A Framework for State-of-the-Art Retail Sales Forecasting: An Integrated Solution for Competitive Time Series Analysis

Section 1: Architectural Blueprint and Validation Protocol

This report outlines a comprehensive and robust framework for tackling a complex, large-scale retail sales forecasting challenge. The objective is to predict the quantity of items sold per product, per store, on a weekly basis for a future period (January 2023), using historical data from the preceding year (2022). The problem is defined by two critical constraints that elevate its complexity beyond a standard regression task: the composition of the test set is unknown, and the final submission is strictly limited to 1,500,000 rows. These constraints necessitate a sophisticated, multi-stage solution that addresses not only the magnitude of sales but also the probability of a sale occurring. This initial section establishes the strategic blueprint for the entire project, defining the problem's unique characteristics and constructing a rigorous validation protocol that ensures the reliability and generalizability of the developed models. This protocol forms the bedrock upon which all subsequent feature engineering, modeling, and ensembling work is built.

1.1 Deconstructing the Forecasting Problem: A Dual Challenge

A successful approach begins with a precise definition of the problem. The competition's structure, particularly the unknown test set and the submission row limit, dictates that this is not a single regression task. Instead, it must be framed as a dual challenge comprising two distinct but interconnected sub-problems:

1. **Activity Forecasting (Classification):** Predicting *which* specific product-store-week combinations will have non-zero sales. This is a classification task to identify the active, sale-generating entities from a vast universe of possibilities.
2. **Volume Forecasting (Regression):** For the combinations identified as active, predicting the *quantity* of items that will be sold. This is the traditional regression component of the forecast.

This dual-problem framing is a direct and necessary consequence of the competition's constraints. The 1,500,000 row limit is not merely a technical hurdle; it functions as a strategic filter that fundamentally redefines the objective. A naive approach of predicting sales for every product-store combination that has ever recorded a sale would generate a submission file far exceeding this limit. The constraint implicitly forces a prioritization of model precision over recall, rewarding solutions that can accurately identify the "signal" (the most probable active sales combinations) from the vast "noise" (all other potential combinations). This transforms the problem from one of pure prediction into one of resource allocation under uncertainty, where the limited "prediction budget" of 1.5 million rows must be allocated to the candidates with the highest likelihood of generating a return on the evaluation metric.

The dataset's inherent characteristics, which share significant parallels with those from well-documented challenges like the M5 forecasting competition, further inform the solution architecture.[1] These characteristics include:

- **Hierarchical Structure:** The data is naturally organized in a hierarchy. Individual product sales (SKUs) can be aggregated into departments, which in turn form categories. Likewise, store-level sales can be aggregated to the state and national levels.[2] This structure is not a complication but a powerful asset. It allows for the creation of robust features and provides a principled mechanism for addressing the "cold start" problem for new products or stores by borrowing statistical strength from related time series at higher, more stable levels of aggregation.[3]
- **High-Frequency and Intermittent Data:** The availability of daily or weekly sales data provides high granularity but also introduces the challenge of intermittency—a high proportion of zero-sales observations for many items, particularly at the lowest level of the hierarchy.[1] This zero-inflation can severely bias standard regression models and necessitates specialized modeling techniques, such as the two-stage approach detailed later in this report.[5] The task requires forecasting not only "how much" but also "when" a sale will occur.[1]
- **Exogenous Variables:** The dataset includes external information, such as product prices and promotional events.[2] This data is a critical advantage for machine learning models, which can explicitly incorporate these drivers into the prediction, unlike many classical statistical methods that are purely univariate.[1]

1.2 Establishing a Robust Time-Aware Validation Framework

The single most critical component of a winning solution in any time series competition is a validation framework that accurately mimics the final evaluation scenario and rigorously prevents data leakage. Standard cross-validation techniques, such as k-fold, are fundamentally flawed for time series data. They operate on the assumption that data points are independent and identically distributed, an assumption that is violated by the temporal dependencies inherent in sales data (e.g., trends, seasonality, and autocorrelation).[8] Randomly shuffling and splitting time series data allows the model to train on future information to predict the past, a phenomenon known as "temporal leakage." This leads to artificially inflated and dangerously misleading performance estimates, causing models to fail when deployed on genuinely unseen future data.[8]

To overcome this, a time-aware validation strategy is essential. The recommended approach is **Rolling-Origin Cross-Validation**, also known as expanding window or walk-forward validation.[8] This method respects the chronological order of the data and simulates the real-world process of periodically retraining a model as new data becomes available. The implementation, often facilitated by libraries such as

scikit-learn's TimeSeriesSplit, follows a precise sequence [8]:

1. The historical training data (the year 2022) is divided into a series of sequential folds.
2. In the first fold, the model is trained on an initial block of data (e.g., January-September 2022).
3. The model's performance is then evaluated on the immediately subsequent period (e.g., October 2022), which serves as the validation set.
4. For the second fold, the training window expands to include the data from the previous validation set (e.g., training on January-October 2022) and is evaluated on the next block of time (e.g., November 2022).
5. This process repeats, with the training window progressively expanding, until all available data has been used.

The parameters of this validation scheme—specifically the number of splits and the size of each validation window—must be chosen to align with the competition's forecast horizon. Since the final task is to predict four weeks of sales in January 2023, it is logical to set the validation window size in each fold to four weeks. This ensures that all feature engineering, model selection, and hyperparameter tuning decisions are optimized for the precise task at hand.[9]

A properly configured time-aware validation framework acts as a virtual "time machine," allowing the competitor to simulate the final evaluation on the January 2023 data multiple times using only the historical 2022 data. The final leaderboard is determined by a single evaluation on a fixed, future period. During development, there is no access to this test data. The rolling-window validation creates multiple "pseudo-competitions" within the training set. For instance, one can use data up to October to predict November, and then data up to November to predict December. By averaging the model's performance across these folds, one obtains a stable and realistic estimate of its true generalization capability. Any feature, model, or hyperparameter that consistently improves the average score across these simulated evaluation periods is highly likely to perform well on the final, unseen test set. Without this rigorous protocol, development decisions are based on unreliable metrics, and the process is akin to navigating without a compass.

Section 2: High-Dimensional Feature Engineering: The True Differentiator

For modern machine learning models, particularly gradient boosting decision trees, the quality and richness of the feature set are the most significant determinants of predictive performance.[12] The core strategy is to transform the time series forecasting problem into a tabular, supervised learning problem. Each row in the transformed dataset represents a single

product-store-week to be predicted, and the columns (features) contain a comprehensive snapshot of all historical and contextual information available at the moment of prediction.[14] This process, known as feature engineering, is where domain knowledge and creativity have the greatest impact. The features detailed below are inspired by state-of-the-art approaches and winning solutions from major retail forecasting competitions, such as the M5.[12]

2.1 Temporal Feature Construction (The Calendar Context)

These features provide the model with the fundamental temporal context of the forecast date, allowing it to learn cyclical patterns and seasonal effects. They are derived directly from the date and require no historical sales data.

- **Standard Calendar Features:** These are the foundational time-based attributes. Examples include the day of the week, week of the year, month, day of the month, year, quarter, and day of the year.[12]
- **Event and Holiday Features:** Retail sales are heavily influenced by holidays and special events. This can be captured by creating binary indicator variables (flags) for specific dates corresponding to major holidays (e.g., Christmas, Thanksgiving) and promotional

events mentioned in the data.[12] Furthermore, it is often more powerful to create features that capture the lead-up to and aftermath of an event, such as days_until_next_holiday or days_since_last_promotion, as consumer behavior often changes in anticipation of and following these periods.[15]

2.2 Lag and Rolling Window Features (The Historical Context)

This is unequivocally the most powerful and critical class of features for time series forecasting with machine learning models. These features explicitly encode the past behavior of the time series into the model's input.

- **Lag Features:** These are the direct sales values from previous time steps. They are the primary mechanism for capturing autocorrelation and seasonality. The choice of lag periods is crucial and should be guided by expected business cycles. For instance, sales_lag_7 (for daily data) or sales_lag_1 (for weekly data) captures the value from the same day of the previous week, which is a strong predictor due to weekly shopping habits. Similarly, sales_lag_28 (4 weeks prior) and sales_lag_364 (approximately one year prior) capture monthly and annual seasonality, respectively.[13]
- **Rolling Window Statistics:** These features are aggregations of sales over a moving window of time. Their primary purpose is to smooth out short-term noise and capture recent trends, momentum, and volatility. A variety of statistical measures should be computed over different window sizes (e.g., 7, 14, 28, 90 days). Examples include rolling_mean_sales_14_days, rolling_std_dev_sales_28_days, rolling_max_sales_28_days, and rolling_skewness_sales_90_days. It is also highly effective to create these rolling features with a lag offset. For example, rolling_mean_sales_7_days_lag_28 would be the average sales in the 7-day window that ended 28 days ago. This provides the model with information about trends from different historical periods, not just the most recent one.[12]

The most potent features are often not static attributes but dynamic representations of relationships and momentum. While a feature like sales_lag_28 is informative, a feature constructed as a ratio, such as (sales_lag_7 / rolling_mean_sales_28_days), can be far more powerful. The raw lag value simply tells the model, "sales were X units last week." This lacks crucial context. Was X an unusually high or low value compared to the monthly trend? The ratio feature normalizes the recent performance against the recent trend. A value greater than 1.0 indicates that recent momentum is positive and sales are accelerating, while a value less than 1.0 indicates a slowdown. This relational feature encodes the *change* in behavior, which is often more predictive for tree-based models than the absolute behavior itself. Gradient boosting models excel at identifying critical thresholds in such ratio-based features, a technique that was central to many top-performing solutions in the M5 competition.[12]

2.3 Price and Promotion Dynamics

These features are designed to model the economic drivers of demand, such as price elasticity and the impact of marketing activities.

- **Price Features:** Raw price itself is a useful feature. However, more advanced price features can capture complex dynamics. Examples include price_momentum (current price divided by the average price over the last 30 days), price_change_vs_last_week, and price_relative_to_category (the item's price divided by the average price of all items in its parent category on that day). The latter feature is particularly useful for modeling substitution effects and product cannibalization, where a change in one item's price affects the sales of its competitors.[12]
- **Promotion Features:** A simple binary flag, promotion_active, is the most basic feature. More sophisticated features can include the duration of the promotion or the type of promotion if that information is available.

2.4 Hierarchical and Cross-Sectional Features (The Broader Context)

These features are a cornerstone of modern retail forecasting and provide a powerful method for regularizing predictions and addressing the cold start problem. They work by borrowing statistical strength from related, often more stable, time series.[3] Instead of relying solely on an item's own, potentially sparse and noisy, sales history, the model is given the context of its broader group.

- **Examples:** For a specific product-store combination, one can compute features like category_level_rolling_mean_sales_7_days, store_level_total_sales_lag_7, or state_level_dept_sales_lag_28. By providing the model with the recent trend of the product's parent category or its store's overall performance, we provide contextual signals that are often more stable and less intermittent than the item-level series itself. This helps the model make more reasonable predictions, especially for low-volume items.

Ultimately, feature engineering for time series should be viewed as an act of explicitly encoding domain knowledge and hypotheses into the data. Each feature should be backed by a clear, testable hypothesis about what drives sales. For example, the creation of a rolling_max_28_days feature is based on the hypothesis that peak sales events, such as a major promotion, have a lasting impact on consumer perception or inventory cycles that influence future sales. The creation of price_relative_to_category is based on the hypothesis that consumers make purchasing decisions not based on absolute price, but on the perceived relative value compared to substitute goods.[12] Framing the process in this way transforms it from a brute-force, "try everything" approach into a more scientific, hypothesis-driven methodology. This leads to a more interpretable, robust, and ultimately more powerful feature set.

The following table provides a structured compendium of the feature engineering strategy.

Feature Category	Feature Name / Example	Description	Hypothesis / Rationale
Temporal	week_of_year, day_of_week	Extracts calendar components from the date.	Captures weekly, monthly, and annual seasonal patterns in consumer behavior.
	days_until_holiday	Calculates the number of days until the next major holiday.	Models anticipatory purchasing behavior leading up to holidays.
Lag	sales_lag_28	The sales value from 28 days prior.	Captures monthly seasonality and autocorrelation in sales patterns.
	sales_lag_364	The sales value from approximately one year prior.	Captures strong annual seasonality, a dominant pattern in retail.
Rolling Window	rolling_mean_sales_14	The average sales over the preceding 14 days.	Smooths short-term noise to reveal the underlying recent trend.
	rolling_std_dev_sales_28	The standard deviation of sales over the preceding	Captures the recent volatility or stability of sales for

		28 days.	an item.
	rolling_max_sales_28	The maximum sales value in the preceding 28 days.	Models the impact of recent peak demand events (e.g., promotions).
Relational / Momentum	sales_lag_7 / rolling_mean_28	Ratio of last week's sales to the last month's average sales.	Normalizes recent performance to capture momentum (acceleration/deceleration).
Price	price_momentum	Current price divided by the average price over the last 30 days.	Captures recent price inflation or deflation for an item.
	price_relative_to_category	Item's price divided by the average price in its category.	Models price elasticity and substitution effects relative to competing products.
Hierarchical	category_sales_rolling_mean_14	The 14-day rolling average sales for the item's entire product category.	Provides a stable, less noisy signal of the trend for similar items.
	store_total_sales_lag_7	The total sales for the item's store one week prior.	Captures store-level traffic and performance, which affects all items within it.

Section 3: Core Predictive Modeling with Gradient

Boosting

With a robust validation framework and a rich feature set in place, the next step is to select, implement, and tune the core predictive models. For a large-scale, tabular forecasting problem of this nature, Gradient Boosting Decision Tree (GBDT) models represent the state of the art.[14] This section justifies the selection of LightGBM as the primary workhorse, details the strategy for its optimization, and presents a specialized two-stage modeling approach to handle the critical challenge of data intermittency.

3.1 Model Selection Rationale: LightGBM as the Primary Workhorse

The choice of modeling algorithm is a strategic decision that impacts not only accuracy but also the feasibility of experimentation within the competition's timeframe. While several GBDT implementations exist, the primary contenders are LightGBM and XGBoost.[16] A comparative analysis reveals that for this specific problem context, LightGBM holds several decisive advantages.[16]

- **Training Speed:** LightGBM is significantly faster than XGBoost, particularly on large datasets.[16] This is due to its use of a highly optimized histogram-based algorithm for finding splits and its novel Gradient-based One-Side Sampling (GOSS) technique, which focuses the learning process on data points with larger gradients (i.e., those that are poorly predicted).[19] This speed is not merely a convenience but a profound strategic asset. A forecasting competition is an iterative process of hypothesis testing through feature engineering and model tuning. A faster model allows for a vastly greater number of experiments to be conducted in the same amount of time, dramatically increasing the probability of discovering a breakthrough feature or an optimal set of hyperparameters. The choice between LightGBM and XGBoost is therefore not just a tactical one about algorithmic superiority, but a strategic one about maximizing the rate of experimentation and learning.
- **Memory Usage:** LightGBM is more memory-efficient due to its histogram-based approach, which buckets continuous features into discrete bins.[18] This is a critical advantage when working with the high-dimensional feature sets common in these competitions, which can easily run into millions or billions of data points.
- **Performance:** With proper tuning, the predictive accuracy of LightGBM and XGBoost is generally comparable.[19] Given the substantial gains in speed and efficiency, LightGBM emerges as the more pragmatic and effective choice for the primary modeling engine.

Other model families, while powerful in other contexts, are less suitable as the primary model for this challenge. Classical statistical models like SARIMA are ill-suited for large-scale, multivariate problems, as they typically require fitting a separate model for each of the thousands of individual time series, a computationally infeasible task.[25] Deep learning models like LSTMs can be highly effective but generally require vast amounts of data per series, are notoriously difficult and time-consuming to train and tune, and are less straightforward to apply to tabular feature sets compared to GBDTs.[28] These models may have a role in an advanced ensemble but are not the optimal choice for the core workhorse model.

The following table summarizes the comparative analysis of potential model candidates.

Model Candidate	Scalability (to thousands of series)	Feature Handling (Categorical, Numerical)	Training Speed	Intermittency Handling	Cold Start Capability	Overall Suitability
LightGBM	Excellent	Native support for both	Very Fast	Good (via two-stage model)	Good (via hierarchical features)	**Excellent**
XGBoost	Very Good	Native support for both	Moderate to Slow	Good (via two-stage model)	Good (via hierarchical features)	**Good**
LSTM	Moderate	Requires extensive preprocessing	Very Slow	Moderate	Moderate (requires transfer learning)	**Poor**
SARIMA	Poor	Exogenous variables only (SARIMAX)	Slow (per series)	Poor (struggles with high zero-counts)	Very Poor (no mechanism)	**Unsuitable**

3.2 Hyperparameter Tuning Strategy

Optimizing the performance of a LightGBM model requires a systematic approach to hyperparameter tuning. The most critical parameters to tune include num_leaves (controls tree complexity), learning_rate (controls the step size of the optimization), feature_fraction and bagging_fraction (for regularization), min_data_in_leaf (also for regularization), and n_estimators (the number of boosting rounds).[16]

The tuning process must be guided by the time-aware validation framework established in Section 1.2. The objective is to find the set of hyperparameters that maximizes the average performance score across all validation folds. Given the large search space, automated hyperparameter optimization libraries such as Optuna or Hyperopt are highly recommended. These tools use intelligent search algorithms (e.g., Bayesian optimization) to more efficiently explore the parameter space than a simple grid search. A common and effective strategy is to set a small learning_rate (e.g., 0.01-0.05) and use the early stopping mechanism within the LightGBM training function. Early stopping monitors the performance on a holdout set (in this case, the validation set of each fold) and automatically stops the training when the score ceases to improve for a specified number of rounds, thus finding the optimal n_estimators for that learning rate.

3.3 Modeling Intermittency: The Two-Stage Approach

As noted previously, retail sales data at the SKU level is often highly intermittent, with a large proportion of zero-sales days.[1] A standard regression model trained with a loss function like Mean Squared Error (MSE) will be pulled towards predicting small values for these items, resulting in biased and inaccurate forecasts for both the zero and non-zero cases.

A demonstrably superior method is to decouple the problem into two distinct stages, creating a "hurdle" model:

1. **Stage 1 (Classification Model):** A LightGBM classification model is trained on the entire dataset. The goal of this model is to predict the probability that the sales for a given product-store-week will be greater than zero, i.e., $P(\text{sales}>0)$. The target variable is binary (1 if sales > 0, 0 otherwise), and the model is optimized using a binary cross-entropy (logloss) objective function.
2. **Stage 2 (Regression Model):** A second LightGBM regression model is trained, but critically, it is trained *only on the subset of the data where sales were greater than zero*. This model's task is to predict the expected quantity of sales, *conditional on a sale occurring*, i.e., $E[\text{sales} \mid \text{sales}>0]$. This model can be optimized using a standard

regression objective like L1 (MAE) or L2 (MSE) loss.

The final forecast for any given product-store-week is then calculated as the product of the outputs from these two models:

$$\text{Final Forecast} = P(\text{sales} > 0) \times E[\text{sales} \mid \text{sales} > 0]$$

This two-stage approach explicitly models the zero-generation process separately from the quantity-generation process, which almost always leads to significant improvements in accuracy for intermittent data. Furthermore, this architecture provides an elegant and powerful solution to the competition's candidate generation problem. The output of the Stage 1 classifier—the probability of a non-zero sale—is precisely the signal needed to rank and select which combinations to include in the final 1,500,000-row submission file. This creates a unified framework where the solution to a core statistical modeling challenge (intermittency) directly provides the solution to a critical logistical constraint of the competition, avoiding the need to develop a separate, disconnected system for candidate selection.

Section 4: Advanced Ensemble Methods for Performance Maximization

While a single, well-tuned LightGBM model can achieve a high level of performance, top-tier solutions in competitive machine learning almost invariably rely on ensembling—the technique of combining predictions from multiple models to produce a final forecast that is more accurate and robust than any of its individual components.[31] The underlying principle is that different models, when properly constructed, will make different, uncorrelated errors. By combining their predictions, these errors tend to cancel each other out, leading to a more stable and accurate result.[32]

4.1 Constructing a Diverse Model Portfolio

The single most important criterion for a successful ensemble is **diversity** among the base models.[34] Simply training the same model architecture on the same data with different random seeds will produce highly correlated models and yield minimal ensemble benefit. True diversity must be actively engineered by forcing the models to learn different aspects of the data or to see the data from different perspectives. This can be achieved through several complementary strategies:

- **Algorithmic Diversity:** While LightGBM is the primary workhorse, the ensemble can be strengthened by including models based on different algorithms. For instance, an XGBoost model, despite being slower, may capture certain data patterns differently due to its level-wise tree growth. A much simpler model, like a Ridge or Lasso linear regression, can also be a valuable addition, as its linear nature provides a very different "view" of the data compared to the highly non-linear tree-based models.
- **Feature Diversity:** Train multiple LightGBM models on different subsets of the engineered features. For example, one model could be trained on a feature set emphasizing short-term dynamics (e.g., lags and rolling windows up to 28 days), while another focuses on long-term seasonality (e.g., lags of 364 days and features aggregated by month or quarter). This encourages specialization among the base models.
- **Data Diversity (Bagging):** Bootstrap Aggregating, or Bagging, involves training the same model architecture multiple times on different random subsamples of the training data (with replacement).[32] This is a classic and effective way to reduce the variance of a single model and generate a diverse set of predictors for the ensemble.
- **Target Transformation Diversity:** Models can be made diverse by training them to predict different transformations of the target variable. For example, one model could be trained to predict the raw sales quantity, while another is trained to predict the logarithm of sales (log(1+sales)). The logarithmic transformation often helps models better handle skewed distributions and focus on relative errors. Because their error distributions will be different, their combination in an ensemble can be particularly powerful.

4.2 Stacking and Blending Architecture

Simple ensembling methods, like taking the average or weighted average of model predictions, can be effective. However, a more powerful and sophisticated approach is **stacking**, also known as stacked generalization.[16] Stacking treats the predictions of the base models as features for a second-level model, which learns the optimal way to combine them. This can be conceptualized as a form of automated feature engineering where the "features" are abstract representations of data patterns as learned by a diverse set of expert models. A standard feature like

day_of_week encodes a simple, raw pattern. A base model, in contrast, learns a complex, non-linear function of hundreds of raw features, and its prediction is a single number that encapsulates this complex relationship. When these predictions are fed as features to a second-level model, that model is essentially learning from the "judgments" of the base models. If one base model is an expert on holiday effects and another is an expert on price dynamics, the second-level model acts as a "manager of experts," learning when to trust each one based on the context provided by their collective predictions.

A robust stacking architecture for this competition would be structured in two levels:

- **Level 0 (Base Models):** This level consists of the diverse portfolio of models developed in the previous step (e.g., multiple LightGBMs with different features, an XGBoost model, a linear model). To generate the training data for the next level without data leakage, the out-of-fold predictions from the time-aware cross-validation framework are used. For each fold, each base model is trained on the training portion and makes predictions on the validation portion. By concatenating the predictions from all folds, a complete set of unbiased, out-of-sample predictions is generated for the entire training dataset.
- **Level 1 (Meta-Model):** A relatively simple meta-model, such as a Ridge Regression or another lightweight LightGBM instance, is trained. Its input features are not the original feature set, but rather the out-of-fold predictions generated by the Level 0 models. The target variable remains the actual sales quantity. The goal of this meta-model is to learn the optimal, potentially non-linear, combination of the base model predictions to produce the final forecast.[38]

It is important to distinguish this stacking approach from a similar technique called blending. Blending also uses a meta-model but trains it on predictions made on a single, held-out validation set rather than on cross-validated predictions.[35] For time series problems where data is precious, the stacking approach integrated with the rolling-window cross-validation is generally more data-efficient and produces a more robust meta-model, as it is trained on predictions from the entire historical dataset.

Section 5: Strategic Candidate Generation and Final Submission

The final stage of the solution involves operationalizing the entire framework to produce the submission file, with a specific focus on addressing the competition's unique constraints: the 1,500,000 row limit and the potential for new, unseen items in the test set (the "cold start" problem). This section outlines the practical pipeline for generating the final, optimized predictions.

5.1 Identifying Active Product-Store Combinations: The Candidate Filter

As established, the submission row limit necessitates an intelligent filtering process to select which product-store-week combinations to forecast. The two-stage modeling architecture from Section 3.3 provides the ideal mechanism for this. The Stage 1 classification model, which predicts the probability of non-zero sales, serves as the core of the candidate generation system. The workflow is as follows:

1. **Generate a Universe of Plausible Candidates:** First, create a comprehensive list of all *plausible* product-store combinations for the forecast period (January 2023). A reasonable heuristic is to include any combination that recorded at least one sale in the last quarter of the training data (e.g., October-December 2022). This initial filtering step reduces the search space from all theoretically possible combinations to a more manageable set.
2. **Predict Sale Probabilities:** For each of these plausible combinations and for each of the four weeks in January 2023, use the trained Stage 1 classification model (or the ensemble of classifiers) to predict the probability of sales being greater than zero, P(sales>0).
3. **Apply a Threshold to Select Final Candidates:** Rank all the generated (product-store-week, probability) tuples in descending order of their predicted probability. The final candidate set is then selected by taking the top 1,500,000 entries from this ranked list.

This method ensures that the submission "budget" is allocated to the combinations with the highest likelihood of actually being sold. The probability threshold used for this selection is effectively a critical hyperparameter that governs a trade-off between precision and recall. A higher threshold would lead to a smaller, higher-confidence set of predictions (high precision, low recall), potentially missing some low-probability sales. A lower threshold would include more potential sales (higher recall) but risks filling the submission file with many predictions that turn out to be zero (lower precision). The optimal threshold should be tuned within the time-aware validation framework by evaluating which threshold value leads to the best final forecasting score on the validation sets, while respecting the row limit constraint. This directly connects the performance of the classification task to the ultimate regression-based evaluation metric of the competition.

5.2 Addressing the Cold Start Problem: Forecasting for the Unknown

The test set may contain new products or stores that have no sales history in the 2022 training data. This is the classic "cold start" problem in forecasting.[40] A model trained on 2022 data would have no features for these new entities and would likely predict zero. A more intelligent strategy is required, based on the principle of leveraging hierarchical information and metadata to infer behavior from similar, existing entities. This is fundamentally an act of data augmentation through logical inference, where a "synthetic" history is created for new items by assuming they will behave like their closest peers.[4]

A robust strategy for handling cold-start items involves using the data hierarchy:

- **Identify New Entities:** The first step is to identify which product-store combinations in the test set are new (i.e., have no corresponding sales history in the training data).
- **Hierarchical Inference:** For a new product, its forecast cannot be based on its own history. Instead, it must be based on the history of a related group. The product's metadata (e.g., its department, category, and price point) becomes crucial. The forecast can be proxied by the average sales pattern of a relevant aggregate group. For example, the initial forecast for a new brand of soda in a particular store could be based on the average weekly sales of all other sodas in that same store and department, potentially adjusted for its relative price compared to the category average.[4]
- **Level of Aggregation:** The key decision is choosing the right level of the hierarchy for the proxy. A new soda is more likely to behave like other sodas (a low level in the hierarchy) than like the average of all products in the store (a high level). The effectiveness of this strategy depends on the granularity of the product hierarchy and the quality of the available metadata.

5.3 Final Prediction Pipeline and Post-Processing

The end-to-end process for generating the final submission.csv file integrates all the components discussed in this report into a single, sequential pipeline:

1. **Load Trained Models:** Load all the trained models from the ensemble, including the Level 0 base models and the Level 1 meta-model for both the classification and regression stages.
2. **Generate Candidate Set:** Execute the candidate generation workflow from Section 5.1 to produce the final list of 1,500,000 product-store-week combinations for which predictions are required.
3. **Feature Engineering:** For each candidate in the set, construct the complete feature vector as detailed in Section 2. This involves retrieving historical sales, price data, and calendar information to compute all necessary lag, rolling, temporal, and hierarchical features.
4. **Generate Level 0 Predictions:** Feed the feature vectors for all candidates through each of the Level 0 base models in the ensemble. This will produce a set of out-of-sample predictions from each base model.
5. **Generate Final Forecasts:** Use the Level 0 predictions as input features for the Level 1 meta-model to compute the final, ensembled forecast for each candidate. Remember to

perform this for both the classification (P(sales > 0)) and regression (E[sales | sales > 0]) components and then multiply the results.

6. **Handle Cold-Start Items:** For any candidates identified as cold-start items, overwrite the model's prediction (which would be unreliable) with a forecast generated using the hierarchical inference strategy from Section 5.2.
7. **Post-Processing and Formatting:** Apply final cleaning steps to the predictions. Ensure all forecasted quantities are non-negative and are rounded to the nearest integer, as fractional sales are not possible. Format the final data into a two-column file (id, sales) according to the precise specifications provided in the competition guidelines.
8. **Final Validation:** As a last check, verify that the submission file contains the correct number of rows (not exceeding 1,500,000) and that all identifiers are valid before submission.

Conclusion

The framework presented in this report provides a comprehensive, state-of-the-art solution for a challenging retail sales forecasting competition. The proposed strategy moves beyond a simplistic regression approach to address the core complexities of the problem: the uncertainty of the test set composition, the strict submission limits, data intermittency, and the cold-start problem. The success of this solution is built upon several key strategic pillars:

- **A Dual-Problem Framing:** By explicitly separating the task into an activity forecasting (classification) problem and a volume forecasting (regression) problem, the solution directly tackles the challenges of intermittency and candidate generation in a unified and elegant manner.
- **Rigorous Time-Aware Validation:** The implementation of a rolling-origin cross-validation protocol is non-negotiable. It is the only reliable method for model evaluation, feature selection, and hyperparameter tuning that prevents temporal data leakage and provides a realistic estimate of performance on unseen future data.
- **Hypothesis-Driven Feature Engineering:** The performance of the chosen gradient boosting models is overwhelmingly dependent on the quality of the features. The emphasis on creating dynamic, relational, and hierarchical features—each backed by a clear hypothesis about consumer behavior—is what unlocks the model's predictive power.
- **Advanced Ensembling:** The use of a stacked ensemble architecture allows for the combination of diverse models, creating a final prediction that is more robust and accurate than any single model. This is a hallmark of top-performing solutions in competitive machine learning.
- **Strategic Constraint Management:** The solution directly incorporates the competition's constraints into its design. The candidate generation filter intelligently allocates the

limited submission rows, while the hierarchical inference strategy provides a principled approach to the cold-start problem.

Ultimately, achieving a winning rank in a competition of this nature is not the result of discovering a single "magic" algorithm. Rather, it is the outcome of a systematic and rigorous engineering process that combines domain knowledge, robust validation techniques, sophisticated feature creation, and advanced modeling strategies into a cohesive and powerful predictive pipeline.

Works cited

1. Introduction to the M5 forecasting competition Special Issue - ResearchGate, accessed September 20, 2025, https://www.researchgate.net/publication/361544132_Introduction_to_the_M5_forecasting_competition_Special_Issue
2. Introduction to the M5 forecasting competition Special Issue - PMC - PubMed Central, accessed September 20, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9232271/
3. 3 Oracle Retail Demand Forecasting Methods, accessed September 20, 2025, https://docs.oracle.com/cd/E12475_01/rdf/pdf/160/html/user_guide/output/fc_methods.htm
4. New take on hierarchical time series forecasting improves accuracy ..., accessed September 20, 2025, https://www.amazon.science/blog/new-take-on-hierarchical-time-series-forecasting-improves-accuracy
5. M5 Forecasting Accuracy Competition - Christophe Nicault, accessed September 20, 2025, https://www.christophenicault.com/post/m5_forecasting_accuracy/
6. Forecasting in retail - you should know these 5 examples - Blog - pacemaker.ai, accessed September 20, 2025, https://www.pacemaker.ai/en/blog/forecasting-in-retail-you-should-know-these-5-examples
7. 12 proven sales forecasting methods to predict revenue - Outreach, accessed September 20, 2025, https://www.outreach.io/resources/blog/forecasting-methods
8. Time Series Cross-Validation: Best Practices | Medium, accessed September 20, 2025, https://medium.com/@pacosun/respect-the-order-cross-validation-in-time-series-7d12beab79a1
9. Manually Perform Time Series Forecasting Using Ensembles of Boosted Regression Trees - MATLAB & Simulink - MathWorks, accessed September 20, 2025, https://www.mathworks.com/help/stats/time-series-forecasting-using-ensemble-of-boosted-regression-trees.html
10. Time Series Cross-Validation - GeeksforGeeks, accessed September 20, 2025, https://www.geeksforgeeks.org/machine-learning/time-series-cross-validation/

11. Time series cross-validation | R, accessed September 20, 2025, https://campus.datacamp.com/courses/forecasting-in-r/benchmark-methods-and-forecast-accuracy?ex=9
12. Machine Learning for Retail Sales Forecasting — Features ..., accessed September 20, 2025, https://s-saci95.medium.com/machine-learning-for-retail-sales-forecasting-features-engineering-4edfee7c9cbc
13. Practical Guide for Feature Engineering of Time Series Data - dotData, accessed September 20, 2025, https://dotdata.com/blog/practical-guide-for-feature-engineering-of-time-series-data/
14. Time Series forecasting feature creation/engineering - Data Science Stack Exchange, accessed September 20, 2025, https://datascience.stackexchange.com/questions/126967/time-series-forecasting-feature-creation-engineering
15. Feature Engineering on Demand Prediction - Halodoc Blog, accessed September 20, 2025, https://blogs.halodoc.io/wip-feature-engineering-on-demand-prediction/
16. LightGBM vs XGBoost for time series analysis - Kaggle, accessed September 20, 2025, https://www.kaggle.com/code/nowakjakub/lightgbm-vs-xgboost-for-time-series-analysis
17. xgboost vs lightgbm - Kaggle, accessed September 20, 2025, https://www.kaggle.com/code/ahm6644/xgboost-vs-lightgbm
18. What is Light GBM? Advantages & Disadvantages? Light GBM vs XGBoost? | Kaggle, accessed September 20, 2025, https://www.kaggle.com/general/264327
19. LightGBM vs XGBoost - Kaggle, accessed September 20, 2025, https://www.kaggle.com/discussions/questions-and-answers/175536
20. CatBoost vs. LightGBM vs. XGBoost | Towards Data Science, accessed September 20, 2025, https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db
21. Which algorithm takes the crown: Light GBM vs XGBOOST? - Analytics Vidhya, accessed September 20, 2025, https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
22. XGBoost vs LightGBM: How Are They Different - neptune.ai, accessed September 20, 2025, https://neptune.ai/blog/xgboost-vs-lightgbm
23. How to Use XGBoost and LGBM for Time Series Forecasting? - 365 Data Science, accessed September 20, 2025, https://365datascience.com/tutorials/python-tutorials/xgboost-lgbm/
24. Benchmarking state-of-the-art gradient boosting algorithms for classification - arXiv, accessed September 20, 2025, https://arxiv.org/pdf/2305.17094
25. SARIMA (Seasonal Autoregressive Integrated Moving Average ..., accessed September 20, 2025,

https://www.geeksforgeeks.org/machine-learning/sarima-seasonal-autoregressive-integrated-moving-average/
26. Using the SARIMA Model to Forecast the Fourth Global Wave of Cumulative Deaths from COVID-19: Evidence from 12 Hard-Hit Big Countries - MDPI, accessed September 20, 2025, https://www.mdpi.com/2225-1146/10/2/18
27. ARIMA & SARIMA: Real-World Time Series Forecasting - neptune.ai, accessed September 20, 2025, https://neptune.ai/blog/arima-sarima-real-world-time-series-forecasting-guide
28. deep learning - Time series prediction using ARIMA vs LSTM - Data ..., accessed September 20, 2025, https://datascience.stackexchange.com/questions/12721/time-series-prediction-using-arima-vs-lstm
29. Sales Forecasting Models: Comparison between ARIMA, LSTM and Prophet, accessed September 20, 2025, https://thescipub.com/pdf/jcssp.2024.1222.1230.pdf
30. Retail sales forecasting using LSTM and ARIMA-LSTM: A comparison with traditional econometric models and Artificial Neural Networks - Erasmus University Thesis Repository, accessed September 20, 2025, https://thesis.eur.nl/pub/53546/Cracan_Thesis.pdf
31. A Study on Ensemble Learning for Time Series Forecasting and the Need for Meta-Learning, accessed September 20, 2025, https://neclab.eu/technology/blog/a-study-on-ensemble-learning-for-time-series-forecasting-and-the-need-for-meta-learning
32. Ensemble Forecasting: The Difference Between Staying Ahead or Falling Behind - Logility, accessed September 20, 2025, https://www.logility.com/blog/ensemble-forecasting-the-difference-between-staying-ahead-or-falling-behind/
33. [2304.04308] Ensemble Modeling for Time Series Forecasting: an Adaptive Robust Optimization Approach - arXiv, accessed September 20, 2025, https://arxiv.org/abs/2304.04308
34. Ensembles for Time Series Forecasting - Proceedings of Machine Learning Research, accessed September 20, 2025, http://proceedings.mlr.press/v39/oliveira14.pdf
35. Stacking and Blending — An Intuitive Explanation | by Steven Yu | Medium, accessed September 20, 2025, https://medium.com/@stevenyu530_73989/stacking-and-blending-intuitive-explanation-of-advanced-ensemble-methods-46b295da413c
36. Stacking and Blending ensemble techniques - Kaggle, accessed September 20, 2025, https://www.kaggle.com/code/dhanishahahaha/stacking-and-blending-ensemble-techniques
37. The stacking ensemble method | Towards Data Science, accessed September 20, 2025, https://towardsdatascience.com/the-stacking-ensemble-method-984f5134463a/
38. Stacking and blending - Packt+ | Advance your knowledge in tech, accessed

September 20, 2025, https://www.packtpub.com/en-IN/product/modern-time-series-forecasting-with-python-second-edition-9781835883181/chapter/ensembling-and-stacking-11/section/stacking-and-blending-ch11lvl1sec76
39. Ensemble Machine Learning Technique: Blending & Stacking - YouTube, accessed September 20, 2025, https://www.youtube.com/watch?v=3VKK-6pG4IM
40. The Cold-Start Problem In Machine Learning Explained & 6 Mitigating Strategies, accessed September 20, 2025, https://spotintelligence.com/2024/02/08/cold-start-problem-machine-learning/
41. Tackling the Cold Start Challenge in Demand Forecasting - PyCon DE Berlin 2024, accessed September 20, 2025, https://www.youtube.com/watch?v=dbU4xdWwGEw
42. Overcoming the Cold Start Problem of CRM using a Probabilistic Machine Learning Approach - Columbia University, accessed September 20, 2025, http://www.columbia.edu/~np2506/papers/padilla_ascarza_coldstart.pdf
43. Cold-Start Demand Prediction for New Products: A Meta-Learning Approach on the M5 Competition Dataset - OpenReview, accessed September 20, 2025, https://openreview.net/pdf?id=n7KIwNy0zp
44. Global Time Series Model to project an aggregate series and the cold start problem. - Reddit, accessed September 20, 2025, https://www.reddit.com/r/datascience/comments/10hz0c2/global_time_series_model_to_project_an_aggregate/