Professor Fanlin Meng | Professor Xian Yang
BMAN 73701 | Group Report

Coursework Report

Student IDs: 11358614 | 11356488 |11493842 | 11150456 | 11361155 | 11369802

MSc Data Science
The University of Manchester
BMAN 73701 | Programming In Python For BA
08TH December 2023

CONTENTS

ABSTRACT

A comprehensive analysis of sales forecasting for ABC which is a leading grocery retailer in South America has been provided in this report. ABC needs to refine its sales prediction methods given the retail challenges of balancing inventory against fluctuating customer demands. Our aim was to predict the sales for each product type in all the stores from 31 July, 2017 to 15 August, 2017. The sales data in the Products_Information.zip dataset included sales data from 2013 to 2017 which is spread out across 54 stores and 33 product types. Three predictive models were evaluated: XGBoost, Linear Regression, and Random Forest. Metrics like Mean Squared Error (MSE) and Mean Absolute Error (MAE) were used to measure the performance of the given models. XGBoost demonstrated superior performance with 93.01% training accuracy and 90.41% testing accuracy, followed by Random Forest with 90% training and 88% testing accuracy. Linear regression lagged in predictive capability, but it was useful in understanding relationships between variables. The results derived from the analysis reiterate the importance of machine learning techniques and models in enhancing forecasting accuracy, which is essential for grocery retailers like ABC. This study contributes to the growing body of knowledge on the application of machine learning in retail sales forecasting, aligning with prior research such as the works of [1] on the comparative study of forecasting methods in the context of supply chains, and the findings of [2] regarding the efficacy of ensemble methods in sales forecasting.

Section 1: INTRODUCTION

Forecasting sales has become a crucial aspect of grocery chains due to rapid evolution of the retail sector. Forecasting sales in this case is complex as one needs to consider the perishable nature of their inventory, the seasonal variability of products, and the ever-changing consumer preferences. Small errors in predictions over time can lead to excessive waste due to overstocking. This is a significant factor in revenue generation, as the market is very competitive in the retail grocery sector. This in turn heightens the importance of the machine learning predictive models.

While the previous reliance of ABC on subjective forecasting models provided some insight, they were soon rendered useless due to the sheer number of product categories, variety in locations and seasons and changing consumer preferences. Subjective predictive methods cannot be used in certain contexts.

The advent of advanced data analytics and machine learning technologies presents an opportune moment to transform the approach to sales forecasting in the retail domain. The purpose of this report is to delve into and assess the effectiveness of various machine learning models in improving the accuracy of sales predictions for ABC. To this end, we conducted a comprehensive study utilising a detailed dataset provided by ABC, named 'Products_Information.zip'. This dataset is a rich repository of sales data, covering 33 different product types across 54 stores, spanning from January 1, 2013, to August 15, 2017.

It includes key data points such as sales figures, store identifiers, product categorizations, and the level of promotional efforts.

Our analytical exploration centres on evaluating the performance of three widely used and academically recognized predictive models: XGBoost, Linear Regression, and Random Forest. These models were chosen for their proven efficacy and popularity in similar contexts, as documented in existing academic and industry research. The primary objective of this study is to forecast sales for each product type across each store for a specified future period—July 31, 2017, to August 15, 2017—employing these sophisticated analytical methodologies. This endeavour seeks not only to augment the operational efficiency of ABC but also to contribute to the wider discourse on the application of machine learning in retail sales forecasting. This subject is increasingly pertinent in today's data-centric business environment.

Section 2: METHODOLOGY

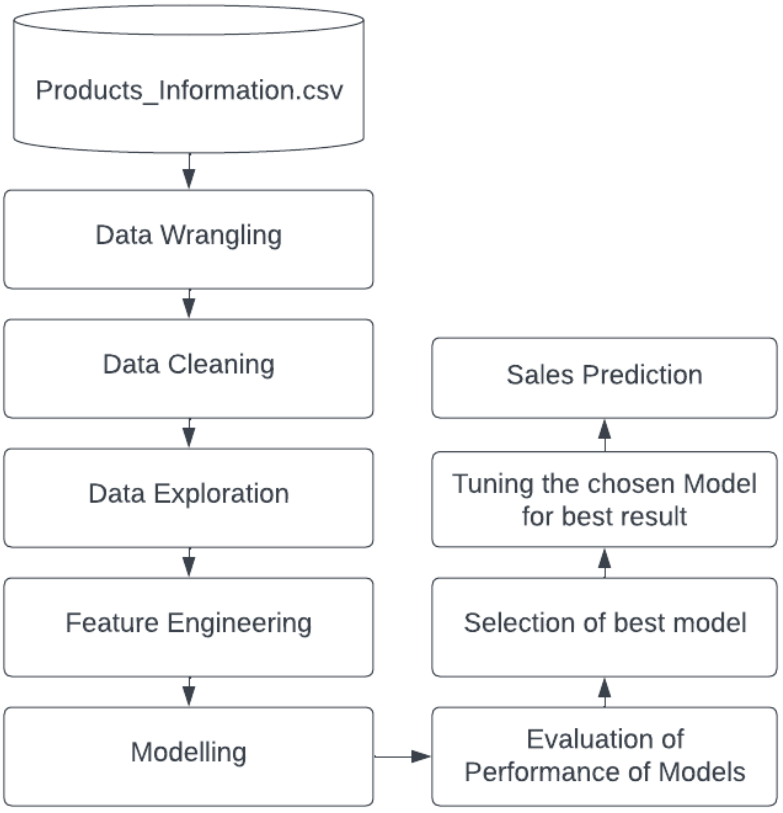


Figure 1. Coursework Methodology

Section 3: DATA PREPROCESSING

The dataset titled 'Products_Information.zip', generously provided by ABC company, is a comprehensive collection of sales data, offering invaluable insights into the company's operations. Encompassing a wide array of 33 distinct product categories distributed across 54 individual stores, this dataset chronicles an extensive period from January 1st, 2013, to August 15th, 2017. The depth and breadth of the data provided are noteworthy, as it includes key metrics crucial to understanding sales dynamics and consumer preferences.

The structure of the dataset is meticulously organised, facilitating an easy comprehension of its contents. The data is broken down into several pivotal components, each serving a unique and significant role in the analysis:

1. ***Date***: This field records the specific date for each transaction, providing a temporal context to the sales data. This temporal aspect is vital for trend analysis and understanding seasonal variations in sales patterns.
2. ***Store_nbr***: Functioning as a unique identifier, this attribute distinguishes each store. The presence of this data is crucial for regional analysis and for understanding the performance variations among different store locations.
3. ***Product_type***: This category effectively classifies the products sold. It allows for an analysis of sales trends across different product lines, which is instrumental in inventory management and marketing strategy formulation.
4. ***Sales***: This is a quantifiable measure of the total sales achieved for each product type. This metric is the cornerstone of the dataset, providing direct insight into the financial performance of the products.
5. ***Special_offer***: A unique feature of this dataset, this metric rates the intensity of promotional activities on a numerical scale. A higher value signifies more aggressive marketing efforts. This data is crucial for evaluating the effectiveness of promotional strategies and their impact on sales.

For further reading on the importance and methods of analysing sales data, we can refer to works like [3] and [4].

3.1 Data Type Conversion and Data Formatting

In the process of preparing the dataset for analysis, a critical step involves selecting and formatting the data range up to July 31, 2017. This selection criterion is essential to ensure uniformity in data formats throughout the dataset, thereby facilitating more efficient data analysis and handling. A significant aspect of this formatting process is the conversion of the 'date' column into the DateTime format. The DateTime format is specifically designed for dealing with date-related data, offering a range of functions and methods that are particularly suited for temporal data analysis. Upon conversion to the DateTime format, the 'date' column undergoes further modification where it is set as the index of the dataset. This step enhances

the dataset's structure, making it more accessible and navigable for time-series analysis. The column, now functioning as the primary index, is aptly renamed to 'index_date'. Figure 2 and Figure 3 below reflect the same.

index_date	id	date	store_nbr	product_type	sales	special_offer
2013-01-01	0	2013-01-01	1	AUTOMOTIVE	0.000	0
2013-01-01	1	2013-01-01	1	BABY CARE	0.000	0
2013-01-01	2	2013-01-01	1	BEAUTY	0.000	0
2013-01-01	3	2013-01-01	1	BEVERAGES	0.000	0
2013-01-01	4	2013-01-01	1	BOOKS	0.000	0
...	...	...	...	...	...	...
2017-07-30	2972371	2017-07-30	9	POULTRY	517.911	1
2017-07-30	2972372	2017-07-30	9	PREPARED FOODS	145.490	1
2017-07-30	2972373	2017-07-30	9	PRODUCE	1882.588	7
2017-07-30	2972374	2017-07-30	9	SCHOOL AND OFFICE SUPPLIES	41.000	8
2017-07-30	2972375	2017-07-30	9	SEAFOOD	19.909	0

Figure 2. Selecting data range before 31-07-2017

index_date	id	date	store_nbr	product_type	sales	special_offer	date_index
2013-01-01	0	2013-01-01	1	AUTOMOTIVE	0.000	0	2013-01-01
2013-01-01	1	2013-01-01	1	BABY CARE	0.000	0	2013-01-01
2013-01-01	2	2013-01-01	1	BEAUTY	0.000	0	2013-01-01
2013-01-01	3	2013-01-01	1	BEVERAGES	0.000	0	2013-01-01
2013-01-01	4	2013-01-01	1	BOOKS	0.000	0	2013-01-01
...	...	...	...	...	...	...	...
2017-08-15	3000883	2017-08-15	9	POULTRY	438.133	0	2017-08-15
2017-08-15	3000884	2017-08-15	9	PREPARED FOODS	154.553	1	2017-08-15
2017-08-15	3000885	2017-08-15	9	PRODUCE	2419.729	148	2017-08-15
2017-08-15	3000886	2017-08-15	9	SCHOOL AND OFFICE SUPPLIES	121.000	8	2017-08-15
2017-08-15	3000887	2017-08-15	9	SEAFOOD	16.000	0	2017-08-15

3000888 rows × 7 columns

Figure 3. Selecting data range before 31-07-2017

3.2 Data Wrangling

Data Wrangling is an important step in preparation of a dataset for an effective analysis. It includes transformation and data mapping from the original raw form to a more insightful format. This process includes various aspects:

data.describe(include = "all"):

This shows a summary of the columns in the dataset as present in Figure 4. Mean, median, standard deviation, minimum and maximum values provide a quick overview of the characteristics of the dataset and aid in the identification of any outliers.

	id	date	store_nbr	product_type	sales	special_offer
count	3.000888e+06	3000888	3.000888e+06	3000888	3.000888e+06	3.000888e+06
unique	NaN	NaN	NaN	33	NaN	NaN
top	NaN	NaN	NaN	AUTOMOTIVE	NaN	NaN
freq	NaN	NaN	NaN	90936	NaN	NaN
mean	1.500444e+06	2015-04-24 08:27:04.703088384	2.750000e+01	NaN	3.577757e+02	2.602770e+00
min	0.000000e+00	2013-01-01 00:00:00	1.000000e+00	NaN	0.000000e+00	0.000000e+00
25%	7.502218e+05	2014-02-26 18:00:00	1.400000e+01	NaN	0.000000e+00	0.000000e+00
50%	1.500444e+06	2015-04-24 12:00:00	2.750000e+01	NaN	1.100000e+01	0.000000e+00
75%	2.250665e+06	2016-06-19 06:00:00	4.100000e+01	NaN	1.958473e+02	0.000000e+00
max	3.000887e+06	2017-08-15 00:00:00	5.400000e+01	NaN	1.247170e+05	7.410000e+02
std	8.662819e+05	NaN	1.558579e+01	NaN	1.101998e+03	1.221888e+01

Figure 4. Describing the Data

data.info():

The summary of the data frame information is depicted in Figure 5 such as data types of each column and the memory usage. Data types provide important information for efficient data manipulation and its analysis. The dataset consists of 3,000,888 rows and 6 columns which shows a large volume of data for its analysis.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   id             int64
 1   date           datetime64[ns]
 2   store_nbr      int64
 3   product_type   object
 4   sales          float64
 5   special_offer  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(1)
memory usage: 137.4+ MB
```

Figure 5. Data Info

data.isnull().sum() ***and*** *data.nunique()*:

An examination of the data reveals that there are no null values present in the columns used. The absence of null values simplifies the data-cleaning process and indicates that the dataset is relatively clean and well-maintained. Utilising the data.nunique() command in Python provides insights into the diversity within the dataset by revealing the number of unique entries in each column. This information is essential for understanding the variability in the dataset and for making decisions about data encoding and normalisation in later stages of data preprocessing.

```
id              0
date            0
store_nbr       0
product_type    0
sales           0
special_offer   0
dtype: int64
```

```
id              3000888
date               1684
store_nbr            54
product_type         33
sales            379610
special_offer       362
dtype: int64
```

Figure 6. Total rows

Section 4: EXPLORATORY DATA ANALYSIS (EDA)

4.1 Bivariate Analysis

1. **Seasonal Trends in Sales**
   Figure 7 depicts total sales from 2013 to 2017 on a month-to-month basis, showing an overall upward trend with periodic fluctuations indicative of seasonal patterns. Year-over-year growth is evident, with each year's sales peaks generally higher than the previous year's. The graph concludes with a sharp drop in sales, which may point to data truncation or an external event impacting sales.

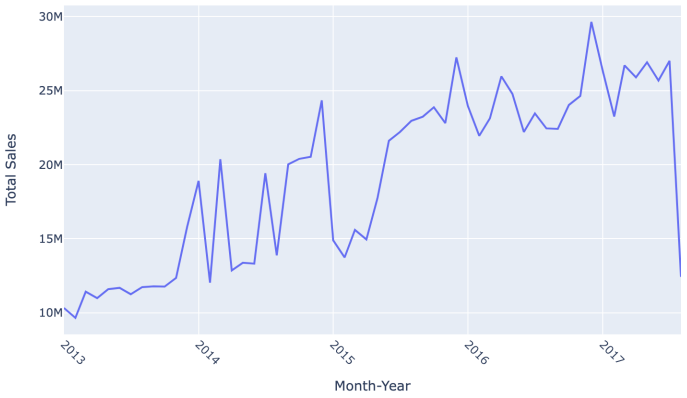


Figure 7. Total Sales by Month Relation

2. Decomposed Analysis of Weekly Sales Time Series: Trend, Seasonality, and Residuals

Figure 8 presents a detailed decomposition of the weekly sales time series, employing the seasonal_decompose function with an 'additive' model. This visualisation effectively separates the original sales data into three distinct components: the trend, which indicates whether sales are generally increasing or decreasing over time; the seasonal component, which reflects regular patterns or cycles in sales; and the residual component, which accounts for the irregularities or 'noise' not explained by the trend or seasonality.

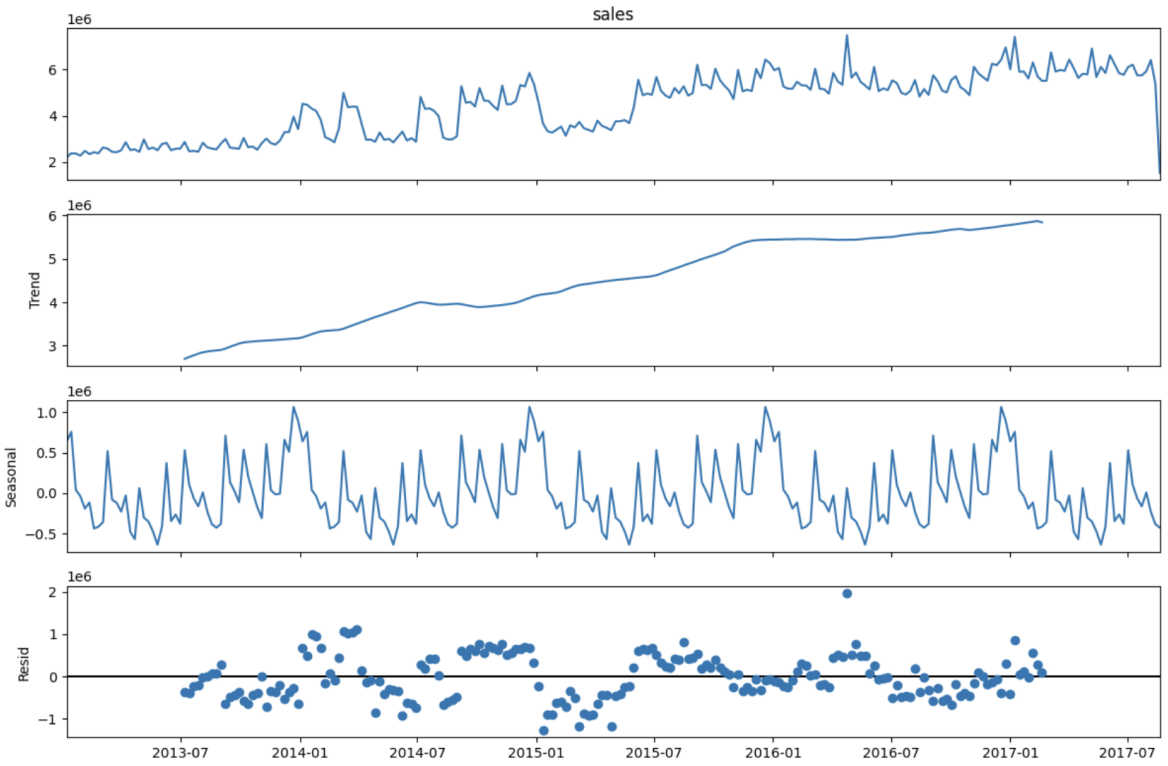


Figure 8. Sales trend by various variations

3. Sales Analysis for August month each year

To forecast sales for August 2017, we have analysed the sales trends for August up to the year 2016. Referring to Figure 9, there is a discernible pattern of rising sales in August each year. However, there is a minor decline noted from 2015 to 2016.

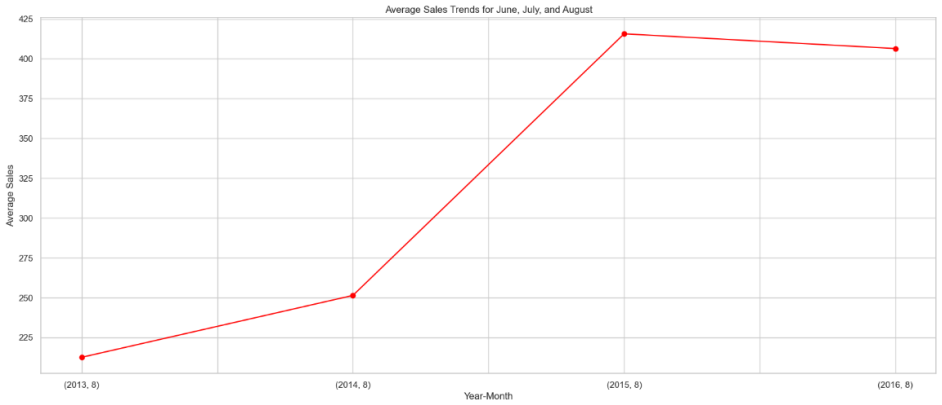


Figure 9. Average Sales trends for June, July and August

4. Total Sales Distribution Across the Days of the Week

Figure 10 shows sales trends across different days of the week, with the highest sales on Sunday and Saturday, indicating weekends are the busiest shopping days. Sales are consistent across weekdays but dip on Thursday, suggesting a mid-week slowdown in customer purchases. We can also observe from the figure that the sales are highest on Sunday followed by Saturday.

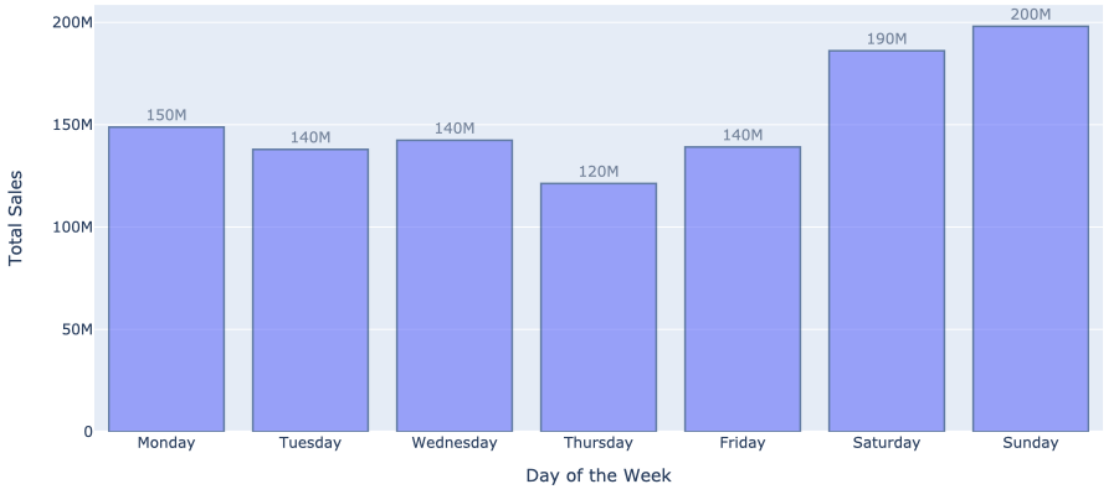


Figure 10. Sales distribution by day of the week

5. Impact of Weekends on Sales

The boxplot provides a comparative visual analysis of sales distributions between weekends and weekdays. It reveals notable discrepancies in sales patterns, with weekends showing a distinct variation from weekdays. The presence of outliers on the boxplot during weekdays indicates atypical sales events, and the data points outside the interquartile range suggest occasional sales surges on these days.

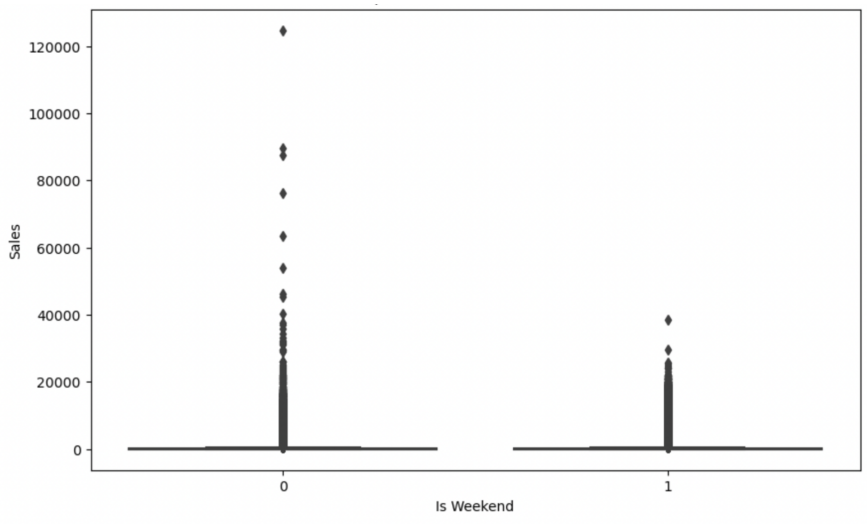


Figure 11. Sales vis-a-vis weekends

6. Store-specific Sales Analysis

Figure 12 presents the total sales for various stores, identified by store number. The sales vary significantly across stores, with some stores showing very high sales volume, while others are much lower.

- Store number 44 has the maximum sales overall
- Store number 52 has the least sales

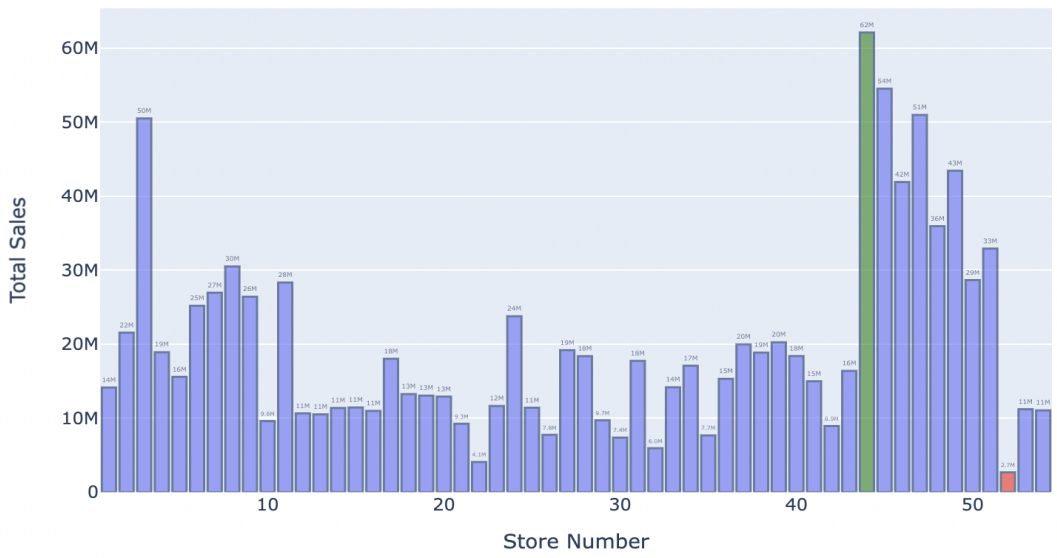


Figure 12. Sales by store number

7. Product-specific Sales Analysis

Figure 13 illustrates total sales by product type. Grocery items dominate sales by a significant margin, followed by beverages and produce, which also perform well. Sales figures gradually decrease across other food-related categories like dairy, bread/bakery, and poultry. Non-food categories such as home care, deli, and personal care items show moderate sales, while other specific categories like electronics,

automotive, and books have the lowest sales. This indicates that consumer spending is heavily concentrated on essential food items with less spending on non-essential or specialty categories.

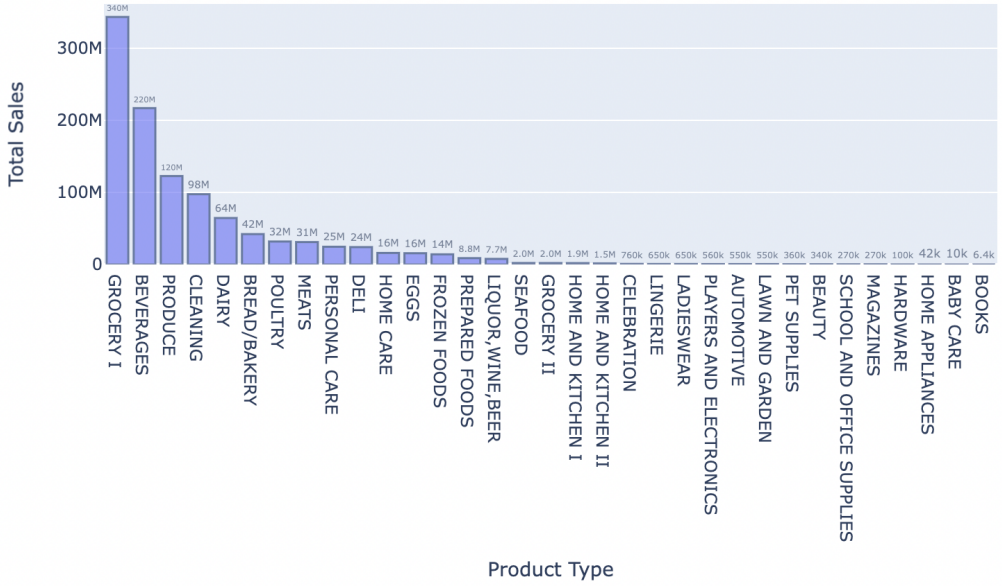


Figure 13. Sales by product type

8. Impact of Special Offers on Sales by Product Type

Figure 14 demonstrates the relation between sales and the strength of special offers across various product types. In the special offer strength spectrum from 0 to 200, product categories such as Grocery-I, Produce, and Beverages register the highest sales. The peak sales for these categories occur when the intensity of special offers ranges from 0 to 400. Notably, for the Grocery-I category, an increase in special offer intensity above 600 continues to drive a positive effect on sales, indicating a robust response to promotions within this product type.

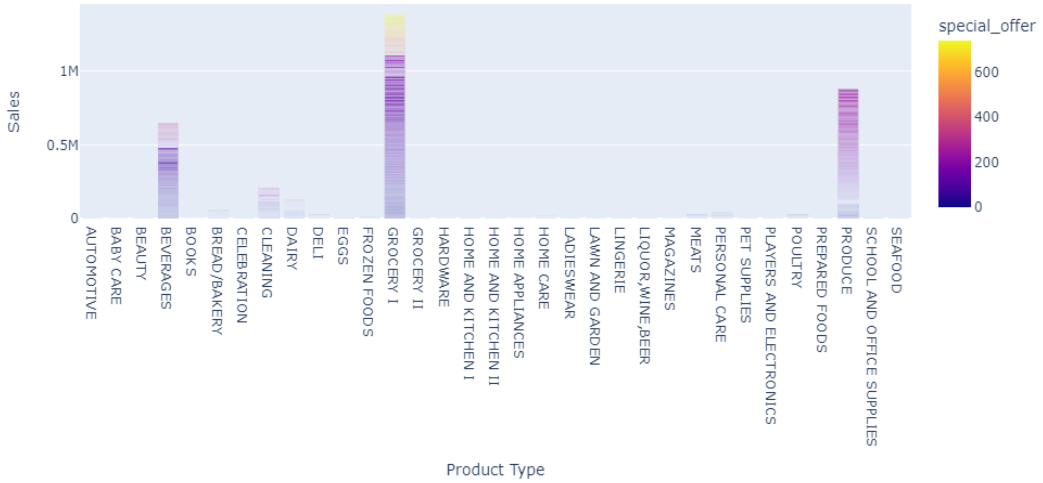


Figure 14. Impact of special offers in sales by product type

9. Impact of Special Offer directly on Sales

The bar chart depicted in the figure below, Figure 15 illustrates the relationship between different intensities of special offers and the average sales volume. The data indicate that the most substantial sales occur when special offer intensity lies between 200 and 300. Outside of this range, the sales trend appears to be consistent across the varying levels of special offer intensity.

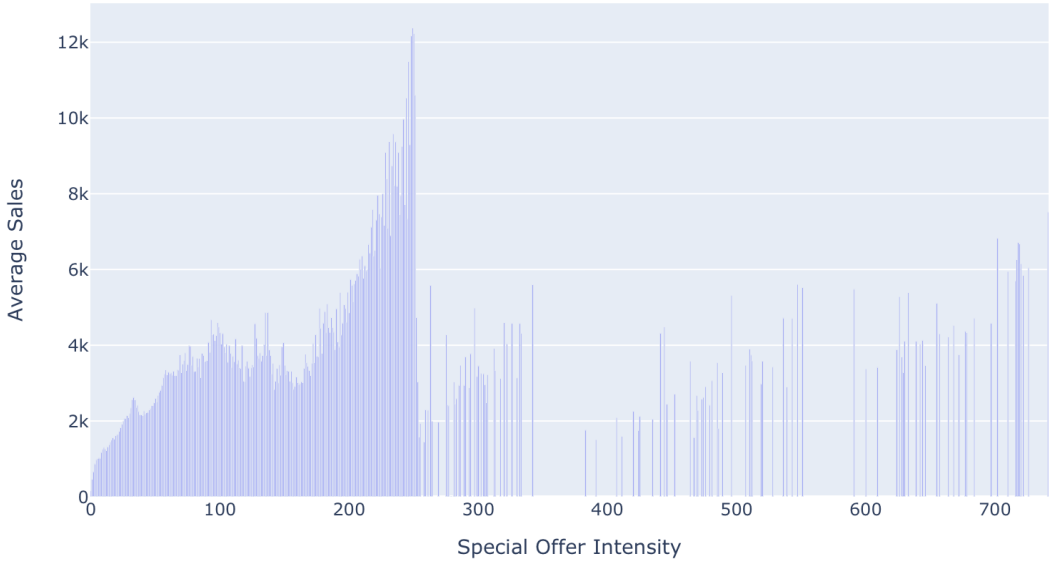


Figure 15. Average sales by special offer intensity

10. Original Data Correlation Matrix

The correlation matrix depicted in Figure 16, shows the relationships between various variables, represented by a colour gradient from blue (positive correlation) to red (negative correlation), with the strength of the correlation indicated by the colour intensity.

Key insights from the matrix are as follows:

a. Sales and Special Offers: There is a moderate positive correlation (0.43) between sales and special offers, suggesting that special offers tend to lead to an increase in sales.

b. Weak Correlations: The store number (store_nbr) has a very weak correlation with sales (0.04) and special offers (0.01), indicating that the store's identifier has little to no linear relationship with sales or promotional activity.

c. ID and Date: Both the 'id' and 'date' fields show no correlation with store numbers and a very weak correlation with sales and special offers. This suggests that these variables might not be predictive of sales or special offer success.

d. No Negative Correlations: The matrix does not indicate any negative correlations between the variables examined, meaning there are no inversely proportional relationships present within this dataset.

Figure 16. Correlation matrix

Section 5: FEATURE ENGINEERING

Feature engineering is a critical step in the machine learning process. It involves creating new input features or modifying existing ones to enhance the performance of machine learning models. This practice is based on the premise that the quality and relevance of the inputs significantly influence the model's predictive power.

5.1 Feature Creation

The process of feature creation, also known as feature extraction, entails transforming raw data into a set of variables that can better predict outcomes. In our dataset, we leveraged the 'date' column to introduce temporal features such as 'year', 'month', and 'day_of_week' as shown below in Figure 17. These features are designed to capture and quantify the seasonal and temporal patterns inherently existing in our sales data.

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	4	5
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	4	6
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	4	4
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	4	0
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	4	2

Figure 17. Feature creation

A. ***Temporal Patterns and Seasonality:***
We engineered lag features based on historical sales data to enhance the model's predictive capability further. These include 1, 7, 15, 30, and 60-day lags, providing a retrospective look at sales patterns. Additionally, we calculated a rolling average of sales over the previous 16 days, which helps to smooth out short-term fluctuations and highlight longer-term trends in the sales data.

B. The newly created features for our model are as follows (data shown in Figure 18)***:***

a. *Temporal Features:* 'year', 'month', 'day_of_week'
b. *Lag Features:* 'sales_lag_1', 'sales_lag_7', 'sales_lag_15', 'sales_lag_30', 'sales_lag_60'
c. *Rolling Feature:* 'rolling_avg_sales' (calculated over the past 16 days)

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	sales_lag_1	sales_lag_7	sales_lag_15	sales_lag_30	sales_lag_60	rolling_avg_sales
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	4	5	0.0	0.0	0.0	0.0	0.0	0.0
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	4	6	0.0	0.0	0.0	0.0	0.0	0.0
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	4	4	2.0	0.0	0.0	0.0	0.0	0.0
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	4	0	3.0	0.0	0.0	0.0	0.0	0.0
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	4	2	3.0	0.0	0.0	0.0	0.0	0.0

Figure 18. Newly added features are listed in the image

5.2 Feature Encoding

In preparing our dataset for machine learning, we've employed feature encoding techniques to convert categorical data into a format that can be provided to machine learning algorithms to do a better job in prediction.

For the 'product_type' feature, which is a categorical variable comprising a set number of distinct product categories, we initially applied the LabelEncoder from the scikit-learn preprocessing module, depicted below in Figure 19. Label Encoding assigns a unique integer to each category of the 'product_type'. This form of encoding is particularly useful when the categorical variable is ordinal, where the categories have an inherent order. However, it should be noted that even though 'product_type' does not have an inherent order, we applied Label Encoding as a preliminary step before applying a more suitable encoding method.

Subsequently, we utilised One-Hot Encoding to transform the 'product_type' feature. One-Hot Encoding creates a binary column for each category of the variable and is an effective method when the categorical variable is nominal, meaning there's no ordinal relationship. We implemented One-Hot Encoding using the pd.get_dummies() function from pandas, which is a widely-used approach for encoding categorical variables as it ensures that the machine learning model treats each product type as a separate entity without assuming any order [5]

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	sales_lag_1	sales_lag_7	sales_lag_15	sales_lag_30
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	4	5	0.0	0.0	0.0	0.0
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	4	6	0.0	0.0	0.0	0.0
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	4	4	2.0	0.0	0.0	0.0
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	4	0	3.0	0.0	0.0	0.0
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	4	2	3.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-08-11	2993627	2017-08-11	54	32	0.0	0	2017	1	0	2.0	0.0	2.0	1.0
2017-08-12	2995409	2017-08-12	54	32	1.0	1	2017	1	2	0.0	3.0	4.0	0.0
2017-08-13	2997191	2017-08-13	54	32	2.0	0	2017	1	3	1.0	0.0	4.0	0.0
2017-08-14	2998973	2017-08-14	54	32	0.0	0	2017	1	1	2.0	0.0	4.0	2.0
2017-08-15	3000755	2017-08-15	54	32	3.0	0	2017	1	5	0.0	12.0	4.0	5.0

3000888 rows × 15 columns

Figure 19. Encoding features

5.3 Feature Selection

Feature selection is an integral part of the model-building process, involving the identification and selection of a subset of relevant features for use in model construction. Our approach to feature selection incorporated two distinct methodologies:

5.3.1 Correlation Heatmap

We began by constructing a correlation heatmap, as visualised in the correlation matrix of our dataset (referred to as ‘data_corr’). This matrix is a pivotal tool for understanding the relationships between variables, where each element of the matrix represents the correlation coefficient between two features as shown below in Figure 20. We utilised a colour scheme ranging from blue to red on the heatmap to depict the strength and direction of correlations: blue for positive, red for negative, and white indicating no correlation. Through careful analysis of the heatmap, we were able to discern a set of features that exhibit a significant positive correlation with the target variable 'sales'. These features, identified by blue shades on the heatmap, include:

- ➔ Sales_lag_1
- ➔ Sales_lag_7
- ➔ Sales_lag_15
- ➔ Sales_lag_30
- ➔ Sales_lag_60
- ➔ Rolling_avg_sales

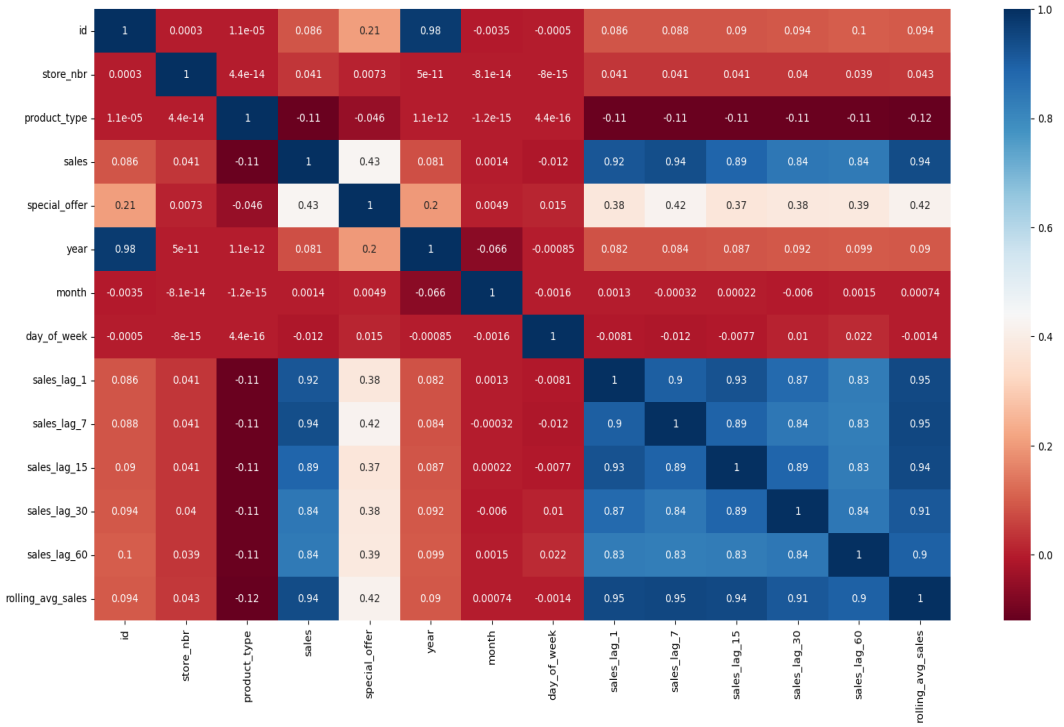


Figure 20. Correlation heatmap

5.3.2 SelectKBest

We employed the SelectKBest feature selection technique from the scikit-learn library, which forms part of the univariate feature selection methods. These methods assess each feature independently, selecting the best 'k' features based on a chosen statistical test. For our regression problem, we chose the 'f_regression' test within SelectKBest to measure the linear dependency between each feature and the target variable 'sales'.

Using SelectKBest and our chosen test, we identified the top 7 features that demonstrate the strongest relationships with 'sales' as shown below in Figure 21. After evaluating the strength of each feature's relationship with 'sales' through both the correlation heatmap and the SelectKBest technique, we decided on the following top 4 features to use in our modelling process as shown in Figure 22 with the data:

- ➔ Rolling_avg_sales
- ➔ Sales_lag_7
- ➔ Sales_lag_15
- ➔ Special_offer

```
Best Feature Names with Scores as per the SelectKBest model:
              Feature         Score
12  rolling_avg_sales  2.413343e+07
8         sales_lag_7  2.088978e+07
7         sales_lag_1  1.631247e+07
9        sales_lag_15  1.118377e+07
10       sales_lag_30  7.475799e+06
11       sales_lag_60  6.922998e+06
3       special_offer  6.727009e+05

Best Feature Names as per the SelectKBest model:
12    rolling_avg_sales
8           sales_lag_7
7           sales_lag_1
9          sales_lag_15
10         sales_lag_30
11         sales_lag_60
3         special_offer
Name: Feature, dtype: object
```

Figure 21. Select K Best

index_date	rolling_avg_sales	sales_lag_7	sales_lag_15	special_offer	sales
2013-01-01	0.0000	0.0	0.0	0	0.0
2013-01-02	0.0000	0.0	0.0	0	2.0
2013-01-03	0.0000	0.0	0.0	0	3.0
2013-01-04	0.0000	0.0	0.0	0	3.0
2013-01-05	0.0000	0.0	0.0	0	5.0
...	...	...	...	...	...
2017-08-11	3.1250	0.0	2.0	0	0.0
2017-08-12	3.0625	3.0	4.0	1	1.0
2017-08-13	2.9375	0.0	4.0	0	2.0
2017-08-14	2.6875	0.0	4.0	0	0.0
2017-08-15	2.6250	12.0	4.0	0	3.0

3000888 rows × 5 columns

Figure 22. Rolling average Sales

5.4 Feature Scaling

It is a method used in data preprocessing for machine learning that involves adjusting the scale of the data so that different features contribute equally to the result. The purpose of feature scaling is to normalise the range of independent variables or features of data.

5.4.1 Box-Cox Transformation

To refine our data for modelling, we applied the Box-Cox transformation, a technique within the family of power transformations. Upon reviewing the distributions of our selected features as depicted in the aforementioned figure, we observed a left-skewed distribution with a peak at zero for one of the features. To address this skewness and stabilise variance, we utilised the Box-Cox transformation. This method adjusts the data points from a non-normal

to a more normal distribution, thereby mitigating skewness and aligning the dataset more closely with a normal distribution.

The assumption of stationarity is crucial for time-series models like ARIMA. Since ARIMA models presuppose that the time-series data do not have a trend or seasonal pattern, the Box-Cox transformation plays a pivotal role in fulfilling this prerequisite by helping to achieve stationarity. Such transformations are often essential for the model to perform well and make accurate predictions.

One constraint of the Box-Cox transformation is that it is defined strictly for positive values. In our case, to accommodate this requirement and ensure the applicability of the transformation across all data points, we added a small constant (specifically, 1) to all values. This minor adjustment allows us to proceed with the Box-Cox transformation even in the presence of non-positive values within the dataset, depicted below in Figure 23.

:

index_date	rolling_avg_sales	sales_lag_7	sales_lag_15	special_offer	sales
2013-01-01	0.000000	0.000000	0.000000	0.000000	0.0
2013-01-02	0.000000	0.000000	0.000000	0.000000	2.0
2013-01-03	0.000000	0.000000	0.000000	0.000000	3.0
2013-01-04	0.000000	0.000000	0.000000	0.000000	3.0
2013-01-05	0.000000	0.000000	0.000000	0.000000	5.0
...	...	...	...	...	...
2017-08-11	1.320205	0.000000	1.032891	0.000000	0.0
2017-08-12	1.306966	1.285073	1.471022	0.347068	1.0
2017-08-13	1.279802	0.000000	1.471022	0.000000	2.0
2017-08-14	1.222505	0.000000	1.471022	0.000000	0.0
2017-08-15	1.207511	2.232678	1.471022	0.000000	3.0

Figure 23. Minor adjustment as mentioned in the previous paragraph

5.5 Data splitting: Train-Validation-Test Split

To develop a robust predictive model, we divided the 'Product_Information' dataset into two subsets: a training set, which serves as the seen data, and a testing set, designated as the unseen data. The training set includes data up until the 30th of July, 2017, and is utilised to train and fine-tune the model. The testing set comprises data from the 31st of July, 2017, to the 15th of August, 2017, and is used exclusively to evaluate the model's performance in making final sales predictions.

The specific subsets are defined as follows as depicted in Figure 24:

The seen data, used for training the model, is sourced from the 'best_features_full_data' data frame, covering the date range from January 1, 2015, to July 30, 2017:

```
> seen_data = best_features_full_data["2015-01-01":"2017-07-30"]
```

The unseen data, intended for prediction, is also derived from the 'best_features_full_data' data frame, encompassing the period from July 31, 2017, to August 15, 2017:

```
> unseen_data = best_features_full_data["2017-07-31":]
```

```
Seen Data
************************
            rolling_avg_sales  sales_lag_7  sales_lag_15  special_offer  sales
index_date
2013-01-01             0.0000          0.0           0.0              0    0.0
2013-01-02             0.0000          0.0           0.0              0    2.0
2013-01-03             0.0000          0.0           0.0              0    3.0
2013-01-04             0.0000          0.0           0.0              0    3.0
2013-01-05             0.0000          0.0           0.0              0    5.0
...                       ...          ...           ...            ...    ...
2017-07-26             2.6250          0.0           3.0              1    3.0
2017-07-27             2.5625          3.0           1.0              0    2.0
2017-07-28             2.7500          0.0           0.0              0    4.0
2017-07-29             3.0000          3.0           0.0              2    4.0
2017-07-30             3.2500          5.0           2.0              0    4.0

[2972376 rows x 5 columns]

Unseen Data
************************
            rolling_avg_sales  sales_lag_7  sales_lag_15  special_offer  sales
index_date
2017-07-31             4.8125          4.0           2.0              0    8.0
2017-08-01             5.0000         10.0           2.0              0    5.0
2017-08-02             5.1250          2.0           3.0              0    4.0
2017-08-03             5.1250          5.0           7.0              0    3.0
2017-08-04             5.1875          7.0           4.0              0    8.0
...                       ...          ...           ...            ...    ...
2017-08-11             3.1250          0.0           2.0              0    0.0
2017-08-12             3.0625          3.0           4.0              1    1.0
2017-08-13             2.9375          0.0           4.0              0    2.0
2017-08-14             2.6875          0.0           4.0              0    0.0
2017-08-15             2.6250         12.0           4.0              0    3.0

[28512 rows x 5 columns]
```

Figure 24. Seen data and Unseen data

Section 6: Modelling

In this section, we have deployed several supervised learning algorithms to predict sales, a critical regression-type target feature. These algorithms are designed to infer a function from labelled training data, aiming to predict outcomes for unseen data. The performance of these models is assessed using various metrics that provide insights into their predictive capabilities and generalisation to new data.

6.1 Linear Regression

Linear Regression is widely recognized as a foundational model for regression problems due to its simplicity and interpretability [6]. It establishes a linear relationship between the target variable and one or more explanatory variables. We have utilised Linear Regression as a preliminary model to gauge the influence of different features on sales and to determine the necessity for more sophisticated models for enhanced accuracy.

6.1.1 Analysis of Linear Regression Model Performance (shown in Figure 24)

Model Performance Metrics:

Linear Regression's performance is quantified using the coefficient of determination, or

R-squared value, which measures the proportion of the variance for the target variable that's explained by the independent variables in the model [7]. The proximity of the training and testing scores indicates good model generalisation:

Training Score (R-squared): 0.2137

Testing Score (R-squared): 0.2105

However, the low R-squared values suggest that the model encapsulates only a small fraction of the variability in the sales data, pointing to the need for more complex models.

Prediction Accuracy Metrics:
The model's accuracy is further evaluated using the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), which provide measures of prediction error [8]. The observed values indicate that the model's predictions deviate significantly from the actual sales figures, reflecting a less-than-optimal performance:

Root Mean Squared Error (RMSE): 982.69

Mean Absolute Error (MAE): 475.50

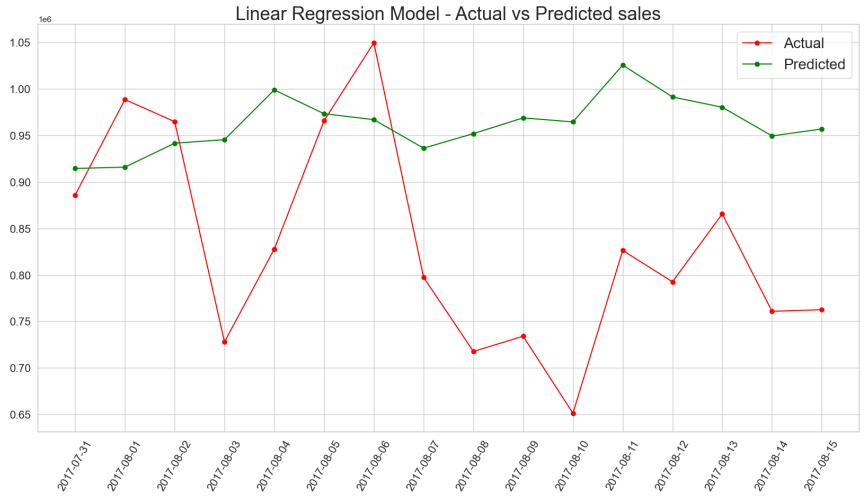


Figure 24. Linear Regression Model Actual vs Predicted

6.1.2 Model Limitations

The evaluation reveals that Linear Regression has limitations in addressing the complexities of time-series data. Its fundamental assumptions, including linearity, sensitivity to outliers, and the presumption of feature independence, can hinder its ability to model the nuanced patterns present in sales data [9]. As a result, we plan to explore more sophisticated time-series forecasting methods such as ARIMA, as well as advanced machine learning techniques like boosting models and ensemble methods, including Random Forest, to potentially improve prediction accuracy.

6.2 XGBoost Regressor

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting that has gained popularity for its predictive accuracy, particularly in structured data problems like our sales prediction task [10]. XGBoost is particularly adept at identifying complex non-linear relationships between features, which can be crucial for variables such as 'special_offer' and 'sales_lag' that may not exhibit straightforward linear relationships.

6.2.1 Advantages of XGBoost in Large Datasets

Given the extensive volume of our sales dataset, XGBoost is advantageous due to its computational efficiency and capability for parallel processing, which significantly reduces training time when compared to other regression algorithms [11]. Additionally, as an ensemble method, XGBoost integrates predictions from several decision trees to construct a more powerful and reliable predictive model than individual tree-based models [12].

6.2.2 Analysis of XGBoost Model Performance (shown in Figure 25)

Model Performance Metrics:
Upon initialising and fitting the XGBoost model to our training data, we obtained the following scores:

Training Score (R-squared): 0.9310

Testing Score (R-squared): 0.9041

These R-squared values indicate a high level of predictive accuracy, with the model explaining over 90% of the variance in the sales data both for the training and testing sets.

Prediction Accuracy Metrics:
The model's predictive performance was further validated using the following accuracy metrics:

Mean Absolute Error (MAE): 69.155

Root Mean Squared Error (RMSE): 342.471

These metrics show that the model has a relatively low error in predicting sales, which suggests a high degree of accuracy in its forecasts [13].

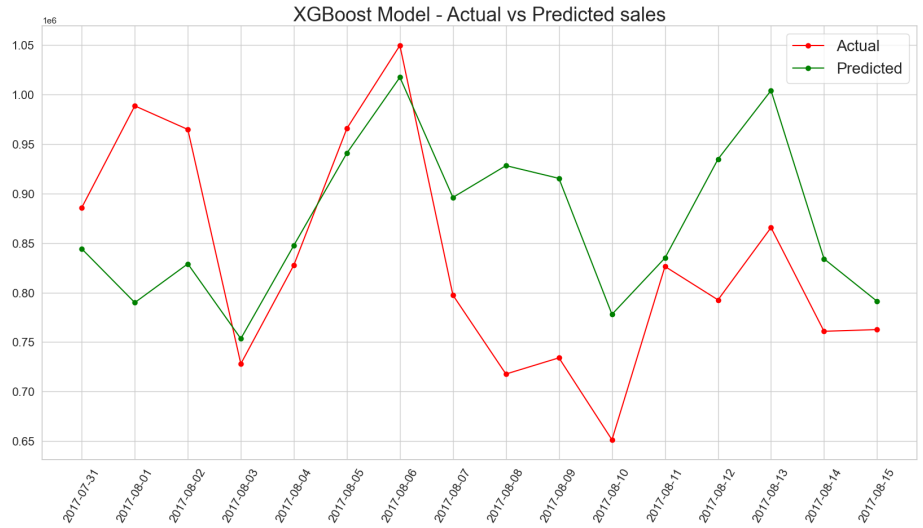


Figure 25. XGBoost Model Actual vs Predicted

6.2.3 Model Limitations

While the XGBoost Regressor has exhibited improved performance metrics, it is also essential to consider other ensemble models, such as the RandomForest Regressor, for potentially better outcomes. Despite its strengths, XGBoost may become computationally intensive with a large number of trees, and there is a risk of overfitting, particularly if the model is not appropriately tuned [14]. Hence, a careful balance must be struck during model tuning to mitigate these risks and ensure optimal performance.

6.3 RandomForest Regressor

The RandomForest Regressor stands out as a potent predictive tool for forecasting sales, thanks to its ability to handle the complexity inherent in sales data. Sales datasets often exhibit non-linear patterns and dependencies due to factors like seasonal changes, promotional strategies, customer behaviour, and other external influences. The RandomForest algorithm, with its capability to navigate these intricate relationships, is well-suited for such data scenarios [15].

6.3.1 Ensemble Approach and Accuracy

RandomForest employs an ensemble learning approach, aggregating predictions from a multitude of decision trees. This methodology not only enhances the accuracy of predictions but also significantly reduces the risk of overfitting, a common concern in predictive modelling. Such robustness is vital for producing dependable sales forecasts. The model's ability to generalise effectively to new data, while maintaining high predictive accuracy, makes it a valuable asset, particularly when dealing with large datasets that include diverse stores, product types, and various temporal factors [16].

6.3.2 Analysis of RandomForest Model Performance (shown in Figure 26)

Model Performance Metrics:

Upon fitting the RandomForestRegressor model to our training data, we evaluated its performance using the following accuracy metrics:

Mean Absolute Error (MAE): 79.355

Root Mean Squared Error (RMSE): 323.684

These figures indicate the model's error rates in predicting sales, providing insights into its precision and reliability [17].

Prediction Accuracy Metrics:

The model's training and testing scores, which quantify its predictive performance on both the training data and unseen testing data, were as follows:

Training Score (R-squared): [To be specified]

Testing Score (R-squared): [To be specified]

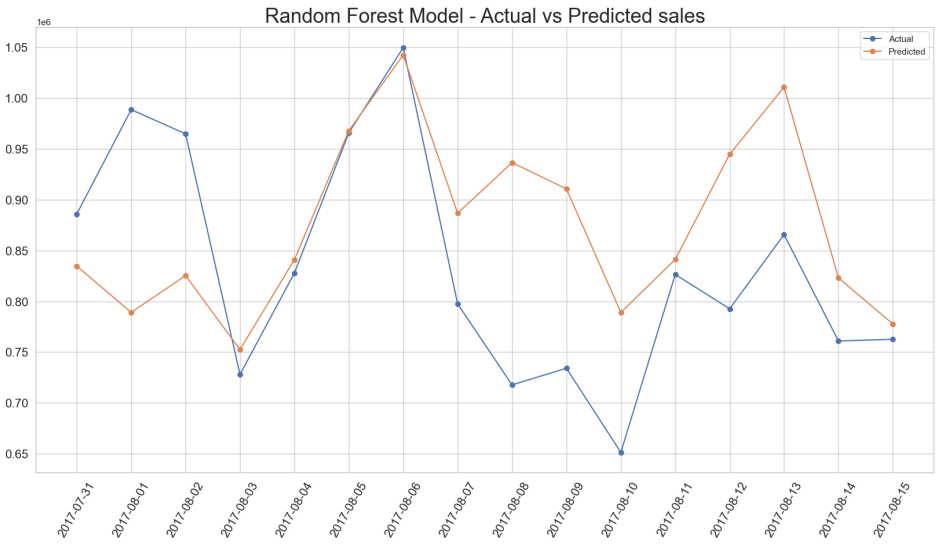


Figure 26. Random Forest Model Actual vs Predicted

6.3.3 Model Limitations

While the RandomForest Regressor is a powerful modelling tool, it is not without its challenges. Training a vast number of decision trees on large datasets can be computationally demanding. Moreover, the performance of the RandomForest model is highly sensitive to its hyperparameters. Precise tuning of these parameters is essential but can be a computationally intensive process, especially in large-scale applications [18].

6.4 ARIMA (Autoregressive Integrated Moving Average)

ARIMA stands as a prominent model in the realm of sales forecasting, particularly due to its specialisation in time series analysis. This model is especially relevant in the context of grocery sales forecasting, where time-dependent patterns play a crucial role.

6.4.1 ARIMA's Time Series Forecasting Approach

ARIMA's approach to forecasting is threefold, involving autoregression, integration, and moving averages:

Autoregression (AR): This aspect of the model, denoted by 'p', captures the relationship between a current observation and a specified number of lagged observations. This feature is crucial for understanding how past sales data influence future sales.

Integration (I): Represented by 'd', integration involves differencing the raw observations to make the time series stationary. Stationarity is a key requirement for many time series models, as it ensures consistent statistical properties over time, which is vital for reliable forecasting.

Moving Average (MA): Denoted by 'q', this component of the model addresses the relationship between an observation and the moving average of lagged observations. The moving average component helps in smoothing out short-term fluctuations, allowing the model to capture more stable trends in the data.

Figure 27 below depicts how ARIMA predicted the sales trend for the given period.

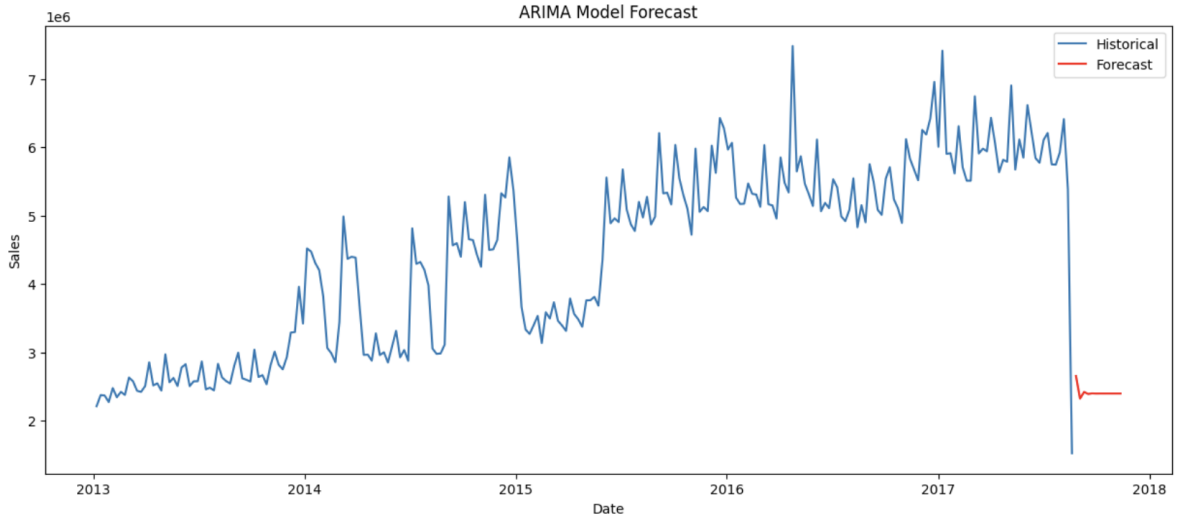


Figure 27. ARIMA Model Forecast

Section 7: Comparing Models

7.1 Importance of Suitable Evaluation Metrics

In the realm of sales forecasting, the selection of appropriate evaluation metrics is essential. These metrics should not only capture the underlying patterns in the sales data but also effectively address seasonal trends and temporal aspects of the data. The choice of metrics plays a pivotal role in determining the most effective model for our sales forecasting task.

7.2 Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE)

Two widely recognized metrics in the field of predictive modelling are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE):

RMSE: RMSE is particularly valuable due to its sensitivity to larger errors. This metric helps in understanding the impact of larger deviations from the actual values. The RMSE score,

being in the same units as the target variable (in this case, 'sales'), offers a direct and interpretable measure of prediction error [19].

MAE: MAE is favoured for its robustness to outliers and its straightforward interpretability. It represents the average absolute difference between the predicted values and the actual values, making it an intuitive measure of model accuracy [20].

7.3 Model Performance Analysis

Utilising RMSE and MAE, we have assessed the performance of our models: Linear Regression, XGBoost Regressor, and RandomForest Regressor. The comparative analysis is tabulated below in Table 1:

Rank	Model	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)
1	RandomForestRegressor	79.355	323.684
2	XGBoostRegressor	78.305	324.131
3	Linear Regression	554.014	1054.399

Table1. Model Performance Analysis, MAE and RMSE

Based on this comparative analysis, RandomForestRegressor emerges as the most effective model for our sales forecasting task. Exhibiting the lowest values of both MAE and RMSE, it demonstrates greater accuracy in predicting sales, making it the preferable choice for our dataset.

7.4 Hyperparameter Optimization Using GridSearchCV for RandomForest:

7.4.1 Fine-Tuning RandomForest Regressor

After identifying the RandomForest Regressor as the most effective model among those tested, our next step involved hyperparameter tuning. This process is critical to optimise the model's performance, particularly when applied to real-world sales data.

7.4.2 Challenges in GridSearchCV Application

We employed GridSearchCV, a comprehensive hyperparameter optimization tool, to systematically iterate through multiple combinations of hyperparameter values. However, due

to the extensive size of our dataset and the limitations of our computational resources, this approach faced challenges in yielding results.

The following parameters combinations were iterated and tested with cv = 5:

```
param_grid = {
    'n_estimators': [300],
    'min_samples_split': [5],
    'min_samples_leaf': [5],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

```
best_params
```

```
{'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 5,
 'n_estimators': 300}
```

```
best_model
```

```
RandomForestRegressor
RandomForestRegressor(max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=5, n_estimators=300, oob_score=True,
                      warm_start=True)
```

Optimised Graph after Hyperparameter Tuning using GridSearchCV

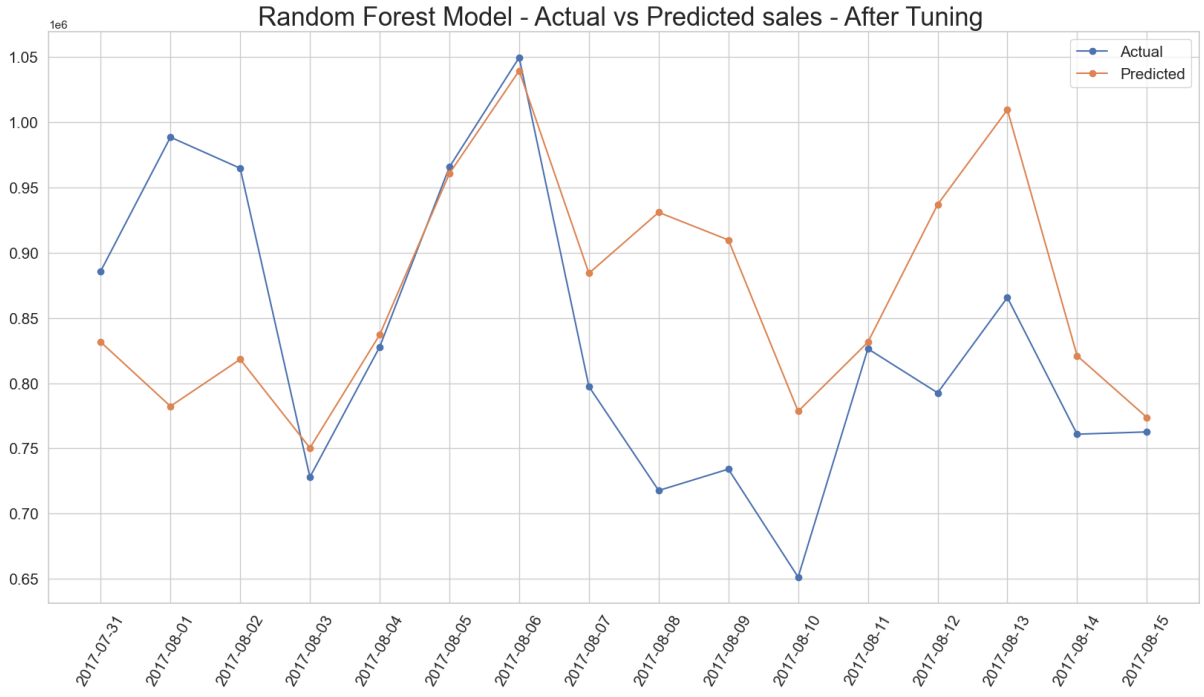


Figure 28. Random Forest Model Actual vs Predicted after optimization

Optimization Results

Rank	Model	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)
1	RandomForestRegressor	79.355	323.684
2	XGBoostRegressor	78.305	324.131
3	Linear Regression	554.014	1054.399
4	**RandomForestRegressor - After Optimization**	**76.853**	**314.748**

Table1. Model Performance Analysis, MAE and RMSE

Optimization Results on Unseen Data

Rank	Model	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)
1	RandomForestRegressor - Optimization	**83.966**	**290.730**

Table1. Model Performance Analysis, MAE and RMSE

Section 8: PERFORMANCE EVALUATION ON UNSEEN DATA

8.1 Implementing RandomForest Regressor on Unseen Data

After thorough model comparisons, the RandomForest Regressor was chosen as our most reliable model. The next critical step involved evaluating its effectiveness on unseen data.

8.2 Predictions for Unseen Data Period

The unseen data encompasses the period from July 31, 2017, to August 15, 2017. We used the RandomForest model to predict sales during this timeframe. Subsequently, the performance of these predictions was assessed using key metrics:

Mean Absolute Error (MAE): 87.0905

Mean Squared Error (MSE): 90492.345

Root Mean Squared Error (RMSE): 300.819

8.3 Examining Model Robustness

To ensure the integrity and reliability of our RandomForest model, we addressed potential issues of overfitting and data leakage:

Handling Data Leakage and Overfitting: We mitigated data leakage concerns by utilising the Out-of-Bag (OOB) score parameter. This feature in RandomForest models ensures a fair assessment by using a portion of the training data that was not used during the tree-building process for validation. Our model's performance was further evaluated for overfitting as well.

8.4 OOB Error Trend Analysis

Figure 28 provides a visual representation of the OOB error trend in our RandomForest model. Initially, the addition of trees aids in enhancing the diversity and generalisation ability of the ensemble. However, as the model captures most patterns from the training data, further increasing the number of trees becomes less effective in boosting model performance.

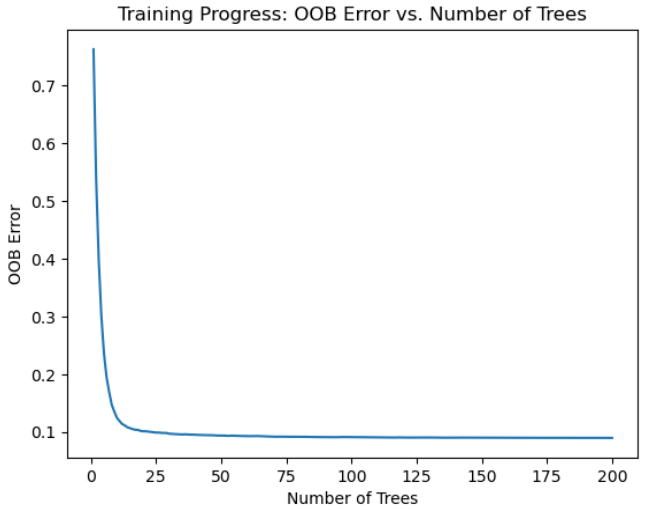
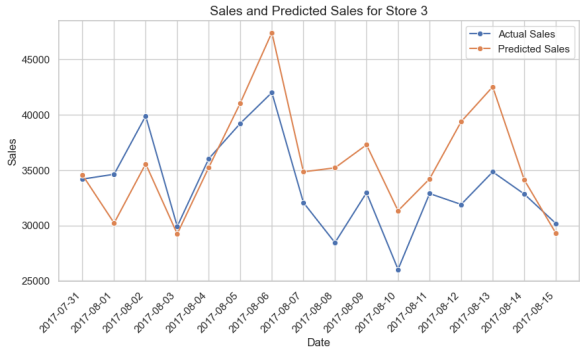


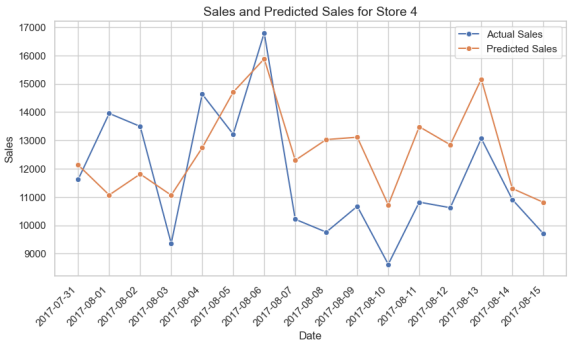
Figure 29. Training Process OOB Error vs No. of Trees

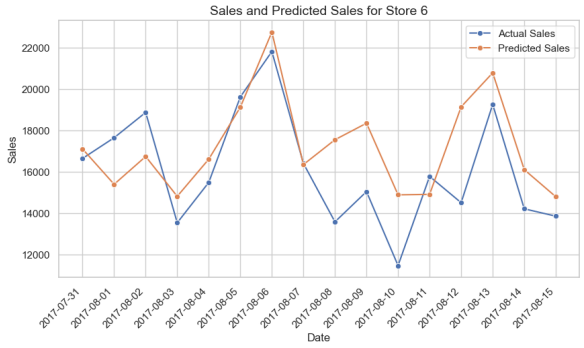
Section 9: Results Presentation

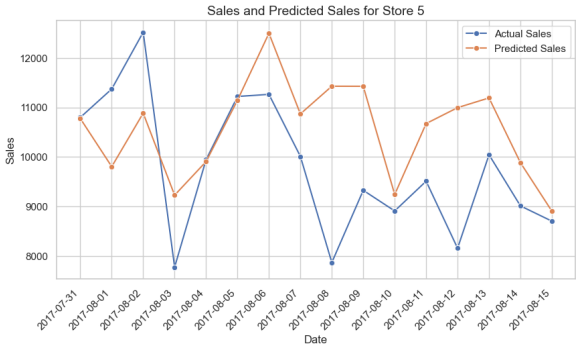
The outcome of our **Random Forest model** has given the predictions of the sales for each store and for each product type for 16 days (31st July 2017 till 15th August 2017)

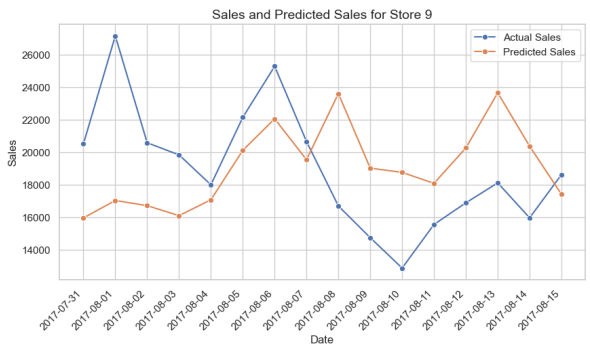
1. Actual sales vs Predicted sales in 54 stores:

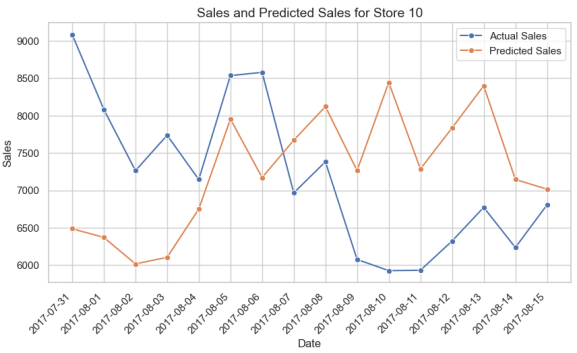


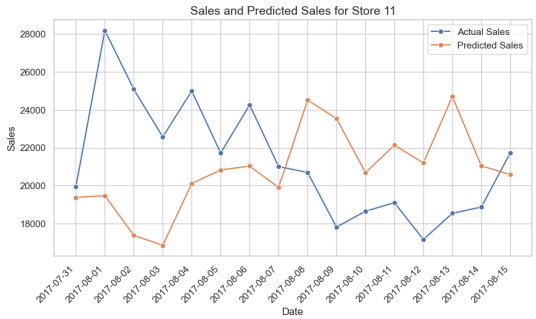


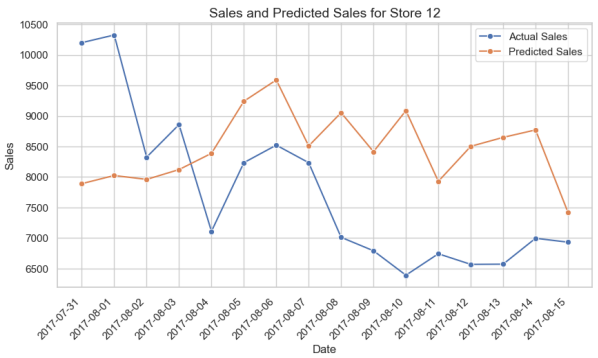


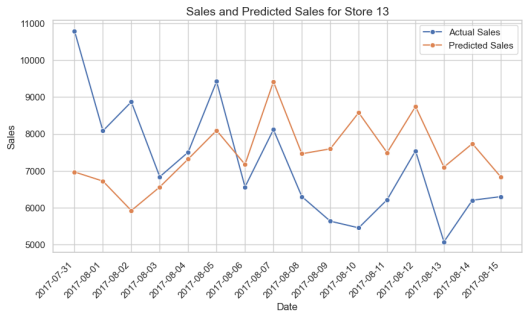


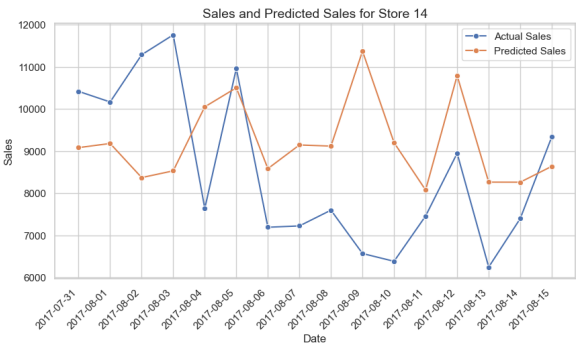

Sales and Predicted Sales for Store 9
Actual Sales
Predicted Sales
Sales
Date

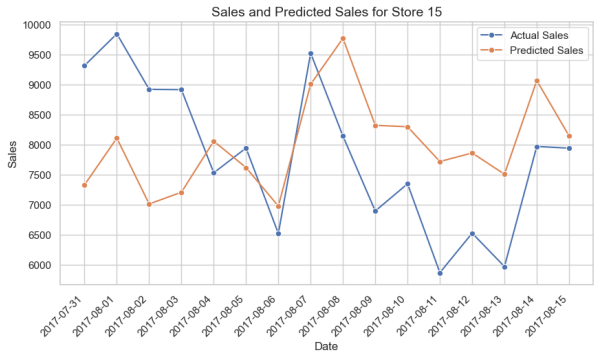

Sales and Predicted Sales for Store 10
Actual Sales
Predicted Sales
Sales
Date

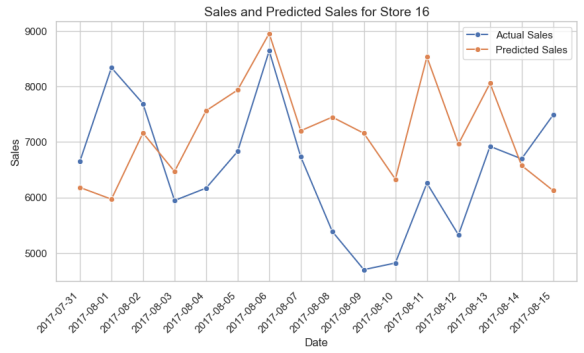

Sales and Predicted Sales for Store 11
Actual Sales
Predicted Sales
Sales
Date

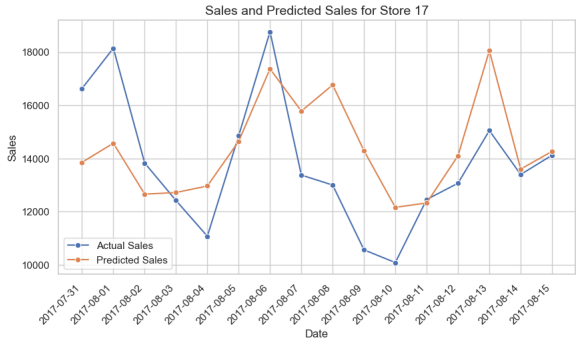

Sales and Predicted Sales for Store 12
Actual Sales
Predicted Sales
Sales
Date

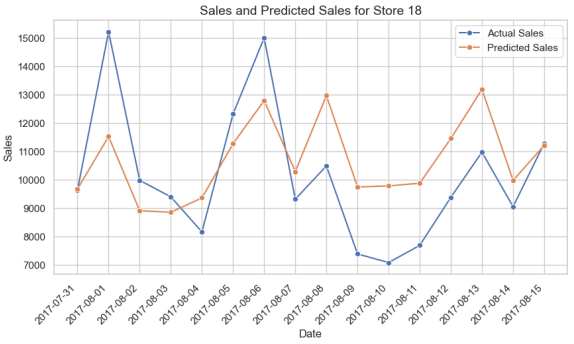

Sales and Predicted Sales for Store 13
Actual Sales
Predicted Sales
Sales
Date

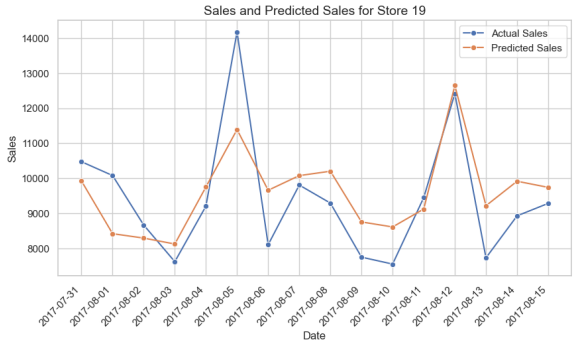

Sales and Predicted Sales for Store 14
Actual Sales
Predicted Sales
Sales
Date

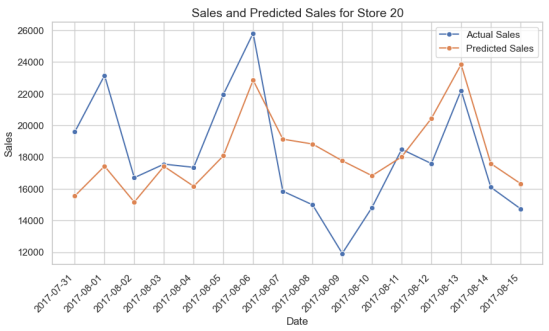

Sales and Predicted Sales for Store 15
Actual Sales
Predicted Sales
Sales
Date

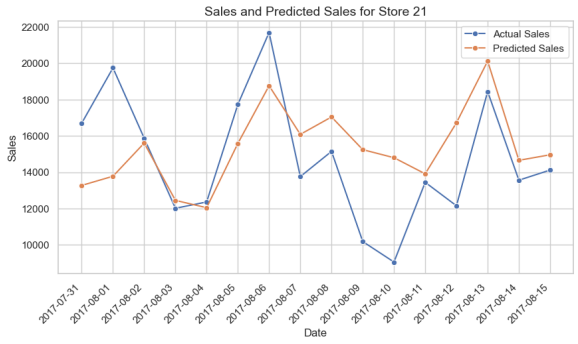

Sales and Predicted Sales for Store 16
Actual Sales
Predicted Sales
Sales
Date

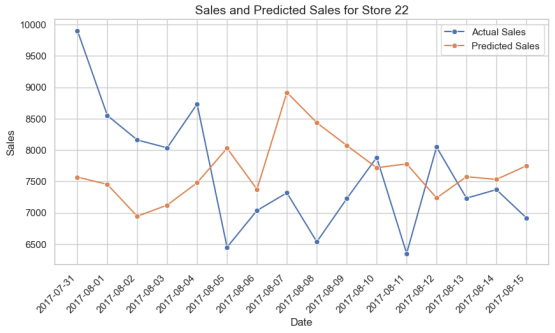

Sales and Predicted Sales for Store 17
Actual Sales
Predicted Sales
Sales
Date

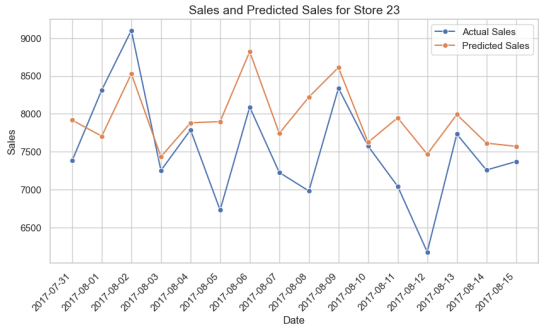

Sales and Predicted Sales for Store 18
Actual Sales
Predicted Sales
Sales
Date

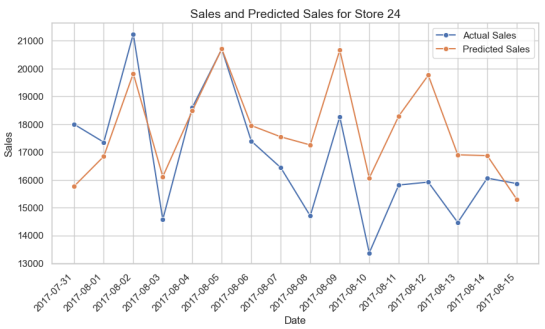

Sales and Predicted Sales for Store 19
Actual Sales
Predicted Sales
Sales
Date

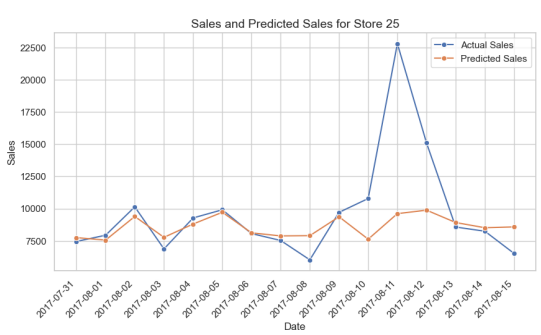

Sales and Predicted Sales for Store 20
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 21
Actual Sales
Predicted Sales
Sales
Date

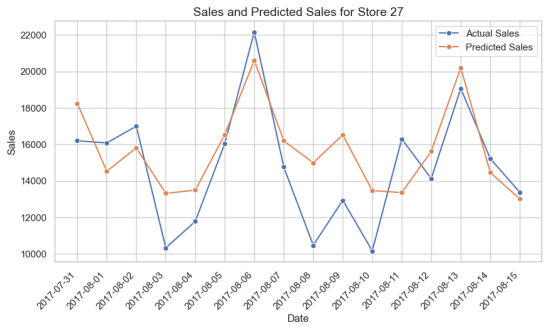

Sales and Predicted Sales for Store 22
Actual Sales
Predicted Sales
Sales
Date

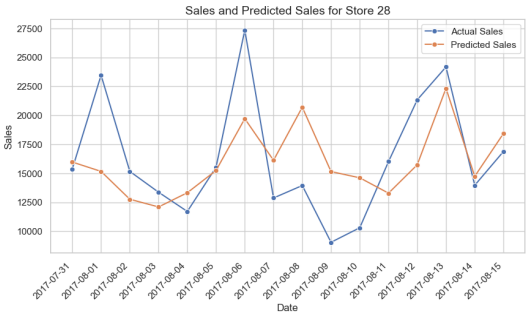

Sales and Predicted Sales for Store 23
Actual Sales
Predicted Sales
Sales
Date

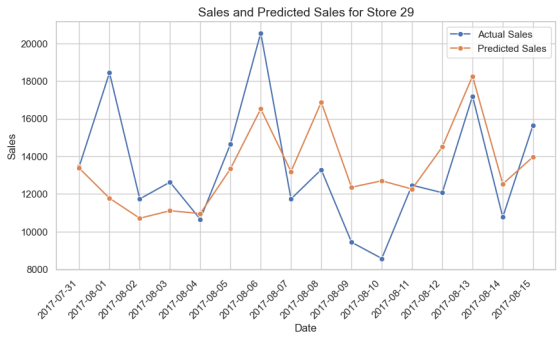

Sales and Predicted Sales for Store 24
Actual Sales
Predicted Sales
Sales
Date

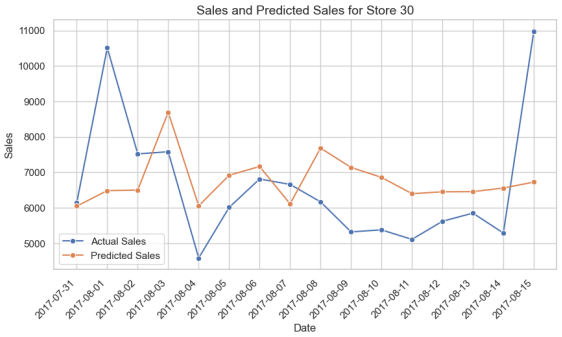

Sales and Predicted Sales for Store 25
Actual Sales
Predicted Sales
Sales
Date

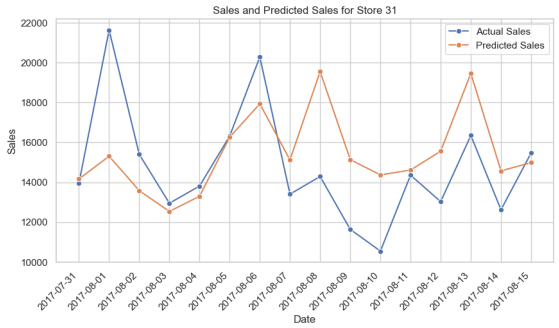

Sales and Predicted Sales for Store 26
Actual Sales
Predicted Sales
Sales
Date

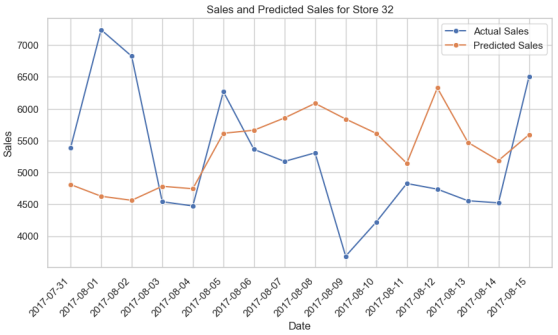

Sales and Predicted Sales for Store 27
Actual Sales
Predicted Sales
Sales
Date

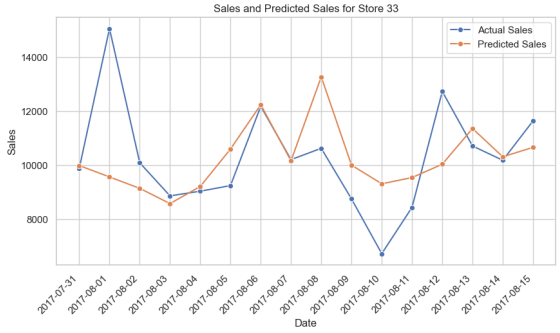

Sales and Predicted Sales for Store 28
Actual Sales
Predicted Sales
Sales
Date

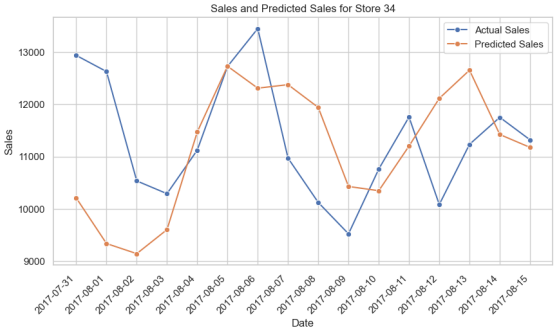

Sales and Predicted Sales for Store 29
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 30
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 31
Actual Sales
Predicted Sales
Sales
Date

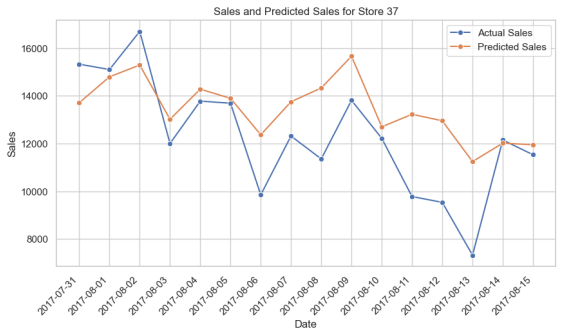

Sales and Predicted Sales for Store 32
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 33
Actual Sales
Predicted Sales
Sales
Date

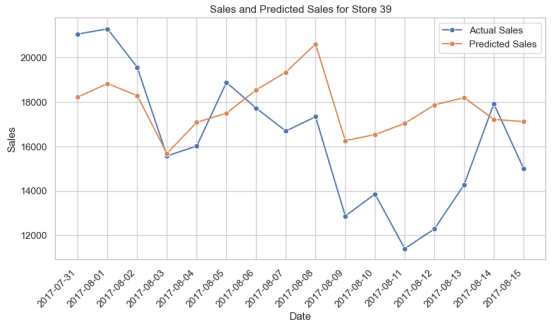

Sales and Predicted Sales for Store 34
Actual Sales
Predicted Sales
Sales
Date

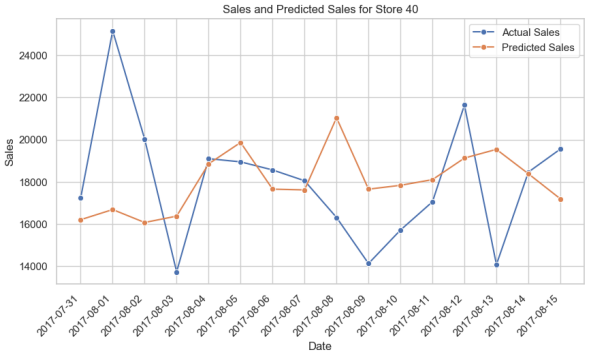

Sales and Predicted Sales for Store 35
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 36
Actual Sales
Predicted Sales
Sales
Date

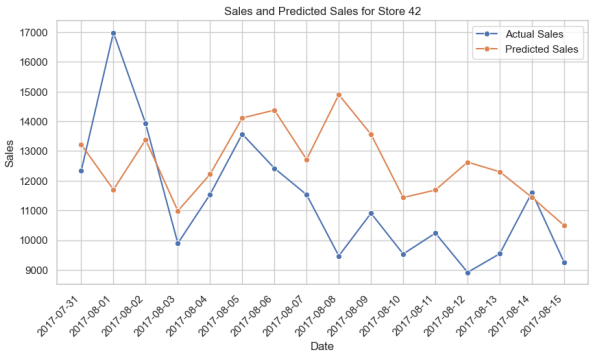

Sales and Predicted Sales for Store 37
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 38
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 39
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 40
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 41
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 42
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 43
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 44
Actual Sales
Predicted Sales
Sales
Date

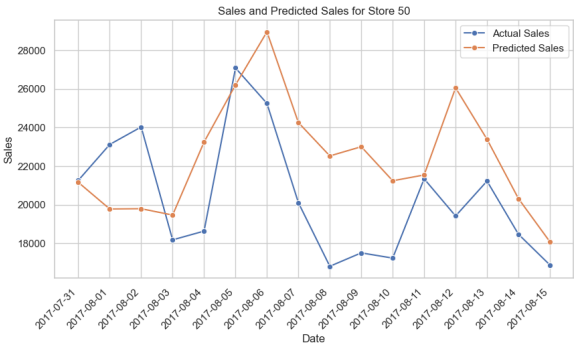

Sales and Predicted Sales for Store 45
Actual Sales
Predicted Sales
Sales
Date

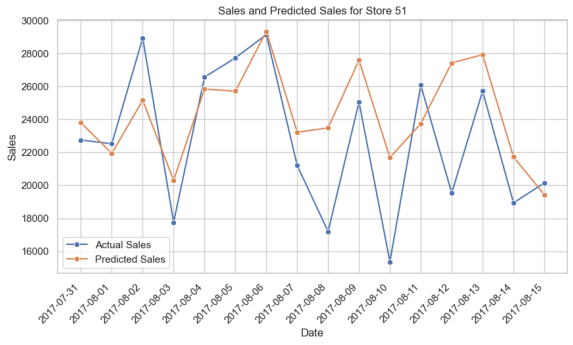

Sales and Predicted Sales for Store 46
Actual Sales
Predicted Sales
Sales
Date

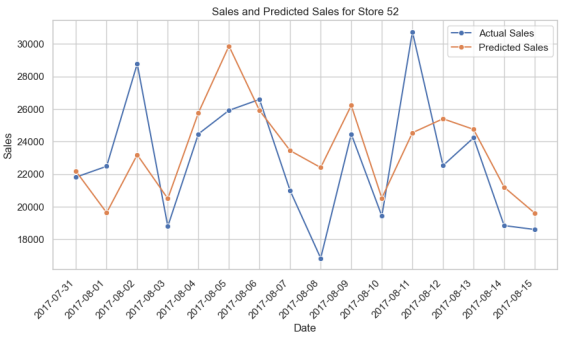

Sales and Predicted Sales for Store 47
Actual Sales
Predicted Sales
Sales
Date

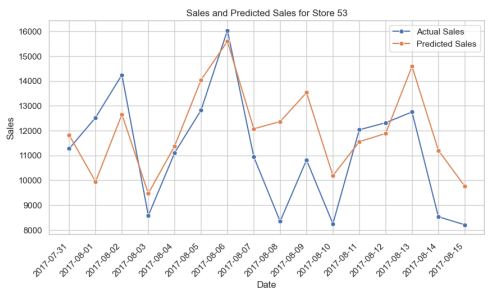

Sales and Predicted Sales for Store 48
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 49
Actual Sales
Predicted Sales
Sales
Date


Sales and Predicted Sales for Store 50
Actual Sales
Predicted Sales
Sales
Date

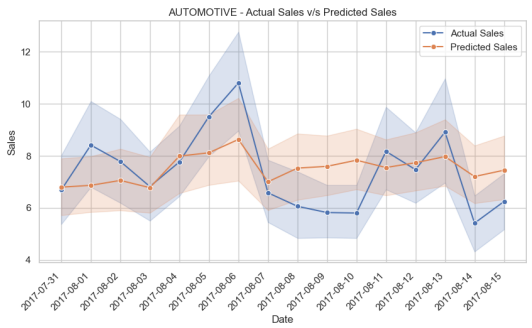

Sales and Predicted Sales for Store 51
Actual Sales
Predicted Sales
Sales
Date

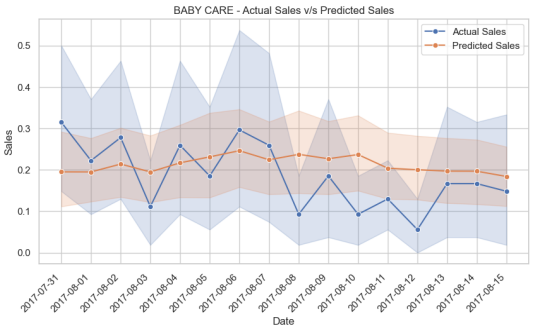

Sales and Predicted Sales for Store 52
Actual Sales
Predicted Sales
Sales
Date

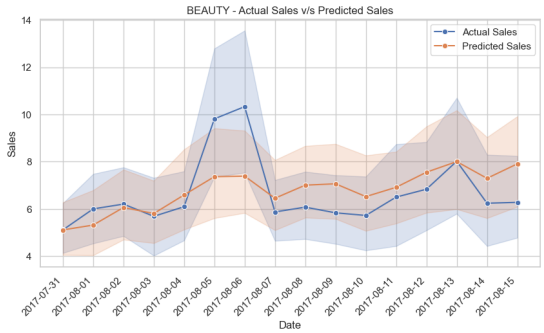

Sales and Predicted Sales for Store 53
Actual Sales
Predicted Sales
Sales
Date

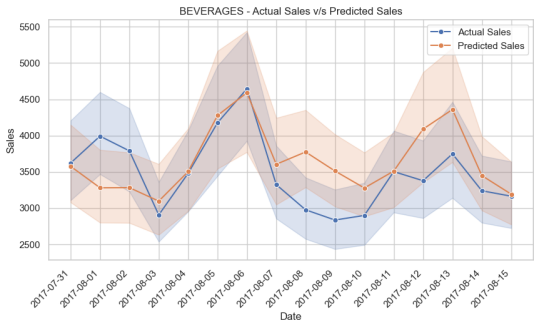

Sales and Predicted Sales for Store 54
Actual Sales
Predicted Sales
Sales
Date

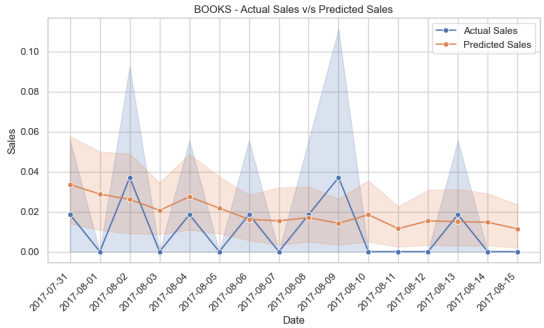
Actual sales vs Predicted sales on 33 product types:

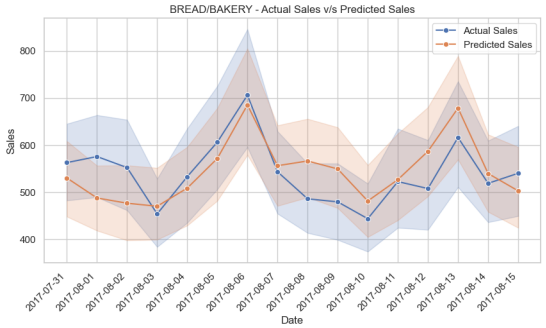


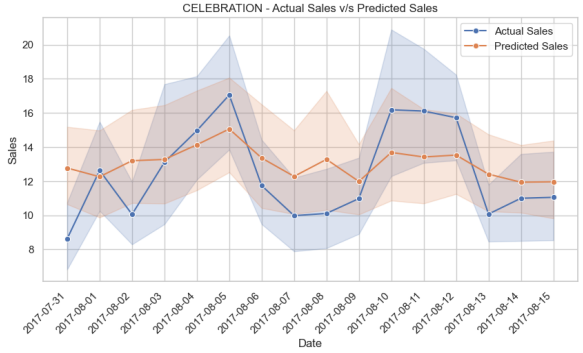


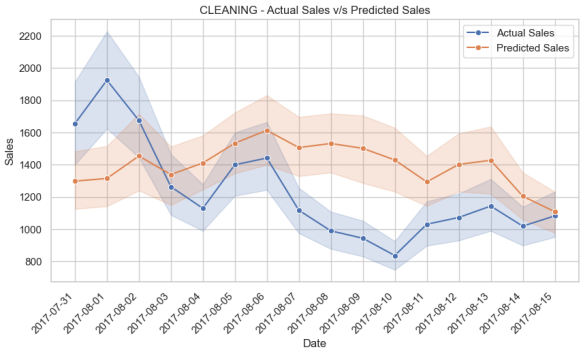


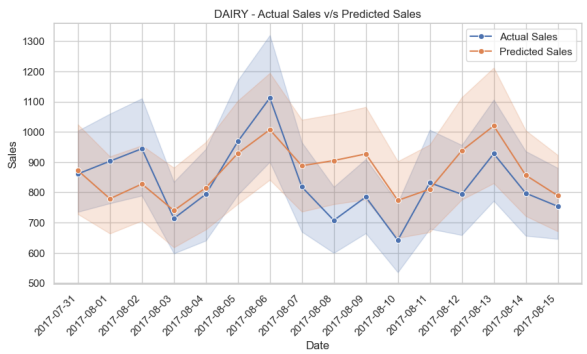


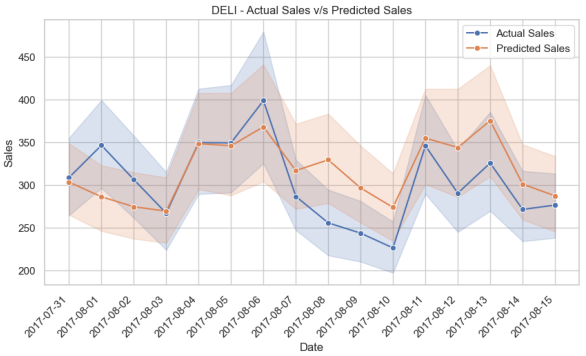


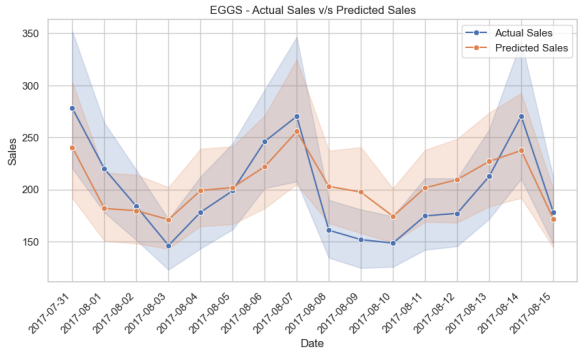


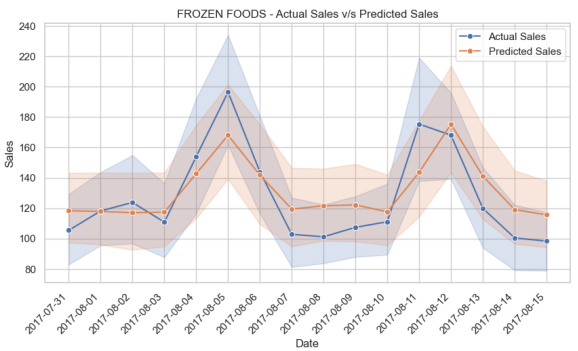


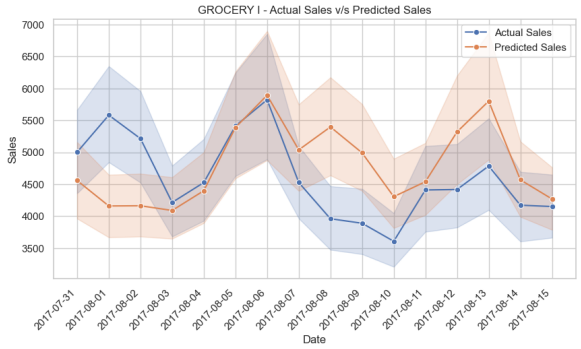


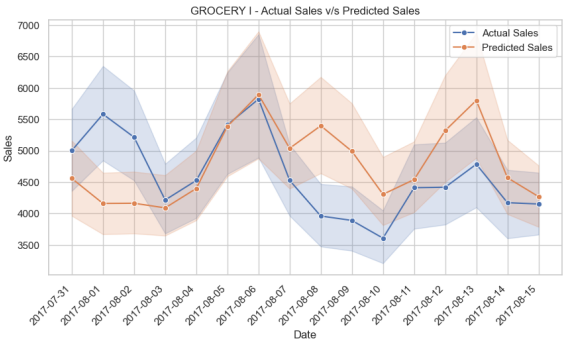

DAIRY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

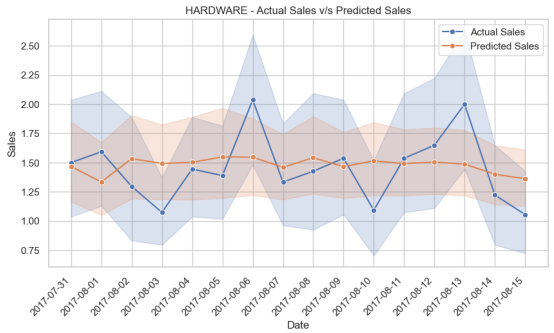

DELI - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

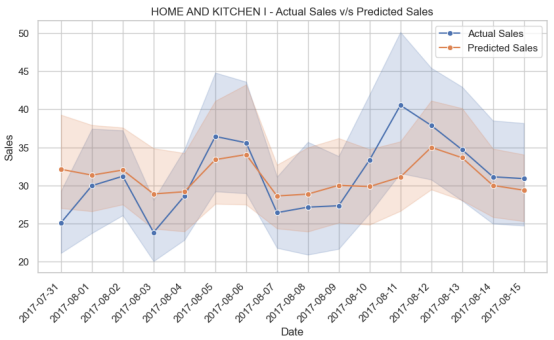

EGGS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

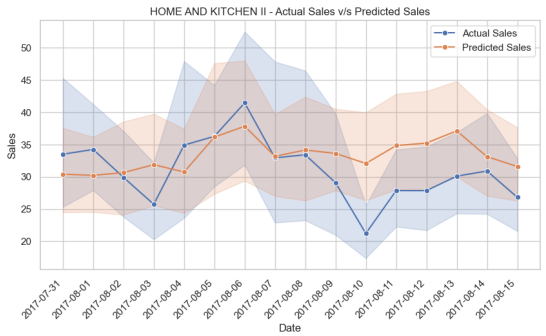

FROZEN FOODS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

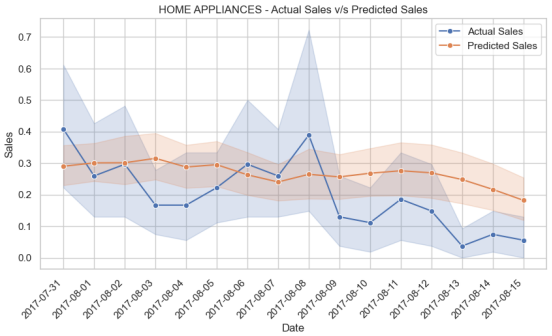

GROCERY I - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

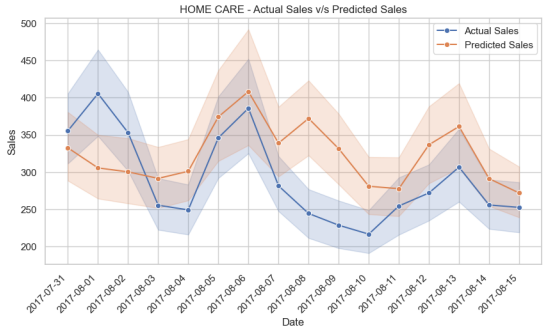

GROCERY I - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

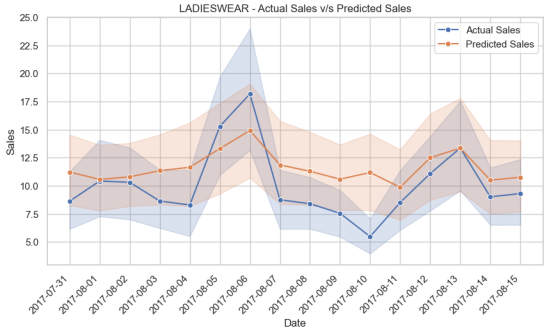

HARDWARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

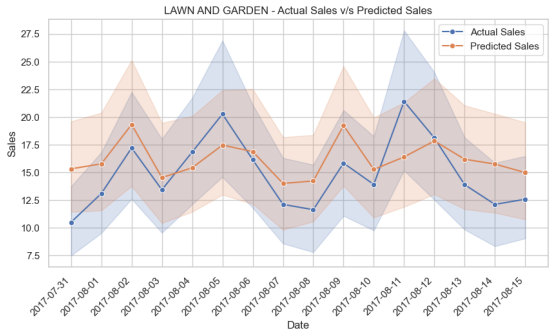

HOME AND KITCHEN I - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

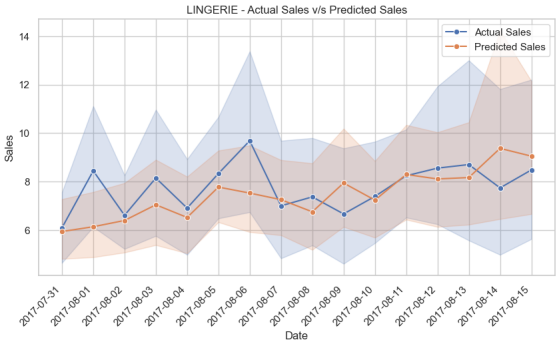

HOME AND KITCHEN II - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

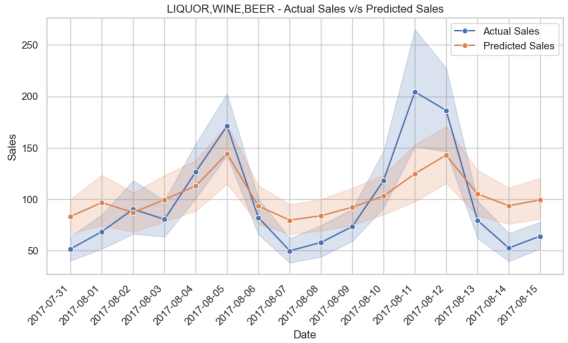

HOME APPLIANCES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

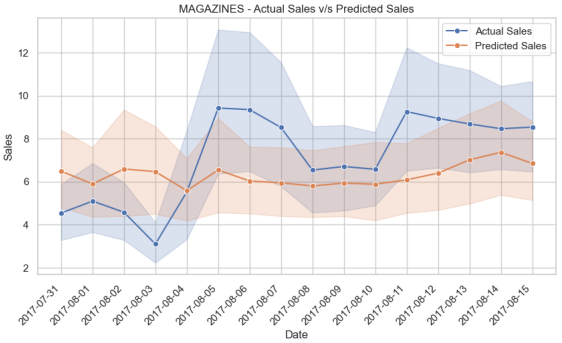

HOME CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

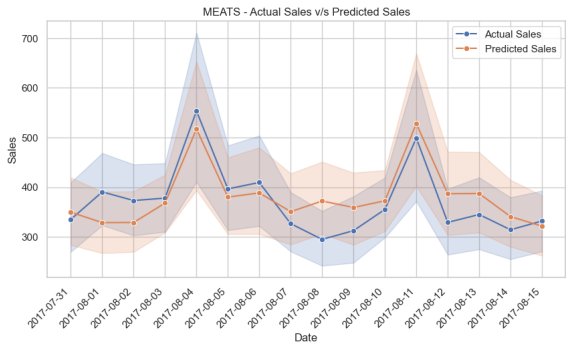

LADIESWEAR - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

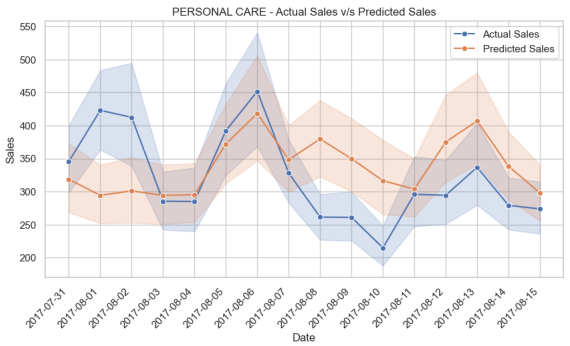

LAWN AND GARDEN - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

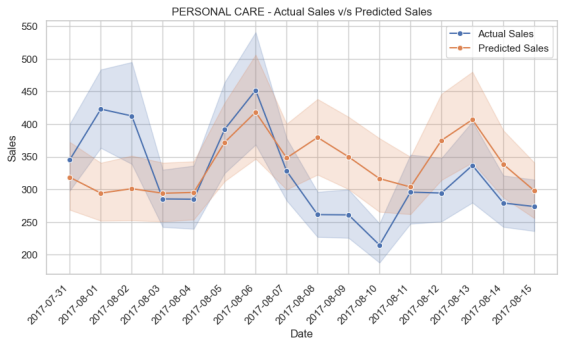

LINGERIE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

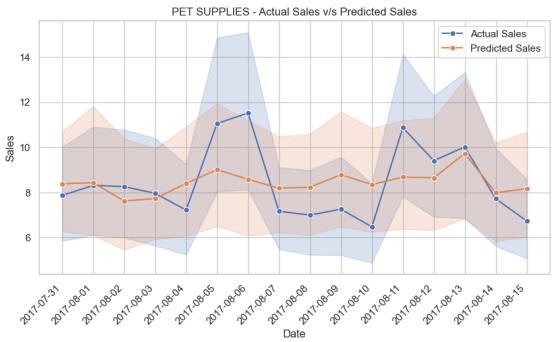

LIQUOR,WINE,BEER - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

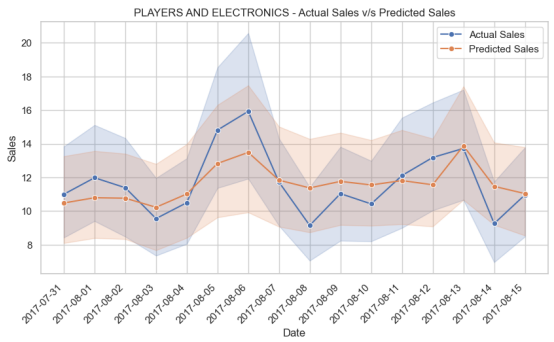

MAGAZINES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

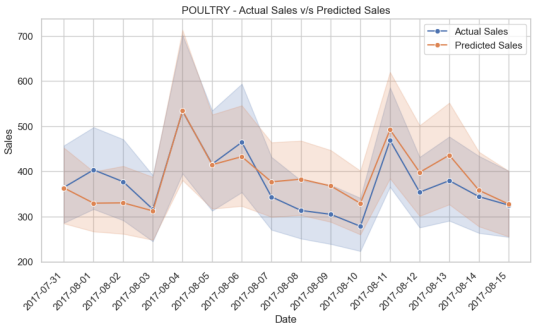

MEATS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

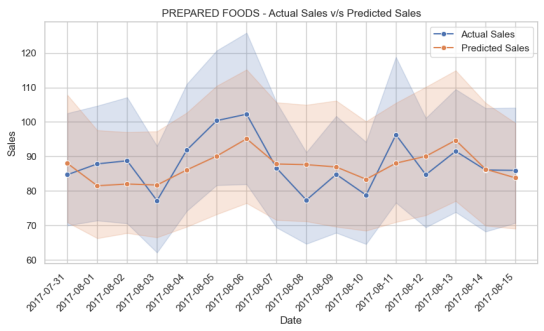

PERSONAL CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

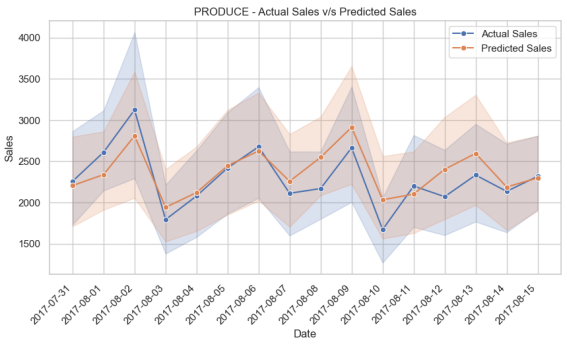

PERSONAL CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

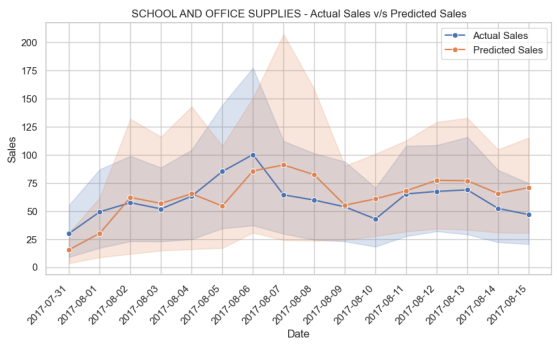

PET SUPPLIES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date

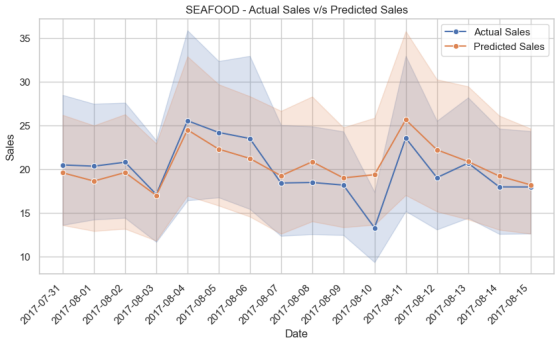

PLAYERS AND ELECTRONICS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date


POULTRY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date


PREPARED FOODS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date


PRODUCE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date





Section 10: Conclusion

As we conclude this report, it is evident that for retailers and businesses alike, the ability to forecast future demands is invaluable. In our exploration, we evaluated four distinct algorithms – Linear Regression, XGBoost, Random Forest, and ARIMA – for the purpose of sales forecasting. Our analysis, encompassing five years of historical data, revealed that the RandomForest Regression algorithm outperformed its counterparts. With a Root Mean Squared Error (RMSE) of 79.355 and a Mean Absolute Error (MAE) of 323.684, it demonstrated superior predictive accuracy. Section 9 illustrates a comparative analysis of the actual versus predicted sales values for each product in every store for the period from July 31, 2017, to August 15, 2017.

This report's insights are particularly relevant for ABC grocery retailers. The refined RandomForest model can serve as a tool for forecasting daily sales, thereby enabling data-driven decisions that can enhance sales, improve customer satisfaction, and optimise inventory management across key departments. Looking ahead, our objective is to further refine our sales forecasting approach. We plan to delve deeper into more advanced time series algorithms that can capture long-term trends and subtle patterns more effectively. This will involve incorporating a broader historical data spectrum and experimenting with new feature extraction, such as examining the interplay between special offers and sales. We also intend to integrate external factors like holidays, significant events, and the proximity of competitor stores into our analysis. Additionally, investing in high-end technical infrastructure will enable us to harness the full potential of our extensive datasets.

Section 11: References

[1] Forecasting: Methods and Applications" by Spyros Makridakis, Steven C. Wheelwright, and Rob J Hyndman
[2] Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die" by Eric Siegel
[3] Data-Driven Marketing: The 15 Metrics Everyone in Marketing Should Know" by Mark Jeffery
[4] Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results" by Bernard Marr
[5] McKinney, W. (2017). "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython". O'Reilly Media, which details the use of pandas for data manipulation and encoding techniques.
[6] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). "An Introduction to Statistical Learning." Springer Texts in Statistics.
[7] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). "Introduction to Linear Regression Analysis." Wiley.
[8] Hyndman, R. J., & Athanasopoulos, G. (2018). "Forecasting: principles and practice." OTexts.
[9] Draper, N. R., & Smith, H. (1998). "Applied Regression Analysis." Wiley Series in Probability and Statistics.
[10] Chen, T., & Guestrin, C. (2016). "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
[11] Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." The Annals of Statistics.
[12] Breiman, L. (1996). "Bagging predictors." Machine Learning.
[13] Willmott, C. J., & Matsuura, K. (2005). "Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance." Climate Research.
[14] Probst, P., Wright, M. N., & Boulesteix, A. L. (2019). "Hyperparameters and tuning strategies for random forest." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery.
[15] Breiman, L. (2001). "Random Forests." Machine Learning.
[16] Liaw, A., & Wiener, M. (2002). "Classification and Regression by randomForest." R News.
[17] Willmott, C. J., & Matsuura, K. (2005). "Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance." Climate Research.
[18] Probst, P., Wright, M. N., & Boulesteix, A. L. (2019). "Hyperparameters and tuning strategies for random forest." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery.
[19] Chai, T., & Draxler, R. R. (2014). "Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? – Arguments against avoiding RMSE in the literature." Geoscientific Model Development.
[20] Willmott, C. J., & Matsuura, K. (2005). "Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance." Climate Research.

Importing Libraries

In [221…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from plotly.offline import plot, iplot, init_notebook_mode
from ydata_profiling import ProfileReport
from scipy.stats import ttest_ind
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression, chi2
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error


import warnings
warnings.filterwarnings('ignore')
```

In [222…

```
data = pd.read_csv("Products_Information.csv")
data.head()
```

Out[222]:

	id	date	store_nbr	product_type	sales	special_offer
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BEAUTY	0.0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0
4	4	2013-01-01	1	BOOKS	0.0	0

In [253…

```
data.iloc[0:]
```

Out[253]:

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	is_weekend	sales_lag_1	sales_lag_7	sales_lag_15	sales_lag_30	sales_lag_60	rolling_avg_sales
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	0	1	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	0	2	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	0	3	0	2.0	0.0	0.0	0.0	0.0	0.0000
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	0	4	0	3.0	0.0	0.0	0.0	0.0	0.0000
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	0	5	1	3.0	0.0	0.0	0.0	0.0	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-08-11	2993627	2017-08-11	54	32	0.0	0	2017	7	4	0	2.0	0.0	2.0	1.0	5.0	3.1250
2017-08-12	2995409	2017-08-12	54	32	1.0	1	2017	7	5	1	0.0	3.0	4.0	0.0	0.0	3.0625
2017-08-13	2997191	2017-08-13	54	32	2.0	0	2017	7	6	1	1.0	0.0	4.0	0.0	3.0	2.9375
2017-08-14	2998973	2017-08-14	54	32	0.0	0	2017	7	0	0	2.0	0.0	4.0	2.0	0.0	2.6875
2017-08-15	3000755	2017-08-15	54	32	3.0	0	2017	7	1	0	0.0	12.0	4.0	5.0	2.0	2.6250

3000888 rows × 16 columns

Converting 'date' to datetime object

In [224…

```
data['date'] = pd.to_datetime(data['date'])  # Converting 'date' to datetime object
data.index = data['date']
data = data.rename_axis('index_date')
```

In [225…

```
data.describe(include = "all")
```

Out[225]:

	id	date	store_nbr	product_type	sales	special_offer
count	3.000888e+06	3000888	3.000888e+06	3000888	3.000888e+06	3.000888e+06
unique	NaN	1684	NaN	33	NaN	NaN
top	NaN	2013-01-01 00:00:00	NaN	AUTOMOTIVE	NaN	NaN
freq	NaN	1782	NaN	90936	NaN	NaN
first	NaN	2013-01-01 00:00:00	NaN	NaN	NaN	NaN
last	NaN	2017-08-15 00:00:00	NaN	NaN	NaN	NaN
mean	1.500444e+06	NaN	2.750000e+01	NaN	3.577757e+02	2.602770e+00
std	8.662819e+05	NaN	1.558579e+01	NaN	1.101998e+03	1.221888e+01
min	0.000000e+00	NaN	1.000000e+00	NaN	0.000000e+00	0.000000e+00
25%	7.502218e+05	NaN	1.400000e+01	NaN	0.000000e+00	0.000000e+00
50%	1.500444e+06	NaN	2.750000e+01	NaN	1.100000e+01	0.000000e+00
75%	2.250665e+06	NaN	4.100000e+01	NaN	1.958473e+02	0.000000e+00
max	3.000887e+06	NaN	5.400000e+01	NaN	1.247170e+05	7.410000e+02

In [226…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3000888 entries, 2013-01-01 to 2017-08-15
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   id             int64
 1   date           datetime64[ns]
 2   store_nbr      int64
 3   product_type   object
 4   sales          float64
 5   special_offer  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(1)
memory usage: 160.3+ MB
```

In [227…
```
data.shape
```

Out[227]:
```
(3000888, 6)
```

In [228…
```
data.isnull().sum()
```

Out[228]:
```
id              0
date            0
store_nbr       0
product_type    0
sales           0
special_offer   0
dtype: int64
```

In [229…
```
data.nunique()
```

Out[229]:
```
id              3000888
date               1684
store_nbr            54
product_type         33
sales             379610
special_offer        362
dtype: int64
```

In [230…
```
data["store_nbr"].unique()
```

Out[230]:
```
array([ 1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  2, 20, 21, 22, 23, 24,
       25, 26, 27, 28, 29,  3, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,  4,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49,  5, 50, 51, 52, 53, 54,  6,
        7,  8,  9], dtype=int64)
```

In [254…
```
data["product_type"].unique()
```

Out[254]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32])
```

Creating date features

In [232…
```
# Creating date features
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day_of_week'] = data['date'].dt.dayofweek + 1  # Adding 1 to start the week from Monday (1) to Sunday (7)
data['is_weekend'] = (data['date'].dt.weekday // 5 ==1).astype(int)

data
```

Out[232]:

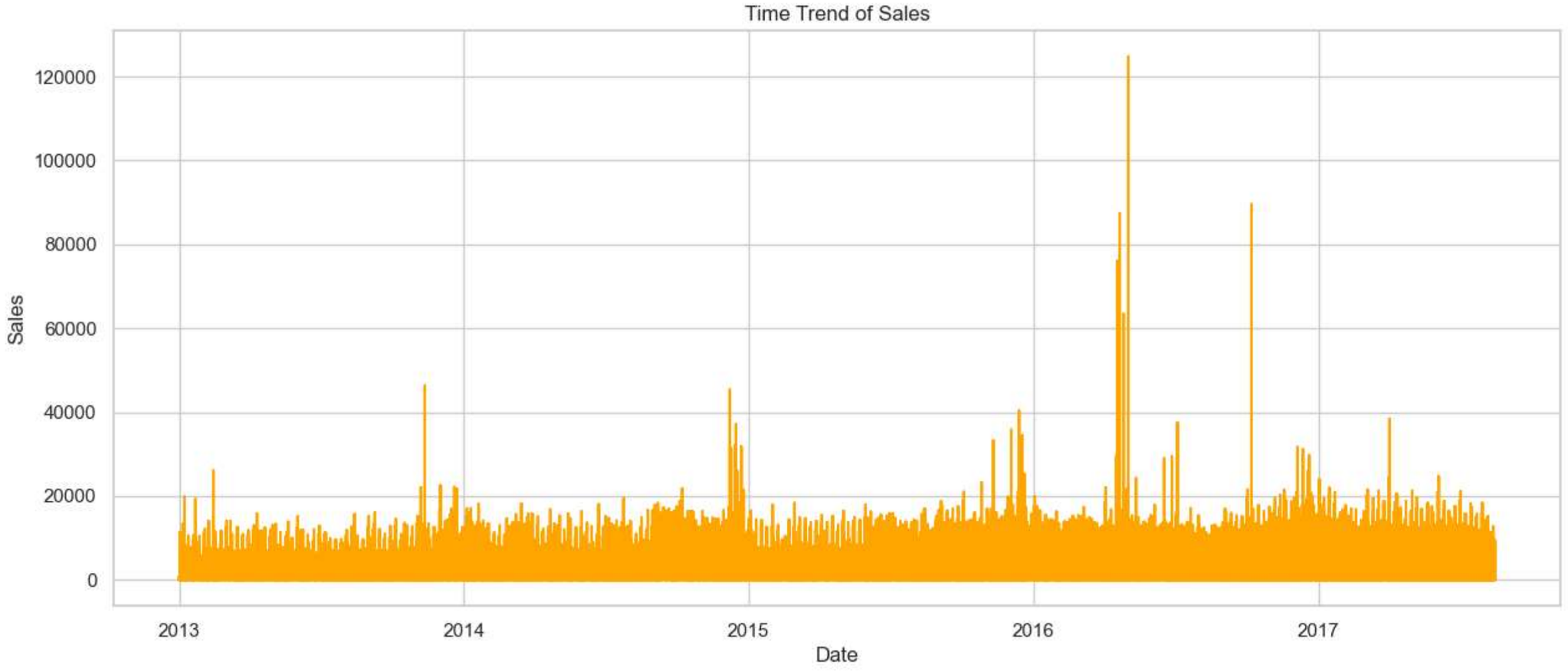
index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	is_weekend
2013-01-01	0	2013-01-01	1	AUTOMOTIVE	0.000	0	2013	1	2	0
2013-01-01	1	2013-01-01	1	BABY CARE	0.000	0	2013	1	2	0
2013-01-01	2	2013-01-01	1	BEAUTY	0.000	0	2013	1	2	0
2013-01-01	3	2013-01-01	1	BEVERAGES	0.000	0	2013	1	2	0
2013-01-01	4	2013-01-01	1	BOOKS	0.000	0	2013	1	2	0
...	...	...	...	...	...	...	...	...	...	...
2017-08-15	3000883	2017-08-15	9	POULTRY	438.133	0	2017	8	2	0
2017-08-15	3000884	2017-08-15	9	PREPARED FOODS	154.553	1	2017	8	2	0
2017-08-15	3000885	2017-08-15	9	PRODUCE	2419.729	148	2017	8	2	0
2017-08-15	3000886	2017-08-15	9	SCHOOL AND OFFICE SUPPLIES	121.000	8	2017	8	2	0
2017-08-15	3000887	2017-08-15	9	SEAFOOD	16.000	0	2017	8	2	0

3000888 rows × 10 columns

Exploratory Data Analysis

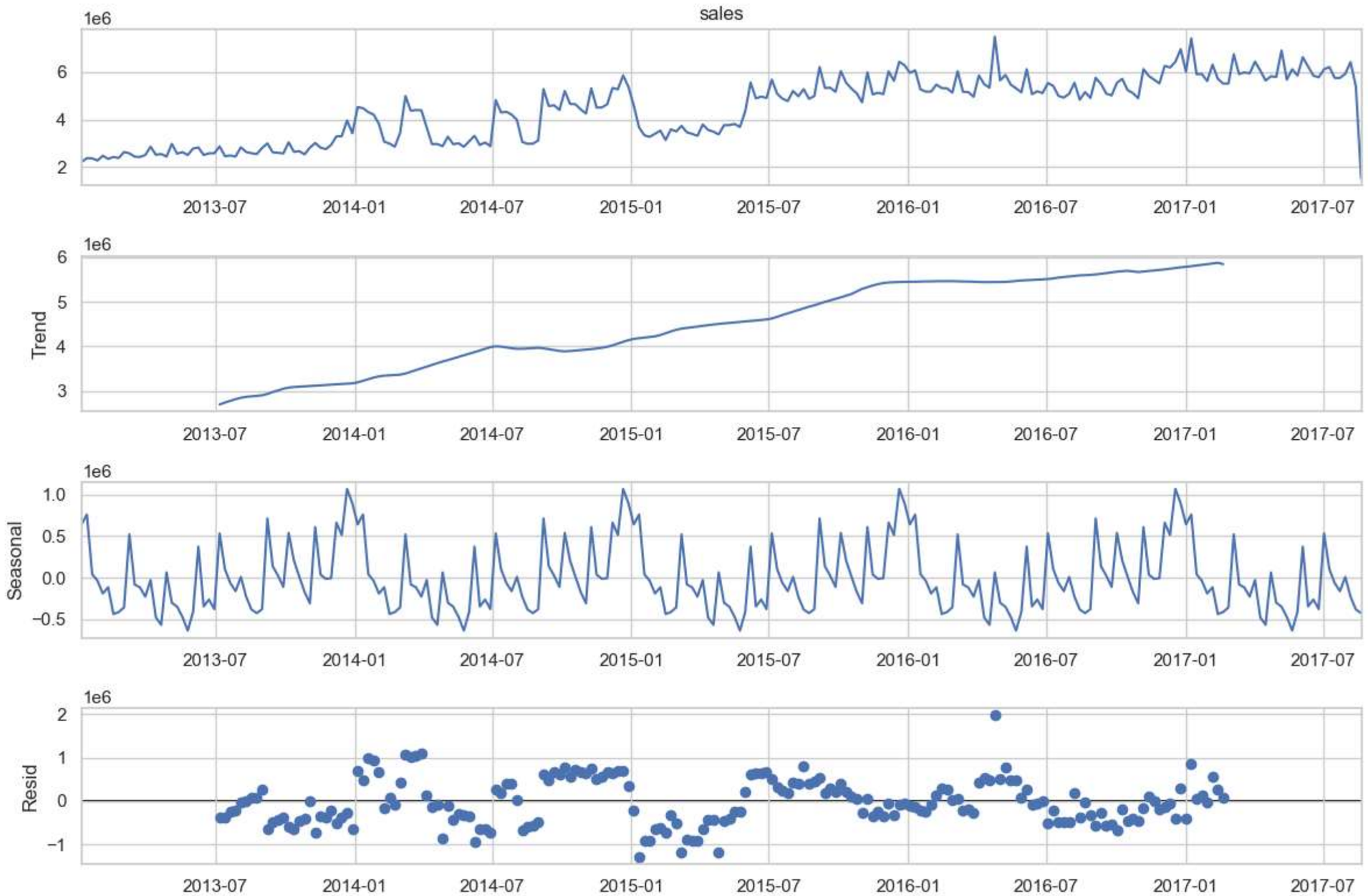
In [233…

```
# Time trend of sales
plt.figure(figsize=(15, 6))
plt.plot(data['sales'], color='orange')
plt.title('Time Trend of Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.show()
```



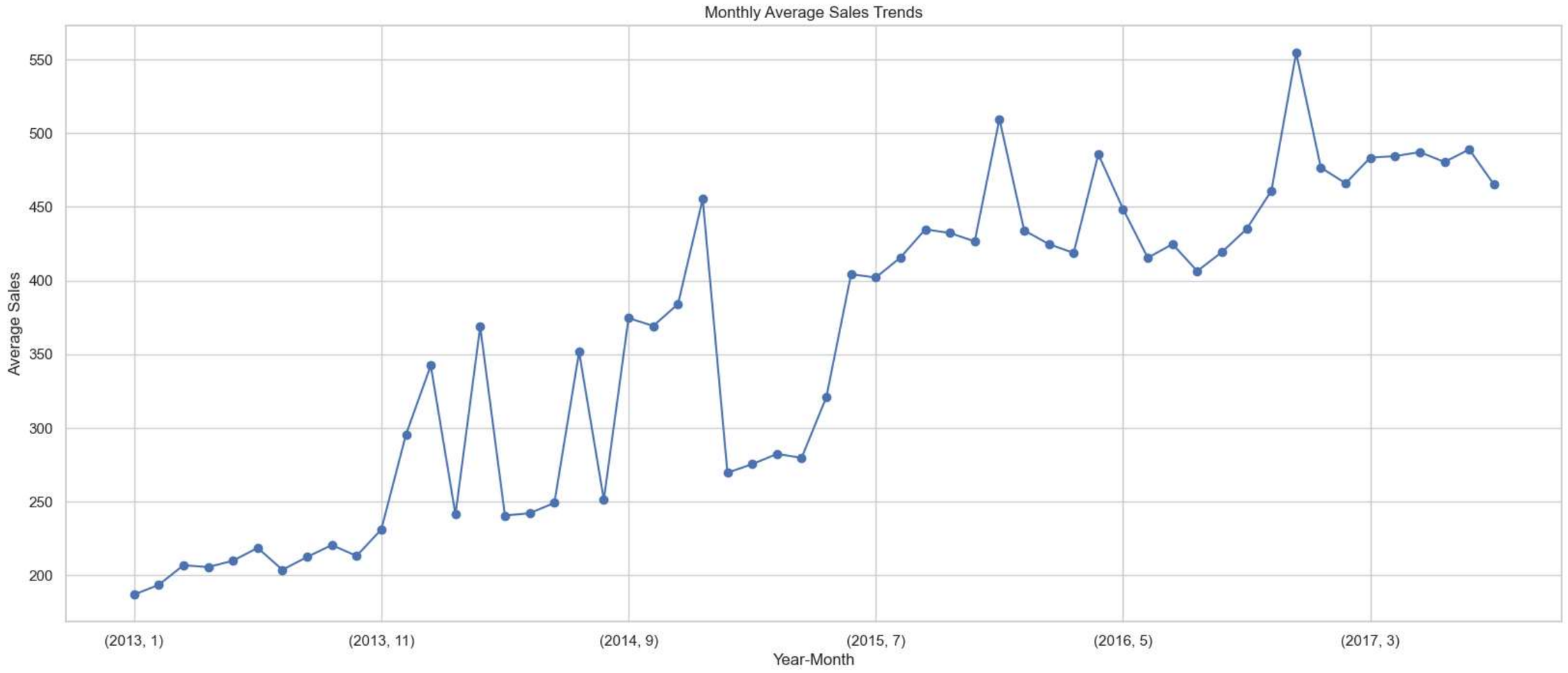
In [234…

```
weekly_sales = data['sales'].resample('W').sum()

# Decompose the time series
result = seasonal_decompose(weekly_sales, model='additive')

plt.rcParams['figure.figsize'] = [12, 8]
result.plot()
plt.show()
```



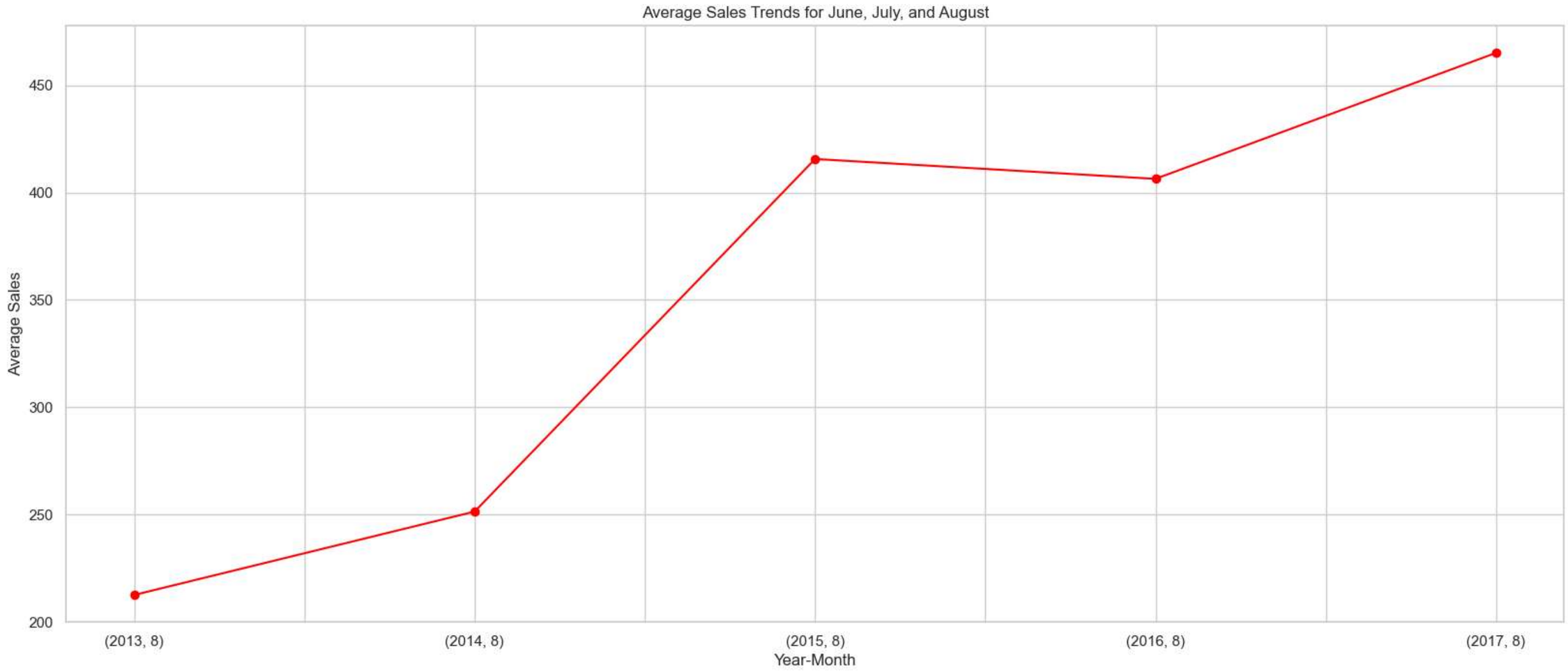
In [235…

```
# Monthly Sales Trends
monthly_sales_trends = data.groupby(['year', 'month'])['sales'].mean()
monthly_sales_trends.plot(figsize=(20, 8), marker='o')
plt.title('Monthly Average Sales Trends')
plt.xlabel('Year-Month')
plt.ylabel('Average Sales')

plt.show()
```



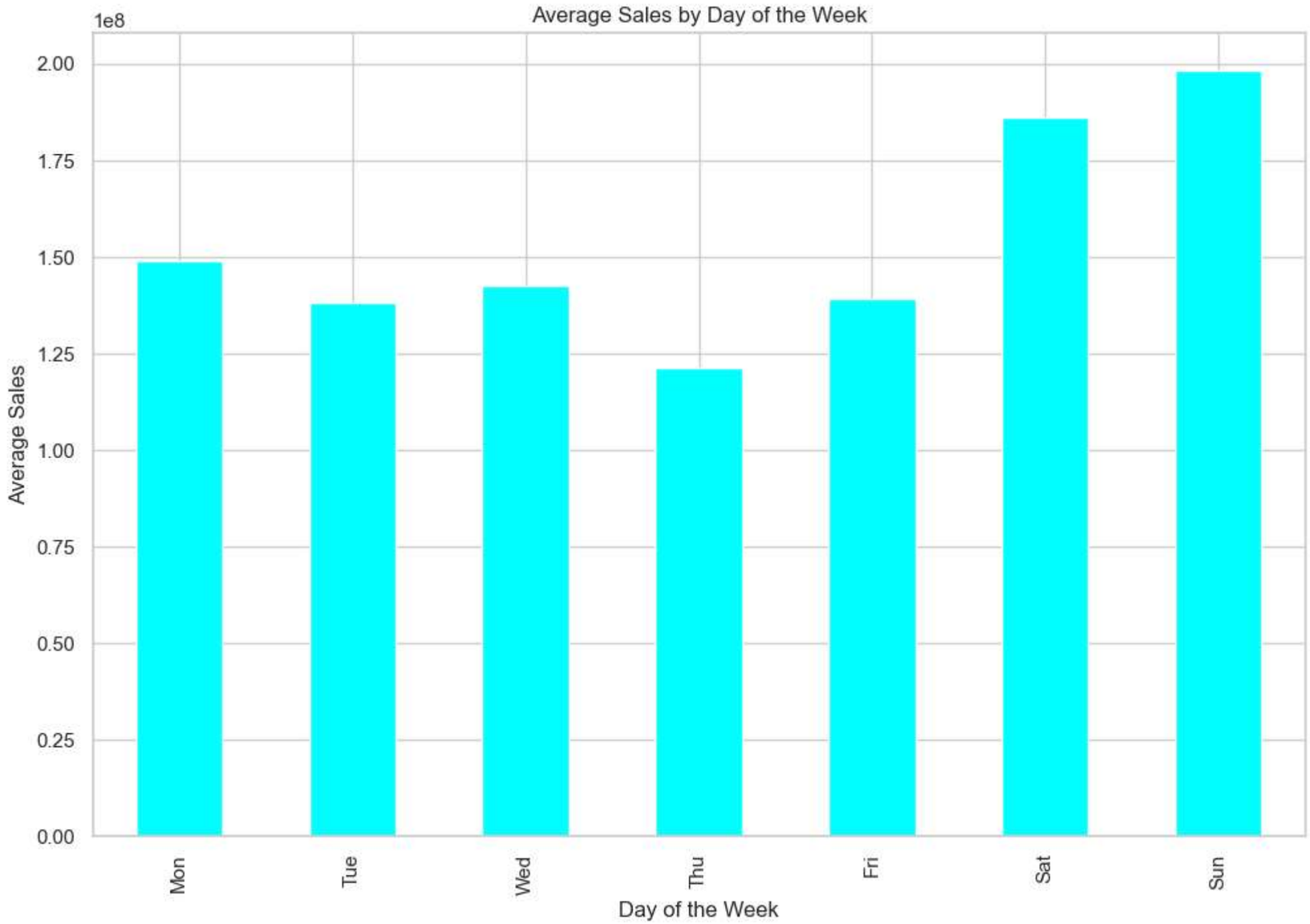
In [236…

```
# Filter data for June, July, and August
summer_months_data = data[data['month'].isin([8])]

# Monthly Sales Trends for June, July, and August
summer_monthly_sales_trends = summer_months_data.groupby(['year', 'month'])['sales'].mean()
summer_monthly_sales_trends.plot(figsize=(20, 8), marker='o', color='red')
plt.title('Average Sales Trends for June, July, and August')
plt.xlabel('Year-Month')
plt.ylabel('Average Sales')
plt.show()
```



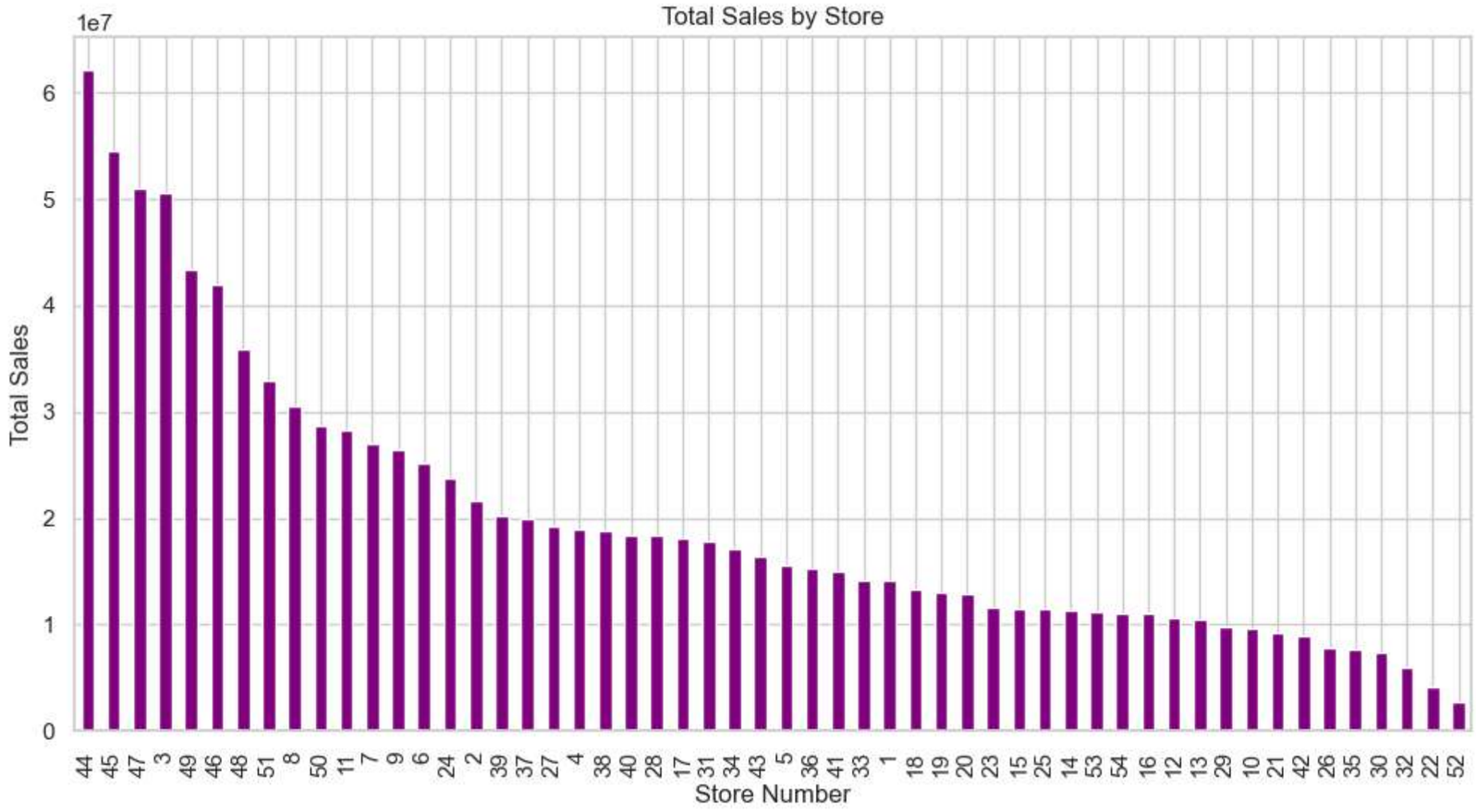
In [237…

```
# Day of the Week Sales Analysis
day_of_week_sales = data.groupby('day_of_week')['sales'].sum()
day_of_week_sales.index = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
day_of_week_sales.plot(kind='bar', color='cyan')
plt.title('Average Sales by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Average Sales')
plt.show()
```



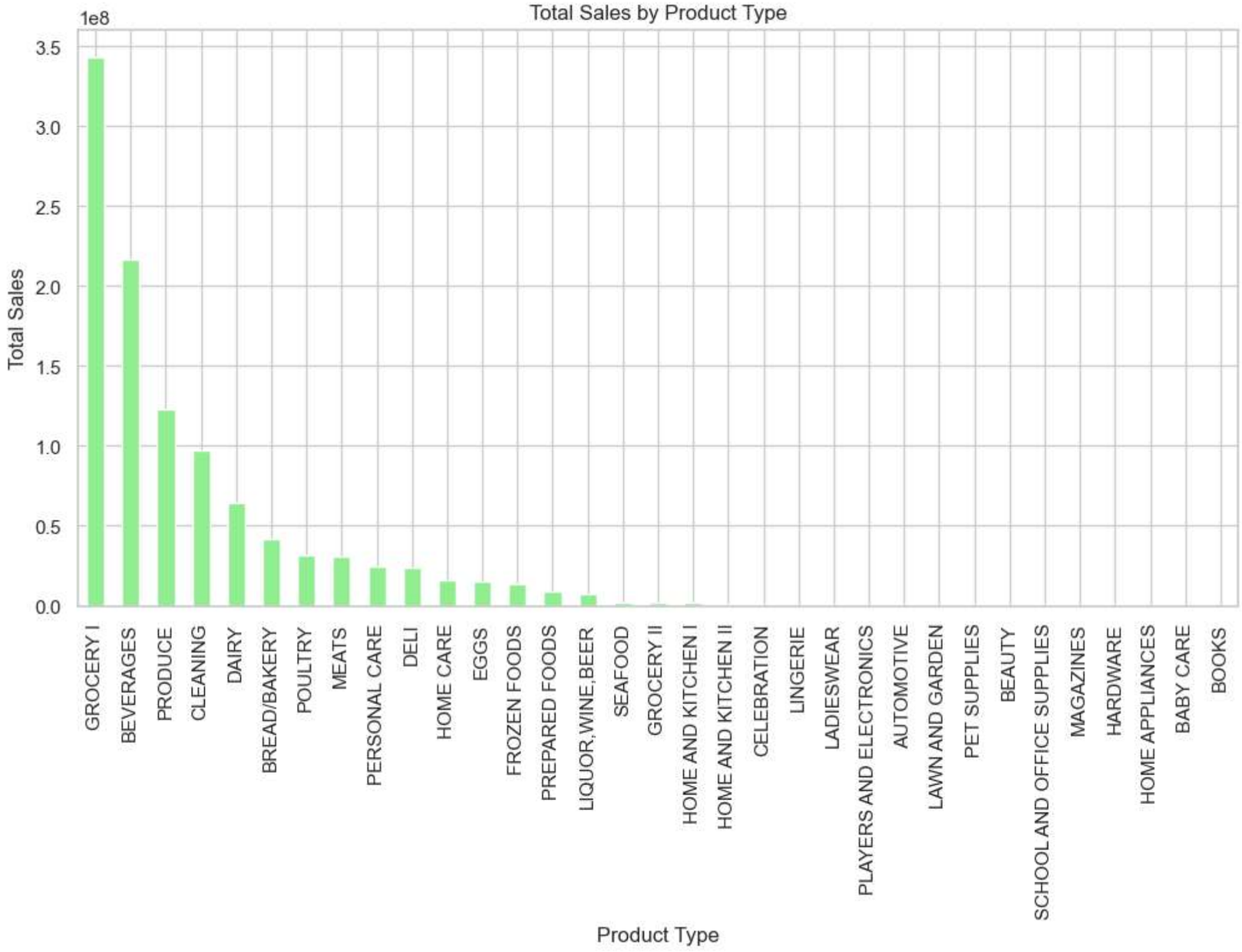
In [267…

```
# Store-wise Sales Analysis
store_wise_sales = data.groupby('store_nbr')['sales'].sum().sort_values(ascending=False)
store_wise_sales.plot(kind='bar', figsize=(12, 6), color='purple')
plt.title('Total Sales by Store')
plt.xlabel('Store Number')
plt.ylabel('Total Sales')
plt.show()
```



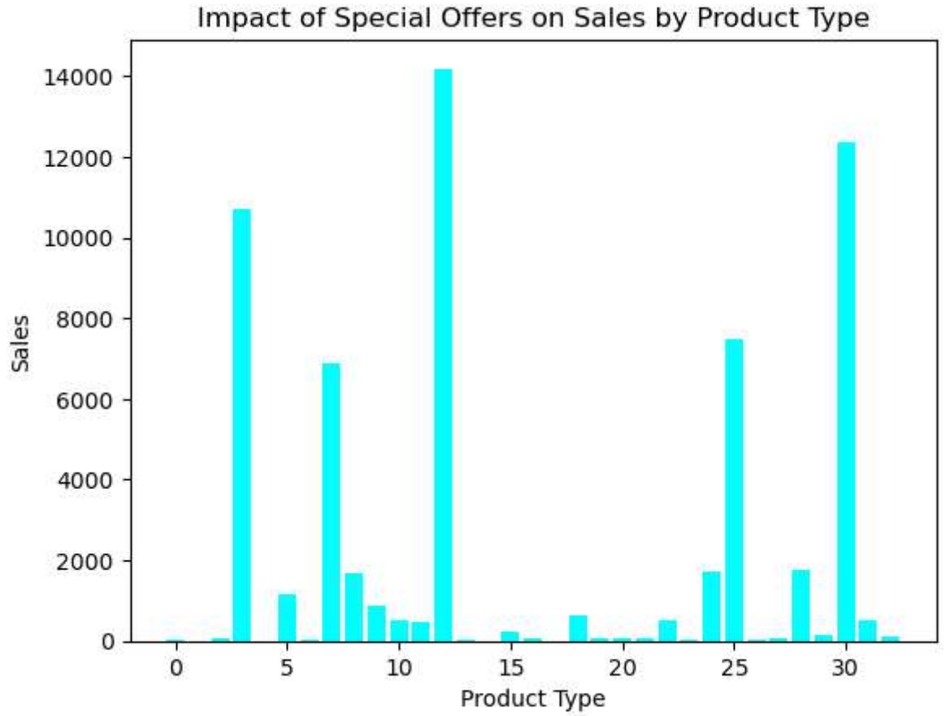
In [239…

```
# Product-wise Sales Analysis
product_wise_sales = data.groupby('product_type')['sales'].sum().sort_values(ascending=False)
product_wise_sales.plot(kind='bar', figsize=(12, 6), color='lightgreen')
plt.title('Total Sales by Product Type')
plt.xlabel('Product Type')
plt.ylabel('Total Sales')
plt.show()
```



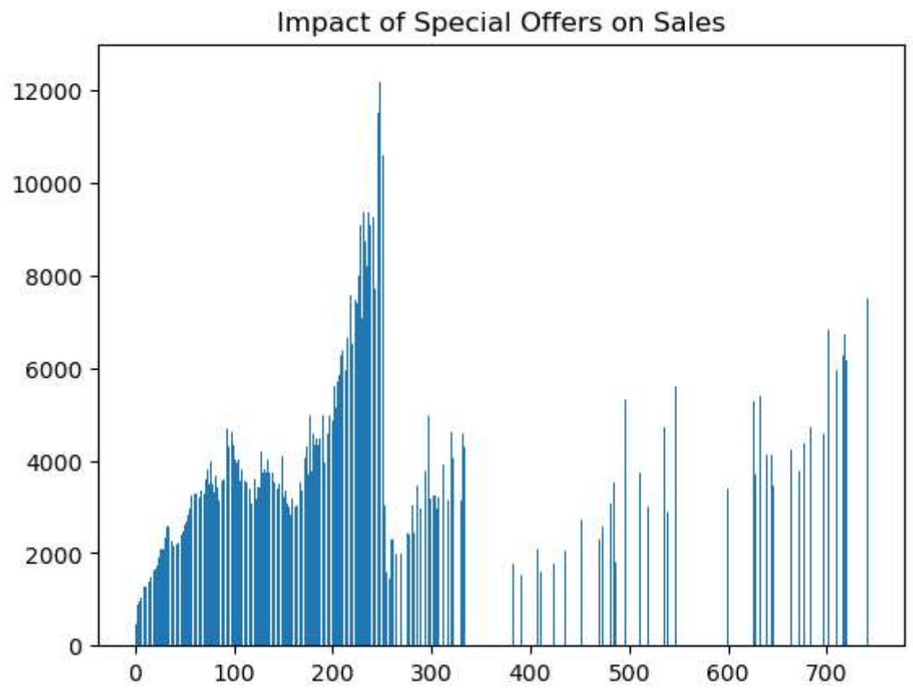
In [278…

```
offer_impact_by_product = data.groupby(['product_type', 'special_offer'])['sales'].mean().reset_index()
plt.style.use('default')
plt.bar(offer_impact_by_product['product_type'], offer_impact_by_product['sales'], color='cyan')  # Change 'cyan' to your desired color
plt.title('Impact of Special Offers on Sales by Product Type')
plt.xlabel('Product Type')
plt.ylabel('Sales')

plt.show()
```

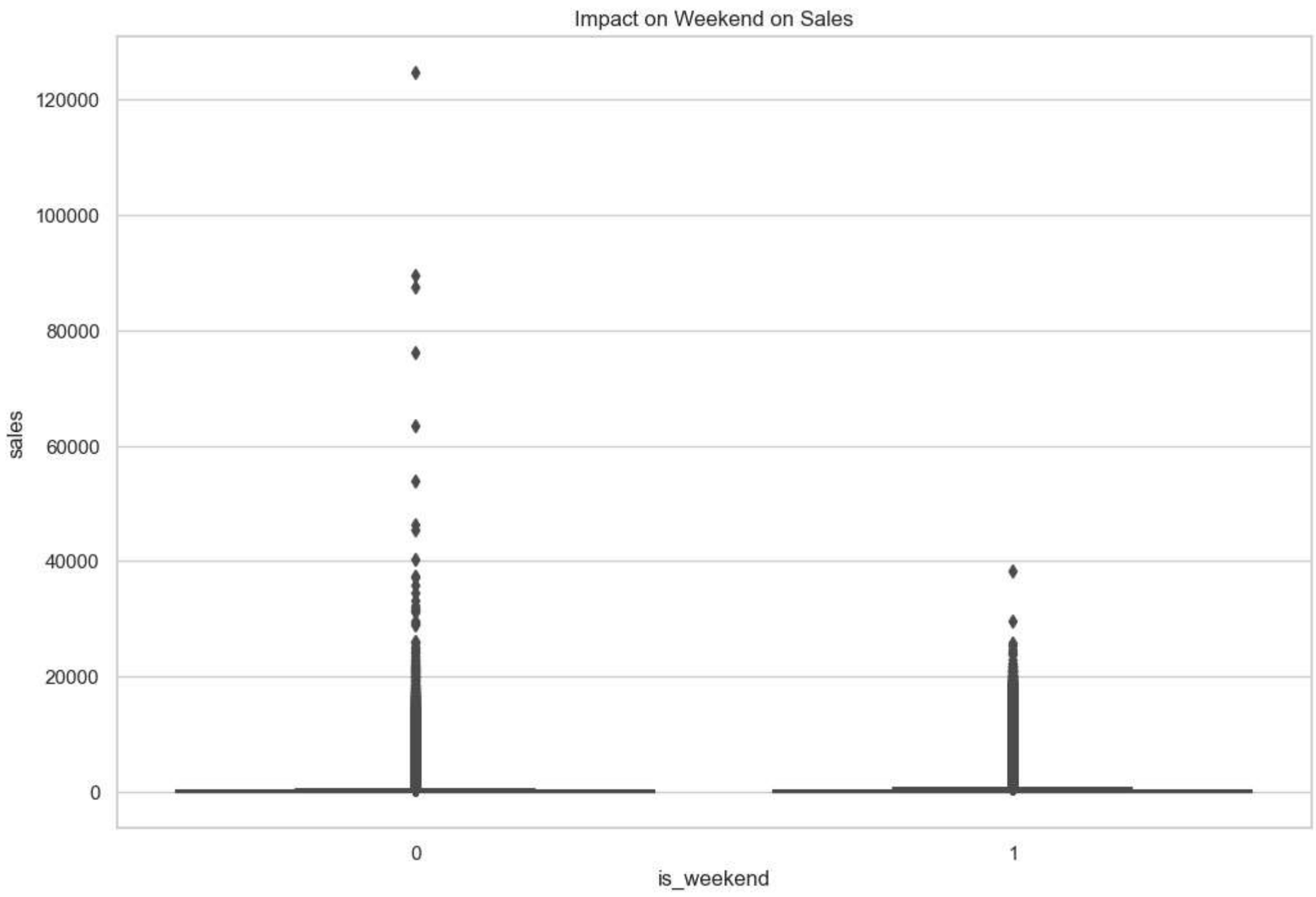


In [277…
```
offer_impact = data.groupby('special_offer')['sales'].mean().reset_index()
plt.style.use('default')
plt.bar(offer_impact['special_offer'],offer_impact['sales'])
plt.title('Impact of Special Offers on Sales')
```

Out[277]:
```
Text(0.5, 1.0, 'Impact of Special Offers on Sales')
```



In [242…
```
#impact of weekend on sales
sns.boxplot(x=data['is_weekend'],y=data['sales'])
plt.title('Impact on Weekend on Sales')
plt.show()
```



Feature Engineering

Feature engineering or feature extraction or feature discovery is the process of extracting features from raw data.

- Temporal features like year, month and day_of_week are created.
- lagged features of 1,7,15,30,60 days
- Moving average of sales (e.g., over the past 16 days)

In [243…
```
data
```

Out[243]:

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	is_weekend
2013-01-01	0	2013-01-01	1	AUTOMOTIVE	0.000	0	2013	1	2	0
2013-01-01	1	2013-01-01	1	BABY CARE	0.000	0	2013	1	2	0
2013-01-01	2	2013-01-01	1	BEAUTY	0.000	0	2013	1	2	0
2013-01-01	3	2013-01-01	1	BEVERAGES	0.000	0	2013	1	2	0
2013-01-01	4	2013-01-01	1	BOOKS	0.000	0	2013	1	2	0
...	...	...	...	...	...	...	...	...	...	...
2017-08-15	3000883	2017-08-15	9	POULTRY	438.133	0	2017	8	2	0
2017-08-15	3000884	2017-08-15	9	PREPARED FOODS	154.553	1	2017	8	2	0
2017-08-15	3000885	2017-08-15	9	PRODUCE	2419.729	148	2017	8	2	0
2017-08-15	3000886	2017-08-15	9	SCHOOL AND OFFICE SUPPLIES	121.000	8	2017	8	2	0
2017-08-15	3000887	2017-08-15	9	SEAFOOD	16.000	0	2017	8	2	0

3000888 rows × 10 columns

In [256…

```
data = data.sort_values(by=['store_nbr', 'product_type', 'date'])
data
```

Out[256]:

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	is_weekend	sales_lag_1	sales_lag_7	sales_lag_15	sales_lag_30	sales_lag_60	rolling_avg_sales
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	0	1	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	0	2	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	0	3	0	2.0	0.0	0.0	0.0	0.0	0.0000
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	0	4	0	3.0	0.0	0.0	0.0	0.0	0.0000
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	0	5	1	3.0	0.0	0.0	0.0	0.0	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-08-11	2993627	2017-08-11	54	32	0.0	0	2017	7	4	0	2.0	0.0	2.0	1.0	5.0	3.1250
2017-08-12	2995409	2017-08-12	54	32	1.0	1	2017	7	5	1	0.0	3.0	4.0	0.0	0.0	3.0625
2017-08-13	2997191	2017-08-13	54	32	2.0	0	2017	7	6	1	1.0	0.0	4.0	0.0	3.0	2.9375
2017-08-14	2998973	2017-08-14	54	32	0.0	0	2017	7	0	0	2.0	0.0	4.0	2.0	0.0	2.6875
2017-08-15	3000755	2017-08-15	54	32	3.0	0	2017	7	1	0	0.0	12.0	4.0	5.0	2.0	2.6250

3000888 rows × 16 columns

Feature Creation

In [245…

```
# Creating lagged sales features
lag_values = [1,7,15,30,60]  # Adjust as needed
```

```
for lag in lag_values:
    data[f'sales_lag_{lag}'] = data.groupby(['store_nbr', 'product_type'])['sales'].shift(lag)
```

In [246…
```
# Moving average of sales (e.g., over the past 16 days)
window_size = 16
rolling_avg_sales = data.groupby(['store_nbr', 'product_type'])['sales'].rolling(window=window_size).mean()
data['rolling_avg_sales'] = rolling_avg_sales.reset_index(level=[0,1], drop=True)
```

In [257…
```
data = data.fillna(0)
```

Feature Encoding

In [248…
```
# Applying Label Encoder
labelencoder = LabelEncoder()

data['month'] = labelencoder.fit_transform(data['month'])
data['day_of_week'] = labelencoder.fit_transform(data['day_of_week'])
data['year'] = data['year'].astype(int)

data['product_type'] = labelencoder.fit_transform(data['product_type'])
data
```

Out[248]:

index_date	id	date	store_nbr	product_type	sales	special_offer	year	month	day_of_week	is_weekend	sales_lag_1	sales_lag_7	sales_lag_15	sales_lag_30	sales_lag_60	rolling_avg_sales
2013-01-01	0	2013-01-01	1	0	0.0	0	2013	0	1	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-02	1782	2013-01-02	1	0	2.0	0	2013	0	2	0	0.0	0.0	0.0	0.0	0.0	0.0000
2013-01-03	3564	2013-01-03	1	0	3.0	0	2013	0	3	0	2.0	0.0	0.0	0.0	0.0	0.0000
2013-01-04	5346	2013-01-04	1	0	3.0	0	2013	0	4	0	3.0	0.0	0.0	0.0	0.0	0.0000
2013-01-05	7128	2013-01-05	1	0	5.0	0	2013	0	5	1	3.0	0.0	0.0	0.0	0.0	0.0000
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2017-08-11	2993627	2017-08-11	54	32	0.0	0	2017	7	4	0	2.0	0.0	2.0	1.0	5.0	3.1250
2017-08-12	2995409	2017-08-12	54	32	1.0	1	2017	7	5	1	0.0	3.0	4.0	0.0	0.0	3.0625
2017-08-13	2997191	2017-08-13	54	32	2.0	0	2017	7	6	1	1.0	0.0	4.0	0.0	3.0	2.9375
2017-08-14	2998973	2017-08-14	54	32	0.0	0	2017	7	0	0	2.0	0.0	4.0	2.0	0.0	2.6875
2017-08-15	3000755	2017-08-15	54	32	3.0	0	2017	7	1	0	0.0	12.0	4.0	5.0	2.0	2.6250

3000888 rows × 16 columns

Note: The choice of a 16-day rolling average of sales as a feature in predictive modeling, especially for retail sales forecasting, is based on several practical reasons:

Weekly Sales Patterns: Retail sales often exhibit weekly patterns. For instance, certain products might sell more during weekends or weekdays. A 7-day rolling average helps in capturing these weekly trends.

Smoothing Out Daily Fluctuations: Daily sales data can be quite volatile due to various factors such as promotions, weather, or random variations. A rolling average smoothens these fluctuations, providing a more stable and representative view of the sales trend.

Seasonal Adjustments: Weekly averages can adjust for minor seasonal variations, like increased shopping activity on specific days of the week, which might not be apparent in daily data.

Data Consistency: In a dataset spanning multiple years, a 16-day average ensures consistency in the data, making it easier to compare periods across different years.

Predictive Relevance: For short-term forecasting, such as predicting sales in the next few weeks, the recent sales trend (like the past week's average) is often more relevant than older data.

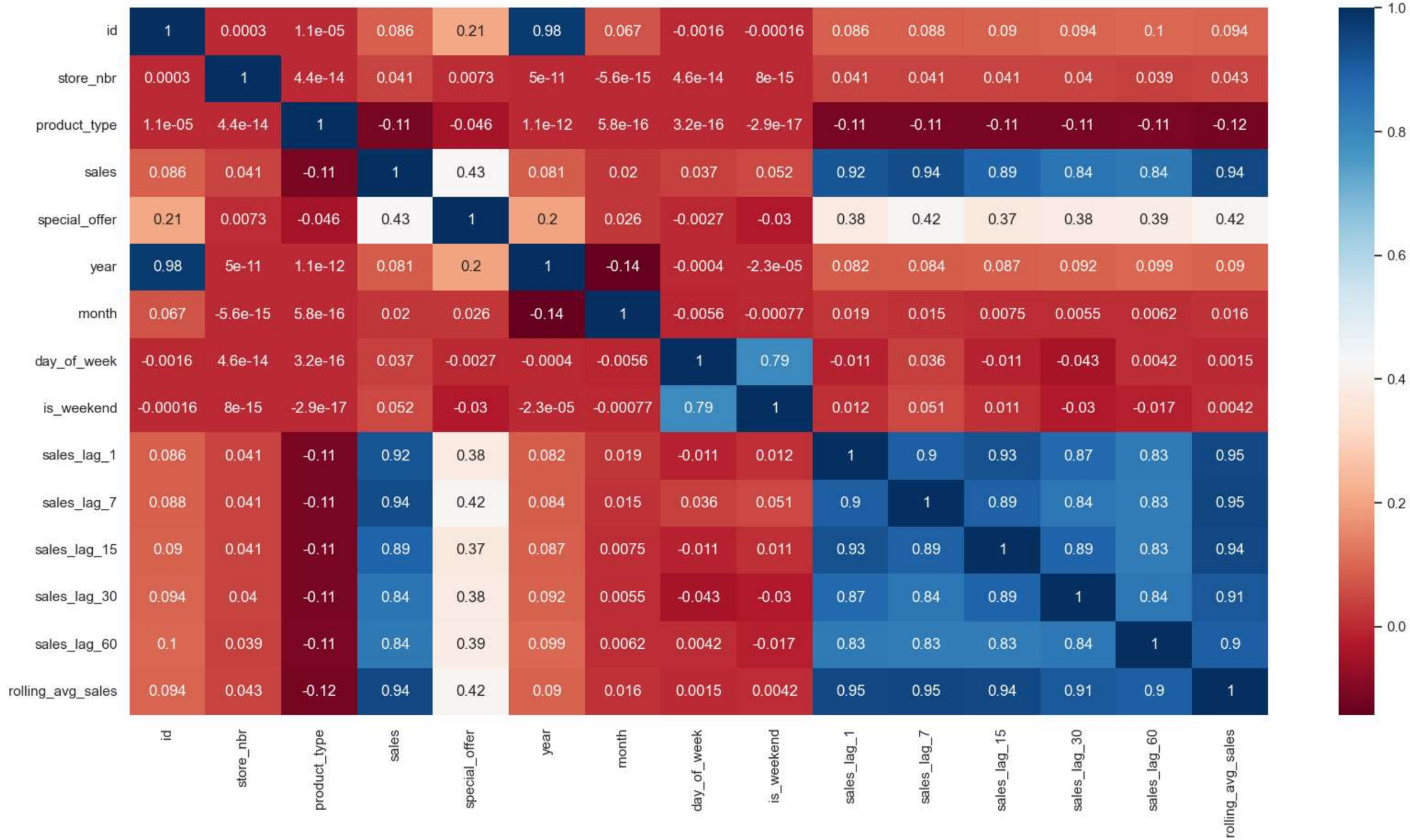
Alignment with Promotional and Marketing Activities: Retail promotions are often run on a weekly basis. A 7-day average can effectively capture the impact of such activities.

Feature Selection

Correlation HeatMap

In [258…

```
plt.figure(figsize=(20,10))
data_corr = data.corr()
sns.heatmap(data_corr,annot=True,cmap='RdBu')
plt.show()
```



SelectKBest

Feature selection is the process of selecting a subset of relevant features for use in model construction.

Why SelectKBest? SelectKBest is a feature selection technique used within the scikit-learn library in Python. It is part of the univariate feature selection methods which work by selecting the best features based on univariate statistical tests (These tests are intended to determine the strength of the relationship between each individual feature and the target).

For regression problems: f_regression, which measures linear dependency between two random variables. For classification problems: chi2, f_classif, or mutual_info_classif.

In [250…

```
X = data.drop(data[['sales','date']], axis=1)
y = data['sales']

k =7
bestfeatures = SelectKBest(score_func=f_regression, k=k)
fit = bestfeatures.fit(X, y)

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Feature', 'Score']
print("Best Feature Names with Scores as per the SelectKBest model:")
print(featureScores.nlargest(k, 'Score'))
print()
selected_features = featureScores.nlargest(k, 'Score')['Feature']
print("Best Feature Names as per the SelectKBest model:")
print(selected_features)

X_new = X[selected_features]
train_data_new_with_sales = pd.concat([X_new, y], axis=1)
```

```
Best Feature Names with Scores as per the SelectKBest model:
             Feature         Score
13  rolling_avg_sales  2.413343e+07
9         sales_lag_7  2.088978e+07
8         sales_lag_1  1.631247e+07
10       sales_lag_15  1.118377e+07
11       sales_lag_30  7.475799e+06
12       sales_lag_60  6.922998e+06
3       special_offer  6.727009e+05
Best Feature Names as per the SelectKBest model:
13    rolling_avg_sales
9           sales_lag_7
8           sales_lag_1
10         sales_lag_15
11         sales_lag_30
12         sales_lag_60
3         special_offer
Name: Feature, dtype: object
```

In [251…

```
best_features_full_data = train_data_new_with_sales[['rolling_avg_sales','sales_lag_7','sales_lag_15','special_offer','sales']]
```

In [259…

```
best_features_full_data
```

Out[259]:

index_date	rolling_avg_sales	sales_lag_7	sales_lag_15	special_offer	sales
2013-01-01	0.0000	0.0	0.0	0	0.0
2013-01-02	0.0000	0.0	0.0	0	2.0
2013-01-03	0.0000	0.0	0.0	0	3.0
2013-01-04	0.0000	0.0	0.0	0	3.0
2013-01-05	0.0000	0.0	0.0	0	5.0
...	...	...	...	...	...
2017-08-11	3.1250	0.0	2.0	0	0.0
2017-08-12	3.0625	3.0	4.0	1	1.0
2017-08-13	2.9375	0.0	4.0	0	2.0
2017-08-14	2.6875	0.0	4.0	0	0.0
2017-08-15	2.6250	12.0	4.0	0	3.0

3000888 rows × 5 columns

Feature Scaling

In [24]:
```
#Scaling - seen data - BoxCox Transformer
from scipy import stats

# Replace 0 with a small positive value if zeros are present in the data
constant = 1

# Apply Box-Cox transformations
best_features_full_data['rolling_avg_sales'], _ = stats.boxcox(best_features_full_data['rolling_avg_sales'] + constant)
best_features_full_data['sales_lag_7'], _ = stats.boxcox(best_features_full_data['sales_lag_7'] + constant)
best_features_full_data['sales_lag_15'], _ = stats.boxcox(best_features_full_data['sales_lag_15'] + constant)
best_features_full_data['special_offer'], _ = stats.boxcox(best_features_full_data['special_offer'] + constant)
```

In [25]:
```
#scaler = preprocessing.StandardScaler()
#best_features_full_data[['special_offer','rolling_avg_sales','sales_lag_7','sales_lag_15']]=scaler.fit_transform(best_features_full_data[['special_offer','rolling_avg_sales','sales_lag_7','sale
```

In [26]:
```
best_features_full_data
```

Out[26]:

index_date	rolling_avg_sales	sales_lag_7	sales_lag_15	special_offer	sales
2013-01-01	0.000000	0.000000	0.000000	0.000000	0.0
2013-01-02	0.000000	0.000000	0.000000	0.000000	2.0
2013-01-03	0.000000	0.000000	0.000000	0.000000	3.0
2013-01-04	0.000000	0.000000	0.000000	0.000000	3.0
2013-01-05	0.000000	0.000000	0.000000	0.000000	5.0
...	...	...	...	...	...
2017-08-11	1.320205	0.000000	1.032891	0.000000	0.0
2017-08-12	1.306966	1.285073	1.471022	0.347068	1.0
2017-08-13	1.279802	0.000000	1.471022	0.000000	2.0
2017-08-14	1.222505	0.000000	1.471022	0.000000	0.0
2017-08-15	1.207511	2.232678	1.471022	0.000000	3.0

3000888 rows × 5 columns

In [27]:
```
#Spliting the data

seen_data = best_features_full_data["2015-01-01":"2017-07-30"]

unseen_data = best_features_full_data["2017-07-31":]

print("Seen Data")
print("************************")
print(seen_data)
print()
print("Unseen Data")
print("************************")
print(unseen_data)
```

```
Seen Data
************************
            rolling_avg_sales  sales_lag_7  sales_lag_15  special_offer  sales
index_date
2015-01-01           1.192230     1.750430      1.621349       0.000000    0.0
2015-01-02           1.144551     0.667198      1.852312       0.000000    2.0
2015-01-03           1.128007     1.034379      0.000000       0.000000    6.0
2015-01-04           1.192230     0.000000      1.282754       0.000000    4.0
2015-01-05           1.222505     0.667198      0.666588       0.000000    5.0
...                       ...          ...           ...            ...    ...
2017-07-26           1.207511     0.000000      1.282754       0.347068    3.0
2017-07-27           1.192230     1.285073      0.666588       0.000000    2.0
2017-07-28           1.237221     0.000000      0.000000       0.000000    4.0
2017-07-29           1.293501     1.285073      0.000000       0.400908    4.0
2017-07-30           1.346033     1.625109      1.032891       0.000000    4.0

[1675080 rows x 5 columns]

Unseen Data
************************
            rolling_avg_sales  sales_lag_7  sales_lag_15  special_offer  sales
index_date
2017-07-31           1.612267     1.474097      1.032891       0.000000    8.0
2017-08-01           1.638796     2.105780      1.032891       0.000000    5.0
2017-08-02           1.655980     1.034379      1.282754       0.000000    4.0
2017-08-03           1.655980     1.625109      1.852312       0.000000    3.0
2017-08-04           1.664428     1.857270      1.471022       0.000000    8.0
...                       ...          ...           ...            ...    ...
2017-08-11           1.320205     0.000000      1.032891       0.000000    0.0
2017-08-12           1.306966     1.285073      1.471022       0.347068    1.0
2017-08-13           1.279802     0.000000      1.471022       0.000000    2.0
2017-08-14           1.222505     0.000000      1.471022       0.000000    0.0
2017-08-15           1.207511     2.232678      1.471022       0.000000    3.0

[28512 rows x 5 columns]
```

Modelling

Train Test Split

In [28]:
```
X = seen_data.drop("sales", axis = 1)
y = seen_data["sales"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

1. Linear Regression

In [29]:
```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)
```

Out[29]:
```
▾ LinearRegression
LinearRegression()
```

In [30]:
```
print("Training Score -------",lr_model.score(X_train,y_train))
print("Testing Score -------",lr_model.score(X_test,y_test))
```

```
Training Score ------- 0.26105669757418926
Testing Score ------- 0.26543014471372195
```

In [31]:
```
y_pred_lr = lr_model.predict(X_test)
```

In [32]:
```
mse = mean_squared_error(y_test,y_pred_lr,squared=False)
mae = mean_absolute_error(y_test,y_pred_lr)
print("MSE -------",mse)
print("MAE -------",mae)
```

```
MSE ------- 1054.399595875114
MAE ------- 554.0148741083113
```

Predictions on Unseen Data

In [73]:
```
X_unseen_test_lr = unseen_data.drop(columns=['sales','store_nbr','product_type'])
predictions_lr = lr_model.predict(X_unseen_test_lr)

mae = mean_absolute_error(unseen_data['sales'], predictions_lr)
mse = mean_squared_error(unseen_data['sales'], predictions_lr)
rmse = mean_squared_error(unseen_data['sales'], predictions_lr,squared=False)


print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Absolute Error: 603.0182940018728
Mean Squared Error: 1128307.332077711
Root Mean Squared Error: 1062.2181188803506
```

Combining Actual and predicted sales

In [74]:
```
final_data_lr = pd.concat([unseen_data[['store_nbr','product_type','sales']].reset_index(),pd.DataFrame(predictions_lr)],axis=1)
```

In [75]:
```
final_data_lr.rename(columns={0:'predicted_sales'},inplace=True)
final_data_lr.rename(columns={'index_date':'date'},inplace=True)
```

In [76]:
```
final_data_lr
```

Out[76]:

	date	store_nbr	product_type	sales	predicted_sales
0	2017-07-31	1	0	8.0	24.192141
1	2017-08-01	1	0	5.0	47.402041
2	2017-08-02	1	0	4.0	18.665501
3	2017-08-03	1	0	3.0	18.458346
4	2017-08-04	1	0	8.0	36.695455
...	...	...	...	...	...
28507	2017-08-11	54	32	0.0	-93.735323
28508	2017-08-12	54	32	1.0	260.345104
28509	2017-08-13	54	32	2.0	-116.497913
28510	2017-08-14	54	32	0.0	-132.372230
28511	2017-08-15	54	32	3.0	-80.469841

28512 rows × 5 columns

In [ ]:
```
final_data_lr.to_csv("Unseen_pred_lr.csv")
```

Linear Regression: Actual vs Predicted

Total sales vs predicted (all time)

In [263…
```
final_lr = pd.read_csv("Unseen_pred_lr.csv")
```

In [264…
```
final_lr
```

Out[264]:

	Unnamed: 0	date	store_nbr	product_type	sales	predicted_sales
0	0	2017-07-31	1	0	8.0	24.192141
1	1	2017-08-01	1	0	5.0	47.402041
2	2	2017-08-02	1	0	4.0	18.665501
3	3	2017-08-03	1	0	3.0	18.458346
4	4	2017-08-04	1	0	8.0	36.695455
...	...	...	...	...	...	...
28507	28507	2017-08-11	54	32	0.0	-93.735323
28508	28508	2017-08-12	54	32	1.0	260.345104
28509	28509	2017-08-13	54	32	2.0	-116.497913
28510	28510	2017-08-14	54	32	0.0	-132.372230
28511	28511	2017-08-15	54	32	3.0	-80.469841

28512 rows × 6 columns

In [265…
```
summed_lr = pd.DataFrame(final_lr.groupby('date')['sales'].sum())
```

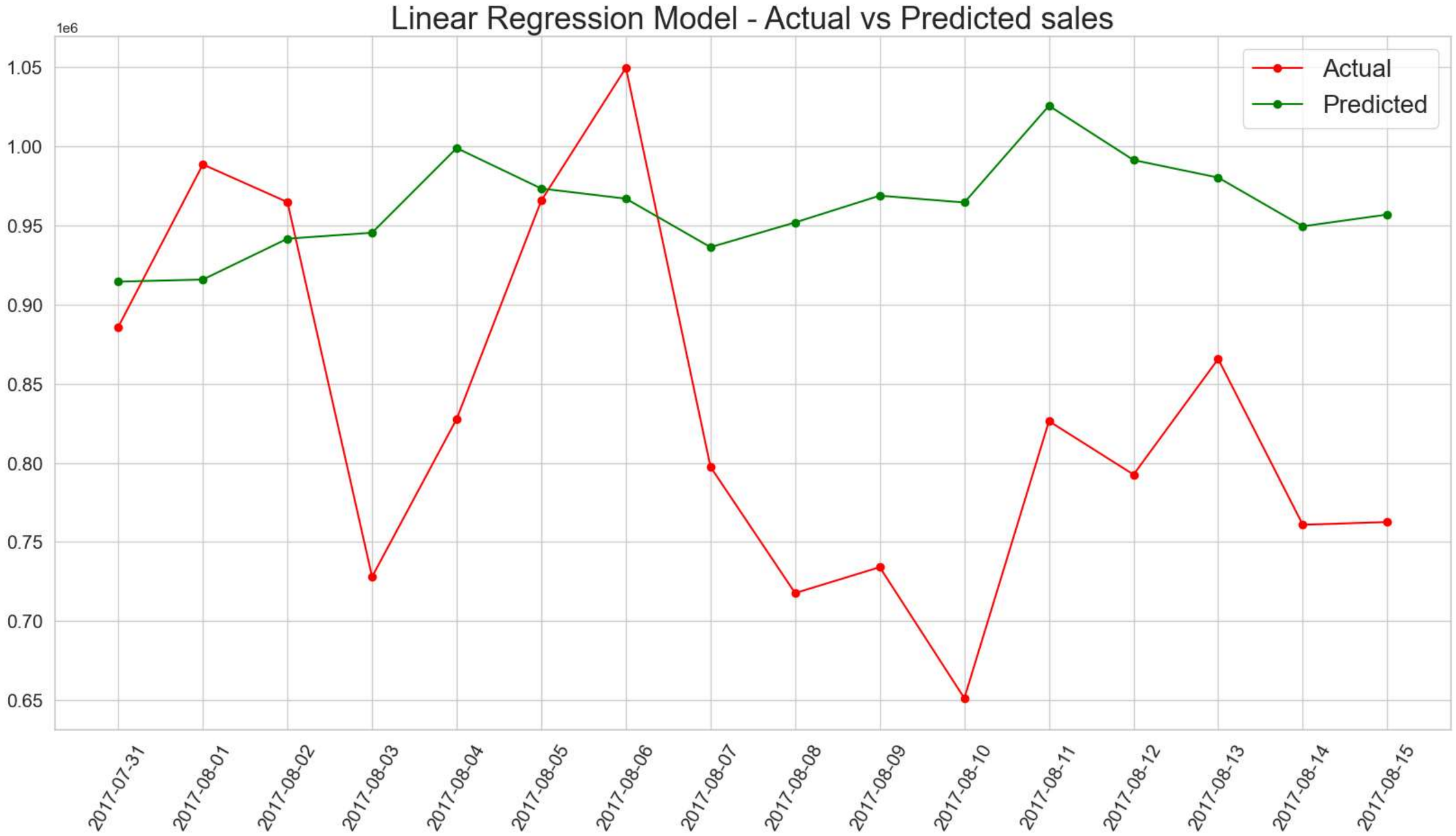
```
pred_summed_lr = pd.DataFrame(final_lr.groupby('date')['predicted_sales'].sum())
```

In [266…

```
plt.figure(figsize=(20,10))
plt.plot(summed_lr['sales'],marker='o',label='Actual',color='red')
plt.plot(pred_summed_lr['predicted_sales'],marker='o',label='Predicted',color='green')
plt.legend(fontsize=20)
plt.title("Linear Regression Model - Actual vs Predicted sales",fontsize=25)
plt.xticks(fontsize=15, rotation=60)
plt.yticks(fontsize=15)
```

Out[266]:

```
(array([ 600000.,  650000.,  700000.,  750000.,  800000.,  850000.,
         900000.,  950000., 1000000., 1050000., 1100000.]),
 [Text(0, 600000.0, '0.60'),
  Text(0, 650000.0, '0.65'),
  Text(0, 700000.0, '0.70'),
  Text(0, 750000.0, '0.75'),
  Text(0, 800000.0, '0.80'),
  Text(0, 850000.0, '0.85'),
  Text(0, 900000.0, '0.90'),
  Text(0, 950000.0, '0.95'),
  Text(0, 1000000.0, '1.00'),
  Text(0, 1050000.0, '1.05'),
  Text(0, 1100000.0, '1.10')])
```

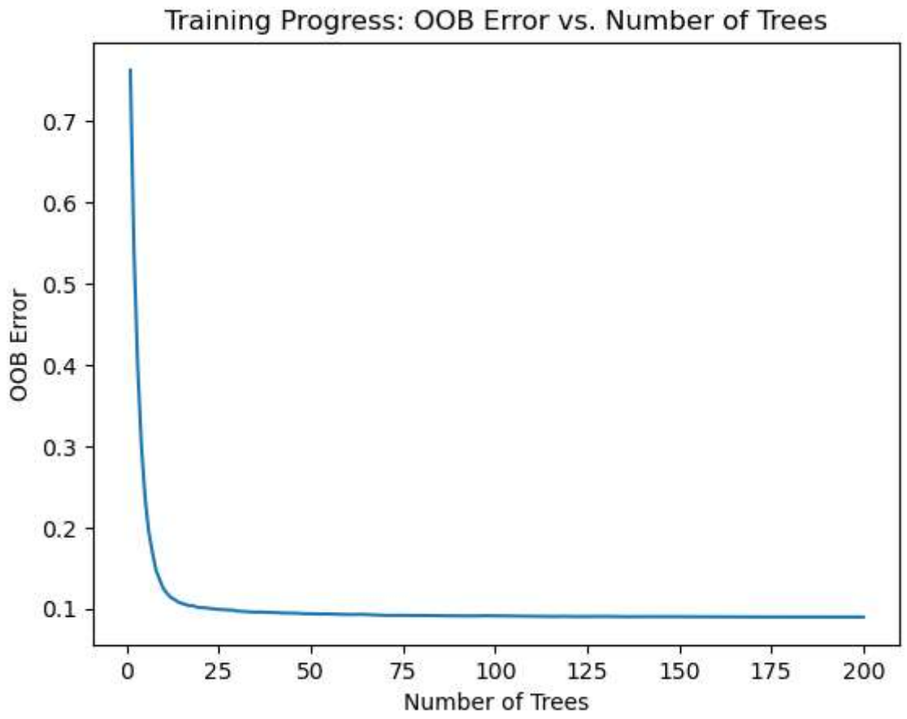


In [ ]:

2. Random Forest

In [37]:
```
n_estimators=200
rf_model = RandomForestRegressor(n_estimators=n_estimators,oob_score=True,warm_start=True)


oob_errors = []
for i in range(1, n_estimators+1):
    rf_model.n_estimators = i  # Set the number of trees
```

```
    rf_model.fit(X_train, y_train)
    oob_error = 1 - rf_model.oob_score_
    oob_errors.append(oob_error)

plt.plot(range(1, n_estimators+1), oob_errors)
plt.xlabel('Number of Trees')
plt.ylabel('OOB Error')
plt.title('Training Progress: OOB Error vs. Number of Trees')
plt.show()
```



In [285…
```
rf_model.oob_score_
```

Out[285]:
```
0.9102744398290942
```

In [66]:
```
importances = rf_model.feature_importances_
importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
feature_importance_df
```

Out[66]:

	Feature	Importance
0	rolling_avg_sales	0.871867
1	sales_lag_7	0.087130
2	sales_lag_15	0.025210
3	special_offer	0.015794

In [40]:
```
print("Training Score -------",rf_model.score(X_train,y_train))
print("Testing Score -------",rf_model.score(X_test,y_test))
```

```
Training Score ------- 0.9882797820001183
Testing Score ------- 0.9307743577055378
```

In [41]:
```
y_pred_rf = rf_model.predict(X_test)
```

In [50]:
```
export_pred_rf = pd.DataFrame(y_pred_rf)
export_pred_rf.to_csv("export_pred_rf_2015.csv")
```

In [42]:
```
mse = mean_squared_error(y_test,y_pred_rf,squared=False)
mae = mean_absolute_error(y_test,y_pred_rf)
print("MSE -------",mse)
print("MAE -------",mae)
```

```
MSE ------- 323.6847038306698
MAE ------- 79.35536247211368
```

Prediction on Unseen Data

In [44]:
```
X_unseen_test = unseen_data.drop(columns=['sales'])
predictions = rf_model.predict(X_unseen_test)

mae = mean_absolute_error(unseen_data['sales'], predictions)
mse = mean_squared_error(unseen_data['sales'], predictions)
rmse = mean_squared_error(unseen_data['sales'], predictions,squared=False)


print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Absolute Error: 87.09056187006601
Mean Squared Error: 90492.34544734615
Root Mean Squared Error: 300.81945656381026
```

In [ ]:
```

```

In [46]:
```
data_trimmed = data['2017-07-31':]
```

In [48]:
```
unseen_data['store_nbr']=data_trimmed['store_nbr']
unseen_data['product_type']=data_trimmed['product_type']
```

In [50]:
```
final_data = pd.concat([unseen_data[['store_nbr','product_type','sales']].reset_index(),pd.DataFrame(predictions)],axis=1)
```

In [55]:
```
final_data.rename(columns={0:'predicted_sales'},inplace=True)
final_data.rename(columns={'index_date':'date'},inplace=True)
```

In [56]:
```
final_data
```

Out[56]:

	date	store_nbr	product_type	sales	predicted_sales
0	2017-07-31	1	0	8.0	4.778179
1	2017-08-01	1	0	5.0	7.581328
2	2017-08-02	1	0	4.0	5.442089
3	2017-08-03	1	0	3.0	4.572344
4	2017-08-04	1	0	8.0	4.816661
...	...	...	...	...	...
28507	2017-08-11	54	32	0.0	3.443701
28508	2017-08-12	54	32	1.0	5.491697
28509	2017-08-13	54	32	2.0	2.754819
28510	2017-08-14	54	32	0.0	2.810651
28511	2017-08-15	54	32	3.0	2.076917

28512 rows × 5 columns

In [57]:
```
final_data.to_csv("UnseenData_predictions.csv")
```

3.XG Boost

In [33]:
```
xgb = XGBRegressor(enable_categorical=True)
xgb.fit(X_train,y_train)
```

Out[33]:

XGBRegressor

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

In [34]:
```
print("Training Score -------",xgb.score(X_train,y_train))
print("Testing Score -------",xgb.score(X_test,y_test))
```

```
Training Score ------- 0.9362428736510318
Testing Score ------- 0.9305830176762462
```

In [35]:
```
y_pred_xgb = xgb.predict(X_test)
```

In [36]:
```
mse = mean_squared_error(y_test,y_pred_xgb,squared=False)
mae = mean_absolute_error(y_test,y_pred_xgb)
print("MSE -------",mse)
print("MAE -------",mae)
```

```
MSE ------- 324.13172823767025
MAE ------- 78.30522111634008
```

Predictions on Unseen Data

In [68]:
```
X_unseen_test_xgb = unseen_data.drop(columns=['sales','store_nbr','product_type'])
predictions_xgb = xgb.predict(X_unseen_test)

mae = mean_absolute_error(unseen_data['sales'], predictions_xgb)
mse = mean_squared_error(unseen_data['sales'], predictions_xgb)
rmse = mean_squared_error(unseen_data['sales'], predictions_xgb,squared=False)


print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Absolute Error: 85.14151681991574
Mean Squared Error: 90784.97556264253
Root Mean Squared Error: 301.30545226172484
```

Combining Actual and predicted sales

In [69]:
```
final_data_xgb = pd.concat([unseen_data[['store_nbr','product_type','sales']].reset_index(),pd.DataFrame(predictions_xgb)],axis=1)
```

In [70]:
```
final_data_xgb.rename(columns={0:'predicted_sales'},inplace=True)
final_data_xgb.rename(columns={'index_date':'date'},inplace=True)
```

In [71]:
```
final_data_xgb.to_csv("Unseen_pred_xgb.csv")
```

XGBRegressor: Actual vs Predicted

Total sales vs predicted (all time)

In [260…
```
final_xgb = pd.read_csv("Unseen_pred_xgb.csv")
final_xgb
```

Out[260]:

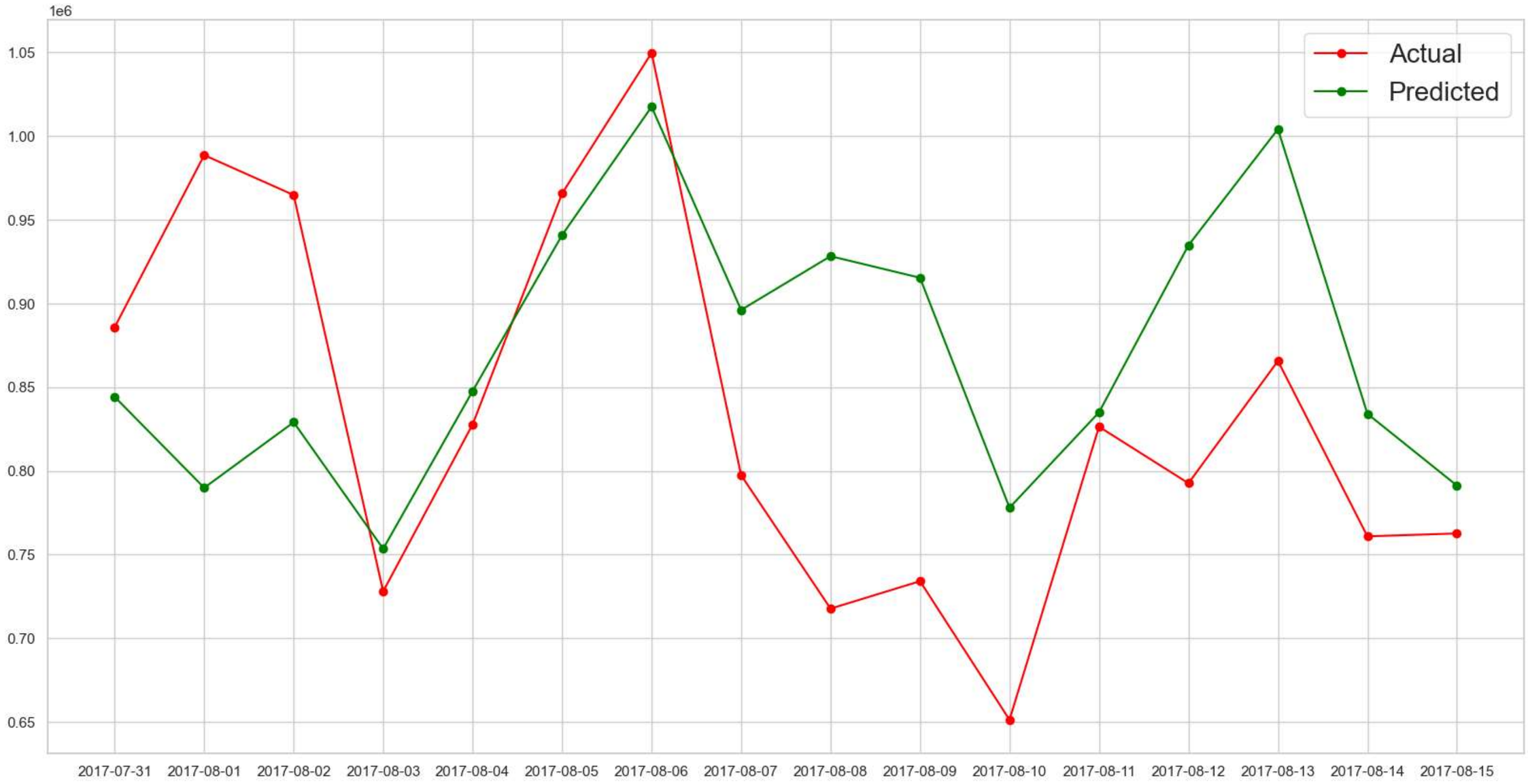
	Unnamed: 0	date	store_nbr	product_type	sales	predicted_sales
0	0	2017-07-31	1	0	8.0	4.764376
1	1	2017-08-01	1	0	5.0	4.764376
2	2	2017-08-02	1	0	4.0	4.764376
3	3	2017-08-03	1	0	3.0	4.764376
4	4	2017-08-04	1	0	8.0	4.764376
...	...	...	...	...	...	...
28507	28507	2017-08-11	54	32	0.0	0.281239
28508	28508	2017-08-12	54	32	1.0	4.729339
28509	28509	2017-08-13	54	32	2.0	0.281239
28510	28510	2017-08-14	54	32	0.0	0.281239
28511	28511	2017-08-15	54	32	3.0	2.823062

28512 rows × 6 columns

In [261…
```
summed_xgb = pd.DataFrame(final_xgb.groupby('date')['sales'].sum())

pred_summed_xgb = pd.DataFrame(final_xgb.groupby('date')['predicted_sales'].sum())
```

In [262…
```
plt.figure(figsize=(20,10))
plt.plot(summed_xgb['sales'],marker='o',label='Actual',color='red')
plt.plot(pred_summed_xgb['predicted_sales'],marker='o',label='Predicted',color='green')
plt.legend(fontsize=20)
```

Out[262]:
```
<matplotlib.legend.Legend at 0x19613567100>
```



In [ ]:

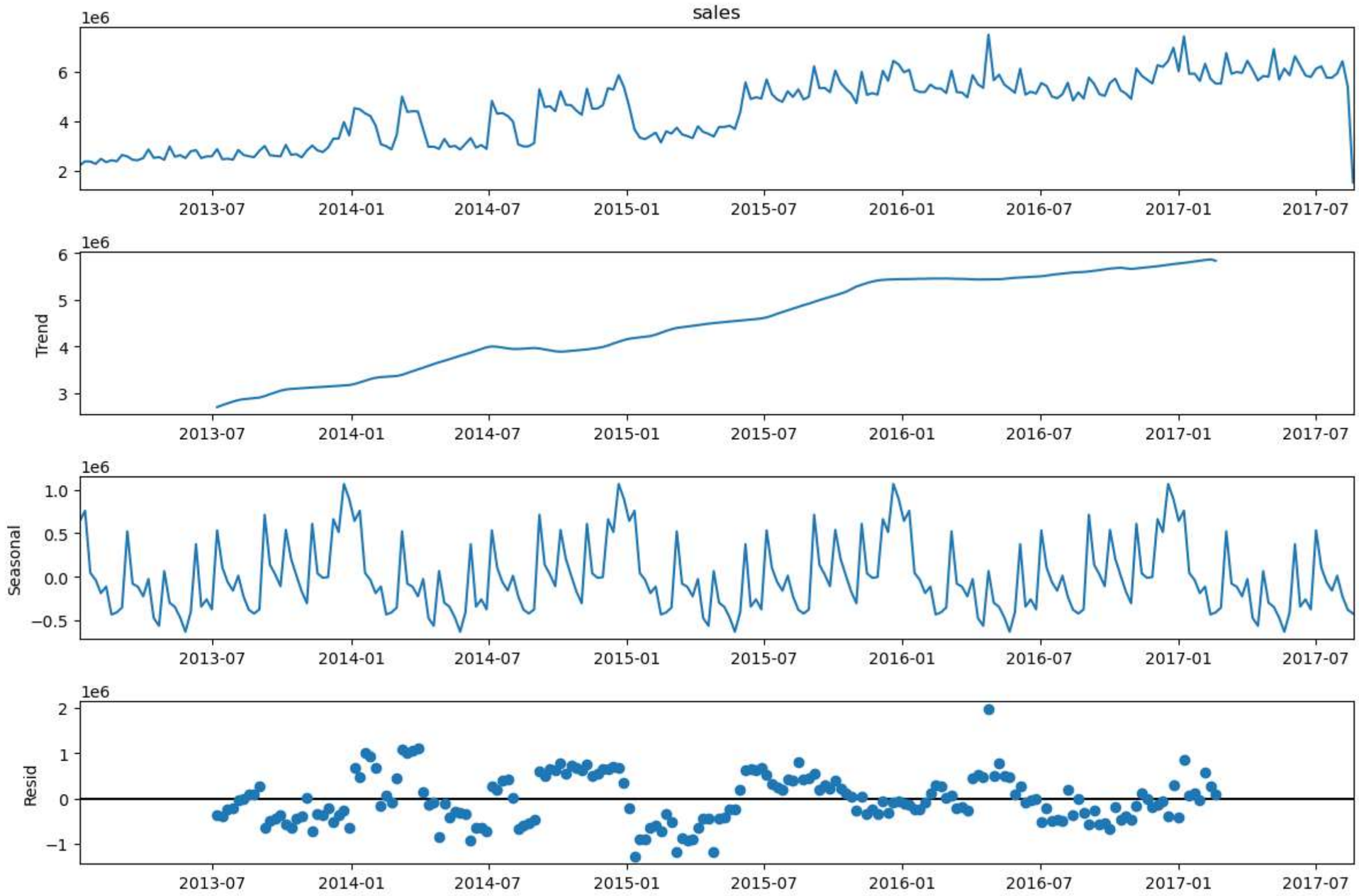
Time Series Analysis

Weekly Sales and Decomposing the time series

In [279…

```
df = pd.read_csv("Products_Information.csv")
df['date'] = pd.to_datetime(df['date'])  # Converting 'date' to datetime object

df.set_index('date', inplace=True)
df.sort_index(inplace=True)

# Weekly resampling
weekly_sales = df['sales'].resample('W').sum()
```

```
# Decompose the time series
result = seasonal_decompose(weekly_sales, model='additive')

plt.rcParams['figure.figsize'] = [12, 8]
result.plot()
plt.show()
```



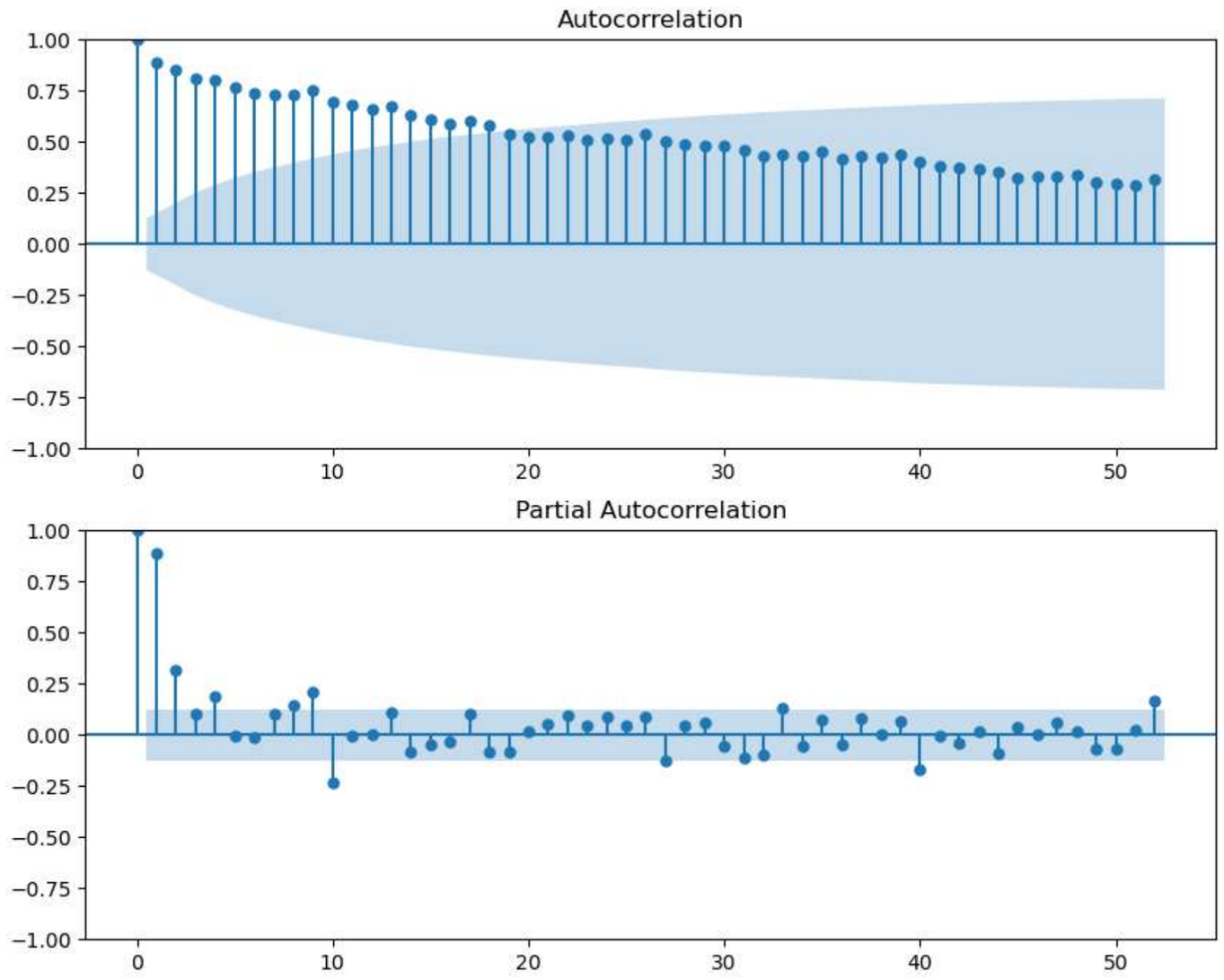
Calculating the Augmented Dickey-Fuller (ADF) and p-value

In [280…

```
result = adfuller(weekly_sales)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
fig, ax = plt.subplots(2,1, figsize=(10,8))
plot_acf(weekly_sales, ax=ax[0], lags=52)
plot_pacf(weekly_sales, ax=ax[1], lags=52)
plt.show()
```

```
ADF Statistic: -1.913783
p-value: 0.325605
```



ADF Statistic and p-value:

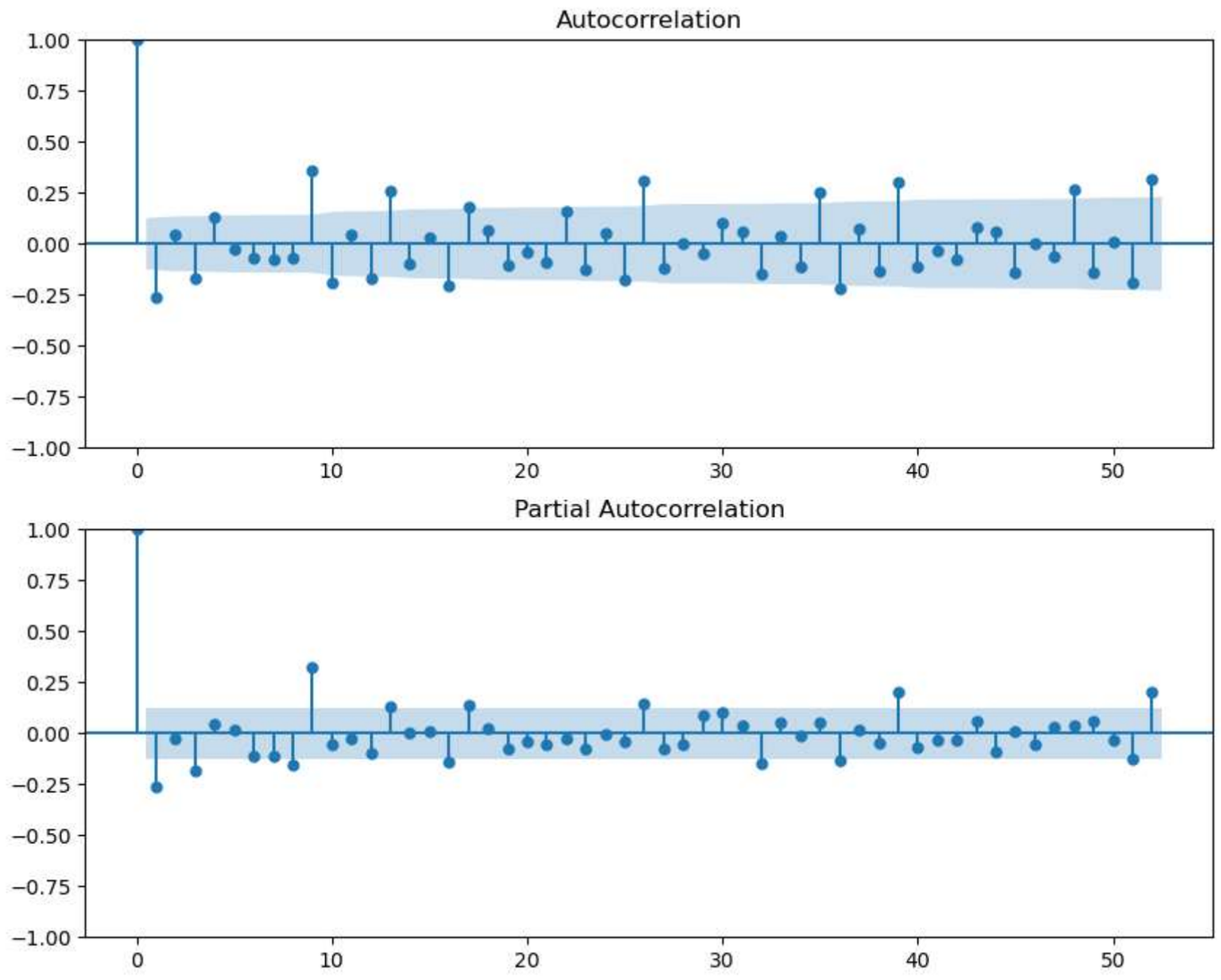
ADF Statistic: A more negative value suggests stronger stationarity. p-value: If the p-value is less than a significance level (commonly 0.05), the null hypothesis of the ADF test (which states the time series is non-stationary) can be rejected. In our case, with an ADF statistic of -1.913783 and a p-value of 0.325605, the test suggests that the time series is not stationary, as the p-value is higher than 0.05.

Selecting appropriate p, d and q for ARIMA based on Autocorrelation and Partial Autocorrelation Analysis

In [281…

```
# First differencing the series if d is 1
data_diff = weekly_sales.diff().dropna()
```

```
# Plot ACF and PACF
fig, ax = plt.subplots(2,1, figsize=(10,8))
plot_acf(data_diff, ax=ax[0], lags=52)  # ACF plot
plot_pacf(data_diff, ax=ax[1], lags=52)  # PACF plot
plt.show()
```



ACF Plot:

This indicates the correlation between the time series and its lagged values. For ARIMA, significant lags in the ACF plot suggest the q parameter, which corresponds to the MA (Moving Average) part.

PACF Plot:

This indicates the partial correlation between the time series and its lagged values, accounting for the values of the time series at all shorter lags. Significant lags here suggest the p parameter, which corresponds to the AR (Autoregressive) part.

Looking at our ACF plot, it seems that after the initial lag, the correlations quickly fall within the confidence interval, which suggests that the MA component (q) could be zero or one, given that the first lag is close to the significance limit.

In the PACF plot, there's a significant spike at lag 1, and then other spikes are within the confidence interval, which suggests an AR(1) process. Therefore, p is likely 1.

The d value corresponds to the level of differencing required to make the series stationary. Since the ADF test suggests non-stationarity (p-value > 0.05), you'll likely need to difference the data at least once, so d could be 1.

Note:

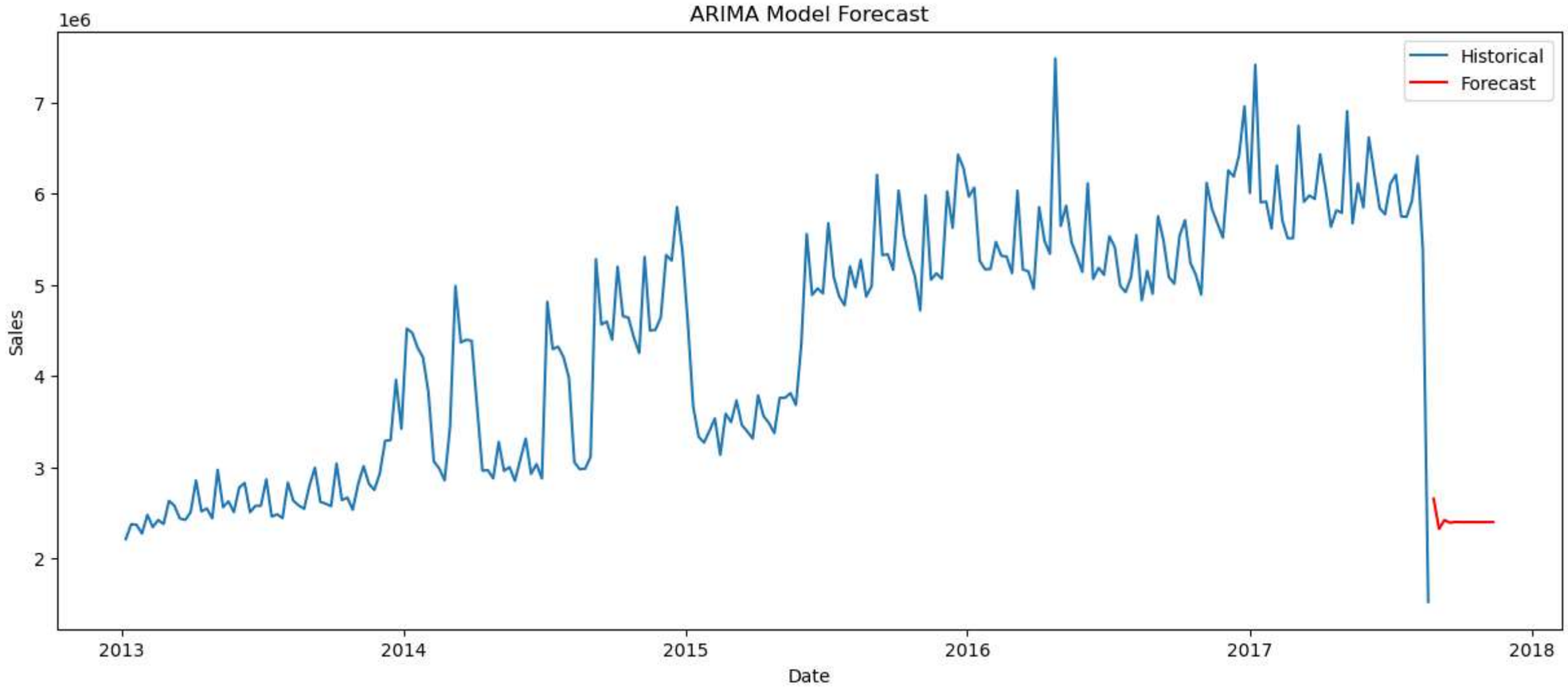
The choice of lags=52 in the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots for weekly sales data is likely based on the assumption that the data might have yearly seasonality. In short using lags=52 for weekly data is a strategic choice to explore potential yearly seasonal effects in your time series. If significant correlations are observed at lag 52 (or near it), it would suggest that incorporating seasonal components in your time series model could be beneficial.

ARIMA

ARIMA is an autoregressive integrated moving average (ARIMA) model is a statistical analysis model that leverages time series data to forecast future trends.

In [282…

```
p, d, q = 1, 1, 0

# Fit the ARIMA model
model = ARIMA(weekly_sales, order=(p, d, q))
model_fit = model.fit()

# Forecast
forecast = model_fit.forecast(steps=12)  # Forecasting next 24 periods

# Plotting the results
plt.figure(figsize=(15, 6))
plt.plot(weekly_sales, label='Historical')
plt.plot(forecast, label='Forecast', color='red')
plt.title('ARIMA Model Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()
```



Random Forest: Actual vs Predicted Graphs - Storewise and Productwise

Total sales vs predicted (all time)

In [78]:
```
final = pd.read_csv("UnseenData_predictions.csv")
```

In [79]:
```
final
```

Out[79]:

	Unnamed: 0	date	store_nbr	product_type	sales	predicted_sales
0	0	2017-07-31	1	0	8.0	4.778179
1	1	2017-08-01	1	0	5.0	7.581328
2	2	2017-08-02	1	0	4.0	5.442089
3	3	2017-08-03	1	0	3.0	4.572344
4	4	2017-08-04	1	0	8.0	4.816661
...	...	...	...	...	...	...
28507	28507	2017-08-11	54	32	0.0	3.443701
28508	28508	2017-08-12	54	32	1.0	5.491697
28509	28509	2017-08-13	54	32	2.0	2.754819
28510	28510	2017-08-14	54	32	0.0	2.810651
28511	28511	2017-08-15	54	32	3.0	2.076917

28512 rows × 6 columns

In [80]:
```
summed = pd.DataFrame(final.groupby('date')['sales'].sum())
```

In [81]:
```
summed
```

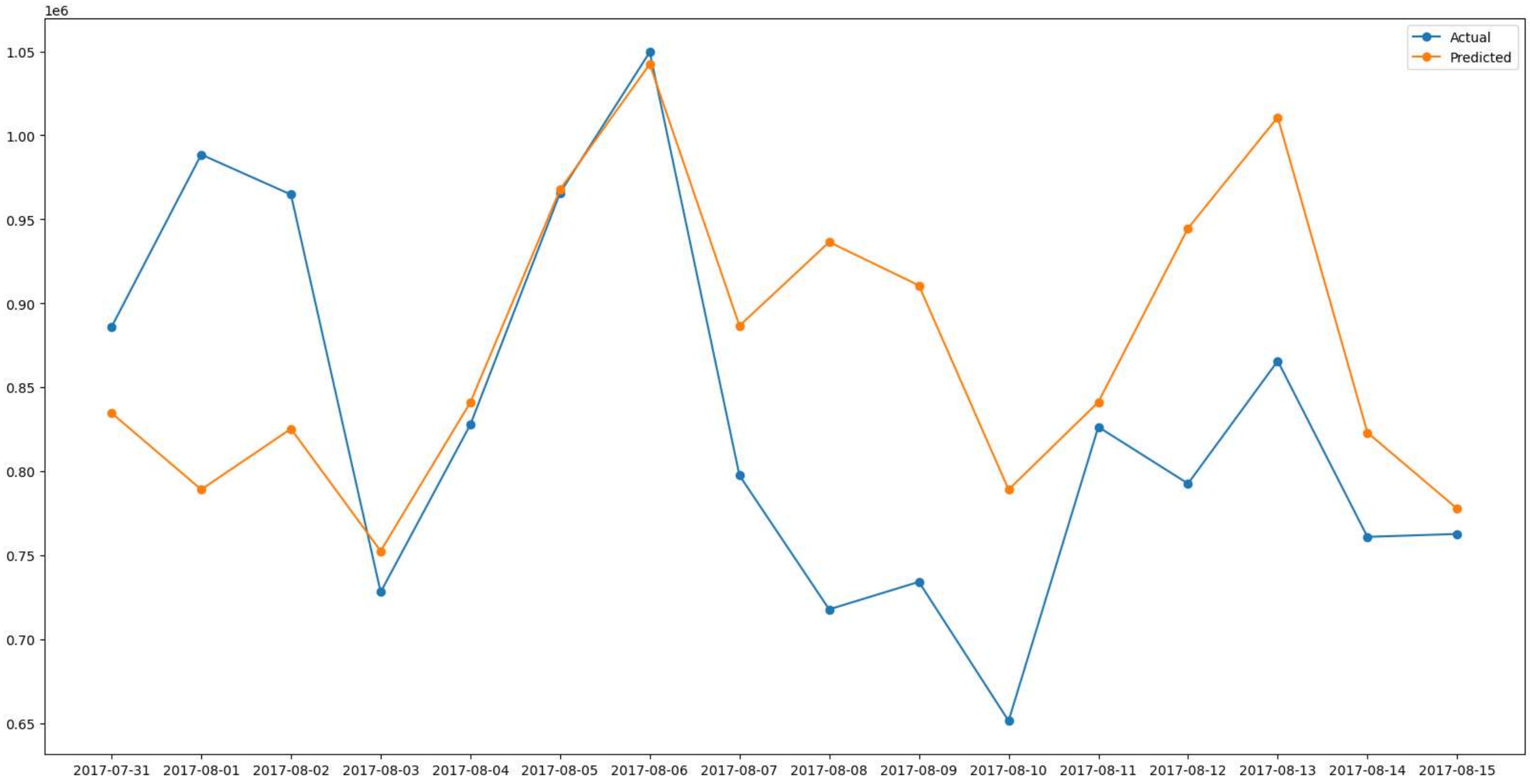
Out[81]:

	sales
date	
2017-07-31	8.858568e+05
2017-08-01	9.885278e+05
2017-08-02	9.647120e+05
2017-08-03	7.280685e+05
2017-08-04	8.277757e+05
2017-08-05	9.656937e+05
2017-08-06	1.049559e+06
2017-08-07	7.974650e+05
2017-08-08	7.177663e+05
2017-08-09	7.341397e+05
2017-08-10	6.513869e+05
2017-08-11	8.263737e+05
2017-08-12	7.926305e+05
2017-08-13	8.656397e+05
2017-08-14	7.609224e+05
2017-08-15	7.626619e+05

In [82]:
```
pred_summed = pd.DataFrame(final.groupby('date')['predicted_sales'].sum())
```

In [83]:
```
plt.figure(figsize=(20,10))
plt.plot(summed['sales'],marker='o',label='Actual')
plt.plot(pred_summed['predicted_sales'],marker='o',label='Predicted')
plt.legend()
```

Out[83]:
```
<matplotlib.legend.Legend at 0x196ec62b880>
```



Storewise graphs

In [84]:
```
# Filter data for the specified date range and unique stores
filtered_df = final[(final['date'] >= '2017-07-31') & (final['date'] <= '2017-08-15') &
                (final['store_nbr'].between(1, 54))]

# Group by 'date' and 'store_nbr' and sum the 'sales' for each date and store
summed_sales= filtered_df.groupby(['date', 'store_nbr'])['sales'].sum().reset_index()
```

```
# Print or plot the results
print(summed_sales)
```

```
           date  store_nbr         sales
0    2017-07-31          1  11626.725000
1    2017-07-31          2  13739.413996
2    2017-07-31          3  34192.049000
3    2017-07-31          4  11622.438000
4    2017-07-31          5  10798.739000
..          ...        ...           ...
859  2017-08-15         50  16879.121004
860  2017-08-15         51  20154.559000
861  2017-08-15         52  18600.046000
862  2017-08-15         53   8208.189000
863  2017-08-15         54  12666.858000

[864 rows x 3 columns]
```

In [85]:
```
# Filter data for the specified date range and unique stores
filtered_df = final[(final['date'] >= '2017-07-31') & (final['date'] <= '2017-08-15') &
                (final['store_nbr'].between(1, 54))]

# Group by 'date' and 'store_nbr' and sum the 'sales' for each date and store
summed_predicted_sales = filtered_df.groupby(['date', 'store_nbr'])['predicted_sales'].sum().reset_index()

# Print or plot the results
print(summed_predicted_sales)
```

```
           date  store_nbr  predicted_sales
0    2017-07-31          1     10610.021719
1    2017-07-31          2     14636.663155
2    2017-07-31          3     34543.187913
3    2017-07-31          4     12129.760010
4    2017-07-31          5     10776.877988
..          ...        ...              ...
859  2017-08-15         50     18078.063624
860  2017-08-15         51     19428.022989
861  2017-08-15         52     19606.851438
862  2017-08-15         53      9760.900824
863  2017-08-15         54     11647.420235

[864 rows x 3 columns]
```

In [86]:
```
merged_sales = pd.merge(summed_sales,summed_predicted_sales,how='inner',on=['store_nbr','date'])
```

In [87]:
```
merged_sales
```

Out[87]:

	date	store_nbr	sales	predicted_sales
0	2017-07-31	1	11626.725000	10610.021719
1	2017-07-31	2	13739.413996	14636.663155
2	2017-07-31	3	34192.049000	34543.187913
3	2017-07-31	4	11622.438000	12129.760010
4	2017-07-31	5	10798.739000	10776.877988
...	...	...	...	...
859	2017-08-15	50	16879.121004	18078.063624
860	2017-08-15	51	20154.559000	19428.022989
861	2017-08-15	52	18600.046000	19606.851438
862	2017-08-15	53	8208.189000	9760.900824
863	2017-08-15	54	12666.858000	11647.420235

864 rows × 4 columns

In [88]:

```
sns.set(style="whitegrid")

# Iterate over unique store numbers
for store_nbr in merged_sales['store_nbr'].unique():
    # Filter data for the current store
    store_data = merged_sales[merged_sales['store_nbr'] == store_nbr]

    # Create a line plot for sales and predicted sales
    plt.figure(figsize=(10, 5))
    sns.lineplot(x='date', y='sales', data=store_data, label='Actual Sales',marker='o')
    sns.lineplot(x='date', y='predicted_sales', data=store_data, label='Predicted Sales',marker='o')

    # Set plot labels and title
    plt.xlabel('Date')
    plt.ylabel('Sales')
    plt.title(f'Sales and Predicted Sales for Store {store_nbr}')

    # Rotate x-axis labels for better visibility
    plt.xticks(rotation=45, ha='right')

    # Show the legend
    plt.legend()

    # Show the plot
    plt.show()
```


Sales and Predicted Sales for Store 1
Actual Sales
Predicted Sales
Sales
Date
14000
12000
10000
8000
6000
4000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 2
Actual Sales
Predicted Sales
Sales
17000
16000
15000
14000
13000
12000
11000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 3
Actual Sales
Predicted Sales
Sales
45000
40000
35000
30000
25000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 4
Actual Sales
Predicted Sales
Sales
Date
17000
16000
15000
14000
13000
12000
11000
10000
9000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 5
Actual Sales
Predicted Sales
Sales
12000
11000
10000
9000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 6
Actual Sales
Predicted Sales
Sales
22000
20000
18000
16000
14000
12000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

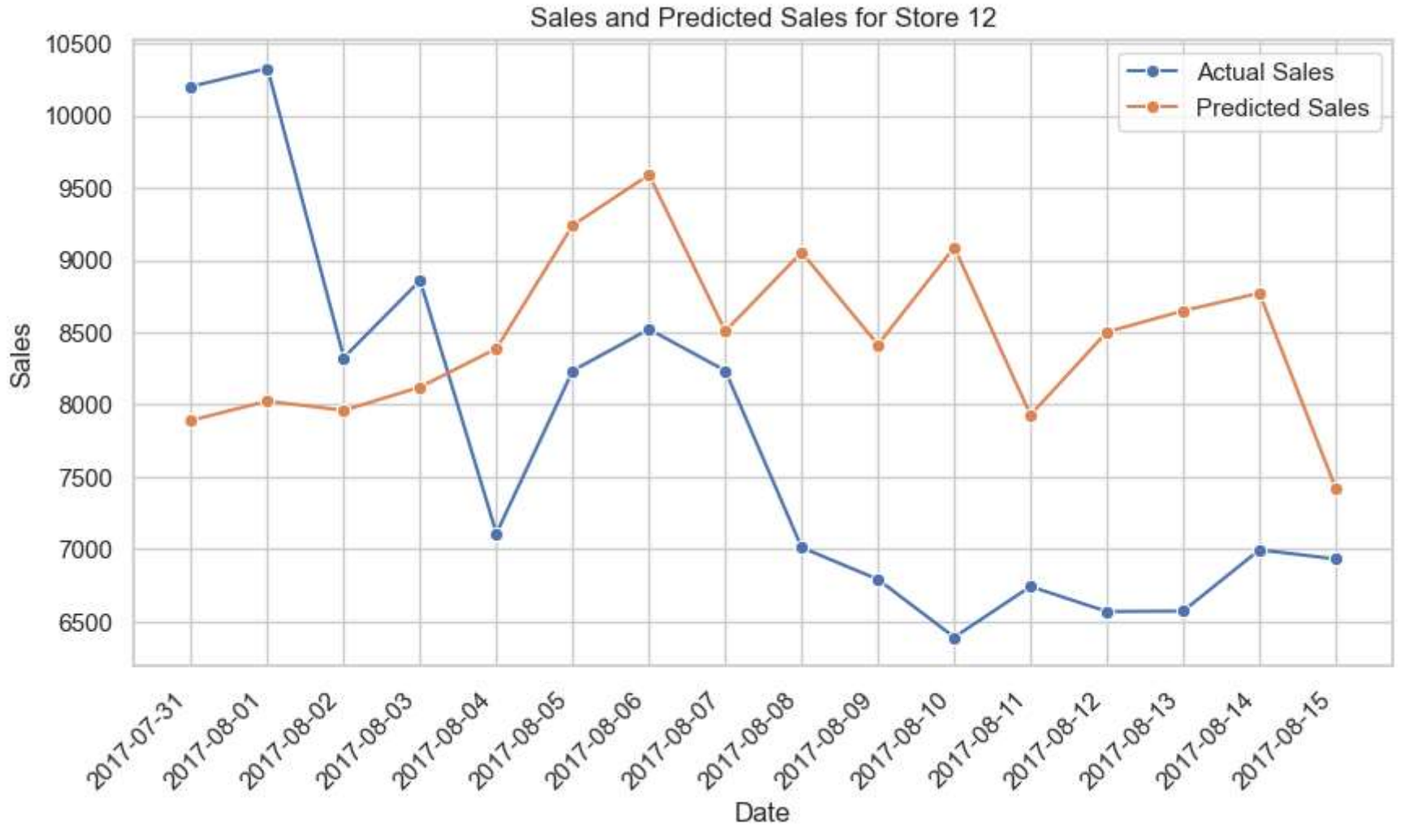

Sales and Predicted Sales for Store 7
Actual Sales
Predicted Sales
Sales
Date
24000
22000
20000
18000
16000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 8
Actual Sales
Predicted Sales
Sales
Date
26000
24000
22000
20000
18000
16000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

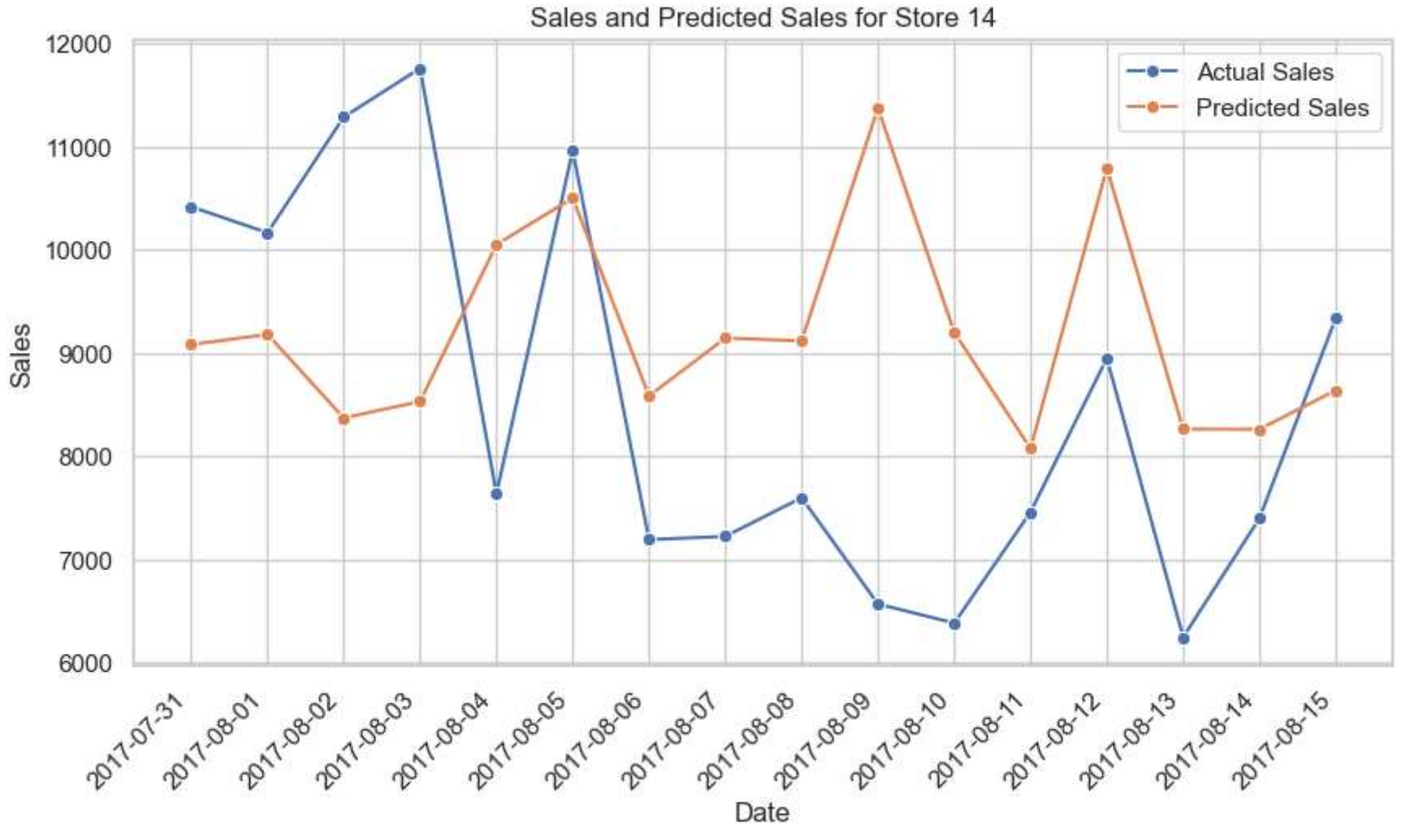

Sales and Predicted Sales for Store 9
Actual Sales
Predicted Sales
Sales
26000
24000
22000
20000
18000
16000
14000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

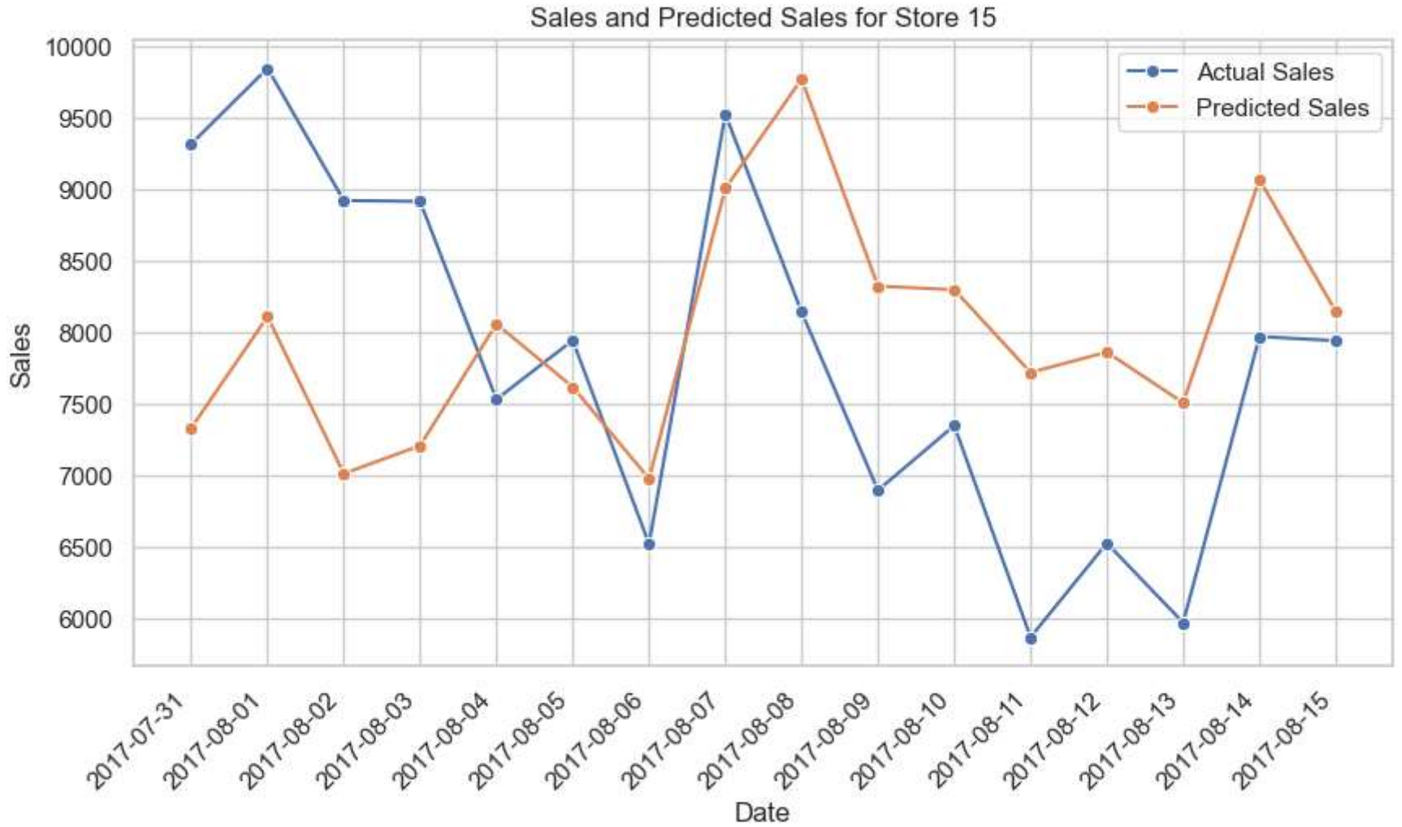

Sales and Predicted Sales for Store 10
Actual Sales
Predicted Sales
Sales
9000
8500
8000
7500
7000
6500
6000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 11
Actual Sales
Predicted Sales
Sales
28000
26000
24000
22000
20000
18000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

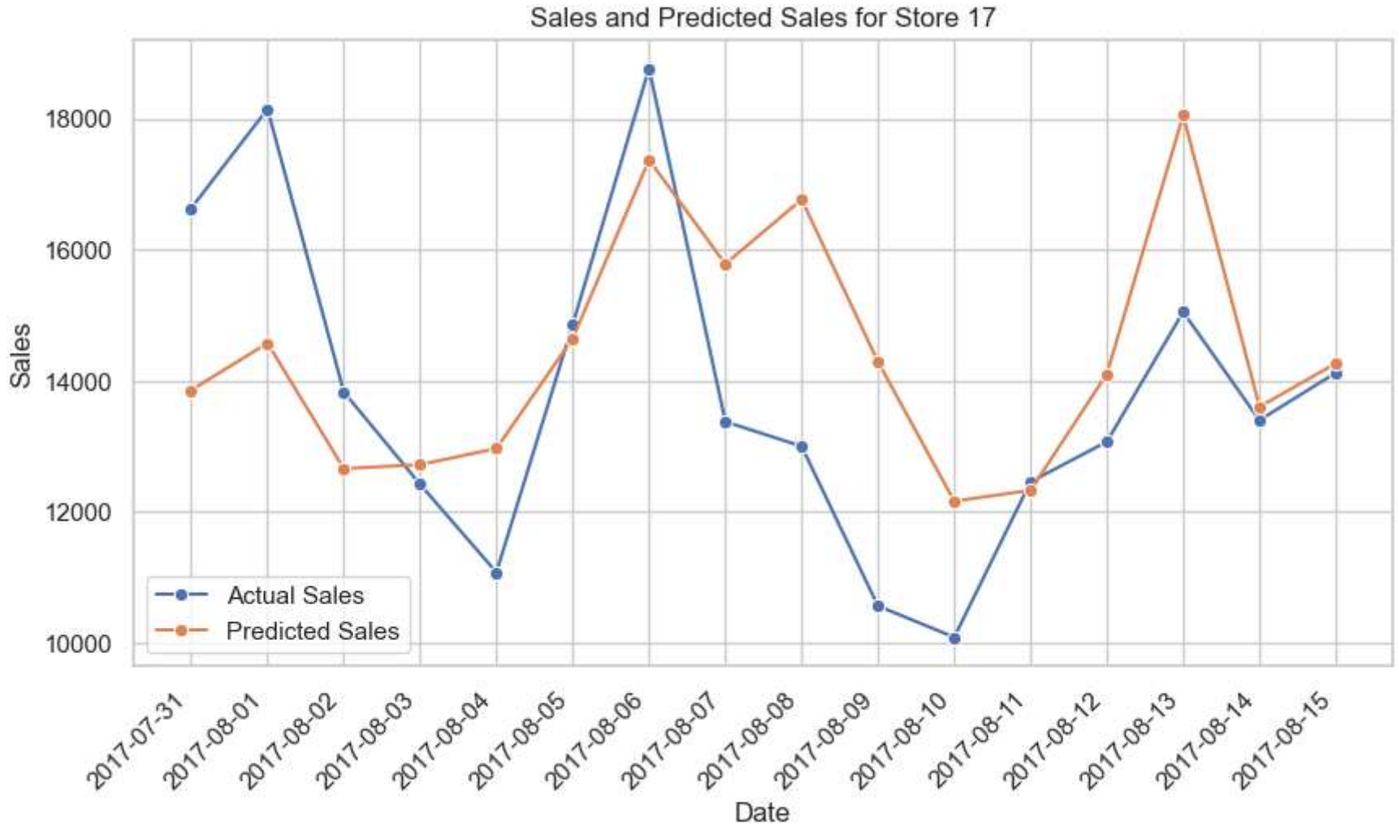

Sales and Predicted Sales for Store 12
Actual Sales
Predicted Sales
Sales
Date
10500
10000
9500
9000
8500
8000
7500
7000
6500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

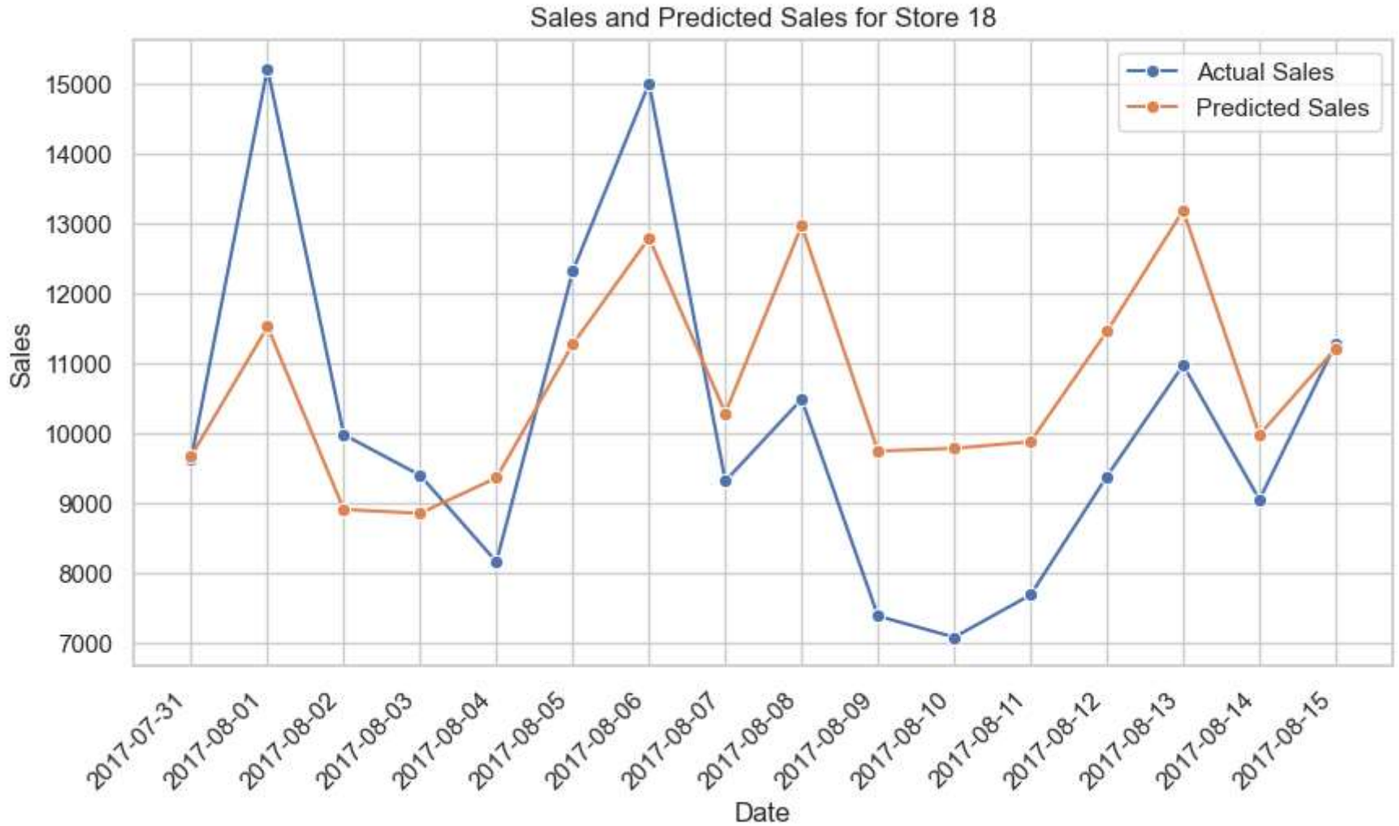

Sales and Predicted Sales for Store 13
Actual Sales
Predicted Sales
Sales
Date
11000
10000
9000
8000
7000
6000
5000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

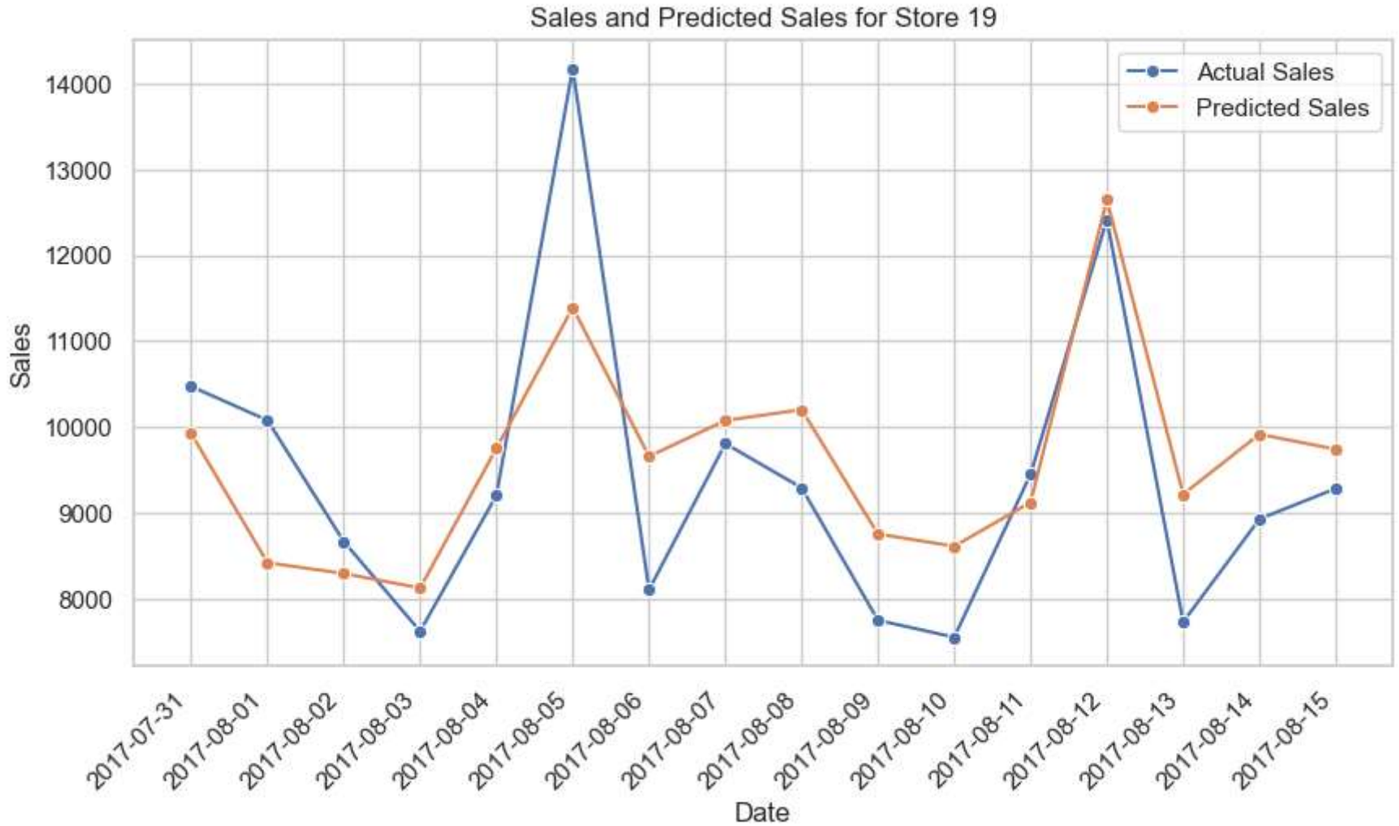

Sales and Predicted Sales for Store 14
Actual Sales
Predicted Sales
Sales
Date
12000
11000
10000
9000
8000
7000
6000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 15
Actual Sales
Predicted Sales
Sales
Date
10000
9500
9000
8500
8000
7500
7000
6500
6000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 16
Actual Sales
Predicted Sales
Sales
Date
9000
8000
7000
6000
5000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

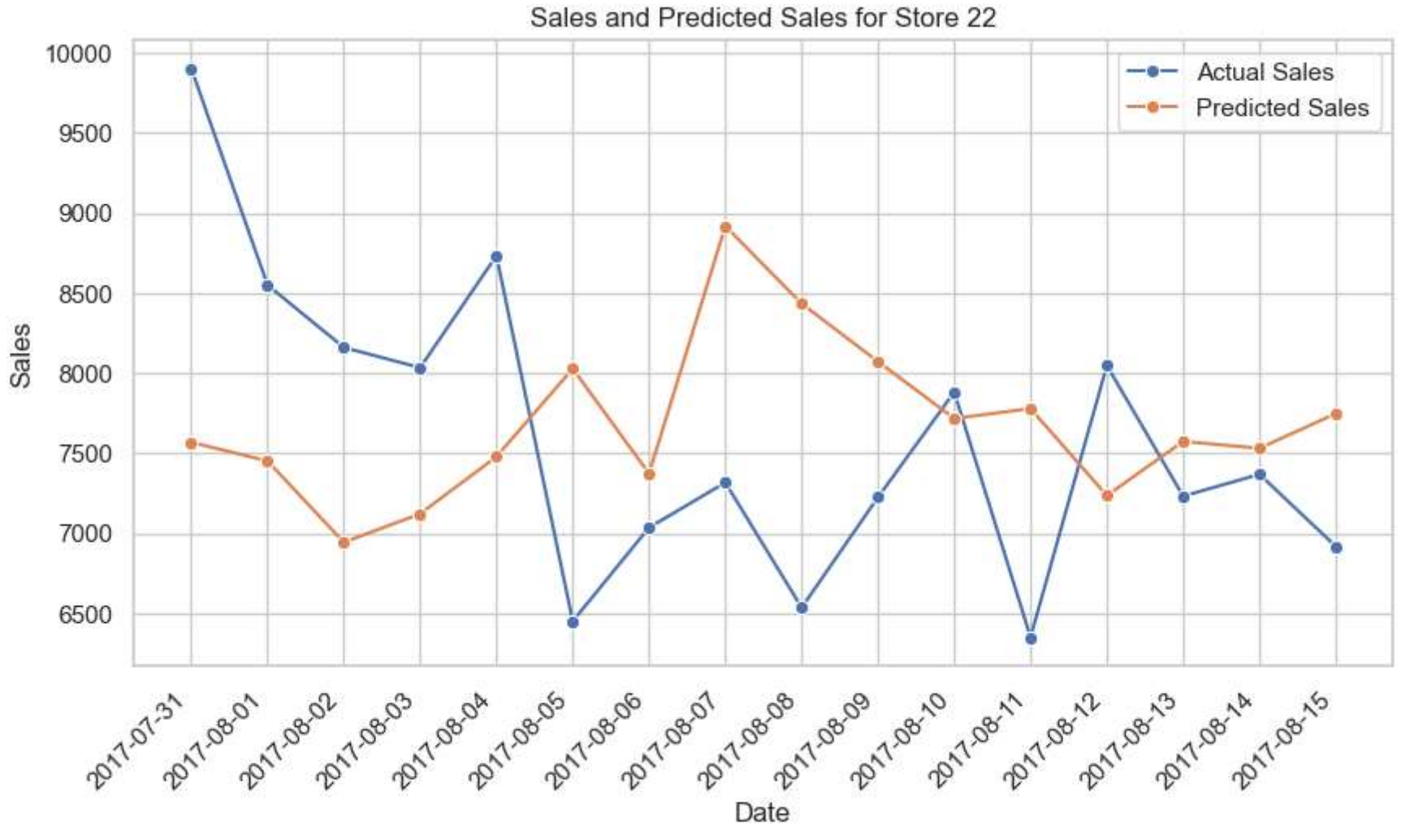

Sales and Predicted Sales for Store 17
Sales
Date
Actual Sales
Predicted Sales
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 18
Actual Sales
Predicted Sales
Sales
15000
14000
13000
12000
11000
10000
9000
8000
7000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 19
Actual Sales
Predicted Sales
Sales
14000
13000
12000
11000
10000
9000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

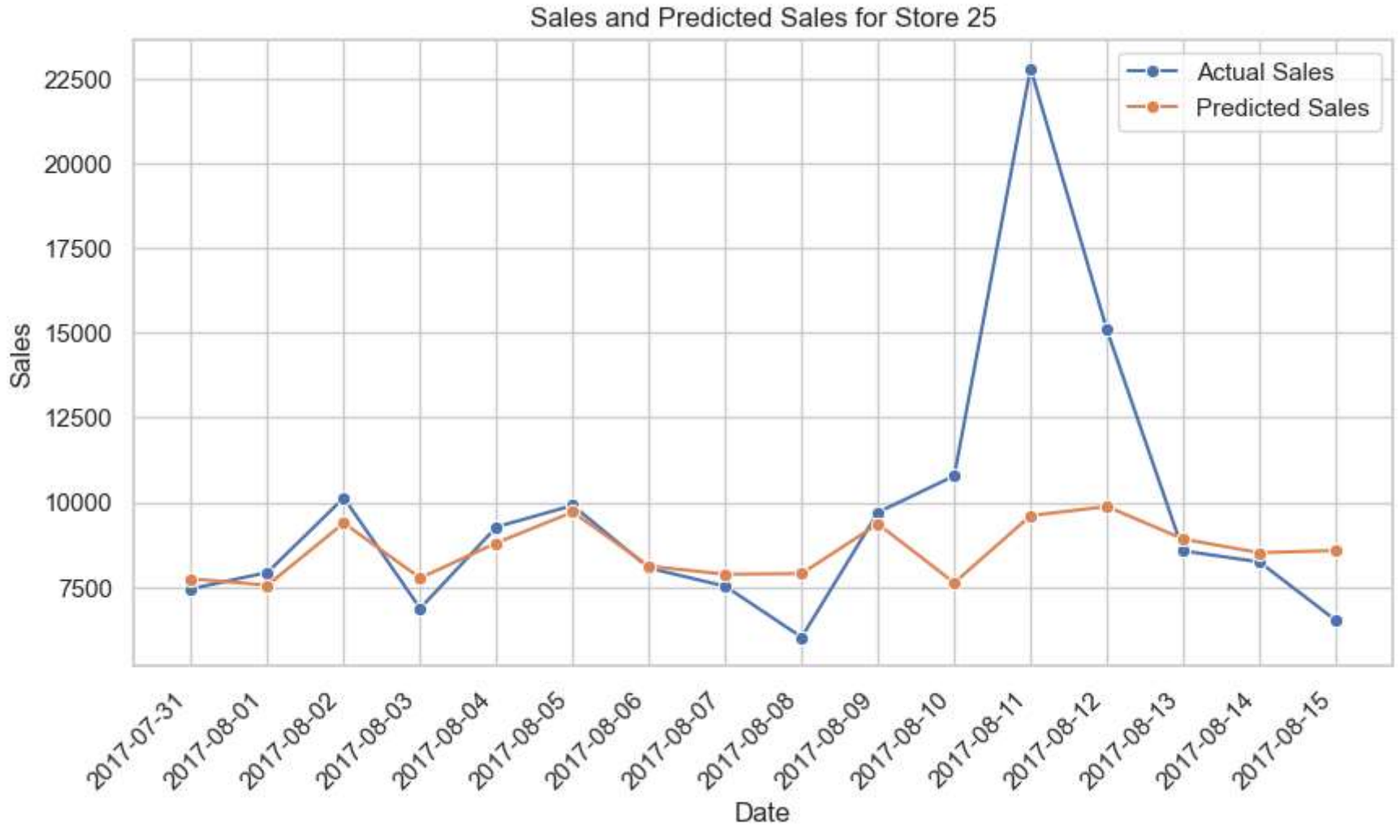

Sales and Predicted Sales for Store 20
Actual Sales
Predicted Sales
Sales
Date
26000
24000
22000
20000
18000
16000
14000
12000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 21
Actual Sales
Predicted Sales
Sales
Date
22000
20000
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 22
Actual Sales
Predicted Sales
Sales
Date
10000
9500
9000
8500
8000
7500
7000
6500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 23
Actual Sales
Predicted Sales
Sales
9000
8500
8000
7500
7000
6500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

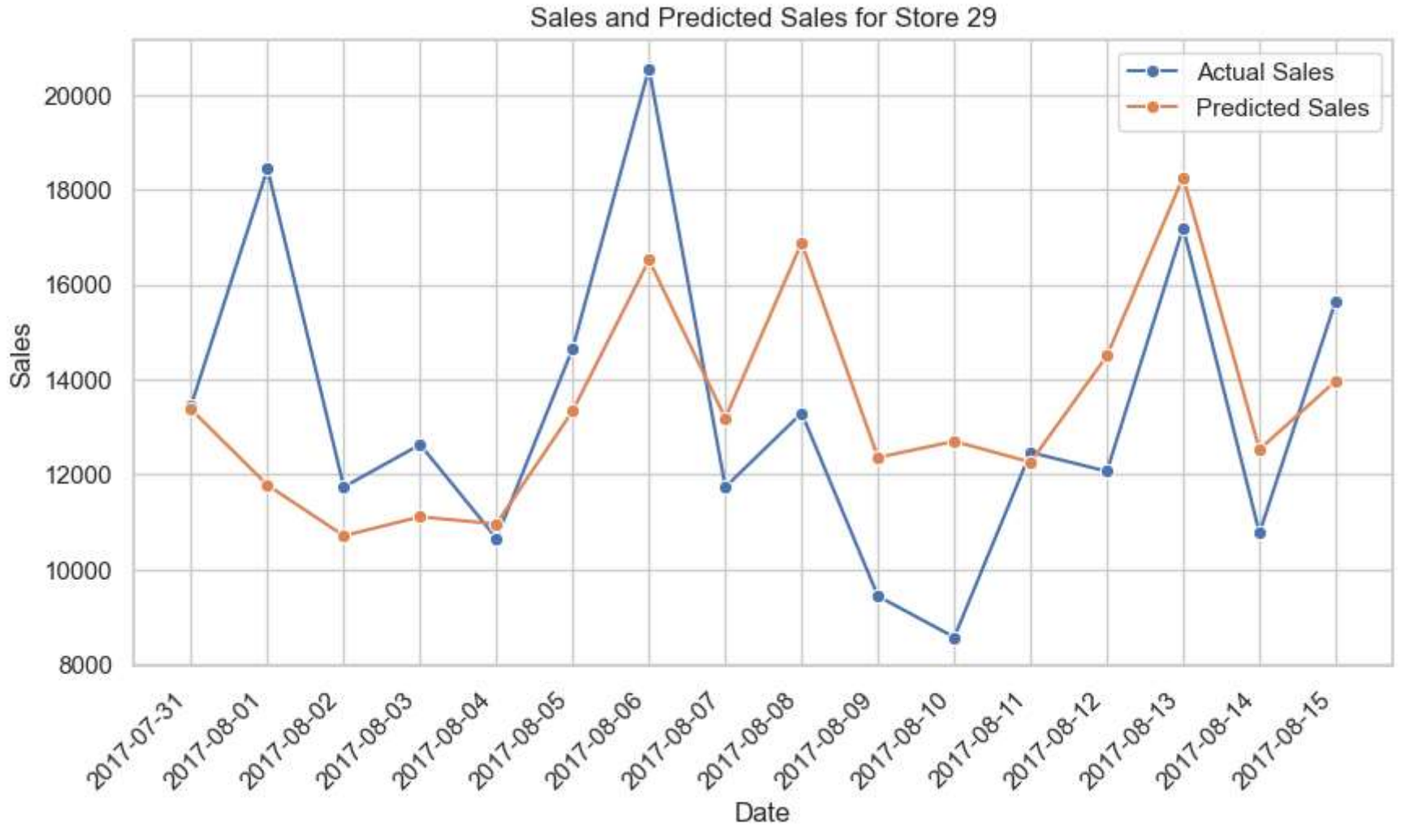

Sales and Predicted Sales for Store 24
Actual Sales
Predicted Sales
Sales
Date
21000
20000
19000
18000
17000
16000
15000
14000
13000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

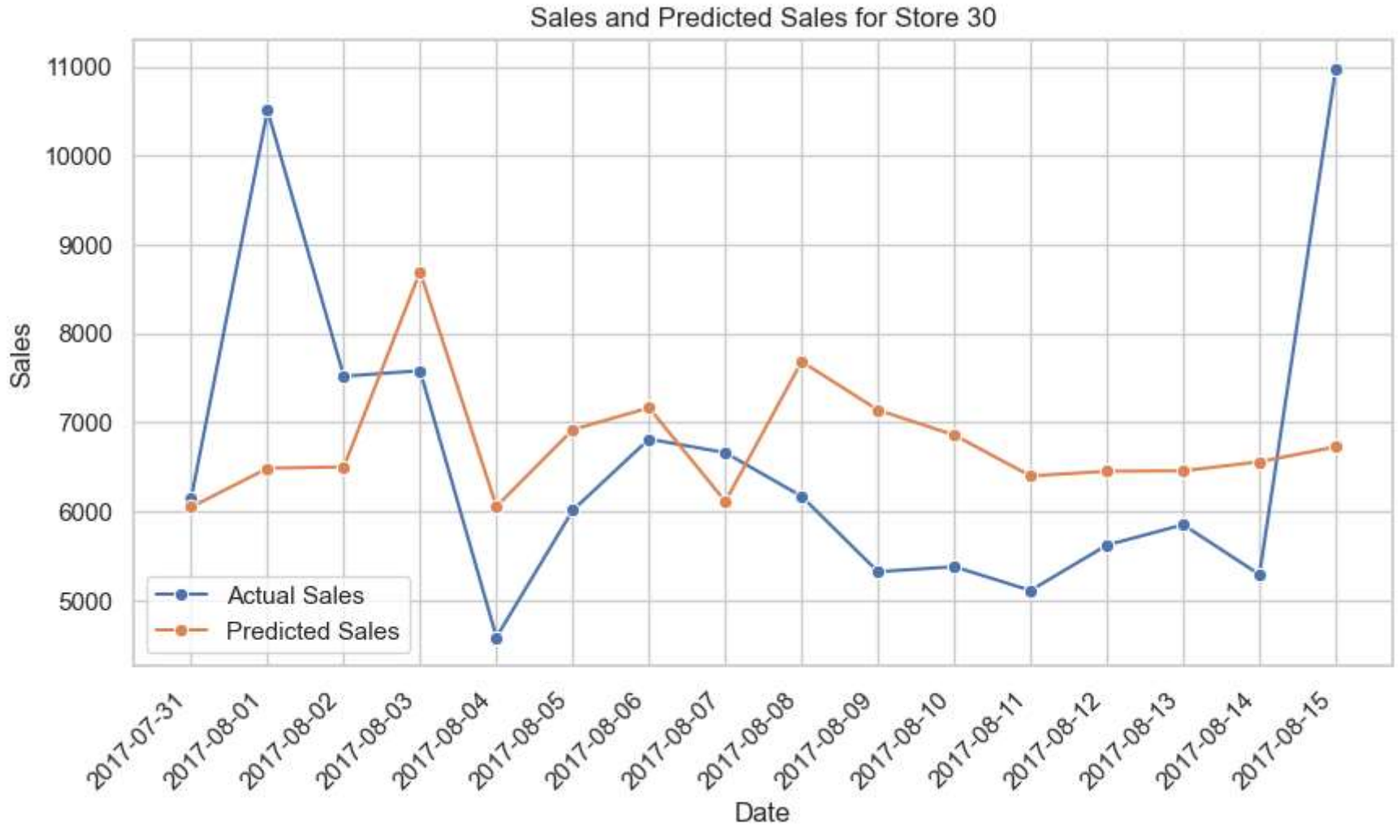

Sales and Predicted Sales for Store 25
Actual Sales
Predicted Sales
Sales
22500
20000
17500
15000
12500
10000
7500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 26
Actual Sales
Predicted Sales
Sales
8000
7000
6000
5000
4000
3000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 27
Actual Sales
Predicted Sales
Sales
Date
22000
20000
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 28
Actual Sales
Predicted Sales
Sales
Date
27500
25000
22500
20000
17500
15000
12500
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

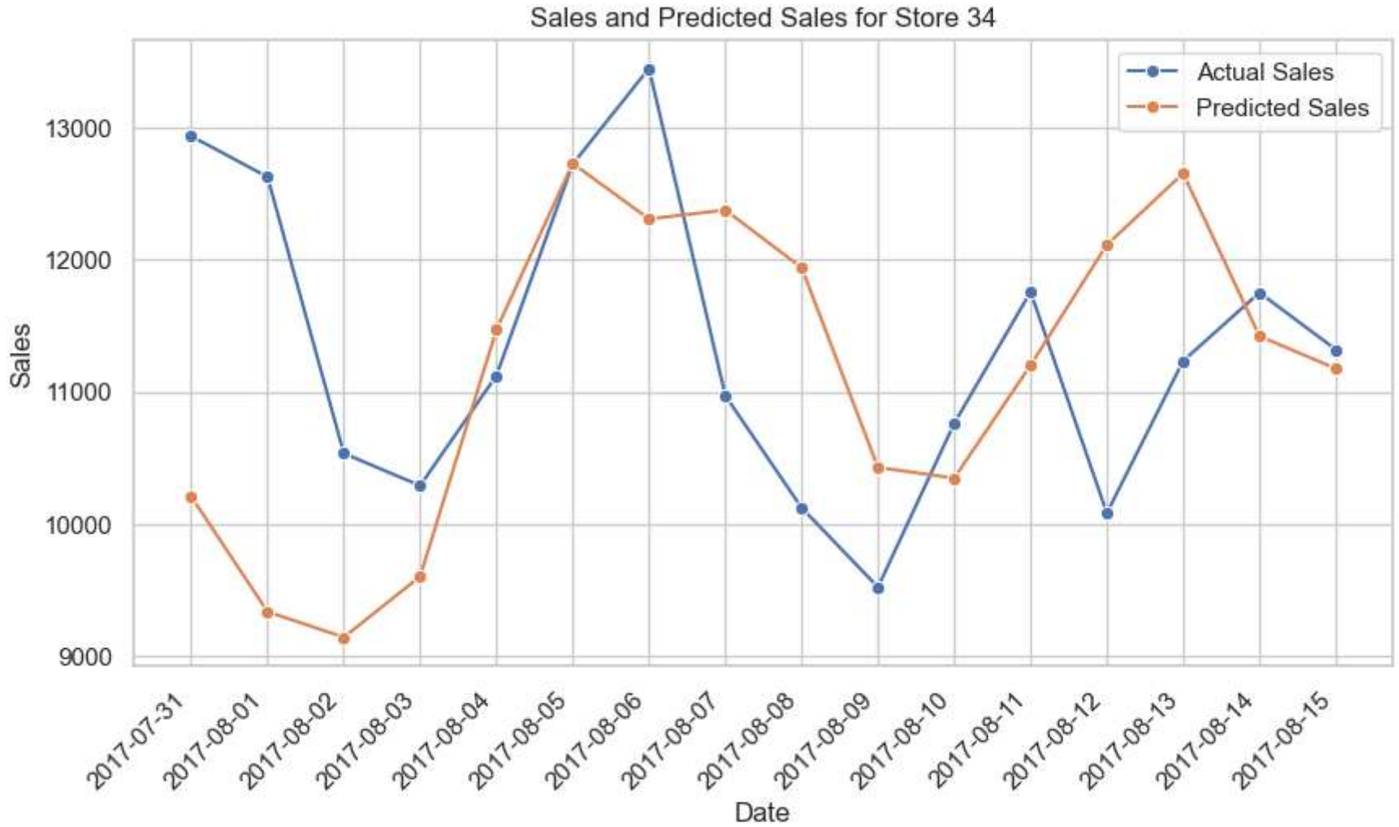

Sales and Predicted Sales for Store 29
Actual Sales
Predicted Sales
Sales
Date
20000
18000
16000
14000
12000
10000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 30
Sales
Date
Actual Sales
Predicted Sales
11000
10000
9000
8000
7000
6000
5000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 31
Actual Sales
Predicted Sales
Sales
Date
22000
20000
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 32
Actual Sales
Predicted Sales
Sales
7000
6500
6000
5500
5000
4500
4000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

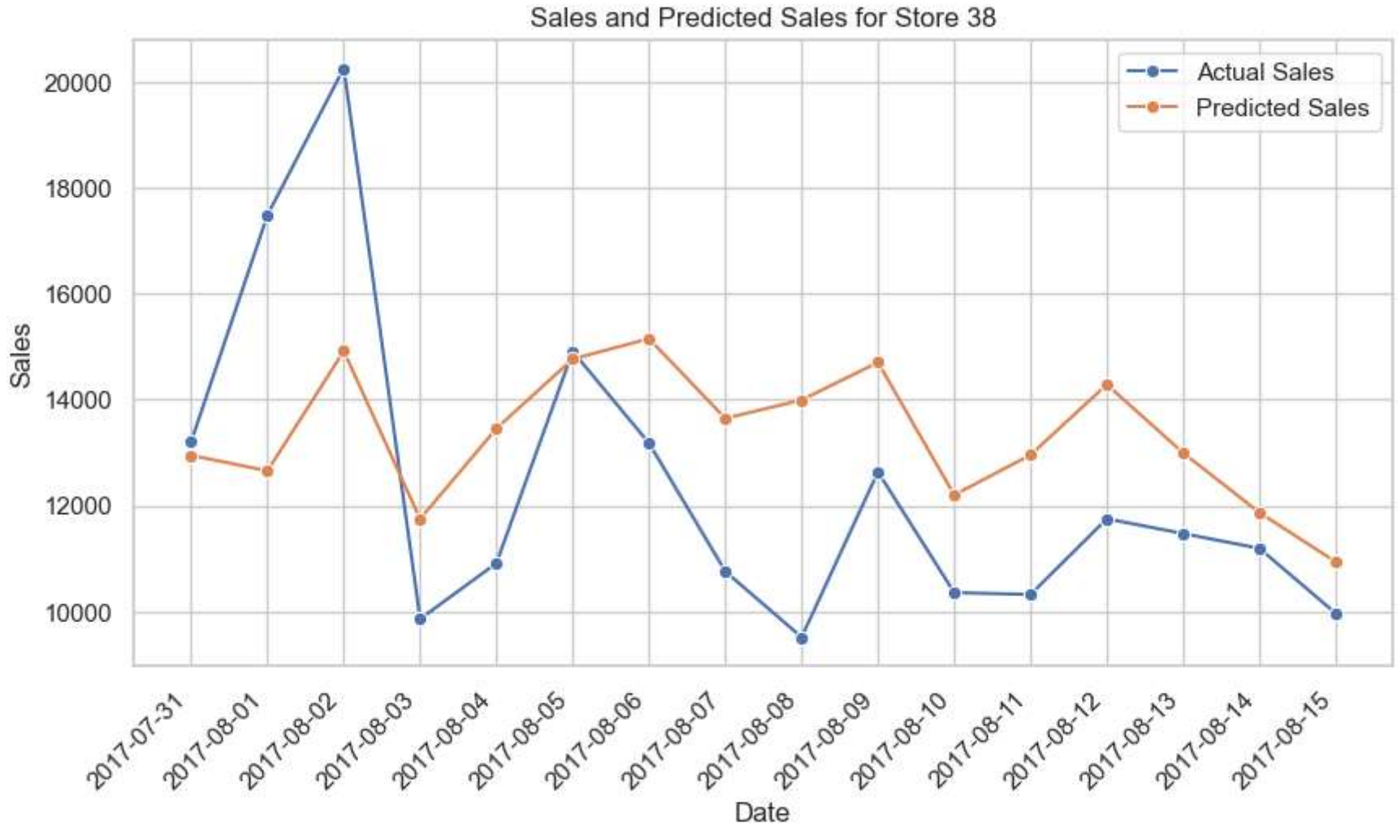

Sales and Predicted Sales for Store 33
Actual Sales
Predicted Sales
Sales
14000
12000
10000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

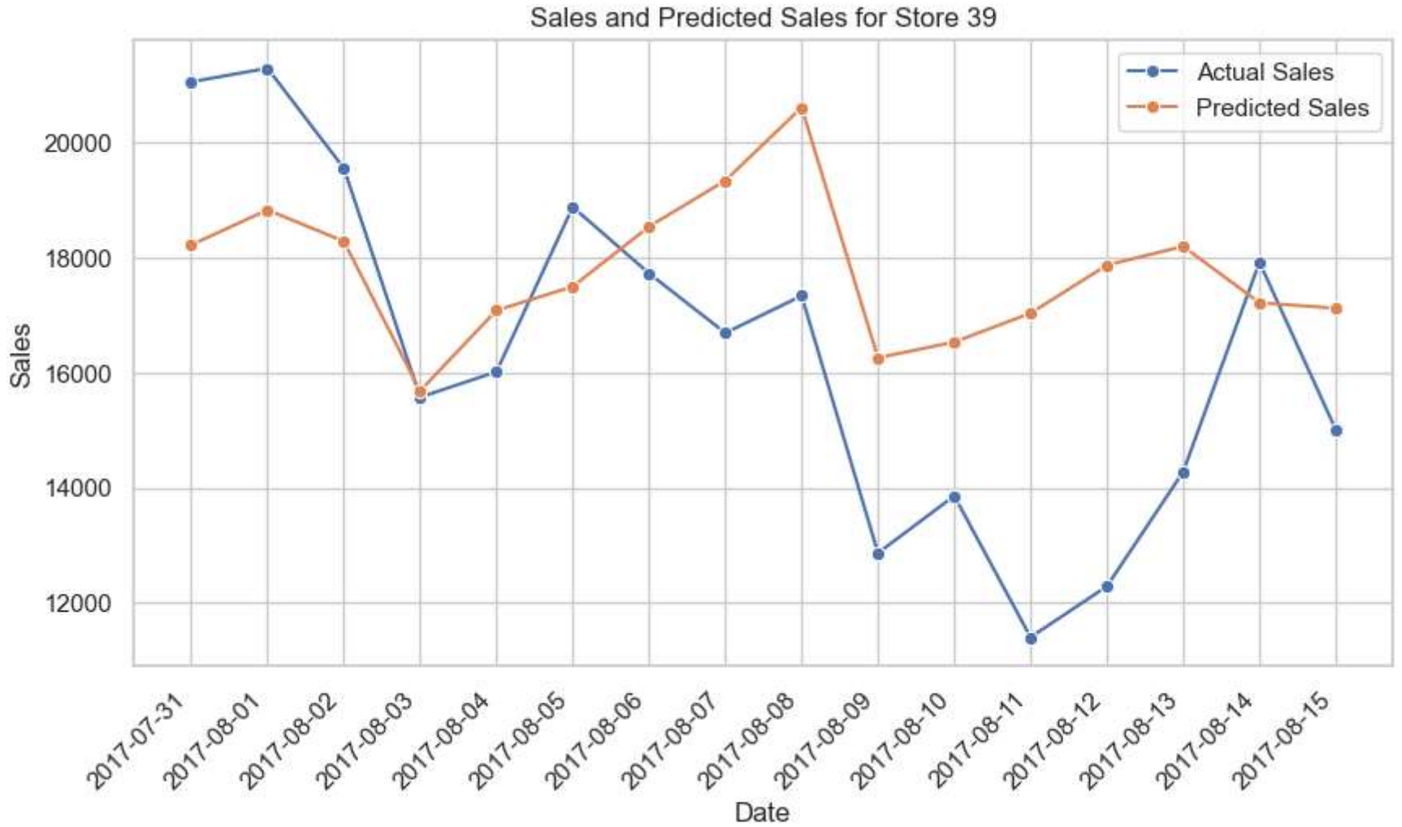

Sales and Predicted Sales for Store 34
Actual Sales
Predicted Sales
Sales
13000
12000
11000
10000
9000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 35
Actual Sales
Predicted Sales
Sales
7000
6500
6000
5500
5000
4500
4000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

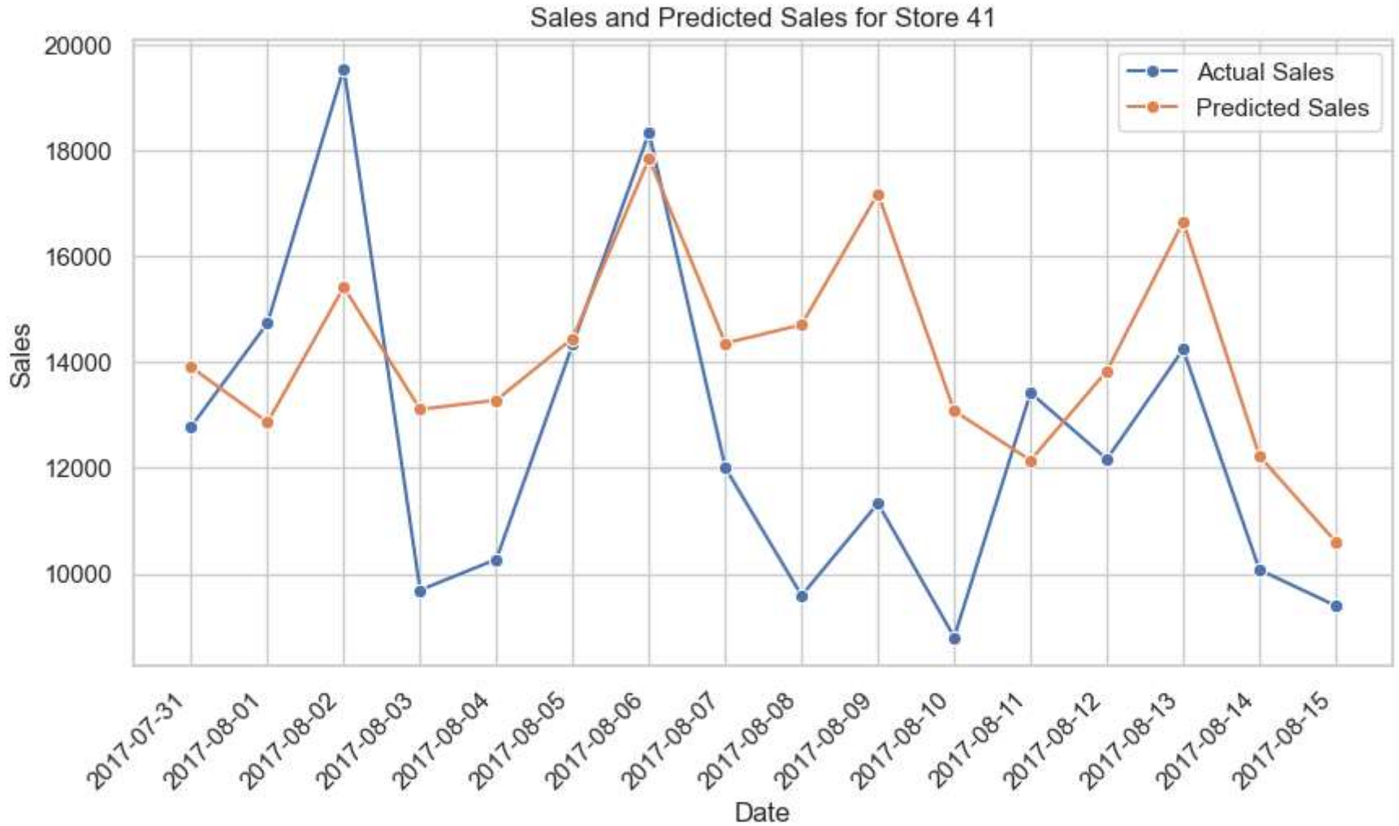

Sales and Predicted Sales for Store 36
Actual Sales
Predicted Sales
Sales
18000
16000
14000
12000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

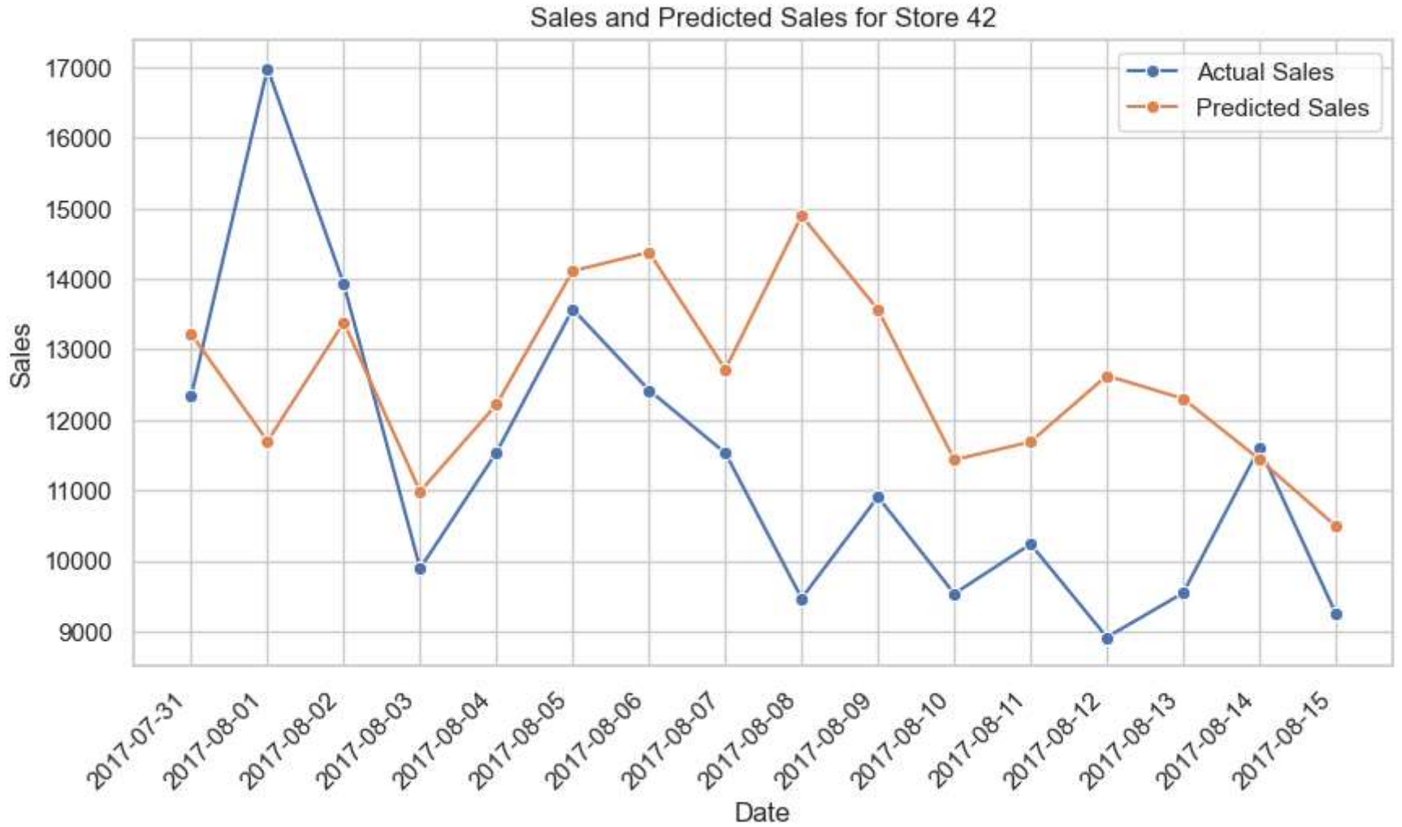

Sales and Predicted Sales for Store 37
Actual Sales
Predicted Sales
Sales
16000
14000
12000
10000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 38
Actual Sales
Predicted Sales
Sales
20000
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

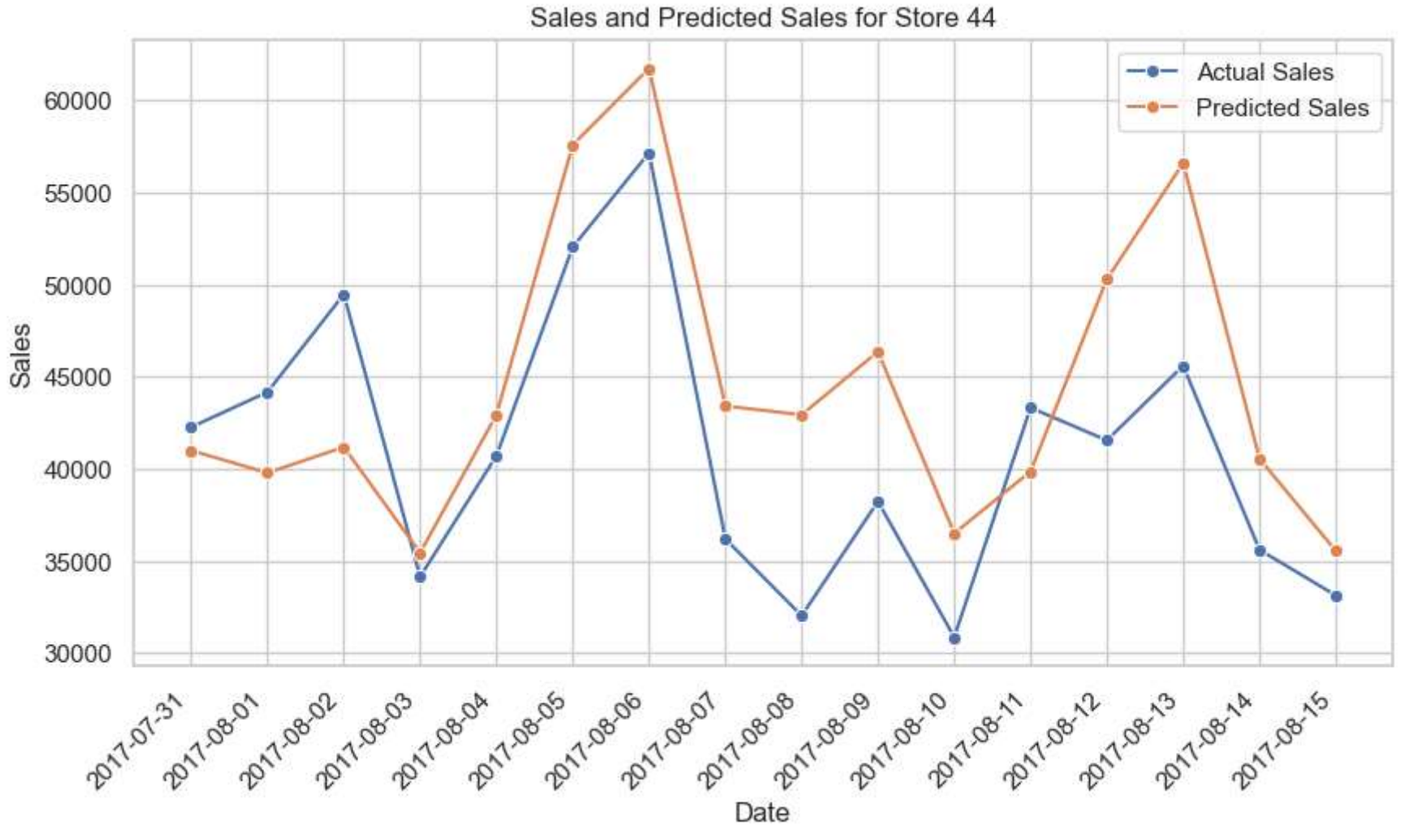

Sales and Predicted Sales for Store 39
Actual Sales
Predicted Sales
Sales
20000
18000
16000
14000
12000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 40
Actual Sales
Predicted Sales
Sales
24000
22000
20000
18000
16000
14000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

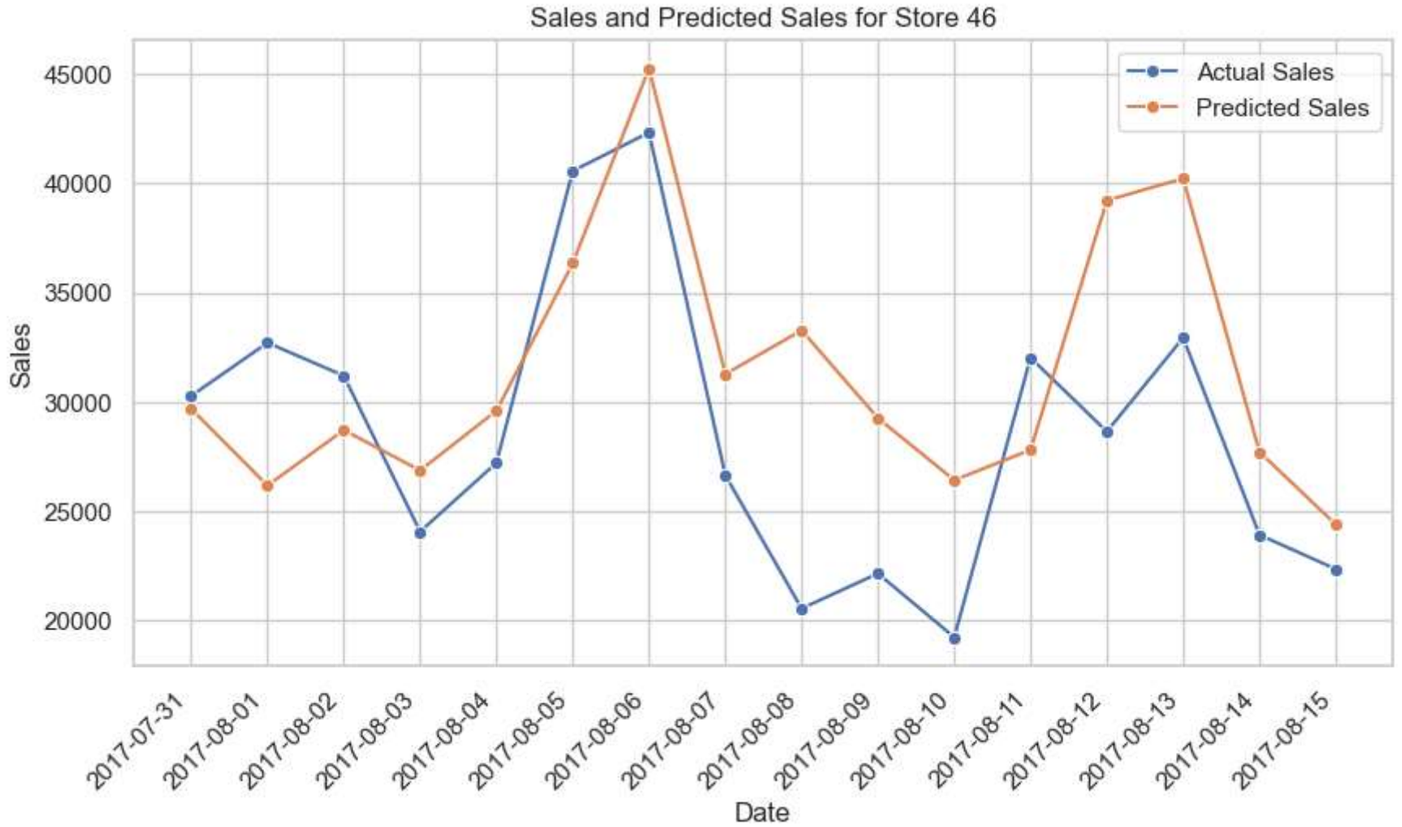

Sales and Predicted Sales for Store 41
Actual Sales
Predicted Sales
Sales
Date
20000
18000
16000
14000
12000
10000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 42
Actual Sales
Predicted Sales
Sales
Date
17000
16000
15000
14000
13000
12000
11000
10000
9000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 43
Actual Sales
Predicted Sales
Sales
Date
20000
19000
18000
17000
16000
15000
14000
13000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 44
Actual Sales
Predicted Sales
Sales
60000
55000
50000
45000
40000
35000
30000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

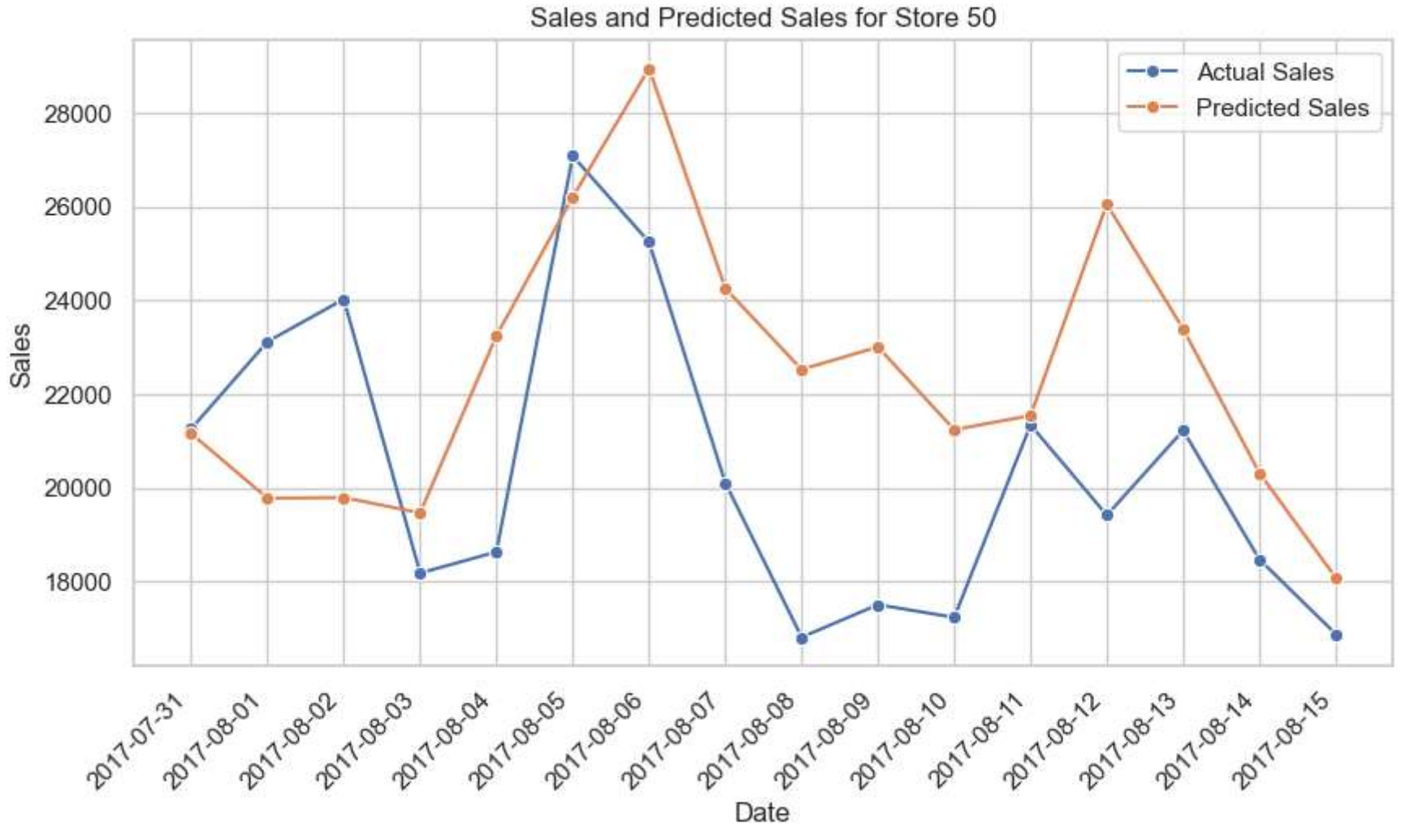

Sales and Predicted Sales for Store 45
Actual Sales
Predicted Sales
Sales
55000
50000
45000
40000
35000
30000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 46
Actual Sales
Predicted Sales
Sales
Date
45000
40000
35000
30000
25000
20000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 47
Actual Sales
Predicted Sales
Sales
50000
45000
40000
35000
30000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 48
Actual Sales
Predicted Sales
Sales
40000
35000
30000
25000
20000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

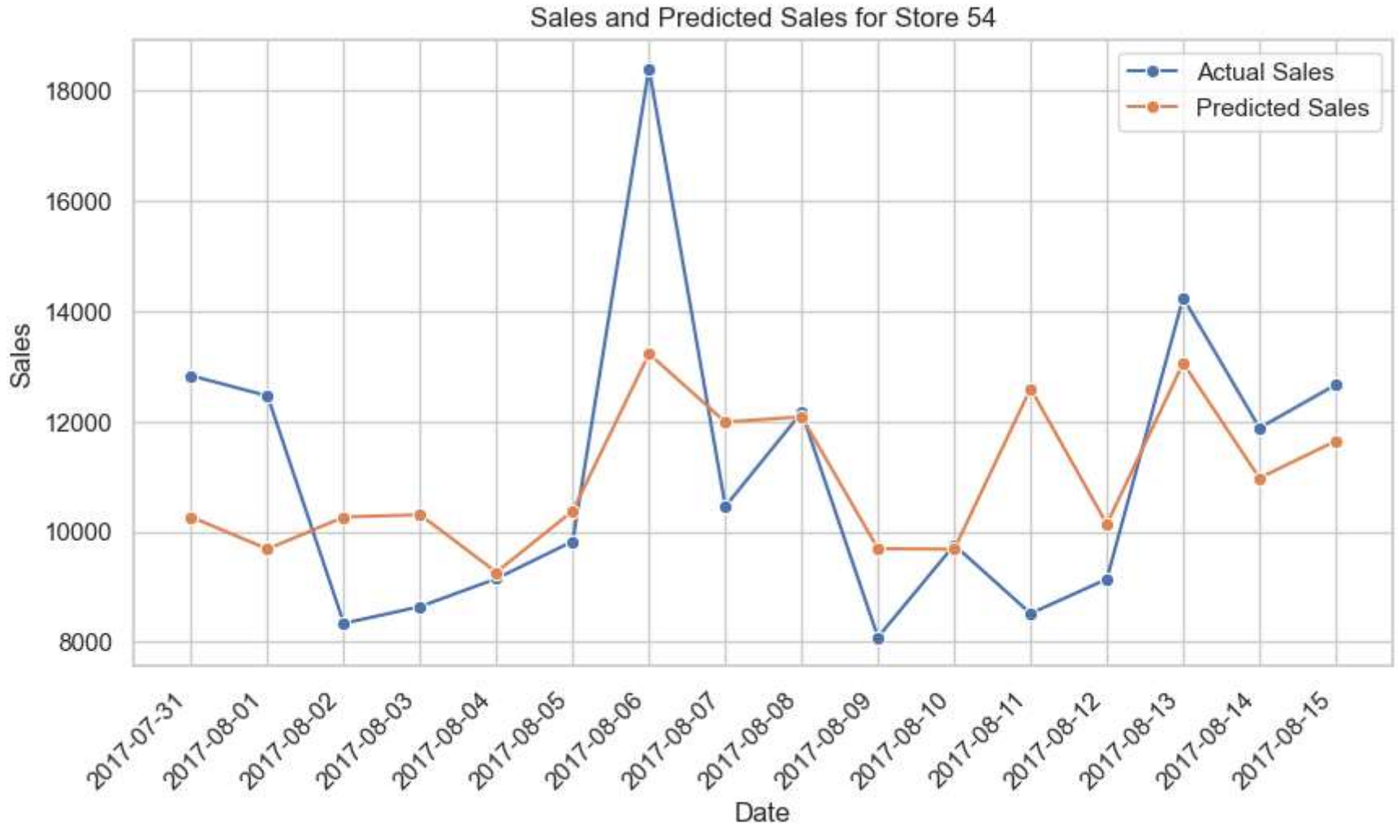

Sales and Predicted Sales for Store 49
Actual Sales
Predicted Sales
Sales
Date
42000
40000
38000
36000
34000
32000
30000
28000
26000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 50
Actual Sales
Predicted Sales
Sales
28000
26000
24000
22000
20000
18000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 51
Sales
Date
Actual Sales
Predicted Sales
30000
28000
26000
24000
22000
20000
18000
16000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15


Sales and Predicted Sales for Store 52
Actual Sales
Predicted Sales
Sales
30000
28000
26000
24000
22000
20000
18000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


Sales and Predicted Sales for Store 53
Actual Sales
Predicted Sales
Sales
16000
15000
14000
13000
12000
11000
10000
9000
8000
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date



Product Wise Graph

In [98]:
```
final['product_type'] = labelencoder.inverse_transform(final['product_type'])
```

In [99]:
```
final
```

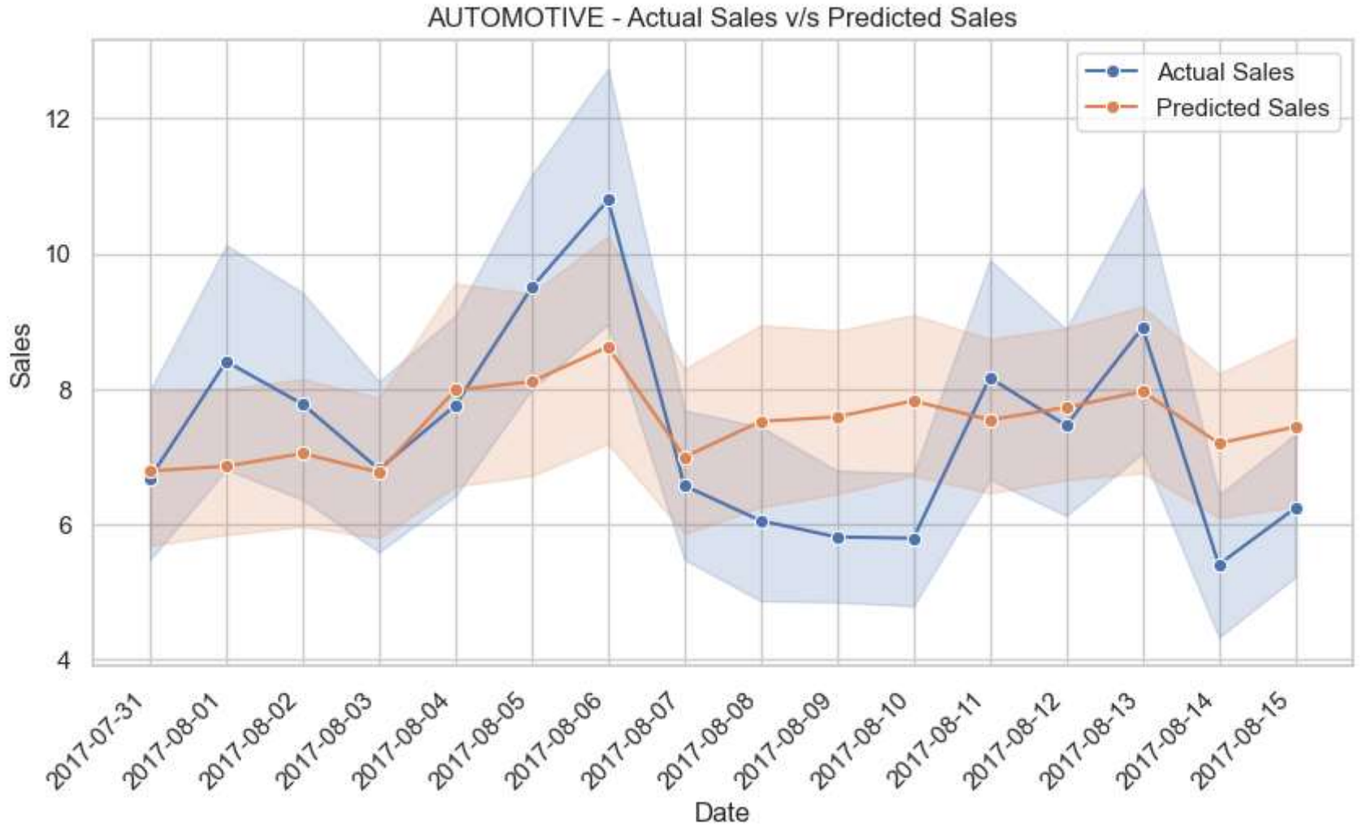
Out[99]:

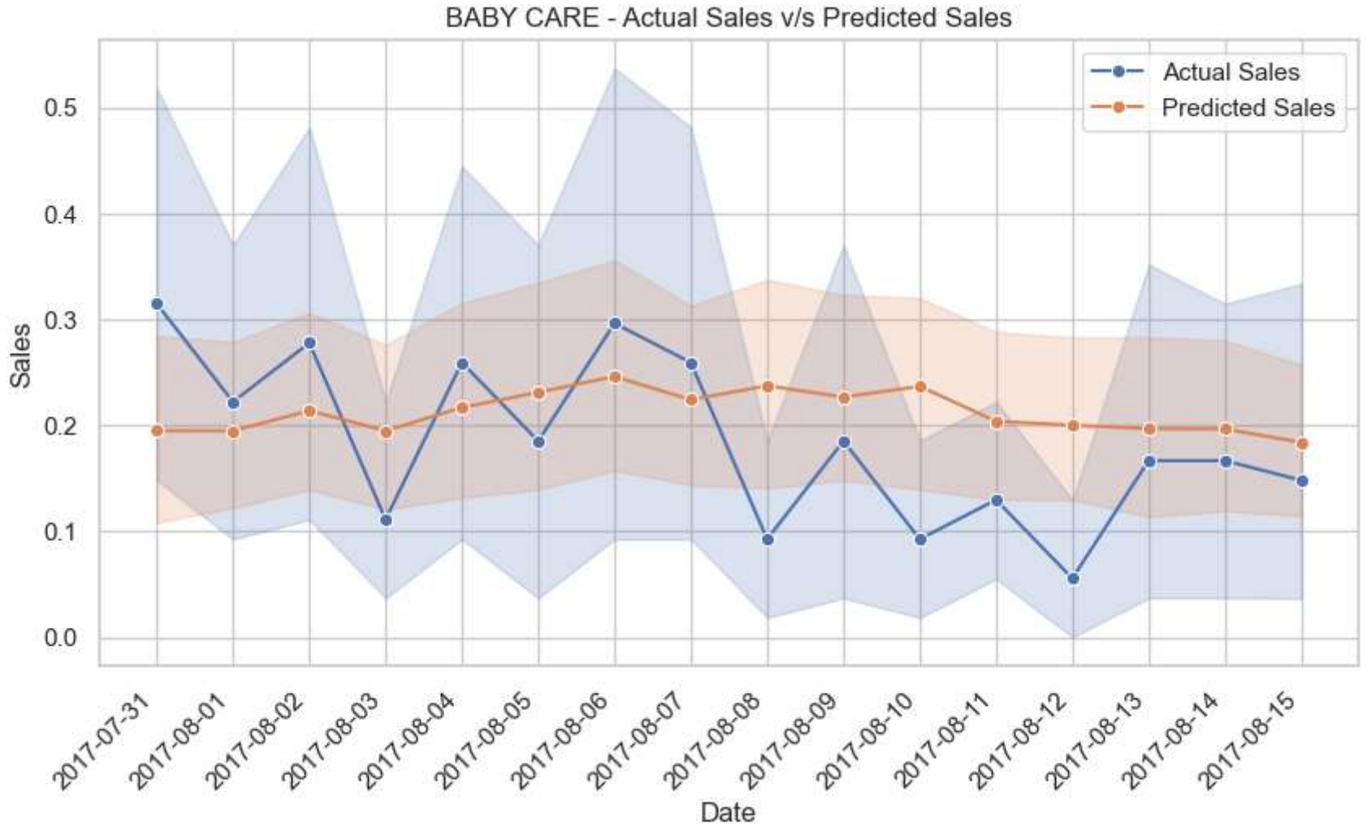
	Unnamed: 0	date	store_nbr	product_type	sales	predicted_sales
0	0	2017-07-31	1	AUTOMOTIVE	8.0	4.778179
1	1	2017-08-01	1	AUTOMOTIVE	5.0	7.581328
2	2	2017-08-02	1	AUTOMOTIVE	4.0	5.442089
3	3	2017-08-03	1	AUTOMOTIVE	3.0	4.572344
4	4	2017-08-04	1	AUTOMOTIVE	8.0	4.816661
...	...	...	...	...	...	...
28507	28507	2017-08-11	54	SEAFOOD	0.0	3.443701
28508	28508	2017-08-12	54	SEAFOOD	1.0	5.491697
28509	28509	2017-08-13	54	SEAFOOD	2.0	2.754819
28510	28510	2017-08-14	54	SEAFOOD	0.0	2.810651
28511	28511	2017-08-15	54	SEAFOOD	3.0	2.076917

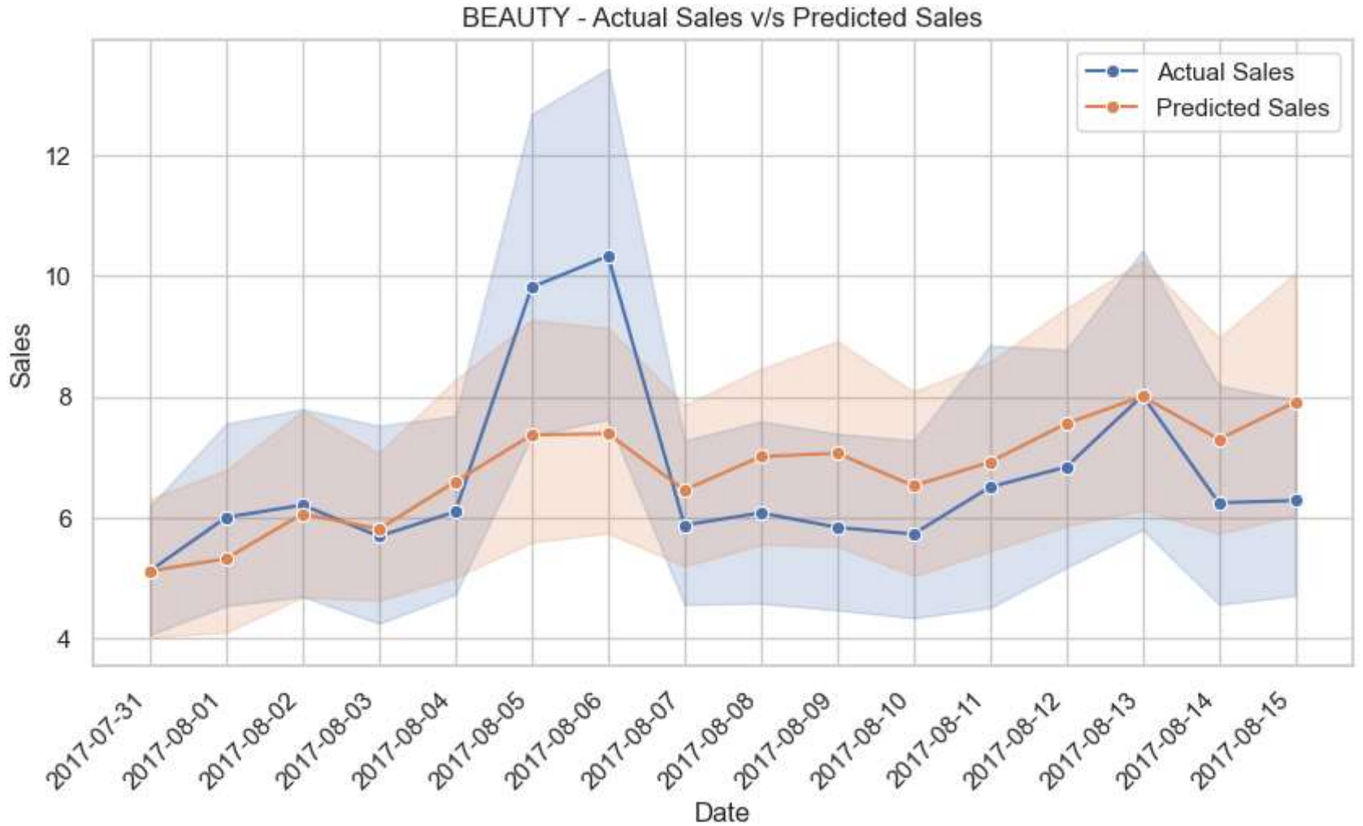
28512 rows × 6 columns

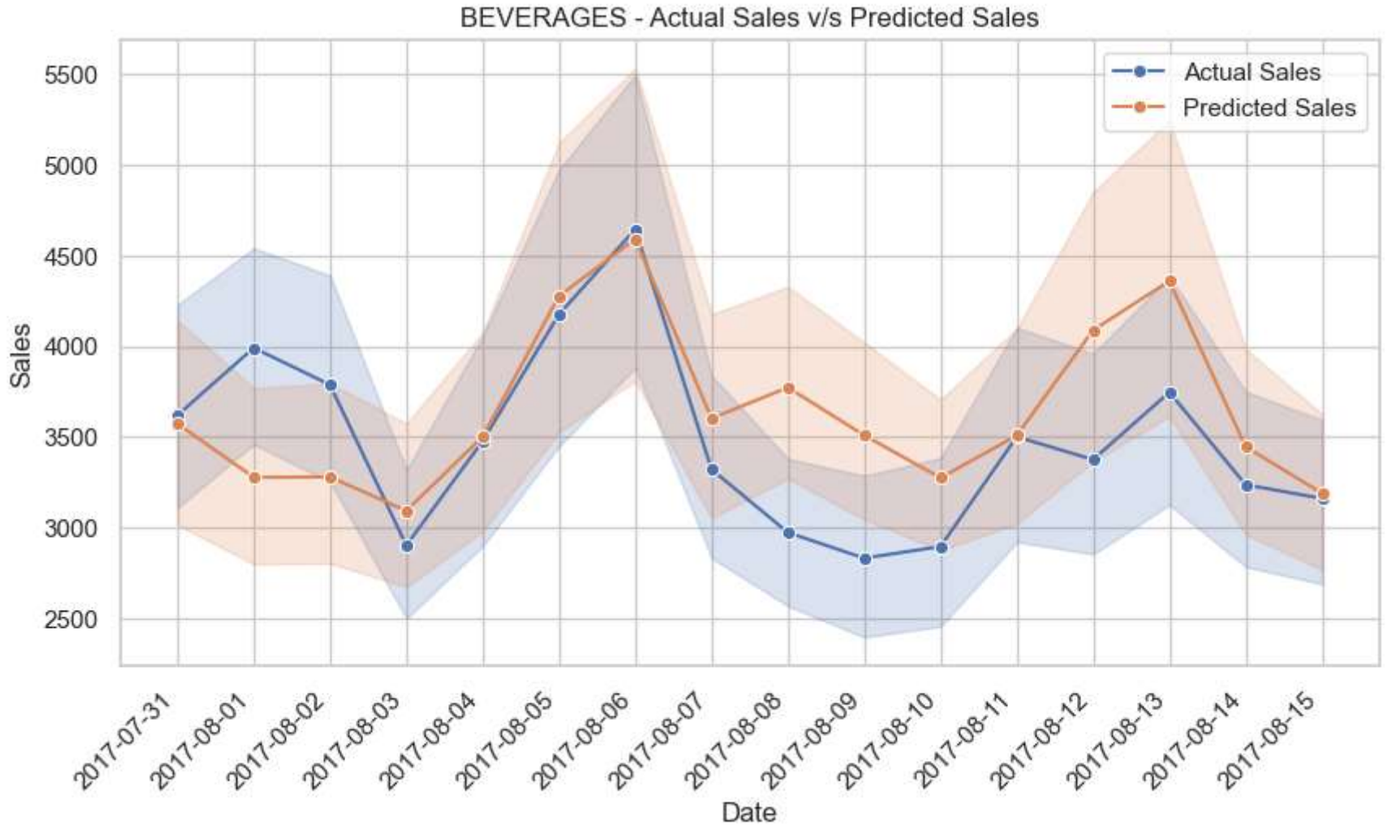
In [100…

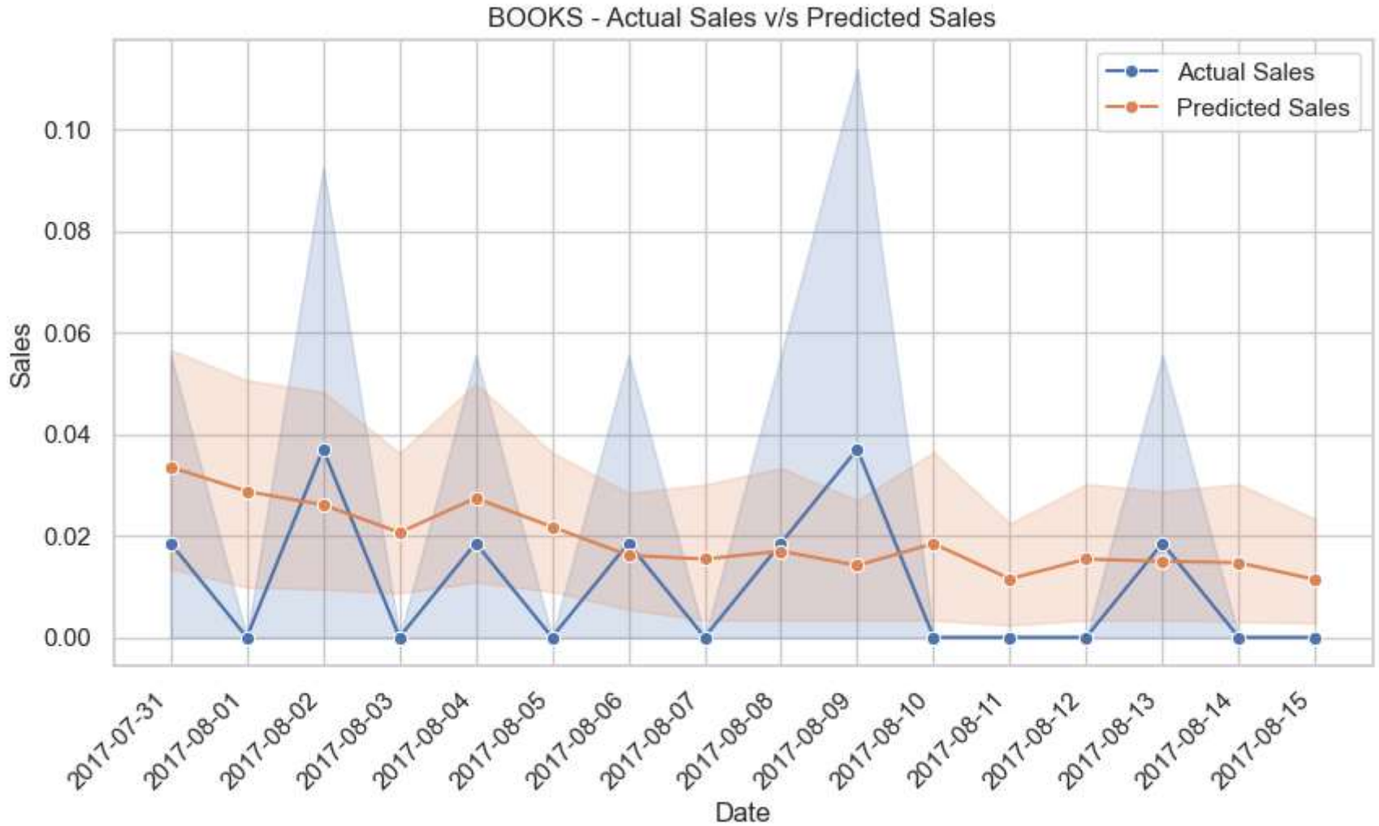
```
## Actual vs predicted sales for every product type - 16 days period
# Iterate over unique product types
for product_type in final['product_type'].unique():
    # Filter data for the current product type
    product_data = final[final['product_type'] == product_type]

    # Create a line plot for sales and predicted sales
    plt.figure(figsize=(10, 5))
    sns.lineplot(x='date', y='sales', data=product_data, label='Actual Sales', marker='o')
    sns.lineplot(x='date', y='predicted_sales', data=product_data, label='Predicted Sales', marker='o')

    # Set plot labels and title
    plt.xlabel('Date')
    plt.ylabel('Sales')
    plt.title(f'{product_type} - Actual Sales v/s Predicted Sales')

    # Rotate x-axis labels for better visibility
    plt.xticks(rotation=45, ha='right')

    # Show the legend
    plt.legend()

    # Show the plot
    plt.show()
```

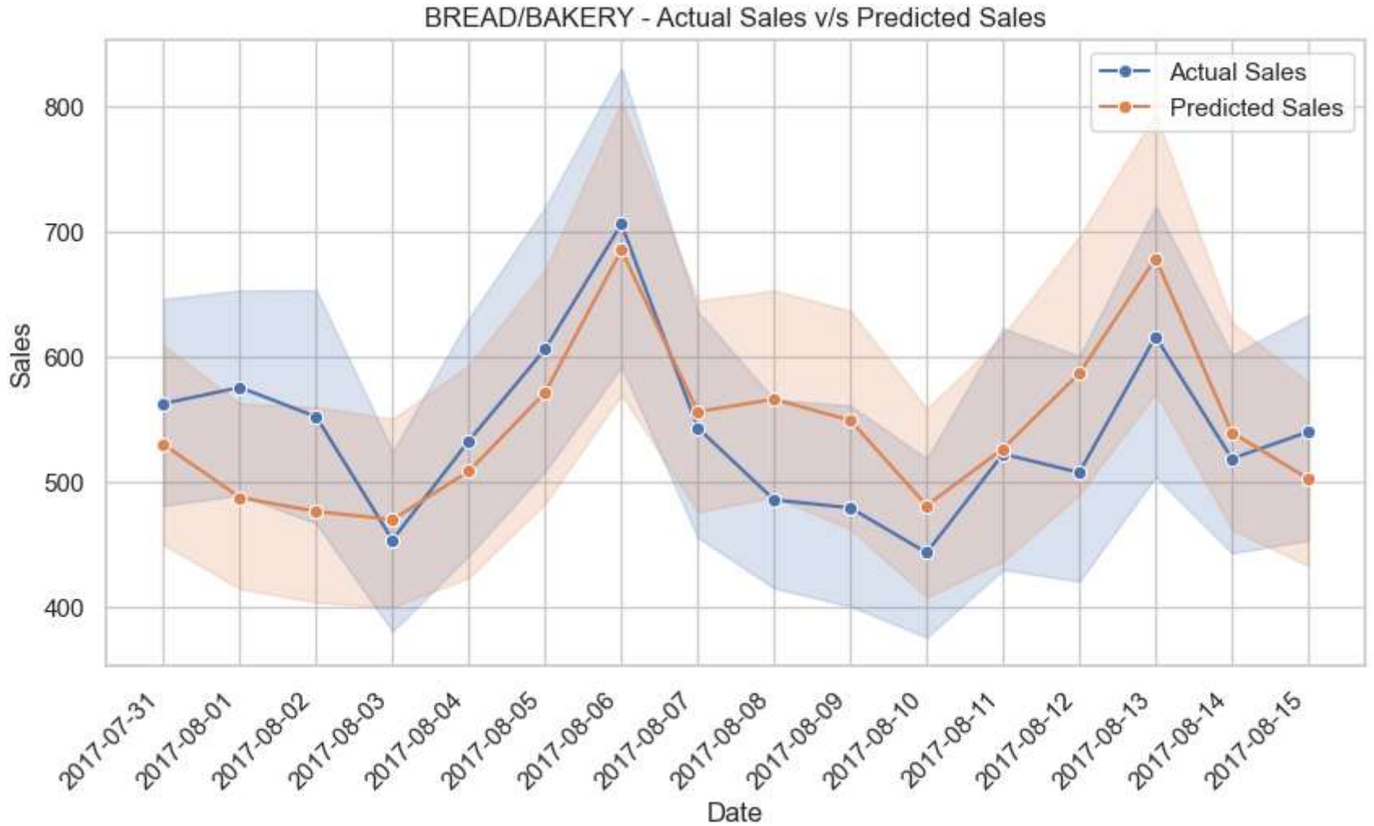

AUTOMOTIVE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
12
10
8
6
4
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

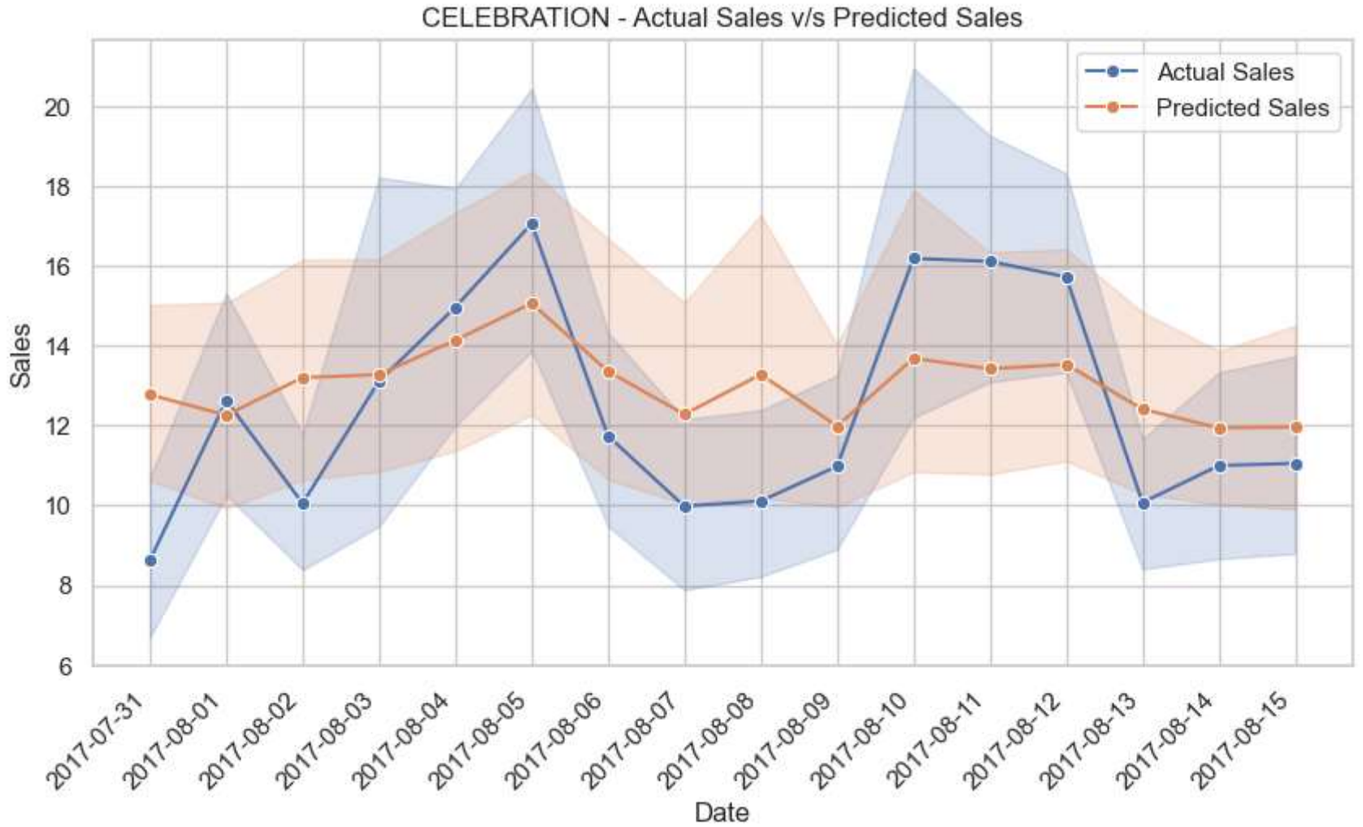

BABY CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
0.0
0.1
0.2
0.3
0.4
0.5
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

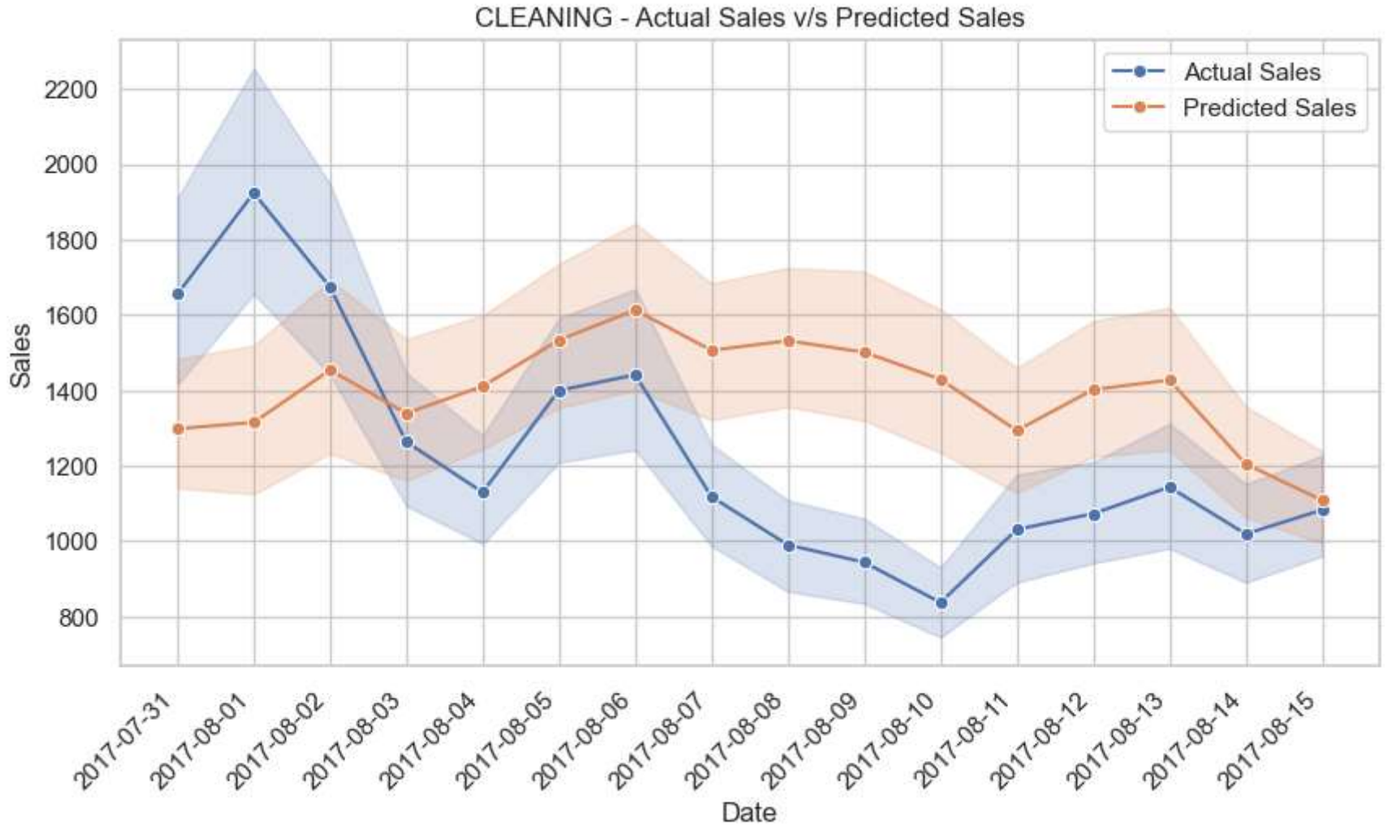

BEAUTY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
12
10
8
6
4
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

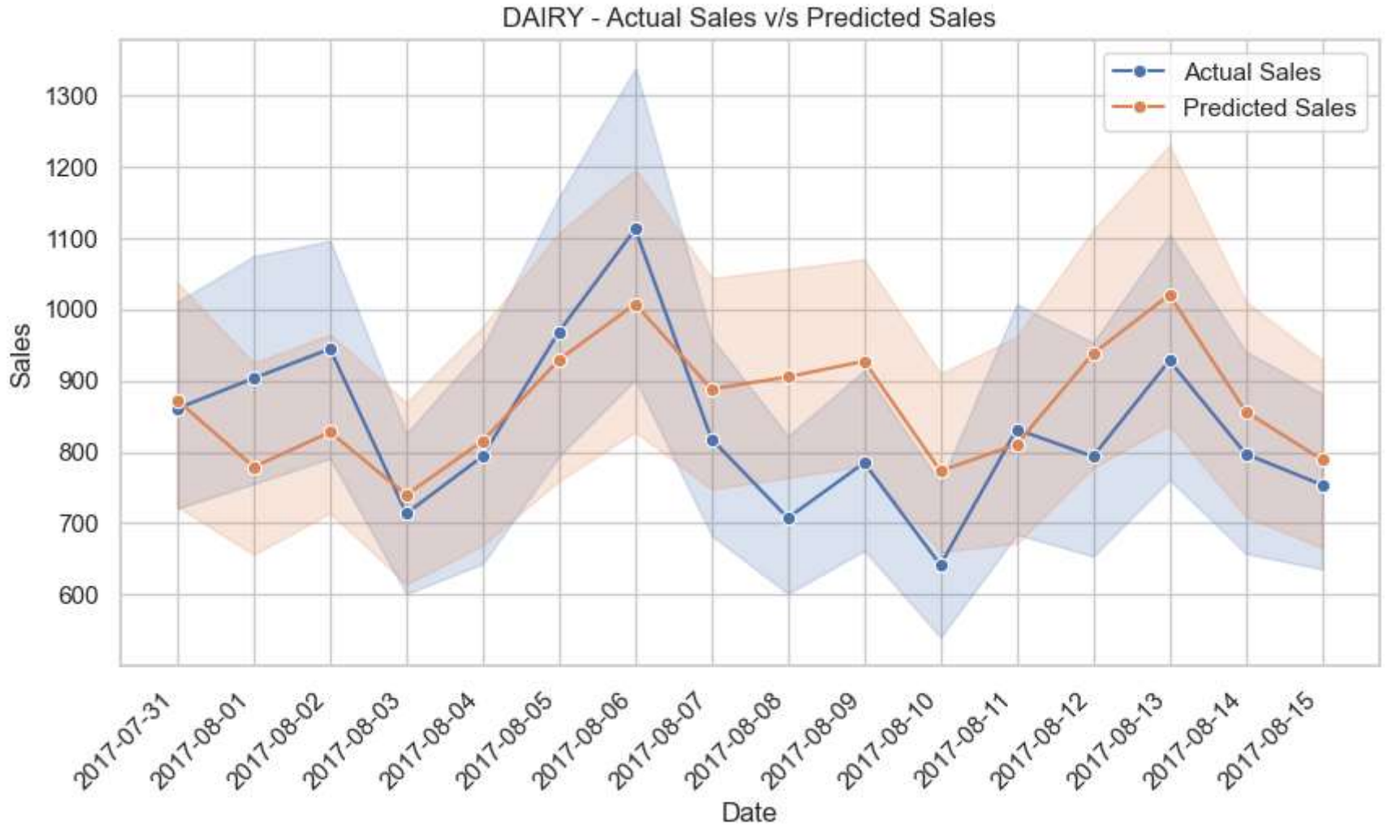

BEVERAGES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
5500
5000
4500
4000
3500
3000
2500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

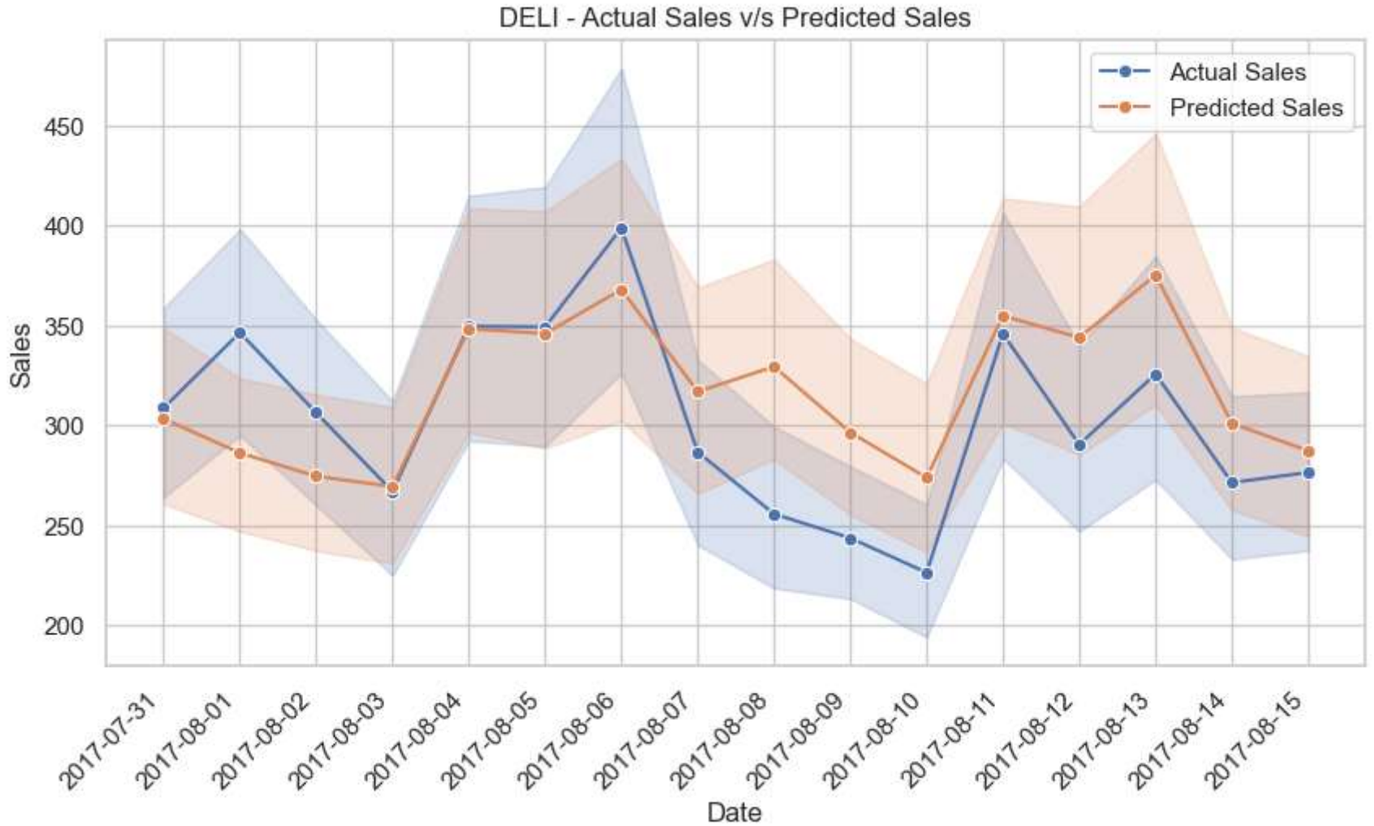

BOOKS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
0.10
0.08
0.06
0.04
0.02
0.00
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

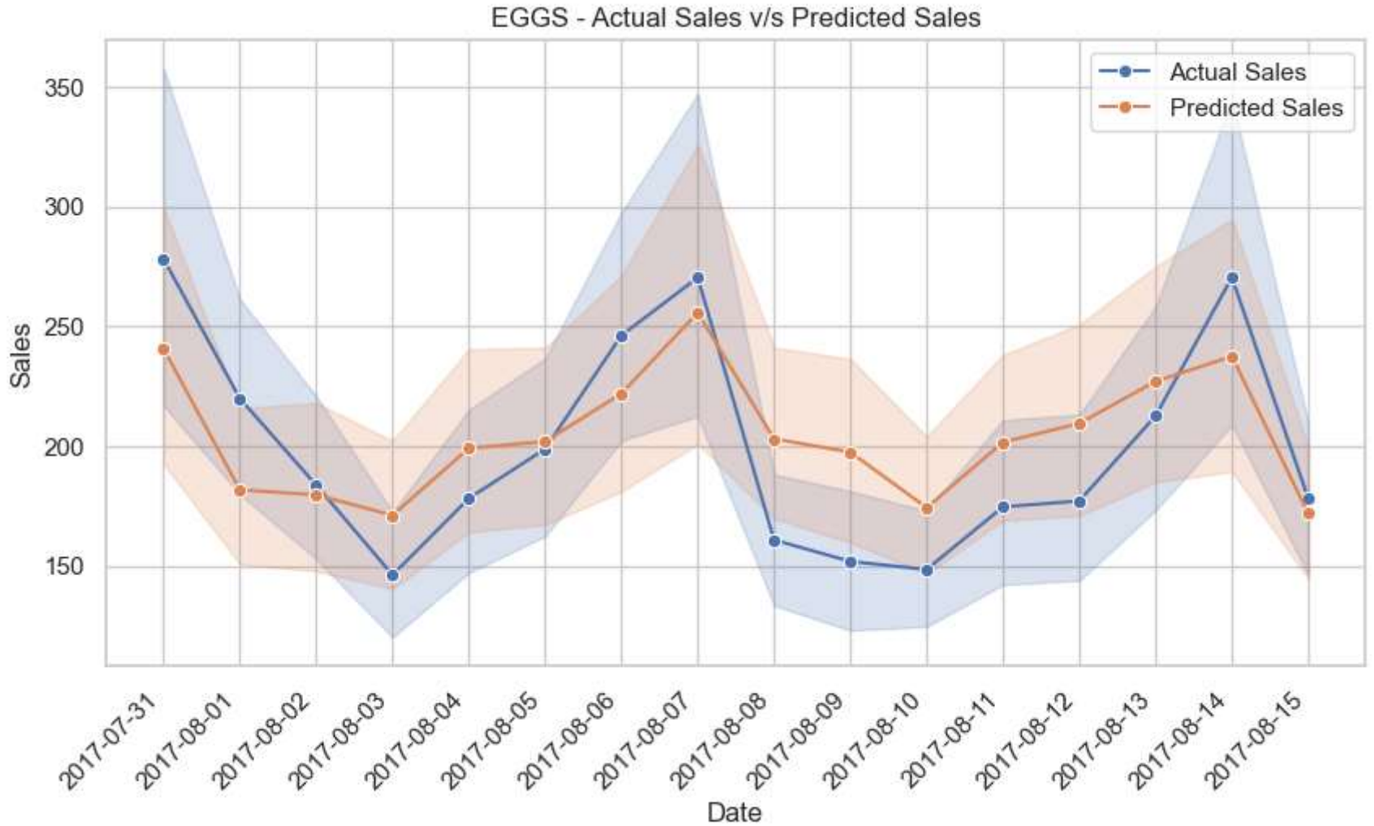

BREAD/BAKERY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
800
700
600
500
400
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

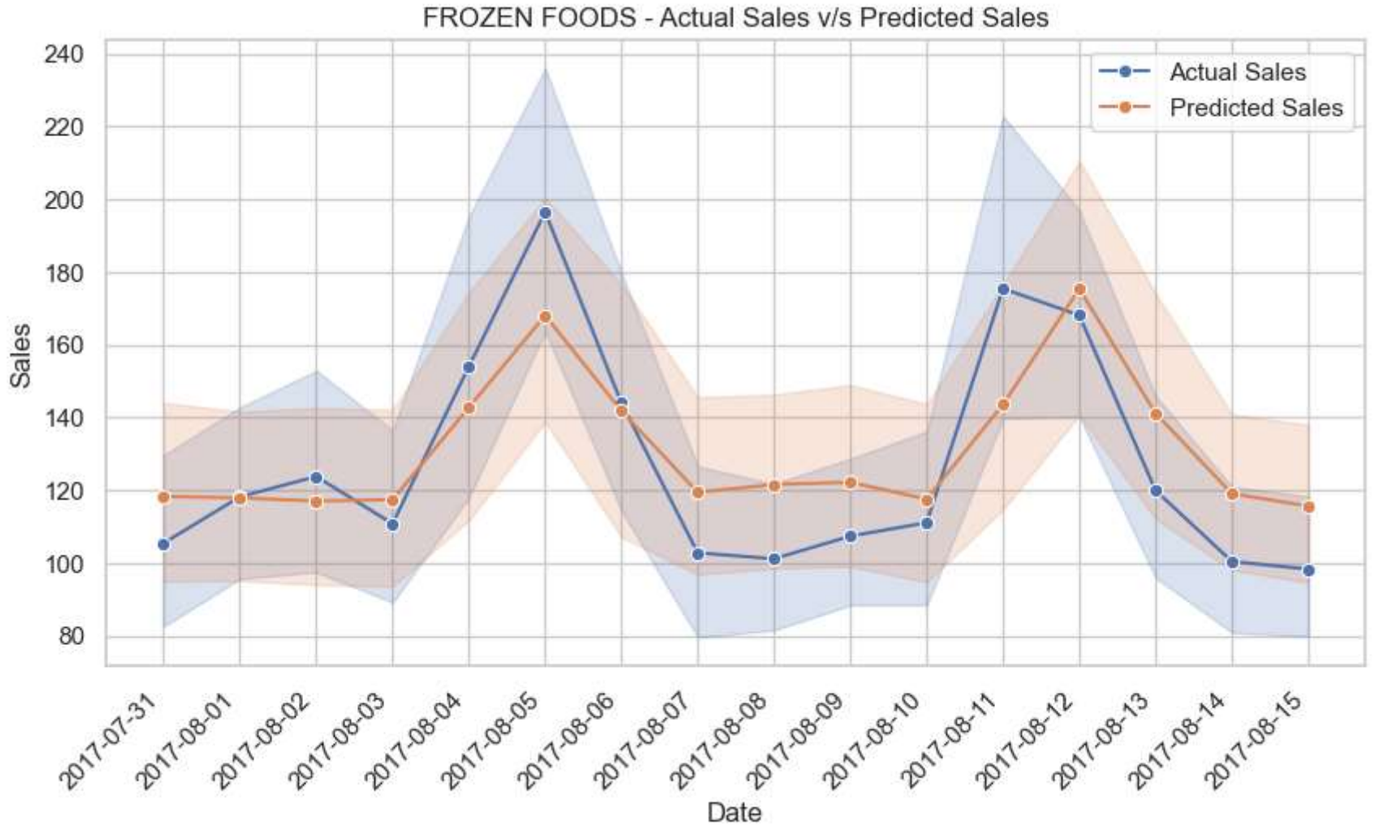

CELEBRATION - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
20
18
16
14
12
10
8
6
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

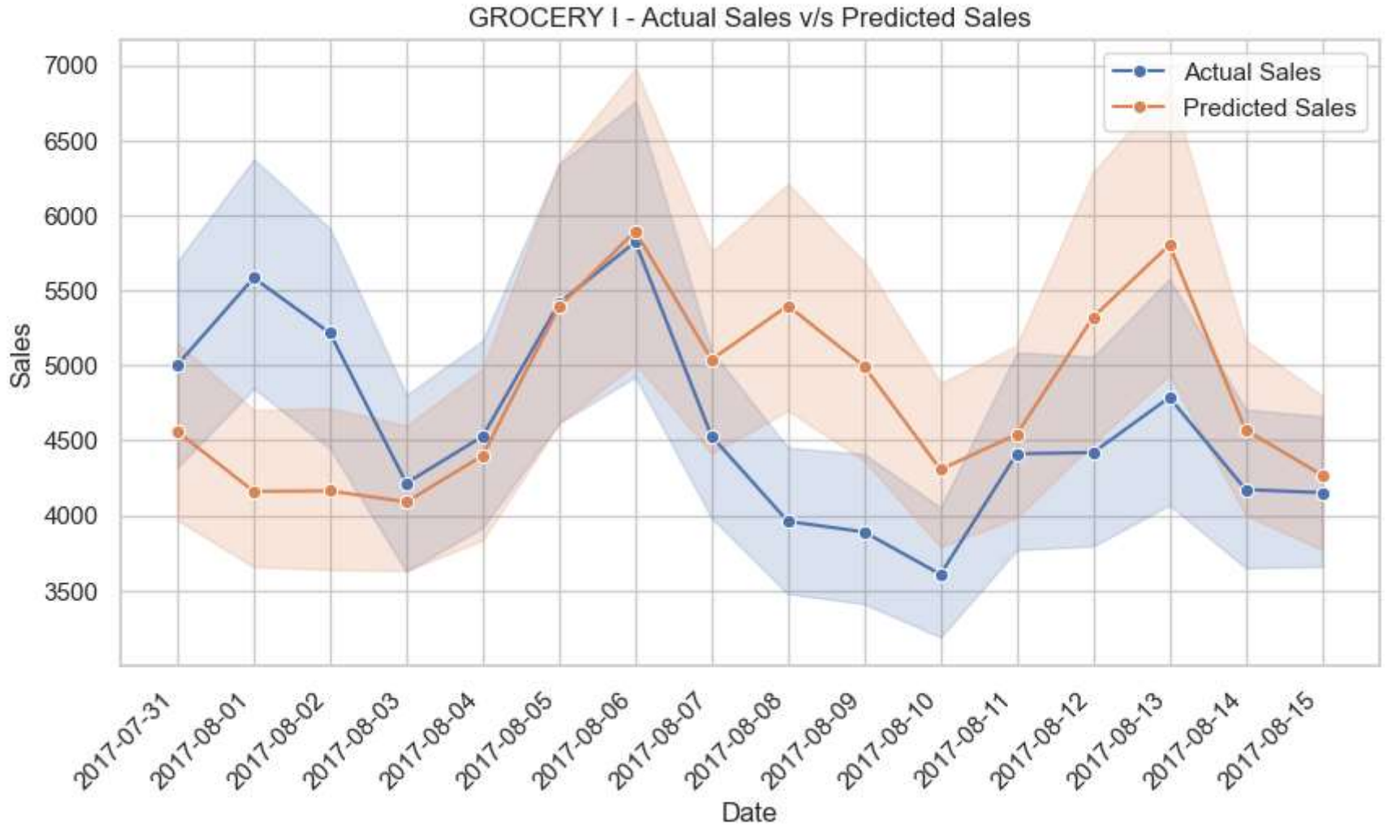

CLEANING - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
2200
2000
1800
1600
1400
1200
1000
800
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

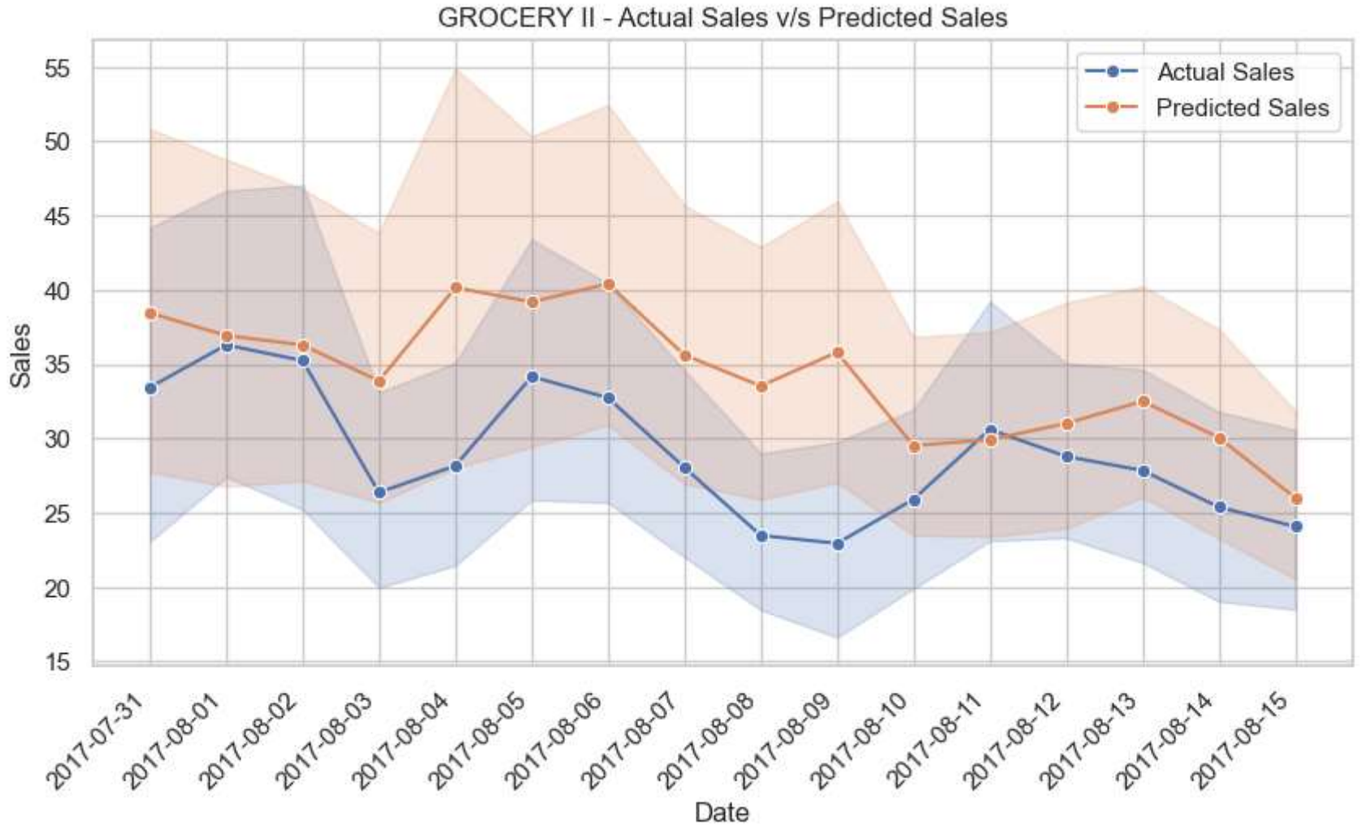

DAIRY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
1300
1200
1100
1000
900
800
700
600
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

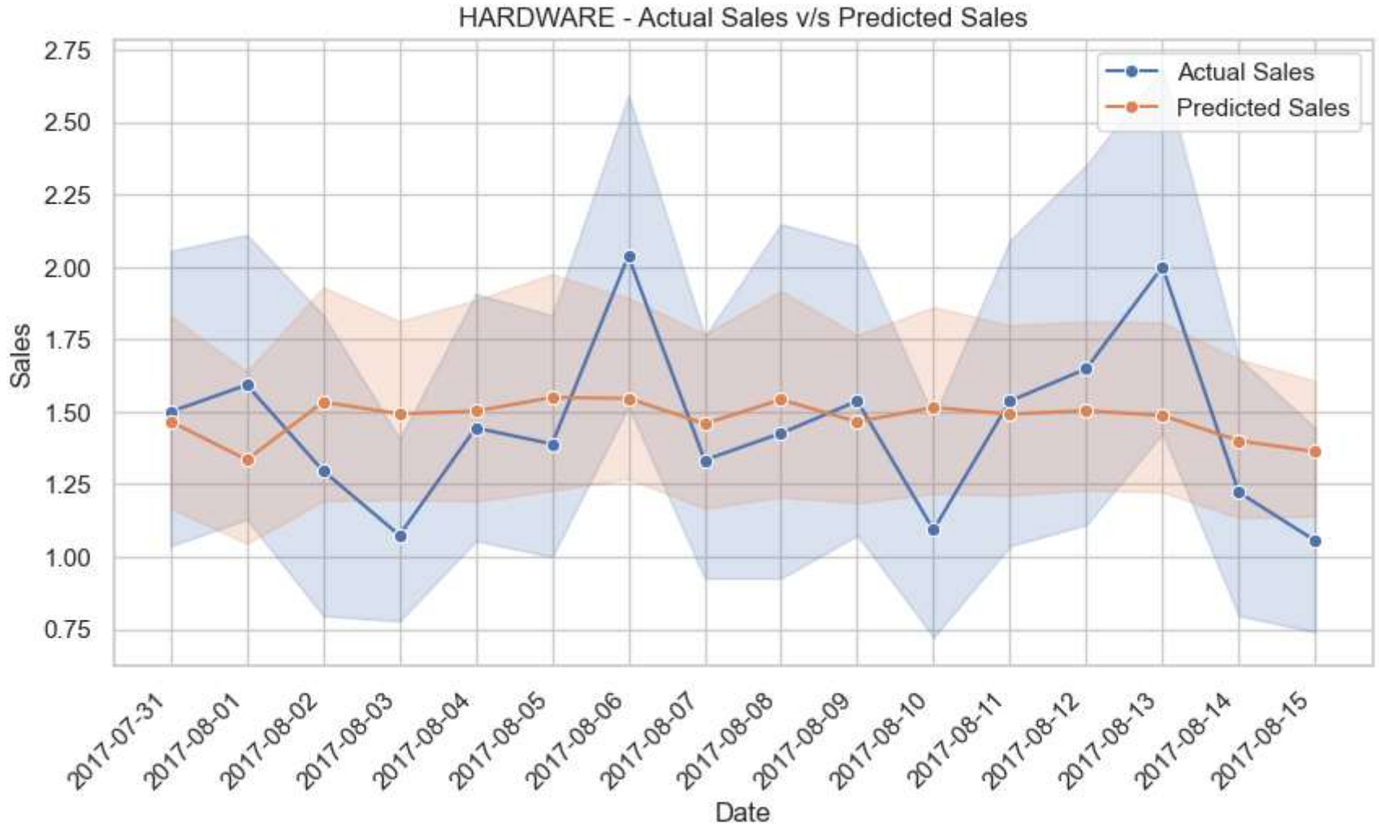

DELI - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
450
400
350
300
250
200
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

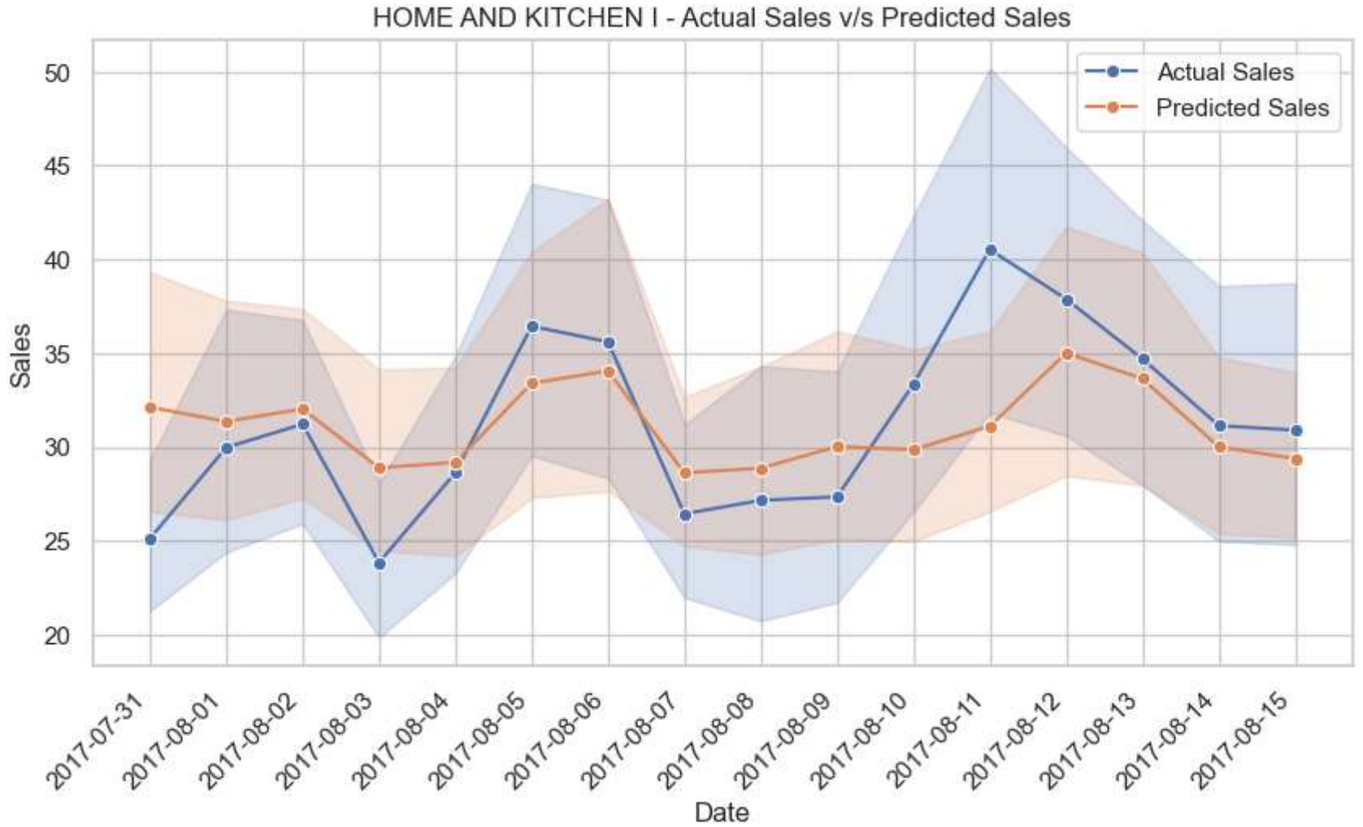

EGGS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
350
300
250
200
150
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

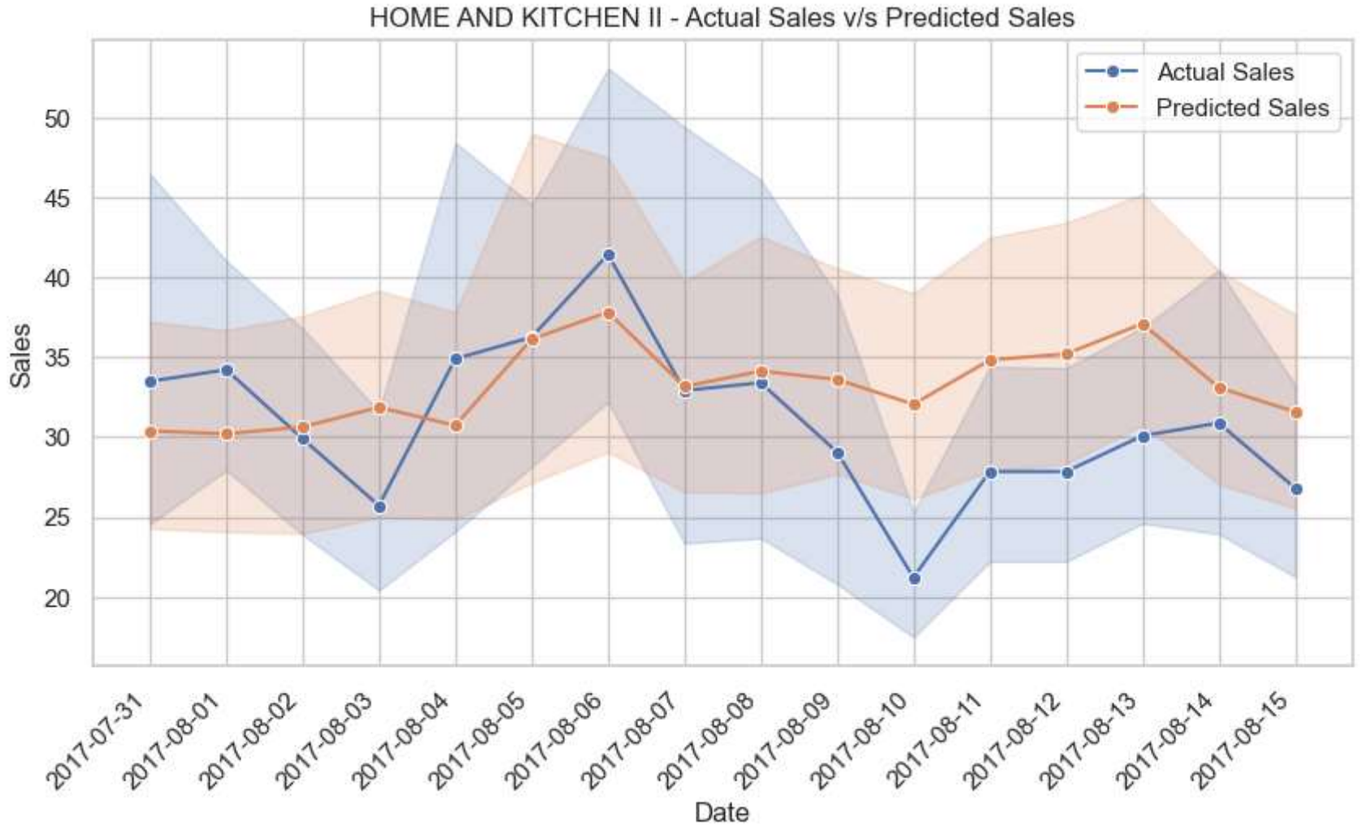

FROZEN FOODS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
240
220
200
180
160
140
120
100
80
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

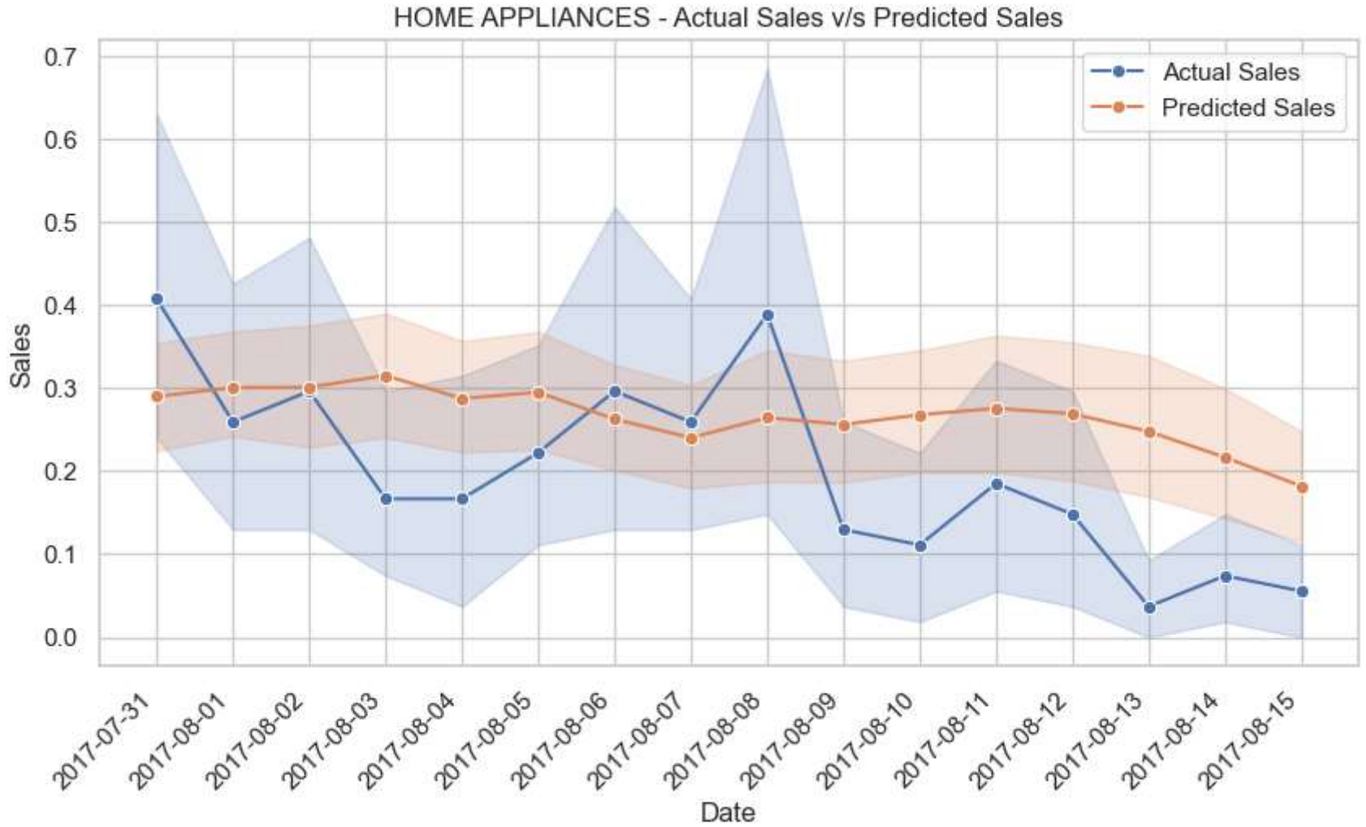

GROCERY I - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
7000
6500
6000
5500
5000
4500
4000
3500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

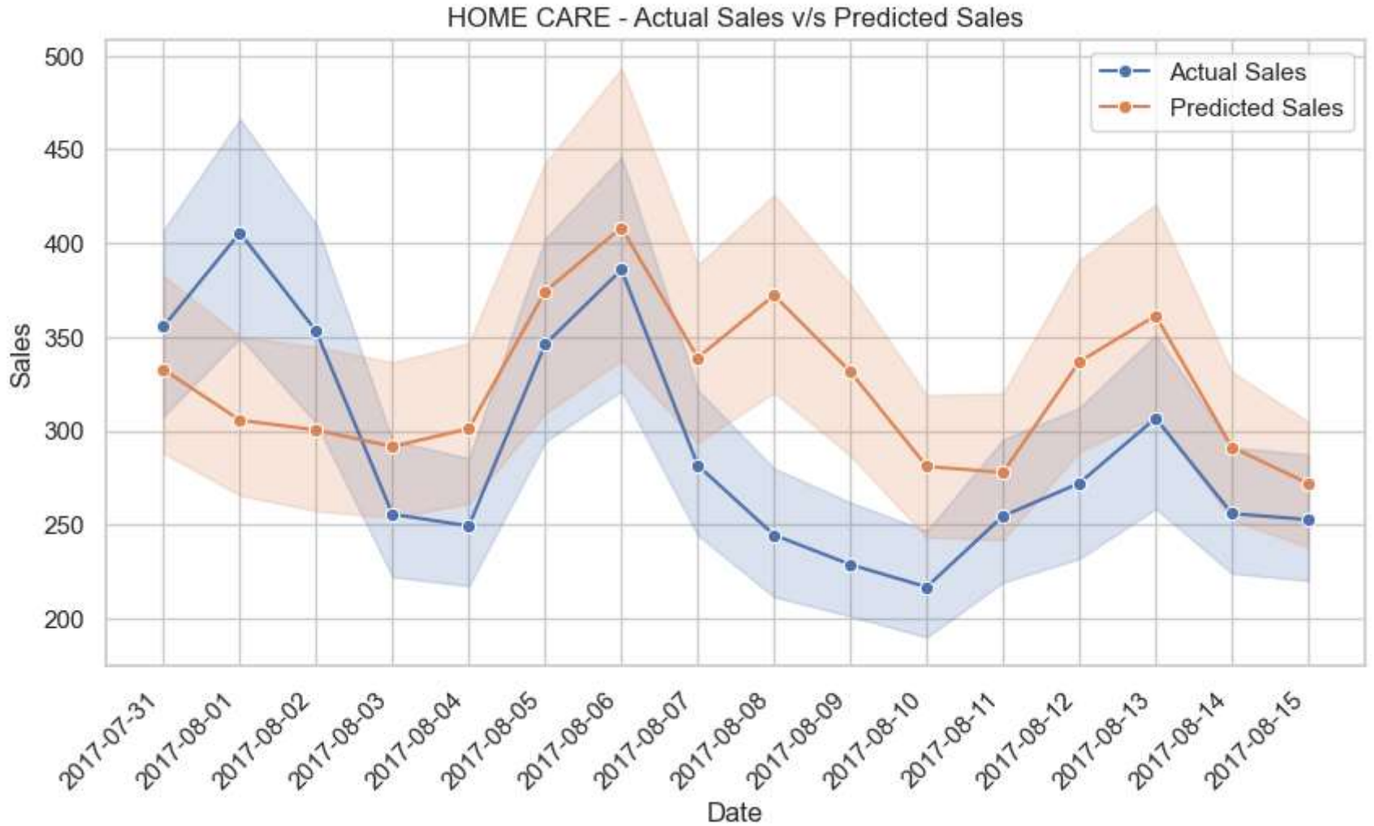

GROCERY II - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
55
50
45
40
35
30
25
20
15
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

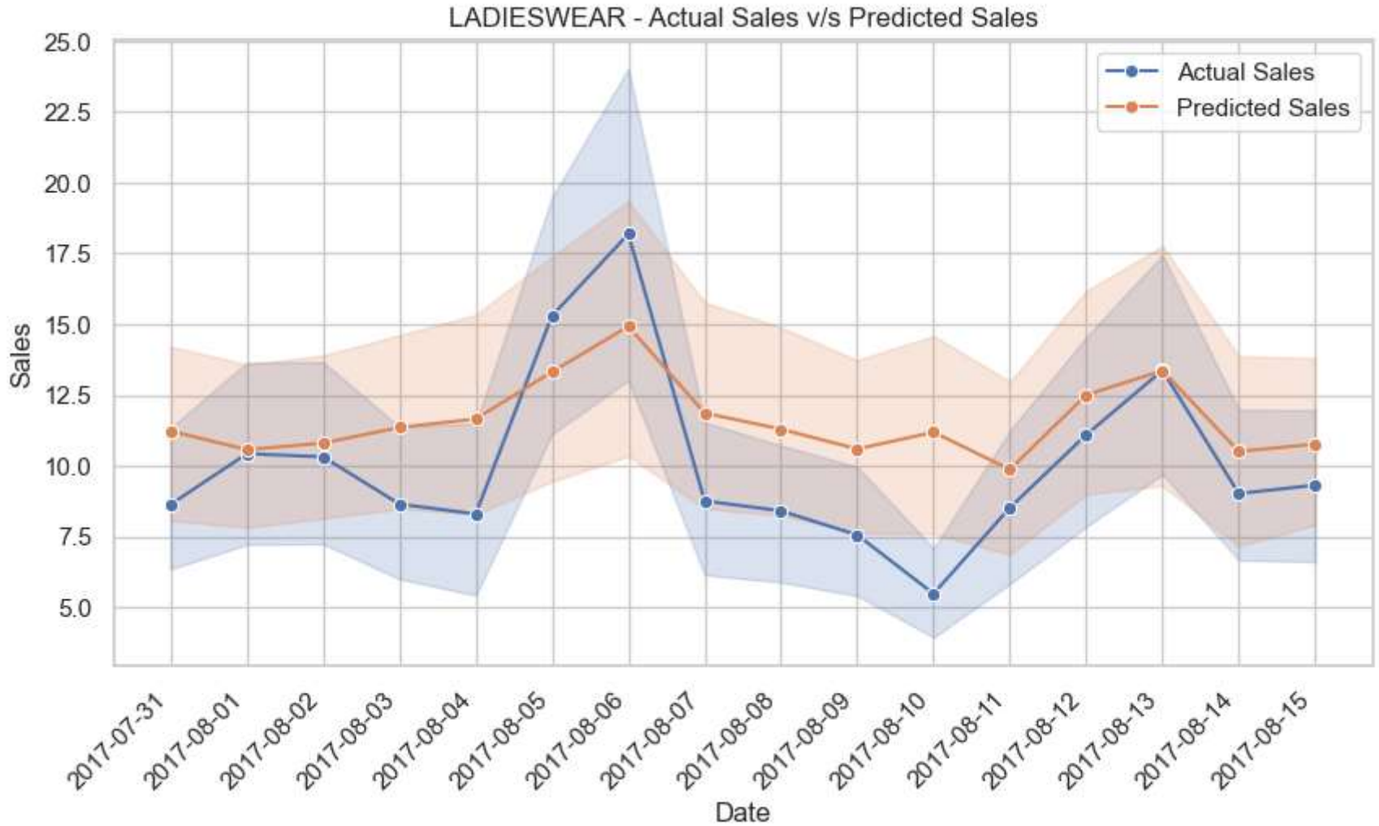

HARDWARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

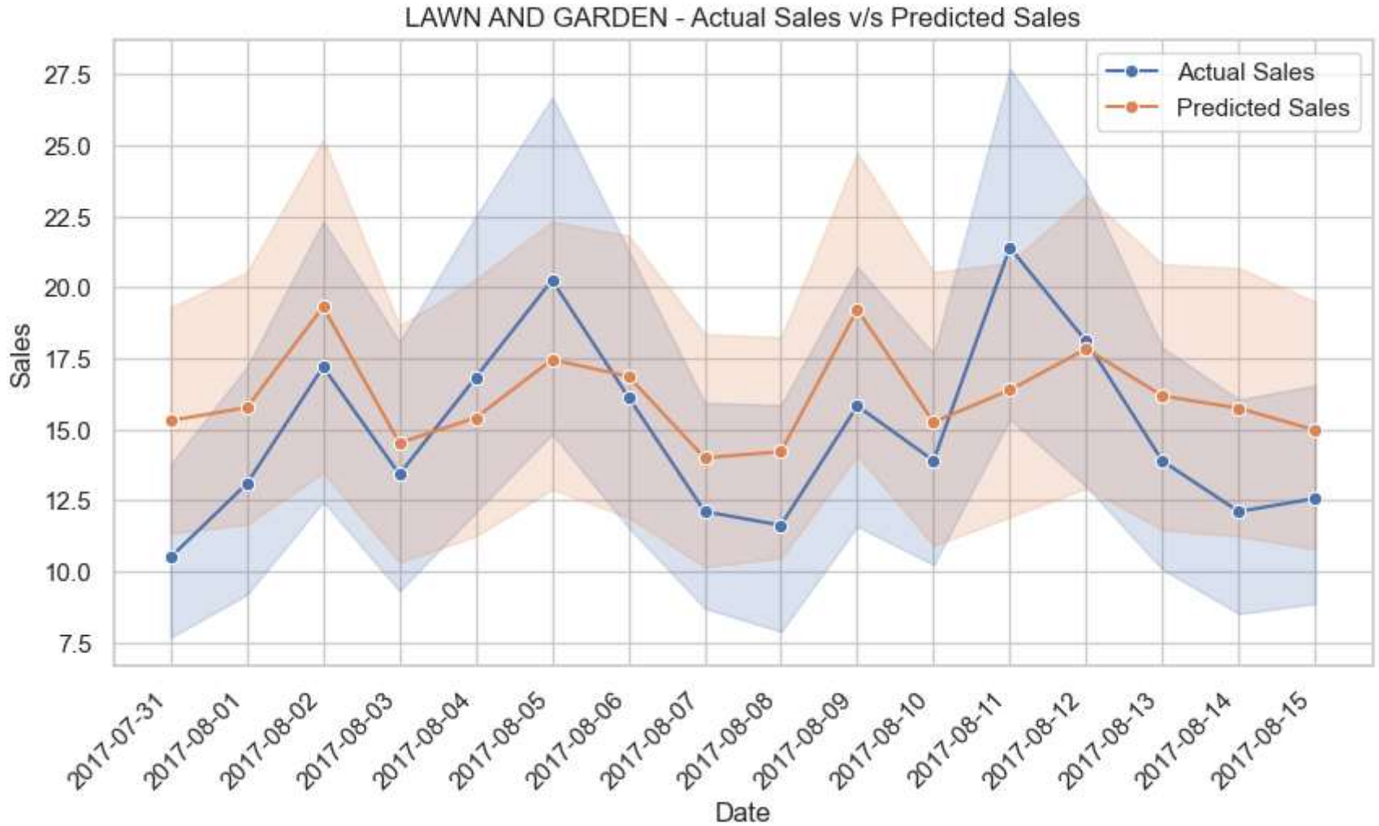

HOME AND KITCHEN I - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
50
45
40
35
30
25
20
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

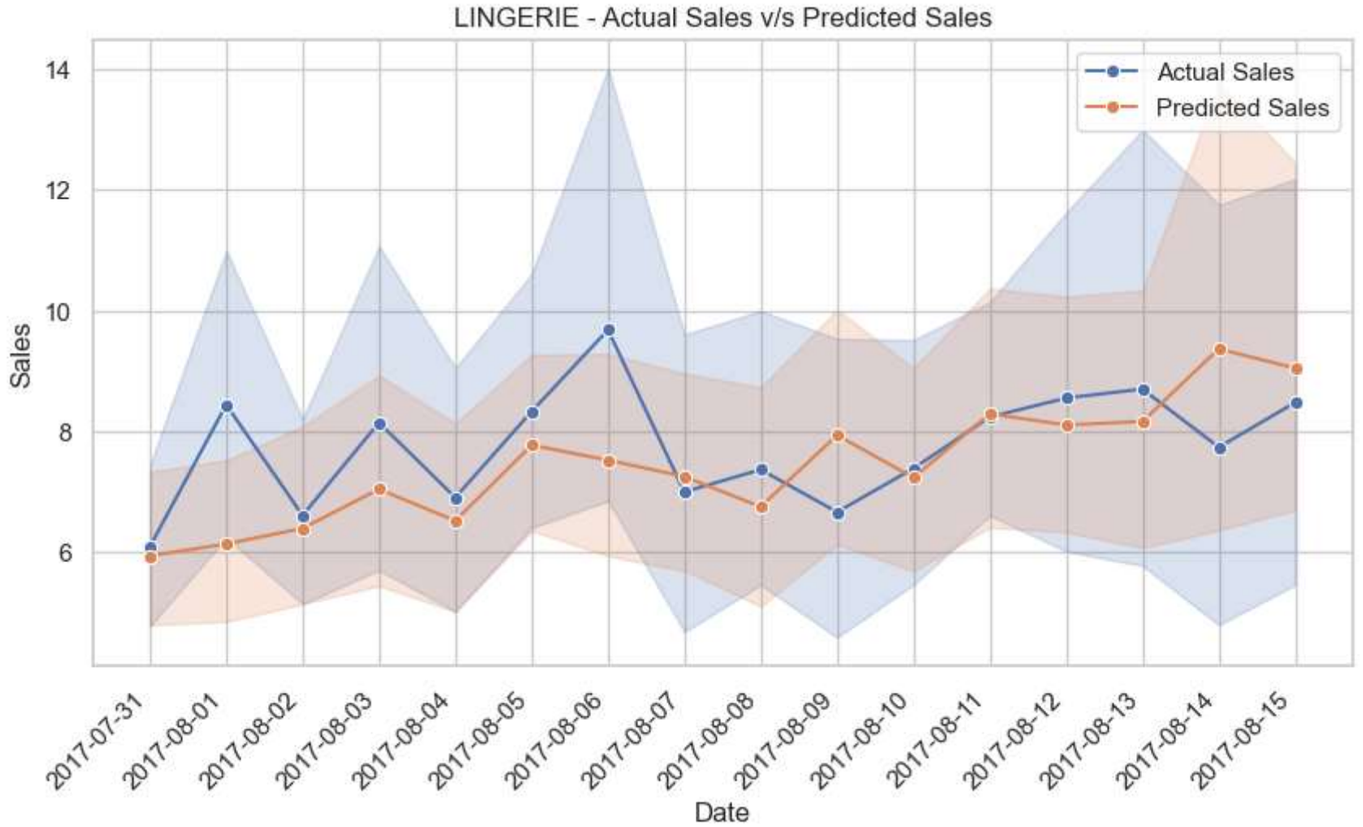

HOME AND KITCHEN II - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
50
45
40
35
30
25
20
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

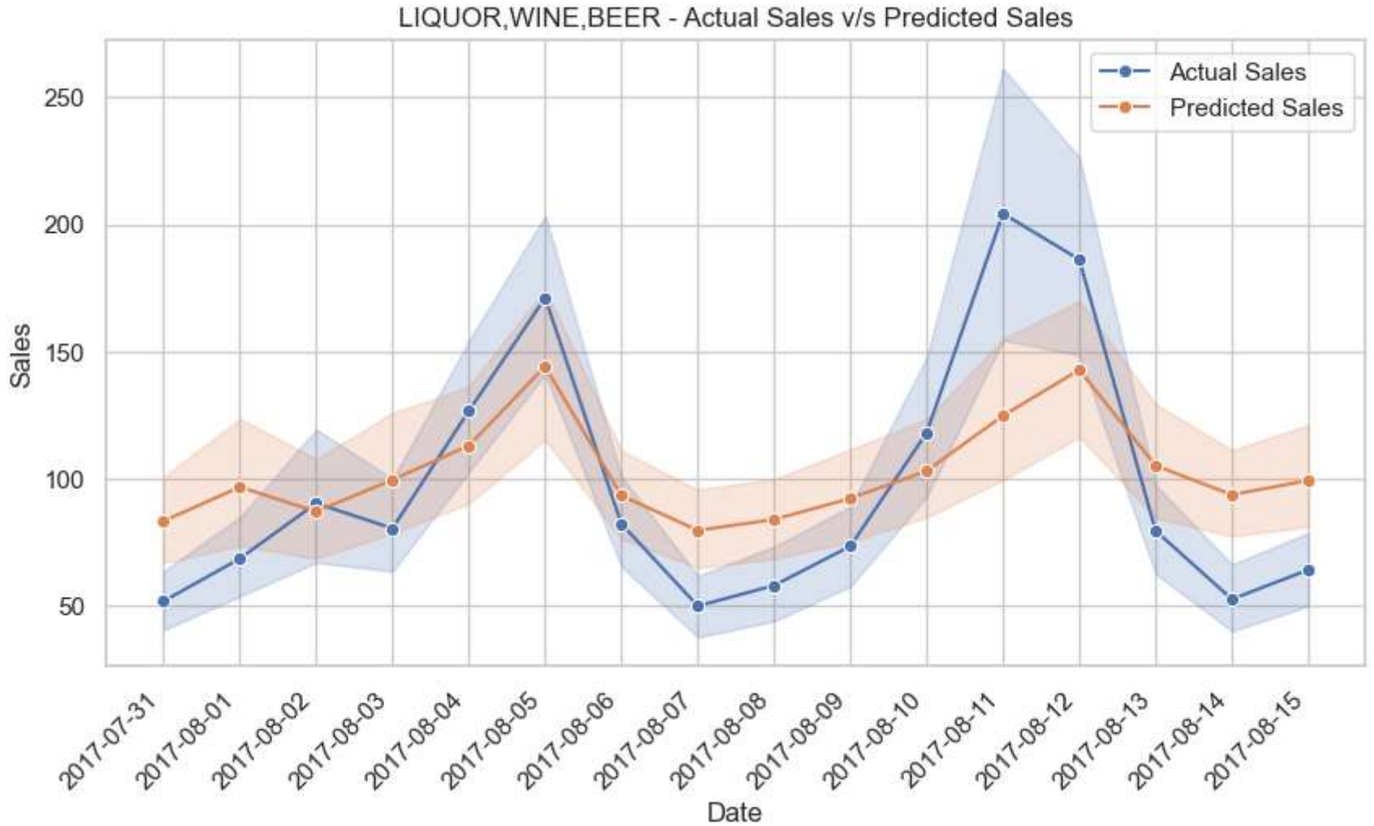

HOME APPLIANCES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

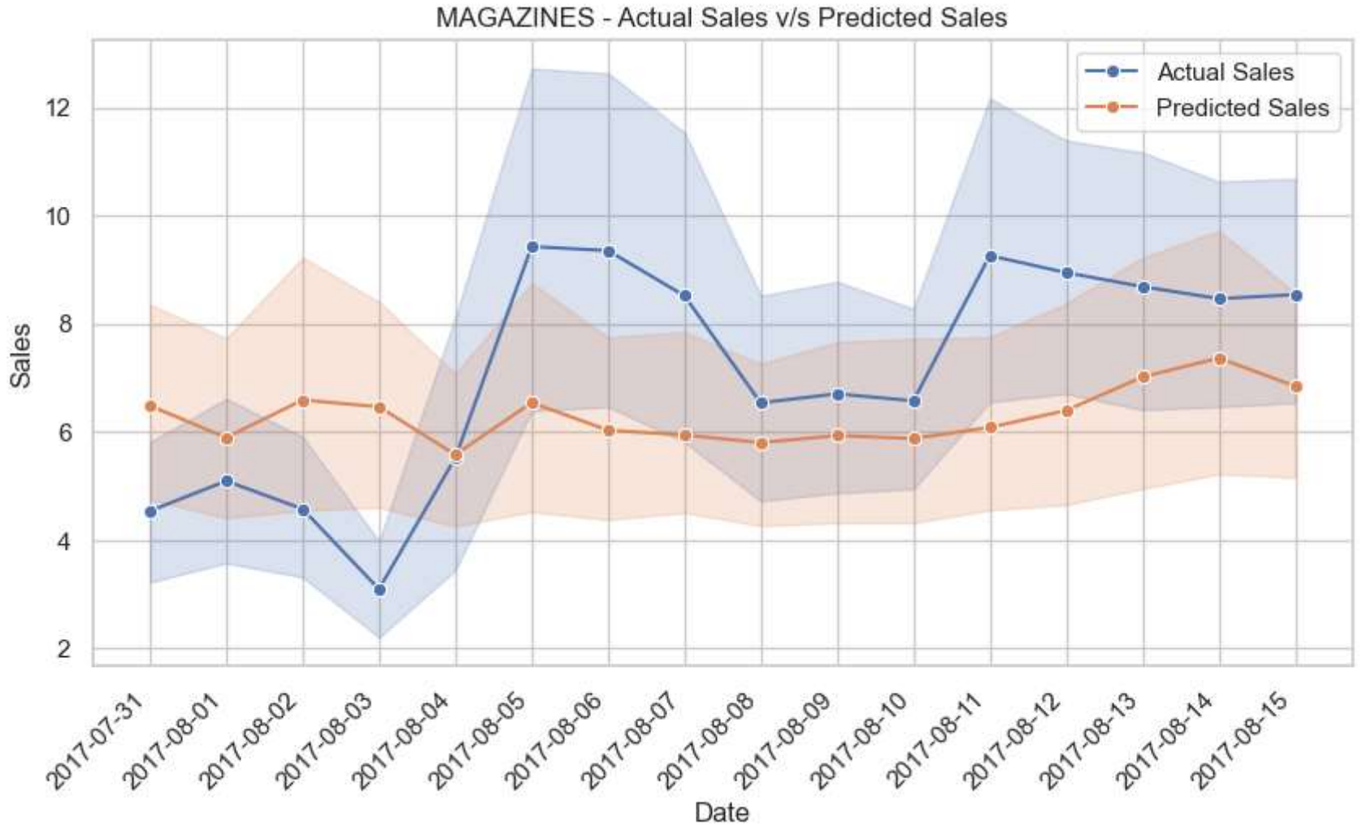

HOME CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
500
450
400
350
300
250
200
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

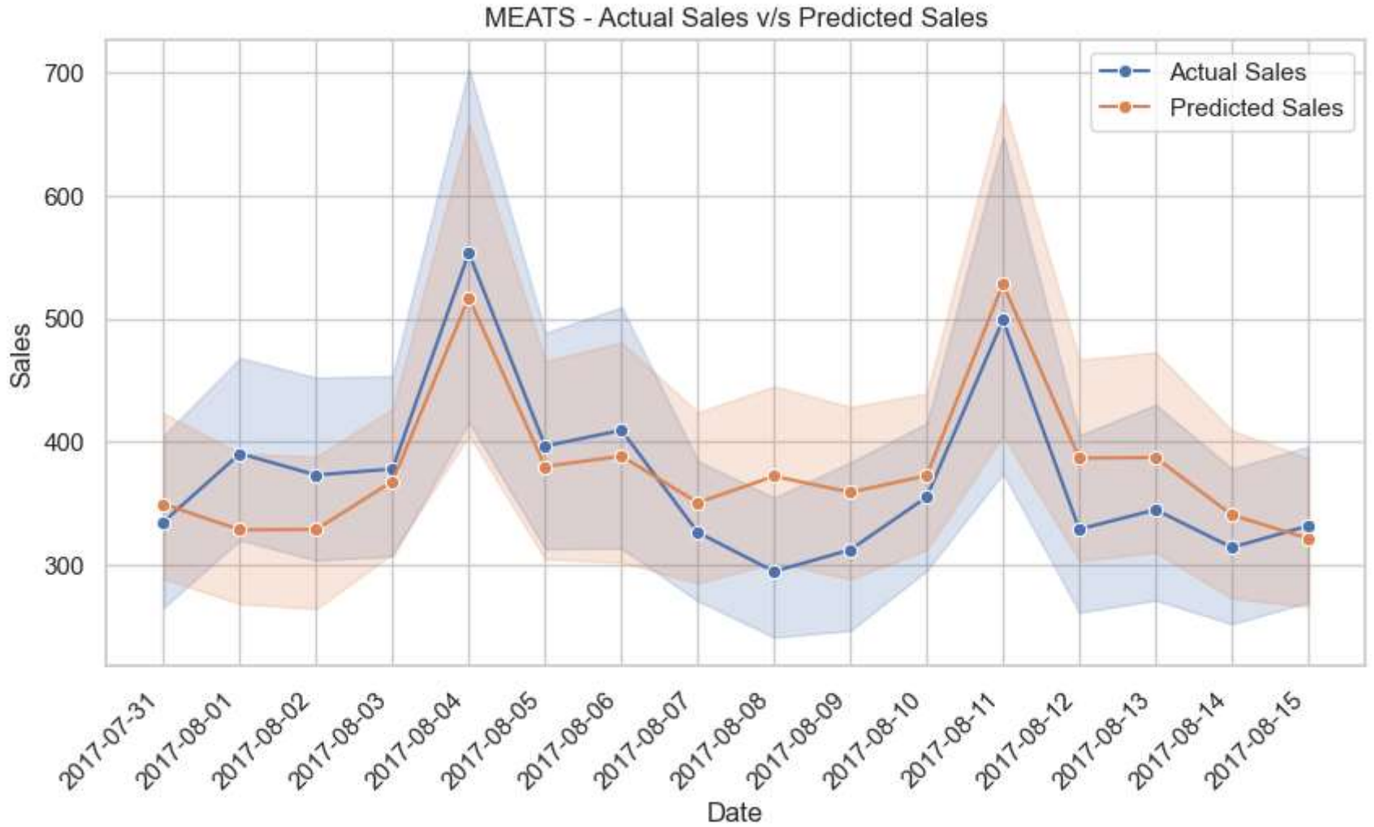

LADIESWEAR - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

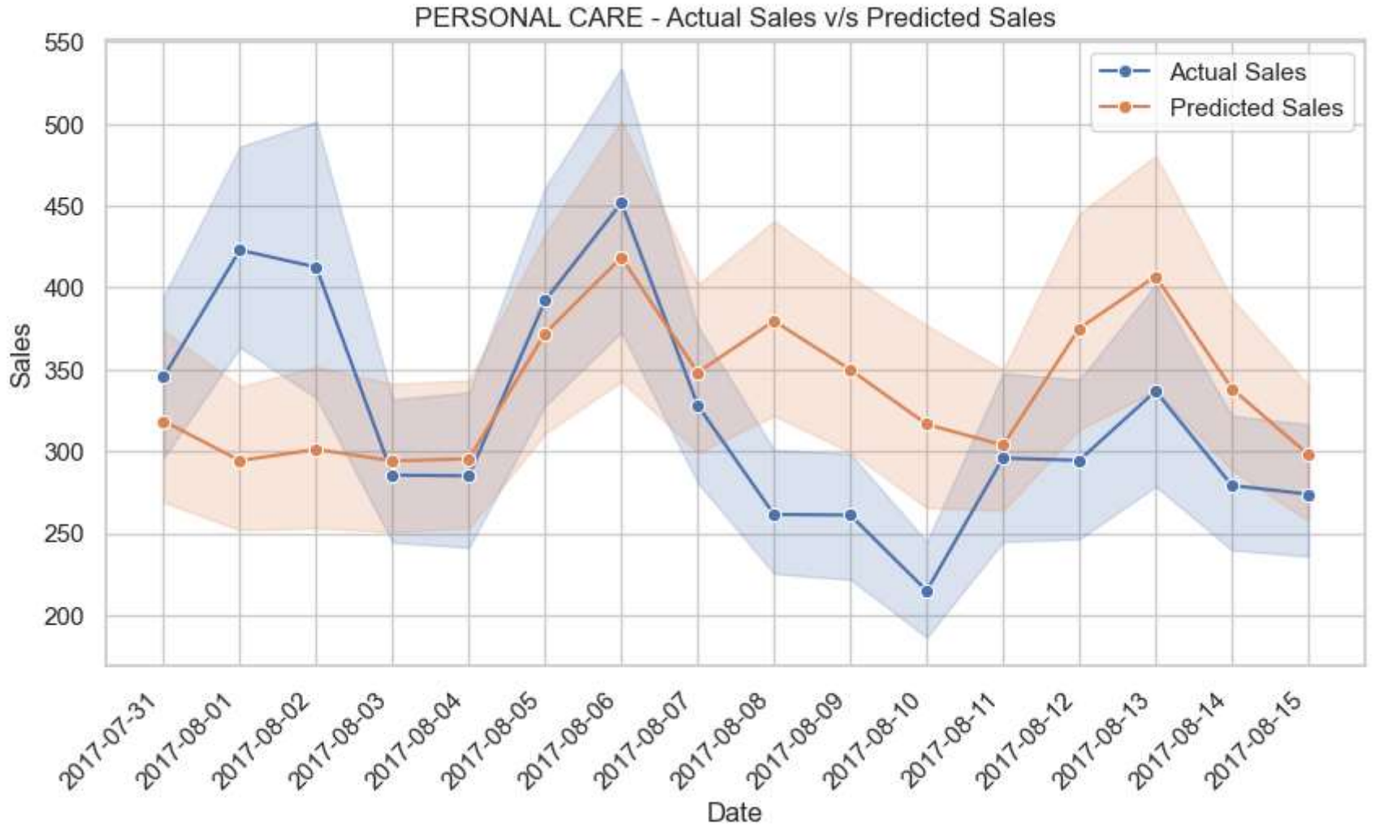

LAWN AND GARDEN - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

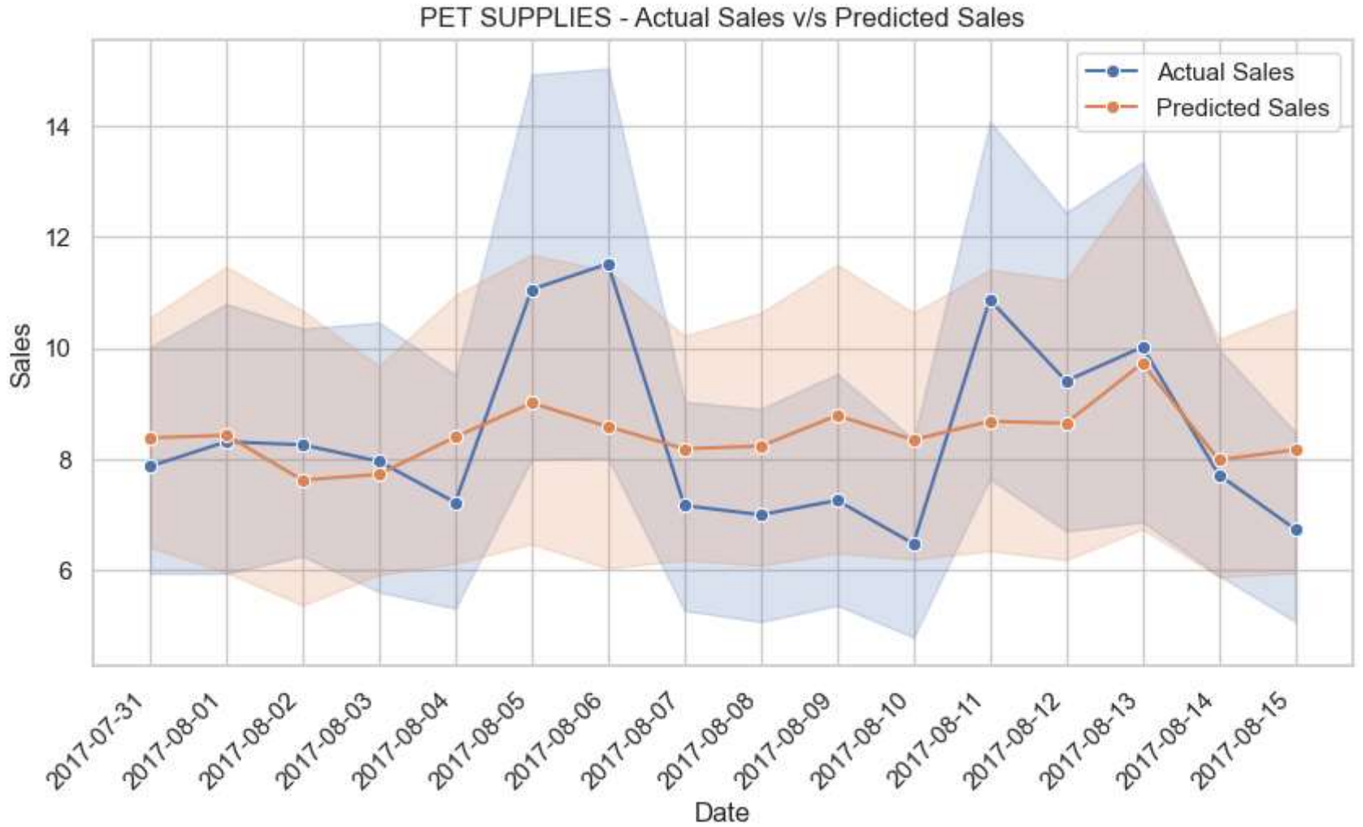

LINGERIE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
14
12
10
8
6
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

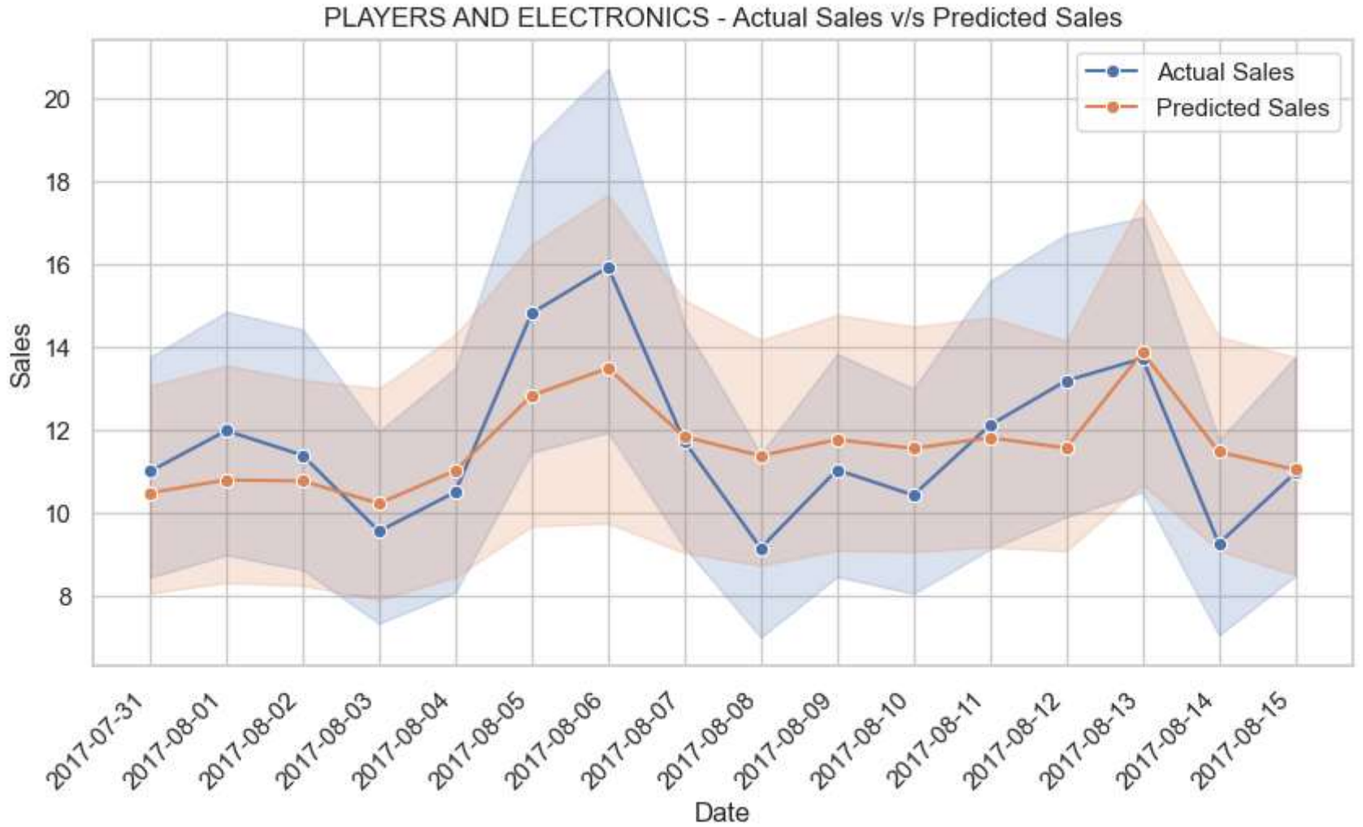

LIQUOR,WINE,BEER - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
250
200
150
100
50
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

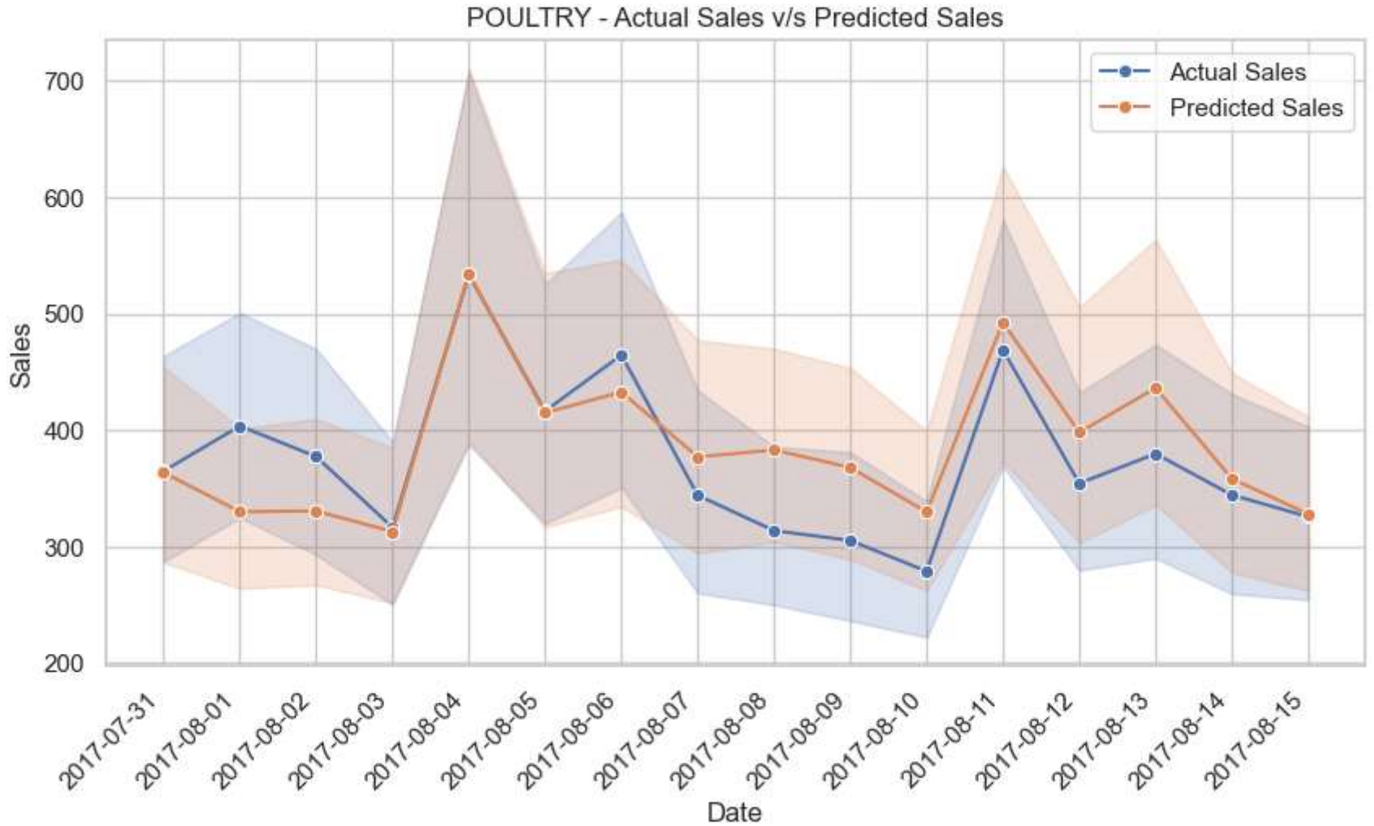

MAGAZINES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
12
10
8
6
4
2
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

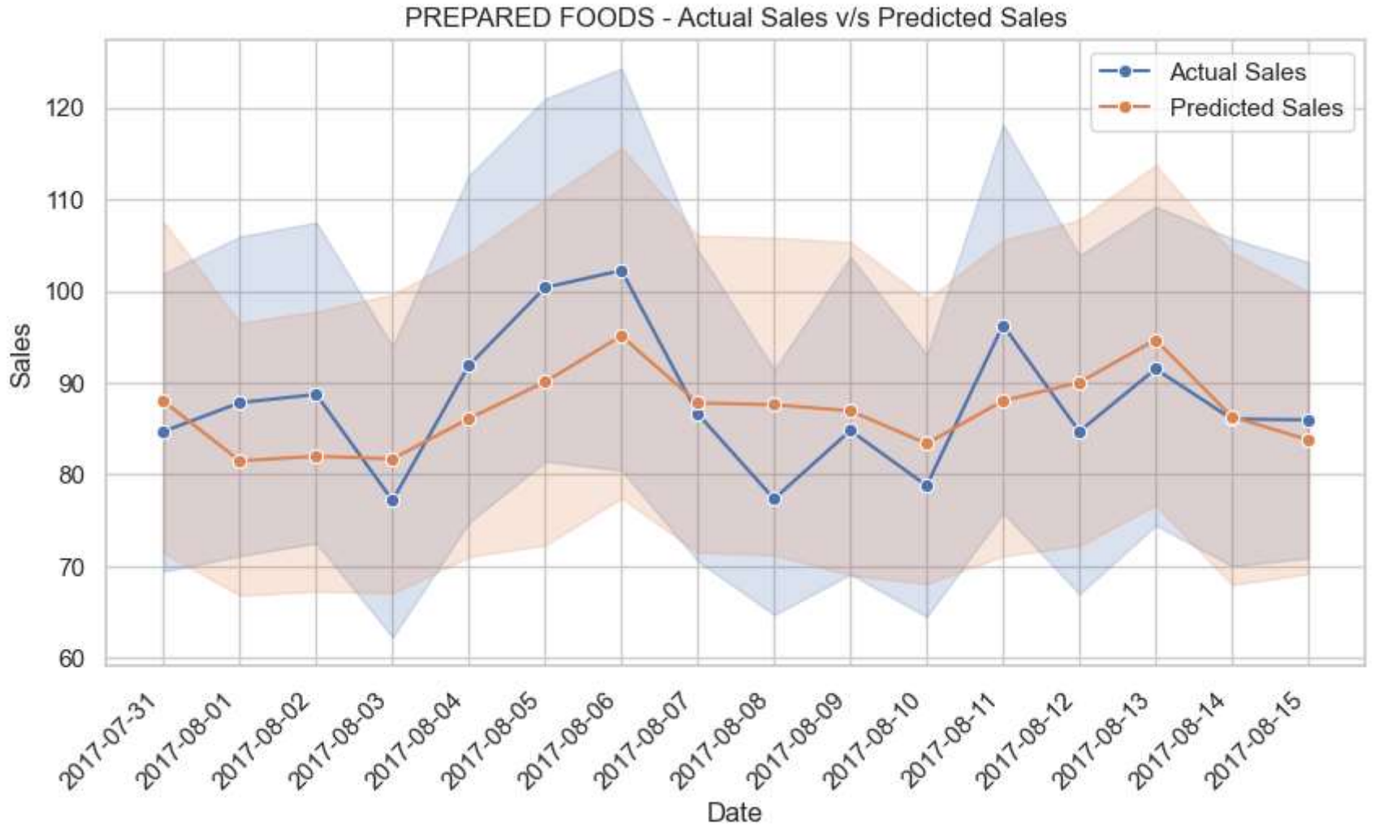

MEATS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
700
600
500
400
300
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

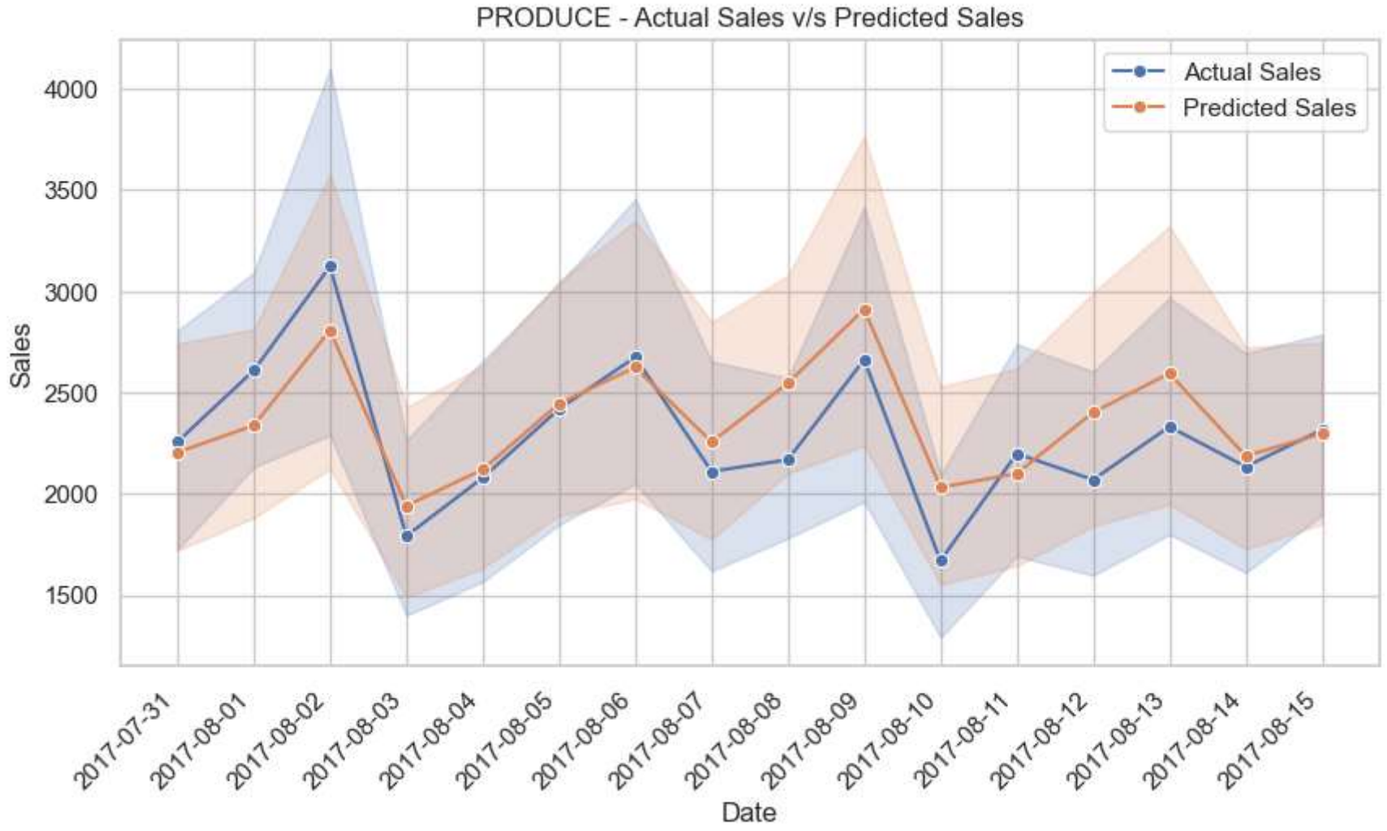

PERSONAL CARE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
550
500
450
400
350
300
250
200
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

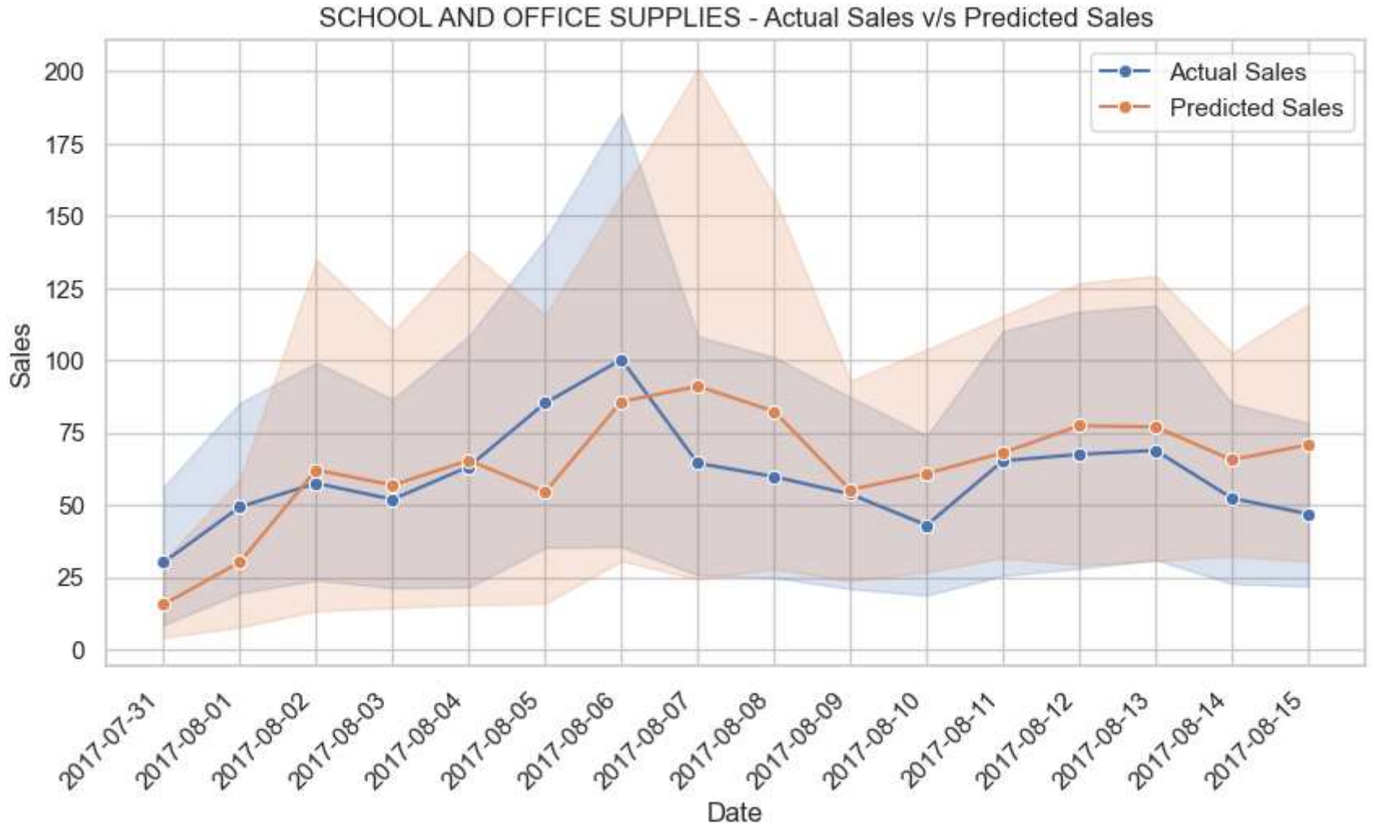

PET SUPPLIES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
14
12
10
8
6
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date

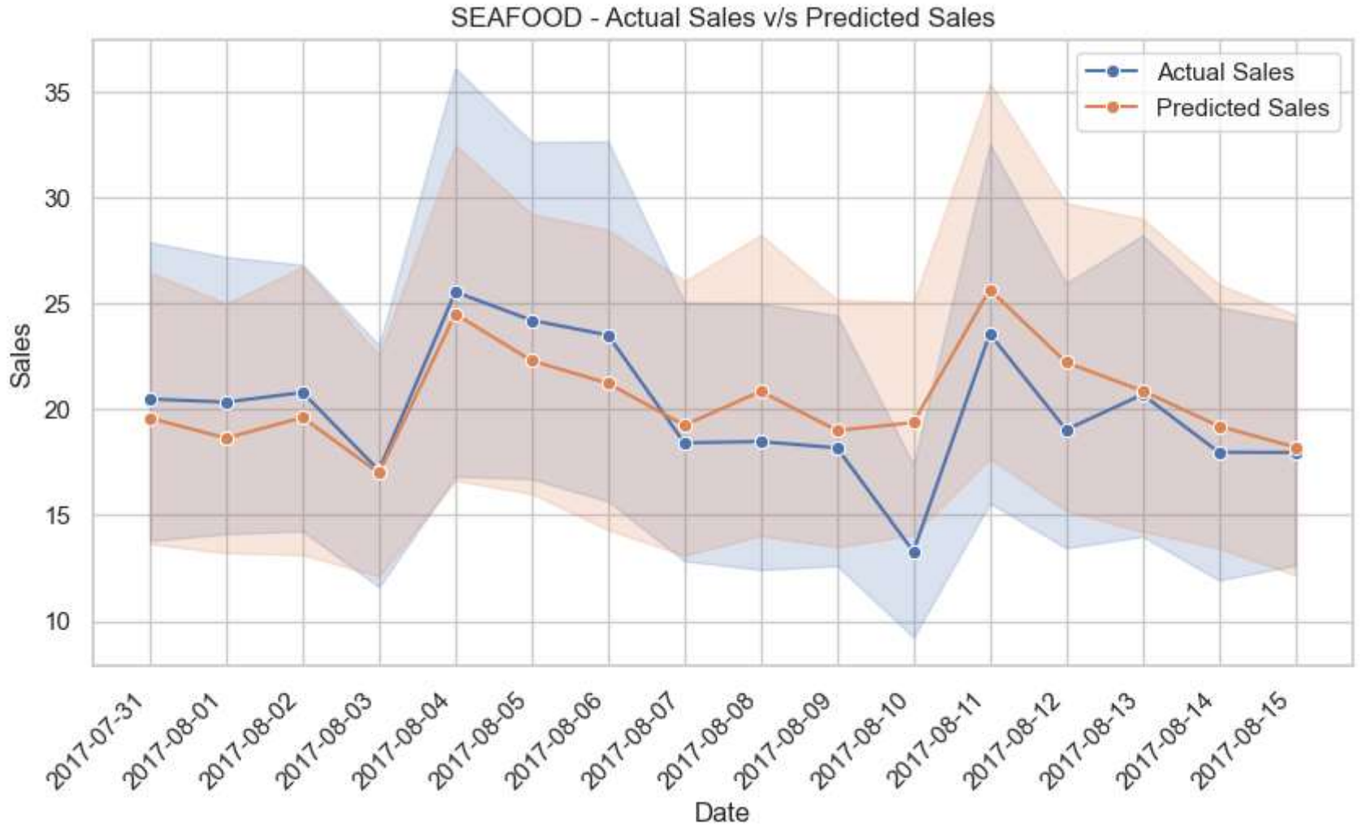

PLAYERS AND ELECTRONICS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
20
18
16
14
12
10
8
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


POULTRY - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
700
600
500
400
300
200
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


PREPARED FOODS - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
120
110
100
90
80
70
60
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


PRODUCE - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
4000
3500
3000
2500
2000
1500
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15
Date


SCHOOL AND OFFICE SUPPLIES - Actual Sales v/s Predicted Sales
Actual Sales
Predicted Sales
Sales
Date
0
25
50
75
100
125
150
175
200
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15



HyperParameter Tuning

In [286…
```
param_grid = {
    'n_estimators': [300],
    'min_samples_split': [5],
    'min_samples_leaf': [5],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search = GridSearchCV(rf_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

Out[286]:

GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

In [315…
```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

In [316…
```
best_params
```

Out[316]:
```
{'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 5,
 'n_estimators': 300}
```

In [289…
```
best_model
```

Out[289]:
```
RandomForestRegressor(max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=5, n_estimators=300, oob_score=True,
                      warm_start=True)
```

In [290…
```
tuned_predictions = best_model.predict(X_test)
test_mse = mean_squared_error(y_test, tuned_predictions)
test_rmse = mean_squared_error(y_test, tuned_predictions, squared=False)
```

In [291…
```
test_mse
```

Out[291]:
```
99066.7603304566
```

In [292…
```
test_rmse
```

Out[292]:
```
314.7487257010846
```

In [294…
```
X_unseen_test = unseen_data.drop(columns=['sales','product_type','store_nbr'])
tuned_predictions = best_model.predict(X_unseen_test)

mae = mean_absolute_error(unseen_data['sales'], tuned_predictions)
mse = mean_squared_error(unseen_data['sales'], tuned_predictions)
rmse = mean_squared_error(unseen_data['sales'], tuned_predictions,squared=False)


print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Absolute Error: 83.96644820198296
Mean Squared Error: 84524.423897688
Root Mean Squared Error: 290.7308444208973
```

Total sales vs predicted (all time)

In [297…
```
final_data_tuned = pd.concat([unseen_data[['store_nbr','product_type','sales']].reset_index(),pd.DataFrame(tuned_predictions)],axis=1)
```

In [301…
```
final_data_tuned.rename(columns={0:'predicted_tuned_sales'},inplace=True)
final_data_tuned.rename(columns={'index_date':'date'},inplace=True)
```

In [302…
```
final_data_tuned
```

Out[302]:

	date	store_nbr	product_type	sales	predicted_tuned_sales
0	2017-07-31	1	0	8.0	4.706833
1	2017-08-01	1	0	5.0	6.764011
2	2017-08-02	1	0	4.0	5.327129
3	2017-08-03	1	0	3.0	4.683065
4	2017-08-04	1	0	8.0	5.009571
...	...	...	...	...	...
28507	2017-08-11	54	32	0.0	3.436092
28508	2017-08-12	54	32	1.0	3.588129
28509	2017-08-13	54	32	2.0	2.733216
28510	2017-08-14	54	32	0.0	2.714036
28511	2017-08-15	54	32	3.0	2.284172

28512 rows × 5 columns

In [305…
```
summed_actual = pd.DataFrame(final_data_tuned.groupby('date')['sales'].sum())
```

In [304…
```
pred_tuned_sales = pd.DataFrame(final_data_tuned.groupby('date')['predicted_tuned_sales'].sum())
```

In [306…
```
summed_actual
```

Out[306]:

date	sales
2017-07-31	8.858568e+05
2017-08-01	9.885278e+05
2017-08-02	9.647120e+05
2017-08-03	7.280685e+05
2017-08-04	8.277757e+05
2017-08-05	9.656937e+05
2017-08-06	1.049559e+06
2017-08-07	7.974650e+05
2017-08-08	7.177663e+05
2017-08-09	7.341397e+05
2017-08-10	6.513869e+05
2017-08-11	8.263737e+05
2017-08-12	7.926305e+05
2017-08-13	8.656397e+05
2017-08-14	7.609224e+05
2017-08-15	7.626619e+05

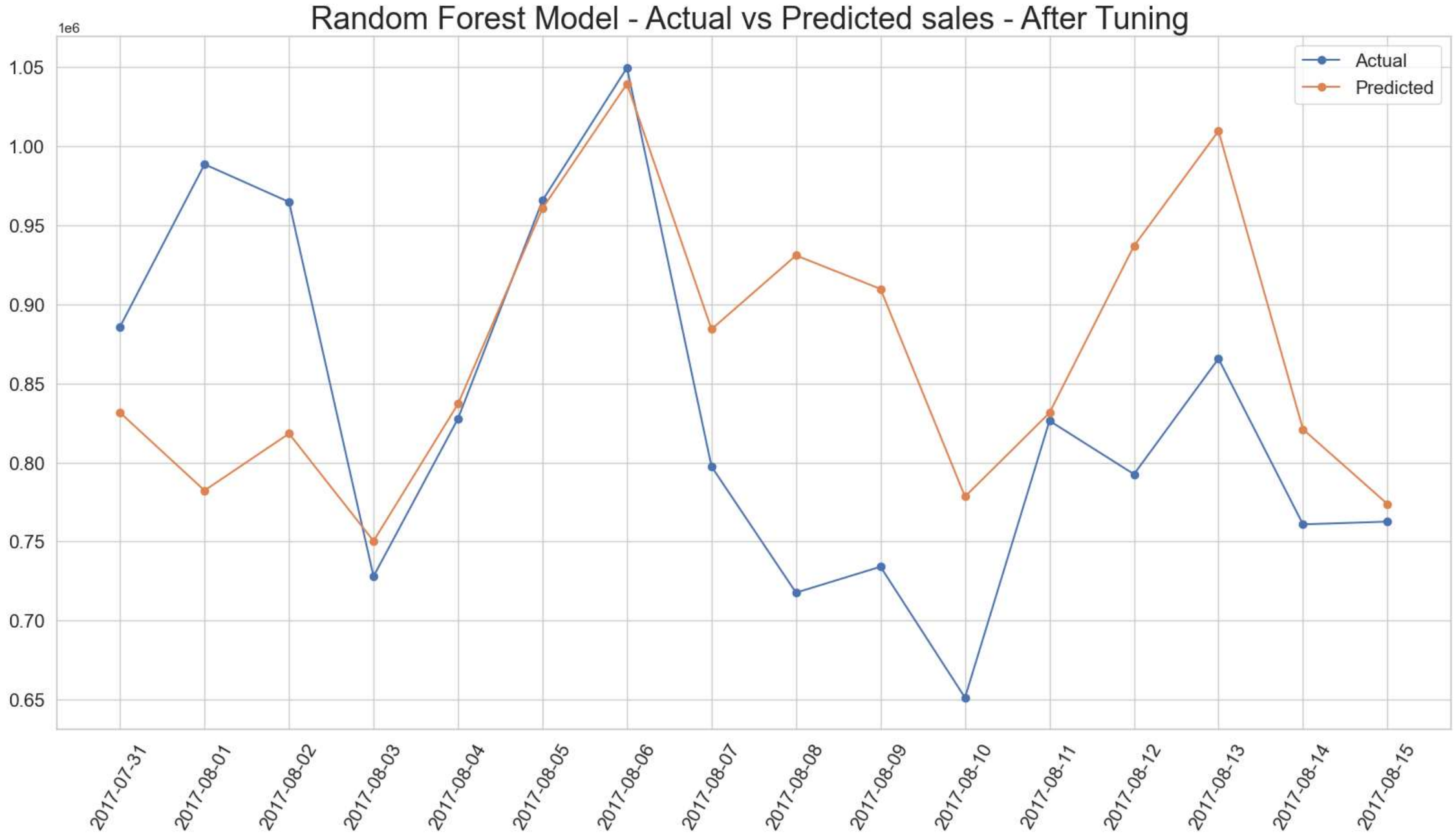
In [307…
```
pred_tuned_sales
```

Out[307]:

date	predicted_tuned_sales
2017-07-31	8.315525e+05
2017-08-01	7.823436e+05
2017-08-02	8.182895e+05
2017-08-03	7.504103e+05
2017-08-04	8.370540e+05
2017-08-05	9.606962e+05
2017-08-06	1.039434e+06
2017-08-07	8.842570e+05
2017-08-08	9.309164e+05
2017-08-09	9.097588e+05
2017-08-10	7.784736e+05
2017-08-11	8.314143e+05
2017-08-12	9.369276e+05
2017-08-13	1.009556e+06
2017-08-14	8.210299e+05
2017-08-15	7.737720e+05

In [314…
```
data_values = pred_tuned_sales.index
sns.set(style="whitegrid")
plt.figure(figsize=(20,10))
plt.plot(summed_actual['sales'],marker='o',label='Actual')
plt.plot(pred_tuned_sales['predicted_tuned_sales'],marker='o',label='Predicted')
plt.legend(fontsize=15)
plt.title("Random Forest Model - Actual vs Predicted sales - After Tuning ",fontsize=25)
plt.xticks(data_values,fontsize=15, rotation=60)
plt.yticks(fontsize=15)
```

Out[314]:
```
(array([ 600000.,  650000.,  700000.,  750000.,  800000.,  850000.,
         900000.,  950000., 1000000., 1050000., 1100000.]),
 [Text(0, 600000.0, '0.60'),
  Text(0, 650000.0, '0.65'),
  Text(0, 700000.0, '0.70'),
  Text(0, 750000.0, '0.75'),
  Text(0, 800000.0, '0.80'),
  Text(0, 850000.0, '0.85'),
  Text(0, 900000.0, '0.90'),
  Text(0, 950000.0, '0.95'),
  Text(0, 1000000.0, '1.00'),
  Text(0, 1050000.0, '1.05'),
  Text(0, 1100000.0, '1.10')])
```


Random Forest Model - Actual vs Predicted sales - After Tuning
1e6
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Actual
Predicted
2017-07-31
2017-08-01
2017-08-02
2017-08-03
2017-08-04
2017-08-05
2017-08-06
2017-08-07
2017-08-08
2017-08-09
2017-08-10
2017-08-11
2017-08-12
2017-08-13
2017-08-14
2017-08-15

In [ ]: