Forecasting Project Process Documentation V2.0

March 10, 2025

Zhiqi Ma, Ruoheng Du, Xiaojiang Wu, Yijian Liu

Contents

Introduction

We are working on an electricity usage forecasting project. The dataset consists of historical electricity consumption data with a strong seasonal component. We are forecasting weekly overall electricity consumption, aiming to capture seasonal patterns and structural shifts to improve predictive accuracy. Our initial modeling approach includes SARIMA, with plans to explore additional methods such as machine learning-based forecasting models. The programming language used for this analysis is Python, leveraging libraries such as *statsmodels* and *pmdarima* for time series modeling.

The purpose of this document is to provide a detailed technical overview of:

1. Design specifications and parameters
2. Data inclusions and exclusions
3. Predictive variable creation process
4. Target variable definition
5. Iterative model building process
6. Metric evaluation process

Data Extraction

Workflow

This section explains the process of extracting, processing, and storing the electricity usage data. The dataset originates from the *ElectricityLoadDiagrams20112014* dataset, which contains electricity consumption records for 370 clients over multiple years. The data is provided with semicolon delimiters and includes timestamps and electricity consumption values in kW per 15-minute intervals.

Processing Steps

- The dataset is first extracted from a .zip file, yielding a structured text file.
- Decimal values are encoded as 0,00, requiring conversion to standard numerical formats.
- Consumption values are originally in kW per 15 minutes, and for conversion to kWh, they are divided by 4.
- Missing values do not exist within the dataset, but accounts created after the recorded timeframe are encoded as zeros.

This workflow ensures that the extracted data is cleaned, correctly formatted, and ready for further analysis in the forecasting pipeline.

Data Processing

Data Overview

This section details the data extraction process, necessary transformations, and diagnostic checks performed before modeling. The dataset consists of electricity consumption records for 370 clients, recorded at 15-minute intervals from 2011 to 2014.

Data Extraction Process

The dataset was loaded and preprocessed using Python (pandas) with the following steps:

- Data was imported from a semicolon delimited text file using *pd.read_csv()*.
- Decimal values were encoded using a comma *(0,00)* and were converted correctly using the *decimal=','* parameter.
- The first column, containing timestamps, was parsed as a datetime index using *parse_dates=[0]*, *index_col=0*, ensuring proper time alignment for resampling.
- Low memory mode was disabled (*low_memory=False*) to efficiently handle large data processing.

Data Diagnostics

A series of quality checks were performed to ensure data integrity:

- Total number of records verification
- Detection of duplicate entries
- Identification of missing values (none were found in the dataset)
- Validation of numerical field sums and consistency
- Confirmation of time period coverage and time difference between records

Modeling Data Creation

To prepare the dataset for forecasting, additional preprocessing steps were applied:

- Resampling: Data was aggregated from 15-minute intervals to weekly usage for efficiency in long-term trend analysis.
- Format Restructuring: The dataset was transformed into a long-table format to facilitate time series modeling.
- Overall Electricity Usage: The total weekly consumption across all clients was computed to analyze macro consumption trends.

Besides, daylight saving time does not require adjustment as the dataset maintains a consistent 15-minute interval, ensuring uniform time indexing. These preprocessing steps ensured that the dataset was structured, cleaned, and optimized for time series forecasting.

Target Variables

In this project, the target variable represents the weekly electricity usage to be forecasted based on historical consumption patterns. The objective is to develop a predictive model that accurately estimates future electricity demand, considering both seasonal trends and overall consumption patterns.

The target variable is defined as:

- Usage (kWh per week) – Aggregated from individual 15-minute interval data across all clients.

By predicting weekly electricity consumption, this model can help in energy demand planning, load balancing, and optimizing power distribution strategies.

Predictive Variables

Predictive variables are the features used to forecast weekly electricity usage in this project. These variables were selected based on their ability to capture consumption trends and seasonality, ensuring a robust forecasting model.

We categorize the predictive variables into two types:

1. Direct Variables – Extracted directly from the dataset, such as historical electricity consumption.
2. Derived Variables – Created through aggregations of the direct variables to enhance predictive performance, such as the weekly overall electricity consumption.

Variable List

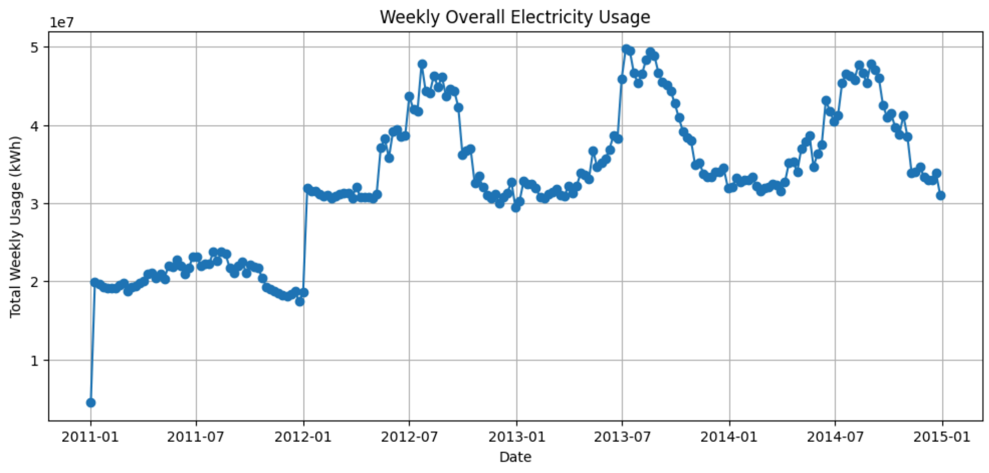
1. Weekly Electricity Consumption: The total electricity usage aggregated from 15-minute interval data to a weekly level. This serves as the primary predictor, capturing past consumption trends and seasonality.
2. Daily Electricity Consumption: The total electricity usage aggregated from 15-minute interval data to a daily level. This also serves as the primary predictor.
3. Time Features: Year, month, and week indicators help capture seasonal variations and long-term trends in electricity demand.

These predictive variables allow the model to effectively learn from historical consumption patterns and anticipate future electricity usage.

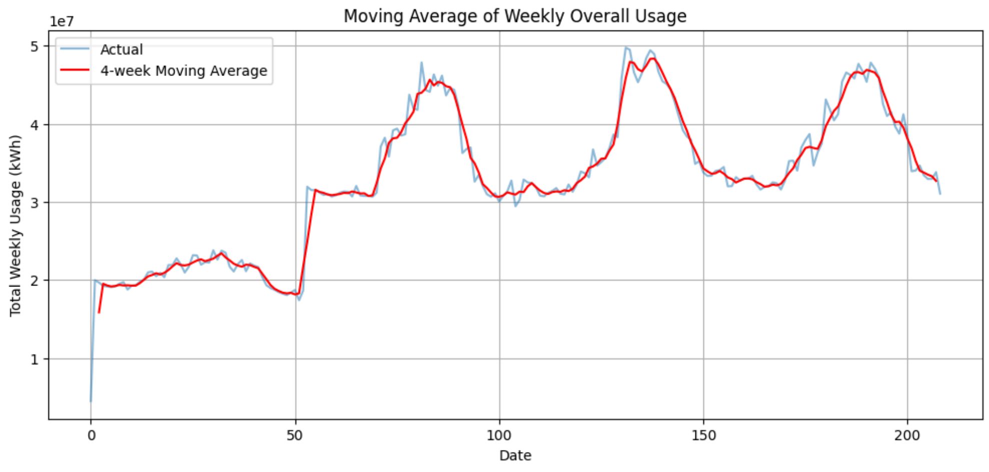
Pre-modeling

The dataset spans from 2011 to 2015, capturing weekly electricity usage trends. The forecasting model is trained on historical consumption data and aims to predict future usage patterns. To better understand the characteristics of electricity consumption, we conducted exploratory data analysis (EDA), including trend analysis, periodicity detection, and anomaly detection. Below are key insights derived from the analysis:

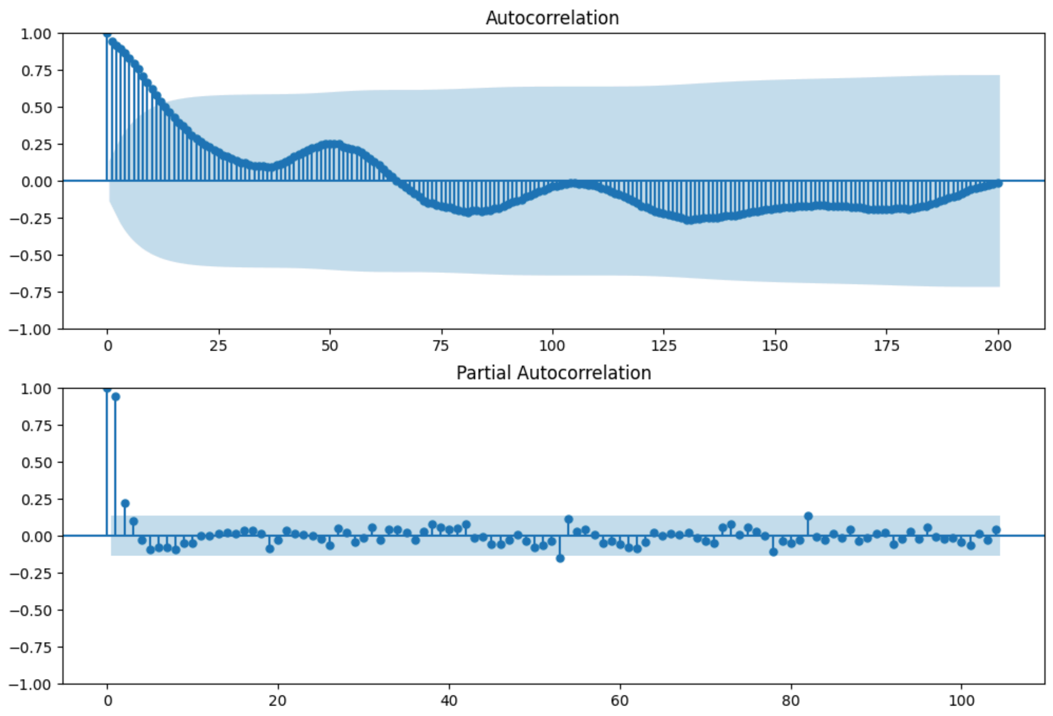
- Long-Term Consumption Trends
    - *Image 1: Weekly Overall Electricity Usage*



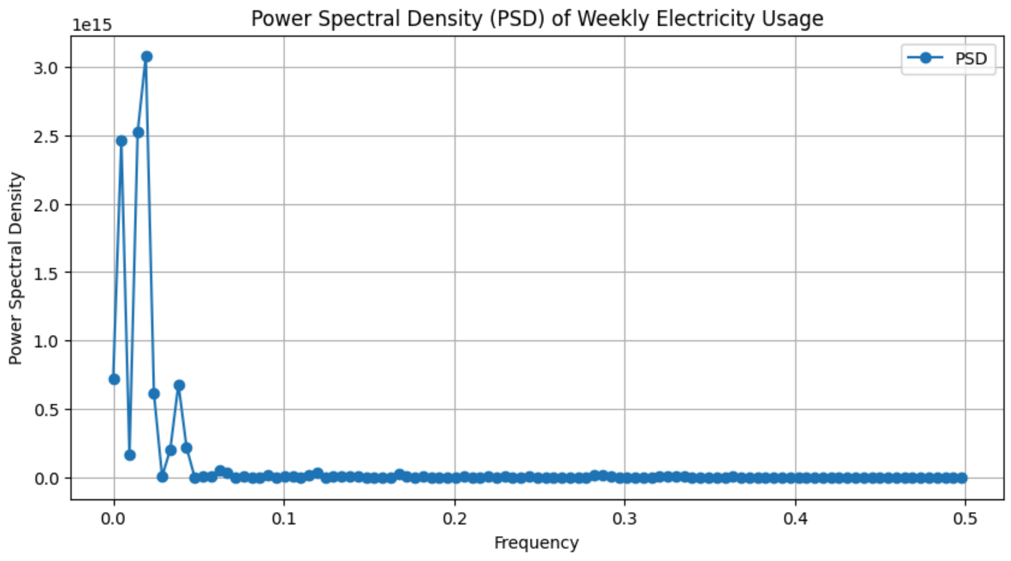
        - The time series plot of weekly electricity usage reveals seasonal fluctuations and long-term trends.
    - *Image 2: Moving Average of Weekly Usage*



        - A 4-week moving average was applied to smooth short-term fluctuations and highlight seasonal patterns.
- Periodicity Detection
    - *Image 3: Autocorrelation and Partial Autocorrelation (ACF/PACF)*

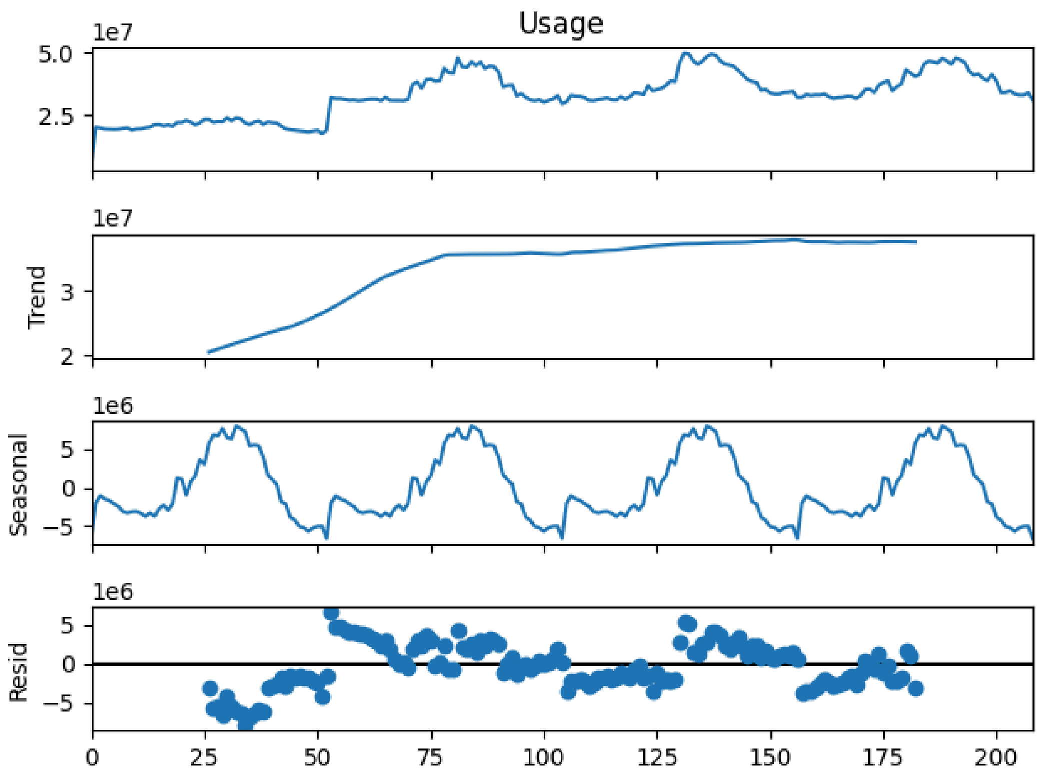


        - The autocorrelation function (ACF) indicates strong periodicity, confirming seasonal dependencies in electricity usage.
        - The partial autocorrelation function (PACF) suggests that recent past values influence future consumption, but correlations weaken over longer lags.
    - *Image 4: Power Spectral Density (PSD)*



- High spectral density at low frequencies further supports the presence of annual seasonality in electricity consumption.

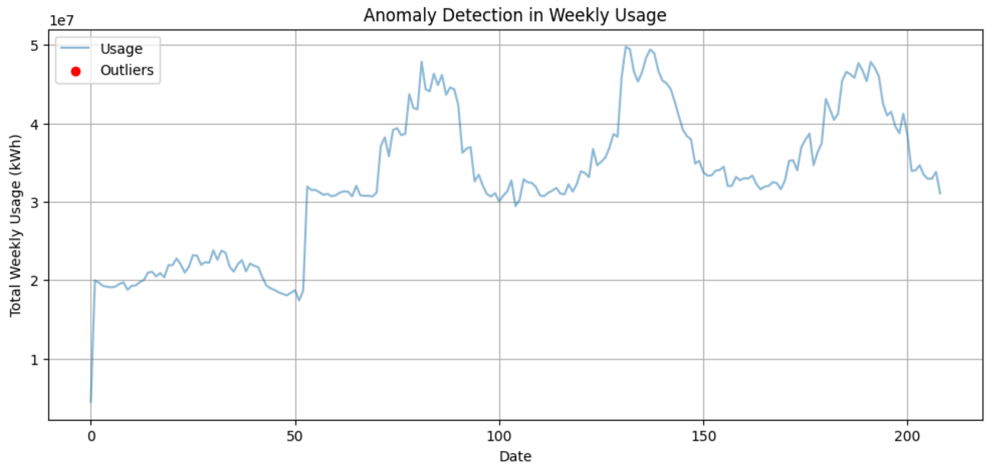
- Time Series Decomposition
  - *Image 5: Trend, Seasonal, and Residual Components*



- The decomposition confirms a gradual upward trend, a strong seasonal pattern, and residual variations capturing short-term fluctuations.

- Anomaly Detection

- *Image 6: Outlier Detection in Weekly Usage*



    - No significant anomalies were detected, indicating stable data quality for modeling.

Training & Testing Split

1. For SARIMA model and Prophet Model:

To ensure robust model performance, the dataset was split into training, validation, and test sets as follows:

- Training Set (60%): Used for model learning.
- Validation Set (20%): Used for hyperparameter tuning and model selection.
- Test Set (20%): Used for final model evaluation.

This approach ensures that the model generalizes well by testing it on unseen future data while maintaining a balanced ratio across training and evaluation phases.

2. For Chronos model:

Chronos performs zero-shot forecasting, i.e., predict the future points without retraining the history dataset, and even for fine-tuning the model, we don't need to (there is 'eval_during_fine_tune' feature in the model during fine-tuning but we cannot manually set the validation size) define validation dataset. Thus, only train-test split allowed here, and we did:

- Training Set (80%): Used for model learning.
- Test Set (20%): Used for final model evaluation.

Modeling

Introduction

After defining the target and predictive variables, we applied statistical and machine learning techniques to forecast electricity usage. The model selection process involved optimizing hyperparameters and evaluating different configurations based on forecast accuracy.

Model Design

Model 1: SARIMA

Given the seasonal nature of the dataset, we employed SARIMA, an extension of ARIMA that incorporates seasonality, to capture periodic patterns effectively. SARIMA (p, d, q) × (P, D, Q, s) extends ARIMA by integrating seasonal components, making it suitable for time series data with periodic fluctuations. The parameters include:

Parameter	**Description**
p	Number of autoregressive terms (past observations used for regression)
d	Number of differencing steps to make the series stationary
q	Number of past error terms used for smoothing (moving average)
P	Seasonal autoregressive terms
D	Seasonal differencing steps
Q	Seasonal moving average terms

Parameter	Description
s	Seasonal cycle length (weekly periodicity: s=52)

Model 2: Amazon's Chronos

Chronos is a family of pretrained time series forecasting (foundation) models based on T5 architecture. The only difference is in the vocabulary size: Chronos-T5 models use 4096 different tokens, compared to 32128 of the original T5 models, resulting in fewer parameters. Chronos models have been trained on 84B publicly available time series data, as well as synthetic data generated using Gaussian processes.

During the pre-training process: A time series is transformed into a sequence of tokens via scaling and quantization, and a language model is trained on these tokens using the cross-entropy loss, very similar to how people train a non-time-series LLM model. It uses TSMixup and KernelSynth to augment data quality.

Once trained: the model predicts time series using zero-shot probabilistic forecasts by sampling multiple future trajectories given the historical context.

For details on Chronos models, please refer to the paper. In this project we mainly explore **Chronos-Bolt** – the most recent version of Chronos model.

To use Chronos model, **AutoGluon-TimeSeries (AG-TS)** is the best choice, which provides a robust and easy way to use Chronos through the familiar TimeSeriesPredictor API. Roughly speaking, we can:

- Use Chronos in **zero-shot** mode to make forecasts without any dataset-specific training
- **Fine-tune** Chronos models on custom data to improve the accuracy
- **Handle covariates & static features** by combining Chronos with a tabular regression model

Model Usage (zero-shot forecasting) Pipeline (tutorial given in AG-TS):

1. first transform data into specified format – dataset contains 3 columns:
    a. item_id – id for time series (we can concat multiple time series as input and model will make forecasts for each separately);
    b. timestamp - i.e. the time in the time series
    c. target – time series data value
2. define prediction length and do train/test split
3. construct a predictor: set `prediction_length`, `train_data,` and the model version about to use, e.g. `presets = 'bolt_small'`
4. fit the predictor: the fit call serves as a proxy for the TimeSeriesPredictor to do some of its chores under the hood, such as inferring the frequency of time series and saving the predictor's state to disk, it does not train the data
5. use predict to generate forecasts, and use plot to visualize them.
6. use evaluate to get test statistics, or use leaderboard to get all details for test results.

That is, we only need to set the train_data, prediction length, and model preset we want to use.

Model Fine-tuning Pipeline (also included in tutorial in AG-TS):

1. steps before constructing predictor are the same as above
2. construct a predictor: besides prediction_length and train_data, also define hyperparameters = {"Chronos":[{...}...{...}]}, hyperparameters inside {...} may include (details in Model Zoo on AG-TS):
    a. model_path – Model path used for the model: e.g. 'bolt_small'
    b. fine_tune (*bool, default = False*) – If True, the pretrained model will be fine-tuned
    c. fine_tune_lr (*float, default = 1e-5*) – learning rate used for fine-tuning. This default is suitable for Chronos-Bolt models
    d. fine_tune_steps (*int, default = 1000*) – number of gradient update steps to fine-tune for
    e. fine_tune_batch_size (*int, default = 32*) – batch size to use for fine-tuning
    f. eval_during_fine_tune (*bool, default = False*) – If True, validation will be performed during fine-tuning to select the best checkpoint. Setting it to True may result in slower fine-tuning
    g. …… more details on AG-TS
    h. define multiple model settings inside {...}, and during evaluation, calling the leaderboard will output results for all of the models
    i. define time_limit to control the fine-tuning time; chronos bolt models can all be fine-tuned via CPU, and faster via GPU
    j. To define a hyperparameter space that contains multiple parameter choices (details in Search Space on AG-TS), follow this rule:

```Python
from autogluon.common import space

categorical_space = space.Categorical('a', 'b', 'c')  # Nested search space for hyperparameters which are categorical.
real_space = space.Real(0.01, 0.1)  # Search space for numeric hyperparameter that takes continuous values
int_space = space.Int(0, 100)  # Search space for numeric hyperparameter that takes integer values
bool_space = space.Bool()  # Search space for hyperparameter that is either True or False.
```

3. steps after defining the predictor are the same as zero-shot forecasting.

Refer to project code for more detailed illustration.

Model Covariate Analysis (also included in tutorial in AG-TS):

Idea: Chronos is a univariate model, meaning it relies solely on the historical data of the target time series for making predictions. AG-TS now features covariate regressors that can be combined with univariate models like Chronos-Bolt to incorporate exogenous information. A covariate_regressor in AG-TS is a tabular regression model that fits on the known covariates and static features to predict the target column at each time step.

We haven't tried this feature in our project. This can be tested by first spending some time looking for suitable covariate data to work together with electricity data.

Model 3: Facebook Prophet

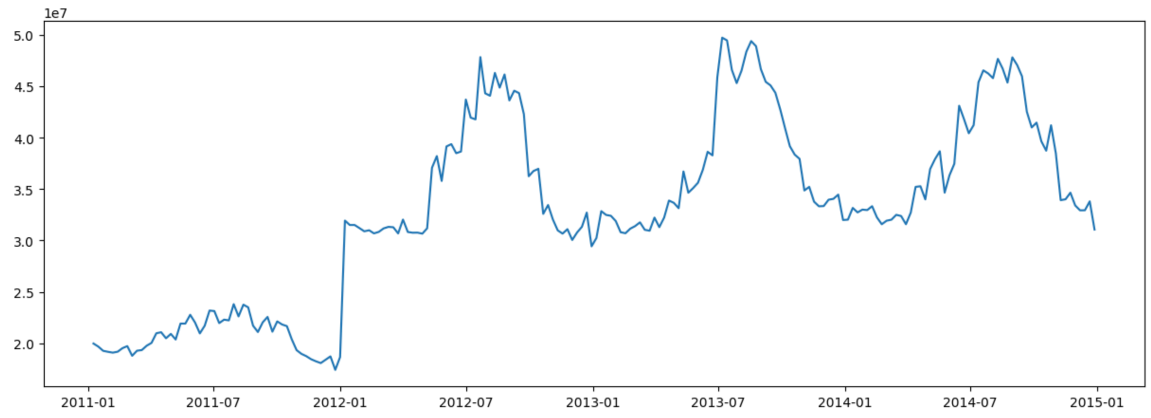
Prophet is a time series data forecasting process. Its foundation is an additive model that takes into account seasonality on a daily, weekly, or annual basis, trends, and holiday effects. It is robust to missing data and trend shifts, but it can perform best with time series that have strong seasonalities and trends.

Prophet only accepts the input dataframe with two columns: **ds** (datestamp) and **y** (value). The ds column is of a format expected by Pandas, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp. The y column has to be numeric and the value we aim to learn and predict.

In Prophet, we mainly model following components:

1. **Seasonality**: **Daily**, **Weekly**, **Monthly**,**Yearly**. Even though when we construct the Prophet object, the Prophet model will defaultly enable weekly and yearly seasonality but we can also customize our own seasonality by setting "period" and "fourier_order". These components represent the repeated modes with different periods. Daily seasonality reveals the mode occurring daily, weekly seasonality is set with period = 7, monthly seasonality is set with period = 30.5 and yearly seasonality is set with period = 365.25. In this project, we use the default **yearly seasonality** and customized **monthly seasonality** and **weekly seasonality**. Seasonalities are estimated using a partial Fourier sum. By tuning "fourier_order" for the seasonality, we can adjust the flexibility of seasonality. Increasing the number of Fourier terms allows the seasonality to fit faster changing cycles, but can also lead to overfitting.
2. **Trend**: The trend in Prophet can be piecewise so Prophet provides some parameters like **"changepoint_prior_scale"** to change the flexibility of trend. In this model, we focus on tuning **"changepoint_prior_scale"** to avoid overfitting or underfitting.
3. **Holiday**: Prophet provides a very powerful function. If there are holidays or other recurring events that are about to be modeled, a dataframe would be created for them by creating four columns (**holiday, ds, lower_window, upper_window**). If the holidays occur repeatedly, this indicates to include all occurrences of the holiday, both in the past and future. If one holiday occurs only once, just include that date time for that holiday in **ds**. In this project, we observe there is a huge jump around 2012-01 and this could possibly be explained by the creation of new user accounts. This could not be expected in the future and there is no meaningful mode baked in this jump so we decide to make it a **"holiday"** . In this way, the Prophet won't be misled by this unusual phenomenon.

Prophet Model Forecasting Growth :



- **Growth = “Linear”**: Prophet defaultly uses a linear model.
- **Growth = “Logistic”** : Instead Prophet can also utilize a logistic growth trend model with a specific carrying capacity. We must specify the carrying capacity in a column **cap** and a column **floor**. Here we will assume a particular value, but this would usually be set using data or expertise about the electricity market size. In this project, we assume the “cap” is equal to 1.1*max(**y** in train) and “floor” is equal to 0.9*min(**y** in train), which is reasonable because the electricity load won’t change a lot and is constrained by other factors.

Comparisons of Models

Model	**Pros**	**Cons**
SARIMA	- Well-suited for capturing seasonal and trend components in time series data - Provides interpretable parameters (AR, MA, differencing terms)	- Assumes linear relationships, limiting its ability to capture complex dependencies - Sensitive to structural shifts in the data, requiring manual adjustments
Chronos	- Leverages foundation models / deep learning techniques for time series forecasting - Can handle complex temporal dependencies and nonlinear relationships - Can perform zero-shot forecasting, which does not need to train the data and is fast	- Requires a large amount of data for effective training - Fine-tuning is computationally expensive and harder to interpret compared to traditional models
Prophet	- Automatically detects seasonal trends and holiday effects - Enable customized seasonal trends and holiday effects - Flexible and robust against missing data and outliers	- Less effective for highly non-stationary time series - Can struggle with abrupt structural shifts if not properly configured

Evaluation Metric

To assess model accuracy, we used Mean Absolute Percentage Error (**MAPE**), a commonly used metric in forecasting:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{Y_t - \hat{Y}_t}{Y_t}\right| \times 100\%$$

- Y_t = actual value at time t
- $\hat{Y}_t$ = predicted value at time t
- n = total number of observations

MAPE provides an intuitive measure of forecasting error as a percentage, making it easier to compare across different models and datasets.

Deliverables

Remind that our target variable is: Weekly Overall Electricity Consumption (kWh per week) – Aggregated from individual 15-minute interval data across all clients.

Model 1 Results: SARIMA

This section presents the results of the SARIMA model applied to electricity consumption forecasting. The analysis includes model selection, performance evaluation, and comparative results for different SARIMA configurations.

1. Model Selection and Training Setup

To optimize the performance of the SARIMA model, different parameter selection methods were used:

- EDA-based parameter tuning
  - Parameters manually selected based on exploratory data analysis (SARIMA (1,1,1) × (1,1,1,52)).
- Auto-ARIMA optimization
  - Parameters automatically optimized using AIC/BIC minimization (SARIMA (0,1,2) × (1,0,0,52)).
- Grid Search-based tuning
  - Parameters derived from stationarity testing and differencing selection (SARIMA (0,1,0) × (1,1,1,52)).

2. Overall Model Performance

The table below summarizes the test performance across different SARIMA configurations:

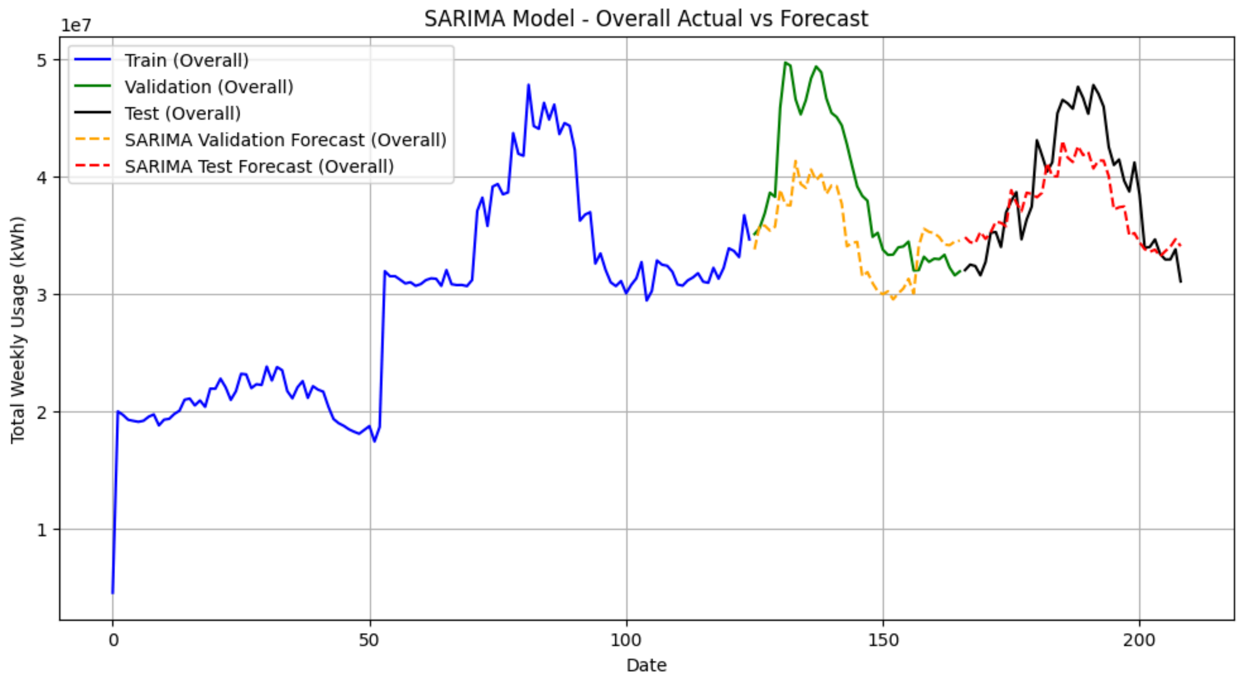
SARIMA Model (s=52)	Overall Test MAPE	Test Period I MAPE	Test Period II MAPE	Test Period III MAPE
(1,1,1) × (1,1,1)	6.68%	5.50%	8.76%	5.85%
(0,1,2) × (1,0,0)	14.29%	7.98%	23.89%	11.44%
(0,1,0) × (1,1,1)	7.29%	4.57%	10.71%	6.76%

- Best-performing model: SARIMA (1,1,1) × (1,1,1,52) achieved the lowest overall MAPE of 6.68%, indicating superior performance in capturing consumption patterns.

- Worst-performing model: SARIMA (0,1,2) × (1,0,0,52) showed the highest error (MAPE = 14.29%), suggesting that automatic model selection did not yield optimal results.

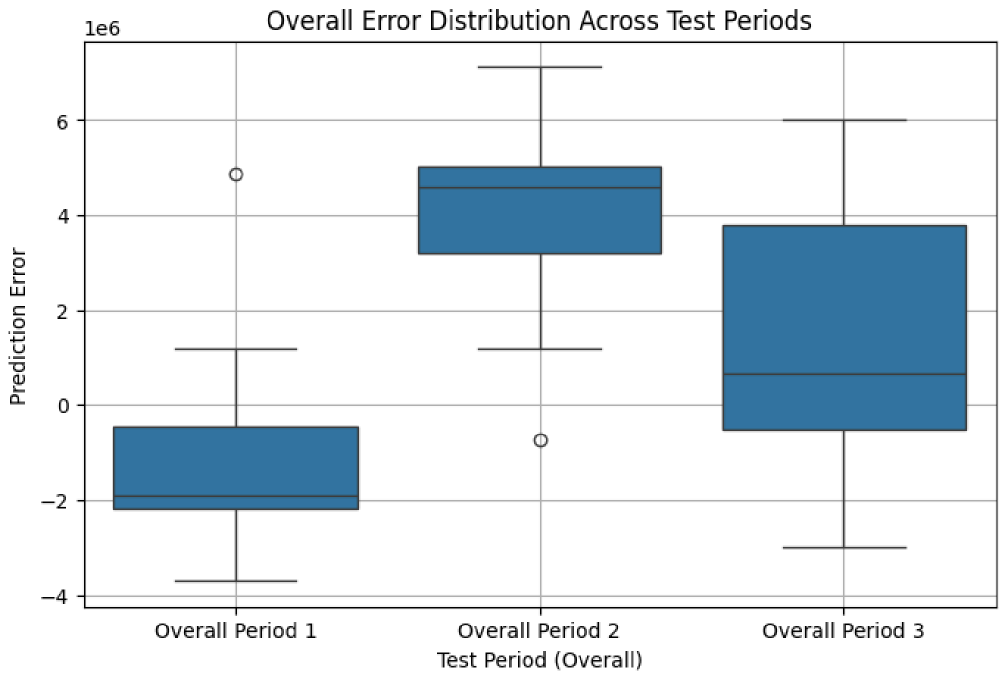
3. Visualization of Best Model Performance: SARIMA (1,1,1) × (1,1,1,52)

3.1 Actual vs. Forecasted Usage



- The SARIMA (1,1,1) × (1,1,1,52) model effectively follows the seasonal consumption pattern.
- Validation and test set predictions relatively closely align with actual electricity usage trends.

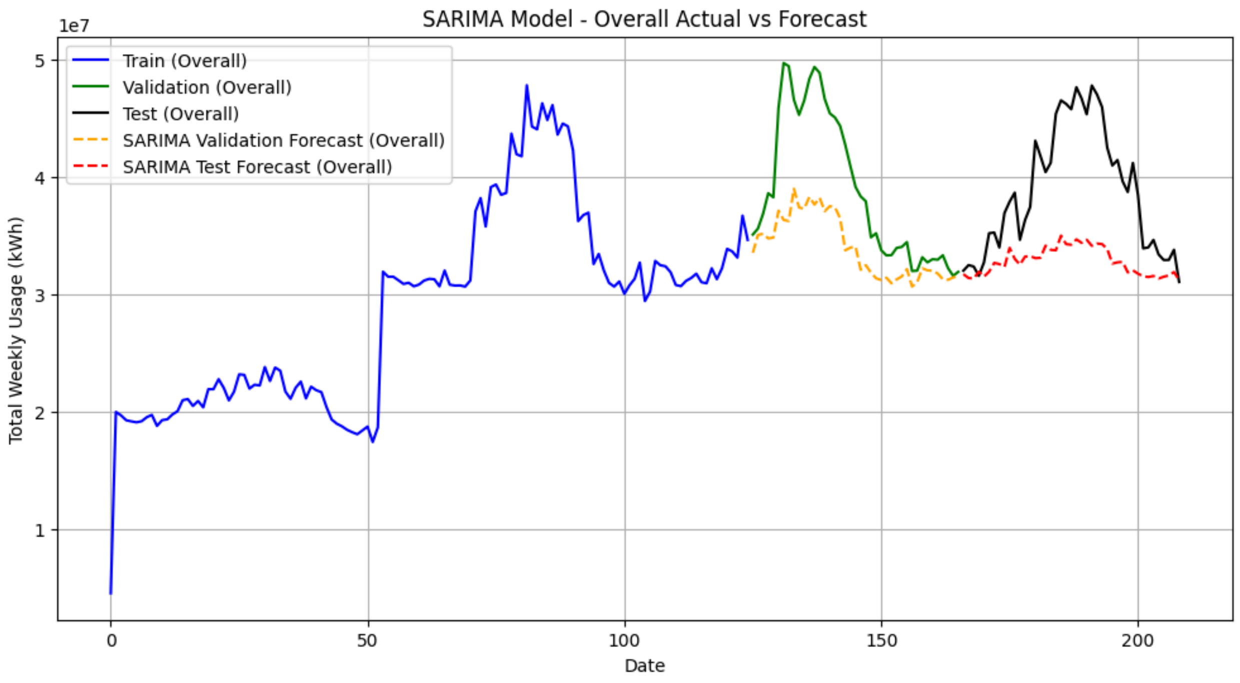
3.2 Error Distribution Across Test Periods



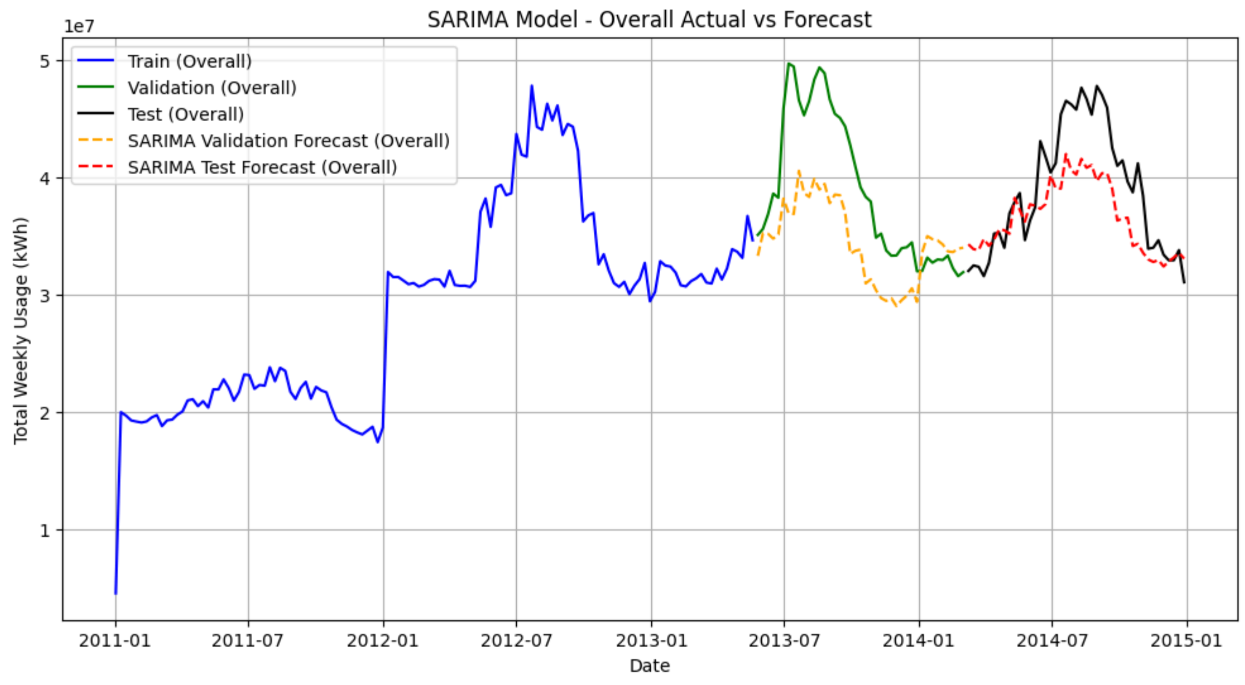
- The best model exhibits a relatively stable error distribution across test periods.

4. Visualization of Other Model Performance

4.1 Actual vs. Forecasted Usage

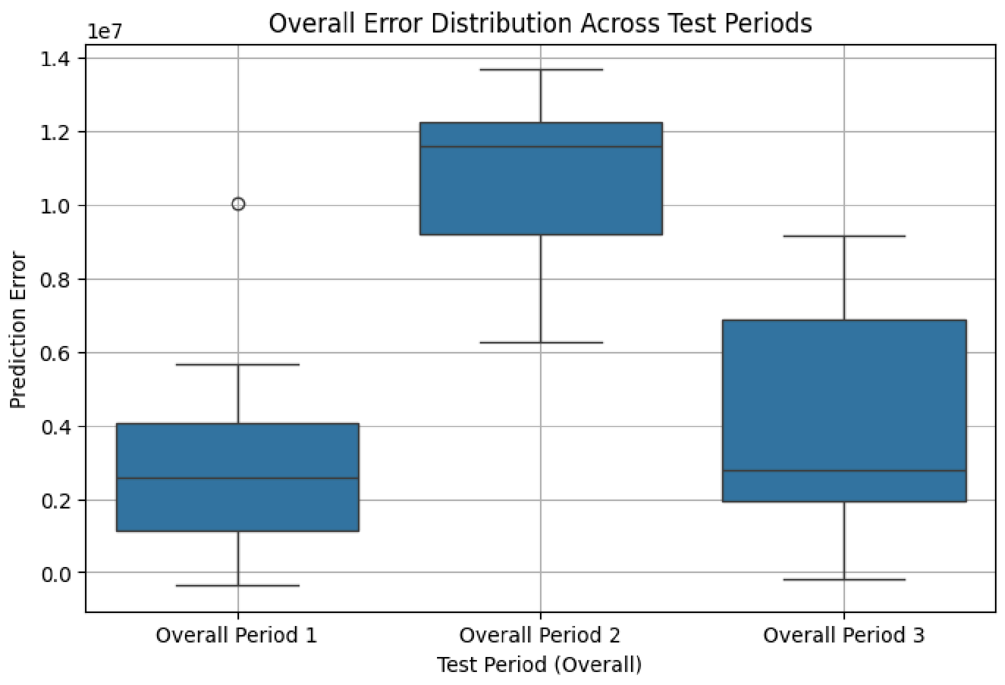


- SARIMA (0,1,2) × (1,0,0,52) model result.

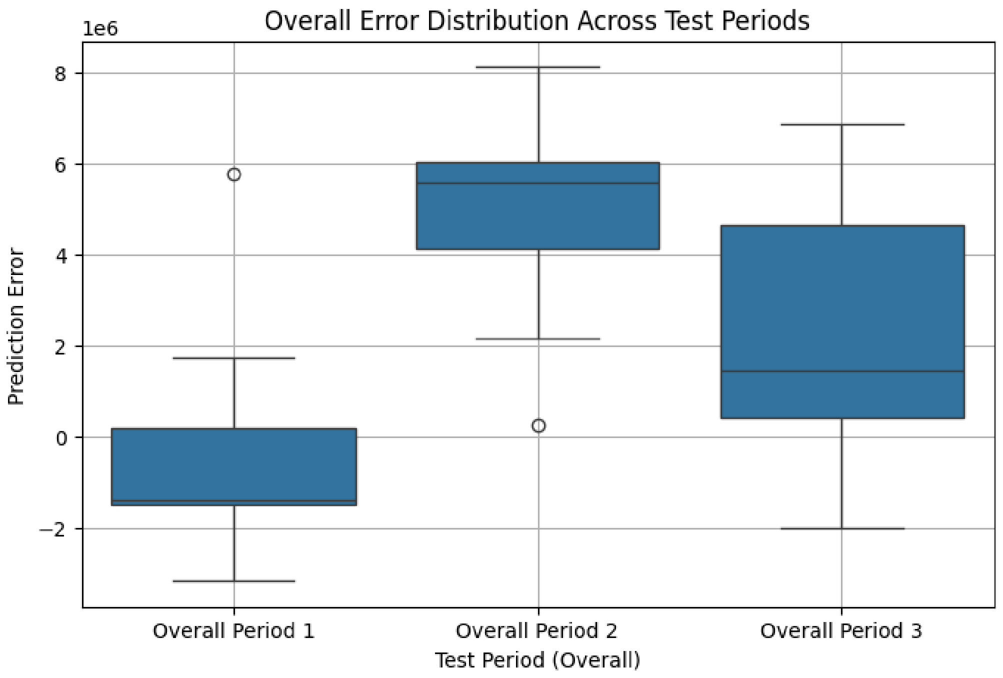


- SARIMA (0,1,0) × (1,1,1,52) model result.

4.2 Error Distribution Across Test Periods



- SARIMA (0,1,2) × (1,0,0,52) model result.



- SARIMA (0,1,0) × (1,1,1,52) model result.

Model 2 Results: Chronos

This section presents the results of the Chronos model applied to electricity consumption forecasting. The analysis includes model selection, performance evaluation, and comparative results for different Chronos configurations.

1. Model Setup and Selection

- **Zero-shot forecasting**:
  a. use Chronos **bolt_small** preset model: as explained in Model Design section above, only prediction_length, train_data, and model preset are needed, no more parameters;
  b. use Chronos **bolt_base** preset model: for the same reason, no more explanation needed.
- **Fine-tuning**:
  a. fine-tune Chronos **bolt_small** using **default** fine-tune setting for **120 seconds**: i.e. use hyperparameters from default setting – only include fine_tune = True and time_limit = 120;
  b. fine-tune Chronos **bolt_small** by **manually-defined** hyperparameters for **600 seconds**, hyperparameters include: fine_tune_lr = [1e-5, 1e-4], fine_tune_steps = [125,250,500,1000], fine_tune_batch_size = [16, 32, 64], and time_limit = 600;
  c. fine-tune Chronos **bolt_small** by **manually-defined** hyperparameters for **900 seconds**: same hyperparameters as (b), except time_limit = 900;
  d. fine-tune Chronos **bolt_base** using **default** fine-tune setting for **120 seconds**: same as (a), except we use bolt_base model preset;

e. fine-tune Chronos **bolt_base** by **manually-defined** hyperparameters for **600 seconds**: same as (b), except we use bolt_base model preset.
f. fine-tune Chronos **bolt_base** by **manually-defined** hyperparameters for **700 seconds**: same as (c), except we use bolt_base model preset. (tuning longer leads to performance drop)

2. Overall Model Performance

Note: Below are evaluation metrics given by leaderboard method, models on AG-TS report scores in a "higher is better" format, meaning that **most forecasting error metrics like WQL are multiplied by -1 when reported**.

- **Zero-shot forecasting**:
  a. Chronos **bolt_small** preset model:

model	WQL	MAE	MSE	RMSE	MAPE
Chronos[bolt_small]	-0.085167	-4.630450e+06	-2.847313e+13	-5.336022e+06	-0.118819

  b. Chronos **bolt_base** preset model:

model	WQL	MAE	MSE	RMSE	MAPE
Chronos[bolt_base]	-0.083125	-4.242827e+06	-2.231087e+13	-4.723438e+06	-0.106833

- **Fine-tuning**:
  a. Chronos **bolt_small** by **default** fine-tune setting for **120 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_small]	-0.053611	-2.292272e+06	-7.352864e+12	-2.711617e+06	-0.059662

  b. Chronos **bolt_small** by **manually-defined** fine-tune setting for **600 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_small]/T1	-0.055450	-2.578221e+06	-9.023399e+12	-3.003897e+06	-0.066755

  c. Chronos **bolt_small** by **manually-defined** fine-tune setting for **900 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_small]/T1	-0.055450	-2.578221e+06	-9.023399e+12	-3.003897e+06	-0.066755

  d. Chronos **bolt_base** by **default** fine-tune setting for **120 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_base]	-0.065991	-3.488302e+06	-1.642860e+13	-4.053221e+06	-0.087580

  e. Chronos **bolt_base** by **manually-defined** fine-tune setting for **600 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_base]/T1	-0.045370	-2.269731e+06	-8.050230e+12	-2.837293e+06	-0.059613

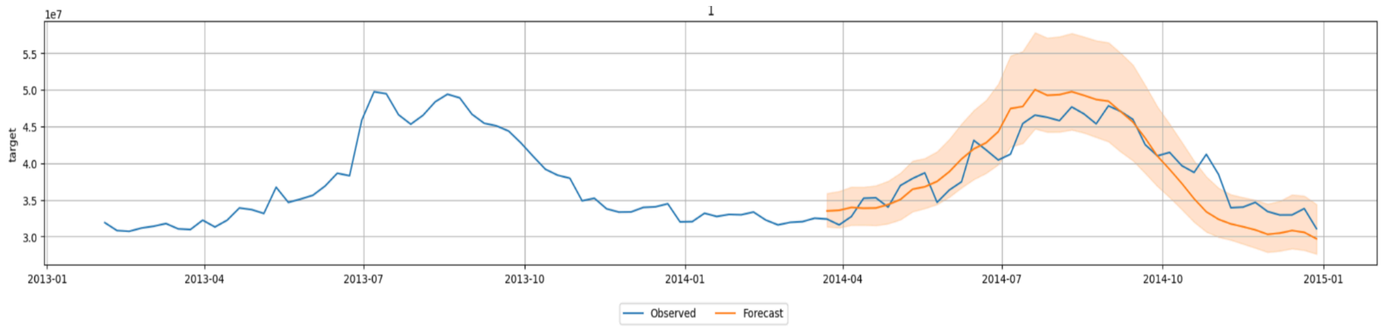
f. Chronos **bolt_base** by **manually-defined** fine-tune setting for **700 seconds**:

model	WQL	MAE	MSE	RMSE	MAPE
ChronosFineTuned[bolt_base]/T1	-0.044651	-2.235663e+06	-7.660431e+12	-2.767748e+06	-0.058854

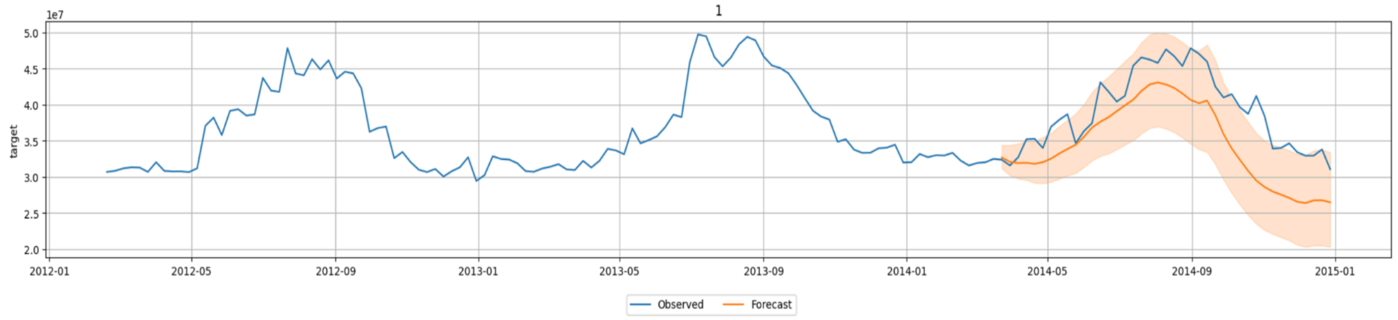
- Best-performing model: ChronosFineTuned[bolt_base]/T1 achieved the lowest overall MAPE of 5.89%.
- Worst-performing model: Chronos [bolt_small] showed the highest MAPE of 11.88%

3. Visualization of Model Performance

3.1 Best Model Actual vs. Forecasted Usage

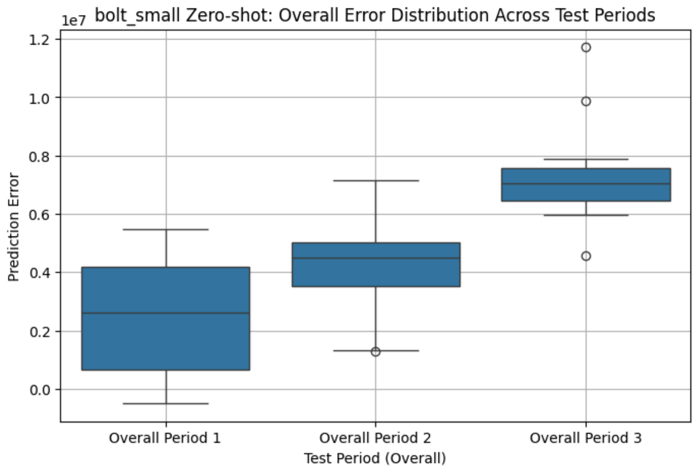
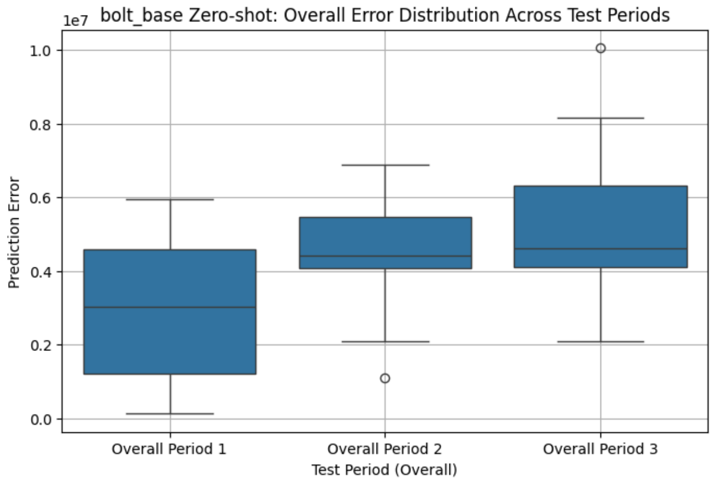
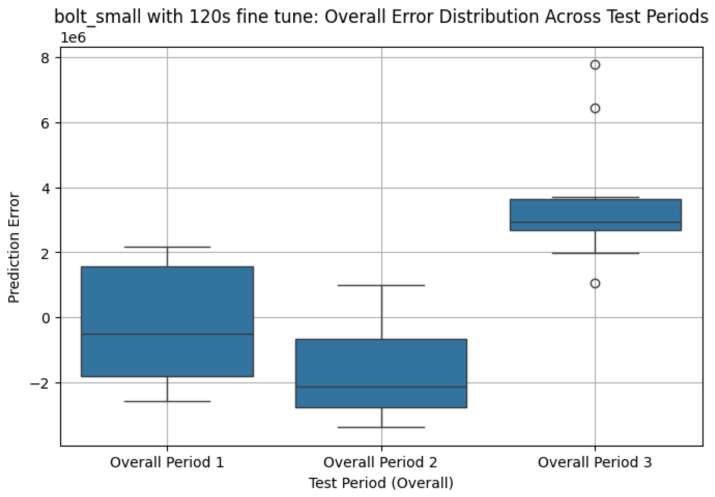


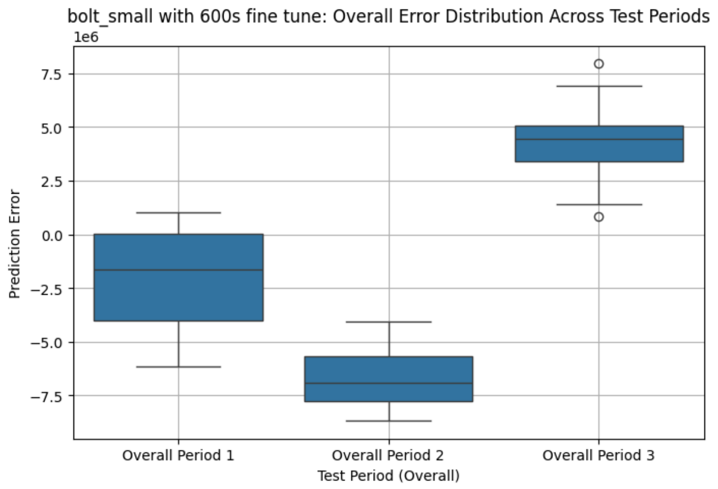
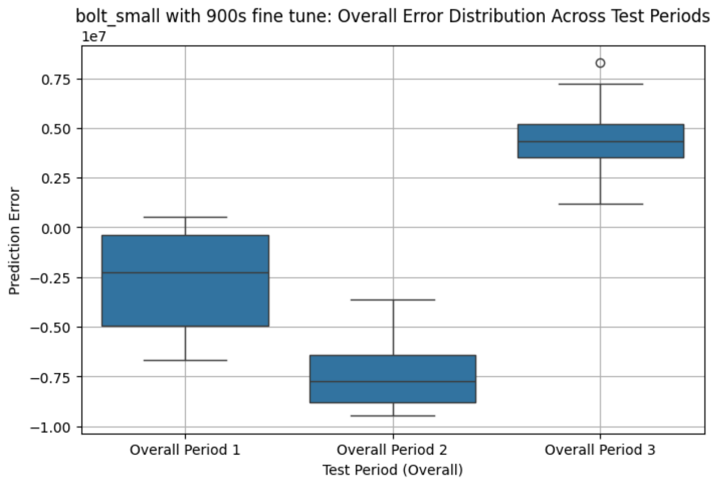
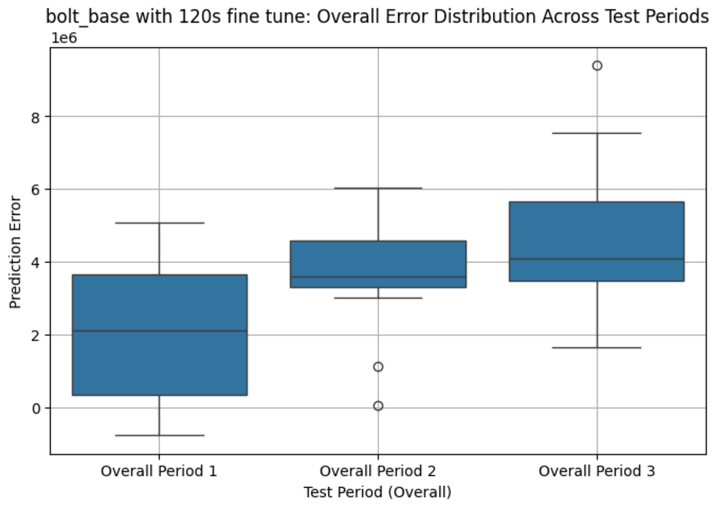
3.2 Worst Model Actual vs. Forecasted Usage

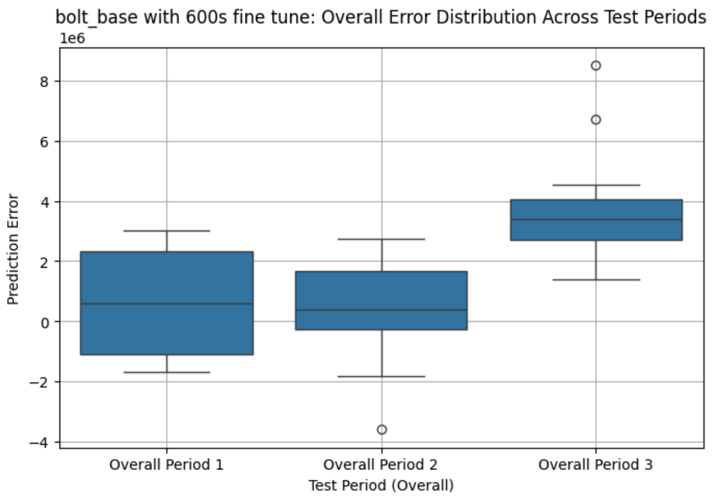
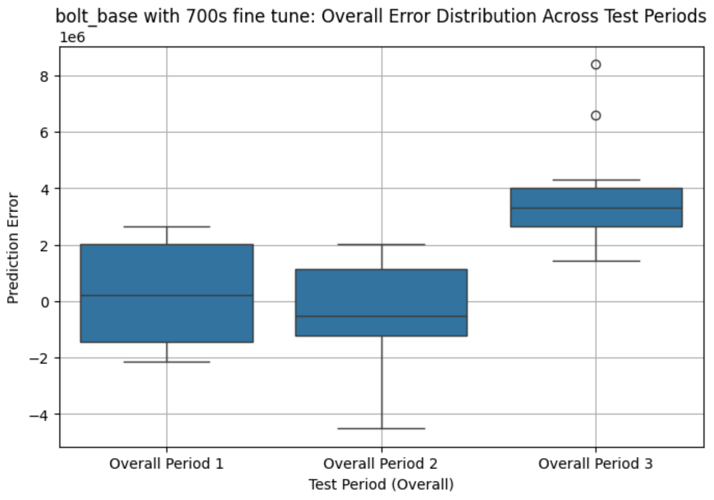


3.3 Error Distribution Across Test Periods

Model	MAPE Distribution Plot	Overall MAPE for Period 1, 2, 3

worst mode: bolt_small		6.57%, 9.36%, 20.32%
bolt_base		7.74%, 9.99%, 14.59%
bolt_small: 120s fine tune		4.47%, 4.31%, 9.37%

bolt_small: 600s fine tune		6.26%, 15.06%, 12.18%
bolt_small: 900s fine tune		7.46%, 16.35%, 12.57%
bolt_base: 120s fine tune		5.59%, 8.10%, 12.89%

Best Model: bolt_base: 600s fine tune		4.59%, 3.08%, 10.54%
bolt_base: 700s fine tune		4.45%, 3.28%, 10.24%

All Chronos models show the best performance in terms of average MAPE during recent time periods of training data and the worst performance when predicting a more distant time period.

4. Additional Comment for this model:

We also tried Chronos on Weekly Individual Electricity Usage data (concat weekly usage for each individual client to be a long time series), as well as Daily Overall Electricity Usage data (similar to Weekly Overall Usage, rather use daily temporal resolution), but we haven't tried Daily Individual Electricity Usage data. Details can also be found in our code notebook:

- For **Weekly Individual Electricity Usage data** (also for **Daily Individual data**), we found a potential issue due to the collection of this dataset: "*Some clients were created after 2011. In these cases consumption was considered zero*." For this reason, **there are lots of 0 for these clients' time series**, thus contributing to **shorter history** for the model to learn the trend of time series data, and **impairing the performance** – our performance for Weekly Individual Usage data is not promising (MAPE about 20%). Though we haven't had the chance to explore Daily Individual Usage data, we suspect the same issue applies to that case.

- Thus, those zero data may be deleted, then perform the model again to see if there will be improvements.
- Our results for **Daily Overall Electricity Usage data** seem to be promising as well, but for comparison with other models in our group, we haven't expanded its analysis on this tech doc. Similarly, this can be explored for continuation.

Model 3 Results: Prophet

1. Model Selection and Training Setup

We have tried following models for Prophet:

- Model 1: Prophet with **Linear** Trends, **Customized Weekly** Seasonality, **Customized Monthly** Seasonality, **Default Yearly** Seasonality
- Model 2: Prophet with **Logistic** Trends, **Customized Weekly** Seasonality, **Customized Monthly** Seasonality, **Default Yearly** Seasonality

For both models, do hyperparameter tuning:

- Changepointg_prior_scale: `np.arange(0.2, 1, 0.05)`
- Monthly seasonality Fourier_order: `range(5, 15)`
- Weekly seasonality Fourier_order: `range(3, 10)`

Train models using first 60% data, do model selection by validating using sequential 20% data and lastly test using last 20% data

2. Overall Model Performance

Best parameters found during hyperparameter tuning:

Prophet Model	**Best Hyperparameters**
Model 1 (Linear)	{'scale': 0.35, 'monthly_order': 14, 'weekly_order': 7}
Model 2 (Logistic)	{'scale': 0.98, 'monthly_order': 11, 'weekly_order': 9}

The corresponding Model performance:

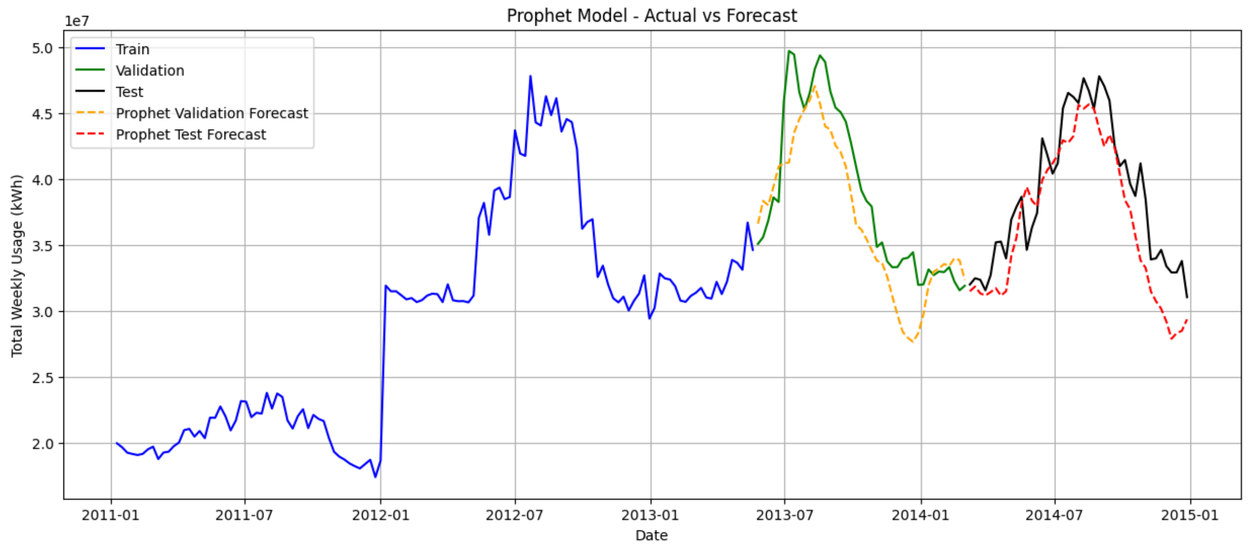
Prophet Model	**Overall Test MAPE**	**Test Period I MAPE**	**Test Period II MAPE**	**Test Period III MAPE**

Model 1 (Linear)	10.75%	15.61%	9.00%	7.28%
Model 2 (Logistic)	6.66%	5.58%	4.11%	10.38%

- Best-performing model: Logistic Prophet model with {'scale': 0.98, 'monthly_order': 11, 'weekly_order': 9} achieved MAPE of 6.66%.
- Worst-performing model: LinearProphet model with {'scale': 0.35, 'monthly_order': 14, 'weekly_order': 7} achieved MAPE of 10.75%

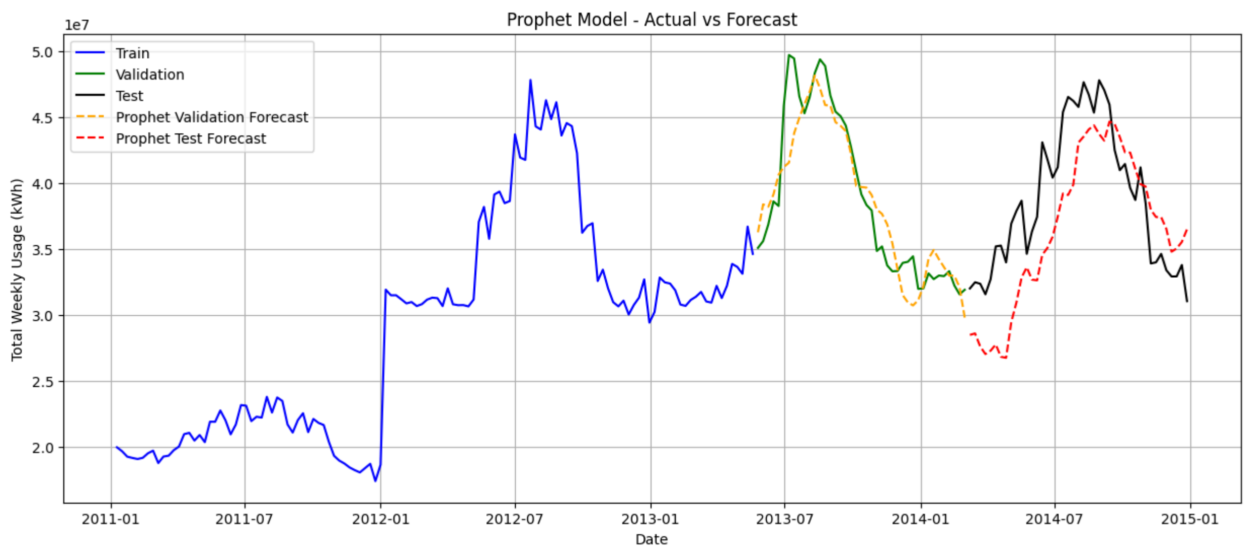
3. Visualization of Model Performance

3.1 Best Model Actual vs. Forecasted Usage



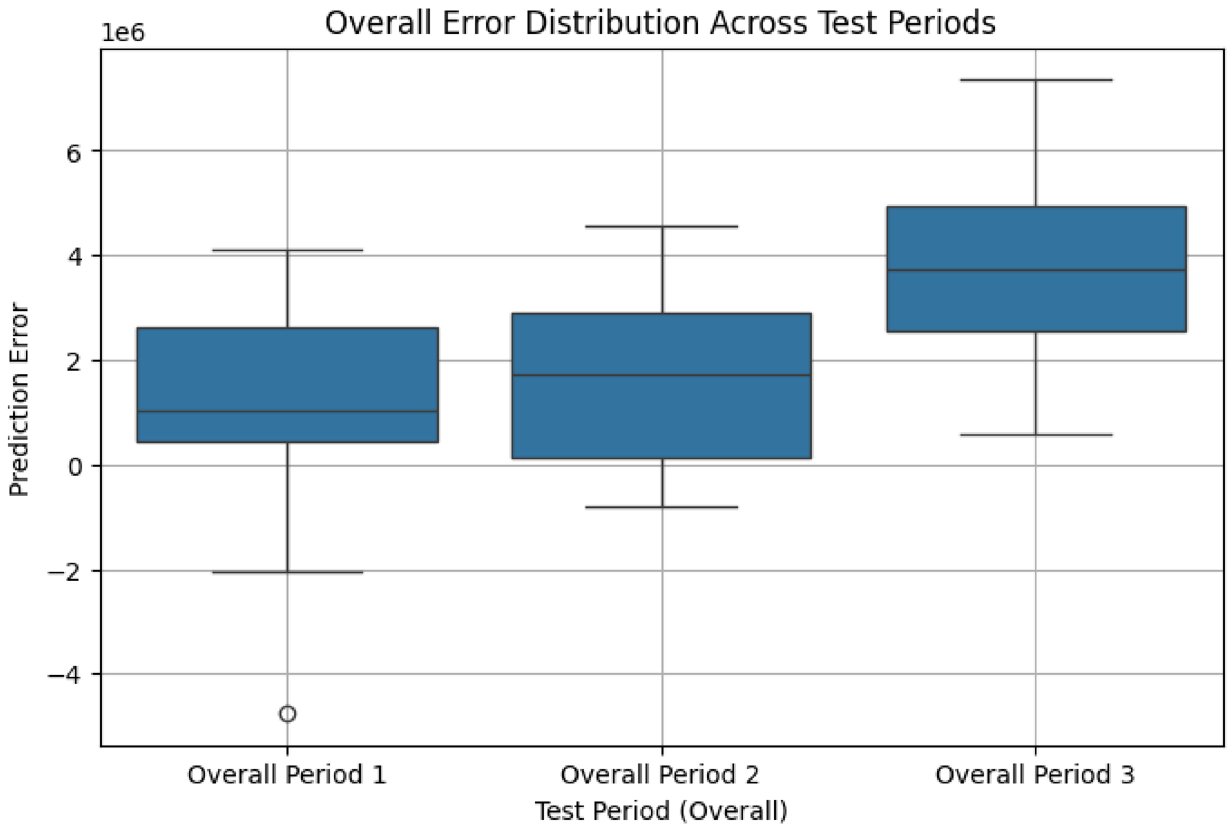
- The model captures the most patterns including seasonality and trends. A slight deviation is observed around the end of year 2023. The high frequency components seem lacking.

3.2 Worst Model Actual vs. Forecasted Usage

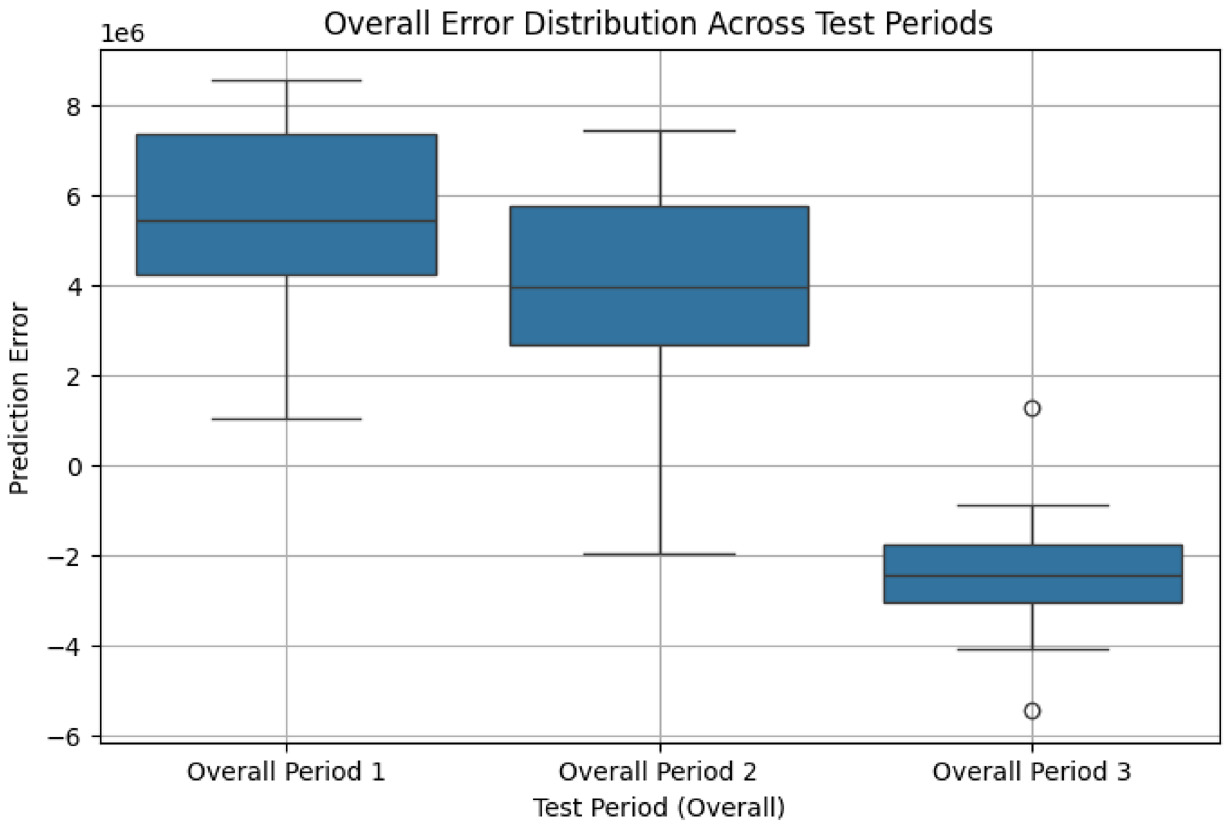


- The model captures the most patterns for validation range but for the test range, the model gave a bad prediction with some systematic deviation.

3.3 Error Distribution Across Test Periods



- The best model exhibits a relatively stable error distribution across test periods.



- The worst model displays a relatively unstable model. In period 3, the model gives some outliers.

Further Improvements

For SARIMA Model:

- Addressing seasonal effects with SARIMAX: One of the error plots revealed a significantly higher error in one of the test periods, which is likely due to seasonal fluctuations in electricity consumption. Currently, the SARIMA model does not incorporate exogenous variables, meaning it relies solely on past values of the target variable. To address this, SARIMAX can be implemented by adding external factors such as holiday indicators and temperature variations.
- Handling structural shift: The dataset shows a large structural shift in energy consumption around 2012, which is the result of a fundamental change in user behavior. Future steps could consider creating a dummy variable to indicate the period before and after the shift to adjust predictions accordingly.

For Chronos Model:

- Part of future improvements are listed in the additional comments from the previous section above.
- Consider adding covariate analysis as mentioned when we introduce how to use this model in the model design section.
- Switch to using GPU rather than CPU for model fine-tuning, and also change hyperparameters, then check if performance will improve.

For Prophet Model:

- Add customized yearly seasonality and daily seasonality and tune the Fourier Order of them, which could capture more patterns.
- Add more necessary holiday components in addition to what we have already modeled here for the big jump. With additional holiday components, the trends and seasonalities can be better learnt.

Team Information

Zhiqi Ma (zm2467)

Ruoheng Du (rd3165)

Xiaojiang Wu (xw3022)

Yijian Liu (yl5778)