ARIMA and Seasonal ARIMA

Autoregressive Integrated Moving Averages

The general process for ARIMA models is the following:

- Visualize the Time Series Data
- Make the time series data stationary
- Plot the Correlation and AutoCorrelation Charts
- Construct the ARIMA Model or Seasonal ARIMA based on the data
- Use the model to make predictions

Let's go through these steps!

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
```

```
Bad key "text.kerning_factor" on line 4 in
C:\Users\91920\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_cla
ssic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
 (https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templat
e)
or from the matplotlib source distribution
```

In [2]:

```
filename="E:\python\perrin-freres-monthly-champagne-.csv"

df=pd.read_csv(filename)
df.head()
```

Out[2]:

	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

In [3]:

```
df.tail()
```

Out[3]:

	Month	Perrin Freres monthly champagne sales millions ? 64-?72
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

In [4]:

```
## Cleaning up the data
df.columns=["Month","Sales"]
df.head()
```

Out[4]:

	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

In [5]:

```
## Drop last 2 rows
df.drop(106,axis=0,inplace=True)
```

In [6]:

```
df.tail()
```

Out[6]:

	Month	Sales
101	1972-06	5312.0
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0
105	NaN	NaN

In [7]:

```
df.drop(105,axis=0,inplace=True)
```

In [8]:

```
df.tail()
```

Out[8]:

	Month	Sales
100	1972-05	4618.0
101	1972-06	5312.0
102	1972-07	4298.0
103	1972-08	1413.0
104	1972-09	5877.0

In [9]:

```
# Convert Month into Datetime
df['Month']=pd.to_datetime(df['Month'])
```

In [10]:

```
df.head()
```

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

In [11]:

```
df.set_index('Month',inplace=True)
```

In [12]:

```
df.head()
```

Out[12]:

Month	Sales
1964-01-01	2815.0
1964-02-01	2672.0
1964-03-01	2755.0
1964-04-01	2721.0
1964-05-01	2946.0

In [13]:

```
df.describe()
```

Out[13]:

	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

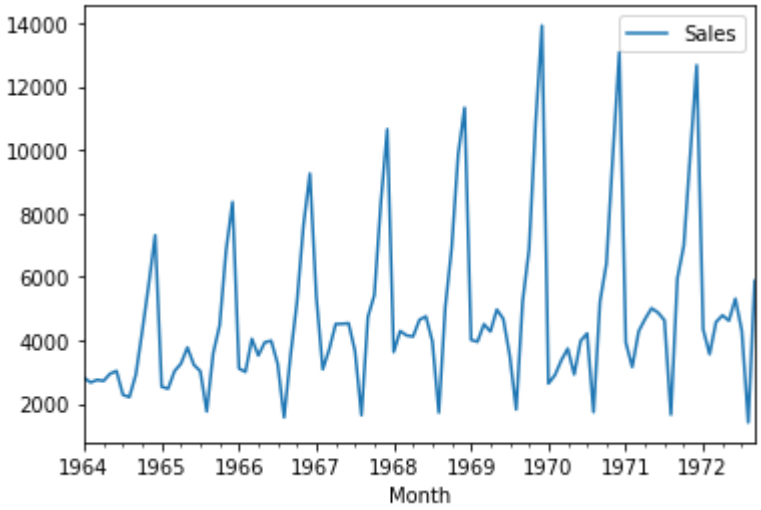
Step 2: Visualize the Data

In [14]:

```
df.plot()
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee79cfa488>
```



In [15]:

```
# Rolling Statistics

rolmean = df.rolling(window=12).mean()
rolstd = df.rolling(window=12).std()
print(rolmean,rolstd)
```

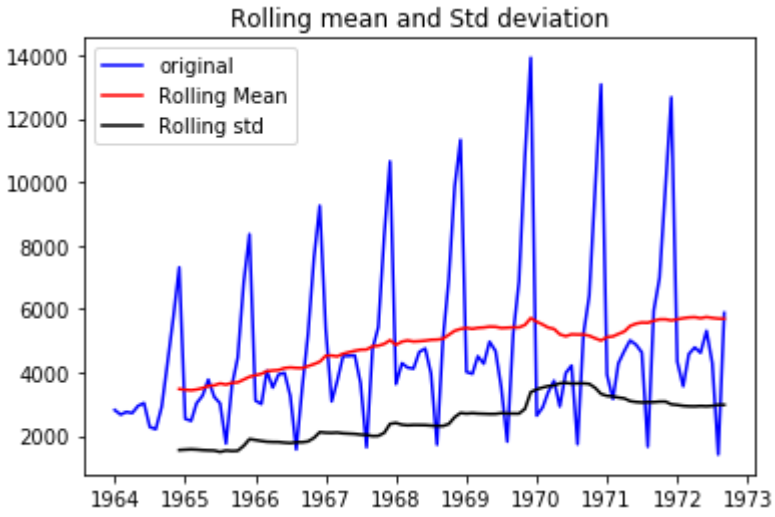
```
                 Sales
Month
1964-01-01         NaN
1964-02-01         NaN
1964-03-01         NaN
1964-04-01         NaN
1964-05-01         NaN
...                ...
1972-05-01  5709.500000
1972-06-01  5746.000000
1972-07-01  5718.083333
1972-08-01  5697.583333
1972-09-01  5691.416667

[105 rows x 1 columns]                  Sales
Month
1964-01-01         NaN
1964-02-01         NaN
1964-03-01         NaN
1964-04-01         NaN
1964-05-01         NaN
...                ...
1972-05-01  2943.699327
1972-06-01  2935.100773
1972-07-01  2948.213094
1972-08-01  2979.690449
1972-09-01  2979.194841

[105 rows x 1 columns]
```

In [16]:

```
orig = plt.plot(df,color='blue',label='original')
mean = plt.plot(rolmean,color='red',label='Rolling Mean')
std = plt.plot(rolstd,color='black',label='Rolling std')
plt.legend()
plt.title('Rolling mean and Std deviation')
plt.show()
```



In [17]:

```
### Testing For Stationarity using Dickey-fuller test

from statsmodels.tsa.stattools import adfuller
```

In [18]:

```
test_result=adfuller(df['Sales'])
test_result  # 'ADF Test Statistic','p-value','#Lags Used','Number of Observations Used'
```

Out[18]:

```
(-1.8335930563276297,
 0.3639157716602417,
 11,
 93,
 {'1%': -3.502704609582561,
  '5%': -2.8931578098779522,
  '10%': -2.583636712914788},
 1478.4633060594724)
```

In [19]:

```
#Ho: It is not stationary
#H1: It is stationary

def adfuller_test(sales):
    result=adfuller(sales)
    print(result)
    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.
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicati
```

In [20]:

```
adfuller_test(df['Sales'])
```

```
(-1.8335930563276297, 0.3639157716602417, 11, 93, {'1%': -3.502704609582561,
'5%': -2.8931578098779522, '10%': -2.583636712914788}, 1478.4633060594724)
ADF Test Statistic : -1.8335930563276297
p-value : 0.3639157716602417
#Lags Used : 11
Number of Observations Used : 93
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

Differencing

In [21]:

```
df['Sales First Difference'] = df['Sales'] - df['Sales'].shift(1)
```

In [22]:

```
df['Sales'].shift(1)
```

Out[22]:

```
Month
1964-01-01       NaN
1964-02-01    2815.0
1964-03-01    2672.0
1964-04-01    2755.0
1964-05-01    2721.0
               ...
1972-05-01    4788.0
1972-06-01    4618.0
1972-07-01    5312.0
1972-08-01    4298.0
1972-09-01    1413.0
Name: Sales, Length: 105, dtype: float64
```

In [23]:

```
df['Seasonal First Difference']=df['Sales']-df['Sales'].shift(12)
```

In [24]:

```
df.head(14)
```

Out[24]:

Month	Sales	Sales First Difference	Seasonal First Difference
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
1964-11-01	5764.0	1463.0	NaN
1964-12-01	7312.0	1548.0	NaN
1965-01-01	2541.0	-4771.0	-274.0
1965-02-01	2475.0	-66.0	-197.0

In [25]:

```
## Again test dickey fuller test
adfuller_test(df['Seasonal First Difference'].dropna())
```

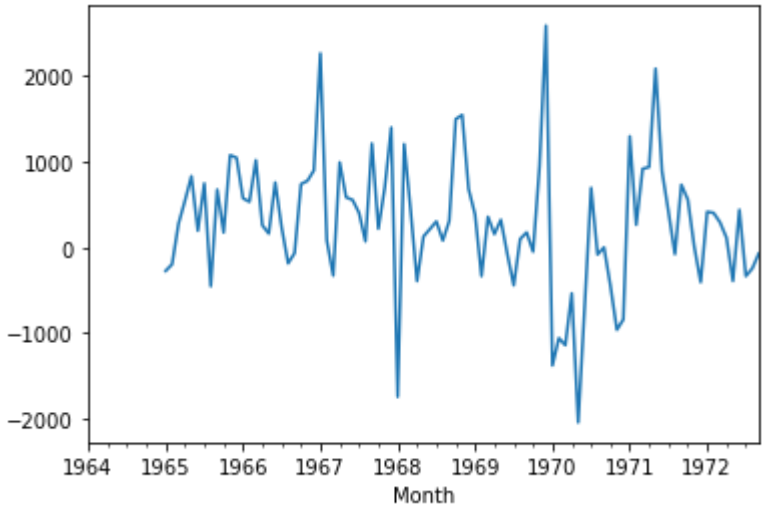
```
(-7.626619157213163, 2.060579696813685e-11, 0, 92, {'1%': -3.50351457965192
7, '5%': -2.893507960466837, '10%': -2.583823615311909}, 1294.7753384560438)
ADF Test Statistic : -7.626619157213163
p-value : 2.060579696813685e-11
#Lags Used : 0
Number of Observations Used : 92
strong evidence against the null hypothesis(Ho), reject the null hypothesis.
Data has no unit root and is stationary
```

In [26]:

```
df['Seasonal First Difference'].plot()
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee7d358588>
```



In [27]:

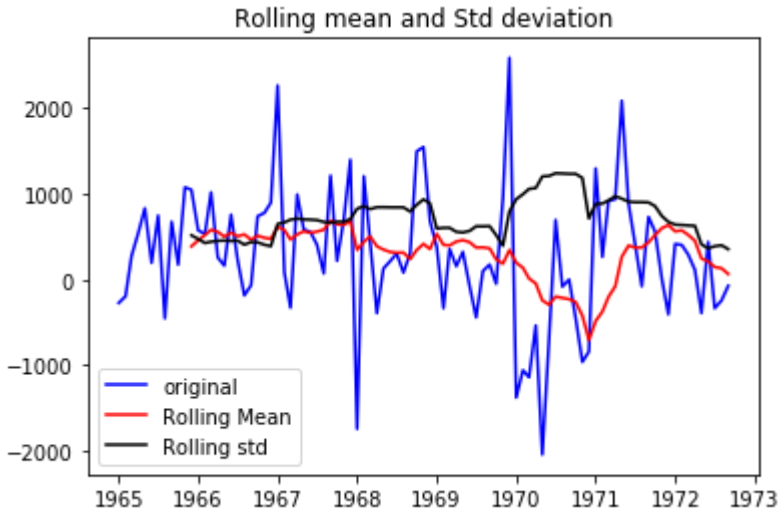
```
# Rolling Statistics

rolmean = df['Seasonal First Difference'].rolling(window=12).mean()
rolstd = df['Seasonal First Difference'].rolling(window=12).std()
print(rolmean,rolstd)
```

```
Month
1964-01-01           NaN
1964-02-01           NaN
1964-03-01           NaN
1964-04-01           NaN
1964-05-01           NaN
                 ...
1972-05-01    245.166667
1972-06-01    207.666667
1972-07-01    145.083333
1972-08-01    131.166667
1972-09-01     64.166667
Name: Seasonal First Difference, Length: 105, dtype: float64 Month
1964-01-01           NaN
1964-02-01           NaN
1964-03-01           NaN
1964-04-01           NaN
1964-05-01           NaN
                 ...
1972-05-01    408.824887
1972-06-01    362.515287
1972-07-01    387.260180
1972-08-01    398.871552
1972-09-01    354.158114
Name: Seasonal First Difference, Length: 105, dtype: float64
```

In [28]:

```
orig = plt.plot(df['Seasonal First Difference'],color='blue',label='original')
mean = plt.plot(rolmean,color='red',label='Rolling Mean')
std = plt.plot(rolstd,color='black',label='Rolling std')
plt.legend()
plt.title('Rolling mean and Std deviation')
plt.show()
```



Auto Regressive Model

Final Thoughts on Autocorrelation and Partial Autocorrelation

- Identification of an AR model is often best done with the PACF.
  - For an AR model, the theoretical PACF “shuts off” past the order of the model. The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point. Put another way, the number of non-zero partial autocorrelations gives the order of the AR model. By the “order of the model” we mean the most extreme lag of x that is used as a predictor.

- Identification of an MA model is often best done with the ACF rather than the PACF.
  - For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model.

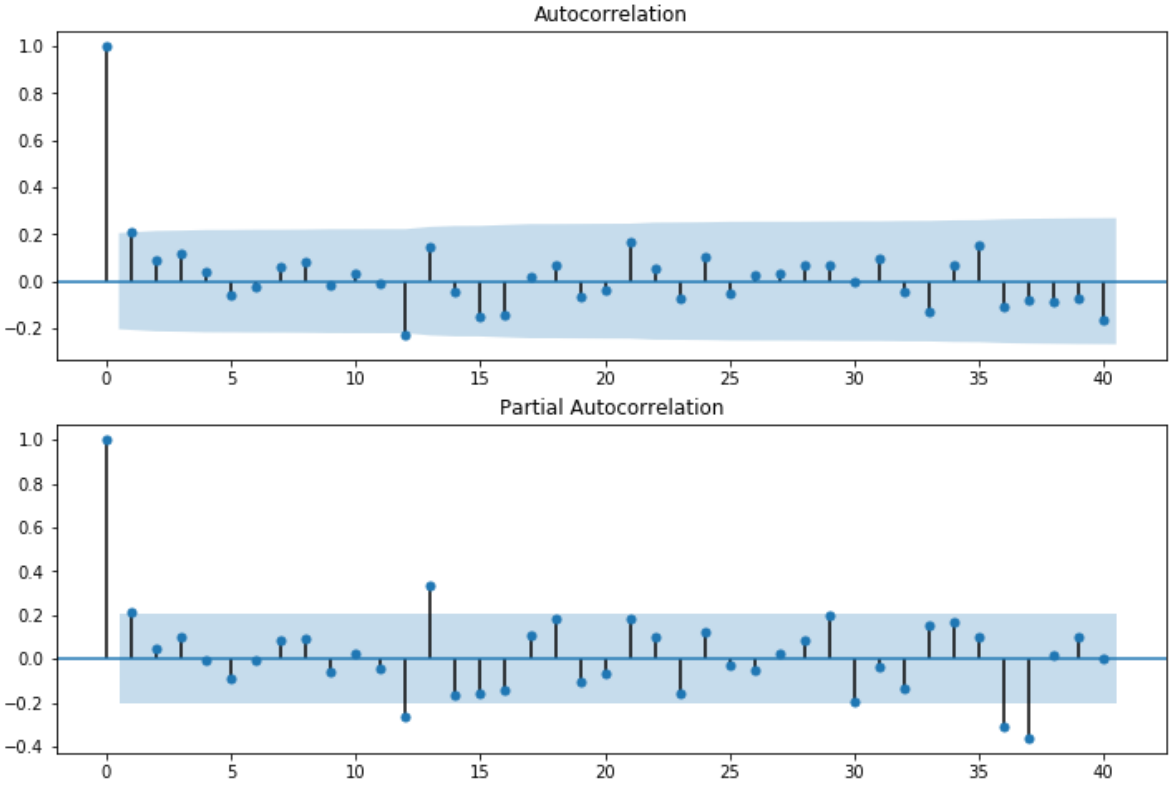
    p,d,q p AR model lags d differencing q MA lags

In [31]:

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm
```

In [32]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['Seasonal First Difference'].iloc[13:],lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['Seasonal First Difference'].iloc[13:],lags=40,ax=ax2)
```



In [52]:

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

In [53]:

```
model=ARIMA(df['Sales'],order=(1,1,1))
model_fit=model.fit()
```

```
C:\Users\91920\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:162: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  % freq, ValueWarning)
C:\Users\91920\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:162: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  % freq, ValueWarning)
```

In [54]:

```
model_fit.summary()
```

Out[54]:

ARIMA Model Results

Dep. Variable:	D.Sales	**No. Observations:**	104
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-951.126
Method:	css-mle	**S.D. of innovations**	2227.262
Date:	Tue, 04 Jan 2022	**AIC**	1910.251
Time:	02:33:53	**BIC**	1920.829
Sample:	02-01-1964	**HQIC**	1914.536
	- 09-01-1972		

	coef	std err	z	P>|z|	[0.025	0.975]
const	22.7829	12.405	1.837	0.066	-1.531	47.097
ar.L1.D.Sales	0.4343	0.089	4.866	0.000	0.259	0.609
ma.L1.D.Sales	-1.0000	0.026	-38.503	0.000	-1.051	-0.949

Roots

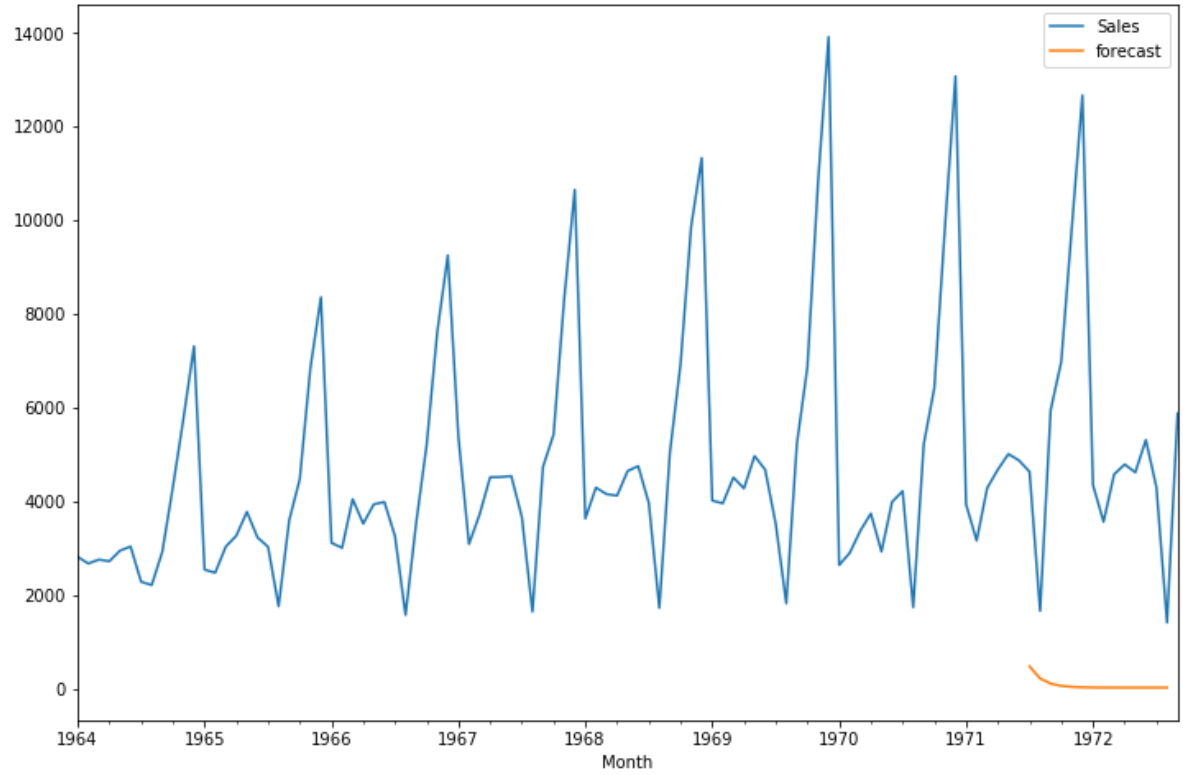
	Real	Imaginary	Modulus	Frequency
AR.1	2.3023	+0.0000j	2.3023	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000

In [55]:

```
df['forecast']=model_fit.predict(start=90,end=103,dynamic=True)
df[['Sales','forecast']].plot(figsize=(12,8))
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee010f3fc8>
```



In [60]:

```
import statsmodels.api as sm
```

In [61]:

```
model=sm.tsa.statespace.SARIMAX(df['Sales'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
results=model.fit()
```

```
C:\Users\91920\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:162: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  % freq, ValueWarning)
C:\Users\91920\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:162: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  % freq, ValueWarning)
```

```
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.

In this case, our model diagnostics suggests that the model residuals are normally distributed based on the following:

In the top right plot, we see that the red KDE line follows closely with the N(0,1) line (where N(0,1)) is the standard notation for a normal distribution with mean 0 and standard deviation of 1). This is a good indication that the residuals are normally distributed.
The qq-plot on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this is a strong indication that the residuals are normally distributed.
The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. This is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.
Those observations lead us to conclude that our model produces a satisfactory fit that could help us understand our time series data and forecast future values.

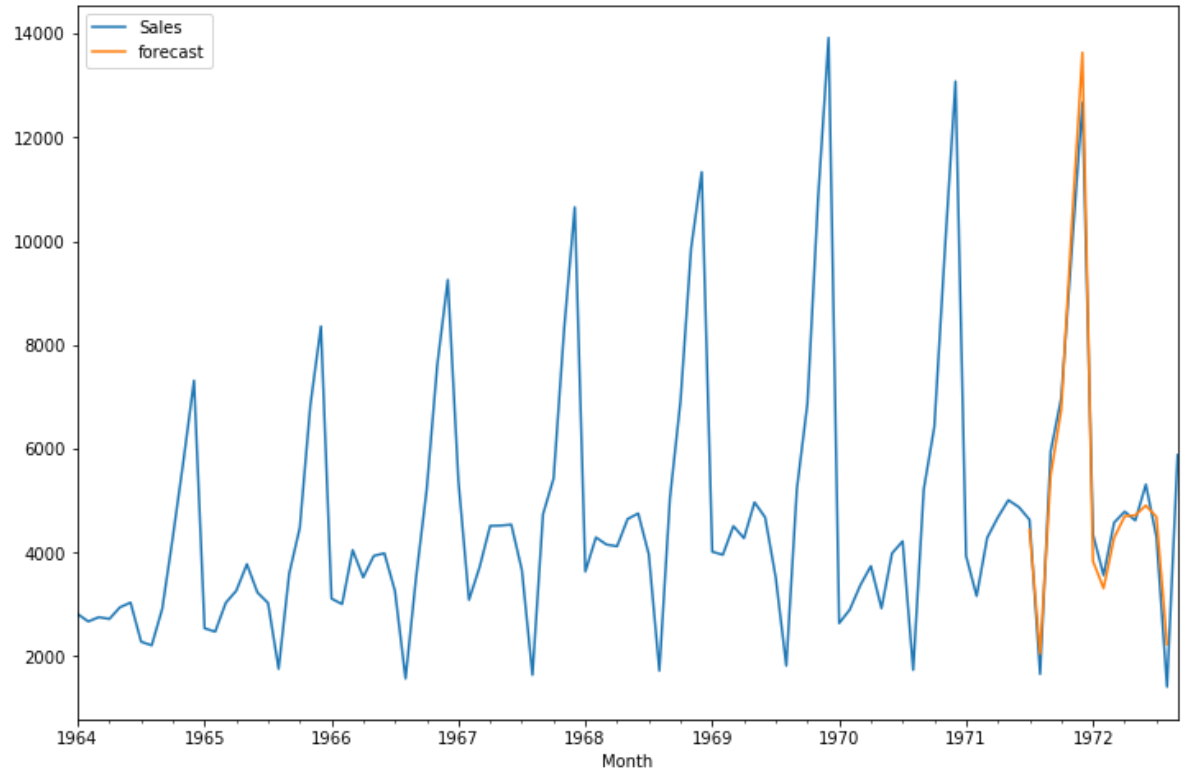
Although we have a satisfactory fit, some parameters of our seasonal ARIMA model could be changed to improve our model fit.

In [39]:

```
df['forecast']=results.predict(start=90,end=103,dynamic=True)
df[['Sales','forecast']].plot(figsize=(12,8))
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee7e417248>
```



In [40]:

```
from pandas.tseries.offsets import DateOffset
future_dates=[df.index[-1]+ DateOffset(months=x)for x in range(0,24)]
future_dates
```

Out[40]:

```
[Timestamp('1972-09-01 00:00:00'),
 Timestamp('1972-10-01 00:00:00'),
 Timestamp('1972-11-01 00:00:00'),
 Timestamp('1972-12-01 00:00:00'),
 Timestamp('1973-01-01 00:00:00'),
 Timestamp('1973-02-01 00:00:00'),
 Timestamp('1973-03-01 00:00:00'),
 Timestamp('1973-04-01 00:00:00'),
 Timestamp('1973-05-01 00:00:00'),
 Timestamp('1973-06-01 00:00:00'),
 Timestamp('1973-07-01 00:00:00'),
 Timestamp('1973-08-01 00:00:00'),
 Timestamp('1973-09-01 00:00:00'),
 Timestamp('1973-10-01 00:00:00'),
 Timestamp('1973-11-01 00:00:00'),
 Timestamp('1973-12-01 00:00:00'),
 Timestamp('1974-01-01 00:00:00'),
 Timestamp('1974-02-01 00:00:00'),
 Timestamp('1974-03-01 00:00:00'),
 Timestamp('1974-04-01 00:00:00'),
 Timestamp('1974-05-01 00:00:00'),
 Timestamp('1974-06-01 00:00:00'),
 Timestamp('1974-07-01 00:00:00'),
 Timestamp('1974-08-01 00:00:00')]
```

In [41]:

```
df.columns
```

Out[41]:

```
Index(['Sales', 'Sales First Difference', 'Seasonal First Difference',
       'forecast'],
      dtype='object')
```

In [42]:

```
future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df.columns,dtype='int')
```

In [43]:

```
future_datest_df.tail()
```

Out[43]:

	Sales	Sales First Difference	Seasonal First Difference	forecast
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 [44]:

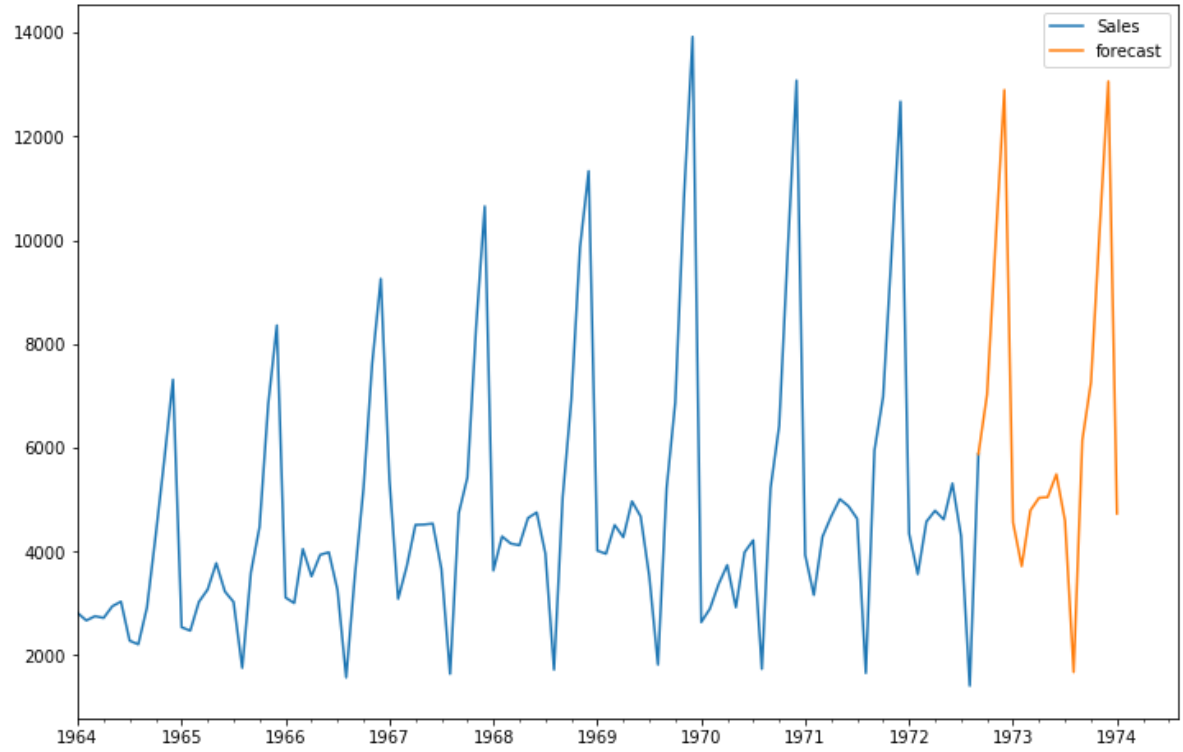
```
future_df=pd.concat([df,future_datest_df])
```

In [45]:

```
future_df['forecast'] = results.predict(start = 104, end = 120, dynamic= True)
future_df[['Sales', 'forecast']].plot(figsize=(12, 8))
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee7e6bc648>
```



In [ ]:

```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [46]:

```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in c:\users\91920\anaconda3\lib\site
-packages (1.8.4)
Requirement already satisfied: Cython!=0.29.18,>=0.29 in c:\users\91920\anac
onda3\lib\site-packages (from pmdarima) (0.29.15)
Requirement already satisfied: pandas>=0.19 in c:\users\91920\anaconda3\lib
\site-packages (from pmdarima) (1.0.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\91920\anaconda3\lib
\site-packages (from pmdarima) (1.4.1)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\users\91920
\anaconda3\lib\site-packages (from pmdarima) (52.0.0.post20210125)
Requirement already satisfied: urllib3 in c:\users\91920\anaconda3\lib\site-
packages (from pmdarima) (1.26.4)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\91920\anaconda
3\lib\site-packages (from pmdarima) (0.24.2)
Requirement already satisfied: joblib>=0.11 in c:\users\91920\anaconda3\lib
\site-packages (from pmdarima) (0.14.1)
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in c:\users\91920
\anaconda3\lib\site-packages (from pmdarima) (0.11.0)
Requirement already satisfied: numpy>=1.19.3 in c:\users\91920\anaconda3\lib
\site-packages (from pmdarima) (1.20.3)
Requirement already satisfied: pytz>=2017.2 in c:\users\91920\anaconda3\lib
\site-packages (from pandas>=0.19->pmdarima) (2021.1)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\91920\anac
onda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\91920\anaconda3\lib\site
-packages (from python-dateutil>=2.6.1->pandas>=0.19->pmdarima) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\91920\anacon
da3\lib\site-packages (from scikit-learn>=0.22->pmdarima) (2.1.0)
Requirement already satisfied: patsy>=0.5 in c:\users\91920\anaconda3\lib\si
te-packages (from statsmodels!=0.12.0,>=0.11->pmdarima) (0.5.1)

WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -umpy (c:\users\91920\anaconda3\lib\s
ite-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\91920\anaconda3
\lib\site-packages)
```

In [47]:

```
import pmdarima as pm
def arimamodel(timeseries):
    automodel = pm.auto_arima(timeseries,
                              start_p=0,
                              start_q=0,
                              max_p=5,
                              max_q=5,
                              test="adf",
                              seasonal=True,
                              trace=True)
    return automodel
```

In [49]:

```
arimamodel(df['Sales'])
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1948.469, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=1925.227, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1918.917, Time=0.06 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=2104.630, Time=0.00 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=1922.735, Time=0.03 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=1920.359, Time=0.04 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=1922.465, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=2012.608, Time=0.04 sec

Best model:  ARIMA(0,0,1)(0,0,0)[0] intercept
Total fit time: 0.285 seconds
```

Out[49]:

```
ARIMA(order=(0, 0, 1), scoring_args={}, suppress_warnings=True)
```

In [ ]: