Comparative Analysis of Forecasting Models for Household Energy Consumption: A London Smart Meters Project

Hien Nguyen[1],

[1] Independent Researcher, [Wellington, New Zealand]

1 Introduction

Energy consumption forecasting is vital for modern smart grids, helping utilities optimize resources, stabilize the grid, and promote sustainable energy use. The rise of smart meters and detailed household consumption data has enabled the development of advanced forecasting models. This study compares three forecasting paradigms—classical statistical methods, machine learning, and deep learning—using the London smart meters dataset. Classical methods like AutoETS and ARIMA rely on historical temporal patterns for prediction. Machine learning enhances accuracy by incorporating multivariate features such as weather and sociodemographic data. Deep learning models excel at capturing complex, non-linear temporal dependencies that traditional methods cannot effectively model.

The fundamental question is whether increased model complexity translates to meaningful improvements in forecasting performance for residential energy consumption. By implementing a rigorous experimental framework that includes feature engineering, preprocessing pipelines, and consistent evaluation metrics across multiple households, we provide empirical evidence for the relative effectiveness of different modeling approaches. The research evaluates classical methods (AutoETS, ARIMA), machine learning algorithms (Lasso Regression, XGBoost Random Forest, LightGBM), and deep learning models (LSTM, TSMixer) using comprehensive performance metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and forecast bias analysis.

The methodological framework and implementation approaches presented in this report are adapted from the techniques outlined in "Modern Time Series Forecasting with Python - Second Edition" by Manu Joseph and Jeffrey Tackes [1]. The rest of the research is organized as follows: Section 2 introduces the background and various techniques used in time series forecasting. Section 3 mentions the data preparation process in subsection 3.1 and the exploratory data analysis in subsection 3.2. Chapter 4 covers methods, results and discussion, starting with classical models in subsection 4.1, following by machine learning models in subsection 4.2, then 4.3 is about deep learning models. Finally, the paper concludes in Section 5 and ends with Section 6 Appendix.

2 Background

Time series forecasting has evolved significantly over recent decades, progressing from classical statistical methods to sophisticated machine learning and deep learning approaches. This section explores the fundamental concepts and methodologies that form the backbone of modern time series analysis and prediction.

2.1 Time series fundamentals

Time series data consists of sequential observations taken over time, categorized into regular (fixed intervals like hourly or monthly) and irregular (varying intervals, such as patient lab tests) types. Stationarity is a fundamental concept in time series analysis, though most real-world time series are non-stationary. A stationary time series maintains the same probability distribution across different time windows.

Non-stationarity typically manifests in three ways: changes in mean over time (visible as trends), seasonality (predictable cyclical patterns, like temperature variations across seasons), and changes in variance over time (heteroscedasticity, where fluctuations increase or decrease over time). These characteristics are crucial to understand when modeling time series data, as many analytical approaches assume stationarity as a prerequisite condition.

2.2 Classical forecasting methods

Time series forecasting remains a fundamental challenge across numerous domains including economics, finance, operations, and environmental sciences. Two classical forecasting methodologies that have maintained their relevance and utility despite the emergence of more complex techniques: Exponential Smoothing (ETS) and AutoRegressive Integrated Moving Average (ARIMA) models.

2.2.1 Exponential Smoothing. Exponential smoothing (ETS) is a popular forecasting method developed in the 1950s that balances the strengths of naïve methods (which only use the most recent data point) and moving averages (which treat several recent observations as equally important). ETS uses a weighted average approach where all historical data matters, but recent data receives exponentially higher weights. Simple exponential smoothing (SES) works best for data without trends or seasonality, producing flat-line forecasts. Double exponential smoothing (DES) adds trend modeling capabilities through two smoothing equations. Triple exponential smoothing (Holt-Winters) incorporates seasonality using three parameters (α, β, γ) and can handle both additive and multiplicative seasonal patterns.

2.2.2 AutoRegressive Integrated Moving Average (ARIMA). ARIMA models are classic forecasting methods that complement ETS techniques but focus on autocorrelation rather than trend and seasonality. The ARIMA family includes: AR(p) models using p previous values as predictors; MA(q) models using past forecast errors; and the "I" component representing differencing (d) needed to achieve stationarity. The complete $ARIMA(p, d, q)$ model combines these elements. For seasonal data, $SARIMA(p, d, q)(P, D, Q)m$ extends the concept by adding seasonal parameters (P,D,Q) that operate on seasonal lags rather than consecutive ones, where m represents the seasonal period. SARIMAX further enhances the model by incorporating external variables. Unlike ETS methods, ARIMA requires time series to be stationary, often achieved through differencing operations.

2.3 Machine learning forecasting methods

Regression in machine learning predicts continuous variables using input features and targets. However, it is fundamentally incompatible with time series forecasting which requires predicting future values based on historical data. Time series forecasting is an extrapolation problem (predicting beyond known data), whereas regression typically handles interpolation (predicting within known data ranges), making the former inherently more challenging. Additionally, regression assumes independent and identically distributed (iid) samples, but time series data violates this assumption due to the dependency between sequential observations.

Despite these challenges, time series forecasting can be transformed into a regression problem by introducing memory through feature engineering techniques, with time-delay embedding and temporal embedding being powerful approaches that capture the temporal dependencies in sequential data. Time-delay embedding creates features by using lagged values of the time series as inputs to the model, effectively transforming the temporal structure into a spatial structure that machine learning algorithms can process. Meanwhile, temporal embedding techniques like T-Rep learn vector representations of time alongside feature extractors to capture important temporal characteristics such as trends, periodicity, and distribution shifts, which helps models better understand the underlying patterns in time series data.

2.3.1 Lasso Regression. Linear models can become overly complex when coefficients reach extreme values, where small input changes cause dramatic prediction fluctuations. Weight decay regularization addresses this by adding a penalty term to the loss function that constrains coefficient magnitudes. The regularized sum of squares (RSS) combines the traditional error term with a regularization component controlled by parameter λ. Common regularization approaches include L1 norm (lasso regression) and L2 norm (ridge regression), with L1 defined as the sum of absolute coefficient values. This regularization serves dual purposes: it reduces model complexity and performs implicit feature selection by minimizing or eliminating coefficients for less influential features.

2.3.2 Ensemble learning. Random Forest represents an ensemble technique that constructs numerous decision trees during its training phase, then combines their outputs to achieve greater accuracy and resilience. This approach performs exceptionally well for classification and regression problems by mitigating overfitting and strengthening predictive capabilities through bagging and random feature selection.

Gradient-boosted decision trees are powerful ensemble learning models that sequentially build multiple decision trees, with each new tree focusing on correcting the errors made by previous trees, resulting in a strong predictive model that combines the strengths of many weak learners. Though Random Forests and gradient-boosted decision trees share similar model structures and inference methods, they employ distinct training algorithms. While XGBoost is typically employed for training gradient-boosted decision tree models, it can also be utilized to develop standalone random forests. In our context, we implement standalone random forest models using XGBoost, rather than using random forest as a foundation for gradient boosting.

Another key technique is subsampling, which can be applied to rows and columns. Row subsampling resembles bootstrapping, where each ensemble candidate is trained on a dataset subsample. Column subsampling parallels random feature selection in Random Forest. Both techniques introduce regularization effects to the ensemble and help reduce generalization errors. Some gradient-boosted tree implementations, such as LightGBM, also incorporate L1 and L2 regularization directly in their objective functions for additional model control.

2.4 Deep learning forecasting methods

Deep learning models frequently use fully connected (dense) layers where each neuron connects to all neurons in the previous layer, allowing for comprehensive pattern recognition. Feed Forward Networks (FFNs) exemplify this architecture, with information moving one-way from input to output through interconnected layers. The structure includes an input layer matching input dimensions, an output layer sized to desired outputs, and intervening hidden layers. The network's complexity is determined by two key hyperparameters: the number of hidden layers and units per layer. In time series applications, FFNs function as encoders (converting time series into regression problems) and decoders (processing encoded information to generate forecasts).

2.4.1 Long short-term memory networks. Recurrent neural networks (RNNs) were specifically designed to handle sequential data, addressing limitations of FFNs in processing temporal information. Building on this foundation, Hochreiter and Schmidhuber introduced Long Short-Term Memory (LSTM) networks in 1997 to solve the vanishing and exploding gradient problems that plagued vanilla RNNs.

LSTMs incorporate a novel memory cell component that functions alongside the traditional hidden-state memory. This design includes multiple specialized gates that control information flow-reading, adding, and forgetting information from these memory cells. The memory cell creates a gradient highway, allowing signals to pass relatively unhindered through the network, which effectively prevents vanishing gradients and enables learning of long-term dependencies in sequential data.

2.4.2 TSMixer. While deep learning is often viewed as entirely data-driven, architectural design choices still introduce important inductive biases. Different architectures perform better on specific data types because of these inherent biases-CNNs excel with images due to their spatial inductive bias, while other architectures may better suit sequential data.

As Transformer-based models gained prominence, a parallel research track emerged using Multi-Layer Perceptrons (MLPs) as primary learning units. This trend began in 2021 when MLP-Mixer demonstrated state-of-the-art performance in vision tasks using only MLPs instead of Convolutional Neural Networks. In 2023, Google researchers extended this approach to time series forecasting with TSMixer.

TSMixer draws inspiration from Transformer architecture while replacing key components with MLPs. Where Transformers use Multi-Head Attention to mix information across timesteps, followed by Position-Wise Feed Forward networks to mix features, TSMixer implements dedicated Time-Mixing and Feature-Mixing components within each Mixer block. The model processes multivariate time series through multiple sequential mixer layers before passing the learned representations to a temporal projection layer that generates the final forecast. In this context, "features" refers to different time series in a multivariate setting, creating a streamlined yet powerful architecture for time series prediction.

3 Data preprocessing and exploratory analysis

3.1 Data preparation

London Smart Meters dataset contains energy consumption readings for 5,567 London households in the UK Power Networks-led Low Carbon London project between November 2011 and February 2014. Readings were taken at half-hourly intervals. Some metadata about the households is also available as part of the dataset. Jean-Michel D also enriched the dataset with weather and UK bank holiday data. Each household has a unique LCLid and is stored in hhblock_dataset folder. Algorithm 1 implements a data processing pipeline for London smart meter data, transforming raw half-hourly energy consumption records into a structured format enriched with household characteristics, weather conditions, and holiday information.

3.2 Exploratory data analysis

Exploratory Data Analysis (EDA) utilizes visualization techniques to reveal patterns, identify anomalies, and develop hypotheses, with thorough dataset understanding enabling better feature engineering and model selection that ultimately enhances performance.

3.2.1 Missing values. The missingno package provides powerful visualization tools for analyzing missing data patterns in time series datasets. Figure 1a displays dates versus households, clearly showing misalignment in data collection periods through white gaps at series beginnings and endings. On the other hand, smaller gaps represent genuine missing values. A complementary sparkline indicates the density of missing values at each timestamp, with rightward extension showing completeness and leftward movement indicating more missing data. Figure 1b examines a specific household (MAC000193) to illustrate missing data between 2012-10-18 and 2012-10-19.

An artificial missing data section spanning 2012-10-07 to 2012-10-08 is introduced to evaluate different imputation techniques, and deliberately null out values during this period (Figure 1c). We then apply various imputation methods and assess their accuracy using Mean Absolute Error (MAE)-the average absolute difference between imputed values and the actual data across all time steps. This approach will allow us to quantitatively compare the effectiveness of different missing data handling strategies.

One approach involves calculating an hourly profile from the data-determining mean consumption for each hour-and using these averages to fill missing values. It leverages the cyclical nature of energy consumption patterns, where households typically exhibit consistent usage behaviors at specific times of day, making hourly averages more representative than global means for imputation. This method produces a generalized curve. The resulting profile successfully captures the expected hourly fluctuations in energy usage, yielding a Mean Absolute Error (MAE) of 0.121, as illustrated in Figure 2a.

We can further enhance the imputation strategy by developing specific profiles for each day of the week. The refinement acknowledges the logical assumption that usage patterns differ between weekdays and weekends. We create a more nuanced imputation model by calculating separate average hourly consumption profiles for Monday, Tuesday, and so on. While visually similar to the general hourly profile in our example (primarily because we are imputing a weekday, and weekday patterns tend to share similarities), this day-specific approach delivers superior results with a reduced MAE of 0.117 (Figure 2b). Given this improved accuracy, we will adopt the weekday-specific hourly profile method for all subsequent imputation tasks.

3.2.2 Analysing time series data. Given the thousands of households that participated in the survey, only household MAC000193 is analyzed as a representative example for visualization purpose. The line chart Figure 3 illustrates the fundamental seasonal patterns in household energy consumption for MAC000193. The visualization reveals clear cyclical behavior with consumption peaks occurring during winter months and troughs during summer period, reflecting London's climate patterns where increased heating requirements during colder months drive higher energy usage. The comparative line chart Figure 4 examines the relationship between temperature fluctuations and energy consumption patterns. We can visually assess our hypothesis that temperature significantly influences energy usage by plotting these variables together. The chart provides insights into the inverse relationship between these factors. Figure 5 displays average monthly energy consumption across multiple years (2012-2013), highlighting consistent seasonal patterns and notable deviations. October 2013 is an anomaly, where consumption patterns diverged from the previous year. The accompanying temperature data reveals that October 2013 maintained warmer temperatures for an extended period, explaining this consumption pattern difference and demonstrating how weather variations can disrupt otherwise predictable seasonal energy usage trends (Figure 6 and 7). Box plot Figure 8 shows the median energy consumption across different hours of the day. The data shows consumption peaks beginning at 9 A.M., accompanied by increased variability during daytime hours. Another critical temporal dimension beyond hourly patterns is the distinction between weekdays and weekends, as consumption behaviors likely differ significantly between work days and rest days. Figure 9 calendar heatmap offers a visualization of energy consumption patterns across days of the week and hours of the day, with lighter colors indicating higher consumption values. The heatmap efficiently condenses what would otherwise require multiple separate charts into a single, information-dense visualization. It reveals that Monday through Saturday share similar bimodal consumption patterns with distinct morning and evening peaks. On the other hand, Sunday exhibits a unique pattern characterized by more consistent elevated consumption throughout the day, reflecting different household behaviors on weekends.

Autocorrelation measures the relationship between a time series and lagged versions of itself, unlike standard correlation which examines relationships between different variables. The temporal dependency is significant, as most series exhibit strong connections to their previous values—a fundamental concept incorporated into many forecasting models. The degree to which past observations influence current values provides crucial insights into the underlying data structure and informs model selection. Figure 10 represents the autocorrelation and partial autocorrelation plots for the energy consumption time series. The partial autocorrelation function reveals that the first lag (t-1) exerts the strongest influence on current values, with this effect rapidly diminishing to near zero for subsequent lags. This pattern indicates that energy consumption on any given day is predominantly influenced by the consumption level of the immediately preceding day, suggesting a first-order autoregressive process.

3.3 Feature engineering and model selection

Based on the exploratory data analysis of household energy consumption patterns, several key insights emerge for feature engineering and model selection. The autocorrelation analysis (Figure 10) indicates strong first-order temporal dependency, with the first lag (t-1) showing the strongest influence on current values, suggesting that incorporating lag features, particularly the previous day's consumption, would be highly beneficial for forecasting models. The distinct hourly patterns (Figure 8) showing peaks beginning at 9 A.M. with increased daytime variability suggest that hour-of-day features are essential. The calendar heatmap (Figure 9) reveals significant differences between weekdays and weekends, particularly Sunday's unique pattern, indicating that day-of-week features would be valuable predictors. The clear seasonal patterns with winter peaks and summer troughs (Figure 3) highlight the importance of including month-of-year or season indicators, and time-based feature extraction techniques like Fourier transforms would help capture the cyclical patterns observed in the data. The strong inverse relationship between temperature and energy consumption (Figure 4) and the anomaly in October 2013 explained by temperature variations (Figures 6-7) emphasize the importance of including weather variables, particularly temperature, as key predictors. Given the large number of households, FeatureHasher could be valuable for dimensionality reduction while preserving important patterns across multiple households.

For model selection, the clear seasonal patterns and trends in the line charts make ETS models suitable for capturing these components explicitly, particularly for households with regular, consistent patterns without many external influences. The strong first-order autocorrelation pattern suggests ARIMA models with appropriate AR terms would be effective, such as an ARIMA(1,1,0) with seasonal components likely performing well. Given the multiple potential predictors, Lasso regression would be valuable for automatic feature selection while preventing overfitting, particularly when incorporating many external variables and lag features. Random Forest models would be effective at capturing the non-linear relationships between temperature and consumption, naturally handling complex interactions between temporal features and remaining robust to anomalies. Finally, the complex temporal patterns across multiple dimensions make this dataset suitable for TSMixer's specialized architecture, with its ability to mix information across both time and features being valuable for capturing the interactions between consumption patterns and external factors like temperature.

4 Methods, results and discussion

4.1 Classical models

Exponential smoothing (ETS) is a forecasting method that assigns exponentially decreasing weights to historical data, with recent observations receiving higher importance. Simple exponential smoothing works for data without trends or seasonality, while double and triple exponential smoothing (Holt-Winters) accommodate trends and seasonal patterns. ARIMA models focus on autocorrelation patterns in time series data, combining autoregressive (AR) components using previous values, moving average (MA) components using past errors, and integration (I) through differencing to achieve stationarity.

4.1.1 Implementation. Algorithm 2 presents a framework for analyzing household energy consumption patterns and developing predictive models using AutoETS and ARIMA. The pipeline initializes the environment, loads household data with ACORN socio-demographic classifications, and samples across different socioeconomic segments for representative analysis. It then processes smart meter data blocks, converting them from compact storage to an expanded format that separates time series measurements from static household features. The temporal data is then chronologically partitioned into training (pre-2014), validation (January 2014), and test (February 2014) sets, with missing values addressed through the weekday-specific hourly profile imputation technique. The algorithm implements multiple forecasting methodologies, including ARIMA, and AutoETS, first on a representative household for detailed visualization and analysis, then scaling to hundreds of households to assess model robustness across diverse consumption patterns. Performance evaluation employs multiple metrics (mean absolute error, mean squared error, root mean squared error, and forecast bias - defined as the difference between the sum of the forecast and the sum of the actual values, expressed as a percentage over the sum of all actuals) to provide an assessment of prediction accuracy. The final stage combines the training and validation datasets to develop optimized models, which are then evaluated on the previously untouched test data to measure generalization capability.

4.1.2 Results and discussion. Following the implementation, we evaluate the performance of both AutoETS and ARIMA models across the sampled households. By applying consistent evaluation metrics across multiple households, we can assess the overall performance differences, and examine how these models behave across different consumption patterns. The results presented below demonstrate the comparative effectiveness of these approaches when applied to residential energy consumption forecasting.

Metrics	ARIMA	AutoETS
MSE	0.0694	0.0586
RMSE	0.2634	0.2422
MAE	0.1327	0.1191
Forecast Bias	13.3147	10.7079

Table 1. Performance of classical methods.

Table 1 shows a comparison of performance metrics between ARIMA and AutoETS models for the London smart meters dataset. The AutoETS model outperforms the ARIMA model across all four error metrics, though the margin of improvement varies. For MSE (Mean Squared Error), AutoETS achieves 0.0586 compared to ARIMA's 0.0694, representing approximately a 15.6% reduction in squared error. AutoETS produces forecasts with fewer large deviations from actual values. The RMSE (Root Mean Squared Error) values follow a similar pattern, with AutoETS scoring 0.2422 versus ARIMA's 0.2634. Since RMSE is in the same units as the original data (kWh for energy consumption), AutoETS predictions typically deviate about 0.02 kWh less from actual values than ARIMA predictions. For MAE (Mean Absolute Error), AutoETS again demonstrates superior performance with 0.1191 compared to ARIMA's 0.1327. The 10.2% improvement suggests that AutoETS produces forecasts that are, on average, closer to the actual consumption values regardless of whether the errors are positive or negative. Regarding forecast bias, both models show negative values (-13.3147 for ARIMA vs. -10.7079 for AutoETS), indicating a systematic tendency to underestimate energy consumption. While neither model achieves ideal unbiased forecasting (which would be closer to zero), AutoETS demonstrates less pronounced underestimate.

The comparative analysis of AutoETS and ARIMA models across multiple error metrics reveals interesting performance patterns when applied to household energy consumption forecasting. As illustrated in Figures 11, both models demonstrate similar error distributions. The RMSE distribution (Figure 11a) shows that AutoETS generally achieves slightly lower error rates than ARIMA, with both models exhibiting right-skewed distributions centered around 0.2, indicating that most households can be forecast with reasonable accuracy. The MSE distributions (Figure 11b) further emphasize this pattern, with a heavy concentration of values near zero and a long tail of outliers, suggesting that while both models perform well for most households, they struggle with certain challenging consumption patterns.

The MAE distributions (Figure 11c) provide additional insight, showing AutoETS maintaining a slight edge over ARIMA, with most values falling below 0.2, which indicates that the typical absolute deviation between forecasted and actual values is relatively small for both models. Perhaps most revealing is the forecast bias distribution (Figure 11d), AutoETS exhibits a more centralized bias distribution around zero compared to ARIMA, whose distribution is slightly shifted toward positive values, which suggests that while ARIMA tends to overestimate energy consumption, AutoETS produces more balanced forecasts with less systematic error. These findings align with previous research [2, 3] suggesting that ETS models often outperform ARIMA models in forecasting applications despite potentially fitting training data less closely, demonstrating the principle that good in-sample fit does not necessarily translate to superior forecasting performance.

4.2 Machine learning models

Time series forecasting differs fundamentally from regression. It involves extrapolation (predicting beyond known data) rather than interpolation. Thus, it violates the independent and identically distributed (iid) assumption due to sequential dependencies. However, time series problems can be reformulated as regression tasks through feature engineering techniques like time-delay embedding and temporal embedding. Various machine learning approaches can then be applied, including Lasso regression, Random Forests, and gradient-boosted decision trees.

4.2.1 Implementation. The Feature Engineering for Time Series Data in Algorithm 5 transforms raw temporal data into meaningful predictive features that enhance machine learning model performance. The dual-function algorithm addresses two fundamental aspects of time series feature engineering: temporal feature extraction and lag feature creation.

The AddTemporalFeatures function systematically extracts time-based characteristics from datetime columns, converting temporal information into numerical features that capture cyclical patterns and temporal relationships. Key temporal features include day of week, month, hour, and elapsed time since a reference point, which are particularly valuable for capturing seasonality and business cycles in energy consumption data. The AddLags function implements time-delay embedding by creating lagged versions of target variables, effectively transforming the temporal structure into spatial features that machine learning algorithms can process. The approach addresses the fundamental challenge of time series forecasting where sequential dependencies must be captured within a regression framework. The lag selection strategy incorporates short-term dependencies (1-5 day lags), medium-term patterns (46-50 day lags), and long-term seasonal effects (weekly lags around 334-338), enabling models to capture both immediate autocorrelations and longer-term seasonal dependencies characteristic of household energy consumption patterns.

Algorithm 5's practical application involves loading preprocessed smart meter data, applying both temporal and lag feature engineering techniques, and systematically saving the enhanced datasets for subsequent modeling. The memory-efficient approach using 32-bit data types and the preservation of data partitioning (train/validation/test) ensures that the feature engineering process maintains experimental integrity while optimizing computational resources.

Unlike classical forecasting methods that rely solely on the target variable's historical values, machine learning models can leverage multiple independent variables to enhance prediction accuracy. Consequently, beyond the temporal and lag feature engineering discussed previously, machine learning approaches require additional preprocessing of non-numeric features (see details in Algorithm 6).

Building upon the above preprocessing foundation, machine learning workflow integrates feature engineering with model training and evaluation. Algorithm 7 implements a machine learning pipeline for household energy consumption forecasting that evaluates multiple models across diverse households.

The program begins by loading and combining preprocessed smart meter datasets (obtained from Algorithm 5 - Feature Engineering), then configures feature extraction parameters including continuous variables (weather data, lag features), categorical variables (temporal features, weather conditions), and missing value imputation strategies. For each household in the dataset, the pipeline systematically applies three machine learning models: Lasso Regression with cross-validation for feature selection, XGBoost Random Forest (XGB-RF) for capturing non-linear relationships, and LightGBM for efficient gradient boosting. The feature engineering process includes one-hot encoding of categorical variables with consistent encoder fitting on training data and transformation on test data to prevent data leakage. Each model undergoes training on household-specific data and generates predictions for the test period, with performance evaluation using multiple metrics including MSE, RMSE, MAE, and forecast bias.

4.2.2 Results and discussion. The comparison between classical and machine learning forecasting methods has become increasingly important as machine learning techniques gain popularity in time series forecasting applications. It helps establish whether the added complexity of machine learning models translates to meaningful improvements in forecasting accuracy, and provides objective benchmarks for method selection.

Metrics	AutoETS	Lasso Regression	XGB-RF	LightGBM
MSE	0.0586	0.0258	0.0292	0.0265
RMSE	0.2422	0.1607	0.1709	0.1628
MAE	0.1191	0.0775	0.0768	0.0734
Forecast Bias	-10.7079	-1.3277	0.4232	-1.7538

Table 2. Comparison performance of classical and machine learning forecast methods.

Table 2 presents a performance comparison between classical and machine learning approaches applied to household energy consumption forecasting, using multiple evaluation metrics to assess their relative effectiveness. AutoETS, representing the classical methodology, demonstrates competitive performance with an MSE of 0.0586, RMSE of 0.2422, and MAE of 0.1191. Its forecast bias of -10.7079 indicates systematic underestimation that is relatively moderate compared to other methods. In contrast, machine learning methods show superior overall performance, with Lasso Regression achieving the best results across all metrics (MSE of 0.0258, RMSE of 0.1607, and MAE of 0.0775), followed by LightGBM (MSE of 0.0265, RMSE of 0.1628, and MAE of 0.0734) and XGB Random Forest (MSE of 0.0292, RMSE of 0.1709, and MAE of 0.0768). Lasso Regression achieves approximately 56% lower MSE compared to AutoETS, suggesting that for multivariate time series problems with rich feature sets including weather data, temporal features, and lag variables, machine learning methods can effectively leverage the additional information to improve forecasting accuracy. Regarding forecast bias, most methods exhibit negative values indicating systematic underestimation of energy consumption. The magnitude varies slightly between machine learning models, with XGB Random Forest showing the smallest absolute bias (0.4232) suggesting more balanced predictions compared to Lasso Regression (-1.3277).

Figure 12 presents the error distribution and forecast bias analysis for three machine learning models—Lasso Regression, XGB Random Forest, and LightGBM—highlighting their comparative performance, consistency, and calibration in forecasting household energy consumption across diverse households. The RMSE and MSE distributions show remarkably consistent patterns across all three models. Lasso Regression demonstrates the tightest error distribution with the lowest median values and smallest interquartile range, indicating superior and more consistent performance across households. The box plots reveal that Lasso achieves the most concentrated error distribution around 0.15 RMSE, with minimal outliers extending beyond 0.4. XGB Random Forest and LightGBM exhibit similar distribution patterns, with slightly higher median errors and broader interquartile ranges compared to Lasso. Both tree-based models show comparable performance levels, with LightGBM displaying marginally tighter distributions. The presence of outliers extending to 0.8-1.0 RMSE for both tree-based models suggests they struggle with certain household consumption patterns that Lasso handles more robustly.

The forecast bias distributions reveal the most striking differences between models (as illustrated in Figure 12d). Lasso Regression shows the most concentrated distribution around zero bias, with a narrow spread where most values lie between -4 and +2. It appears to have the least systematic bias among the three models, as indicated by the box plot which shows a tight interquartile range with minimal outliers. XGB Random Forest displays a wider distribution compared to Lasso. It exhibits a slight positive bias tendency, with the distribution skewed toward positive values. The predictions show more variability, with forecast bias ranging from -8 to +10, and several outliers are visible in the box plot. LightGBM exhibits the widest distribution of forecast bias and the most variability, with bias values extending roughly from -10 to +8. Its distribution appears slightly right-skewed, indicating a positive bias. The box plot for LightGBM shows the largest interquartile range and the most extreme outliers. All models demonstrate slight positive bias tendencies, meaning they tend to overpredict on average. Additionally, LightGBM and XGB Random Forest are more prone to extreme bias values, indicating potential sensitivity to certain data patterns.

The distribution analysis reveals that Lasso Regression provides the most robust and reliable forecasting performance across the household dataset. Its consistently tight error distributions and minimal bias indicate superior generalization capabilities. The tree-based models have competitive average performance, but show greater variability in their predictions, suggesting they may require additional regularization or hyperparameter tuning to achieve more consistent performance across different household consumption profiles.

Examining daily forecast performance provides crucial insights into model behavior during different consumption patterns and extreme events, revealing how well each algorithm captures the temporal dynamics and variability inherent in household energy consumption. The detailed temporal analysis is essential for understanding model reliability and responsiveness, as it demonstrates whether forecasting accuracy remains consistent across varying demand scenarios or deteriorates during peak consumption periods. Figure 13 and 14 are forecast comparison charts between actual consumption and predicted consumption for the test household during February 1-8, 2014. Both models face significant challenges in accurately predicting the extreme consumption peaks that occur on February 4th and 5th, where actual consumption reaches values above 1.0. Lasso Regression shows better peak-following behavior, with its predictions more closely tracking the magnitude and timing of consumption spikes. The model captures approximately 40-50% of the peak values, demonstrating reasonable responsiveness to high-demand periods. XGB Random Forest exhibits more conservative peak predictions, consistently underestimating the magnitude of consumption spikes. The model appears to be more heavily regularized, resulting in smoother but less responsive forecasts during extreme consumption events. Both models successfully capture the underlying daily consumption patterns. The models demonstrate a comparable understanding of the household's consumption rhythm, with predictions following the general shape and timing of actual consumption cycles throughout the week-long test period.

4.3 Deep learning models

Deep learning models commonly employ fully connected layers where neurons connect comprehensively across layers, with Feed Forward Networks (FFNs) exemplifying one-way information flow architecture from input to output through hidden layers. Long Short-Term Memory (LSTM) networks, introduced in 1997, address sequential data processing limitations by incorporating specialized memory cells and gates that control information flow, creating gradient highways to prevent vanishing gradients and enable long-term dependency learning. While deep learning appears data-driven, architectural choices introduce important inductive biases that make certain models better suited for specific data types. TSMixer, developed by Google in 2023, draws inspiration from Transformer architecture but replaces Multi-Head Attention with dedicated Time-Mixing and Feature-Mixing components using Multi-Layer Perceptrons (MLPs) as primary learning units. The model processes multivariate time series through sequential mixer layers before generating forecasts through a temporal projection layer, creating a streamlined architecture for time series prediction.

4.3.1 Implementation. While machine learning models often perform better with one-hot encoding because they rely on explicit feature independence and interpretability, deep learning models can leverage their representational learning capabilities to extract meaningful patterns even from the compressed, collision-prone feature space that hashing creates. Feature hashing is particularly advantageous for deep learning models over one-hot encoding because deep learning architectures can effectively learn meaningful representations from the compressed, fixed-dimensional feature space that hashing provides, while handling hash collisions through their inherent ability to discover complex patterns. The key advantage is memory efficiency and scalability - feature hashing maps potentially massive categorical feature spaces to a fixed dimension, making it ideal for high-cardinality features like user IDs, product codes, or large vocabularies that would create prohibitively sparse one-hot vectors. Thus, in this section, we will use feature hasher for feature preprocessing instead of one-hot encoding like previous section. Algorithm 8 depicts all the preprocessing steps on data obtained from Algorithm 5. Algorithm 8 implements a feature preprocessing pipeline specifically designed for deep learning models. The process begins with systematic missing value imputation using configurable strategies including backward fill, forward fill, and zero fill for specified columns, followed by default imputation using column means for numeric features and "NA" for categorical features. The core innovation lies in the categorical feature encoding approach, which employs feature hashing to transform high-cardinality categorical variables into fixed-dimensional dense representations rather than sparse one-hot vectors. Each categorical column is processed through a FeatureHasher that maps string values to a predetermined number of hash features, creating memory-efficient encodings that are particularly suitable for deep learning architectures. The algorithm then consolidates all hashed categorical features with the original continuous and boolean features, converts boolean variables to integer format for computational efficiency, and applies standard normalization to continuous features using StandardScaler.

Algorithm 9 implements a deep learning pipeline for household energy consumption forecasting using state-of-the-art time series models. The process begins by loading preprocessed smart meter data and applying feature hashing-based preprocessing to handle categorical variables efficiently while maintaining fixed input dimensions suitable for neural networks. The pipeline configures two deep learning architectures - LSTM for sequential pattern learning, and TSMixer for MLP-based time series mixing - each with specific hyperparameters optimized for 96-timestep input sequences and 48-timestep forecast horizons.

The algorithm systematically processes data for 50 households by converting raw datasets into Darts TimeSeries format with 30-minute frequency, creating temporal overlap between training and test periods to ensure continuity, and applying separate scaling transformations for target variables and covariates. For each model configuration, the pipeline executes training on household-specific time series data with covariates, generates forecasts for the test period, and evaluates performance using multiple metrics. The final output combines predictions and performance metrics across all models and households, enabling a comparison of deep learning approaches for residential energy consumption forecasting while leveraging temporal patterns and external covariates like weather data and engineered features.

4.3.2 Results and discussion. In the previous section, we conducted a comparison between classical statistical models and machine learning approaches to establish baseline performance benchmarks. Following the established analytical tradition of progressive model complexity evaluation, this section extends our investigation by comparing machine learning models against state-of-the-art deep learning architectures.

Metrics	Lasso Regression	LSTM	TSMixer
MSE	0.0258	0.0107	0.0008
RMSE	0.1607	0.1038	0.0286
MAE	0.0775	0.0525	0.0171
Forecast Bias	-1.3277	5.5132	12.8070

Table 3. Comparison performance of machine learning and deep learning forecast methods.

Table 3 presents a comparative analysis of forecasting performance across three different modeling approaches: machine learning (Lasso Regression) and deep learning methods (LSTM and TSMixer). TSMixer emerges as the clear winner across all evaluation metrics, demonstrating superior forecasting accuracy with the lowest MSE (0.0005), RMSE (0.0268), and MAE (0.0171). However, it demonstrates the highest forecast bias at 12.8070. LSTM shows moderate performance, ranking second with MSE of 0.0107, RMSE of 0.1038, MAE of 0.0525, and a moderate positive forecast bias of 5.5132. Lasso Regression performs least favorably among the three models, with the highest error metrics: MSE (0.0258), RMSE (0.1607), and MAE (0.0775). Interestingly, it has the smallest forecast bias of -1.3277. The results strongly support the effectiveness of modern deep learning architectures for time series forecasting. TSMixer's superior performance likely stems from its ability to capture complex temporal patterns and dependencies that traditional machine learning methods cannot effectively model.

Figure 15 contains four distribution charts that provide a comparison of LSTM and TSMixer performance across key forecasting metrics, revealing significant differences in model behavior and accuracy. In the RMSE distribution, TSMixer demonstrates superior performance with a highly concentrated distribution near zero, peaking around 0.02-0.03. LSTM shows a much wider spread extending to 0.25, with box plots confirming TSMixer's tighter interquartile range and minimal outliers compared to LSTM's broader distribution with several extreme values. The MSE distribution pattern is even more pronounced, with TSMixer achieving extremely low values concentrated near zero. LSTM exhibits substantially higher error magnitudes, as the quadratic error metric amplifies the performance difference and shows TSMixer's predictions are consistently closer to actual values. Similarly, the MAE distribution shows TSMixer maintaining a tight distribution around 0.02 while LSTM spreads across a wider range up to 0.25, reinforcing TSMixer's superior accuracy in terms of average prediction deviation. The forecast bias distribution reveals contrasting systematic tendencies between the models. LSTM shows a negative bias distribution centered around -2 to 0, indicating a tendency to underpredict, while TSMixer exhibits a positive bias distribution centered around +12 to +15, suggesting systematic overprediction. Both models show relatively tight bias distributions, indicating consistent directional tendencies.

The results demonstrate TSMixer's clear superiority in prediction accuracy across all error metrics, with errors approximately 5-10 times smaller than LSTM. However, the bias analysis reveals an important trade-off: while TSMixer achieves much lower absolute errors, it exhibits a more pronounced systematic bias toward overprediction compared to LSTM's underprediction tendency. This performance pattern aligns with research findings showing TSMixer's effectiveness in time series forecasting, where its all-MLP architecture with time-mixing and feature-mixing components outperforms traditional recurrent architectures like LSTM in terms of raw accuracy metrics, though potentially at the cost of calibrated predictions [4].

To better understand the comparative strengths of these models, we now examine their detailed forecasting accuracy, pattern recognition capabilities, and bias characteristics. Figure 16 and 17 provide a comparison between LSTM and TSMixer models for energy consumption prediction over a week-long period in February 2014. TSMixer demonstrates exceptional forecasting accuracy with remarkably low error metrics: MSE of 0.0008, RMSE of 0.0286, and MAE of 0.0202. The orange dashed line (TSMixer predictions) closely tracks the blue solid line (actual consumption) throughout the forecast period, capturing the general consumption patterns and specific fluctuations with high precision. LSTM shows significantly higher error rates with an MSE of 0.0107, RMSE of 0.1038, and MAE of 0.0525 - approximately 13 times higher than TSMixer's MSE. While the LSTM predictions generally follow the consumption trends, there are notable deviations, particularly during peak consumption periods where the model appears to lag behind actual values. Both models successfully capture the daily consumption cycles evident in the data, with regular patterns of low overnight consumption (around 0.0-0.1) and higher daytime usage (reaching 0.3-0.5). However, TSMixer exhibits the superior ability to predict the magnitude and timing of consumption spikes, particularly visible around February 3rd, 5th, and 8th when actual consumption reaches 0.4-0.7.

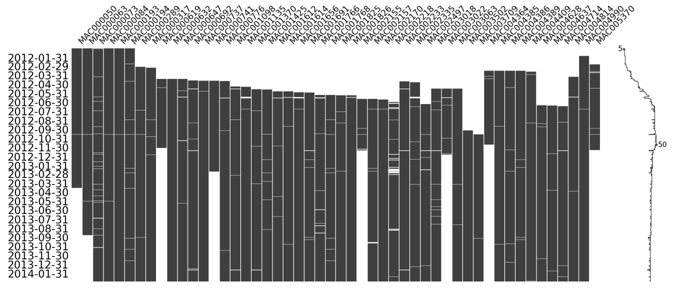
This research evaluates three distinct modeling paradigms for household energy consumption forecasting using London smart meter data: classical statistical methods, machine learning approaches, and deep learning architectures. Classical models, including AutoETS and ARIMA, demonstrated reasonable baseline performance with AutoETS achieving superior results (MSE: 0.0586, RMSE: 0.2422, MAE: 0.1191) compared to ARIMA, though both exhibited systematic underestimation bias. Machine learning models showed substantial improvements over classical approaches, with Lasso Regression leading the category (MSE: 0.0258, RMSE: 0.1607, MAE: 0.0775), followed closely by LightGBM and XGB Random Forest, achieving approximately 56% lower MSE compared to AutoETS by effectively leveraging multivariate features including weather data, temporal patterns, and lag variables. Deep learning models demonstrated the most impressive performance gains, with TSMixer emerging as the clear winner across all accuracy metrics (MSE: 0.0008, RMSE: 0.0286, MAE: 0.0171), representing approximately 13 times better performance than LSTM and dramatically outperforming all previous approaches. However, the superior accuracy came with a trade-off in forecast bias, as TSMixer exhibited the highest systematic overprediction tendency (bias: 12.8070) compared to LSTM's moderate bias (5.5132) and Lasso Regression's minimal bias (-1.3277). The progressive improvement from classical to machine learning to deep learning models validates the effectiveness of increasingly sophisticated architectures for capturing complex temporal dependencies in energy consumption data.

5 Conclusion

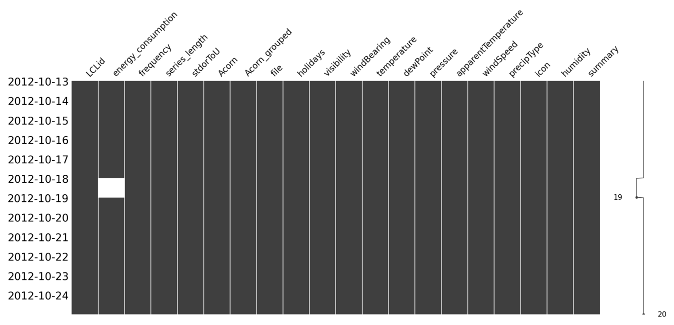
This project evaluated three forecasting paradigms—classical statistical methods, machine learning approaches, and deep learning architectures—for household energy consumption prediction using London smart meters data. The research compared AutoETS and ARIMA (classical), Lasso Regression, XGBoost Random Forest, and LightGBM (machine learning), and LSTM and TSMixer (deep learning) models. Results showed a clear performance hierarchy with increasing model complexity, where classical methods established baseline performance with AutoETS outperforming ARIMA. Machine learning models achieved substantial improvements, with Lasso Regression showing 56% lower MSE than AutoETS by leveraging multivariate features. Deep learning models demonstrated exceptional gains, with TSMixer achieving the best accuracy metrics (MSE: 0.0008) and outperforming LSTM by 13 times. However, TSMixer's superior accuracy came with higher systematic overprediction bias compared to other models. The research provides comprehensive benchmarks using real-world data and validates TSMixer's all-MLP architecture as effective for energy forecasting. The results suggest significant potential for deep learning approaches to enhance grid stability and sustainable energy management.

Future research should focus on addressing the systematic bias characteristics observed in high-performing models, exploring ensemble approaches that combine the accuracy of deep learning with the calibration properties of classical methods, and extending the analysis to larger-scale datasets encompassing diverse geographical and demographic contexts to validate the generalization of these findings across different energy consumption patterns and grid configurations.

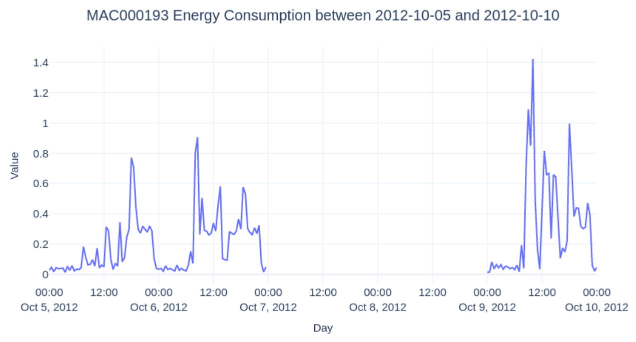
6 Appendix



(a) Visualization of the missing data.

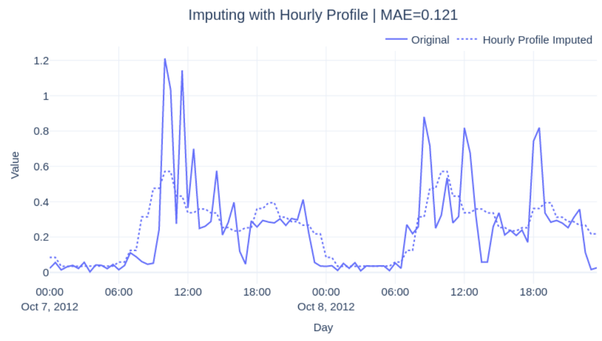


(b) Visualization of missing data of MAC000193 between 2012-09-30 and 2012-10-31.

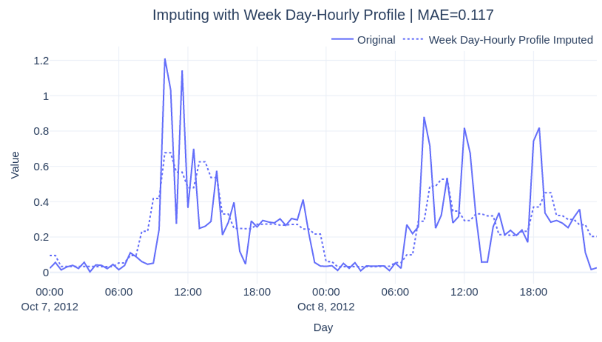


(c) Visualization of missing data of MAC000193 between 2012-10-07 and 2012-10-08.

Figure 1. Missing data visualization.



(a) Imputing with an hourly profile.



(b) Imputing the hourly average for each weekday

Figure 2. Imputation visualization.

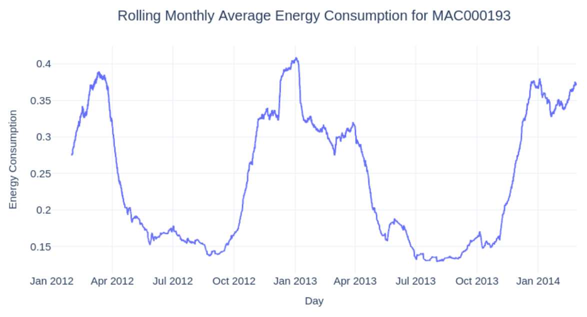


Figure 3. Rolling monthly average energy consumption of household MAC000193.

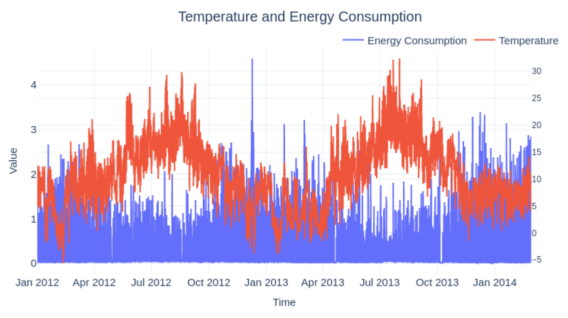


Figure 4. Temperature and energy consumption.

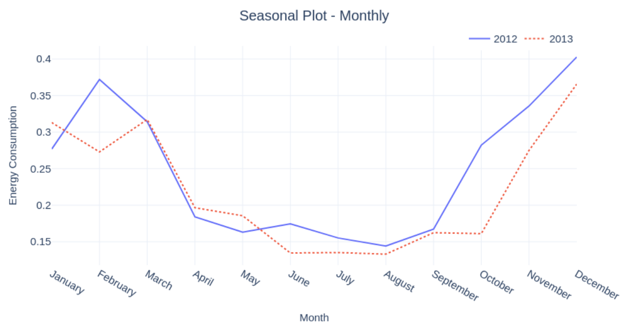


Figure 5. Seasonal plot at a monthly resolution.

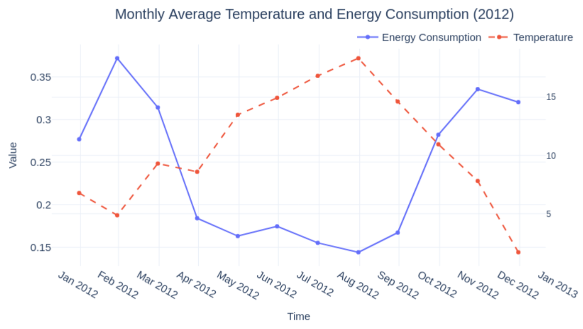


Figure 6. Energy consumption versus temperature (2012).

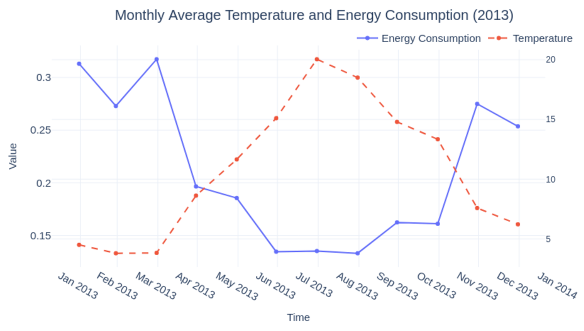


Figure 7. Energy consumption versus temperature (2013).

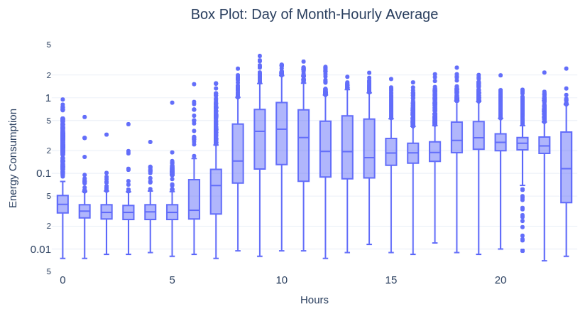


Figure 8. Seasonal box plot at an hourly resolution.

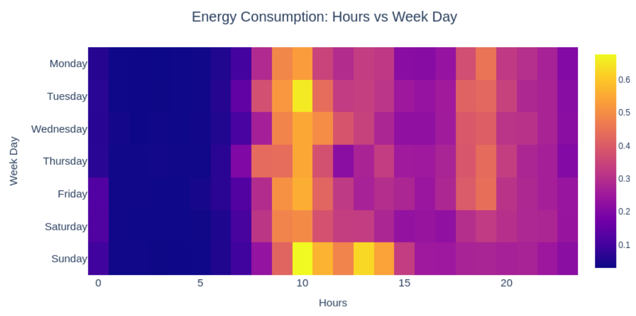


Figure 9. A calendar heatmap for energy consumption.

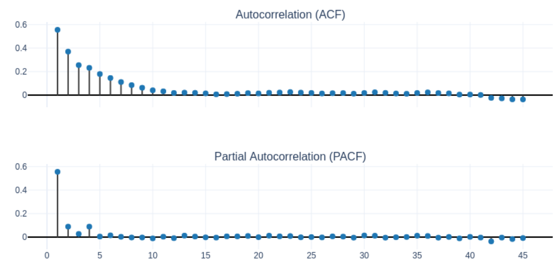
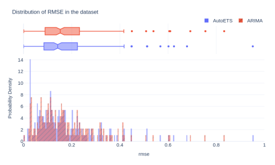
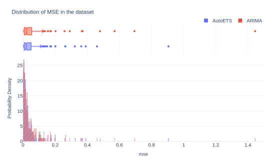


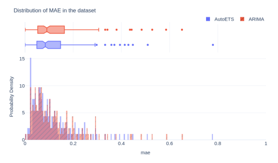
Figure 10. Autocorrelation and partial autocorrelation plots.



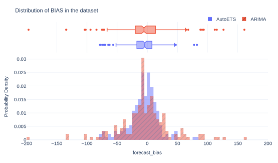
(a) RMSE distribution.



(b) MSE distribution.

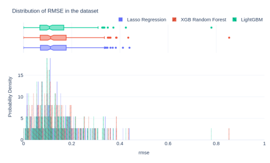


(c) MAE distribution.

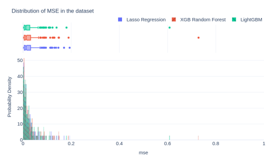


(d) Forecast Bias distribution.

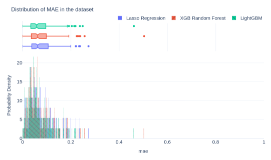
Figure 11. Metrics distribution of AutoETS compared to ARIMA.



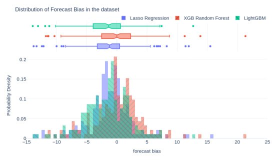
(a) RMSE distribution.



(b) MSE distribution.

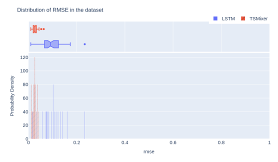


(c) MAE distribution.

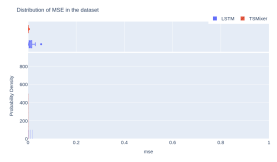


(d) Forecast Bias distribution.

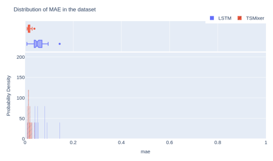
Figure 12. Metrics distribution of Lasso Regression, XGB Random Forest and LightGBM.



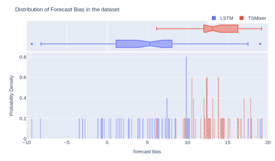
(a) RMSE distribution.



(b) MSE distribution.



(c) MAE distribution.



(d) Forecast Bias distribution.

Figure 15. Metrics distribution of LSTM and TSMixer.

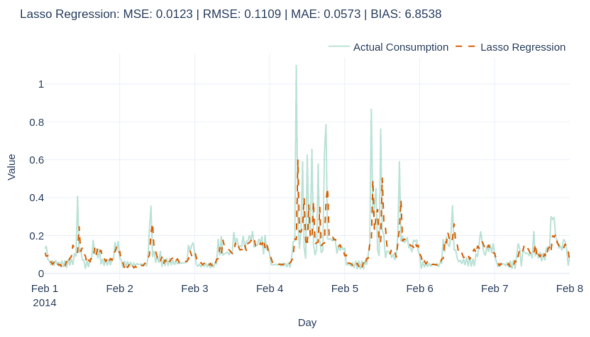


Figure 13. Lasso Regression forecast of MAC003069 (test set).

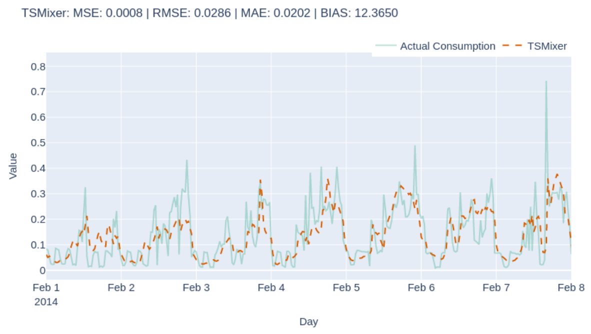


Figure 16. LSTM forecast of MAC000061 (test set).

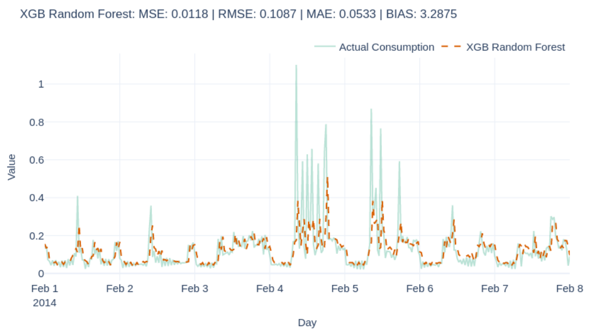


Figure 14. XGB-FR forecast of MAC003069 (test set).

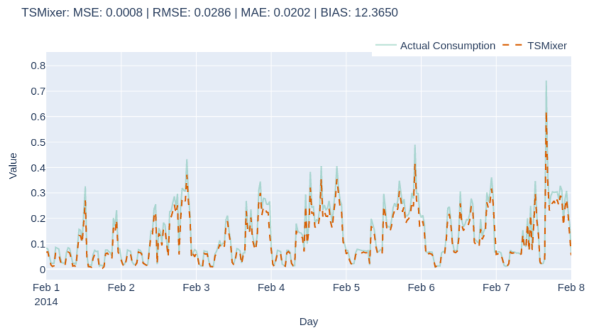


Figure 17. TSMixer forecast of MAC000061 (test set).

Algorithm 1 London Smart Meters Data Preparation Pipeline

```
Require: Source data directory containing HH block dataset (energy consumption data), household information, bank holidays, and weather data
Ensure: Processed and merged smart meter data saved as parquet files
 1: Import required libraries (pandas, pathlib, tqdm, numpy, etc.)
 2: Set source data path and block data path
 3: Initialize block_df_l ← empty list
 4: for each file $f$ in sorted block_data_path do
 5:     Read CSV file $f$ into dataframe block_df
 6:     Convert 'day' column to datetime
 7:     Filter records where day $\geq$ '2012-01-01'
 8:     Reshape dataframe to long form with hour blocks along rows
 9:     Create numerical hourblock column from 'hh_' prefix
10:     Process block into compact form and append to block_df_l
11: end for
12: Concatenate all energy consumption blocks in list block_df_l into hhblock_df
13: Load household information from CSV
14: Merge household information with hhblock_df
15: Load bank holidays data from CSV
16: Convert 'Bank holidays' to datetime and set as index
17: Resample bank holidays to 30-minute intervals
18: Load weather data from CSV
19: Convert 'time' to datetime and set as index
20: Resample weather data to 30-minute intervals with forward fill
21: Apply mapping of weather and holidays to hhblock_df
22: Save LCLid-Acorn mapping to pickle file
23: Split blocks into 8 chunks
24: for each chunk of blocks do
25:     Filter hhblock_df for current chunk
26:     Save chunk to parquet file
27: end for
```

Algorithm 2 Data Processing and Classical Forecasting Pipeline

```
Require: Smart meter data, household information
Ensure: Processed data and forecasting model evaluation
 1: Import required libraries and set random seeds
 2: Set project paths for data sources and outputs
 3: Load household mapping data with ACORN socio-demographic classifications
 4: Sample households from different ACORN groups (Affluent, Comfortable, Adversity)
 5: Load and Concatenate smart meter data blocks for selected households
 6: Convert data from compact to expanded form with time series and static features
 7: Split data into train (pre-2014), validation (Jan 2014), and test (Feb 2014) sets
 8: Save split datasets to parquet files
 9: Perform missing value imputation on all datasets
10: Save imputed datasets
11: Initialize metrics dataframe for model evaluation
12: Select representative household (MAC000193) for visualization
13: for each forecasting model in [ARIMA, AutoETS] do
14:     Train model on training data
15:     Generate forecasts for validation period
16:     Evaluate performance using MAE, MSE, RMSE, and forecast bias
17:     Visualize forecasts against actual values
18:     Save visualization to image file
19: end for
20: Scale up evaluation to multiple households (up to 500)
21: Train models on training data for all selected households
22: Generate forecasts for validation period
23: Calculate and Visualize error distributions across households
24: Save validation predictions and metrics
25: Create combined train+validation dataset
26: Train final models on combined dataset
27: Generate forecasts for test period
28: Evaluate final performance on test data
29: Save test predictions and metrics
```

Algorithm 3 Feature Engineering - Adding Temporal Features

```
1:  function AddTemporalFeatures(df, field_name, frequency, add_elapsed, prefix, drop, use_32_bit)
2:      $field \leftarrow df[field\_name]$
3:      $prefix$ ← (replace "Date" in $field\_name$ with empty string if $prefix$ is None) + "_"
4:      $attr$ ← timeFeaturesByFrequency($frequency$)
5:      $added_features$ ← empty list
6:      for all $n \in attr$ do
7:          if $n \neq$ "Week" then
8:              if $use\_32\_bit$ then
9:                  $df[prefix + n] \leftarrow field.dt.n.lower().astype("int32")$
10:             else
11:                 $df[prefix + n] \leftarrow field.dt.n.lower()$
12:             end if
13:             Append $prefix + n$ to $added\_features$
14:         end if
15:     end for
16:     if "Week" ∈ $attr$ then
17:         if field.dt has isocalendar attribute then
18:             $week \leftarrow field.dt.isocalendar().week$
19:         else
20:             $week \leftarrow field.dt.week$
21:         end if
22:         if $use\_32\_bit$ then
23:             Insert $week.astype("int32")$ at position 3 with name $prefix$ + "Week"
24:         else
25:             Insert $week$ at position 3 with name $prefix$ + "Week"
26:         end if
27:         Append $prefix$ + "Week" to $added\_features$
28:     end if
29:     if $add\_elapsed$ then
30:         $mask$ ← not $field.isna()$
31:         $df[prefix + "Elapsed"]$ ← where($mask$, $field.values.astype(np.int64)/10^9$, None)
32:         if $use\_32\_bit$ then
33:             if no null values in $df[prefix + "Elapsed"]$ then
34:                 $df[prefix + "Elapsed"] \leftarrow df[prefix + "Elapsed"].astype("int32")$
35:             else
36:                 $df[prefix + "Elapsed"] \leftarrow df[prefix + "Elapsed"].astype("float32")$
37:             end if
38:         end if
39:         Append $prefix$ + "Elapsed" to $added\_features$
40:     end if
41:     if $drop$ then
42:         Drop column $field\_name$ from $df$
43:     end if
44:     return $df, added\_features$
45: end function
```

Algorithm 4 Feature Engineering - Adding Lag Features

```
1:  function AddLags($df, lags, column, ts\_id, use\_32\_bit$)
2:      Verify $lags$ is list-like
3:      Verify $column$ exists in $df$
4:      $\_32\_bit_dtype$ ← appropriate 32-bit dtype for $df[column]$
5:      $added\_features$ ← empty list
6:      if $ts\_id$ is None then
7:          Warn "Assuming one unique time series"
8:          for all $l \in lags$ do
9:              if $use\_32\_bit$ and $\_32\_bit\_dtype$ exists then
10:                 $df[column\_lag\_l] \leftarrow df[column].shift(l).astype(\_32\_bit\_dtype)$
11:             else
12:                 $df[column\_lag\_l] \leftarrow df[column].shift(l)$
13:             end if
14:             Append $column\_lag\_l$ to $added\_features$
15:         end for
16:     else
17:         Verify $ts\_id$ exists in $df$
18:         for all $l \in lags$ do
19:             if $use\_32\_bit$ and $\_32\_bit\_dtype$ exists then
20:                 $df[column\_lag\_l] \leftarrow df.groupby([ts\_id])[column].shift(l).astype(\_32\_bit\_dtype)$
21:             else
22:                 $df[column\_lag\_l] \leftarrow df.groupby([ts\_id])[column].shift(l)$
23:             end if
24:             Append $column\_lag\_l$ to $added\_features$
25:         end for
26:     end if
27:     return $df, added\_features$
28: end function
```

Algorithm 5 Feature Engineering for Time Series Data

```
Require: DataFrame, time column, frequency, lag parameters
Ensure: Enhanced DataFrame with temporal and lag features
1: Load train, validation, and test dataframes
2: Combine dataframes with type labels and sort by LCLid and timestamp
3: Define lags as 1-5, 46-50, and 7-day lags (334-338)
4: $full\_df, features$ ← ApplyFeatureEngineering    ▷ Adding temporal features and adding lag features
5: Split and save enhanced dataframes by type (train, validation, test)
```

Algorithm 6 Feature Preprocessing and Missing Value Imputation for Machine Learning Models

```
Require: DataFrame, MissingValueConfig
Ensure: DataFrame with imputed missing values
 1: function ImputeMissingValues($df$, $missing\_config$)
 2:     $df \leftarrow$ copy of $df$
 3:     Apply backward fill, forward fill, and zero fill strategies per configuration
 4:     Fill remaining numeric columns with column means
 5:     Fill remaining categorical columns with "NA"
 6:     return $df$
 7: end function
 8: function PreprocessFeatures($X$, $model\_config$, $missing\_config$)
 9:     if $model\_config.fill\_missing$ then
10:         $X \leftarrow$ ImputeMissingValues($X$, $missing\_config$)
11:     end if
12:     if $model\_config.encode\_categorical$ then
13:         $X \leftarrow categorical\_encoder.fit\_transform(X)$
14:     end if
15:     if $model\_config.normalize$ then
16:         $X \leftarrow scaler.fit\_transform(X)$
17:     end if
18:     return $X$
19: end function
```

Algorithm 7 Machine Learning Pipeline

```
Require: Training data, test data, feature configuration, model configurations
Ensure: Predictions and performance metrics for all models
 1: Initialize project paths and load preprocessed datasets
 2: Combine training and validation datasets
 3: Load baseline metrics for comparison
 4: Configure FeatureConfig with continuous, categorical, and boolean features
 5: Configure MissingValueConfig with backward fill strategy for lag features
 6: Define model configurations: LassoCV, XGBRFRegressor, LGBMRegressor
 7: Initialize $all\_preds \leftarrow$ empty list
 8: Initialize $all\_metrics \leftarrow$ empty list
 9: for all $lcl\_id \in sorted(household\_ids)$ do
10:     for all $model\_config \in models\_to\_run$ do
11:         Clone $model\_config$ to avoid state interference
            ▷ Feature extraction and encoding
12:         $X\_train, y\_train \leftarrow$ feat_config.get_X_y(train_data, categorical=True, test=False)
13:         $X_{test}, y_{test} \leftarrow$ feat_config.get_X_y(test_data, categorical=True, test=True)
14:         $y\_pred, metrics, feature\_importance \leftarrow$ evaluate_model(model_config, X_train, y_train, X_test, y_test)    ▷ Model training and evaluation
15:         Create prediction dataframe with LCLid and Algorithm labels
16:         Add actual consumption values for comparison
17:         Append predictions to $all\_preds$
18:         Append metrics to $all\_metrics$
19:     end for
20: end for
21: Concatenate all predictions into $pred\_df$
22: Concatenate all metrics into $metrics\_df$
23: for all algorithm in [Lasso Regression, XGB Random Forest, LightGBM] do
24:     Filter predictions for current algorithm
25:     Calculate MSE, RMSE, MAE, and Forecast Bias
26:     Store overall metrics
27: end for
28: Combine baseline and ML metrics into summary dataframe
29: Create error distribution histograms for MAE, MSE, RMSE, and Forecast Bias
30: Generate feature importance plot for LightGBM
31: Create forecast plots for representative household
32: Save all visualizations and results
```

Algorithm 8 Feature Preprocessing with Hashing and Normalization for Deep Learning Models

```
Require: DataFrame $df$, feature type lists, hashing parameters, missing value strategies
Ensure: Preprocessed DataFrame with encoded and normalized features
 1: function PreprocessFeatures($df$, $continuous\_features$, $categorical\_features$, $boolean\_features$, $n\_hashed\_features$)
 2:     Apply backward fill, forward fill, and zero fill strategies per configuration
 3:     Fill remaining numeric columns with column means
 4:     Fill remaining categorical columns with "NA"
 5:     for all $col \in categorical\_features$ do
 6:         $hasher \leftarrow$ FeatureHasher($n\_hashed\_features$, input_type='string')
 7:         $hashed\_col \leftarrow hasher.transform([[str(val)]$ for val in $df[col]])$
 8:         Add $hashed\_col$ to $hashed\_features\_list$
 9:     end for
10:     $hashed\_df \leftarrow$ concatenate(drop categorical columns, stack hashed features)
11:     Convert boolean features to integers
12:     Normalize continuous features using StandardScaler
13:     return $hashed\_df$
14: end function
```

Algorithm 9 Deep Learning Pipeline for Energy Consumption Forecasting

```
Require: Training and test datasets, model configurations, household IDs
Ensure: Trained models with predictions and performance metrics
 1: Initialize project paths and load preprocessed datasets with temporal and lags features
 2: Combine training and validation datasets
 3: Define feature categories: continuous, categorical, boolean, and target variables
                                                        ▷ Data Preprocessing
 4: Configure missing value imputation strategies (backward fill for lag features)
 5: Set hashing parameters: $n\_hashed\_features = 32$
 6: for all dataset in [train, test] do
 7:     Convert timestamps and set multi-index [timestamp, LCLid]
 8:     Apply frequency resampling to 30-minute intervals
 9:     $processed\_data \leftarrow$ preprocess_features(dataset, feature_types, hashing_params)
10: end for
                                                        ▷ Model Configuration
11: Define model configurations:
12:   LSTM: input_length=96, output_length=48, hidden_dim=128, layers=3
13:   TSMixer: input_length=96, output_length=48, hidden_size=64, blocks=2
                                                        ▷ Time Series Preparation
14: Initialize target_scaler and covariates_scaler
15: Initialize empty lists for train/test series and covariates
16: for all household_id in household_ids[:50] do
17:     Extract household data from processed train and test sets
18:     Create overlap period: test_start - 2 days for continuity
19:     Convert to Darts TimeSeries format with 30-minute frequency
20:     Apply scaling: target_scaler.fit_transform() and covariates_scaler.fit_transform()
21:     Add scaled series to respective lists
22: end for
                                                        ▷ Model Training and Evaluation
23: for all model_config in [LSTM, TSMixer] do
24:     $predictions, metrics \leftarrow$ train_and_evaluate_model(
25:         model_config, train_series, train_covariates,
26:         test_series, test_covariates, household_ids, target_scaler)
27:     Save predictions and metrics to output files
28: end for
29: Combine all predictions and metrics from different models
30: Generate comparative performance analysis
```

References

[1] Manu Joseph and Jeffrey Tackes. *Modern Time Series Forecasting with Python*. 2nd ed. Packt Publishing, Oct. 2024.

[2] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. 3rd. Chapter 9.10: ARIMA vs ETS. OTexts, 2021. Chap. 9.10. URL: `https://otexts.com/fpp3/`.

[3] Jiu Oh and Byeongchan Seong. "Forecasting with a combined model of ETS and ARIMA". In: *Communications for Statistical Applications and Methods* 31.1 (Jan. 2024), pp. 143–154. ISSN: 2287-7843. DOI: `10.29220/CSAM.2024.31.1.143`. URL: `http://www.csam.or.kr/journal/view.html?doi=10.29220/CSAM.2024.31.1.143`.

[4] Si-An Chen et al. "TSMixer: An All-MLP Architecture for Time Series Forecasting". In: *Transactions on Machine Learning Research* (Sept. 2023). arXiv:2303.06053. URL: `https://arxiv.org/abs/2303.06053`.