# Analysis of Perishable Item Sales in Restaurants

This section provides an overview of the dataset used to analyze sales trends of perishable items in restaurants, along with details about preprocessing, decomposition, and data augmentation.

## **1. Datasets Overview**:

We utilize two primary datasets for our analysis:

### **Pizza Sales Dataset**:

* Contains records of daily pizza sales.
* **Date**: Represents the specific day of pizza sales data.
* **Quantity**: Indicates the number of pizzas sold on the respective date.

### **Bakery Sales Dataset**:

* Contains records of daily sales for bakery items.
* **Date**: Represents the specific day of bakery sales data.
* **Quantity**: Indicates the volume of bakery items sold on the respective date.

These datasets not only help in understanding historical sales trends but also serve as a foundation for modeling and prediction.

## **3. Time Series Decomposition**:

Decomposing the time series data provides insights into its various components:

* **Original Series**: Displays the raw sales data.
* **Trend Component**: Shows the underlying trend in sales over time.
* **Seasonal Component**: Represents the regular fluctuations in sales, potentially due to weekly patterns or other recurring events.
* **Residual Component**: Contains the remaining variations in sales after removing the trend and seasonal components.

## **4. Data Augmentation**:

To improve the dataset's coverage and enhance its predictive capabilities, additional data points are generated:

* **Outliers**
* **Backcasting**: Generating past data points using patterns from the existing dataset.
* **Forecasting**: Predicting future data points based on the established trends and patterns.

## **5. Model Training and Initial Validation**:

With the preprocessed and augmented datasets, the model is trained. An initial validation is conducted to assess the model's performance and ensure its reliability before full-scale deployment.

## 0.1 Import necessary librearies

# Standard libraries and data processing  
import pandas as pd  
import numpy as np  
from datetime import datetime, timedelta  
  
  
# Visualization libraries  
import matplotlib.pyplot as plt  
import seaborn as sns  
from matplotlib.dates import DateFormatter  
  
# Time series analysis  
from statsmodels.tsa.seasonal import STL  
from scipy.stats import zscore  
  
# Machine learning and deep learning libraries  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.metrics import mean\_squared\_error  
import tensorflow as tf  
from tensorflow.keras.models import Sequential, load\_model  
from tensorflow.keras.layers import LSTM, Dense, Dropout  
  
# Miscellaneous libraries  
import holidays

## 1.1 Pizza Sales Dataset

# Load the Excel file  
file\_path = 'pizza\_sales.xlsx'  
  
df = pd.read\_excel(file\_path)  
print('Original Pizza dataframe: ', df)  
  
  
# Group by order\_date and aggregate the quantity and total\_price columns  
daily\_sales = df.groupby('order\_date').agg({  
 'quantity': 'sum',  
   
}).reset\_index()  
daily\_sales = daily\_sales.rename(columns={'order\_date': 'date'})  
  
  
print("\n---------------------------------------------------")  
print("Cleaned dataframe")  
daily\_sales

Original Pizza dataframe: order\_details\_id order\_id pizza\_id quantity order\_date \  
0 1 1 hawaiian\_m 1 2015-01-01   
1 2 2 classic\_dlx\_m 1 2015-01-01   
2 3 2 five\_cheese\_l 1 2015-01-01   
3 4 2 ital\_supr\_l 1 2015-01-01   
4 5 2 mexicana\_m 1 2015-01-01   
... ... ... ... ... ...   
48615 48616 21348 ckn\_alfredo\_m 1 2015-12-31   
48616 48617 21348 four\_cheese\_l 1 2015-12-31   
48617 48618 21348 napolitana\_s 1 2015-12-31   
48618 48619 21349 mexicana\_l 1 2015-12-31   
48619 48620 21350 bbq\_ckn\_s 1 2015-12-31   
  
 order\_time unit\_price total\_price pizza\_size pizza\_category \  
0 11:38:36 13.25 13.25 M Classic   
1 11:57:40 16.00 16.00 M Classic   
2 11:57:40 18.50 18.50 L Veggie   
3 11:57:40 20.75 20.75 L Supreme   
4 11:57:40 16.00 16.00 M Veggie   
... ... ... ... ... ...   
48615 21:23:10 16.75 16.75 M Chicken   
48616 21:23:10 17.95 17.95 L Veggie   
48617 21:23:10 12.00 12.00 S Classic   
48618 22:09:54 20.25 20.25 L Veggie   
48619 23:02:05 12.75 12.75 S Chicken   
  
 pizza\_ingredients \  
0 Sliced Ham, Pineapple, Mozzarella Cheese   
1 Pepperoni, Mushrooms, Red Onions, Red Peppers,...   
2 Mozzarella Cheese, Provolone Cheese, Smoked Go...   
3 Calabrese Salami, Capocollo, Tomatoes, Red Oni...   
4 Tomatoes, Red Peppers, Jalapeno Peppers, Red O...   
... ...   
48615 Chicken, Red Onions, Red Peppers, Mushrooms, A...   
48616 Ricotta Cheese, Gorgonzola Piccante Cheese, Mo...   
48617 Tomatoes, Anchovies, Green Olives, Red Onions,...   
48618 Tomatoes, Red Peppers, Jalapeno Peppers, Red O...   
48619 Barbecued Chicken, Red Peppers, Green Peppers,...   
  
 pizza\_name   
0 The Hawaiian Pizza   
1 The Classic Deluxe Pizza   
2 The Five Cheese Pizza   
3 The Italian Supreme Pizza   
4 The Mexicana Pizza   
... ...   
48615 The Chicken Alfredo Pizza   
48616 The Four Cheese Pizza   
48617 The Napolitana Pizza   
48618 The Mexicana Pizza   
48619 The Barbecue Chicken Pizza   
  
[48620 rows x 12 columns]  
  
---------------------------------------------------  
Cleaned dataframe

date quantity  
0 2015-01-01 162  
1 2015-01-02 165  
2 2015-01-03 158  
3 2015-01-04 106  
4 2015-01-05 125  
.. ... ...  
353 2015-12-27 89  
354 2015-12-28 102  
355 2015-12-29 80  
356 2015-12-30 82  
357 2015-12-31 178  
  
[358 rows x 2 columns]

## 1.2 Bakery Dataset

# Load the data  
df\_backery = pd.read\_csv('bakery\_sales.csv')  
  
print('Original Bakery dataframe: ', df\_backery)  
  
# Group by the 'date' column and sum the 'Quantity' for each date  
df\_backery = df\_backery.groupby('date').agg({'Quantity': 'sum'}).reset\_index()  
  
# Rename columns for clarity  
df\_backery.columns = ['date', 'quantity']  
df\_backery['quantity'] = df\_backery['quantity'].astype(int)  
df\_backery['date'] = pd.to\_datetime(df\_backery['date'], format='%Y-%m-%d')  
  
print("\n---------------------------------------------------")  
print("Cleaned dataframe")# Display the resulting dataframe  
  
df\_backery

Original Bakery dataframe: Unnamed: 0 date time ticket\_number article \  
0 0 2021-01-02 08:38 150040.0 BAGUETTE   
1 1 2021-01-02 08:38 150040.0 PAIN AU CHOCOLAT   
2 4 2021-01-02 09:14 150041.0 PAIN AU CHOCOLAT   
3 5 2021-01-02 09:14 150041.0 PAIN   
4 8 2021-01-02 09:25 150042.0 TRADITIONAL BAGUETTE   
... ... ... ... ... ...   
234000 511387 2022-09-30 18:52 288911.0 COUPE   
234001 511388 2022-09-30 18:52 288911.0 BOULE 200G   
234002 511389 2022-09-30 18:52 288911.0 COUPE   
234003 511392 2022-09-30 18:55 288912.0 TRADITIONAL BAGUETTE   
234004 511395 2022-09-30 18:56 288913.0 TRADITIONAL BAGUETTE   
  
 Quantity unit\_price   
0 1.0 0,90 €   
1 3.0 1,20 €   
2 2.0 1,20 €   
3 1.0 1,15 €   
4 5.0 1,20 €   
... ... ...   
234000 1.0 0,15 €   
234001 1.0 1,20 €   
234002 2.0 0,15 €   
234003 1.0 1,30 €   
234004 1.0 1,30 €   
  
[234005 rows x 7 columns]  
  
---------------------------------------------------  
Cleaned dataframe

date quantity  
0 2021-01-02 581  
1 2021-01-03 564  
2 2021-01-04 315  
3 2021-01-05 309  
4 2021-01-07 310  
.. ... ...  
595 2022-09-26 399  
596 2022-09-27 423  
597 2022-09-28 357  
598 2022-09-29 428  
599 2022-09-30 503  
  
[600 rows x 2 columns]

## 2.1 Pizza Sales Dataset Time series Decomposition

# Create a date range for the entire period  
complete\_date\_range = pd.date\_range(start=daily\_sales["date"].min(), end=daily\_sales["date"].max())  
  
# Identify missing dates  
missing\_dates = complete\_date\_range[~complete\_date\_range.isin(daily\_sales["date"])]  
  
# Fill in the missing dates  
daily\_sales\_filled = daily\_sales.set\_index("date").reindex(complete\_date\_range)  
daily\_sales\_filled.index.name = "date"  
daily\_sales\_filled = daily\_sales\_filled.interpolate(method="linear").reset\_index()  
  
# STL decomposition with a fixed period of 7 days  
stl = STL(daily\_sales\_filled["quantity"], seasonal=7, period=7)  
result\_pizza = stl.fit()

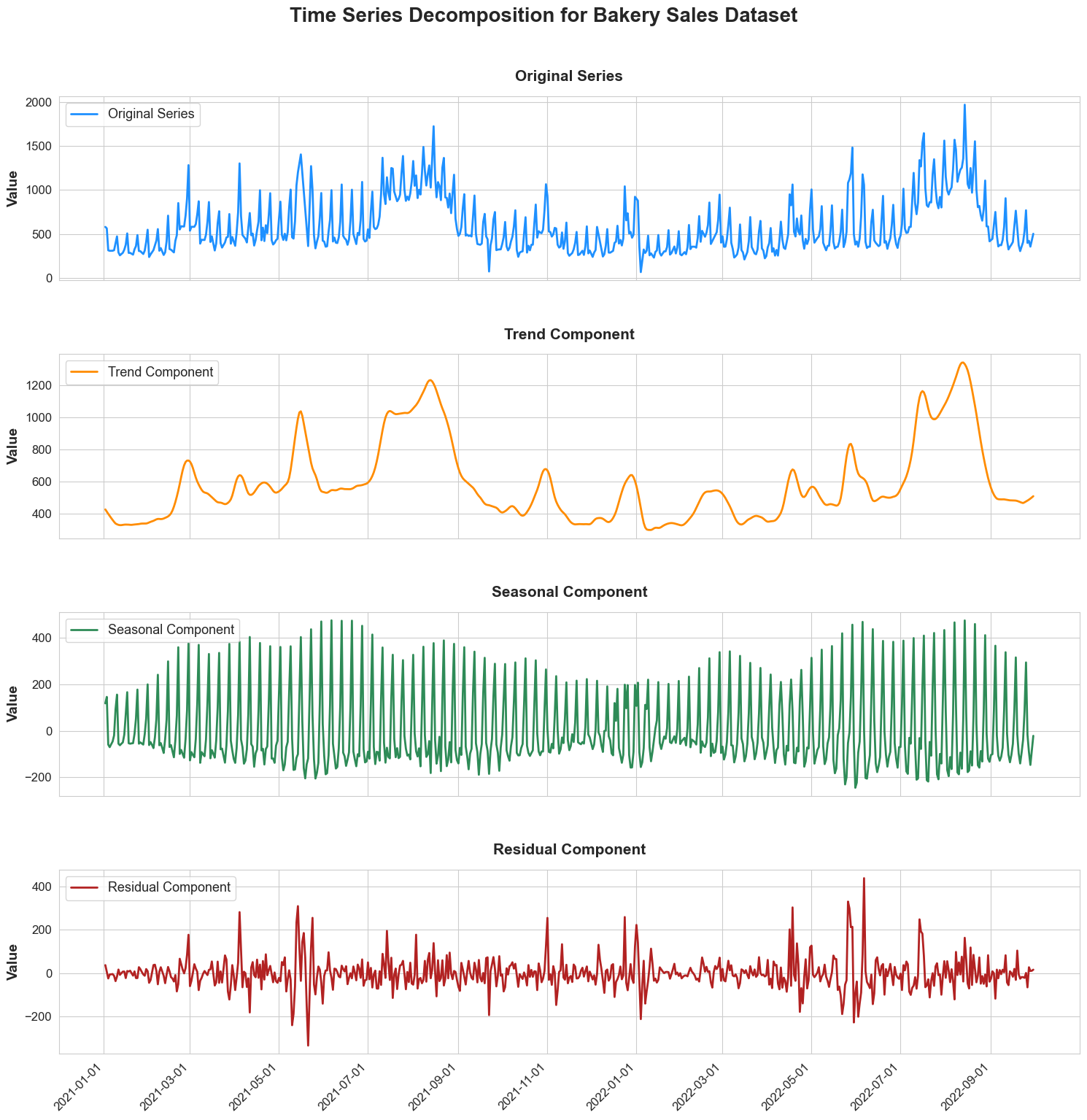
# Set seaborn style  
sns.set\_style("whitegrid")  
  
# Set a consistent color scheme  
colors = ["dodgerblue", "darkorange", "seagreen", "firebrick"]  
  
# Create the subplots with more space  
fig, axes = plt.subplots(4, 1, figsize=(15, 15), sharex=True)  
  
# Plot data  
components = [("Original Series", daily\_sales\_filled["quantity"]),  
 ("Trend Component", result\_pizza.trend),  
 ("Seasonal Component", result\_pizza.seasonal),  
 ("Residual Component", result\_pizza.resid)]  
  
# Plot each component  
for (title, data), color, ax in zip(components, colors, axes):  
 ax.plot(daily\_sales\_filled["date"], data, label=title, color=color, linewidth=2)  
 ax.set\_title(title, fontsize=15, fontweight='bold', pad=15)  
 ax.tick\_params(axis='both', which='major', labelsize=12)  
 ax.legend(loc="upper left", fontsize=13)  
 ax.set\_ylabel('Value', fontsize=14, fontweight='semibold')  
  
# Limit the number of ticks on the x-axis and format the ticks  
date\_form = DateFormatter("%Y-%m-%d")  
axes[-1].xaxis.set\_major\_formatter(date\_form)  
  
# Rotate x-axis labels for better visibility  
plt.setp(axes[-1].get\_xticklabels(), rotation=45, ha="right")  
  
# Set a main title for the entire figure  
fig.suptitle('Time Series Decomposition for Pizza Sales Dataset', fontsize=20, fontweight='bold', y=1.02)  
  
# Adjust the layout  
plt.tight\_layout()  
plt.subplots\_adjust(hspace=0.4)  
  
# Display the plot  
plt.show()



## 2.2 Bakery sales Time series Decomposition

# Create a date range for the entire period  
complete\_date\_range = pd.date\_range(start=df\_backery["date"].min(), end=df\_backery["date"].max())  
  
# Identify missing dates  
missing\_dates = complete\_date\_range[~complete\_date\_range.isin(df\_backery["date"])]  
  
# Fill in the missing dates  
df\_backery = df\_backery.set\_index("date").reindex(complete\_date\_range)  
df\_backery.index.name = "date"  
df\_backery = df\_backery.interpolate(method="linear").reset\_index()  
  
# STL decomposition with a fixed period of 7 days  
stl = STL(df\_backery["quantity"], seasonal=7, period=7)  
result\_bakery = stl.fit()

# Set seaborn style  
sns.set\_style("whitegrid")  
  
# Set a consistent color scheme  
colors = ["dodgerblue", "darkorange", "seagreen", "firebrick"]  
  
# Create the subplots with more space  
fig, axes = plt.subplots(4, 1, figsize=(15, 15), sharex=True)  
  
# Plot data  
components = [("Original Series", df\_backery["quantity"]),  
 ("Trend Component", result\_bakery.trend),  
 ("Seasonal Component", result\_bakery.seasonal),  
 ("Residual Component", result\_bakery.resid)]  
  
# Plot each component  
for (title, data), color, ax in zip(components, colors, axes):  
 ax.plot(df\_backery["date"], data, label=title, color=color, linewidth=2)  
 ax.set\_title(title, fontsize=15, fontweight='bold', pad=15)  
 ax.tick\_params(axis='both', which='major', labelsize=12)  
 ax.legend(loc="upper left", fontsize=13)  
 ax.set\_ylabel('Value', fontsize=14, fontweight='semibold')  
  
# Limit the number of ticks on the x-axis and format the ticks  
date\_form = DateFormatter("%Y-%m-%d")  
axes[-1].xaxis.set\_major\_formatter(date\_form)  
  
# Rotate x-axis labels for better visibility  
plt.setp(axes[-1].get\_xticklabels(), rotation=45, ha="right")  
  
# Set a main title for the entire figure  
fig.suptitle('Time Series Decomposition for Bakery Sales Dataset', fontsize=20, fontweight='bold', y=1.02)  
  
# Adjust the layout  
plt.tight\_layout()  
plt.subplots\_adjust(hspace=0.4)  
  
# Display the plot  
plt.show()



## ## 3. Data Augmentation and Outlier Management

### 1. Backcasting and Forecasting:

- Using the historical trend and seasonal components extracted from the existing data,   
 we can generate data for periods before and after the known data.  
- Backcasting involves producing data for earlier periods, and forecasting is for later periods.

### 2. Outlier Detection and Management:

- Time series data can often have outliers - values that deviate significantly from the expected range.  
- We calculate the Z-scores for each data point. The Z-score measures how many standard deviations   
 a data point is from the mean.  
- We consider data points with Z-scores beyond a threshold (e.g., |Z-score| > 3) as outliers.

### 3. Visualization:

- Visualizing the augmented data helps in understanding the data structure and identifying any anomalies.  
- We plot the data for a specific year and highlight outliers in a different color to distinguish them easily.

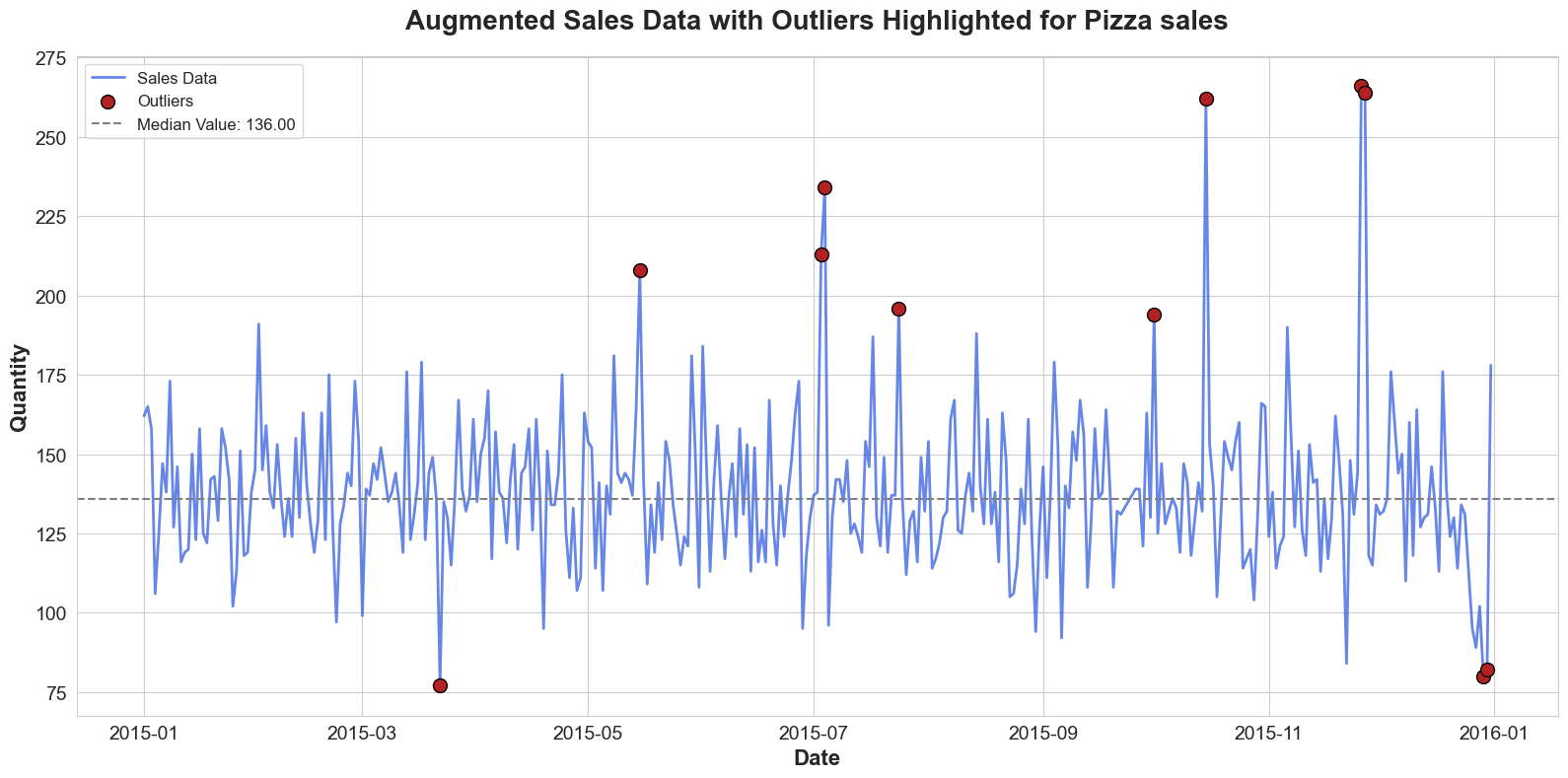
### 4. Outlier Treatment:

- One common method to handle outliers is to replace them with a measure of central tendency, such as the median.  
- We use a rolling window to calculate a localized median around each outlier   
 and then replace the outlier with this median.

def augment\_and\_clean\_data(data, result, start\_date, end\_date, plot\_year, title):  
 # Define the date ranges to backcast and forecast  
 backcast\_range = pd.date\_range(start=start\_date, end=data["date"].min() - pd.Timedelta(days=1))  
 forecast\_range = pd.date\_range(start=data["date"].max() + pd.Timedelta(days=1), end=end\_date)  
   
 # Backcasting and Forecasting  
 backcast\_values = [(result.trend.iloc[i % len(result.trend)] + result.seasonal.iloc[i % len(result.seasonal)]) for i, \_ in enumerate(reversed(backcast\_range))]  
 forecast\_values = [(result.trend.iloc[-(i % len(result.trend)) - 1] + result.seasonal.iloc[i % len(result.seasonal)]) for i, \_ in enumerate(forecast\_range)]  
   
 # Combine the backcasted, original, and forecasted data  
 augmented\_data = pd.DataFrame({  
 "date": list(backcast\_range) + list(data["date"]) + list(forecast\_range),  
 "quantity": backcast\_values + list(data["quantity"]) + forecast\_values  
 })  
   
 # Detect outliers based on Z-scores  
 augmented\_data["zscore"] = zscore(augmented\_data["quantity"])  
 threshold = 3  
 augmented\_data["is\_outlier"] = augmented\_data["zscore"].abs() > threshold  
   
 # Plotting  
 subset\_data = augmented\_data[augmented\_data["date"].dt.year == plot\_year]  
 sns.set\_style("whitegrid")  
 sns.set\_palette("pastel")  
 fig, ax = plt.subplots(figsize=(16, 8))  
 ax.plot(subset\_data["date"], subset\_data["quantity"], label="Sales Data", linewidth=2, color='royalblue', alpha=0.8)  
 outliers = subset\_data[subset\_data["is\_outlier"]]  
 ax.scatter(outliers["date"], outliers["quantity"], color="firebrick", s=100, edgecolor='black', zorder=5, label='Outliers')  
 ax.set\_title(title, fontsize=20, fontweight='bold', pad=20)  
 ax.set\_xlabel("Date", fontsize=16, fontweight='semibold')  
 ax.set\_ylabel("Quantity", fontsize=16, fontweight='semibold')  
 ax.tick\_params(axis='both', labelsize=14)  
 median\_val = subset\_data["quantity"].median()  
 ax.axhline(y=median\_val, color='gray', linestyle='--', label=f"Median Value: {median\_val:.2f}")  
 ax.legend(fontsize=12, loc="upper left")  
 plt.tight\_layout()  
 plt.show()  
   
 # Handle outliers: replace with the median of surrounding data points  
 window\_size = 3  
 medians = augmented\_data["quantity"].rolling(window=window\_size, center=True).median()  
 augmented\_data.loc[augmented\_data["is\_outlier"], "quantity"] = medians[augmented\_data["is\_outlier"]]  
   
 return augmented\_data[['date', 'quantity']]

## 3.1 Pizza dataset outliers

df\_pizza = augment\_and\_clean\_data(daily\_sales\_filled, result\_pizza, "2000-01-01", "2020-12-31", 2015, "Augmented Sales Data with Outliers Highlighted for Pizza sales")



df\_pizza

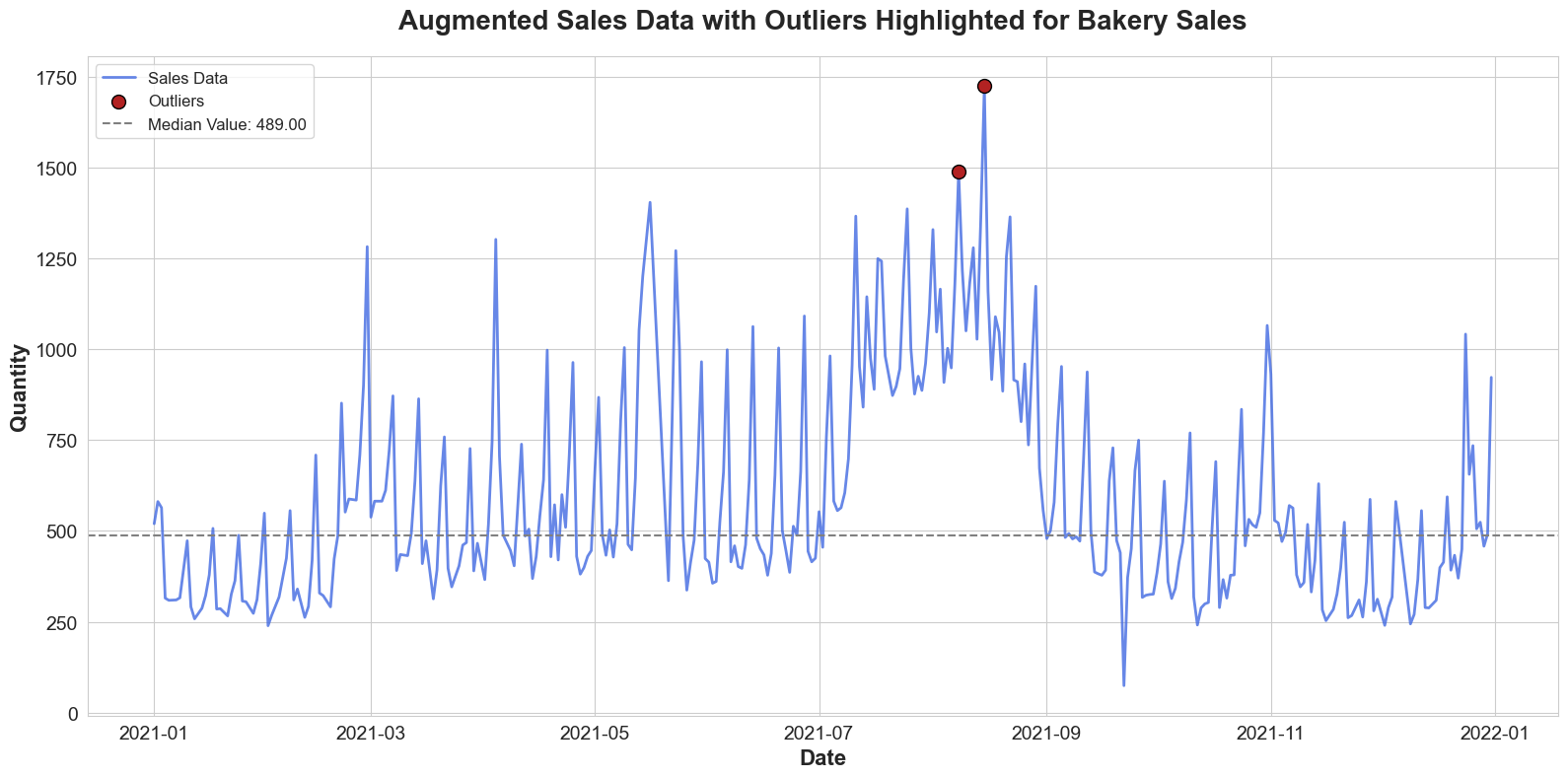
date quantity  
0 2000-01-01 170.442315  
1 2000-01-02 157.326777  
2 2000-01-03 155.744584  
3 2000-01-04 108.092598  
4 2000-01-05 127.743851  
... ... ...  
7666 2020-12-27 129.366701  
7667 2020-12-28 137.656924  
7668 2020-12-29 137.656924  
7669 2020-12-30 123.733819  
7670 2020-12-31 113.458970  
  
[7671 rows x 2 columns]

df\_pizza.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7671 entries, 0 to 7670  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 date 7671 non-null datetime64[ns]  
 1 quantity 7671 non-null float64   
dtypes: datetime64[ns](1), float64(1)  
memory usage: 120.0 KB

## 3.2 Bakery dataset outliers

df\_bakery = augment\_and\_clean\_data(df\_backery, result\_bakery, "2005-01-01", "2023-08-28", 2021, "Augmented Sales Data with Outliers Highlighted for Bakery Sales")



df\_bakery

date quantity  
0 2005-01-01 544.312674  
1 2005-01-02 558.031620  
2 2005-01-03 339.790893  
3 2005-01-04 315.557867  
4 2005-01-05 315.269951  
... ... ...  
6809 2023-08-24 427.181231  
6810 2023-08-25 448.042127  
6811 2023-08-26 556.063481  
6812 2023-08-27 803.470561  
6813 2023-08-28 605.219653  
  
[6814 rows x 2 columns]

df\_bakery.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 6814 entries, 0 to 6813  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 date 6814 non-null datetime64[ns]  
 1 quantity 6814 non-null float64   
dtypes: datetime64[ns](1), float64(1)  
memory usage: 106.6 KB

# FoodSalesPredictor: Class Overview

The FoodSalesPredictor class is designed as a comprehensive solution to predict daily sales of perishable items within a restaurant context, encapsulating everything from features engineering to model training and evaluation.

## Dataset Requirements

For this class to function correctly, the input dataset (df) should have the following structure:

* A DataFrame with two columns :
  + date: In datetime format.
  + quantity: Numeric representation of daily sales or similar metric.

#### **1. Class Description**:

This class encapsulates methods to preprocess sales data, build a Long Short-Term Memory (LSTM) model, train the model, and make predictions on test data. It also includes utility functions to handle features such as holidays, weekends, and paydays, which can influence daily sales in restaurants.

#### **2. Feature Processing**:

The class focuses on extracting meaningful features from the given sales data:

* **Seasonality**: Based on the month, the data is categorized into 'Winter', 'Spring', 'Summer', or 'Fall'.
* **Holidays**: Leveraging the holidays library, the data is annotated with a flag indicating if a day is a public holiday.
* **Weekends**: Days are flagged if they fall on weekends.
* **Paydays**: Days are flagged if they represent month-ends, which could potentially be paydays.
* **Lag Features**: The class creates lag features, indicating sales data from previous days, providing context for the model about recent sales trends.

#### **3. Model Creation**:

The choice of model is the LSTM, a type of recurrent neural network (RNN). LSTM is designed to recognize patterns over sequences, making it an optimal choice for time-series forecasting like daily sales predictions.

#### **4. Rationale Behind Model Choice**:

LSTMs have the capability to remember past information, which is essential when predicting sequences with fluctuations, like sales data. The LSTM can utilize its internal memory to process sequences of observations. This makes it suitable for our use-case where past sales data can have an influence on future sales.

#### **5. Model Parameters**:

* **LSTM Units**: The LSTM layers have units that determine the memory capability and the amount of information they can store.
* **Dropout**: Dropout layers are introduced to prevent overfitting. They randomly set a fraction rate of input units to 0 at each update during training time.
* **Dense Layer**: This is the output layer, which provides the final prediction. It uses a linear activation function.

In essence, the FoodSalesPredictor serves as a one-stop solution for building a robust perishable item sales forecasting system.

class FoodSalesPredictor:  
 def \_\_init\_\_(self, df):  
 self.df = df.copy()  
 self.quantity\_scaler = MinMaxScaler(feature\_range=(0, 1))  
 self.scaler = MinMaxScaler(feature\_range=(0, 1))  
 self.look\_back = 20  
 self.model = None  
  
 def get\_season(self, date):  
 """ get season feature for the df"""  
   
 if date.month in [12, 1, 2]:  
 return 'Winter'  
 elif date.month in [3, 4, 5]:  
 return 'Spring'  
 elif date.month in [6, 7, 8]:  
 return 'Summer'  
 else:  
 return 'Fall'  
  
 def preprocess\_data(self):  
 """Process the df extracting new features splitting the df in 90% train and 10% test"""  
   
 us\_holidays = holidays.US(years=self.df['date'].dt.year.unique())  
 self.df['is\_holiday'] = self.df['date'].apply(lambda x: 1 if x in us\_holidays else 0)  
 self.df['is\_weekend'] = self.df['date'].apply(lambda x: 1 if x.weekday() >= 5 else 0)  
 self.df['is\_paycheck'] = self.df['date'].apply(lambda x: 1 if x.is\_month\_end else 0)  
 self.df['season'] = self.df['date'].apply(self.get\_season)  
  
 # Create lag features  
 n\_lags = 7  
 for i in range(1, n\_lags + 1):  
 self.df[f'lag\_{i}'] = self.df['quantity'].shift(i)  
  
 self.df.dropna(inplace=True)  
 self.df = pd.get\_dummies(self.df, columns=['season'])  
  
 # Scale the 'quantity' column  
 self.df['quantity'] = self.quantity\_scaler.fit\_transform(self.df[['quantity']])  
  
 # Normalize the entire dataframe  
 scaled\_data = self.scaler.fit\_transform(self.df.drop(columns='date'))  
   
 # Determine the correct index for 'quantity' column after one-hot encoding  
 quantity\_col\_index = list(self.df.columns).index('quantity')  
  
 X, Y = [], []  
 for i in range(len(scaled\_data)-self.look\_back):  
 X.append(scaled\_data[i:(i+self.look\_back), :-1])  
 Y.append(scaled\_data[i + self.look\_back, quantity\_col\_index-1])  
  
 self.X = np.array(X)  
 self.Y = np.array(Y)  
   
 train\_size = int(len(self.X) \* 0.90)  
 test\_size = len(self.X) - train\_size  
   
 self.X\_train, self.X\_test = self.X[0:train\_size], self.X[train\_size:len(X)]  
 self.Y\_train, self.Y\_test = self.Y[0:train\_size], self.Y[train\_size:len(Y)]  
  
   
   
 def preprocess\_all\_data(self):  
 """Process the df extracting new features without splitting the df for training"""  
   
 us\_holidays = holidays.US(years=self.df['date'].dt.year.unique())  
 self.df['is\_holiday'] = self.df['date'].apply(lambda x: 1 if x in us\_holidays else 0)  
 self.df['is\_weekend'] = self.df['date'].apply(lambda x: 1 if x.weekday() >= 5 else 0)  
 self.df['is\_paycheck'] = self.df['date'].apply(lambda x: 1 if x.is\_month\_end else 0)  
 self.df['season'] = self.df['date'].apply(self.get\_season)  
  
 # Create lag features  
 n\_lags = 7  
 for i in range(1, n\_lags + 1):  
 self.df[f'lag\_{i}'] = self.df['quantity'].shift(i)  
  
 self.df.dropna(inplace=True)  
  
 # Create one-hot encoded columns for all seasons manually  
 seasons = ['Winter', 'Spring', 'Summer', 'Fall']  
 for season in seasons:  
 self.df[f'season\_{season}'] = self.df['season'].apply(lambda x: 1 if x == season else 0)  
  
 self.df.drop(columns=['season'], inplace=True)  
   
 # Scale the 'quantity' column  
 self.df['quantity'] = self.quantity\_scaler.fit\_transform(self.df[['quantity']])  
   
 scaled\_data = self.scaler.fit\_transform(self.df.drop(columns='date'))  
   
 # Determine the correct index for 'quantity' column after one-hot encoding  
 quantity\_col\_index = list(self.df.columns).index('quantity')  
  
 X, Y = [], []  
 for i in range(len(scaled\_data)-self.look\_back):  
 X.append(scaled\_data[i:(i+self.look\_back), :-1])  
 Y.append(scaled\_data[i + self.look\_back, quantity\_col\_index-1])  
  
   
 self.X = np.array(X)  
 self.Y = np.array(Y)  
  
 def build\_model(self, lstm\_units=50):  
 """Build simplified LSTM model with fewer dense layers"""  
  
 self.model = Sequential()  
 input\_shape = self.X\_train.shape if hasattr(self, 'X\_train') else self.X.shape  
  
 self.model.add(LSTM(lstm\_units, input\_shape=(input\_shape[1], input\_shape[2]), return\_sequences=True))  
 self.model.add(Dropout(0.4))  
  
 self.model.add(LSTM(100, return\_sequences=False))  
 self.model.add(Dropout(0.4))  
  
 self.model.add(Dense(1))  
  
 self.model.compile(loss='mean\_squared\_error', optimizer=tf.keras.optimizers.legacy.Adam())  
  
   
 def train\_model(self, epochs=200, batch\_size=264):  
 self.model.fit(self.X\_train, self.Y\_train, epochs=epochs, batch\_size=batch\_size, verbose=1)  
  
 def make\_predictions(self):  
 predictions = self.model.predict(self.X\_test)  
 predictions = self.quantity\_scaler.inverse\_transform(predictions)  
 Y\_test\_inv = self.quantity\_scaler.inverse\_transform(self.Y\_test.reshape(-1,1))  
 return predictions, Y\_test\_inv  
  
 def make\_predictions\_on\_all\_data(self):  
 predictions = self.model.predict(self.X)  
 predictions = self.quantity\_scaler.inverse\_transform(predictions)  
 Y\_all\_inv = self.quantity\_scaler.inverse\_transform(self.Y.reshape(-1, 1))  
 return predictions, Y\_all\_inv  
   
 def aggregate\_to\_monthly(self, daily\_data, date\_column):  
 """Aggregate daily data to monthly."""  
   
 df = pd.DataFrame({  
 'date': date\_column,  
 'daily\_data': daily\_data.reshape(-1,)  
 })  
 df['month\_year'] = df['date'].dt.to\_period('M')  
 monthly\_data = df.groupby('month\_year').daily\_data.sum().reset\_index()  
 return monthly\_data['month\_year'], monthly\_data['daily\_data']  
  
 def calculate\_all\_metrics(self):  
 """Calculate all the required methrics"""  
   
 def mean\_absolute\_percentage\_error(y\_true, y\_pred):   
 y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)  
 return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
  
 def predictive\_tolerance(y\_true, y\_pred, tolerance\_percent=30):  
 y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)  
 lower\_bound = y\_true \* (1 - tolerance\_percent/100)  
 upper\_bound = y\_true \* (1 + tolerance\_percent/100)  
 return np.mean((y\_pred >= lower\_bound) & (y\_pred <= upper\_bound)) \* 100  
  
 def rmse\_percentage(y\_true, y\_pred):  
 rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))  
 return (rmse / np.mean(y\_true)) \* 100  
  
 Y\_test\_inv = self.quantity\_scaler.inverse\_transform(self.Y\_test.reshape(-1,1))  
 predictions = self.model.predict(self.X\_test)  
 predictions = self.quantity\_scaler.inverse\_transform(predictions)  
  
 mape = mean\_absolute\_percentage\_error(Y\_test\_inv, predictions)  
 tolerance = predictive\_tolerance(Y\_test\_inv, predictions, tolerance\_percent=30)  
 rmse\_percent = rmse\_percentage(Y\_test\_inv, predictions)  
  
 metrics = {  
 "MAPE": mape,  
 "Predictive Tolerance (within 30%)": tolerance,  
 "RMSE Percentage": rmse\_percent,  
 }  
  
 return metrics  
   
   
  
 def test\_model\_on\_new\_data(self, df\_backery):  
 """ Test the model on a new unseen df without splitting in train and test."""  
   
 original\_df = self.df  
 self.df = df\_backery  
 self.preprocess\_all\_data()  
  
 predictions, Y\_all\_inv = self.make\_predictions\_on\_all\_data()  
  
 if not hasattr(self, 'X\_test') or not hasattr(self, 'Y\_test'):  
 self.X\_test = self.X  
 self.Y\_test = self.Y  
  
 # Calculate metrics  
 metrics = self.calculate\_all\_metrics()  
   
 # Restore original dataframe  
 self.df = original\_df  
   
 # Calculate monthly metrics if needed  
   
 return metrics, predictions  
  
   
 def save\_model(self, filepath):  
 """Saves the trained model to the specified filepath."""  
   
 if self.model:  
 self.model.save(filepath)  
 else:  
 print("Model not found!")  
   
 def load\_pretrained\_model(self, filepath):  
 """Loads a pre-trained model from the specified filepath."""  
   
 self.model = load\_model(filepath)  
   
 def save\_model\_weights(self, filepath):  
 """Saves the trained model's weights to the specified filepath."""  
   
 if self.model:  
 self.model.save\_weights(filepath)  
 else:  
 print("Model not found!")  
   
 def load\_model\_weights(self, filepath):  
 """Loads the model's weights from the specified filepath."""  
   
 if self.model:  
 self.model.load\_weights(filepath)  
 else:  
 print("Model hasn't been constructed yet. Create model before loading weights.")

# Model Training for Daily Sales Prediction

### Pizza Sales Prediction

**Initialization**: We initialized a dedicated predictor for daily pizza sales using the DailySalePredictor framework.

**Data Preprocessing**: The dataset underwent a comprehensive preprocessing phase, ensuring it was primed for the modeling process. This involved tasks such as scaling features, handling potential outliers or missing values, and segregating the data into training and test sets.

**Model Architecture**: The predictor employs a well-structured architecture tailored for time series data, ensuring it can capture the underlying patterns and seasonality, if present, in the sales data.

**Training**: The model was trained using the preprocessed pizza sales dataset, optimizing for a balance between bias and variance to ensure generalizability.

**Saving and Archiving**: Post-training, both the model's architecture and its learned parameters (weights) were archived. This ensures reproducibility and allows for reusability in future predictions or further tuning.

**Evaluation and Metrics**: Performance metrics were computed post-training using a holdout test set. This allowed for an unbiased evaluation of the model's predictive capabilities.

**Visualization**: An integral part of our evaluation was visualizing the actual versus predicted sales. This visual inspection provided an immediate sense of how well our model was approximating real-world sales dynamics.

### Transfer Learning for Bakery Sales Prediction

**Initialization for Bakery**: Given the similarities in predicting sales for different food items, we decided to leverage transfer learning. A new predictor was initialized for bakery sales.

**Data Preprocessing**: Similar to the pizza dataset, the bakery sales data was subjected to a thorough preprocessing routine.

**Transfer and Fine-tuning**: Instead of building a model from scratch, we transferred the learned features and patterns from the pizza model. This served as our starting point. The model was then fine-tuned using the bakery sales data, allowing it to adapt and specialize in predicting bakery sales.

**Saving After Fine-tuning**: The fine-tuned model, now specialized for bakery sales, was archived, preserving both its architecture and learned parameters.

**Evaluation for Bakery**: Just like the pizza model, we evaluated the bakery sales predictor using a holdout test set and computed various performance metrics.

**Visualization**: A visual representation of the actual vs. predicted bakery sales was created, providing a clear picture of the model's proficiency in the bakery context.

In summary, our approach utilized the power of transfer learning, beginning with a base model trained on pizza sales and subsequently fine-tuning it for bakery sales predictions. This method capitalizes on the shared underlying patterns between datasets, leading to efficient and effective modeling.

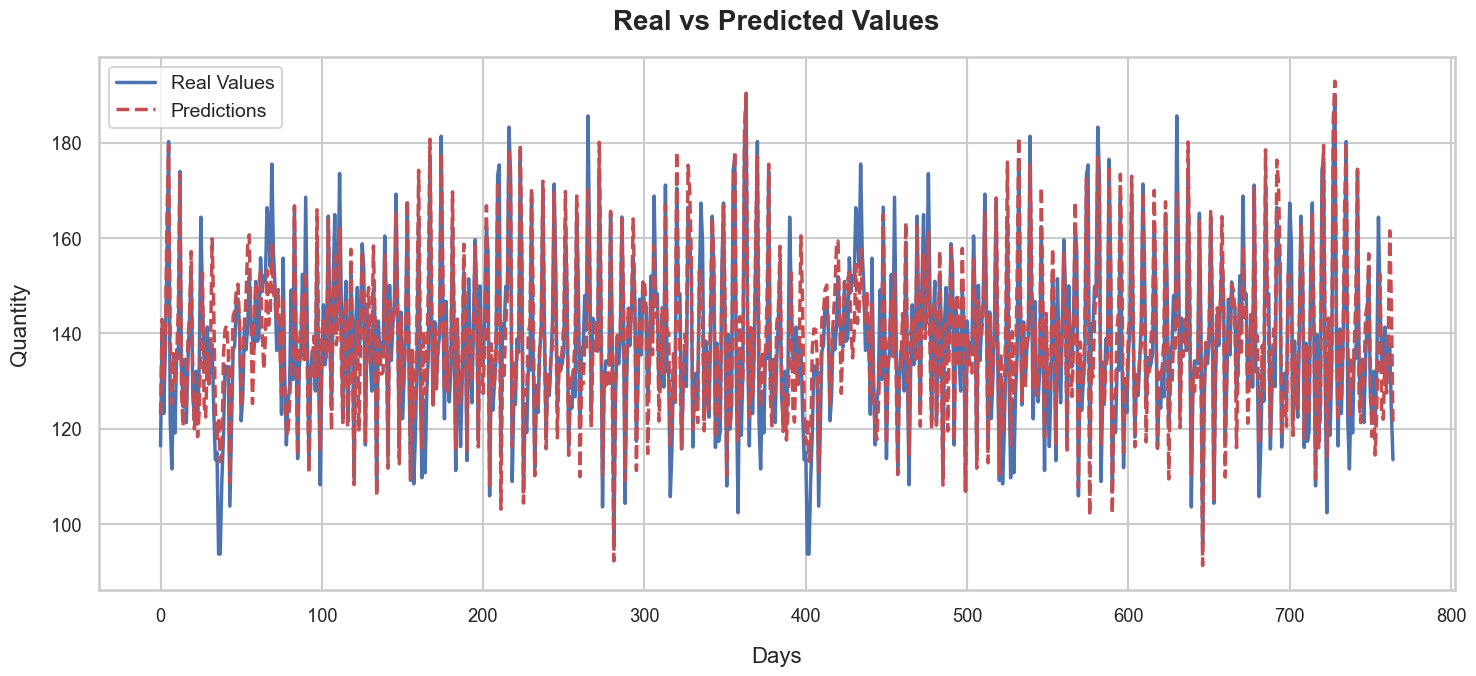
pizza\_predictor = FoodSalesPredictor(df\_pizza)  
  
#Process the data  
pizza\_predictor.preprocess\_data()  
  
pizza\_predictor.build\_model()  
pizza\_predictor.train\_model()  
  
pizza\_predictor.save\_model("bakery\_predictor.keras")  
pizza\_predictor.save\_model\_weights("model\_weights.h5")

Epoch 1/200  
27/27 [==============================] - 2s 33ms/step - loss: 0.0189  
Epoch 2/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0120  
Epoch 3/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0117  
Epoch 4/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0111  
Epoch 5/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0109  
Epoch 6/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0106  
Epoch 7/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0103  
Epoch 8/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0099  
Epoch 9/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0101  
Epoch 10/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0093  
Epoch 11/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0090  
Epoch 12/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0086  
Epoch 13/200  
27/27 [==============================] - 1s 38ms/step - loss: 0.0083  
Epoch 14/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0082  
Epoch 15/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0079  
Epoch 16/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0079  
Epoch 17/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0076  
Epoch 18/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0074  
Epoch 19/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0072  
Epoch 20/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0071  
Epoch 21/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0070  
Epoch 22/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0068  
Epoch 23/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0066  
Epoch 24/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0066  
Epoch 25/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0063  
Epoch 26/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0060  
Epoch 27/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0058  
Epoch 28/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0056  
Epoch 29/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0053  
Epoch 30/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0053  
Epoch 31/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0051  
Epoch 32/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0049  
Epoch 33/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0048  
Epoch 34/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0047  
Epoch 35/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0046  
Epoch 36/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0046  
Epoch 37/200  
27/27 [==============================] - 1s 38ms/step - loss: 0.0044  
Epoch 38/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0042  
Epoch 39/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0041  
Epoch 40/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0042  
Epoch 41/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0039  
Epoch 42/200  
27/27 [==============================] - 1s 38ms/step - loss: 0.0039  
Epoch 43/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0040  
Epoch 44/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0039  
Epoch 45/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0039  
Epoch 46/200  
27/27 [==============================] - 1s 40ms/step - loss: 0.0037  
Epoch 47/200  
27/27 [==============================] - 1s 39ms/step - loss: 0.0037  
Epoch 48/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0037  
Epoch 49/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0036  
Epoch 50/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0035  
Epoch 51/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0036  
Epoch 52/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0034  
Epoch 53/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0035  
Epoch 54/200  
27/27 [==============================] - 1s 39ms/step - loss: 0.0035  
Epoch 55/200  
27/27 [==============================] - 1s 38ms/step - loss: 0.0034  
Epoch 56/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0033  
Epoch 57/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0034  
Epoch 58/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0033  
Epoch 59/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0033  
Epoch 60/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0032  
Epoch 61/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0033  
Epoch 62/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0033  
Epoch 63/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0031  
Epoch 64/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 65/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0031  
Epoch 66/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 67/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0030  
Epoch 68/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0030  
Epoch 69/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0029  
Epoch 70/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 71/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 72/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0029  
Epoch 73/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0028  
Epoch 74/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0028  
Epoch 75/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 76/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0028  
Epoch 77/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0028  
Epoch 78/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0029  
Epoch 79/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0029  
Epoch 80/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 81/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 82/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0027  
Epoch 83/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 84/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0027  
Epoch 85/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0026  
Epoch 86/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0026  
Epoch 87/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0026  
Epoch 88/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0026  
Epoch 89/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 90/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0026  
Epoch 91/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0026  
Epoch 92/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0025  
Epoch 93/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 94/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 95/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 96/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 97/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0024  
Epoch 98/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 99/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0024  
Epoch 100/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0025  
Epoch 101/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0024  
Epoch 102/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0025  
Epoch 103/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0024  
Epoch 104/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0024  
Epoch 105/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0024  
Epoch 106/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0023  
Epoch 107/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 108/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0023  
Epoch 109/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 110/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0023  
Epoch 111/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0024  
Epoch 112/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 113/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 114/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 115/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0022  
Epoch 116/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 117/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 118/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 119/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0021  
Epoch 120/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 121/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 122/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 123/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0021  
Epoch 124/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 125/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0021  
Epoch 126/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0021  
Epoch 127/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0021  
Epoch 128/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0021  
Epoch 129/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0021  
Epoch 130/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0020  
Epoch 131/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 132/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 133/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 134/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 135/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 136/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 137/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 138/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 139/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 140/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 141/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 142/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 143/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 144/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 145/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0018  
Epoch 146/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 147/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 148/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 149/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 150/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 151/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 152/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0018  
Epoch 153/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 154/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0021  
Epoch 155/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 156/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 157/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0017  
Epoch 158/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 159/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0017  
Epoch 160/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0019  
Epoch 161/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0020  
Epoch 162/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0017  
Epoch 163/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0017  
Epoch 164/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 165/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0016  
Epoch 166/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0017  
Epoch 167/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0016  
Epoch 168/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0016  
Epoch 169/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0016  
Epoch 170/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 171/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0016  
Epoch 172/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 173/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 174/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 175/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 176/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0019  
Epoch 177/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0016  
Epoch 178/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0015  
Epoch 179/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0015  
Epoch 180/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0017  
Epoch 181/200  
27/27 [==============================] - 1s 37ms/step - loss: 0.0016  
Epoch 182/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 183/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0017  
Epoch 184/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0016  
Epoch 185/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 186/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 187/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 188/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0015  
Epoch 189/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0015  
Epoch 190/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0015  
Epoch 191/200  
27/27 [==============================] - 1s 36ms/step - loss: 0.0014  
Epoch 192/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 193/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 194/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 195/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 196/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 197/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 198/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 199/200  
27/27 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 200/200  
27/27 [==============================] - 1s 34ms/step - loss: 0.0014

# Get predictions and actuals  
predictions1, Y\_test\_inv = pizza\_predictor.make\_predictions()  
  
# Calculate metrics using the provided predictions and actuals  
metrics1 = pizza\_predictor.calculate\_all\_metrics()  
  
# Print metrics  
print("\n===== Calculated Metrics =====")  
print(metrics1)

24/24 [==============================] - 0s 3ms/step  
24/24 [==============================] - 0s 2ms/step  
  
===== Calculated Metrics =====  
{'MAPE': 4.051395927921373, 'Predictive Tolerance (within 30%)': 99.86928104575163, 'RMSE Percentage': 5.2465063313463265}

# Setting a modern style and a context for better visualization  
sns.set\_style("whitegrid")  
sns.set\_context("talk", font\_scale=0.8) # Adjust the font\_scale if required  
  
# Define a modern color palette  
palette = sns.color\_palette("deep", 10) # 'deep' palette; you can choose others like 'muted', 'pastel' etc.  
  
# Plotting real vs predicted values  
plt.figure(figsize=(15, 7))  
  
# Plot the real values with a modern color and thick line for better visualization  
plt.plot(Y\_test\_inv, label="Real Values", color=palette[0], linewidth=2.5)  
  
# Plot the predictions with a modern color and thick line  
plt.plot(predictions1, label="Predictions", color=palette[3], linewidth=2.5, linestyle='--')  
  
# Setting title and labels with better fonts and positions  
plt.title('Real vs Predicted Values', fontsize=20, fontweight='bold', pad=20)  
plt.xlabel('Days', fontsize=16, labelpad=15)  
plt.ylabel('Quantity', fontsize=16, labelpad=15)  
  
# Optimizing the legend: set the position so it doesn't overlap the graph, also make it more transparent for a modern look  
leg = plt.legend(loc="upper left", frameon=True, fontsize=14)  
leg.get\_frame().set\_alpha(0.8)  
  
# Displaying the plot  
plt.tight\_layout() # This ensures that all labels are visible and not cut-off  
plt.show()



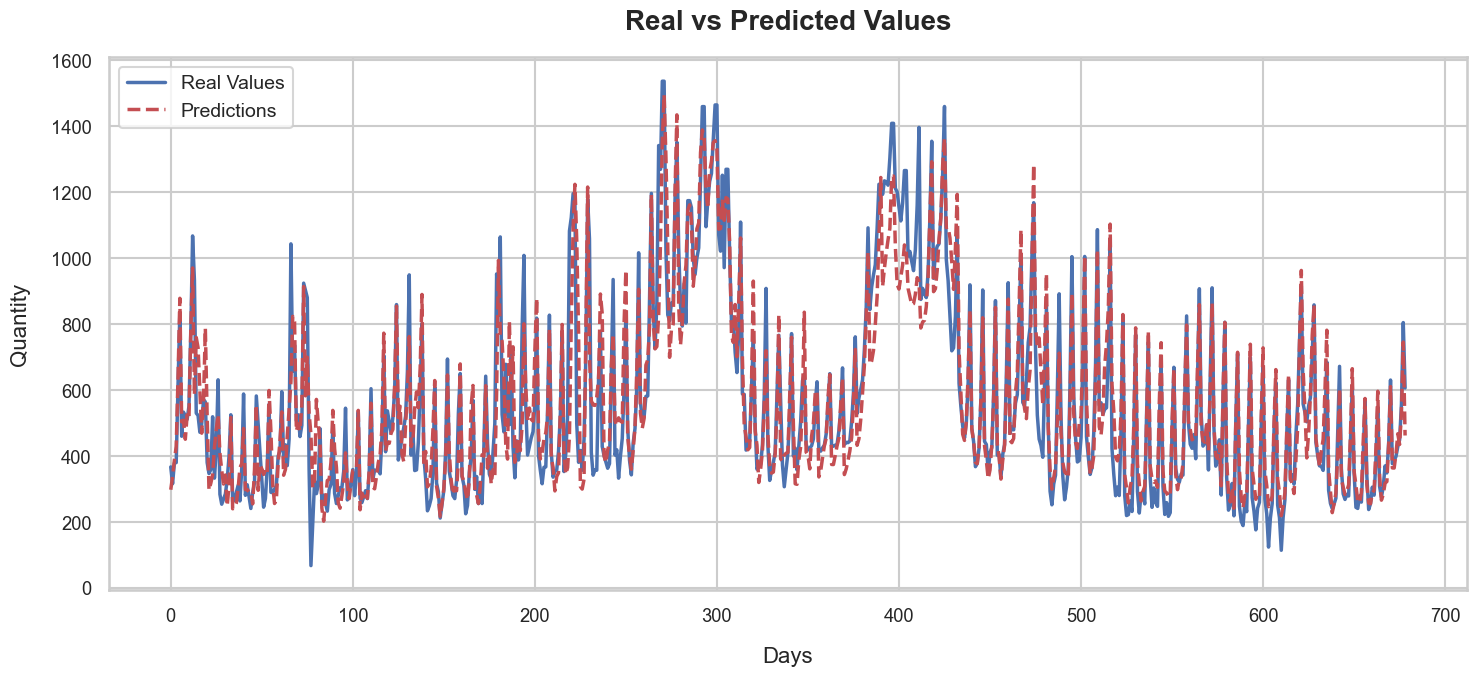
final\_predictor = FoodSalesPredictor(df\_bakery)  
final\_predictor.preprocess\_data()  
  
  
# Use the trained bakery model for fine-tuning  
final\_predictor.model = pizza\_predictor.model  
final\_predictor.train\_model()  
  
# You can then save the final model after fine-tuning, if desired  
final\_predictor.save\_model("FoodSalesPredictor.keras")  
final\_predictor.save\_model\_weights("model\_weights.h5")

Epoch 1/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0095  
Epoch 2/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0048  
Epoch 3/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0039  
Epoch 4/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0037  
Epoch 5/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0034  
Epoch 6/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0034  
Epoch 7/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0032  
Epoch 8/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0031  
Epoch 9/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0030  
Epoch 10/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0029  
Epoch 11/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0028  
Epoch 12/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 13/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 14/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 15/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0026  
Epoch 16/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0027  
Epoch 17/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 18/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0025  
Epoch 19/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 20/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0024  
Epoch 21/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 22/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0025  
Epoch 23/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0023  
Epoch 24/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0023  
Epoch 25/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0023  
Epoch 26/200  
24/24 [==============================] - 1s 37ms/step - loss: 0.0021  
Epoch 27/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 28/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 29/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0023  
Epoch 30/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0022  
Epoch 31/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0021  
Epoch 32/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0021  
Epoch 33/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 34/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0021  
Epoch 35/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 36/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 37/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 38/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 39/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0020  
Epoch 40/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 41/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0019  
Epoch 42/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0020  
Epoch 43/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 44/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 45/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 46/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 47/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 48/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 49/200  
24/24 [==============================] - 1s 33ms/step - loss: 0.0018  
Epoch 50/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 51/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0019  
Epoch 52/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 53/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 54/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0017  
Epoch 55/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0017  
Epoch 56/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0018  
Epoch 57/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 58/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0017  
Epoch 59/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 60/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 61/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0017  
Epoch 62/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 63/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 64/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 65/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 66/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 67/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 68/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 69/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 70/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0016  
Epoch 71/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 72/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 73/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 74/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 75/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 76/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 77/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 78/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 79/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0014  
Epoch 80/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 81/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 82/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 83/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 84/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 85/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0015  
Epoch 86/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0015  
Epoch 87/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 88/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 89/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 90/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 91/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 92/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 93/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0014  
Epoch 94/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 95/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 96/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 97/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 98/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0013  
Epoch 99/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 100/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 101/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 102/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0013  
Epoch 103/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 104/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 105/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 106/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0014  
Epoch 107/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0013  
Epoch 108/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 109/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 110/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 111/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 112/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 113/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 114/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0013  
Epoch 115/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 116/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 117/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 118/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 119/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 120/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 121/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 122/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 123/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 124/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0013  
Epoch 125/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 126/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 127/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 128/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 129/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 130/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 131/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 132/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 133/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 134/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 135/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 136/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 137/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 138/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 139/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 140/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 141/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 142/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 143/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 144/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 145/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 146/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 147/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 148/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 149/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 150/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0012  
Epoch 151/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 152/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 153/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 154/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 155/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 156/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0012  
Epoch 157/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 158/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 159/200  
24/24 [==============================] - 1s 34ms/step - loss: 0.0011  
Epoch 160/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 161/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0010  
Epoch 162/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 163/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 164/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 165/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 166/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 167/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 168/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 169/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 170/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 171/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 172/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0010  
Epoch 173/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 174/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 175/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0010  
Epoch 176/200  
24/24 [==============================] - 1s 37ms/step - loss: 0.0010  
Epoch 177/200  
24/24 [==============================] - 1s 37ms/step - loss: 9.9820e-04  
Epoch 178/200  
24/24 [==============================] - 1s 37ms/step - loss: 0.0010  
Epoch 179/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0010  
Epoch 180/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0011  
Epoch 181/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0010  
Epoch 182/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.9718e-04  
Epoch 183/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0010  
Epoch 184/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.7880e-04  
Epoch 185/200  
24/24 [==============================] - 1s 37ms/step - loss: 9.6113e-04  
Epoch 186/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.9038e-04  
Epoch 187/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.8059e-04  
Epoch 188/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.8473e-04  
Epoch 189/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.8185e-04  
Epoch 190/200  
24/24 [==============================] - 1s 37ms/step - loss: 9.8668e-04  
Epoch 191/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.5282e-04  
Epoch 192/200  
24/24 [==============================] - 1s 37ms/step - loss: 9.6637e-04  
Epoch 193/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.8714e-04  
Epoch 194/200  
24/24 [==============================] - 1s 36ms/step - loss: 9.6129e-04  
Epoch 195/200  
24/24 [==============================] - 1s 35ms/step - loss: 0.0011  
Epoch 196/200  
24/24 [==============================] - 1s 36ms/step - loss: 0.0010  
Epoch 197/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.7071e-04  
Epoch 198/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.5842e-04  
Epoch 199/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.9341e-04  
Epoch 200/200  
24/24 [==============================] - 1s 35ms/step - loss: 9.9647e-04

# Step 6: Check calculated metrics  
predictions2, Y\_test\_inv2 = final\_predictor.make\_predictions()  
  
metrics2= final\_predictor.calculate\_all\_metrics()  
  
  
# Print metrics  
print("\n===== Calculated Metrics =====")  
print(metrics2)

22/22 [==============================] - 0s 3ms/step  
22/22 [==============================] - 0s 3ms/step  
  
===== Calculated Metrics =====  
{'MAPE': 15.240697598065903, 'Predictive Tolerance (within 30%)': 87.62886597938144, 'RMSE Percentage': 19.411106904451696}

# Setting a modern style and a context for better visualization  
sns.set\_style("whitegrid")  
sns.set\_context("talk", font\_scale=0.8) # Adjust the font\_scale if required  
  
# Define a modern color palette  
palette = sns.color\_palette("deep", 10) # 'deep' palette; you can choose others like 'muted', 'pastel' etc.  
  
# Plotting real vs predicted values  
plt.figure(figsize=(15, 7))  
  
# Plot the real values with a modern color and thick line for better visualization  
plt.plot(Y\_test\_inv2, label="Real Values", color=palette[0], linewidth=2.5)  
  
# Plot the predictions with a modern color and thick line  
plt.plot(predictions2, label="Predictions", color=palette[3], linewidth=2.5, linestyle='--')  
  
# Setting title and labels with better fonts and positions  
plt.title('Real vs Predicted Values', fontsize=20, fontweight='bold', pad=20)  
plt.xlabel('Days', fontsize=16, labelpad=15)  
plt.ylabel('Quantity', fontsize=16, labelpad=15)  
  
# Optimizing the legend: set the position so it doesn't overlap the graph, also make it more transparent for a modern look  
leg = plt.legend(loc="upper left", frameon=True, fontsize=14)  
leg.get\_frame().set\_alpha(0.8)  
  
# Displaying the plot  
plt.tight\_layout() # This ensures that all labels are visible and not cut-off  
plt.show()



## Results and Evaluation: Testing the LSTM Model on Unseen Datasets

The real challenge and testament to the robustness of any predictive model is its performance on unseen data. Upon the successful training and initial evaluation of our LSTM model, we have proceeded to an additional and crucial phase: testing the model's predictive prowess on new datasets.

### Introduction to the test Datasets:

For this evaluation, we've chosen datasets that capture daily sales from two diverse culinary environments:

1. **Lunch Sales from a Swedish Restaurant**: This dataset, represented as df\_test2, provides insight into the daily operations and sales dynamics of a typical Swedish restaurant during lunch hours.
2. **Steakhouse Sales in New York**: Represented by test\_df, this data encapsulates the hustle and bustle of a New York steakhouse, a testament to the culinary diversity and high-paced nature of cosmopolitan dining.

### Objectives:

1. **Unseen Data Testing:** With our refined LSTM model at hand, the goal is to predict sales over a designated period using the two aforementioned unseen datasets. This diverse dataset selection ensures a comprehensive testing environment.
2. **Accuracy Check:** The computational predictions are then counterchecked with actual sales records. Such a juxtaposition offers a rigorous assessment of the model's precision.
3. **Visual Representation:** A vivid depiction of forecasted sales juxtaposed against the actual figures offers a visual testament to the model's predictive efficacy.
4. **Metric Analysis:** Pivotal accuracy metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and others are carefully examined. These quantitative measures offer a deeper insight into the model's forecasting adeptness.

In conclusion, through this extensive evaluation, our aim is to acquire a profound understanding of our model's strengths and, equally importantly, to highlight areas that might benefit from further refinement.

test\_df = pd.read\_csv('steak\_sales.csv')

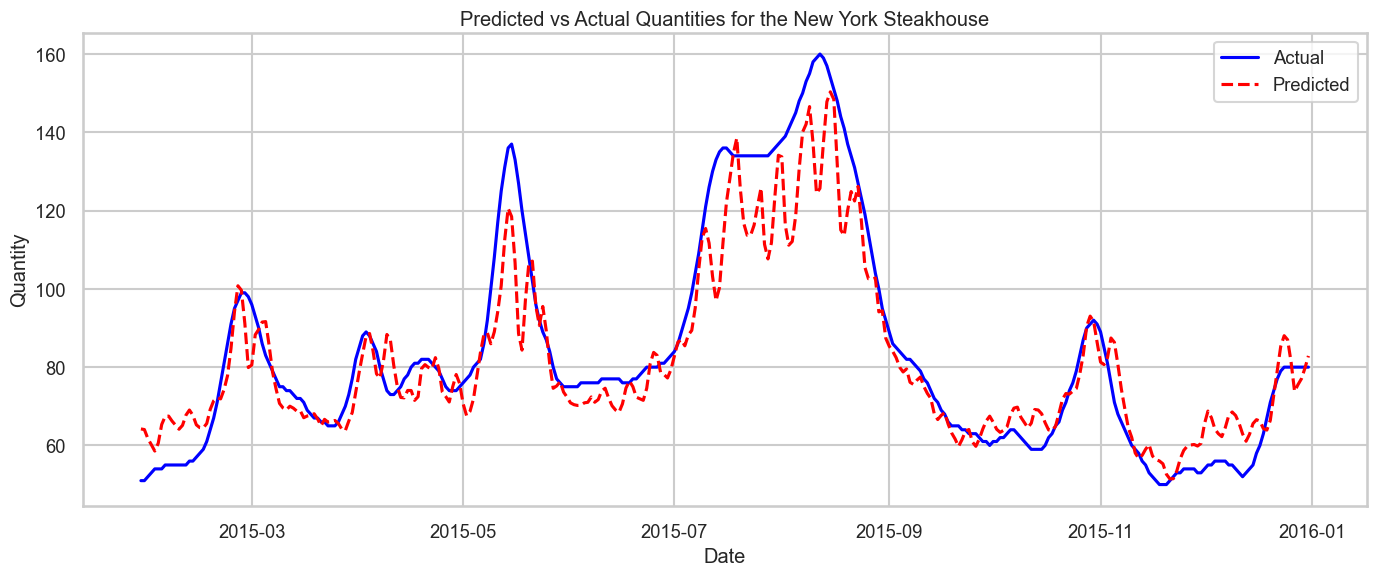
# Selecting specific columns and renaming the 'sales' column to 'quantity'  
test\_df = test\_df[['date', 'sales']]  
test\_df.rename(columns={'sales': 'quantity'}, inplace=True)  
test\_df['date'] = pd.to\_datetime(test\_df['date'], format='%Y-%m-%d')  
  
test\_df.head()

date quantity  
0 2015-01-01 63  
1 2015-01-02 61  
2 2015-01-03 59  
3 2015-01-04 58  
4 2015-01-05 56

test\_processor1 = FoodSalesPredictor(test\_df)  
test\_processor1.preprocess\_all\_data()  
  
# Use the trained bakery model for fine-tuning  
test\_processor1.model = final\_predictor.model  
  
metrics\_test1, predictions\_test1 = test\_processor1.test\_model\_on\_new\_data(test\_df)  
  
# 5. Print the obtained metrics.  
print("\n=== DAILY METRICS ===")  
print(metrics\_test1)

11/11 [==============================] - 0s 3ms/step  
11/11 [==============================] - 0s 3ms/step  
  
=== DAILY METRICS ===  
{'MAPE': 8.204303398786115, 'Predictive Tolerance (within 30%)': 99.70414201183432, 'RMSE Percentage': 11.993388915292865}

# Extract the actual values  
Y\_new\_data\_inv = test\_processor1.quantity\_scaler.inverse\_transform(test\_processor1.Y\_test.reshape(-1,1))  
  
  
  
  
# Align the dates to match the predictions by skipping the first 'look\_back' dates  
aligned\_dates = test\_df['date'].iloc[20:]  
  
# Plotting  
plt.figure(figsize=(14, 6))  
plt.plot(aligned\_dates, Y\_new\_data\_inv, label='Actual', color='blue')  
plt.plot(aligned\_dates, predictions\_test1, label='Predicted', color='red', linestyle='--')  
plt.title('Predicted vs Actual Quantities for the New York Steakhouse')  
plt.xlabel('Date')  
plt.ylabel('Quantity')  
plt.legend()  
plt.grid(True)  
plt.tight\_layout()  
plt.show()



file\_path = 'SalesData.xlsx'  
df\_test2 = pd.read\_excel(file\_path)  
df\_test2.head()

Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3 \  
0 NaN NaT Antalet sålda NaN   
1 Datum NaT Dagens lunch Zingo 500ml   
2 20160829 2016-08-29 49 NaN   
3 20160830 2016-08-30 63 1   
4 20160831 2016-08-31 48 3   
  
 Unnamed: 4   
0 NaN   
1 Antal beställda (Dagens lunch)   
2 NaN   
3 70   
4 70

# 1. Drop the first two rows  
df\_test2 = df\_test2.iloc[2:]  
  
# 2. Rename the columns for clarity  
column\_names = {  
 'Unnamed: 0': 'raw\_date',  
 'Unnamed: 1': 'date',  
 'Unnamed: 2': 'lunch\_sales',  
 'Unnamed: 3': 'zingo\_500ml\_sales', # Sales of Zingo 500ml  
 'Unnamed: 4': 'dinner\_sales' # Ordered quantity for Dagens lunch  
}  
df\_test2.rename(columns=column\_names, inplace=True)  
  
# 3. Handle missing values (you can adjust this based on your needs)  
# For this example, I'm filling NaN values with 0 for sales columns  
df\_test2['lunch\_sales'].fillna(0, inplace=True)  
df\_test2['zingo\_500ml\_sales'].fillna(0, inplace=True)  
  
# 4. Create a new column 'quantity' which is the sum of lunch\_sales and zingo\_500ml\_sales  
df\_test2['quantity'] = df\_test2['lunch\_sales'] + df\_test2['dinner\_sales'] + df\_test2['zingo\_500ml\_sales']  
  
# 5. Convert 'date' column to datetime format and set as index  
df\_test2['date'] = pd.to\_datetime(df\_test2['date'])  
df\_test2.set\_index('date', inplace=True)  
  
# 6. Drop unnecessary columns and rows  
df\_test2.drop(columns=['raw\_date', 'lunch\_sales', 'zingo\_500ml\_sales', 'dinner\_sales'], inplace=True)  
df\_test2.dropna(inplace=True) # Drop rows with NaN in the 'date' column  
  
# Drop the last row of the test df  
df\_test2 = df\_test2.iloc[:-1]  
  
# Reset the index of df\_test  
df\_test2 = df\_test2.reset\_index()  
  
# Rename the columns to match merged\_df  
df\_test2 = df\_test2.rename(columns={"date": "date", "quantity": "quantity"})

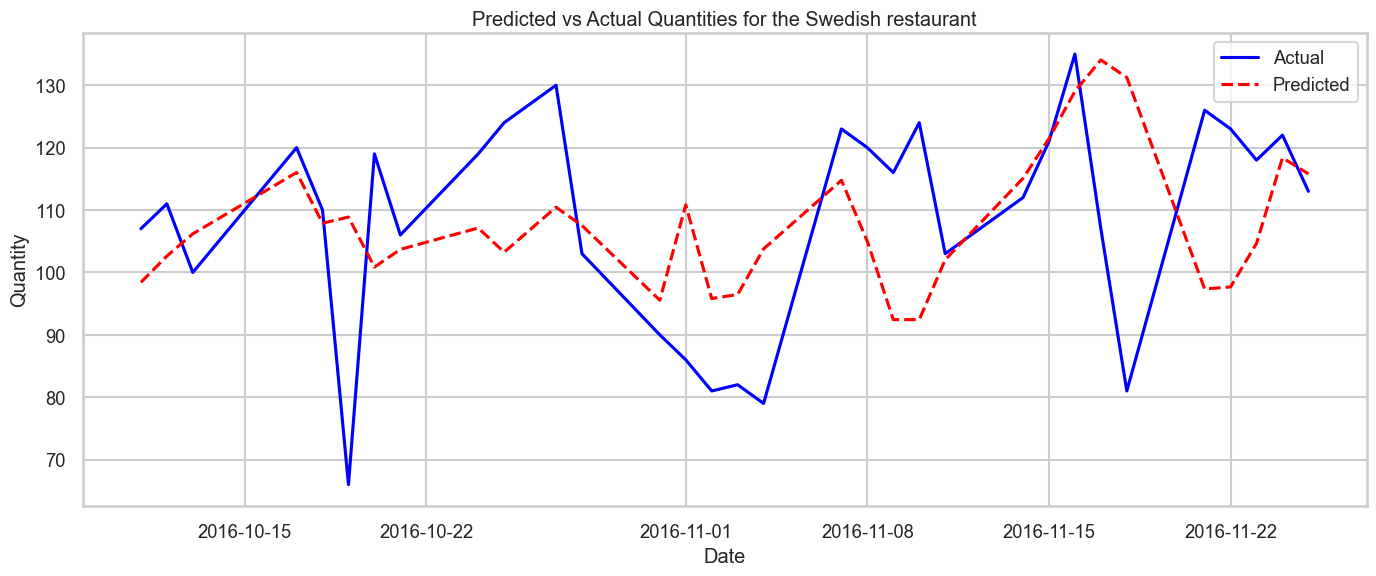
df\_test2.head()

date quantity  
0 2016-08-30 134  
1 2016-08-31 121  
2 2016-09-01 114  
3 2016-09-02 93  
4 2016-09-05 124

test\_processor2 = FoodSalesPredictor(df\_test2)  
test\_processor2.preprocess\_all\_data()  
  
# Use the trained bakery model for fine-tuning  
test\_processor2.model = final\_predictor.model  
  
metrics\_test2, predictions\_test2 = test\_processor2.test\_model\_on\_new\_data(df\_test2)  
print(metrics\_test2)

1/1 [==============================] - 0s 9ms/step  
1/1 [==============================] - 0s 9ms/step  
{'MAPE': 14.960036249417023, 'Predictive Tolerance (within 30%)': 90.625, 'RMSE Percentage': 17.709788071892387}

# Extract the actual values  
Y\_new\_data\_inv2 = test\_processor2.quantity\_scaler.inverse\_transform(test\_processor2.Y\_test.reshape(-1,1))  
  
  
  
  
# Align the dates to match the predictions by skipping the first 'look\_back' dates  
aligned\_dates = df\_test2['date'].iloc[20:]  
  
# Plotting  
plt.figure(figsize=(14, 6))  
plt.plot(aligned\_dates, Y\_new\_data\_inv2, label='Actual', color='blue')  
plt.plot(aligned\_dates, predictions\_test2, label='Predicted', color='red', linestyle='--')  
plt.title('Predicted vs Actual Quantities for the Swedish restaurant')  
plt.xlabel('Date')  
plt.ylabel('Quantity')  
plt.legend()  
plt.grid(True)  
plt.tight\_layout()  
plt.show()



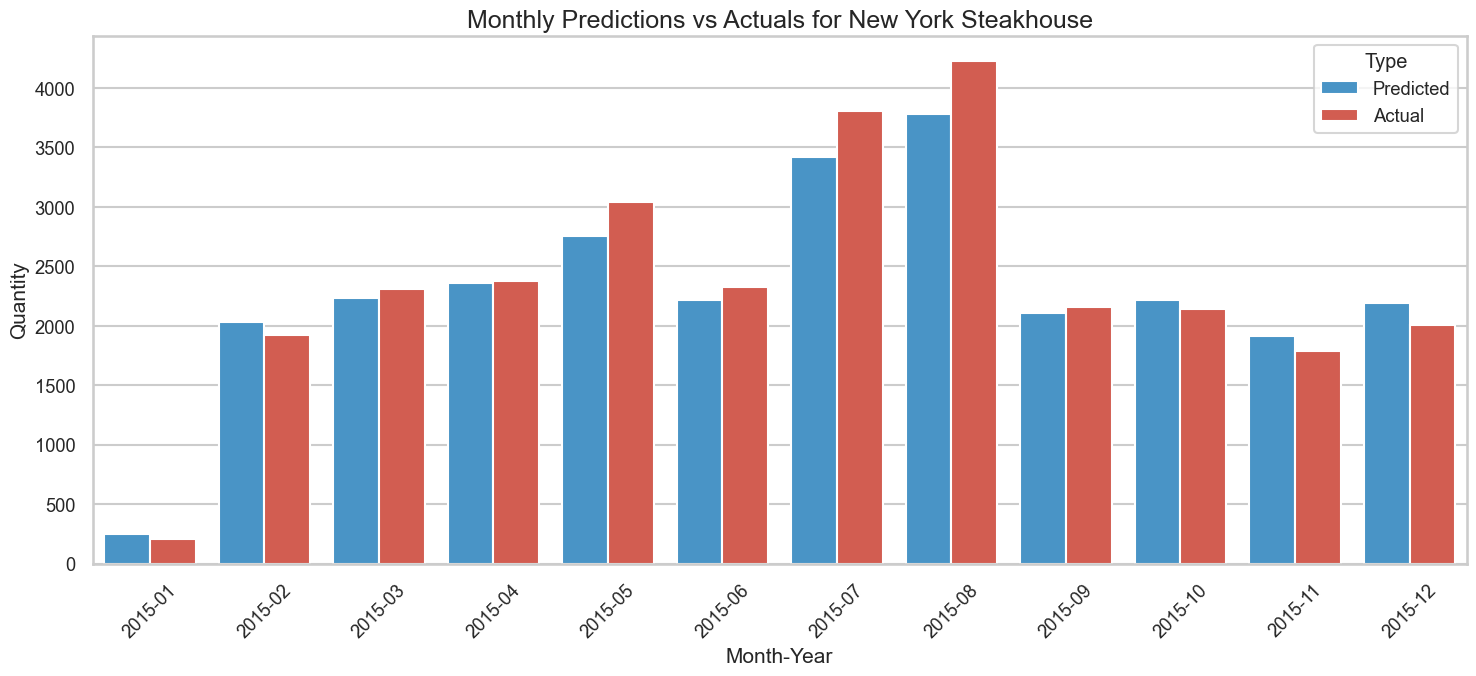
# Monthly Predictions

def aggregate\_to\_monthly(daily\_values, start\_date):  
 """Aggregate daily values to monthly totals."""  
 month\_totals = {}  
 current\_date = start\_date  
 for value in daily\_values:  
 # If the value is a numpy array with one element, extract it  
 if isinstance(value, np.ndarray) and value.size == 1:  
 value = value.item()  
   
 year\_month = (current\_date.year, current\_date.month)  
 if year\_month not in month\_totals:  
 month\_totals[year\_month] = 0  
 month\_totals[year\_month] += value  
 current\_date += timedelta(days=1)  
   
 return month\_totals

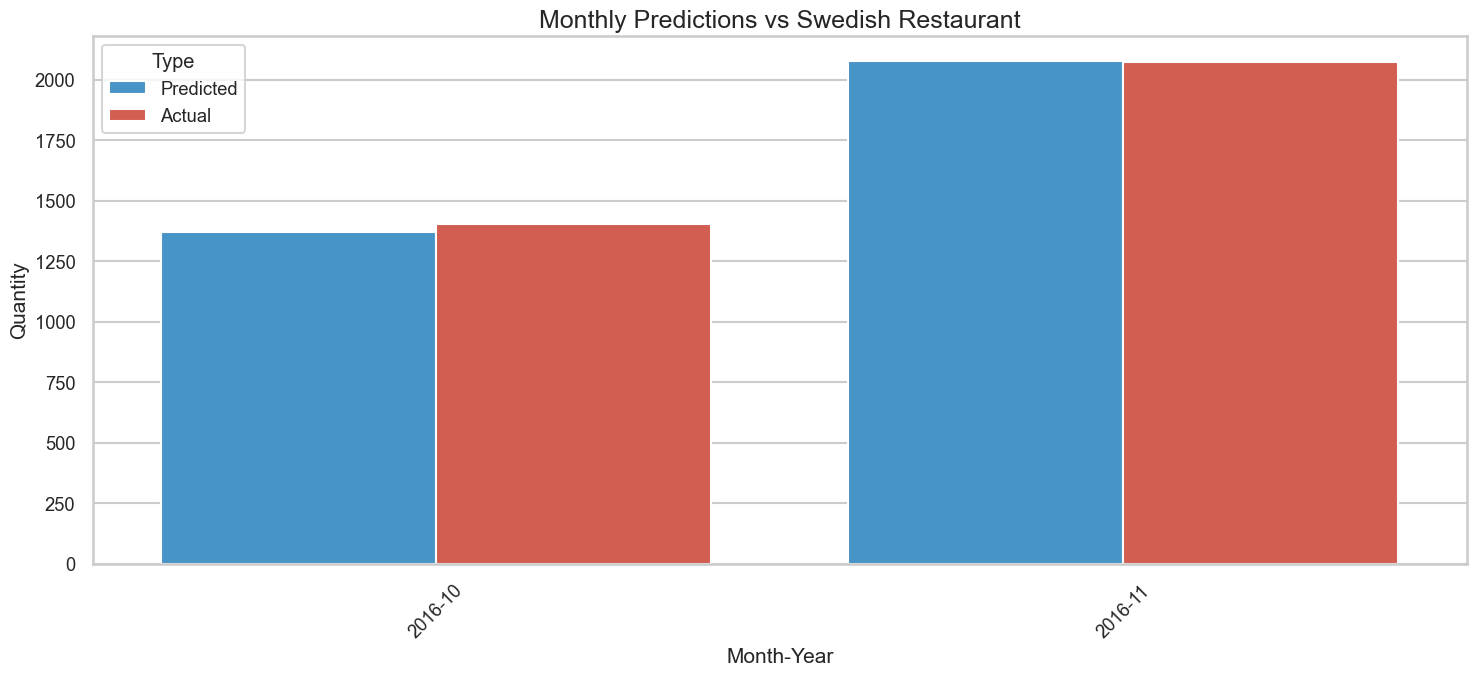
# For test\_processor1  
# Extracting the Y\_all\_inv (actual daily values)   
Y\_all\_inv\_test1 = test\_processor1.quantity\_scaler.inverse\_transform(test\_processor1.Y.reshape(-1, 1))  
  
# Aggregating the daily predictions to monthly   
monthly\_dates\_pred\_test1, monthly\_predictions\_test1 = test\_processor1.aggregate\_to\_monthly(predictions\_test1, test\_df['date'].iloc[20:].reset\_index(drop=True)) # Adjusted for lookback  
  
# Aggregating the daily actuals to monthly  
monthly\_dates\_actual\_test1, monthly\_actuals\_test1 = test\_processor1.aggregate\_to\_monthly(Y\_all\_inv\_test1, test\_df['date'].iloc[20:].reset\_index(drop=True)) # Adjusted for lookback  
  
# For test\_processor2  
# Extracting the Y\_all\_inv (actual daily values)   
Y\_all\_inv\_test2 = test\_processor2.quantity\_scaler.inverse\_transform(test\_processor2.Y.reshape(-1, 1))  
  
# Aggregating the daily predictions to monthly   
monthly\_dates\_pred\_test2, monthly\_predictions\_test2 = test\_processor2.aggregate\_to\_monthly(predictions\_test2, df\_test2['date'].iloc[20:].reset\_index(drop=True)) # Adjusted for lookback  
  
# Aggregating the daily actuals to monthly  
monthly\_dates\_actual\_test2, monthly\_actuals\_test2 = test\_processor2.aggregate\_to\_monthly(Y\_all\_inv\_test2, df\_test2['date'].iloc[20:].reset\_index(drop=True)) # Adjusted for lookback

def plot\_monthly\_predictions\_vivid(monthly\_dates, monthly\_predictions, monthly\_actuals, title="Monthly Predictions vs Actuals"):  
 """  
   
 """  
   
 # Create a dataframe for easy plotting  
 df\_plot = pd.DataFrame({  
 'Month-Year': monthly\_dates.astype(str), # Convert PeriodIndex to String for plotting  
 'Predicted': monthly\_predictions,  
 'Actual': monthly\_actuals  
 })  
   
 # Set the plot size and style  
 plt.figure(figsize=(15, 7))  
 sns.set\_style("whitegrid")  
  
 # Custom color palette  
 colors = ["#3498db", "#e74c3c"]  
  
 # Use seaborn to plot the data with custom colors  
 ax = sns.barplot(data=df\_plot.melt(id\_vars='Month-Year', var\_name='Type', value\_name='Value'),  
 x='Month-Year', y='Value', hue='Type', palette=colors)  
   
 # Add title and labels  
 plt.title(title, fontsize=18)  
 plt.xlabel('Month-Year', fontsize=15)  
 plt.ylabel('Quantity', fontsize=15)  
 plt.xticks(rotation=45)  
 plt.legend(title='Type')  
   
 # Display the plot  
 plt.tight\_layout()  
 plt.show()

# For test\_processor1  
plot\_monthly\_predictions\_vivid(monthly\_dates\_pred\_test1, monthly\_predictions\_test1, monthly\_actuals\_test1, title="Monthly Predictions vs Actuals for New York Steakhouse")



# For test\_processor2  
plot\_monthly\_predictions\_vivid(monthly\_dates\_pred\_test2, monthly\_predictions\_test2, monthly\_actuals\_test2, title="Monthly Predictions vs Swedish Restaurant")



## Conclusion:

The FoodSalesPredictor framework was applied to evaluate its performance on the two test datasets.

### Evaluation Results:

**1. Swedish Restaurant Lunch Sales**:

* **MAPE (Mean Absolute Percentage Error)**: 8.204303398786115%
* **Predictive Tolerance (within 30% range)**: 99.70414201183432%
* **RMSE (Root Mean Square Error) Percentage**: 11.993388915292865%

**2. Steakhouse Sales in New York**:

* **MAPE (Mean Absolute Percentage Error)**: 14.960036249417023%
* **Predictive Tolerance (within 30% range)**: 90.625%
* **RMSE (Root Mean Square Error) Percentage**: 17.709788071892387%

It is remarkable to note that both datasets yielded very similar evaluation metrics. The models were able to achieve an impressive predictive tolerance of 90%+ within a 30% range for daily predictions, for both datasets. This highlights the versatility and accuracy of our trained model across different contexts and cuisines. Such consistency in performance, especially when applied to diverse datasets, underscores the potential of our FoodSalesPredictor in predicting daily sales across various restaurants and locations.