Frequency Response Forecasting for Battery Storage

Group 10: Daniel Zhang, Tumisang Tsolope, Fanqiong Meng

April 2025

1 Introduction

The integration of renewable energy sources into the electricity generation mix results in numerous challenges such as intermittency and flexible energy management. This intermittency results in the need to constantly balance supply and demand on the grid, which requires flexible grid management. The battery energy storage system (BESS) is becoming increasingly crucial to stabilizing electricity grids because of its ability to rapidly respond to supply and demand fluctuations - which lead to frequency fluctuations - on the grid.

In recent years, Dynamic Frequency Response (DFR) services have been introduced in the Great Britain (GB) energy markets to maintain the frequency of the grid to within 1% of 50Hz by charging or discharging during frequency fluctuations. As a result, the prices for these DFR services become crucial for battery operators and the National Energy System Operator (NESO) to optimize the allocation of resources. This project is led by LCP Delta and the ICL Energy Society, aiming to develop a machine learning (ML) forecast for the respective day-ahead DFR service prices during different times of the day.

2 Data Preparation and Feature Engineering

2.1 Data Selection Aligned with Post-Reform Market Conditions

The datasets we have used are the day-ahead prices and the forecast and accepted volume for the DFR service we are interested in - Dynamic Containment (DC), Dynamic Moderation (DM) and Dynamic Regulation (DR), the day-ahead hourly N2EX and EPEX prices, and the day-ahead half-hourly national demand forecast and EPEX prices.

We have used these data from 2024 due to the changes in the DFR service markets. The Enduring Auction Capability (EAC) was launched in November 2023, introducing joint procurement of DC, DM and DR through a single auction and also allowing negative pricing. In the same month, the Firm Frequency Response (FFR) was discontinued. As the ancillary frequency markets may require time to adjust to these changes, the models generated in this work are based on 2024 data to reflect the current pricing landscape.

2.2 Data Cleaning

There are 18 missing values in the volume requirement forecast of DC-H and DC-L respectively, and one missing day-ahead hourly N2EX price. We used the accepted volume and the EPEX prices as proxies to fill them in, justified by high correlation and relevant plots.

2.3 Data Granularity Alignment

Since the ancillary prices and volumes dataset is with time step of 4 hours, corresponding to the EFA block operation of the energy markets, while the day-ahead N2EX and EPEX prices is a hourly dataset and the day-ahead demand forecast is half-hourly. To realign these datasets for the ML model, we explored two distinct approaches for aligning data granularity: one tailored for up-sampling, and the other for down-sampling.

For the up-sampling, these data sets were resampled to 30 minute periods by forward filling to match the

granularity of the National Demand Forecast which was used as an independent variable in the model training.

For the down-sampling, these datasets were resampled to 4-hour blocks to match the granularity of the target variable - ancillary service prices, so that it preserves the market structure and avoid introducing synthetic data. To ensure the variance of the finer datasets is not distorted, we added aggregate statistics - mean, max, min, delta=max-min, standard deviation and slope for each EFA block. These statistics could be interpreted from a market perspective if they are proven important during model training.

2.4 Timezone change in GB

Since the timezone changes in GB between summer and winter, there is a mismatch of EFA block, e.g. the 03:00 block changes to 02:00 block in UTC time in summer. In order to simplify the grouping later, we tried changing the timestamps for all the datasets to British time.

2.5 Introduce Lag features

Day-ahead accepted ancillary volumes are only known after the bids have been submitted for each day-ahead auction and therefore are classified as lagged indicators. As a result, it would not be meaningful for the ML model to incorporate them as independent variables as these would be unknown at the time of price prediction. However, the current day's accepted ancillary volumes are known as these are published by NESO after gate closure. The model presented in this work thus uses the current day's accepted ancillary volumes as an independent variable in the prediction of tomorrow's ancillary service prices. This is done by shifting the accepted ancillary volumes data of the current day by 24 hours to align with the price prediction of day-ahead ancillary service.

Besides the volume accepted at the same time the previous day, we could also add the ancillary price at the same time the previous day, and the previous day's average price and average volume accepted. Lastly, we also added the previous EFA block's price as a lag feature. But the missing data of the lag features at the start of the year needs to be cleaned before used for model training.

Categorical features like EFA block, month, workday or weekend, holiday or not could also be added to explore their impact on the model.

2.6 Feature Selection

We demonstrate the feature selection of the DC-H ancillary service here, where we used correlation matrix and permutation importance to explore the linear and non-linear relationship of the features we have with the DC-H price we wish to predict.

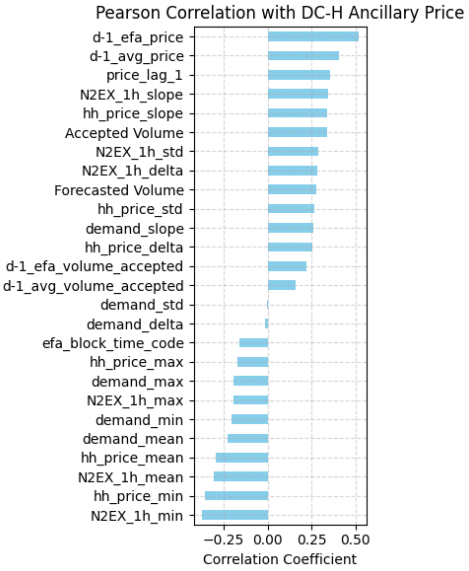


Figure 1: Correlation Matrix

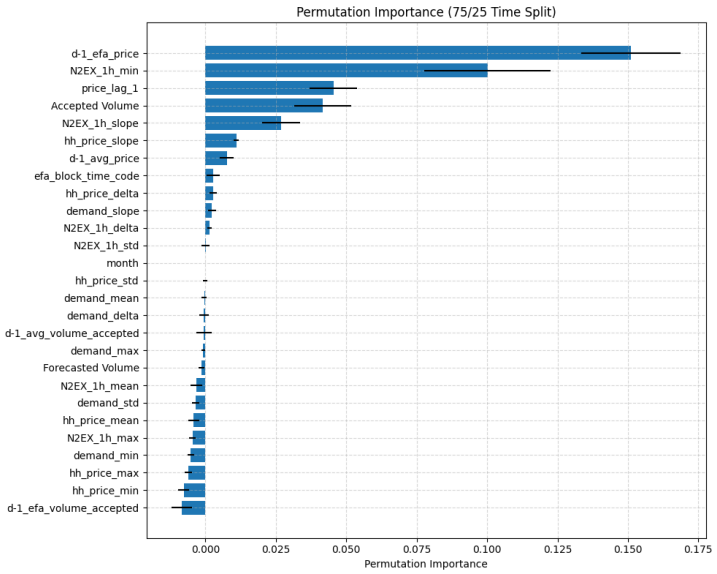
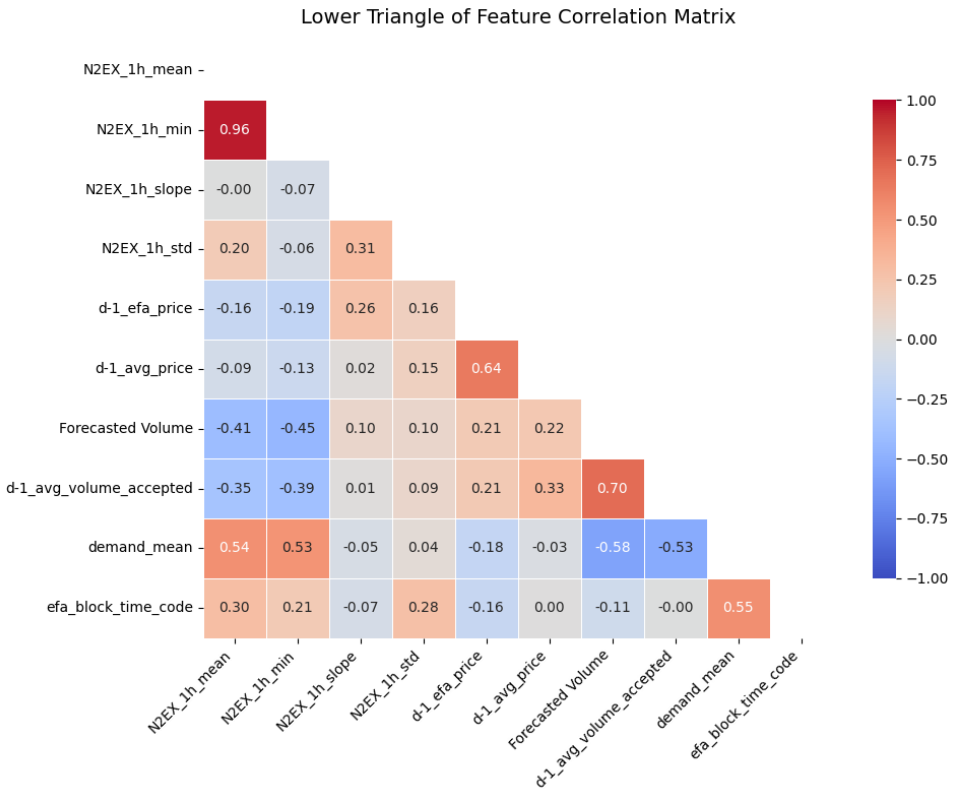


Figure 2: Permutation Importance

Based on the above plots, we select features that correlate with DC-H strongly, for example we include: the price for the same EFA block the previous day and the averaged price the previous day, the volume forecast and the average volume accepted the previous day, the N2EX hourly price's minimum, mean, slope and standard deviation in the EFA block, the demand mean and the EFA block it is in.

Now we wish to further select the features by eliminating the highly collinear features. Set a threshold for correlation and look at the heatmap of the selected features. Notice that the N2EX mean and minimum are highly correlated. Since the minimum shows high correlation in the permutation importance plot. We could consider keeping it and drop drop N2EX mean, which gives us the final set of features to feed into the model.



3 Model Selection

The initial choice of model for performing the regression analysis was based on models that can handle non-linear relationships contained within the data. This distinction is important because the combination of factors (such as electricity demand and price, generation mix, forecasted DFR service demand etc.) that determines the price of ancillary services is complex and may not be captured well by models that assume linear relationships within the data. As a result, five models were chosen to perform the initial regression analysis. These models are K-Nearest Neighbors (KNN), Decision Trees, Random Forest, AdaBoost and Gradient Boosting. The final choice of model to perform the prediction is based on the performance of the selected models on the training and testing sets. The initial training results of these models are given in Table 1 while the testing results are given in Table 2.

Model	Root mean squared error	Mean absolute error	R-squared score
K-Nearest Neighbors	756.32	10.38	0.68
Decision Tree	0.00	0.00	1.00
Random Forest	16.85	1.00	0.99
AdaBoost	1066.60	12.20	0.52
Gradient Boost	659.69	8.56	0.75

Table 1: Results of the training data set with the selected models.

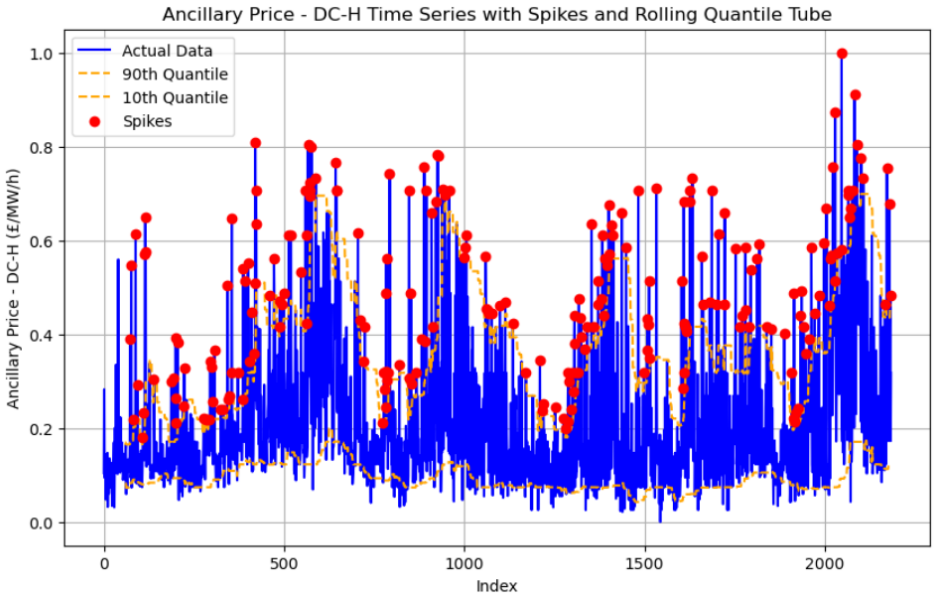
Model	Root mean squared error	Mean absolute error	R-squared score
K-Nearest Neighbors	2912.38	22.75	-0.04
Decision Tree	2416.49	17.68	0.21
Random Forest	2006.12	16.04	0.32
AdaBoost	1692.77	15.42	0.36
Gradient Boost	1685.91	14.86	0.36

Table 2: Results of the testing data set with the selected models.

According to the results in Table 2, the Gradient Boost model is the best performing model simultaneously achieving the lowest root mean squared and mean absolute error and the highest r-squared score. However, it also evident from the results that the chosen models perform significantly better on the training data set than on the testing data set which may be indicative of overfitting. Further analysis was performed using LASSO model in subsection 5.1 by analysing the spikes and the pre-spike window in the data to improve the model prediction. Unfortunately, due to time constraints, we could not provide comparable performance metrics for Spike LASSO (our own model) - so far it has only been optimised to predict DC-H price. However, Spike LASSO can easily be adapted to predict the other DFR prices, simply by changing the features and parameters. This model will be explained in the following subsection.

3.1 Spike Prediction

Spikes in the test set can be predicted by looking at "pre-spike windows" in the training set: in a relatively short time window (on the scale of days) leading up to a spike in DFR prices, it is evident from the time series plots that certain variables behave differently, acting as precursors. More precisely, if a spike occurs at time $t + \Delta t$, then we define its pre-spike window is $(t - \tau, t - \tau + 1, ..., t)$, and we say a spike occurs whenever the target variable exceeds the 90th quantile of a 60-EFA block rolling window (90 and 60 were chosen arbitrarily):



Sudden drops could be defined analogously. However, the time series plots alone do not reveal any obvious patterns shared across pre-spike windows, which suggests complex, possibly nonlinear interactions between multiple variables. This is where the signature transform, from rough path theory, is particularly useful. In recent years, signature-based ML models have been successfully applied across a variety of domains. Unlike Fourier and wavelet transforms which treat the time series of each variable independently, the signature can naturally capture multivariate information such as the order of events happening across different channels. We will now introduce the precise mathematical framework. Denote the forecast horizon by Δt (= 6 EFA blocks, or 1 day ahead), the target variable by $y = \{y_t\}_{t\in T}$ and let $x_{\text{aug}} = \{t, x_t\}_{t\in T} = \{t, x_t^1, ..., x_t^d\}_{t\in T}$ be the time series of explanatory variables other than the target variable y, where T is the set of time indices. Time is included as this improves prediction accuracy, and the underlying theory is explained in [1]. x and x_{aug} can be viewed as paths in d-dimensional and $(d+1)$-dimensional space respectively, where x_{aug} is known as the time-augmented path. The signature, which we will introduce shortly, captures the shape of the path, storing geometric information such as areas and volumes. The depth-n signature of a path x (in our case, x_{aug}) is defined as the following set of iterated integrals:

$$S^{\leq n}(x) = \big(1, S^1(x), \dots, S^n(x)\big)$$

where

$$S^k(x)_{[a,b]} = \int_{a\leq t_1<\cdots<t_k\leq b} dx_{t_1} \otimes \cdots \otimes dx_{t_k} \quad \text{for } 1 \leq k \leq n,$$

and $\otimes$ denotes the tensor product. The first level $S^1(x)$ is a vector where the i^{th} element is the net change, or displacement, in variable x_i over the time interval [a, b], while the k^{th} level $S^k(x)$ for $k \geq 2$ is a k^{th}-order tensor (for example, $S^2(x)$ is a matrix while $S^3(x)$ is a cube), capturing interactions between all possible combinations of k variables. In practice, the signature is flattened into a vector, and the depth needs to optimised. For this forecasting problem, we have chosen to use depth 3. A key property of the signature, known as universal nonlinearity, is that regression on time series becomes linear after applying the signature transform. In particular, we can rewrite the spike forecasting problem

$$y_{t+\Delta t} = f(y_{t-\tau:t}, x_{t-\tau:t}) + \text{noise}$$

as a linear combination of signature coefficients:

$$y_{t+\Delta t} = S(y_{t-\tau:t}, x_{t-\tau:t}) \cdot \theta + \text{noise}$$

This leads to the following optimisation problem:

$$\min_\theta ||Y - S\theta||_2^2$$

where $Y = (y_{t+\Delta t})_{t\in T_{\text{train}}}$ and $S = [S((y, x)_{t-\tau:t})]_{t\in T_{\text{train}}}$. We will optimise θ in two stages. The first stage is LASSO regularisation, as we wish to remove redundant variables and avoid overfitting. However,

LASSO can introduce bias by shrinking coefficients too aggressively. In the second stage, the LASSO estimate θ^* is re-estimated through ordinary least squares, subject to the constraint that the entries of θ^* shrunk to zero remain zero, ensuring sparsity. This is a convex optimisation problem, so the new parameter estimate $\hat{\theta}$ can be found via projected gradient descent. Our spike predictions can be formulated as follows:

$$\hat{y}^{\text{SPIKE}}_{t+\Delta t} = S((y,x)_{t-\tau:t}) \cdot \hat{\theta}$$

The weights $w(t-\tau:t, s-\tau:s)$ are then computed from the formula below, where γ controls the model's sensitivity to dissimilarity between the current test window and pre-spike training windows. Note that here, we just use x rather than x_{aug} because we found that when time is included as an additional channel, the distance becomes distorted.

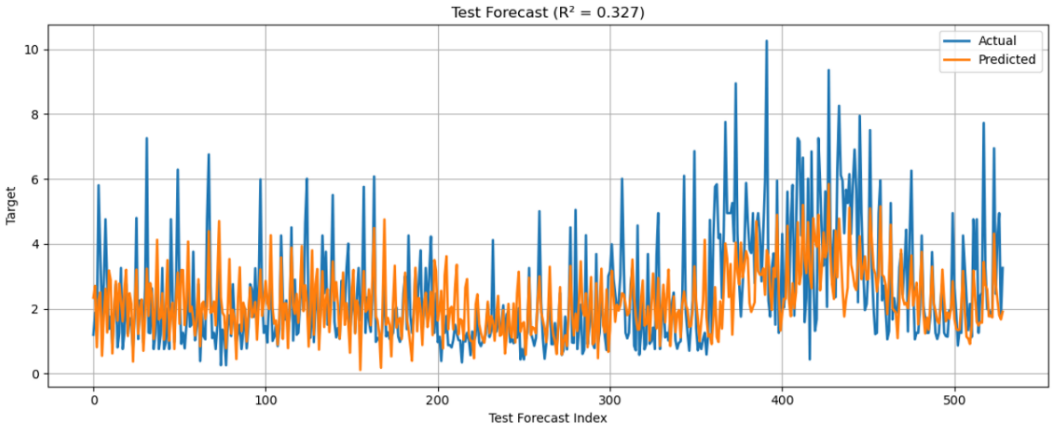
$$w(t-\tau:t, s-\tau:s) = \frac{\exp(-\gamma d(t,s))}{\sum_{s \in T_{\text{train}} \cap T_{\text{pre-spike}}} \exp(-\gamma d(t,s))}$$

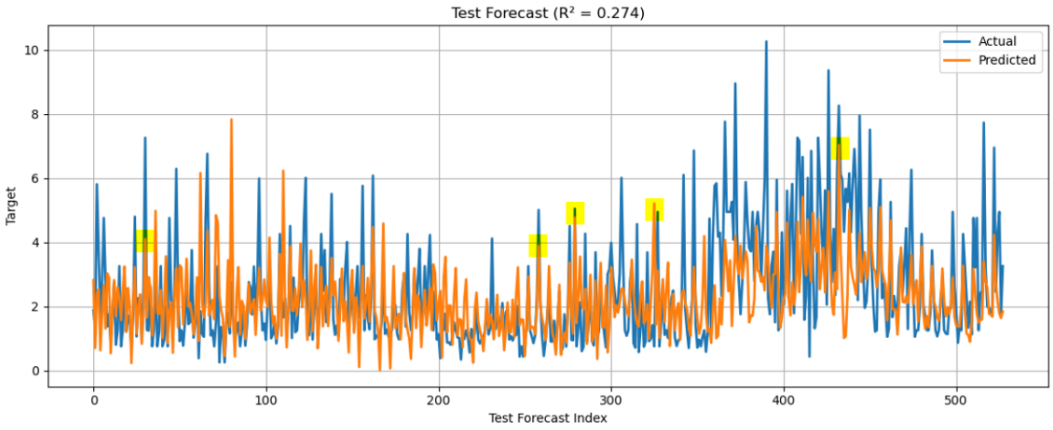
The distance $d(t,s)$ is defined as $||S(z)_{t-\tau:t} - S(z)_{s-\tau:s}||_2$, where $z = (z_1, ..., z_p)$ are the p variables most closely linked to spike occurrence. If $M \in (0,1)$ represents the similarity threshold, then

$$\hat{y}_{t+\Delta t} = \begin{cases} \hat{y}^{\text{SPIKE}}_{t+\Delta t} & \text{if } |\{s: \ w(t,s) > M\}| \geq 1, \\ \hat{y}^{\text{LASSO}}_{t+\Delta t} & \text{otherwise.} \end{cases}$$

In other words, if there is at least one pre-spike window in the training set above a certain similarity level to the current test window, then use the spike model, as the current test window is also likely to precede a spike. Otherwise, proceed with the LASSO model.

Setting the similarity threshold to a value above 1 reduces the model to regular LASSO. Despite achieving a lower R^2 on the test set than standard LASSO, Spike LASSO predicts the occurrence and height of spikes more accurately, as highlighted in the second plot below (the first plot is standard LASSO):





This is one reason why R^2 should not be the only metric used to measure model accuracy. One limitation of LASSO-Spike is that it can predict false spikes, as shown above. However, this could also be considered a strength because some false spikes are predicted just before an actual spike, serving as a warning that the price may actually spike in the near future.

The two-stage optimisation and similarity weights are inspired by a novel algorithm that has been integrated into one of Amazon's forecasting models since November 2022 [2], which is also based on the universal nonlinearity of the signature transform.

4 Observations

4.1 Spikes

The largest DC-L price spike (70.69) happens when Generation - Fuel - OCGT is unusually high. This makes sense because gas has a higher marginal cost of production (MCP) compared to wind and nuclear energy, and OCGT has higher operating costs than CCGT as OCGT is simpler in design. We can see that the wind generated drops significantly before the price spike, so on a graph of price vs.quantity produced, the MCP curve (essentially the supply curve) would shift left: wind and nuclear energy are no longer sufficient to meet the demand, leading to more expensive sources like OCGT being used. As a result, equilibrium price rises and equilibrium quantity drops, which could explain the price surge and the dramatic reduction in volume accepted.

We also note that, while the day-ahead price movements do not always align with the ancillary price dynamics, it seems to be the main explanatory variable when the day-ahead price is extreme (roughly 200-300).

4.2 Collinearity

When initially selecting the model, the collinearity limit between the predictor variables was set to 0.65. This led to the results indicated in Table 1 and Table 2. However, it is observed that increasing this limit to 0.85 improves the model prediction as shown in tables 3 and 4. Increasing this limit means that the model drops predictor variables with a collinearity that is higher than 0.85 compared to a collinearity of 0.65 initially. This enables the model to capture more information between the predictor variables and thus make improved predictions. However, it is important to note that while increasing the limit of collinearity to 0.85 improves the prediction, this value is often a result of extensive domain expertise. Without much knowledge on the industry standard of this value, this work settled on a value of 0.85.

Model	Root mean squared error	Mean absolute error	R-squared score
K-Nearest Neighbors	138.23	5.10	0.72
Decision Tree	0.00	0.00	1.00
Random Forest	0.01	0.03	1.00
AdaBoost	21.02	2.48	0.63
Gradient Boost	1.1277	0.5605	0.81

Table 3: Results of the training data set with the selected models.

Model	Root mean squared error	Mean absolute error	R-squared score
K-Nearest Neighbors	1241.62	15.99	0.05
Decision Tree	186.26	2.60	0.37
Random Forest	159.48	2.36	0.48
AdaBoost	187.16	4.10	0.49
Gradient Boost	158.94	2.42	0.50

Table 4: Results of the testing data set with the selected models.

5 Conclusion

The work presented achieves an ancillary service price prediction with an R^2 value of 0.50 using Gradient Boost regression and with further analysis using LASSO, achieves an R^2 value of 0.56. Most importantly, it is noted from this work that accurate predictions are a result of capturing complex relationships in energy market data. Another key finding is that simpler, more interpretable models like Spike LASSO can achieve similar performance to or even outperform more advanced ones such as Random Forests. Finally, R^2 alone does not provide a complete picture of model accuracy.

6 Future Improvement

6.1 Additional data

Historical data: More data could be included to improve the model if it is proven useful, including data from previous years (this could be set as a categorical feature to explore the impact of EAC and other changes on the pricing model) and additional data such as the rolling system frequency (the continuously updated measurement of the electrical frequency of the grid over time).

Adaptive Training: As new data becomes available over time, the model should improve by retraining on both the existing training data and the real data from the day after each forecast.

Dip-LASSO: It may be possible to predict sudden drops in price, or downward spikes, in an analogous way to the Spike-LASSO model.

6.2 Seasonality

Fluctuations in the generation mix caused by seasonality and changes in weather patterns could be included in the model to improve its accuracy. Modelling by seasonality captures and leverages the repeating patterns that occur at regular time intervals. These patterns can provide powerful insights that can be used to optimise the model.

7 Appendix: Extra Plots

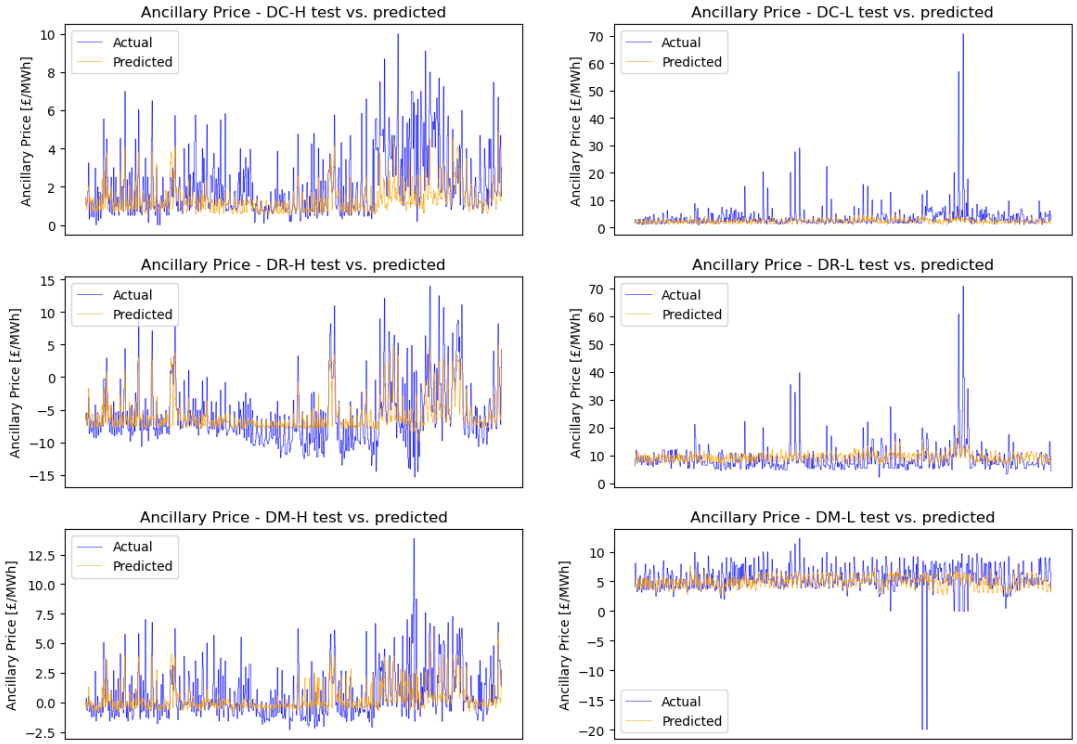


Figure 3: Actual ancillary prices vs predicted ancillary prices for obtained using Gradient Boost regressor.

Citations

[1] Cristopher Salvi and Thomas Cass. "Lecture notes on Rough Paths and Applications to Machine Learning". In: *arXiv* (2024). URL: https://arxiv.org/pdf/2404.06583.

[2] Haotian Gu et al. "Transportation Marketplace Rate Forecast Using Signature Transform". In: *arXiv* (2024). URL: https://arxiv.org/pdf/2401.04857.

Data_Preparation

April 7, 2025

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import linregress
import seaborn as sns
import numpy as np
```

[2]:
```
# Load the dataset

ancillary_2024 = pd.read_csv("/content/Ancillary Volumes & Prices (4H).csv")
day_ahead_prices_2024 = pd.read_csv("/content/Day-Ahead Price (1H).csv")
wholesale_prices_and_volumes_2024 = pd.read_csv("/content/Prices & Forecasts␣
 ↪(HH).csv")

# Convert Time column to datetime
ancillary_2024["GMT Time"] = pd.to_datetime(ancillary_2024["GMT Time"],␣
 ↪format="%d/%m/%Y %H:%M").dt.tz_localize("GMT")
ancillary_2024.set_index("GMT Time", inplace=True)
day_ahead_prices_2024["GMT Time"] = pd.to_datetime(day_ahead_prices_2024["GMT␣
 ↪Time"], format="%d/%m/%Y %H:%M").dt.tz_localize("GMT")
wholesale_prices_and_volumes_2024["GMT Time"] = pd.
 ↪to_datetime(wholesale_prices_and_volumes_2024["GMT Time"], format="%d/%m/%Y␣
 ↪%H:%M").dt.tz_localize("GMT")
```

1 Data Preparation

1.1 Handle missing values in all three data files

[3]:
```
for name, df in {
    'Ancillary Prices': ancillary_2024,
    'Day Ahead Prices': day_ahead_prices_2024,
    'Wholesale demand': wholesale_prices_and_volumes_2024
}.items():
    missing = df.isna().sum()
    missing = missing[missing > 0]
    if not missing.empty:
        print(f"\nMissing values in {name}:")
        for col, count in missing.items():
```

```
            print(f"{col}: {count}")
```

```
Missing values in Ancillary Prices:
Volume Requirements Forecast - DC-H - GB (MW): 18
Volume Requirements Forecast - DC-L - GB (MW): 18

Missing values in Day Ahead Prices:
Day Ahead Price (N2EX, local) - GB (£/MWh): 1
```

[4]:
```
# Make deep copies of the original data to make modifications
ancillary_modified = ancillary_2024.copy(deep=True)

hourly_prices_modified = day_ahead_prices_2024.copy(deep=True)
hourly_prices_modified.columns = ["Time", "N2EX", "EPEX"]

hh_modified = wholesale_prices_and_volumes_2024.copy(deep=True)
hh_modified.columns = ["Time", "Demand", "HH_Price"]
```

[5]:
```
# For the missing volume requirements forcast in the ancillary data, use␣
 ↪accepted volume as proxy*
ancillary_modified["Volume Requirements Forecast - DC-H - GB (MW)"] =␣
 ↪ancillary_modified["Volume Requirements Forecast - DC-H - GB (MW)"].
 ↪fillna(ancillary_modified["Ancillary Volume Accepted - DC-H - GB (MW)"])
ancillary_modified["Volume Requirements Forecast - DC-L - GB (MW)"] =␣
 ↪ancillary_modified["Volume Requirements Forecast - DC-L - GB (MW)"].
 ↪fillna(ancillary_modified["Ancillary Volume Accepted - DC-L - GB (MW)"])

# Check how many missing values remain after filling
missing_values_a = ancillary_modified.isna().sum()
print(missing_values_a[missing_values_a > 0])
```

```
Series([], dtype: int64)
```

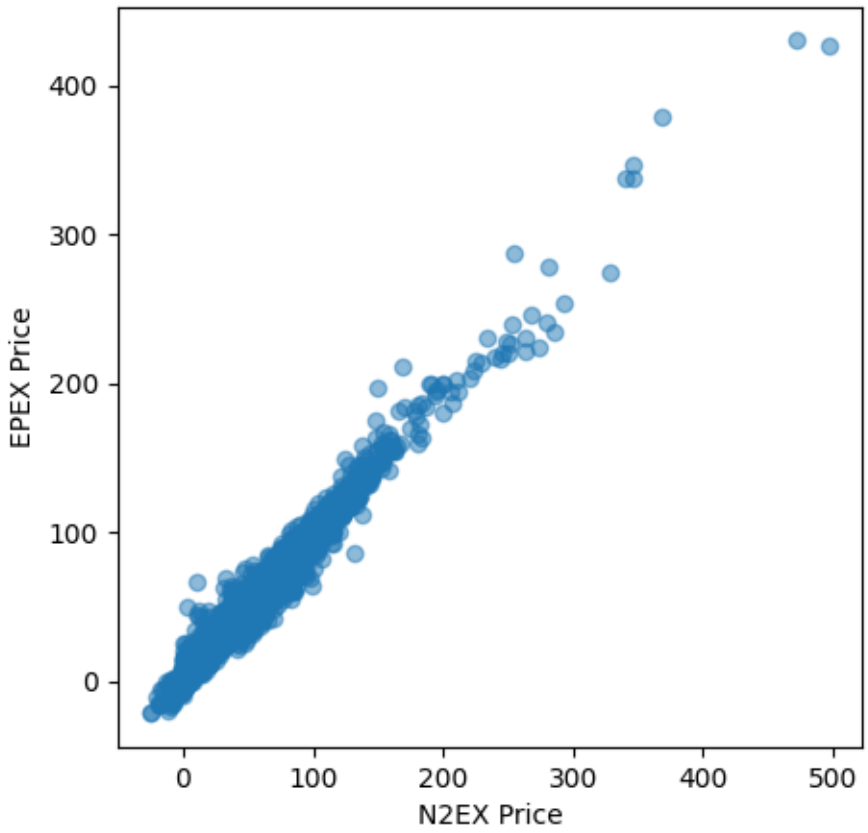
[6]:
```
# Calculate the correlation between N2EX and EPEX prices.
# A strong correlation indicates they could be used as replacements for missing␣
 ↪values.
between_hourly_correlation = hourly_prices_modified["N2EX"].
 ↪corr(hourly_prices_modified["EPEX"])
print(f"Correlation between N2EX and EPEX prices: {between_hourly_correlation:.
 ↪3f}")
```

```
Correlation between N2EX and EPEX prices: 0.984
```

[7]:
```
# Plot hourly N2EX price against EPEX price
plt.figure(figsize=(5, 5))
```

```
plt.scatter(hourly_prices_modified["N2EX"], hourly_prices_modified["EPEX"],␣
 ↪alpha=0.5)
plt.xlabel("N2EX Price")
plt.ylabel("EPEX Price")
```

[7]: Text(0, 0.5, 'EPEX Price')



[8]:
```
# Fill missing values using EPEX price as proxy
hourly_prices_modified["N2EX"] = hourly_prices_modified["N2EX"].
 ↪fillna(hourly_prices_modified["EPEX"])

# Check how many missing values in the day_ahead houyrly prices after filling
missing_values_b = hourly_prices_modified.isna().sum()
print(missing_values_b[missing_values_b > 0])
```

```
Series([], dtype: int64)
```

1.2 Split the ancillary data into separate df's corresponing to different services

[9]:
```
# Define service labels
services = ["DC-H", "DC-L", "DM-H", "DM-L", "DR-H", "DR-L"]

# Define base column patterns to extract
col_patterns = {
    "price": "Ancillary Price - {} - GB (£/MW/h)",
    "fc_volume": "Volume Requirements Forecast - {} - GB (MW)",
    "accepted_volume": "Ancillary Volume Accepted - {} - GB (MW)"
}

# Create the 6 DataFrames
service_dfs = {}

for service in services:
    # Define column names for this service
    cols = {
        "Ancillary Price": col_patterns["price"].format(service),
        "Forecasted Volume": col_patterns["fc_volume"].format(service),
        "Accepted Volume": col_patterns["accepted_volume"].format(service),
    }

    # Build the DataFrame
    service_df = ancillary_modified[list(cols.values())].copy()
    service_df.columns = list(cols.keys())  # Rename columns

    # Add to dictionary
    service_dfs[service] = service_df
```

[10]:
```
# Example: access DC-H dataframe
dc_h_df = service_dfs["DC-H"]
print(dc_h_df.head())
```

```
                           Ancillary Price  Forecasted Volume  Accepted Volume
GMT Time
2024-01-01 03:00:00+00:00             5.00             1250.0             1250
2024-01-01 07:00:00+00:00             2.31             1250.0             1250
2024-01-01 11:00:00+00:00             1.25             1250.0             1160
2024-01-01 15:00:00+00:00             2.21             1137.0              997
2024-01-01 19:00:00+00:00             1.10             1007.0             1087
```

1.3 Change the timestamps of ancillary prices to British time.

For the ancillary price data file, this means the EFA blocks' time are invariant under the change of time zone in the UK, which helps us to group them later.

[11]:
```
# For each service's df, set the index to be the British timestamps
for service in service_dfs:
  service_dfs[service].index = service_dfs[service].index.tz_convert("Europe/
 ↪London").tz_localize(None)
  service_dfs[service].index.name = "British_Time"
```

[12]:
```
# Check that on March 31st, the EFA block at GMT 3:00 is not shown as 2:00, so␣
 ↪that we can group all the 3:00 blocks together later
print(service_dfs['DC-H'].index[538:544])
```

```
DatetimeIndex(['2024-03-30 19:00:00', '2024-03-30 23:00:00',
               '2024-03-31 03:00:00', '2024-03-31 07:00:00',
               '2024-03-31 11:00:00', '2024-03-31 15:00:00'],
              dtype='datetime64[ns]', name='British_Time', freq=None)
```

1.4 Add lag features - the price in the same EFA block the day before, and the average price in the previous day; And add month and EFA block as categorical features.

[13]:
```
# Add the price of the previous EFA block price_lag_1
for service in service_dfs:
  service_dfs[service]["price_lag_1"] = service_dfs[service]["Ancillary Price"].
 ↪shift(1)
```

[14]:
```
# Add the previous day's average price and the same EFA block's price; Add␣
 ↪categorical features - month and EFA_block
for service in service_dfs:
    df = service_dfs[service]

    # Separate date and time parts
    df['Date'] = df.index.date
    df['EFA_block'] = df.index.time  # EFA block as time of day

    # Market-aligned trading day
    df['Market_Day'] = (df.index + pd.Timedelta(hours=1)).date

    daily_avg = df.groupby('Market_Day')['Ancillary Price'].mean().round(3) #␣
 ↪Compute daily average price per Market_Day
    d1_avg_shifted = daily_avg.shift(1)  # Shift that series to align with the␣
 ↪NEXT day
    df['d-1_avg_price'] = df['Market_Day'].map(d1_avg_shifted) # Map the␣
 ↪shifted daily averages back to each row using their Market_Day

    # Add previous day's averaged volume accepted
    daily_volume = df.groupby('Market_Day')['Accepted Volume'].mean().round(3)
    d1_volume_shifted = daily_volume.shift(1)
    df['d-1_avg_volume_accepted'] = df['Market_Day'].map(d1_volume_shifted)
```

```
    # Price of the same EFA block on the previous day
    df['d-1_efa_price'] = df.groupby('EFA_block')['Ancillary Price'].shift(1)
    df['d-1_efa_volume_accepted'] = df.groupby('EFA_block')['Accepted Volume'].
↪shift(1)

    # Categorical features
    df['month'] = df.index.month.astype('category')          # categorical month
    # df['EFA_block'] = df['EFA_block'].astype('category')    # keep EFA block␣
↪as categorical
    df['EFA_block'] = df.index.time.astype(str)          # convert time to␣
↪string if needed*
    efa_block_order = {
        '23:00:00': 0,
        '03:00:00': 1,
        '07:00:00': 2,
        '11:00:00': 3,
        '15:00:00': 4,
        '19:00:00': 5
    }
    df["efa_block_time_code"] = df["EFA_block"].map(efa_block_order)


    # Drop only temporary column
    df.drop(columns=['Date', 'Market_Day','EFA_block'], inplace=True)

    # Save changes back
    service_dfs[service] = df
```

1.5 Match time steps by finding mean, max, min, delta=max-min, std, slope in each EFA block for the hourly N2EX price

[15]:
```
# Drop the first 3 rows and the last row, so it starts at 03:00 1st Jan and␣
↪ends at 23:00 31st Dec, match anxillary price data
hourly_prices_modified = hourly_prices_modified.iloc[3:-1].copy()
```

[16]:
```
print(hourly_prices_modified.head(3))
print(hourly_prices_modified.tail(3))
```

```
                       Time  N2EX  EPEX
3 2024-01-01 03:00:00+00:00  0.15  25.8
4 2024-01-01 04:00:00+00:00 -0.29  20.6
5 2024-01-01 05:00:00+00:00 -0.20  15.3
                          Time   N2EX  EPEX
8780 2024-12-31 20:00:00+00:00  40.00  36.0
8781 2024-12-31 21:00:00+00:00  27.71  19.9
8782 2024-12-31 22:00:00+00:00   2.06   7.0
```

[17]:
```
# Function to assign each row to its EFA block start time
def get_efa_block_start(time):
  # very important to have the following line to get rid of the time zone
  time = time.tz_localize(None)

  if time.hour < 3:
    # Assign to 23:00 of previous day
    return (time - pd.Timedelta(days=1)).replace(hour=23, minute=0, second=0,␣
 ↪microsecond=0)
  else:
    # Align to nearest EFA block start (fixed: 03, 07, 11, 15, 19, 23)
    block_start_hour = ((time.hour+1)//4) * 4  - 1
    return time.replace(hour=block_start_hour, minute=0, second=0,␣
 ↪microsecond=0)
```

[18]:
```
# Create a new column with British timestamps to align EFA blocks
# have defined columns = ["Time", "N2EX", "EPEX"]
hourly_prices_modified["British_Time"] = hourly_prices_modified["Time"].dt.
 ↪tz_convert("Europe/London")
```

[19]:
```
# Apply EFA block mapping
hourly_prices_modified["EFA_block"] = hourly_prices_modified["British_Time"].
 ↪apply(get_efa_block_start)

# Group by EFA block and calculate required features on N2EX(EPEX correlates␣
 ↪strongly with it so use only N2EX for now)
hourly_N2EX_efa = hourly_prices_modified.groupby("EFA_block")["N2EX"].agg(
    N2EX_1h_mean="mean",
    N2EX_1h_max="max",
    N2EX_1h_min="min",
    N2EX_1h_std="std",
    # count="count"  # track number of hours per block to verify DST behaviour
)

# Calculate price delta
hourly_N2EX_efa["N2EX_1h_delta"] = hourly_N2EX_efa["N2EX_1h_max"] -␣
 ↪hourly_N2EX_efa["N2EX_1h_min"]

# Compute slope per EFA block
def compute_slope(group):
    x = range(len(group))
    y = group.values  # since group is already a Series of N2EX values
    if len(y) < 2:
        return 0.0  # avoid linregress errors on too few points
    slope, _, _, _, _ = linregress(x, y)
    return slope
```

```
slope_series = hourly_prices_modified.groupby("EFA_block")["N2EX"].
 ↪apply(compute_slope)
hourly_N2EX_efa["N2EX_1h_slope"] = slope_series

# Round all values to 2 decimal places
hourly_N2EX_efa = hourly_N2EX_efa.round(2)

# Save to file
hourly_N2EX_efa.to_csv("efa_block_hourly_prices_N2EX.csv")
```

[20]:
```
# Check where the timezone changes*
# print(hourly_N2EX_efa.head(5))
```

1.6 Match time steps by finding mean, max, min, delta=max-min and slope in each EFA block for the forcasted half-hourly power demand

[21]:
```
# Drop the first 5 rows and the last two rows, so it starts at 03:00 1st Jan␣
 ↪and ends at 23:00 31st Dec.
hh_modified = hh_modified.iloc[6:-2].copy()
```

[22]:
```
print(hh_modified.head(3))
print(hh_modified.tail(3))
```

```
                       Time  Demand  HH_Price
6 2024-01-01 03:00:00+00:00   19638      24.8
7 2024-01-01 03:30:00+00:00   18884      11.2
8 2024-01-01 04:00:00+00:00   18296      11.0
                           Time  Demand  HH_Price
17563 2024-12-31 21:30:00+00:00   24629     15.00
17564 2024-12-31 22:00:00+00:00   23482     10.29
17565 2024-12-31 22:30:00+00:00   22625    -20.00
```

[23]:
```
# Create a new column with British timestamps to align EFA blocks
# have defined hh_modified.columns = ["Time", "Demand", "HH_Price"]
hh_modified["British_Time"] = hh_modified["Time"].dt.tz_convert("Europe/London")
```

[24]:
```
# Check the time zone change days
# print(hh_modified[4312:4318])
# print(hh_modified[14392:14398])
```

[25]:
```
# Apply the same EFA block mapping
hh_modified["EFA_block"] = hh_modified["British_Time"].
 ↪apply(get_efa_block_start)
```

[26]:
```
# Check the time zone change days
# print(hh_modified[4312:4318])
```

```
# print(hh_modified[14392:14398])
```

[27]:
```
def summarise_efa_block(group):
    result = {}

    for col in ["Demand", "HH_Price"]:
        x = range(len(group))
        y = group[col].values
        slope, _, _, _, _ = linregress(x, y)

        prefix = col.lower()  # e.g. "demand", "hh_price"
        result.update({
            f"{prefix}_mean": y.mean(),
            f"{prefix}_max": y.max(),
            f"{prefix}_min": y.min(),
            f"{prefix}_delta": y.max() - y.min(),
            f"{prefix}_slope": slope,
            f"{prefix}_std": y.std()
            # "count": len(y),
        })

    return pd.Series(result)
```

[28]:
```
# Filter rows where count is not 8 and check the DST transitions are correct
# invalid_counts = hh_demand_efa[(hh_demand_efa["count"] != 8)]
# print(invalid_counts)
```

[29]:
```
# Group by EFA block and apply the custom function
hh_efa = hh_modified.drop(columns="EFA_block").
 ↪groupby(hh_modified["EFA_block"]).apply(summarise_efa_block)

# Round if needed
hh_efa = hh_efa.round(2)

# Save to CSV
hh_efa.to_csv("hh_efa.csv")
```

[30]:
```
# hh_efa.head(5)
```

1.7 Ensure all the data have the same length

[31]:
```
# service_dfs['DC-H'].tail(3)
```

[32]:
```
# Delete the last row (23:00 efa block) of anxillary data file to match the␣
 ↪length
for service in service_dfs:
```

```
    service_dfs[service] = service_dfs[service].iloc[:-1]

# Check the length for the modified data files
for df in [service_dfs['DC-H'], hourly_N2EX_efa, hh_efa]:
    print(len(df))
```

```
2195
2195
2195
```

1.8 Merge the data for each service and drop the rows with missing value

[33]:
```
# Create a dict to hold merged versions
merged_service_dfs = {}

# Loop through each service
for service, df in service_dfs.items():
    # Concatenate side-by-side along columns (axis=1)
    merged = pd.concat([df, hourly_N2EX_efa, hh_efa], axis=1)
    merged_service_dfs[service] = merged
```

[34]:
```
merged_service_dfs['DC-H'].columns
```

[34]:
```
Index(['Ancillary Price', 'Forecasted Volume', 'Accepted Volume',
       'price_lag_1', 'd-1_avg_price', 'd-1_avg_volume_accepted',
       'd-1_efa_price', 'd-1_efa_volume_accepted', 'month',
       'efa_block_time_code', 'N2EX_1h_mean', 'N2EX_1h_max', 'N2EX_1h_min',
       'N2EX_1h_std', 'N2EX_1h_delta', 'N2EX_1h_slope', 'demand_mean',
       'demand_max', 'demand_min', 'demand_delta', 'demand_slope',
       'demand_std', 'hh_price_mean', 'hh_price_max', 'hh_price_min',
       'hh_price_delta', 'hh_price_slope', 'hh_price_std'],
      dtype='object')
```

[35]:
```
# Check for missing values in the merged df, take DC-H as the typical case

df = merged_service_dfs['DC-H']

# Step 1: Define target and features*
target_col = "Ancillary Price"

# X = all columns except the target
X = df.drop(columns=[target_col])

# y = target column
y = df[target_col]

# Inspect missing values
```

```python
missing_X = X[X.isnull().any(axis=1)]
print("Missing values in X:")
print(missing_X)

missing_y = y[y.isnull()]
print("Missing values in y:")
print(missing_y)

# Combine to see full picture (if any row has missing in X or y)
missing_rows = X.index[X.isnull().any(axis=1) | y.isnull()]
print("Rows with missing in X or y:")
print(X.loc[missing_rows])
```

```
Missing values in X:
                     Forecasted Volume  Accepted Volume  price_lag_1  \
2024-01-01 03:00:00             1250.0             1250          NaN
2024-01-01 07:00:00             1250.0             1250         5.00
2024-01-01 11:00:00             1250.0             1160         2.31
2024-01-01 15:00:00             1137.0              997         1.25
2024-01-01 19:00:00             1007.0             1087         2.21
2024-01-01 23:00:00             1051.0             1250         1.10

                     d-1_avg_price  d-1_avg_volume_accepted  d-1_efa_price  \
2024-01-01 03:00:00            NaN                      NaN            NaN
2024-01-01 07:00:00            NaN                      NaN            NaN
2024-01-01 11:00:00            NaN                      NaN            NaN
2024-01-01 15:00:00            NaN                      NaN            NaN
2024-01-01 19:00:00            NaN                      NaN            NaN
2024-01-01 23:00:00          2.374                   1148.8            NaN

                     d-1_efa_volume_accepted month  efa_block_time_code  \
2024-01-01 03:00:00                      NaN     1                    1
2024-01-01 07:00:00                      NaN     1                    2
2024-01-01 11:00:00                      NaN     1                    3
2024-01-01 15:00:00                      NaN     1                    4
2024-01-01 19:00:00                      NaN     1                    5
2024-01-01 23:00:00                      NaN     1                    0

                     N2EX_1h_mean  …  demand_min  demand_delta  \
2024-01-01 03:00:00          0.20  …     18060.0        1578.0
2024-01-01 07:00:00         28.82  …     20002.0        4518.0
2024-01-01 11:00:00         57.22  …     25351.0        4154.0
2024-01-01 15:00:00         92.12  …     29994.0        3806.0
2024-01-01 19:00:00         48.13  …     23100.0        7643.0
2024-01-01 23:00:00          8.92  …     19435.0        2374.0

                     demand_slope  demand_std  hh_price_mean  hh_price_max  \
```

```
2024-01-01 03:00:00        -30.24      548.73           7.05          24.8
2024-01-01 07:00:00        692.65     1613.58          44.72          68.0
2024-01-01 11:00:00        582.95     1386.61          69.66          80.0
2024-01-01 15:00:00        313.68     1229.05          81.53          96.0
2024-01-01 19:00:00      -1052.33     2417.37          51.21          72.0
2024-01-01 23:00:00       -274.26      705.41           5.91          19.6

                     hh_price_min  hh_price_delta  hh_price_slope  \
2024-01-01 03:00:00         -11.6            36.4           -2.02
2024-01-01 07:00:00          25.0            43.0            5.44
2024-01-01 11:00:00          55.0            25.0            1.16
2024-01-01 15:00:00          60.0            36.0            0.64
2024-01-01 19:00:00           9.6            62.4           -5.03
2024-01-01 23:00:00          -9.2            28.8           -0.67

                     hh_price_std
2024-01-01 03:00:00          9.88
2024-01-01 07:00:00         14.88
2024-01-01 11:00:00          7.38
2024-01-01 15:00:00         13.05
2024-01-01 19:00:00         18.44
2024-01-01 23:00:00          9.27

[6 rows x 27 columns]
Missing values in y:
Series([], Name: Ancillary Price, dtype: float64)
Rows with missing in X or y:
                     Forecasted Volume  Accepted Volume  price_lag_1  \
2024-01-01 03:00:00             1250.0             1250          NaN
2024-01-01 07:00:00             1250.0             1250         5.00
2024-01-01 11:00:00             1250.0             1160         2.31
2024-01-01 15:00:00             1137.0              997         1.25
2024-01-01 19:00:00             1007.0             1087         2.21
2024-01-01 23:00:00             1051.0             1250         1.10

                     d-1_avg_price  d-1_avg_volume_accepted  d-1_efa_price  \
2024-01-01 03:00:00            NaN                      NaN            NaN
2024-01-01 07:00:00            NaN                      NaN            NaN
2024-01-01 11:00:00            NaN                      NaN            NaN
2024-01-01 15:00:00            NaN                      NaN            NaN
2024-01-01 19:00:00            NaN                      NaN            NaN
2024-01-01 23:00:00          2.374                   1148.8            NaN

                     d-1_efa_volume_accepted  month  efa_block_time_code  \
2024-01-01 03:00:00                      NaN      1                    1
2024-01-01 07:00:00                      NaN      1                    2
2024-01-01 11:00:00                      NaN      1                    3
2024-01-01 15:00:00                      NaN      1                    4
```

```
2024-01-01 19:00:00                    NaN     1                    5
2024-01-01 23:00:00                    NaN     1                    0

                     N2EX_1h_mean  …  demand_min  demand_delta  \
2024-01-01 03:00:00          0.20  …     18060.0        1578.0
2024-01-01 07:00:00         28.82  …     20002.0        4518.0
2024-01-01 11:00:00         57.22  …     25351.0        4154.0
2024-01-01 15:00:00         92.12  …     29994.0        3806.0
2024-01-01 19:00:00         48.13  …     23100.0        7643.0
2024-01-01 23:00:00          8.92  …     19435.0        2374.0

                     demand_slope  demand_std  hh_price_mean  hh_price_max  \
2024-01-01 03:00:00        -30.24      548.73           7.05          24.8
2024-01-01 07:00:00        692.65     1613.58          44.72          68.0
2024-01-01 11:00:00        582.95     1386.61          69.66          80.0
2024-01-01 15:00:00        313.68     1229.05          81.53          96.0
2024-01-01 19:00:00      -1052.33     2417.37          51.21          72.0
2024-01-01 23:00:00       -274.26      705.41           5.91          19.6

                     hh_price_min  hh_price_delta  hh_price_slope  \
2024-01-01 03:00:00         -11.6            36.4           -2.02
2024-01-01 07:00:00          25.0            43.0            5.44
2024-01-01 11:00:00          55.0            25.0            1.16
2024-01-01 15:00:00          60.0            36.0            0.64
2024-01-01 19:00:00           9.6            62.4           -5.03
2024-01-01 23:00:00          -9.2            28.8           -0.67

                     hh_price_std
2024-01-01 03:00:00          9.88
2024-01-01 07:00:00         14.88
2024-01-01 11:00:00          7.38
2024-01-01 15:00:00         13.05
2024-01-01 19:00:00         18.44
2024-01-01 23:00:00          9.27

[6 rows x 27 columns]
```

[36]:
```python
# Drop rows where X or y has missing values for all the services' merged df
for service, df in merged_service_dfs.items():

    X = df.drop(columns=[target_col])
    y = df[target_col]

    # Drop rows with missing values in either X or y
    mask = X.notna().all(axis=1) & y.notna()
    X = X[mask]
    y = y[mask]
```

```
    # Reassemble cleaned df
    merged_service_dfs[service] = pd.concat([X, y], axis=1)

    # Print True if there are no missing values in either X or y
    print((X.isna().sum().sum() + y.isna().sum()) == 0)

    # Show the new shape for X and y
    print(f"Dropped {len(df) - len(X)} rows due to missing values.")
    print(X.shape)
    print(y.shape)
```

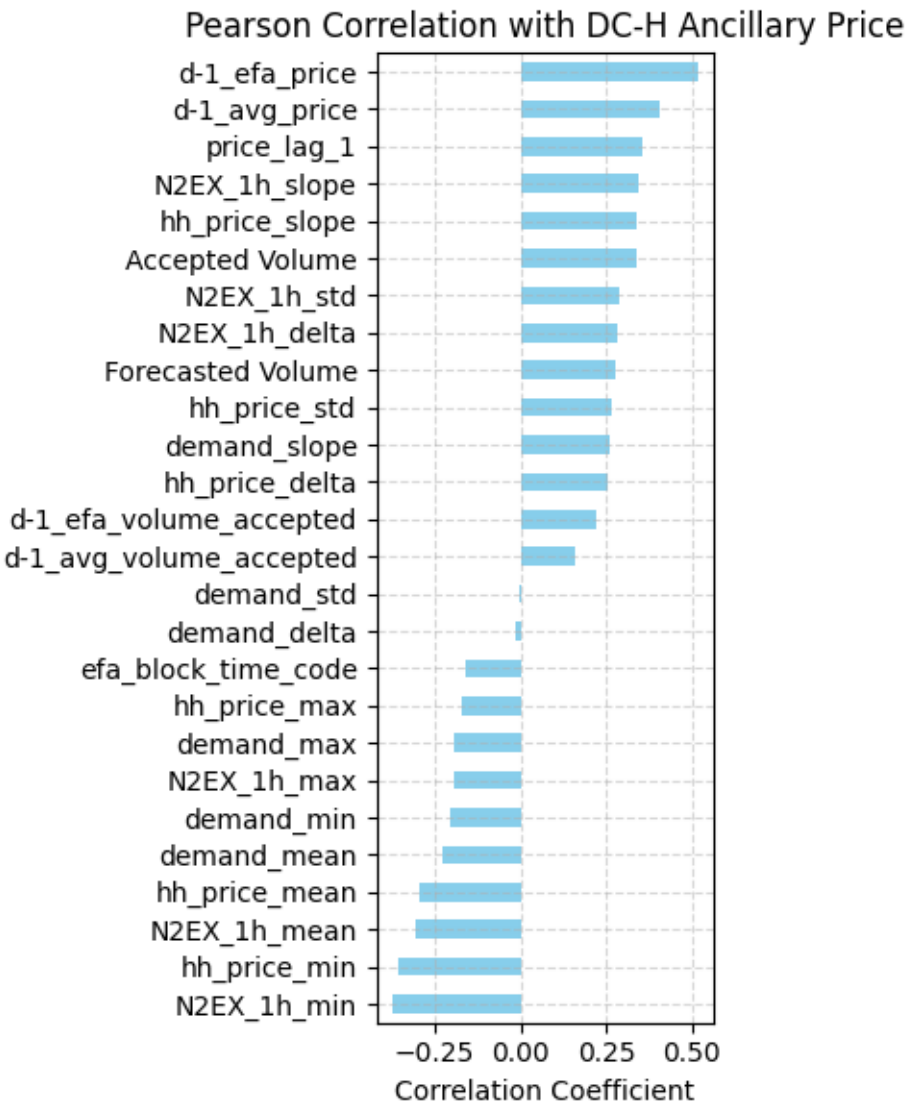
```
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
True
Dropped 6 rows due to missing values.
(2189, 27)
(2189,)
```

2 Feature selection using correlation and permutation importance. In the end, check conllinearity.

[37]:
```
def correlation_plot(service):
    df = merged_service_dfs[service]

    # Drop non-numeric columns
    corr_df = df.select_dtypes(include="number")
```

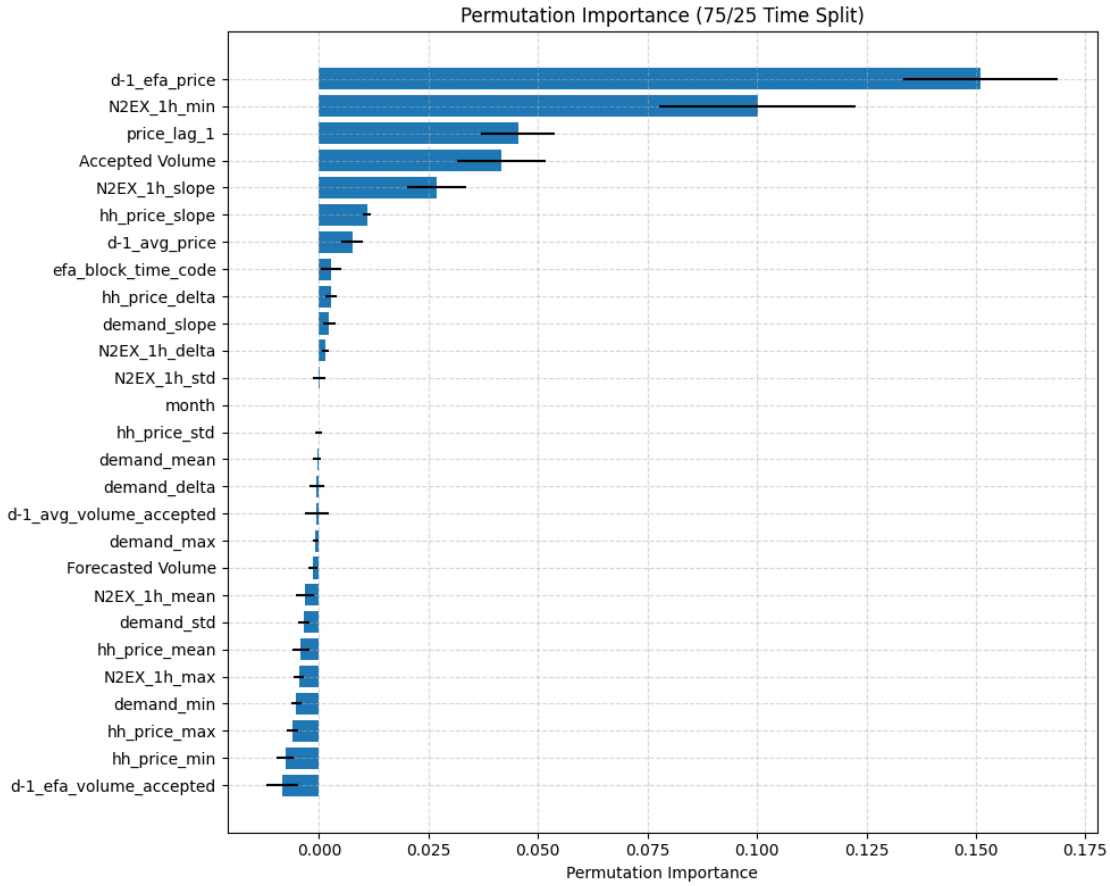
```
    # Compute correlation matrix
    corr_matrix = corr_df.corr()

    # Extract correlation with the target column
    target_corr = corr_matrix["Ancillary Price"].drop("Ancillary Price")

    # Plot only
    plt.figure(figsize=(4, 6))
    target_corr.sort_values().plot(kind='barh', color='skyblue')
    plt.title(f"Pearson Correlation with {service} Ancillary Price")
    plt.xlabel("Correlation Coefficient")
    plt.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.show()
```

[38]:
```
correlation_plot('DC-H')
```



[39]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance

# Use correct dataset
df = merged_service_dfs['DC-H']
X = df.drop(columns=["Ancillary Price"])
y = df["Ancillary Price"]

# Ensure sorted by time
X = X.sort_index()
y = y.loc[X.index]

# Use same 75%-25% time-based split as later
```

```
split_idx = int(len(X) * 0.75)
X_train = X.iloc[:split_idx]
X_test = X.iloc[split_idx:]
y_train = y.iloc[:split_idx]
y_test = y.iloc[split_idx:]

# Train the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Compute permutation importance
result = permutation_importance(model, X_test, y_test, n_repeats=10,␣
 ↪random_state=42)

# Create DataFrame
importances = pd.DataFrame({
    "feature": X.columns,
    "importance_mean": result.importances_mean,
    "importance_std": result.importances_std
}).sort_values(by="importance_mean", ascending=True)

# Plot
plt.figure(figsize=(10, 8))
plt.barh(importances["feature"], importances["importance_mean"],␣
 ↪xerr=importances["importance_std"])
plt.xlabel("Permutation Importance")
plt.title("Permutation Importance (75/25 Time Split)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

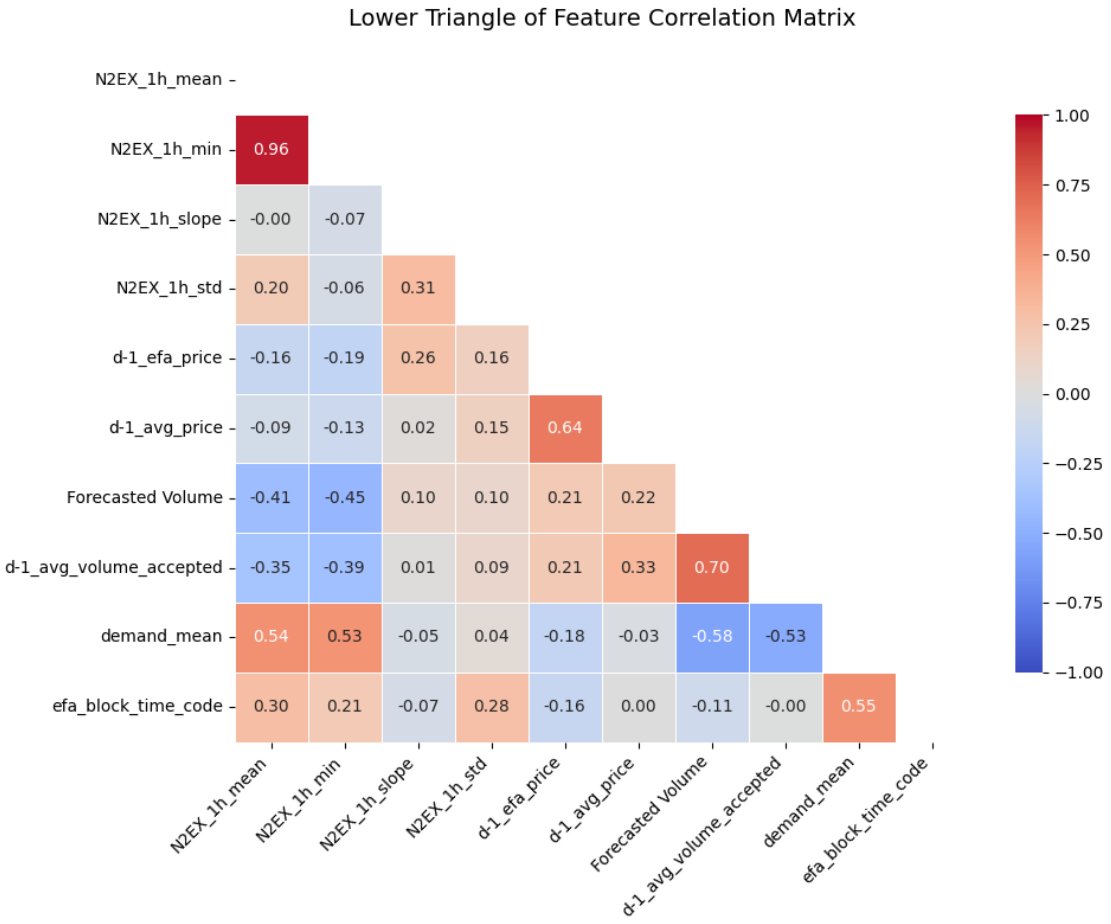


[40]:
```
# DC-H important features, Note Accepted Volume,previous EFA block's ancillary␣
 ↪price,
# and half-hourly prices are lagged variables so we exclude them and other␣
 ↪unimportant features.
features_to_keep = [
    "Ancillary Price",          # Target
    "N2EX_1h_mean",
    "N2EX_1h_min",              # Important non-linear effect
    "N2EX_1h_slope",
    "N2EX_1h_std",
    "d-1_efa_price",            # Previous day's DC-H price of the same EFA␣
 ↪block
    "d-1_avg_price",            # Day-ahead average price
    "Forecasted Volume",        # Available before dispatch
    "d-1_avg_volume_accepted",  # Day-ahead average volume accepted
    "demand_mean",              # Demand level context
    "efa_block_time_code",      # Time-of-day indicator
]
```

```
dch_fe = merged_service_dfs['DC-H'][[col for col in features_to_keep if col in␣
 ↪merged_service_dfs['DC-H'].columns]]
```

[41]:
```
#Drop highly collinear column
df = dch_fe
feature_only_df = df.drop(columns=["Ancillary Price"])

# Set correlation threshold
threshold = 0.7

# Compute correlation matrix
corr_matrix = feature_only_df.corr()

# Mask upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Create the heatmap
sns.heatmap(
    corr_matrix,
    mask=mask,
    cmap='coolwarm',
    annot=True,
    fmt=".2f",
    vmin=-1,
    vmax=1,
    linewidths=0.5,
    cbar_kws={"shrink": 0.8}
)

plt.title("Lower Triangle of Feature Correlation Matrix", fontsize=14, pad=12)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# Identify highly correlated pairs
high_corr_pairs = []
for i in range(corr_matrix.shape[0]):
    for j in range(i):
        if np.abs(corr_matrix.iloc[i, j]) > threshold:
            col1 = corr_matrix.index[i]
            col2 = corr_matrix.columns[j]
            corr_val = corr_matrix.iloc[i, j]
            high_corr_pairs.append((col1, col2, corr_val))
```

```
# Display the high correlation pairs
if high_corr_pairs:
    print(f"Highly correlated feature pairs (threshold > {threshold}):\n")
    for col1, col2, corr_val in sorted(high_corr_pairs, key=lambda x: -x[2]):
        print(f"{col1} and {col2} - correlation: {corr_val:.2f}")
else:
    print(f"No feature pairs with correlation above {threshold}.")
```




```
Highly correlated feature pairs (threshold > 0.7):

N2EX_1h_min and N2EX_1h_mean - correlation: 0.96
```

[42]:
```
# Drop N2EX_1h_mean due to high collinearity
dch_fe.drop(columns=["N2EX_1h_mean", ], inplace=True)
```

```
<ipython-input-42-fd3baf6e596c>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dch_fe.drop(columns=["N2EX_1h_mean", ], inplace=True)
```

BESS_forecasting

April 7, 2025

1 1. Import the relevant packages

[138]:
```
#import relevant packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

2 2. Import the data sets

[139]:
```
#load the data sets
ancillary_volumes_and_prices_4H_2024=pd.read_csv("Ancillary Volumes & Prices␣
 ↪(4H)_2024.csv") #ancillary volumes and prices 2024 data set
day_ahead_prices_H_2024=pd.read_csv("Day-Ahead Price (1H)_2024.csv") #day ahead␣
 ↪price on hourly basis 2024 data set
prices_and_forecasts_HH_2024=pd.read_csv("Prices & Forecasts (HH)_2024.csv")␣
 ↪#prices and forecasts on half hourly basis
```

3 3. Data processing

3.1 Data processing for the Ancillary Volumes & Prices (4H) data set

[140]:
```
#display the datatypes contained within the ancillary volumes and prices dataset
ancillary_volumes_and_prices_4H_2024.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2196 entries, 0 to 2195
Data columns (total 19 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   GMT Time                                        2196 non-null   object
 1   Volume Requirements Forecast - DC-H - GB (MW)   2178 non-null   float64
 2   Volume Requirements Forecast - DC-L - GB (MW)   2178 non-null   float64
 3   Volume Requirements Forecast - DR-H - GB (MW)   2196 non-null   int64
 4   Volume Requirements Forecast - DR-L - GB (MW)   2196 non-null   int64
```

```
 5   Volume Requirements Forecast - DM-H - GB (MW)  2196 non-null   int64
 6   Volume Requirements Forecast - DM-L - GB (MW)  2196 non-null   int64
 7   Ancillary Volume Accepted - DC-H - GB (MW)     2196 non-null   int64
 8   Ancillary Volume Accepted - DC-L - GB (MW)     2196 non-null   int64
 9   Ancillary Volume Accepted - DR-H - GB (MW)     2196 non-null   int64
 10  Ancillary Volume Accepted - DR-L - GB (MW)     2196 non-null   int64
 11  Ancillary Volume Accepted - DM-H - GB (MW)     2196 non-null   int64
 12  Ancillary Volume Accepted - DM-L - GB (MW)     2196 non-null   int64
 13  Ancillary Price - DC-H - GB (£/MW/h)           2196 non-null   float64
 14  Ancillary Price - DC-L - GB (£/MW/h)           2196 non-null   float64
 15  Ancillary Price - DR-H - GB (£/MW/h)           2196 non-null   float64
 16  Ancillary Price - DR-L - GB (£/MW/h)           2196 non-null   float64
 17  Ancillary Price - DM-H - GB (£/MW/h)           2196 non-null   float64
 18  Ancillary Price - DM-L - GB (£/MW/h)           2196 non-null   float64
dtypes: float64(8), int64(10), object(1)
memory usage: 326.1+ KB
```

[141]:
```
#display a statistical description of the data contained in the file
ancillary_volumes_and_prices_4H_2024.describe()
```

[141]:
```
       Volume Requirements Forecast - DC-H - GB (MW)  \
count                                    2178.000000
mean                                     1240.994949
std                                       125.781264
min                                       885.000000
25%                                      1161.000000
50%                                      1250.000000
75%                                      1343.000000
max                                      1643.000000

       Volume Requirements Forecast - DC-L - GB (MW)  \
count                                    2178.000000
mean                                     1117.454545
std                                       120.332223
min                                       772.000000
25%                                      1042.000000
50%                                      1116.500000
75%                                      1203.750000
max                                      1433.000000

       Volume Requirements Forecast - DR-H - GB (MW)  \
count                                         2196.0
mean                                           330.0
std                                              0.0
min                                            330.0
25%                                            330.0
50%                                            330.0
```

```
75%                                          330.0
max                                          330.0

       Volume Requirements Forecast - DR-L - GB (MW)  \
count                                       2196.0
mean                                         330.0
std                                            0.0
min                                          330.0
25%                                          330.0
50%                                          330.0
75%                                          330.0
max                                          330.0

       Volume Requirements Forecast - DM-H - GB (MW)  \
count                                    2196.000000
mean                                      179.881603
std                                        37.433429
min                                       150.000000
25%                                       150.000000
50%                                       200.000000
75%                                       200.000000
max                                       480.000000

       Volume Requirements Forecast - DM-L - GB (MW)  \
count                                    2196.000000
mean                                      162.367942
std                                        18.539949
min                                       150.000000
25%                                       150.000000
50%                                       170.000000
75%                                       170.000000
max                                       330.000000

       Ancillary Volume Accepted - DC-H - GB (MW)  \
count                                  2196.000000
mean                                   1235.194900
std                                     129.799061
min                                     422.000000
25%                                    1157.750000
50%                                    1250.000000
75%                                    1342.000000
max                                    1643.000000

       Ancillary Volume Accepted - DC-L - GB (MW)  \
count                                  2196.000000
mean                                   1117.377049
std                                     122.849319
```

```
min                                    764.000000
25%                                   1040.000000
50%                                   1112.000000
75%                                   1205.000000
max                                   1439.000000

       Ancillary Volume Accepted - DR-H - GB (MW)  \
count                                 2196.000000
mean                                   344.853825
std                                     20.053869
min                                    148.000000
25%                                    350.000000
50%                                    350.000000
75%                                    350.000000
max                                    350.000000

       Ancillary Volume Accepted - DR-L - GB (MW)  \
count                                 2196.000000
mean                                   300.559199
std                                     40.430707
min                                    136.000000
25%                                    271.000000
50%                                    328.000000
75%                                    330.000000
max                                    335.000000

       Ancillary Volume Accepted - DM-H - GB (MW)  \
count                                 2196.000000
mean                                   186.360656
std                                     40.081752
min                                     22.000000
25%                                    152.000000
50%                                    200.000000
75%                                    213.250000
max                                    500.000000

       Ancillary Volume Accepted - DM-L - GB (MW)  \
count                                 2196.000000
mean                                   158.597905
std                                     26.090590
min                                      0.000000
25%                                    150.000000
50%                                    166.500000
75%                                    170.000000
max                                    350.000000

       Ancillary Price - DC-H - GB (£/MW/h)  \
```

```
count                            2196.000000
mean                                1.916453
std                                 1.522454
min                                -0.260000
25%                                 0.937500
50%                                 1.400000
75%                                 2.490000
max                                10.000000

       Ancillary Price - DC-L - GB (£/MW/h)  \
count                            2196.000000
mean                                2.241261
std                                 2.742733
min                                -0.710000
25%                                 1.120000
50%                                 1.670000
75%                                 2.560000
max                                70.690000

       Ancillary Price - DR-H - GB (£/MW/h)  \
count                            2196.000000
mean                               -4.798402
std                                 3.847461
min                               -15.350000
25%                                -7.252500
50%                                -5.620000
75%                                -3.305000
max                                13.960000

       Ancillary Price - DR-L - GB (£/MW/h)  \
count                            2196.000000
mean                                8.180679
std                                 3.391109
min                                 1.630000
25%                                 6.380000
50%                                 7.520000
75%                                 9.260000
max                                70.690000

       Ancillary Price - DM-H - GB (£/MW/h)  \
count                            2196.000000
mean                                0.561821
std                                 1.944715
min                                -6.010000
25%                                -0.712500
50%                                -0.110000
75%                                 1.250000
```

```
max                                 13.860000

       Ancillary Price - DM-L - GB (£/MW/h)
count                           2196.000000
mean                               4.203934
std                                1.790640
min                              -20.000000
25%                                3.187500
50%                                3.955000
75%                                5.000000
max                               12.250000
```

[142]:
```
#check which fields have missing values and the total number of missing values
ancillary_volumes_and_prices_4H_2024.isnull().sum()
```

[142]:
```
GMT Time                                            0
Volume Requirements Forecast - DC-H - GB (MW)      18
Volume Requirements Forecast - DC-L - GB (MW)      18
Volume Requirements Forecast - DR-H - GB (MW)       0
Volume Requirements Forecast - DR-L - GB (MW)       0
Volume Requirements Forecast - DM-H - GB (MW)       0
Volume Requirements Forecast - DM-L - GB (MW)       0
Ancillary Volume Accepted - DC-H - GB (MW)          0
Ancillary Volume Accepted - DC-L - GB (MW)          0
Ancillary Volume Accepted - DR-H - GB (MW)          0
Ancillary Volume Accepted - DR-L - GB (MW)          0
Ancillary Volume Accepted - DM-H - GB (MW)          0
Ancillary Volume Accepted - DM-L - GB (MW)          0
Ancillary Price - DC-H - GB (£/MW/h)                0
Ancillary Price - DC-L - GB (£/MW/h)                0
Ancillary Price - DR-H - GB (£/MW/h)                0
Ancillary Price - DR-L - GB (£/MW/h)                0
Ancillary Price - DM-H - GB (£/MW/h)                0
Ancillary Price - DM-L - GB (£/MW/h)                0
dtype: int64
```

[143]:
```
#display the missing records
ancillary_volumes_and_prices_4H_2024[ancillary_volumes_and_prices_4H_2024["Volume␣
 ↪Requirements Forecast - DC-L - GB (MW)"].isnull()]
```

[143]:
```
              GMT Time  Volume Requirements Forecast - DC-H - GB (MW)  \
1812  29/10/2024 03:00                                            NaN
1813  29/10/2024 07:00                                            NaN
1814  29/10/2024 11:00                                            NaN
1815  29/10/2024 15:00                                            NaN
1816  29/10/2024 19:00                                            NaN
1817  29/10/2024 23:00                                            NaN
```

1818	30/10/2024 03:00	NaN
1819	30/10/2024 07:00	NaN
1820	30/10/2024 11:00	NaN
1821	30/10/2024 15:00	NaN
1822	30/10/2024 19:00	NaN
1823	30/10/2024 23:00	NaN
1824	31/10/2024 03:00	NaN
1825	31/10/2024 07:00	NaN
1826	31/10/2024 11:00	NaN
1827	31/10/2024 15:00	NaN
1828	31/10/2024 19:00	NaN
1829	31/10/2024 23:00	NaN

	Volume Requirements Forecast - DC-L - GB (MW) \
1812	NaN
1813	NaN
1814	NaN
1815	NaN
1816	NaN
1817	NaN
1818	NaN
1819	NaN
1820	NaN
1821	NaN
1822	NaN
1823	NaN
1824	NaN
1825	NaN
1826	NaN
1827	NaN
1828	NaN
1829	NaN

	Volume Requirements Forecast - DR-H - GB (MW) \
1812	330
1813	330
1814	330
1815	330
1816	330
1817	330
1818	330
1819	330
1820	330
1821	330
1822	330
1823	330
1824	330

```
1825                                                330
1826                                                330
1827                                                330
1828                                                330
1829                                                330

      Volume Requirements Forecast - DR-L - GB (MW)  \
1812                                                330
1813                                                330
1814                                                330
1815                                                330
1816                                                330
1817                                                330
1818                                                330
1819                                                330
1820                                                330
1821                                                330
1822                                                330
1823                                                330
1824                                                330
1825                                                330
1826                                                330
1827                                                330
1828                                                330
1829                                                330

      Volume Requirements Forecast - DM-H - GB (MW)  \
1812                                                200
1813                                                200
1814                                                200
1815                                                200
1816                                                200
1817                                                200
1818                                                200
1819                                                200
1820                                                200
1821                                                200
1822                                                200
1823                                                200
1824                                                200
1825                                                200
1826                                                200
1827                                                200
1828                                                200
1829                                                200

      Volume Requirements Forecast - DM-L - GB (MW)  \
```

1812	170
1813	170
1814	170
1815	170
1816	170
1817	170
1818	170
1819	170
1820	170
1821	170
1822	170
1823	170
1824	170
1825	170
1826	170
1827	170
1828	170
1829	170

	Ancillary Volume Accepted - DC-H - GB (MW) \
1812	1255
1813	1127
1814	1076
1815	1060
1816	1084
1817	1170
1818	1202
1819	1087
1820	1060
1821	1051
1822	1103
1823	1258
1824	1337
1825	1147
1826	1128
1827	1077
1828	1165
1829	1285

	Ancillary Volume Accepted - DC-L - GB (MW) \
1812	1164
1813	1145
1814	1172
1815	1086
1816	1098
1817	1146
1818	1162

1819	1065
1820	1061
1821	1118
1822	1109
1823	1258
1824	1287
1825	1209
1826	1216
1827	1193
1828	1197
1829	1259

	Ancillary Volume Accepted - DR-H - GB (MW) \
1812	350
1813	350
1814	350
1815	333
1816	350
1817	350
1818	350
1819	350
1820	350
1821	350
1822	350
1823	350
1824	350
1825	350
1826	350
1827	350
1828	350
1829	350

	Ancillary Volume Accepted - DR-L - GB (MW) \
1812	325
1813	330
1814	330
1815	330
1816	330
1817	330
1818	330
1819	330
1820	330
1821	330
1822	330
1823	330
1824	330
1825	330

1826	330
1827	330
1828	330
1829	330

	Ancillary Volume Accepted - DM-H - GB (MW) \
1812	200
1813	220
1814	200
1815	200
1816	220
1817	201
1818	200
1819	220
1820	201
1821	200
1822	220
1823	200
1824	200
1825	216
1826	200
1827	200
1828	208
1829	200

	Ancillary Volume Accepted - DM-L - GB (MW) \
1812	170
1813	170
1814	170
1815	56
1816	166
1817	170
1818	170
1819	170
1820	170
1821	61
1822	170
1823	170
1824	170
1825	170
1826	170
1827	169
1828	170
1829	170

	Ancillary Price - DC-H - GB (£/MW/h) \
1812	2.50

1813	1.71
1814	2.99
1815	3.75
1816	1.03
1817	1.67
1818	2.42
1819	1.00
1820	2.12
1821	3.00
1822	1.39
1823	2.12
1824	5.50
1825	1.00
1826	2.09
1827	1.37
1828	1.03
1829	2.00

	Ancillary Price - DC-L - GB (£/MW/h) \
1812	1.63
1813	2.90
1814	2.75
1815	20.30
1816	5.40
1817	1.66
1818	2.16
1819	2.75
1820	1.25
1821	14.40
1822	3.00
1823	4.03
1824	2.11
1825	3.17
1826	2.04
1827	7.00
1828	2.15
1829	3.37

	Ancillary Price - DR-H - GB (£/MW/h) \
1812	-3.72
1813	-8.01
1814	-5.39
1815	-7.50
1816	-7.92
1817	-7.02
1818	-3.79
1819	-9.15

1820	-6.94
1821	-8.53
1822	-8.03
1823	-6.28
1824	-2.00
1825	-7.96
1826	-6.28
1827	-8.12
1828	-7.63
1829	-6.51

	Ancillary Price - DR-L - GB (£/MW/h) \
1812	10.69
1813	8.69
1814	5.22
1815	20.00
1816	9.00
1817	6.02
1818	7.00
1819	8.00
1820	8.69
1821	13.09
1822	7.02
1823	7.60
1824	6.00
1825	5.51
1826	6.00
1827	10.00
1828	6.00
1829	7.00

	Ancillary Price - DM-H - GB (£/MW/h) \
1812	0.25
1813	-0.25
1814	1.50
1815	5.67
1816	-0.28
1817	0.00
1818	1.23
1819	-1.12
1820	-0.35
1821	2.75
1822	-1.59
1823	0.24
1824	3.75
1825	-0.54
1826	0.00

```
1827                                  2.23
1828                                 -1.25
1829                                  0.11

      Ancillary Price - DM-L - GB (£/MW/h)
1812                                  5.11
1813                                  6.95
1814                                  4.83
1815                                 10.00
1816                                  8.91
1817                                  4.61
1818                                  5.23
1819                                  7.06
1820                                  4.67
1821                                 10.00
1822                                  5.95
1823                                  5.28
1824                                  5.28
1825                                  5.40
1826                                  4.38
1827                                  8.75
1828                                  5.11
1829                                  5.78
```

[144]:
```
#Fill in the missing data

#insert data for DC-H for 29/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1812:1813,1:2]=int(1185)
ancillary_volumes_and_prices_4H_2024.iloc[1813:1814,1:2]=int(1230)
ancillary_volumes_and_prices_4H_2024.iloc[1814:1815,1:2]=int(1108)
ancillary_volumes_and_prices_4H_2024.iloc[1815:1816,1:2]=int(1060)
ancillary_volumes_and_prices_4H_2024.iloc[1816:1817,1:2]=int(1056)
ancillary_volumes_and_prices_4H_2024.iloc[1817:1818,1:2]=int(1090)

#insert data for DC-H for 30/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1818:1819,1:2]=int(1172)
ancillary_volumes_and_prices_4H_2024.iloc[1819:1820,1:2]=int(1200)
ancillary_volumes_and_prices_4H_2024.iloc[1820:1821,1:2]=int(1095)
ancillary_volumes_and_prices_4H_2024.iloc[1821:1822,1:2]=int(1060)
ancillary_volumes_and_prices_4H_2024.iloc[1822:1823,1:2]=int(1052)
ancillary_volumes_and_prices_4H_2024.iloc[1823:1824,1:2]=int(1097)

#insert data for DC-H for 31/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1824:1825,1:2]=int(1255)
ancillary_volumes_and_prices_4H_2024.iloc[1825:1826,1:2]=int(1337)
ancillary_volumes_and_prices_4H_2024.iloc[1826:1827,1:2]=int(1156)
ancillary_volumes_and_prices_4H_2024.iloc[1827:1828,1:2]=int(1109)
```

```
ancillary_volumes_and_prices_4H_2024.iloc[1828:1829,1:2]=int(1087)
ancillary_volumes_and_prices_4H_2024.iloc[1829:1830,1:2]=int(1138)

#insert data for DC-L for 29/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1812:1813,2:3]=int(1146)
ancillary_volumes_and_prices_4H_2024.iloc[1813:1814,2:3]=int(1162)
ancillary_volumes_and_prices_4H_2024.iloc[1814:1815,2:3]=int(1111)
ancillary_volumes_and_prices_4H_2024.iloc[1815:1816,2:3]=int(1030)
ancillary_volumes_and_prices_4H_2024.iloc[1816:1817,2:3]=int(1013)
ancillary_volumes_and_prices_4H_2024.iloc[1817:1818,2:3]=int(1087)

#insert data for DC-L for 30/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1818:1819,2:3]=int(1145)
ancillary_volumes_and_prices_4H_2024.iloc[1819:1820,2:3]=int(1173)
ancillary_volumes_and_prices_4H_2024.iloc[1820:1821,2:3]=int(1103)
ancillary_volumes_and_prices_4H_2024.iloc[1821:1822,2:3]=int(1048)
ancillary_volumes_and_prices_4H_2024.iloc[1822:1823,2:3]=int(1025)
ancillary_volumes_and_prices_4H_2024.iloc[1823:1824,2:3]=int(1123)

#insert data for DC-L for 31/10/2024
ancillary_volumes_and_prices_4H_2024.iloc[1824:1825,2:3]=int(1258)
ancillary_volumes_and_prices_4H_2024.iloc[1825:1826,2:3]=int(1326)
ancillary_volumes_and_prices_4H_2024.iloc[1826:1827,2:3]=int(1272)
ancillary_volumes_and_prices_4H_2024.iloc[1827:1828,2:3]=int(1199)
ancillary_volumes_and_prices_4H_2024.iloc[1828:1829,2:3]=int(1172)
ancillary_volumes_and_prices_4H_2024.iloc[1829:1830,2:3]=int(1238)
```

[145]:
```
#check again for missing data
ancillary_volumes_and_prices_4H_2024["Volume Requirements Forecast - DC-L - GB␣
↪(MW)"].isnull().sum()
```

[145]:
```
0
```

No missing data in the ancillary volumes data set

3.2 Data processing for the Day-Ahead Price (1H) dataset

[146]:
```
#statistical description of the data contained in the dataset
day_ahead_prices_H_2024.describe()
```

[146]:
```
       Day Ahead Price (N2EX, local) - GB (£/MWh)  \
count                                 8783.000000
mean                                    72.574383
std                                     30.003754
min                                    -24.680000
25%                                     60.075000
50%                                     73.870000
```

```
75%                                      88.010000
max                                     496.300000

       Day Ahead Price (EPEX, local) - GB (£/MWh)
count                                  8784.000000
mean                                     72.262324
std                                      28.692083
min                                     -21.300000
25%                                      60.200000
50%                                      73.200000
75%                                      87.000000
max                                     430.200000
```

[147]:
```
#data types contained within the dataset
day_ahead_prices_H_2024.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 3 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   GMT Time                                      8784 non-null   object
 1   Day Ahead Price (N2EX, local) - GB (£/MWh)    8783 non-null   float64
 2   Day Ahead Price (EPEX, local) - GB (£/MWh)    8784 non-null   float64
dtypes: float64(2), object(1)
memory usage: 206.0+ KB
```

[148]:
```
#check for any missing data points
day_ahead_prices_H_2024.isnull().sum()
```

[148]:
```
GMT Time                                      0
Day Ahead Price (N2EX, local) - GB (£/MWh)    1
Day Ahead Price (EPEX, local) - GB (£/MWh)    0
dtype: int64
```

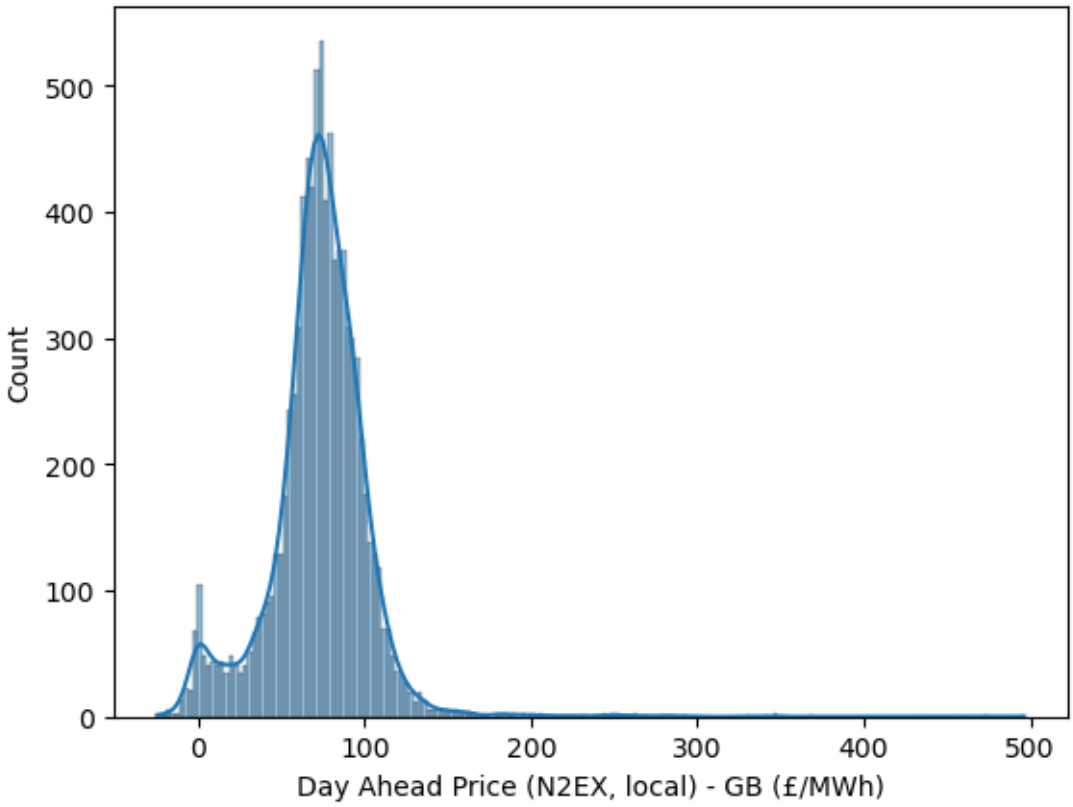
[149]:
```
#view the missing data point
day_ahead_prices_H_2024[day_ahead_prices_H_2024["Day Ahead Price (N2EX, local)␣
 ↪- GB (£/MWh)"].isnull()]
```

[149]:
```
              GMT Time  Day Ahead Price (N2EX, local) - GB (£/MWh)  \
4366  30/06/2024 22:00                                         NaN

      Day Ahead Price (EPEX, local) - GB (£/MWh)
4366                                        80.0
```

[150]:
```
#see whether the data follows any distribution
```

```
sns.histplot(day_ahead_prices_H_2024["Day Ahead Price (N2EX, local) - GB (£/
↪MWh)"],kde=True)
```

[150]: <Axes: xlabel='Day Ahead Price (N2EX, local) - GB (£/MWh)', ylabel='Count'>



From the scatterplot, we can see that throughout the year, the day ahead prices on the N2EX market mainly stay at the mean price. As a result, we can fill in the missing data point with the mean

[151]:
```
#fill in the missing data with the mean of the column
day_ahead_prices_H_2024["Day Ahead Price (N2EX, local) - GB (£/
↪MWh)"]=day_ahead_prices_H_2024["Day Ahead Price (N2EX, local) - GB (£/MWh)"].
↪fillna(day_ahead_prices_H_2024["Day Ahead Price (N2EX, local) - GB (£/MWh)"].
↪mean())
```

3.3 Data processing for the Prices & Forecasts (HH) dataset

[152]:
```
#statistical description of the data
prices_and_forecasts_HH_2024.describe()
```

[152]:
```
       National Demand Forecast (NDF) - GB (MW)  \
count                                  17568.000000
mean                                   26278.461293
std                                     5966.705317
min                                    14880.000000
25%                                    21598.000000
50%                                    25348.000000
75%                                    30231.500000
max                                    45300.000000

       Day Ahead Price (EPEX half-hourly, local) - GB (£/MWh)
count                                          17568.000000
mean                                              72.418544
std                                               30.612697
min                                              -40.000000
25%                                               60.000000
50%                                               73.680000
75%                                               88.000000
max                                              506.560000
```

[153]:
```
#check if there are any missing data points
prices_and_forecasts_HH_2024.isnull().sum()
```

[153]:
```
GMT Time                                                  0
National Demand Forecast (NDF) - GB (MW)                  0
Day Ahead Price (EPEX half-hourly, local) - GB (£/MWh)    0
dtype: int64
```

We can see that there are no missing data points from this data set

4 4. Feature engineering

The data we have is mismatched. The frequency response data is on a four hourly basis while the electricity prices are on an hourly and half hourly basis.

4.1 Resample the data from the 4H and 1H data to HH

[154]:
```
#All the datetime columns are shown as objects in each data set. Convert these␣
 ↪columns to datetime format
ancillary_volumes_and_prices_4H_2024["GMT Time"]=pd.
 ↪to_datetime(ancillary_volumes_and_prices_4H_2024["GMT Time"],format='%d/%m/
 ↪%Y %H:%M')
day_ahead_prices_H_2024["GMT Time"]=pd.to_datetime(day_ahead_prices_H_2024["GMT␣
 ↪Time"],format='%d/%m/%Y %H:%M')
prices_and_forecasts_HH_2024["GMT Time"]=pd.
 ↪to_datetime(prices_and_forecasts_HH_2024["GMT Time"],format='%d/%m/%Y %H:%M')
```

[155]:
```
#resample the ancillary_volumes_and_prices_4H_ to 30 minutes by forward filling
ancillary_volumes_and_prices_4H_2024.set_index("GMT Time",inplace=True) #first␣
 ↪set the GMT time column as the index
ancillary_volumes_and_prices_HH_2024_resampled=ancillary_volumes_and_prices_4H_2024.
 ↪resample('30min').ffill() #resample and forward fill the data to create half␣
 ↪hourly data points
```

[156]:
```
#Resampling and forward filling of day ahead prices
day_ahead_prices_H_2024.set_index("GMT Time",inplace=True) #first set the GMT␣
 ↪time column as the index
day_ahead_prices_HH_2024_resampled=day_ahead_prices_H_2024.resample('30min').
 ↪ffill() #resample and forward fill the data to create half hourly data points
```

[157]:
```
prices_and_forecasts_HH_2024_resampled=prices_and_forecasts_HH_2024
```

4.2 Merge the data sets

[158]:
```
#merge the ancillary data set and day ahead pricing data set for easier␣
 ↪processing
#name the merged data set ancillary_data_and_day_ahead_prices_2024
ancillary_data_and_day_ahead_prices_2024=pd.
 ↪merge(ancillary_volumes_and_prices_HH_2024_resampled,day_ahead_prices_HH_2024_resampled,on=
 ↪Time",how='inner')
```

[159]:
```
#merge all the data on the GMT time column
merged_data_2024=pd.
 ↪merge(prices_and_forecasts_HH_2024_resampled,ancillary_data_and_day_ahead_prices_2024,on="G
 ↪Time",how='inner')
```

[160]:
```
#set the index of the merged data to GMT Time
merged_data_2024.set_index('GMT Time')
```

[160]:
```
                     National Demand Forecast (NDF) - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                      19638
2024-01-01 03:30:00                                      18884
2024-01-01 04:00:00                                      18296
2024-01-01 04:30:00                                      18060
2024-01-01 05:00:00                                      18092
…                                                        …
2024-12-31 21:00:00                                      26077
2024-12-31 21:30:00                                      24629
2024-12-31 22:00:00                                      23482
2024-12-31 22:30:00                                      22625
2024-12-31 23:00:00                                      21752

                     Day Ahead Price (EPEX half-hourly, local) - GB (£/MWh)  \
```

GMT Time	
2024-01-01 03:00:00	24.80
2024-01-01 03:30:00	11.20
2024-01-01 04:00:00	11.00
2024-01-01 04:30:00	-11.60
2024-01-01 05:00:00	10.00
…	…
2024-12-31 21:00:00	46.00
2024-12-31 21:30:00	15.00
2024-12-31 22:00:00	10.29
2024-12-31 22:30:00	-20.00
2024-12-31 23:00:00	12.00

GMT Time	Volume Requirements Forecast - DC-H - GB (MW) \
2024-01-01 03:00:00	1250.0
2024-01-01 03:30:00	1250.0
2024-01-01 04:00:00	1250.0
2024-01-01 04:30:00	1250.0
2024-01-01 05:00:00	1250.0
…	…
2024-12-31 21:00:00	1205.0
2024-12-31 21:30:00	1205.0
2024-12-31 22:00:00	1205.0
2024-12-31 22:30:00	1205.0
2024-12-31 23:00:00	1275.0

GMT Time	Volume Requirements Forecast - DC-L - GB (MW) \
2024-01-01 03:00:00	933.0
2024-01-01 03:30:00	933.0
2024-01-01 04:00:00	933.0
2024-01-01 04:30:00	933.0
2024-01-01 05:00:00	933.0
…	…
2024-12-31 21:00:00	1256.0
2024-12-31 21:30:00	1256.0
2024-12-31 22:00:00	1256.0
2024-12-31 22:30:00	1256.0
2024-12-31 23:00:00	1324.0

GMT Time	Volume Requirements Forecast - DR-H - GB (MW) \
2024-01-01 03:00:00	330
2024-01-01 03:30:00	330
2024-01-01 04:00:00	330
2024-01-01 04:30:00	330

```
2024-01-01 05:00:00                                          330
…                                                            …
2024-12-31 21:00:00                                          330
2024-12-31 21:30:00                                          330
2024-12-31 22:00:00                                          330
2024-12-31 22:30:00                                          330
2024-12-31 23:00:00                                          330

                    Volume Requirements Forecast - DR-L - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                          330
2024-01-01 03:30:00                                          330
2024-01-01 04:00:00                                          330
2024-01-01 04:30:00                                          330
2024-01-01 05:00:00                                          330
…                                                            …
2024-12-31 21:00:00                                          330
2024-12-31 21:30:00                                          330
2024-12-31 22:00:00                                          330
2024-12-31 22:30:00                                          330
2024-12-31 23:00:00                                          330

                    Volume Requirements Forecast - DM-H - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                          150
2024-01-01 03:30:00                                          150
2024-01-01 04:00:00                                          150
2024-01-01 04:30:00                                          150
2024-01-01 05:00:00                                          150
…                                                            …
2024-12-31 21:00:00                                          200
2024-12-31 21:30:00                                          200
2024-12-31 22:00:00                                          200
2024-12-31 22:30:00                                          200
2024-12-31 23:00:00                                          200

                    Volume Requirements Forecast - DM-L - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                          150
2024-01-01 03:30:00                                          150
2024-01-01 04:00:00                                          150
2024-01-01 04:30:00                                          150
2024-01-01 05:00:00                                          150
…                                                            …
2024-12-31 21:00:00                                          170
2024-12-31 21:30:00                                          170
2024-12-31 22:00:00                                          170
```

```
2024-12-31 22:30:00                                          170
2024-12-31 23:00:00                                          170

                    Ancillary Volume Accepted - DC-H - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                         1250
2024-01-01 03:30:00                                         1250
2024-01-01 04:00:00                                         1250
2024-01-01 04:30:00                                         1250
2024-01-01 05:00:00                                         1250
…                                                            …
2024-12-31 21:00:00                                         1286
2024-12-31 21:30:00                                         1286
2024-12-31 22:00:00                                         1286
2024-12-31 22:30:00                                         1286
2024-12-31 23:00:00                                         1319

                    Ancillary Volume Accepted - DC-L - GB (MW)  …  \
GMT Time                                                        …
2024-01-01 03:00:00                                         1039  …
2024-01-01 03:30:00                                         1039  …
2024-01-01 04:00:00                                         1039  …
2024-01-01 04:30:00                                         1039  …
2024-01-01 05:00:00                                         1039  …
…                                                           …  …
2024-12-31 21:00:00                                         1351  …
2024-12-31 21:30:00                                         1351  …
2024-12-31 22:00:00                                         1351  …
2024-12-31 22:30:00                                         1351  …
2024-12-31 23:00:00                                         1285  …

                    Ancillary Volume Accepted - DM-H - GB (MW)  \
GMT Time
2024-01-01 03:00:00                                          150
2024-01-01 03:30:00                                          150
2024-01-01 04:00:00                                          150
2024-01-01 04:30:00                                          150
2024-01-01 05:00:00                                          150
…                                                            …
2024-12-31 21:00:00                                          200
2024-12-31 21:30:00                                          200
2024-12-31 22:00:00                                          200
2024-12-31 22:30:00                                          200
2024-12-31 23:00:00                                          200

                    Ancillary Volume Accepted - DM-L - GB (MW)  \
GMT Time
```

2024-01-01 03:00:00	150
2024-01-01 03:30:00	150
2024-01-01 04:00:00	150
2024-01-01 04:30:00	150
2024-01-01 05:00:00	150
…	…
2024-12-31 21:00:00	170
2024-12-31 21:30:00	170
2024-12-31 22:00:00	170
2024-12-31 22:30:00	170
2024-12-31 23:00:00	170

GMT Time	Ancillary Price - DC-H - GB (£/MW/h) \
2024-01-01 03:00:00	5.00
2024-01-01 03:30:00	5.00
2024-01-01 04:00:00	5.00
2024-01-01 04:30:00	5.00
2024-01-01 05:00:00	5.00
…	…
2024-12-31 21:00:00	1.51
2024-12-31 21:30:00	1.51
2024-12-31 22:00:00	1.51
2024-12-31 22:30:00	1.51
2024-12-31 23:00:00	3.00

GMT Time	Ancillary Price - DC-L - GB (£/MW/h) \
2024-01-01 03:00:00	0.78
2024-01-01 03:30:00	0.78
2024-01-01 04:00:00	0.78
2024-01-01 04:30:00	0.78
2024-01-01 05:00:00	0.78
…	…
2024-12-31 21:00:00	4.40
2024-12-31 21:30:00	4.40
2024-12-31 22:00:00	4.40
2024-12-31 22:30:00	4.40
2024-12-31 23:00:00	2.50

GMT Time	Ancillary Price - DR-H - GB (£/MW/h) \
2024-01-01 03:00:00	2.17
2024-01-01 03:30:00	2.17
2024-01-01 04:00:00	2.17
2024-01-01 04:30:00	2.17
2024-01-01 05:00:00	2.17

```
…                                                …
2024-12-31 21:00:00                             -1.15
2024-12-31 21:30:00                             -1.15
2024-12-31 22:00:00                             -1.15
2024-12-31 22:30:00                             -1.15
2024-12-31 23:00:00                              4.25

                    Ancillary Price - DR-L - GB (£/MW/h)  \
GMT Time
2024-01-01 03:00:00                              7.65
2024-01-01 03:30:00                              7.65
2024-01-01 04:00:00                              7.65
2024-01-01 04:30:00                              7.65
2024-01-01 05:00:00                              7.65
…                                                …
2024-12-31 21:00:00                              9.77
2024-12-31 21:30:00                              9.77
2024-12-31 22:00:00                              9.77
2024-12-31 22:30:00                              9.77
2024-12-31 23:00:00                              4.35

                    Ancillary Price - DM-H - GB (£/MW/h)  \
GMT Time
2024-01-01 03:00:00                              3.59
2024-01-01 03:30:00                              3.59
2024-01-01 04:00:00                              3.59
2024-01-01 04:30:00                              3.59
2024-01-01 05:00:00                              3.59
…                                                …
2024-12-31 21:00:00                              1.40
2024-12-31 21:30:00                              1.40
2024-12-31 22:00:00                              1.40
2024-12-31 22:30:00                              1.40
2024-12-31 23:00:00                              2.76

                    Ancillary Price - DM-L - GB (£/MW/h)  \
GMT Time
2024-01-01 03:00:00                              3.49
2024-01-01 03:30:00                              3.49
2024-01-01 04:00:00                              3.49
2024-01-01 04:30:00                              3.49
2024-01-01 05:00:00                              3.49
…                                                …
2024-12-31 21:00:00                              6.50
2024-12-31 21:30:00                              6.50
2024-12-31 22:00:00                              6.50
2024-12-31 22:30:00                              6.50
```

```
2024-12-31 23:00:00                                    3.31

                     Day Ahead Price (N2EX, local) - GB (£/MWh)  \
GMT Time
2024-01-01 03:00:00                                          0.15
2024-01-01 03:30:00                                          0.15
2024-01-01 04:00:00                                         -0.29
2024-01-01 04:30:00                                         -0.29
2024-01-01 05:00:00                                         -0.20
…                                                             …
2024-12-31 21:00:00                                         27.71
2024-12-31 21:30:00                                         27.71
2024-12-31 22:00:00                                          2.06
2024-12-31 22:30:00                                          2.06
2024-12-31 23:00:00                                          0.03

                     Day Ahead Price (EPEX, local) - GB (£/MWh)
GMT Time
2024-01-01 03:00:00                                          25.8
2024-01-01 03:30:00                                          25.8
2024-01-01 04:00:00                                          20.6
2024-01-01 04:30:00                                          20.6
2024-01-01 05:00:00                                          15.3
…                                                             …
2024-12-31 21:00:00                                          19.9
2024-12-31 21:30:00                                          19.9
2024-12-31 22:00:00                                           7.0
2024-12-31 22:30:00                                           7.0
2024-12-31 23:00:00                                           6.0

[17561 rows x 22 columns]
```

[161]:
```
#statistical description of the merged data
merged_data_2024.describe()
```

[161]:
```
       National Demand Forecast (NDF) - GB (MW)  \
count                              17561.000000
mean                               26280.255054
std                                 5967.186850
min                                14880.000000
25%                                21600.000000
50%                                25350.000000
75%                                30234.000000
max                                45300.000000

       Day Ahead Price (EPEX half-hourly, local) - GB (£/MWh)  \
count                                        17561.000000
```

mean	72.433814
std	30.606106
min	-40.000000
25%	60.000000
50%	73.700000
75%	88.000000
max	506.560000

	Volume Requirements Forecast - DC-H - GB (MW) \
count	17561.000000
mean	1240.131826
std	125.800583
min	885.000000
25%	1159.000000
50%	1250.000000
75%	1337.000000
max	1643.000000

	Volume Requirements Forecast - DC-L - GB (MW) \
count	17561.000000
mean	1117.628381
std	120.056206
min	772.000000
25%	1042.000000
50%	1117.000000
75%	1203.000000
max	1433.000000

	Volume Requirements Forecast - DR-H - GB (MW) \
count	17561.0
mean	330.0
std	0.0
min	330.0
25%	330.0
50%	330.0
75%	330.0
max	330.0

	Volume Requirements Forecast - DR-L - GB (MW) \
count	17561.0
mean	330.0
std	0.0
min	330.0
25%	330.0
50%	330.0
75%	330.0
max	330.0

```
       Volume Requirements Forecast - DM-H - GB (MW)  \
count                                   17561.000000
mean                                      179.879278
std                                        37.430531
min                                       150.000000
25%                                       150.000000
50%                                       200.000000
75%                                       200.000000
max                                       480.000000

       Volume Requirements Forecast - DM-L - GB (MW)  \
count                                   17561.000000
mean                                      162.367177
std                                        18.539032
min                                       150.000000
25%                                       150.000000
50%                                       170.000000
75%                                       170.000000
max                                       330.000000

       Ancillary Volume Accepted - DC-H - GB (MW)  \
count                                 17561.000000
mean                                   1235.169466
std                                     129.791271
min                                     422.000000
25%                                    1157.000000
50%                                    1250.000000
75%                                    1342.000000
max                                    1643.000000

       Ancillary Volume Accepted - DC-L - GB (MW)  …  \
count                                 17561.000000  …
mean                                   1117.336883  …
std                                     122.815355  …
min                                     764.000000  …
25%                                    1040.000000  …
50%                                    1112.000000  …
75%                                    1205.000000  …
max                                    1439.000000  …

       Ancillary Volume Accepted - DM-H - GB (MW)  \
count                                 17561.000000
mean                                    186.358750
std                                      40.080663
min                                      22.000000
25%                                     152.000000
```

```
50%                                    200.000000
75%                                    214.000000
max                                    500.000000

       Ancillary Volume Accepted - DM-L - GB (MW)  \
count                                17561.000000
mean                                   158.595183
std                                     26.089485
min                                      0.000000
25%                                    150.000000
50%                                    166.000000
75%                                    170.000000
max                                    350.000000

       Ancillary Price - DC-H - GB (£/MW/h)  \
count                           17561.000000
mean                                1.916122
std                                 1.522260
min                                -0.260000
25%                                 0.940000
50%                                 1.400000
75%                                 2.490000
max                                10.000000

       Ancillary Price - DC-L - GB (£/MW/h)  \
count                           17561.000000
mean                                2.241599
std                                 2.742860
min                                -0.710000
25%                                 1.120000
50%                                 1.670000
75%                                 2.560000
max                                70.690000

       Ancillary Price - DR-H - GB (£/MW/h)  \
count                           17561.000000
mean                               -4.801852
std                                 3.843184
min                               -15.350000
25%                                -7.260000
50%                                -5.620000
75%                                -3.310000
max                                13.960000

       Ancillary Price - DR-L - GB (£/MW/h)  \
count                           17561.000000
mean                                8.182455
```

```
std                                3.390350
min                                1.630000
25%                                6.380000
50%                                7.520000
75%                                9.260000
max                               70.690000

       Ancillary Price - DM-H - GB (£/MW/h)  \
count                          17561.000000
mean                               0.561065
std                                1.944187
min                               -6.010000
25%                               -0.720000
50%                               -0.110000
75%                                1.250000
max                               13.860000

       Ancillary Price - DM-L - GB (£/MW/h)  \
count                          17561.000000
mean                               4.204893
std                                1.790867
min                              -20.000000
25%                                3.190000
50%                                3.960000
75%                                5.000000
max                               12.250000

       Day Ahead Price (N2EX, local) - GB (£/MWh)  \
count                                  17561.000000
mean                                      72.591005
std                                       29.993630
min                                      -24.680000
25%                                       60.080000
50%                                       73.870000
75%                                       88.010000
max                                      496.300000

       Day Ahead Price (EPEX, local) - GB (£/MWh)
count                                  17561.000000
mean                                      72.275981
std                                       28.687314
min                                      -21.300000
25%                                       60.200000
50%                                       73.200000
75%                                       87.000000
max                                      430.200000
```

```
[8 rows x 22 columns]
```

4.3 Data realigning

Realign the accepted ancillary volumes to enhance the prediction

[162]:
```
#lets shift the volumes accepted by one day so that we can use the lagged data␣
 ↪for prediction
#this is so that we can use ancillary volumes from the day before to predict␣
 ↪the price of ancillary data for the day
merged_data_2024['Ancillary Volume Accepted - DC-H - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DC-H - GB␣
 ↪(MW)'].shift(48)
merged_data_2024['Ancillary Volume Accepted - DC-L - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DC-L - GB␣
 ↪(MW)'].shift(48)
merged_data_2024['Ancillary Volume Accepted - DR-H - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DR-H - GB␣
 ↪(MW)'].shift(48)
merged_data_2024['Ancillary Volume Accepted - DR-L - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DR-L - GB␣
 ↪(MW)'].shift(48)
merged_data_2024['Ancillary Volume Accepted - DM-H - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DM-H - GB␣
 ↪(MW)'].shift(48)
merged_data_2024['Ancillary Volume Accepted - DM-L - GB␣
 ↪(MW)_shifted']=merged_data_2024['Ancillary Volume Accepted - DM-L - GB␣
 ↪(MW)'].shift(48)
```

[163]:
```
#add the missing data from the previous year

#31/12/2023 03:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][0:8]=1250
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][0:8]=991
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][0:8]=330
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][0:8]=149
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][0:8]=149
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][0:8]=149

#31/12/2023 07:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][8:
 ↪16]=1250
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][8:16]=961
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][8:16]=330
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][8:16]=210
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][8:16]=150
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][8:16]=150
```

```
#31/12/2023 11:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][16:
 ↪24]=1143
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][16:
 ↪24]=929
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][16:
 ↪24]=350
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][16:
 ↪24]=270
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][16:
 ↪24]=164
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][16:
 ↪24]=150

#31/12/2023 15:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][24:
 ↪32]=1019
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][24:
 ↪32]=911
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][24:
 ↪32]=350
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][24:
 ↪32]=210
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][24:
 ↪32]=150
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][24:
 ↪32]=150

#31/12/2023 19:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][32:
 ↪40]=1250
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][32:
 ↪40]=987
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][32:
 ↪40]=350
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][32:
 ↪40]=210
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][32:
 ↪40]=170
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][32:
 ↪40]=150

#31/12/2023 23:00
merged_data_2024['Ancillary Volume Accepted - DC-H - GB (MW)_shifted'][40:
 ↪48]=1250
```

```
merged_data_2024['Ancillary Volume Accepted - DC-L - GB (MW)_shifted'][40:
↪48]=1061
merged_data_2024['Ancillary Volume Accepted - DR-H - GB (MW)_shifted'][40:
↪48]=350
merged_data_2024['Ancillary Volume Accepted - DR-L - GB (MW)_shifted'][40:
↪48]=102
merged_data_2024['Ancillary Volume Accepted - DM-H - GB (MW)_shifted'][40:
↪48]=153
merged_data_2024['Ancillary Volume Accepted - DM-L - GB (MW)_shifted'][40:
↪48]=150
```

```
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][0:8]=1250
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][0:8]=991
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][0:8]=330
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][0:8]=149
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][0:8]=149
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:9:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][0:8]=149
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:12:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][8:16]=1250
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:13:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][8:16]=961
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:14:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][8:16]=330
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:15:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][8:16]=210
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:16:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][8:16]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:17:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][8:16]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:20:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][16:24]=1143
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:21:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][16:24]=929
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:22:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][16:24]=350
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:23:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][16:24]=270
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:24:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][16:24]=164
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:25:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][16:24]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:28:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][24:32]=1019
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:29:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][24:32]=911
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][24:32]=350
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:31:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][24:32]=210
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:32:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][24:32]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:33:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][24:32]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:36:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][32:40]=1250
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:37:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][32:40]=987
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:38:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][32:40]=350
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:39:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][32:40]=210
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:40:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][32:40]=170
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:41:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][32:40]=150
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:44:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-H - GB
(MW)_shifted'][40:48]=1250
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:45:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DC-L - GB
(MW)_shifted'][40:48]=1061
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:46:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-H - GB
(MW)_shifted'][40:48]=350
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:47:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DR-L - GB
(MW)_shifted'][40:48]=102
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:48:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

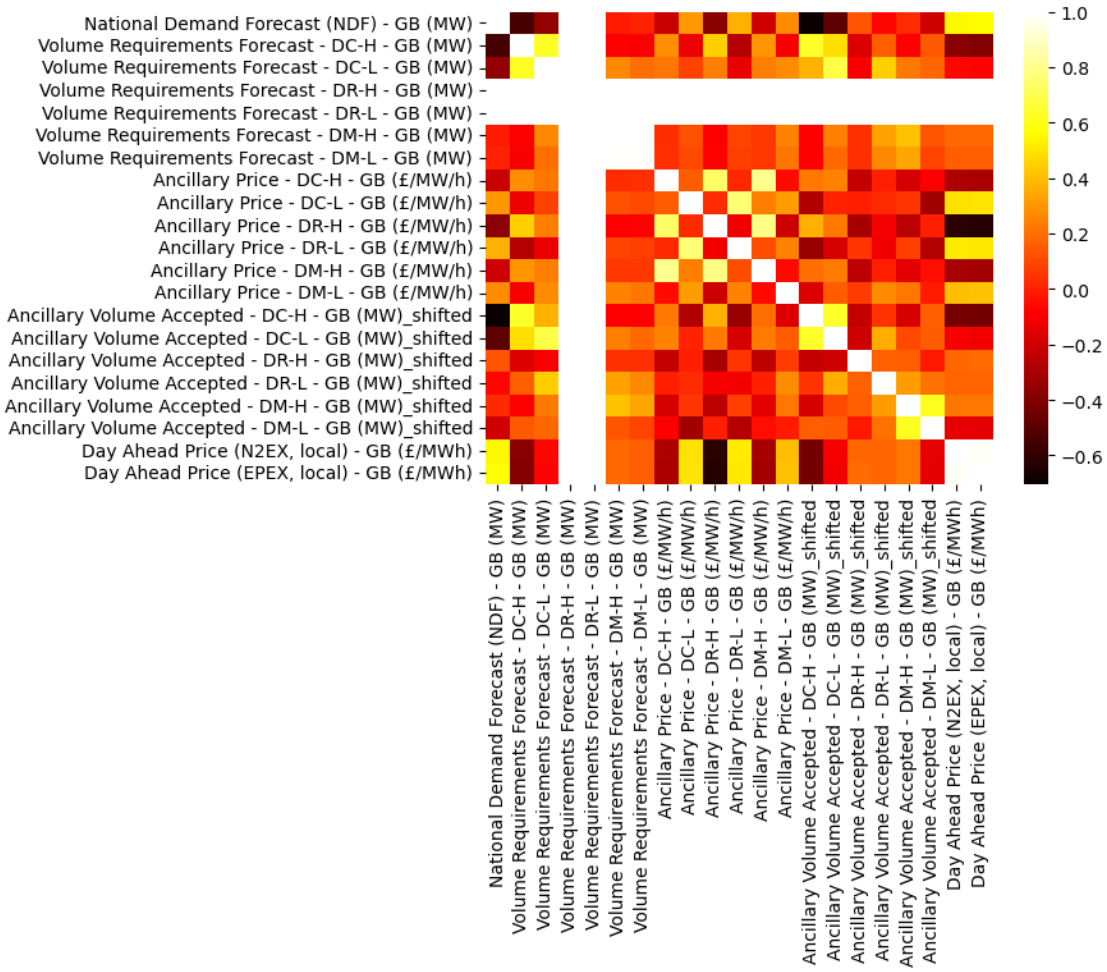
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-H - GB
(MW)_shifted'][40:48]=153
C:\Users\tumis\AppData\Local\Temp\ipykernel_10900\1132183928.py:49:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  merged_data_2024['Ancillary Volume Accepted - DM-L - GB
(MW)_shifted'][40:48]=150
```

4.4 Correlate the data and plot heat map to visualise the correlation between the data

[164]:
```
#calculate correlation using only non-lagged data
correlation_calculated=merged_data_2024[['National Demand Forecast (NDF) - GB
 ↪(MW)','Volume Requirements Forecast - DC-H - GB (MW)','Volume Requirements
 ↪Forecast - DC-L - GB (MW)',
                                         'Volume Requirements Forecast - DR-H -
 ↪GB (MW)','Volume Requirements Forecast - DR-L - GB (MW)','Volume
 ↪Requirements Forecast - DM-H - GB (MW)','Volume Requirements Forecast - DM-L
 ↪- GB (MW)',
                                         'Ancillary Price - DC-H - GB (£/MW/
 ↪h)','Ancillary Price - DC-L - GB (£/MW/h)','Ancillary Price - DR-H - GB (£/
 ↪MW/h)','Ancillary Price - DR-L - GB (£/MW/h)','Ancillary Price - DM-H - GB
 ↪(£/MW/h)','Ancillary Price - DM-L - GB (£/MW/h)',
                                         'Ancillary Volume Accepted - DC-H - GB
 ↪(MW)_shifted','Ancillary Volume Accepted - DC-L - GB
 ↪(MW)_shifted','Ancillary Volume Accepted - DR-H - GB
 ↪(MW)_shifted','Ancillary Volume Accepted - DR-L - GB
 ↪(MW)_shifted','Ancillary Volume Accepted - DM-H - GB
 ↪(MW)_shifted','Ancillary Volume Accepted - DM-L - GB (MW)_shifted',
                                         'Day Ahead Price (N2EX, local) - GB (£/
 ↪MWh)','Day Ahead Price (EPEX, local) - GB (£/MWh)']].corr()
```

[165]:
```
sns.heatmap(correlation_calculated,cmap='hot')
```

[165]:
```
<Axes: >
```



5 5. Machine learning

5.1 Split the data into dependent and independent features

[166]:
```
# X is the independent feature
X=merged_data_2024.drop(['Day Ahead Price (EPEX half-hourly, local) - GB (£/
↪MWh)','Ancillary Price - DC-H - GB (£/MW/h)','Ancillary Price - DC-L - GB (£/
↪MW/h)','Ancillary Price - DR-H - GB (£/MW/h)','Ancillary Price - DR-L - GB␣
↪(£/MW/h)','Ancillary Price - DM-H - GB (£/MW/h)','Ancillary Price - DM-L -␣
↪GB (£/MW/h)',
                        'Ancillary Volume Accepted - DC-H - GB (MW)','Ancillary␣
↪Volume Accepted - DC-L - GB (MW)','Ancillary Volume Accepted - DR-H - GB␣
↪(MW)','Ancillary Volume Accepted - DR-L - GB (MW)','Ancillary Volume␣
↪Accepted - DM-H - GB (MW)','Ancillary Volume Accepted - DM-L - GB (MW)',
                        ],axis=1)
X.set_index('GMT Time',inplace=True)
#y is the dependent feature i.e. the ancillary prices
```

```
y=merged_data_2024[['Ancillary Price - DC-H - GB (£/MW/h)','Ancillary Price -␣
↪DC-L - GB (£/MW/h)','Ancillary Price - DR-H - GB (£/MW/h)','Ancillary Price␣
↪- DR-L - GB (£/MW/h)','Ancillary Price - DM-H - GB (£/MW/h)','Ancillary␣
↪Price - DM-L - GB (£/MW/h)',
                    'Ancillary Volume Accepted - DC-H - GB␣
↪(MW)_shifted','Ancillary Volume Accepted - DC-L - GB␣
↪(MW)_shifted','Ancillary Volume Accepted - DR-H - GB␣
↪(MW)_shifted','Ancillary Volume Accepted - DR-L - GB␣
↪(MW)_shifted','Ancillary Volume Accepted - DM-H - GB␣
↪(MW)_shifted','Ancillary Volume Accepted - DM-L - GB (MW)_shifted']]
```

5.2 Split the data into training and testing data

[167]:
```
#Split the data into training and testing data using Train Test Split
#Size of training data is 75%, testing data is 25%

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪25,random_state=42,shuffle=False)
```

5.3 Feature selection based on correlation

[168]:
```
X_train.corr()
```

[168]:
```
                                                    National Demand Forecast
(NDF) - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
1.000000
Volume Requirements Forecast - DC-H - GB (MW)
-0.537244
Volume Requirements Forecast - DC-L - GB (MW)
-0.493633
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
-0.196820
Volume Requirements Forecast - DM-L - GB (MW)
-0.145465
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.437626
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.449112
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
-0.675295
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
-0.620359
```

```
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.125320
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
-0.189664
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
-0.140408
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
-0.128530

                                                    Volume Requirements Forecast
- DC-H - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
-0.537244
Volume Requirements Forecast - DC-H - GB (MW)
1.000000
Volume Requirements Forecast - DC-L - GB (MW)
0.741635
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.044780
Volume Requirements Forecast - DM-L - GB (MW)
0.019959
Day Ahead Price (N2EX, local) - GB (£/MWh)
-0.269428
Day Ahead Price (EPEX, local) - GB (£/MWh)
-0.282566
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.623926
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.626402
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
-0.168885
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.294253
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.015744
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.071086

                                                    Volume Requirements Forecast
- DC-L - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
-0.493633
Volume Requirements Forecast - DC-H - GB (MW)
```

```
0.741635
Volume Requirements Forecast - DC-L - GB (MW)
1.000000
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.262118
Volume Requirements Forecast - DM-L - GB (MW)
0.203844
Day Ahead Price (N2EX, local) - GB (£/MWh)
-0.060919
Day Ahead Price (EPEX, local) - GB (£/MWh)
-0.070391
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.518142
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.724200
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
-0.102093
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.435932
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.230030
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.185186

                                                    Volume Requirements Forecast
- DR-H - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
NaN
Volume Requirements Forecast - DC-H - GB (MW)
NaN
Volume Requirements Forecast - DC-L - GB (MW)
NaN
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
NaN
Volume Requirements Forecast - DM-L - GB (MW)
NaN
Day Ahead Price (N2EX, local) - GB (£/MWh)
NaN
Day Ahead Price (EPEX, local) - GB (£/MWh)
```

```
NaN
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
NaN

                                                    Volume Requirements Forecast
- DR-L - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
NaN
Volume Requirements Forecast - DC-H - GB (MW)
NaN
Volume Requirements Forecast - DC-L - GB (MW)
NaN
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
NaN
Volume Requirements Forecast - DM-L - GB (MW)
NaN
Day Ahead Price (N2EX, local) - GB (£/MWh)
NaN
Day Ahead Price (EPEX, local) - GB (£/MWh)
NaN
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
NaN
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
NaN
```

```
                                                        Volume Requirements Forecast
- DM-H - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
-0.196820
Volume Requirements Forecast - DC-H - GB (MW)
0.044780
Volume Requirements Forecast - DC-L - GB (MW)
0.262118
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
1.000000
Volume Requirements Forecast - DM-L - GB (MW)
0.987781
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.091740
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.085722
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.077582
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.256217
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.027014
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.273201
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.431662
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.322072

                                                        Volume Requirements Forecast
- DM-L - GB (MW)  \
National Demand Forecast (NDF) - GB (MW)
-0.145465
Volume Requirements Forecast - DC-H - GB (MW)
0.019959
Volume Requirements Forecast - DC-L - GB (MW)
0.203844
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.987781
```

```
Volume Requirements Forecast - DM-L - GB (MW)
1.000000
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.079641
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.074643
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.053173
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.199069
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.027226
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.221935
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.352491
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.263394

                                                    Day Ahead Price (N2EX,
local) - GB (£/MWh)  \
National Demand Forecast (NDF) - GB (MW)
0.437626
Volume Requirements Forecast - DC-H - GB (MW)
-0.269428
Volume Requirements Forecast - DC-L - GB (MW)
-0.060919
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.091740
Volume Requirements Forecast - DM-L - GB (MW)
0.079641
Day Ahead Price (N2EX, local) - GB (£/MWh)
1.000000
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.977148
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
-0.259688
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
-0.108774
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.209382
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.161020
```

```
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.144031
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.043619

                                                    Day Ahead Price (EPEX,
local) - GB (£/MWh)  \
National Demand Forecast (NDF) - GB (MW)
0.449112
Volume Requirements Forecast - DC-H - GB (MW)
-0.282566
Volume Requirements Forecast - DC-L - GB (MW)
-0.070391
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.085722
Volume Requirements Forecast - DM-L - GB (MW)
0.074643
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.977148
Day Ahead Price (EPEX, local) - GB (£/MWh)
1.000000
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
-0.271695
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
-0.122462
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.225801
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.151306
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.145238
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.041661

                                                    Ancillary Volume Accepted -
DC-H - GB (MW)_shifted  \
National Demand Forecast (NDF) - GB (MW)
-0.675295
Volume Requirements Forecast - DC-H - GB (MW)
0.623926
Volume Requirements Forecast - DC-L - GB (MW)
0.518142
Volume Requirements Forecast - DR-H - GB (MW)
```

```
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.077582
Volume Requirements Forecast - DM-L - GB (MW)
0.053173
Day Ahead Price (N2EX, local) - GB (£/MWh)
-0.259688
Day Ahead Price (EPEX, local) - GB (£/MWh)
-0.271695
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
1.000000
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.752172
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
-0.243687
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.145313
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
-0.059624
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.025713

                                                    Ancillary Volume Accepted -
DC-L - GB (MW)_shifted  \
National Demand Forecast (NDF) - GB (MW)
-0.620359
Volume Requirements Forecast - DC-H - GB (MW)
0.626402
Volume Requirements Forecast - DC-L - GB (MW)
0.724200
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.256217
Volume Requirements Forecast - DM-L - GB (MW)
0.199069
Day Ahead Price (N2EX, local) - GB (£/MWh)
-0.108774
Day Ahead Price (EPEX, local) - GB (£/MWh)
-0.122462
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.752172
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
```

```
1.000000
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
-0.195718
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.324944
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.141400
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.146109

                                                    Ancillary Volume Accepted -
DR-H - GB (MW)_shifted  \
National Demand Forecast (NDF) - GB (MW)
0.125320
Volume Requirements Forecast - DC-H - GB (MW)
-0.168885
Volume Requirements Forecast - DC-L - GB (MW)
-0.102093
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.027014
Volume Requirements Forecast - DM-L - GB (MW)
0.027226
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.209382
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.225801
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
-0.243687
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
-0.195718
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
1.000000
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.152924
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.075843
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
-0.056614

                                                    Ancillary Volume Accepted -
DR-L - GB (MW)_shifted  \
National Demand Forecast (NDF) - GB (MW)
-0.189664
```

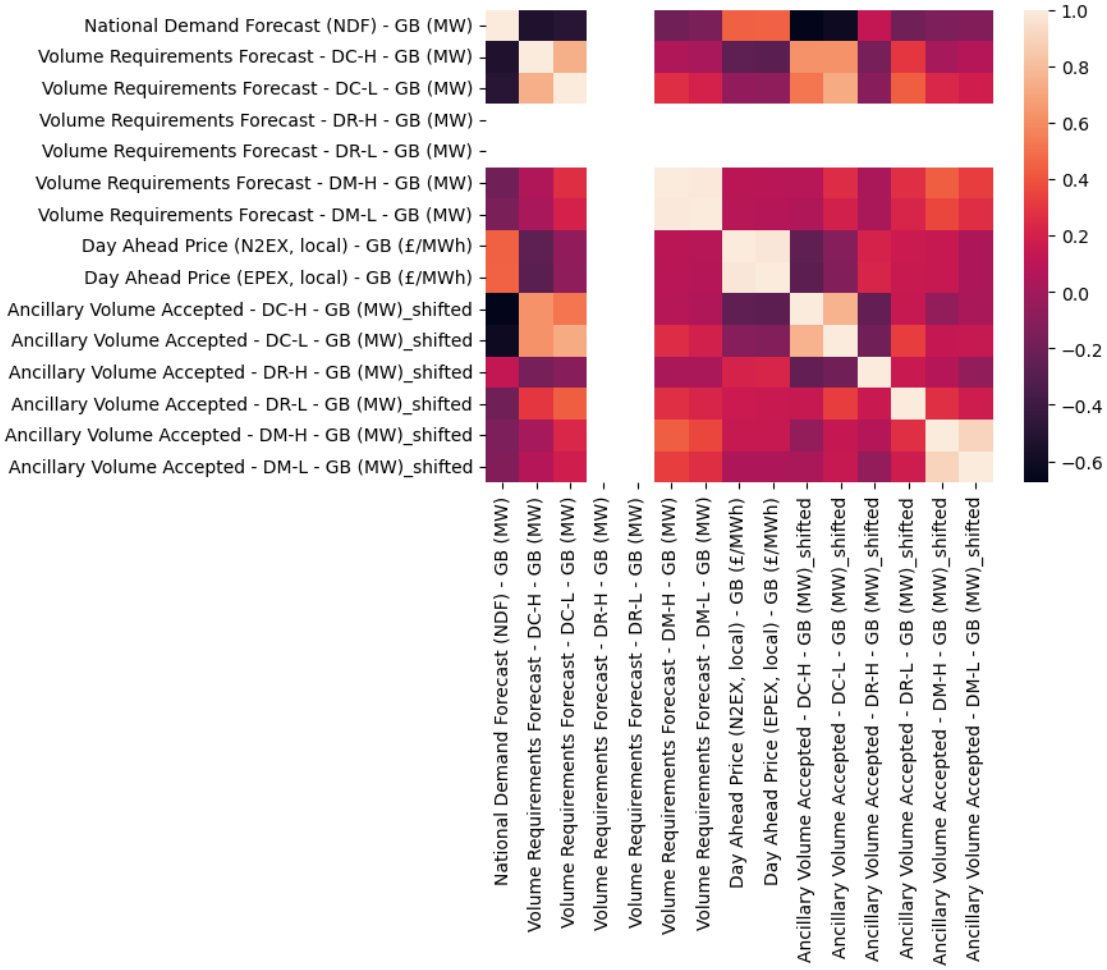
```
Volume Requirements Forecast - DC-H - GB (MW)
0.294253
Volume Requirements Forecast - DC-L - GB (MW)
0.435932
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.273201
Volume Requirements Forecast - DM-L - GB (MW)
0.221935
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.161020
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.151306
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.145313
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.324944
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.152924
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
1.000000
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.267680
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.171227

                                                    Ancillary Volume Accepted -
DM-H - GB (MW)_shifted  \
National Demand Forecast (NDF) - GB (MW)
-0.140408
Volume Requirements Forecast - DC-H - GB (MW)
0.015744
Volume Requirements Forecast - DC-L - GB (MW)
0.230030
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.431662
Volume Requirements Forecast - DM-L - GB (MW)
0.352491
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.144031
```

Day Ahead Price (EPEX, local) - GB (£/MWh)
0.145238
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
-0.059624
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.141400
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
0.075843
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.267680
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
1.000000
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
0.895505

Ancillary Volume Accepted -
DM-L - GB (MW)_shifted
National Demand Forecast (NDF) - GB (MW)
-0.128530
Volume Requirements Forecast - DC-H - GB (MW)
0.071086
Volume Requirements Forecast - DC-L - GB (MW)
0.185186
Volume Requirements Forecast - DR-H - GB (MW)
NaN
Volume Requirements Forecast - DR-L - GB (MW)
NaN
Volume Requirements Forecast - DM-H - GB (MW)
0.322072
Volume Requirements Forecast - DM-L - GB (MW)
0.263394
Day Ahead Price (N2EX, local) - GB (£/MWh)
0.043619
Day Ahead Price (EPEX, local) - GB (£/MWh)
0.041661
Ancillary Volume Accepted - DC-H - GB (MW)_shifted
0.025713
Ancillary Volume Accepted - DC-L - GB (MW)_shifted
0.146109
Ancillary Volume Accepted - DR-H - GB (MW)_shifted
-0.056614
Ancillary Volume Accepted - DR-L - GB (MW)_shifted
0.171227
Ancillary Volume Accepted - DM-H - GB (MW)_shifted
0.895505
Ancillary Volume Accepted - DM-L - GB (MW)_shifted
1.000000

[169]:
```
# Plot heat map to correlate the training data
sns.heatmap(X_train.corr())
```

[169]: <Axes: >



5.4 Drop the colinear features

[170]:
```
def correlation(dataset, threshold):
    col_corr=set()
    corr_matrix=dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j])>threshold:
                colname=corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

[171]:
```
#threshold--Domain expertise
#collect the independent features which are strongly correlated to each other␣
 ↪and have a correlation of 0.65
corr_features=correlation(X_train,0.85)
```

[172]:
```
## drop the features where the correlation is more than 0.65

X_train.drop(corr_features,axis=1,inplace=True)
X_test.drop(corr_features,axis=1,inplace=True)
X_train.shape, X_test.shape
```

[172]:
```
((13170, 12), (4391, 12))
```

5.5 Feature scaling or standardization

[173]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

5.6 Import the regression models

[174]:
```
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
 ↪GradientBoostingRegressor
from sklearn.multioutput import MultiOutputRegressor
#from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

[175]:
```
#Create a function to evaluate model
def evaluate_model(true,predicted):
    mae=mean_absolute_error(true,predicted)
    mse=mean_squared_error(true,predicted)
    rmse=np.sqrt(mean_squared_error(true,predicted))
    r2_square=r2_score(true,predicted)

    return mae, mse, r2_square
```

[176]:
```
##Beginning model training

## Beginning Model Training
models = {
    #"Linear Regression": MultiOutputRegressor(LinearRegression()),
    #"Lasso": MultiOutputRegressor(Lasso()),
    #"Ridge": MultiOutputRegressor(Ridge()),
    "K-Neighbors Regressor": MultiOutputRegressor(KNeighborsRegressor()),
```

```
    "Decision Tree": MultiOutputRegressor(DecisionTreeRegressor()),
    "Random Forest Regressor": MultiOutputRegressor(RandomForestRegressor()),
    "AdaBoost Regressor": MultiOutputRegressor(AdaBoostRegressor()),
    "Gradient Boost Regressor":␣
 ↪MultiOutputRegressor(GradientBoostingRegressor())

}

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2 =␣
 ↪evaluate_model(y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(y_test,␣
 ↪y_test_pred)


    print(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))

    print('='*35)
    print('\n')
```

```
K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 138.2296
- Mean Absolute Error: 5.0983
- R2 Score: 0.7164
----------------------------------
```

```
Model performance for Test set
- Root Mean Squared Error: 1241.6195
- Mean Absolute Error: 15.9846
- R2 Score: 0.0456
==================================


Decision Tree
Model performance for Training set
- Root Mean Squared Error: 0.0000
- Mean Absolute Error: 0.0000
- R2 Score: 1.0000
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 183.2062
- Mean Absolute Error: 2.6199
- R2 Score: 0.3632
==================================


Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 0.0129
- Mean Absolute Error: 0.0341
- R2 Score: 0.9974
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 159.9938
- Mean Absolute Error: 2.3688
- R2 Score: 0.4794
==================================


AdaBoost Regressor
Model performance for Training set
- Root Mean Squared Error: 21.1899
- Mean Absolute Error: 2.4608
- R2 Score: 0.6122
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 193.5333
- Mean Absolute Error: 4.0933
- R2 Score: 0.4734
==================================


Gradient Boost Regressor
Model performance for Training set
```

```
- Root Mean Squared Error: 1.1277
- Mean Absolute Error: 0.5605
- R2 Score: 0.8143
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 159.3040
- Mean Absolute Error: 2.4119
- R2 Score: 0.5030
==================================
```

Observation: Decision tree and random forest regressor are the best performing models. However, decision tree may be overfitting. Perform hyperparameter tuning on those models.

6 6. Hyperparameter tuning

[177]:
```
#Initialize few parameter for Hyperparamter tuning
gb_params={"learning_rate":[0.1,0.5,2.0,5.0,9.0],
            "n_estimators": [100, 200, 500, 1000],
            "loss":['squared_error', 'absolute_error', 'huber', 'quantile'],
            "criterion":['squared_error', 'friedman_mse', 'absolute_error'],
            "min_samples_split":[2,8,15,20,25],
            "min_samples_leaf":[1,5,10,15,20]}
ab_params= {"n_estimators": [100, 200, 500, 1000],
            "learning_rate":[0.1,0.5,2.0,5.0,9.0],
            "loss":['linear', 'square', 'exponential']}
rf_params = {"max_depth": [5, 8, 15, None, 10],
             "max_features": [5, 7, "auto", 8],
             "min_samples_split": [2, 8, 15, 20],
             "n_estimators": [100, 200, 500, 1000]}
```

[188]:
```
# Models list for Hyperparameter tuning
randomcv_models = [#('GB', GradientBoostingRegressor(), gb_params),
                   ("RF", RandomForestRegressor(), rf_params),
                   #("AB", AdaBoostRegressor(), ab_params)
                   ]
```

[ ]:
```
##Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV

model_param = {}
for name, model, params in randomcv_models:
    random = RandomizedSearchCV(estimator=model,
                                param_distributions=params,
```

```
                                    n_iter=100,
                                    cv=3,
                                    verbose=2,
                                    n_jobs=-1)
    random.fit(X_train, y_train)
    model_param[name] = random.best_params_

for model_name in model_param:
    print(f"---------------- Best Params for {model_name} -------------------")
    print(model_param[model_name])
```

[ ]:
```
## Retraining the models with best parameters
models = {
    "Random Forest Regressor":␣
 ↪MultiOutputRegressor(RandomForestRegressor(n_estimators=100,␣
 ↪min_samples_split=2, max_features=8, max_depth=None,
                                                n_jobs=-1)),
     #"Decision Tree":␣
 ↪MultiOutputRegressor(DecisionTreeRegressor(splitter='random',␣
 ↪min_weight_fraction_leaf=0.0, min_samples_split=8, min_samples_leaf=1,
                                                                    ␣
 ↪max_features='log2', max_depth=None, criterion='squared_error'))

}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    model_train_mae , model_train_rmse, model_train_r2 =␣
 ↪evaluate_model(y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(y_test,␣
 ↪y_test_pred)

    print(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')
```

```
    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))

    print('='*35)
    print('\n')
```

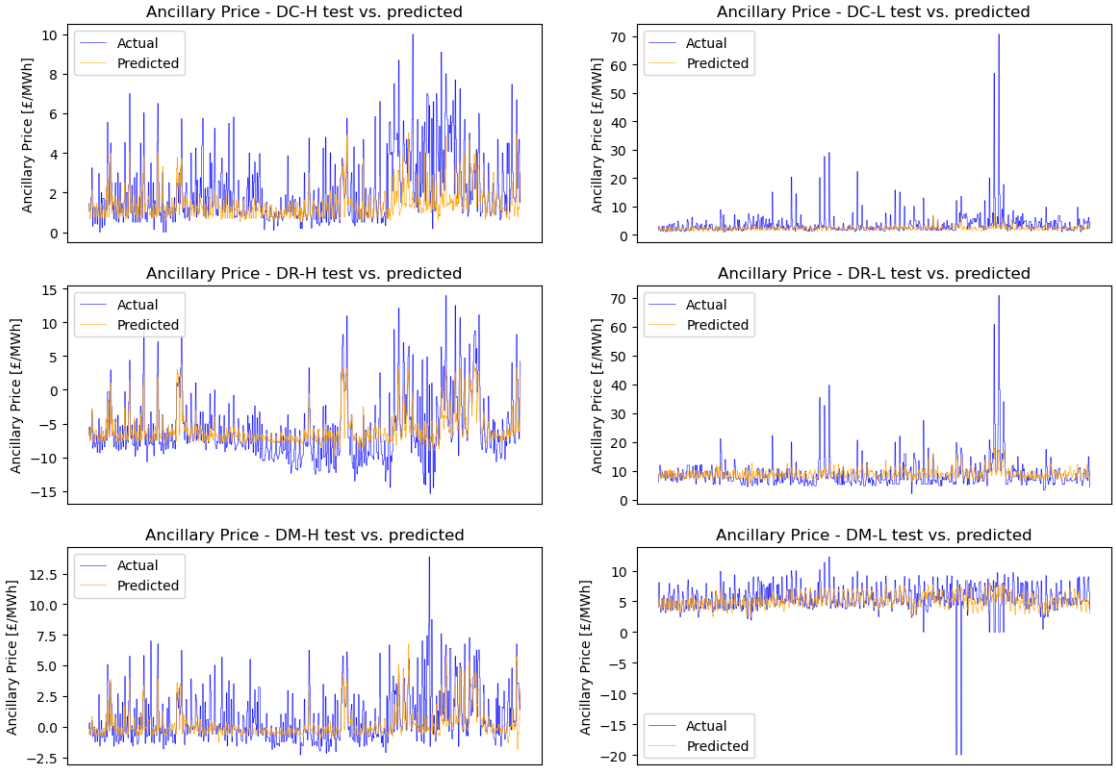
[ ]:
```
'models = {\n    "Random Forest Regressor":
MultiOutputRegressor(RandomForestRegressor(n_estimators=100,
min_samples_split=2, max_features=8, max_depth=None, \n
n_jobs=-1)),\n     "Decision Tree":
MultiOutputRegressor(DecisionTreeRegressor(splitter=\'random\',
min_weight_fraction_leaf=0.0, min_samples_split=8, min_samples_leaf=1, \n
max_features=\'log2\', max_depth=None, criterion=\'squared_error\'))\n
\n}\nfor i in range(len(list(models))):\n    model = list(models.values())[i]\n
model.fit(X_train, y_train) # Train model\n\n    # Make predictions\n
y_train_pred = model.predict(X_train)\n    y_test_pred =
model.predict(X_test)\n\n    model_train_mae , model_train_rmse, model_train_r2
= evaluate_model(y_train, y_train_pred)\n\n    model_test_mae , model_test_rmse,
model_test_r2 = evaluate_model(y_test, y_test_pred)\n    \n
print(list(models.keys())[i])\n    \n    print(\'Model performance for Training
set\')\n    print("- Root Mean Squared Error:
{:.4f}".format(model_train_rmse))\n    print("- Mean Absolute Error:
{:.4f}".format(model_train_mae))\n    print("- R2 Score:
{:.4f}".format(model_train_r2))\n\n
print(\'----------------------------------\')\n    \n    print(\'Model
performance for Test set\')\n    print("- Root Mean Squared Error:
{:.4f}".format(model_test_rmse))\n    print("- Mean Absolute Error:
{:.4f}".format(model_test_mae))\n    print("- R2 Score:
{:.4f}".format(model_test_r2))\n    \n    print(\'=\'*35)\n    print(\'\n\')'
```

[ ]:
```
y1=y_test
y2=pd.DataFrame(y_test_pred)
```

[ ]:
```
plt.figure(figsize=(14,10))
plt.subplot(3,2,1)
plt.plot(y_test.index,y1['Ancillary Price - DC-H - GB (£/MW/
 ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[0],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DC-H test vs. predicted')
plt.legend(loc="upper left")
plt.xticks([])

plt.subplot(3,2,2)
```

```
plt.plot(y_test.index,y1['Ancillary Price - DC-L - GB (£/MW/
  ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[1],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DC-L test vs. predicted')
plt.legend(loc="upper left")
plt.xticks([])

plt.subplot(3,2,3)
plt.plot(y_test.index,y1['Ancillary Price - DR-H - GB (£/MW/
  ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[2],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DR-H test vs. predicted')
plt.legend(loc="upper left")
plt.xticks([])

plt.subplot(3,2,4)
plt.plot(y_test.index,y1['Ancillary Price - DR-L - GB (£/MW/
  ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[3],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DR-L test vs. predicted')
plt.legend(loc="upper left")
plt.xticks([])

plt.subplot(3,2,5)
plt.plot(y_test.index,y1['Ancillary Price - DM-H - GB (£/MW/
  ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[4],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DM-H test vs. predicted')
plt.legend(loc="upper left")
plt.xticks([])

plt.subplot(3,2,6)
plt.plot(y_test.index,y1['Ancillary Price - DM-L - GB (£/MW/
  ↪h)'],label='Actual',color='blue',linewidth=0.4)
plt.plot(y_test.index,y2[5],label='Predicted',color='orange',linewidth=0.4)
plt.ylabel('Ancillary Price [£/MWh]')
plt.title('Ancillary Price - DM-L test vs. predicted')
plt.legend(loc="lower left")
plt.xticks([])
```

[ ]: ([], [])


Ancillary Price - DC-H test vs. predicted
Ancillary Price - DC-L test vs. predicted
Ancillary Price - DR-H test vs. predicted
Ancillary Price - DR-L test vs. predicted
Ancillary Price - DM-H test vs. predicted
Ancillary Price - DM-L test vs. predicted
Actual
Predicted
Ancillary Price [£/MWh]

[ ]:

In [1]:
```
import pandas as pd

# Load main dataset
file_path = r'C:\Users\lezha\OneDrive\Desktop\Imperial\Grid Modelling\2024\Ancillary Volumes & Prices (4H).csv'
df = pd.read_csv(file_path)

# Convert to datetime with UK timezone (this ensures pandas recognises BST/GMT correctly)
df["GMT Time"] = pd.to_datetime(df["GMT Time"], format="%d/%m/%Y %H:%M")
# df.set_index("GMT Time", inplace=True)
```

In [2]:
```
df.head()
```

Out[2]:

	GMT Time	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Volume Requirements Forecast - DR-H - GB (MW)	Volume Requirements Forecast - DR-L - GB (MW)	Volume Requirements Forecast - DM-H - GB (MW)	Volume Requirements Forecast - DM-L - GB (MW)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Ancilla Volur Accept - DR-H GB (MW
0	2024-01-01 03:00:00	1250.0	933.0	330	330	150	150	1250	1039	3
1	2024-01-01 07:00:00	1250.0	983.0	330	330	150	150	1250	1004	3
2	2024-01-01 11:00:00	1250.0	956.0	330	330	150	150	1160	974	3
3	2024-01-01 15:00:00	1137.0	924.0	330	330	150	150	997	960	3
4	2024-01-01 19:00:00	1007.0	912.0	330	330	150	150	1087	1002	3

In [3]:
```
#insert data for DC-H for 29/10/2024
'''
df.iloc[1812:1813,1:2]=int(1185)
df.iloc[1813:1814,1:2]=int(1230)
df.iloc[1814:1815,1:2]=int(1108)
df.iloc[1815:1816,1:2]=int(1060)
df.iloc[1816:1817,1:2]=int(1056)
df.iloc[1817:1818,1:2]=int(1090)

#insert data for DC-H for 30/10/2024
df.iloc[1818:1819,1:2]=int(1172)
df.iloc[1819:1820,1:2]=int(1200)
df.iloc[1820:1821,1:2]=int(1095)
df.iloc[1821:1822,1:2]=int(1060)
df.iloc[1822:1823,1:2]=int(1052)
df.iloc[1823:1824,1:2]=int(1097)

#insert data for DC-H for 31/10/2024
df.iloc[1824:1825,1:2]=int(1255)
df.iloc[1825:1826,1:2]=int(1337)
df.iloc[1826:1827,1:2]=int(1156)
df.iloc[1827:1828,1:2]=int(1109)
df.iloc[1828:1829,1:2]=int(1087)
df.iloc[1829:1830,1:2]=int(1138)

#insert data for DC-L for 29/10/2024
df.iloc[1812:1813,2:3]=int(1146)
df.iloc[1813:1814,2:3]=int(1162)
df.iloc[1814:1815,2:3]=int(1111)
df.iloc[1815:1816,2:3]=int(1030)
df.iloc[1816:1817,2:3]=int(1013)
df.iloc[1817:1818,2:3]=int(1087)

#insert data for DC-L for 30/10/2024
df.iloc[1818:1819,2:3]=int(1145)
df.iloc[1819:1820,2:3]=int(1173)
df.iloc[1820:1821,2:3]=int(1103)
df.iloc[1821:1822,2:3]=int(1048)
df.iloc[1822:1823,2:3]=int(1025)
df.iloc[1823:1824,2:3]=int(1123)

#insert data for DC-L for 31/10/2024
df.iloc[1824:1825,2:3]=int(1258)
df.iloc[1825:1826,2:3]=int(1326)
df.iloc[1826:1827,2:3]=int(1272)
```

```
df.iloc[1827:1828,2:3]=int(1199)
df.iloc[1828:1829,2:3]=int(1172)
df.iloc[1829:1830,2:3]=int(1238)
'''
```

Out[3]:
```
'\ndf.iloc[1812:1813,1:2]=int(1185)\ndf.iloc[1813:1814,1:2]=int(1230)\ndf.iloc[1814:1815,1:2]=int(1108)\ndf.ilo
c[1815:1816,1:2]=int(1060)\ndf.iloc[1816:1817,1:2]=int(1056)\ndf.iloc[1817:1818,1:2]=int(1090)\n\n#insert data
for DC-H for 30/10/2024\ndf.iloc[1818:1819,1:2]=int(1172)\ndf.iloc[1819:1820,1:2]=int(1200)\ndf.iloc[1820:1821,
1:2]=int(1095)\ndf.iloc[1821:1822,1:2]=int(1060)\ndf.iloc[1822:1823,1:2]=int(1052)\ndf.iloc[1823:1824,1:2]=int(
1097)\n\n#insert data for DC-H for 31/10/2024\ndf.iloc[1824:1825,1:2]=int(1255)\ndf.iloc[1825:1826,1:2]=int(133
7)\ndf.iloc[1826:1827,1:2]=int(1156)\ndf.iloc[1827:1828,1:2]=int(1109)\ndf.iloc[1828:1829,1:2]=int(1087)\ndf.il
oc[1829:1830,1:2]=int(1138)\n\n#insert data for DC-L for 29/10/2024\ndf.iloc[1812:1813,2:3]=int(1146)\ndf.iloc[
1813:1814,2:3]=int(1162)\ndf.iloc[1814:1815,2:3]=int(1111)\ndf.iloc[1815:1816,2:3]=int(1030)\ndf.iloc[1816:1817
,2:3]=int(1013)\ndf.iloc[1817:1818,2:3]=int(1087)\n\n#insert data for DC-L for 30/10/2024\ndf.iloc[1818:1819,2:
3]=int(1145)\ndf.iloc[1819:1820,2:3]=int(1173)\ndf.iloc[1820:1821,2:3]=int(1103)\ndf.iloc[1821:1822,2:3]=int(10
48)\ndf.iloc[1822:1823,2:3]=int(1025)\ndf.iloc[1823:1824,2:3]=int(1123)\n\n#insert data for DC-L for 31/10/2024
\ndf.iloc[1824:1825,2:3]=int(1258)\ndf.iloc[1825:1826,2:3]=int(1326)\ndf.iloc[1826:1827,2:3]=int(1272)\ndf.iloc
[1827:1828,2:3]=int(1199)\ndf.iloc[1828:1829,2:3]=int(1172)\ndf.iloc[1829:1830,2:3]=int(1238)\n'
```

In [4]:
```
# Define delay
delta_t = 6  # 6 EFA blocks

# Shift the volume accepted variables 1 day, or 6 EFA blocks, 1nto the future (as these are lagged)
ancillary_volume_cols = [
    "Ancillary Volume Accepted - DC-H - GB (MW)",
    "Ancillary Volume Accepted - DC-L - GB (MW)",
    "Ancillary Volume Accepted - DR-H - GB (MW)",
    "Ancillary Volume Accepted - DR-L - GB (MW)",
    "Ancillary Volume Accepted - DM-H - GB (MW)",
    "Ancillary Volume Accepted - DM-L - GB (MW)",
]

# Create a copy of the original DataFrame
df_shifted = df.copy()

# Apply the shift to the specified columns
df_shifted[ancillary_volume_cols] = df_shifted[ancillary_volume_cols].shift(delta_t)
```

In [5]:
```
df_shifted.head(10)
```

Out[5]:

	GMT Time	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Volume Requirements Forecast - DR-H - GB (MW)	Volume Requirements Forecast - DR-L - GB (MW)	Volume Requirements Forecast - DM-H - GB (MW)	Volume Requirements Forecast - DM-L - GB (MW)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Ancilla Volun Accept - DR-H GB (MW
0	2024-01-01 03:00:00	1250.0	933.0	330	330	150	150	NaN	NaN	Na
1	2024-01-01 07:00:00	1250.0	983.0	330	330	150	150	NaN	NaN	Na
2	2024-01-01 11:00:00	1250.0	956.0	330	330	150	150	NaN	NaN	Na
3	2024-01-01 15:00:00	1137.0	924.0	330	330	150	150	NaN	NaN	Na
4	2024-01-01 19:00:00	1007.0	912.0	330	330	150	150	NaN	NaN	Na
5	2024-01-01 23:00:00	1051.0	934.0	330	330	150	150	NaN	NaN	Na
6	2024-01-02 03:00:00	1250.0	935.0	330	330	150	150	1250.0	1039.0	330
7	2024-01-02 07:00:00	1250.0	967.0	330	330	150	150	1250.0	1004.0	326
8	2024-01-02 11:00:00	1250.0	947.0	330	330	150	150	1160.0	974.0	350
9	2024-01-02 15:00:00	996.0	870.0	330	330	150	150	997.0	960.0	350

In [6]:
```
import pandas as pd
```

```
# Load Generation by Fuel dataset
file_path_2 = r'C:\Users\lezha\OneDrive\Desktop\Imperial\Grid Modelling\2024\Generation by Fuel.csv'
df2 = pd.read_csv(file_path_2)

# Convert to datetime with UK timezone (this ensures pandas recognises BST/GMT correctly)
df2["GMT Time"] = pd.to_datetime(df2["GMT Time"], format="%d/%m/%Y %H:%M")
```

In [7]:
```
df2.head()
```

Out[7]:

	GMT Time	Generation - Fuel - Biomass - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Coal - GB (MW)	Generation - Fuel - Hydro - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - Oil - GB (MW)	Generation - Fuel - Pumped Storage - GB (MW)	Generation - Fuel - Wind - GB (MW)	Gene Intercon - G
0	2024-01-01 00:00:00	986	2856	230	535	4673	0	0	-812	10402	
1	2024-01-01 00:30:00	1002	3025	231	534	4671	0	0	-898	10845	
2	2024-01-01 01:00:00	992	3015	229	533	4673	0	0	-1458	11218	
3	2024-01-01 01:30:00	982	2989	230	531	4674	0	0	-1566	11223	
4	2024-01-01 02:00:00	1009	3431	230	735	4672	48	0	-1460	11234	

In [8]:
```
# If using EFA block granularity, aggregate finer data into 4 hour periods (but not always 4 hours due to time
# Look at mean, max-min, std. "catastrophic cancellation" error computing skewness and kurtosis so don't look a
```

In [9]:
```
# code for summary stats is not working, return to this later
```

In [10]:
```
import pandas as pd

# Load MEL below PN dataset
file_path_3 = r'C:\Users\lezha\OneDrive\Desktop\Imperial\Grid Modelling\2024\MEL below PN.csv'
df3 = pd.read_csv(file_path_3)

# Convert to datetime with UK timezone
df3["GMT Time"] = pd.to_datetime(df3["GMT Time"], format="%d/%m/%Y %H:%M")
```

In [11]:
```
df3.head()
```

Out[11]:

	GMT Time	MEL Below PN - Fuel - CCGT - GB (MW)	MEL Below PN - Fuel - Nuclear - GB (MW)	MEL Below PN - Fuel - Coal - GB (MW)	MEL Below PN - Fuel - Biomass - GB (MW)	MEL Below PN - Fuel - Pumped Storage - GB (MW)	MEL Below PN - Fuel - Interconnectors - GB (MW)
0	2024-01-01 00:00:00	-1.0	0.0	0.0	-215.0	0.0	0
1	2024-01-01 00:30:00	-1.0	0.0	0.0	-215.0	0.0	0
2	2024-01-01 01:00:00	-1.0	0.0	0.0	-215.0	0.0	0
3	2024-01-01 01:30:00	-1.0	0.0	0.0	-210.8	0.0	0
4	2024-01-01 02:00:00	-1.0	0.0	0.0	-206.0	0.0	0

In [12]:
```
import pandas as pd

# Load the dataset
file_path_4 = r'C:\Users\lezha\OneDrive\Desktop\Imperial\Grid Modelling\2024\Prices & Forecasts (HH).csv'
df4 = pd.read_csv(file_path_4)

# Ensure 'GMT Time' is timezone-aware in GMT
df4["GMT Time"] = pd.to_datetime(df4["GMT Time"], format="%d/%m/%Y %H:%M")

# Shift the dataset 48 half hours into the future as the indicators are lagged
# Shift all columns (except 'GMT Time') 48 rows forward
```

```
df4_shifted = df4.copy()
for column in df4.columns:
    if column != "GMT Time":
        df4_shifted[column] = df4[column].shift(48)
```

In [13]:
```
df4.shape
```

Out[13]: (17568, 3)

In [14]:
```
import pandas as pd

# Load Prices & Forecasts dataset
file_path_5 = r'C:\Users\lezha\OneDrive\Desktop\Imperial\Grid Modelling\2024\Day-Ahead Price (1H).csv'
df5 = pd.read_csv(file_path_5)

# Convert to datetime with UK timezone (this ensures pandas recognises BST/GMT correctly)
df5["GMT Time"] = pd.to_datetime(df5["GMT Time"], format="%d/%m/%Y %H:%M")


# Ensure 'GMT Time' is timezone-aware in GMT
df5["GMT Time"] = pd.to_datetime(df5["GMT Time"])
```

In [15]:
```
df5.head()
```

Out[15]:

	GMT Time	Day Ahead Price (N2EX, local) - GB (£/MWh)	Day Ahead Price (EPEX, local) - GB (£/MWh)
0	2024-01-01 00:00:00	46.60	50.0
1	2024-01-01 01:00:00	43.42	47.0
2	2024-01-01 02:00:00	18.03	33.0
3	2024-01-01 03:00:00	0.15	25.8
4	2024-01-01 04:00:00	-0.29	20.6

In [16]:
```
df5.shape
```

Out[16]: (8784, 3)

Feature Selection

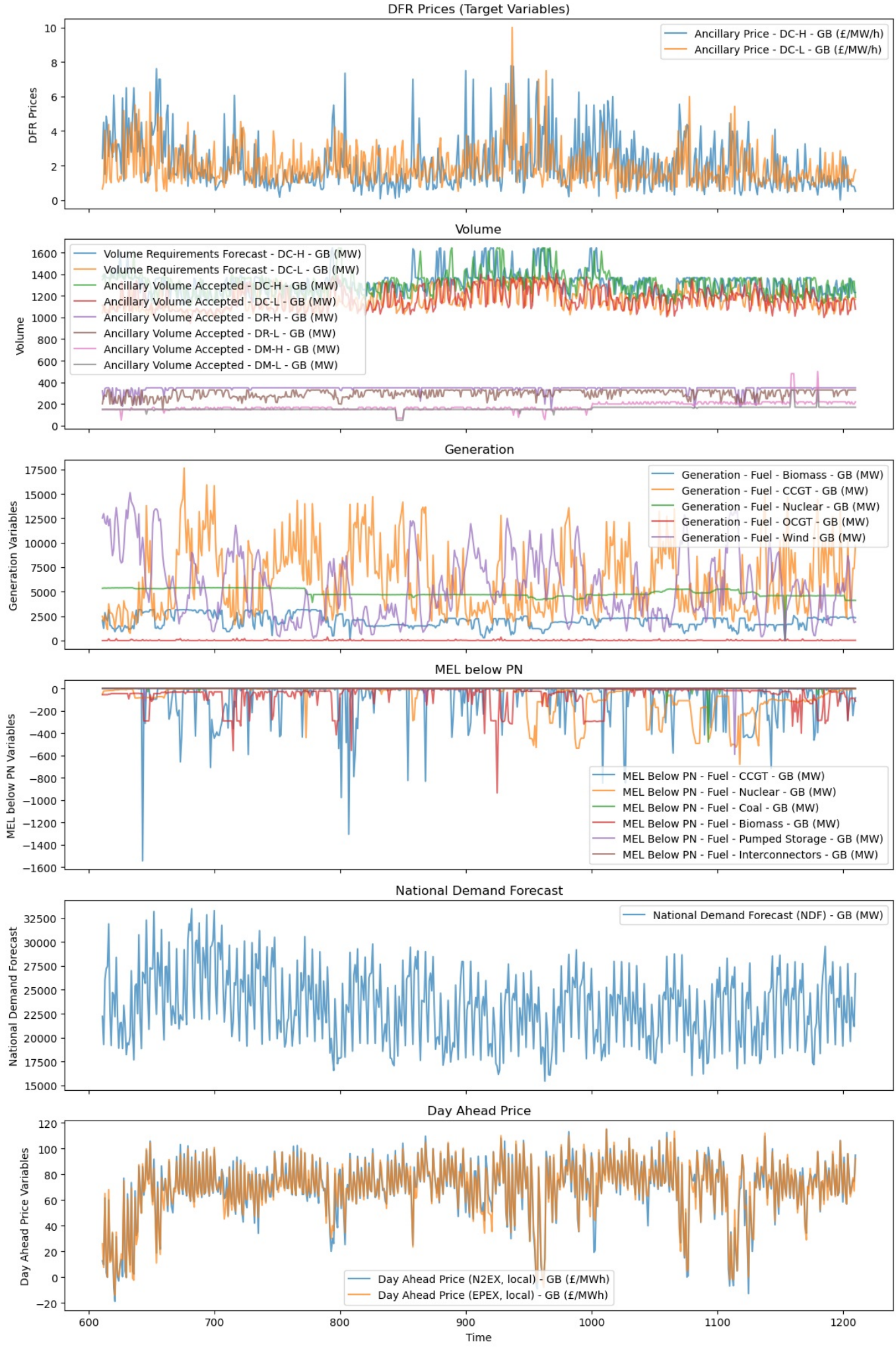
We will first plot the time series to infer any simple relationships.

In [18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

'''
df_combined = df2
df_combined = df_combined.merge(df_shifted, on='GMT Time', how='left', suffixes=('', '_df'))
df_combined = df_combined.merge(df3, on='GMT Time', how='left', suffixes=('', '_df3'))
df_combined = df_combined.merge(df4, on='GMT Time', how='left', suffixes=('', '_df4'))
df_combined = df_combined.merge(df5, on='GMT Time', how='left', suffixes=('', '_df5'))


'''
df_combined = df_shifted
df_combined = df_combined.merge(df2, on='GMT Time', how='left', suffixes=('', '_df2'))
df_combined = df_combined.merge(df3, on='GMT Time', how='left', suffixes=('', '_df3'))
df_combined = df_combined.merge(df4_shifted, on='GMT Time', how='left', suffixes=('', '_df4'))
df_combined = df_combined.merge(df5, on='GMT Time', how='left', suffixes=('', '_df5'))




# Remove the first 6 rows as they are NaNs (2024-01-01)
# Remove the next 5 because the EFA day starts at 11pm
shift = 11
df_combined = df_combined.iloc[shift:]


# Remove the first 48 rows as they are NaNs (2024-01-01)
# Remove the next 44 rows and the final row because the EFA day starts at 11pm
# df_combined = df_combined.iloc[94:-1]


# Forward fill rather than linear or cubic interpolation so our model cannot access future data.
# Also, EFA prices are fixed over each 4 hour block, so forward fill makes the most sense.
df_combined = df_combined.ffill()
```

```python
# Check shape
print(df_combined.shape)

# Select a subset of observations

df_subset = df_combined.iloc[600:1200]


# Define columns for plotting
price_cols = [
    "Ancillary Price - DC-H - GB (£/MW/h)",
    "Ancillary Price - DC-L - GB (£/MW/h)",
    # "Ancillary Price - DR-H - GB (£/MW/h)",
    # "Ancillary Price - DR-L - GB (£/MW/h)",
    # "Ancillary Price - DM-H - GB (£/MW/h)",
    # "Ancillary Price - DM-L - GB (£/MW/h)",
]

remaining_price_volume_cols = [
    "Volume Requirements Forecast - DC-H - GB (MW)",
    "Volume Requirements Forecast - DC-L - GB (MW)",
    # "Volume Requirements Forecast - DR-H - GB (MW)",
    # "Volume Requirements Forecast - DR-L - GB (MW)",
    # "Volume Requirements Forecast - DM-H - GB (MW)",
    # "Volume Requirements Forecast - DM-L - GB (MW)",
    "Ancillary Volume Accepted - DC-H - GB (MW)",
    "Ancillary Volume Accepted - DC-L - GB (MW)",
    "Ancillary Volume Accepted - DR-H - GB (MW)",
    "Ancillary Volume Accepted - DR-L - GB (MW)",
    "Ancillary Volume Accepted - DM-H - GB (MW)",
    "Ancillary Volume Accepted - DM-L - GB (MW)",
]

generation_columns = [
    "Generation - Fuel - Biomass - GB (MW)",
    "Generation - Fuel - CCGT - GB (MW)",
    # "Generation - Fuel - Coal - GB (MW)",
    # "Generation - Fuel - Hydro - GB (MW)",
    "Generation - Fuel - Nuclear - GB (MW)",
    "Generation - Fuel - OCGT - GB (MW)",
    # "Generation - Fuel - Oil - GB (MW)",
    # "Generation - Fuel - Pumped Storage - GB (MW)",
    "Generation - Fuel - Wind - GB (MW)",
    # "Generation - Fuel - Interconnectors - GB (MW)"
]

MEL_below_PN_columns = [
    "MEL Below PN - Fuel - CCGT - GB (MW)",
    "MEL Below PN - Fuel - Nuclear - GB (MW)",
    "MEL Below PN - Fuel - Coal - GB (MW)",
    "MEL Below PN - Fuel - Biomass - GB (MW)",
    "MEL Below PN - Fuel - Pumped Storage - GB (MW)",
    "MEL Below PN - Fuel - Interconnectors - GB (MW)"
]

prices_and_forecasts_columns = [
    "National Demand Forecast (NDF) - GB (MW)"
]

day_ahead_price_columns = [
    "Day Ahead Price (N2EX, local) - GB (£/MWh)",
    "Day Ahead Price (EPEX, local) - GB (£/MWh)"
]


# Create one subplot for each dataframe
fig, axes = plt.subplots(nrows=6, ncols=1, figsize=(12, 18), sharex=True)

df_subset[price_cols].plot(ax=axes[0], title="DFR Prices (Target Variables)", alpha=0.7)
axes[0].set_ylabel("DFR Prices")

df_subset[remaining_price_volume_cols].plot(ax=axes[1], title="Volume", alpha=0.7)
axes[1].set_ylabel("Volume")

df_subset[generation_columns].plot(ax=axes[2], title="Generation", alpha=0.7)
axes[2].set_ylabel("Generation Variables")

df_subset[MEL_below_PN_columns].plot(ax=axes[3], title="MEL below PN", alpha=0.7)
axes[3].set_ylabel("MEL below PN Variables")

df_subset[prices_and_forecasts_columns].plot(ax=axes[4], title="National Demand Forecast", alpha=0.7)
axes[4].set_ylabel("National Demand Forecast")
```

```
df_subset[day_ahead_price_columns].plot(ax=axes[5], title="Day Ahead Price", alpha=0.7)
axes[5].set_ylabel("Day Ahead Price Variables")

plt.xlabel("Time")
plt.tight_layout()
plt.show()
```

```
(2185, 39)
```



In [19]:
```
df_filtered = df_combined[[

    "Ancillary Price - DC-H - GB (£/MW/h)",
    "Ancillary Price - DC-L - GB (£/MW/h)",
```

```
    "Ancillary Volume Accepted - DC-H - GB (MW)",
    "Ancillary Volume Accepted - DC-L - GB (MW)",

    "Day Ahead Price (EPEX, local) - GB (£/MWh)",

    "Volume Requirements Forecast - DC-H - GB (MW)",
    "Volume Requirements Forecast - DC-L - GB (MW)",

    "Generation - Fuel - OCGT - GB (MW)",
    "Generation - Fuel - CCGT - GB (MW)",
    "Generation - Fuel - Nuclear - GB (MW)",
    "Generation - Fuel - Wind - GB (MW)",
    "Generation - Fuel - Biomass - GB (MW)",

    "MEL Below PN - Fuel - CCGT - GB (MW)",
    "MEL Below PN - Fuel - Nuclear - GB (MW)",
    "MEL Below PN - Fuel - Coal - GB (MW)",
    "MEL Below PN - Fuel - Biomass - GB (MW)",
    "MEL Below PN - Fuel - Pumped Storage - GB (MW)",
    "MEL Below PN - Fuel - Interconnectors - GB (MW)",

    "National Demand Forecast (NDF) - GB (MW)"

]]
```

In [20]:
```
df_filtered.head()
```

Out[20]:

	Ancillary Price - DC-H - GB (£/MW/h)	Ancillary Price - DC-L - GB (£/MW/h)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Day Ahead Price (EPEX, local) - GB (£/MWh)	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Generatio - Fue Wind - G (MW
11	0.81	0.55	1250.0	1073.0	35.7	1118.0	931.0	1	3541	4664	109
12	2.64	0.65	1250.0	1049.0	31.6	1250.0	942.0	0	3789	4653	100
13	0.72	1.30	1250.0	999.0	71.0	1250.0	967.0	2	7727	4659	89
14	0.70	1.07	964.0	881.0	74.4	1127.0	937.0	49	13049	4660	64
15	1.04	3.01	952.0	880.0	73.0	967.0	825.0	0	15772	4660	60

In [21]:
```
df_filtered.shape
```

Out[21]: (2185, 19)

Normalise variables to ensure that our model is not handling excessively large or small values.

In [23]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Select all columns for scaling.
cols_to_scale = df_filtered.columns[:]

# Apply MinMaxScaler only to the selected columns.
scaler = MinMaxScaler()
df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])

print(df_filtered.shape)
df_filtered.head()
```

```
(2185, 19)
```

```
C:\Users\lezha\AppData\Local\Temp\ipykernel_35356\268783779.py:9: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value '[0.0013986 0.        0.0027972 ... 0.0013
986 0.0013986 0.       ]' has dtype incompatible with int64, please explicitly cast to a compatible dtype first.
  df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])
C:\Users\lezha\AppData\Local\Temp\ipykernel_35356\268783779.py:9: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value '[0.13203326 0.14128044 0.28811663 ... 0.1
8863492 0.18542824 0.14549387]' has dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])
C:\Users\lezha\AppData\Local\Temp\ipykernel_35356\268783779.py:9: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value '[0.82257496 0.82063492 0.82169312 ... 0.8
9118166 0.89259259 0.89241623]' has dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])
C:\Users\lezha\AppData\Local\Temp\ipykernel_35356\268783779.py:9: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value '[0.64123206 0.58680434 0.52076771 ... 0.8
9627815 0.89913662 0.86833508]' has dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])
C:\Users\lezha\AppData\Local\Temp\ipykernel_35356\268783779.py:9: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value '[0.26970704 0.2718212  0.73089701 ... 0.3
4974328 0.34672304 0.2358804 ]' has dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df_filtered.loc[:, cols_to_scale] = scaler.fit_transform(df_filtered.loc[:, cols_to_scale])
```

Out[23]:

	Ancillary Price - DC-H - GB (£/MW/h)	Ancillary Price - DC-L - GB (£/MW/h)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Day Ahead Price (EPEX, local) - GB (£/MWh)	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Generati - Fu Wind - ((M
11	0.104288	0.017647	0.678133	0.457778	0.142167	0.307388	0.240545	0.001399	0.132033	0.822575	0.6412
12	0.282651	0.019048	0.678133	0.422222	0.130509	0.481530	0.257186	0.000000	0.141280	0.820635	0.5868
13	0.095517	0.028151	0.678133	0.348148	0.242536	0.481530	0.295008	0.002797	0.288117	0.821693	0.5207
14	0.093567	0.024930	0.443898	0.173333	0.252204	0.319261	0.249622	0.068531	0.486558	0.821869	0.3782
15	0.126706	0.052101	0.434070	0.171852	0.248223	0.108179	0.080182	0.000000	0.588091	0.821869	0.3530

In [24]:
```
# Get the scaling factor for DC-H price
scaling_factor_DC_H = scaler.scale_[0]
print(f"Scaling factor for DC-H price: {scaling_factor_DC_H}")
```

```
Scaling factor for DC-H price: 0.09746588693957116
```

In [25]:
```
df_filtered = df_filtered.reset_index(drop=True)
df_filtered
```

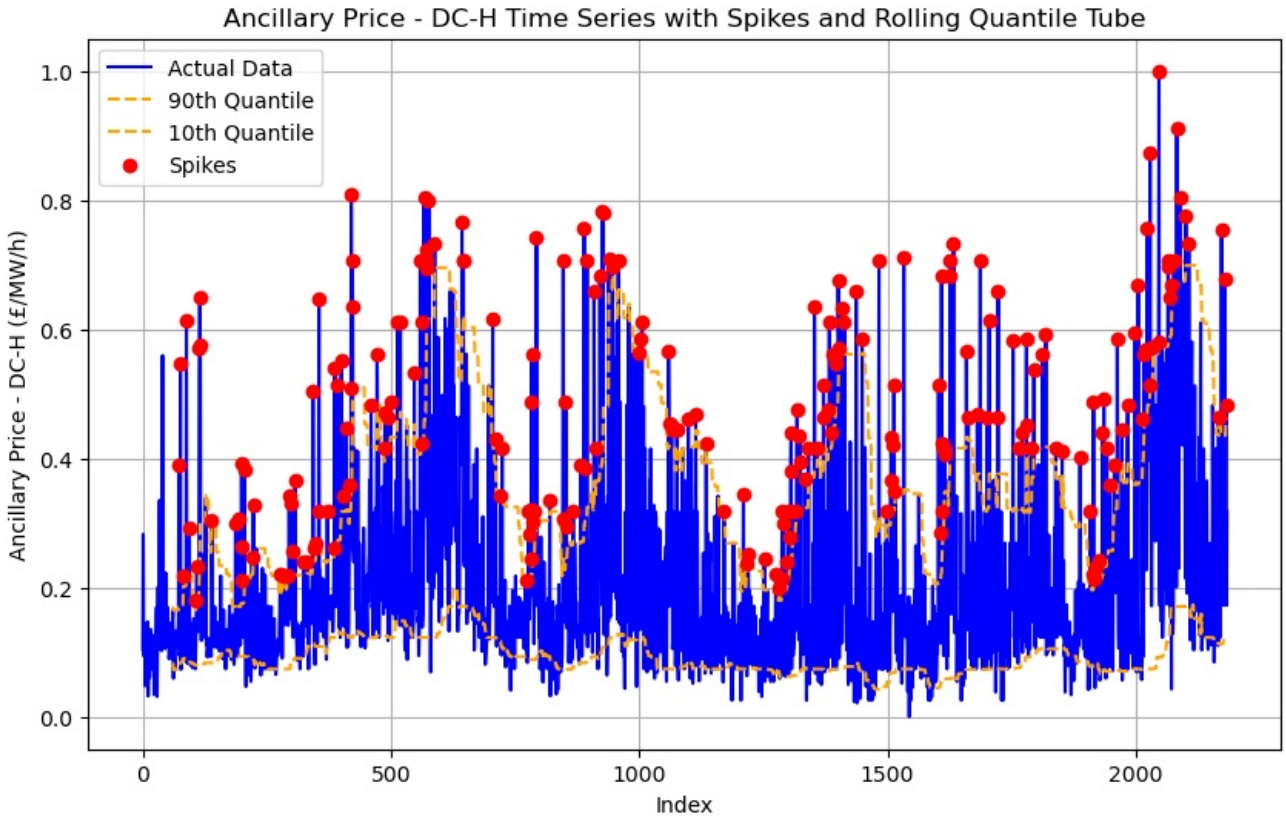
Out[25]:

	Ancillary Price - DC-H - GB (£/MW/h)	Ancillary Price - DC-L - GB (£/MW/h)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Day Ahead Price (EPEX, local) - GB (£/MWh)	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Gener - F Wind
0	0.104288	0.017647	0.678133	0.457778	0.142167	0.307388	0.240545	0.001399	0.132033	0.822575	0.64
1	0.282651	0.019048	0.678133	0.422222	0.130509	0.481530	0.257186	0.000000	0.141280	0.820635	0.58
2	0.095517	0.028151	0.678133	0.348148	0.242536	0.481530	0.295008	0.002797	0.288117	0.821693	0.52
3	0.093567	0.024930	0.443898	0.173333	0.252204	0.319261	0.249622	0.068531	0.486558	0.821869	0.37
4	0.126706	0.052101	0.434070	0.171852	0.248223	0.108179	0.080182	0.000000	0.588091	0.821869	0.35
...	...	...	...	...	...	...	...	...	...	...	
2180	0.239766	0.075630	0.706798	0.844444	0.233438	0.596306	1.000000	0.001399	0.162385	0.890300	0.85
2181	0.415205	0.039916	0.639640	0.758519	0.262724	0.543536	0.883510	0.001399	0.177784	0.890300	0.87
2182	0.482456	0.095098	0.608518	0.672593	0.256753	0.419525	0.798790	0.001399	0.188635	0.891182	0.89
2183	0.172515	0.071569	0.673219	0.832593	0.188655	0.422164	0.732224	0.001399	0.185428	0.892593	0.89
2184	0.317739	0.044958	0.734644	0.777778	0.057720	0.514512	0.835098	0.000000	0.145494	0.892416	0.86

2185 rows × 19 columns

Define spikes as anything above the 90th quantile of a 60-EFA block rolling window

In [80]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define rolling quantile function (0.10 and 0.90 quantiles)
def rolling_quantiles(series, window):
    # Compute the rolling 0.10 and 0.90 quantiles
    rolling_q90 = series.rolling(window).quantile(0.90)
    rolling_q10 = series.rolling(window).quantile(0.10)
    return rolling_q10, rolling_q90

# Set window size for rolling quantiles
window_size = 60  # Adjust this based on how "smooth" you want the bounds to be

# Calculate the rolling 0.10 and 0.90 quantiles
q10, q90 = rolling_quantiles(df_filtered["Ancillary Price - DC-H - GB (£/MW/h)"], window_size)

# Identify spikes: points greater than the upper 0.90 quantile
spikes = df_filtered[df_filtered["Ancillary Price - DC-H - GB (£/MW/h)"] > q90]

# Plot the time series, the tube (envelope), and spikes
plt.figure(figsize=(10, 6))

# Plot the actual data
plt.plot(df_filtered.index, df_filtered["Ancillary Price - DC-H - GB (£/MW/h)"], label="Actual Data", color='blu

# Plot the upper and lower bounds
plt.plot(df_filtered.index, q90, label="90th Quantile", color='orange', linestyle='--')
plt.plot(df_filtered.index, q10, label="10th Quantile", color='orange', linestyle='--')

# Plot the spikes (anything above the upper 0.90 quantile)
plt.scatter(spikes.index, spikes["Ancillary Price - DC-H - GB (£/MW/h)"], color='red', label="Spikes", zorder=5

# Labels and title
plt.xlabel("Index")
plt.ylabel("Ancillary Price - DC-H (£/MW/h)")
plt.title("Ancillary Price - DC-H Time Series with Spikes and Rolling Quantile Tube")
plt.legend()
plt.grid(True)

# Show the plot
plt.show()

# Print the spikes indices for reference
print("Spikes detected at indices:", spikes.index)

# Display spikes subset of dataframe
spikes
```




```
Spikes detected at indices: Index([  73,   76,   82,   88,   94,  108,  109,  114,  115,  116,
       ...
       2073, 2077, 2083, 2089, 2101, 2107, 2167, 2173, 2179, 2182],
      dtype='int64', length=225)
```

Out[80]:

	Ancillary Price - DC-H - GB (£/MW/h)	Ancillary Price - DC-L - GB (£/MW/h)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Day Ahead Price (EPEX, local) - GB (£/MWh)	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Gener - F Wind
73	0.388889	0.000000	0.678133	0.362963	0.200455	0.481530	0.228442	0.000000	0.126254	0.518166	0.84
76	0.546784	0.096779	0.466830	0.071111	0.279215	0.124011	0.258699	0.069930	0.746076	0.436684	0.79
82	0.218324	0.086975	0.385749	0.154074	0.307364	0.055409	0.027231	0.074126	0.870167	0.438095	0.56
88	0.614035	0.132493	0.330876	0.183704	0.306796	0.002639	0.000000	0.081119	0.945375	0.437566	0.30
94	0.293372	0.175490	0.319410	0.161481	0.266989	0.050132	0.004539	0.001399	0.870502	0.438272	0.55
...	...	...	...	...	...	...	...	...	...	...	
2107	0.731969	0.075630	0.749386	0.831111	0.044214	0.514512	0.733737	0.000000	0.097095	0.817813	0.86
2167	0.463938	0.041457	0.657658	0.638519	0.265852	0.522427	0.733737	0.001399	0.219881	0.897707	0.43
2173	0.753411	0.055182	0.749386	0.908148	0.151834	0.521108	0.685325	0.001399	0.090906	0.891887	0.71
2179	0.677388	0.075630	0.749386	0.884444	0.120273	0.572559	0.771558	0.000000	0.121779	0.890476	0.80
2182	0.482456	0.095098	0.608518	0.672593	0.256753	0.419525	0.798790	0.001399	0.188635	0.891182	0.89

225 rows × 19 columns

In [28]:
```
spike_indices = spikes.index.to_numpy()
spike_indices
```

Out[28]:
```
array([  73,   76,   82,   88,   94,  108,  109,  114,  115,  116,  139,
        187,  193,  198,  199,  200,  205,  220,  223,  277,  283,  285,
        291,  295,  300,  301,  307,  325,  331,  343,  344,  349,  354,
        355,  372,  386,  387,  393,  400,  403,  409,  417,  420,  421,
        422,  423,  460,  472,  487,  488,  493,  499,  513,  519,  547,
        559,  561,  562,  567,  571,  572,  573,  576,  588,  643,  645,
        646,  705,  711,  721,  724,  772,  777,  778,  781,  782,  783,
        784,  785,  790,  793,  820,  846,  847,  852,  853,  867,  883,
        889,  891,  895,  909,  913,  921,  925,  927,  939,  946,  958,
        999, 1002, 1006, 1059, 1060, 1065, 1066, 1076, 1099, 1114, 1135,
       1171, 1210, 1216, 1219, 1252, 1275, 1282, 1287, 1288, 1291, 1297,
       1300, 1302, 1305, 1306, 1309, 1315, 1318, 1323, 1324, 1333, 1341,
       1353, 1359, 1371, 1372, 1381, 1383, 1387, 1389, 1395, 1396, 1401,
       1402, 1408, 1413, 1437, 1449, 1483, 1498, 1507, 1509, 1510, 1513,
       1515, 1533, 1603, 1608, 1609, 1610, 1611, 1612, 1614, 1615, 1626,
       1627, 1633, 1659, 1663, 1681, 1687, 1701, 1705, 1722, 1723, 1753,
       1765, 1771, 1779, 1780, 1789, 1795, 1813, 1819, 1839, 1852, 1888,
       1909, 1914, 1915, 1916, 1921, 1927, 1933, 1936, 1942, 1948, 1957,
       1963, 1972, 1984, 1999, 2005, 2014, 2017, 2018, 2023, 2027, 2029,
       2035, 2046, 2047, 2065, 2066, 2068, 2073, 2077, 2083, 2089, 2101,
       2107, 2167, 2173, 2179, 2182], dtype=int64)
```

In [29]:
```python
# Define the spike detection window size and delta t
spike_detection_window_size = 8  # Lookback window size for spike prediction

delta_t = 6   # Forecast horizon is 6 EFA blocks

# Function to get the relevant window of data around each spike
def get_spike_window(df, spike_index, window_size, delta_t):
    # Ensure we do not go below the first index of the dataframe
    return df.iloc[spike_index - window_size - delta_t:spike_index - delta_t + 1]

# Create a new dictionary to store dataframes of each spike window
spike_windows_df = {}

# Get the spike windows for all detected spikes
for spike_idx in spikes.index:
    spike_windows_df[spike_idx] = get_spike_window(df_filtered, spike_idx, spike_detection_window_size, delta_t

spike_windows_df[76]
```

Out[29]:

	Ancillary Price - DC-H - GB (£/MW/h)	Ancillary Price - DC-L - GB (£/MW/h)	Ancillary Volume Accepted - DC-H - GB (MW)	Ancillary Volume Accepted - DC-L - GB (MW)	Day Ahead Price (EPEX, local) - GB (£/MWh)	Volume Requirements Forecast - DC-H - GB (MW)	Volume Requirements Forecast - DC-L - GB (MW)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Generati - Fu Wind - ((M
62	0.059454	0.025490	0.449631	0.167407	0.256753	0.379947	0.316188	0.002797	0.556024	0.547619	0.4529
63	0.097466	0.018347	0.384111	0.048889	0.258175	0.259894	0.287443	0.002797	0.609381	0.547619	0.4137
64	0.083821	0.027591	0.384930	0.074074	0.258743	0.077836	0.127080	0.001399	0.691898	0.547266	0.4085
65	0.072125	0.023249	0.455364	0.160000	0.273813	0.072559	0.239032	0.002797	0.750774	0.548501	0.3940
66	0.111111	0.013025	0.599509	0.280000	0.253625	0.133245	0.189107	0.000000	0.434468	0.544797	0.3654
67	0.133528	0.009664	0.678133	0.296296	0.233438	0.352243	0.223903	0.000000	0.375033	0.547619	0.3264
68	0.128655	0.023950	0.552826	0.241481	0.233722	0.481530	0.308623	0.000000	0.396510	0.532628	0.3769
69	0.115984	0.016947	0.437346	0.084444	0.253625	0.481530	0.293495	0.000000	0.425221	0.532275	0.5805
70	0.122807	0.029972	0.432432	0.090370	0.259881	0.141161	0.151286	0.001399	0.498080	0.531570	0.7102

Only select relevant variables for spike forecasting

In [31]:
```python
# Define spike variables
spike_variables = [0, 7, 8, 9, 10, 11, 12, 13]  # the subset of variables that are likely to be precursors for s
spike_variable_columns = ['Variable ' + str(i) for i in spike_variables]

# Create a new dictionary to store the modified spike windows with time as the last channel
modified_spike_windows_df = {}

# Iterate over the spike windows in spike_windows_df
for spike_idx, window_df in spike_windows_df.items():
    # Extract the relevant spike variables
    spike_data = window_df.iloc[:, spike_variables]

    # Create a time channel, normalised from 0 to the length of the window
    time_channel = np.linspace(0, len(spike_data) - 1, len(spike_data), dtype=np.float32).reshape(-1, 1)
```

```
    # Concatenate time as the last column
    augmented_window = np.concatenate([spike_data.to_numpy(), time_channel], axis=1)

    # Create a DataFrame with time as the last column
    augmented_window_df = pd.DataFrame(augmented_window, columns=spike_variable_columns + ['Time'])

    # Store the modified window in the new dictionary
    modified_spike_windows_df[spike_idx] = augmented_window_df

# Example output for spike index 76
modified_spike_windows_df[76]
```

Out[31]:

	Variable 0	Variable 7	Variable 8	Variable 9	Variable 10	Variable 11	Variable 12	Variable 13	Time
0	0.059454	0.002797	0.556024	0.547619	0.452981	0.352159	0.968084	1.000000	0.0
1	0.097466	0.002797	0.609381	0.547619	0.413779	0.508608	0.992845	1.000000	1.0
2	0.083821	0.001399	0.691898	0.547266	0.408529	0.449713	0.996848	1.000000	2.0
3	0.072125	0.002797	0.750774	0.548501	0.394061	0.423437	0.996449	1.000000	3.0
4	0.111111	0.000000	0.434468	0.544797	0.365418	0.374207	0.993719	1.000000	4.0
5	0.133528	0.000000	0.375033	0.547619	0.326450	0.338870	0.995930	0.998917	5.0
6	0.128655	0.000000	0.396510	0.532628	0.376911	0.330414	0.979284	0.901653	6.0
7	0.115984	0.000000	0.425221	0.532275	0.580562	0.677741	0.998668	0.996265	7.0
8	0.122807	0.001399	0.498080	0.531570	0.710244	0.678345	0.814823	0.996265	8.0

In [32]:
```
# Define spike variables
spike_variables = [0, 7, 8, 9, 10, 11, 12, 13]  # the subset of variables that are likely to be precursors for s
spike_variable_columns = ['Variable ' + str(i) for i in spike_variables]

# Create a new dictionary to store the modified spike windows without time augmentation
modified_spike_windows_df_no_time_aug = {}

# Iterate over the spike windows in spike_windows_df
for spike_idx, window_df in spike_windows_df.items():
    # Extract the relevant spike variables
    spike_data = window_df.iloc[:, spike_variables]

    # Store the modified window (without the time augmentation) in the new dictionary
    modified_spike_windows_df_no_time_aug[spike_idx] = spike_data

# Example output for spike index 76
modified_spike_windows_df_no_time_aug[76]
```

Out[32]:

	Ancillary Price - DC-H - GB (£/MW/h)	Generation - Fuel - OCGT - GB (MW)	Generation - Fuel - CCGT - GB (MW)	Generation - Fuel - Nuclear - GB (MW)	Generation - Fuel - Wind - GB (MW)	Generation - Fuel - Biomass - GB (MW)	MEL Below PN - Fuel - CCGT - GB (MW)	MEL Below PN - Fuel - Nuclear - GB (MW)
62	0.059454	0.002797	0.556024	0.547619	0.452981	0.352159	0.968084	1.000000
63	0.097466	0.002797	0.609381	0.547619	0.413779	0.508608	0.992845	1.000000
64	0.083821	0.001399	0.691898	0.547266	0.408529	0.449713	0.996848	1.000000
65	0.072125	0.002797	0.750774	0.548501	0.394061	0.423437	0.996449	1.000000
66	0.111111	0.000000	0.434468	0.544797	0.365418	0.374207	0.993719	1.000000
67	0.133528	0.000000	0.375033	0.547619	0.326450	0.338870	0.995930	0.998917
68	0.128655	0.000000	0.396510	0.532628	0.376911	0.330414	0.979284	0.901653
69	0.115984	0.000000	0.425221	0.532275	0.580562	0.677741	0.998668	0.996265
70	0.122807	0.001399	0.498080	0.531570	0.710244	0.678345	0.814823	0.996265

Compute signatures of time series segments preceding spikes

In [34]:
```
# pip install keras_sig
from keras_sig import jax_gpu_signature

signatures_dict = {}
signatures_dict_no_time_aug = {}

signature_depth = 3

# Iterate over the modified spike windows and compute signatures
for spike_idx, augmented_window_df in modified_spike_windows_df.items():
    # Convert the augmented window to a numpy array (excluding the time column)
```

```
    augmented_window = augmented_window_df.to_numpy()

    # Extract the paths (including time) and compute their signature
    paths = np.expand_dims(augmented_window, axis=0)  # Shape: (1, length, channels + 1)

    # Compute the signature using the GPU (or CPU depending on your setup)
    signature = jax_gpu_signature(paths, depth=signature_depth, stream=False)  # Adjust depth as needed

    # Store the computed signature in the dictionary
    signatures_dict[spike_idx] = signature


# Repeat but this time no time augmentation
for spike_idx, augmented_window_df in modified_spike_windows_df_no_time_aug.items():
    # Convert the augmented window to a numpy array (excluding the time column)
    augmented_window = augmented_window_df.to_numpy()

    # Extract the paths (including time) and compute their signature
    paths = np.expand_dims(augmented_window, axis=0)  # Shape: (1, length, channels + 1)

    # Compute the signature using the GPU (or CPU depending on your setup)
    signature = jax_gpu_signature(paths, depth=signature_depth, stream=False)  # Adjust depth as needed

    # Store the computed signature in the dictionary
    signatures_dict_no_time_aug[spike_idx] = signature

# Example output for spike index 76
signatures_dict[76].shape, signatures_dict_no_time_aug[76].shape
```

Out[34]:
```
((1, 819), (1, 584))
```

In [35]:
```
signatures_dict.keys()
```

Out[35]:
```
dict_keys([73, 76, 82, 88, 94, 108, 109, 114, 115, 116, 139, 187, 193, 198, 199, 200, 205, 220, 223, 277, 283,
285, 291, 295, 300, 301, 307, 325, 331, 343, 344, 349, 354, 355, 372, 386, 387, 393, 400, 403, 409, 417, 420, 4
21, 422, 423, 460, 472, 487, 488, 493, 499, 513, 519, 547, 559, 561, 562, 567, 571, 572, 573, 576, 588, 643, 64
5, 646, 705, 711, 721, 724, 772, 777, 778, 781, 782, 783, 784, 785, 790, 793, 820, 846, 847, 852, 853, 867, 883
, 889, 891, 895, 909, 913, 921, 925, 927, 939, 946, 958, 999, 1002, 1006, 1059, 1060, 1065, 1066, 1076, 1099, 1
114, 1135, 1171, 1210, 1216, 1219, 1252, 1275, 1282, 1287, 1288, 1291, 1297, 1300, 1302, 1305, 1306, 1309, 1315
, 1318, 1323, 1324, 1333, 1341, 1353, 1359, 1371, 1372, 1381, 1383, 1387, 1389, 1395, 1396, 1401, 1402, 1408, 1
413, 1437, 1449, 1483, 1498, 1507, 1509, 1510, 1513, 1515, 1533, 1603, 1608, 1609, 1610, 1611, 1612, 1614, 1615
, 1626, 1627, 1633, 1659, 1663, 1681, 1687, 1701, 1705, 1722, 1723, 1753, 1765, 1771, 1779, 1780, 1789, 1795, 1
813, 1819, 1839, 1852, 1888, 1909, 1914, 1915, 1916, 1921, 1927, 1933, 1936, 1942, 1948, 1957, 1963, 1972, 1984
, 1999, 2005, 2014, 2017, 2018, 2023, 2027, 2029, 2035, 2046, 2047, 2065, 2066, 2068, 2073, 2077, 2083, 2089, 2
101, 2107, 2167, 2173, 2179, 2182])
```

Two-stage optimisation for spike parameter estimation

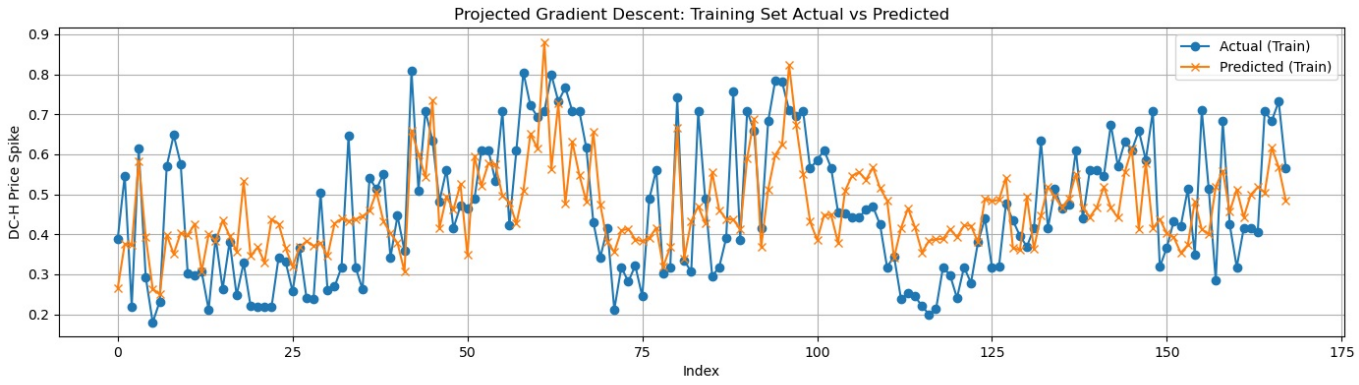
1) LASSO for sparsity: fewer nonzero coefficients means fewer redundant variables and less chance of overfitting.

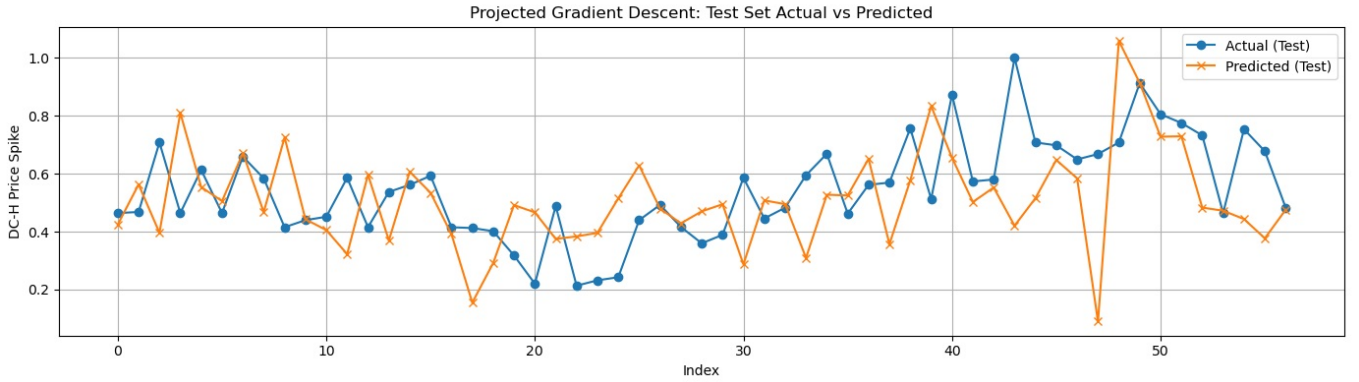
2) Re-estimate the parameters using Ordinary Least Squares (OLS), under the constraint that any coefficients shrunk to zero by LASSO remain zero. This is achieved via Projected Gradient Descent (PGD).

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from keras_sig import jax_gpu_signature

# Assuming spike_windows_df, spike_variables, and spikes are already defined

# Time-augmented path collection
paths = []
for spike_idx, window_df in spike_windows_df.items():
    subset = window_df.iloc[:, spike_variables].to_numpy().astype(np.float32)  # shape: (length, channels)

    # Create a time channel and concatenate it as the first channel
    time_channel = np.linspace(0, subset.shape[0], subset.shape[0], dtype=np.float32).reshape(-1, 1)  # Normali
    augmented_path = np.concatenate([time_channel, subset], axis=1)  # shape: (length, channels+1)

    paths.append(augmented_path)

# Stack into a batch: shape (batch_size, length, channels+1)
paths_batch = np.stack(paths)

# Compute signatures using GPU
signatures = jax_gpu_signature(paths_batch, depth=3, stream=False)  # shape: (batch_size, signature_dim)

# Map back to spike indices if needed
signatures_dict = {spike_idx: signatures[i] for i, spike_idx in enumerate(spike_windows_df.keys())}
```

```
# Extract target variable
dc_h_price_spikes = spikes["Ancillary Price - DC-H - GB (£/MW/h)"]

# Step 1: Stack signatures into a matrix for model input (X)
X = np.stack([signatures_dict[idx] for idx in dc_h_price_spikes.index])

# Step 2: Convert target vector (y) to NumPy array
y = dc_h_price_spikes.values

# Step 3: Train-test split (75% train, 25% test) without shuffling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, shuffle=False)

# Step 4: Fit Lasso regression on training data (including intercept)
lasso = Lasso(alpha=0.004)
lasso.fit(X_train, y_train)

# Extract coefficients
theta_lasso = lasso.coef_
intercept = lasso.intercept_

# Combine intercept with coefficients
theta_lasso_combined = np.concatenate(([intercept], theta_lasso))

# Indices of coefficients that were zeroed out by Lasso
lasso_zeroed_indices = np.where(theta_lasso == 0)[0] + 1  # shift to account for intercept

# Step 5: Projected Gradient Descent

def projected_gradient_descent(X, y, initial_theta, lasso_zeroed_indices, learning_rate=0.01, max_iter=10000):
    m = X.shape[0]

    # Add bias column
    X_with_bias = np.hstack([np.ones((m, 1)), X])

    theta = initial_theta.copy()

    for i in range(max_iter):
        # Compute gradient
        gradients = -2 * X_with_bias.T.dot(y - X_with_bias.dot(theta)) / m

        # Gradient descent step
        theta = theta - learning_rate * gradients

        # Project (excluding bias term at index 0)
        theta[lasso_zeroed_indices] = 0

        if np.linalg.norm(gradients) < 1e-7:
            break

    return theta


# Step 6: Run PGD
theta_pgd_combined = projected_gradient_descent(X_train, y_train, theta_lasso_combined, lasso_zeroed_indices)

# Step 7: Predictions using optimised coefficients
y_pred_train_pgd = X_train.dot(theta_pgd_combined[1:]) + theta_pgd_combined[0]
y_pred_test_pgd = X_test.dot(theta_pgd_combined[1:]) + theta_pgd_combined[0]

# Step 8: Plotting results
plt.figure(figsize=(14, 8))

# Training set
plt.subplot(2, 1, 1)
plt.plot(y_train, label="Actual (Train)", marker='o')
plt.plot(y_pred_train_pgd, label="Predicted (Train)", marker='x')
plt.title("Projected Gradient Descent: Training Set Actual vs Predicted")
plt.xlabel("Index")
plt.ylabel("DC-H Price Spike")
plt.legend()
plt.grid(True)

# Test set
plt.subplot(2, 1, 2)
plt.plot(y_test, label="Actual (Test)", marker='o')
plt.plot(y_pred_test_pgd, label="Predicted (Test)", marker='x')
plt.title("Projected Gradient Descent: Test Set Actual vs Predicted")
plt.xlabel("Index")
plt.ylabel("DC-H Price Spike")
plt.legend()
plt.grid(True)

plt.tight_layout()
```

```
plt.show()
```





Time indices of spikes that are in the training set

In [39]:
```
train_len = int(0.75 * len(df_filtered))
train_spike_keys = [k for k in signatures_dict.keys() if k < train_len]
print(train_spike_keys)
len(train_spike_keys)
```

```
[73, 76, 82, 88, 94, 108, 109, 114, 115, 116, 139, 187, 193, 198, 199, 200, 205, 220, 223, 277, 283, 285, 291, 2
95, 300, 301, 307, 325, 331, 343, 344, 349, 354, 355, 372, 386, 387, 393, 400, 403, 409, 417, 420, 421, 422, 423
, 460, 472, 487, 488, 493, 499, 513, 519, 547, 559, 561, 562, 567, 571, 572, 573, 576, 588, 643, 645, 646, 705,
711, 721, 724, 772, 777, 778, 781, 782, 783, 784, 785, 790, 793, 820, 846, 847, 852, 853, 867, 883, 889, 891, 89
5, 909, 913, 921, 925, 927, 939, 946, 958, 999, 1002, 1006, 1059, 1060, 1065, 1066, 1076, 1099, 1114, 1135, 1171
, 1210, 1216, 1219, 1252, 1275, 1282, 1287, 1288, 1291, 1297, 1300, 1302, 1305, 1306, 1309, 1315, 1318, 1323, 13
24, 1333, 1341, 1353, 1359, 1371, 1372, 1381, 1383, 1387, 1389, 1395, 1396, 1401, 1402, 1408, 1413, 1437, 1449,
1483, 1498, 1507, 1509, 1510, 1513, 1515, 1533, 1603, 1608, 1609, 1610, 1611, 1612, 1614, 1615, 1626, 1627, 1633
]
```

Out[39]: 167

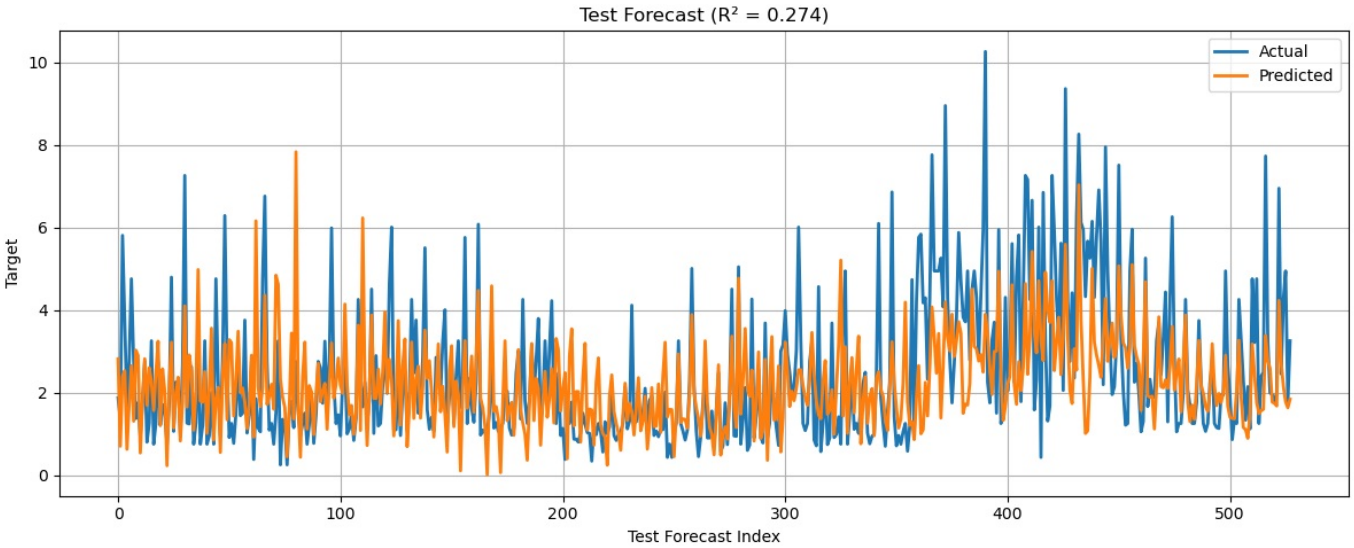
Forecasting with Spike LASSO

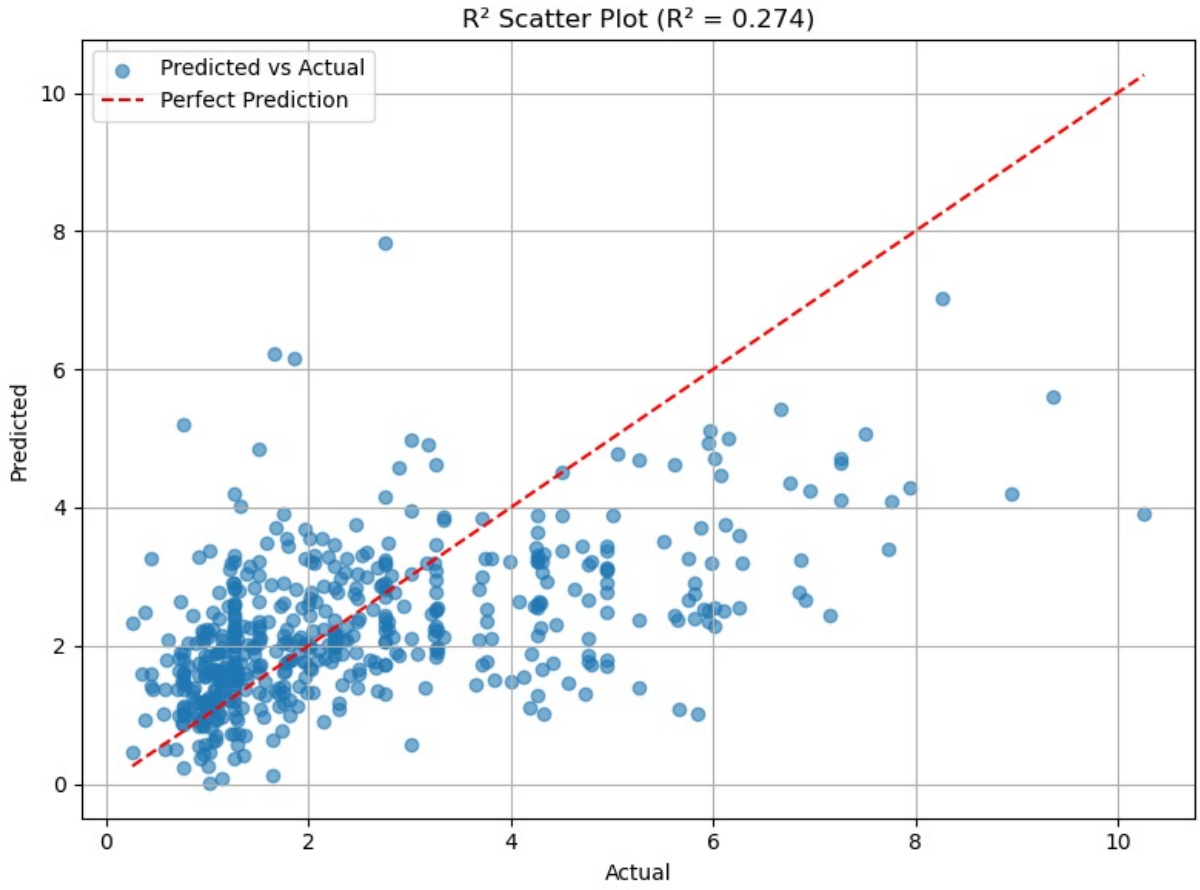
In [90]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score, root_mean_squared_error, mean_absolute_error

# --------------------------------
# PARAMETERS
# --------------------------------

# Columns for forecasting
input_columns = [0, 2, 5, 7, 8, 10, 11]  # For Lasso features
spike_columns = spike_variables

target_column = 0

lasso_window_size = 13           # Spans 3 EFA blocks of the same hour
delta_t = 6                      # Forecast horizon: predict value at t+delta_t
spike_prediction_window_size = 8

gamma = 5
similarity_threshold = 0.1


# --------- DATA PREPARATION
# --------------------------------

# Input variables
X_full = df_filtered.iloc[:, input_columns].to_numpy().astype(np.float32)
```

```
# Target values
y_full = df_filtered.iloc[:, target_column].to_numpy().astype(np.float32)

# Spike data
X_spike_full = df_filtered.iloc[:, spike_variables].to_numpy().astype(np.float32)

# Split data into training and testing sets (here, 75% training)
train_len = int(0.75 * len(X_full))
X_train_full, X_test_full = X_full[:train_len], X_full[train_len:]
y_train_full, y_test_full = y_full[:train_len], y_full[train_len:]
X_spike_test = X_spike_full[train_len:]

# ------------------------------
# LASSO TRAINING
# ------------------------------

# Build Lasso training windows.
lasso_X_train = []
lasso_y_train = []
for t in range(lasso_window_size, len(X_train_full) - delta_t):
    window = X_train_full[t - lasso_window_size : t + 1]  # window shape: (lasso_window_size, n_features)
    lasso_X_train.append(window.flatten())
    target_index = t + delta_t
    lasso_y_train.append(y_train_full[target_index])

lasso_X_train = np.array(lasso_X_train)
lasso_y_train = np.array(lasso_y_train)

lasso = Lasso(alpha=1e-6)
lasso.fit(lasso_X_train, lasso_y_train)

# ------------------------------
# LASSO TEST SETUP
# ------------------------------

lasso_X_test = []
for t in range(lasso_window_size, len(X_test_full) - delta_t):
    window = X_test_full[t - lasso_window_size : t + 1]
    lasso_X_test.append(window.flatten())

lasso_X_test = np.array(lasso_X_test)
lasso_preds = lasso.predict(lasso_X_test)

# ------------------------------
# FINAL FORECASTING WITH SPIKE SIMILARITY ADJUSTMENT
# ------------------------------

y_pred_final = []
y_actual_final = []

for t in range(lasso_window_size, len(X_test_full) - delta_t):
    # Base Lasso forecast for this test time
    lasso_index = t - lasso_window_size
    lasso_forecast = lasso_preds[lasso_index]

    # ---- Spike-based Forecast Adjustment ----
    # Step 1: Extract spike window (using spike columns from X_spike_test)
    t_start = t - spike_prediction_window_size
    t_end = t
    spike_window = X_spike_test[t_start:t_end + 1, :]  # Shape: (spike_prediction_window_size + 1, n_spike_featu

    # Step 2: Compute the spike signature for the test window.
    # No time augmentation as it distorts the distance
    ref_sig = jax_gpu_signature(spike_window[None, :, :], depth=signature_depth, stream=False)

    # Step 3: Compute distances between this ref_sig and each training spike signature
    all_train_sigs = np.stack([np.ravel(signatures_dict_no_time_aug[k]) for k in train_spike_keys])
    dists = np.linalg.norm(all_train_sigs - ref_sig, axis=1)

    # Step 4: Compute Boltzmann weights from the distances
    weights = np.exp(-gamma * dists)
    weights /= (weights.sum() + 1e-12)

    # Step 5: If any training pre-spike window is similar enough (weight > threshold), use spike-based forecast
    if np.any(weights > similarity_threshold):

        # Time augment the spike window: create a time channel with values from 0 to spike_prediction_window_si
        time_aug = np.linspace(0, spike_prediction_window_size, spike_prediction_window_size + 1).reshape(-1, 1

        # Concatenate time as an extra column (to the left or right—here we put it on the left)
        spike_window_aug = np.hstack([time_aug, spike_window])  # shape: (spike_prediction_window_size + 1, n_sp
```

```
        # Compute the signature of the time-augmented spike window
        ref_sig = jax_gpu_signature(spike_window_aug[None, :, :], depth=signature_depth, stream=False)

        # Flatten the signature
        ref_sig_flat = np.ravel(ref_sig)

        # Prepend a 1 to the signature as keras_sig does not compute the 0th level of the signature
        ref_sig_with_bias = np.concatenate(([1], ref_sig_flat))

        # Compute spike-based forecast using the dot product with the parameter found from the two-step LASSO-O
        adjusted_forecast = np.dot(theta_pgd_combined, ref_sig_with_bias)
        # adjusted_forecast = np.dot(theta_pgd, ref_sig_with_bias)

        lasso_forecast = adjusted_forecast  # Replace Lasso forecast with spike-based forecast

    # Store final forecast and corresponding actual target (at time t+delta_t)
    y_pred_final.append(lasso_forecast)
    y_actual_final.append(y_test_full[t + delta_t])

y_pred_final = np.array(y_pred_final) / scaling_factor_DC_H
y_actual_final = np.array(y_actual_final) / scaling_factor_DC_H


# ------------------------------
# EVALUATION AND PLOTTING
# ------------------------------

# Metrics
r2_test = r2_score(y_actual_final, y_pred_final)
rmse_test = root_mean_squared_error(y_actual_final, y_pred_final)
mae_test = mean_absolute_error(y_actual_final, y_pred_final)

# Print metrics
print(f"Test R²:   {r2_test:.4f}")
print(f"Test RMSE: {rmse_test:.4f}")
print(f"Test MAE:  {mae_test:.4f}")

# Time series forecast plot
plt.figure(figsize=(12, 5))
plt.plot(y_actual_final, label="Actual", linewidth=2)
plt.plot(y_pred_final, label="Predicted", linewidth=2)
plt.title(f'Test Forecast (R² = {r2_test:.3f})')
plt.xlabel('Test Forecast Index')
plt.ylabel('Target')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Scatter plot: Predicted vs Actual
plt.figure(figsize=(8, 6))
plt.scatter(y_actual_final, y_pred_final, alpha=0.6, label='Predicted vs Actual')
plt.plot([y_actual_final.min(), y_actual_final.max()],
         [y_actual_final.min(), y_actual_final.max()],
         color='red', linestyle='--', label='Perfect Prediction')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title(f'R² Scatter Plot (R² = {r2_test:.3f})')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\lezha\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:697: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations, check the scale of the features
or consider increasing regularisation. Duality gap: 5.123e-02, tolerance: 3.186e-03
  model = cd_fast.enet_coordinate_descent(
Test R²:   0.2740
Test RMSE: 1.4892
Test MAE:  1.0716
```





In [ ]: