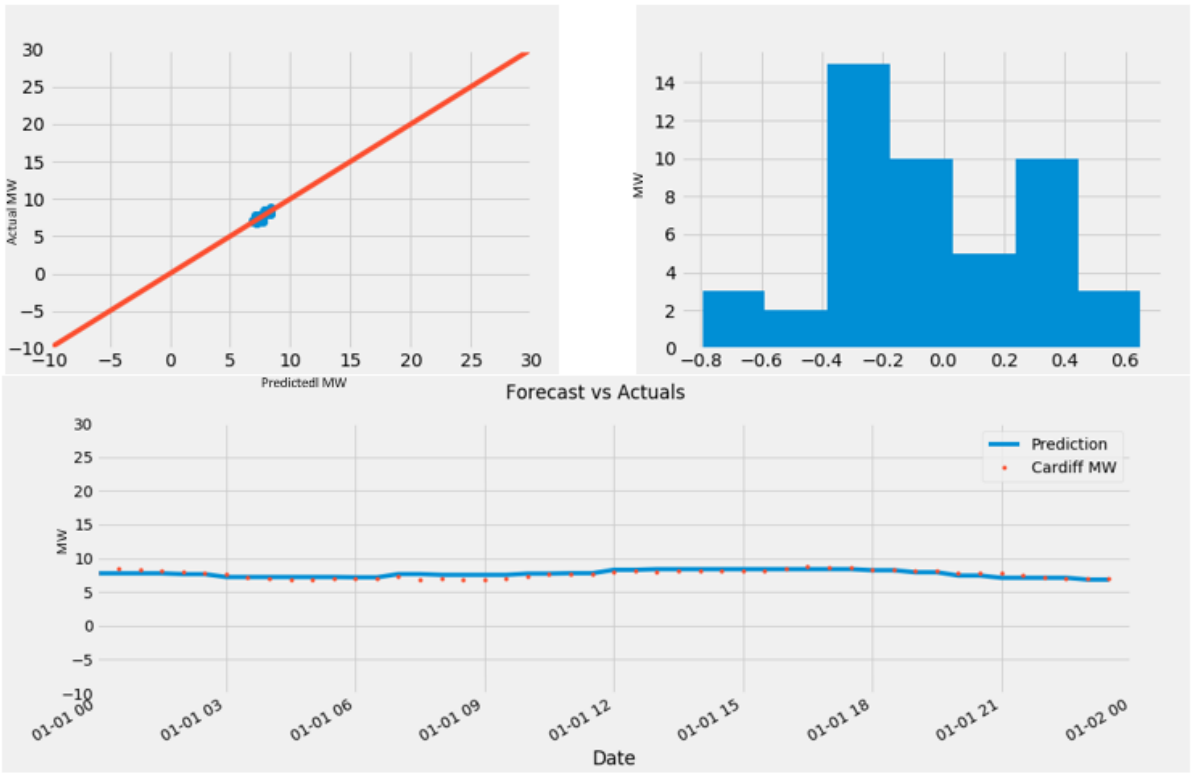


Figure 118: Day Ahead results for real power, 1st January 2016.

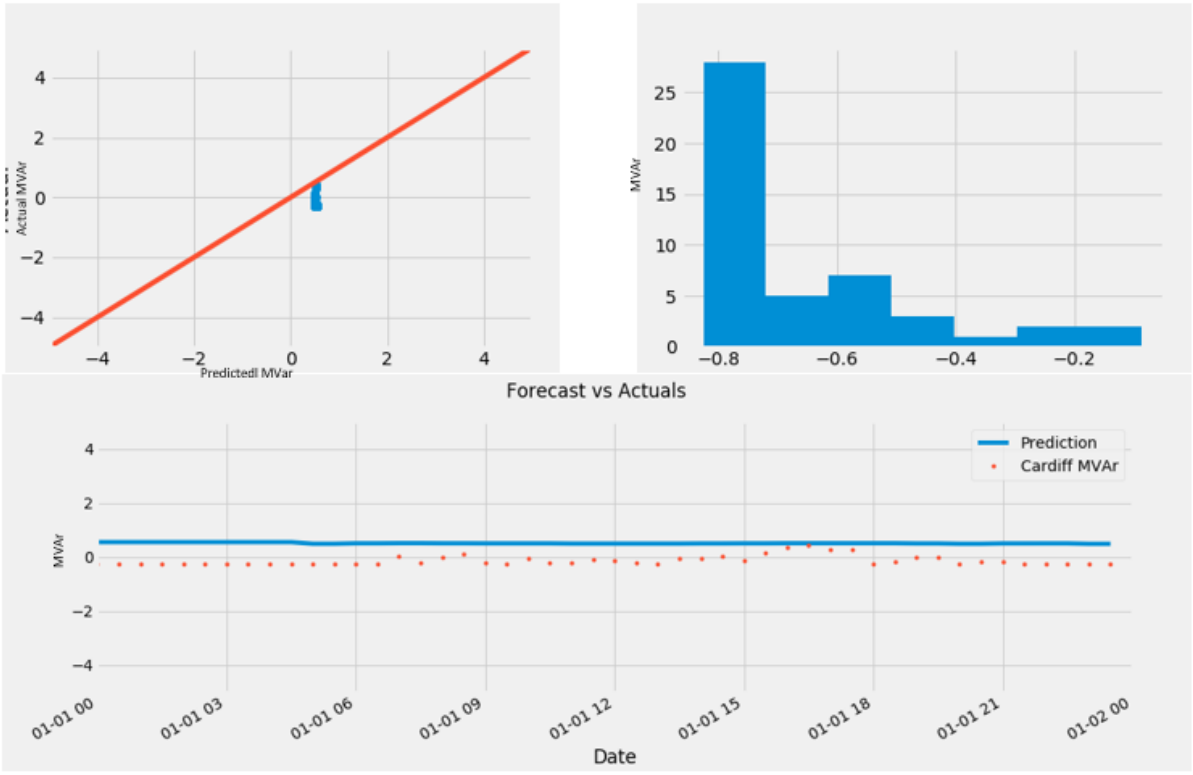


Figure 119: Day Ahead results for reactive power, 1st January 2016.

11.2.5. Hour Ahead

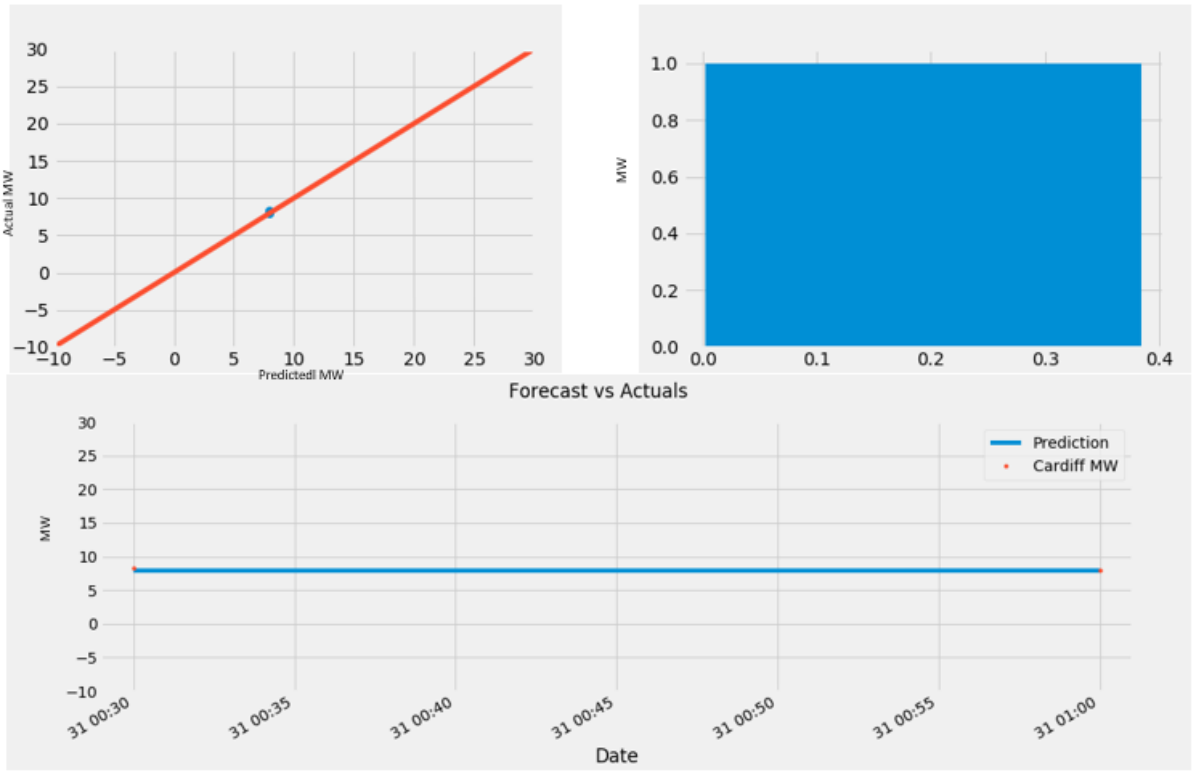


Figure 120: Hour Ahead results for real power, 31st December 2015.

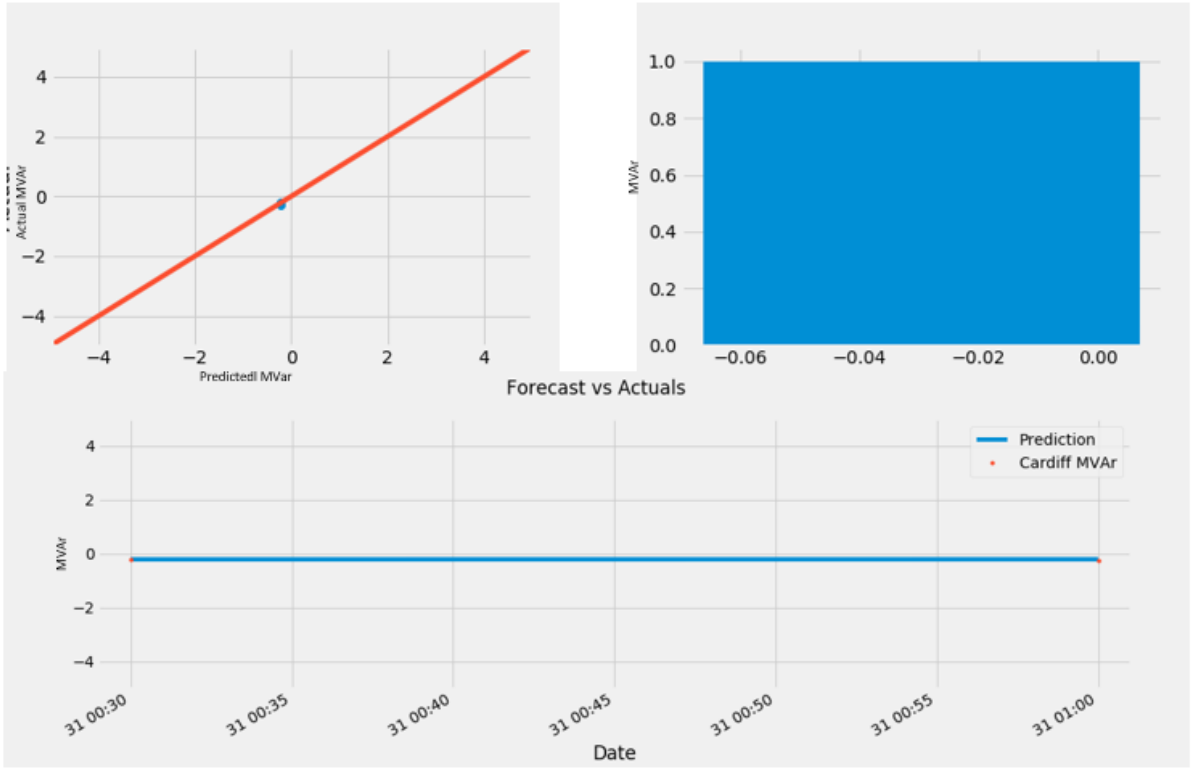


Figure 121: Hour Ahead results for reactive power, 31st December 2015.

11.3. Prince Rock

11.3.1. Six Months Ahead

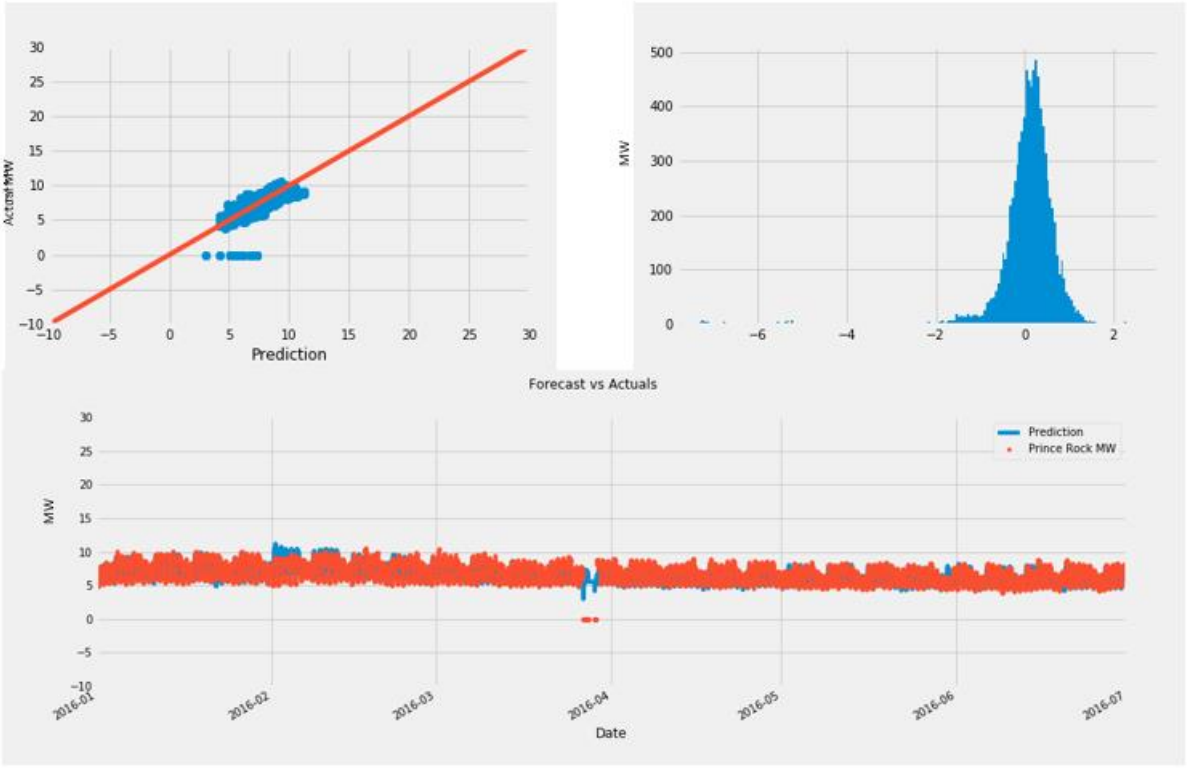


Figure 122: Six Month Ahead results for real power, January-June 2016.

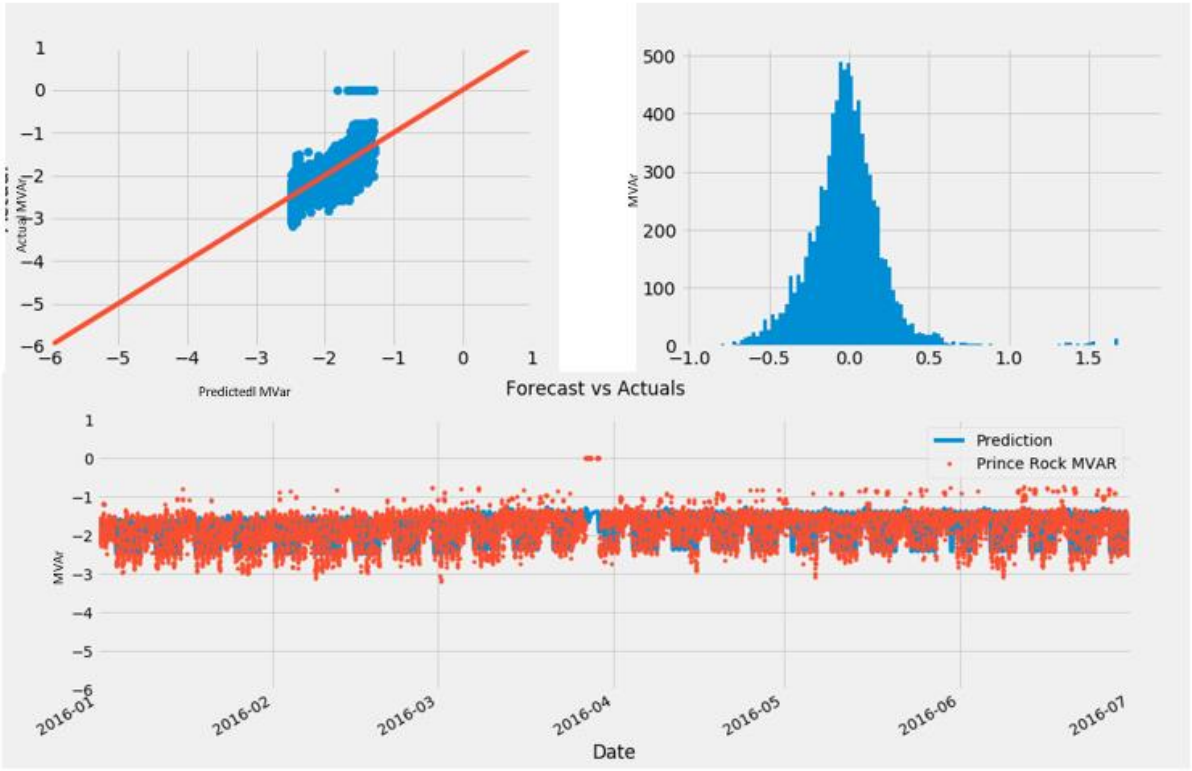


Figure 123: Six Month Ahead results for reactive power, January-June 2016.

11.3.2. Month Ahead

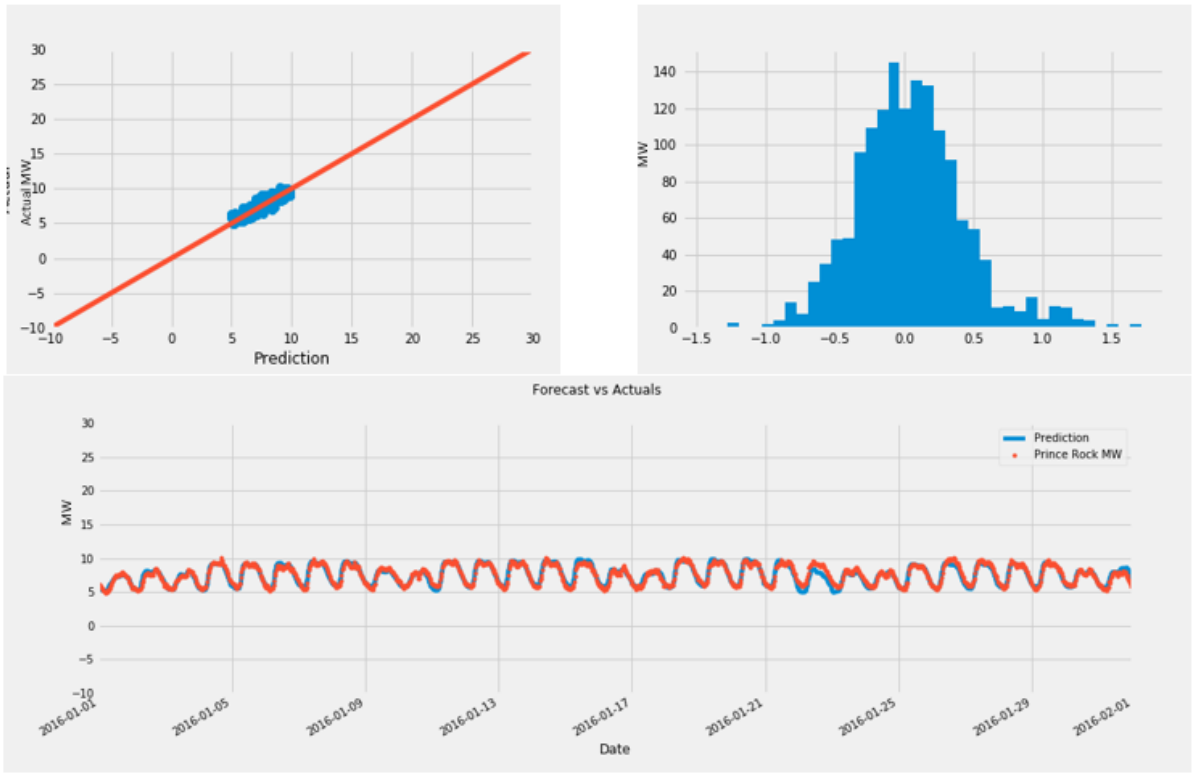


Figure 124: Month Ahead results for real power, January 2016.

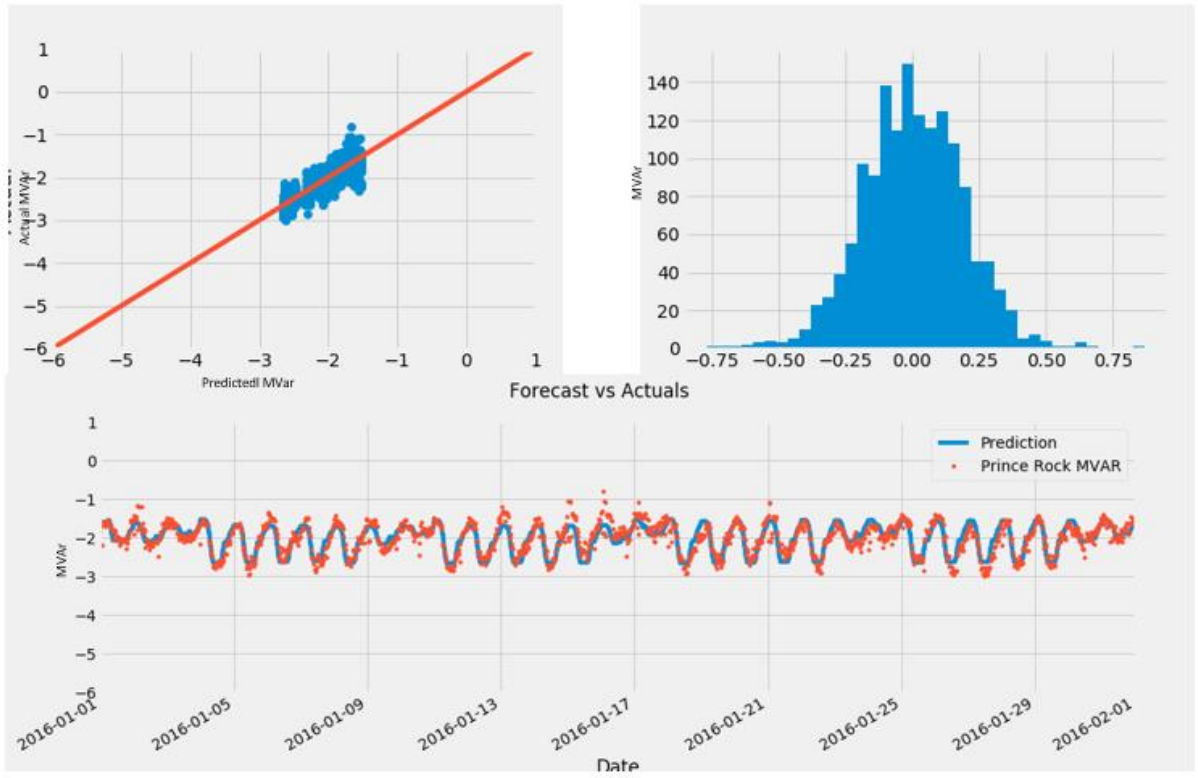


Figure 125: Month Ahead results for reactive power, January 2016.

11.3.3. Week Ahead

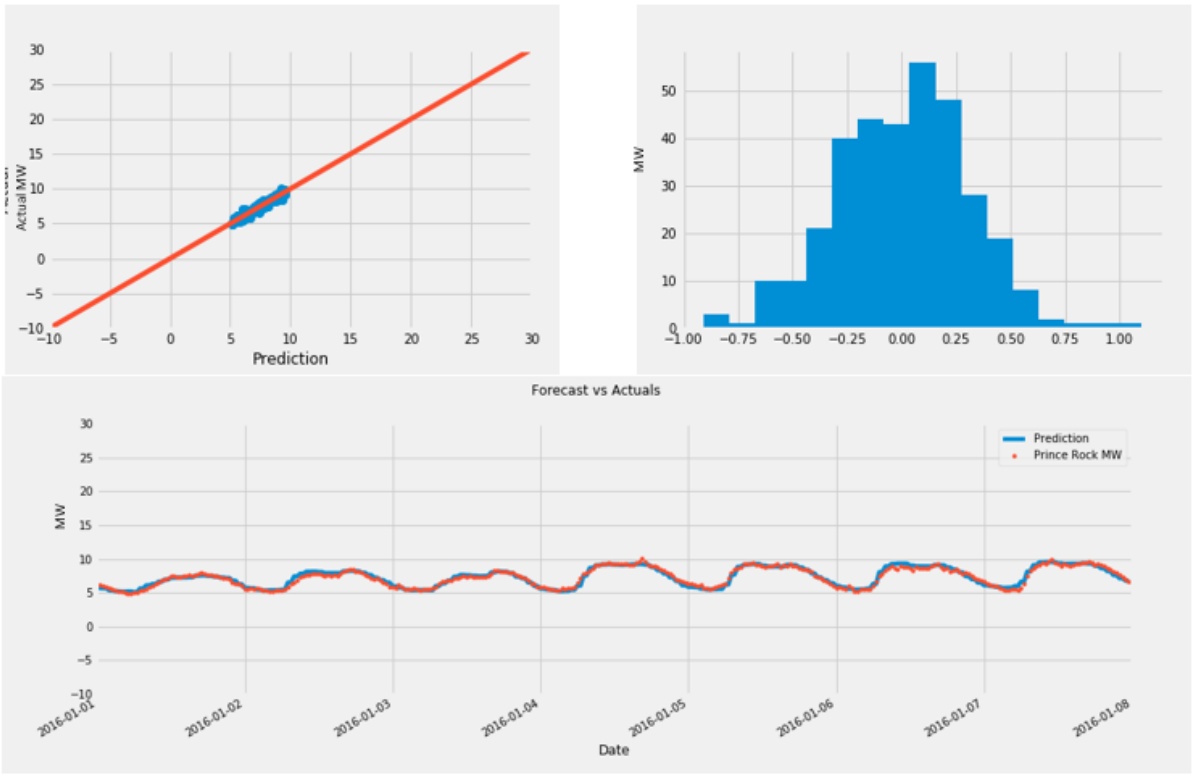


Figure 126: Week Ahead results for real power, 1-7 January 2016.

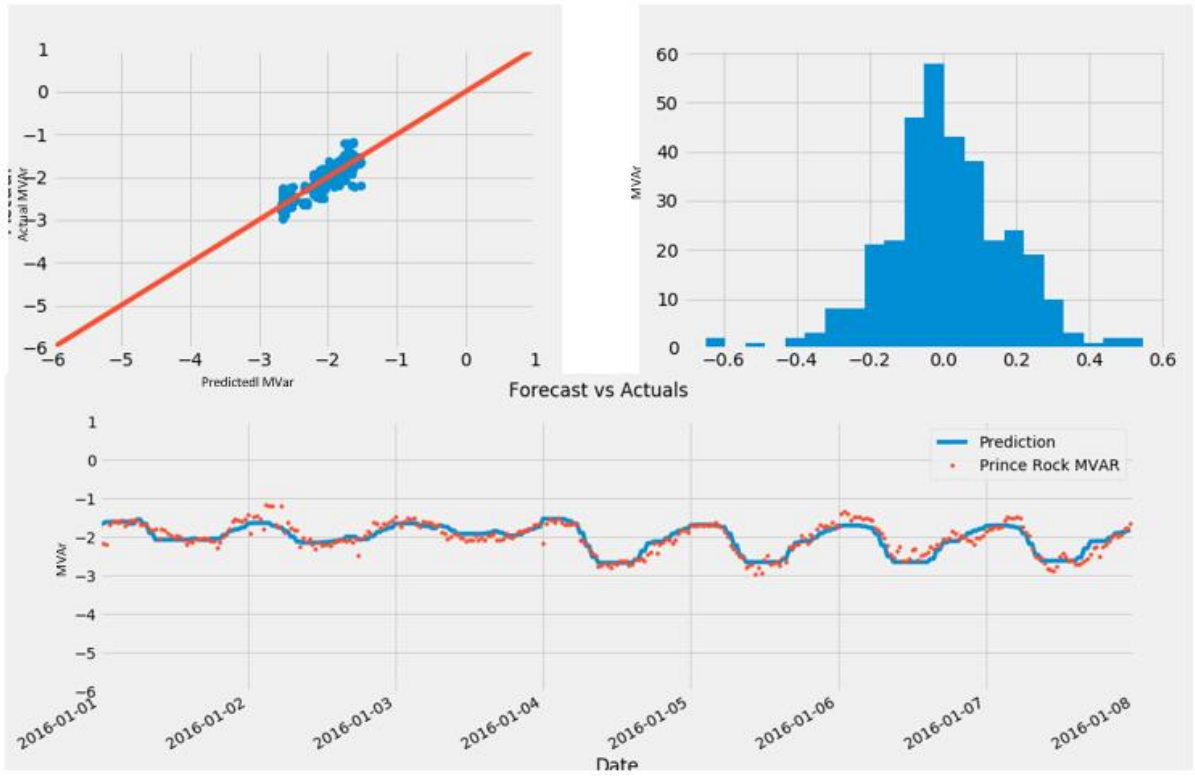


Figure 127: Week Ahead results for reactive power, 1-7 January 2016.

11.3.4. Day Ahead

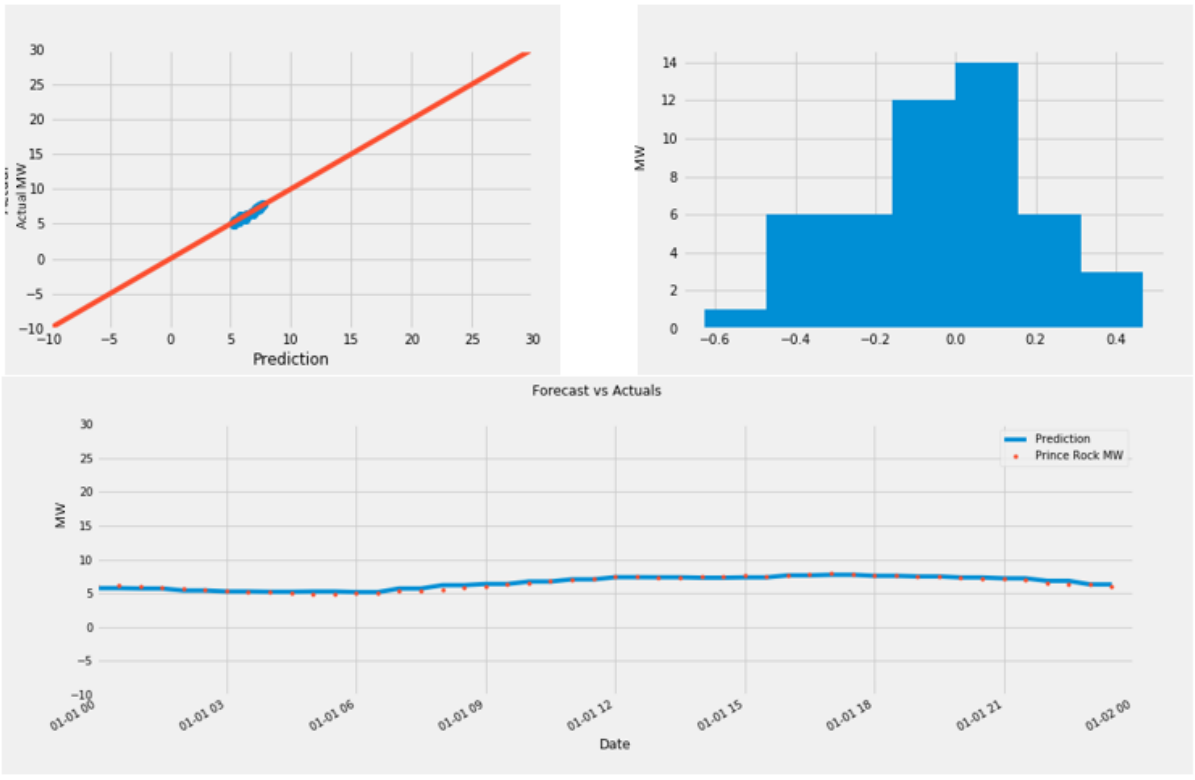


Figure 128: Day Ahead results for real power, 1st January 2016.

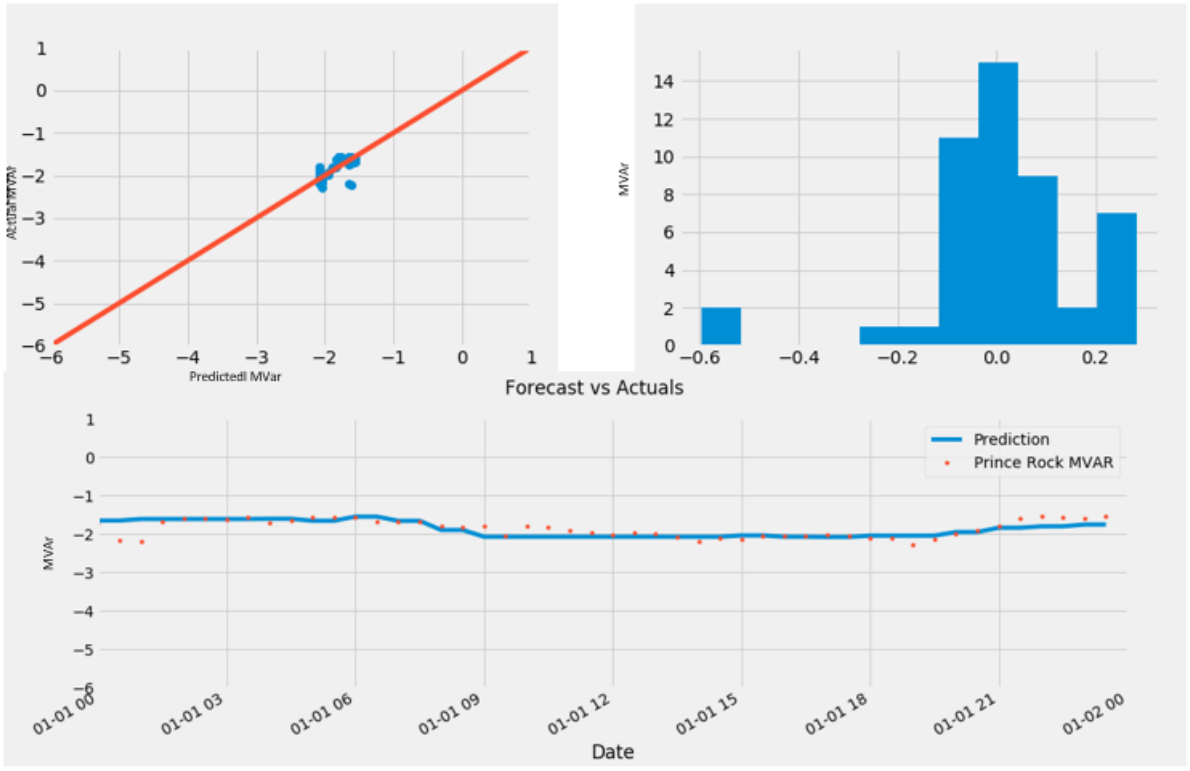


Figure 129: Day Ahead results for reactive power, 1st January 2016.

11.3.5. Hour Ahead

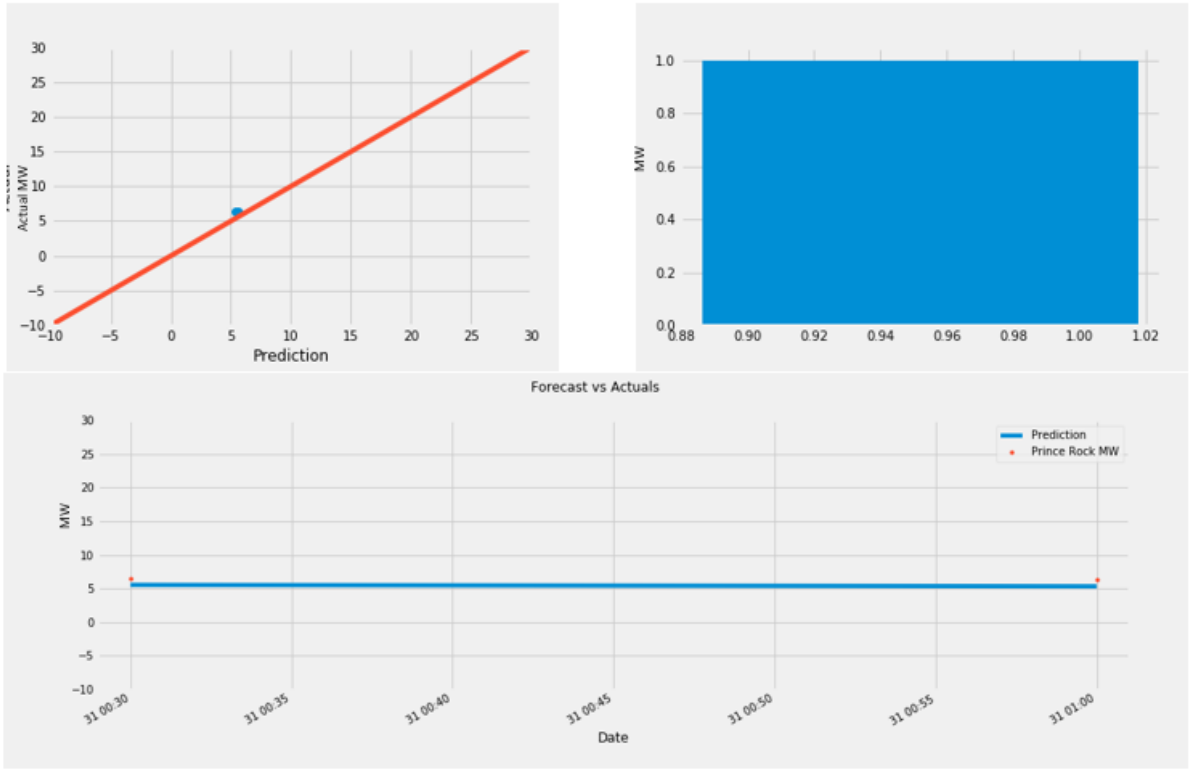


Figure 130: Hour Ahead results for real power, 31st December 2015.

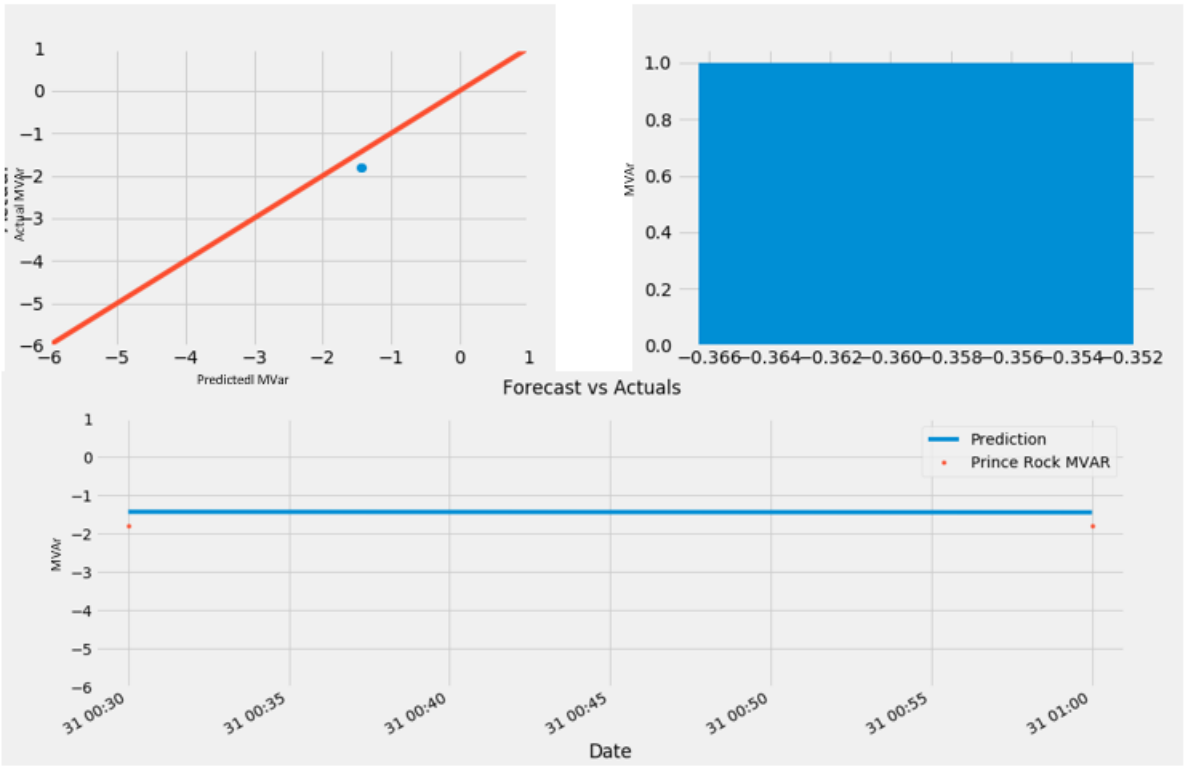


Figure 131: Hour Ahead results for reactive power, 31st December 2015.

11.4. Truro

11.4.1. Six Months Ahead

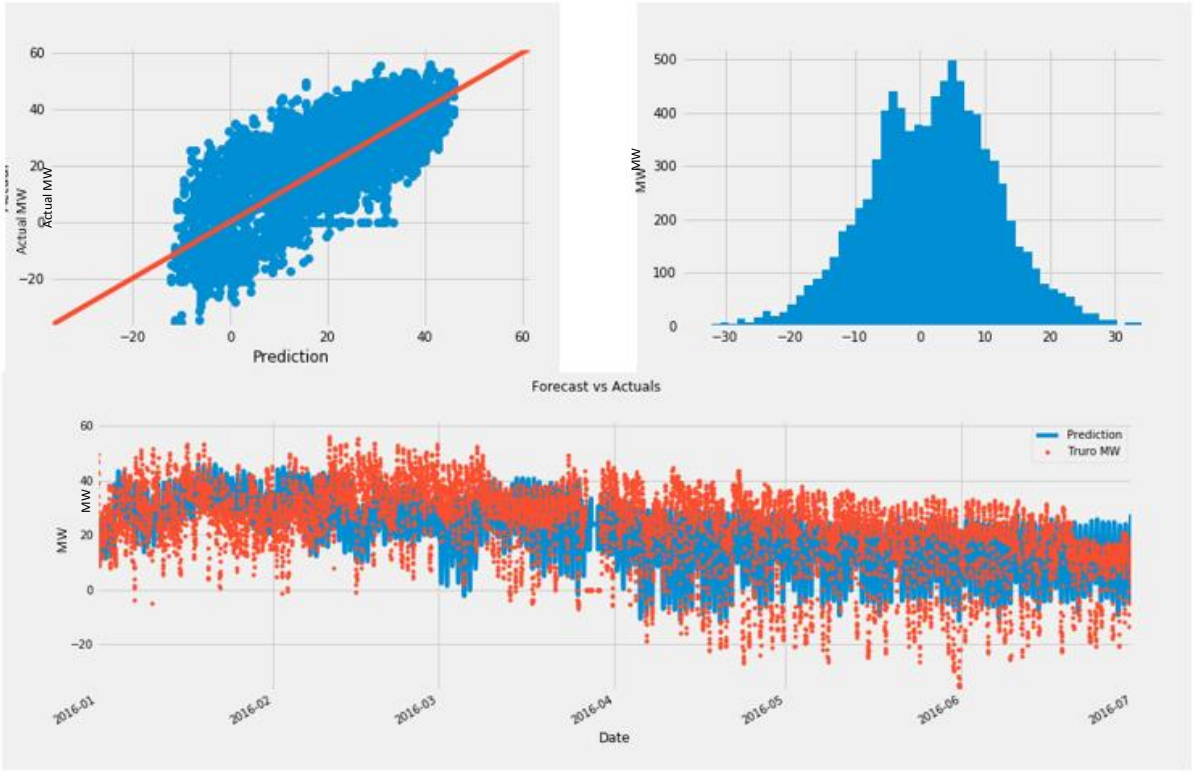


Figure 132: Six Month Ahead results for real power, January-June 2016.

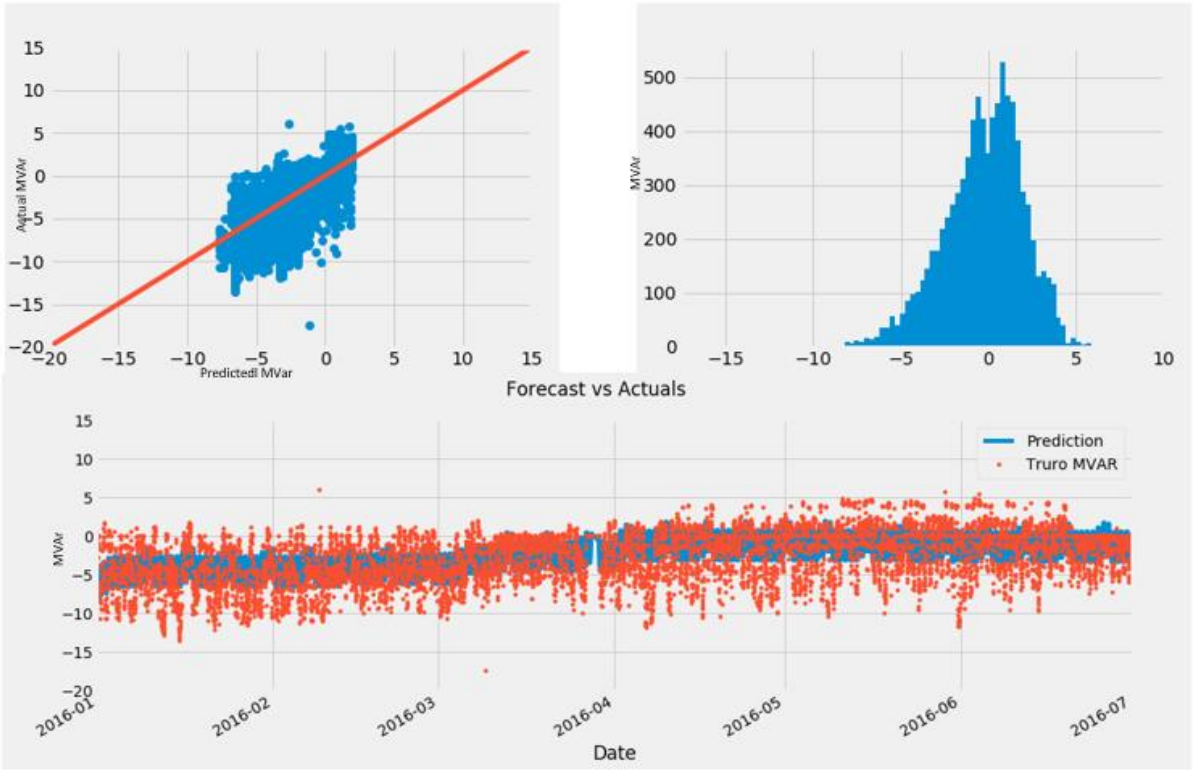


Figure 133: Six Month Ahead results for reactive power, January-June 2016.

11.4.2. Month Ahead

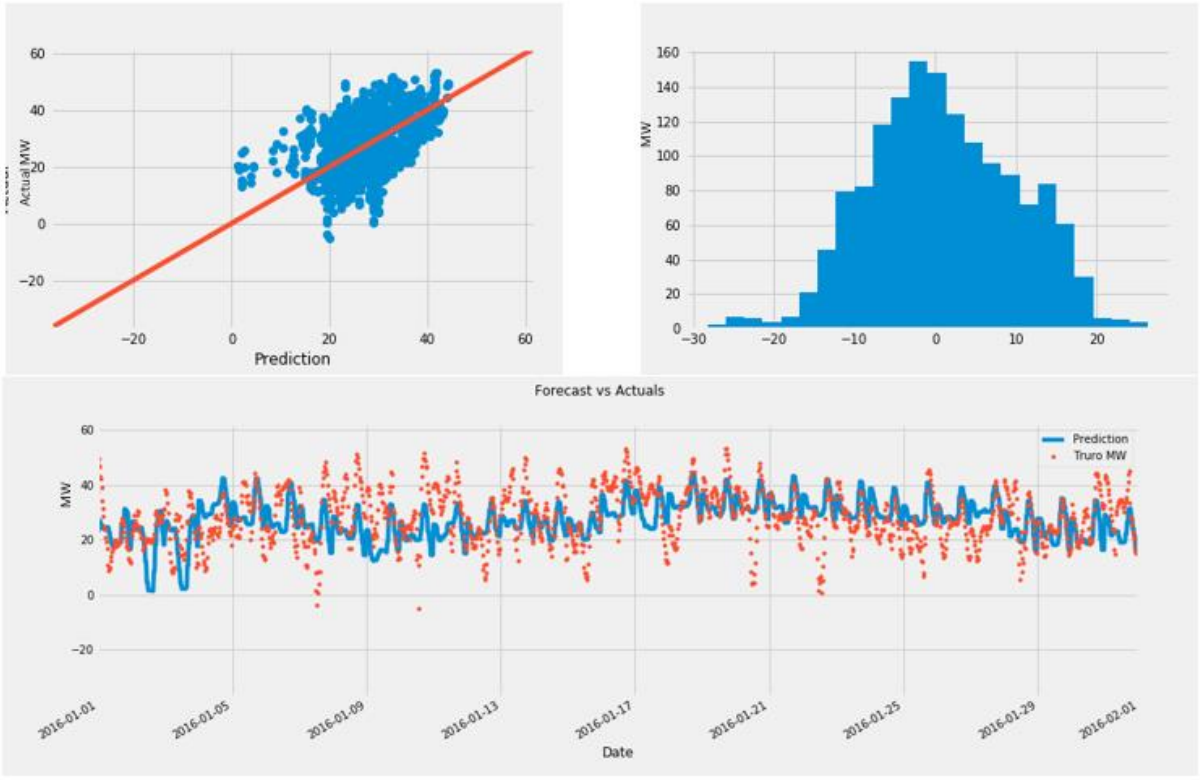


Figure 134: Month Ahead results for real power, January 2016.

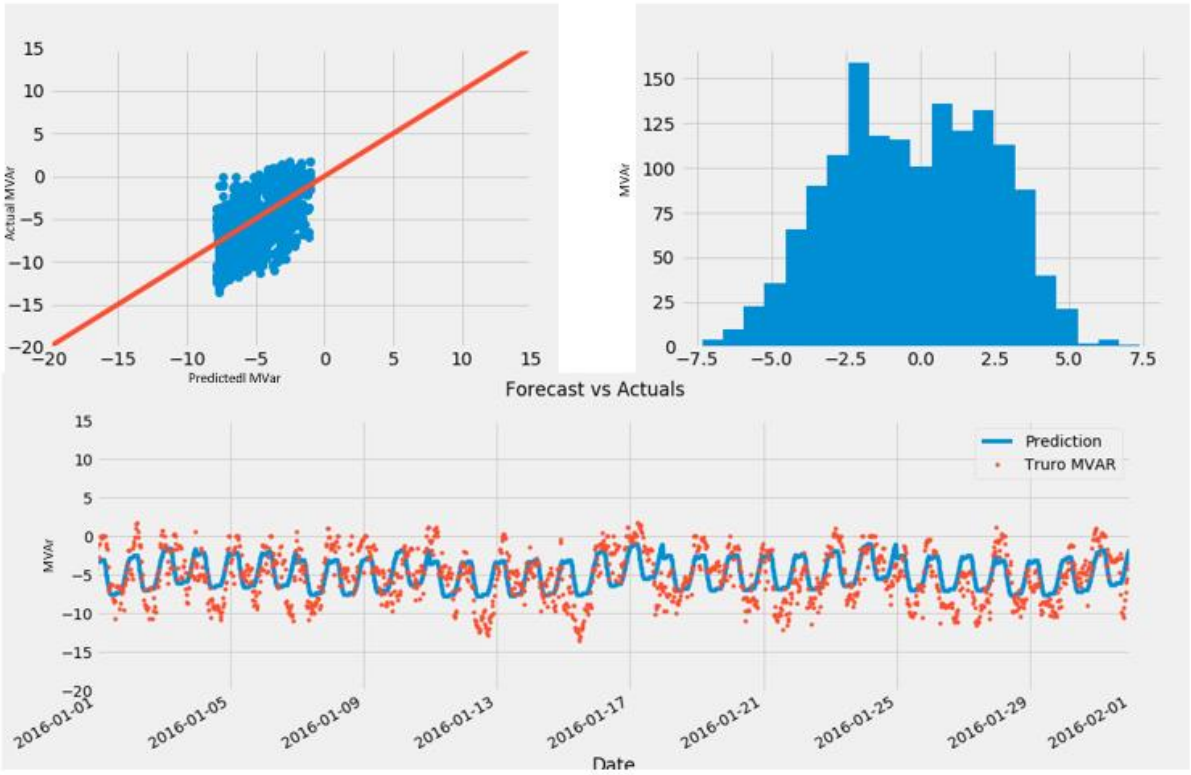


Figure 135: Month Ahead results for reactive power, January 2016.

11.4.3. Week Ahead

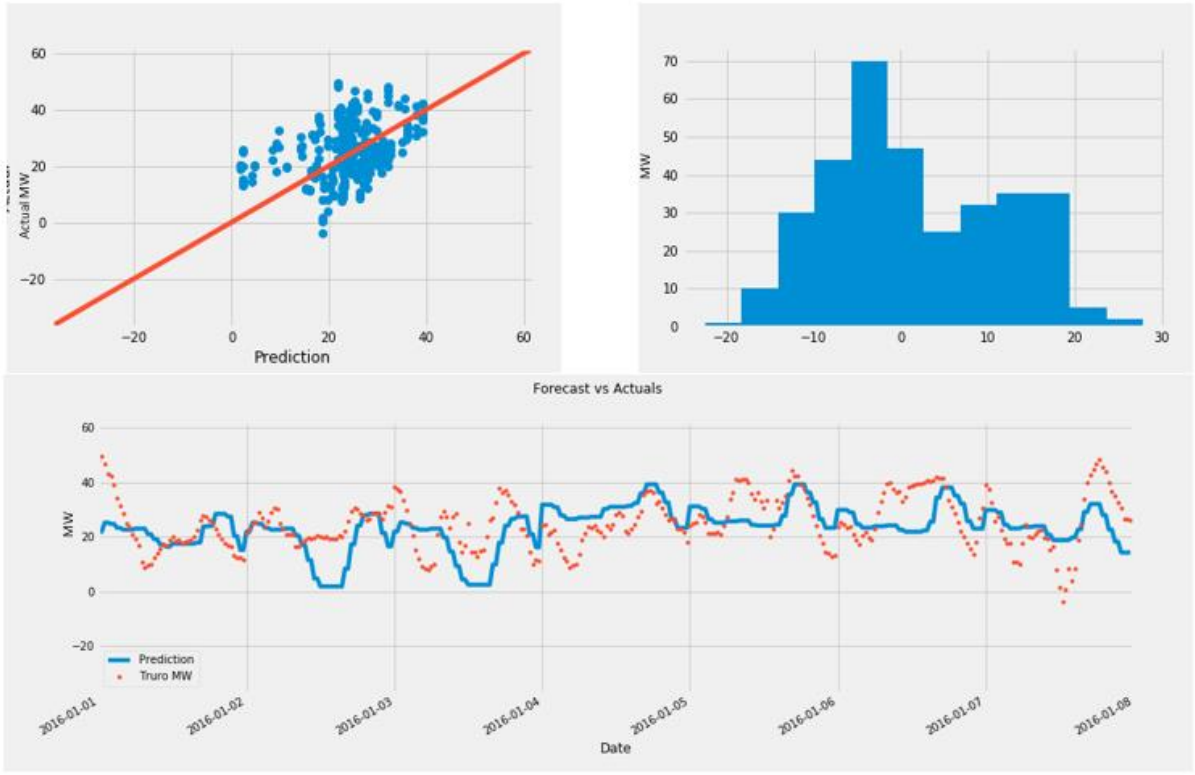


Figure 136: Week Ahead results for real power, 1-7 January 2016.

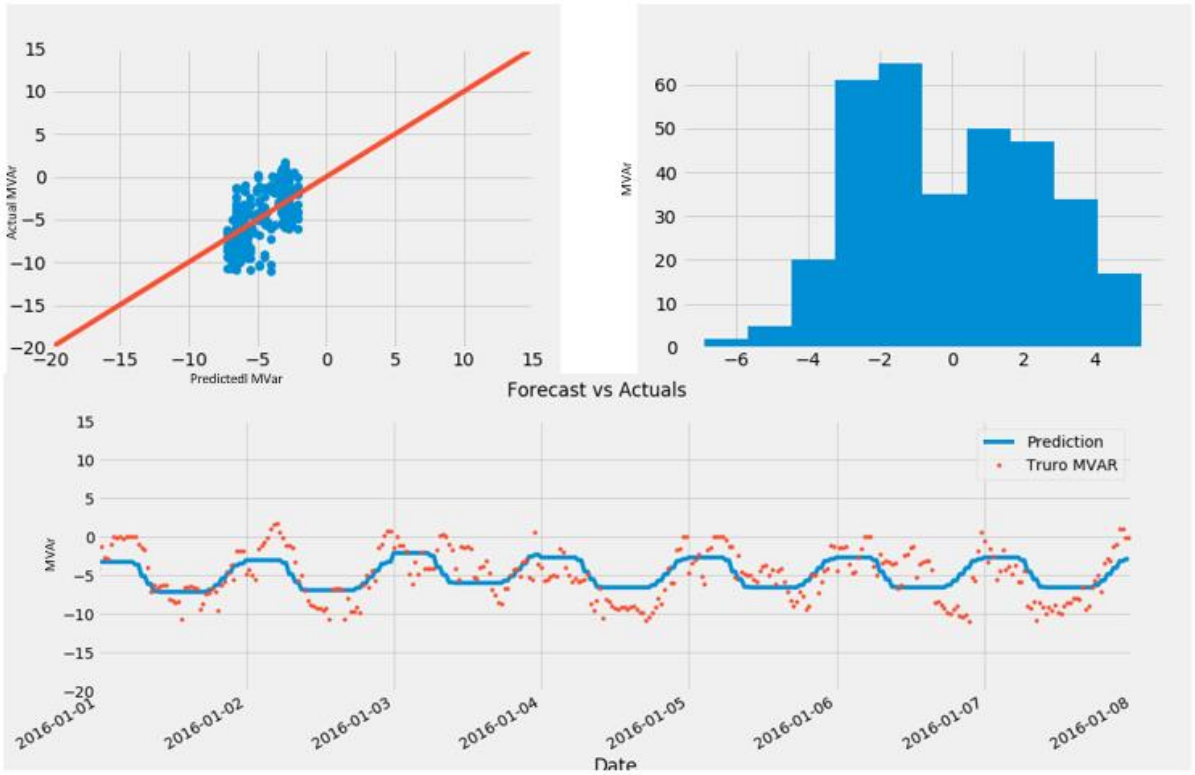


Figure 137: Week Ahead results for reactive power, 1-7 January 2016.

11.4.4. Day Ahead

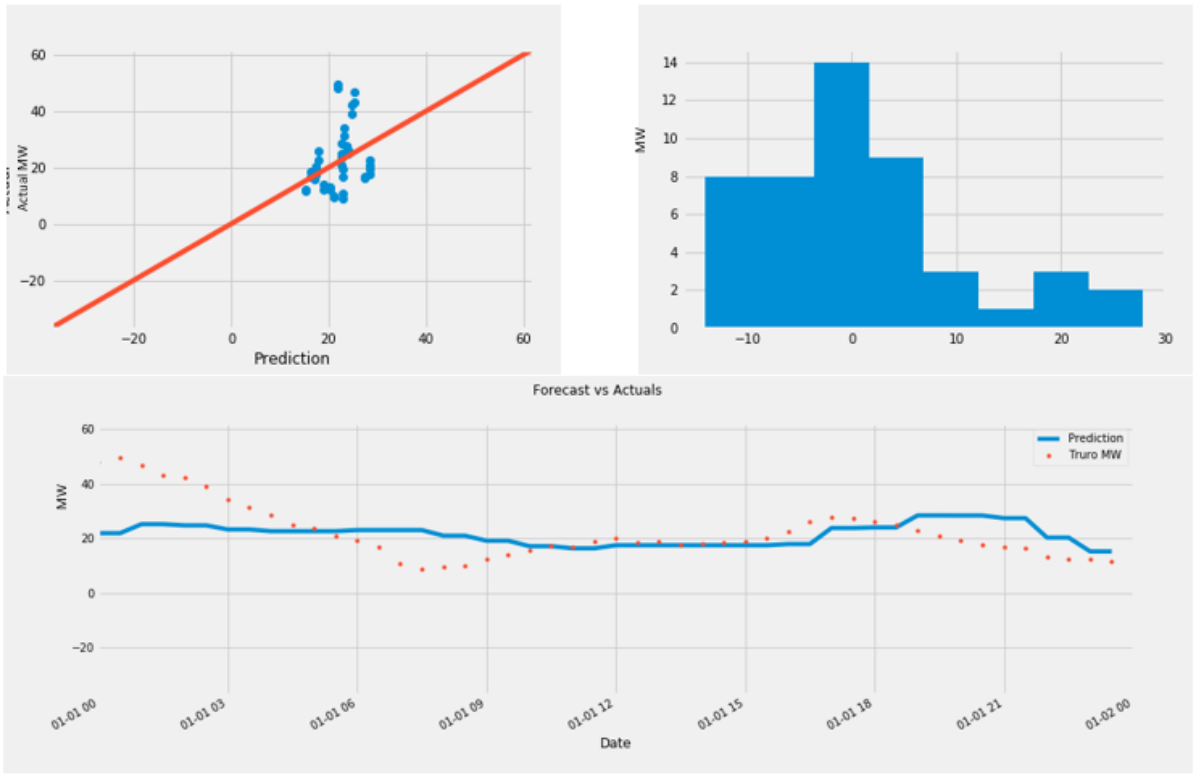


Figure 138: Day Ahead results for real power, 1st January 2016.

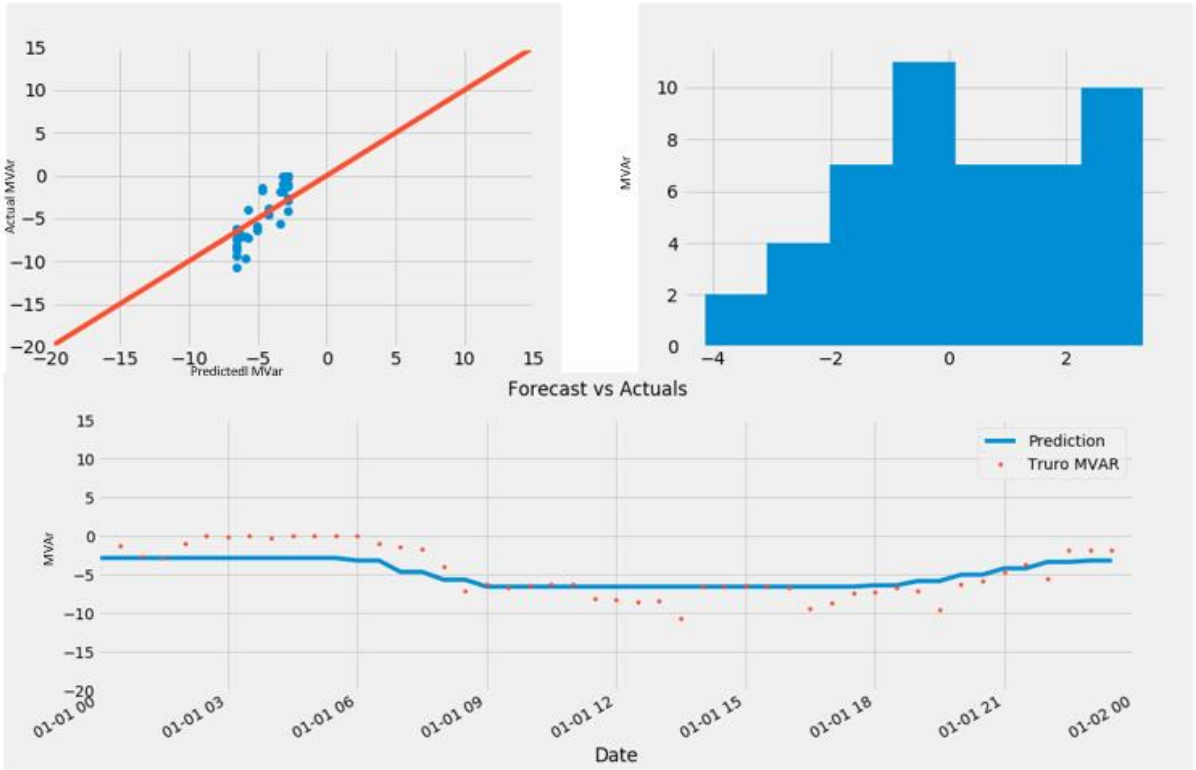


Figure 139: Day Ahead results for reactive power, 1st January 2016.

11.4.5. Hour Ahead

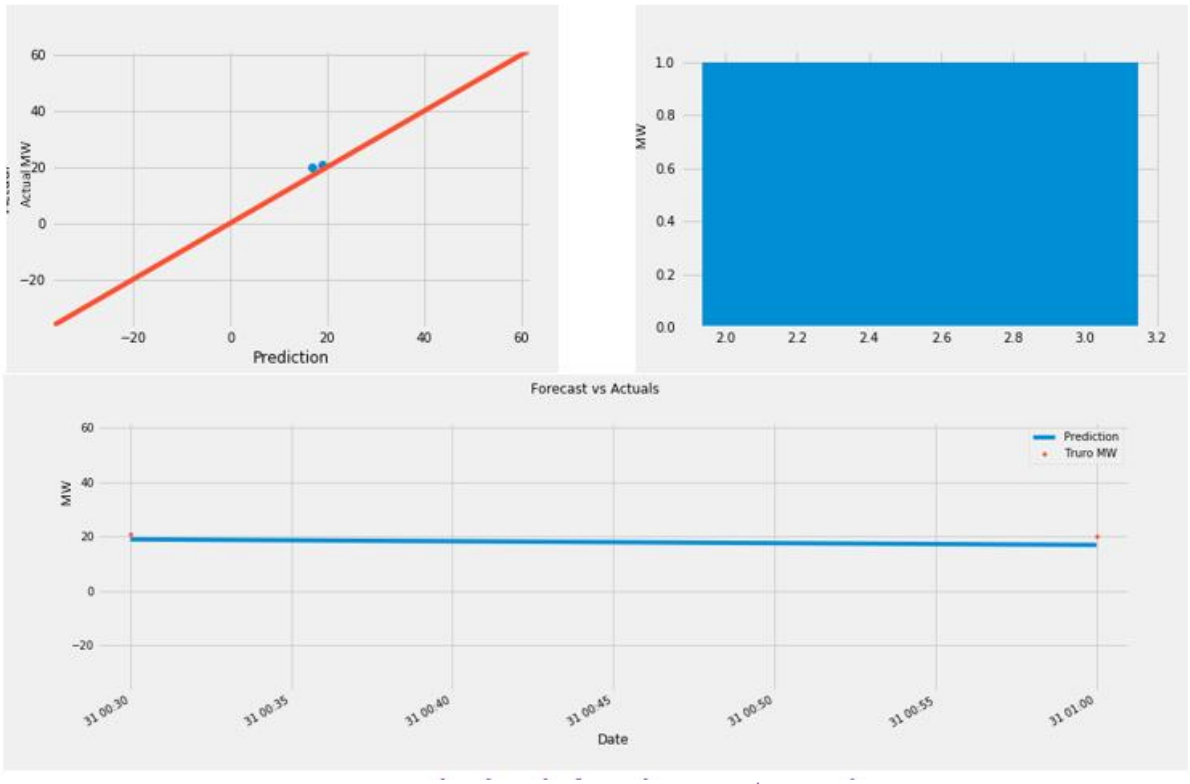


Figure 140: Hour Ahead results for real power, 31st December 2015.

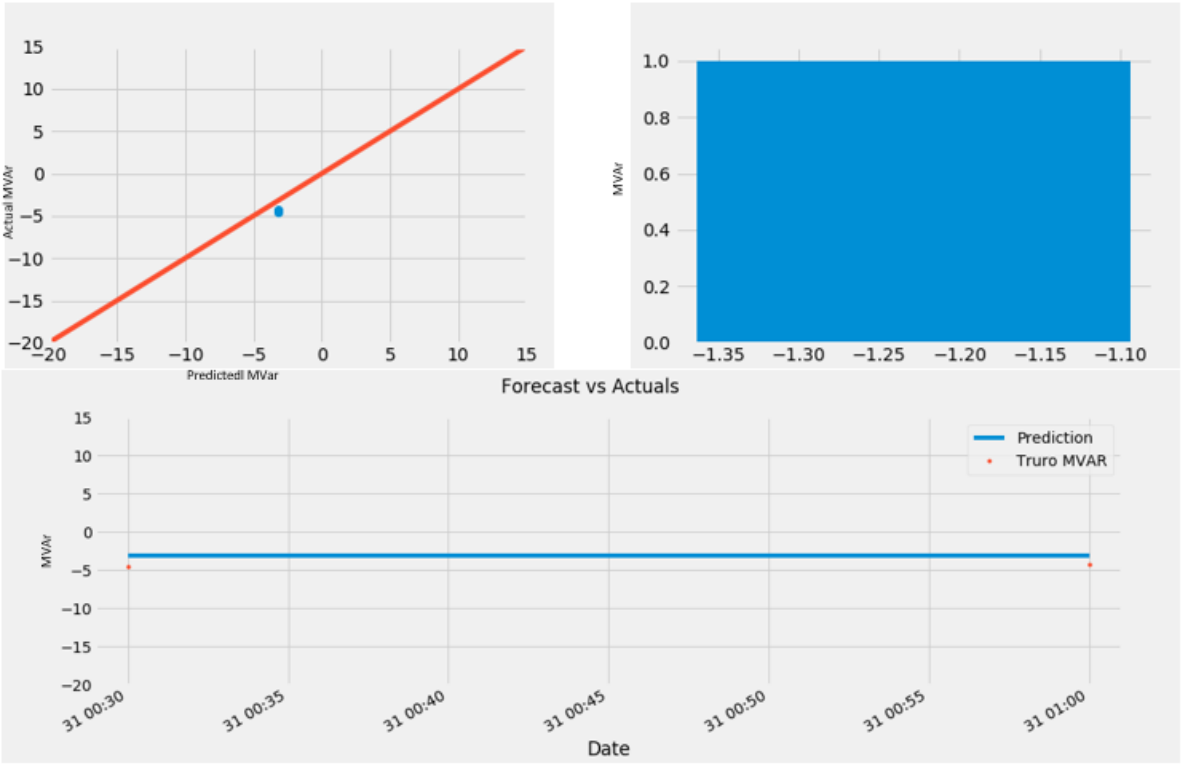


Figure 141: Hour Ahead results for reactive power, 31st December 2015.

11.5. Llynfi Valley

11.5.1. Six Months Ahead

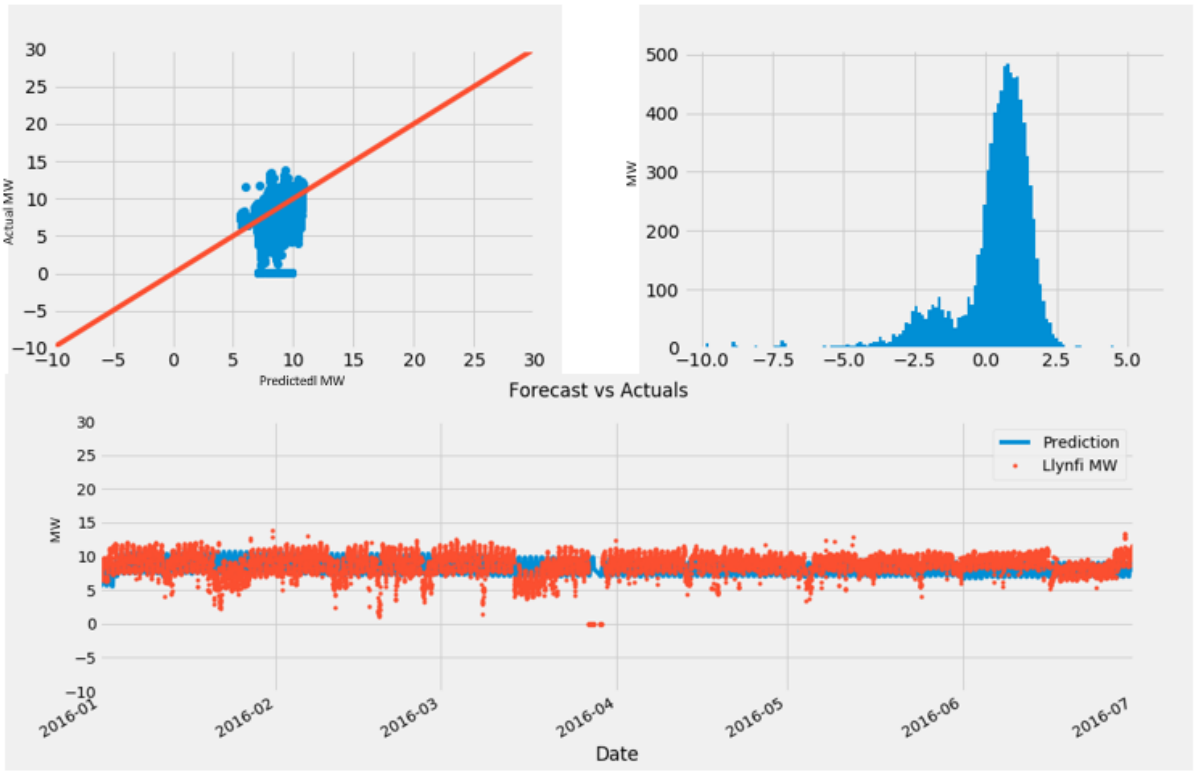


Figure 142: Six Month Ahead results for real power, January-June 2016.

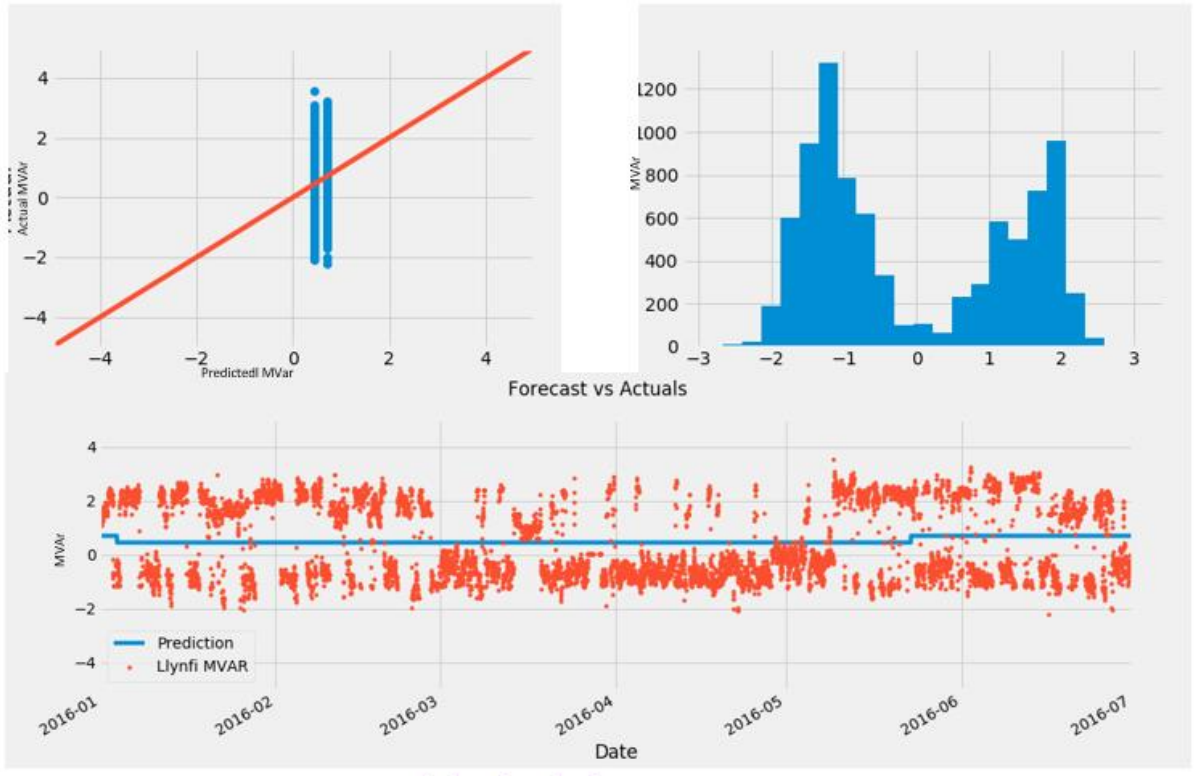


Figure 143: Six Month Ahead results for reactive power, January-June 2016.

11.5.2. Month Ahead

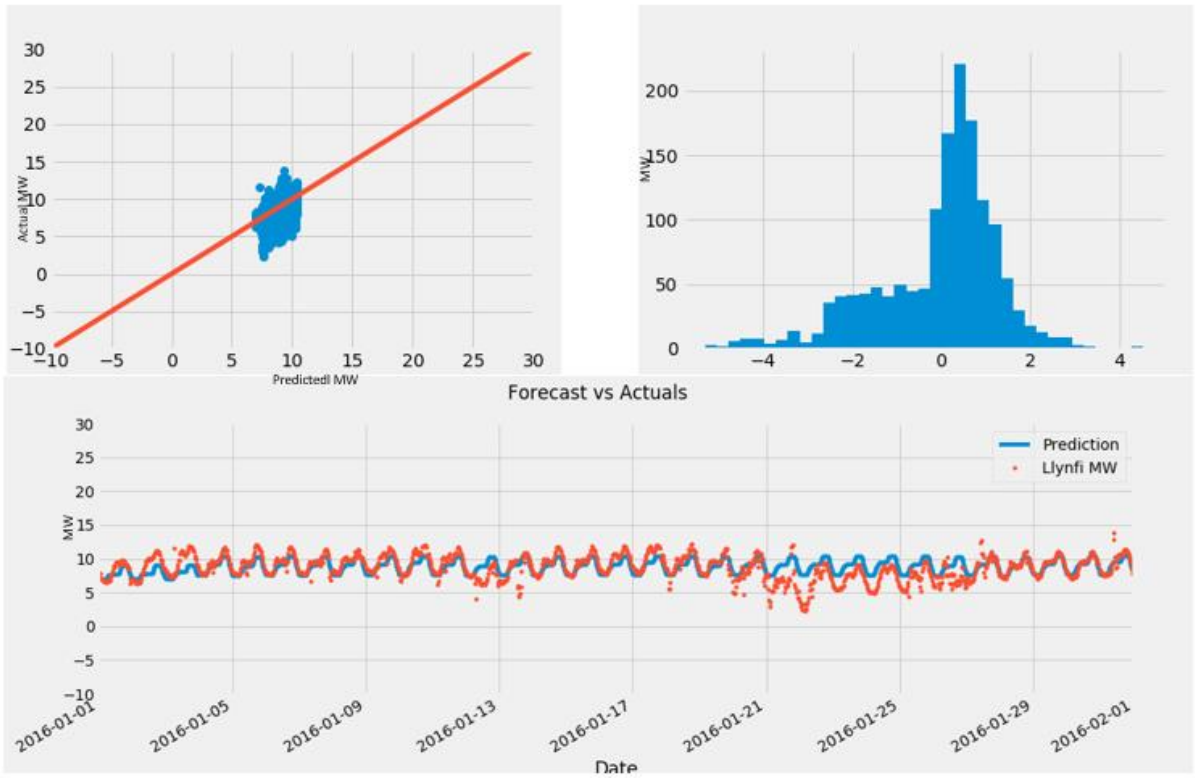


Figure 144: Month Ahead results for real power, January 2016.

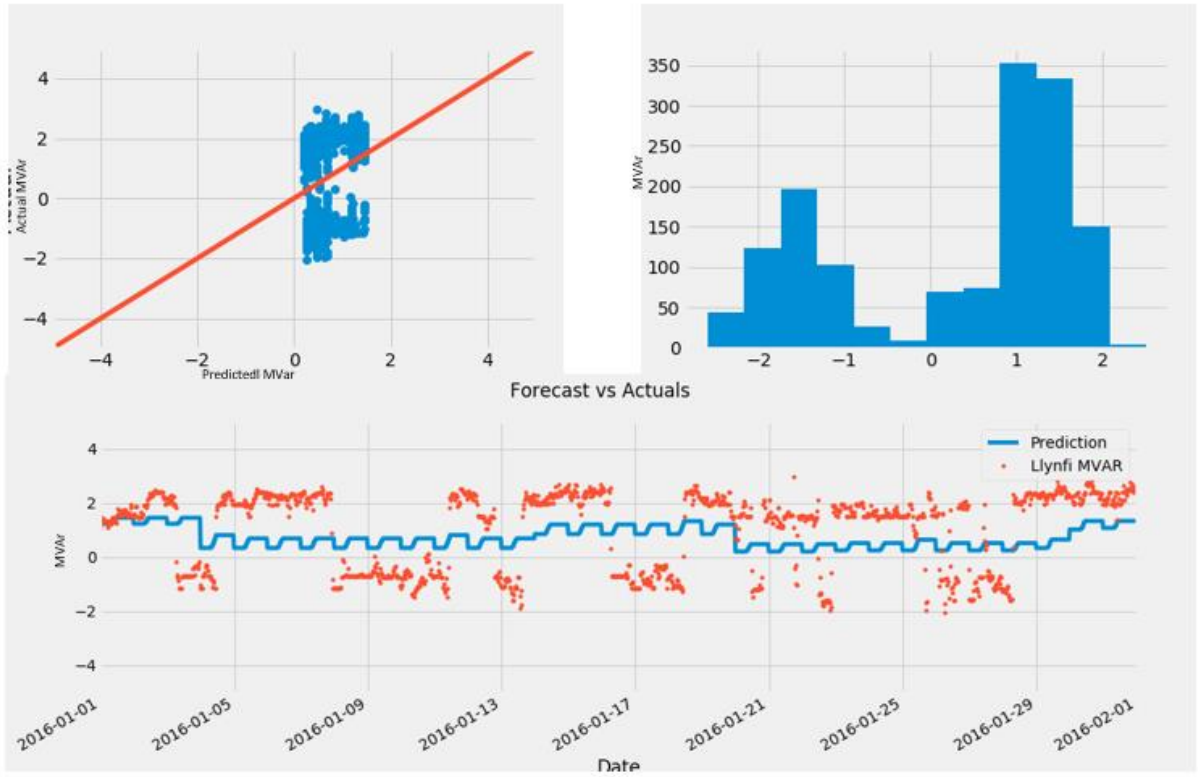


Figure 145: Month Ahead results for reactive power, January 2016.

11.5.3. Week Ahead

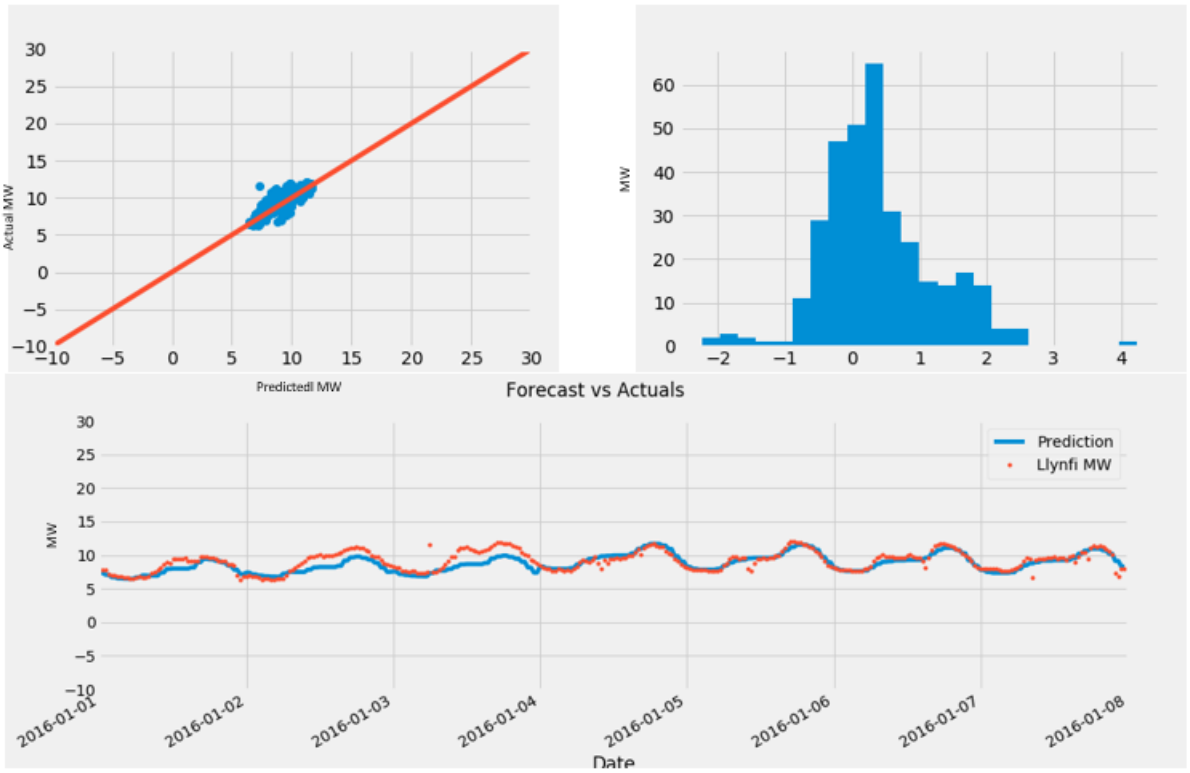


Figure 146: Week Ahead results for real power, 1-7 January 2016.

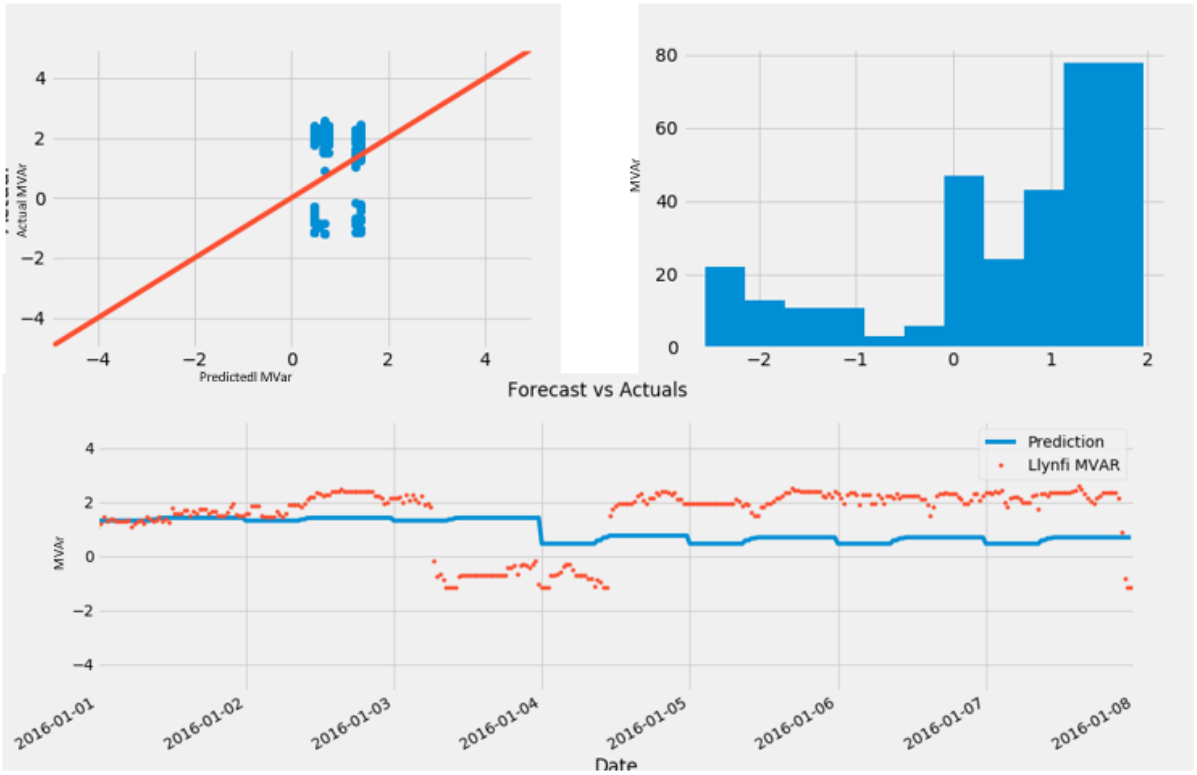


Figure 147: Week Ahead results for reactive power, 1-7 January 2016.

11.5.4. Day Ahead

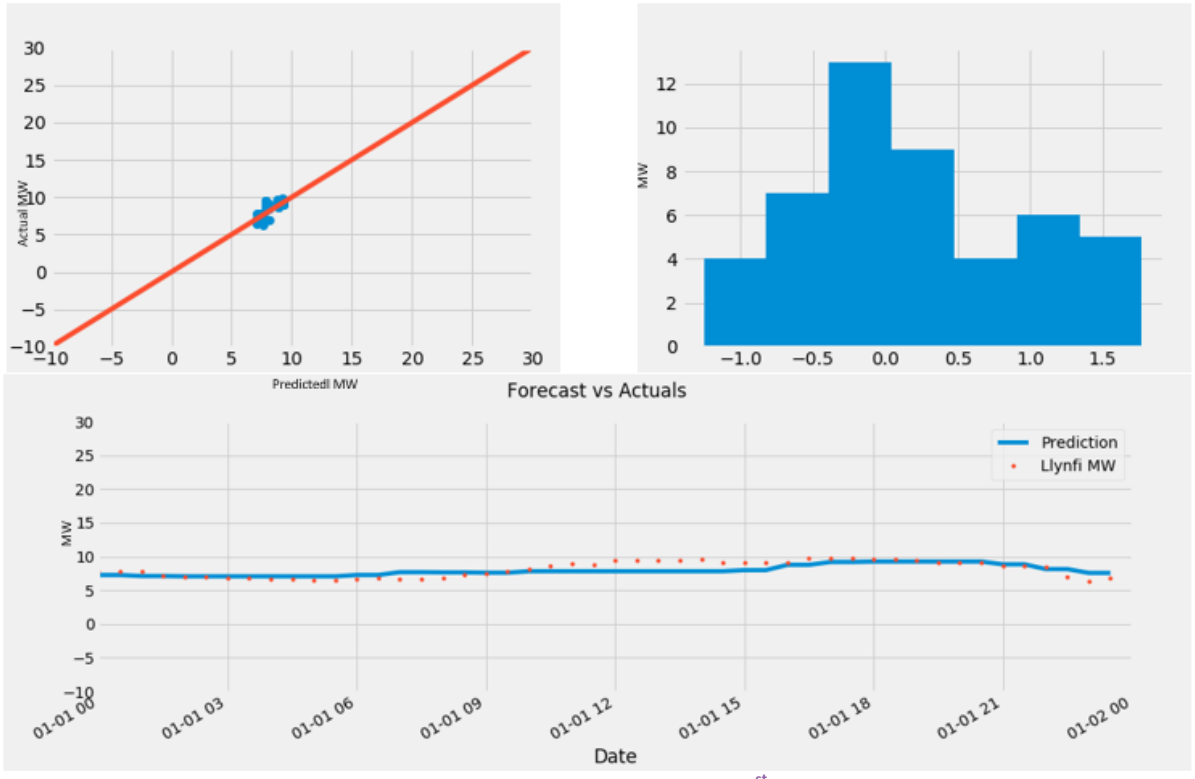


Figure 148: Day Ahead results for real power, 1st January 2016.

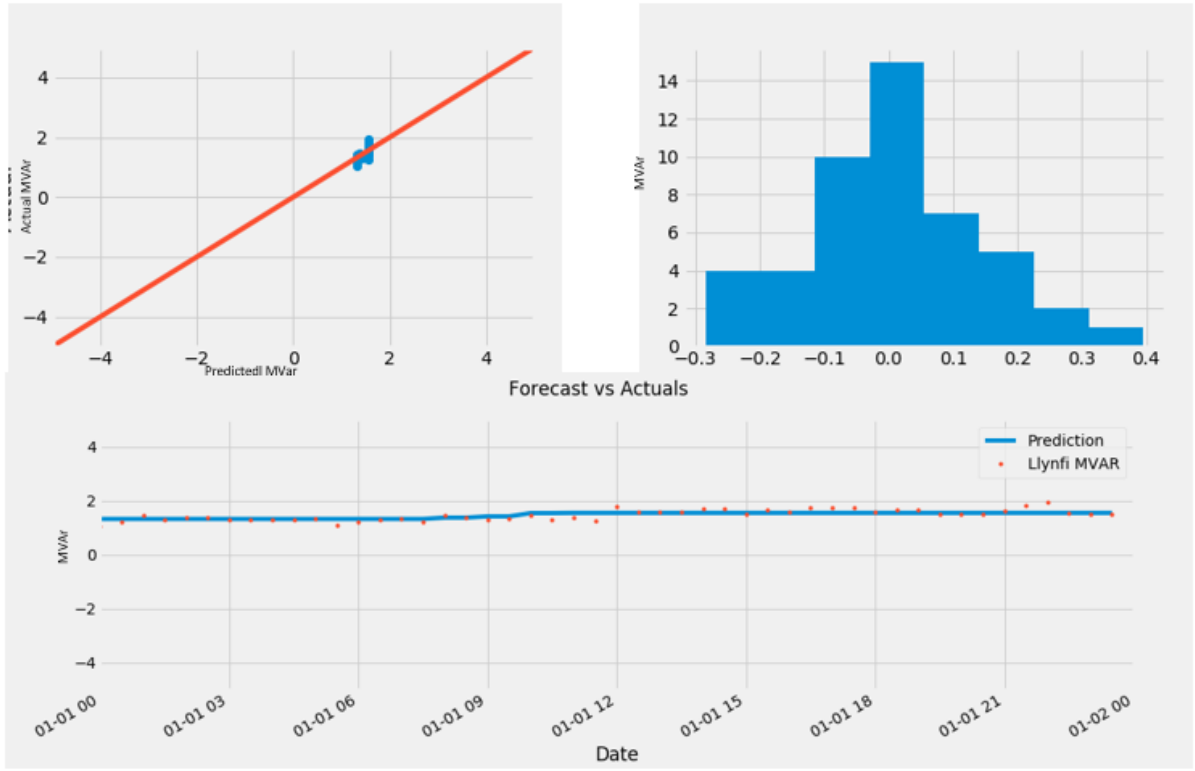


Figure 149: Day Ahead results for reactive power, 1st January 2016.

11.5.5. Hour Ahead

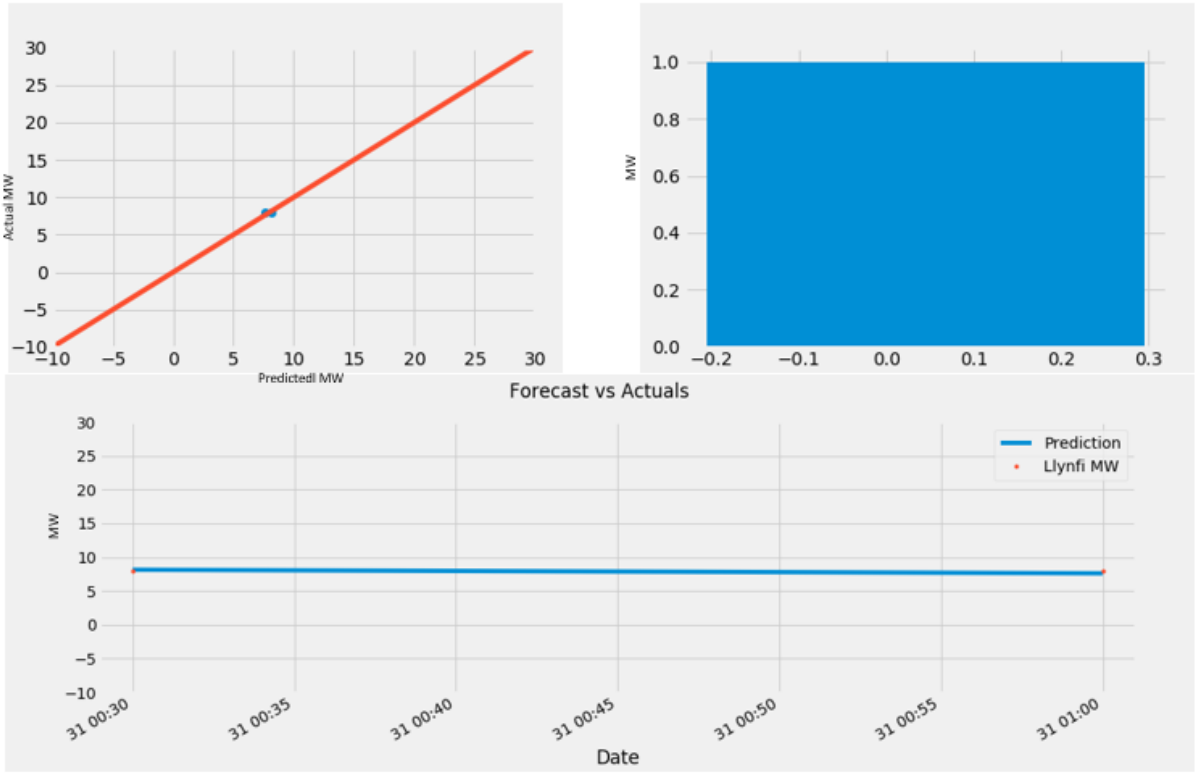


Figure 150: Hour Ahead results for real power, 31st December 2015.

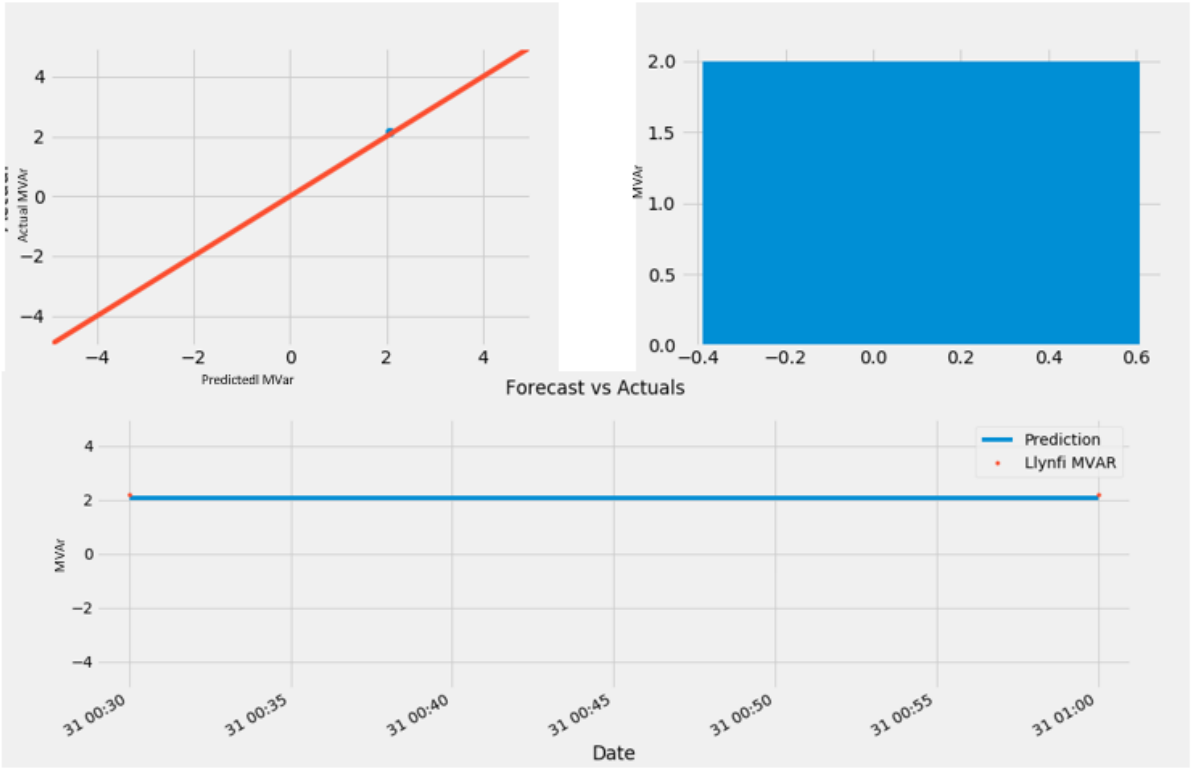


Figure 151: Hour Ahead results for reactive power, 31st December 2015.

11.6. Generator Customer Wind Farm

11.6.1. Six Months Ahead

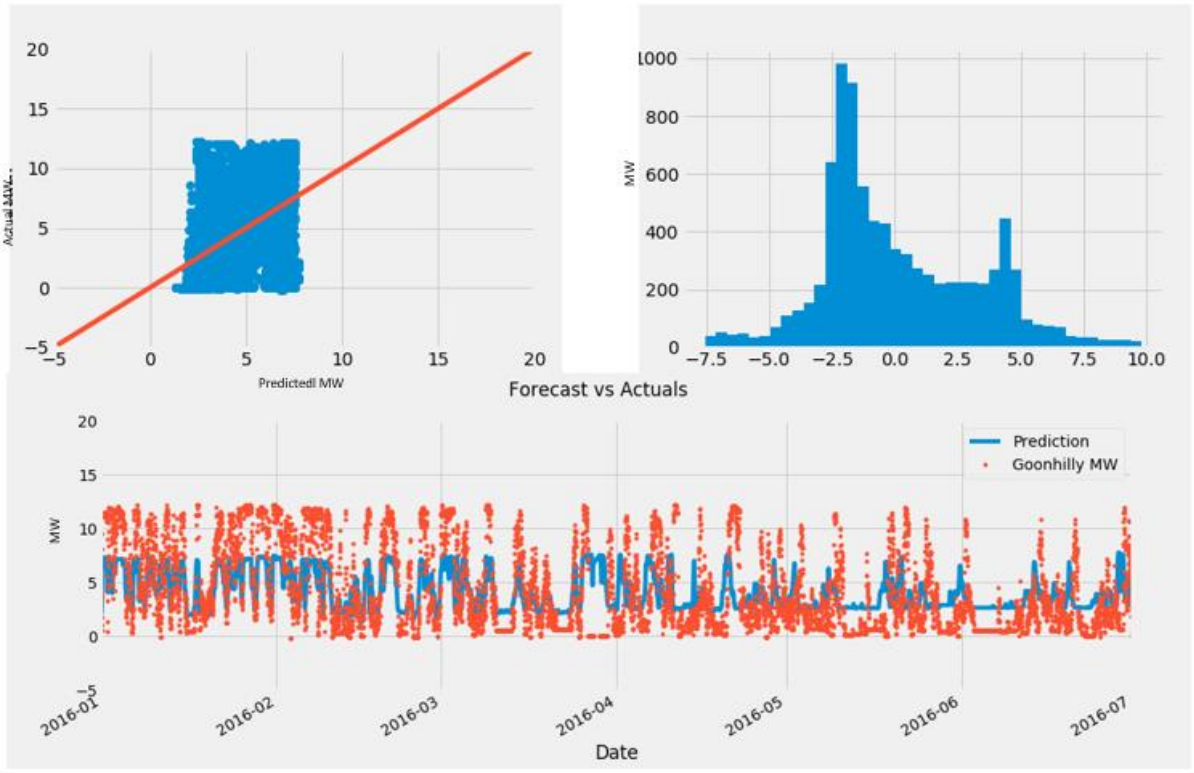


Figure 152: Six Month Ahead results for real power, January-June 2016.

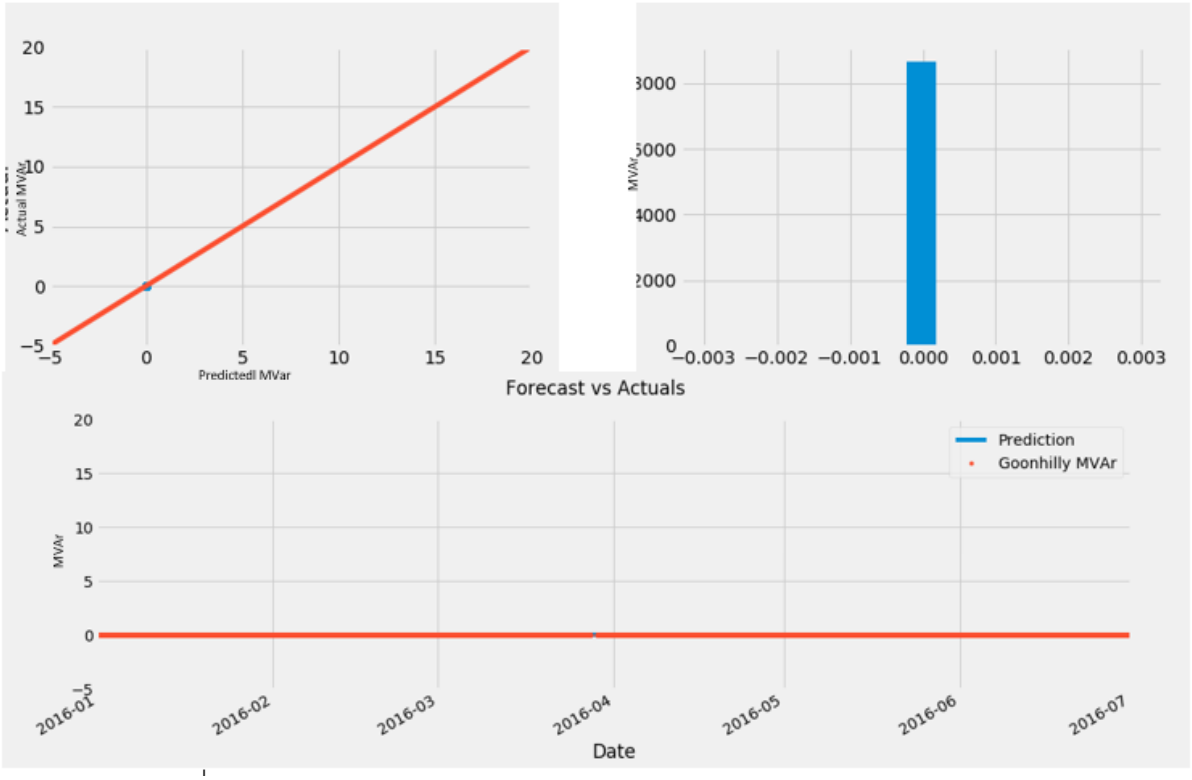


Figure 153: Six Month Ahead results for reactive power, January-June 2016.

11.6.2. Month Ahead

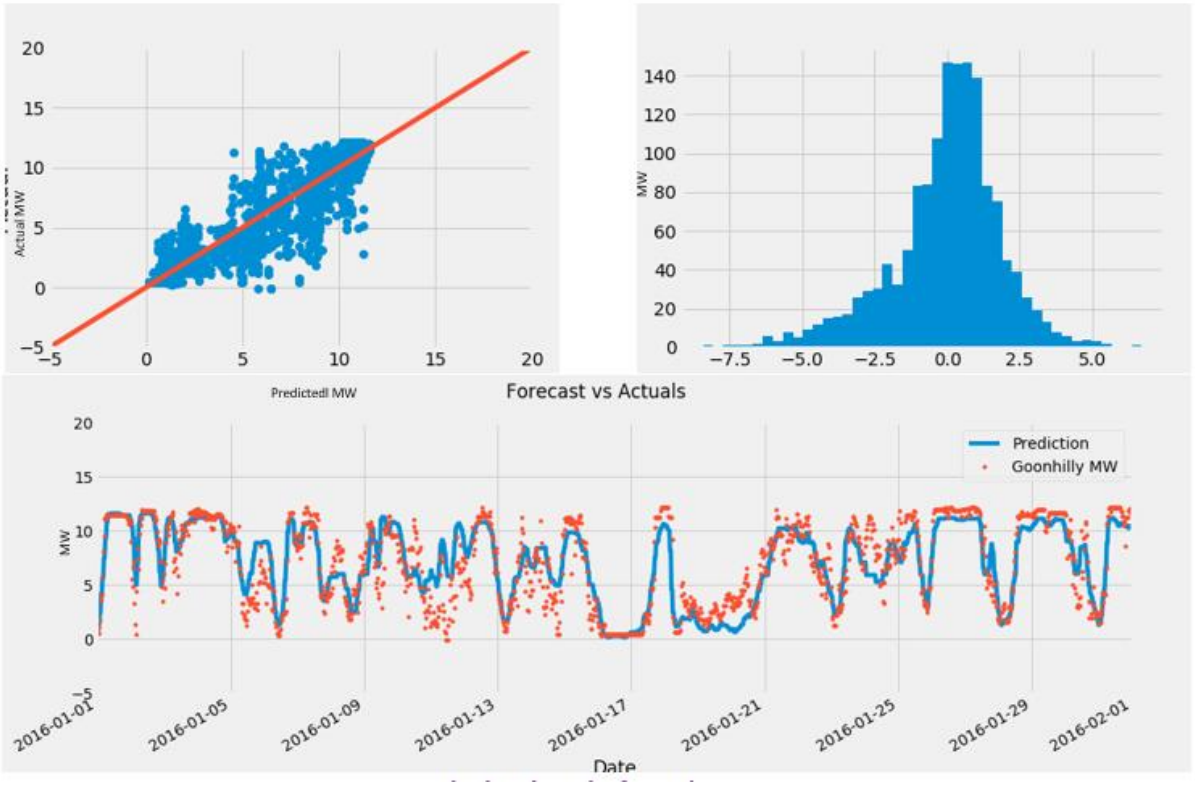


Figure 154: Month Ahead results for real power, January 2016.

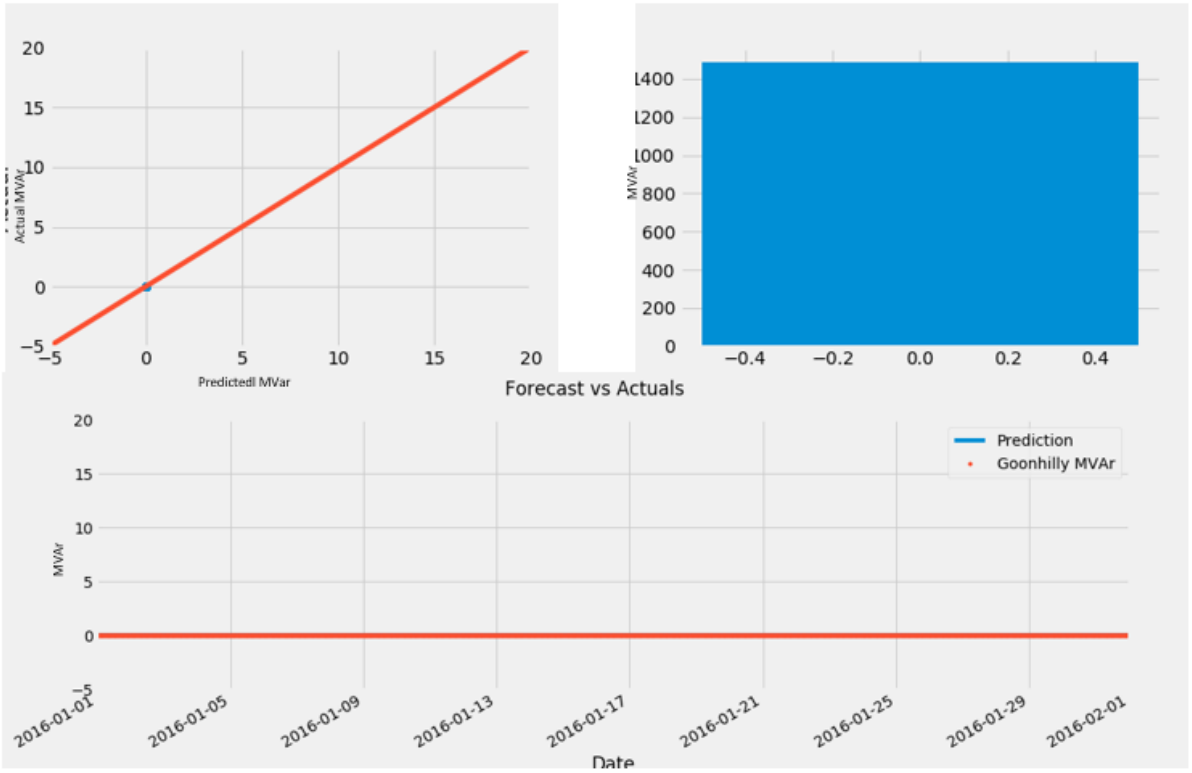


Figure 155: Month Ahead results for reactive power, January 2016.

11.6.3. Week Ahead

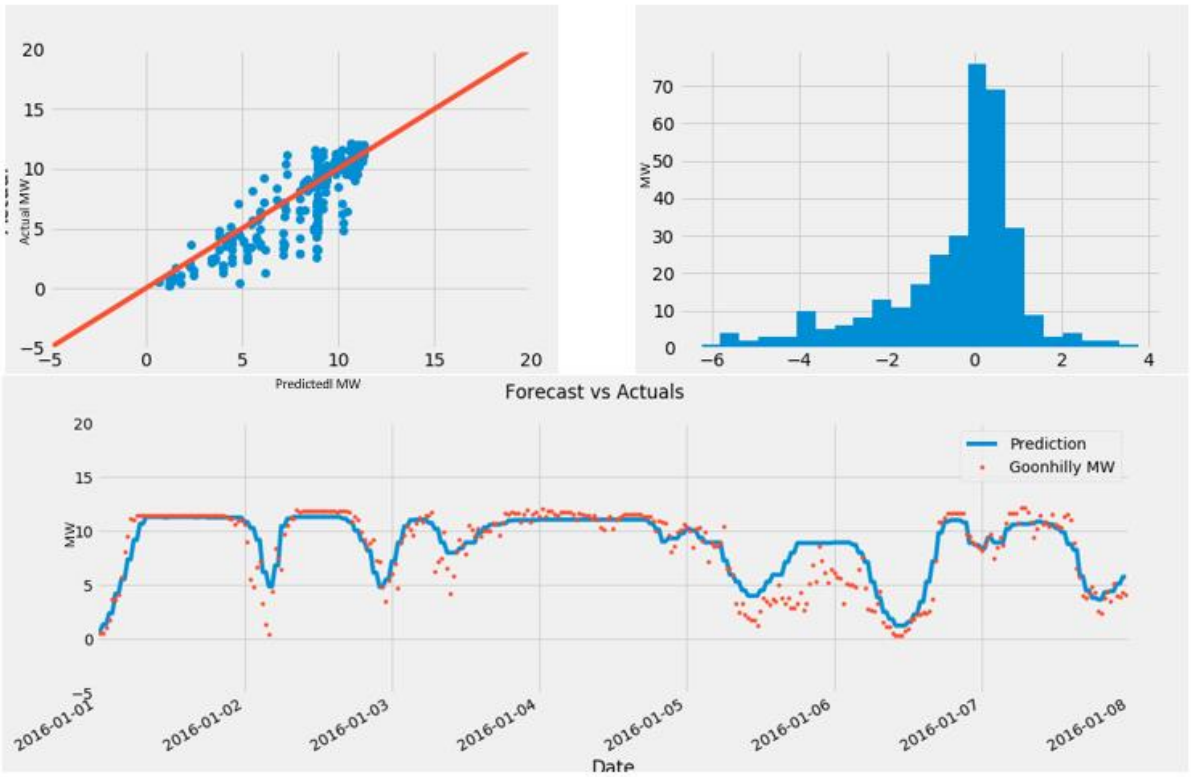


Figure 156: Week Ahead results for real power, 1-7 January 2016.

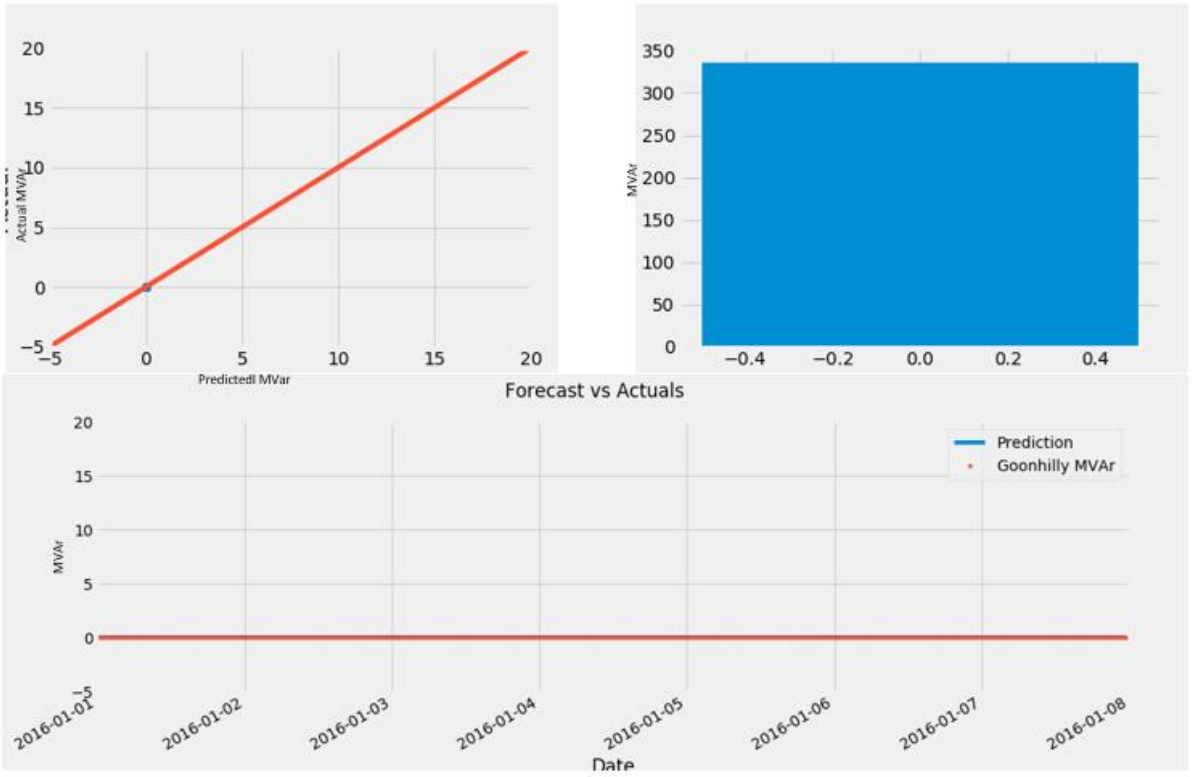


Figure 157: Week Ahead results for reactive power, 1-7 January 2016.

11.6.4. Day Ahead

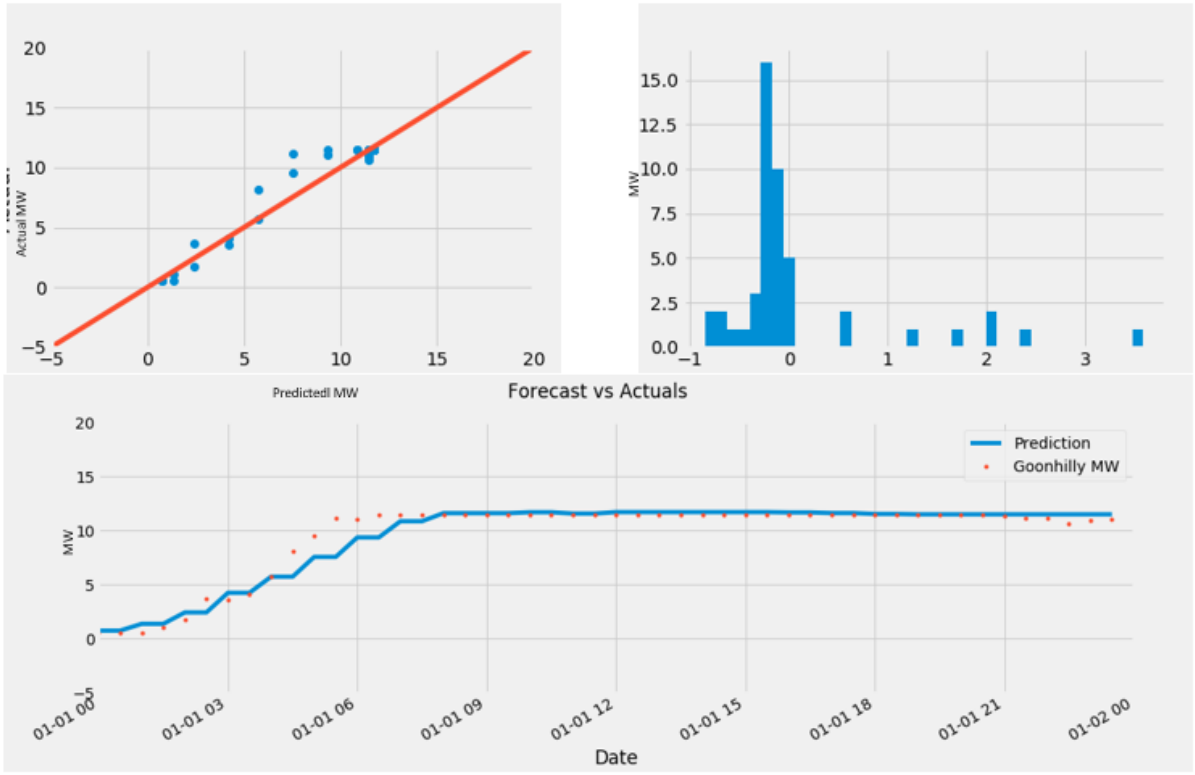


Figure 158: Day Ahead results for real power, 1st January 2016.

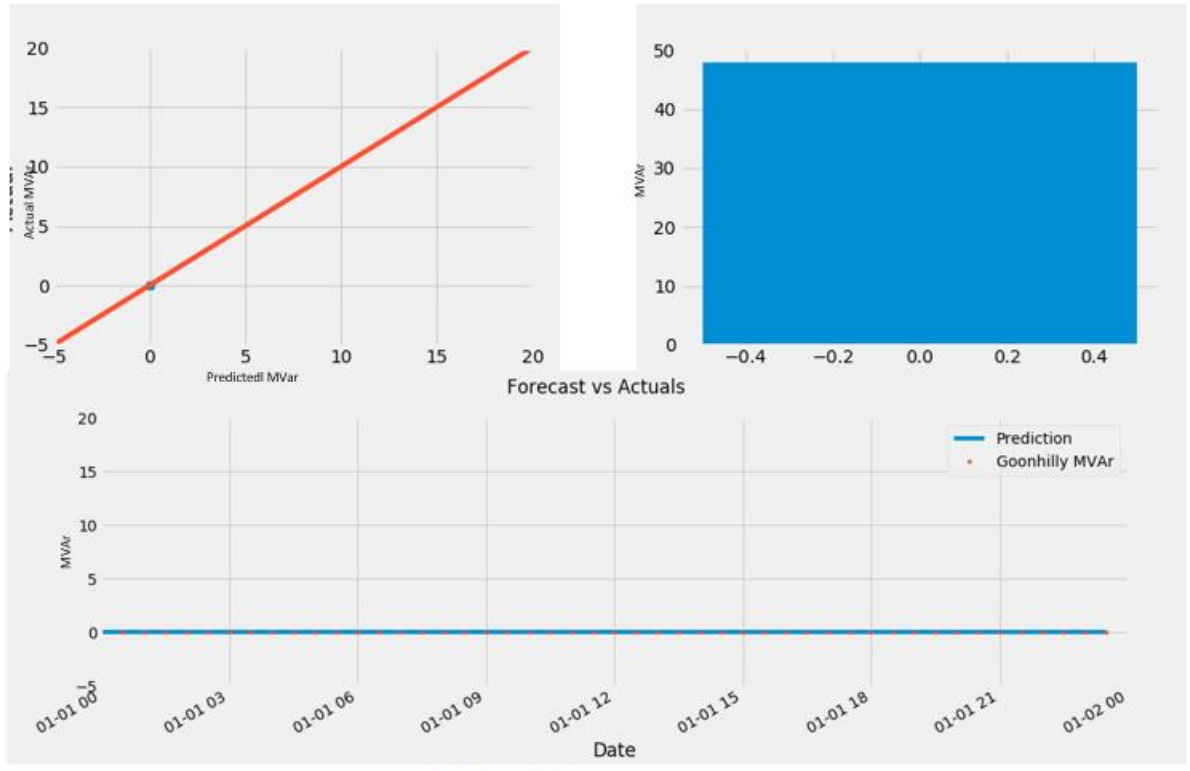


Figure 159: Day Ahead results for reactive power, 1st January 2016.

11.6.5. Hour Ahead

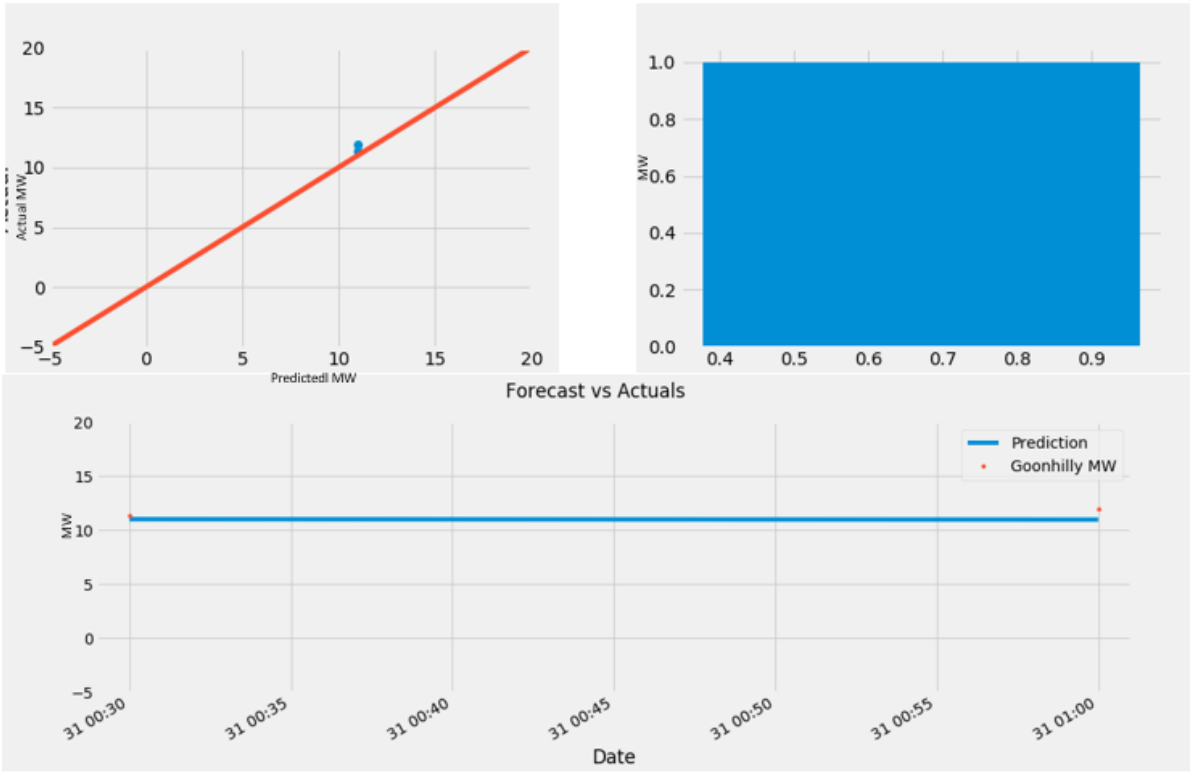


Figure 160: Hour Ahead results for real power, 31st December 2015.

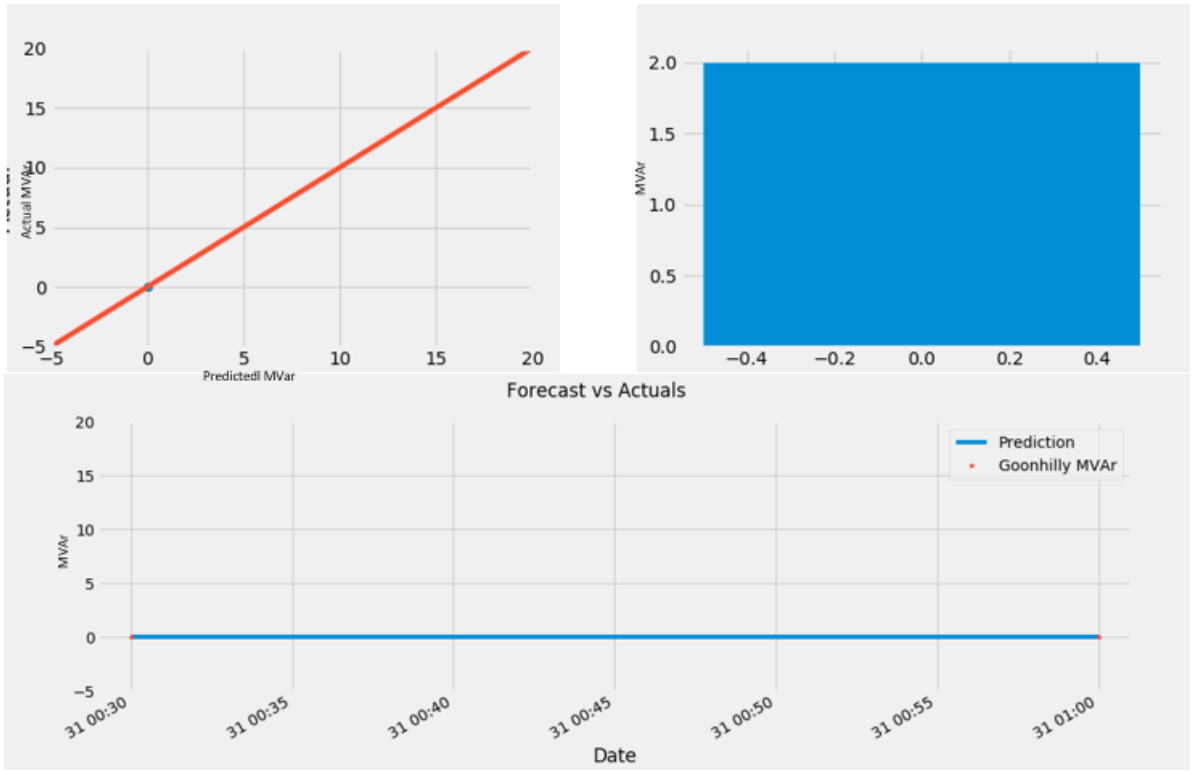


Figure 161: Hour Ahead results for reactive power, 31st December 2015.

11.7. Solar Farm

11.7.1. Six Month Ahead

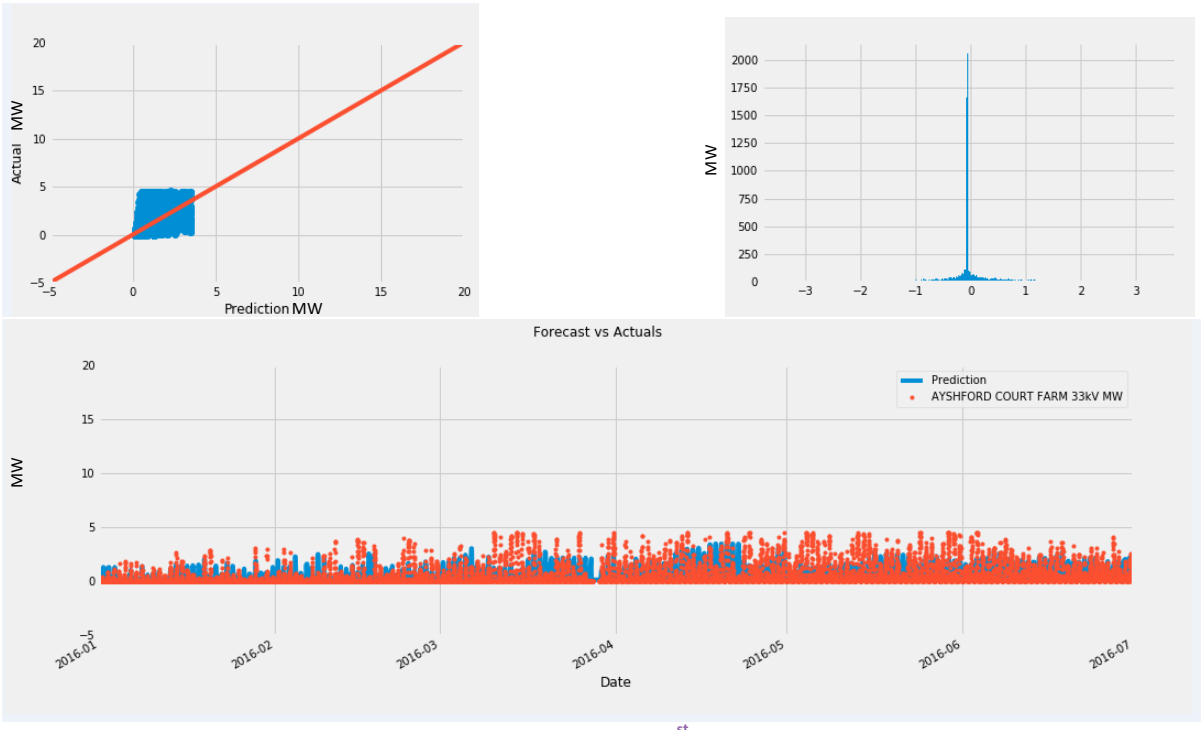


Figure 162: Six Month Ahead :31st December 2016

11.7.2. Day Ahead

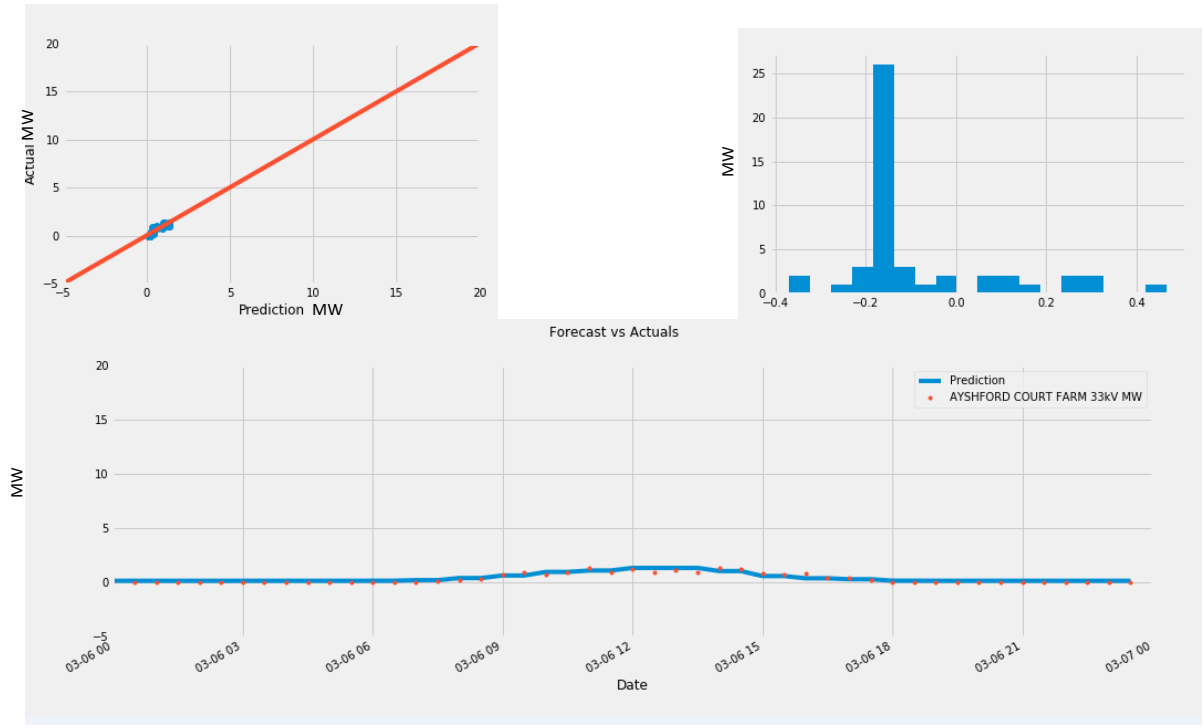


Figure 163:Day Ahead 3rd June 2016

11.8. Load Customer

11.8.1. Month Ahead

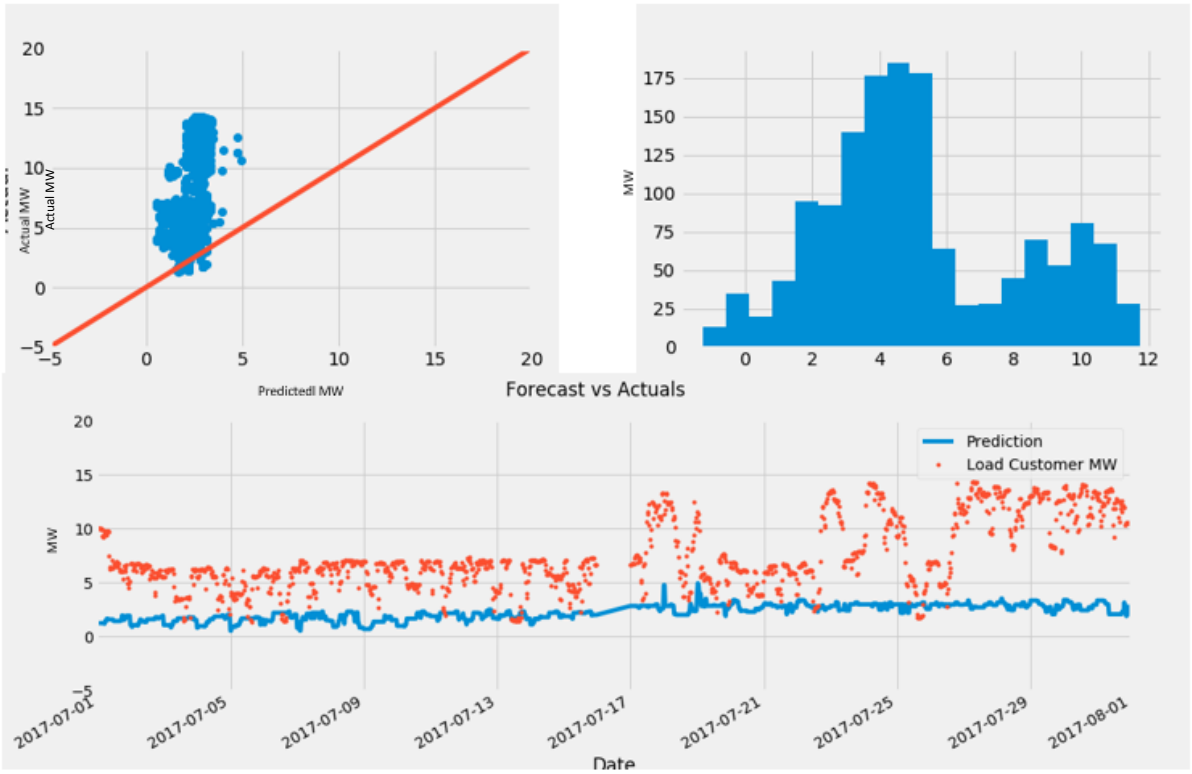


Figure 164: Month Ahead results for real power, July 2017.

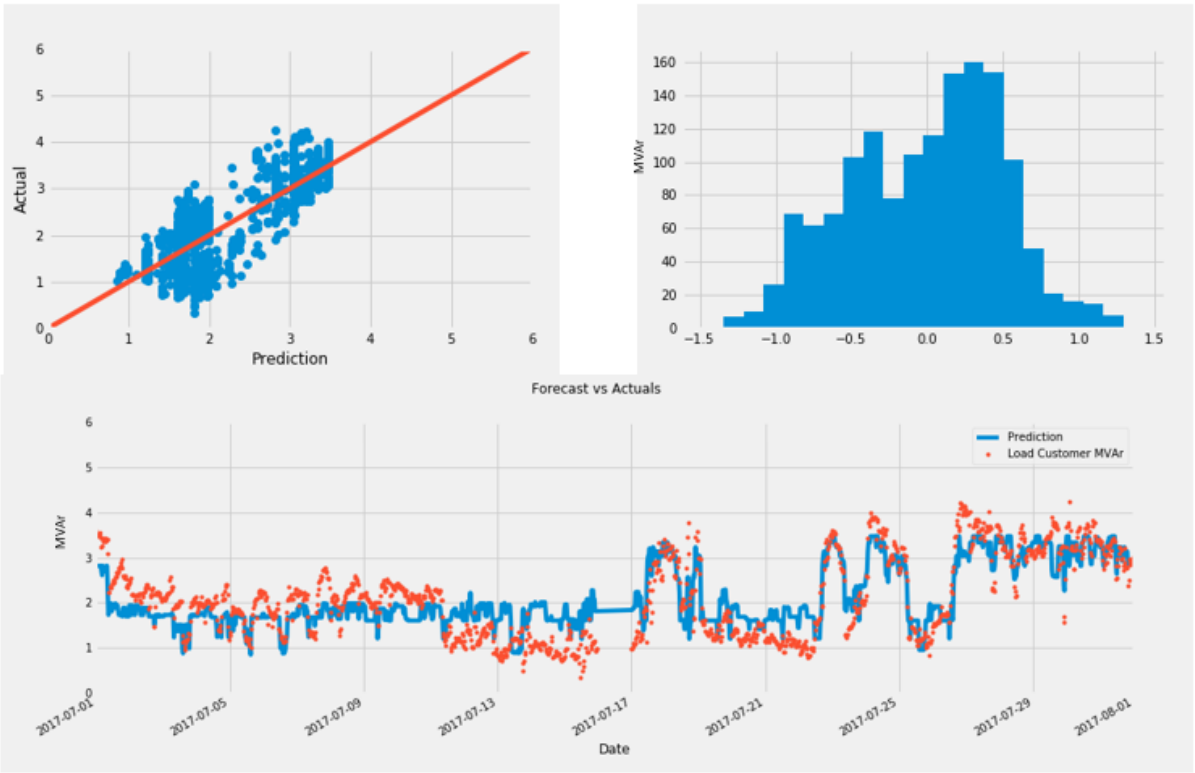


Figure 165: Month Ahead results for reactive power, July 2017.

11.8.2. Week Ahead

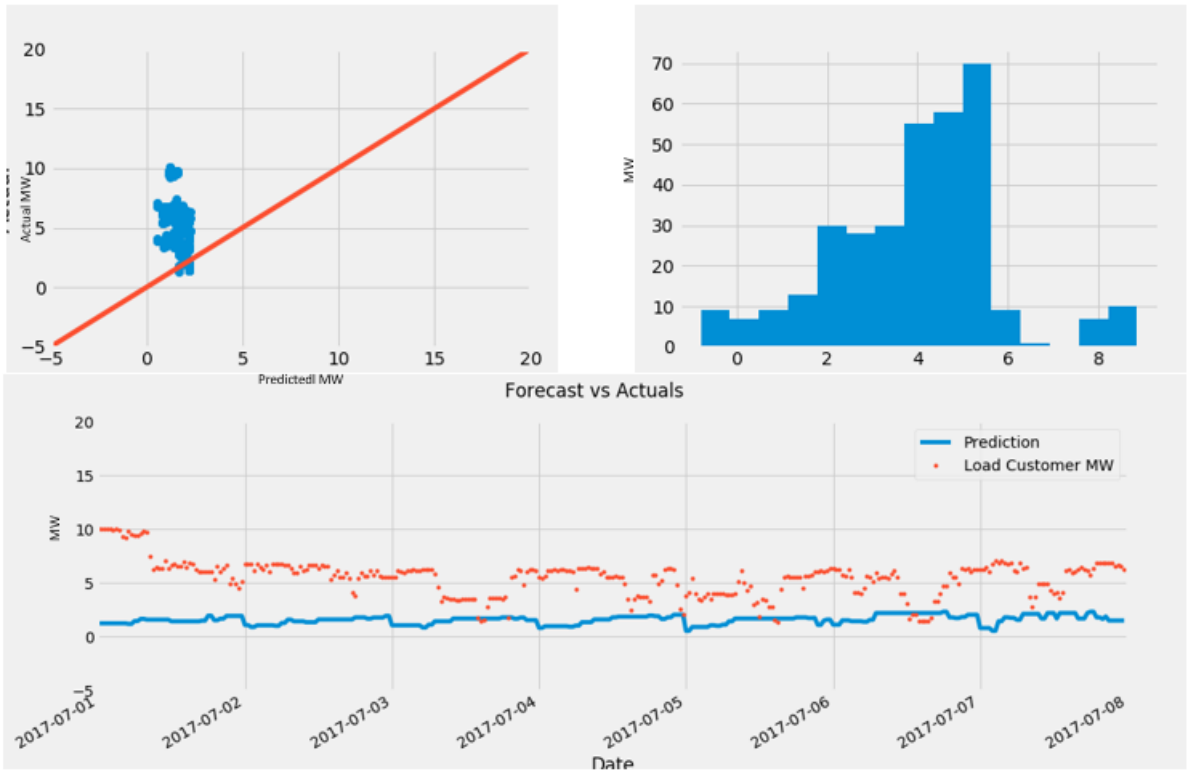


Figure 166: Week Ahead results for real power, 1-7 July 2017.

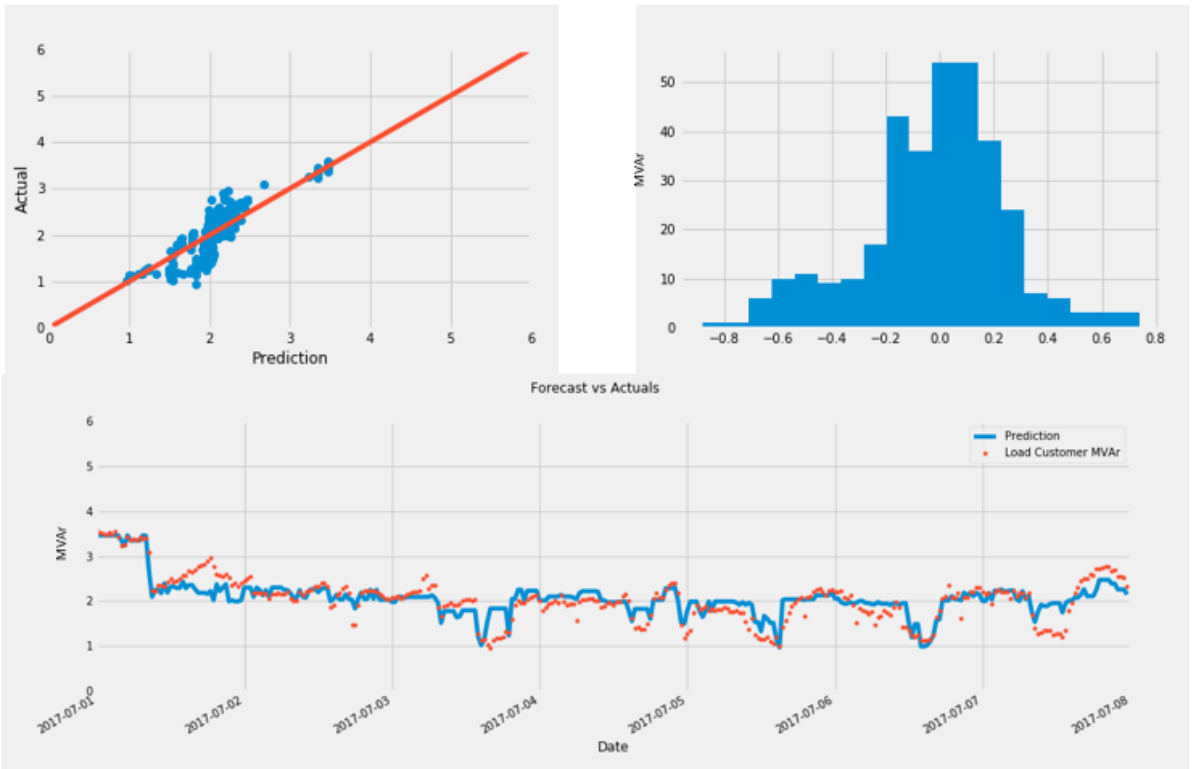


Figure 167: Week Ahead results for reactive power, 1-7 July 2017.

11.8.3. Day Ahead

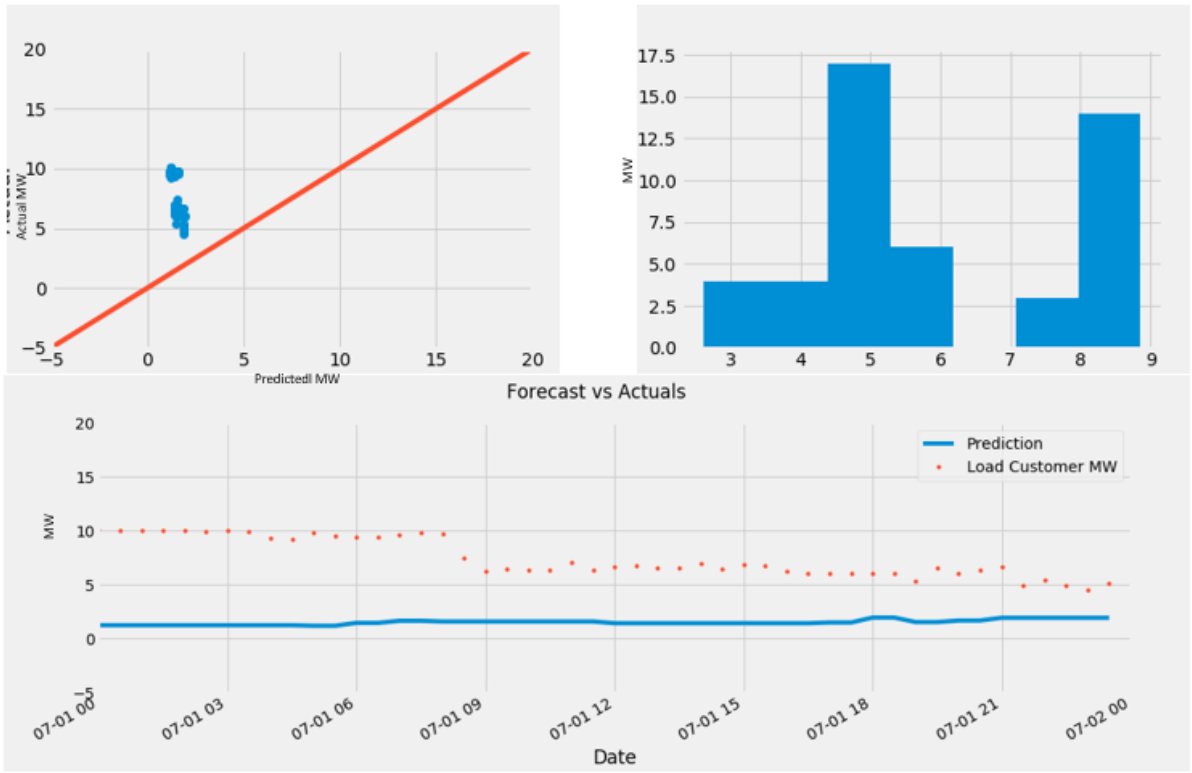


Figure 168: Day Ahead results for real power, 1st July 2017.

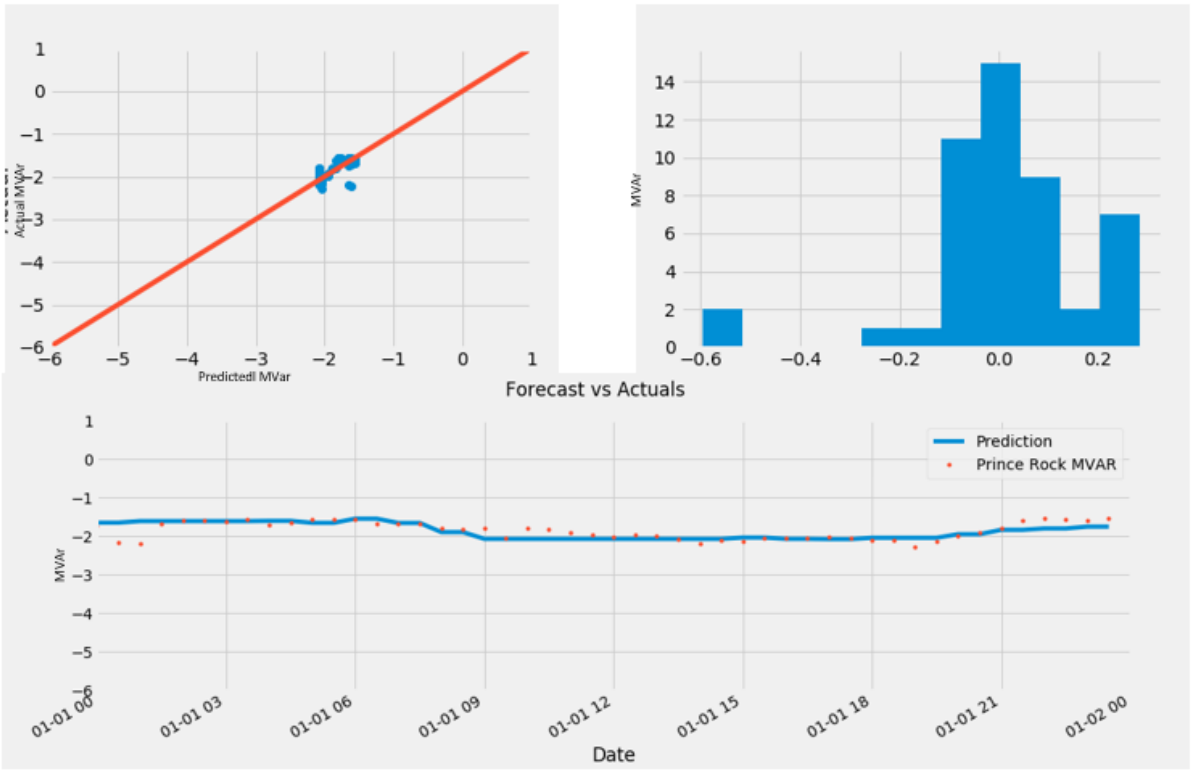


Figure 169: Day Ahead results for reactive power, 1st July 2017.

11.8.4. Hour Ahead

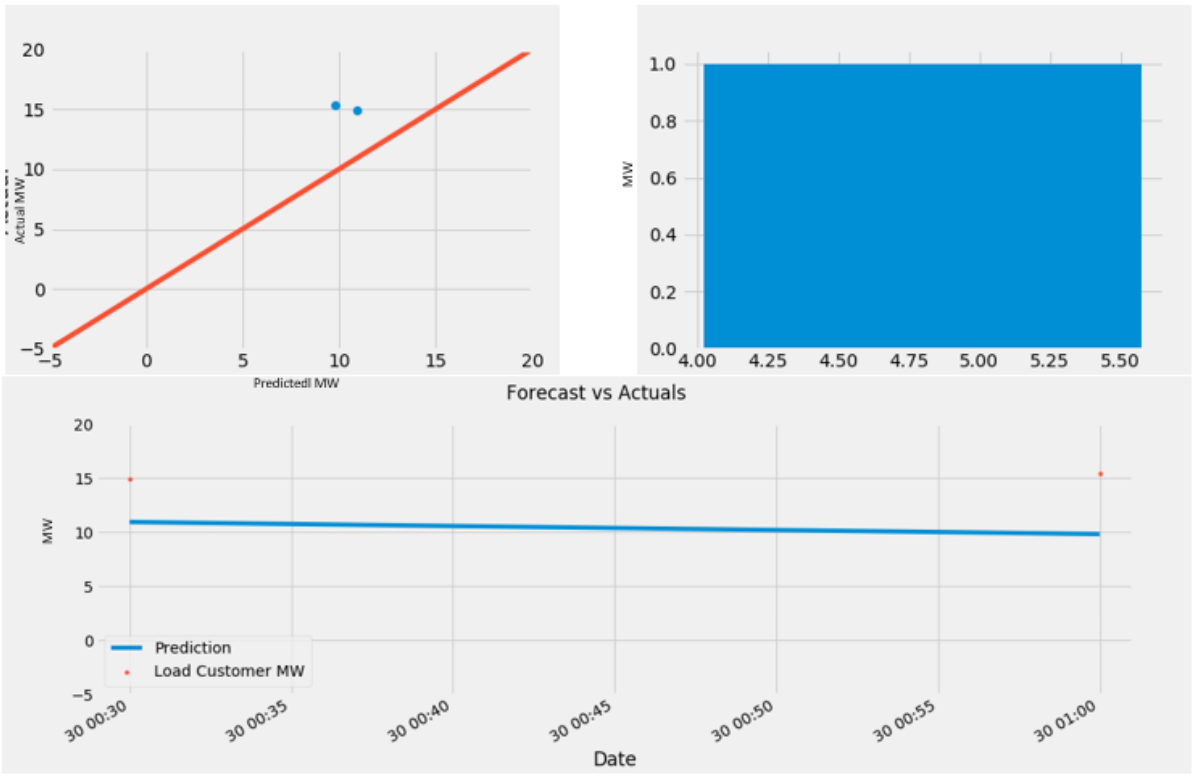


Figure 170: Hour Ahead results for real power, 30th June 2017.

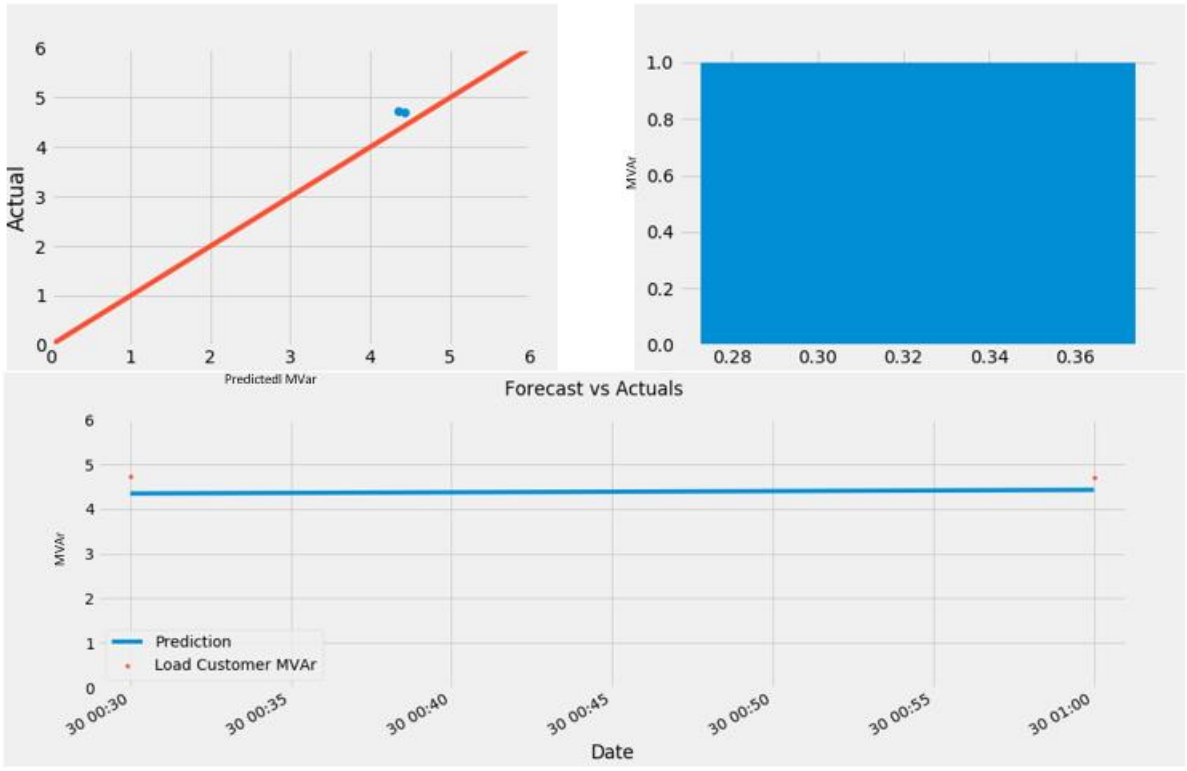


Figure 171: Hour Ahead results for reactive power, 30th June 2017.