In [1]:
```
#### ARIMA MODEL OR SEASONAL ARIMA MODEL
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#### import data
data = pd.read_csv(r"C:\Users\shubham lokare\Downloads\perrin-freres-monthly-champagne-.csv")
```

In [3]:
```
data
```

Out[3]:

	Month	Perrin Freres monthly champagne sales millions ?64-?72
0	1964-01	2815.0
1	1964-02	2672.0
2	1964-03	2755.0
3	1964-04	2721.0
4	1964-05	2946.0
...	...	...
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN
106	Perrin Freres monthly champagne sales millions...	NaN

107 rows × 2 columns

In [4]:
```
data.head(10)
```

Out[4]:

	Month	Perrin Freres monthly champagne sales millions ?64-?72
0	1964-01	2815.0
1	1964-02	2672.0
2	1964-03	2755.0
3	1964-04	2721.0
4	1964-05	2946.0
5	1964-06	3036.0
6	1964-07	2282.0
7	1964-08	2212.0
8	1964-09	2922.0
9	1964-10	4301.0

In [5]:
```
### change the column name
data.columns=['Month' ,'Sales']
data
```

Out[5]:

	Month	Sales
0	1964-01	2815.0
1	1964-02	2672.0
2	1964-03	2755.0
3	1964-04	2721.0
4	1964-05	2946.0
...	...	...
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN
106	Perrin Freres monthly champagne sales millions...	NaN

107 rows × 2 columns

In [6]:
```
### check any missing values in data
data.isna().sum()     #### there are some missing values in the data
```

Out[6]:
```
Month    1
Sales    2
dtype: int64
```

In [7]:
```
data.drop(105 , axis =0 ,inplace = True)
```

In [8]:
```
data.drop(106 , axis =0 , inplace = True)
```

In [9]:
```
### then we need to chane the month data in the date and time
data['Month']=pd.to_datetime(data['Month'])
```

In [10]:
```
data
```

Out[10]:

	Month	Sales
0	1964-01-01	2815.0
1	1964-02-01	2672.0
2	1964-03-01	2755.0
3	1964-04-01	2721.0
4	1964-05-01	2946.0
...	...	...
100	1972-05-01	4618.0
101	1972-06-01	5312.0
102	1972-07-01	4298.0
103	1972-08-01	1413.0
104	1972-09-01	5877.0

105 rows × 2 columns

In [11]:
```
#### set month as indesx
data.set_index('Month' , inplace =True)
```

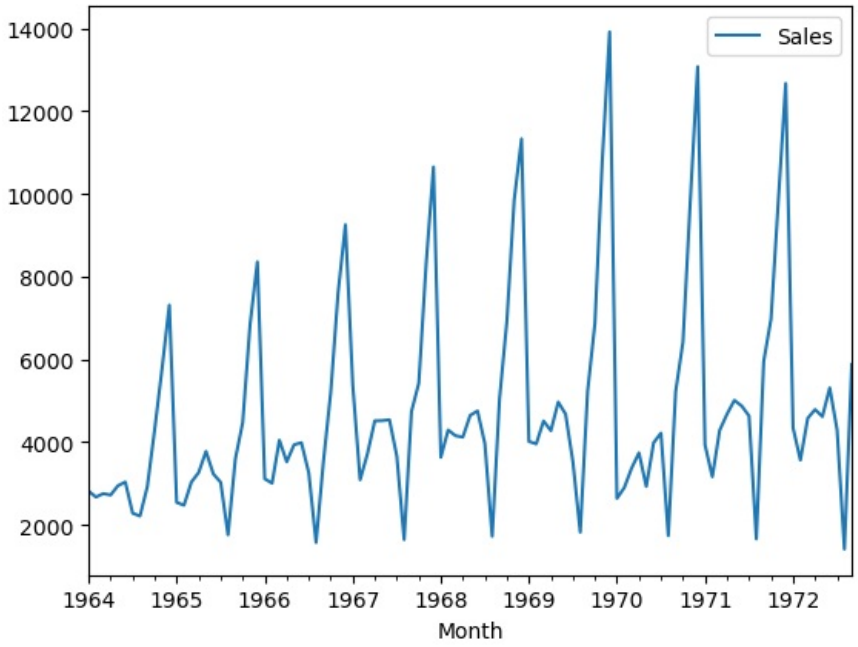
In [12]:
```
data.describe()
```

Out[12]:

	Sales
count	105.000000
mean	4761.152381
std	2553.502601
min	1413.000000
25%	3113.000000
50%	4217.000000
75%	5221.000000
max	13916.000000

In [13]:
```
### the we need to visualize the data
data.plot()
```

Out[13]:
```
<Axes: xlabel='Month'>
```



In [14]:
```
### then we need to check weather the data is stationary or not
from statsmodels.tsa.stattools import adfuller
```

In [15]:
```
test_data = adfuller(data['Sales'])
```

In [16]:
```
#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit ro
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationa
```

In [17]:
```
adfuller_test(data['Sales'])
```

```
ADF Test Statistic : -1.833593056327623
p-value : 0.363915771660245
#Lags Used : 11
Number of Observations Used : 93
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [18]:
```
#### based on that the data is not stationary
### we need to find out the difference
data['Sales Frist_defference'] = data['Sales'] - data['Sales'].shift(1)
```

In [19]:
```
### add your data having the seasonality  we need to do another shift based on seasonality
data['Sales Based on Seasonality'] = data['Sales'] - data['Sales'].shift(12)
```

In [21]:
```
data.head(10)
```

Out[21]:

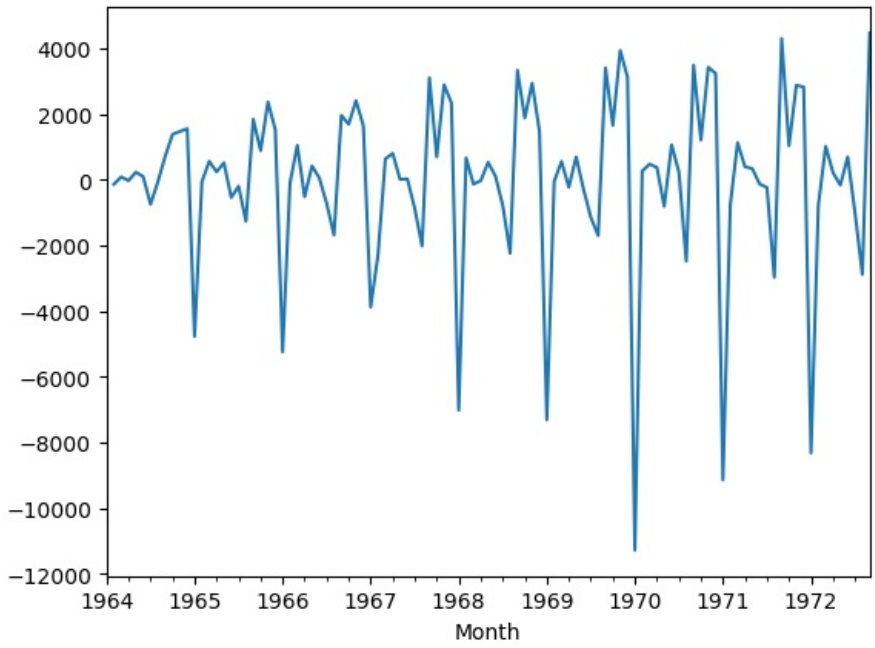
Month	Sales	Sales Frist_defference	Sales Based on Seasonality
1964-01-01	2815.0	NaN	NaN
1964-02-01	2672.0	-143.0	NaN
1964-03-01	2755.0	83.0	NaN
1964-04-01	2721.0	-34.0	NaN
1964-05-01	2946.0	225.0	NaN
1964-06-01	3036.0	90.0	NaN
1964-07-01	2282.0	-754.0	NaN
1964-08-01	2212.0	-70.0	NaN
1964-09-01	2922.0	710.0	NaN
1964-10-01	4301.0	1379.0	NaN

In [22]:
```
#### the we need to remove nan value
## Again test dickey fuller test
adfuller_test(data['Sales Based on Seasonality'].dropna())
```

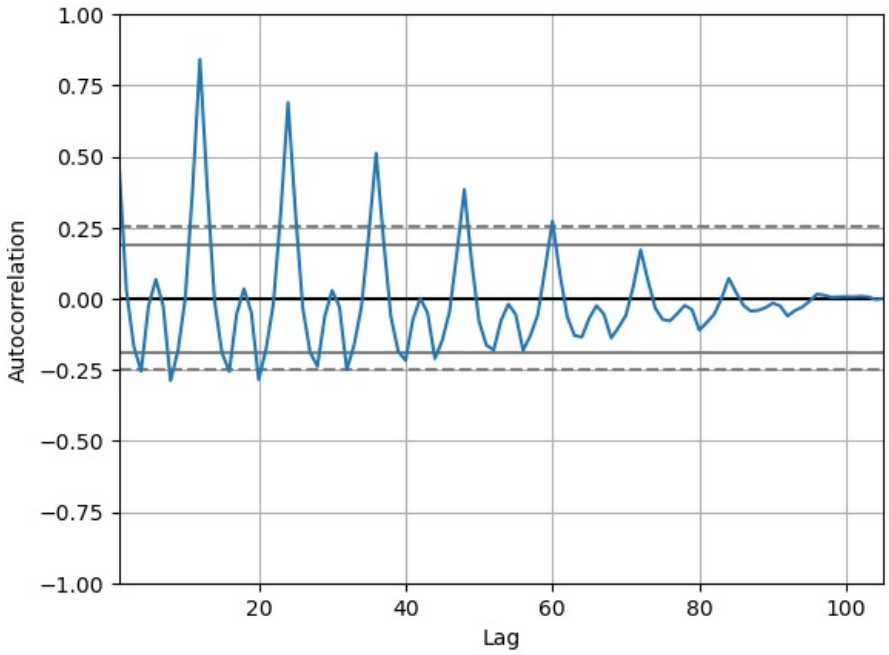
```
ADF Test Statistic : -7.626619157213163
p-value : 2.060579696813685e-11
#Lags Used : 0
Number of Observations Used : 92
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is statio
nary
```

In [23]:
```
#### plot the graph to check the data is stationary or not
### here p values > 0.05
data['Sales Frist_defference'].plot()   #### based on this the data is stationary
```

Out[23]:
```
<Axes: xlabel='Month'>
```



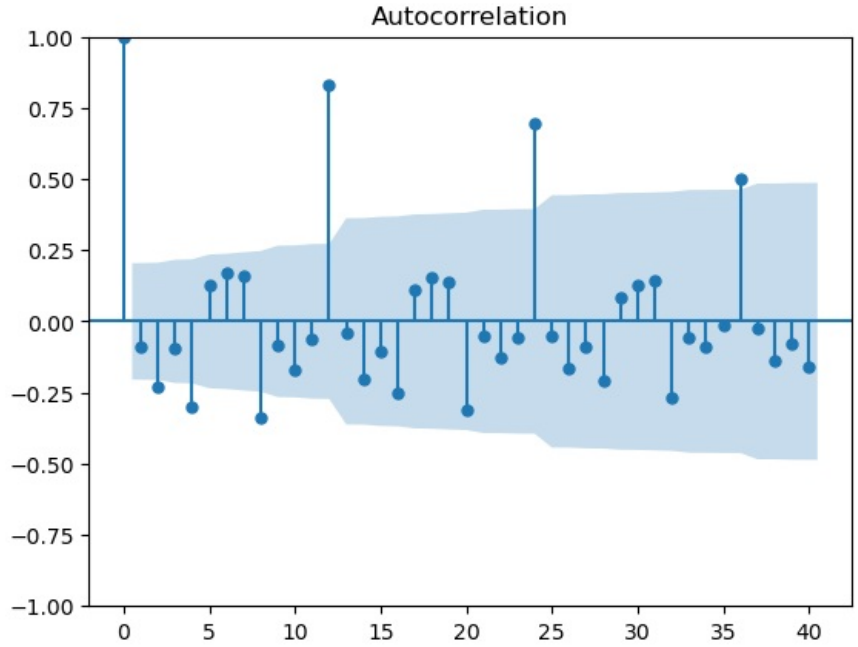
In [25]:
```
#### then we plot autocorreletion plot
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(data['Sales'])
plt.show()
```



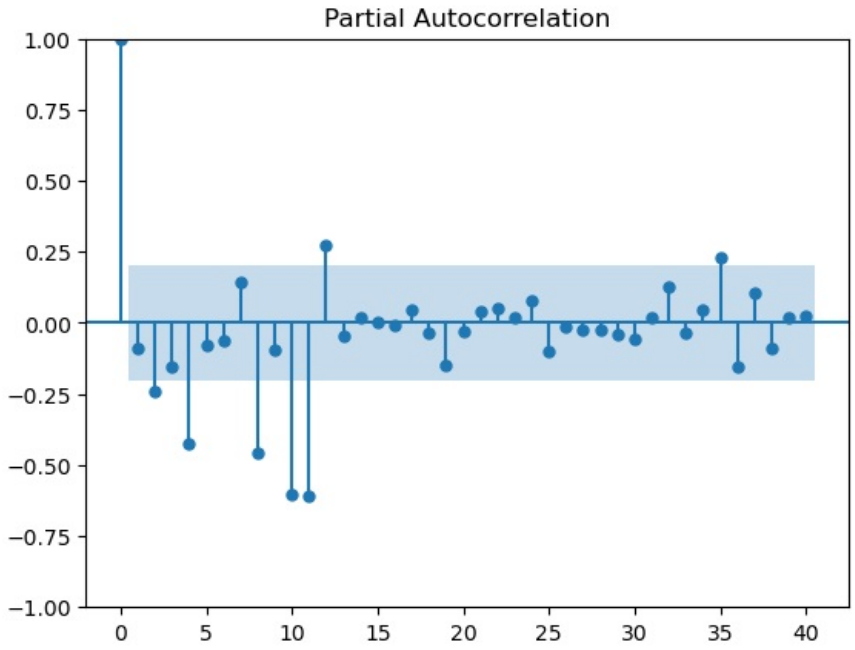
In [26]:
```
### the we plot ACF and PACF plot
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [35]:
```
#### auto correlation plot
plt.figure(figsize=(12,6))
plot1 =plot_acf(data['Sales Frist_defference'].iloc[13:] , lags=40)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



In [36]:
```
#### partial auto correlation plot
PACF_pot = plot_pacf(data['Sales Frist_defference'].iloc[13:] ,lags=40)
```



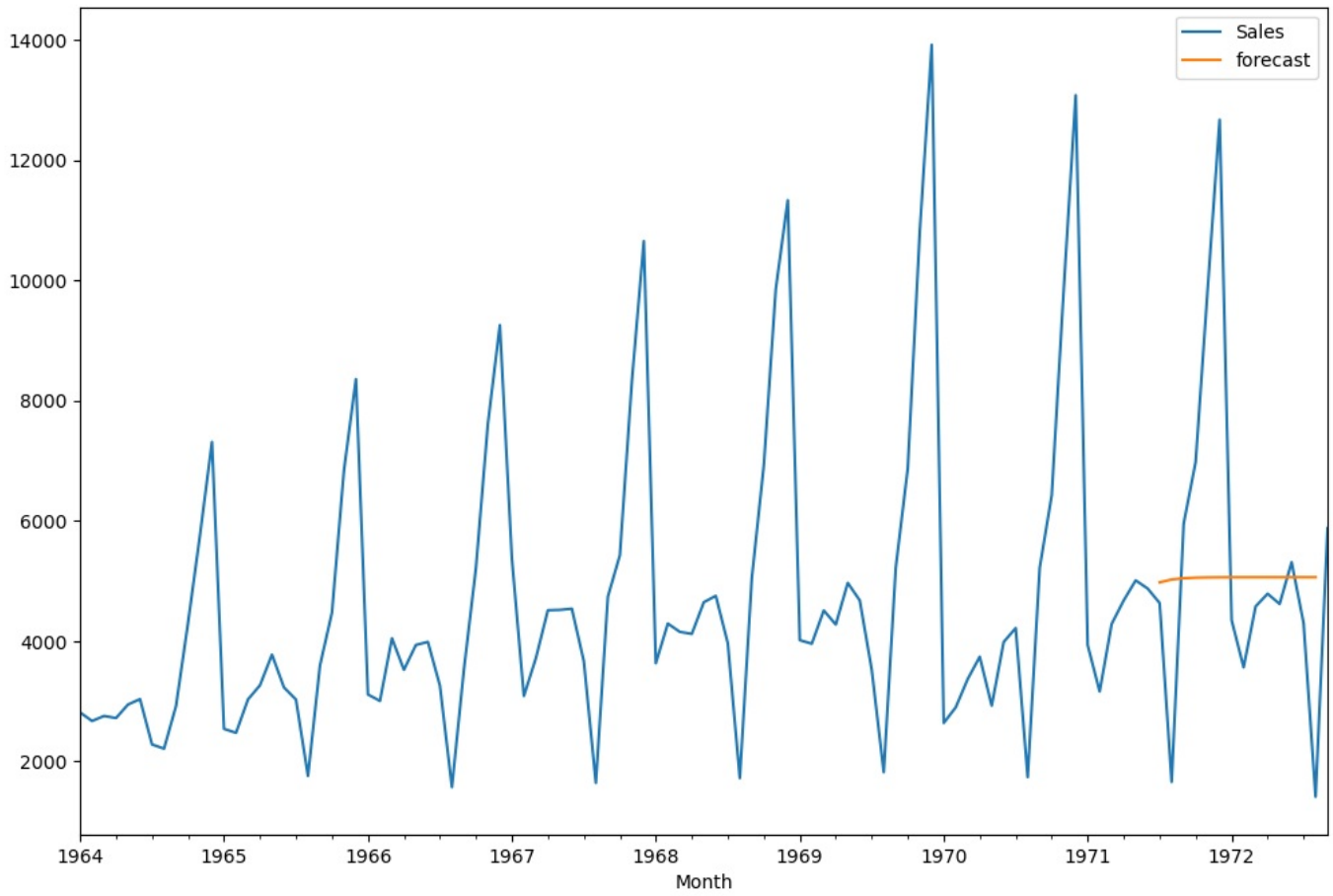
In [37]:
```
##For non-seasonal data
#p=1, d=1, q=0 or 1
from statsmodels.tsa.arima_model import ARIMA
```

In [39]:
```
from statsmodels.tsa.arima.model import ARIMA

# Fit the ARIMA model
model = ARIMA(data['Sales'], order=(1, 1, 1))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())
```

```
C:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                  105
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -952.814
Date:                Tue, 08 Oct 2024   AIC                           1911.627
Time:                        15:20:10   BIC                           1919.560
Sample:                    01-01-1964   HQIC                          1914.841
                         - 09-01-1972
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4545      0.114      3.999      0.000       0.232       0.677
ma.L1         -0.9666      0.056    -17.316      0.000      -1.076      -0.857
sigma2      5.226e+06   6.17e+05      8.473      0.000    4.02e+06    6.43e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.91   Jarque-Bera (JB):                 2.59
Prob(Q):                              0.34   Prob(JB):                         0.27
Heteroskedasticity (H):               3.40   Skew:                             0.05
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.77
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [40]:
```
#### forecasting sales
data['forecast']=model_fit.predict(start=90,end=103,dynamic=True)
data[['Sales','forecast']].plot(figsize=(12,8))
```

Out[40]: <Axes: xlabel='Month'>

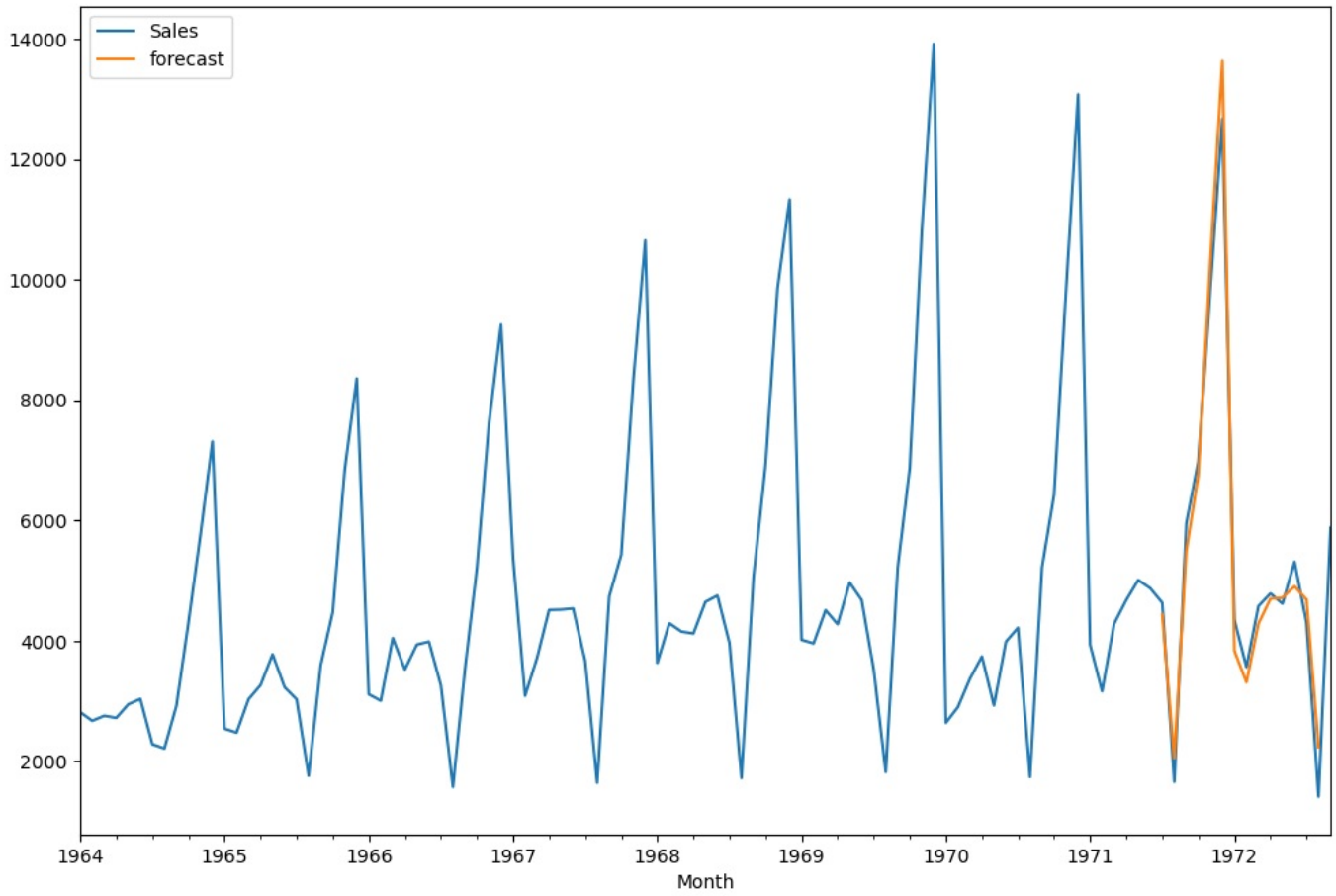


In [41]:
```
##### due to data having sasonality the arima model is not working it can not give proper result so we used SARI
import statsmodels.api as sm
model=sm.tsa.statespace.SARIMAX(data['Sales'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
results=model.fit()
```

```
C:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\anaconda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

In [44]:
```
#### after that forecsating sales
data['forecast']=results.predict(start=90,end=103,dynamic=True)
data[['Sales','forecast']].plot(figsize=(12,8))
```

Out[44]: <Axes: xlabel='Month'>



In [45]:
```
### to find out the next year sales
from pandas.tseries.offsets import DateOffset
future_dates=[data.index[-1]+ DateOffset(months=x)for x in range(0,24)]
```

In [46]:
```
future_sales = pd.DataFrame(index =future_dates[1:] ,columns=data.columns)
```

In [47]:
```
future_sales
```

Out[47]:

	Sales	Sales Frist_defference	Sales Based on Seasonality	forecast
1972-10-01	NaN	NaN	NaN	NaN
1972-11-01	NaN	NaN	NaN	NaN
1972-12-01	NaN	NaN	NaN	NaN
1973-01-01	NaN	NaN	NaN	NaN
1973-02-01	NaN	NaN	NaN	NaN
1973-03-01	NaN	NaN	NaN	NaN
1973-04-01	NaN	NaN	NaN	NaN
1973-05-01	NaN	NaN	NaN	NaN
1973-06-01	NaN	NaN	NaN	NaN
1973-07-01	NaN	NaN	NaN	NaN
1973-08-01	NaN	NaN	NaN	NaN
1973-09-01	NaN	NaN	NaN	NaN
1973-10-01	NaN	NaN	NaN	NaN
1973-11-01	NaN	NaN	NaN	NaN
1973-12-01	NaN	NaN	NaN	NaN
1974-01-01	NaN	NaN	NaN	NaN
1974-02-01	NaN	NaN	NaN	NaN
1974-03-01	NaN	NaN	NaN	NaN
1974-04-01	NaN	NaN	NaN	NaN
1974-05-01	NaN	NaN	NaN	NaN
1974-06-01	NaN	NaN	NaN	NaN
1974-07-01	NaN	NaN	NaN	NaN
1974-08-01	NaN	NaN	NaN	NaN

In [49]:
```
### concat the orignal sales and future sales
sales = pd.concat([data , future_sales])
```

```
C:\Users\shubham lokare\AppData\Local\Temp\ipykernel_20928\1689533271.py:2: FutureWarning: The behavior of DataF
rame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude
empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant ent
ries before the concat operation.
  sales = pd.concat([data , future_sales])
```

In [50]:
```
sales
```

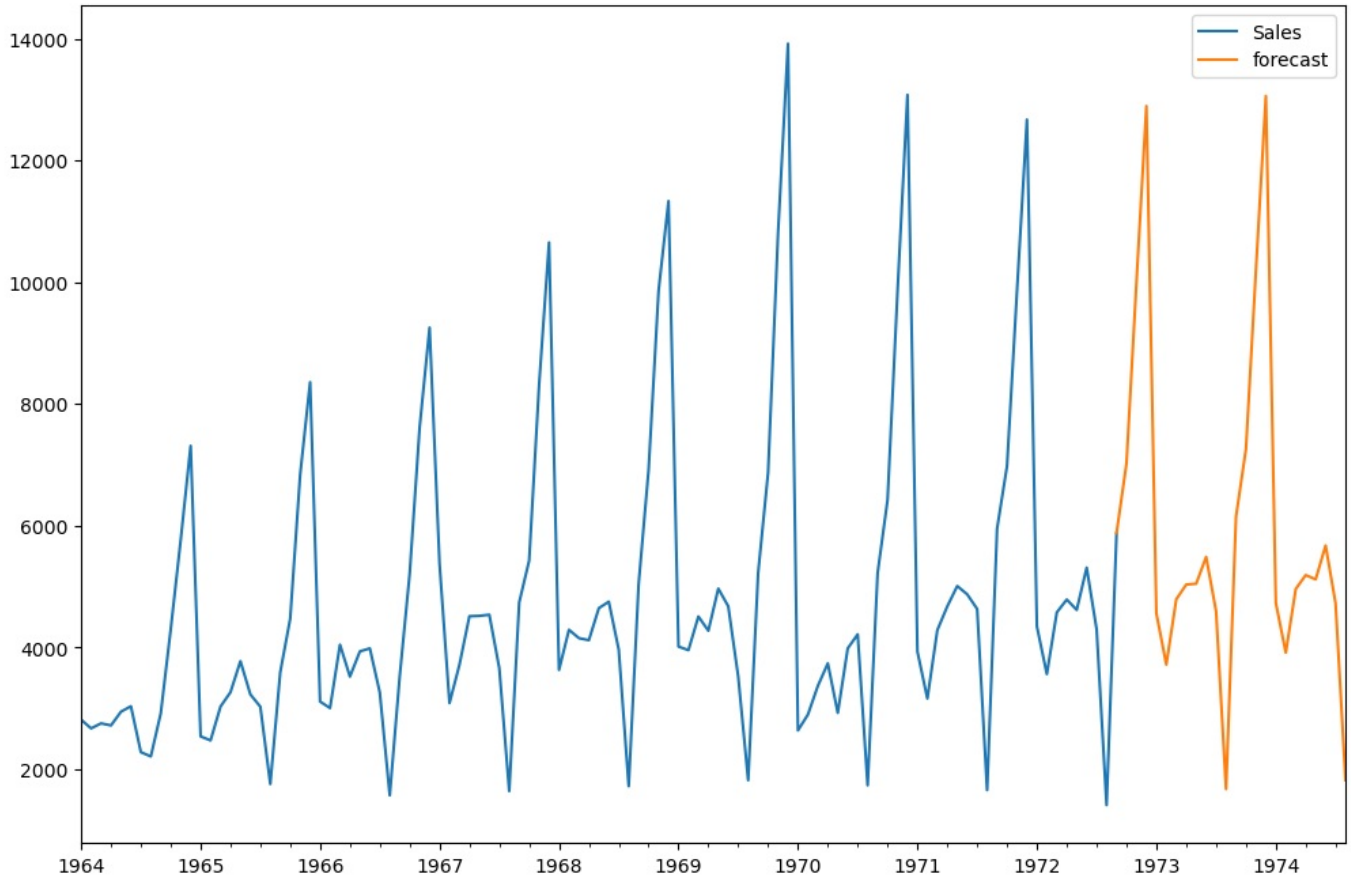
Out[50]:

	Sales	Sales Frist_defference	Sales Based on Seasonality	forecast
1964-01-01	2815.0	NaN	NaN	NaN
1964-02-01	2672.0	-143.0	NaN	NaN
1964-03-01	2755.0	83.0	NaN	NaN
1964-04-01	2721.0	-34.0	NaN	NaN
1964-05-01	2946.0	225.0	NaN	NaN
...	...	...	...	...
1974-04-01	NaN	NaN	NaN	NaN
1974-05-01	NaN	NaN	NaN	NaN
1974-06-01	NaN	NaN	NaN	NaN
1974-07-01	NaN	NaN	NaN	NaN
1974-08-01	NaN	NaN	NaN	NaN

128 rows × 4 columns

In [56]:
```
#### plot the graph of future sales
sales['forecast'] = results.predict(start = 104, end = 150, dynamic= True)
sales[['Sales', 'forecast']].plot(figsize=(12, 8))
```

Out[56]: <Axes: >



In [ ]: