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

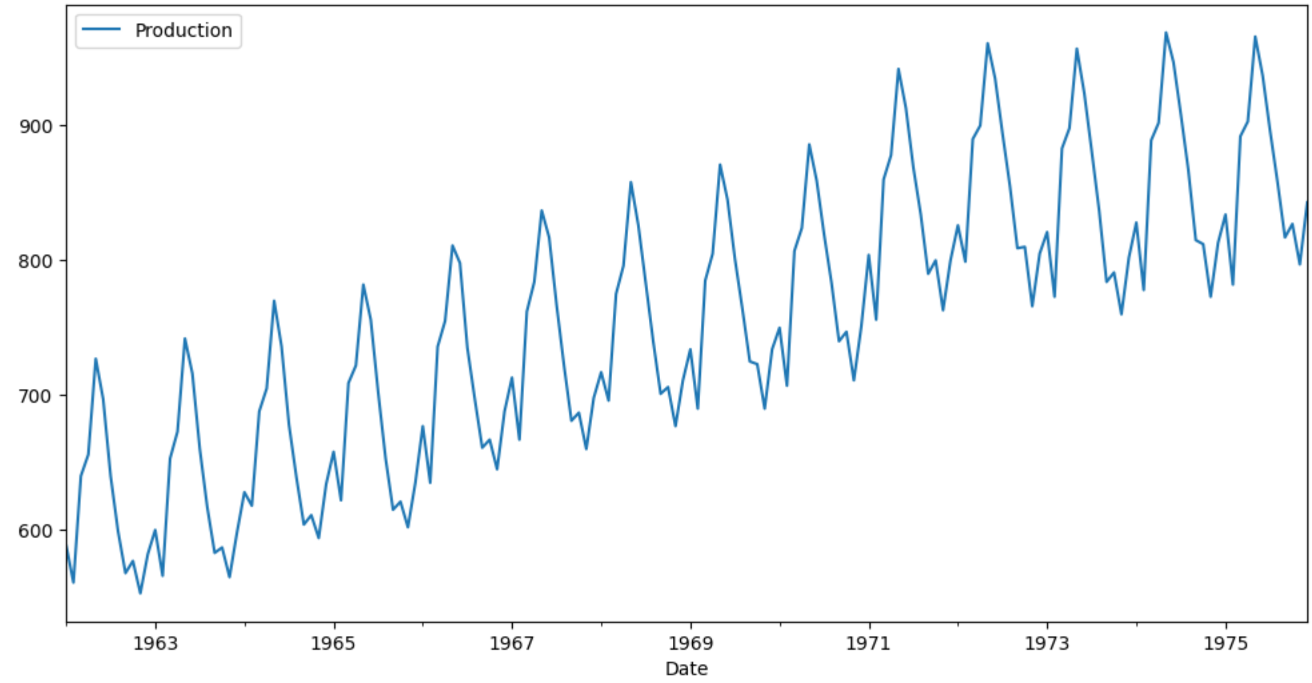
```
df = pd.read_csv('/content/monthly_milk_production.csv',index_col='Date',parse_dates=True)
df.index.freq='MS'
```

```
df.head()
```

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

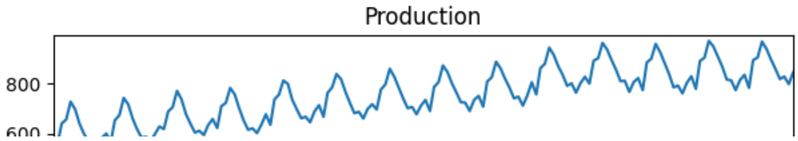
```
df.plot(figsize=(12,6))
```

```
<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
```

```
# define generator
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)
```

```
# We do the same thing, but now instead for 12 months
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


# 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')


model.summary()
```

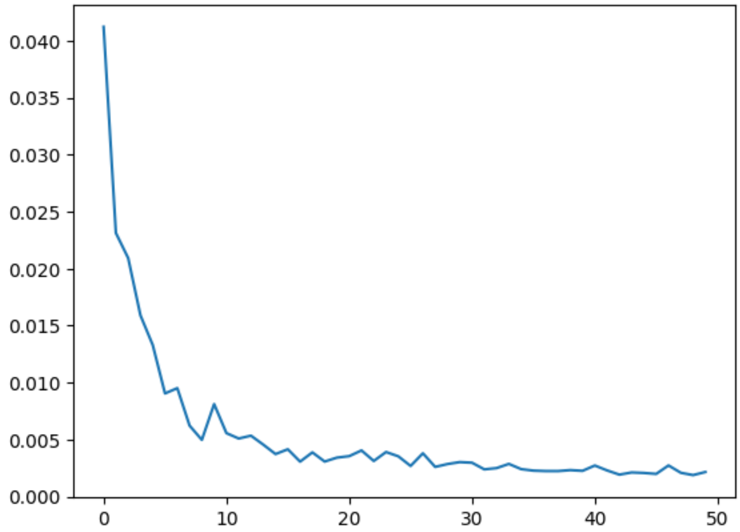
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               40800

 dense (Dense)               (None, 1)                 101

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

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

```
Epoch 1/50
144/144 [==============================] - 3s 10ms/step - loss: 0.0412
Epoch 2/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0231
Epoch 3/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0209
Epoch 4/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0159
Epoch 5/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0133
Epoch 6/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0090
Epoch 7/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0095
Epoch 8/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0062
Epoch 9/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0050
Epoch 10/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0081
Epoch 11/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0056
Epoch 12/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0051
Epoch 13/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0053
Epoch 14/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0046
Epoch 15/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0037
Epoch 16/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0041
Epoch 17/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0031
Epoch 18/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0039
Epoch 19/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0031
Epoch 20/50
144/144 [==============================] - 1s 9ms/step - loss: 0.0034
Epoch 21/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0035
Epoch 22/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0040
Epoch 23/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0031
Epoch 24/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0039
Epoch 25/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0035
Epoch 26/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0027
Epoch 27/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0038
Epoch 28/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0026
Epoch 29/50
```

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

```
[<matplotlib.lines.Line2D at 0x7940b2ec7100>]
```




```
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 206ms/step
array([[0.6740278]], dtype=float32)
```

```
scaled_test[0]
```

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

```
test_predictions = []

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

for i in range(len(test)):

    # get the prediction value for the first batch
    current_pred = model.predict(current_batch)[0]

    # append the prediction into the array
    test_predictions.append(current_pred)

    # use the prediction to update the batch and remove the first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

```
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
```

```
test_predictions
```

```
[array([0.6740278], dtype=float32),
 array([0.6351541], dtype=float32),
 array([0.81465983], dtype=float32),
 array([0.8758115], dtype=float32),
 array([0.9919943], dtype=float32),
 array([0.9742231], dtype=float32),
 array([0.9031107], dtype=float32),
 array([0.8039626], dtype=float32),
```

```
       array([0.69187504], dtype=float32),
       array([0.66172874], dtype=float32),
       array([0.60486245], dtype=float32),
       array([0.65057373], dtype=float32)]
```

```
true_predictions = scaler.inverse_transform(test_predictions)
```

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

```
20.078180854790947
```