HYBRID MODEL APPROACHES TO IMPROVE SHORT-TERM ENERGY DEMAND FORECASTS IN NEW SOUTH WALES, AUSTRALIA

David Valido Ramos (z5516338), Katelyn Kemp (z5459347), Nick Mutton (z5549371), Senarath Seelanatha (z5595581), Shanjay Perinpanathan (z5339723), Waseem Alashqar (z5514810).

School of Mathematics and Statistics
UNSW Sydney

April 2025

Submitted in partial fulfilment of the requirements of the capstone course ZZSC9020

Abstract

Electricity demand forecasting is a difficult problem every country faces. In this paper, we attempt to utilise the concept of hybrid models to improve the energy forecast of AEMO, the Australian body that manages power systems and markets, to predict energy demand in NSW. It was found that historical forecast errors and weather variables had some correlation with forecasting errors, therefore were included in the models. SARIMA, Random Forest, and XGBoost models were tested to determine the best fit for correcting AEMO forecasting bias and reducing overall energy demand forecast error. We argue the hybrid modification enables us to correctly factor relationships not supported by AEMO's original model. All hybrid models tested provided some reduction in the overall forecast error and supported the hybrid model process.

Contents

Chapter 1
Introduction

Energy forecasts play a crucial role in planning and maintaining the energy sector. Ensuring forecast accuracy helps to manage imbalances in energy production and consumption, reduce power system costs, and improve operational safety (Mystakidis et al., 2024). Energy demand management is also linked with self sufficiency and cost effectiveness that facilitate sustainable economic development (Suganthi and Samiel, 2012). Energy forecasting therefore has a broad impact on a wide variety of stakeholders including residential customers, power generators, retailers, traders, industrial and commercial customers, system operators, and financial investors (Ghalehkhondabi et al., 2016).

There are many risks in inaccurate energy forecasting. Over forecasting has cost and resource implications for providers, as well as environmental impacts. Under forecasting can cause outages, as well as having down the stream increased costs from inconsistent supply (Suganthi and Samuel, 2012). Shortages are also linked to political instability (Rakpho and Yamaka, 2021).

The Australian Energy Market Operator (AEMO) is responsible for managing Australia's electricity and gas systems and markets to ensure Australians have access to reliable, affordable and secure energy. AEMO performs a wide range of functions, however one of their key roles is to balance electricity supply and demand through dispatching electricity generation based on forecasts updated every 5 minutes. It is therefore critical that their electricity demand forecasting is accurate to reduce the risks associated with over, or under supply of electricity to the market. While the AEMO short-term electricity forecast is generally quite accurate, it is valuable to understand where the forecast may be underperforming to consider how accuracy could be improved.

The goal of this report is to identify variables that may contribute to errors in AEMO's electricity demand forecasts, with the aim of using these insights to improve forecast accuracy.

The report will consider a 12-hour interval with a step-ahead forecast. This is considered a 'pre-dispatch' interval based on AEMO's definitions and is important for operational planning, therefore its accuracy is critical.

Chapter 2

Literature Review

2.1 Forecasting electricity demand background

In today's context of near-constant energy consumption, the task of energy forecasting has become increasingly complex. Given the absence of a universally applicable forecasting method, the selection of an appropriate technique is typically guided by the nature of the available data and the specific objectives of the forecasting exercise (Pinheiro, Madeira, & Francisco, 2023). Additionally, the forecast interval, which often reflects the purpose of the forecast, plays a large role in determining the suitability of different modeling approaches.

Forecasting models are typically categorised into short-, medium-, and long-term, and while there is not a unanimous definition of what constitutes these time periods, researchers generally agree that short-term is a few minutes up to a few days (Ahmad and Chen, 2018) or two weeks (Klyuev et al., 2022), medium-term as one month to one year, and long-term as one year to ten years (Ahmad and Chen, 2018). AEMO defines its short term forecast as up to 7 days ahead (AEMO, 2023).

Short-term intervals tend to require the greatest accuracy as they support a wide variety of operational planning, or network management activities including scheduling, planning of power generation, cost optimisation and guaranteeing continuous electricity supply (Sanhudo, Rodrigues and Filho, 2021). Short-term forecast methods can be broadly categorised into two categories – mathematical algorithms such as time-series analysis and logistic regression, and artificial intelligence (AI) algorithms such as machine learning, deep learning and ensemble learning models (Deng et al., 2022). For short-term forecasting, AI methods are becoming more popular as they can consider the non-linear nature of power demand. Short term forecasting is also generally more interested in the accuracy of the forecast rather than the interpretability of the results which makes these 'black box' approaches appropriate (Phyo and Byun, 2021). Other studies have found that machine learning models tend to outperform traditional models such as ARIMA in short-term forecasting (Divina et al., 2019).

Medium- and long-term forecasting supports the planning and maintenance of the electrical network such as smart grid eco-systems (Ahmad & Chen, 2018). Furthermore, long-term forecasting is more strategic and is necessary for the development of energy systems, planning capital construction at production or infrastructure facilities (Klyuev et al., 2022). These forecast intervals typically use econometric models, system dynamics, and grey prediction, with a focus on policy adjustments, economic indicators (such as GDP and CPI), and population trends (Koukaras et al., 2024).

2.2 Weather in forecasting electricity demand

Temperature is a primary driver of electricity demand, shaping heating and cooling loads that dictate energy consumption. Research consistently identifies it as the dominant weather factor in electricity demand prediction, especially during peak periods. Liu et al. (2021) demonstrate that extreme temperatures lead to increased residential electricity consumption, finding that for each additional day in which the mean temperature exceeds 30 °C, there is an 16.8% increase in monthly residential electricity consumption. Similarly, for each additional day below -6 °C there is a 6% increase in monthly residential electricity consumption. This underscores temperature's critical role in accurate demand forecasting, as it directly influences consumption patterns.

Extreme temperatures can lead to significant errors in electricity demand forecasts, often underestimating demand. During Winter Storm Uri in Texas in February 2021 (Añel, 2024), minimum extreme cold temperatures of –34 °C and high winds of 260 km/h impacted 170 million people. Due to this extreme weather event, electricity demand unexpectedly increased from 40 GW to over 70 GW, resulting in blackouts that affected more than 4 million people. The economic cost of the power outages and disruption has been estimated between 26.1 and 130 billion U.S. dollars.

Other weather variables, particularly humidity and "feels like" temperature, enhance forecasting accuracy. Maia-Silva et al. (2020) found that using humidity-related measures, such as dew point and heat index, improves prediction accuracy, especially in high-energy-consuming regions, with improvements up to 8-9%. This highlights the need to consider composite weather indices, as air temperature alone underestimates demand.

2.3 Historical Forecasting Error Incorporation

Besides temperature and other weather components, historical measurements of energy demand or forecasted energy demand are highly reliable factors for predicting future energy demand (Singh and Yassine, 2018). Historical energy demand is important for capturing seasonal effects in different time horizons (day, week, month, season etc). However, historical energy forecasts (and by extension their differentials) are valuable because, in addition to seasonal effects, they capture bias and allow for corrections to the future forecast. Historical forecast factors are so influential that there is evidence that it can create reasonable forecasts without additional weather variables (Boroojeni et al., 2017).

2.4 Modelling electricity demand

As previously stated, short-term energy models can be effectively categorised into two groups: classical statistical techniques, and machine learning or AI techniques. Traditional statistical and econometric models tend to be explainable and interpretable. While often less accurate, these models are widely used in energy demand forecasting and include methods such as regression (Papalexopoulos and Hesterberg, 1990) (Ertuğrul, Tekin and Tekin, 2020) and time-series such as ARIMA (Tarmanini et al., 2023) (Ediger and Akar, 2007). They also have natural extrapolations to medium-to-long term models, that are also econometric-based due to their relationship with longitudinal factors such as policy changes, modifications to the energy grid, or economic factors (such as GDP and population) (Ardakani and Ardehali, 2014). While a machine learning model, decision tree methods also provide interpretability in energy demand forecasting (Kopyt et al., 2024) (Wang et al., 2018).

Black box machine learning models provide a greater focus on model accuracy rather than interpretability. Some common models used in energy demand forecasting include Neural Networks (Manno, Martelli and Amaldi, 2022) (Kuo and Huang, 2018) (Pao, 2009), Support Vector Machines (Ahmad et al., 2014) (Ahmad et al., 2020), and ensemble methods, such as Random Forests (Divina et al., 2019) and XGBoost (Abbasi et al., 2019).

Divina et al. (2019) studied short-term energy consumption forecasting in smart buildings using several models such as linear regression, auto-regressive integrated moving average (ARIMA), artificial Neural Networks (ANNs) and ensemble methods such as random forests (RF) and extreme gradient boosting (XGBoost). They measured the performance of these models using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). They found that the best performing models were machine based approaches, and more so ensemble methods such as RF, GBM and XGBoost. On the other hand, ARIMA was the worst performing method that was tested. Further, the optimal historical window was found to be 10 days where accuracy improves up to this point, but does not improve much beyond this. Tarmanini et al. (2023) considered ARIMA and Artificial neural network (ANN) models to forecast daily electricity load in Ireland. The study found that both ARIMA and ANN produced more error in winter than in other seasons. Despite this, the ANN method performed better in terms of accuracy due to it better coping with non-linear data, but suggest a hybrid approach may provide more accurate results.

Other studies have also concluded that the best performing models tend to be hybrid models which use a combination of explainable and/or black-box methods, such as NN–ARIMA or CNN-LTSM due to their stability and potential to reduce overfitting (Deng et al. 2022). For example, Suganthi and Samuel (2012) compared various approaches to energy demand forecasting and found often hybrid approaches such as linking ARIMA models with neural networks often produce more accurate results. The superiority of hybrid models for energy forecasts are due to corrections of the original forecast output in the second modelling component (Savić, Selakov and Milošević, 2014).

One important method used in hybrid modelling is residual error forecasting. Andronikos, Tzelepi and Tefas (2023) proposed a residual error learning methodology for electricity demand forecasting which involved training a model on actual load values, then calculating the residual errors which would subsequently be used as targets to train a second model. The final prediction of forecast load would then be the sum of the first model's prediction and the second model's prediction. The authors found that if the errors have an underlying structure, the residual error forecasting method will improve forecasting accuracy.

A common method used in many hybrid studies which also aligns closely with modelling residuals is to decompose time series data into trend and residual components and model these components separately with appropriate methods. The forecasts from each component are then summed together for the final forecast. Amara et al. (2019) used decomposition to extract the temperature-related component that makes up electricity demand and then analysed and forecasted the residual component. The two forecasts were then summed to produce the final forecast. This allowed for understanding of periodicity in the residuals and to improve the overall forecast accuracy. Zhang et al (2022) considered the Australian electricity market in their study using a decomposition-hybrid approach. They first extracted a trend component from the original electricity load, then obtained the nonlinear component by subtracting the trend component from the original electricity load. The two components were forecast separately and then added together to make up the final forecast. Their proposed model improved the forecasting accuracy against all comparison models. Another approach to hybrid modelling considered by Pao (2009) was a two step approach where a linear model was built and the results of this inputted into a neural network model to capture both linear and non-linear relationships in the data. This showed to produce superior predictions to a linear model alone.

The approach proposed in this report is based on a hybrid approach where the AEMO forecast will act as the initial model and a new model will be built considering the errors from that model in an attempt to improve the overall forecast accuracy.

2.5 AEMO forecasting methodology

The AEMO load Forecasting Methodology (AEMO, 2023) details the organisation's approach to forecasting electricity demand. With particular relevance to this report, AEMO pre-dispatch forecasts are short-term electricity demand forecasts that include intervals up to 40 hours. One of the important uses of pre-dispatch forecasting is to support operational planning that ensures electricity reliability and security of the network. The key inputs into the forecast include:

- Historical demand (such as recent load patterns)
- Weather forecast variables (particularly those that describe the temperature profile)
- Calendar variables (e.g. weekday or weekend, public or school holiday, daylight savings)
- Solar and wind generation forecasts.

There is very little manual intervention for these forecasts, with AEMO's Demand Forecasting System (DFS) generating forecasts automatically through a combination of statistical and machine learning models, every half hour.

The pre-dispatch load forecasting error threshold for NSW is 150 MW based on historical peak demand for NSW and previous forecasting performance. The load forecast is reviewed whenever the forecast error is greater than the threshold for two consecutive 30-minute periods, therefore at an overall level the forecast is already quite accurate.

Chapter 3

Material and Methods

Figure 3.1 shows the overall structure of this project designed to address the research question. The process began with data collection and pre-processing, including calculating the forecast error. Exploratory data analysis was conducted to understand relationships between different variables and the forecast error. The data were then split into training and testing samples for models to be built, fine-tuned and compared. The remaining sections of this report detail the steps undertaken in the modelling as well as analysis and presentation of the results.

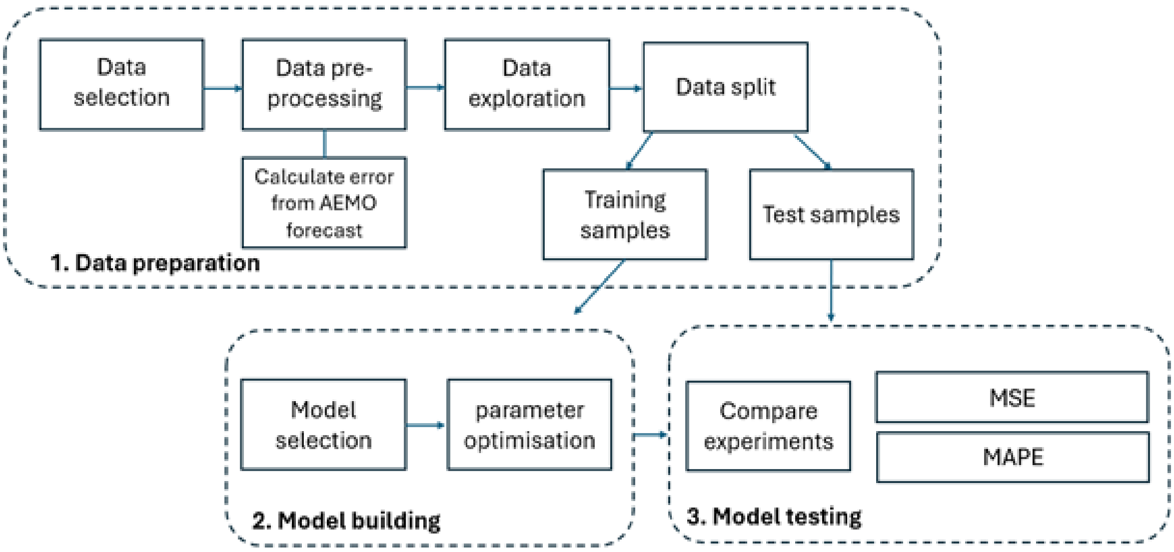

Figure 3.1: Project structure followed to address the research question

3.1 Software

Python was the primary software used for data analysis and modelling based on its flexibility in data visualisations and ability to execute machine learning models.

To ensure reproducibility, RMarkdown was used to prepare the final report. Power BI was also used in initial data exploration to understand high-level trends in the data. A Github repository was used to store data, code and working documents. The repository can be found here: `https://github.com/unswnick/project`. All relevant code for this project can be found in the Appendices. Appendix A contains data processing code, Appendix B contains Modelling code, and Appendix C contains code for plots.

A summary of software used as part of the project is summarised in Table 3.1.

Table 3.1: Summary of software used

Software	Library(s)	Purpose
Python	Pandas	Reading, manipulating, cleaning and analysing datasets.
	Numpy	Manipulating data and mathematical calculations.
	Matplotlib, Seaborn	Visualising data to understand trends and patterns.
	Scikit-learn	Implementing and evaluating machine learning algorithms.
PowerBI	-	Summarising and visualising data.
RMarkdown	-	Writing final report.
Github	-	Repository for project documents

3.2 Description of the Data

Table 3.2 describes the data that was used in analysis. In addition to the data files provided by the client, historical weather forecasts including temperature, humidity and wind speed were sourced from OpenWeather (OpenWeather, 2025), a global company specialising in environmental data products. Forecast weather data rather than actual weather data was used as an input to ensure the forecast models were realistic.

Table 3.2: Datasets used in this project and their properties

Data	Description
Electricity demand Use for both training and testing models.	Electricity demand from 2010 to 2021. Well-structured and low complexity with no duplicates and no null values. Variables: Date-time, totalDemand, regionID Format: CSV, Storage: Github, Size: 6 Mb, Rows: 196,513
Forecast demand Used as a baseline forecast model and improve on.	Provides forecasted demand data from 2010 to 2021. Well-structured with no null values. It is high complexity due to uneven time increments and duplicate rows. Variables: Date-time, forecastDemand, totalDemand, regionID, preDispatchSeqNo, periodID, lastChange Format: CSV, Storage: Github, Size: 722 Mb, Rows: 10,906,019
Forecast weather indicators Exogenous variables included in modelling.	Provides previous forecast weather data for Bankstown from October 7 2017. Well-structured with no null values. Variables: Date-time, temperature, humidity, wind speed, rain Format: CSV, Storage: GitHub, Sharepoint/Teams, Size: 1327.1 MB, Rows: 10854100

3.3 Data Cleaning

Data was found to be complete for Electricity Demand data Some forecast data were missing for forecast intervals >12 hours. To ensure complete data was used, and to reduce computational complexity, the forecast models were trained and tested on 12 hour forecast intervals only. There were no missing values in the Forecast Weather Indicators data, however the available data begins on October 7 2017. Consequently, the relevant data used to train and test the forecast model was between October 7 2017 and 17 March 2021 with a 12 hour forecast interval. No further missing values were present in the data.

Outliers were not removed from the data to ensure data completeness and to avoid introducing bias through exclusions. Further advice from industry experts would be required to determine which outliers, if any, should be removed based on appropriate criteria.

Additional data cleaning steps performed on all datasets are detailed below:

1. Date/time variables were formatted consistently (i.e. d/m/y H:M)
2. Date/time variables were rounded to the nearest 30 minute increment to provide consistent 30-minute intervals
3. Duplicate date/time rows were removed to ensure each date/time row was unique

After each dataset was cleaned and checked, they were merged into one clean dataset, joined on the unique date/time variable.

3.4 Data pre-processing

Outlined below are the steps undertaken to pre-process the data:

1. **Feature extraction** – The following features were extracted from date/time variables:
   - Hour_of_day
   - Month_of_year
   - Day_of_week
2. **Label enconding** – Hour_of_day, Day_of_week and Month_of_year variables were one-hot-encoded into binary variables
3. **Feature engineering** – The following new features were created:
   - Forecast interval (date/time future – date/time current)
   - Forecast error (total demand – forecast demand)
   - 24-hour Forecast Error (Forecast error from 24 hours ago)
   - 48-hour Forecast Error (Forecast error from 48 hours ago)
   - 72-hour Forecast Error (Forecast error from 72 hours ago)
   - 7-day Forecast Error (Forecast error from 7 days ago)
   - 14-day Forecast Error (Forecast error from 14 days ago)
   - Relative error (Forecast error / total demand)
   - Hour × Temperature (Hour * Temperature)
   - Hour (Sine) (hour_sin) — sin(2 × Hour / 24)
   - Hour (Cosine) (hour_cos) — cos(2 × Hour / 24)
   - Month × Temperature (MonthNumb * Temperature)
   - Hour × Forecast Demand (Hour * forecast_demand)
   - Temperature × Forecast Demand (Temperature * forecast_demand)
   - Temperature × Hour (Sine) (Temperature * hour_sin)

    - Temperature × Hour (Cosine) (Temperature * hour_cos)
    - Forecast Demand × Hour (Sine) (forecast_demand * hour_sin)
    - Forecast Demand × Hour (Cosine) (forecast_demand * hour_cos)
    - 24-hour Forecast Error × Hour (Cosine) (24hrpreverrors * hour_cos)
    - 24-hour Forecast Error × Hour (Sine) (24hrpreverrors * hour_sin)
4. **Splitting the data** - As a final step in pre-processing, the data were split into 70% training 7 October 2017 – 5 March 2020) and 30% testing (6 March 2020 – 17 March 2021). This split allowed for a large number of data to be trained on, and a full year to test which captured all seasonal effects. The same split was used across the models.

Note that modelling methods chosen did not require normalisation of the data.

3.5 Assumptions

- AEMO's forecast data is released every 5 minutes, therefore forecast data for the 12 hour interval is available to use in the model
- Temperature/weather forecasts are available for 12 hours into the future.
- Bankstown weather variables are reasonable representations of weather conditions across New South Wales.

3.6 Modelling Methods

The following methods were in this study:

- Linear Regression: Baseline model for improving forecasts due to its simple implementation and interpretability.
- SARIMA: EDA identified autocorrelation between forecast errors. Due to the seasonal nature of electricity demand, SARIMA modelling was conducted.
- Decision Trees: EDA identified non-linearity between electricity demand and its explanatory variables. As such, decisions trees were implemented to explore simpler non-linear behaviors.
- XGBoost: Implemented to explore non-linear behaviors using advanced techniques.

These modelling methodologies are described below.

ARIMA

Auto Regressive Integrated Moving Average (ARIMA) is a time series forecasting model. Besides being well-researched and more readily explainable compared to machine learning models, its algorithm specifications make it suitable for energy demand forecasting. The model consists of three main components:

Auto Regression: The model utilises lagged observations or previous time points. Due to the weather conditions of previous days having a direct influence on future weather, previous time points are relevant for forecasting. In addition, energy demand also exhibits seasonality that can be captured by previous inputs.

Differencing (Integration): Energy and weather demands over different time horizons exhibit slight trend. Raw observations are differenced to make statistical properties (such as mean or variance) stabilised over time.

Moving average: Smooths variance by modelling a moving average of lagged variables against point residuals. This reduces noise in highly variable factors susceptible to measurement error like weather.

Seasonal Auto Regressive Intergrated Moving Average (SARIMA) is an extension of the ARIMA model. SARIMA is designed to support seasonality in time series data. It can be modified to incorporate seasonality in different time horizons such as weekly, monthly, or quarterly time frames. The model parameters are the same as ARIMA with the inclusion of seasonal variants to control for seasonal effects: seasonal autoregressive order, seasonal differencing order, and seasonal moving average order.

Decision Trees

Decision Trees are a type of explainable machine learning model. They are trained by recursively dividing the dataset into subsets using entropy (a measure of impurity or randomness in the dataset) and optimise for information gain. The impurity is in context to the target variable. When a subset of data is comprised of an entire class, it is considered pure. It is interpretable because the model construction can be read as a series of conditional IF statements to achieve certain outputs.

A Random Forest is a collection of generated Decision Trees. The generation formula is consistent across each decision tree, the difference being each tree is generated from a different bootstrap sample. The prediction outputs for regression tasks, such as energy demand forecasting, is an average of all decision tree outputs. Random Forests lose the ability of decision trees to be interpretable, the benefit however, is improved accuracy and robustness.

XGBoost

XGBoost, short for extreme gradient boosting, is a gradient descent machine learning method. Its formulation is by use of a loss function to measure the difference between predicted and actual values and a regularization term to penalize complex models.

It functions by building decision trees sequentially. Each tree is trained to predict the residuals from previous trees. Each tree split mechanism follows the process of regular decision tree training. Each tree's contribution to the final prediction is weighted by a learning rate. It generally outperforms regular decision tree models due to its internal corrections of error and feature selection. Its construction makes it suitable for regression tasks such as energy demand forecasting.

Chapter 4

Exploratory Data Analysis

This section presents an exploratory analysis of the temperature, forecasted demand, and actual electricity demand data. Exploratory data analysis (EDA) explored how demand responds to temperature variations and where forecast discrepancies are most pronounced. The data is manipulated and visualised with Python.

We begin the analysis by focusing on the individual distributions and characteristics of each dataset. This stage provides context on the seasonal variability of the data.

4.1 Electricity Demand

Electricity demand shows a cyclical pattern with a downward trend when observing it throughout the years (Figure 4.1). This may be due to more households investing in embedded generation to supplement their electricity supply.

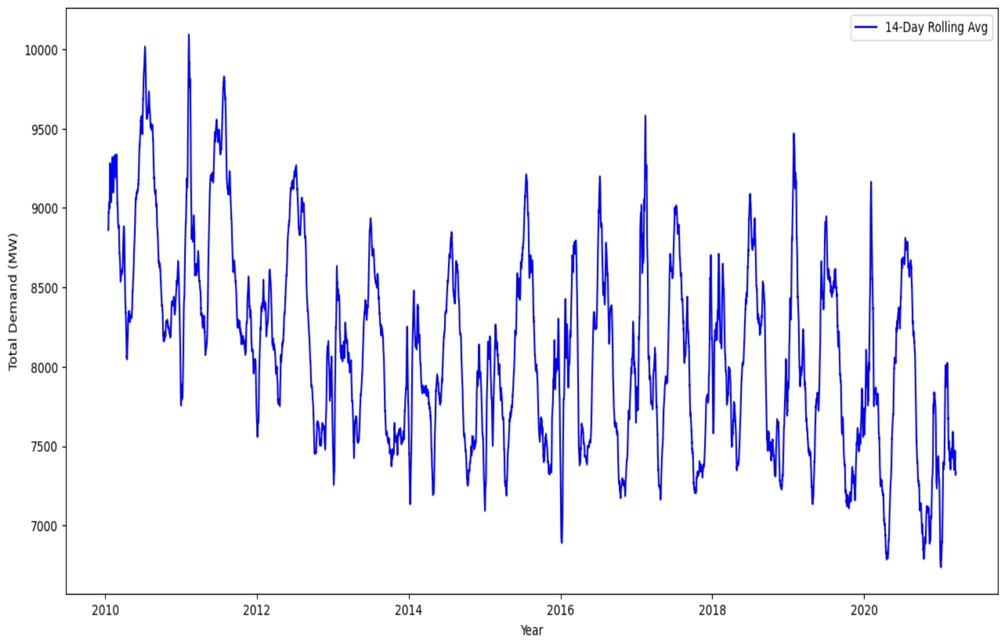

Figure 4.1: Electricity Demand vs Time

Electricity demand is higher during winter and summer months (Figure 4.2). This is likely due to higher consumption of electricity to power heating and cooling appliances.

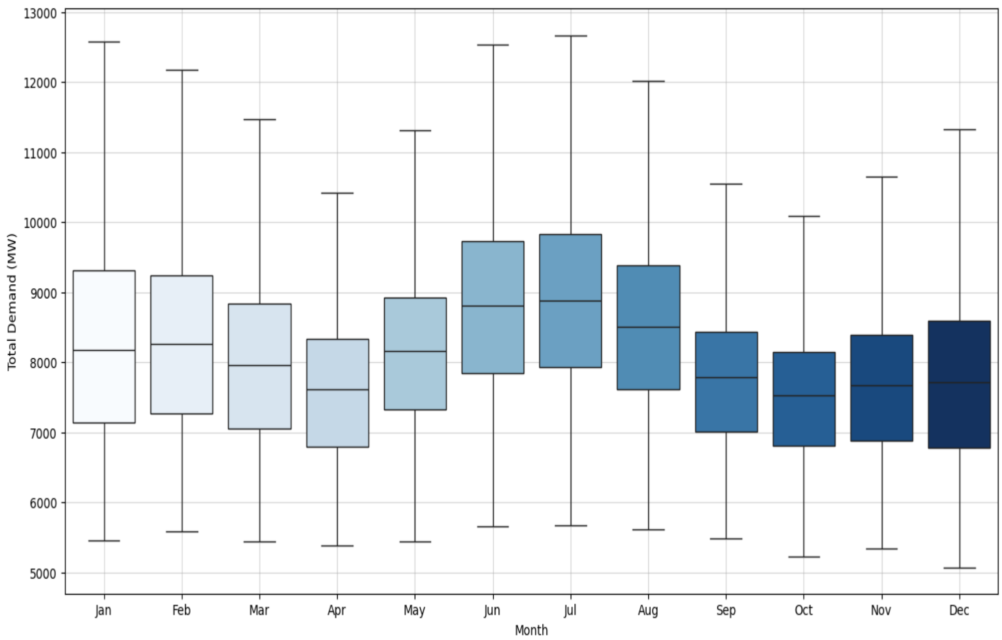

Figure 4.2: Total Demand by Month

Demand was observed to be greater in weekdays than weekends (Figure 4.3). This may be due to many businesses closing during weekends.

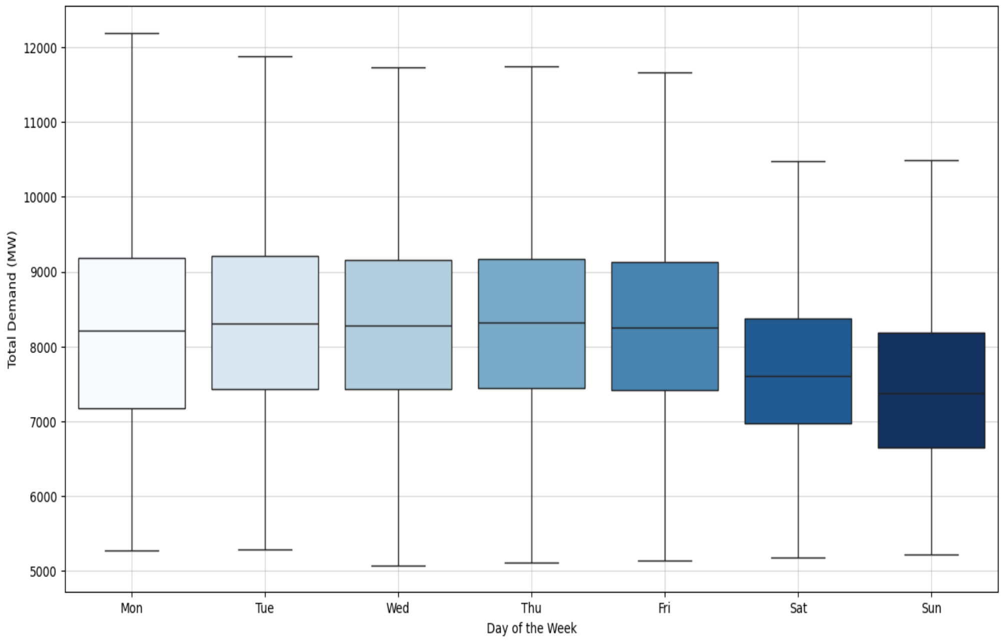

Figure 4.3: Total Demand by Day of the Week

Electricity demand is relatively high between 8am and 11pm (Figure 4.4), likely due to the human sleeping cycle.

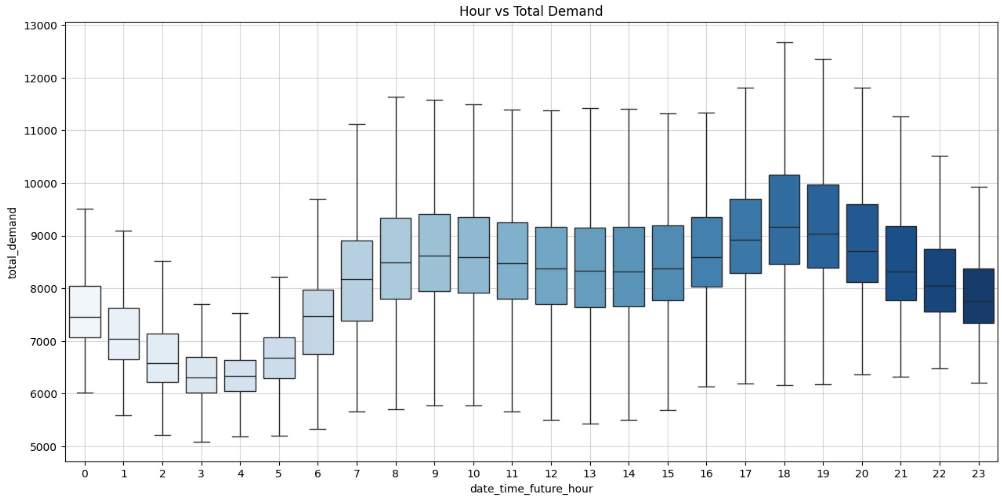

Figure 4.4: Total Demand by Hour of Day

4.2 Forecast Electricity Demand

This dataset contains electricity demand forecasts made every 30 minutes. Each time a forecast is made, it includes 48 predictions—one for each half-hour period from 30 minutes ahead up to 24 hours ahead.

The scatter plot of electricity demand forecasts vs actual electricity demand across different prediction time periods (Figure 4.5), reveals lower correlation as both the prediction time period and actual electricity demand increase.

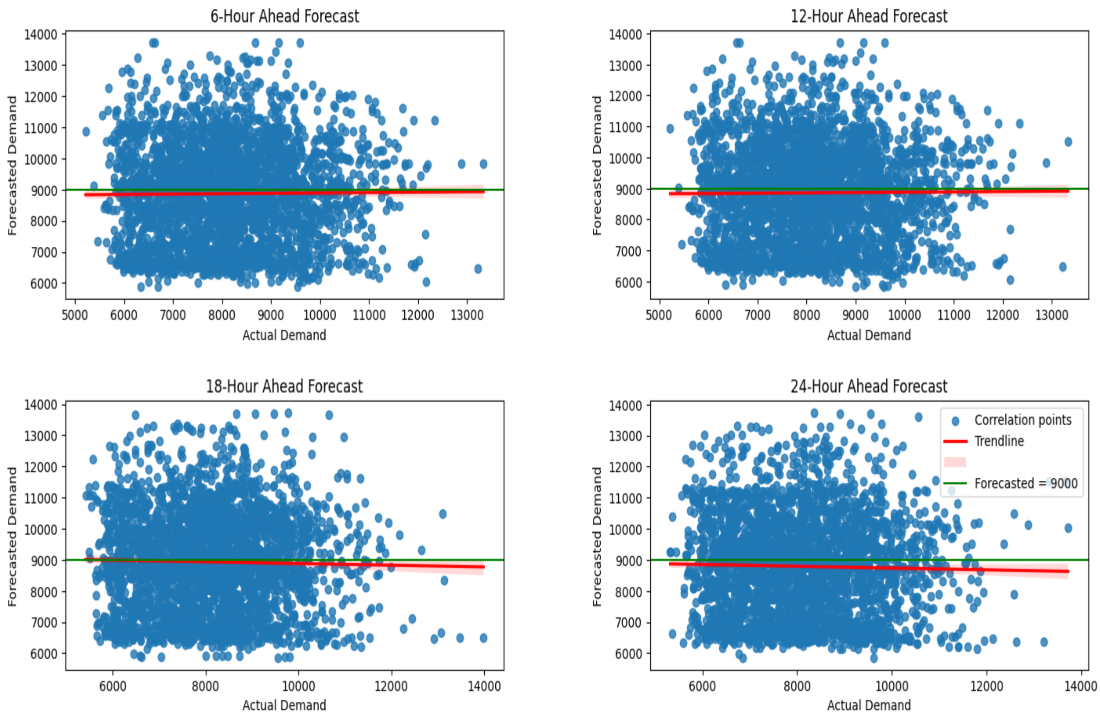

Figure 4.5: Scatter plots of Forecast Demand vs Total Demand by Lag interval

4.3 Weather Variables vs Electricity Demand

In this next section, we will examine if and how weather affects both the electricity demand and its forecast.

The correlation of relevant weather variables with electricity demand shows weak correlation across all variables, with humidity having the highest correlation and rain having the lowest (Figure 4.6).

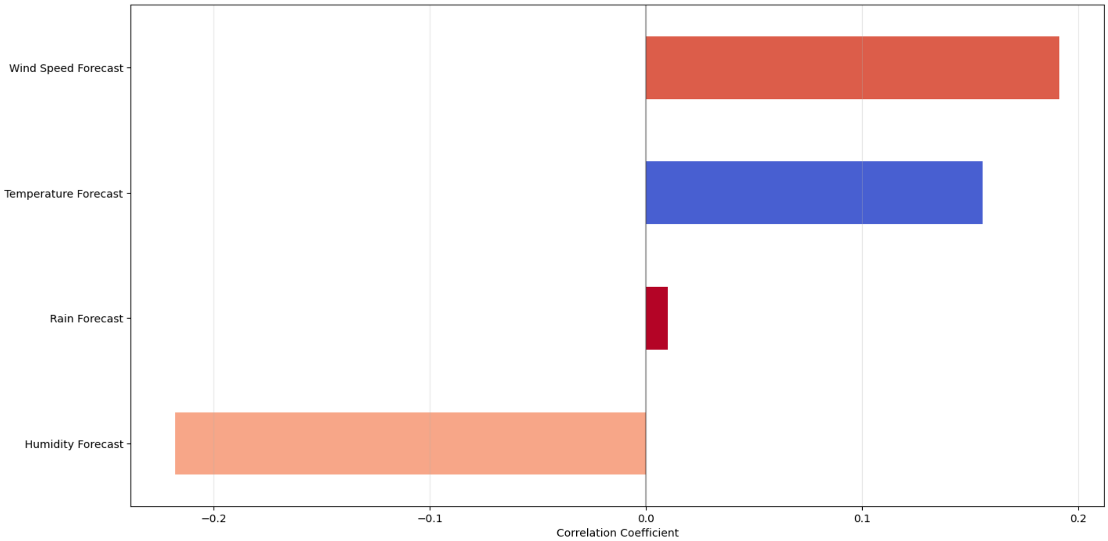

Figure 4.6: Correlation of Weather variables with Electricity Demand

The plot of temperature against electricity demand reveals a distinct U-shaped correlation. This pattern reflects energy usage behaviour in response to extreme temperatures (Figure 4.7). The lowest demand levels generally occur in temperate conditions where neither heating nor cooling is heavily used.

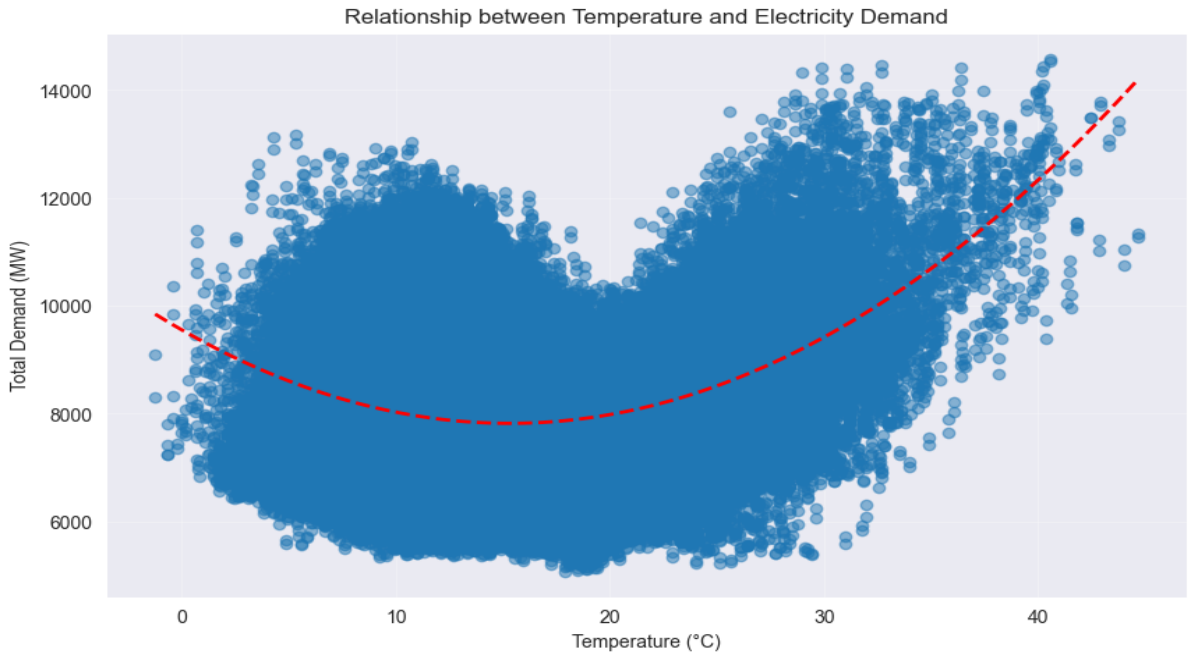

Figure 4.7: Scatter plot of Electricity demand vs Temperature

4.4 Forecast Error of Electricity Demand

In this section, we will explore whether forecast inaccuracies are correlated with known variables which contribute to electricity demand. Forecast error was defined as actual demand less forecast demand.

Figure 4.8 shows that forecast error increases with temperature for temperatures greater than ~29°C. This suggests the current forecasting model may lack information regarding forecasted temperatures. The trend also occurs for normalised demand (Figure 4.9).

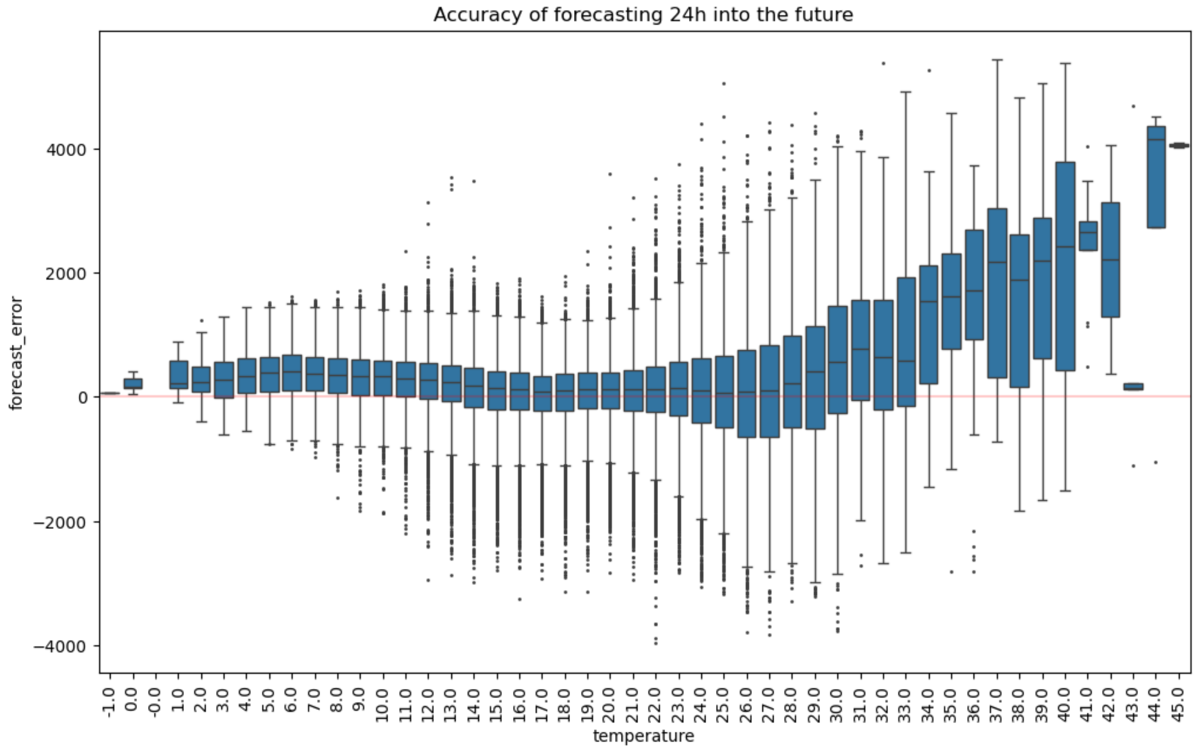

Figure 4.8: Forecast Error vs Temperature Forecast

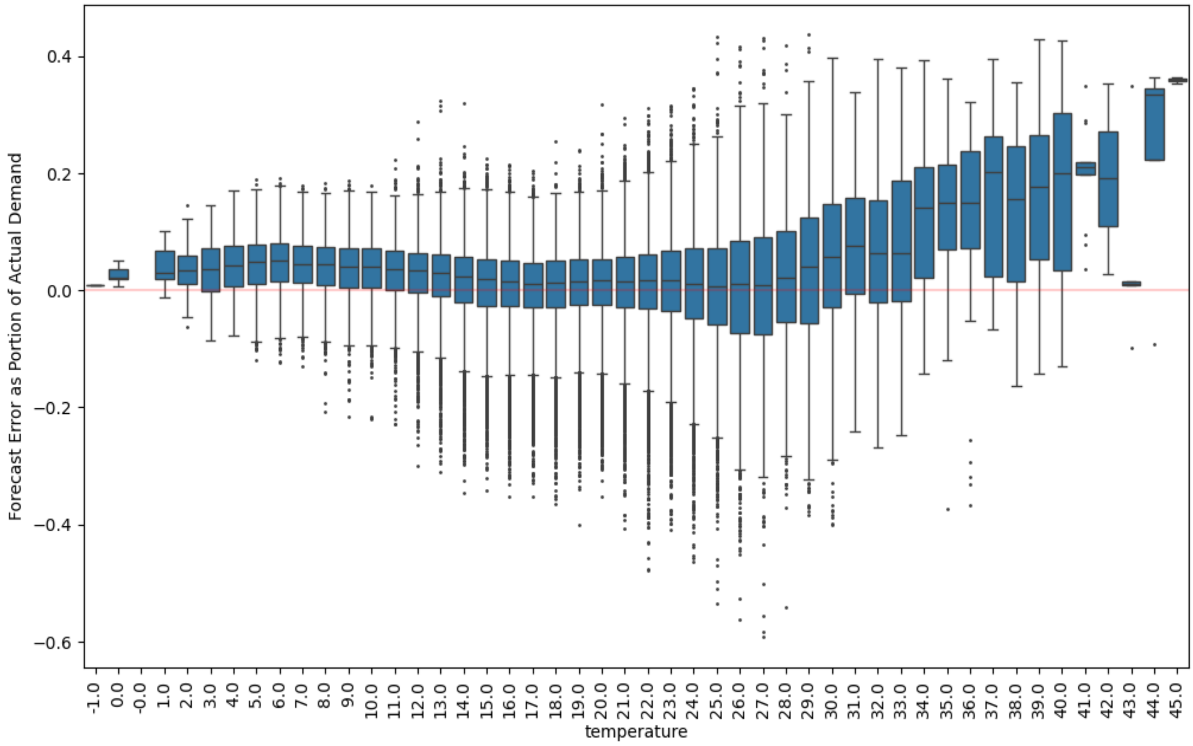

Figure 4.9: Normalised forecast error vs temperature forecast

4.4.1 Time Series Analysis

Time series analysis of the forecast error was conducted to understand whether errors persisted with time. It was conducted for 6, 12, 18 and 24-hour forecasts.

Augmented Dickey-Fuller test (ADF Test) was conducted. It showed significant evidence for stationary forecast errors (Table 4.1).

Table 4.1: ADF tests conducted for 6, 12, 18 and 24-hour forecasts

Hour of day = 6	Hour of Day = 18
ADF Statistic: -33.863838 p-value: 0.000000 Critical Values: 1%: -3.430 5%: -2.862 10%: -2.567 Stationary	ADF Statistic: -31.926828 p-value: 0.000000 Critical Values: 1%: -3.430 5%: -2.862 10%: -2.567 Stationary
Hour of Day = 12	**Hour of Day = 24**
ADF Statistic: -33.407029 p-value: 0.000000 Critical Values: 1%: -3.430 5%: -2.862 10%: -2.567 Stationary	ADF Statistic: -29.030458 p-value: 0.000000 Critical Values: 1%: -3.430 5%: -2.862 10%: -2.567 Stationary

Autocorrelation function (ACF) and partial autocorrelation function (PACF) plots were generated to understand the relationship between forecast errors and lagged versions of itself over successive time lags (Figure 4.10, Figure 4.11). PACF plots showed that forecast errors were significantly partially correlated with the most recent forecasts and ones made 24 and 48 hours prior. The partial correlation of the 24-hour lagged forecast error was of note due its greater significance than the 48-hour lag and its availability when forecasting.

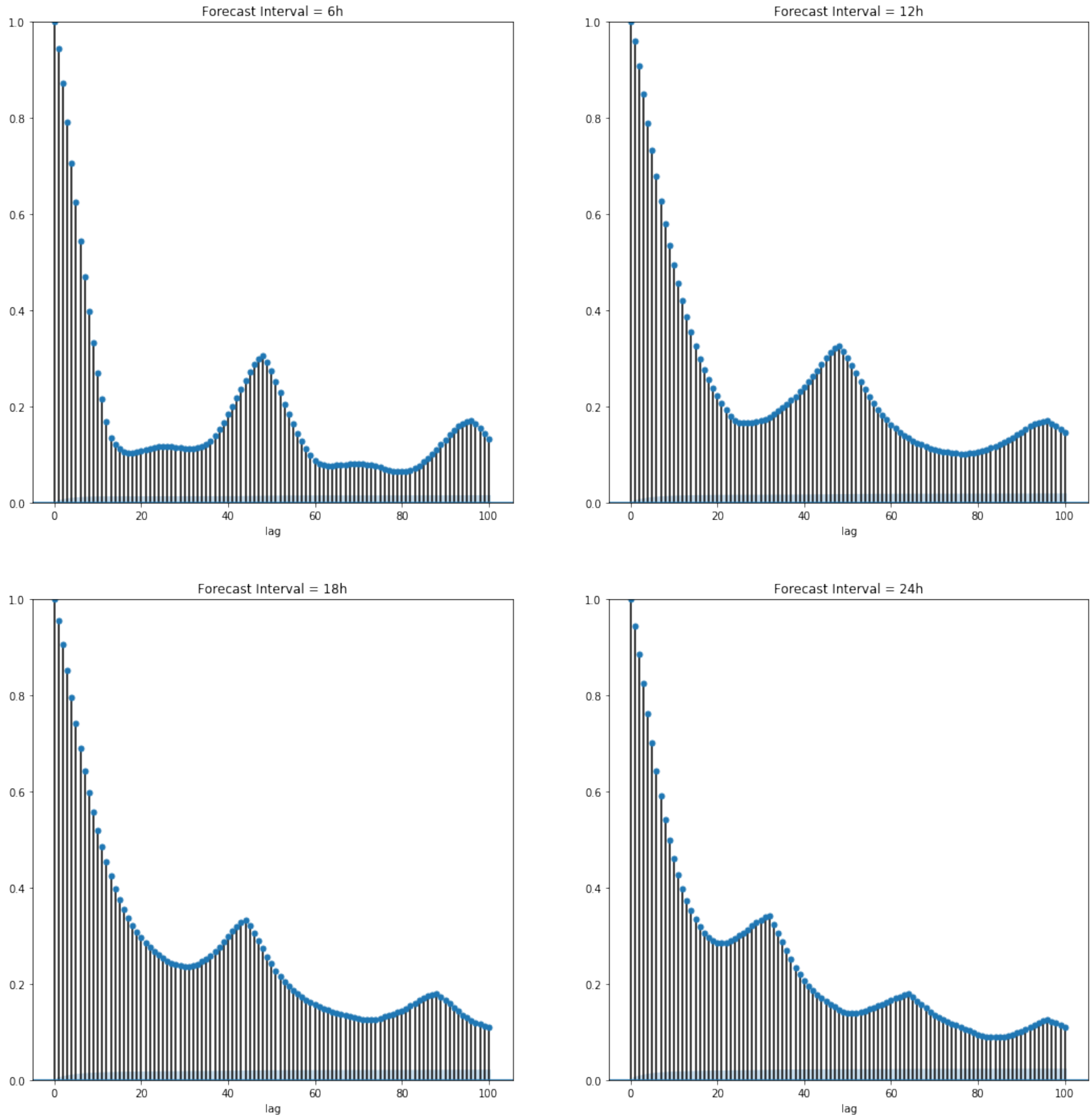

Figure 4.10: ACF of forecast errors

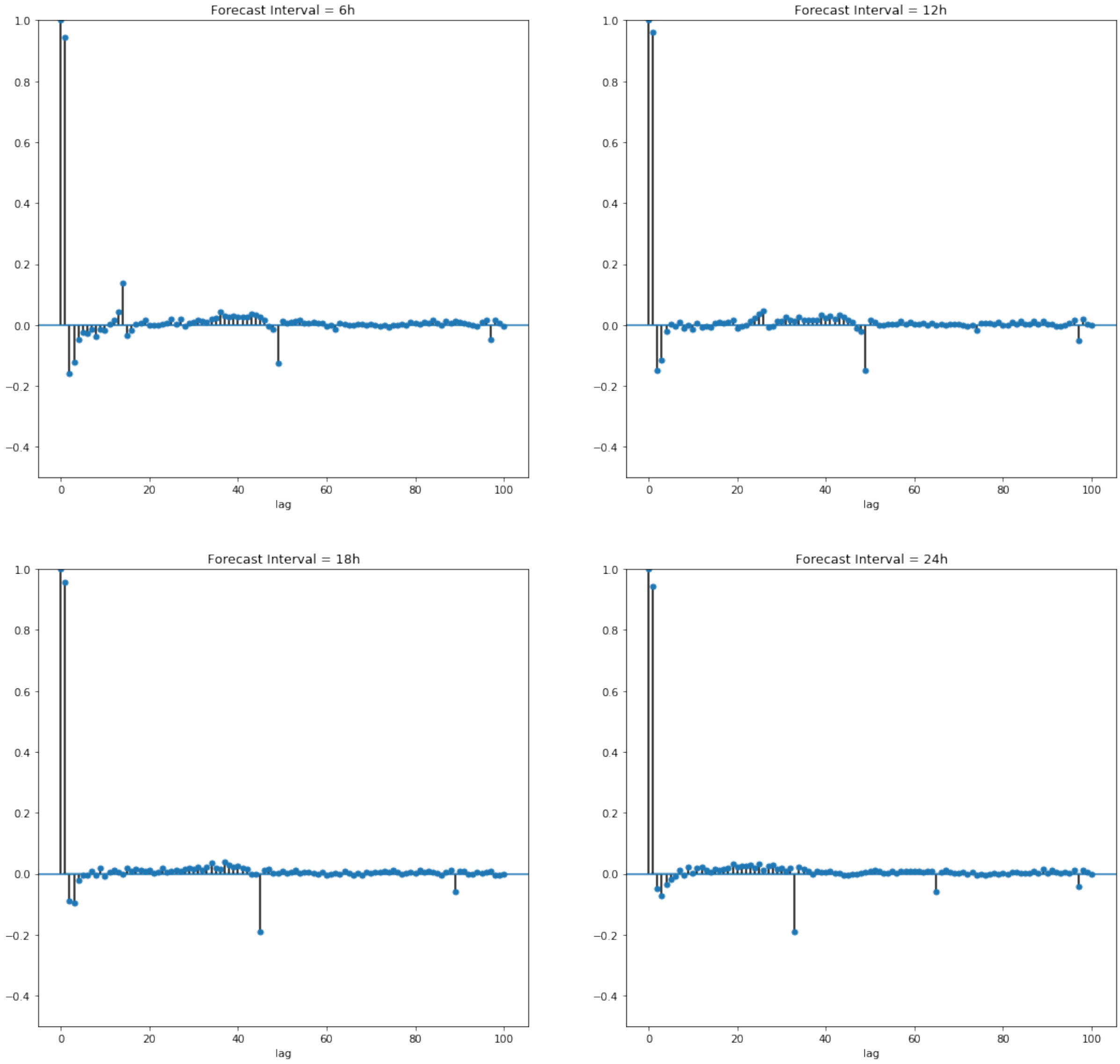

Figure 4.11: PACF of forecast errors

Scatterplots and correlations for forecast errors and its 24-hour lagged error can be seen in Figure 4.12.

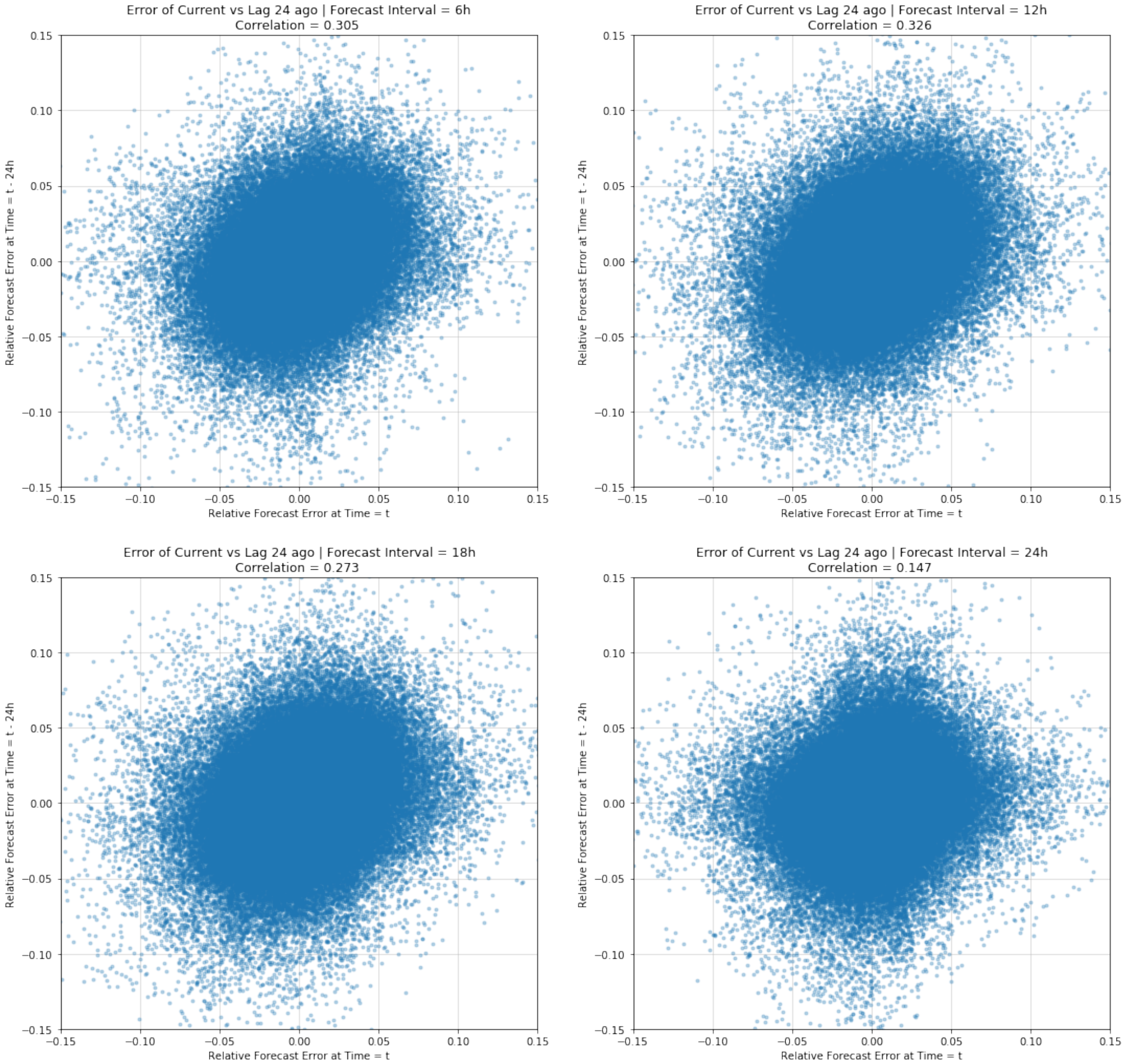

Figure 4.12: Scatterplots and correlations for forecast errors and its 24-hour lagged errors

4.5 Summary of Key Findings

- **U-shaped relationship between temperature and electricity demand:** Electricity demand increases during both extreme cold and extreme heat conditions, with the lowest demand observed during temperate conditions. This pattern is consistent with expected heating and cooling behavior and is evident in both actual and forecasted demand data.
- **Forecasting models capture seasonal trends:** Forecasted electricity demand shows a similar U-shaped relationship with temperature, indicating that the models are aligned with seasonal usage patterns.
- **Forecast error increases non-linearly with temperature, especially during extreme heat:** Forecast accuracy deteriorates significantly at higher temperatures, suggesting that current models underperform during periods of extreme heat. In comparison, performance during extreme cold is better, though still less accurate than under mild conditions.

- **Forecast errors are autocorrelated with past errors:** Forecast errors may be modelled by understanding historical forecast errors. Of note, the 24-hour lagged forecast error may be used for improving forecasts.
- **The forecast has the potential to be improved:** These findings highlight some correlations between forecast errors, temperature and time variables such as season which may indicate the model could be improved by modelling forecast errors.

Chapter 5

Analysis and Results

5.1 Performance measures

Two common performance measures were chosen to calculate prediction accuracy and compare models. Mean square error (MSE) and mean absolute percentage error (MAPE) were chosen to be the most appropriate measures. MSE penalises large errors which is useful to assess when the aim is to reduce large errors. Furthermore, MAPE provides an easily interpretable and comparable result. The measures are described below:

MSE is the average of the squared difference between actual demand and forecasted demand.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MAPE takes the absolute value of the difference between the actual demand and forecast demand expresses it as a percentage of actual demand, and takes the average of this.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - F_i|}{A_i} * 100$$

For both measures, a smaller value represents higher accuracy, and a better performing model.

MSE and MAPE values for the original forecast can be seen below.

$$\text{Existing model MSE} = 55159$$
$$\text{Existing model MAPE} = 2.19\%$$

5.2 Linear Regression

Forecast error was defined as actual demand less forecast demand (Equation (5.2.1)).

$$\varepsilon_t = y_t - \hat{y}_t \tag{5.2.1}$$

Two linear regression models were trained for predicting the forecast error (Equation (5.2.2)). The predicted forecast error was then used to update the forecast (Equation (5.2.3)). The aim of the models was to reduce the updated forecast error (i.e. $\varepsilon_t^* < \varepsilon$).

$$\varepsilon_t = \hat{\varepsilon}_t(...) + \varepsilon_t^* \tag{5.2.2}$$

$$\hat{y}_t^* = \hat{y}_t + \hat{\varepsilon}_t(...) + \varepsilon_t^* \tag{5.2.3}$$

5.2.1 Model Construction

Linear Regression - Model 1

The first model used only the 24-hour lag forecast error for predicting the forecast error (Equation (5.2.4)). Hence, it was a simple autoregressive model.

$$\varepsilon_t^{(\mathrm{LR1})} = \theta_0 + \theta_1 \varepsilon_{t-24h} \tag{5.2.4}$$

The model summary can be seen in Table 5.1. It showed that all variables are significant at the 0.05 significance level.

Table 5.1: Linear Regression Model 1, OLS Regression Results

OLS Regression Results			
Dep. Variable:	y	R-squared:	0.113
Model:	OLS	Adj. R-squared:	0.113
Method:	Least Squares	F-statistic:	2696.
Date:	Sun, 20 Apr 2025	Prob (F-statistic):	0.00
Time:	11:19:06	Log-Likelihood:	-1.4233e+05
No. Observations:	21064	AIC:	2.847e+05
Df Residuals:	21062	BIC:	2.847e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.5613	1.438	7.346	0.000	7.743	13.379
forecast_error	0.3369	0.006	51.927	0.000	0.324	0.350

Omnibus:	2737.590	Durbin-Watson:	0.201
Prob(Omnibus):	0.000	Jarque-Bera (JB):	23155.543
Skew:	-0.344	Prob(JB):	0.00
Kurtosis:	8.090	Cond. No.	222.

Linear Regression - Model 2

The second model used the 24-hour lag forecast error and all possible explanatory variables for the forecast error identified in EDA (Equation (5.2.5)).

$$\begin{aligned}\varepsilon_t^{(LR2)} = &\theta_0 + \theta_1 \varepsilon_{t-24h} + \theta_2 forecastTemperature_t + \\ &\theta_3 forecastHumidity_t + \theta_4 forecastWind_t + \theta_5 forecastRain_t + \\ &\theta_6 isSaturday_t + \theta_7 isSunday_t + \theta_8 isJanuary_t + \\ &\theta_9 isNovember_t + \theta_{10} isDecember_t\end{aligned} \tag{5.2.5}$$

The model summary can be seen in Table 5.2. It showed that all variables, except *forecastRain*, are significant at the 0.05 significance level.

Table 5.2: Linear Regression Model 2, OLS Regression Results

OLS Regression Results			
Dep. Variable:	y	R-squared:	0.124
Model:	OLS	Adj. R-squared:	0.124
Method:	Least Squares	F-statistic:	298.6
Date:	Sun, 20 Apr 2025	Prob (F-statistic):	0.00
Time:	11:19:08	Log-Likelihood:	-1.4220e+05
No. Observations:	21064	AIC:	2.844e+05
Df Residuals:	21053	BIC:	2.845e+05
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.5613	1.438	7.346	0.000	7.743	13.379
forecast_error	0.3369	0.006	51.927	0.000	0.324	0.350
Temperature	-1.1963	0.296	-4.040	0.000	-1.777	-0.616
Humidity	0.9220	0.094	9.811	0.000	0.738	1.106
Wind_speed	6.7613	0.927	7.296	0.000	4.945	8.578
Rain	-3.8983	2.807	-1.389	0.165	-9.399	1.603
isSaturday	29.1366	4.143	7.033	0.000	21.016	37.257
isSunday	8.2777	4.149	1.995	0.046	0.145	16.410
isDecember	20.6354	4.986	4.139	0.000	10.863	30.408
isJanuary	13.2394	5.233	2.530	0.011	2.982	23.497
isNovember	10.3352	4.832	2.139	0.032	0.865	19.805

Omnibus:	2701.773	Durbin-Watson:	0.203
Prob(Omnibus):	0.000	Jarque-Bera (JB):	23691.264
Skew:	-0.315	Prob(JB):	0.00
Kurtosis:	8.157	Cond. No.	1.67e+03

5.2.2 Model Performance

The predicted forecast error was then used to update the forecast error (Equation (5.2.3). Model evaluation (MSE and MAPE) can be seen below.

LRegression Model 1 Performance
- MSE: 49348
- MAPE: 2.06%

LRegression Model 2 Performance
- MSE: 49718
- MAPE: 2.078%

Model 1 performed better as it minimised both MAPE and MSE values.

5.3 S-ARIMA

Two SARIMA model collections were trained for predicting the forecast error (Equation (5.2.2). The predicted forecast error was then used to update the forecast (Equation (5.2.3)). The aim of the models was to reduce the updated forecast error (i.e. $\varepsilon_t^* < \varepsilon_t$).

A model collection contained a SARIMA model for each hour of the day. This reduced overall computation time, while allowing hour of day to be an explanatory variable (note, training the model on all data was not feasible due to limited computing power). The large data size should allow for data segmentation to have minimal impact on model training.

EDA, conducted earlier, showed that forecast errors are partially correlated with lagged values of itself in 24-hour intervals. As such, SARIMA modelling only considered lags of 24-hours.

5.3.1 Parameter Selection

ADF tests conducted showed significant evidence for stationary forecast errors, after segmentation by hour of day (Table 5.3). As such no differencing (d, D) was considered for SARIMA modelling.

Table 5.3: ADF tests for 4, 10, 16, 22-hour forecasts

Hour of day = 4	Hour of Day = 10
ADF Statistic: -5.875132 p-value: 0.000000 Critical Values: 1%: -3.432 5%: -2.862 10%: -2.567 Stationary	ADF Statistic: -7.680207 p-value: 0.000000 Critical Values: 1%: -3.432 5%: -2.862 10%: -2.567 Stationary
Hour of Day = 16	**Hour of Day = 22**
ADF Statistic: -9.601104 p-value: 0.000000 Critical Values: 1%: -3.432 5%: -2.862 10%: -2.567 Stationary	ADF Statistic: -7.968233 p-value: 0.000000 Critical Values: 1%: -3.432 5%: -2.862 10%: -2.567 Stationary

ACF and PACF plots were generated to assist in SARIMA parameters selection (Figure 5.1, Figure 5.2). The PACF plot showed that, generally, forecast errors are partially correlated with the first lagged term, followed by the next six lagged term, then 1-week and 2-week lags. As such, auto-regressed parameters (p) considered were 1, 2, 6 and 7, and the auto-regressed seasonal parameters (P) considered were 1 and 2. The seasonality parameter (s) was set at 7 for a weekly seasonality.

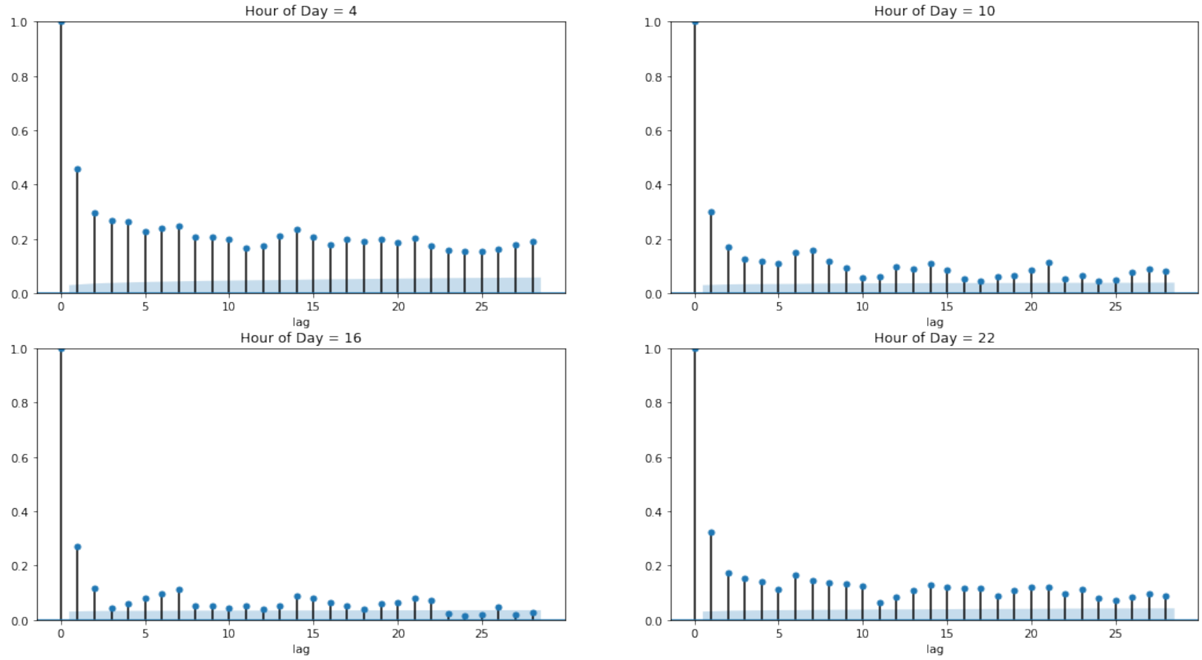

Figure 5.1: ACF of forecast errors with 24-hour lags

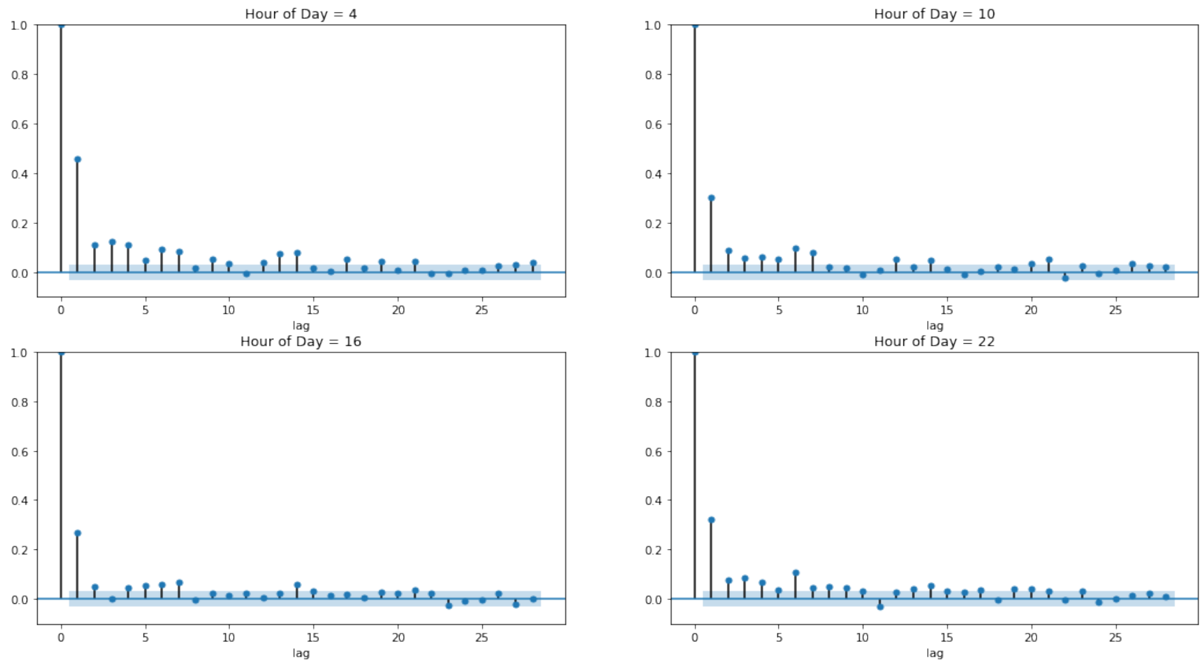

Figure 5.2: PACF of forecast errors with 24-hour lags

Moving average parameters considered (q, Q) were 0, 1 and 2. A summary of model parameters can be seen in Table 5.4.

Table 5.4: SARIMA Model parameters

ID	ARIMA Order			Seasonal Order			
	p	d	q	p	D	Q	s
0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
2	7	0	1	0	0	0	0
3	7	0	7	0	0	0	0
4	1	0	0	1	0	1	7
5	6	0	2	1	0	1	7
6	6	0	2	1	0	2	7
7	8	0	1	2	0	1	7
8	2	0	1	2	0	1	7

MSE and MAPE values were generated for each model in Table 5.4 (Figure 5.3, Figure 5.4). Models 5, 6 and 7 equally improved MSE and minimised MAPE values. Model 5 was selected as it was the least complex of the three.

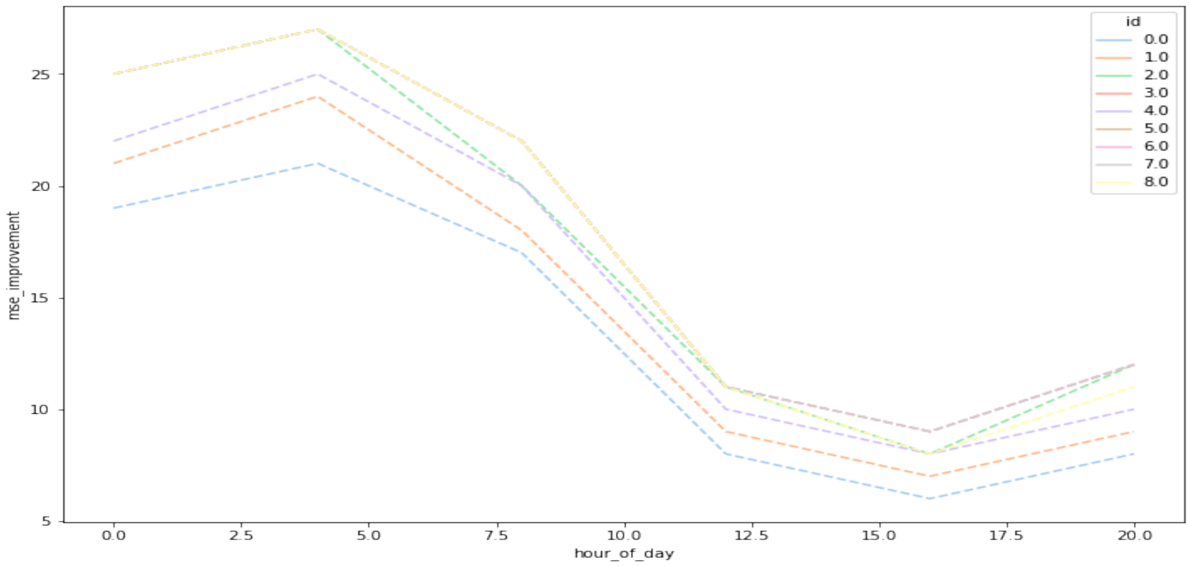

Figure 5.3: MSE improvement for each SARIMA model

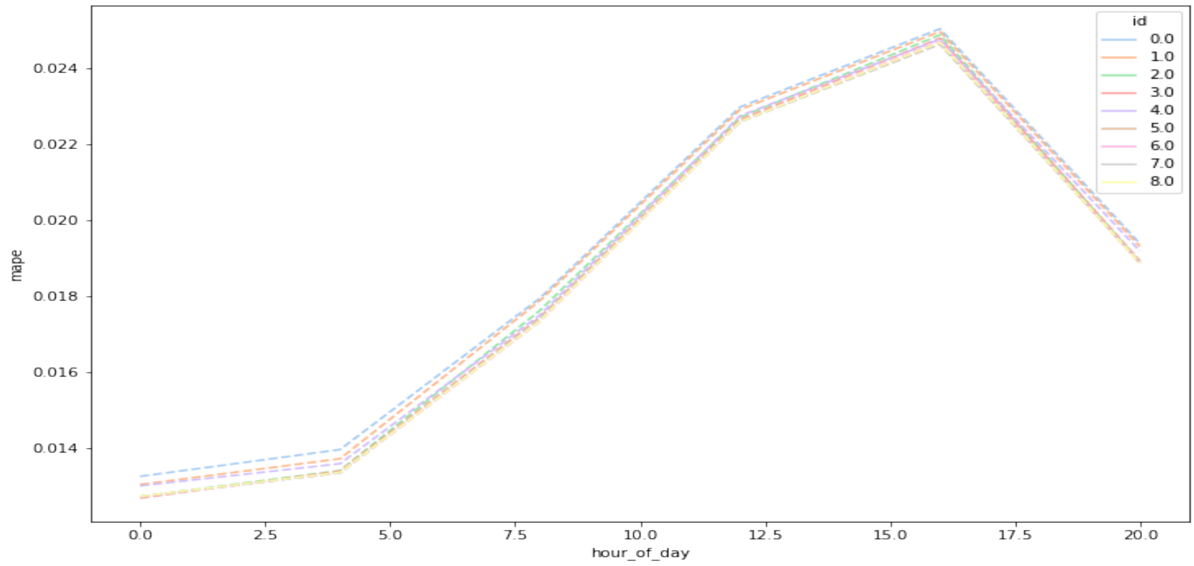

Figure 5.4: MAPE for each SARIMA model

5.3.2 Model Construction

SARIMA - Model 1

The first model used only lagged versions of the forecast error for predicting the forecast error (Equation (5.3.1)).

$$\varepsilon_t^{(\mathrm{SARIMA1})} = \mathrm{SARIMA}(6,0,2)(1,0,1,7) \tag{5.3.1}$$

SARIMA - Model 2

The second model used lagged versions of the forecast error and exogenous data (forecast temperature, humidity, wind and rainfall) for predicting the forecast error (Equation (5.3.2)).

$$\begin{aligned} \varepsilon_t^{(SARIMA2)} =& (SARIMA)(6,0,2)(1,0,1,7)+ \\ & \theta_1 forecastTemperature_t+ \\ & \theta_2 forecastHumidity_t + \theta_3 forecastWind_t+ \\ & \theta_4 forecastRain_t \end{aligned} \tag{5.3.2}$$

5.3.3 Model Performance

The predicted forecast error was then used to update the forecast error (Equation (5.2.3)). Model evaluation (MSE and MAPE) can be seen below.

Old Model Performance	**SARIMA Model (no Exog) Performance**	**SARIMA Model (with Exog) Performance**
- MSE: 55078.33198	- MSE: 49519.31974	- MSE: 49165.32932
- MAPE: 2.178%	- MAPE: 2.049%	- MAPE: 2.082%

Model 1 performed better as it minimised MAPE, which was given greater importance.

5.4 Random Forest

Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the average prediction of the individual trees.

The Random Forest model discussed aims to reduce the demand forecast error by predicting demand directly rather than predicting the error and then updating the original forecast.

5.4.1 Model Construction

RFMF1 - Model 1

The first model used the lag of the forecast error. The accuracy of predictions improved slightly in this model.

RFMF1 - Model 2

The second model used the lag of the forecast error and included weather variables (forecast temperature, wind speed, humidity and rain). The model predictions improved in this model (Figure 5.6).

5.4.2 Parameter Selection (Fine Tuning)

Fine tuning was done by utilising Grid Search on the Random Forest Model.

By trialing many parameters combinations, the following combination was found to be the best performing.

$$\begin{aligned} n\_estimators &= 200 \\ max\_depth &= 10 \\ min\_samples\_split &= 5 \\ min\_samples\_leaf &= 2 \end{aligned}$$

Where $n\_estimator$ is the number of trees, max_depth is the maximum depth of each individual tree, $min\_samples\_split$ is the minimum number of samples required to split an internal node and, $min\_samples\_leaf$ is the minimum number of samples required to be at a leaf node.

5.4.3 Model Performance

Setting up models with the values above had the following results (Figure 5.5, Figure 5.6):

RFMF1 Performance
- MSE: 51971.086
- MAPE: 2.111

RFMF2 Performance
- MSE: 51395.334
- MAPE: 2.095

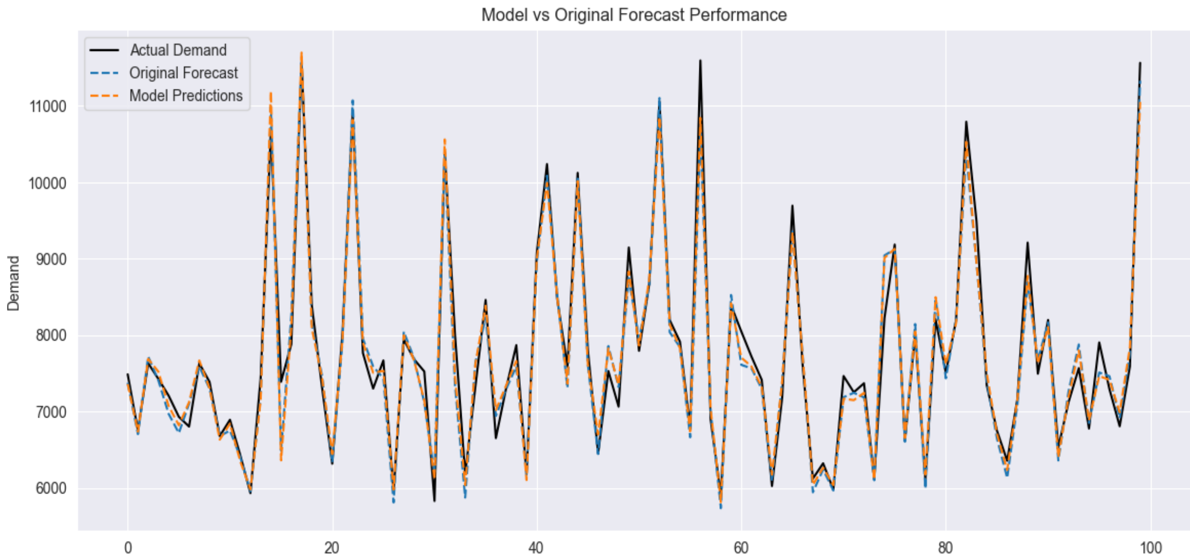

Figure 5.5: Line Plot of RFMF1's Performance vs Original Forecast Model Performance

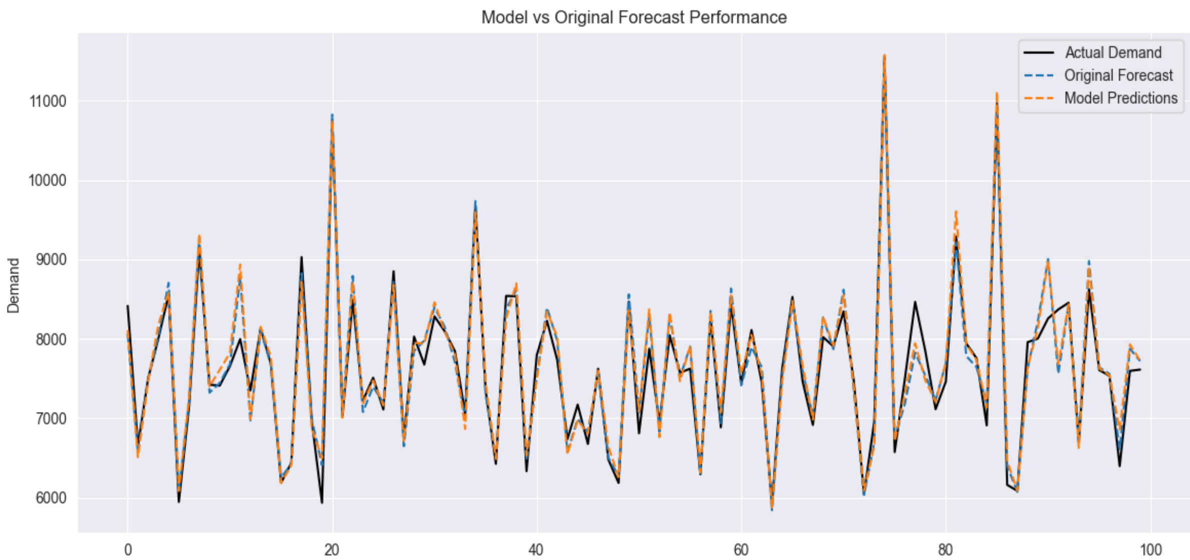

Figure 5.6: Line Plot of RFMF2's Performance vs Original Forecast Model Performance

5.5 XGBoost

Extreme Gradient Boosting (XGBoost) is a machine learning algorithm that utilises gradient boosting decision trees that generates fast and effective models used for forecasting, classification and regression problems.

As discussed above, forecasting has been seen to improve when incorporating the new weather forecast values combined with previous errors. The overall aim being to reduce the demand forecast error.

The XGBoost model discussed aims to reduce the demand forecast error by predicting demand directly rather than predicting the error and then updating the original forecast.

5.5.1 Model Construction

The base model involved using forecasted temperature, humidity, wind speed and rain, combining that with the hour, month and the day of week. Taking the model to the next level involved including the previous forecasted demand and the previous forecast error from 24 hours, 48 hours, 7 days and 14 days ago.

Assessing where a base model performs worse based on hour of day yields the following.

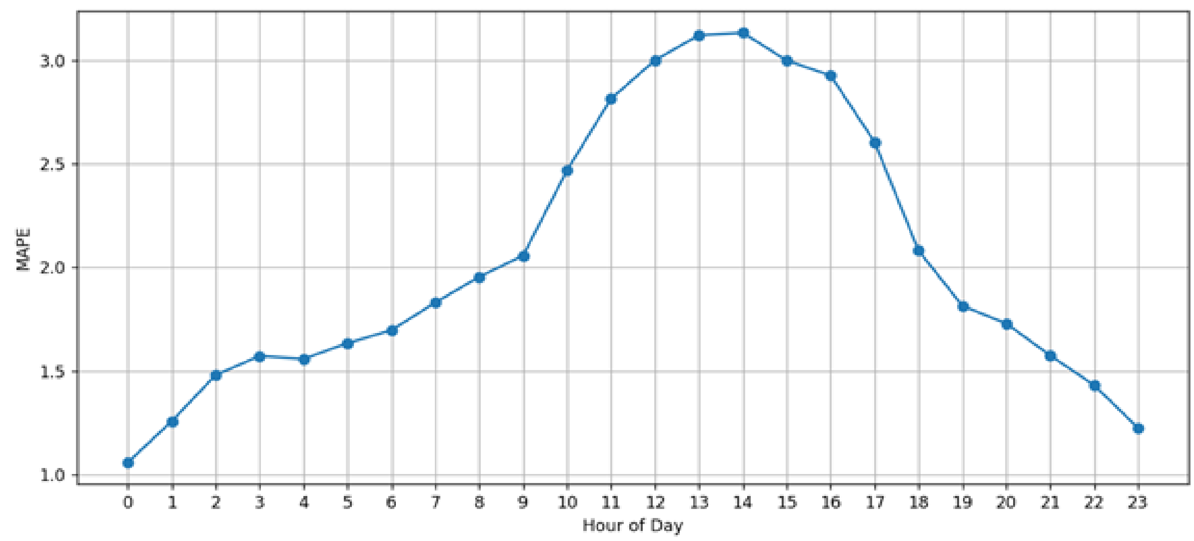

Figure 5.7: MAPE by Hour of the Day

Taking this error as a wave format, the model improved when variables were combined with a sin or cos wave. Specifically, combining hour with sin/cos wave and then multiplying by forecasted temperature improved the model.

5.5.2 Parameter Selection (Fine Tuning)

Fine tuning is especially important for XGBoost and a grid search was utilised to find the highest performing combination from a wide distribution of parameters. The following was found to be the most effective combination of parameters.

$$\begin{aligned} learning\_rate &= 0.1, \\ n\_estimators &= 150, \\ max\_depth &= 3, \\ subsample &= 0.8 \end{aligned}$$

5.5.3 Model Performance

Utilising the above parameters gave accuracy scores of

$$\begin{aligned} \text{MSE} &= 46526.86 \\ \text{MAPE} &= 2.042\% \end{aligned}$$

5.5.4 Combining variables

Introducing new variables as functions of other variables boosted the performance of XGBoost. Whilst in theory introducing variables such as Temperature * Humidity could improve the model, introducing them created unnecessary complexity that reduced the accuracy of the model. It could also been seen that XGBoost took these into account inside the algorithm.

5.6 Model Comparison

Comparison of all models tested against the baseline AEMO model (Table 5.5). The XG Boost model produced the highest level of accuracy of the models considered. This is further evident when observing prediction error distribution (Figure 5.8).

Table 5.5: Models compared by MSE and MAPE

		Measure	
		MSE	MAPE
Model	**AEMO**	55,159	2.190%
	Linear Regression	49,718	2.080%
	SARIMA	49,519	2.049%
	XGBoost	46,526	2.042%
	Random Forest	51,395	2.095%

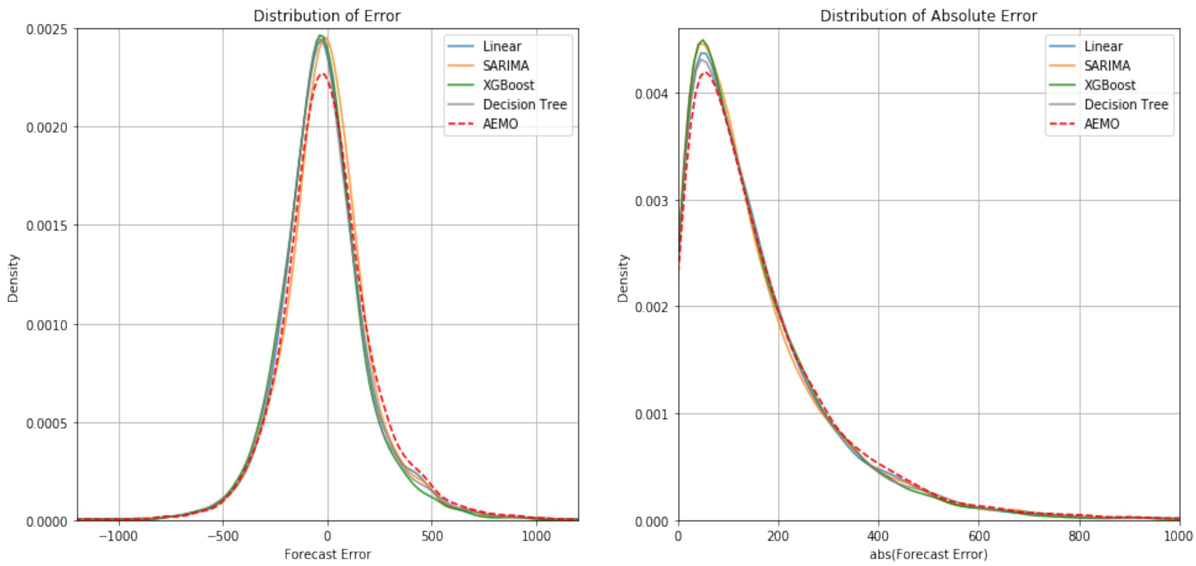

Figure 5.8: MSE Distribution

5.7 Summary of Key Findings

- **All models outperformed AEMO's model in terms of MSE and MAPE.** This indicates that there are likely underlying patterns in the AEMO residuals that are not currently captured in their model, therefore their model could be improved. This also validates forecasting methodologies which aims to use forecasting errors in modelling to further improve forecasting accuracy.
- **XGBoost performed best compared with linear regression, SARIMA and Decision trees.** This may indicate that some underlying patterns in AEMO's

forecast errors are likely non-linear, and therefore best forecasted by a black box method that can handle non-linear relationships.

- **SARIMA also performed strongly**, indicating seasonality in the forecast error. This reinforces EDA findings where correlations existed between lagged forecast errors and current forecast errors. The SARIMA model also performed quite strongly based on MSE and MAPE. Furthermore, the SARIMA model which only included the lagged error performed the strongest, which provides further evidence of this relationship.
- **Both machine learning models performed better when including weather related variables.** This may indicate a non-linear relationship between the forecast error (at least partly) and weather indicators (note, SARIMA would be limited in its ability to capture non-linearity).

Chapter 6

Discussion

6.1 Interpretation of results

The results of this project demonstrate the validity of using forecast errors in modelling to further improve electricity demand predictions. This is particularly important for short-term energy demand forecasting which relies on the precision of forecasts to balance electricity demand and supply rather than requiring an interpretable model.

While XGBoost was the best performing model for the datasets considered, this may not be the case for all forecasts, depending on the underlying patterns in the forecast errors. If patterns in forecast errors are linear, traditional models such as ARIMA may perform better to improve the forecast. On the other hand, this project demonstrated that non-linear trends were present in the forecast errors which allowed the XGBoost model to be the better performing model.

Both machine learning methods performed better when incorporating weather indicator variables which may point to different models capturing different types of relationships (e.g. SARIMA capturing the effect of the lagged error and XGBoost capturing effects of weather-related variables). This could indicate that modelling the forecast errors of the best model (XGBoost) with a different model (e.g. SARIMA) has the potential to produce even more accurate results.

6.2 Implications for energy planning

Future short-term electricity forecasts should consider how residuals could be used to further improve forecasting accuracy. This could involve decomposing the data in the first instance as others have done and discussed in the literature review, or incorporating residuals from an initial forecasted model in a subsequent model. Improving forecast accuracy will likely have the benefit of greater efficiency in managing electricity generation.

It should be noted that the method described in this report has proven useful to increase accuracy of forecast predictions which is essential for short-term forecasting, however may not be appropriate for longer-term forecasting where interpretability of the model is more important.

6.3 Limitations, challenges, and further research

While this project was able to improve on the AEMO forecast by modelling the forecast error, there is the potential that this could have been achieved more efficiently if a detailed AEMO forecasting methodology was available. This could have provided a better idea of what information or trends could be missing from the original forecast, and therefore which method would have been most useful to model the residuals.

This study only considered 12-hour interval due to the complexity of including several intervals and computational burden. Future research could consider longer or shorter forecast intervals to test the impact of modelling forecasting errors to improve accuracy for different intervals. Further research could also consider forecasting model errors using multiple models to capture the different patterns of errors.

Chapter 7

Conclusion and Further Issues

Hybrid models are known to improve the accuracy of electricity demand forecasts. This report took the concepts of a hybrid model to improve AEMO's short-term electricity demand forecast by incorporating AEMO forecast errors in a subsequent model to produce a more accurate prediction. The report found that for all models tested, the accuracy of the short-term forecast improved. This is valuable to industry as balancing the supply and demand of energy requires highly accurate forecasting. While this report only considered a 12-hour forecast interval, future studies could investigate different forecast intervals or multiple-iteration error-corrections to improve the accuracy of energy demand forecasts.

References

Abbasi, R.A., Javaid, N., Ghuman, M.N.J., Khan, Z.A., Ur Rehman, S. & Amanullah (2019). 'Short Term Load Forecasting Using XGBoost', *Advances in Intelligent Systems and Computing*, pp.1120–1131. doi:https://doi.org/10.1007/978-3-030-15035-8_108.

AEMO. (2023). Load forecasting. Available at: https://aemo.com.au/-/media/files/electricity/nem/security_and_reliability/power_system_ops/procedures/so_op_3710-load-forecasting.pdf?la=en [Accessed 24 Mar. 2025].

Ahmad, A.S., Hassan, M.Y., Abdullah, M.P., Rahman, H.A., Hussin, F., Abdullah, H. & Saidur, R. (2014). 'A review on applications of ANN and SVM for building electrical energy consumption forecasting', *Renewable and Sustainable Energy Reviews*, [online] 33, pp.102–109. doi:https://doi.org/10.1016/j.rser.2014.01.069.

Ahmad, T. & Chen, H. (2018). 'Potential of three variant machine-learning models for forecasting district level medium-term and long-term energy demand in smart grid environment', *Energy*, 160, pp.1008–1020. doi:https://doi.org/10.1016/j.energy.2018.07.084.

Ahmad, W., Ayub, N., Ali, T., Irfan, M., Awais, M., Shiraz, M. & Glowacz, A. (2020). 'Towards short term electricity load forecasting using improved support vector machine and extreme learning machine', *Energies*, 13(11), p.2907. doi:https://doi.org/10.3390/en13112907.

Amara, F., Agbossou, K., Dubé, Y., Kelouwani, S., Cardenas, A. & Hosseini, S.S. (2019). 'A residual load modeling approach for household short-term load forecasting application', *Energy and Buildings*, 187, pp.132–143. doi:https://doi.org/10.1016/j.enbuild.2019.01.009.

Andronikos, A., Tzelepi, M. & Tefas, A. (2023). 'Residual Error Learning for Electricity Demand Forecasting', In: Iliadis, L., Maglogiannis, I., Alonso, S., Jayne, C. & Pimenidis, E. (eds) *Engineering Applications of Neural Networks.* EANN 2023. Communications in Computer and Information Science, vol 1826. Springer, Cham. doi:https://doi.org/10.1007/978-3-031-34204-2_33.

Añel, J.A., Pérez-Souto, C., Bayo-Besteiro, S., Prieto-Godino, L., Bloomfield, H., Troccoli, A. & Laura (2024). 'Extreme weather events and the energy sector in 2021', *Weather Climate and Society.* doi:https://doi.org/10.1175/wcas-d-23-0115.1.

Ardakani, F.J. & Ardehali, M.M. (2014). 'Long-term electrical energy consumption forecasting for developing and developed economies based on different optimized models and historical data types', *Energy*, 65, pp.452–461. doi:https://doi.org/10.1016/j.energy.2013.12.031.

Boroojeni, K.G., Amini, M.H., Bahrami, S., Iyengar, S.S., Sarwat, A.I. & Karabasoglu, O. (2017). 'A novel multi-time-scale modeling for electric power demand forecasting: From short-term to medium-term horizon', *Electric Power Systems Research*, 142, pp.58–73. doi:https://doi.org/10.1016/j.epsr.2016.08.031.

Deng, X., Ye, A., Zhong, J., Xu, D., Yang, W., Song, Z., Zhang, Z., Guo, J., Wang, T., Tian, Y., Pan, H., Zhang, Z., Wang, H., Wu, C., Shao, J. & Chen, X. (2022).

‘Bagging–XGBoost algorithm based extreme weather identification and short-term load forecasting model’, *Energy Reports*, 8, pp.8661–8674. doi:https://doi.org/10.1016/j.egyr.2022.06.072.

Divina, F., García Torres, M., Goméz Vela, F.A. & Vázquez Noguera, J.L. (2019). ‘A Comparative Study of Time Series Forecasting Methods for Short Term Electric Energy Consumption Prediction in Smart Buildings’, *Energies*, 12(10), p.1934. doi:https://doi.org/10.3390/en12101934.

Ediger, V.Ş. & Akar, S. (2007). ‘ARIMA forecasting of primary energy demand by fuel in Turkey’, *Energy Policy*, 35(3), pp.1701–1708. doi: https://doi.org/10.1016/j.enpol.2006.05.009.

Ertuğrul, Ö.F., Tekin, H. & Tekin, R. (2020). ‘A novel regression method in forecasting short-term grid electricity load in buildings that were connected to the smart grid’, *Electrical Engineering*, 103, pp: 717-728. doi:https://doi.org/10.1007/s00202-020-01114-3.

Ghalehkhondabi, I., Ardjmand, E., Weckman, G.R. & Young, W.A. (2016). ‘An overview of energy demand forecasting methods published in 2005–2015’, *Energy Systems*, 8(2), pp.411–447. doi:https://doi.org/10.1007/s12667-016-0203-y.

Klyuev, R.V., Morgoev, I.D., Morgoeva, A.D., Gavrina, O.A., Martyushev, N.V., Efremenkov, E.A. & Mengxu, Q. (2022). ‘Methods of Forecasting Electric Energy Consumption: A Literature Review’, *Energies*, 15(23), p.8919. doi:https://doi.org/10.3390/en15238919.

Kopyt, M., Piotrowski, P. & Baczyński, D. (2024). ‘Short-Term Energy Generation Forecasts at a Wind Farm—A Multi-Variant Comparison of the Effectiveness and Performance of Various Gradient-Boosted Decision Tree Models’, *Energies*, 17(23), p.6194. doi:https://doi.org/10.3390/en17236194.

Koukaras, P., Mustapha, A., Mystakidis, A. & Tjortjis, C. (2024). ‘Optimizing Building Short-Term Load Forecasting: A Comparative Analysis of Machine Learning Models’, *Energies*, 17(6), p.1450. doi:https://doi.org/10.3390/en17061450.

Kuo, P.-H. & Huang, C.-J. (2018). ‘A High Precision Artificial Neural Networks Model for Short-Term Energy Load Forecasting’, *Energies*, 11(1), p.213. doi:https://doi.org/10.3390/en11010213.

Liu, X.-Q., Zhang, C., Zhou, Y. & Liao, H. (2021). ‘Temperature change and electricity consumption of the group living: A case study of college students’, *Science of The Total Environment*, 781, p.146574. doi:https://doi.org/10.1016/j.scitotenv.2021.146574.

Maia-Silva, D., Kumar, R. & Nateghi, R. (2020). ‘The critical role of humidity in modeling summer electricity demand across the United States’, *Nature Communications*, 11, p.1686. doi:https://doi.org/10.1038/s41467-020-15393-8.

Manno, A., Martelli, E. and Amaldi, E. (2022). ‘A Shallow Neural Network Approach for the Short-Term Forecast of Hourly Energy Consumption’, *Energies*, [online] 15(3), pp.958. doi:https://doi.org/10.3390/en15030958.

Marco G. Pinheiro, Sara C. Madeira, Alexandre P. Francisco, (2023). ‘Short-term electricity load forecasting—A systematic approach from system level to secondary substations’, *Applied Energy*, 332, pp.120493, ISSN 0306-2619, doi:https://doi.org/10.1016/j.apenergy.2022.120493.

Mystakidis, A., Koukaras, P., Tsalikidis, N., Ioannidis, D. and Tjortjis, C. (2024). ‘Energy Forecasting: A Comprehensive Review of Techniques and Technologies’, *Energies*, [online] 17(7), pp.1662. doi:https://doi.org/10.3390/en17071662.

OpenWeather. (2025). *Custom Weather Products.* [Online] Available at: https://home.openweathermap.org/marketplace [Accessed 29 Mar. 2025]

Pao, H.T. (2009). Forecasting energy consumption in Taiwan using hybrid nonlinear models. *Energy*, 34(10), pp.1438–1446. doi:https://doi.org/10.1016/j.energy.2009.04.026.

Papalexopoulos, A.D. and Hesterberg, T.C. (1990). 'A regression-based approach to short-term system load forecasting', *IEEE Transactions on Power Systems*, 5(4), pp.1535–1547. doi:https://doi.org/10.1109/59.99410.

Phyo, P.P. and Byun, Y.-C. (2021). 'Hybrid Ensemble Deep Learning-Based Approach for Time Series Energy Prediction', *Symmetry*, 13(10), pp.1942. doi:https://doi.org/10.3390/sym13101942.

Rakpho, P. and Yamaka, W. (2021). 'The forecasting power of economic policy uncertainty for energy demand and supply', *Energy Reports*, 7, pp.338–343. doi:https://doi.org/10.1016/j.egyr.2021.06.059.

Sanhudo, L., Rodrigues, J. and Filho, Ê.V. (2021). 'Multivariate time series clustering and forecasting for building energy analysis: Application to weather data quality control', *Journal of Building Engineering*, 35, pp.101996. doi:https://doi.org/10.1016/j.jobe.2020.101996.

Savić, S., Selakov, A. and Milošević, D. (2014). 'Cold and warm air temperature spells during the winter and summer seasons and their impact on energy consumption in urban areas', *Natural Hazards*, 73(2), pp.373–387. doi:https://doi.org/10.1007/s11069-014-1074-y.

Singh, S. and Yassine, A. (2018). 'Big Data Mining of Energy Time Series for Behavioral Analytics and Energy Consumption Forecasting', *Energies*, 11(2), pp.452. doi:https://doi.org/10.3390/en11020452.

Suganthi, L. and Samuel, A.A. (2012). 'Energy models for demand forecasting—A review', *Renewable and Sustainable Energy Reviews*, 16(2), pp.1223–1240. doi:https://doi.org/10.1016/j.rser.2011.08.014.

Tarmanini, C., Sarma, N., Gezegin, C. and Ozgonenel, O. (2023). 'Short term load forecasting based on ARIMA and ANN approaches', *Energy Reports*, 9, pp.550–557. doi:https://doi.org/10.1016/j.egyr.2023.01.060.

Wang, J., Li, P., Ran, R., Che, Y. and Zhou, Y. (2018). 'A Short-Term Photovoltaic Power Prediction Model Based on the Gradient Boost Decision Tree', *Applied Sciences*, 8(5), pp.689. doi:https://doi.org/10.3390/app8050689.

Zhang, S., Guo, Q., Smyth, R. and Yao, Y. (2022). 'Extreme temperatures and residential electricity consumption: Evidence from Chinese households', *Energy Economics*, pp.105890. doi:https://doi.org/10.1016/j.eneco.2022.105890.

Appendix

Appendix A: Data Processing

Packages used for data cleaning

```
import pandas as pd
import numpy as np

pd.options.mode.chained_assignment = None
```

Importing the data

```
# Forecast Data
## Reading data and formating data-time columns
df_forecast = pd.read_csv('data/forecastdemand_nsw.csv', names =
    ['id', 'region_id', 'period_id', 'forecast_demand',
    'date_time_current', 'date_time_future'], skiprows = 1)
df_forecast.date_time_current = pd.to_datetime(
    df_forecast.date_time_current, format = "%Y-%m-%d %H:%M:%S")
df_forecast.date_time_future = pd.to_datetime(
    df_forecast.date_time_future, format = "%Y-%m-%d %H:%M:%S")

## Using 'period_id' to round 'current time'
df_forecast["date_time_current_rounded"] = df_forecast.period_id.apply(
    lambda x: pd.Timedelta(hours = x/2))
df_forecast.date_time_current_rounded = df_forecast.date_time_future -
    df_forecast.date_time_current_rounded

# Demand Data
## Reading data and formating data-time columns
df_demand = pd.read_csv('data/totaldemand_nsw.csv', names =
    ['date_time', 'total_demand', 'region_id'], skiprows = 1)
df_demand.date_time = pd.to_datetime(df_demand.date_time,
    format = "%d/%m/%Y %H:%M")


# Forecast temperature data
df_weather_forecast = pd.read_csv('data/forecast_temperatre.csv')

df_weather_forecast = df_weather_forecast.rename(
  {'forecast dt iso': 'date_time_current_utc',
   'slice dt iso': 'date_time_future_utc',
   'temperature': 'temperature_future_forecast',
   'humidity': 'humidity_future_forecast',
```

```
    'rain': 'rain_future_forecast',
    'wind_speed': 'wind_speed_future_forecast'}, axis = 1)

df_weather_forecast["date_time_current_rounded"] =
    pd.to_datetime(df_weather_forecast.date_time_current_utc,
    format = "%Y-%m-%d %H:%M:%S +0000 UTC") + pd.Timedelta(hours = 10)
df_weather_forecast["date_time_future"] =
    pd.to_datetime(df_weather_forecast.date_time_future_utc,
    format = "%Y-%m-%d %H:%M:%S +0000 UTC") + pd.Timedelta(hours = 10)

df_weather_forecast =
    df_weather_forecast[['date_time_current_rounded', 'date_time_future',
      'temperature_future_forecast', 'humidity_future_forecast',
      'rain_future_forecast', 'wind_speed_future_forecast']]
```

Merging the data

```
#Merging Datasets
df_all = pd.merge(df_forecast, df_demand[["date_time", "total_demand"]],
    left_on = "date_time_future", right_on = "date_time").drop(
        columns = "date_time")

df_all = pd.merge(df_all,
    df_temperature[["date_time_30m", "temperature"]],
        left_on = "date_time_future", right_on = "date_time_30m")
df_all = df_all.drop(
    columns = ["date_time_30m", "region_id"]).rename(
        {"temperature": "temperature_future"}, axis = 1)

df_all = pd.merge(df_all,
    df_temperature[["date_time_30m", "temperature"]],
    left_on = "date_time_current_rounded", right_on = "date_time_30m")
df_all = df_all.drop(columns =
    "date_time_30m").rename({"temperature": "temperature_current"},
    axis = 1)

df_all = pd.merge(df_all, df_weather_forecast, on =
    ["date_time_current_rounded", "date_time_future"], how = 'left')
```

Format data.

```
df_all["forecast_interval"] = df_all.date_time_future -
    df_all.date_time_current_rounded
df_all["forecast_error"] = df_all.total_demand -
    df_all.forecast_demand
df_all["forecast_error_relative"] =
    df_all.forecast_error/df_all.total_demand

df_all["date_time_future_month"] = df_all.date_time_future.dt.month
df_all["date_time_future_year"] = df_all.date_time_future.dt.year
```

```python
df_all["date_time_future_weekday"] = df_all.date_time_future.dt.dayofweek
df_all["date_time_future_hour"] = df_all.date_time_future.dt.hour

df_all["week_day_name"] = df_all.date_time_future.dt.day_name()

df_all["isSaturday"] = df_all.week_day_name.apply(
    lambda x: 1 if x == 'Saturday' else 0)
df_all["isSunday"] = df_all.week_day_name.apply(
    lambda x: 1 if x == 'Sunday' else 0)

df_all["isDecember"] = df_all.date_time_future_month.apply(
    lambda x: 1 if x == 12 else 0)
df_all["isJanuary"] = df_all.date_time_future_month.apply(
    lambda x: 1 if x == 1 else 0)
df_all["isFebruary"] = df_all.date_time_future_month.apply(
    lambda x: 1 if x == 2 else 0)
df_all["isNovember"] = df_all.date_time_future_month.apply(
    lambda x: 1 if x == 11 else 0)
```

Appendix B: Models

B1. AEMO Model

Import packages

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from matplotlib.pyplot import figure
```

```
delta = 24

df_lag = df_all.loc[df_all.period_id == delta].sort_values(
    "date_time_future").reset_index(drop = True)
df_lag_temp =
df_lag.copy()[
  ["forecast_error", "forecast_error_relative", "date_time_future"]
  ].rename({"forecast_error" : "forecast_error_24h_ago",
            "forecast_error_relative": "forecast_error_relative_24h_ago",
            "date_time_future": "date_time_future_24h_ago"}, axis = 1)
df_lag["date_time_current_24h_ago"] =
    df_lag.date_time_current - pd.DateOffset(hours = 24)
df_lag["date_time_future_24h_ago"] =
    df_lag.date_time_future - pd.DateOffset(hours = 24)

df_lag = df_lag.loc[
      df_lag.date_time_future_24h_ago >= min(df_lag.date_time_future)]
df_lag = pd.merge(df_lag, df_lag_temp,
    on = "date_time_future_24h_ago", how = 'left')
df_lag = df_lag.loc[df_lag.forecast_error_relative_24h_ago.notna()]

train_test_split = 0.7
split_int = int(train_test_split * len(df_lag))
df_lag_train, df_lag_test = df_lag[:split_int], df_lag[split_int:]
```

```
mse = mean_squared_error(df_lag_test.forecast_demand,
    df_lag_test.total_demand)
mape =  mean_absolute_percentage_error(df_lag_test.forecast_demand,
    df_lag_test.total_demand)

print(f"Existing model MSE = {round(mse)}")
print(f"Existing model MAPE = {round(100*mape,2)}%")
```

B2. Linear Regression

Import packages

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import warnings

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from statsmodels.tsa.stattools import adfuller
from matplotlib.pyplot import figure
from statsmodels.graphics.api import qqplot
```

Linear Regression Model 1

```
x_columns = ["forecast_error_24h_ago"]
x = sm.add_constant(df_lag_train[x_columns])
x = sm.add_constant(x)
y = np.array(df_lag_train.forecast_error)

model = sm.OLS(y, x)
results = model.fit()
print(results.summary())

df_lag_test["lm_forecast_error_pred"] =
    results.predict(sm.add_constant(df_lag_test[x_columns]))
df_lag_test["lm_forecast_demand_new"] =
    df_lag_test.forecast_demand + df_lag_test.lm_forecast_error_pred

mse_lm1 = mean_squared_error(df_lag_test.lm_forecast_demand_new,
    df_lag_test.total_demand)
mape_lm1 = mean_absolute_percentage_error(
  df_lag_test.lm_forecast_demand_new,df_lag_test.total_demand)

print(f"\nNew model MSE = {round(mse_lm1)}")
print(f"New model MAPE = {round(100*mape_lm1,3)}%")
```

Linear Regression Model 2

```
x_columns = ["forecast_error_24h_ago", "Temperature", "Humidity",
            "Wind_speed", "Rain", "isSaturday", "isSunday", "isDecember",
            "isJanuary", "isNovember"]
x = sm.add_constant(df_lag_train[x_columns])
x = sm.add_constant(x)
y = np.array(df_lag_train.forecast_error)

model = sm.OLS(y, x)
results = model.fit()
print(results.summary())
```

```
df_lag_test["lm2_forecast_error_pred"] =
    results.predict(sm.add_constant(df_lag_test[x_columns]))
df_lag_test["lm2_forecast_demand_new"] = df_lag_test.forecast_demand +
    df_lag_test.lm2_forecast_error_pred

mse_lm2 = mean_squared_error(
    df_lag_test.lm2_forecast_demand_new, df_lag_test.total_demand)
mape_lm2 = mean_absolute_percentage_error(
    df_lag_test.lm2_forecast_demand_new, df_lag_test.total_demand)

print(f"\nNew model MSE = {round(mse_lm2)}")
print(f"New model MAPE = {round(100*mape_lm2,2)}%")
```

B2. SARIMA

Import packages

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import warnings

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from matplotlib.pyplot import figure
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
```

SARIMA models considered:

- order=(1,0,0), seasonal_order=(0, 0, 0, 0)
- order=(1,0,1), seasonal_order=(0, 0, 0, 0)
- order=(7,0,1), seasonal_order=(0, 0, 0, 0)
- order=(7,0,7), seasonal_order=(0, 0, 0, 0)
- order=(1,0,0), seasonal_order=(1, 0, 1, 7)
- order=(6,0,2), seasonal_order=(1, 0, 1, 7)
- order=(6,0,2), seasonal_order=(1, 0, 2, 7)
- order=(6,0,1), seasonal_order=(2, 0, 1, 7)
- order=(2,0,1), seasonal_order=(2, 0, 1, 7)

Model evaluation and tuning

```
sarimas = pd.DataFrame({"order":[(1,0,0), (1,0,1), (7,0,1), (7,0,7),
                                 (1,0,0), (6,0,2), (6,0,2), (6,0,1),
                                 (2,0,1)],

                        "seasonal_order": [(0, 0, 0, 0), (0, 0, 0, 0),
                                           (0, 0, 0, 0), (0, 0, 0, 0),
```

```
                                          (1, 0, 1, 7), (1, 0, 1, 7),
                                          (1, 0, 2, 7), (2, 0, 1, 7),
                                          (2, 0, 1, 7)]})
sarimas = sarimas.reset_index().rename({"index": "id"}, axis = 1)

columns = ["sarima_id", "hour_of_day", "order", "seasonal_order",
    "ljung_val", "ljung_p", "jb_val", "jb_p", "hetro_val", "hetro_p",
    "skew", "kurtosis", "aic", "bic", "n_observations", "mse_pre",
    "mse_post", "mape"]
sarima_tune = pd.DataFrame(columns = columns)

period_id = 24

hours_all = [0, 4, 8, 12, 16, 20]

for hour_of_day in hours_all:
    for index, row in sarimas.iterrows():
        order = row["order"]
        seasonal_order = row["seasonal_order"]
        sarima_id = row["id"]

        df_all_delta = df_all.loc[(df_all.period_id == period_id) &
        (df_all.date_time_future_hour == hour_of_day) &
        (df_all.date_time_future.dt.minute == 0)].sort_values(
            "date_time_future").reset_index(drop = True)

        # Model fit
        model = ARIMA(df_all_delta.forecast_error, order = order,
            seasonal_order = seasonal_order)
        model_fit = model.fit()
        df_all_delta["predicted_forecast_error"] = model_fit.fittedvalues
        df_all_delta["new_forecast"] = df_all_delta.forecast_demand +
            df_all_delta.predicted_forecast_error

        # Model Evaluation (MSE)
        mse_pre = mean_squared_error(df_all_delta.total_demand,
            df_all_delta.forecast_demand)
        mse_post = mean_squared_error(df_all_delta.total_demand,
            df_all_delta.new_forecast)
        mape = mean_absolute_percentage_error(df_all_delta.total_demand,
            df_all_delta.new_forecast)

        # Model Evaluation (Crit values)
        stat_tests = pd.read_html(model_fit.summary().tables[2].as_html(),
            header=None,index_col=0)[0]
        ljung_val, ljung_p = stat_tests[1].iloc[0], stat_tests[1].iloc[1],
        jb_val, jb_p = stat_tests[3].iloc[0], stat_tests[3].iloc[1],
        hetro_val, hetro_p = stat_tests[1].iloc[2], stat_tests[1].iloc[3],
```

```
        skew, kurtosis = stat_tests[3].iloc[2], stat_tests[3].iloc[3]

        # Model Evaluation (AIC, BIC)
        stat_tests = pd.read_html(model_fit.summary().tables[0].as_html(),
            header=None,index_col=0)[0]
        aic, bic = stat_tests[3].iloc[2], stat_tests[3].iloc[3]
        n_observations = stat_tests[3].iloc[0]
        sarima_tune = sarima_tune.append(pd.DataFrame([[sarima_id,
            hour_of_day, order, seasonal_order, ljung_val, ljung_p,
            jb_val, jb_p, hetro_val, hetro_p, skew, kurtosis, aic,
            bic, n_observations, mse_pre, mse_post, mape]],
            columns=columns), ignore_index=True)
```

```
sarima_tune["mse_improvement"] = round(100*(sarima_tune.mse_pre -
    sarima_tune.mse_post)/sarima_tune.mse_pre)
sarima_tune = pd.merge(sarima_tune, sarimas, on =
    ["order", "seasonal_order"], how = "left").sort_values("id")

plot = sarima_tune.groupby(["order", "seasonal_order", "hour_of_day"],
    as_index = False).mean()
sns.lineplot(data = plot, x = 'hour_of_day', y = 'mape', hue = 'id',
    palette = 'pastel', alpha = 1, linestyle = '--')
```

```
sarima_tune["mse_improvement"] = round(100*(sarima_tune.mse_pre -
    sarima_tune.mse_post)/sarima_tune.mse_pre)

plot = sarima_tune.groupby(["order", "seasonal_order", "hour_of_day"],
    as_index = False).mean()
sns.lineplot(data = plot, x = 'hour_of_day', y = 'mse_improvement',
    hue = 'id', palette = 'pastel', alpha = 1, linestyle = '--')
```

Parameters chosen

```
period_id = 24
arima_order = (6,0,2)
arima_season_order = (1, 0, 1, 7)

train_test_split = 0.7
```

textbf{SARIMA Model 1 - Without Exogenous Variables}

```
df_predict = pd.DataFrame(columns = ["period_id", "date_time_future",
    "new_forecast", "forecast_demand", "total_demand"])

for hour_of_day in set(df_all.date_time_future_hour):
    df_delta = df_all.loc[(df_all.period_id == period_id) &
        (df_all.date_time_future_hour == hour_of_day) &
        (df_all.date_time_future.dt.minute == 0)].sort_values(
            "date_time_future").reset_index(drop = True)

    # Test/Train split
```

```
    split_int = int(train_test_split * len(df_delta))
    df_delta_train, df_delta_test =
        df_delta[:split_int], df_delta[split_int:]
    x_all, x_train, x_test = df_delta.forecast_error,
        df_delta_train.forecast_error, df_delta_test.forecast_error

    # Model - Train Data
    arima_model_train = ARIMA(x_train, order = arima_order,
        seasonal_order = arima_season_order)
    arima_mode_train_fit = arima_model_train.fit()

    # Model - Test Data
    arima_model_test = ARIMA(x_all, order = arima_order,
        seasonal_order = arima_season_order)
    arima_model_test_fit = arima_model_test.filter(
        arima_mode_train_fit.params)

    # Predicted Values
    arima_model_test_predict =
        arima_model_test_fit.predict().loc[split_int:]

    # Calculate new forecast
    df_delta_test["predicted_forecast_error"] = arima_model_test_predict
    df_delta_test["new_forecast"] = df_delta_test.forecast_demand +
        df_delta_test.predicted_forecast_error

    # Model evaluation
    mse_pre = mean_squared_error(df_delta_test.total_demand,
        df_delta_test.forecast_demand)
    mse_post = mean_squared_error(df_delta_test.total_demand,
        df_delta_test.new_forecast)
    mape_pre = mean_absolute_percentage_error(df_delta_test.total_demand,
        df_delta_test.forecast_demand)
    mape_post = mean_absolute_percentage_error(df_delta_test.total_demand,
        df_delta_test.new_forecast)

    df_predict = pd.concat([df_predict, df_delta_test[["period_id",
        "date_time_future", "new_forecast", "forecast_demand",
        "total_demand"]]])
```

textbf{SARIMA Model 2 - With Exogenous Variables}

```
df_predict_with_exog = pd.DataFrame(columns = ["period_id",
    "date_time_future", "new_forecast", "forecast_demand",
    "total_demand"])
exog_vars = ["Temperature", "Humidity", "Wind_speed", "Rain"]

for hour_of_day in set(df_all.date_time_future_hour):
    df_delta = df_all.loc[(df_all.period_id == period_id) &
```

```
(df_all.date_time_future_hour == hour_of_day) &
(df_all.date_time_future.dt.minute == 0)].sort_values(
    "date_time_future").reset_index(drop = True)

# Test/Train split
split_int = int(train_test_split * len(df_delta))
df_delta_train, df_delta_test =
    df_delta[:split_int], df_delta[split_int:]
x_all, x_train, x_test =
df_delta.forecast_error, df_delta_train.forecast_error,
df_delta_test.forecast_error
exog_all, exog_train, exog_test =
df_delta[exog_vars], df_delta_train[exog_vars],
df_delta_test[exog_vars]

# Model - Train Data
arima_model_train = ARIMA(x_train, exog = exog_train,
    order = arima_order, seasonal_order = arima_season_order)
arima_mode_train_fit = arima_model_train.fit()

# Model - Test Data
arima_model_test = ARIMA(x_all, exog = exog_all,
    order = arima_order,
    seasonal_order = arima_season_order)
arima_model_test_fit =
  arima_model_test.filter(arima_mode_train_fit.params)

# Predicted Values
arima_model_test_predict =
  arima_model_test_fit.predict().loc[split_int:]

# Calculate new forecast
df_delta_test["predicted_forecast_error"] = arima_model_test_predict
df_delta_test["new_forecast"] = df_delta_test.forecast_demand
    + df_delta_test.predicted_forecast_error

# Model evaluation
mse_pre = mean_squared_error(df_delta_test.total_demand,
    df_delta_test.forecast_demand)
mse_post = mean_squared_error(df_delta_test.total_demand,
    df_delta_test.new_forecast)
mape_pre = mean_absolute_percentage_error(df_delta_test.total_demand,
    df_delta_test.forecast_demand)
mape_post = mean_absolute_percentage_error(df_delta_test.total_demand,
    df_delta_test.new_forecast)

df_predict_with_exog = pd.concat([df_predict_with_exog,
    df_delta_test[["period_id", "date_time_future",
```

```
            "new_forecast", "forecast_demand", "total_demand"]]])
```

Model evaluation

```
df_predict["forecast_error_old"] =
    df_predict.total_demand - df_predict.forecast_demand
df_predict["forecast_error_new"] =
    df_predict.total_demand - df_predict.new_forecast
df_predict_with_exog["forecast_error_new"] =
    df_predict_with_exog.total_demand - df_predict_with_exog.new_forecast

mse_pre = mean_squared_error(
      df_predict.total_demand, df_predict.forecast_demand)
mse_sarima = mean_squared_error(
      df_predict.total_demand, df_predict.new_forecast)
mse_sarima_with_exog = mean_squared_error(
      df_predict_with_exog.total_demand,
      df_predict_with_exog.new_forecast)

mape_pre = mean_absolute_percentage_error(
      df_predict.total_demand, df_predict.forecast_demand)
mape_sarima = mean_absolute_percentage_error(
      df_predict.total_demand, df_predict.new_forecast)
mape_sarima_with_exog = mean_absolute_percentage_error(
      df_predict_with_exog.total_demand,
      df_predict_with_exog.new_forecast)
```

B3. Random Forest

Import packages

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Processing

```
# Filter for PERIODID 24 and sort
df_sliced = df[df["period_id"] == 24].copy()
df_sliced = df_sliced.sort_values("date_time_future")
df_sliced = df_sliced.dropna()

# Get forecast error
df_sliced['forecast_error'] = df_sliced['total_demand'] -
    df_sliced['forecast_demand']
df_sliced['forecast_error_lag24h'] = df_sliced.sort_values(
      'date_time_current_rounded')['forecast_error'].shift(24)

# Check for NaN counts in key columns
```

```python
print("\nNaN counts in key columns:")
for col in ['demand_lag_24h', 'demand_lag_48h', 'demand_lag_7d',
    'forecast_error_lag12h']:
    if col in df_sliced.columns:
        print(f"{col}: {df_sliced[col].isna().sum()} NaNs")
```

RFMF1 - Random Forest Model 1

```python
# Define features that would be available at prediction
# time (12 hours ahead)
features = [
    # Basic time features
    # Original forecast
    'forecast_demand',
    'forecast_error_lag24h'
]
# Define target
target = 'total_demand'

# Create the modeling dataframe
model_df = df_sliced[features + [target] +
    ['date_time_current_rounded']].copy()

# Print the shape before dropping missing values
print(f"\nShape before dropping missing values: {model_df.shape}")

# Drop rows with NaN values
model_df = model_df.dropna()
print(f"Shape after dropping NaN values: {model_df.shape}")

# If we still have no data, show a clear error and exit
if len(model_df) == 0:
    print("ERROR: No data left after dropping NaN values!")
    import sys
    sys.exit(1)

# Sort data to ensure temporal order
model_df = model_df.sort_values(
      'date_time_current_rounded').reset_index(drop=True)

# Split data temporally - use 70-30 split
X = model_df[features]
y = model_df[target]
train_size = 0.7
split_idx = int(len(model_df) * train_size)

# Split into train/test
X_train = X.iloc[:split_idx]
y_train = y.iloc[:split_idx]
```

```
X_test = X.iloc[split_idx:]
y_test = y.iloc[split_idx:]

# Train model
model = RandomForestRegressor(
    n_estimators=200,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_original = X_test["forecast_demand"]
```

Evaluate performance

```
# Evaluate performance
def calculate_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mape = np.mean(
        np.abs((y_true - y_pred) / np.maximum(0.001, y_true))) * 100
    return mse, mape

# Model metrics
model_mse, model_mape = calculate_metrics(y_test, y_pred)

# Original forecast metrics
original_mse, original_mape = calculate_metrics(y_test, y_pred_original)

# Print formatted results
print("\nModel Performance:")
print(f"- MSE: {model_mse:.3f}")
print(f"- MAPE: {model_mape:.3f}%")

print("\nOriginal Forecast Performance:")
print(f"- MSE: {original_mse:.3f}")
print(f"- MAPE: {original_mape:.3f}%")

# Calculate improvement percentages
improvement_mse = (1 - model_mse/original_mse) * 100
improvement_mape = (1 - model_mape/original_mape) * 100

print("\nImprovement Over Original Forecast:")
print(f"- MSE: {improvement_mse:.3f}%")
print(f"- MAPE: {improvement_mape:.3f}%")
```

```
# Feature importance
feature_importance = pd.DataFrame(
    {'Feature': features,
     'Importance': model.feature_importances_}
).sort_values('Importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)
```

RFMF2 - Random Forest Model 2

```
# Define features that would be available at prediction
# time (12 hours ahead)
features = [
    'forecast_demand',
    'Temperature',
    'Humidity',
    'Wind_speed',
    'Rain',
    'forecast_error_lag24h'
]


# Define target
target = 'total_demand'


# Print the number of NaN values for each feature
print("\nNaN counts in features:")
for feature in features:
    print(f"{feature}: {df_sliced[feature].isna().sum()} NaNs")

# Create the modeling dataframe
model_df = df_sliced[features + [target] ].copy()
# Drop rows with NaN values
model_df = model_df.dropna()
print(f"Shape after dropping NaN values: {model_df.shape}")


# Split data temporally - using 70-30 split
X = model_df[features]
y = model_df[target]
train_size = 0.7
split_idx = int(len(model_df) * train_size)

# Split into train/test
X_train = X.iloc[:split_idx]
y_train = y.iloc[:split_idx]
X_test = X.iloc[split_idx:]
```

```python
y_test = y.iloc[split_idx:]


# Train model
model = RandomForestRegressor(
    n_estimators=200,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1
)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_original = X_test["forecast_demand"]
```

Evaluate performance

```python
def calculate_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mape = np.mean(
        np.abs((y_true - y_pred) / np.maximum(0.001, y_true))) * 100
    return mse, mape

# Model metrics
model_mse, model_mape = calculate_metrics(y_test, y_pred)

# Original forecast metrics
original_mse, original_mape = calculate_metrics(y_test, y_pred_original)

# Print formatted results
print("\nModel Performance:")
print(f"- MSE: {model_mse:.3f}")
print(f"- MAPE: {model_mape:.3f}%")

print("\nOriginal Forecast Performance:")
print(f"- MSE: {original_mse:.3f}")
print(f"- MAPE: {original_mape:.3f}%")

# Calculate improvement percentages
improvement_mse = (1 - model_mse/original_mse) * 100
improvement_mape = (1 - model_mape/original_mape) * 100

print("\nImprovement Over Original Forecast:")
print(f"- MSE: {improvement_mse:.3f}%")
print(f"- MAPE: {improvement_mape:.3f}%")
```

```
# Feature importance
feature_importance = pd.DataFrame(
    {'Feature': features,
     'Importance': model.feature_importances_}
).sort_values('Importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)
```

B4. XGBoost

Import packages

```
import pandas as pd
import numpy as np
from sklearn.model_selection import RandomizedSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
import xgboost as xgb
import shap
import matplotlib.pyplot as plt
```

Data Processing

```
df['24hrpreverrors'] = df['forecast_error'].shift(24)
df['48hrpreverrors'] = df['forecast_error'].shift(48)
df['7daypreverrors'] = df['forecast_error'].shift(24 * 7)
df['14daypreverrors'] = df['forecast_error'].shift(24 * 14)
# Time-based features
df["Hour"] = df.date_time_future.dt.hour
df["MonthNumb"] = df.date_time_future.dt.month
df["Day of week"] = df.date_time_future.dt.dayofweek

df = df.dropna()



# Encode Hour as cyclic features
df["hour_sin"] = np.sin(2 * np.pi * df["Hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["Hour"] / 24)

# Interaction features
df["hour_x_temp"] = df["Hour"] * df["Temperature"]
df["month_x_temp"] = df["MonthNumb"] * df["Temperature"]
df["hour_x_forecast"] = df["Hour"] * df["forecast_demand"]
df["temp_x_forecast"] = df["Temperature"] * df["forecast_demand"]
df["temp_x_hour_sin"] = df["Temperature"] * df["hour_sin"]
df["temp_x_hour_cos"] = df["Temperature"] * df["hour_cos"]
df["forecast_x_hour_sin"] = df["forecast_demand"] * df["hour_sin"]
df["forecast_x_hour_cos"] = df["forecast_demand"] * df["hour_cos"]
```

```
df["forecast_24_hour_cos"] = df["24hrpreverrors"] * df["hour_cos"]
df["forecast_24_hour_sin"] = df["24hrpreverrors"] * df["hour_sin"]
```

Parameter Selection

```
features = [
    'Temperature', 'Humidity',
    'Wind_speed', 'Rain',
    'hour_sin', 'hour_cos',
    'MonthNumb', 'Day of week',
    'forecast_demand',
    '24hrpreverrors',
    '48hrpreverrors', '7daypreverrors', '14daypreverrors',
    'hour_x_temp', 'month_x_temp', 'hour_x_forecast', 'temp_x_forecast',
    'temp_x_hour_sin', 'temp_x_hour_cos',
    'forecast_x_hour_sin', 'forecast_x_hour_cos',
    'forecast_24_hour_cos', 'forecast_24_hour_sin'

]

train_df = df[(df['date_time_future'] >= "2017-10-07 23:00:00") &
    (df['date_time_future'] <= "2020-03-05 23:00:00")]
test_df = df[(df['date_time_future'] > "2020-03-06 23:00:00") &
    (df['date_time_future'] <= "2021-03-17 23:00:00")]

# Prepare train/test split sets
X_train = train_df[features]
y_train = train_df['total_demand']
X_test = test_df[features]
y_test = test_df['total_demand']

# Baseline metrics from forecast and total demand
original_mse = mean_squared_error(y_test,
    test_df['forecast_demand'])
original_mape = mean_absolute_percentage_error(y_test,
    test_df['forecast_demand']) * 100

# TimeSeriesSplit to respect time order
tscv = TimeSeriesSplit(n_splits=3)

# Parameter grid for randomized search
param_dist = {
    'n_estimators': [100, 150, 200, 250],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.03, 0.05, 0.1],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0]
}
```

```python
# Create base model
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror',
    tree_method='hist',
    random_state=42
)

# Randomized search
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_dist,
    n_iter=2000,
    scoring='neg_mean_absolute_percentage_error',
    cv=tscv,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# Run the search
random_search.fit(X_train, y_train)

# Use the best model
model = random_search.best_estimator_

# Optional: Print best parameters
print("Best Parameters:", random_search.best_params_)

###Output learning_rate=0.1, n_estimators=150, max_depth=3,
### subsample=0.8
```

Tuned Model

```python
X_train = train_df[features]
y_train = train_df['total_demand']
X_test = test_df[features]
y_test = test_df['total_demand']

# Baseline metrics from forecast and total demand
original_mse = mean_squared_error(y_test, test_df['forecast_demand'])
original_mape = mean_absolute_percentage_error(y_test,
    test_df['forecast_demand']) * 100

# Model creation, taken from fine tuning
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
    tree_method='hist', random_state=42)

model = xgb.XGBRegressor(
    objective='reg:squarederror',
```

```
    learning_rate=0.1,
    n_estimators=150,
    max_depth=3,
    subsample=0.8,
    random_state=42
)

model.fit(X_train, y_train)


# Predict and evaluate
y_pred = model.predict(X_test)
model_mse = mean_squared_error(y_test, y_pred)
model_mape = mean_absolute_percentage_error(y_test, y_pred) * 100
```

Evaluate Performance

```
# Results
print(f"Original Forecast MSE: {original_mse:.2f}")
print(f"Original Forecast MAPE: {original_mape:.3f}%")
print(f"XGBoost Tuned Model MSE: {model_mse:.2f}")
print(f"XGBoost Tuned Model MAPE: {model_mape:.3f}%")


# Explain model predictions using SHAP
explainer = shap.Explainer(model, X_test)
shap_values = explainer(X_test)
shap_df = pd.DataFrame(shap_values.values, columns=X_test.columns)

# Forecast demand skews the plot so hide it
filtered_shap_values = shap_df.drop(columns=["forecast_demand"])
filtered_X_test = X_test.drop(columns=["forecast_demand"])

shap.summary_plot(
    filtered_shap_values.values,
    features=filtered_X_test,
    feature_names=filtered_X_test.columns
)
```

Appendix C: Plots

Packages used for plotting data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 4.1

```
df_demand_14d = df_demand[['date_time','total_demand']].copy()

df_demand_14d['dem_14d'] =
    df_demand.total_demand.rolling(window=672).mean()

plt.figure(figsize=(6, 4))
plt.plot(df_demand_14d['date_time'], df_demand_14d['dem_14d'],
    label='14-Day Rolling Avg', color='blue')
plt.xlabel('Year')
plt.ylabel('Total Demand (MW)')
plt.legend()
plt.show()
```

Figure 4.2

```
df_demand_month = df_demand[['date_time','total_demand']].copy()

df_demand_month['month'] = df_demand_month.date_time.dt.month
df_demand_month['month_name'] =
    df_demand_month.date_time.dt.month_name().str[:3]

plt.figure(figsize = (6,4))
sns.boxplot(
    data = df_demand_month.groupby(
      "date_time", as_index = False).first().sort_values("month"),
      x = 'month_name', y = "total_demand", hue = 'month',
      palette = 'Blues', showfliers = False, legend = False)
plt.xlabel('Month')
plt.ylabel('Total Demand (MW)')
plt.grid(alpha = 0.5)
plt.show()
```

Figure 4.3

```
df_demand_weekday = df_demand[['date_time','total_demand']].copy()

df_demand_weekday['weekday'] = df_demand_weekday.date_time.dt.day_of_week
df_demand_weekday['weekday_name'] =
    df_demand_weekday.date_time.dt.day_name().str[:3]

plt.figure(figsize = (6,4))
sns.boxplot(data = df_demand_weekday.groupby("date_time",
  as_index = False).first().sort_values("weekday"),
  x = 'weekday_name', y = "total_demand",
  hue = 'weekday', palette = 'Blues', showfliers = False,
  legend = False)
plt.grid(alpha = 0.5)
plt.xlabel('Day of the Week')
```

```
plt.ylabel('Total Demand (MW)')
plt.show()
```

Figure 4.4

```
df_hour = df_all.copy()
df_hour["date_time_future_hour"] = df_hour.date_time_future.dt.hour
df_hour = df_hour.sort_values("date_time_future_hour")



plt.figure(figsize = (12,6))
sns.boxplot(data = df_hour.groupby(
  "date_time_future", as_index = False).first().sort_values(
      "date_time_future_hour"),
  x="date_time_future_hour", y = "total_demand",
  palette = 'Blues',   showfliers = False)
plt.grid(alpha = 0.5)
plt.title("Hour vs Total Demand");

time_decomposition_error_plots(df = df_hour,
    x = "date_time_future_hour", time_interval = "Hour",
    show_outliers = False, forecast_interval = 12,
    show_relative_error_all = True,
    show_relative_error_interval = True)
```

Figure 4.5

```
forecast_df['forecast_hours'] = (forecast_df['DATETIME'] -
    forecast_df['LASTCHANGED']).dt.total_seconds() / 3600


merged_df = forecast_df.merge(
    actual_df,
    on=['DATETIME'],
    how='inner'
)


hours = [6, 12, 18, 24]
dfs = {
    h: merged_df[
        merged_df['forecast_hours'].round() == h].sample(n=3000,
          random_state=42)
    for h in hours
}


fig, axes = plt.subplots(2, 2, figsize=(8, 6))
for ax, h in zip(axes.flat, hours):
```

```
    sns.regplot(
        data=dfs[h],
        x='TOTALDEMAND',
        y='FORECASTDEMAND',
        line_kws={'color': 'red'},
        ax=ax
    )
    ax.set_title(f'{h}-Hour Ahead Forecast')
    ax.set_xlabel('Actual Demand')
    ax.set_ylabel('Forecasted Demand')
    ax.axhline(y = 9000,
               color = 'green')
plt.legend(['Correlation points', 'Trendline','',
                    'Forecasted = 9000'])
plt.tight_layout()
plt.show()
```

Figure 4.6

```
df_corr_demand = df_all[['total_demand',
    'temperature_future_forecast','humidity_future_forecast',
    'rain_future_forecast','wind_speed_future_forecast']]

df_corr_demand = df_corr_demand.rename(
  columns={'temperature_future_forecast': 'Temperature Forecast',
          'humidity_future_forecast': 'Humidity Forecast',
          'rain_future_forecast': 'Rain Forecast',
          'wind_speed_future_forecast': 'Wind Speed Forecast'})

correlation_demand = df_corr_demand.corr()
correlationsD = correlation_demand['total_demand'].drop('total_demand')

plt.figure(figsize=(10, 6))
correlationsD.sort_values().plot(kind='barh',
    color=plt.cm.coolwarm(np.abs(correlationsD)/max(abs(correlationsD))))
plt.xlabel('Correlation Coefficient')
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()
```

Figure 4.7

```
hourly_temp = temperature_df.groupby(
      ['HOUR', 'LOCATION'])['TEMPERATURE'].mean().reset_index()
print(f"Aggregated temperature data: {len(hourly_temp)} rows")

merged_df = pd.merge(
    demand_df,
    hourly_temp,
```

```
    left_on='HOUR',
    right_on='HOUR',
    how='inner'
)


plt.figure(figsize=(10, 6))
plt.scatter(merged_df['TEMPERATURE'], merged_df['TOTALDEMAND'],
  alpha=0.5)
plt.title('Relationship between Temperature and Electricity Demand')
plt.xlabel('Temperature (°C)')
plt.ylabel('Total Demand (MW)')
plt.grid(True, alpha=0.3)

# Add trend line
z = np.polyfit(merged_df['TEMPERATURE'], merged_df['TOTALDEMAND'], 2)
p = np.poly1d(z)
temp_range = np.linspace(merged_df['TEMPERATURE'].min(),
    merged_df['TEMPERATURE'].max(), 100)
plt.plot(temp_range, p(temp_range), "r--", linewidth=2)

plt.savefig('temperature_vs_demand_scatter.png')
```

Figure 4.8

```
interval = 60*60 #sets the interval in seconds
df_forecast["forecast_interval"] = df_forecast.date_time_prediction -
    df_forecast.date_time_forecast
df_forecast.forecast_interval = df_forecast.forecast_interval.apply(
      lambda x: x.total_seconds()/interval)


interval_min, interval_max = 23 , 25 #sets a window for forecast periods
df_forecast_near24hour =
    df_forecast.loc[(df_forecast.forecast_interval > interval_min) &
    (df_forecast.forecast_interval < interval_max)]
df_forecast_near24hour["date_time_forecast_rounded"] =
    df_forecast_near24hour.date_time_forecast.apply(
        lambda x: x.round(freq='30min'))
df_forecast_near24hour_1instance =
    df_forecast_near24hour.loc[
        df_forecast_near24hour.groupby(
            "date_time_forecast_rounded")["forecast_interval"].idxmax()]


df_forecast_near24hour_1instance_with_demand =
  pd.merge(df_forecast_near24hour_1instance,
    df_demand, left_on = "date_time_forecast_rounded",
    right_on = "date_time")
```

```
df_forecast_near24hour_1instance_with_demand["forecast_error"] =
  df_forecast_near24hour_1instance_with_demand.total_demand -
  df_forecast_near24hour_1instance_with_demand.forecast_demand

df_forecast_near24hour_1instance_with_demand_temperature =
    pd.merge(df_forecast_near24hour_1instance_with_demand,
        df_temperature, left_on = "date_time_forecast_rounded",
        right_on = "date_time")
df_forecast_near24hour_1instance_with_demand_temperature[
  "forecast_error_relative"] =
  df_forecast_near24hour_1instance_with_demand_temperature.forecast_error/
  df_forecast_near24hour_1instance_with_demand_temperature.total_demand

df_plot = df_forecast_near24hour_1instance_with_demand_temperature[[
  "temperature", "forecast_error", "forecast_error_relative"]].copy()
df_plot.temperature = df_plot.temperature.round()

plt.figure(figsize = (12,7))
sns.boxplot(data=df_plot, x="temperature", y="forecast_error",
    fliersize = 1)
plt.axhline(0, color='r', alpha = 0.2)
plt.xticks(rotation = 90);
plt.title("Accuracy of forecasting 24h into the future")
```

Figure 4.9

```
plt.figure(figsize = (12,7))
sns.boxplot(data=df_plot, x="temperature",
    y="forecast_error_relative", fliersize = 1)
plt.axhline(0, color='r', alpha = 0.2)
plt.xticks(rotation = 90);
plt.ylabel("Forecast Error as Portion of Actual Demand")
```

Figure 4.10 & Figure 4.11

```
for i, delta in enumerate([12, 24, 36, 48]):
    df_all_delta = df_all.loc[
      df_all.period_id == delta].sort_values(
        "date_time_future").reset_index(drop = True)
    delta_24h_later = 48 - delta
    previous_lag = 48

    x = df_all_delta.forecast_error_relative[
        previous_lag:len(df_all_delta)]
    y = df_all_delta.forecast_error_relative[
        0:len(df_all_delta)-previous_lag]

def check_stationarity(series):
```

```
    result = adfuller(series.values)

    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

    if (result[1] <= 0.05) & (result[4]['5%'] > result[0]):
        print("\u001b[32mStationary\u001b[0m")
    else:
        print("\x1b[31mNon-stationary\x1b[0m")

fig1, ax1 = plt.subplots(2,2, figsize = (18, 18))
fig2, ax2 = plt.subplots(2,2, figsize = (18, 18))

i_subplot = {0: [0,0], 1: [0,1], 2: [1,0], 3: [1,1]}

for i, period_id in enumerate([12, 24, 36, 48]):
    print(f"Forecast Interval = {round(period_id/2)}")

    df_all_delta = df_all.loc[df_all.period_id ==
      period_id].sort_values(
        "date_time_future").reset_index(drop = True)

    check_stationarity(df_all_delta.forecast_error_relative)

    plot_acf(df_all_delta.forecast_error_relative, lags = 100,
      ax = ax1[i_subplot[i][0]][i_subplot[i][1]])
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_xlabel('lag')
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_title(f'Forecast Interval =
      {round(period_id/2)}h')
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_ylim(0,1)

    plot_pacf(df_all_delta.forecast_error_relative, lags = 100,
        ax = ax2[i_subplot[i][0]][i_subplot[i][1]])
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_xlabel('lag')
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_title(f'Forecast Interval =
        {round(period_id/2)}h')
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_ylim(-0.5,1)

#fig1.suptitle('Autocorrelation')
#fig2.suptitle('Partial Autocorrelation')
plt.show()
```

Figure 4.12

```
delta = 24
```

```
df_all_delta = df_all.loc[df_all.period_id == delta].sort_values(
    "date_time_future").reset_index(drop = True)

x = df_all_delta.forecast_error_relative[delta:len(df_all_delta)]
y = df_all_delta.forecast_error_relative[0:len(df_all_delta)-delta]

plt.subplots(2,2, figsize = (18, 18))

for i, delta in enumerate([12, 24, 36, 48]):
    df_all_delta = df_all.loc[df_all.period_id == delta].sort_values(
        "date_time_future").reset_index(drop = True)
    delta_24h_later = 48 - delta
    previous_lag = 48

    x = df_all_delta.forecast_error_relative[
        previous_lag:len(df_all_delta)]
    y = df_all_delta.forecast_error_relative[
        0:len(df_all_delta)-previous_lag]

    #plt.figure(figsize = (12, 9))
    plt.subplot(2,2,i+1)
    plt.plot(np.array(x), np.array(y), '.', alpha = 0.3)
    #plt.plot(0,0, 'r.')
    plt.xlim(-0.15, 0.15)
    plt.ylim(-0.15, 0.15)
    plt.grid(alpha = 0.5)
    plt.xlabel('Relative Forecast Error at Time = t')
    plt.ylabel('Relative Forecast Error at Time = t - 24h')
```

Figure 5.1 & Figure 5.2

```
df_all["forecast_error_relative"] =
    df_all.forecast_error/df_all.total_demand

df_all["date_time_future_month"] = df_all.date_time_future.dt.month
df_all["date_time_future_year"] = df_all.date_time_future.dt.year
df_all["date_time_future_weekday"] = df_all.date_time_future.dt.dayofweek
df_all["date_time_future_yearTime"] =
  df_all.date_time_future_year.apply(
    lambda x: pd.DateOffset(years=x-2000))
df_all["date_time_future_hour"] = df_all.date_time_future.dt.hour

def check_stationarity(series):

    result = adfuller(series.values)

    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
```

```
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

    if (result[1] <= 0.05) & (result[4]['5%'] > result[0]):
        print("\u001b[32mStationary\u001b[0m")
    else:
        print("\x1b[31mNon-stationary\x1b[0m")


period_id = 24

fig1, ax1 = plt.subplots(2,2, figsize = (18, 10))
fig2, ax2 = plt.subplots(2,2, figsize = (18, 10))

i_subplot = {0: [0,0], 1: [0,1], 2: [1,0], 3: [1,1]}

for i, hour_of_day in enumerate([4, 10, 16, 22]):
    print(f"Hour of Day = {hour_of_day}")
    df_all_delta = df_all.loc[(df_all.period_id == period_id) &
      (df_all.date_time_future_hour == hour_of_day) &
      (df_all.date_time_future.dt.minute == 0)].sort_values(
         "date_time_future").reset_index(drop = True)

    check_stationarity(df_all_delta.forecast_error_relative)

    plot_acf(df_all_delta.forecast_error_relative, lags = 28,
        ax = ax1[i_subplot[i][0]][i_subplot[i][1]])
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_xlabel('lag')
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_title(
        f'Hour of Day = {hour_of_day}')
    ax1[i_subplot[i][0]][i_subplot[i][1]].set_ylim(0,1)

    plot_pacf(df_all_delta.forecast_error_relative,
      lags = 28, ax = ax2[i_subplot[i][0]][i_subplot[i][1]])
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_xlabel('lag')
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_title(
        f'Hour of Day = {hour_of_day}')
    ax2[i_subplot[i][0]][i_subplot[i][1]].set_ylim(-0.1,1)
```

Figure 5.3 & Figure 5.4

```
sarima_tune["mse_improvement"] =
    round(100*(sarima_tune.mse_pre -
    sarima_tune.mse_post)/sarima_tune.mse_pre)
sarima_tune = pd.merge(sarima_tune, sarimas,
    on = ["order", "seasonal_order"],
    how = "left").sort_values("id")

plot = sarima_tune.groupby(
```

```
    ["order", "seasonal_order", "hour_of_day"],
    as_index = False).mean()
sns.lineplot(data = plot, x = 'hour_of_day', y = 'mape',
    hue = 'id', palette = 'pastel', alpha = 1, linestyle = '--')
```

Figure 5.5

```
sample = np.random.choice(len(y_test), 100, replace=False)
x_axis = range(len(sample))

plt.figure(figsize=(14, 6))
sns.lineplot(x=x_axis, y=y_test.iloc[sample],
    label='Actual Demand', color='black')
sns.lineplot(x=x_axis, y=y_pred_original.iloc[sample],
    label='Original Forecast', linestyle='--')
sns.lineplot(x=x_axis, y=y_pred[sample],
    label='Model Predictions', linestyle='--')
plt.title("Model vs Original Forecast Performance")
plt.ylabel("Demand")
plt.show()
```

Figure 5.6

```
sample = np.random.choice(len(y_test), 100, replace=False)
x_axis = range(len(sample))

plt.figure(figsize=(14, 6))
sns.lineplot(x=x_axis, y=y_test.iloc[sample],
    label='Actual Demand', color='black')
sns.lineplot(x=x_axis, y=y_pred_original.iloc[sample],
    label='Original Forecast', linestyle='--')
sns.lineplot(x=x_axis, y=y_pred[sample],
    label='Model Predictions', linestyle='--')
plt.title("Model vs Original Forecast Performance")
plt.ylabel("Demand")
plt.show()
```

Figure 5.7

```
output_df = test_df.copy()
output_df["xgb_prediction"] = y_pred

export_cols = ["date_time_future", "total_demand",
    "forecast_demand", "xgb_prediction"]
output_df[export_cols].to_csv("finalresultsxgboost.csv", index=False)


# Calculate absolute percentage error per row
output_df["abs_pct_error"] = np.abs((output_df["total_demand"] -
    output_df["xgb_prediction"]) / output_df["total_demand"]) * 100
```

```python
# Extract hour from datetime
output_df["Hour"] = pd.to_datetime(output_df["date_time_future"]).dt.hour

# Group by hour and calculate mean error
hourly_error =
    output_df.groupby("Hour")["abs_pct_error"].mean().reset_index()

# Plot
plt.figure(figsize=(10, 5))
plt.plot(hourly_error["Hour"], hourly_error["abs_pct_error"], marker='o')
plt.title("MAPE by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("MAPE")
plt.grid(True)
plt.xticks(range(0, 24))
plt.tight_layout()
plt.show()
```

Figure 5.8

```python
lm_results = pd.read_csv("data/results_LM.csv")
lm_results["forecast_error"] = lm_results.total_demand -
    lm_results.lm_prediction

sarima_results = pd.read_csv("data/results_SARIMA.csv")
sarima_results["forecast_error"] = sarima_results.total_demand -
    sarima_results.sarima_prediction

xgboost_results = pd.read_csv("data/results_XGBoost.csv")
xgboost_results["forecast_error"] = xgboost_results.total_demand -
    xgboost_results.xgb_prediction

decisionT_results = pd.read_csv('data/results_DecisionTree.csv')
decisionT_results["forecast_error"] =
    decisionT_results.total_demand - decisionT_results.model_prediction

results_all = {"Linear": lm_results,
               "SARIMA": sarima_results,
               "XGBoost": xgboost_results,
               "Decision Tree": decisionT_results}

colors = ['#1f77b4', '#ff7f0e', 'g', '#7f7f7f']

plt.subplots(1, 2, figsize = (16,7))

plt.subplot(1,2,1)
for i, model in enumerate(results_all):
    model_result = results_all[model]
    sns.kdeplot(model_result.forecast_error, label = model,
```

```
      color = colors[i], alpha = 0.8)

sns.kdeplot(lm_results.total_demand -
    lm_results.forecast_demand, label = "AEMO", color = 'r', ls = '--')
plt.xlim(-1200, 1200);
plt.ylim(0, 0.0025)
plt.xlabel('Forecast Error')
plt.grid()
plt.legend()
plt.title("Distribution of Error");

plt.subplot(1,2,2)
for i, model in enumerate(results_all):
    model_result = results_all[model]
    sns.kdeplot(abs(model_result.forecast_error),
      label = model, color = colors[i], alpha = 0.8)

sns.kdeplot(abs(lm_results.total_demand -
    lm_results.forecast_demand), label = "AEMO", color = 'r',
    ls = '--')
plt.xlim(0, 1000);
plt.ylim(0, 0.0046)
plt.grid()
plt.legend()
plt.xlabel('abs(Forecast Error)')
plt.title("Distribution of Absolute Error");
```