Capstone Project

A retail store that has multiple outlets across the country are facing issues in managing the inventory to match the demand with respect to supply.

Dataset - Walmart.csv

Table of Contents

Problem Statement

The company is suffering from a financial crisis, and they aim to identify the key factors influencing sales across their 45 stores. By analysing weekly sales patterns, holiday effects, and the impact of external variables such as fuel prices, temperature, CPI, and unemployment, they hope to develop data-driven strategies to optimize operations, improve profitability, and overcome financial challenges.

Project Objective

For any company, it is important that they expand their business, and they are seeking new ways to expand their horizons. The cleaning process may vary depending on the domain of the chosen company.

So, in this project I am going to analyse the data that Walmart has collected over certain time intervals from 45 of its stores. By examining trends in sales, the impact of holidays, and external variables like fuel prices, temperature, Consumer Price Index (CPI) and unemployment rates, the goal is to derive valuable insights that can help the company combat its financial crisis. Their sales do increase of fuel rates effect their store sales or does the temperature contribute anything to their sales. This involves identifying patterns, improving operational efficiency, and developing strategic solutions to drive profitability and long-term growth.

So, based on our conclusions the Walmart can provide alternative methods to increase their sales in upcoming times.

Data Description

This dataset provides sales and related metrics for 45 Walmart stores over weekly time intervals, over a nearly three-year period spanning from February 2010 to October 2012. This dataset focuses on weekly sales data and several external factors that may influence those sales.

Features of the Dataset:

- Store: Each of the 45 stores is assigned a unique numeric identifier, making it easy to analyse individual store performance or compare them against one another.
- Date: Indicates the date in the format DD-MM-YYYY.
- Weekly_Sales: The dataset concentrates around the column "Weekly_Sales," which depicts the total sales revenue for a particular store during a specific week. This is the primary metric to evaluate performance in our project.
- Holiday_Flag: The "Holiday_Flag" column indicates whether a week contains a holiday or not where 1 represents a holiday and 0 represents a working day. Holidays often bring a surge in sales due to promotional events, festive shopping, or seasonal demand. It allows for distinguishing between normal and holiday-influenced weeks.
- Temperature: The average temperature during the week, which may affect shopping habits. For example, colder weeks might see increased sales of warm clothing or heaters, while hotter weeks might favour air conditioners or summer apparel.
- Fuel_Price: Represents the cost of fuel, which could impact consumer mobility and transportation costs for goods.

- CPI: The Consumer Price Index reflects the average price changes in consumer goods and services. It's a critical metric for understanding inflation and purchasing power during the period.
- Unemployment Rate: Provides insights into the economic environment. Higher unemployment may correlate with reduced consumer spending, while lower unemployment could signify robust spending potential.

The information from this dataset makes it ideal for decision-making, as it relates sales data with contextual elements, revealing patterns and opportunities for actionable insights.

Various insights from the data

a. If the weekly sales are affected by the unemployment rate, if yes- which stores are suffering the most?

```
df[['Weekly_Sales', 'Unemployment']].corr()
```

	Weekly_Sales	Unemployment
Weekly_Sales	1.000000	-0.106176
Unemployment	-0.106176	1.000000

Correlation is significantly negative suggesting that an increase in unemployment rate tends to decrease weekly sales.

Stores with a strong negative correlation between unemployment and sales will emerge as those most affected by economic conditions.

```
# Group by Store and calculate correlation for each store
store_correlations = df.groupby('Store').apply(lambda x: x[['Weekly_Sales', 'Unemployment']].corr().iloc[0, 1])
store_correlations.sort_values()
```

More negative correlation suggests the store's sales drop more as unemployment increases.

```
# Top 10 suffering stores based on unemployment rate
top_10_stores = store_correlations.sort_values().head(10)
top_10_stores
```

Store	0
38	-0.785290
44	-0.780076
39	-0.384681
42	-0.356355
41	-0.350630
4	-0.337015
17	-0.263600
3	-0.230413
37	-0.221287
5	-0.207043

The top most suffering stores due to unemployment rate are store numbers as follows: 38, 44, 39, 42, 41 4, 17, 3, 37, 5.

b. If the weekly sales show a seasonal trend, when and what could be the reason?

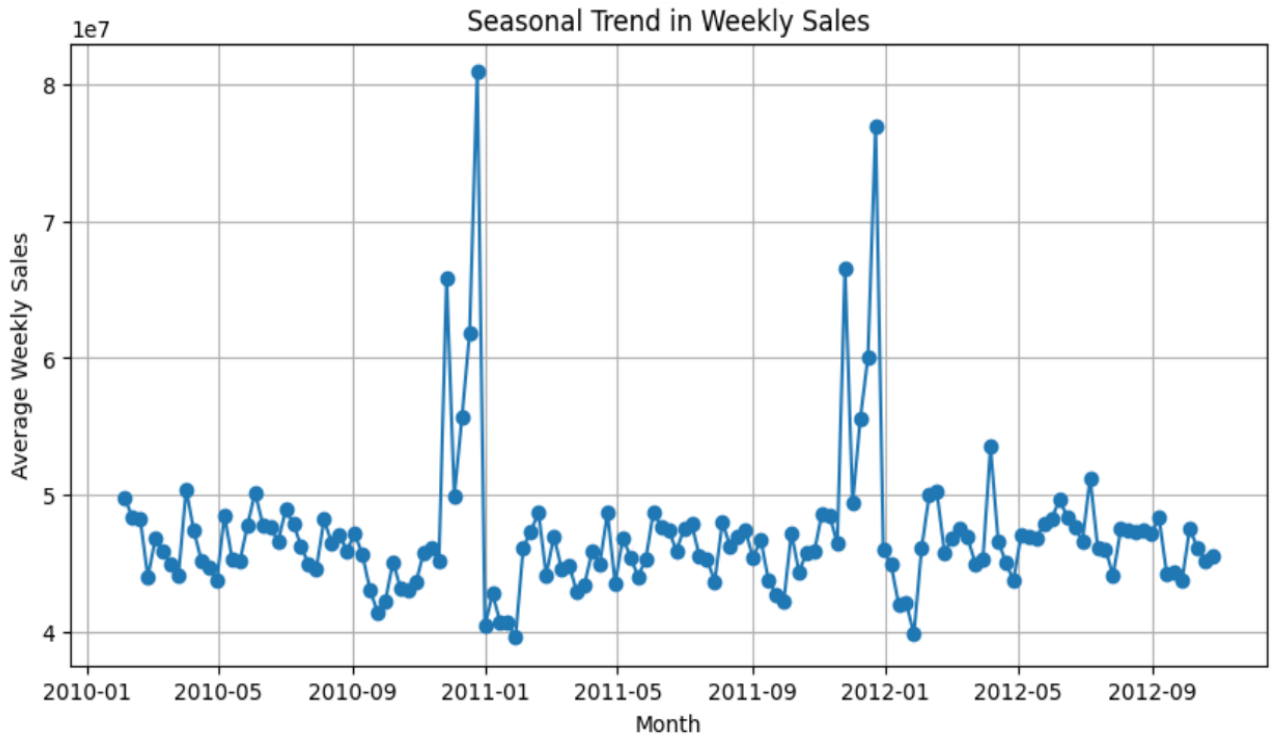
```
# Aggregate weekly_sales
weekly_sales = df.groupby('Date')['Weekly_Sales'].sum().reset_index()
weekly_sales.set_index('Date', inplace=True)
weekly_sales
```

Date	Weekly_Sales
2010-02-05	49750740.50
2010-02-12	48336677.63
2010-02-19	48276993.78
2010-02-26	43968571.13
2010-03-05	46871470.30
...	...
2012-09-28	43734899.40
2012-10-05	47566639.31
2012-10-12	46128514.25
2012-10-19	45122410.57
2012-10-26	45544116.29

143 rows × 1 columns

Plotting the weekly sales of each store.

```
# Plot the seasonal trend
plt.figure(figsize=(10, 5))
plt.plot(weekly_sales.index, weekly_sales.values, marker='o', linestyle='-')
plt.xlabel("Month")
plt.ylabel("Average Weekly Sales")
plt.title("Seasonal Trend in Weekly Sales")
#plt.xticks(range(1, 13))  # Set x-axis from Jan (1) to Dec (12)
plt.grid()
plt.show()
```



Insights from season trend in Weekly Sales.

- There are two significant peaks around last weeks of December both in 2010 & 2011. These indicate notable events or changes during these periods. It's likely due to shopping during Christmas and New Year’s Eve.
- Aside from the spikes, the data generally fluctuates between a particular range, suggesting a stable trend with minor variations throughout the observed period.
- The regularity of the spikes suggests a possible seasonal or annual event impacting the data around late November & December each year.

c. Does temperature affect the weekly sales in any manner?

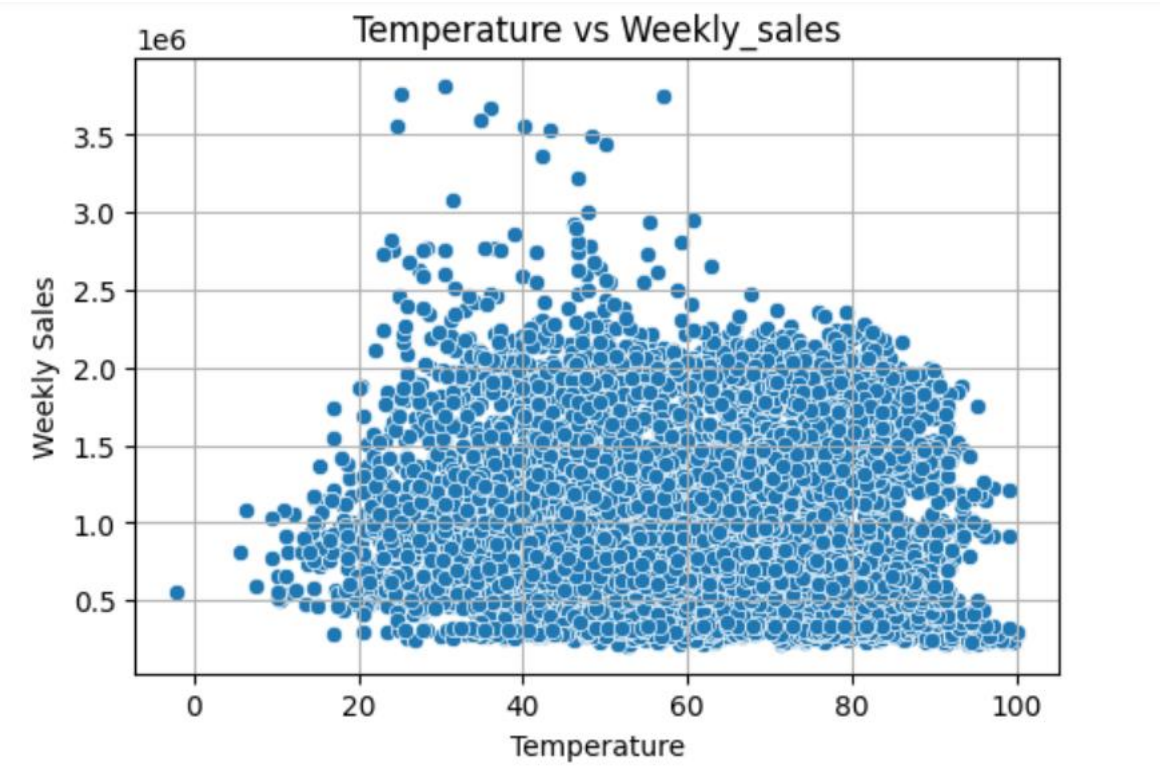
```
df[['Weekly_Sales', 'Temperature']].corr()
```

	Weekly_Sales	Temperature
Weekly_Sales	1.00000	-0.06381
Temperature	-0.06381	1.00000

Correlation is slightly negative suggesting that an increase in temperature tends to decrease weekly sales.

Plotting Temperature vs Weekly_sales:

```
# Temperature vs Weekly_sales
plt.figure(figsize=(6,4))
sns.scatterplot(data=df, x='Temperature', y='Weekly_Sales')
plt.title('Temperature vs Weekly_sales')
plt.xlabel("Temperature")
plt.ylabel("Weekly Sales")
plt.grid(True)
plt.show()
```



Insights from Weekly Sales vs Temperature

- Most data points fall within the temperature range of 20 to 80 degrees, indicating that this is the typical temperature range for Weekly Sales Data.
- There is a significant dense concentration of sales at temperatures between 20 to 60 degrees, suggesting that this temperature range is optimal for higher sales.
- Temperature increases beyond 60 degrees; the density of higher sales data points decreases indicating that higher temperatures negatively impact weekly sales.
- There are fewer data points at extreme temperatures (below 20 degrees and above 80 degrees). These temperatures are less relevant for weekly sales.

- The scatter plot does not show a clear linear relationship between temperature and weekly sales, indicating that other factors might influence sales.

d. How is the Consumer Price index affecting the weekly sales of various stores?

```
# Correlation between Weekly Sales and CPI
df[['Weekly_Sales', 'CPI']].corr()
```

	Weekly_Sales	CPI
Weekly_Sales	1.000000	-0.072634
CPI	-0.072634	1.000000

Negative correlation: Increase in CPI leads to lower weekly sales.

```
# CPI impact per store
store_corr = df.groupby('Store').apply(lambda x: x[['Weekly_Sales', 'CPI']].corr().iloc[0,1])
store_corr
```

Store	0
1	0.225408
2	-0.062431
3	0.317250
4	0.351157

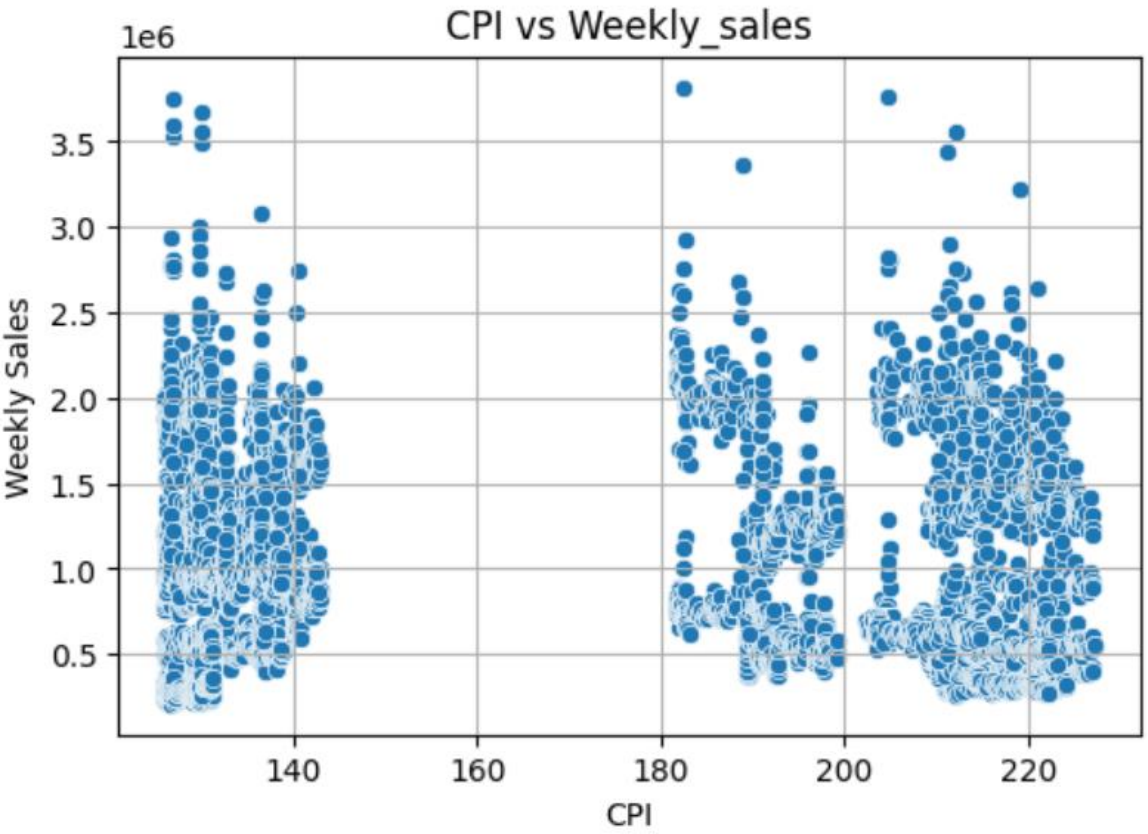
Most effected stores based on CPI value

```
# Most Affected Stores
most_affected_stores = store_corr.sort_values().head(10)
most_affected_stores
```

Store	0
36	-0.915095
35	-0.424107
14	-0.419755
30	-0.298188
43	-0.285686
27	-0.213082
18	-0.193541
15	-0.174467
21	-0.163649
19	-0.155341

Plotting CPI vs Weekly sales

```
# Visualize CPI vs Weekly_sales Relationship
plt.figure(figsize=(6,4))
sns.scatterplot(data=df, x='CPI', y='Weekly_Sales')
plt.title('CPI vs Weekly_sales')
plt.xlabel("CPI")
plt.ylabel("Weekly Sales")
plt.grid(True)
plt.show()
```



Insights from CPI vs Weekly_Sales

- The data points are grouped into three distinct clusters based on CPI values: around 130-150, 180-200, and 210-230. This suggests that certain CPI ranges have different sales patterns.
- First Cluster (CPI 130-150): Weekly sales range from 500K to 3.5 million, with a higher density between 500K and 2.5 million.
- Second Cluster (CPI 180-200): Weekly sales range from 500K to 2.5 million, with a higher density between 500K and 1.5 million.
- Third Cluster (CPI 210-230): Weekly sales range from 500K to 2.5 million, with a higher density between 500K and 1.5 million.

- The data points are widely scattered within each cluster, suggesting there is no clear linear relationship between CPI and Weekly Sales.

e. Top performing stores according to the historical data.

Total and average sales per store as

```
# Total and average sales per store
store_performance = df.groupby('Store')['Weekly_Sales'].agg(['sum', 'mean']).reset_index()
store_performance.columns = ['Store', 'Total_Sales', 'Average_Weekly_Sales']
store_performance.sort_values(by='Total_Sales', ascending=False)
```

Standard deviation of weekly sales of each store

```
# Standard deviation of weekly sales of each store
store_performance['Sales_Stability'] = df.groupby('Store')['Weekly_Sales'].std().reset_index(drop=True)
store_performance
```

So finally, we get top 10 stores according to historical data

```
top_stores = store_performance.sort_values(by='Sales_Stability')
top_stores.head(10)
```

	Store	Total_Sales	Average_Weekly_Sales	Sales_Stability
36	37	74202740.32	518900.281958	21837.461190
29	30	62716885.12	438579.616224	22809.665590
32	33	37160221.96	259861.692028	24132.927322
43	44	43293087.84	302748.866014	24762.832015
4	5	45475688.90	318011.810490	37737.965745
42	43	90565435.41	633324.723147	40598.413260
37	38	55159626.42	385731.653287	42768.169450
2	3	57586735.07	402704.441049	46319.631557
41	42	79565752.43	556403.863147	50262.925530
35	36	53412214.97	373511.992797	60725.173579

Bar graph representing top 10 performing stores

```
top10 = top_stores.head(10)

# Visualise the top10 stores
plt.figure(figsize=(10,4))
sns.barplot(data=top10, x='Store', y='Total_Sales')
plt.xlabel("Store ID")
plt.ylabel("Total Sales")
plt.title("Top 10 Performing Stores Based on historical data")
plt.xticks(rotation=45)
plt.show()
```

We can draw the following conclusions:

- Among the top 10 stores, Store 43 stands out with the highest total sales, depicting exceptional performance compared to others.
- There is significant variation in total sales across these stores, indicating that some stores contribute much more to overall revenue than others.
- Store 33, while part of the top-performing group, has the lowest total sales within this category, suggesting room for improvement compared to its competitors.

f. The worst performing store, and how significant is the difference between the highest and lowest performing stores.

Finding the best and worst performing stores

```
# Finding the best and worst performing stores
best = store_performance.loc[store_performance['Total_Sales'].idxmax()]
worst = store_performance.loc[store_performance['Total_Sales'].idxmin()]
```

```
print("Best Performing Store:")
print(best)
print("\nWorst Performing Store:")
print(worst)
```

```
Best Performing Store:
Store                   2.000000e+01
Total_Sales             3.013978e+08
Average_Weekly_Sales    2.107677e+06
Sales_Stability         2.759006e+05
Name: 19, dtype: float64

Worst Performing Store:
Store                   3.300000e+01
Total_Sales             3.716022e+07
Average_Weekly_Sales    2.598617e+05
Sales_Stability         2.413293e+04
Name: 32, dtype: float64
```

Sales difference & percentage difference in total sales

```
# sales difference & percentage difference in total sales
sales_difference = best['Total_Sales'] - worst['Total_Sales']
percentage_difference = (sales_difference / best['Total_Sales']) * 100
```

```
sales_difference
```

```
264237570.49999997
```

```
percentage_difference
```

```
87.67070533042084
```

Plotting best store vs worst store

```
stores = ['Best Store', 'Worst Store']
sales = [best['Total_Sales'], worst['Total_Sales']]

plt.figure(figsize=(6,4))
plt.bar(stores, sales, color=['blue', 'red'])
plt.xlabel("Store Type")
plt.ylabel("Total Sales")
plt.title("Best vs. Worst Performing Store Sales")
plt.show()
```

Valuable insights are:

- The best-performing store has substantially higher total sales than the worst-performing store. This indicates a major disparity in store performance within the dataset.

- The sales value of the best-performing store can serve as a benchmark for evaluating other stores and identifying improvement opportunities for lower-performing ones.
- The clear gap suggests that the company could focus its efforts on analysing factors contributing to the success of the top store and apply these insights to boost the performance of struggling stores.
- Potential for further analysis involves investigating operational strategies, location demographics, or market conditions that might explain the significant difference in sales.

Data Preprocessing Steps

1. Basic information about the Walmart Dataset:

```
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   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
```

- The dataset consists of 8 columns and 6435 entries (rows), spanning data about 45 stores over weekly intervals.
- Store column has integer type data.
- Date column is object type data.

- Weekly_Sales has float datatype as sales value has continuous numbers.
- Holiday_Flag again have integer datatype.
- Temperature has float datatype for numbers with a decimal point.
- Fuel_price has float datatype for numbers with a decimal point.
- CPI has float datatype for numbers with a decimal point.
- Unemployment has float datatype for numbers with a decimal point showing percentage values.

2. Checking for null values:

```
## Checking for nul values
df.isnull().sum()
```

	0
Store	0
Date	0
Weekly_Sales	0
Holiday_Flag	0
Temperature	0
Fuel_Price	0
CPI	0
Unemployment	0

dtype: int64

- Displays a summary of missing values in the Walmart dataset using the **df.isnull().sum()** command in Python. This method checks each column for null values, and the output confirms the data's integrity.

- Every column in the dataset has 0 null values, meaning the dataset is complete and contains no missing entries.

- Since there are no missing values, preprocessing steps such as data imputation or handling null values can be avoided at ease.

- The complete dataset is ready for detailed analysis, making it easier to explore relationships between sales and external factors, forecast trends, and develop strategies to improve store performance.

3. Describing data

```
## Describing data
df.describe()
```

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999151
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875885
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879000
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891000
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874000
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622000
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000

The dataset summary provided by the function **df.describe()** reveals following key statistical insights about the data:

- Total weekly sales vary widely, from a minimum of $209,986.20 to a maximum of $3,818,686.00. The mean weekly sales per store is around $1,046,965, with a significant standard deviation

of $564,366.60, reflecting a notable variance in sales performance.

- Weekly average temperatures range from-2.06°F (extremely cold) to 100.14°F (hot), with a mean of 60.66°F. This wide range suggests that stores are located in diverse climatic regions, potentially influencing shopping behaviours.
- Fuel prices vary from $2.47 to $4.47, with an average of $3.36. Such variations might impact consumer mobility and logistics costs.
- The CPI ranges from 126.06 units to 227.23 units, with an average of 171.58 units. This indicates fluctuations in the cost of consumer goods over the dataset's timeframe.
- Unemployment rates vary from 3.88% (low) to 14.31% (high), with an average of 7.99%. This wide range reflects different economic conditions across regions.

The dataset is diverse, covering a wide range of stores, sales figures, environmental factors (temperature, fuel price), and economic indicators (CPI, unemployment). The high variability in key columns like Weekly_Sales, Temperature, and Unemployment suggests significant differences across stores and regions. These statistics provide a foundation for deeper analysis, such as identifying patterns, trends, and correlations.

4. Convert date feature to datetime type

```
# convert date feature to datetime type
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
df['Date']
```

	Date
0	2010-02-05
1	2010-02-12
2	2010-02-19
3	2010-02-26
4	2010-03-05
...	...
6430	2012-09-28
6431	2012-10-05
6432	2012-10-12
6433	2012-10-19
6434	2012-10-26

6435 rows × 1 columns

Converting the 'Date' column from a string format to a proper datetime format. This was achieved using the Python pandas function **to_datetime().** The command takes the 'Date' column in the DataFrame, which is originally in the format **DD-MM-YYYY**, and converts it into the standard datetime type in pandas, formatted as

YYYY-MM-DD. This ensures consistency and makes it compatible for time-based operations like sorting, filtering, or aggregating data.

Finally, our pre-processed dataframe looks like this where 'Date' column has been assigned the new **datetime** datatype.

```
df.head()
```

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	2010-02-05	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	2010-02-12	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	2010-03-05	1554806.68	0	46.50	2.625	211.350143	8.106

```
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   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: datetime64[ns](1), float64(5), int64(2)
```

5. Creating a separate year, month, day columns

```
# creating a seperate year, Month , day columns
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
```

```
df.head()
```

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Year	Month	Day
0	1	2010-02-05	1643690.90	0	42.31	2.572	211.096358	8.106	2010	2	5
1	1	2010-02-12	1641957.44	1	38.51	2.548	211.242170	8.106	2010	2	12
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	2010	2	19
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	2010	2	26
4	1	2010-03-05	1554806.68	0	46.50	2.625	211.350143	8.106	2010	3	5

The Python code snippet converts the Date column into three separate components: Year, Month and Day.

For example, if we have a date **2010-02-05**, it can be represented as

Date- 5,

Month- 02,

Year- 2010.

This preprocessing step enhances the dataset's usability for time-based exploration, trend analysis, and forecasting. Breaking down the Date column into smaller components makes it easier to identify patterns and correlations that can inform business strategies.

6. Correlation of other features with Weekly Sales

```
## Correlation of other features with Weekly Sales
df.corr()['Weekly_Sales'].sort_values(ascending=False)
```

	Weekly_Sales
Weekly_Sales	1.000000
Month	0.076143
Holiday_Flag	0.036891
Fuel_Price	0.009464
Date	0.006949
Day	-0.017409
Year	-0.018378
Temperature	-0.063810
CPI	-0.072634
Unemployment	-0.106176
Store	-0.335332

Positive Correlations:

- Month- 0.0761 Weakly positively correlated, possibly indicating some periodic sales trends.
- Holiday_Flag: 0.0369 Weak positive correlation suggests a small uptick in sales during holiday weeks.
- Fuel_Price: 0.0095 Negligible positive correlation, implying fuel price variations do not strongly influence sales.

- Date: 0.0069 Almost no correlation, which is expected unless analysed within specific timeframes

Negative Correlations:

- Day: -0.0174 and Year-0.0184, Minimal negative correlation, showing no significant influence on weekly sales.
- Temperature:-0.0638 Weak negative correlation, suggesting that higher temperatures may slightly reduce sales.
- CPI: -0.0726 Weak negative correlation, reflecting minor sales decline with rising consumer price levels.
- Unemployment: -0.1062, The strongest negative correlation in the dataset, indicating that higher unemployment significantly impacts sales negatively.
- Store: -0.3353, Moderate negative correlation, showing that sales vary across stores, possibly due to location or size differences.

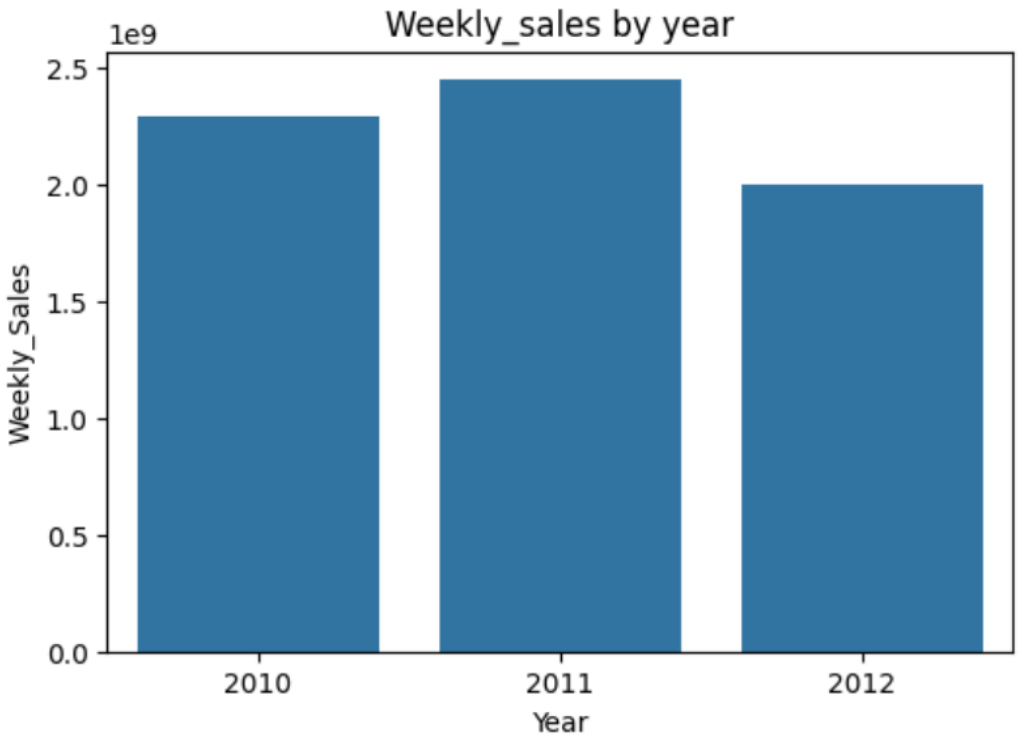
The analysis reveals that Unemployment has the most substantial negative impact on sales, while Month and Holiday_Flag show the most positive influence, though weakly.

7. Weekly sales by year

```
## weekly sales by year
weekly_sales_by_year = df.groupby(['Year'])['Weekly_Sales'].sum().reset_index()
weekly_sales_by_year
```

	Year	Weekly_Sales
0	2010	2.288886e+09
1	2011	2.448200e+09
2	2012	2.000133e+09

```
# Plotting Weekly_sales by year
plt.figure(figsize=(6,4))
sns.barplot(data=weekly_sales_by_year, x='Year', y='Weekly_Sales')
plt.title('Weekly_sales by year')
plt.show()
```

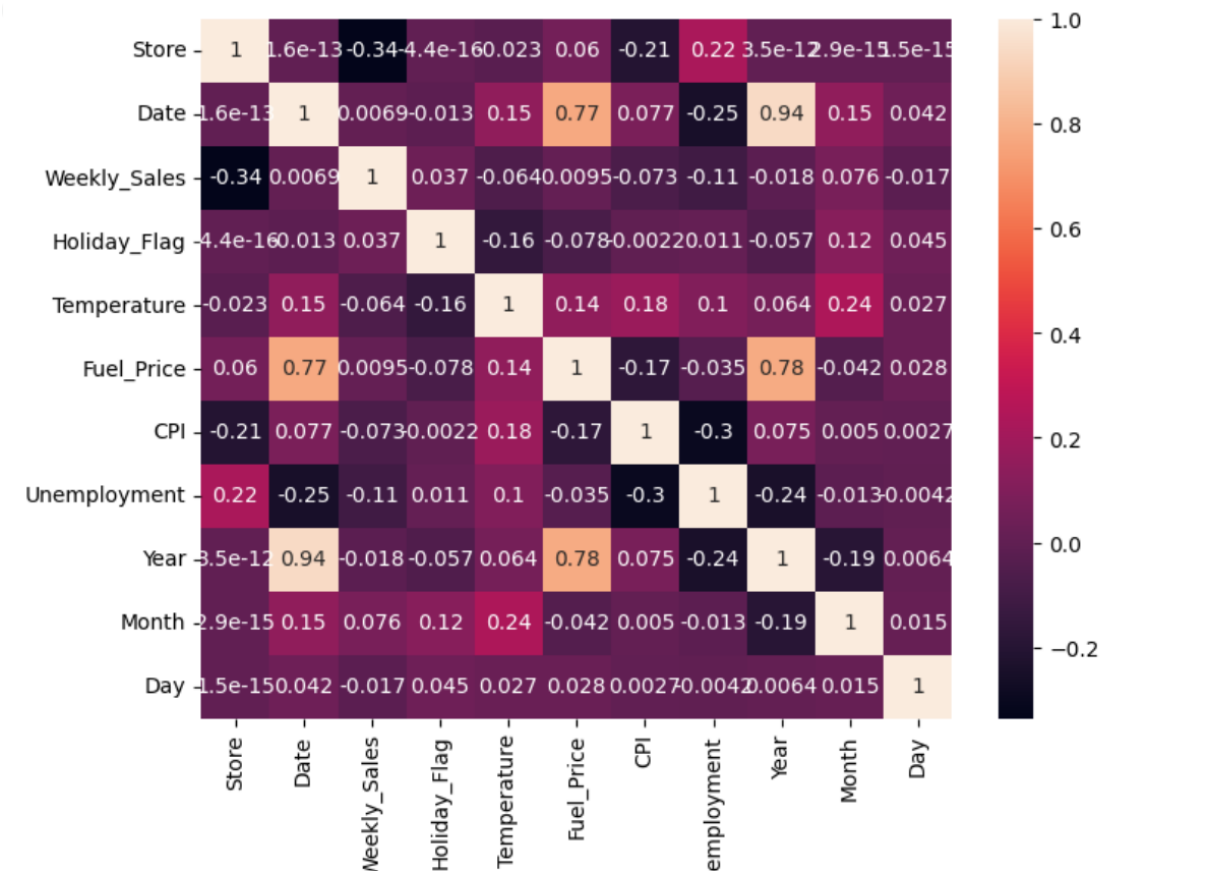


- 2011 stands out as the year with the highest weekly sales.
- 2010 ranks second, with sales slightly lower than in 2011.

- 2012 has the lowest weekly sales, reflecting a decline in sales trends over the years.

8. Correlation Heatmap

```
[ ]  ## Correlation Heatmap
     plt.figure(figsize=(8,6))
     sns.heatmap(df.corr(), annot=True)
     plt.show()
```



9. Taking date as index for time series analysis

```
data.set_index('Date', inplace=True)
data.head()
```

Date	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
2010-02-05	1	1643690.90	0	42.31	2.572	211.096358	8.106
2010-02-12	1	1641957.44	1	38.51	2.548	211.242170	8.106
2010-02-19	1	1611968.17	0	39.93	2.514	211.289143	8.106
2010-02-26	1	1409727.59	0	46.63	2.561	211.319643	8.106
2010-03-05	1	1554806.68	0	46.50	2.625	211.350143	8.106

After setting 'Date' as the index, it becomes a key element for organizing the dataset chronologically. This is especially beneficial for time-series analysis.

10. Total sales by date

```
# Total sales by date
total_sales_by_date = df.groupby(['Date'])['Weekly_Sales'].sum().reset_index()
total_sales_by_date.set_index('Date', inplace=True)
total_sales_by_date
```

Date	Weekly_Sales
2010-02-05	49750740.50
2010-02-12	48336677.63
2010-02-19	48276993.78
2010-02-26	43968571.13
2010-03-05	46871470.30

- The dataframe is grouped by the 'Date' column to aggregate weekly sales for each unique date.

- After grouping, the sum() function computes the total weekly sales for each date across all stores.
- The result includes a new summary table with two columns, ‘Date’ represents each unique date & ‘Weekly_Sales’ shows the summed weekly sales for that date.
- reset_index() is used to ensure the output is structured as a regular DataFrame instead of retaining group keys as an index.
- set_index(‘Date’, inplace=True) then sets the ‘Date’ column as the index for easier time-series manipulation.

Choosing the Algorithm for the Project

The ARIMA (AutoRegressive Integrated Moving Average) model is a widely-used statistical method for time series forecasting. It is designed to analyse and predict future values in a time series by understanding the past data patterns.

Components of ARIMA:

I. AR (p): AutoRegressive is a regression model that utilizes the dependent relationship between a current observation and observations over a previous period. An auto regressive (*AR(p)*) component refers to the use of past values in the regression equation for the time series.
II. I (d): Integrated component uses differencing of observations (subtracting an observation from observation at the previous time step) in order to make the time series stationary. Differencing involves the subtraction of the current values of a series with its previous values d number of times.
III. MA (q): MovingAverage model uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. A moving average component depicts the error of the model as a combination of previous error terms. The order *q* represents the number of terms to be included in the model.

Parameters (p, d, q):

- p: The number of lag observations in the AR component.
- d: The degree of differencing applied to make the series stationary.
- q: The size of the moving average window for the MA component.

ORDER SELECTION WITH ACF & PACF:

Arguably the most common method for identifying the proper orders of an *ARMA/ARIMA* model is with using the **ACF** (Auto Correlation Function) and the **PACF** (Partial Auto Correlation Function).

- **ACF:** The Autocorrelation Function computes the autocorrelations for a given time series. Autocorrelation is the correlation between observations of a time series separated by *k* time steps.
- **PACF:** The Partial Autocorrelation Function computes the strength of relationship with also accounting for any intermediate lags.

Both ACF and PACF produce plots for up to any arbitrary number of lags, which easily visualizes the auto correlational strength that each lag has on a given observation.

ARIMA Model

```
from statsmodels.tsa.arima.model import ARIMA
```

```
from pmdarima import auto_arima
```

```
auto_model = auto_arima(x_train['Weekly_Sales'], seasonal=True, trace=True)
print("Best order (p, d, q):", auto_model.order)
```

Observed Best Model via AutoARIMA

```
Best model:  ARIMA(5,0,0)(0,0,0)[0] intercept
Total fit time: 7.077 seconds
Best order (p, d, q): (5, 0, 0)
```

Predictions by ARIMA Model:

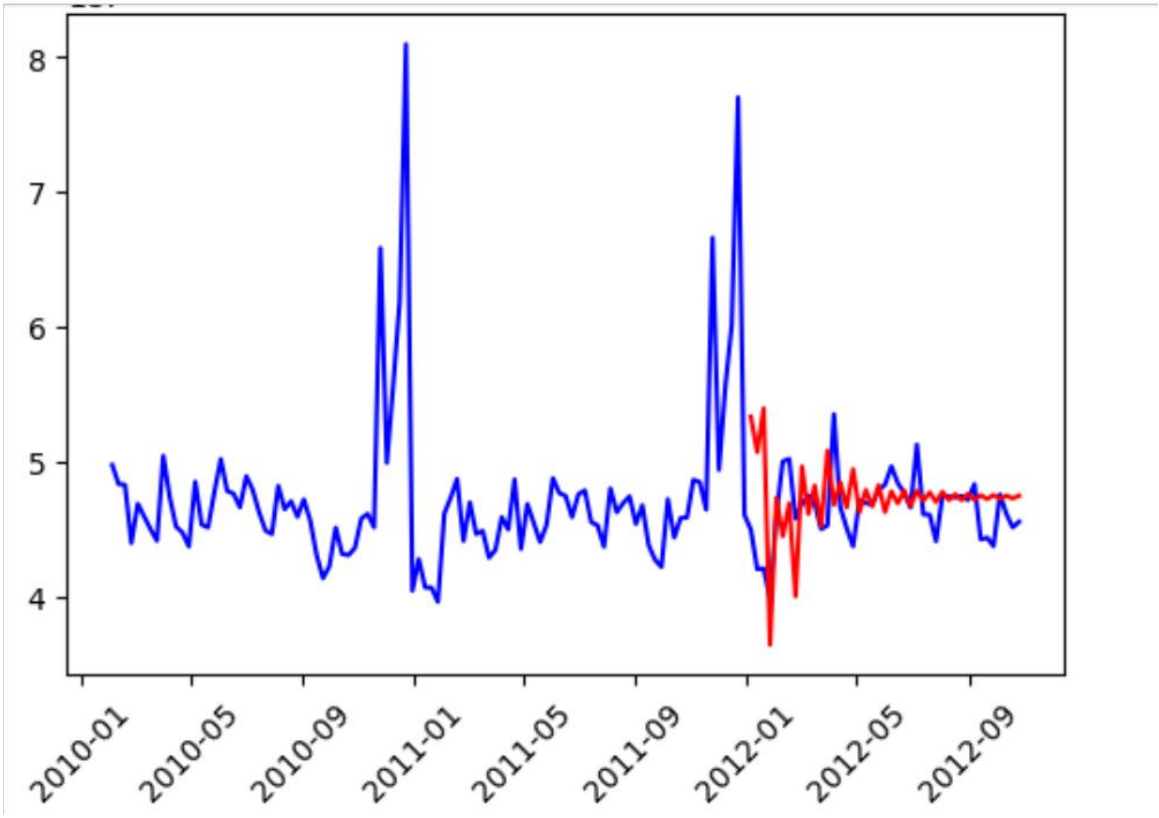
```
model = ARIMA(x_train['Weekly_Sales'], order=auto_model.order)
model = model.fit()
```

```
new_data = total_sales_by_date
new_data['Arima_Predicted'] = model.predict(start=len(x_train), end =len(x_train)+len(x_test)-1)
new_data
```

Plotting the Predicted Data:

```
plt.figure(figsize=(6,4))

plt.plot( new_data['Weekly_Sales'], color='blue')
plt.plot( new_data['Arima_Predicted'], color='red')
plt.xticks(rotation=45)
plt.show()
```

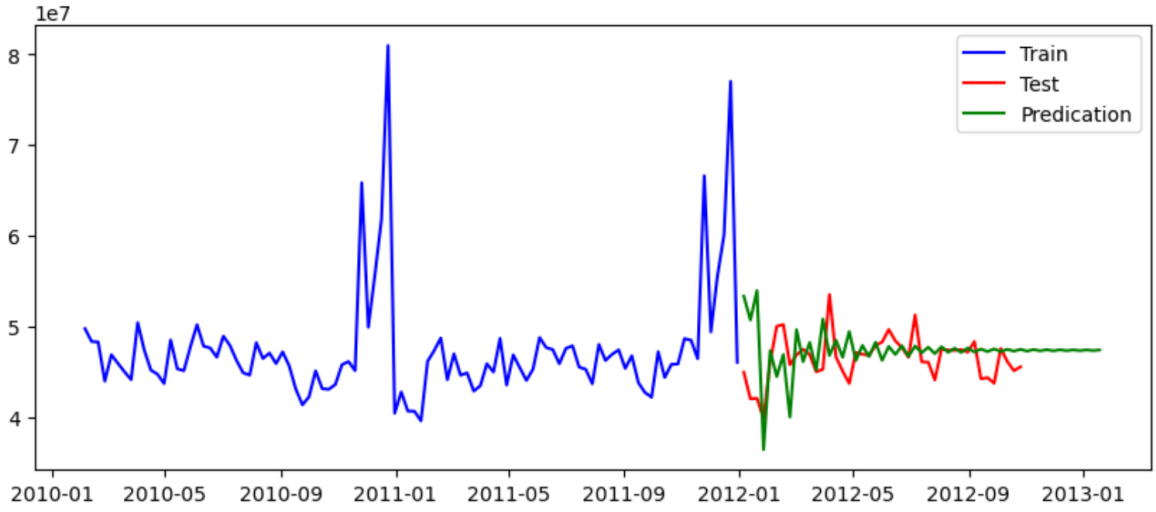


Forecast for next 12 Weeks:

```
forecast = model.forecast(steps=55)
print("Forecasted Weekly Sales for the next 12 weeks:")
print(forecast.tail(12))
```

```
Forecasted Weekly Sales for the next 12 weeks:
2012-11-02    4.727540e+07
2012-11-09    4.745051e+07
2012-11-16    4.730759e+07
2012-11-23    4.743546e+07
2012-11-30    4.730788e+07
2012-12-07    4.742768e+07
2012-12-14    4.732504e+07
2012-12-21    4.741270e+07
2012-12-28    4.732902e+07
2013-01-04    4.740954e+07
2013-01-11    4.733763e+07
2013-01-18    4.739916e+07
Freq: W-FRI, Name: predicted_mean, dtype: float64
```

Forecast Plotting:



But after data analysis it's observed that ARIMA Model has not performed efficiently in future sales forecasting. Hence choosing the SARIMAX (Seasonal AutoRegressive Integrated Moving Average with Exogenous Variables) model for time series forecasting is a great decision, especially for a dataset like this.

I have chosen the SARIMAX model for this project for the following reasons:

1. SARIMAX allows to include external variables (like CPI, Unemployment, Fuel_Prices, or even Holiday_Flags) as predictors for the model. These factors can significantly impact retail sales, and SARIMAX can capture their effects.
2. SARIMAX accounts for seasonality explicitly. In your dataset, weekly sales trends may vary based on seasonal holidays or shopping patterns, and SARIMAX can model these recursive seasonal behaviours effectively.

3. If the time series exhibits trends or seasonal patterns, SARIMAX handles them efficiently by differencing (both regular and seasonal trends), ensuring accurate predictions even for non-stationary data.
4. The combination of AutoRegressive (AR), Integrated (I), and Moving Average (MA) components allows SARIMAX to capture complex temporal dependencies in sales data. This includes relationships between past values, past errors, and current outcomes.
5. SARIMAX supports a wide range of parameter tuning, making it adaptable to different datasets. By adjusting the seasonal (p, d, q, s) and non-seasonal (p, d, q) components, we can develop the model to capture both long-term and short-term trends.
6. Including exogenous variables improves forecast accuracy by leveraging additional information about external factors influencing sales. This makes SARIMAX more robust and realistic compared to models that rely solely on past data.
7. Including other dependent variables improves forecast accuracy by leveraging additional information about external factors influencing sales. This makes SARIMAX more robust and realistic compared to models that rely solely on past data.
8. Retail sales are influenced by multiple factors, including economic conditions, seasonal demand, and local events. SARIMAX's ability to combine time series components with external predictors makes it particularly well-suited for such scenarios.

Hence, SARIMAX's ability to handle seasonality and external influences makes it a powerful choice for your Walmart dataset.

SARIMAX Model

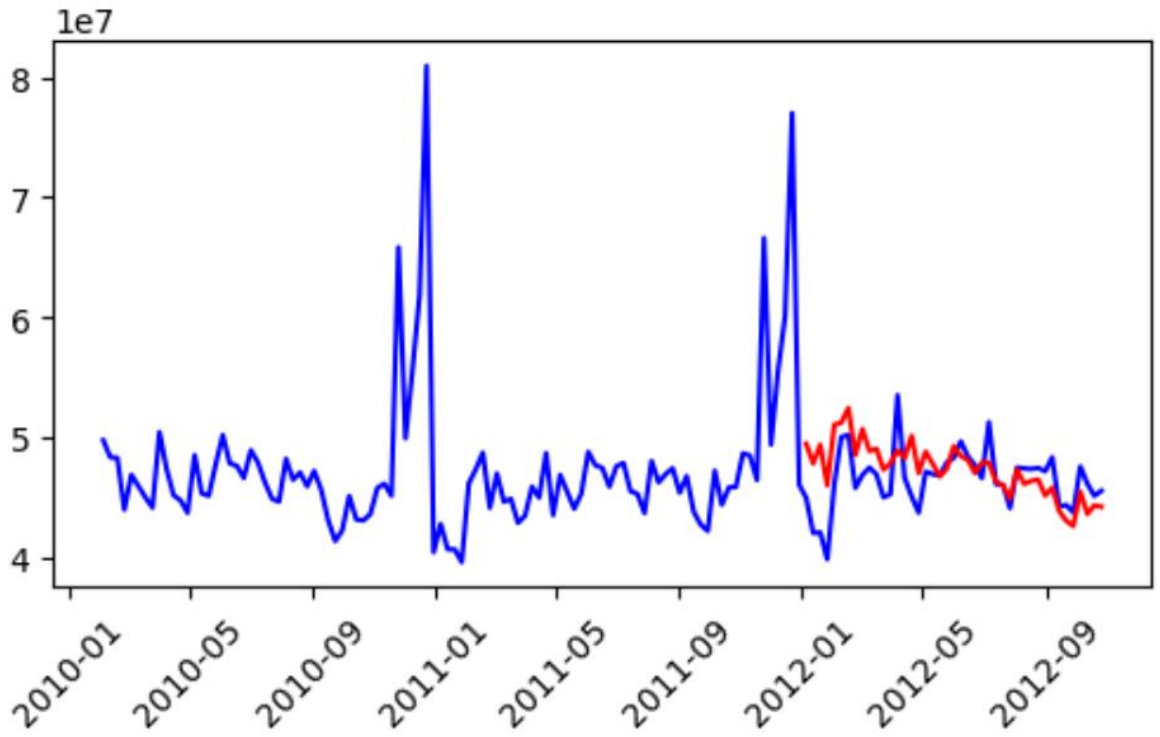
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
model = SARIMAX(x_train['Weekly_Sales'], order=auto_model.order, seasonal_order=(5,0,0,52))
model = model.fit()
```

Predictions by SARIMAX Model:

```
new_data['Sarima_Predicted'] = model.predict(start=len(x_train), end =len(x_train)+len(x_test)-1)
new_data
```

Plotting the Predicted Data:



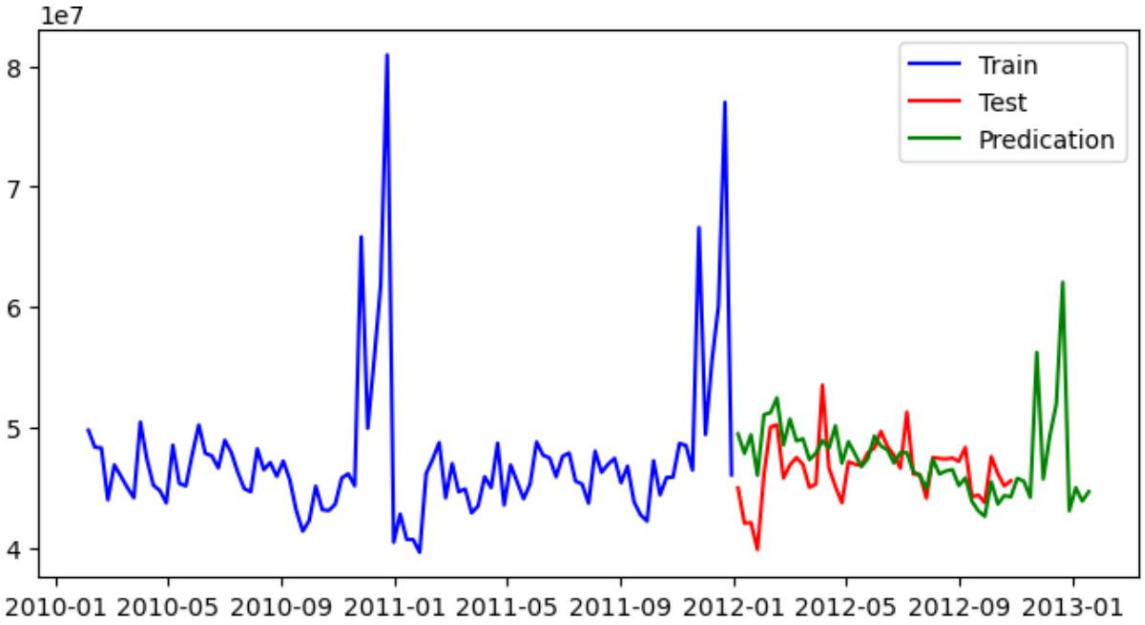
Forecast for next 12 Weeks:

```
future = model.forecast(steps=55)
print("Forecasted Weekly Sales for the next 12 weeks:")
print(future.tail(12))
```

```
Forecasted Weekly Sales for the next 12 weeks:
2012-11-02    4.575298e+07
2012-11-09    4.551310e+07
2012-11-16    4.415825e+07
2012-11-23    5.621986e+07
2012-11-30    4.569667e+07
2012-12-07    4.929167e+07
2012-12-14    5.191657e+07
2012-12-21    6.204544e+07
2012-12-28    4.305538e+07
2013-01-04    4.497664e+07
2013-01-11    4.385153e+07
2013-01-18    4.464253e+07
```

Forecasted Data plotting:

```
plt.figure(figsize=(8,4))
plt.plot(x_train, color='blue', label='Train')
plt.plot(x_test, color='red', label='Test')
plt.plot(future, color='green', label='Predication')
plt.legend(loc='best')
plt.show()
```


1e7
8
7
6
5
4
Train
Test
Predication
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
2013-01

Assumptions

The following assumptions were made in order to create the model for Walmart project.

1. Assumptions for the Dataset:

- The sales data provided is accurate, consistent, and complete for the specified period from February 2010 to October 2012.
- No errors or discrepancies exist in the dataset like incorrect sales values or missing entries.
- Each store operates independently, and external factors affecting one store may not necessarily impact others.
- Factors such as holidays, economic indicators (CPI, Unemployment), and environmental conditions (Temperature, Fuel_Prices) influence sales trends.

2. Assumptions for the Time Series Analysis:

- The dataset can be transformed to achieve stationarity (i.e., constant mean and variance) for effective time series modelling.
- There are recurring seasonal patterns in sales, such as holiday-related peaks, which can be captured by the SARIMAX model.
- The sales trends observed in the historical data are sufficient for making reliable forecasts.

3. Assumptions for the Business Context:

- Future predictions assume no extreme economic changes like recession, major policy changes that could drastically alter sales behaviour.

- Holiday weeks typically lead to higher sales compared to non-holiday weeks, and this trend is consistent across all stores.
- Factors like CPI, unemployment, and fuel prices are treated as external (exogenous) variables that influence sales rather than being driven by sales.
- The analysis and forecasting outputs align with Walmart's goal to identify sales drivers and create actionable strategies to address financial challenges.

4. Assumptions for Modelling:

- The relationships between sales and external variables are assumed to be linear or relatively straightforward for the model to capture effectively.
- Weekly sales data is sufficient for trend and seasonal analysis without needing more detailed daily-level insights.
- The forecasting model generalizes well to predict future sales, assuming no major structural changes occur in Walmart’s operations.

Model Evaluation and Technique

Model evaluation assesses the performance and reliability of a predictive model, ensuring its outputs are accurate and meaningful. For the Walmart project, this involves examining how well the model forecasts sales and identifying areas for improvement. Key aspects of model evaluation include:

1. Splitting Data:

- Training Set: Used to build the model.
- Testing Set: Used to evaluate the model's performance on unseen data.

2. Evaluation Metrics:

- Mean Absolute Error (MAE): Measures the average difference between predicted and actual values.
- Root Mean Squared Error (RMSE): Penalizes larger errors, emphasizing major deviations.
- Mean Absolute Percentage Error (MAPE): Represents the error as a percentage of actual values.

3. Residual Analysis:

- Residuals are the differences between actual and predicted values. Analyse residuals to ensure they are randomly distributed, with no patterns, which validates the model's assumptions.

For the Walmart sales dataset, the SARIMAX (Seasonal AutoRegressive Integrated Moving Average with Exogenous Variables) model is a powerful approach. It combines traditional ARIMA time-series

forecasting with the ability to include external predictors like holidays, economic indicators, or weather conditions.

The evaluation report suggests the following:

- Metrics like RMSE & MAE indicate how closely the model's predictions align with actual sales data.
- A low RMSE or MAE suggests the model is making accurate predictions, while high values indicate room for improvement.
- Residuals between actual and predicted values should resemble random white noise, with no specific patterns.
- If residuals show trends or seasonality, it might imply that the model needs adjustments, such as better parameter tuning.
- Exogenous factors like Holiday_Flag, CPI, and Unemployment are incorporated, and their contribution to sales prediction can be inferred from the model coefficients.
- SARIMAX models are designed to handle seasonal fluctuations and long-term trends. The evaluation will confirm whether the model successfully captures holiday-related spikes or cyclical sales trends.
- Evaluation metrics across different subsets of data (e.g., training and testing) demonstrate how well the model generalizes. Consistent performance on testing data suggests the model is robust and reliable for future forecasting.
- If the model struggles with accuracy or residuals show systematic patterns, improvements could include tuning hyperparameters (p, d, q, P, D, Q, s), including additional external variables for better predictive power.

Inferences from the Project

The comparison between the Weekly_Sales (actual sales) and SARIMA_Predicted (forecasted sales) columns provide key inferences about the performance and reliability of the SARIMA model in forecasting Walmart's sales.

1. Model Accuracy

    - On 2012-09-28, the actual sales were 43,734,899.40, while the SARIMA prediction was 42,601,950.00. The relatively small difference indicates that the model captures trends effectively.
    - he consistency in low prediction errors across many dates suggests the SARIMA model is reliable.

2. Error Patterns

    - The difference (residual) between actual and predicted values can reveal systematic patterns. If errors are random, the model is performing well and if errors are larger during certain weeks (e.g., holiday weeks), this may indicate the need for refinement in capturing holiday effects.

3. Business Planning

    - Predicted values can assist Walmart in forecasting future sales, optimizing inventory, scheduling promotions, and preparing for high-demand weeks.

4. Seasonal Trends

    - SARIMA successfully predicts recurring patterns in sales, such as holiday spikes or seasonal slowdowns. The alignment between

actual and predicted sales confirms its ability to account for seasonality in the data.

5. Insights for Store-Level Strategies

- By analysing how closely SARIMA predicts weekly sales for different time periods, Walmart can identify trends that are consistent across stores and apply strategic changes accordingly.

This comparison of actual and predicted sales validates SARIMA's effectiveness in capturing overall trends while also highlighting areas for refinement.

Future Possibilities and Limitations

Future Possibilities can be-

1. Enhanced Forecasting: The model can be expanded to provide predictions for longer time horizons, enabling Walmart to plan future strategies for inventory, staffing, and promotions more effectively.
2. Real-Time Analytics: The project could evolve to predict and analyse sales data in real-time, offering immediate insights for operational decisions.
3. Geographic Scope: Extend the analysis to include data from global Walmart stores to identify region-specific trends and seasonal influences.
4. External Variables: Additional external factors, such as competitor data, social media sentiment, weather data, and marketing campaigns, can enhance the model's predictive power.
5. Dynamic Recommendations: The insights could be used to recommend dynamic pricing strategies, targeted marketing, and customized promotions for different stores.
6. Store-Level Optimization: Insights from the project can support various strategies for each store, improving their individual profitability and customer satisfaction.

Limitations-

1. Dependence on Historical Data: The model relies heavily on past data to forecast future trends. Any drastic changes in customer behaviour or economic conditions may lead to inaccurate predictions.
2. Handling of Unforeseen Events: Events like natural disasters, pandemics, or sudden economic policy shifts are difficult to account for in the model.
3. Seasonality Challenges: While SARIMAX captures seasonality effectively, irregular or evolving patterns may limit its ability to adapt and capture to new trends.
4. Impact of External Variables: Exogenous variables like CPI and unemployment are assumed to have linear relationships with sales. Complex, nonlinear interactions might not be fully captured by SARIMAX.
5. Scalability: As the dataset grows in size or complexity, the current model may require significant computational resources or modifications to remain effective.

The Walmart sales analysis project provides a solid foundation for understanding trends and forecasting sales. However, addressing these limitations while embracing future possibilities can ensure the project remains relevant, accurate, and impactful over time.

Conclusion

The Walmart sales forecasting project is a comprehensive initiative aimed at analysing historical sales data, identifying patterns, and providing actionable insights. Here's an in-depth conclusion that encapsulates the findings, model effectiveness, and business implications.

The primary goal of the project was to analyse Walmart's weekly sales data across its 45 stores and predict future sales trends. By considering factors like holidays, fuel prices, temperature, Consumer Price Index (CPI), and unemployment rates, the project sought to uncover the underlying drivers of sales performance and offer strategic recommendations to improve profitability.

Based on the thorough analysis of Walmart's dataset from 45 stores over specific time intervals, it is evident that holidays have a significant impact on the store's sales. The observations clearly indicate that during holiday periods, the store experiences a notable increase in sales. This finding highlights the importance of accounting for the holiday factor when predicting weekly sales for any store in a similar domain. However, it is crucial to note that while holidays have a discernible impact on sales, other factors may also influence the store's performance. To build more accurate and robust predictive models, it is essential to consider additional variables such as weather conditions, promotional activities, and economic indicators that may interact with the holiday factor.

The initial data exploration and analysis yielded valuable insights:

- Seasonal Patterns: Sales data exhibited seasonal peaks, with significant increases during holiday periods.
- Influence of Economic Indicators: CPI and unemployment rates showed correlations with weekly sales, reflecting the sensitivity of consumer spending to economic conditions.
- Temperature Variations: Slight fluctuations in sales were observed based on temperature, indicating a potential impact of weather on shopping behaviour.
- Store-Level Variability: Sales performance varied significantly across stores, highlighting the need for store-specific strategies.

The SARIMAX (Seasonal AutoRegressive Integrated Moving Average with Exogenous Variables) model was chosen for its ability to capture seasonal trends and recurring patterns in weekly sales, incorporate exogenous variables like holidays, unemployment, and CPI, which significantly influence sales and handle the non-stationarity present in the data through differencing and seasonal adjustments.

The Walmart sales forecasting project successfully leveraged the SARIMAX model to analyse and predict weekly sales trends. By uncovering the drivers of sales performance and identifying actionable insights, the project provides Walmart with the tools to enhance operational efficiency, improve customer satisfaction, and drive profitability. While the model performed well, opportunities for refinement and expansion exist, ensuring that Walmart can continue adapting to an ever-changing retail landscape.