PREDICTING ENERGY PRICES AND LOAD DEMAND

By

(TEAM G)

ANIL KUMAR MALYALA
MOHAMMED ABDUL RASHEED KHAN
PAVAN KUMAR BODDU
SNEHA REDDY MADHIREDDY

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2024

ABSTRACT

Given the volatility and complexity of energy markets, accurate forecasts are critical for efficient energy management and planning. This project addresses the challenge of predicting energy prices and load demands by leveraging a comprehensive dataset that integrates energy consumption, generation metrics, and weather data across Spain from 2015 to 2018. Various visualizations and statistical analyses were performed to identify patterns and correlations between weather variables and energy metrics. Predictive models, including linear regression, Random Forest, and deep learning models like LSTM were employed to forecast energy load and prices. The models were evaluated using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared. The results indicated significant correlations between weather parameters and energy consumption and pricing, with the predictive models showing varying degrees of accuracy. Notably, the deep learning models demonstrated superior performance in capturing complex temporal patterns in the data. This study contributes to the existing literature by showcasing the effectiveness of combining weather data with traditional energy metrics, offering a robust framework for predictive analysis in the energy sector.

LIST OF ABBREVIATIONS AND SYMBOLS

CNN: Convolutional Neural Network

EDA: Exploratory Data Analysis

GBM: Gradient Boosting Machine

kW: kilowatt

kWh: kilowatt-hour

LSTM: Long Short-Term Memory

MAE: Mean Absolute Error

MAPE: Mean Absolute Percentage Error

MSE: Mean Squared Error

MW: Megawatt

MWh: Megawatt-hour

PCA: Principal Component Analysis

PV: Photovoltaic

RMSE: Root Mean Squared Error

R^2: Coefficient of Determination

TSO: Transmission System Operator

XGBoost: eXtreme Gradient Boosting

ACKNOWLEDGMENTS

We acknowledge and thank Prof. Unal Sakoglu for providing feedback and guiding this research project.

TEAM MEMBERS' CONTRIBUTIONS

Anil Kumar Malyala was instrumental in feature engineering and model evaluation. He worked diligently on normalizing the data and applying the MinMaxScaler, which was crucial for model performance. Anil also implemented the LSTM model, contributing significantly to handling sequential data. Furthermore, Anil played a major role in compiling the final report and creating the presentation slides.

Mohammed Abdul Rasheed Khan contributed to initial data collection and preliminary data analysis. He assisted in visualizing the data, identifying key patterns, and supporting data preprocessing and model evaluation.

Pavan Kumar Boddu implemented the Random Forest and Linear Regression models and worked on evaluating their performance. He assisted in data normalization and feature engineering and contributed to integrating weather data into the models. Pavan also helped in writing sections of the final report.

Sneha Reddy Madhireddy played a crucial role in data preprocessing and feature engineering. She was primarily responsible for handling missing data, aligning timestamps, and creating new features to improve model performance. Sneha also implemented the XGBoost model, significantly enhancing prediction accuracy. Additionally, Sneha contributed significantly to the development of the presentation slides and the final report.

Name	Duties	Achievements
Anil Kumar Malyala	Feature engineering, normalization using MinMaxScaler, implementation of LSTM model, presentation, and report writing.	Played a significant role in feature engineering, model evaluation, data normalization, and analysis. Contributed to the presentation and report.
Mohammed Abdul Rasheed Khan	Initial data collection, preliminary data analysis, visualization, data preprocessing, model evaluation	Contributed to data collection, preliminary analysis, data visualization, and preprocessing.
Pavan Kumar Boddu	Implementation of Random Forest and Linear Regression models, data normalization, feature engineering, and report writing.	Implemented and evaluated Random Forest and Linear Regression models. Assisted in feature engineering and contributed to the report.

Sneha Reddy Madhireddy	Data preprocessing, feature engineering, implementation of XGBoost model, model evaluation, presentation, and report writing.	Successfully handled data preprocessing and feature engineering. Enhanced prediction accuracy. Contributed to the presentation and report.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The energy sector is increasingly leveraging advanced machine learning techniques to enhance the accuracy and efficiency of forecasting models. This evolution is driven by the growing complexity of energy systems and the critical need to integrate renewable energy sources, which are heavily influenced by variable factors such as weather conditions. Techniques such as XGBoost, LSTM, and Random Forest have emerged as powerful tools for predicting energy prices and load demands with high precision. These models offer robust predictions and provide insights into the intricate relationships between energy consumption, production rates, and environmental influences. This paper explores the application of these models in the context of hourly energy generation and weather data to predict energy prices and total load actual. Through a detailed analysis of various forecasting models, this study aims to contribute to optimizing energy management and planning strategies, ensuring more sustainable and efficient energy utilization.

The following sections will present a comprehensive literature review, outline the methodologies employed, discuss the results obtained, and conclude with the implications of our findings and potential avenues for future research.

I.1 Problem Statement

Energy management is critical due to increasing demand and the need for sustainable practices. Accurate forecasting of energy prices and load demands is essential for efficient energy management, allowing market participants to make informed decisions. Traditional models for forecasting energy demands often rely heavily on historical consumption and production data. However, these models may need to account sufficiently for the impact of external variables, such as weather, which can lead to significant inaccuracies. This project addresses this gap by integrating energy and weather data into the predictive models, aiming to improve the forecasts significantly.

I.2 Dataset Description

The dataset used in this project is a unique compilation of energy generation and consumption data linked with detailed weather conditions across Spain, collected hourly from 2015 to 2018. The energy dataset includes metrics such as total load, actual price, and generation by fuel type, totaling 35046 records and 29 features. The weather dataset consists of parameters like temperature, wind speed, humidity, and precipitation, enhancing the models' predictive capability by providing context on weather conditions that directly affect energy metrics. The Weather Dataset has 178396 records and 17 features. After preprocessing and feature engineering, the combined dataset comprises 35,046 records and 74 features.

energy_df

```
In [9]: energy_df.head()
```

Out[9]:

time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated	generation hydro pumped storage consumption	generation hydro run-of-river and poundage	gen re
2015-01-01 00:00:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0	0.0	0.0	NaN	920.0	1009.0	
2015-01-01 01:00:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0	0.0	0.0	NaN	1164.0	973.0	
2015-01-01 02:00:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0	0.0	0.0	NaN	1503.0	949.0	
2015-01-01 03:00:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0	0.0	0.0	NaN	1826.0	953.0	
2015-01-01 04:00:00	410.0	178.0	0.0	4038.0	3590.0	156.0	0.0	0.0	0.0	NaN	2109.0	952.0	

```
In [10]:
```

Fig.1.1 Energy Dataframe

Out[40]:

dt_iso	city_name	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	rain_1h	rain_3h	snow_3h	clouds_all	weather_id	weather_ma
2014-12-31 23:00:00	Valencia	270.475	270.475	270.475	1001	77	1	62	0.0	0.0	0.0	0	800	cle
2015-01-01 00:00:00	Valencia	270.475	270.475	270.475	1001	77	1	62	0.0	0.0	0.0	0	800	cle
2015-01-01 01:00:00	Valencia	269.686	269.686	269.686	1002	78	0	23	0.0	0.0	0.0	0	800	cle
2015-01-01 02:00:00	Valencia	269.686	269.686	269.686	1002	78	0	23	0.0	0.0	0.0	0	800	cle
2015-01-01 03:00:00	Valencia	269.686	269.686	269.686	1002	78	0	23	0.0	0.0	0.0	0	800	cle

```
In [41]:
```

Fig.1.2 Weather Dataframe

I.3 Literature Review

The study by Huan Zheng and Yanghui Wu presents an innovative approach to short-term wind power forecasting through an XGBoost model enhanced with weather similarity analysis and feature engineering (Zheng & Wu, 2019). By categorizing historical weather patterns using k-means clustering and optimizing feature selection, this method achieves superior accuracy compared to traditional forecasting models such as BPNN, CART, RF, and SVR. This advanced XGBoost-based model significantly improves the efficiency and effectiveness of short-term wind power predictions.

Chen Li and Zhenyu Chen propose a novel approach for power load forecasting (Li et al. 2019), crucial for guiding power construction and grid operations. Their combined forecast model integrates Long Short-Term Memory (LSTM) and XGBoost techniques to enhance accuracy. Initially, individual LSTM and XGBoost models were established to predict power load. Subsequently, the combined model employs an error reciprocal method to integrate the forecasts from both models. Experimental validation using data from The Electrician

Mathematical Contest in Modeling demonstrates the combined model's remarkable forecast accuracy of 0.57%, significantly surpassing that of individual models.

Nan Lu, Quan Ouyang, Yang Li, and Changfu Zou present a novel hybrid LSTM-based model (Lu et al., 2024) with online correction for day-ahead electrical load forecasting, crucial for modern power system efficiency. Extracting four features from the original dataset and integrating LSTM and fully connected neural network blocks, temporal and non-temporal aspects are effectively modeled. The model is fortified against disturbances and adapts to evolving load data distributions through a gradient regularization-based offline training algorithm and an output layer parameter fine-tuning-based online correction method. Extensive experiments confirm the strategy's superior accuracy over commonly used forecasting models, demonstrating its effectiveness in real-world applications.

Lei Wu and Mohammad Shahidehpour address the intertwined nature of electricity price and load forecasting in smart grids, where dynamic pricing necessitates a holistic approach (Wu & Shahidehpour, 2014). They introduce a two-stage integrated forecasting framework that acknowledges the mutual influence between price and load signals. Employing a hybrid time-series and adaptive wavelet neural network model at each stage, the method captures both linear and nonlinear relationships, considering factors such as multivariate autoregressive integrated moving average and generalized autoregressive conditional heteroscedasticity. Through criteria like average mean absolute percentage error and error variance, they demonstrate the efficacy of their approach using forecasting examples from the New York Independent System Operator market.

These studies collectively demonstrate the growing reliance on sophisticated machine learning techniques to enhance the accuracy and reliability of energy forecasting, crucial for effective energy management and planning in response to both consumer demand and weather variability.

I. METHODS

II.1 Data Collection and Preprocessing

The data for this project was collected from Kaggle (*Hourly Energy Demand Generation and Weather*, 2019) and includes two primary datasets: energy consumption and generation data, and weather data. The energy dataset spans from 2015 to 2018 and includes metrics such as total load, actual price, and generation by fuel type, totaling over 35,000 records. The weather dataset includes parameters like temperature, wind speed, humidity, and precipitation, totaling 178,396 records.

Data Cleaning:

Initially, columns with a significant number of missing values, such as 'generation hydro pumped storage aggregated' and 'forecast wind offshore day ahead,' were dropped. Other missing values were filled using forward fill (ffill) and backward fill (bfill) methods. Columns like 'generation fossil coal-derived gas,' 'generation fossil oil shale,' and others with only zero values were also removed. Linear interpolation was used to fill any remaining missing values after ffill and bfill. Columns such as 'weather_icon,' 'weather_main,' 'weather_description,' and 'weather_id' were dropped. Additionally, 'seville_snow_3h' and 'barcelona_snow_3h' were removed due to their lack of significant correlation with the target variables, determined using the Pearson correlation method.

Dataset Merging:

A new data frame was created to store forecast-related columns such as 'total load forecast' and 'price day ahead.' These columns were then removed from the main energy data frame. Due to the weather data having around 35,000 records per city (totaling 178,396 records), City names were transformed into columns using one-hot encoding. This transformation helped create new columns such as 'seville_temp', 'seville_rain_1h', and similar columns for other cities, transforming the initial 11 columns into 55 columns to accommodate all the cities. Five separate data frames were created for each city(Madrid, Bilbao, Seville, Barcelona, and Valencia). Weather data from different cities was cleaned and merged into a single data frame. Outliers in columns such as 'wind_speed,' 'rain_3h,' 'rain_1h,' and 'snow_3h' were capped to realistic maximum values. The cleaned energy and weather data frames were merged on the timestamp index to form a combined dataset with 74 features as shown in Fig. 3.

II.2 Feature Engineering

Additional features such as 'hour', 'weekday', 'month', and 'business hour' were extracted from the timestamp to capture time-related patterns. A new feature 'generation coal all' was created by combining 'generation fossil hard coal' and 'generation fossil brown coal/lignite. PCA was initially applied to reduce the dataset's dimensionality, but it did not enhance the model's efficiency. Consequently, PCA was excluded from the process. To enhance the accuracy of our analysis and modeling, normalization techniques are employed. Specifically, the MinMaxScaler is used to scale the data values to a normalized range between 0 and 1. This scaling is crucial for

the performance of many statistical methods and machine learning models, as it ensures that the data is within a consistent range, facilitating more effective learning and prediction.

```
energy_weather_df.head(5)
```

hour	generation biomass	generation fossil brown coal/ lignite	generation fossil gas	generation fossil hard coal	generation fossil oil	generation hydro pumped storage consumption	generation hydro run-of-river and poundage	generation hydro water reservoir	generation nuclear	generation other	generation other renewable	generation solar	generation waste	generation wind onshore	total load actual	price actual	madrid_temp	madrid_temp_min	madrid_temp
2015-01-01 00:00:00	449.0	328.0	5196.0	4755.0	158.0	920.0	1009.0	1658.0	7096.0	43.0	71.0	50.0	195.0	5890.0	24382.0	64.92	267.325	267.325	26
2015-01-01 01:00:00	448.0	323.0	4857.0	4581.0	157.0	1164.0	973.0	1371.0	7099.0	43.0	73.0	50.0	196.0	5461.0	22734.0	64.48	266.186	266.186	26
2015-01-01 02:00:00	438.0	254.0	4314.0	4131.0	160.0	1503.0	949.0	779.0	7098.0	43.0	75.0	50.0	191.0	5238.0	21286.0	59.32	266.186	266.186	26
2015-01-01 03:00:00	428.0	187.0	4130.0	3840.0	156.0	1826.0	953.0	720.0	7097.0	43.0	74.0	42.0	189.0	4935.0	20264.0	56.04	266.186	266.186	26
2015-01-01 04:00:00	410.0	178.0	4038.0	3590.0	156.0	2109.0	952.0	743.0	7098.0	43.0	74.0	34.0	188.0	4618.0	19905.0	53.63	265.442	265.442	26

```
[78] energy_weather_df.shape
(35063, 73)
```

Fig. 2.1 Energy and Weather Combined Dataframe

II.3 Exploratory Data Analysis

We have utilized various plots to analyze the dataset. Fig.3 shows the target variable's load and energy price distribution. This plot gives us an estimate of how the target variables are distributed. We can see the average recorded temperatures by city with Serville leading with the highest temperature of 293 °K.

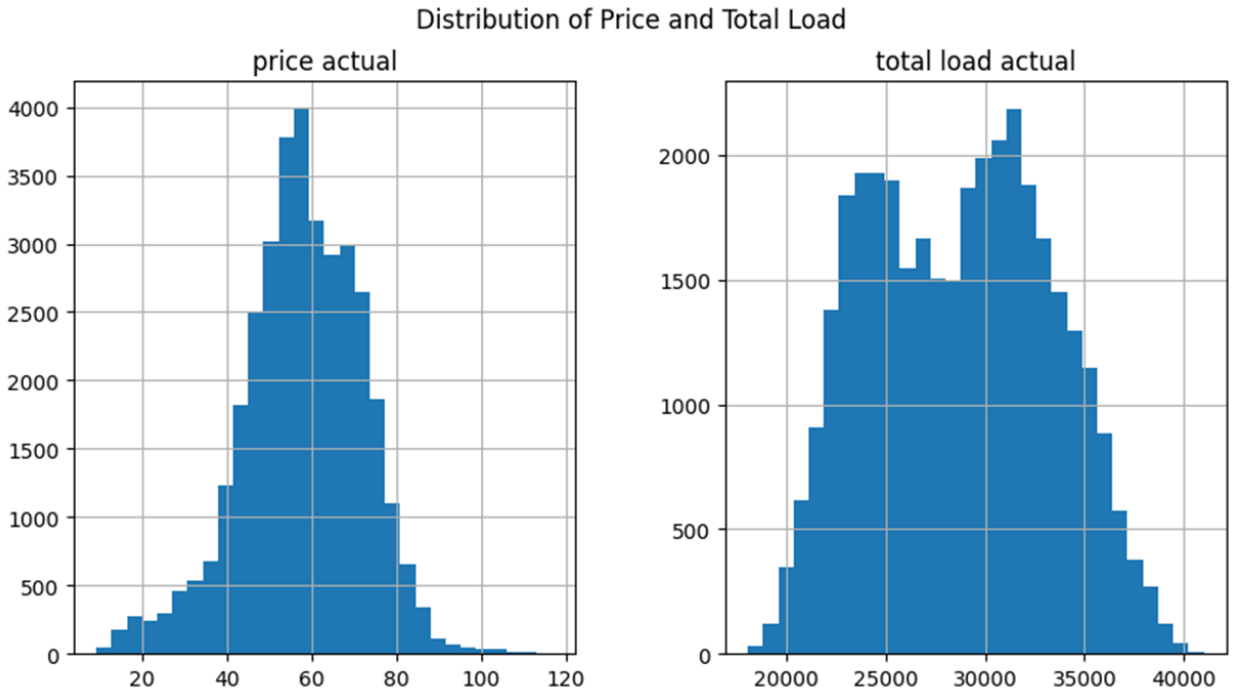

Fig.2.2 Electricity Price and Load Demand Distribution

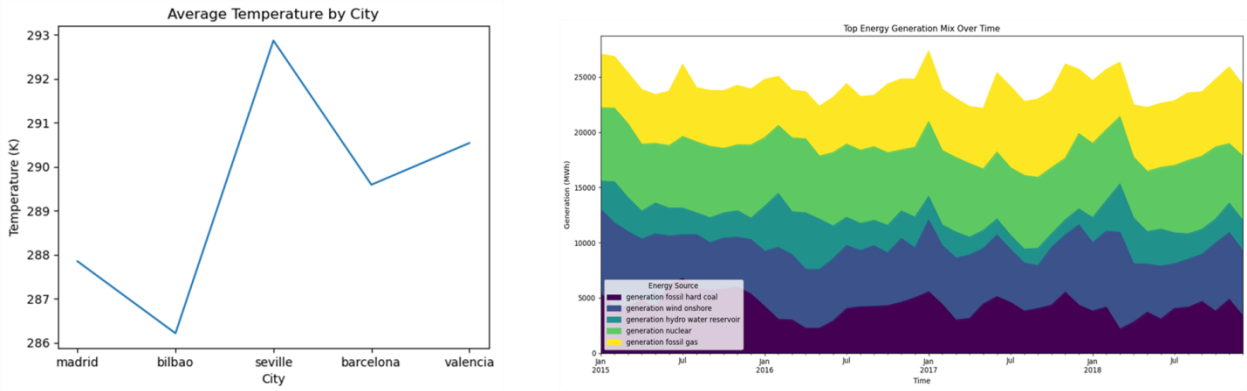

Fig.2.3 Temperature by City and Top Energy Generation Sources

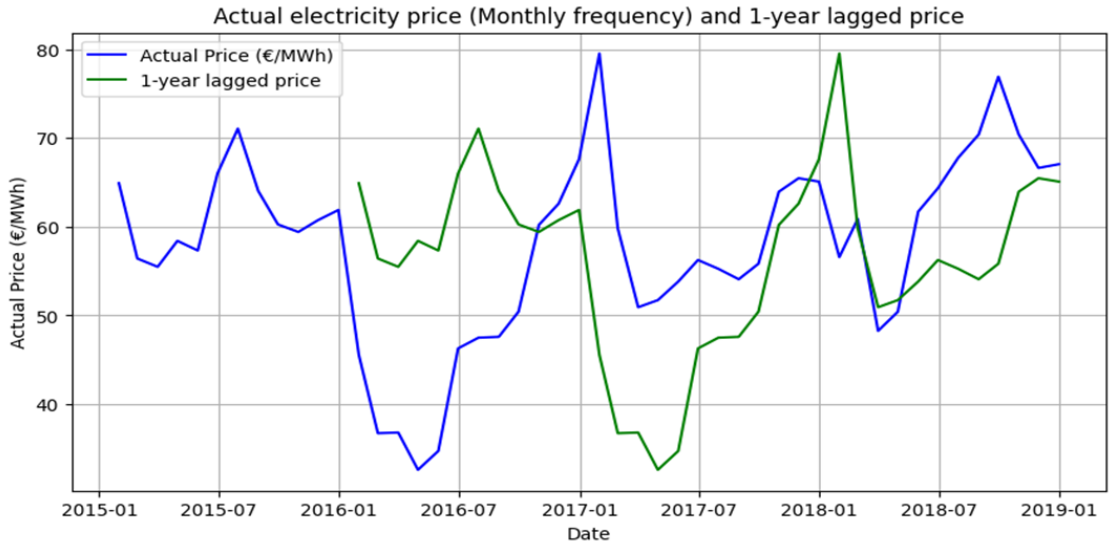

Fig.2.4 Actual electricity price vs 1-year lagged price

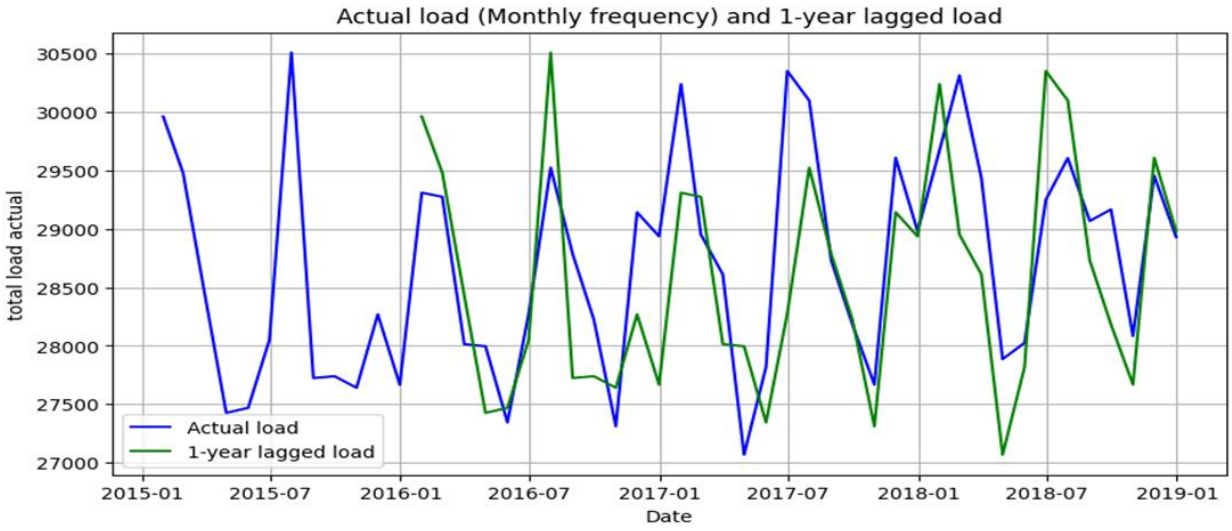

Fig. 2.5 Actual load vs 1-year lagged load

Fig. 6 depicts the relationship between the current electricity price and 1-year lagged electricity price value. We can infer that there is not much relationship between the current and the lagged price values. On the other hand, we can see a timely correlation between the current electricity load demand and the lagged electricity load demand from Fig.7.

II.4 Model Development

Several machine learning models were developed to predict energy prices and total load. These include Linear Regression, Random Forest, LSTM (Long Short-Term Memory), and XGBoost (Extreme Gradient Boosting).

II.5 Model Training and Evaluation:

The dataset was divided into training and test sets. The training set comprised three years of hourly data, totaling 26,304 rows (from 2015 to 2017). Various test sets such as 1 day, 1 week, 1 month, 6 months, and 1 year were created from the data related to the year 2018 and evaluated to measure the model performance across different test sizes.

For the XGBoost model, a grid search was conducted to find the optimal hyperparameters, including 'max_depth,' 'learning_rate,' and 'n_estimators. Various epoch sizes were experimented with for the LSTM model, and ultimately, a total of 75 epochs were chosen for training to minimize error. Models were evaluated using metrics such as Normalized Mean Absolute Error (NMAE), Normalized Mean Squared Error (NMSE), and R-squared (R2). The evaluation was conducted across various test sizes ranging from 1 day to 1 year. Fig.8 shows the methodology followed for the project.

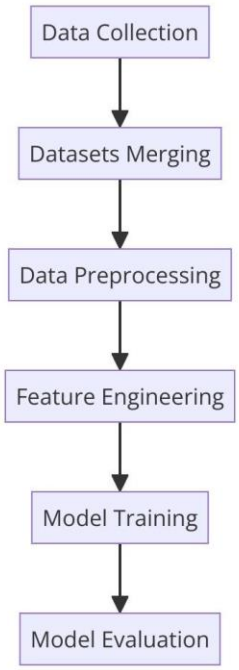

Fig.2.6 Workflow Diagram

III. RESULTS

III.1 Electricity Load Prediction Results

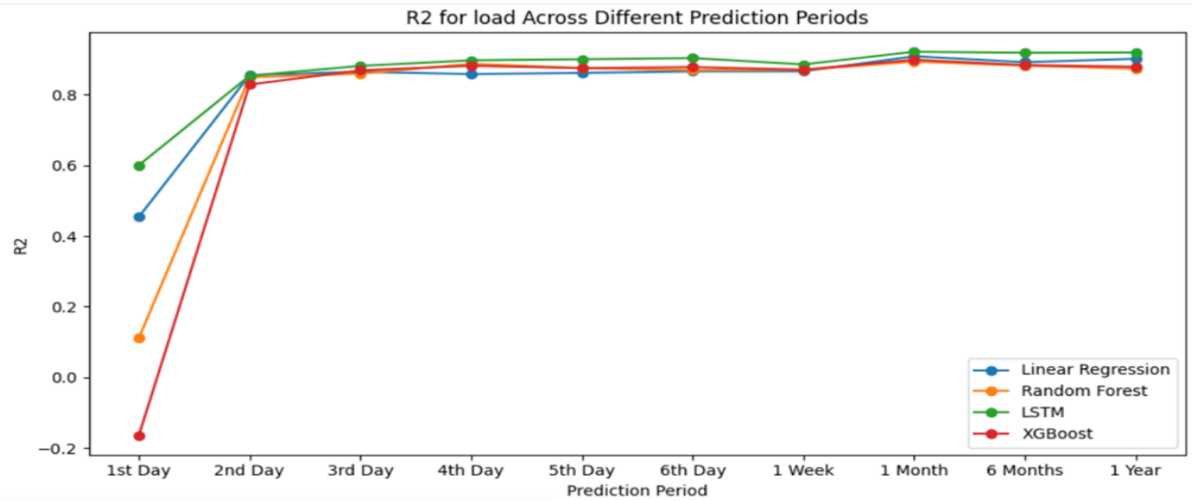

Fig.3.1 R2 scores of electricity load prediction

Electricity load values were predicted for the various test sets of the year 2018 using the selected machine learning models. Fig.9 shows the accuracy plot of all the models across the various test sizes. The accuracy was low for the test size of 1 day, but gradually improved as the test size increased and was almost constant with the test size. Among all the models, LSTM performed well for all the test sizes. A detailed comparison of the evaluation metrics of all the models can be found in Appendix I.

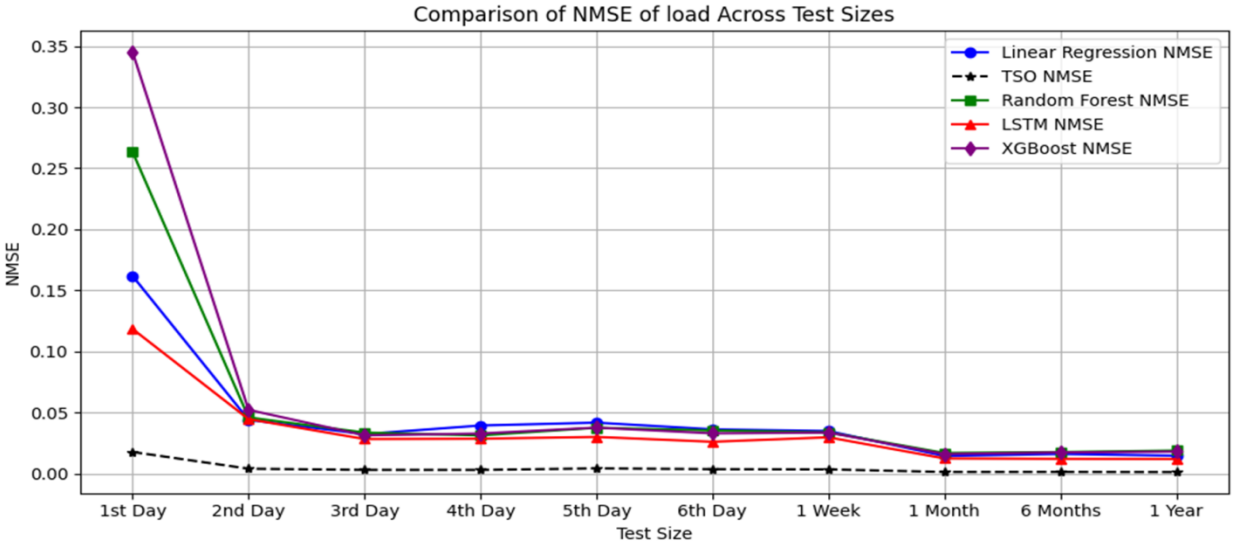

Fig.3.2 Load prediction comparison with TSO

We compared our electricity load prediction with TSO (Eléctrica, n.d.) predicted values as shown in Fig. 10. We can see that TSO predictions are good compared to our model predictions. One reason for this might be the continuously updating training set of the TSO models, while our models training is confined to 3 years(2015 to 2017) data,

III.2 Electricity Price Prediction Results

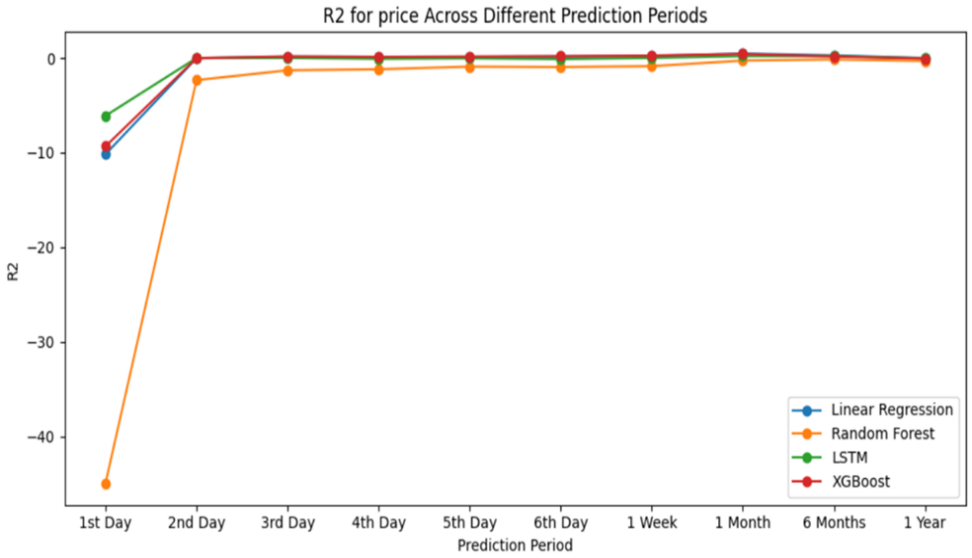

Fig. 3.3 R2 Scores of Electricity Load Prediction

Electricity price values were predicted for the various test sets of the year 2018 using the selected machine learning models. Fig.10 shows the accuracy plot of all the models across the various test sizes. The accuracy was low for the test size of 1 day, but gradually improved as the test size increased and was almost constant with the test size. Among all the models, XG Boost performed well for most of the test sizes. A detailed comparison of the evaluation metrics of all the models can be found in Appendix II.

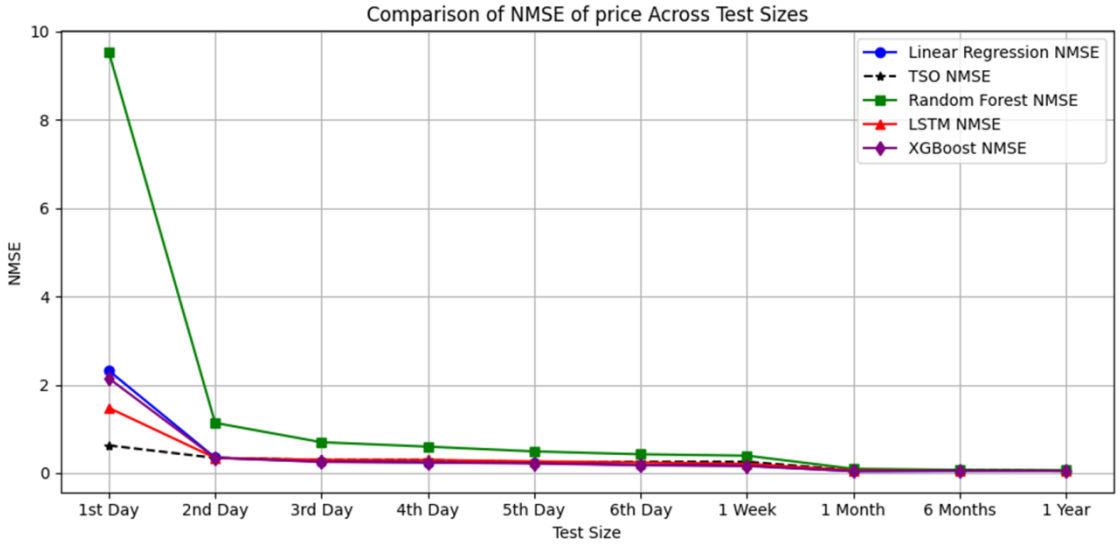

Fig.3.4 Price prediction comparison with TSO

We compared our electricity price prediction with TSO predicted values as shown in Fig. 11. We can see that TSO predictions are good compared to our model predictions. One reason for this might be the continuously updating training set of the TSO models, while our models training is confined to 3 years(2015 to 2017) data,

IV. DISCUSSIONS AND CONCLUSIONS

Our study demonstrates the substantial potential of various machine learning models in forecasting energy prices and load demands. The detailed preprocessing, feature engineering, and application of models such as Linear Regression, Random Forest, XGBoost, and LSTM have shown significant predictive accuracy. Normalizing data using the MinMaxScaler was essential for optimal model performance. Each model exhibited unique strengths: Linear Regression provided a baseline, Random Forest captured non-linear relationships, XGBoost was robust and efficient, and LSTM excelled in handling sequential data. The integration of weather data proved crucial in enhancing prediction accuracy, underscoring the impact of external variables on energy metrics. This integration allowed our models to make more precise forecasts compared to traditional methods.

In conclusion, this study highlights the importance of integrating advanced machine learning models and external variables in energy forecasting like weather data. Models like XGBoost and LSTM, offer superior predictive performance. Continuous refinement of these models and incorporating a broader range of influencing factors will likely yield more accurate and reliable forecasts, aiding in the optimization of energy management and planning strategies.

V. FUTURE WORK

Future research could explore combining different models such as LSTM with XGBoost and LSTM with Neural Networks to capture various data trends more effectively. These hybrid models can be fine-tuned to achieve better results compared to individual models. Moreover, incorporating external factors like crude oil prices, geopolitical events, and economic indicators could further improve predictions. Expanding the dataset to include more recent data and additional locations can enhance model robustness. Implementing real-time forecasting systems that update predictions as new data becomes available would also be valuable, requiring the integration of streaming data processing frameworks to handle continuous data flows for up-to-date and accurate predictions.

VI. REFERENCES

Electricity Consumption Prediction using Energy Data, Socio-economic and Weather Indicators. A Case Study of Spain. (2021, November 11). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9646220

ESIOS Electricidad. (n.d.). https://www.esios.ree.es/en

Hourly energy demand generation and weather. (2019, October 10).

https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-

and-weather/data

Kerenhalevy. (n.d.). GitHub - kerenhalevy/EDA_Energy_Demand_Generation_and_Weather: The dataset contains hourly intervals of electrical consumption, generation, pricing and weather data in Spain. The project focused on Exploratory Data Analysis (EDA): data cleaning and updating, trends and relationship between variables. GitHub. https://github.com/kerenhalevy/EDA_Energy_Demand_Generation_and_Weather

Li, C., Chen, Z., Liu, J., Li, D., Gao, X., Di, F., Li, L., & Ji, X. (2019). Power Load Forecasting Based on the Combined Model of LSTM and XGBoost. Doi. https://doi.org/10.1145/3357777.3357792

Lu, N., Ouyang, Q., Li, Y., & Zou, C. (2024, March 6). Electrical Load Forecasting Model Using Hybrid LSTM Neural Networks with Online Correction. arXiv.org. https://arxiv.org/abs/2403.03898

Mahajan, P., Uddin, S., Hajati, F., & Moni, M. A. (2023). Ensemble Learning for Disease Prediction: A review. Healthcare, 11(12), 1808. https://doi.org/10.3390/healthcare11121808

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. WIREs Data Mining and Knowledge Discovery, 8(4). https://doi.org/10.1002/widm.1249

scikit-learn: machine learning in Python — scikit-learn 1.4.2 documentation. (n.d.).

https://scikit-learn.org/stable/

TensorFlow. (n.d.). *TensorFlow*. https://www.tensorflow.org/

Welcome to Python.org. (n.d.). Python.org. https://www.python.org/doc/

Wu, L., & Shahidehpour, M. (2014). A hybrid model for integrated day-ahead electricity price and load forecasting in smart grid. IET Generation, Transmission & Distribution, 8(12), 1937–1950. https://doi.org/10.1049/iet-gtd.2013.0927

Zheng, H., & Wu, Y. (2019). A XGBoost Model with Weather Similarity Analysis and Feature Engineering for Short-Term Wind Power Forecasting. Applied Sciences, 9(15), 3019. https://doi.org/10.3390/app9153019

VII. APPENDICES

APPENDIX VII.1

Electricity Load Evaluation Metrics											
Model	**Metric**	1st Day	2nd Day	3rd Day	4th Day	5th Day	6th Day	1 Week	1 Month	5 Months	1 Year
Linear Regression	**NMAE**	0.37	0.21	0.17	0.20	0.21	0.19	0.18	0.11	0.11	0.11
	NMSE	0.16	0.04	0.03	0.04	0.04	0.04	0.03	0.01	0.02	0.01
	R2	0.45	0.86	0.86	0.86	0.86	0.87	0.87	0.91	0.89	0.90
	TSO_NMSE	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Random Forest	**NMAE**	0.53	0.23	0.19	0.18	0.20	0.19	0.18	0.12	0.11	0.12
	NMSE	0.26	0.05	0.03	0.03	0.04	0.04	0.03	0.02	0.02	0.02
	R2	0.11	0.85	0.86	0.89	0.88	0.87	0.87	0.89	0.88	0.87
	TSO_NMSE	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LSTM	**NMAE**	0.33	0.22	0.17	0.17	0.18	0.16	0.17	0.10	0.09	0.09
	NMSE	0.12	0.04	0.03	0.03	0.03	0.03	0.03	0.01	0.01	0.01
	R2	0.60	0.85	0.88	0.90	0.90	0.90	0.89	0.92	0.92	0.92
	TSO_NMSE	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
XGBoost	**NMAE**	0.64	0.25	0.18	0.18	0.20	0.18	0.18	0.11	0.11	0.12
	NMSE	0.35	0.05	0.03	0.03	0.04	0.03	0.03	0.02	0.02	0.02
	R2	-0.16	0.83	0.87	0.88	0.87	0.88	0.87	0.90	0.88	0.88
	TSO_NMSE	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

1.1 Electricity Load Evaluation Metrics of all test sizes

APPENDIX VII.2

Electricity Price Evaluation Metrics											
Model	**Metric**	**1st Day**	**2nd Day**	**3rd Day**	**4th Day**	**5th Day**	**6th Day**	**1 Week**	**1 Month**	**5 Months**	**1 Year**
Linear Regression	**NMAE**	1.65	0.60	0.48	0.47	0.47	0.39	0.38	0.15	0.18	0.19
	NMSE	2.32	0.35	0.26	0.24	0.23	0.19	0.17	0.04	0.05	0.05
	R2	-10.18	-0.02	0.14	0.12	0.11	0.14	0.19	0.46	0.27	-0.01
	TSO_NMSE	0.63	0.34	0.30	0.30	0.25	0.26	0.26	0.06	0.07	0.05
Random Forest	**NMAE**	3.35	1.20	0.91	0.81	0.74	0.64	0.61	0.25	0.24	0.22
	NMSE	9.54	1.14	0.70	0.60	0.49	0.43	0.39	0.10	0.07	0.06
	R2	-44.95	-2.34	-1.31	-1.19	-0.92	-0.96	-0.87	-0.28	-0.16	-0.32
	TSO_NMSE	0.63	0.34	0.30	0.30	0.25	0.26	0.26	0.06	0.07	0.05
LSTM	**NMAE**	1.25	0.62	0.56	0.54	0.51	0.46	0.41	0.20	0.19	0.19
	NMSE	1.48	0.34	0.30	0.30	0.26	0.24	0.21	0.06	0.05	0.05
	R2	-6.11	-0.01	0.01	-0.09	-0.03	-0.11	0.01	0.19	0.20	-0.04
	TSO_NMSE	0.63	0.34	0.30	0.30	0.25	0.26	0.26	0.06	0.07	0.05
XGBoost	**NMAE**	1.54	0.58	0.48	0.48	0.46	0.39	0.37	0.17	0.20	0.20
	NMSE	2.15	0.35	0.25	0.24	0.22	0.18	0.16	0.04	0.05	0.05
	R2	-9.34	-0.02	0.17	0.12	0.15	0.19	0.25	0.42	0.15	-0.06
	TSO_NMSE	0.63	0.34	0.30	0.30	0.25	0.26	0.26	0.06	0.07	0.05

2.1 Electricity Price Evaluation Metrics of all test sizes

APPENDIX VII.3 Programming Language and Version

The programming code is written in Python and the Python version used is 3.12.3. Google Colab is used as an IDE(Integrated Development Environment) to run our Python code.

APPENDIX VII.4 Python Programming Code

Code snippet:

```
# Installing required libraries
!pip install tensorflow
!pip install xgboost


import os
import pandas as pd
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import xgboost as xgb
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from math import sqrt
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=(FutureWarning, UserWarning))
from tensorflow.keras.layers import Dense, LSTM, Conv1D, MaxPooling1D, TimeDistributed, Flatten,
Dropout, RepeatVector
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.decomposition import PCAPrice prediction comparison with TSO
from sklearn.model_selection import train_test_split
from math import sqrt

# %matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')

# Load energy data
energy_df = pd.read_csv(r"/content/drive/MyDrive/combined_ds/energy_dataset.csv",
parse_dates=['time'], index_col='time')

# Load weather dataset
weather_df = pd.read_csv(r"/content/drive/MyDrive/combined_ds/weather_features.csv",
parse_dates=['dt_iso'], index_col='dt_iso')

energy_df.head()

energy_df.index = pd.to_datetime(energy_df.index, utc=True).tz_localize(None)

# the energy_df has extra records from 2014 where as weather_df has values from 2015
# it would be ideal to remove the rows with the times that are not present in both dataframes
energy_df = energy_df[energy_df.index.year >=2015]

weather_df.head()

"""# energy_df"""

energy_df.head()

energy_df.info()

"""Dropping columns with NaN values."""

# Dropping 2 columns due to NaN values.
energy_df.drop(['generation hydro pumped storage aggregated', 'forecast wind offshore eday ahead'],
axis=1, inplace=True)

"""Dropping the columns with 0 values."""

# Confirming values of 6 columns are 0.
energy_df[['generation fossil coal-derived gas',
           'generation fossil oil shale',
           'generation fossil peat',
           'generation geothermal',
           'generation marine',
           'generation wind offshore']].value_counts()

# Dropping 6 columns due to 0 values
energy_df.drop(['generation fossil coal-derived gas',
           'generation fossil oil shale',
           'generation fossil peat',
           'generation geothermal',
           'generation marine',
           'generation wind offshore'], axis=1, inplace=True)
```

```
# checking for duplicate values
energy_df.index.has_duplicates

"""## Forecast Dataframe

creating forecast dataframe and adding columns like 'forecast solar day ahead', 'forecast wind onshore day ahead', 'total load forecast TSO', 'price day ahead' from energy dataframe to forecast dataframe.
"""

forecast=pd.DataFrame(energy_df[['total load forecast',
                                 'price day ahead']])
forecast.head()

forecast.index = pd.to_datetime(forecast.index, utc=True).tz_localize(None)

forecast.index.rename('hour', inplace=True)

forecast.head()

new_index=forecast.index

scaler = MinMaxScaler()
forecast = pd.DataFrame(scaler.fit_transform(forecast), columns=forecast.columns, index=new_index)



forecast.head()

target_datee=forecast.index[26305]

"""Deleting forecasted columns from energy dataframe"""

# Dropping 4 columns from energy dataframe as we have moved them into new datframe called 'forecast'.
energy_df.drop(['forecast solar day ahead',
        'forecast wind onshore day ahead',
        'total load forecast',
        'price day ahead'], axis=1, inplace=True)

# A total of 12 columns have been dropped.
energy_df.info()

"""Finding number of null values in each column of energy dataframe"""

energy_df.isnull().sum()

"""We will use forward filling method to fill the null values in each column. We can see that all the null values are filled using forward fill."""

energy_df.fillna(method='ffill', inplace=True)
energy_df.isnull().sum()
```

```
"""### it seems that even after ffill or bfill we are still seeing some missing values and that implies

### the dataframe must have continous missing values so the above methods wouldnt work, we have to proceed with interpolation
"""

# Linear interpolation
energy_df.interpolate(method='linear', inplace=True)
energy_df.isna().sum()

energy_df.shape

energy_df.describe()

"""# VISUALIZATIONS ON ENERGY DATASET"""

energy_sources = [
        "generation biomass",
        "generation fossil brown coal/lignite",
        "generation fossil gas",
        "generation fossil hard coal",
        "generation fossil oil",
        "generation hydro pumped storage consumption",
        "generation hydro run-of-river and poundage",
        "generation hydro water reservoir",
        "generation nuclear",
        "generation other",
        "generation other renewable",
        "generation solar",
        "generation waste",
        "generation wind onshore"
]

# Calculate correlations
corr = energy_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f")
plt.title('Correlation Heatmap Between Energy Sources')
plt.show()

import matplotlib.pyplot as plt

# Number of plots in each row
plots_per_row = 3

# Calculate the number of rows needed
num_columns = len(energy_df.columns)
num_rows = (num_columns + plots_per_row - 1) // plots_per_row
```

```
# Create a figure and set of subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=plots_per_row, figsize=(20, num_rows * 5))

# Flatten axes array if more than one row
axes = axes.flatten() if num_rows > 1 else [axes]

# Plot data
for idx, column in enumerate(energy_df.columns):
        energy_df[column].plot(kind='box', ax=axes[idx])
        axes[idx].set_title(f'Distribution of {column} Energy Generation')
        axes[idx].set_xlabel('Generation')


for idx in range(len(energy_df.columns), len(axes)):
        axes[idx].set_visible(False)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()

energy_mix = energy_df[['generation biomass', 'generation fossil brown coal/lignite',
        'generation fossil gas', 'generation fossil hard coal',
        'generation fossil oil', 'generation hydro pumped storage consumption',
        'generation hydro run-of-river and poundage',
        'generation hydro water reservoir', 'generation nuclear',
        'generation other', 'generation other renewable', 'generation solar',
        'generation waste', 'generation wind onshore', 'total load actual',
        'price actual']]
energy_mix_resampled = energy_mix.resample('M').mean()

# Plotting
energy_mix_resampled.plot(kind='area', stacked=True, figsize=(14, 7))
plt.title('Energy Generation Mix Over Time')
plt.ylabel('Generation (MWh)')
plt.xlabel('Time')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()

# Calculating the correlation matrix
correlation_matrix = energy_df.corr()

# Correlations with total load and price
corr_with_load = correlation_matrix['total load actual'].sort_values(ascending=False)
corr_with_price = correlation_matrix['price actual'].sort_values(ascending=False)

# Print top correlated sources with total load and price for inspection
print("Top correlated sources with total load:\n", corr_with_load.head(6))
print("\nTop correlated sources with price:\n", corr_with_price.head(6))

# Calculating the average generation for each source to find the highest generating sources
```

```
average_generation = energy_df.loc[:, 'generation biomass':'generation wind
onshore'].mean().sort_values(ascending=False)

# Print top energy-generating sources for inspection
print("\nTop energy-generating sources:\n", average_generation.head())

#energy_sources = ["generation biomass", "generation fossil gas", "generation nuclear", "generation
solar", "generation wind onshore"]
energy_sources = ["generation nuclear", "generation fossil gas", "generation wind onshore", "generation
fossil hard coal",   "generation hydro water reservoir"]
energy_df[energy_sources].plot(figsize=(15, 7))
plt.title("Energy Generation Over Time by top corelated sources")
plt.ylabel("Generation (MWh)")
plt.xlabel("Time")
plt.legend()
plt.show()

top_sources_list = ["generation nuclear", "generation fossil gas", "generation wind onshore", "generation
fossil hard coal",   "generation hydro water reservoir"]
# Filter the dataframe for these top sources
energy_top_sources = energy_df[top_sources_list]

# Resample to monthly data for smoother visualization
energy_top_sources_resampled = energy_top_sources.resample('M').mean()

# Plotting
energy_top_sources_resampled.plot(kind='area', stacked=True, figsize=(14, 7), colormap='viridis')
plt.title('Top Energy Generation Mix Over Time')
plt.ylabel('Generation (MWh)')
plt.xlabel('Time')
plt.legend(title='Energy Source')
plt.tight_layout()
plt.show()

energy_df[['price actual', 'total load actual']].hist(bins=30, figsize=(10, 5))
plt.suptitle("Distribution of Price and Total Load")
plt.show()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Box plot for 'price actual' on the first subplot
ax1.boxplot(energy_df['price actual'])
ax1.set_title('Price Actual')
ax1.set_ylabel('Price (€)')
ax1.set_xticklabels(['Price'])

# Box plot for 'total load actual' on the second subplot
ax2.boxplot(energy_df['total load actual'])
ax2.set_title('Total Load Actual')
ax2.set_ylabel('Total Load (MW)')
ax2.set_xticklabels(['Load'])
```

```
# Overall figure title
plt.suptitle('Box Plot for Price and Total Load')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plot
plt.show()

# Check for any negative values in specified columns
negative_price = (energy_df['price actual'] < 0).any()
negative_load = (energy_df['total load actual'] < 0).any()

# Print results
print(f"Negative values in 'price actual': {'Yes' if negative_price else 'No'}")
print(f"Negative values in 'total load actual': {'Yes' if negative_load else 'No'}")

"""

# Weather Dataset"""

weather_df.head()

#converting the date of weather dataframe into utc timezone, same as energy dataframe
weather_df.index = pd.to_datetime(weather_df.index, utc=True).tz_localize(None)
weather_df.head()

weather_df.info()

weather_df.isnull().sum()

#dropping weather_icon column from weather dataset
weather_df.drop(columns=['weather_icon','weather_main', 'weather_description','weather_id'],
inplace=True)

import matplotlib.pyplot as plt
import pandas as pd

# Filter out non-numeric columns
numeric_cols = weather_df.select_dtypes(include=[np.number]).columns

# Define the number of plots per row
plots_per_row = 4

# Get the total number of numeric columns
num_columns = len(numeric_cols)

# Calculate the number of rows needed based on the number of numeric columns
num_rows = (num_columns + plots_per_row - 1) // plots_per_row  # Rounds up if not an even multiple
```

```
# Create a figure with multiple subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=plots_per_row, figsize=(20, 5 * num_rows))

# Flatten axes array to simplify the indexing in the loop (works for both single and multiple rows)
axes = axes.flatten()

# Loop through each numeric column in the DataFrame
for idx, column in enumerate(numeric_cols):
        # Plot the box plot for each numeric column
        weather_df[column].plot(kind='box', ax=axes[idx])
        axes[idx].set_title(f'Box Plot of {column}')
        axes[idx].set_ylabel(column)
        axes[idx].set_xlabel('Value')

# Turn off any unused axes
for idx in range(num_columns, len(axes)):
        axes[idx].set_visible(False)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()

"""from the above plot we can see that there are few outliers in rain_3h,rain_1h, pressure and wind speed
we will try to remove to increase the efficiency of the model


"""

#applying  constraints to windspeed column
weather_df['wind_speed'] = weather_df['wind_speed'].apply(lambda x: 60 if x > 60 else (0 if x < 0 else
x))

#applying constraints to rain_3h column
weather_df['rain_3h'] = weather_df['rain_3h'].apply(lambda x: 0.3 if x > 0.3 else (0 if x < 0 else x))

#applying constraints to rain column
weather_df['rain_1h'] = weather_df['rain_1h'].apply(lambda x: 4 if x > 4  else x)

#applying constraints to snow column
weather_df['snow_3h'] = weather_df['snow_3h'].apply(lambda x: 10 if x > 10 else x)

weather_conditions = ['temp', 'pressure', 'humidity', 'wind_speed', 'clouds_all']

for condition in weather_conditions:
        plt.figure(figsize=(10, 4))
        weather_df[condition].hist(bins=30)
        plt.title(f'Distribution of {condition}')
        plt.xlabel(condition)
        plt.ylabel('Frequency')
        plt.show()
```

```
weather_df['city_name'].value_counts()

# filtering and creating new dataframes as per city

madird_df = weather_df[weather_df['city_name'] == 'Madrid']
bilbao_df = weather_df[weather_df['city_name'] == 'Bilbao']
seville_df = weather_df[weather_df['city_name'] == 'Seville']
barcelona_df = weather_df[weather_df['city_name'] == ' Barcelona']
valencia_df = weather_df[weather_df['city_name'] == 'Valencia']
valencia_df.tail()

# Renaming columns with the city name prefix
# dropping city_name column as if is no longer needed

new_column_madrid = ['madrid' + '_' + column for column in madird_df.columns]
madird_df.columns = new_column_madrid
madird_df.rename(columns={'madrid_dt_iso': '_dt_iso'}, inplace=True)
madird_df.drop(columns=['madrid_city_name'], inplace=True)



new_column_bilbao = ['bilbao' + '_' + column for column in bilbao_df.columns]
bilbao_df.columns = new_column_bilbao
bilbao_df.rename(columns={'bilbao_dt_iso': '_dt_iso'}, inplace=True)
bilbao_df.drop(columns=['bilbao_city_name'], inplace=True)



new_column_seville = ['seville' + '_' + column for column in seville_df.columns]
seville_df.columns = new_column_seville
seville_df.rename(columns={'seville_dt_iso': '_dt_iso'}, inplace=True)
seville_df.drop(columns=['seville_city_name'], inplace=True)



new_column_barcelona = ['barcelona' + '_' + column for column in barcelona_df.columns]
barcelona_df.columns = new_column_barcelona
barcelona_df.rename(columns={'barcelona_dt_iso': '_dt_iso'}, inplace=True)
barcelona_df.drop(columns=['barcelona_city_name'], inplace=True)



new_column_valencia = ['valencia' + '_' + column for column in valencia_df.columns]
valencia_df.columns = new_column_valencia
valencia_df.rename(columns={'valencia_dt_iso': '_dt_iso'}, inplace=True)
valencia_df.drop(columns=['valencia_city_name'], inplace=True)



valencia_df.head()
```

```
# printing the shapes of all city dataframes in order to see whether all the shapes match for proper merging
print(f'Shape of all the city data is {madird_df.shape}, {bilbao_df.shape}, {seville_df.shape}, {barcelona_df.shape}, {valencia_df.shape}')

# merging all the city dataframes into single weather dataframes
merged_df = pd.merge(madird_df, bilbao_df, left_index=True, right_index=True, how='inner')
merged_df1 = pd.merge(merged_df, seville_df, left_index=True, right_index=True, how='inner')
merged_df2 = pd.merge(merged_df1, barcelona_df, left_index=True, right_index=True, how='inner')
merged_df3 = pd.merge(merged_df2, valencia_df, left_index=True, right_index=True, how='inner')

merged_df3.head()

#naming the merged weather city dataframes as weather_df_merged
weather_df_merged=merged_df3

#final dataframe from weather sheet after all modifications
weather_df_merged.head()

weather_df_merged.describe()

cities = ['madrid', 'bilbao', 'seville', 'barcelona', 'valencia']
avg_temps = [weather_df_merged[f'{city}_temp'].mean() for city in cities]

plt.plot(cities, avg_temps)
plt.title("Average Temperature by City")
plt.ylabel("Temperature (K)")
plt.xlabel("City")
plt.show()

"""# Merging Energy and Weather dataframe"""

energy_weather_df = pd.merge(energy_df, weather_df_merged, left_index=True, right_index=True, how='inner')
energy_weather_df.head()

energy_weather_df.head()

# Reset the index and rename the index column
energy_weather_df.reset_index(inplace=True)
energy_weather_df.rename(columns={'index': 'hour'}, inplace=True)

energy_weather_df=energy_weather_df.set_index('hour')

energy_weather_df.head()

energy_weather_df = energy_weather_df[~energy_weather_df.index.duplicated(keep='first')]
monthly_load = energy_weather_df['total load actual'].asfreq('M')

# Checking for duplicate rows in the DataFrame
duplicates = energy_weather_df.duplicated()
```

```
# Counting the number of duplicate rows
num_duplicates = duplicates.sum()

# Displaying the number of duplicates
print(f"Number of duplicate rows: {num_duplicates}")

# Optionally, to see the duplicate rows:
if num_duplicates > 0:
        print("Duplicate rows:")
        print(energy_weather_df[duplicates])

monthly_load = energy_weather_df['total load actual'].resample('M').mean()
shifted = monthly_load.shift(12)

# Create a plot
plt.figure(figsize=(10, 5))
plt.plot(monthly_load, label='Actual load ', color='blue')
plt.plot(shifted, label='1-year lagged load', color='green')
plt.title('Actual load (Monthly frequency) and 1-year lagged load')
plt.ylabel('total load actual')
plt.xlabel('Date')
plt.legend()
plt.grid(True)
plt.show()

monthly_price = energy_weather_df['price actual'].resample('M').mean()
shifted = monthly_price.shift(12)

# Create a plot
plt.figure(figsize=(10, 5))
plt.plot(monthly_price, label='Actual price ', color='blue')
plt.plot(shifted, label='1-year lagged price', color='green')
plt.title('Actual price (Monthly frequency) and 1-year lagged price')
plt.ylabel('price actual')
plt.xlabel('Date')
plt.legend()
plt.grid(True)
plt.show()

correlations = energy_weather_df.corr(method='pearson')
print(correlations['price actual'].sort_values(ascending=False).to_string())

energy_weather_df = energy_weather_df.drop(['barcelona_snow_3h', 'seville_snow_3h'],
                axis=1)

# Plot pair wise correlation matrix

correlations = energy_weather_df.corr(method='pearson')
fig = plt.figure(figsize=(24,24))
sns.heatmap(correlations, annot=True, fmt='.2f')
```

```
plt.title('Pair wise correlation')
plt.show()

for i in range(len(energy_weather_df)):
        position = energy_weather_df.index[i]
        hour = position.hour
        weekday = position.weekday()
        month = position.month
        energy_weather_df.loc[position, 'hour'] = hour
        energy_weather_df.loc[position, 'weekday'] = weekday
        energy_weather_df.loc[position, 'month'] = month



energy_weather_df.head(5)

for i in range(len(energy_weather_df)):
        position = energy_weather_df.index[i]
        weekday = position.weekday()
        if (weekday == 6):
        energy_weather_df.loc[position, 'weekday'] = 2
        elif (weekday == 5):
        energy_weather_df.loc[position, 'weekday'] = 1
        else:
        energy_weather_df.loc[position, 'weekday'] = 0


energy_weather_df.head(5)
energy_weather_df.shape

"""# Model Development"""

target_date=energy_weather_df.index[26305]
print(f'target_date={target_date}')
#energy_weather_df.loc[target_date]

training_end_idx = 26304

"""## Total Load Predict Function"""

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from tqdm import tqdm  # Import tqdm for the progress bar
```

```
def normalize_data(df):
	scaler_X = MinMaxScaler()
	scaler_y = MinMaxScaler()
	X_scaled = scaler_X.fit_transform(df.drop(columns=['total load actual']))
	y_scaled = scaler_y.fit_transform(df[['total load actual']])
	return X_scaled, y_scaled, scaler_X, scaler_y

def evaluate_models(energy_weather_df, forecast, training_end_idx):
	# Define detailed test sizes
	test_sizes = {
	'1st Day': 24,
	'2nd Day': 24 * 2,
	'3rd Day': 24 * 3,
	'4th Day': 24 * 4,
	'5th Day': 24 * 5,
	'6th Day': 24 * 6,
	'1 Week': 7 * 24,
	'1 Month': 30 * 24,
	'6 Months': 180 * 24,
	'1 Year': len(energy_weather_df) - training_end_idx  # All remaining data
	}
	results = {}
	X_scaled, y_scaled, _, _ = normalize_data(energy_weather_df)
	y = y_scaled.flatten()
	X_train = X_scaled[:training_end_idx]
	y_train = y[:training_end_idx]

	# Initialize models
	models = {
	'Linear Regression': LinearRegression(),
	'Random Forest': RandomForestRegressor(),
	'LSTM': Sequential([
	LSTM(50, return_sequences=True, input_shape=(1, X_train.shape[1])),
	LSTM(50),
	Dense(1)
	])
	}
	models['LSTM'].compile(optimizer='adam', loss='mean_squared_error')

	xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
	param_grid = {
	'max_depth': [3, 7],
	'learning_rate': [0.02, 0.05],
	'n_estimators': [100, 150]
	}
	grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=2,
scoring='neg_mean_squared_error', verbose=2, n_jobs=1)
	grid_search.fit(X_train, y_train)
	models['XGBoost'] = grid_search.best_estimator_
```

```
        for name, test_size in tqdm(test_sizes.items(), desc="Evaluating models"):
        test_end_idx = training_end_idx + test_size
        X_test = X_scaled[training_end_idx:test_end_idx]
        y_test = y[training_end_idx:test_end_idx]
        tso_load = forecast['total load forecast'][training_end_idx:test_end_idx]
        X_test_lstm = np.array(X_test).reshape(-1, 1, X_test.shape[1])

        results[name] = {}
        for model_name, model in models.items():
        if model_name == 'LSTM':
        model.fit(X_train.reshape(-1, 1, X_train.shape[1]), y_train, epochs=75, batch_size=32,
verbose=0)
        y_pred = model.predict(X_test_lstm).flatten()
        else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results[name][model_name] = {
        'NMAE': mean_absolute_error(y_test, y_pred) / np.mean(y_test),
        'NMSE': mean_squared_error(y_test, y_pred) / np.mean(np.square(y_test)),
        'R2': r2_score(y_test, y_pred),
        'TSO_NMSE': mean_squared_error(y_test, tso_load) / np.mean(np.square(y_test))
        }
        print(grid_search.best_params_)
        print(grid_search.best_score_)
        return results, models

# Set the training_end_idx
training_end_idx = 26304  # Updated index

# Call the evaluate_models function
load_evaluation_results, models = evaluate_models(energy_weather_df, forecast, training_end_idx)

# Print the evaluation results
for test_size, test_results in load_evaluation_results.items():
        print(f"Test Size: {test_size}")
        for model_name, metrics in test_results.items():
        print(f"{model_name}:")
        for metric_name, value in metrics.items():
        print(f"  {metric_name}: {value}")
        print()

# Create plots for each metric across different prediction periods
metrics_to_plot = ['NMAE', 'NMSE', 'R2']
colors = ['blue', 'green', 'red']
fig, axs = plt.subplots(len(metrics_to_plot), 1, figsize=(10, 15))

for i, metric in enumerate(metrics_to_plot):
        for model_name in models.keys():
        metric_values = [load_evaluation_results[test_size][model_name][metric] for test_size in
load_evaluation_results]
```

```
        test_sizes = list(load_evaluation_results.keys())
        axs[i].plot(test_sizes, metric_values, marker='o', label=model_name)

        axs[i].set_title(f'{metric} for load Across Different Prediction Periods')
        axs[i].set_xlabel('Prediction Period')
        axs[i].set_ylabel(metric)
        axs[i].legend()

plt.tight_layout()
plt.show()

# Create plots for each metric across different prediction periods
metrics_to_plot = ['NMAE', 'NMSE', 'R2']
colors = ['blue', 'green', 'red', 'brown']
fig, axs = plt.subplots(len(metrics_to_plot), 1, figsize=(10, 15))

for i, metric in enumerate(metrics_to_plot):
        for model_name in models.keys():
        metric_values = [load_evaluation_results[test_size][model_name][metric] for test_size in
load_evaluation_results]
        test_sizes = list(load_evaluation_results.keys())
        axs[i].plot(test_sizes, metric_values, marker='o', label=model_name)

        axs[i].set_title(f'{metric} for load Across Different Prediction Periods')
        axs[i].set_xlabel('Prediction Period')
        axs[i].set_ylabel(metric)
        axs[i].legend()

plt.tight_layout()
plt.show()

"""## Predicted Load Comparison with TSO"""

import matplotlib.pyplot as plt

def compare_tso_with_other_models_line_plot(load_evaluation_results):
        # This function will plot line comparisons of TSO_NMSE with other models' NMSE for each test
size.
        test_sizes = list(load_evaluation_results.keys())
        models = list(next(iter(load_evaluation_results.values())).keys())

        # Prepare the figure
        fig, ax = plt.subplots(figsize=(10, 5))

        markers = ['o', 's', '^', 'd', '*']
        colors = ['blue', 'green', 'red', 'purple', 'orange']

        for i, model in enumerate(models):
        nmse_values = [load_evaluation_results[test_size][model]['NMSE'] for test_size in test_sizes]
        tso_nmse_values = [load_evaluation_results[test_size][model]['TSO_NMSE'] for test_size in
test_sizes]
```

```
        # Line plot for model NMSE
        ax.plot(test_sizes, nmse_values, marker=markers[i % len(markers)], color=colors[i %
len(colors)], label=f'{model} NMSE')

        # Line plot for TSO NMSE
        if model == models[0]:
        ax.plot(test_sizes, tso_nmse_values, marker=markers[-1], color='black', linestyle='--', label='TSO
NMSE')

        ax.set_xlabel('Test Size')
        ax.set_ylabel('NMSE')
        ax.set_title('Comparison of NMSE of load Across Test Sizes')
        ax.legend()
        ax.grid(True)

        plt.tight_layout()
        plt.show()


compare_tso_with_other_models_line_plot(load_evaluation_results)



import matplotlib.pyplot as plt
import numpy as np
from itertools import cycle
from tqdm import tqdm  # Import tqdm for the progress bar

# Define metrics to plot
metrics_to_plot = ['NMAE', 'NMSE', 'R2']
# Define a vibrant color palette
colors = cycle(['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'])
# Prepare the figure layout
fig, axs = plt.subplots(len(metrics_to_plot), 1, figsize=(12, 18))

# Padding between groups of bars
bar_width = 0.25
# Define additional space between groups
group_spacing = 0.1

prediction_periods = sorted(load_evaluation_results.keys())
num_models = len(models)
# Array of prediction period indices, modified to include group spacing
index = np.arange(len(prediction_periods)) * (num_models * bar_width + group_spacing)

# Iterate over metrics with a progress bar
for i, metric in enumerate(tqdm(metrics_to_plot, desc="Processing Metrics")):
        for j, model_name in enumerate(models.keys()):
        # Collect metric values for each prediction period, ensuring all are included
```

```
        metric_values = [load_evaluation_results.get(period, {}).get(model_name, {}).get(metric,
np.nan) for period in prediction_periods]
        # Calculate the position for the bars for this model
        positions = index + bar_width * j

        axs[i].bar(positions, metric_values, bar_width, label=model_name, color=next(colors))

        axs[i].set_title(f'{metric} Across Different Prediction Periods', fontsize=16)
        axs[i].set_xlabel('Prediction Period', fontsize=14)
        axs[i].set_ylabel(metric, fontsize=14)
        axs[i].set_xticks(index + bar_width * (num_models - 1) / 2)
        axs[i].set_xticklabels(prediction_periods, fontsize=12)
        axs[i].legend(fontsize=12)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()



"""##Total Price Predict Function"""

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from tqdm import tqdm  # Import tqdm for the progress bar

def normalize_data(df):
        scaler_X = MinMaxScaler()
        scaler_y = MinMaxScaler()
        X_scaled = scaler_X.fit_transform(df.drop(columns=['price actual']))
        y_scaled = scaler_y.fit_transform(df[['price actual']])
        return X_scaled, y_scaled, scaler_X, scaler_y

def evaluate_models(energy_weather_df, forecast, training_end_idx):
        # Define detailed test sizes as requested
        test_sizes = {
        '1st Day': 24,
        '2nd Day': 24 * 2,
        '3rd Day': 24 * 3,
        '4th Day': 24 * 4,
        '5th Day': 24 * 5,
        '6th Day': 24 * 6,
        '1 Week': 7 * 24,
```

```
        '1 Month': 30 * 24,
        '6 Months': 180 * 24,
        '1 Year': len(energy_weather_df) - training_end_idx  # All remaining data
        }
        results = {}
        X_scaled, y_scaled, _, _ = normalize_data(energy_weather_df)
        y = y_scaled.flatten()
        X_train = X_scaled[:training_end_idx]
        y_train = y[:training_end_idx]

        # Initialize models
        models = {
        'Linear Regression': LinearRegression(),
        'Random Forest': RandomForestRegressor(),
        'LSTM': Sequential([
        LSTM(50, return_sequences=True, input_shape=(1, X_train.shape[1])),
        LSTM(50),
        Dense(1)
        ])
        }
        models['LSTM'].compile(optimizer='adam', loss='mean_squared_error')

        xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
        param_grid = {
        'max_depth': [3, 7],
        'learning_rate': [0.02, 0.05],
        'n_estimators': [100, 150]
        }
        grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=2,
scoring='neg_mean_squared_error', verbose=2, n_jobs=1)
        grid_search.fit(X_train, y_train)
        models['XGBoost'] = grid_search.best_estimator_

        for name, test_size in tqdm(test_sizes.items(), desc="Evaluating models"):
        test_end_idx = training_end_idx + test_size
        X_test = X_scaled[training_end_idx:test_end_idx]
        y_test = y[training_end_idx:test_end_idx]
        tso_price = forecast['price day ahead'][training_end_idx:test_end_idx]
        X_test_lstm = np.array(X_test).reshape(-1, 1, X_test.shape[1])

        results[name] = {}
        for model_name, model in models.items():
        if model_name == 'LSTM':
        model.fit(X_train.reshape(-1, 1, X_train.shape[1]), y_train, epochs=75, batch_size=32,
verbose=0)
        y_pred = model.predict(X_test_lstm).flatten()
        else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results[name][model_name] = {
```

```
            'NMAE': mean_absolute_error(y_test, y_pred) / np.mean(y_test),
            'NMSE': mean_squared_error(y_test, y_pred) / np.mean(np.square(y_test)),
            'R2': r2_score(y_test, y_pred),
            'TSO_NMSE': mean_squared_error(y_test, tso_price) / np.mean(np.square(y_test))
            }
            print("the best hyper parameters: ", grid_search.best_params_)
            print(grid_search.best_score_)
            return results, models

# Set the training_end_idx
training_end_idx = 26304  # Updated index

# Call the evaluate_models function
price_evaluation_results, models = evaluate_models(energy_weather_df, forecast, training_end_idx)

# Print the evaluation results
for test_size, test_results in price_evaluation_results.items():
            print(f"Test Size: {test_size}")
            for model_name, metrics in test_results.items():
            print(f"{model_name}:")
            for metric_name, value in metrics.items():
            print(f"  {metric_name}: {value}")
            print()

# Create plots for each metric across different prediction periods
metrics_to_plot = ['NMAE', 'NMSE', 'R2']
colors = ['blue', 'green', 'red','brown']
fig, axs = plt.subplots(len(metrics_to_plot), 1, figsize=(10, 15))

for i, metric in enumerate(metrics_to_plot):
            for model_name in models.keys():
            metric_values = [price_evaluation_results[test_size][model_name][metric] for test_size in
price_evaluation_results]
            test_sizes = list(price_evaluation_results.keys())
            axs[i].plot(test_sizes, metric_values, marker='o', label=model_name)

            axs[i].set_title(f'{metric} for price Across Different Prediction Periods')
            axs[i].set_xlabel('Prediction Period')
            axs[i].set_ylabel(metric)
            axs[i].legend()

plt.tight_layout()
plt.show()

"""## Predicted Price Comparision with TSO"""

import matplotlib.pyplot as plt

def compare_tso_with_other_models_line_plot(price_evaluation_results):
            # This function will plot line comparisons of TSO_NMSE with other models' NMSE for each test
size.
```

```
        test_sizes = list(price_evaluation_results.keys())
        models = list(next(iter(price_evaluation_results.values())).keys())

        # Prepare the figure
        fig, ax = plt.subplots(figsize=(10, 5))

        markers = ['o', 's', '^', 'd', '*']
        colors = ['blue', 'green', 'red', 'purple', 'orange']

        for i, model in enumerate(models):
        nmse_values = [price_evaluation_results[test_size][model]['NMSE'] for test_size in test_sizes]
        tso_nmse_values = [price_evaluation_results[test_size][model]['TSO_NMSE'] for test_size in
test_sizes]

        # Line plot for model NMSE
        ax.plot(test_sizes, nmse_values, marker=markers[i % len(markers)], color=colors[i %
len(colors)], label=f'{model} NMSE')

        # Line plot for TSO NMSE
        if model == models[0]:
        ax.plot(test_sizes, tso_nmse_values, marker=markers[-1], color='black', linestyle='--', label='TSO
NMSE')

        ax.set_xlabel('Test Size')
        ax.set_ylabel('NMSE')
        ax.set_title('Comparison of NMSE of price Across Test Sizes')
        ax.legend()
        ax.grid(True)

        plt.tight_layout()
        plt.show()


compare_tso_with_other_models_line_plot(price_evaluation_results)

import matplotlib.pyplot as plt

def compare_tso_with_both_models_line_plot(price_evaluation_results, load_evaluation_results):
        # Prepare the figure layout with two subplots
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 7), sharey=True)

        # Define common markers and colors
        markers = ['o', 's', '^', 'd', '*']
        colors = ['blue', 'green', 'red', 'purple', 'orange']

        # Plot for price
        test_sizes = list(price_evaluation_results.keys())
        models = list(next(iter(price_evaluation_results.values())).keys())
        for i, model in enumerate(models):
        nmse_values = [price_evaluation_results[test_size][model]['NMSE'] for test_size in test_sizes]
```

```
        tso_nmse_values = [price_evaluation_results[test_size][model]['TSO_NMSE'] for test_size in
test_sizes]
        ax1.plot(test_sizes, nmse_values, marker=markers[i % len(markers)], color=colors[i %
len(colors)], label=f'{model} NMSE')
        if model == models[0]:  # Plot TSO NMSE once
        ax1.plot(test_sizes, tso_nmse_values, marker=markers[-1], color='black', linestyle='--',
label='TSO NMSE')
        ax1.set_title('Comparison of NMSE of Price Across Test Sizes')
        ax1.set_xlabel('Test Size')
        ax1.set_ylabel('NMSE')
        ax1.legend()
        ax1.grid(True)

        # Plot for load
        test_sizes = list(load_evaluation_results.keys())
        models = list(next(iter(load_evaluation_results.values())).keys())
        for i, model in enumerate(models):
        nmse_values = [load_evaluation_results[test_size][model]['NMSE'] for test_size in test_sizes]
        tso_nmse_values = [load_evaluation_results[test_size][model]['TSO_NMSE'] for test_size in
test_sizes]
        ax2.plot(test_sizes, nmse_values, marker=markers[i % len(markers)], color=colors[i %
len(colors)], label=f'{model} NMSE')
        if model == models[0]:  # Plot TSO NMSE once
        ax2.plot(test_sizes, tso_nmse_values, marker=markers[-1], color='black', linestyle='--',
label='TSO NMSE')
        ax2.set_title('Comparison of NMSE of Load Across Test Sizes')
        ax2.set_xlabel('Test Size')
        ax2.legend()
        ax2.grid(True)

        plt.tight_layout()
        plt.show()


compare_tso_with_both_models_line_plot(price_evaluation_results, load_evaluation_results)
```