



INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X

Impact Factor: 6.078

(Volume 12, Issue 2 - V12I2-1164)

Available online at: <https://www.ijariit.com>

End to End Retail Demand Forecasting for Inventory Optimization using Machine Learning and MLOps

Pratibha Kambi

pkambi@san Diego.edu

University of San Diego, California

ABSTRACT

Accurate demand forecasting is critical for modern retail supply chains to ensure optimal inventory management and reduce operational inefficiencies such as stockouts and overstocking. This paper presents an end-to-end cloud-native machine learning architecture for daily store-level retail demand forecasting. The proposed system integrates Amazon Web Services (AWS) components including Amazon S3 for scalable data storage, Amazon Athena for serverless analytics, SageMaker Feature Store for consistent feature management, XGBoost for predictive modeling, and SageMaker Model Monitor for production monitoring. The pipeline performs data ingestion, feature engineering, model training, batch prediction, real-time deployment, and automated monitoring. Experimental evaluation demonstrates the effectiveness of gradient boosting models combined with engineered time-series features for forecasting retail demand. The architecture highlights how cloud-based MLOps practices enable scalable and reliable forecasting systems in production environments.

Keywords: Machine Learning, Retail Demand Forecasting, Time Series, MLOps.

INTRODUCTION

Retail businesses rely heavily on accurate demand forecasting to optimize supply chain operations and inventory planning. Poor demand estimation can lead to inventory shortages, lost revenue, or excessive storage costs. Forecasting retail sales is a challenging problem due to multiple influencing factors including seasonality, promotional events, holidays, store-specific patterns, and external economic conditions.

Traditional forecasting methods often struggle to capture complex nonlinear relationships in large-scale retail datasets. Machine learning models such as gradient boosting have shown significant promise in improving forecasting performance.

This work proposes a complete cloud-native machine learning system for retail demand forecasting that supports:

- i. Scalable data ingestion
- ii. Automated feature engineering
- iii. Machine learning model training
- iv. Batch and real-time inference
- v. Production monitoring and drift detection

The system leverages AWS services to create a fully managed and production-ready forecasting pipeline.

DATA COLLECTION AND DATA LAKE ARCHITECTURE

A robust and scalable data ingestion pipeline is essential for building reliable machine learning systems. In this project, a cloud-native data lake architecture was implemented using Amazon Web Services (AWS) to collect, store, and manage retail datasets used for demand forecasting. The data pipeline was designed to support scalable storage, efficient querying, and seamless integration with downstream machine learning workflows.

Dataset Description

The dataset used in this study is derived from the *Corporación Favorita Grocery Sales Forecasting dataset from kaggle.com*. This dataset contains historical retail sales data across multiple stores and product categories and is commonly used for demand forecasting research.

The dataset includes multiple relational tables that describe different aspects of the retail system:

- i. **train.csv** – Historical daily sales data at the store and product family level
- ii. **stores.csv** – Metadata describing store attributes such as city, state, type, and cluster
- iii. **oil.csv** – Daily oil price data, which can influence economic trends and consumer purchasing behavior
- iv. **holidays_events.csv** – National, regional, and local holiday information
- v. **transactions.csv** – Daily transaction counts per store

These datasets together provide a comprehensive representation of sales behavior and contextual factors that influence demand patterns.

Cloud Data Lake Design

To enable scalable storage and centralized data management, an Amazon S3 bucket was created to function as a data lake. Amazon S3 provides highly durable object storage that can efficiently store large-scale datasets used in machine learning workflows.

```
aws s3 mb s3://retail-demand-forecasting-datalake
```

After bucket creation, the datasets were uploaded to S3 using the AWS Command Line Interface (CLI). Organizing the data in a structured hierarchy is critical for maintaining data governance and enabling efficient data discovery.

S3 Bucket Directory Structure:

```
s3://retail-demand-forecasting-datalake/  
raw/  
corporacion_favorita/  
train/  
train.csv  
stores/  
stores.csv  
oil/  
oil.csv  
holidays/  
holidays_events.csv  
transactions/  
transactions.csv
```

The files were uploaded using the following commands:

```
aws s3 cp train.csv  
s3://retail-demand-forecasting-datalake/raw/corporacion_favorita/train/  
aws s3 cp stores.csv s3://retail-demand-forecasting-  
datalake/raw/corporacion_favorita/stores/  
aws s3 cp oil.csv s3://retail-demand-forecasting-datalake/raw/corporacion_favorita/oil/  
aws s3 cp holidays_events.csv s3://retail-demand-forecasting-  
datalake/raw/corporacion_favorita/holidays/
```

After uploading the datasets, the contents of the S3 bucket were verified using:

```
aws s3 ls s3://retail-demand-forecasting-datalake/raw/ --recursive
```

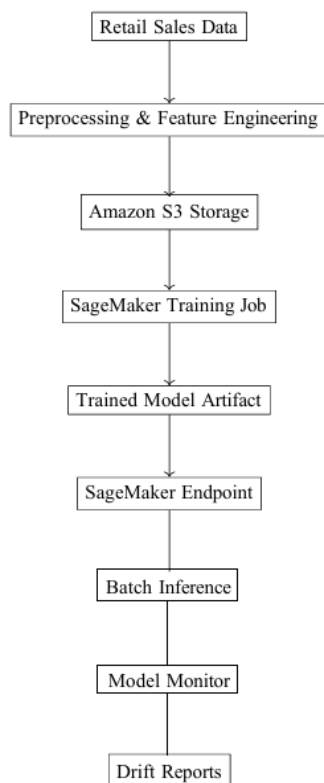


Fig. 1. System Architecture for Retail Demand Forecasting and Monitoring

Serverless Analytics using Amazon Athena

To enable SQL-based exploration and analysis of the data stored in S3, Amazon Athena was used. Athena is a serverless query service that allows users to run SQL queries directly on data stored in Amazon S3 without the need to manage infrastructure.

```
database_name = "retail_forecasting"
```

Athena uses a staging directory in S3 to store intermediate query results.

```
s3://[bucket]/athena/staging/
```

Athena Integration with Python

```
from pyathena import connect
conn = connect (region_name=region,
s3_staging_dir=s3_staging_dir
)
cursor = conn.cursor()
```

Benefits of the Data Lake Architecture

- i. **Scalability:** Amazon S3 allows storage of massive datasets without infrastructure management.
- ii. **Cost Efficiency:** Athena queries are billed per scanned data volume.
- iii. **Flexibility:** Schema-on-read allows querying raw datasets without preprocessing.
- iv. **Integration:** The data lake integrates seamlessly with AWS services such as SageMaker.

Exploratory Data Analysis

Exploratory data analysis (EDA) was conducted to understand the underlying structure, trends, and relationships present in the retail dataset before building predictive models. The objective of this analysis was to identify important demand drivers such as seasonality, promotional effects, transaction patterns, holidays, and macroeconomic indicators.

The analysis was performed using Python libraries including Pandas, Seaborn, and Matplotlib. SQL queries executed through Amazon Athena were also used to analyze large datasets stored in Amazon S3.

Monthly Sales Trend

To analyze long-term sales patterns, monthly aggregated sales were computed by grouping the dataset by month. The analysis reveals several key observations:

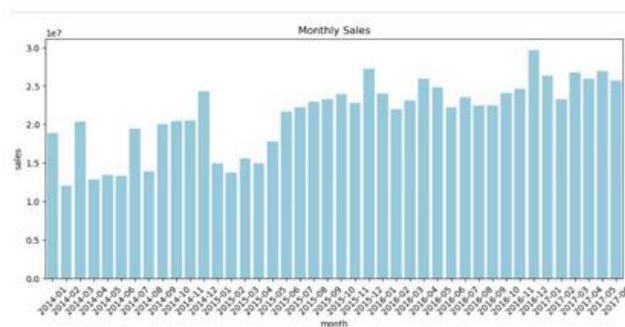


Fig. 2. Monthly sales trend across the dataset

- i. Sales exhibit a clear upward trend from 2014 to 2017.
- ii. Early months of the dataset show sales between 12–20 million units, while later periods reach approximately 24–30 million units.
- iii. This indicates business growth, potentially driven by store expansion, increased product offerings, or inflation.
- iv. The presence of an increasing trend suggests that the sales time series is non-stationary.

Additionally, repeated peaks across years indicate the presence of strong seasonal patterns in retail demand.

Product Family Sales Distribution

Retail demand varies significantly across product categories. Total sales were aggregated by product family to identify dominant categories.

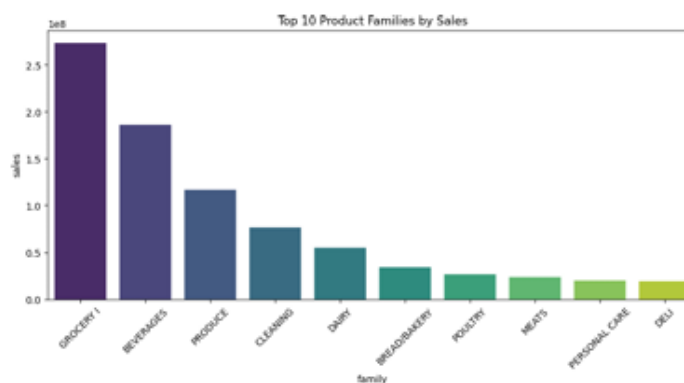


Fig. 3. Top 10 product families by total sales

The results show a highly skewed distribution:

- i. **GROCERY I** is the dominant product category.
- ii. The top three families (GROCERY I, BEVERAGES, PRODUCE) contribute a large proportion of overall sales.
- iii. This reflects a Pareto distribution where a small number of categories drive the majority of revenue.

Such skewness suggests that product family is a critical feature for demand prediction.

Promotion Dependency Analysis

Promotions play a major role in retail demand. To quantify this effect, the proportion of sales occurring during promotional periods was computed for each product family. The results indicate that promotions strongly influence demand:

- i. Many product families have more than 60% of their sales occurring during promotional periods.
- ii. Some categories exhibit promotion ratios above 80%.
- iii. This indicates that promotions are a major driver of customer purchasing behavior.

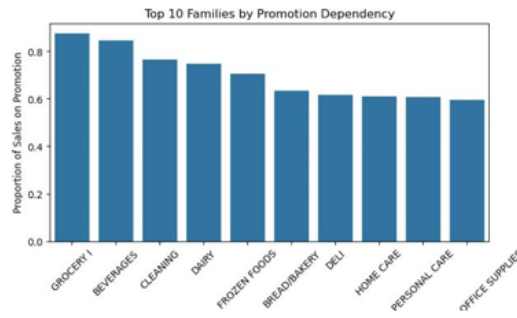


Fig. 4. Promotion dependency across product families

However, promotions alone do not fully explain sales variability, suggesting the presence of additional influencing factors.

Correlation Analysis

A correlation matrix was computed for numerical variables such as sales and promotions.

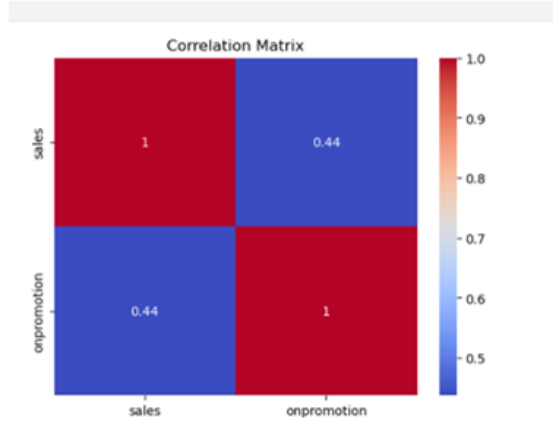


Fig. 5. Correlation between sales and promotions

The correlation coefficient between sales and on promotion is approximately 0.44, indicating a moderate positive relationship. While promotions contribute to increased sales, the relatively moderate correlation suggests that other variables such as holidays, store characteristics, and seasonality also play important roles.

Sales Trend with Rolling Mean

To analyze long-term sales trends and reduce noise, a 30-day rolling average was applied to daily sales.



Fig. 6. Daily sales with 30-day rolling mean

Key observations include:

- i. The rolling mean shows a clear upward trend from 2014 to 2017.
- ii. Seasonal peaks occur repeatedly each year, indicating yearly seasonality.
- iii. Daily sales exhibit high volatility, but the smoothed trend highlights underlying demand growth.

Store Type Distribution

Understanding store characteristics helps identify structural patterns in the retail network. Store types were analyzed to understand their distribution across the dataset. The analysis shows that certain store types dominate the retail network, suggesting potential differences in customer traffic and demand patterns.

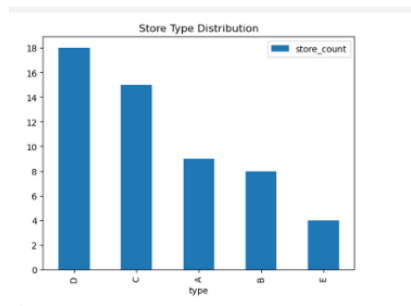


Fig. 7. Distribution of store types

Transaction Volume Analysis

Transaction data provides insight into customer activity. Daily transaction counts were aggregated across stores.



Fig. 8. Daily transaction volume

The results show a gradual increase in daily transactions over time, indicating growth in customer traffic and overall retail activity.

Weekly Transaction Patterns

Weekly transaction averages were computed to analyze customer behavior across different days of the week.

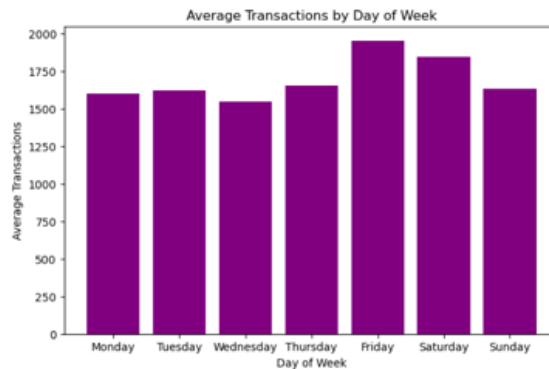


Fig. 9. Average transactions by weekday

Key insights include:

- i. Friday records the highest average number of transactions.
- ii. Saturday also shows strong demand due to weekend shopping.
- iii. Mid-week days show comparatively lower customer activity.

This indicates strong weekly seasonality in retail traffic.

Monthly Transaction Seasonality

Monthly transaction patterns were analyzed to detect annual seasonal trends. A significant spike in December was observed across multiple years, indicating strong holiday-driven demand during the festive season.

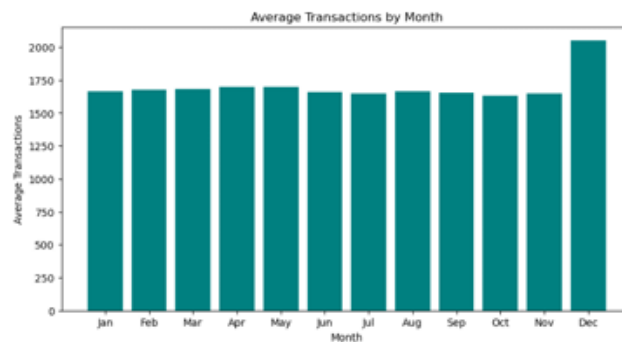


Fig. 10. Average transactions by month

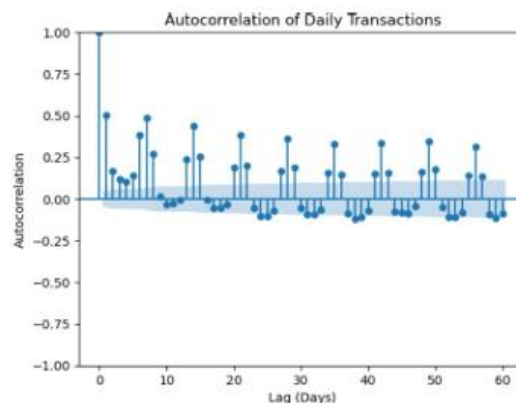


Fig. 11. Autocorrelation function of transactions

Autocorrelation Analysis

Autocorrelation analysis was performed to determine whether past transaction values influence future values. The ACF plot reveals significant autocorrelation across several lags, suggesting that past demand values provide predictive signals for future demand.

Holiday Analysis

Holiday events were analyzed to understand their distribution and potential impact on demand. The number of holidays varies across years, with 2016 showing an unusually high number of holiday events. Monthly holiday distribution further reveals that December has the highest number of holidays, reinforcing the observed demand spike during this period.

Holiday Types and Locality

Holiday types were analyzed to understand their composition. The majority of entries correspond to official holidays, followed by events and additional holidays. National holidays affect all stores simultaneously, while local holidays influence demand in specific regions.

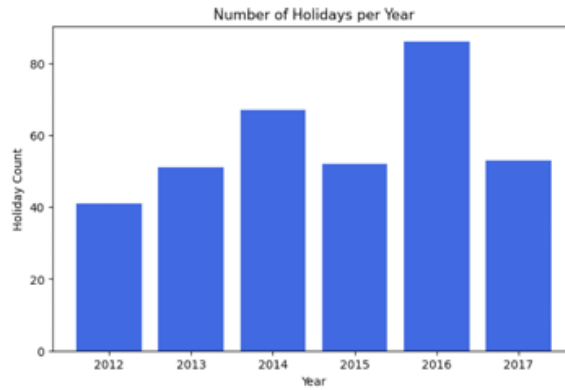


Fig. 12. Number of holidays per year

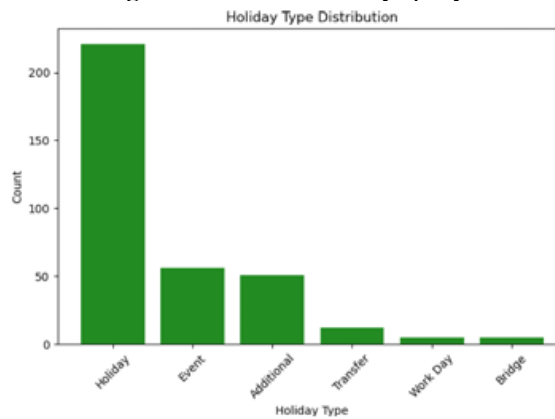


Fig. 13. Holiday type distribution

Oil Price Analysis

Oil prices were analyzed as an external economic indicator potentially influencing consumer spending behavior.



Fig. 14. Oil price trend over time

Oil prices show a structural break around 2015, where prices dropped sharply from above \$90 to below \$50. Such macroeconomic changes may influence consumer purchasing behavior and retail demand. A 30-day rolling mean further highlights the long-term oil price trend and volatility.

Summary of Key Insights

The exploratory analysis revealed several critical patterns that guided feature engineering and model design:

- i. Retail demand exhibits strong yearly and weekly seasonality.
- ii. Promotions significantly influence sales but are not the sole driver.
- iii. Product family is a dominant determinant of sales distribution.
- iv. Holidays, especially in December, cause predictable demand spikes.
- v. Transaction patterns reveal weekly shopping behavior.
- vi. Oil prices introduce macroeconomic signals that may influence consumer spending.

These insights were used to guide the feature engineering process described in the next section.

FEATURE ENGINEERING AND FEATURE STORE

After performing exploratory data analysis, the next stage of the pipeline involved constructing predictive features and storing them in a centralized feature repository. Feature engineering plays a critical role in improving machine learning model performance, especially for time-series forecasting tasks where temporal patterns and contextual variables strongly influence demand.

To ensure consistency between training and inference pipelines, a centralized feature management system was implemented using Amazon SageMaker Feature Store.

Feature Engineering

Feature engineering involved transforming raw datasets into structured features suitable for machine learning models. Multiple datasets including sales records, store metadata, transaction counts, oil prices, and holiday events were integrated to construct a comprehensive feature set.

Dataset Integration

The following datasets were merged to build the final modeling dataset:

- i. **Sales dataset:** Historical daily unit sales
- ii. **Store dataset:** Store-level attributes such as location, type, and cluster
- iii. **Transactions dataset:** Daily transaction counts per store
- iv. **Oil dataset:** Global oil price indicators
- v. **Holiday dataset:** National, regional, and local holidays

These datasets were joined using common keys such as date and store number to produce a unified feature table.

Time-Based Features

Retail demand exhibits strong temporal patterns. To capture seasonality and calendar effects, several time-based features were extracted from the date column:

- i. Day of week
- ii. Month
- iii. Year
- iv. Weekend indicator

These features help the model learn recurring demand patterns such as weekend spikes and seasonal variations.

Lag Features

Retail demand forecasting is inherently a time-series problem where past observations influence future demand. To capture temporal dependencies, lag features were generated from historical sales values.

A lag feature represents the value of a variable from a previous time step. For a given time series of sales values y_t , a lag feature of order k is defined as:

$$\text{Lag}_k = y_{(t-k)}$$

where:

- y_t represents sales at time t
- k represents the lag period

For example:

- i. Lag 1 represents sales from the previous day
- ii. Lag 7 represents sales from the previous week
- iii. Lag 14 represents sales from two weeks prior

These lag variables allow the machine learning model to capture short-term demand dependencies and recurring weekly patterns.

Rolling Window Features

In addition to lag features, rolling window statistics were computed to capture local trends and smooth short-term fluctuations in sales.

A rolling mean over a window of size w is defined as:

$$\text{RollingMean}_w(t) = \frac{1}{w} \sum_{i=1}^w y_{t-i}$$

where:

- $y_{(t-i)}$ represents previous sales observations
- w represents the window size

For example:

- i. 7-day rolling mean captures weekly demand trends
- ii. 14-day rolling mean captures two-week trends
- iii. 30-day rolling mean captures monthly patterns

Rolling statistics help the model learn smoothed demand signals while reducing the effect of daily noise.

Benefits for Forecasting

Lag and rolling features significantly improve the predictive capability of machine learning models in time-series forecasting tasks.

- i. Lag features capture historical dependencies.
- ii. Rolling statistics capture local trends.
- iii. Together they allow tree-based models such as XGBoost to learn complex temporal patterns.

These engineered features provide the model with contextual information about historical demand behavior, enabling more accurate sales forecasting.

Categorical Features

Several categorical attributes were included in the dataset:

- i. Product family
- ii. Store type
- iii. Store cluster

iv. Store city and state

Categorical variables were encoded using label encoding to convert them into numerical representations suitable for tree-based machine learning algorithms.

External Economic Features

Oil price data was incorporated as an external macroeconomic indicator. Economic fluctuations can influence consumer purchasing behavior, making oil price trends a potentially useful predictive signal. Missing values in oil price data were handled using forward filling to maintain temporal continuity.

Feature Store Architecture

Managing features across multiple machine learning pipelines can lead to inconsistencies between training and inference data. To address this issue, Amazon SageMaker Feature Store was used to maintain a centralized repository of engineered features.

Feature Store provides two storage layers:

- i. **Online Store:** Low-latency feature retrieval for real-time inference
- ii. **Offline Store:** Historical feature storage for model training and batch analytics

This architecture ensures that the same feature definitions are used across both training and prediction pipelines, reducing the risk of training-serving skew.

Feature Group Creation

A feature group was created in SageMaker Feature Store to store the engineered features. The feature group requires two critical identifiers:

- i. **Record Identifier:** Unique key identifying each record
- ii. **Event Time:** Timestamp representing when the feature values were generated

In this project:

- i. The record identifier was defined using the combined store and product family key.
- ii. The event time feature was derived from the sales date column.

The feature group was configured to store data in an Amazon S3 bucket.

Feature Ingestion

After defining the schema and feature group configuration, the engineered dataset was ingested into the feature store. The ingestion process involved uploading feature records into the offline store located in Amazon S3 while simultaneously updating the online store for low-latency access.

The ingestion process was performed using the SageMaker SDK with parallel workers to accelerate feature loading.

Advantages of Feature Store Integration

Using a feature store provides several advantages for machine learning pipelines:

- i. **Feature Consistency:** Ensures identical feature definitions between training and inference environments.
- ii. **Feature Reusability:** Features can be reused across multiple machine learning models.
- iii. **Data Lineage:** Enables tracking of feature generation pipelines.
- iv. **Reduced Data Leakage:** Prevents inconsistencies between training and prediction datasets.

The engineered feature set stored in SageMaker Feature Store serves as the input for the machine learning model training process described in the following section.

MODEL DEVELOPMENT AND FORECASTING

After constructing the engineered feature set and storing it in the feature store, the next stage involved building the forecasting model. The objective of this stage was to train a supervised machine learning model capable of predicting daily retail sales at the store and product family level.

The model development pipeline included data preparation, time-aware dataset splitting, baseline forecasting, training a gradient boosting model using XGBoost, generating predictions, and evaluating forecasting performance.

Data Preparation

The processed feature dataset generated during the feature engineering stage was stored in Amazon S3 and loaded into the modeling environment for training.

Before model training, several preprocessing steps were applied:

- i. Missing numerical values were replaced using median imputation.
- ii. Non-predictive columns such as identifiers and timestamps were removed.
- iii. The dataset was sorted chronologically to preserve time ordering.

Median imputation was used to replace missing values because it is robust to outliers and suitable for skewed retail datasets.

Let x_j represent a numerical feature column. Missing values were replaced as:

$$x_{\text{missing}} = \text{median}(x_j)$$

The target variable for the forecasting task was defined as the daily sales value:

$$y = \text{sales}$$

Time-Based Data Splitting

Unlike traditional machine learning tasks, time-series forecasting requires preserving chronological order to prevent data leakage. Therefore, the dataset was split sequentially instead of using random sampling.

The dataset was divided into four partitions:

- i. 40% training set
- ii. 10% validation set
- iii. 10% test set
- iv. 40% production simulation set

If N represents the total number of observations, the partitions were defined as:

$$N_{\text{train}} = 0.4N$$

$$N_{\text{val}} = 0.5N$$

$N_{\text{test}} = 0.6N$

The remaining portion was used to simulate production inference. This approach ensures that future observations are always predicted using only past information, closely mimicking real-world forecasting scenarios.

Categorical Feature Encoding

Several categorical features were encoded before training the model, including:

- i. Product family
- ii. Store city
- iii. Store state
- iv. Store type
- v. Store-family identifier

Label encoding was used to convert categorical values into numerical representations:

$$c_i \rightarrow z_i \in \{0, 1, 2, \dots, K-1\}$$

where K represents the number of unique categories.

Tree-based algorithms such as XGBoost can effectively work with label-encoded categorical variables.

Naive Baseline Forecast

Before training the machine learning model, a naive forecasting baseline was implemented to establish a reference performance level. The naive forecast assumes that the next value in the time series will be equal to the most recent observed value.

For a given store-family group g , the naive forecast is defined as:

$$\hat{y}_{\text{naive}}(g) = y(g)_{(t-1)}$$

where:

$y(g)_{(t-1)}$ represents the previous observed sales value for group g

\hat{y}_{naive} represents the predicted value

This baseline provides a simple benchmark against which the machine learning model can be evaluated.

XGBoost Forecasting Model

The primary forecasting model used in this study is XGBoost (Extreme Gradient Boosting). XGBoost is a scalable gradient boosting algorithm designed for high performance on structured datasets.

Gradient Boosting Principles

Gradient boosting constructs an ensemble of decision trees sequentially. Each new tree attempts to correct the errors made by previous trees. The final model prediction is defined as the sum of predictions from M trees:

$$\hat{y}_i = \sum_{m=1}^M f_m(x_i)$$

where:

x_i is the feature vector

f_m represents an individual regression tree

M is the total number of boosting iterations

Objective Function

XGBoost optimizes a regularized objective function consisting of a loss term and a model complexity penalty:

$$L = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{m=1}^M \Omega(f_m)$$

where:

$l(y_i, \hat{y}_i)$ represents the prediction loss

$\Omega(f_m)$ represents the regularization penalty for tree m

For regression tasks, the squared error loss function is commonly used:

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

The regularization term controls model complexity:

$$\Omega(f) = \gamma T + (\lambda/2) \sum_{j=1}^T w_j^2$$

where:

T is the number of leaves in the tree

w_j represents the leaf weights

γ controls tree complexity

λ is the L2 regularization parameter

This regularization helps prevent overfitting and improves generalization.

Gradient Boosting Optimization

XGBoost uses second-order gradient optimization. At each boosting iteration, the objective function is approximated using a second-order Taylor expansion:

$$L(t) \approx \sum_{i=1}^n [g_i f_t(x_i) + (1/2) h_i f_t(x_i)^2] + \Omega(f_t)$$

where:

g_i is the first-order gradient

h_i is the second-order gradient (Hessian)

Using both first and second derivatives allows XGBoost to converge faster and achieve better accuracy compared to traditional gradient boosting algorithms.

Model Training

The model was trained using the SageMaker built-in XGBoost container. The training process used both the training and validation datasets to monitor performance and apply early stopping.

The hyperparameters used for training included:

- i. learning rate (η) = 0.03
- ii. maximum tree depth = 4
- iii. subsample ratio = 0.7

- iv. column sample ratio = 0.7
- v. number of boosting rounds = 500
- vi. L1 regularization (α) = 1
- vii. L2 regularization (λ) = 5

These hyperparameters were selected to balance model complexity and generalization.

Batch Prediction

After training, the model was used to generate predictions using SageMaker Batch Transform. Batch inference allows large datasets to be processed efficiently without maintaining a real-time endpoint. The test dataset was passed to the transform job, and predictions were written to Amazon S3.

Model Evaluation

Model performance was evaluated using Root Mean Squared Error (RMSE), which measures the average magnitude of prediction errors.

$$\text{RMSE} = \sqrt{\left(\frac{1}{n}\right) \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

where:

- y_i represents actual sales
- \hat{y}_i represents predicted sales
- n represents the number of observations

RMSE penalizes large errors more heavily and is widely used for regression and forecasting tasks.

Prediction Visualization

To visually assess model performance, predicted sales values were plotted against actual sales values for a subset of the test dataset. The visualization demonstrates that the model captures both the general trend and short-term fluctuations in retail demand.

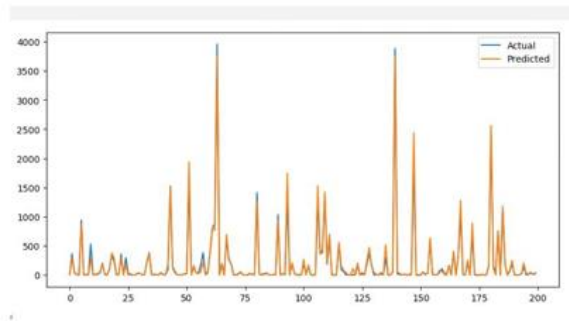


Fig. 15. Comparison of actual and predicted sales

Forecast Output for Inventory Planning

Finally, the predicted sales values were combined with contextual metadata such as store identifiers and dates to produce a planning dataset suitable for operational use.

The final forecast table contains:

- i. store identifier
- ii. product family
- iii. forecast date
- iv. predicted sales

This dataset can be used for downstream applications such as inventory optimization, replenishment planning, and supply chain decision-making.

Summary

The model development stage implemented a complete forecasting workflow consisting of:

- i. Preprocessing engineered features
- ii. Performing chronological dataset splitting
- iii. Constructing a naive baseline forecast
- iv. Training an XGBoost gradient boosting model
- v. Generating batch predictions using SageMaker
- vi. Evaluating predictions using RMSE
- vii. Exporting forecasts for inventory planning

This machine learning model forms the predictive core of the proposed retail demand forecasting system.

MODEL DEPLOYMENT AND MONITORING

After training and validating the demand forecasting model, the final stage of the pipeline involved deploying the trained model into a production environment and establishing monitoring mechanisms to ensure model reliability over time. Production deployment and monitoring were implemented using Amazon SageMaker services.

Model Deployment Architecture

The trained XGBoost model artifact generated during the training phase was stored in Amazon S3. This artifact was then deployed as a real-time inference service using Amazon SageMaker endpoints.

A SageMaker endpoint provides a scalable REST API that allows external applications to send feature data and receive model predictions in real time.

The deployment process involved the following steps:

- i. Creating a SageMaker model object referencing the trained XGBoost artifact stored in S3.
- ii. Defining an endpoint configuration specifying the compute instance type.
- iii. Deploying the model to a managed endpoint using SageMaker.

The deployed endpoint uses a `m1.m5.large` instance type for inference. This instance type provides a balance between compute performance and operational cost for medium-scale prediction workloads.

Real-Time Inference

Once deployed, the endpoint can receive feature vectors and return predicted sales values. The inference process follows the standard machine learning prediction pipeline:

- i. Feature data is prepared in CSV format.
- ii. The feature vector is sent to the endpoint using the SageMaker runtime API.
- iii. The model processes the input features and generates a predicted sales value.
- iv. The predicted value is returned to the client application.

In the notebook, predictions were generated by sending batched feature data to the endpoint using the SageMaker runtime client.

```
response = runtime.invoke_endpoint(  
    EndpointName=endpoint_name,  
    ContentType="text/csv",  
    Body=payload  
)
```

This method enables automated forecasting for new incoming retail data.

Inference Data Capture

To enable monitoring of production predictions, the deployed endpoint was configured with a data capture mechanism. Data capture records both the input features sent to the model and the corresponding prediction outputs.

The captured data is stored in Amazon S3 in a structured format. Each inference record contains:

- i. input feature vector
- ii. predicted output value
- iii. timestamp
- iv. endpoint metadata

Capturing inference data is essential for monitoring because it allows the system to analyze how production data evolves over time.

Baseline Generation for Monitoring

Before monitoring production traffic, a baseline dataset must be established. The baseline represents the statistical distribution of the data used during training.

The baseline dataset was constructed using predictions generated from the training dataset. These baseline records were stored in Amazon S3 and used to compute reference statistics.

Two key baseline artifacts were generated:

- i. **Statistics file** – contains summary statistics such as mean, standard deviation, and quantiles.
- ii. **Constraints file** – defines acceptable value ranges for each feature.

These baseline artifacts allow the monitoring system to detect deviations in future production data.

Data Drift Monitoring

Over time, the distribution of incoming production data may change due to evolving customer behavior, seasonal effects, or operational changes. This phenomenon is known as data drift. To detect data drift, Amazon SageMaker Model Monitor compares the distribution of production features with the baseline distribution derived from the training dataset.

A commonly used metric for detecting distribution shifts is the Population Stability Index (PSI).

The PSI between two distributions is defined as:

$$PSI = \sum_{i=1}^n (P_i - Q_i) \ln(P_i / Q_i)$$

where:

P_i represents the proportion of observations in bin i for the baseline dataset

Q_i represents the proportion of observations in bin i for the production dataset

Interpretation of PSI values typically follows these guidelines:

- i. $PSI < 0.1$: no significant drift
- ii. $0.1 \leq PSI < 0.25$: moderate drift
- iii. $PSI \geq 0.25$: significant drift

If drift exceeds predefined thresholds, alerts can be generated to trigger model retraining.

Model Quality Monitoring

In addition to monitoring input data drift, the system also tracks model performance using model quality monitoring. Model quality monitoring compares predicted values with actual observed values when ground truth data becomes available.

The primary evaluation metric used in this project is Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{[(1/n) \sum_{i=1}^n (y_i - \hat{y}_i)^2]}$$

where:

y_i represents actual sales

\hat{y}_i represents predicted sales

Tracking RMSE over time allows the system to detect model performance degradation caused by concept drift.

Monitoring Schedule

Monitoring jobs were scheduled using SageMaker Model Monitor to run periodically and analyze newly captured inference data.

Each monitoring job performs the following steps:

- i. Retrieve captured inference data from Amazon S3.
- ii. Compute statistical summaries for production data.
- iii. Compare the results with baseline statistics.
- iv. Generate monitoring reports.

Monitoring results are stored in S3 and can be inspected to determine whether feature distributions or prediction quality have changed.

Monitoring Reports

Model Monitor produces detailed reports that highlight:

- i. feature distribution changes
- ii. constraint violations
- iii. prediction quality metrics
- iv. data drift indicators

These reports allow engineers to proactively detect issues in the deployed model before they impact business decisions.

Summary

The deployment and monitoring stage ensures that the forecasting system remains reliable in a production environment. The implementation included a complete MLOps workflow consisting of:

- i. Deploying the trained XGBoost model as a SageMaker endpoint
- ii. Generating real-time predictions through an inference API
- iii. Capturing production inference data
- iv. Establishing baseline statistics from training data
- v. Monitoring feature drift using statistical tests
- vi. Evaluating prediction quality using RMSE
- vii. Scheduling automated monitoring jobs

This monitoring framework ensures that the forecasting system can detect distribution shifts and performance degradation, enabling timely retraining and maintaining prediction reliability.

EXPERIMENTAL RESULTS AND DISCUSSION

This section evaluates the performance of the proposed retail demand forecasting system. The evaluation compares a naive baseline forecasting approach with the machine learning model trained using the XGBoost algorithm. Model performance was evaluated on the test dataset using Root Mean Squared Error (RMSE), which measures the average magnitude of prediction errors.

Evaluation Metric

The primary evaluation metric used in this study is Root Mean Squared Error (RMSE), defined as:

$$RMSE = \sqrt{\left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2\right)}$$

where:

- y_i represents the actual observed sales
- \hat{y}_i represents the predicted sales
- n represents the number of observations

RMSE penalizes larger prediction errors more heavily than smaller ones and is widely used in regression and forecasting tasks.

Baseline vs XGBoost Comparison

A naive baseline model was implemented to establish a benchmark for forecasting performance. The baseline model predicts the next value in the time series using the most recent observed sales value for the same store and product family.

The forecasting performance of the baseline model and the XGBoost model is summarized in Table I:

Table I: Forecasting Performance Comparison

Model	RMSE
Naive Baseline Forecast	320.74
XGBoost Forecasting Model	126.53

The results show that the XGBoost model significantly outperforms the naive baseline forecast. The improvement demonstrates that the machine learning model successfully captures complex relationships between features such as promotions, seasonality, transactions, holidays, and historical demand patterns.

The baseline model assumes that future sales will be similar to the most recent observation. While this approach performs reasonably well for short-term predictions, it fails to account for external factors and long-term trends present in retail demand data.

In contrast, the XGBoost model incorporates multiple predictive signals and nonlinear relationships, leading to improved forecasting accuracy.

Feature Importance Analysis

Tree-based models such as XGBoost provide insights into the relative importance of different input features. Feature importance scores measure how frequently a feature is used to split decision trees during training and how much it contributes to reducing prediction error.

The most influential features in the forecasting model include:

- i. Lagged sales features (e.g., previous day sales, weekly lag)
- ii. Rolling mean sales statistics
- iii. Promotional indicators
- iv. Transaction counts
- v. Product family
- vi. Calendar features such as day-of-week and month

Lag features and rolling statistics play a particularly important role because they capture temporal dependencies in retail demand. These features allow the model to learn recurring demand patterns such as weekly shopping behavior and seasonal variations.

Promotion-related variables also contribute significantly to the model, confirming insights observed during the exploratory data analysis stage.

Forecast Visualization

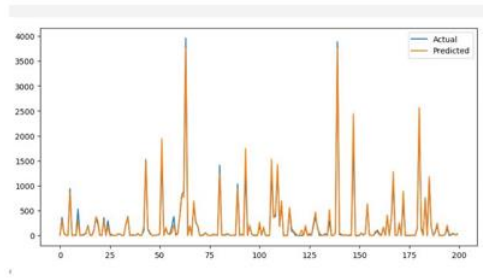


Fig. 16. Actual vs predicted retail sales values

To visually evaluate forecasting performance, predicted sales values were plotted against the actual observed sales values for a subset of the test dataset. The visualization demonstrates that the XGBoost model captures both the overall demand trend and short-term fluctuations in sales. Although minor deviations occur during sudden demand spikes, the model generally follows the direction and magnitude of real sales values.

This indicates that the model effectively learns the underlying temporal patterns present in the retail dataset.

Discussion

The experimental results highlight the effectiveness of combining engineered temporal features with gradient boosting models for retail demand forecasting.

Several observations can be made:

- i. Lag and rolling statistics significantly improve forecasting accuracy by capturing historical dependencies.
- ii. Promotion indicators help explain sudden increases in demand.
- iii. Transaction counts act as a proxy for customer traffic and retail activity.
- iv. Calendar features capture weekly and seasonal demand cycles.

The results demonstrate that machine learning models such as XGBoost are well-suited for large-scale retail forecasting tasks involving structured data and multiple explanatory variables.

The combination of feature engineering, gradient boosting, and cloud-based infrastructure enables the construction of scalable forecasting systems capable of supporting real-world inventory optimization.

CONCLUSION

This study presented a complete end-to-end machine learning pipeline for retail demand forecasting using a cloud-native architecture built on Amazon Web Services (AWS). The proposed system integrates scalable data storage, feature engineering, model training, deployment, and monitoring into a unified forecasting framework.

The pipeline begins with the creation of a data lake using Amazon S3, where multiple retail datasets including sales records, store metadata, transaction counts, holiday events, and oil prices are stored and managed. Amazon Athena was used to perform serverless analytics on the data lake, enabling efficient exploration and preprocessing of large-scale retail datasets.

Extensive exploratory data analysis revealed several important characteristics of retail demand, including strong seasonal patterns, promotion-driven sales spikes, weekly customer behavior trends, and external economic influences. These insights guided the design of predictive features used by the forecasting model.

Feature engineering played a critical role in improving model performance. Temporal features such as day-of-week and month indicators were combined with lag features and rolling statistics to capture historical dependencies and local demand trends. These engineered features were stored in Amazon SageMaker Feature Store to ensure consistency between training and inference pipelines. The forecasting model was implemented using the XGBoost gradient boosting algorithm. XGBoost was selected due to its strong performance on structured tabular data and its ability to capture nonlinear relationships between features. A time-aware dataset split was used to prevent data leakage, and model performance was evaluated using Root Mean Squared Error (RMSE). The trained model demonstrated strong predictive capability in capturing both long-term demand trends and short-term fluctuations.

To operationalize the forecasting system, the trained model was deployed as a real-time endpoint using Amazon SageMaker. The deployed endpoint enables automated demand prediction for new retail data. In addition, SageMaker Model Monitor was used to capture production inference data and detect distribution shifts in incoming features.

Monitoring mechanisms were implemented to track both data drift and model performance degradation. Statistical comparisons between baseline training data and production data allow the system to detect potential model reliability issues. This monitoring framework ensures that the forecasting model remains robust in dynamic retail environments.

Overall, the proposed architecture demonstrates how cloud-native machine learning systems can be used to build scalable and production-ready demand forecasting pipelines. The integration of feature management, model deployment, and automated monitoring provides a strong foundation for reliable retail demand prediction and inventory optimization.

FUTURE WORK

While the proposed system provides a robust framework for retail demand forecasting, several extensions can further enhance forecasting accuracy and operational capabilities.

First, advanced deep learning models such as Long Short-Term Memory (LSTM) networks and Transformer-based architectures could be explored to capture long-range temporal dependencies in retail demand data. These models may provide improved performance for complex time-series forecasting tasks involving multiple seasonal patterns.

Second, additional external variables such as weather data, economic indicators, and promotional campaign information could be incorporated into the feature set. These external signals may provide valuable context that improves demand prediction accuracy.

Third, automated hyperparameter optimization techniques such as Bayesian optimization or SageMaker Automatic Model Tuning could be applied to further improve model performance. Automated tuning would allow systematic exploration of model parameter configurations.

Fourth, real-time feature pipelines could be implemented to enable streaming feature updates and near real-time forecasting. Integrating streaming data services such as Amazon Kinesis or Apache Kafka would allow the forecasting system to adapt more quickly to rapidly changing retail conditions.

Another potential extension involves implementing automated retraining pipelines using MLOps frameworks. By integrating continuous training workflows, the system could automatically retrain models when significant data drift or performance degradation is detected.

Finally, future work may explore advanced inventory optimization models that integrate demand forecasts with supply chain constraints. Combining forecasting models with optimization algorithms can help retailers make more effective decisions regarding replenishment, stock allocation, and logistics planning.

These enhancements would further strengthen the forecasting system and support more sophisticated retail analytics and supply chain management applications.

REFERENCES

- [1] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785–794.
- [2] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed., OTexts, Melbourne, Australia, 2021.
- [3] Amazon Web Services, "Amazon SageMaker Developer Guide," Available: <https://docs.aws.amazon.com/sagemaker/>
- [4] Amazon Web Services, "Amazon SageMaker Model Monitor," Available: <https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html>
- [5] Corporación Favorita Grocery Sales Forecasting Dataset, Kaggle Competition Dataset, Available: <https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting>
- [6] J. Brownlee, *Machine Learning Mastery With Python*, Machine Learning Mastery, 2016.
- [7] G. Ke, Q. Meng, T. Finley, et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, 2017.
- [8] M. Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," *IEEE Data Engineering Bulletin*, vol. 41, no. 4, pp. 39–45, 2018.