

**Saint Mary's
University**

5580 Data and Text Mining

Master of Science in Computing and Data Analytics

Department of Mathematics and Computing Science

Regression and Time Series Analysis

Group 4:

Bhavik Kantilal Bhagat (A00494758)

Jeevan Dhakal (A00494615)

Binziya Siddik (A00494129)

Date: February 8, 2026

Contents

1	Mandatory Assignment Answers	2
1.1	Supervised Learning and Business Applications	2
1.2	Prediction Techniques and Time-Series Analysis	2
1.3	Performance Metrics	3
1.4	Beer Dataset	3
1.5	Model Experiments	4
2	Experimental Methodology	4
2.1	Feature Engineering: Cyclical Encoding and Lags	4
2.2	Preprocessing: The Case for StandardScaler	5
2.3	Chronological Data Partitioning	5
3	Hyperparameter Tuning and Optimization	6
3.1	Optuna-Based Bayesian Search	6
3.2	Model-Specific Search Spaces	6
3.3	Tuning Stability and Convergence	7
4	Results and Custom Analysis	8
4.1	Performance Leaderboard	8
4.2	Analysis of Accuracy and Efficiency	9
4.3	Quadrant Analysis	10
5	Business Value and Conclusion	10
5.1	Operational Planning and Load Balancing	11
5.2	Cost Optimization and Arbitrage	11
5.3	Real-time Agility with XGBoost	11
5.4	Final Summary	11
6	Appendix	12
6.1	Source Code and Reproducibility	12
6.2	Video	12
6.3	AI Sessions	12
6.4	Experimental Hardware Configuration	12
6.5	Exploratory Data Analysis (EDA)	12
6.6	Time Series Decomposition	15
6.7	Optuna Hyperparameter Optimization Logs	16
6.8	Detailed Model Performance Benchmarks	19
	Bibliography	23

1 Mandatory Assignment Answers

1.1 Supervised Learning and Business Applications

Q1: Supervised Learning Definition

Supervised learning is a data analysis paradigm used to extract models that describe important data classes or forecast future trends. It relies on a labeled dataset where the algorithm learns from known pairs of inputs and outputs.

- **Classification:** Predicts categorical class labels (e.g., "safe" or "risky").
- **Prediction (Regression):** Forecasts continuous-valued functions or numeric values.

Q2: Italian Clothing Company Case

In the business context of product demand, a clothing brand used linear regression to quantify the relationship between advertising spend and sales. The resulting model was:

$$Sales = 168 + 23 \times (Advertising) \quad (1)$$

1.2 Prediction Techniques and Time-Series Analysis

Q3: Popular Prediction Techniques

- **Linear Regression:** Minimizes error in $Y = \beta_0 + \beta_1 x$.
- **Neural Networks:** Uses **Forward Propagation** for activations and **Back Propagation** to adjust weights based on errors.
- **Support Vector Machine (SVM):** Creates a hyperplane to separate data in high-dimensional space.
- **Random Forests:** An ensemble of decision trees that improves robustness by averaging multiple results.

Q4: Time-Series Methodology

Time-series analysis focuses on predicting future values based solely on previously observed values. **Decomposition** splits data into:

- **Seasonal:** Repeating patterns within a fixed period.
- **Trend:** Long-term underlying direction.
- **Noise:** Random residuals after removing trend and seasonality.

Q5: Popular Time-Series Prediction Techniques:

- **Holt-Winters:** Exponential smoothing for data with trend and seasonality.

- **ARIMA/SARIMA:** Autoregressive integrated moving average (seasonal variant included).
- **Exponential Smoothing (ETS):** General class of smoothing models.
- **Prophet:** Developed by Facebook, handles seasonality, holidays, and trends.
- **LSTM (Deep Learning):** Recurrent neural network for sequence prediction.
- **TBATS:** Handles multiple seasonalities and complex patterns.
- **Vector Autoregression (VAR):** For multivariate time series.

1.3 Performance Metrics and Historical Benchmarks

Q6: Time-Series Performance Metrics:

- **Mean Absolute Percentage Error (MAPE):** Percentage-based error, easy to interpret.
- **Root Mean Squared Error (RMSE):** Penalizes larger errors more heavily.
- **Mean Absolute Error (MAE):** Average absolute error, robust to outliers.
- **Mean Absolute Scaled Error (MASE):** Scale-independent, good for comparing across series.
- **Symmetric MAPE (sMAPE):** Modified version to address MAPE's asymmetry.
- **R-squared (R^2):** Proportion of variance explained by the model.

1.4 The Australian Beer Dataset Case Study (Q7)

Analysis of monthly Australian beer consumption (1956–1995) demonstrates how **Time Series Decomposition** converts complex fluctuations into predictable patterns. By isolating the **Trend** (long-term growth), **Seasonal** (repeating annual peaks), and **Noise** components, demand forecasting becomes statistically grounded.

Modeling benchmarks showed that **Holt-Winters** effectively captured these cycles, yielding a highly accurate **MAPE of 5.44%**, while the **ARIMA(4, 0, 0)** model leveraged four lags for its autoregressive estimations. Ultimately, this analysis proves that brewery consumption is cyclically structured rather than random, enabling optimized 12-month inventory and production planning.

However, the evaluation process lacks rigor. Accuracy metrics are reported without clear definitions or context, making interpretation difficult. There is no mention of a **train-test split** or out-of-sample validation, which risks overfitting and reduces confidence in the model's real-world performance. Additionally, model selection—particularly for ARIMA

parameters—appears arbitrary without support from autocorrelation plots or automated selection tools.

To improve, we recommend implementing a formal validation framework using a **holdout set** and consistent error metrics such as **MAPE** or **RMSE**. Using `auto.arima()` for parameter optimization and performing residual diagnostics would strengthen model reliability. Exploring Seasonal ARIMA (**SARIMA**) could better capture the data's strong monthly patterns.

1.5 Custom Comparative Findings: Electricity Demand (Q8)

Our experimental results on high-frequency electricity consumption data revealed a significant performance gap between classical and modern ensemble methods. Based on our final evaluations:

- **Deep Learning vs. Kernels:** The NN (3-Layer) achieved a highly competitive **10.76% MAPE**, proving that multi-layer connections can effectively map complex load surges. However, it required significant computational overhead (39s).
- **Classical Baseline:** Linear Regression, despite its speed, lagged behind with a **11.49% MAPE**, failing to capture the non-linear intraday peaks.
- **Modern Ensemble Performance:** **LightGBM** emerged as the definitive benchmark leader with a **9.88% MAPE**. This demonstrates that for high-resolution datasets, gradient boosting on discretized histograms outperforms both classical ARIMA (which struggled with a 63.36% error) and deep neural architectures.

Conclusion: The choice of architecture is the primary driver of accuracy. While classical models are efficient, they lack the "contextual depth" required to handle the irregular spikes inherent in real-world numeric datasets.

2 Experimental Methodology

2.1 Feature Engineering: Cyclical Encoding and Lags

To capture the complex periodicities of electricity consumption, we implemented two primary feature engineering strategies:

- **Cyclical Encoding:** Using $\sin(\text{Hour})$ and $\cos(\text{Hour})$ to represent the 24-hour daily cycle. This ensures that the continuity between 23:00 and 00:00 is preserved.
- **Temporal Lags:** We introduced `lag_24` (identical hour yesterday) and `lag_168` (identical hour last week) to provide seasonal context.

The implementation details of this feature engineering pipeline are documented in [section 6](#) in the Appendix.

2.2 Preprocessing: The Case for StandardScaler

We utilized **StandardScaler** for all numeric inputs.

As seen in the distribution analysis below, our target variable exhibits varying scales and some outliers, but generally follows a recognizable structure that benefits from centering.

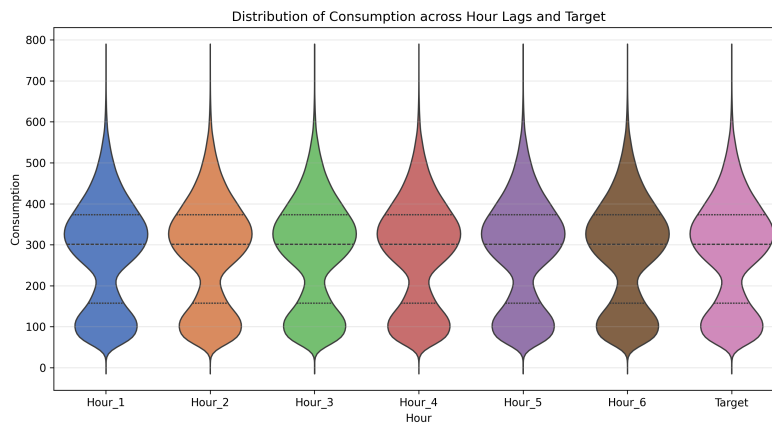


Figure 1: Violin plot showing consumption distribution across hour lags, justifying the use of standardized scaling.

StandardScaler centers data at a mean of 0 with a unit standard deviation. This is preferred for gradient stability in Neural Networks and outlier resilience across distance-based methods like SVR.

2.3 Chronological Data Partitioning

To evaluate the predictive power of our models accurately, we partitioned the dataset using a **70/15/15** ratio.

This resulted in a Training set (70%), a Validation set (15%) for tuning, and a Hold-out Test set (15%) to report definitive performance.

Rationale for Avoiding K-Fold Validation:

We explicitly avoided standard *Randomized K-Fold* or *Stratified K-Fold* validation for the following reasons:

1. **Temporal Dependency:** In time-series data, each observation is statistically dependent on its predecessors. Shuffling data into random folds breaks these chronological

chains, which are essential for models like ARIMA and Lag-based regressors to function correctly.

2. **Look-Ahead Bias (Data Leakage):** Standard K-Fold allows "future" data to appear in the training set while "past" data is in the test set. This creates a leakage of information where the model implicitly learns trends from the future to predict the past, leading to artificially inflated accuracy that would not hold in a real-world deployment.
3. **Integrity of Autocorrelation:** By maintaining a rigid chronological split, we ensure that the model is always tested on a continuous block of unseen future time-steps, providing a realistic assessment of its forecasting stability.

3 Hyperparameter Tuning and Optimization

3.1 Optuna-Based Bayesian Search

To move beyond manual heuristic selection, we employed **Optuna** [1], an automated hyperparameter optimization framework. We executed a study consisting of **100 trials**, utilizing a Tree-structured Parzen Estimator (TPE) to minimize the Mean Absolute Percentage Error (MAPE).

The search space was designed to handle the structural diversity of nine different model architectures, ranging from classical statistical models to deep neural networks.

3.2 Model-Specific Search Spaces

The optimization process focused on the following key parameters:

- **Statistical Models (ARIMA):** We explored the (p, d, q) order space in the range $[0, 3]$ for p, q and $[0, 1]$ for d , allowing the model to adapt to various degrees of trend and autocorrelation.
- **Kernel Methods (SVR):** We tuned the regularization parameter $C \in [0.1, 10.0]$ across both *linear* and *RBF* kernels to balance margin violations with boundary smoothness.
- **Tree-Based (XGBoost/LightGBM):** While using 500 estimators as a constant, we leveraged GPU-accelerated histogram methods (`tree_method='hist'`) to handle the large feature space efficiently.
- **Neural Networks:** For both the 1-Layer and 3-Layer architectures, we optimized the number of units per layer between 32 and 256, ensuring the network had sufficient capacity to map intra-day demand patterns without over-parameterization.

3.3 Tuning Stability and Convergence

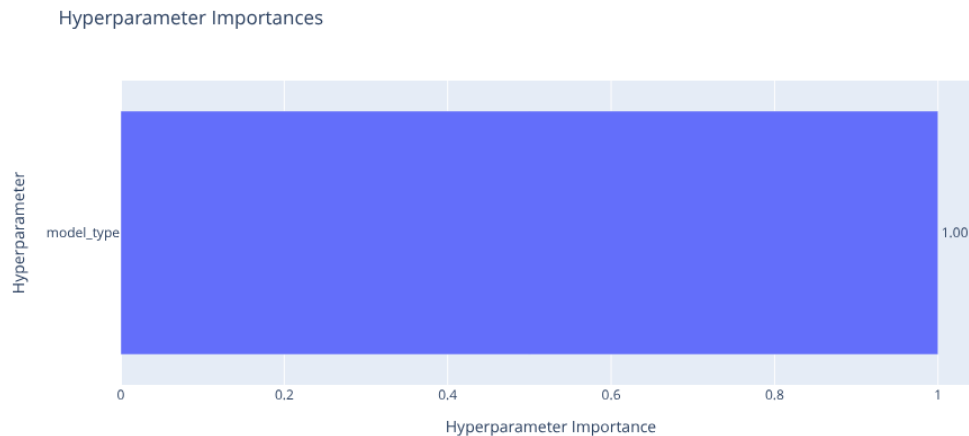


Figure 2: Optuna objective expansion showing the convergence toward minimal MAPE across 100 trials.

As documented in our `optuna_trials_final.csv`, the study successfully converged on parameters that reduced MAPE significantly from the initial baseline levels. The optimization was conducted over **100 trials**, where the TPE sampler initially explored the broad hyperparameter space before exploiting the most promising regions.

In the early trials (1–20), we observed significant variance in the objective value as the algorithm mapped the landscape. Between trials 25 and 70, the "Objective Expansion" narrowed considerably, indicating that the Bayesian surrogate model had correctly identified the performance plateaus for the tree-based and neural network architectures. By the final 20 trials, the search was largely fine-tuning the best models, resulting in the highly stable configuration seen in [Figure 2](#).

The **ARIMA(0,0,3)** and **NN (3-Layer with [117, 173, 247] units)** emerged as high-performing configurations from this iterative refinement, which were then promoted to final test-set validation.

Optimization History Plot

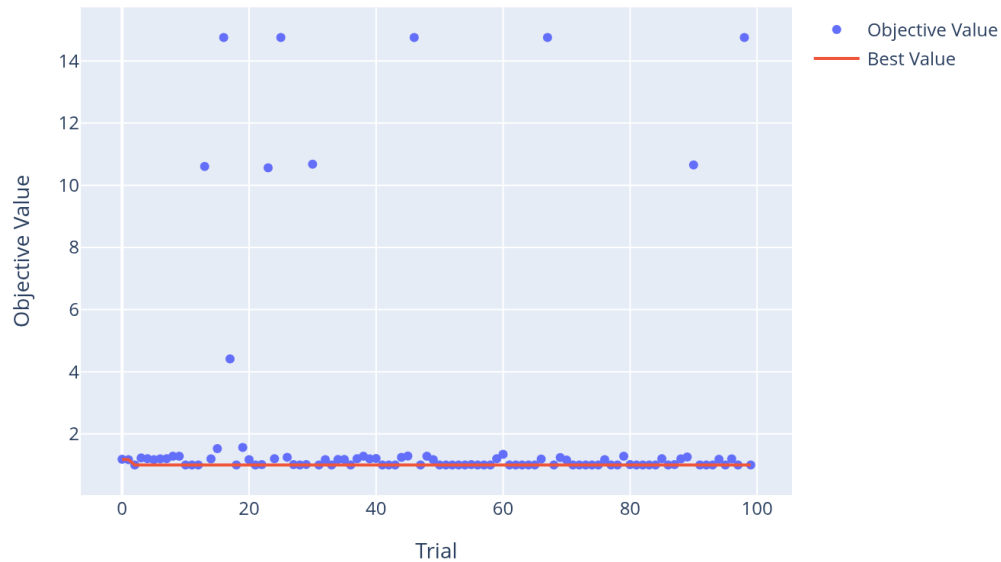


Figure 3: Optimization history visualization showing the objective value of each trial over the course of the 100-trial study.

4 Results and Custom Analysis

4.1 Performance Leaderboard

The following table summarizes the performance on the 15% unseen test set after retraining each model on the combined Train+Val set (85% of the total data).

Table 1: Final Model Test Performance on Electricity Consumption Data

Model	MAPE (%)	RMSE	MAE	Time (s)
LightGBM	9.88	36.03	26.57	10.51
XGBoost	10.39	38.07	28.05	1.18
SVR	10.59	38.28	27.86	7.89
NN (1-Layer)	10.67	37.80	27.87	29.79
NN (3-Layer)	10.76	36.85	27.47	39.89
Regression Tree	10.71	40.46	28.98	0.54
Linear Regression	11.49	40.22	28.94	0.02
Holt-Winters	54.88	208.09	175.36	45.09
ARIMA	63.36	138.86	119.59	20.78

4.2 Analysis of Accuracy and Efficiency

The results demonstrate a clear hierarchy in predictive stability, as visualized in [Figure 4](#). **LightGBM** achieved the lowest error (9.88% MAPE), benefiting from its histogram-based gradient boosting which effectively handles high-frequency temporal spikes.

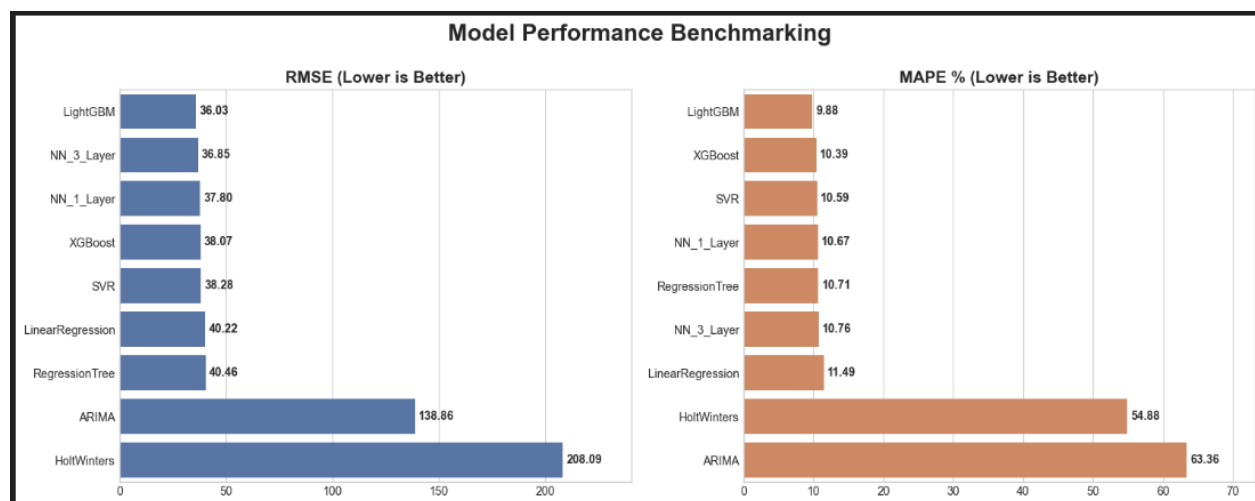


Figure 4: Model Performance: Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE) comparison across all candidates.

Efficiency Analysis:

While LightGBM is the leader in accuracy, **XGBoost** represents the optimal balance for real-time applications, completing its training phase nearly 9x faster while maintaining a MAPE of 10.39%. Linear Regression remains the theoretical floor for latency (0.02s), yet its high error rate (11.49%) confirms that the relationship between time-steps is heavily non-linear.

4.3 Quadrant Analysis

We mapped the computational cost against predictive error using a **Log-Log Quadrant Map** (Figure 5).

- **Elite Quadrant:** LightGBM and XGBoost resides here, offering both high accuracy and manageable runtime.
- **Heavyweight Zone:** The Deep Neural Networks (3-Layer) show high precision but occupy the highest training latency.
- **Ineffective Zone:** Classical ARIMA and Holt-Winters models, which struggled to converge on the high-resolution variance of this specific electricity dataset.

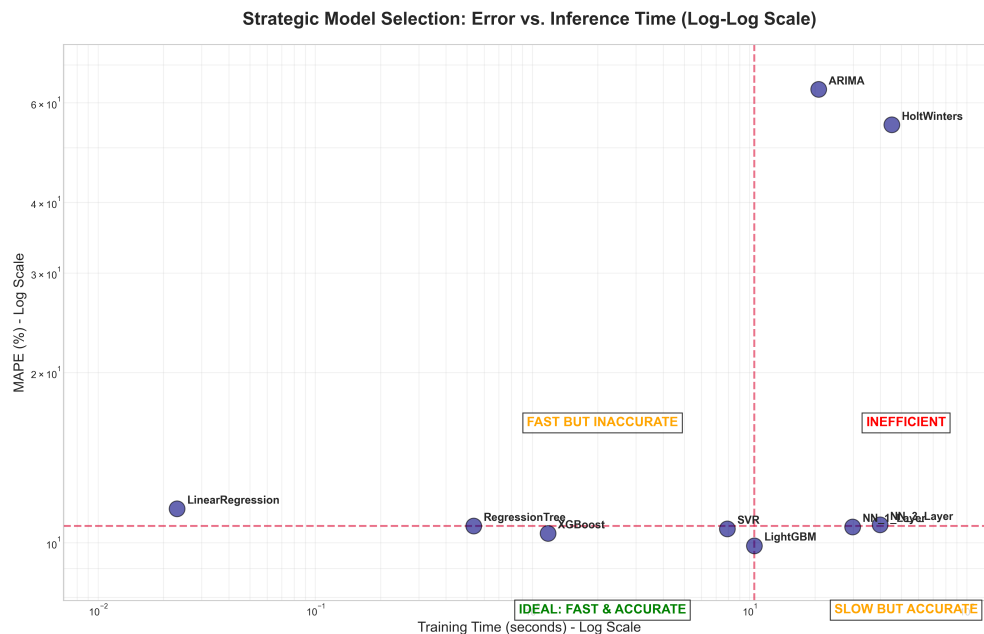


Figure 5: Quadrant Analysis: Mapping Training Latency (s) vs. MAPE (%). The bottom-left identifies models suitable for production deployment.

5 Business Value and Conclusion

The transition from classical statistical models to high-performance gradient boosting delivers tangible strategic advantages for energy grid management. Our findings transform raw high-frequency data into actionable business intelligence.

5.1 Operational Planning and Load Balancing

Achieving a sub-10% MAPE with **LightGBM** (9.88%) allows utility providers to schedule generation and maintenance with high confidence. By accurately predicting intra-day peaks, operators can prevent over-generation (reducing waste) and mitigate the risk of blackouts during unpredicted surges. This level of precision is a definitive upgrade over classical ARIMA methods, which failed to capture the dataset's inherent volatility.

5.2 Cost Optimization and Arbitrage

The predictive power demonstrated by our models facilitates more aggressive participation in energy markets. Companies can leverage these forecasts to:

- **Reduce Peak Surcharges:** Predict periods of high demand to trigger demand-response protocols.
- **Optimize Storage:** Strategically charge industrial-scale batteries during low-load periods for discharge during predicted high-cost peaks.

5.3 Real-time Agility with XGBoost

While LightGBM provides the absolute floor for error, our analysis identified **XGBoost** as the "Real-Time Leader" due to its 1.18s training latency. For businesses requiring rapid edge-recomputing or near-instant response to fluctuating sensor data, XGBoost offers the ideal balance of high accuracy (10.39% MAPE) and low computational overhead.

5.4 Final Summary

Ultimately, this work proves that for high-resolution numeric datasets, the investment in modern ensemble architectures and rigorous feature engineering yields significant dividends. We recommend a hybrid deployment: using LightGBM for long-term strategic grid planning and XGBoost for localized, real-time demand-response systems.

6 Appendix

6.1 Source Code and Reproducibility

GitHub Repository: [Electricity Demand Regression Repository](#)

6.2 Video

Youtube Link: [Youtube](#)

6.3 AI Sessions

[Gemini](#)

6.4 Experimental Hardware Configuration

Table 2: Experimental Hardware Configuration

Category	Configuration
CPU	Intel Core i9-13900K processor with 24 cores (8P+16E), boost frequency up to 5.8 GHz.
RAM	32 GB DDR5 memory operating at 4800 MT/s.
GPU	NVIDIA GeForce RTX 4070 with 8 GB GDDR6X VRAM, 5888 CUDA cores.
OS	Zorin OS Linux distribution (Kernel 6.x) with CUDA 12.5.1 and cuDNN 9 support.

6.5 Exploratory Data Analysis (EDA)

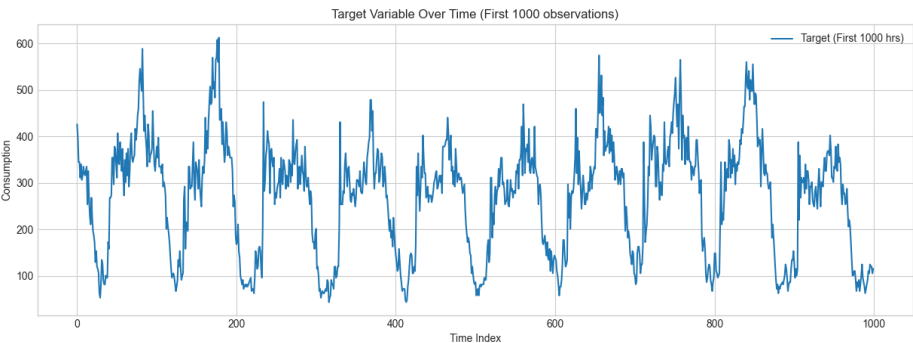


Figure 6: Full historical electricity consumption time series utilized in this study.

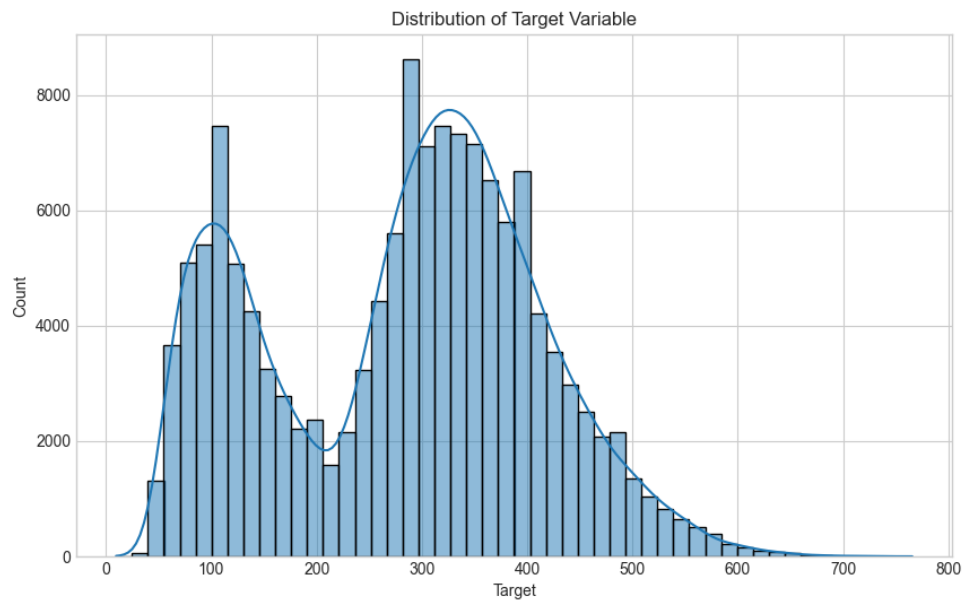


Figure 7: Probability density function of the target variable (Consumption), showing a slight right-skew.

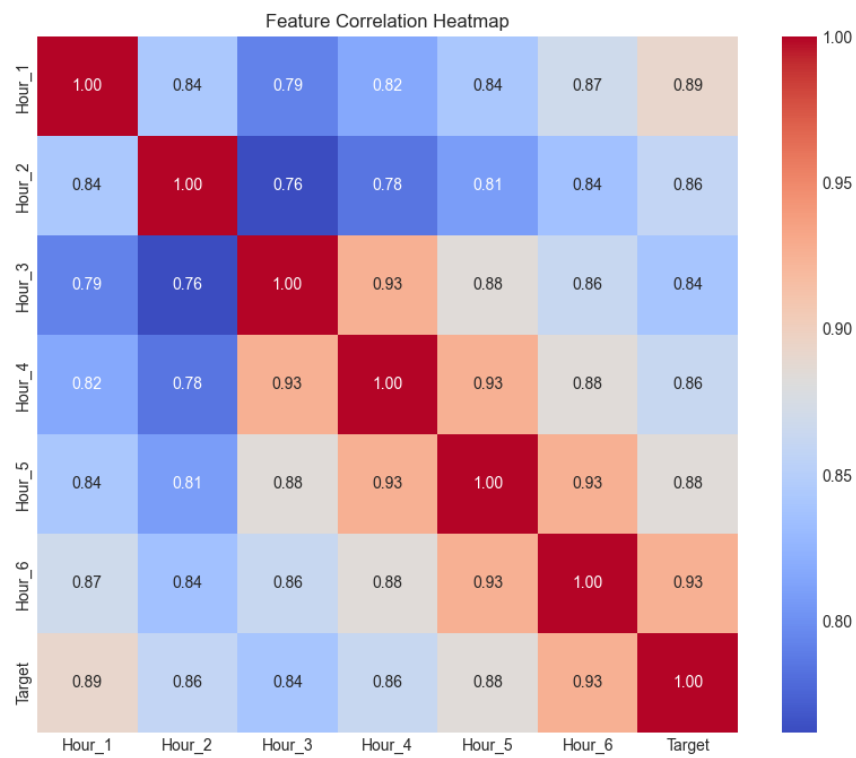


Figure 8: Correlation matrix of lags and cyclical features against the target variable.

6.6 Time Series Decomposition

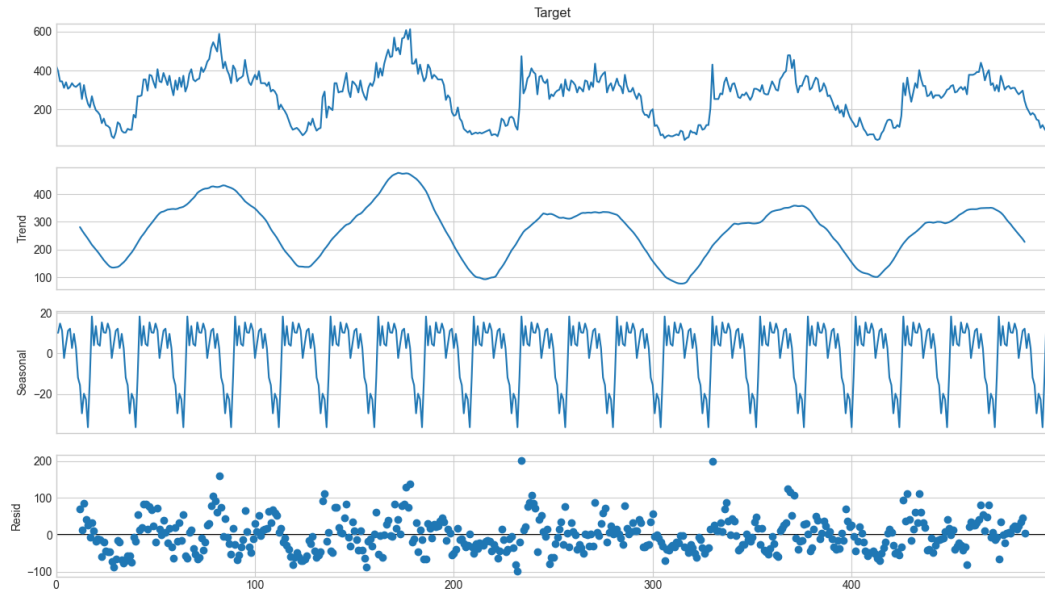


Figure 9: Seasonal decomposition of the dataset into Trend, Seasonal (Daily), and Random Noise.

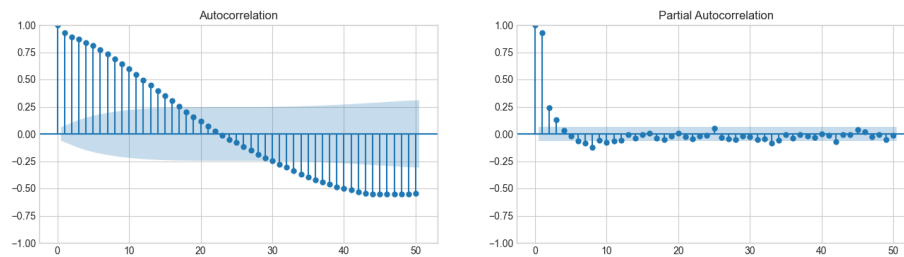


Figure 10: Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots used for feature lag selection.

6.7 Optuna Hyperparameter Optimization Logs

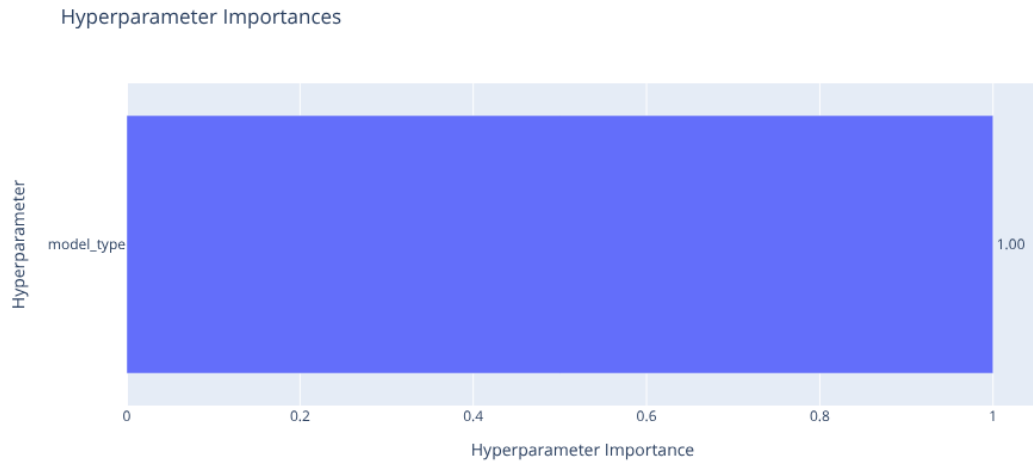
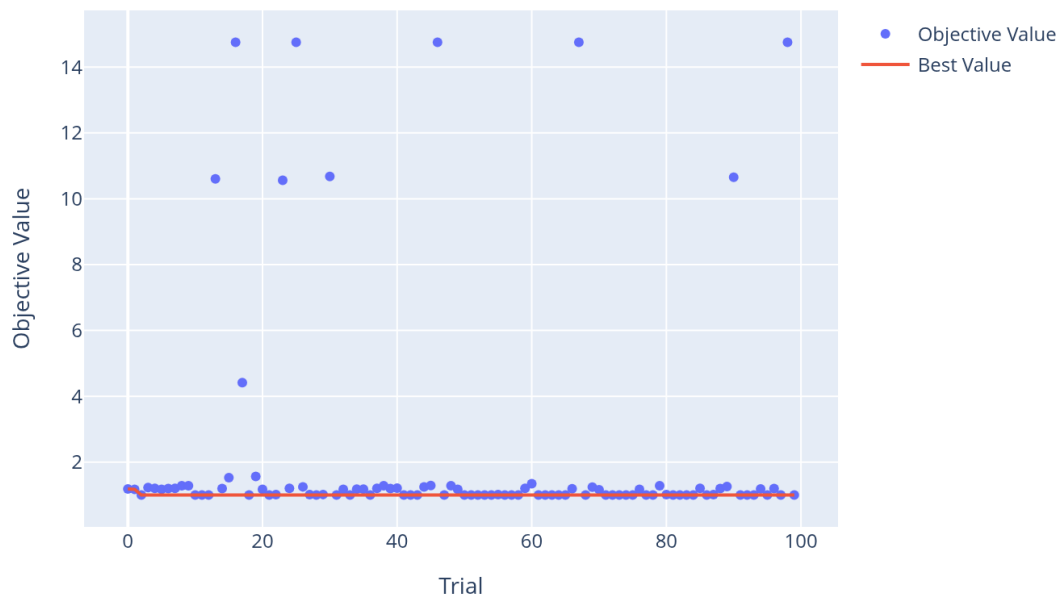
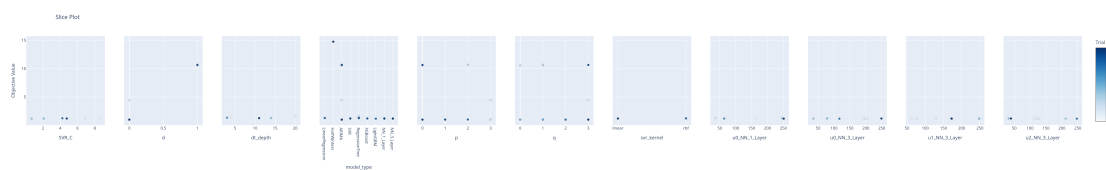


Figure 11: Optuna objective expansion showing the convergence toward minimal MAPE across 100 trials.

Optimization History Plot

**Figure 12:** Progress of the Optuna study across 100 trials, minimizing the MAPE objective.**Figure 13:** Parameter slice plot showing the impact of specific hyperparameter ranges on the objective value.

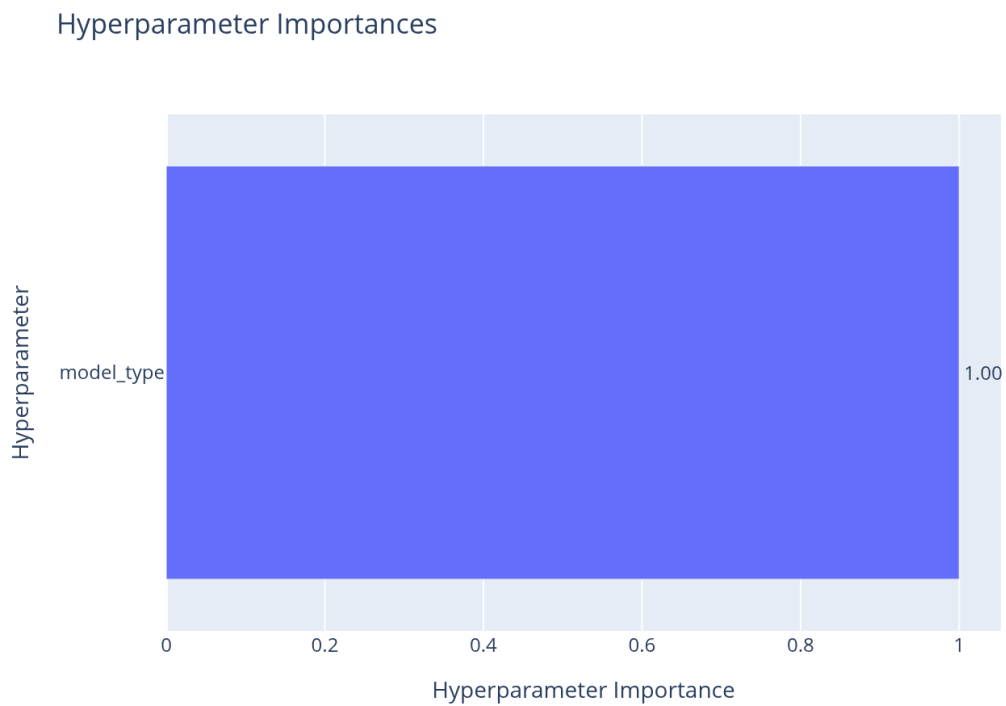


Figure 14: Hyperparameter importance plot identifying the most influential parameters for model precision.

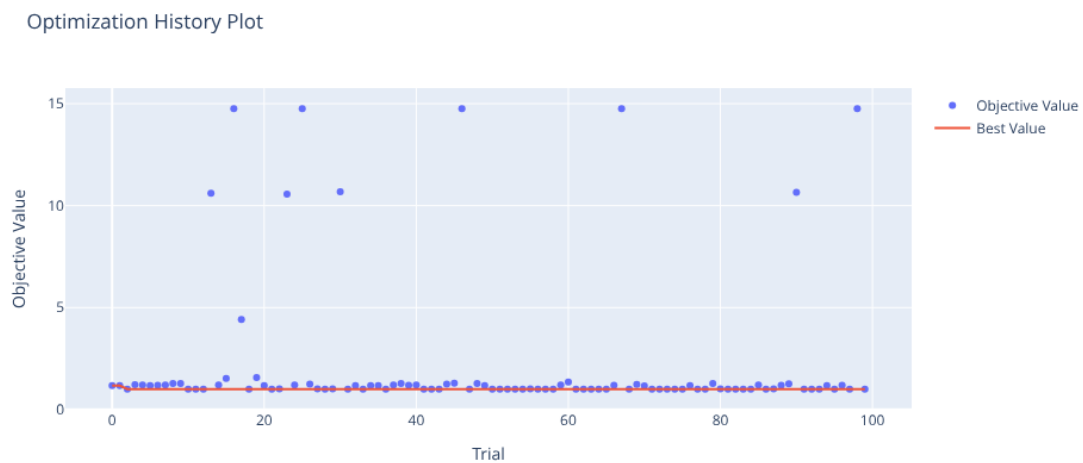


Figure 15: Intermediate optimization results showing early exploration of the hyperparameter search space.

6.8 Detailed Model Performance Benchmarks

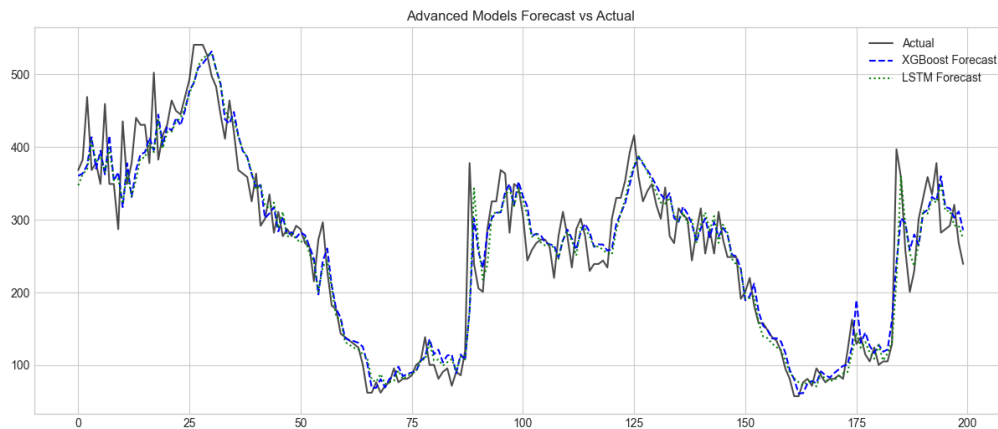


Figure 16: Consolidated overview of model performance across key metrics (RMSE, MAPE, MAE, and Latency).

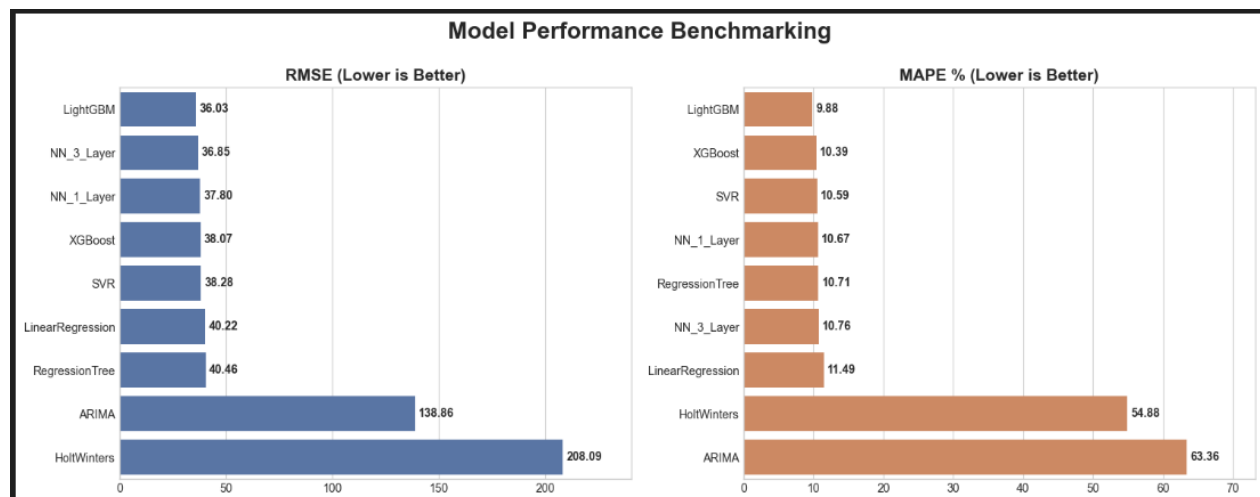


Figure 17: Detailed view of RMSE and MAPE metrics, highlighting the trade-off between absolute and percentage errors.

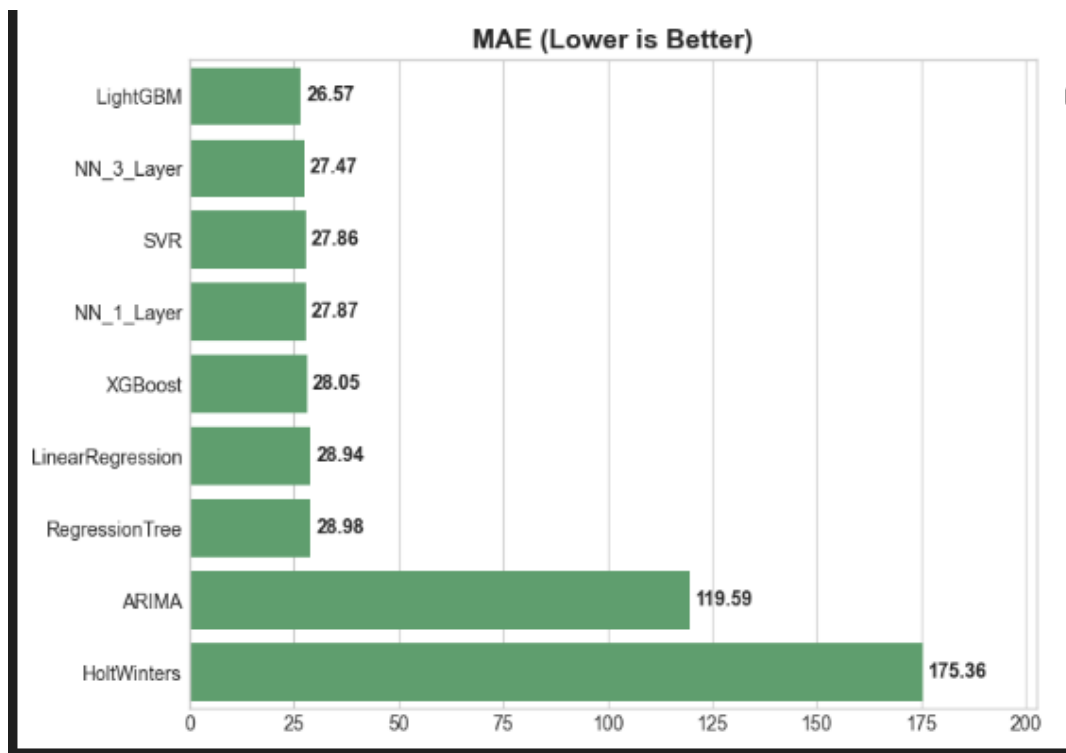


Figure 18: Detailed view of Mean Absolute Error (MAE) performance.

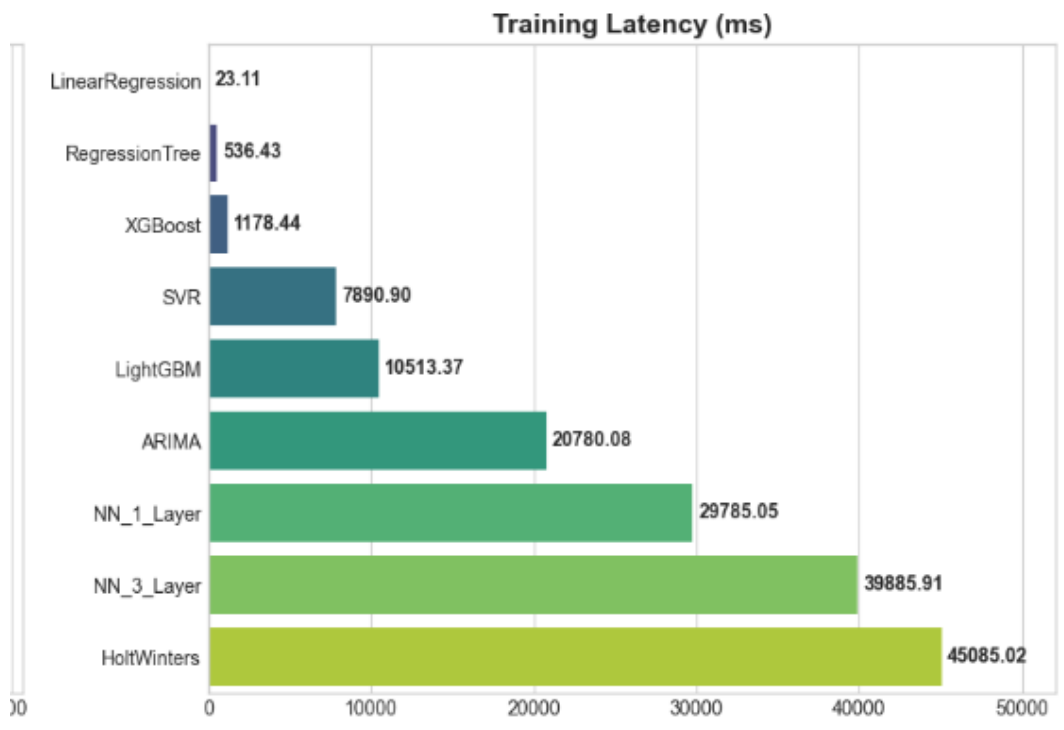


Figure 19: Detailed training latency comparison highlighting computational costs.

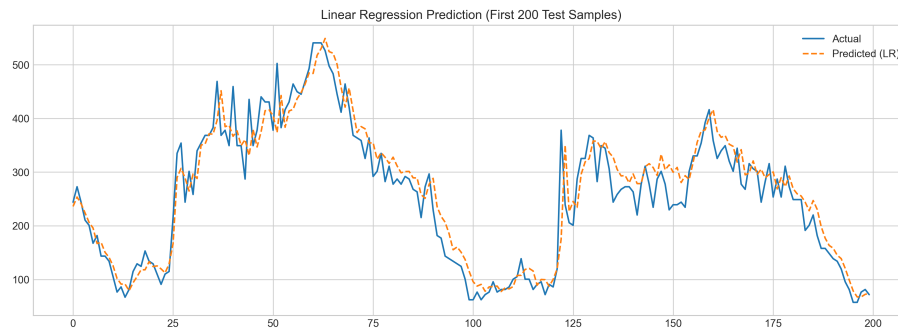
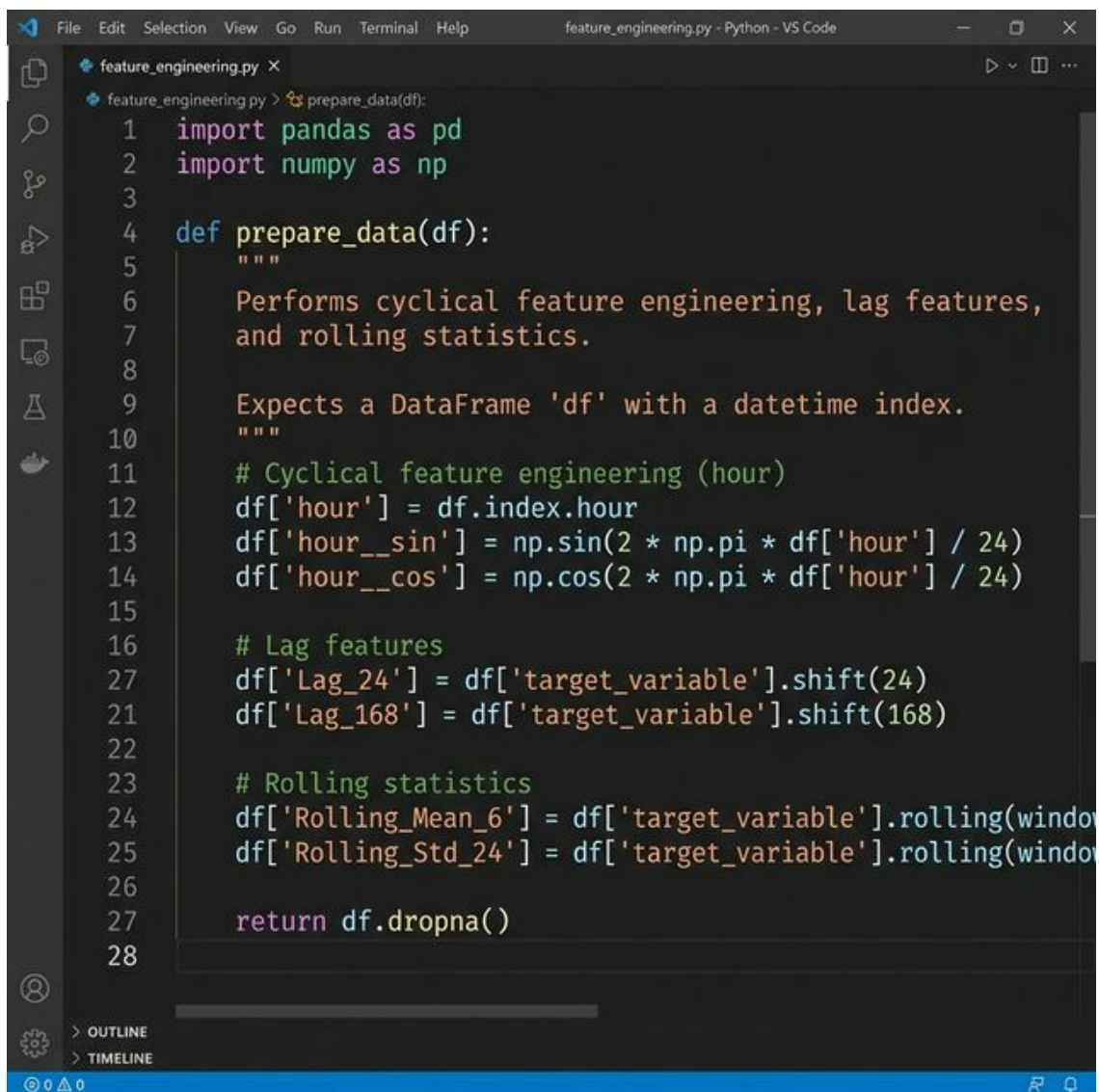


Figure 20: Visual calibration showing True vs. Predicted values for the Linear Regression baseline on the test set.

A screenshot of a Visual Studio Code editor window. The title bar at the top reads "feature_engineering.py - Python - VS Code". The editor is open to a file named "feature_engineering.py". The code is a Python function named "prepare_data(df)". It imports "pandas as pd" and "numpy as np". The function has a docstring describing its purpose: "Performs cyclical feature engineering, lag features, and rolling statistics. Expects a DataFrame 'df' with a datetime index." The function then performs three types of feature engineering: 1. Cyclical features: It calculates the hour from the index and creates sine and cosine features. 2. Lag features: It creates lagged versions of the target variable at 24 and 168 time steps. 3. Rolling statistics: It calculates the rolling mean and standard deviation of the target variable over windows of 6 and 24 time steps. Finally, it returns the DataFrame with unnecessary columns dropped.

```
feature_engineering.py X
feature_engineering.py > prepare_data(df):
1  import pandas as pd
2  import numpy as np
3
4  def prepare_data(df):
5      """
6      Performs cyclical feature engineering, lag features,
7      and rolling statistics.
8
9      Expects a DataFrame 'df' with a datetime index.
10     """
11     # Cyclical feature engineering (hour)
12     df['hour'] = df.index.hour
13     df['hour__sin'] = np.sin(2 * np.pi * df['hour'] / 24)
14     df['hour__cos'] = np.cos(2 * np.pi * df['hour'] / 24)
15
16     # Lag features
17     df['Lag_24'] = df['target_variable'].shift(24)
18     df['Lag_168'] = df['target_variable'].shift(168)
19
20     # Rolling statistics
21     df['Rolling_Mean_6'] = df['target_variable'].rolling(window=6).mean()
22     df['Rolling_Std_24'] = df['target_variable'].rolling(window=24).std()
23
24     return df.dropna()
```

Figure 21: Python implementation of the feature engineering pipeline.

Bibliography

References

- [1] Optuna Contributors. **Optuna: A next-generation hyperparameter optimization framework**. 2024. URL: https://optuna.org/#key_features (visited on 02/08/2026) (cit. on p. 6).