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

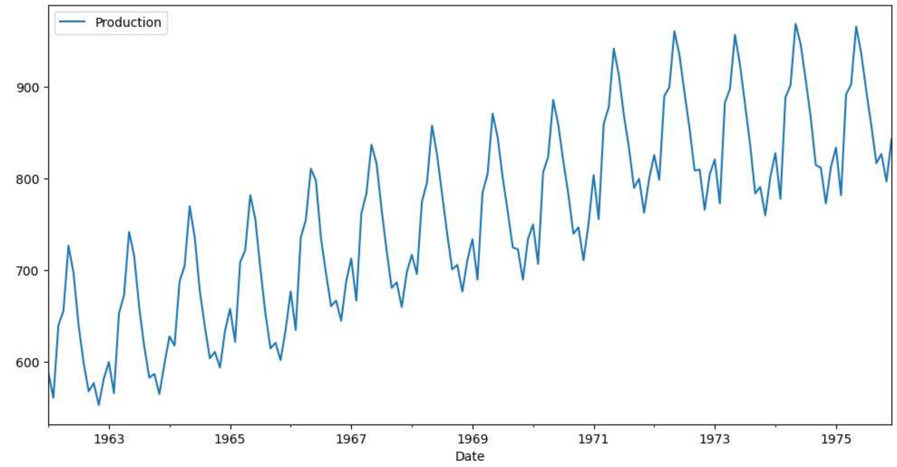
```
df = pd.read_csv('/content/monthly_milk_production.csv',index_col='Date',parse_dates=True)
df.index.freq='MS'
```

```
df.head()
```

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

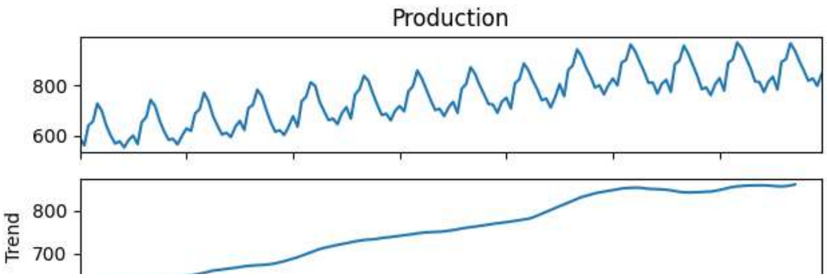
```
df.plot(figsize=(12,6))
```

```
<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_1"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     lstm_1 (LSTM)               (None, 100)               40800     
                                                                     
     dense_1 (Dense)             (None, 1)                 101       
                                                                     
    =================================================================
    Total params: 40901 (159.77 KB)
    Trainable params: 40901 (159.77 KB)
    Non-trainable params: 0 (0.00 Byte)
    _________________________________________________________________
```

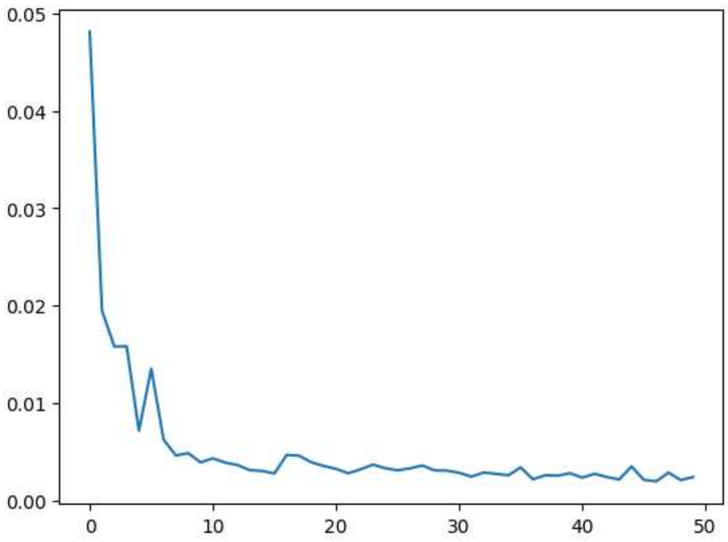
```
# fit model
model.fit(generator,epochs=50)
```

```
    Epoch 20/50
    144/144 [==============================] - 1s 7ms/step - loss: 0.0035
    Epoch 21/50
    144/144 [==============================] - 1s 7ms/step - loss: 0.0033
    Epoch 22/50
```

```
Epoch 42/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0027
Epoch 43/50
144/144 [==============================] - 1s 8ms/step - loss: 0.0024
Epoch 44/50
144/144 [==============================] - 2s 11ms/step - loss: 0.0022
Epoch 45/50
144/144 [==============================] - 2s 11ms/step - loss: 0.0035
Epoch 46/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0021
Epoch 47/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0020
Epoch 48/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0029
Epoch 49/50
```

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

```
[<matplotlib.lines.Line2D at 0x7cbc7f7b21a0>]
```



```
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 286ms/step
array([[0.69518054]], 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 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
```

```
test_predictions
```

```
[array([0.69518054], dtype=float32),
 array([0.6638044], dtype=float32),
 array([0.88048863], dtype=float32),
 array([0.95451444], dtype=float32),
 array([1.0739182], dtype=float32),
 array([1.0499061], dtype=float32),
 array([0.9602645], dtype=float32),
 array([0.8459554], dtype=float32),
 array([0.723558], dtype=float32),
 array([0.6854317], dtype=float32),
 array([0.62957466], dtype=float32),
 array([0.6697108], 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_predictions = scaler.inverse_transform(test_predictions)
```

```
test['Predictions'] = true_predictions
```

```
<ipython-input-64-920b79c3c314>: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-a-c
  test['Predictions'] = true_predictions
```

```
test.plot(figsize=(14,5))
```

```
<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)
```

```
37.088609967664325
```