Data Science Project Final Report

CPSC 5305 01 25FQ Introduction to Data Science

PHU HAN, RIZVAN AHMED RAFSAN

Final Report: Integrated Forecasting for Retail Inventory

Instructor:

Dr. Diala Ezzeddine

Team Members:

1. Phu Han
2. Rizvan Ahmed Rafsan

Problem Statement

Motivation:

Motivated by our experience in e-commerce operations and our shared interest in addressing real-world retail challenges, our team chose to explore challenges businesses face regarding managing inventory and pricing. From what we have seen in multi-warehouse setups, it is easy for stock levels to get out of sync. For example, when a nearby warehouse runs out of an item, companies often need to ship it from farther away, which drives up costs, slows down delivery, and leaves customers less satisfied.

Business Problem:

In large-scale retails with multiple product lines and warehouses, inefficient inventory management and static pricing lead to significant operational waste and lost revenue. Specifically, the businesses struggle with **increased logistics costs** (transferring out-of-stock items), **lost revenue** (sales lost due to out-of-stock items, excessive overstocking of slow-selling products) [1].

---

*The main goal: Replace current decision-making process with a **predictive model of future product demands** to **optimize inventory placement**.*

The Solutions We Aim to Provide:

- **Demand Forecasting:** Specific daily unit sales for every product across all stores for the next few days/weeks/months or next campaigns, sales events.
- **Inventory Optimization:** Based on sales predictions, predicting products are essential to keep in stock (positive demand), and which should be considered for removal from stock (zero demand).

Importance:

This project demonstrates that machine learning can significantly outperform naive averages in retail planning. A more accurate forecast directly translates to the bottom line: it reduces the "buffer" stock required to maintain service levels, lowers storage costs, and ensures high-demand items are available to customers when they need them, ultimately boosting both operational efficiency and customer satisfaction.

Data Sources

M5 Forecasting - Accuracy Dataset

The dataset [2] consists of the following csv files:

- `calendar.csv`: contains daily information such as dates, weekdays, special events, and holidays.
- `sales_train_validation.csv`: historical daily unit sales for each product and store, providing our main training data.
- `sales_train_evaluation.csv:` similar structure to validation data, used for testing or future forecast evaluation.
- `sell_prices.csv`: gives price information for each product, store, and week, capturing the effect of dynamic pricing.

Challenges Associated with the Dataset

- The four files are required to be combined into a single unified DataFrame. This requires a memory-intensive melting operation on the wide-format sales data, followed by multiple, large-scale merges using the primary keys: `item_id`, `store_id`, and `date`. This is computationally complex due to the volume of data.
- The final merged dataset contains an estimated 60 million rows (one row per product, store, and date combination). Running analysis and models on this scale is infeasible on standard hardware. Therefore, an extensive, mandatory preprocessing step will involve aggressive data type optimization (downcasting) for managing memory to reduce memory.
- The 12+ million `NaN` values in `sell_price.csv` will require careful and robust imputation to ensure the price feature is representative (zero-day sales) and usable without introducing bias. The sales data has many zero-sales days for specific products, which must be addressed during feature engineering or through specialized models designed for count data.
- Data cleaning will also involve handling outliers in sales data and extensive feature engineering from the date columns (e.g., creating lag features, rolling/moving statistics, and identifying trends).
- Training baseline models on the full dataset is causing **memory overload** due to its massive size (nearly 60 million rows). This is leading to slow processing, crashes due to RAM overload, and limiting the ability to run even simple models efficiently on standard hardware.
- It takes a lot of time when training the models.

Risks Associated with the Dataset

- Although memory usage was reduced in initial exploration, the dataset still has 59 million rows. This scale remains a significant risk for model training and perhaps hyperparameter tuning on standard platforms like Google Colab without further reducing the number of data points or variables.
- Setting the initial missing sell price values to 0.00 and then replacing them with a global or yearly median in the imputed features might be a usable technique. But it introduces a risk of bias (if a product's price was genuinely a zero-sale price (e.g., a giveaway); the imputation is incorrect.
- Conversely, if a product was truly discontinued, inputting a price is also incorrect. The success of the model will depend on how effectively a newly created absent flag accounts for this imputation.
- The dataset already has a lot of features, and if we use feature engineering, we are adding more features, so that it can make the training even more difficult.
- Doing cross-validation for time-series dataset may have risks like losing patterns in the inherent chronology of the time-series. Splitting the dataset randomly is not correct. (because splitting by time).

Data Processing:

To process this large-scale, complex retail dataset and answer the core questions of demand forecasting, inventory optimization, and. dynamic pricing, we are adopting technologies like:

- Pandas and NumPy: These are the primary tools used to handle the volume and complexity of the data. A major achievement is the use of a custom memory optimization function that applies aggressive data type downcasting (e.g., converting large integers to smaller int16 or int8 types).
- Matplotlib and Seaborn: These libraries are being used to perform initial Exploratory Data Analysis (EDA) and visualize key trends (e.g., sales volume year-over-year and month-over-month), which is crucial for understanding the data's temporal features and identifying anomalies and seasonality.

Calendar data

- Missing Values: The event columns in the calendar dataset initially contained many empty values indicating the absence of an event. These were imputed with a string "No event".
- Data Type Conversion: The date column was explicitly converted to a datetime object, and the event-related columns were converted to categorical data types after filling missing values.

Sales data

- Melting Data: The sales DataFrame, which was in a wide format with daily sales (for 1941 days) as separate columns, was melted into a long format. This transformation created two new columns, one representing each day and sales (representing the sales volume for that day).

Final Merged Data

- The reshaped sales data was merged with the pre-processed calendar information based on their common date identifier. During this merge some records were found to lack corresponding sales data, indicating calendar entries beyond the sales period. These extraneous rows were precisely removed, ensuring the dataset accurately reflected only valid sales records.
- Following this, the pricing information was integrated into the combined sales and calendar data. This step linked unit prices to each item's daily sales record using a combination of store, product, and weekly period identifiers.
- Post-merge, a substantial number of missing values appeared in the unit price column. To address these gaps, an imputation strategy was employed: the last known price for each unique product was propagated forward, and any remaining initial missing prices were filled backward, ensuring a complete pricing record for all sales entries.
- After merging all the datasets the final DataFrame had a total of 22 features including all the calendar, sales, and sell price related features.

Exploratory Data Analysis (EDA)

Sales Distribution

Approximately 68.00% of the data instances showed zero sales for items, indicating a highly sparse sales pattern that requires careful handling. The distribution of non-zero daily sales is heavily skewed towards

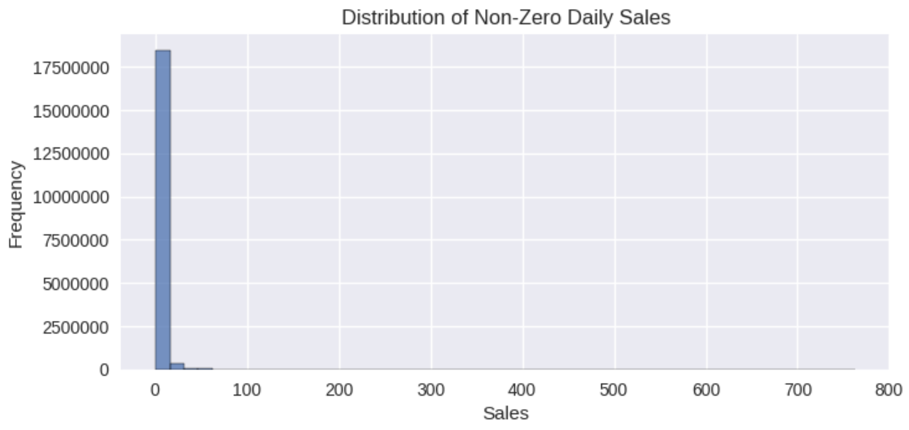


Figure 1: Sales distribution of non-zero sales instances.

lower sales values, meaning most active sales days have modest figures, with only occasional spikes of very high sales. A histogram showing that the vast majority of non-zero sales records fall into a low-volume band (close to zero), with frequencies dropping sharply as sales volume increases, illustrating the high sparsity and positive skew [Figure 1].

Feature Correlation

The correlation heatmap generated for the numerical columns shows the Pearson correlation coefficient between each numeric feature [Figure 2]. The scatter plot for sell price and sales amount revealed no clear linear relationship or strong correlation between these features [Figure 3]. This suggests that price alone might not be the primary driver of sales, or the relationship is more complex.

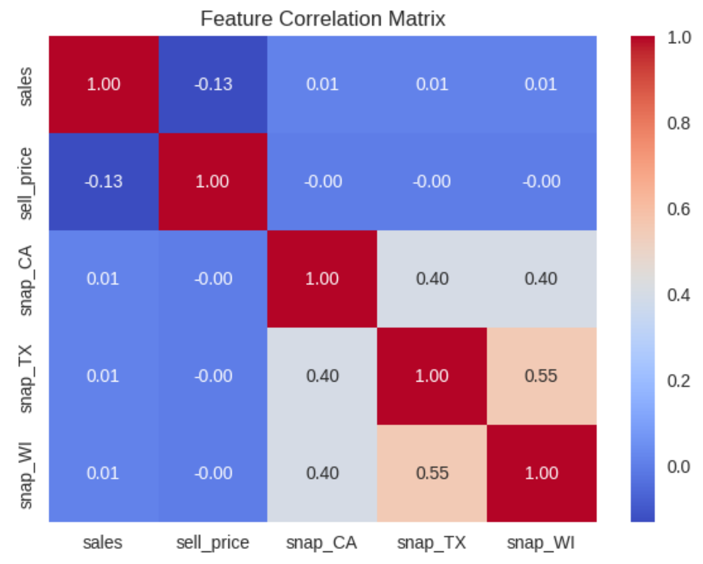


Figure 2: Correlation heat map of the numeric columns

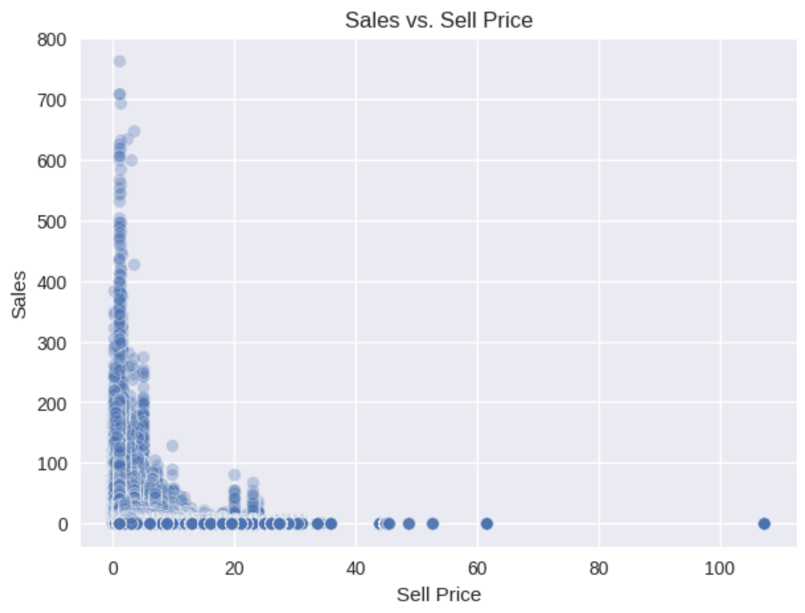


Figure 3: Scatterplot of sales amount and sell price for each item showing no clear linear relationship or correlation between them.

Sales Trends

There is significant variation in total sales across the different stores, indicating that sales volume differs not only across states but also between individual store locations. [Figure 4]. The bar chart ranks stores by their total units sold, clearly shows that some stores (like `CA_3`) consistently achieve much higher total sales than others (like `CA_4`).

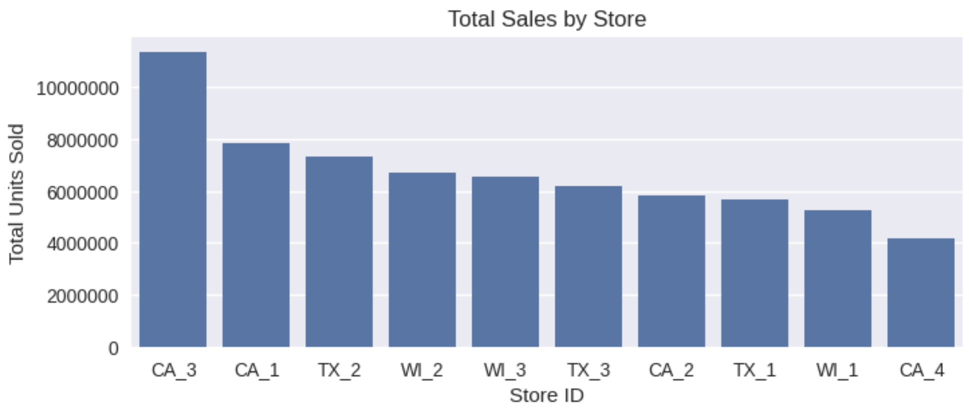


Figure 4: Total sales figure by stores across all the states

Box plots of daily sales per store revealed major differences in both baseline sales (median performance) and volatility (interquartile range). All stores exhibit a large interquartile range, confirming high day-to-day fluctuation, and frequent high-value outliers highlight periodic demand spikes (e.g., promotions, holidays) across the entire network. [Figure 5]. Stores vary greatly in median performance, from high-volume locations like `CA_3` to lower-volume ones like `CA_4`. Critically, all stores show a large interquartile range, confirming high day-to-day sales fluctuation. Furthermore, frequent high-value outliers across the entire network highlight periodic demand spikes (e.g., promotions or holidays), confirming that all locations, regardless of their average performance, experience periods of exceptionally high sales.

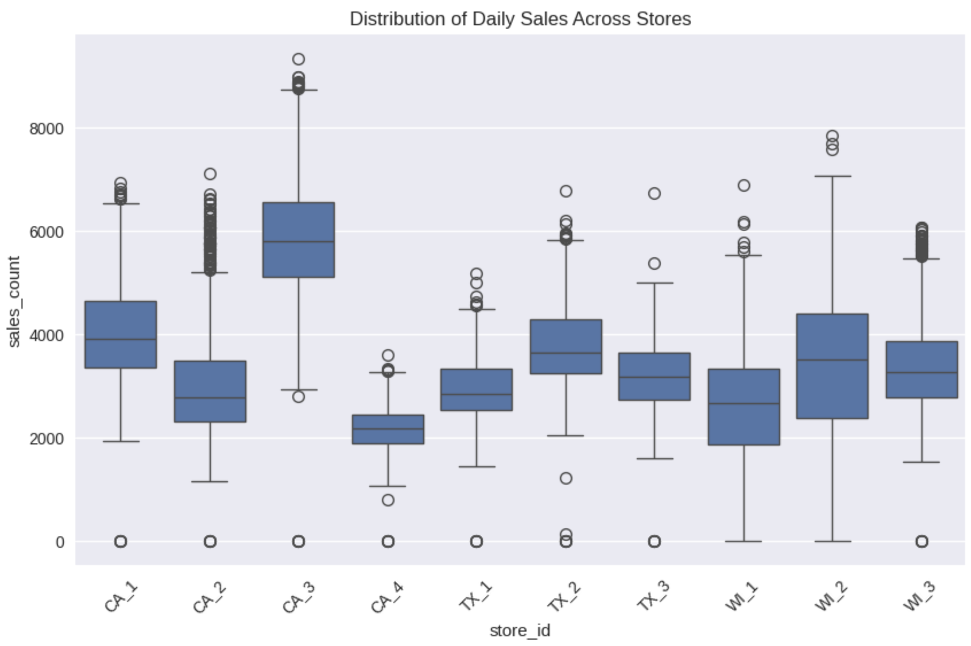


Figure 5: Sales distribution across stores

The analysis of sales trends reveals significant variation and strong temporal patterns across the retail network. Figure 6, showing Total Daily Sales by State, confirms that sales volumes differ geographically, with California (CA) consistently achieving the highest total daily sales. Despite the difference in magnitude, all three states, California, Texas, and Wisconsin, exhibit similar seasonal fluctuations in their daily sales over the observed period (2011 to 2016).

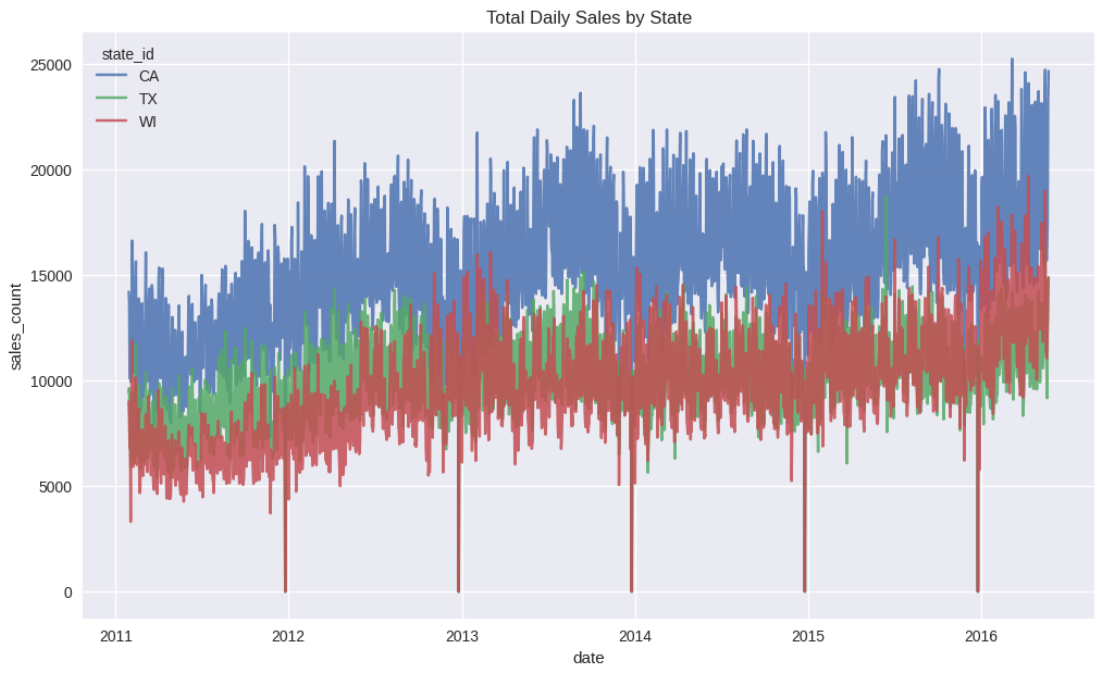


Figure 6: Total daily sales pattern across the stores for each state.

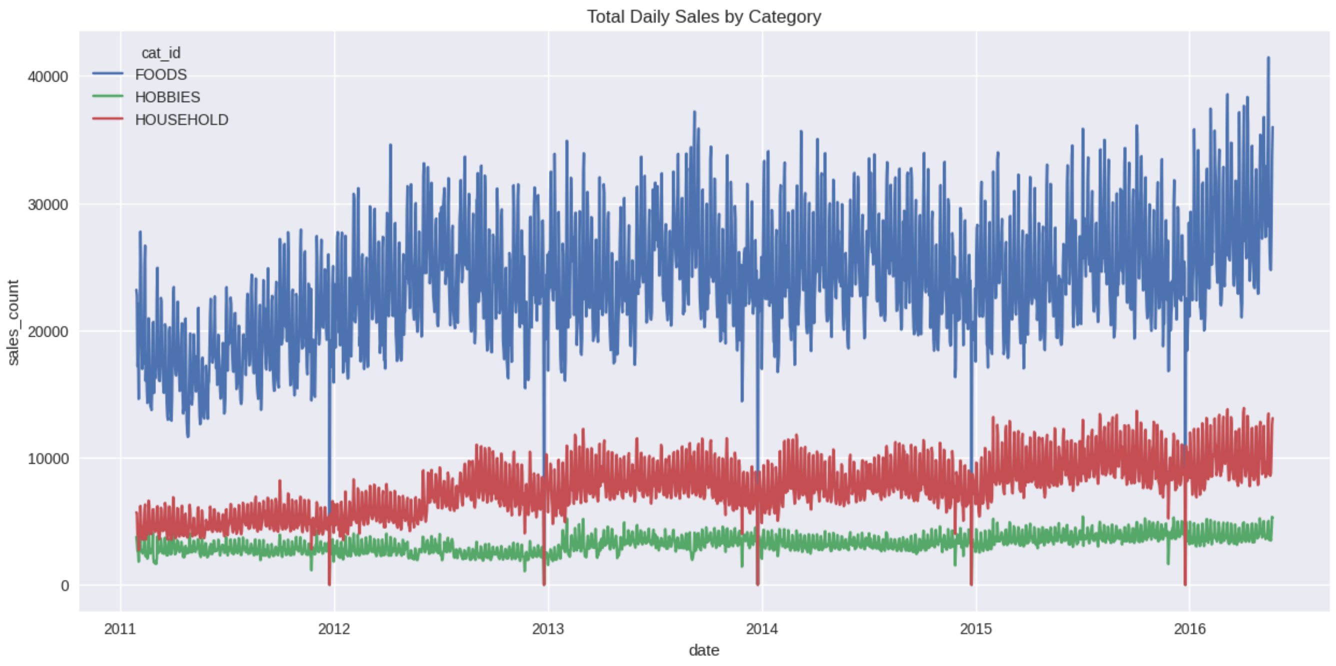


Figure 7: Total daily sales by different item categories.

Concurrently, Figure 7, which breaks down Total Daily Sales by Category, clearly demonstrates that sales are heavily concentrated in the FOODS category, which maintains a vastly higher daily sales volume than both the HOUSEHOLD and HOBBIES categories. Like the state-level data, all product categories display similar underlying seasonal and weekly patterns in their fluctuations. Collectively, these two visualizations emphasize the need for a segmented forecasting strategy that can account for both the distinct high volume of FOODS and CA sales while accurately modeling the universal seasonal and temporal dependencies shared across all states and categories.

Temporal analysis reveals distinct sales patterns across weekly and monthly dimensions [Figure 8]. Weekly sales exhibit clear variation, with relatively stable performance during weekdays (Monday-Thursday) clustering around normalized values of 1.0. Sales increase moderately on Fridays before peaking during weekends (Saturday-Sunday) at approximately 1.35-1.40 times the weekday baseline, reflecting consumer preference for weekend shopping activities.

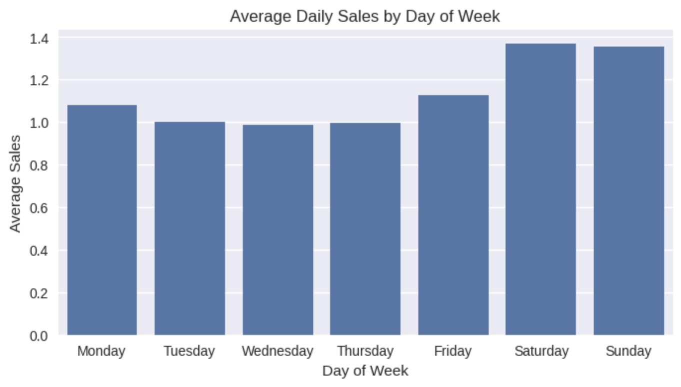


Figure 8: Average daily sales by day of week (normalized), showing weekend peaks at 1.35-1.40 times weekday baseline.

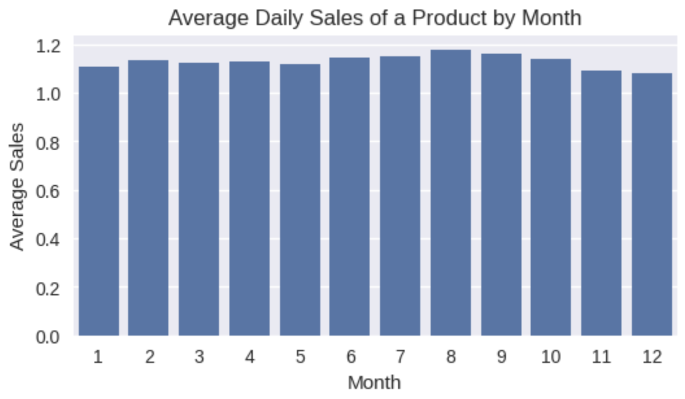


Figure 9: Average daily sales by month (normalized), demonstrating consistent patterns throughout the year with minimal seasonal variation.

Monthly sales distribution demonstrates remarkable consistency throughout the year, with values ranging from 1.0 to 1.15. This uniform pattern indicates minimal seasonal variation, though subtle fluctuations suggest potential influences from promotional periods or holidays. The stability across months contrasts sharply with the pronounced day-of-week effects, suggesting that weekly shopping patterns exert stronger influence than seasonal factors.

Geographic analysis reveals performance heterogeneity across states. California and Wisconsin show comparable sales volumes, both exceeding Texas. Wisconsin marginally outperforms Texas, which consistently records the lowest sales among the three markets, likely reflecting differences in market size, demographics, or retail penetration.

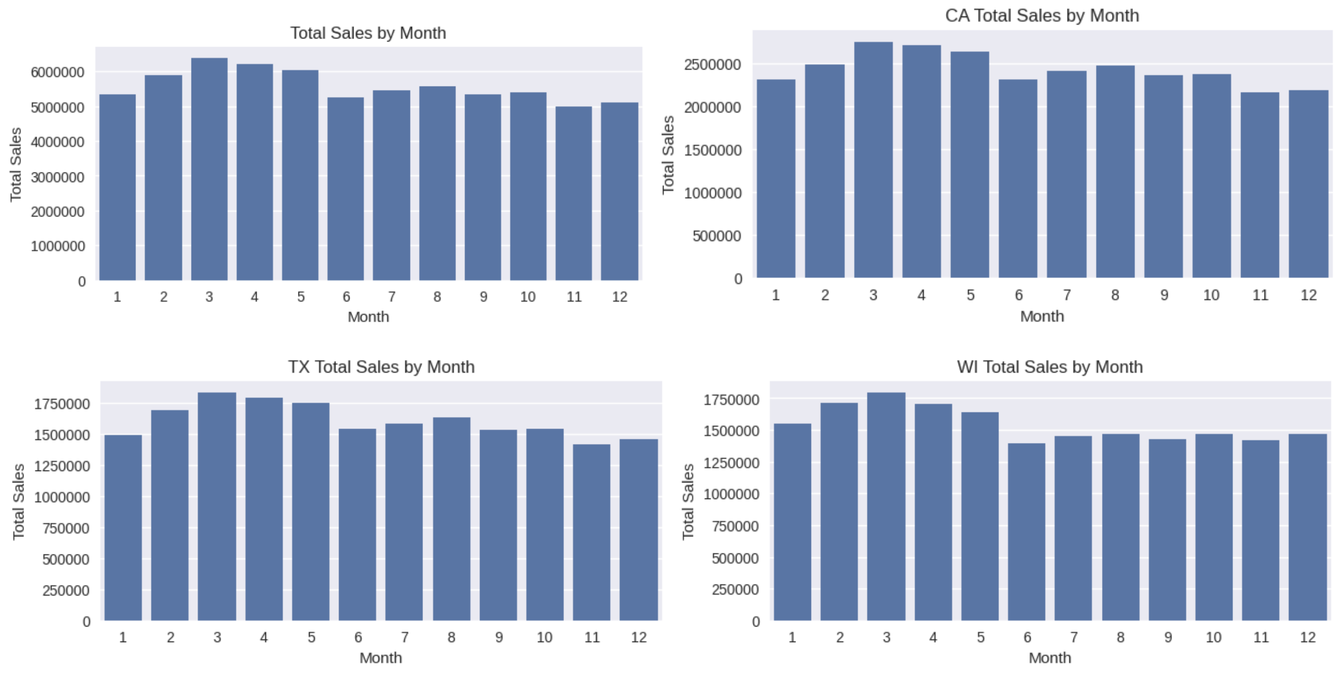


Figure 10: Total monthly sales by state (CA, TX, WI), showing seasonal patterns with early-year peaks and mid-year declines across all markets.

Seasonal Sales Trend

State-level monthly sales patterns reveal both commonalities and distinct regional variations [Figure 10]. All three states exhibit a characteristic seasonal trend with peak sales occurring during the first half of the year (months 1-5), followed by a notable decline in mid-year (months 6-7), and relative stabilization in the latter months (months 8-12).

California demonstrates the highest absolute sales volumes, ranging from approximately 2.0 to 2.6 million units monthly, with peak performance in months 3-5. The state shows the most pronounced mid-year decline, dropping to roughly 2.2 million in month 6 before stabilizing around 2.1-2.4 million through year-end. Texas exhibits intermediate sales volumes between 1.4 and 1.8 million units, following a similar temporal pattern but with less dramatic fluctuations. Peak sales occur in months 3-4, with a gradual decline through mid-year and stabilization around 1.4-1.5 million in the final quarter. Wisconsin records the lowest absolute volumes, ranging from 1.3 to 1.8 million units monthly. The state displays the steepest relative decline from peak months (2-3) to mid-year lows (month 6), with sales dropping approximately 25% before recovering modestly in the latter half of the year.

These patterns suggest common market forces affecting all regions, such as post-holiday consumer behavior in early months and mid-year demand contraction, while absolute differences reflect varying market sizes and economic scales across states.

Event Impact on Sales

Event categorization reveals significant variations in sales impact across different event types. Religious events demonstrate the strongest correlation with sales volumes, approximately 1.8 million, followed by National events at 1.5 million and Cultural events at 1.3 million. Sporting events show moderate impact at 0.6 million, while periods without events register minimal sales activity, highlighting the substantial role of promotional events in driving consumer purchasing behavior.

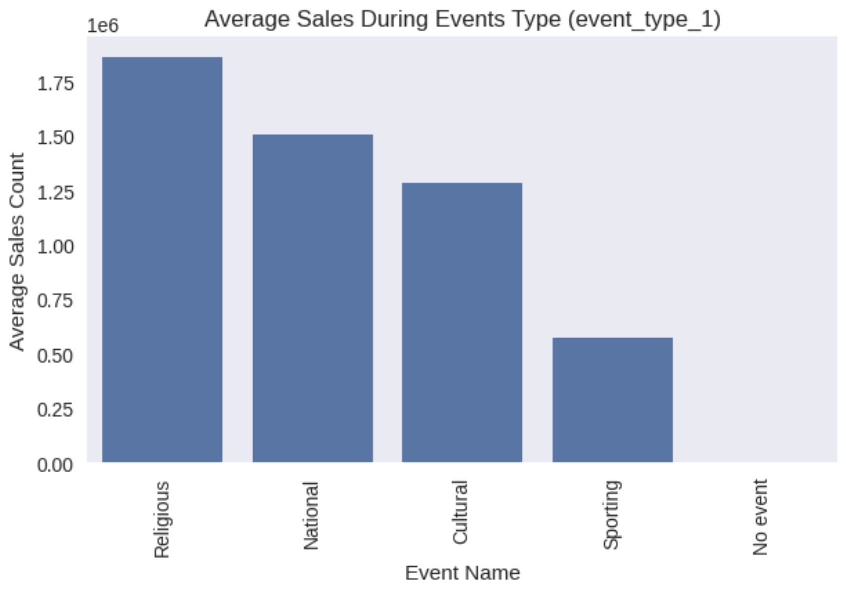


Figure 11: Average sales by event type, showing religious events generating highest sales (1.8 units), followed by National (1.5) and Cultural (1.3) categories. All the values are in millions.

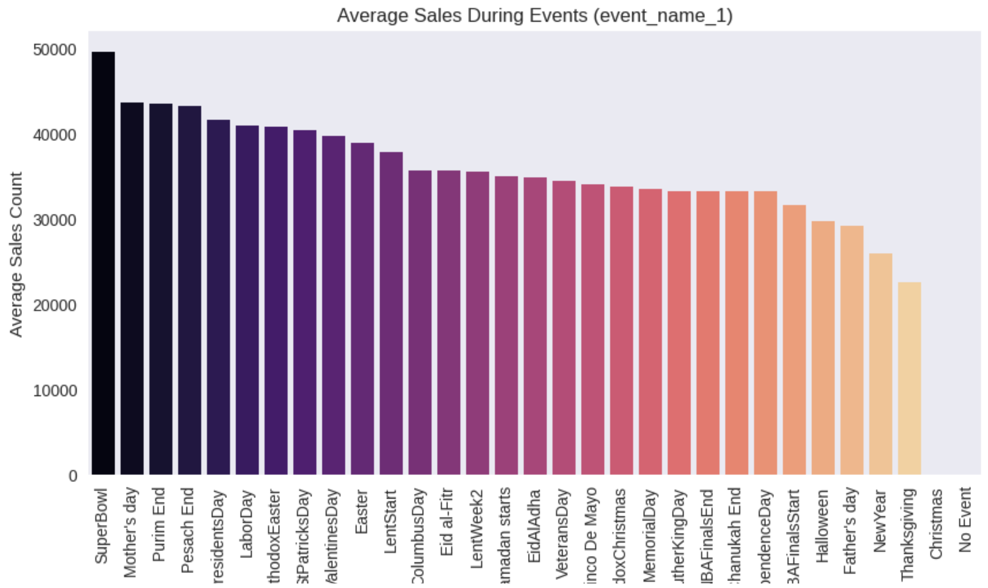


Figure 12: Average sales by specific event name, with SuperBowl and Mother's Day leading at 50,000+ units, while Christmas shows zero sales due to store closures.

Granular analysis of specific event names reveals a pronounced hierarchy in sales effectiveness. SuperBowl generates the highest average sales count at approximately 50,000 units, followed closely by Mother's Day, Palm Sunday, and Easter, each exceeding 40,000 units. Mid-tier events including PresidentsDay, Memorial Day, and various cultural celebrations maintain average sales between 30,000-40,000 units. Lower-performing events such as Halloween, Father's Day, and Thanksgiving generate 25,000-30,000 units, while Christmas records the lowest sales impact at roughly 20,000 units. Notably, Christmas Day registers zero sales across all stores, indicating complete retail closure during this holiday.

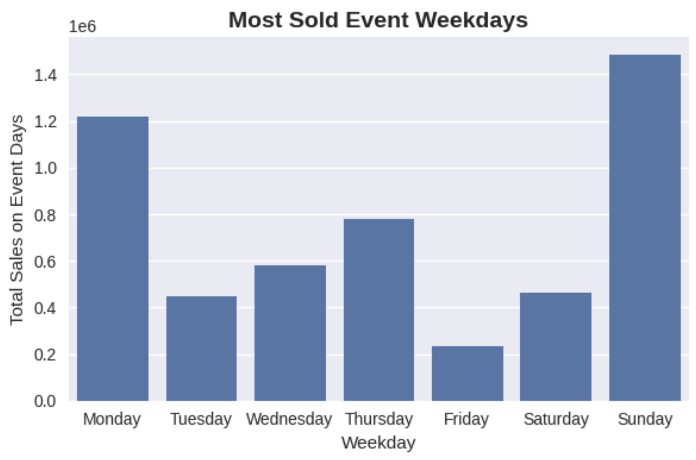


Figure 13: Total sales on event days by weekday, showing Sunday events generating peak sales at 1.5M units, followed by Monday at 1.2M, while mid-week events demonstrate significantly lower sales.

Analysis of event-driven sales across weekdays reveals a pronounced weekend concentration pattern [Figure 13]. Sunday demonstrates the highest event-day sales volume at approximately 1.5 million units, representing a 25% increase over Monday's 1.2 million units. Monday ranks as the second-highest event day, suggesting strategic promotional timing at the week's beginning. Mid-week days (Tuesday through Saturday) exhibit substantially lower event sales, ranging from 0.25 to 0.8 million units, with Friday recording the minimum at approximately 0.25 million units.

This distribution indicates that retailers strategically concentrate promotional events on Sundays to capitalize on peak weekend shopping traffic, while Monday events likely target early-week consumer engagement. The relatively sparse event activity during Tuesday through Saturday suggests either reduced promotional scheduling or lower sales effectiveness during mid-week periods. The dramatic disparity between Sunday (1.5M) and Friday (0.25M) event sales (a six-fold difference) underscores the critical importance of event timing in maximizing promotional impact. These patterns align with general consumer shopping behaviors but demonstrate amplification effects, where weekend events generate disproportionately higher sales than their non-event weekend counterparts.

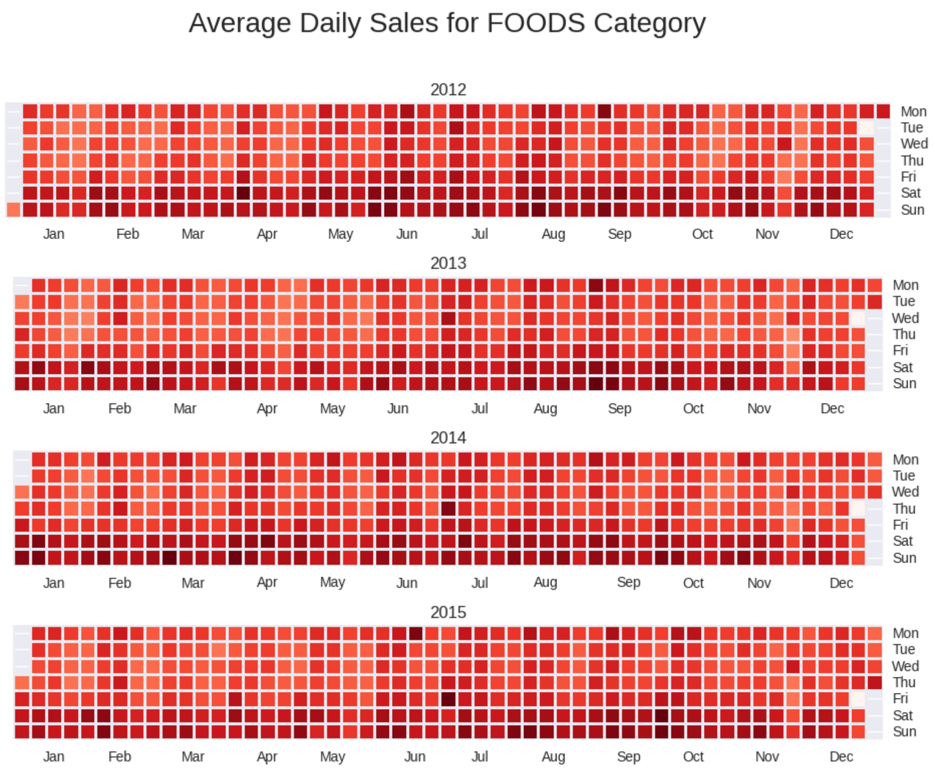


Figure 14: Calendar heatmap of average daily sales for FOODS category (2012-2015), showing strong weekend effects and seasonal peaks during mid-year and holiday periods.

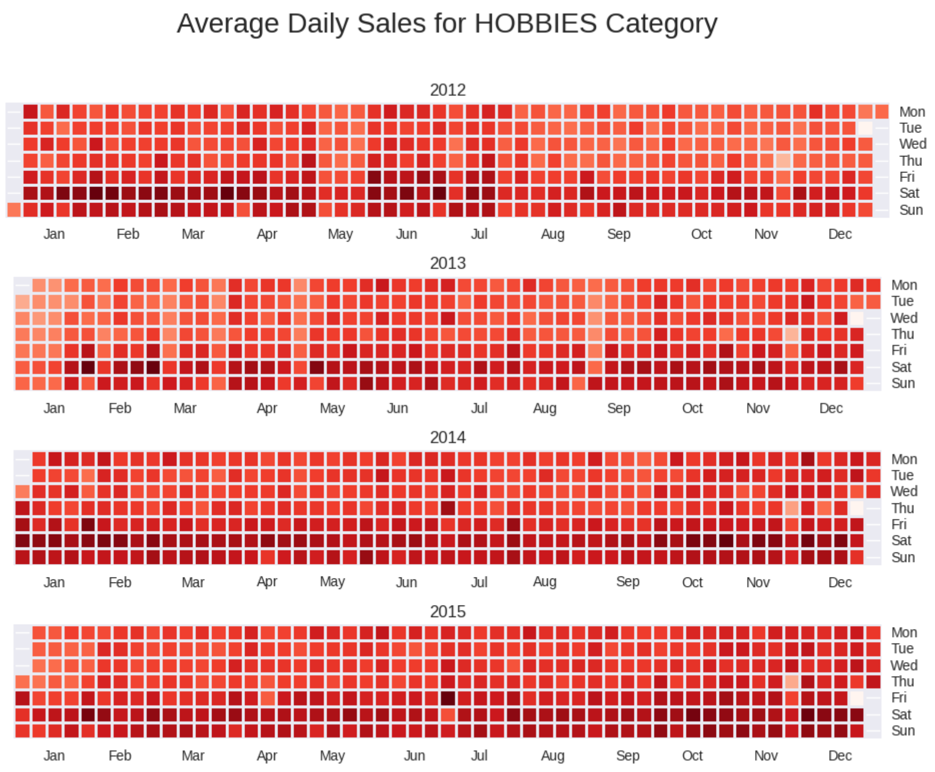


Figure 15: Calendar heatmap of average daily sales for HOBBIES category (2012-2015), displaying pronounced Q4 intensification and moderate weekend patterns.

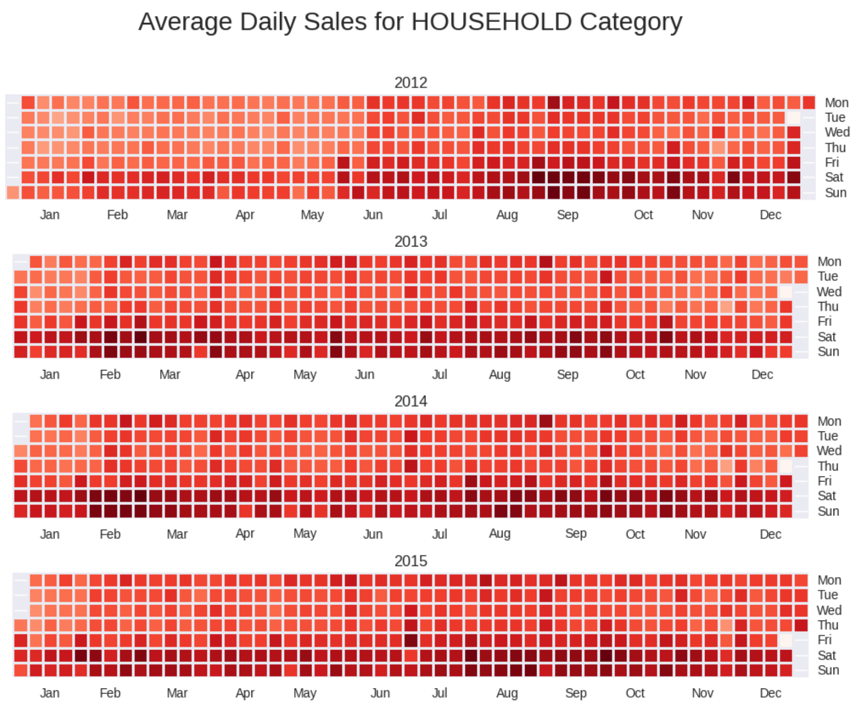


Figure 16: Calendar heatmap of average daily sales for HOUSEHOLD category (2012-2015), demonstrating uniform weekday distribution with gradual annual intensification toward year-end.

Yearly Heatmaps

FOODS Category: Exhibits the most pronounced weekend effect, with consistently darker cells appearing on Saturday-Sunday rows throughout all years. This pattern intensifies during specific periods, particularly mid-year months (May-July) and holiday seasons (November-December), suggesting seasonal demand peaks for food products. The weekend concentration remains stable across all four years, indicating persistent consumer behavior patterns [].

HOBBIES Category: Demonstrates moderate weekend effects with notable seasonal variation. Sales intensity increases substantially during the final quarter (October-December), likely reflecting holiday gift-purchasing behavior. Early-year months (January-March) show relatively lighter intensity, suggesting post-holiday demand contraction. Weekend patterns are present but less pronounced than in FOODS, indicating that hobby-related purchases are more evenly distributed across weekdays.

HOUSEHOLD Category: Displays the most uniform temporal distribution among the three categories, with minimal day-of-week variation. Sales intensity shows gradual intensification throughout each year, with peak periods concentrated in late summer through year-end (August-December). The relatively consistent color patterns across weekdays suggest that household product purchases are driven more by necessity than discretionary weekend shopping behavior.

All three categories exhibit year-over-year consistency in their temporal patterns, demonstrating stable seasonal and weekly cycles. The December period shows universal sales intensification across categories, though Christmas Day consistently appears as a white cell (zero sales) due to store closures.

EDA Summary

- **Data Sparsity and Distribution:** Approximately 68% of observations exhibited zero sales, indicating highly sparse sales patterns requiring specialized handling. Non-zero sales displayed heavy right-skew, with the majority of active selling days recording modest volumes and only occasional high-volume spikes, suggesting intermittent demand patterns rather than consistent purchasing behavior.
- **Feature Relationships:** Correlation analysis revealed weak linear relationships between price and sales volume (r ≈ -0.13), suggesting that pricing alone does not directly drive demand through simple linear mechanisms. However, geographic identifiers showed moderate correlations (`snap_CA`, `snap_TX`, `snap_WI`: r = 0.40-0.55), indicating regional market dependencies.
- **Temporal Patterns:** Three distinct temporal hierarchies emerged:
    - Weekly cycles demonstrated pronounced weekend effects, with Saturday-Sunday sales reaching 1.35-1.40 x weekday baseline levels.
    - Monthly patterns exhibited remarkable stability year-round (normalized values 1.0-1.15), indicating minimal seasonal variation despite common retail expectations.
    - Annual trends revealed early-year peaks (months 1-5) followed by mid-year contraction and year-end stabilization, consistent across all geographic markets.
- **Geographic Heterogeneity:** Sales volumes varied substantially by state, with California consistently achieving highest performance, followed by Wisconsin and Texas. Individual store-level analysis revealed high day-to-day volatility (large interquartile ranges) across all locations, with frequent outliers indicating promotional or event-driven demand spikes.
- **Event Impact:** Event categorization demonstrated significant sales amplification, with religious events generating highest volumes (1.8M units), followed by National (1.5M) and Cultural (1.3M) events. Event timing proved critical - Sunday events generated 1.5M units versus Friday's 0.25M (6 x difference), indicating strategic promotional scheduling concentrated on peak shopping days.
- **Category Dynamics**: Calendar heatmap analysis across 2012-2015 revealed category-specific patterns: FOODS exhibited strong weekend concentration and holiday intensification; HOBBIES showed pronounced Q4 elevation reflecting gift-purchasing behavior; HOUSEHOLD demonstrated uniform weekday distribution, suggesting necessity-driven rather than discretionary purchasing.
- These findings necessitate a segmented forecasting approach that accounts for
    - sparse zero-inflated data,
    - strong weekly seasonality through cyclic feature encoding,
    - category-specific behaviors through stratified sampling, and
    - event-driven demand spikes through explicit event feature engineering.

Feature Engineering

To enhance model predictive capability, the dataset was systematically augmented with engineered features capturing temporal patterns, price dynamics, historical trends, and seasonality.

Temporal Features

Calendar-based features were extracted from the date column to capture cyclical patterns:

Basic Temporal Identifiers:

- `quarter`: Quarter of the year (1-4)
- `week_of_year`: ISO calendar week (1-53)
- `day_of_year`: Day of the year (1-366)

Binary Temporal Indicators:

- `weekend`: Binary flag for Saturday-Sunday (0/1)
- `payday`: Binary flag for 15th and last day of month (0/1)
- `summer`: Binary flag for June-August period (0/1)

Cyclic Temporal Encodings

Standard numerical representations of cyclical features can mislead models into assuming linear relationships (e.g., December being "far" from January). Sine-cosine transformations were applied to preserve cyclical nature:

- `month_sin, month_cos`: Cyclical encoding of month (1-12)
- `day_of_month_sin, day_of_month_cos`: Cyclical encoding of day within month (1-31)
- `day_of_week_sin, day_of_week_cos`: Cyclical encoding of weekday (0-6)

These transformations enable models to recognize that adjacent periods in the cycle (e.g., January and December) are temporally proximate.

Price Dynamics Features

- Understanding price context is critical for sales forecasting, as raw sell_price alone lacks relative information. Multi-horizon price features were engineered at 1-day, 7-day, 15-day, and 28-day windows:
- Lag Features (`price_lag_1, price_lag_7, price_lag_15, price_lag_28`): Historical prices at specified intervals, capturing price memory and enabling detection of price change patterns.
- Price Differentials (`price_diff_1, price_diff_7, price_diff_15, price_diff_28`): Calculated as current price minus lagged price. Negative values signal discounts or promotional onset, while positive values indicate price increases.

Rolling Statistics:

- `price_rolling_mean_*`: Moving average prices over specified windows, providing baseline price trends
- `price_rolling_std_*`: Price volatility measures, indicating promotional frequency and price elasticity

These multi-horizon features enable the model to detect both short-term promotions and longer-term pricing strategies.

Sales History Features

Past sales performance serves as a powerful predictor of future demand. Multi-window lagged and rolling statistics were computed at 1-day, 7-day, 15-day, and 28-day horizons:

- Lag Features (`sales_lag_1, sales_lag_7, sales_lag_15, sales_lag_28`): Historical sales at specified intervals, capturing:
- `sales_lag_1`: Immediate short-term momentum
- `sales_lag_7`: Weekly seasonality patterns

- `sales_lag_15`: Mid-term trends
- `sales_lag_28`: Monthly and longer-term cyclical patterns

Rolling Statistics:

- `sales_rolling_mean_*`: Moving average sales, providing trend indicators at different time scales
- `sales_rolling_std_*`: Sales volatility measures, capturing demand variability

All temporal features were computed after sorting by item-store combination (id) and date to ensure correct chronological order. A 1 day shift operation was applied before calculating rolling statistics to prevent data leakage (i.e., ensuring current-day sales do not influence current-day features).

Feature Consolidation

The SNAP (Supplemental Nutrition Assistance Program) day information, originally spread across three state-specific columns (`snap_CA`, `snap_TX`, `snap_WI`), was consolidated into a single snap binary indicator. For each row, the relevant state-specific SNAP status was transferred to the unified feature, reducing dimensionality while preserving information.

Missing Value Treatment

After creating lag and rolling features, NaN values naturally appeared at the beginning of each item-store time series (e.g., first 28 days for `sales_lag_28`). These represent periods where insufficient historical data existed rather than true missing values. Rows containing NaN in critical lag features were removed to ensure model training occurred only on complete feature sets, preventing misleading zero-imputation that would distort historical patterns.

Final Feature Organization

Features were systematically organized into logical groups for modeling clarity:

- (1) Identifiers (item, department, category, store, state)
- (2) Temporal features (date, year, quarter, week indicators)
- (3) Engineered calendar features (cyclic encodings, binary flags)
- (4) Event and SNAP features
- (5) Price features (raw and engineered)
- (6) Sales features (lag and rolling statistics), with the target variable sales_count positioned as the final column.

This comprehensive feature engineering framework provides the model with multi-scale temporal context, price dynamics, demand patterns, and seasonality information necessary for accurate retail demand forecasting.

Modeling

Model Selection Strategy

The modeling strategy was designed to address the unique challenges of retail demand forecasting while establishing a progression from baseline to advanced methods. Three algorithms were selected to capture different aspects of the forecasting problem: Random Forest, LightGBM, and XGBoost.

Random Forest was selected as the baseline model for several compelling reasons. First, it provides robust performance out-of-the-box with minimal hyperparameter tuning, making it an ideal reference point for evaluating more complex methods. Second, its ensemble nature (averaging multiple decision trees) naturally handles non-linear relationships and feature interactions common in retail data, such as the interplay between price cha nges, events, and day-of-week effects. Third, Random Forest is relatively insensitive to feature scaling and can handle mixed data types (categorical and numerical) without extensive preprocessing. Finally, it provides straightforward feature importance metrics through mean decrease in impurity, offering interpretable insights into which factors most strongly influence sales predictions. These characteristics make Random Forest an appropriate baseline to establish whether more sophisticated gradient boosting methods yield meaningful performance improvements.

Both LightGBM and XGBoost were selected as advanced alternatives due to their proven superiority in time-series forecasting competitions and their ability to capture complex temporal dependencies. LightGBM offers computational efficiency through gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB), making it particularly well-suited for the dataset's scale (59+ million observations). Its leaf-wise tree growth strategy can achieve higher accuracy with fewer iterations compared to level-wise approaches. XGBoost, meanwhile, provides robust regularization capabilities (L1 and L2) that help prevent overfitting on the sparse sales data (68% zero values), along with built-in handling of missing values and sophisticated tree pruning mechanisms.

Random Forrest Modeling

Handling the massive M5 dataset (58.3 million observations) required a rigorous memory optimization strategy before invoking memory-intensive bagging algorithms. A custom `reduce_mem_usage()` function was applied immediately upon loading the Parquet files, downcasting numerical datatypes to their most efficient bit-representations (e.g., float64 to float32, int64 to int8/int16). This operation reduced the dataframe's memory footprint from 10.07 GB to 6.01 GB, a 40.3% reduction, which was a prerequisite for avoiding OOM (Out of Memory) errors during the highly parallelized Random Forest training. Following optimization, the data was partitioned using a time-series validation split: data prior to 2016-04-25 formed the training set (57.4 million rows), while the final 28 days served as the validation set (853,720 rows), strictly preserving temporal order to prevent data leakage.

While linear models were considered, they were deemed unsuitable due to their inability to capture non-linear interactions (e.g., the specific lift of SNAP events on weekends) without extensive feature engineering. Consequently, Random Forest was selected for its inherent ability to model high-order interactions and handle high-cardinality categorical data. To manage dimensionality, Label Encoding was applied to the nine categorical features (including `item_id`, `store_id`, and `event_name`). Unlike One-Hot Encoding, which would have exploded the feature space by adding over 3,000 columns for items alone, Label Encoding compressed these categories into single integer columns. This kept the total feature count to a manageable 38, allowing the tree-based algorithm to split on categorical indices effectively without exhausting RAM.

Training a Random Forest on 58 million rows is computationally expensive due to the algorithm's memory requirements. To make hyperparameter optimization feasible, a 10% sampling strategy was implemented directly within the Optuna objective function. For each trial, a new random subset of the training data (approximately 5.8 million rows) was drawn using np.random.choice without replacement. This approach enabled rapid experimentation while maintaining enough data diversity to approximate the distribution of the full dataset. Using this sampled training procedure, Optuna executed twenty trials with the Tree-structured Parzen Estimator (TPE) sampler to optimize five key hyperparameters:

- `n_estimators` (50-300): The number of trees in the forest.
- `max_depth` (10-30): Controlling the complexity of individual learners.
- `min_samples_leaf` (20-150): A regularization constraint to prevent overfitting on noisy sales data.
- `min_samples_split` (2-50): The threshold for creating new nodes.
- `max_features` (sqrt, log2, None): The number of features considered for the best split. The optimization converged on Trial 10 with a validation RMSE of 1.9584.

The optimal configuration favored a robust ensemble (`n_estimators` = 234) with constrained depth (`max_depth` = 25) and `min_samples_leaf` = 'log2', indicating that restricting the feature set at each split helped decorrelate the trees and improve generalization. Final Training and Computational Performance The final model was trained on the full 57.4 million observation dataset using the optimal hyperparameters. To handle the computational load, the training was parallelized across all available cores (`n_jobs` = -1. The final model achieved a Validation RMSE of 1.9392 (MSE: 3.7606, MAE: 0.9933). However, post-training feature importance analysis revealed a critical insight: the model relies disproportionately on sales_rolling_mean_7, which accounted for the vast majority of the predictive power. While this reliance provided a strong baseline metric, it suggests the model acts as a "lazy learner", effectively mimicking the previous week's trend while under-utilizing exogenous features like price changes or calendar events. This finding indicates that while the Random Forest architecture is robust, future improvements will likely require boosting algorithms (like LightGBM or XGBoost) to force the learning of subtler signals from non-dominant features.

XGBoost Modeling

Given the substantial scale of the dataset, aggressive memory optimization was implemented prior to model training. The dataset was partitioned using a time-based validation strategy, with the final 28 days reserved for hold-out validation. This approach preserved temporal ordering (critical for time-series forecasting) resulting in a 58.3 million observation training set and 854,000 observation validation set. All features were sorted by item-store combination (`item_id, store_id`) and date to maintain chronological integrity throughout the training pipeline.

Direct hyperparameter optimization on the full 58 million observation dataset proved computationally prohibitive. To balance optimization quality with computational efficiency, a stratified sampling strategy was employed. Rather than simple random sampling, which might under-represent minority product departments, sampling was stratified by `dept_id` to ensure proportional representation across all product categories. A 10% sample fraction was applied within each department, yielding 5.8 million observations from 3,049 unique item-store time series. This stratification ensured that hyperparameter optimization reflected the full diversity of product behaviors (e.g., high-volume FOODS items, seasonal HOBBIES products, steady HOUSEHOLD goods) rather than being dominated by the most common categories.

XGBoost requires numerical inputs, necessitating encoding of the nine categorical features: `item_id`, `dept_id`, `cat_id`, `store_id`, `state_id`, and four event-related columns. Target Encoding was selected over alternatives (one-hot encoding, label encoding) for several compelling reasons. First, one-hot encoding would have created thousands of binary columns given the high cardinality of features like `item_id` (3,049 unique products), leading to extreme dimensionality and memory exhaustion. Second, simple label encoding assigns arbitrary numerical ordering to categories, which tree-based models can misinterpret as meaningful relationships. Target Encoding addresses both issues by replacing categorical values with the mean target value for that category, with Bayesian smoothing (smoothing parameter = 1.0) to prevent overfitting on rare categories. This approach captures the predictive signal of each category while maintaining a compact numerical representation. The encoder was fit on the training sample and subsequently applied to both training and validation sets, ensuring proper prevention of data leakage.

The choice of loss function directly impacts model performance on zero-inflated count data. The `reg:tweedie` objective was selected specifically to address the dataset's severe sparsity (68% zero sales). Tweedie distributions are compound Poisson-Gamma distributions that naturally model scenarios where many observations are exactly zero, with positive values following a right-skewed distribution, precisely matching retail sales patterns. The Tweedie variance power parameter was set to 1.5, positioning the distribution between Poisson (variance power = 1) and Gamma (variance power = 2). This intermediate value allows the model to simultaneously handle:

- (1) the high frequency of zero-sale days
- (2) the right-skewed distribution of non-zero sales
- (3) overdispersion where variance exceeds the mean.

Alternative objectives like squared error (`reg:squarederror`) penalize all errors equally, leading to systematic under-prediction of zero values and over-prediction of high sales. The Tweedie objective's asymmetric loss function appropriately weights these distinct regimes.

Hyperparameter optimization was conducted using **Optuna's Tree-structured Parzen Estimator** (TPE) algorithm rather than grid search or random search. TPE builds a probabilistic model of the hyperparameter space, focusing computational resources on promising regions while pruning unlikely configurations early. This Bayesian optimization approach is significantly more sample-efficient than exhaustive grid search, particularly critical given the computational cost of training XGBoost models on millions of observations. Twenty optimization trials were executed, with each trial evaluated using 3-fold `TimeSeriesSplit` cross-validation on the stratified sample. `TimeSeriesSplit` maintains temporal ordering by creating sequential train-validation splits, preventing future data leakage into past predictions. Nine hyperparameters were optimized simultaneously:

Tree Structure Parameters:

- `n_estimators` (200-800): Number of boosting rounds, controlling model complexity and training time
- `max_depth` (4-12): Maximum tree depth, balancing expressiveness against overfitting
- `min_child_weight` (1-10): Minimum sum of instance weights in child nodes, preventing excessive splitting on sparse data

Learning Parameters:

- `learning_rate` (0.01-0.3, log-scale): Step size for each boosting iteration, traded against n_estimators

- `subsample` (0.6-1.0): Fraction of observations sampled per tree, introducing stochasticity for regularization
- `colsample_bytree` (0.6-1.0): Fraction of features sampled per tree, reducing feature correlation

Regularization Parameters:

- `gamma` (0-0.5): Minimum loss reduction required for splits, pruning insignificant branches
- `reg_alpha` (0-1.0): L1 regularization on leaf weights, encouraging sparsity
- `reg_lambda` (0.1-5.0): L2 regularization on leaf weights, smoothing predictions

The optimal configuration revealed several notable patterns: moderate tree depth (7), suggesting complex but not overly intricate decision boundaries; high feature sampling (0.986), indicating that most features contribute useful signal; and substantial L2 regularization (3.16), confirming the need for smoothing given data sparsity.

LightGBM Modeling

The LightGBM implementation leveraged the identical data processing pipeline established for XGBoost, utilizing the memory-optimized dataset (downcasted types) and the time-series validation split. To maintain consistency in handling high-cardinality features like `item_id`, the Target Encoding strategy described in the XGBoost section was applied to the nine categorical variables. This ensured that both gradient boosting models operated on the same numerical representation of the data, allowing for a direct comparison of algorithmic efficiency and architectural differences.

To address the high sparsity of the sales data (68% zeros), LightGBM was also configured with the Tweedie objective. By maintaining the tweedie_variance_power at 1.5, the model treated the target variable as a compound Poisson-Gamma distribution. This ensured that the loss function appropriately penalized errors across both the zero-inflated regime and the continuous sales distribution, preventing the under-prediction bias common with standard regression objectives.

Hyperparameter optimization was also conducted using Optuna's TPE, utilizing the same 10% stratified sample strategy to balance computational speed with representative data distribution. Twenty trials were executed using 3-fold `TimeSeriesSplit` cross-validation. While sharing learning rate and regularization parameters with XGBoost, specific tree-growth parameters were tuned to exploit LightGBM's leaf-wise growth strategy:

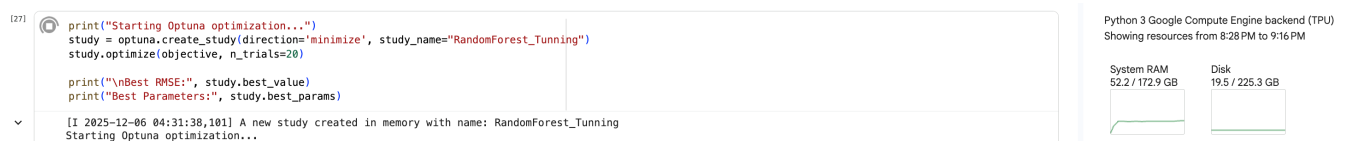
- `num_leaves` (31-127): Controls the complexity of the tree model; leaf-wise growth typically achieves lower loss than level-wise growth for the same number of leaves.
- `feature_fraction & bagging_fraction` (0.6-1.0): To prevent overfitting and speed up training.
- `lambda_l1 & lambda_l2`: Regularization terms to handle noise in the sparse data.

Training the Models

To improve model stability, we explicitly dropped the date and `wm_yr_wk` (week ID) columns prior to training. Since tree-based models cannot process raw datetime objects and might misinterpret the arbitrary integer increment of the week ID as a linear trend, these features introduced noise. Instead, we relied on the engineered cyclical features (sine/cosine transformations of month and day) and lag features, which provided the models with a more mathematically accurate representation of seasonality and temporal continuity without the risk of data leakage or confusion.

Running Bayesian optimization (Optuna) on the full dataset of 58 million rows was computationally infeasible. To overcome this, we employed a stratified sampling strategy, using only 10% of the training data for the hyperparameter search. This approach allowed us to run 20 optimization trials in a fraction of the time, operating on the assumption that the hyperparameter surface of the representative sample would map closely to the full dataset. Once the optimal parameters were identified, they were applied to the full dataset for the final training run, striking a balance between model tuning precision and training time.

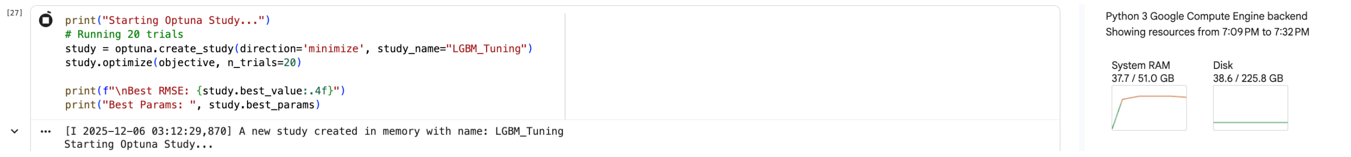
We faced significant hardware limitations regarding System RAM. Gradient Boosting models (LightGBM and XGBoost) store feature histograms in memory; unconstrained tree growth on a dataset of this magnitude frequently caused Out-Of-Memory (OOM) crashes on standard Google Colab runtimes. To mitigate this, we had to carefully bound parameters such as `max_depth` and `num_leaves` during the optimization phase to prevent exponential memory growth. Ultimately, to accommodate the memory footprint of the full training set, we utilized the NVIDIA A100 GPU runtime, utilizing its high-bandwidth memory to handle the large feature matrix that standard GPUs could not process.



```
[27]
print("Starting Optuna optimization...")
study = optuna.create_study(direction='minimize', study_name="RandomForest_Tunning")
study.optimize(objective, n_trials=20)

print("\nBest RMSE:", study.best_value)
print("Best Parameters:", study.best_params)

[I 2025-12-06 04:31:38,101] A new study created in memory with name: RandomForest_Tunning
Starting Optuna optimization...
```

```
Python 3 Google Compute Engine backend (TPU)
Showing resources from 8:28 PM to 9:16 PM

System RAM          Disk
52.2 / 172.9 GB     19.5 / 225.3 GB
```

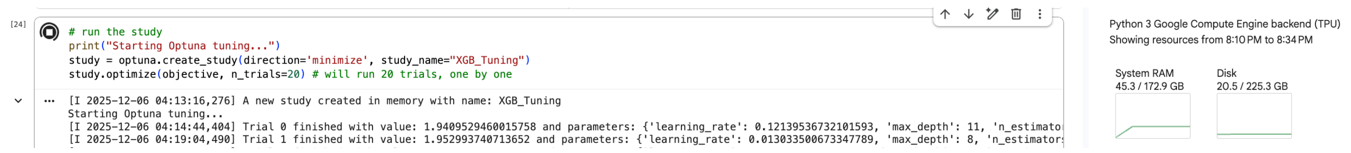
Figure 17: Random Forest Model Tunning with Optuna



```
[27]
print("Starting Optuna Study...")
# Running 20 trials
study = optuna.create_study(direction='minimize', study_name="LGBM_Tuning")
study.optimize(objective, n_trials=20)

print(f"\nBest RMSE: {study.best_value:.4f}")
print("Best Params: ", study.best_params)

[I 2025-12-06 03:12:29,870] A new study created in memory with name: LGBM_Tuning
Starting Optuna Study...
```

```
Python 3 Google Compute Engine backend
Showing resources from 7:09 PM to 7:32 PM

System RAM          Disk
37.7 / 51.0 GB      38.6 / 225.8 GB
```

Figure 18: LightGBM Model Tunning with Optuna



```
[24]
# run the study
print("Starting Optuna tuning...")
study = optuna.create_study(direction='minimize', study_name="XGB_Tuning")
study.optimize(objective, n_trials=20) # will run 20 trials, one by one

[I 2025-12-06 04:13:16,276] A new study created in memory with name: XGB_Tuning
Starting Optuna tuning...
[I 2025-12-06 04:14:44,404] Trial 0 finished with value: 1.9409529460015758 and parameters: {'learning_rate': 0.12139536732101593, 'max_depth': 11, 'n_estimato
[I 2025-12-06 04:19:04,490] Trial 1 finished with value: 1.952993740713652 and parameters: {'learning_rate': 0.013033500673347789, 'max_depth': 8, 'n_estimator
```

```
Python 3 Google Compute Engine backend (TPU)
Showing resources from 8:10 PM to 8:34 PM

System RAM          Disk
45.3 / 172.9 GB     20.5 / 225.3 GB
```

Figure 19: XGBoost Model Tunning with Optuna

Results

Model Performance Comparison

Feature importance analysis reveals the hierarchical structure of predictive signals within the forecasting problem. Figure 20 presents the top 30 features ranked by their contribution to model performance.

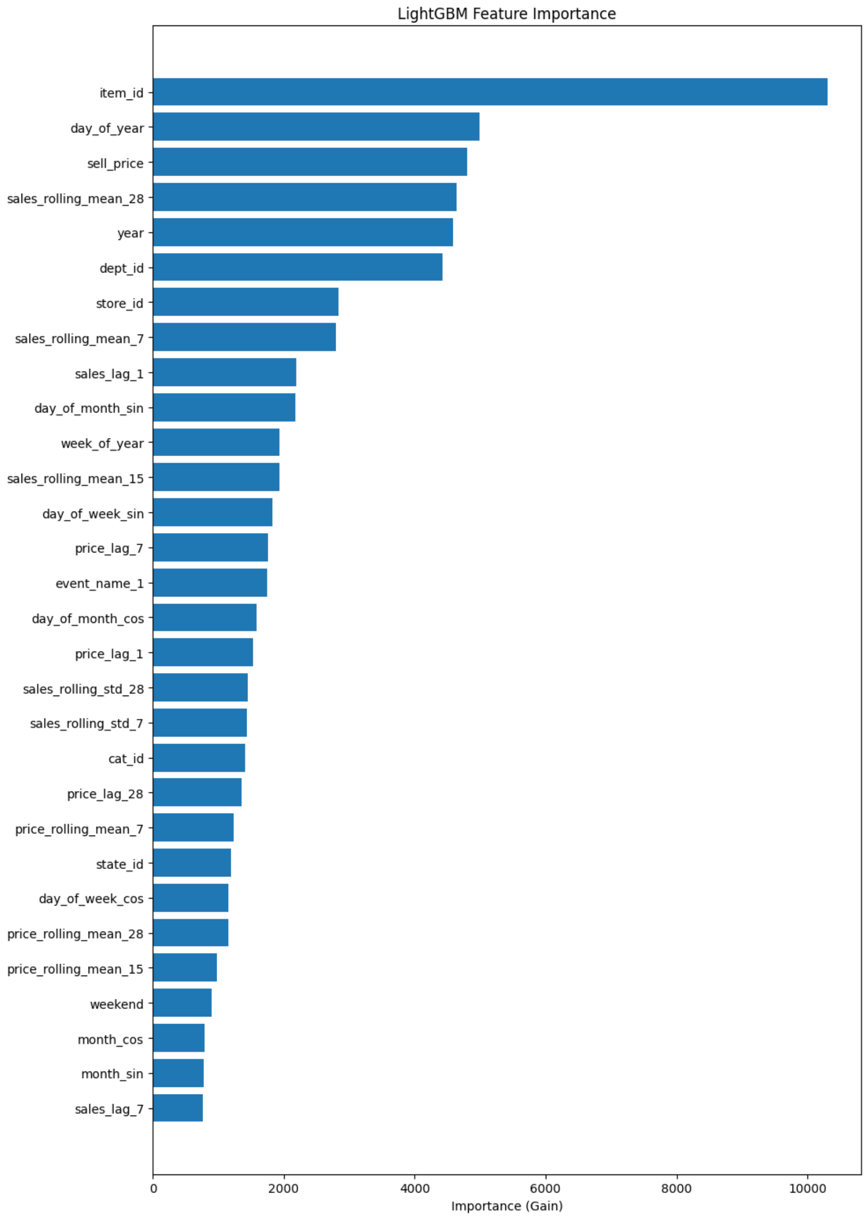


Figure 20: Top 30 important features from used by the LightGBM model.

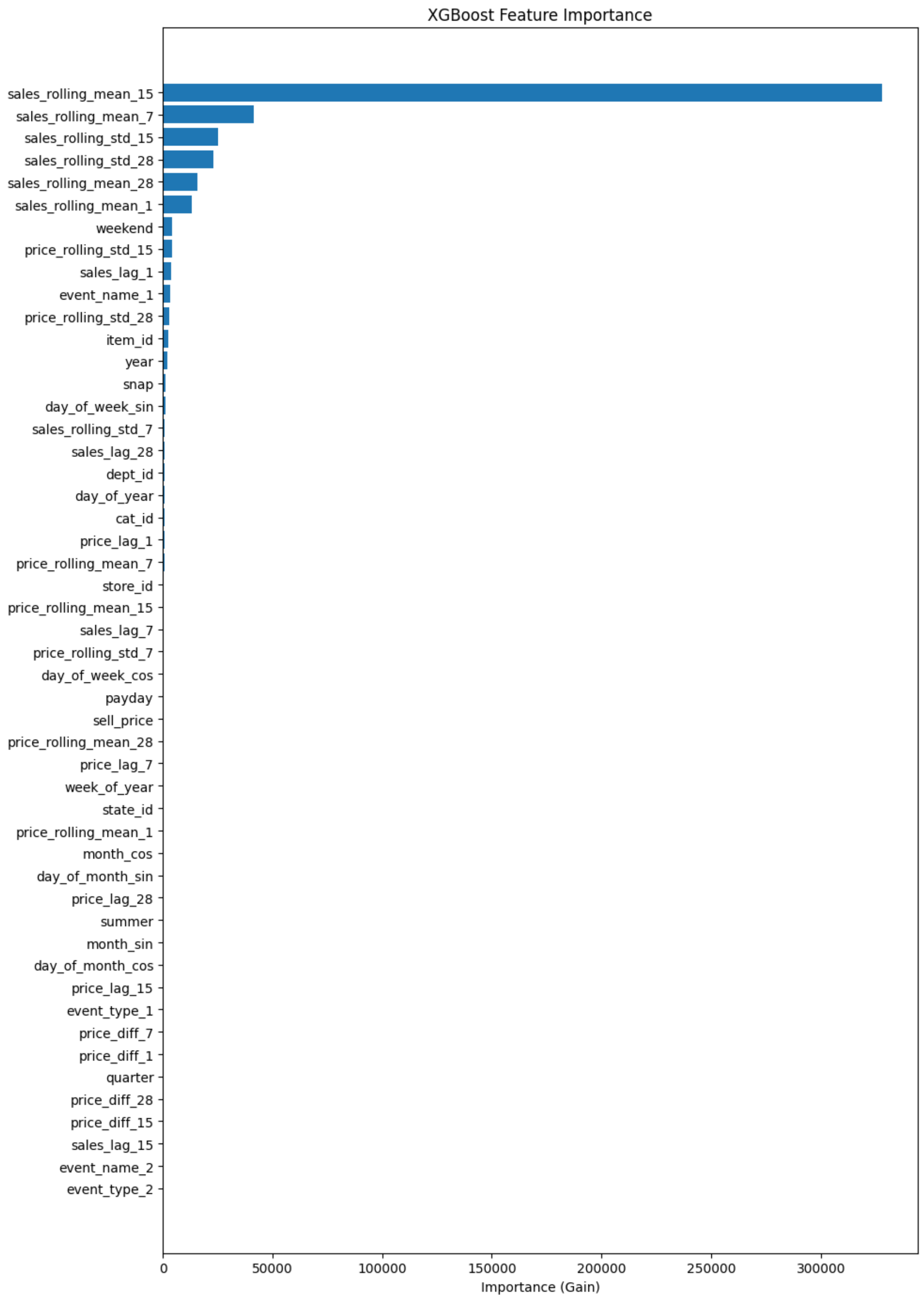


Figure 21: Feature Importances for the XGBoost model.

The analysis reveals a strong reliance on engineered temporal features, with item_id (product identifier) exhibiting the highest importance score (~10,000 gain), followed by temporal indicators day_of_year and sell_price (each ~5,000 gain). This hierarchical pattern suggests that, product-level heterogeneity exerts the strongest influence on sales patterns, indicating that individual item characteristics (category, brand positioning, price point) fundamentally determine demand trajectories. Cyclical temporal patterns captured through day_of_year enable the model to learn seasonal purchasing cycles and holiday effects Price sensitivity manifests as the third-most influential feature, validating the importance of dynamic pricing strategies in demand forecasting.

Multi-horizon rolling statistics demonstrate differential predictive power:

- sales_rolling_mean_28 (28-day moving average): ~4,000 importance, capturing monthly trend momentum
- sales_rolling_mean_7 (weekly average): ~3,000 importance, reflecting short-term demand patterns
- sales_lag_1 (previous day sales): ~2,500 importance, indicating strong autocorrelation

This declining importance with increasing time horizon suggests that recent sales history provides stronger predictive signals than distant past performance, consistent with the non-stationary nature of retail demand.

Calendar and Event Features: Mid-tier importance features include:

- Cyclic encodings (day_of_month_sin, week_of_year): ~2,000-2,500 importance
- Binary indicators (weekend, event_name_1): ~1,500-2,000 importance
- Event types and SNAP benefits: ~1,000-1,500 importance

The moderate contribution of event features (despite their dramatic EDA-observed sales impacts) likely reflects the sparsity of promotional periods relative to regular trading days, resulting in lower aggregate feature importance despite high marginal effects during event windows.

For comparison, Figure 21 presents XGBoost's feature importance distribution:

Cross-Model Consistency: XGBoost exhibits a remarkably different feature importance profile compared to LightGBM, with:

- sales_rolling_mean_15 dominating (~300,000 importance) versus LightGBM's more distributed importance
- Substantially lower relative importance for product identifiers and temporal features
- Similar low-tier positioning for price-related features

This divergence reflects algorithmic differences in tree construction strategies: XGBoost's level-wise growth tends to concentrate importance on features that provide consistent splits across all samples, while LightGBM's leaf-wise approach distributes importance more evenly across features that optimize loss reduction in specific data partitions.

Residual Analysis and Error Distribution

Residual analysis provides critical insights into model bias, heteroscedasticity, and systematic misprediction patterns across different product segments. Disaggregating performance by product category reveals systematic heterogeneity in model accuracy (Table 1).

Despite FOODS representing the highest volume category (mean 2.069 units/day), it exhibits the highest absolute error (MAE = 1.256) but paradoxically the best relative performance (60.7% error rate relative to mean sales). The small positive mean residual (+0.045) indicates systematic under-prediction, likely attributable to:

- Perishability-driven demand volatility not captured by lag features
- Weekend shopping spikes that exceed historical rolling averages

- Promotional events generating outsized demand increases

Table 1: Error Metrics by Product Category for the LightGBM model

Category	Mean Residual	Mean Absolute Error	Mean Sales Count
FOODS	0.045	1.256	2.069
HOBBIES	0.018	0.740	0.732
HOUSEHOLD	0.021	0.746	0.967

HOBBIES demonstrates the lowest MAE (0.740) but the highest relative error rate (101.1%), indicating that while absolute errors are small, they represent substantial proportions of typical sales volumes. The near-zero mean residual (+0.018) suggests unbiased predictions on average, with error driven primarily by:

- High inherent demand variability for discretionary products
- Seasonal concentration of purchases (birthday/holiday gift cycles)
- Stock-keeping unit (SKU) level heterogeneity within the category

HOUSEHOLD occupies an intermediate position with MAE = 0.746 and 77.1% error rate. The category exhibits stable baseline demand with moderate volatility, lower promotional sensitivity compared to FOODS, predictable replenishment cycles for consumable household goods.

Similar errors were found for the XGBoost Model [Table 2].

Table 2: Error Metrics by Product Category for the XGBoost model

Category	Mean Residual	Mean Absolute Error	Mean Sales Count
FOODS	0.048	1.256	2.069
HOBBIES	0.022	0.739	0.732
HOUSEHOLD	0.022	0.745	0.967

The predicted vs actual plots for the both models are shown in Figure 22

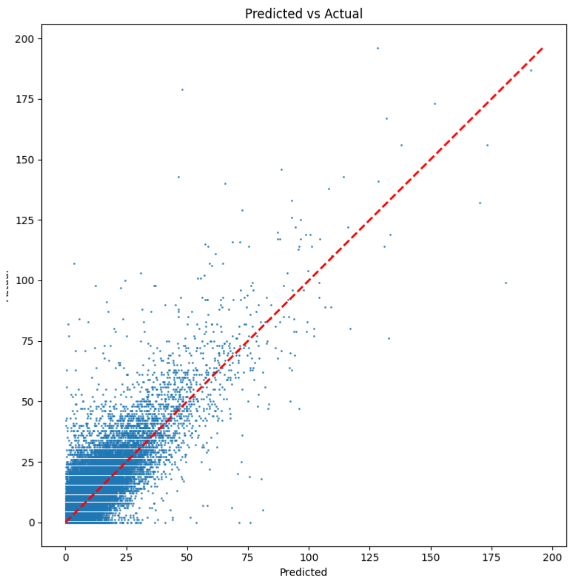


Figure 22: Predicted vs Actual values for the XGBoost model.

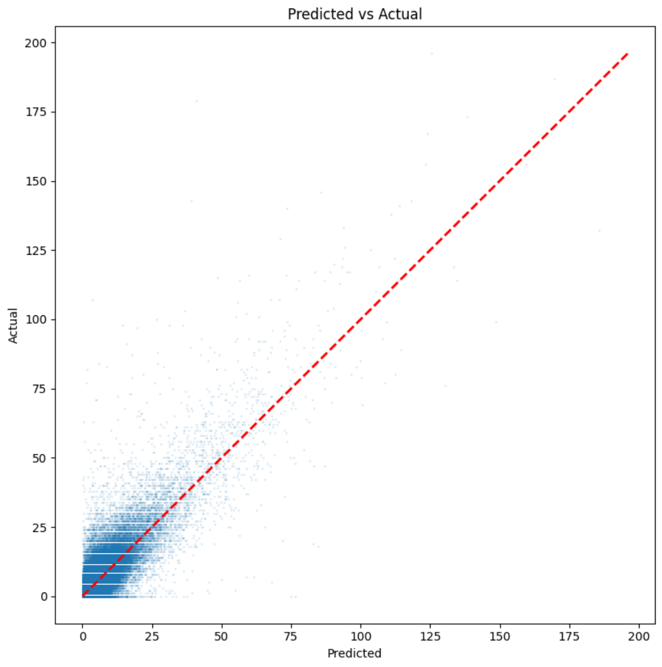


Figure 23: Predicted vs Actual values for the LightGBM model.

Forecast Performance Validation

The temporal validation plots (Figure X) demonstrate the LightGBM model's forecasting capability across the three geographic markets over the 28-day validation period (late April to May 2016). The visualizations reveal strong pattern recognition in all three states, with the forecast (blue line) closely tracking the actual values (green line) and successfully capturing the characteristic weekly seasonality observed throughout the training period (gray line). Notably, the model exhibits superior performance in California, where forecast predictions align almost perfectly with realized sales, including accurate prediction of weekend demand spikes that reach approximately 24,000 units. Texas and Wisconsin forecasts similarly track actual patterns well, though with slightly more variance during mid-week periods, consistent with these markets' lower absolute sales volumes (Texas: 9,000-16,000 units; Wisconsin: 10,000-19,000 units) and consequently higher relative volatility.

The vertical dotted line demarcating the train-test boundary illustrates the model's seamless transition from historical fitting to out-of-sample prediction, with no apparent degradation in pattern recognition or systematic bias at the forecast horizon. The consistent preservation of weekly cyclicality across all three states validates the effectiveness of the engineered temporal features (day-of-week encodings, weekend flags) and confirms that the model has learned generalizable demand patterns rather than overfitting to training-specific noise. Minor deviations between forecast and actual values occur primarily at extreme peaks (e.g., Wisconsin's ~19,000 unit spike in late April), reflecting the inherent difficulty in predicting promotional or event-driven demand surges that exceed historical baselines - a limitation consistent with the positive residual skewness discussed in the error analysis. For reference XGBoost's temporal validation plot is also shown in Figure

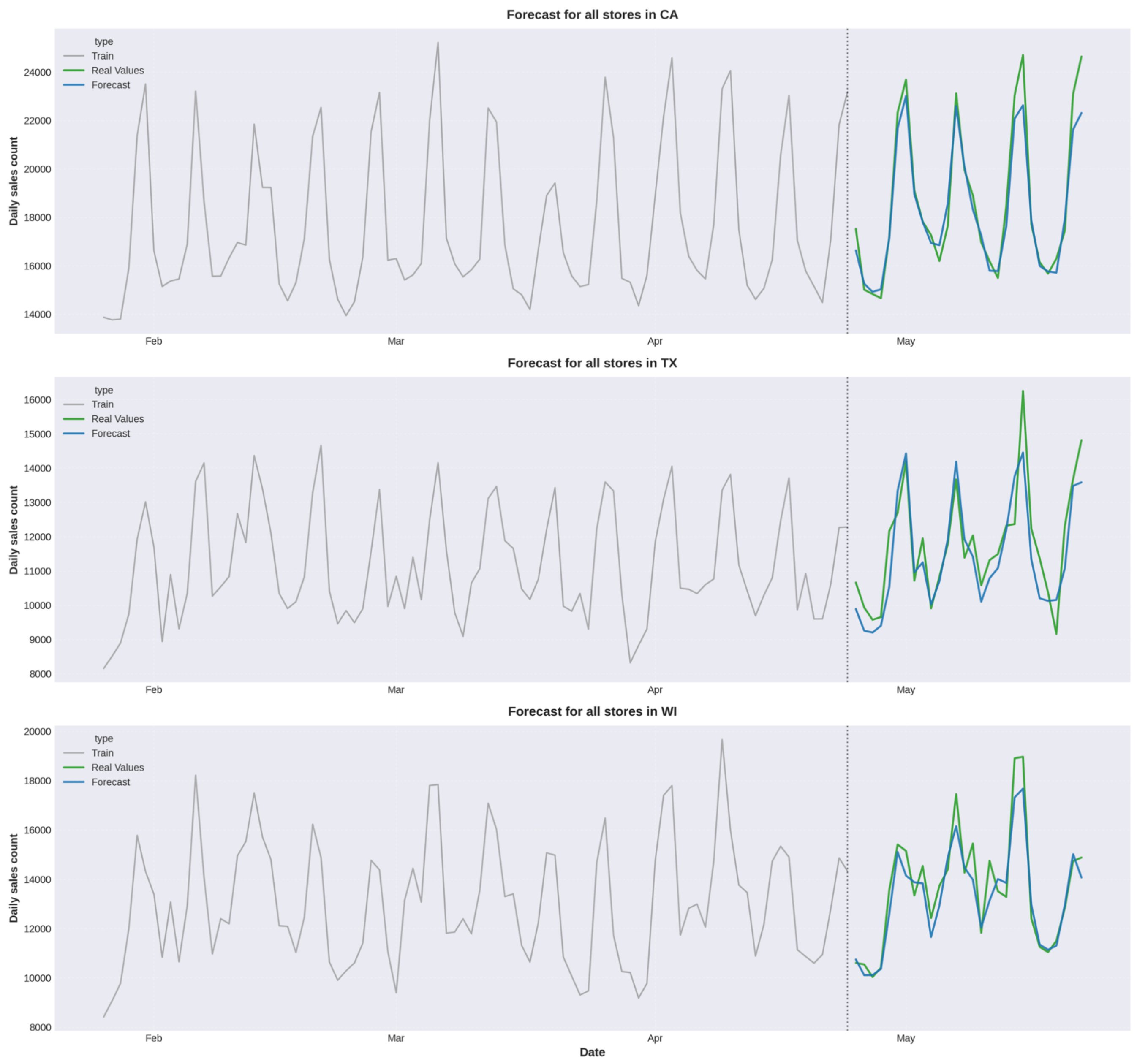


Figure 24: Prediction of sales for the stores in each state (LightGBM)

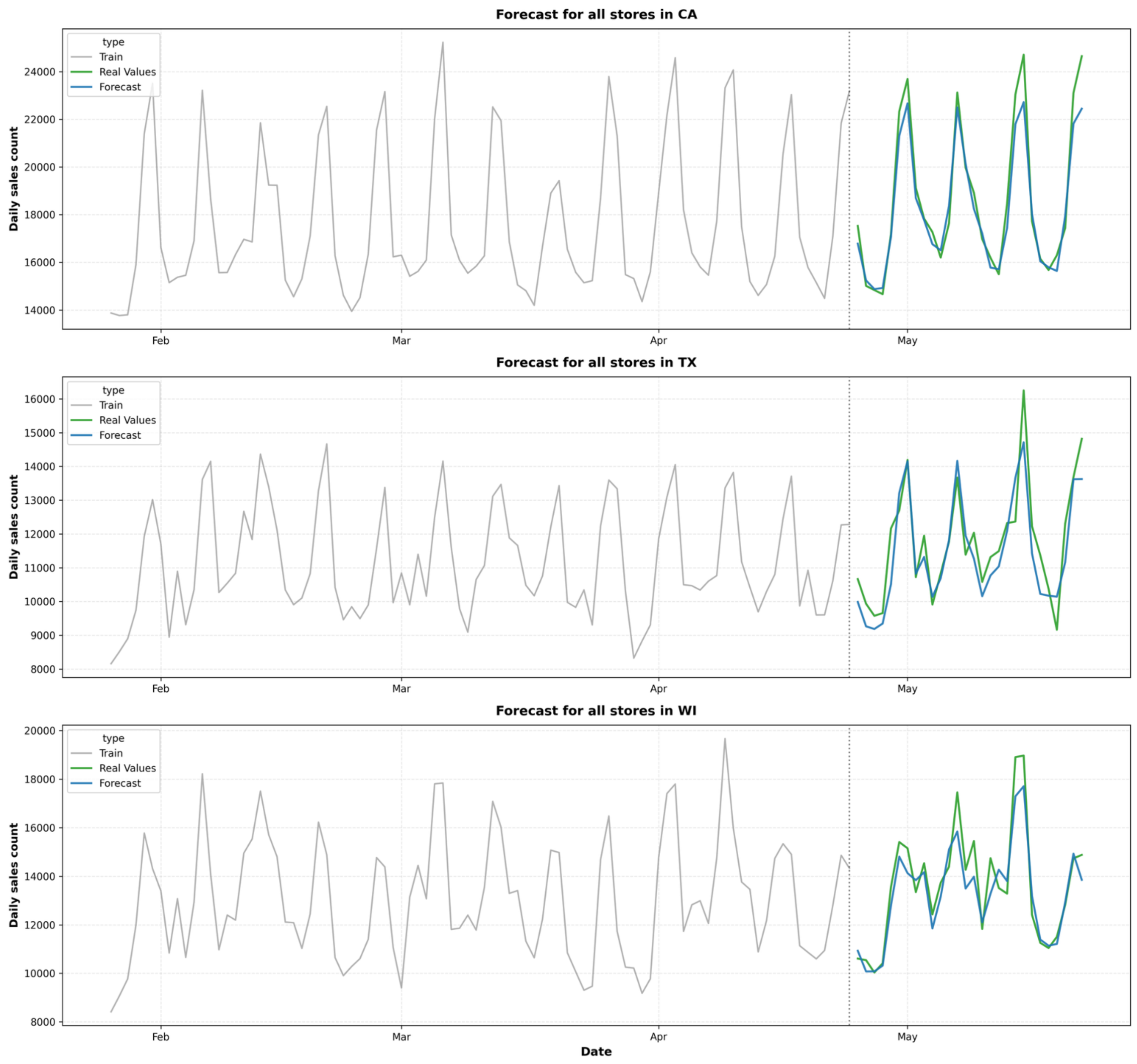


Figure 25: Prediction of sales for the stores in each state (XGBoost)

Analysis of Results

We ultimately selected LightGBM as the final forecasting model, determining that its superior speed and predictive precision justified a minor trade-off in memory consumption. While LightGBM required slightly more system RAM (47.5 GB) compared to XGBoost (45.2 GB), this marginal increase of 2.3 GB was negligible when weighed against the model's substantial efficiency gains. LightGBM completed the entire tuning and training pipeline in just 56 minutes. This represents a dramatic improvement in speed: it is approximately 41% faster than XGBoost (1 hour 35 minutes) and 55% faster than the Random Forest baseline, which took nearly 2 hours and 6 minutes (126 minutes) to complete. Furthermore, this specific architectural choice yielded the highest accuracy, with LightGBM achieving a Final Validation RMSE of 1.9213, effectively outperforming both XGBoost (1.934) and Random Forest (1.9392). Thus, the decision to prioritize LightGBM reflects a strategic choice to maximize computational throughput without compromising forecast quality..

Model	Tunning Time	Training Time	PEAK RAM	Final RMSE
LightGBM	51 minutes	5 minutes	47.5 GB	1.9213
XGBoost	1 hr 25 minutes	10 minutes	45.2 GB	1.934
Random Forest	1 hr 39 minutes	27 minutes	128 GB	1.9392

Future Works:

- Hierarchical forecasting: Reconciling individual item forecasts with category/store-level constraints using MinT or ERM approaches
- Deep learning architectures: Temporal Fusion Transformers or N-BEATS for capturing long-range dependencies
- Probabilistic forecasting: Quantile regression or distributional models for uncertainty quantification
- Causal inference: Difference-in-differences or synthetic control methods for isolating promotional effects

Citations:

1. Paper: Fildes, R., Ma, S., & Kolassa, S. (2022). Retail forecasting: Research and practice. International Journal of Forecasting, 38(4), 1283-1318.
2. M5 Forecast Accuracy. University of Nicosia · Featured Prediction Competition. Can be accessed at: https://www.kaggle.com/competitions/m5-forecasting-accuracy
3. Mastering Time Series Analysis in Python: Tools for Time Series Analysis and Forecasting in Python. Sadrach Pierr. Link: https://towardsdatascience.com/mastering-time-series-analysis-in-python-8219047a0351/
4. AdaBoost in Machine Learning. Geeksforgeeks. Link: https://www.geeksforgeeks.org/machine-learning/AdaBoost-in-Machine-Learning/
5. Ke, G., Meng, Q., Finley, T., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.
6. A PySpark Example for Dealing with Larger than Memory Datasets. Georgia Deaconu. Link: https://towardsdatascience.com/a-pyspark-example-for-dealing-with-larger-than-memory-datasets-70dbc82b0e98/

Note:

The current works of this project can be found at:

- Google Drive: https://drive.google.com/drive/folders/1A4Z3vvPe3zk1H781INxY8sH3koFI0viX?usp=sharing (the link is currently for view only)
- Github Repository: https://github.com/htnphu/retail-demand-forecasting