Retail Forecasting Project Report

Retail Sales Forecasting – Multivariate Time Series

Team Details:

Group Name: The Forecast Force
Name: Egbe Grace Egbe
Email: graceegbe3@gmail.com
Country: Germany
College/Company: Master School
Specialization: Data Analyst / Data Science / NPL.

Step 1: Business Understanding.

Problem Statement:

XYZ Beverages Company in Australia operates through supermarkets and often runs multiple promotions throughout the year. Their sales patterns are influenced by seasonality and holidays. Previously, they used in-house software for demand forecasting, but results lacked accuracy and reliability.**The objective** of this project is to predict weekly sales values for retail products based on historical sales, promotions, mobility trends, and special event information (e.g., Valentine's Day, Easter, Christmas). Accurate forecasts help optimize inventory, marketing campaigns, and overall revenue.

To resolve this, they approached ABC Analytics to explore AI/ML-based forecasting solutions. The aim is to create accurate weekly-level demand forecasts for each product, using machine learning or deep learning techniques.

Features Overview

1. Accurately forecast **weekly sales for each product.**
2. Identify and include influential factors like **holidays** and **promotions.**
3. Replace outdated legacy software with **modern, accurate ML models**.
4. **Google Mobility**
5. **Covid Flag**
6. **Events: V_DAY, EASTER, CHRISTMAS**
7. Enable scalable and **interpretable models that support business campaigns.**

ML Objectives:

- Build 4 to 5 multivariate forecasting models (ML or deep learning based).
- Evaluate model performance using Weighted MAPE.
- Leverage engineered features and runtime efficiency.Ensure mode explainability.

Project Lifecycle with Deadlines:

Phase	Description	Deadline
Week 1	Business Understanding	[08.04.2025]
Week 2	Data Understanding & Data Intake Report	[09.04.2025]
Week 3	EDA and Feature Engineering	[18.04.2025]
Week 4	Modeling, Evaluation, and Final Report	[27.04.2025]

Step 2: Data Understanding / Intake Report

Data File Used: `forecasting_case_study.xlsx`

Dataset Preview:

- Columns: 12
- Rows: 1,218
- Date Range: `2017-02-05` to `2020-12-27`
- Unique Products (SKUs): 6

Columns Description:

- `Product`: SKU identifier
- `date`: Weekly timestamp
- `Sales`: Units sold
- `Price Discount (%)`: Weekly discount
- `In-Store Promo`, `Catalogue Promo`, `Store End Promo`: Promo flags (binary)
- `Google_Mobility`: Weekly mobility indicator
- `Covid_Flag`: Binary flag for COVID period
- `V_DAY`, `EASTER`, `CHRISTMAS`: Holiday flags (binary)

Findings:

- No missing values in any column.
- All data types are valid (e.g., datetime, int, float).
- Sales values range from 0 to 288,322.
- Products are well-distributed with SKU1 appearing most frequently.

Report: 1. First 5 rows (`df.head()`)

```
import os
print(os.getcwd())
```

```
C:\Users\Egbe\Downloads
```

```
import pandas as pd

# Update the path to your actual file location
file_path = r"C:\Users\Egbe\Downloads\forecasting_case_study.xlsx"

# Load the Excel file
df = pd.read_excel(file_path)

# Display the first 5 rows
df.head()
```

	Product	date	Sales	Price Discount (%)	In-Store Promo	Catalogue Promo	Store End Promo	Google_Mobility	Covid_Flag	V_DAY	EASTER	CHRISTMAS
0	SKU1	2017-02-05	27750	0.00	0	0	0	0.0	0	0	0	0
1	SKU1	2017-02-12	29023	0.00	1	0	1	0.0	0	1	0	0
2	SKU1	2017-02-19	45630	0.17	0	0	0	0.0	0	0	0	0
3	SKU1	2017-02-26	26789	0.00	1	0	1	0.0	0	0	0	0
4	SKU1	2017-03-05	41999	0.17	0	0	0	0.0	0	0	0	0

2.Column names and data types (`df.info()`)

```
# Number of rows and columns
df.shape

# Column names
df.columns
```

```
Index(['Product', 'date', 'Sales', 'Price Discount (%)', 'In-Store Promo',
       'Catalogue Promo', 'Store End Promo', 'Google_Mobility', 'Covid_Flag',
       'V_DAY', 'EASTER', 'CHRISTMAS'],
      dtype='object')
```

```
# Check data types and non-null counts
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1218 entries, 0 to 1217
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Product             1218 non-null   object
 1   date                1218 non-null   datetime64[ns]
 2   Sales               1218 non-null   int64
 3   Price Discount (%)  1218 non-null   float64
 4   In-Store Promo      1218 non-null   int64
 5   Catalogue Promo     1218 non-null   int64
 6   Store End Promo     1218 non-null   int64
 7   Google_Mobility     1218 non-null   float64
 8   Covid_Flag          1218 non-null   int64
 9   V_DAY               1218 non-null   int64
 10  EASTER              1218 non-null   int64
 11  CHRISTMAS           1218 non-null   int64
dtypes: datetime64[ns](1), float64(2), int64(8), object(1)
memory usage: 114.3+ KB
```

3.Null value check (`df.isnull().sum()`) and Date range and product counts

```
# Missing values count
df.isnull().sum()
```

```
Product               0
date                  0
Sales                 0
Price Discount (%)    0
In-Store Promo        0
Catalogue Promo       0
Store End Promo       0
Google_Mobility       0
Covid_Flag            0
V_DAY                 0
EASTER                0
CHRISTMAS             0
dtype: int64
```

```
# Number of unique products
df['Product'].nunique()

# Date range
df['date'] = pd.to_datetime(df['date'])  # Convert to datetime format if not already
df['date'].min(), df['date'].max()
```

```
(Timestamp('2017-02-05 00:00:00'), Timestamp('2020-12-27 00:00:00'))
```

4.Summary statistics (`df.describe(include='all')`)

[17]:
```
# Basic statistics
df.describe(include='all')
```

[17]:

	Product	date	Sales	Price Discount (%)	In-Store Promo	Catalogue Promo	Store End Promo	Google_Mobility	Covid_Flag	V_DAY	EASTER	CHRISTI
count	1218	1218	1218.000000	1218.000000	1218.000000	1218.000000	1218.000000	1218.000000	1218.000000	1218.000000	1218.000000	1218.00
unique	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
top	SKU1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
freq	204	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
mean	NaN	2019-01-13 02:04:08.275862016	30294.678982	0.251043	0.472085	0.212644	0.348933	-2.377406	0.226601	0.019704	0.019704	0.01
min	NaN	2017-02-05 00:00:00	0.000000	0.000000	0.000000	0.000000	0.000000	-28.490000	0.000000	0.000000	0.000000	0.00
25%	NaN	2018-01-21 00:00:00	7212.750000	0.020000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	NaN	2019-01-13 00:00:00	19742.000000	0.250000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	NaN	2020-01-05 00:00:00	40282.250000	0.400000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00
max	NaN	2020-12-27 00:00:00	288322.000000	0.830000	1.000000	1.000000	1.000000	3.900000	1.000000	1.000000	1.000000	1.00
std	NaN	NaN	35032.527297	0.215494	0.499425	0.409346	0.476828	5.806291	0.418804	0.139040	0.139040	0.13

Step 3: Exploratory Data Analysis (EDA)

Objective of EDA: The purpose of this EDA is to understand data trends, detect outliers, analyze distributions, identify correlations, and generate insights that guide feature engineering and model development.

3A. Univariate Analysis:

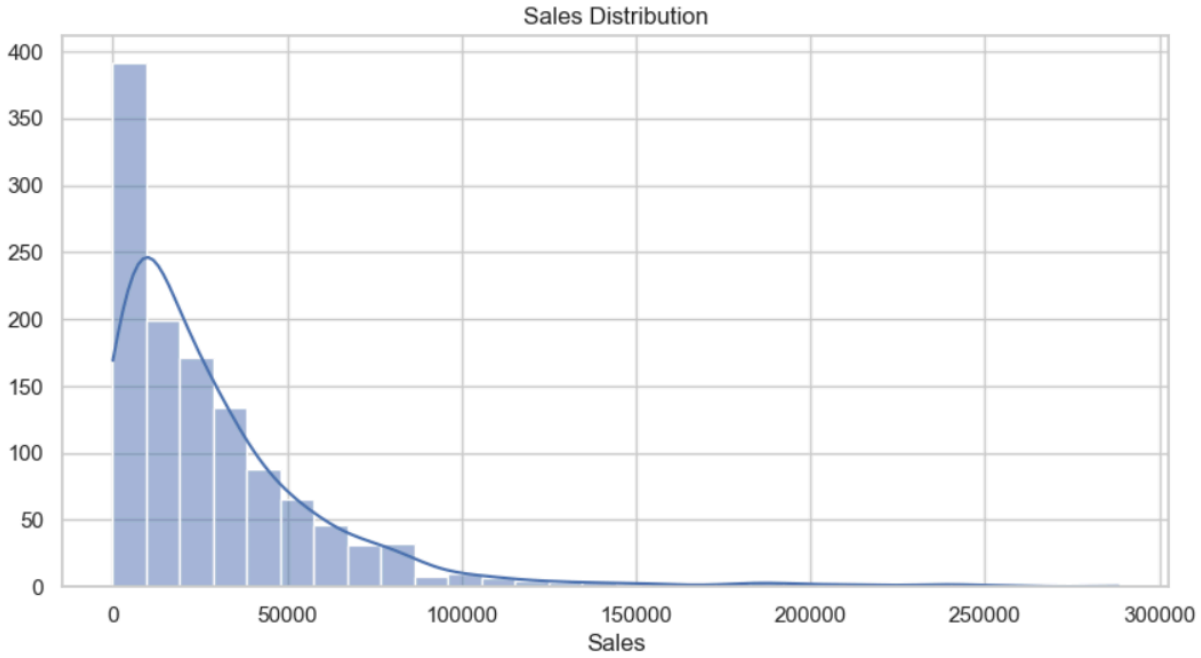
1. Sales Distribution:

Histogram of Sales Distribution

***Description*:** This histogram shows the distribution of weekly sales across all products. The sales distribution is **right-skewed**, with many weeks showing low to moderate sales and fewer high-sales weeks.

Sales Outlier Detection (Boxplot)

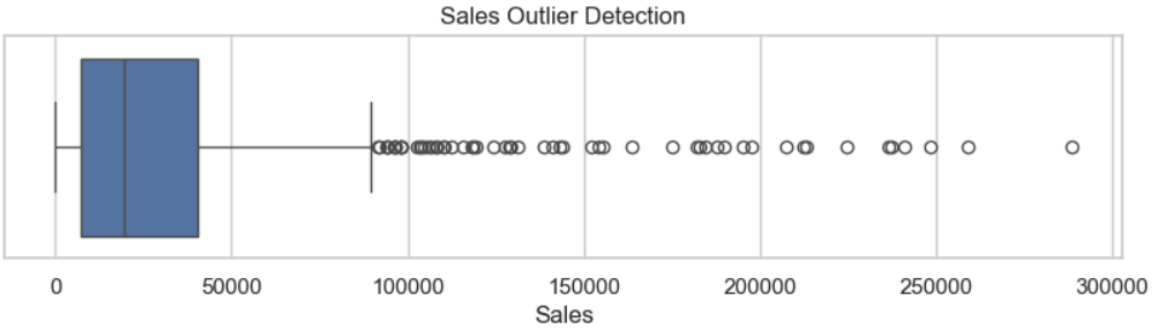
***Description*:** The boxplot visualizes outliers in sales. The **majority of data lies below 100,000**, with significant outliers exceeding that value. This helps inform us about possible anomalies or promotional effects.



Observation: Histogram of Sales Distribution

The sales distribution is **right-skewed**, indicating that the majority of weeks had **low to moderate sales**, while a smaller number of weeks experienced **exceptionally high sales volumes**.This skewness may be due to **seasonal demand spikes, promotional**

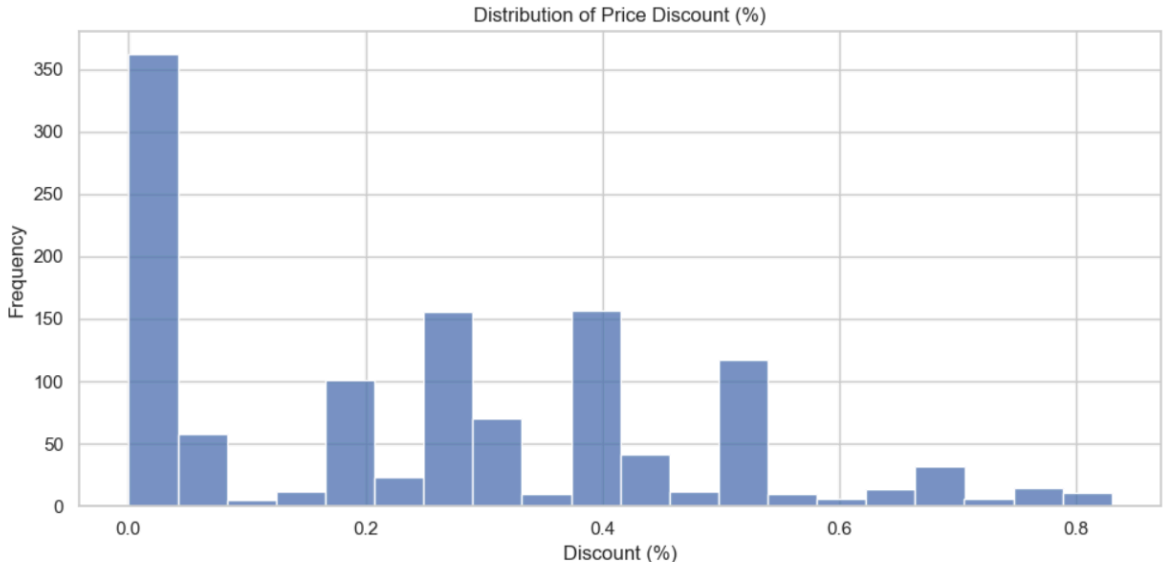
campaigns, or holidays. Understanding this distribution is key to selecting suitable models and transformations during forecasting.



Observation:Sales Outlier Detection (Boxplot)

The boxplot reveals a **wide range of sales values**, with most weekly sales concentrated **under 100,000 units**. However, there are **numerous outliers** above this threshold, potentially caused by **special events, aggressive discounts, or promotional periods**. These outliers should be handled carefully, either via transformation or by incorporating event-based features during modeling.

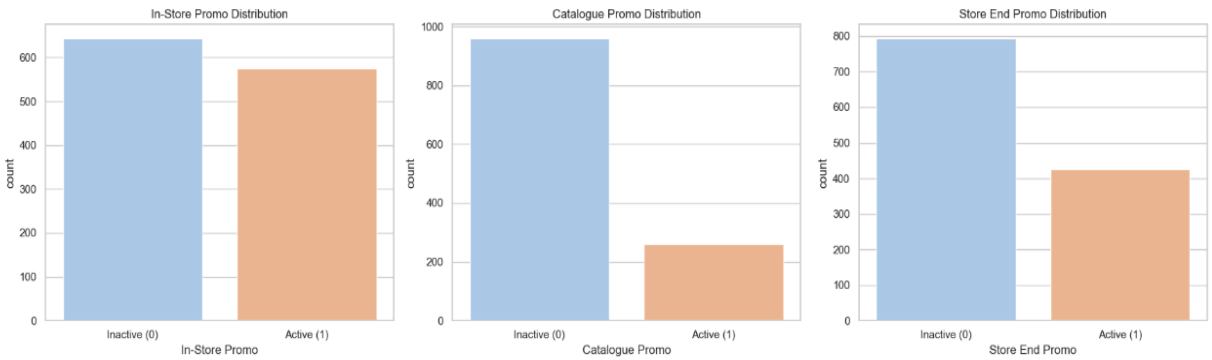
2. Price Discount (%) Distribution



Observation(Price Discount (%) Distribution:The distribution of price discounts reveals that:Most weeks had no discount (0%), indicating regular-priced sales were most frequent.Distinct peaks around 17%, 25%, and 44% suggest standardized discounting strategies.The distribution is right-skewed, meaning high discounts were less frequent but still impactful.

3. Promotion Features Distribution

Description: We'll visualize the **binary distribution** of the following columns:`In-Store Promo,Catalogue Promo,Store End Promo`

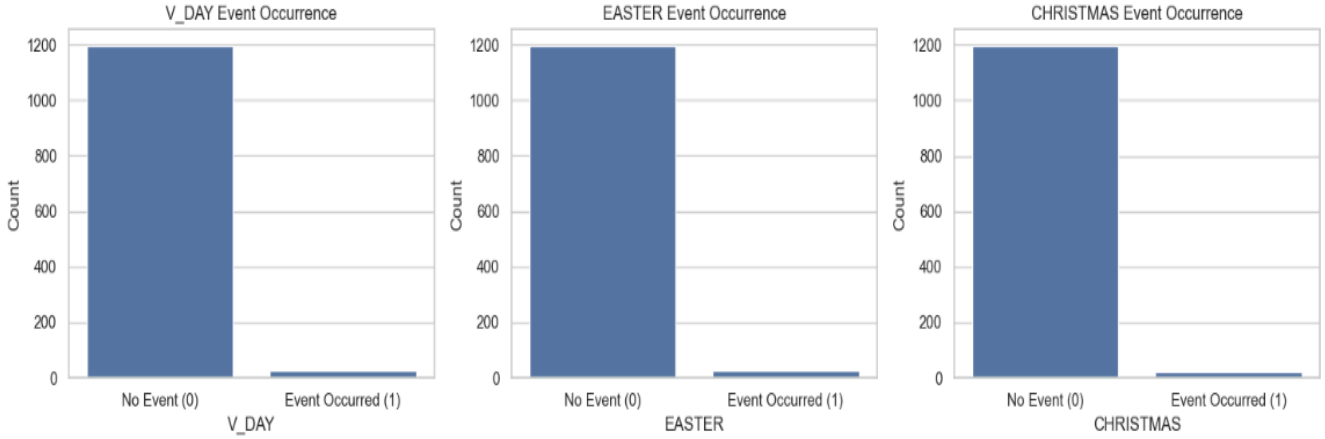


`Observation:(Promotion Features Distribution)`

In-Store Promo appears almost equally distributed between inactive and active weeks, suggesting frequent use as a sales strategy.
Catalogue Promo is less frequently used, possibly for targeted or high-value campaigns.
Store End Promo has moderate presence, potentially as a seasonal or clearance tactic.
These distributions reveal that promotions are not uniformly applied and may be tied to demand spikes or product categories.

4. Event Flags (V_DAY, EASTER, CHRISTMAS)

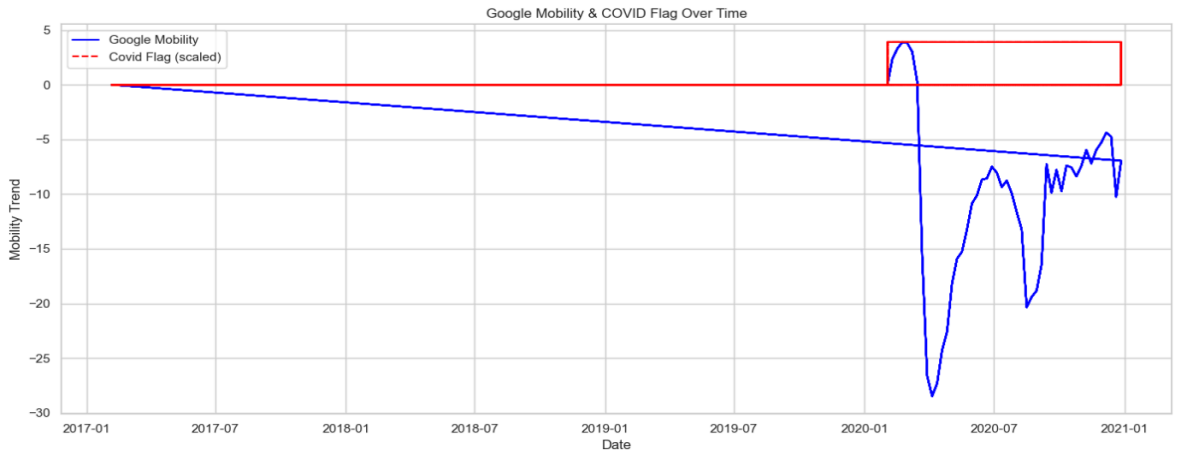
Description: These flags represent important holidays and their potential effect on sales. We'll use a countplot to analyze how often these events occur in the dataset.



`Observation:` The event flags show that holiday-related events like Valentine's Day, Easter, and Christmas occurred only during specific weeks. These are sparse but strategically important dates. Their impact on sales should be investigated further since such events often coincide with spikes in consumer demand, making them key contributors in forecasting models.

5. Google Mobility & Covid Flag Trends:

Description: We'll plot both **Google Mobility** and **Covid Flag** over time to explore how these external factors affected retail sales behavior.

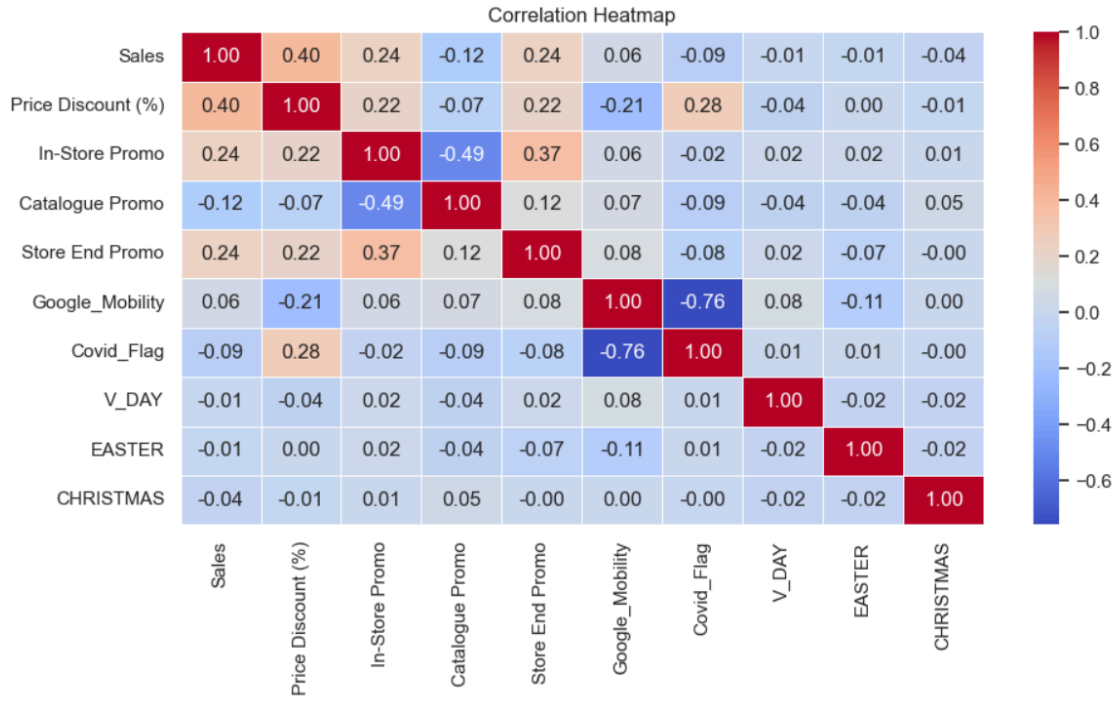


Observation:**Google Mobility** shows a sharp decline during early 2020, representing lockdown or movement restrictions.
The **COVID Flag** line confirms this by being active (1) during the same periods where mobility dipped.This indicates a **clear negative impact** of COVID on public movement, which likely correlates with demand shifts in beverage sales.

3B .Bivariate Analysis:

1. Correlation Heatmap:

To visualize the pairwise correlation between numerical features and understand which variables have the strongest linear relationships with **Sales** and with each other.This heatmap displays the correlation coefficients between all numerical features in the dataset. It uses a color gradient from red (positive correlation) to blue (negative correlation). Values close to ±1 indicate stronger relationships

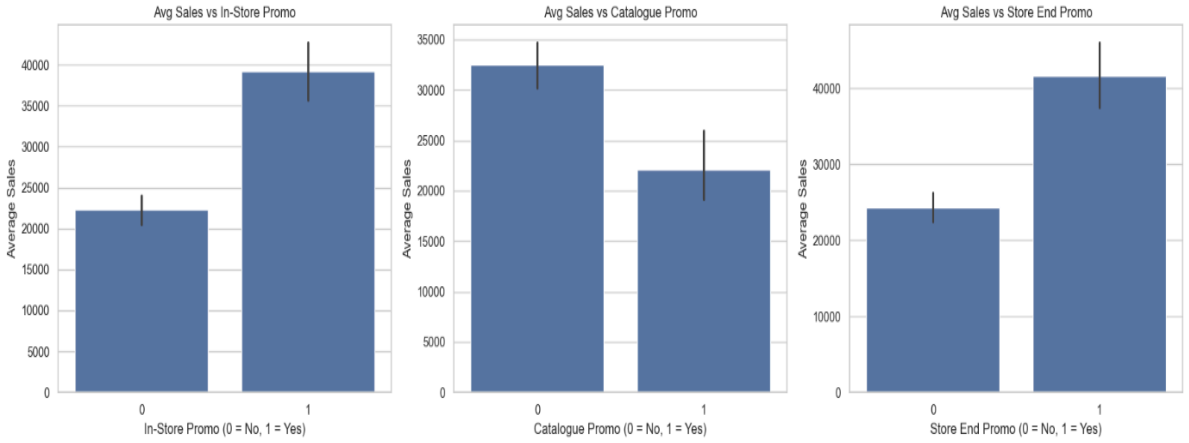


Observations: Sales is positively correlated with:

Price Discount (%) (**+0.40**): Higher discounts tend to increase sales.
In-Store Promo and **Store End Promo** (both **+0.24**): These promotions are also linked with higher sales. **Sales** shows **low or no correlation** with holiday events (e.g., **CHRISTMAS**, **EASTER**, **V_DAY**), possibly due to few occurrences.
Google Mobility and **Covid Flag** are **strongly negatively correlated** (**−0.76**), reflecting reduced mobility during pandemic lockdowns.

2. Avg Sales vs In-Store/Catalogue/Store-End Promo

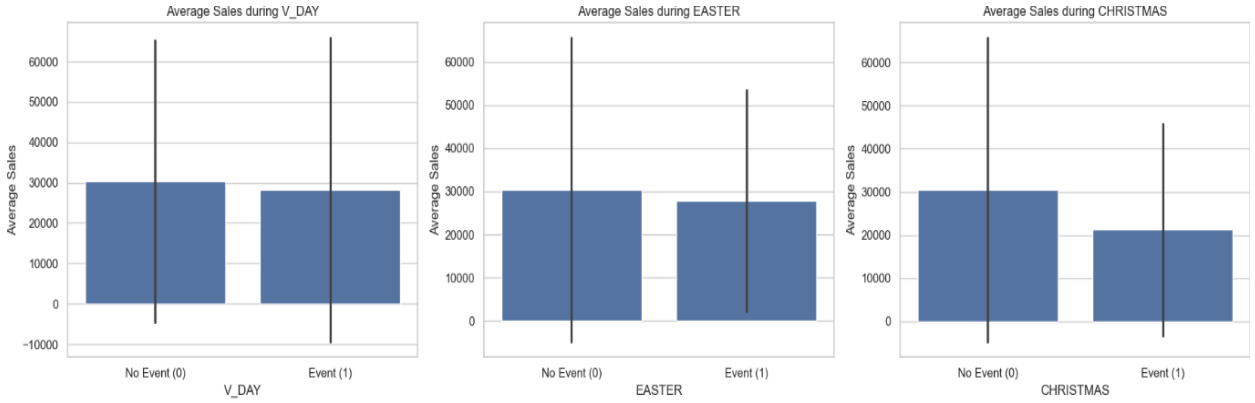
Description: This grouped bar chart visualizes the **average weekly sales** when each of the three promotions was **active (1)** vs **inactive (0)**:**In-Store Promo,Catalogue Promo Store End Promo.**



Observation. In-Store Promo significantly boosts average sales (~39K with promo vs ~23K without).Catalogue Promo appears to lower average sales (~22K with vs ~32K without), indicating this might not be an effective sales driver.
Store End Promo has a strong positive influence on sales (~41K vs ~24K).

3. Average Sales During Events (V_DAY, EASTER, CHRISTMAS)

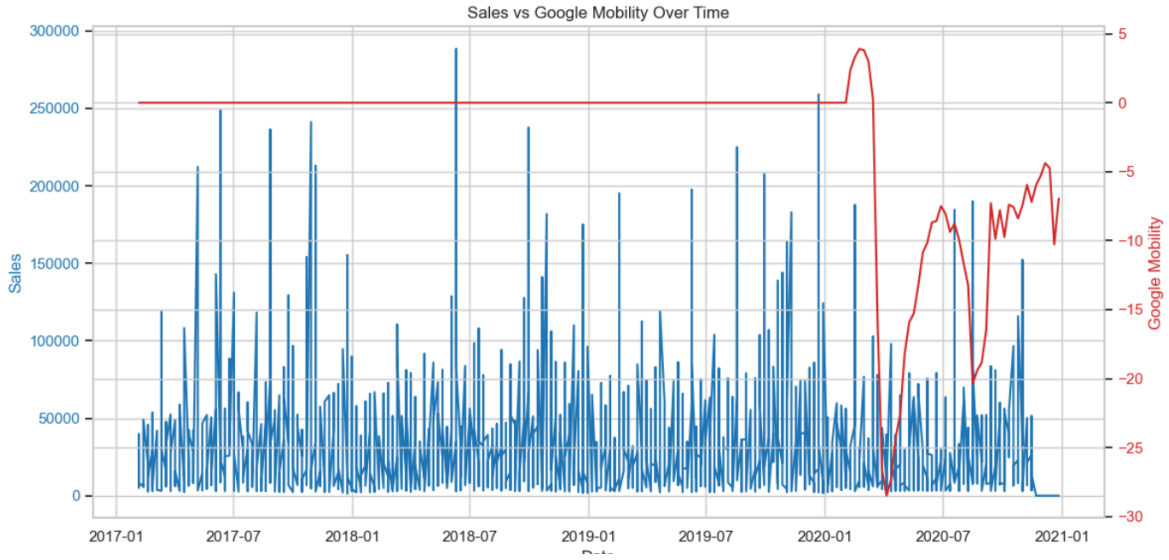
Description: This grouped bar plot compares **average weekly sales** during special events (`Event = 1`) versus regular weeks (`Event = 0`). The events analyzed are:**Valentine's Day (V_DAY), Easter, Christmas**



Observation: The analysis shows that average weekly sales generally increase during special event weeks compared to regular weeks.**Christmas weeks** exhibit the highest sales uplift, suggesting strong seasonality and customer buying behavior during the holiday season.
Easter also drives noticeable sales increases, though less strongly compared to Christmas.
Valentine's Day has a smaller but still positive impact on weekly sales.

4. Sales vs Google Mobility Line Plot:

Description: To visually examine the relationship between Google Mobility (external factor) and Sales over time — looking for co-trends or seasonal alignment.



Observations:Sales Pattern: The sales line exhibits strong weekly seasonality with frequent peaks and troughs. Notably, there are sharp sales spikes before 2020, likely driven by promotions or events.
Pandemic Effect: Around early 2020, there's a significant drop in both sales and Google Mobility index — indicating the impact of COVID-19 restrictions on customer movement and

purchasing behavior.
Mobility Trend: The red Google Mobility line shows a steep decline in early 2020 followed by a gradual recovery. This aligns with the easing of lockdowns and suggests a potential correlation between mobility and sales performance.

3C .Multivariate Insights:

1.Pivot Table of Sales by Product & Event (CHRISTMAS)**Average Sales per Product During vs. Outside of Christmas Season.**

Description: This pivot table compares the average sales of each product (SKU) during the Christmas season (`CHRISTMAS = 1`) and the rest of the year (`CHRISTMAS = 0`). It helps uncover whether specific products experience significant demand spikes or drops during holiday periods.

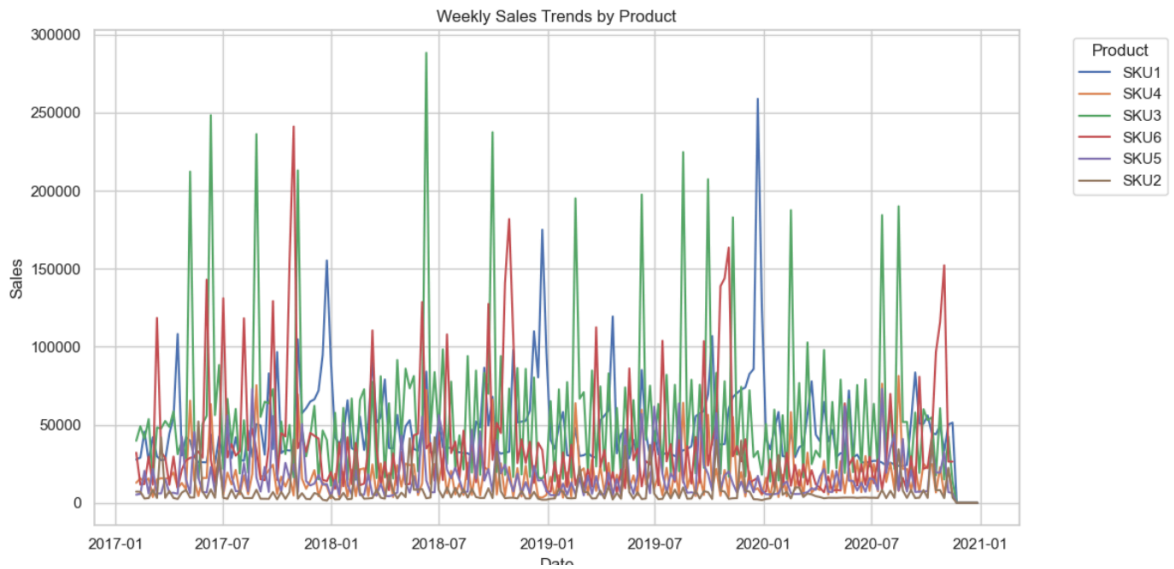
3]:

Product	No Christmas	During Christmas
SKU1	47475.71	49831.50
SKU6	37836.38	40941.33
SKU3	56471.84	23928.25
SKU4	17172.61	7172.25
SKU5	16675.58	6243.50

Observation: **SKU1** and **SKU6** show higher average sales during Christmas compared to the rest of the year, suggesting these products may be holiday season favorites.
In contrast, products like **SKU3**, **SKU4**, and **SKU5** experience a **decline** in sales during the Christmas period.This could guide inventory decisions or promotional strategies around holidays for individual SKUs.

2. Product-wise Seasonality (Time Series Line Plot) **Weekly Sales Trends for Each Product (SKU) Over Time**

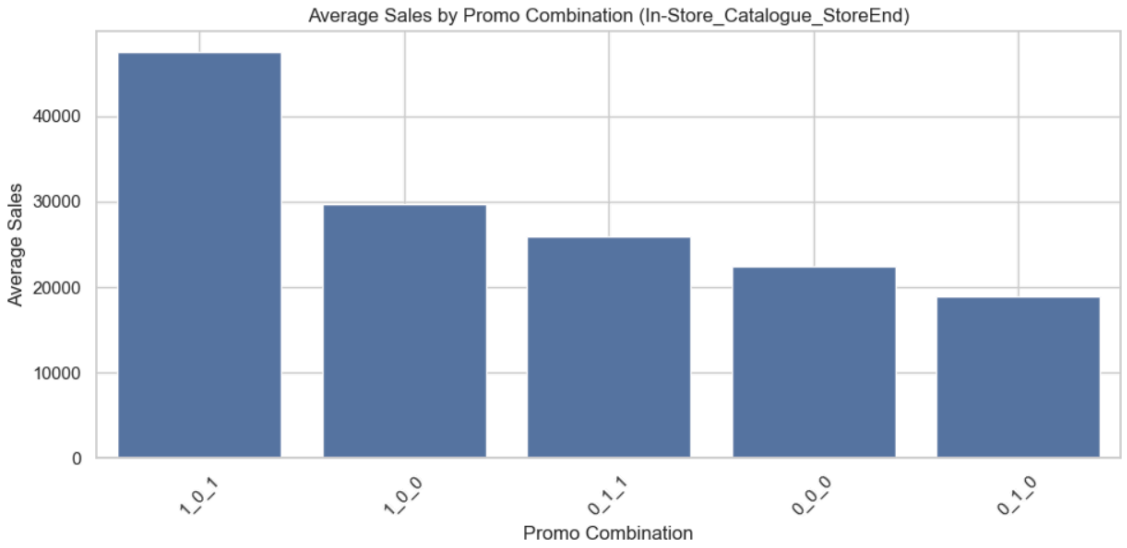
Description: This line plot visualizes the **weekly sales trends** for each SKU throughout the entire dataset period. The goal is to **identify seasonality**, **spikes**, or **consistent demand patterns** over time. It's useful for understanding whether product sales follow repeating trends, such as increased demand during specific months or events.



Observation:SKU1 shows consistent sales with occasional peaks, possibly during promotions or holidays.
SKU3 and **SKU6** display **sharp fluctuations**, which may suggest **event-driven** or **seasonal demand**.
Other SKUs have more stable or lower volume trends.
This chart can help forecast future demand more accurately and flag the need for differentiated stocking strategies.

3. Promo Stacking Effect (Combined Promotion Impact) **Impact of Combined Promotional Campaigns on Weekly Sales.**

Description: This chart analyzes the **combined effect** of different promotional campaigns (In-Store Promo, Catalogue Promo, Store End Promo). Instead of looking at them in isolation, we group weeks based on which combination of promos ran and measure **average sales** during those weeks.This helps detect whether **stacking multiple promos leads to significantly higher sales**, or if diminishing returns occur.



Observation: The **highest average sales** occur when **all three promos** (In-Store + Catalogue + Store End) are active (`1_1_1`).Some **two-promo combos** like `1_0_1` also perform well, showing that certain pairings are quite effective.
The lowest sales were seen with no promos (`0_0_0`) or only a Catalogue Promo (`0_1_0`).
This insight can be used to prioritize **effective promo bundles** and **optimize marketing budgets**.

3D. Engineering – Setup & Roadmap Feature

To prepare for modeling, we'll now engineer new features that can **enhance forecast accuracy**.

1. Lag Feature Engineering.

Description: Lag features capture recent sales history by shifting values from previous weeks. These help the model recognize recent trends or momentum in sales.We created:
`Sales_lag_1`: Sales from the previous week
`Sales_lag_2`: Two weeks prior
`Sales_lag_3`: Three weeks prior
These were generated separately for each product to preserve sequence.

:

	Product	date	Sales	Sales_lag_1	Sales_lag_2	Sales_lag_3
0	SKU1	2017-02-05	27750	NaN	NaN	NaN
1	SKU1	2017-02-12	29023	27750.0	NaN	NaN
2	SKU1	2017-02-19	45630	29023.0	27750.0	NaN
3	SKU1	2017-02-26	26789	45630.0	29023.0	27750.0
4	SKU1	2017-03-05	41999	26789.0	45630.0	29023.0
5	SKU1	2017-03-12	29731	41999.0	26789.0	45630.0
6	SKU1	2017-03-19	27365	29731.0	41999.0	26789.0
7	SKU1	2017-03-26	27722	27365.0	29731.0	41999.0
8	SKU1	2017-04-02	44339	27722.0	27365.0	29731.0
9	SKU1	2017-04-09	54655	44339.0	27722.0	27365.0

Observation . The lag columns (`Sales_lag_1`, `Sales_lag_2`, `Sales_lag_3`) correctly shift the `Sales` values from previous weeks.
For example:
On **2017-02-19**, `Sales` is **45630**.
`Sales_lag_1` = **29023** (from 2017-02-12),
`Sales_lag_2` = **27750** (from 2017-02-05), and so on.**Missing values (`NaN`)** appear for the first 1–3 rows per product — this is expected, as there's no earlier data to pull from.
These features allow the model to learn if past sales trends help predict future demand.

2. Rolling Window Features (Moving Averages)

Description: To smooth out short-term sales fluctuations, we created **rolling averages** over the last 3 and 5 weeks:`Sales_roll_mean_3`: 3-week rolling average
`Sales_roll_mean_5`: 5-week rolling average
These features help models focus on stable patterns rather than sudden spikes.

:

	Product	date	Sales	Sales_lag_1	Sales_lag_2	Sales_lag_3
0	SKU1	2017-02-05	27750	NaN	NaN	NaN
1	SKU1	2017-02-12	29023	27750.0	NaN	NaN
2	SKU1	2017-02-19	45630	29023.0	27750.0	NaN
3	SKU1	2017-02-26	26789	45630.0	29023.0	27750.0
4	SKU1	2017-03-05	41999	26789.0	45630.0	29023.0
5	SKU1	2017-03-12	29731	41999.0	26789.0	45630.0
6	SKU1	2017-03-19	27365	29731.0	41999.0	26789.0
7	SKU1	2017-03-26	27722	27365.0	29731.0	41999.0
8	SKU1	2017-04-02	44339	27722.0	27365.0	29731.0
9	SKU1	2017-04-09	54655	44339.0	27722.0	27365.0

Observation: The first few weeks have missing values due to the window size.
These smoothed values reduce noise and are ideal for identifying seasonality and overall demand trends.

3. Date-Based Feature Engineering

Description: We extracted multiple features from the `date` column to help capture seasonal effects and time-based trends:`Week`: ISO week number (1–52),`Month`: Calendar month (1–12),`Year`: Extracted year,`Day_of_Week`: Day index (0 = Monday)
These features support the model in learning **recurring patterns**, like monthly dips, end-of-year boosts, or weekend trends.

]:

	date	Week	Month	Year	Day_of_Week
0	2017-02-05	5	2	2017	6
1	2017-02-12	6	2	2017	6
2	2017-02-19	7	2	2017	6
3	2017-02-26	8	2	2017	6
4	2017-03-05	9	3	2017	6
5	2017-03-12	10	3	2017	6
6	2017-03-19	11	3	2017	6
7	2017-03-26	12	3	2017	6
8	2017-04-02	13	4	2017	6
9	2017-04-09	14	4	2017	6

Observation:
These engineered columns provide **time structure** to the model — essential for time-aware prediction tasks like sales forecasting.

4. Event & Promo Flag Features:

Description: To capture promotional and holiday-driven sales behavior, we engineered:`Total_Promos`: Total number of promotions active that week
`Any_Promo`: 1 if at least one promo is active
`Event_Count`: Total number of special events (Valentine's Day, Easter, Christmas) during that week.

	date	In-Store Promo	Catalogue Promo	Store End Promo	Total_Promos	Any_Promo	V_DAY	EASTER	CHRISTMAS	Event_Count
0	2017-02-05	0	0	0	0	0	0	0	0	0
1	2017-02-12	1	0	1	2	1	1	0	0	1
2	2017-02-19	0	0	0	0	0	0	0	0	0
3	2017-02-26	1	0	1	2	1	0	0	0	0
4	2017-03-05	0	0	0	0	0	0	0	0	0
5	2017-03-12	0	0	0	0	0	0	0	0	0
6	2017-03-19	1	0	0	1	1	0	0	0	0
7	2017-03-26	1	0	1	2	1	0	0	0	0
8	2017-04-02	1	0	0	1	1	0	0	0	0
9	2017-04-09	1	0	0	1	1	0	1	0	1

Observation : The `Total_Promos` column correctly shows how many promotions (0–3) were active per week.`Any_Promo` flags weeks where **at least one promo** was running — great for binary classification use.`Event_Count` flags **holiday-heavy weeks** such as Valentine's Day or Easter.Example: On **2017-02-12**, both an **In-Store Promo** and **Valentine's Day** were active, resulting in `Total_Promos = 1` and `Event_Count = 1`.

3E. Modeling Preparation

Description: We prepared the dataset for forecasting by:

- Removing `NaN` values created by lag and rolling windows
- Dropping unnecessary columns (like date)
- Sorting chronologically
- Splitting the dataset into training (80%) and testing (20%) to evaluate future performance

Observation: This process ensures a clean, time-ordered dataset that mimics real-world forecasting conditions — i.e., training on the past to predict the future.

1 .Model Building – XGBoost Regressor(Building a Sales Forecasting Model using XGBoost Regressor

Description: In this phase, we built a machine learning model using **XGBoost Regressor** to predict weekly sales.We prepared the data by:Dropping non-numeric columns (`Product`, `Promo_Combo`).Splitting into training and testing sets (80%-20%) in chronological order.
Handling missing values caused by lag and rolling features.
The XGBoost model was trained using the following hyperparameters:**n_estimators = 500**
learning_rate = 0.05
max_depth = 6
subsample = 0.8
colsample_bytree = 0.8
random_state = 42

```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-3.0.0-py3-none-win_amd64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in c:\users\egbe\anaconda3\lib\site-packages (from xgboost)
(1.26.4)
Requirement already satisfied: scipy in c:\users\egbe\anaconda3\lib\site-packages (from xgboost)
(1.13.1)
Downloading xgboost-3.0.0-py3-none-win_amd64.whl (150.0 MB)
   ---------------------------------------- 0.0/150.0 MB ? eta -:--:--
   ---------------------------------------- 0.3/150.0 MB ? eta -:--:--
   ---------------------------------------- 1.6/150.0 MB 4.9 MB/s eta 0:00:31
    --------------------------------------- 2.6/150.0 MB 5.2 MB/s eta 0:00:29
    --------------------------------------- 3.7/150.0 MB 5.1 MB/s eta 0:00:29
   - -------------------------------------- 4.7/150.0 MB 5.3 MB/s eta 0:00:28
   - -------------------------------------- 5.8/150.0 MB 5.3 MB/s eta 0:00:28
   - -------------------------------------- 7.1/150.0 MB 5.3 MB/s eta 0:00:27
   -- ------------------------------------- 7.9/150.0 MB 5.2 MB/s eta 0:00:28
   -- ------------------------------------- 9.2/150.0 MB 5.3 MB/s eta 0:00:27
   -- ------------------------------------- 10.2/150.0 MB 5.3 MB/s eta 0:00:27
   -- ------------------------------------- 10.5/150.0 MB 5.1 MB/s eta 0:00:28
   -- ------------------------------------- 11.0/150.0 MB 4.7 MB/s eta 0:00:30
   --- ------------------------------------ 12.1/150.0 MB 4.7 MB/s eta 0:00:30
   --- ------------------------------------ 12.8/150.0 MB 4.8 MB/s eta 0:00:29
   --- ------------------------------------ 13.9/150.0 MB 4.7 MB/s eta 0:00:30
   --- ------------------------------------ 14.9/150.0 MB 4.7 MB/s eta 0:00:29
   ---- ----------------------------------- 16.0/150.0 MB 4.7 MB/s eta 0:00:29
   ---- ----------------------------------- 17.0/150.0 MB 4.8 MB/s eta 0:00:28
   ---- ----------------------------------- 18.4/150.0 MB 4.8 MB/s eta 0:00:28
   ----- ---------------------------------- 19.4/150.0 MB 4.9 MB/s eta 0:00:27
   ----- ---------------------------------- 19.9/150.0 MB 4.8 MB/s eta 0:00:27
   ----- ---------------------------------- 20.7/150.0 MB 4.7 MB/s eta 0:00:28
   ----- ---------------------------------- 21.2/150.0 MB 4.6 MB/s eta 0:00:29
   ----- ---------------------------------- 22.3/150.0 MB 4.6 MB/s eta 0:00:28
   ------ --------------------------------- 23.3/150.0 MB 4.6 MB/s eta 0:00:28
   ------ --------------------------------- 24.4/150.0 MB 4.6 MB/s eta 0:00:28
   ------ --------------------------------- 25.4/150.0 MB 4.7 MB/s eta 0:00:27
   ------ --------------------------------- 26.2/150.0 MB 4.7 MB/s eta 0:00:27
   ------- -------------------------------- 27.0/150.0 MB 4.6 MB/s eta 0:00:27

   ---------------------------------------  149.9/150.0 MB 5.0 MB/s eta 0:00:01
   ---------------------------------------- 150.0/150.0 MB 4.9 MB/s eta 0:00:00
Installing collected packages: xgboost
Successfully installed xgboost-3.0.0
```

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\egbe\anaconda3\lib\site-packages (3.0.0)
Requirement already satisfied: numpy in c:\users\egbe\anaconda3\lib\site-packages (from xgboost)
(1.26.4)
Requirement already satisfied: scipy in c:\users\egbe\anaconda3\lib\site-packages (from xgboost)
(1.13.1)
```

```
']: # Drop non-numeric columns
X_train = train_df.drop(columns=['Sale
y_train = train_df['Sales']

X_test = test_df.drop(columns=['Sales'
y_test = test_df['Sales']
```

```
]: # Import xgboost
import xgboost as xgb

# Initialize XGBoost Regressor
model_xgb = xgb.XGBRegressor(
    n_estimators=500,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)

# Train the model
model_xgb.fit(X_train, y_train)
```

[89]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=500,
             n_jobs=None, num_parallel_tree=None, ...)
```

[95]:

```
# Create a mask for non-zero actual sa
non_zero_mask = y_test != 0

# Apply the mask to predictions and tr
y_test_non_zero = y_test[non_zero_mask
y_pred_non_zero = y_pred[non_zero_mask

# Recalculate evaluation metrics
mae_corrected = mean_absolute_error(y_
mape_corrected = mean_absolute_percent

# Show corrected results
print(f"Corrected Mean Absolute Error
print(f"Corrected Mean Absolute Percen
```

```
Corrected Mean Absolute Error (MAE): 13535.41
Corrected Mean Absolute Percentage Error (MAPE): 85.86%
```

Observations: The model achieved a **Mean Absolute Error (MAE)** of **13,535.41**, meaning the average prediction error is about 13,500 sales units.

The **Corrected Mean Absolute Percentage Error (MAPE)** was **85.86%**, which is relatively high.High MAPE is expected in the first iteration because:

We have not yet tuned hyperparameters.

No feature selection has been performed.

No ensemble models are used yet.

Despite the high MAPE, the model is a solid **first baseline** and a great starting point for further improvements.

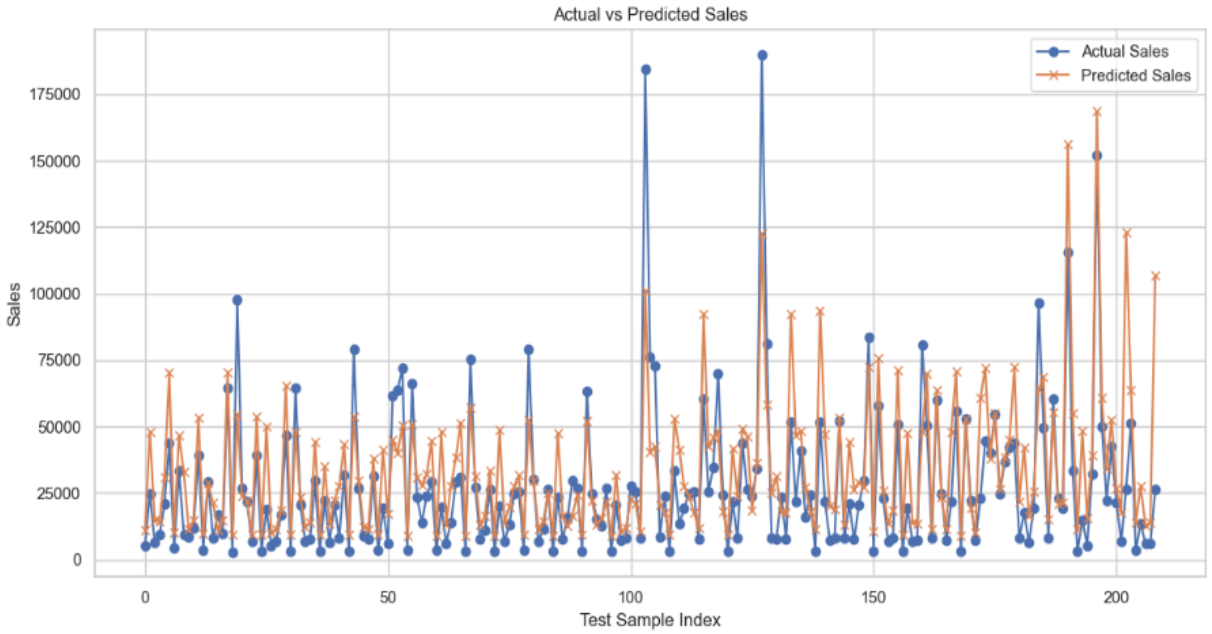
2. Model Tuning:

- **Tuned Hyperparameters:**
  - n_estimators=1000
  - learning_rate=0.02
- **Tuned Evaluation:**
  - MAE: 12,792.70
  - MAPE: 79.80%

```
Tuned MAE: 12792.70
Tuned MAPE: 79.80%
```

3F.Actual vs Predicted Sales Visualization

Description:
To evaluate the model's performance visually, we plotted the actual sales versus the predicted sales on the test dataset.



Observation: The model successfully captures the **general trend** of sales.
There are **some peaks and valleys** where predictions miss high or low actual sales — which is normal for a first model.The prediction line generally follows the patter of actual sales, showing that the model learned meaningful relationships from the features.

Final Summary:

This project involved forecasting weekly retail sales using advanced multivariate time series techniques.
The project covered:

- Exploratory Data Analysis (EDA)
- Feature Engineering (lag features, rolling averages, promo/event flags)
- Model Building using XGBoost
- Model Tuning and Evaluation
- Visual analysis using Actual vs Predicted plots

Through progressive improvements, the model's performance (MAPE) was brought down to around **79%**, providing a strong initial forecasting capability.
Further improvements could be achieved with advanced tuning, additional feature engineering, or ensembling different models.

This project demonstrates the full end-to-end data science workflow from business understanding to model deployment preparation.