

## Contents

1	Introduction.....	1
2	Data Cleansing, Exploratory Data Analysis (EDA) and Feature Engineering .....	2
2.1	Time Series Analysis Summary .....	2
2.2	NA Data Values .....	3
2.3	Outliers .....	3
2.4	Feature engineering: Features creation using the datetime.....	3
2.5	Analysis of Average Active Power Consumption.....	4
2.6	Feature Engineering: Creating Lagged and Rolling Average Features Based on Temporal Dependencies.....	5
2.7	Target Shifting and Column Selection for Feature Engineering .....	6
2.8	Analysis of the Correlation Matrix.....	6
2.9	Removal of Redundant Description Column .....	7
2.10	Interaction features.....	7
3	Regression Model - LightGBM .....	8
3.1	Preliminary Steps for Modeling .....	8
3.2	Benchmarking Machine Learning Models .....	8
3.3	LightGBM Model Optimization and Final Evaluation for Active Power Prediction .....	8
3.4	Comparison of Actual vs. Predicted Active Power Over Time.....	9
3.5	Feature Importance Analysis for LightGBM Model.....	9
3.6	Final Model Training and Evaluation on Full Dataset.....	10
4	LSTM .....	11
5	Prophet Model.....	12
6	Other Model Alternatives for Time Series Forecasting .....	13
7	Deployment Solution.....	14
7.1	Containerization and Docker Setup.....	14
7.2	Serving the Model with FastAPI .....	14
7.3	Endpoint Functionality .....	14
7.4	Summary.....	14
8	Cloud Infrastructure Schema .....	15
9	Project Deliverables .....	16

## Figures

Figure 1 Initial dataset .....	2
Figure 2 Time series analysis summary.....	2
Figure 3 Outliers summary table.....	3
Figure 4 Average active power per Year, Season, Month and Weeday .....	4
Figure 5 Average Active Power per Hour reveals certain patterns .....	5
Figure 6 Average active power Daytime or Nighttime. Higher consumption during nighttime.....	5
Figure 7 ACF and PACF analysis of Active Power .....	6
Figure 8 Correlation matrix .....	7
Figure 9 Models evaluation. LightGBM emerged as the most promising model .....	8
Figure 10 LightGBM Actual vs Predicted values. The model does not capture extreme values. ....	9
Figure 11 Feature importance using the LightGBM model.....	10
Figure 12 LSTM Actual vs Predicted values. The model does not capture extreme values and is less efficient than the LightLGBM.....	11
Figure 13 Prophet Model Actual vs Predicted values. The model does not capture extreme values and is less efficient than the LightLGBM.....	12

## 1 Introduction

The project involves developing a predictive model for forecasting the next minute's active power consumption in a household based on historical energy and weather data. The dataset, collected over 14 months from a residence in Mexico, includes minute-level measurements of electrical characteristics—such as active power, current, voltage, apparent power, reactive power, and power factor—as well as environmental factors like temperature, humidity, and weather conditions.

Scope:

- **Data Analysis and Preprocessing:** Conduct thorough Exploratory Data Analysis (EDA) and data cleansing to prepare the data for modeling. This includes handling missing values, identifying and addressing outliers, and transforming the data for optimal model performance.
- **Feature Engineering:** Create new features that may capture additional information, such as interaction terms and groupings, to improve the model's predictive power.
- **Model Development:** Develop and evaluate various machine learning models to predict churn, using the insights and features derived from the data analysis and feature engineering phases.
- **Implementation and Recommendations:** Implement the chosen model.

## 2 Data Cleansing, Exploratory Data Analysis (EDA) and Feature Engineering

The goal is to gain a comprehensive understanding of the data (Figure 1), identify potential issues, and prepare the dataset for the subsequent modeling phase.

date	active_power	current	voltage	reactive_power	apparent_power	power_factor	main	description	temp	feels_like	temp_min	temp_max	pressure	humidity	speed	deg
2022-11-05T14:05:00.000000000	265.1	2.53	122.2	159.09	309.17	0.8575	CL	clear sky	24...	23.68	23.44	27.5	1013	39	0.0	0
2022-11-05T14:06:00.000000000	265.1	2.53	122.2	159.09	309.17	0.8575	CL	clear sky	24...	23.68	23.44	27.5	1013	39	0.0	0
2022-11-05T14:07:00.000000000	265.1	2.53	122.2	159.09	309.17	0.8575	CL	clear sky	24...	23.68	23.44	27.5	1013	39	0.0	0

Figure 1 Initial dataset

### 2.1 Time Series Analysis Summary

The time series plots (Figure 2) provide a clear view of the variations and patterns in the key features—active power, current, voltage, apparent power, and temperature—over the observed period. Active power and current exhibit frequent fluctuations, with noticeable peaks and troughs, suggesting variable power consumption possibly due to appliance usage patterns. Voltage shows a relatively stable trend with minor variations, which is typical in household power supply. Apparent power mirrors the fluctuations in active power, as expected, since they are closely related. Temperature follows a seasonal trend, with gradual increases and decreases that likely correlate with outdoor climate changes, potentially impacting energy consumption. Although we cannot draw clear conclusion this initial analysis reveals periodic trends and seasonal components, setting a foundation for selecting appropriate time-series models and feature engineering for further analysis.

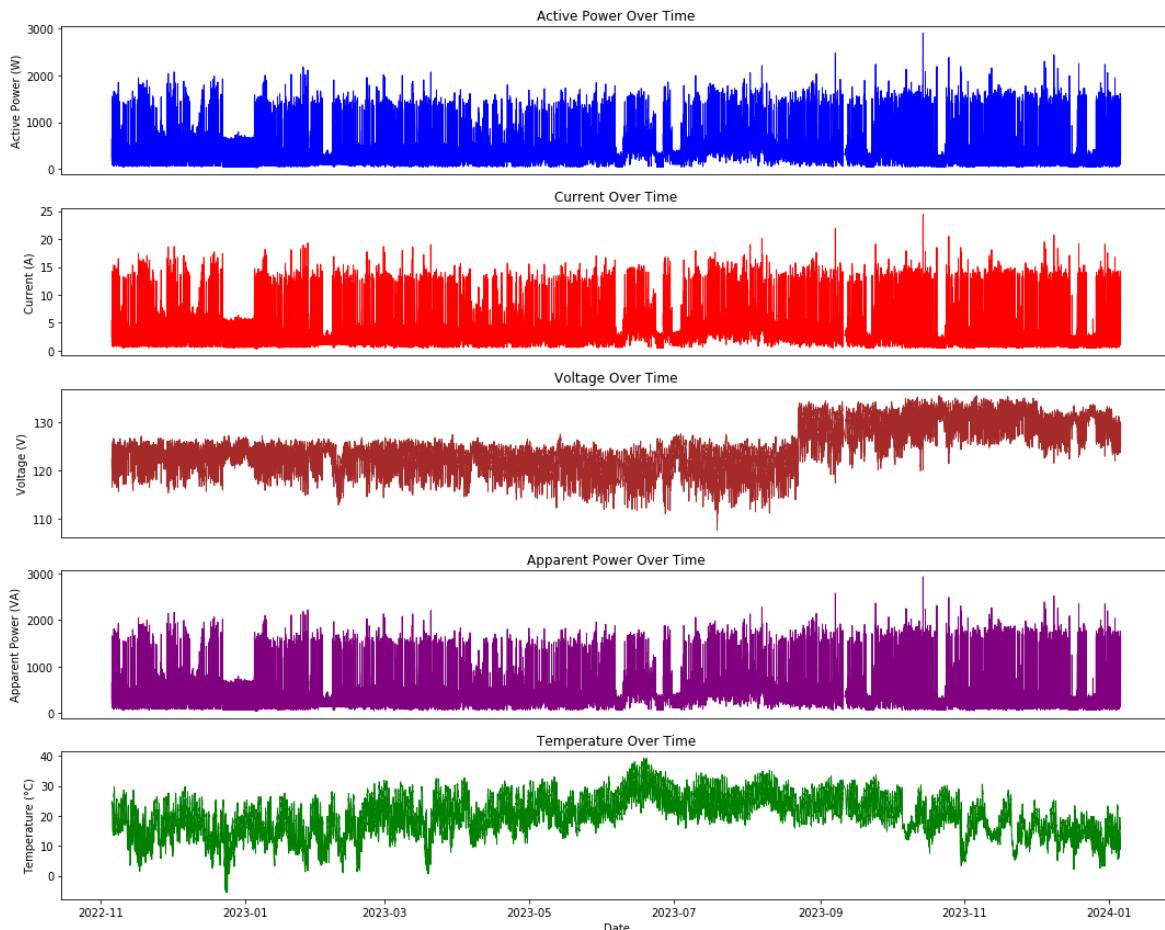


Figure 2 Time series analysis summary.

## 2.2 NA Data Values

The dataset contains no NA or missing values directly recorded in the data, as confirmed by checking for null entries across all columns. However, according to the documentation, there are gaps in the time series due to missing data points, amounting to approximately 1.43% of the total dataset. These gaps range from 1 to 1,572 minutes, with an average gap length of 7.38 minutes.

The missing rows in the dataset could theoretically be imputed using various techniques, such as filling with the mean or median values, carrying forward the most recent known values, or applying more advanced machine learning methods. However, given that these missing data points represent only about 1.43% of the entire dataset, no imputation or additional action was taken in this case. The small proportion of missing values is unlikely to significantly impact the analysis or modeling, allowing us to proceed with the available data without substantial adjustments.

## 2.3 Outliers

The outliers summary table (Figure 3) provides a detailed view of the distribution of values in each numerical column, highlighting potential outliers based on the interquartile range (IQR) method. Key columns like active\_power, current, reactive\_power, and apparent\_power show a notable percentage of positive outliers, indicating frequent peaks in power usage. These could correspond to real events, such as the use of high-power appliances, rather than data anomalies. Columns like voltage, power\_factor, and humidity have very few outliers, suggesting these measurements are relatively stable and less likely to be influenced by irregular usage patterns. The absence of extreme placeholder values, such as 99999, confirms that the data entries appear to be realistic within the household energy context.

In terms of handling these outliers, typical approaches include removing them, winsorizing (limiting extreme values), or applying transformations. However, without sufficient business or industry knowledge, it is challenging to determine whether these outliers are genuine or anomalous. For instance, temperature outliers might simply reflect natural seasonal variations. Consulting with the data collection team would be beneficial for understanding whether certain spikes in active\_power and current are expected. Given this context, we've decided not to transform or remove the data at this stage, as outliers could hold important insights. Additionally, since we won't rely on models sensitive to outliers, like linear regression, retaining these values should not negatively impact the analysis. This decision allows us to preserve the data's natural variation, which might be relevant for understanding real-world energy consumption patterns.

Column	Median	Mean	Max Value	Min Value	Lower Threshold	Upper Threshold	Negative Outliers	Negative Outliers (%)	Positive Outliers	Positive Outliers (%)	Total Outliers	Total Outliers (%)
active_power	250.3	285.978	2900	24.4	-148.6	673.8	0	0	23400	3.86611	23400	3.86611
current	2.29	2.58691	24.41	0.3	-0.085	5.635	0	0	25563	4.22347	25563	4.22347
voltage	124.4	125.299	135.5	107.6	112.25	139.85	158	0.0261045	0	0	158	0.0261045
reactive_power	128.5	132.546	1293.58	4.73	-81.3037	333.026	0	0	3730	0.616264	3730	0.616264
apparent_power	286.44	321.832	2931.64	37.14	-93.255	698.305	0	0	23813	3.93434	23813	3.93434
power_factor	0.8917	0.854237	1	0.2018	0.4615	1.2431	54	0.00892179	0	0	54	0.00892179
temp	19.49	19.5204	39.37	-5.56	0.650001	38.41	628	0.103757	78	0.012887	786	0.116644
feels_like	18.9	18.71	36.7	-6.13	-0.995001	38.725	1287	0.212636	0	0	1287	0.212636
temp_min	18.06	18.0582	37.59	-5.56	-0.285001	36.315	596	0.0984701	328	0.0528698	916	0.15134
temp_max	20.44	20.7822	39.44	-5.56	2.29	39.33	392	0.0647656	146	0.0241219	538	0.0888874
pressure	1015	1015.22	1035	996	999	1031	98	0.0161914	796	0.131514	894	0.147705
humidity	47	47.9546	100	1	-14.5	189.5	0	0	0	0	0	0
speed	2.24	2.62386	10.29	0	-3.555	8.725	0	0	1776	0.293428	1776	0.293428
deg	144	152.736	360	0	-297	575	0	0	0	0	0	0

Figure 3 Outliers summary table

## 2.4 Feature engineering: Features creation using the datetime

To enhance the dataset with temporal context, several new columns were created. We added year, month, season, and day of the year to capture seasonal and yearly trends, enabling analysis of long-

term patterns. Additionally, we created columns for weekday (1 = Monday to 7 = Sunday) and a weekend indicator (0 = weekday, 1 = weekend) to differentiate between workdays and weekends, as energy usage can vary. Hour of the day and a binary nighttime indicator were added to capture daily cycles, with nighttime defined as 12 PM to 8 AM. These features enrich the dataset by allowing models to capture variations in energy usage across different times, days, and seasons.

## 2.5 Analysis of Average Active Power Consumption

The plots (Figure 4) provide insights into the variations in average active power across different time scales. The yearly plot shows a peak in 2023, suggesting an increase in energy consumption compared to 2022 and 2024. The seasonal plot indicates the highest average active power during the summer (Season 3), with a significant drop in winter (Season 1), suggesting higher energy use in warmer months, likely due to cooling needs.

The monthly plot further refines this trend, showing a clear peak in July, with elevated energy usage in the summer months (June to August). Energy consumption is lowest in winter months, especially around December and January, likely due to reduced demand for cooling. Lastly, the weekday plot reveals that energy consumption is highest on Monday, Tuesday, Wednesday, and Sunday, while Friday and Saturday have the lowest consumption. Considering the amount of variability of energy consumption during weekends we will drop the feature weekend.

Overall, these temporal patterns provide valuable context for understanding energy demand across different times, guiding feature engineering and model building.

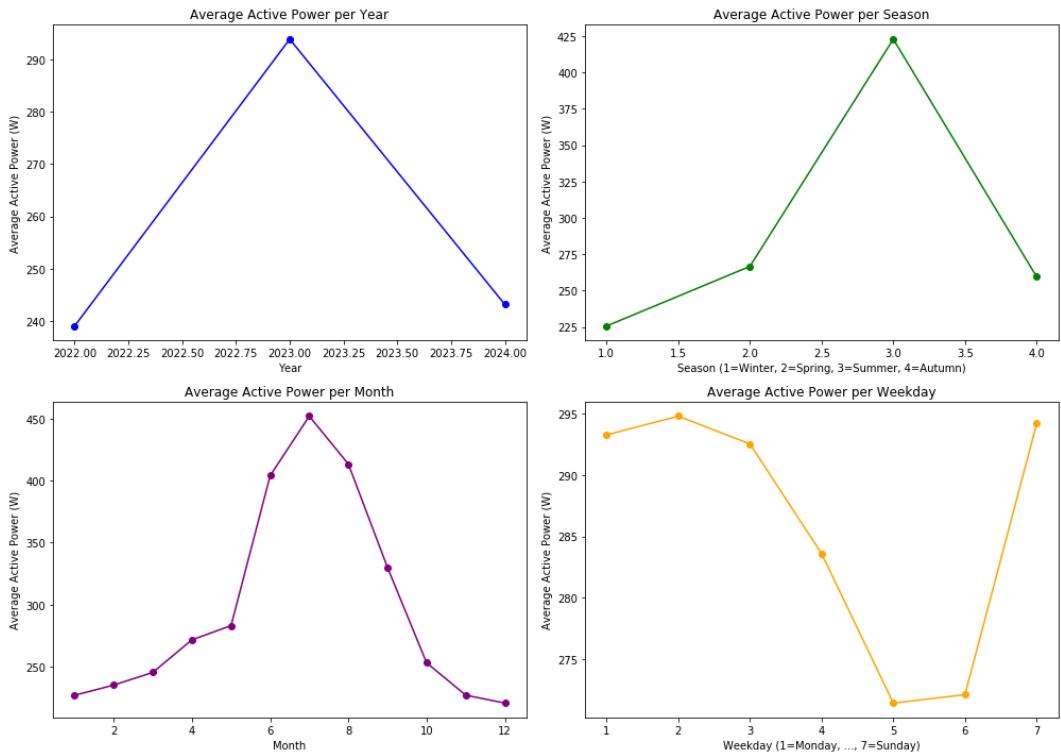


Figure 4 Average active power per Year, Season, Month and Weekday

The plot of average active power per hour (Figure 5) reveals clear daily consumption patterns. Energy usage is lowest during the early morning hours, particularly between 3 AM and 5 AM, which aligns with typical low-activity periods in households. Starting around 6 AM, there is a noticeable increase in power usage, peaking between 8 AM and 10 AM, likely due to morning routines. After a slight dip during midday, energy consumption rises again in the evening, reaching another peak around 8 PM to 9 PM, which could be attributed to evening activities. This pattern highlights two main periods of high energy usage—morning and evening—while nighttime hours consistently show the lowest

consumption.

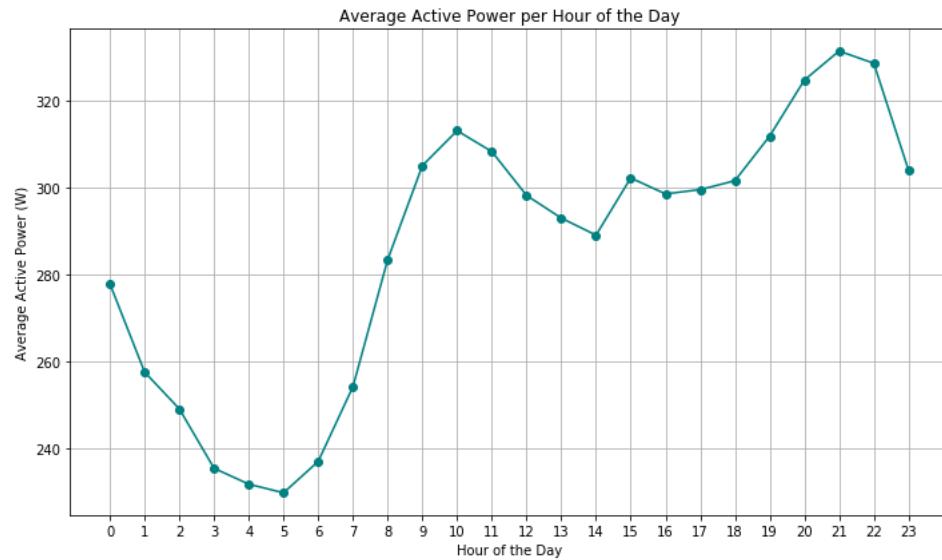


Figure 5 Average Active Power per Hour reveals certain patterns

The bar plot comparing **average active power** between daytime and nighttime (Figure 6) aligns with our earlier observations of lower energy consumption during nighttime hours. Daytime consumption is higher on average, reflecting increased household activity, while nighttime usage drops significantly. This supports the daily cycle pattern observed, with minimal energy usage during nighttime and peaks occurring in the morning and evening. This distinction between daytime and nighttime usage can provide valuable context for understanding daily energy demands.

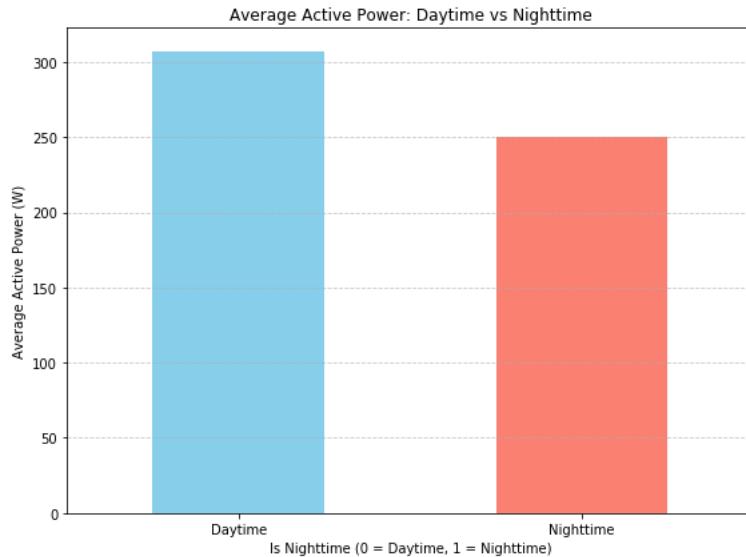


Figure 6 Average active power Daytime or Nighttime. Higher consumption during nighttime

## 2.6 Feature Engineering: Creating Lagged and Rolling Average Features Based on Temporal Dependencies

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots provided insights into the temporal dependencies within the active power data. The PACF plot revealed strong correlations at lags 1 and 2, indicating that the most immediate past values have a significant influence

on the current active power level. The ACF plot, with its gradual decline over multiple lags, suggested that there is also a longer-term dependency, where recent values up to 30 minutes prior contribute meaningful context.

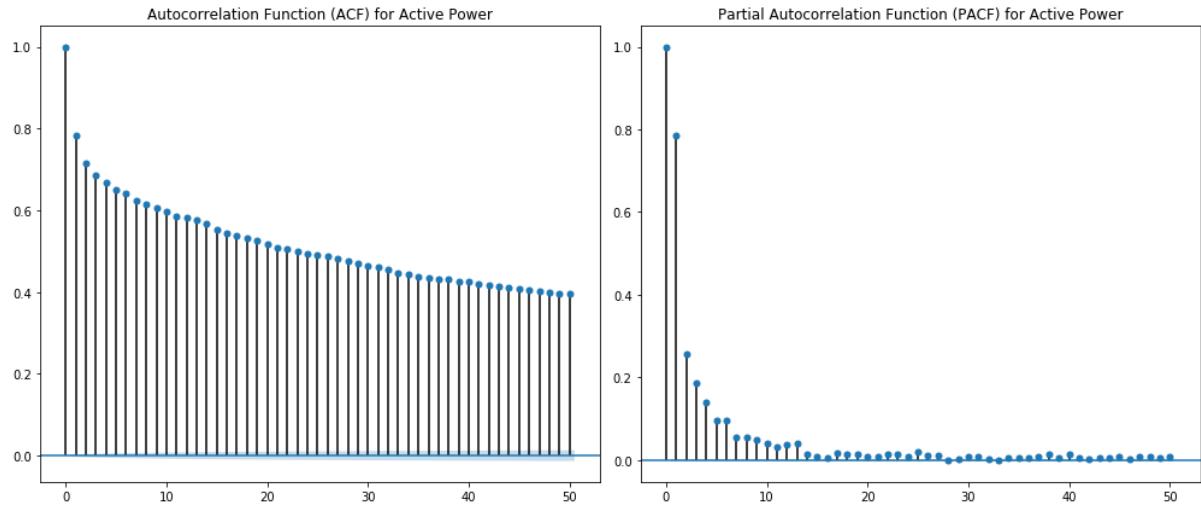


Figure 7 ACF and PACF analysis of Active Power

Based on this analysis, we created five new features to capture both short-term dependencies and broader trends. Lagged values at 1 and 2 minutes provide immediate historical information, helping the model understand how the current active power level relates to the last two minutes. Additionally, we calculated a 20-minute rolling average to smooth out short-term fluctuations and provide a clearer view of recent trends. To capture variability over different time windows, we added a 20-minute rolling standard deviation, which indicates the stability or volatility in energy usage within this period, helping to highlight any unusual activity in the immediate past.

For longer-term trends and variability, we included a 60-minute rolling mean and a 60-minute rolling standard deviation. The 60-minute rolling mean reflects the average active power over the past hour, offering a broader view of the power consumption pattern over a typical daily cycle. Similarly, the 60-minute rolling standard deviation shows how stable or variable the power usage is within this hour, which can be useful in detecting periods of consistently high or low variability.

## 2.7 Target Shifting and Column Selection for Feature Engineering

To prepare the dataset for predicting the next time step, we shifted the target variable **active\_power** by one row to create a new column, **active\_power\_target**, representing the value at **t+1**. This ensures that each row's features correspond to the subsequent target value, aligning with our objective of forecasting the next time step based on current information.

After shifting, rows with **NaN values** were dropped to maintain data consistency, as these NaNs were introduced by the target shift and previously created lagged/rolling features. Additionally, we removed redundant columns, such as **day\_of\_year**, **temp\_min**, **temp\_max**, and **feels\_like** to simplify the feature set. The original **active\_power** column was also dropped to avoid overlap with the shifted target variable. This refined dataset provides a clean foundation for further feature engineering and model training.

## 2.8 Analysis of the Correlation Matrix

The correlation heatmap (Figure 8) reveals that **current**, **apparent\_power**, **power\_factor**, **active\_power\_lag\_1**, **active\_power\_lag\_2**, **active\_power\_rolling\_20**, and **active\_power\_rolling\_mean\_60** exhibit the highest positive correlations with **active\_power\_target**. Notably, **apparent\_power** and **power\_factor** show particularly high correlations with other features, which is expected since they are derived from the multiplication of existing features. Due to their high

multicollinearity with other variables, we will remove apparent\_power and power\_factor to simplify the feature set and avoid redundancy. This decision will help improve model interpretability and reduce the risk of overfitting.

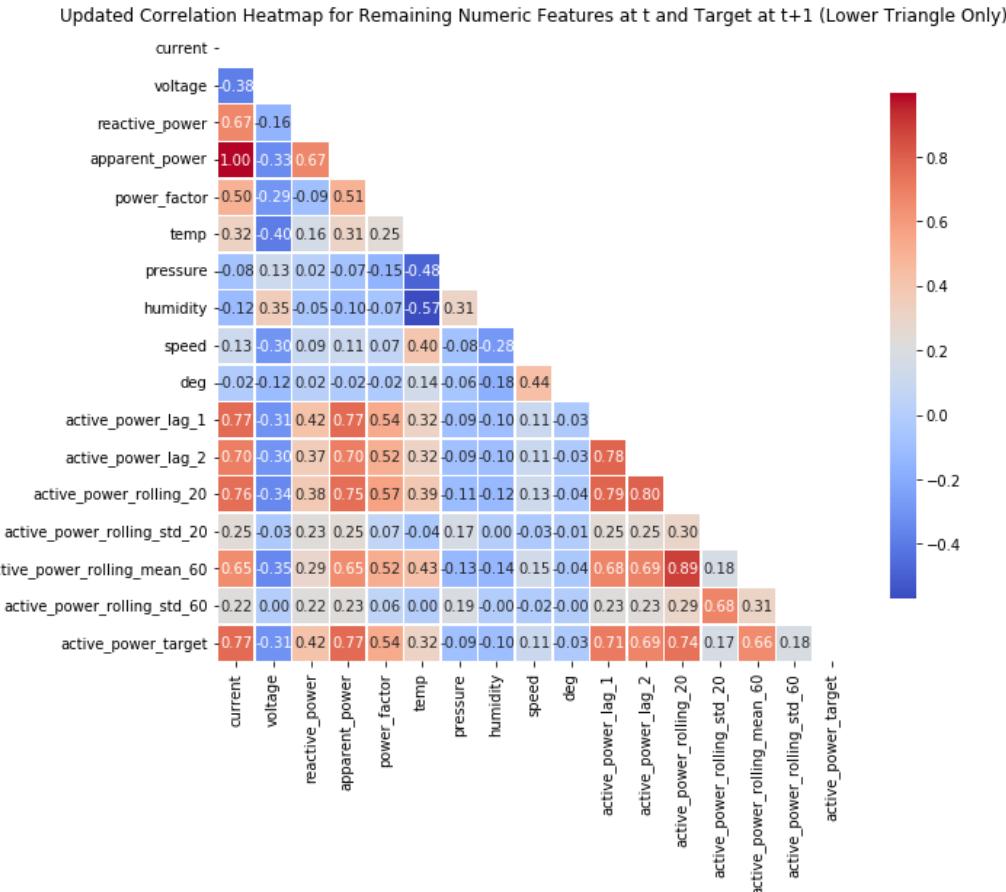


Figure 8 Correlation matrix

## 2.9 Removal of Redundant Description Column

During exploratory data analysis, we identified that the description column contains detailed weather conditions that are largely captured by the main column. Since the description values provide additional detail without significant variance from main, we decided to remove it to avoid redundancy and simplify the dataset. This step helps reduce dimensionality and improve model interpretability without sacrificing relevant information.

## 2.10 Interaction features

Interaction features can capture relationships between variables that individual features might miss, potentially enhancing model performance. However, identifying the most meaningful interactions requires extensive analysis and domain knowledge to avoid adding noise. Given the effort required to rigorously evaluate and select interaction features, examining them in-depth was considered out of scope for this project.

### 3 Regression Model - LightGBM

#### 3.1 Preliminary Steps for Modeling

To prepare the dataset for modeling, several crucial preprocessing steps were implemented to ensure data cleanliness and suitability for machine learning:

- **Scaling and Normalization:** Numerical features were standardized using StandardScaler, normalizing them to a mean of 0 and a standard deviation of 1. This process is vital for algorithms sensitive to input scales, ensuring that each feature contributes equally to the model and preventing biases from features with larger magnitudes.
- **One-Hot Encoding of Categorical Features:** Categorical variables were converted into numerical format using one-hot encoding, which creates binary columns for each category. This approach avoids the pitfalls of ordinal encoding and ensures the model treats all categories as distinct and unrelated, which is essential for non-ordinal categories.
- **Train-Test Split:** The dataset was split into training and testing sets to evaluate model performance and generalizability.

These preprocessing steps are foundational for accurate and efficient model training and evaluation, ensuring the data is well-structured and models can learn effectively.

#### 3.2 Benchmarking Machine Learning Models

**Model Evaluation Process:** To identify the most suitable regression algorithm for predicting active power, we evaluated multiple models that are robust to outliers. We compared LightGBM, Gradient Boosting, XGBoost, Bagging Regressor, ElasticNet, Random Forest, and AdaBoost. Each model was evaluated using R<sup>2</sup>, Adjusted R<sup>2</sup>, RMSE, and MAE on the test set.

**Results and Interpretation:** Among the models, LightGBM achieved the highest performance (Figure 9), with an R<sup>2</sup> of approximately 0.53 and the lowest RMSE and MAE values (120.00 and 40.81, respectively), indicating that it captured the data's patterns effectively. Gradient Boosting closely followed with a comparable R<sup>2</sup> and slightly higher RMSE and MAE.

rows x 68 columns]					
	Model	R2	Adjusted R2	RMSE	M
	LightGBM	0.525752	0.525489	120.000483	40.8107
	GradientBoosting	0.517064	0.516796	121.094684	42.5668
	XGBoost	0.468888	0.468594	126.991074	48.9914
	Bagging	0.461626	0.461328	127.856311	67.3075
	ElasticNet	0.452889	0.452586	128.889671	63.5383
	RandomForest	0.395917	0.395582	135.434320	76.1101
	AdaBoost	-1.298820	-1.300093	264.199929	215.0197

Figure 9 Models evaluation. LightGBM emerged as the most promising model

#### 3.3 LightGBM Model Optimization and Final Evaluation for Active Power Prediction

**Process Overview:** We used a systematic approach to optimize a LightGBM model for predicting the next-minute active power. The initial steps involved defining a parameter grid to explore a range of hyperparameters for the model, including num\_leaves, max\_depth, learning\_rate, n\_estimators, subsample, and colsample\_bytree. We used **TimeSeriesSplit** with 3 splits to maintain the temporal structure in the training data and performed grid search cross-validation with GridSearchCV, optimizing for negative mean squared error. Once the optimal hyperparameters were identified, we retrained the model on the full training set and evaluated it on the test set to obtain final performance metrics.

**Final Model Performance:** The tuned LightGBM model achieved a **Test RMSE of 119.75**, **MAE of 41.62**, and **R<sup>2</sup> of 0.528** on the test set. These metrics indicate a modest improvement over the baseline model, with RMSE slightly reduced, suggesting the model's ability to capture subtle patterns in the data. However, the R<sup>2</sup> score suggests there may still be unexplained variability, which could be due to the inherent complexity of the data or the limits of the current feature set. Overall, the model demonstrates reasonable predictive performance with balanced parameter tuning, though further enhancements might involve additional feature engineering or exploring ensemble methods.

### 3.4 Comparison of Actual vs. Predicted Active Power Over Time

This time-series plot (Figure 10) illustrates the model's predicted active power values against the actual values over the test period. We can observe that the model captures some general patterns but tends to underpredict peak values significantly. The predicted values are more stable and less variable compared to the actual values, suggesting that the model struggles to fully capture the variability and spikes in active power. This may indicate that additional feature engineering or a model that better handles extreme values (such as incorporating peak handling techniques) could further improve prediction accuracy, especially during high-consumption periods. Overall, while the model captures trends, further refinement is needed for better alignment with actual fluctuations.

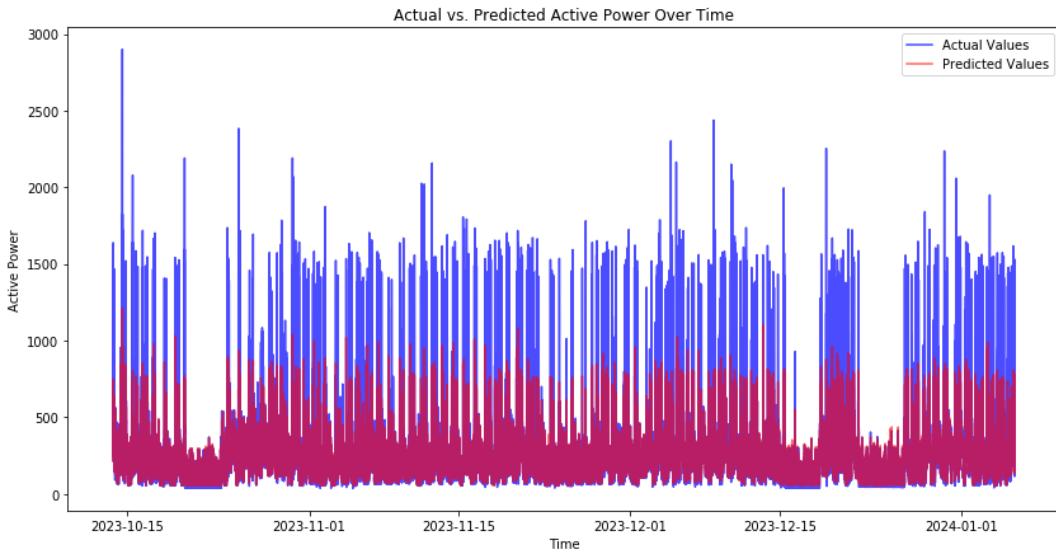


Figure 10 LightGBM Actual vs Predicted values. The model does not capture extreme values.

### 3.5 Feature Importance Analysis for LightGBM Model

The feature importance plot (Figure 11) highlights the relative contribution of each feature in the LightGBM model for predicting active power. The top features include **current**, **active\_power\_lag\_1**, **reactive\_power**, and **active\_power\_rolling\_std\_20**, indicating that recent power and energy-related variables play a crucial role in the model's predictions. Temporal and environmental features such as **temperature**, **humidity**, and **year 2023** also show moderate importance, suggesting that these factors add valuable context for energy consumption patterns.

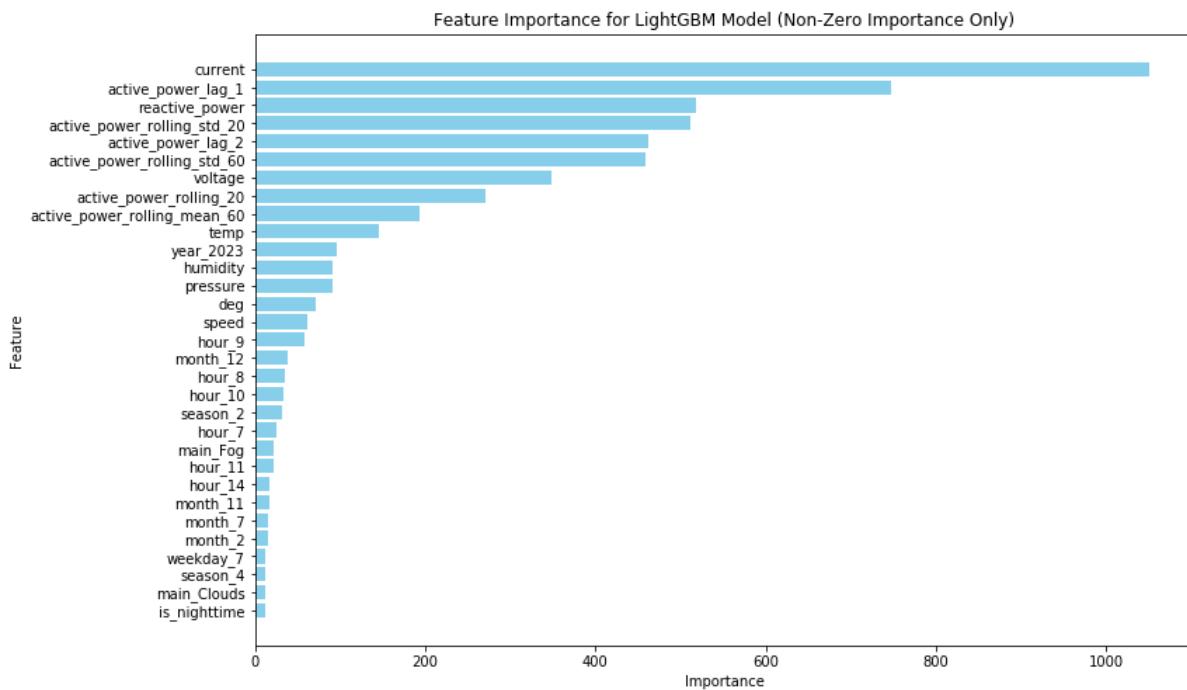


Figure 11 Feature importance using the LightGBM model

### 3.6 Final Model Training and Evaluation on Full Dataset

In this final phase, we prepared our best-performing model for deployment by retraining it on the entire dataset. To ensure consistent scaling, we reused the previously fitted scaler, transforming the complete feature set without re-fitting. This approach avoids any distributional shift that might occur with a new scaler and aligns the deployment model with the transformations applied during initial training. We then initialized the LightGBM model with the optimal hyperparameters found during tuning, allowing it to leverage all available data for maximum predictive power.

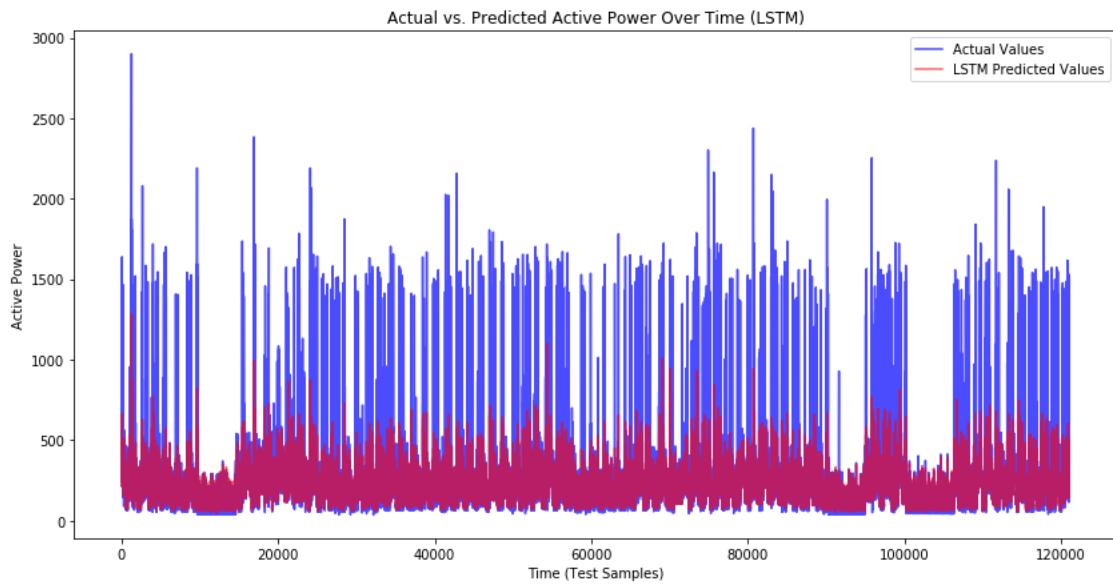
After training, the final model was saved as `final_lightgbm_model_full.joblib`, making it ready for deployment. An optional evaluation on the full dataset revealed an RMSE of 101.01, MAE of 34.37, and an R<sup>2</sup> of 0.716, showing robust performance and confirming that the model effectively captures the target's variance. This final setup ensures that the model is not only optimized but also consistently scaled, providing reliable predictions in production.

## 4 LSTM

For this project, we developed an LSTM model to predict active power consumption, aiming to achieve high accuracy and capture temporal dependencies within the dataset. Initially, we experimented with various look-back periods, testing up to 60 time steps, but found that a shorter look-back of 3 time steps produced the most consistent and stable results. This finding aligns with the nature of the data, where shorter intervals seemed to better capture the power consumption patterns without overfitting.

We also tested different LSTM architectures, including more complex multi-layer setups, but simpler architectures yielded the best performance. Our final model, which consists of a single LSTM layer with 50 units, followed by a dropout layer and a dense output layer, provided the most balanced results in terms of both training stability and predictive power. The model achieved a Root Mean Squared Error (RMSE) of 133.31 and a Mean Absolute Error (MAE) of 53.93 on the test set, with an  $R^2$  score of 0.41, indicating moderate predictive accuracy.

The model manages to capture the general trend but often fails to predict extreme fluctuations accurately. This discrepancy is evident in the peaks, where the actual power values spike much higher than the model's predictions. The underestimation of high peaks and the slight overestimation of lower values indicate that the model's current architecture may not fully capture the dynamics of high-variance regions.



*Figure 12 LSTM Actual vs Predicted values. The model does not capture extreme values and is less efficient than the LightGBM.*

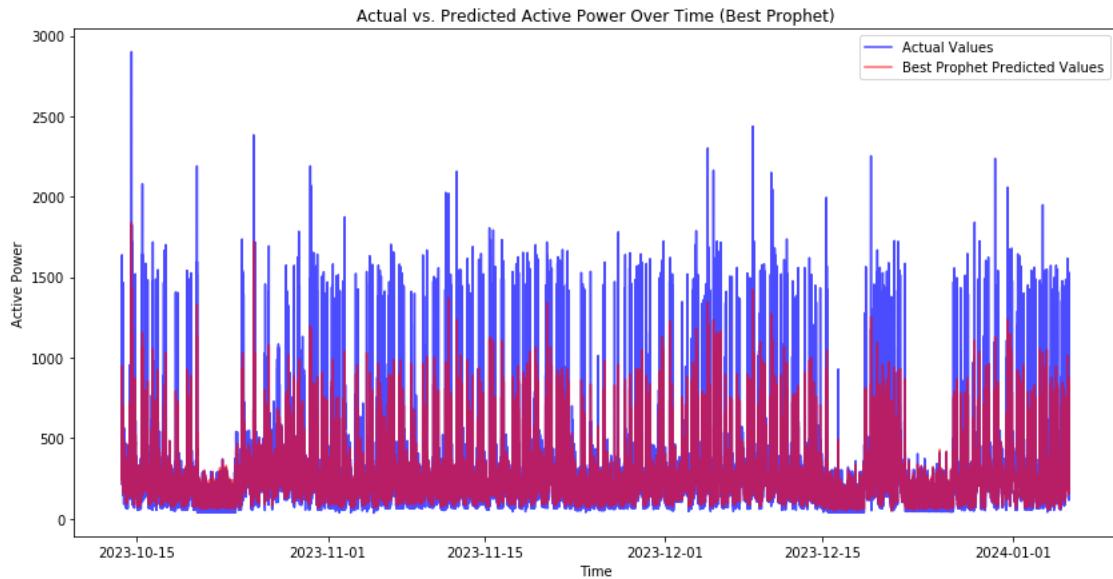
LightGBM achieved lower error metrics compared to the LSTM model, showcasing its strength in capturing the underlying patterns without needing sequential learning. Although LightGBM lacks the inherent time-dependency capabilities of LSTM, it provided competitive and, in some cases, superior performance, making it a strong candidate for this type of prediction.

To further enhance our model, future steps could include more advanced feature engineering techniques. Additionally, experimenting with hybrid models that combine LSTM's temporal learning capabilities with LightGBM's efficiency could create a more robust prediction framework. Further exploration of advanced LSTM architectures, such as stacked or bidirectional LSTMs, could also provide additional gains in accuracy, helping refine the model for even better predictive performance.

## 5 Prophet Model

In this section, we applied **Prophet**, a time-series forecasting tool developed by Facebook, designed for data with strong seasonal patterns. Prophet decomposes the time series data into trend, seasonality, and holidays components, making it effective for business and energy datasets with clear periodic cycles. We experimented with hyperparameter tuning for Prophet, focusing on parameters like `changepoint_prior_scale`, `seasonality_mode`, `seasonality_prior_scale`, and `weekly_seasonality`. The best model was configured with an additive seasonality mode, a `changepoint_prior_scale` of 0.05, daily seasonality enabled, and weekly seasonality enabled.

The tuned Prophet model achieved a **Test RMSE of 125.64**, a **MAE of 54.15**, and an **R<sup>2</sup> score of 0.48** on the test set. These results indicate that Prophet was able to capture part of the seasonal structure in the data. However, it performed slightly worse than the best-performing model, **LightGBM**, which had an RMSE of around 119.75 and a higher R<sup>2</sup> score. The LightGBM model excelled likely because it could capture non-linear dependencies in the data, particularly leveraging the lagged and rolling features created earlier. But the we notice (Figure 13) that Prophet is able to capture higher fluctuations and peaks.



*Figure 13 Prophet Model Actual vs Predicted values. The model does not capture extreme values and is less efficient than the LightLGBM.*

In conclusion, while Prophet effectively models seasonality and trends, it struggled to capture the short-term fluctuations present in minute-level data as well as LightGBM. This limitation suggests that models like LightGBM, which excel with exogenous and lagged features, might be better suited for granular and complex time-series data where seasonality alone is insufficient to describe the patterns.

## 6 Other Model Alternatives for Time Series Forecasting

In addition to Regression Models (LightGBM), LSTM and Prophet, there are several other models commonly used for time-series forecasting that could be considered in future analyses. Here's a brief overview of these methods:

- **ARIMA (AutoRegressive Integrated Moving Average):** ARIMA models are popular for univariate time series data without strong seasonal components. They are based on the principles of autoregression (AR) and moving averages (MA) and require the data to be stationary..
- **SARIMA (Seasonal ARIMA):** SARIMA extends ARIMA by adding support for seasonality. It introduces additional terms to capture seasonal patterns and is suitable for time series data with periodic patterns.
- **SARIMAX (Seasonal ARIMA with eXogenous variables):** SARIMAX further extends SARIMA by allowing the inclusion of exogenous variables, making it more versatile for multivariate time series forecasting. This model can incorporate external factors or lagged values as inputs, which could help capture complex dependencies. However, SARIMAX models can be computationally expensive, especially with multiple exogenous features and seasonal components.
- **Generalized Additive Models (GAMs):** GAMs offer a flexible framework to model non-linear relationships in time series data. By using smooth splines to capture trends, seasonality, and even non-linear relationships between features, GAMs allow for a more interpretable approach.
- **Ensemble Models (Stacking Models):** For complex time-series data, an ensemble approach that combines the predictions of multiple models (e.g., LightGBM, SARIMA, and Prophet) could improve performance by leveraging the strengths of each model.

## 7 Deployment Solution

To ensure a scalable and portable deployment of our active\_power prediction model, we chose to containerize the application using Docker and serve it via the FastAPI web framework. Containerization with Docker provides an isolated environment that includes all dependencies, making it easier to deploy across different platforms and environments without compatibility issues.

### 7.1 Containerization and Docker Setup

We created a Dockerfile to define the environment and configurations necessary for the application. The Dockerfile begins by using an official Python image, ensuring a consistent runtime environment. It then installs all required dependencies specified in requirements.txt, which includes FastAPI, LightGBM, and other necessary libraries. The Dockerfile also copies relevant files (e.g., predict.py, models, and data) into the container, setting up everything needed to run the model and API seamlessly.

By containerizing our application, we can build and deploy it on any system that supports Docker. This process improves scalability and makes it easier to manage dependencies, making our solution highly portable and efficient for production deployment.

### 7.2 Serving the Model with FastAPI

To serve the model, we selected FastAPI as the web framework due to its asynchronous capabilities, fast performance, and intuitive interface for building APIs. FastAPI provides automatic OpenAPI documentation, which allows for easy interaction with the API through a web interface. In predict.py, we defined an endpoint /predict that accepts POST requests containing input data in JSON format. The API processes the data, applies the necessary transformations (using a pre-trained pipeline), and feeds it into the LightGBM model for prediction.

The /predict endpoint is structured to handle input validation, ensuring that the data has all required fields before proceeding to prediction. We also added an endpoint /example\_data to return sample input data in JSON format, making it easy for users to understand the data structure expected by the API.

### 7.3 Endpoint Functionality

The API is designed to take input data (a JSON structure containing relevant features), preprocess it, and generate a prediction for the active\_power variable. Each request goes through the following steps:

- Data Validation: Ensures all required columns are present in the input.
- Data Transformation: Applies the feature engineering pipeline and scaling.
- Prediction: Passes the processed data to the trained LightGBM model and returns the result in a structured JSON format.

### 7.4 Summary

This containerized solution, with Docker and FastAPI, makes the model accessible as a RESTful API, allowing clients to request predictions remotely. It provides a reliable, consistent way to deploy and scale the model in production, enhancing accessibility and robustness. With FastAPI's documentation and the isolated Docker environment, this setup is ideal for production-grade deployment, ensuring easy integration with other services and scalability for handling larger request volumes.

## 8 Cloud Infrastructure Schema

To deploy and serve the predictive model for "active\_power," we recommend a cloud infrastructure solution that uses containerized deployment to ensure scalability, reliability, and ease of maintenance. The setup includes the following components:

1. **Containerized Model Deployment:** The Dockerized application, including the predict.py API, is deployed to a cloud container service such as AWS Fargate, Google Cloud Run, or Azure Container Instances. This containerization simplifies scaling and ensures a consistent environment, reducing dependency issues.
2. **API Gateway and Load Balancer:** A cloud-based API Gateway routes incoming requests to the containerized API, handling traffic routing and load balancing. This ensures efficient distribution of requests, especially during high-demand periods, and provides features like authentication, logging, and monitoring.
3. **Data Storage and Model Management:** Model files, such as final\_lightgbm\_model\_full.joblib and scaler.joblib, can be stored in a cloud storage solution (e.g., AWS S3, Google Cloud Storage, or Azure Blob Storage). This enables easy access for updates and allows the model to be versioned and reloaded if necessary.
4. **Monitoring and Logging:** Integrate with a monitoring service (e.g., AWS CloudWatch, Google Cloud Monitoring) to track API performance, monitor model predictions, and capture logs. This helps in identifying and troubleshooting issues in real-time.
5. **Continuous Integration and Continuous Deployment (CI/CD):** Implement a CI/CD pipeline (e.g., GitHub Actions, GitLab CI/CD, AWS CodePipeline) to automate testing and deployment of new model versions or updates to the API.
6. **Scalability:** The containerized approach supports horizontal scaling, where additional containers can be added as demand increases. Auto-scaling features provided by cloud services ensure that resources are adjusted dynamically based on traffic, optimizing both performance and cost.
7. **Orchestration:** Using a container orchestration platform, such as Kubernetes (or managed services like AWS EKS, Google Kubernetes Engine), allows better management of containerized workloads. This can handle container scheduling, scaling, and resilience, ensuring that the model API remains available even during high traffic.
8. **Maintenance and Monitoring:** Monitoring systems continuously track API performance, usage, and error rates. Maintenance involves regular updates of model files and dependencies, which can be managed through the CI/CD pipeline to minimize downtime. Alerts can be configured to notify the team of any anomalies, ensuring timely responses to issues.
9. **Model Drift:** To address data and model drift, set up regular evaluations of the model's performance on new data. Retraining the model on fresh data, followed by deployment via the CI/CD pipeline, keeps predictions accurate over time. Automated drift detection mechanisms, where model accuracy is evaluated on a fixed schedule, can trigger alerts or redeployment when performance drops.
10. **Additional considerations:**
  - **Authentication and Authorization:** Integrate an authentication layer (e.g., API keys, OAuth) through the API Gateway to secure access to the model API. This ensures only authorized users or services can make requests.
  - **Scheduled Scaling:** If the model is used during predictable periods (e.g., peak business hours), implement scheduled scaling rules to add or remove resources at specific times, reducing unnecessary costs during off-hours.
  - **Automated Retraining:** In cases where the input data changes frequently, we can consider setting up an automated retraining pipeline. This could be done on a schedule (e.g., weekly) or triggered by drift detection alerts.

## 9 Project Deliverables

- **Active\_Power\_Prediction.py:** Script for initial data analysis, feature engineering, model training, and evaluation.
- **data/ :** Contains input data files:
  - example\_data.json:** Sample data in JSON format for testing the API.
  - recent\_data.csv:** Recent data used for testing predictions through the API.
- **Dockerfile:** Defines the environment for deploying the application in a Docker container, ensuring consistency across different setups.
- **models/:** Model-related files:
  - expected\_columns.txt:** List of required features for the model input.
  - final\_lightgbm\_model\_full.joblib:** Trained LightGBM model.
  - scaler.joblib:** Scaler for preprocessing features before prediction.
- **predict.py:** Main FastAPI script, loading the model and pipeline, and providing /predict for predictions and /example\_data for sample input.
- **README.txt:** Documentation with an overview of the project, setup, and usage instructions.
- **requirements.txt:** Lists dependencies required for the project.
- **send\_data\_to\_predict\_API.py:** Script to test the /predict API endpoint by sending JSON data and retrieving predictions.
- **src/:** Contains the data\_pipeline.joblib file with preprocessing steps applied to data before prediction.