In [17]:
```
#Step 1: Import Required Libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

In [18]:
```
#Step 2: Load the Dataset
df=pd.read_csv(r"D:\Downloads\archive (1)\Walmart.csv")
df.head()
```

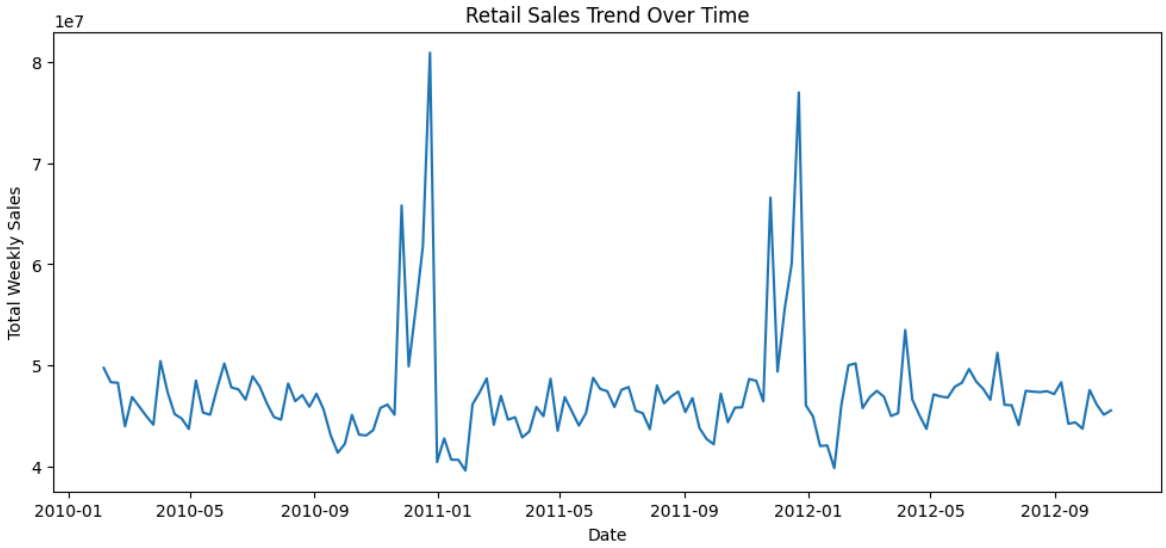
Out[18]:

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

In [19]:
```
#Step 3: Convert Date & Prepare Time Series
# Convert Date column to datetime
df['Date'] = pd.to_datetime(df['Date'],dayfirst=True)

# Sort by date
df = df.sort_values('Date')

# Aggregate weekly sales across all stores
sales_ts = df.groupby('Date')['Weekly_Sales'].sum()

sales_ts.head()
```

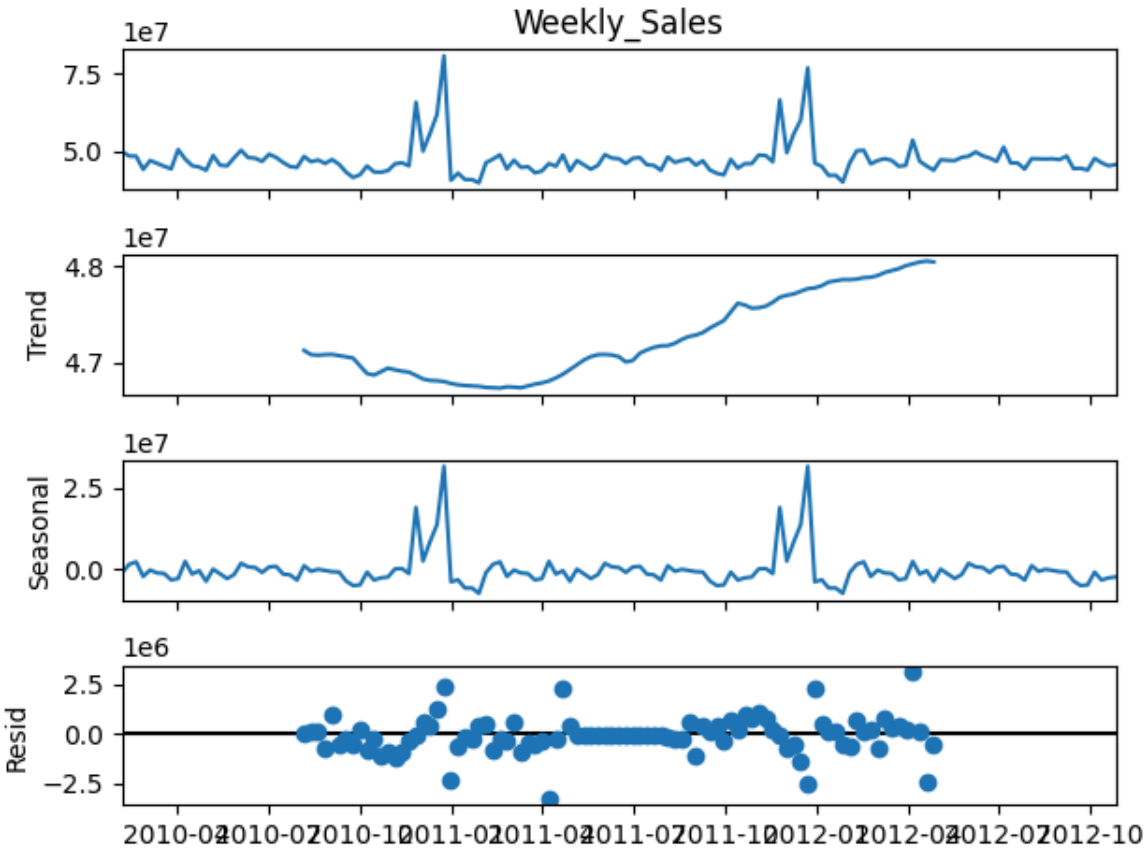
Out[19]:
```
Date
2010-02-05    49750740.50
2010-02-12    48336677.63
2010-02-19    48276993.78
2010-02-26    43968571.13
2010-03-05    46871470.30
Name: Weekly_Sales, dtype: float64
```

In [20]:
```
#Step 4:Plot Sales Trend Over Time
plt.figure(figsize=(12,5))
plt.plot(sales_ts)
plt.title("Retail Sales Trend Over Time")
plt.xlabel("Date")
plt.ylabel("Total Weekly Sales")
plt.show()
```



In [ ]:
```
"""
Step 4: Plot Sales Trend Over Time

This step visualizes the total weekly retail sales across all stores.
The line plot helps observe how sales change over time and identifies
overall trends, fluctuations, and patterns in the sales data.
"""
```

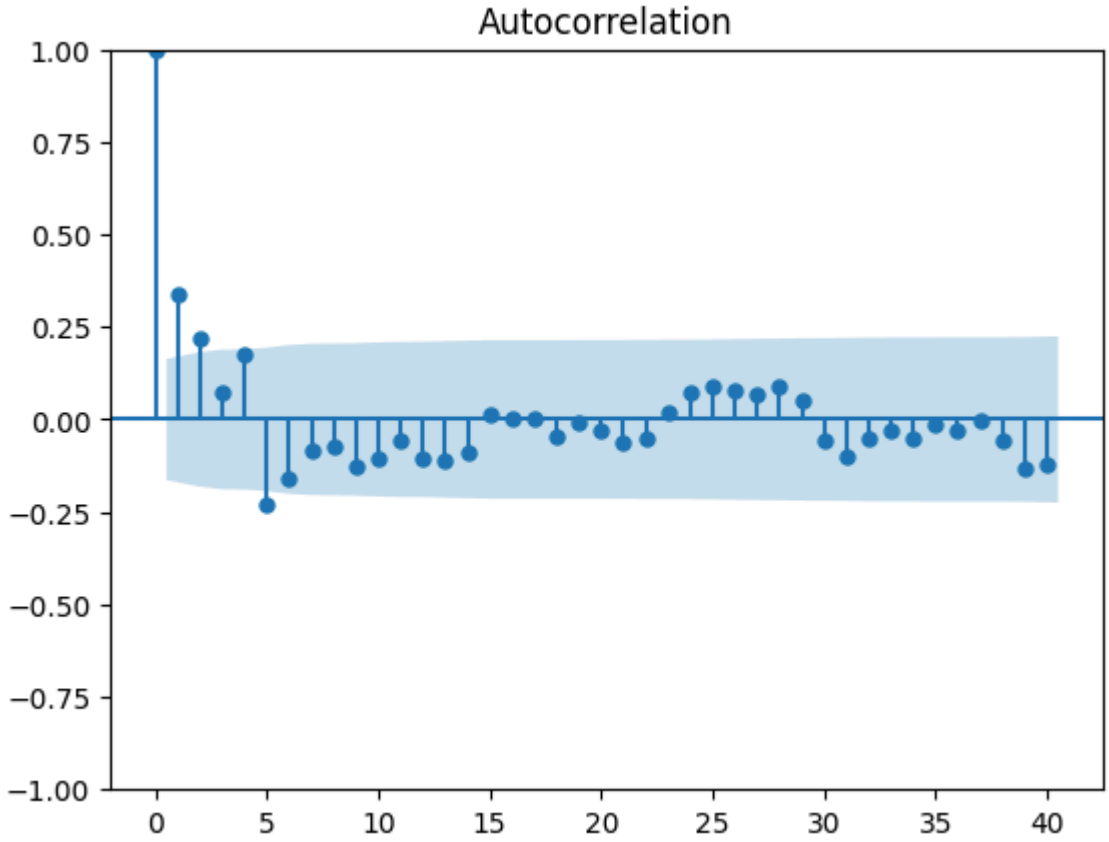
In [21]:
```
#Step 5:Seasonal Decomposition (Trend + Seasonality + Noise)
decomposition = seasonal_decompose(sales_ts, model='additive', period=52)

decomposition.plot()
plt.show()
```



In [ ]:
```
"""
Step 5: Seasonal Decomposition

This step decomposes the retail sales time series into three components:
trend, seasonality, and residual (noise).
The additive model is used because seasonal variations remain
approximately constant over time.
The period is set to 52 to represent yearly seasonality
in weekly sales data.
"""
```

In [22]:
```
#Step 6:Autocorrelation (ACF) Plot
plt.figure(figsize=(10,4))
plot_acf(sales_ts, lags=40)
plt.show()
```

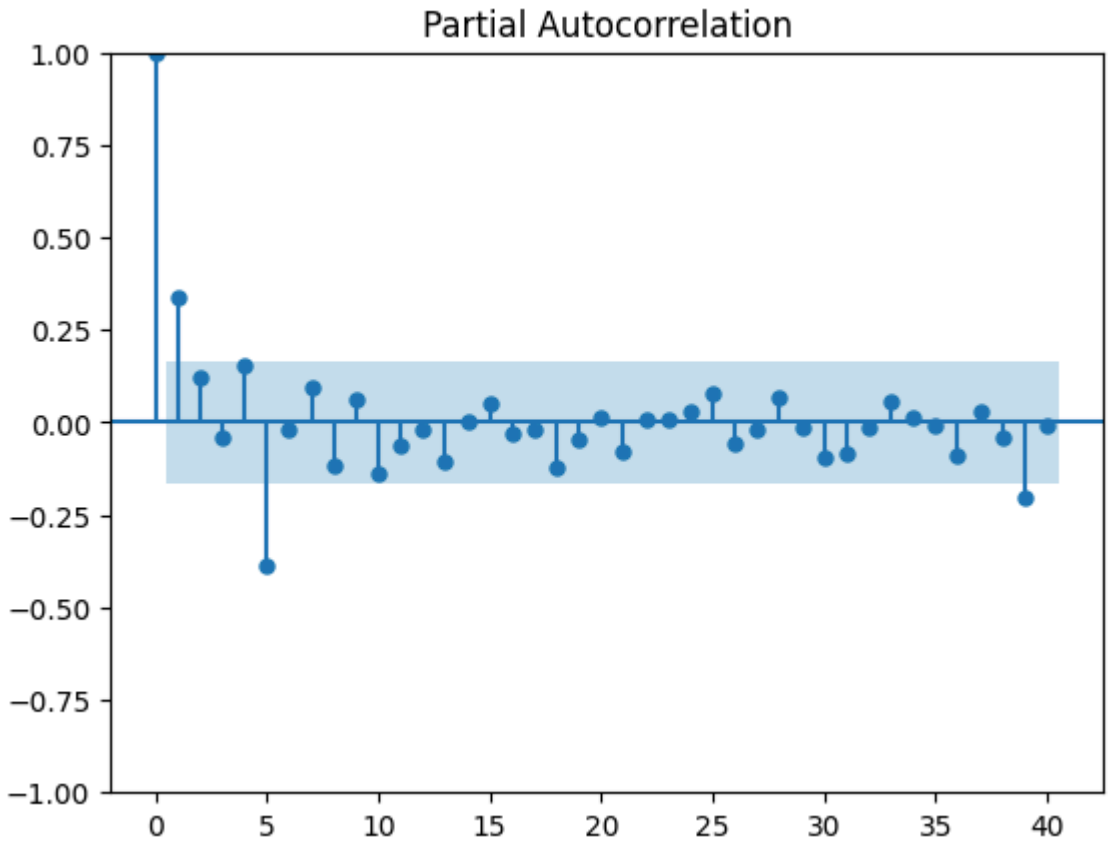
```
<Figure size 1000x400 with 0 Axes>
```



In [ ]:
```
"""
Step 6: Autocorrelation (ACF) Plot

This step plots the Autocorrelation Function (ACF) of the sales time series.
It shows the correlation between current sales values and past sales values
at different time lags.
This helps identify repeating patterns and time dependency in the data.
"""
```

In [23]:
```
#Step 7:Partial Autocorrelation (PACF) Plot
plt.figure(figsize=(10,4))
plot_pacf(sales_ts, lags=40)
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```

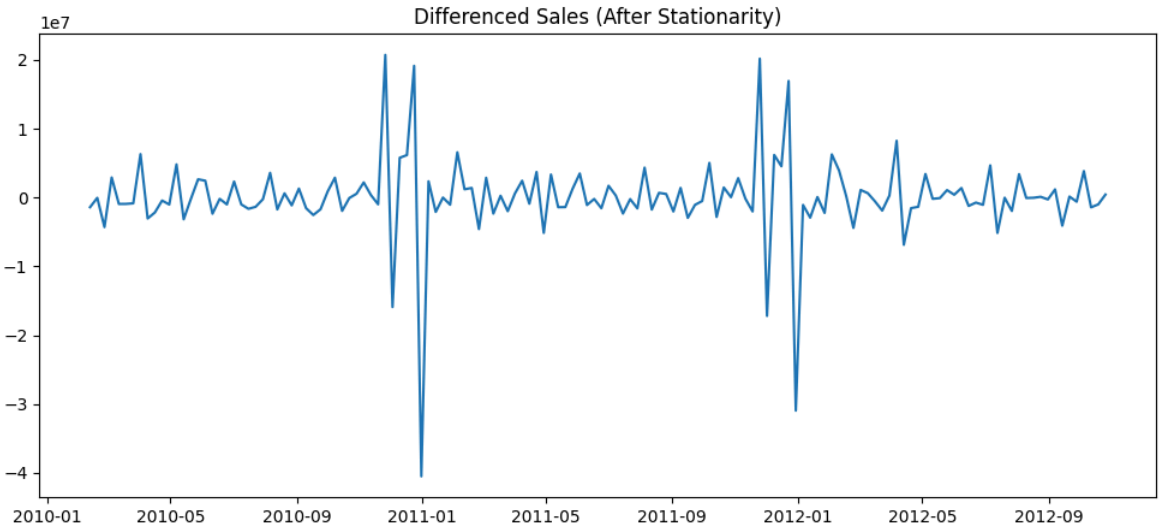


In [ ]:
```
"""
Step 7: Partial Autocorrelation (PACF) Plot

This step plots the Partial Autocorrelation Function (PACF) of the sales time se
It shows the direct relationship between current sales values and specific past
while removing the effect of intermediate lags.
This helps identify the most significant lag values in the data.
"""
```

In [24]:
```
#Step 8: Stationarity Test (ADF Test)
adf_test = adfuller(sales_ts)

print("ADF Statistic:", adf_test[0])
print("p-value:", adf_test[1])
```

```
ADF Statistic: -5.908297957186336
p-value: 2.675979158986003e-07
```

In [25]:
```
#Step 9:Make Data Stationary (If Needed)
sales_diff = sales_ts.diff().dropna()

plt.figure(figsize=(12,5))
plt.plot(sales_diff)
plt.title("Differenced Sales (After Stationarity)")
plt.show()
```



In [26]:
```
#Step 10 : Re-check stationarity:
adf_test_diff = adfuller(sales_diff)

print("ADF Statistic:", adf_test_diff[0])
print("p-value:", adf_test_diff[1])
```

```
ADF Statistic: -6.699469309617217
p-value: 3.922578707076831e-09
```