WALMART SALES PREDICTION

INTRODUCTION

Walmart, one of the top retailers in the US, would want to make accurate sales and demand predictions. Each day, specific occasions and holidays have an influence on sales. Data on sales are available for 45 Walmart locations. Unforeseen demand is a hurdle for the company, and occasionally supply runs out because of a bad machine learning system. A perfect machine learning algorithm will precisely estimate demand and take into account variables like the CPI, unemployment rate, and other economic indicators.

Every year, Walmart holds a number of promotional discount sales. The Super Bowl, Labour Day, Thanksgiving, and Christmas are the four biggest holidays that are preceded by these markdowns. In comparison to non-holiday weeks, the evaluation of the weeks that include certain holidays is weighted five times more heavily. Modelling the impact of markdowns on these holiday periods in the absence of complete/ideal historical data is one of the challenges provided by this competition. For 45 Walmart locations spread across several geographies, historical sales information is available.

Dataset

Store - The store number

Date - The week of sales

Weekly_Sales - sales for the given store

Holiday_Flag - whether the week is a special holiday week 1 -

Holiday week 0 - non-holiday week

Temperature - Temperature on the day of sale

Fuel_Price - Cost of fuel in the region

CPI - Prevailing consumer price index

Unemployment - Prevailing unemplyment rate

Libraries

In [81]:
```
import numpy as np # Importing the NumPy for numerical computing
import pandas as pd # Importing the Pandas for data manipulation and analysis
import matplotlib.pyplot as plt  # Importing the pyplot module for creating stat
import seaborn as sns # Importing the Seaborn provides a high-level interface fo
```

```
sns.set() # setting the default style
import plotly.express as px # Importing the plotly express that allows you to cr

# Ignore warnings
import warnings
warnings.filterwarnings('ignore') # Ignore all warnings that may occur

import os # Importing for operating system-related functions
for dirname, _, filenames in os.walk('Desktop/Jupyter Notebook'): # walking thro
    for filename in filenames:   # Looping over the filenames
        print(os.path.join(dirname, filename)) # Printing the absolute path of e
```

Data Exploration

In [83]:
```
Walmart_ds = pd.read_csv('C:/Users/Admin/OneDrive/Desktop/Jupyter Notebook/Walma
#read the file from the folder
```

In [84]:
```
Walmart_ds.head() # display the first few rows
```

Out[84]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployme
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.1
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.1
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.1
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.1
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.1

In [85]:
```
Walmart_ds.info() # get a concise summary of the information
```

```
<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 [86]:
```
Walmart_ds.isna().sum() # provides information about the missing values
```

Out[86]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

In [87]:
```
Walmart_ds.duplicated().sum() # the data has no duplicates or missing values.
```

Out[87]:
```
0
```

In [88]:
```
Walmart_ds.describe() # generates descriptive statistcs of the numerical columns
```

Out[88]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unem
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	64
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	

In [89]:
```
# returns the status based on the month value
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'
```

In [90]:
```
#converts date column to datetime datatype
Walmart_ds['Date'] = pd.to_datetime(df['Date'],dayfirst=True)
```

In [91]:
```
# Now let's extract Month and Year from Date column
Walmart_ds['Month'] = Walmart_ds['Date'].dt.month
Walmart_ds['Year'] = Walmart_ds['Date'].dt.year
```

In [92]:
```
# Here we extract Season from Month column
Walmart_ds['Season'] = Walmart_ds['Month'].apply(get_season)
```

In [93]:
```
# three randomly selected rows
```

```
Walmart_ds.sample(3)
```

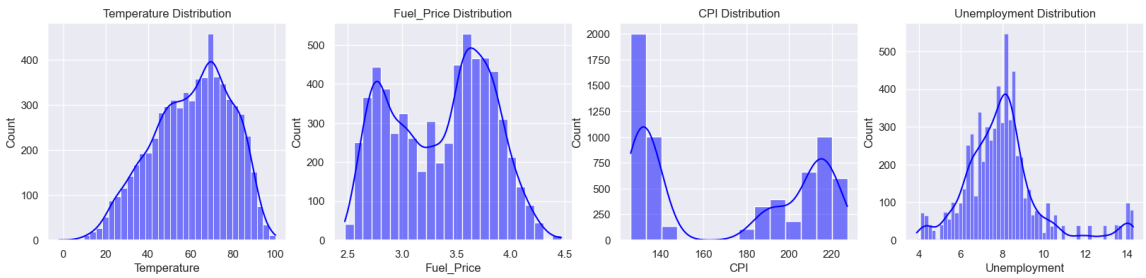
Out[93]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemplo
1124	8	2012-06-15	916918.70	0	77.14	3.393	225.313474	
2703	19	2012-07-27	1248915.43	0	74.43	3.820	138.203387	
4118	29	2012-04-13	520493.83	0	49.89	4.025	137.868000	

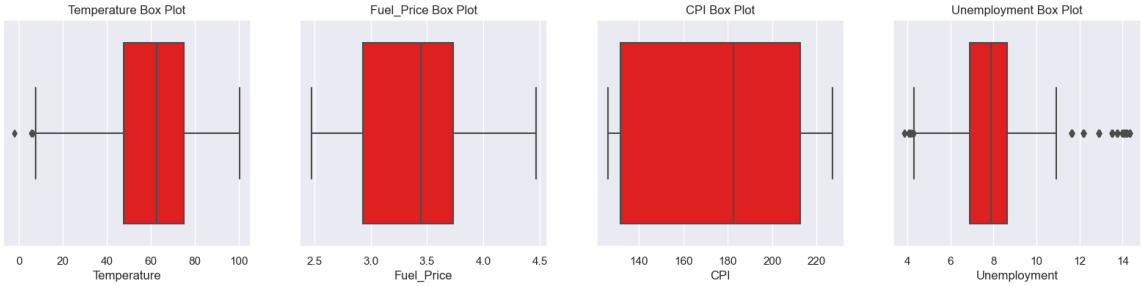
Exploratory Data Analysis

Univariate Analysis

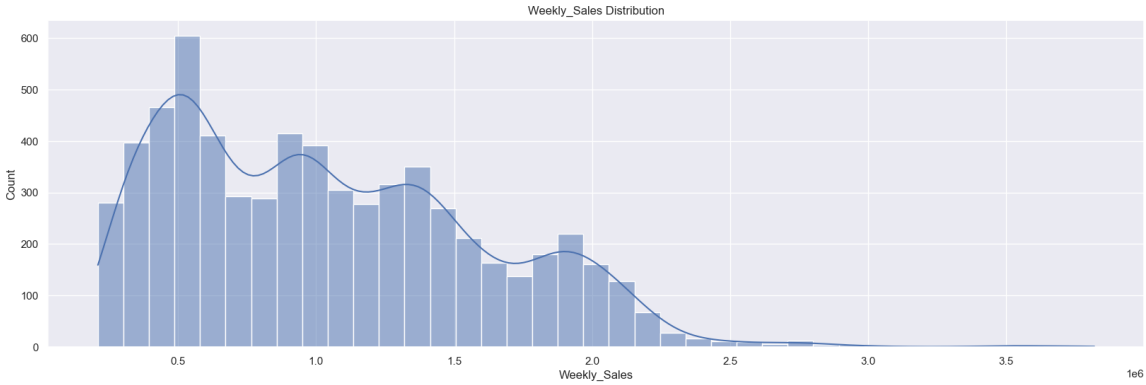
In [99]:
```
# CPI and Fuel_Price have bimodal distribution
# Temperature and Unemployment have normal distribution
# Four subplots are arranged in a single row
fig , ax = plt.subplots(1,4,figsize=(20,4))
cols = ['Temperature','Fuel_Price','CPI','Unemployment']
for i,col in enumerate(cols):
    # creates histogram with a kernel density estimate(KDE)
    sns.histplot(Walmart_ds,
                 x=col,
                 ax=ax[i],
                 kde=True,
                 color = 'blue'
                )
    ax[i].set_title(f'{col} Distribution')
fig.show()
```



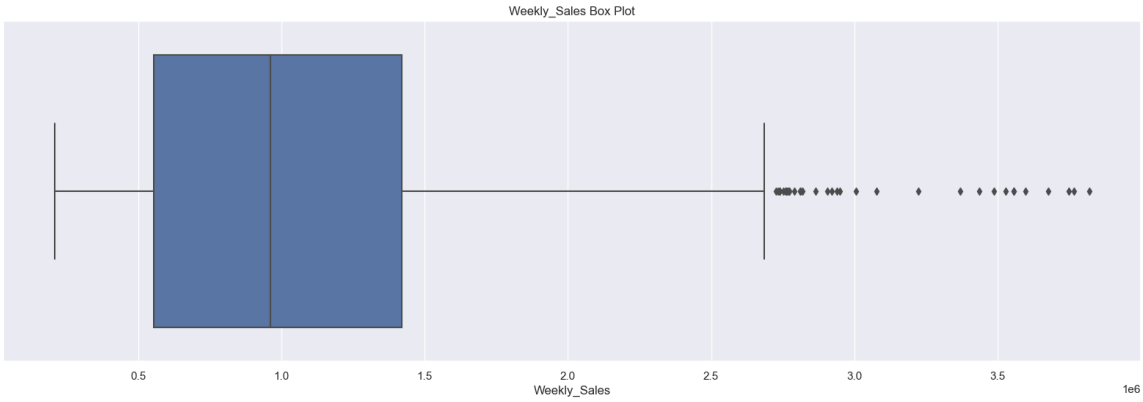
In [100…
```
# Represents a box plot
# Four subplots are arranged in a single row
fig , ax = plt.subplots(1,4,figsize=(20,4))
cols = ['Temperature','Fuel_Price','CPI','Unemployment']
for i,col in enumerate(cols):
    sns.boxplot(Walmart_ds,
                 x=col,
                 ax=ax[i],
                 color = 'Red'
                )
    ax[i].set_title(f'{col} Box Plot')
fig.show()
```



In [101…
```
# creates a histogram plot using seaborn's 'histplot' function
# Weekly_Sales distribution is right skewed
plt.figure(figsize=(20,6))
sns.histplot(x=Walmart_ds['Weekly_Sales'],kde=True);
plt.title('Weekly_Sales Distribution');
```



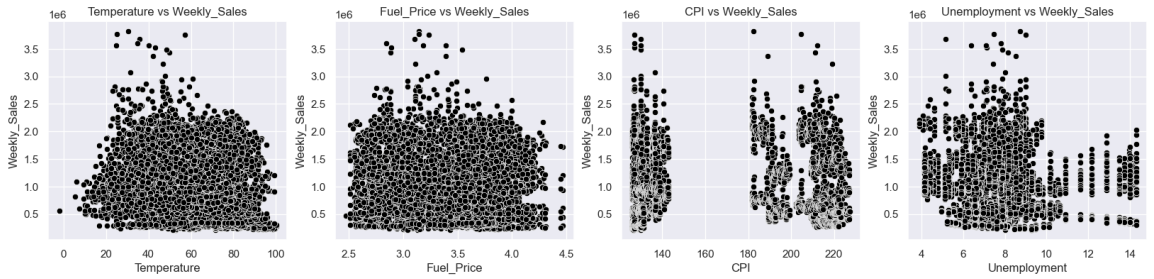
In [97]:
```
# box_plot of the 'weekly_sales' column
plt.figure(figsize=(20,6))
sns.boxplot(x=Walmart_ds['Weekly_Sales']);
plt.title('Weekly_Sales Box Plot');
```



Bivariate Analysis

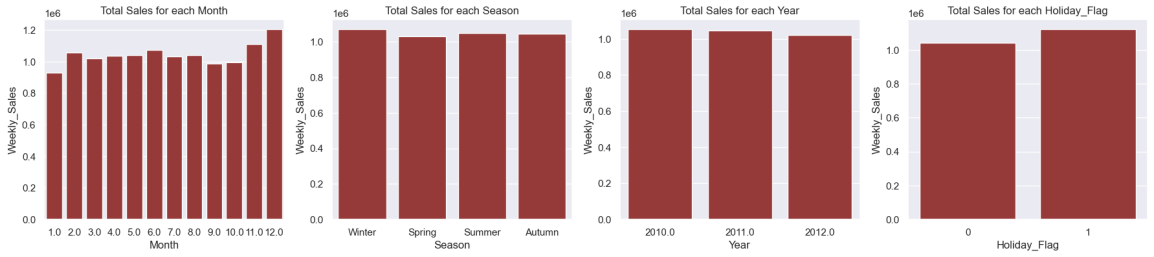
In [102…
```
# provides scatter plot ,showing the relationship between columns provided below
fig , ax = plt.subplots(1,4,figsize=(20,4))
cols = ['Temperature','Fuel_Price','CPI','Unemployment'] # List of column names
for i,col in enumerate(cols):
    sns.scatterplot(Walmart_ds,
                    y='Weekly_Sales',x=col,
                    ax=ax[i],
                    color = 'black'
                   )
```

```
    ax[i].set_title(f'{col} vs Weekly_Sales')
fig.show()
```



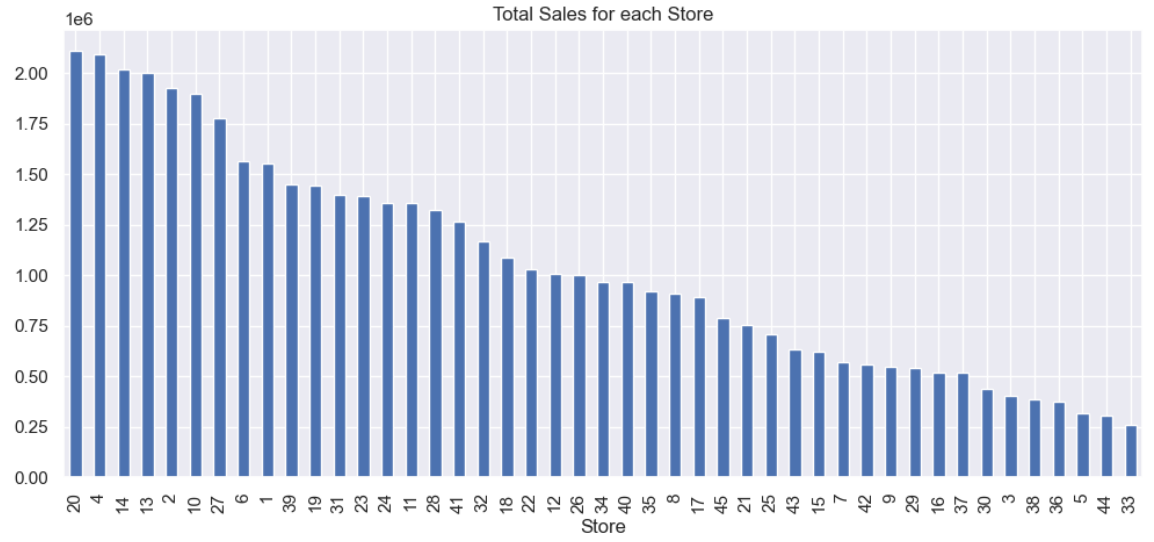
In [103…

```
# Generates box_plots,showing the total sales for a specific variable
fig , ax = plt.subplots(1,4,figsize=(22,4))
cols = ['Month','Season','Year','Holiday_Flag']
for i,col in enumerate(cols):
    sns.barplot(Walmart_ds,
                x=col,y='Weekly_Sales',
                ax=ax[i],
                errorbar=None,
                color='brown'
               )
    ax[i].set_title(f'Total Sales for each {col}')
fig.show()
```



In [47]:

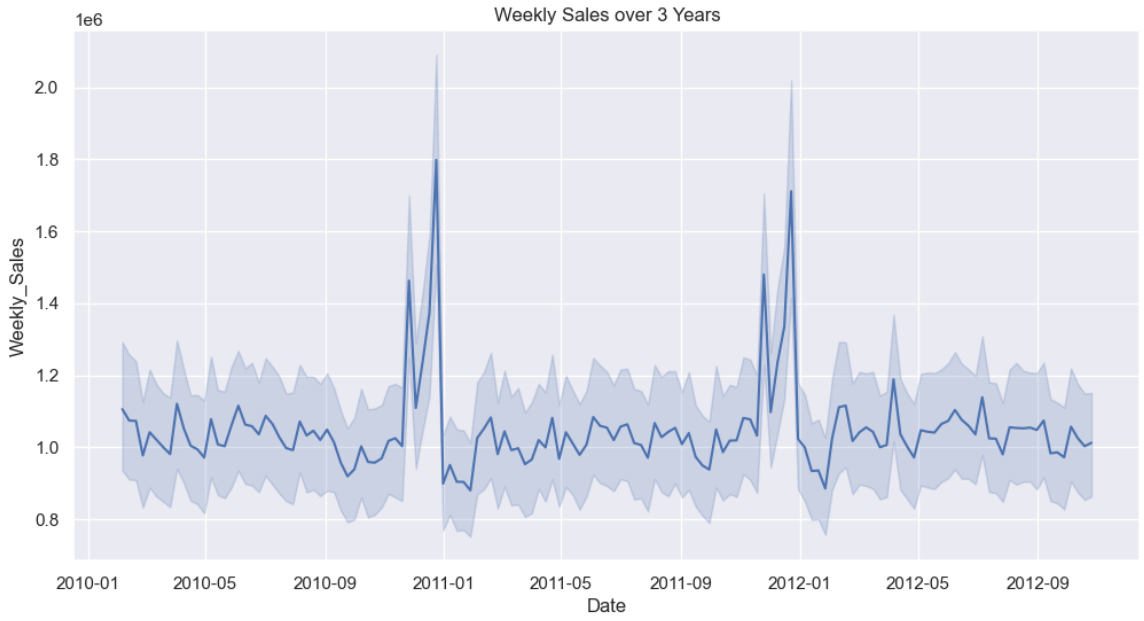
```
#1. Sales tend to be higher in winter and holidays.
#2. Sales are higher in Months 11 ,12.
plt.figure(figsize=(12,5))
Walmart_ds.groupby('Store')['Weekly_Sales'].mean().sort_values(ascending=False).
plt.title('Total Sales for each Store');
```



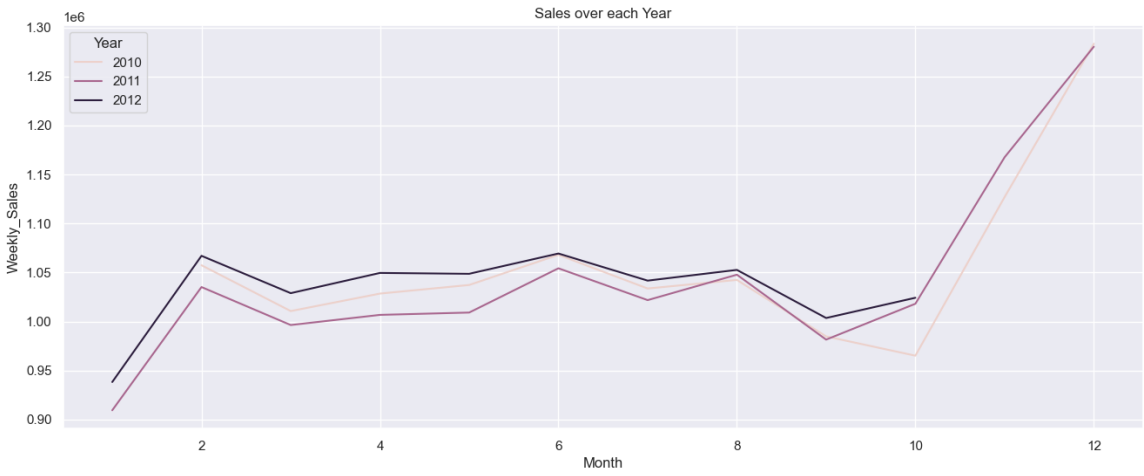
In [48]:

```
#Stores 20 ,4 ,14 ,13 ,2 achieved the highest sales.
plt.figure(figsize=(12,6))
```

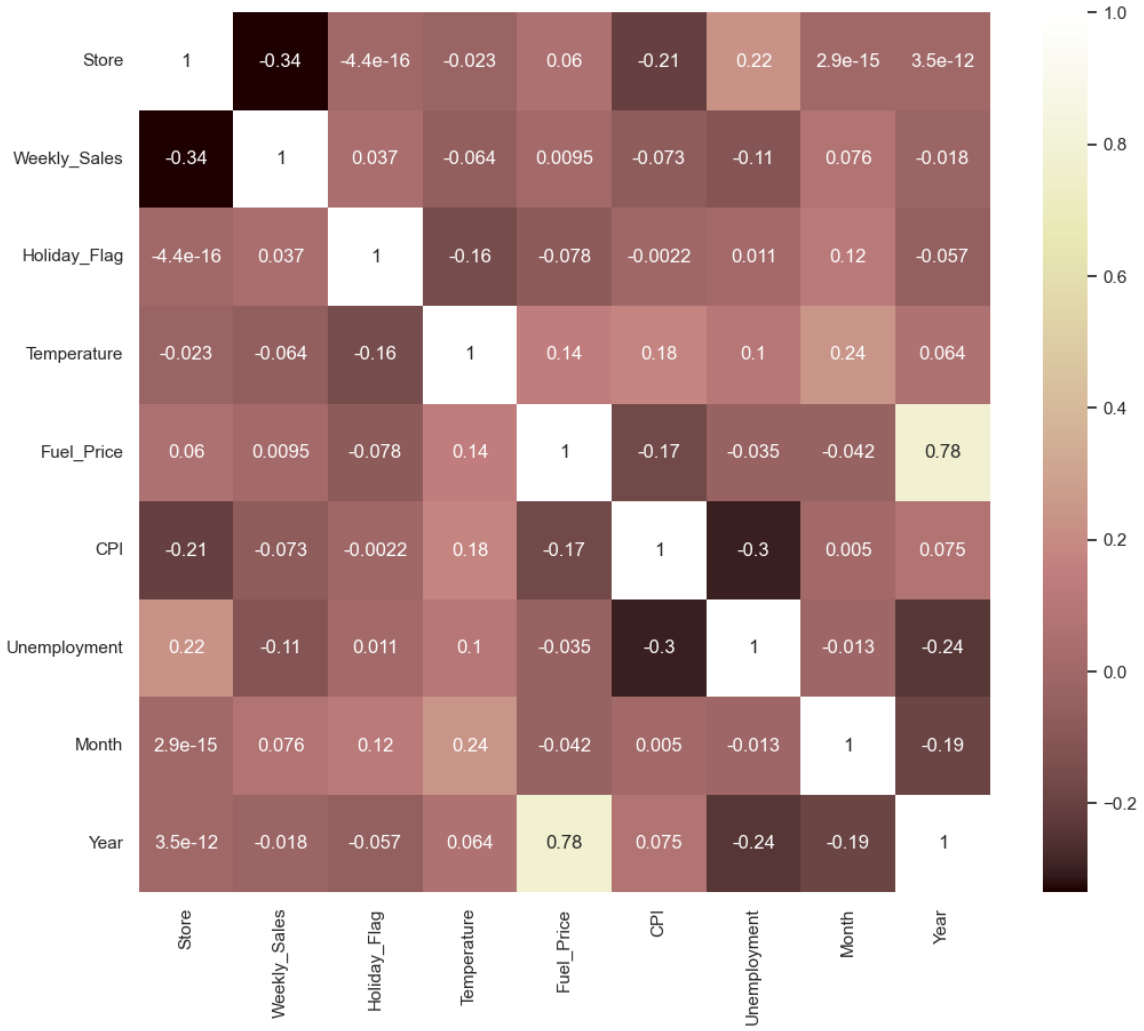
```
sns.lineplot(x=Walmart_ds["Date"],y=df['Weekly_Sales']);
plt.title('Weekly Sales over 3 Years');
```



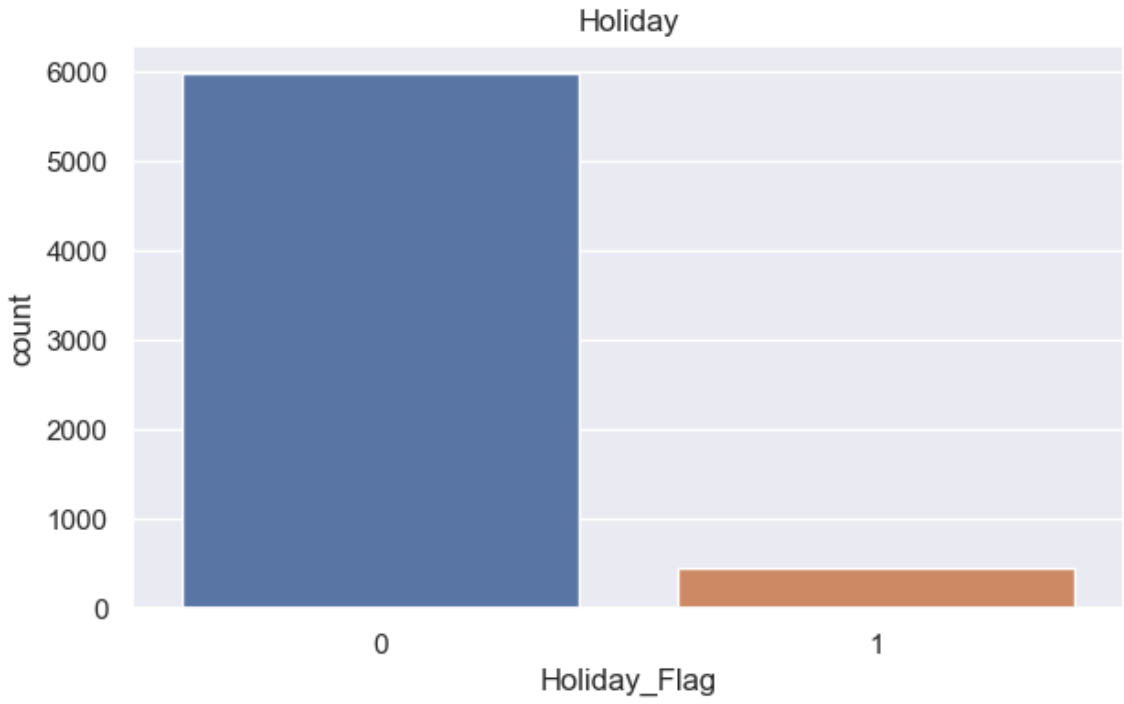
In [49]:
```
# As we mentioned before ,Sales are higher in Months 11 ,12
plt.figure(figsize=(16,6))
sns.lineplot(x=Walmart_ds["Month"],y=Walmart_ds['Weekly_Sales'],hue=Walmart_ds['
plt.title('Sales over each Year');
```



In [15]:
```
# The code generates a heatmap that visualizes the correlation between the numer
plt.figure(figsize=(12,10))
sns.heatmap(Walmart_ds.corr(numeric_only=True),cmap='pink',annot=True);
```



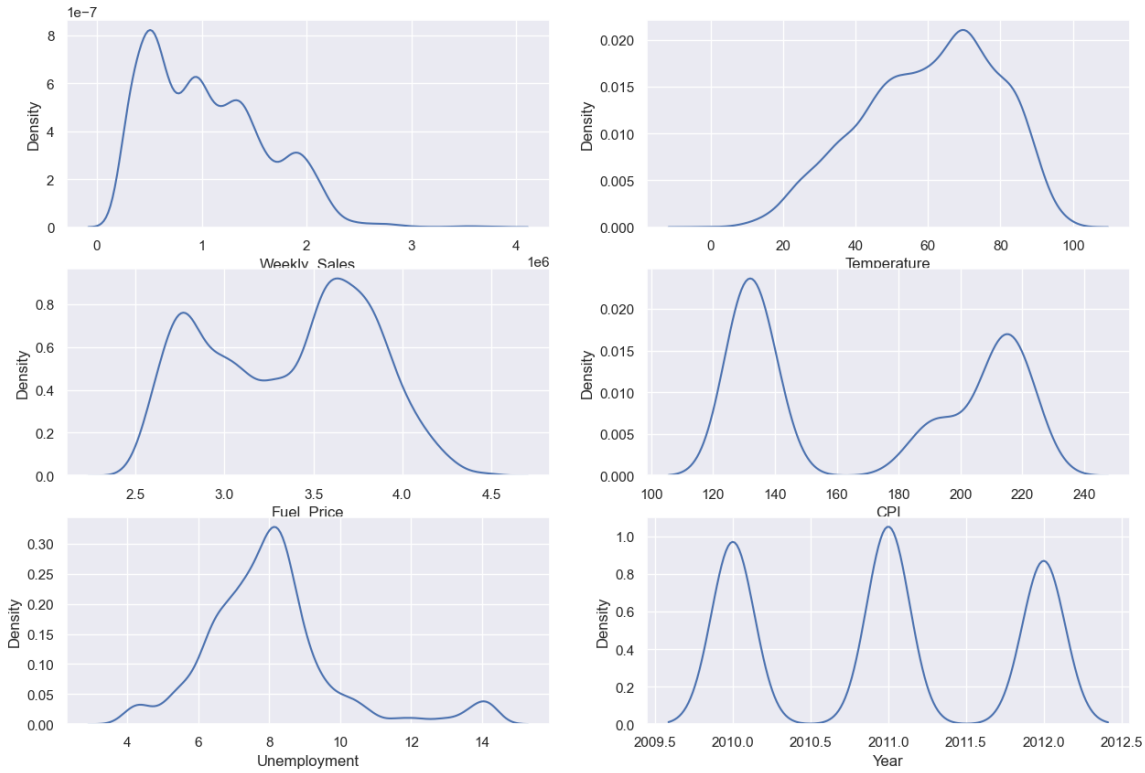
In [16]:
```
# generates a countplot that displays the distribution of the 'Holiday_flag'
plt.figure(figsize=(7,4))

sns.countplot(x= Walmart_ds.Holiday_Flag)
plt.title('Holiday')

plt.show()
```



In [50]:
```
# Returns a count of unique values in the 'Holiday_flag'
Walmart_ds.Holiday_Flag.value_counts()
```

Out[50]:
```
Holiday_Flag
0    5985
1     450
Name: count, dtype: int64
```

In [51]:
```
# The code creates subplots to visualize the kernel density estimates(KDE) plots
n = 1
plt.figure(figsize=(15,10))
for i in ['Weekly_Sales','Temperature', 'Fuel_Price','CPI', 'Unemployment','Year
        if n<=6:
            plt.subplot(3,2,n);
            n+=1
            sns.kdeplot(x = Walmart_ds[i])
            plt.xlabel(i)
```



Removing Outliners

In [52]:
```
num_features = ['Temperature','Fuel_Price','CPI','Unemployment','Weekly_Sales']
for feature in num_features:
    q1 = Walmart_ds[feature].quantile(0.25)
    q3 = Walmart_ds[feature].quantile(0.75)
    iqr = q3-q1
    lower = q1 - 1.5*iqr
    upper = q3 + 1.5*iqr
    Walmart_ds = Walmart_ds[(Walmart_ds[feature] >= lower) & (Walmart_ds[feature
```

Model Building

In [61]:
```
def model_tunning(model,X_train,y_train,params):
    # uses 'GridsearchCV' to perform an exhaustive search
    '''it fits the model on the training data and performs cross validation to f
    the best parameter that maximizes the 'r2' score'''
    grid_search = GridSearchCV(estimator= model,param_grid= param_grid,cv=5,scor
    grid_search.fit(X_train,y_train)
    best_params = grid_search.best_params_
    best_estimator = grid_search.best_estimator_
    best_score = grid_search.best_score_
    print("Best parameters are: ",best_params)
    print('Mean cross-validated score of the best_estimator is: ',best_score)
    return best_estimator
```

In [62]:
```
'''Imports function which used to split the dataset into training and testing se
The optimum hyperparameters for a particular model are found via grid search, wh
cross_val_score perfroms cross-validation to evaluate model performance'''
from sklearn.model_selection import train_test_split , GridSearchCV , cross_val_
'''For preprocessing it imports standardScaler for standardizing numerical featu
for removing the mean and scaling to unit variance,imports class which is used f
```

```
imports class which is used for one-hot encoding categorical features'''
from sklearn.preprocessing import StandardScaler ,PolynomialFeatures , OneHotEnc
# Creates pipeline of data preprocessing
from sklearn.pipeline import make_pipeline
# impliments linear regression models, implements ridge regression, implements l
from sklearn.linear_model import LinearRegression , Ridge , Lasso
# imports support vector regression
from sklearn.svm import SVR
#imports k-nearest neighbors regression
from sklearn.neighbors import KNeighborsRegressor
#used for binary encoding categorical features
from category_encoders import BinaryEncoder
```

Data Splitting

In [63]:
```
#The weekly sales is assigned to target variable
features = Walmart_ds.columns.drop(['Weekly_Sales','Date'])
target = 'Weekly_Sales'

X = Walmart_ds[features]
y = Walmart_ds[target]
```

In [64]:
```
# Performs train-test splitting of the feature matrix 'X' and the targey variabl
X_train , X_test , y_train , y_test = train_test_split(X,y,random_state=42 ,  te
```

Data Preprocessing

In [65]:
```
# converting store column to object datatype
Walmart_ds['Store'] = Walmart_ds['Store'].astype('object')
```

In [66]:
```
# combining these two preprocessing steps into the pipeline
preprocessor = make_pipeline(
        BinaryEncoder(cols=['Store','Season']),
        StandardScaler()
)
```

Model Selection

1. Linear Regression

The report you offered shows the outcomes of utilising grid search to fine-tune the linear regression model with polynomial features.

Best criteria: 'polynomialfeatures__degree' was the most effective grid search parameter, with a value of 3. This indicates that the degree 3 has the highest performance for the polynomial characteristics. It implies that the model's performance was at its best when third-degree polynomial features were included.

Mean cross-validated score: The best estimator has a mean cross-validated score of 0.9604197711731995. This rating shows the best estimator's cross-validation average R-squared value. R-squared is a statistical metric that shows how well the model fits the

data. In this scenario, the model obtained a high R-squared score, suggesting a strong fit to the data. A number that is near to 1 denotes a better match.

Overall, the results indicate that the model functioned satisfactorily with third-degree polynomial features and that it had a high degree of goodness of fit, with an R-squared score of around 0.96.

In [68]:
```
lin_reg = make_pipeline(
        preprocessor,
        PolynomialFeatures(degree=2),
        LinearRegression()
)
param_grid = {'polynomialfeatures__degree':[2,3,4,5]}
lin_reg = model_tunning(lin_reg,X_train,y_train,param_grid)
```

```
Best parameters are:  {'polynomialfeatures__degree': 3}
Mean cross-validated score of the best_estimator is:  0.9604197711731995
```

After being fitted to the training data and assessed on both the training and test sets, the output you gave displays the linear regression model's accuracy on the training set and test set.

Accuracy of the training set: The accuracy of the training set is 97.9%. This shows that the linear regression model can account for about 97.9% of the variation in the target variable (Weekly_Sales) using the training data. The model may have learnt the patterns and correlations in the training data rather well if the training set accuracy is greater.

Accuracy of the test set: The test set accuracy is 96.1%. This shows that the linear regression model can explain around 96.1% of the variation in the target variable when it is applied to the test data (data that it has not seen during training). The model's capacity to produce accurate predictions on fresh, untested samples is demonstrated by the high test set accuracy, which shows that the model generalises effectively to new, untested data.

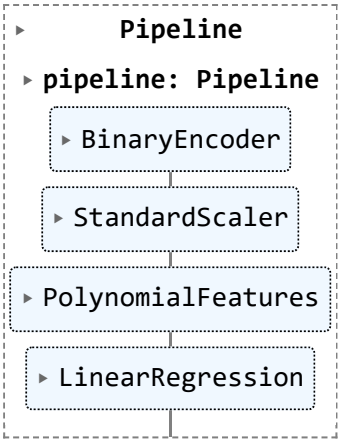
In [70]:
```
lin_reg.fit(X_train,y_train)
print("Training set Accuracy:",round(lin_reg.score(X_train,y_train),3)*100)
print("Test set Accuracy:",round(lin_reg.score(X_test,y_test),3)*100)
```

```
Training set Accuracy: 97.89999999999999
Test set Accuracy: 96.1
```

Displaying the details and parameters of the linear regression model that has been fitted to the training data

In [75]:
```
lin_reg
```

Out[75]:



K Nearest Neighbors Regression

The output shows the best parameters found during the tuning process for the K-Nearest Neighbors (KNN) regression model. The best parameters are as follows:

Metric: 'manhattan' Number of neighbors: 5 Weighting scheme: 'distance'

The KNeighborsRegressor method is followed by the preprocessor, which comprises binary encoding of categorical variables and feature standardisation. The KNN algorithm calculates the distances between data points using the supplied metric and bases its predictions on the weighted average of the target values of its neighbours.

According to the results, the KNN model with the optimal parameter values had a mean cross-validated score of around 0.936, which indicates how well it performed at predicting the intended variable.

In [72]:
```
knn = make_pipeline(
    preprocessor,
    KNeighborsRegressor()
)
param_grid = {
    'kneighborsregressor__n_neighbors': np.arange(1, 21),
    'kneighborsregressor__weights': ['uniform', 'distance'],
    'kneighborsregressor__metric': ['euclidean', 'manhattan', 'minkowski']
}

knn = model_tunning(knn,X_train,y_train,param_grid)
```

```
Best parameters are:  {'kneighborsregressor__metric': 'manhattan', 'kneighborsr
egressor__n_neighbors': 5, 'kneighborsregressor__weights': 'distance'}
Mean cross-validated score of the best_estimator is:  0.9361216579155347
```

The KNN model accurately predicts the target variable on the training data, as shown by the calculation of the training set accuracy, which is 100.0%. The model may have memorised the training set rather than recognising broader patterns if it achieved 100% accuracy on the training set, which might indicate overfitting.

The accuracy of the test set is estimated to be 93.8%. This rating shows how well the model can forecast the target variable using previously unobserved data. The model appears to function well and generalise to new data with a test set accuracy of 93.8%. It

suggests that the KNN model can identify trends and connections among the characteristics that aid in predicting the target variable.

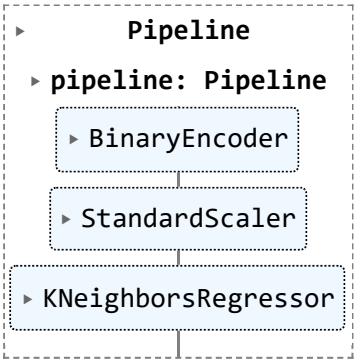
It is significant to observe that despite the fact that the accuracy of the training set is higher than the accuracy of the test set, the two numbers are still very close.

In [73]:
```
knn.fit(X_train,y_train)
print("Training set Accuracy:",round(knn.score(X_train,y_train),3)*100)
print("Test set Accuracy:",round(knn.score(X_test,y_test),3)*100)
```

```
Training set Accuracy: 100.0
Test set Accuracy: 93.8
```

In [74]:
```
knn
```

Out[74]:



Random Forest

Both the training and test sets showed excellent accuracy for the random forest model. The model's fit to the training set data is pretty good, capturing the underlying patterns and connections, as seen by the training set accuracy of 99.38. According to the test set accuracy of 95.71, the model generalises effectively to previously unobserved data and makes reliable predictions about novel cases.

It's crucial to keep in mind that accuracy could not give a full picture of the model's performance. To determine the robustness and dependability of the model, it is advised to take into account other assessment metrics and maybe carry out cross-validation.

In [80]:
```
from sklearn.preprocessing import OneHotEncoder

# Creating an instance of the OneHotEncoder
encoder = OneHotEncoder()

# Fit the encoder to the column 'store' and 'season'
encoder.fit(X_train[['Store', 'Season']])

# The categorical columns in the training and test data are transformed
X_train_encoded = encoder.transform(X_train[['Store', 'Season']])
X_test_encoded = encoder.transform(X_test[['Store', 'Season']])

# Concatenate the encoded features with the remaining numerical features
X_train_encoded = np.concatenate([X_train_encoded.toarray(), X_train.drop(['Stor
X_test_encoded = np.concatenate([X_test_encoded.toarray(), X_test.drop(['Store',
```

```
# Create an instance
random_forest = RandomForestRegressor()

# random forest model is fitted to the training data
random_forest.fit(X_train_encoded, y_train)

# predictions are made
train_predictions = random_forest.predict(X_train_encoded)

# Make predictions on the test set
test_predictions = random_forest.predict(X_test_encoded)

train_accuracy = random_forest.score(X_train_encoded, y_train)
test_accuracy = random_forest.score(X_test_encoded, y_test)

print("Training set Accuracy:", round(train_accuracy * 100, 2))
print("Test set Accuracy:", round(test_accuracy * 100, 2))
```

```
Training set Accuracy: 99.38
Test set Accuracy: 95.71
```

Conclusion

According to the research, the random forest algorithm had the greatest results in forecasting sales based on the provided characteristics, with a test set accuracy of 95.71%. This ensemble technique effectively handles complicated interactions by combining various decision trees. While the decision tree model had overfitting symptoms, linear regression and k-nearest neighbours both performed rather well. Therefore, for this regression issue, the random forest technique is suggested.

Reference

https://www.kaggle.com/code/yasserh/walmart-sales-prediction-best-ml-algorithms

https://www.kaggle.com/code/shatabdi5/95-walmart-sales-prediction-using-regressor

https://www.kaggle.com/datasets/yasserh/walmart-dataset/code?datasetId=1820993&sortBy=voteCount

In [ ]: