DSM

May 7, 2024

[7]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from pandas.plotting import scatter_matrix
import warnings
from datetime import datetime
```

[8]:

```
df = pd.read_csv(r"C:/Users/Abdo/Desktop/Projects/DSM_Project/walmart-sales.
 ↪csv")
```

[9]:

```
df.head()
```

[9]:

```
   Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0      1  05-02-2010    1643690.90             0        42.31       2.572
1      1  12-02-2010    1641957.44             1        38.51       2.548
2      1  19-02-2010    1611968.17             0        39.93       2.514
3      1  26-02-2010    1409727.59             0        46.63       2.561
4      1  05-03-2010    1554806.68             0        46.50       2.625

          CPI  Unemployment
0  211.096358         8.106
1  211.242170         8.106
2  211.289143         8.106
3  211.319643         8.106
4  211.350143         8.106
```

[10]:

```
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
```

[11]:

```
df.dtypes
```

[11]:

```
Store                    int64
Date            datetime64[ns]
Weekly_Sales           float64
Holiday_Flag             int64
Temperature            float64
Fuel_Price             float64
```

```
CPI                        float64
Unemployment               float64
dtype: object
```

[12]:

```
# Data Preprocessing:

# Display sum of missing values before cleaning
# Replace infinite values with NaN
print("Sum of NA before cleaning:")
print(df.isna().sum())

# Drop rows with missing values
df_cleaned = df.dropna()

# Display sum of missing values after cleaning
print("\nSum of NA after cleaning:")
print(df_cleaned.isna().sum())

# Check for duplicates
print("\nDuplicates before removal:")
print(df_cleaned.duplicated().sum())

# Remove duplicates
df_no_duplicates = df_cleaned.drop_duplicates()

#df handling inconsistencies
df_no_duplicates = df.columns.str.lower()
print("\nDuplicates after removal:")
print(df_no_duplicates.duplicated().sum())

# Handling missing values
# For Temperature, Fuel_Price, CPI, and Unemployment columns, we'll fill␣
  ↪missing values with the median
df['Temperature'].fillna(df['Temperature'].median(), inplace=True)
df['Fuel_Price'].fillna(df['Fuel_Price'].median(), inplace=True)
df['CPI'].fillna(df['CPI'].median(), inplace=True)
df['Unemployment'].fillna(df['Unemployment'].median(), inplace=True)
df.replace([np.inf, -np.inf], np.nan, inplace=True)

numerical_columns = ['Store','Weekly_Sales',
                     'Holiday_Flag','Temperature',
                     'Fuel_Price','CPI','Unemployment']
```

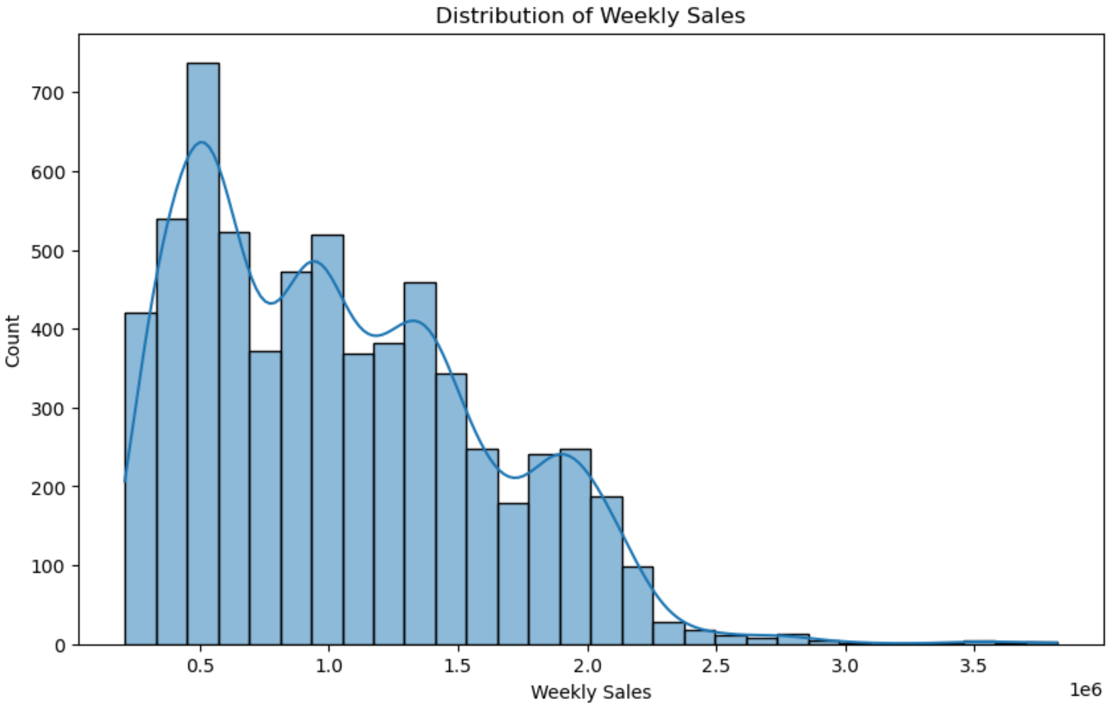
```
Sum of NA before cleaning:
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
```

```
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64

Sum of NA after cleaning:
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64

Duplicates before removal:
0

Duplicates after removal:
0
```

[13]:

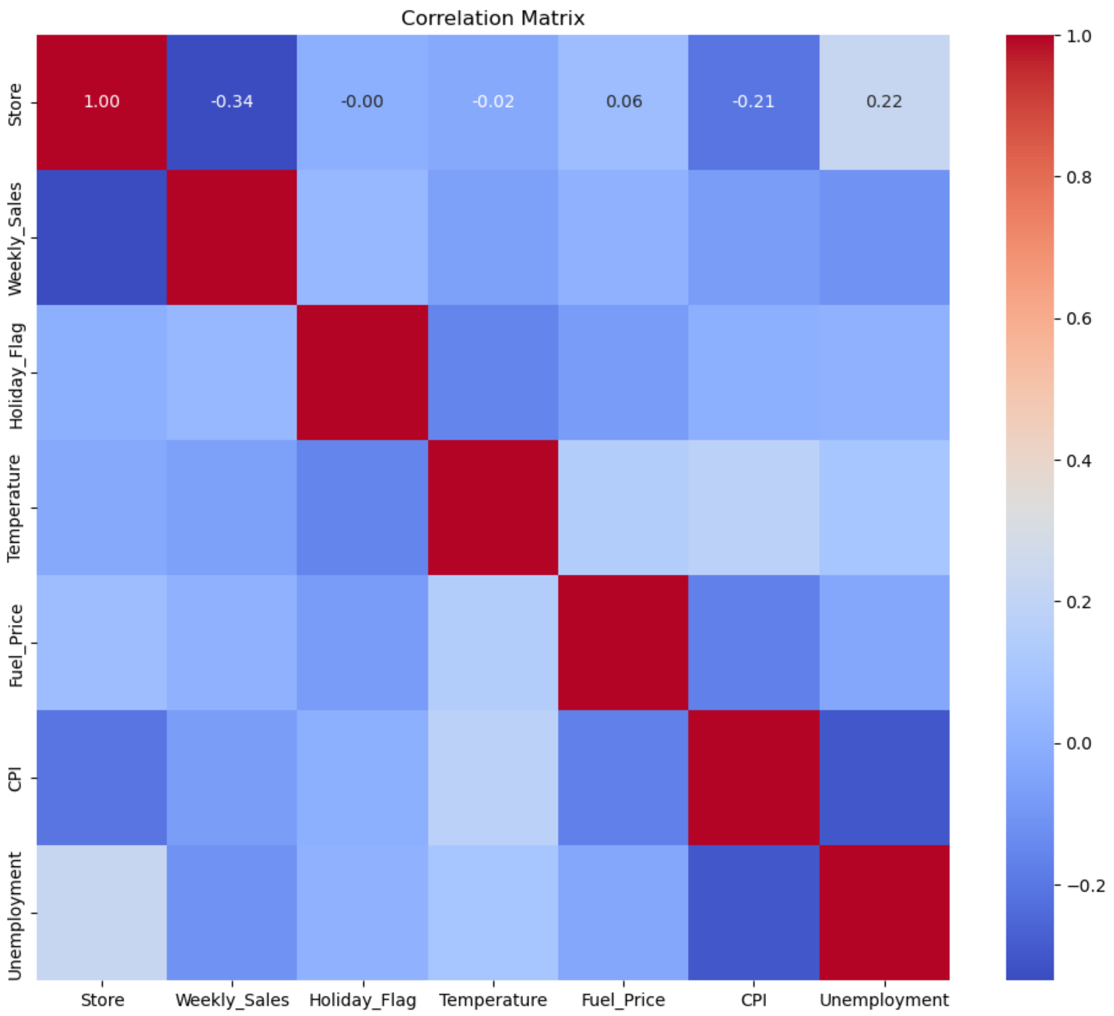
```python
#histograms
# Plotting distribution of weekly sales
plt.figure(figsize=(10, 6))
sns.histplot(df['Weekly_Sales'], bins=30, kde=True)
plt.title('Distribution of Weekly Sales')
plt.xlabel('Weekly Sales')
plt.show()
```

```
C:\Users\Abdo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[14]:

```python
# Calculate correlation matrix
correlation_matrix = df[numerical_columns].corr()

# Create heatmap for correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



[15]:

```
df_max_sales = df.groupby('Store')['Weekly_Sales'].sum().idxmax()
print(f"The store with maximum sales is Store {df_max_sales}")
```

```
The store with maximum sales is Store 20
```

[16]:

```
df_max_std_dev = df.groupby('Store')['Weekly_Sales'].std().idxmax()
print(f"The store with maximum standard deviation of sales is Store␣
  ↪{df_max_std_dev}")
```

```
The store with maximum standard deviation of sales is Store 14
```

[17]:

```
# Calculate the mean of weekly sales in the non-holiday season across all stores
overall_non_holiday_mean_sales = df[df['Holiday_Flag'] == 0]['Weekly_Sales'].
  ↪mean()
```

```
# Find holidays with sales higher than the overall mean sales in the␣
 ↪non-holiday season
holiday_sales_above_mean = df[(df['Holiday_Flag'] == 1) & (df['Weekly_Sales'] >␣
 ↪overall_non_holiday_mean_sales)]

# Extract unique dates from the DataFrame
unique_dates = holiday_sales_above_mean['Date'].unique()

# Print the unique dates of holidays with higher sales than the mean sales in␣
 ↪the non-holiday season
print("Unique dates of holidays with higher sales than the mean sales in the␣
 ↪non-holiday season for all stores together:")
print(unique_dates)
```

```
Unique dates of holidays with higher sales than the mean sales in the non-
holiday season for all stores together:
<DatetimeArray>
['2010-02-12 00:00:00', '2010-09-10 00:00:00', '2010-11-26 00:00:00',
 '2010-12-31 00:00:00', '2011-02-11 00:00:00', '2011-09-09 00:00:00',
 '2011-11-25 00:00:00', '2011-12-30 00:00:00', '2012-02-10 00:00:00',
 '2012-09-07 00:00:00']
Length: 10, dtype: datetime64[ns]
```

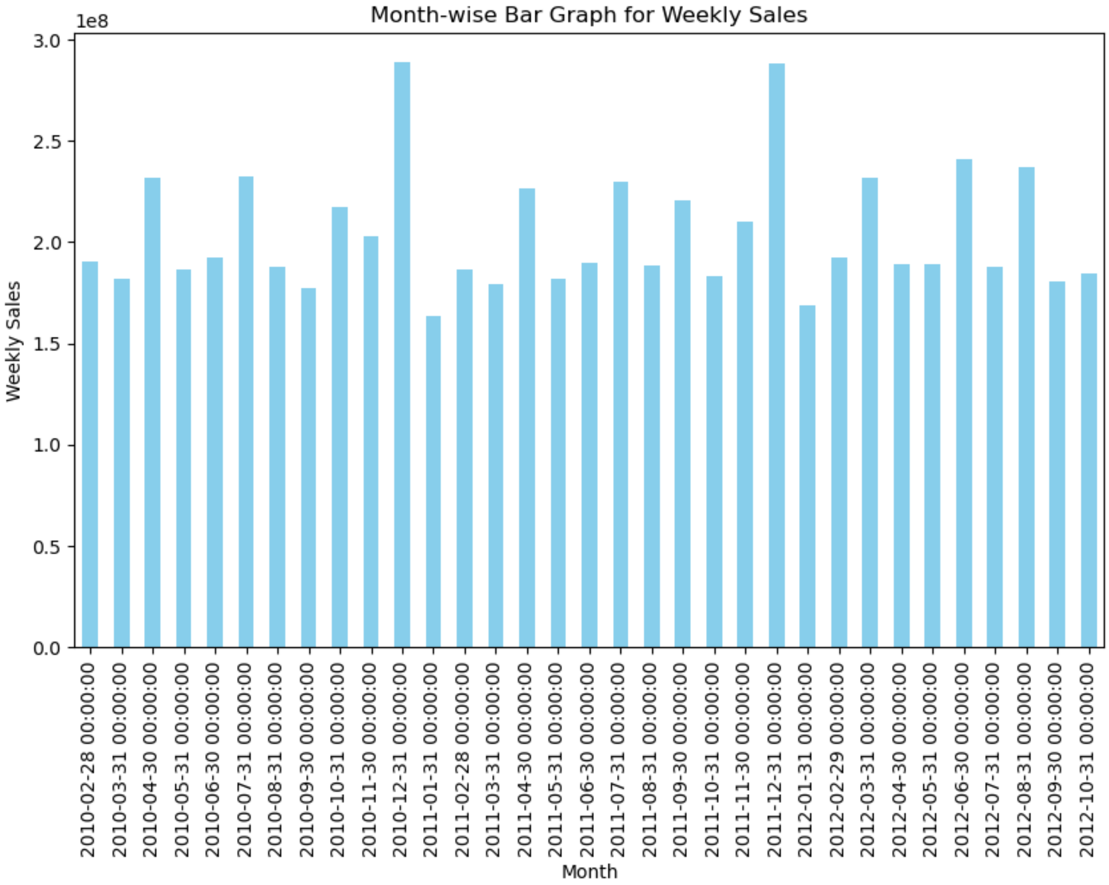
[18]:

```
# Provide a monthly view of sales
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
df['Month'] = df['Date'].dt.month
monthly_sales = df.groupby('Month')['Weekly_Sales'].sum()
print("Monthly view of sales:")
print(monthly_sales)
```

```
Monthly view of sales:
Month
1     3.325984e+08
2     5.687279e+08
3     5.927859e+08
4     6.468598e+08
5     5.571256e+08
6     6.226299e+08
7     6.500010e+08
8     6.130902e+08
9     5.787612e+08
10    5.847848e+08
11    4.130157e+08
12    5.768386e+08
Name: Weekly_Sales, dtype: float64
```

[22]:

```
# Group by month
monthly_sales = df.resample('M', on='Date')['Weekly_Sales'].sum()
# Plot month bar graph
plt.figure(figsize=(10, 6))
monthly_sales.plot(kind='bar', color='skyblue')
plt.xlabel('Month')
plt.ylabel('Weekly Sales')
plt.title('Month-wise Bar Graph for Weekly Sales')
plt.show()
```



[19]:

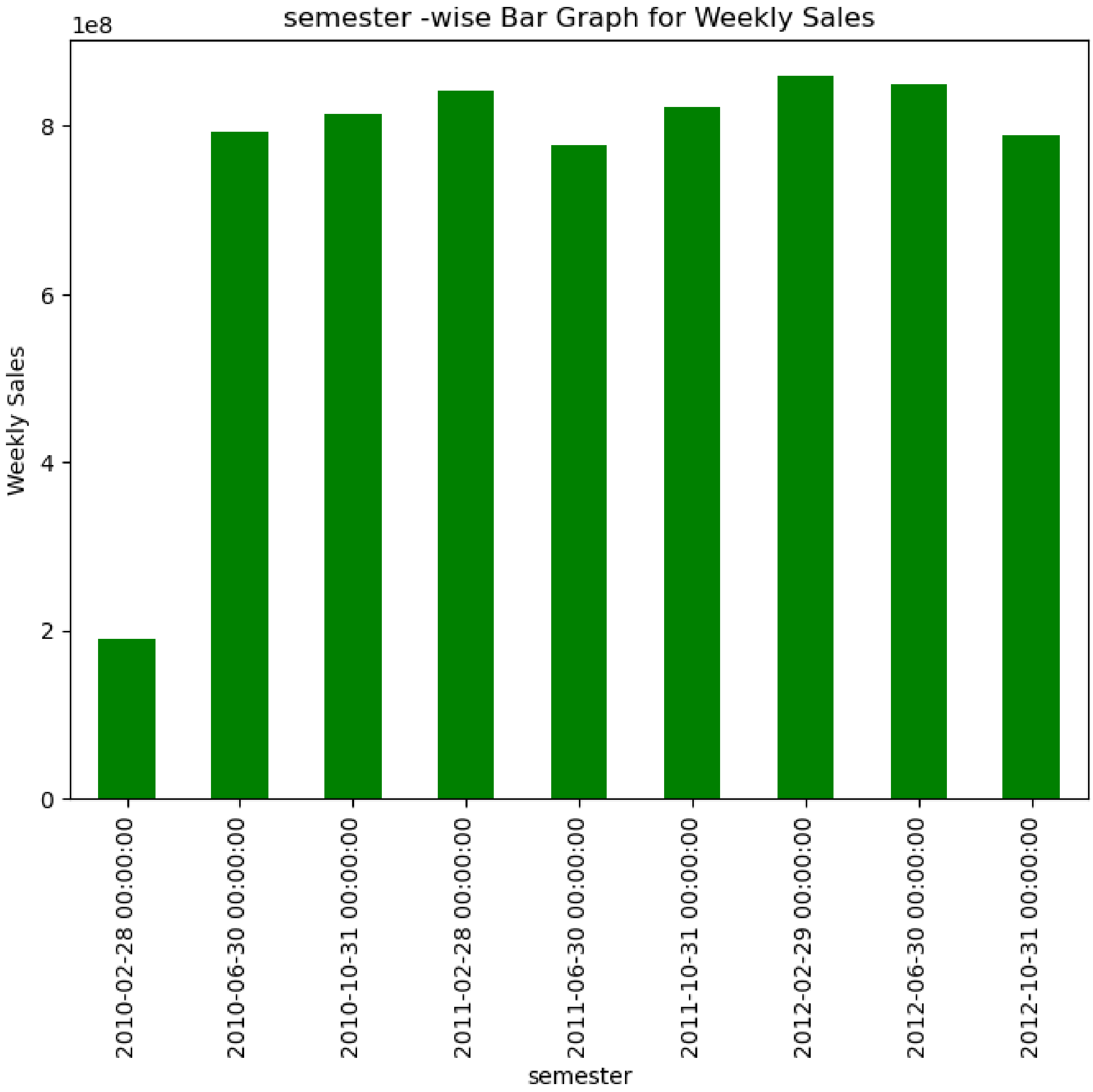
```
# Provide a semester view of sales
df['Semester'] = df['Date'].dt.quarter
semester_sales = df.groupby('Semester')['Weekly_Sales'].sum()
print("\nSemester view of sales:")
print(semester_sales)
```

```
Semester view of sales:
Semester
```

```
1    1.494112e+09
2    1.826615e+09
3    1.841852e+09
4    1.574639e+09
Name: Weekly_Sales, dtype: float64
```

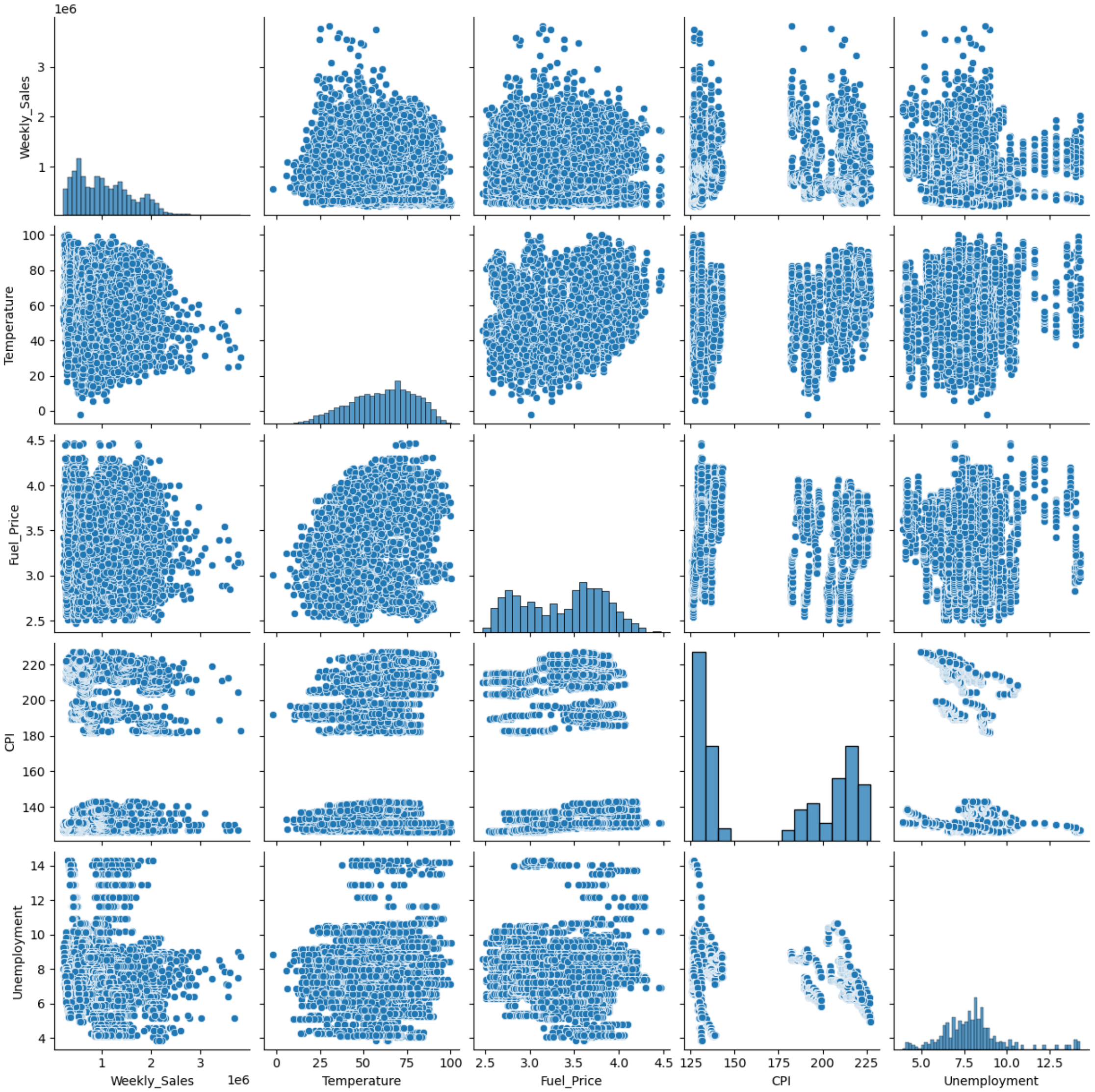
[25]:

```
# Group by semester
semester_sales = df.resample('4M', on='Date')['Weekly_Sales'].sum()
# Plotting
plt.figure(figsize=(8, 6))
semester_sales.plot(kind='bar', color='green')
plt.xlabel('semester')
plt.ylabel('Weekly Sales')
plt.title('semester -wise Bar Graph for Weekly Sales')
plt.show()
```



[20]:

```python
# Plotting relationships between weekly sales and other numeric features
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    with pd.option_context('mode.use_inf_as_na', True):
        sns.pairplot(df[['Weekly_Sales', 'Temperature', 'Fuel_Price', 'CPI',␣
 ↪'Unemployment']].replace([np.inf, -np.inf], np.nan))
        plt.show()
```



[ ]: