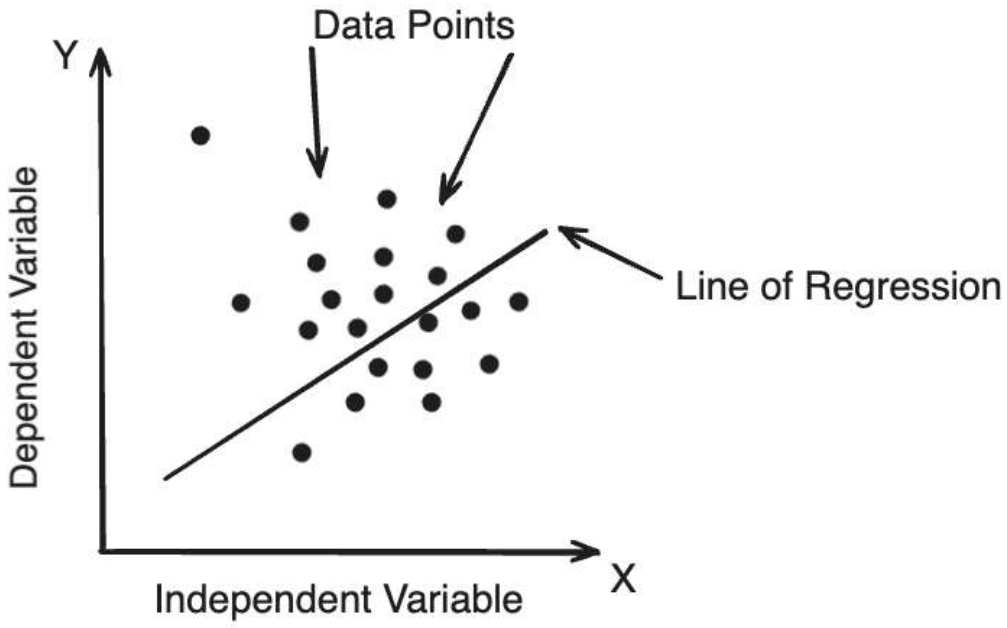
Walmart Sales On Linear Regression

About Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a straight line to the observed data points. It helps analyze and quantify the association between variables, enabling predictions and understanding of trends in data.

Think of linear regression like drawing the best straight line through a bunch of dots on a graph. It helps us see the relationship between two things, like how test scores change as study hours increase. It's a way to predict one thing based on another, using a simple straight line.



3 Main reasons why we use linear Regression-

Simple Interpretation: It provides a straightforward interpretation of the relationship between variables. With a linear equation, it's easy to understand how a change in one variable affects another.

Prediction: Linear regression allows us to predict the value of the dependent variable based on the values of the independent variables. This predictive capability is valuable in various fields for making informed decisions.

Understanding Relationships: It helps us understand the relationship between variables by quantifying how they are associated. This understanding is crucial for making hypotheses, testing theories, and drawing conclusions.

When we should Linear Regression

Linear Relationships: When the relationship between the dependent variable and the independent variable(s) appears to be linear. If you can visually observe a linear trend in your data, linear regression may be appropriate.

Interpretability: When you need a simple and interpretable model that provides insights into the relationship between variables. Linear regression coefficients represent the magnitude and direction of the relationship, making it easy to understand.

Predictive Modeling: When you want to make predictions based on continuous variables. Linear regression provides a straightforward framework for predicting values within the range of the independent variables, making it useful in forecasting and regression analysis.

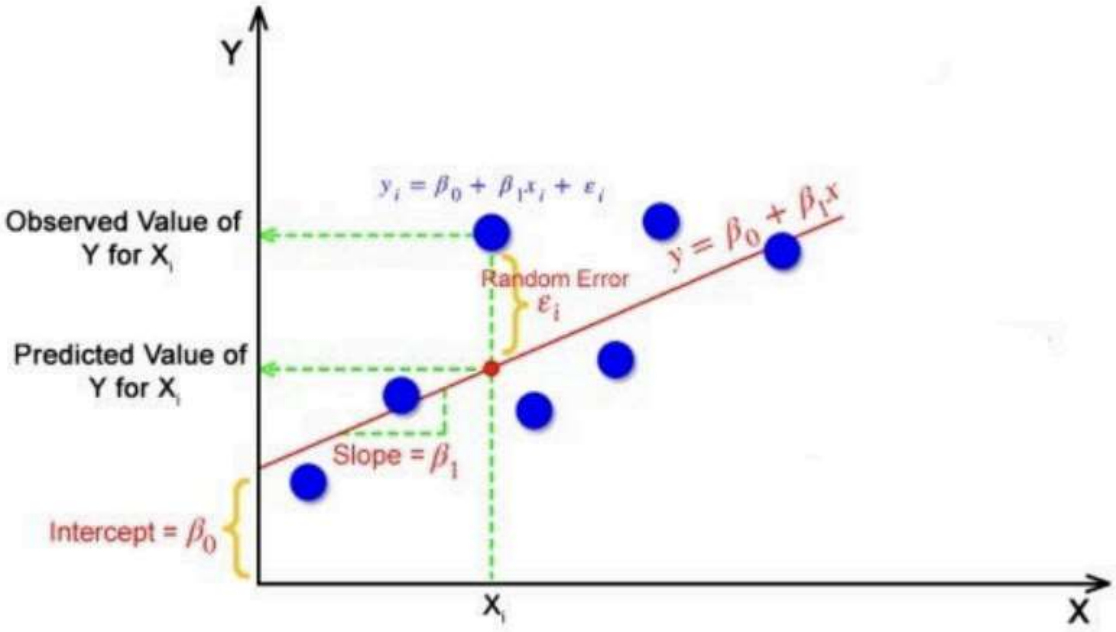
Statistical Analysis for Linear Regression

To calculate best-fit line linear regression uses a traditional slope-intercept form which is given below,

$Yi = \beta 0 + \beta 1 Xi$

where Yi = Dependent variable, β0 = constant/Intercept, β1 = Slope/Intercept, Xi = Independent variable.

This algorithm explains the linear relationship between the dependent(output) variable y and the independent(predictor) variable X using a straight line Y= B0 + B1 X.



But how the linear regression finds out which is the best fit line?

The goal of the linear regression algorithm is to get the best values for B0 and B1 to find the best fit line. The best fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.

Types of Linear regression

Simple Linear Regression

Multiple Linear Regression

Polynomial Regression

Ridge Regression

Lasso Regression

ElasticNet Regression

Quantile Regression

What is best fit lines

The best fit line is a line that fits the given scatter plot in the best way. Mathematically, the best fit line is obtained by minimizing the Residual Sum of Squares(RSS

Steps to implement Linear Regression

Import Libraries: Import necessary libraries such as NumPy, Pandas, and scikit-learn.

Load Data: Load your dataset into a Pandas DataFrame.

Split Data: Split the dataset into training and testing sets using train_test_split from scikit-learn.

Instantiate Model: Create an instance of the linear regression model from scikit-learn.

Fit Model: Fit the model to the training data using the fit method.

Make Predictions: Use the trained model to make predictions on the test data using the predict method.

Evaluate Model: Evaluate the model's performance using evaluation metrics such as mean squared error or R-squared.

Visualize Results: Optionally, visualize the model's predictions compared to the actual values using plots or charts

Loss function Cost function

A loss function, also known as a cost function or error function, is a mathematical function that measures the difference between the actual values of the dependent variable and the values predicted by a model. In the context of linear regression, the loss function quantifies how well the model's predictions match the observed data. The goal is to minimize this difference, indicating a better fit of the model to the data.

In linear regression, the most commonly used loss function is the Mean Squared Error (MSE) or its variants. The MSE is calculated by taking the average of the squared differences between the actual and predicted values for all data points in the dataset

Types of Cost Functions:

Mean Squared Error (MSE):

Calculates the average of the squared differences between the actual and predicted values.

Root Mean Squared Error (RMSE):

The square root of the MSE, providing an interpretable measure of the average deviation in the same units as the dependent variable.

Mean Absolute Error (MAE):

Calculates the average of the absolute differences between the actual and predicted values.

Evaluation Metrics

Evaluation metrics for linear regression assess the performance of the model in predicting the dependent variable based on the independent variables

Difference between Evalutaion Metrics and Cost Function

Evaluation metrics and cost functions serve different purposes in the context of machine learning models, although they are related. Here's the difference:

Evaluation Metrics:

Evaluation metrics are used to assess the performance of a machine learning model. They provide a quantitative measure of how well the model is performing on a particular task or dataset.

Evaluation metrics are typically chosen based on the specific objectives and requirements of the problem at hand. For example, in a regression problem, common evaluation metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared.

Evaluation metrics help practitioners understand how well a model generalizes to new, unseen data and compare the performance of different models.

Cost Functions:

Cost functions, also known as loss functions or objective functions, are used during the training process of a machine learning model to quantify how well the model is performing on the training data.

Cost functions measure the difference between the predicted values of the model and the actual values in the training data. The goal during training is to minimize this difference, i.e., minimize the cost function.

The choice of cost function depends on the type of machine learning task (e.g., regression, classification) and the specific requirements of the problem. For example, in linear regression, the cost function is often the Mean Squared Error (MSE).

Application of Simple Linear Regression

Sales Forecasting:

Simple linear regression can be used to predict sales based on a single predictor variable, such as advertising spending. For example, a company may use historical data on advertising expenditures and corresponding sales figures to develop a linear regression model. This model can then be used to forecast future sales based on planned advertising budgets.

GPA Prediction:

In education, simple linear regression can be used to predict a student's GPA based on a single predictor variable, such as the number of hours spent studying per week. By analyzing past student data, a university or educational institution can develop a linear regression model to understand the relationship between study hours and GPA. This model can then be used to predict the GPA of future students based on their study habits.

About Dataset

The dataset Walmart Sales contains information about weekly sales in different stores along with various other features. Here's a brief description of the dataset:

Store: The store number.

Date: The date of the sales data.

Weekly_Sales: The total sales for the week.

Holiday_Flag: A binary flag indicating whether the week includes a holiday (1) or not (0).

Temperature: The temperature on the date of sales.

Fuel_Price: The fuel price on the date of sales.

CPI: The Consumer Price Index on the date of sales.

Unemployment: The unemployment rate on the date of sales.

The code we performed demonstrates building a simple linear regression model using the given dataset. Here's a summary of the steps:

Load the Data: The dataset is loaded into a pandas DataFrame.

Prepare the Data: Features (independent variables) and the target variable (Weekly_Sales) are selected from the dataset.

Split the Data: The dataset is split into training and testing sets using train_test_split().

Build the Model: A simple linear regression model is created using LinearRegression() from scikit-learn.

Train the Model: The model is trained on the training data using the fit() method.

Make Predictions: Predictions are made on the testing data using the predict() method.

Evaluate the Model: Model performance is evaluated using mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score.

Visualize the Results: Actual vs predicted values are plotted to visualize the performance of the model.

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
```

Load Data

In [3]:
```
df = pd.read_csv('Walmart Sales Data.csv')
```

Reading data

In [4]:
```
df
```

Out[4]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...	...	...
6430	45	28-09-2012	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	05-10-2012	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	12-10-2012	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	19-10-2012	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	26-10-2012	760281.43	0	58.85	3.882	192.308899	8.667

6435 rows × 8 columns

In [5]:
```
df.head(5)
```

Out[5]:

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

In [6]:
```
df.tail(5)
```

Out[6]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
6430	45	28-09-2012	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	05-10-2012	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	12-10-2012	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	19-10-2012	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	26-10-2012	760281.43	0	58.85	3.882	192.308899	8.667

Data pre-processing and EDA

Info about data

In [7]:
```
df.info() # info about data
```

```
<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
```

In [8]:
```
df.describe() #statistical description of data
```

Out[8]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployme
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.0000
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.9991
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.8758
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.8790
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.8910
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.8740
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.6220
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.3130

Checking null values

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

Object to float

In [11]:
```
df['Date'] = pd.to_numeric(df['Date'], errors='coerce')
```

In [14]:
```
# Convert date column to numerical representation (Unix timestamp in seconds)
df['Date'] = pd.to_datetime(df['Date']).view('int64') // 10**9
```

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Date          6435 non-null   int64  
 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(3)
memory usage: 402.3 KB
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Store', 'Date', 'Weekly_Sales', 'Holiday_Flag', 'Temperature',
       'Fuel_Price', 'CPI', 'Unemployment'],
      dtype='object')
```

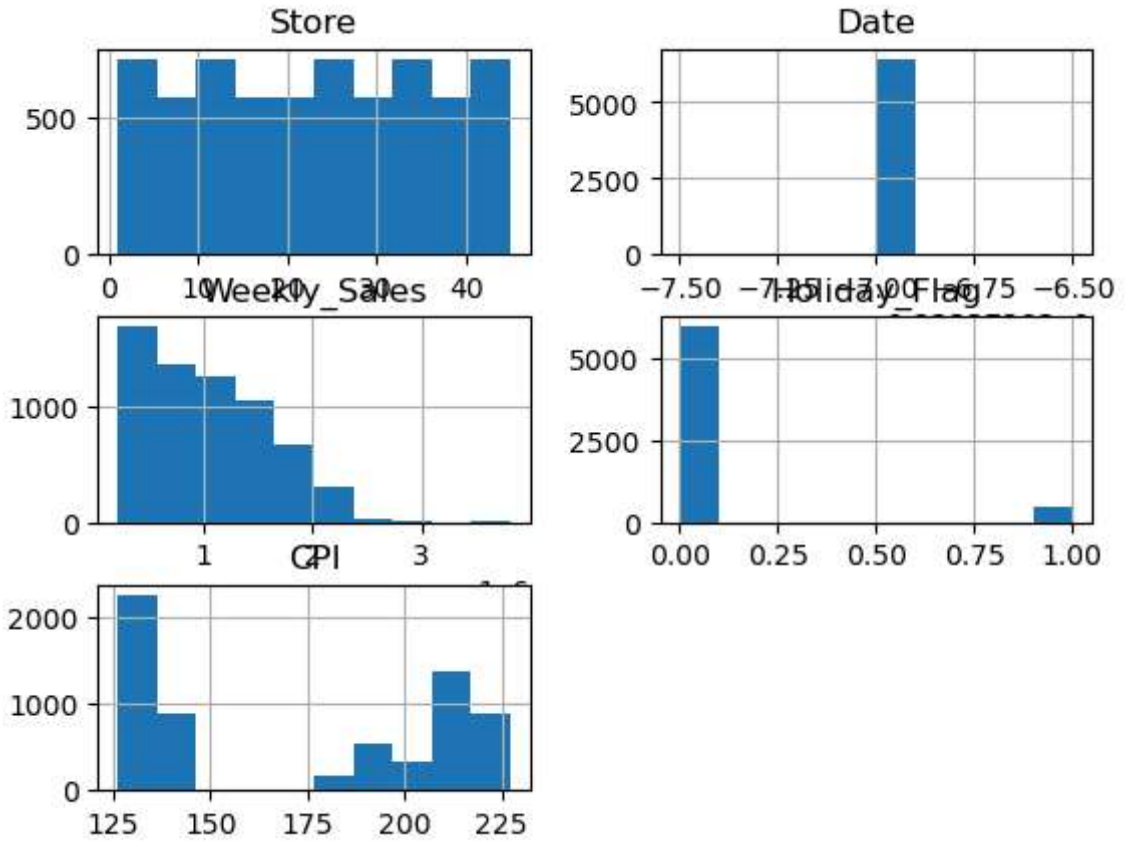
In [17]:
```
df.columns
ndf=df[['Store','Date','Weekly_Sales','Holiday_Flag','CPI']]
ndf.head()
```

Out[17]:

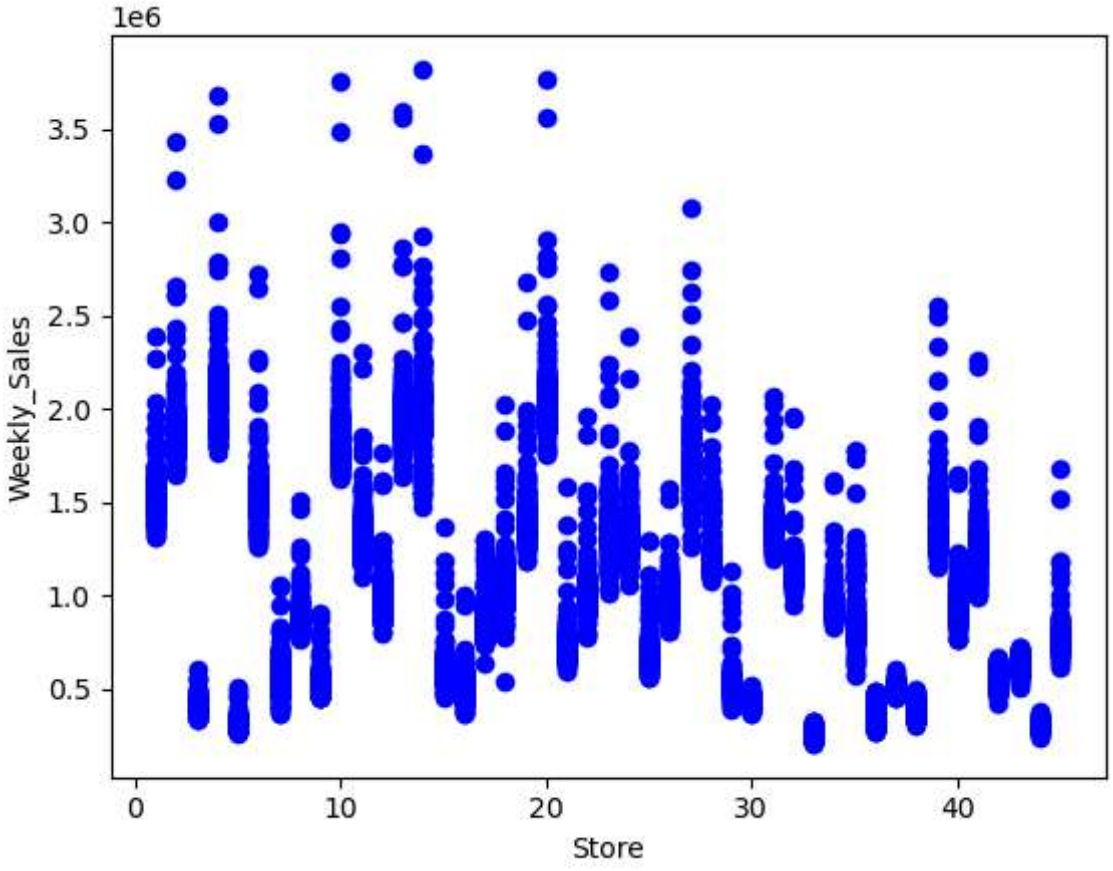
	Store	Date	Weekly_Sales	Holiday_Flag	CPI
0	1	-9223372037	1643690.90	0	211.096358
1	1	-9223372037	1641957.44	1	211.242170
2	1	-9223372037	1611968.17	0	211.289143
3	1	-9223372037	1409727.59	0	211.319643
4	1	-9223372037	1554806.68	0	211.350143

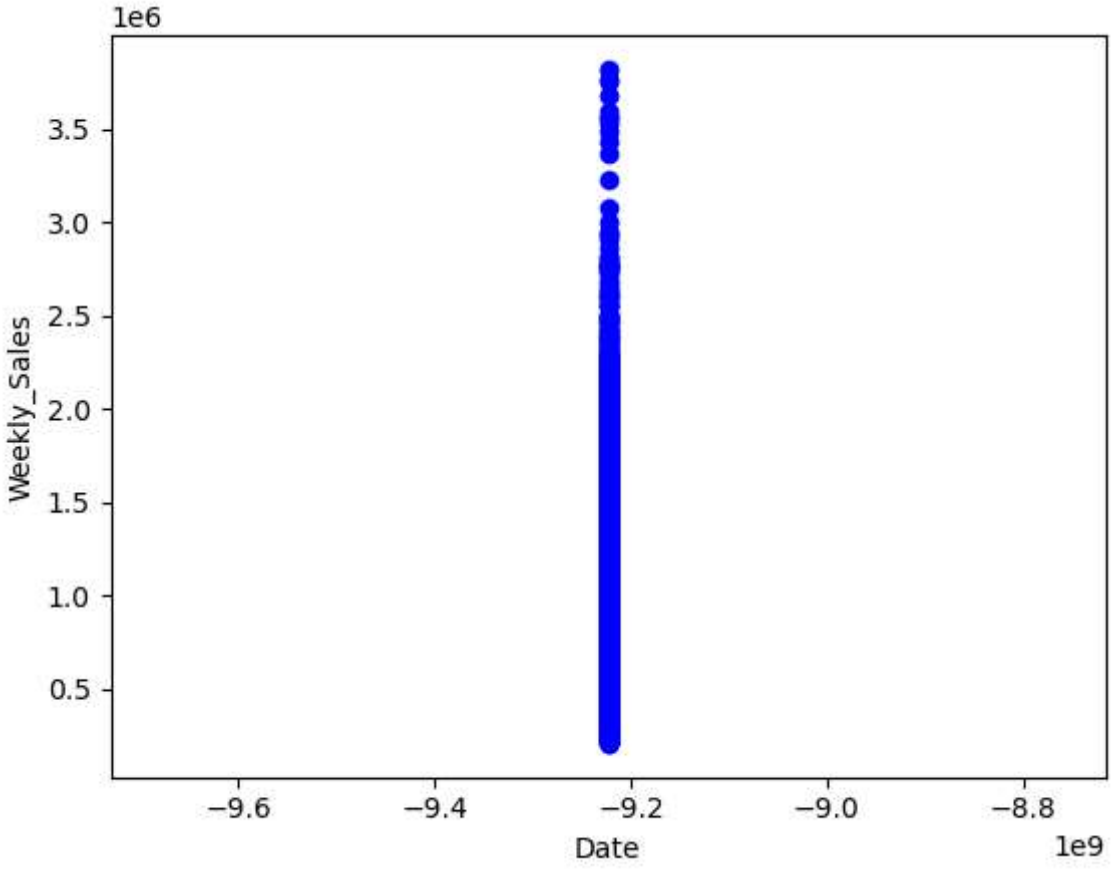
Visualization

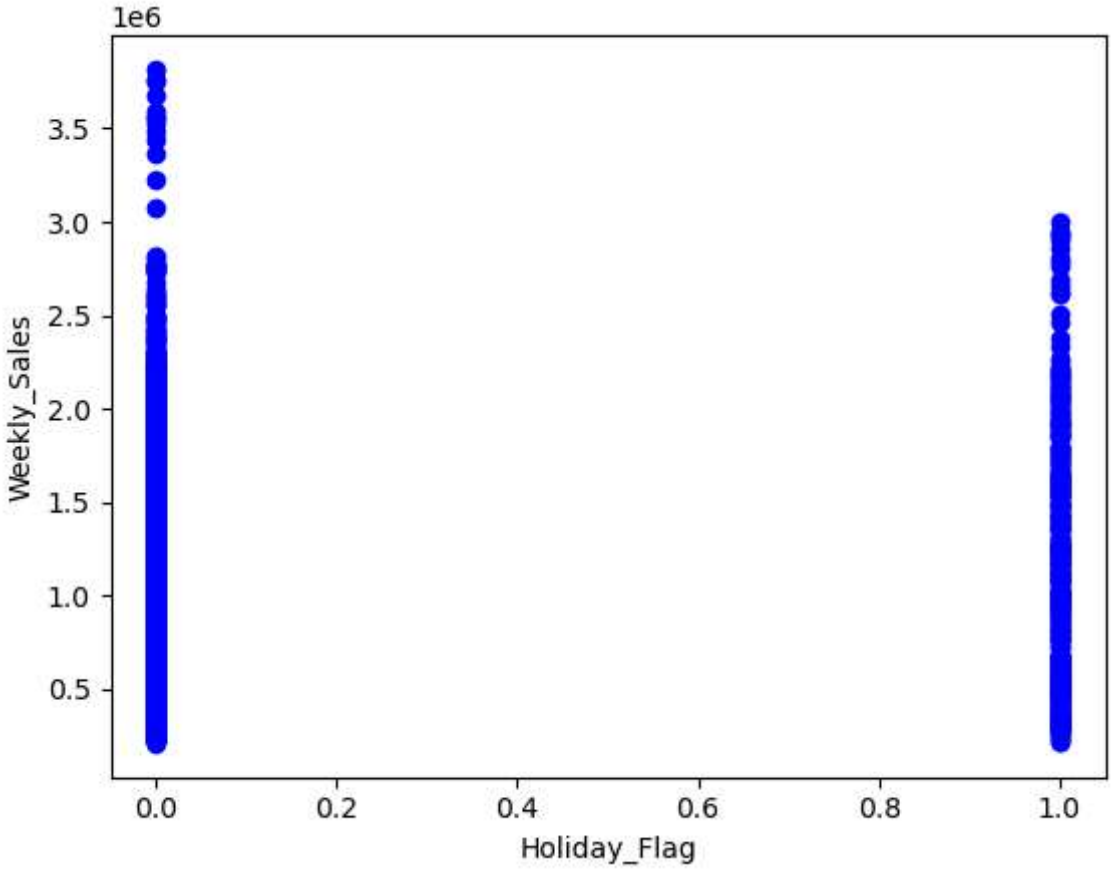
In [18]:
```
#histogram
viz = ndf[['Store','Date','Weekly_Sales','Holiday_Flag','CPI']]
viz.hist()
plt.show()
```

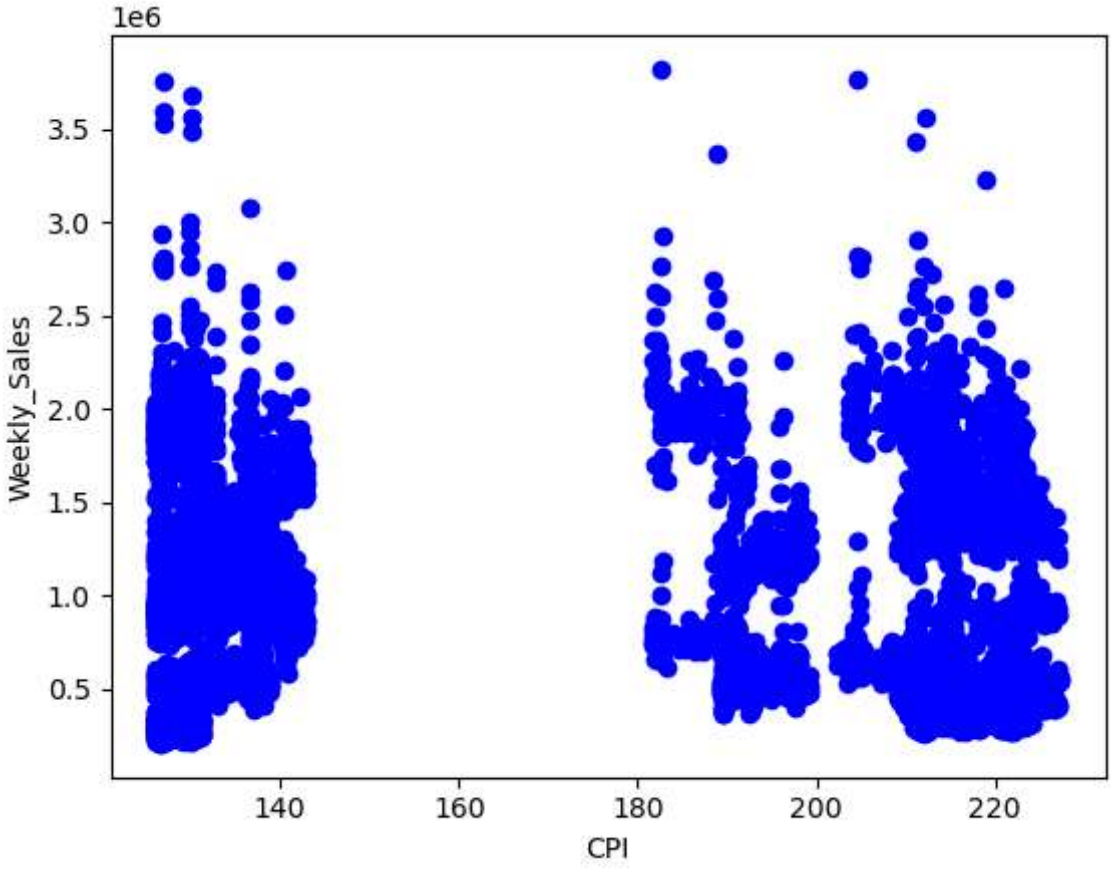


In [19]:
```
#scatter plot
for i in ndf[['Store','Date','Holiday_Flag','CPI']]:
    plt.scatter(ndf[i],ndf['Weekly_Sales'],color='blue')
    plt.xlabel(i)
    plt.ylabel("Weekly_Sales")
    plt.show()
```


1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Weekly_Sales
0
10
20
30
40
Store

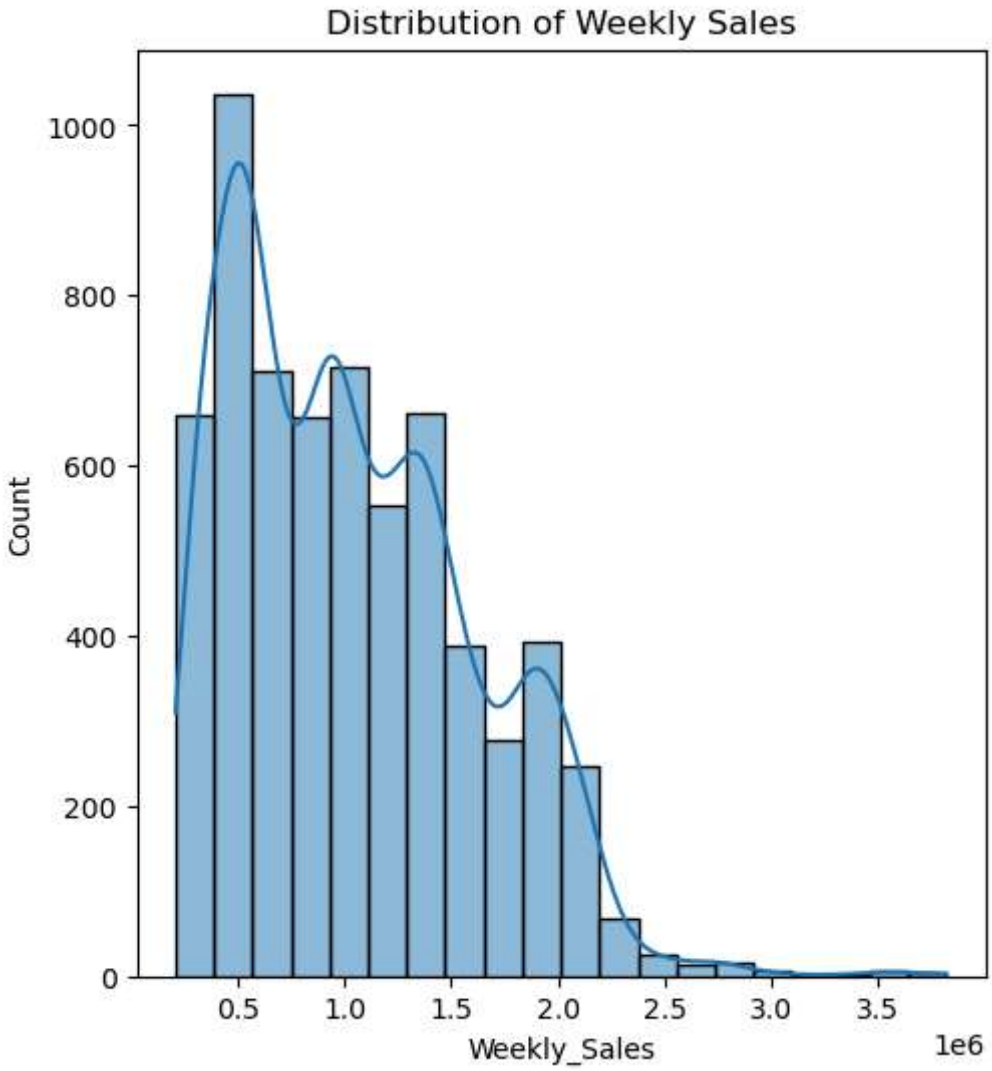

1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Weekly_Sales
−9.6
−9.4
−9.2
−9.0
−8.8
Date
1e9





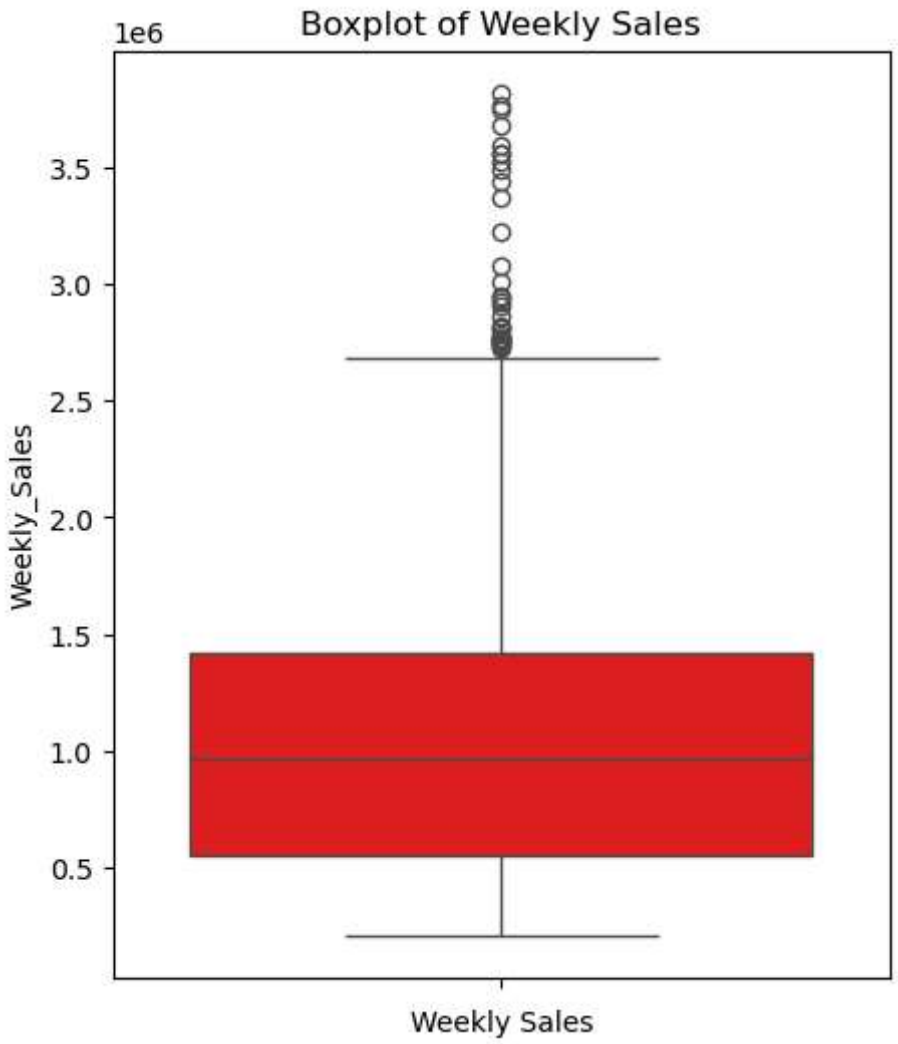
In [20]:
```
#histgram
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
```

```
sns.histplot(ndf['Weekly_Sales'], bins=20, kde=True)
plt.title('Distribution of Weekly Sales');
```



In [21]:
```
#boxplot
plt.figure(figsize=(5, 6))
sns.boxplot(ndf['Weekly_Sales'], color='red')
plt.title('Boxplot of Weekly Sales')
plt.xlabel('Weekly Sales')
```

Out[21]:
```
Text(0.5, 0, 'Weekly Sales')
```



Prepare the data

In [23]:
```
X = df[['Date','Store', 'Fuel_Price','CPI']]  # Features
y = df['Weekly_Sales']
```

Split the data into training and testing sets

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

Build Model

In [25]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[25]:
```
LinearRegression
LinearRegression()
```

Make predictions

In [26]:
```
y_pred = model.predict(X_test)
```

Evaluate the mode

In [28]:
```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 275788220815.3611
```

In [29]:
```
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Root Mean Squared Error (RMSE): 525155.4253888662
C:\Users\chira\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Future
Warning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calcul
ate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

In [30]:
```
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Absolute Error (MAE): 434617.00260962656
```

In [31]:
```
r2 = r2_score(y_test, y_pred)
print("R-squared (R2 Score):", r2)
```

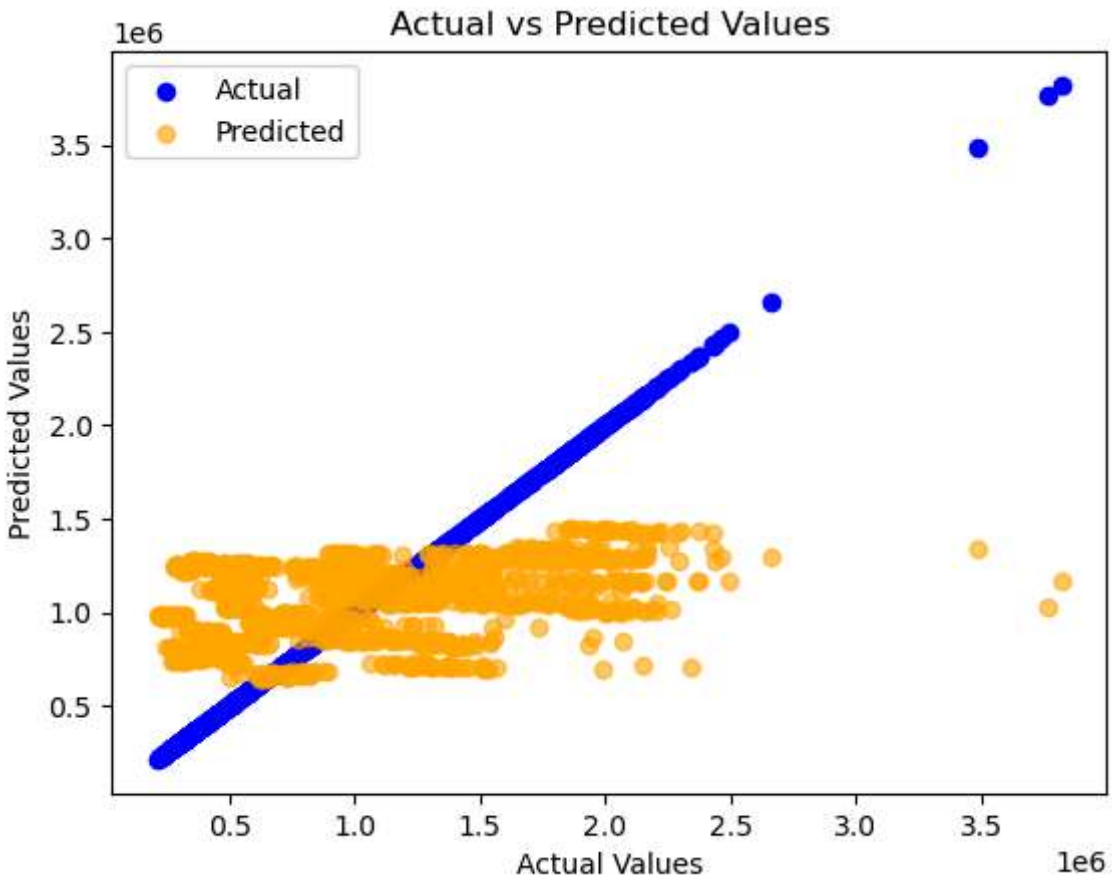
```
R-squared (R2 Score): 0.14392577013796892
```

In [32]:
```
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [     0.         -15707.73626386  -5038.60137214  -2077.47983658]
Intercept: 1781601.1174337873
```

Visualize the results

In [34]:
```
plt.scatter(y_test, y_test, color='blue', label='Actual')
plt.scatter(y_test, y_pred, color='orange', label='Predicted', alpha=0.6)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.legend()
plt.show()
```


Actual vs Predicted Values
1e6
Actual
Predicted
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Predicted Values
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Actual Values
1e6

In [ ]: