CAPSTONE PROJECT

WALMART DATASET

Submitted by-

Siddharth Meher

TABLE OF CONTENT

PROBLEM STATEMENT

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. You are a data scientist, who has to come up with useful insights using the data and make prediction models to forecast the sales for X number of months/years.

Task to perform:-

1. Using data, come up with useful insights that can be used by each of the stores to improve in various areas.
2. Forecast the sales for each store for the next 12 weeks.

PROJECT OBJECTIVE

The primary objective of this project is to develop accurate and reliable time series forecasting models for predicting the weekly sales of Walmart stores. Accurate sales predictions are crucial for Walmart's inventory management, resource planning, and overall business strategy. By leveraging historical sales data, we aim to build robust models that can provide valuable insights and aid decision-making processes.

Sales Optimization: One of the main goals of this project is to optimize sales for each Walmart store. By accurately predicting weekly sales, store managers can make informed decisions about inventory replenishment, ensuring that they have sufficient stock to meet customer demand without overstocking and tying up capital unnecessarily. This optimization can lead to cost savings and improved profitability for the company.

Resource Planning: Effective resource planning is essential for Walmart to efficiently allocate its resources, such as staff, space, and promotional activities. By forecasting sales accurately, Walmart can ensure that the right number of staff members are present during peak sales periods, leading to better customer service and increased customer satisfaction.

Seasonal Demand Forecasting: Walmart experiences fluctuations in sales due to various seasonal factors, such as holidays, weather conditions, and special events. The project aims to capture these seasonal patterns in the data and create seasonal forecasting models using SARIMA to predict sales accurately during different seasons. This will enable Walmart to proactively plan for peak periods and effectively manage its inventory.

Trend Analysis: Understanding long-term trends in sales is essential for strategic decision-making. The project aims to analyze historical sales data to identify any underlying trends, such as overall sales growth or decline, and assess their impact on future sales projections. This information can help Walmart adapt its business strategies and stay competitive in the market.

DATA DESCRIPTION

The dataset used for this project contains historical sales data for various Walmart stores. 6435 rows of data are present in 8 columns.

Feature Name	Description
Store	Store Number
Date	Week Of Sales
Weekly_sales	Sales of given store in that week
Holiday_Flag	If it is a holiday Week
Temperature	Temperature on Day of Sales
Fuel_Price	Cost of fuel in the region
CPI	Consumer Price Index
Unemployment	Unemployment Rate

Data Description: The data description showing the total count, mean, standard deviation(std), minimum,etc. for the numeric features is as shown in the Table

	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

Data Types of all the column present in the dataset:

```
data.dtypes
```

```
Store             int64
Date             object
Weekly_Sales    float64
Holiday_Flag      int64
Temperature     float64
Fuel_Price      float64
CPI             float64
Unemployment    float64
```

Count of Null values in the data set features:

```
#Checking NULL values
data.isnull().sum()
```

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

Correlation between the features:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
Store	1.000000e+00	-0.335332	-4.386841e-16	-0.022659	0.060023	-0.209492	0.223531
Weekly_Sales	-3.353320e-01	1.000000	3.689097e-02	-0.063810	0.009464	-0.072634	-0.106176
Holiday_Flag	-4.386841e-16	0.036891	1.000000e+00	-0.155091	-0.078347	-0.002162	0.010960
Temperature	-2.265908e-02	-0.063810	-1.550913e-01	1.000000	0.144982	0.176888	0.101158
Fuel_Price	6.002295e-02	0.009464	-7.834652e-02	0.144982	1.000000	-0.170642	-0.034684
CPI	-2.094919e-01	-0.072634	-2.162091e-03	0.176888	-0.170642	1.000000	-0.302020
Unemployment	2.235313e-01	-0.106176	1.096028e-02	0.101158	-0.034684	-0.302020	1.000000

DATA PREPROCESSING & PREPARATION

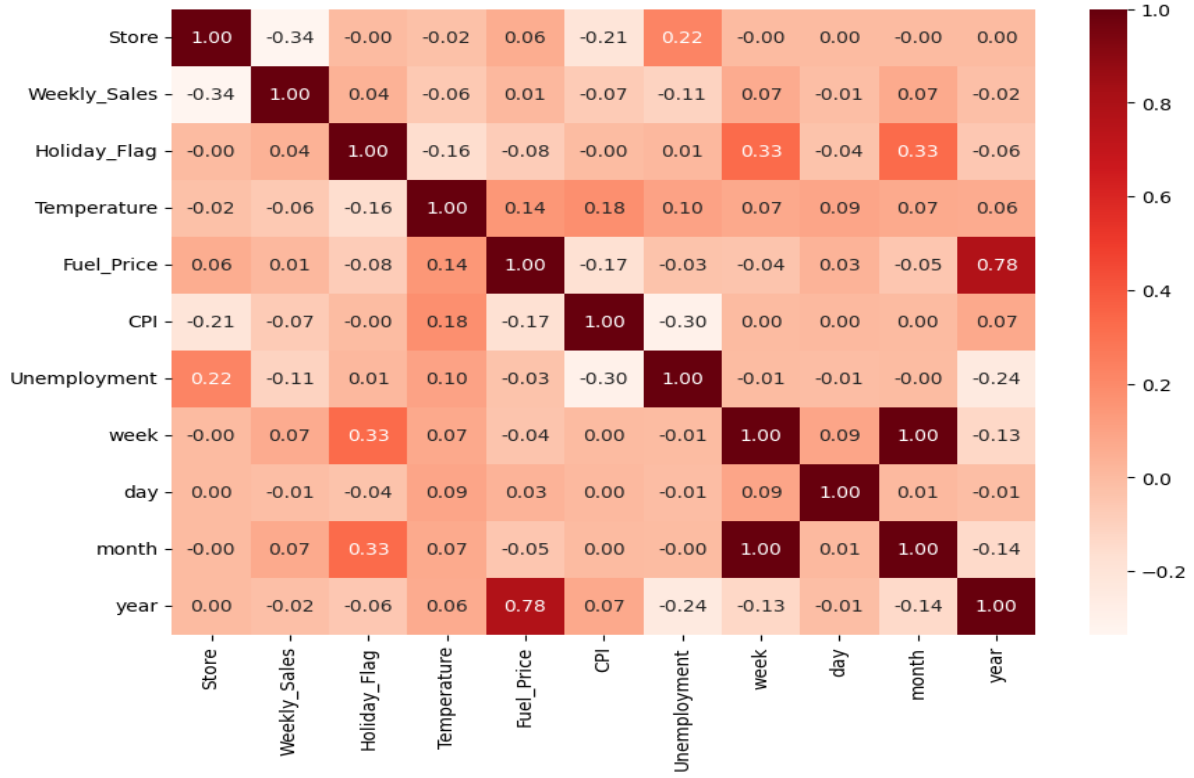
The preprocessing of the data included the following steps:

1. Loading the “Walmart” Dataset into pandas dataframe
2. Check the basic information about the dataset, such as the number of rows and columns and data types.
3. Inspect for missing data using **“isnull()”** function and take necessary action as per the following:
    a. Remove the rows with missing value if their number is insignificant
    b. Replace the missing values with the mean or median if the feature is numeric
4. Convert the 'Date' column from a string format to a datetime format using the **“pd.to_datetime()”** function.
5. Sorting the dataset in ascending order based on the 'Date' column to ensure that the data is arranged chronologically.
6. Extract additional time-related features from the 'Date' column, such as week, day, month, and year.
7. Check for and remove any duplicate rows in the dataset using the **“duplicated()”** and **“drop_duplicates()”** functions.
8. Explore the data using summary statistics, correlation matrices, and visualizations to gain insights into the data distribution and relationships between variables.
9. Since the dataset contains data of 45 different stores, a function named **'select_store()'** is defined to extract data for a specific store number. The data is sorted chronologically based on the 'Date' column to ensure time-based analysis.
10. The dataset is split into training and testing sets for model building and evaluation.

DATA EXPLORATION AND VISUALIZATION (EDA)

---

1. The correlation matrix is computed using “**data.corr()”** and visualized using a heatmap with the “**sns.heatmap()”** function. This helps identify correlations between numerical variables.

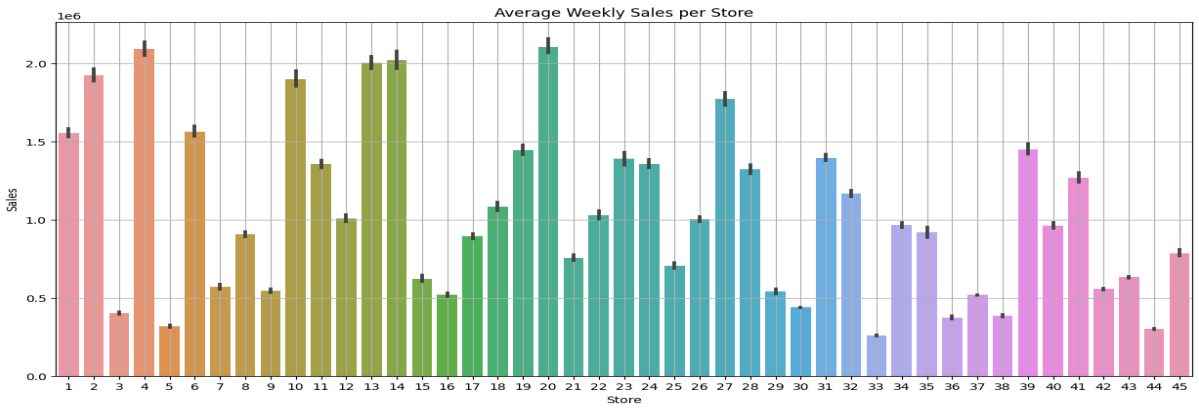


OBSERVATION:

a. Based solely on correlation, we can infer that there is a positive correlation between Fuel Price and year

b. Temperature,Fuel price, CPI and Unemployment are very weakly correlated with the weekly sales

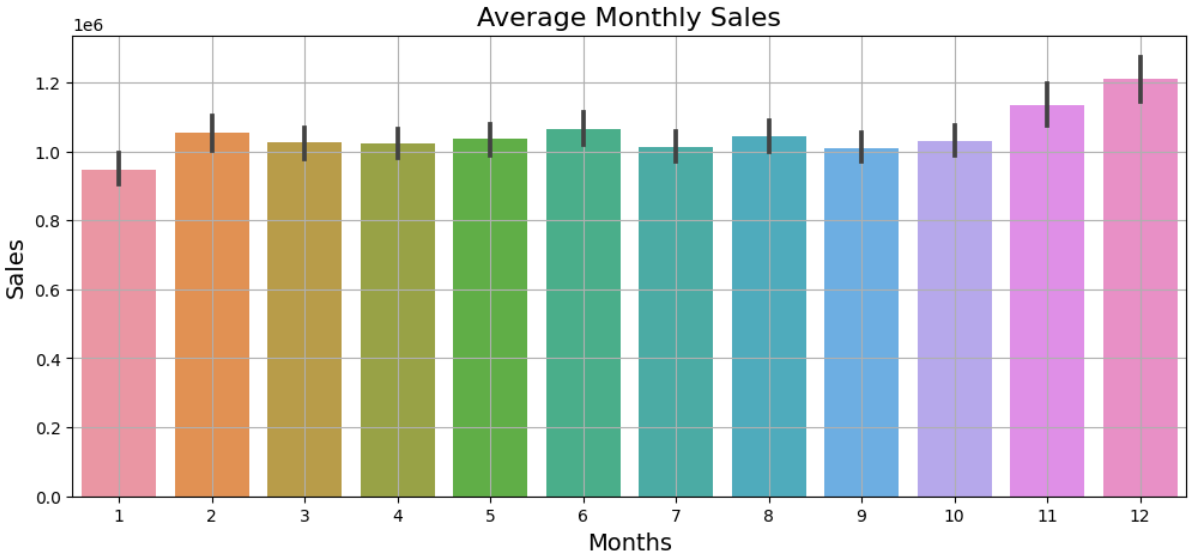
2. The average weekly sales per store are plotted using a bar plot (sns.barplot()) to visualize the variation in sales among different stores.



OBSERVATION:

a. There is a very high difference between the average sales for each of the stores.
b. store numbers 4,13,14, and 20 have the highest average sales.

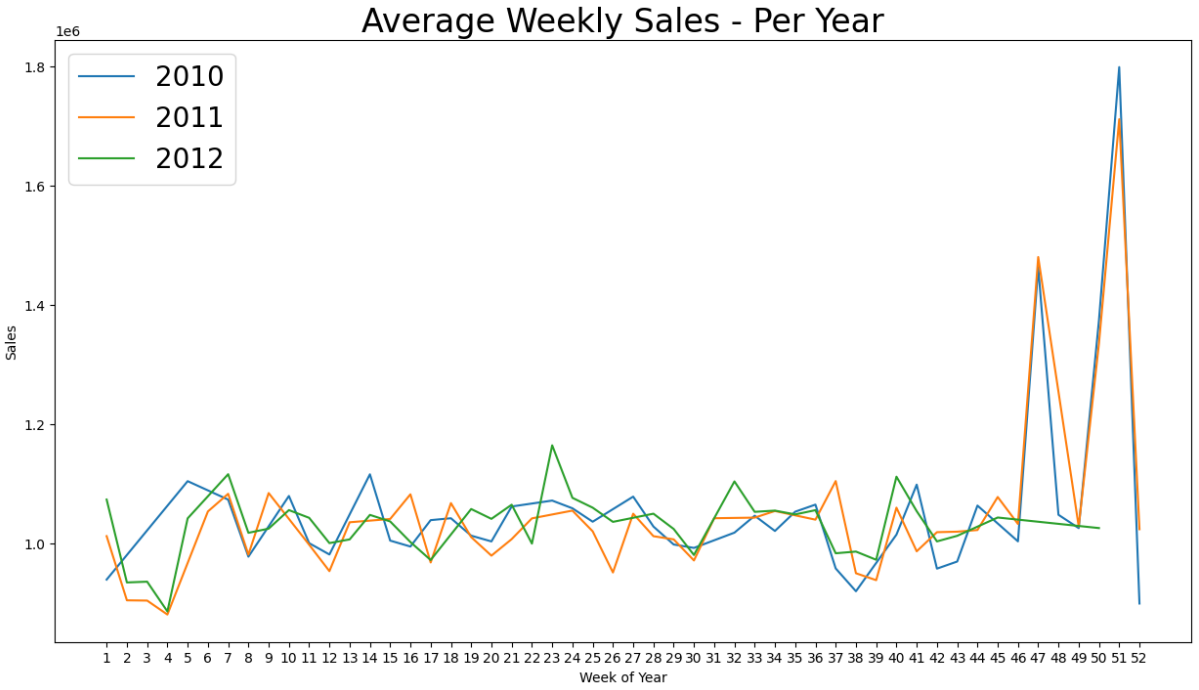
3. The average monthly sales is visualized using another bar plot, showing the variation in sales across different months.



OBSERVATION:

a. Month of January witnessed the lowest sales every year.
b. From February till October the weekly sales nearly remains constant.
c. Months of November and December recorded the highest average sales every year.

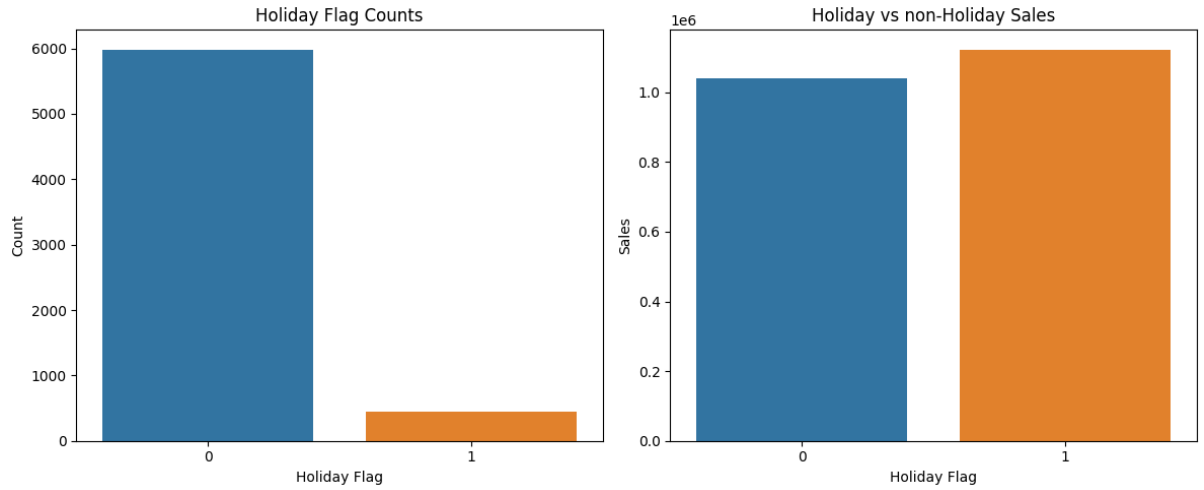
4. The average weekly sales for each year are plotted using line plots, which display the sales trends over the years.



OBSERVATION:

a. There's a clear pattern about the sales across the years, by the end of year the sales rise up by a huge margin.

5. Holiday Flag Count Plot and Holiday vs Non Holiday sales are plotted using bar plot

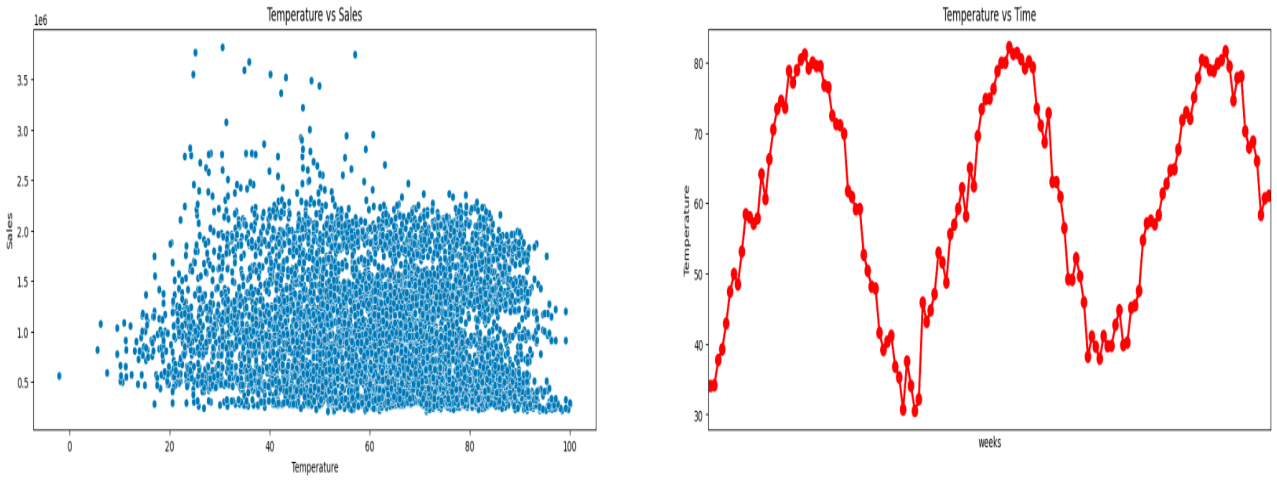


OBSERVATION:

a. Holidays significantly impact Walmart's weekly sales, with sales being considerably higher during holiday weeks compared to non-holiday weeks.

b.Despite comprising less percentage of holiday weeks the sales in the holidays week are higher than in the non-holiday weeks

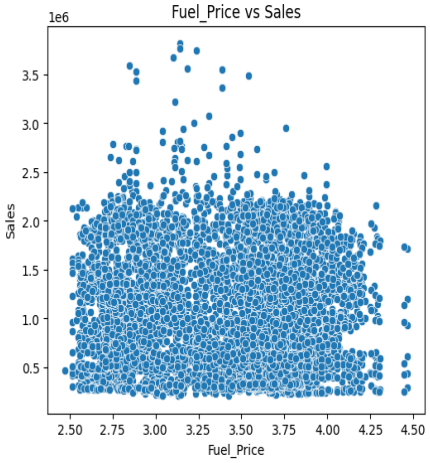
6. Scatter Plot is used to explore the relationship between Temperature and sales and a point plot is used to show the change in temperature over the weeks.

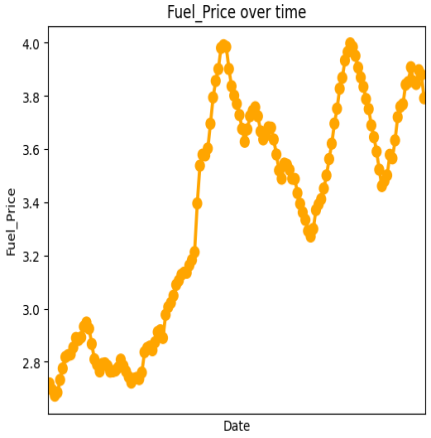


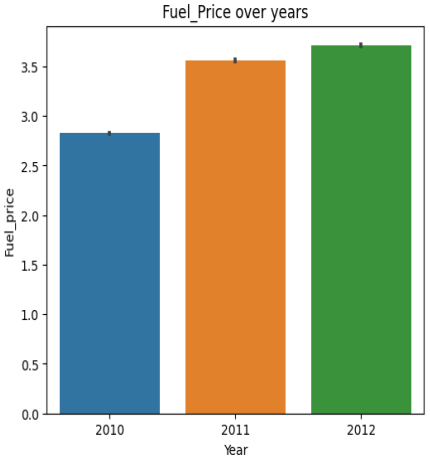
OBSERVATION:

a. Sales are relatively lower for very low and very high temperatures but seem to be adequately high for favorable climate conditions.
b. The temperature remains relatively consistent over the entire time frame, with minor fluctuations (seasonal and repeated in cycle).

7. Scatter Plot is used to explore the relationship between Fuel price and sales Also a point plot and bar plot is used to show the change in fuel price over time.



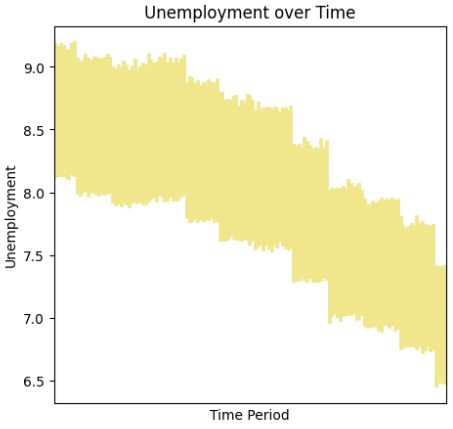




OBSERVATION:

a. While there is no definite pattern between Fuel price and sales but some observations do support the theory that lower fuel prices encourage higher sales.
b. Fuel prices are increasing with time.

8. Scatter plot is used to show relationship between Unemployment and sales also a pointplot is used to show change in unemployment over time.

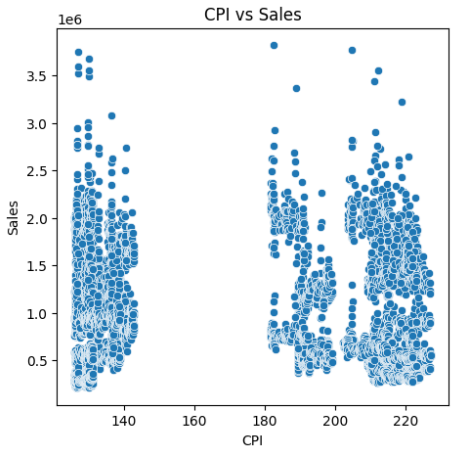


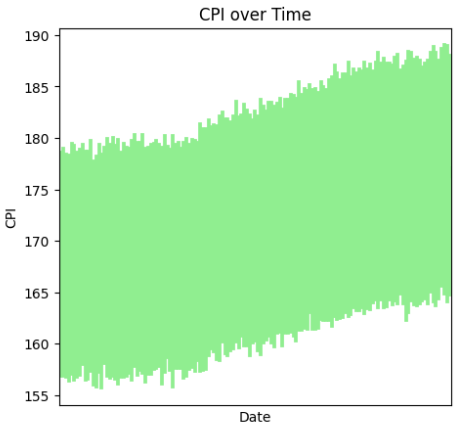
OBSERVATION:

a. There seems to be a visible decrease in sales when the unemployment index is higher than 11.

b. The point plot shows that the unemployment rate has been gradually decreasing over the analyzed time frame.

9. Scatter plot is used to show relationship between CPI and sales also a pointplot is used to show change in CPI over time.





OBSERVATION:

a. In our scatter plot above, we can identify three different clusters around different ranges of CPI, the CPI does not seem to exhibit any consistent pattern or trend that directly influences weekly sales.

b. The graph indicates that the CPI has experienced fluctuations throughout the analyzed time frame.

CHOOSING ALGORITHM FOR THE PROJECT

The choice for my algorithm for the project was dictated by the project objective which is to make a prediction model to forecast the sales for X number of months/years. As the dataset involves weekly sales data over time, which exhibits temporal patterns and dependencies, a time series model was chosen. Time series models are specifically designed to capture and exploit these temporal dependencies, making them well-suited for forecasting future sales based on historical patterns. Additionally, time series models consider seasonality and trend components, which are crucial factors in retail sales forecasting.

By using time series models like ARIMA and SARIMA, we can effectively handle the seasonality and autocorrelation present in the data. These models allow us to identify and capture periodic patterns

ASSUMPTIONS

The assumptions made for the time series modeling are as follows:

1. The residuals of the models are independent and do not exhibit any systematic patterns or correlations.
2. It is assumed that there are no significant outliers that could distort the model's predictions.
3. Since we are focused on time series modeling, which typically involves working with a single time series variable, there is no need to worry about multicollinearity for this specific analysis.
4. The sample size of the dataset is that it is large enough to provide reliable and statistically significant results.
5. **The linear relationship** - changes in the independent variable(s) have a consistent and proportional effect on the dependent variable.

MODEL BUILDING

1. Import required libraries.
2. Load the dataset.
3. Perform Data Cleaning
4. Perform EDA
5. Extract the data for specific store numbers and sort them into ascending order as the order of observation matters in time series modeling.
6. Use the **'seasonal_decompose()'** function from statsmodels library to get insights into the underlying patterns and seasonality in the data.
7. Rolling mean and rolling standard deviation are calculated and plotted using the custom made function to visualize the variations in the mean and standard deviation over time and identify any trends or patterns.
8. The Augmented Dickey-Fuller (ADF) test is conducted using a custom made **'adf_test()'** to determine the stationarity of the weekly sales data.
9. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots are used to identify the potential values of the autoregressive (AR) and moving average (MA) terms in the ARIMA and SARIMA models. OR **'auto_arima'** function can be used to find the optimum value for p,d,q.
10. Data is then split into train and test using **'train_test_split'** and the shuffle was set to 'False' to not disturb the order of observation.
11. MODEL TRAINING
    a) **<u>ARIMA Model</u>**
        - The ARIMA model is built using the 'ARIMA()' functions from the statsmodels library.
        - The model is fitted to the training data using the fit method.
        - The model is used to make predictions on the testing set using the predict method.
        - The predicted values are compared with the actual values in the testing set to evaluate the model's performance.
        - Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are calculated to measure the accuracy of the predictions.
        - A line graph was plotted between the training and testing data along with the predictions made by the ARIMA model

b) <u>SARIMA Model</u>

- The SARIMA model is built using the 'SARIMAX()' functions from the statsmodels library.
- The model is fitted to the training data using the fit method.
- The model is used to make predictions on the testing set using the predict method.
- The predicted values are compared with the actual values in the testing set to evaluate the model's performance.
- Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are calculated to measure the accuracy of the predictions.
- A line graph was plotted between the training and testing data along with the predictions made by the SARIMA model

12. Future Prediction: A SARIMA model is trained on the entire dataset, and future sales for the next 12 weeks are forecasted to predict sales for future time periods and a line graph was plotted for the given value along with predicted value

INFERENCES FROM THE PROJECT

Based on the overall analysis and modeling of the dataset, the following inferences can be drawn:

1. The ARIMA model captures the general trends and seasonality in the training data.However, the predictions seem to have some fluctuations and do not perfectly fit the test data.
2. The SARIMA model performs better than the ARIMA model as it considers the seasonality in the data.The SARIMA model's predictions show better alignment with the test data and capture the seasonal patterns more accurately.
3. Both models have been used to forecast weekly sales for the stores, and the choice between ARIMA and SARIMA depends on the specific requirements of the analysis. If seasonality is present in the data, the SARIMA model is more appropriate as it takes into account the seasonal components. On the other hand, if there is no clear seasonality in the data, the ARIMA model may be sufficient.

FUTURE POSSIBILITIES

The time series models developed for forecasting weekly sales offer several future possibilities and avenues for improvement. **Fine-tuning** the model hyperparameters can potentially enhance the forecasting accuracy.**Implementing ensemble techniques** such as combining multiple models (e.g., ARIMA, SARIMA, or other time series models) can lead to better predictions. Ensemble methods often help mitigate biases and increase overall forecasting performance.**Exploring additional features** and external factors that could impact sales, such as special events, holidays, marketing campaigns, or economic indicators, may lead to better predictions.Conducting rigorous model evaluation and validation procedures is essential. **Cross-validation techniqu**es and out-of-sample testing can help assess the models' generalization and reliability.Time series data is dynamic, and patterns may change over time. **Periodically updating** the models with new data and retraining them can ensure they stay relevant and accurate in the long run.

An important aspect of this study is also to try and understand customer buying behavior This customer segmentation can help the organization in creating and communicating targeted messages for customers belonging to a particular category,, establishing better customer relationships, focusing on profitable regions, and identifying ways to improve products and services in specific regions or for specific customers.
Another aspect that would be worth exploring with this study is identifying trends with sales for each of the stores and predicting future trends based on the available sales data.

CONCLUSION

The project provided a great opportunity to appreciate the power of time series forcasting in Machine Learning for analyzing and forecasting sequential data in various domains.The exploration and preprocessing of data provide valuable insights into underlying patterns, trends, and seasonality, enabling better decision-making and strategic planning.

Time series analysis has broad applications, including sales forecasting, economic predictions, weather forecasting, stock market analysis, and more. It provides valuable tools for understanding temporal dependencies and making predictions based on historical patterns.

When building time series models, it is essential to perform thorough evaluation and validation to ensure the accuracy and reliability of the predictions. Techniques like train-test splits, cross-validation, and evaluation metrics help assess model performance and identify areas for improvement.

Overall, time series modeling plays a crucial role in extracting valuable insights from sequential data and empowering businesses and researchers to make informed decisions. As technology and methodologies continue to evolve, time series analysis will remain a fundamental tool for understanding and forecasting temporal patterns in diverse fields.

REFERENCES

Links:

1. https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html
2. https://www.statsmodels.org/stable/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html
3. https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f
4. https://medium.com/@manuktiwary/time-series-forecasting-concepts-and-methods-with-implementation-examples-edaf40dceee5
5. Hands-On Machine Learning from Scratch: Develop a Deeper Understanding of Machine Learning Models by Implementing Them from Scratch in Python
6. "Introduction to Time Series and Forecasting" by Peter J. Brockwell and Richard A. Davis.

PYTHON CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
!pip install pmdarima
from pmdarima import auto_arima
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import *

from google.colab import drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/Colab Notebooks/Walmart.csv'
data = pd.read_csv(file_path)
```

#Data Exploration

```
data.head()

data.dtypes

data.shape

data.info()

data.describe()

#Checking NULL values
data.isnull().sum()

#Checking for duplicate values
data.duplicated().sum()

#Checking for unique values in each column
data.nunique().sort_values()

data.corr()
```

#Data Transformation

```
# Conversion of 'Date' column: The 'Date' column  from a string format
to a datetime format using the pd.to_datetime() function.
# Creating New column for week,day,month,year
```

```
# Sorting by 'Date': The datasets is sorted in ascending order based
on the 'Date' column to ensure that the data is arranged
chronologically.

#Converting Data Columnmn
data.Date=pd.to_datetime(data['Date'])

#Creating New Columns
data['week'] = data['Date'].dt.week
data['day'] = data['Date'].dt.day
data['month'] = data['Date'].dt.month
data['year'] = data['Date'].dt.year
data.head()

#Sorting the Column in ascending order based on date column
#data = data.sort_values('Date')

data.head()

data.tail()

plt.figure(figsize=(10, 7))
sns.heatmap(data.corr(),annot = True,fmt='.2f',cmap='Reds')
plt.show()

# Observations
# 1. Based solely on corelation, we can infer that their is a positive
corelation between Fuel Price and year
# 2. Temperature,Fuel price, CPI and Unemployment are very weakly
coorelated with the weekly sales

# distribution of Weekly_Sales
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='Weekly_Sales', kde=True)
plt.title('Distribution of Weekly Sales')
plt.show()
```

#EDA

```
#Average Weekly Sales Store wise
plt.figure(figsize=(15,7))
sns.barplot(x='Store',y='Weekly_Sales',data=data,estimator='mean')
plt.grid()
plt.title('Average Weekly Sales per Store', fontsize=12)
plt.ylabel('Sales')
plt.xlabel('Store')
plt.show()

# Average weekly sales of 80% of stores is below 1500000
```

```
#Average Monthly Sales
plt.figure(figsize=(12,5))
sns.barplot(x='month',y='Weekly_Sales',data=data)
plt.ylabel('Sales',fontsize=14)
plt.xlabel('Months',fontsize=14)
plt.title('Average Monthly Sales',fontsize=16)
plt.grid()
plt.show()


plt.figure(figsize=(12, 5))
sns.barplot(x='month', y='Weekly_Sales', hue='year', data=data)
plt.ylabel('Sales', fontsize=14)
plt.xlabel('Months', fontsize=14)
plt.title('Average Monthly Sales by Year', fontsize=16)
plt.grid()
plt.legend(title='Year', title_fontsize='12', fontsize='12')
plt.show()

#Month of January witnessed the lowest sales
#From Feburary till October the weekly sales nearly remains constant
#November and December showed the highest sales every year

#Average weekly salves VS year

#filters the data for each year and calculates the mean of the
'Weekly_Sales' column for each week
weekly_sales_2010 = data[data.year==2010].groupby('week')
['Weekly_Sales'].mean()
weekly_sales_2011 = data[data.year==2011].groupby('week')
['Weekly_Sales'].mean()
weekly_sales_2012 = data[data.year==2012].groupby('week')
['Weekly_Sales'].mean()

plt.figure(figsize=(15,8))
plt.plot(weekly_sales_2010.index, weekly_sales_2010.values)
plt.plot(weekly_sales_2011.index, weekly_sales_2011.values)
plt.plot(weekly_sales_2012.index, weekly_sales_2012.values)

plt.xticks(np.arange(1, 53, step=1), fontsize=10)
plt.yticks( fontsize=10)
plt.xlabel('Week of Year', fontsize=10)
plt.ylabel('Sales', fontsize=10)

plt.title("Average Weekly Sales - Per Year", fontsize=24)
plt.legend(['2010', '2011', '2012'], fontsize=20);
plt.show()

# Line plot of weekly sales over time
plt.figure(figsize=(8, 6))
sns.lineplot(data=data, x='Date', y='Weekly_Sales')
```

```
plt.title('Weekly Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.xticks(rotation=45)
plt.show()

# There's a clear pattern about the sales across the years, by the end
of year the sales rise up by a huge margin.

holiday_counts = data.Holiday_Flag.value_counts()

holiday_sales = data.groupby('Holiday_Flag')['Weekly_Sales'].mean()

plt.figure(figsize=(12, 5))

# Plot 1 - Holiday Flag Counts
plt.subplot(1, 2, 1)
sns.barplot(x=holiday_counts.index, y=holiday_counts.values)
plt.ylabel('Count')
plt.xlabel('Holiday Flag')
plt.title('Holiday Flag Counts')

# Plot 2 - Holiday vs non-Holiday Sales
plt.subplot(1, 2, 2)
sns.barplot(x=holiday_sales.index, y=holiday_sales.values)
plt.ylabel('Sales')
plt.xlabel('Holiday Flag')
plt.title('Holiday vs non-Holiday Sales')

plt.tight_layout()
plt.show()

#Despite being the less percentage of holiday weeks the sales in the
holidays week are higher than in the non-holiday weeks

#Relationship between Temperature and sales
plt.figure(figsize=(30, 5))

plt.subplot(1, 2, 1)
sns.scatterplot(x=data.Temperature, y=data.Weekly_Sales)

plt.xlabel('Temperature')
plt.ylabel('Sales')
plt.title('Temperature vs Sales')

plt.subplot(1, 2, 2)
sns.pointplot(x="Date", y="Temperature", data=data, color =
'red',linestyles='solid',errorbar=None)
plt.xlabel('weeks')
plt.ylabel('Temperature')
plt.title('Temperature vs Time')
```

```
plt.xticks([])
plt.show()

# There seems to be no relatiobship between the temperature in the
region and weekly sales of the stores.
# At low and very high temperatures the sales seems to dip a bit but
in general there doesn't exist a clear relationship
# We can clearly shows Temperature is more of a seasonal and repeated
in cycle

#Relationship between FuelPrice and sales
plt.figure(figsize=(20,5))

plt.subplot(1, 3, 1)
sns.scatterplot(x=data.Fuel_Price, y=data.Weekly_Sales);

plt.xlabel('Fuel_Price')
plt.ylabel('Sales')
plt.title('Fuel_Price vs Sales')

#Fuel Price over the time
plt.subplot(1, 3, 2)
sns.pointplot(x="Date", y="Fuel_Price", data=data,color =
'orange',linestyles='solid',errorbar=None)
plt.xlabel('Date')
plt.ylabel('Fuel_Price')
plt.title('Fuel_Price over time')
plt.xticks([])

#Fuel price over the years
plt.subplot(1, 3, 3)
sns.barplot(x=data['year'],y=data['Fuel_Price'])
plt.xlabel('Year')
plt.ylabel('Fuel_price')
plt.title('Fuel_Price over years')
plt.show()

# Between fuel price and the sales there doesn't seem to exist any
clear relationship
# As the year increases fuel prices also increases

#Relationship between CPI and sales
plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
sns.scatterplot(x=data.CPI, y=data.Weekly_Sales);

plt.xlabel('CPI')
plt.ylabel('Sales')
plt.title('CPI vs Sales')

# Change between CPI over time
```

```
plt.subplot(1, 2, 2)
sns.pointplot(x="Date", y="CPI", data=data,color='lightgreen')
plt.xlabel('Date')
plt.ylabel('CPI')
plt.title('CPI over Time')
plt.xticks([])


plt.subplots_adjust(wspace=0.9)
plt.show()

# There are 3 clear clusters but there doesn't exist any clear
correlation between CPI and weekly sales

#Relationship between Unemployment and sales

plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
sns.scatterplot(x=data.Unemployment, y=data.Weekly_Sales);

plt.xlabel('Unemployment')
plt.ylabel('Sales')
plt.title('Unemployment vs Sales')

# Change in Unemployment over time
plt.subplot(1, 2, 2)
sns.pointplot(x="Date", y="Unemployment", data=data, color='khaki')
plt.xlabel('Time Period')
plt.ylabel('Unemployment')
plt.title('Unemployment over Time')
plt.xticks([])

plt.subplots_adjust(wspace=0.9)
plt.show()

# In relation to unemployment, it can be seen that the lower the
Unemployment, higher the sales
# Unemployment has decresed over time
```

#Model

```
data.head()

#Extracting required data
data1 = data[['Store', 'Date','Weekly_Sales']]
data1.head()

#preparing data for time series model
#This extract the data for the given store number and prepares the
dataset for time series model
```

```
def select_store(data, store_number):
    # Extract data for the specified store number
    data_store = data[data['Store'] == store_number].drop('Store',
axis=1)

    # Set the 'Date' column as the index
    data_store.index = pd.to_datetime(data_store['Date'])
    del data_store['Date']

    # Sort the DataFrame based on the index (date) in ascending order
    data_store = data_store.sort_index(ascending=True)

    return data_store



# Call the function with the store number
#store_number = 1
#data_store = select_store(data1, store_number)

# Display the processed data for Store 1
#data_store
```

Store 1

```
# For Store 1
store_number = 1
data_store_1 = select_store(data1, store_number)

#from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(data_store_1.Weekly_Sales,
period=52)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(12, 10)
plt.show()

#Function to calculate and plot rolling statistics
def plot_rolling_stats(data, window_size):

    # Sort the DataFrame based on the index (date) in ascending order
    data = data.sort_index(ascending=True)

    # Calculate rolling statistics (mean and standard deviation)
    data['Rolling_Mean'] =
data['Weekly_Sales'].rolling(window=window_size).mean()
    data['Rolling_Std'] =
data['Weekly_Sales'].rolling(window=window_size).std()
```

```
    # Plot 'Weekly_Sales', rolling mean, and rolling standard
deviation
    plt.figure(figsize=(10, 6))
    plt.plot(data.index, data['Weekly_Sales'], label='Weekly Sales')
    plt.plot(data.index, data['Rolling_Mean'], label='Rolling Mean',
linestyle='--')
    plt.plot(data.index, data['Rolling_Std'], label='Rolling Std',
linestyle='-.')
    plt.xlabel('Date')
    plt.ylabel('Weekly Sales')
    plt.title('Weekly Sales with Rolling Mean and Rolling Std')
    plt.legend()
    plt.grid(True)
    plt.show()


plot_rolling_stats(data_store_1, 4)

#Function to check Checking the Stationarity of data
#ADF test
def adf_test(dataset):
     dftest = adfuller(dataset, autolag = 'AIC')
     print("1. ADF : ",dftest[0])
     print("2. P-Value : ", dftest[1])
     print("3. Num Of Lags : ", dftest[2])
     print("4. Num Of Observations Used For ADF Regression:",
dftest[3])
     print("5. Critical Values :")
     for key, val in dftest[4].items():
         print("\t",key, ": ", val)
     if dftest[1] <= 0.05:
      print("strong evidence against the null hypothesis, reject the
null hypothesis. Data has no unit root and is stationary")
     else:
      print("weak evidence against null hypothesis, time series has a
unit root, indicating it is non-stationary ")


#If p<0.05 ; Data is stationary
#if p>0.05; Data is not stationary

adf_test(data_store_1['Weekly_Sales'])

#ACF & PACF (to find p,d,q)
#from statsmodels.graphics.tsaplots import plot_acf,plot_pacf

# Plot ACF
plt.figure(figsize=(10, 5))
plot_acf(data_store_1)
plt.title('Autocorrelation Function (ACF)')
```

```
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.show()

# Plot PACF
plt.figure(figsize=(10, 5))
plot_pacf(data_store_1)
plt.title('Partial Autocorrelation Function (PACF)')
plt.xlabel('Lags')
plt.ylabel('Partial Autocorrelation')
plt.show()

# we can use this module to get the optimumn value for p,d,q

#from pmdarima import auto_arima

order1 = auto_arima(data_store_1['Weekly_Sales'], trace=True)
order1.summary()

# from the output we can see that the optimum value for (p,d,q) is
(0,1,2)

# Splitting the time series into train and test sets

#from sklearn.model_selection import train_test_split
train1,test1 = train_test_split(data_store_1,test_size =
0.10,shuffle=False)

# Plotting both graphs in one figure with different colors
plt.figure(figsize=(20, 6))
plt.plot(train1.index, train1, label='train', color='blue')
plt.plot(test1.index, test1, label='test', color='red')

plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Train and Test Time Series')
plt.legend()
plt.show()
```

ARIMA Model

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

model_Store1_1=ARIMA(train1['Weekly_Sales'],order=(0,1,2))
model_Store1_1_fit=model_Store1_1.fit()
#model_Store1_1_fit.summary()

prediction_Store1_1 =
model_Store1_1_fit.predict(start=len(train1),end=(len(data_store_1)-
1))
```

```
test1_index = test1.index
prediction_Store1_1.index = test1_index

plt.figure(figsize=(20, 6))
plt.plot(train1.index, train1, label='train', color='blue')
plt.plot(test1.index, test1, label='test', color='orange')
plt.plot(prediction_Store1_1.index, prediction_Store1_1, label='test',
color='red')

prediction_Store1_1
prediction_df =
prediction_Store1_1.to_frame(name='Predicted_Weekly_Sales')
prediction_df['Predicted_Weekly_Sales'] =
prediction_df['Predicted_Weekly_Sales'].astype(int)
prediction_df

mae = mean_absolute_error(test1['Weekly_Sales'],
prediction_df['Predicted_Weekly_Sales'])
print("ARIMA Model - MAE:",mae)
mse = mean_squared_error(test1['Weekly_Sales'],
prediction_df['Predicted_Weekly_Sales'])
print("ARIMA Model - MSE:", mse)
rmse = rmse = np.sqrt(mse)
print("ARIMA Model - RMSE:", rmse)
```

SARIMA Model

```
#from statsmodels.tsa.statespace.sarimax import SARIMAX
model_Store1_2 =
SARIMAX(train1['Weekly_Sales'],order=(0,1,2),seasonal_order=(0,1,2,52)
)
model_Store1_2_fit=model_Store1_2.fit()
#model_Store1_2_fit.summary()

prediction_Store1_2 =
model_Store1_2_fit.predict(start=len(train1),end=(len(data_store_1)-
1))

test1_index = test1.index
prediction_Store1_2.index = test1_index

plt.figure(figsize=(20, 6))
plt.plot(train1.index, train1, label='train', color='blue')
plt.plot(test1.index, test1, label='test', color='orange')
plt.plot(prediction_Store1_2.index, prediction_Store1_2, label='test',
color='red')
```

Future Prediction

```
#Forcasting values for next 12 weeks

# Create the SARIMA model and fit it to the entire dataset.
model1_f = SARIMAX(data_store_1['Weekly_Sales'], order=(0, 1, 2),
seasonal_order=(0, 1, 2, 52))
model1_f_fit = model1_f.fit()

# Predict data for the next 12 weeks.
predictionf_1 = model1_f_fit.predict(start=len(data_store_1),
end=len(data_store_1) + 11)

# Create a date range for the next 12 weeks (assuming 'data_store_1'
has a DatetimeIndex).
next_12_weeks = pd.date_range(start=data_store_1.index[-1],
periods=12, freq='W')

# Assign the index to the predictions for plotting.
predictionf_1.index = next_12_weeks

# Convert the prediction Series to a DataFrame.
predictionf_1_df =
predictionf_1.to_frame(name='Predicted_Weekly_Sales')
predictionf_1_df['Predicted_Weekly_Sales'] =
predictionf_1_df['Predicted_Weekly_Sales'].astype(int)

# Merge the predicted DataFrame with the original data_store_1
DataFrame.
data_with_predictions_1 = pd.concat([data_store_1, predictionf_1_df])

# Plot the graph.
plt.figure(figsize=(10, 6))
plt.plot(data_with_predictions_1.index,
data_with_predictions_1['Weekly_Sales'], label='Actual')
plt.plot(data_with_predictions_1.index,
data_with_predictions_1['Predicted_Weekly_Sales'], label='Predicted',
color='red')
plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.title('Weekly Sales Prediction for the Next 12 Weeks of Store 1')
plt.legend()
plt.grid(True)
plt.show()

#Function to create 1) ARIMA Model
#                   2) SARIMA Model
#                   3) Future Forecast
#and plot their graphs

def arima_forecast(train, test, order):
```

```
  with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    # Fit the ARIMA model
    model = ARIMA(train['Weekly_Sales'], order=order)
    model_fit = model.fit()

    # Make predictions
    predictions = model_fit.predict(start=len(train), end=(len(train)
+ len(test) - 1))

    # Set prediction index to match test index
    predictions.index = test.index

    # Plot the results
    plt.figure(figsize=(20, 6))
    plt.plot(train.index, train['Weekly_Sales'], label='train',
color='blue')
    plt.plot(test.index, test['Weekly_Sales'], label='test',
color='orange')
    plt.plot(predictions.index, predictions, label='predictions',
color='red')
    plt.title('Weekly Sales Prediction using ARIMA Model')
    plt.xlabel('Date')
    plt.ylabel('Weekly Sales')
    plt.legend()
    plt.show()

    return predictions

def sarima_forecast(train, test, seasonal_order,order):
  with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    # Fit the SARIMA model
    model = SARIMAX(train['Weekly_Sales'], order=order,
seasonal_order=seasonal_order)
    model_fit = model.fit()

    # Make predictions
    predictions = model_fit.predict(start=len(train), end=(len(train)
+ len(test) - 1))

    # Set prediction index to match test index
    predictions.index = test.index

    # Plot the results
    plt.figure(figsize=(20, 6))
    plt.plot(train.index, train['Weekly_Sales'], label='train',
color='blue')
    plt.plot(test.index, test['Weekly_Sales'], label='test',
```

```
color='orange')
    plt.plot(predictions.index, predictions, label='predictions',
color='red')
    plt.title('Weekly Sales Prediction using SARIMA Model')
    plt.xlabel('Date')
    plt.ylabel('Weekly Sales')
    plt.legend()
    plt.show()


    return predictions

def future_forecast(data, order, seasonal_order):
  with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    # Create the SARIMA model and fit it to the entire dataset.
    model = SARIMAX(data['Weekly_Sales'], order=order,
seasonal_order=seasonal_order)
    model_fit = model.fit()

    # Predict data for the next 12 weeks.
    prediction = model_fit.predict(start=len(data), end=len(data) +
11)

    # Create a date range for the next 12 weeks (assuming 'data' has a
DatetimeIndex).
    next_12_weeks = pd.date_range(start=data.index[-1], periods=12,
freq='W')

    # Assign the index to the predictions for plotting.
    prediction.index = next_12_weeks

    # Convert the prediction Series to a DataFrame.
    prediction_df = prediction.to_frame(name='Predicted_Weekly_Sales')
    prediction_df['Predicted_Weekly_Sales'] =
prediction_df['Predicted_Weekly_Sales'].astype(int)

    # Merge the predicted DataFrame with the original data DataFrame.
    data_with_predictions = pd.concat([data, prediction_df])

    # Plot the graph.
    plt.figure(figsize=(20, 6))
    plt.plot(data_with_predictions.index,
data_with_predictions['Weekly_Sales'], label='Actual',color='blue')
    plt.plot(data_with_predictions.index,
data_with_predictions['Predicted_Weekly_Sales'], label='Predicted',
color='red')
    plt.xlabel('Date')
    plt.ylabel('Weekly Sales')
    plt.title('Weekly Sales Prediction for the Next 12 Weeks')
```

```
    plt.legend()

    plt.show()

    return data_with_predictions


# For Store 25
store_number = 25
data_store_25 = select_store(data1, store_number)
train25,test25 = train_test_split(data_store_25,test_size =
0.10,shuffle=False)

train = train25
test = test25
order = (0, 1, 2)
seasonal_order = (0,1,2,52)
data = data_store_25
arima_forecast(train, test, order)
sarima_forecast(train, test, seasonal_order,order)
future_forecast(data, order, seasonal_order)

# For Store 35
store_number = 35
data_store_35 = select_store(data1, store_number)
train35,test35 = train_test_split(data_store_35,test_size =
0.10,shuffle=False)

train = train35
test = test35
order = (0, 1, 2)
seasonal_order = (0,1,2,52)
data = data_store_35
arima_forecast(train, test, order)
sarima_forecast(train, test, seasonal_order,order)
future_forecast(data, order, seasonal_order)
```