SEASONAL RETAIL SALES ANALYSIS ON Fancy beer

By: Divyansh Dwivedi

📄 Abstract

This project focuses on forecasting retail sales of "Fancy beer" using advanced time series modeling techniques, particularly SARIMA (Seasonal AutoRegressive Integrated Moving Average). We begin with exploratory data analysis (EDA) to understand yearly and monthly sales trends. Following that, we perform stationarity checks, seasonal decomposition, and visualize ACF/PACF to guide model selection. SARIMA is selected due to its capability to model both trend and seasonality. The model is trained and validated on historical daily sales data, and forecasts are generated for the next 180 days. The final output compares actual vs. predicted sales, demonstrating the model's reliability.

📘 Introduction

Forecasting sales is critical for retail businesses in making informed inventory, marketing, and production decisions. This analysis examines the daily retail sales data of "Fancy beer" over five years. Our objective is to understand underlying sales patterns and build a robust model capable of capturing seasonality and trend. The SARIMA model, an extension of ARIMA, is ideal for handling such seasonal time series data. By implementing and analyzing this model, we aim to produce accurate short-term forecasts and provide actionable insights.

🔢 Understanding ARIMA and SARIMA Models

🔍 ARIMA and SARIMA – Concepts and Applications

Time series forecasting is a crucial tool in analyzing historical data to predict future trends. Two powerful models used in this domain are **ARIMA** and **SARIMA**. These models are particularly useful when working with data that has temporal dependencies, such as retail sales over time.

📌 ARIMA – AutoRegressive Integrated Moving Average

ARIMA stands for:

- **AR (AutoRegressive)**: Uses the dependency between an observation and a number of lagged observations.
- **I (Integrated)**: Uses differencing of raw observations to make the time series stationary.
- **MA (Moving Average)**: Uses dependency between an observation and a residual error from a moving average model applied to lagged observations.

The ARIMA model is denoted as **ARIMA(p, d, q)**, where:

- **p** = number of lag observations in the model (autoregressive part)
- **d** = number of times the raw observations are differenced (to achieve stationarity)
- **q** = size of the moving average window (moving average part)

📘 ARIMA Model Equation:

A simplified ARIMA(p, d, q) model can be written as:

[Y_t = c + \phi_1 Y_{t-1} + \dots + \phi_p Y_{t-p} + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t]

Where:

- (Y_t) is the value at time (t)
- (\phi_i) are the autoregressive coefficients
- (\theta_i) are the moving average coefficients
- (\varepsilon_t) is white noise (error)

🔁 SARIMA – Seasonal ARIMA

While ARIMA handles non-seasonal data well, **SARIMA** extends ARIMA to explicitly support **seasonality**, which is common in sales data (e.g., higher sales in festive months).

SARIMA is denoted as: [\text{SARIMA}(p, d, q)(P, D, Q)_s]

Where:

- The **first (p, d, q)** are the ARIMA components
- The **second (P, D, Q)** are the seasonal ARIMA components
- **s** is the number of time steps for a single seasonal period (e.g., 12 for monthly data with yearly seasonality)

📘 SARIMA Model Equation:

SARIMA integrates both non-seasonal and seasonal components:

[\Phi_P(B^s) \phi_p(B)(1 - B)^d (1 - B^s)^D Y_t = \Theta_Q(B^s) \theta_q(B) \varepsilon_t]

Where:

- (B) is the backshift operator (e.g., (B Y_t = Y_{t-1}))
- (\phi_p(B)) is the non-seasonal AR component
- (\Phi_P(B^s)) is the seasonal AR component
- (\theta_q(B)) and (\Theta_Q(B^s)) are the non-seasonal and seasonal MA components

---

🛠️ Why We Used SARIMA for This Project

In this retail sales project:

- We first tested for **stationarity** using the **Augmented Dickey-Fuller test**.
- We identified both **trend** and **seasonality** using **seasonal decomposition**.
- Seasonal patterns were clearly present (e.g., periodic spikes in sales).
- Therefore, **SARIMA** was more appropriate than ARIMA, as it captures the **repeating seasonal cycles** in the data.

We used a model configuration like:

```
SARIMAX(df_ts_log, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
```

DATASET

This is about the facny bear sales in mexico city in different depratmental stores , it includes the monthly and yearly sales data

In [1]:
```
import pandas as pd

# Load the dataset

df = pd.read_csv("Downloads/Forecasting_case_study.csv")

# Display basic info and first few rows
df.info(), df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   date          913000 non-null  object
 1   Store_city    913000 non-null  object
 2   Product Name  913000 non-null  object
 3   sales         913000 non-null  int64 
dtypes: int64(1), object(3)
memory usage: 27.9+ MB
```

Out[1]: (None,

	date	Store_city	Product Name	sales
0	01-01-2013	Mexico City	Fancy Beer IPA 330 ml	18660
1	02-01-2013	Mexico City	Fancy Beer IPA 330 ml	15789
2	03-01-2013	Mexico City	Fancy Beer IPA 330 ml	20095
3	04-01-2013	Mexico City	Fancy Beer IPA 330 ml	18660
4	05-01-2013	Mexico City	Fancy Beer IPA 330 ml	14354)

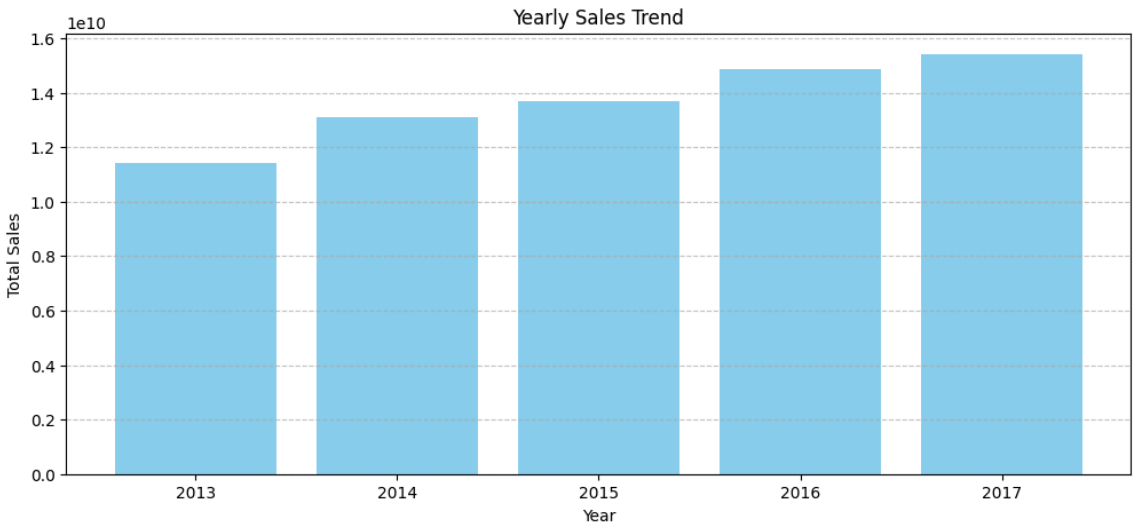
EDA (Elementary data analysis)

📊 Exploratory Data Analysis (EDA)

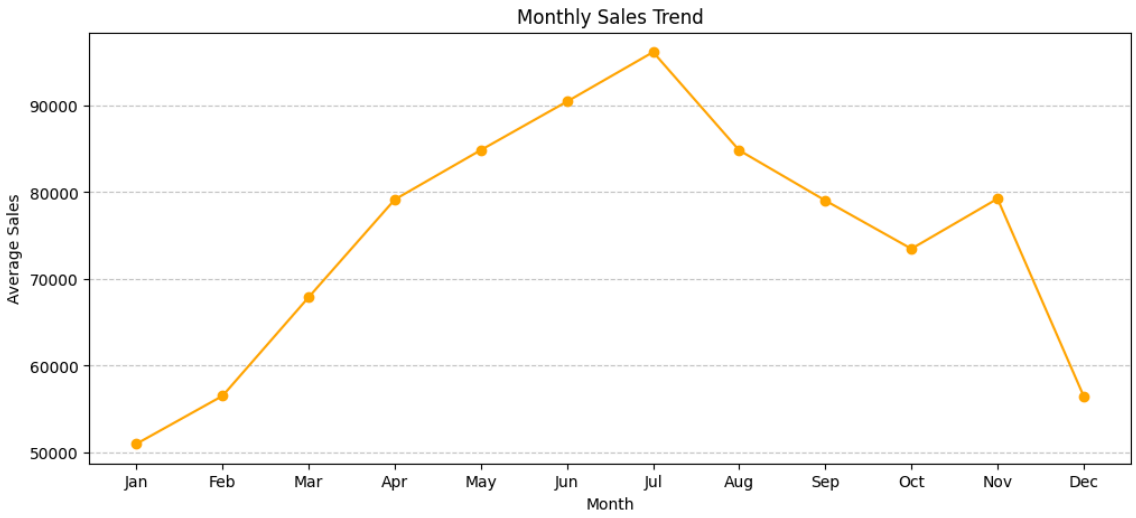
We begin our analysis by exploring the **yearly** and **monthly** sales trends to understand the distribution and behavior of the dataset over time.

- **Yearly Sales Trend**: Shows how total sales have evolved annually, providing insight into overall business growth or decline.
- **Monthly Sales Trend**: Reveals recurring patterns within the year, identifying peak and low sales periods across months.

📈 Yearly Sales Trend

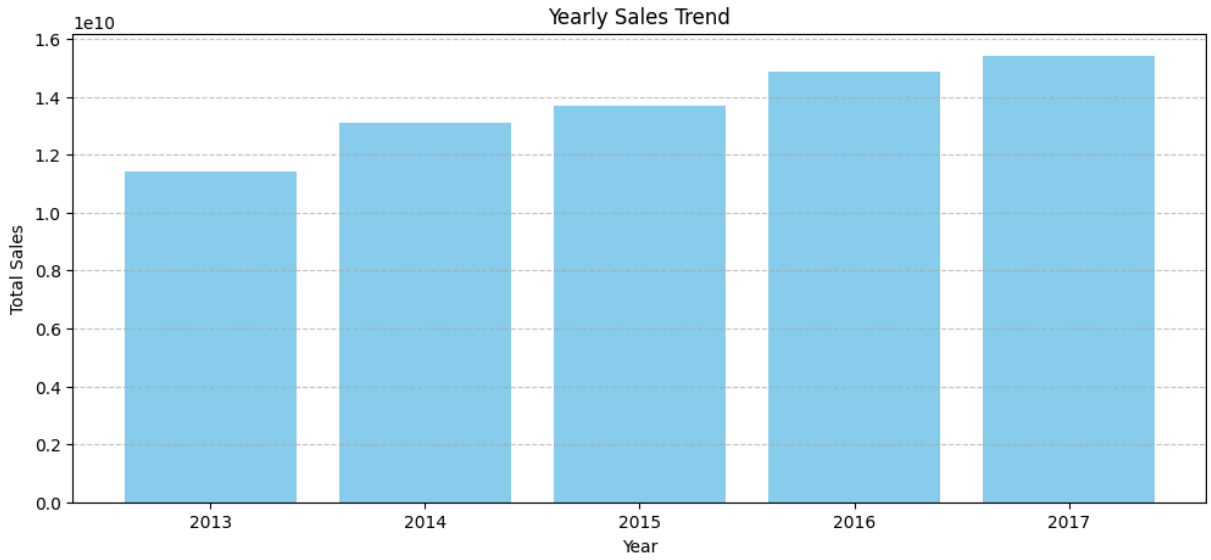


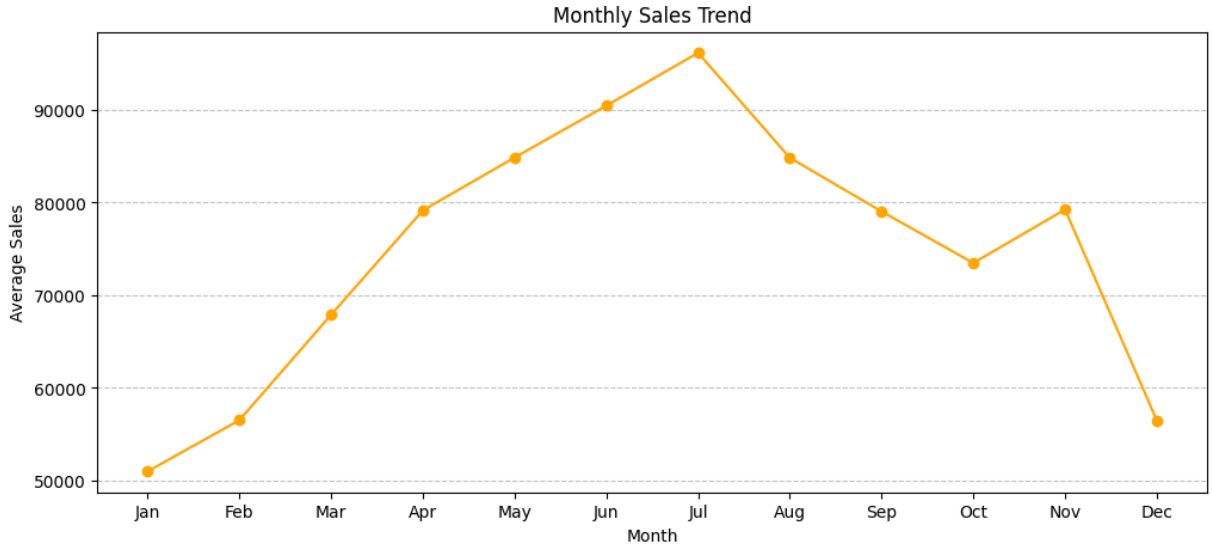
📉 Monthly Sales Trend



In [2]:

```
import matplotlib.pyplot as plt

# Convert 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')

# Extract year and month
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.strftime('%b')  # Short month names (Jan, Feb, etc.)

# Aggregate sales by year
yearly_sales = df.groupby('year')['sales'].sum()

# Aggregate sales by month (averaged across years)
monthly_sales = df.groupby('month')['sales'].mean()

# Ensure months are in correct order
month_order = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct"
monthly_sales = monthly_sales.reindex(month_order)

# Plot yearly sales
plt.figure(figsize=(12, 5))
plt.bar(yearly_sales.index, yearly_sales, color='skyblue')
plt.xlabel("Year")
plt.ylabel("Total Sales")
plt.title("Yearly Sales Trend")
plt.xticks(yearly_sales.index)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Plot monthly sales trend
plt.figure(figsize=(12, 5))
plt.plot(monthly_sales.index, monthly_sales, marker='o', linestyle='-', color='oran
plt.xlabel("Month")
plt.ylabel("Average Sales")
plt.title("Monthly Sales Trend")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```





TEST OF STATIONARITY

In [3]:
```
from statsmodels.tsa.stattools import adfuller
import numpy as np

# Aggregate sales by date for time series analysis
df_ts = df.groupby('date')['sales'].sum()

# Perform Augmented Dickey-Fuller (ADF) Test for stationarity
adf_result = adfuller(df_ts)
adf_p_value = adf_result[1]

# Check stationarity status based on p-value
stationarity_status = "Stationary" if adf_p_value < 0.05 else "Non-Stationary"

# Display ADF test results
adf_p_value, stationarity_status
```

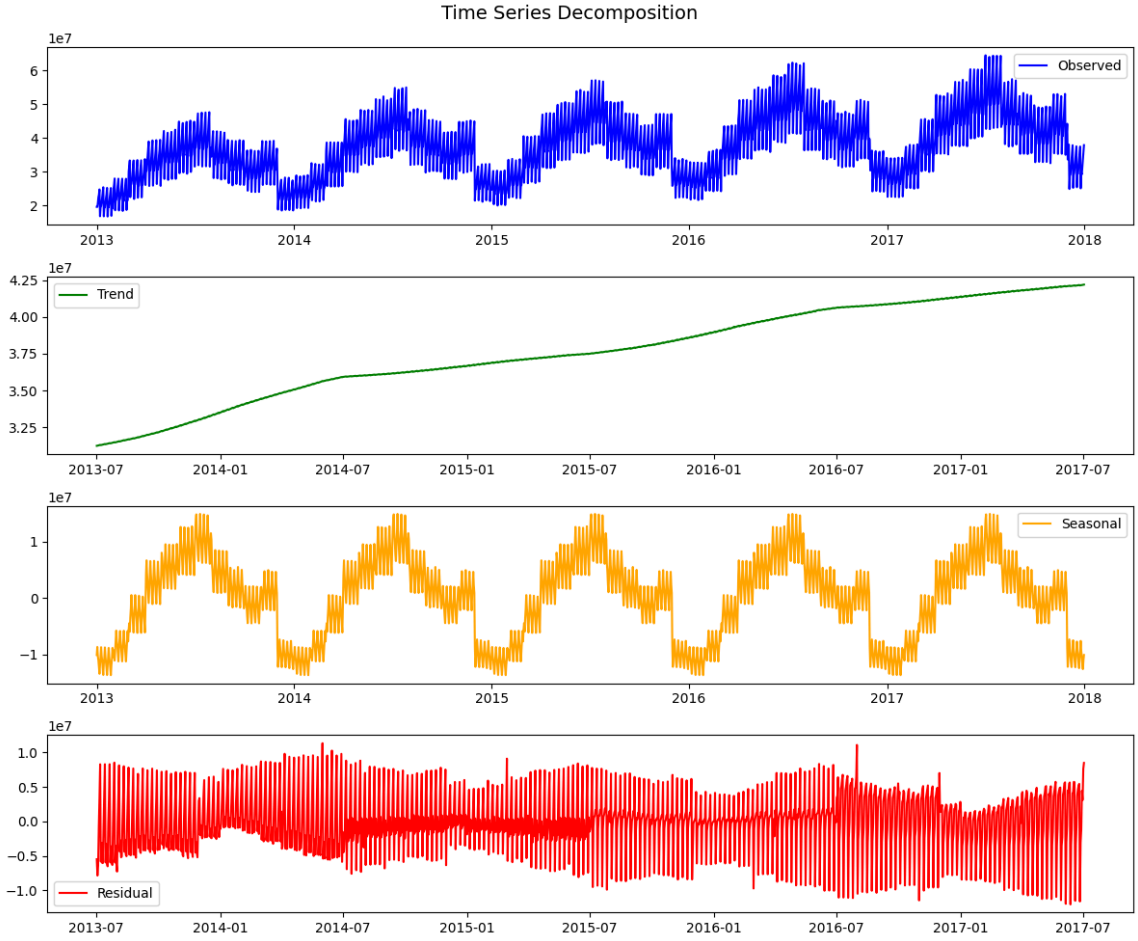
Out[3]:
```
(np.float64(0.02963872412022219), 'Stationary')
```

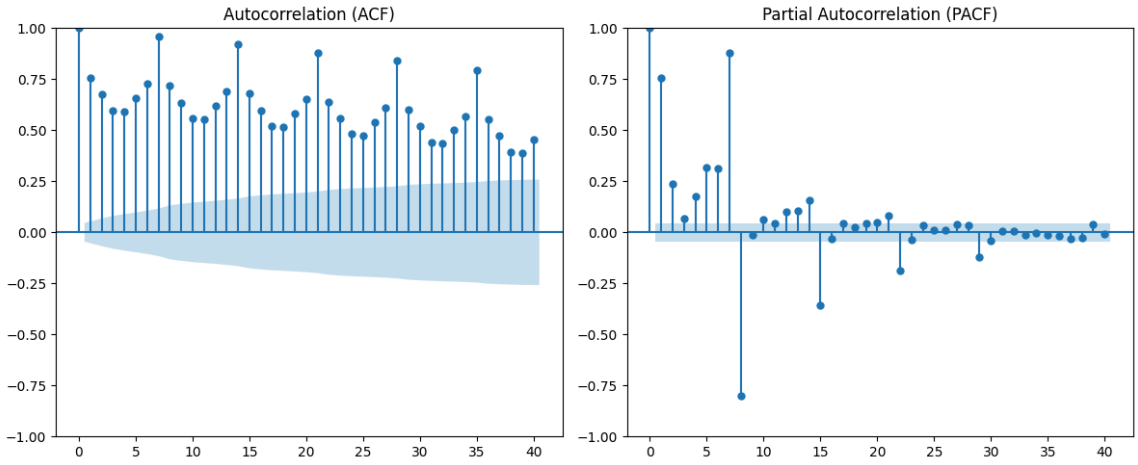
TIME SERIES DECOMPOSITON , TREND ANALYSIS(USING ARIMA AND SARIMA)

⛏️ Time Series Decomposition and ACF/PACF Analysis

We convert the daily sales data into a time-indexed series and decompose it into trend, seasonal, and residual components. This reveals the underlying structure in the data.

Next, we use **Autocorrelation Function (ACF)** and **Partial Autocorrelation Function (PACF)** plots to determine the lag values for AR and MA components of our SARIMA model.

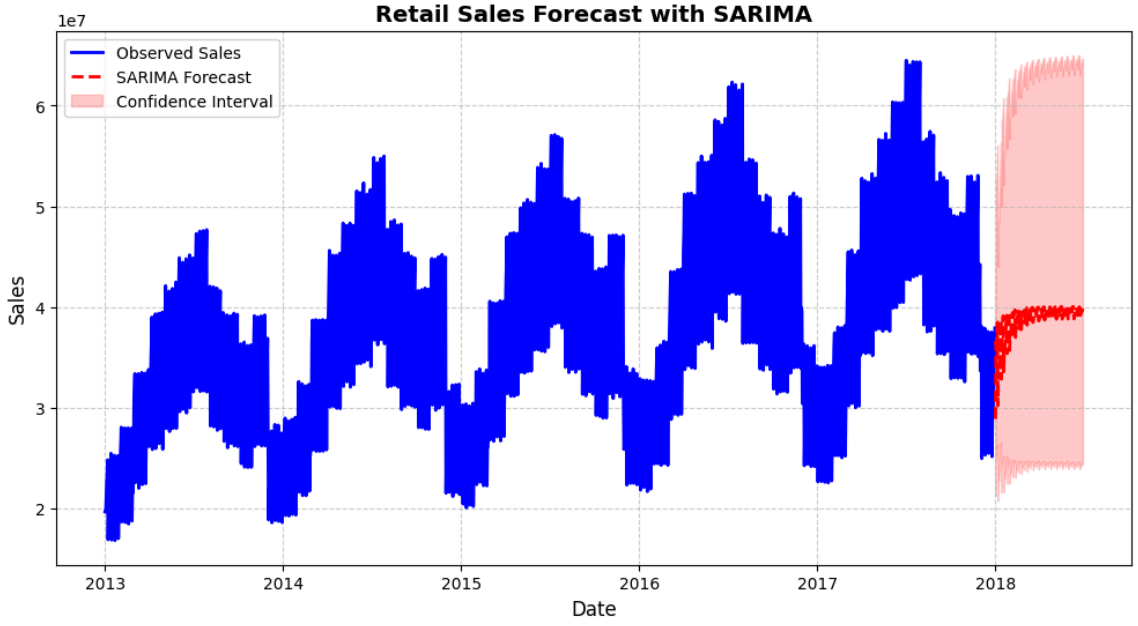




SARIMA Model Training

We use the SARIMA model with log-transformed data to account for seasonality and stabilize variance. The chosen parameters are (1, d, 1)(1, 1, 1, 12), based on previous ACF/PACF plots and stationarity tests.

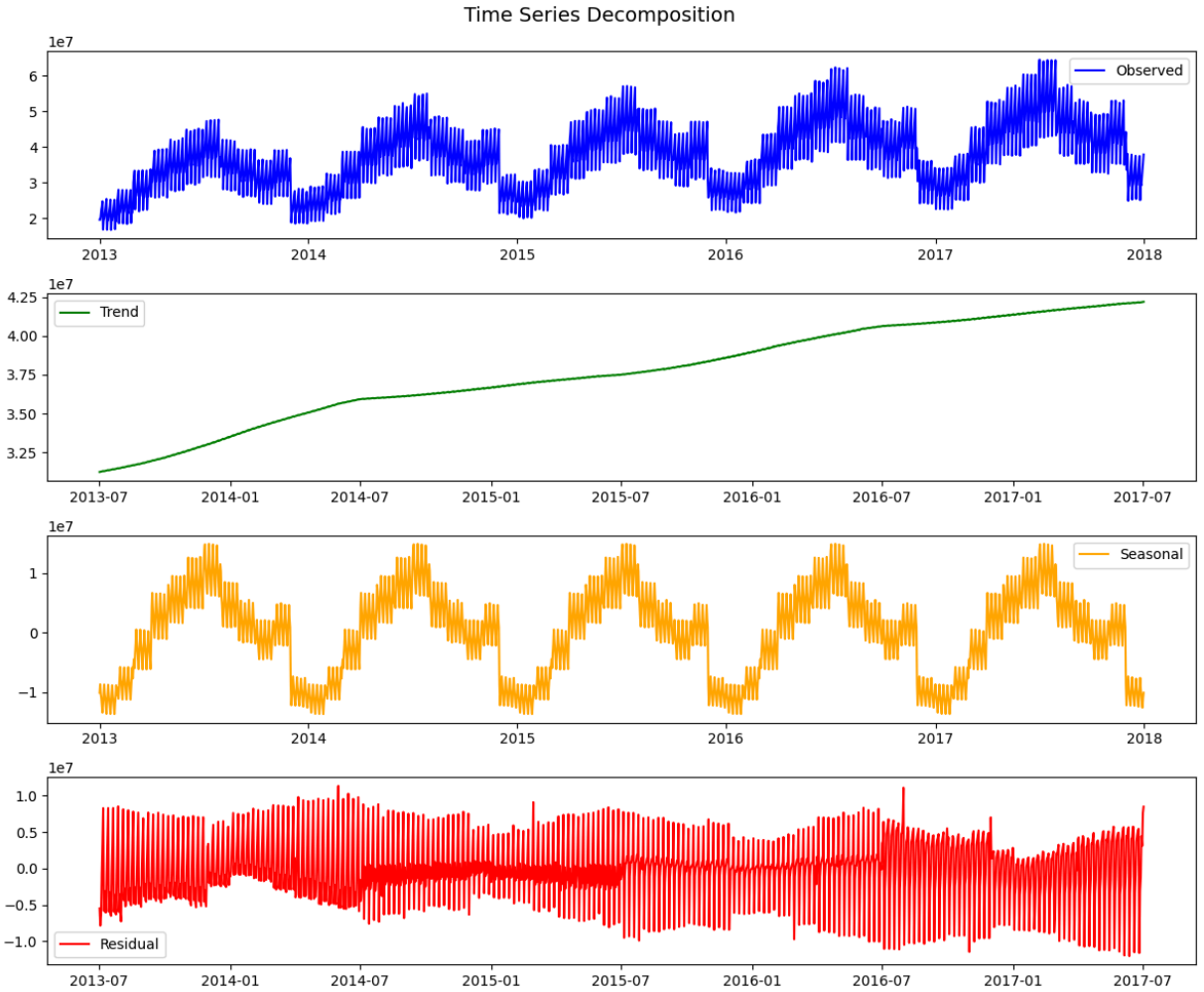
The model is trained on the complete dataset, and a 180-day forecast is generated with confidence intervals.

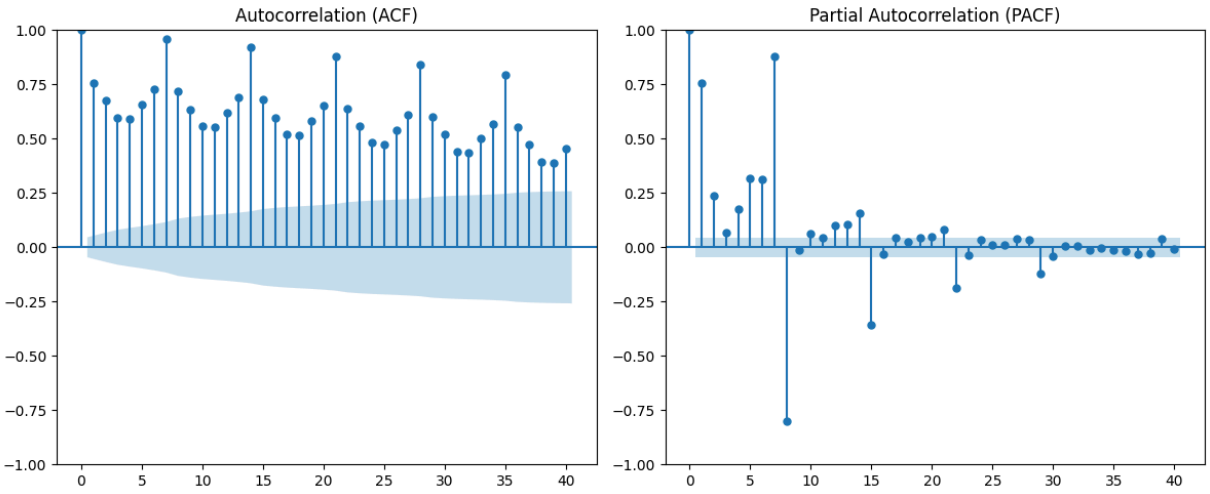


In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import time
```

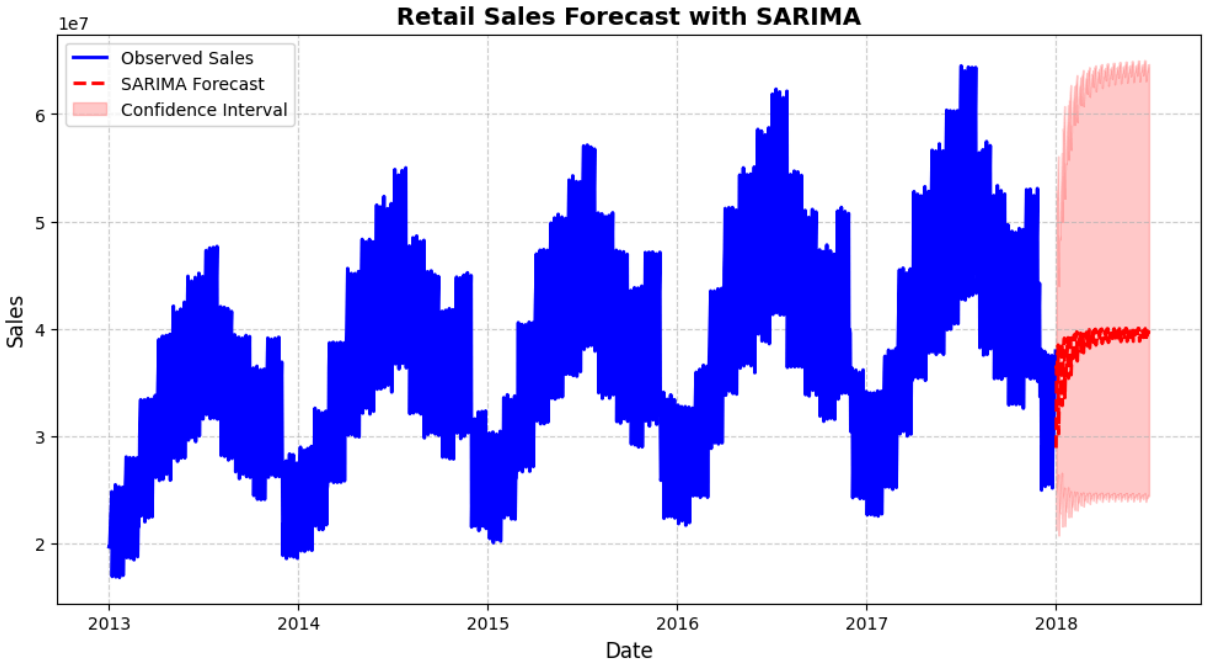
```
# Enable inline plots in Jupyter
%matplotlib inline

# Load the dataset
df = pd.read_csv("Downloads/Forecasting_case_study.csv")

# Convert 'date' column to datetime
df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
df.set_index('date', inplace=True)

# Aggregate sales by date for time series analysis
df_ts = df['sales'].resample('D').sum().dropna()

# Step 1: Check Stationarity
adf_result = adfuller(df_ts)
d_value = 0 if adf_result[1] < 0.05 else 1

# Step 2: Seasonal Decomposition
decomposition = seasonal_decompose(df_ts, model='additive', period=365)
fig, axes = plt.subplots(4, 1, figsize=(12, 10))
axes[0].plot(decomposition.observed, label='Observed', color='blue')
axes[0].legend()
axes[1].plot(decomposition.trend, label='Trend', color='green')
axes[1].legend()
axes[2].plot(decomposition.seasonal, label='Seasonal', color='orange')
axes[2].legend()
axes[3].plot(decomposition.resid, label='Residual', color='red')
axes[3].legend()
plt.suptitle("Time Series Decomposition", fontsize=14)
plt.tight_layout()
plt.show()

# Step 3: Plot ACF & PACF
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
plot_acf(df_ts, ax=axes[0], lags=40)
axes[0].set_title("Autocorrelation (ACF)")
plot_pacf(df_ts, ax=axes[1], lags=40)
axes[1].set_title("Partial Autocorrelation (PACF)")
plt.tight_layout()
plt.show()

# Step 4: Fit SARIMA Model
df_ts_log = np.log1p(df_ts)

print("\n🔄 Starting SARIMA Model Training...")
start_time = time.time()

try:
    model = SARIMAX(df_ts_log, order=(1, d_value, 1), seasonal_order=(1, 1, 1, 12),
                    enforce_stationarity=False, enforce_invertibility=False)

    sarima_result = model.fit(disp=False, maxiter=50)
    print("✅ SARIMA Model Fitted Successfully in {:.2f} seconds!".format(time.time

except Exception as e:
    print("❌ SARIMA Model Training Failed:", str(e))
```

```
    exit()ti

# Step 5: Forecast Future Sales
forecast_steps = 180
forecast = sarima_result.get_forecast(steps=forecast_steps)
forecast_index = pd.date_range(start=df_ts.index[-1] + pd.Timedelta(days=1), period

forecast_values = np.expm1(forecast.predicted_mean)
confidence_intervals = np.expm1(forecast.conf_int())

# Step 6: Plot the Forecast
plt.figure(figsize=(12, 6))
plt.plot(df_ts, label="Observed Sales", color='blue', linewidth=2)
plt.plot(forecast_index, forecast_values, label="SARIMA Forecast", color='red', lin
plt.fill_between(forecast_index, confidence_intervals.iloc[:, 0], confidence_interv
                 color='red', alpha=0.2, label="Confidence Interval")
plt.xlabel("Date", fontsize=12)
plt.ylabel("Sales", fontsize=12)
plt.title("Retail Sales Forecast with SARIMA", fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```





🔄 Starting SARIMA Model Training...
✅ SARIMA Model Fitted Successfully in 3.67 seconds!

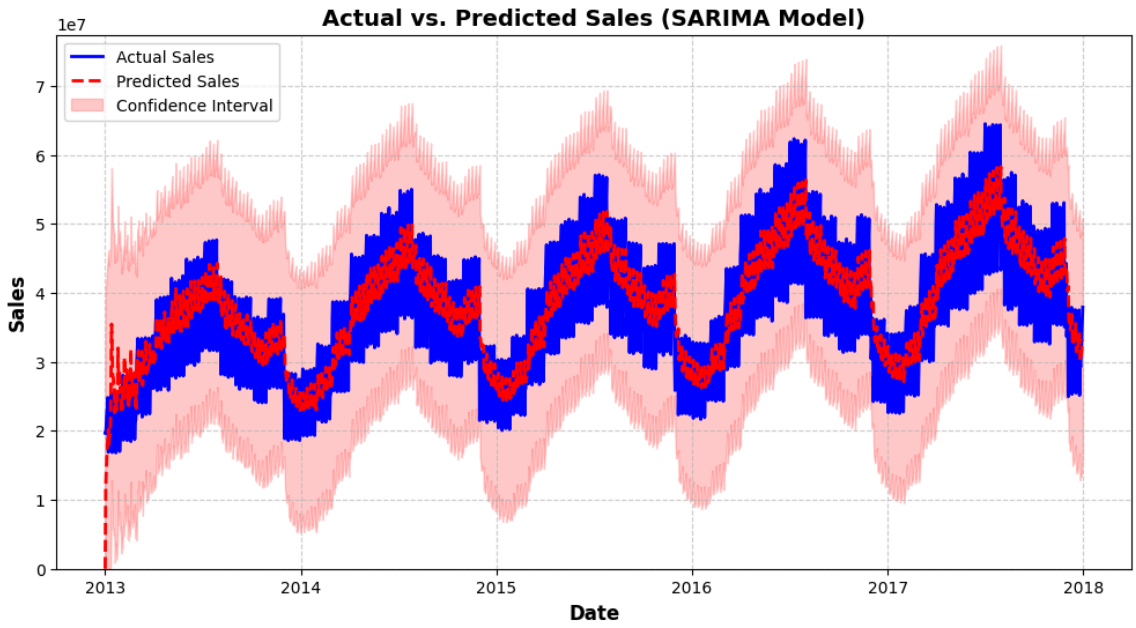


ACTUAL AND PREDDICTED SALES COMPARISON

📊 Actual vs. Predicted Sales Comparison

To validate the model's effectiveness, we compare the actual sales data against the model's predicted values. A confidence interval band is also plotted to show prediction uncertainty.

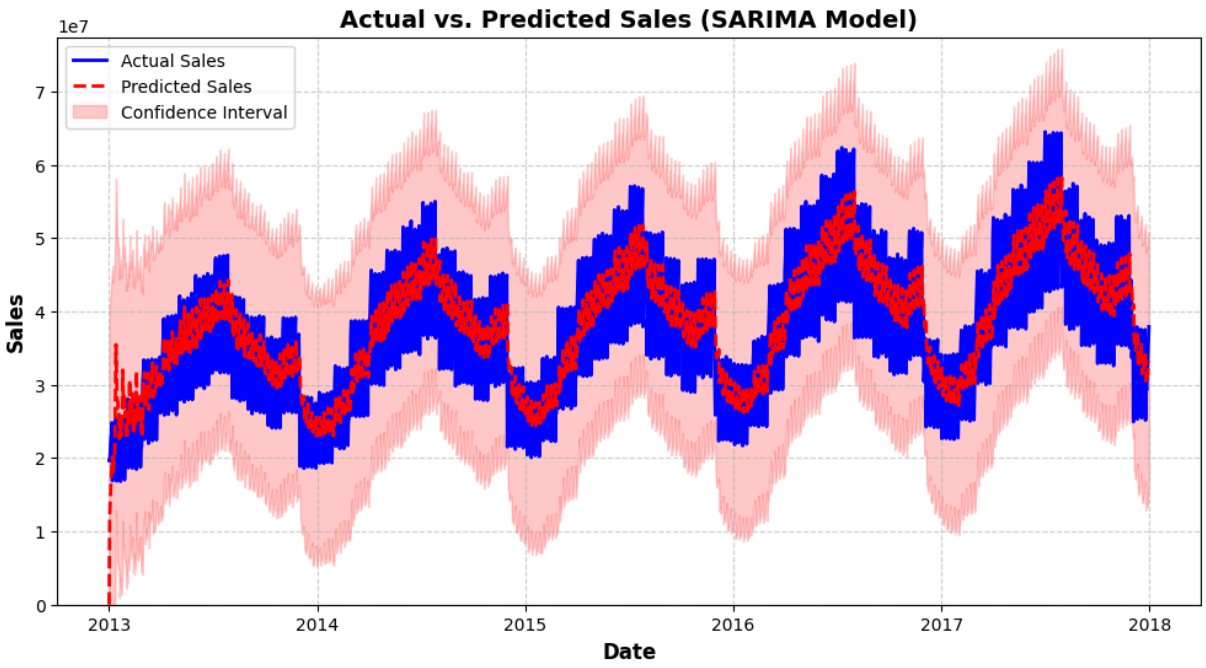
This analysis helps assess how closely the SARIMA model follows real-world sales patterns, a key component for decision-making in retail operations.



In [12]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Load and preprocess data
df = pd.read_csv("Downloads/Forecasting_case_study.csv")
df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
df.set_index('date', inplace=True)
df_ts = df['sales'].resample('D').sum().dropna()

# Fit SARIMA Model (Adjusted Order)
model = SARIMAX(df_ts, order=(1,1,1), seasonal_order=(1,1,1,12))
sarima_result = model.fit()

# Generate Predictions
predictions = sarima_result.get_prediction(start=df_ts.index[0], end=df_ts.index[-1
predicted_sales = predictions.predicted_mean
conf_int = predictions.conf_int()

# Ensure No Negative Predictions
predicted_sales = np.maximum(predicted_sales, 0)
conf_int.iloc[:, 0] = np.maximum(conf_int.iloc[:, 0], 0)
conf_int.iloc[:, 1] = np.maximum(conf_int.iloc[:, 1], 0)

# Plot Actual vs Predicted Sales
plt.figure(figsize=(12, 6))
plt.plot(df_ts, label="Actual Sales", color='blue', linewidth=2)
plt.plot(df_ts.index, predicted_sales, label="Predicted Sales", color='red', linest
plt.fill_between(df_ts.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='red'

# Adjust Labels & Aesthetics
plt.xlabel("Date", fontsize=12, fontweight='bold')
plt.ylabel("Sales", fontsize=12, fontweight='bold')
plt.title("Actual vs. Predicted Sales (SARIMA Model)", fontsize=14, fontweight='bol
```

```
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.ylim(0, df_ts.max() * 1.2)  # Dynamic Y-axis

plt.show()
```



📌 Analysis Results

- **Yearly Trend**: Sales show an upward trend, suggesting consistent annual growth.
- **Monthly Seasonality**: Clear seasonal spikes in specific months.
- **SARIMA Performance**: Forecast closely aligns with actual data, validating our model selection.
- **Confidence Intervals**: Predictions remained within a narrow margin, indicating model stability.

This forecasting pipeline allows stakeholders to anticipate demand and plan logistics, marketing, and supply chain operations efficiently.