# **Milestone 3: Machine Learning Model Development and Evaluation**

## **1. Objective**

Now that the dataset was fully cleaned and packed with meaningful features, the next logical step was building a machine learning model that could accurately predict daily product sales. Our main goal in this milestone was to choose the right algorithms, train them on our enriched data, and evaluate how well they performed.

We tried to approach the modeling phase with both accuracy and practicality in mind. We wanted models that not only predicted well but also made sense in a business context — where interpretability, speed, and robustness matter.

## **2. Setting Up the Problem**

We approached this as a **regression problem** since we were predicting a continuous variable — sales — for specific item-store-date combinations.

Instead of randomly splitting the data (which would create leakage in time series), we made sure to split it **chronologically**. The training set included older data, while the test set included more recent dates. This way, we simulated real-world forecasting: training the model on the past and using it to predict the future.

## **3. Data Splitting**

**Train and Evaluation Set Split (by Year)**

* **train\_set** includes data from 2013 to 2016.
* **eval\_set** includes data from 2017, which is reserved for model evaluation.

This year-based split mimics a real-world scenario of training on historical data and testing on future unseen data.

## **4. Feature and Target Separation**

X\_train, X\_test: Feature sets

y\_train, y\_test: Target variable (log\_sales)

## **5. Modeling**

### **1. Linear Regression (Baseline)**

We started with a basic linear regression model just to get a sense of how the features behave. It was fast and easy to interpret, but not great at capturing the complex, non-linear relationships in our data.

### **2. Random Forest Regressor**

Next, we tried a Random Forest, which is an ensemble of decision trees. This model handles non-linearity and interactions between features very well and gives us access to feature importance plots. It worked significantly better than linear regression.

### **3. XGBoost Regressor**

We also trained an XGBoost model, which is a more advanced tree-based method using gradient boosting. It’s known for being highly effective with structured/tabular data and tends to perform better than Random Forest in many real-world cases.

### **4. LightGBM Regressor**

Finally, we used LightGBM, a fast, distributed, and efficient implementation of gradient boosting developed by Microsoft. It supports histogram-based learning and categorical features, making it especially powerful for large datasets.

## 

## **6. Evaluation Metrics**

To measure performance, we used several regression metrics:

|  |  |
| --- | --- |
| **Metric** | **Why It Matters** |
| **MAE (Mean Absolute Error)** | Easy to interpret — tells us how far off, on average, our predictions were |
| **MSE (Mean Squared Error)** | Penalizes bigger mistakes more heavily |
| **RMSE (Root Mean Squared Error)** | More sensitive to large errors, but still in the same units as the target |
| **R² Score** | Shows how much of the variation in sales is explained by the model |

These metrics gave us a well-rounded picture of how well each model was performing.

## **7. Results and Observations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **MAE** | **RMSE** | **R² Score** | **RMSLE** | **MSE** |
| Linear Regression | 1.57 | 1.88 | 0.5 | 0.65 | 3.54 |
| Random Forest | 0.43 | 0.64 | 0.94 | 0.26 | 0.42 |
| LGBM | 0.42 | 0.57 | 0.95 | 0.25 | 0.42 |
| **XGBoost** | **0.39** | **0.23** | **0.96** | **0.23** | **0.28** |

**XGBoost clearly performed the best** in our testing. It handled complex patterns like weekly seasonality, promotional effects, and holiday spikes much more accurately than the other models. The **Random Forest** wasn’t far behind and still gave us useful insights, especially through its feature importance rankings.

## **8. Hyperparameter Tuning**

The **XGBoost** model was fine-tuned using a **randomized search** to identify the optimal combination of hyperparameters that would maximize its performance. The search led to the following best hyperparameters:

**Number of Estimators (Trees)**: 200

**Maximum Depth**: 9

**Learning Rate**: 0.1

**Subsample**: 0.8

**Colsample Bytree**: 0.9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **MAE** | **RMSE** | **R² Score** | **RMSLE** | **MSE** |
| **XGBoost** | **0.2** | **0.44** | **0.97** | **0.2** | **0.31** |

## **8. Feature Importance Insights**

We used built-in feature importance tools from both Random Forest and XGBoost to understand which features had the most impact. The top features were:

* price
* family
* onpromotion
* dcoilwtico
* store\_nbr and store\_type (to a lesser extent)

This confirmed what we’d seen earlier in the EDA: recent performance and short-term trends are strong predictors of future demand.

## **9. Error Behavior**

We also looked at how the models behaved when they made mistakes. We found:

* Good accuracy for steady products (things that sell in stable amounts day-to-day)
* Underprediction during **holidays and major promotions**, likely because those events are harder to model unless they appear frequently in the training data
* The residuals are roughly centered around zero, indicating no systematic over- or underestimation. However, the increasing spread of residuals at higher predicted values suggests heteroscedasticity, meaning the model's prediction error increases with the magnitude of predictions.

## **10. Conclusion**

In this milestone, we successfully trained and tested several machine learning models on our feature-rich dataset. The results showed that **XGBoost was the best-performing model**, balancing accuracy and flexibility.

The models captured important signals like lag effects, short-term trends, and promotional boosts. More importantly, they provided a solid foundation for making actual sales forecasts that businesses could use for planning inventory, scheduling staff, and timing promotions.

The next step was to make these predictions more accessible — which we tackled in the deployment and web integration phase.