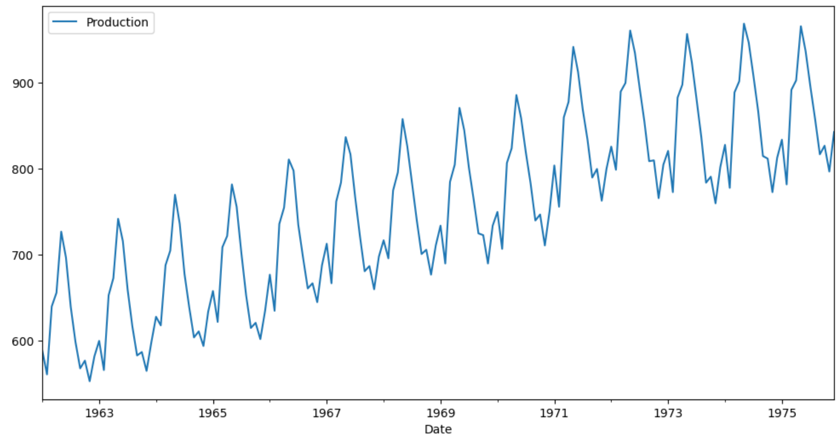
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

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

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/monthly_milk_production.csv',index_col = 'Date',parse_dates= True)
df.head()
```

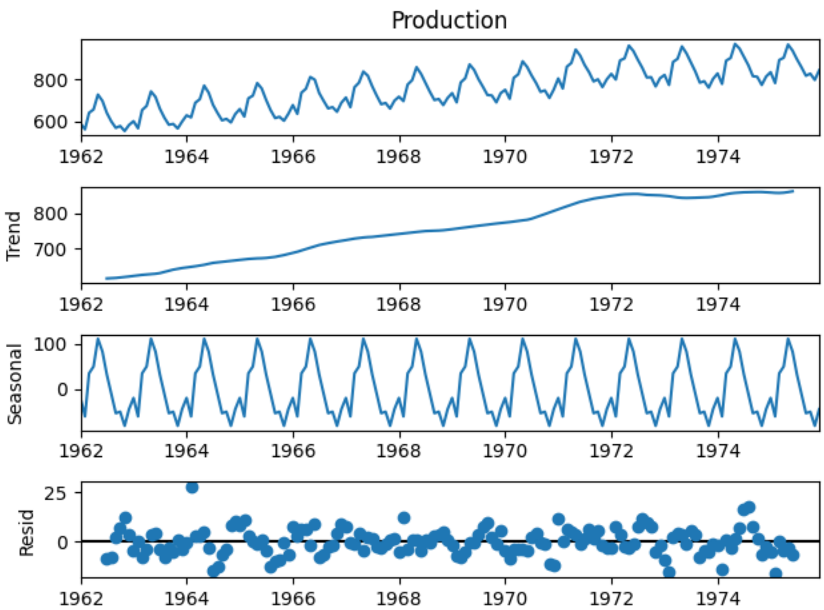
Date	Production
1962-01-01	589
1962-02-01	561
1962-03-01	640
1962-04-01	656
1962-05-01	727

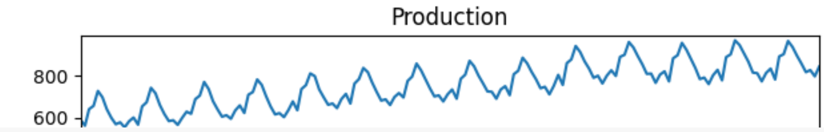
```
df.plot(figsize = (12,6))
```

<Axes: xlabel='Date'>




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






```
df.shape
```

```
(168, 1)
```

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

```
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.tail()
```

Date	Production
1975-08-01	858
1975-09-01	817
1975-10-01	827
1975-11-01	797
1975-12-01	843

```
df.iloc[:156].describe()
```

	Production
count	156.000000
mean	746.403846
std	100.277536

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

```
scaler.fit(train)
scaler_train = scaler.transform(train)
scaler_test = scaler.transform(test)
```

```
scaler_train[:5]
```

```
array([[0.08653846],
       [0.01923077],
       [0.20913462],
       [0.24759615],
       [0.41826923]])
```

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

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

```
X,y = generator[1]
```

```
X
```

```
array([[[0.01923077],
        [0.20913462],
        [0.24759615]]])
```

```
y
```

```
array([[0.41826923]])
```

```
X.shape
```

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

```
# consider n_input = 12
n_input = 12
n_features = 1
generator = TimeseriesGenerator(scaler_train,scaler_train,length = n_input,batch_size = n_features)
```

```
# define model

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

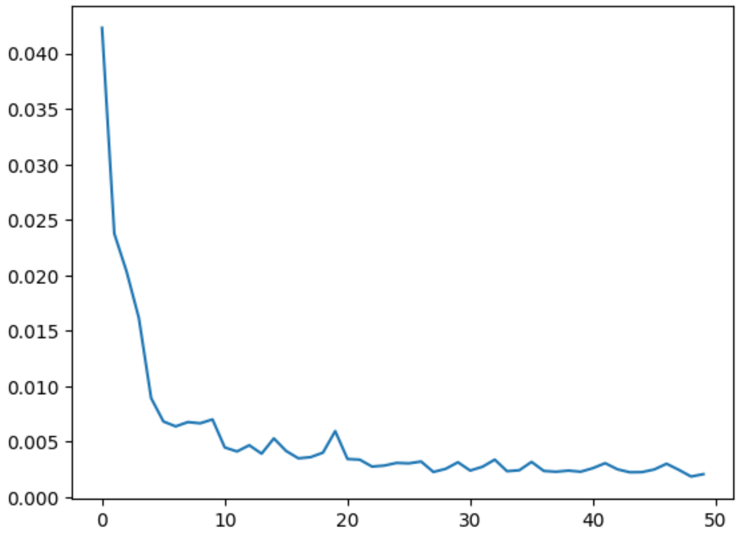
```
# fit model

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

```
Epoch 1/50
144/144 [==============================] - 4s 6ms/step - loss: 0.0423
Epoch 2/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0238
Epoch 3/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0203
Epoch 4/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0162
Epoch 5/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0089
Epoch 6/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0068
Epoch 7/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0064
```

```
Epoch 8/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0068
Epoch 9/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0067
Epoch 10/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0070
Epoch 11/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0045
Epoch 12/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0041
Epoch 13/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0047
Epoch 14/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0039
Epoch 15/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0053
Epoch 16/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0042
Epoch 17/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0035
Epoch 18/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0036
Epoch 19/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0040
Epoch 20/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0060
Epoch 21/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0035
Epoch 22/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0034
Epoch 23/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0028
Epoch 24/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0029
Epoch 25/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0031
Epoch 26/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0031
Epoch 27/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0032
Epoch 28/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0023
Epoch 29/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0026
```

```
# plot loss

loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

```
[<matplotlib.lines.Line2D at 0x7f0d65340640>]
```




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

```
array([[0.66105769],
       [0.54086538],
       [0.80769231],
       [0.83894231],
       [1.        ],
       [0.94711538],
       [0.85336538],
       [0.75480769],
       [0.62980769],
       [0.62259615],
```

```
       [0.52884615],
       [0.625     ]])
```

```
last_train_batch.shape
```

```
(12, 1)
```

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

```
(1, 12, 1)
```

```
model.predict(last_train_batch)[0]
```

```
1/1 [==============================] - 0s 35ms/step
array([0.6542587], dtype=float32)
```

```
scaler_test[0]
```

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

```
# prediction on test data

test_pred_list = []
first_eval_batch = scaler_train[-12:]
current_batch = first_eval_batch.reshape(1,n_input,n_features)
```

```
current_batch
```

```
array([[[0.66105769],
        [0.54086538],
        [0.80769231],
        [0.83894231],
        [1.        ],
        [0.94711538],
        [0.85336538],
        [0.75480769],
        [0.62980769],
        [0.62259615],
        [0.52884615],
        [0.625     ]]])
```

```
current_batch[:,1:,:]
```

```
array([[[0.54086538],
        [0.80769231],
        [0.83894231],
        [1.        ],
        [0.94711538],
        [0.85336538],
        [0.75480769],
        [0.62980769],
        [0.62259615],
        [0.52884615],
        [0.625     ]]])
```

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

```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```

```
test_pred_list
```

```
[array([0.6542587], dtype=float32),
 array([0.6213167], dtype=float32),
 array([0.80373406], dtype=float32),
```

```
 array([0.8725196], dtype=float32),
 array([0.98384434], dtype=float32),
 array([0.9645657], dtype=float32),
 array([0.89062935], dtype=float32),
 array([0.79255223], dtype=float32),
 array([0.6760939], dtype=float32),
 array([0.63659084], dtype=float32),
 array([0.5772177], dtype=float32),
 array([0.6196732], dtype=float32)]
```

```
test.head()
```

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

```
true_pred = scaler.inverse_transform(test_pred_list)
true_pred
```

```
array([[825.17163086],
       [811.46773529],
       [887.35337067],
       [915.96815872],
       [962.27924538],
       [954.25932884],
       [923.50181007],
       [882.70172882],
       [834.25505257],
       [817.82178879],
       [793.12256241],
       [810.78404808]])
```

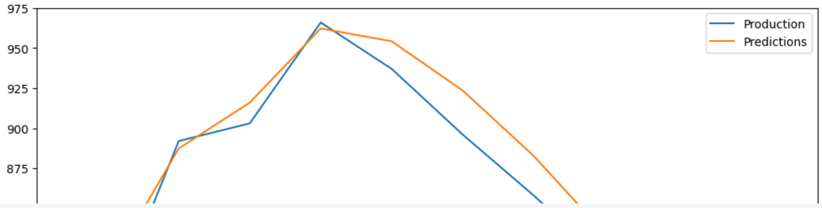
```
test['Predictions'] = true_pred
test.head()
```

```
<ipython-input-52-cda35cb79f6b>: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
  test['Predictions'] = true_pred
```

	Production	Predictions
Date		
1975-01-01	834	825.171631
1975-02-01	782	811.467735
1975-03-01	892	887.353371
1975-04-01	903	915.968159
1975-05-01	966	962.279245

```
test.plot(figsize = (12,5))
```

<Axes: xlabel='Date'>




```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(test['Production'],test['Predictions']))
print(rmse)
```

18.81750895940628

✓ 0s completed at 21:55