Importing packages and setup the dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Changing Index and parsing date column

In [2]:
```
df = pd.read_csv('Chiali_Sales.csv',index_col='Month',parse_dates=True)
df.index.freq='MS'
```

In [3]:
```
df.head()
```

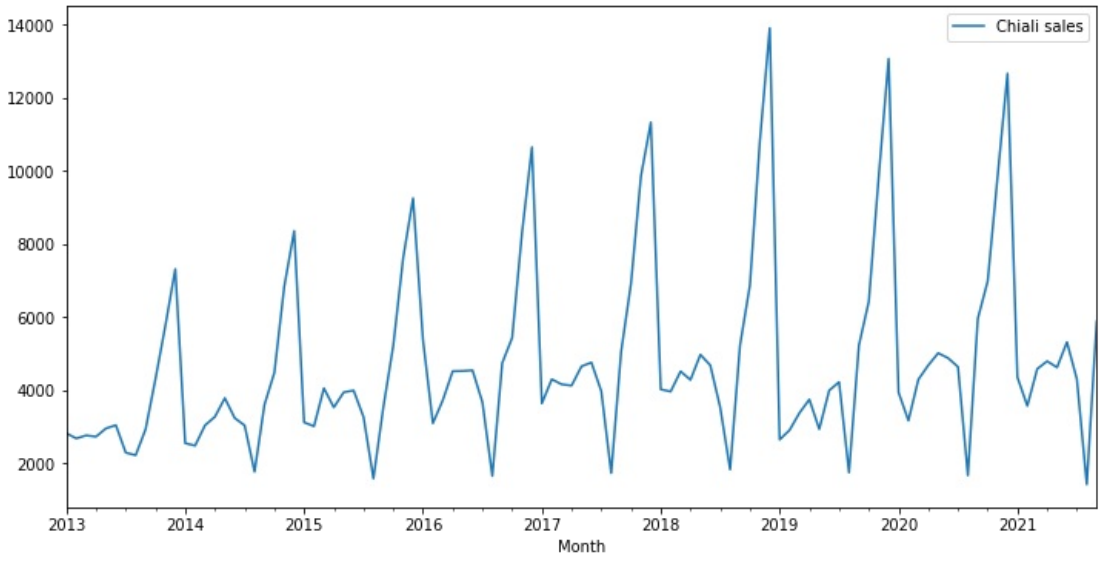
Out[3]:

Month	Chiali sales
2013-01-01	2815
2013-02-01	2672
2013-03-01	2755
2013-04-01	2721
2013-05-01	2946

Visualizing data

In [4]:
```
df.plot(figsize=(12,6))
```

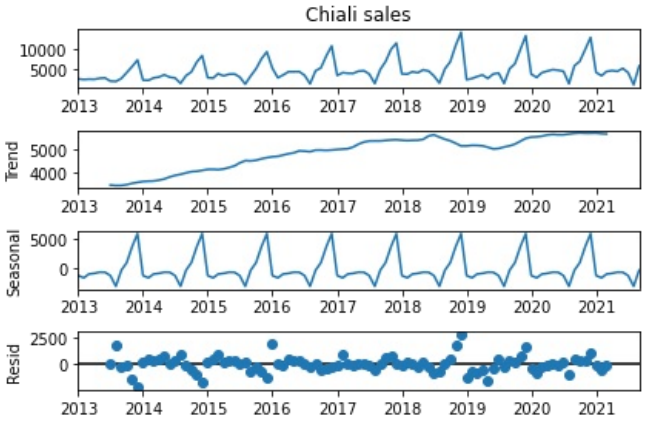
Out[4]:
```
<AxesSubplot:xlabel='Month'>
```



Decomposing the dataset in order to insepct its trends, seasonality and noises

In [5]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
df.sort_index(inplace= True)
```

In [6]:
```
results = seasonal_decompose(df['Chiali sales'])
results.plot();
```



In [7]:
```
len(df)
```

Out[7]:
```
105
```

Split the data into train 80% and test 20% sets

We chose the test data to be the last year 12 months to predict

In [8]:
```
train = df.iloc[:93]
test = df.iloc[93:]
```

Using MinMaxScaler to scale data between 0-1 to improve the model efficiency

In [9]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [10]:
```
df.head(),df.tail()
```

Out[10]:
```
(            Chiali sales
 Month
 2013-01-01          2815
 2013-02-01          2672
 2013-03-01          2755
 2013-04-01          2721
 2013-05-01          2946,
             Chiali sales
 Month
 2021-05-01          4618
 2021-06-01          5312
 2021-07-01          4298
 2021-08-01          1413
 2021-09-01          5877)
```

In [11]:
```
scaler.fit(train)
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

In [12]:
```
scaled_train[:10]
```

Out[12]:
```
array([[0.10062384],
       [0.08903832],
       [0.09576278],
       [0.09300818],
       [0.11123714],
       [0.11852872],
       [0.05744146],
       [0.05177023],
       [0.10929272],
       [0.22101596]])
```

In [13]:
```
from keras.preprocessing.sequence import TimeseriesGenerator
```

Example to showcase the recurrent neural network functionality

In [14]:
```
# define generator
n_input = 3
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

In [15]:
```
X,y = generator[0]
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.10062384 0.08903832 0.09576278]
Predict this y:
 [[0.09300818]]
```

In [16]:
```
X.shape
```

Out[16]:
```
(1, 3, 1)
```

In [17]:
```
# 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)
```

Creating the model

In [18]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

In [19]:
```
# 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')
```

In [20]:
```
model.summary()
```

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

Fitting the model and insepct the model's overfitting

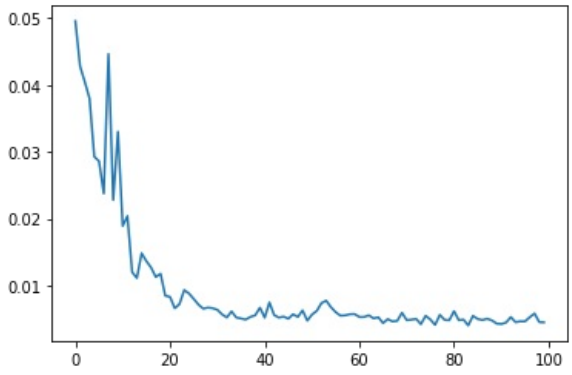
In [21]:
```
# fit model
model.fit(generator,epochs=100)
```

```
Epoch 1/100
81/81 [==============================] - 2s 5ms/step - loss: 0.0496
Epoch 2/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0428A: 0s - loss:
Epoch 3/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0405
Epoch 4/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0380
Epoch 5/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0293
Epoch 6/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0286
Epoch 7/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0238
Epoch 8/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0446
Epoch 9/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0228
Epoch 10/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0330
Epoch 90/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0043
Epoch 91/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0043
Epoch 92/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0045
Epoch 93/100
81/81 [==============================] - 1s 6ms/step - loss: 0.0053
Epoch 94/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0045
Epoch 95/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0047
Epoch 96/100
81/81 [==============================] - 1s 6ms/step - loss: 0.0047
Epoch 97/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0053
Epoch 98/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0058
Epoch 99/100
81/81 [==============================] - 0s 5ms/step - loss: 0.0045
Epoch 100/100
81/81 [==============================] - 0s 6ms/step - loss: 0.0045
```

Out[21]:
```
<keras.callbacks.History at 0x1ea728ced90>
```

In [22]:
```
loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x1ea74e307f0>]
```



In [23]:
```
last_train_batch = scaled_train[-12:]
```

In [24]:
```
last_train_batch = last_train_batch.reshape((1, n_input, n_features))
```

In [25]:
```
model.predict(last_train_batch)
```

Out[25]:
```
array([[0.44591102]], dtype=float32)
```

In [26]:
```
scaled_test[0]
```

Out[26]:
```
array([0.43814308])
```

Implementing the model into the last 12 months of the data

In [27]:
```
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)
```

Results

In [28]:
```
test_predictions
```

Out[28]:
```
[array([0.44591102], dtype=float32),
 array([0.7682245], dtype=float32),
 array([1.0433576], dtype=float32),
 array([0.21084248], dtype=float32),
 array([0.14406237], dtype=float32),
 array([0.21294564], dtype=float32),
 array([0.24958393], dtype=float32),
 array([0.2826083], dtype=float32),
 array([0.28106833], dtype=float32),
 array([0.2671063], dtype=float32),
 array([0.10059892], dtype=float32),
 array([0.37614554], dtype=float32)]
```

In [29]:
```
test.head()
```

Out[29]:

	Chiali sales
Month	
2020-10-01	6981
2020-11-01	9851
2020-12-01	12670
2021-01-01	4348
2021-02-01	3564

Re-scaling the data into its original form

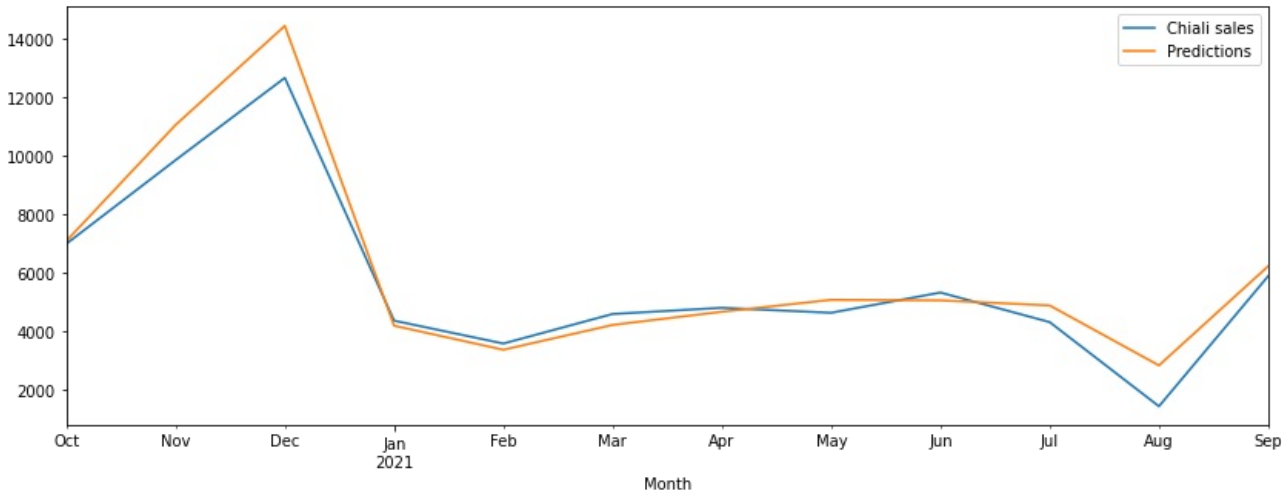
In [30]:
```
true_predictions = scaler.inverse_transform(test_predictions)
```

In [31]:
```
test['Predictions'] = true_predictions
```

Visualizing the results

In [32]:
```
test.plot(figsize=(14,5))
```

Out[32]:
```
<AxesSubplot:xlabel='Month'>
```



In [33]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse=sqrt(mean_squared_error(test['Chiali sales'],test['Predictions']))
print(rmse)
```

```
792.7716805658163
```

In [ ]:
```
```