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

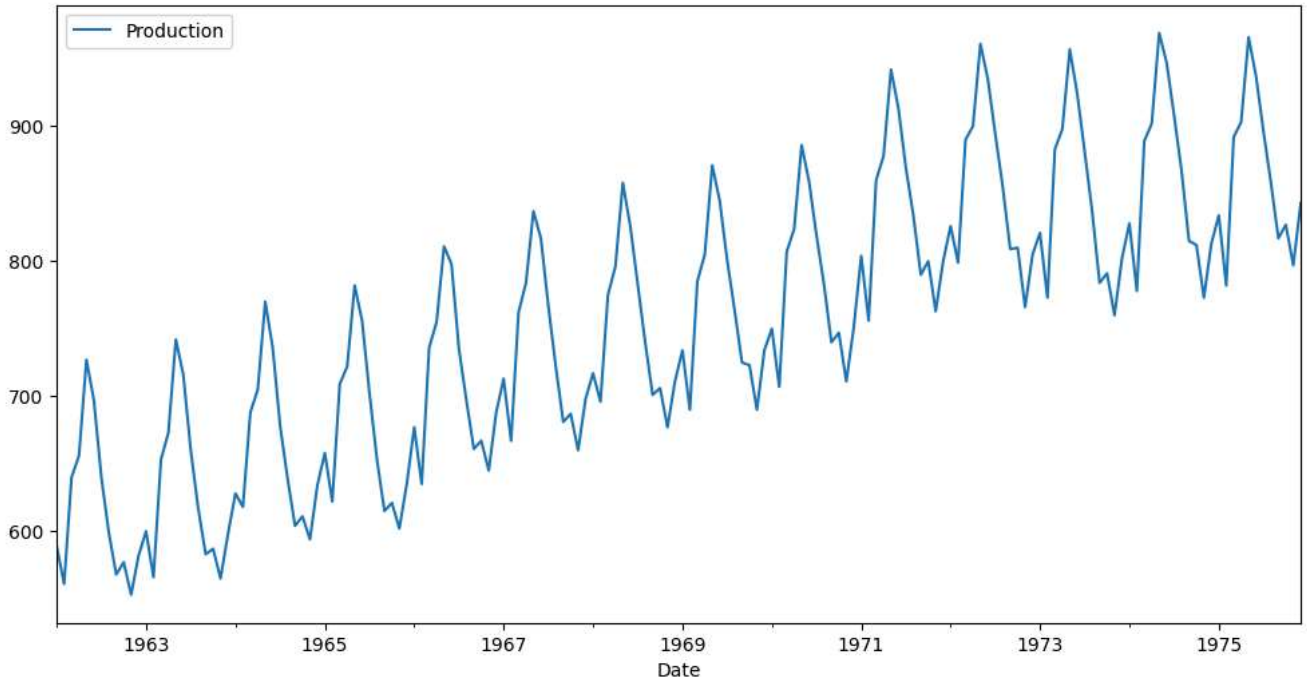
```
df=pd.read_csv('/content/monthly_milk_production.csv',index_col='Date',parse_dates=True)
df.index.freq='MS'
```

```
df.head()
```

Date	Production
1962-01-01	589
1962-02-01	561
1962-03-01	640
1962-04-01	656
1962-05-01	727

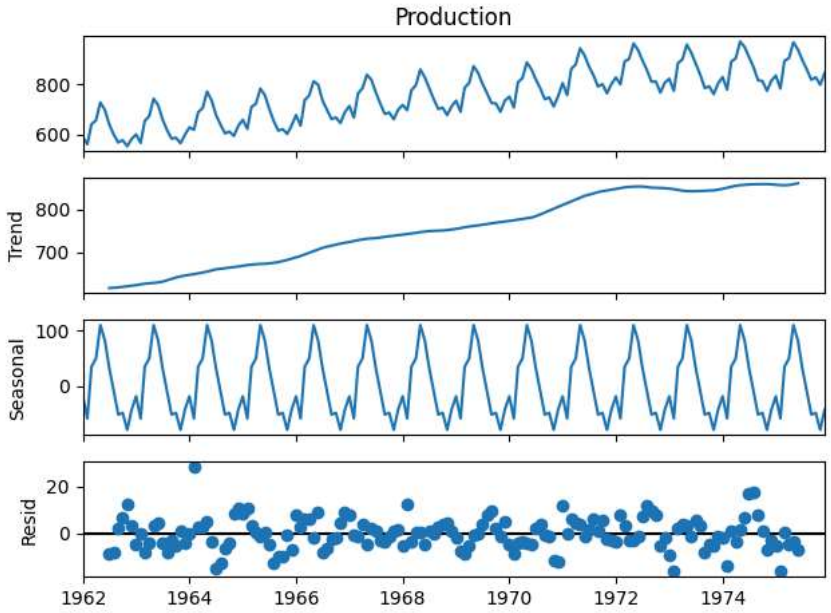
```
df.plot(figsize=(12,6))
```

```
<Axes: xlabel='Date'>
```




```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
results =seasonal_decompose(df['Production'])
results.plot();
```




```
len(df)
```

```
168
```

```
train = df.iloc[:156]
test = df.iloc[156:]
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
df.head(),df.tail()
```

```
(            Production
      Date
      1962-01-01         589
      1962-02-01         561
      1962-03-01         640
      1962-04-01         656
      1962-05-01         727,
                  Production
      Date
      1975-08-01         858
      1975-09-01         817
      1975-10-01         827
      1975-11-01         797
      1975-12-01         843)
```

```
scaler.fit(train)
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

```
scaled_train[:10]
```

```
array([[0.08653846],
            [0.01923077],
            [0.20913462],
            [0.24759615],
            [0.41826923],
            [0.34615385],
            [0.20913462],
            [0.11057692],
            [0.03605769],
            [0.05769231]])
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
n_input=3
n_features=1
generator = TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=1)
```

```
x,y=generator[0]
print(f'Given the Array: \n{x.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.08653846 0.01923077 0.20913462]
Predict this y:
 [[0.24759615]]
```

```
x.shape
```

```
(1, 3, 1)
```

```
n_input=12
generator = TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=1)
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
Model = Sequential()
Model.add(LSTM(100,activation='relu',input_shape=(n_input,n_features)))
Model.add(Dense(1))
Model.compile(optimizer='adam',loss='mse')
```

```
Model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               40800

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 40901 (159.77 KB)
Trainable params: 40901 (159.77 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

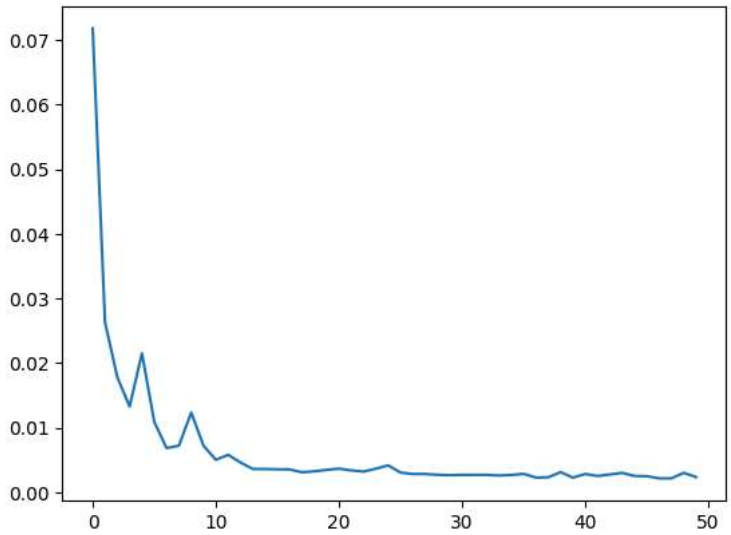
```
Model.fit(generator,epochs=50)
```

```
Epoch 1/50
144/144 [==============================] - 3s 8ms/step - loss: 0.0718
Epoch 2/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0263
Epoch 3/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0178
Epoch 4/50
144/144 [==============================] - 2s 11ms/step - loss: 0.0133
Epoch 5/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0215
Epoch 6/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0109
Epoch 7/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0068
Epoch 8/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0072
Epoch 9/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0123
Epoch 10/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0072
Epoch 11/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0050
Epoch 12/50
144/144 [==============================] - 1s 10ms/step - loss: 0.0058
Epoch 13/50
144/144 [==============================] - 2s 10ms/step - loss: 0.0046
Epoch 14/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0036
Epoch 15/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0036
Epoch 16/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0036
Epoch 17/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0035
Epoch 18/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0031
Epoch 19/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0033
Epoch 20/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0035
```

```
Epoch 21/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0037
Epoch 22/50
144/144 [==============================] - 1s 10ms/step - loss: 0.0034
Epoch 23/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0032
Epoch 24/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0037
Epoch 25/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0042
Epoch 26/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0031
Epoch 27/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0028
Epoch 28/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0028
Epoch 29/50
```

```
loss_pre_epoch = Model.history.history['loss']
plt.plot(range(len(loss_pre_epoch)),loss_pre_epoch)
```

```
[<matplotlib.lines.Line2D at 0x7e7ce41af760>]
```




```
last_train_batch= scaled_train[-12:]
```

```
last_train_batch = last_train_batch.reshape((1,n_input,n_features))
```

```
Model.predict(last_train_batch)
```

```
1/1 [==============================] - 0s 218ms/step
array([[0.62260085]], dtype=float32)
```

```
scaled_test[0]
```

```
array([0.67548077])
```

```
test_predicition = []
first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1,n_input,n_features))

for i in range(len(test)):
  current_pred = Model.predict(current_batch)[0]
  test_predicition.append(current_pred)
  current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
```

```
test.head()
```

Date	Production
1975-01-01	834
1975-02-01	782
1975-03-01	892
1975-04-01	903
1975-05-01	966

```
true_predicition = scaler.inverse_transform(test_predicition)
```

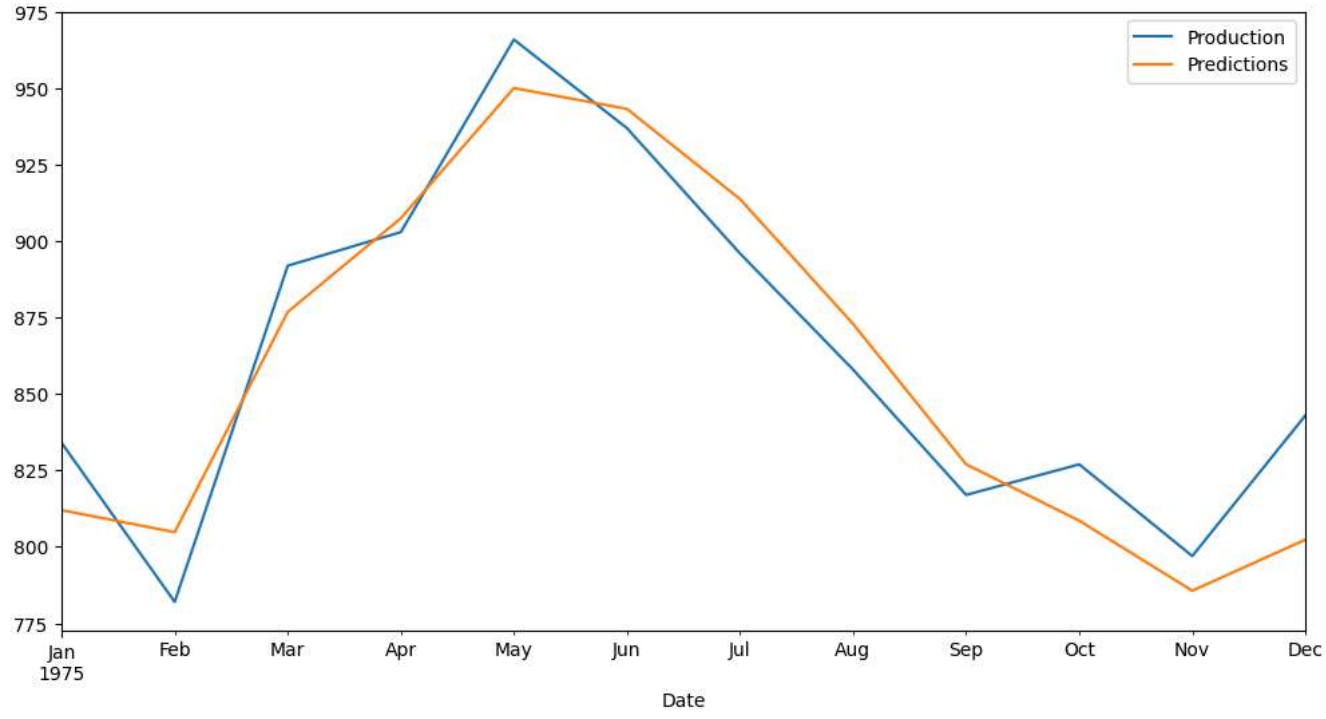
```
test['Predictions'] = true_predicition
```

```
<ipython-input-65-b4e797b13392>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  test['Predictions'] = true_predicition
```

```
test.plot(figsize=(12,6))
```

```
<Axes: xlabel='Date'>
```



Start coding or generate with AI