Store Sales Forecasting Using Time-Series Feature Engineering and Machine Learning Models

Technical Project Report

Sakina Saifuddin

Abstract

This project develops a supervised machine learning pipeline for store-level sales forecasting using historical sales, promotions, oil prices, transactions, holiday indicators, and store metadata. The workflow follows a complete predictive analytics process: data loading, relational merging, exploratory data analysis, missing-value treatment, time-series feature engineering, time-aware validation, model training, and final model comparison. Because sales are highly right-skewed and include many zero-sales observations, the target variable is transformed using $\log(1 + y)$. Three models are evaluated: Linear Regression, Random Forest, and LightGBM. Model performance is measured using root mean squared error (RMSE) on the log-transformed target. The final comparison shows that LightGBM achieves the lowest final test RMSE of 0.4516, followed by Random Forest with 0.4845, while Linear Regression performs substantially worse with 1.3541. The results indicate that nonlinear tree-based models are better suited for the complex, seasonal, promotional, and store-specific patterns in retail demand forecasting.

Contents

1 Introduction **2**

2 Dataset and Problem Formulation **2**

2.1 Raw Data Sources ... 2

2.2 Prediction Target ... 3

3 Data Loading and Merging **3**

4 Exploratory Data Analysis **3**

4.1 Sales Distribution ... 3

4.2 Temporal Sales Patterns ... 4

4.3 Store, Product Family, and Weekday Effects ... 5

4.4 Promotion and Transaction Effects ... 6

5 Feature Engineering **7**

5.1 Missing-Value Treatment ... 7

5.2 Calendar Features ... 7

5.3 Lag Features ... 7

5.4 Rolling Mean Features ... 8

5.5 Target Transformation and Final Feature Matrix ... 8

6 Modeling Methodology **8**
6.1 Time-Aware Train/Validation Design ... 8
6.2 Evaluation Metric ... 9
6.3 Linear Regression Baseline ... 9
6.4 Random Forest ... 9
6.5 LightGBM ... 9

7 Experimental Results **9**
7.1 K-Fold Cross-Validation Results ... 9
7.2 Final Holdout Test Results ... 9
7.3 Prediction Examples on Original Sales Scale ... 12

8 Discussion **12**

9 Limitations and Future Work **12**

10 Conclusion **12**

A Important Code Snippets **13**
A.1 Data Loading and Merging ... 13
A.2 Feature Engineering ... 13
A.3 Time-Series Cross-Validation ... 14
A.4 Random Forest Model ... 14
A.5 LightGBM Model ... 14
A.6 Inverse Transformation of Predictions ... 14

1 Introduction

Retail sales forecasting is an important predictive analytics problem because inventory planning, replenishment, promotion design, and staffing decisions depend on accurate estimates of future demand. In this project, the objective is to predict sales at the store–product-family–date level using a structured feature set that combines sales history with external and operational drivers.

The project is organized as a reproducible notebook-based pipeline. The original raw files are first merged into a single analytical dataset. Exploratory data analysis is then used to understand the distribution of sales and the relationship between sales and important predictors. Next, the dataset is transformed into a model-ready time-series feature matrix. Finally, multiple models are evaluated using time-series cross-validation and a final unseen holdout set.

2 Dataset and Problem Formulation

2.1 Raw Data Sources

The project uses the following tables in the data loading and merging stage:

- train.csv: historical sales observations with id, date, store nbr, family, sales, and onpromotion.
- stores.csv: store-level metadata including city, state, store type, and cluster.
- oil.csv: daily oil price variable dcoilwtico.
- transactions.csv: daily store-level transaction counts.
- holidays events.csv: holiday type, locale, and transferred-holiday information.

After merging, the training dataset contains 3,054,348 rows and 15 columns. The merged fields include the target variable sales, promotion counts, store metadata, oil prices, transactions, and holiday attributes.

2.2 Prediction Target

Let $y_{i,t}$ denote the observed sales for product-family/store observation i at date t. The modeling target is the log-transformed sales value

$$z_{i,t} = \log(1 + y_{i,t}). \tag{1}$$

The inverse transformation used to return predictions to the original sales scale is

$$\hat{y}_{i,t} = \exp(\hat{z}_{i,t}) - 1. \tag{2}$$

This transformation is appropriate because the sales distribution is highly right-skewed and contains extreme values.

3 Data Loading and Merging

The raw files are loaded using pandas, date columns are converted to datetime format, and the tables are merged using the date and store identifiers. Store metadata is merged by store nbr. Oil data is merged by date. Transactions are merged by both date and store nbr. Holiday features are merged by date.

The merge process produces missing values in variables that are not observed for every date or every store. In the initial merged dataset, the main missing columns are holiday fields, oil prices, and transactions. Holiday missingness means that a day is not a holiday; therefore, these values are later encoded as None. Oil prices are time-based external variables and are filled using forward/backward time-based imputation. Transactions are operational variables and missing transaction counts are set to zero.

4 Exploratory Data Analysis

4.1 Sales Distribution

Figure 1 shows that the original sales distribution is extremely right-skewed. Most observations have low sales, while a relatively small number of observations have very large sales values.

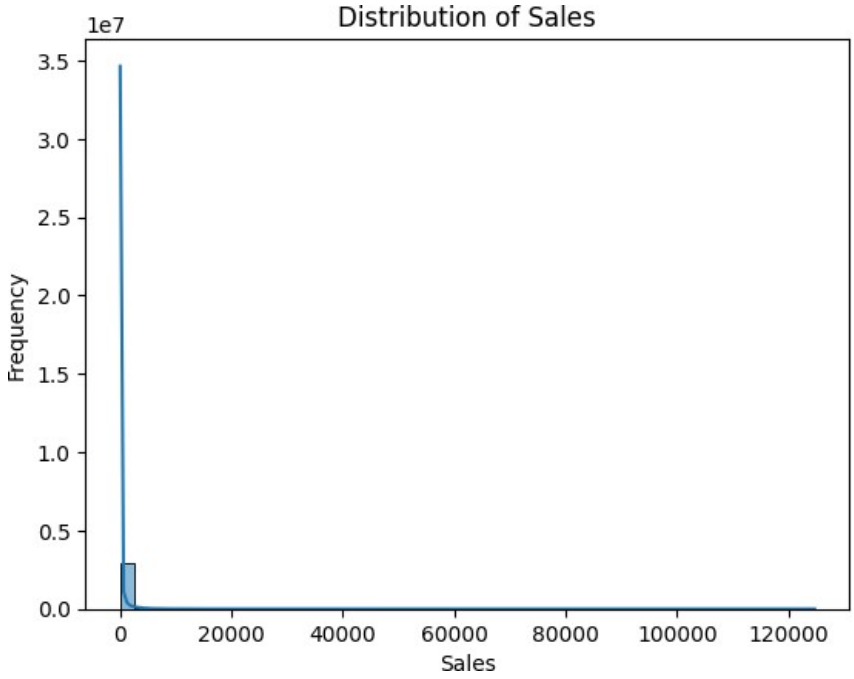


Figure 1: Distribution of sales in the merged training data. The distribution is strongly right-skewed, motivating the log transformation in Equation (1).

The notebooks show that 953,741 rows have zero sales, which corresponds to approximately 31.23% of the dataset.

This is important because forecasting methods must handle both continuous positive sales values and zero-demand observations.

4.2 Temporal Sales Patterns

Daily and monthly sales plots show strong time-dependent behavior. Figure 2 indicates that total sales change over time and include spikes, drops, and trend-like movement. Figure 3 shows that monthly sales generally increase over the observed period, although seasonality and sharp changes are also present.

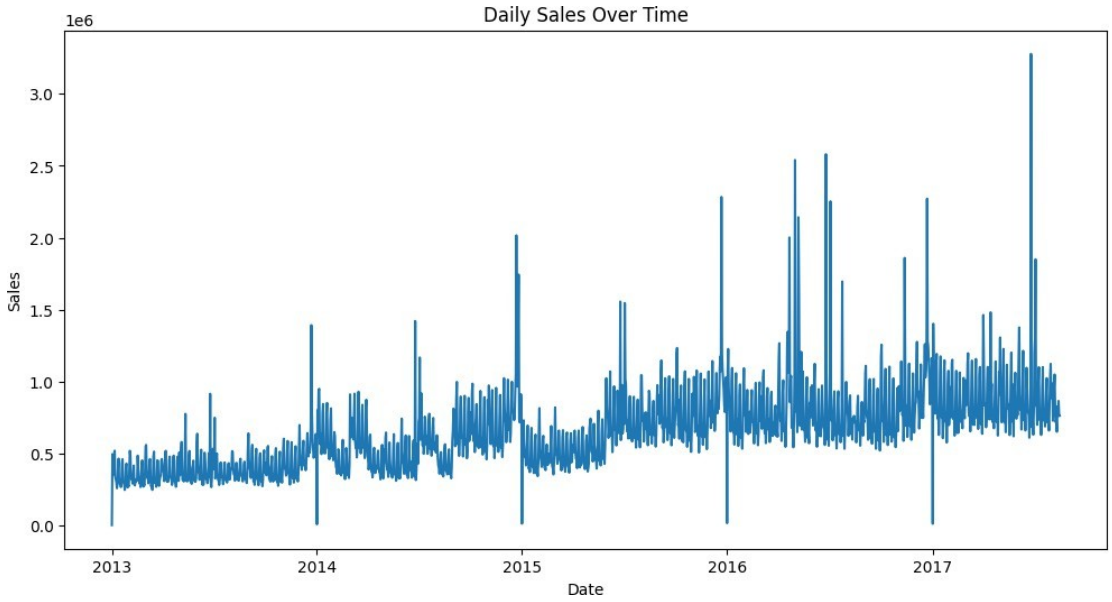


Figure 2: Daily aggregate sales over time. The plot shows nonstationary behavior, seasonal variation, and high-demand spikes.

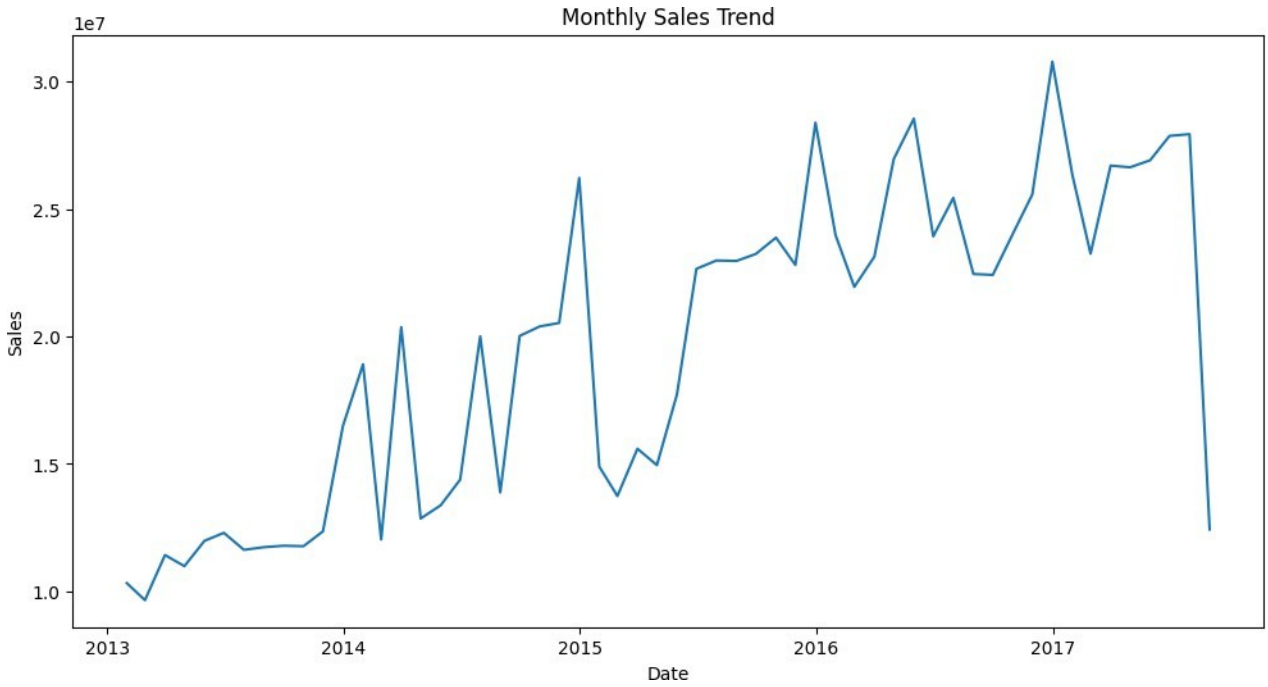


Figure 3: Monthly sales trend. The monthly aggregation smooths the daily noise and reveals broader demand movement over the full time span.

4.3 Store, Product Family, and Weekday Effects

Sales vary substantially by store and product family. Figure 4 shows that some stores contribute much more total sales than others. Figure 5 shows that product families such as grocery and beverage categories dominate total volume. Figure 6 shows that average sales are higher on weekends than on many weekdays.

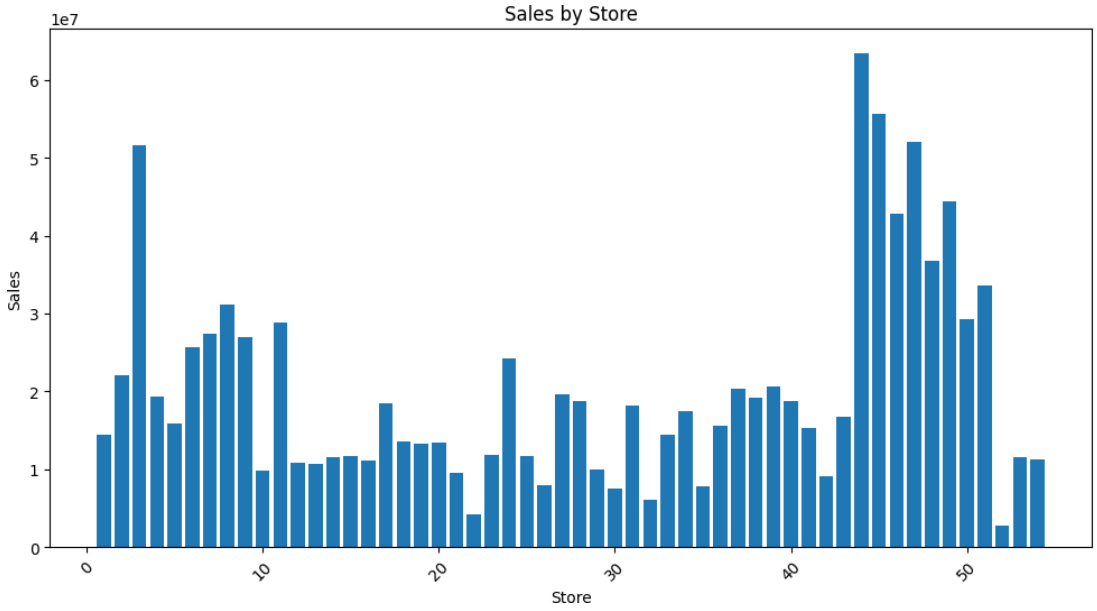


Figure 4: Total sales by store. The variability across stores supports including store-specific features and store metadata.

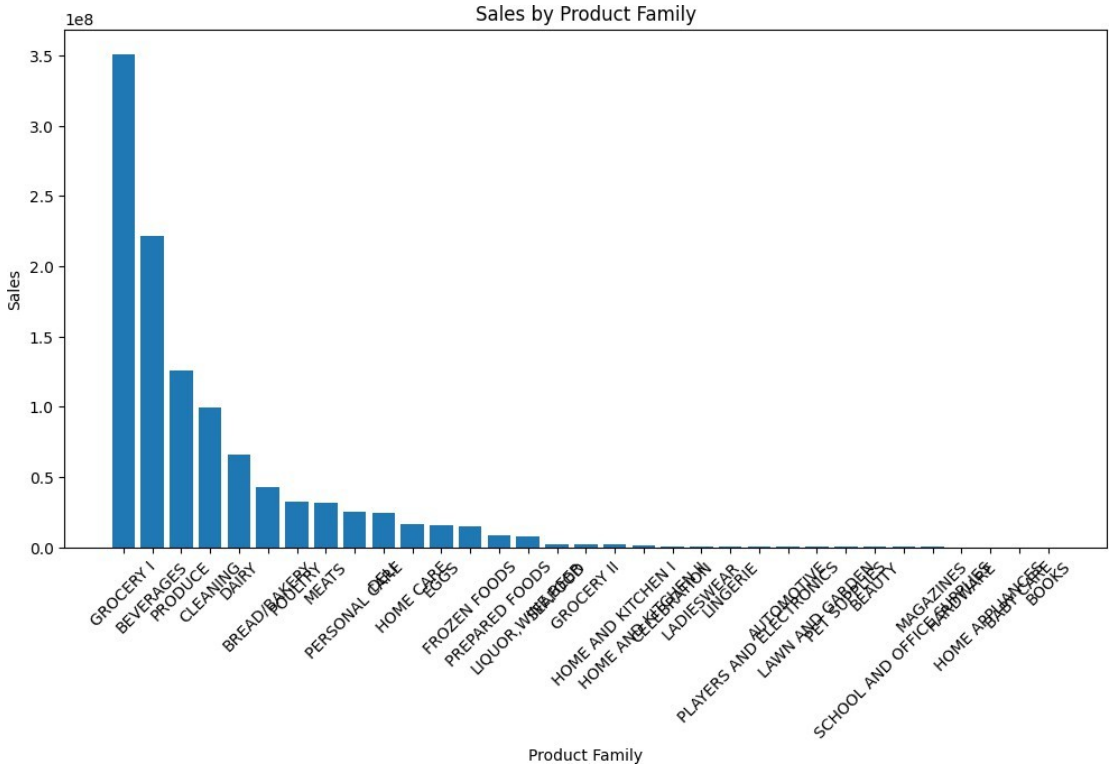


Figure 5: Sales by product family. The distribution across families is uneven, indicating strong category-specific demand behavior.

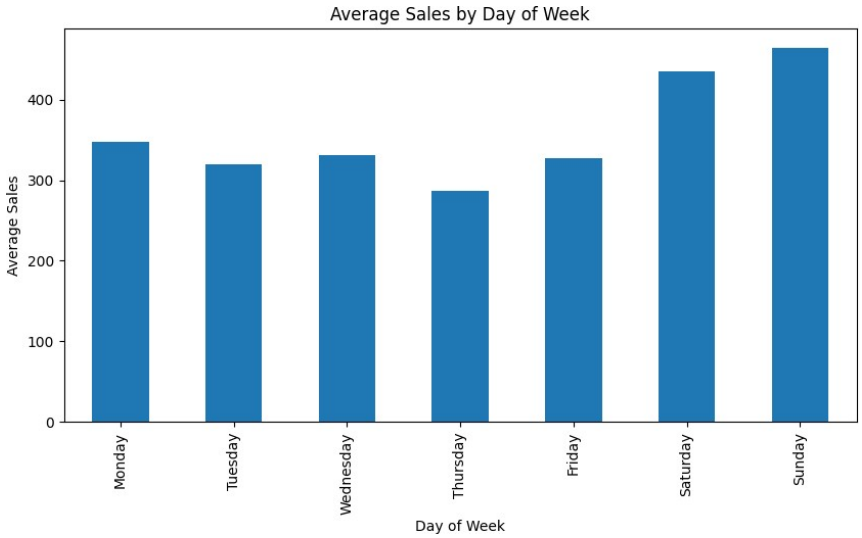


Figure 6: Average sales by day of week. Weekend demand is higher than many weekday averages.

4.4 Promotion and Transaction Effects

Promotions have a large observed association with average sales. The notebook output reports mean sales of 157.81 when products are not on promotion and 1139.83 when products are on promotion. Figure 7 visualizes this difference.

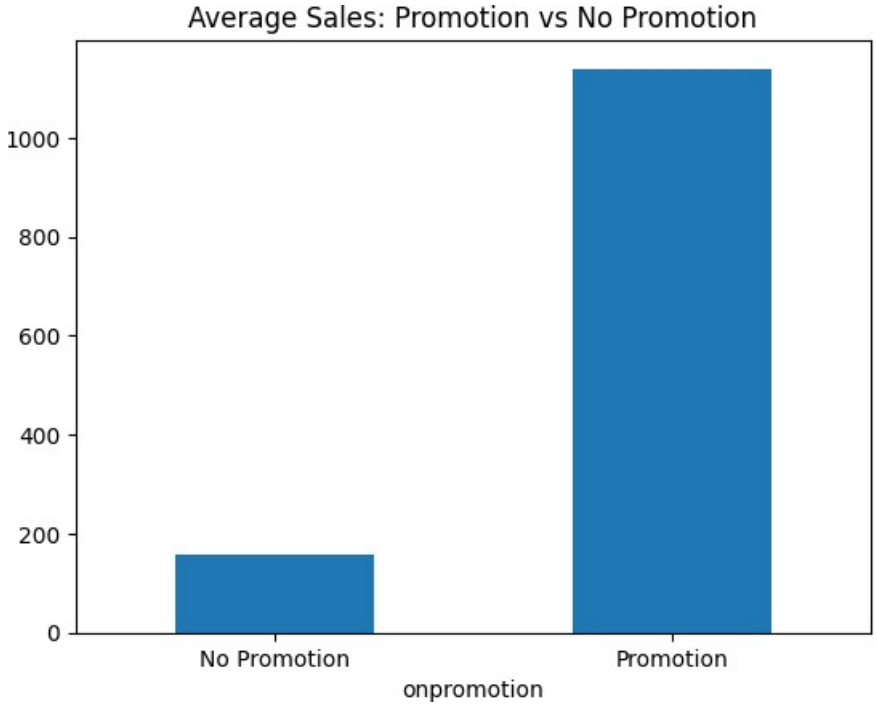


Figure 7: Average sales with and without promotion. The large difference indicates that promotion variables are important predictors.

Figure 8 shows a weak positive correlation between transactions and sales, approximately 0.21 in the plotted correlation matrix. This suggests that transaction counts contain useful information, but they do not alone explain sales variation.



Figure 8: Correlation matrix for transactions and sales. The relationship is positive but weak, so transactions are retained as one feature among many.

5 Feature Engineering

5.1 Missing-Value Treatment

The preprocessing notebook treats missing values according to variable meaning:

- Holiday columns are filled with None because missing holidays indicate non-holiday dates.
- transferred is converted to an integer indicator.
- Oil prices are filled by time-based forward and backward filling.
- Missing transaction counts are filled with 0.

After this step, the notebook output confirms that no missing values remain.

5.2 Calendar Features

Calendar variables are created from the date column:

$$\boldsymbol{x}_t^{\text{cal}} = [\text{year}_t,\ \text{month}_t,\ \text{day}_t,\ \text{day of week}_t,\ \text{week of year}_t,\ \text{is weekend}_t]\ . \tag{3}$$

These features allow the models to represent trend, weekly patterns, and seasonal effects.

5.3 Lag Features

For each store and product family, lag features are created using previous sales values. For lag $\ell \in \{1, 7, 14, 28\}$,

$$\text{Lag}_\ell(i,\ t) = y_{i,t-\ell}. \tag{4}$$

The 1-day lag captures immediate persistence, the 7-day lag captures weekly repetition, and the 14-day and 28-day lags capture longer demand memory.

5.4 Rolling Mean Features

Rolling features summarize recent demand while preventing target leakage. The rolling mean for window $w \in \{7, 14, 28\}$ is computed after shifting sales by one day:

$$\text{RollMean}_w(i, t) = \frac{1}{w} \sum_{k=1}^{w} y_{i,t-k}. \tag{5}$$

The shift by one day ensures that the current target value $y_{i,t}$ is not included in the feature vector used to predict itself.

5.5 Target Transformation and Final Feature Matrix

Figure 9 compares the original sales distribution and the log-transformed sales distribution. The transformed target is more balanced, which improves numerical stability and reduces the influence of extreme values in RMSE-based training and evaluation.

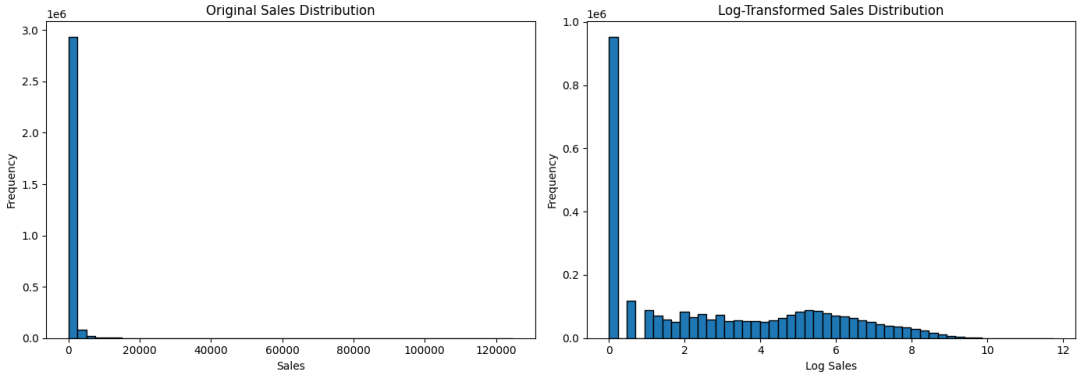


Figure 9: Original sales distribution compared with log-transformed sales distribution. The transformation reduces right skewness.

After lag and rolling features are created, rows without sufficient historical context are removed. The dataset decreases from 3,054,348 rows to 3,004,452 rows. After one-hot encoding categorical variables, the model-ready dataset contains 3,004,452 rows and 107 columns. The final feature matrix has 104 predictors, and the target vector contains 3,004,452 log-sales values.

6 Modeling Methodology

6.1 Time-Aware Train/Validation Design

Because the observations are time-indexed, random splitting would leak future information into the training process. Instead, the notebooks use a time-ordered split and TimeSeriesSplit. The training set contains 2,402,136 rows, and the final holdout validation/test set contains 602,316 rows. For cross-validation, each fold trains on earlier dates and validates on later dates.

Table 1: Time-series cross-validation fold structure.

Fold	Train Rows	Validation Rows	Train End	Validation Start	Validation End
1	400,356	400,356	2013-09-06	2013-09-06	2014-04-19
2	800,712	400,356	2014-04-19	2014-04-19	2014-11-25
3	1,201,068	400,356	2014-11-25	2014-11-26	2015-07-04
4	1,601,424	400,356	2015-07-04	2015-07-04	2016-02-14

5	2,001,780	400,356	2016-02-14	2016-02-14	2016-09-16

6.2 Evaluation Metric

The metric used throughout the project is RMSE on the log-transformed target:

$$\text{RMSE}_{\log} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(z_j - \hat{z}_j)^2}, \tag{6}$$

where $z_j = \log(1 + y_j)$ and $\hat{z}_j$ is the model prediction. A lower value indicates better predictive accuracy on the transformed target scale.

6.3 Linear Regression Baseline

Linear Regression is used as the baseline model. It assumes an additive linear relationship between the predictors and log-sales:

$$\hat{z}_i = \beta_0 + \sum_{p=1}^{P} \beta_p x_{ip}. \tag{7}$$

This model is useful as a benchmark, but it is limited because retail demand contains nonlinear interactions among store, family, promotion, date, and historical sales features.

6.4 Random Forest

Random Forest is an ensemble of decision trees trained on bootstrapped samples and randomized feature subsets. The prediction is the average of B regression trees:

$$\hat{z}_i = \frac{1}{B}\sum_{b=1}^{B} T_b(\boldsymbol{x}_i), \tag{8}$$

where T_b is the prediction from tree b. The notebook uses $B = 100$ trees, max depth=15, random state=42, and parallel training with n jobs=-1.

6.5 LightGBM

LightGBM is a gradient boosting method that builds trees sequentially. At iteration m, the model is updated by adding a new tree f_m:

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \eta f_m(\boldsymbol{x}), \tag{9}$$

where η is the learning rate. The LightGBM model in the notebook uses 500 estimators, learning rate 0.05, maximum depth 10, 31 leaves, and random seed 42. LightGBM is expected to perform well because it can capture nonlinearities and interactions efficiently in large tabular datasets.

7 Experimental Results

7.1 K-Fold Cross-Validation Results

Table 2 reports the fold-level RMSE values for the three models. Linear Regression has much larger error than the two tree-based models. Random Forest and LightGBM are close in average cross-validation performance, but LightGBM has a slightly lower average RMSE.

7.2 Final Holdout Test Results

Table 3 summarizes the final model comparison on the unseen holdout set. LightGBM has the lowest final test RMSE and is therefore selected as the best-performing model.

Table 2: K-Fold RMSE on log-transformed sales. Lower is better.

Fold	Linear Regression	Random Forest	LightGBM
1	1.4012	0.5901	0.6411
2	1.4453	0.4487	0.4374
3	1.6055	0.4468	0.4233
4	1.4234	0.5610	0.5348
5	1.2694	0.4439	0.4493
Average	1.4290	0.4981	0.4972

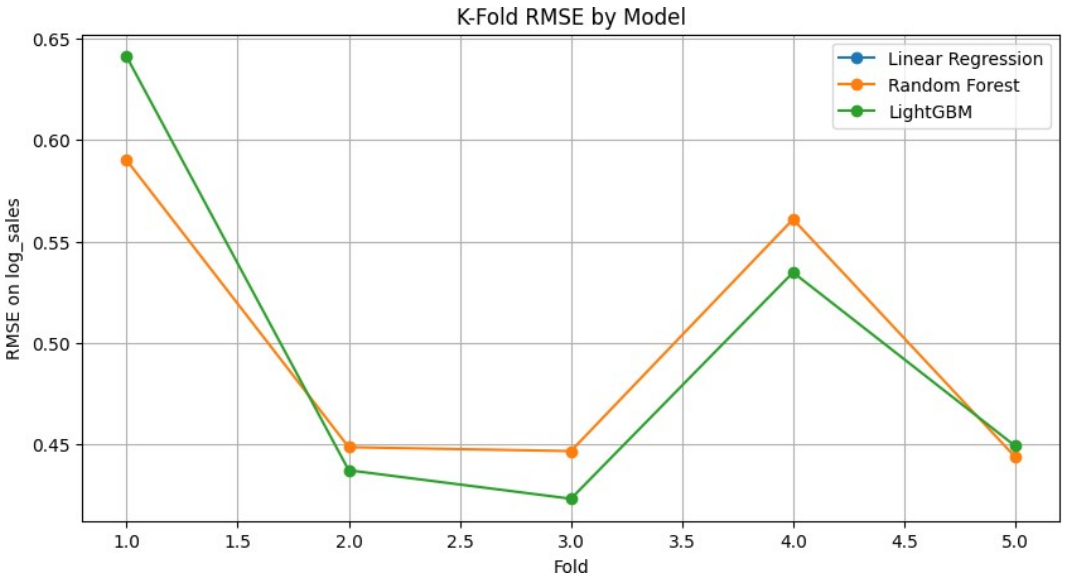


Figure 10: K-Fold RMSE by model across folds. The tree-based methods are much more accurate than Linear Regression.

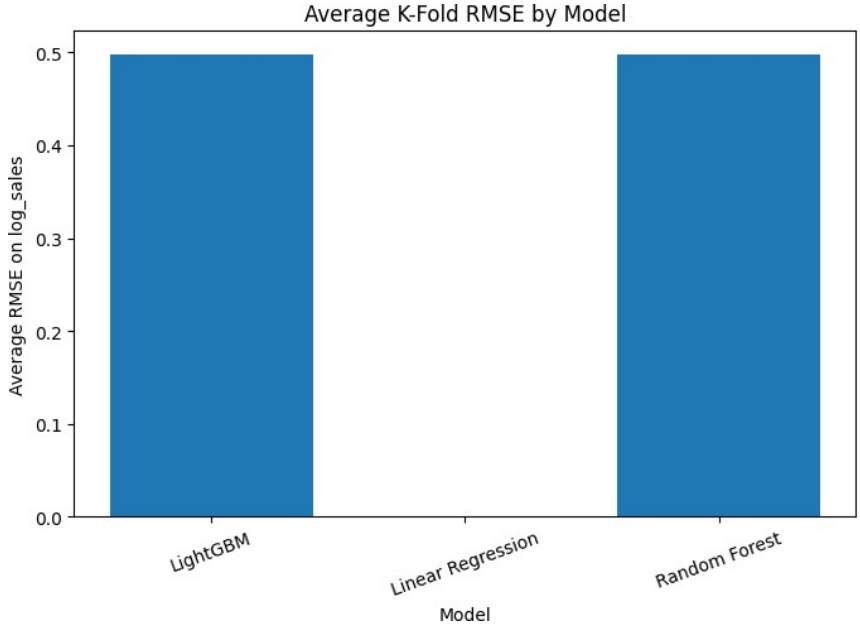


Figure 11: Average K-Fold RMSE by model as produced in the notebook. The Linear Regression bar is not shown in this plot because its fold-score column was named `rmse`, while the plotted column was `rmse log sales`. The corrected Linear Regression average is reported in Table 2.

Table 3: Final model comparison on unseen holdout data.

Model	Average K-Fold RMSE	Final Test RMSE	Test Rows
LightGBM	0.4972	0.4516	602,316
Random Forest	0.4981	0.4845	602,316
Linear Regression	1.4290	1.3541	602,316

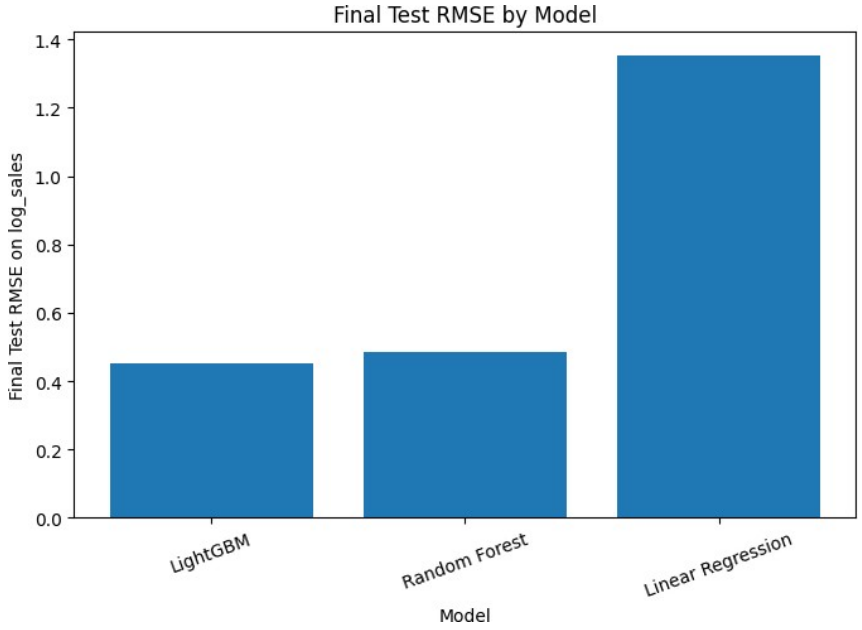


Figure 12: Final test RMSE by model. LightGBM gives the lowest final error.

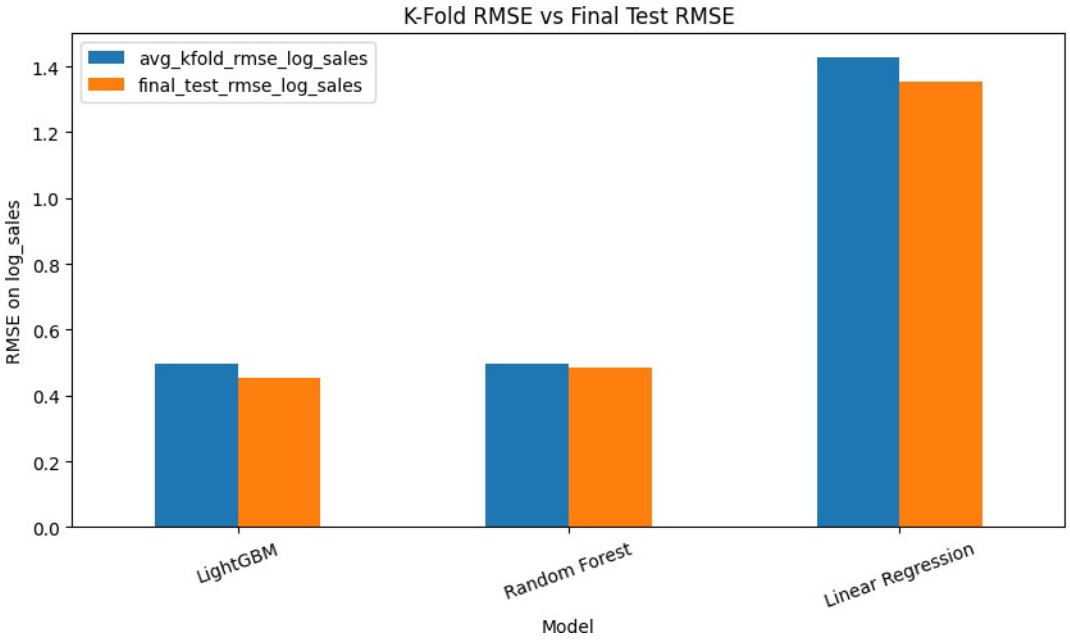


Figure 13: Comparison of average K-Fold RMSE and final test RMSE. The final test result is consistent with cross-validation: nonlinear tree-based models strongly outperform Linear Regression.

7.3 Prediction Examples on Original Sales Scale

The notebooks convert predicted log-sales back to the original sales scale using Equation (2). Table 4 shows the first five holdout predictions for each model. These examples show why nonlinear models perform better: Random Forest and LightGBM are much closer to high-volume observations than Linear Regression.

Table 4: First five holdout examples after inverse transformation to original sales scale.

Actual Sales	Linear Regression	Random Forest	LightGBM
2255.728	290.103	2233.615	2194.472
2165.483	36.597	2312.256	2245.528
10.000	19.701	6.495	7.315
253.000	271.049	283.044	290.422
795.000	677.494	930.969	889.111

8 Discussion

The results support three major findings. First, sales forecasting requires time-series-aware feature engineering. Lag features and rolling statistics provide direct historical context, and calendar variables capture weekly and seasonal patterns. Second, nonlinear relationships are central to the problem. The poor Linear Regression result shows that an additive linear model cannot adequately represent interactions among promotions, product families, stores, holidays, and previous sales. Third, LightGBM provides the best final predictive performance while remaining computationally efficient for a large tabular dataset.

The EDA also explains why the modeling pipeline needs multiple feature groups. Promotions produce much higher average sales, store and product family effects are highly uneven, weekends have different demand patterns, and the target distribution is skewed with many zeros. The final model must therefore handle sparse demand, extreme values, seasonality, and nonlinear feature interactions simultaneously.

9 Limitations and Future Work

This project provides a strong tabular machine learning baseline, but several extensions could improve forecasting quality:

- Add more domain-specific calendar features, such as payday effects, national events, and school/holiday season indicators.
- Use hierarchical forecasting to reconcile predictions across store, product family, and total-sales levels.
- Tune hyperparameters using a larger search space for Random Forest and LightGBM.
- Add prediction intervals using quantile regression or probabilistic forecasting.
- Evaluate business metrics such as stockout risk, overstock risk, service level, and inventory cost.
- Compare with specialized time-series models or global deep learning models for multi-series forecasting.

10 Conclusion

This project built a complete machine learning pipeline for store sales forecasting. The final dataset contained over three million processed observations and more than one hundred model-ready variables. The target variable was transformed using $\log(1 + y)$, and time-aware cross-validation was used to avoid

future-data leakage. Among the three evaluated models, LightGBM achieved the best final test RMSE of 0.4516 on log-sales, followed by Random Forest at 0.4845. Linear Regression had substantially higher error at 1.3541. Therefore, the recommended model for this forecasting task is LightGBM, because it provides the strongest holdout performance and can capture nonlinear interactions in large-scale tabular retail data.

A Important Code Snippets

The following snippets summarize the key implementation components from the project notebooks. Full code is available in the original notebooks.

A.1 Data Loading and Merging

```
from pathlib import Path
import pandas as pd import
numpy as np

DATA_DIR = Path("../data/csv/store-sales")
OUTPUT_DIR = Path("../data/processed")
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

train = pd.read_csv(DATA_DIR / "train.csv")
test = pd.read_csv(DATA_DIR / "test.csv")
stores = pd.read_csv(DATA_DIR / "stores.csv") oil
= pd.read_csv(DATA_DIR / "oil.csv")
transactions = pd.read_csv(DATA_DIR / "transactions.csv") holidays
= pd.read_csv(DATA_DIR / "holidays_events.csv")

for data in [train, test, transactions, oil, holidays]:
    data["date"] = pd.to_datetime(data["date"])

train_merged = train.merge(stores, on="store_nbr", how="left")
train_merged = train_merged.merge(oil, on="date", how="left")
train_merged = train_merged.merge(
    transactions, on=["date", "store_nbr"], how="left"
)

holiday_features = holidays[["date", "type", "locale", "transferred"]].copy()
holiday_features = holiday_features.rename(columns={
    "type": "holiday_type",
    "locale": "holiday_locale"
})
train_merged = train_merged.merge(holiday_features, on="date", how="left")
train_merged.to_csv(OUTPUT_DIR / "train_merged.csv", index=False)
```

A.2 Feature Engineering

```
df["date"] = pd.to_datetime(df["date"])

# Missing-value treatment
for col in ["holiday_type", "holiday_locale"]: df[col]
    = df[col].fillna("None")

df["transferred"] = df["transferred"].fillna(False).astype(int)
df["dcoilwtico"] = df["dcoilwtico"].ffill().bfill() df["transactions"]
= df["transactions"].fillna(0)

# Calendar features
df["year"] = df["date"].dt.year
df["month"] = df["date"].dt.month
df["day"] = df["date"].dt.day df["day_of_week"]
= df["date"].dt.dayofweek
df["week_of_year"] = df["date"].dt.isocalendar().week.astype(int)
df["is_weekend"] = df["day_of_week"].isin([5, 6]).astype(int)

# Lag and rolling features
df = df.sort_values(["store_nbr", "family", "date"]) for
lag in [1, 7, 14, 28]:
    df[f"sales_lag_{lag}"] = (
        df.groupby(["store_nbr", "family"])["sales"].shift(lag)
    )

for window in [7, 14, 28]:
    shifted_sales = df.groupby(["store_nbr", "family"])["sales"].shift(1) df[f"sales_roll_mean_{window}"]
    = (
        shifted_sales.groupby([df["store_nbr"], df["family"]])
        .rolling(window)
        .mean()
        .reset_index(level=[0, 1], drop=True)
    )
```

```
# Target transformation
df["log_sales"] = np.log1p(df["sales"])
```

A.3 Time-Series Cross-Validation

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error import
numpy as np

tscv = TimeSeriesSplit(n_splits=5)
fold_results = []

for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1): X_train_fold
    = X.iloc[train_idx]
    X_val_fold = X.iloc[val_idx]
    y_train_fold = y.iloc[train_idx]
    y_val_fold = y.iloc[val_idx]

    model.fit(X_train_fold, y_train_fold)
    preds = model.predict(X_val_fold)

    rmse = np.sqrt(mean_squared_error(y_val_fold, preds))
    fold_results.append({
        "fold": fold, "rmse_log_sales":
        rmse, "train_rows":
        len(train_idx),
        "validation_rows": len(val_idx)
    })
```

A.4 Random Forest Model

```
from sklearn.ensemble import RandomForestRegressor

final_rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=15,
    random_state=42,
    n_jobs=-1
)

final_rf_model.fit(X_train, y_train)
val_preds_log = final_rf_model.predict(X_val)
test_rmse_log = np.sqrt(mean_squared_error(y_val, val_preds_log))
```

A.5 LightGBM Model

```
from lightgbm import LGBMRegressor

final_lgbm_model = LGBMRegressor(
    n_estimators=500,
    learning_rate=0.05, max_depth=10,
    num_leaves=31,
    random_state=42,
    n_jobs=-1
)

final_lgbm_model.fit(X_train_model, y_train) val_preds_log
= final_lgbm_model.predict(X_val_model)
test_rmse_log = np.sqrt(mean_squared_error(y_val, val_preds_log))
```

A.6 Inverse Transformation of Predictions

```
actual_sales = np.expm1(y_val)
predicted_sales = np.expm1(val_preds_log)
predicted_sales = np.maximum(predicted_sales, 0)
```