Forecasting Online Product Sales

A Study of Multi-Series Time Series Models with LLM-Based Clustering

Brianna Ta
bht2118@columbia.edu

Erin Ma
em3910@columbia.edu

Luiz do Valle
lld2131@columbia.edu

Kennard Mah
ksm2198@columbia.edu

May 2025

Abstract

Accurate product-level demand forecasting is critical for inventory optimization, operational planning, and customer satisfaction in online retail. However, traditional univariate time series models struggle with data sparsity and fail to leverage cross-series patterns. In this study, we evaluate modern multi-series forecasting approaches—including Prophet, LSTM, DeepAR, and Temporal Fusion Transformer (TFT)—to predict daily unit sales for products sold by a UK-based online retailer. To enhance model performance, we experiment with both classical clustering techniques (K-Means, DBSCAN, HDBSCAN) and novel approaches using Large Language Model (LLM) embeddings (Gemini 2.5) derived from product descriptions. We assess forecasting accuracy using the symmetric mean absolute percentage error (sMAPE), analyze model behavior across clusters, and explore the added value of explanatory variables such as macroeconomic indicators and seasonal features.

Our results show that deep learning-based models, particularly TFT, outperformed other approaches when trained across multiple series, though the improvements were marginal—likely due to the limited availability of clean data. Both DeepAR and TFT achieved lower sMAPE values even when trained on the full dataset, whereas Prophet and LSTM performed well only when modeling a small number of individual products. We conclude with a discussion on the trade-offs between accuracy, interpretability, and scalability, and highlight future directions for improving LLM-based clustering methods and applying these models to real-world challenges such as data sparsity, cold-start problems, and uncertainty estimation.

Contents

1 Introduction

Forecasting plays a critical role in modern retail operations, enabling businesses to anticipate demand, manage inventory, optimize supply chains, and improve customer satisfaction. Traditional forecasting methods like SARIMAX [1] and regression often model each product (time series) in isolation, limiting their ability to generalize across similar entities with limited historical data. With the rise of deep learning, new models have emerged that can capture shared patterns across multiple time series, learn from auxiliary features, and produce probabilistic forecasts that reflect uncertainty in future demand.

In this paper, we aim to forecast the number of units sold of various products offered by a UK-based, non-store online retailer. We explore four modern time series forecasting models: Facebook Prophet [2], Recurrent Neural Network (RNN–LSTM) [3], Temporal Fusion Transformer (TFT) [4], and DeepAR [5]. We compare their performance to each other and to that of two simpler models: the vanilla regression decision tree [6], and XGBoost [7]. These models span both statistical and neural paradigms and differ in how they handle seasonality, cross-series generalization, and static metadata.

To further improve forecasting accuracy, we attempt to cluster products together. In addition to 'classical' clustering techniques like KMeans with Dynamic Time Warping (DTW) and KShape, we leverage Large Language Models (LLMs) in the clustering pipeline. By clustering, we aim to provide structured groupings that can inform model training and better handle cold start scenarios.

Contribution

Our contributions are as follows:

1. We benchmark four multi-series forecasting models—Prophet, LSTM, DeepAR, and TFT—on daily product-level sales using a real-world online retail dataset.
2. We introduce a novel clustering approach using LLM-based embeddings (Gemini 2.5) to group products and evaluate their effectiveness with models that require clustering against single product forecasting.
3. We compare clustering-dependent models to modern global models like TFT and DeepAR that learn shared patterns without explicit clustering.
4. We incorporate explanatory variables, such as macroeconomic indicators, calendar features, and product-level pricing to improve forecasting accuracy.
5. We assess the trade-offs between forecasting accuracy, interpretability, and scalability under data sparsity and product heterogeneity.

This study is situated at the intersection of time series modeling and innovative clustering methods for heterogeneous product patterns, offering insights into how forecasting performance can be improved in data-driven retail settings with similar challenges.

2 Background Research and Methodology

2.1 Model Selection

Table 1: Model Analysis: Feature Comparison Across Forecasting Methods

Model	**Ref.**	Multi-Series	Static Features	Cold Start	Inter-pretable	Needs Clustering
Prophet	[2]	–	–	–	○	○
RNN (LSTM)	[3]	△	△	–	–	△
TFT	[4]	○	○	△	○	–
DeepAR	[5]	○	○	○	△	–

Table 1, designed from the findings of the background research conducted, compares the models that were investigated for this paper based on the following criterias:

- Multi-Series Forecast: Forecast multiple related time series (one-per-product) using a single shared model.
- Static Features: Able to use static features that do not change overtime.
- Cold Start: Ability to make prediction for low-history products with limited data.
- Interpretable: Understand the reasoning behind predictions (e.g., feature importance, attention).
- Needs Clustering: Pre-clustering requirements for better performance.

These criteria were selected based on a real-world application of online retail products and the business application of forecasting. In addition to these criteria, this paper will cover how accurate each model performs, conducting a fair comparison between each models. Section 2.1 covers background research for each model that were selected for the analysis.

2.1.1 Facebook Prophet

Prophet is an open-source forecasting tool introduced by Taylor and Letham [**?**] to facilitate fast and automated forecasts at scale. It uses a decomposable time series model with additive components—trend, seasonality, and holidays—following principles laid out in structural time series modeling [8]. The model is suited for business applications, designed with practicality and interpretability in mind, and features parameters that can be automatically adjusted to improve fit [9]. Prophet is commonly used as a baseline in forecasting research due to its robust handling of recurring patterns (e.g., annual or weekly seasonality, holiday effects) and automatic changepoint detection.

In terms of **Multi-Series**, Prophet is inherently a univariate model—fit individually for each time series. It does not learn across multiple series jointly, so grouped forecasting is handled by independently fitting models to each series. This independence is a limitation compared to other models designed for multi-series learning.

Regarding **Static Features** and **Cold Start** scenarios, Prophet allows for additional regressors, but these are typically time-varying inputs such as promotional periods or holidays. Purely static attributes (e.g., category, product type) are not directly leveraged unless manually encoded, limiting the model's ability to generalize to new series with sparse data.

A key strength of Prophet is its **Interpretability**. By design, it provides clear decomposition into trend, seasonality, and holiday effects, allowing users to inspect and adjust each component. However, this simplicity assumes that the time series can be captured by additive effects, which may underfit more complex or non-linear dynamics that deep learning models are better suited for.

Finally, with respect to **Clustering Needs**, Prophet does not inherently perform clustering of time series. If grouping is desired, preprocessing is required to cluster series based on similarity. Overall, Prophet prioritizes interpretability and simplicity over advanced cross-series learning mechanisms.

2.1.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) networks, have become popular for time series forecasting in the last decade. RNNs process sequences through recurrent connections, making them well-suited to model temporal dependencies. LSTM, in particular, introduces gating mechanisms to overcome the vanishing gradient problem, enabling the capture of long-term dependencies in sequence data. In the context of forecasting, LSTM models can learn complex patterns such as non-linear trends and temporally varying effects that classical models may not capture [10].

LSTM models have demonstrated strong performance in diverse domains, including stock price prediction [11] and weather forecasting [12]. Many academic studies adopt LSTM-based models to tackle challenges like multi-step or multivariate prediction, leveraging the ability of RNNs to encode temporal information across time steps.

When it comes to **Multi-Series** and grouped forecasts, RNNs are traditionally trained on a single time series. However, more recent approaches involve training a single LSTM model on a collection of

related series, effectively learning a global model. This enables the network to share parameters and extract common temporal patterns across groups of series, as demonstrated in models like DeepAR [5]. Grouped forecasting with LSTMs improves accuracy when individual series are limited in history but share seasonal or behavioral patterns. For example, Bandara et al. [13] propose an LSTM ensemble that captures multiple seasonal patterns across related series.

LSTM models can incorporate both time-varying and **Static Features**. Static features—such as store ID or product category—can be embedded and input at each time step to guide the model's predictions [5]. This allows the model to adjust forecasts based on cross-sectional information. Such representations support **Cold Start** scenarios by enabling generalization to new series with limited history, as long as descriptive features are provided.

Interpretability is a common limitation of LSTM models. Unlike additive models like Prophet, LSTMs function as black boxes, with internal states that lack transparent structure. Interpretability efforts have focused on input perturbation, feature attribution, or the use of attention layers. Recent hybrid models like the Temporal Fusion Transformer (TFT) address this by combining LSTM backbones with interpretable attention mechanisms [4].

When it comes to **Clustering Needs**, RNN-based models can interact with clustering in two ways. First, time series can be clustered prior to modeling, enabling specialized LSTM models per cluster. Second, the LSTM's learned embeddings can themselves be clustered to reveal latent groupings of similar dynamics. In most applications, clustering is a preprocessing tool to help structure model training across highly heterogeneous series. LSTM models are flexible and powerful, but thoughtful design is required to balance generalization, complexity, and interpretability.

2.1.3 Temporal Fusion Transformer (TFT)

The Temporal Fusion Transformer (TFT) is a state-of-the-art deep learning model for time series forecasting, proposed by Lim et al. [4]. TFT combines the strengths of recurrent neural networks and attention mechanisms to achieve both high forecasting accuracy and interpretability. Designed for multi-horizon forecasting, it predicts a sequence of future values (e.g., the next N time steps) rather than a single point forecast. A major innovation of TFT is its ability to handle heterogeneous data by incorporating static covariates (e.g., product category), known future inputs (e.g., calendar features), and observed historical values. Unlike classical Transformers used in NLP, TFT interleaves recurrent layers to model short-term dependencies and self-attention layers to capture long-range interactions. Gating mechanisms and variable selection networks are further employed to dynamically determine the most relevant features for each prediction [14].

TFT is inherently a global model, meaning it can learn from and predict across **Multi-Series** simultaneously. It achieves this by integrating static metadata such as store ID or product type, enabling the model to capture both shared and series-specific patterns. This architecture is highly effective for grouped forecast scenarios—such as retail demand across thousands of products—where cross-series information is valuable. Lim et al. [4] demonstrate the model's superior performance across electricity, traffic, and retail datasets.

A core strength of TFT lies in its handling of **Static Features** and its robustness in **Cold Start** situations. By feeding static covariates through dedicated encoders, TFT influences both recurrent and attention components of the model. When a new series is introduced with minimal historical data, TFT can still produce informed forecasts by drawing on static attributes and learned relationships from similar series in the training data.

Despite its architectural complexity, TFT was developed with a strong emphasis on **Interpretability**. Through variable selection networks, it identifies the most influential features at each time step, and through self-attention layers, it reveals which past observations contribute most to each prediction. These mechanisms allow practitioners to trace the model's reasoning, such as identifying whether a recent spike was influenced by promotional periods or seasonal patterns.

Regarding **Clustering**, TFT largely eliminates the need for manual preprocessing. The model's ability to condition predictions on static inputs means it can internally differentiate behavior across series—essentially mimicking the effects of clustering. Nonetheless, in cases of extreme data heterogeneity, practitioners may still segment the data before training to further tailor model behavior. Even so, TFT's dynamic structure allows for **adaptive forecasting** across series within a single unified framework.

2.1.4 DeepAR (Autoregressive RNN)

DeepAR is a probabilistic forecasting model developed by Amazon researchers that uses an autoregressive RNN—specifically an LSTM—trained across multiple related time series [5]. Unlike traditional forecasting approaches that model each series independently, DeepAR learns a global model that captures shared dynamics across a **Multi-Series** forecasting. This makes it particularly effective in domains with many series but limited data per series, such as retail or supply chain forecasting.

By incorporating categorical features to identify each time series, the model generalizes patterns like seasonal effects or promotions without requiring manual feature engineering. It produces Monte Carlo samples of future values, enabling users to compute prediction intervals and forecast uncertainty. DeepAR has shown strong empirical results, often outperforming classical models, particularly in datasets with grouped structure and heterogeneous scales.

The model also supports **Static Features**, which are embedded and repeated across time steps to inform the network. These allow DeepAR to address **Cold Start** scenarios—new series with limited history can still be forecasted based on their metadata. For instance, a newly introduced product can be forecasted using learned behavior from others in its category, even with minimal past data.

While DeepAR provides valuable probabilistic forecasts, it lacks native **Interpretability**. The model does not decompose predictions into transparent components like trend or seasonality. Some insights may be inferred post hoc—for example, analyzing response to input perturbations or visualizing the spread of forecast quantiles—but these are limited compared to models like TFT that are explicitly designed for explainability.

Regarding **Clustering Needs**, DeepAR largely obviates the need for manual segmentation. It handles heterogeneity through scaling mechanisms and conditioning on covariates. In fact, clustering by scale or demand volume may degrade performance in heavy-tailed distributions. Still, in cases involving fundamentally different series (e.g., energy vs temperature), separate models may be justified. Overall, DeepAR simplifies forecasting at scale by learning a shared representation, enabling generalization and robust performance without extensive preprocessing.

2.2 LLM-based Clustering Methods

2.2.1 LLMs for Clustering and Knowledge Extraction

Large Language Models (LLMs) such as GPT-4, Claude, Gemini, and DeepSeek have moved beyond traditional language tasks and are increasingly being used for clustering and knowledge extraction. These models, trained on extensive corpora, generate semantically rich embeddings that can be grouped using conventional clustering algorithms like k-means or hierarchical clustering. Texts with similar meanings are placed close in embedding space, leading to more coherent clusters compared to traditional vectorization methods. Studies such as Petukhova et al. [15] have shown that LLM-derived embeddings outperform classical approaches on metrics like cluster purity and silhouette scores.

LLMs also support clustering through few-shot or interactive methods. For instance, Huang et al. [16] demonstrated that by providing a few labeled examples, a GPT-based model could generalize to unseen items with high clustering quality. These models act as domain-aware oracles, using world knowledge to group items semantically—for example, recognizing that "apple" and "banana" belong together in a fruit category. Once clusters are formed, LLMs can also generate concise labels or descriptions that explain the cluster's concept, which is particularly useful for applications like topic modeling.

Few-shot prompting enables zero- or low-example clustering by asking models to infer grouping logic. Wu et al. [17] showed that GPT-4, when prompted appropriately, could organize unlabeled texts into meaningful themes. This strategy is effective especially when the number of categories is small or inferable from context.

In addition to clustering, LLMs excel at knowledge extraction—identifying structured information from unstructured text. By following prompt instructions, they can extract entities, relationships, or even generate knowledge triples in formats like JSON. A single prompt can yield a complete semantic graph, such as linking "Haiti" to "Caribbean Community" through membership relations. Therefore, LLMs can group extracted knowledge by topic or infer schema from repeated patterns—for example, deducing a biography schema like `Person { Occupation { Notable Work` from reading multiple biographies. This can be seen in tasks like document classification, where LLMs can group texts by theme using only

natural prompts. While evaluation remains challenging, this flexibility makes them powerful tools for open-ended clustering.

In summary, LLMs support clustering by embedding semantic information, generalizing from examples, labeling clusters, and extracting structured knowledge. They enable workflows that previously required pipelines of NLP tools, bridging the gap between raw text and structured insights.

2.2.2 Prompt Engineering

Prompt design is critical when using LLMs for clustering or knowledge extraction. Polat et al. [18] studied how different prompting strategies affect extraction performance. They found that providing a single clear example more than doubled accuracy, while adding multiple examples yielded diminishing returns. The quality of the example mattered more than quantity, suggesting that a well-chosen demonstration can be more effective than a large prompt.

Interestingly, chain-of-thought prompting did not improve results in this context. For tasks like clustering or extracting triples, direct prompts often outperformed ones that encouraged the model to explain its reasoning. Retrieval-augmented prompting—where the model is supplied with relevant external context—also improved performance. For example, including Wikipedia snippets helped models assign documents to correct categories by grounding their decisions in external knowledge.

The paper suggest that effective clustering with LLMs depends on three factors: clear instructions, relevant examples, and contextual support. A typical setup might include a short example mapping a customer review to a category, followed by a list of new reviews to classify. Showing just one example per cluster may be sufficient.

Overall, LLMs are well suited to clustering and knowledge extraction when guided by well-crafted prompts. Rather than relying on complex multi-step logic, the best results come from strategic examples, compact instructions, and minimal yet relevant context. This makes LLMs a flexible and scalable alternative to traditional clustering pipelines.

In this paper, we compare clustering performance using the same prompt across several leading LLMs: LLaMA 2 [19], DeepSeek [20], GPT-4.1 [21], and Claude 3.7 Sonnet [22]. These models represent the latest advancements from both open-source and API-accessible systems.

3 Data

3.1 Online retail dataset

The online retail dataset [23] from UC Irvine's Machine Learning Repository contains all the transactions between December 2009 and December 2011 for a United Kingdom (UK) based online-only retailer. The company mainly sells unique all-occasion gift-ware. Examples of products sold by this retailer are 'brocade ring purse', 'strawberry ceramic trinket box', and 'popcorn holder'. Many of the retailer's customers are wholesalers.

The dataset contains the following information for each individual transaction: a description of the product being sold, the number of units sold, the timestamp of the transaction, the price of each unit in sterling, the ID of the customer, and the country where the customer resides.

3.1.1 Target variable

The quantity we seek to predict is the **daily number of units sold for each product**.

As discussed, the dataset contains transaction-level data, with many transactions happening at random times each day for each product. However, many models expect the timestamps to be evenly spaced. In addition, at this granularity the data is very sparse and noisy, making it challenging to learn meaningful patterns. Figure 1 shows four examples.

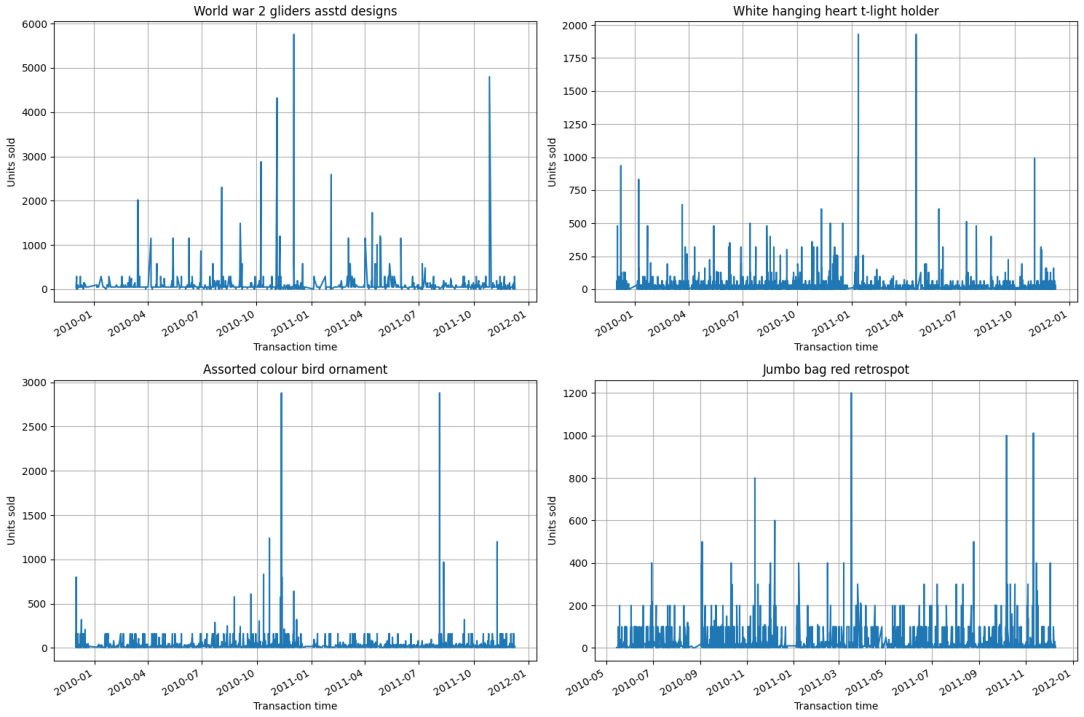


Figure 1: Raw time series of the units sold of a few example products. The unevenly spaced points and high-frequency variations make it challenging to learn patterns from the data.

Thus, we rolled up all the time series to a daily granularity. This has the effect of smoothing out some noise and aligning the time steps of all products. A downside of doing this is that it reduces the number of training points we have available and prevents the models from learning intra-day behaviors. Since the dataset contains only two years of data, any coarser aggregation (say at the weekly level) would leave us with too few data points to train and test our models on. Thus, we determined that daily aggregation strikes the optimal balance. Figure 2 shows an example of the data from Figure 1 aggregated at the daily level.

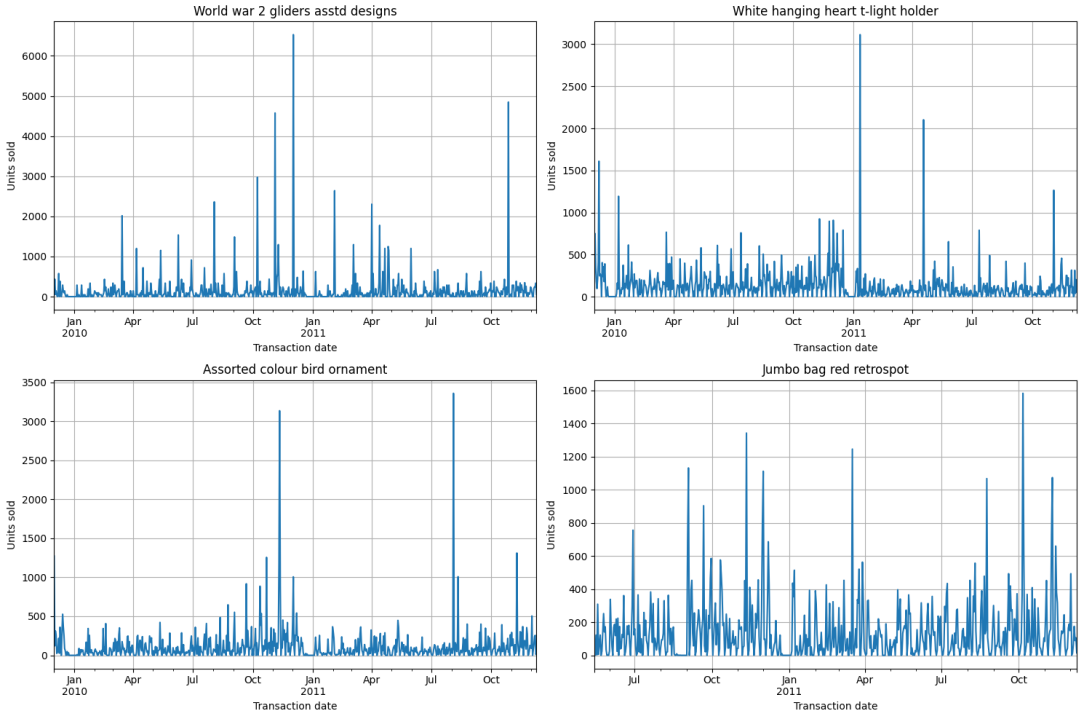


Figure 2: Daily units sold of a few example products. Weekly patterns are clearer and the points are evenly spaced.

3.1.2 Data preparation

The diagram in Figure 3 gives an overview of how we cleaned and prepared the online retail dataset. The corresponding code is here. The following sections go into the details of some of the steps.

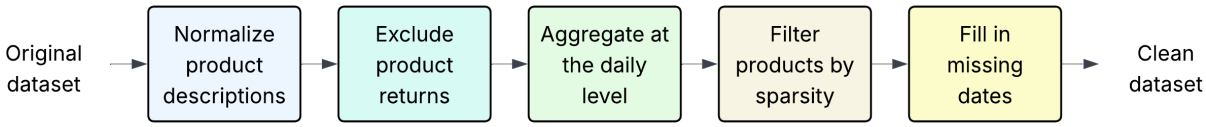


Figure 3: Pipeline used to clean the dataset.

The output of this process has four columns:

1. `Date`: The date the sales happened on.
2. `Description`: The description of the product.
3. `UnitsSold`: The total number of units sold on that date.
4. `RevenuePounds`: The total revenue (in sterling pounds) from the sales of that product on the date.
5. `AveragePricePerUnitPounds`: The revenue divided by the number of units sold.

The data is keyed by `Date` and `Description`. There are a total of 142 products, each with 739 data points.

Normalize product descriptions

Each transaction is associated with an English description of the product that was sold. Table 2 shows some examples.

We would like to aggregate the data by the product description. However, differences in punctuation, capitalization, white spaces, and typos make it so that the same product has several different syntactically

Table 2: Example rows in the online retail dataset some product orders. Some columns were removed for clarity.

Description	Quantity	InvoiceDate	Price
15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95
PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75
WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75
RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10
STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25

different, but semantically identical, descriptions. To address this, we preprocessed the descriptions as illustrated in Figure 4.

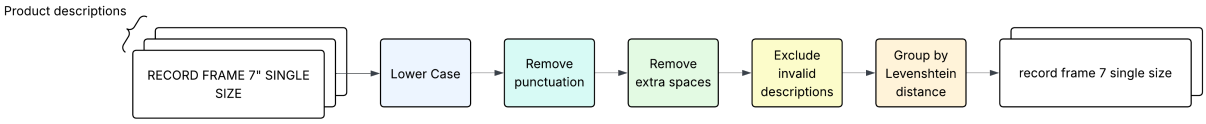


Figure 4: Pipeline used to normalize the product descriptions.

In the 'Exclude invalid descriptions' step (yellow box), we exclude rows that have the following descriptions:

- `?`
- `checked`
- `found`
- `mailout`
- `temp`
- `update`
- `damaged`
- `amazon`
- `amazon adjustment`
- `amendment`
- `this is a test product`
- `temp`
- `postage`
- `dotcom postage`
- `manual`
- `carriage`

It turns out that the dataset contains returns and other types of transactions like postage and carriage. These are low volume transactions and not actual products, so we exclude them.

The last step, 'Group by Levenshtein distance' (orange box), addresses the fact that some descriptions are nearly identical but have subtle syntactic differences (like the ordering of words) that do not change their meaning significantly. For example 'red **stripe** ceramic drawer knob' and 'red **spot** ceramic drawer knob'. We thus use the Levenshtein distance to replace descriptions with a 'canonical' description. For example, 'charlie lola blue hot water bottle', 'charlielola pink hot water bottle', 'charlie lola red hot water bottle' all get mapped to 'charlie lola red hot water bottle'. This grouping has the benfit of making the data less sparse as we combine the sales of similar products. There is the risk that dissimilar products get mapped together, but we used a relatively small distance threshold and manually inspected a few of

the mappings to ensure this was not the case.

Daily aggregation and returns filtering
As discussed in the Target variable section, we rollup the time series to a daily granularity. Before doing this, however, we exclude transactions with negative quantities or prices as they represent returns.

Filter products by sparsity
The entire dataset spans only 373 days. However, some products have very sparse sales, meaning that on several days they do not sell any units at all or they started being sold very close to the end of the time period. For example, some products may only be sold close to Christmas. However, since we only have a single year's worth of data, our models cannot learn this yearly behavior.

We decided to filter the products by two metrics: (1) the number of days spanned by the product and (2) the 'density' of days with non-zero sales. For example, a product may have 365 days between its first sale and its last, but only 50 of those days have any sales at all. This means this product's density is $50/365 \approx 14\%$. We only keep products with **a span of at least 600 days AND a density of at least 60%**. The reason being we would like to test the model on the last month (30 days) of data at the end of the period. Figure 5 illustrates which products make the cut.

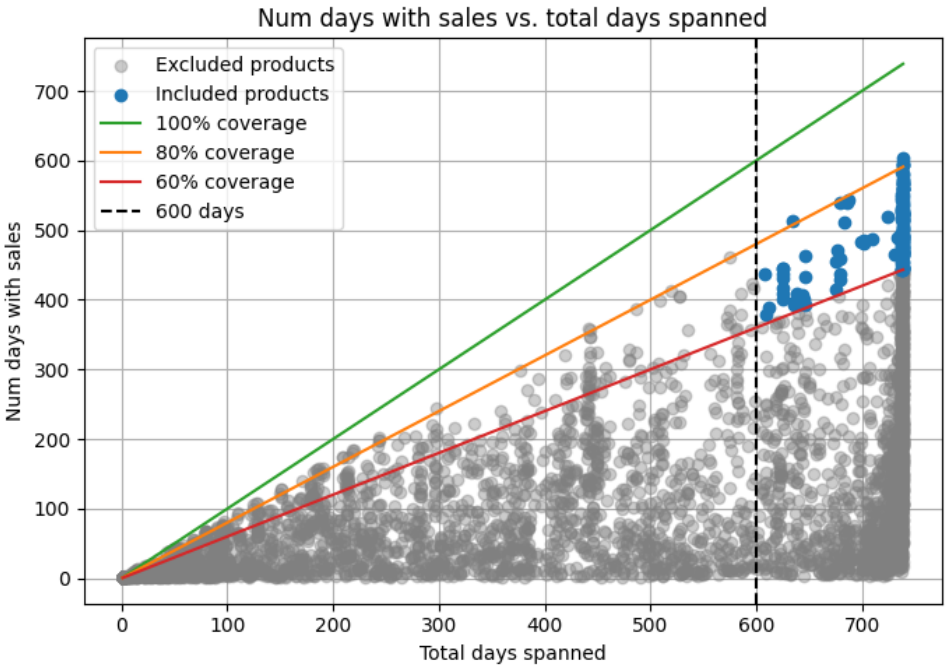


Figure 5: Illustration of the products we keep for modeling.

The result is we keep only 142 of the more than 3.9K products in the dataset. However, these 142 products are the most popular and non-seasonal ones. They account for a disproportionate number of the sales, as Figures 6, 7, and 8 show.

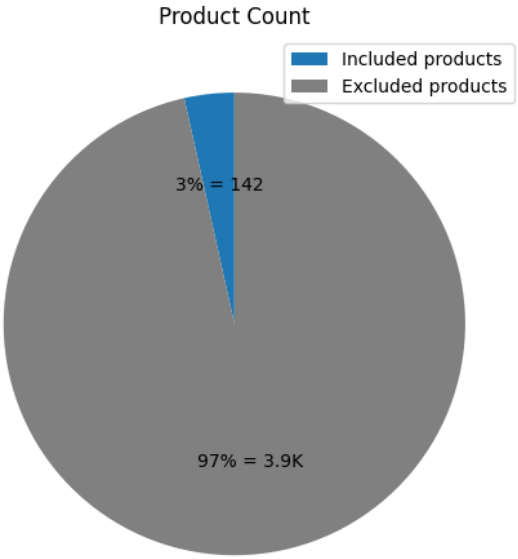


Figure 6: Percentage of products we kept vs. filtered out. We kept relatively small percentage.

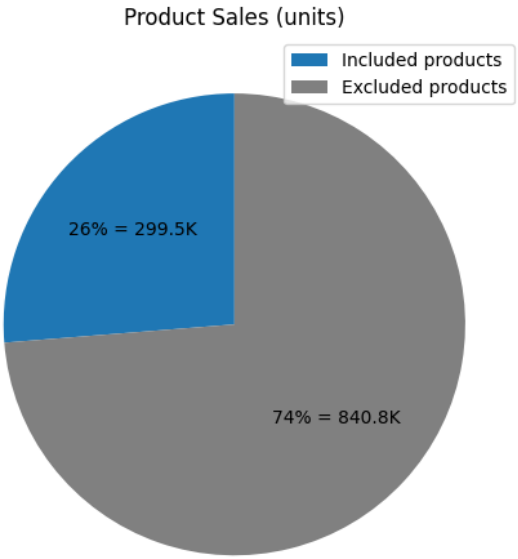


Figure 7: Fraction of unit sales from the products we kept vs. filtered out.

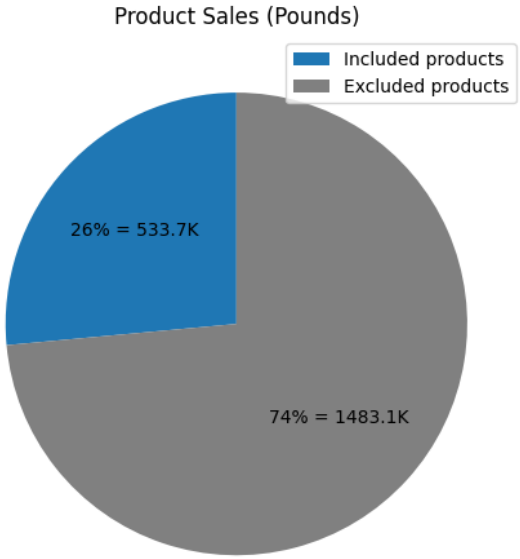


Figure 8: Fraction of total revenue (in sterling pounds) from the products we kept vs. filtered out.

Thus, although we filter out many products (97%), the ones we keep are the ones with the highest quality data and they account for a significant portion (26%) of the retailer's sales. The other products simply don't have enough data for us to learn from.

Figure 9 shows examples of products we kept, while Figure 10 shows some products we removed.

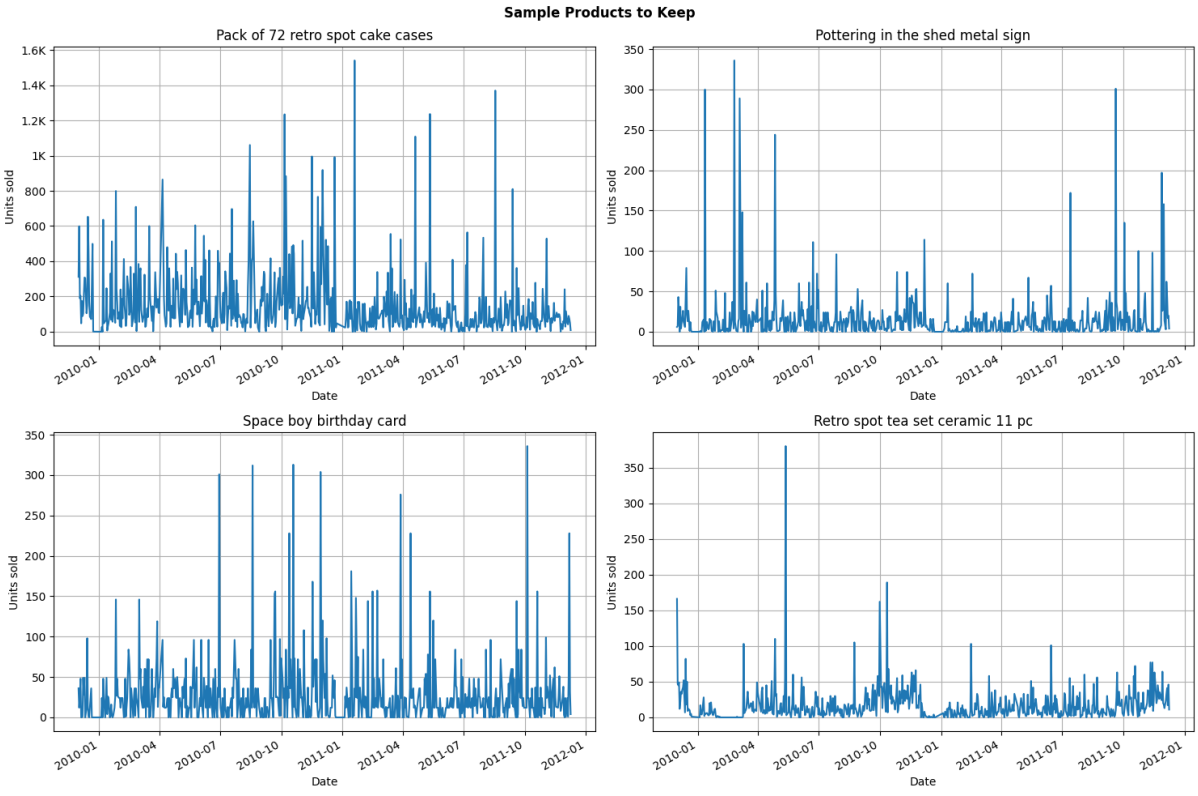


Figure 9: Daily units sold time series for some of the products we kept. Their density is relatively high.

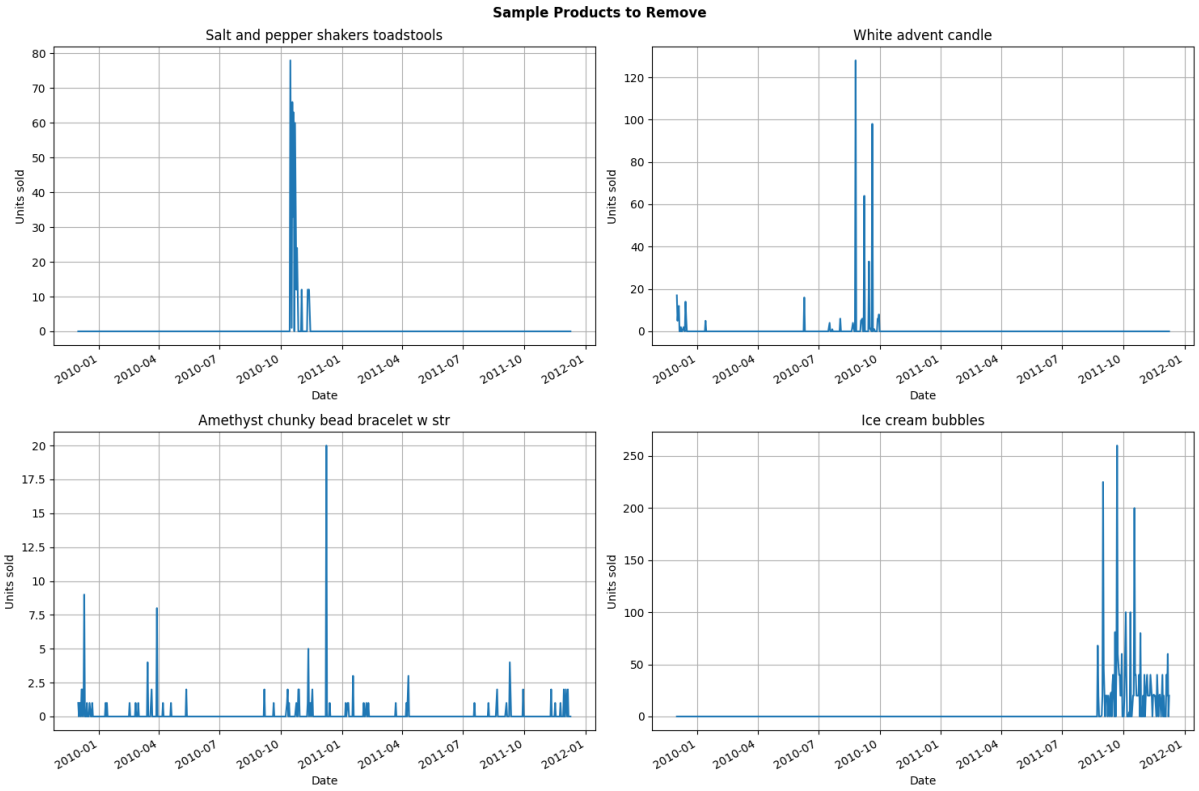


Figure 10: Daily units sold time series for some of the products we removed. Their density is low and their data is more noisy.

Fill in missing dates
Lastly, we fill in the dates with no sales for each product. For the number of units sold and revenue, we fill the missing dates for each product with zeros. For the average price, we fill it with the price of the product on the most recent next date with sales.

3.2 Explanatory variables

The retailer's sales are likely influenced by a variety of external factors, like the state of the economy and holidays. To help the models predict the sales of the various items, we introduced external explanatory variables.

3.2.1 Days of the week

Including days of the week as explanatory variables, such as is_saturday, can significantly improve the accuracy of our time-series models. These variables capture recurring weekly patterns in customer behavior that might not be captured by the date and time alone. For example, we noticed that this retailer sold no units on Saturdays, perhaps indicating that it is closed on Saturdays, making is_saturday a critical feature for predicting sales. Other days may reflect variations due to delivery schedules or customer ordering habits tied to the business week. These factors can help the model distinguish between true trends and predictable weekly fluctuations, leading to more accurate predictions.

3.2.2 Shopping holidays

Shopping holiday indicators (such as is_black_friday and is_cyber_monday) could improve the accuracy of our model by capturing sharp spikes in demand on these dates. Even for a non-retail store in the UK, these global shopping events could influence a buyer's behavior, as these holidays usually correspond

with increased promotions, online traffic, and bulk purchasing. Including these two indicators could help predict outliers in the sales data.

3.2.3 Public holidays

Holidays like Christmas likely impact the volume of sales for the retailer as customers stock up on gifts. We sourced this information from [24], which contains public and local holidays from 2010-2019 for various countries.

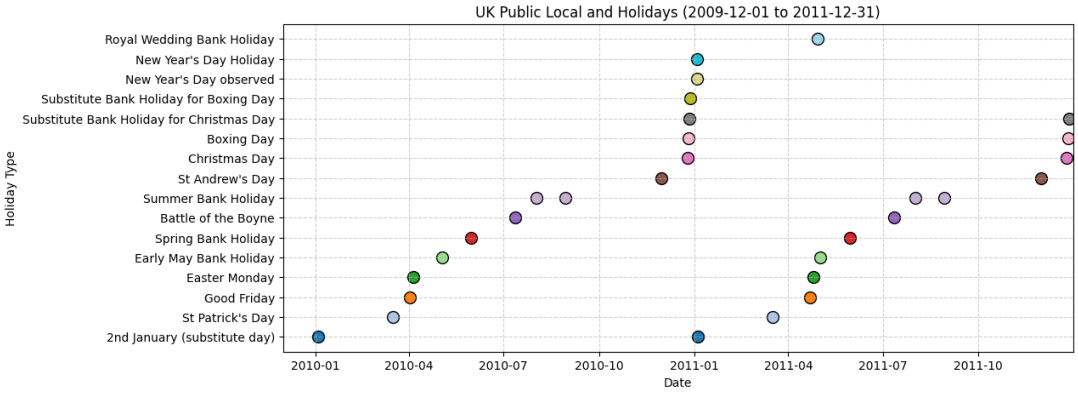


Figure 11: UK public holidays.

3.2.4 Consumer price index (CPI)

Including the Consumer Price Index (CPI) as an explanatory variable helps account for the impact of inflation on consumer purchasing behavior. As prices rise, consumers may reduce spending on non-essential items or shift to cheaper alternatives, leading to lower sales volumes. CPI provides a macroeconomic context that can enhance the model's ability to detect trends not visible through historical sales data alone.

We sourced the dataset from the Office for National Statistics (ONS), UK's largest independent producer of official statistics and its recognized national statistical institute. It contains monthly CPI since 2003. To get the daily data, we filled in the dates in each month with the CPI for the month. Figure 12 shows the result.

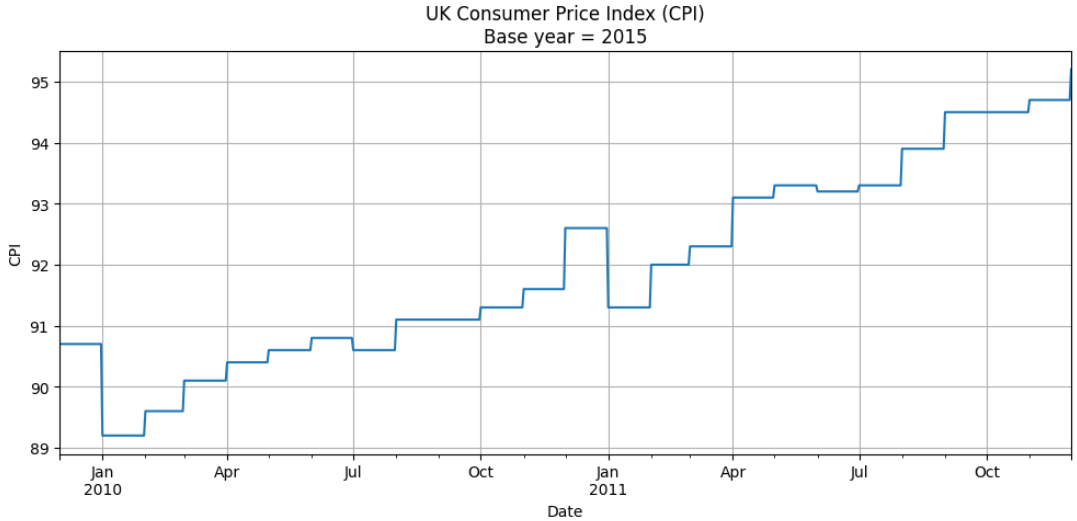


Figure 12: UK Consumer Price Index (CPI)

3.2.5 Consumer Confidence Index (CCI)

The Consumer Confidence Index (CCI) captures how consumers' perceptions of the economy influence their spending behavior. When confidence is high, consumers are more likely to spend on discretionary items, boosting sales; when confidence falls, they may delay or reduce purchases due to economic uncertainty. CCI reflects expectations about income, employment, and general economic conditions, offering valuable insight into demand shifts driven by sentiment rather than just price or seasonality. By incorporating CCI, the model gains a behavioral dimension that complements historical trends and improves forecasting accuracy, particularly during periods of economic volatility.

We collected the dataset from the Organization for Economic Co-operation and Development (OECD), an intergovernmental organization founded in 1961 to promote policies that improve the economic and social well-being of people around the world. Among other responsibilities, the OECD collects and publishes data on a wide range of economic and social issues. The dataset contains the monthly CCI for various countries going back to 1974. To get daily data, we filled in the missing dates with the CCI for that month. Figure 13 shows the result.

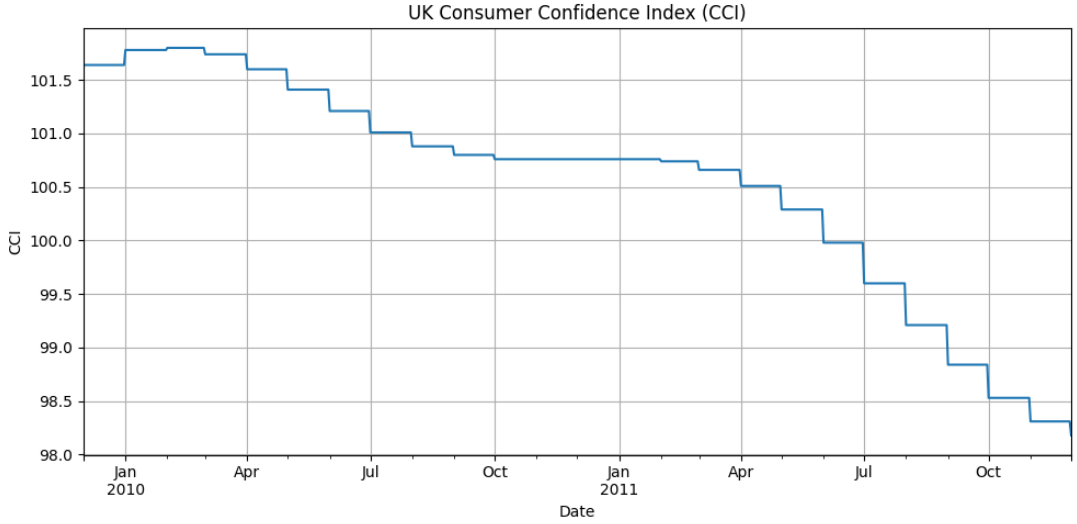


Figure 13: UK Consumer Confidence Index (CCI)

3.2.6 Unemployment

Including unemployment data as an explanatory variable helps account for the direct impact of labor market conditions on consumer spending behavior. When unemployment is high, individuals generally have less disposable income and greater financial uncertainty, leading to reduced spending—especially on non-essential goods. Conversely, lower unemployment typically signals stronger consumer demand and confidence in economic stability.

As for the Consumer price index (CPI), we sourced the dataset from ONS. It contains the seasonally adjusted monthly unemployment rate for ages 16 and over in the UK going back to 1971. To get daily data, we filled in the missing dates with the unemployment rate for that month. Figure 14 shows the result.

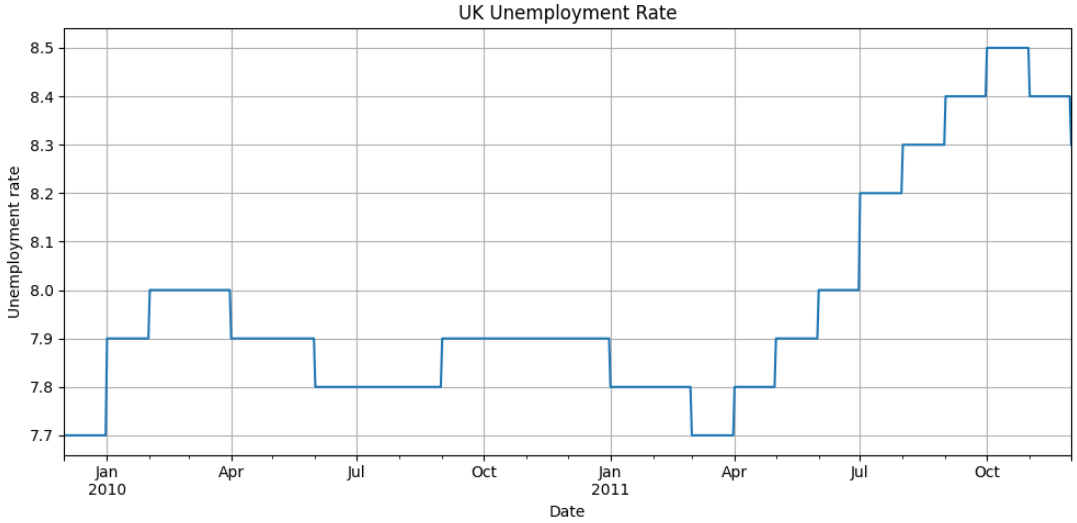


Figure 14: UK unemployment rate for ages 16 and over.

3.2.7 Interest rate

Incorporating interest rate data helps capture the influence of monetary policy on consumer and business spending behavior. When interest rates rise, borrowing becomes more expensive and saving more attractive, which can lead to reduced consumer spending and lower retail sales. Conversely, lower interest rates tend to encourage spending and investment by reducing the cost of credit. The Sterling Overnight Index Average (SONIA) reflects short-term interest rate expectations and central bank policy in the UK, making it a key indicator of financial conditions.

We collect this dataset from the Bank of England, UK's central bank. It contains the daily average of the interest rates that banks pay to borrow sterling overnight from other financial institutions and other institutional investors. A handful of dates were missing, so we filled them with the interest rate from the most recent date with data. Figure 17 shows the result.

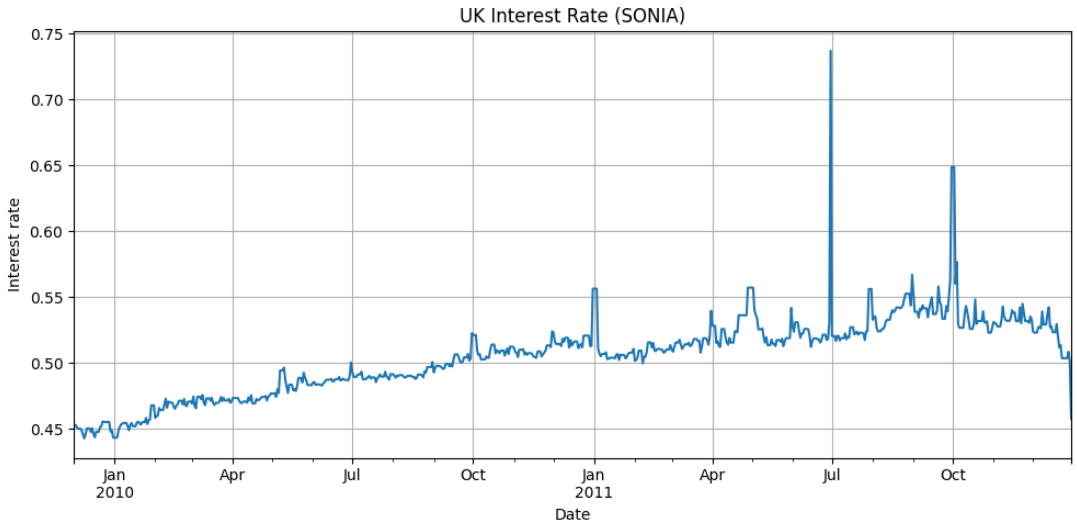


Figure 15: UK short term interest rate.

3.2.8 Retail Sales Index (RSI)

The Retail Sales Index (RSI) is a key economic metric that measures the changes in the retail sector's sales of goods and services over a specific period. This variable reflects the overall health and performance of the retail industry in the UK and can provide insights into consumer spending patterns, trends, and demand. These indicators, such as percentage changes in current prices and chained volume measures over various time frames, capture broader spending trends and shifts in purchasing behavior that influence demand. By including these variables, the model has more visibility into the performance of the wider retail industry, allowing it to adjust for market-wide patterns and better predict the specific store's sales.

3.2.9 Google search trends

Google search trends for terms like "gift ideas" in the UK could be a strong indicator for product sales, especially around seasonal peaks like Christmas. Spikes in search interest reflects rising consumer intent to purchase, correlating well with actual purchases. For example, the search volume for "gift ideas" spikes in December and portrays heightened demand for products that can be used as gifts, aligning with the increased sales in the same period. Including this variable in our time-series models can help capture shifts in consumer interest that traditional economic indicators might miss. This variable allows the model to anticipate sales patterns by sentiment and interest rather than just historical trends and can be particularly useful for non-retail businesses that may have more purchases due to seasonal gift-giving behavior.

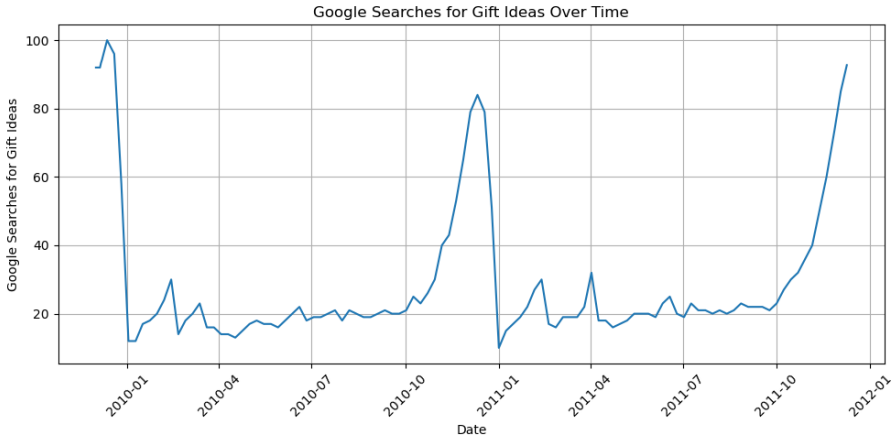


Figure 16: Google Search Trends

3.2.10 Average price

Average price per unit is set by the retailer and captures the direct relationship between product pricing and consumer demand. As prices increase, especially for non-essential items, consumers may reduce the quantity they purchase or switch to alternative products, leading to lower sales volumes. Conversely, lower prices can stimulate demand and increase unit sales. This variable reflects product-level price sensitivity and allows the model to detect demand elasticity—how strongly sales respond to price changes. By incorporating average price per unit, the model can more accurately identify pricing effects and better forecast fluctuations in units sold due to promotional activity or pricing strategies.

We collected this data from the Online retail dataset and defined it as the total revenue divided by the number of units sold on that day. For dates with no sales, we filled the missing values with the next available average price.

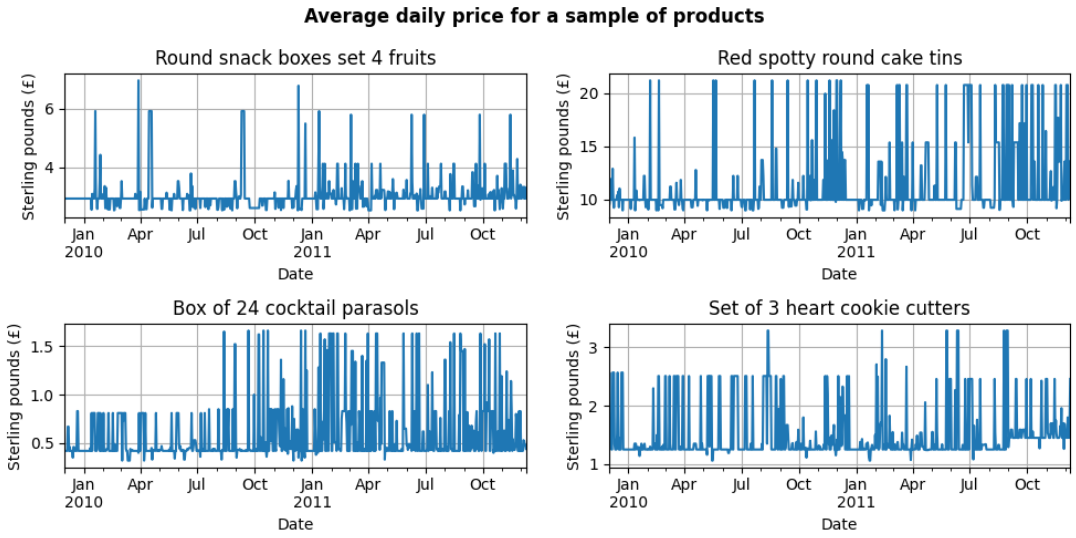


Figure 17: Average daily price for a sample of products. There is a lot of variability, suggesting the retailer may offer deals.

3.3 Exploratory data analysis

In this section, we report some results from the exploratory data analysis we conducted. The goal is to illustrate some key characteristics of the data. We use the online retail data after the pre-processing described in the Data preparation section.

3.3.1 Total revenue

Figure 18 shows the total daily revenue (in Sterling Pounds) across all products over the course of the year. The series exhibits strong weekly seasonality, with regular fluctuations likely reflecting differences in consumer behavior between weekdays and weekends. These recurring peaks and troughs suggest a predictable intra-week pattern in demand.

While daily revenue is highly volatile, with several prominent spikes throughout the year—possibly due to bulk orders or promotions—the overall trend is relatively flat. A fitted linear trend line (shown in red) indicates only a very modest upward slope of approximately £0.71 per day. This suggests that, despite short-term variability and seasonal effects, average daily revenue remained mostly stable throughout the period.

The latter part of the year shows greater amplitude in fluctuations, potentially reflecting holiday-driven activity such as Christmas shopping. However, the increase is not sustained enough to significantly shift the long-term revenue trajectory.

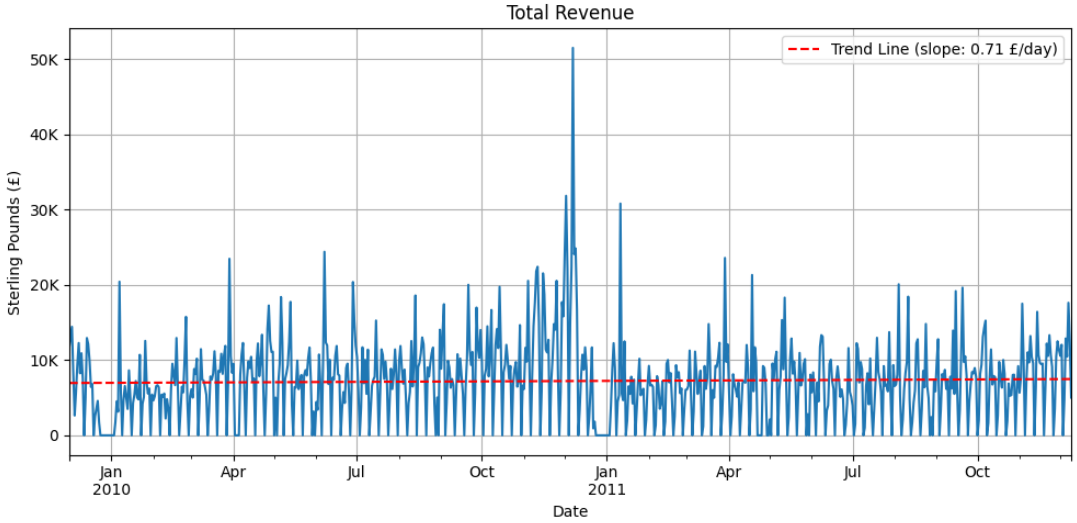


Figure 18: Total daily revenue with fitted linear trend. The red dashed line shows the linear trend (slope $\approx$ £0.71/day).

3.3.2 Units sold distribution

As observed in Figure 18, there appear to be approximately four days each month with zero sales. This pattern suggests that the store might be closed on a specific day of the week. Figure 19 confirms this hypothesis: no units are sold on Saturdays, indicating that the store likely does not operate on that day.

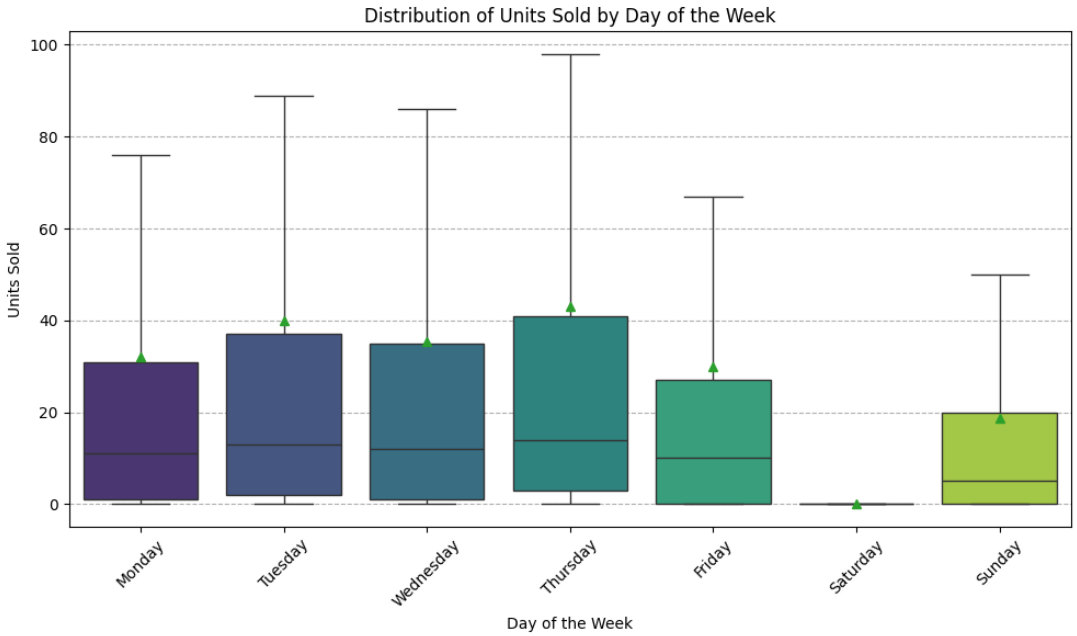


Figure 19: Distribution of units sold by day of week. Nothing is sold on Saturday, suggesting the store may be closed.

Additionally, as shown in Figure 21, the sales volume across most products tends to increase toward the end of the year. This pattern may reflect seasonality in consumer demand, possibly driven by holiday-related shopping.

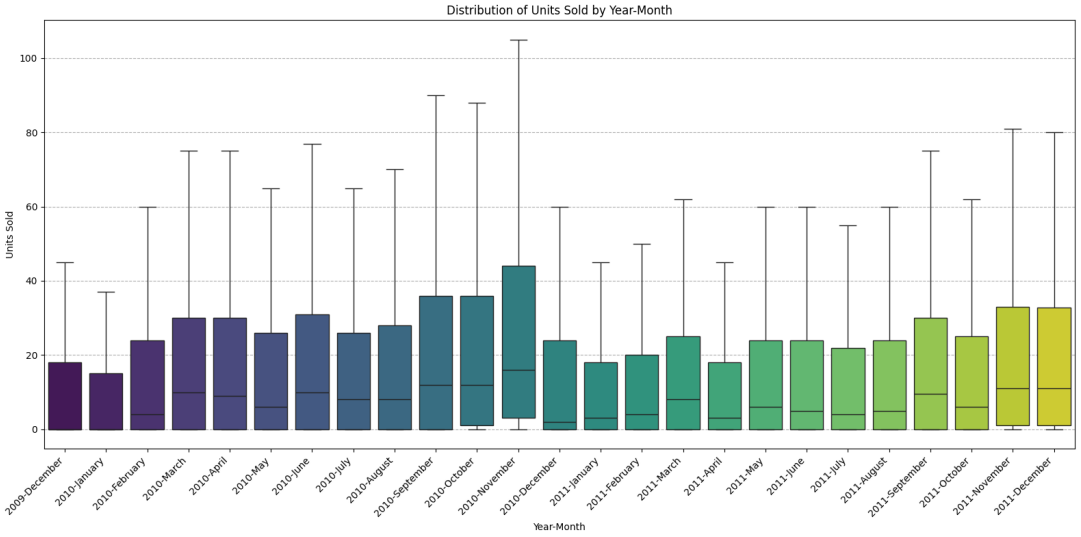


Figure 20: Distribution of units sold by month. The sales volume increases towards the end of the year.

3.3.3 Product spot-checks

In this section, we take a look at the characteristics of the units sold time series for a random sample of four products. The goal is to get a general understanding of the behavior of the target we seek to forecast.

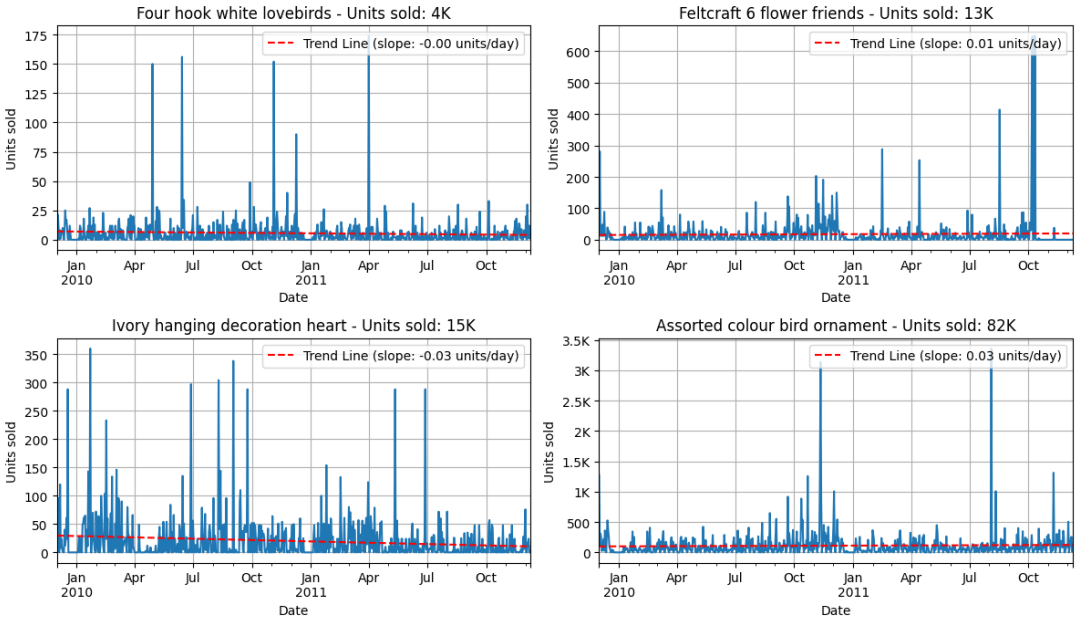


Figure 21: Daily units sold for a random sample of four products. Each plot includes a fitted linear trend line.

Figure 21 presents the daily sales volume of four products, each exhibiting distinct dynamics over the observed period. The plotted trend lines (in red) help quantify long-term tendencies in demand, while the overall patterns reflect the diversity in sales behavior across products:

- **Four hook white lovebirds (4K units sold)** shows relatively low and sporadic sales with occasional spikes. The overall trend is flat (slope ≈ 0), indicating stable long-term demand despite short-term fluctuations.

- **Feltcraft 6 flower friends (13K units sold)** displays mostly low daily sales punctuated by abrupt high-volume events, particularly toward the end of the period. The trend line shows a slight upward slope (0.01 units/day), suggesting mild growth in demand.
- **Ivory hanging decoration heart (15K units sold)** initially has high volatility with numerous peaks, especially in the first half of the time series. However, the fitted trend line indicates a small but consistent decline in daily sales (slope = -0.03 units/day).
- **Assorted colour bird ornament (82K units sold)** is the highest-selling item in the sample, with sustained moderate sales and periodic large spikes. Its trend line has a modest positive slope (0.03 units/day), reflecting gradual growth in popularity over time.

These examples demonstrate the heterogeneity in product-level demand: some items have stable sales punctuated by bursts, while others trend upward or downward. Capturing such diverse behaviors is critical for building effective forecasting models that generalize well across product categories.

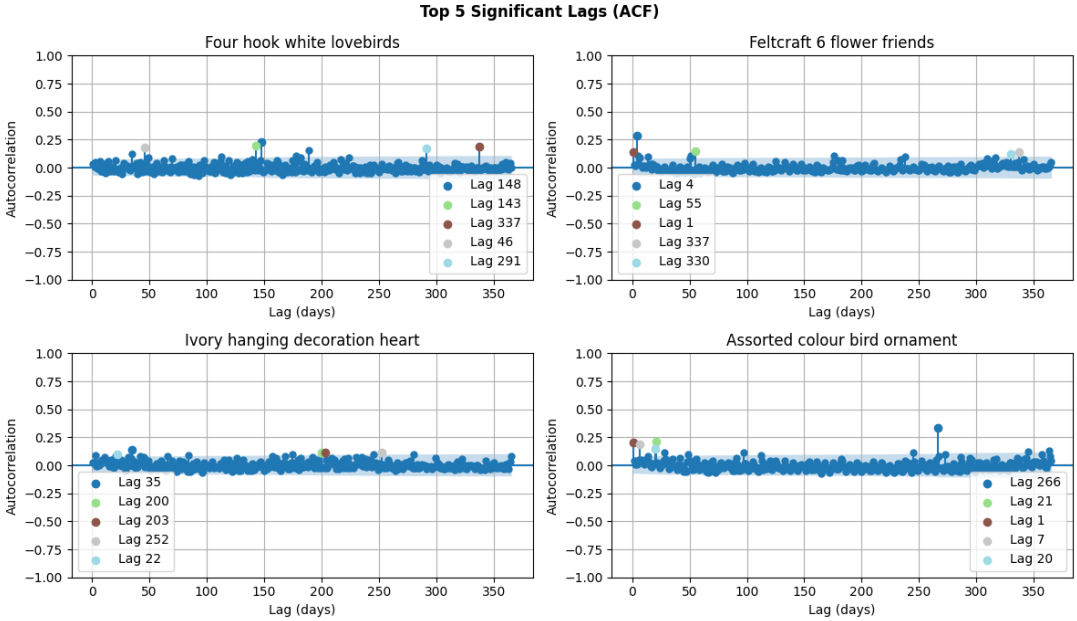


Figure 22: Autocorrelation functions (ACFs) of daily units sold for a sample of four products. The top five significant lags for each product are highlighted with colored dots.

To explore temporal dependencies in product demand, Figure 22 presents the autocorrelation functions (ACFs) for a random sample of four products. The top five statistically significant lags are annotated in each subplot. These peaks in autocorrelation indicate potential seasonality, recurring demand cycles, or memory in the sales process:

- **Four hook white lovebirds** shows scattered significant lags, including days 46, 143, 148, 291, and 337. These long-range lags suggest occasional, loosely recurring spikes in demand rather than a strong short-term memory structure.
- **Feltcraft 6 flower friends** has significant lags at 1, 4, 55, 330, and 337 days. The short lags (1, 4) point to short-term autocorrelation, while the longer lags may correspond to seasonal events or yearly cycles.
- **Ivory hanging decoration heart** exhibits mild autocorrelation at days 22, 35, 200, 203, and 252. These lags are somewhat dispersed, hinting at medium- to long-range periodic influences rather than weekly seasonality.

- **Assorted colour bird ornament** shows a mix of short and long lags (1, 7, 20, 21, 266), including a clear 7-day component that likely reflects weekly shopping behavior. The presence of multiple nearby lags (20, 21) could suggest complex short-term rhythms.

Overall, the ACF analysis reveals that while some products exhibit predictable weekly or yearly autocorrelation, others follow irregular or bursty patterns. This diversity supports the need for flexible forecasting models that can adapt to both memoryless and seasonally structured demand series.

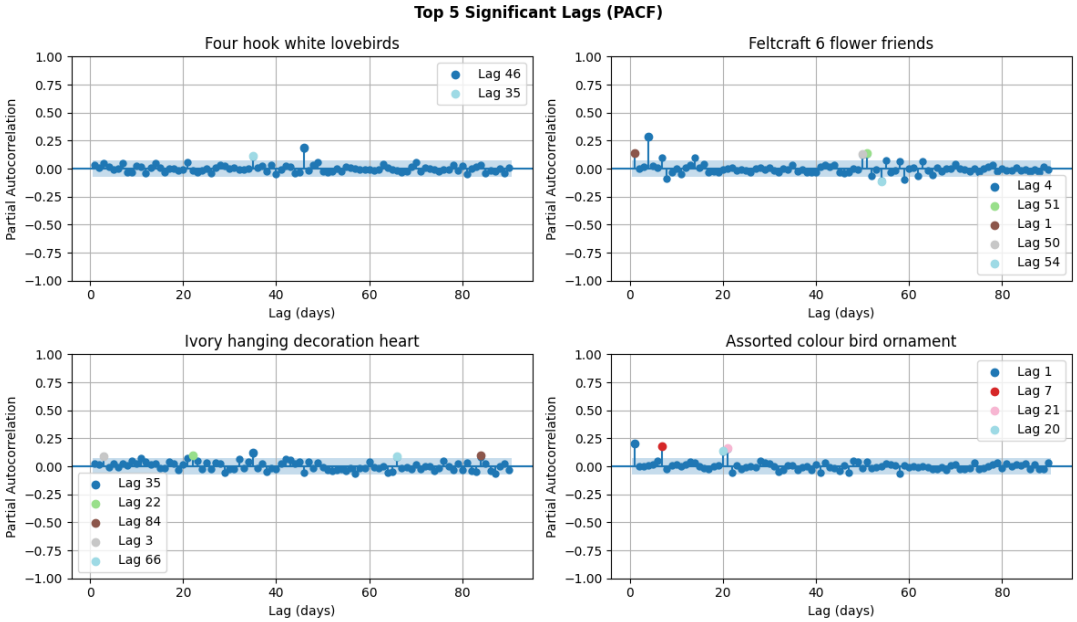


Figure 23: Partial autocorrelation functions (PACFs) of daily units sold for a sample of four products. The top five statistically significant lags for each product are highlighted.

To gain further insight into the structure of temporal dependencies, Figure 23 presents the partial autocorrelation functions (PACFs) for four products. While ACFs capture both direct and indirect correlations, PACFs isolate the unique contribution of each lag, helping to identify how many previous observations are directly useful in predicting the next.

- **Four hook white lovebirds** exhibits weak partial autocorrelation, with significance only at lags 35 and 46. This suggests limited direct temporal structure, possibly reflecting noise or event-driven sales.
- **Feltcraft 6 flower friends** shows short-term structure with lags 1 and 4 being significant, alongside moderate signals at mid-range lags (50, 51, and 54). This points to a combination of autoregressive behavior and potential multi-week cycles.
- **Ivory hanging decoration heart** displays the broadest spread of significant lags, including short (3), medium (22, 35, 66), and longer (84) delays. This mixed pattern may indicate both regular and sporadic influences on demand.
- **Assorted colour bird ornament** features strong and consistent signals at short lags: 1, 7, 20, and 21. These lags align with daily and weekly rhythms, suggesting a predictable, recurring sales structure.

These PACF results underscore the heterogeneity of temporal dynamics across products. While some items benefit from clear short-term predictability, others exhibit more diffuse or delayed dependencies, requiring product-specific modeling strategies and flexible autoregressive inputs.

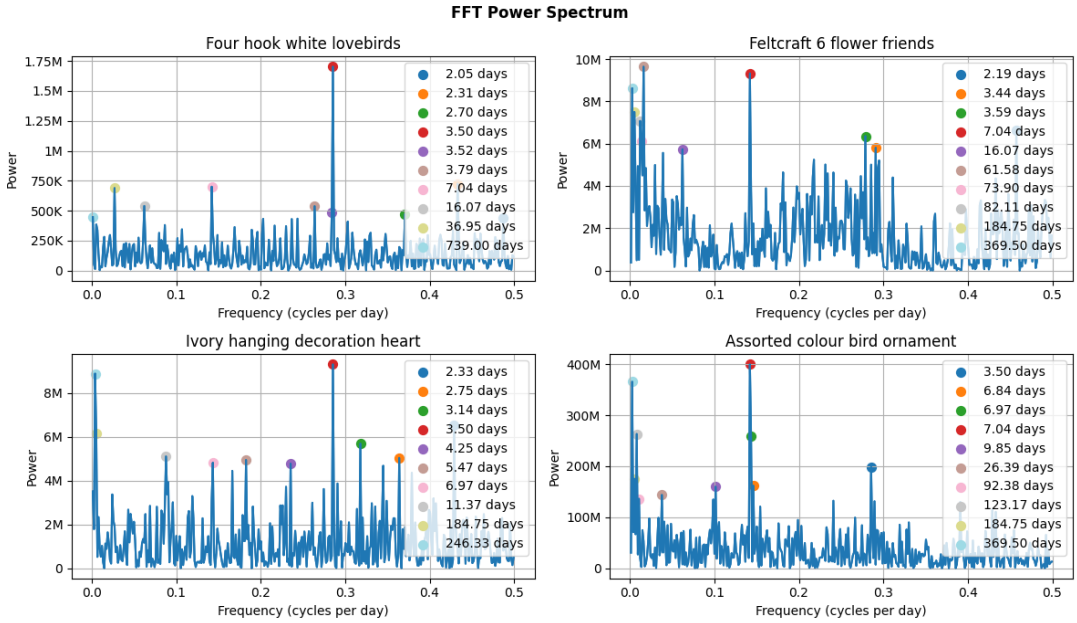


Figure 24: Power spectra of daily units sold for a sample of four products, computed via the Fast Fourier Transform (FFT). The top 10 dominant frequencies are marked, and their corresponding periods (in days) are shown.

While autocorrelation-based methods highlight time-lagged dependencies, frequency-domain analysis using the Fast Fourier Transform (FFT) reveals periodic structures that may not align cleanly with specific lags. Figure 24 presents the power spectra of four products, where dominant frequencies—represented as peaks—suggest the presence of cyclical sales behavior. Each subplot marks the ten most prominent frequencies, annotated by their periods (in days).

- **Four hook white lovebirds** shows a clear dominant peak near a 3.5-day cycle, along with supporting peaks at approximately 2, 7, 16, and 37 days. This suggests multiple short-term rhythms, possibly reflecting promotional timing or replenishment patterns.
- **Feltcraft 6 flower friends** exhibits a rich spectrum of peaks across both short (2–7 days) and long time scales (61–370 days), indicating potential weekly cycles as well as seasonality or annual effects. The dominant peak near 2.7 days suggests rapid fluctuations.
- **Ivory hanging decoration heart** features prominent power at short cycles (2.3–5.5 days), as well as noticeable components near 11, 184, and 246 days. This mixed frequency content points to both intra-week periodicity and broader seasonal trends.
- **Assorted colour bird ornament** shows concentrated peaks between 3.5 and 10 days, reinforcing a strong intra-week signal. Additional peaks at 26, 92, and 123 days hint at monthly or quarterly cycles in demand.

Overall, FFT-based spectral analysis reveals both short- and long-term cyclicality across products. While all four products exhibit strong intra-week signals, some also show clear evidence of higher-level seasonality. These insights are particularly useful for modeling recurring effects in forecasting pipelines.

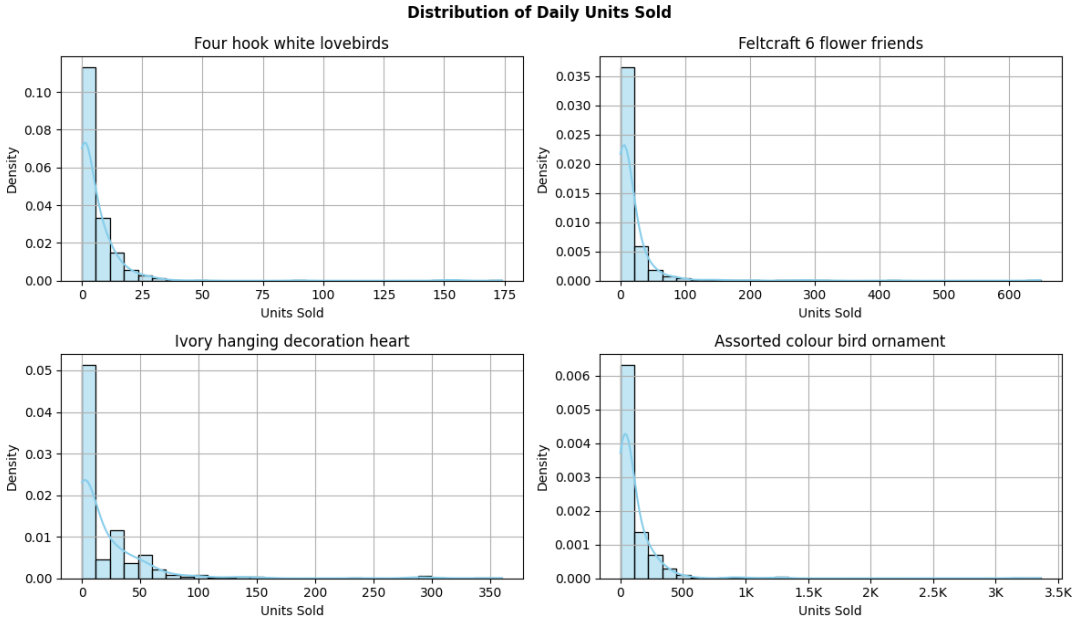


Figure 25: Distribution of daily units sold for a sample of four products. Each histogram is overlaid with a kernel density estimate (KDE) to visualize the smoothed distribution.

Figure 25 presents the empirical distributions of daily units sold for four representative products. Each subplot includes a histogram and kernel density estimate (KDE), offering a view of both discrete frequency and smoothed probability density.

- **Four hook white lovebirds** and **Feltcraft 6 flower friends** display extremely right-skewed distributions, with the vast majority of days yielding fewer than 10 units sold. Nonetheless, both products experience sporadic sales surges exceeding 100 units, underscoring the presence of rare but impactful demand spikes.
- **Ivory hanging decoration heart** also shows a steeply right-skewed distribution but spans a broader range of sales, including many days with 20–50 units sold. This suggests more frequent moderate sales events compared to the first two products.
- **Assorted colour bird ornament** exhibits the most dispersed and heavy-tailed distribution, with daily sales reaching over 3,000 units. The bulk of sales still clusters under 500 units per day, but the long tail reflects high-volume outliers likely associated with bulk purchases or major seasonal demand.

Across all four products, the patterns confirm key challenges in modeling retail sales: zero-inflation, overdispersion, and long-tailed behavior. These properties favor modeling strategies that accommodate heteroscedasticity and skewness, such as logarithmic transformations, quantile-based losses, or probabilistic forecasting models.

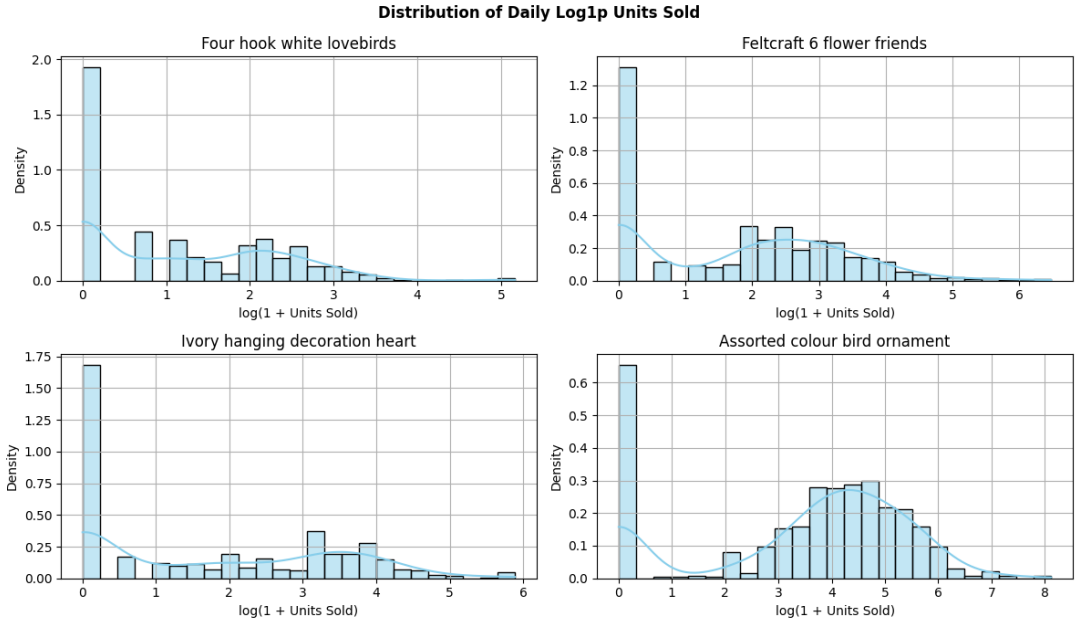


Figure 26: Distribution of daily units sold for a sample of products, after applying a $\log(1+x)$ transformation. Each histogram is overlaid with a KDE to visualize the smoothed distribution.

To mitigate the extreme skewness observed in raw sales data, we apply a log-transformation using $\log(1+x)$ and examine the resulting distributions in Figure 26. This transformation compresses high-volume outliers while preserving zero values, yielding more interpretable and statistically tractable distributions.

- **Four hook white lovebirds**, **Feltcraft 6 flower friends**, and **Ivory hanging decoration heart** retain noticeable right skewness even after transformation, with sales mass concentrated below log-values of 2 or 3 and a tapering tail extending rightward.
- **Assorted colour bird ornament**, the highest-volume product in the sample, displays a clearly unimodal and roughly symmetric distribution after transformation. The log-scale histogram suggests consistently high demand, centered around $\log(1+\text{sales}) \approx 4$ to 5.

These log-transformed distributions more clearly differentiate between product-level sales behavior, making them suitable for forecasting models that assume Gaussian-like errors or require stabilized variance.

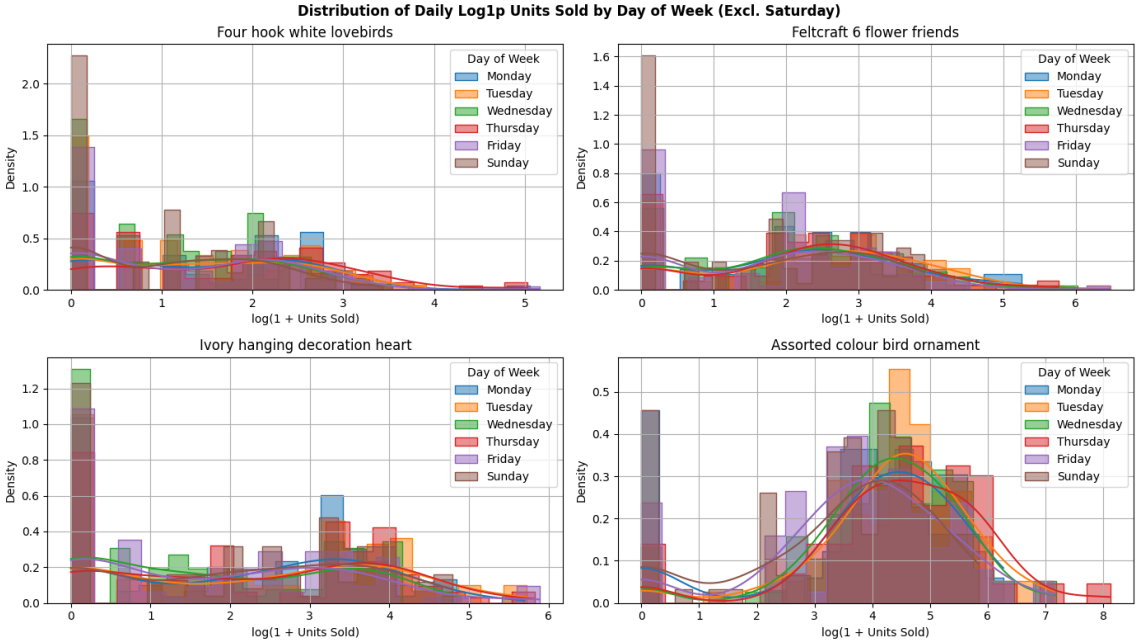


Figure 27: Distribution of daily log-transformed units sold by day of the week (excluding Saturdays), for a sample of four products. Each subplot shows overlaid histograms and KDEs for different weekdays.

To explore intra-week patterns in product demand, Figure 27 shows the distribution of daily units sold—log-transformed using $\log(1+x)$—by day of the week, excluding Saturdays. Each subplot overlays histograms and KDE curves for six weekdays, providing insight into recurring weekly behavior.

- **Four hook white lovebirds** shows relatively flat day-to-day differences, though Sunday appears more concentrated near zero. A small cluster of mid-range values occurs across weekdays, suggesting infrequent but regular sales.
- **Feltcraft 6 flower friends** exhibits mild variation, with Tuesday and Thursday showing denser activity around log-values of 2–4. This suggests slightly stronger sales midweek, though patterns are not sharply distinct.
- **Ivory hanging decoration heart** displays a bimodal shape across most days, reflecting both frequent low-sale days and occasional larger sales bursts. Monday through Friday appear similar, while Sunday again shows slightly reduced density in higher ranges.
- **Assorted colour bird ornament** shows the clearest intra-week regularity. All weekday distributions are approximately bell-shaped, centered near $\log(1 + \mathrm{x}) \approx 4$–5, indicating stable, high-volume demand. Sunday distribution is slightly flatter and shifted, implying a modest dip in weekend sales.

These results reinforce earlier observations: while some products exhibit structured weekly sales cycles, others remain irregular or sparse. This heterogeneity further motivates the inclusion of weekday indicators as features in forecasting models.

3.3.4 Product sale correlations

To explore relationships among products, we compute the pairwise Pearson correlation of daily unit sales across all 142 products. Figure 28 shows the correlation heatmap based on the raw number of units sold, while Figure 29 uses log-transformed values via $\log(1 + x)$ to account for skewness and reduce the influence of large sales outliers.

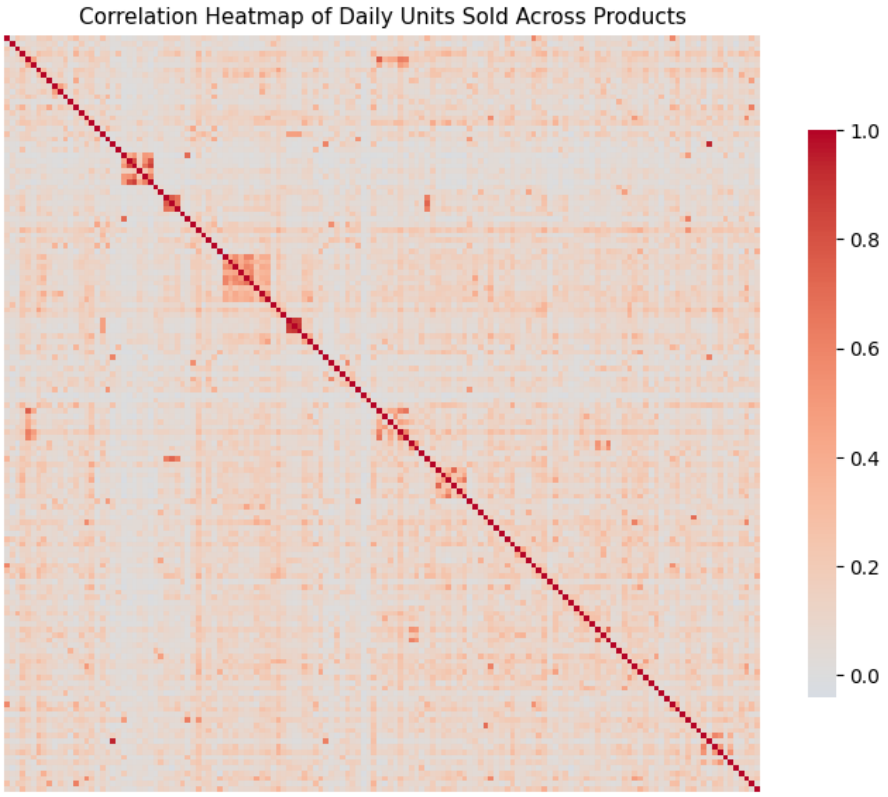


Figure 28: Correlation heatmap of daily units sold across products (raw units). The matrix is sparse, with only a few strong correlations.

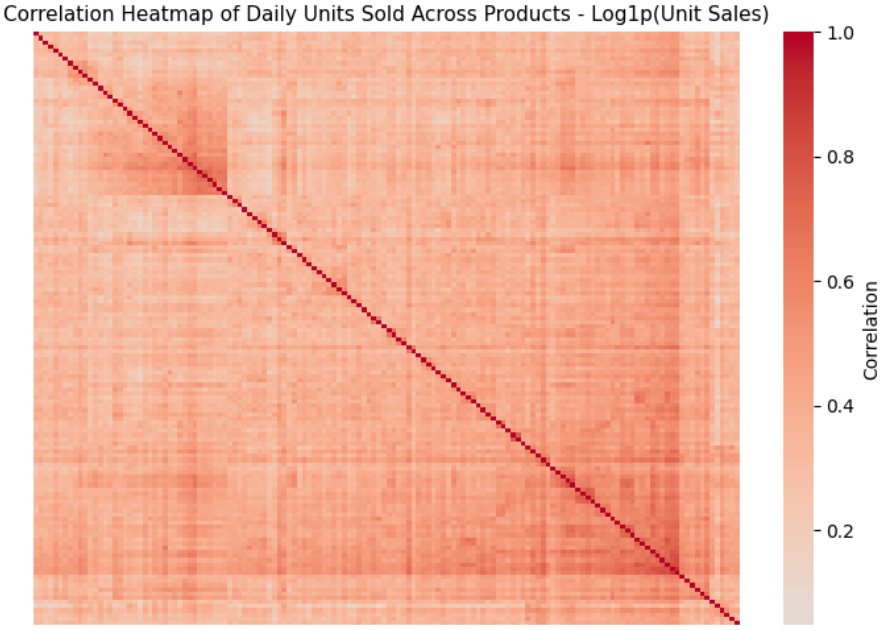


Figure 29: Correlation heatmap of daily units sold across products using log-transformed sales. This reveals more moderate correlations by dampening the impact of large spikes.

The raw-scale heatmap reveals a sparse correlation structure: most product pairs exhibit near-zero

correlation, with only a few isolated groups showing stronger association. However, this representation is heavily influenced by the right-skewed nature of retail sales—products with occasional high-volume days disproportionately dominate the correlation structure. In contrast, the log-transformed version produces a much denser and more interpretable structure, highlighting clusters of moderately correlated products. This transformation helps uncover subtler dependencies among items with low or variable sales volume, and is therefore more suitable for downstream tasks such as clustering or dimensionality reduction.

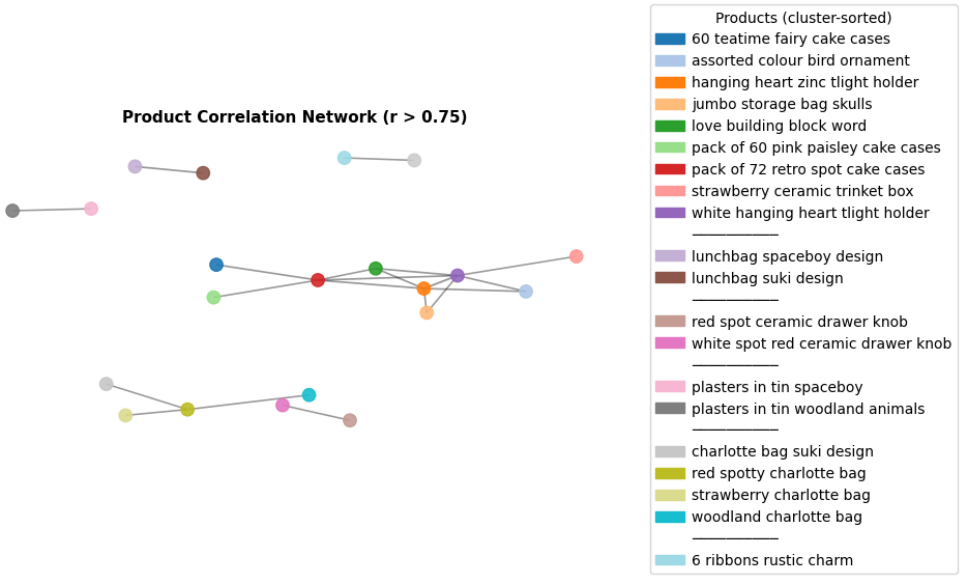


Figure 30: Product correlation network showing pairs of products with Pearson correlation above 0.75 in daily log-transformed sales. Each node represents a product, and edges connect strongly correlated pairs. Products are colored and grouped by cluster.

To identify groups of products with similar sales behavior, we construct a product correlation network using the log-transformed daily units sold. An edge is placed between two products if their Pearson correlation exceeds 0.75, indicating a strong positive relationship. The resulting graph is shown in Figure 30, where each node corresponds to a product, and edges represent high pairwise correlations.

Clusters of correlated products emerge naturally as connected components in the graph. Products within the same connected group are likely to respond similarly to underlying demand factors such as holidays, promotions, or seasonal trends. For instance, several items in the same category or intended for similar occasions (e.g., party accessories, lunchbags, or themed plasters) are grouped together.

This graph-based approach provides an interpretable structure for discovering co-moving product sets, which can inform forecasting strategies, bundling recommendations, or feature engineering for demand models. Clustering by correlation is particularly effective in the log-transformed space, where heavy-tailed volume effects are suppressed, allowing more consistent patterns to emerge.

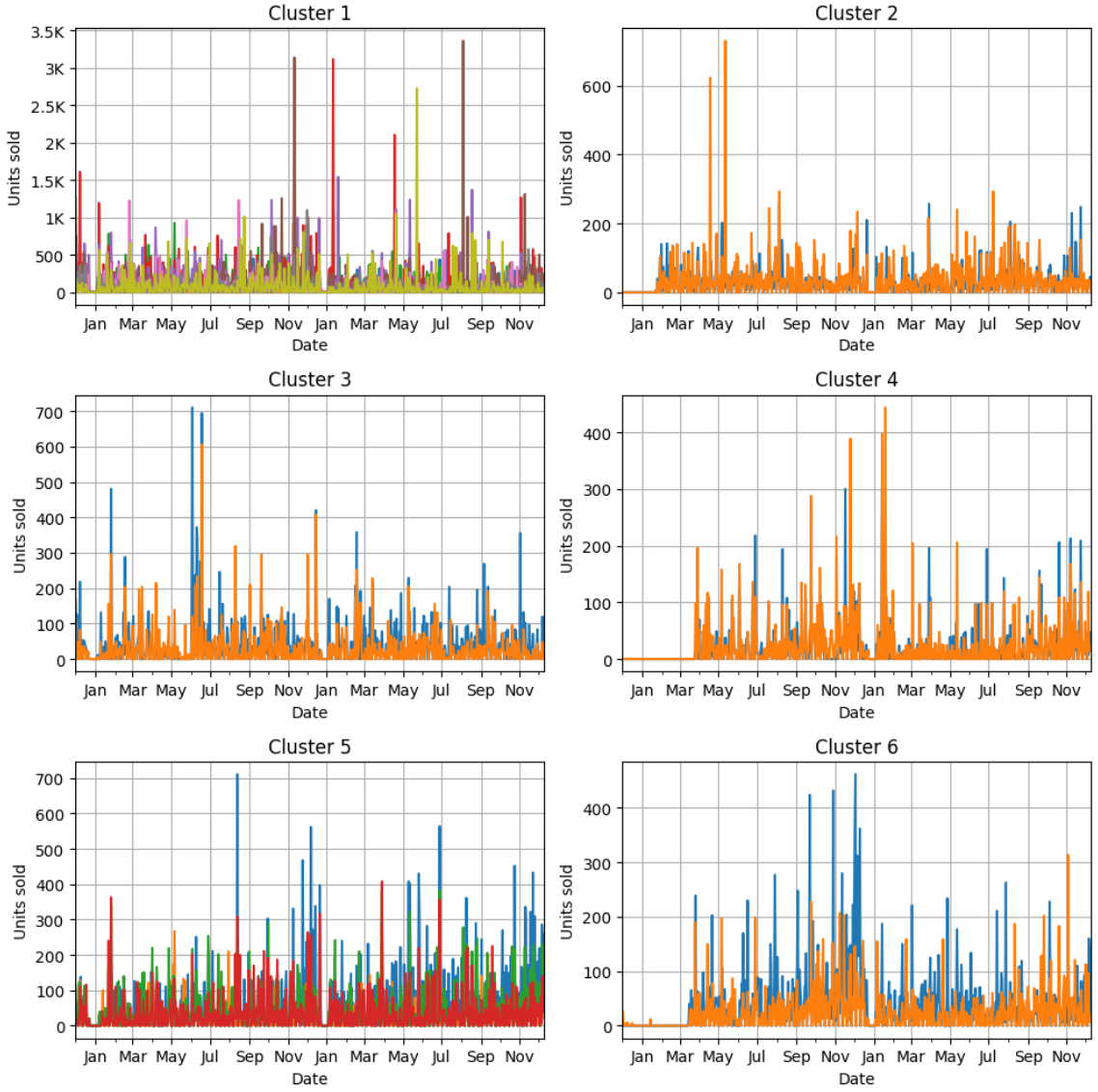


Figure 31: Daily unit sales over time for each product cluster formed from the correlation network. Each subplot corresponds to one cluster and shows the time series of all products in that group.

To better understand the temporal patterns that underlie the product clusters discovered in the correlation network, we plot the daily unit sales of products grouped by cluster in Figure 31. Each subplot corresponds to a different cluster, and includes time series from all products assigned to that group.

Several clusters display visually coherent behavior—products in the same cluster tend to rise and fall together over time, suggesting shared seasonality, promotional timing, or usage context. For example, some clusters show synchronized spikes, which could correspond to promotional events or holiday-driven surges. Other clusters exhibit steadier, low-variance trends, indicating more consistent demand.

3.3.5 Sensitivity to explanatory variables

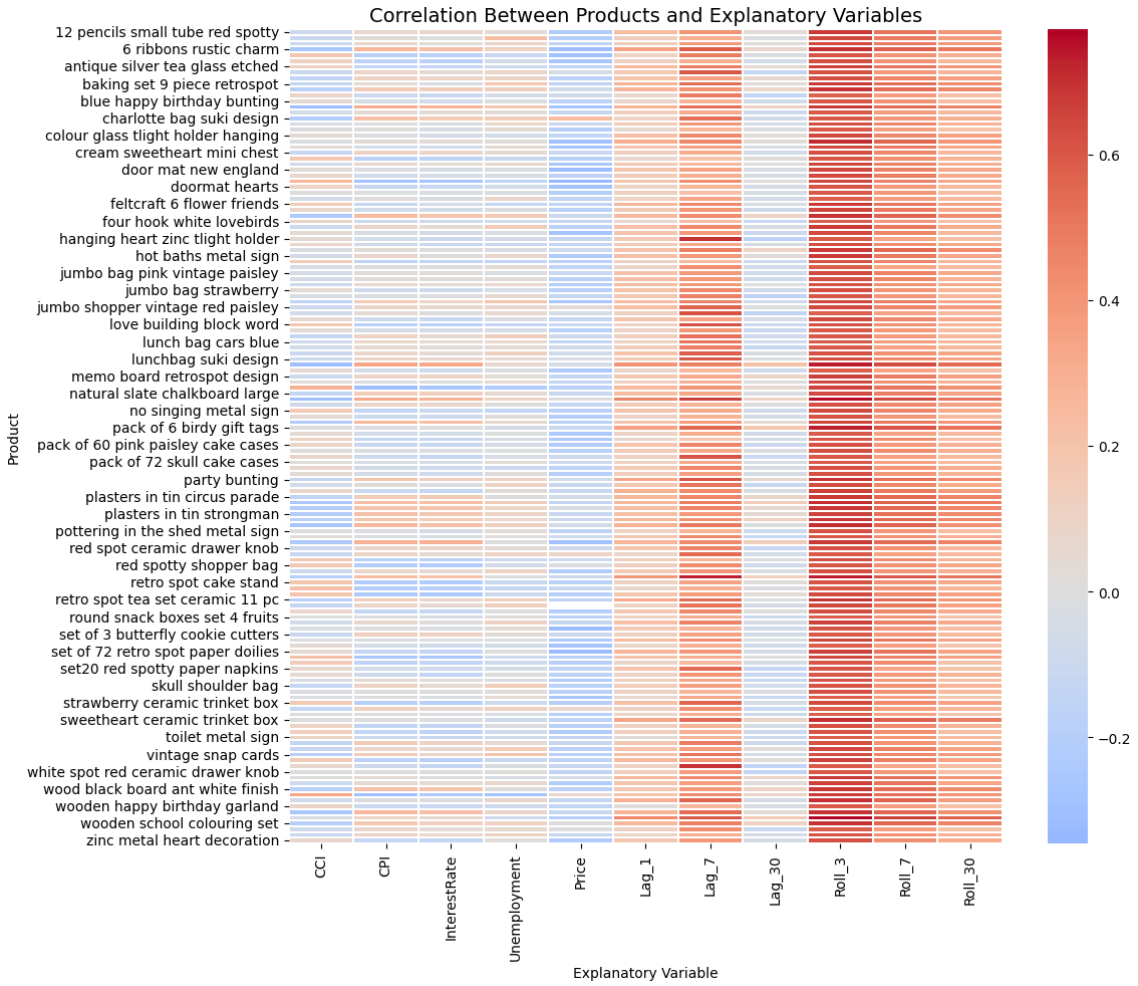


Figure 32: Correlation between daily log-transformed unit sales and explanatory features, including macroeconomic indicators, product-specific price, and short-term lagged or rolling sales statistics.

To explore the sensitivity of product demand to different explanatory factors, Figure 32 presents the Pearson correlation between daily log-transformed unit sales and a set of external and temporal features. The variables considered include macroeconomic indicators (Consumer Confidence Index, Consumer Price Index, interest rate, and unemployment rate), product-specific prices, recent sales lags (1, 7, and 30 days), and rolling averages over the past 3, 7, and 30 days.

The most prominent pattern is the strong and positive correlation between sales and recent performance metrics. In particular, the 3-day and 7-day rolling averages show consistently high correlations across nearly all products, reflecting the inherently autoregressive structure of retail demand: recent sales volumes are often the best predictor of future activity. One-day and one-week lags are also informative, though their effect varies more by product.

Price exhibits a broadly negative relationship with units sold, as expected—higher prices are generally associated with lower demand. This aligns with basic economic principles and highlights price as a reliable predictor in forecasting models.

Macroeconomic variables, on the other hand, show weaker and less consistent associations. Unemployment is mostly uncorrelated with sales at the product level. CPI and interest rate are mildly positively correlated with demand, while consumer confidence tends to have a slight negative correlation. These patterns may seem counterintuitive and likely reflect indirect effects from seasonal or promotional cycles, rather than direct causal relationships.

In summary, while external variables offer some marginal value, the most predictive signals for product-level sales are internal: price and recent demand. This underscores the importance of including lagged and rolling features when modeling retail time series, while using macroeconomic context to complement short-term dynamics.

4 Clustering

As introduced in the background research, each model is tested using various clustering methods, as well as without clustering when applicable. We used Google's Gemini 2.5 Pro with 1,048,576 tokens as the max context window to convert the product descriptions into vector embeddings.

Before converting each product description into a vector, we applied a number of cleaning measures to remove any extra noise that could affect the clustering. In particular, we removed colors, numbers, punctuation, and special characters from the product description strings. We also removed English stop words (common words such as 'with', 'of', and 'in') since these words don't provide meaning to the product descriptions. Lastly, we also lemmatized the words, essentially reducing the different forms of a word to one single form, for example, "builds", "building", and "built" would all become "build." This process would ensure that different forms of the same word would all be converted to the same base word before embedding. As an example, "pack of 60 dinosaur cake cases" became "pack dinosaur cake cases" after the cleaning procedure.

We implemented and compared several different clustering techniques and compared them using silhouette score. The silhouette score is a metric used in cluster analysis to measure how well each data point is grouped within its cluster compared to other clusters. It evaluates the "goodness" of clustering by considering the similarity among data points within the same cluster and how different it is from other clusters [25]. For each clustering technique, we evaluated its silhouette score in its base form and also after applying both PCA (Principal Component Analysis) and UMAP (Uniform Manifold Approximation and Projection) dimensionality reduction. We also performed hyperparameter tuning to achieve the best possible silhouette score and clusters for our models.

4.1 K-Means

K-means is an unsupervised learning algorithm that groups unlabeled data points into groups or clusters. This algorithm is one of the most popular clustering methods used in machine learning because it is efficient, effective, and simple. The k-means clustering algorithm is an example of an exclusive or "hard" clustering method since each data point can only exist in just one cluster.

K-means is an iterative, centroid-based clustering algorithm that partitions a datset into similar groups based on the distance between their centroids. The centroid is the center of the cluster and can be either the mean or median of all the points within the cluster depending on the characteristics of the data.

This algorithm categorizes data points into clusters using a mathematical distance measure, usuallly euclidean, from the cluster center. The goal of this algorithm is to minimize the sum of distances between data points and their assigned clusters. The k represents the number of clusters and we found the most optimal k by using both the elbow method and silhouette scores to find a k that balances performance and number of clusters [26].

The elbow method plots the inertia, which measures how well a dataset has been clustered based on distance metrics. It's calculated by measuring the distance between a datapoint and its centroid, squaring the distance and summing those squares for each data point in the cluster. A lower inertia means that the datapoints within the cluster are compact or more similar. The elbow method helps us choose a k where increasing the k does not add a significant benefit and where the inertia is low enough for good performance[27].

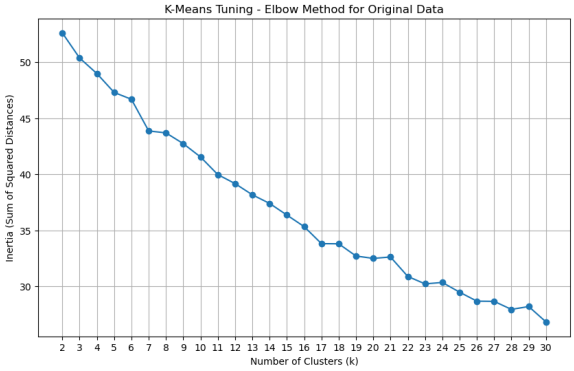


Figure 33: K-Means Elbow Method on Original Data

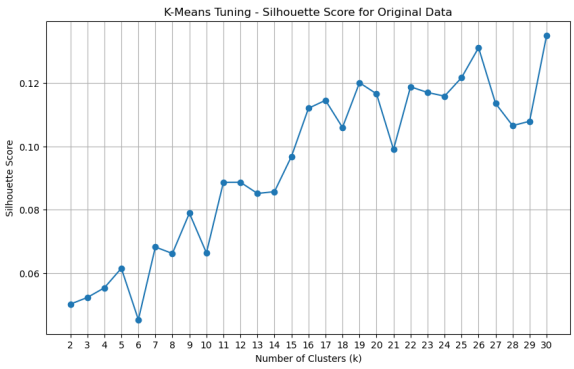


Figure 34: K-Means Silhouette Scores on Original Data

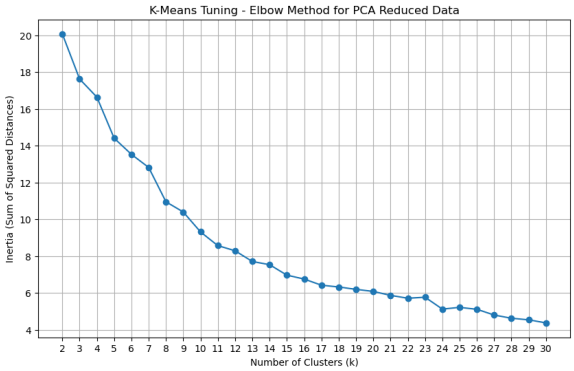


Figure 35: K-Means Elbow Method on PCA Reduced Data

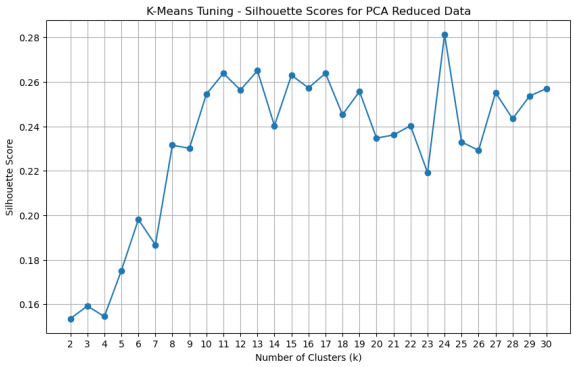


Figure 36: K-Means Silhouette Scores on PCA Reduced Data

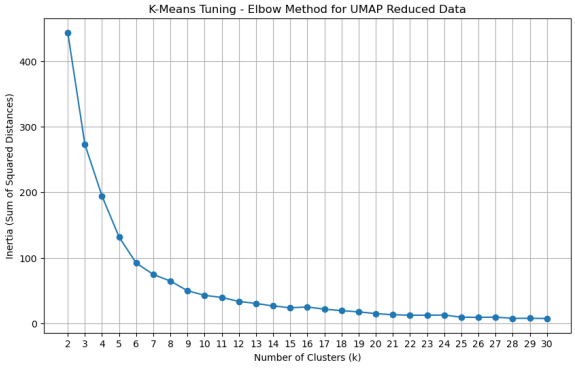


Figure 37: K-Means Elbow Method on UMAP Reduced Data

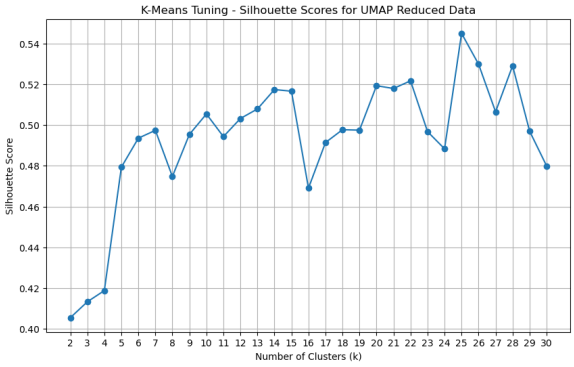


Figure 38: K-Means Silhouette Scores on UMAP Reduced Data

Overall, the k-means clustering method is a good foundational algorithm to start with for our experiments with clustering. It is simple, fast, and scalable and generally does well. However, outliers have a significant impact on k-means clustering and the algorithm performs poorly when the dataset is highly dimensional. Since our vector embeddings are highly dimensional (over 700 dimensions), we noted that the k-means algorithm would work best if performed on the dimensionally reduced data (after PCA or UMAP).

Some of the resulting clusters demonstrated strong internal cohesion, grouping highly similar items based on both semantics and usage context. For instance, Cluster 12 exclusively contained various themed door mats, such as “door mat fairy cake,” “door mat union flag,” and “doormat hearts,” indicating a high degree of topical and functional similarity. Similarly, Cluster 2 was composed entirely of decorative metal signs, including items like “bathroom metal sign,” “gin tonic diet metal sign,” and “hot baths metal sign.” These examples suggest that the clustering process, particularly after dimensionality reduction and parameter tuning, was effective at identifying coherent product groupings that reflect real-world categories.

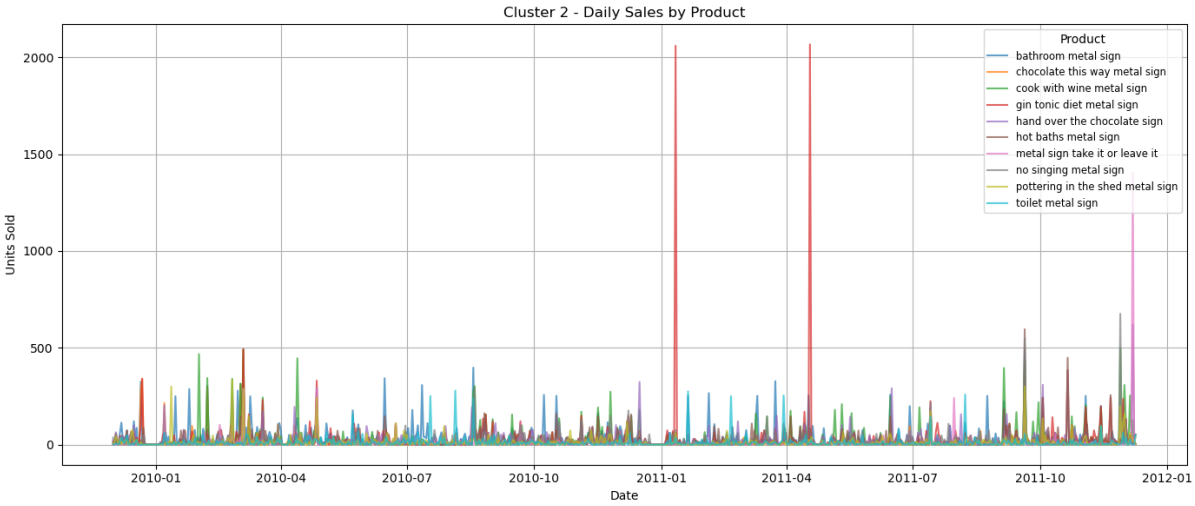


Figure 39: K-Means Cluster 2 (Signs) Sales

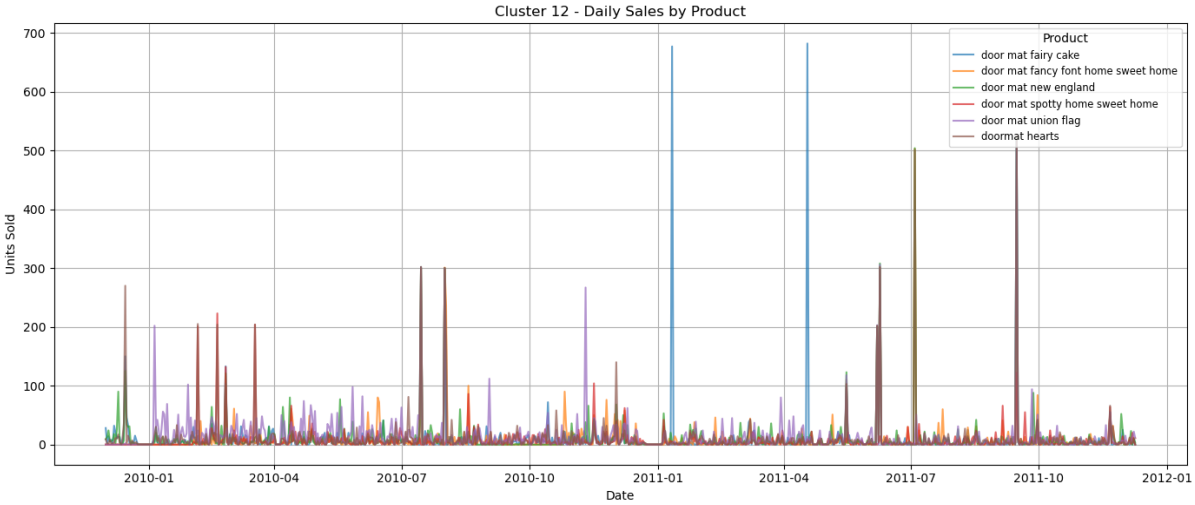


Figure 40: K-Means Cluster 12 (Doormats) Sales

Initial attempts using K-Means clustering alone yielded poor results, failing to capture meaningful groupings within the dataset. However, performance improved significantly after applying dimensionality reduction with UMAP, which preserved the global and local structure of the data. To further refine clustering quality, the optimal number of clusters was determined using the elbow method. With these enhancements, K-Means produced well-separated clusters, achieving a silhouette score of 0.5079, indicating a moderate but meaningful cluster structure.

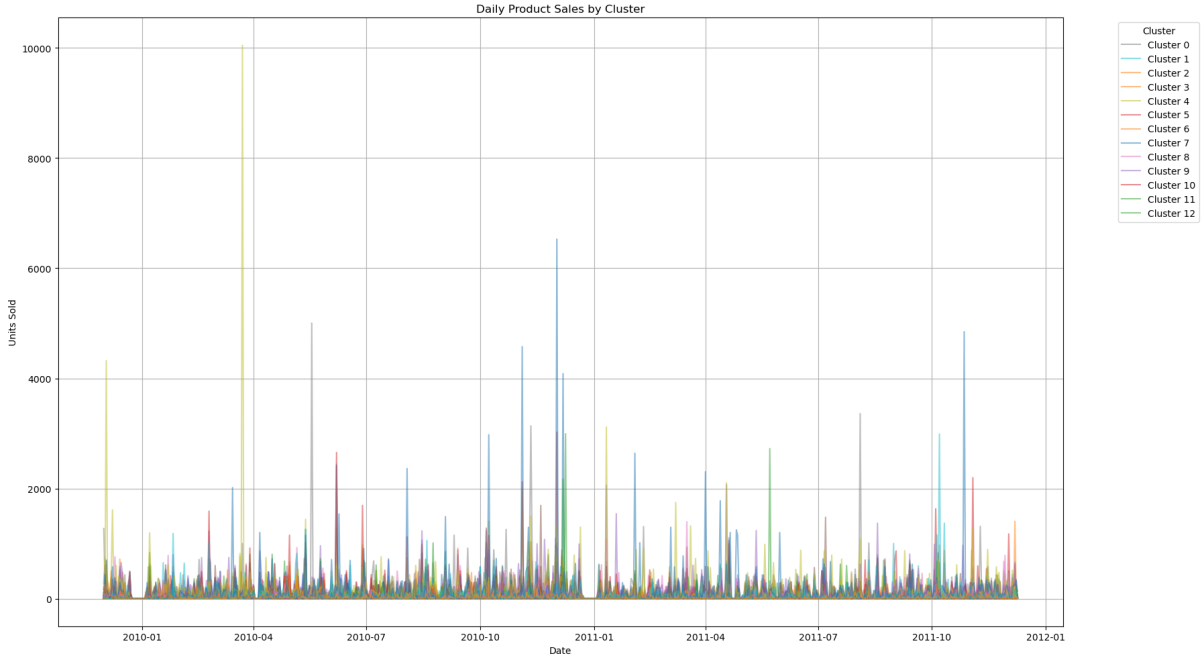


Figure 41: K-Means Cluster Total Sales

4.2 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a widely used density-based clustering algorithm that excels at identifying clusters of arbitrary shape and handling noise in datasets. Unlike partition-based algorithms, like K-Means, DBSCAN does not require the user to predefine the number of clusters. Instead, it relies on two key parameters: ϵ (epislon), which defines the radius of the neighborhood around a datapoint, and min_samples, the minimum number of points required within that radius to form a dense region.

The algorithm classifies points into three categories:

1. **Core points**: points that have at least min_samples within their ϵ-neighborhood
2. **Border points**: points which fall within the ϵ-neighborhood of a core point but do not satisfy the min_samples condition themselves
3. **Noise points**: points that are neither core nor border points

Two important concepts in DBSCAN are direct density reachability, where one point lies within the ϵ-neighborhood of a core point, and density connectivity, which links multiple points through a chain of density-reachable steps. These core ideas help DBSCAN discover clusters based on the spatial distribution of data.

The algorithm begins by identifying all core points, expanding clusters by connecting directly and transitively reachable points, and finally assigning border points to the nearest clusters. Noise points remain unassigned, which is a disadvantage for using this algorithm to cluster our products since we need all products to be assigned to a cluster.

DBSCAN is very robust and can deal with outliers. It is great at detecting non-linear and irregular cluster shapes with minimal parameter tuning. However, it struggles with datasets containing clusters of varying densities and lacks a way for predicting cluster membership for new data points [28].

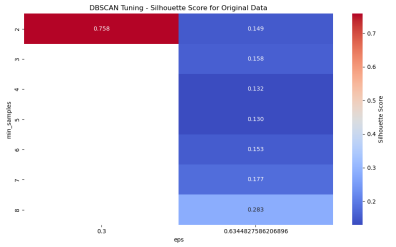


Figure 42: DBSCAN Silhouette Heat Map on Original Data

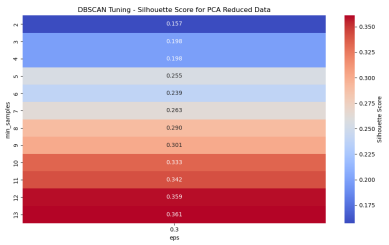


Figure 43: DBSCAN Silhouette Heat Map on PCA Reduced Data

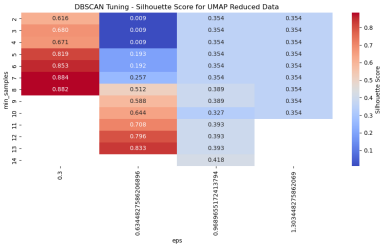


Figure 44: DBSCAN Silhouette Heat Map on UMAP Reduced Data

Beyond the highly cohesive clusters of doormats and decorative metal signs that K-Means also grouped, DBSCAN also successfully isolated other tightly grouped product categories. For example, it formed distinct clusters consisting exclusively of lunch bags and plasters, indicating that it effectively captured subtle but meaningful patterns in product descriptions and features. These results demonstrate DBSCAN's strength in uncovering niche groupings that may be overlooked by centroid-based methods, making it a valuable tool for fine-grained clustering tasks where intra-cluster similarity is prioritized over global separation.

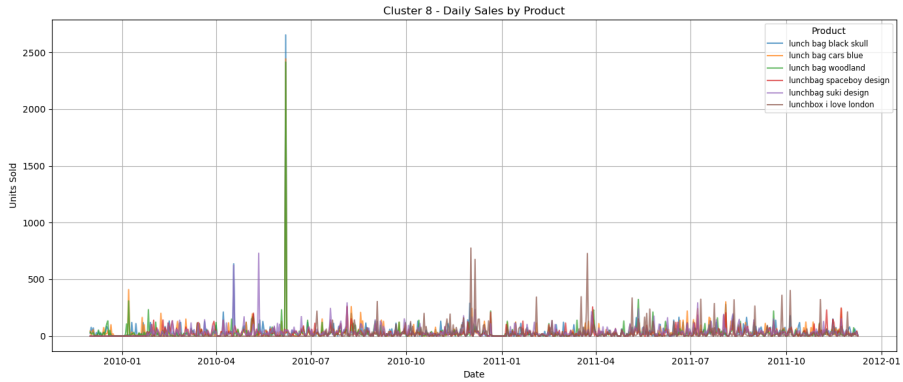


Figure 45: DBSCAN Cluster 8 (Lunch Bags/Boxes) Sales

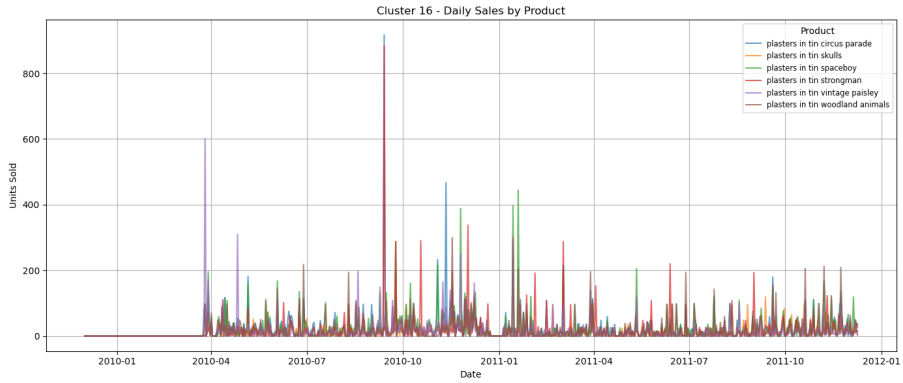


Figure 46: DBSCAN Cluster 16 (Plasters) Sales

Overall, DBSCAN produced its most effective clustering results when applied to the UMAP-reduced data, outperforming its performance on both the original high-dimensional dataset and the PCA-reduced version. UMAP preserved the local and global structure of the data in a way that complemented DBSCAN's density-based approach, enabling clearer identification of dense regions and natural cluster

boundaries. On UMAP-transformed data, DBSCAN yielded the highest number of successful hyperparameter combinations, forming well-defined clusters across a broader range of epsilon and min_samples values. This was evident in the heatmap visualizations, where blank cells indicated models that were skipped due to all points being classified as noise or forming only a single cluster. Notably, when excluding the noise cluster (labeled as -1), DBSCAN achieved a high silhouette score of 0.884, indicating strong intra-cluster similarity and inter-cluster separation among the retained points. However, the algorithm's overall effectiveness remains highly sensitive to hyperparameter selection, and it tends to struggle with datasets containing clusters of varying density. These factors highlight that while DBSCAN can uncover highly coherent and meaningful groupings—especially when paired with effective preprocessing like UMAP—it still requires careful tuning and is best suited for datasets with density structures that match its assumptions.

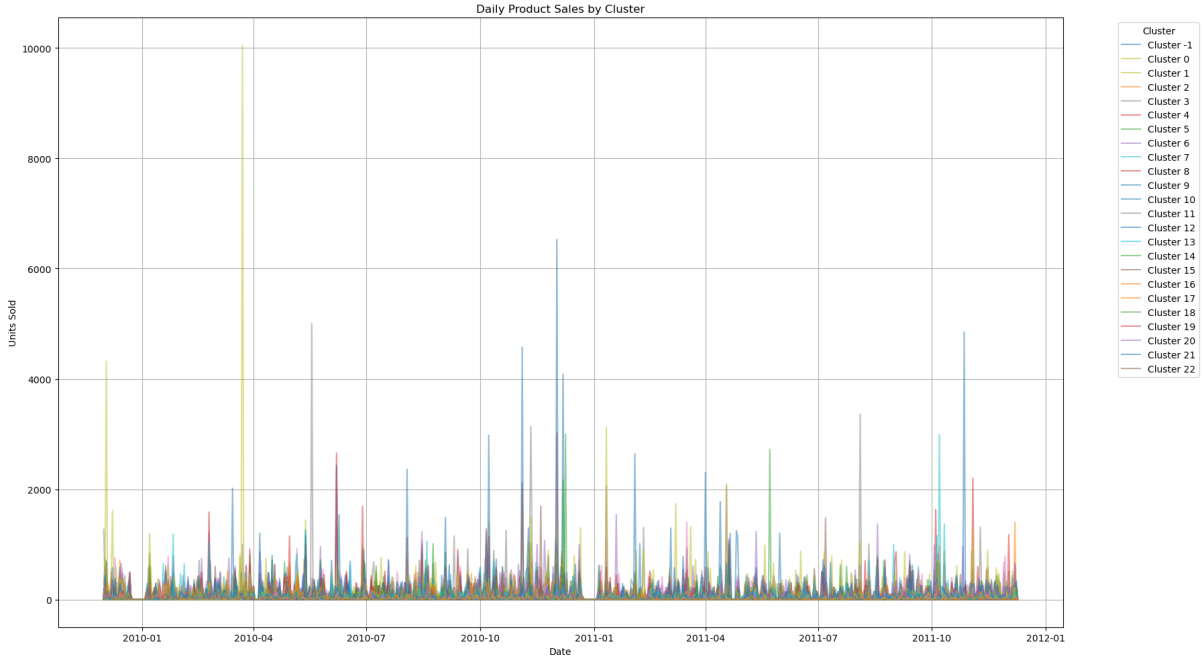


Figure 47: DBSCAN Clusters Total Sales

4.3 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) extends the DBSCAN algorithm by introducing a hierarchical approach that enables the discovery of clusters with varying densities. Like other density-based methods, HDBSCAN identifies clusters as contiguous regions of high density separated by regions of low density, and it excels at detecting arbitrarily shaped clusters while naturally handling noise. However, unlike DBSCAN, which requires a fixed density threshold, HDBSCAN builds a hierarchy of clusters by constructing a minimum spanning tree (MST) of mutual reachability distances. It then condenses this hierarchy and selects the most stable clusters—those that persist across multiple levels of density—without requiring a predefined number of clusters.

HDBSCAN proved especially useful in identifying complex, high-resolution cluster structures within the data, particularly after dimensionality reduction using UMAP. Its ability to model clusters of different densities made it more flexible than DBSCAN or K-Means, especially in heterogeneous datasets. Nevertheless, HDBSCAN has some limitations. It can be computationally expensive due to the MST and hierarchy construction, and it does not guarantee that all data points will be assigned to clusters—some are labeled as noise if they do not fit well into any dense region. Additionally, like most clustering algorithms, HDBSCAN may struggle with high-dimensional data unless preceded by effective dimensionality reduction. Despite these trade-offs, HDBSCAN remains a powerful and robust clustering technique, particularly well-suited for exploratory data analysis and applications where noise handling and density variation are critical [29].

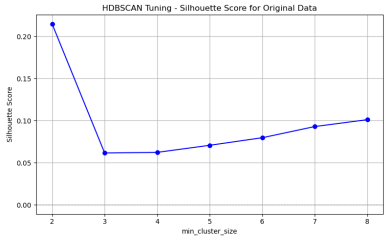


Figure 48: HDBSCAN Silhouette Scores on Original Data

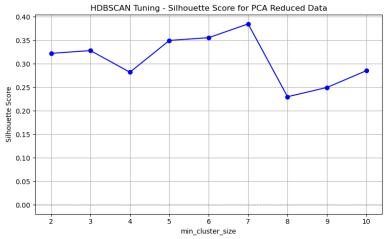


Figure 49: HDBSCAN Silhouette Scores on PCA Reduced Data

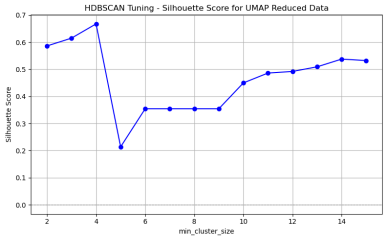


Figure 50: HDBSCAN Silhouette Scores on UMAP Reduced Data

HDBSCAN performed well overall, achieving a silhouette score of 0.6672, which was notably higher than that of K-Means (0.5209) and indicative of strong intra-cluster cohesion and inter-cluster separation. This result highlights HDBSCAN's ability to form well-defined clusters by adapting to local density variations without requiring a predefined number of clusters. However, its performance was still slightly behind DBSCAN, which achieved a silhouette score of 0.884 when excluding noise points. While HDBSCAN's hierarchical approach allowed it to capture meaningful structure in the data and handle clusters of varying densities more gracefully than K-Means, it may have been slightly less precise than DBSCAN in isolating the most coherent groupings under optimal parameter conditions.

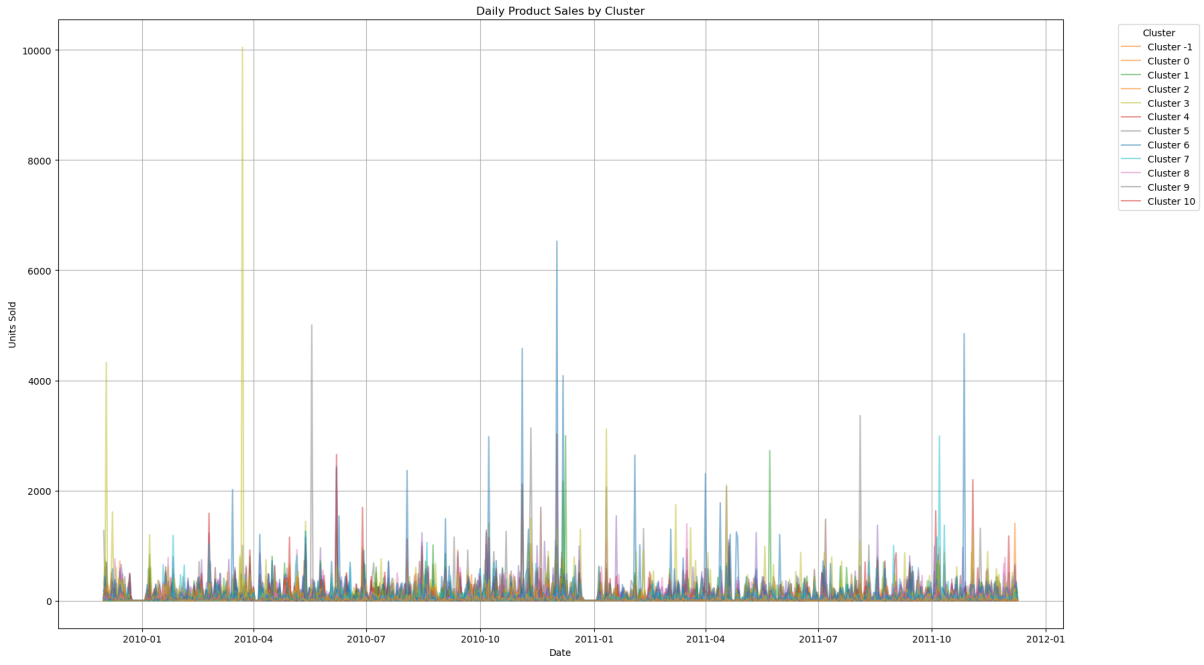


Figure 51: HDBSCAN Clusters Total Sales

HDBSCAN is a powerful and flexible approach to clustering, particularly well-suited for complex datasets with clusters of varying shapes and densities. Its key strengths lie in its ability to automatically determine the number of clusters, handle noise effectively by identifying and excluding outliers, and adapt to local density variations—advantages that make it especially valuable for exploratory data analysis. Additionally, its hierarchical clustering framework allows for more nuanced cluster formation compared to flat clustering methods like K-Means. However, HDBSCAN is not without limitations. It can be computationally intensive due to the construction of a minimum spanning tree and cluster hierarchy, and it does not guarantee that all data points will be assigned to a cluster, which may be undesirable in certain applications. Moreover, while it generally handles high-dimensional data better than some methods, it can still suffer from the curse of dimensionality if not paired with effective dimensionality reduction techniques. Overall, HDBSCAN is a robust and insightful clustering tool, particularly when

used in combination with techniques like UMAP to reveal meaningful structure in complex data.

4.4 Spectral Clusters

Spectral clustering is a graph-based clustering algorithm that identifies clusters by leveraging the eigenvalues and eigenvectors of a graph Laplacian derived from a similarity matrix. Instead of clustering the data points directly in their original space, spectral clustering first transforms the data into a lower-dimensional space where the structure of the data becomes more apparent. This is done by constructing a similarity graph between points, computing the Laplacian matrix of the graph, and then using the top eigenvectors of this matrix to embed the data in a new space. K-Means is then applied to these embedded points to form the final clusters. One of the key strengths of spectral clustering is its ability to detect non-convex and arbitrarily shaped clusters, which often outperform traditional algorithms like K-Means when the underlying structure of the data is complex. It is also relatively simple to implement using standard linear algebra tools. However, spectral clustering has limitations, including sensitivity to the construction of the similarity graph and the choice of the number of clusters k, which must be specified in advance. Additionally, it can be computationally intensive for large datasets due to the eigen decomposition step. Despite these challenges, spectral clustering remains a powerful technique, especially when the relationships among data points are better captured by their connectivity than by geometric proximity alone [30].

Figure 52: Spectral Silhouette Scores on Original Data

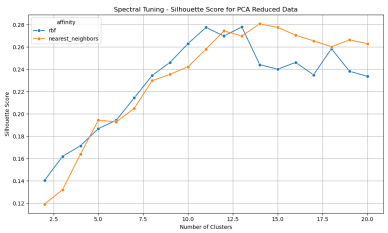


Figure 53: Spectral Silhouette Scores on PCA Reduced Data

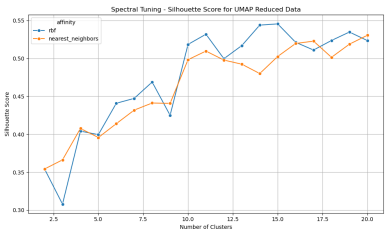


Figure 54: Spectral Silhouette Scores on UMAP Reduced Data

Spectral clustering performed reasonably well in clustering product descriptions, ultimately just barely outperforming K-Means in terms of silhouette score and qualitative coherence of clusters. However, its performance improved significantly when combined with dimensionality reduction techniques. PCA enhanced the separability of the data, resulting in clearer clustering structure, while UMAP further improved results by preserving both local and global relationships more effectively, aligning well with the connectivity-based nature of spectral clustering. In terms of similarity graph construction, we experimented with both RBF (Radial Basis Function) kernels and k-nearest neighbors (k-NN) graphs. The RBF kernel builds a fully connected graph where similarity decays with distance, which can smooth over local structures and emphasize global cohesion. In contrast, the k-NN graph constructs edges only between each point and its k closest neighbors, focusing more tightly on local relationships and often resulting in sparser, more interpretable cluster boundaries. The choice between RBF and k-NN graphs significantly impacts performance, with k-NN generally performing better in capturing the local semantics of product embeddings. Despite its theoretical strengths, spectral clustering's gains over K-Means were modest in practice, likely due to the relatively high dimensionality and subtle differences among many product descriptions.

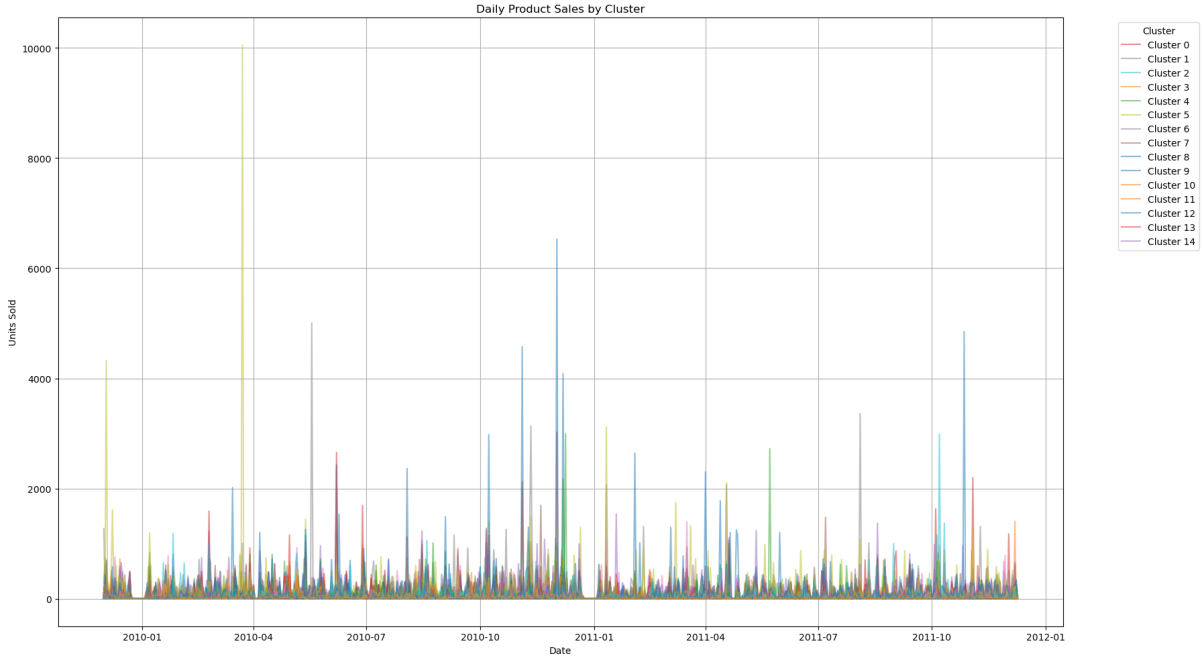


Figure 55: Spectral Clustering Total Sales

Spectral clustering is a versatile technique well-suited for identifying clusters with complex, non-linear boundaries—something that traditional algorithms like K-Means often struggle with. By transforming data into a graph representation and analyzing its structure through eigenvectors of the graph Laplacian, spectral clustering is particularly effective at capturing subtle connectivity patterns within the data. It performs especially well when used alongside dimensionality reduction methods such as PCA or UMAP, which enhance its ability to separate meaningful groupings. However, the method is not without its challenges. It requires prior knowledge of the number of clusters, can be computationally demanding due to its reliance on eigen decomposition, and is sensitive to how the similarity graph is constructed. The algorithm's effectiveness can also diminish in high-dimensional settings unless the data is carefully preprocessed. Despite its limitations, spectral clustering is still a strong choice when the relationships between data points are central to identifying meaningful clusters.

4.5 Correlation Clusters

An alternative approach to clustering products involved using the correlation of their sales patterns over time. The core idea was that products exhibiting similar fluctuations in sales—rising and falling together across weeks or months—likely share similar demand dynamics, seasonal trends, or purchasing contexts. By computing the pairwise correlation coefficients between product sales time series, we were able to construct a similarity graph where edges connected products whose correlation exceeded a predefined threshold. This method focuses not on product content or description, but on behavioral similarity, offering a data-driven way to uncover latent relationships in consumer behavior.

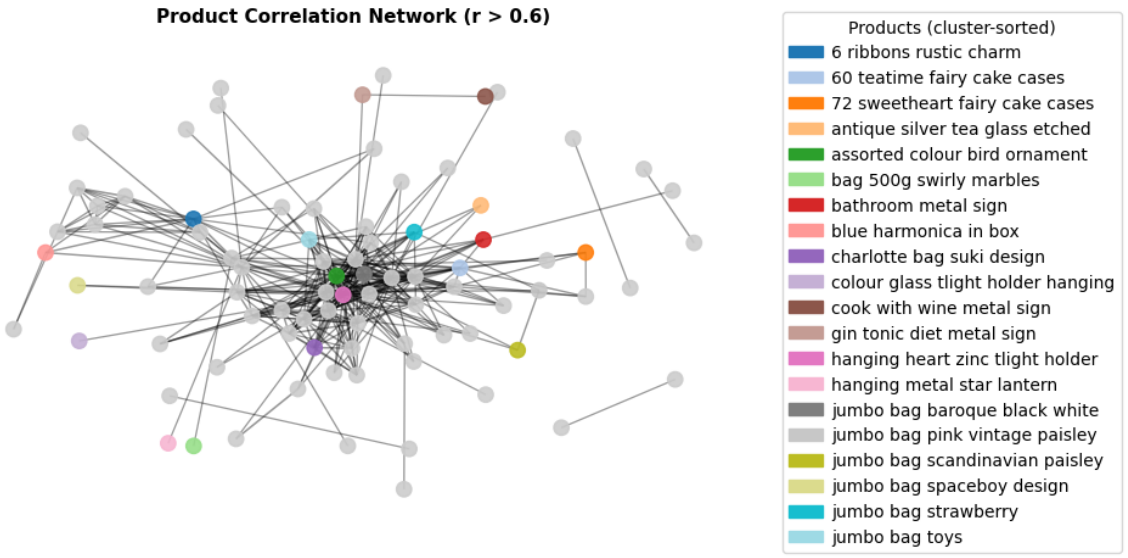


Figure 56: Correlation Clusters

This correlation-based clustering proved effective in identifying tight-knit groups of products with highly synchronized sales patterns, particularly when using higher correlation thresholds such as 0.6 or 0.8. At these levels, the resulting clusters were notably coherent—often grouping items likely to be purchased together or driven by similar external factors. However, a clear trade-off emerged between cluster precision and coverage. At a 0.6 threshold, only about a quarter of the products were clustered, and this number dropped sharply to just 7 products when the threshold was raised to 0.8. This limitation highlights that while correlation-based clustering excels at identifying highly aligned behaviors, it excludes a large portion of products that either exhibit more independent sales patterns or weaker correlations not strong enough to meet the threshold criteria.

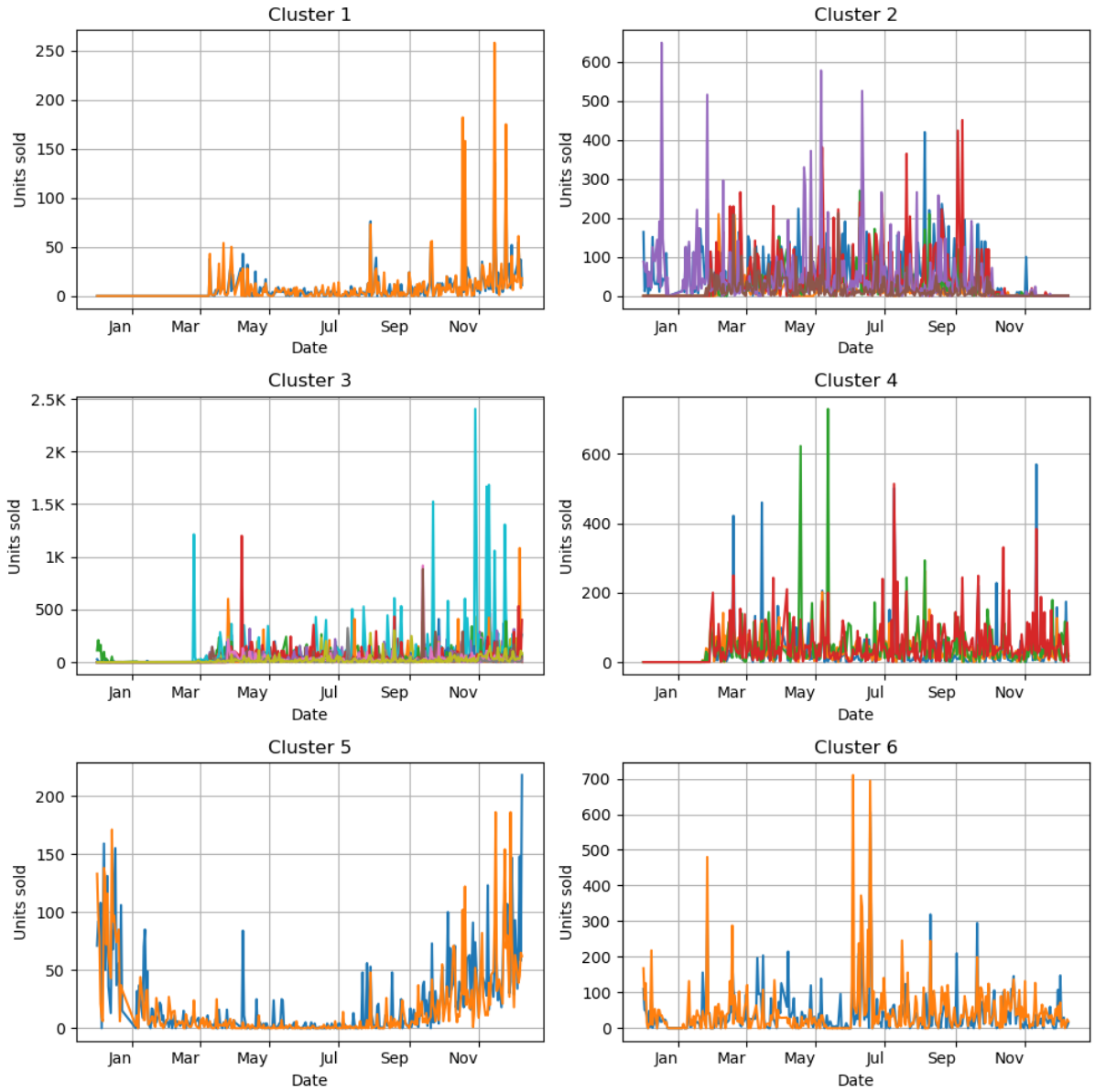


Figure 57: Correlation Clusters Behavior

4.6 Clustering Results

Clustering Method	Silhouette Score	Number of Clusters (n)
Our K-Means	0.5209	13
DBSCAN	**0.8840**	23
HDBSCAN	0.6672	10
Spectral Clustering	0.5267	15
Previous Group (K-Means)	0.0336	10

Table 3: Comparison of clustering methods by silhouette score and number of clusters

As summarized in Table 3, DBSCAN achieved the highest silhouette score of 0.8840, indicating well-separated and coherent clusters when noise points were excluded. HDBSCAN also performed strongly

with a silhouette score of 0.6672, outperforming both our optimized K-Means implementation (0.5209) and Spectral Clustering (0.5267). Notably, Spectral Clustering and K-Means produced a similar number of clusters (13–15), while DBSCAN and HDBSCAN identified more granular and specific groupings (23 and 10 clusters, respectively). The previous group's baseline K-Means implementation with a fixed k=10 achieved a significantly lower silhouette score of 0.0336, suggesting that both the clustering algorithm and preprocessing pipeline used in our study offered substantial improvements.

These improvements are partially due to enhanced product description cleaning and dimensionality reduction techniques. We removed stop words, numbers, colors, and special characters, and applied lemmatization to ensure more meaningful embeddings. Products with insufficient or incomplete sales history were also removed to ensure a cleaner dataset for both clustering and forecasting. Dimensionality reduction via UMAP and PCA proved crucial across all methods, with UMAP providing the most consistent gains—especially for DBSCAN and HDBSCAN, whose density-based mechanisms benefit from spatially meaningful embeddings.

5 Models

5.1 GLM

The Generalized Linear Model (GLM) is a flexible extension of ordinary linear regression that allows the response variable to follow distributions from the exponential family, making it well-suited for non-Gaussian targets such as count data. In this work, we use a Negative Binomial GLM with a log link function to forecast daily product-level sales.

The choice of a Negative Binomial distribution is motivated by the characteristics of the target variable—daily units sold—which is:

- Non-negative
- Discrete
- Often overdispersed (i.e., variance much greater than the mean).

Feature engineering plays a crucial role in this model as, unlike more powerful models, it cannot learn interaction between features by itself. The predictors include:

- **Lagged sales features**: Such as the previous day's sales, weekly lags, and rolling means over 3 and 7 days.
- **Calendar effects**: Indicators for day of the week, month, quarter, holidays, and weekends.
- **Macroeconomic indicators**: Including the Consumer Confidence Index (CCI), Consumer Price Index (CPI), interest rate, and unemployment rate.
- **Interaction terms**: Capturing nonlinear effects between lagged sales and calendar events (e.g., sales on holidays vs. weekdays).

We train **one model per product** because, as the analysis in the Product spot-checks section shows, the characteristics of each product's time series vary widely. Each model is fit independently using maximum likelihood estimation, as implemented in the `statsmodels` library. We train the models on the data from 2009-12-01 to 2011-10-09 (first 678 days) and evaluate the forecasts over the period starting on 2011-10-10 and ending on 2011-12-09 (60 days). Since the Negative Binomial GLM is a parametric model, no hyperparameter tuning is required. Instead, we focus on thoughtful feature construction and regularization through manual selection.

During inference, predictions are made autoregressively: predicted values are fed back as inputs to lagged features when forecasting multiple days ahead. We round predictions to the nearest integer and clip negative or infinite values.

To illustrate the structure of the trained model, Figure 58 shows a sample summary for one product's GLM.

Generalized Linear Model Regression Results

Dep. Variable:	UnitsSold	No. Observations:	678
Model:	GLM	Df Residuals:	642
Model Family:	NegativeBinomial	Df Model:	35
Link Function:	log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-2560.6
Date:	Wed, 07 May 2025	Deviance:	845.94
Time:	23:22:36	Pearson chi2:	1.24e+03
No. Iterations:	88	Pseudo R-squ. (CS):	0.7421
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
AvgPricePerUnitPounds	-0.6109	0.073	-8.361	0.000	-0.754	-0.468
CCI	0.6858	0.656	1.046	0.296	-0.600	1.971
CPI	0.3680	0.324	1.137	0.256	-0.266	1.002
InterestRate	-3.6411	3.022	-1.205	0.228	-9.563	2.281
Unemployment	0.4556	0.492	0.925	0.355	-0.510	1.421
DayBeforeUnitsSold	0.0001	0.000	0.220	0.826	-0.001	0.001
WeekBeforeUnitsSold	-0.0004	0.001	-0.792	0.428	-0.001	0.001
Rolling3DayMeanUnitsSold	-0.0002	0.001	-0.171	0.865	-0.003	0.003
Rolling7DayMeanUnitsSold	-0.0011	0.003	-0.420	0.674	-0.006	0.004
IsWeekend	-45.4169	2.11e+06	-2.15e-05	1.000	-4.13e+06	4.13e+06
IsHoliday	-0.1459	0.481	-0.304	0.761	-1.088	0.796
DayOfWeek_Monday	0.1082	0.155	0.697	0.486	-0.196	0.413
DayOfWeek_Saturday	-90.2420	4.22e+06	-2.14e-05	1.000	-8.26e+06	8.26e+06
DayOfWeek_Sunday	44.8251	2.11e+06	2.13e-05	1.000	-4.13e+06	4.13e+06
DayOfWeek_Thursday	0.1568	0.150	1.043	0.297	-0.138	0.452
DayOfWeek_Tuesday	0.3894	0.150	2.603	0.009	0.096	0.683
DayOfWeek_Wednesday	-0.1462	0.149	-0.979	0.328	-0.439	0.146
Month_August	-24.6290	24.470	-1.006	0.314	-72.590	23.332
Month_December	-25.3152	24.738	-1.023	0.306	-73.800	23.170
Month_February	-100.6145	98.074	-1.026	0.305	-292.837	91.608
Month_January	-99.9533	97.883	-1.021	0.307	-291.800	91.893
Month_July	-24.9384	24.457	-1.020	0.308	-72.873	22.996
Month_June	1.2130	0.297	4.089	0.000	0.632	1.794
Month_March	-100.1803	98.137	-1.021	0.307	-292.524	92.164
Month_May	0.5437	0.213	2.558	0.011	0.127	0.960
Month_November	-24.7016	24.466	-1.010	0.313	-72.653	23.250
Month_October	-25.0216	24.321	-1.029	0.304	-72.691	22.647
Month_September	-25.0565	24.457	-1.024	0.306	-72.992	22.879
Quarter_Q2	-100.3789	98.213	-1.022	0.307	-292.873	92.116
Quarter_Q3	-74.6240	73.384	-1.017	0.309	-218.453	69.205
Quarter_Q4	-75.0383	73.523	-1.021	0.307	-219.142	69.065
DayBeforeUnitsSold_x_IsWeekend	0.7002	3.95e+04	1.77e-05	1.000	-7.75e+04	7.75e+04
DayBeforeUnitsSold_x_IsHoliday	-0.0178	0.011	-1.659	0.097	-0.039	0.003
WeekBeforeUnitsSold_x_IsWeekend	-0.0160	0.005	-3.218	0.001	-0.026	-0.006
WeekBeforeUnitsSold_x_IsHoliday	0.0111	0.005	2.149	0.032	0.001	0.021
Rolling3DayMeanUnitsSold_x_IsWeekend	-0.0036	0.005	-0.726	0.468	-0.013	0.006
Rolling3DayMeanUnitsSold_x_IsHoliday	0.0257	0.017	1.493	0.135	-0.008	0.059
Rolling7DayMeanUnitsSold_x_IsWeekend	0.0009	0.004	0.227	0.821	-0.007	0.008
Rolling7DayMeanUnitsSold_x_IsHoliday	-0.0521	0.019	-2.706	0.007	-0.090	-0.014

Figure 58: Model summary for one product's Negative Binomial GLM. Coefficients represent the log-linear effect of each feature on the expected number of daily units sold. The table includes estimates, standard errors, z-values, and statistical significance.

5.2 Decision tree

The Decision Tree Regressor is a non-parametric model that recursively partitions the feature space to predict a continuous target variable. Unlike linear models, decision trees can automatically capture nonlinearities and high-order interactions without requiring explicit feature transformations. We trained one Decision Tree Regressor per product to forecast daily units sold.

The choice of decision trees is motivated by their flexibility and interpretability. They are especially useful in settings where:

- Relationships between features and the target are highly nonlinear.
- The importance of different features may vary significantly across products.
- We want models that are robust to monotonic transformations and invariant to the scale of the inputs.

Each model was trained on the same set of features and the same time window as that used for the GLM (see Section 5.1). We used the Squared Error Loss and the other default parameters of the `DecisionTreeRegressor` from sklearn. As with the GLM, inference is done autoregressively: predictions are recursively fed back into the lagged features when forecasting multiple days ahead. We round predictions to the nearest integer and clip negative or infinite values.

To gain insight into the structure of the trained trees and the relative influence of features, we visualize the top 3 levels of the fitted models for a sample of products (Figure 59), and we present pie charts of the top feature importances (Figure 60).

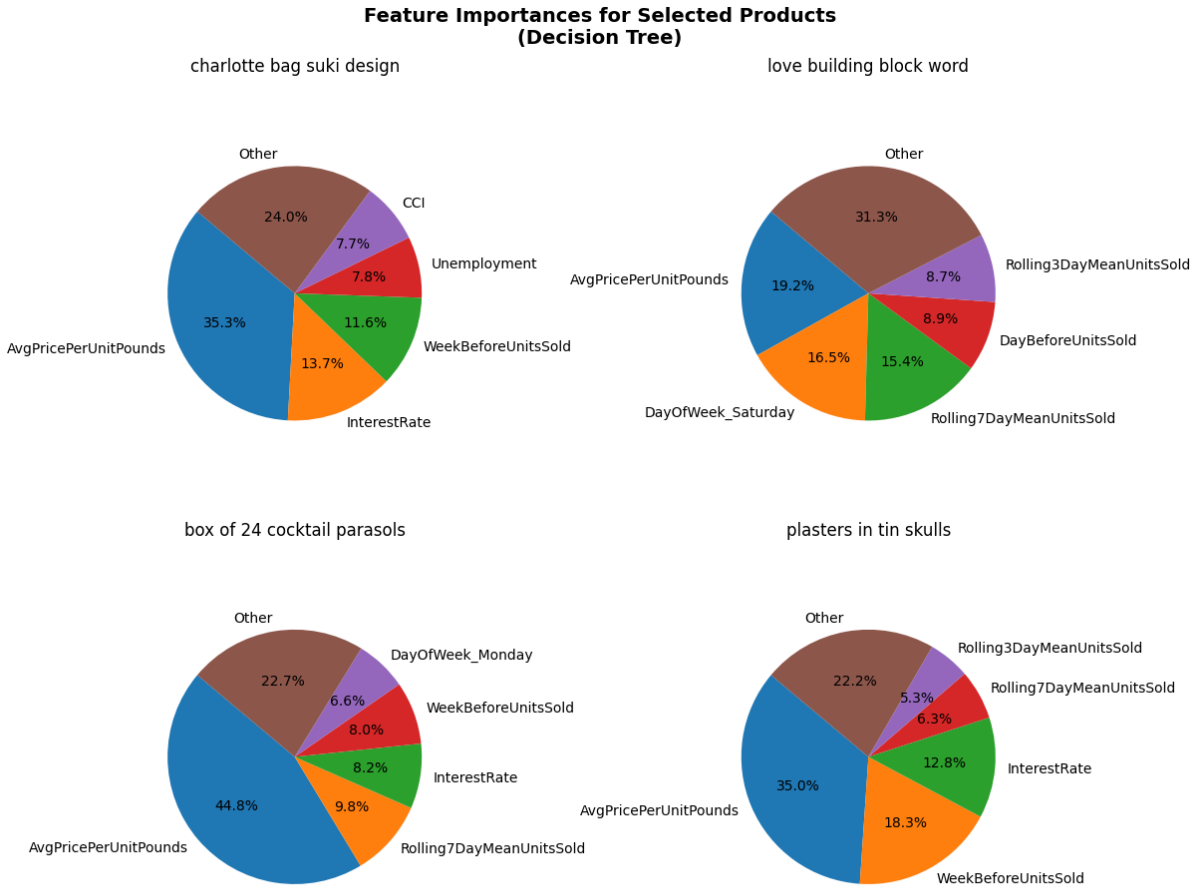


Figure 59: Top feature importances for four selected products. Each pie chart aggregates all but the five most important features into an "Other" category to highlight the key drivers of prediction. Across products, `AvgPricePerUnitPounds`, `WeekBeforeUnitsSold`, and `InterestRate` consistently appear as influential, though some products depend more on calendar features or rolling statistics. These differences support the decision to fit product-specific models.

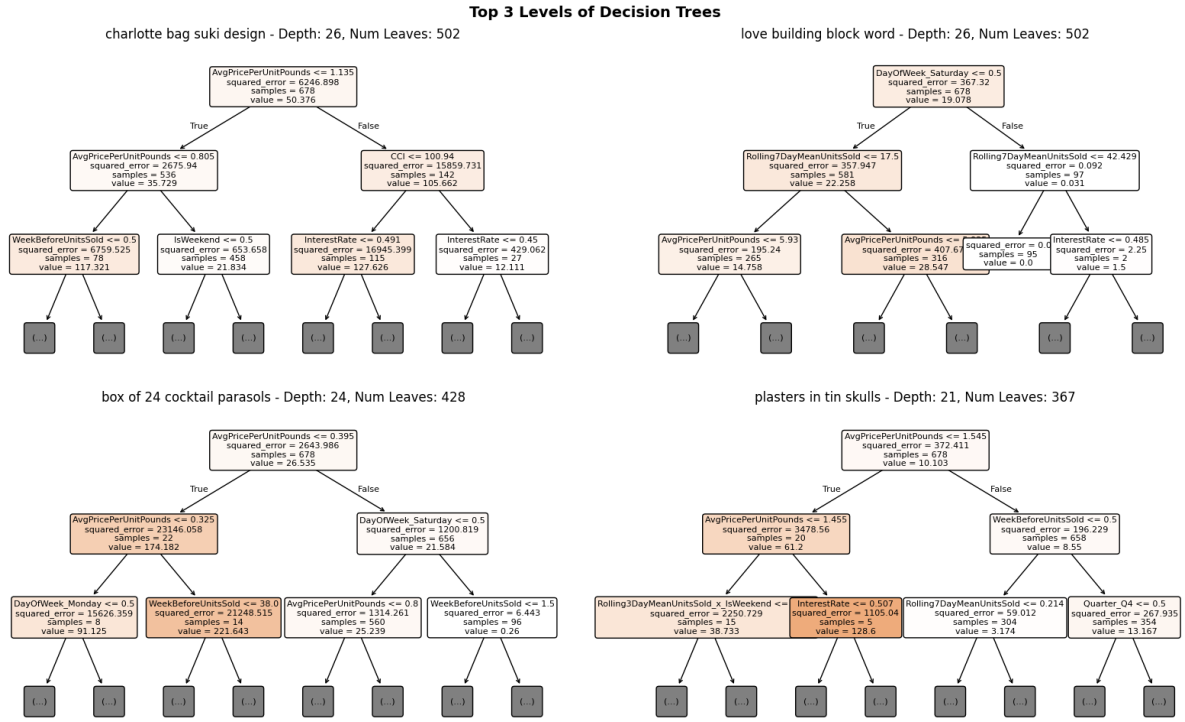


Figure 60: Top 3 levels of the trained Decision Tree Regressors for four sample products. Splits show the most important early decisions based on features such as price, economic indicators, and lagged sales. Despite being fit independently, many models rely on similar predictors (e.g., price and interest rate), though the specific thresholds and sequences of splits vary. This highlights the model's ability to adapt to product-specific patterns.

5.3 XGBoost

XGBoost (Extreme Gradient Boosting) is an ensemble method that builds a sequence of shallow decision trees, where each new tree attempts to correct the residuals of the previous ensemble. Compared to a single decision tree, XGBoost achieves higher predictive performance through additive boosting, shrinkage, and regularization. We trained one XGBoost regressor per product to forecast daily units sold.

The choice of XGBoost is motivated by its strong performance on structured tabular data and its ability to model complex, nonlinear interactions between features. Key advantages include:

- Automatic learning of high-order interactions without the need for manual feature engineering.
- Built-in regularization mechanisms to mitigate overfitting.
- Feature importance scores that provide interpretability of the trained model.

Each model was trained on the same set of features and over the same time window as that used for the GLM (see Section 5.1). To select hyperparameters, we used `Optuna` to perform product-specific optimization. For each product, we ran a 25-trial hyperparameter search minimizing mean squared error on a validation set. The search space included:

- `n_estimators` $\in [50, 300]$
- `max_depth` $\in [3, 10]$
- `learning_rate` $\in [0.01, 0.3]$

- `subsample`, `colsample_bytree` $\in [0.5, 1.0]$
- `gamma` $\in [0, 5]$
- `min_child_weight` $\in [1, 10]$

After tuning, we aggregated the best parameters across all products and selected the most frequently occurring value for each hyperparameter. This yielded a shared configuration that balances individual model performance with consistency across the catalog.

Table 4: Most common XGBoost hyperparameters across all product-specific tuning runs.

Parameter	Value	Description
`n_estimators`	50	Number of boosting rounds (trees) used in the ensemble.
`max_depth`	3	Maximum depth of each individual decision tree.
`learning_rate`	0.1982	Shrinkage factor applied to each tree to control overfitting.
`subsample`	0.9928	Fraction of training data sampled for each boosting round.
`colsample_bytree`	0.9950	Fraction of features randomly selected for each tree.
`gamma`	1.0786	Minimum loss reduction required to make a further partition (regularization).
`min_child_weight`	1	Minimum sum of instance weights needed in a child node.
`objective`	`reg:squarederror`	Loss function used for regression; here, squared error.
`random_state`	42	Seed for reproducibility of training results.

As with the GLM and Decision Tree models, inference is conducted autoregressively: predictions are recursively fed back into the lagged features when forecasting multiple days ahead. Final forecasts are rounded to the nearest integer and clipped to eliminate negative or infinite values.

To interpret the trained models, we visualize the feature importances for a sample of four products in Figure 61. The plots show that while feature importance patterns vary by product, several predictors frequently emerge as dominant—such as `AvgPricePerUnitPounds`, calendar features (e.g., `DayOfWeek_Saturday`, `Quarter_Q4`), and recent lagged sales. This supports the model's ability to adapt to product-specific demand drivers while leveraging general patterns in consumer behavior.

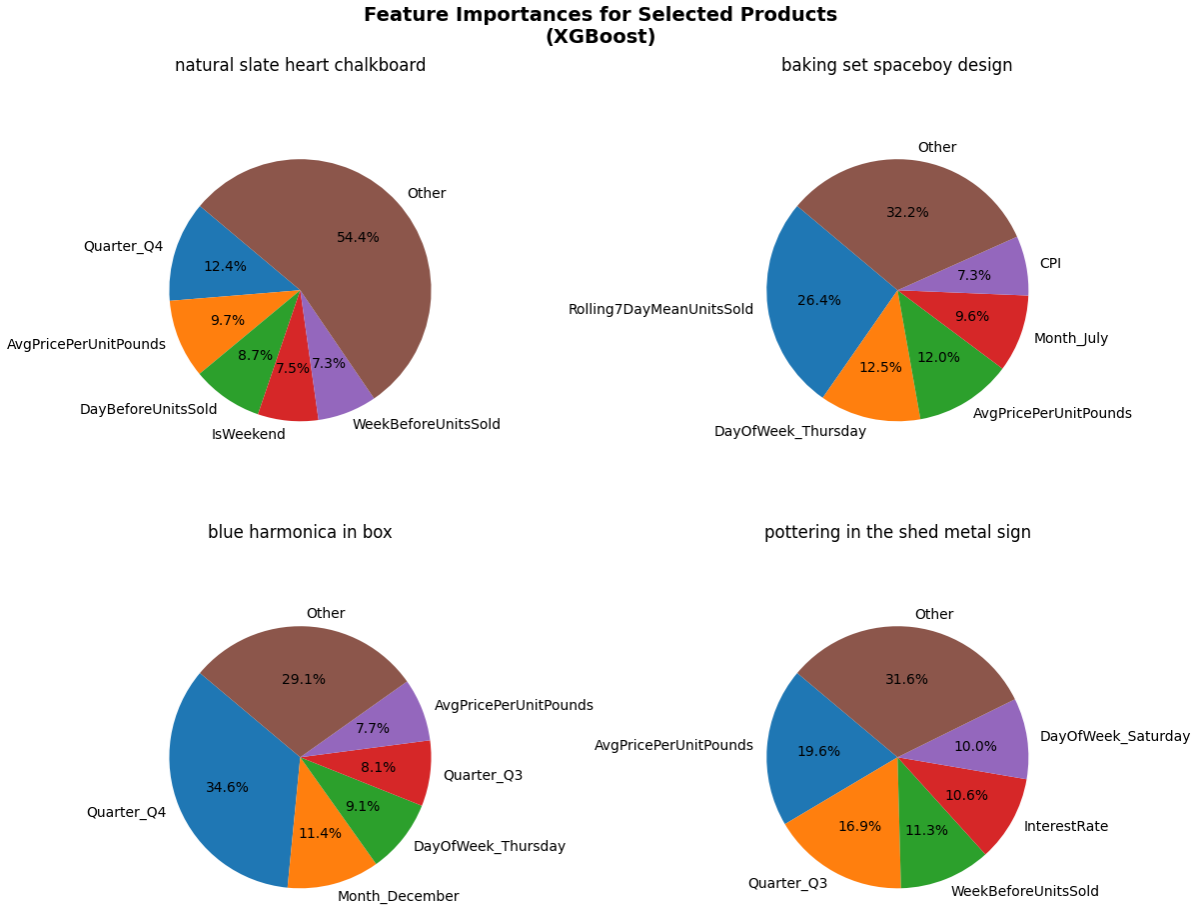


Figure 61: Top feature importances for four selected products using XGBoost. Each pie chart aggregates all but the five most important features into an "Other" category. The dominant features vary across products but often include price, calendar indicators, and recent sales dynamics.

5.4 LSTM

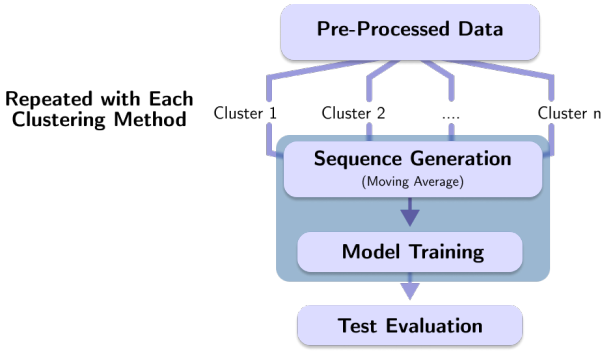


Figure 62: RNN Model Architecture

Figure 62 presents the architecture of the recurrent neural network (RNN) model implemented for time-series forecasting in this project. The model adopts a sequence-based approach by incorporating the previous 30 days of sales data (*UnitsSold*) as input to predict the sales value of the following day.

In addition to these temporal patterns, the model incorporates explanatory variables repeated at each time step to provide contextual information.

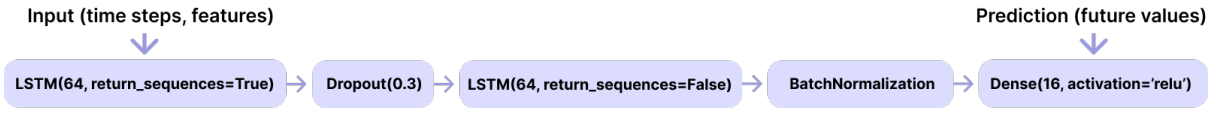


Figure 63: RNN Flow Diagram

As shown in Figure 63, the model begins with an LSTM layer containing 64 units, configured to return sequences so that time-step-level information is retained for downstream processing. This is followed by a dropout layer with a 0.3 rate to reduce the risk of overfitting by randomly deactivating 30% of the neurons during training. A second LSTM layer with 32 units is then applied, this time returning only the final hidden state, which condenses the temporal dynamics into a fixed-length vector. To stabilize training and accelerate convergence, a batch normalization layer is included after the LSTM stack. The output is passed through a fully connected dense layer with 16 neurons and ReLU activation, introducing non-linearity and enabling the model to learn complex relationships. Finally, a single-unit dense layer without activation generates the model's output, making it suitable for regression tasks such as forecasting future values. Overall, this architecture balances depth and regularization, making it well-suited for multivariate time series forecasting, where both short- and long-term dependencies are important.

To investigate the impact of segmentation on model performance, the LSTM model is trained independently on clusters generated using the four unsupervised clustering algorithms described in Section 3: K-Means, DBSCAN, HDBSCAN, and Spectral Clustering. For each cluster within each method, a separate LSTM-based RNN is trained for 10 epochs. This approach allows the model to specialize its predictions according to the unique behavioral patterns present within each cluster.

Model performance is evaluated using symmetric Mean Absolute Percentage Error (sMAPE), chosen for its robustness to scale and its effectiveness in handling small true values. This is further discussed in the analysis section. By assessing the model across all clustering methods and cluster assignments, we aim to evaluate both predictive accuracy and the consistency of performance across segmentation strategies.

Training is conducted over 30 epochs for each model. For example, Figure 64 illustrates the training process for each cluster under the K-Means clustering method. For each run, the model iteration with the lowest Mean Squared Error (MSE) loss is selected, and its corresponding sMAPE is evaluated. In most cases, both training sMAPE and MSE followed similar decreasing trends, while validation metrics also declined but exhibited more variability. Further investigation of the results and the prediction is discussed in the Results and Analysis.

5.5 Facebook Prophet

We implemented the Facebook Prophet model to predict daily product sales in a non-retail store in the UK. Prophet decomposes time-series data into three main components: trend, seasonality, and noise. The trend component is modeled using a piecewise linear function, capturing long-term increases or decreases in sales over time. Seasonality is handled using a Fourier series to detect and model recurring patterns, such as weekly cycles or annual holiday effects. This is particularly valuable in our context, where sales may be influenced by short-term trends and seasonal consumer behaviors.

To improve the model's prediction accuracy, we incorporated a variety of explanatory variables. These include temporal indicators like day-of-week binary indicators and public holidays, as well as external drivers such as the UK Consumer Confidence Index and Google search trends for "gift ideas." By integrating these regressors, the model captures both behavioral and economic signals that influence purchasing patterns beyond what trend and seasonality alone can explain.

Each Prophet model is trained independently on the clusters derived using the four unsupervised learning algorithms described in Section 3: K-Means, DBSCAN, HDBSCAN, and Spectral Clustering. This cluster-based training strategy enables the model to adapt to distinct demand patterns associated with different product segments. In addition to cluster-level modeling, Prophet was also applied to total

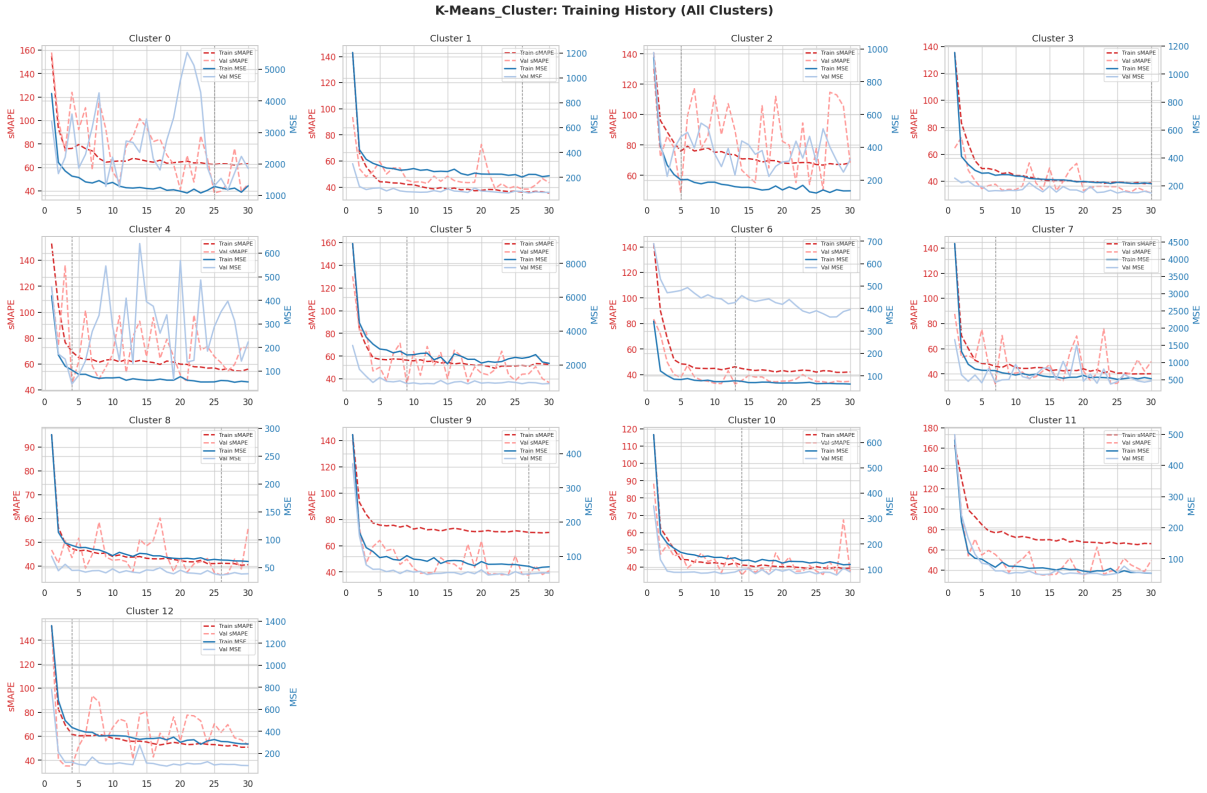


Figure 64: K-Means Clustering: RNN Training Diagram

daily sales, average product sales, and individual sales for each product. Given Prophet's strength in long-term trend modeling, the time series data was split into 60% for training, 20% for validation (used for tuning hyperparameters), and the remaining 20% for testing. Model performance is assessed using the symmetric mean absolute percentage error (sMAPE), allowing for fair comparison across time series with varying scales. Through this approach, we evaluate Prophet's ability to capture both general and segment-specific sales dynamics.

5.6 TFT

The Temporal Fusion Transformer (TFT) is a deep learning architecture designed for interpretable multi-horizon time series forecasting. It integrates recurrent encoders, attention-based decoding, gating mechanisms, and context-variable embeddings into a unified model that handles heterogeneous inputs with both temporal and static components. We use TFT to forecast daily product-level sales across a 7-day prediction horizon.
TFT is particularly well-suited to this task because:

- It captures both short- and long-term dependencies through recurrent and attention layers.
- It supports mixed covariates—including static, known future, and observed historical variables.
- It provides variable importance and interpretable attention mechanisms for enhanced transparency.

Feature engineering and representation. The predictors used by the TFT include:

- **Lagged sales features** (time-varying unknown reals): Previous day's sales, weekly lags, and rolling means over 3 and 7 days.
- **Calendar features** (time-varying known categoricals): Encoded day of week, month, quarter, weekend, and holiday indicators.

- **Macroeconomic indicators** (time-varying known reals): Consumer Confidence Index (CCI), Consumer Price Index (CPI), interest rate, and unemployment.
- **Interaction terms**: Multiplicative features between lagged sales and calendar indicators (e.g., sales on holidays vs. weekdays).
- **Static categoricals**: Product identity (`Description`) to learn product-specific embeddings.

We train a **single model across all products**, leveraging TFT's ability to generalize patterns while preserving product-specific signals via embeddings. Sales units are modeled with a Negative Binomial distribution to accommodate overdispersion in the target variable.

We train the model on the first 678 days of data and forecast over the final 60-day window. The encoder length is set to 30 days, and the decoder (forecast horizon) is fixed at 7 days.

Hyperparameter tuning was done using `Optuna` over 50 trials. The search space included learning rate, dropout, attention head size, hidden dimensions, and gradient clipping values. The final configuration we used to train the model is shown below:

Table 5: Selected hyperparameters for the TFT model after Optuna tuning.

Parameter	**Value**
Gradient clip value	0.086
Hidden size	20
Hidden continuous size	14
Attention head size	4
Dropout	0.222
Learning rate	0.0084
Loss function	Negative Binomial
Optimizer	Ranger
Encoder length	30 days
Decoder length	7 days

We the PyTorch Lightning library for training. Our training loop included early stopping based on validation loss and a model checkpointing strategy to preserve the best-performing epochs by validation loss. The final model is trained using batches of 64 samples with a learning rate scheduler and monitoring tools for loss curves and learning dynamics.

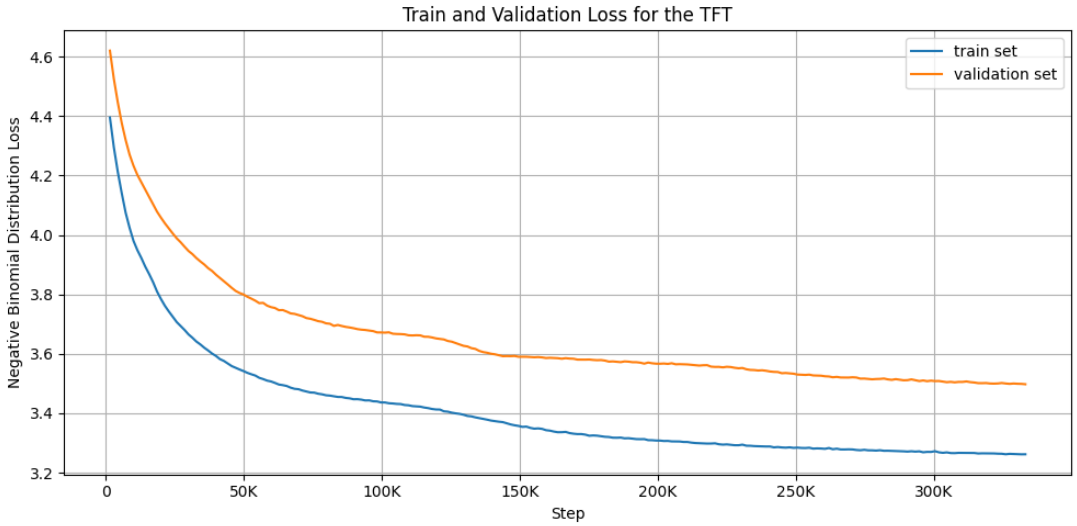


Figure 65: Training and validation loss for the Temporal Fusion Transformer (TFT) model. Loss is computed using the Negative Binomial loss over mini-batches.

Figure 65 shows the training and validation loss curves for the TFT model across all optimization steps. The validation loss steadily decreases alongside the training loss, indicating stable convergence and good generalization. The loss begins to plateau in later epochs, suggesting diminishing returns with continued training—consistent with the use of early stopping. Notably, the validation curve does not exhibit signs of overfitting, reinforcing the suitability of the tuned hyperparameters and regularization choices.

5.7 DeepAR

DeepAR was also utilized to predict daily product sales in this store. DeepAR is a probabilistic forecasting model developed by Amazon for univariate time-series forecasting. DeepAR is implemented in Amazon SageMaker and uses recurrent neural networks (RNNs). DeepAR takes in multiple time series as input each identified with a unique item_id. In the training phase, DeepAR uses the RNN to learn a representation of time-dependent patterns and trains a sliding window over each time series to predict the next time steps. In the prdiction phase you feed in past observations and covariates and the model samples future values using its learned distribution.

As we will discuss deeper in the results section, DeepAR struggled to capture and predict the time series of the provided products. Because of this, the target variable (UnitsSold) was log-transformed in the training period in order to stabilize variance, improve forecasting accuracy, and avoid negative predictions. Additionally, various methods of using explanatory variables and lag were used to help DeepAR understand the economic signals and patterns. Traditionally, DeepAR does not require clustering. However, some K-Means clustering was attempted in an attempt to improve performance.

The hyperparameters of the model are tuned to have a daily time frequency, student-t as the likelihood function, a context length of 365 (to have the whole previous year in context), a max epoch of 100, learning rate of $5e-4$, 3 as the number of layers, and 80 as the number of cells. Most of the data from December 2009 and 2011 were used in training for training. Given DeepAR's results, only 30 days were used for prediction. Model performance can be evaluated through RMSE, MAPE, and sMAPE.

6 Results and Analysis

In the following sections, we discuss the performance of each model. We use the symmetric Mean Absolute Percentage Error (sMAPE) [31] as the primary evaluation metric. Let y_t and $\hat{y}_t$ be the observed and predicted values, respectively, at time step t. Let n be the number of time steps we are forecasting. The following is the formulation of sMAPE we use:

$$\text{sMAPE} := 100\% \times \frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \tag{1}$$

If both y_t and $\hat{y}_t$ are 0, we define sMAPE to be 0 as well.

As defined in Equation 1, sMAPE will be a number between 0% and 100%, with 0% meaning a perfect forecast and 100% being a completely inaccurate one. Because sMAPE is scale-invariant, it allows us to compare the performance of the models on products with different magnitudes of units sold

We opt for sMAPE over the traditional Mean Absolute Percentage Error (MAPE) due to the nature of our data, which includes instances where product sales are zero (see the Units sold distribution section). MAPE is undefined when actual values are zero, as it involves division by the actual value, leading to infinite or undefined errors in such cases. A workaround is to add a small positive constant to the denominator of MAPE to prevent division by zero. However, this results in disproportionally high MAPE scores when the actual value is 0, even when the prediction is close to 0 (say 1). On the other hand, sMAPE would be 100% if the observed value is 0 and the forecast is anything else, and vice-versa.

6.1 GLM

As discussed in Section 5.1, we trained a separate Negative Binomial GLM for each product using the first 678 days of available data. This left the final 60 days of the dataset for testing.

To evaluate model performance, we followed a rolling prediction approach across the test set. The goal is to understand the model's performance on different forecasting horizons. For each product and each day in the test period, we performed the following steps:

1. Load the latest known sales history and all relevant features (e.g., lags, calendar indicators, macroeconomic variables).
2. Use the trained GLM to forecast up to 7 future days of sales, recursively feeding predictions back into the input features for subsequent days.
3. After producing the forecast, we reveal the next day's true sales and incorporate it into the observed history.

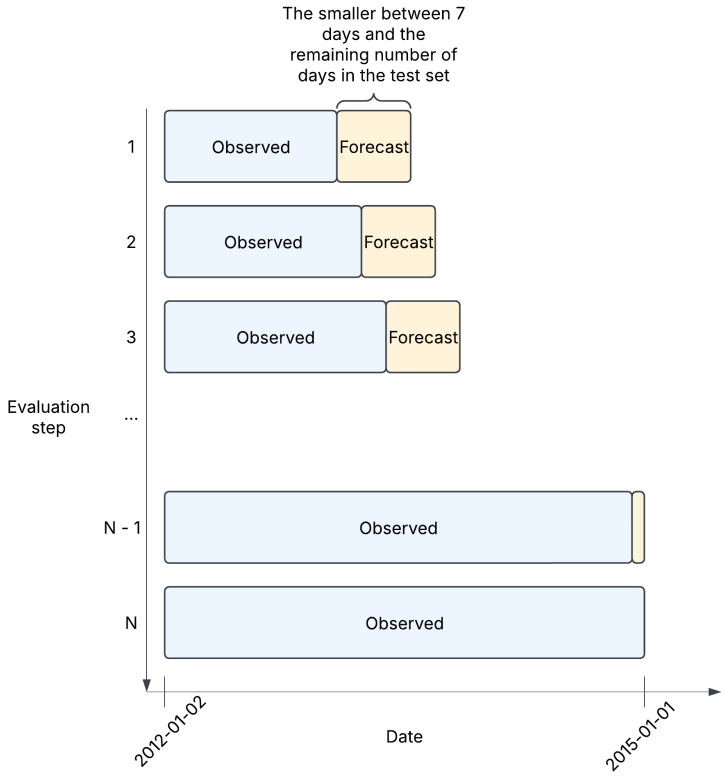


Figure 66: Rolling evaluation procedure for each product. After each 7-day forecast, the next true observation is revealed and used to update the autoregressive inputs.

Using this evaluation framework, we generate seven different forecasts for each date in the test set. The first forecast for a given date is made when that date is 7 days away from the latest observed point, and the last forecast is made when it is only 1 day away. This structure allows us to assess how forecast accuracy evolves with temporal distance from the training data.

For each product and forecast horizon (i.e., offset of 1 to 7 days), we compute a single sMAPE. This yields a total of 7 sMAPE values per product, which we use to analyze performance across products and forecast horizons.

6.1.1 Performance by forecast horizon

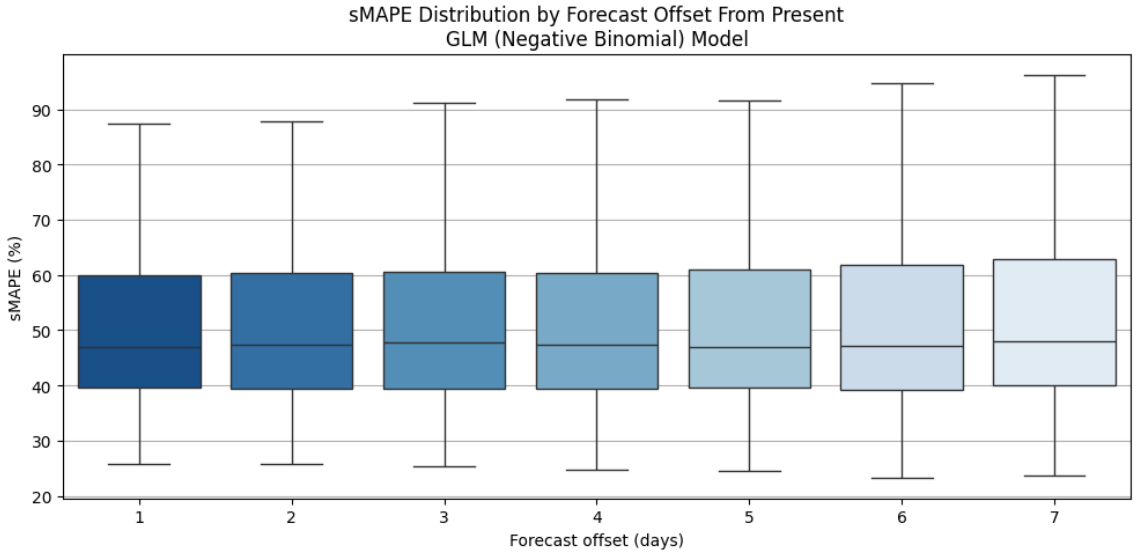


Figure 67: sMAPE distribution by forecast offset. Predictions made with GLM (Negative Binomial) model. Boxplots show increasing error and dispersion with longer horizons.

Table 6: sMAPE statistics by forecast offset for the GLM model. Total number of products: 142.

Forecast offset (days)	Mean	P10	P25	P50	P75	P90
1	51.21%	34.14%	39.52%	47.01%	59.98%	74.95%
2	51.43%	34.10%	39.49%	47.34%	60.37%	75.50%
3	51.60%	34.12%	39.36%	47.73%	60.61%	75.35%
4	51.70%	34.38%	39.38%	47.44%	60.44%	76.35%
5	51.76%	34.23%	39.58%	47.04%	60.91%	76.40%
6	51.74%	34.17%	39.29%	47.12%	61.87%	75.92%
7	52.60%	34.79%	40.01%	47.98%	62.81%	76.92%

Figure 67 and Table 6 summarize how forecast accuracy changes as the prediction horizon increases. As expected, performance generally degrades the farther into the future the model attempts to forecast. The average sMAPE increases modestly from 51.21% at a 1-day offset to 52.60% at a 7-day offset. This trend is consistent across all quantiles. For example, the 75th percentile sMAPE increases from 59.98% to 62.81%, and the 90th percentile rises from 74.95% to 76.92%. The median performance remains stable around 47–48% throughout the horizon.

The relatively small spread between horizons suggests that while performance does worsen over time, the degradation is gradual. This can be a reflection of the fact that the model assigned low weights to autoregressive features, such that even if they are off, the prediction is not affected as much.

6.1.2 Performance by day of week

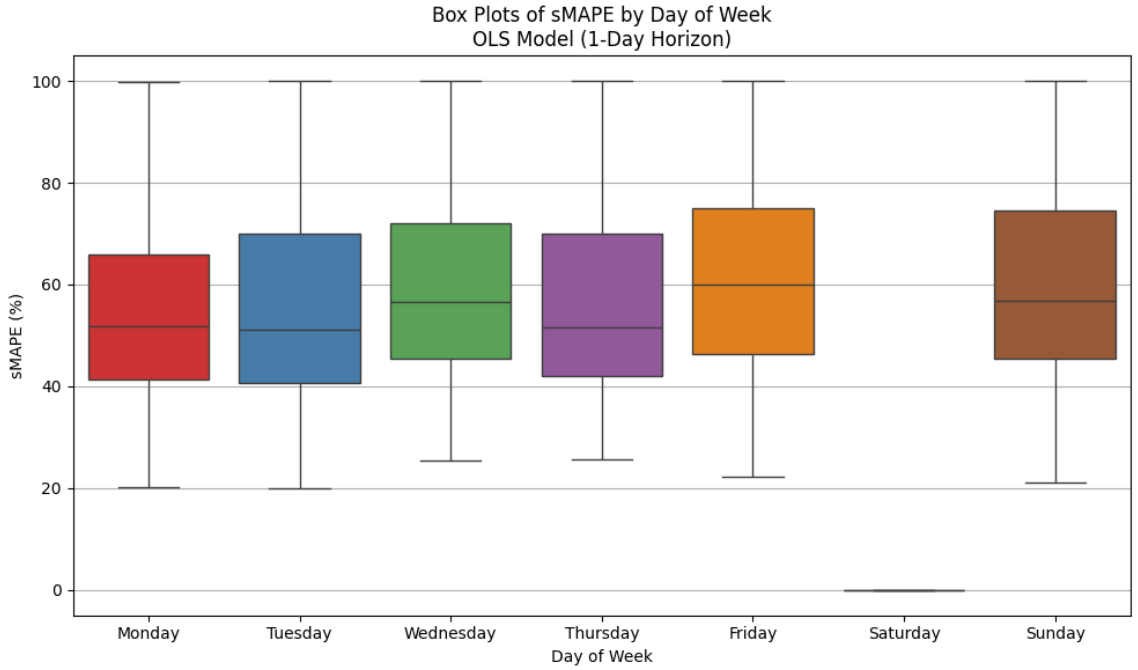


Figure 68: Box plots of sMAPE by day of week for 1-day-ahead forecasts using the GLM (Negative Binomial) model. Variability and accuracy differ by weekday, with notably low error on Saturdays.

Table 7: sMAPE key statistics by day of the week. Total number of products: 142.

Day of Week	Mean	P25	P50	P75	Max
Monday	54.95%	41.40%	51.74%	65.90%	99.76%
Tuesday	54.43%	40.62%	51.10%	70.11%	100.00%
Wednesday	59.26%	45.46%	56.54%	72.05%	100.00%
Thursday	56.06%	42.11%	51.53%	70.04%	100.00%
Friday	60.14%	46.43%	59.93%	75.04%	100.00%
Saturday	9.33%	0.00%	0.00%	0.00%	100.00%
Sunday	60.73%	45.52%	56.87%	74.66%	100.00%

Figure 68 and Table 7 show how forecast accuracy varies across days of the week for the 1-day-ahead GLM forecasts. While the model performs similarly across weekdays, we observe notable variations that may reflect differences in consumer behavior, stock availability, or feature informativeness.

The best performance occurs on Monday and Tuesday, with average sMAPE values of 54.95% and 54.43%, respectively. In contrast, Friday and Sunday show the highest average sMAPE scores (60.14% and 60.73%), indicating that sales on or before weekends are more difficult to predict. These days also show the widest interquartile ranges, suggesting larger variability across products.

Saturday exhibits a unique behavior: its mean sMAPE is only 9.33%, with nearly all products showing 0% error. This is likely due to a data artifact, such as many products having zero or no sales recorded on Saturdays in the test set, resulting in trivially accurate forecasts (e.g., predicting zero units sold).

6.1.3 Case studies

While aggregate statistics like sMAPE provide an overall view of model performance, they can mask important nuances. To better understand how the model behaves across different product types, we

examine a subset of individual prediction.

Figure 69 presents observed and predicted daily sales for four randomly selected products with varying sMAPE scores. In products like *Lunch bag woodland* and *Fancy font birthday card*, the GLM captures the general level and fluctuations of sales reasonably well, despite occasional underestimation of high spikes. For *Cream sweetheart mini chest*, the model maintains the correct overall structure, but underpredicts most of the peaks, which contributes to its moderate sMAPE of 36.22%.

A failure case is shown in *Retro spot tea set ceramic 11 pc*, where the model fails to adapt to a sudden regime shift in November. Here, predicted sales spike to a constant high value, overshooting the actual demand. This behavior likely stems from autoregressive inputs compounding a single high prediction and highlights a known weakness of recursive forecasting in GLMs when faced with structural breaks or outliers.

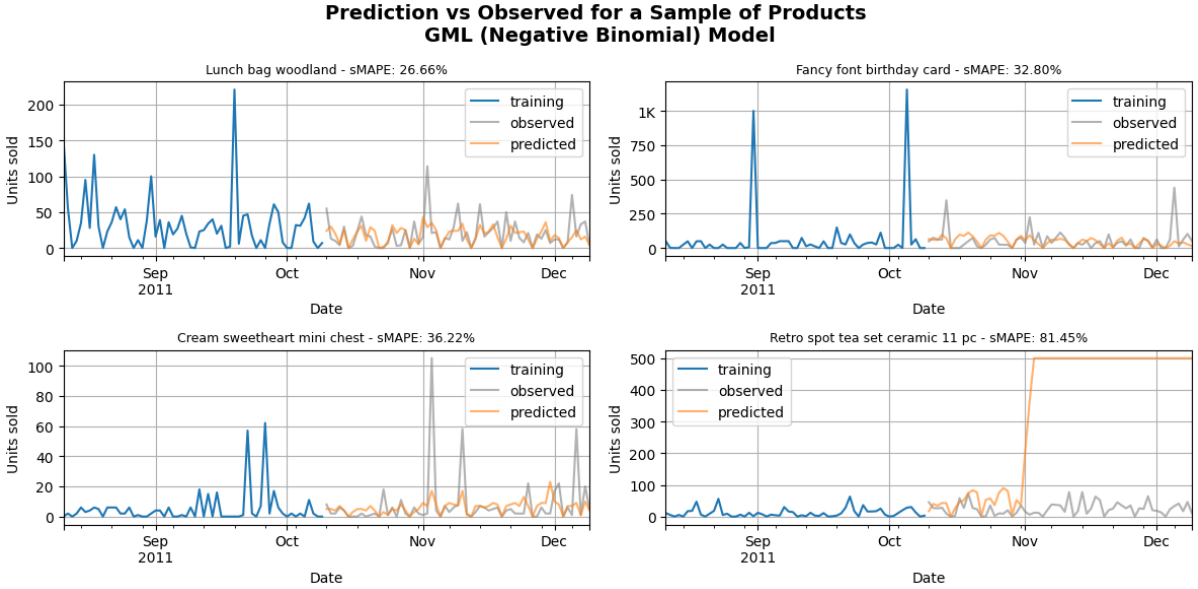


Figure 69: Predicted vs. observed sales for a sample of products using the GLM (Negative Binomial) model. Each panel shows the last 60 days of training and the 60-day test window. Products are selected to illustrate a range of model performance.

To further assess model behavior, we compare the distributions of predicted and observed sales in log-transformed space. Figure 70 shows histograms and kernel density estimates (KDE) of log1p-transformed unit sales for a sample of four products. This visualization provides insight into the distributional alignment between model forecasts and actual demand.

For products like *Lunch bag woodland* and *Fancy font birthday card*, the predicted distributions are reasonably aligned with the observed ones. The predicted values match both the general shape and central tendency of the true sales distribution, despite some underrepresentation of extreme values.

In contrast, *Cream sweetheart mini chest* shows a modest mismatch: the predicted distribution is somewhat smoothed and shifted right compared to the observed data, likely due to the model's tendency to underestimate high-frequency, low-volume patterns.

A clear failure is observed in *Retro spot tea set ceramic 11 pc*. The predicted values are tightly concentrated at the upper bound of the test set, far from the true observed values, which remain centered around log-unit values of 2–3. This confirms the model's earlier failure shown in Figure 69.

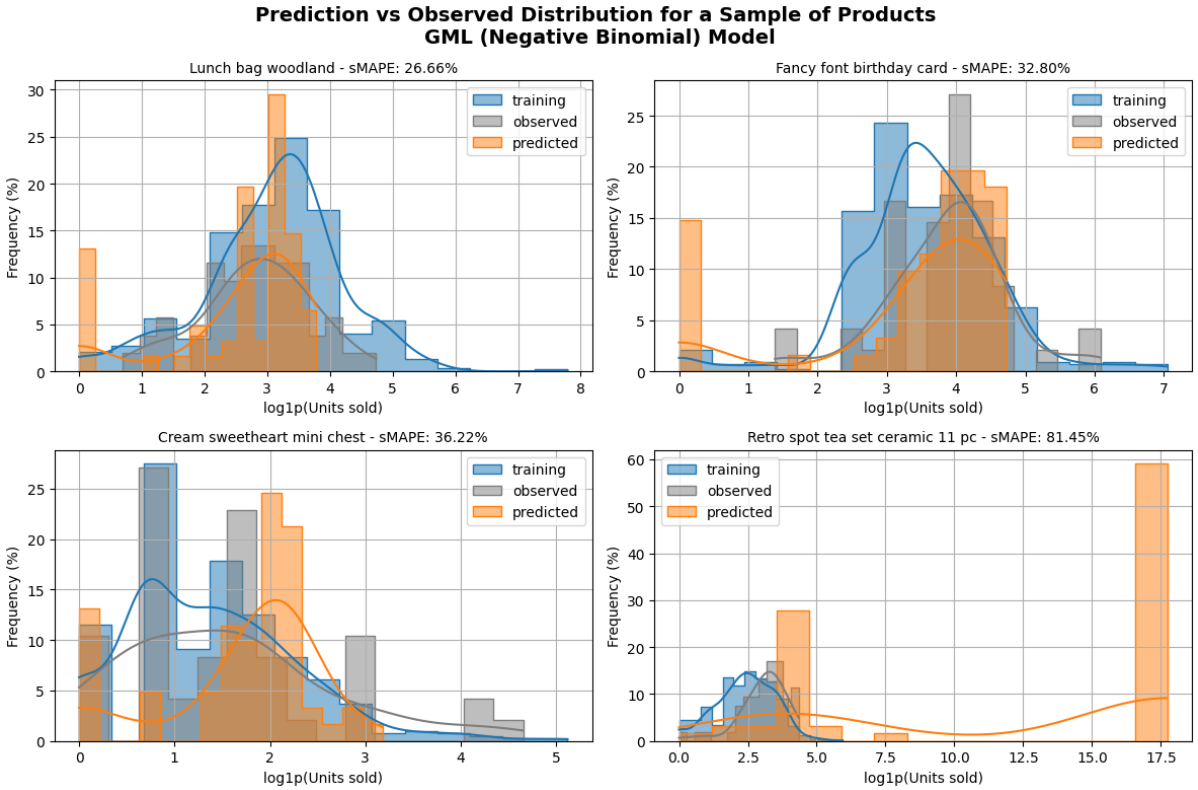


Figure 70: Distribution of log1p-transformed predicted vs. observed daily sales for a sample of products. Distributions are estimated over the 60-day test period, with training distributions included for context.

To assess the bias and heteroskedasticity of the model's predictions, we examine residuals (Predicted – Actual) as a function of predicted values. Figure 71 shows the residual scatter plots for the sample of four products, along with LOESS-smoothed curves to highlight systematic trends.

For *Lunch bag woodland* and *Fancy font birthday card*, we observe an upward trend in residuals at higher predicted values, indicating a tendency to overpredict when demand is expected to be high. At lower predicted volumes, residuals are more symmetric and centered around zero, suggesting reasonable calibration in those regimes.

In *Cream sweetheart mini chest*, most predictions are tightly clustered at low values, and the model appears moderately biased toward overprediction. The residual spread is fairly narrow, which aligns with its intermediate sMAPE score.

A striking anomaly is again seen in *Retro spot tea set ceramic 11 pc*, where residuals grow explosively with predicted volume. The LOESS line reveals a steep, unbounded upward slope, reflecting extreme overprediction.

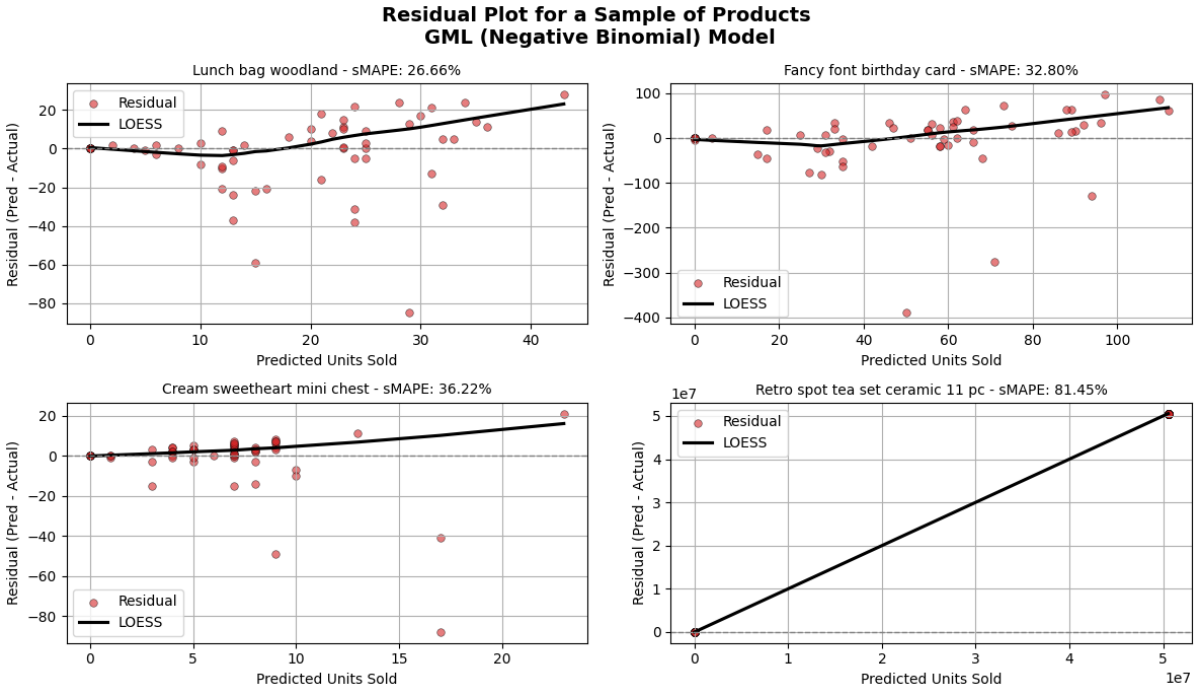


Figure 71: Residuals (Predicted – Actual) vs. Predicted Units Sold for a sample of products. A LOESS trend line is overlaid to highlight systematic bias across the prediction range.

6.2 Decision Tree

As discussed in 5.2, we trained a separate Decision Tree Regressor model for each product using the first 678 days of available data. This left the final 60 days of the dataset for testing. The evaluation strategy is the same as that used for the GLM and described in Section 6.1.

6.2.1 Performance by forecast horizon

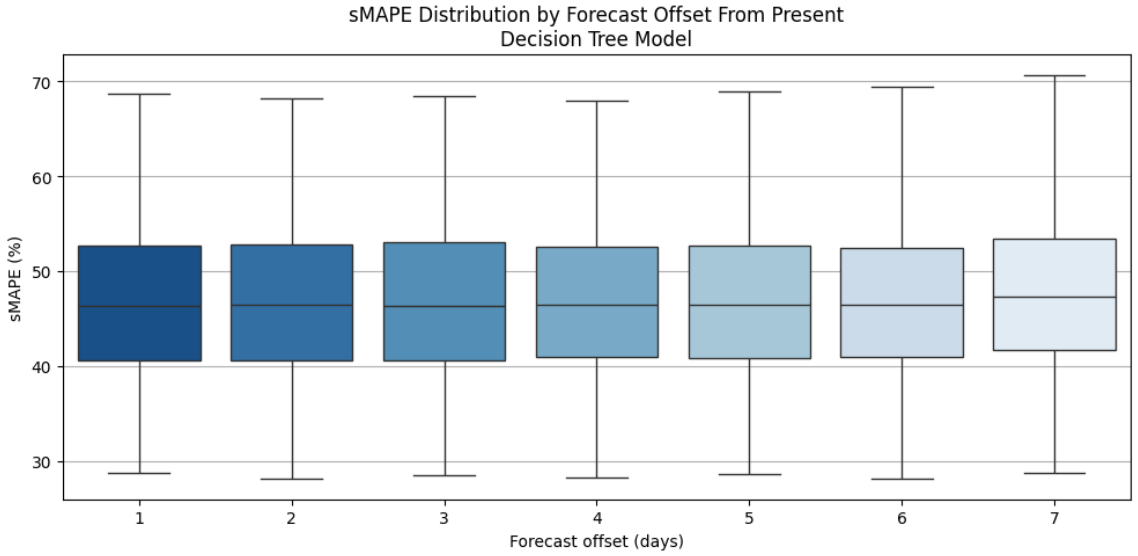


Figure 72: sMAPE distribution by forecast offset. Predictions made with GLM (Negative Binomial) model. Boxplots show increasing error and dispersion with longer horizons.

Table 8: sMAPE statistics by forecast offset for the Decision Tree model. Total number of products: 142.

Forecast offset (days)	Mean	P10	P25	P50	P75	P90
1	47.58%	35.92%	40.61%	46.34%	52.71%	59.03%
2	47.53%	35.97%	40.61%	46.51%	52.78%	59.21%
3	47.43%	35.74%	40.54%	46.28%	53.07%	59.10%
4	47.37%	35.32%	40.97%	46.49%	52.51%	58.64%
5	47.40%	35.45%	40.84%	46.50%	52.63%	58.21%
6	47.36%	35.50%	40.90%	46.43%	52.46%	57.63%
7	48.22%	36.14%	41.65%	47.28%	53.41%	58.68%

Figure 72 and Table 8 summarize how forecast accuracy changes as the prediction horizon increases for the Decision Tree model. The Decision Tree exhibits remarkably stable performance across all horizons. The average sMAPE remains nearly constant, increasing only slightly from 47.58% at a 1-day offset to 48.22% at a 7-day offset.

The interquartile range remains tight and centered around a median of 46–47% throughout the horizon, as seen in the boxplot. This indicates a high degree of consistency in forecast quality across different products and forecast lengths. The gradual degradation likely reflects the tree model's robustness to input uncertainty, as it does not rely heavily on autoregressive chains of predictions. Instead, it prioritizes calendar, price, and macroeconomic variables, which remain reliable across time steps.

6.2.2 Performance by day of week

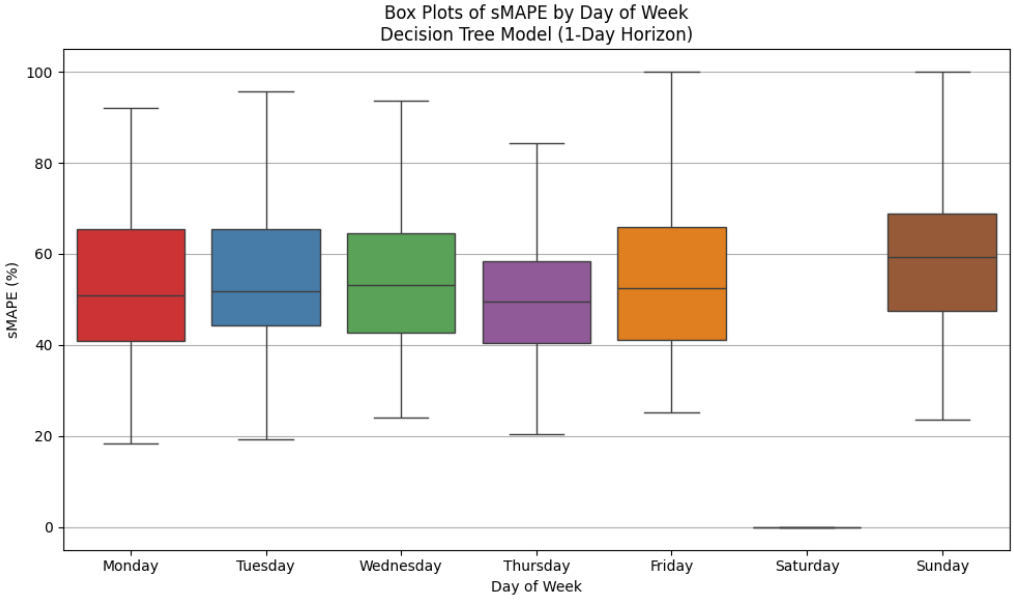


Figure 73: Box plots of sMAPE by day of the week for the Decision Tree model (1-day forecast horizon). Each box summarizes the distribution of sMAPE values across 142 products.

Table 9: sMAPE statistics by day of the week for the Decision Tree model. Total number of products: 142.

Day of Week	Mean	P25	P50	P75	Min	Max
Monday	53.65%	40.84%	50.80%	65.35%	18.38%	92.17%
Tuesday	54.21%	44.30%	51.82%	65.50%	19.35%	95.65%
Wednesday	54.07%	42.59%	53.09%	64.51%	24.13%	93.65%
Thursday	50.60%	40.49%	49.50%	58.45%	20.46%	97.10%
Friday	54.84%	41.11%	52.45%	65.83%	25.10%	100.00%
Saturday	3.26%	0.00%	0.00%	0.00%	0.00%	62.50%
Sunday	58.77%	47.59%	59.35%	68.77%	0.00%	100.00%

Figure 73 and Table 9 show how forecast accuracy varies across days of the week for the 1-day-ahead Decision Tree forecasts. Similar to the GLM, weekday performance is relatively stable, but we observe notable differences on weekends.

Thursdays have the second best accuracy, with a mean sMAPE of 50.60% and relatively tight interquartile spread. Monday, Tuesday, and Wednesday show slightly higher average errors around 54%, but remain consistent across the distribution. Friday and Sunday exhibit the highest mean sMAPE values (54.84% and 58.77%, respectively), with wider interquartile ranges and some extreme outliers—indicating that sales on or near weekends are more challenging to predict, possibly due to erratic demand or promotional effects.

Saturday has the best performance. This is a reflection of the fact that the model successfully learned that the retailer is often closed that day of the week.

6.2.3 Case studies

To better understand how the Decision Tree model behaves across different product types, we examine a subset of individual predictions.

Figure 74 presents observed and predicted daily sales for four randomly selected products with varying sMAPE scores. For *Charlotte bag suki design* and *Love building block word*, the model tracks the overall sales trends reasonably well, although the predictions often appear conservative—missing the peaks and valleys of the observed data. For *Box of 24 cocktail parasols*, the model captures the general range but struggles with the sharp transitions in demand. In contrast, *Plasters in tin skulls* shows a model failure: the predicted series often underestimates the actual sales, leading to the highest sMAPE of the sample (61.29%).

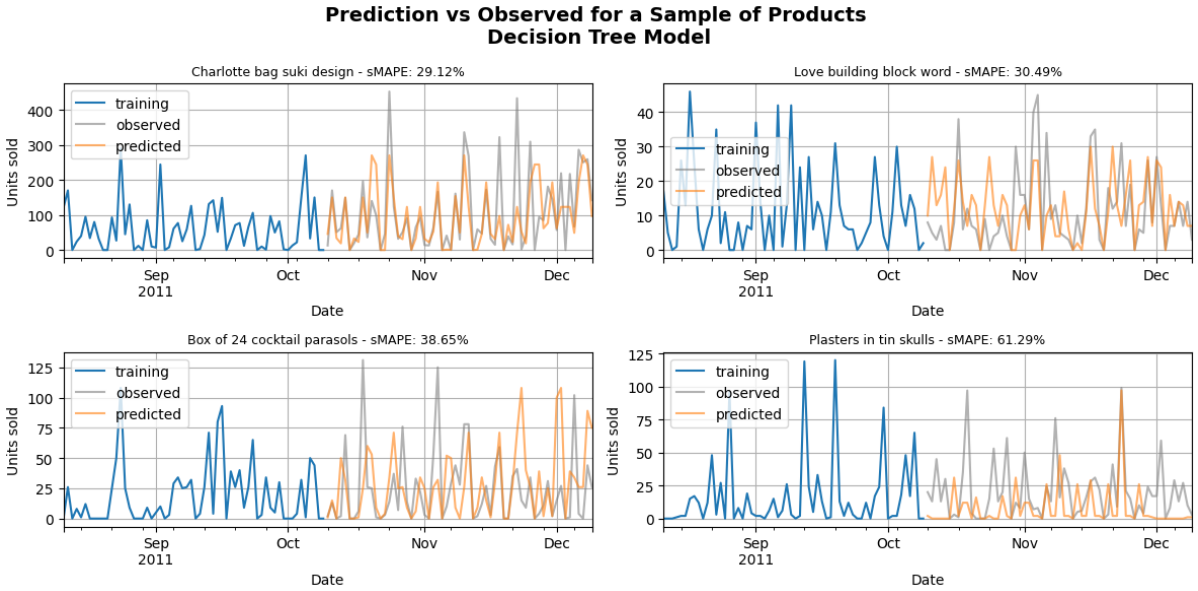


Figure 74: Predicted vs. observed sales for a sample of products using the Decision Tree model. Each panel shows the last 60 days of training and the 60-day test window. Products are selected to illustrate a range of model performance.

To further assess model behavior, we compare the distributions of predicted and observed sales in log-transformed space. Figure 75 shows histograms and kernel density estimates (KDEs) of log1p-transformed unit sales for the same sample of products. These distributions highlight the degree to which the Decision Tree captures the shape and scale of the true sales distribution.

In the case of *Charlotte bag suki design*, the predicted and observed distributions are well aligned, indicating good calibration. For *Love building block word*, predictions match the observed range but show a narrower spread, hinting at over-smoothing. *Box of 24 cocktail parasols* exhibits a mismatch in modality—predicted values underrepresent some peaks, contributing to forecast error. Finally, for *Plasters in tin skulls*, the predicted distribution is tightly centered near zero, missing the observed variation entirely.

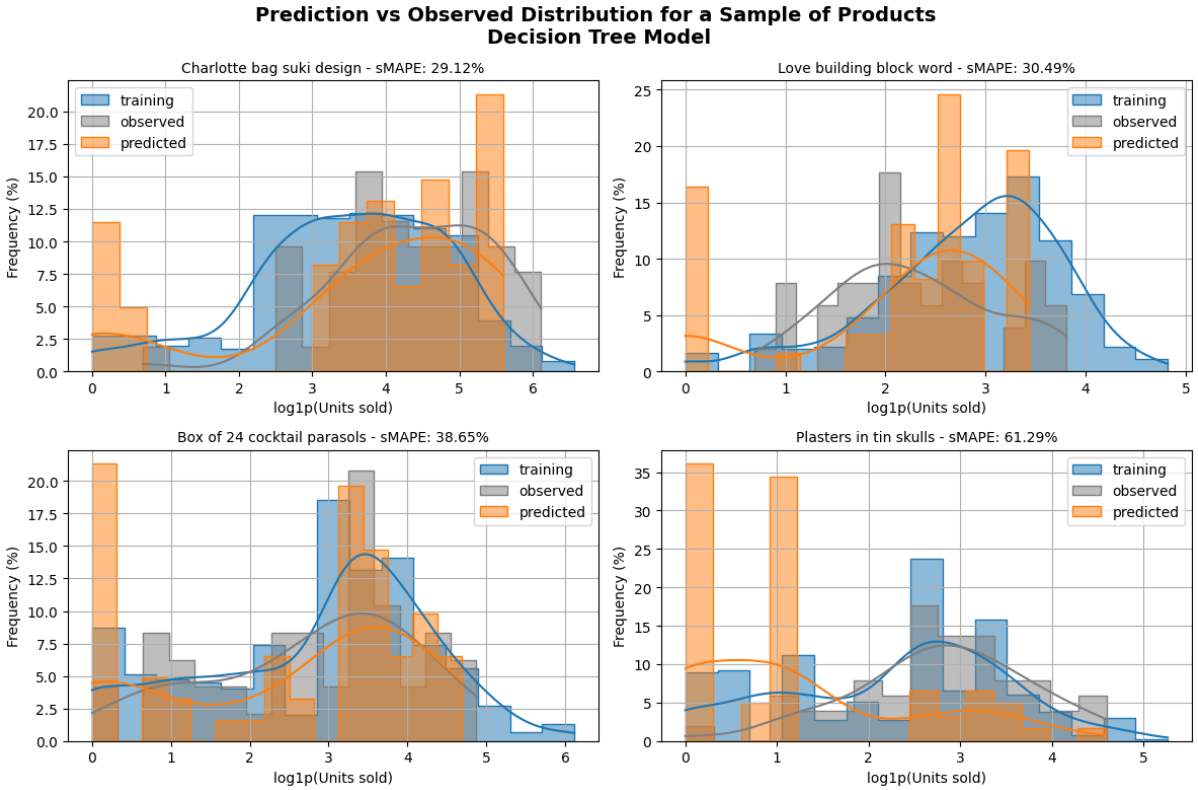


Figure 75: Distribution of log1p-transformed predicted vs. observed daily sales for a sample of products. Distributions are estimated over the 60-day test period, with training distributions included for context.

To assess the bias and heteroskedasticity of the model's predictions, we examine residuals (Predicted – Actual) as a function of predicted values. Figure 76 shows the residual scatter plots with LOESS-smoothed curves overlaid.

Charlotte bag suki design and *Love building block word* show relatively well-centered residuals, although the latter exhibits slight overprediction at mid-range volumes. In *White hanging heart tlight holder*, the residuals increase sharply with predicted volume, suggesting that the model systematically overestimates large values. For *Recycling bag retrospot*, the residual trend is mostly linear and positively biased, indicating a consistent overestimation across the forecast range.

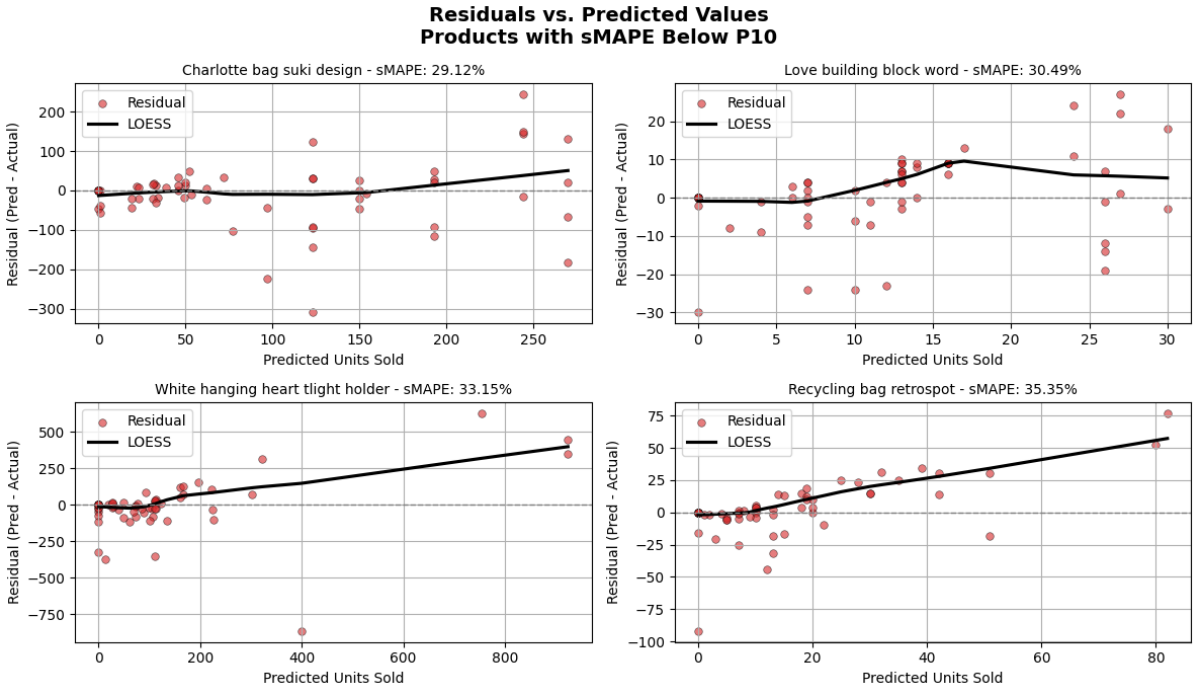


Figure 76: Residuals (Predicted – Actual) vs. Predicted Units Sold for a sample of products. A LOESS trend line is overlaid to highlight systematic bias across the prediction range.

6.3 XGBoost

As described in Section 5.3, we trained a separate XGBoost model for each product using the first 678 days of available data, leaving the final 60 days for evaluation. We selected hyperparameters through product-level tuning with Optuna, and adopted a shared configuration based on the most frequently selected values. The model was trained with squared error loss and evaluated using the same autoregressive forecasting strategy and test protocol as the GLM (Section 6.1).

6.3.1 Performance by forecast horizon

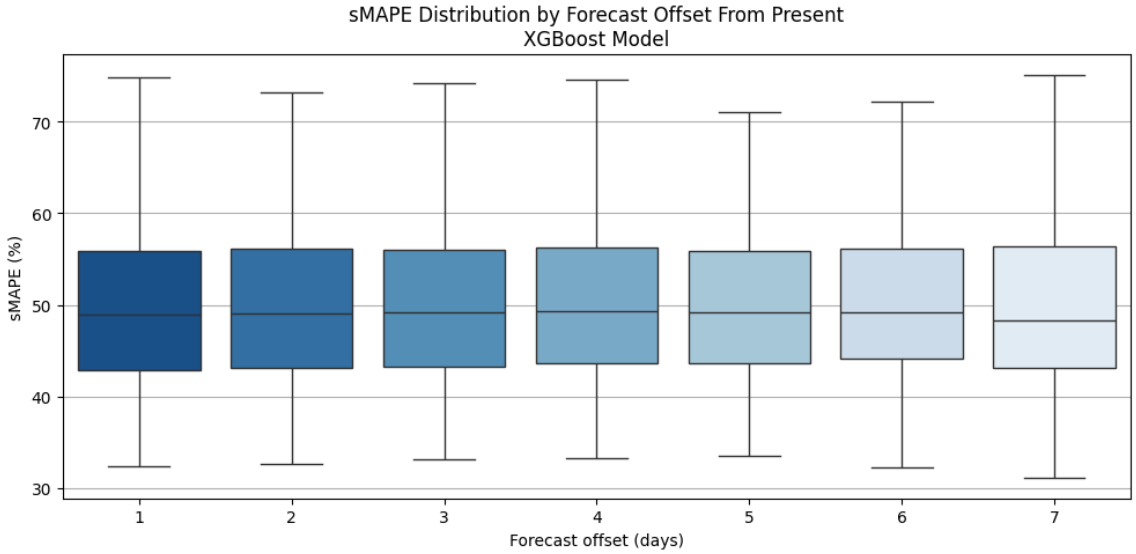


Figure 77: sMAPE distribution by forecast offset. Predictions made with the XGBoost model. Boxplots show consistent performance with moderate dispersion across horizons.

Table 10: sMAPE statistics by forecast offset for the XGBoost model. Total number of products: 142.

Forecast offset (days)	Mean	P10	P25	P50	P75	P90
1	51.07%	39.10%	42.83%	48.91%	55.92%	65.56%
2	51.39%	39.18%	43.11%	49.07%	56.08%	66.64%
3	51.57%	39.71%	43.29%	49.20%	56.01%	67.61%
4	51.66%	39.88%	43.56%	49.34%	56.25%	67.31%
5	51.85%	40.09%	43.62%	49.14%	55.88%	67.83%
6	51.96%	39.80%	44.10%	49.20%	56.17%	67.28%
7	51.36%	39.18%	43.09%	48.27%	56.41%	66.69%

Figure 77 and Table 10 summarize how forecast accuracy varies across the 7-day prediction horizon for the XGBoost model. Like the Decision Tree, XGBoost exhibits stable performance across all forecast lengths. The average sMAPE remains around 51–52%, with minimal increase as the forecast offset grows. The 1-day horizon yields a mean sMAPE of 51.07%, which rises only slightly to 51.96% at the 6-day horizon and dips to 51.36% on day 7.

Across all horizons, the interquartile range remains tight—centered around medians of 48–49%—indicating consistent accuracy across products. The model's resilience to horizon length suggests it effectively balances information from autoregressive, calendar, and price-based features. This stability reflects XGBoost's robustness to small shifts in input quality and its capacity to average over weak learners for regularized forecasts.

6.3.2 Performance by day of week

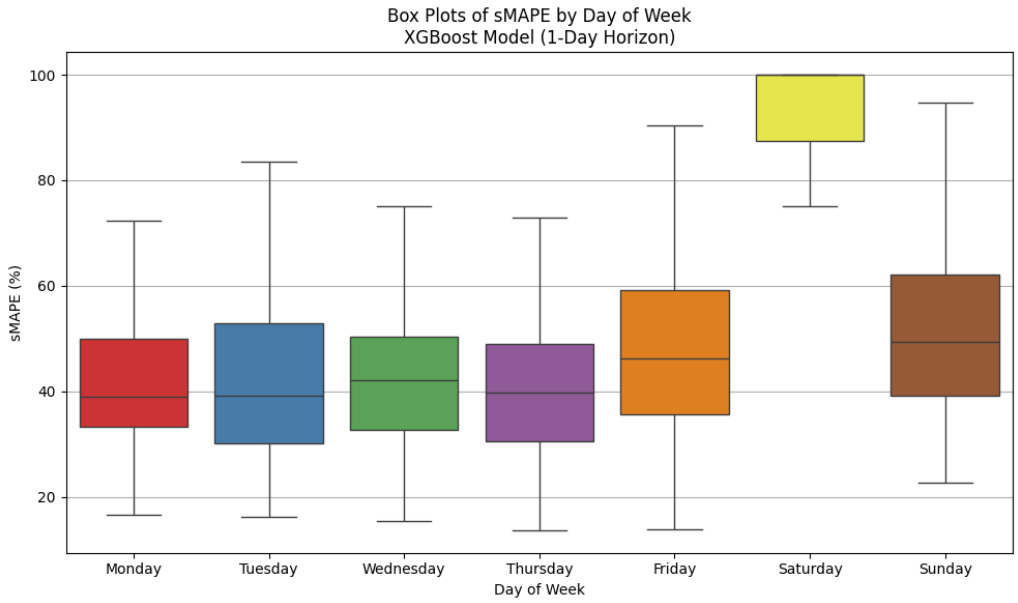


Figure 78: Box plots of sMAPE by day of the week for the XGBoost model (1-day forecast horizon). Each box summarizes the distribution of sMAPE values across 142 products.

Table 11: sMAPE statistics by day of the week for the XGBoost model. Total number of products: 142.

Day of Week	Mean	P25	P50	P75	Min	Max
Monday	42.74%	33.26%	38.86%	49.95%	16.51%	100.00%
Tuesday	42.88%	30.11%	39.19%	52.92%	16.24%	100.00%
Wednesday	44.12%	32.75%	42.04%	50.31%	15.31%	100.00%
Thursday	42.52%	30.50%	39.76%	48.90%	13.67%	100.00%
Friday	47.77%	35.68%	46.14%	59.24%	13.77%	100.00%
Saturday	89.96%	87.50%	100.00%	100.00%	12.50%	100.00%
Sunday	51.87%	39.22%	49.30%	62.07%	22.60%	100.00%

Figure 78 and Table 11 show how forecast accuracy varies across days of the week for the 1-day-ahead XGBoost forecasts. Performance across weekdays is relatively stable, with mean sMAPE values ranging between 42–44%. Thursday achieves the lowest average error (42.52%). Friday and Sunday display higher errors, with mean sMAPE values of 47.77% and 51.87%, respectively.

Saturday, once again, presents a unique pattern. It seems the model was not able to learn that Saturdays often have 0 sales, resulting in non-zero predictions and thus many sMAPE scores of 100%.

6.3.3 Case studies

To better understand how the XGBoost model behaves across different product types, we examine a subset of individual predictions.

Figure 79 presents observed and predicted daily sales for four selected products with varying sMAPE scores. For *Natural slate heart chalkboard* and *Baking set spaceboy design*, the model captures the general sales level but fails to reproduce high-variance spikes, resulting in moderately good performance (sMAPE $\approx 38\%$). *Blue harmonica in box*, a product with highly volatile sales, shows reasonable baseline tracking but underestimates extreme peaks. *Pottering in the shed metal sign* is an example failure case: the

model predicts smooth, conservative values while true sales exhibit sharp bursts, resulting in the highest sMAPE in the group (65.65%).

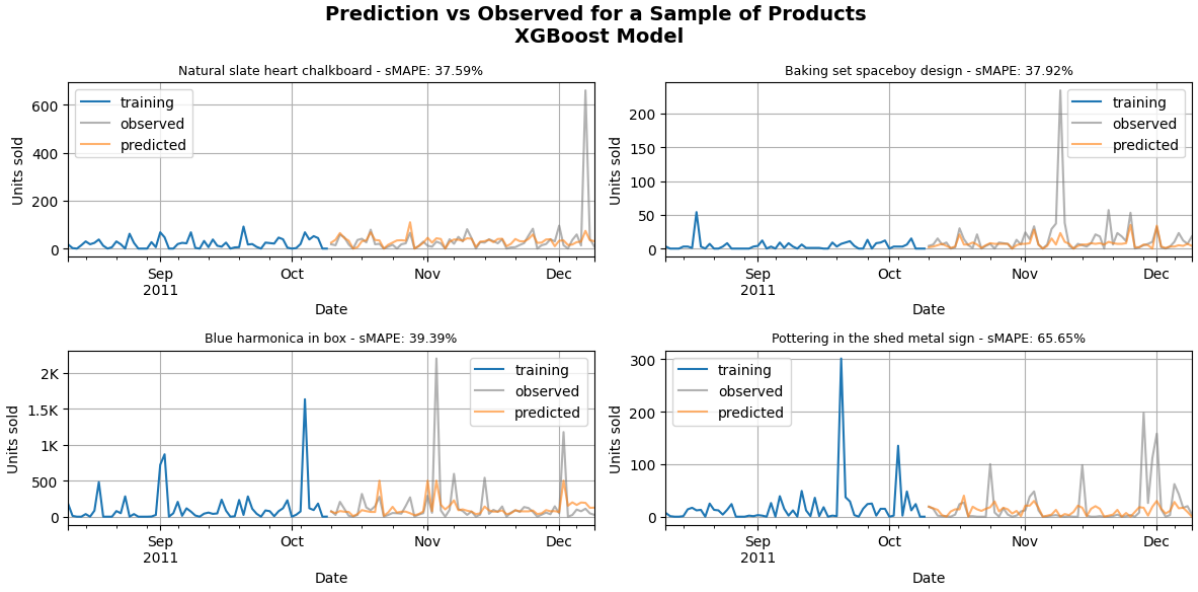


Figure 79: Predicted vs. observed sales for a sample of products using the XGBoost model. Each panel shows the last 60 days of training and the 60-day test window. Products are selected to illustrate a range of model performance.

To further assess model behavior, we examine the distributions of predicted and observed sales in log-transformed space. Figure 80 shows histograms and kernel density estimates (KDEs) of log1p-transformed daily sales. These plots offer insight into the alignment between the forecast and actual demand distributions.

In *Natural slate heart chalkboard*, the predicted and observed distributions closely match in shape and location, but the model seems to struggle with the more extreme values as evidenced by the prediction's lower variance. *Baking set spaceboy design* exhibits slight over-smoothing, with the forecast distribution narrower than the observed one. *Blue harmonica in box* again shows that the model captures the general trend but underrepresents extreme values in the right tail. For *Pottering in the shed metal sign*, the predicted distribution is far more concentrated than the observed one, which is more dispersed and multimodal.

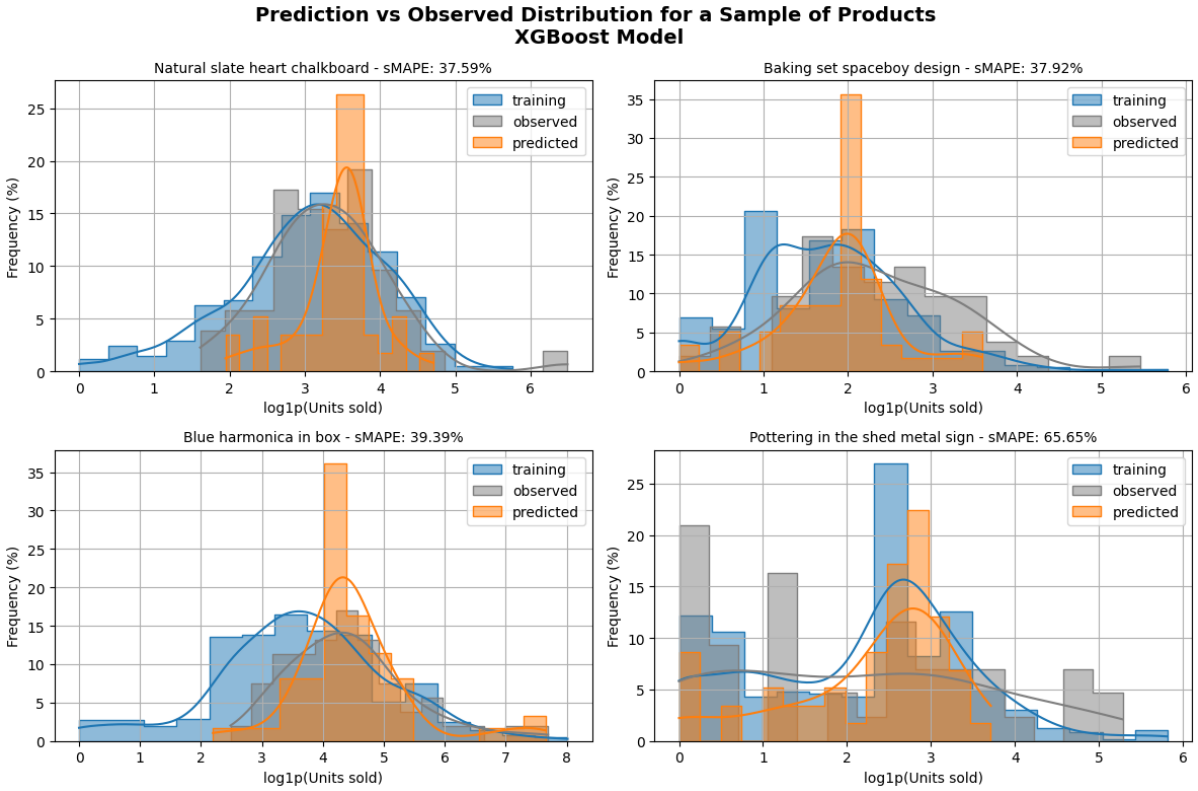


Figure 80: Distribution of log1p-transformed predicted vs. observed daily sales for a sample of products. Distributions are estimated over the 60-day test period, with training distributions included for context.

To evaluate bias and heteroskedasticity, we plot residuals (Predicted – Actual) against predicted values. Figure 81 shows scatter plots with LOESS-smoothed trend lines.

In *Natural slate heart chalkboard*, residuals remain tightly centered around zero across the prediction range, indicating reasonable calibration. *Baking set spaceboy design* exhibits slight underprediction bias at low volumes. *Blue harmonica in box* shows extreme heteroskedasticity—residuals grow rapidly with predicted volume, and many large peaks are underpredicted. Finally, *Pottering in the shed metal sign* displays systematic underestimation at moderate to high predicted values, confirming its poor forecast alignment.

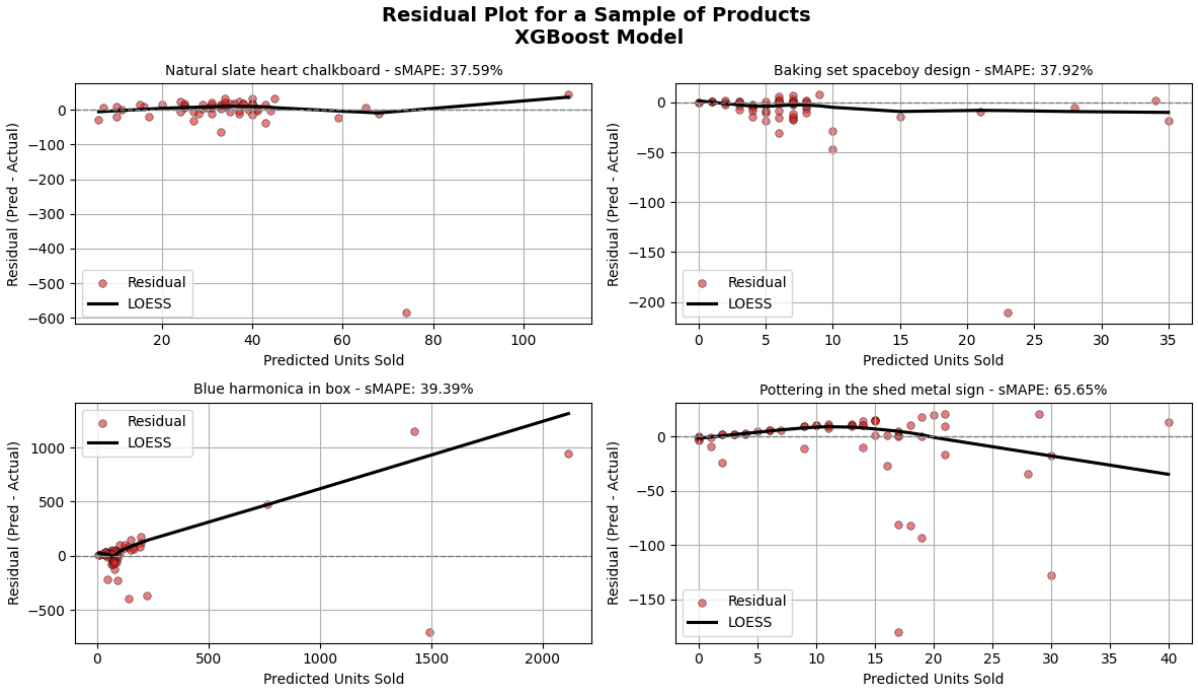


Figure 81: Residuals (Predicted – Actual) vs. Predicted Units Sold for a sample of products. A LOESS trend line is overlaid to highlight systematic bias across the prediction range.

6.4 Prophet

6.4.1 Total Product Sales

We applied Facebook Prophet to forecast total daily product sales for the non-retail store in the UK, aiming to capture long-term trends and seasonality in the aggregated demand. Before tuning, the model achieved a validation sMAPE of 33.21% and a test sMAPE of 41.23%, indicating room for improvement in forecasting accuracy. To refine the model, we performed hyperparameter tuning over key components including changepoint_prior_scale, seasonality_prior_scale, and fourier_order, which control trend flexibility, seasonal effect strength, and the complexity of yearly seasonality, respectively. The best performance was achieved with changepoint_prior_scale = 0.01, seasonality_prior_scale = 1, and fourier_order = 5. With these optimized settings, validation sMAPE improved to 31.83%, and test sMAPE decreased to 36.61%, demonstrating a meaningful improvement in predictive accuracy. These results suggest that careful tuning of Prophet's parameters can significantly enhance its ability to model complex sales patterns, even in non-retail contexts.

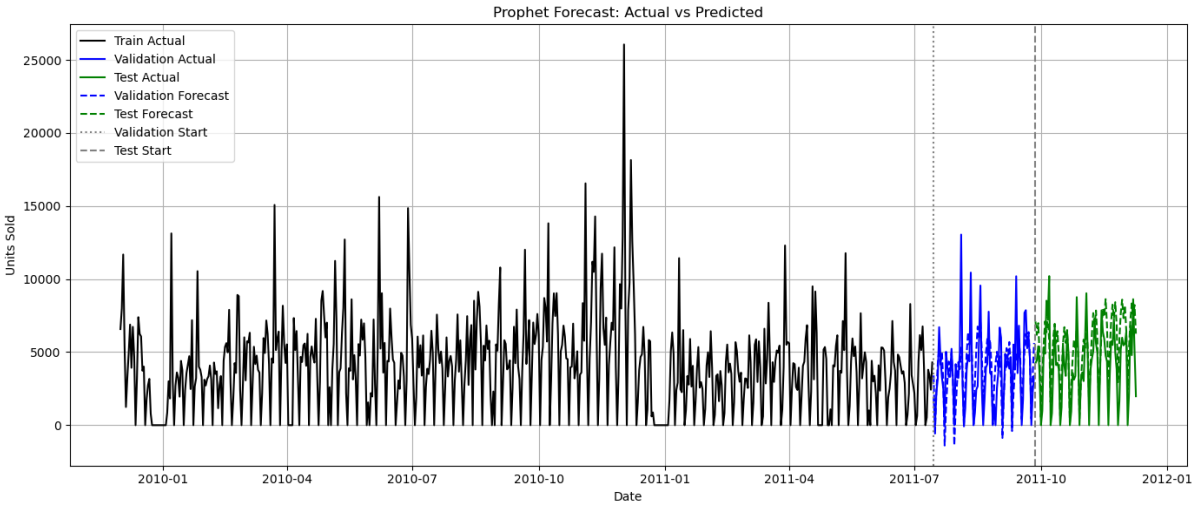


Figure 83: Total Product Sales

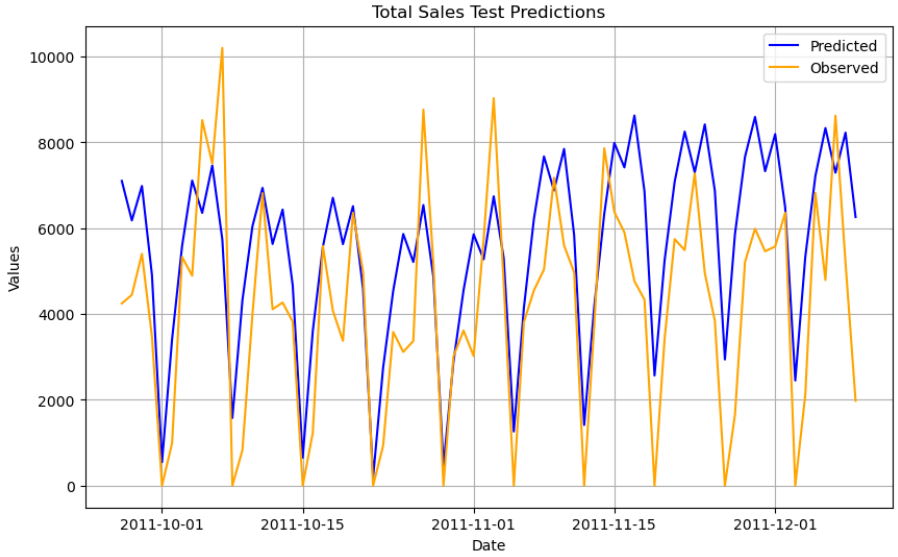


Figure 82: Total Test Product Sales

6.4.2 Total Product Sales with Explanatory Variables

To enhance the accuracy of our sales forecasting, we extended the Prophet model by incorporating a comprehensive set of explanatory variables (regressors) reflecting economic indicators, calendar events, product pricing, and consumer behavior. These included variables such as average price per unit, Google search trends for "gift ideas", Black Friday and Cyber Monday indicators, and macroeconomic metrics like CPI, CCI, interest rate, and unemployment rate, along with weekday and holiday flags. When compared to the baseline model trained only on total product sales—which achieved a validation sMAPE of 33.21% and test sMAPE of 41.23%—the initial inclusion of these regressors resulted in a higher validation sMAPE (40.75%) and test sMAPE (52.56%), suggesting the need for tuning. However, after optimizing Prophet's parameters, the model significantly improved, reaching a best validation sMAPE of 20.87%—a 37% relative improvement over the total sales baseline. Interestingly, test sMAPE increased to 79.62%, possibly due to overfitting on the validation set or increased noise in the test period. These results highlight the potential of explanatory variables to dramatically improve forecasting performance,

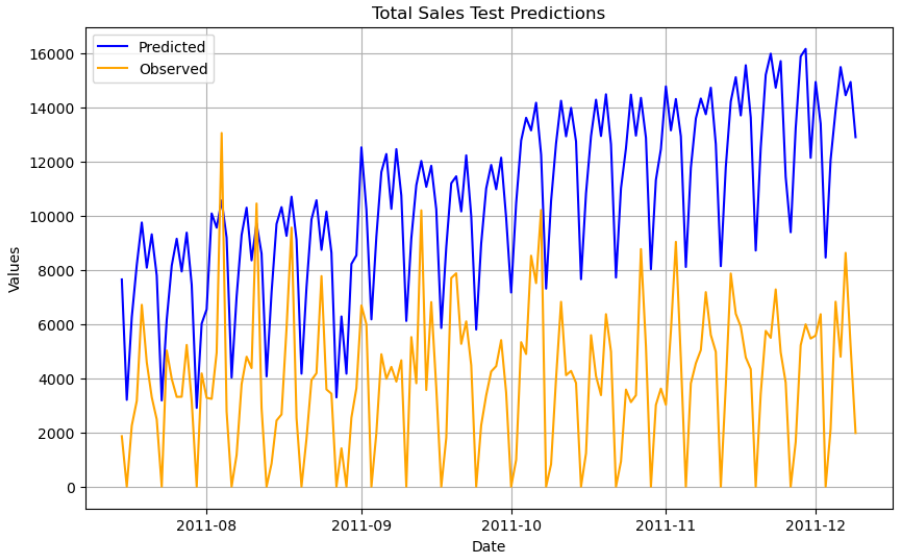


Figure 84: Total Test Product Sales with Explanatory Variable Before Tuning

while also underscoring the importance of regularization and careful validation to ensure generalizability.

6.4.3 Total Product Sales with Explanatory and Lag/Rolling Statistics

Building on our earlier models, we further improved forecasting performance by incorporating a richer set of explanatory variables, combining the previously used economic, calendar, and behavioral indicators with time series-derived features such as lags (lag_1, lag_7, lag_14) and rolling statistics (Rolling_Mean_7, Rolling_Std_7). These additions allowed the model to capture short-term temporal dependencies and local trends in sales that are not easily explained by external factors alone. Before tuning, this enhanced model achieved a validation sMAPE of 38.12% and a test sMAPE of 41.46%, showing modest improvements over the earlier version with only external regressors. After hyperparameter tuning, the model significantly improved, reducing validation sMAPE to 20.29%—a slight improvement over the 20.87% achieved previously. However, the test sMAPE increased to 72.12%, suggesting that while the model generalized well on the validation set, it may have overfit or failed to adapt to changes in the test period. Overall, this experiment demonstrated that adding lag and rolling features can enrich the model's understanding of recent trends and short-term dynamics, but also highlighted the importance of balancing feature complexity with generalization performance.

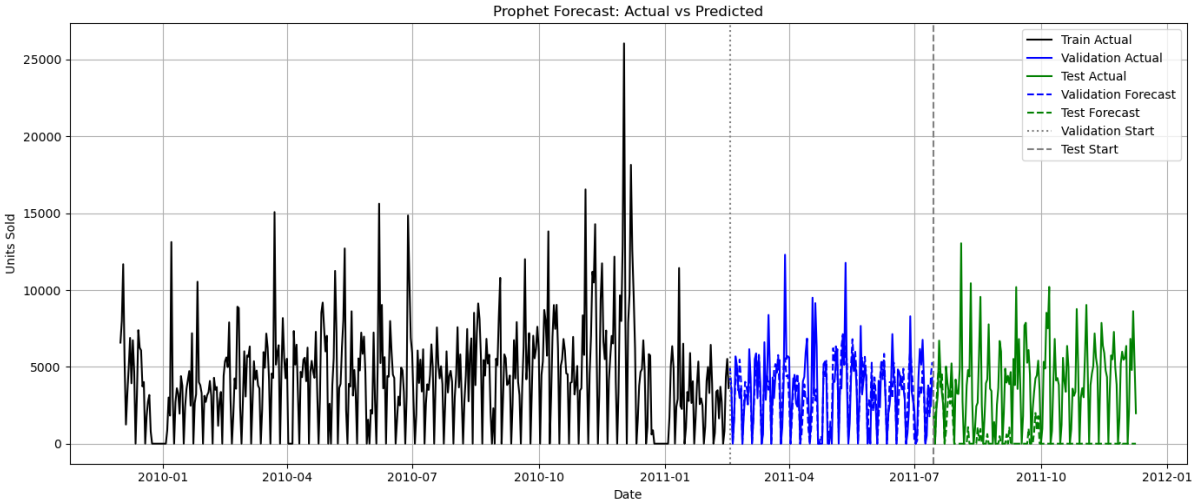


Figure 85: Total Product Sales with Explanatory Variables

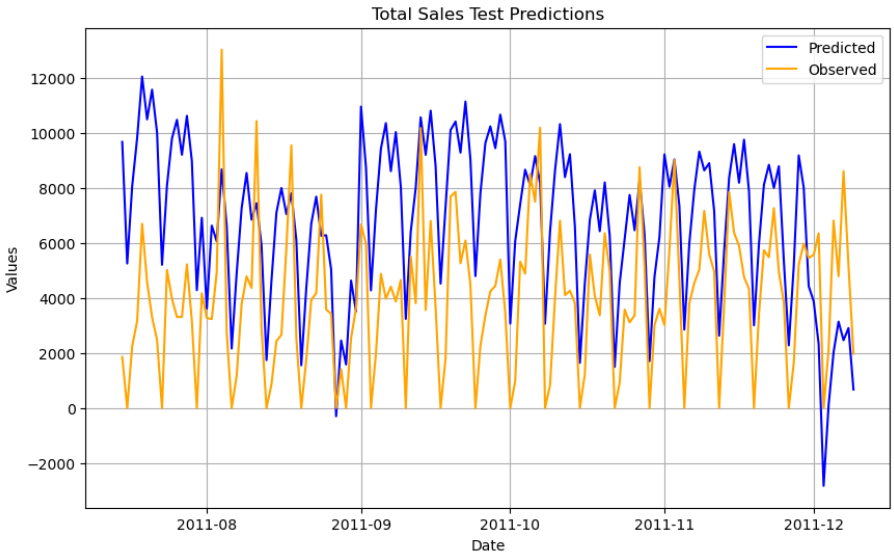


Figure 86: Total Test Product Sales with Extra Explanatory Variables Before Tuning

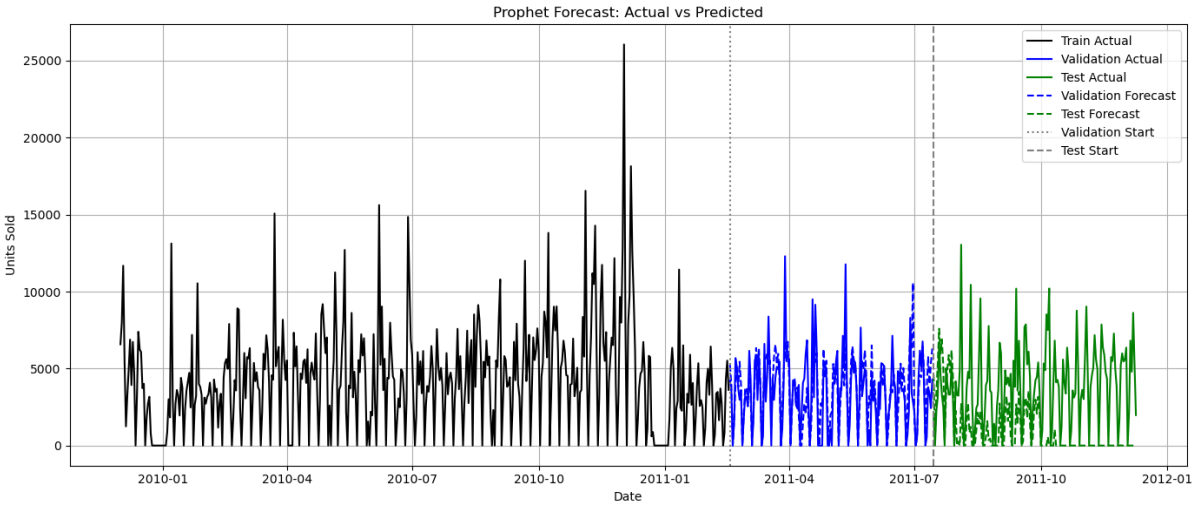


Figure 87: Total Product Sales with Extra Explanatory Variables

6.4.4 Total Product Sales with Filtered Explanatory Variables

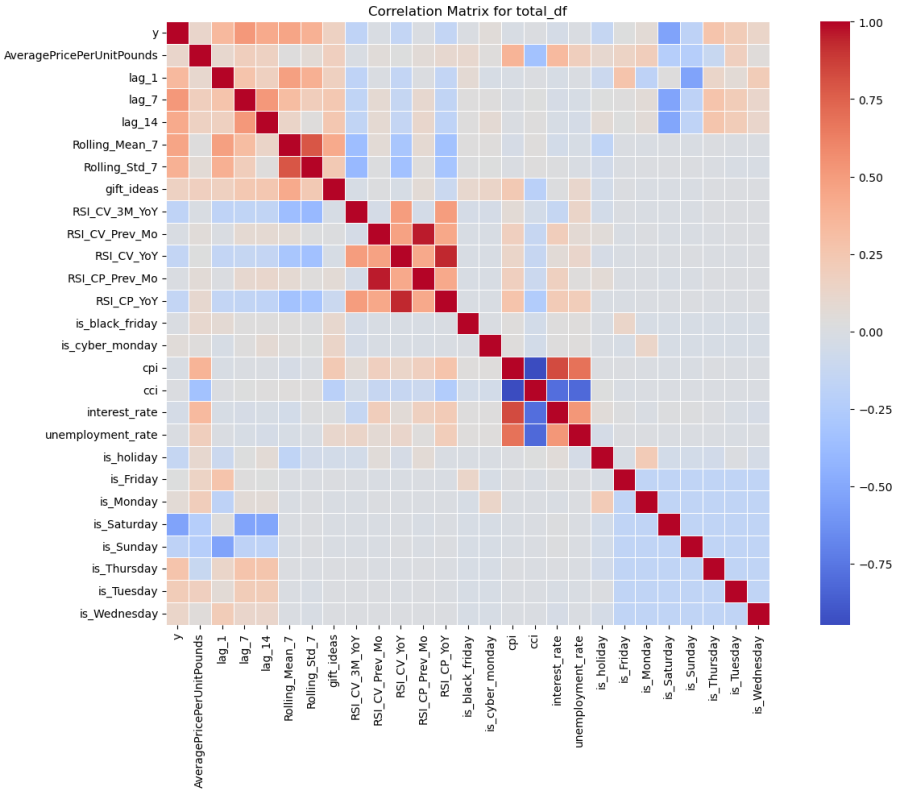


Figure 88: Correlation Matrix of Explanatory Variables and UnitsSold Column (y)

In the final iteration of forecasting total product sales, we refined our feature set by applying a correlation-based feature selection strategy. Specifically, we calculated the correlation matrix between all candidate explanatory variables and the target variable (UnitsSold) and retained only those features with a correlation above 0.2. This filtering process resulted in a compact yet effective set of regressors: lag_1, lag_7, lag_14, Rolling_Mean_7, Rolling_Std_7, is_Tuesday, and is_Thursday. By focusing on only the most

relevant predictors, the model became more interpretable and less prone to overfitting. Even before tuning, the reduced model achieved strong results with a validation sMAPE of 30.27% and test sMAPE of 29.06%, marking a significant improvement over earlier iterations. After hyperparameter tuning, the model further improved on the validation set with a sMAPE of 16.95%, although the test sMAPE rose slightly to 41.05%, suggesting potential sensitivity to unseen data. Nevertheless, this approach demonstrated that careful feature selection based on statistical relevance can enhance model performance while reducing complexity, making it a promising direction for future forecasting pipelines.

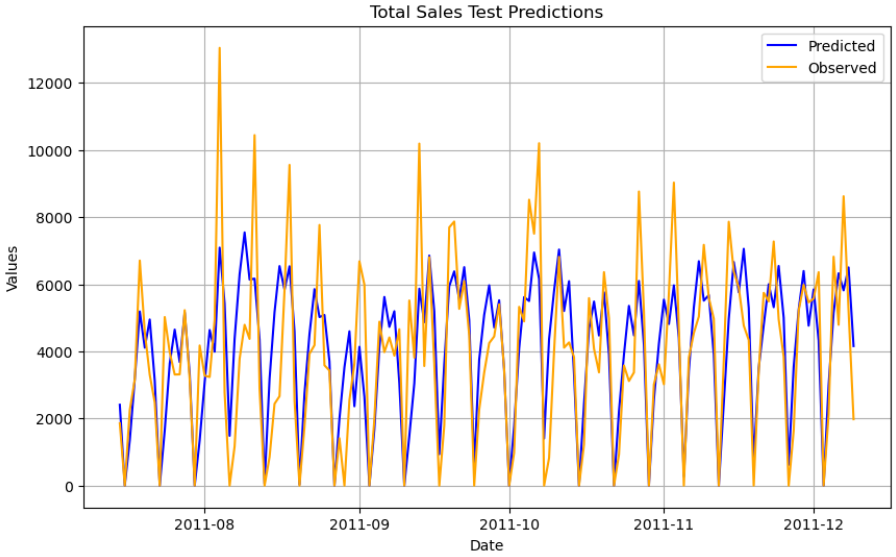


Figure 89: Total Test Product Sales with Filtered Explanatory Variables

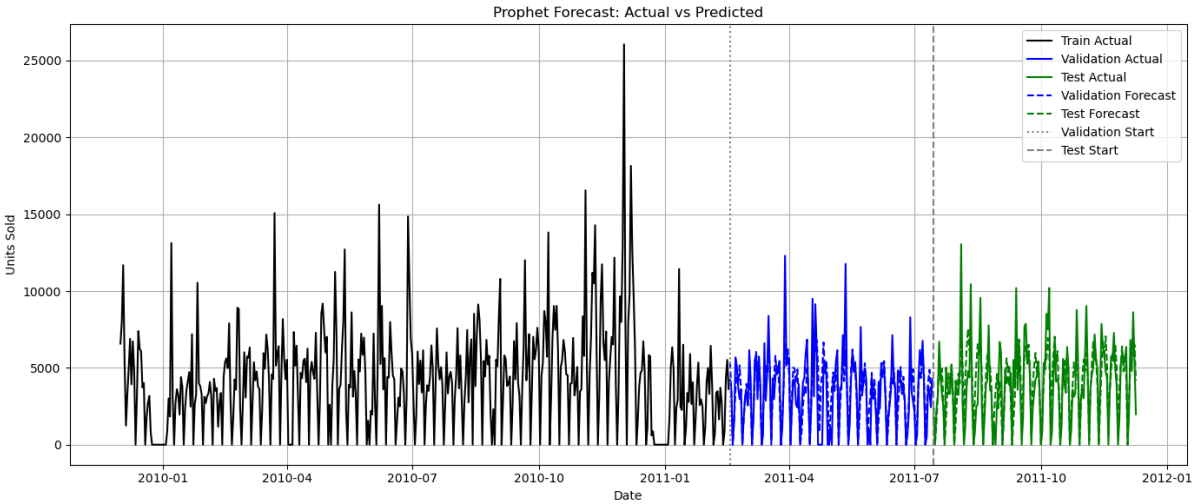


Figure 90: Total Product Sales with Filtered Explanatory Variables

6.4.5 Prophet on Total Sales

Model Variant	sMAPE Before Tuning (%)	sMAPE After Tuning (%)
Total Sales Only	33.21	31.83
Total + Explanatory Variables	40.75	20.87
Total + Extra Explanatory (Lags + Rolling)	38.12	20.29
Total + Filtered (Correlated) Variables	30.27	**16.95**

Table 12: Comparison of Prophet Forecasting Performance Across Different Feature Sets

Table 18 highlights the impact of different feature sets on the forecasting performance of the Prophet model, measured by validation sMAPE before and after tuning. Starting with only total product sales, the model achieved a modest improvement after tuning, reducing sMAPE from 33.21% to 31.83%. Introducing a large set of explanatory variables, including economic indicators, calendar events, and behavioral features, significantly improved accuracy, cutting the validation sMAPE from 40.75% to 20.87%. Adding time series features such as lags and rolling statistics led to a further gain, lowering post-tuning sMAPE to 20.29%. Finally, by applying a correlation-based feature selection and retaining only the most predictive variables, the model achieved its best performance, reducing validation sMAPE from 30.27% to **16.95%**. These results show the importance of thoughtful feature engineering and selection, indicating that both external variables and time-series features can substantially improve forecasting accuracy.

6.4.6 K-Means Clusters

Prophet's performance on the K-Means clusters benefited from both cluster-level tuning and the inclusion of carefully selected explanatory variables. Each cluster model was trained with a consistent set of regressors identified through correlation analysis with the target variable, including lag features, rolling statistics, and weekday indicators. Tuning was performed using a defined grid of parameters (changepoint_prior_scale, seasonality_prior_scale, and fourier_order), allowing each cluster to optimize its fit individually. This approach led to strong performance in several clusters—most notably Cluster 7, which achieved a validation sMAPE of 17.66%. However, results varied across clusters, with Cluster 9 reaching a much higher error at 53.58%, suggesting less predictable or more irregular sales behavior. These findings show the value of both clustering and targeted feature selection, while also highlighting the limitations of Prophet when faced with noisy or volatile demand patterns.

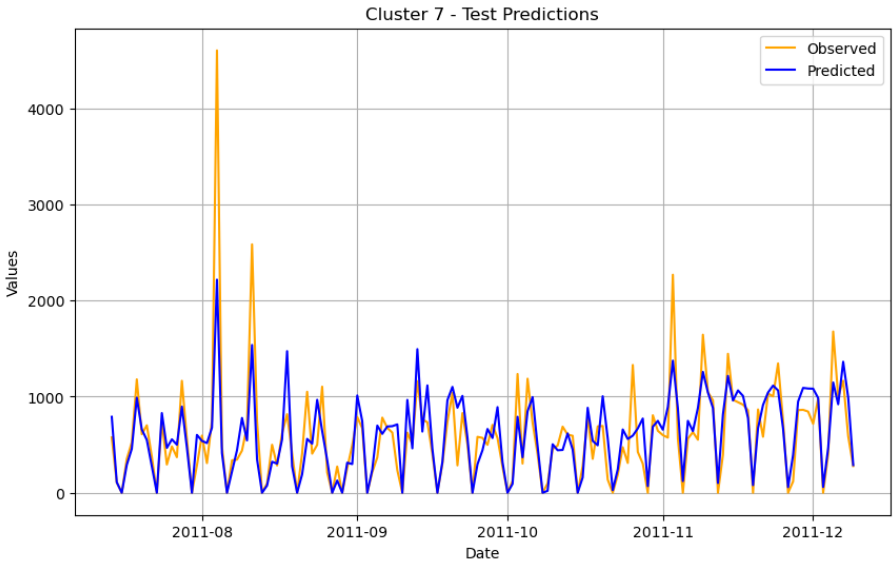


Figure 91: Best K-Means Cluster for Prophet

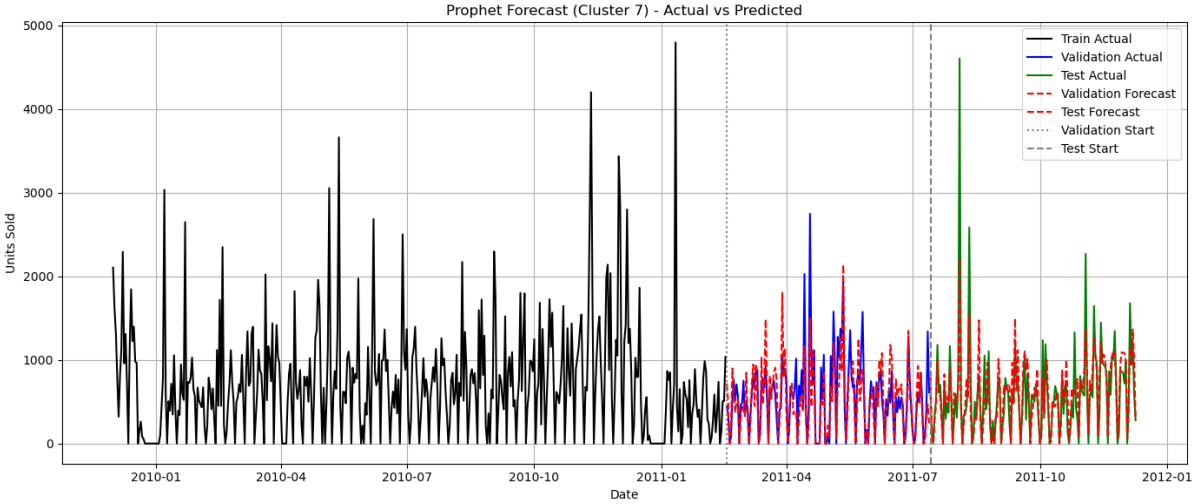


Figure 92: Best K-Means Cluster for Prophet

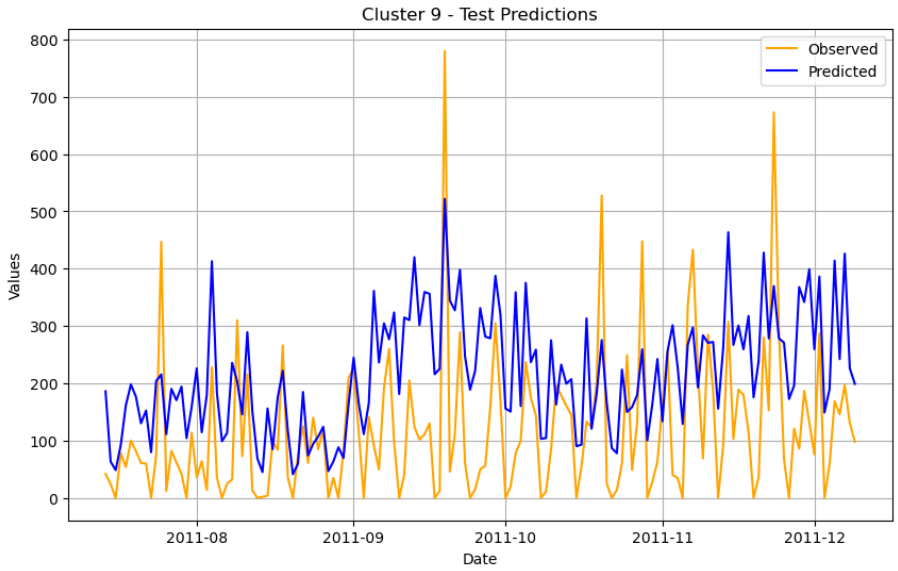


Figure 93: Worst K-Means Cluster for Prophet

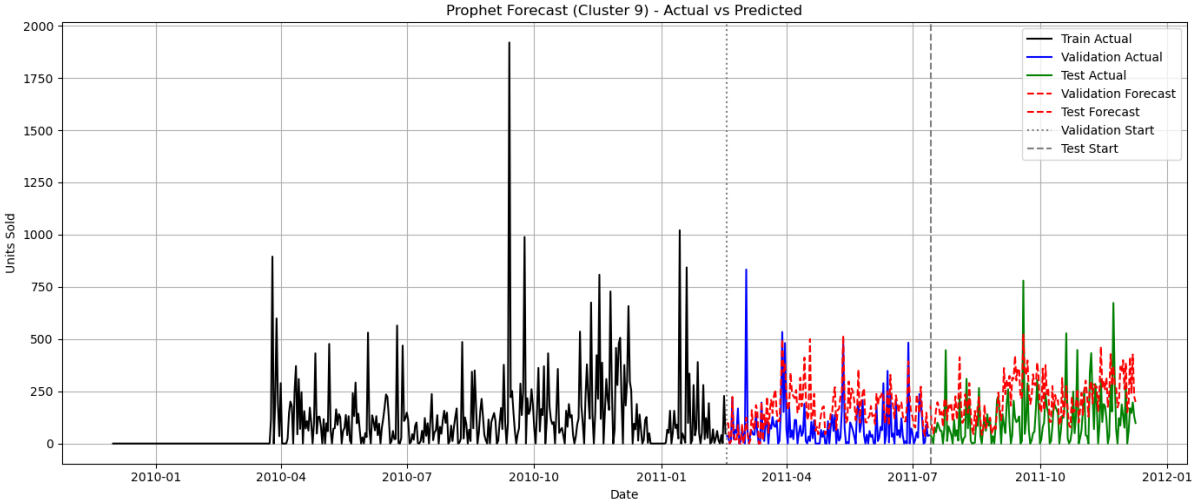


Figure 94: Worst K-Means Cluster for Prophet

6.4.7 DBSCAN Clusters

Prophet's performance on the DBSCAN-generated clusters varied considerably, reflecting the algorithm's sensitivity to the underlying demand patterns within each group. The best-performing DBSCAN cluster achieved a strong validation sMAPE of 20.00%, on par with the best K-Means cluster, while the worst-performing cluster (Cluster 16) reached a much higher error of 53.71%. As seen in the plot for Cluster 16, Prophet struggled to capture the magnitude and frequency of the large demand spikes, consistently underpredicting high-volume days. This suggests that although DBSCAN can isolate dense, behaviorally coherent product clusters, the presence of high volatility or erratic demand within some clusters (like Cluster 16) poses a challenge for Prophet's smooth trend and seasonality modeling. In comparison, K-Means clusters generally yielded more stable performance with tighter error ranges, likely due to their more consistent group composition. Overall, DBSCAN enabled highly accurate forecasts for select clusters but also showed Prophet's limitations when faced with extreme variability.

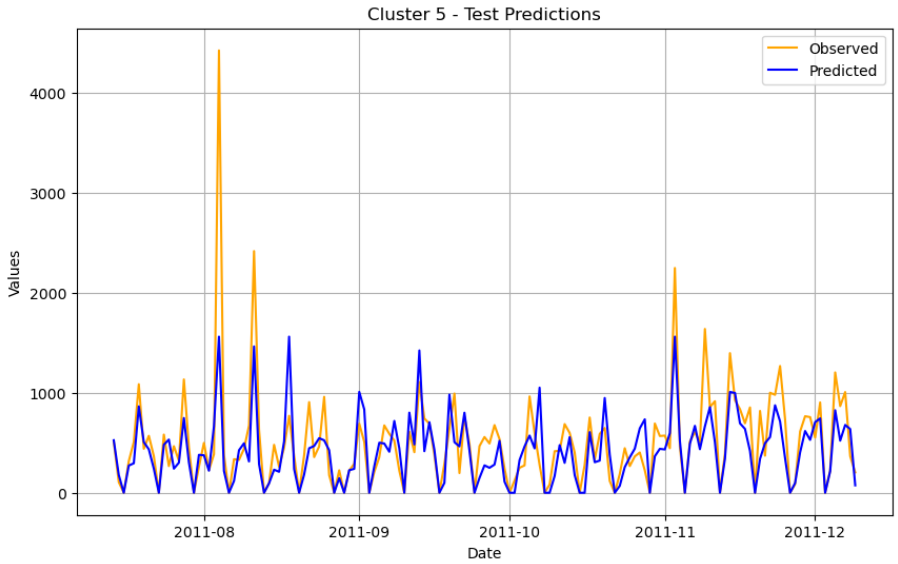


Figure 95: Best DBSCAN Cluster for Prophet

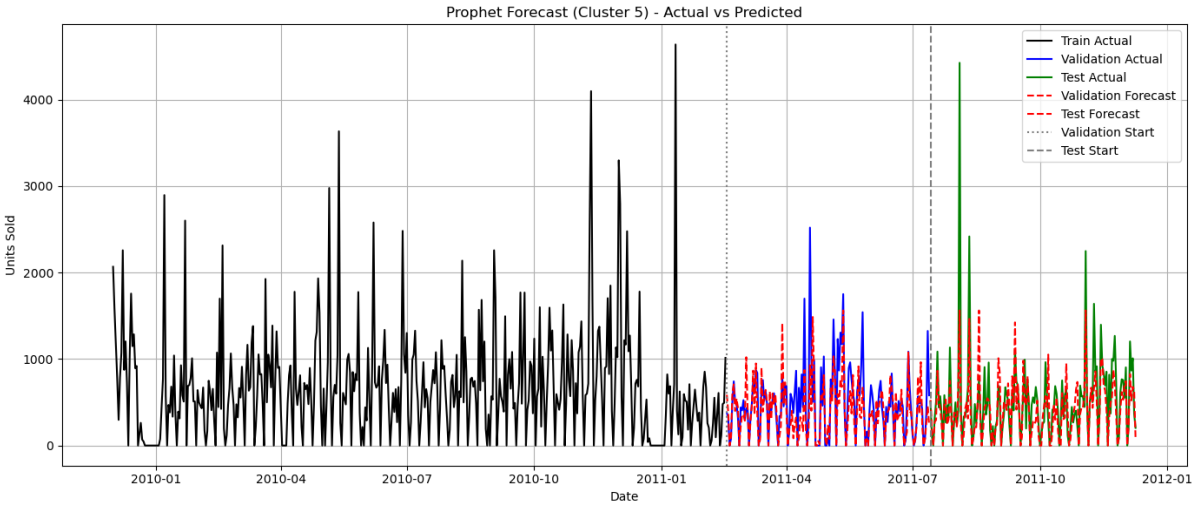


Figure 96: Best DBSCAN Cluster for Prophet

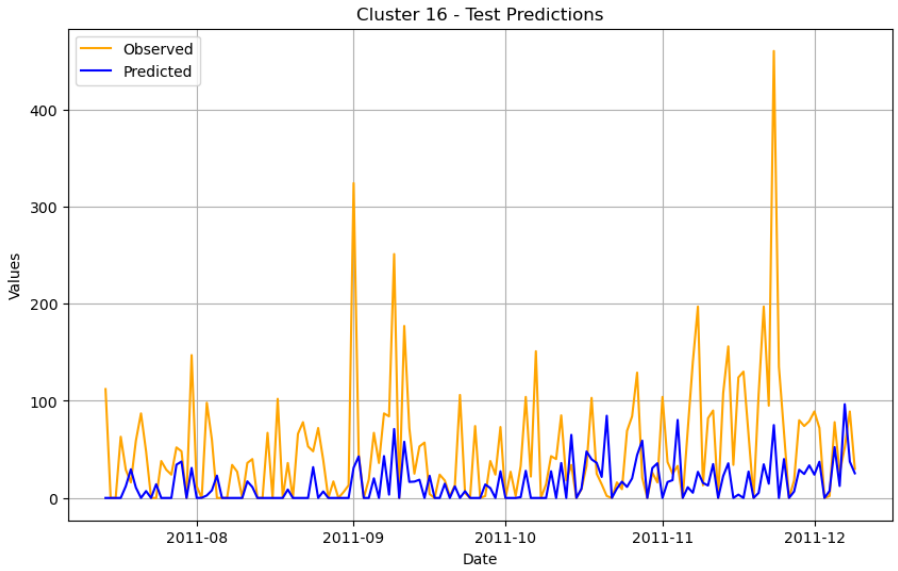


Figure 97: Worst DBSCAN Cluster for Prophet

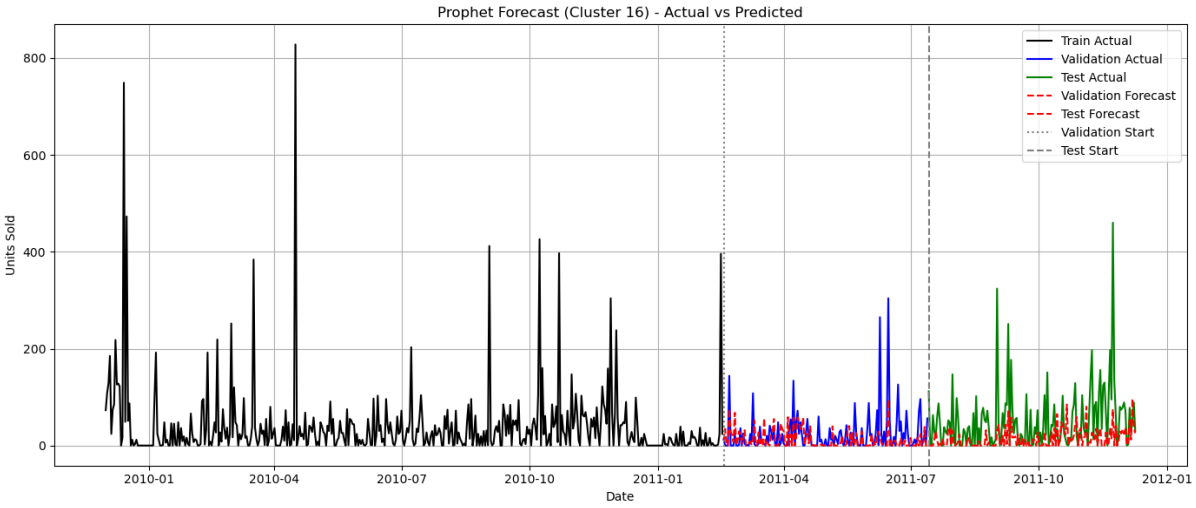


Figure 98: Worst DBSCAN Cluster for Prophet

6.4.8 HDBSCAN Clusters

Prophet's performance on HDBSCAN clusters varied notably across groups, reflecting the heterogeneous nature of the product segments identified by this density-based clustering method. The model performed best on Cluster 9, achieving a strong validation sMAPE of 15.90%, where its predictions closely followed the observed trend despite volatility. Conversely, Cluster 2 presented a significant challenge, with a high sMAPE of 53.83%, suggesting that the series exhibited irregular or extreme fluctuations that Prophet's additive model struggled to capture. Overall, while HDBSCAN helped isolate some well-behaved and predictable clusters, others exposed Prophet's limitations in modeling sharp demand spikes or irregular seasonality.

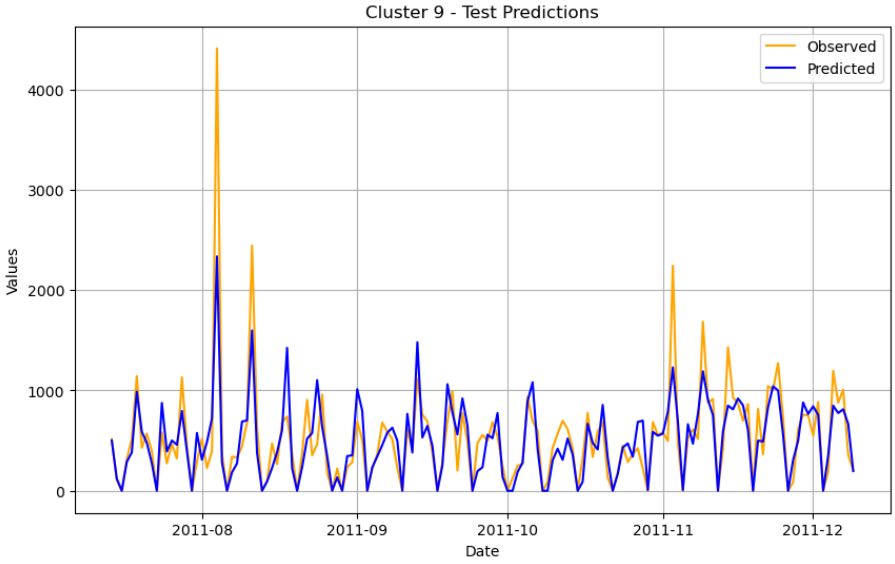


Figure 99: Best HDBSCAN Cluster for Prophet

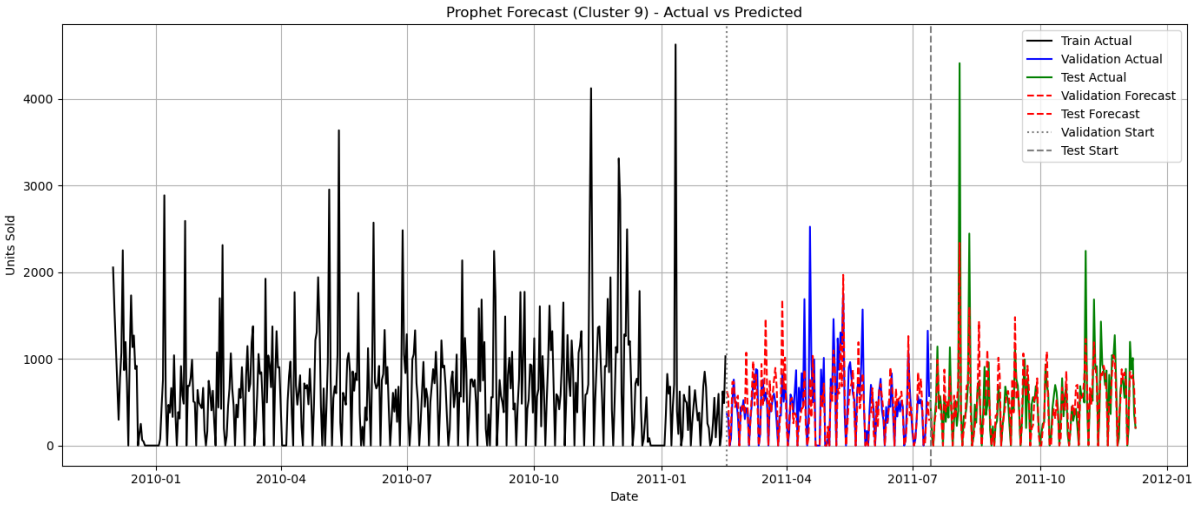


Figure 100: Best HDBSCAN Cluster for Prophet

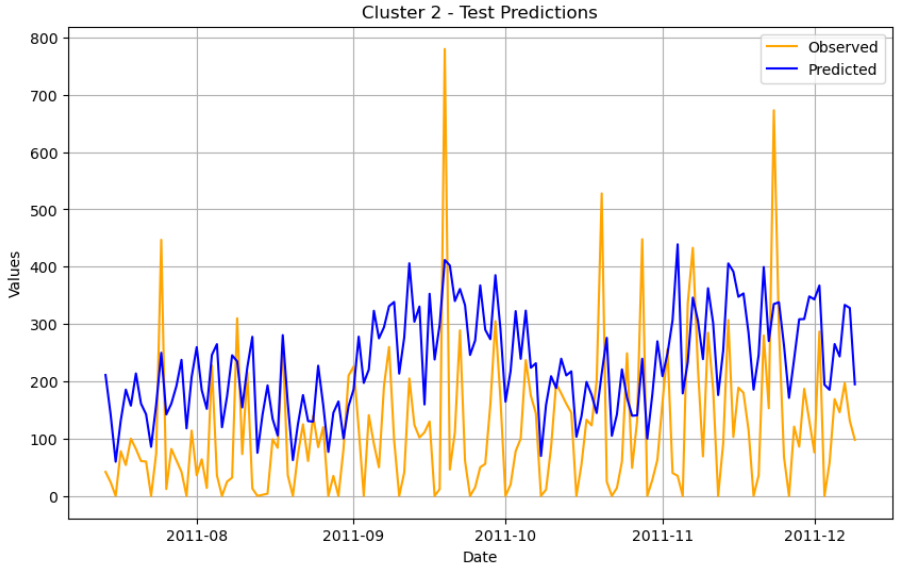


Figure 101: Worst HDBSCAN Cluster for Prophet

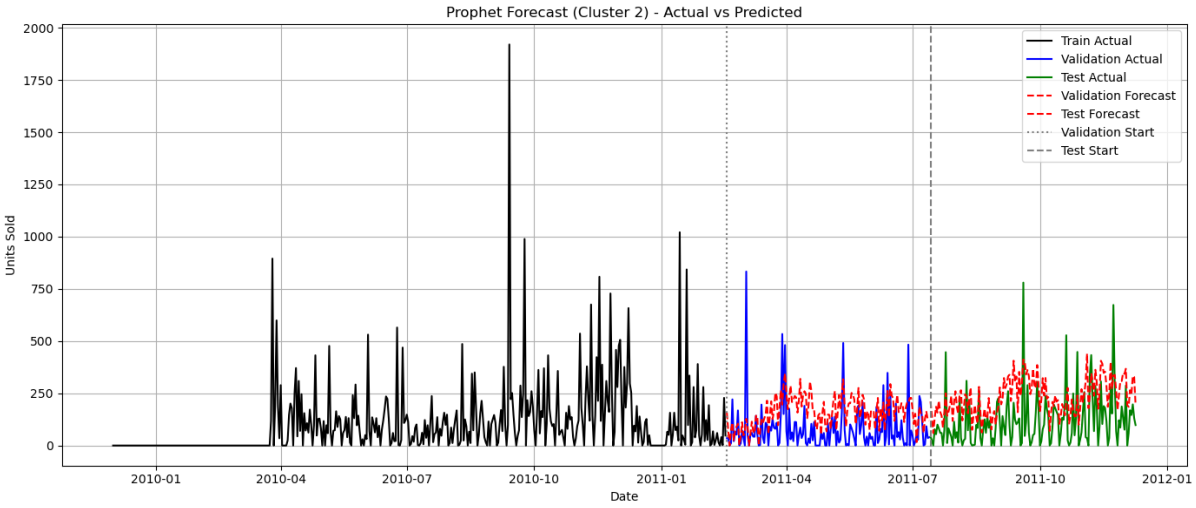


Figure 102: Worst HDBSCAN Cluster for Prophet

6.4.9 Spectral Clusters

Prophet's performance on the spectral clustering results showed moderate forecasting success, with some clusters achieving relatively low error rates while others lagged behind. The best-performing cluster was Cluster 1, with a validation SMAPE of 18.74%, closely followed by Cluster 0 at 20.02% and Cluster 14 at 22.30%. These clusters appear to have had more stable or predictable patterns, which Prophet handled well. On the other end, Cluster 11 had the worst performance with a SMAPE of 52.79%, indicating substantial difficulty in capturing its underlying dynamics, possibly due to erratic or highly variable sales behavior. Overall, spectral clustering offered some useful groupings for Prophet to model, though performance varied significantly depending on the characteristics of each cluster.

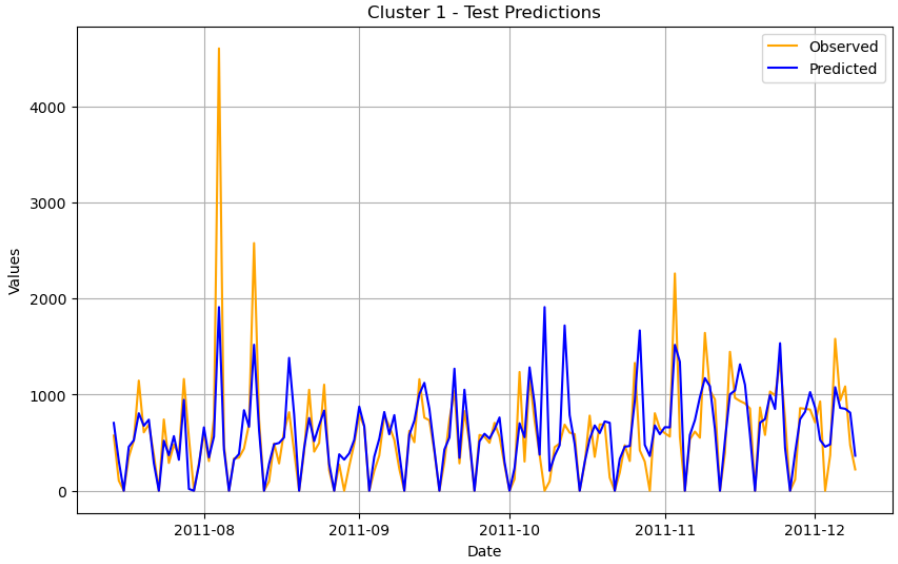


Figure 103: Best Spectral Cluster for Prophet

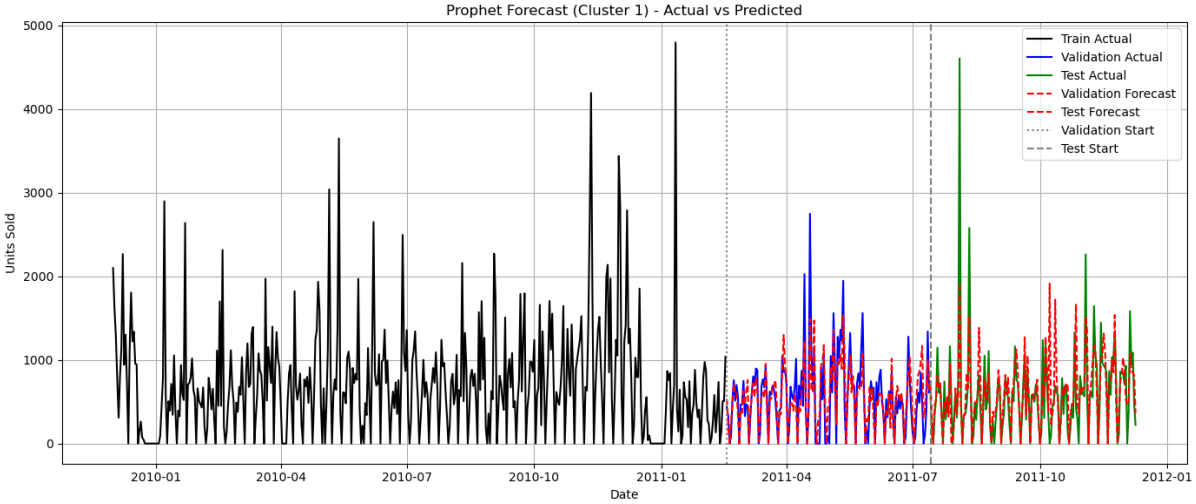


Figure 104: Best Spectral Cluster for Prophet

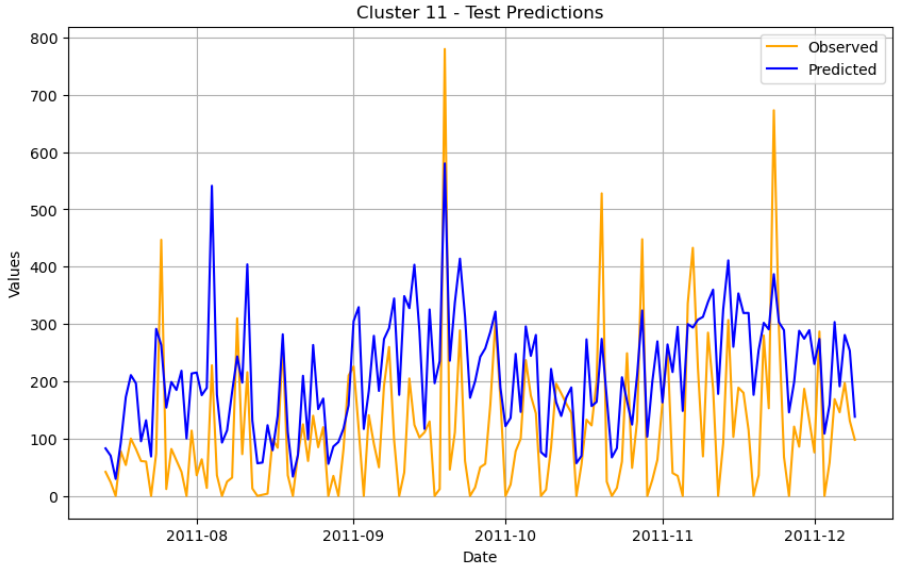


Figure 105: Worst Spectral Cluster for Prophet

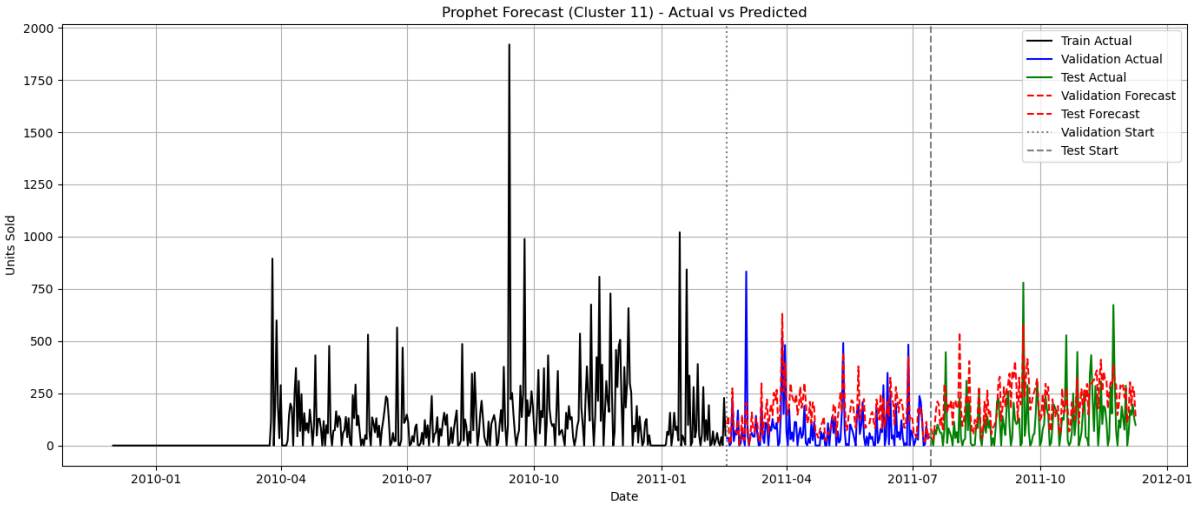


Figure 106: Worst Spectral Cluster for Prophet

6.4.10 Sales for Individual Products

Prophet demonstrated solid forecasting performance on individual products with high sales volume and clear seasonal trends. For instance, products like the "hanging heart zinc tlight holder" and "white hanging heart tlight holder" saw low SMAPE values (17.8% and 26.8%, respectively), indicating accurate predictions. These items tend to exhibit smoother, more consistent purchasing patterns, allowing Prophet's additive model—particularly its decomposition of trend and seasonality—to fit the data well. The model effectively captured weekly cycles and holiday effects for these series, especially when enhanced with engineered features such as lags, rolling averages, and day-of-week indicators.

However, Prophet struggled on products with erratic, sparse, or highly intermittent demand. Items like the "memo board retrospot design" and "12 pencils tall tube woodland" resulted in much higher SMAPE values (over 65%), with forecasts failing to capture sharp, isolated spikes in unit sales. Prophet's reliance on smooth trend and seasonality assumptions made it poorly suited for capturing irregular bursts of demand or promotional-driven sales events. Additionally, its inability to model intra-day patterns or capture dependencies beyond additive components limited its accuracy on volatile series. This suggests Prophet works best when time series exhibit clear structure and enough historical data—while more advanced, probabilistic, or deep learning models may be necessary for highly irregular product sales.

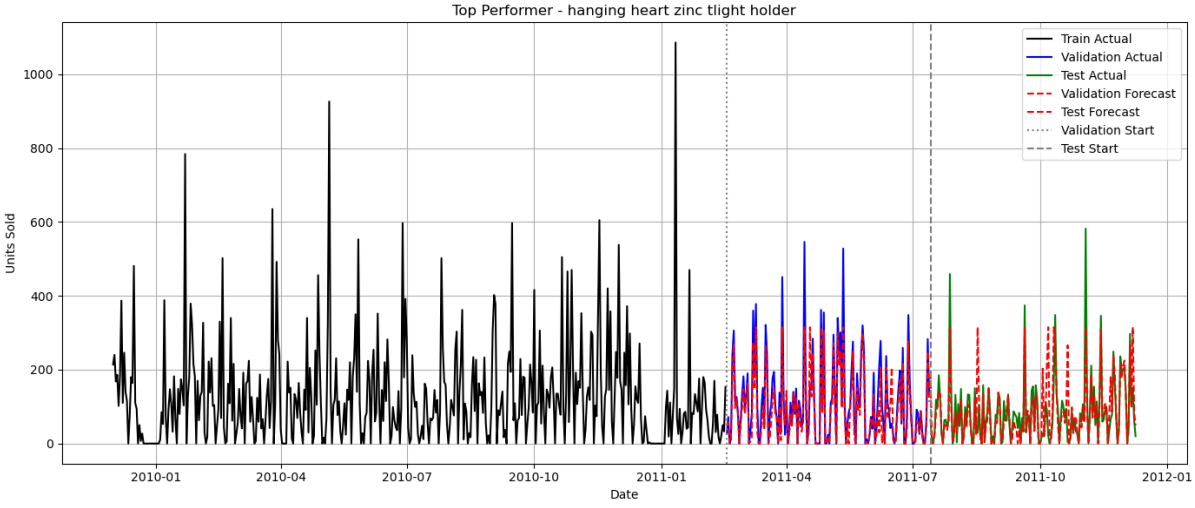


Figure 107: Best Individual Product Predictions for Prophet

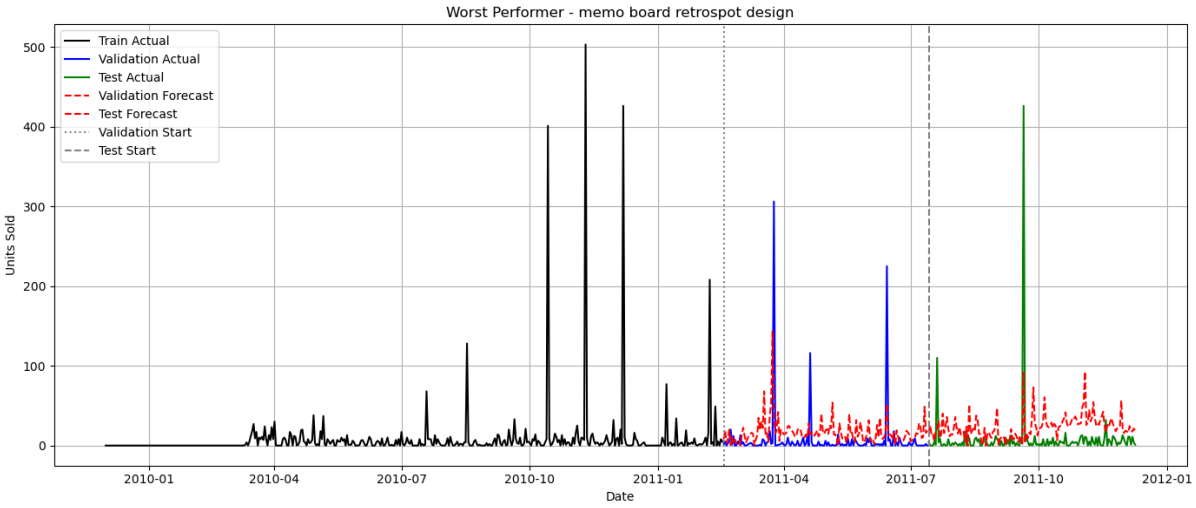


Figure 108: Worst Individual Product Predictions for Prophet

6.5 RNN

The RNN model leverages the same correlation matrix and explanatory variables used in previous models. To enhance forecasting performance, the model was trained on a smoothed version of the dataset, as the original data yielded suboptimal results. The RNN was evaluated across four clustering methods and also on an individual product, *12 pencils small tube red spotty*, to compare performance on single-product data versus clustered data.

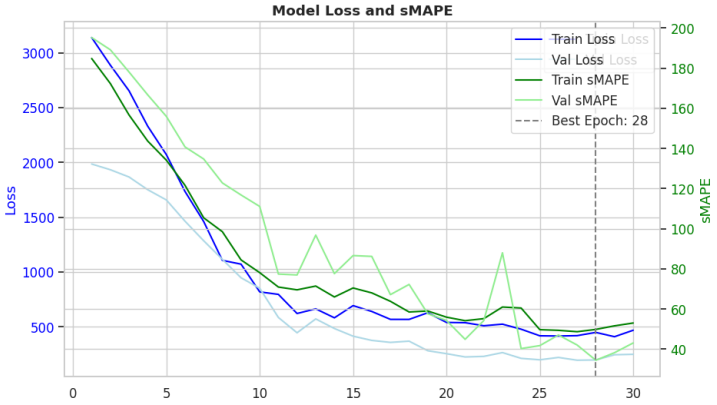


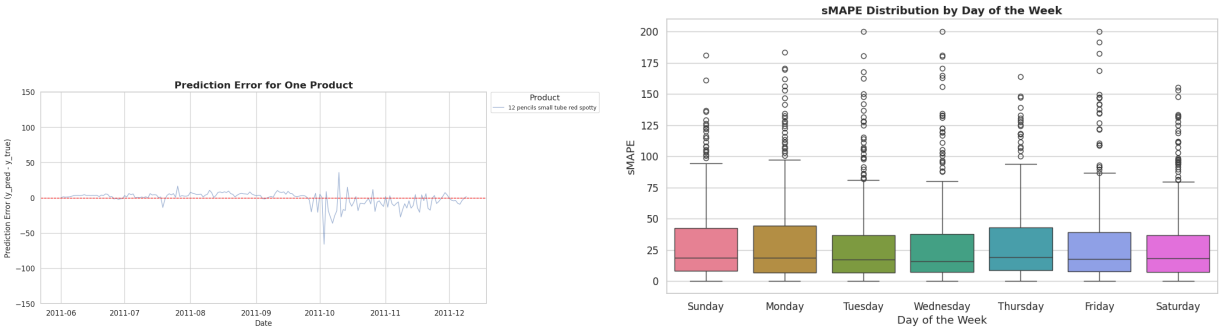
Figure 109: Training History for One Product: Model Loss and sMAPE

Figure 109 illustrates the model's training and validation loss and sMAPE over epochs. The 28th epoch was selected as the final model checkpoint, corresponding to the lowest validation loss and validation sMAPE.

Table 13: sMAPE Scores for One Product

Product	Train sMAPE	Val sMAPE	Test sMAPE
12 pencils small tube red spotty	48.66	34.45	40.49

As shown in Table 13, the model achieved a training sMAPE of approximately 48.66%, a validation sMAPE of 34.45%, and a test sMAPE of 40.49%.

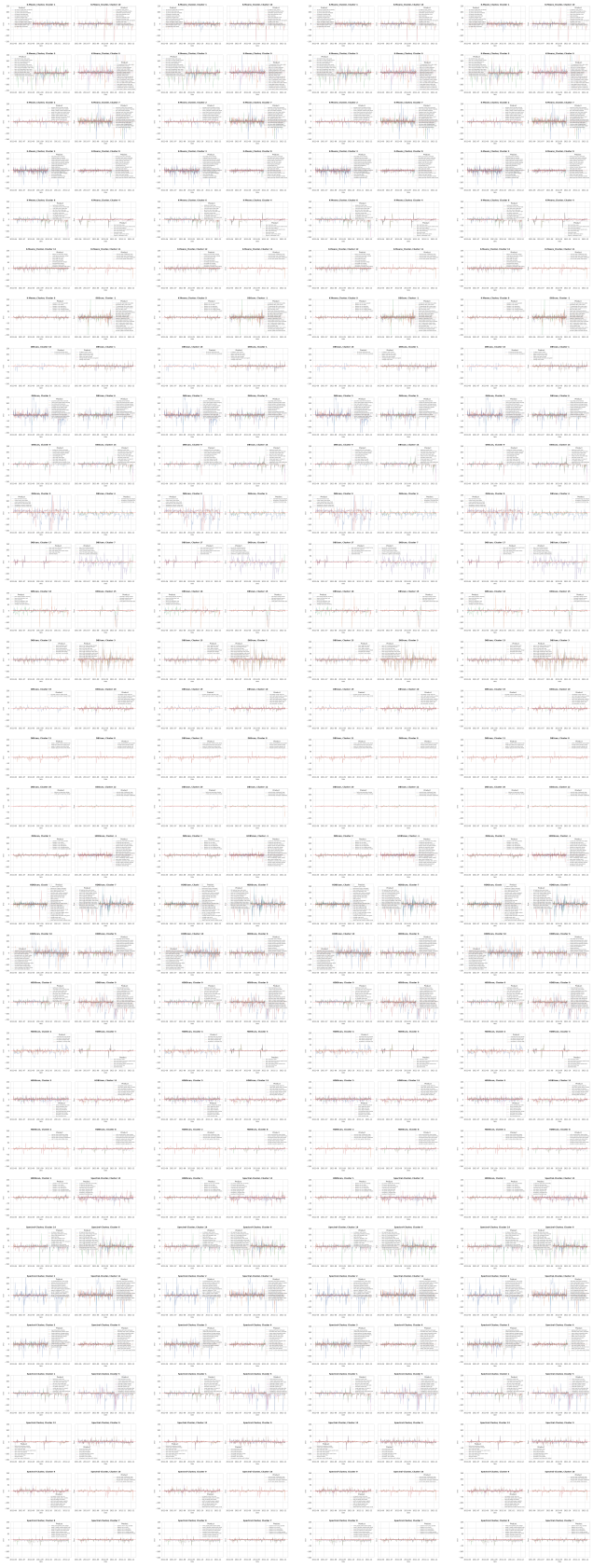


(a) Prediction Error

(b) sMAPE Box Plot

Figure 110: RNN Prediction Error for One Plot (Left) and sMAPE Box Plot through the Week (Right)

Figure 110 shows the RNN's prediction errors for *12 pencils small tube red spotty*. Predictions were generally stable, with some notable deviations occurring between October and November. Now, we will look at how RNN performs when building one model per cluster.



(a) DBSCAN (b) HDBSCAN (c) K-Means (d) Spectral

Figure 111: RNN prediction errors across different clustering methods. For high-resolution versions, refer to the project repository.

Table 14: sMAPE scores for each Clustering Algorithms and Clusters

Algorithm	Cluster	Train sMAPE	Val sMAPE	Test sMAPE
DBScan	-1	38.06	34.72	155.34
	0	65.18	50.19	177.52
	1	37.35	33.32	118.88
	2	69.42	37.65	115.92
	3	52.89	37.95	194.05
	4	31.38	28.72	88.52
	5	38.95	33.10	141.14
	6	41.63	34.90	116.32
	7	62.16	43.51	185.31
	8	68.50	53.86	107.84
	9	44.96	42.01	111.91
	10	42.50	46.11	97.20
	11	70.95	43.32	176.72
	12	75.43	39.11	83.53
	13	55.61	33.19	128.12
	14	38.87	32.93	89.87
	15	42.83	34.16	124.20
	16	34.89	55.14	71.83
	17	49.70	36.42	142.86
	18	37.96	36.17	73.61
	19	38.34	28.90	91.02
	20	50.20	41.80	121.52
	Avg	**49.44**	**38.96**	**123.33**
HDBScan	-1	39.95	36.35	168.95
	0	38.67	33.11	125.75
	1	69.96	40.55	127.63
	2	66.94	34.55	121.29
	3	45.11	37.35	144.14
	4	28.18	28.44	55.38
	5	45.26	31.61	130.09
	6	43.06	33.16	79.20
	7	48.64	36.87	197.07
	8	70.77	47.97	179.62
	9	67.11	39.85	166.86
	10	37.72	32.79	158.81
	11	40.01	42.15	168.24
	12	39.37	36.36	121.50
	Avg	**48.63**	**36.51**	**138.90**
K-Means	0	62.09	38.25	191.07
	1	35.55	38.54	131.71
	2	66.77	48.40	160.43
	3	38.32	31.41	102.91
	4	54.51	44.52	145.87
	5	49.36	35.49	182.89
	6	41.83	32.68	113.22
	7	39.95	31.98	165.04
	8	40.35	36.23	95.12
	9	69.72	37.61	111.79
	10	39.04	35.39	129.02
	11	65.24	35.30	96.46
	12	50.83	35.11	153.00
	Avg	**50.27**	**36.99**	**136.81**
Spectral	0	47.10	34.20	136.43
	1	39.00	33.61	106.53
	2	50.34	39.93	186.16
	3	40.76	33.13	86.75
	4	41.48	40.21	94.24
	5	59.72	38.14	185.88
	6	50.51	40.94	149.46
	7	71.26	39.40	87.34
	8	70.94	46.50	182.95
	9	38.78	40.43	130.69
	10	68.02	38.27	95.78
	11	37.05	33.83	163.92
	12	44.21	41.62	124.46
	13	33.50	33.07	111.27
	14	37.78	35.44	129.27
	Avg	**48.70**	**37.91**	**131.41**

Table 14 reports the sMAPE across training, validation, and test sets for each cluster generated by the four clustering algorithms: DBScan, HDBScan, K-Means, and Spectral Clustering. Each cluster was assigned its own RNN model to better capture localized temporal patterns in product-level sales.

Overall, the models exhibit reasonably low sMAPE scores on the training and validation sets, typically ranging from 30 to 50. However, test set performance is significantly worse, with average sMAPE values between 120 and 140. This discrepancy suggests possible overfitting, a distributional shift between train and test data, or sparsity in the test samples. Additionally, the variation in cluster sizes likely influenced performance—smaller clusters tended to yield lower test sMAPE scores, likely due to more homogeneous patterns within those subsets.

Among the clustering methods, DBScan achieved the lowest average test sMAPE (123.33), slightly outperforming Spectral Clustering (131.41), K-Means (136.81), and HDBScan (138.90). However, this advantage should be interpreted cautiously. DBScan produced a large number of small clusters (including noise, labeled as -1), and these small clusters, especially those containing only a few highly consistent products, tended to perform better. For instance, cluster 4 under DBScan and cluster 4 under HDBScan had notably low test errors. While DBScan showed slightly better performance on average, the results indicate that clustering quality, and by extension, forecasting accuracy, is highly sensitive to the size, cohesion, and interpretability of the clusters. Future clustering work could explore refined clustering strategies or hybrid models to better balance model specificity and generalization.

When comparing single-product prediction to multi-product prediction, the RNN demonstrates significantly better performance on the single-product case, as reflected by a noticeably lower test sMAPE. This discrepancy may be attributed to two main factors:

- Temporal mismatch in the data split: The training period spans January to December of the previous year, while the test set covers the latter half of the following year. If consumer behavior shifts year over year, the model may struggle to generalize, especially with limited historical data. Furthermore, validation performance may appear better because it corresponds to the first half of the test year, a period that may exhibit more stable and predictable purchasing patterns.

- Ineffective clustering: The clustering algorithm may not be grouping products with similar demand behaviors, leading to heterogeneity within clusters. This undermines the model's ability to learn consistent patterns across grouped products, reducing its generalization performance compared to modeling a single, coherent product series.

In conclusion, while the RNN model is capable of handling both single-product and multi-series forecasts, it performs significantly better when trained on a single product. This suggests that the current clustering methods may not effectively group products with similar demand patterns, and that temporal inconsistencies in the train-test split may further hinder model generalization. These findings highlight the importance of refining both data segmentation and clustering strategies to improve multi-product forecasting performance.

6.6 TFT

As described in Section 5.6, we trained a single Temporal Fusion Transformer model across all products using the first 678 days of data, reserving the final 60 days for evaluation. Hyperparameters were selected using Optuna over 50 trials, with the final configuration chosen based on validation performance. The model was trained using the Negative Binomial loss and optimized with the Ranger optimizer. Forecasting was performed in an autoregressive fashion over a 7-day horizon, using the same evaluation framework and metrics as described for the GLM, decision tree, and XGBoost models (see Section 6.1).

6.6.1 Performance by forecast horizon

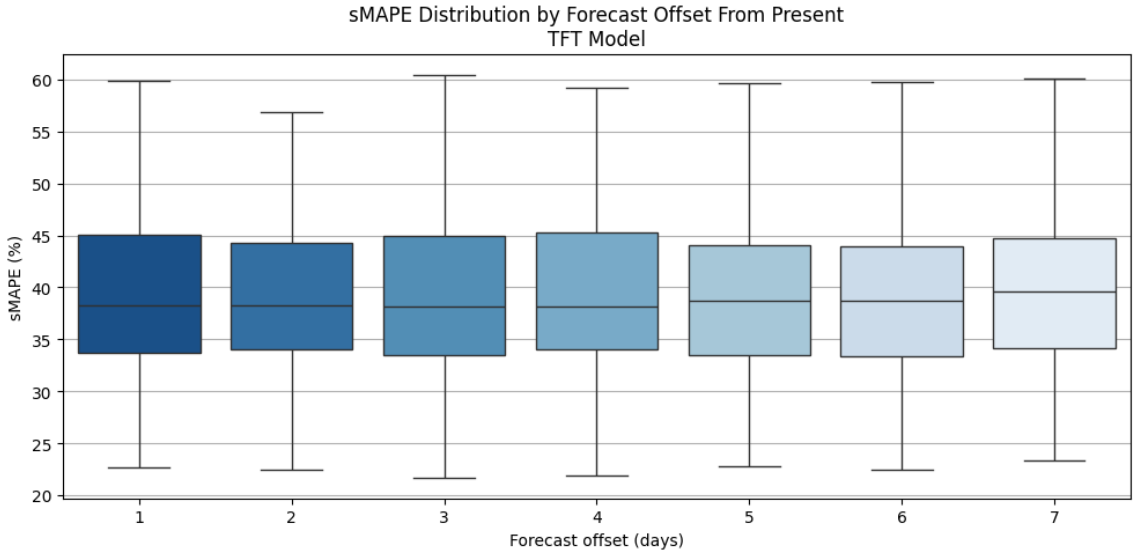


Figure 112: sMAPE distribution by forecast offset. Predictions made with the Temporal Fusion Transformer (TFT) model. Boxplots show consistently strong performance across all forecast lengths.

Table 15: sMAPE statistics by forecast offset for the TFT model. Total number of products: 142.

Forecast offset (days)	Mean	P10	P25	P50	P75	P90
1	39.84%	29.01%	33.71%	38.28%	45.02%	51.16%
2	39.90%	29.79%	34.07%	38.23%	44.32%	50.72%
3	39.76%	29.88%	33.45%	38.12%	44.91%	50.92%
4	39.82%	30.20%	34.03%	38.19%	45.28%	50.71%
5	39.83%	30.31%	33.44%	38.75%	44.09%	50.78%
6	39.60%	30.28%	33.35%	38.75%	43.96%	50.18%
7	40.49%	30.63%	34.11%	39.63%	44.76%	50.89%

Figure 112 and Table 15 summarize how forecast accuracy varies over the 7-day prediction horizon using the Temporal Fusion Transformer. The TFT achieves reasonable overall horizon-wise performance among all models evaluated, with average sMAPE values consistently below 40.5%. The 1-day forecast achieves a mean sMAPE of 39.84%, and even at the 7-day horizon, performance only rises marginally to 40.49%.

The interquartile ranges remain narrow and stable across all forecast offsets, with medians centered around 38–39.6%. This consistency highlights the model's ability to maintain forecast quality over time without degradation. These results suggest that TFT effectively integrates temporal, categorical, and macroeconomic features—combined with autoregressive context—to produce highly reliable multi-step forecasts.

6.6.2 Performance by day of week

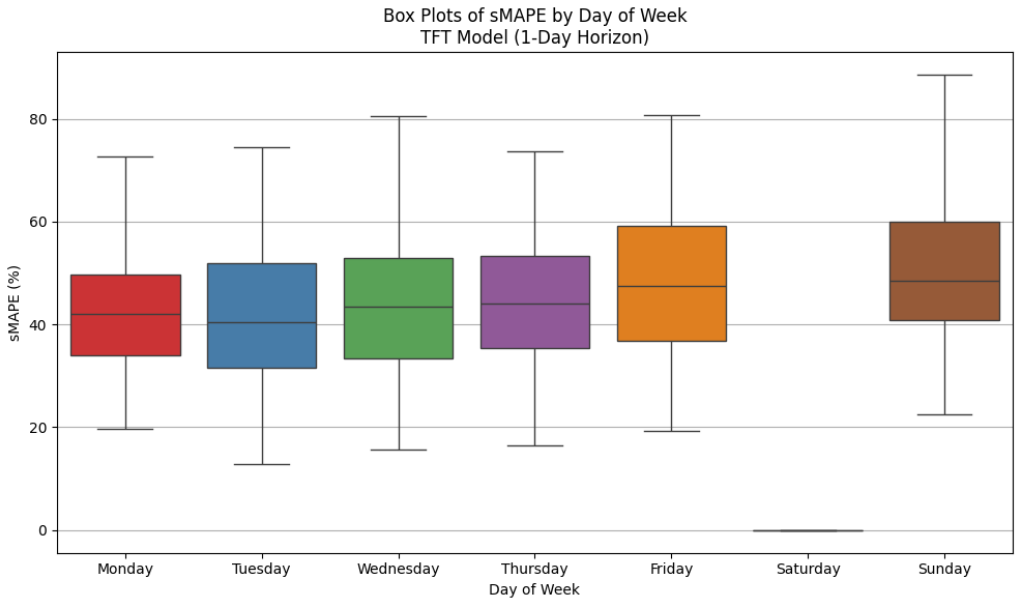


Figure 113: Box plots of sMAPE by day of the week for the TFT model (1-day forecast horizon). Each box summarizes the distribution of sMAPE values across 142 products.

Table 16: sMAPE statistics by day of the week for the TFT model. Total number of products: 142.

Day of Week	Mean	P25	P50	P75	Min	Max
Monday	43.79%	34.06%	42.08%	49.76%	19.70%	100.00%
Tuesday	42.85%	31.67%	40.36%	51.87%	12.89%	100.00%
Wednesday	44.02%	33.43%	43.49%	52.86%	15.75%	100.00%
Thursday	45.63%	35.49%	44.11%	53.35%	16.47%	100.00%
Friday	48.47%	36.87%	47.59%	59.27%	19.27%	100.00%
Saturday	5.02%	0.00%	0.00%	0.00%	0.00%	87.50%
Sunday	50.46%	40.83%	48.41%	60.05%	22.49%	100.00%

Figure 113 and Table 16 present the day-of-week performance of the Temporal Fusion Transformer. Forecast accuracy is relatively stable on weekdays, with mean sMAPE values ranging from 42.85% (Tuesday) to 45.63% (Thursday). Friday and Sunday show elevated error rates, with mean sMAPE values of 48.47% and 50.46%, respectively.

Saturday again exhibits a distinct pattern. Most products report a sMAPE of 0%, suggesting the model successfully learned that sales are typically zero on this day. However, a few large errors still occur, possibly due to rare events with actual sales.

6.6.3 Case studies

To better understand how the TFT model behaves across different product types, we examine a subset of individual predictions.

Figure 114 presents observed and predicted daily sales for four selected products with varying sMAPE scores. For *Lunch bag black skull* and *Natural slate heart chalkboard*, the model provides a reasonably close fit to actual sales levels, despite some underestimation of short-term fluctuations. *Love building block word* demonstrates a more noticeable smoothing effect, with the TFT underpredicting sharp spikes

while still tracking overall trends (sMAPE $\approx 32.44\%$). *Toilet metal sign* is a challenging case—predictions remain conservative and fail to capture the bursty, irregular nature of the true sales, leading to a high sMAPE of 51.31%.

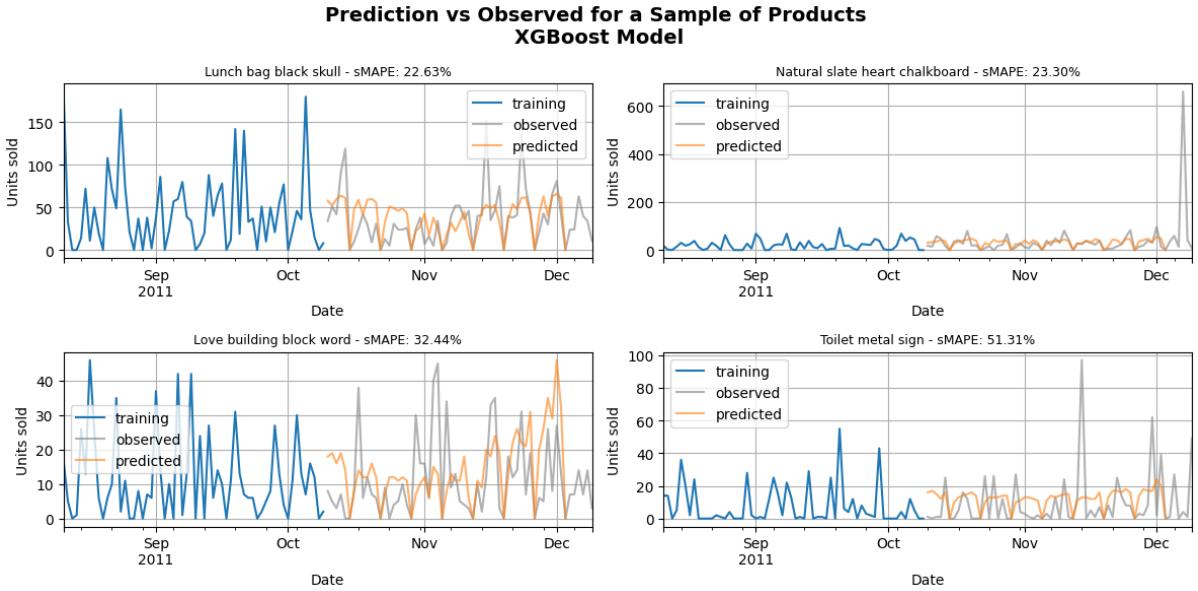


Figure 114: Predicted vs. observed sales for a sample of products using the TFT model. Each panel shows the last 60 days of training and the 60-day test window. Products are selected to illustrate a range of model performance.

To further assess model behavior, we examine the distributions of predicted and observed sales in log-transformed space. Figure 115 shows histograms and kernel density estimates (KDEs) of log1p-transformed daily sales. These plots offer insight into the alignment between the forecast and actual demand distributions.

In *Lunch bag black skull*, the predicted and observed distributions are closely aligned in shape and location, indicating strong calibration. *Natural slate heart chalkboard* shows a slight underrepresentation of extreme values in the predicted distribution. For *Love building block word*, the predicted distribution is narrower, suggesting the model smoothed away high-variance behavior. *Toilet metal sign* again stands out with a predicted distribution that is overly concentrated around low values, failing to capture the multimodal, heavy-tailed nature of the actual sales distribution.

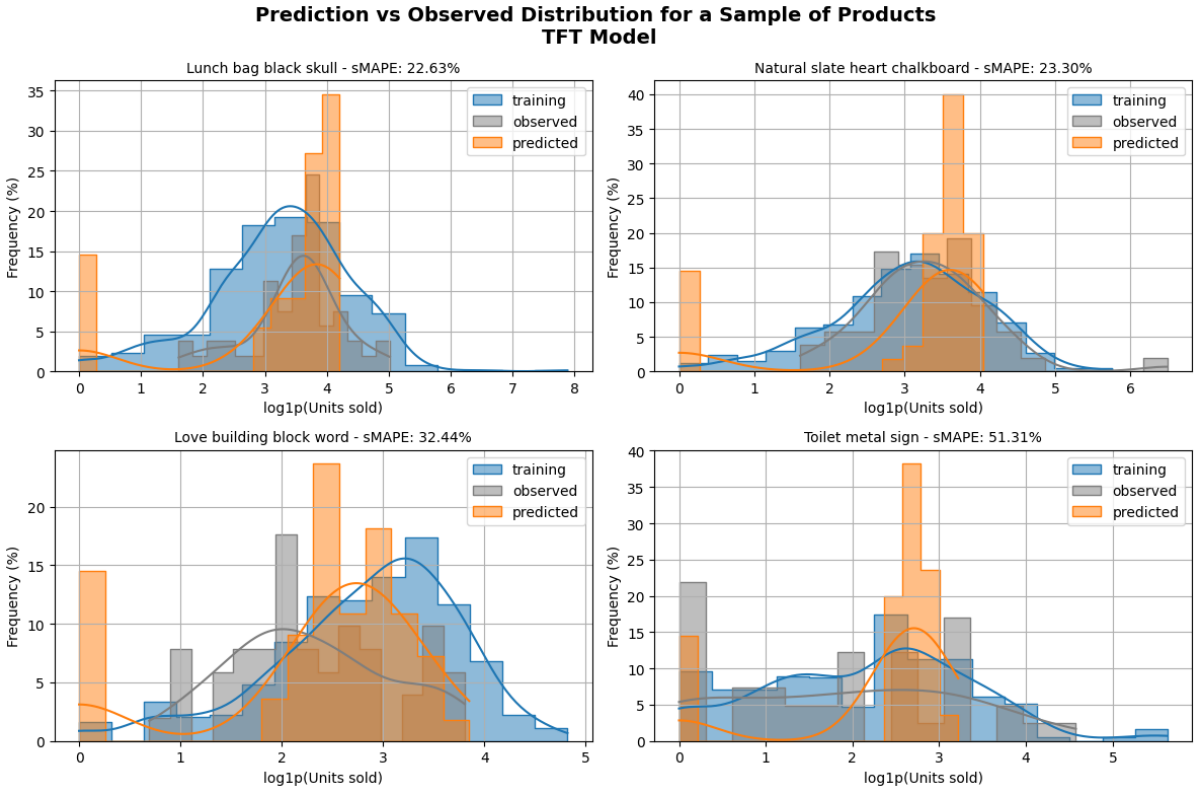


Figure 115: Distribution of log1p-transformed predicted vs. observed daily sales for a sample of products. Distributions are estimated over the 60-day test period, with training distributions included for context.

To evaluate bias and heteroskedasticity, we plot residuals (Predicted – Actual) against predicted values. Figure 116 shows scatter plots with LOESS-smoothed trend lines.

Lunch bag black skull and *Natural slate heart chalkboard* exhibit reasonably well-centered residuals across most prediction values, although the latter shows a slight underestimation bias for higher volumes. *Love building block word* displays mild heteroskedasticity, with residuals increasing alongside predicted values. *Toilet metal sign* shows strong systematic underprediction, with residuals trending sharply upward as predicted volume increases—evidence of the model's inability to anticipate higher demand bursts.

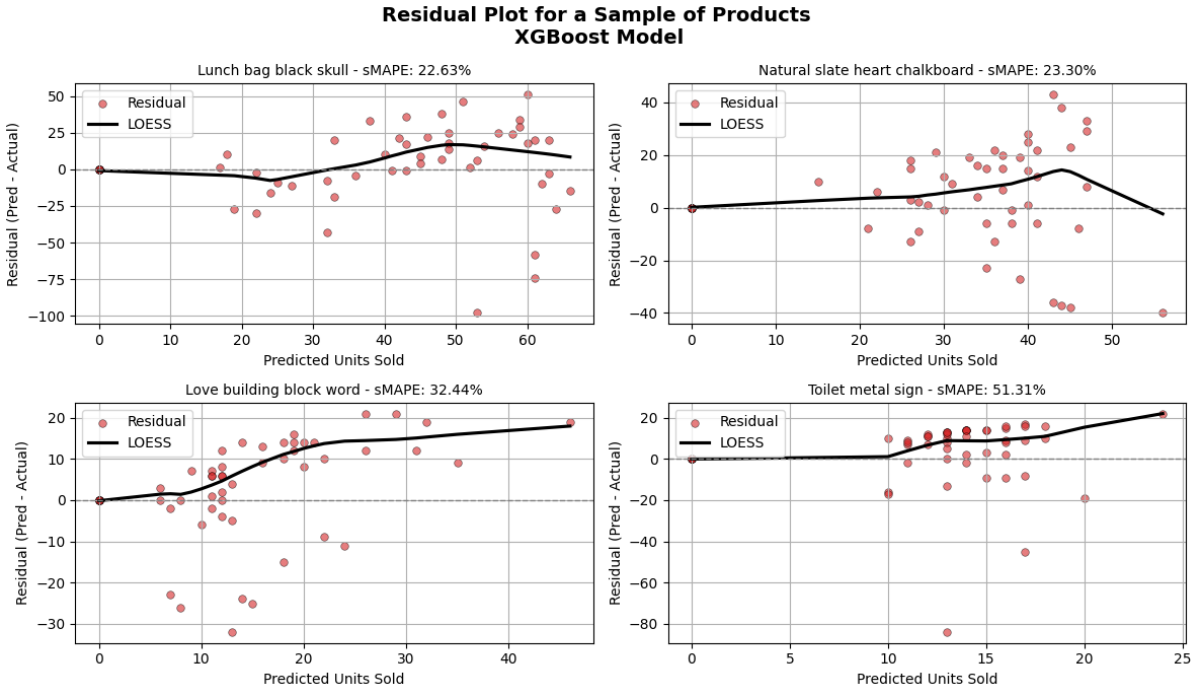


Figure 116: Residuals (Predicted – Actual) vs. Predicted Units Sold for a sample of products. A LOESS trend line is overlaid to highlight systematic bias across the prediction range.

6.7 DeepAR

When first training the DeepAR model after adjusting hyperparameters, quantile losses (an evaluation of how well predicted quantiles match the actual values) of around 0.689 were reported. Additionally, a test RMSE is 67.47 was reported (Figure 117). After checking and seeing that the data was greatly spread, with a max sale value of 5005 with only a mean of 25.85 and a median of 6, the target values were log-transformed in order to minimize the dispersion.

After adjusting the targets to be log transformed, performance can be seen to have increased with a lower quantile loss and a smaller RMSE. Given that the RMSE is now 1.6 in comparison to the new target mean of 1.93 and standard deviation of 1.72, we can conclude that the RMSE is very good for this log transformed training.

Performance

Name	Value
test:mean_wQuantileLoss	0.6889982223510742
train:loss:batch	3.161548376083374
train:progress	100
train:loss	3.1794123649597717
train:final_loss	3.0599069595336914
train:throughput	44.548099517822266
test:RMSE	67.47460174560547

Figure 117: Initial Performance of DeepAR on the Dataset

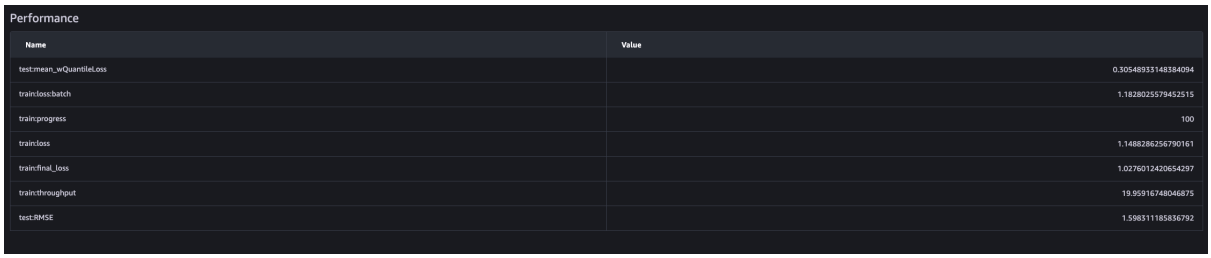
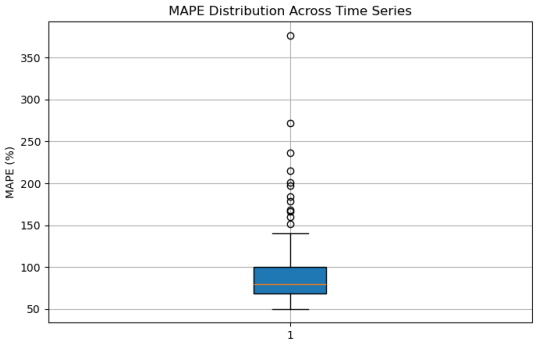


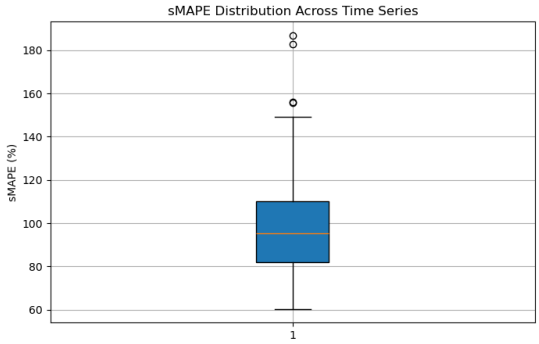
Figure 118: Log Performance of DeepAR on the Dataset

6.7.1 Predicting Product Sales

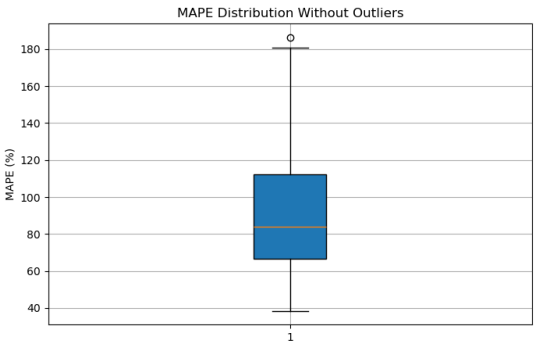
When evaluating the data on the test set, we use the median, or the 0.5 quantile prediction, as the forecast over the mean in order to mitigate outlier effects. Additionally, our log transformed targets are transformed back into the original sales values in order to accurately measure the effects of prediction. However, our results using MAPE and sMAPE are less than ideal with a Mean MAPE of 107.42% and and a Mean sMAPE of 92.51%. After filtering out outliers in our MAPE range, the average MAPE came down to 92.96% while the average sMAPE came down to 89.81%. Because MAPE generally yields a better result than sMAPE it is used here for comparison.



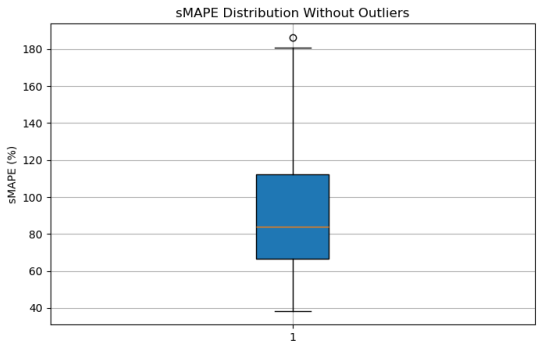
(a) Distributions of MAPE for different time series



(b) Distributions of sMAPE for different time series



(c) Distributions of MAPE for different time series after outlier removal



(d) Distributions of sMAPE for different time series after outlier removal

Figure 119: MAPE and sMAPE distribution with DeepAR predictions

The best time series prediction for this dataset seems to be item number 30 in our data which corresponds to "feltcraft 6 flower friends" with a sMAPE of 43.58%. Its performance is demonstrated below.

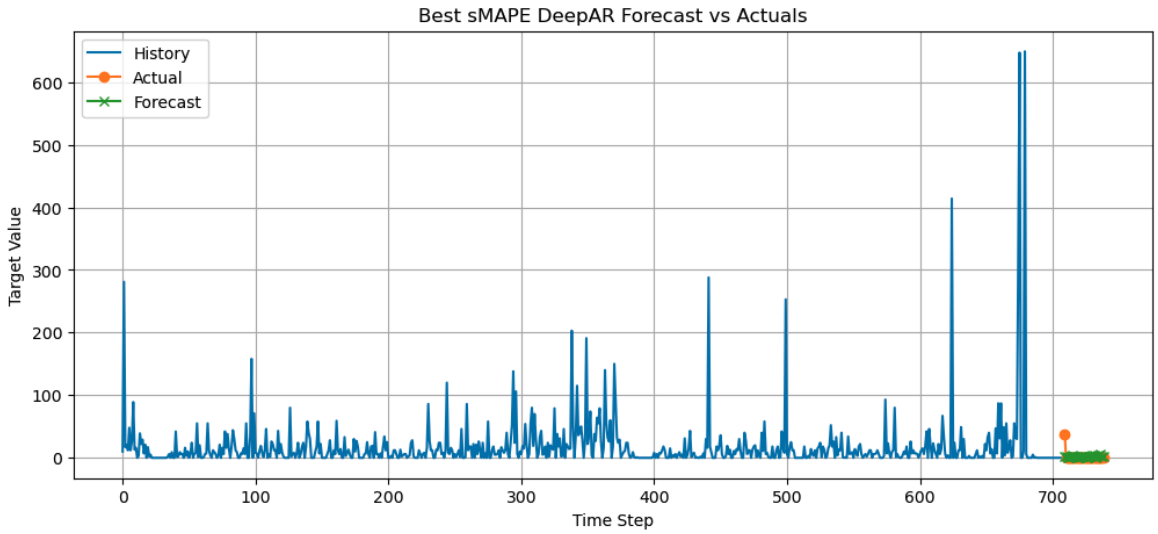


Figure 120: Performance on Best DeepAR Fit "feltcraft 6 flower friends"

6.7.2 Predicting Product Sales With K-Means Clustering

Although traditionally DeepAR does not need clustering, the performance of our model on the raw data implies that the data might do better with training different models to different subsections of our data. In order to achieve this, we utilized the K-Means clusters defined earlier in the paper. This DeepAR model was tested on cluster 0 (14 Items), cluster 6 (10 Items), and cluster 12 (6 Items). Although there was some improvement from the previous model, the fact that the MAPE and sMAPE remained relatively high (over 50%) coupled with the fact that clustering is not a necessity for DeepAR stopped further exploration. The results for the clusters are listed below.

Cluster	Mean MAPE	Mean sMAPE	MAPE w/o Outliers	sMAPE w/o Outliers
1	91.54	116.83	77.51	120.29
6	71.28	83.33	71.28	83.33
12	68.56	87.2	68.56	87.2

Table 17: Comparison of MAPE and sMAPE across Clusters

From the results, it seems that DeepAR performs better when less items are grouped together. With this, we can say that the other clustering methods would not be beneficial as they contain fewer or similar clusters meaning that each cluster would have more items (therefore hindering performance of DeepAR).

For cluster 1, DeepAR performed best on item 10 or "red toadstool led night light". For cluster 6, DeepAR performed best on item 9 or "wooden picture frame white finish". For cluster 12, DeepAR performed best on item 1 or "door mat fancy font home sweet home".

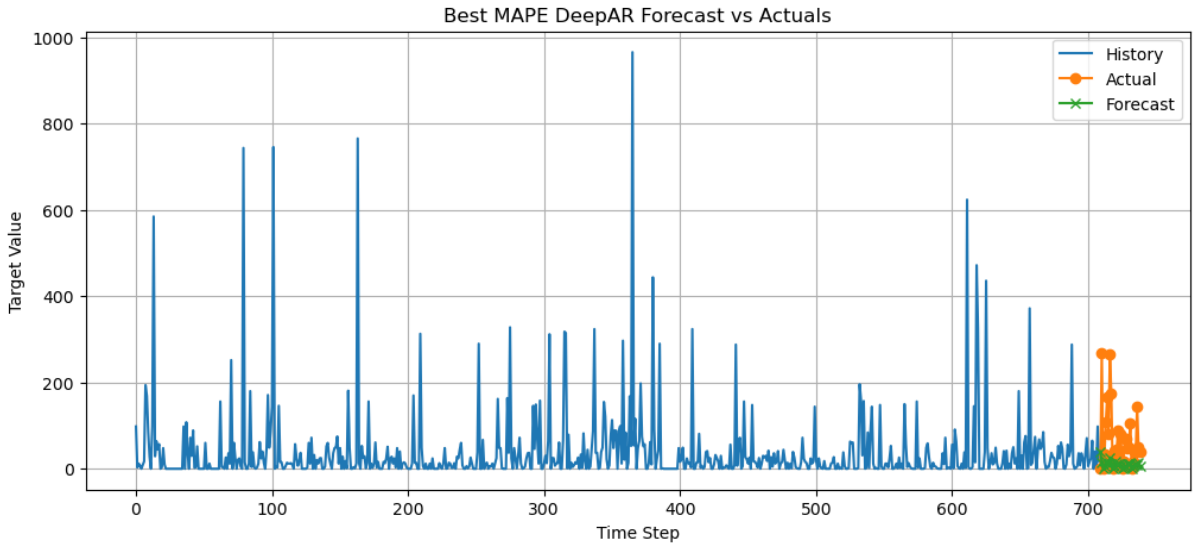


Figure 121: Performance on Best DeepAR Fit for Cluster 1 "red toadstool led night light"

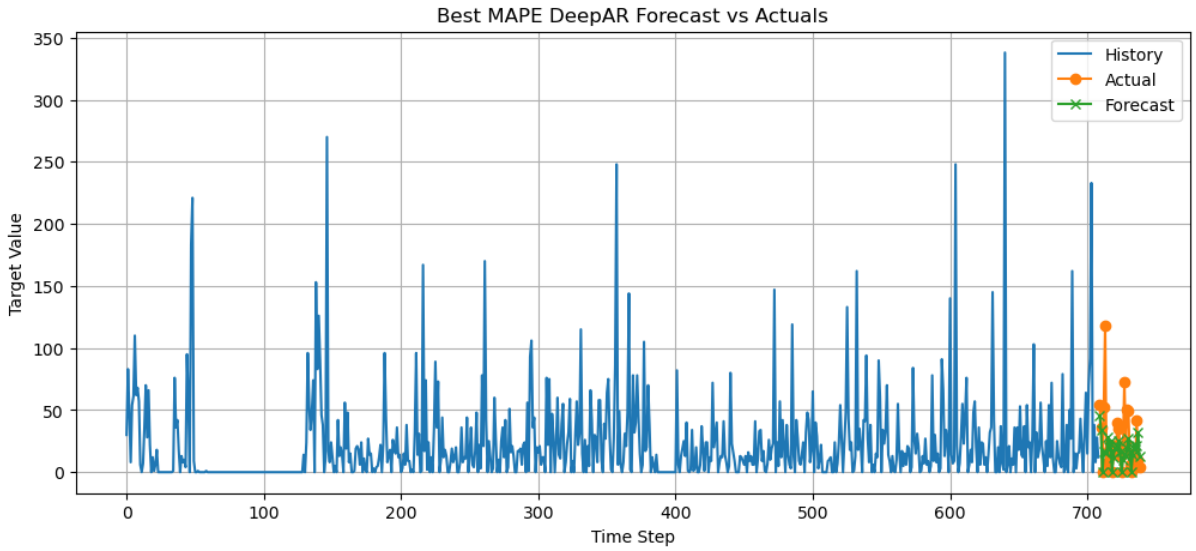


Figure 122: Performance on Best DeepAR Fit for Cluster 6 "wooden picture frame white finish"

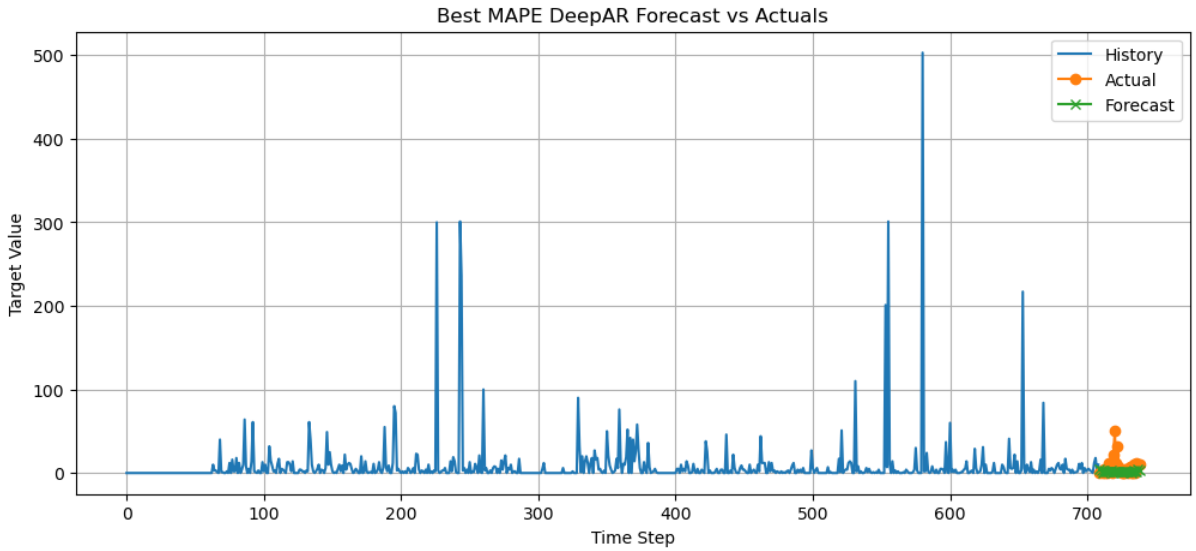


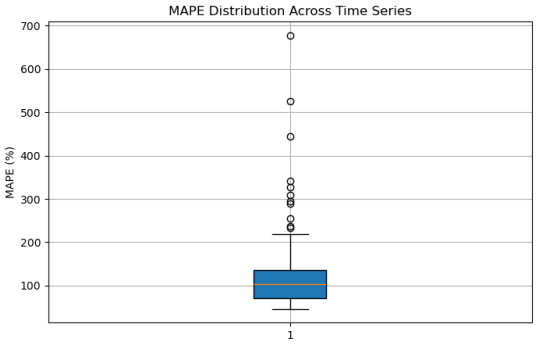
Figure 123: Performance on Best DeepAR Fit for Cluster 12 "door mat fancy font home sweet home"

6.7.3 Predicting Product Sales With Explanatory Variables

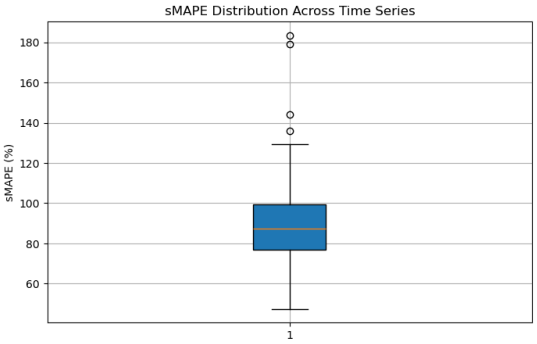
Next, we attempted to add explanatory variables to our model as a way of helping DeepAR understand the underlying patterns behind the data. The previously mentioned explanatory variables were utilized except that this time the training and testing was split into three sections: Dynamic Real Variables, Binary Variables, and All Variables.

DeepAR generally only takes continuous, real-valued, time-varying features as covariates which is why the model was first only tested on the listed Dynamic Real Variables: "gift_ideas", "cpi", "cci", "interest_rate", "unemployment_rate", "chained-volume-percentage-change-3-months-on-same-period-a-year-earlier", "chained-volume-percentage-change-on-previous-month", "chained-volume-percentage-change-on-same-month-a-year-earlier", "current-prices-percentage-change-on-previous-month", "current-prices-percentage-change-on-same-month-a-year-earlier"

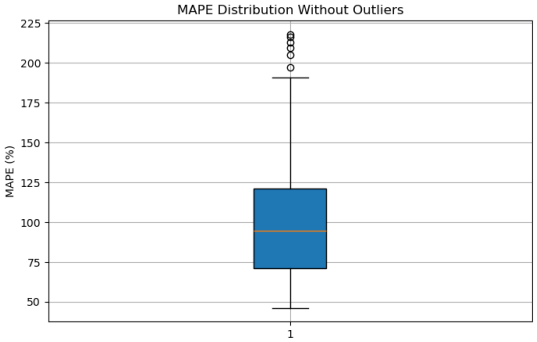
The results yield a mean MAPE of 94.62%, a mean sMAPE of 99.72%, a mean MAPE of 102.96% without outliers, and a mean sMAPE of 87.77% without outliers. Item 74, "pack of 72 retro spot cake cases", performed the best.



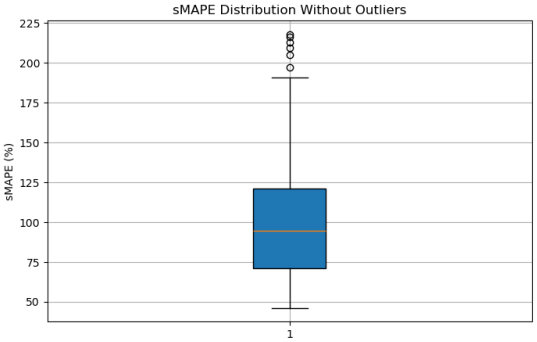
(a) Distributions of MAPE for different time series



(b) Distributions of sMAPE for different time series



(c) Distributions of MAPE for different time series after outlier removal



(d) Distributions of sMAPE for different time series after outlier removal

Figure 124: MAPE and sMAPE distribution with DeepAR predictions with Dynamic Explanatory Variables

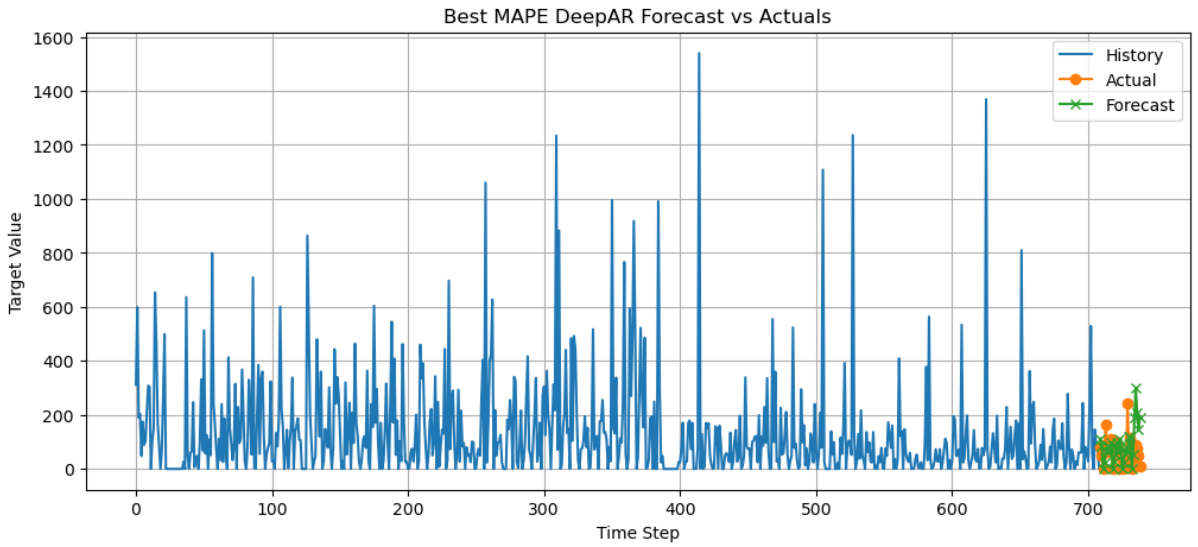
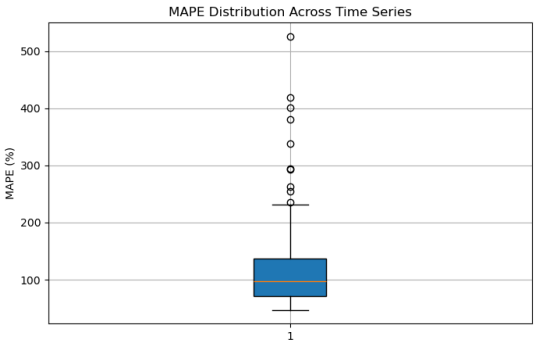


Figure 125: Best Performing Item Results ""pack of 72 retro spot cake cases""

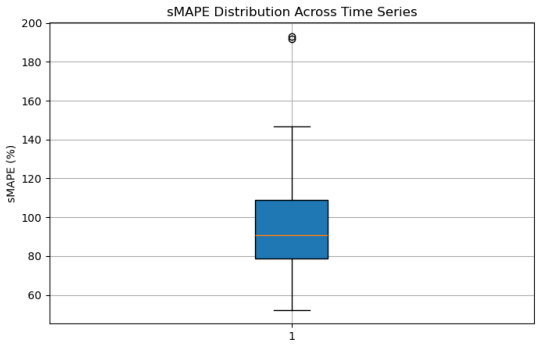
Next, to test their impact, we analyzed binary predictors on their own. The following categorical variables were utilized: "is_black_friday", "is_cyber_monday", "is_Monday", "is_holiday", "is_Tuesday",

"is_Wednesday", "is_Thursday", "is_Friday", "is_Saturday", "is_Sunday". The values of these variables were converted into floats and added as part of the dynamic real features part of the encoding.

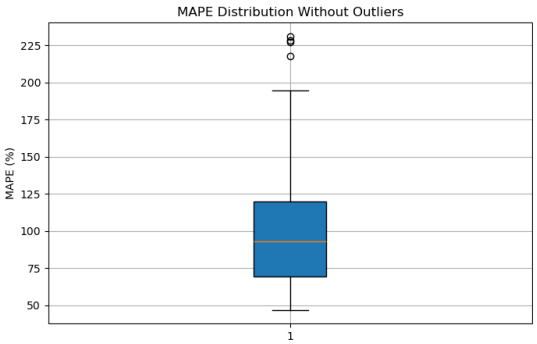
This resulted in a mean MAPE of 122.66%, a mean sMAPE of 90.10%, a mean MAPE of 89.07% without outliers and a mean sMAPE of 93.33% without outliers. Item 74 once again had the best predictions.



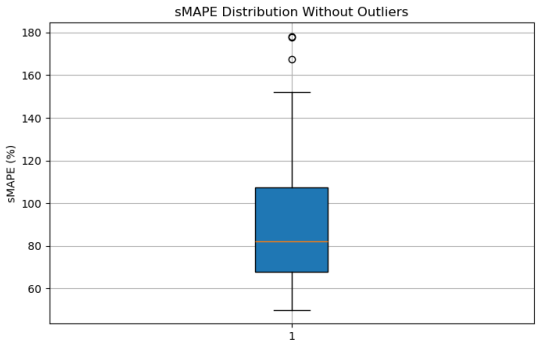
(a) Distributions of MAPE for different time series



(b) Distributions of sMAPE for different time series



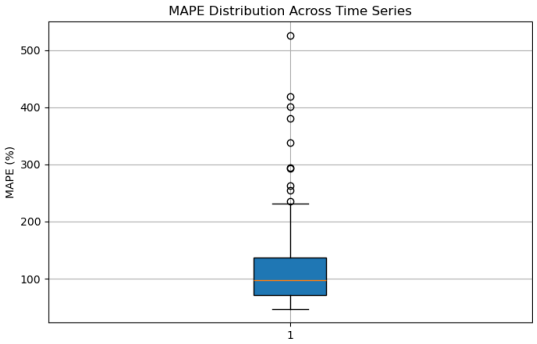
(c) Distributions of MAPE for different time series after outlier removal



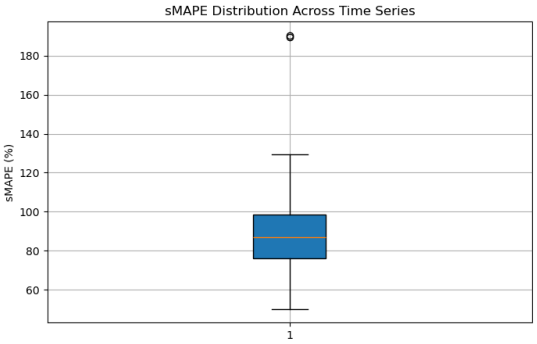
(d) Distributions of sMAPE for different time series after outlier removal

Figure 126: MAPE and sMAPE distribution with DeepAR predictions with Binary Explanatory Variables

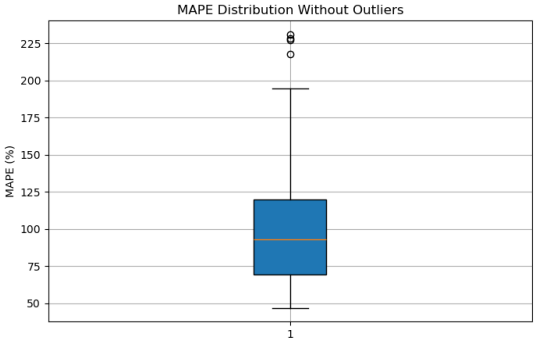
Finally, we combine both dynamic and explanatory variables to analyze their effects on the model. The results give us a mean MAPE of 120.09%, a mean sMAPE of 89.11%, a mean MAPE of 103.26% without outliers, and a mean sMAPE of 87.66% without outliers. Once again item 74 performed the best



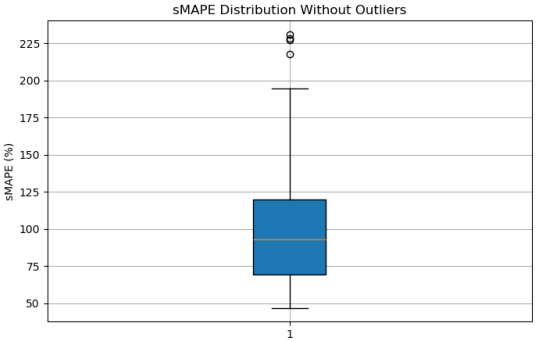
(a) Distributions of MAPE for different time series



(b) Distributions of sMAPE for different time series



(c) Distributions of MAPE for different time series after outlier removal

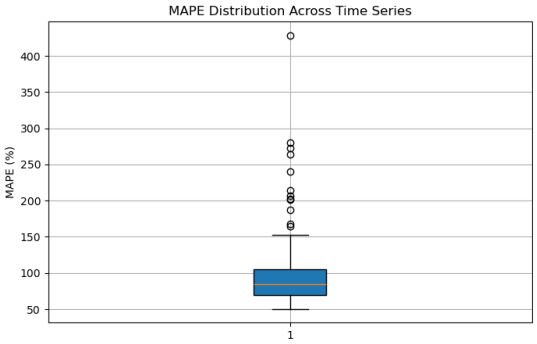


(d) Distributions of sMAPE for different time series after outlier removal

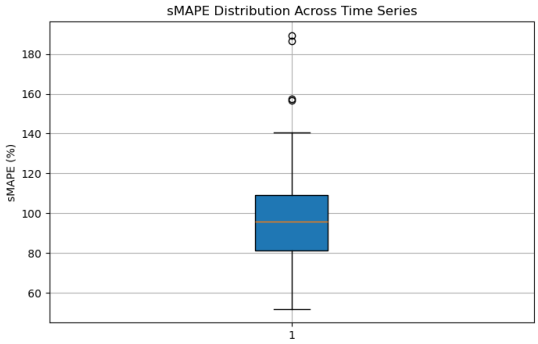
Figure 127: MAPE and sMAPE distribution with DeepAR predictions with Binary Explanatory Variables

6.7.4 Predicting Product Sales With Lag Variables

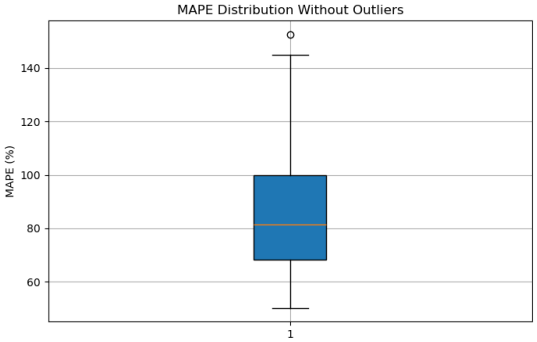
Lastly, to diversify each product's explanatory variables, we explored adding lag to each explanatory item. 1 Day, 7 Day, and 14 Day Lag were added as variables. NaN values from the lack of early data were filled with 0s. Once again results are similar to previous attempts with a Mean MAPE of 97.95%, mean sMAPE of 97.66%, a mean MAPE of 85.11% without outliers, and a mean sMAPE of 95.48% without outliers. Once again, item 74 was the best performing item in this model.



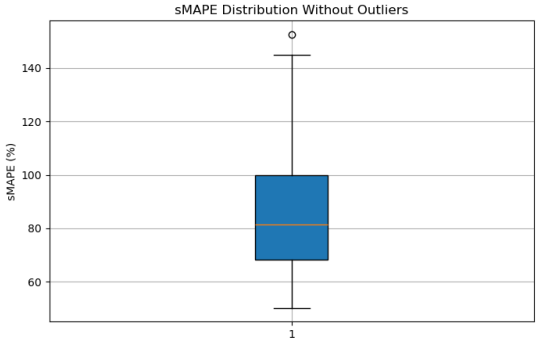
(a) Distributions of MAPE for different time series



(b) Distributions of sMAPE for different time series



(c) Distributions of MAPE for different time series after outlier removal



(d) Distributions of sMAPE for different time series after outlier removal

Figure 128: MAPE and sMAPE distribution with DeepAR predictions with Binary Explanatory Variables

6.7.5 Method Comparison

When looking at the three methods we explored in full for DeepAR, the results are below:

Method	Mean MAPE	Mean sMAPE	MAPE w/o Outliers	sMAPE w/o Outliers
Regular Data	107.42	92.51	92.96	89.81
Exploratory Dynamic	94.62	99.72	102.96	87.77
Exploratory Binary	122.66	90.1	89.07	93.33
Exploratory Data	120.09	89.11	103.26	87.66
Lag Data	97.95	97.66	85.11	95.48

Table 18: Comparison of MAPE and sMAPE across Different Methods

Overall, it seems that the addition of explanatory variables seems to be rather beneficial but there seems to be not enough improvement to make DeepAR a "good" model to predict this dataset. This could be because of the fact that clustering might be needed or due to the large range of sales or due to DeepAR's limitations (for example when adding explanatory variables, since the explanatory variables were joined by date, all variables received the same explanatory patterns).

7 Discussion

7.1 Extending on Previous Group

7.1.1 Clustering Method

Compared to the previous team's approach, which relied solely on a basic K-Means implementation with a fixed k=10, our clustering methodology introduced several key improvements that led to significantly better results. Most notably, we replaced their use of BERT for generating product embeddings with Google's Gemini 2.5 Pro, which offers a much larger context window and improved semantic understanding. This allowed us to capture richer and more accurate representations of product descriptions. We also implemented a more thorough preprocessing pipeline—removing noise such as colors, numbers, punctuation, and stop words, and applying lemmatization—to ensure consistency and clarity in the text inputs. Beyond K-Means, we explored a broader range of clustering algorithms, including DBSCAN, HDBSCAN, and Spectral Clustering, each evaluated with both PCA and UMAP dimensionality reduction. We tuned hyperparameters carefully and selected the optimal number of clusters using silhouette scores and elbow plots. As a result, our best K-Means model achieved a silhouette score of 0.5209, a significant improvement over the previous group's 0.0336, while our top-performing DBSCAN model reached 0.884 (excluding noise), revealing much stronger cluster structure and interpretability.

7.1.2 Explanatory Variables

We significantly enhanced the predictive capability of our time-series models by incorporating a broader and more diverse set of explanatory variables beyond the basic time-based features used by the previous group. While the prior approach included standard temporal indicators such as day of the week, holiday flags, and lag/rolling statistics, our model extended this foundation with additional economic, behavioral, and product-specific signals. We integrated macroeconomic indicators like CPI, CCI, unemployment rate, and interest rates to capture broader market dynamics. We also included Google search trends to reflect shifts in consumer intent and average product prices to model demand elasticity. These enhancements provided our models with richer context, allowing them to better distinguish between genuine trends, seasonal spikes, and promotional effects. As a result, our forecasts are more sensitive to real-world factors that influence purchasing behavior, leading to more accurate and actionable predictions.

7.1.3 Model Selection

The previous team used ARIMA, SARIMA, Prophet, BiLSTM, along with specific model choices within each that differ from our implementation.

We began by extending and refining traditional Prophet and LSTM models to establish a strong forecasting baseline while fitting the context of our goal. Building on the foundations of Prophet, LSTM, and foundational models like GLM, Decision Tree, and XGBoost that provided more insights into the data, we explored more advanced deep learning models such as TFT and DeepAR, which are designed to handle multi-series forecasting and capture long-term dependencies. These models allowed us to assess how well newer architectures performed relative to simpler baselines, especially in the presence of clustered or grouped time series data.

When comparing our LSTM to their BiLSTM model, the previous group achieved a MAPE score (37.19%), likely due to their use of a Bidirectional LSTM, which leverages both past and future context to make predictions. While this can improve performance in some tasks, we believed it was not ideal for forecasting, particularly at inference time when future values are unknown. They also used Isolation Forest to remove outliers, replacing them with the average of neighboring values. In contrast, we applied a moving average to smooth the data, as we considered that certain spikes could be tied to holidays or other important factors captured by our explanatory variables that we did not want to entirely remove. Furthermore, our LSTM architecture incorporated Batch Normalization for training stability and Dropout to mitigate overfitting, especially given the limited size of our dataset. In our work, LSTM was mainly used as a mediator between multi-series and single-product forecasting, enabling us to better understand how well the model performs when learning generalizable patterns across clusters versus focusing on individual product-level signals.

TFT and DeepAR were entirely new models that the previous team did not explore. We believed these were strong additions to our analysis, as both are specifically designed for multi-series forecasting and can handle complex temporal patterns that traditional models often miss. DeepAR is well-suited for probabilistic forecasting across grouped time series and TFT introduces attention mechanisms and variable selection to dynamically focus on the most relevant inputs at each time step. By incorporating these models, we aimed to benchmark the performance of SoTA architectures against more conventional approaches like LSTM and Prophet.

7.1.4 Model Evaluation

The previous team used Mean Absolute Percentage Error (MAPE) to evaluate their model's performance. In our case, we chose to use symmetric MAPE (sMAPE) instead, as we believed it was more appropriate for our forecasting scenario. While MAPE can be heavily skewed when actual values are close to zero, sMAPE provides a more balanced view by normalizing the absolute error by the average of the actual and predicted values. This symmetry makes it more robust when dealing with a wide range of product sales volumes, especially in our dataset where some products exhibited low or highly variable sales. Overall, we switched to sMAPE because it offered a fairer comparison across time series and clusters.

7.2 Limitations and Future Work

While our approach yielded several meaningful insights, there were notable limitations that shaped the scope and outcomes of this project. The most prominent challenge was data sparsity—many products had limited or inconsistent historical records, making it difficult to train reliable models or form consistent clusters. Additionally, although we experimented with a range of models and clustering techniques, we were unable to exhaust the space of hybrid or transfer learning approaches, nor could we fully explore semi-supervised or zero-shot methods for cold-start scenarios. Lastly, our evaluation focused primarily on predictive accuracy metrics such as sMAPE, but future work could benefit from incorporating calibration and uncertainty quantification to better assess model reliability.

Table 19: Comparison of LLM APIs

Provider	Model	Max Context Window	Version Date
OpenAI	GPT-4.1	1,047,576 tokens	April 2025
Anthropic	Claude 3.7 Sonnet	200,000 tokens	March 2025
Google	Gemini 2.5 Pro (Preview)	1,048,576 tokens	March 2025

For future work, we aim to expand our clustering experiments by exploring a wider range of embedding models and clustering techniques. While Gemini 2.5 Pro provided high-quality embeddings that improved clustering performance, we plan to compare and combine embeddings from other leading large language models, such as OpenAI's GPT-4 and Anthropic's Claude, to evaluate how different architectures and training data influence cluster quality. In addition, we are interested in experimenting with ensemble clustering approaches, which aggregate results from multiple algorithms to create more robust and consensus-based groupings. Techniques like clustering aggregation, co-association matrices, or meta-clustering could help balance the strengths and weaknesses of individual methods. We also see potential in developing hybrid approaches that incorporate both semantic similarity (from text embeddings) and behavioral similarity (e.g., sales patterns) to form more contextually aware clusters. These enhancements could lead to more nuanced and actionable segmentations, especially in applications like personalized marketing or demand forecasting.

Additionally, we aim to further explore methods for addressing sparse and imbalanced data, which posed a significant challenge in this project. While we looked into clustering techniques—such as HDBSCAN or Spectral Clustering—to better group products that exhibit similar temporal or pricing behavior, the lack of clean data (196 products) make it challenging to create meaningful clusters that have similar behavioral patterns. By clustering effectively and with more data, we can transfer knowledge from data-

rich products to those with limited historical records, thereby improving both training and inference outcomes.

We recognize that certain models are inherently more robust to sparse data and cold-start problems:

- DeepAR handles cold-start scenarios relatively well by learning global patterns across multiple time series, making it capable of generalizing to new series with limited data.
- TFT incorporates static covariates and attention mechanisms, allowing it to focus on the most relevant features—even when historical data is incomplete.
- Cluster-based LSTMs may enable grouped forecasting by training on clusters rather than individual series, allowing sparse products to benefit from shared structure.

By continuing to refine clustering and selecting models that are designed for variability and missingness, we hope to build more resilient forecasting pipelines that can handle real-world data constraints more effectively, and provide meaningful insights regarding product sales forecasting.

8 Conclusion

In this project, we explored the complex problem of forecasting online product sales using a comprehensive pipeline that integrates clustering, feature engineering, and both traditional and modern time series models. By evaluating a range of forecasting methods (GLM, Decision Trees, XGBoost, Facebook Prophet, LSTM, TFT, and DeepAR) across multiple clustering strategies (K-Means, DBSCAN, HDBSCAN, and Spectral), we demonstrated the trade-offs between interpretability, accuracy, and scalability in multi-series retail forecasting.

Our results show the importance of thoughtful preprocessing, clustering, and feature selection. Dimensionality reduction techniques such as UMAP significantly improved clustering performance, particularly for density-based algorithms like DBSCAN, which achieved the highest silhouette score. We also found that lag features and rolling statistics had strong predictive power, often outperforming macroeconomic indicators. While simpler models like GLM and XGBoost offered consistent performance and interpretability, advanced deep learning models such as DeepAR and RNNs showed potential when tuned carefully and paired with the right clustering strategy, although they struggled with generalization and data sparsity in some scenarios.

Overall, this paper highlights that there is no forecasting model that can accomplish all of our goals holistically. Instead, the best approach depends on the structure of the data, availability of features, and objectives such as transparency versus accuracy. By combining large language model-driven clustering with robust forecasting techniques, we developed a flexible framework that can be adapted to a variety of retail environments and pave the way for more personalized and robust product demand forecasting systems.

References

[1] Fahad Radhi Alharbi and Denes Csala. A seasonal autoregressive integrated moving average with exogenous factors (sarimax) forecasting model-based time series approach. *Inventions*, 7(4), 2022.

[2] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[4] Bryan Lim, Stefan Zohren, and Stephen Roberts. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[5] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2019.

[6] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1st edition, 1984.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM, August 2016.

[8] Andrew C. Harvey and Simon Peters. Estimation procedures for structural time series models. *Journal of Forecasting*, 9(2):89–108, 1990.

[9] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for r. Technical Report 6/07, Monash University, Department of Econometrics and Business Statistics, 2007.

[10] Ahmed Elsayed and Yun Zhao. Lstm models for time series forecasting: Challenges and advances. *Applied Intelligence*, 57(2):145–163, 2025.

[11] Datacamp. Lstm in python: Stock market prediction. https://www.datacamp.com/tutorial/lstm-python-stock-market. Accessed April 2025.

[12] Tushar Patil. Next 30 days weather prediction using lstm. https://www.kaggle.com/code/tusharspatil/next30-days-weather-prediction-lstm. Accessed April 2025.

[13] Kasun Bandara, Christoph Bergmeir, and Hansika Hewamalage. Lstm-msnet: Leveraging forecasts on sets of related time series with multiple seasonal patterns. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1586–1599, 2020.

[14] Google AI Blog. Interpretable deep learning for time series forecasting. https://research.google/blog/interpretable-deep-learning-for-time-series-forecasting/, 2021. Accessed April 2025.

[15] Anna Petukhova, James Lee, and Navid Moradi. Clustering texts with large language model embeddings. *arXiv preprint arXiv:2503.01234*, 2025.

[16] Yang Huang, Kevin Lin, and J. Wang. Few-shot clustering with language models. *Proceedings of the NeurIPS Workshop on Self-Supervised Learning*, 2023.

[17] David Wu, Ming Zhang, and Liang Chen. Zero-shot text clustering with gpt-4. *arXiv preprint arXiv:2303.10530*, 2023.

[18] Furkan Polat, Ilaria Tiddi, and Paul Groth. Testing prompt engineering methods for knowledge extraction from text. *Semantic Web Journal*, 2024. Forthcoming.

[19] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. https://ai.meta.com/llama/, 2023. Meta AI.

[20] DeepSeek AI. Deepseek llm: Open-source generalist language model. https://huggingface.co/deepseek-ai, 2023. Accessed April 2025.

[21] OpenAI. Gpt-4.1 technical report. https://openai.com/index/gpt-4-1/, 2025. Accessed April 2025.

[22] Anthropic. Introducing claude 3.7 sonnet. https://www.anthropic.com/news/claude-3-7-sonnet, 2025. Accessed April 2025.

[23] Daqing Chen. Online Retail II. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5CG6D.

[24] Shengjie Lai, Alessandro Sorichetta, Jessica Steele, Corrine W. Ruktanonchai, Alexander D. Cunningham, Grant Rogers, Patrycja Koper, Dorothea Woods, Maksym Bondarenko, Nick W. Ruktanonchai, Weifeng Shi, and Andrew J. Tatem. Global holiday datasets for understanding seasonal human mobility and population dynamics. *Scientific Data*, 9(1):17, 2022.

[25] Hazal G"ultekin. What is silhouette score?, Sep 2023.

[26] Ibm. What is k-means clustering?, Apr 2025.

[27] ZalaRushirajsinh. The elbow method: Finding the optimal number of clusters, Nov 2023.

[28] Sachinsoni. Clustering like a pro: A beginner's guide to dbscan, Dec 2023.

[29] Arize AI. Understanding hdbscan: A deep dive into hierarchical density-based clustering, September 2023. Accessed: 2025-05-09.

[30] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Aug 2007.

[31] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529, 1993.

Appendix

RNN: Cluster-based Training History Graph

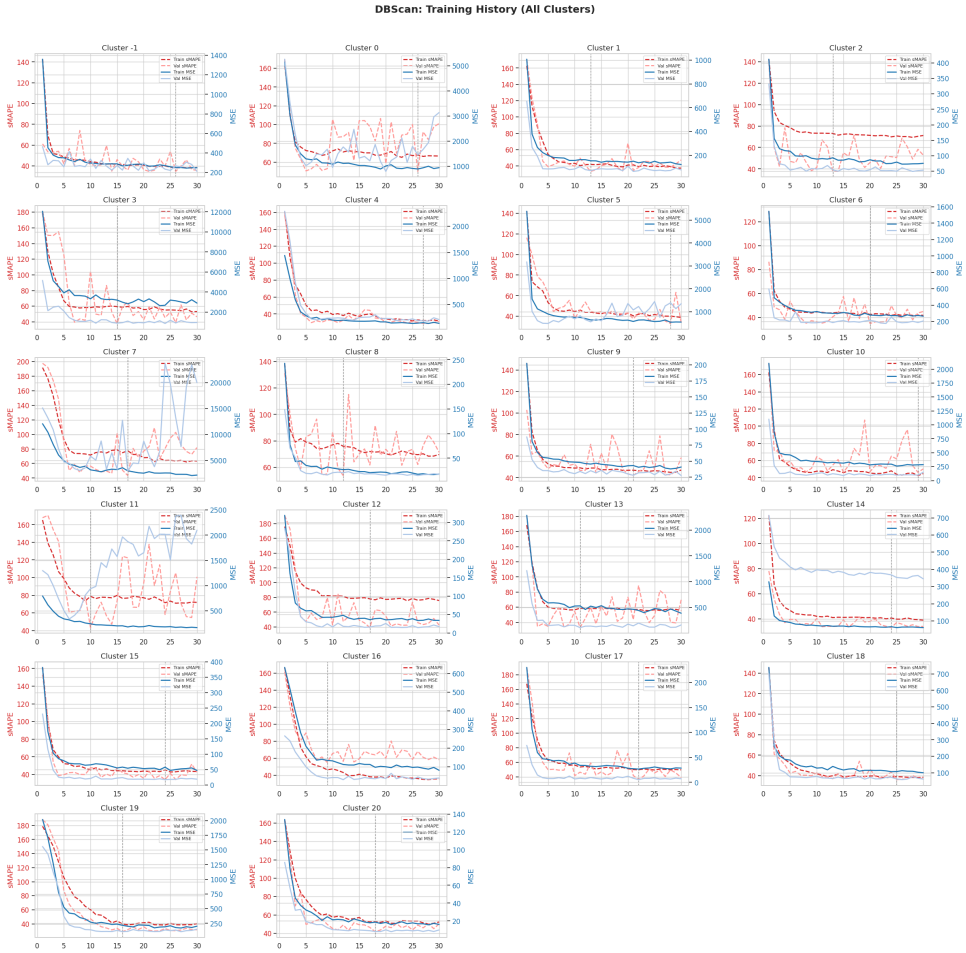


Figure 129: RNN Training for DB-Scan Cluster

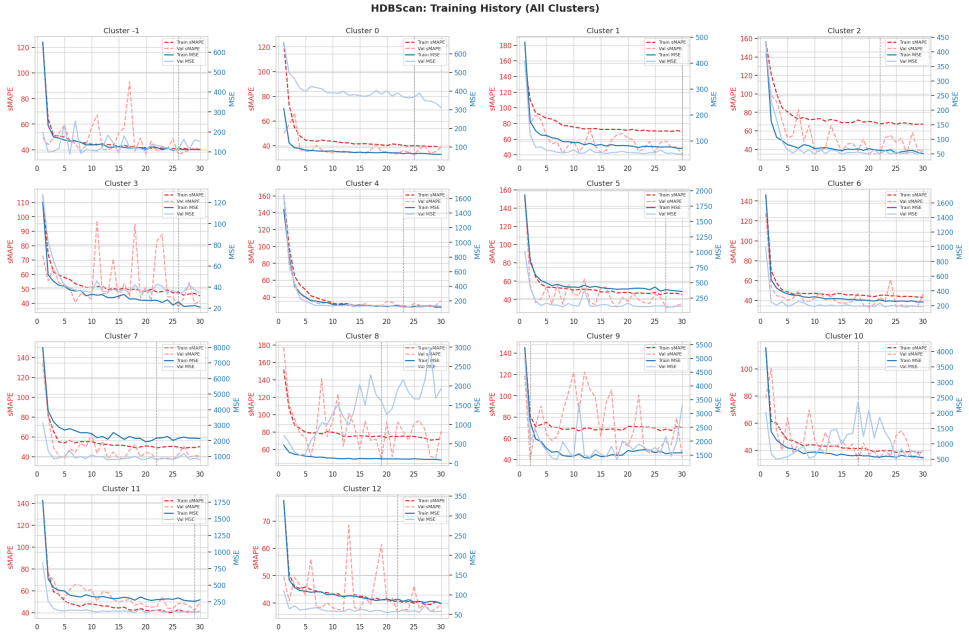


Figure 130: RNN Training for HDBScan Cluster

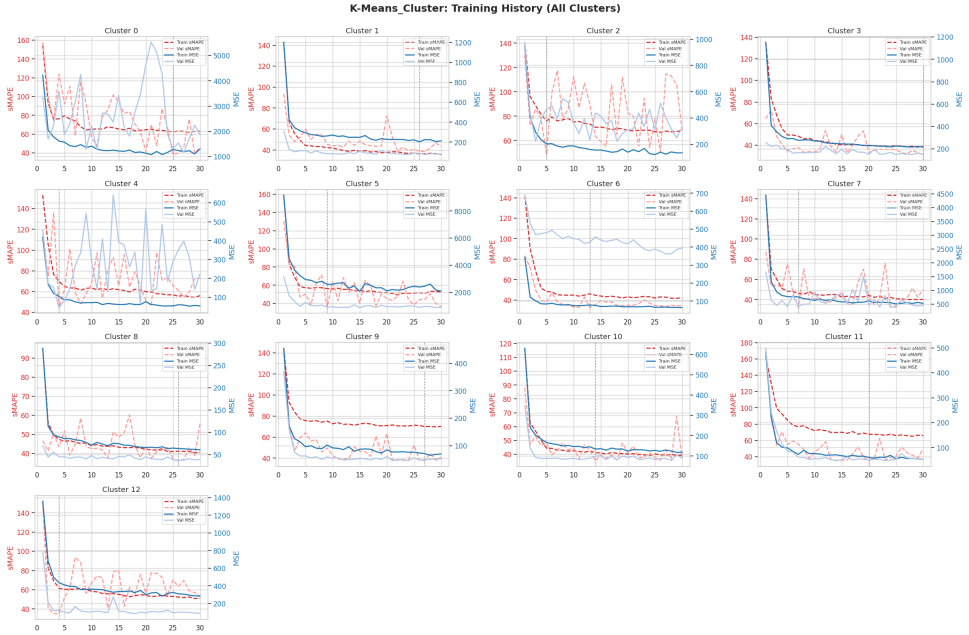


Figure 131: RNN Training for K-Means Cluster

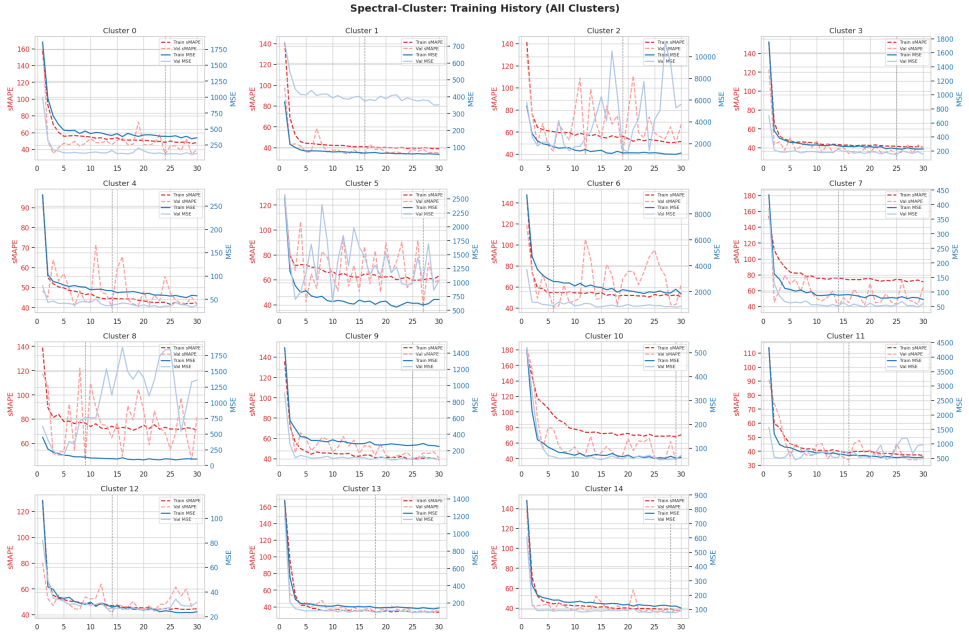


Figure 132: RNN Training for Spectral Cluster

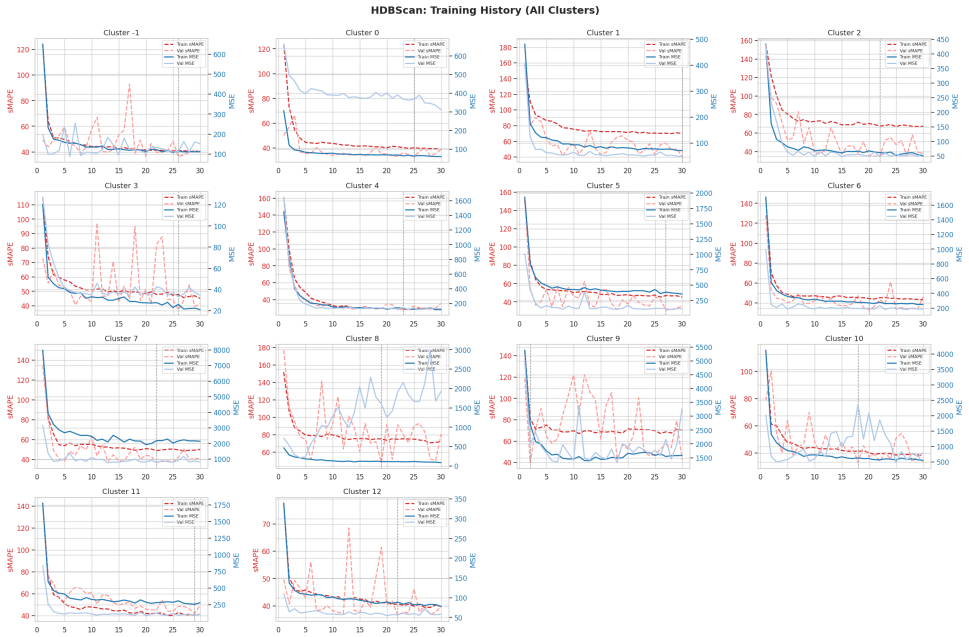


Figure 133: RNN Training for HDB-Scan Cluster

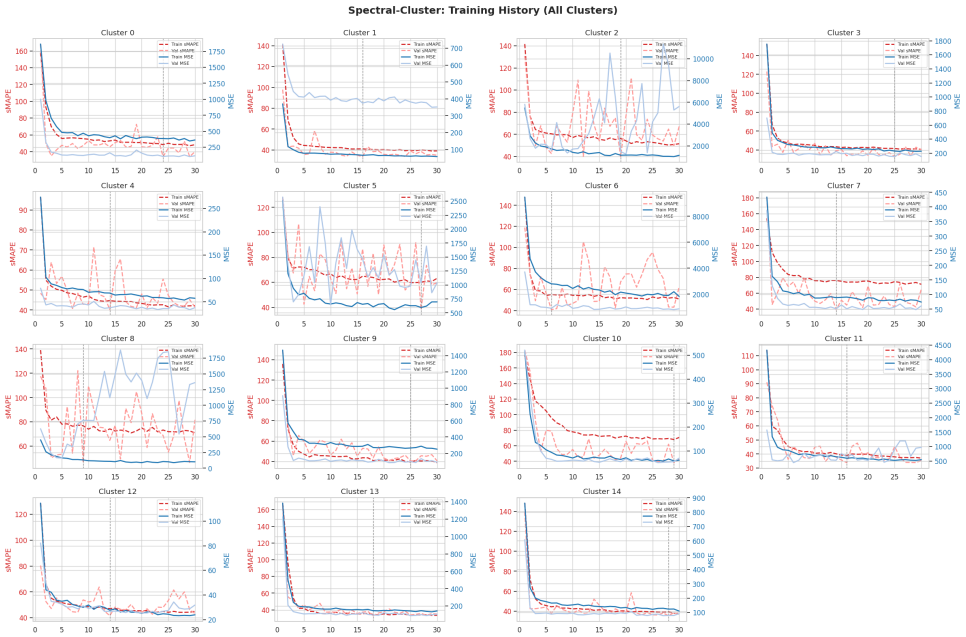


Figure 134: RNN Training for Spectral-Cluster