```python

from pandas import DataFrame

from pandas import Series

from pandas import concat

from pandas import read\_csv

from pandas import datetime

#from sklearn.metrics import median\_absolute\_error

#from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from math import sqrt

from matplotlib import pyplot

import numpy as np

import pandas as pd

```

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\ipykernel\_launcher.py:5: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas in a future version. Import from datetime module instead.

"""

Using TensorFlow backend.

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint8 = np.dtype([("qint8", np.int8, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint8 = np.dtype([("quint8", np.uint8, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint16 = np.dtype([("qint16", np.int16, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint16 = np.dtype([("quint16", np.uint16, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint32 = np.dtype([("qint32", np.int32, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

np\_resource = np.dtype([("resource", np.ubyte, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint8 = np.dtype([("qint8", np.int8, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint8 = np.dtype([("quint8", np.uint8, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint16 = np.dtype([("qint16", np.int16, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_quint16 = np.dtype([("quint16", np.uint16, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

\_np\_qint32 = np.dtype([("qint32", np.int32, 1)])

C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\tensorboard\compat\tensorflow\_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.

np\_resource = np.dtype([("resource", np.ubyte, 1)])

```python

# Hardcode all variables

predict\_batch\_size\_exp = 1

epoch\_exp = 50

neurons\_exp = 5

predict\_values\_exp = 168

lag\_exp=24

```

```python

# frame a sequence as a supervised learning problem

def timeseries\_to\_supervised(data, lag=1):

df = DataFrame(data)

columns = [df.shift(i) for i in range(1, lag+1)]

columns.append(df)

df = concat(columns, axis=1)

df.fillna(0, inplace=True)

return df

```

```python

def difference(dataset, interval=1):

diff = list()

for i in range(interval, len(dataset)):

value = dataset[i] - dataset[i - interval]

diff.append(value)

return Series(diff)

```

```python

# invert differenced value

def inverse\_difference(history, yhat, interval=1):

return yhat + history[-interval]

```

```python

# scale train and test data to [-1, 1]

def scale(train, test):

# fit scaler

scaler = MinMaxScaler(feature\_range=(-1, 1))

scaler = scaler.fit(train)

# transform train

train = train.reshape(train.shape[0], train.shape[1])

train\_scaled = scaler.transform(train)

# transform test

test = test.reshape(test.shape[0], test.shape[1])

test\_scaled = scaler.transform(test)

return scaler, train\_scaled, test\_scaled

```

```python

def invert\_scale(scaler, X, value):

new\_row = [x for x in X] + [value]

array = np.array(new\_row)

array = array.reshape(1, len(array))

inverted = scaler.inverse\_transform(array)

return inverted[0, -1]

```

```python

# fit an LSTM network to training data

from keras.layers import Activation, Dense, BatchNormalization, TimeDistributed

def fit\_lstm(train, batch\_size, nb\_epoch, neurons):

X, y = train[:, 0:-1], train[:, -1]

X = X.reshape(X.shape[0], 1, X.shape[1])

model = Sequential()

model.add(LSTM(neurons\_exp, dropout = 0.1 ,batch\_input\_shape=(batch\_size, X.shape[1], X.shape[2]), stateful=True))

model.add(BatchNormalization())

model.add(Dense(50))

model.add(Activation('relu'))

model.add(Dense(50))

model.add(Activation('tanh'))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

for i in range(nb\_epoch):

model.fit(X, y, epochs=1, batch\_size=batch\_size, verbose=1, shuffle=False)

model.reset\_states()

return model

```

```python

# make a one-step forecast with new model with batch size 1

def forecast\_lstm(model, batch\_size, X):

X = X.reshape(1, 1, len(X))

# re-define model

new\_model = Sequential()

new\_model.add(LSTM(neurons\_exp, dropout = 0.1 , batch\_input\_shape=(batch\_size, X.shape[1], X.shape[2]), stateful=True))

new\_model.add(BatchNormalization())

new\_model.add(Dense(50))

new\_model.add(Activation('relu'))

new\_model.add(Dense(50))

new\_model.add(Activation('tanh'))

new\_model.add(Dense(1))

# copy weights

old\_weights = model.get\_weights()

new\_model.set\_weights(old\_weights)

# compile model

new\_model.compile(loss='mean\_squared\_error', optimizer='adam')

#print(X)

yhat = new\_model.predict(X, batch\_size=1)

return yhat[0,0]

```

```python

''' Loading data '''

import pandas as pd

series = pd.read\_excel('AL\_WIND\_07\_12.xlsx',index\_col="DateTime")

series.head()

```

<div>

<style scoped>

.dataframe tbody tr th:only-of-type {

vertical-align: middle;

}

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</style>

<table border="1" class="dataframe">

<thead>

<tr style="text-align: right;">

<th></th>

<th>Air temperature | ('C)</th>

<th>Pressure | (atm)</th>

<th>Wind speed | (m/s)</th>

<th>Wind direction | (deg)</th>

<th>Power generated by system | (kW)</th>

</tr>

<tr>

<th>DateTime</th>

<th></th>

<th></th>

<th></th>

<th></th>

<th></th>

</tr>

</thead>

<tbody>

<tr>

<th>2007-01-01 00:00:00</th>

<td>10.926</td>

<td>0.979103</td>

<td>9.014</td>

<td>229</td>

<td>33688.1</td>

</tr>

<tr>

<th>2007-01-01 01:00:00</th>

<td>9.919</td>

<td>0.979566</td>

<td>9.428</td>

<td>232</td>

<td>37261.9</td>

</tr>

<tr>

<th>2007-01-01 02:00:00</th>

<td>8.567</td>

<td>0.979937</td>

<td>8.700</td>

<td>236</td>

<td>30502.9</td>

</tr>

<tr>

<th>2007-01-01 03:00:00</th>

<td>7.877</td>

<td>0.980053</td>

<td>8.481</td>

<td>247</td>

<td>28419.2</td>

</tr>

<tr>

<th>2007-01-01 04:00:00</th>

<td>7.259</td>

<td>0.979867</td>

<td>8.383</td>

<td>256</td>

<td>27370.3</td>

</tr>

</tbody>

</table>

</div>

```python

```

```python

from sklearn.metrics import median\_absolute\_error

from sklearn.preprocessing import MinMaxScaler

```

```python

'''Drop all the features as we will not be having any in production'''

del series['Air temperature | (\'C)']

del series['Pressure | (atm)']

del series['Wind speed | (m/s)']

del series['Wind direction | (deg)']

series.head()

```

<div>

<style scoped>

.dataframe tbody tr th:only-of-type {

vertical-align: middle;

}

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</style>

<table border="1" class="dataframe">

<thead>

<tr style="text-align: right;">

<th></th>

<th>Power generated by system | (kW)</th>

</tr>

<tr>

<th>DateTime</th>

<th></th>

</tr>

</thead>

<tbody>

<tr>

<th>2007-01-01 00:00:00</th>

<td>33688.1</td>

</tr>

<tr>

<th>2007-01-01 01:00:00</th>

<td>37261.9</td>

</tr>

<tr>

<th>2007-01-01 02:00:00</th>

<td>30502.9</td>

</tr>

<tr>

<th>2007-01-01 03:00:00</th>

<td>28419.2</td>

</tr>

<tr>

<th>2007-01-01 04:00:00</th>

<td>27370.3</td>

</tr>

</tbody>

</table>

</div>

```python

for i in range(0,16):

series = series[:-1]

series.tail()

```

<div>

<style scoped>

.dataframe tbody tr th:only-of-type {

vertical-align: middle;

}

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</style>

<table border="1" class="dataframe">

<thead>

<tr style="text-align: right;">

<th></th>

<th>Power generated by system | (kW)</th>

</tr>

<tr>

<th>DateTime</th>

<th></th>

</tr>

</thead>

<tbody>

<tr>

<th>2012-12-31 03:00:00</th>

<td>22650.00</td>

</tr>

<tr>

<th>2012-12-31 04:00:00</th>

<td>14845.10</td>

</tr>

<tr>

<th>2012-12-31 05:00:00</th>

<td>11449.70</td>

</tr>

<tr>

<th>2012-12-31 06:00:00</th>

<td>11637.20</td>

</tr>

<tr>

<th>2012-12-31 07:00:00</th>

<td>6465.35</td>

</tr>

</tbody>

</table>

</div>

```python

# transform data to be stationary

raw\_values = series.values

diff\_values = difference(raw\_values, 1)

```

```python

# transform data to be supervised learning

supervised = timeseries\_to\_supervised(diff\_values, lag\_exp)

supervised\_values = supervised.values

```

```python

# split data into train and test-sets

train, test = supervised\_values[0:-predict\_values\_exp], supervised\_values[-predict\_values\_exp:]

```

```python

# transform the scale of the data

scaler, train\_scaled, test\_scaled = scale(train, test)

```

```python

# fit the model

fit\_batch\_size\_exp = 419

lstm\_model = fit\_lstm(train\_scaled, fit\_batch\_size\_exp, epoch\_exp, neurons\_exp)

```

WARNING:tensorflow:From C:\Users\radha\anaconda3\envs\deeplearning\lib\site-packages\keras\backend\tensorflow\_backend.py:422: The name tf.global\_variables is deprecated. Please use tf.compat.v1.global\_variables instead.

Epoch 1/1

52375/52375 [==============================] - 3s 62us/step - loss: 0.0221

Epoch 1/1

52375/52375 [==============================] - 1s 21us/step - loss: 0.0153

Epoch 1/1

52375/52375 [==============================] - 1s 20us/step - loss: 0.0151

Epoch 1/1

52375/52375 [==============================] - 1s 19us/step - loss: 0.0150

Epoch 1/1

52375/52375 [==============================] - 1s 21us/step - loss: 0.0149

Epoch 1/1

52375/52375 [==============================] - 1s 22us/step - loss: 0.0148

Epoch 1/1

52375/52375 [==============================] - 1s 21us/step - loss: 0.0146

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0146

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0144

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0144

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0143

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0143

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0142

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0142

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0142

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0141

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0140

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0140

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0139

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0138

Epoch 1/1

52375/52375 [==============================] - 1s 19us/step - loss: 0.0137

Epoch 1/1

52375/52375 [==============================] - 1s 19us/step - loss: 0.0137

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0137

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0137

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0136

Epoch 1/1

52375/52375 [==============================] - 1s 18us/step - loss: 0.0137

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0136

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 16us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 17us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 16us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 15us/step - loss: 0.0135

Epoch 1/1

52375/52375 [==============================] - 1s 15us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 15us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 15us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 14us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 16us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 14us/step - loss: 0.0134

Epoch 1/1

52375/52375 [==============================] - 1s 14us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 13us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 15us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 14us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 13us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 12us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 12us/step - loss: 0.0132

Epoch 1/1

52375/52375 [==============================] - 1s 13us/step - loss: 0.0132: 0s -

Epoch 1/1

52375/52375 [==============================] - 1s 13us/step - loss: 0.0132

Epoch 1/1

52375/52375 [==============================] - 1s 12us/step - loss: 0.0133

Epoch 1/1

52375/52375 [==============================] - 1s 12us/step - loss: 0.0132

```python

# walk-forward validation on the test data

predictions = list()

expectations = list()

test\_pred = list()

for i in range(len(test\_scaled)):

# make one-step forecast

X, y = test\_scaled[i, 0:-1], test\_scaled[i, -1]

yhat = forecast\_lstm(lstm\_model, 1, X)#batch\_size\_exp to 1

'''# Start Debug prints

print("X: %", X)

print("yhat: %", yhat)

# End Debug prints'''

# Replacing value in test scaled with the predicted value.

test\_pred = [yhat] + test\_pred

if len(test\_pred) > lag\_exp+1:

test\_pred = test\_pred[:-1]

if i+1<len(test\_scaled):

if i+1 > lag\_exp+1:

test\_scaled[i+1] = test\_pred

else:

test\_scaled[i+1] = np.concatenate((test\_pred, test\_scaled[i+1, i+1:]),axis=0)

# invert scaling

yhat = invert\_scale(scaler, X, yhat)

# invert differencing

yhat = inverse\_difference(raw\_values, yhat, len(test\_scaled)+1-i)

# store forecast

expected = raw\_values[len(train) + i + 1]

if expected != 0:

predictions.append(yhat)

expectations.append(expected)

print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

```

Hour=1, Predicted=31949.045020, Expected=29145.800000

Hour=2, Predicted=28398.711870, Expected=29066.600000

Hour=3, Predicted=29495.763692, Expected=25478.700000

Hour=4, Predicted=25370.969938, Expected=18275.000000

Hour=5, Predicted=17381.112231, Expected=19510.200000

Hour=6, Predicted=17897.895820, Expected=19959.700000

Hour=7, Predicted=18480.312701, Expected=9201.150000

Hour=8, Predicted=8110.826898, Expected=4013.650000

Hour=9, Predicted=3421.550790, Expected=3423.460000

Hour=10, Predicted=2301.637474, Expected=5700.400000

Hour=11, Predicted=4726.862900, Expected=17196.400000

Hour=12, Predicted=16605.241360, Expected=9121.700000

Hour=13, Predicted=8652.808518, Expected=11736.800000

Hour=14, Predicted=10998.000500, Expected=11387.100000

Hour=15, Predicted=10655.450730, Expected=11611.200000

Hour=16, Predicted=11364.284564, Expected=15168.500000

Hour=17, Predicted=14964.937629, Expected=17830.000000

Hour=18, Predicted=17542.710267, Expected=14867.700000

Hour=19, Predicted=14851.178437, Expected=6515.390000

Hour=20, Predicted=6217.248274, Expected=4546.470000

Hour=21, Predicted=4736.598188, Expected=7461.770000

Hour=22, Predicted=7664.688980, Expected=5648.590000

Hour=23, Predicted=5877.306611, Expected=9508.370000

Hour=24, Predicted=9627.415223, Expected=0.000000

Hour=25, Predicted=163.972556, Expected=8662.060000

Hour=26, Predicted=8788.527424, Expected=33313.400000

Hour=27, Predicted=33478.162241, Expected=18847.400000

Hour=28, Predicted=18919.071275, Expected=21983.700000

Hour=29, Predicted=22041.158580, Expected=27665.600000

Hour=30, Predicted=27641.025448, Expected=25627.700000

Hour=31, Predicted=25642.633469, Expected=15047.900000

Hour=32, Predicted=15023.688144, Expected=6001.030000

Hour=33, Predicted=5936.802002, Expected=2831.900000

Hour=34, Predicted=2754.970301, Expected=3438.420000

Hour=35, Predicted=3333.501290, Expected=2174.210000

Hour=36, Predicted=2080.073887, Expected=3697.180000

Hour=37, Predicted=3581.710149, Expected=3698.550000

Hour=38, Predicted=3566.184690, Expected=2191.970000

Hour=39, Predicted=2081.467163, Expected=1260.400000

Hour=40, Predicted=1111.563770, Expected=0.000000

Hour=41, Predicted=-103.176566, Expected=0.000000

Hour=42, Predicted=-129.372407, Expected=0.000000

Hour=43, Predicted=-107.787415, Expected=6011.980000

Hour=44, Predicted=5903.655379, Expected=15081.800000

Hour=45, Predicted=14989.095382, Expected=18350.300000

Hour=46, Predicted=18256.546687, Expected=22881.600000

Hour=47, Predicted=22803.076644, Expected=29675.600000

Hour=48, Predicted=29592.045643, Expected=15974.300000

Hour=49, Predicted=15905.744553, Expected=13353.300000

Hour=50, Predicted=13279.926434, Expected=8196.580000

Hour=51, Predicted=8133.629805, Expected=8520.750000

Hour=52, Predicted=8454.476863, Expected=11346.600000

Hour=53, Predicted=11282.359653, Expected=12161.000000

Hour=54, Predicted=12099.886552, Expected=18754.700000

Hour=55, Predicted=18689.821006, Expected=13102.700000

Hour=56, Predicted=13040.465604, Expected=5068.350000

Hour=57, Predicted=5003.030955, Expected=3092.510000

Hour=58, Predicted=3025.424930, Expected=9189.220000

Hour=59, Predicted=9123.587346, Expected=19598.200000

Hour=60, Predicted=19526.491937, Expected=18850.800000

Hour=61, Predicted=18782.553773, Expected=26651.500000

Hour=62, Predicted=26577.208877, Expected=41222.500000

Hour=63, Predicted=41150.154311, Expected=44307.600000

Hour=64, Predicted=44232.255896, Expected=45091.100000

Hour=65, Predicted=45016.172272, Expected=54367.300000

Hour=66, Predicted=54290.401529, Expected=55807.500000

Hour=67, Predicted=55731.693801, Expected=56061.100000

Hour=68, Predicted=55983.195084, Expected=56544.300000

Hour=69, Predicted=56468.286736, Expected=56349.800000

Hour=70, Predicted=56271.997137, Expected=56311.400000

Hour=71, Predicted=56235.208551, Expected=56247.900000

Hour=72, Predicted=56170.999999, Expected=56152.000000

Hour=73, Predicted=56075.774058, Expected=22555.500000

Hour=74, Predicted=22479.802997, Expected=53470.800000

Hour=75, Predicted=53394.917465, Expected=56660.300000

Hour=76, Predicted=56585.346146, Expected=52066.200000

Hour=77, Predicted=51991.167668, Expected=57112.800000

Hour=78, Predicted=57038.134343, Expected=57622.700000

Hour=79, Predicted=57548.548690, Expected=57467.300000

Hour=80, Predicted=57392.824672, Expected=53316.500000

Hour=81, Predicted=53242.803131, Expected=56056.300000

Hour=82, Predicted=55982.111236, Expected=58362.600000

Hour=83, Predicted=58288.952218, Expected=58434.400000

Hour=84, Predicted=58360.437488, Expected=58554.200000

Hour=85, Predicted=58480.462718, Expected=58694.800000

Hour=86, Predicted=58620.813097, Expected=58575.200000

Hour=87, Predicted=58501.410467, Expected=53010.700000

Hour=88, Predicted=52936.533076, Expected=47433.300000

Hour=89, Predicted=47359.402699, Expected=42583.800000

Hour=90, Predicted=42509.483160, Expected=37533.400000

Hour=91, Predicted=37459.293185, Expected=30802.900000

Hour=92, Predicted=30728.522541, Expected=22118.300000

Hour=93, Predicted=22043.957035, Expected=19872.700000

Hour=94, Predicted=19798.294272, Expected=20219.700000

Hour=95, Predicted=20145.193342, Expected=22311.600000

Hour=96, Predicted=22237.141817, Expected=16743.400000

Hour=97, Predicted=16668.848234, Expected=11397.100000

Hour=98, Predicted=11322.565992, Expected=7571.350000

Hour=99, Predicted=7496.826707, Expected=7530.230000

Hour=100, Predicted=7455.655375, Expected=5941.390000

Hour=101, Predicted=5866.895996, Expected=6121.930000

Hour=102, Predicted=6047.372213, Expected=1487.690000

Hour=103, Predicted=1413.211815, Expected=0.000000

Hour=104, Predicted=-74.508699, Expected=725.355000

Hour=105, Predicted=650.900797, Expected=314.976000

Hour=106, Predicted=240.505775, Expected=0.000000

Hour=107, Predicted=-74.414198, Expected=0.000000

Hour=108, Predicted=-74.449815, Expected=0.000000

Hour=109, Predicted=-74.379296, Expected=0.000000

Hour=110, Predicted=-74.429506, Expected=0.000000

Hour=111, Predicted=-74.368377, Expected=0.000000

Hour=112, Predicted=-74.404197, Expected=0.000000

Hour=113, Predicted=-74.376337, Expected=0.000000

Hour=114, Predicted=-74.385317, Expected=0.000000

Hour=115, Predicted=-74.387563, Expected=1279.770000

Hour=116, Predicted=1205.389581, Expected=2610.950000

Hour=117, Predicted=2536.557845, Expected=2893.040000

Hour=118, Predicted=2818.649376, Expected=680.289000

Hour=119, Predicted=605.894498, Expected=1316.470000

Hour=120, Predicted=1242.066109, Expected=2730.490000

Hour=121, Predicted=2656.090599, Expected=8771.750000

Hour=122, Predicted=8697.338353, Expected=11282.000000

Hour=123, Predicted=11207.593558, Expected=13774.400000

Hour=124, Predicted=13699.984577, Expected=19978.700000

Hour=125, Predicted=19904.287434, Expected=23318.400000

Hour=126, Predicted=23243.981107, Expected=12301.800000

Hour=127, Predicted=12227.385802, Expected=5331.850000

Hour=128, Predicted=5257.426107, Expected=1479.590000

Hour=129, Predicted=1405.175700, Expected=0.000000

Hour=130, Predicted=-74.423179, Expected=1699.950000

Hour=131, Predicted=1625.535802, Expected=4932.380000

Hour=132, Predicted=4857.958760, Expected=10551.200000

Hour=133, Predicted=10476.785189, Expected=11277.200000

Hour=134, Predicted=11202.783046, Expected=20258.100000

Hour=135, Predicted=20183.682638, Expected=27235.300000

Hour=136, Predicted=27160.885495, Expected=29297.100000

Hour=137, Predicted=29222.684883, Expected=50140.200000

Hour=138, Predicted=50065.786516, Expected=28258.100000

Hour=139, Predicted=28183.686516, Expected=0.000000

Hour=140, Predicted=-74.412361, Expected=3868.210000

Hour=141, Predicted=3793.796924, Expected=0.000000

Hour=142, Predicted=-74.412974, Expected=0.000000

Hour=143, Predicted=-74.410116, Expected=0.000000

Hour=144, Predicted=-74.411545, Expected=4135.800000

Hour=145, Predicted=4061.388455, Expected=13607.200000

Hour=146, Predicted=13532.787741, Expected=24571.700000

Hour=147, Predicted=24497.288149, Expected=32106.600000

Hour=148, Predicted=32032.186414, Expected=36286.000000

Hour=149, Predicted=36211.588761, Expected=32040.800000

Hour=150, Predicted=31966.387026, Expected=27413.500000

Hour=151, Predicted=27339.088965, Expected=20005.800000

Hour=152, Predicted=19931.386516, Expected=19870.600000

Hour=153, Predicted=19796.187945, Expected=23296.700000

Hour=154, Predicted=23222.286720, Expected=36275.400000

Hour=155, Predicted=36200.987026, Expected=41119.700000

Hour=156, Predicted=41045.286822, Expected=45831.200000

Hour=157, Predicted=45756.787128, Expected=38451.600000

Hour=158, Predicted=38377.187128, Expected=38107.700000

Hour=159, Predicted=38033.287128, Expected=36325.100000

Hour=160, Predicted=36250.686924, Expected=37641.300000

Hour=161, Predicted=37566.886720, Expected=36876.500000

Hour=162, Predicted=36802.086720, Expected=35723.600000

Hour=163, Predicted=35649.186720, Expected=28221.500000

Hour=164, Predicted=28147.086720, Expected=22650.000000

Hour=165, Predicted=22575.586720, Expected=14845.100000

Hour=166, Predicted=14770.686312, Expected=11449.700000

Hour=167, Predicted=11375.287128, Expected=11637.200000

Hour=168, Predicted=11562.785904, Expected=6465.350000

```python

expectations = np.array(expectations)

predictions = np.array(predictions)

print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations - predictions) / expectations))\*100))

```

Mean Absolute Percent Error: 41.260067541788054

```python

```

```python

# line plot of observed vs predicted

pyplot.figure(figsize=(20,8))

pyplot.plot(expectations, label="True")

pyplot.plot(predictions, label="Predicted")

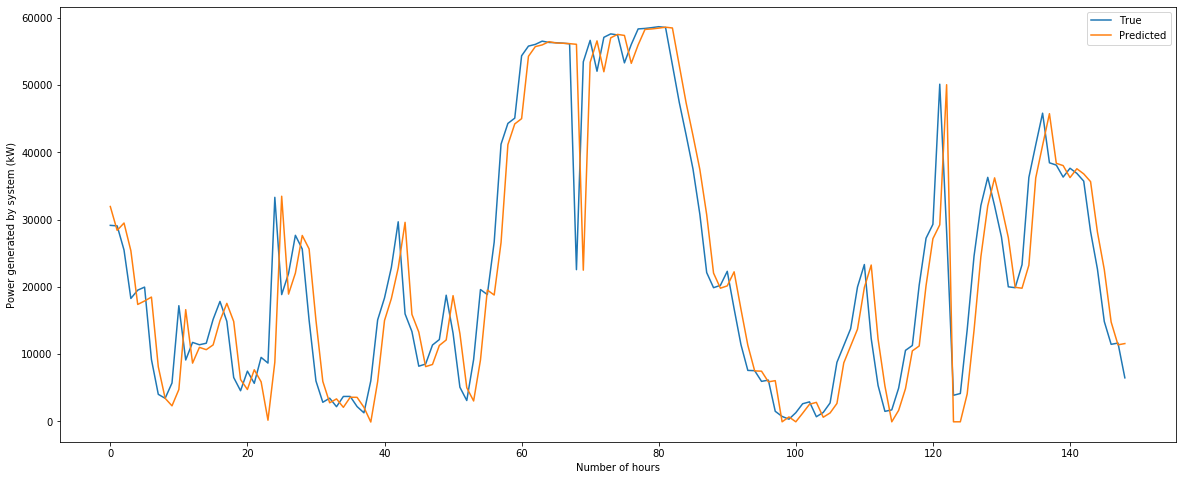
pyplot.legend(loc='upper right')

pyplot.xlabel("Number of hours")

pyplot.ylabel("Power generated by system (kW)")

pyplot.show()

```



```python

```

```python

```

```python

```

```python

```

```python

```

```python

```

```python

```