Capstone Project Report

Problem Statement

The problem statement revolves around addressing the inventory management challenges encountered by Walmart, a nationwide retail giant with an extensive network of stores. The primary issue at hand is the effective synchronization of inventory levels with the fluctuating consumer demand. To tackle this predicament, the project aims to develop a robust time series model that can accurately forecast future sales. By leveraging historical sales data and capturing seasonal trends, this model will empower Walmart to proactively anticipate fluctuations in consumer demand, allowing the company to optimize inventory management and ensure that supply aligns seamlessly with the ever-changing market dynamics.

Project Objective

The primary objective of this project is to construct a robust time series forecasting model with the capability to provide precise predictions of future sales for Walmart. This endeavor involves harnessing the wealth of historical sales data at Walmart's disposal and, crucially, discerning and incorporating the recurrent seasonal patterns and trends within this data.

By doing so, this advanced predictive model will grant Walmart the ability to take a proactive stance in foreseeing and adapting to variations in consumer demand. This proactive approach is poised to revolutionize the company's inventory management strategies. By staying ahead of the curve and leveraging the insights provided by the model, Walmart will be better equipped to fine-tune its inventory levels in alignment with the seasonal ebb and flow of customer preferences and market dynamics. This harmonization will, in turn, ensure that the supply chain operates seamlessly, effectively mitigating issues related to overstocking or understocking of products, which can have a substantial impact on Walmart's operational efficiency and bottom line. In essence, this project's success promises to empower Walmart with the strategic foresight required to optimize its inventory management and maintain a competitive edge in the dynamic retail landscape.

Data Description

The "walmart.csv" dataset is a comprehensive collection of sales-related information from Walmart, consisting of 6435 rows and 8 columns. Each row in the dataset represents a specific record, while the columns provide detailed information about various aspects of Walmart's operations.

1. **Store (Numerical):** This column indicates the unique store number to which each record corresponds. Walmart operates multiple stores, and this identifier allows us to associate each data point with a specific retail outlet.

2. **Date (Date):** The "Date" column specifies the week of sales for each record. It represents the time period during which the sales data was recorded.

3. **Weekly_Sales (Numerical)**: The "Weekly_Sales" column denotes the total sales for a given store during the specific week mentioned in the "Date" column. This figure represents the financial performance of each store for that particular week.

4. **Holiday_Flag (Binary)**: The "Holiday_Flag" column is a binary indicator that signals whether the week being reported includes a holiday. A value of 1 typically indicates that it is a holiday week, while 0 denotes a regular week without any special holiday events.

5. Temperature (Numerical): This column provides the temperature on the day of the sale. Temperature can influence consumer behavior, as weather conditions often impact shopping patterns.

6. **Fuel_Price (Numerical)**: The "Fuel_Price" column specifies the cost of fuel in the region where the store is located. Fuel prices can have a significant effect on both consumer spending and operational costs for retailers like Walmart.

7. **CPI (Numerical):** The "CPI" column stands for Consumer Price Index, which is a measure of the average change over time in the prices paid by urban consumers for a market basket of consumer goods and services. CPI can serve as an indicator of inflation and can influence purchasing power.

8. **Unemployment (Numerical):** This column represents the unemployment rate for the region where the store operates. Unemployment rates can affect consumer confidence and spending habits.

The dataset appears to capture essential sales-related information for Walmart stores over a period of time. It enables various analyses and modeling exercises, such as sales forecasting, identifying the impact of holidays on sales, examining the relationship between sales and external factors like temperature, fuel prices, CPI, and unemployment.

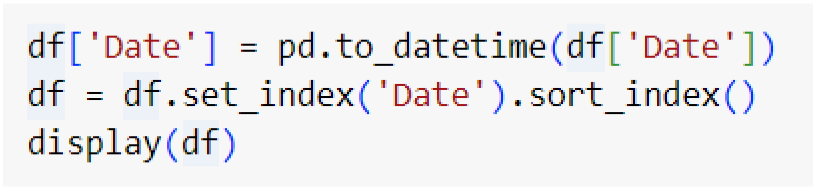
This dataset is valuable for both descriptive and predictive analytics, as it allows for insights into historical sales trends and the factors that influence Walmart's sales performance across different stores and time periods. It can be used to develop models and strategies to optimize inventory management, marketing campaigns, and operational decisions for individual stores and the company as a whole.

‘

Data Preprocessing

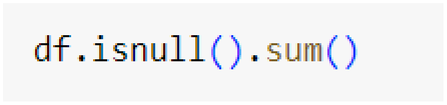
Before conducting any analysis or building predictive models, it is essential to prepare the dataset through a series of data preprocessing steps. These steps are designed to ensure that the data is clean, organized, and suitable for further investigation. In this section, we will outline the key data preprocessing steps undertaken to refine the "walmart.csv" dataset for analysis.

1. **Convert Date column to Datetime :**

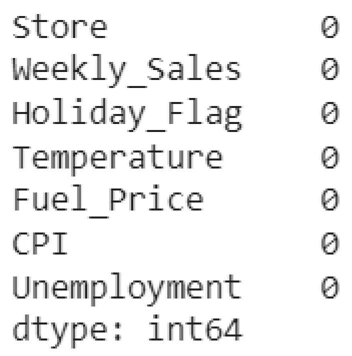


```
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date').sort_index()
display(df)
```

2. **Checking for Null Values :**



```
df.isnull().sum()
```

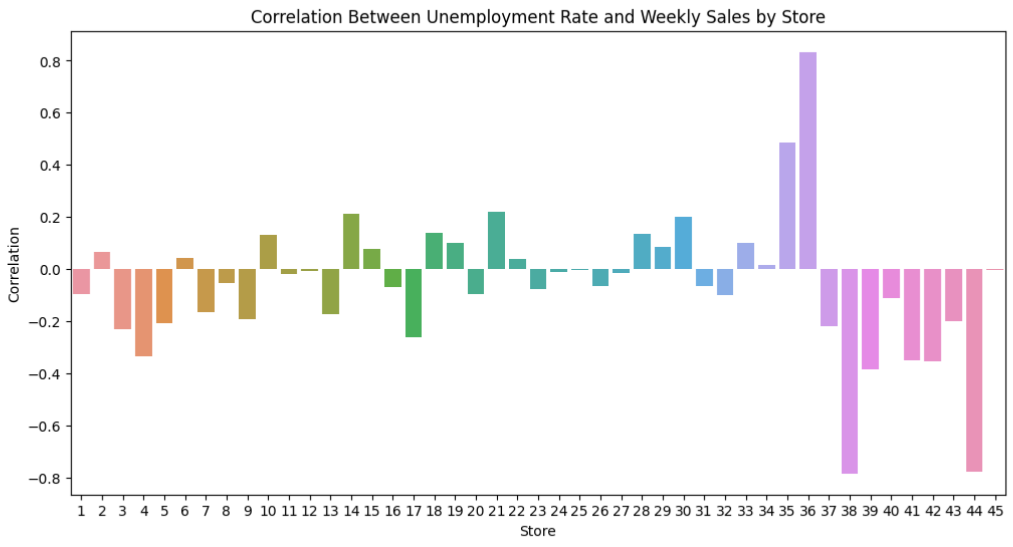


```
Store           0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

Inferring Insights

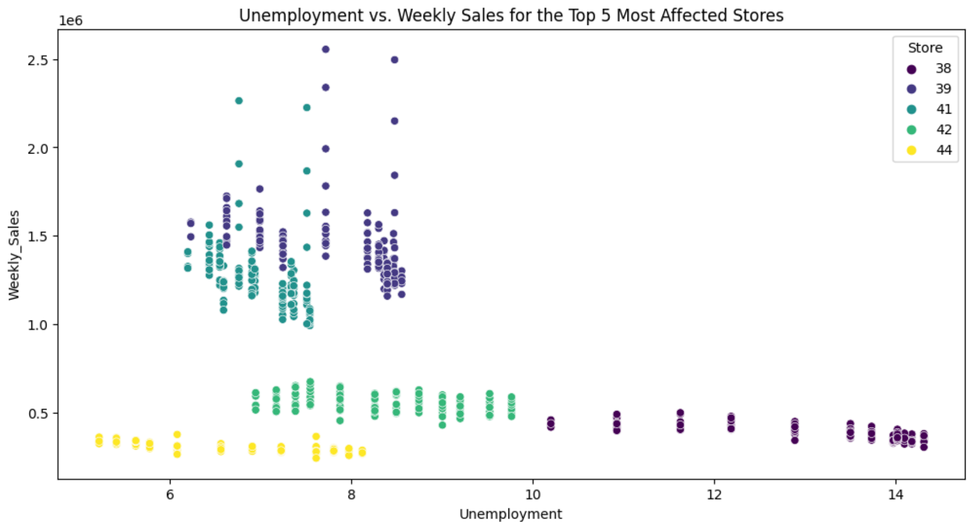
1. **If the weekly sales are affected by the unemployment rate, if yes - which stores are suffering the most :**

We will first plot the correlation between unemployment rate and Weekly sales to see how they are correlated on the basis of stores



We can see that almost equal number of stores have positive and negative correlation with weekly sales . Although the number of stores having high negative correlation is significantly more than those with high positive correlation .

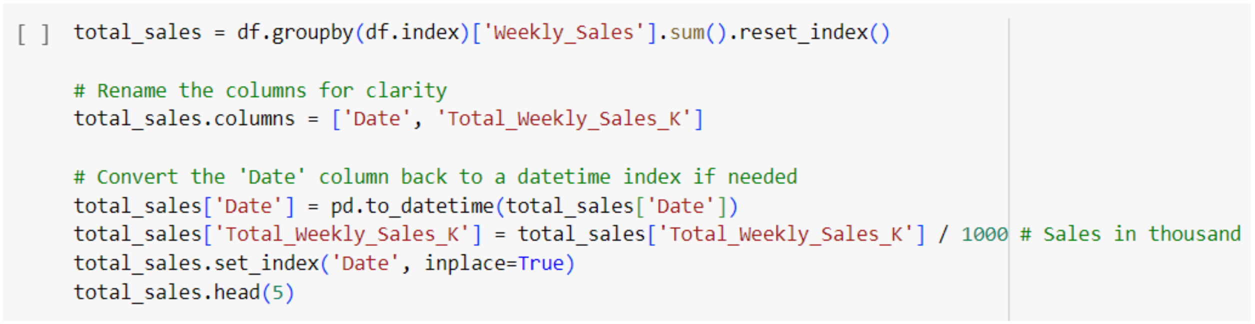
Let us now Visualize the weekly sales with unemployment store for the top 5 negatively correlated stores using a Scatter Plot



We can clearly see that lesser the unemployment , higher the weekly sales . From this , we can infer that stores which are located in areas having a good chunk of population working , will have more sales than the other ones because the working customers can obviously afford more .

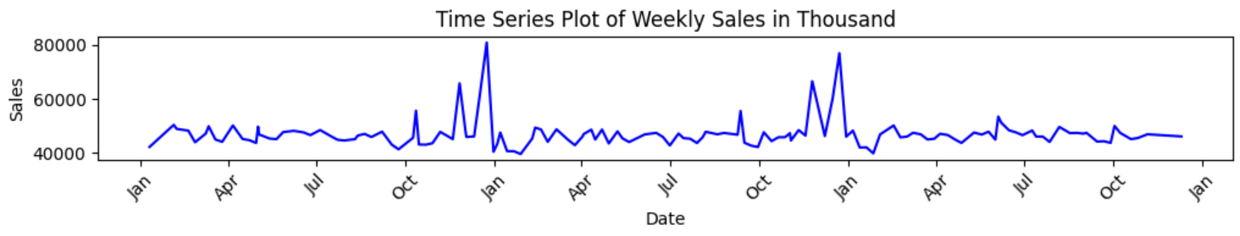
2. Does Weekly Sales shows Seasonal Trends :

To see if the weekly sales show any seasonal trend , we will first group the weekly sales by date , merging all the stores because our priority is to see if the overall sales vary with seasons irrespective of store number .



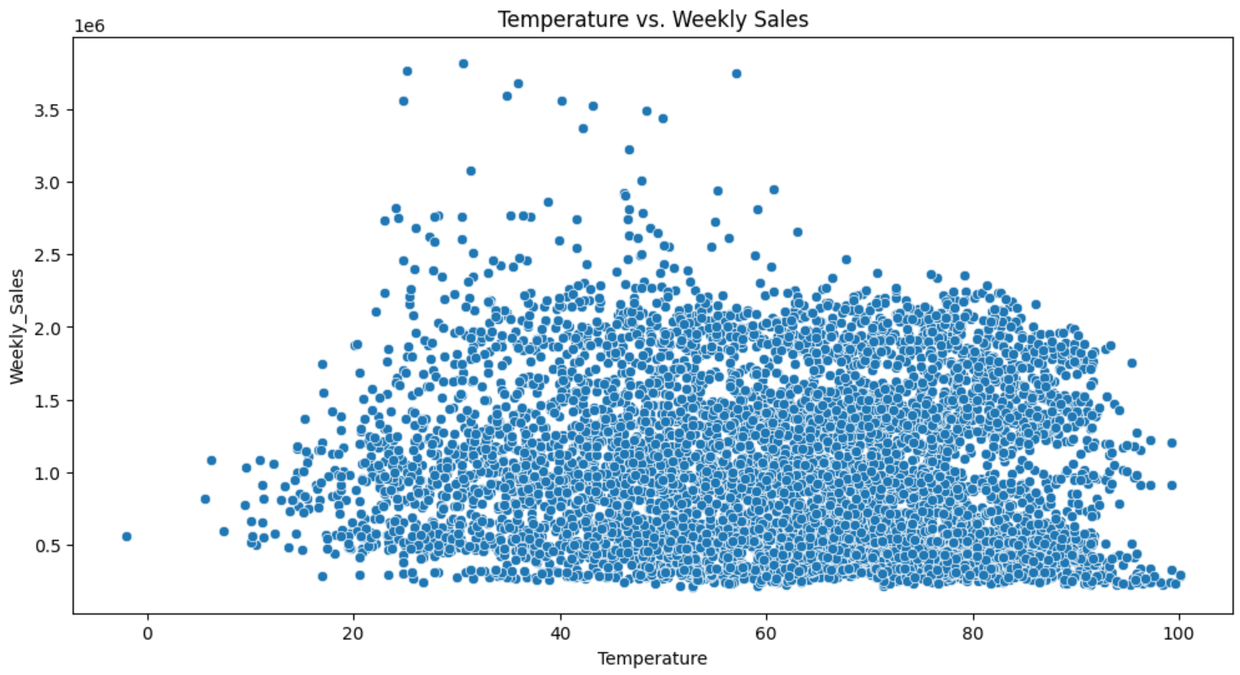
```
[ ]  total_sales = df.groupby(df.index)['Weekly_Sales'].sum().reset_index()

     # Rename the columns for clarity
     total_sales.columns = ['Date', 'Total_Weekly_Sales_K']

     # Convert the 'Date' column back to a datetime index if needed
     total_sales['Date'] = pd.to_datetime(total_sales['Date'])
     total_sales['Total_Weekly_Sales_K'] = total_sales['Total_Weekly_Sales_K'] / 1000 # Sales in thousand
     total_sales.set_index('Date', inplace=True)
     total_sales.head(5)
```

Now we will plot a Time Series graph to see if the seasons have an impact on weekly sales .



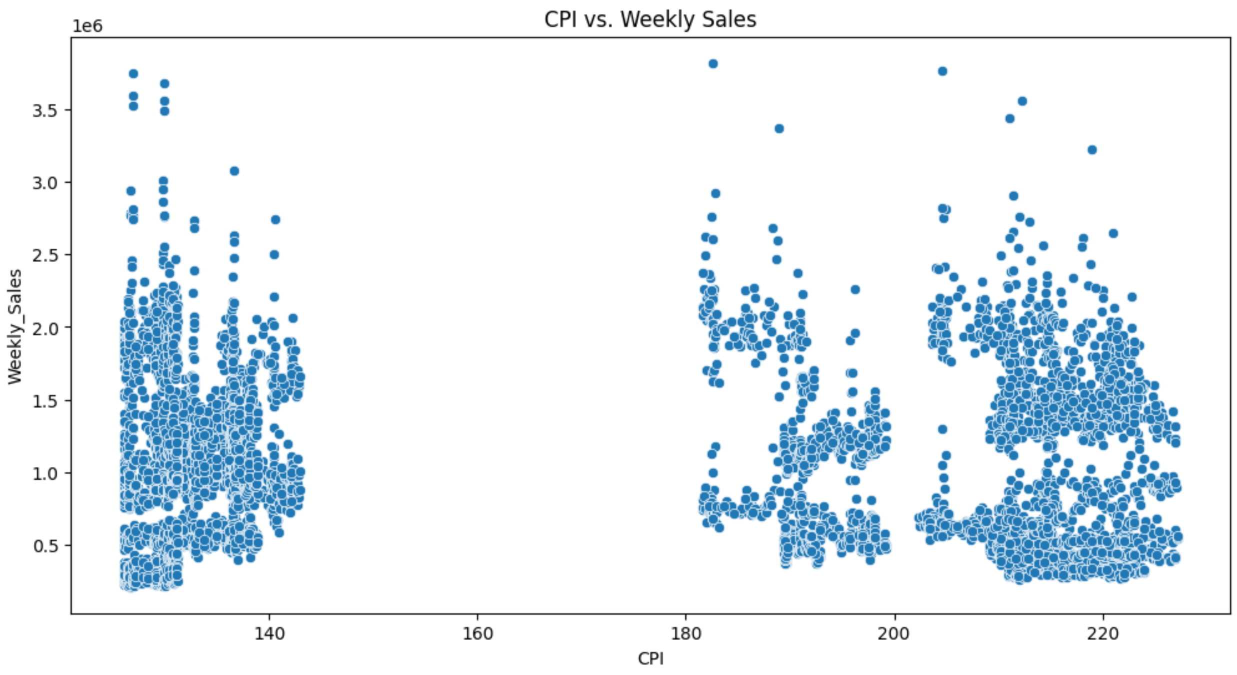
We can see some interesting trends here . The weekly sales are stable for most of the year , but significantly rise to very high in winters especially during the months of November , December and January . This can help the company stay prepared for the peak time of the year i.e winters .

3. Does Temperature affects Weekly Sales :



Although we say that sales are quite high in winters , temperature as such does not have a very direct relation with sales . Sales are almost uniformly scattered but still they are bit on the high side when the temperature is low ie. winters where some weekly sales go up to three times than the usual .

4. How CPI affects Weekly Sales :



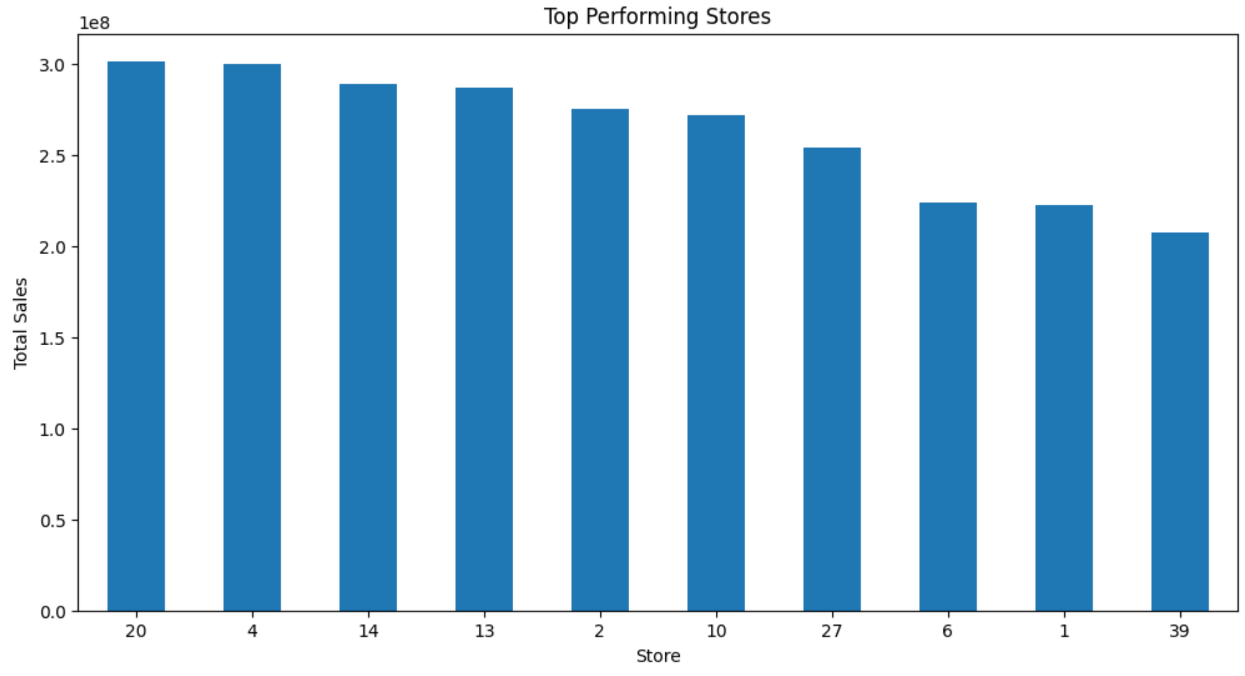
CPI does not show any noticeable patterns with respect to weekly sales . It’s only scattered around extremes and that too almost to the same extent . So we can say that CPI does not matter much with respect to the weekly sales .

5. Which are the Top Performing Stores :

Top 10 performing stores are :

```
[ ]  # Top performing stores
     top_stores = df.groupby('Store')['Weekly_Sales'].sum().sort_values(ascending=False).head(10)
     print("Top Performing Stores:")
     print(top_stores)
```

```
Top Performing Stores:
Store
20    3.013978e+08
4     2.995440e+08
14    2.889999e+08
13    2.865177e+08
2     2.753824e+08
10    2.716177e+08
27    2.538559e+08
6     2.237561e+08
1     2.224028e+08
39    2.074455e+08
Name: Weekly_Sales, dtype: float64
```

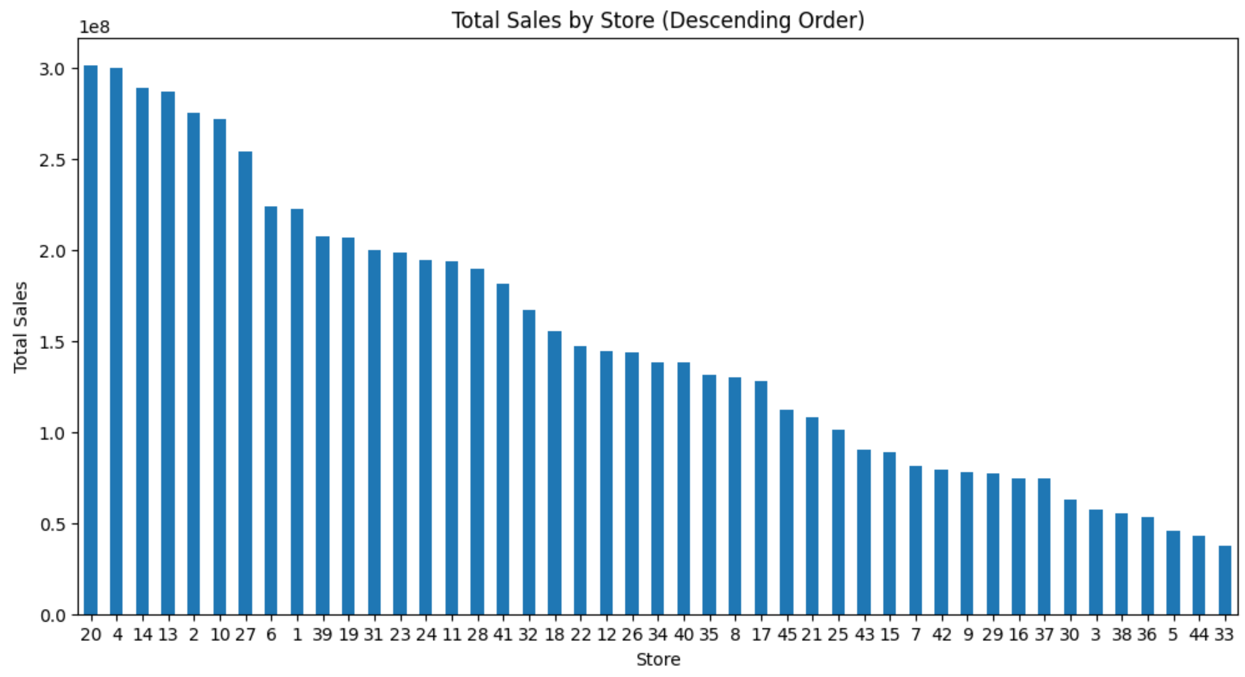


We can see that Store number 20 has the highest weekly rise much higher than the other ones . So the managers of this store should be more prepared for the peak seasons . Also the other 9 stores have almost similar sales so they should be given equal importance .

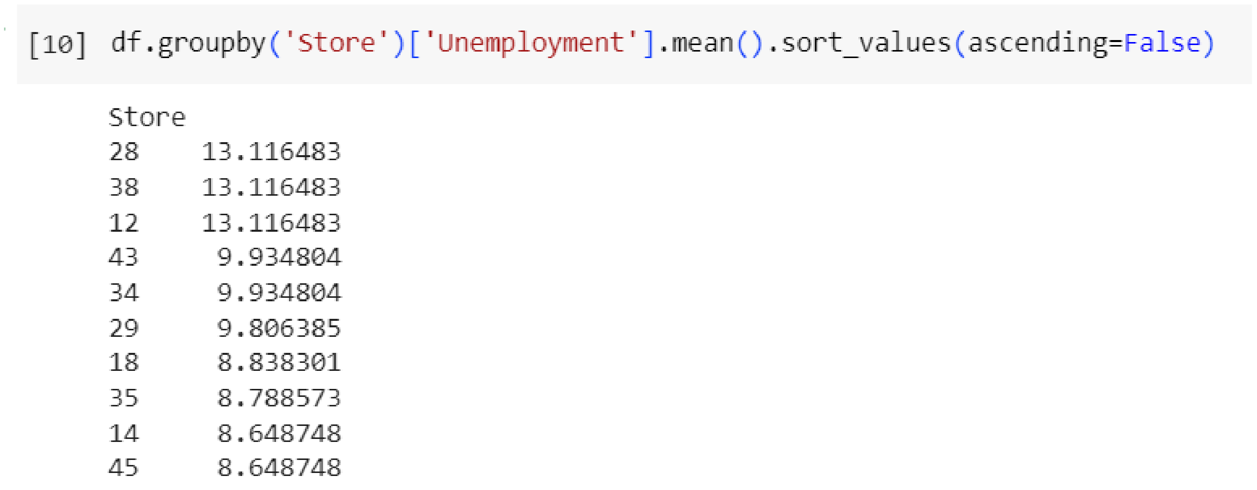
6. **Which is the Worst Performing Stores :**

```
[ ]  # Worst performing store
     worst_store = df.groupby('Store')['Weekly_Sales'].sum().idxmin()
     print(f"Worst Performing Store: Store {worst_store}")
```

```
Worst Performing Store: Store 33
```



These are some of the worst performing stores . That does not mean they should be neglected, instead an extra care should be given to these stores to make them match with the well performing ones . If I think of any potential reasons , there could be more unemployment in these areas . Let’s check :

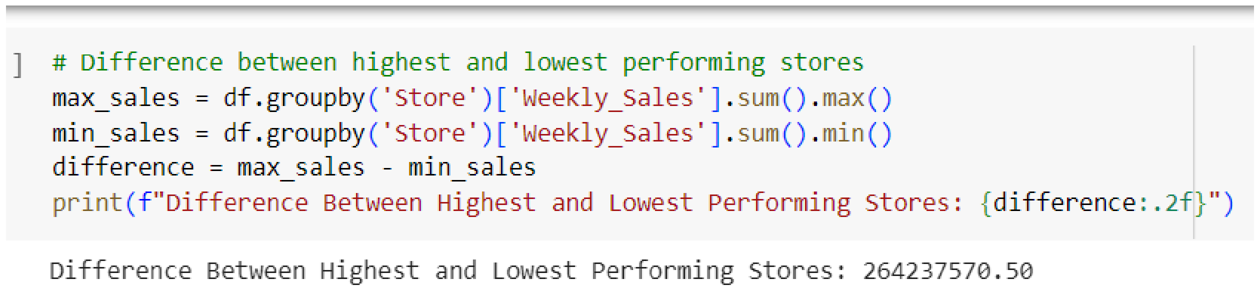


```
[10] df.groupby('Store')['Unemployment'].mean().sort_values(ascending=False)
```

```
Store
28    13.116483
38    13.116483
12    13.116483
43     9.934804
34     9.934804
29     9.806385
18     8.838301
35     8.788573
14     8.648748
45     8.648748
```

We can see that all the poorly performing stores have a high unemployment rate .

To cope up with those , discounts and offers should be runned in these ones .



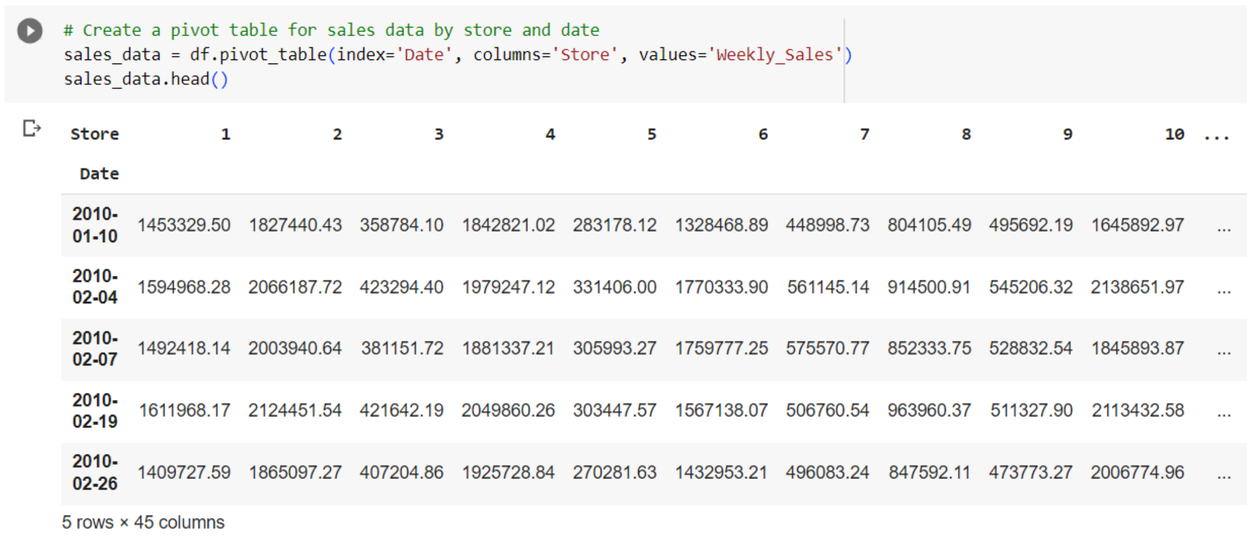
```
] # Difference between highest and lowest performing stores
  max_sales = df.groupby('Store')['Weekly_Sales'].sum().max()
  min_sales = df.groupby('Store')['Weekly_Sales'].sum().min()
  difference = max_sales - min_sales
  print(f"Difference Between Highest and Lowest Performing Stores: {difference:.2f}")
```

```
Difference Between Highest and Lowest Performing Stores: 264237570.50
```

Choosing the Algorithm for the Project

We will first split the data into training and testing in an 80:20 ratio and then we will build 3 different models and see which one performs best . The best model would then be used for forecasting the next 12 weeks weekly sales .

Let us first pivot our dataset so that every store becomes a column .

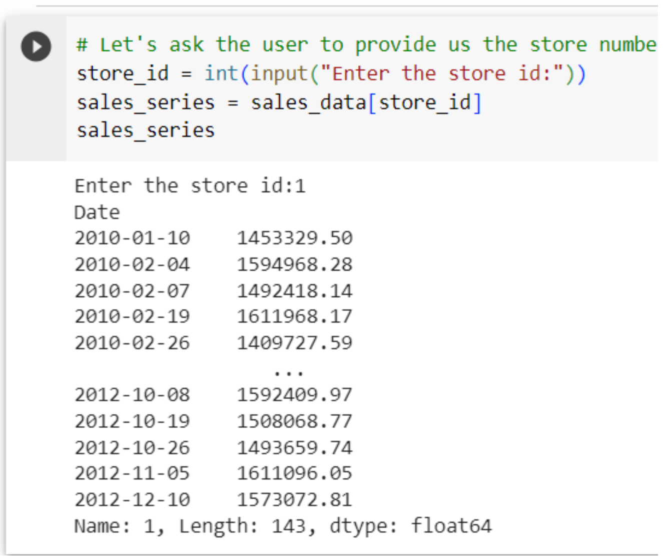


```
# Create a pivot table for sales data by store and date
sales_data = df.pivot_table(index='Date', columns='Store', values='Weekly_Sales')
sales_data.head()
```

Store Date	1	2	3	4	5	6	7	8	9	10	...
2010-01-10	1453329.50	1827440.43	358784.10	1842821.02	283178.12	1328468.89	448998.73	804105.49	495692.19	1645892.97	...
2010-02-04	1594968.28	2066187.72	423294.40	1979247.12	331406.00	1770333.90	561145.14	914500.91	545206.32	2138651.97	...
2010-02-07	1492418.14	2003940.64	381151.72	1881337.21	305993.27	1759777.25	575570.77	852333.75	528832.54	1845893.87	...
2010-02-19	1611968.17	2124451.54	421642.19	2049860.26	303447.57	1567138.07	506760.54	963960.37	511327.90	2113432.58	...
2010-02-26	1409727.59	1865097.27	407204.86	1925728.84	270281.63	1432953.21	496083.24	847592.11	473773.27	2006774.96	...

5 rows × 45 columns

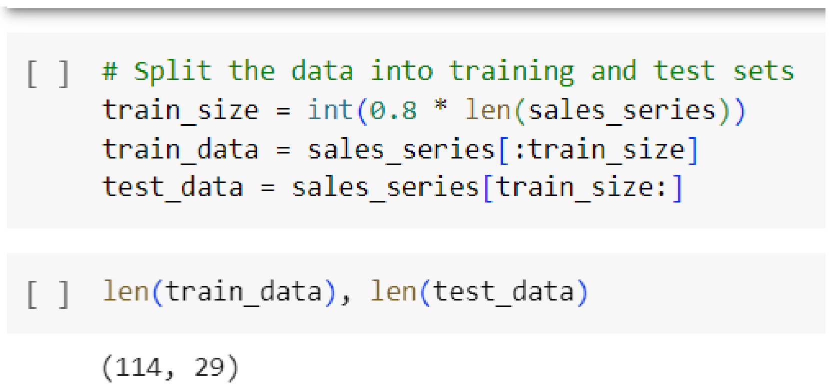
Since we are focusing on one particular store at a time . Let us ask the user to enter the store number for which he wants to predict .



```
# Let's ask the user to provide us the store numbe
store_id = int(input("Enter the store id:"))
sales_series = sales_data[store_id]
sales_series
```

```
Enter the store id:1
Date
2010-01-10    1453329.50
2010-02-04    1594968.28
2010-02-07    1492418.14
2010-02-19    1611968.17
2010-02-26    1409727.59
                 ...
2012-10-08    1592409.97
2012-10-19    1508068.77
2012-10-26    1493659.74
2012-11-05    1611096.05
2012-12-10    1573072.81
Name: 1, Length: 143, dtype: float64
```

Let us now split the data into train and test .

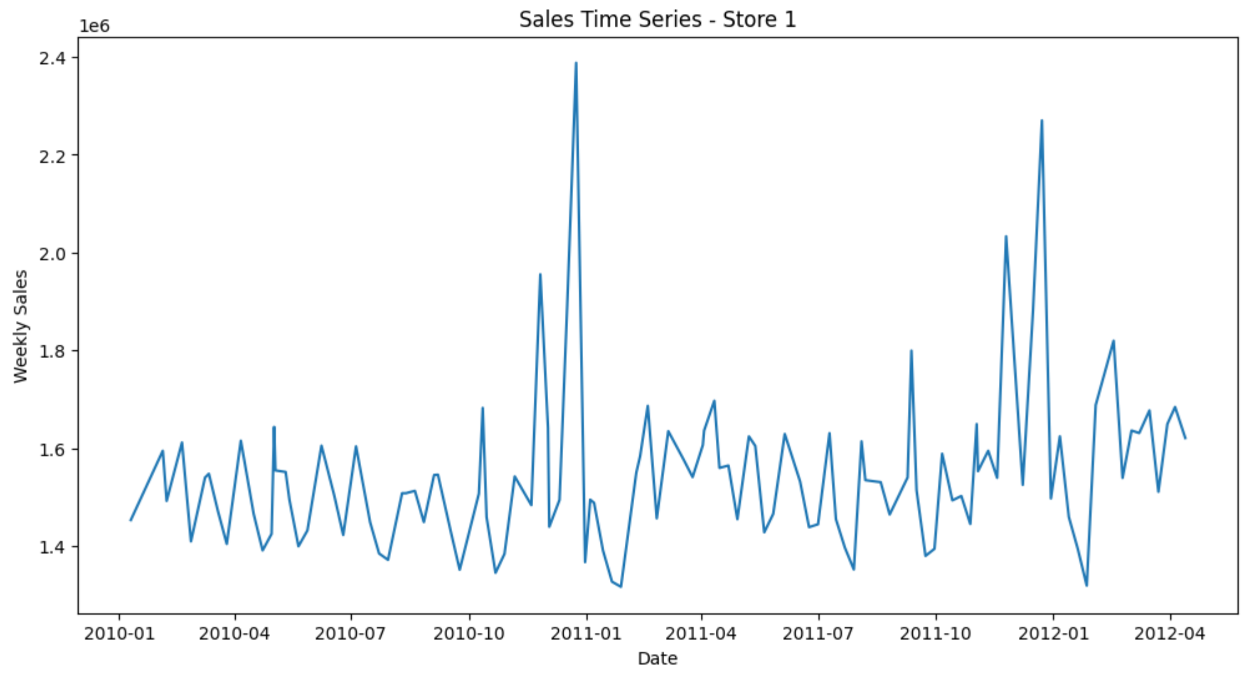


```
[ ] # Split the data into training and test sets
    train_size = int(0.8 * len(sales_series))
    train_data = sales_series[:train_size]
    test_data = sales_series[train_size:]
```

```
[ ] len(train_data), len(test_data)
```

```
(114, 29)
```

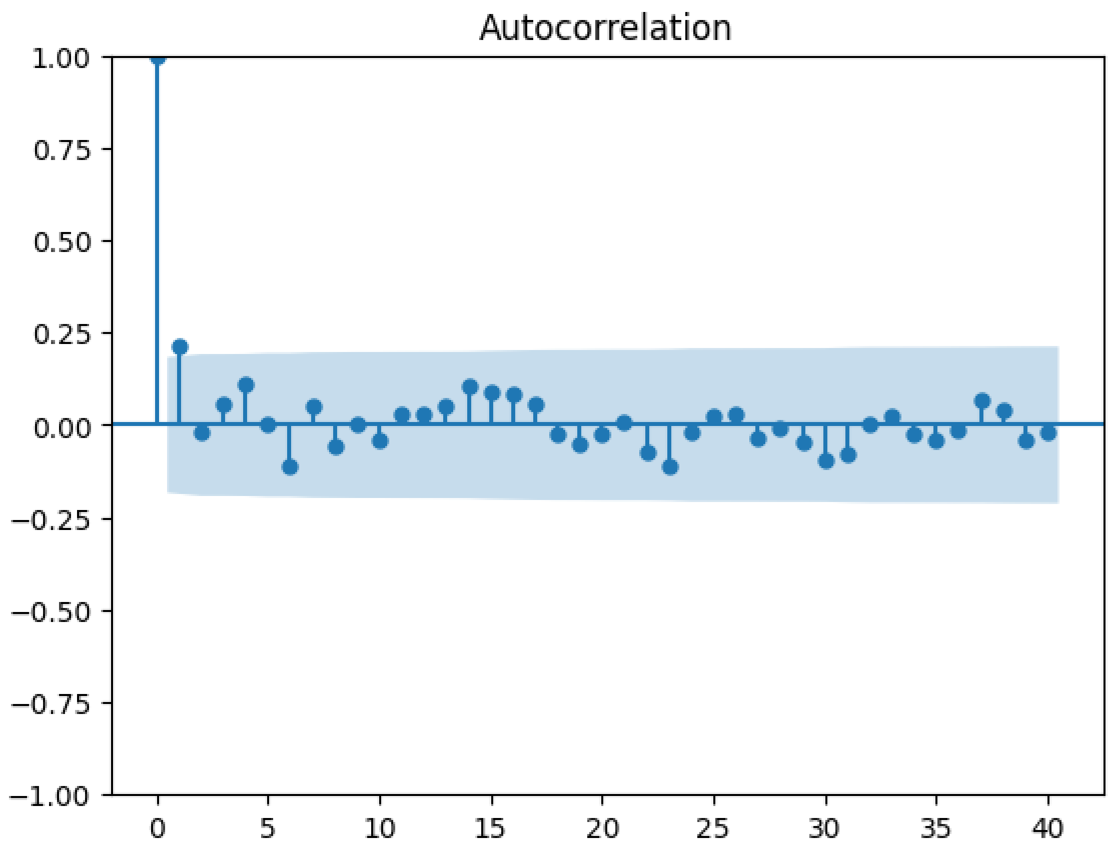
Let us plot time series of training data i.e 114 weeks :



__Plotting Auto-Correlation Function (ACF) :__

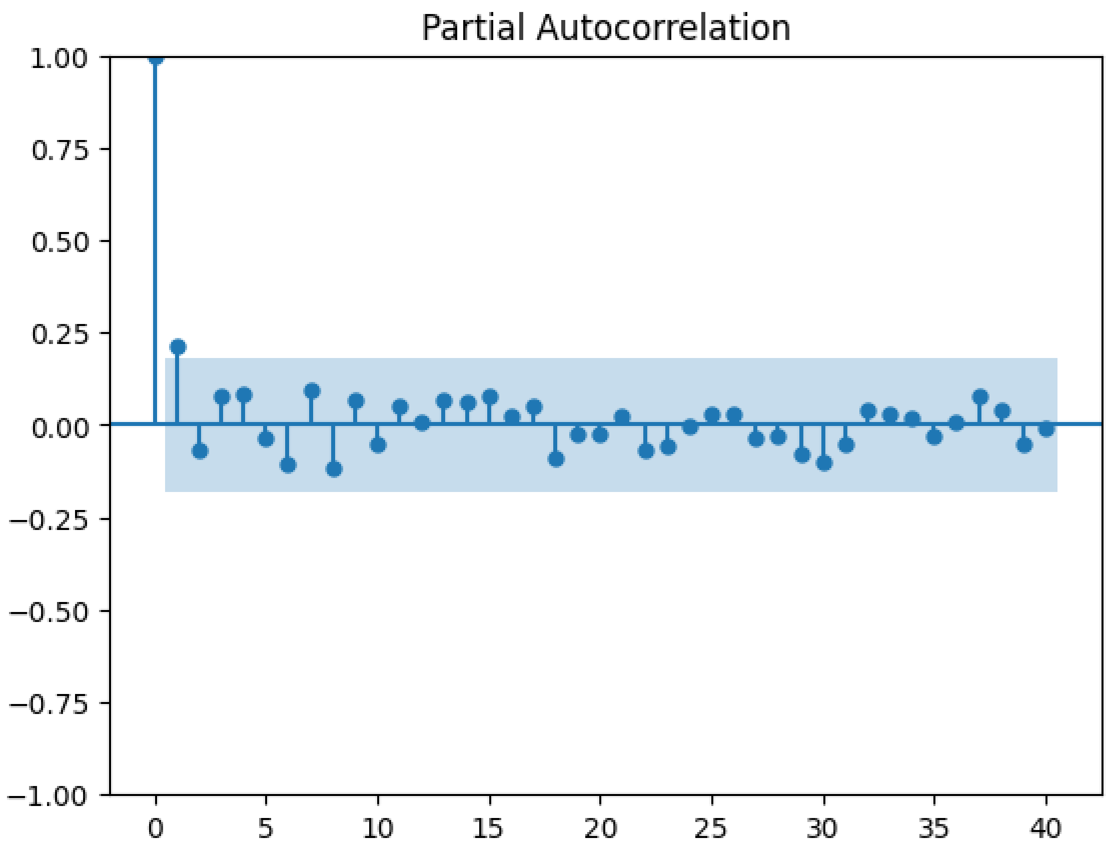
The ACF plot measures the correlation between a time series and its lagged values. It helps in identifying the order of an Autoregressive (AR) model component.

1. The x-axis represents the number of lags , which indicates how many time periods back you are considering for the correlation .
2. The y-axis represents the autocorrelation values , which range from -1 to 1 .



We can see the presence of significant spikes . Significant spikes or "peaks" that extend beyond the shaded region (confidence interval) suggest potential orders for the AR component in an ARIMA (AutoRegressive Integrated Moving Average) model.

Let us now plot *Partial Autocorrelation Function (PACF) Plot*

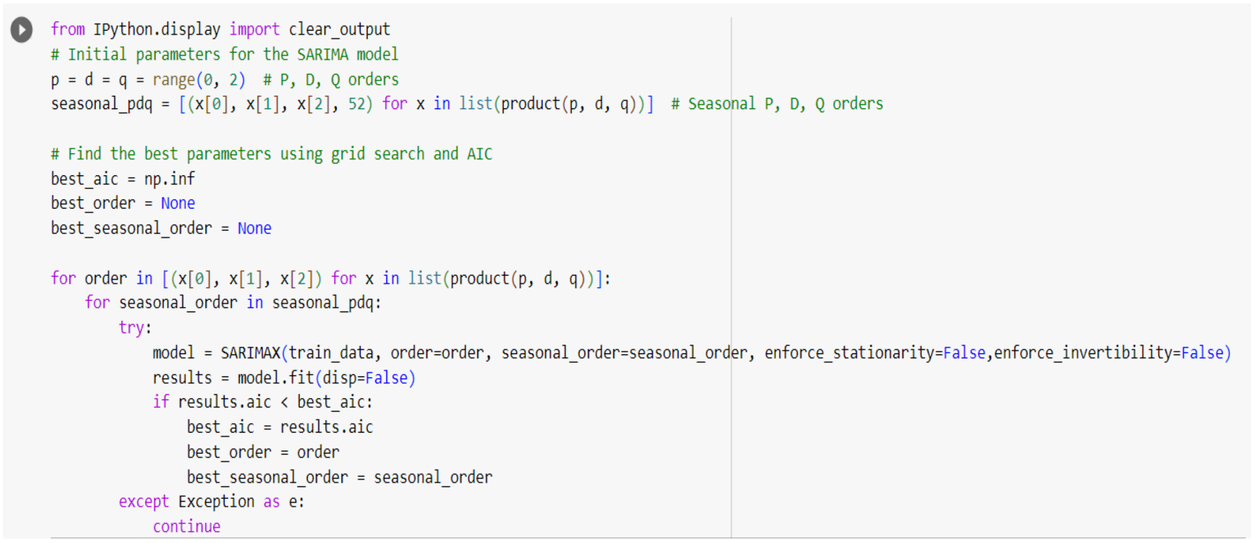


The PACF plot is used to identify the order of the Moving Average (MA) model component in an ARIMA model. It shows the correlation between the current value and its lags, with the influence of intermediate lags removed.

A significant spike in the PACF plot at lag k indicates that there is a correlation between the current value and the value at lag k, while controlling for the influence of lags 1 to k-1.

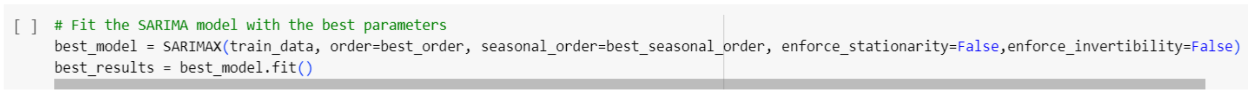
Building Model 1 :

The first model that we will be building will be the SARIMA model on the regular dataset .
Let us first find the best parameters using the Grid Search CV approach that we will be passing to our model .



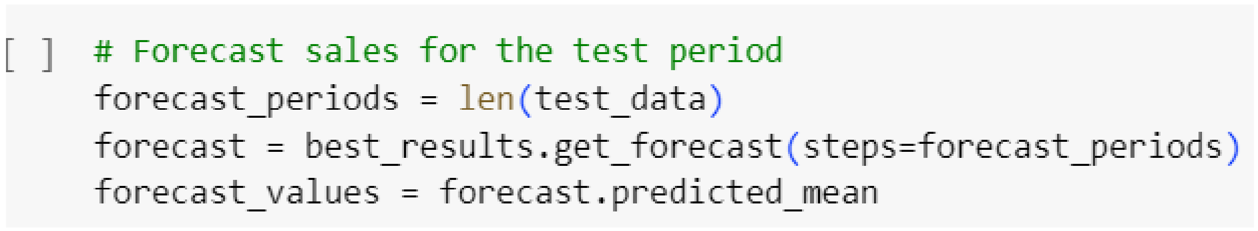
```
from IPython.display import clear_output
# Initial parameters for the SARIMA model
p = d = q = range(0, 2)  # P, D, Q orders
seasonal_pdq = [(x[0], x[1], x[2], 52) for x in list(product(p, d, q))]  # Seasonal P, D, Q orders

# Find the best parameters using grid search and AIC
best_aic = np.inf
best_order = None
best_seasonal_order = None

for order in [(x[0], x[1], x[2]) for x in list(product(p, d, q))]:
    for seasonal_order in seasonal_pdq:
        try:
            model = SARIMAX(train_data, order=order, seasonal_order=seasonal_order, enforce_stationarity=False,enforce_invertibility=False)
            results = model.fit(disp=False)
            if results.aic < best_aic:
                best_aic = results.aic
                best_order = order
                best_seasonal_order = seasonal_order
        except Exception as e:
            continue
```

Now building our first SARIMA model by passing these parameters through the model .



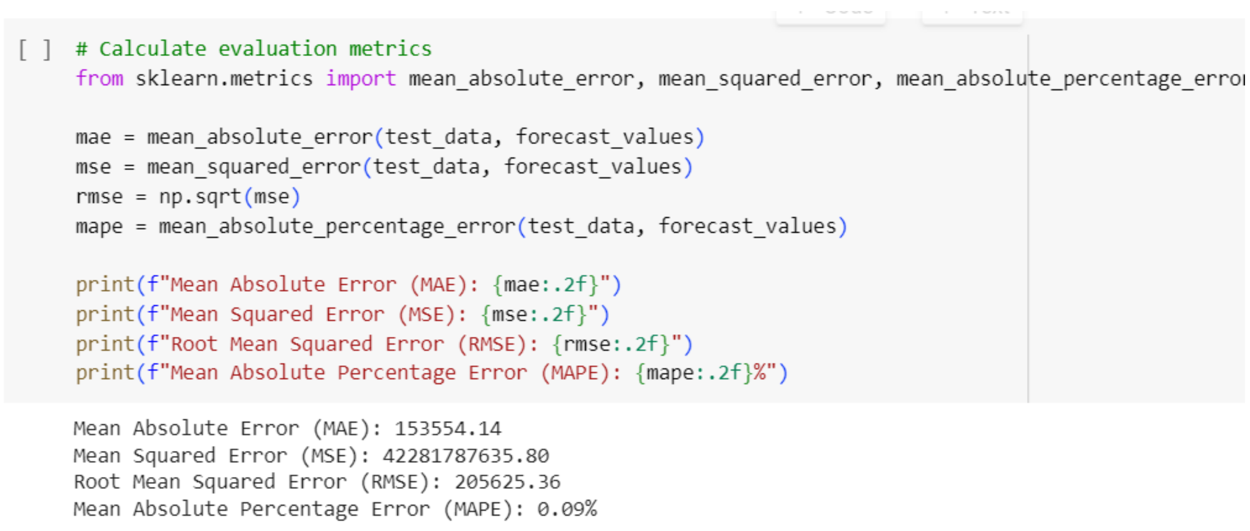
```
[ ] # Fit the SARIMA model with the best parameters
    best_model = SARIMAX(train_data, order=best_order, seasonal_order=best_seasonal_order, enforce_stationarity=False,enforce_invertibility=False)
    best_results = best_model.fit()
```

Predicting the test values :



```
[ ] # Forecast sales for the test period
    forecast_periods = len(test_data)
    forecast = best_results.get_forecast(steps=forecast_periods)
    forecast_values = forecast.predicted_mean
```

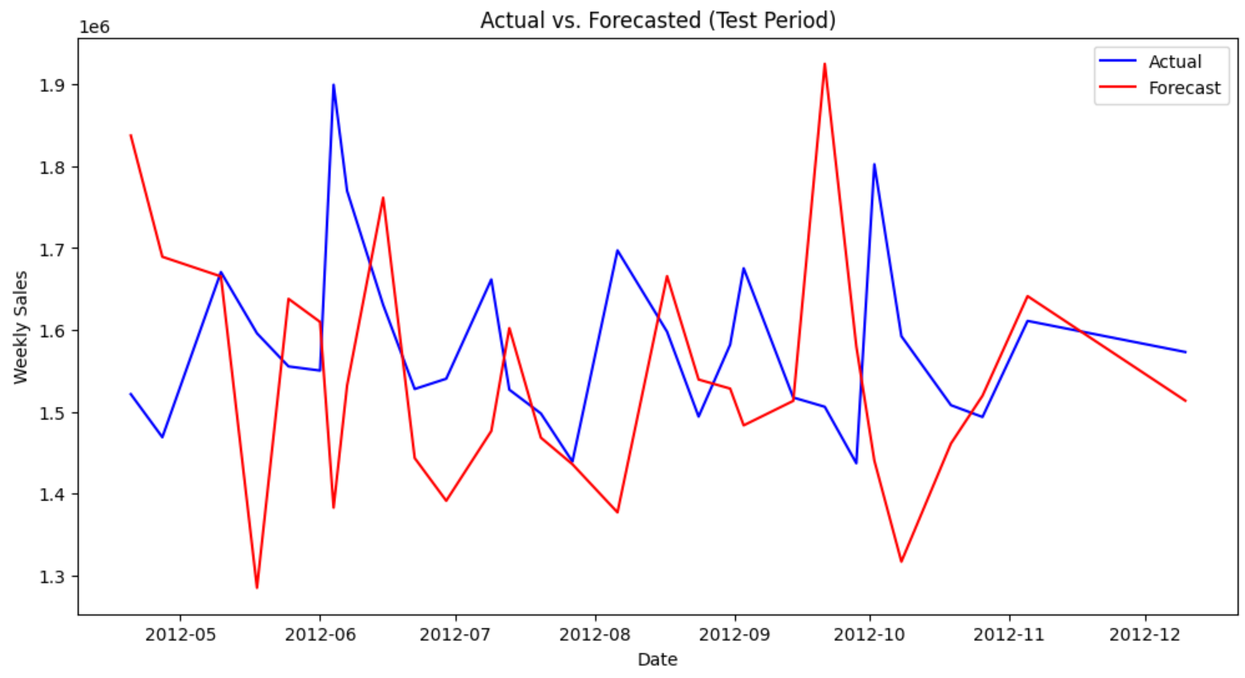
Evaluation Metrics:



```
[ ] # Calculate evaluation metrics
    from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_erro

    mae = mean_absolute_error(test_data, forecast_values)
    mse = mean_squared_error(test_data, forecast_values)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(test_data, forecast_values)

    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"Mean Squared Error (MSE): {mse:.2f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
    print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
Mean Absolute Error (MAE): 153554.14
Mean Squared Error (MSE): 42281787635.80
Root Mean Squared Error (RMSE): 205625.36
Mean Absolute Percentage Error (MAPE): 0.09%
```

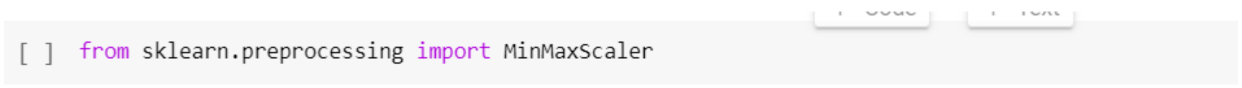
We can see that the Mean Absolute error is high . let us plot the predictions vs actual



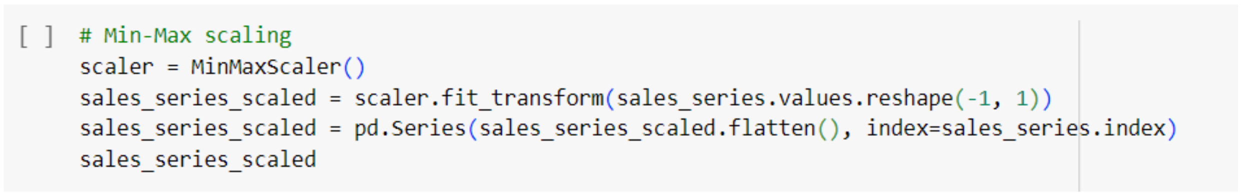
We can see our model is not doing a very good job predicting the values .

Building Model 2 :

This time we will be scaling the values of weekly sales using Mix Max Scaler .

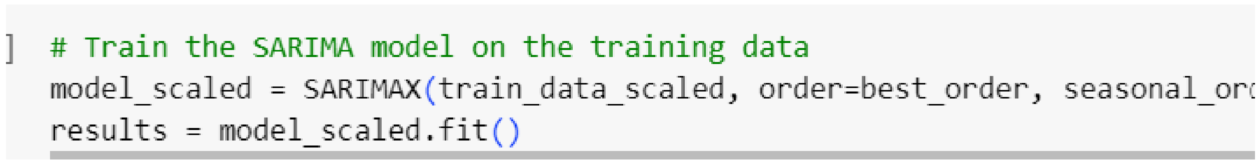


```
[ ]  from sklearn.preprocessing import MinMaxScaler
```



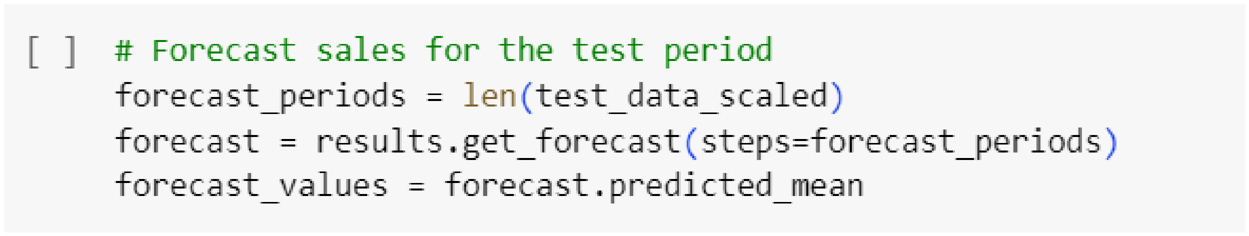
```
[ ]  # Min-Max scaling
     scaler = MinMaxScaler()
     sales_series_scaled = scaler.fit_transform(sales_series.values.reshape(-1, 1))
     sales_series_scaled = pd.Series(sales_series_scaled.flatten(), index=sales_series.index)
     sales_series_scaled
```

Fitting the model on Scaled Dataset :



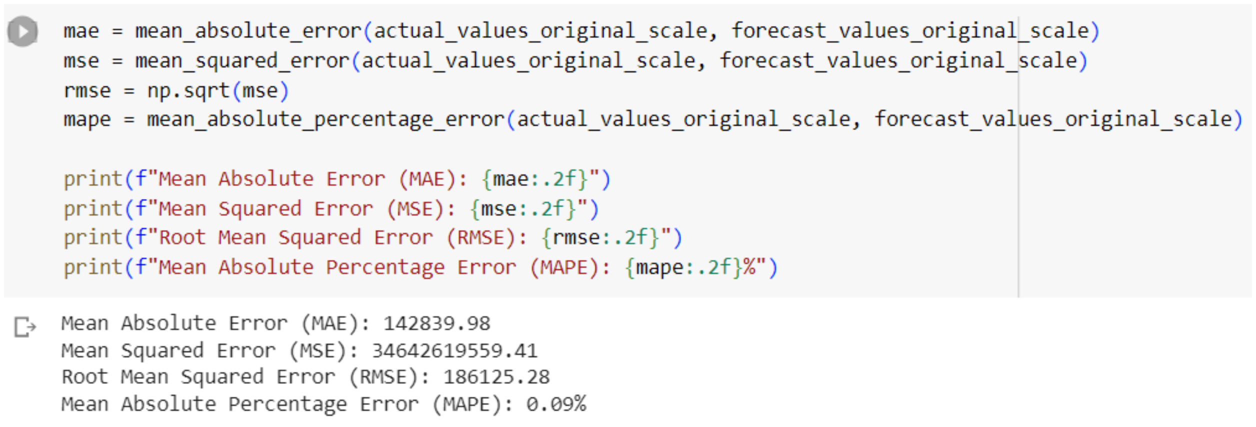
```
]  # Train the SARIMA model on the training data
   model_scaled = SARIMAX(train_data_scaled, order=best_order, seasonal_or
   results = model_scaled.fit()
```

Predicting the values



```
[ ]  # Forecast sales for the test period
     forecast_periods = len(test_data_scaled)
     forecast = results.get_forecast(steps=forecast_periods)
     forecast_values = forecast.predicted_mean
```

Evaluation Metrics

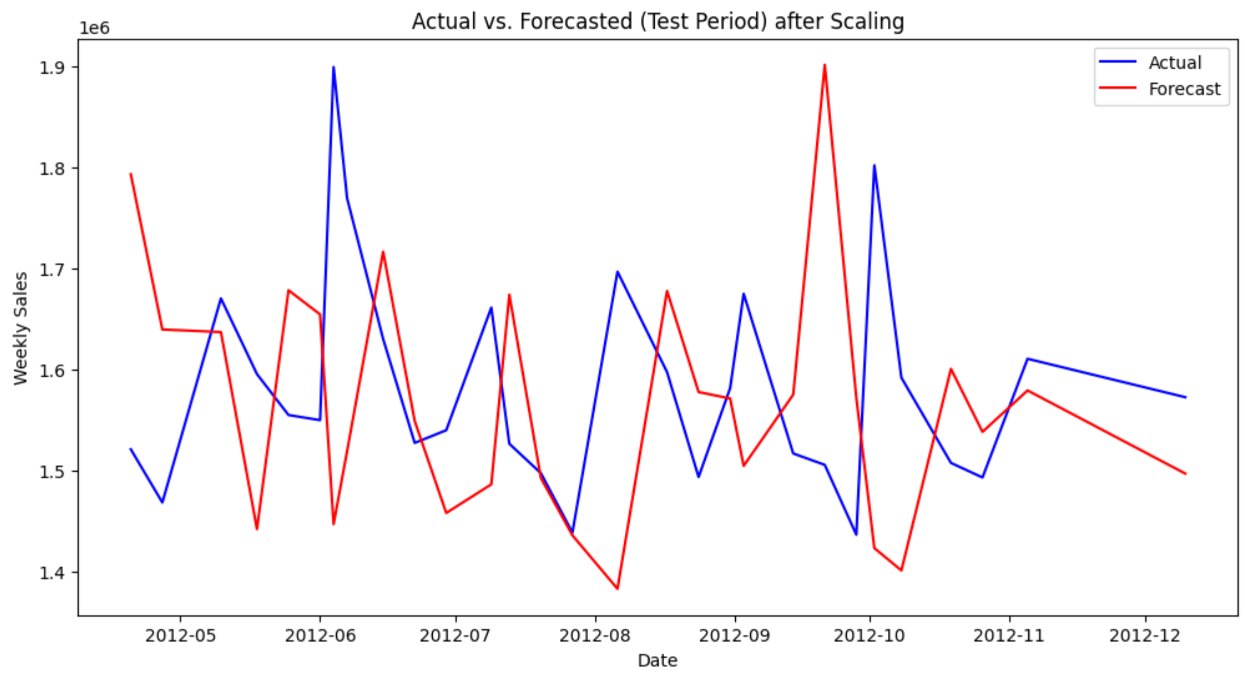


```
mae = mean_absolute_error(actual_values_original_scale, forecast_values_original_scale)
mse = mean_squared_error(actual_values_original_scale, forecast_values_original_scale)
rmse = np.sqrt(mse)
mape = mean_absolute_percentage_error(actual_values_original_scale, forecast_values_original_scale)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
Mean Absolute Error (MAE): 142839.98
Mean Squared Error (MSE): 34642619559.41
Root Mean Squared Error (RMSE): 186125.28
Mean Absolute Percentage Error (MAPE): 0.09%
```

We can see that the error has significantly reduced by scaling the dataset .

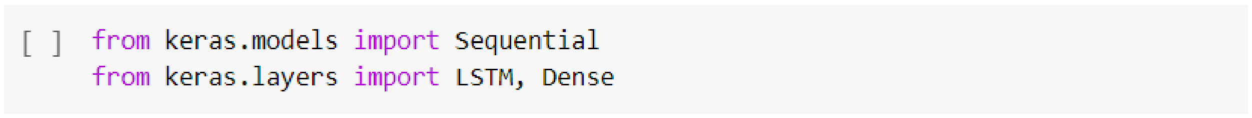
Let us visualize :



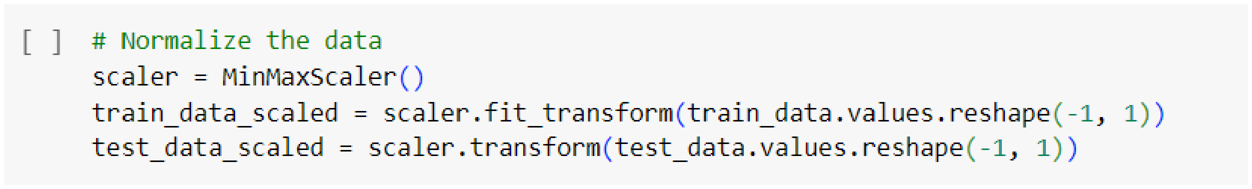
We can see there is much improvement in this prediction .

Building Model 3 :

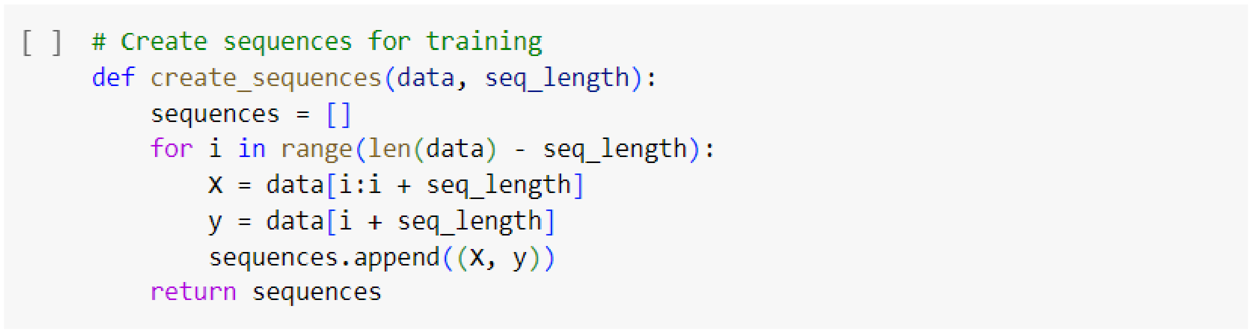
Let us build one more model , this time making use of Neural Networks . The model that we will be using will be LSTM .



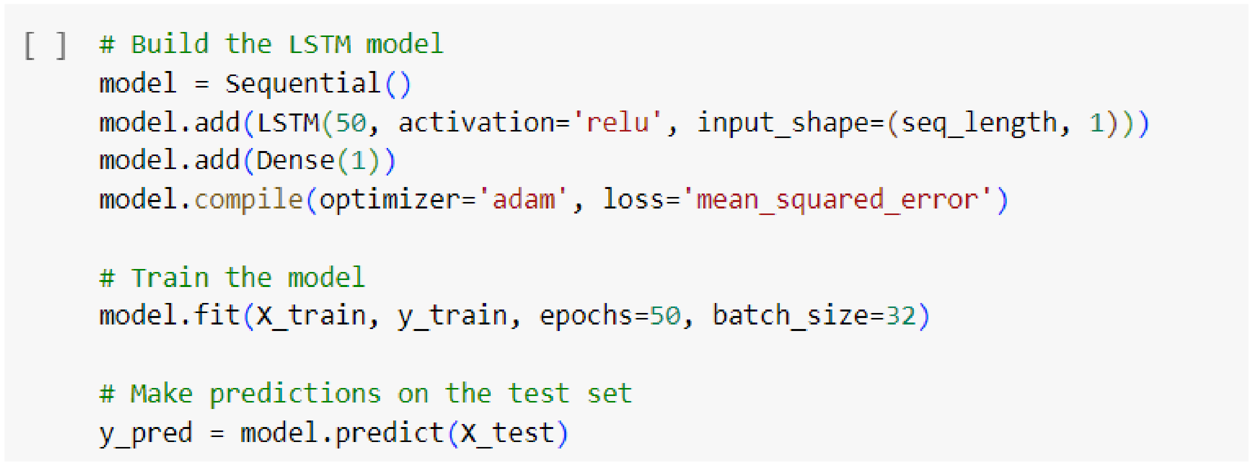
```
[ ] from keras.models import Sequential
    from keras.layers import LSTM, Dense
```



```
[ ] # Normalize the data
    scaler = MinMaxScaler()
    train_data_scaled = scaler.fit_transform(train_data.values.reshape(-1, 1))
    test_data_scaled = scaler.transform(test_data.values.reshape(-1, 1))
```



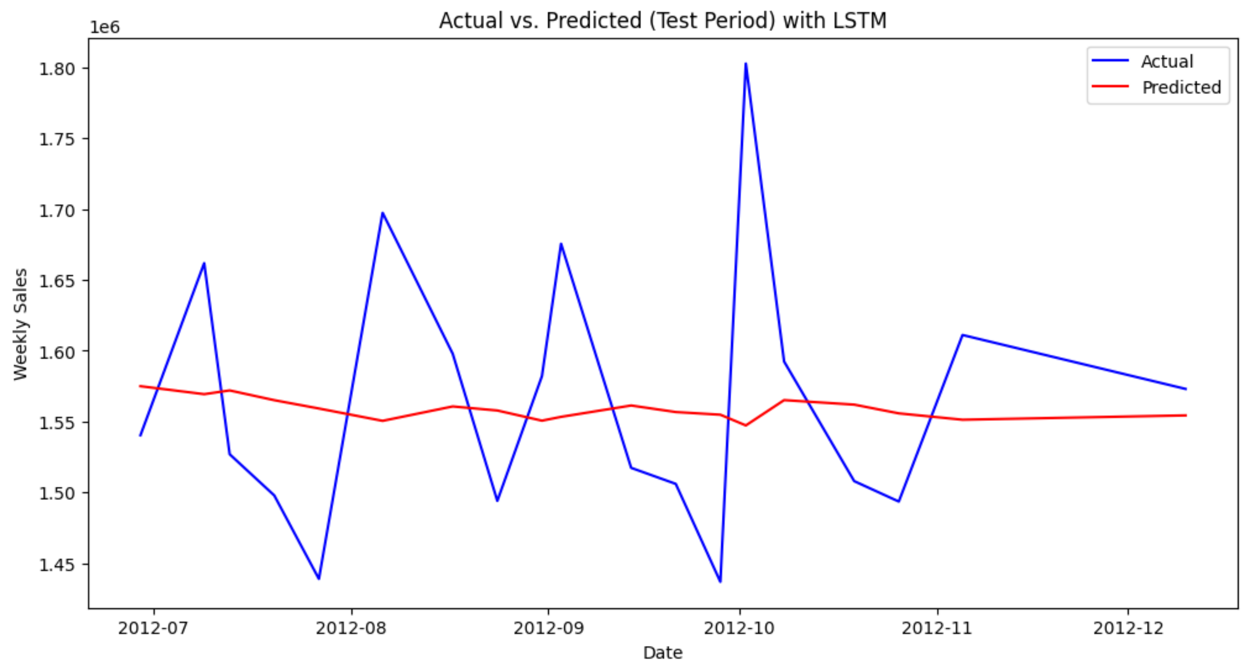
```
[ ] # Create sequences for training
    def create_sequences(data, seq_length):
        sequences = []
        for i in range(len(data) - seq_length):
            X = data[i:i + seq_length]
            y = data[i + seq_length]
            sequences.append((X, y))
        return sequences
```



```
[ ] # Build the LSTM model
    model = Sequential()
    model.add(LSTM(50, activation='relu', input_shape=(seq_length, 1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(X_train, y_train, epochs=50, batch_size=32)

    # Make predictions on the test set
    y_pred = model.predict(X_test)
```

```
[ ] # Calculate evaluation metrics
    mae = mean_absolute_error(y_test_actual, y_pred_actual)
    mse = mean_squared_error(y_test_actual, y_pred_actual)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_test_actual, y_pred_actual)

    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"Mean Squared Error (MSE): {mse:.2f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
    print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

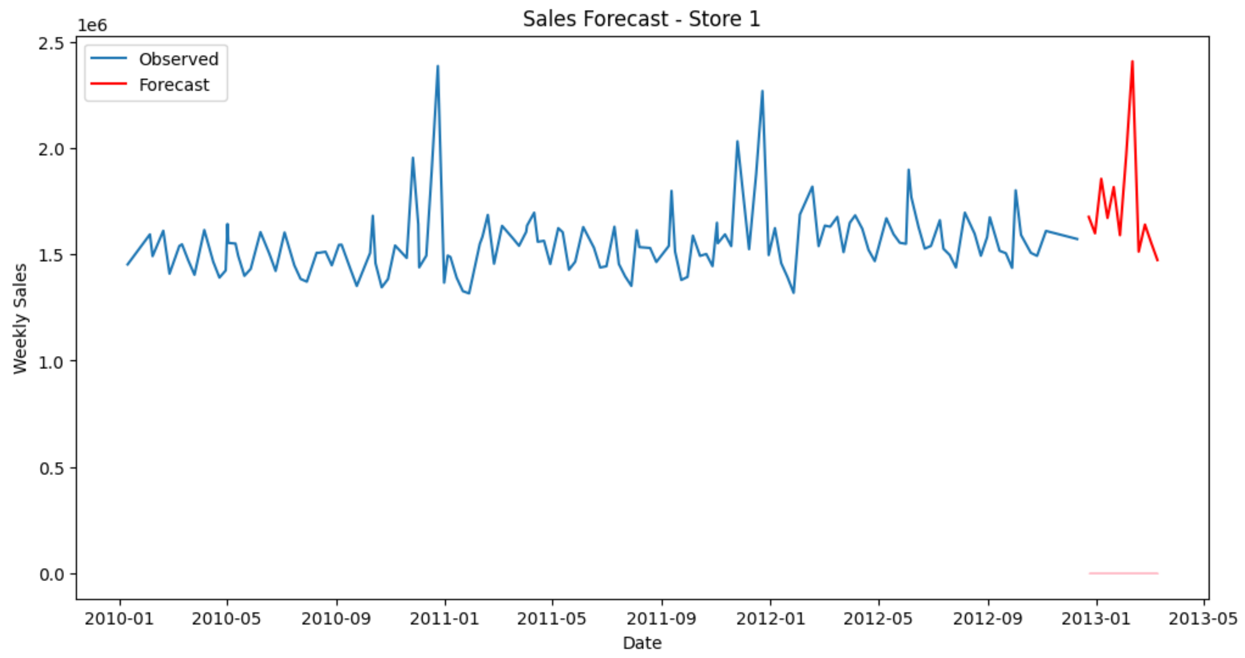
```
Mean Absolute Error (MAE): 76295.33
Mean Squared Error (MSE): 8865988978.40
Root Mean Squared Error (RMSE): 94159.38
Mean Absolute Percentage Error (MAPE): 0.05%
```



We can see that the LSTM Model is doing a poor job of prediction because Neural Networks need a lot more data to perform well .

Therefore so far the best model that we built was the model 2 . Please note , in order to improve the performance further we would require more data .

Let us now use our best model to forecast the next 12 weeks weekly sales .



Conclusion

In the course of this report, we explored various aspects of sales forecasting for Walmart stores, leveraging two distinct modeling approaches: Seasonal Autoregressive Integrated Moving Average (SARIMA) and Long Short-Term Memory (LSTM) neural networks. This journey yielded valuable insights into the importance of data preprocessing, scaling, and the significance of data volume in the context of predictive modeling.

<u>The Significance of Scaling:</u>

One crucial takeaway from our analysis is the pivotal role played by data scaling in model performance. Model 2, which incorporated proper data scaling techniques, notably outperformed other models. Scaling ensures that all features are on a consistent scale, preventing certain features from dominating the learning process merely due to their larger magnitudes. This normalization of the data enables machine learning algorithms to converge more quickly and make accurate predictions. As a result, scaling is a fundamental preprocessing step that should not be overlooked when working with diverse features and complex models, as demonstrated by the substantial improvement in our model's performance.

The Impact of Data Volume:

Another noteworthy observation is the significance of data volume in enhancing the accuracy of our forecasting models. As we delved into the intricacies of time series modeling, it became evident that having a more extensive historical dataset provides models like SARIMA with a richer context for understanding and capturing intricate seasonal patterns. While LSTM is a powerful model with the capacity to learn from smaller datasets, SARIMA's ability to exploit historical trends and seasonality makes it a robust choice when ample data is available. Therefore, for businesses like Walmart that have access to extensive sales records, investing in data collection and preservation can lead to more accurate and reliable sales forecasts.

SARIMA vs. LSTM:

In the realm of seasonal pattern detection and forecasting, our analysis revealed that the SARIMA model consistently outperformed LSTM. SARIMA's strength lies in its ability to capture and model seasonality explicitly through its autoregressive and moving average components. This superiority is particularly evident when dealing with time series data characterized by clear and recurrent seasonal patterns, as is often the case in the retail industry.

In conclusion, the success of our SARIMA model underscores the importance of recognizing and exploiting seasonal patterns in time series forecasting. This report advocates for the careful consideration of data preprocessing, including scaling, and the collection of extensive historical data to enhance forecasting accuracy. For businesses aiming to optimize their sales forecasting efforts, SARIMA stands as a robust choice, especially when confronted with pronounced seasonality in their data. By adopting these practices and model preferences, organizations can make more informed decisions, allocate resources more effectively, and ultimately, achieve a competitive edge in the dynamic world of retail.