----------------------------> Capstone Project - MIT <-----------------------------

A) Problem Statement 1:

> **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.**

B) 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.

C) Data Description

Dataset Information:
The walmart.csv contains 6435 rows and 8 columns.

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

In [1]:
```
# Impoting the Libraries
import pandas as pd
import numpy as np
from numpy import log

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
```

```python
import math

from statsmodels.tsa.stattools import adfuller, acf,pacf
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt

import pmdarima as pm

import warnings
warnings.filterwarnings("ignore")
```

```
/Users/jagannathprasad/opt/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >
=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```python
# Loading the data set
df = pd.read_csv("Walmart.csv")
```

In [3]:

```python
df.head()
```

Out[3]:

	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

D) Data Pre-processing Steps and Inspiration

The Data pre-processing for this Walmart dataset is based on the specific use case for forcasting. Here are some common steps that could be applied and sources of inspiration:

1.Data Cleaning: Removing missing or duplicate values, correcting inconsistent data, and handling outliers.

> **Inspiration: Common data quality issues, such as missing values, duplicate records, and inconsistent data, can be addressed by following best practices and utilizing tools such as data profiling.**

2.Data Transformation: Scaling, normalizing, and encoding the data so that all features are in the same range and format.

> **Inspiration: Domain knowledge about the Walmart dataset and its features can guide the decision making on the appropriate transformation techniques to use.**

3.Data Integration: Combining multiple datasets, such as sales data, customer data, and product data, into a single one to create a comprehensive view of the data.

> **Inspiration: The business problem you are trying to solve may dictate which datasets are necessary to integrate and how they should be combined.**

4.Data Reduction: Reducing the dimensionality of the data through techniques such as feature selection and principal component analysis.

> **Inspiration: The specific use case and the size of the Walmart dataset may determine the need for data reduction techniques and which techniques are appropriate to use.**

These are just a few examples of common data pre-processing step. These steps help us to understand the data and the problem we are trying to solve before starting the pre-processing steps to make informed decisions.

In [4]:
```
# Checking for the the information of the data set
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   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
```

Observation:

> **The "Data" column is in the object data type which is an inappropriate format**

In [5]:

```
# convert Date column data type to date-time
df['Date'] = pd.to_datetime(df.Date)
```

In [6]:

```
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   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
```

Comment:

> **The Date column dtype has been change to datetime dtype successfully**

In [7]:
```
# Creating a new Data Frame copy for further analysis.
df_new = df.copy()
```

In [8]:
```
# Verifying weather the data has been copied properly
df_new.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
```

In [9]:
```
#Now adding the month name and month from the date column
df_new['month'] = df_new['Date'].dt.month
df_new['month_name'] = df_new['Date'].dt.month_name()
```

In [10]:
```
# Adding the week column
df_new['week'] =df_new['Date'].dt.week
```

In [11]:
```
# Adding the year column
df_new['year'] =df_new['Date'].dt.year
```

In [12]:
```
# Adding the year column
df_new['Day_Name'] =df_new['Date'].dt.day_name()
```

In [13]:
```
df_new
```

Out[13]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	month	month_name	week	year	Day_
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	5	May	17	2010	S
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	12	December	48	2010	Th
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	2	February	7	2010	
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	2	February	8	2010	
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	5	May	18	2010	M
...	...	...	...	...	...	...	...	...	...	...	...	...	
6430	45	2012-09-28	713173.95	0	64.88	3.997	192.013558	8.684	9	September	39	2012	
6431	45	2012-05-10	733455.07	0	64.89	3.985	192.170412	8.667	5	May	19	2012	Th
6432	45	2012-12-10	734464.36	0	54.47	4.000	192.327265	8.667	12	December	50	2012	M
6433	45	2012-10-19	718125.53	0	56.47	3.969	192.330854	8.667	10	October	42	2012	
6434	45	2012-10-26	760281.43	0	58.85	3.882	192.308899	8.667	10	October	43	2012	

6435 rows × 13 columns

In [14]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 13 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
 8   month         6435 non-null   int64
 9   month_name    6435 non-null   object
 10  week          6435 non-null   int64
 11  year          6435 non-null   int64
 12  Day_Name      6435 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(5), object(2)
memory usage: 653.7+ KB
```

df_new.reset_index(drop=True)

In [15]:
```
#Checking for null values
df_new.isnull().sum()
```

Out[15]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
month           0
month_name      0
week            0
year            0
Day_Name        0
dtype: int64
```

In [16]:
```
# check duplicates
df_new[df_new.duplicated()]
```

Out[16]:

Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	month	month_name	week	year	Day_Name

Comment:

> **The dataset is free from duplicate entires.**

Checking the the distribution of the features of the dataset

In [17]:
```
df_new.hist(figsize=(30,20));
```

From the above histograms, we can understand that:

> - The number of transactions occurred almost evenly across various stores and years.
> - The distribution of weekly_sales right-skewed. Only a few of the weekly sales are above 2 million USD.

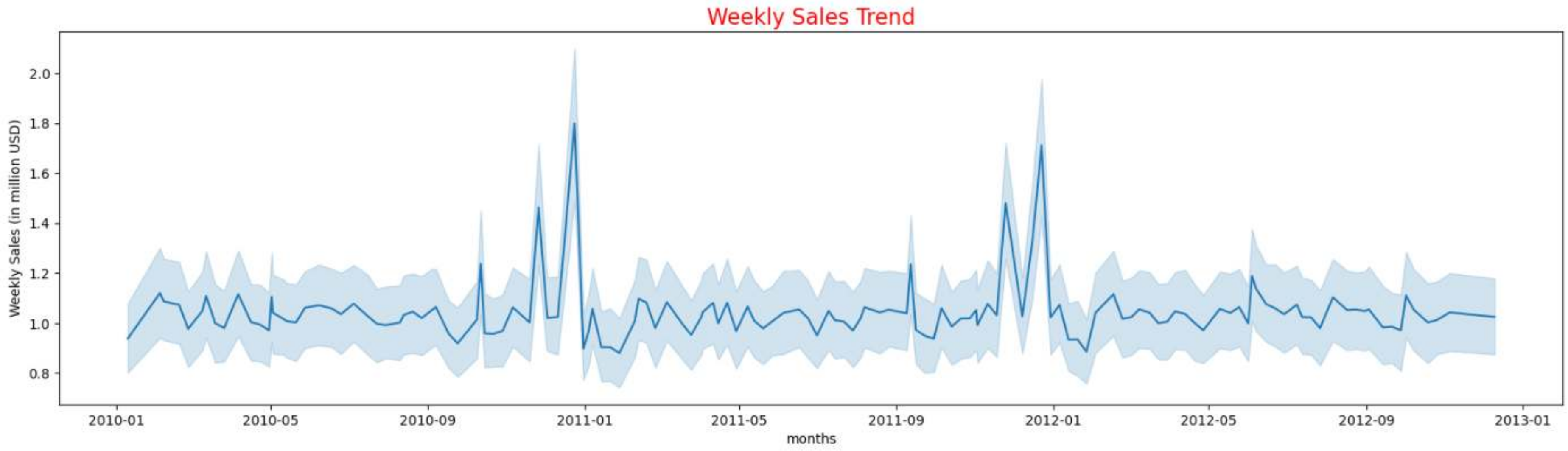
> - The distribution of temperature is approximately normal.
> - The distribution of fuel_price is bi-modal.
> - CPI formed two clusters.
> - unemployment rate is near normally distributed.
> - Four consecutive months November-February recorded the highest sales.

Overall trend in sales over time

> - Sales trend analysis involves examining the historical sales data of a business or product over time to understand patterns, trends, and changes in sales performance.
> - It is an important tool for businesses to identify opportunities for growth, understand their customers' behaviour, optimise resources, and make informed decisions about future sales.
> - We will aggregate the average weekly sales by months for the three year and visualise the trend using a line plot.

In [18]:

```
# plot the line chart of the weekly_sales
fig, ax = plt.subplots(figsize=(20, 5))
sns.lineplot(x=df_new.Date, y=(df_new.Weekly_Sales/1e6))
plt.xlabel('months')
plt.ylabel('Weekly Sales (in million USD)')
plt.title('Weekly Sales Trend',fontdict={'fontsize': 16, 'color':'red'}, pad=5)

annot = ax.annotate("", xy=(0,0), xytext=(20,20),textcoords="offset points",
                    bbox=dict(boxstyle="round", fc="w"),
                    arrowprops=dict(arrowstyle="->"))
annot.set_visible(False)

plt.show()
```



Comment

> - **The line plot reveals that weekly sales at Walmart generally remain stable throughout the year, with the exception of November and December, which experience a significant increase in sales.**
> - **This trend is likely due to the holiday season, when consumers typically make more purchases and retailers offer promotions and discounts.**
> - **To capitalize on this behavior, Walmart could consider offering seasonal discounts and promotions, as well as ensuring a seamless and enjoyable shopping experience through their mobile and web outlets during festive periods.**
> - **By doing so, they can encourage more customers to make purchases and potentially drive up sales.**

Checking for any seasonality trends in the dataset

> - **Seasonality trends analysis can be extremely valuable for businesses, as it allows us to better forecast future sales, make more informed decisions about inventory and staffing, and understand the drivers of customer demand leading to improved efficiency and profitability.**
>
> - **We will create a pivot table to group the data by month and year and calculate the average sales for each period. We will then plot the average sales of the table using line chart for the three years. This will allow us to see if there are any patterns in the data that repeat at regular intervals.**

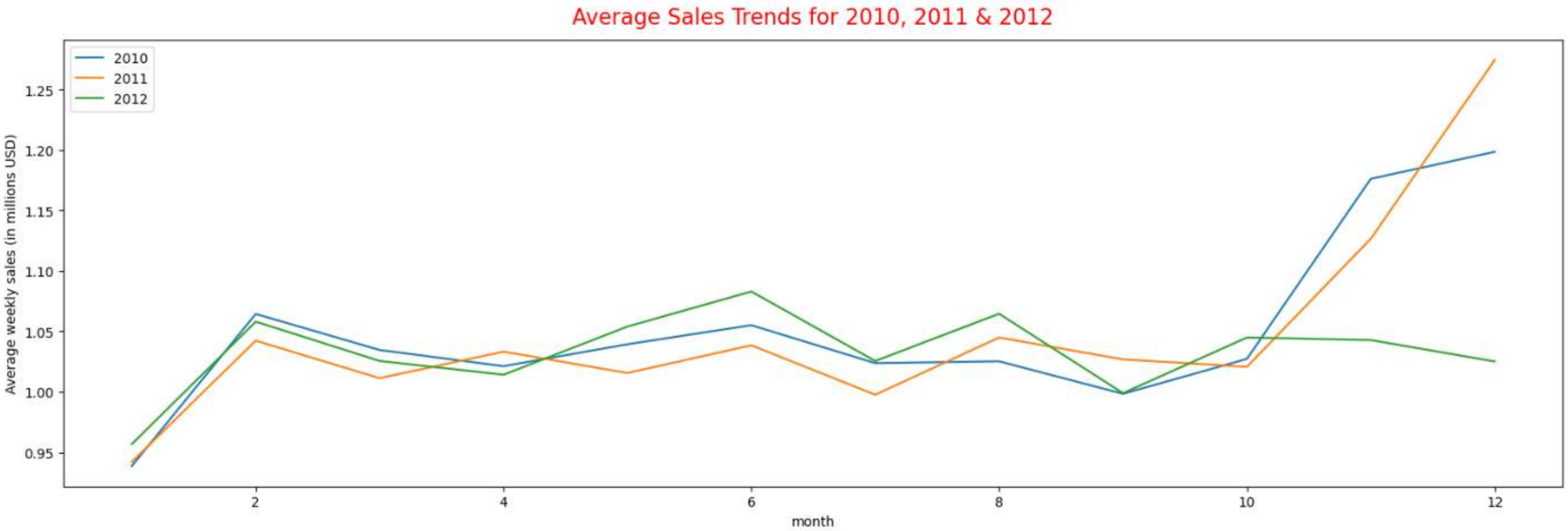
In [19]:
```python
# create the pivot table
pivot_table = df_new.pivot_table(index='month', columns='year', values='Weekly_Sales')
# display the pivot table
pivot_table
```

Out[19]:

year month	2010	2011	2012
1	9.386639e+05	9.420697e+05	9.567817e+05
2	1.064372e+06	1.042273e+06	1.057997e+06
3	1.034590e+06	1.011263e+06	1.025510e+06
4	1.021177e+06	1.033220e+06	1.014127e+06
5	1.039303e+06	1.015565e+06	1.053948e+06
6	1.055082e+06	1.038471e+06	1.082920e+06
7	1.023702e+06	9.976049e+05	1.025480e+06
8	1.025212e+06	1.044895e+06	1.064514e+06
9	9.983559e+05	1.026810e+06	9.988663e+05
10	1.027201e+06	1.020663e+06	1.044885e+06
11	1.176097e+06	1.126535e+06	1.042797e+06
12	1.198413e+06	1.274311e+06	1.025078e+06

In [20]:
```python
# plot the average sales
fig, ax = plt.subplots(figsize=(20, 6))
sns.set_palette("bright")
sns.lineplot(x=pivot_table.index, y=pivot_table[2010]/1e6, ax=ax, label='2010')
sns.lineplot( x=pivot_table.index, y=pivot_table[2011]/1e6, ax=ax, label='2011')
sns.lineplot( x=pivot_table.index, y=pivot_table[2012]/1e6, ax=ax, label='2012')
plt.ylabel('Average weekly sales (in millions USD)')
plt.title('Average Sales Trends for 2010, 2011 & 2012', fontdict ={'fontsize':16,
                                                                   'color':'red',
                                                                   'horizontalalignment': 'center'},
                                                                   pad=12)

# Add a legend
```

```
plt.legend()
plt.show()
```



Comment:

> - **We can observe that the line charts for the three years for the month of January to October simultaneously follow a sawtooth shape with big rises experienced in November and December due to holidays.**
> - **This indicates seasonality trends as months do have consistencies in bigger or smaller sales for the three years.**
> - **We can also observe that although 2011 performed worst than 2010 in terms of average sales for Walmart, the trend was reversed for the year 2012 which performed better than 2010. However, the data for 2012 ends in October, which may explain the significant drop in sales for November."**

In [21]:
```
# Checking the number of stores in the data set
a = df_new['Store'].nunique()
print(f"There are {a} Stores in this data")
```

```
There are 45 Stores in this data
```

In [22]:
```
print("The Number of unique values in the data set in each column \n")
df_new.nunique()
```

```
The Number of unique values in the data set in each column
```

Out[22]:

```
Store             45
Date             143
Weekly_Sales    6435
Holiday_Flag       2
Temperature     3528
Fuel_Price       892
CPI             2145
Unemployment     349
month             12
month_name        12
week              52
year               3
Day_Name           7
dtype: int64
```

In [23]:

```python
print("The Statistical discription of the data set.\n")
df_new.describe()
```

```
The Statistical discription of the data set.
```

Out[23]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	month	week	
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.0
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999151	6.475524	26.000000	2010.9
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875885	3.321797	14.511794	0.7
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879000	1.000000	1.000000	2010.0
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891000	4.000000	14.000000	2010.0
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874000	6.000000	26.000000	2011.0
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622000	9.000000	38.000000	2012.0
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000	12.000000	52.000000	2012.0

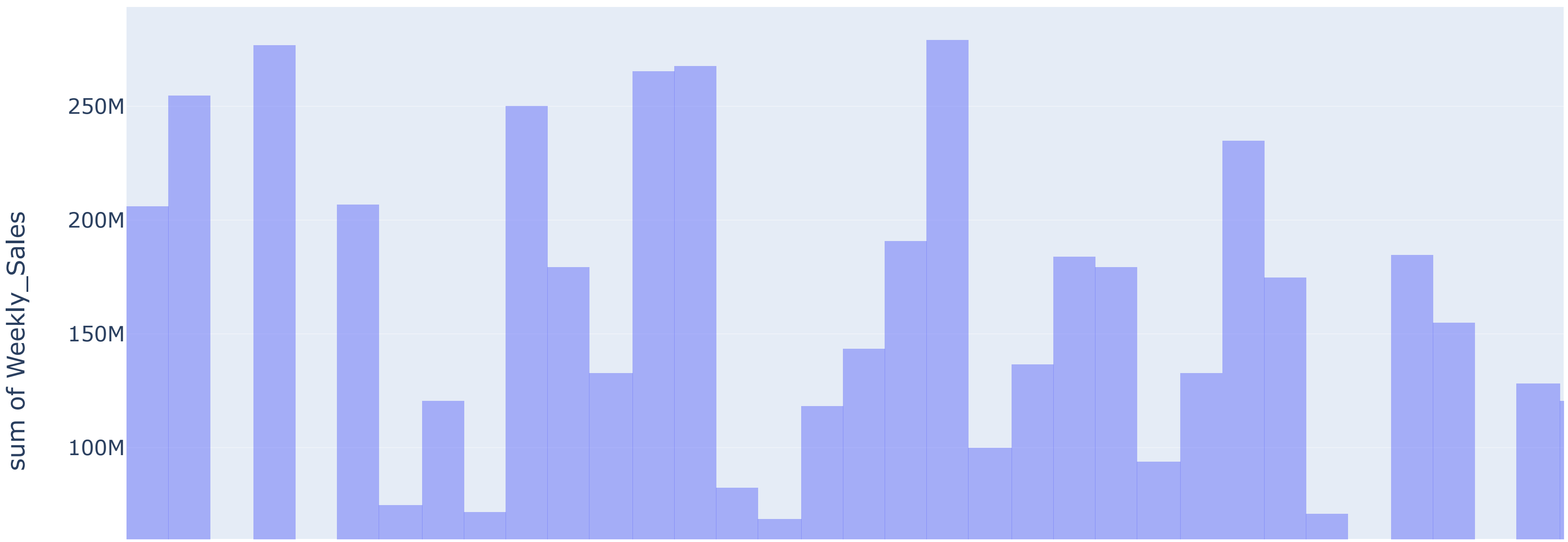
> **We can Observed that the Max temperature was 100 degree "F".

In [24]:

```python
df_new.columns
```

```
Out[24]: Index(['Store', 'Date', 'Weekly_Sales', 'Holiday_Flag', 'Temperature',
                'Fuel_Price', 'CPI', 'Unemployment', 'month', 'month_name', 'week',
                'year', 'Day_Name'],
               dtype='object')
```

```
In [25]: px.histogram(df_new, x = "Store", y = "Weekly_Sales", color = "Holiday_Flag", barmode="overlay")
```

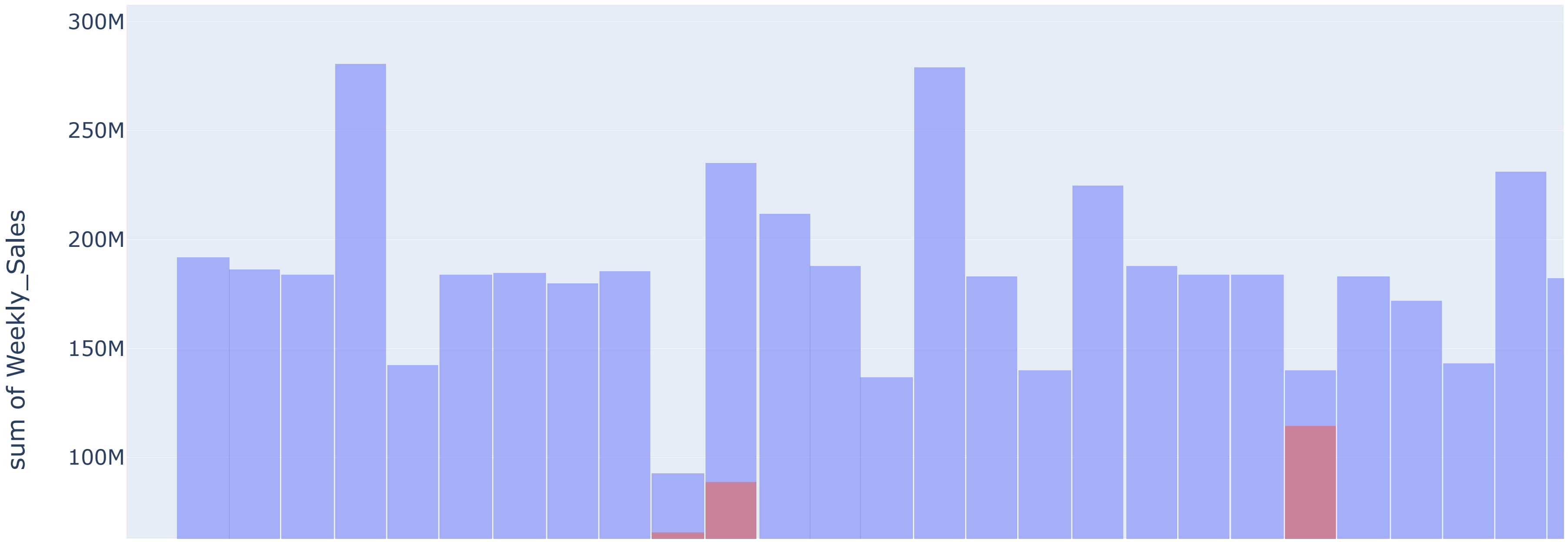


Observation:

Store 20 has the highest Weekly sales

Store 33 has the least Weekly sales

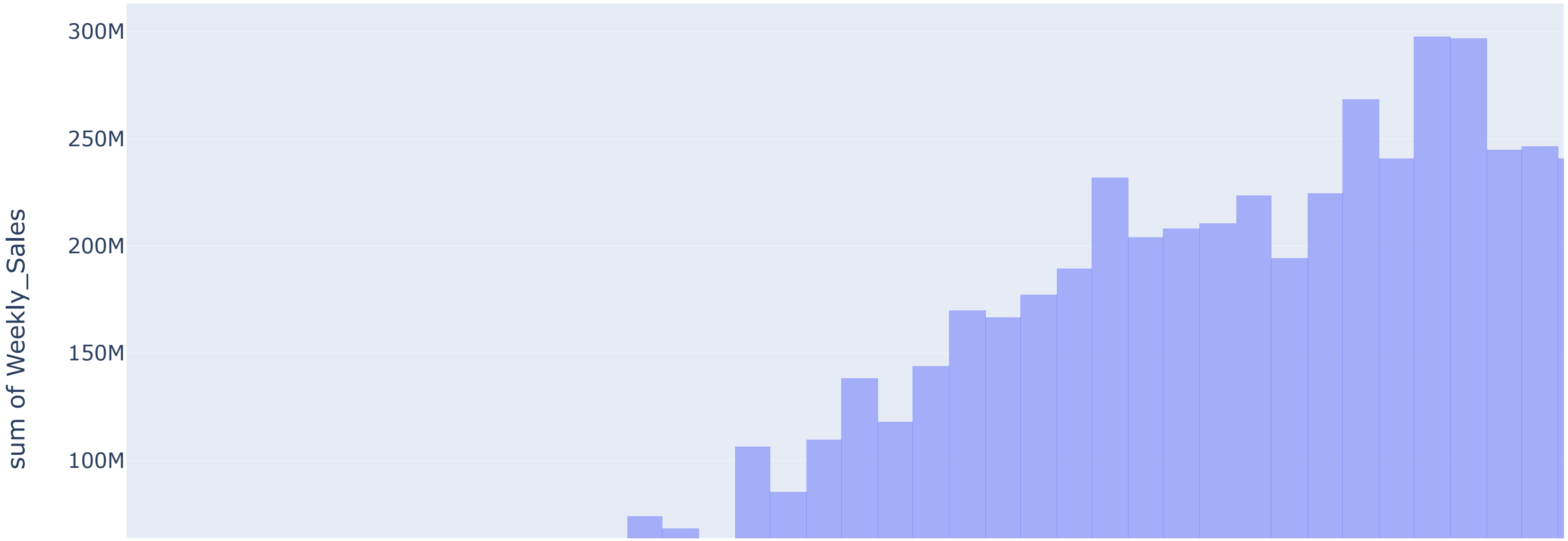
In [26]:
```
px.histogram(df_new, x = "Date", y = "Weekly_Sales", color = "Holiday_Flag", barmode="overlay")
```



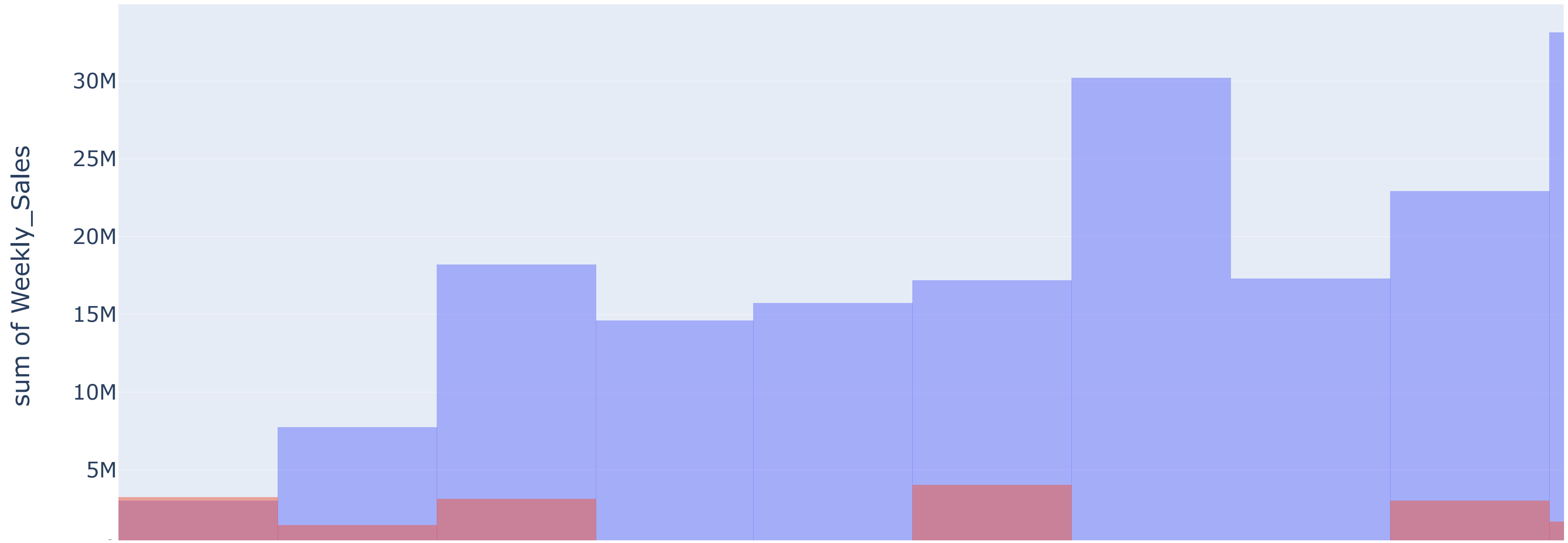
Observation:

On the time period from 1st June 2012 to 30th June 2012 has the highest sum of Weekly sales

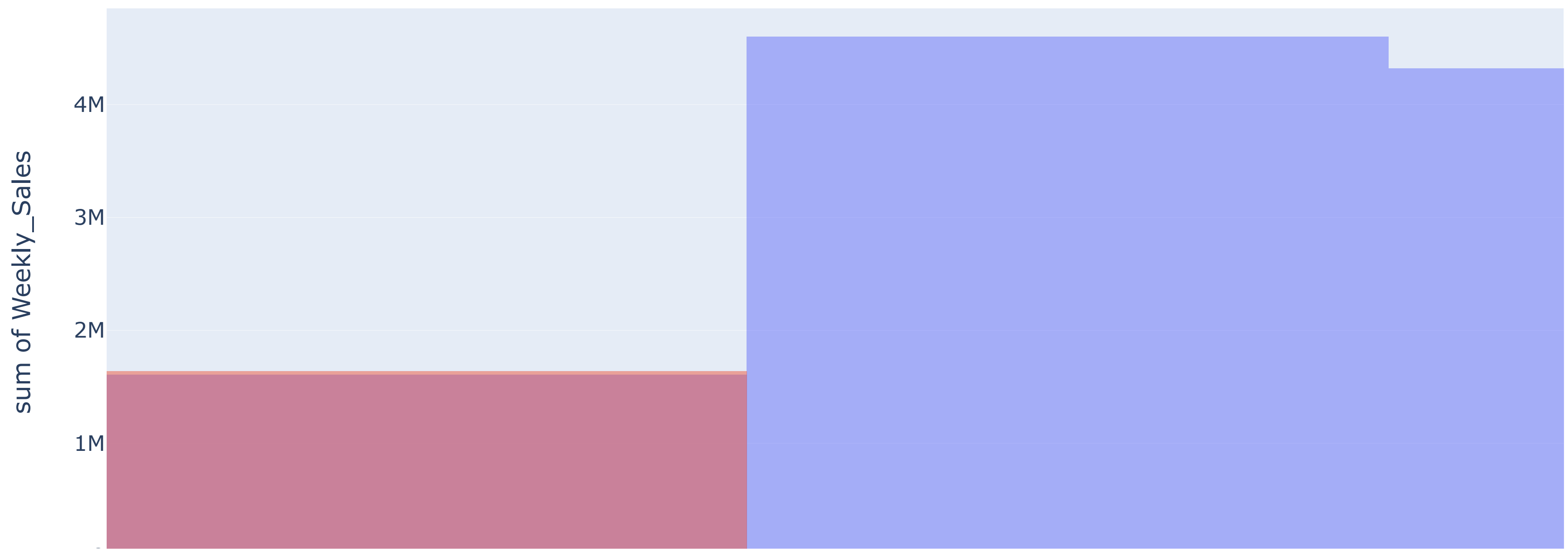
In [27]:
```
px.histogram(df_new, x = "Temperature" , y = "Weekly_Sales", color='Holiday_Flag',barmode='overlay')
```



In [28]:
```
px.histogram(df_new, x = "Temperature" , y = "Weekly_Sales", color='Holiday_Flag',barmode='overlay',
             animation_frame='Store')
```

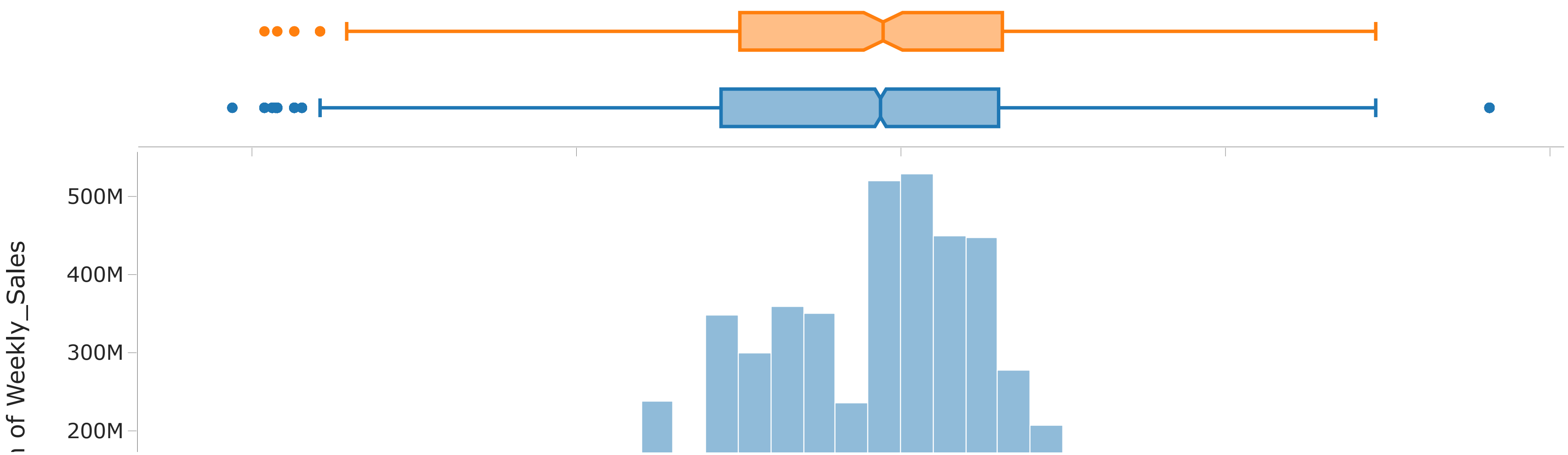


In [29]:
```
px.histogram(df_new, x = "Temperature" , y = "Weekly_Sales", color='Holiday_Flag',barmode='overlay',
             animation_frame='Unemployment')
```

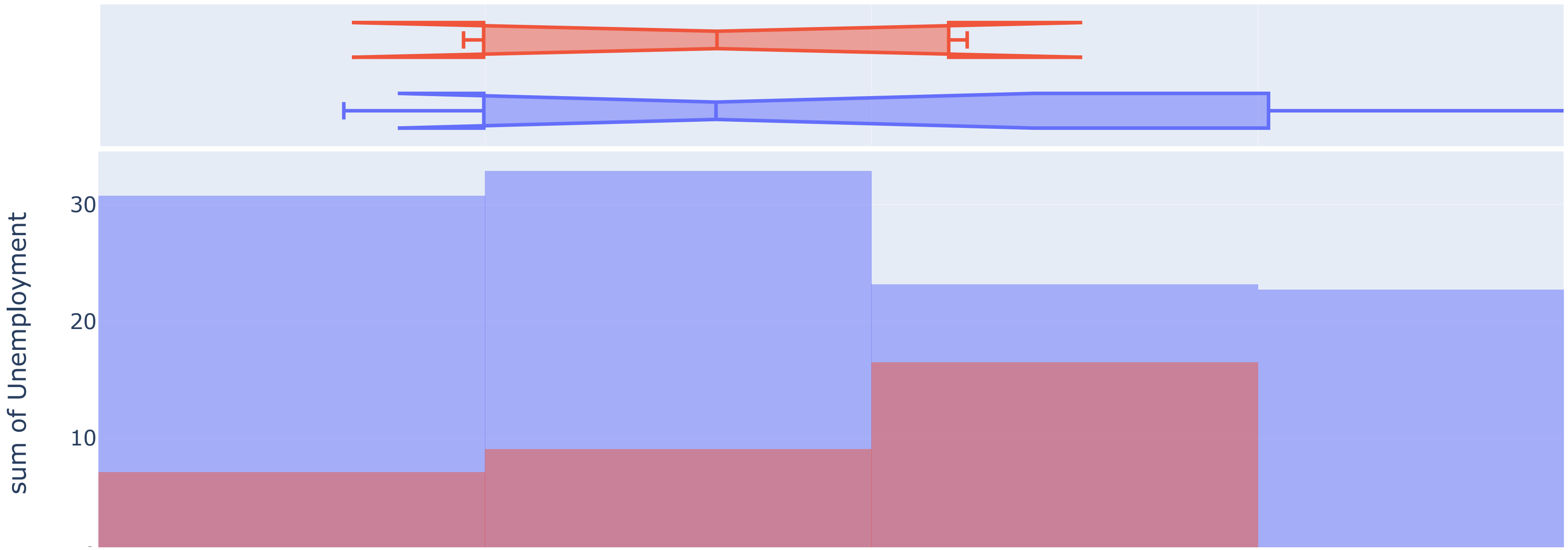


In [30]:
```
fig = px.histogram(df_new, x='Unemployment', y ='Weekly_Sales', marginal='box', color='Holiday_Flag',
                   barmode='overlay',title = 'Affect of Unemployment on sales',template='simple_white')
fig.show()
```

Affect of Unemployment on sales

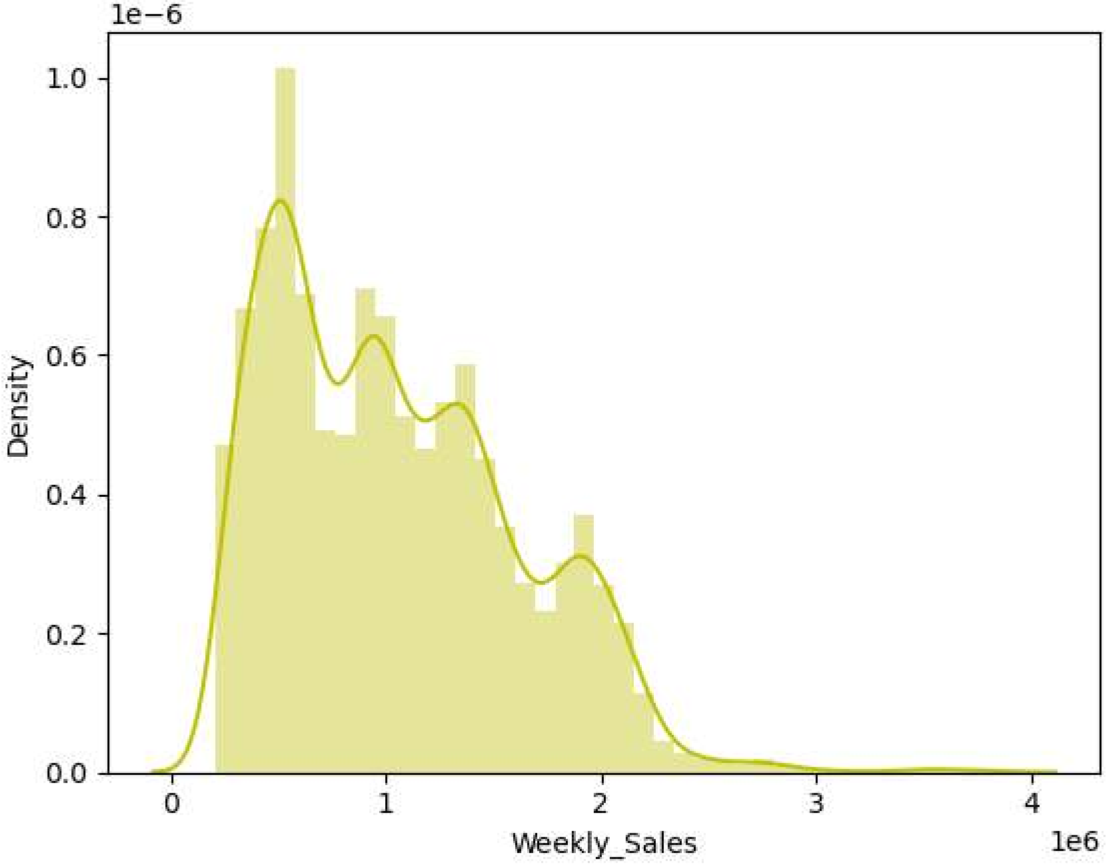



```
In [31]: fig = px.histogram(df_new, x = "Weekly_Sales", y = "Unemployment", marginal="box", color = "Holiday_Flag"
                          ,animation_frame="Fuel_Price",barmode="overlay"  )
         fig.show()
```



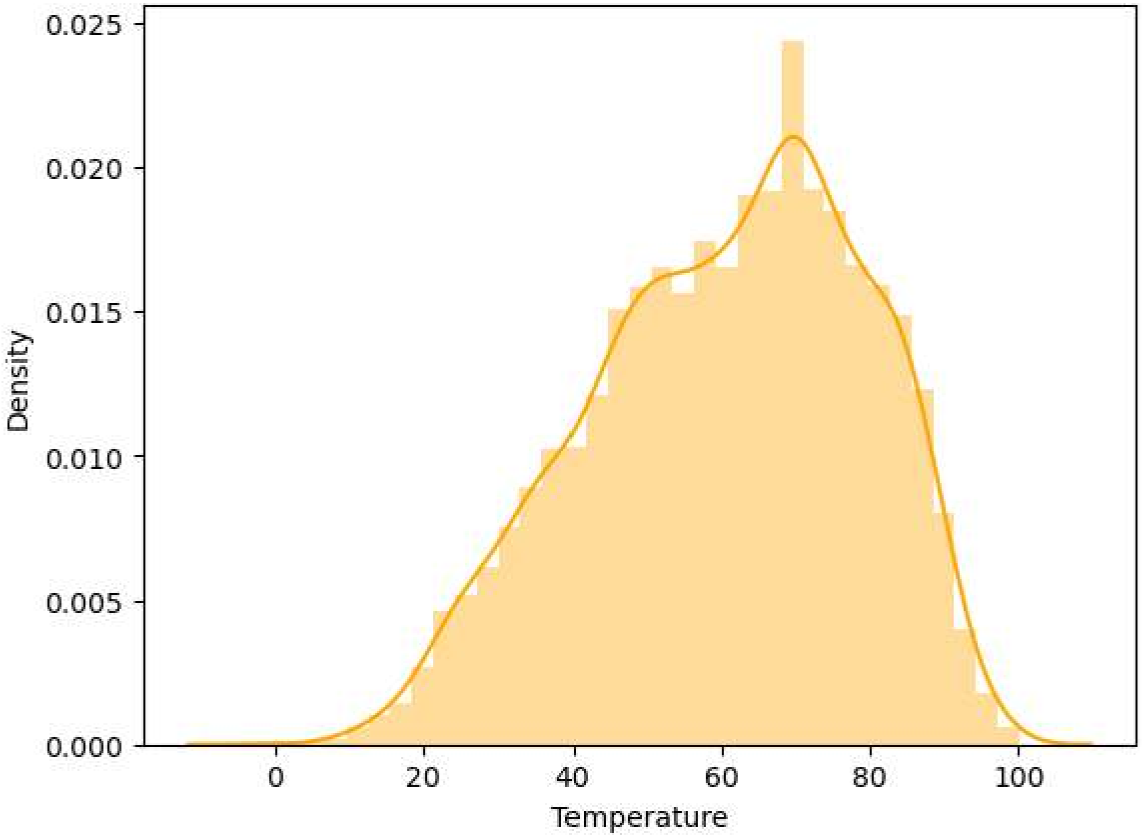

```
In [32]: sns.distplot(df_new["Weekly_Sales"],kde=True,color = "y")
```

```
Out[32]: <AxesSubplot:xlabel='Weekly_Sales', ylabel='Density'>
```



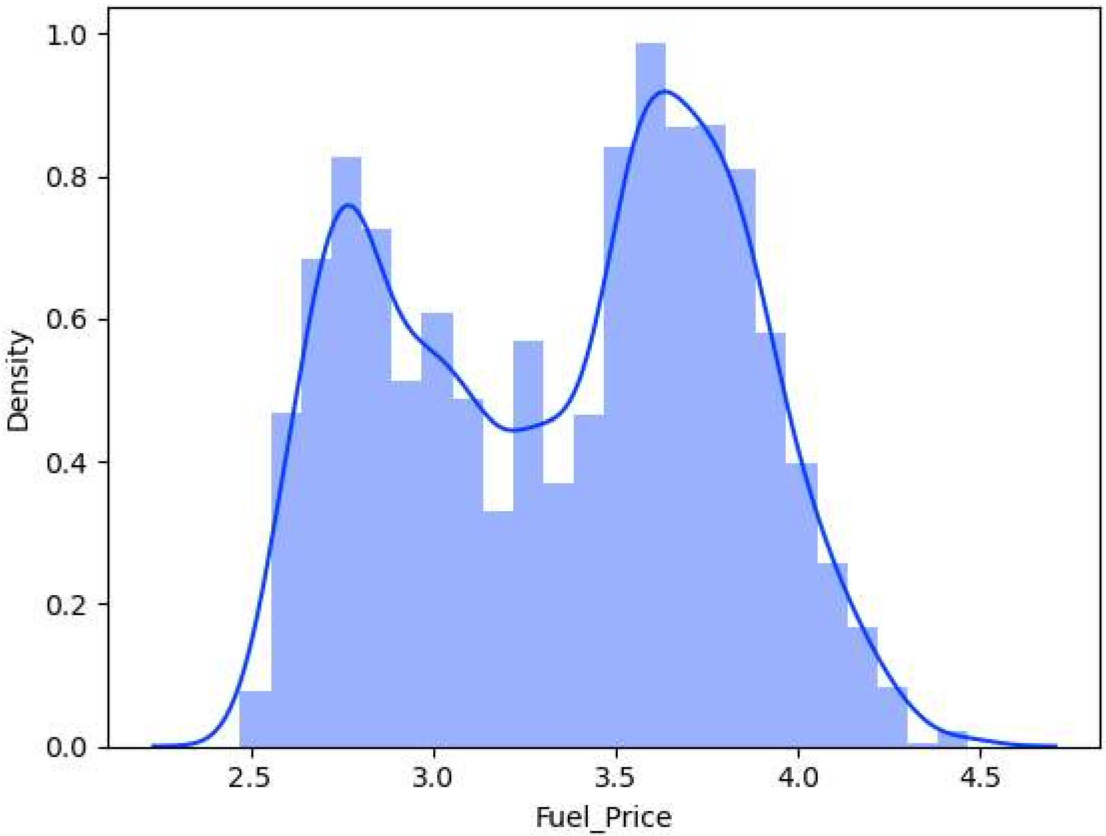
In [33]:
```
sns.distplot(df_new['Temperature'], kde = True,color="Orange")
```

Out[33]:
```
<AxesSubplot:xlabel='Temperature', ylabel='Density'>
```



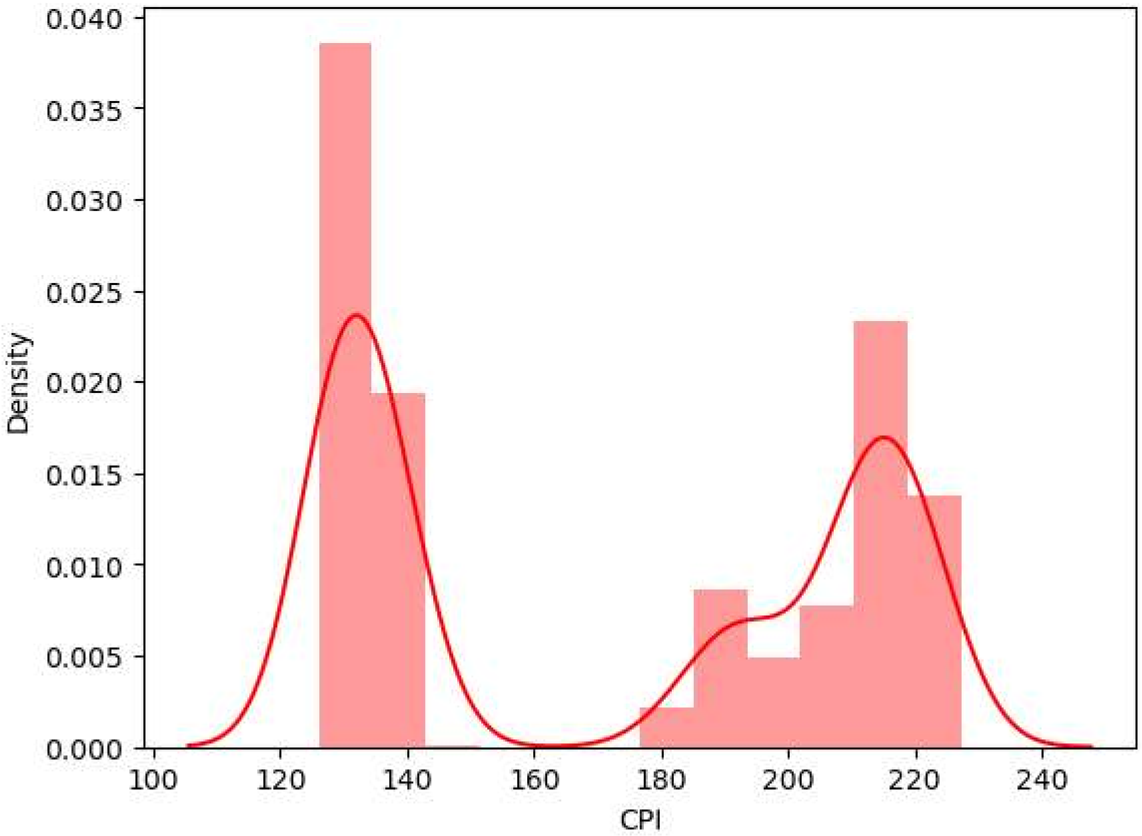
In [34]:
```
sns.distplot(df_new['Fuel_Price'], kde = True)
```

Out[34]:
```
<AxesSubplot:xlabel='Fuel_Price', ylabel='Density'>
```



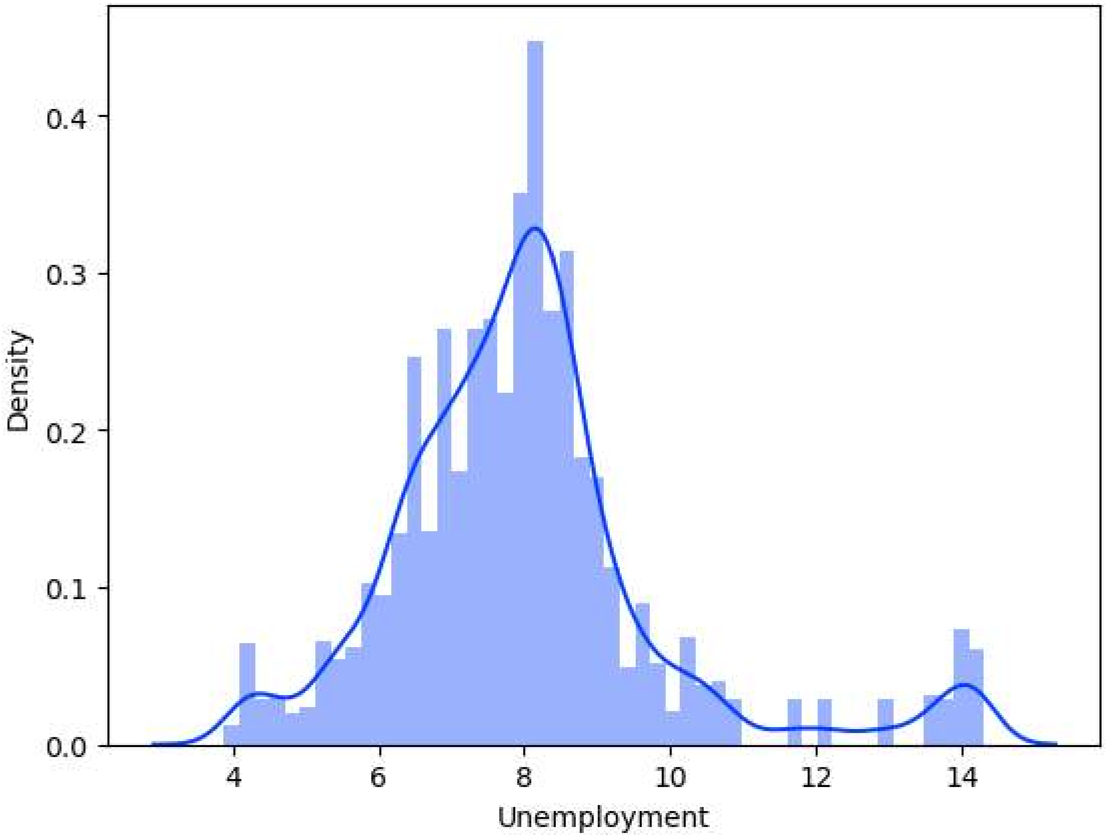
In [35]:
```
sns.distplot(df_new['CPI'], kde = True,color="red")
```

Out[35]: <AxesSubplot:xlabel='CPI', ylabel='Density'>



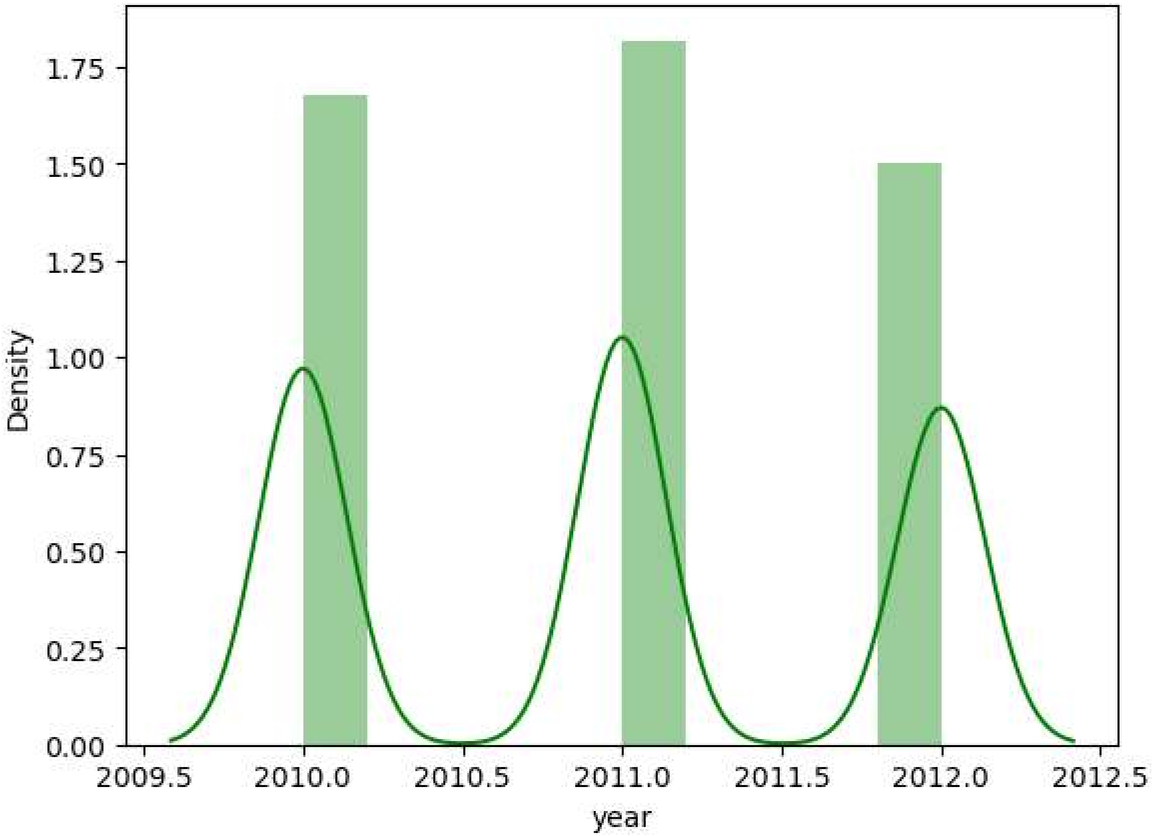

```
In [36]: sns.distplot(df_new['Unemployment'], kde=True)
```

Out[36]: <AxesSubplot:xlabel='Unemployment', ylabel='Density'>



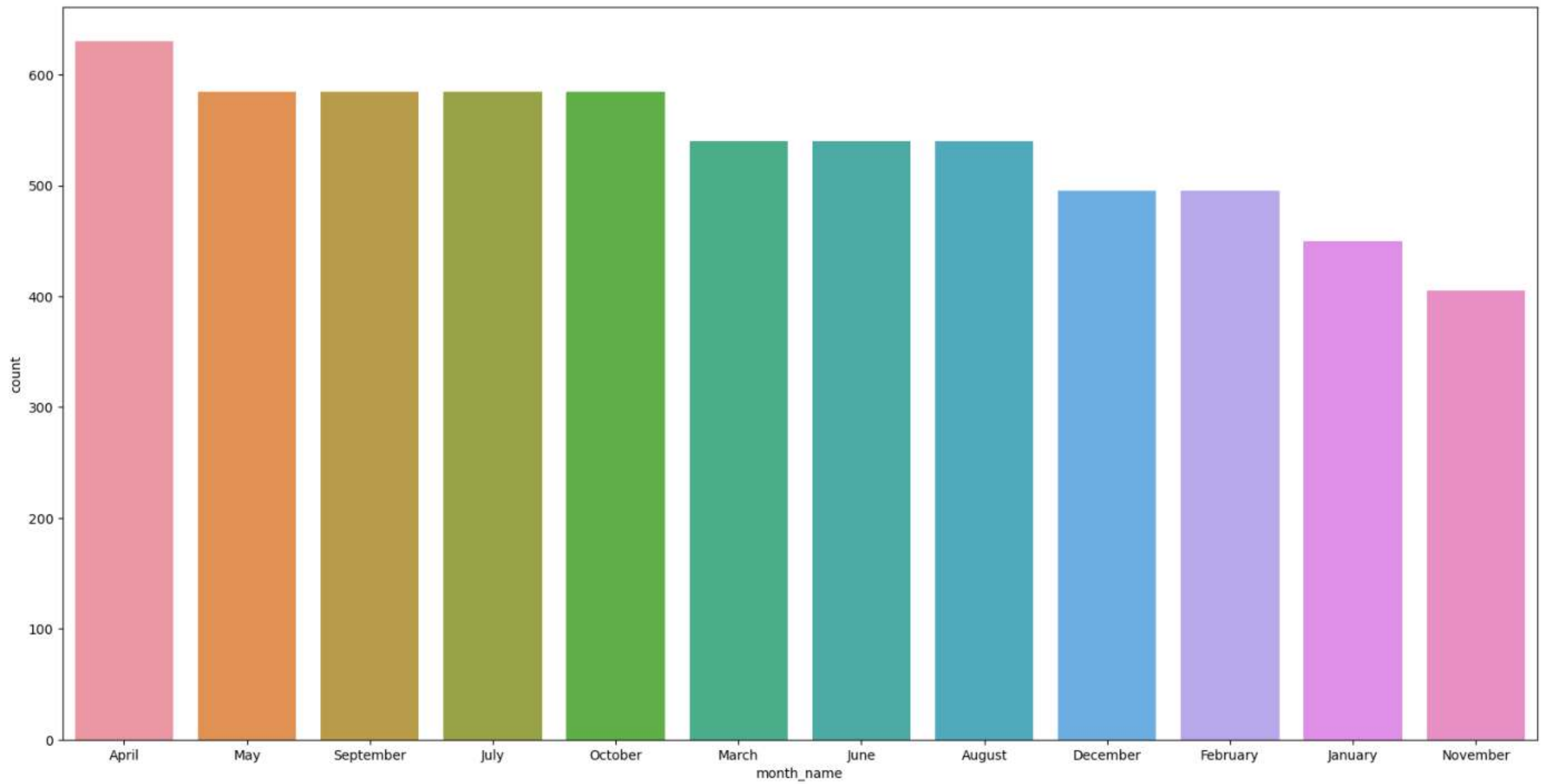
In [37]:
```
sns.distplot(df_new["year"],kde=True,color="g")
```

Out[37]:
```
<AxesSubplot:xlabel='year', ylabel='Density'>
```



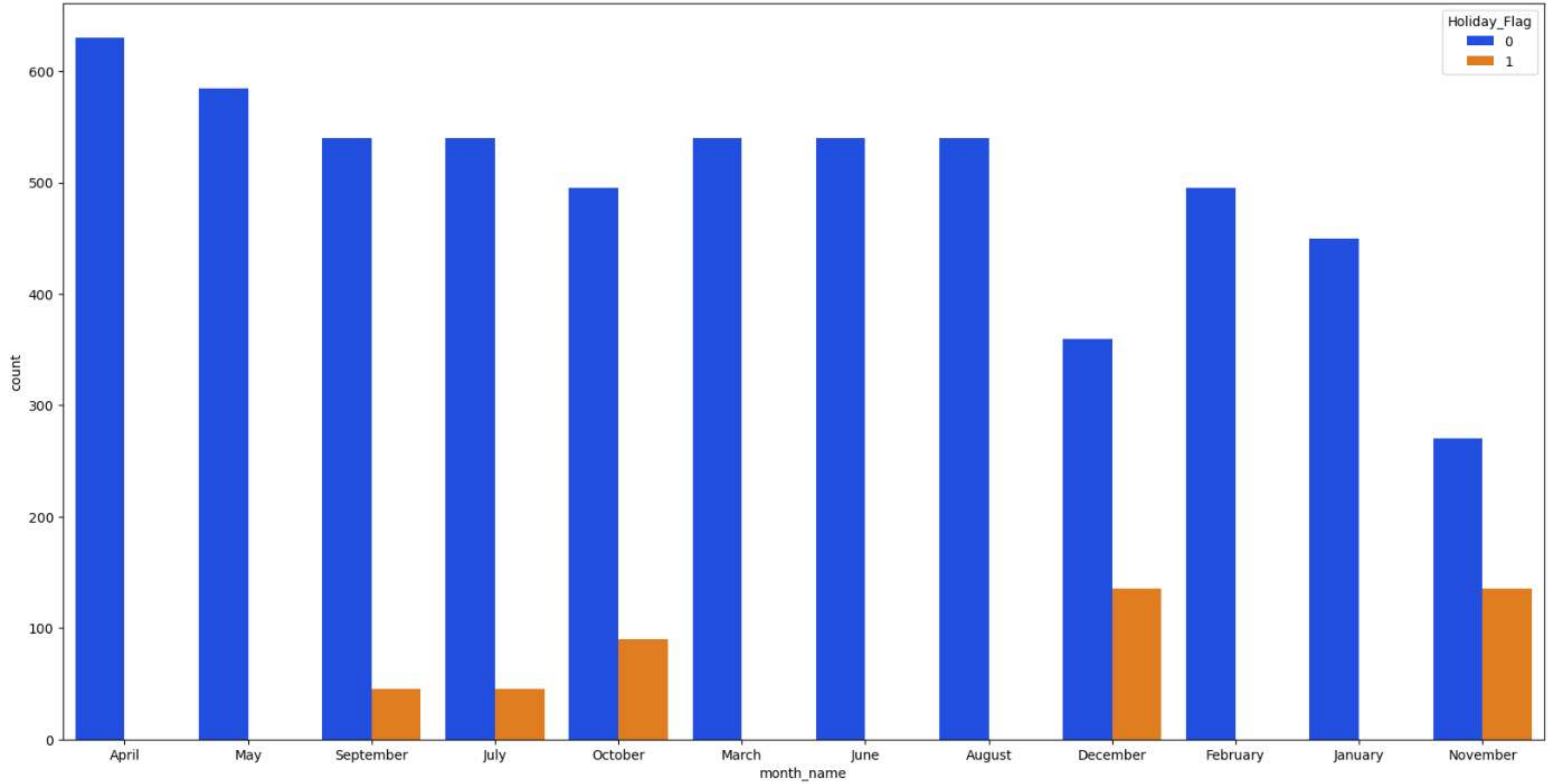
In [38]:
```
plt.figure(figsize=(20,10))
sns.countplot(df_new["month_name"],order=df_new["month_name"].value_counts().index)
```

Out[38]: <AxesSubplot:xlabel='month_name', ylabel='count'>



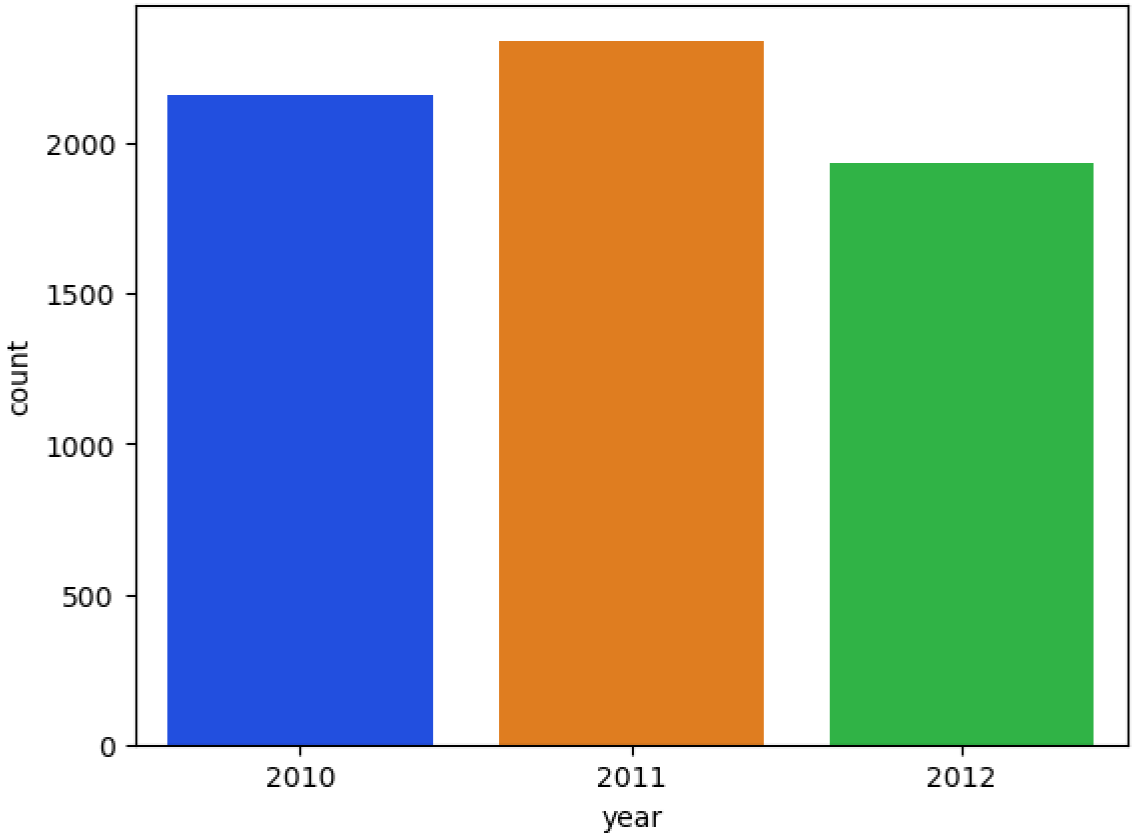
In [39]:
```
plt.figure(figsize=(20,10))
sns.countplot(df_new["month_name"],hue=df_new["Holiday_Flag"],order=df_new["month_name"].value_counts().index)
```

Out[39]:
```
<AxesSubplot:xlabel='month_name', ylabel='count'>
```



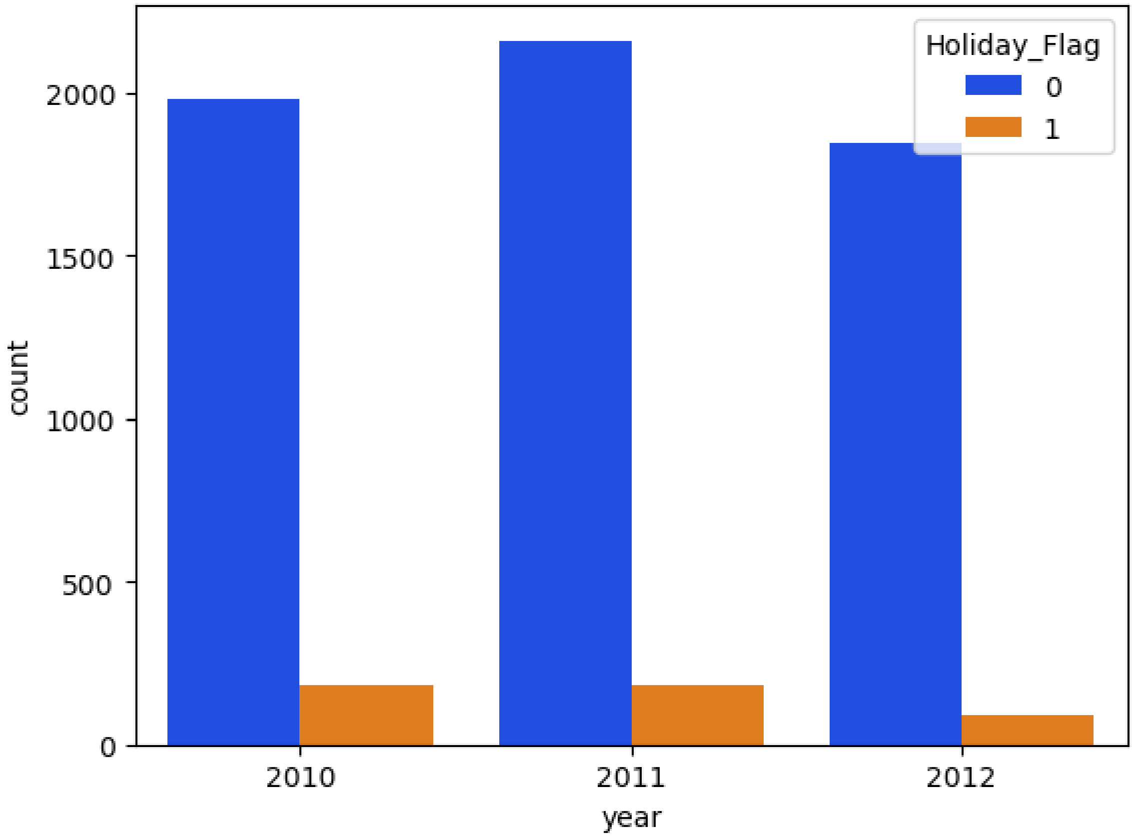
In [40]:
```
sns.countplot(df_new["year"])
```

Out[40]: <AxesSubplot:xlabel='year', ylabel='count'>



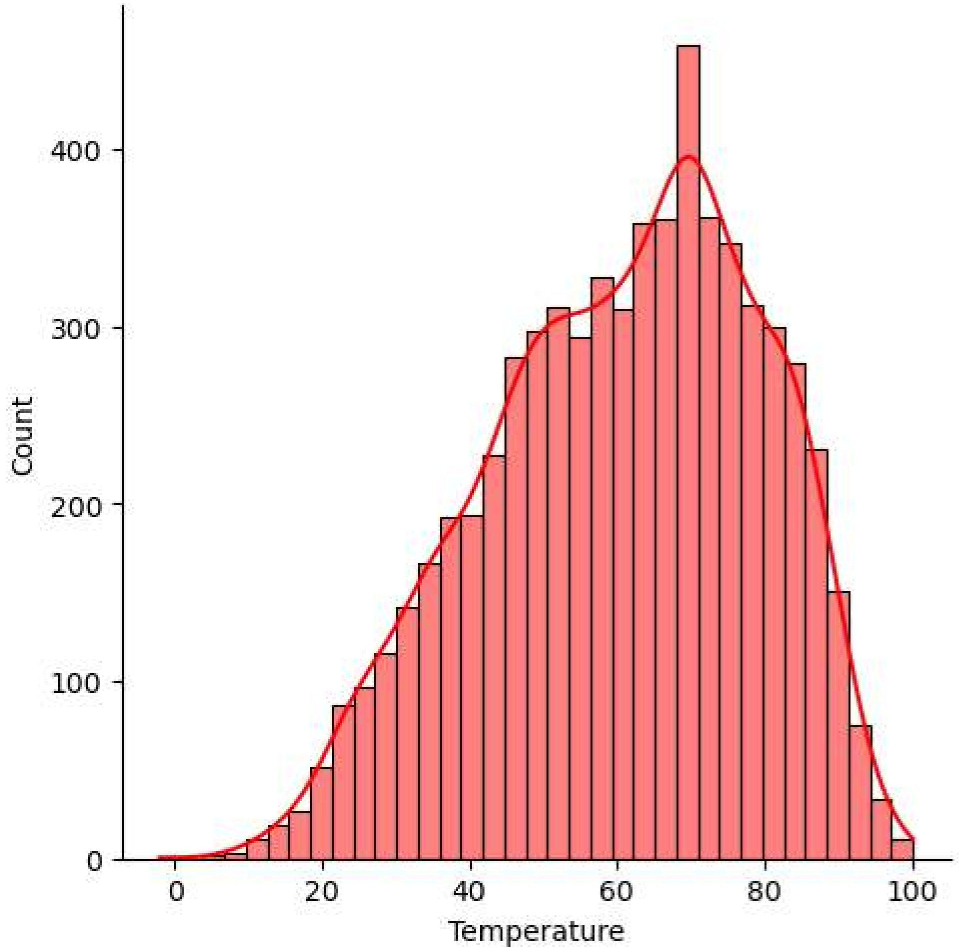
In [41]:
```
sns.countplot(df_new["year"], hue=df_new["Holiday_Flag"])
```

Out[41]:
```
<AxesSubplot:xlabel='year', ylabel='count'>
```

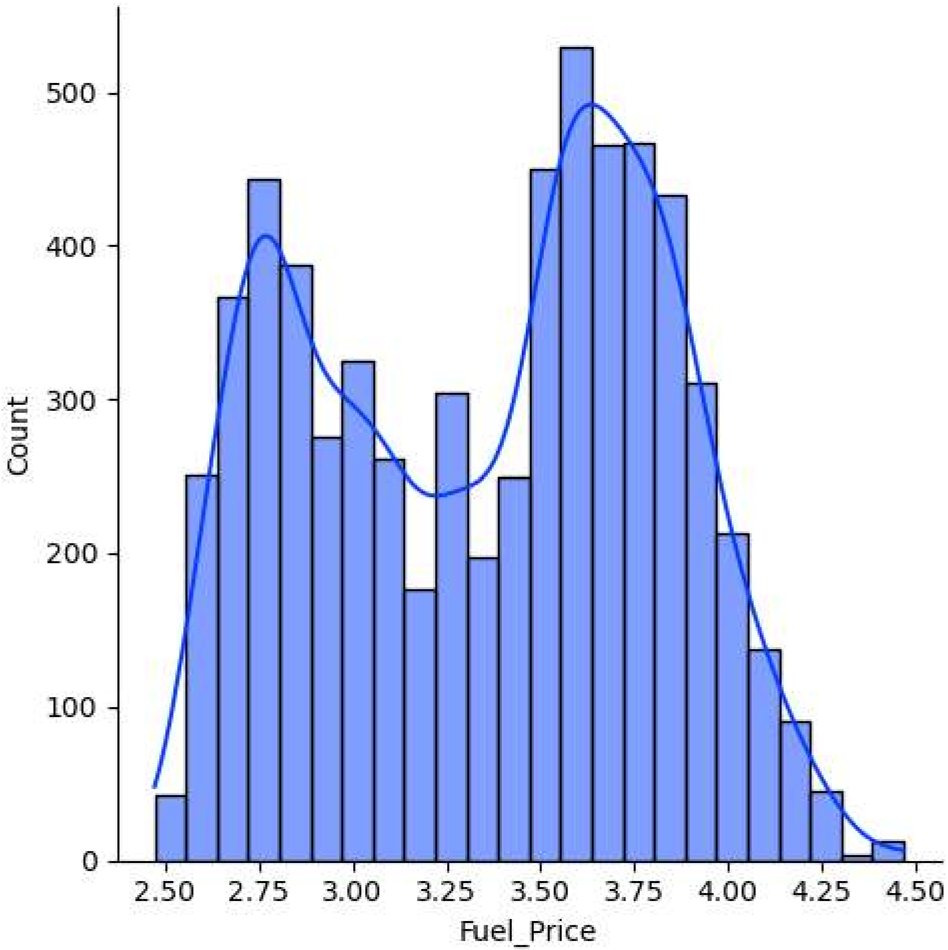


In [42]:
```
sns.displot(df['Temperature'],kde=True,color = 'r')#, ax=axes[1])
sns.displot(df['Fuel_Price'],kde=True)
sns.displot(df['Unemployment'],kde=False)
```

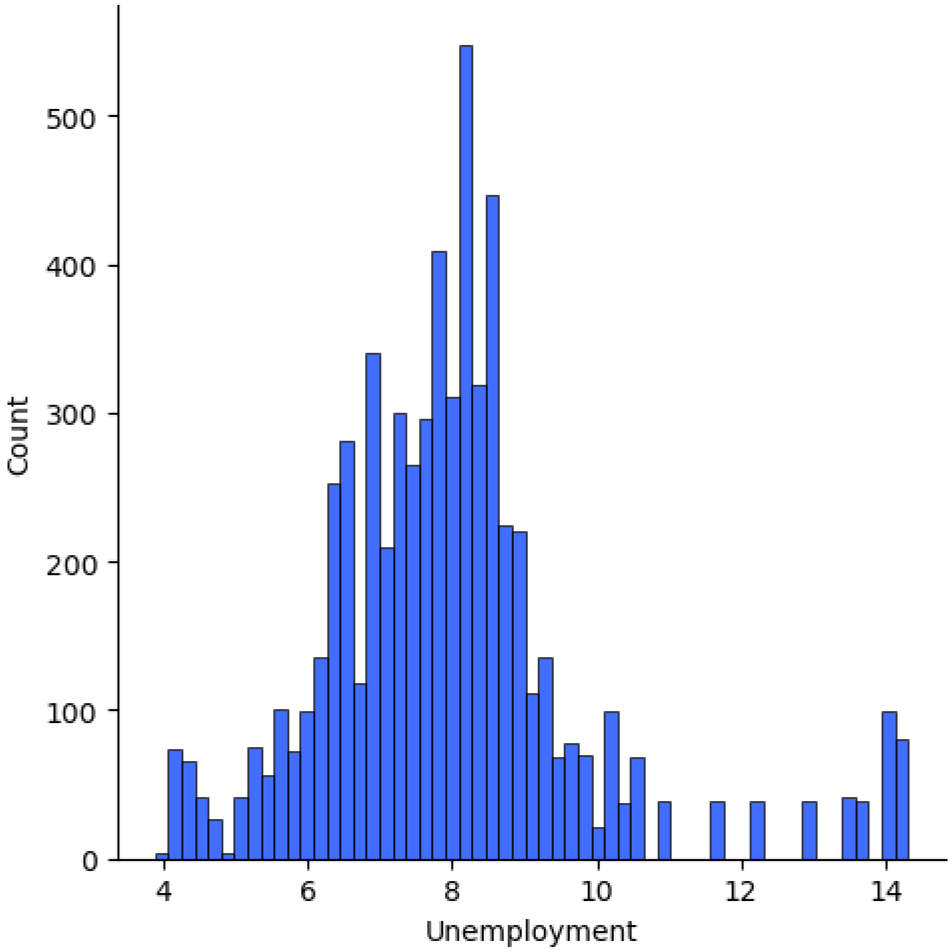
Out[42]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb230cd3790>
```



400
300
200
100
0
Count
0
20
40
60
80
100
Temperature

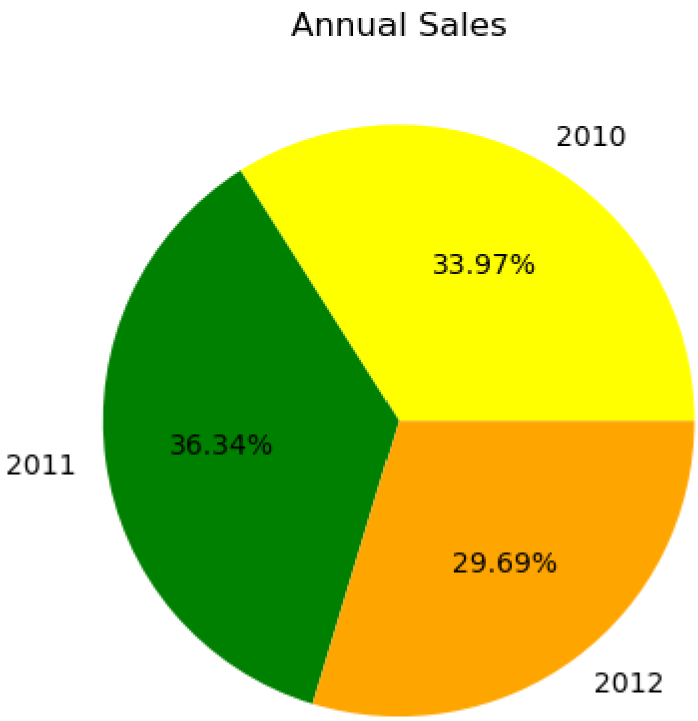


Count
500
400
300
200
100
0
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50
Fuel_Price



In [43]:
```
plt.pie(df_new.groupby('year')['Weekly_Sales'].sum(),labels=df_new['year'].unique(),normalize=True,
        autopct='%1.2f%%',colors=['Yellow','green','orange'])
plt.title('Annual Sales')
```

Out[43]:
```
Text(0.5, 1.0, 'Annual Sales')
```

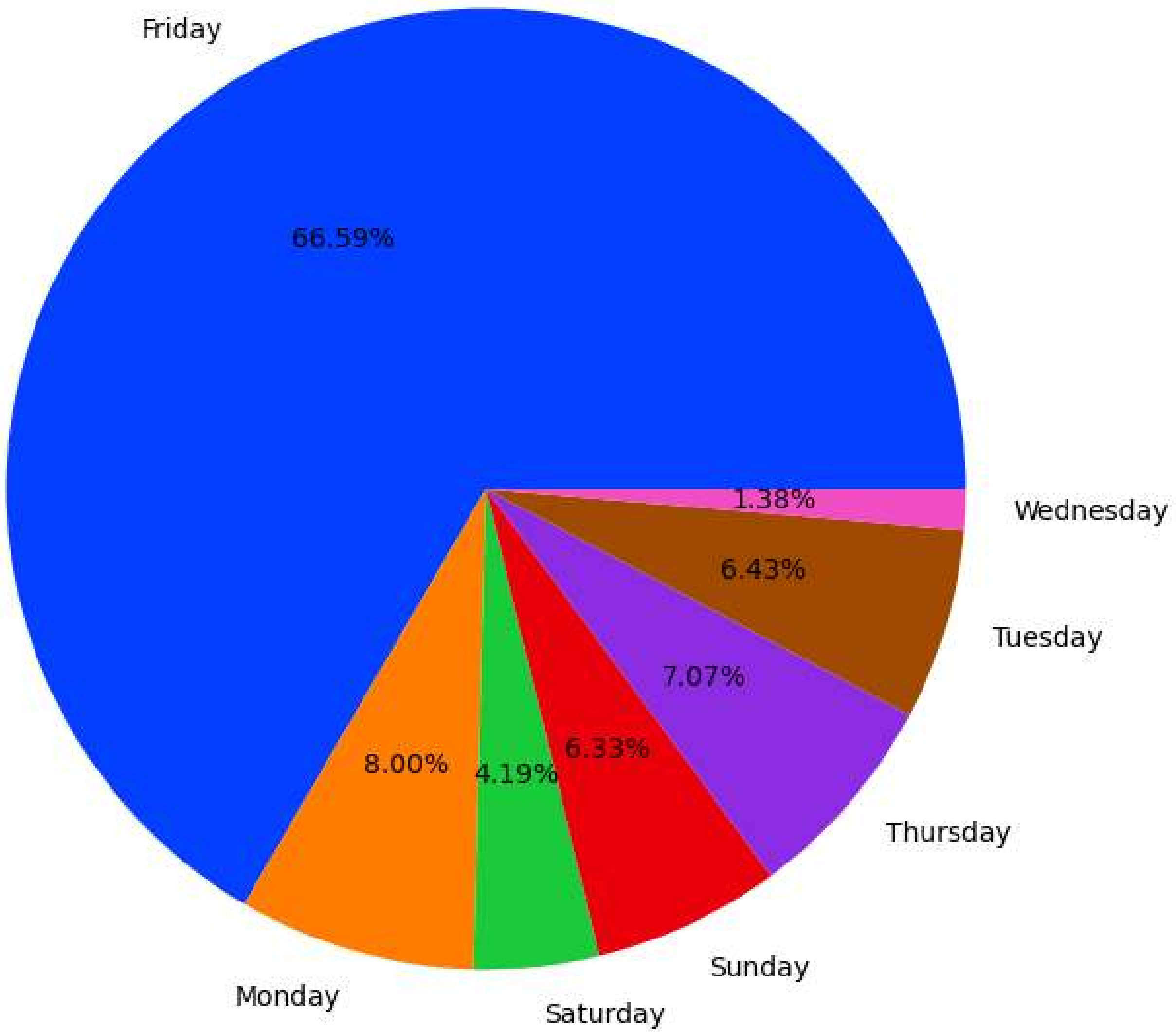


Observation:

The year 2011 has the highest percentage of 36.34% Weekly sales, followed by the year 2010 of about 33.97%.

In [44]:
```
df2 = df_new.groupby('Day_Name')['Weekly_Sales'].sum().reset_index()
plt.figure(figsize=(10,8))
plt.pie(df2['Weekly_Sales'],labels= df2['Day_Name'],autopct='%1.2f%%', normalize=True)
```

```
Out[44]: ([<matplotlib.patches.Wedge at 0x7fb216922d30>,
  <matplotlib.patches.Wedge at 0x7fb21692c490>,
  <matplotlib.patches.Wedge at 0x7fb21692cbb0>,
  <matplotlib.patches.Wedge at 0x7fb216937310>,
  <matplotlib.patches.Wedge at 0x7fb216937a30>,
  <matplotlib.patches.Wedge at 0x7fb216942190>,
  <matplotlib.patches.Wedge at 0x7fb2169428e0>],
 [Text(-0.5476787396163149, 0.9539643589633131, 'Friday'),
  Text(-0.30095024733679887, -1.0580306936133375, 'Monday'),
  Text(0.11632476607467147, -1.0938320478015229, 'Saturday'),
  Text(0.46515280776702955, -0.9968113489655147, 'Sunday'),
  Text(0.8320368042989099, -0.7195239789555715, 'Thursday'),
  Text(1.0545042229298378, -0.31308280665526667, 'Tuesday'),
  Text(1.0989722017440637, -0.04754050687366608, 'Wednesday')],
 [Text(-0.2987338579725354, 0.5203441957981707, '66.59%'),
  Text(-0.16415468036552663, -0.5771076510618204, '8.00%'),
  Text(0.06344987240436625, -0.5966356624371942, '4.19%'),
  Text(0.2537197133274706, -0.543715281253917, '6.33%'),
  Text(0.45383825689031443, -0.39246762488485715, '7.07%'),
  Text(0.5751841215980933, -0.1707724399937818, '6.43%'),
  Text(0.5994393827694892, -0.02593118556745422, '1.38%')])
```

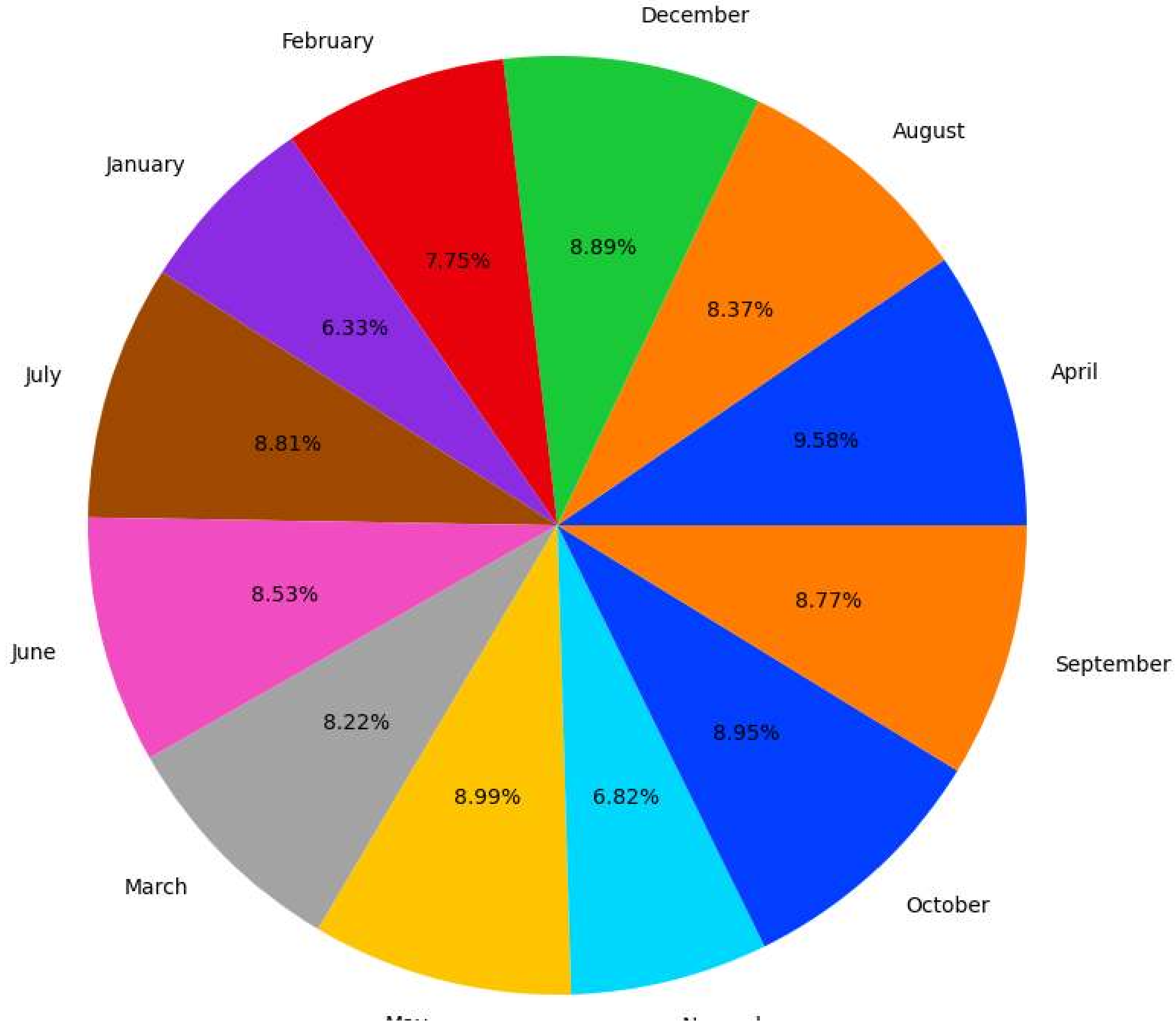


Observation:

From all the week days Friday has the major percentage of Weekly_sales.

In [45]:
```
plt.figure(figsize=(10,10))
df3 = df_new.groupby('month_name')['Weekly_Sales'].sum().reset_index()
plt.pie(df3['Weekly_Sales'],labels=df3['month_name'],normalize=True,autopct='%1.2f%%')
```

Out[45]:
```
([<matplotlib.patches.Wedge at 0x7fb2322588b0>,
  <matplotlib.patches.Wedge at 0x7fb23223db80>,
  <matplotlib.patches.Wedge at 0x7fb23223a880>,
  <matplotlib.patches.Wedge at 0x7fb232233130>,
  <matplotlib.patches.Wedge at 0x7fb232218160>,
  <matplotlib.patches.Wedge at 0x7fb232209ee0>,
  <matplotlib.patches.Wedge at 0x7fb232200b80>,
  <matplotlib.patches.Wedge at 0x7fb2321e8fa0>,
  <matplotlib.patches.Wedge at 0x7fb2321dea90>,
  <matplotlib.patches.Wedge at 0x7fb2321c77c0>,
  <matplotlib.patches.Wedge at 0x7fb232258880>,
  <matplotlib.patches.Wedge at 0x7fb2321af130>],
 [Text(1.0505715420703448, 0.3260359412579202, 'April'),
  Text(0.7135560893168025, 0.8371605027704733, 'August'),
  Text(0.17894470808660848, 1.0853473137423792, 'December'),
  Text(-0.38685422481986037, 1.0297299688457286, 'February'),
  Text(-0.7903461501230193, 0.7650836313670042, 'January'),
  Text(-1.0529134511925988, 0.3183916837759601, 'July'),
  Text(-1.0654902855032773, -0.27336870980078276, 'June'),
  Text(-0.7840328017971336, -0.771552049900806, 'March'),
  Text(-0.2752227535923997, -1.0650128806286887, 'May'),
  Text(0.26538381737518074, -1.0675071098008562, 'November'),
  Text(0.7408373645676252, -0.8131174572351131, 'October'),
  Text(1.0585579767623141, -0.29909030380952856, 'September')],
 [Text(0.5730390229474608, 0.1778377861406837, '9.58%'),
  Text(0.3892124123546195, 0.4566330015111672, '8.37%'),
  Text(0.09760620441087733, 0.592007625677661 4, '8.89%'),
  Text(-0.21101139535628746, 0.5616708920976701, '7.75%'),
  Text(-0.4310979000671014, 0.4173183443820022, '6.33%'),
  Text(-0.5743164279232357, 0.17366819115052368, '8.81%'),
  Text(-0.5811765193654239, -0.14911020534588149, '8.53%'),
  Text(-0.4276542555257092, -0.4208465726731672, '8.22%'),
  Text(-0.15012150195949073, -0.5809161167065574, '8.99%'),
  Text(0.1447548094773713, -0.5822766053459214, '6.82%'),
  Text(0.4040931079459773, -0.44351861303733436, '8.95%'),
  Text(0.5773952600521713, -0.16314016571428827, '8.77%')])
```

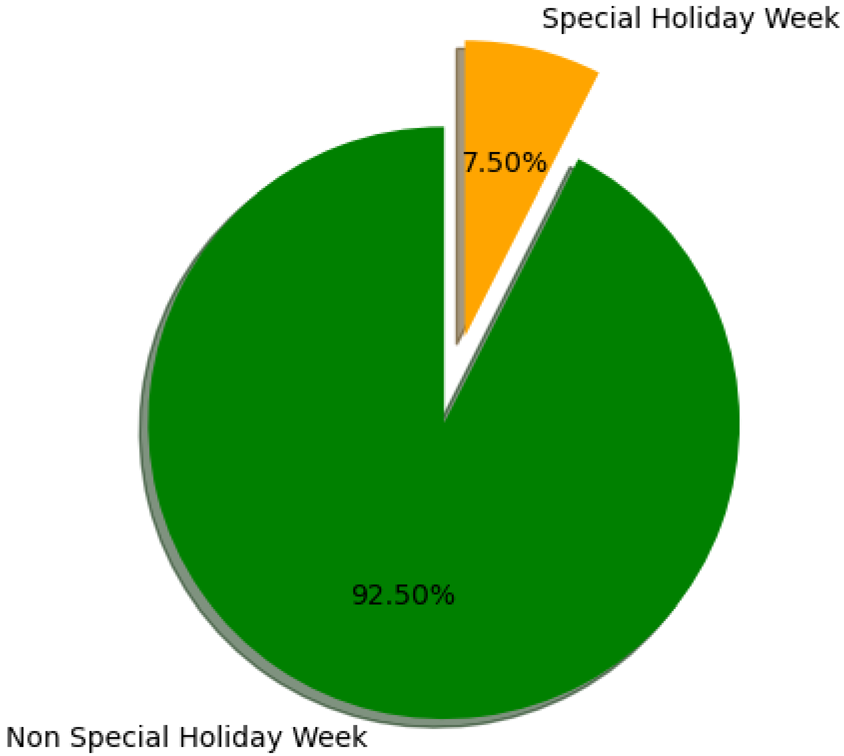


December
February
August
January
8.89%
7.75%
8.37%
6.33%
April
July
9.58%
8.81%
8.53%
8.77%
June
September
8.22%
8.95%
8.99%
6.82%
March
October

May November

In [46]:
```
df4 = df_new.groupby('Holiday_Flag')['Weekly_Sales'].sum().reset_index()
plt.pie(df4['Weekly_Sales'],labels= ['Non Special Holiday Week','Special Holiday Week'],
        normalize=True,autopct='%1.2f%%',startangle=90,explode=[0,0.3],shadow=True,colors=['green','Orange'])
```

Out[46]:
```
([<matplotlib.patches.Wedge at 0x7fb231fbd250>,
  <matplotlib.patches.Wedge at 0x7fb231fa6f70>],
 [Text(-0.25679386593526243, -1.0696059603508306, 'Non Special Holiday Week'),
  Text(0.3268284610531304, 1.361316699760068, 'Special Holiday Week')],
 [Text(-0.14006938141923403, -0.5834214329186348, '92.50%'),
  Text(0.2101040106770124, 0.8751321641314721, '7.50%')])
```



In [47]:
```
plt.figure(figsize=(30,25))
df5 = df_new.groupby('Store')['Weekly_Sales'].sum().reset_index()
```

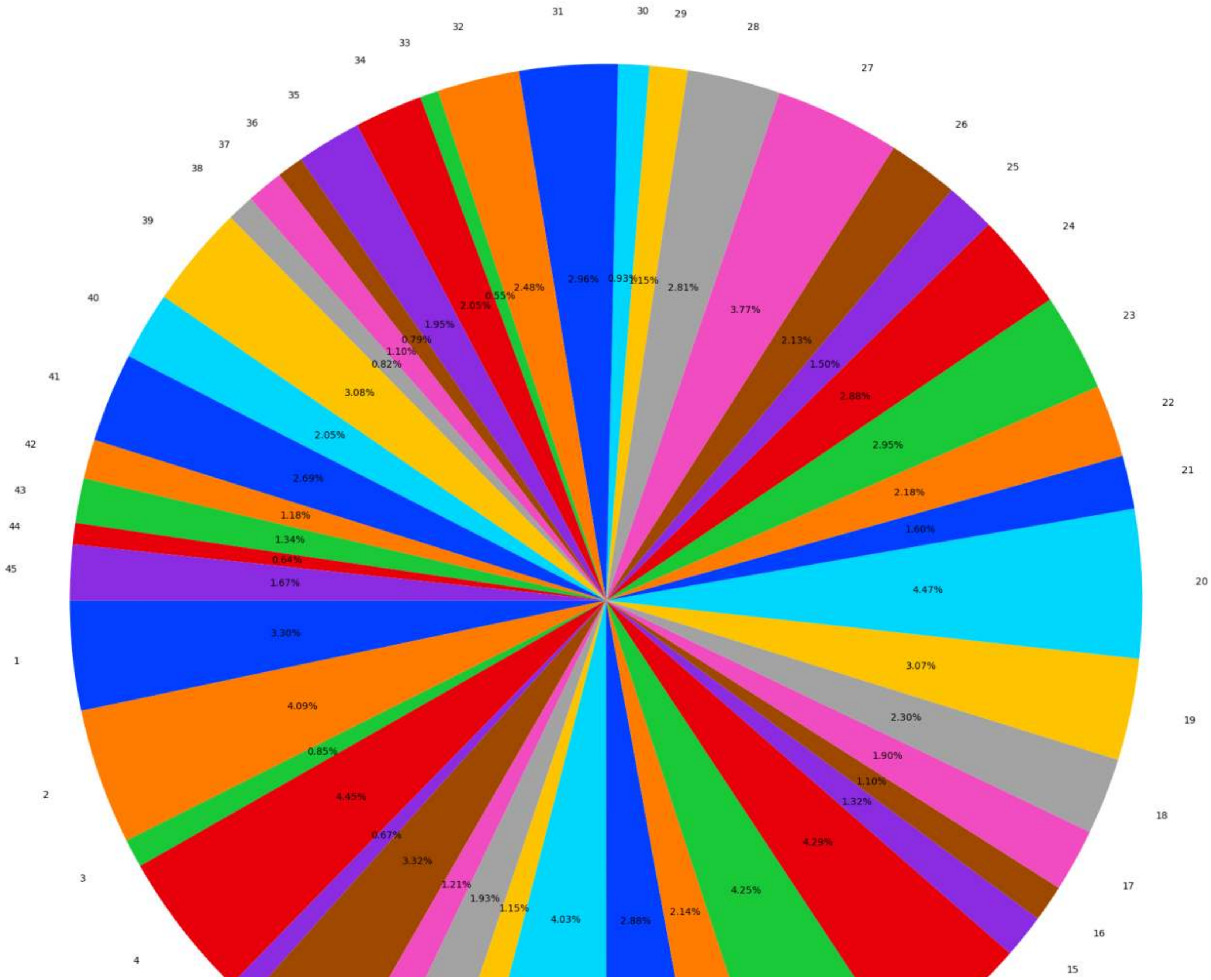
```
plt.pie(df5['Weekly_Sales'],labels=df5['Store'], normalize=True, autopct='%1.2f%%',startangle = 180)
```

Out[47]:
```
([<matplotlib.patches.Wedge at 0x7fb231d30280>,
  <matplotlib.patches.Wedge at 0x7fb231d26220>,
  <matplotlib.patches.Wedge at 0x7fb231cf5b50>,
  <matplotlib.patches.Wedge at 0x7fb231d04520>,
  <matplotlib.patches.Wedge at 0x7fb231cee9a0>,
  <matplotlib.patches.Wedge at 0x7fb231ce5eb0>,
  <matplotlib.patches.Wedge at 0x7fb231cda730>,
  <matplotlib.patches.Wedge at 0x7fb231cbec40>,
  <matplotlib.patches.Wedge at 0x7fb231cb5190>,
  <matplotlib.patches.Wedge at 0x7fb231cb0190>,
  <matplotlib.patches.Wedge at 0x7fb231d309d0>,
  <matplotlib.patches.Wedge at 0x7fb231ca5910>,
  <matplotlib.patches.Wedge at 0x7fb231c9e9a0>,
  <matplotlib.patches.Wedge at 0x7fb231c992b0>,
  <matplotlib.patches.Wedge at 0x7fb231c8e8e0>,
  <matplotlib.patches.Wedge at 0x7fb231c887c0>,
  <matplotlib.patches.Wedge at 0x7fb231c80c40>,
  <matplotlib.patches.Wedge at 0x7fb231c7e5b0>,
  <matplotlib.patches.Wedge at 0x7fb231c7b400>,
  <matplotlib.patches.Wedge at 0x7fb231c73a90>,
  <matplotlib.patches.Wedge at 0x7fb231c6e1f0>,
  <matplotlib.patches.Wedge at 0x7fb231bfed30>,
  <matplotlib.patches.Wedge at 0x7fb231bfaf10>,
  <matplotlib.patches.Wedge at 0x7fb231bf7af0>,
  <matplotlib.patches.Wedge at 0x7fb231bf30a0>,
  <matplotlib.patches.Wedge at 0x7fb231bed670>,
  <matplotlib.patches.Wedge at 0x7fb231be6640>,
  <matplotlib.patches.Wedge at 0x7fb231be46a0>,
  <matplotlib.patches.Wedge at 0x7fb231be23d0>,
  <matplotlib.patches.Wedge at 0x7fb231bdec10>,
  <matplotlib.patches.Wedge at 0x7fb231bda9d0>,
  <matplotlib.patches.Wedge at 0x7fb231bd22e0>,
  <matplotlib.patches.Wedge at 0x7fb231bcf370>,
  <matplotlib.patches.Wedge at 0x7fb231bca370>,
  <matplotlib.patches.Wedge at 0x7fb231bc8250>,
  <matplotlib.patches.Wedge at 0x7fb231bc60a0>,
  <matplotlib.patches.Wedge at 0x7fb231bc3d60>,
  <matplotlib.patches.Wedge at 0x7fb231bc2880>,
  <matplotlib.patches.Wedge at 0x7fb231bc0850>,
  <matplotlib.patches.Wedge at 0x7fb230a8f610>,
  <matplotlib.patches.Wedge at 0x7fb230cff850>,
  <matplotlib.patches.Wedge at 0x7fb230d58520>,
  <matplotlib.patches.Wedge at 0x7fb230d7ed60>,
  <matplotlib.patches.Wedge at 0x7fb231100850>,
```

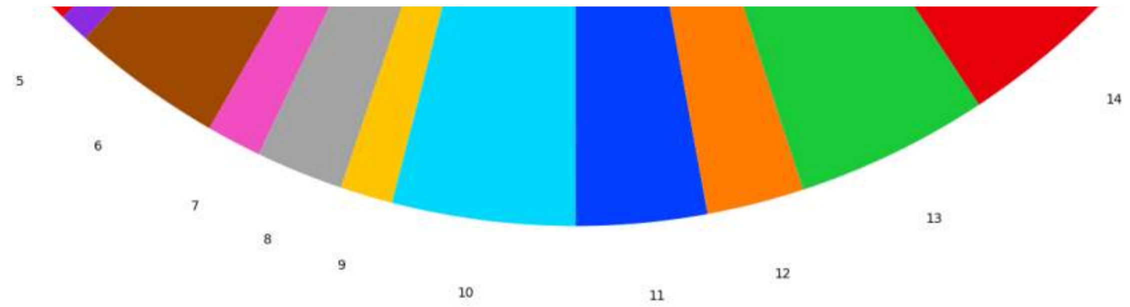
```
 <matplotlib.patches.Wedge at 0x7fb230b92880>],
[Text(-1.0940899331227671, -0.11387369423804206, '1'),
 Text(-1.0385520375997501, -0.3625047105865O735, '2'),
 Text(-0.9700005022718932, -0.5187475547819722, '3'),
 Text(-0.8705921022444361, -0.6723610574011656, '4'),
 Text(-0.751643177977816, -0.8031391741158, '5'),
 Text(-0.6451629674046351, -0.8909347593901844, '6'),
 Text(-0.5122036889645173, -0.9734718182932366, '7'),
 Text(-0.41383976218286195, -1.019184306804433, '8'),
 Text(-0.3133253343227192, -1.0544321859994394, '9'),
 Text(-0.13813631695101694, -1.0912920589554422, '10'),
 Text(0.10017134122314016, -1.095429460256368, '11'),
 Text(0.27099158097791487, -1.0660973515768108, '12'),
 Text(0.47826856910583454, -0.990585269325896, '13'),
 Text(0.7238092216160834, -0.8283116627837132, '14'),
 Text(0.85788312415197, -0.6885031193068453, '15'),
 Text(0.9077991874955498, -0.621209010867051, '16'),
 Text(0.9622108914623131, -0.533057408119708, '17'),
 Text(1.0239659273912294, -0.40186288649465973, '18'),
 Text(1.0768994397552685, -0.22424896132376865, '19'),
 Text(1.0994509824243348, 0.034749636633571925, '20'),
 Text(1.0728704383614798, 0.2427941978096806, '21'),
 Text(1.0364593898024501, 0.3684452921267051, '22'),
 Text(0.9638537026054148, 0.530081163571988, '23'),
 Text(0.8511915315841465, 0.6967589084894682, '24'),
 Text(0.7475778130767046, 0.8069246640148333, '25'),
 Text(0.6509342321933937, 0.886726917014927, '26'),
 Text(0.4764689577214625, 0.9914521331500696, '27'),
 Text(0.26292602013089034, 1.0681151192348748, '28'),
 Text(0.128553389708541, 1.092462368228052, '29'),
 Text(0.05708396391606562, 1.0985178291969728, '30'),
 Text(-0.07738491013968342, 1.0972746127030704, '31'),
 Text(-0.2628374780490863 6, 1.0681369107622845, '32'),
 Text(-0.36309368064278275, 1.038346271278169, '33'),
 Text(-0.4467159743317225, 1.0052088530633125, '34'),
 Text(-0.5693031841643095, 0.9412193604576876, '35'),
 Text(-0.6482531408456551, 0.8886888462131971, '36'),
 Text(-0.699958039185949, 0.8485627515858575, '37'),
 Text(-0.7498410683952885, 0.8048219505877138, '38'),
 Text(-0.8425336115738177, 0.7072037283332008, '39'),
 Text(-0.9450185537225316, 0.5629741851276795, '40'),
 Text(-1.018058391904619, 0.41660186110071745, '41'),
 Text(-1.0610929764077532, 0.28996843865865013, '42'),
 Text(-1.0807355296290426, 0.20496515556902126, '43'),
```

```
 Text(-1.0914162521029793, 0.13715161189532687, '44'),
 Text(-1.098489574186852, 0.05762512822362057, '45')],
[Text(-0.5967763271578729, -0.06211292412984112, '3.30%'),
 Text(-0.5664829295998637, -0.19772984213809489, '4.09%'),
 Text(-0.5290911830573962, -0.2829532116992575, '0.85%'),
 Text(-0.47486841940605595, -0.3667423949460903, '4.45%'),
 Text(-0.40998718798789957, -0.4380759131540727, '0.67%'),
 Text(-0.3519070731298009, -0.4859644142128278, '3.32%'),
 Text(-0.2793838303442821, -0.5309846281599472, '1.21%'),
 Text(-0.22573077937247013, -0.5559187128024179, '1.93%'),
 Text(-0.17090472781239227, -0.5751448287269668, '1.15%'),
 Text(-0.07534708197328195, -0.5952502139756957, '4.03%'),
 Text(0.05463891339444008, -0.5975069783216551, '2.88%'),
 Text(0.14781358962431718, -0.581507646314624, '2.14%'),
 Text(0.2608737649668188, -0.540319237814125, '4.25%'),
 Text(0.39480502997240907, -0.45180636151838893, '4.29%'),
 Text(0.46793624953743806, -0.37554715598555194, '1.32%'),
 Text(0.4951631931793907, -0.33884127865475505, '1.10%'),
 Text(0.5248423044339889, -0.2907585862471134, '1.90%'),
 Text(0.5585268694861251, -0.2191979380879962, '2.30%'),
 Text(0.5873996944119646, -0.12231761526751014, '3.07%'),
 Text(0.5997005358678189, 0.018954347254675593, '4.47%'),
 Text(0.5852020572880798, 0.13243319880528032, '1.60%'),
 Text(0.5653414853467909, 0.20097015934183912, '2.18%'),
 Text(0.5257383832393171, 0.28913518013017525, '2.95%'),
 Text(0.46428628995498894, 0.38005031372152803, '2.88%'),
 Text(0.40776971622365704, 0.44014072582627267, '1.50%'),
 Text(0.3550550357418511, 0.4836692274626874, '2.13%'),
 Text(0.2598921587571613, 0.5407920726273107, '3.77%'),
 Text(0.14341419279866743, 0.5826082468553863, '2.81%'),
 Text(0.07012003075011326, 0.5958885644880283, '1.15%'),
 Text(0.031136707590581243, 0.5991915431983487, '0.93%'),
 Text(-0.04220995098528186, 0.5985134251107657, '2.96%'),
 Text(-0.14336589711768347, 0.5826201331430642, '2.48%'),
 Text(-0.19805109853242692, 0.5663706934244558, '0.55%'),
 Text(-0.24366325872639405, 0.548295738034534, '2.05%'),
 Text(-0.31052900954416873, 0.5133923784314659, '1.95%'),
 Text(-0.3535926222794482, 0.48473937066174383, '0.79%'),
 Text(-0.3817952941014267, 0.4628524099559222, '1.10%'),
 Text(-0.4090042191247028, 0.43899379122966203, '0.82%'),
 Text(-0.4595637881311732, 0.3857474881817458, '3.08%'),
 Text(-0.5154646656668354, 0.30707682825146154, '2.05%'),
 Text(-0.5553045774025194, 0.22723737878220948, '2.69%'),
 Text(-0.5787779871315017, 0.15816460290471823, '1.18%'),
```

```
Text(-0.5894921070703868, 0.11179917576492067, '1.34%'),
Text(-0.5953179556925341, 0.07480997012472373, '0.64%'),
Text(-0.5991761313746464, 0.03143188812197485, '1.67%')])
```

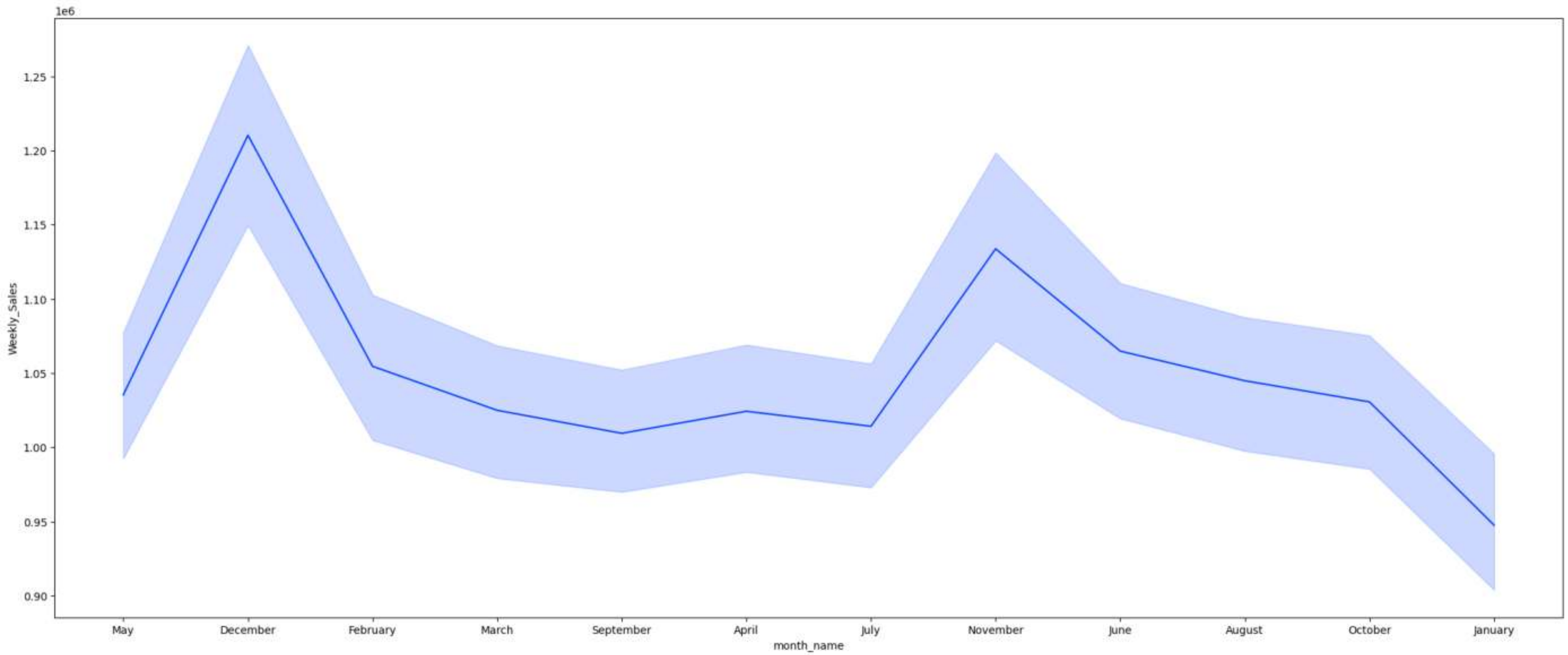


1
3.30%
2
4.09%
3
0.85%
4
4.45%
0.67%
3.32%
1.21%
1.93%
1.15%
4.03%
2.88%
2.14%
4.25%
4.29%
15
1.32%
16
1.10%
17
1.90%
18
2.30%
19
3.07%
20
4.47%
21
1.60%
22
2.18%
23
2.95%
24
2.88%
25
1.50%
26
2.13%
27
3.77%
28
2.81%
29
1.15%
30
0.93%
31
2.96%
32
2.48%
33
0.55%
34
2.05%
35
1.95%
36
0.79%
37
1.10%
38
0.82%
39
3.08%
40
2.05%
41
2.69%
42
1.18%
43
1.34%
44
0.64%
45
1.67%



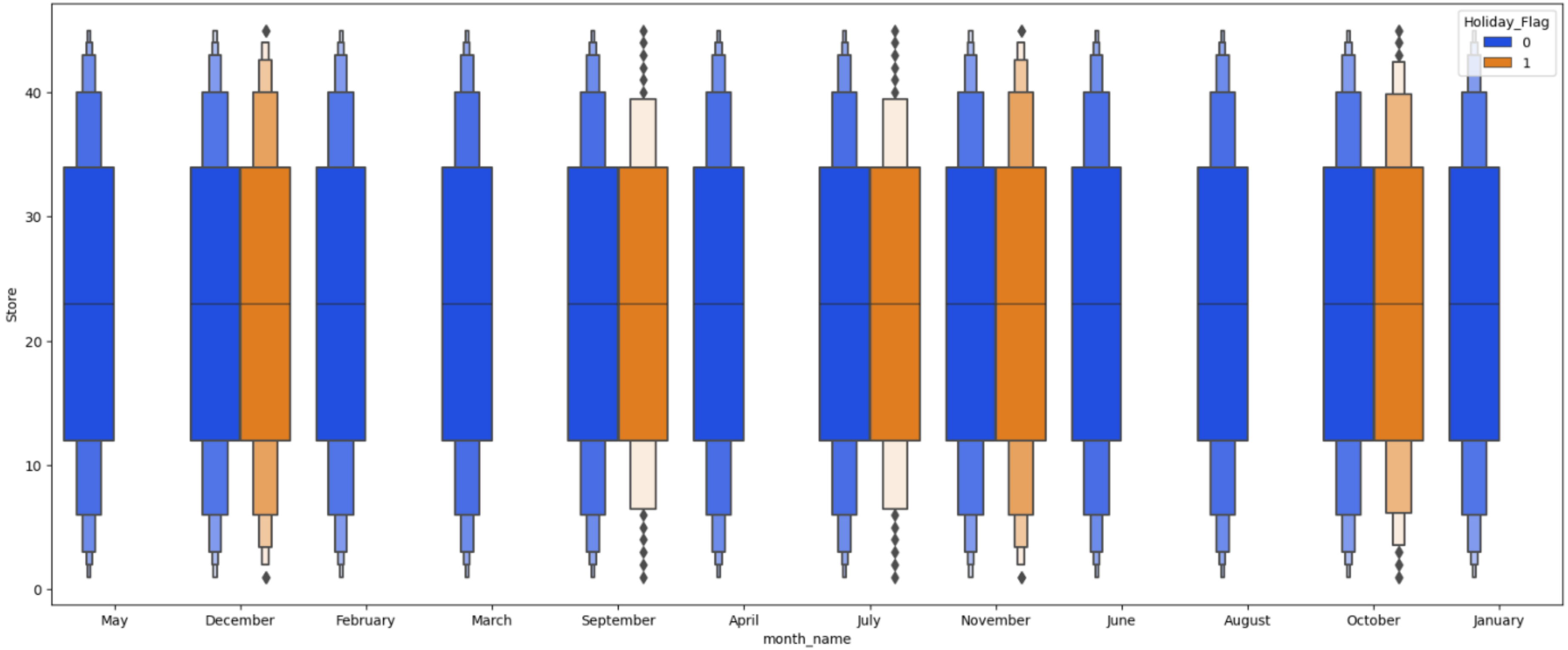
In [48]:
```python
# Checking for any kind of trend in between month and Weekly sales
plt.figure(figsize=(25,10))
sns.lineplot(x=df_new["month_name"], y = df_new["Weekly_Sales"],)
```

Out[48]: <AxesSubplot:xlabel='month_name', ylabel='Weekly_Sales'>



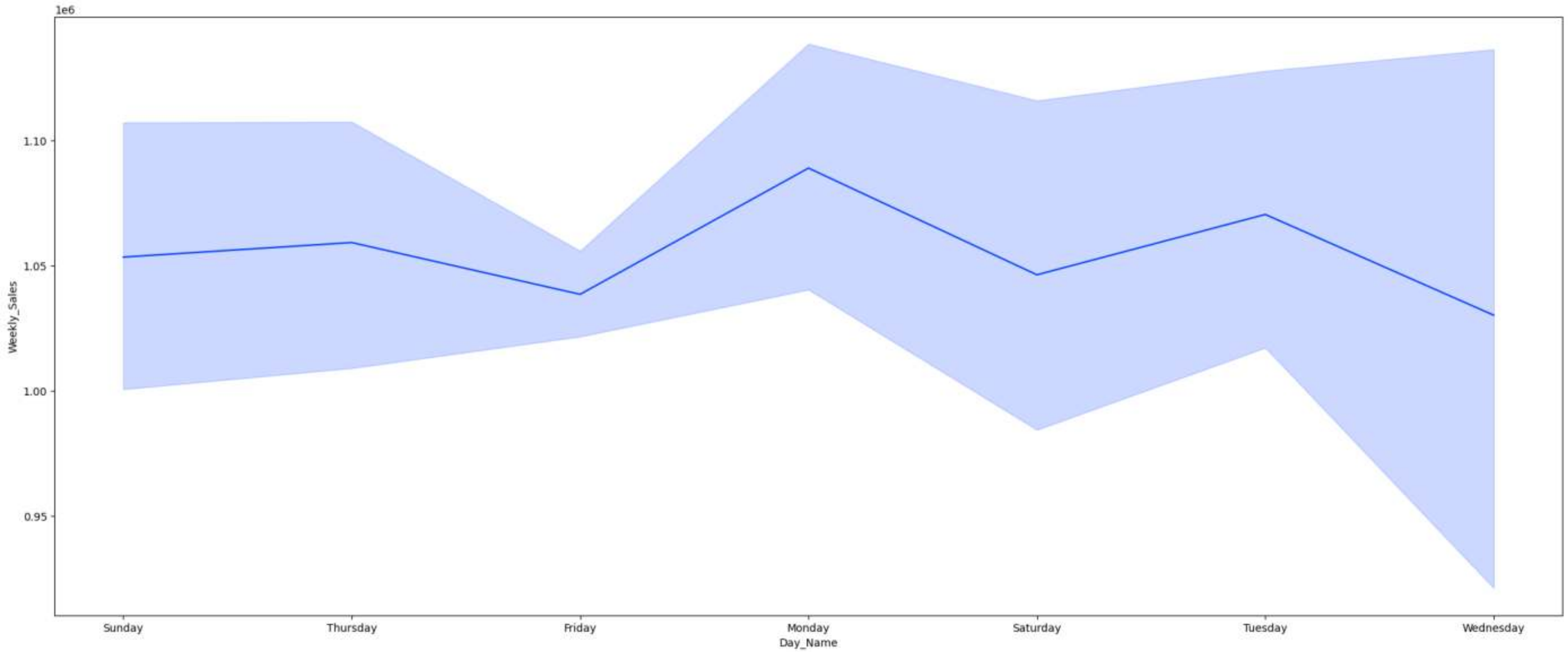
In [49]:
```
plt.figure(figsize=(20,8))
sns.boxenplot(x=df_new["month_name"],y = df_new["Store"],hue=df_new["Holiday_Flag"])
```

Out[49]: <AxesSubplot:xlabel='month_name', ylabel='Store'>



In [50]:
```
# Checking for any kind of trend in between month and Weekly sales
plt.figure(figsize=(25,10))
sns.lineplot(x=df_new["Day_Name"], y = df_new["Weekly_Sales"],)
```

Out[50]: <AxesSubplot:xlabel='Day_Name', ylabel='Weekly_Sales'>



Stores had the highest and lowest average revenues over the years

> - **Identifying the top performing and low performing stores or products in sales analysis can be useful for a variety of purposes.**
> - **By analysing the sales data for different stores, businesses can identify opportunities for growth, understand customer preferences, optimise inventory levels, and identify potential problems or areas for improvement.**
> - **Understanding the performance of different stores can inform product development and marketing efforts, as well as help businesses allocate resources more effectively and make more informed business decisions.**

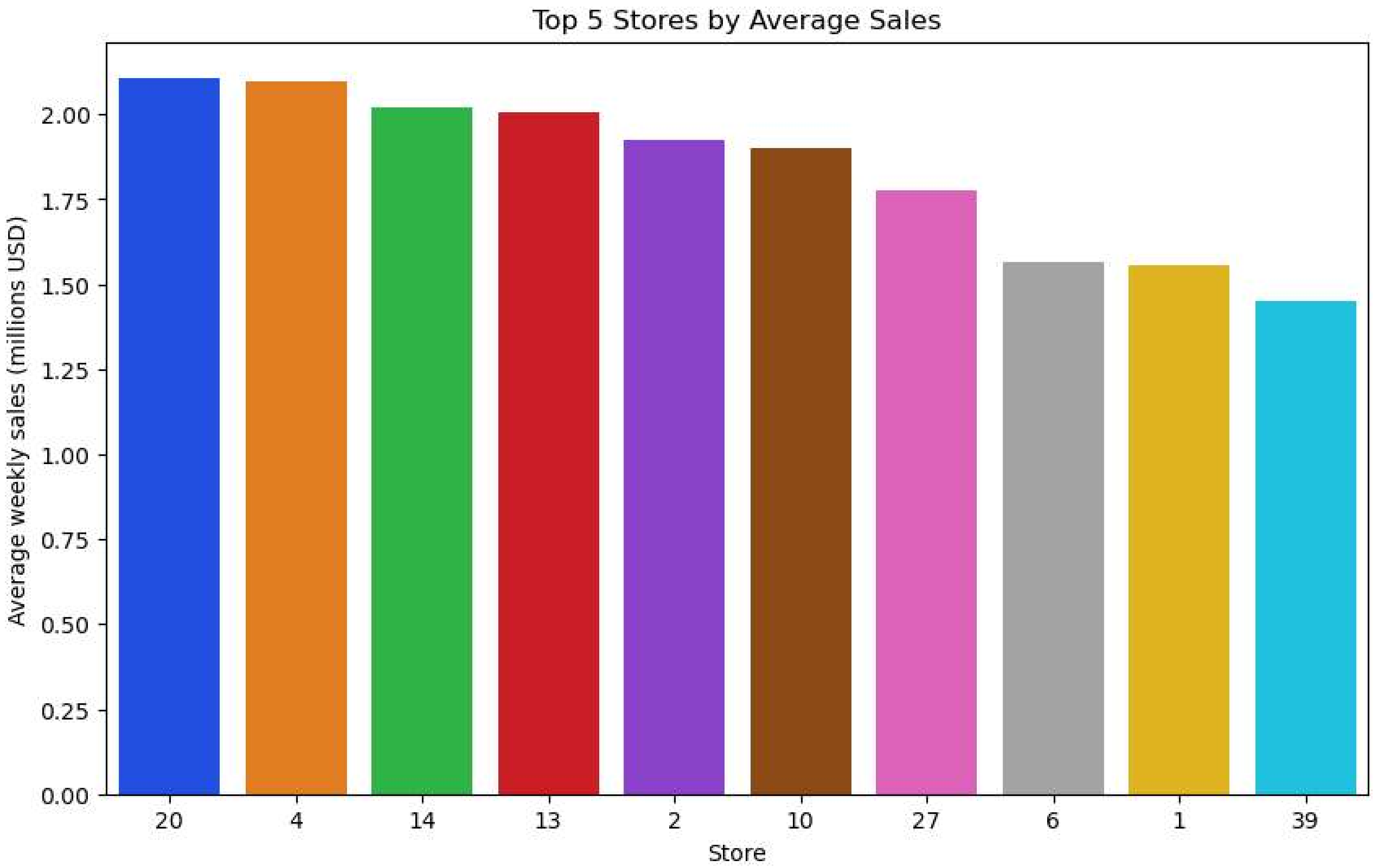
Creating a function that takes a dataframe as input and generates two plots showing the top and bottom performing stores in terms of average sales.

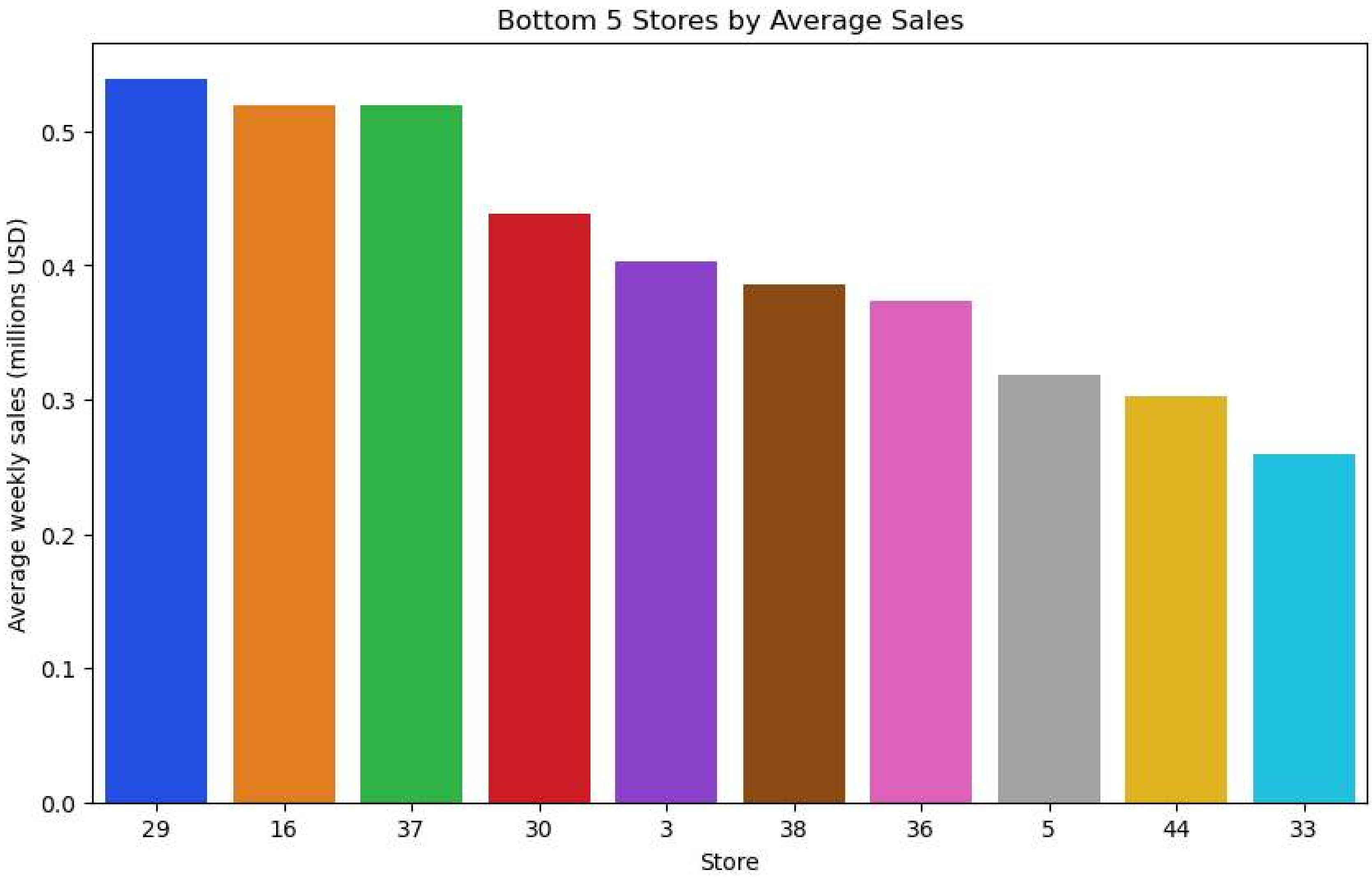
In [51]:
```
def plot_top_and_bottom_stores(df, col):
    """
    Plot the top and bottom 10 stores based on their average weekly sales.
```

```
    Parameters:
    df (pandas DataFrame): The dataframe containing the sales data.
    col (str): The name of the column to group the data by.

    Returns:
    None
    """
    # Group the data by the specified column and sort it by sales in descending order
    df = df.groupby(col).mean().sort_values(by='Weekly_Sales', ascending=False)

    # Select the top 5 and bottom 5 products
    top_stores = df.head(10)
    bottom_stores = df.tail(10)

    # Set the color palette
    sns.set_palette("bright")

    # Create a bar chart of the top 5 products
    fig, ax = plt.subplots(figsize=(10, 6))
    sns.barplot(x=top_stores.index, y=top_stores['Weekly_Sales']/1e6, order=top_stores.index)
    plt.title('Top 5 Stores by Average Sales')
    plt.ylabel('Average weekly sales (millions USD)')
    plt.show()

    # Create a bar chart of the bottom 5 products
    fig, ax = plt.subplots(figsize=(10, 6))
    sns.barplot(x=bottom_stores.index, y=bottom_stores['Weekly_Sales']/1e6, order=bottom_stores.index)
    plt.title('Bottom 5 Stores by Average Sales')
    plt.ylabel('Average weekly sales (millions USD)')
    plt.show()
```

In [52]:

```
plot_top_and_bottom_stores(df_new, 'Store')
```


Top 5 Stores by Average Sales
Average weekly sales (millions USD)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
20
4
14
13
2
10
27
6
1
39
Store



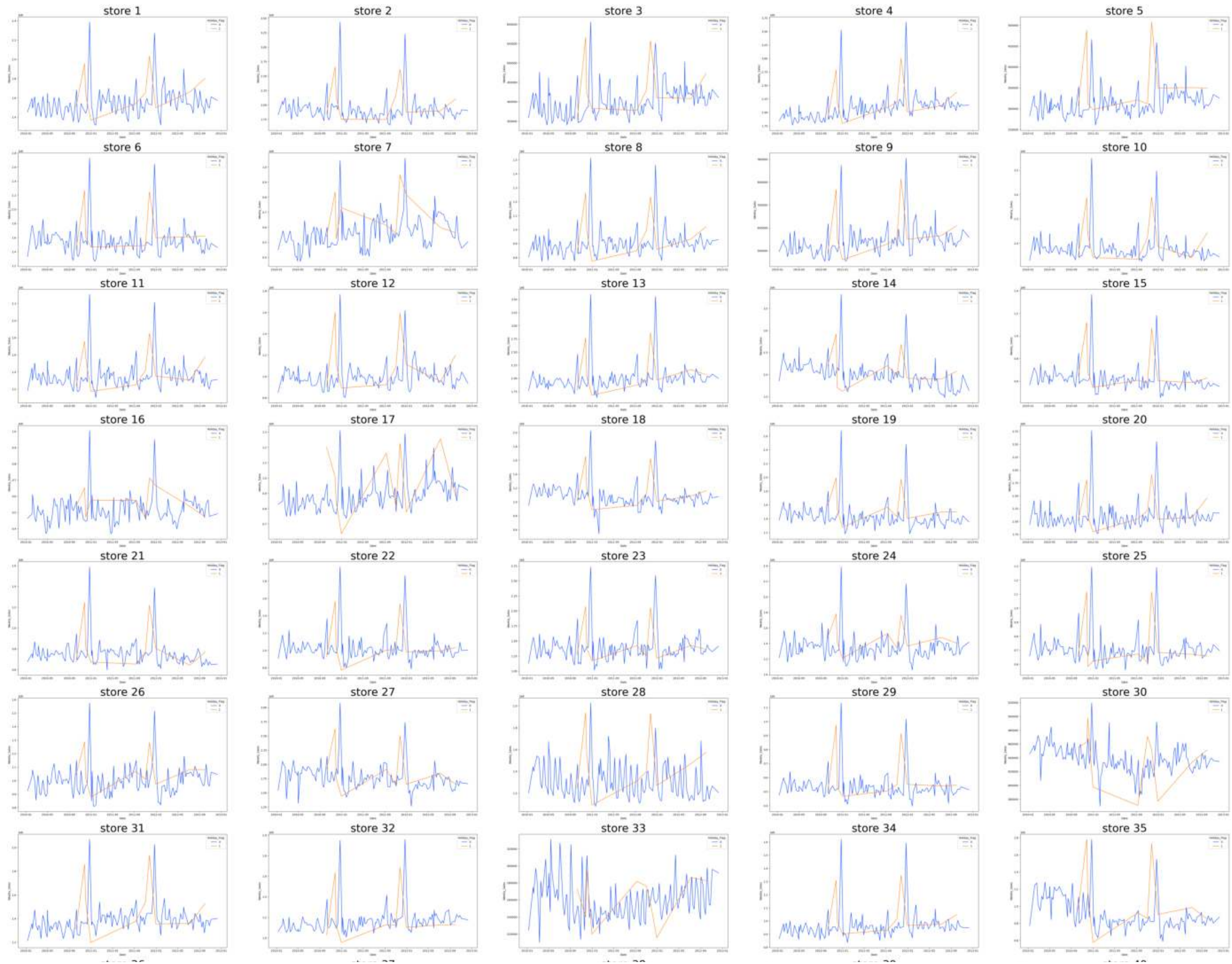
In [53]:

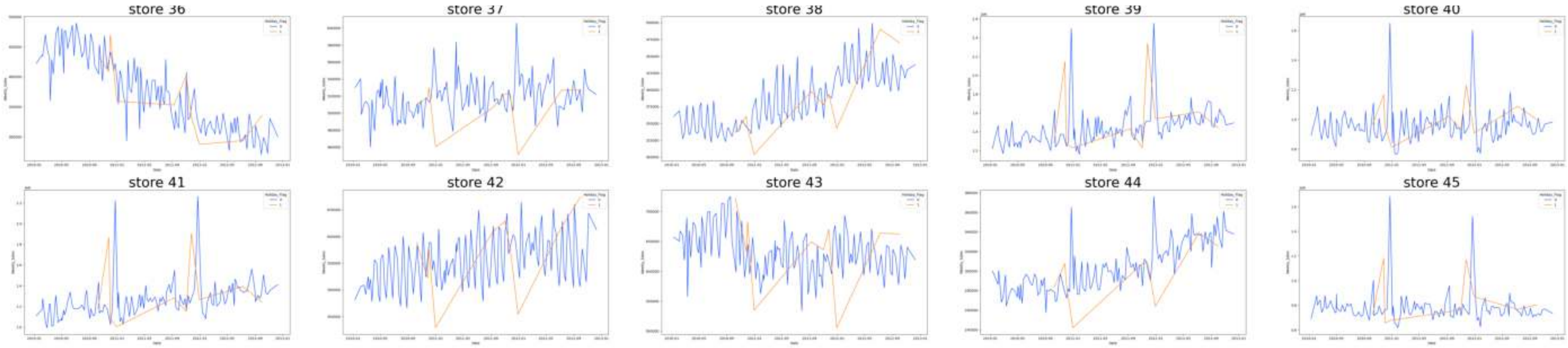
```
# weekly sales of All stores

plt.figure(figsize=(80,80))
for i in range(1,46):
    plt.subplot(9,5,i)
    y= df_new['Weekly_Sales'][df_new['Store'] == i]
    x= df_new['Date'][df_new['Store'] == i]
    plt.title(f'store {i}',fontsize =40)
```

```
    sns.lineplot( x ,y )
plt.show()
```


store 1
store 2
store 3
store 4
store 5
store 6
store 7
store 8
store 9
store 10
store 11
store 12
store 13
store 14
store 15
store 16
store 17
store 18
store 19
store 20
store 21
store 22
store 23
store 24
store 25
store 26
store 27
store 28
store 29
store 30
store 31
store 32
store 33
store 34
store 35

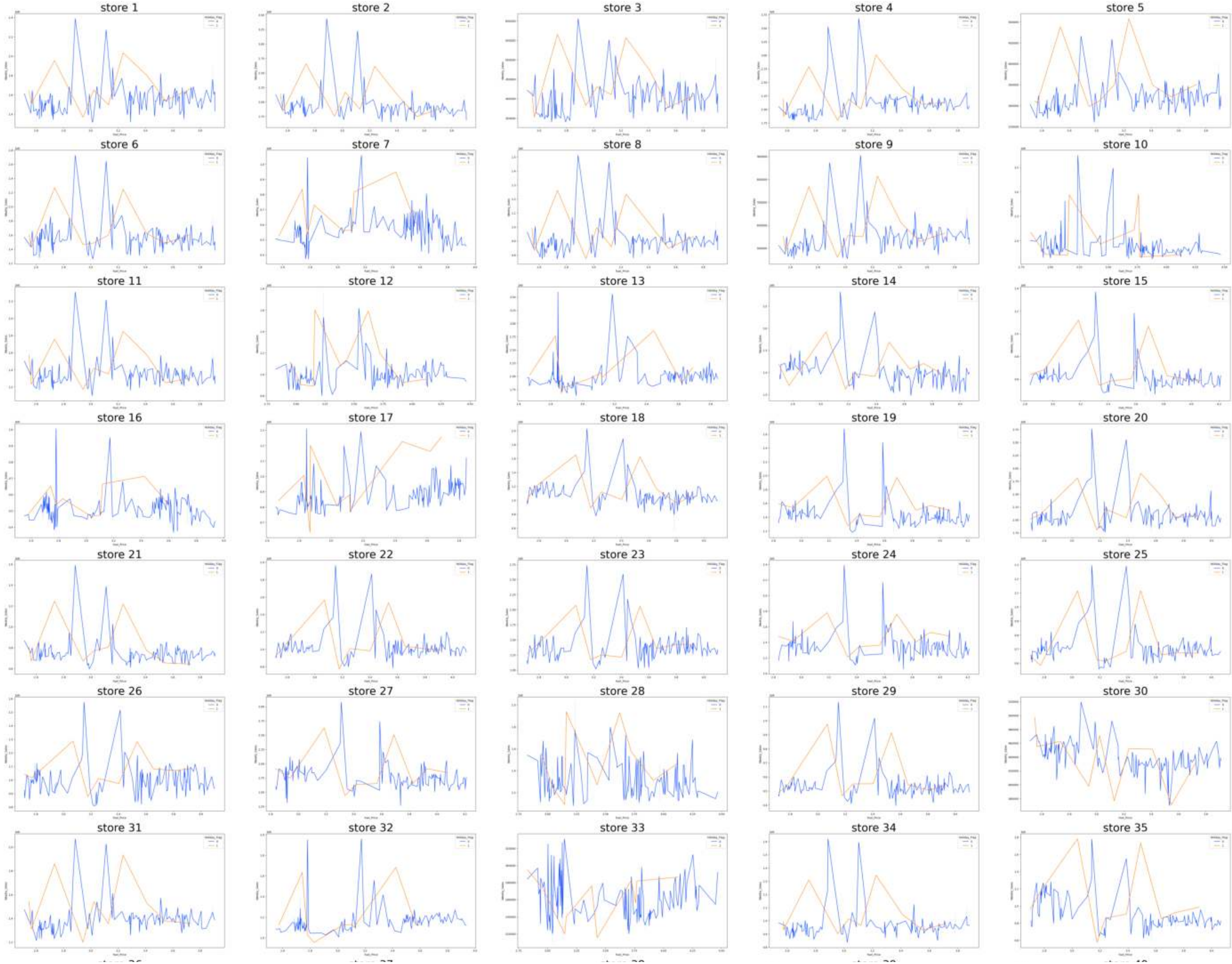
In [54]:
```
# effect of holiday on weekkly sales
plt.figure(figsize= (80,80))
for i in range(1,46):
    plt.subplot(9,5,i)
    x = df_new['Date'][df_new['Store'] == i]
    y = df_new['Weekly_Sales'][df_new['Store'] == i]

    plt.title(f'store {i}',fontsize =40)
    sns.lineplot(x,y,hue = df_new['Holiday_Flag'] )

plt.show()
```

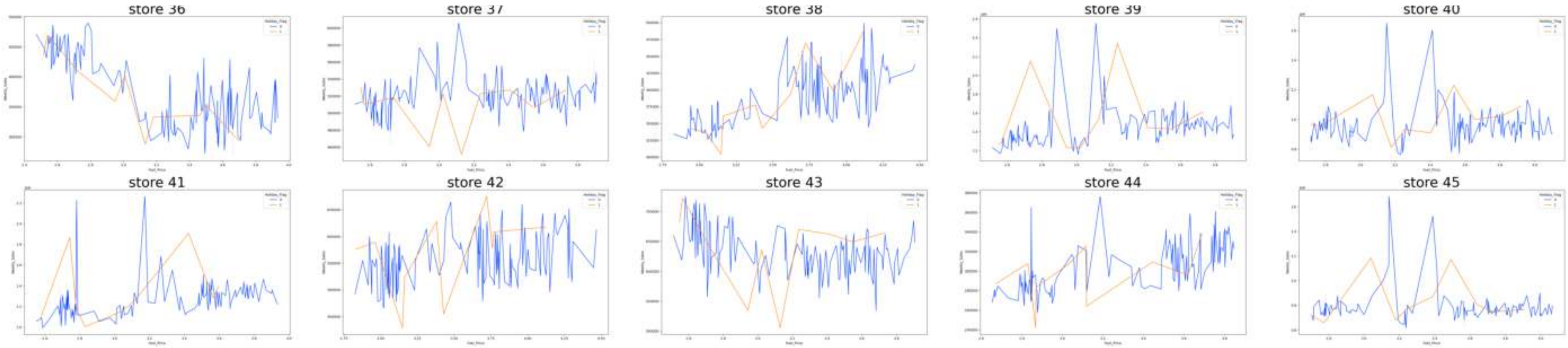

store 1
store 2
store 3
store 4
store 5
store 6
store 7
store 8
store 9
store 10
store 11
store 12
store 13
store 14
store 15
store 16
store 17
store 18
store 19
store 20
store 21
store 22
store 23
store 24
store 25
store 26
store 27
store 28
store 29
store 30
store 31
store 32
store 33
store 34
store 35



In [55]:

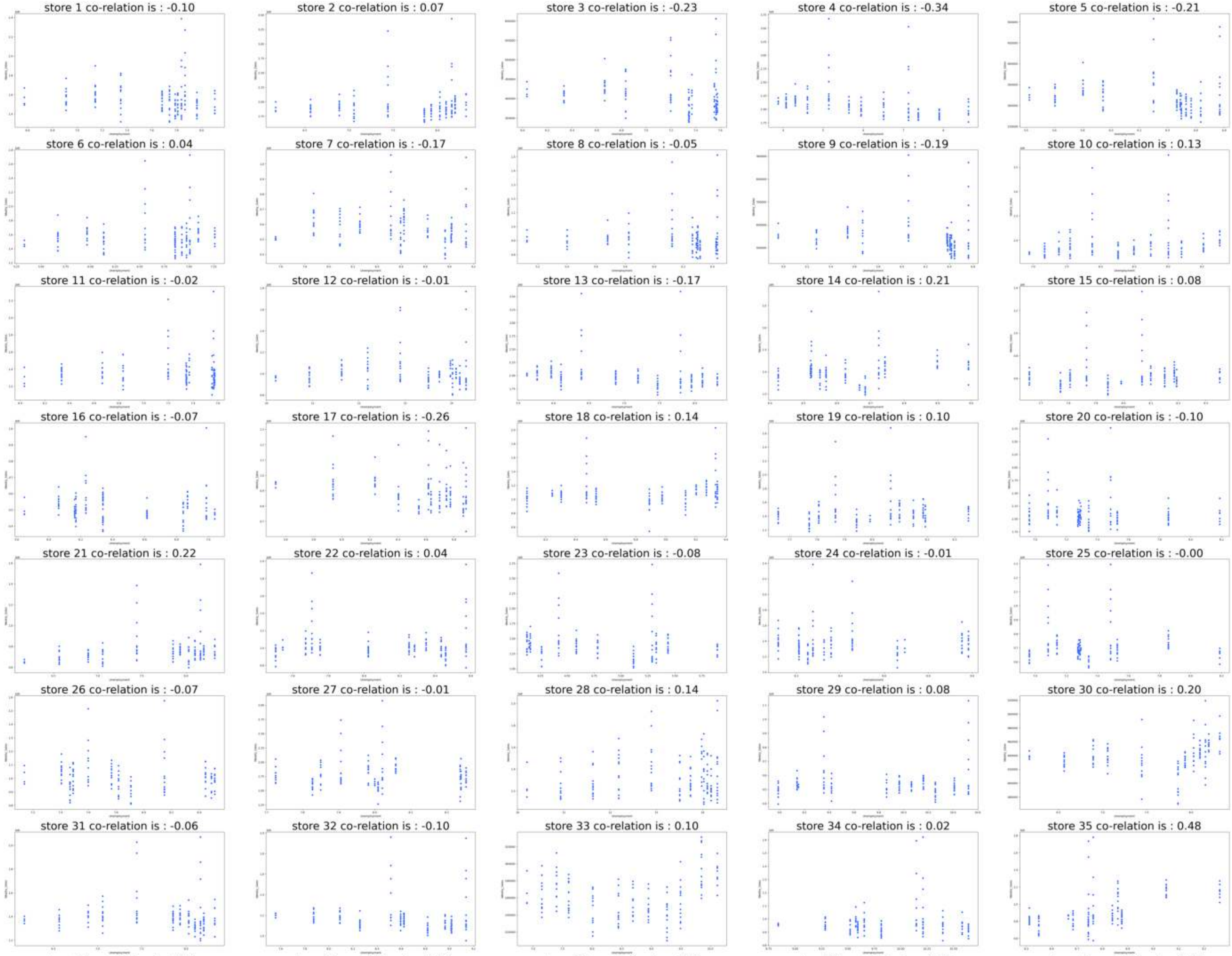
```
# effect of holiday on weekkly sales
plt.figure(figsize= (80,80))
for i in range(1,46):
    plt.subplot(9,5,i)
    x = df_new['Fuel_Price'][df_new['Store'] == i]
    y = df_new['Weekly_Sales'][df_new['Store'] == i]

    plt.title(f'store {i}',fontsize =40)
    sns.lineplot(x,y,hue = df_new['Holiday_Flag'] )

plt.show()
```

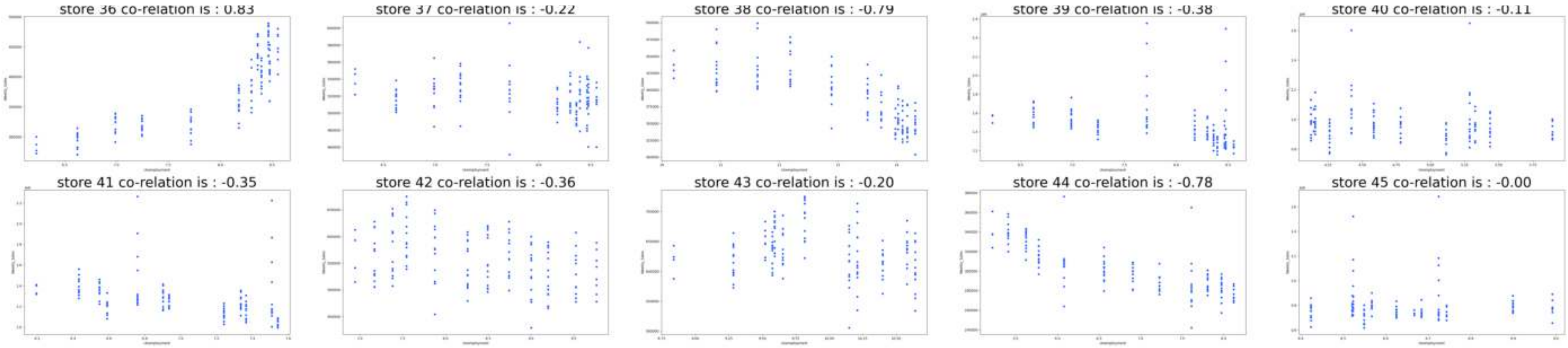

store 1
store 2
store 3
store 4
store 5
store 6
store 7
store 8
store 9
store 10
store 11
store 12
store 13
store 14
store 15
store 16
store 17
store 18
store 19
store 20
store 21
store 22
store 23
store 24
store 25
store 26
store 27
store 28
store 29
store 30
store 31
store 32
store 33
store 34
store 35



In [56]:

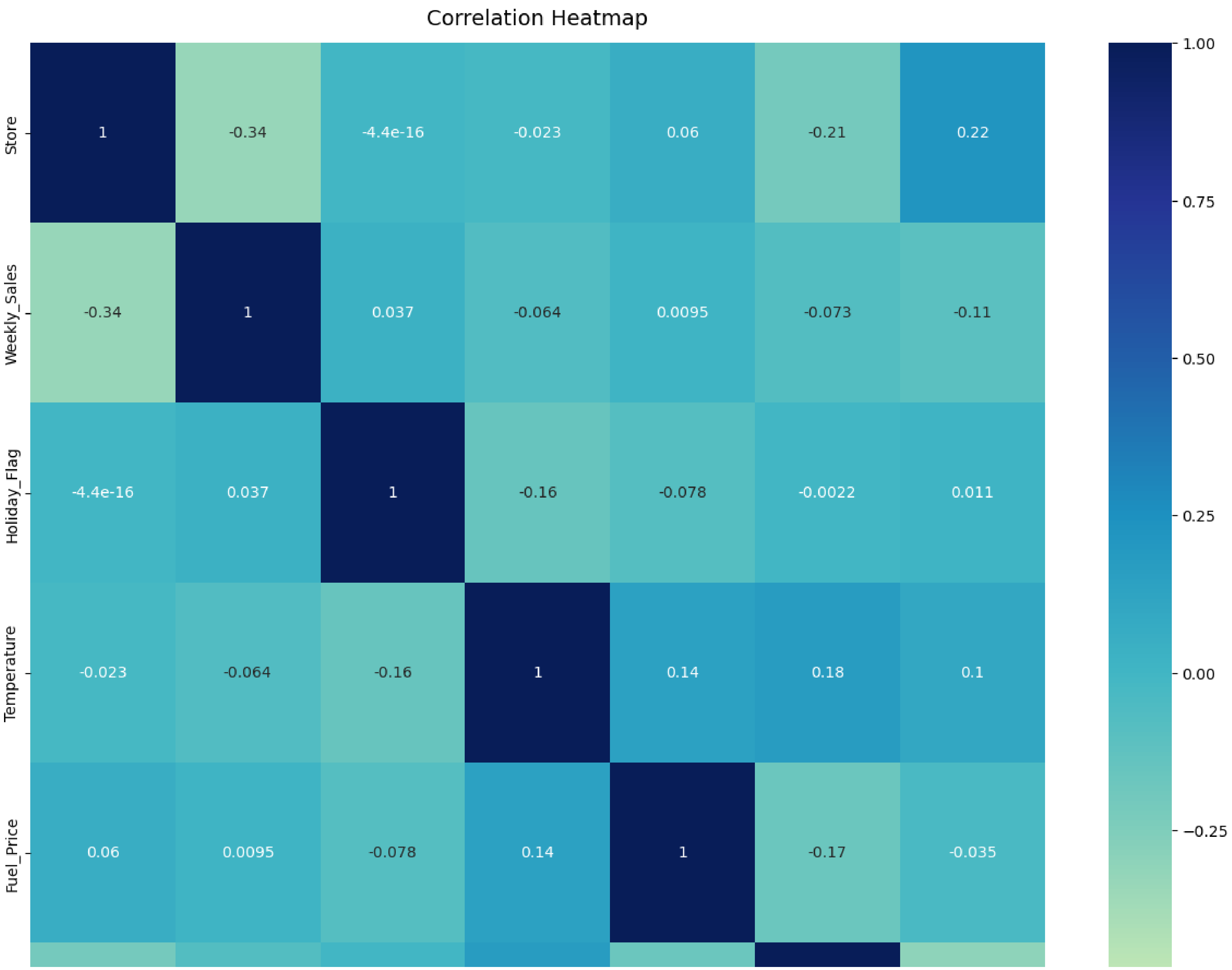
```python
# effect of Unemployment in Weekly Sale
plt.figure(figsize= (80,80))
for i in range(1,46):
    plt.subplot(9,5,i)
    x = df_new['Unemployment'][df_new['Store'] == i]
    y = df_new['Weekly_Sales'][df_new['Store'] == i]

    plt.title(f'store {i} co-relation is : {np.corrcoef(x,y)[0][1]:.2f}',fontsize =40)
    sns.scatterplot(x,y, )

plt.show()
```

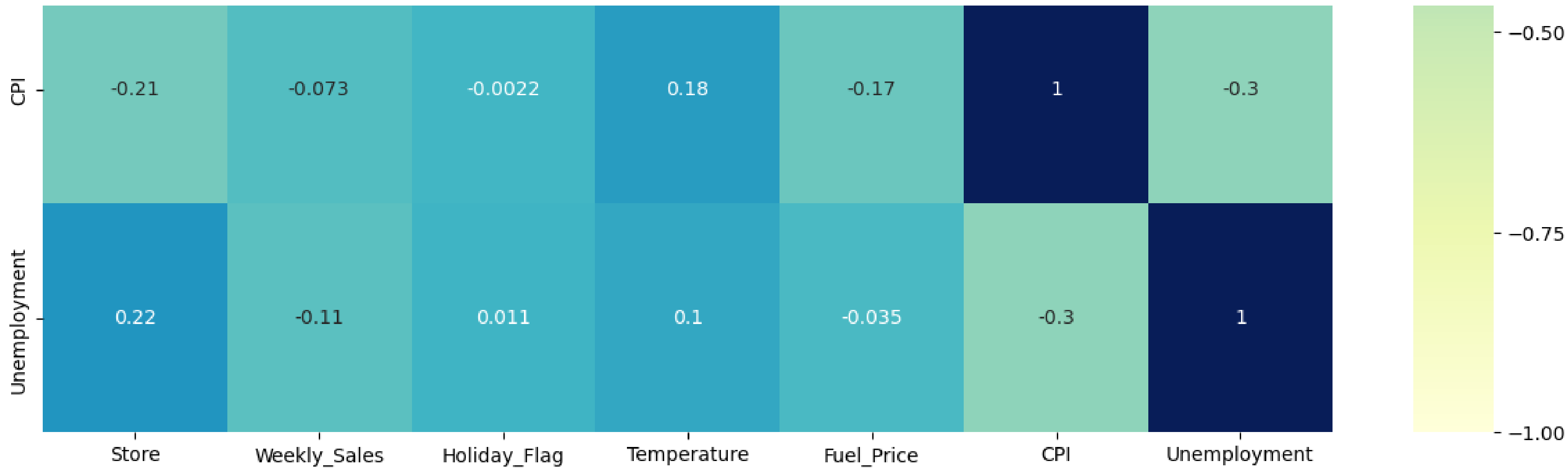

store 1 co-relation is : -0.10
store 2 co-relation is : 0.07
store 3 co-relation is : -0.23
store 4 co-relation is : -0.34
store 5 co-relation is : -0.21
store 6 co-relation is : 0.04
store 7 co-relation is : -0.17
store 8 co-relation is : -0.05
store 9 co-relation is : -0.19
store 10 co-relation is : 0.13
store 11 co-relation is : -0.02
store 12 co-relation is : -0.01
store 13 co-relation is : -0.17
store 14 co-relation is : 0.21
store 15 co-relation is : 0.08
store 16 co-relation is : -0.07
store 17 co-relation is : -0.26
store 18 co-relation is : 0.14
store 19 co-relation is : 0.10
store 20 co-relation is : -0.10
store 21 co-relation is : 0.22
store 22 co-relation is : 0.04
store 23 co-relation is : -0.08
store 24 co-relation is : -0.01
store 25 co-relation is : -0.00
store 26 co-relation is : -0.07
store 27 co-relation is : -0.01
store 28 co-relation is : 0.14
store 29 co-relation is : 0.08
store 30 co-relation is : 0.20
store 31 co-relation is : -0.06
store 32 co-relation is : -0.10
store 33 co-relation is : 0.10
store 34 co-relation is : 0.02
store 35 co-relation is : 0.48



In [57]:

```
fig, ax = plt.subplots(figsize=(15,15))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap ="YlGnBu")
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
```


Correlation Heatmap
Store
Weekly_Sales
Holiday_Flag
Temperature
Fuel_Price
1
-0.34
-4.4e-16
-0.023
0.06
-0.21
0.22
-0.34
1
0.037
-0.064
0.0095
-0.073
-0.11
-4.4e-16
0.037
1
-0.16
-0.078
-0.0022
0.011
-0.023
-0.064
-0.16
1
0.14
0.18
0.1
0.06
0.0095
-0.078
0.14
1
-0.17
-0.035
1.00
0.75
0.50
0.25
0.00
−0.25

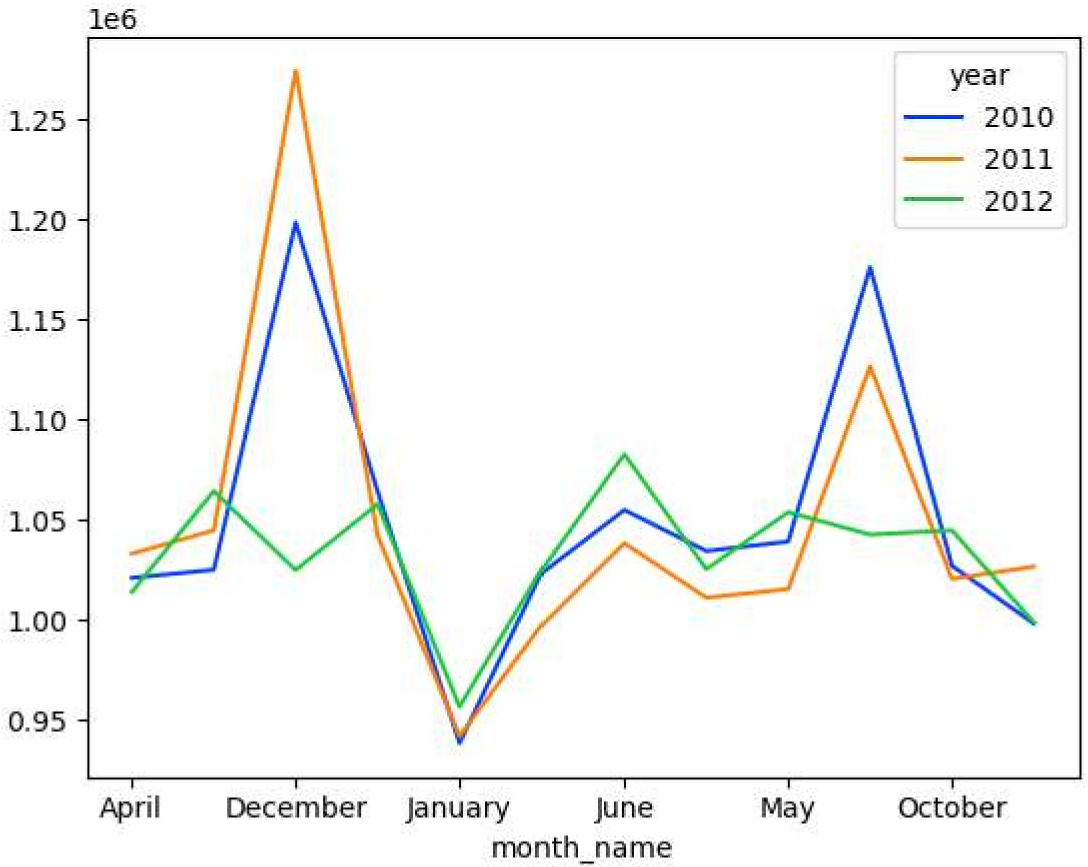


Comment

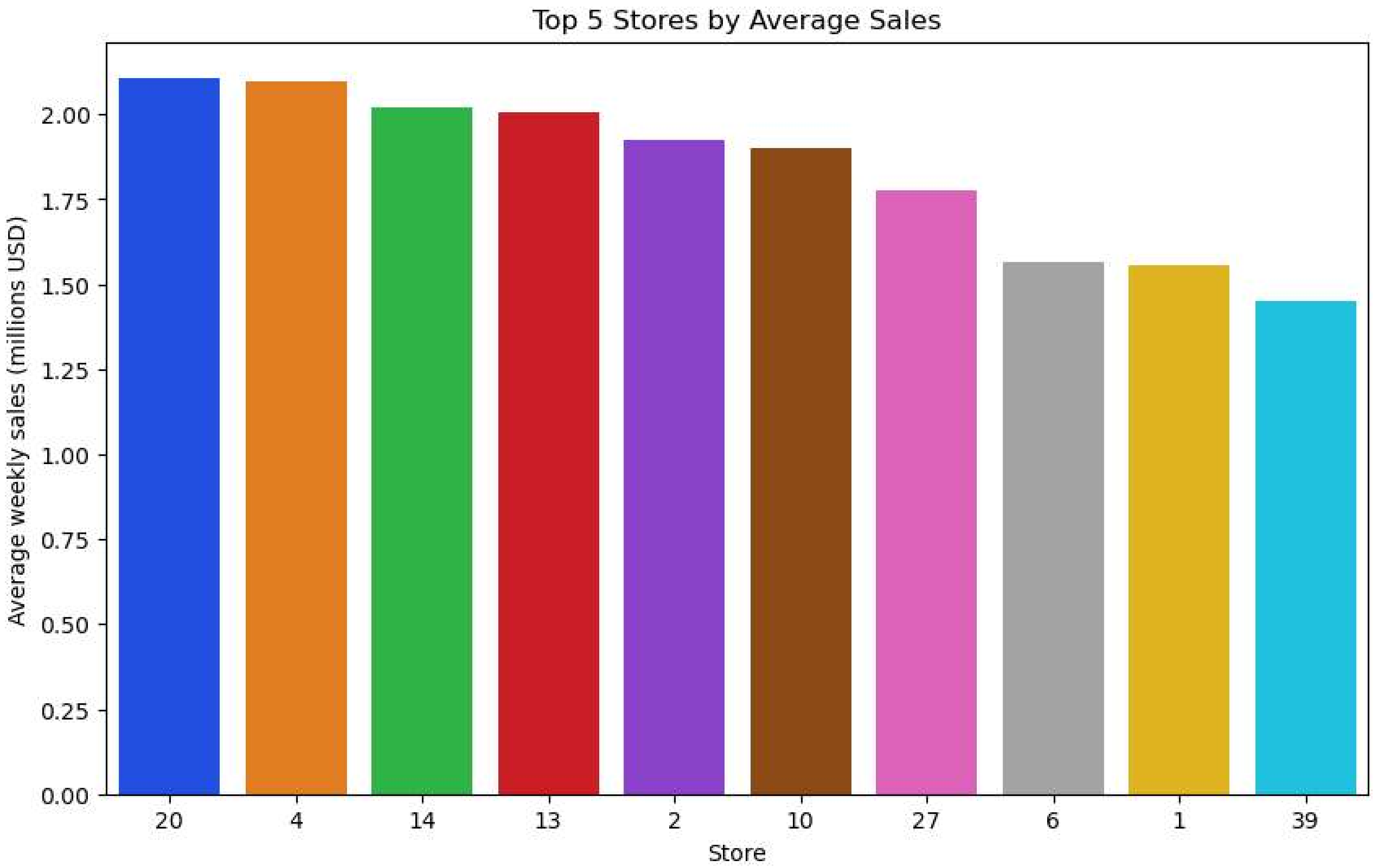
> - **Of all the weaker correlations, employment is the strongest with 0.11 correlation coefficient.**

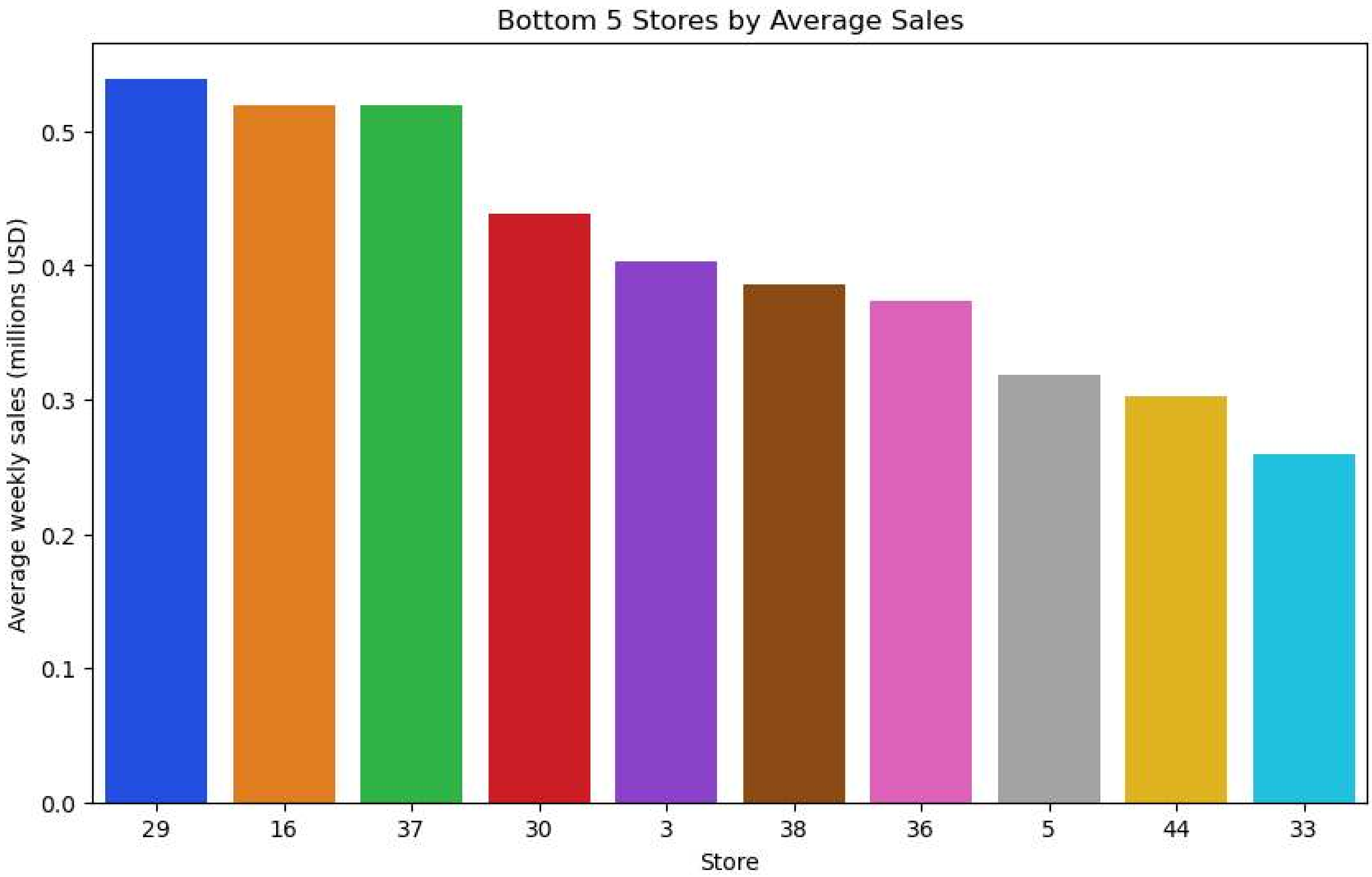
In [58]:
```
month_wise_sales = pd.pivot_table(df_new, values= "Weekly_Sales", columns= "year", index = "month_name",)
month_wise_sales.plot()
```

Out[58]:
```
<AxesSubplot:xlabel='month_name'>
```



In [59]:
```
plot_top_and_bottom_stores(df_new, 'Store')
```


Top 5 Stores by Average Sales
Average weekly sales (millions USD)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
20
4
14
13
2
10
27
6
1
39
Store



In [60]:
```
Stores_Maximum_sales = (20,4,14,13,2,10,27,6,1,39)

for i in Stores_Maximum_sales:
    a1 = df_new.loc[df_new['Store'] == i, 'Weekly_Sales'].sum()
    c1 = round(a1)
    print(f'Store No.{i} has Weekly sales = {c1}\n')
```

```
Store No.20 has Weekly sales = 301397792

Store No.4 has Weekly sales = 299543953

Store No.14 has Weekly sales = 288999911

Store No.13 has Weekly sales = 286517704

Store No.2 has Weekly sales = 275382441

Store No.10 has Weekly sales = 271617714

Store No.27 has Weekly sales = 253855917

Store No.6 has Weekly sales = 223756131

Store No.1 has Weekly sales = 222402809

Store No.39 has Weekly sales = 207445542
```

In [61]:

```
Stores_Minimum_sales = (29,16,37,30,3,38,36,5,44,33)

for i in Stores_Minimum_sales:
    a = df_new.loc[df_new['Store'] == i, 'Weekly_Sales'].sum()
    c = round(a)
    print(f'Store No.{i} has Weekly sales = {c}\n')
```

```
Store No.29 has Weekly sales = 77141554

Store No.16 has Weekly sales = 74252425

Store No.37 has Weekly sales = 74202740

Store No.30 has Weekly sales = 62716885

Store No.3 has Weekly sales = 57586735

Store No.38 has Weekly sales = 55159626

Store No.36 has Weekly sales = 53412215

Store No.5 has Weekly sales = 45475689

Store No.44 has Weekly sales = 43293088

Store No.33 has Weekly sales = 37160222
```

E. Choosing the Algorithm for the Project

> - **The choice of algorithm for this Walmart data set project is Time Series Forecasting.**

- **The Algorithm has been choosen based on several factors such as:**

> **1. The Problem Statemment provided for this project - "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 t come up with useful insights using the data and make prediction models to forecast the sales for X number of months/years.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."**
>
> **2. The Desired Outcome is to forcast the sales based on time, hence we have choosen the time serries algorithm for prediction.**

> **Time Series Forecasting: As the problem statement is to forecast future sales or demand, hence time series forecasting algorithms such as ARIMA, SARIMA is the most suitable algorithm that could be applied on this dataset**

In [62]:

```
low_sales_stores = [29,16,37,30,3,38,36,5,44,33]


top_sales_stores = [20,4,14,13,2,10,27,6,1,39]
```

In [63]:

```
# Selecting the specific Store for prediction
a=int(input('Enter the store id:'))


df_store=df_new[df_new.Store==a]
print("\n\n\nChecking weather the Store selected for prediction has been filtered")
df_store
```

```
Enter the store id:20



Checking weather the Store selected for prediction has been filtered
```

Out[63]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	month	month_name	week	year	Day_
2717	20	2010-05-02	2401395.47	0	25.92	2.784	204.247194	8.187	5	May	17	2010	S
2718	20	2010-12-02	2109107.90	1	22.12	2.773	204.385747	8.187	12	December	48	2010	Th
2719	20	2010-02-19	2161549.76	0	25.43	2.745	204.432100	8.187	2	February	7	2010	
2720	20	2010-02-26	1898193.95	0	32.32	2.754	204.463087	8.187	2	February	8	2010	
2721	20	2010-05-03	2119213.72	0	31.75	2.777	204.494073	8.187	5	May	18	2010	M
...	...	...	...	...	...	...	...	...	...	...	...	...	
2855	20	2012-09-28	2008350.58	0	58.65	3.997	215.736716	7.280	9	September	39	2012	
2856	20	2012-05-10	2246411.89	0	60.77	3.985	215.925886	7.293	5	May	19	2012	Th
2857	20	2012-12-10	2162951.36	0	47.20	4.000	216.115057	7.293	12	December	50	2012	M
2858	20	2012-10-19	1999363.49	0	56.26	3.969	216.146470	7.293	10	October	42	2012	
2859	20	2012-10-26	2031650.55	0	60.04	3.882	216.151590	7.293	10	October	43	2012	

143 rows × 13 columns

F. Motivation and Reasons For Choosing the Algorithm

The Reason and Motivation for choosing time series algorithm for the Walmart dataset was depend on several factors:

> - **The nature of this data is time dependent**
> - **The goal of this analysis that is to find useful insights that can be used by each of the stores to improve in various areas and match the demand with respect to supply.**

Here are some common motivations and reasons for choosing a time series algorithm:

- **Time Dependency:** The data has a clear temporal dependence, hence a time series algorithm would be appropriate this data includes sales data that is recorded over time.

- **Forecasting:** One of the primary uses of time series algorithms is to forecast future values.

- **Trend Analysis:** Time series algorithms also is used to identify trends in the data, such as upward or downward trends in sales. This information can be used to make informed business decisions.

- **Seasonality:** Time series algorithms is also used to identify seasonality in the data, such as an increase in sales during the holiday season. This information can also be used to make informed business decisions and match the demand with the supply.

- **Anomaly Detection:** Time series algorithms is also used to detect unusual or unexpected events in the data, such as a sudden drop in sales. This information can be used to identify potential problems and take corrective action.

In [64]:
```
df_store = df_store[['Date','Weekly_Sales']]
```

In [65]:
```
df_store
```

Out[65]:

	Date	Weekly_Sales
2717	2010-05-02	2401395.47
2718	2010-12-02	2109107.90
2719	2010-02-19	2161549.76
2720	2010-02-26	1898193.95
2721	2010-05-03	2119213.72
...	...	...
2855	2012-09-28	2008350.58
2856	2012-05-10	2246411.89
2857	2012-12-10	2162951.36
2858	2012-10-19	1999363.49
2859	2012-10-26	2031650.55

143 rows × 2 columns

In [66]:

```
# Indexing the date column
df_store = df_store.set_index(['Date'])
```

In [67]:

```
df_store.sort_index(ascending=True, inplace=True)
```

In [68]:

```
df_store
```

Out[68]:

Date	Weekly_Sales
2010-01-10	1933719.21
2010-02-04	2405395.22
2010-02-07	2143676.77
2010-02-19	2161549.76
2010-02-26	1898193.95
...	...
2012-10-08	2144245.39
2012-10-19	1999363.49
2012-10-26	2031650.55
2012-11-05	2168097.11
2012-12-10	2162951.36

143 rows × 1 columns

In [69]:

```
#Spliting the data into train and test data

""" Note: In time series data one cannot divide the data randomly as the main factor in time series data is time."""

df_train = df_store[:110]
df_test = df_store[110:]
```

In [70]:

```
df_train
```

Out[70]:

	Weekly_Sales
Date	
2010-01-10	1933719.21
2010-02-04	2405395.22
2010-02-07	2143676.77
2010-02-19	2161549.76
2010-02-26	1898193.95
...	...
2012-02-17	2309025.16
2012-02-24	2045837.55
2012-03-02	2203523.20
2012-03-08	2094515.71
2012-03-16	2064991.71

110 rows × 1 columns

In [71]:

```
df_test
```

Out[71]:

	Weekly_Sales
Date	
2012-03-23	1992436.96
2012-03-30	2074721.74
2012-04-05	2163510.89
2012-04-13	2045396.06
2012-04-20	1884427.84
2012-04-27	1886503.93
2012-05-10	2246411.89
2012-05-18	2039222.26
2012-05-25	2114989.00
2012-06-01	1964701.94
2012-06-04	2565259.92
2012-06-07	2358055.30
2012-06-15	2165160.29
2012-06-22	2060588.69
2012-06-29	2055952.61
2012-07-09	2080529.06
2012-07-13	2134680.12
2012-07-20	1970170.29
2012-07-27	1911559.10
2012-08-06	2231962.13
2012-08-17	2045061.22
2012-08-24	2005341.43
2012-08-31	2062481.56
2012-09-03	2139265.40

	Weekly_Sales
Date	
2012-09-14	2047949.98
2012-09-21	2028587.24
2012-09-28	2008350.58
2012-10-02	2462978.28
2012-10-08	2144245.39
2012-10-19	1999363.49
2012-10-26	2031650.55
2012-11-05	2168097.11
2012-12-10	2162951.36

G. Assumptions

When applying a time series model, There are certain assumptions are made about the nature of the data and the relationship between the variables. Here are some common assumptions made in time series analysis:

> - **Stationarity:** It is often assumed that the time series is stationary, meaning that the statistical properties of the series do not change over time. This includes the mean, variance, and autocorrelation structure.
> - **Trend and Seasonality:** Many time series models assume the presence of a trend and/or seasonality in the data. For example, the presence of a trend might indicate that the data is increasing or decreasing over time, while the presence of seasonality might indicate that the data follows a recurring pattern.
> - **Linearity:** Some time series models, such as linear regression, assume that the relationship between the variables is linear. This means that changes in the independent variable are directly proportional to changes in the dependent variable.
> - **Homoscedasticity:** Some time series models assume that the errors or residuals in the model are homoscedastic, meaning that the variance of the errors is constant over time.

> - **Independence of Errors:** Some time series models assume that the errors or residuals are independent, meaning that the errors are not related to each other in any way.
>
> - **Normality:** Some time series models assume that the errors are normally distributed, meaning that the distribution of the errors follows a bell-shaped curve.

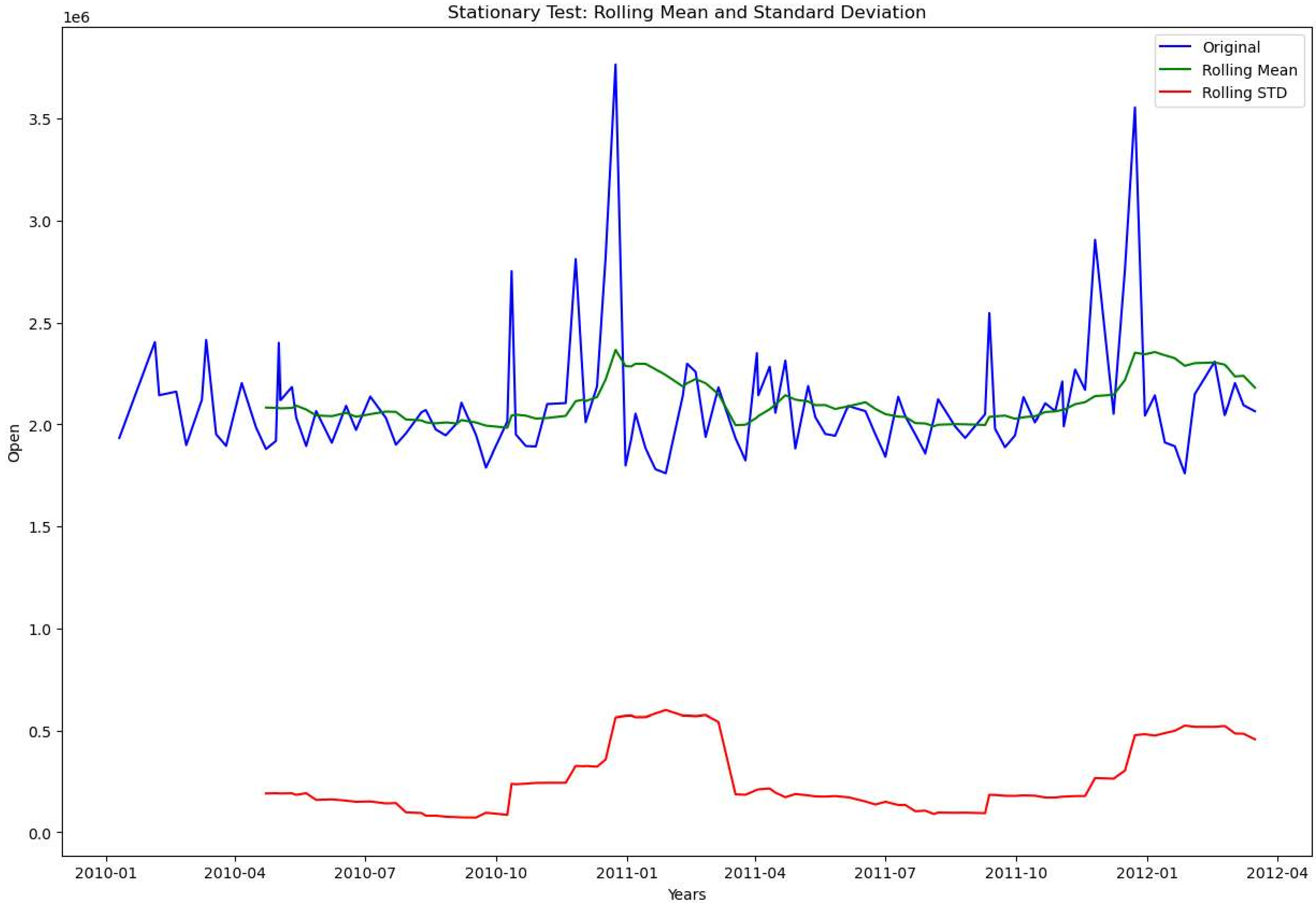
It's important to keep these assumptions in mind when applying a time series model, as violating these assumptions can lead to biased or incorrect results. Before applying a time series model, it's often a good idea to check the data to see if these assumptions are satisfied.

Checking for confirmation about the data is Stationary or not!

In [72]:
```
#Creating a fuunction to ttest weather the data is stationanary or not?

def stationarity_test(timeseries):
    #Get tolling for the window = 12 i.e yearly statistics
    rolling_mean = timeseries.rolling(window = 12).mean()
    rolling_std = timeseries.rolling(window = 12).std()

    # Plotting rolling Statistics
    plt.figure(figsize= (15,10))
    plt.title("Stationary Test: Rolling Mean and Standard Deviation")
    plt.xlabel("Years")
    plt.ylabel("Open")
    plt.plot(timeseries, color = "blue", label = "Original")
    plt.plot(rolling_mean, color = "green", label = "Rolling Mean")
    plt.plot(rolling_std, color = "red", label = "Rolling STD")
    plt.legend()
    plt.show()

    #Dickey-Fuller test
    print("Results of Dickey-fuller Test")
    df_test = adfuller(timeseries)
    df_output = pd.Series(df_test[0:4],index=["Test Statistic","p-value","#lags Used","Number of observations Used"])
    for key, value in df_test[4].items():
        df_output['Critical value (%s)'%key] = value
    print(df_output)
```

In [73]:
```
# Testing the Stationary Score of the data
```

```
stationarity_test(df_train)
```

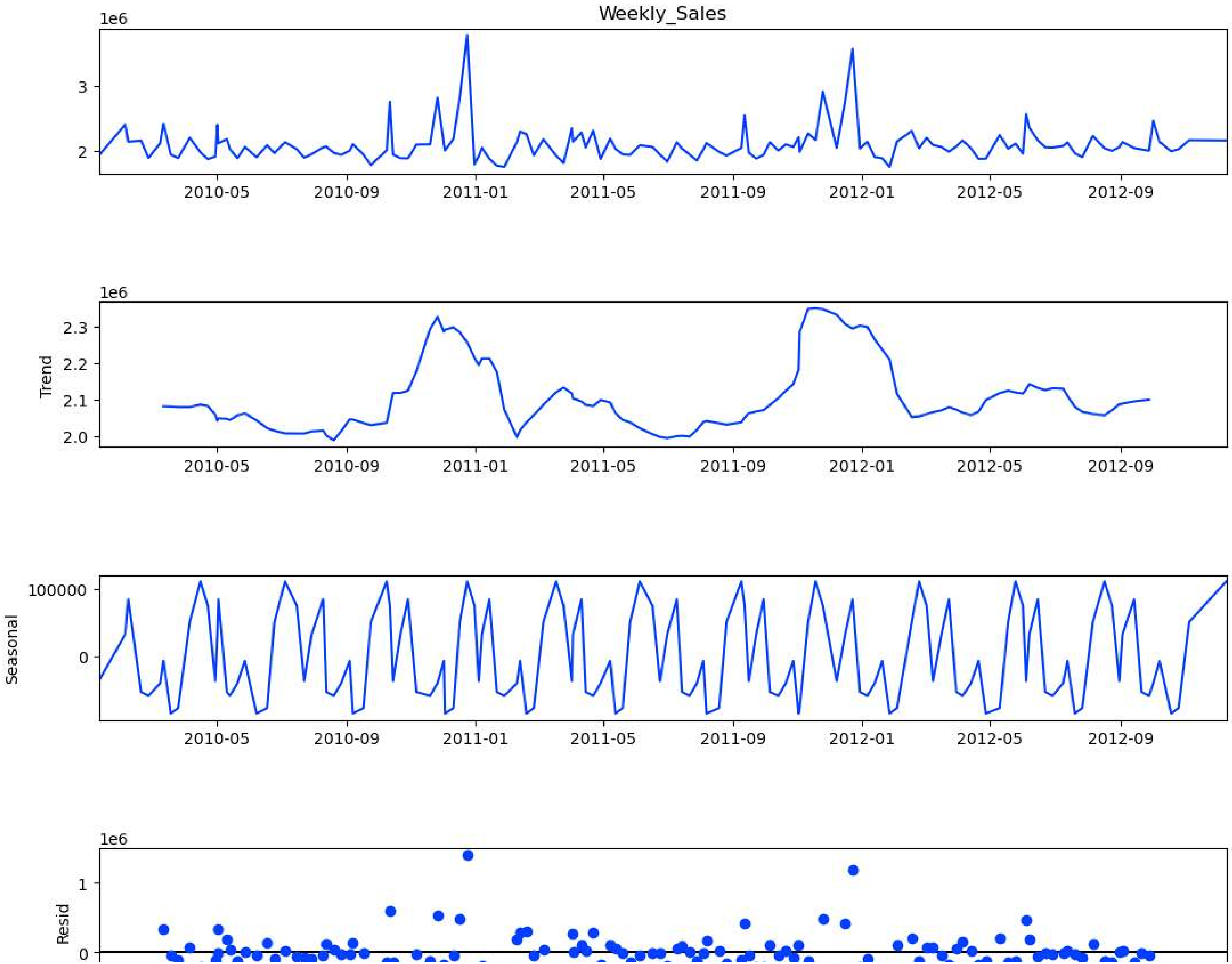


```
Results of Dickey-fuller Test
Test Statistic                 -8.529266e+00
p-value                         1.048433e-13
#lags Used                      0.000000e+00
Number of observations Used     1.090000e+02
Critical value (1%)            -3.491818e+00
Critical value (5%)            -2.888444e+00
Critical value (10%)           -2.581120e+00
dtype: float64
```

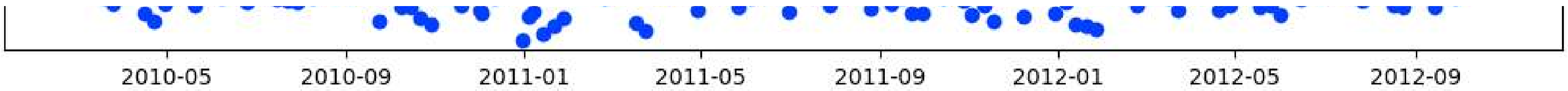
In [74]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

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

```
<Figure size 640x480 with 0 Axes>
```



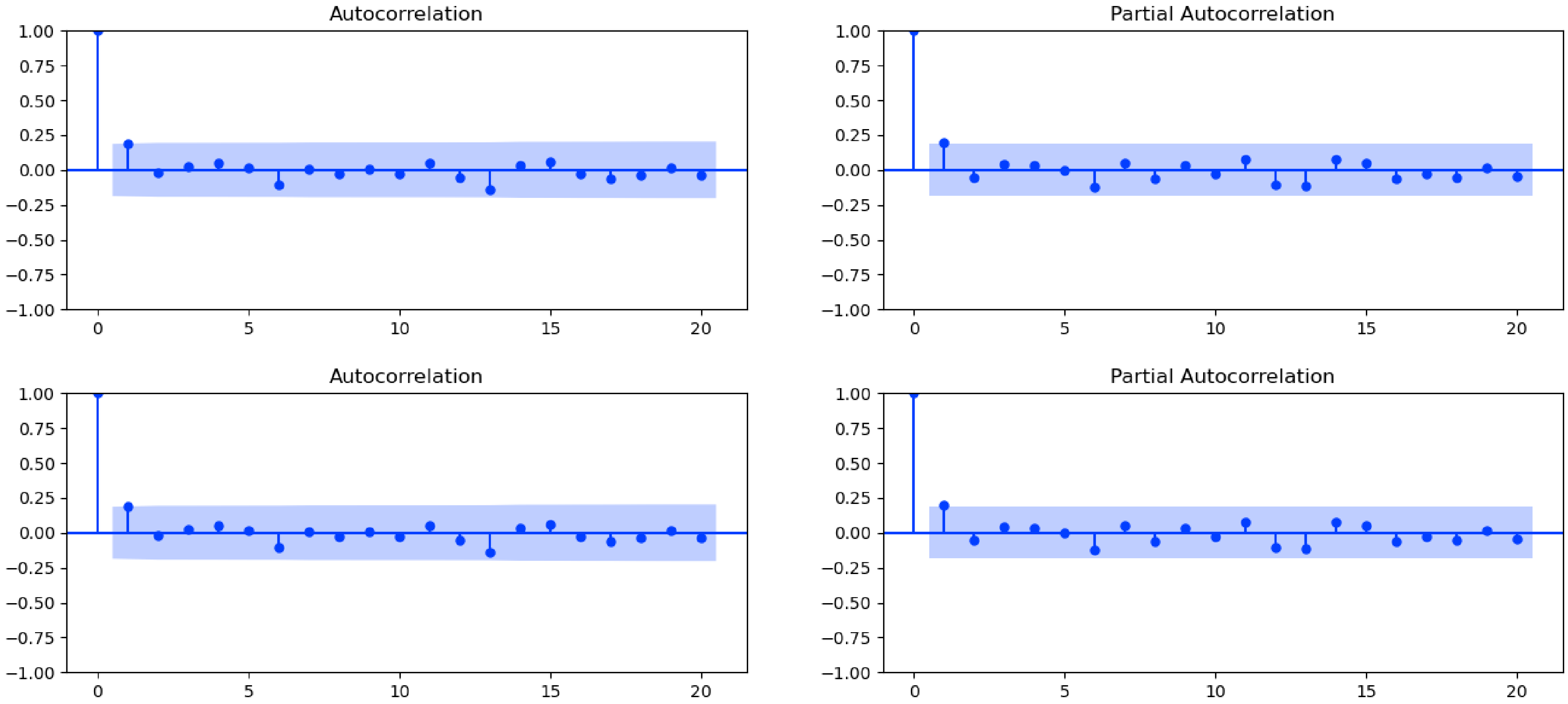
Weekly_Sales
1e6
3
2
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
1e6
Trend
2.3
2.2
2.1
2.0
Seasonal
100000
0
1e6
Resid
1
0



In [ ]:

In [75]:
```
fig,axes = plt.subplots(1,2,figsize = (16,3), dpi=100)
plot_acf(df_train["Weekly_Sales"].tolist(), lags=20, ax=axes[0])
plot_pacf(df_train["Weekly_Sales"].tolist(), lags = 20, ax = axes[1])
```

Out[75]:



In [76]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [77]:
```
Model = ARIMA(df_train, order = (3,0,4)) #[Order = "P"(3), "D"(0), "Q"(4)]
Results_ARIMA = Model.fit()
print(Results_ARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:           Weekly_Sales   No. Observations:                  110
Model:                 ARIMA(3, 0, 4)   Log Likelihood               -1540.174
Date:                Mon, 13 Feb 2023   AIC                           3098.347
Time:                        13:42:09   BIC                           3122.652
Sample:                             0   HQIC                          3108.205
                                - 110
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        2.11e+06   6.85e+04     30.799      0.000    1.98e+06    2.24e+06
ar.L1          0.0652      0.203      0.321      0.748      -0.333       0.463
ar.L2         -0.0587      0.189     -0.310      0.756      -0.429       0.312
ar.L3         -0.8334      0.161     -5.180      0.000      -1.149      -0.518
ma.L1          0.1764      0.281      0.627      0.531      -0.375       0.728
ma.L2         -0.0198      0.268     -0.074      0.941      -0.545       0.505
ma.L3          0.9730      0.166      5.862      0.000       0.648       1.298
ma.L4          0.1697      0.146      1.162      0.245      -0.117       0.456
sigma2      9.605e+10      0.000   2.43e+14      0.000     9.6e+10     9.6e+10
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               295.14
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               2.56   Skew:                             2.29
Prob(H) (two-sided):                  0.01   Kurtosis:                         9.58
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.28e+32. Standard errors may be unstable.
```

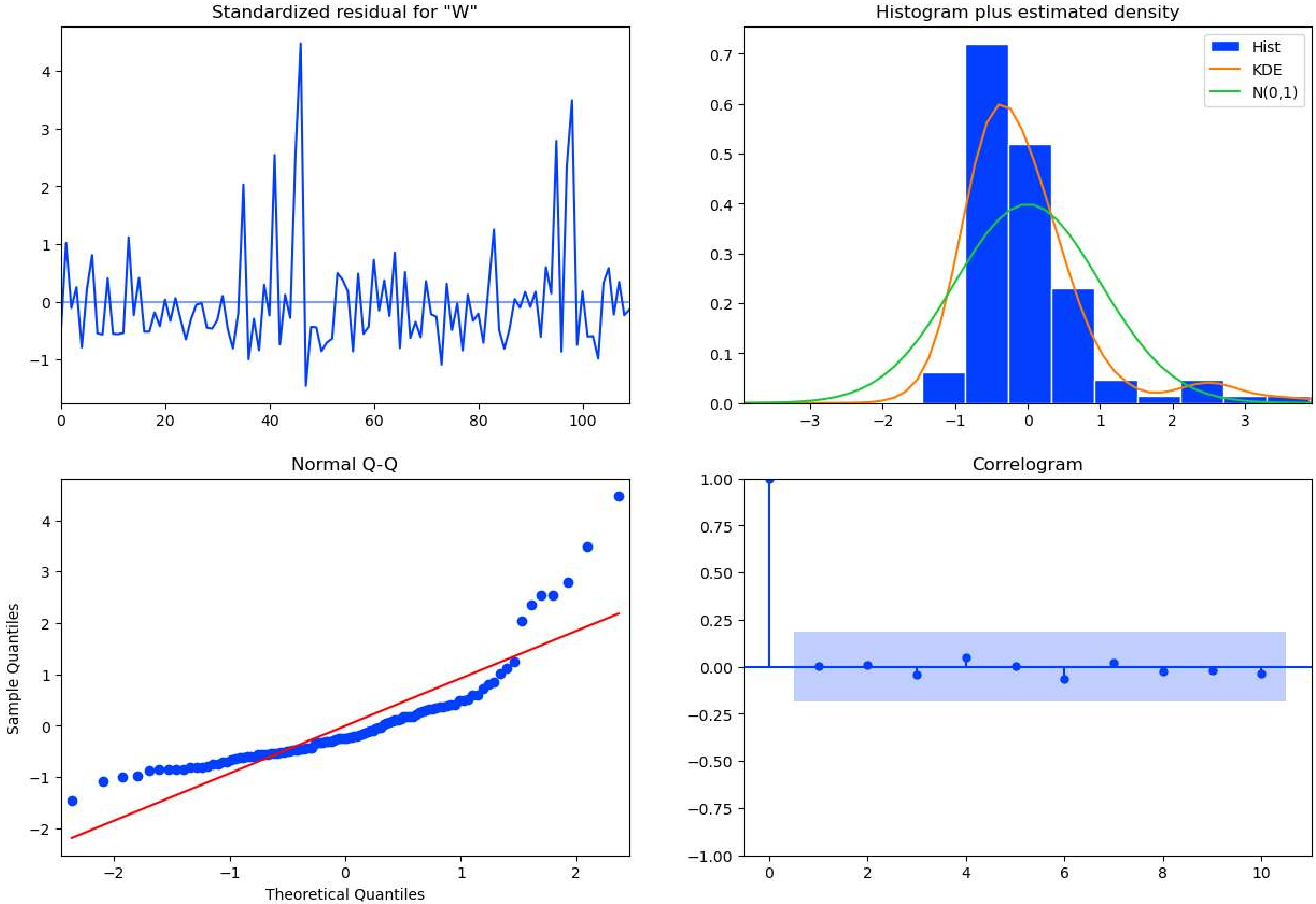
H. Model Evaluation and Techniques

There are several techniques for evaluating the performance of a time series model. Some common evaluation metrics include:

> - **Mean Absolute Error (MAE):** This metric measures the average absolute difference between the predicted values and the actual values. A smaller MAE indicates a better fit of the model to the data.

- **Mean Squared Error (MSE):** This metric measures the average squared difference between the predicted values and the actual values. A smaller MSE indicates a better fit of the model to the data.
- **Root Mean Squared Error (RMSE):** This metric is the square root of the MSE and is often used as a more interpretable measure of the fit of the model to the data.
- **Mean Absolute Percentage Error (MAPE):** This metric measures the average absolute percentage difference between the predicted values and the actual values. It can be useful when comparing models, as it provides a measure of the fit of the model in percentage terms.
- **Correlation Coefficient:** This metric measures the correlation between the predicted values and the actual values. A value close to 1 indicates a strong positive correlation, while a value close to -1 indicates a strong negative correlation.
- **Residual Plots:** Residual plots show the difference between the predicted values and the actual values. Ideally, the residuals should be randomly distributed and have a constant variance over time.

In [78]:
```
Results_ARIMA.plot_diagnostics(figsize=(15,10))
plt.show()
```



In [79]:
```
predict = Results_ARIMA.predict()
```

In [80]:
```
predict
```

Out[80]:
```
Date
2010-01-10    2.110411e+06
2010-02-04    2.077178e+06
2010-02-07    2.179016e+06
2010-02-19    2.080825e+06
2010-02-26    2.150386e+06
                  ...
2012-02-17    2.126635e+06
2012-02-24    2.115603e+06
2012-03-02    2.096253e+06
2012-03-08    2.167290e+06
2012-03-16    2.107235e+06
Name: predicted_mean, Length: 110, dtype: float64
```

In [81]:
```
predict = predict.to_frame()
```

In [82]:
```
predict.head()
```

Out[82]:

	predicted_mean
Date	
2010-01-10	2.110411e+06
2010-02-04	2.077178e+06
2010-02-07	2.179016e+06
2010-02-19	2.080825e+06
2010-02-26	2.150386e+06

In [83]:
```
def mean_absolute_percentage_error(y_true, y_predict):
    return np.mean(np.abs((y_true - y_predict)/y_true))*100
```

In [84]:
```
#Calculating the MAPE (Mean_Absolute_Percentage_Error)

print("Train MAPE :", mean_absolute_percentage_error(df_train.Weekly_Sales,predict.predicted_mean))
```

```
Train MAPE : 8.626854813352265
```

In [85]:
```
forcast_test = Results_ARIMA.forecast(12)
```

In [86]:
```
forcast_test = forcast_test.to_frame()
```

In [87]:
```
forcast_test.head()
```

Out[87]:

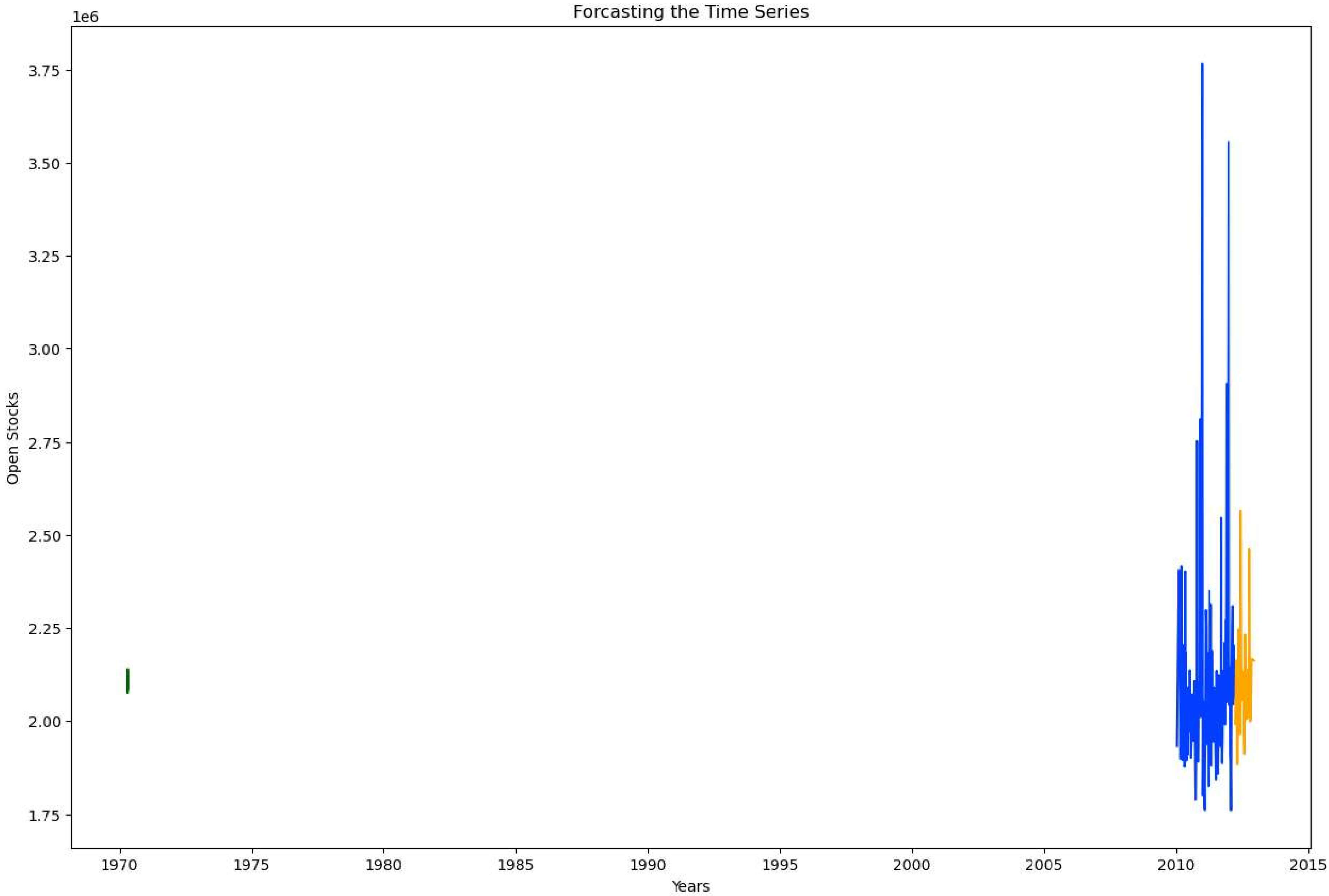
	predicted_mean
110	2.116499e+06
111	2.075059e+06
112	2.092561e+06
113	2.099142e+06
114	2.140187e+06

In [88]:
```
#Calculating the MAPE (Mean_Absolute_Percentage_Error)

print("Test MAPE :", mean_absolute_percentage_error(df_test.Weekly_Sales,forcast_test.predicted_mean))
```

```
Test MAPE : nan
```

In [89]:
```
plt.figure(figsize=(15,10))

plt.plot(df_train.Weekly_Sales)
plt.plot(df_test.Weekly_Sales, color = "Orange")
plt.plot(forcast_test.predicted_mean, color = "darkgreen")

plt.xlabel("Years")
plt.ylabel("Open Stocks")
plt.title("Forcasting the Time Series")
```

Out[89]:
```
Text(0.5, 1.0, 'Forcasting the Time Series')
```



In [90]:
```
# Define the p,d,q to take any value between 0 to 5
p = d = q = range(0,5)

import itertools
```

```python
#Generate all different combinations of p,d,q triplets
pdq = list(itertools.product(p,d,q))


#Generate all different combinations of  seasonal p,d,q triplets
seasonal_pdq = [(x[0], x[1], x[2],52) for x in list(itertools.product(p,d,q))]
```

In [91]:
```python
import statsmodels.api as sm


mod = sm.tsa.statespace.SARIMAX(df_train,
                                order=(4,4,3),
                                seasonal_order = (1,1,0,52), #Enfore_Sationary = False
                                enforce_invertibility = False)

results = mod.fit()

print(results.summary().tables[1])
```

```
This problem is unconstrained.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            9     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.32113D+00    |proj g|=  1.49397D-01

At iterate    5    f=  7.15289D+00    |proj g|=  1.95522D-01

At iterate   10    f=  7.08587D+00    |proj g|=  2.63954D+00

At iterate   15    f=  7.07157D+00    |proj g|=  2.60877D-01

At iterate   20    f=  7.06459D+00    |proj g|=  4.12130D-02

At iterate   25    f=  7.06295D+00    |proj g|=  9.20689D-02

At iterate   30    f=  7.06220D+00    |proj g|=  1.68401D-01

At iterate   35    f=  7.06158D+00    |proj g|=  1.02763D+00

At iterate   40    f=  7.04832D+00    |proj g|=  4.91915D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    9     41     66      1     0     0   0.000D+00  -0.000D+00
  F =  -0.0000000000000000

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -3.6274         -0        inf      0.000      -3.627      -3.627
ar.L2         -5.2537         -0        inf      0.000      -5.254      -5.254
ar.L3         -3.6245         -0        inf      0.000      -3.624      -3.624
ar.L4         -0.9982   9.98e-08     -1e+07      0.000      -0.998      -0.998
ma.L1        -10.3096         -0        inf      0.000     -10.310     -10.310
ma.L2         13.8452   3.12e-08   4.44e+08      0.000      13.845      13.845
ma.L3         -5.0800   8.49e-08  -5.98e+07      0.000      -5.080      -5.080
ar.S.L52       0.9979         -0       -inf      0.000       0.998       0.998
sigma2      2.765e+11   1.56e-18   1.77e+29      0.000    2.76e+11    2.76e+11
==============================================================================
```

In [92]:

```
df_test.index = np.arange(110,143)

sarima_1 = sm.tsa.statespace.SARIMAX(df_train["Weekly_Sales"], seasonal_order = (1,1,0,52),
                                    enforce_stationarity=False, enforce_invertibility=False,).fit()
sarima_1.summary()


pred_train_sea_1 = sarima_1.predict()
pred_train_sea_1
pred_train_sea_1 =pred_train_sea_1.to_frame()
pred_train_sea_1.head()

print("Train MAPE Seasonal:", mean_absolute_percentage_error(df_train['Weekly_Sales'],
                                                            pred_train_sea_1.predicted_mean))

forecast_test_sea_1 = sarima_1.forecast(34)
forecast_test_sea_1


forecast_test_sea_1 =forecast_test_sea_1.to_frame()
forecast_test_sea_1.head()

forecast_extra_sea_1 = sarima_1.forecast(48)


error =  mean_absolute_percentage_error(df_test['Weekly_Sales'],forecast_test_sea_1.predicted_mean)
print("Test MAPE Seasonal:",error)
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.11344D-01    |proj g|=  4.93826D-03
 This problem is unconstrained.
           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3      3      5      1     0     0   8.721D-06   6.105D-01
  F =  0.61048379569928246

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
Train MAPE Seasonal: 100.78291672474053
Test MAPE Seasonal: 8.561695506503591
```

Auto ARIMA on Train Data set

In [93]:
```
model_2 = pm.auto_arima(df_train, start_p=1, start_q=1,
                       test = "adf",   # Using the adf test to find the optimal "D"
                       max_p = 10, max_q =10,  # Maximum p and Q
                       m = 1,                  # Frequency of the Series
                       d = 2,                  # Let the model decide "D"
                       seasonal = False,       # No Seasonality
                       start_P = 0,
                       D = 0,
```

```
                       trace = True,
                       error_acion = "ignor",
                       suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,2,1)(0,0,0)[0] intercept   : AIC=3106.377, Time=0.08 sec
 ARIMA(0,2,0)(0,0,0)[0] intercept   : AIC=3198.393, Time=0.01 sec
 ARIMA(1,2,0)(0,0,0)[0] intercept   : AIC=3164.357, Time=0.02 sec
 ARIMA(0,2,1)(0,0,0)[0] intercept   : AIC=3108.631, Time=0.04 sec
 ARIMA(0,2,0)(0,0,0)[0]             : AIC=3196.417, Time=0.01 sec
 ARIMA(2,2,1)(0,0,0)[0] intercept   : AIC=3101.989, Time=0.05 sec
 ARIMA(2,2,0)(0,0,0)[0] intercept   : AIC=3139.428, Time=0.04 sec
 ARIMA(3,2,1)(0,0,0)[0] intercept   : AIC=3101.647, Time=0.06 sec
 ARIMA(3,2,0)(0,0,0)[0] intercept   : AIC=3127.920, Time=0.03 sec
 ARIMA(4,2,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.13 sec
 ARIMA(3,2,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.14 sec
 ARIMA(2,2,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.19 sec
 ARIMA(4,2,0)(0,0,0)[0] intercept   : AIC=3116.470, Time=0.06 sec
 ARIMA(4,2,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.21 sec
 ARIMA(3,2,1)(0,0,0)[0]             : AIC=3097.957, Time=0.06 sec
 ARIMA(2,2,1)(0,0,0)[0]             : AIC=3098.139, Time=0.05 sec
 ARIMA(3,2,0)(0,0,0)[0]             : AIC=3125.881, Time=0.03 sec
 ARIMA(4,2,1)(0,0,0)[0]             : AIC=3096.102, Time=0.10 sec
 ARIMA(4,2,0)(0,0,0)[0]             : AIC=3114.480, Time=0.04 sec
 ARIMA(5,2,1)(0,0,0)[0]             : AIC=3097.903, Time=0.15 sec
 ARIMA(4,2,2)(0,0,0)[0]             : AIC=inf, Time=0.16 sec
 ARIMA(3,2,2)(0,0,0)[0]             : AIC=inf, Time=0.13 sec
 ARIMA(5,2,0)(0,0,0)[0]             : AIC=3113.406, Time=0.04 sec
 ARIMA(5,2,2)(0,0,0)[0]             : AIC=inf, Time=0.19 sec

Best model:  ARIMA(4,2,1)(0,0,0)[0]
Total fit time: 2.054 seconds
```

In [ ]:

```
```

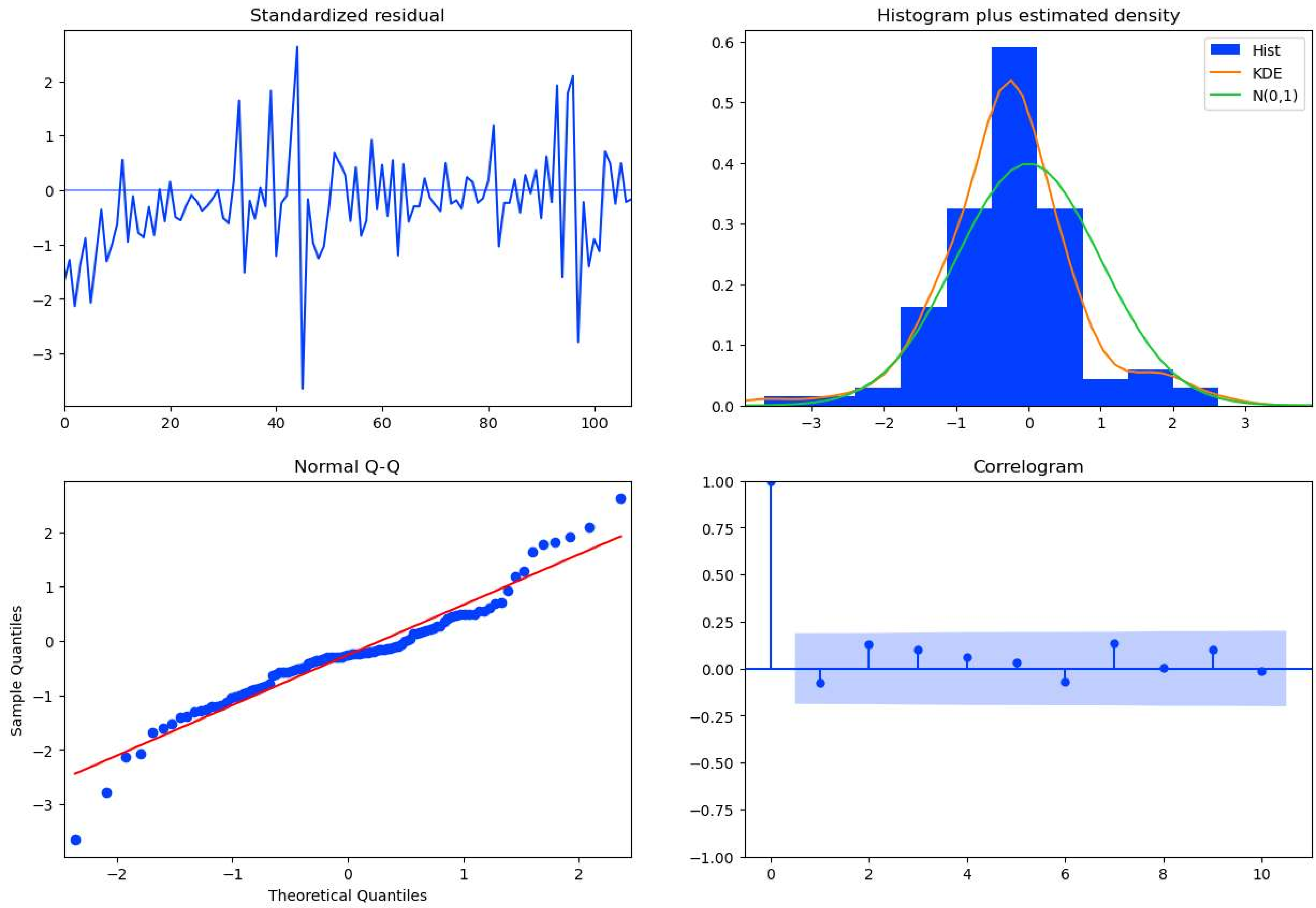
In [94]:

```
print(model_2.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:                      y   No. Observations:                  110
Model:               SARIMAX(4, 2, 1)   Log Likelihood               -1542.051
Date:                Mon, 13 Feb 2023   AIC                           3096.102
Time:                        13:42:52   BIC                           3112.195
Sample:                             0   HQIC                          3102.627
                                - 110                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3572      0.071     -5.043      0.000      -0.496      -0.218
ar.L2         -0.4158      0.090     -4.624      0.000      -0.592      -0.240
ar.L3         -0.2545      0.097     -2.622      0.009      -0.445      -0.064
ar.L4         -0.1432      0.084     -1.695      0.090      -0.309       0.022
ma.L1         -0.9849      0.083    -11.810      0.000      -1.148      -0.821
sigma2      1.548e+11   1.99e-13   7.76e+23      0.000    1.55e+11    1.55e+11
===================================================================================
Ljung-Box (L1) (Q):                   0.65   Jarque-Bera (JB):                21.29
Prob(Q):                              0.42   Prob(JB):                         0.00
Heteroskedasticity (H):               1.01   Skew:                            -0.00
Prob(H) (two-sided):                  0.99   Kurtosis:                         5.18
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 8.35e+38. Standard errors may be unstable.
```

In [95]:

```
model_2.plot_diagnostics(figsize=(15,10))
plt.show()
```



In [96]:
```
test_predict_2 = model_2.predict()
```

In [97]:
```
test_predict_2 = test_predict_2.to_frame()
```

In [98]:
```
test_predict_2.head()
```

Out[98]:

	0
110	2.135147e+06
111	2.144273e+06
112	2.151703e+06
113	2.148368e+06
114	2.150845e+06

In [99]:
```
test_predict_2.rename(columns = {0:"Prediction"}, inplace=True)
```

In [100…
```
test_predict_2.head()
```

Out[100]:

	Prediction
110	2.135147e+06
111	2.144273e+06
112	2.151703e+06
113	2.148368e+06
114	2.150845e+06

In [101…
```
#Calculating the MAPE (Mean_Absolute_Percentage_Error)

print("Test MAPE :", mean_absolute_percentage_error(df_test.Weekly_Sales,test_predict_2.Prediction))
```

```
Test MAPE : 7.037904733732731
```

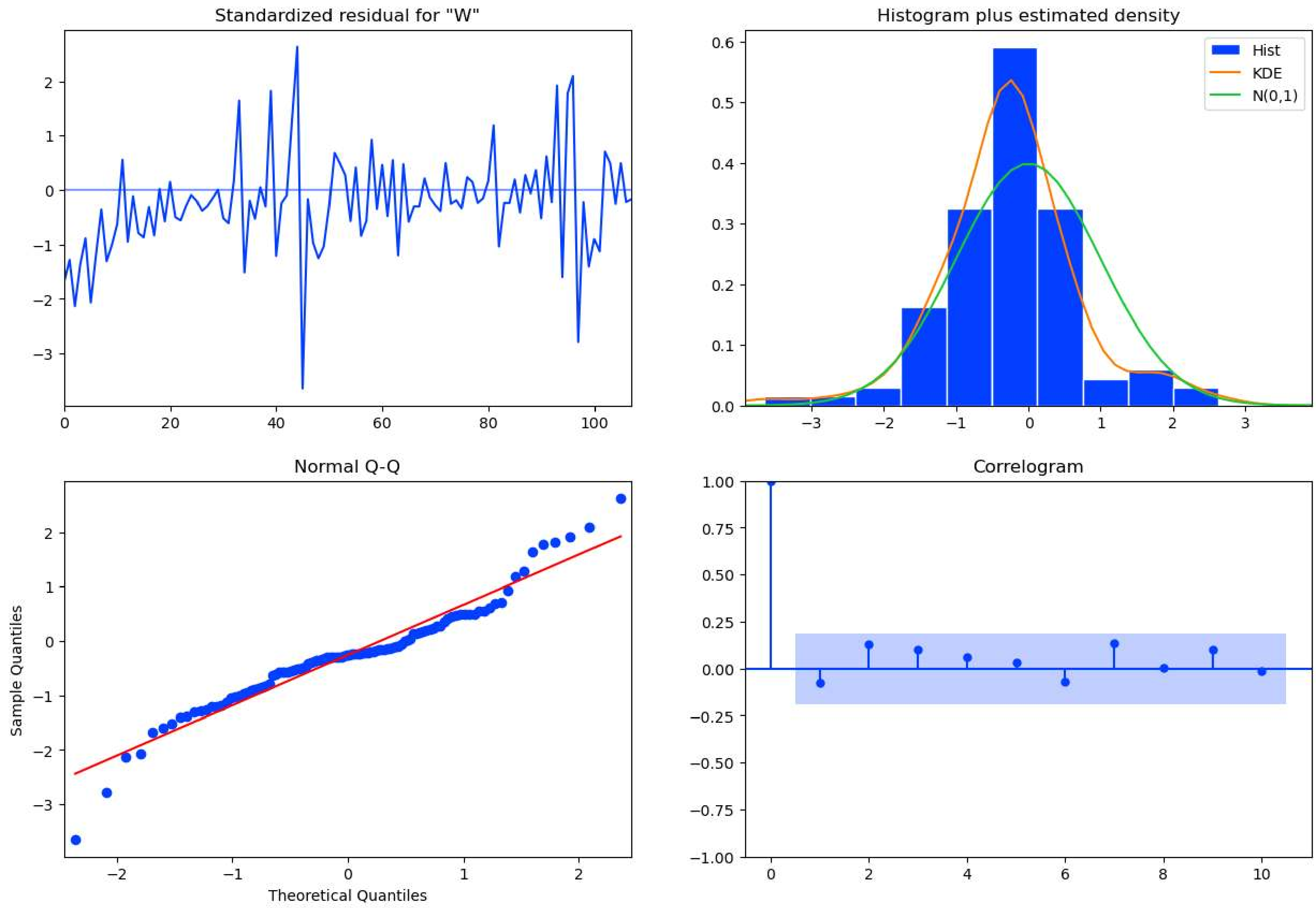
Cross Checking Auto ARIMA and ARIMA

In [102…
```python
Model_3 = ARIMA(df_train, order=(4,2,1))
Result_ARIMA_3 = Model_3.fit()
```

In [103…
```python
print(Result_ARIMA_3.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:           Weekly_Sales   No. Observations:                  110
Model:                 ARIMA(4, 2, 1)   Log Likelihood               -1542.051
Date:                Mon, 13 Feb 2023   AIC                           3096.102
Time:                        13:42:53   BIC                           3112.195
Sample:                             0   HQIC                          3102.627
                                - 110                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3572      0.071     -5.043      0.000      -0.496      -0.218
ar.L2         -0.4158      0.090     -4.624      0.000      -0.592      -0.240
ar.L3         -0.2545      0.097     -2.622      0.009      -0.445      -0.064
ar.L4         -0.1432      0.084     -1.695      0.090      -0.309       0.022
ma.L1         -0.9849      0.083    -11.810      0.000      -1.148      -0.821
sigma2      1.548e+11   1.99e-13   7.76e+23      0.000    1.55e+11    1.55e+11
===================================================================================
Ljung-Box (L1) (Q):                   0.65   Jarque-Bera (JB):                21.29
Prob(Q):                              0.42   Prob(JB):                         0.00
Heteroskedasticity (H):               1.01   Skew:                            -0.00
Prob(H) (two-sided):                  0.99   Kurtosis:                         5.18
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 8.35e+38. Standard errors may be unstable.
```

In [104…
```python
Result_ARIMA_3.plot_diagnostics(figsize =(15,10))
plt.show()
```



In [105…

```
train_predict = Result_ARIMA_3.predict()
train_predict = train_predict.to_frame()
```

In [106…
```
train_predict.head()
```

Out[106]:

	predicted_mean
Date	
2010-01-10	0.000000e+00
2010-02-04	2.944880e+06
2010-02-07	3.036746e+06
2010-02-19	2.793917e+06
2010-02-26	2.875507e+06

In [107…
```
#Calculating the MAPE (Mean_Absolute_Percentage_Error)

print("Train MAPE Model_3:", mean_absolute_percentage_error(df_train.Weekly_Sales,train_predict.predicted_mean))
```

```
Train MAPE Model_3: 14.187807737885297
```

In [108…
```
test_predict_3 = Result_ARIMA_3.forecast(10)
```

In [109…
```
test_predict_3 = test_predict_3.to_frame()
```

In [110…
```
test_predict_3.head()
```

Out[110]:

	predicted_mean
110	2.135147e+06
111	2.144273e+06
112	2.151703e+06
113	2.148368e+06
114	2.150845e+06

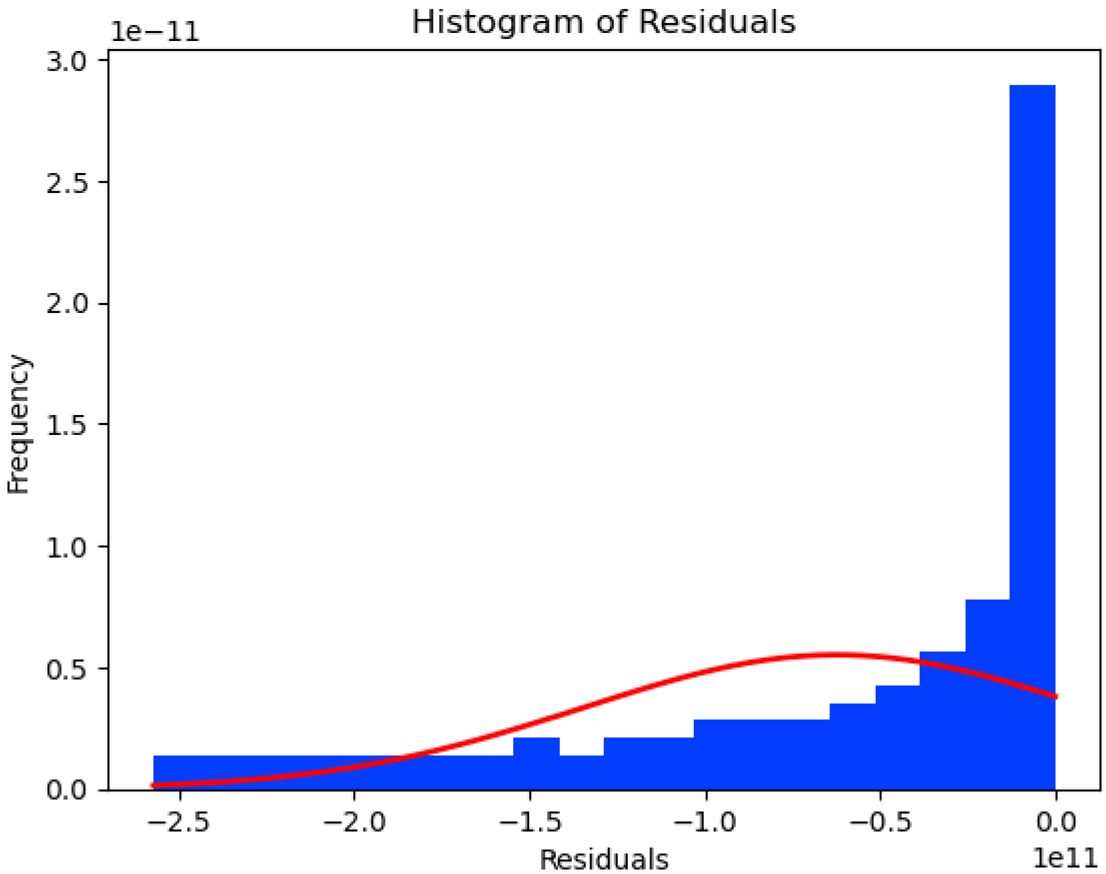
In [111…
```
#Calculating the MAPE (Mean_Absolute_Percentage_Error)

print("Test MAPE Model_3:", mean_absolute_percentage_error(df_test.Weekly_Sales,test_predict_3.predicted_mean))
```

```
Test MAPE Model_3: 7.037904733732731
```

In [ ]:

In [113…

```python
from scipy import stats
# Calculate the residuals
residuals = results.resid

# Plot the residuals as a histogram
plt.hist(residuals, bins=20, density=True)

# Fit a normal distribution to the residuals
mu, std = stats.norm.fit(residuals)
x = np.linspace(residuals.min(), residuals.max(), 100)
pdf = stats.norm.pdf(x, mu, std)
plt.plot(x, pdf, 'r', linewidth=2)

plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()

# Perform the Shapiro-Wilk test for normality
w, p = stats.shapiro(residuals)
print("Shapiro-Wilk test statistic: ", w)
print("Shapiro-Wilk p-value: ", p)
```




```
Shapiro-Wilk test statistic:  0.8144296407699585
Shapiro-Wilk p-value:  1.876110933274333e-10
```

I.Future Possibilities of the Project

The future possibilities of a time series analysis project on the Walmart data set can depend on the specific goals and objectives of the project. However, some common areas for future work include:

> - **Refining the model:** The model can be further refined by testing different time series models and parameters, and selecting the one that best fits the data and provides the most accurate forecasts.

> - **Improved forecasting:** The accuracy of the forecasts can be improved by incorporating additional information, such as economic indicators or competitor data, into the model.
> - **Forecast uncertainty:** The model can be modified to provide a measure of uncertainty around the forecasts, which can be useful for decision-making purposes.
> - **Multivariate analysis:** The analysis can be expanded to include multiple variables, such as prices or promotions, to better understand the factors that influence sales.
> - **Long-term planning:** The results of the analysis can be used to develop long-term plans for sales and inventory management, resource allocation, and business strategy.
> - **Predictive maintenance:** The analysis can be applied to other data sets, such as machine learning data, to improve predictive maintenance and reduce costs.

These are just a few of the many possibilities for future work on a time series analysis project on the Walmart data set. The specific future work will depend on the goals and objectives of the project, as well as the data available and the resources available for the analysis.

Just trying to apply Regression model on this data

In [114…

```
# import the preprocessing classes
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# import train/test split module
```

```python
from sklearn.model_selection import train_test_split

# import the regressors
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline

# import metrics
from sklearn.metrics import mean_squared_error

# import warnings
import warnings
warnings.filterwarnings('ignore')
```

In [115…

```python
df_new
```

Out[115]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	month	month_name	week	year	Day
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	5	May	17	2010	
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	12	December	48	2010	T
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	2	February	7	2010	
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	2	February	8	2010	
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	5	May	18	2010	
...	...	...	...	...	...	...	...	...	...	...	...	...	
6430	45	2012-09-28	713173.95	0	64.88	3.997	192.013558	8.684	9	September	39	2012	
6431	45	2012-05-10	733455.07	0	64.89	3.985	192.170412	8.667	5	May	19	2012	T
6432	45	2012-12-10	734464.36	0	54.47	4.000	192.327265	8.667	12	December	50	2012	
6433	45	2012-10-19	718125.53	0	56.47	3.969	192.330854	8.667	10	October	42	2012	
6434	45	2012-10-26	760281.43	0	58.85	3.882	192.308899	8.667	10	October	43	2012	

6435 rows × 13 columns

In [116…

```
# make a copy of the dataset
df_regression = df_new.copy()
```

In [117…
```
# drop the date and unemployment columns
df_regression.drop(['Date', 'Unemployment','month_name','week','Day_Name'], axis=1, inplace=True)
# check

df_regression.head()
```

Out[117]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	month	year
0	1	1643690.90	0	42.31	2.572	211.096358	5	2010
1	1	1641957.44	1	38.51	2.548	211.242170	12	2010
2	1	1611968.17	0	39.93	2.514	211.289143	2	2010
3	1	1409727.59	0	46.63	2.561	211.319643	2	2010
4	1	1554806.68	0	46.50	2.625	211.350143	5	2010

In [118…
```
X = df_regression.drop('Weekly_Sales', axis=1)
y = df_regression['Weekly_Sales']
```

Scaling the features Scaling is a preprocessing step that transforms the features of a dataset so that they have a similar scale and can improve the performance of some regression algorithms and facilitate comparison of the model's coefficients. In this project we will use standard scaler to standardize the features of the dataset.

In [119…
```
# scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [120…
```
# split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=50)
```

In [121…
```
def evaluate_model(model, X_train, y_train, X_test, y_test):
    """
    Evaluate a model on training and test data.

    Parameters
    ----------
    model : object
        A scikit-learn estimator object.
    X_train : array-like or pd.DataFrame
        Training data with shape (n_samples, n_features).
    y_train : array-like
```

```
        Training labels with shape (n_samples,).
    X_test : array-like or pd.DataFrame
        Test data with shape (n_samples, n_features).
    y_test : array-like
        Test labels with shape (n_samples,).

    Returns
    -------
    rmse : float
        Root mean squared error between the test labels and the predictions.
    """
    # train
    model.fit(X_train, y_train)
    # predict
    y_pred = model.predict(X_test)
    # calculate MSE
    mse = mean_squared_error(y_test, y_pred)
    # calculate RMSE
    rmse = np.sqrt(mse)
    return rmse
```

In [122…

```
def evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test, y_test):
    """
    This function takes a list of regressors, their names, and the training and test data as input
    and returns a dataframe with the names of the regressors and their root mean squared error (RMSE)
    on the test data.

    Parameters:
    -----------
    regressors (list): a list of scikit-learn compatible regression models
    regressor_names (list): a list of strings containing the names of the regression models
    X_train (pandas DataFrame): a pandas DataFrame containing the features for training the models
    y_train (pandas Series): a pandas Series containing the target values for training the models
    X_test (pandas DataFrame): a pandas DataFrame containing the features for testing the models
    y_test (pandas Series): a pandas Series containing the target values for testing the models

    Returns:
    --------
    pandas DataFrame: a dataframe containing the names of the regressors and their corresponding RMSE on the test data
    """

    # evaluate the models and compute their RMSE on the test data
    rmses = [evaluate_model(regressor, X_train, y_train, X_test, y_test) for regressor in regressors]
```

```python
    # create a dictionary mapping the names of the regressors to their RMSE
    regressor_rmses = dict(zip(regressor_names, rmses))

    # convert the dictionary to a pandas dataframe
    df = pd.DataFrame.from_dict(regressor_rmses, orient='index')

    # reset the index of the dataframe
    df = df.reset_index()

    # rename the columns of the dataframe
    df.columns = ['regressor_name', 'rmse']

    # sort the dataframe by RMSE in ascending order
    return df.sort_values('rmse', ignore_index=True)
```

In [123…

```python
# initialize the regressors
linear_regressor = LinearRegression()
decision_tree_regressor = DecisionTreeRegressor()
random_forest_regressor = RandomForestRegressor()
boosted_tree_regressor = GradientBoostingRegressor()
support_vector_regressor = SVR()
knn_regressor = KNeighborsRegressor(n_neighbors=5, weights='uniform')
```

In [124…

```python
# collect the list of regressors
regressors = [linear_regressor, decision_tree_regressor, random_forest_regressor,
              boosted_tree_regressor, support_vector_regressor, knn_regressor]

# collect the names of regressors
regressor_names = ["Linear Regression", "Decision Tree Regression",
                   "Random Forest Regression","Boosted Tree Regression", "Support Vector Regression",
                   "K-Nearest Neighbour Regression"]
```

In [125…

```python
print('\033[1m Table of regressors and their RMSEs')
evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test, y_test)
```

Table of regressors and their RMSEs

Out[125]:

	regressor_name	rmse
0	Random Forest Regression	143675.203275
1	Boosted Tree Regression	186705.186164
2	Decision Tree Regression	195783.202244
3	K-Nearest Neighbour Regression	481499.886054
4	Linear Regression	521772.888793
5	Support Vector Regression	568793.169161

In [126…

```
# evaluate rmse for the regressors
rmse = evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test, y_test)
```

In [127…

```
# pick the best rmse
best_rmse = rmse.iloc[0]['rmse']
# compute the median of the weekly sales
median_sale = df_regression['Weekly_Sales'].median()
# compute percentage error
percent_deviation = round((best_rmse*100/median_sale), 2)
# print the result
print('The model has average percentage error of {}%'.format(percent_deviation))
```

```
The model has average percentage error of 15.2%
```
