Forecasting Project
Technical Documentation

May 1, 2025

Zhiqi Ma, Ruoheng Du, Xiaojiang Wu, Yijian Liu

Contents

Introduction

With the rapid expansion of e-commerce, online retail has become a crucial component of the global economy, providing businesses with unprecedented opportunities to reach and engage consumers. As customer demand fluctuates and market dynamics evolve, the ability to accurately forecast sales has emerged as a critical challenge. Reliable sales forecasting enables organizations to make informed decisions regarding inventory management, resource allocation, and operational planning. To enhance the accuracy of predictive models, categorizing products based on shared characteristics can offer valuable insights into distinct sales patterns. In this study, product categorization is performed using unsupervised clustering techniques, specifically KMeans, to group similar products without relying on predefined labels. The primary focus of this research is to evaluate the predictive performance of machine learning models on clustered retail sales data, rather than to assess profitability or optimize business operations. By systematically analyzing forecasting results across different product clusters, this project aims to contribute to a better understanding of model capabilities in the context of complex retail datasets.

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
7. Further Improvement

Data Description

The dataset we used is **Online Retail II** from UCI. The dataset comprises transactional records from a UK-based, non-store online retailer specializing in unique all-occasion giftware. The dataset captures all transactions between **December 1, 2009, and December 9, 2011**, providing insights into customer purchasing behavior over two years. The retailer primarily serves wholesalers. Each transaction includes invoice number, stock code, product description, quantity purchased, invoice date, unit price, customer ID, and country. The dataset contains **1,067,371 instances**.

- **InvoiceNo**: Invoice number. Nominal. A 6-digit integral number is uniquely assigned to each transaction. If this code starts with the letter 'c,' it indicates a cancellation.

- **StockCode**: Product (item) code. Nominal. A 5-digit integral number is uniquely assigned to each distinct product.

- **Description**: Product (item) name. Nominal.

- **Quantity**: The quantities of each product (item) per transaction. Numeric.

- **InvoiceDate**: Invoice date and time. Numeric. The day and time when a transaction was generated.

- **UnitPrice**: Unit price. Numeric. Product price per unit in sterling (£).

- **CustomerID**: Customer number. Nominal. A 5-digit integral number is uniquely assigned to each customer.

- **Country**: Country name. Nominal. The name of the country where a customer resides.

Research Question

The previous team's research question is that:

How can sales forecasting be conducted for different product categories in an online retail environment to support organizational profitability? In other words, **which category is the most profitable in the next quarter**?

We revised the research question to be:

How accurately can statistical, machine-learning-based, and foundation models forecast sales in the retail sector, both forecasting directly and when retail product categories are determined through unsupervised clustering techniques such as KMeans, with the primary objective of **assessing the predictive performance of the models rather than evaluating the profitability of individual product groups**?

Data Processing

This section details the necessary data transformations and data cleaning, as well as diagnostic checks performed before modeling.

Data Cleaning

Notice that the last team did not perform any data cleaning except removing missing Description rows, but it turns out that the raw dataset contains a huge number of abnormal data entries that should be excluded to ensure data integrity and smooth modeling in later sections.

The raw dataset exhibited several common data quality issues, including missing values, duplicate entries, irrelevant records, and inconsistencies in product identifiers. The following cleaning procedures were applied to ensure the integrity and usability of the dataset:

2.1 Missing Values and Data Type Validation

- The InvoiceDate field was converted to pandas datetime format for temporal operations.
- Only Description and CustomerID columns contain missing values: about 0.41% (4,283 missing values) and 22.77% (243,007 missing values) respectively.
- Given the dataset's large size, containing 1,067,371 instances, rows with missing Description were removed, as this field is central to product categorization.
- The CustomerID variable was only used for exploratory data analysis and not included in modeling, thus missing CustomerID values were not dropped before modeling.
- After removing missing Description rows, we found that: among all retail records, there are 5,305 types of products, and 53,628 transactions from 5,942 customers.

2.2 Negative Quantities and Cancellations

- Transactions with Quantity ≤ 0 or UnitPrice ≤ 0 were removed unless they represented valid cancellations.
- Cancellations were identified by InvoiceNo values beginning with 'C'. To distinguish between actual reversals and unmatched anomalies:
    - For each cancellation entry, a matching original sale with equal and opposite quantity, same StockCode and CustomerID, was searched.
    - Only if an exact match was found, both records were dropped (netting out the effect).
    - All unmatched cancellation records were then removed as they represented incomplete or inconsistent transaction history.
- This logic reduced the dataset while preserving only verified, unreversed transactions.

- We found that: Number of cancelled records: 19,494; and Number of cancellations with exact matching original: 15,715. Notice that these two numbers did not match, maybe due to missing records or input errors, but it's hard to fix such issues.

2.3 Duplicate and Redundant Records

- Duplicate rows across all columns were detected using df.duplicated() and dropped.
- This step eliminated redundancies often caused by multiple system writes or reprocessed invoices.

2.4 Non-Product StockCode and Description Filters

To ensure that the dataset only includes records representing actual salable items:

- **Stock Code Filtering**:
    - All entries with StockCode not matching a numeric or alphanumeric SKU format were inspected (not directly dropped since there is no specific rule regarding StockCode's naming policy provided, so we need to verify them later).
    - Administrative or placeholder codes such as 'POST', 'ADJUST', 'DOT', 'TEST001', 'BANK CHARGES', and gift voucher codes like 'gift_0001_30' were identified and excluded.
    - Valid alphanumeric codes (e.g., DCGS0003, 84032B) were retained if they referred to actual product descriptions.
    - A manually curated whitelist/blacklist was built after investigating product descriptions and string patterns.
- **Description Filtering**:
    - Since there were still some special cases in the Description column, we also double checked and tried to detect them.
    - Descriptions containing metadata phrases or non-product notes (e.g., "Adjustment by John…", "Manual Entry", "Check…", "Missing item", "Wet Return", 'broken', 'crushed', 'no sale') were identified via keyword matching and removed after manually inspection..
    - A structured keyword dictionary was built to flag such non-salable items.

2.5 Normalization and Cleanup

- All product descriptions were converted to lowercase.
- Special characters were removed (e.g., @!#%&*?) using regex, reducing noise for downstream NLP tasks.
- White spaces were normalized, and records with placeholder or corrupted values (e.g., '?', '...', or missing terms) were discarded.

2.6 Final Output Construction

- An additional variable – Sales – calculated as UnitPrice × Quantity, was introduced for modeling.
- A fully cleaned version of the dataset (*'cleaned_retail_dataset.csv'*) was retained after dropping CustomerID to be used for modeling.

Notice: when the last team implemented the Prophet model to forecast daily sales, they excluded all days with 0 sales (there are lots of dates with 0 sales as indicated by time series plot from below) before training the model – this should not be allowed and cannot represent the real time series. While as for the other models from the last team, they kept the dates with 0 sales, so it seems that they hadn't matched within the team regarding this issue.

We thus fixed this issue, and not removed dates with 0 sales, in our modeling process.

Data Diagnostics

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

- Total number of records verification
- Detection of duplicate entries (none were found in the cleaned dataset)
- Identification of missing values (none were found in the cleaned dataset)
- Validation of numerical field sums and consistency

Modeling Data Creation

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

- Resampling: Data was aggregated from minute level granularity to daily or weekly granularity for the model to use.

- Categorization: Data may also be aggregated based on product category, depending on model's specification.

Target Variables

The project aims to predict total sales in weekly granularity for the online retail dataset by leveraging historical sales patterns. The objective is to develop some predictive models that can accurately estimate future weekly sales, taking into account both seasonal trends and overall sales dynamics.

The target variable is defined as:

- Sales per week – Aggregated from individual sales data across all products.

By predicting weekly sales, this model can help in inventory management, demand planning, and optimizing business strategies.

Predictive Variables

Predictive variables are the features used to forecast weekly total sales in this project. These variables were selected based on their ability to capture overall sales trends, seasonality, and short-term fluctuations, ensuring a robust forecasting model.

We categorize the predictive variables into two types:

1. **Direct Variables** – Extracted directly from the dataset, such as historical sales records aggregated at different time resolutions.
2. **Derived Variables** – Created through transformations or aggregations of the direct variables to better enhance predictive power, such as lag features and date-based features.

Variable List

1. **Weekly Total Sales**: The sum of sales aggregated to a weekly level. This serves as the primary predictor, capturing broad consumption patterns and seasonal trends.
2. **Time Features**: Calendar-related variables such as the week number, month, and day of the week, used to model recurring seasonal effects and potential sales spikes or drops.

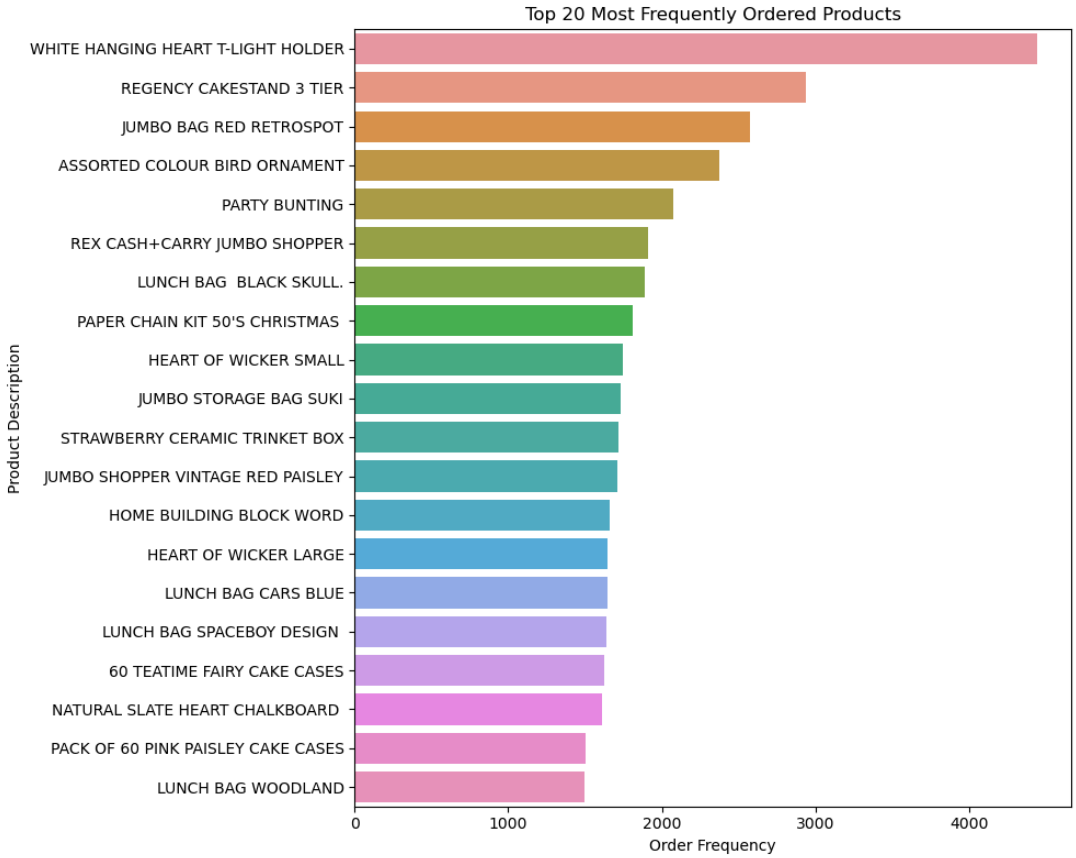
These predictive variables form the baseline across models. Additional features or transformations may vary depending on the specific forecasting model applied (e.g., XGBoost, Prophet, or Chronos), as some models benefit from richer engineered inputs while others automatically learn temporal structures.

Pre-modeling EDA

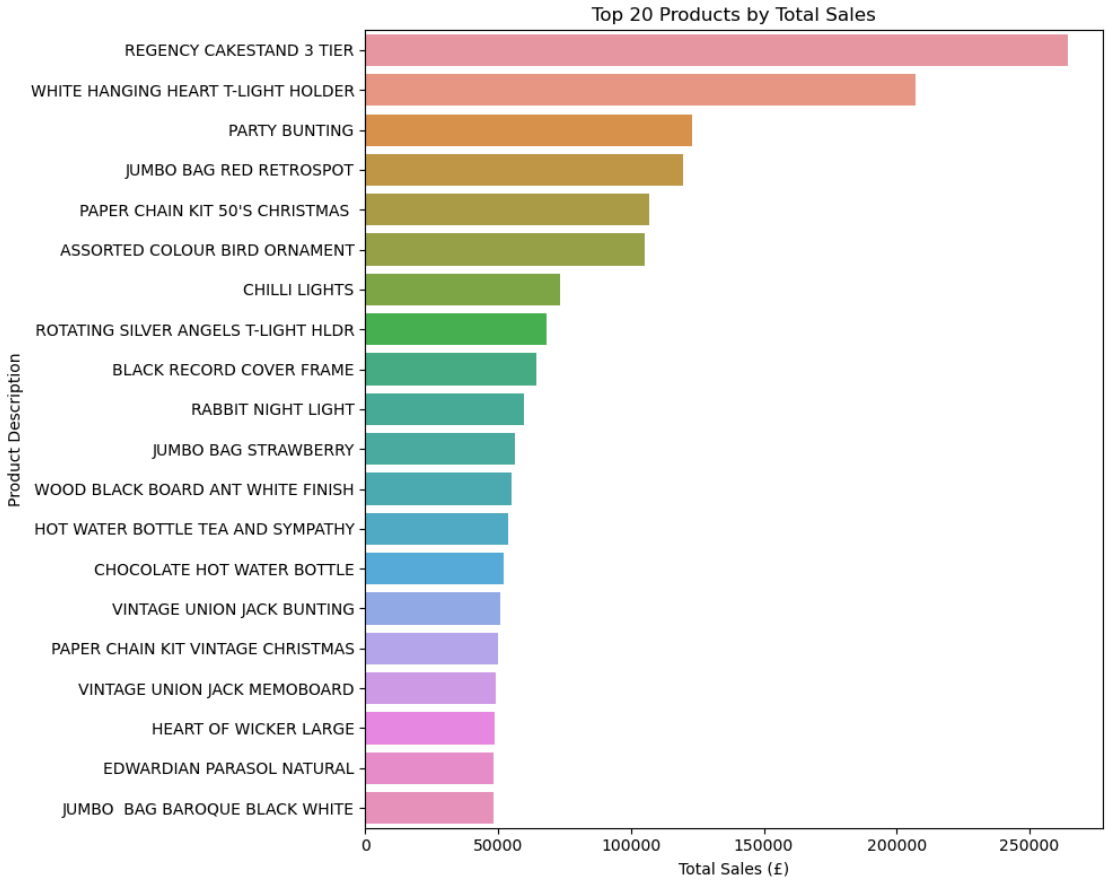
The dataset spans from 2009 to 2011, capturing daily retail sales records trends. The forecasting model is trained on historical consumption data and aims to predict future usage patterns. To better understand the characteristics of retail sales, we conducted exploratory data analysis (EDA), including trend analysis, periodicity detection, and anomaly detection. Below are key insights derived from the analysis:

Transaction Pattern

- *Image 1: Top 20 Most Frequently Ordered Products*

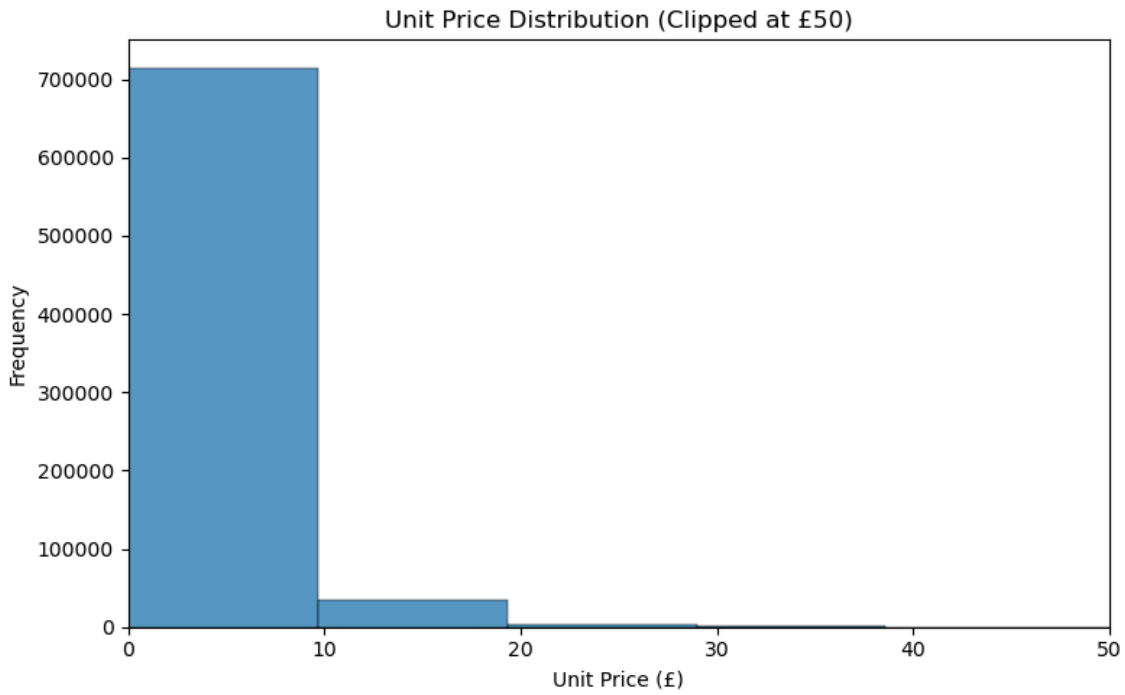


- *Image 2: Top 20 Products by Total Sales*

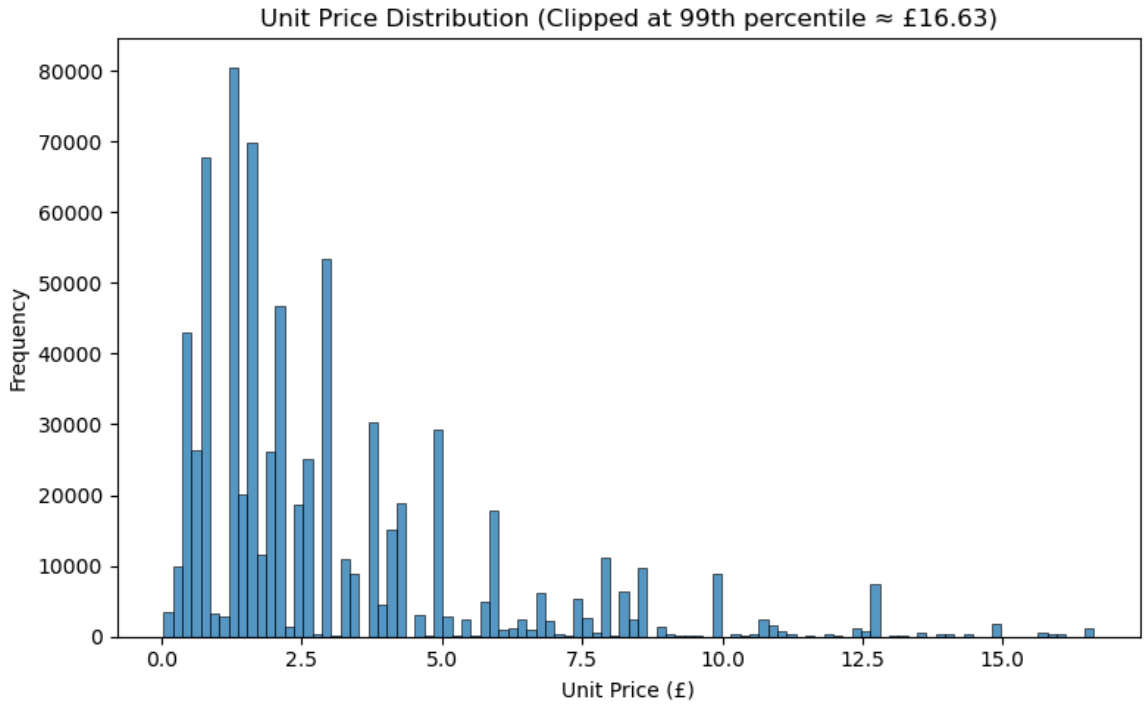


Product UnitPrice Pattern

- *Image 3: UnitPrice Distribution (Clipped at 50)*
    - Notice that there are 753,156 entries in the cleaned data, and about 720,000 entries have unit price within the interval 0-10.

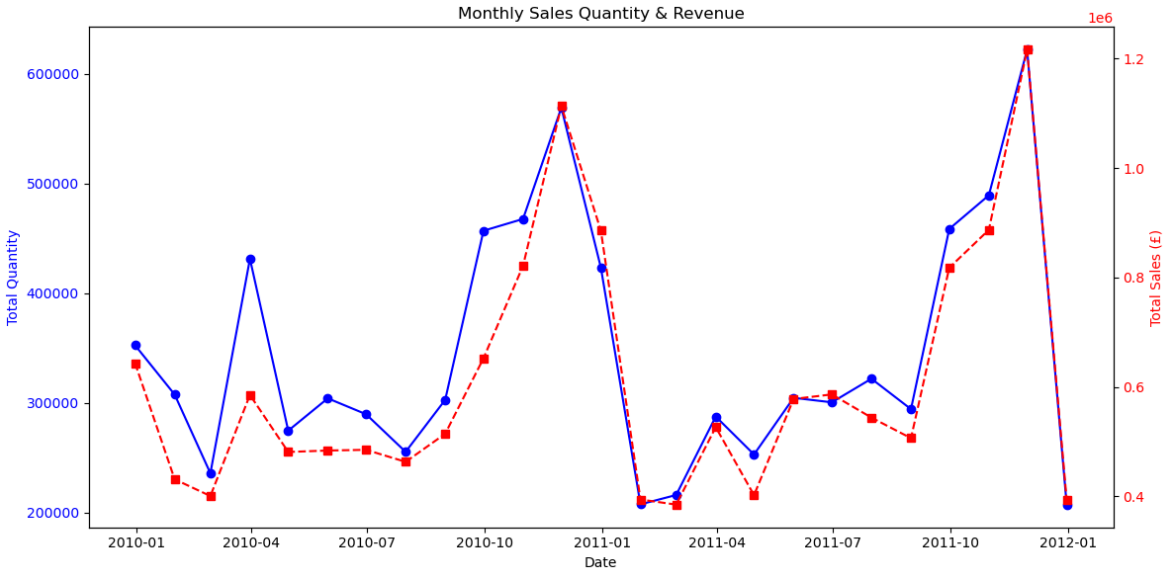


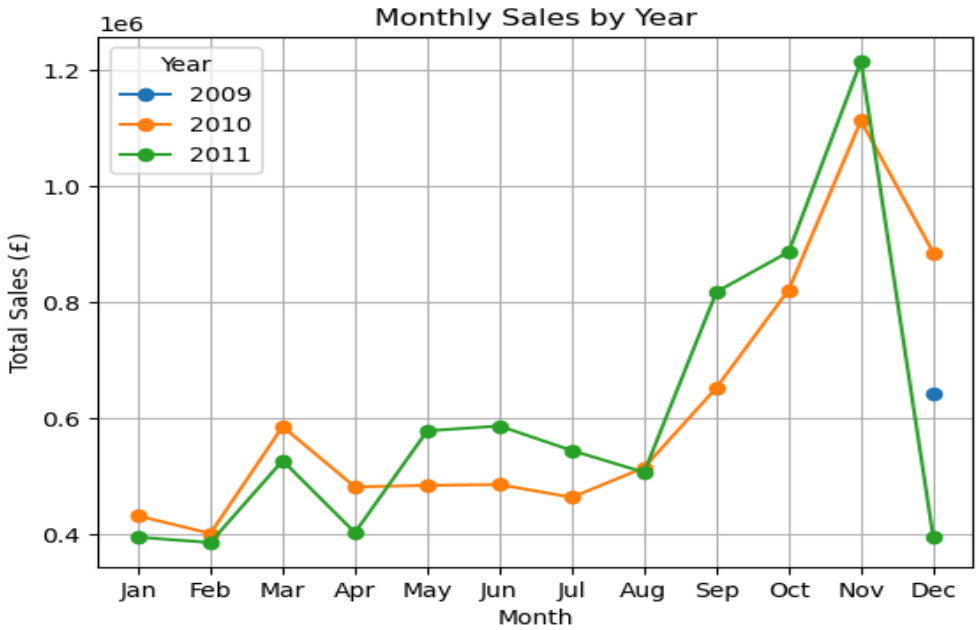
- *Image 4: UnitPrice Distribution (Clipped at 99th percentile)*
    - We can see more details in this plot – a huge proportion of products have unit prices between 1 to 5.



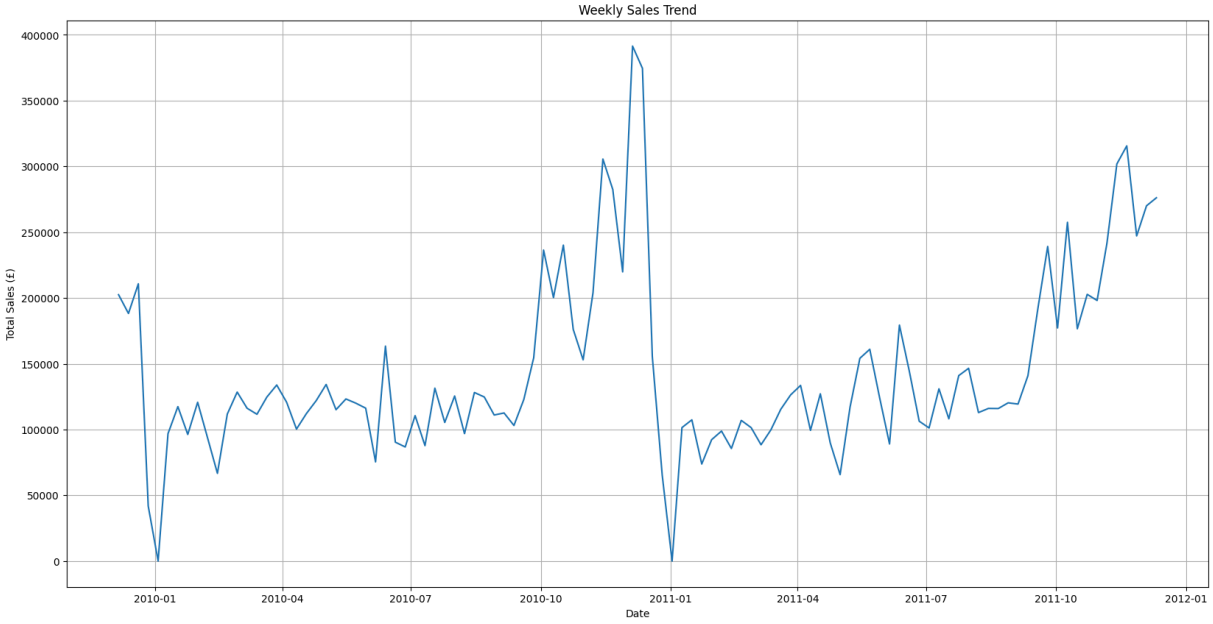
Long-Term Consumption Trends

- *Image 5: Monthly Product Sales*



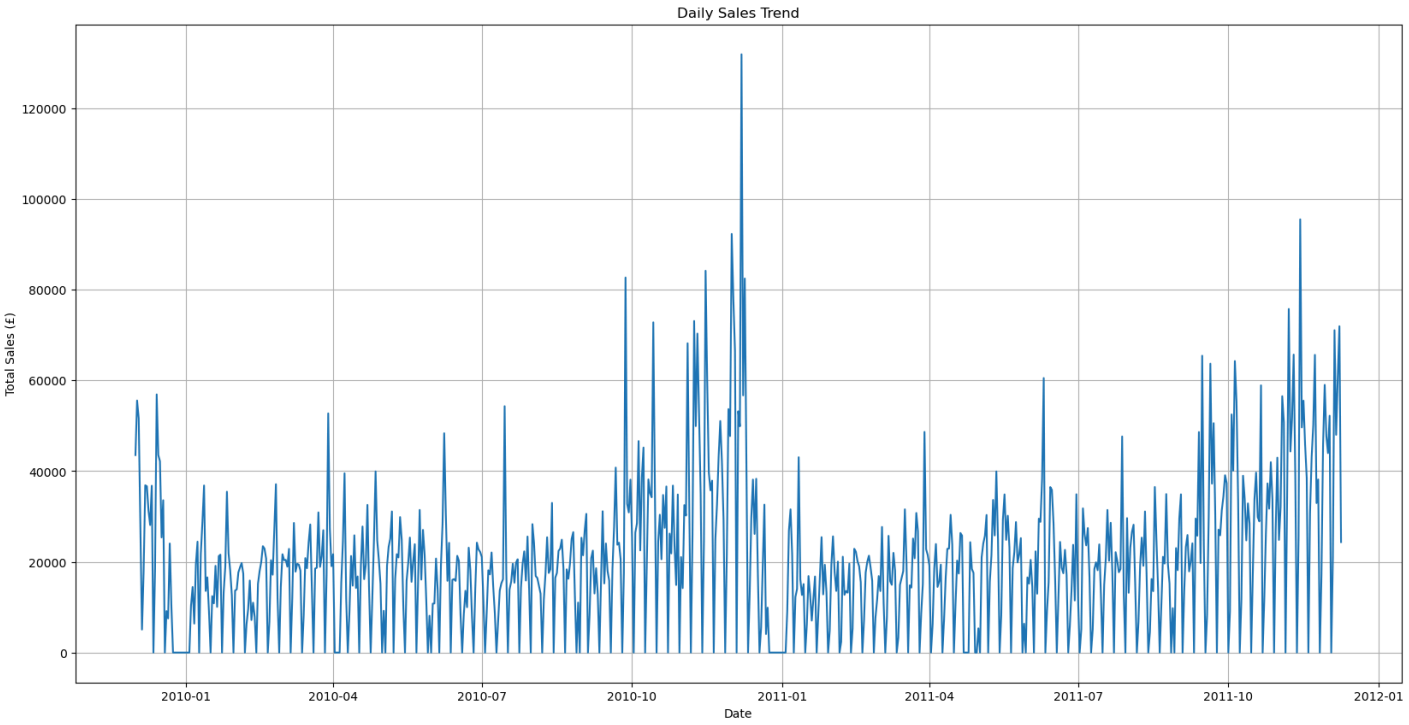


- *Image 6: Weekly Product Sales*



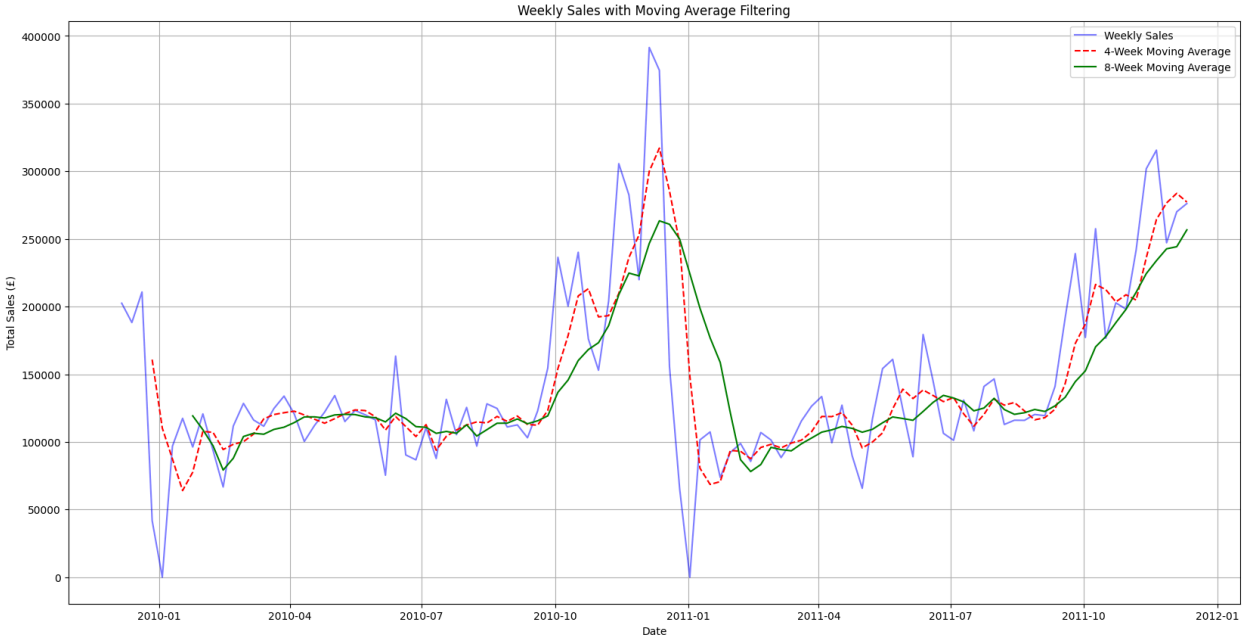
- The weekly total sales exhibit strong seasonality with clear spikes during the year-end holiday periods and occasional sharp drops, indicating irregular large-scale sales events or data anomalies.

- *Image 7: Daily Product Sales*



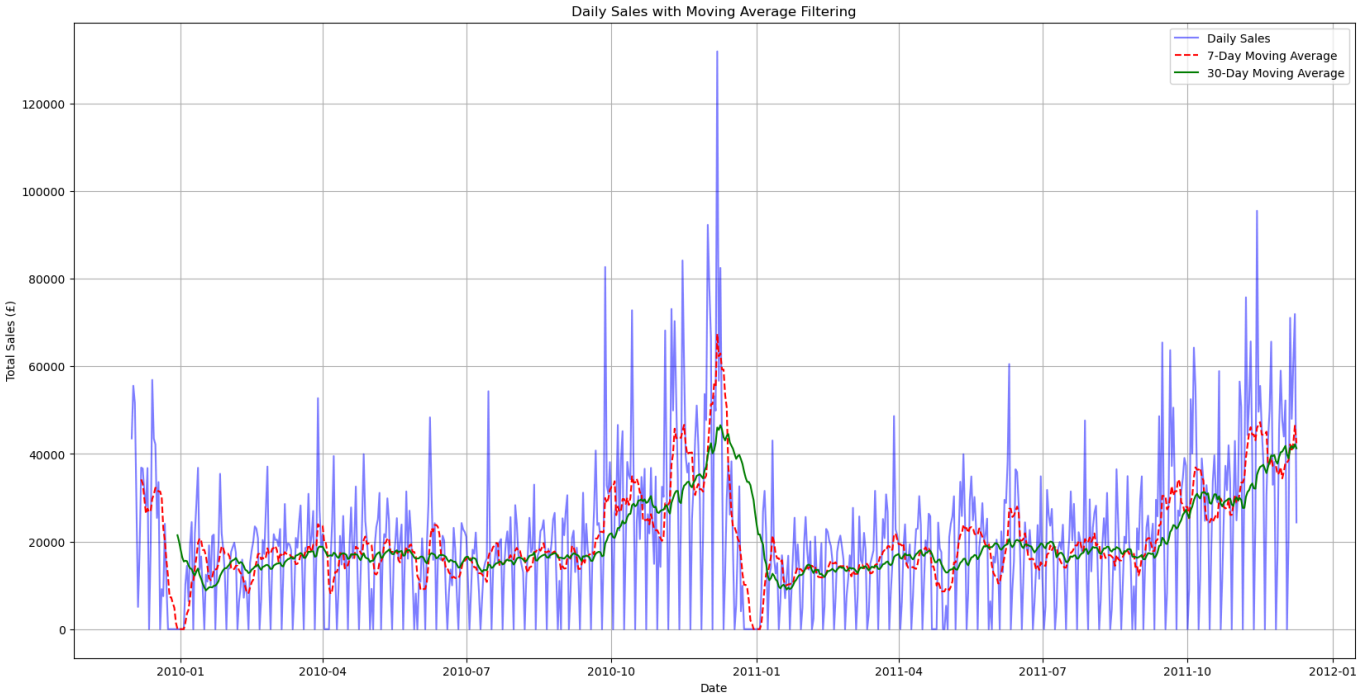
- The daily sales exhibit high volatility with frequent sharp spikes and dips, suggesting irregular large transactions or special events impacting sales patterns.

- *Image 8 : Moving Average of Weekly Sales*



- Applying 4-week and 8-week moving averages smooths out short-term volatility, revealing an overall upward sales trend with recurring seasonal peaks around the end of each year.

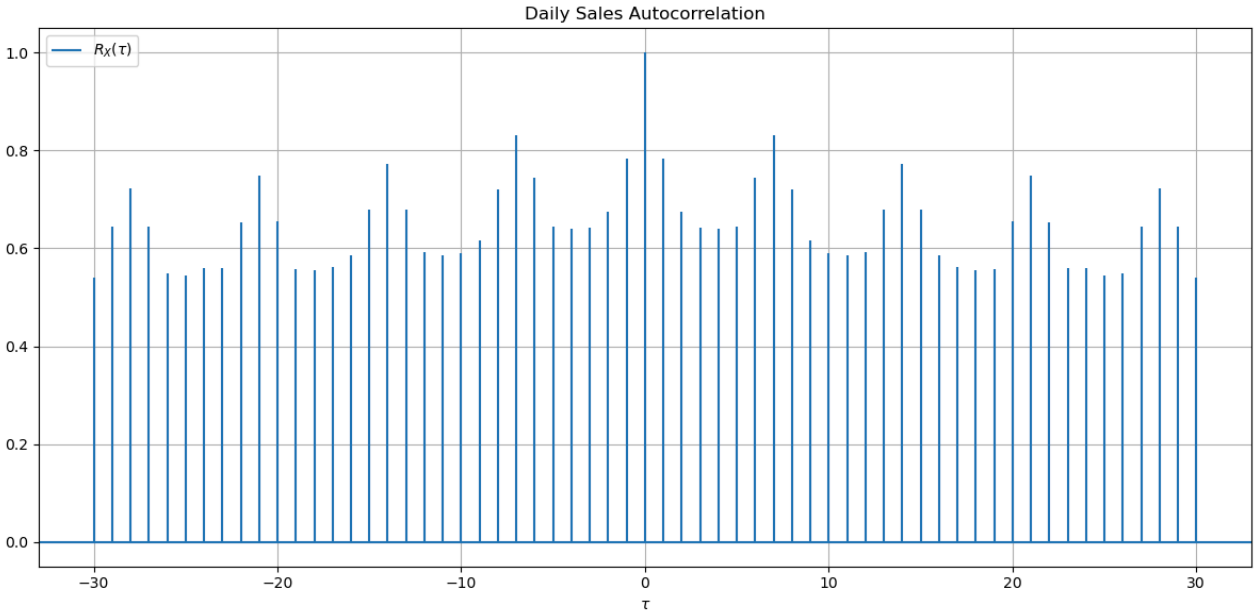
- *Image 9: Moving Average of Daily Sales*



- Smoothing the daily sales with 7-day and 30-day moving averages reveals an underlying gradual upward trend, punctuated by seasonal peaks likely corresponding to holiday sales periods.

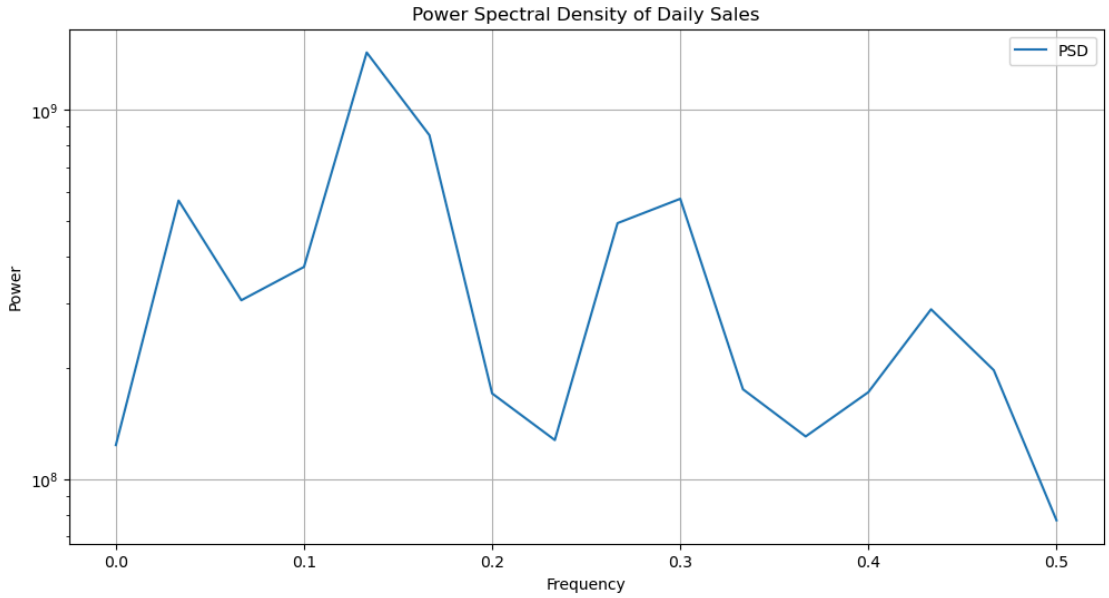
Periodicity Detection

- *Image 10 : Daily Sales Autocorrelation*

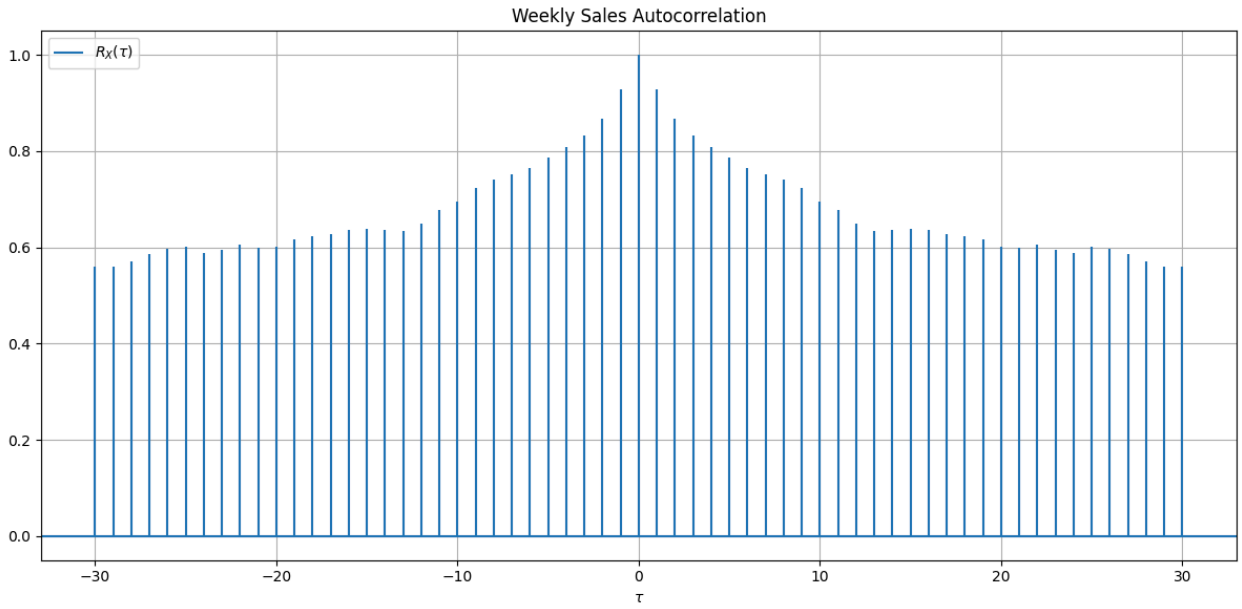


- We don’t use statistical models that require data to be stationary like GLM, so autocorrelation and PSD plots are only shown for illustration purposes. We’ll not handle stationarity here.

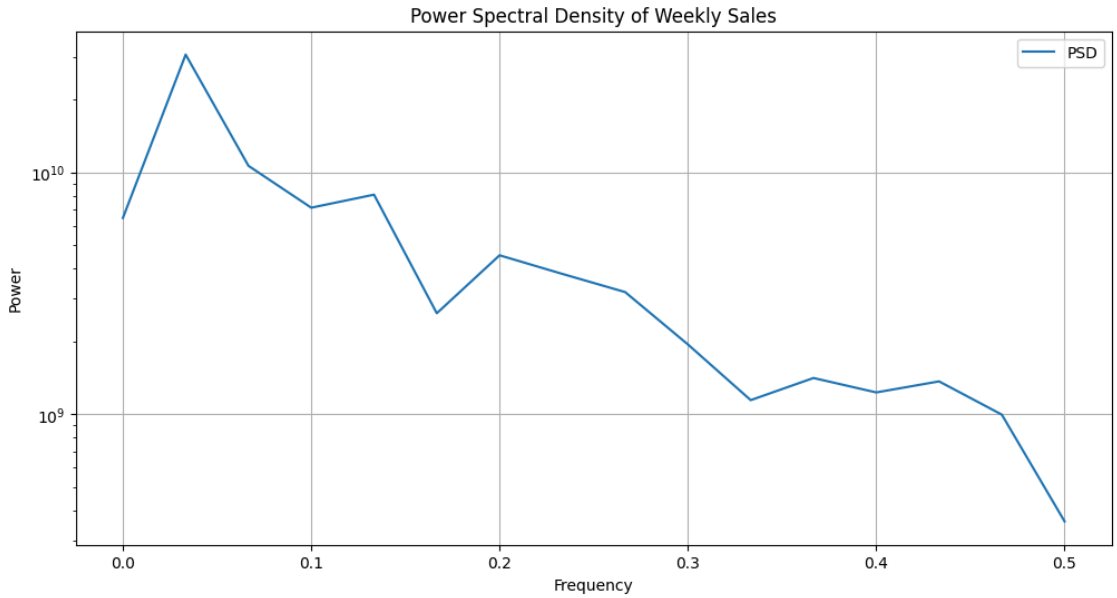
- *Image 11 : Daily Sales Power Spectral Density (PSD)*



- *Image 12 : Weekly Sales Autocorrelation*

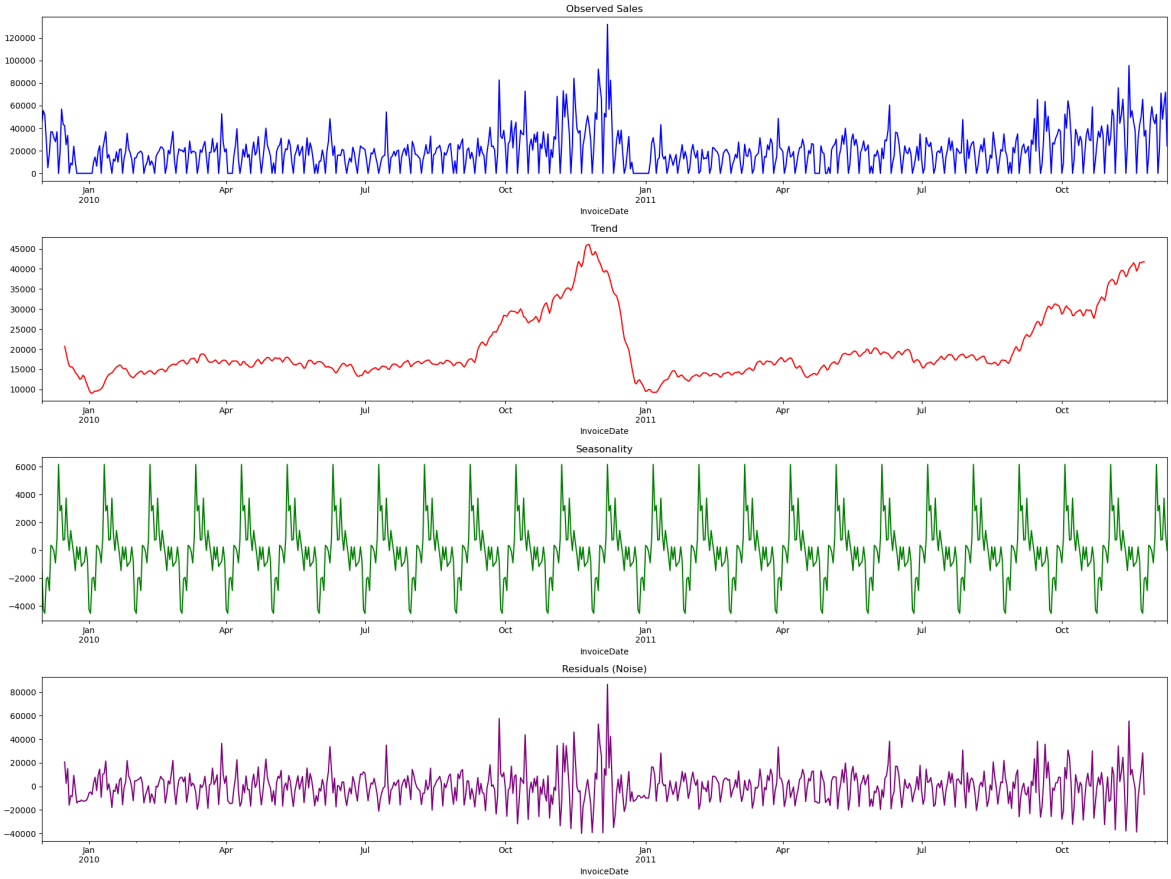


- *Image 13 : Weekly Sales Power Spectral Density (PSD)*

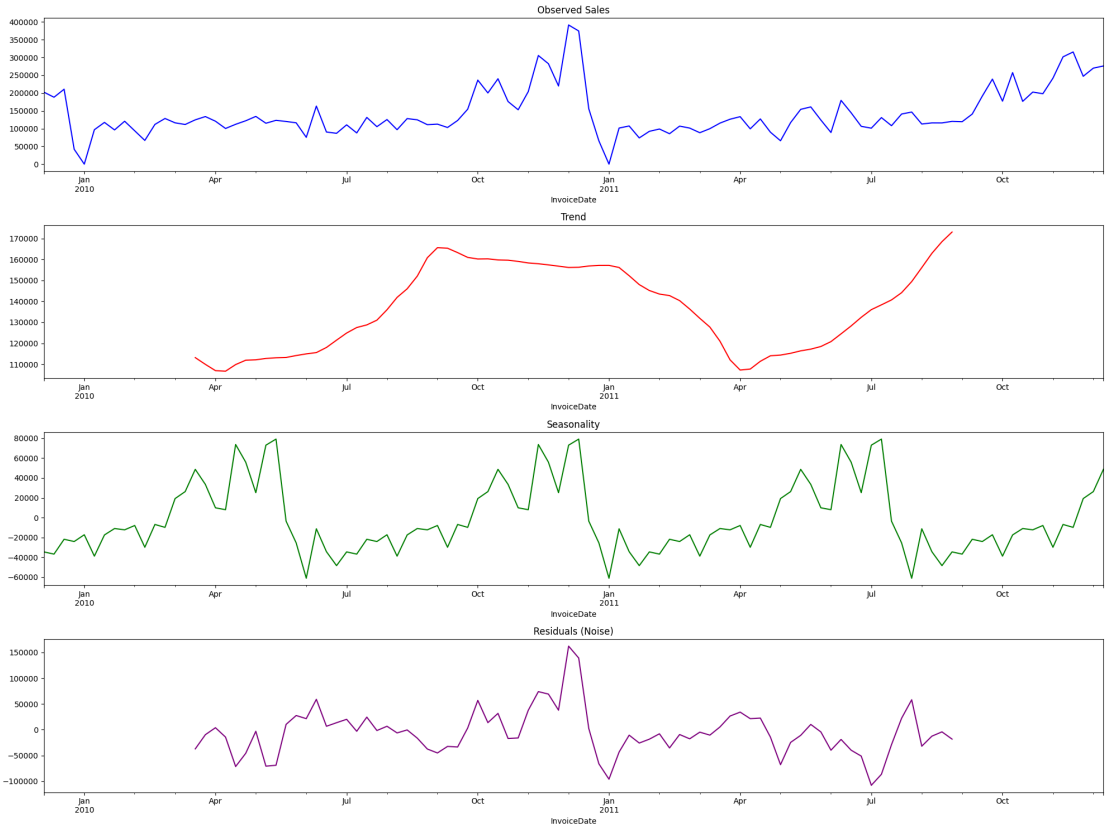


Time Series Decomposition

- *Image 14 : Daily Sales Trend, Seasonal, and Residual Components*

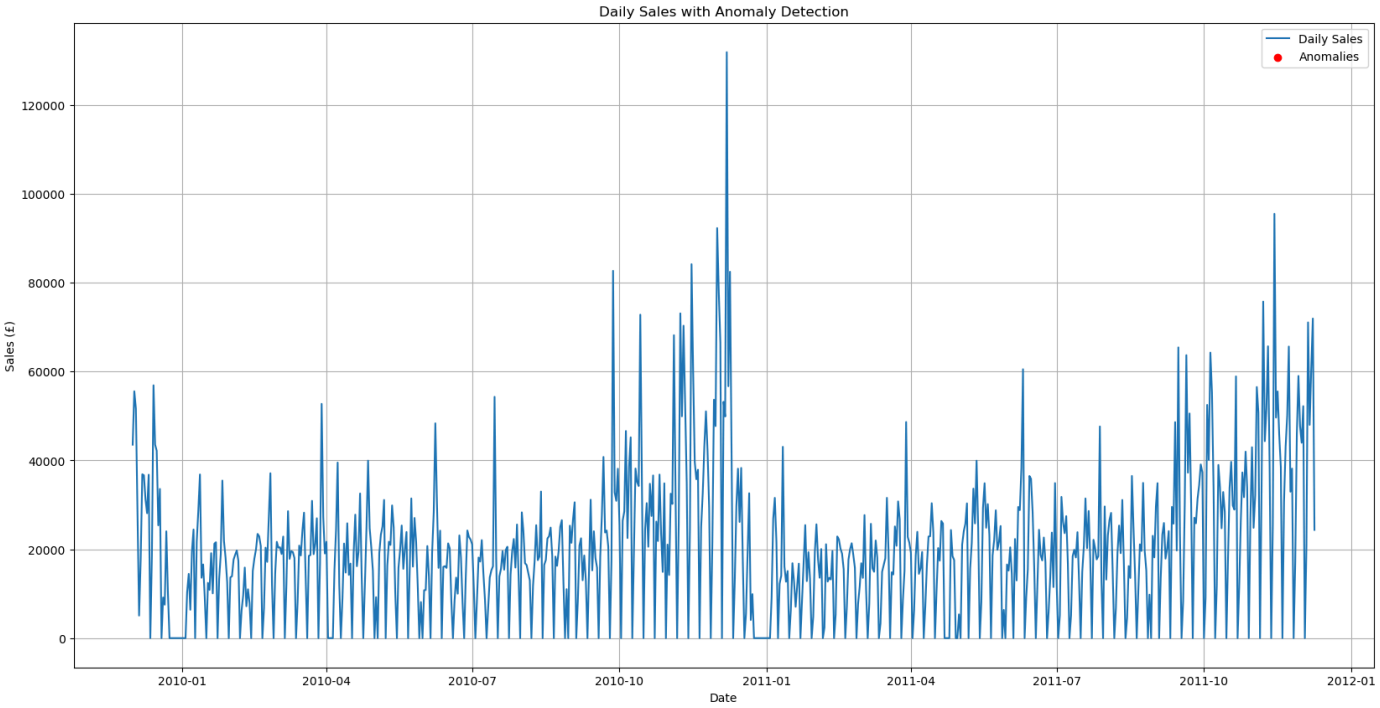


- *Image 15 : Weekly Sales Trend, Seasonal, and Residual Components*



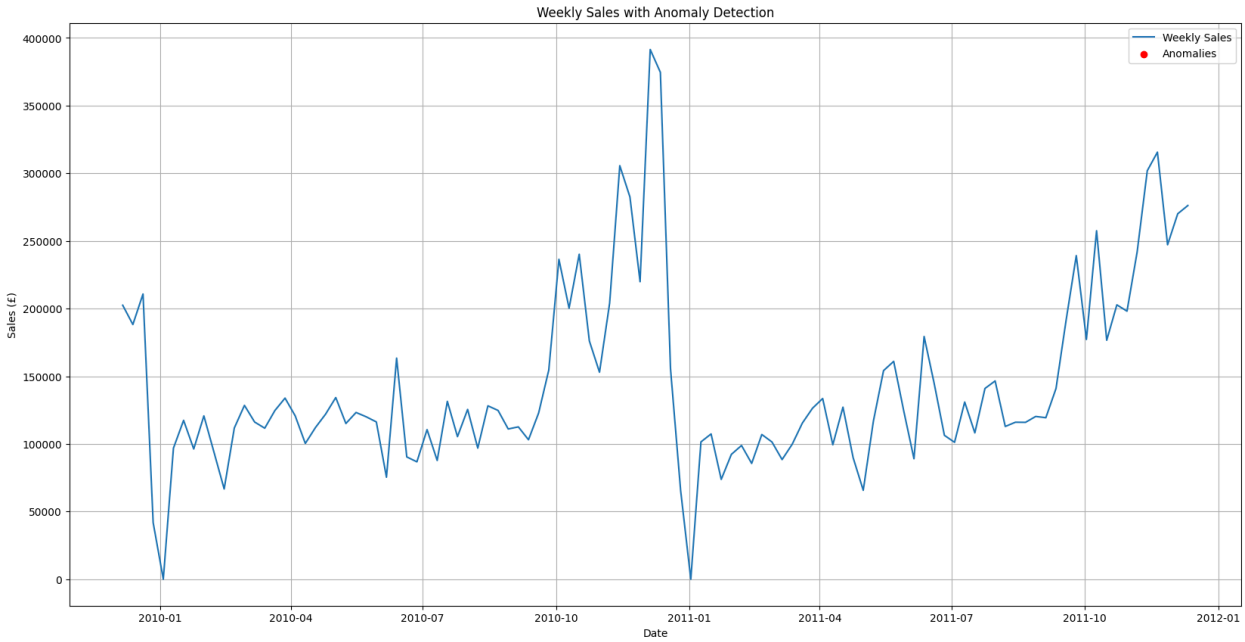
Anomaly Detection

- *Image 16 : Outlier Detection in Daily Sales*



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

- *Image 17 : Outlier Detection in Weekly Sales*



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

Product Categorization

Previous Product Categorization

The original dataset contained over **5,000 unique product descriptions**, and the last team categorized them into **5 predefined groups**: Lights & Decorations, Fashion & Accessories, Home & Storage, Kitchen & Dining, and Toys & Gifts. Then classify the product using OpenAI's GPT-3.5 model with **LangChain**, using the **prompt**:

> *Which category does '{description}' belong to: Lights & Decorations, Fashion & Accessories, Home & Storage, Kitchen & Dining, Toys & Gifts? Please give me the category name only.*

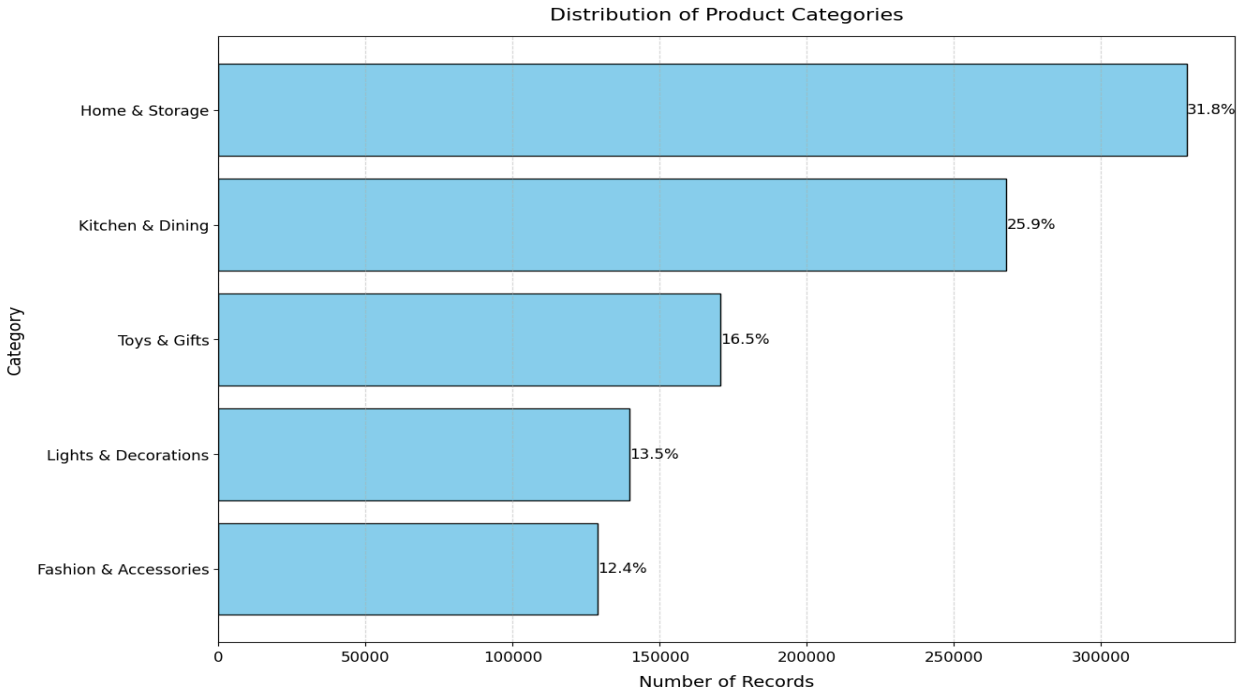
Once the category was completed, they analyzed the results by checking the count of each category. There seemed to be an issue with LLM's classification, since they found that there were **3 more categories in the result that contained a total of five descriptions**, and they manually reassigned these descriptions to the Home & Storage category, with unclear explanations.

As indicated in the last team's report: Toys & Gifts and Lights & Decorations categories' data are non-stationary, while the remaining categories' data are stationary.

- The final distribution of unique descriptions in their 5 categories is as follows:

Category	Home & Storage	Fashion & Accessories	Kitchen & Dining	Toys & Gifts	Lights & Decorations
Count	1883	1052	1050	857	856

- The distribution of each category is shown in the figure below:



Our New Product Categorization Method

1. Methodology

To group products based on their sales behaviors, we explored and compared three different clustering methods:

- K-Means Clustering
- Agglomerative Hierarchical Clustering
- DBSCAN Clustering

The comparison was based on silhouette scores across different numbers of clusters, evaluating the quality of clustering structure. Below is a visualization of the silhouette score comparison across different methods and cluster numbers:

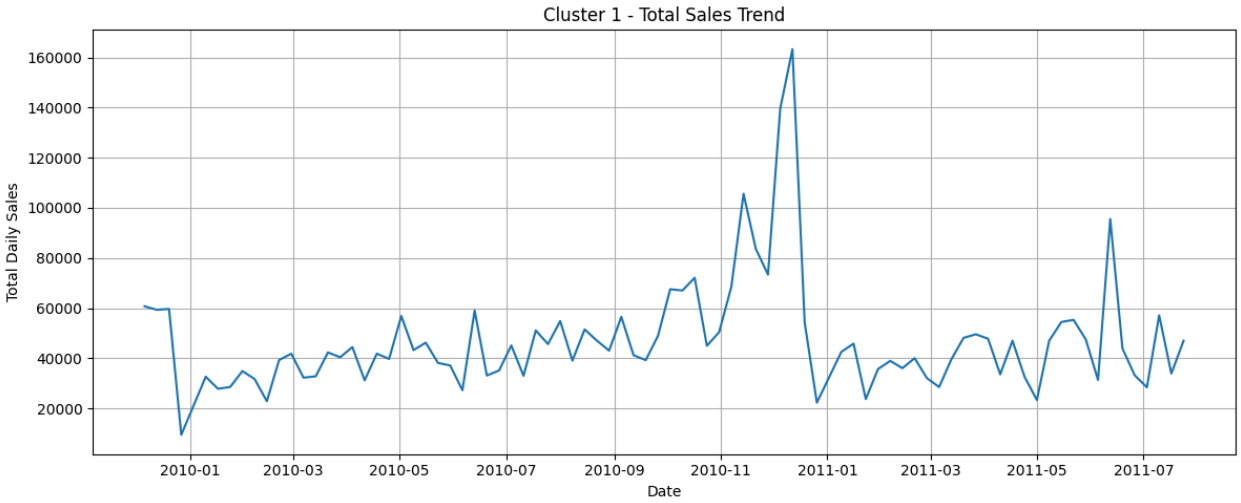
```
            method   k  silhouette
0           kmeans   2    0.834409
9   agglomerative   2    0.833363
11  agglomerative   4    0.816141
10  agglomerative   3    0.816091
1           kmeans   3    0.792991
18          dbscan   2    0.781888
2           kmeans   4    0.692337
3           kmeans   5    0.688980
7           kmeans   9    0.643391
8           kmeans  10    0.641955
4           kmeans   6    0.615573
5           kmeans   7    0.615563
6           kmeans   8    0.599671
14  agglomerative   7    0.530514
13  agglomerative   6    0.530501
12  agglomerative   5    0.529487
17  agglomerative  10    0.524521
16  agglomerative   9    0.523955
15  agglomerative   8    0.523885
```

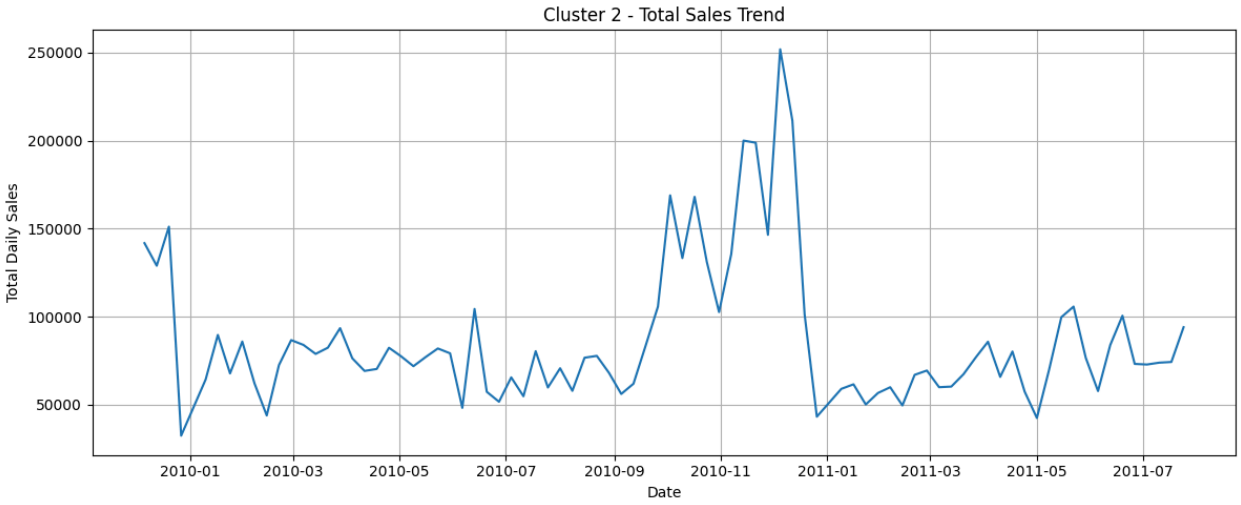
2. Results

After analysis, K-Means clustering with 2 clusters was selected as the final categorization method. It achieved a balance between a high silhouette score and practical interpretability of product sales patterns. Specifically, K-Means was able to clearly separate products into two groups, each showing distinct sales trends over time.

- Products were successfully divided into two groups with differentiated sales dynamics.
- Visual inspection confirmed that the two clusters exhibited distinct seasonality, volatility, and overall sales levels.

Cluster	Count
Cluster 1	130
Cluster 2	4673





This clustering-based categorization was subsequently used to train separate forecasting models for each group, improving overall prediction accuracy.

Training & Testing Split

1. For XGBoost model, LightGBM 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 retail sales. The model selection process involved optimizing hyperparameters and evaluating different configurations based on forecast accuracy.

Previous Model Design

The previous team employed a structured approach to developing a sales forecasting model involving data preprocessing, mode training, selection, and evaluation. Since some categories of datasets are non-stationary, they applied appropriate modeling techniques that can handle both trend variations and seasonal fluctuations.

First, they performed a train-test split to divide the dataset into training and testing sets. They then applied cross-validation and conducted hyperparameter tuning to optimize model performance by selecting the best set of parameters for each forecasting approach.

They utilized Generalized Linear Models (GLM) and Facebook Prophet as primary forecasting techniques for modeling. However, GLM struggles with non-stationary data, so they implemented a Long Short-Term Memory (LSTM) neural network for these non-stationary datasets. Thus, they assigned 2 model combinations: Prophet vs. GLM/LSTM. By comparing the MAPE values of GLM, Prophet, and LSTM, they determined the most suitable model for final deployment.

Our Model Design

Model 1: XGBoost

XGBoost (Extreme Gradient Boosting) is a tree-based ensemble learning method that builds additive models in a forward, stage-wise fashion. Its capability to capture complex feature interactions makes it particularly suitable for sales forecasting tasks involving time series with irregular trends or seasonal fluctuations.

XGBoost was chosen as one of the forecasting algorithms due to its strong performance in modeling non-linear temporal patterns, robustness against overfitting, and computational efficiency.

To prepare the input features for XGBoost, the following variables were constructed:

- Lag Features:
  Seven lagged sales variables (`lag_1` to `lag_7`) were created to capture short-term temporal dependencies.
- Calendar Features:
  - Day of the week
  - Month of the year
  - Day of the month

These features allow the model to learn from both past sales values and seasonal calendar effects (e.g., weekly seasonality, monthly trends).

Hyperparameters such as `n_estimators`, `learning_rate`, and `max_depth` were optimized using a grid search strategy based on validation set performance (measured by MAPE). After tuning, the best-performing model configuration was used to retrain on the combined training and validation sets, and final evaluation was performed on the hold-out test set.

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:

```
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.

Please 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. In this project, we focused on using prophet to predict the weekly total sales and also tried K Means to do clustering based on the time series data of products before training.

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**:
    a. **Components:** For the weekly total sale, we tried combinations of daily, weekly, monthly and yearly components for the prophet. We actually can incorporate all these components together in one model and search for the optimal 'fourier order' for each

component. It will be hard to manually determine which seasonality components should be included in the model. Our choice is to add the binary logic gate such as 'add_weekly', 'add_monthly' in the parameter grid. By this means, the model will try with and without the corresponding seasonality components and finalize the choice by comparing in the validation set. This method is more reasonable and won't be disturbed by the subjective judgement

   b. **fourier_order** : 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.
   c. **seasonality_prior_scale:** We also tune 'seasonality_prior_scale' which similarly adjusts the extent to which the seasonality model will fit the data.
   d. **seasonality_mode**: There are two kinds of mode: 'additive', 'multiplicative' . 'Additive' represents that seasonal effects are added directly to the trend component. 'Multiplicative' means that seasonal effects are applied as a proportion of the trend. There are two different ways to model the time series data.

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 also tune "changepoint_prior_scale" to avoid overfitting or underfitting.
3. **Holiday**: In prophet, we can also add a holiday component. Since the context of the dataset is in England (UK), we can conveniently use the built-in "holidays=uk_holidays" parameter when creating the instance of the prophet. However we also want to see if the holiday component can really improve the performance, we also include an ablation study here.

Model 4: LightGBM

Light Gradient Boosting Machine is a highly efficient gradient boosting framework developed by Microsoft. It is designed to be fast, scalable, and memory-efficient, especially for large datasets or datasets with many features. Gradient Boosting is a technique for both regression and classification problems. It builds an ensemble of weak prediction models, typically decision trees, in stages. Each new tree is trained to correct the errors made by the previous ensemble, leading to progressively better performance. Compared to other gradient boosting implementations like XGBoost, LightGBM grows trees leaf-wise, choosing the leaf with the maximum split gain to grow, whereas others grows trees level-wise, expanding all leaves at a given depth before going deeper. LightGBM's leaf-wise strategy allows it to converge faster and achieve better accuracy. LightGBM is also faster and more memory efficient than most alternatives when handling large datasets.

Model Input: To use LightGBM for time series we can create additional features as follows: lag features: values of the target at previous time steps. It is important to select appropriate lags based on the periodicity and behavior of the series such as 1 day ago, 7 days ago. This process requires familiarity with the data and domain knowledge. calendar-based features: current month, day of week, is_weekend, etc that help the model understand the cyclical nature of the time series. target: the value we are predicting.

Model Parameters (source):

LightGBM contains parameters commonly found in traditional tree-based and gradient boosting models, along with unique parameters such as feature_fraction, bagging_fraction, and num_leaves that specifically enhance its efficiency and prevent overfitting.

- num_leaves: Maximum tree leaves for base learners.
- learning_rate: Boosting learning rate. You can shrink the feature weights to make the boosting process more conservative."
- feature_fraction: LightGBM will randomly select part of features on each iteration (tree) if feature_fraction is smaller than 1.0. It is used to prevent overfitting.
- bagging_fraction: Like feature_fraction, but this will randomly select part of data without resampling.
- bagging_freq: Frequency for bagging. 0 means no bagging. k means perform bagging at every k iteration.
- min_data_in_leaf: Minimal number of data in one leaf. It is a very important parameter to deal with overfitting.
- max_depth: Limit the max depth for tree model. This is used to deal with overfitting when it is set.

Comparisons of Models

Model	**Pros**	**Cons**
XGBoost	- Efficient and scalable for large datasets - Handles missing values and various feature types well - Capable of modeling complex nonlinear relationships - Provides good regularization (L1 & L2) to prevent overfitting	- Requires careful hyperparameter tuning for optimal performance - May not naturally capture sequential or temporal patterns without engineered lag features - Can be sensitive to noise if not tuned properly
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
LightGBM	-Fast to train	-Tree based structure not ideal for handling sequential patterns naturally.

	-Can handle complex non linear pattern if features are derived properly	- Deriving new features requires domain knowledge and can easily lead to bias.

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: The aggregated sales amount across all products, grouped by week. This variable captures the overall business performance trend over time and serves as the primary variable to forecast.

Previous Categorization Results:

GLM/ LSTM & Prophet

The previous team compared the forecasting accuracy of Facebook Prophet and a GLM/LSTM hybrid model on five different product categories using **Test MAPE** as the evaluation metric.

Category	Home & Storage	Fashion & Accessories	Kitchen & Dining	Toy & Gifts	Lights & Decorations
Prophet	45.69%	45.99%	72.94%	40.37%	34.60%
GLM/LSTM	31.12%	36.68%	33.05%	36.06%	37.63%

Based on the MAPE score on the test set, the GLM model for the stationary category dataset and the LSTM model for the non-stationary category dataset have a lower score than the Facebook Prophet model on all four product categories, except Lights & Decorations, where lower MAPE values mean better forecasting.

Chronos

We aimed to test whether applying more advanced forecasting models alone could improve accuracy, without modifying the original data preparation. So, we evaluated the model performance using the same cleaned data and product categorization as used by the previous team. The table below summarizes the performance comparison:

Category	**Home & Storage**	**Fashion & Accessories**	**Kitchen & Dining**	**Lights & Decorations**	**Toys & Gifts**	**Weighted MAPE**	**Improvement**
Percentage	31.8000%	12.4000%	25.9000%	13.5000%	16.5000%		
PREVIOUS	**31.1200%**	**36.6800%**	**33.0500%**	**36.0600%**	**37.6300%**	**34.0815%**	

zero-shot bolt_tiny	32.9582%	32.5504%	32.1771%	**35.5352%**	39.0039%		
zero-shot bolt_mini	32.2490%	30.6645%	32.2588%	39.6002%	37.5858%		
zero-shot bolt_small	32.0139%	31.1872%	30.0368%	37.7398%	**36.2885%**		
zero-shot bolt_base	**31.9659%**	**30.5643%**	**29.9891%**	39.2078%	37.2806%		
fine-tuned best	**29.5886%**	**30.5643%**	**28.6320%**	**32.8461%**	**34.3786%**	**30.7215%**	**3.3600%**

* Other details could be found in the *Model 2 Results: Chronos section 2.1*.

Compared to the previous team's model, fine-tuned Chronos models achieved a moderate reduction in weighted MAPE (around 3.36% improvement). However, despite this tuning effort, all categories still suffered from relatively high error rates. After analysis, we believe that this may be caused by the following two reasons:

- **Insufficient data cleaning:** The raw dataset contained many inconsistencies, missing values, and noisy records that were not properly handled in the previous team's setup.
- **Suboptimal product categorization:** The original clustering did not adequately reflect products' sales patterns or seasonality, leading to poor group-level modeling.

As a result, simply changing the forecasting model could not fully address the fundamental issues in the dataset and categorization that the previous team used. Therefore, we decided to revisit data preprocessing and redesign product categorization, in order to achieve more substantial improvements in forecasting accuracy.

Our Categorization Results:

Model 1 Results: XGBoost

1. Model Selection and Training Setup

We applied the XGBoost algorithm to forecast the aggregated weekly product sales across different product clusters. Products were initially clustered into two groups using KMeans based on their historical weekly sales patterns. The model was trained using features including lagged sales values (lags 1 to 7 weeks), day-of-week, month, and day indicators. We performed parameter tuning based on a validation split (60% train, 20% validation, 20% test) to identify the best hyperparameters.

The parameter grid included variations in:

- `n_estimators`: 100, 200, 300
- `learning_rate`: 0.1, 0.05, 0.01
- `max_depth`: 3, 5, 10

The best configuration selected based on validation MAPE was:

```
{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 3}
```

2. Overall Model Performance

After identifying the best hyperparameters, we retrained the model using the full training + validation data and evaluated on the hold-out test set.

The best global MAPE on the test set was: 17.97%.

Additionally, performance was assessed separately across three equal-length periods within the test set to observe temporal changes in error:

Period	MAPE (%)
Period I	9.83%
Period II	23.11%
Period III	20.58%

To better illustrate the effectiveness of our approach, the table below compares the overall forecasting performance with the previous team's model and our two versions of XGBoost models:

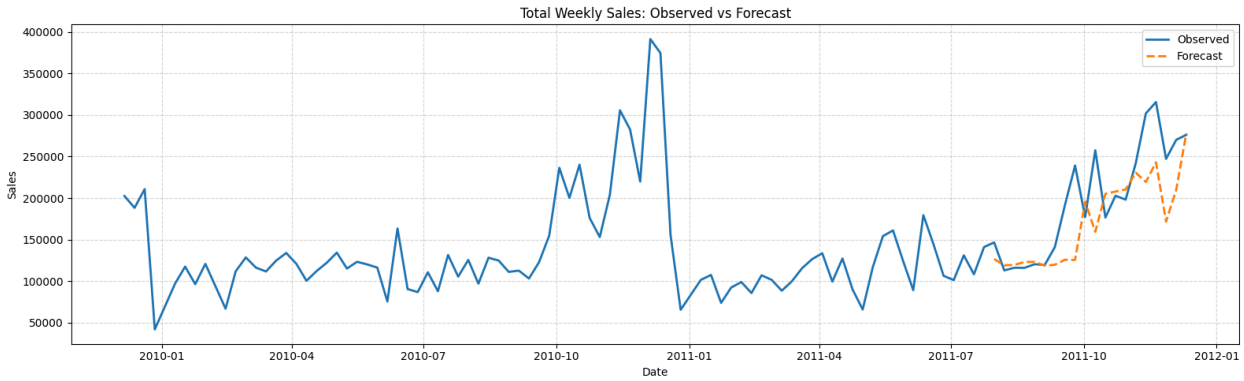
PREVIOUS Weighted MAPE	**34.0815%**	**Improvement**
XGBoost (Kmeans: k=2 & Fine-tuned)	**17.97%**	**16.1115%**
XGBoost (Kmeans: k=2)	19.49%	14.5915%

As shown, both versions of our XGBoost models achieved substantial improvements over the previous team's results, with fine-tuning further reducing the overall MAPE by around 1.5% compared to the basic model. This highlights the effectiveness of our improved data cleaning, product categorization, and hyperparameter optimization steps.

3. Visualization of Best Model Performance:

3.1 Actual vs. Forecasted Usage

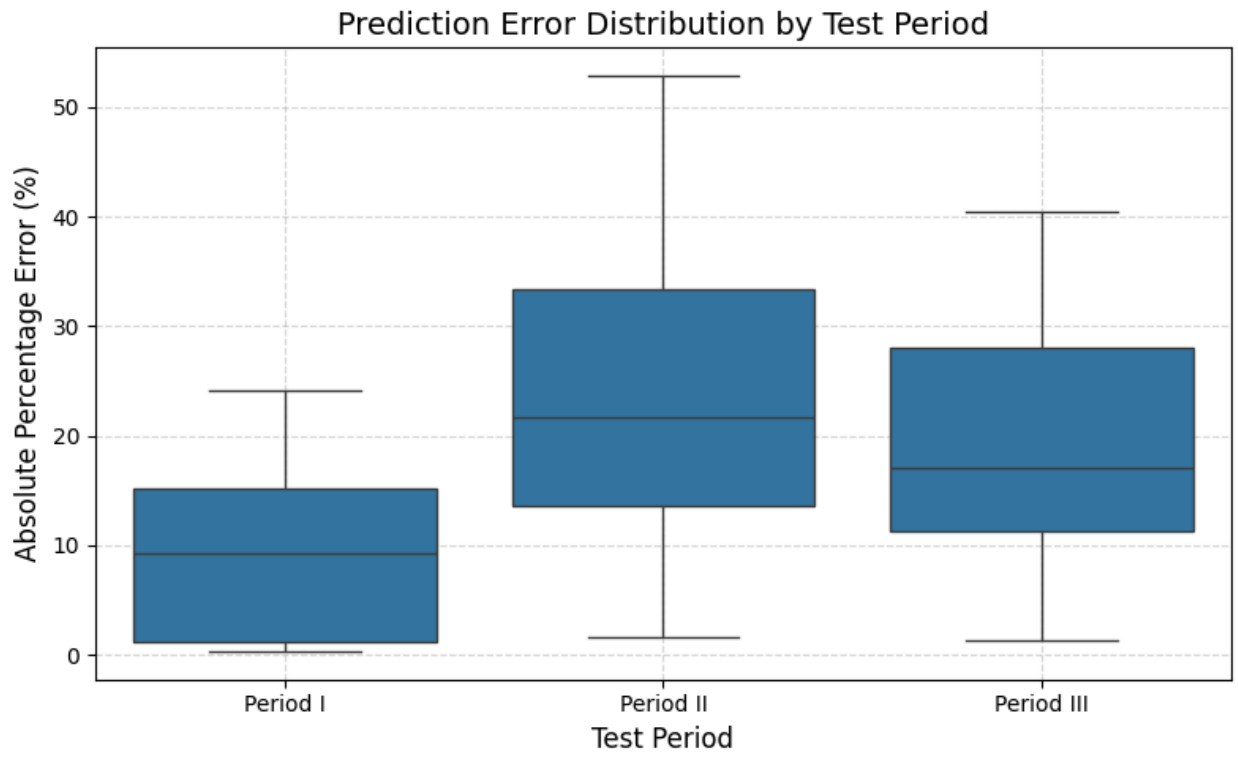
The following plot shows the true total weekly sales across the full time range (training + testing) compared against model forecasts on the test set:



- The blue line represents the observed (true) weekly sales.
- The orange line represents the forecasted sales during the test period.
- Overall, the model captures the general increasing trend but tends to slightly underpredict peak sales weeks.

3.2 Error Distribution Across Test Periods

To further analyze model performance over time, we visualized the distribution of prediction errors across the three test periods:



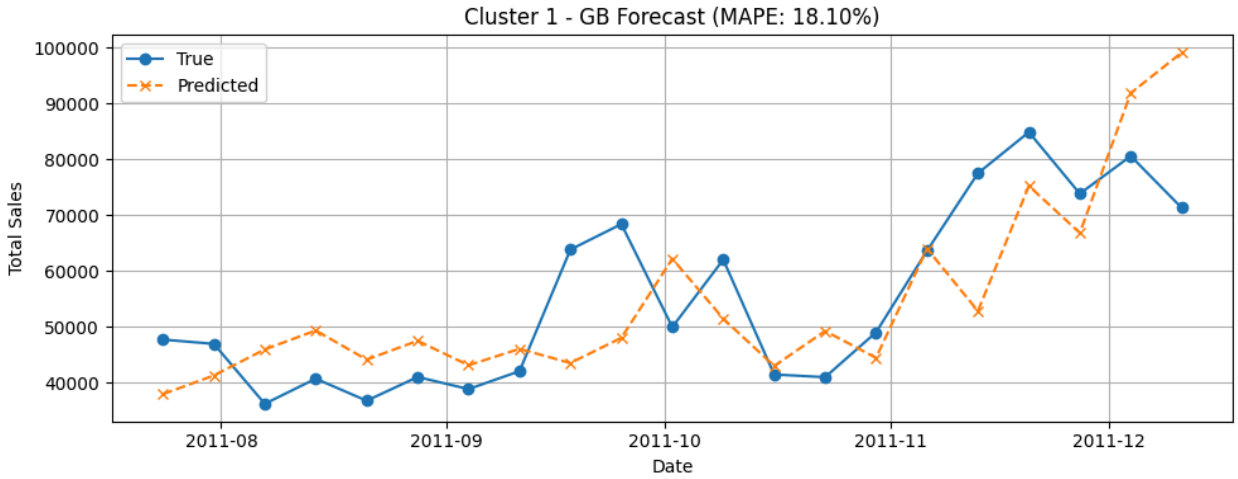
- Period I exhibited the lowest error dispersion, indicating the model fit better at the start of the test set.
- Periods II and III showed greater variability, likely reflecting increased volatility in product sales.

4. Visualization of Other Model Performance

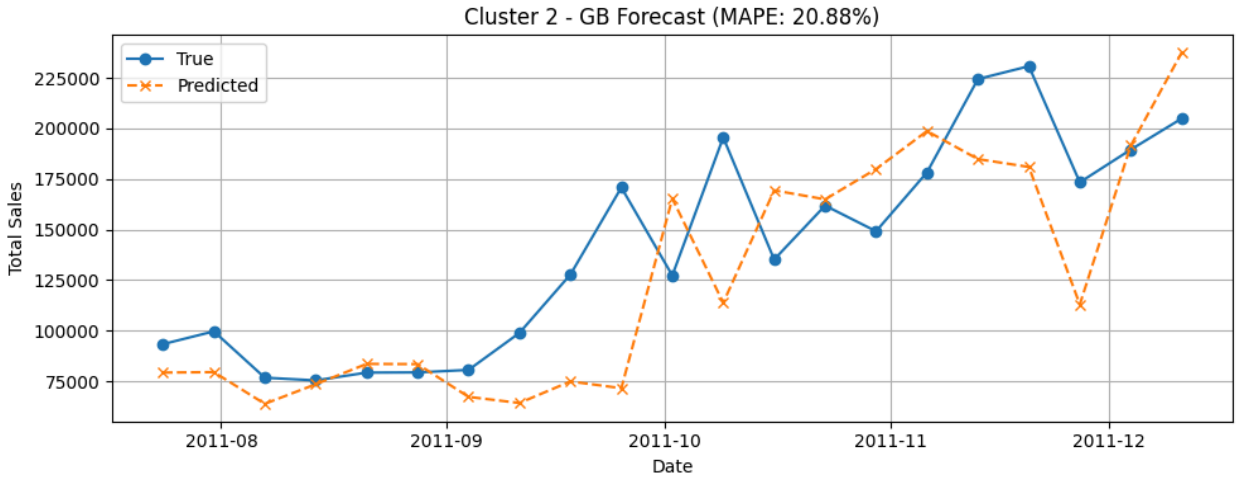
In addition to the tuned XGBoost model, we also evaluated a baseline forecasting setup based on the 2-category KMeans-based product categorization without further fine-tuning.

The plot below shows the aggregated true vs. predicted total weekly sales using the non-fine-tuned models:

- **Cluster 1** (MAPE = **18.10%**)



- **Cluster 2** (MAPE = **20.88%**)



As seen in the plots, the model captures the overall upward trends reasonably well, though some short-term fluctuations are less accurately predicted without fine-tuning. The overall test MAPE across both clusters is **19.49%**, serving as a baseline for comparison against the fine-tuned model.

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.

Firstly, we use Chronos to do forecasting in the same setting as the previous team's data processing procedure and product categorization method, trying to compare with the previous team directly by changing a new model.

Secondly, we use Chronos to do forecasting on the data after data cleaning, resampling the data to predict weekly sales.

Finally, we also use Chronos to do forecasting on the same cleaned data to predict daily sales.

1. Model Setup and Selection

- **Zero-shot forecasting**: use Chronos **bolt_mini, bolt_tiny, bolt_small, and bolt_base** preset models: as explained in Model Design section above, only prediction_length, train_data, and model preset are needed, no more parameters;
- **Fine-tuning**: fine-tune Chronos **bolt_mini, bolt_tiny, bolt_small, and bolt_base** by **manually-defined** hyperparameters for **600-700 seconds.** Take bolt_tiny as an example:

```
{"model_path": "bolt_tiny",
"fine_tune": True,
"fine_tune_lr": space.Real(0.000001, 0.001, log=True),
"fine_tune_steps": space.Categorical(125, 250, 500, 1000),
"fine_tune_batch_size": space.Categorical(16, 32, 64),}
```

2. Overall Model Performance

2.1 Forecasting with previous team's data & categorization setting

Category	Home & Storage	Fashion & Accessories	Kitchen & Dining	Lights & Decorations	Toys & Gifts	Weighted MAPE	Improvement
Percentage	31.8000%	12.4000%	25.9000%	13.5000%	16.5000%		
PREVIOUS	**31.1200%**	**36.6800%**	**33.0500%**	**36.0600%**	**37.6300%**	**34.0815%**	
zero-shot bolt_tiny	32.9582%	32.5504%	32.1771%	**35.5352%**	39.0039%		
zero-shot bolt_mini	32.2490%	30.6645%	32.2588%	39.6002%	37.5858%		
zero-shot	32.0139%	31.1872%	30.0368%	37.7398%	**36.2885%**		

bolt_small							
zero-shot bolt_base	**31.9659%**	**30.5643%**	**29.9891%**	39.2078%	37.2806%		
fine-tuned best *	**29.5886%**	**30.5643%**	**28.6320%**	**32.8461%**	**34.3786%**	**30.7215%**	**3.3600%**

*: Here fine-tuned best means fine-tuning using the best zero-shot model preset, for example, since zero-shot bolt_base achieves the best MAPE for Home & Storage category's data, we still fine-tune using bolt_base for this category's data.

We see a slight increase in MAPE after both zero-shot and fine-tuned forecasting using Chronos compared with the previous team's model results. But as pointed out before in the data cleaning section, since the previous team didn't do necessary data cleaning, the data they used for forecasting contains lots of useless information and noise, which seriously impacts the forecasting accuracy! Let's see what will happen after we do forecasting using the cleaned data.

2.2 Forecasting using cleaned data, weekly sales prediction

PREVIOUS Weighted MAPE	**34.0815%**	**Improvement**
zero-shot bolt_tiny	**28.7973%**	**5.2842%**
zero-shot bolt_mini	24.4255%	**9.6560%**
zero-shot bolt_small	19.1217%	**14.9598%**
zero-shot bolt_base	**14.2683%**	**19.8132%**
fine-tuned bolt_tiny	28.5000%	**5.5815%**
fine-tuned bolt_mini	24.0416%	**10.0399%**
fine-tuned bolt_small	17.5394%	**16.5421%**
fine-tuned bolt_base	14.2683%	**19.8132%**

- We see performance improvement for all Chronos models compared with the previous team's result.
- **Best-performing model**: Chronos[bolt_base] achieves the lowest overall MAPE of 14.2683%, with a MAPE improvement of 19.8132%.
- **Worst-performing model**: Chronos[bolt_tiny] shows the highest MAPE of 28.7973%, with a MAPE improvement of 5.2842%.

2.3 Forecasting using cleaned data, daily sales prediction

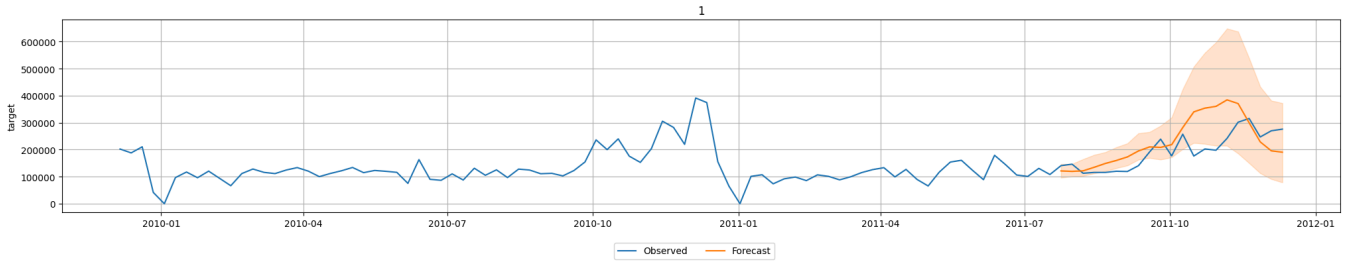
PREVIOUS Weighted MAPE	**34.0815%**	**Improvement**
zero-shot bolt_tiny	**29.4251%**	**4.6564%**
zero-shot bolt_mini	29.0504%	**5.0311%**
zero-shot bolt_small	28.8359%	**5.2456%**

zero-shot bolt_base	27.9666%	**6.1149%**
fine-tuned bolt_tiny	27.4432%	**6.6383%**
fine-tuned bolt_mini	**24.9931%**	**9.0884%**
fine-tuned bolt_small	26.9026%	**7.1789%**
fine-tuned bolt_base	26.5394%	**7.5421%**

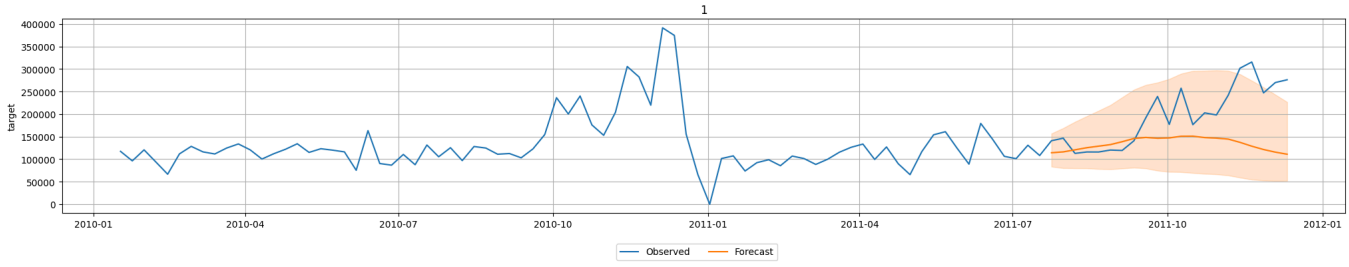
- Again, we see performance improvement for all Chronos models compared with the previous team's result.
- **Best-performing model**: ChronosFineTuned[bolt_mini] achieves the lowest overall MAPE of 24.9931%, with a MAPE improvement of 9.0884%.
- **Worst-performing model**: Chronos[bolt_tiny] shows the highest MAPE of 29.4251%, with a MAPE improvement of 4.6564%.

3. Visualization of Model Performance for Weekly sales forecasting using cleaned data

3.1 Best Model Actual vs. Forecasted Sales

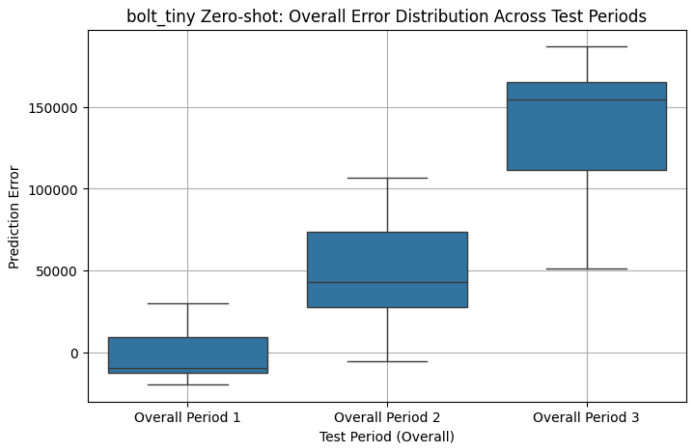
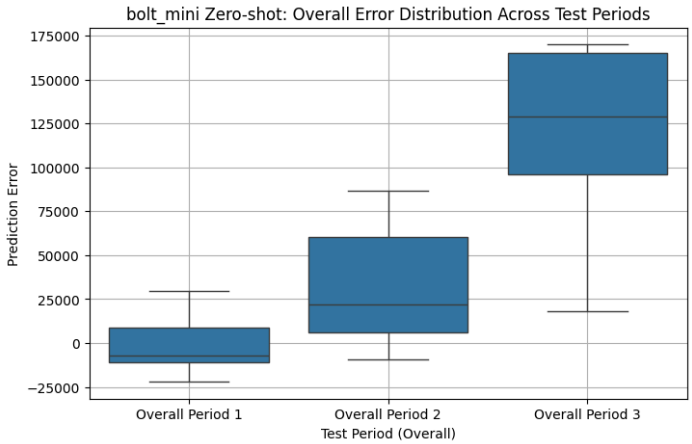
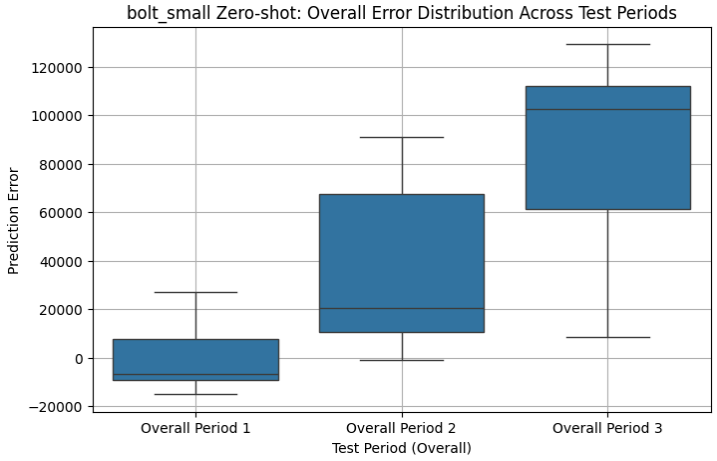


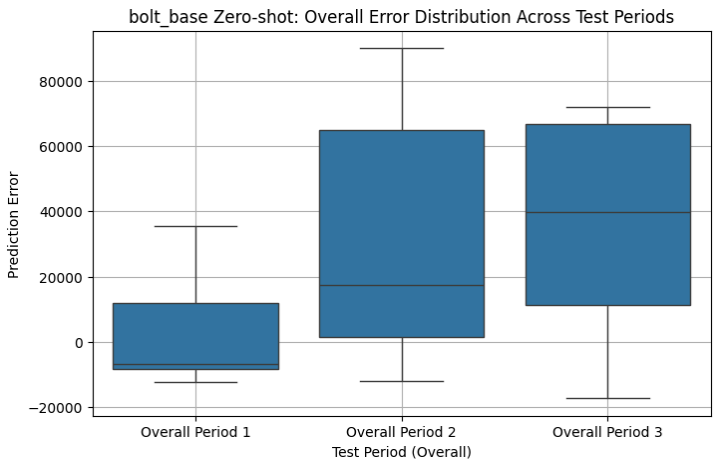
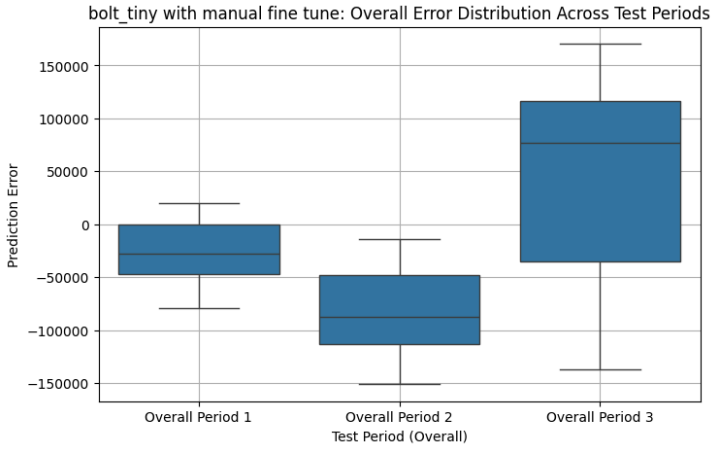
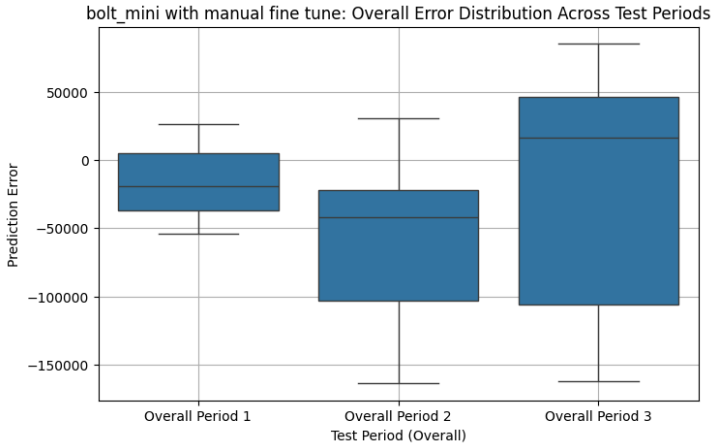
3.2 Worst Model Actual vs. Forecasted Sales

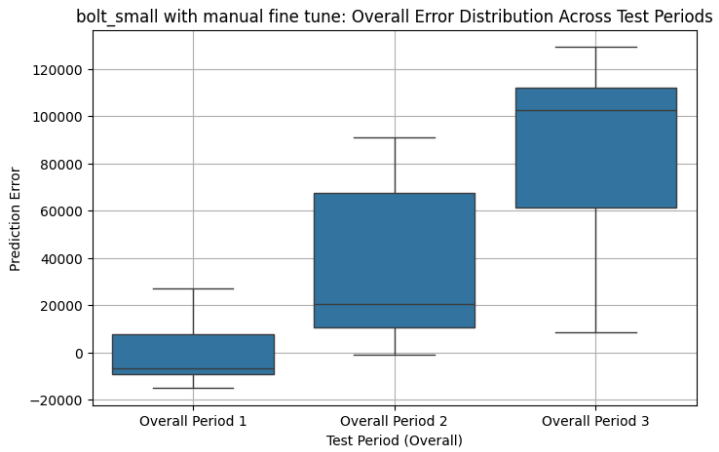
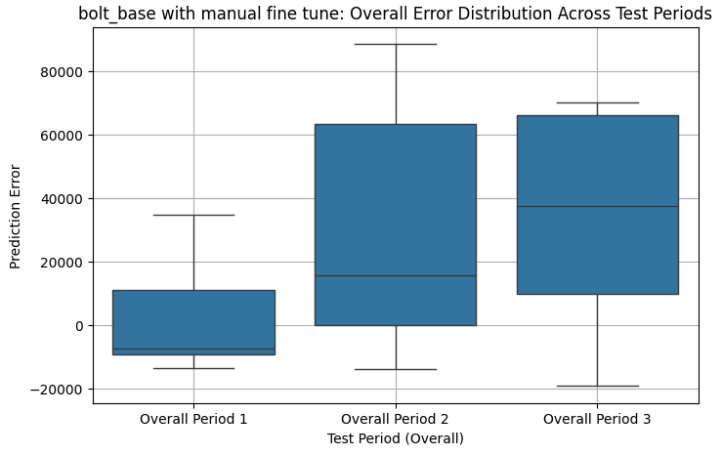


3.3 Error Distribution Across Test Periods

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

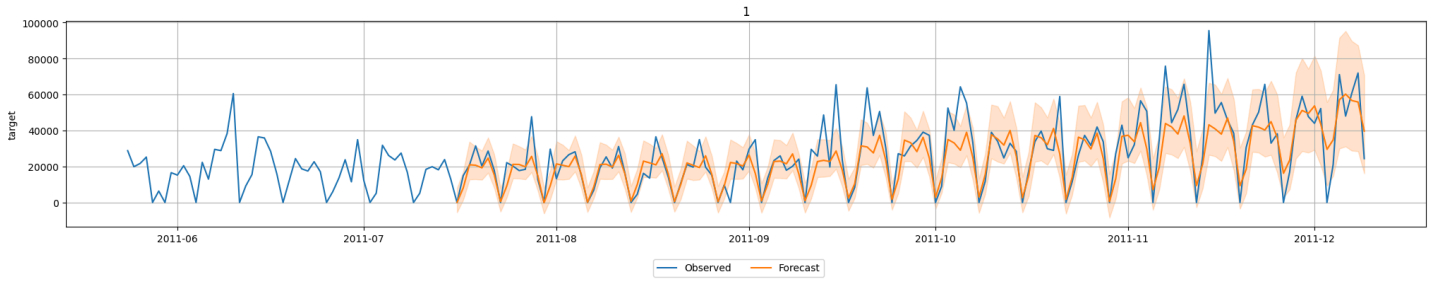
worst model: zero-shot bolt_tiny		13.22%, 23.53%, 49.65%
zero-shot bolt_mini		12.53%, 17.18%, 43.57%
zero-shot bolt_small		10.22%, 17.01%, 30.13%

best model: zero-shot bolt_base		11.59%, 16.04%, 15.18%
fine-tuned bolt_tiny		30.88%, 45.02%, 40.83%
fine-tuned bolt_mini		23.39%, 37.45%, 33.55%

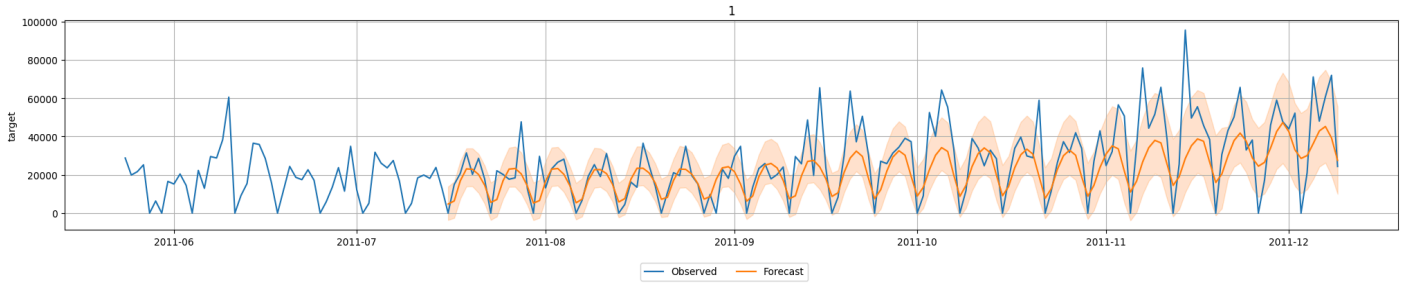
fine-tuned bolt_small		10.22%, 17.01%, 30.13%
fine-tuned bolt_base		12.08%, 15.70%, 15.11%

4. Visualization of Model Performance for Daily sales forecasting using cleaned data

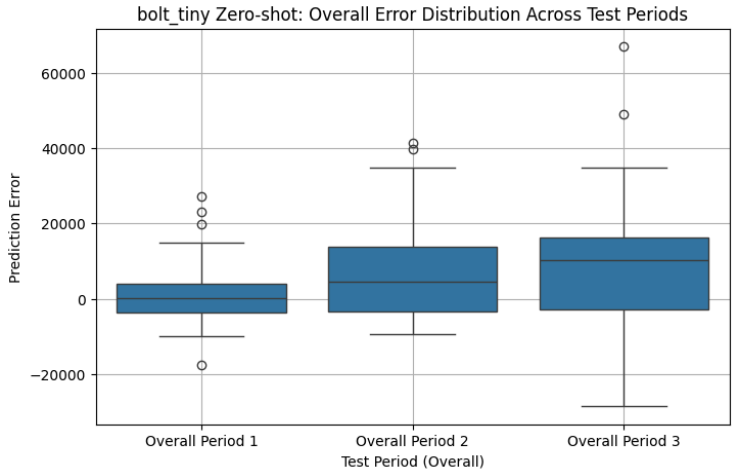
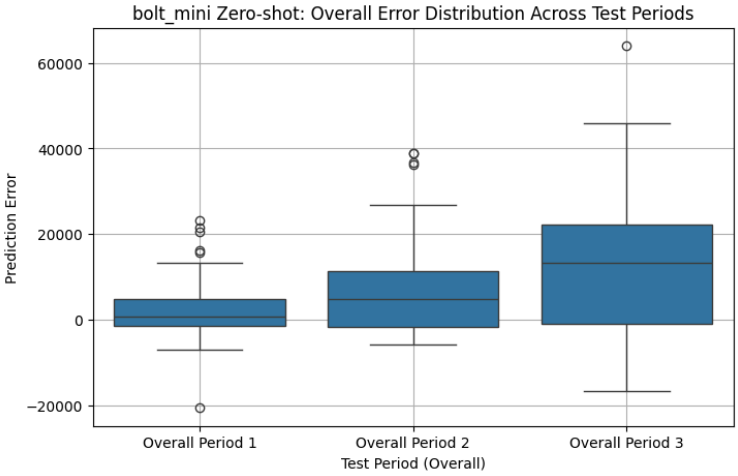
4.1 Best Model Actual vs. Forecasted Sales

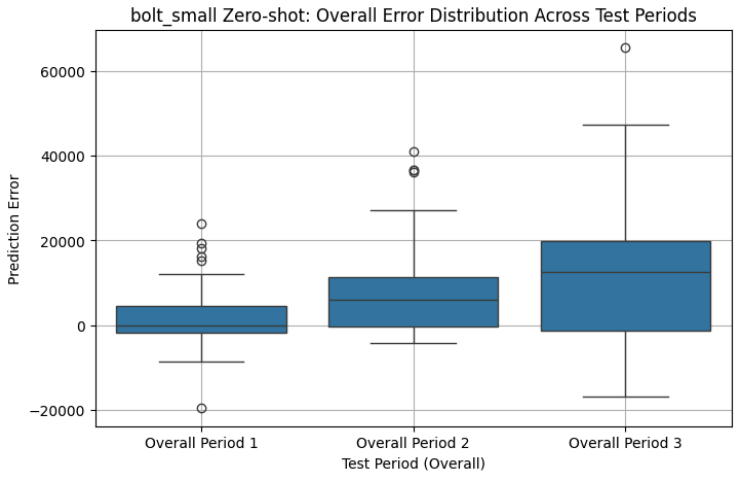
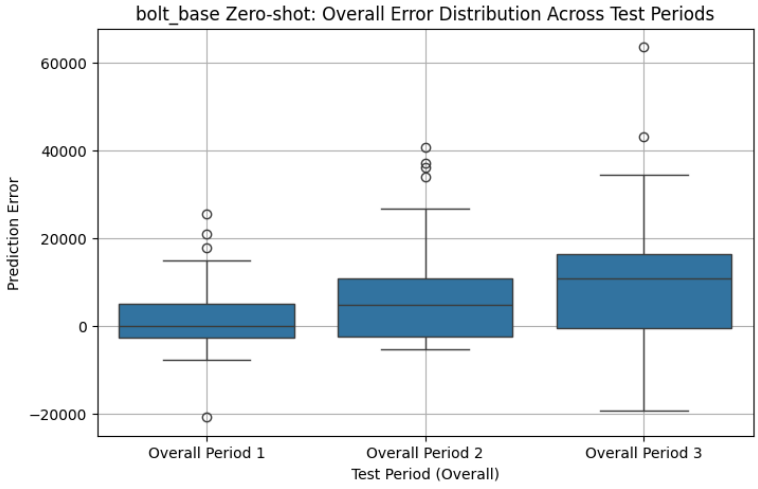
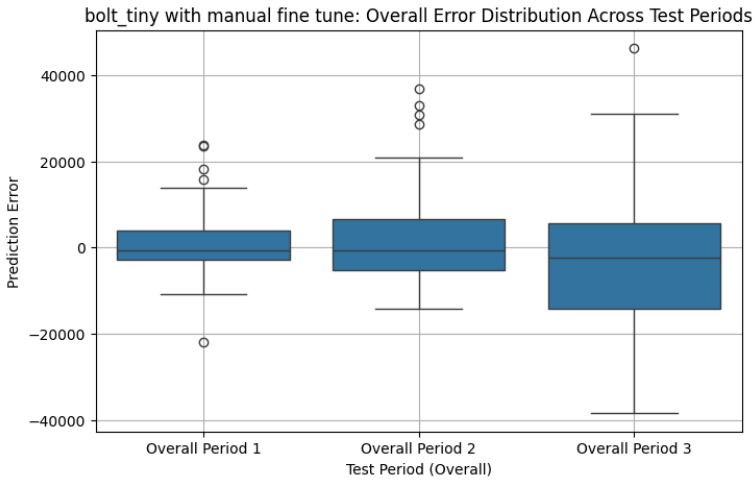


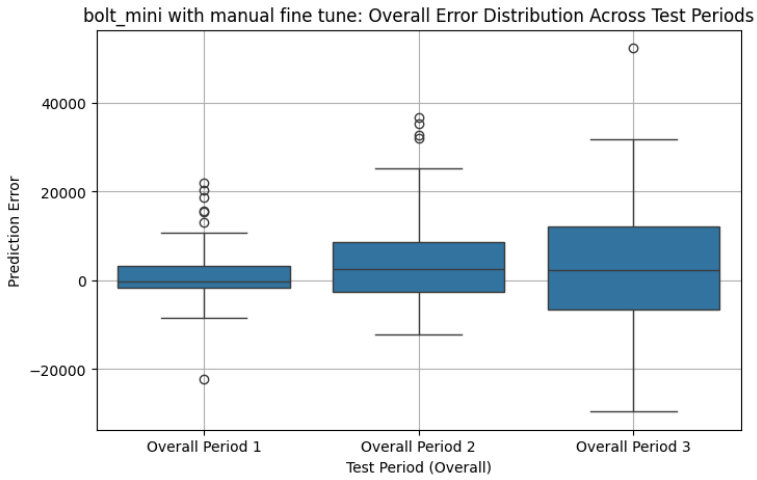
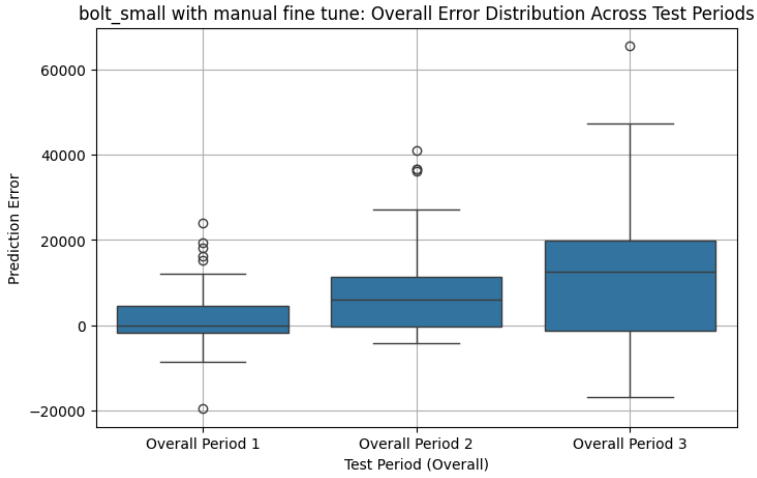
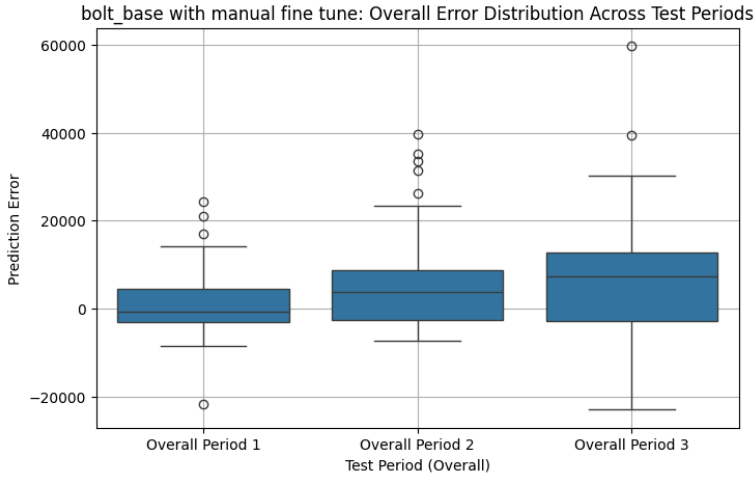
4.2 Worst Model Actual vs. Forecasted Sales



4.3 Error Distribution Across Test Periods

Model	Error Distribution Plot	Overall MAPE for Period 1, 2, 3
worst model: zero-shot bolt_tiny		25.80%, 31.73%, 30.66%
zero-shot bolt_mini		24.25%, 29.90%, 32.89%

zero-shot bolt_small		24.65%, 28.99%, 32.77%
zero-shot bolt_base		25.02%, 29.17%, 29.64%
fine-tuned bolt_tiny		25.39%, 28.94%, 27.95%

best model: fine-tuned bolt_mini		24.19%, 26.49%, 24.27%
fine-tuned bolt_small		24.65%, 28.99%, 32.77%
fine-tuned bolt_base		25.67%, 27.12%, 26.81%

5. Additional Comment for this model

Since the sales data is aggregated on a daily/weekly basis, the number of data points available is relatively small – 739 and 106 respectively, with 592 and 85 data as history context for the model to fit for prediction. Based on the official model usage tutorial and our previous related experience using Chronos, its performance usually worse without sufficient data – usually when we have 3000+ or even 10000+ data points available, the model will be better able to capture the intricate pattern of the time series and perform better, both in zero-shot forecasting and fine-tuned forecasting, especially for the fine-tuning part.

Because when we don't have enough data, fine-tuning the model will possibly lead to model overfitting, especially for the model with huge parameter size like Chronos Bolt_base and Bolt_small. Due to this reason, as we can see from the results above, fine-tuning the model will not necessarily lead to performance improvement.

Moreover, notice that Chronos performs probabilistic forecasting, and when the situation becomes unpredictable and the time series is volatile, it tends to give more conservative forecasting, as shown in the orange shadow region in the Observed vs. Forecast plot above. Such a forecasting scheme is actually not bad in practice, especially when we consider the retail sales, since the sales managing team can see what will possibly happen in both the worst and best case scenarios and correspondingly set up different marketing strategies and different plans to manage the inventory for different cases!

Finally, clearly there is seasonality in the retail data, as well as frequently appearing '0' sales date. Some models like Prophet can directly add 'holiday' to capture such patterns, as explained in the Prophet model section below. In comparison, Chronos does not natively support covariate analysis (e.g., add external variables' data like holidays and weather for model to learn), the **AG-TS** platform covariate analysis tool to be integrated with Chronos. Due to the time constraint, we haven't explored this approach to verify if the performance will improve.

Model 3 Results: Prophet

1. Model Setup and Training Setup

for all the scenarios, the same model setting is following:

```
param_grid = {
    'changepoint_prior_scale': [0.01, 0.1],
    'seasonality_prior_scale': [0.1, 1.0],
    'holidays_prior_scale': [0.1, 1.0],
    'seasonality_mode': ['additive', 'multiplicative'],
    'yearly_fourier': [5, 6, 7, 8, 9, 10],
    'weekly_fourier': [5,10],
    'monthly_fourier': [5,10],
    'use_holidays': [True, False],
    'add_weekly': [True, False],
```

```
    'add_monthly': [True, False],
}
```

The ‘add_weekly’ and ‘add_monthly’ parameters are controlling if we include the weekly component and monthly component or not, which will be decided in the validation set. The other parameters are what we talked about above. For simplicity, we assign [5,6,7,8,9,10] to yearly fourier order and [5,10] to both weekly and monthly. ‘use_holidays’ controls if we take UK holidays into consideration or not.

Changes Compared with the work of previous group:

1. Components: Previous group only includes ‘quarterly’ and ‘half yearly’ components with period = 90 and period = 180 respectively for the model. It is unreasonable not to test which seasonality components should be included. In our work, we initially include the daily, weekly, monthly and yearly components and we let the model determine if it should have each component or not in the validation set, which is like ablation study.
2. Parameter grid: For simplicity, we set less predefined parameters of ‘change_prior_scale’, ‘seasonality_prior_scale’ and ‘holidays_prior_scale’. However, we try to tune the fourier order of yearly, weekly and monthly components. The Fourier order determines the number of sine and cosine terms used to approximate each seasonal effect. Tuning the Fourier order is important because: Higher Fourier orders allow the model to capture more complex, high-frequency seasonal fluctuations. Lower Fourier orders produce smoother, simpler seasonal patterns and help prevent overfitting, especially when data is noisy or when the true seasonality is relatively simple.

2. Overall Model Performance

Forecasting using cleaned data, weekly sales prediction

PREVIOUS Weighted MAPE	**34.0815%**	**Improvement**
Prophet (No Clustering)	**10.65%**	**23.4315%**
Prophet (No Clustering and removing Outliers)	17.34%	16.7415%
Prophet (Clustering:Kmeans k = 2)	12.61%	21.4715%
Prophet (Clustering: Agglomerative (k=2))	20.80%	13.2814%

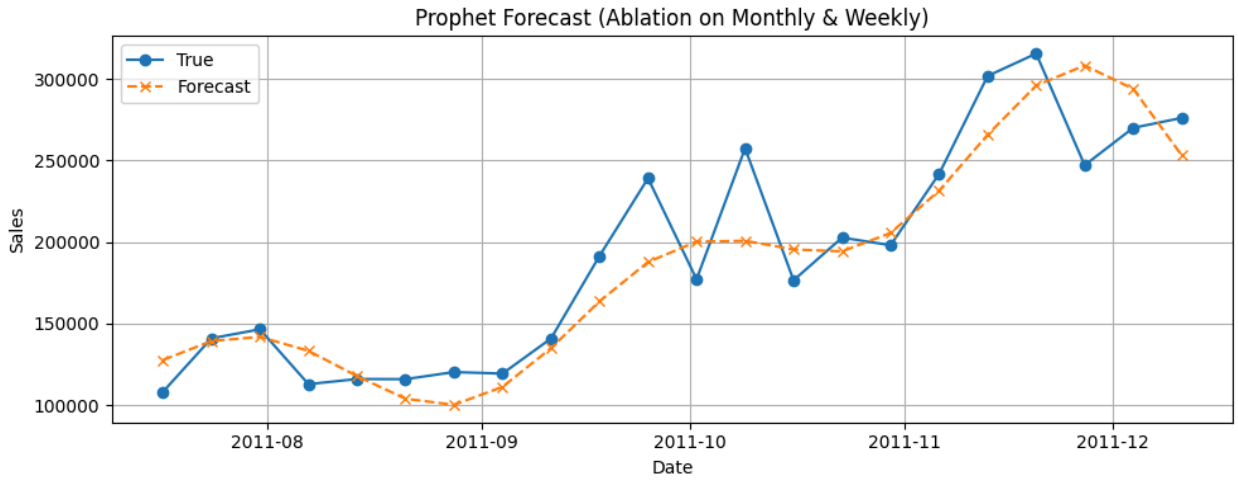
3. Visualization of Model Performance

3.1 Best Model Actual vs. Forecasted Sale

The best model is Prophet without any additional modification and the optimal hyperparameter is:

```
        Best Parameters Found:
changepoint_prior_scale        0.01
seasonality_prior_scale         1.0
holidays_prior_scale            1.0
seasonality_mode           additive
```

```
yearly_fourier                   6
weekly_fourier                  10
monthly_fourier                 10
use_holidays                 False
add_weekly                    True
add_monthly                  False
```



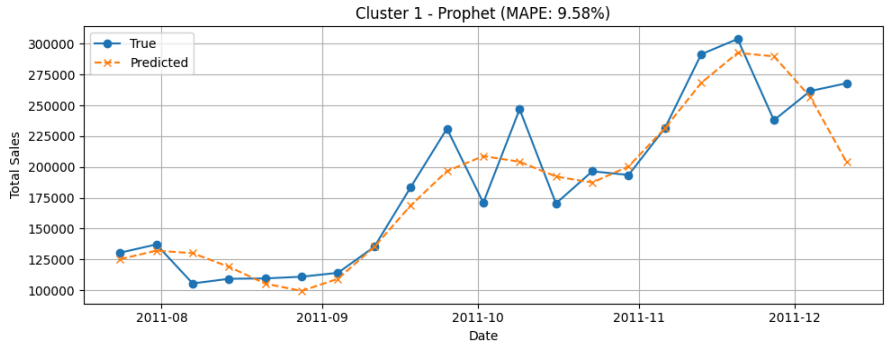
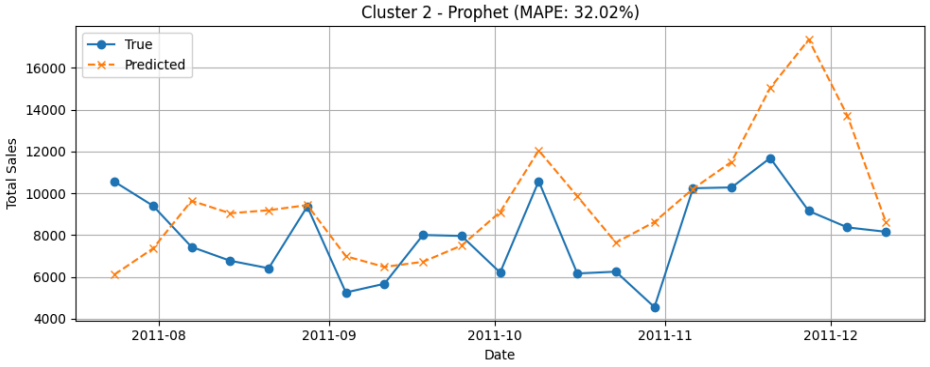
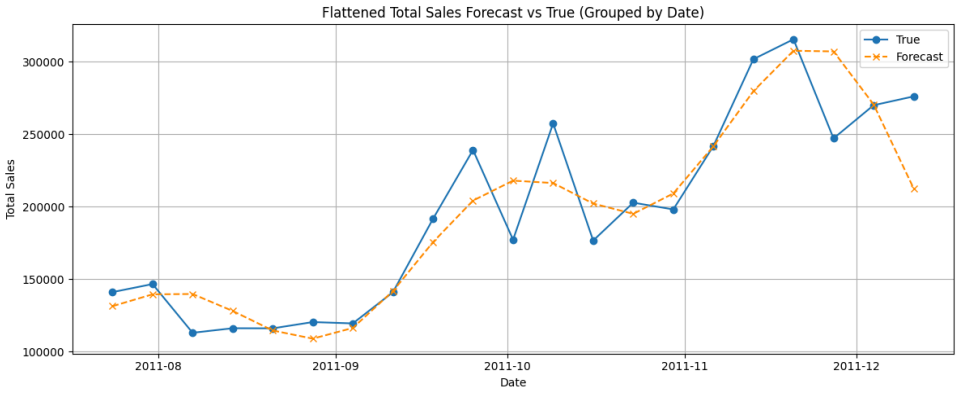
Total MAPE: 10.65 %

3.2 Worst Model Actual vs. Forecasted Sale

The worst model here is Prophet with Clustering using Agglomerative (k=2) . The corresponding hyperparameter is:

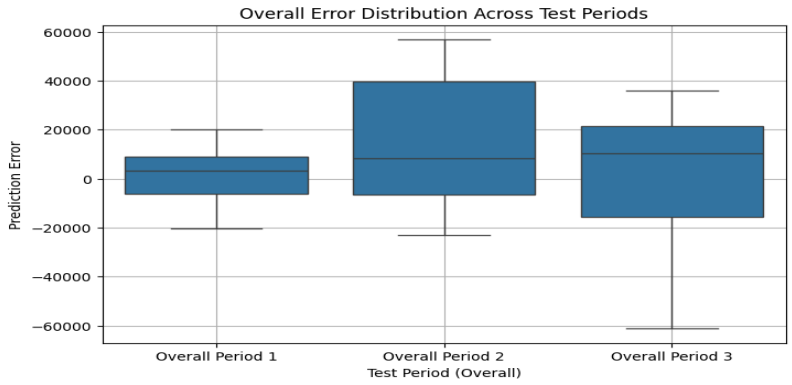
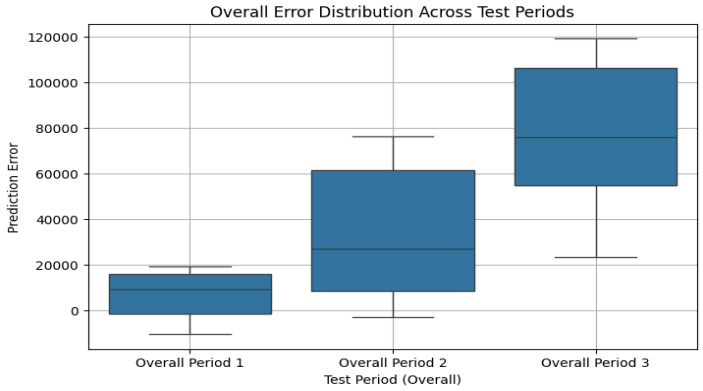
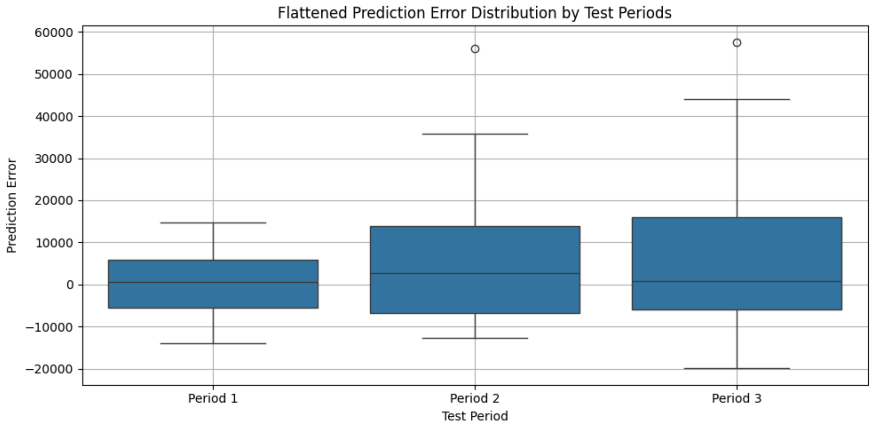
```
Best params for Cluster 1:
{    'changepoint_prior_scale': 0.01,
 'seasonality_prior_scale': 0.1,
  'holidays_prior_scale': 1.0,
 'seasonality_mode': 'additive',
      'yearly_fourier': 7,
     'weekly_fourier': 10,
     'monthly_fourier': 5,
     'use_holidays': True,
      'add_weekly': True,
     'add_monthly': False}
Best params for Cluster 2:
{'changepoint_prior_scale': 0.1,
 'seasonality_prior_scale': 1.0,
  'holidays_prior_scale': 0.1,
'seasonality_mode': 'multiplicative',
     'yearly_fourier': 10,
     'weekly_fourier': 10,
     'monthly_fourier': 5,
    'use_holidays': False,
      'add_weekly': True,
     'add_monthly': True}
```

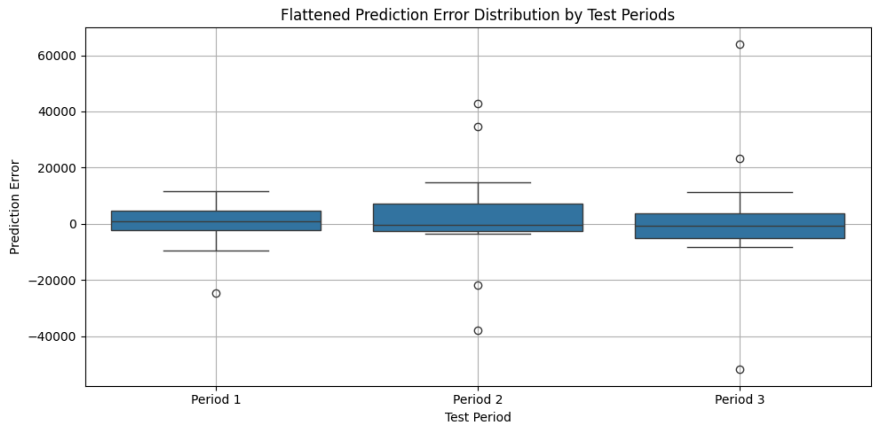
The plot of test set:

# Cluster	PLot	MAPE
Cluster 1		9.5813%
Cluster 2		32.0212%
Total		20.80%

3.3 Error Distribution Across Test Periods

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

Prophet (No Clustering)		Period 1: 9.49% Period 2: 12.89% Period 3: 9.73%
Prophet (No Clustering and removing Outliers)		Period 1: 9.11% Period 2: 16.09% Period 3: 28.00%
Prophet (Clustering:Kmeans k = 2)		Period 1: 12.10% Period 2: 14.46% Period 3: 11.28%

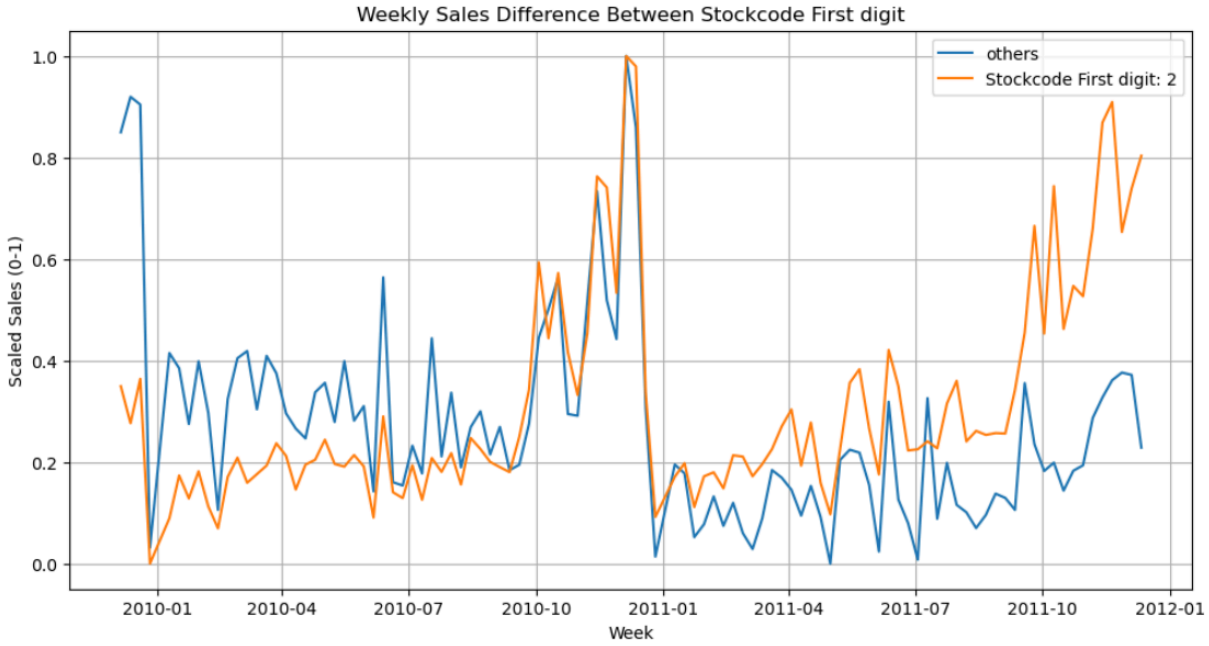
Prophet (Clustering: Agglomerative (k=2))		Period 1: 18.75% Period 2: 18.48% Period 3: 25.18%

Model 4 Results: LightGBM

1. Model Selection and Training Setup

Note: LightGBMl was not used by the previous group.

LightGBM is a gradient boosting framework that is often used in time series prediction even though it is not specifically designed for it. For this section on model training, we tried training separate LightGBM models based on differences in stock codes, as well as training one model on the whole dataset.



As shown in the plot above, we observed a difference in trend between the series of products with stock codes starting with 2 and those starting with other numbers. This difference could be due to variations in the project type or the platform on which the products are being sold. We would like to test whether

combining results from training separate LightGBM models will outperform training a single model on the whole dataset with LGBM.

The predictions are performed on a weekly scale. We created the following lag variables to help the model learn from previous values at different past time points, with lags of [1, 3, 5, 10, 20]. We then trained a LightGBM model using the LGBMRegressor function from the lightgbm package. Hyperparameter tuning was performed based on the following selected candidate parameters: 'n_estimators': [50, 100, 150, 300], 'learning_rate': [0.05, 0.075, 0.1], 'max_depth': [3, 5, 10, 15], and 'num_leaves': [2, 5, 10, 30]. For the two separate models trained based on differences in stock code initials, the aggregated result is the combined prediction result from each model.

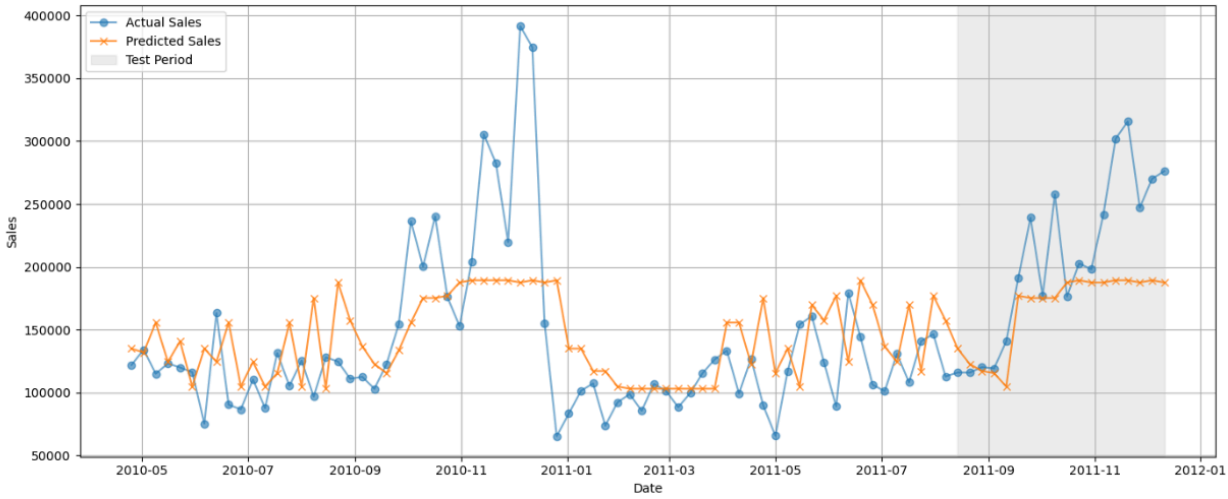
2. Overall Model Performance

Model	MAPE	MAPE improvement from previous group best model(34.0815%)
aggregated model	26.720%	+7.3615%
one model for all	18.034%	+16.0475%

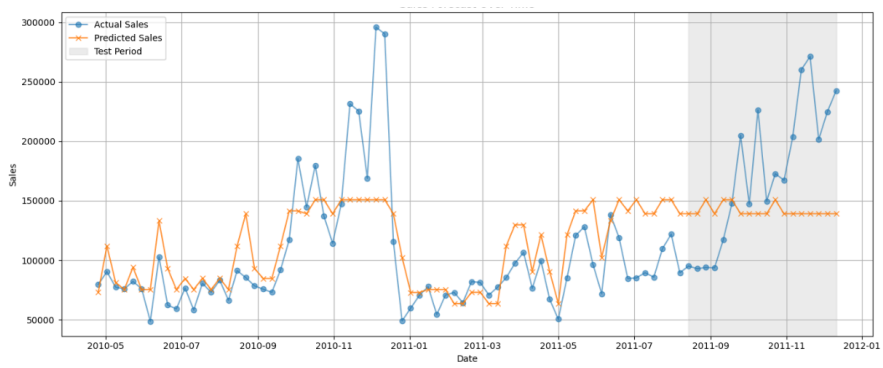
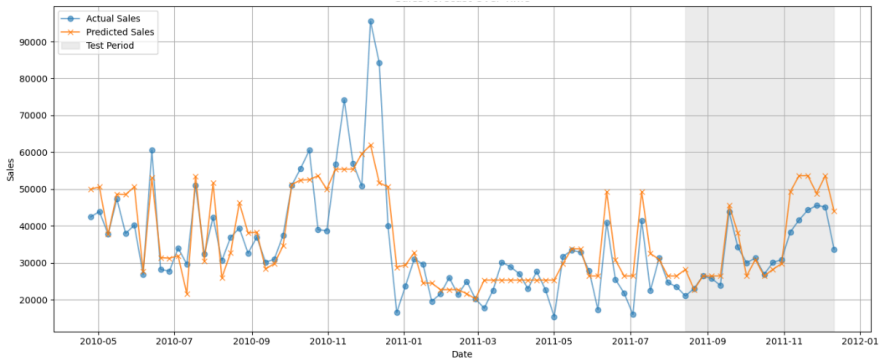
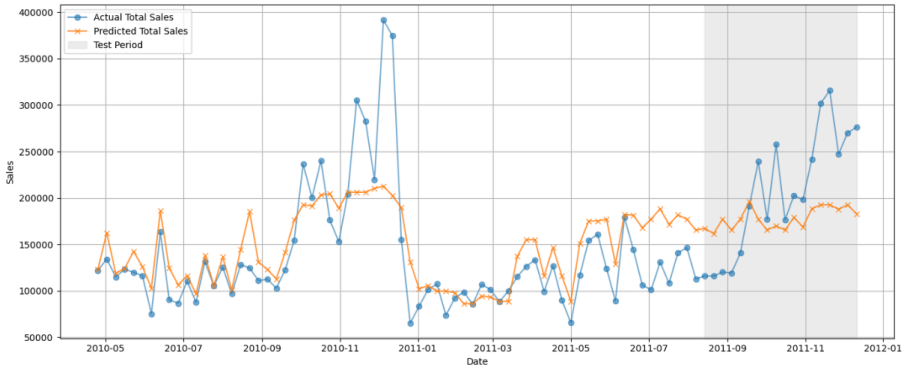
3. Visualization of Model Performance

3.1 Best Model: one model for all data

Best Hyperparameters: {'n_estimators': 50, 'learning_rate': 0.05, 'max_depth': 3, 'num_leaves': 2}



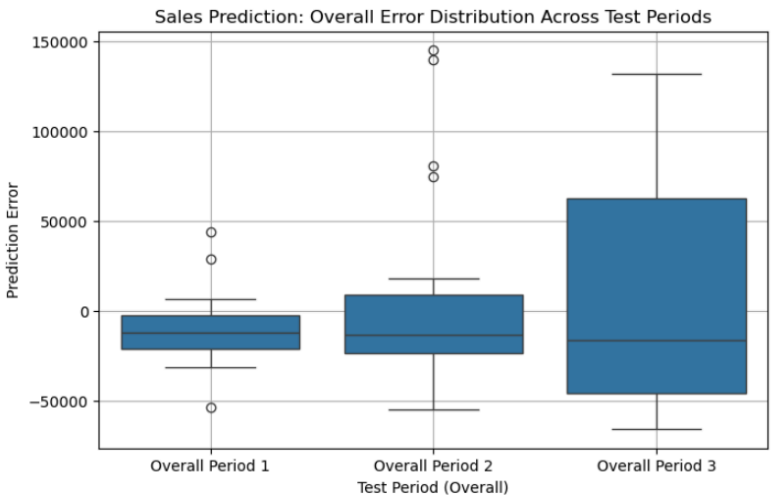
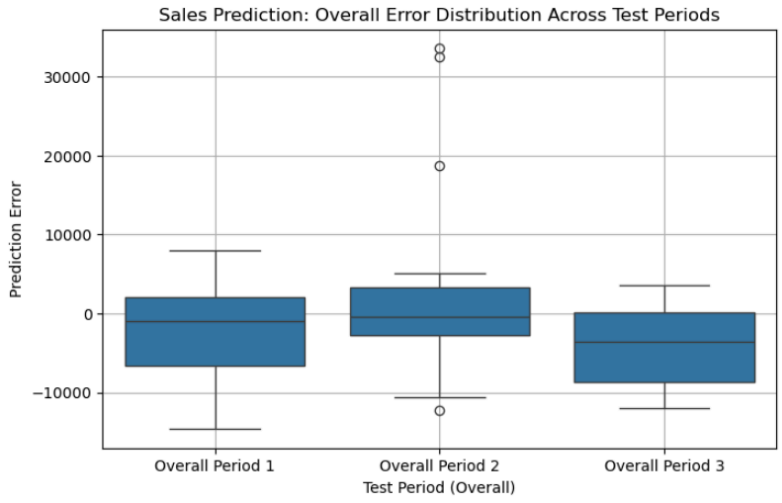
3.2 Worst Model: aggregative prediction from two separate models

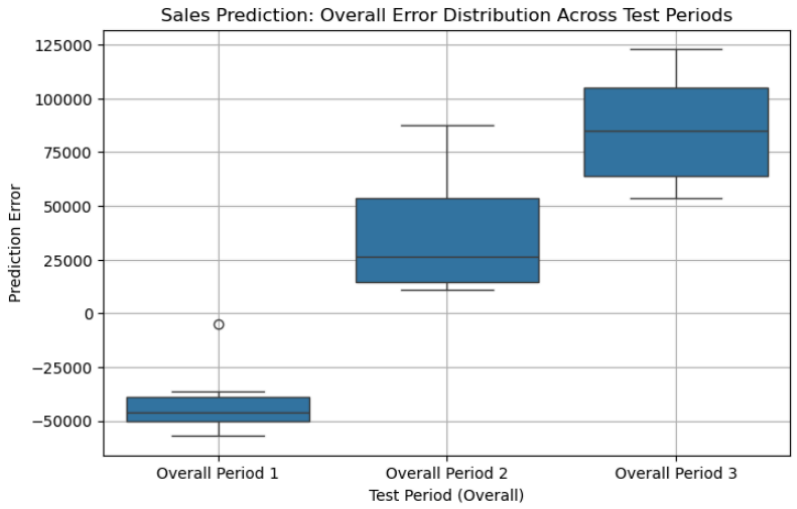
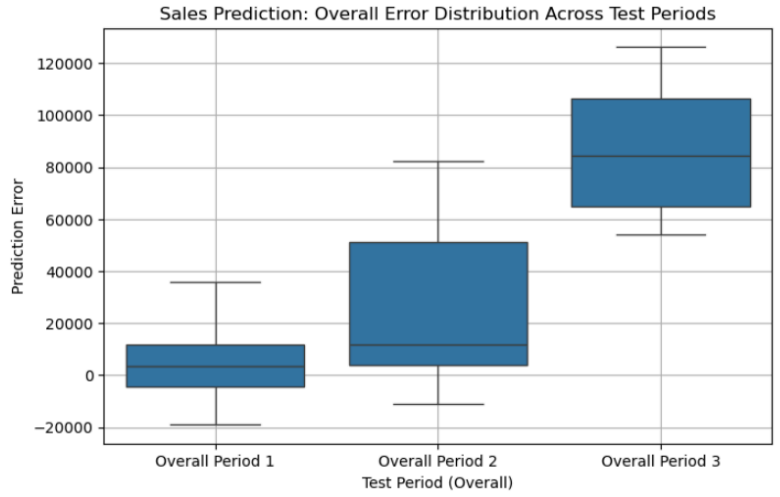
Model	Visualization of Model Performance	best hyperparameters
Model trained on product with stock code initial of 2		‘n_estimators': 50, 'learning_rate': 0.05, 'max_depth': 3, 'num_leaves': 2
Model trained on product with stock code initial beside 2		'n_estimators': 300, 'learning_rate': 0.075, 'max_depth': 3, 'num_leaves': 2
aggregated prediction result		not applicable

3.3 Error Distribution Across Test Periods

Looking at these results, using a single LightGBM model to train on all the data performs better than training separate models based on stock code initials. Although the model trained on products with stock codes other than those starting with 2 shows a better MAPE than the single model, the overall aggregated result is worse because the trend for products with stock codes starting with 2 is more

difficult to capture. For all models, dividing the MAPE into three parts shows that they all struggle more with capturing sudden sales increases at more distant time points.

Model	Error Distribution Plot	Overall MAPE for Period 1, 2, 3
Model trained on product with stock code initial of 2		18.35%, 26.11%, 37.15%
Model trained on product with stock code beside of 2		12.61%, 18.31% 17.49%

aggregated prediction result		33.01%, 16.55%, 30.60%
one model for all		10.19%, 12.97%, 30.95%

Further Improvements

For Product Categorization:

- Incorporate additional product features such as sales volatility, seasonality strength, or product segmentation information.
- Evaluate clustering quality using both internal metrics (e.g., Silhouette Score) and external business validation.

For XGBoost Model:

- Further tuning of advanced hyperparameters could improve the model's ability to generalize, especially under limited training samples.
- Beyond basic lags and date-based features, incorporating engineered features like moving averages, rolling standard deviations, and holiday indicators could help capture more complex sales dynamics.
- Although XGBoost can capture non-linearities, it does not natively handle seasonality. Explicitly creating seasonality-related features (e.g., Fourier terms) may improve prediction performance.

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:

- Even though we have already taken a lot of parameters into consideration, for ‘changepoint_prior_scale’, ‘seasonality_prior_scale’ and ‘holiday_prior_scale’ we have not explored a lot of values since searching in the large parameter grid will be time consuming. So future improvements could explore more values for the hyperparameter.
- It is worth trying if ‘Logistic’ trend could improve the performance since we only use the default version of the trend

For LightGBM Model:

- Add more time-related features and selecting lag values based on relevant literature rather than relying solely on experience (e.g., adding lags like 7 or 30). Consider incorporating features such as whether a date falls on a weekend, the beginning or end of the month, or other calendar-based indicators to help the model better capture time-based patterns.
- For different types of products, employ LGBM with different sets of parameters and features.
- Perform hyperparameter tuning not only for model parameter but also for lag features and calendar-based features.

Team Information

Zhiqi Ma (zm2467)

Ruoheng Du (rd3165)

Xiaojiang Wu (xw3022)

Yijian Liu (yl5778)