```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv("/content/time_data.csv")
data.head()
```

	Month	Sales
0	1964-01	2815
1	1964-02	2672
2	1964-03	2755
3	1964-04	2721
4	1964-05	2946

Create index based on month

```
data.index = pd.to_datetime(data['Month'])
data.drop(columns='Month',inplace=True)
data.head()
```

	Sales
Month	
1964-01-01	2815
1964-02-01	2672
1964-03-01	2755
1964-04-01	2721
1964-05-01	2946

Preprocess data

```
data.isna().sum()
```

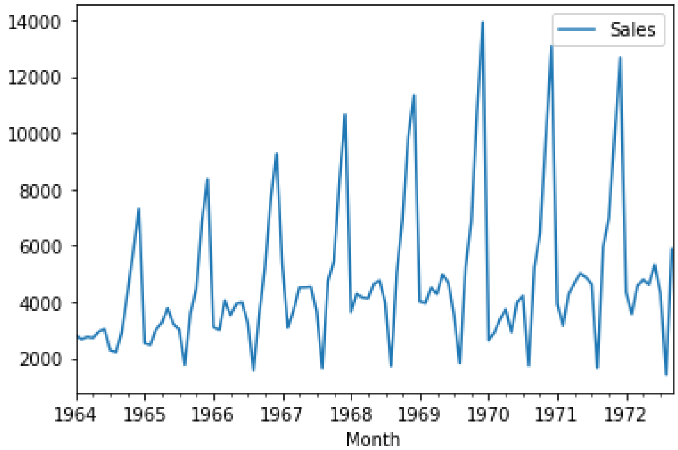
```
Sales    0
dtype: int64
```

```
data.describe()
```

	Sales
count	105.000000
mean	4761.152381

```
data.plot()
```

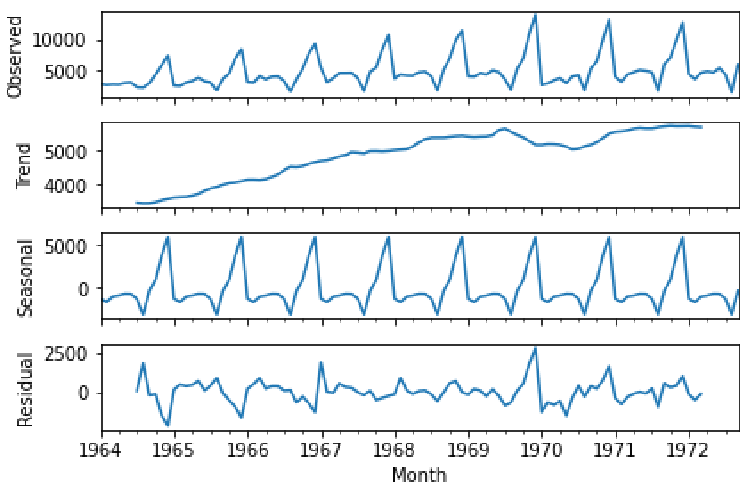
<matplotlib.axes._subplots.AxesSubplot at 0x7f455d747ed0>



Decompose time series

```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_data = seasonal_decompose(data, model="additive")
decompose_data.plot();
```

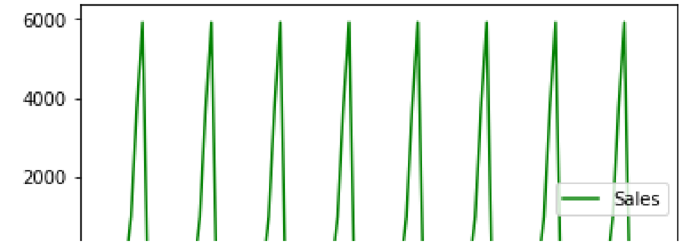
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functio
  import pandas.util.testing as tm



Visualize the seasons

```
seasonality=decompose_data.seasonal
seasonality.plot(color='green')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f4546270910>



Performing the adfuller test


```
from statsmodels.tsa.stattools import adfuller
dftest = adfuller(data.Sales, autolag = 'AIC')
print("1. ADF : ",dftest[0])
print("2. P-Value : ", dftest[1])
print("3. Num Of Lags : ", dftest[2])
print("4. Num Of Observations Used For ADF Regression and Critical Values Calculation :", dftest[3])
print("5. Critical Values :")
for key, val in dftest[4].items():
    print("\t",key, ": ", val)
```

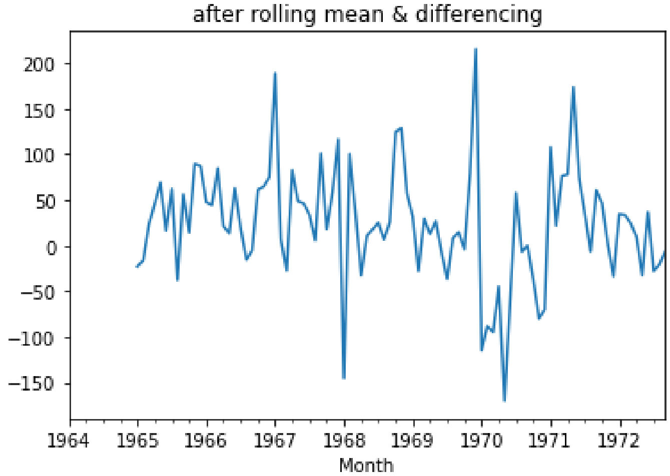
```
1. ADF :  -1.8335930563276228
2. P-Value :  0.363915771660245
3. Num Of Lags :  11
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 93
5. Critical Values :
	1% :  -3.502704609582561
	5% :  -2.8931578098779522
	10% :  -2.583636712914788
```

The p-value is higher than p-0.05. Therefore, we reject the hypothesis that the time series is stationary at a 95% confidence level. Hence the time series is non-stationary. We can make the time series stationary with differencing methods. In this case, we are going ahead with the rolling mean differencing methods.

When there is a strong seasonal effect, we use the rolling mean differencing.

```
rolling_mean = data.rolling(window = 12).mean()
data['rolling_mean_diff'] = rolling_mean - rolling_mean.shift()
ax1 = plt.subplot()
data['rolling_mean_diff'].plot(title='after rolling mean & differencing');
ax2 = plt.subplot()
data.plot(title='original');
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: MatplotlibDeprecationWarning: Adding an axes using the same arguments
  """
```




```
dftest = adfuller(data['rolling_mean_diff'].dropna(), autolag = 'AIC')
print("1. ADF : ",dftest[0])
print("2. P-Value : ", dftest[1])
print("3. Num Of Lags : ", dftest[2])
print("4. Num Of Observations Used For ADF Regression and Critical Values Calculation :", dftest[3])
print("5. Critical Values :")
for key, val in dftest[4].items():
  print("\t",key, ": ", val)
```

```
1. ADF :  -7.626619157213174
2. P-Value :  2.060579696813558e-11
3. Num Of Lags :  0
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 92
5. Critical Values :
	 1% :  -3.503514579651927
	 5% :  -2.893507960466837
	 10% :  -2.583823615311909
```

We can see that the p-value is near about zero and very less than 0.05; now, our time series is stationary. Rule of thumb: low p-value (p-value < alpha) implies stationarity.

ARIMA model

```
from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(data['Sales'],order=(1,1,1))
history=model.fit()
history.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No frequency information was provided, sc
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No frequency information was provided, sc
  % freq, ValueWarning)
```

ARIMA Model Results

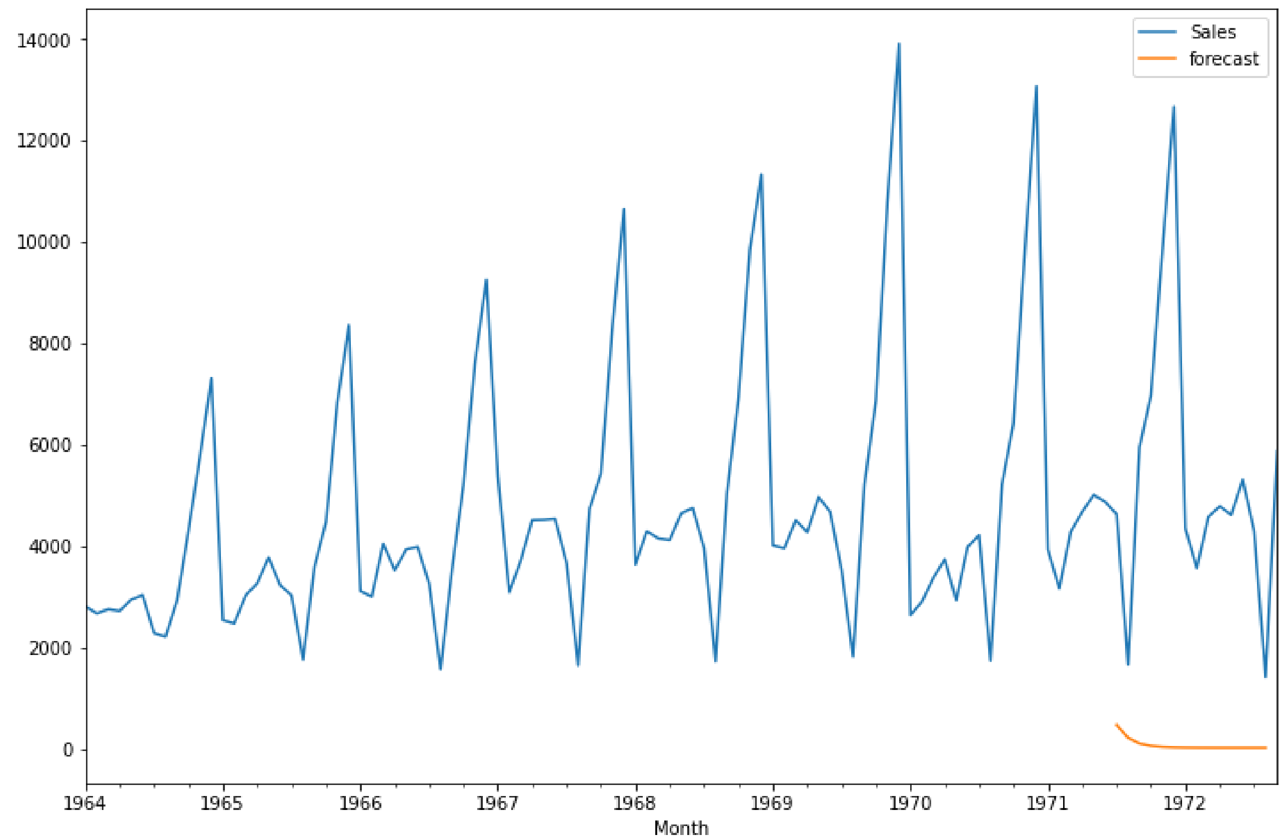
Dep. Variable:	D.Sales	**No. Observations:**	104
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-951.126

Forecasting with ARIMA

Time:	11:00:23	**BIC**	1920.829

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

<matplotlib.axes._subplots.AxesSubplot at 0x7f454289a650>



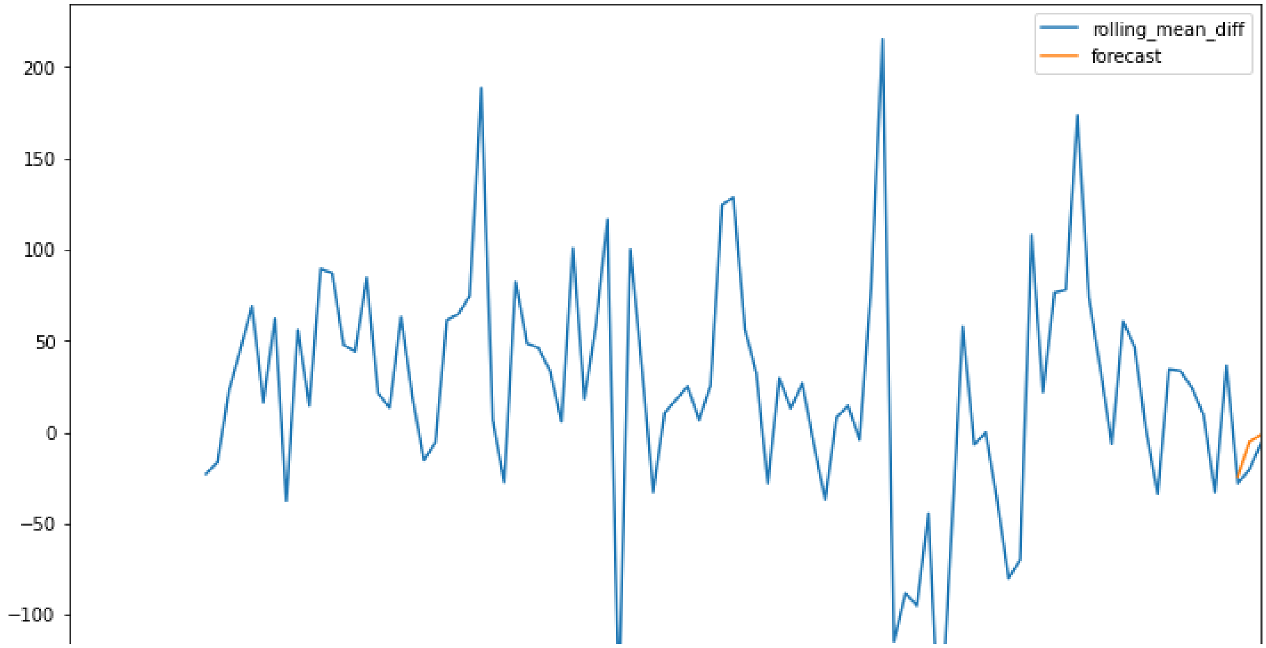
We apply the model with the data after differencing the time series.

```
model=ARIMA(data['rolling_mean_diff'].dropna(),order=(1,1,1))
model_fit=model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No frequency information was provided, sc
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No frequency information was provided, sc
  % freq, ValueWarning)
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4542797190>
```



SARIMAX(Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors) is an updated version of the ARIMA model. ARIMA includes an autoregressive integrated moving average, while SARIMAX includes seasonal effects and eXogenous factors with the autoregressive and moving average component in the model. Therefore, we can say SARIMAX is a seasonal equivalent model like SARIMA and Auto ARIMA.

```
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()
```

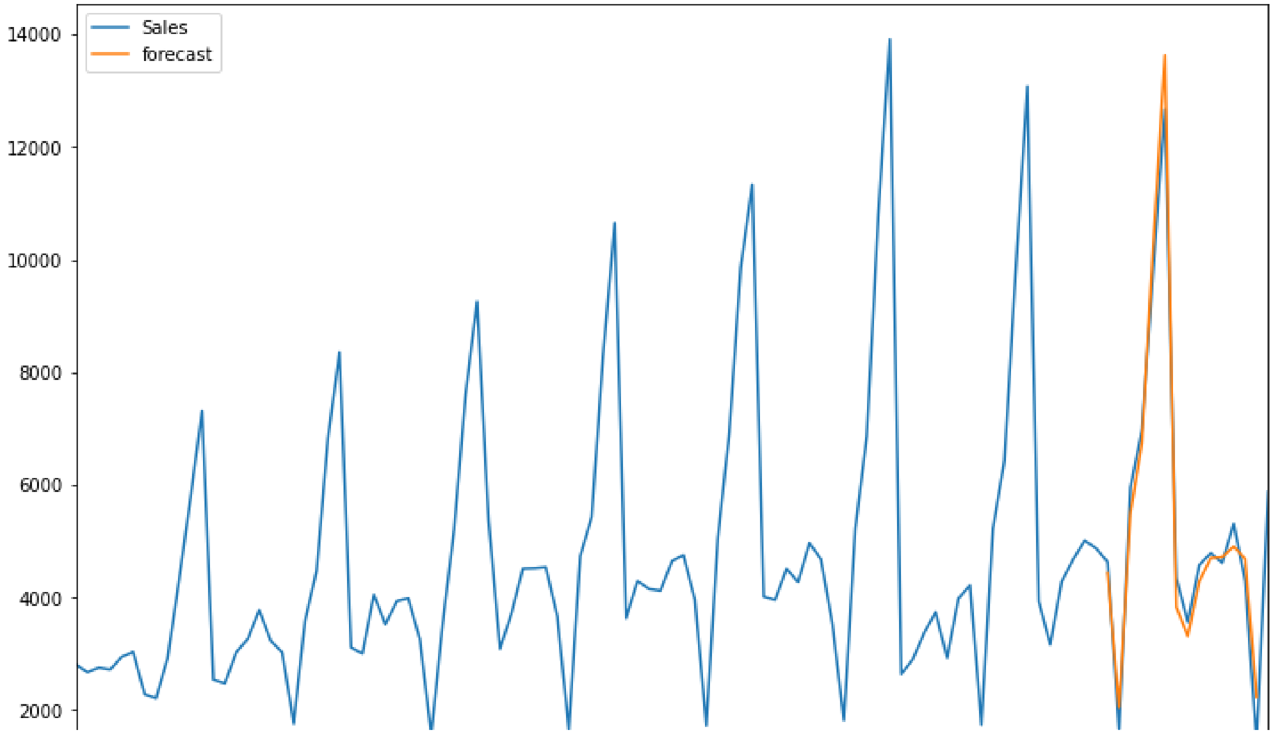
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No frequency information was provided, sc
  % freq, ValueWarning)
```

$$\phi_p(L)\tilde{\phi}_P(L^s)\Delta^d\Delta_s^D y_t = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\epsilon_t$$

- $\phi_p(L)$ is the non-seasonal autoregressive lag polynomial
- $\tilde{\phi}_P(L^s)$ is the seasonal autoregressive lag polynomial
- $\Delta^d\Delta_s^D y_t$ is the time series, differenced d times, and seasonally differenced D times.
- $A(t)$ is the trend polynomial (including the intercept)
- $\theta_q(L)$ is the non-seasonal moving average lag polynomial
- $\tilde{\theta}_Q(L^s)$ is the seasonal moving average lag polynomial

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f454e50f650>
```



Making a NAN value future dataset.

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

```
pred_date=pd.DataFrame(index=pred_date[1:],columns=data.columns)
pred_date
```

	Sales	rolling_mean_diff	forecast
1972-10-01	NaN	NaN	NaN
1972-11-01	NaN	NaN	NaN
1972-12-01	NaN	NaN	NaN
1973-01-01	NaN	NaN	NaN
1973-02-01	NaN	NaN	NaN
1973-03-01	NaN	NaN	NaN
1973-08-01	NaN	NaN	NaN

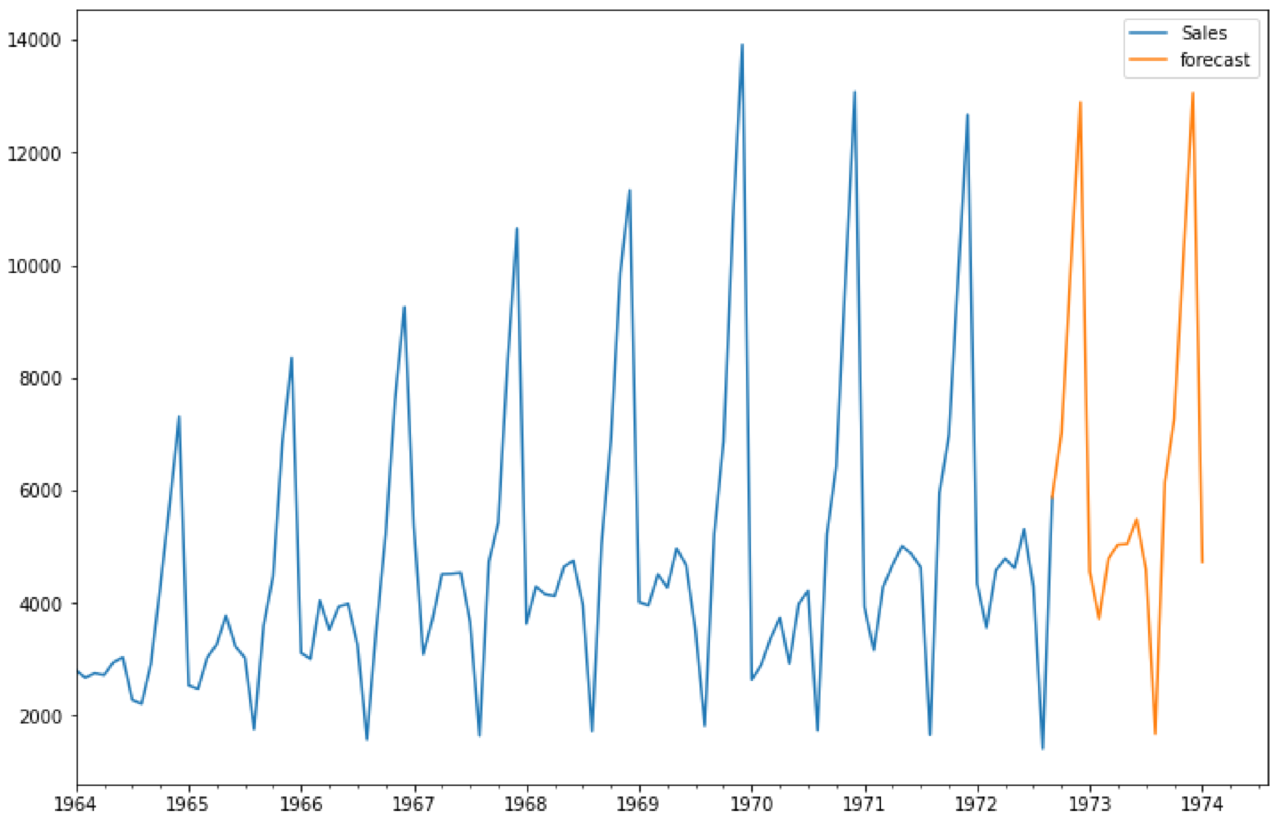
To make forecasted values, we need to concate this blank data with our alcohol sales data.

```
data=pd.concat([data,pred_date])
```

Making the prediction.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4534487c10>
```



0s completed at 7:10 AM