Recurrent Neural Network is a sequence model, used mainly for Natural Language Processing task In overall CNN was typically used for image classification data While RNN is mainly used in NLP.

RNN are typically used in NLP related domains .. some of them are listed for your reference

1) Machine Translation : Google Translator is one of the applications build using RNN

2) Speech Recognition: Alexa, Google Duo, Siri

3) Sentiment Analysis: Predict a sentence whether it is positive or negative.

4) Text Auto Completion: Gmail

In [ ]:

```

```

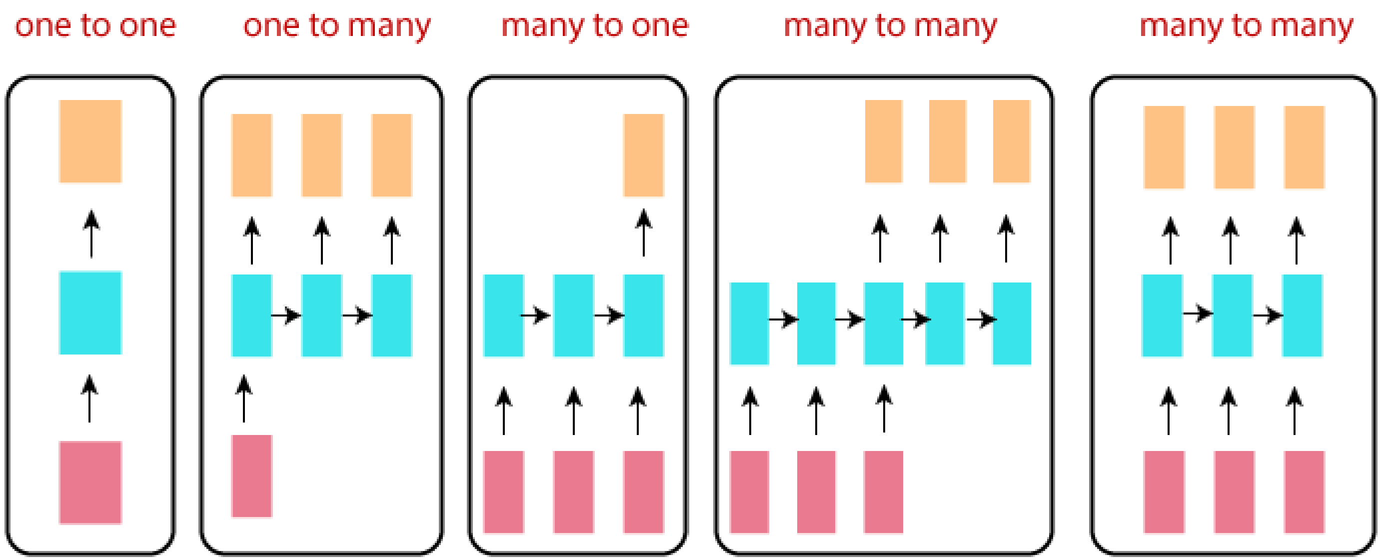
Types of RNN

1) One to One : where number of input is one and the number of output is one

2) One to Many : Number of input = 1 and number of output > 1

3) Many to one : Number of input > 1 and output is = 1

4) Many to Many : Number if input > 1 and number of output > 1



In [ ]:

```
 
```

In [ ]:

```
 
```

Example

In [102]:

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

In [103]:

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

In [104]:
```
df.head()
```

Out[104]:

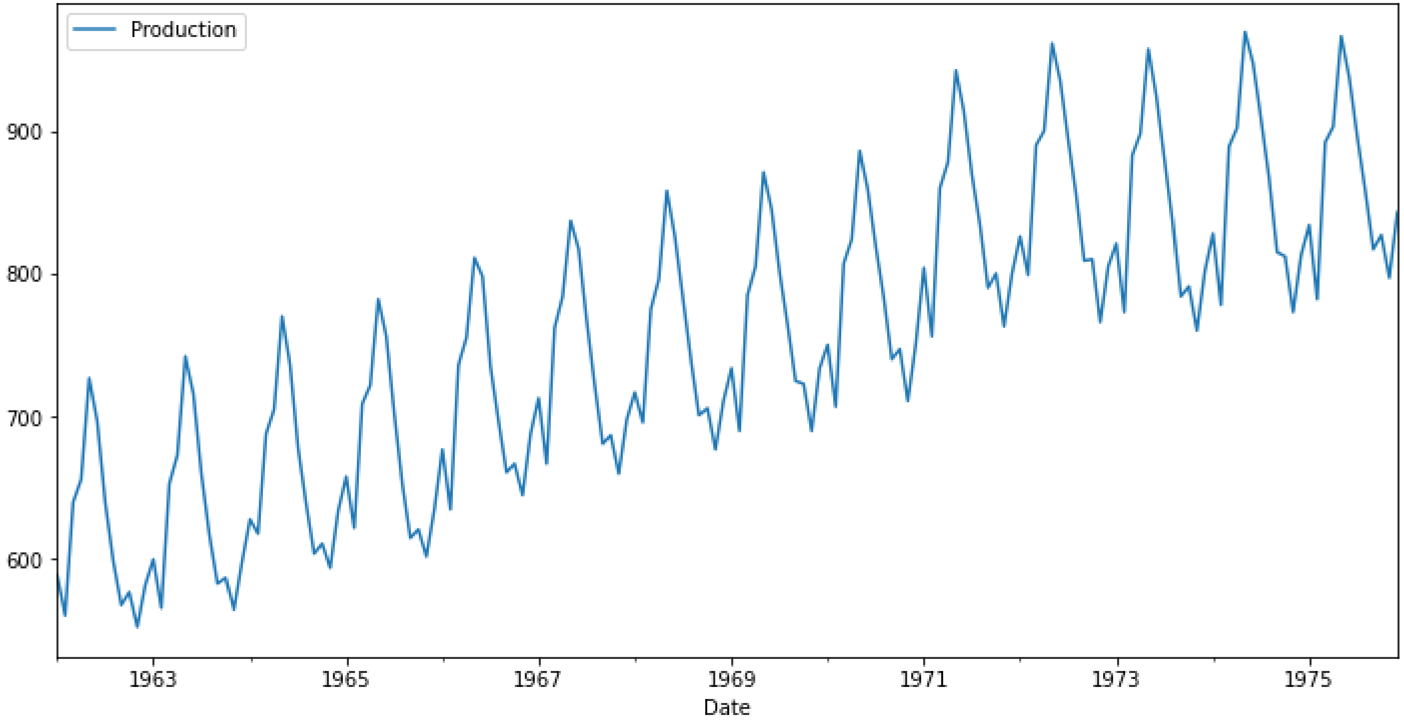
	Production
Date	
1962-01-01	589
1962-02-01	561
1962-03-01	640
1962-04-01	656
1962-05-01	727

In [ ]:
```

```

In [105]:
```
df.plot(figsize=(12,6))
```

Out[105]: <AxesSubplot:xlabel='Date'>



In [106]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

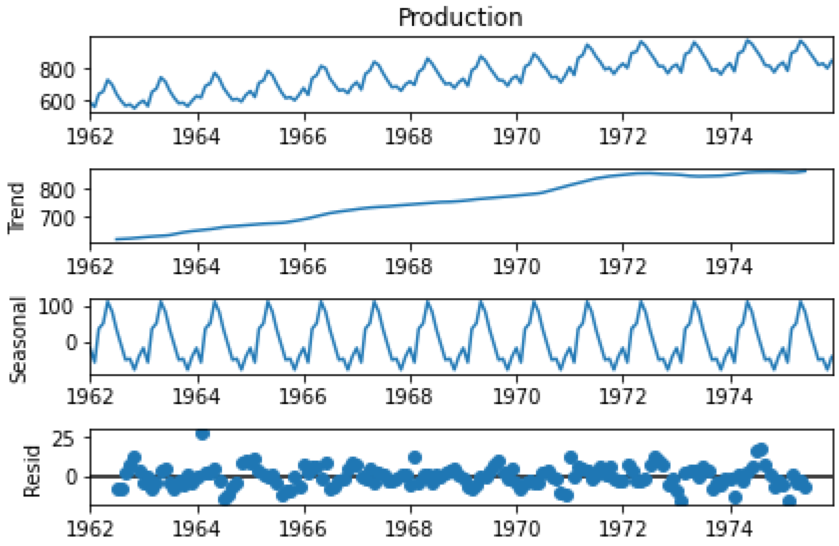
In [107]:
```
result = seasonal_decompose(df['Production'])
```

In [108]:
```
result
```

Out[108]: <statsmodels.tsa.seasonal.DecomposeResult at 0x7efca3edfa90>

In [109]:
```
plt.figure(figsize=(12,6))
result.plot()
plt.show()
```

<Figure size 864x432 with 0 Axes>



In [110]:
```
156-168
```
Out[110]: -12

In [111]:
```
len(df)
```
Out[111]: 168

In [112]:
```
train = df.iloc[:156]
```

In [113]:
```
train
```
Out[113]:

	Production
Date	
1962-01-01	589
1962-02-01	561
1962-03-01	640
1962-04-01	656
1962-05-01	727
...	...
1974-08-01	867
1974-09-01	815
1974-10-01	812
1974-11-01	773
1974-12-01	813

156 rows × 1 columns

In [114]:
```
test = df.iloc[156:]
```

In [115]:
```
test
```

Out[115]:

	Production
Date	
1975-01-01	834
1975-02-01	782
1975-03-01	892
1975-04-01	903
1975-05-01	966
1975-06-01	937
1975-07-01	896
1975-08-01	858
1975-09-01	817
1975-10-01	827
1975-11-01	797
1975-12-01	843

In [116]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [117]:
```
scaler = MinMaxScaler()
```

In [118]:
```
scaler.fit(train)
```

Out[118]: MinMaxScaler()

In [119]:
```
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

In [120]:
```
scaled_test
```

Out[120]:
```
array([[0.67548077],
       [0.55048077],
       [0.81490385],
       [0.84134615],
       [0.99278846],
       [0.92307692],
       [0.82451923],
       [0.73317308],
       [0.63461538],
       [0.65865385],
       [0.58653846],
       [0.69711538]])
```

In [121]:
```
scaled_train
```

```
       [0.05769231],
       [0.        ],
       [0.06971154],
       [0.11298077],
       [0.03125   ],
       [0.24038462],
       [0.28846154],
       [0.45432692],
       [0.39182692],
       [0.25721154],
       [0.15384615],
       [0.07211538],
       [0.08173077],
       [0.02884615],

       [0.10817308],
       [0.18028846],
       [0.15625   ],
       [0.32451923],
       [0.36538462],
```

In [ ]:
```

```

defining the model

In [122]:
```
from keras.preprocessing.sequence import TimeseriesGenerator
```

In [123]:
```
n_input = 3
n_features = 1
generator = TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=n_features)
```

In [124]:
```
print(len(scaled_train))
print(len(generator))
```

```
156
153
```

In [152]:
```
x,y = generator[0]
```

In [155]:
```
x
```

Out[155]:
```
array([[[0.08653846],
        [0.01923077],
        [0.20913462],
        [0.24759615],
        [0.41826923],
        [0.34615385],
        [0.20913462],
        [0.11057692],
        [0.03605769],
        [0.05769231],
        [0.        ],
        [0.06971154]]])
```

In [125]:

```
x,y = generator[0]
print(f'Given Array is \n{x.flatten()}')
print(f'Predicted Array is \n {y}')
```

```
Given Array is
[0.08653846 0.01923077 0.20913462]
Predicted Array is
 [[0.24759615]]
```

In [126]:

```
x.shape
```

Out[126]:

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

In [ ]:

In [127]:

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

In [ ]:

In [128]:

```
# x1,y1 = generator1[0]
# print(f'Given Array is \n{x1.flatten()}')
# print(f'Predicted Array is \n {y1}')
```

In [ ]:

In [129]:

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

In [ ]:

In [130]:

```
#defining the 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')
```

In [131]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_2 (LSTM)               (None, 100)               40800     
                                                                 
 dense_2 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

In [132]:
```
model.fit(generator,epochs=50)
```

```
Epoch 1/50
144/144 [==============================] - 2s 5ms/step - loss: 0.0401
Epoch 2/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0218
Epoch 3/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0179
Epoch 4/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0127
Epoch 5/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0058
Epoch 6/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0057
Epoch 7/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0044
Epoch 8/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0041
Epoch 9/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0029
Epoch 10/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0035
Epoch 11/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0036
Epoch 12/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0036
Epoch 13/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0035
Epoch 14/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0037
Epoch 15/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0038
Epoch 16/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0035
Epoch 17/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0030
Epoch 18/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0033
Epoch 19/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0026
Epoch 20/50
```

```
144/144 [==============================] - 1s 5ms/step - loss: 0.0032
Epoch 21/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0027
Epoch 22/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0030
Epoch 23/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0027
Epoch 24/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0028
Epoch 25/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0040
Epoch 26/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0033
Epoch 27/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0029
Epoch 28/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0027
Epoch 29/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0024
Epoch 30/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0022
Epoch 31/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0022
Epoch 32/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0024
Epoch 33/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0023
Epoch 34/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0024
Epoch 35/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0022
Epoch 36/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0023
Epoch 37/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0020
Epoch 38/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0026
Epoch 39/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0019
Epoch 40/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0020
Epoch 41/50
```

```
144/144 [==============================] - 1s 6ms/step - loss: 0.0024
Epoch 42/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0021
Epoch 43/50
144/144 [==============================] - 1s 7ms/step - loss: 0.0022
Epoch 44/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0021
Epoch 45/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0021
Epoch 46/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0027
Epoch 47/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0021
Epoch 48/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0021
Epoch 49/50
144/144 [==============================] - 1s 6ms/step - loss: 0.0017
Epoch 50/50
144/144 [==============================] - 1s 5ms/step - loss: 0.0023
```

Out[132]:
```
<keras.callbacks.History at 0x7efca3f13ac0>
```

In [133]:
```
loss_per_epoch = model.history.history['loss']
```

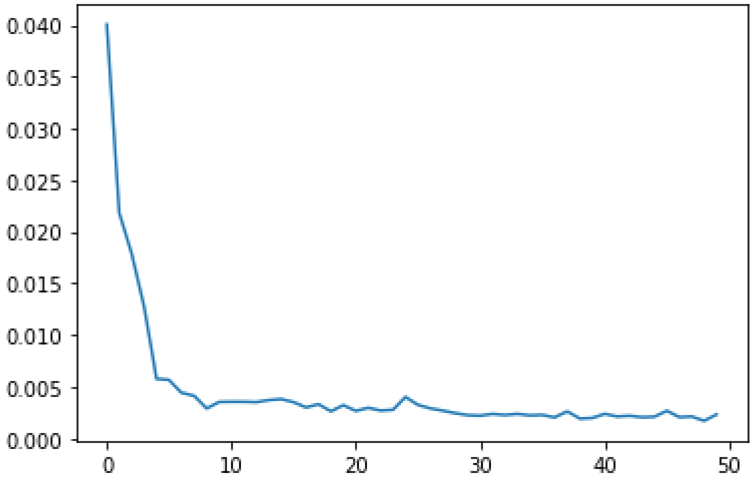
In [134]:
```
loss_per_epoch
```

Out[134]:
```
[0.04006689414381981,
 0.021842949092388153,
 0.01788419671356678,
 0.012746957130730152,
 0.005774316843599081,
 0.0056803287006914616,
 0.004432339686900377,
 0.004116366617381573,
 0.0029001617804169655,
 0.003533754963427782,
 0.0035526466090232134,
 0.0035502261016517878,
 0.003515976946800947,
 0.0037142904475331306,
 0.003810893278568983,
 0.0035076248459517956,
 0.003002116922289133,
 0.0033142208121716976,
 0.002624667249619961,
 0.0032198664266616106,
 0.0026647131890058517,
 0.0029616085812449455,
 0.0026995078660547733,
 0.0027742411475628614,
 0.004015814512968063,
 0.0032504559494554996,
 0.0029056945350021124,
 0.0026901643723249435,
 0.0024448975455015898,
 0.002249694662168622,
 0.0021993413101881742,
 0.002367268083617091,
 0.002265089424327016,
 0.002371768467128277,
 0.0022283869329839945,
 0.002273865509778261,
 0.0020332641433924437,
 0.0026074135676026344,
 0.0019142826786264777,
```

```
0.001977734500542283,
0.0023750613909214735,
0.002119295997545123,
0.002193675609305501,
0.0020502954721450806,
0.002097403397783637,
0.002673913724720478,
0.0020592238288372755,
0.0021182475611567497,
0.0017006704583764076,
0.0023153924848884344]
```

In [135]:
```
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

Out[135]: [<matplotlib.lines.Line2D at 0x7efca3afba00>]



In [156]:
```
len(scaled_train)
```

Out[156]: 156

In [136]:
```
last_train_batch = scaled_train[-12:]
```

In [137]:
```
last_train_batch
```

Out[137]:
```
array([[0.66105769],
       [0.54086538],
       [0.80769231],
       [0.83894231],
       [1.        ],
       [0.94711538],
       [0.85336538],
       [0.75480769],
       [0.62980769],
       [0.62259615],
       [0.52884615],
       [0.625     ]])
```

In [138]:
```
last_train_batch = last_train_batch.reshape((1,n_input,n_features))
```

In [157]:
```
last_train_batch.shape
```

Out[157]:
```
(1, 12, 1)
```

In [140]:
```
model.predict(last_train_batch)
```

Out[140]:
```
array([[0.66922015]], dtype=float32)
```

In [141]:
```
scaled_test[0]
```

Out[141]:
```
array([0.67548077])
```

In [158]:

```
test
```

Out[158]:

Date	Production	predictions
1975-01-01	834	831.395582
1975-02-01	782	810.352978
1975-03-01	892	887.497755
1975-04-01	903	909.254545
1975-05-01	966	949.122810
1975-06-01	937	945.390984
1975-07-01	896	921.605249
1975-08-01	858	886.468988
1975-09-01	817	838.733150
1975-10-01	827	824.217772
1975-11-01	797	796.991980
1975-12-01	843	817.327415

In [164]:

```
test_prediction = []
first_evaluation_batch = scaled_train[-n_input:]
current_batch = first_evaluation_batch.reshape((1,n_input,n_features))
# print(current_batch[0])


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

In [143]:
```
test_prediction
```

Out[143]:
```
[array([0.66922015], dtype=float32),
 array([0.61863697], dtype=float32),
 array([0.80408114], dtype=float32),
 array([0.8563811], dtype=float32),
 array([0.9522183], dtype=float32),
 array([0.94324756], dtype=float32),
 array([0.8860703], dtype=float32),
 array([0.80160815], dtype=float32),
 array([0.68685853], dtype=float32),
 array([0.6519658], dtype=float32),
 array([0.5865192], dtype=float32),
 array([0.63540244], dtype=float32)]
```

In [144]:
```
test.head()
```

Out[144]:

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

In [167]:
```
test_prediction = scaler.inverse_transform(test_prediction)
```

In [168]:
```
test_prediction
```

Out[168]:
```
array([[831.3955822 ],
       [810.35297775],
       [887.49775505],
       [909.25454521],
       [949.12281036],
       [945.39098358],
       [921.6052494 ],
       [886.46898842],
       [838.73315048],
       [824.21777153],
       [796.9919796 ],
       [817.32741547]])
```

In [169]:
```
test['predictions'] = test_prediction
```

```
<ipython-input-169-0f322b0ea925>: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-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  test['predictions'] = test_prediction
```

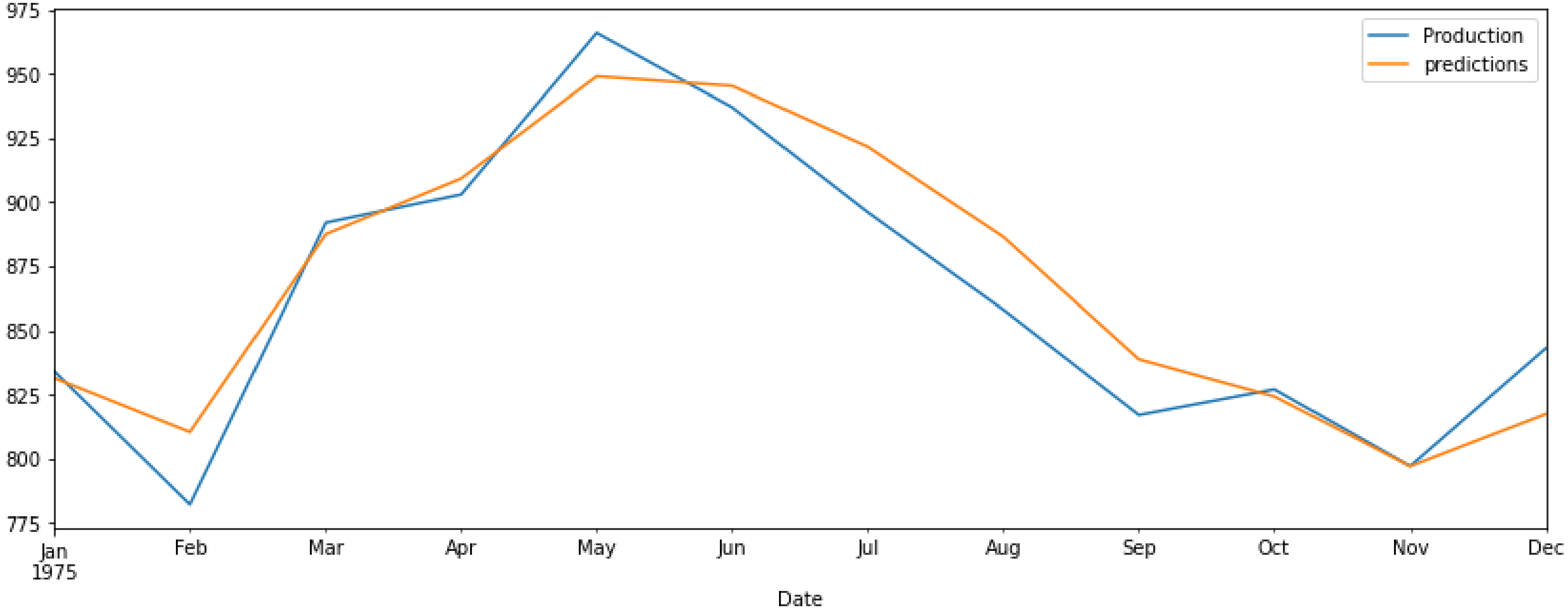
In [170]:
```
test
```

Out[170]:

Date	Production	predictions
1975-01-01	834	831.395582
1975-02-01	782	810.352978
1975-03-01	892	887.497755
1975-04-01	903	909.254545
1975-05-01	966	949.122810
1975-06-01	937	945.390984
1975-07-01	896	921.605249
1975-08-01	858	886.468988
1975-09-01	817	838.733150
1975-10-01	827	824.217772
1975-11-01	797	796.991980
1975-12-01	843	817.327415

In [148]:
```
test.plot(figsize=(14,5))
```

Out[148]: <AxesSubplot:xlabel='Date'>



In [ ]:
```

```