Sales Forecasting Using XGBoost with Bayesian Hyperparameter Optimization — Full Technical Report

1. Introduction

This project aims to deliver an accurate and scalable sales forecasting model using historical sales data. Leveraging machine learning, particularly XGBoost, combined with Bayesian hyperparameter optimization, the objective is to predict daily sales by learning temporal patterns, seasonality, and the effects of domain-specific factors like holidays and promotions.

2. Data preprocessing

The dataset includes sales and profit data over several years, with important columns such as Order Date, Ship Date, Sales, Profit, and categorical variables like Sub-Category.

- Dates (Order Date, Ship Date) were parsed into datetime objects to enable time series analysis.
- Initial exploratory data analysis (EDA) was performed, including distribution plots, boxplots for outliers, and correlation heatmaps, to understand variable relationships and data quality.
- Missing values and duplicates were checked, confirming data consistency.

3. Exploratory Data Analysis (EDA)

EDA reveals trends, outliers, and data distributions, guiding modeling strategies.

Key Visual Analyses:

- **Histogram + KDE** of Profit → Skewness, variability
- **Boxplots** of Sales and Profit → Outlier detection
- **Scatterplot: Sales vs. Profit** → Correlation and profitability patterns

- **Correlation heatmap** → Reveals interdependence between numerical features

Aggregations:

- **Sales by Sub-Category** → Highlights profitable categories
- **Discount vs Profit scatter** → Tests hypothesis: "Do discounts hurt profit?"
- **Monthly trend with moving average** → Seasonality detection
- **Holiday sales vs Non-holiday sales** → Confirms external temporal drivers

This thorough EDA reveals **patterns not obvious in raw data**, forming the foundation for **intelligent feature engineering**.

Feature Creation:

Feature creation is the step where we generate new informative columns based on the original data. The goal is to make hidden patterns more visible for the model, especially when working with time-based Raw sales data, like having just an 'Order Date' column, usually doesn't provide enough context for the model to learn meaningful patterns. To improve this, we create features based on time, previous sales behavior, and special events. This allows the model to better understand:

• When exactly did the sale take place? (day, month, year, or weekend)

• Was the order made on a holiday?

• What were the sales like before and after that day?

Features We Created:

1. Time-Based Features

Purpose	Feature
Which day of the week the order	Day of Week

Numeric month (1 to 12)	Month
Day of the month (1 to 31)	Day
Year of the order	Year
Which quarter of the year (Q1 to Q4)	Quarter

2. Cyclical Transformations

Description	Feature
Converts month into a circular pattern (Dec and Jan become close)	Month
Converts day of the month into a repeating cycle (e.g., 1 and 31 are near)	Day

3.Lag Features

Meaning	Feature
1 if the order date was a public holiday, 0 otherwise	Holiday
1 if the order was on Saturday or Sunday, 0 otherwise	Weekend

Why Is This Important?

Without feature creation, the model would only see one raw date column.
By expanding it into structured features, we make hidden sales behavior visible.
This improves model performance and gives us a better understanding of seasonality, customer habits, and sales trends

Model Selection: XGBoost Regressor

XGBoost (Extreme Gradient Boosting) was selected for the regression task due to its proven superiority in handling tabular data, its robustness to feature scaling, and its ability to model nonlinear relationships and interactions effectively.

Compared to traditional models such as:

- Linear Regression: Limited to linear relationships and unable to capture complex patterns.
- Random Forest: While effective, it can be less efficient and slower in prediction compared to XGBoost.
- ARIMA and other Time Series Models: These require strong assumptions about stationarity and are less flexible when handling multiple covariates or complex feature interactions.

XGBoost combines gradient boosting with regularization techniques, reducing overfitting and increasing predictive accuracy.

Hyperparameter Optimization Using Bayesian Search:

Instead of manual tuning or grid/random search, Bayesian optimization via BayesSearchCV was employed to efficiently explore the hyperparameter space:

- Parameters tuned included n_estimators (2000–3000), early_stopping_rounds (5–15), learning_rate (0.01–0.1), and max_depth (2–5).
- Bayesian optimization models the performance landscape and chooses hyperparameters that maximize model performance with fewer iterations, reducing computational cost.

This approach outperforms random search by focusing on promising regions in the hyperparameter space, resulting in better tuning with fewer evaluations.

Training and Validation

- The dataset was split by date, using all data before 2017 for training, and data from 2017 onwards for testing, simulating a realistic forecasting scenario.
- The model was trained on engineered features and the target variable Sales.
- Early stopping was used to prevent overfitting, based on evaluation on the test set.

Evaluation Metrics

To comprehensively evaluate the model, multiple error metrics were computed:

- Root Mean Squared Error (RMSE): Measures the standard deviation of prediction errors, sensitive to large errors.
- Mean Absolute Error (MAE): Provides average magnitude of errors in the same units as the target.
- Mean Absolute Percentage Error (MAPE): Offers a relative error percentage, useful for interpretability.

The model achieved strong performance on all these metrics, indicating reliable predictive capability.

Feature Importance Analysis:

- Feature importance scores extracted from the XGBoost model reveal which features contributed most to predictions.
- Temporal features such as lagged sales and cyclical date features ranked highly, validating the effectiveness of feature engineering.

Comparison with Alternative Techniques :

Technique	Advantages	Limitations	Why XGBoost is Preferred
Linear Regression	Simple, interpretable	Cannot capture nonlinearities	Sales data has complex seasonal/nonlinear patterns
Random Forest	Handles nonlinearities, less tuning	Can be slower and less accurate than boosting	XGBoost is faster and often more accurate
ARIMA / Time Series	Specifically for sequential data	Requires stationary data, limited features	Our data is multivariate with holidays and lag features
Neural Networks	Powerful for complex patterns	Needs large data, tuning complexity	XGBoost performs well with structured data and less tuning
XGBoost	High accuracy, regularization, fast	Some complexity, requires tuning	Balances accuracy, interpretability, and efficiency

Model Deployment:

- The finalized model is saved using joblib for future use and deployment.
- The pipeline is modular and extensible, allowing retraining with new data or adjustment of features.