Sales Forecasting Across Multiple Retail Stores

In this project, I developed a sales forecasting model for multiple retail stores. I began by cleaning the dataset, which contained over 1 million records, addressing missing values, removing duplicates, and handling outliers. This preparation enabled me to analyze sales trends over time.

For exploratory data analysis (EDA), I uncovered insights about customer behaviour & sales performance, & engineered features related to dates, promotions, and competition.

In the machine learning (ML) part, I built the forecasting model using Sklearn pipelines with the Random Forest Regressor.

In the deep learning (DL) part, I conducted the ADF test to assess the stationarity of the time series data. Then Built the LSTM model using TensorFlow Keras with MLflow autologging. Created a Python file for a Streamlit dashboard that allows users to upload CSV or Excel files, displaying the last available date, a data preview, & generating six-week sales predictions along with a prediction & feature importance graph.

This project integrated data cleaning, EDA, ML & DL model building, and an interactive dashboard to provide a practical solution for sales forecasting.

by Debasis Baidya

Contributions of Key Libraries & Modules to the Project

NumPy, Pandas:

I used NumPy for efficient numerical operations & Pandas for data manipulation, which made handling large datasets straightforward.

Matplotlib, Seaborn:

I leveraged Matplotlib and Seaborn to create insightful visualizations, helping me uncover patterns in sales data during my exploratory analysis.

Logging, OS Module:

The Logging module helped me track the project's progress and debug issues, while the OS module managed file paths for data handling.

Sklearn, Scipy:

Sklearn provided tools for preprocessing and model evaluation, and Scipy offered essential statistical functions, which were vital for model development.

Statsmodels, Pickle:

I used Statsmodels for statistical analysis of sales data and Pickle to save my models, making it easy to load them later.

TensorFlow, Keras:

TensorFlow and Keras were crucial for building and training my LSTM model, allowing me to implement complex neural network architectures.

MLflow:

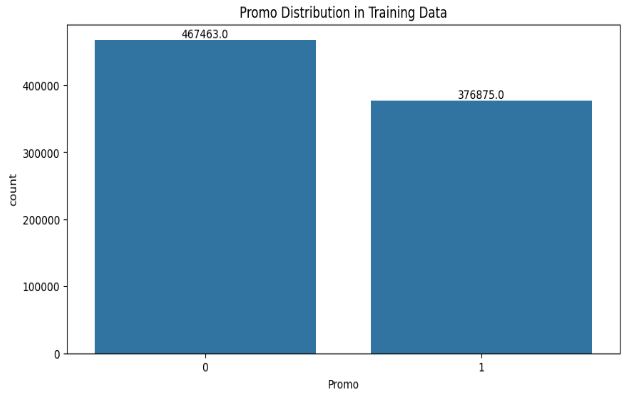
MLflow helped me track experiments and manage the machine learning lifecycle, ensuring reproducibility in my model evaluations.

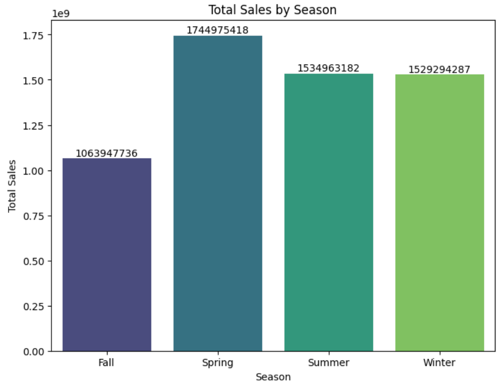
Streamlit:

I used Streamlit to create an interactive web app for visualizing predictions, making it easy for stakeholders to explore the results.

```python
# Check for missing values and duplicates before handling them
print("Missing Values Count in Train Data:")
print(train_data.isnull().sum())

print("\nMissing Values Count in Test Data:")
print(test_data.isnull().sum())

# Check for duplicates
train_duplicates = train_data.duplicated().sum()
test_duplicates = test_data.duplicated().sum()

print(f"\nNumber of Duplicates in Train Data: {train_duplicates}")
print(f"Number of Duplicates in Test Data: {test_duplicates}")
```

```
Missing Values Count in Train Data:
Store                             0
DayOfWeek                         0
Date                              0
Sales                             0
Customers                         0
Open                              0
Promo                             0
StateHoliday                      0
SchoolHoliday                     0
StoreType                         0
Assortment                        0
CompetitionDistance            2642
CompetitionOpenSinceMonth    323348
CompetitionOpenSinceYear     323348
Promo2                            0
Promo2SinceWeek              508031
Promo2SinceYear              508031
PromoInterval                508031
dtype: int64

Missing Values Count in Test Data:
Id                0
Store             0
DayOfWeek         0
Date              0
Open             11
Promo             0
StateHoliday      0
SchoolHoliday     0
dtype: int64
```





Data Exploration, Cleaning, Processing Steps

1. **Logging Configuration**

   I configured logging using Python's logging module to track the project's progress and errors, ensuring all important events were logged to a file for easy debugging.

2. **Merging Train & Store Data**

   I merged the training dataset with store data during my exploratory data analysis (EDA) to better understand the relationships and patterns in the data.

3. **Data Cleaning and Preprocessing**

   I focused on data cleaning and preprocessing to handle missing values, remove duplicates & standardize formats, ensuring the dataset was ready for analysis & modeling.

4. **Feature Engineering**

   I performed feature engineering to create new variables from the existing data, which helped enhance the analysis during my exploratory data analysis (EDA).

5. **Exploratory Data Analysis Visualizations**

   I plotted promo distribution, total sales by season, and sales distribution by promo etc. These visuals helped me identify key patterns during my EDA.

6. **Dropping Feature Engineering Columns**

   I dropped the feature engineering columns from the DataFrame to revert Train Data to its default state, ensuring a clean dataset for further analysis and modeling.

Data Preparation & Visualization Summary

Renaming Train Data Column to Match Test Data Column

```python
# Rename 'Customers' column in Train Data to 'Id'
train_data = train_data.rename(columns={'Customers': 'Id'})  # Rename Customers to Id

# Create a list of columns that exist in both DataFrames
common_columns = test_data.columns.intersection(train_data.columns).tolist()

# Rearranging Train Data columns to match Test Data columns
train_data = train_data[common_columns + [col for col in train_data.columns if col not in common_columns]]

# Print columns present in Train & Test Data after rearranging:
logging.info("Train columns: {}".format(train_data.columns.to_list()))
logging.info("Test columns: {}".format(test_data.columns.to_list()))
```

```
2025-03-10 06:39:34,711 - INFO - Train columns: ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', 'Sales']
2025-03-10 06:39:34,712 - INFO - Test columns: ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday']
```

Visualize Distributions of Features

```python
import matplotlib.pyplot as plt

# Plot histogram for Train Data excluding 'Date' column
train_data.hist(column=train_data.select_dtypes(include=['number']).columns.difference(['Date']),
                figsize=(12, 8), bins=30, edgecolor='black')
plt.suptitle("Train Data Distribution")
plt.show()

# Plot histogram for Test Data excluding 'Date' column
test_data.hist(column=test_data.select_dtypes(include=['number']).columns.difference(['Date']),
               figsize=(12, 8), bins=30, edgecolor='black')
plt.suptitle("Test Data Distribution")
plt.show()
```

Boxplot Comparison of Train and Test Data Distributions: Analyzing Feature Variability and Outliers

```python
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec

# Create a figure with GridSpec
fig = plt.figure(figsize=(15, 6))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 1])  # 1 row, 2 columns

# Boxplot for Train Data
ax1 = fig.add_subplot(gs[0])
sns.boxplot(data=train_data.select_dtypes(include=['number']), ax=ax1)
ax1.set_title("Boxplot of Train Data (Numerical Features)")
ax1.tick_params(axis='x', rotation=45)

# Boxplot for Test Data
ax2 = fig.add_subplot(gs[1])
sns.boxplot(data=test_data.select_dtypes(include=['number']), ax=ax2)
ax2.set_title("Boxplot of Test Data (Numerical Features)")
ax2.tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout()
plt.show()
```

Renamed Customer Column

Renamed the "Customer" column in the training data to "Id" to match the test data column.

Visualized Distributions of Features

Plotted distributions for various features to understand their characteristics.

Boxplot Comparison

Created boxplots to compare train and test data distributions, analyzing feature variability and outliers.

Analyzed Feature Variability

Assessed the spread of features across train and test datasets to identify differences.

Identified Outliers

Used boxplots to pinpoint outliers that may influence model performance.

Checked for Consistency

Ensured that the feature distributions were aligned between train and test datasets.

Winsorization of Sales Data

1

Outlier Treatment

Implemented Winsorization on the 'Sales' column to limit extreme values, enhancing data robustness.

2

Data Integrity Check

Verified the transformation by displaying the original & winsorized sales values side by side.

3

Comparative Visualization

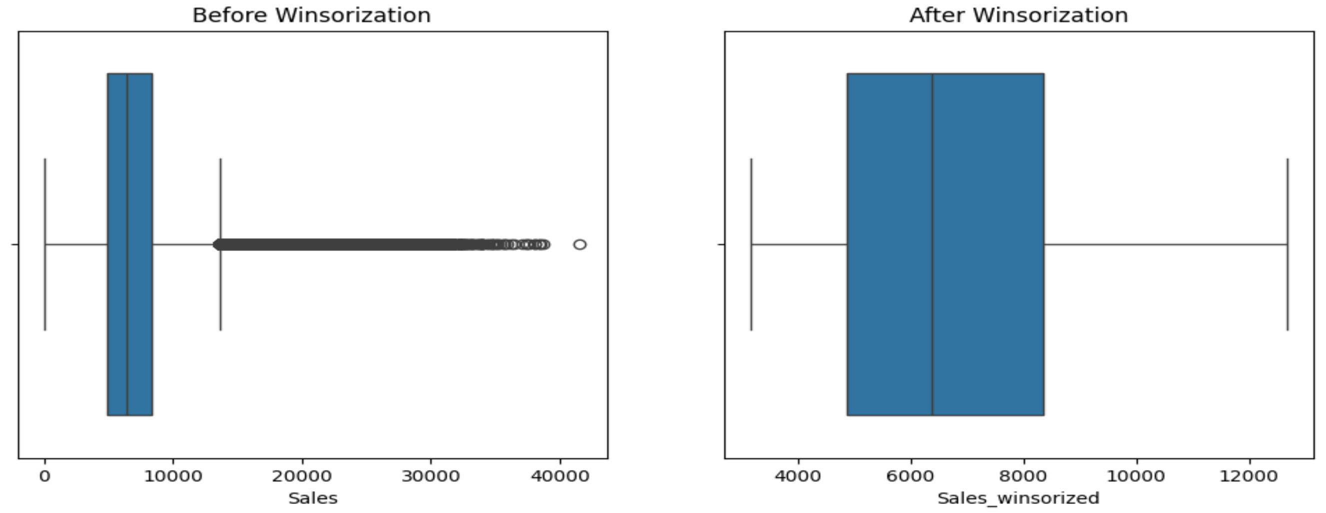
Created boxplots to illustrate the difference in sales distribution before and after Winsorization, highlighting reduced variability.

Capping Extreme Outliers with Winsorization: *Capping prevents extreme values from dominating the model*

```python
import numpy as np
from scipy.stats.mstats import winsorize

# Apply Winsorization with higher limits to handle more outliers
train_data['Sales_winsorized'] = winsorize(train_data['Sales'], limits=[0.05, 0.05])  # Cap 5% from both ends

# Convert winsorized values to integers if needed
train_data['Sales_winsorized'] = train_data['Sales_winsorized'].astype(int)

# Display the first few entries to verify
print(train_data[['Sales', 'Sales_winsorized']].head())
```

```
   Sales  Sales_winsorized
0   5263              5263
1   6064              6064
2   8314              8314
3  13995             12668
4   4822              4822
```

```python
# Checking by plotting a boxplot:
# Plot Before & After Winsorization
plt.figure(figsize=(12, 5))

# Before Winsorization
plt.subplot(1, 2, 1)
sns.boxplot(x=train_data['Sales'])
plt.title("Before Winsorization")

# After Winsorization
plt.subplot(1, 2, 2)
sns.boxplot(x=train_data['Sales_winsorized'])
plt.title("After Winsorization")

plt.show()
```



Model Training With Sklearn Pipeline

2.2 Building models with sklearn pipelines

```
[ ]
# Import Required Libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pickle
from datetime import datetime

# Define features and target variable from Train Data
X_train = train_data.drop(columns=['Sales', 'Sales_winsorized', 'Date'])  # Drop unnecessary columns
y_train = train_data['Sales_winsorized']  # Target variable

# Define Test Data
X_test = test_data.drop(columns=['Sales', 'Date'], errors='ignore')
```

```
[ ]
# Check shapes
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
```

```
X_train shape: (844338, 7)
y_train shape: (844338,)
X_test shape: (41088, 7)
```

```
# Create Pipeline for Scaling & Model Training
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

ml_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1))
])
```

```
# Train Model Using Pipeline
ml_pipeline.fit(X_train, y_train)
```

```
Pipeline
  StandardScaler
  RandomForestRegressor
```

2.3 Choose & Justify Loss Function

```
[ ]
# Compute Evaluation Metrics
mae = mean_absolute_error(y_train, y_train_pred)
mse = mean_squared_error(y_train, y_train_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_train, y_train_pred)

# Print Evaluation Results
print("Model Evaluation Metrics (on Training Data):")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2*100:.2f}%")
```

```
Model Evaluation Metrics (on Training Data):
Mean Absolute Error (MAE): 152.53
Mean Squared Error (MSE): 51475.51
Root Mean Squared Error (RMSE): 226.88
R² Score: 99.22%
```

The chosen loss function is MAE (Mean Absolute Error) because:

- MAE (152.53): The average difference between predicted and actual values is approximately 152.53 units.
- MSE (51475.51): The average squared difference between predicted and actual values is approximately 51475.51 units squared.
- RMSE (226.88): The square root of MSE, representing the average magnitude of errors, is approximately 226.88 units.
- R^2 Score (99.22%): The model explains about 99.22% of the variance in the training data, indicating an excellent fit.

*Note: Although MSE seems suitable based on the high R^2 score and relatively low error values, **MAE** is chosen as the loss function.*

01 Pipeline Construction

Built a Random Forest Regressor Pipeline to streamline the training process.

02 Feature Scaling

Implemented feature scaling to normalize the input data, enhancing model performance.

03 Training Process

Trained the pipeline on the processed data, ensuring that it learned from relevant features.

04 Performance Evaluation Metrics

Evaluated the model's performance using several metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

05 Predictive Ability

Achieved an R^2 Score of 99.22, indicating a strong predictive ability and excellent fit to the training data.

Predictions From The Last Date To The Next 6 Weeks

Sales Prediction Steps for the Last Date

```python
from datetime import timedelta

# Step 1: Identify the last date
last_date = train_data['Date'].max()
print("Last Date in Train Data:", last_date)

# Step 2: Filter data for the last date
last_date_data = train_data[train_data['Date'] == last_date]
print("Data for Last Date:")
print(last_date_data)

# Step 3: Prepare features (X) for prediction
X_last_date = last_date_data.drop(columns=['Sales', 'Sales_winsorized', 'Date'])
print("Features for Last Date:")
print(X_last_date)

# Step 4: Make predictions for the last date
y_last_date_pred = ml_pipeline.predict(X_last_date)
print("Predicted Sales for Last Date:", y_last_date_pred)
```

```
Last Date in Train Data: 2015-07-31 00:00:00
```

Prediction of Sales in various stores up to 6 weeks ahead of time using ML Pipeline

```python
from datetime import datetime, timedelta

# Step 5: Generate future dates (next 6 weeks)
last_date = train_data['Date'].max()
future_dates = [last_date + timedelta(weeks=i) for i in range(1, 7)]
print("Future Dates:")
for i, date in enumerate(future_dates, start=1):
    print(f"Next Week {i} from last date: {date}")

# Step 6: Prepare future data for predictions
future_data = []
feature_columns = X_train.columns  # Get the feature column names from the training data
unique_stores = train_data['Store'].unique()  # Get all unique store IDs

for date in future_dates:
    for store in unique_stores:
        # Create a dictionary for the future data
        future_entry = {
            'Date': date,
            'Store': store
        }
        # Add the feature columns with placeholder values
        for col in feature_columns:
            future_entry[col] = 0
        future_data.append(future_entry)

# Convert to DataFrame
future_df = pd.DataFrame(future_data)

# Ensure the feature columns are correctly aligned
future_df['Date'] = pd.to_datetime(future_df['Date'])

# Predict future sales
y_future_pred = ml_pipeline.predict(future_df[feature_columns])
future_df['Predicted_Sales'] = y_future_pred

# Print the future predictions
print("\nFuture Predictions:")
for i, (date, store, pred) in enumerate(zip(future_dates, future_df['Store'], y_future_pred), start=1):
    print(f"For Store: {store} - Prediction of Next Week {i}: {date} - Predicted Sales: {int(pred)}")
```

1

Last Date Analysis

The last training date is July 31, 2015, with significant sales variability across stores, indicating diverse consumer preferences.

2

Sales Variability

Sales ranged from 4,822 to 27,508, highlighting the impact of location and store strategies.

3

Future Predictions

Store 0's future sales are consistently predicted at 3,174 for the next 6 weeks, suggesting a stable baseline.

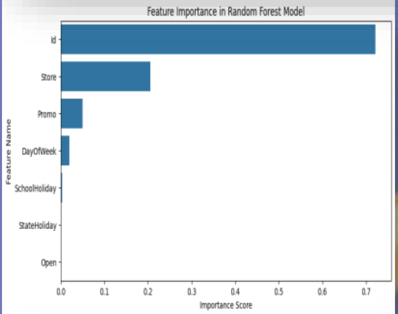
2.4 Post Prediction Analysis - Feature Importance

```python
# Extract feature importance from the trained Random Forest model
feature_importances = ml_pipeline.named_steps['model'].feature_importances_

# Create DataFrame for visualization
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10, 5))
sns.barplot(x=feature_importance_df['Importance'], y=feature_importance_df['Feature'])
plt.title("Feature Importance in Random Forest Model")
plt.xlabel("Importance Score")
plt.ylabel("Feature Name")
plt.show()
```

2.4 Post Prediction Analysis - Confidence Interval Estimation

```python
# Generate predictions using multiple trees in the Random Forest
all_predictions = np.array([tree.predict(X_test) for tree in ml_pipeline.named_steps['model'].estimators_])

# Compute confidence intervals (5th and 95th percentile)
lower_bound = np.percentile(all_predictions, 5, axis=0)
upper_bound = np.percentile(all_predictions, 95, axis=0)

# Store confidence intervals in test data
test_data['Predicted_Sales'] = ml_pipeline.predict(X_test)
test_data['Lower_Bound'] = lower_bound
test_data['Upper_Bound'] = upper_bound

# Save predictions with confidence intervals
test_data[['Predicted_Sales', 'Lower_Bound', 'Upper_Bound']].to_csv('predicted_sales_with_ci.csv', index=False)

# --------------- Print Output to Verify ---------------
print("Sample Predictions with Confidence Intervals:")
print(test_data[['Predicted_Sales', 'Lower_Bound', 'Upper_Bound']].head())
```

```
Sample Predictions with Confidence Intervals:
   Predicted_Sales  Lower_Bound  Upper_Bound
0           3174.0       9120.4      12668.0
1           3174.0       9408.0      12668.0
2           3174.0      12668.0      12668.0
3           3174.0      12668.0      12668.0
4           3174.0      12668.0      12668.0
```



2.5 Serialize the Model: *Save Trained Model as `.pkl` File with Timestamp*

```python
import pickle
from datetime import datetime

# Save Trained Model as `.pkl` File with Timestamp
model_filename = f"rf_model_{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}.pkl"
pickle.dump(ml_pipeline, open(model_filename, 'wb'))

print(f"Model saved as {model_filename}")
```

```
Model saved as rf_model_2025-03-10-06-41-47.pkl
```

Dropping Additional Columns Created and Retaining Original Columns to Save Train and Test Data for Future Reference

```python
# Drop 'Sales_log' from train_data before saving
train_data = train_data.drop(columns=['Sales', 'Sales_log'], errors='ignore')

# Drop 'Predicted_Sales', 'Lower_Bound', 'Upper_Bound' from test_data before saving
test_data = test_data.drop(columns=['Predicted_Sales', 'Lower_Bound', 'Upper_Bound'], errors='ignore')

# Print columns of Train & Test Datasets
print("Train File\n", train_data.columns.to_list())
print("Test File\n", test_data.columns.to_list())
```

```
Train File
 ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', 'Sales_winsorized']
Test File
 ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday']
```

Post-Prediction Analysis & Confidence Intervals

1

Feature Importance Analysis

Identified key predictors, with Store, Promo, and DayOfWeek having the highest impact on sales.

2

Analysis & Confidence Intervals

Estimated confidence intervals (5th & 95th percentile) for predictions, providing uncertainty estimates.

3

Model Serialization

Saved the trained model as a timestamped .pkl file for future deployment. “rf_model_2025-03-10-06-41-47.pkl”.

4

Dropping Additional Columns

Dropped Unwanted Columns: Predicted_Sales (irrelevant for raw test data), Lower & Upper_Bound (confidence interval bounds not needed for predictions).

5

Renaming "Sales_winsorized"

The column 'Sales_winsorized' in train_data has been renamed to 'Sales'.

6

Generated Files

train_file.csv (Processed Training Data)
test_file.csv (Processed Test Data)

Time Series Analysis

Sales Trend

Clear seasonal patterns with periodic peaks and drops.

Stationary Data

ADF test confirms sales data is stationary (p-value: 0.0001).

ACF/PACF

Strong weekly seasonality (lags of 7, 14, 21, 28).

Residual Analysis

Examined residuals for heteroscedasticity and autocorrelation.

Decomposition

Identified trend, seasonal, and residual components.

30-Day Sliding Window Forecast

Different sequences overlap, reflecting diverse sales patterns across various periods.

Aggregate sales by date for time series analysis

```python
### Step 3: Aggregate sales by date for time series analysis

import matplotlib.pyplot as plt

# Aggregate sales by date for time series analysis
sales_trend = train_df.groupby("Date")["Sales"].sum()

plt.figure(figsize=(12,5))
plt.plot(sales_trend, label='Sales Trend')
plt.xlabel("Date")
plt.ylabel("Sales")
plt.title("Sales Trend Over Time")
plt.legend()
plt.show()
```

Autocorrelation (ACF) & Partial Autocorrelation (PACF) Analysis

```python
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Plot ACF
plt.figure(figsize=(12, 6))
plot_acf(sales_trend, lags=30, ax=plt.gca())
plt.title('Autocorrelation Function (ACF)')
plt.show()

# Plot PACF
plt.figure(figsize=(12, 6))
plot_pacf(sales_trend, lags=30, ax=plt.gca())
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()
```

```python
### Step 4: Perform ADF Test for Stationarity
from statsmodels.tsa.stattools import adfuller

# Perform the ADF test on the sales trend data
adf_result = adfuller(sales_trend)

# Output the ADF statistic and p-value
print(f"ADF Statistic: {adf_result[0]}")
print(f"p-value: {adf_result[1]}")

# Interpret the result
if adf_result[1] < 0.05:
    print("Time series is stationary.")
else:
    print("Time series is not stationary, differencing may be required.")
```

```
ADF Statistic: -4.6062197626111505
p-value: 0.00012580255594144246
Time series is stationary.
```

Create 30-Day Sliding Window Forecast

```python
import numpy as np
import matplotlib.pyplot as plt

# Function to create sliding window data for visualization
def create_sliding_window_forecast(df, seq_length=30, sample_size=5):
    forecast_data = []
    indices = np.linspace(0, len(df) - seq_length - 1, sample_size, dtype=int)  # Pick evenly spaced sequences

    for i in indices:
        forecast_data.append(df[['Sales']].iloc[i:i + seq_length].values)

    return np.array(forecast_data)

# Get sample sliding window sequences
X_sliding_sample = create_sliding_window_forecast(train_df, sample_size=5)  # Only visualize 5 sequences

# Plot the sequences
plt.figure(figsize=(12, 6))
for i, sequence in enumerate(X_sliding_sample):
    plt.plot(sequence, label=f'Sequence {i+1}')

plt.title("Visualization of 30-Day Sliding Window Forecast")
plt.xlabel("Days")
plt.ylabel("Sales")
plt.legend()
plt.show()
```

Building the LSTM Model for Time Series Forecasting

Data Preparation

Converted 'Date' to numerical format (days since the first date).

Selected key features for LSTM training.

Selected Features

Features: ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday’]

Target: [‘Sales’]

Key Takeaways

Date conversion is crucial for LSTM.

Diverse features enhance model performance.

Conclusion

Features: ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday’]

Target: [‘Sales’]

Building the LSTM Model for Time Series Forecasting

```python
# Prepare Data for LSTM Training
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import pandas as pd

# Convert Date to datetime format
train_df['Date'] = pd.to_datetime(train_df['Date'])

# Define feature and target columns
feature_columns = ['Id', 'Store', 'DayOfWeek', 'Date', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday']
target_column = 'Sales'

# Extract features and target variable
X = train_df[feature_columns].copy()
y = train_df[target_column].values

# Diplay Max Columns
pd.set_option('display.max_columns', None)

# Print the first 5 rows of X
print("First 5 rows of Feature Matrix X:")
print(X[:5])

# Print the first 5 values of y
print("\nFirst 5 values of Target Array y:")
print(y[:5])
```

```
First 5 rows of Feature Matrix X:
     Id  Store  DayOfWeek        Date  Open  Promo  StateHoliday  SchoolHoliday
0   555      1          5  2015-07-31     1      1             0              1
1   625      2          5  2015-07-31     1      1             0              1
2   821      3          5  2015-07-31     1      1             0              1
3  1498      4          5  2015-07-31     1      1             0              1
4   559      5          5  2015-07-31     1      1             0              1

First 5 values of Target Array y:
[ 5263  6064  8314 12668  4822]
```

Data Scaling for Better LSTM Performance

Purpose of Scaling

Scaled numerical features and target variable to a range of -1 to 1 using MinMaxScaler.

Scaling Process

Created two MinMaxScaler objects: scaler_X: For numerical features. I scaler_y: For target variable (y).

Applied Scaling to Numerical Columns

Id, Store, DayOfWeek, Open, Promo, StateHoliday, SchoolHoliday.

Excluded the Date column from scaling.

Reshaped Scaled Input Data

Reshaped scaled input data (X_scaled) to the format required by LSTM: (samples, timesteps, features)

Outcome

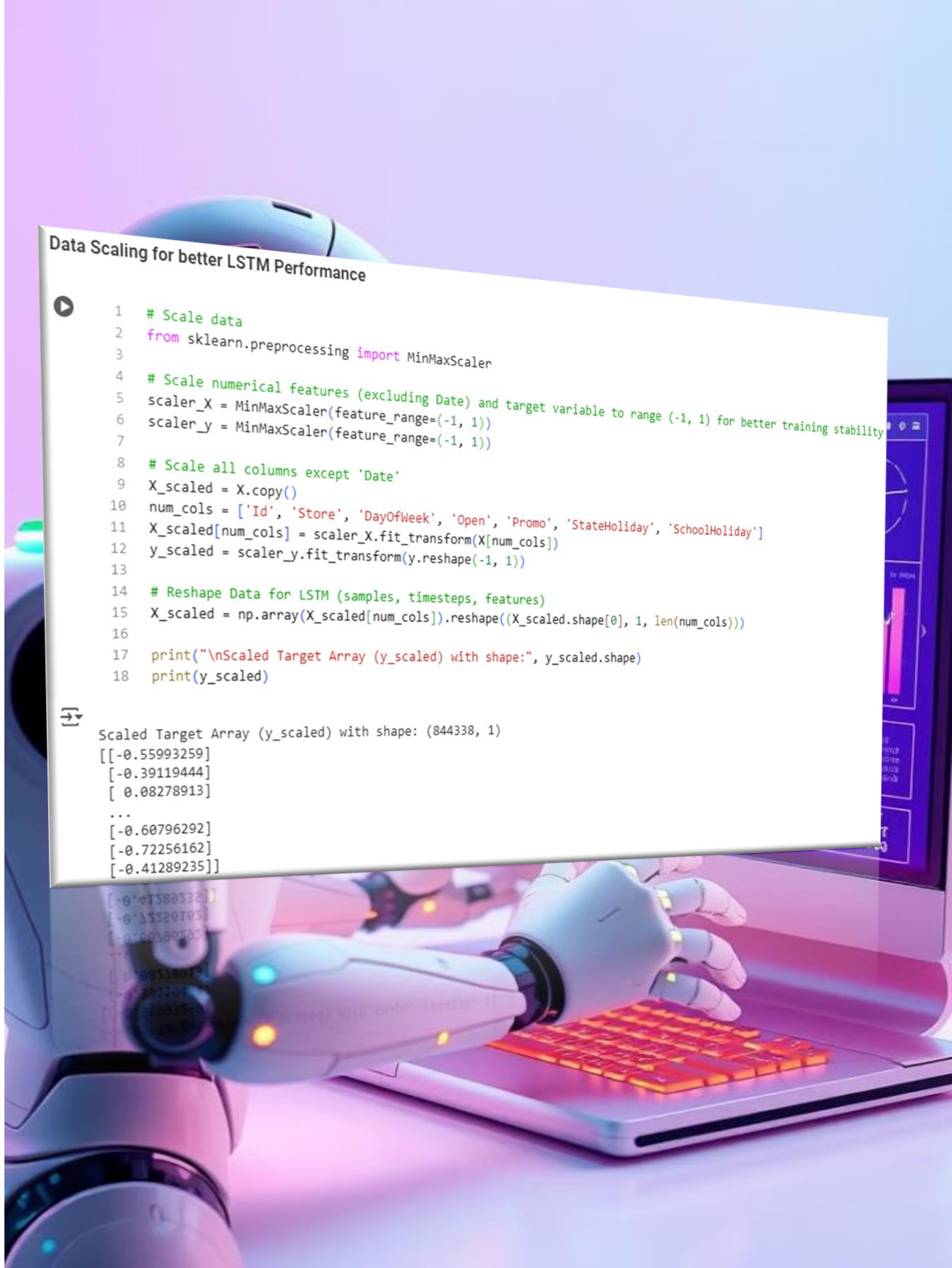
Numerical features and target variable successfully scaled to -1 to 1.

Input data reshaped for LSTM training.

Conclusion

Scaling improves stability and efficiency of LSTM training.

Ensures all features and target variables are on the same scale, enhancing the model's learning process.



Build Optimized LSTM Model

STEP 01

Model Framework

Utilized **Sequential** model from **TensorFlow Keras.**

STEP 02

Layer Implementation

First LSTM Layer:

- ❑ 64 units
- ❑ Dropout (30%) to mitigate overfitting.

Second LSTM Layer:

- ❑ 64 units
- ❑ Dropout (30%)

STEP 03

Output Layer

Dense layer with **linear activation** for regression tasks.

STEP 04

Model Compilation

- ❑ **Optimizer:** Adam for efficient training.
- ❑ **Loss Function:** Mean Squared Error (MSE) for regression.

STEP 05

Model Summary

LSTM Layer 1: (None, 1, 64) | **Params:** 18K | **Dropout 1:** (None, 1, 64)
LSTM Layer 2: (None, 64) | **Params:** 33K | **Dropout 2:** (None, 64)
Dense Layer: (None, 1) | **Params:** 65 | **Total Parameters:** 52K

STEP 06

Conclusion

Optimized LSTM regression model designed to capture sequential patterns while minimizing overfitting. 🚀

Build Optimized LSTM Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Define LSTM model architecture
model = Sequential([
    LSTM(64, activation='tanh', return_sequences=True, input_shape=(1, len(num_cols))),
    Dropout(0.3),
    LSTM(64, activation='tanh'),
    Dropout(0.3),
    Dense(1, activation='linear')  # Linear activation for regression
])

# Compile the model using Adam optimizer and Mean Squared Error loss function
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Print confirmation message
print("Optimized LSTM regression model with two layers has been built and compiled successfully.\n")

# Print Model Summary
model.summary()
```

```
Optimized LSTM regression model with two layers has been built and compiled successfully.
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 1, 64)	18,432
dropout (Dropout)	(None, 1, 64)	0
lstm_1 (LSTM)	(None, 64)	33,024
dropout_1 (Dropout)	(None, 64)	0
dense (Dense)	(None, 1)	65

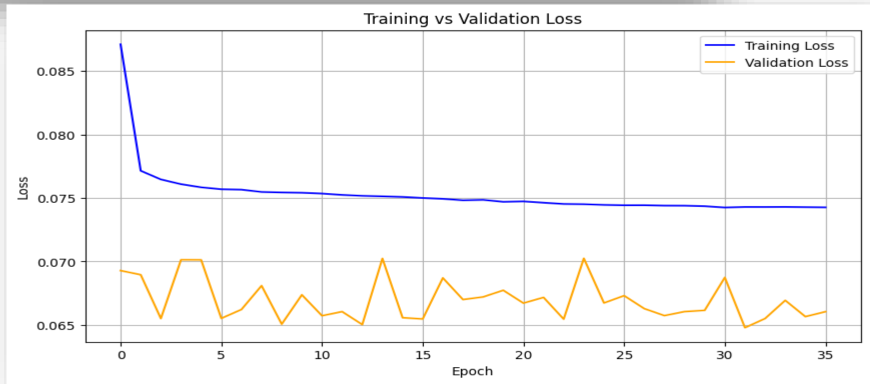
Total params: 51,521 (201.25 KB)
Trainable params: 51,521 (201.25 KB)
Non-trainable params: 0 (0.00 B)

LSTM Model Training & MLflow Logging

Training the LSTM Model and Logging with MLflow

```
# Uninstall & Re-Install the mlflow Library
!pip uninstall mlflow
!pip install mlflow
```

```
# Train Model and Log with MLflow
import mlflow.tensorflow
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping
import os

# Prevent GPU errors in certain environments
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

# Enable automatic logging with MLflow
mlflow.tensorflow.autolog()

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='loss', patience=5, restore_best_weights=True)

# Train the model and save history
with mlflow.start_run():
    history = model.fit(X_scaled, y_scaled, epochs=100, batch_size=128, validation_split=0.2, callbacks=[early_stop])
    mlflow.tensorflow.log_model(model, "lstm_model")

# Plot Training vs Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Training vs Validation Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.grid(True)
plt.show()
```



01 **Model**

LSTM for future sales prediction using time series data.

02 **Tracking**

Utilized MLflow autologging for performance metrics.

03 **Key Techniques**

EarlyStopping (patience = 5) to prevent overfitting.

Disabled GPU: os.environ["CUDA_VISIBLE_DEVICES"] = "-1".

Trained for 100 epochs, batch size of 128, 20% validation split.

04 **Outcomes**

Training Loss: Decreased steadily; final: 0.0743 (MAE: 0.2110).

Validation Loss: Stabilized after initial fluctuations; final: 0.0660 (MAE: 0.1970).

05 **Analysis**

Training Curve: Smooth decline, indicating effective learning.

Validation Curve: Initial spikes but overall stabilization, suggesting good generalization.

06 **Key Takeaways**

✅ Effective learning with minimal overfitting.

✅ Minor validation fluctuations are expected but manageable.

07 **Conclusion**

LSTM model captured sequential patterns well; training logged with MLflow for reproducibility. 🚀

Determining Last Available Date & Saving the Model & Scaler

1

Overview

Developed an LSTM model to predict future sales based on historical data.

2

Model & Scaler Saving

- ❑ Saved LSTM model as **lstm.keras**.
- ❑ MinMaxScaler objects saved as **scaler_X.pkl** and **scaler_y.pkl**

3

Last Training Data Date

Last available date in training data is: 2015-07-31

Determine the last available date in training data

```python
import pandas as pd

# Check the first few rows of the Date column
print(train_df['Date'].head())

# Check the data type of the Date column
print(train_df['Date'].dtype)

# Step 2: Identify the last date
last_date = train_df['Date'].max()
print("\nLast Date in Train Data:", last_date)
```

```
0    2015-07-31
1    2015-07-31
2    2015-07-31
3    2015-07-31
4    2015-07-31
Name: Date, dtype: datetime64[ns]
datetime64[ns]

Last Date in Train Data: 2015-07-31 00:00:00
```

Saving the Trained LSTM Model and Scaler

```python
# Save trained model and scalers
import pickle

# Define filenames
model_filename = "lstm.keras"
scaler_x_filename = "scaler_X.pkl"
scaler_y_filename = "scaler_y.pkl"

# Save the model and scalers
model.save(model_filename)
with open(scaler_x_filename, "wb") as f:
    pickle.dump(scaler_X, f)
with open(scaler_y_filename, "wb") as f:
    pickle.dump(scaler_y, f)

# Confirm saving with formatted strings using placeholders
print(f"Model '{model_filename}' saved successfully.")
print(f"Scaler '{scaler_x_filename}' saved successfully.")
print(f"Scaler '{scaler_y_filename}' saved successfully.")
```

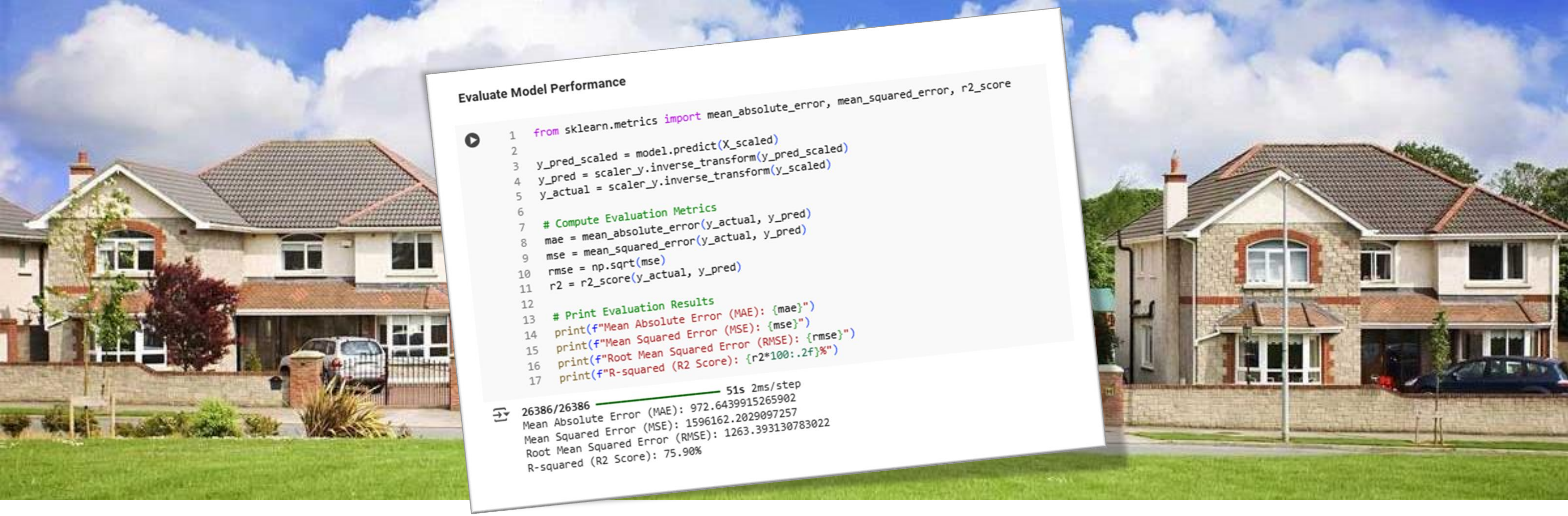
```
Model 'lstm.keras' saved successfully.
Scaler 'scaler_X.pkl' saved successfully.
Scaler 'scaler_y.pkl' saved successfully.
```

Predict Next 6 Weeks Sales

```python
from datetime import timedelta

# Determine the last available date in training data
last_date = train_df['Date'].max()

# Generate future dates for prediction (next 6 weeks = 42 days)
pred_dates = [last_date + timedelta(days=i) for i in range(1, 43)]

# Make predictions
y_pred_scaled = model.predict(X_scaled)
y_pred = scaler_y.inverse_transform(y_pred_scaled)

# Print lengths for debugging
print(f"Length of pred_dates: {len(pred_dates)}")
print(f"Length of y_pred: {len(y_pred)}")

# Print future dates alongside predicted values
print("\nFuture Prediction Dates and Predicted Sales:")
for i in range(min(len(pred_dates), len(y_pred))):  # Adjusting to the minimum length
    print(f"{pred_dates[i]} - Predicted Sales: {y_pred[i]}")
```

```
26386/26386 ━━━━━━━━━━━━━━━━━━━━ 60s 2ms/step
Length of pred_dates: 42
Length of y_pred: 844338
```

```
Future Prediction Dates and Predicted Sales:
2015-08-01 00:00:00 - Predicted Sales: [6088.965]
2015-08-02 00:00:00 - Predicted Sales: [6720.2705]
2015-08-03 00:00:00 - Predicted Sales: [8231.525]
2015-08-04 00:00:00 - Predicted Sales: [11670.013]
2015-08-05 00:00:00 - Predicted Sales: [6125.136]
2015-08-06 00:00:00 - Predicted Sales: [6401.5405]
2015-08-07 00:00:00 - Predicted Sales: [11404.154]
2015-08-08 00:00:00 - Predicted Sales: [8313.937]
2015-08-09 00:00:00 - Predicted Sales: [7229.507]
2015-08-10 00:00:00 - Predicted Sales: [7181.1855]
2015-08-11 00:00:00 - Predicted Sales: [10688.756]
2015-08-12 00:00:00 - Predicted Sales: [9173.822]
2015-08-13 00:00:00 - Predicted Sales: [6410.3784]
2015-08-14 00:00:00 - Predicted Sales: [7408.187]
2015-08-15 00:00:00 - Predicted Sales: [7830.8154]
2015-08-16 00:00:00 - Predicted Sales: [9279.649]
2015-08-17 00:00:00 - Predicted Sales: [9069.689]
2015-08-18 00:00:00 - Predicted Sales: [9004.695]
2015-08-19 00:00:00 - Predicted Sales: [7468.012]
2015-08-20 00:00:00 - Predicted Sales: [9311.368]
2015-08-21 00:00:00 - Predicted Sales: [7184.624]
2015-08-22 00:00:00 - Predicted Sales: [6954.079]
2015-08-23 00:00:00 - Predicted Sales: [6127.023]
2015-08-24 00:00:00 - Predicted Sales: [9891.59]
2015-08-25 00:00:00 - Predicted Sales: [11897.265]
2015-08-26 00:00:00 - Predicted Sales: [6588.5454]
2015-08-27 00:00:00 - Predicted Sales: [10805.882]
2015-08-28 00:00:00 - Predicted Sales: [7027.8447]
2015-08-29 00:00:00 - Predicted Sales: [7608.9785]
2015-08-30 00:00:00 - Predicted Sales: [6708.262]
2015-08-31 00:00:00 - Predicted Sales: [6877.4985]
2015-09-01 00:00:00 - Predicted Sales: [6965.9214]
2015-09-02 00:00:00 - Predicted Sales: [8861.8955]
2015-09-03 00:00:00 - Predicted Sales: [10320.075]
2015-09-04 00:00:00 - Predicted Sales: [7903.761]
2015-09-05 00:00:00 - Predicted Sales: [10765.211]
2015-09-06 00:00:00 - Predicted Sales: [8680.574]
2015-09-07 00:00:00 - Predicted Sales: [6557.0664]
2015-09-08 00:00:00 - Predicted Sales: [8951.579]
2015-09-09 00:00:00 - Predicted Sales: [7144.589]
2015-09-10 00:00:00 - Predicted Sales: [5707.478]
2015-09-11 00:00:00 - Predicted Sales: [10674.02]
```


Evaluate Model Performance

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred_scaled = model.predict(X_scaled)
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_actual = scaler_y.inverse_transform(y_scaled)

# Compute Evaluation Metrics
mae = mean_absolute_error(y_actual, y_pred)
mse = mean_squared_error(y_actual, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_actual, y_pred)

# Print Evaluation Results
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2 Score): {r2*100:.2f}%")
```

```
26386/26386 ━━━━━━━━━━━━━━━━━━━━ 51s 2ms/step
Mean Absolute Error (MAE): 972.6439915265902
Mean Squared Error (MSE): 1596162.2029097257
Root Mean Squared Error (RMSE): 1263.393130783022
R-squared (R2 Score): 75.90%
```

FUTURE PREDICTIONS

- Generated dates for the next 6 weeks (42 days): from 2015-08-01 to 2015-09-11.
- Predictions scaled back to original values.

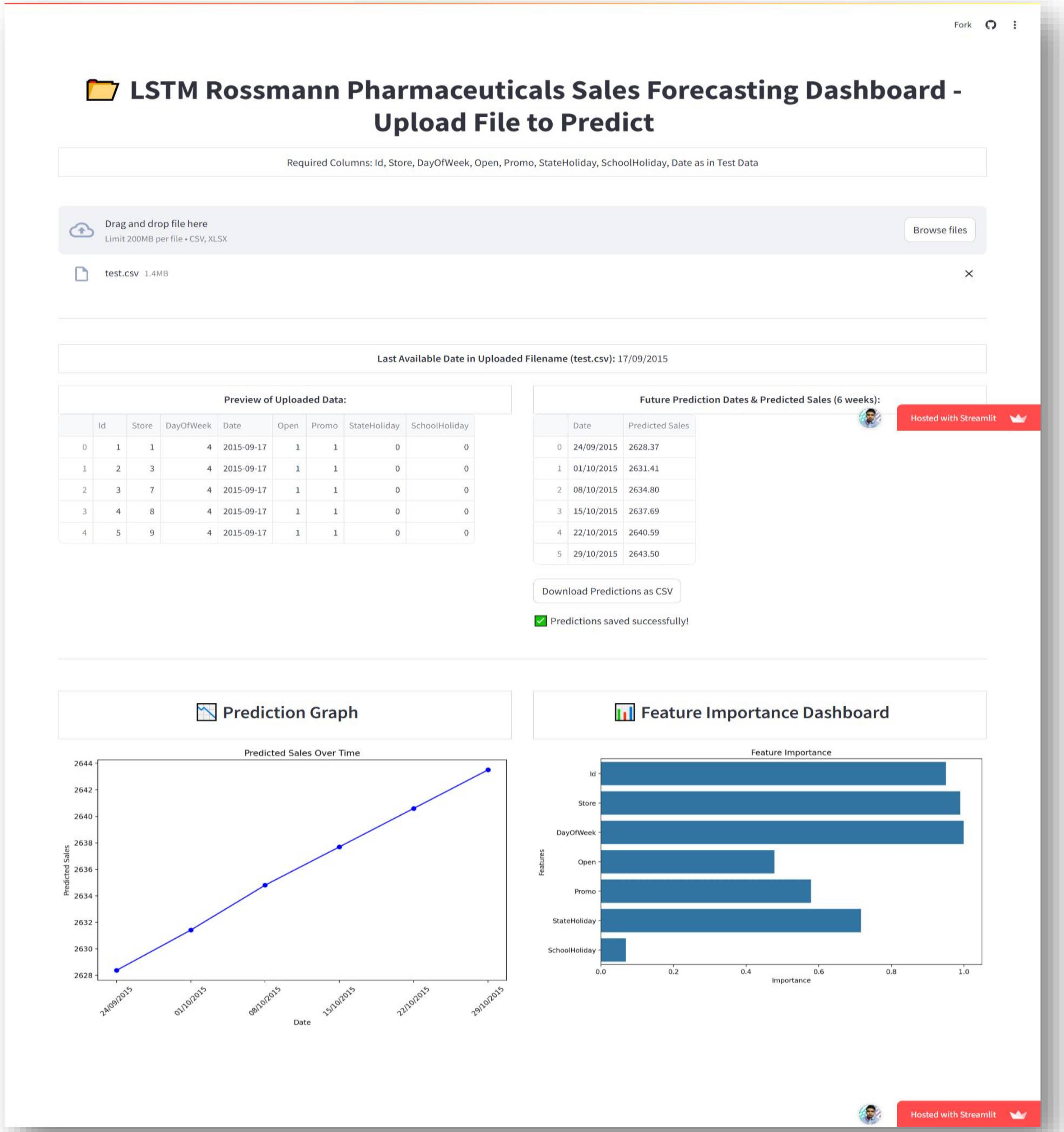
PERFORMANCE METRICS

- **MAE:** 972.64
- **MSE:** 1,596,162.20
- **RMSE:** 1,263.39
- **R^2 Score:** 75.90%

KEY TAKEAWAYS

- Model explains 75.90% of variance; reasonable error margins.
- Ready for future sales forecasting. 🚀

Interactive Sales Forecasting with Streamlit and LSTM



1

Dashboard Overview

Developed & Deployed an interactive dashboard for forecasting sales using LSTM neural networks & Streamlit.

2

Technologies Used

- ❑ **Streamlit:** Web app framework for user interface.
- ❑ **Pandas:** Data manipulation and analysis.
- ❑ **NumPy:** Numerical operations.
- ❑ **TensorFlow/Keras:** LSTM model handling.
- ❑ **Scikit-learn:** Data preprocessing (MinMaxScaler).
- ❑ **Matplotlib & Seaborn:** Data visualization.
- ❑ **MLflow:** Model tracking and management.

3

Key Features

- ❑ **Last Most Recent Date:** Displays initial rows for user verification.
- ❑ **Uploaded Data Preview:** Shows the latest date in the dataset.
- ❑ **Predictions Download as csv:** Option to download predictions as CSV.
- ❑ **Prediction Graph:** Visualizes predicted sales trends.
- ❑ **Feature Importance Dashboard:** Displays feature importance.

4

User Interface

- ❑ Clear instructions for users regarding data requirements.
- ❑ Intuitive layout with separate sections for data preview, predictions, and visualizations.

I appreciate your time and attention. 😊

I hope this presentation was informative and helpful.

Please don't hesitate to contact me with any questions or feedback you may have.

I am more than eager to hear your thoughts and suggestions. 🙂