Capstone Project -

WALMART SALES PREDICTION PROJECT

-by Shivani Gupta

Table of Contents

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.

Project Objective

1. Using the above 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.

Data Description

The walmart.csv dataset contains 6435 rows and 8 columns. The range of the data Date column varies from Jan 2010 to Dec 2012.

Feature Name	**Description**
Store	Store number
Date	Week of Sales
Weekly_Sales	Sales for the given store in that week
Holiday_Flag	If it is a holiday week
Temperature	Temperature on the day of the sale
Fuel_Price	Cost of the fuel in the region
CPI	Consumer Price Index
Unemployment	Unemployment Rate

Data Preprocessing Steps And Inspiration

Checking for the insights into data:

```
In [70]: warnings.filterwarnings("ignore", category=FutureWarning)
         walmart_data.describe(include='all')
```

Out[70]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Unemployment_treated	Year	Month
count	6434.000000	6434	6.434000e+03	6434.000000	6434.000000	6434.000000	6434.000000	6434.000000	6434.000000	6434.000000	6434
unique	NaN	143	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12
top	NaN	2010-05-02 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	April
freq	NaN	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	629
first	NaN	2010-01-10 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	NaN	2012-12-10 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	23.002487	NaN	1.047041e+06	0.069941	60.673531	3.358661	171.575257	7.999024	7.871061	2010.965030	NaN
std	12.987660	NaN	5.643776e+05	0.255067	18.429779	0.459035	39.358967	1.876003	1.520766	0.797081	NaN
min	1.000000	NaN	2.099862e+05	0.000000	5.540000	2.472000	126.064000	3.879000	4.294500	2010.000000	NaN
25%	12.000000	NaN	5.531677e+05	0.000000	47.495000	2.933000	131.735000	6.891000	6.891000	2010.000000	NaN
50%	23.000000	NaN	9.608457e+05	0.000000	62.675000	3.446500	182.616521	7.874000	7.874000	2011.000000	NaN
75%	34.000000	NaN	1.420282e+06	0.000000	74.945000	3.735000	212.745096	8.622000	8.622000	2012.000000	NaN
max	45.000000	NaN	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000	11.218500	2012.000000	NaN

- **Convert Data Types**: Ensure that the data types of variables are appropriate for their respective columns.

```
In [4]: walmart_data.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
```

Since Date column datatype is object so lets change it to DateTime.

```
In [71]: warnings.filterwarnings("ignore", category=UserWarning)
         walmart_data['Date'] = walmart_data['Date'].astype('datetime64')
```

```
In [6]: walmart_data.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)
memory usage: 402.3 KB
```

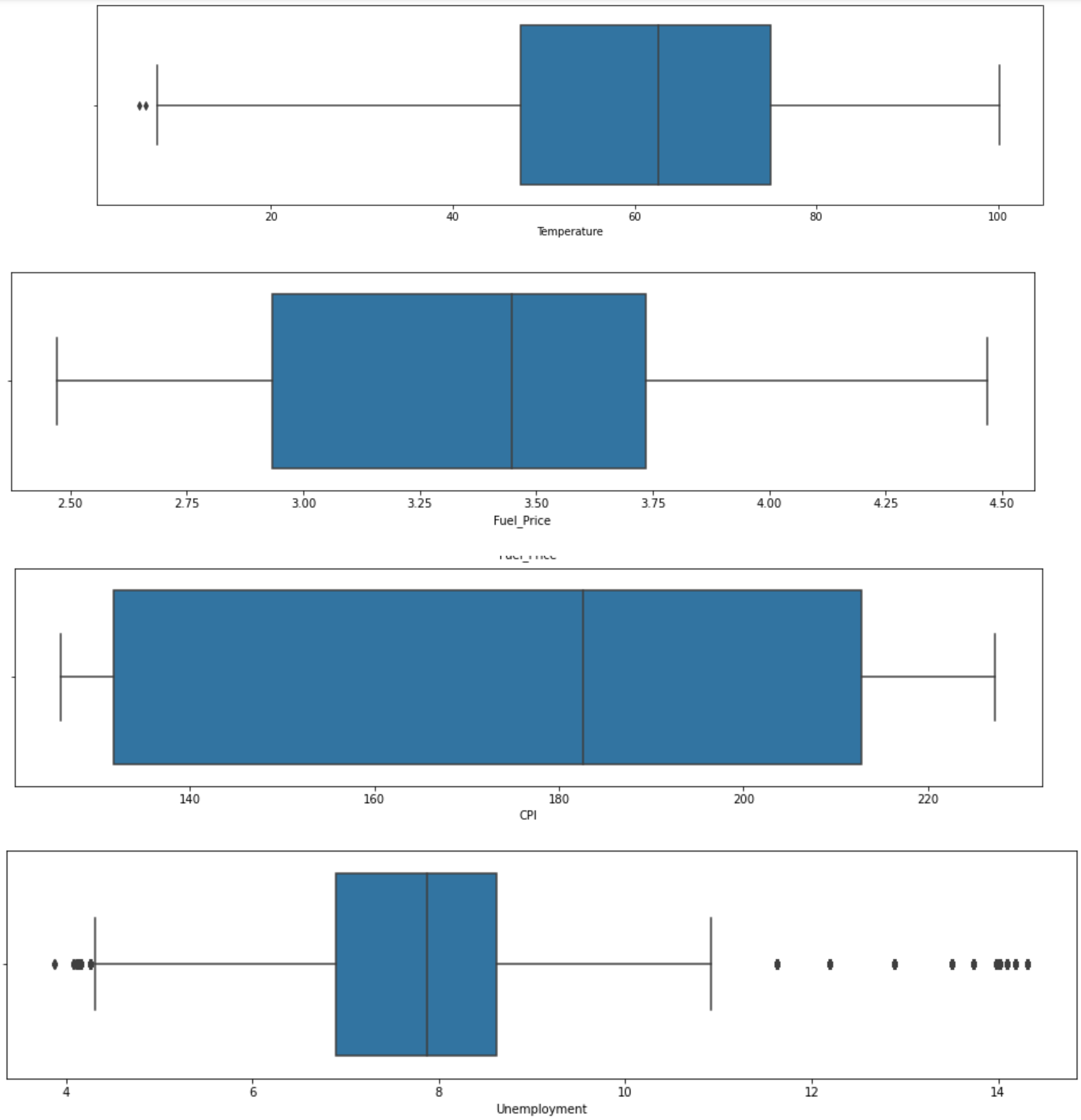
- **Handle Missing Values**: Check for missing/Null values in the dataset and decide on the appropriate strategy for handling them.

```
In [10]: walmart_data.isnull().sum()

Out[10]: Store           0
         Date            0
         Weekly_Sales    0
         Holiday_Flag    0
         Temperature     0
         Fuel_Price      0
         CPI             0
         Unemployment    0
         dtype: int64
```

- **Remove Outliers**: Identify outliers in the data that may adversely affect the analysis or model performance. Decide whether to remove outliers or apply appropriate transformations to mitigate their impact.

Checking for outliers

In [73]:
```
fig,axis = plt.subplots(4,figsize=(16,16))
x=walmart_data[['Temperature','Fuel_Price','CPI','Unemployment']]
for i, column in enumerate(x):
    sns.boxplot(walmart_data[column],ax=axis[i])
warnings.filterwarnings("ignore", category=FutureWarning)
```



We can see that there are outliers for column Unemployment and Temperature.Checking the number of data rows that have outliers and deciding whether to delete them or not.

In [12]:
```python
walmart_data[(walmart_data['Unemployment']<4.5) | (walmart_data['Unemployment']>11)].shape
```

Out[12]:
```
(520, 8)
```

```python
# Since data is larger so we cannot drop this. Treating the outliers here.
```

Since data is larger so we cannot drop this. Treating the outliers here.

In [13]:
```python
# Using IQR Method
def thr_min_max(col):
    p25= walmart_data[col].quantile(0.25)
    p75=walmart_data[col].quantile(0.75)
    IQR=p75-p25

    thr_min,thr_max = p25 - 1.5* IQR , p75+1.5*IQR
    return thr_min,thr_max


def treating(val):
    if(val<thr_min):
        return thr_min
    elif(val>thr_max):
        return thr_max
    else:
        return val
```

In [14]:
```python
thr_min,thr_max = thr_min_max('Unemployment')
print(thr_min,thr_max)
walmart_data['Unemployment_treated'] = walmart_data['Unemployment'].apply(treating)
```

```
4.2945 11.218499999999999
```

So, Unemployment data is treated and new column is created for the same named Unemployment_treated.

In [78]:
```python
walmart_data.sample(7)
```

Out[78]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Unemployment_treated	Year	Month	Day
5360	38	2011-03-06	396826.06	0	71.81	4.031	129.049032	13.736	11.2185	2011	March	6
2260	16	2012-04-20	436221.26	0	43.61	3.936	197.722738	6.169	6.1690	2012	April	20
3703	26	2012-07-20	1049625.90	0	66.75	3.610	138.233193	7.405	7.4050	2012	July	20
3442	25	2010-04-16	715311.60	0	52.16	2.899	203.730749	7.856	7.8560	2010	April	16
5503	39	2011-03-06	1541745.59	0	83.59	3.699	214.016280	8.300	8.3000	2011	March	6
5526	39	2011-11-11	1456957.38	0	63.11	3.297	216.721737	7.716	7.7160	2011	November	11
2651	19	2011-07-29	1298775.80	0	74.86	4.004	135.963935	7.806	7.8060	2011	July	29

Here, we can see that in Unemployment column, value was larger than 11 so we had it replaced with threshold maximum value.

Now, checking number of outlier rows for Temperature column.

In [15]:
```
walmart_data[(walmart_data['Temperature']<5)]
```
Out[15]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Unemployment_treated
910	7	2011-04-02	558027.77	0	-2.06	3.011	191.762589	8.818	8.818

In [16]:
```
# Only one column with outlier of Temperature. So dropping it off.

walmart_data = walmart_data.drop(walmart_data[(walmart_data['Temperature']<5)].index)
```

Since there was only one row so instead of treating it we can delete it directly and it will not have any impact on our further observations.

- **Feature Engineering**: Explore the dataset to identify any potential feature engineering opportunities. This may involve creating new variables, extracting useful information from existing variables, or transforming variables to enhance their predictive power.

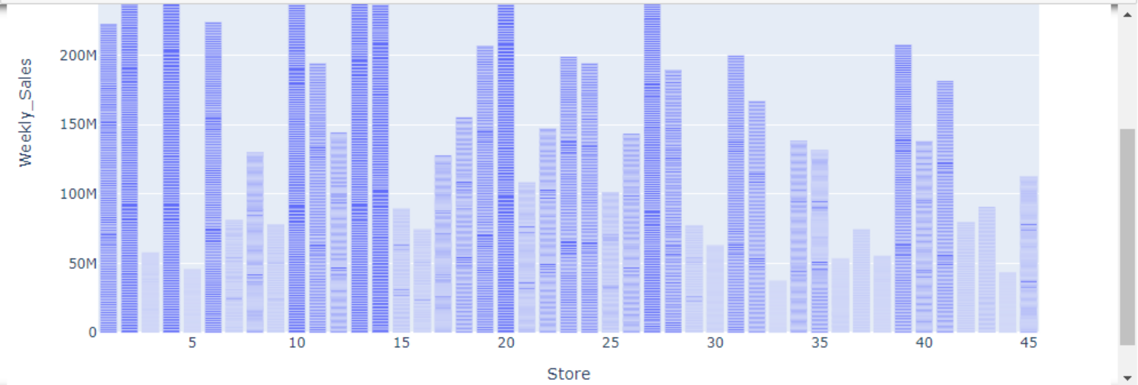
.

In [18]:
```
walmart_data['Store'].unique()
# 45 stores data is present
```
Out[18]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45], dtype=int64)
```

Here, we can see that there are total of 45 stores data present in the dataset.

Checking which store has maximum sales:

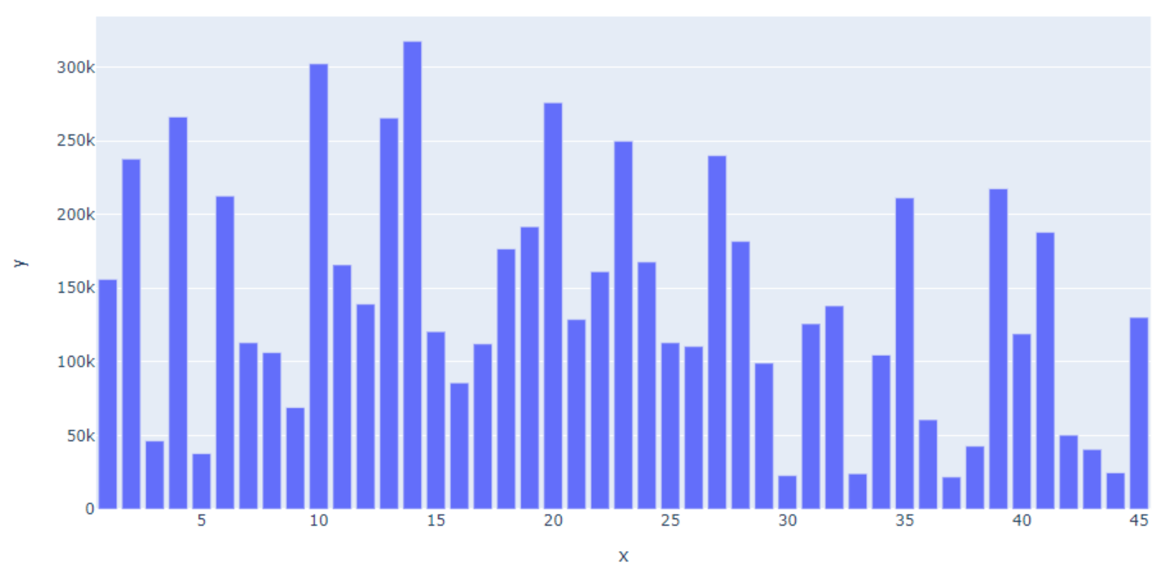
```
In [79]: # Which store has maximum sales
         px.bar(data_frame=walmart_data,x='Store',y='Weekly_Sales')
```



Here we can see store 20 has max sales and then Store 4 has 2nd highest sale.

Checking which store has maximum standard deviation:

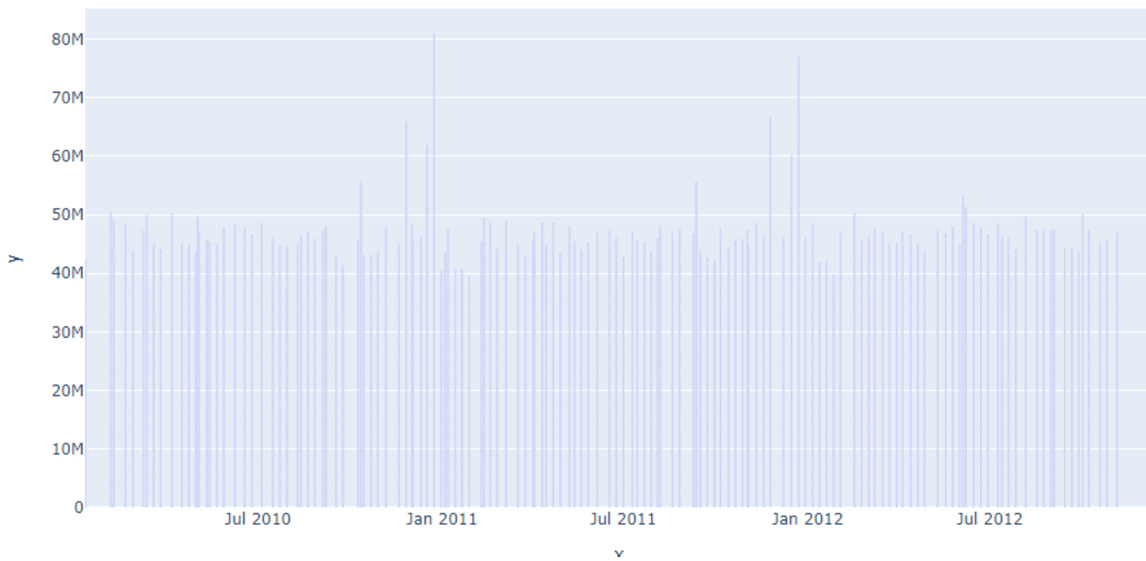
```
# store having maximum standard deviation
store_std = walmart_data.groupby('Store')['Weekly_Sales'].std().reset_index()
px.bar(x=store_std['Store'],y=store_std['Weekly_Sales'])
```



Thus, the store which has maximum standard deviation is store number 14, which means that the sales of Store 14 varies the most and is not constant.

Checking which month has highest sales:

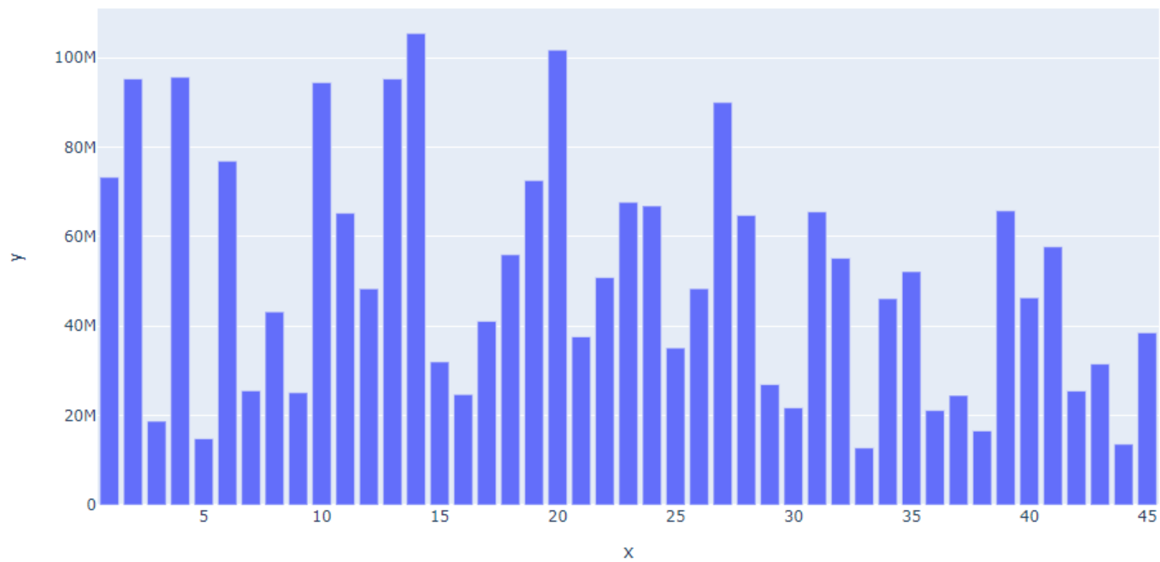
```
px.bar(x=walmart_data['Date'],y=walmart_data['Weekly_Sales'])
```



We can observer from above graph that sales for Month of December is maximum sales in a year. So, during the December months, year end sales and other sales attracts the customer more and also, the number of holidays at that time is more. So, the total sales in that month increases as compared to rest of the year.

Checking which Store has maximum sales for Year 2010:

```
In [26]: data = walmart_data[(walmart_data['Date']> '2010-01-01') & (walmart_data['Date']< '2011-01-01')].groupby('Store')
         ['Weekly_Sales'].sum()
         data=pd.DataFrame(data).reset_index()
         px.bar(x=data['Store'], y=data['Weekly_Sales'])
```



We can see that store 14 has maximum Sales in Year 2010.

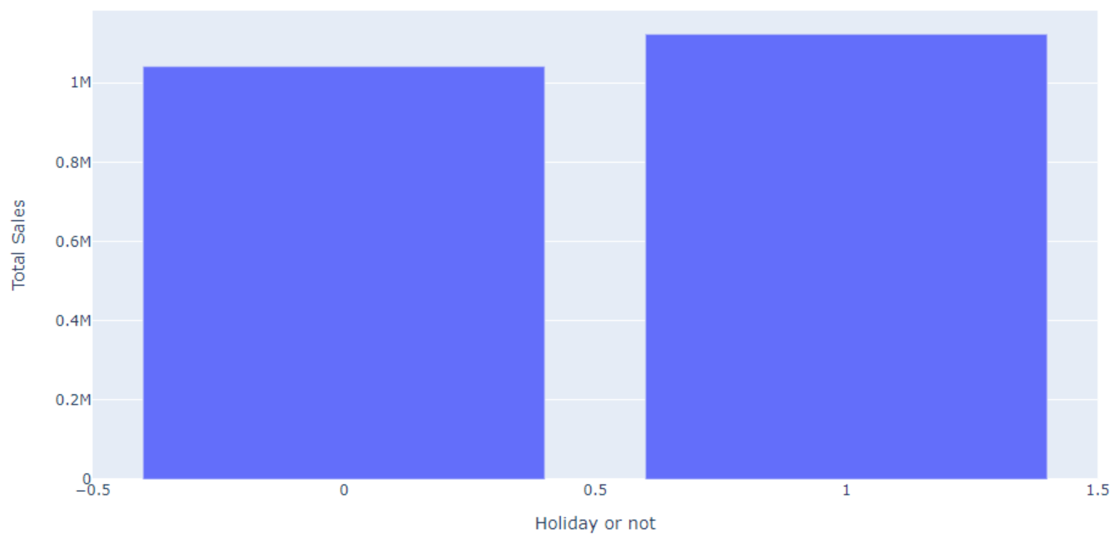
Check whether the Holidays have effect on Sales or not

```
In [27]: # Checking the effect of holiday on Sales
         data = walmart_data.groupby('Holiday_Flag')['Weekly_Sales'].mean()
         data=pd.DataFrame(data).reset_index()
         data
```

Out[27]:

	Holiday_Flag	Weekly_Sales
0	0	1.041337e+06
1	1	1.122888e+06

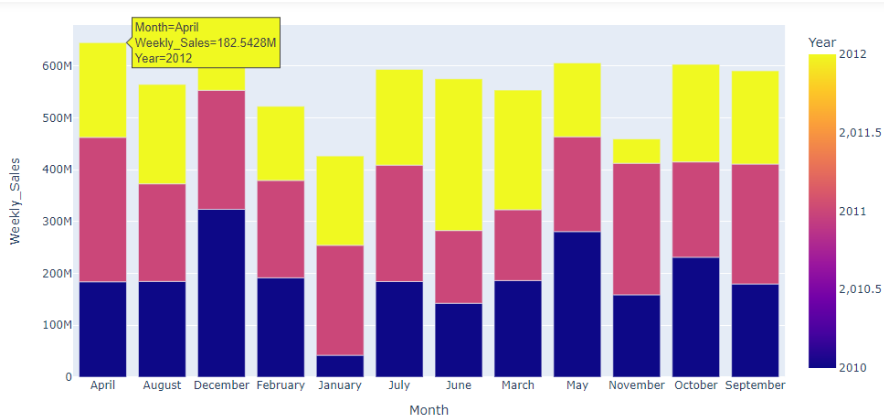
```
In [82]: warnings.filterwarnings("ignore", category=DeprecationWarning)
         px.bar(x=data['Holiday_Flag'],y=data['Weekly_Sales'],labels={'x':'Holiday or not','y':'Total Sales'})
```



Here, we can see that Sales is higher during holidays.

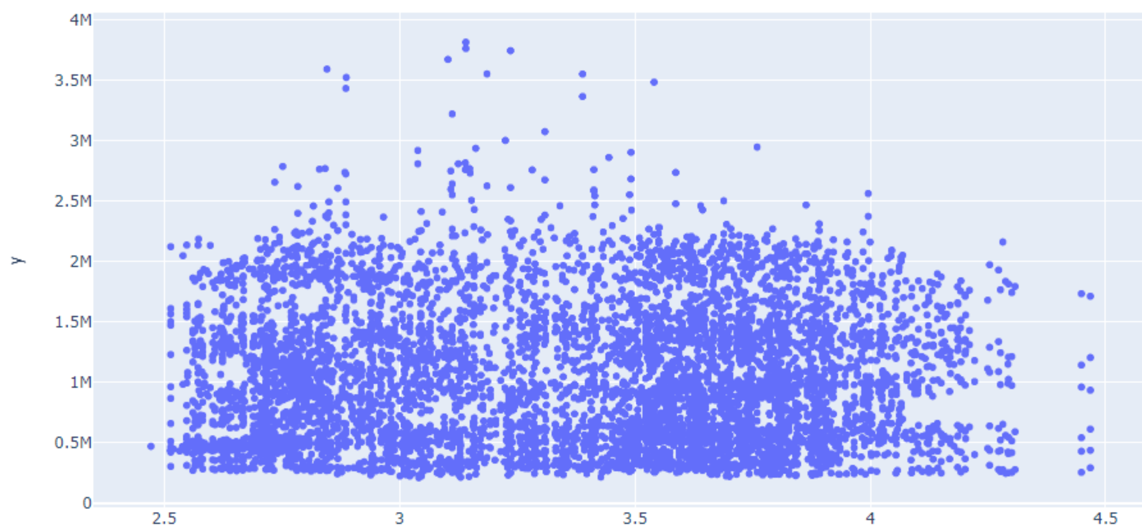
Checking Year Wise Monthly Sales:

In [30]:
```
#Year-wise Monthly Sales

walmart_data['Year'] = pd.to_datetime(walmart_data['Date']).dt.year
walmart_data['Month'] = pd.to_datetime(walmart_data['Date']).dt.month_name()
walmart_data['Day'] = pd.to_datetime(walmart_data['Date']).dt.day
year_month_sales = walmart_data.groupby(['Year', 'Month'])['Weekly_Sales'].sum().reset_index()
print(year_month_sales.sort_values(by=['Year', 'Month']))
px.bar(data_frame=year_month_sales, x='Month',y='Weekly_Sales',color='Year')
```



Check effect of Fuel_Price on Weekly_Sales

```
px.scatter(x=walmart_data['Fuel_Price'],y=walmart_data['Weekly_Sales'])
```



From this we can say there is no direct relation between fuel price and sales.

- **Feature Scaling**: Normalize or standardize numerical features.

```
In [38]: # Standardization

         from sklearn.preprocessing import StandardScaler
         sc= StandardScaler()
         X = walmart_data[['Store','Fuel_Price','CPI','Unemployment','Year']]
         Y = walmart_data['Weekly_Sales']
         X_std = sc.fit_transform(X)
         X_std

Out[38]: array([[-1.69423879, -1.7138609 ,  1.00419743,  0.05702792, -1.21079834],
                [-1.69423879, -1.76614857,  1.00790238,  0.05702792, -1.21079834],
                [-1.69423879, -1.84022277,  1.00909593,  0.05702792, -1.21079834],
                ...,
                [ 1.69385582,  1.39725531,  0.52729082,  0.35609123,  1.29855149],
                [ 1.69385582,  1.32971707,  0.527382  ,  0.35609123,  1.29855149],
                [ 1.69385582,  1.14017428,  0.52682414,  0.35609123,  1.29855149]])
```

Choosing the Algorithm for Walmart Data

I have chosen Random Forest Regressor Algorithm for this project because after checking multiple algorithms we have maximum Accuracy and least error in Random Forest.

I have chosen the Random Forest Regressor algorithm for this project because, after checking multiple algorithms, it has shown the highest accuracy and the least error compared to other algorithms.

Here are some reasons why I have chosen the Random Forest Regressor algorithm:

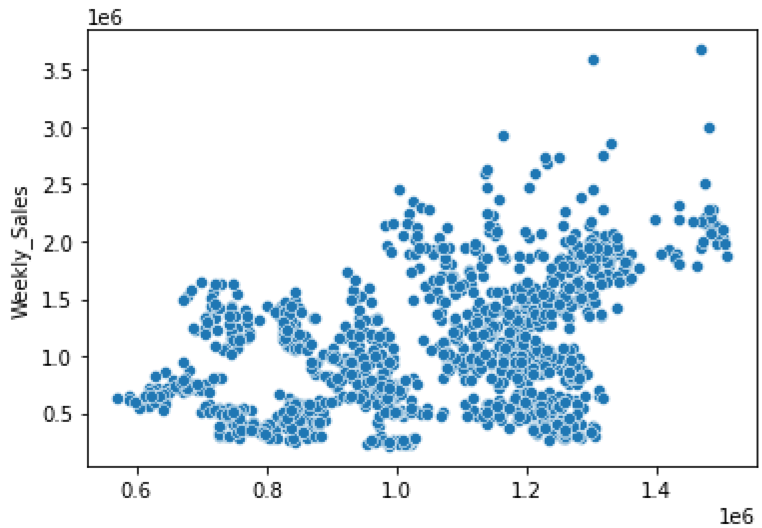
- High Accuracy: In my evaluation of various algorithms, the Random Forest Regressor has demonstrated the highest accuracy in predicting the target variable. It has shown better performance in capturing complex relationships between the features and the target, resulting in more accurate predictions.
- Robustness: Random Forest Regressor is robust against overfitting and tends to generalize well to unseen data. By constructing multiple decision trees and averaging their predictions, it reduces the impact of individual noisy or biased trees, leading to improved robustness and more reliable predictions.
- Feature Importance: Random Forest Regressor provides valuable insights into feature importance. It calculates the importance score of each feature, indicating its contribution to the prediction. This information can help identify the most influential features in the dataset, allowing for better understanding and interpretation of the underlying patterns.
- Handling Nonlinear Relationships: The Random Forest Regressor algorithm can effectively capture nonlinear relationships between the features and the target variable. It does not assume linearity and can handle complex interactions and dependencies among features, making it suitable for datasets with intricate patterns.
- Robustness to Outliers and Missing Values: Random Forest Regressor is robust to outliers and can handle missing values without requiring imputation. It uses the information available in other features to make predictions, reducing the impact of missing or outlier values on the overall model performance.

By selecting the Random Forest Regressor algorithm, I aim to leverage its strengths to build a reliable and accurate prediction model for the given dataset. It has demonstrated superior performance in terms of accuracy and error reduction, making it a suitable choice for this project.

LINEAR REGRESSION MODEL:

```
In [39]: # Linear Regression :

         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression
         X_train, X_test, Y_train, Y_test = train_test_split(X_std,Y,test_size=0.2)
         X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

Out[39]: ((5147, 5), (1287, 5), (5147,), (1287,))
```

```
In [41]: reg = LinearRegression()
         reg.fit(X_train, Y_train)
         Y_test_pred = reg.predict(X_test)
         from sklearn import metrics
         print('Accuracy of test data:',reg.score(X_test, Y_test)*100)
         print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_test_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_test_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, Y_test_pred)))
         sns.scatterplot(Y_test_pred, Y_test)

         Accuracy of test data: 16.872680299024058
         Mean Absolute Error: 430045.12184525817
         Mean Squared Error: 270162159436.3094
         Root Mean Squared Error: 519771.25683930365

Out[41]: <AxesSubplot:ylabel='Weekly_Sales'>
```



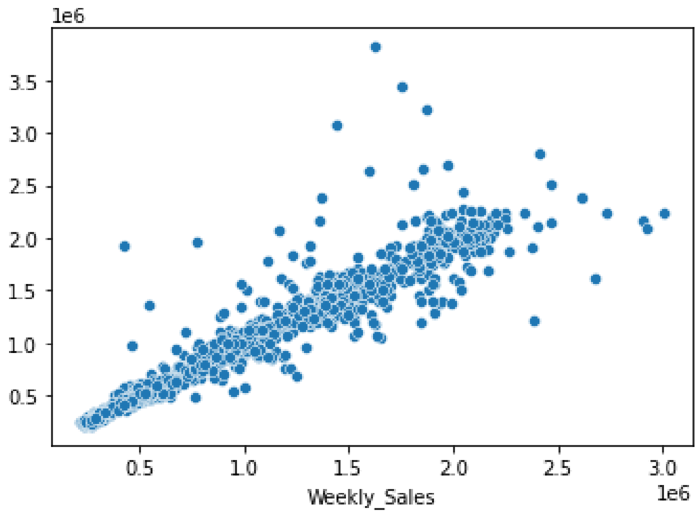
The accuracy is very less. So we cannot say this is a good model.

DECISION TREE MODEL:

In [43]:
```python
# Decision Tree
from sklearn.tree import DecisionTreeRegressor
X_train, X_test, y_train, y_test = train_test_split(X_std,Y,test_size=0.2)
model = DecisionTreeRegressor()
model.fit(X_train,y_train)
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
print("Accuracy of Train data:", model.score(X_train,y_train)*100)
print('Accuracy of test data:',model.score(X_test, y_test)*100)
print("Mean Absolute Error: ",metrics.mean_absolute_error(y_test,y_pred_test))
print("Mean Square Error: ",metrics.mean_squared_error(y_pred_test,y_test))
print("Root Mean Square Error: ",np.sqrt(metrics.mean_squared_error(y_pred_test,y_test)))
sns.scatterplot(y_test,y_pred_test)
```

```
Accuracy of Train data: 100.0
Accuracy of test data: 88.32363820734517
Mean Absolute Error:  92774.59034965035
Mean Square Error:  36454446134.767
Root Mean Square Error:  190930.47460991394
```

Out[43]: <AxesSubplot:xlabel='Weekly_Sales'>

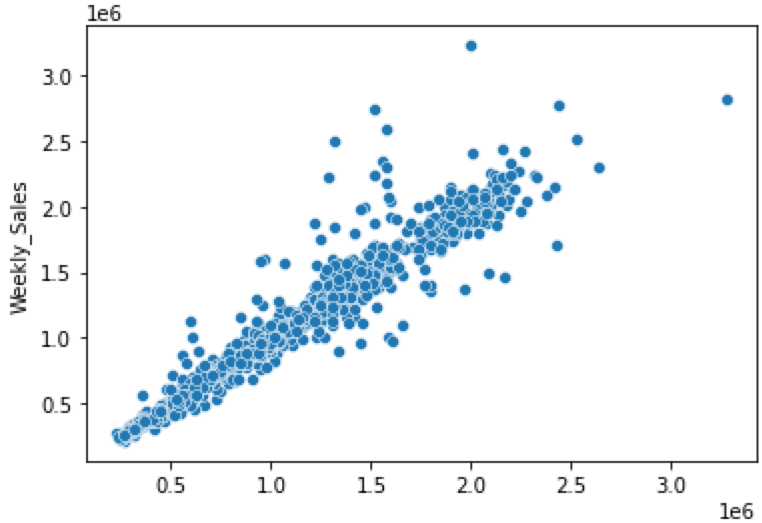


RANDOM FOREST MODEL:

In [89]:
```python
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
randomfrst = RandomForestRegressor()
randomfrst.fit(X_train,Y_train)
Y_pred = randomfrst.predict(X_test)
print('Accuracy of test data:',randomfrst.score(X_test, Y_test)*100)
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
sns.scatterplot(Y_pred, Y_test)
```

```
Accuracy of test data: 93.65615561560301
Mean Absolute Error: 72084.45893123545
Mean Squared Error: 19065726429.965282
Root Mean Squared Error: 138078.69651023392
```

Out[89]: <AxesSubplot:ylabel='Weekly_Sales'>



Here, Accuracy of Linear regression = 10%.

Decision Tree Model = 87%

Random Forest Model = 93%

So, Accuracy and error of Random Forest Regression Model is highest and best, So we will be choosing this model.

Model Evaluation and Technique

Model evaluation is a crucial step in machine learning to assess the performance and effectiveness of a predictive model. It involves using various techniques and metrics to measure how well the model generalizes to new, unseen data and how accurately it predicts the target variable. Here are model evaluation techniques and metrics used:

★ **Train-Test Split**: This technique involves splitting the available data into two subsets: a training set and a test set. The model is trained on the training set and then evaluated on the test set to assess its performance. The accuracy and error metrics on the test set provide an indication of how well the model is expected to perform on new, unseen data.

★ **Metrics for Regression Models:**

➢ Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual values. A lower MSE indicates better performance. We got the lowest MSE for Random Forest Model.

➢ Root Mean Squared Error (RMSE): The square root of MSE, which provides a measure of the average prediction error in the original units of the target variable. We got the lowest RMSE for Random Forest Model.

➢ Mean Absolute Error (MAE): Measures the average absolute difference between the predicted and actual values. It is less sensitive to outliers compared to MSE. We got the lowest MAE for Random Forest Model.

★ **Metrics for Classification Models:**

➢ Accuracy: Measures the proportion of correctly classified instances. It is the most commonly used metric for classification tasks.

```
Accuracy of test data: 94.06832847178373
Mean Absolute Error: 74680.64108453768
Mean Squared Error: 19567181988.359875
Root Mean Squared Error: 139882.74371186702
```

★ **Predicting Values for future Sale per Store:**

To predict Future Sales I have used Time Series model. Checking whether the data is stationary or not.

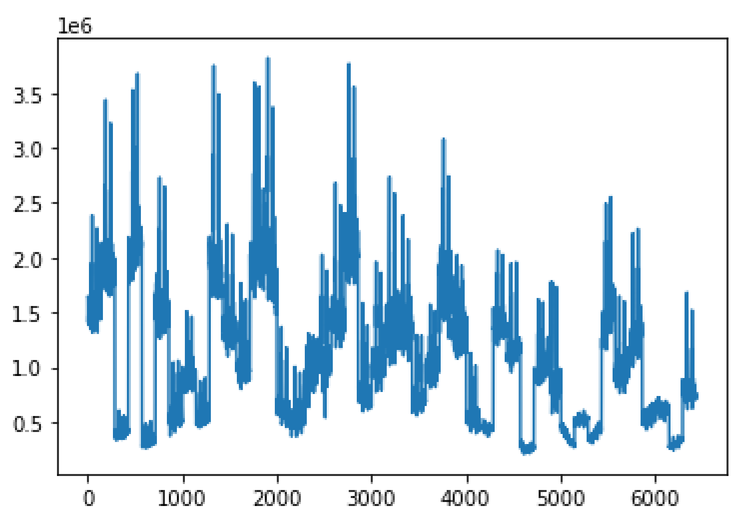
In [54]:
```
from statsmodels.tsa.stattools import adfuller
sales_data = walmart_data['Weekly_Sales']
result = adfuller(sales_data)
p_value = result[1]
print("P_value: ",p_value)
if p_value < 0.05:
    print("The data is stationary.")
else:
    print("The data is not stationary.")
```

```
P_value:  0.00011613258802178222
The data is stationary.
```

To check if the data is Seasonal or not:

In [51]:
```
walmart_data['Weekly_Sales'].plot()
# This is seasonal data
```

Out[51]: <AxesSubplot:>



Here since the plot is symmetric and repeating itself so we can conclude that our data is seasonal. Since our data is symmetric and stationary then we can directly use SARIMA Time Series model to predict the sales for next 12 months.

```
In [62]: import warnings

         with warnings.catch_warnings():
             warnings.filterwarnings("ignore", category=DeprecationWarning)

         import statsmodels.api as sm

         unique_stores = walmart_data['Store'].unique()

         order = (1, 1, 1)
         seasonal_order = (1, 1, 1, 12)

         for store_id in unique_stores:
             store_data = walmart_data[walmart_data['Store'] == store_id]
             model = sm.tsa.statespace.SARIMAX(store_data['Weekly_Sales'], order=order, seasonal_order=seasonal_order)
             model_fit = model.fit()
             forecast = model_fit.forecast(steps=12)
             print(f"Store ID: {store_id}")
             print(forecast)
             print("--------------------")
```

For the above code, we got the forecast store wise for next 12 months.

```
Store ID: 1
143    1.421157e+06
144    1.511237e+06
145    1.498984e+06
146    1.571846e+06
147    1.510895e+06
148    1.472766e+06
149    1.497334e+06
150    1.504977e+06
151    1.498639e+06
152    1.497632e+06
153    1.511394e+06
154    1.588019e+06
Name: predicted_mean, dtype: float64
--------------------
```

```
Store ID: 2
143    1.709647e+06
144    1.818715e+06
145    1.833277e+06
146    1.946534e+06
147    1.790230e+06
148    1.757251e+06
149    1.805713e+06
150    1.806947e+06
151    1.796017e+06
152    1.812438e+06
153    1.827277e+06
154    1.938755e+06
Name: predicted_mean, dtype: float64
--------------------
```

```
Store ID: 3
143    388216.857328
144    404162.782058
145    412566.886013
146    413998.263833
147    404024.811650
148    397529.323935
149    410445.089803
150    399799.721293
151    415801.653782
152    408450.567700
153    416225.871699
154    425200.567853
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 4
143    2.041562e+06
144    2.144343e+06
145    2.151268e+06
146    2.221072e+06
147    2.092061e+06
148    2.075226e+06
149    2.117922e+06
150    2.108841e+06
151    2.102557e+06
152    2.120777e+06
153    2.127436e+06
154    2.254843e+06
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 5
143    301629.058840
144    316485.721285
145    322151.589739
146    332790.165539
147    324724.096373
148    305834.757204
149    314847.238615
150    322344.244309
151    322412.734600
152    319636.809249
153    319885.382386
154    337477.321497
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 6
143    1.326060e+06
144    1.393026e+06
145    1.420462e+06
146    1.514057e+06
147    1.404896e+06
148    1.321009e+06
149    1.373889e+06
150    1.398986e+06
151    1.343666e+06
152    1.362866e+06
153    1.382155e+06
154    1.474965e+06
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 7
142    483026.421967
143    476887.523339
144    505055.258846
145    558702.924950
146    524734.147861
147    499909.135404
148    499370.220851
149    502102.623655
150    474289.182954
151    468152.625244
152    472499.611932
153    517706.128679
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 8
143    856652.166968
144    888352.748699
145    914389.420730
146    953022.732729
147    885269.091547
148    872350.628913
149    883006.468432
150    898874.555100
151    888383.687663
152    897320.218744
153    904271.287712
154    948172.661624
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 9
143    517610.484030
144    538699.155897
145    560763.164998
146    577024.023170
147    541794.325176
148    532093.759780
149    542536.372482
150    545777.050051
151    550092.456763
152    546778.369030
153    548325.312332
154    585263.154930
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 10
143    1.589181e+06
144    1.684137e+06
145    1.684609e+06
146    1.746737e+06
147    1.630742e+06
148    1.583527e+06
149    1.626618e+06
150    1.662555e+06
151    1.620476e+06
152    1.668422e+06
153    1.696395e+06
154    1.820696e+06
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 11
143    1.141184e+06
144    1.228769e+06
145    1.263735e+06
146    1.309240e+06
147    1.206687e+06
148    1.193993e+06
149    1.228307e+06
150    1.235919e+06
151    1.223171e+06
152    1.220821e+06
153    1.221971e+06
154    1.279151e+06
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 43
143    605823.526953
144    642219.939020
145    630465.179414
146    632010.146259
147    643462.345616
148    636426.551157
149    625476.926051
150    627410.091171
151    641311.989911
152    631060.586234
153    635819.639759
154    635610.194316
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 44
143    343302.480489
144    357312.901781
145    362221.014641
146    366679.858277
147    357964.403436
148    367574.672280
149    362833.176926
150    373710.585780
151    362136.162249
152    365218.358786
153    358537.578101
154    379721.270066
Name: predicted_mean, dtype: float64
---------------------
```

```
Store ID: 45
143    688334.902057
144    722057.332235
145    736361.133701
146    782560.025048
147    727876.662515
148    702308.183734
149    713208.502955
150    735743.965671
151    696949.241392
152    711343.324672
153    727171.451288
154    786707.313401
Name: predicted_mean, dtype: float64
---------------------
```

Similarly we got the output for all 45 stores. So, we have successfully predicted the futures sales for next 12 months for each store respectively.

Inferences

Based on the Walmart project, here are some possible inferences and insights that can be drawn from the analysis:

- **Seasonality**: The sales data from Walmart exhibits a clear pattern of seasonality, with higher sales during specific periods of the year. This can be further analyzed to identify the peak seasons and plan inventory, promotions, and staffing accordingly.

- **Store Performance**: The analysis of individual store sales data can provide insights into the performance of each store. Identifying stores with consistently high or low sales can help in understanding the factors contributing to their performance and implementing strategies to improve sales.

- **Promotions and Discounts**: Analyzing the impact of promotions and discounts on sales can help identify effective marketing strategies. By studying the correlation between promotional activities and sales spikes, Walmart can optimize its promotional campaigns and maximize their impact on sales.

- **Pricing Optimization**: Analyzing the relationship between pricing and sales can help identify optimal price points for different products or categories. This analysis can assist in pricing strategies, discount planning, and competitive positioning.

- **Store Layout and Placement:** Analyzing sales data along with store layout and product placement information can reveal insights into the impact of store design on customer behavior. It can help optimize store layouts to enhance customer experience and maximize sales.

- **Forecasting**: By applying time series analysis techniques and predictive models to the sales data, Walmart can forecast future sales and demand. This information can support inventory planning, production scheduling, and supply chain management.

Future Possibilities

The future possibilities for Walmart sales prediction are vast, considering the advancements in data analytics, machine learning, and technology. Here are some potential future possibilities for Walmart sales prediction:

- **Advanced Forecasting Models**: Develop more sophisticated forecasting models using advanced techniques such as deep learning, ensemble methods, or hybrid models that combine multiple algorithms. These models can provide more accurate and granular sales predictions at different levels, including individual stores, product categories, and geographical regions.
- **Real-Time Sales Prediction**: Implement real-time sales prediction systems that continuously analyze incoming data and provide up-to-date sales forecasts. This would enable Walmart to make timely decisions, such as inventory management, staffing adjustments, and pricing strategies, based on the most current sales predictions.
- **Integration of External Data:** Incorporate external data sources, such as social media trends, economic indicators, competitor data, or weather patterns, into the sales prediction models. By analyzing the impact of these external factors on sales, Walmart can gain deeper insights and make more informed business decisions.
- **Demand Sensing**: Use advanced demand sensing techniques to capture and analyze real-time customer demand signals from various sources, including point-of-sale data, online sales, customer reviews, and social media. By leveraging demand sensing capabilities, Walmart can respond quickly to changes in customer preferences, optimize inventory levels, and enhance overall supply chain efficiency.
- **Personalized Sales Prediction:** Develop personalized sales prediction models that consider individual customer behavior, preferences, and purchasing history. By tailoring sales predictions to each customer, Walmart can provide personalized offers, recommendations, and promotions, leading to increased customer satisfaction and loyalty.

- **Integration with IoT and Sensor Data:** Leverage Internet of Things (IoT) devices and sensor data within stores to capture and analyze real-time data on foot traffic, product placements, and customer interactions. This data can be used to enhance sales prediction models, optimize store layouts, improve product placements, and enhance the overall in-store customer experience.
- **Geo-Spatial Analysis:** Apply geo-spatial analysis techniques to understand regional variations in sales patterns. By incorporating geographical factors such as population density, demographics, and local market dynamics into sales prediction models, Walmart can better target specific regions with tailored marketing strategies and optimize product assortments based on regional preferences.
- **Integration with E-commerce:** Integrate sales prediction models with e-commerce platforms to provide accurate sales forecasts for online sales channels. This would enable Walmart to optimize online inventory, improve order fulfillment, and enhance the overall online shopping experience for customers.

These are just a few potential future possibilities for Walmart sales prediction. As technology continues to evolve and new data analysis techniques emerge, Walmart can leverage these advancements to further enhance its sales prediction capabilities and stay at the forefront of retail analytics.

References

1. https://chat.openai.com/
2. https://www.walmart.com/
3. https://www.kaggle.com/
4. https://towardsdatascience.com
5. https://www.analyticsvidhya.com
6. https://github.com/
7. https://scikit-learn.org/
8. https://plotly.com/
9. https://matplotlib.org/