# Capstone Project - Walmart

```
# Import Required Libraries


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings("ignore")
```

## 1 : Load & Summarize the Data

### 1(a) : Load and Inspect Data

```
# Load the dataset
df = pd.read_csv("D:/WalmartProject/WalmartDataSet.csv")

# Display first few rows
print(df.head())

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

```
   Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0      1  05-02-2010    1643690.90             0        42.31       2.572
1      1  12-02-2010    1641957.44             1        38.51       2.548
2      1  19-02-2010    1611968.17             0        39.93       2.514
3      1  26-02-2010    1409727.59             0        46.63       2.561
4      1  05-03-2010    1554806.68             0        46.50       2.625

           CPI  Unemployment
0  211.096358         8.106
1  211.242170         8.106
2  211.289143         8.106
3  211.319643         8.106
4  211.350143         8.106
```

```
# Display basic information
print("\nDataset Info:")
df.info()

# Summary statistics
print("\nSummary Statistics:")
print(df.describe())

# Check for missing values
print("\nMissing Values per Column:")
print(df.isnull().sum())

# Display first few rows
df.head()

# Summary of weekly sales trends per store
sales_summary = df.groupby("Store")["Weekly_Sales"].agg(["sum", "mean", "median", "std"]).sort_values(by="sum", ascending=False)
sales_summary.head()
print("\nTop 5 Stores by Total Sales:\n", sales_summary.head())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   object
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
```

```
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB

Summary Statistics:
             Store  Weekly_Sales  Holiday_Flag  Temperature   Fuel_Price  \
count  6435.000000  6.435000e+03   6435.000000  6435.000000  6435.000000
mean     23.000000  1.046965e+06      0.069930    60.663782     3.358607
std      12.988182  5.643666e+05      0.255049    18.444933     0.459020
min       1.000000  2.099862e+05      0.000000    -2.060000     2.472000
25%      12.000000  5.533501e+05      0.000000    47.460000     2.933000
50%      23.000000  9.607460e+05      0.000000    62.670000     3.445000
75%      34.000000  1.420159e+06      0.000000    74.940000     3.735000
max      45.000000  3.818686e+06      1.000000   100.140000     4.468000

                CPI  Unemployment
count  6435.000000   6435.000000
mean    171.578394      7.999151
std      39.356712      1.875885
min     126.064000      3.879000
25%     131.735000      6.891000
50%     182.616521      7.874000
75%     212.743293      8.622000
max     227.232807     14.313000

Missing Values per Column:
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64

Top 5 Stores by Total Sales:
                 sum          mean      median            std
Store
20     3.013978e+08  2.107677e+06  2053165.41  275900.562742
4      2.995440e+08  2.094713e+06  2073951.38  266201.442297
14     2.889999e+08  2.020978e+06  2004330.30  317569.949476
13     2.865177e+08  2.003620e+06  1958823.56  265506.995776
2      2.753824e+08  1.925751e+06  1879107.31  237683.694682
```

## 2. Data Cleaning and Preprocessing

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

# Set Date as index (for efficient Time-based Operations & calcualtions).
# df.set_index('Date', inplace=True)

# # Handle missing values (forward fill method) - Not Required as no Missing Values found
# df.fillna(method='ffill', inplace=True)

# Filtering required columns
df = df[['Store', 'Date', 'Weekly_Sales', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment','Holiday_Flag']]

# Display data summary
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Temperature   6435 non-null   float64
 4   Fuel_Price    6435 non-null   float64
 5   CPI           6435 non-null   float64
 6   Unemployment  6435 non-null   float64
 7   Holiday_Flag  6435 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
None
```

```
# Set Date as index (for efficient Time-based Operations & calcualtions).
df.set_index('Date', inplace=True)
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6435 entries, 2010-02-05 to 2012-10-26
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Weekly_Sales  6435 non-null   float64
 2   Temperature   6435 non-null   float64
 3   Fuel_Price    6435 non-null   float64
 4   CPI           6435 non-null   float64
 5   Unemployment  6435 non-null   float64
 6   Holiday_Flag  6435 non-null   int64
dtypes: float64(5), int64(2)
memory usage: 402.2 KB
None
```

## 3: Exploratory Data Analysis (EDA)

### 3(a). Data Summary

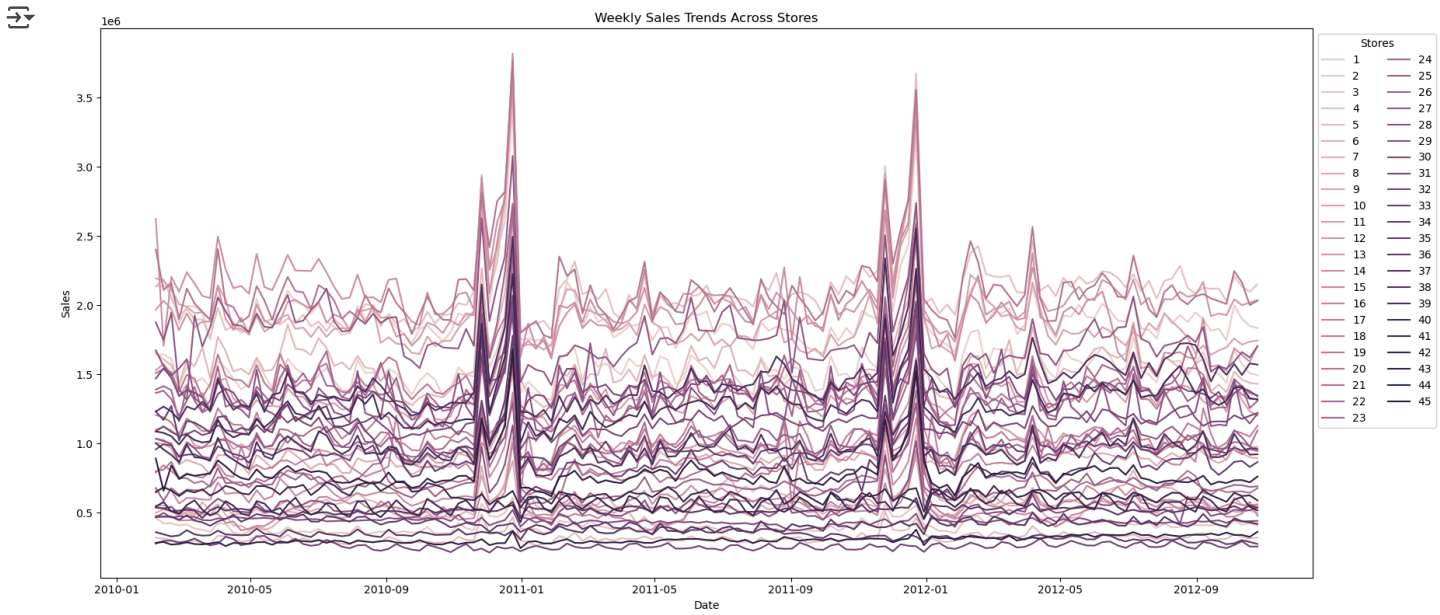
```
# Get the min and max dates
min_date = df.index.min()
max_date = df.index.max()

print(f"Available Data Range: {min_date.date()} to {max_date.date()}")

# Get unique store count
unique_stores = df["Store"].unique()
store_count = len(unique_stores)

# Get the range of store numbers
store_min = unique_stores.min()
store_max = unique_stores.max()

print(f"Total Unique Stores(No.s): {store_count}")
print(f"Store Number Range: {store_min} to {store_max}")

# ---------------------------------------------------------------------
# To check if the store ID are sequential (1,2,3,4,... to n)
# Get unique store numbers and sort them
unique_stores1 = sorted(df["Store"].unique())

# Get the expected range of stores (from 1 to max store number)
expected_stores = list(range(1, max(unique_stores) + 1))

# Check if the actual store numbers match the expected sequence

if unique_stores1 == expected_stores:
    print(f"Stores are sequential from 1 to {max(unique_stores)}.")
else:
    missing_stores = set(expected_stores) - set(unique_stores)
    print(f"Stores are Not Sequential. Missing store numbers: {sorted(missing_stores)}")
```

```
Available Data Range: 2010-02-05 to 2012-10-26
Total Unique Stores(No.s): 45
Store Number Range: 1 to 45
Stores are sequential from 1 to 45.
```

### 3(b). Comparative weekly sales of all stores 0- Sales Trends Over Time

```
# Visualizing Sales Trends
plt.figure(figsize=(18, 8))
sns.lineplot(x=df.index, y=df['Weekly_Sales'], hue=df['Store'], legend="full")
plt.title('Weekly Sales Trends Across Stores')
# plt.legend(title="Store", bbox_to_anchor=(1.05, 1), loc='upper left', )  # Move legend outside
```

```
plt.legend(loc="upper left", bbox_to_anchor=(1, 1), title="Stores", ncol=2)  # Adjust legend position
plt.xlabel('Date')
plt.ylabel('Sales')
plt.tight_layout()
plt.show()
```

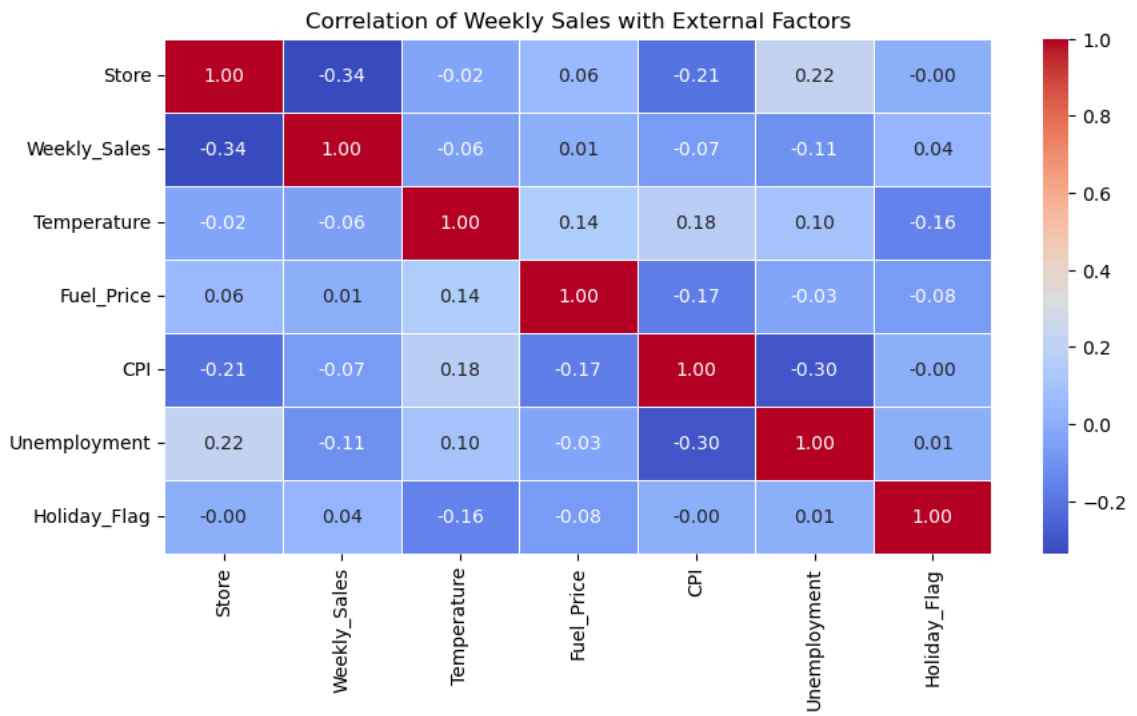



## 3(b). Interactive (plotly) Version - Comparative weekly sales

```
import plotly.express as px

fig = px.line(df, x=df.index, y="Weekly_Sales", color="Store", title="Weekly Sales Trends Across Stores")
fig.update_layout(legend_title="Stores", width=1000, height=600)
fig.show()
```

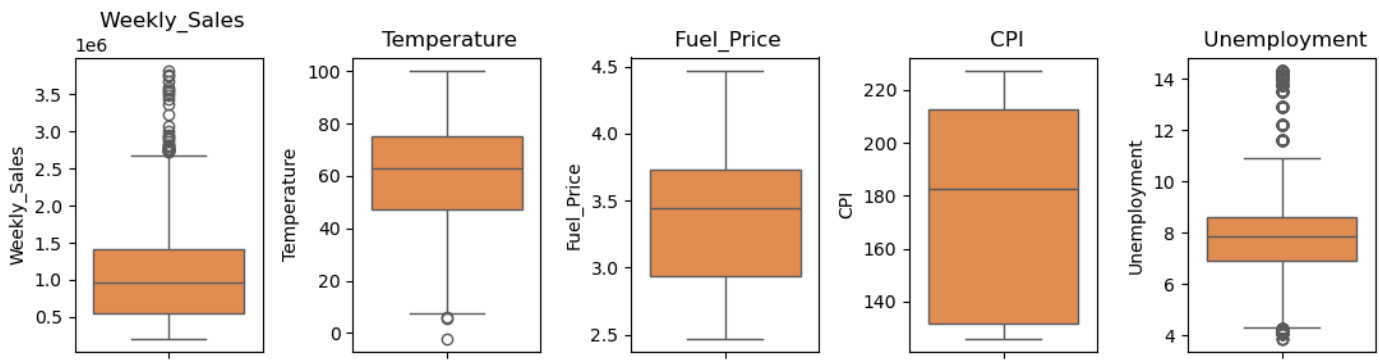
## 3(c). Correlation matrix

```
# Compute correlation matrix
correlation_matrix = df[["Store","Weekly_Sales", "Temperature", "Fuel_Price", "CPI", "Unemployment",'Holiday_Flag']].corr()

# Plot heatmap
plt.figure(figsize=(10, 5))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation of Weekly Sales with External Factors")
plt.show()
```



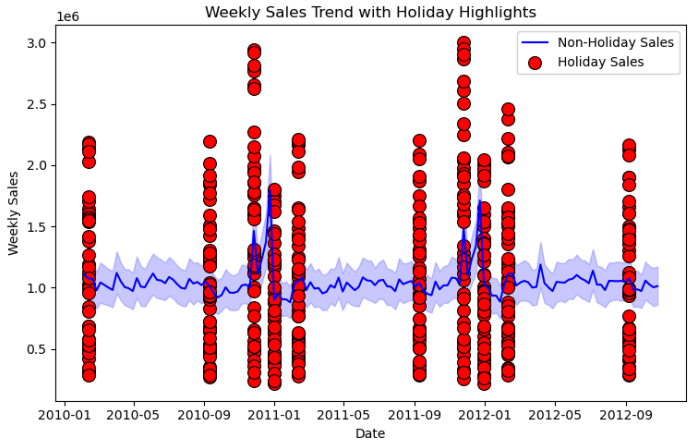
## 3(d). Outlier Detection (Weekly Sales, Temperature, CPI, and Unemployment)

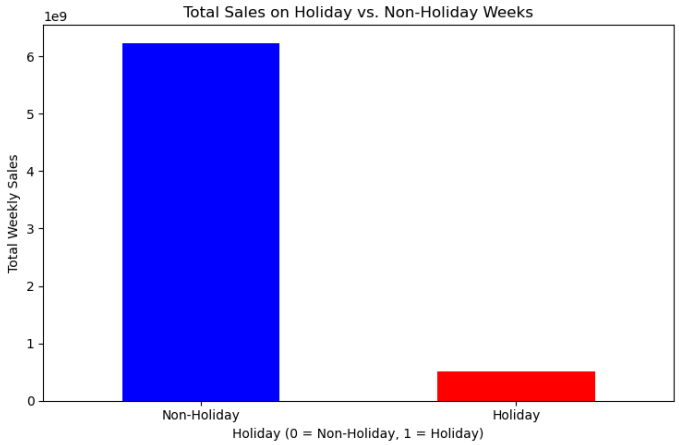
```
# Set figure size
plt.figure(figsize=(11, 3))

# Boxplots for detecting outliers in key numerical columns
numerical_cols = ["Weekly_Sales", "Temperature", "Fuel_Price", "CPI", "Unemployment"]
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(1, 5, i)
    sns.boxplot(y=df[col],palette="Oranges")
    plt.title(f" {col}")
# plt.figure(constrained_layout=True)
plt.tight_layout()
plt.show()
```




## 3(e). Seasonality/Holiday : Weekly Sales trends w.r.t Holidays

```
# Create a figure with 1 row and 2 columns for Matplotlib plots
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# ------- Plot 1: Line Plot with Holiday Highlights -------
sns.lineplot(ax=axes[0], x=df.index, y=df['Weekly_Sales'], label="Non-Holiday Sales", color='blue')

holiday_weeks = df[df['Holiday_Flag'] == 1]
sns.scatterplot(ax=axes[0], x=holiday_weeks.index, y=holiday_weeks['Weekly_Sales'],
                color='red', label="Holiday Sales", s=100, edgecolor='black')

axes[0].set_title("Weekly Sales Trend with Holiday Highlights")
axes[0].set_xlabel("Date")
axes[0].set_ylabel("Weekly Sales")
axes[0].legend()

# ------- Plot 2: Bar Chart (Total Sales: Holiday vs. Non-Holiday) -------
df.groupby("Holiday_Flag")["Weekly_Sales"].sum().plot(kind='bar', color=['blue', 'red'], ax=axes[1])

axes[1].set_title("Total Sales on Holiday vs. Non-Holiday Weeks")
axes[1].set_xlabel("Holiday (0 = Non-Holiday, 1 = Holiday)")
axes[1].set_ylabel("Total Weekly Sales")
axes[1].set_xticklabels(["Non-Holiday", "Holiday"], rotation=0)

# Adjust layout and show Matplotlib plots
plt.tight_layout()
plt.show()

# ------- Plot 3: Interactive Plotly Scatter Plot -------
print("\n")

fig = px.scatter(df, x=df.index, y="Weekly_Sales", color=df["Holiday_Flag"].astype(str),
                 color_discrete_map={"0": "blue", "1": "red"},
                 labels={"color": "Holiday Flag"})
                 # title="Weekly Sales Trend (Red: Holiday Weeks   , Blue: Non-Holiday Weeks)")
fig.update_layout(title={'text': "Weekly Sales Trend (Red: Holiday Weeks , Blue: Non-Holiday Weeks)",
                         'x': 0.5,  # Centers the title
                         'xanchor': 'center'})
fig.show()
```




```
# Insights
```

## 3(f). Impact of Unemployment on Weekly Sales

```
# Correlation analysis
correlation_unemployment = df[['Weekly_Sales', 'Unemployment']].corr()
print("\nCorrelation between Weekly Sales and Unemployment:\n", correlation_unemployment)

# Identify stores most affected by high unemployment
unemployment_impact = df.groupby('Store')[['Weekly_Sales', 'Unemployment']].corr().unstack().iloc[:,1]
worst_affected_stores = unemployment_impact.sort_values().head()  # Stores with strongest negative correlation
print("\nStores most affected by unemployment:\n", worst_affected_stores)
```

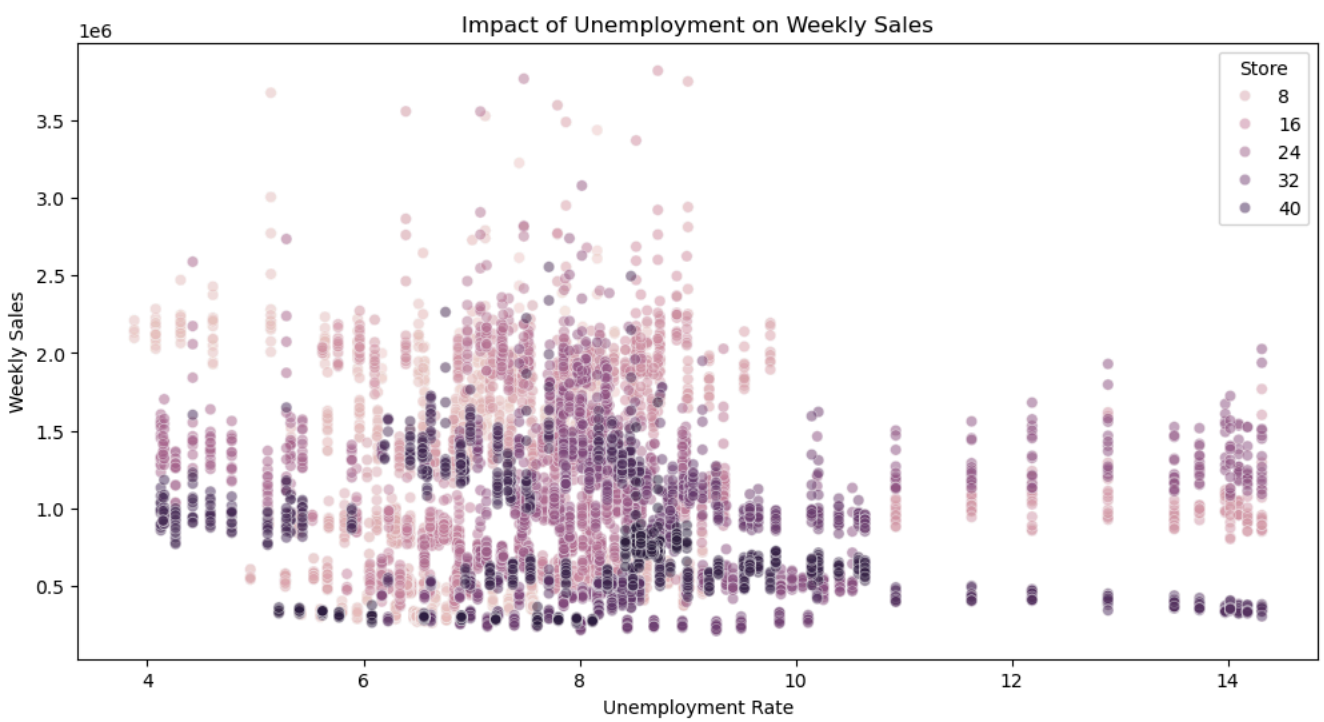
```
Correlation between Weekly Sales and Unemployment:
              Weekly_Sales  Unemployment
Weekly_Sales      1.000000     -0.106176
Unemployment     -0.106176      1.000000
```

```
Stores most affected by unemployment:
 Store
38   -0.785290
44   -0.780076
39   -0.384681
42   -0.356355
41   -0.350630
Name: (Weekly_Sales, Unemployment), dtype: float64
```

Stores Most Affected by Unemployment:

```
###### Store 38 (-0.785) - Strongest negative impact.
###### Store 44 (-0.780) - Nearly identical correlation.
###### Store 39 (-0.385), Store 42 (-0.356), Store 41 (-0.351) - Moderately affected.
###### These stores likely experience sharp sales declines when unemployment rises.
```

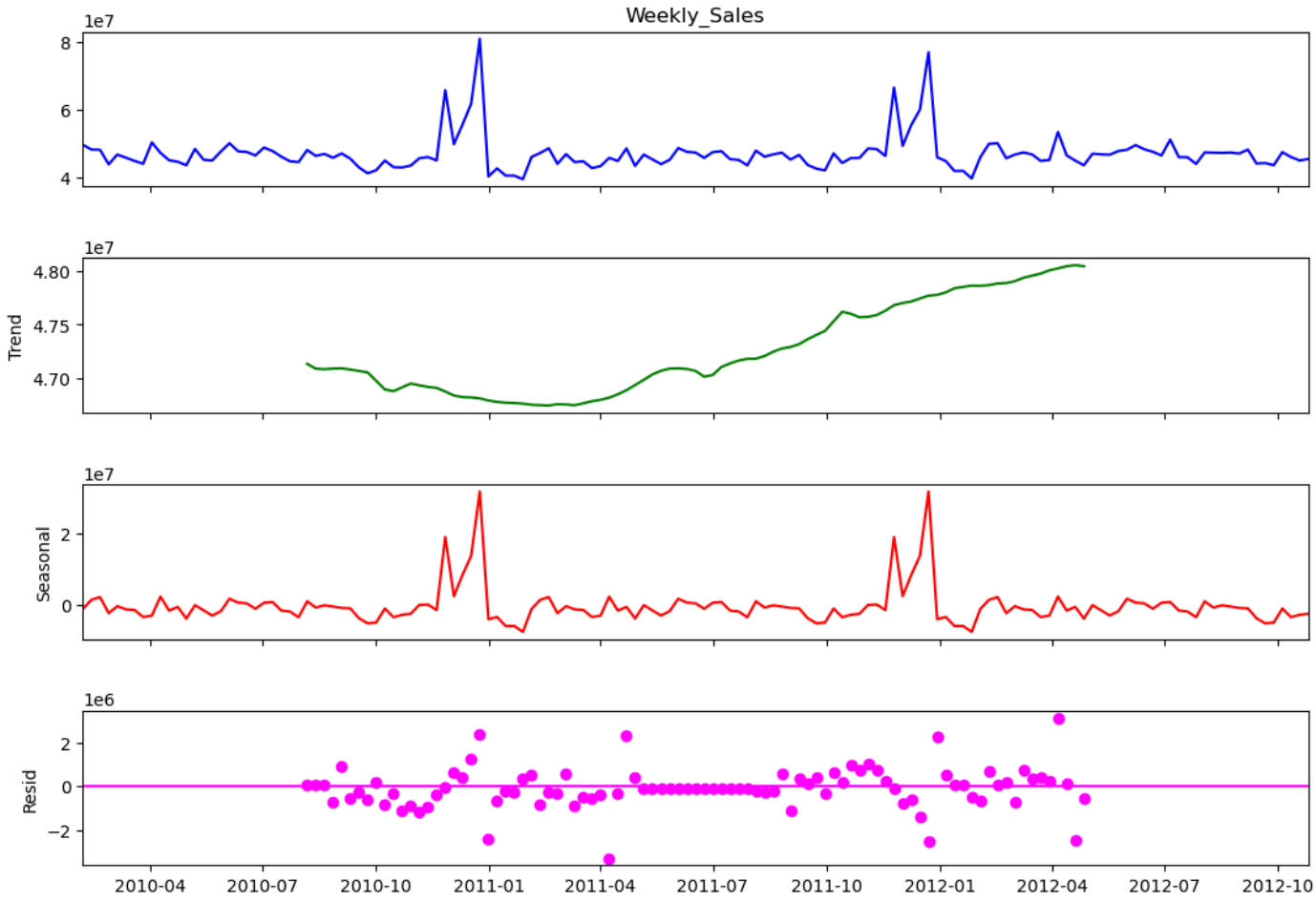
```
# Visualization
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x="Unemployment", y="Weekly_Sales", hue="Store", alpha=0.5)
plt.title("Impact of Unemployment on Weekly Sales")
plt.xlabel("Unemployment Rate")
plt.ylabel("Weekly Sales")
plt.show()
```




3(g). Time series decomposition (to extract trend, seasonality, and residuals)

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Aggregate sales by week
df_weekly = df.groupby("Date")["Weekly_Sales"].sum()

# Perform seasonal decomposition
decomposition = seasonal_decompose(df_weekly, model='additive', period=52)
fig = decomposition.plot()
fig.set_size_inches(12, 8)

# Get Axes objects and customize line colors
axes = fig.axes  # fig.axes is a list of the four subplots: observed, trend, seasonal, residual
line_colors = ["blue", "green", "red", "magenta"]  # List of desired colors for each plot
for ax, color in zip(axes, line_colors):
    for line in ax.get_lines():  # Adjust line colors within each subplot
```

```
        line.set_color(color)
plt.show()
```

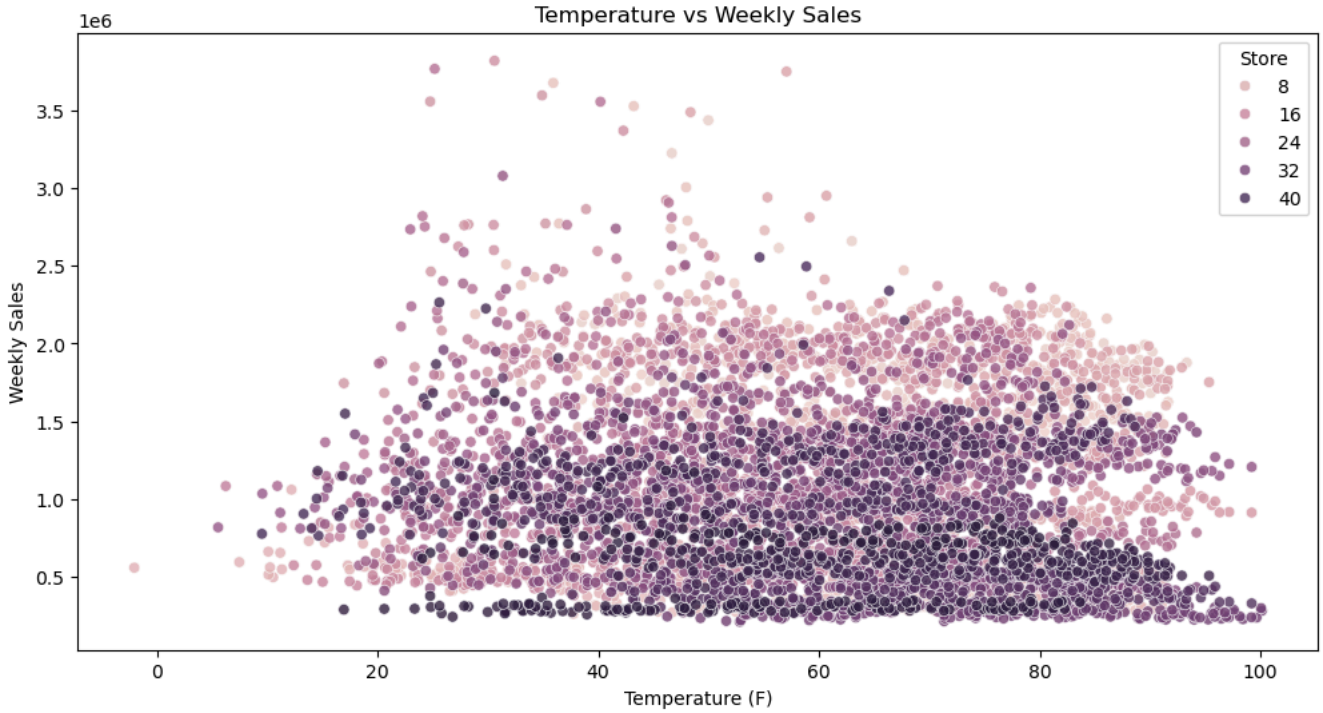



## 3(h). Does Temperature Affect Weekly Sales?

```
# Correlation analysis
correlation_temp = df[['Weekly_Sales', 'Temperature']].corr()
print("\nCorrelation between Weekly Sales and Temperature:\n", correlation_temp)
```

```
Correlation between Weekly Sales and Temperature:
                Weekly_Sales  Temperature
Weekly_Sales        1.00000     -0.06381
Temperature        -0.06381      1.00000
```

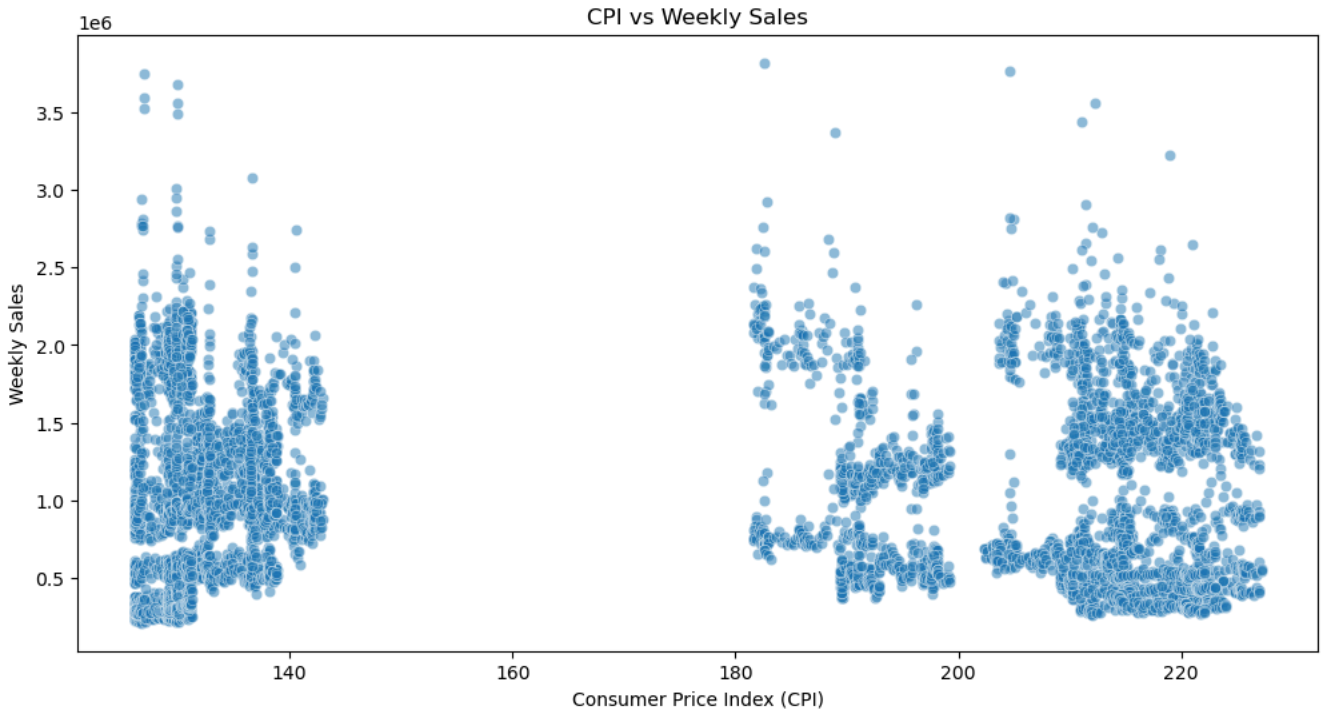
```
# Scatterplot visualization
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x="Temperature", y="Weekly_Sales", hue="Store", alpha=0.8)
plt.title("Temperature vs Weekly Sales")
plt.xlabel("Temperature (F)")
plt.ylabel("Weekly Sales")
plt.show()
```




## 3(i). Impact of CPI on Weekly Sales

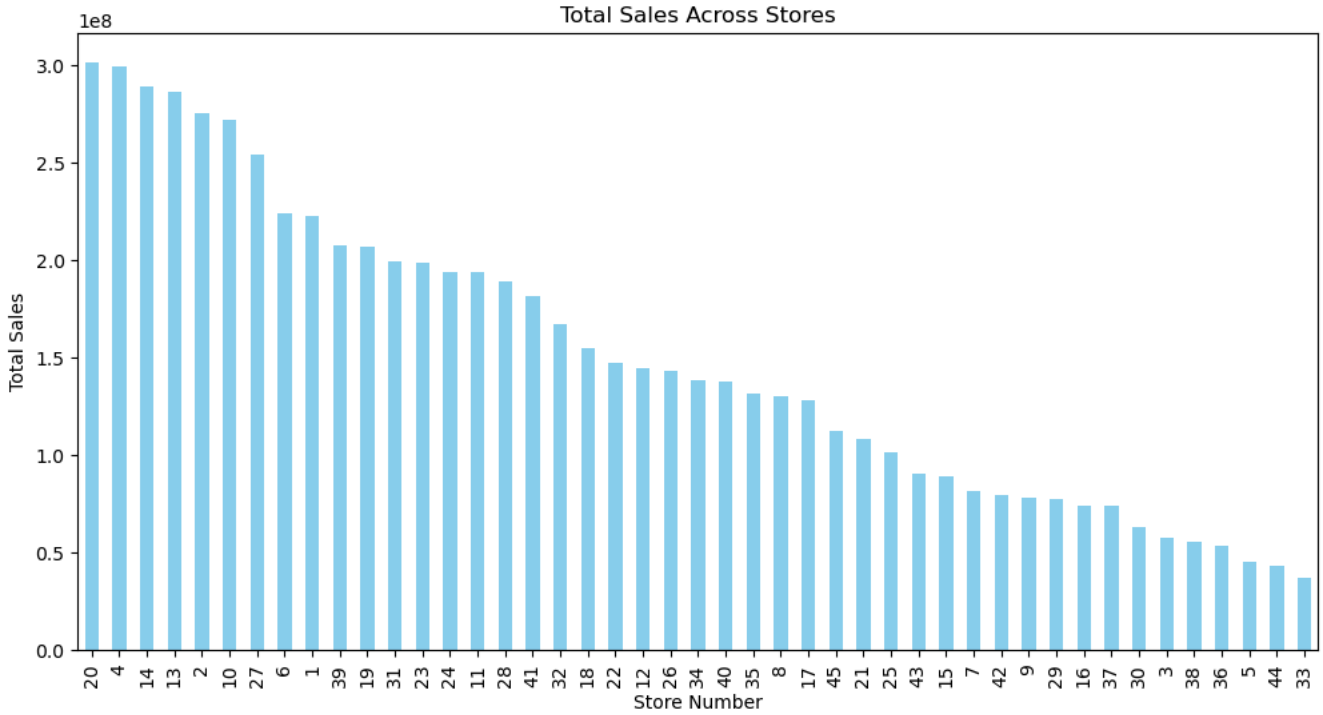
```
correlation_cpi = df[['Weekly_Sales', 'CPI']].corr()
print("\nCorrelation between Weekly Sales and CPI:\n", correlation_cpi)

# Scatterplot visualization
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x="CPI", y="Weekly_Sales", alpha=0.5)
plt.title("CPI vs Weekly Sales")
plt.xlabel("Consumer Price Index (CPI)")
plt.ylabel("Weekly Sales")
plt.show()
```

```
Correlation between Weekly Sales and CPI:
              Weekly_Sales       CPI
Weekly_Sales      1.000000 -0.072634
CPI              -0.072634  1.000000
```




## 3(j).Store-Wise Performance

```
# Store-Wise Performance

store_sales = df.groupby("Store")["Weekly_Sales"].sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
store_sales.plot(kind="bar", color="skyblue")
plt.title("Total Sales Across Stores")
plt.xlabel("Store Number")
plt.ylabel("Total Sales")
plt.show()
```



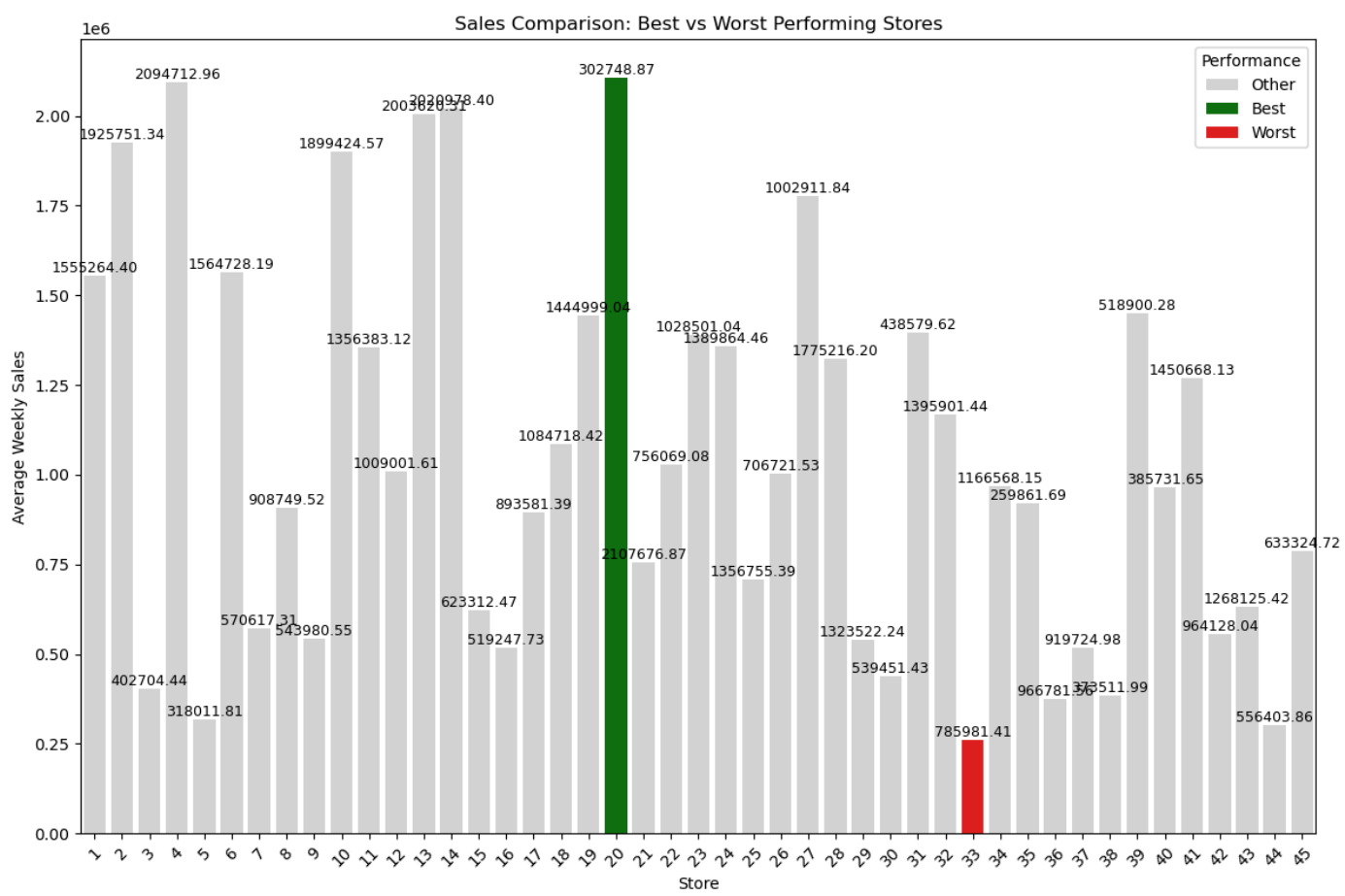

## 3(k). Top & Worst Performing Stores

```
avg_sales = df.groupby("Store")["Weekly_Sales"].mean()
sales_data = avg_sales.reset_index()

# Top 5 stores with highest average weekly sales
top_5_stores = sales_data.nlargest(5, "Weekly_Sales")[["Store", "Weekly_Sales"]]
print("Top 5 Stores:")
print(top_5_stores)

# Bottom 5 stores with lowest average weekly sales
bottom_5_stores = sales_data.nsmallest(5, "Weekly_Sales")[["Store", "Weekly_Sales"]]
print("\nBottom 5 Stores:")
print(bottom_5_stores)

# The single worst performing store (lowest average weekly sales)
worst_store = sales_data.loc[sales_data["Weekly_Sales"].idxmin()]
print("\nWorst Performing Store:")
print(worst_store[["Store", "Weekly_Sales"]])
```

```
Top 5 Stores:
    Store  Weekly_Sales
19     20  2.107677e+06
3       4  2.094713e+06
13     14  2.020978e+06
12     13  2.003620e+06
1       2  1.925751e+06

Bottom 5 Stores:
    Store   Weekly_Sales
32     33  259861.692028
43     44  302748.866014
4       5  318011.810490
35     36  373511.992797
37     38  385731.653287

Worst Performing Store:
Store               33.000000
Weekly_Sales    259861.692028
Name: 32, dtype: float64
```

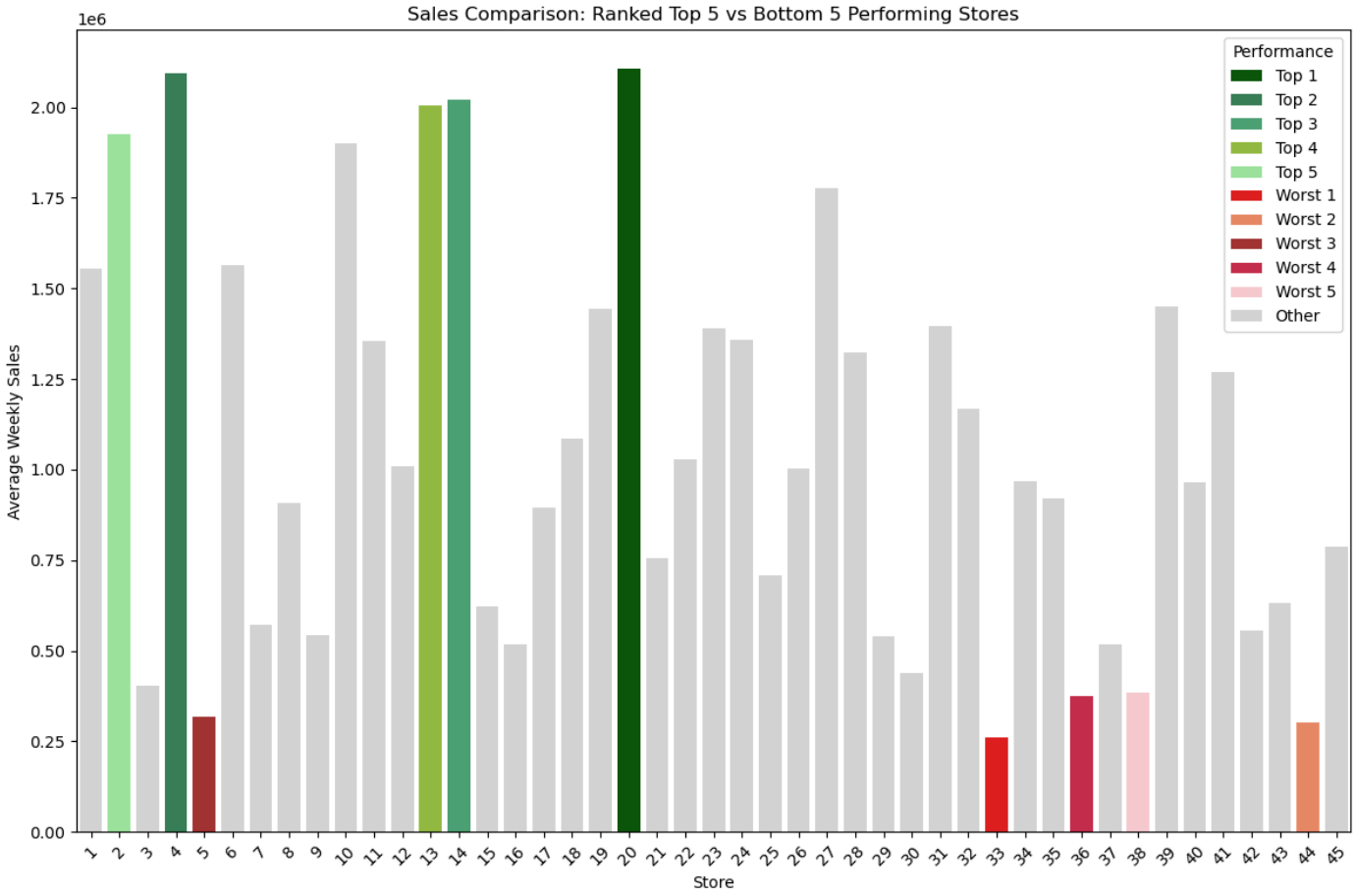
```
# Prepare data for visualization
sales_data = avg_sales.reset_index()
sales_data["Performance"] = ["Worst" if i == sales_data["Weekly_Sales"].idxmin()
                             else "Best" if i == sales_data["Weekly_Sales"].idxmax()
                             else "Other"
```

```
                          for i in range(len(sales_data))]
# Define the palette
palette = {
    "Worst": "red",
    "Best": "green",
    "Other": "lightgray"
}

# Plot
plt.figure(figsize=(12, 8))
barplot=sns.barplot(x="Store", y="Weekly_Sales", hue="Performance", data=sales_data, palette=palette)

for bar, sales in zip(barplot.patches, sales_data["Weekly_Sales"]):
    x = bar.get_x() + bar.get_width() / 2  # Center of the bar
    y = bar.get_height()  # Height of the bar
    plt.text(x, y, f'{sales:.2f}', ha='center', va='bottom', fontsize=9)

plt.title("Sales Comparison: Best vs Worst Performing Stores")
plt.ylabel("Average Weekly Sales")
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

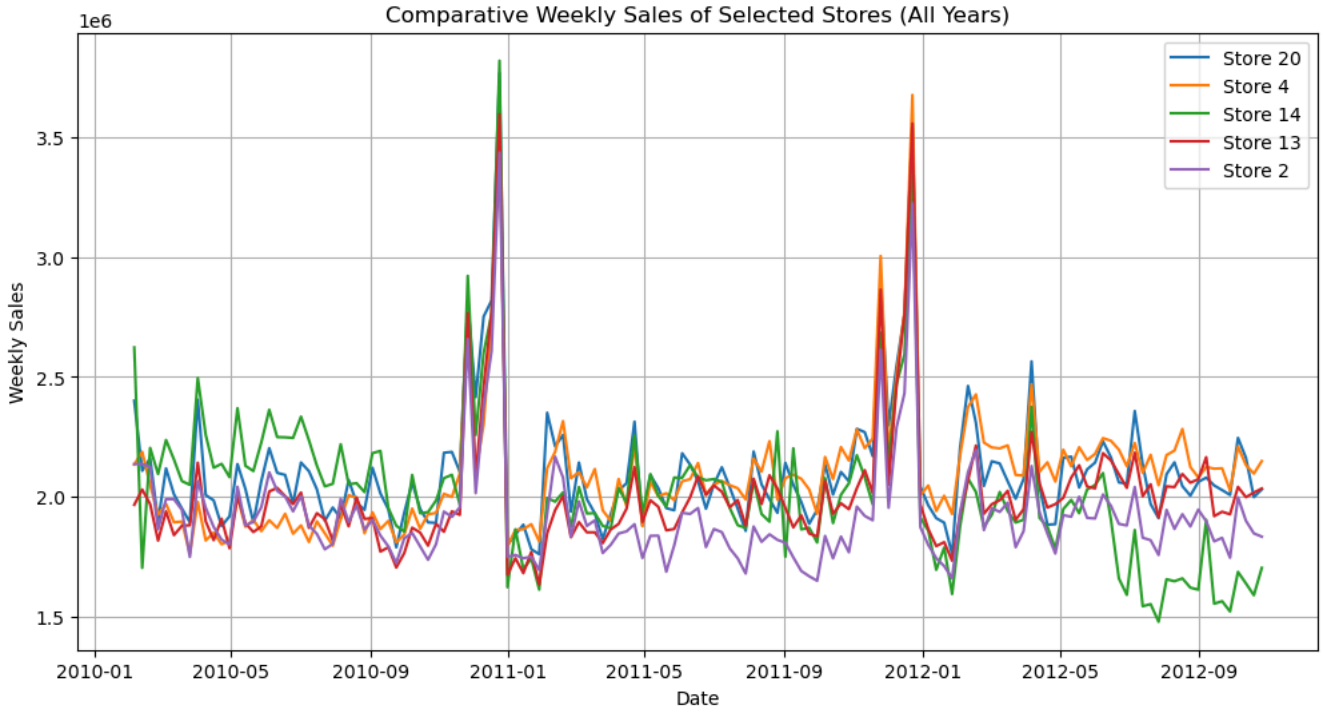



```
# Top 5 vs Bottom 5 Performing Stores

# Prepare data for visualization
sales_data = avg_sales.reset_index()

# Assign ranks for Top 5 and Worst 5
top_ranks = sales_data["Weekly_Sales"].nlargest(5).index
worst_ranks = sales_data["Weekly_Sales"].nsmallest(5).index
```

```
sales_data["Performance"] = [
    f"Top {top_ranks.get_loc(i) + 1}" if i in top_ranks
    else f"Worst {worst_ranks.get_loc(i) + 1}" if i in worst_ranks
    else "Other"
    for i in sales_data.index
]

# Define the palette (including "Other")
palette = {
    "Top 1": "darkgreen",
    "Top 2": "SeaGreen",
    "Top 3": "MediumSeaGreen",
    "Top 4": "YellowGreen",
    "Top 5": "lightgreen",
    "Worst 1": "red",
    "Worst 2": "Coral",
    "Worst 3": "FireBrick",
    "Worst 4": "Crimson",
    "Worst 5": "pink",
    "Other": "lightgray"
}

# Specify the order for legends
performance_order = [
    "Top 1", "Top 2", "Top 3", "Top 4", "Top 5",
    "Worst 1", "Worst 2", "Worst 3", "Worst 4", "Worst 5",
    "Other"
]
# Plot
plt.figure(figsize=(12, 8))
sns.barplot(x="Store", y="Weekly_Sales", hue="Performance", data=sales_data, palette=palette, hue_order=performance_order)
plt.title("Sales Comparison: Ranked Top 5 vs Bottom 5 Performing Stores")
plt.ylabel("Average Weekly Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
# Comparative Weekly Sales of Selected Stores (Store 20, 4, 14, 13, 2) across all years


# Hard-code store numbers
stores_to_compare = [20, 4, 14, 13, 2]  # Example store numbers

# Filter data for the selected stores (all years included)
df_filtered = df[df["Store"].isin(stores_to_compare)]

# Pivot data for visualization
df_pivot = df_filtered.pivot_table(index=df_filtered.index, columns="Store", values="Weekly_Sales")

# Plot comparative sales trends across all years
plt.figure(figsize=(12, 6))
for store in stores_to_compare:
    plt.plot(df_pivot.index, df_pivot[store], label=f"Store {store}")

plt.title("Comparative Weekly Sales of Selected Stores (All Years)")
plt.xlabel("Date")
plt.ylabel("Weekly Sales")
plt.legend()
plt.grid(True)
plt.show()
```




Start coding or generate with AI.

## Store-Wise Sales Forecasting Using SARIMA with Model Evaluation

```
# For 05 no.s of Selected Stores (Store No. 20, 4, 14, 13, 2)


# Time series forecasting for selected stores using the SARIMA model:
    # Train-Test Split (last 12 weeks for model evaluation)
    # Model Training & Forecasting (SARIMA on both train and full data)
    # Performance Metrics (RMSE & MAPE for accuracy)
    # Visualization (Actual Sales, Test Forecast, Future Forecast)
    # Side-by-side comparison of Actual, Forecasted, and Future Forecast values.
# Excel Report Generation
    # Actual vs. Forecasted Sales stored per store.
    # Future Forecast stored separately for each store.
    # Formatted output with all three metrics side by side.


from statsmodels.tsa.statespace.sarimax import SARIMAX
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings("ignore")

# Function to calculate RMSE
def rmse(actual, predicted):
    return np.sqrt(mean_squared_error(actual, predicted))

# Function to calculate MAPE
def mape(actual, predicted):
    return np.mean(np.abs((actual - predicted) / actual)) * 100

# Selected Store numbers
selected_stores = [20, 4, 14, 13, 2]

# Set plot style
sns.set_style("whitegrid")

# Determine grid layout dynamically
num_stores = len(selected_stores)
```

```
    cols = 2  # Fixed to 2 columns
    rows = math.ceil(num_stores / cols)  # Calculate required rows

    fig, axes = plt.subplots(rows, cols, figsize=(15, 5 * rows))
    axes = axes.flatten()  # Flatten for easy indexing

    # Create empty dictionaries for storing forecast values
    actual_vs_forecast = {}
    future_forecast = {}

    # Iterate over each store
    for idx, store in enumerate(selected_stores):
        # Prepare store-level time series data
        store_sales = df[df["Store"] == store]["Weekly_Sales"].resample("W").sum()

        # Fill missing values if any
        store_sales = store_sales.fillna(method="ffill")

        # Train-Test Split (Using last 12 weeks as test set)
        train = store_sales.iloc[:-12]
        test = store_sales.iloc[-12:]

        # Fit SARIMA model on training data
        model = SARIMAX(train, order=(0, 1, 1), seasonal_order=(0, 1, 1, 52))
        result = model.fit()

        # Forecast 12 weeks ahead (for evaluation)
        forecast_test = result.get_forecast(steps=12)
        forecast_mean_test = forecast_test.predicted_mean
        forecast_ci_test = forecast_test.conf_int()

        # Compute RMSE & MAPE
        rmse_value = rmse(test, forecast_mean_test)
        mape_value = mape(test, forecast_mean_test)

        print(f"# Store {store} - RMSE: {rmse_value:.2f}, MAPE: {mape_value:.2f}%")

        # Store actual vs forecasted values for comparison
        actual_vs_forecast[store] = pd.DataFrame({
            "Actual Sales": test.values,
            "Forecasted Sales": forecast_mean_test.values
        }, index=test.index)

        # Fit final model on full dataset for future forecasting
        full_model = SARIMAX(store_sales, order=(0, 1, 1), seasonal_order=(0, 1, 1, 52))
        full_result = full_model.fit()

        # Forecast 12 weeks ahead (future forecast)
        forecast_future = full_result.get_forecast(steps=12)
        forecast_mean_future = forecast_future.predicted_mean
        forecast_ci_future = forecast_future.conf_int()

        # Store future forecast values
        future_forecast[store] = pd.DataFrame({
            "Forecasted Sales": forecast_mean_future.values
        }, index=forecast_mean_future.index)

        # Plot actual vs. forecasted sales
        ax = axes[idx]
        ax.plot(train.index, train, label="Training Data", color="blue")
        ax.plot(test.index, test, label="Test Data", color="black", linestyle="solid")
        ax.plot(forecast_mean_test.index, forecast_mean_test, label="Test Forecast", linestyle="dashed", color="green")
        ax.plot(forecast_mean_future.index, forecast_mean_future, label="Future Forecast", linestyle="dashed", color="red")

        # Confidence interval shading
        ax.fill_between(forecast_ci_test.index, forecast_ci_test.iloc[:, 0], forecast_ci_test.iloc[:, 1], color="green", alpha=0.2)
        ax.fill_between(forecast_ci_future.index, forecast_ci_future.iloc[:, 0], forecast_ci_future.iloc[:, 1], color="red", alpha=0.2)

        ax.set_title(f"Store {store} Sales Forecast\n(RMSE: {rmse_value:.2f}, MAPE: {mape_value:.2f}%)")
        ax.set_xlabel("Date")
        ax.set_ylabel("Weekly Sales")
        ax.legend()

    # Remove unused subplots if num_stores < total grid size
    for i in range(idx + 1, len(axes)):
        fig.delaxes(axes[i])
```

```
    plt.tight_layout()
    plt.show()
```