Electricity power consumption dataset

In [1]:

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

In [2]:

```
df=pd.read_csv(r'C:\Users\chumj\Desktop\household_power_consumption.csv',parse_dates=[['Date',
'Time']],index_col='Date_Time')
```

In [3]:

```
df.index
```

Out[3]:

```
DatetimeIndex(['2007-01-01 00:00:00', '2007-01-01 00:01:00',
               '2007-01-01 00:02:00', '2007-01-01 00:03:00',
               '2007-01-01 00:04:00', '2007-01-01 00:05:00',
               '2007-01-01 00:06:00', '2007-01-01 00:07:00',
               '2007-01-01 00:08:00', '2007-01-01 00:09:00',
               ...
               '2007-06-30 23:50:00', '2007-06-30 23:51:00',
               '2007-06-30 23:52:00', '2007-06-30 23:53:00',
               '2007-06-30 23:54:00', '2007-06-30 23:55:00',
               '2007-06-30 23:56:00', '2007-06-30 23:57:00',
               '2007-06-30 23:58:00', '2007-06-30 23:59:00'],
              dtype='datetime64[ns]', name='Date_Time', length=260640, freq=None)
```

In [4]:

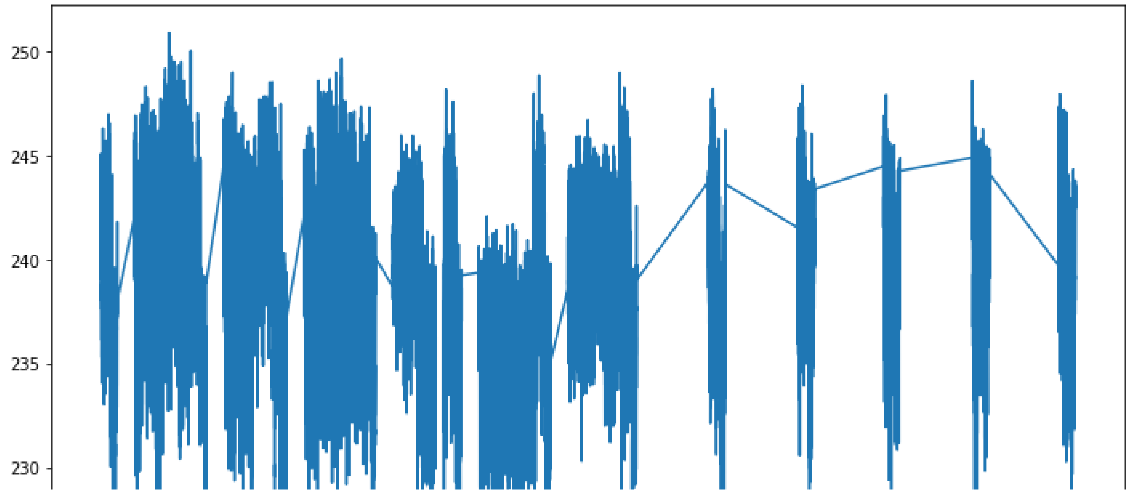
```
df["Voltage"] = pd.to_numeric(df.Voltage, errors='coerce')
df["Global_active_power"] = pd.to_numeric(df.Global_active_power, errors='coerce')
df["Global_reactive_power"] = pd.to_numeric(df.Global_reactive_power, errors='coerce')
df["Sub_metering_1"] = pd.to_numeric(df.Sub_metering_1, errors='coerce')
df["Sub_metering_2"] = pd.to_numeric(df.Sub_metering_2, errors='coerce')
df["Global_intensity"] = pd.to_numeric(df.Global_intensity, errors='coerce')
```

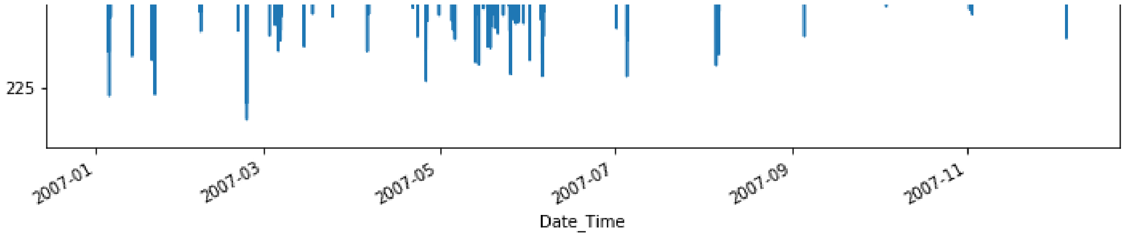
In [5]:

```
df['Voltage'].plot(figsize=(12,8))
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c131cbe88>
```





In [6]:

```
df.corr()
```

Out[6]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_
Global_active_power	1.000000	0.279084	-0.375375	0.998984	0.480525	0.470179	
Global_reactive_power	0.279084	1.000000	-0.101127	0.294772	0.159735	0.178309	
Voltage	-0.375375	-0.101127	1.000000	-0.386419	-0.217893	-0.174190	
Global_intensity	0.998984	0.294772	-0.386419	1.000000	0.485807	0.475781	
Sub_metering_1	0.480525	0.159735	-0.217893	0.485807	1.000000	0.073529	
Sub_metering_2	0.470179	0.178309	-0.174190	0.475781	0.073529	1.000000	
Sub_metering_3	0.609431	0.086682	-0.266190	0.598734	0.127195	0.116649	

In [7]:

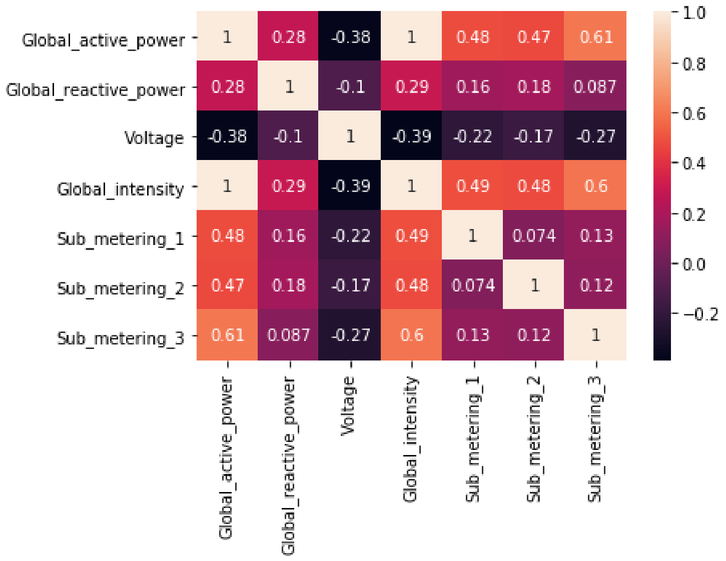
```
import seaborn as sns
```

In [8]:

```
sns.heatmap(df.corr(),annot=True)
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c1633bfc8>
```



In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 260640 entries, 2007-01-01 00:00:00 to 2007-06-30 23:59:00
Data columns (total 7 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   Global_active_power    256869 non-null  float64
 1   Global_reactive_power  256869 non-null  float64
 2   Voltage                256869 non-null  float64
 3   Global_intensity       256869 non-null  float64
 4   Sub_metering_1         256869 non-null  float64
 5   Sub_metering_2         256869 non-null  float64
 6   Sub_metering_3         256869 non-null  float64
dtypes: float64(7)
memory usage: 15.9 MB
```

In [10]:

```
df.describe().transpose()
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Global_active_power	256869.0	1.164937	1.181832	0.082	0.296	0.564	1.606	10.670
Global_reactive_power	256869.0	0.123729	0.111872	0.000	0.000	0.104	0.194	1.148
Voltage	256869.0	239.208981	3.592793	223.490	236.650	239.610	241.810	250.890
Global_intensity	256869.0	4.974755	4.999493	0.400	1.400	2.600	6.800	46.400
Sub_metering_1	256869.0	1.332481	6.704970	0.000	0.000	0.000	0.000	78.000
Sub_metering_2	256869.0	1.670610	6.631361	0.000	0.000	0.000	1.000	78.000
Sub_metering_3	256869.0	5.831825	8.186709	0.000	0.000	0.000	17.000	20.000

In [11]:

```
len(df)
```

Out[11]:

```
260640
```

In [12]:

```
df.tail()
```

Out[12]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
Date_Time							
2007-06-30 23:55:00	2.880	0.360	239.01	12.0	0.0	0.0	18.0
2007-06-30 23:56:00	2.892	0.358	238.86	12.2	0.0	0.0	17.0
2007-06-30 23:57:00	2.882	0.280	239.05	12.0	0.0	0.0	18.0
2007-06-30 23:58:00	2.660	0.290	238.98	11.2	0.0	0.0	18.0
2007-06-30 23:59:00	2.548	0.354	239.25	10.6	0.0	1.0	17.0

In [13]:

```
df.head()
```

Out[13]:

Date_Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2007-01-01 00:00:00	2.580	0.136	241.97	10.6	0.0	0.0	0.0
2007-01-01 00:01:00	2.552	0.100	241.75	10.4	0.0	0.0	0.0
2007-01-01 00:02:00	2.550	0.100	241.64	10.4	0.0	0.0	0.0
2007-01-01 00:03:00	2.550	0.100	241.71	10.4	0.0	0.0	0.0
2007-01-01 00:04:00	2.554	0.100	241.98	10.4	0.0	0.0	0.0

In [14]:

```
df1=df.loc['2007-06-30 13:00:00':]
```

In [15]:

```
df1
```

Out[15]:

Date_Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2007-06-30 13:00:00	0.238	0.096	240.50	1.0	0.0	1.0	0.0
2007-06-30 13:01:00	0.238	0.098	240.65	1.0	0.0	1.0	0.0
2007-06-30 13:02:00	0.238	0.100	241.24	1.0	0.0	1.0	0.0
2007-06-30 13:03:00	0.238	0.098	241.01	1.0	0.0	1.0	0.0
2007-06-30 13:04:00	0.240	0.100	241.65	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...
2007-06-30 23:55:00	2.880	0.360	239.01	12.0	0.0	0.0	18.0
2007-06-30 23:56:00	2.892	0.358	238.86	12.2	0.0	0.0	17.0
2007-06-30 23:57:00	2.882	0.280	239.05	12.0	0.0	0.0	18.0
2007-06-30 23:58:00	2.660	0.290	238.98	11.2	0.0	0.0	18.0
2007-06-30 23:59:00	2.548	0.354	239.25	10.6	0.0	1.0	17.0

660 rows × 7 columns

In [16]:

```
len(df1)
```

Out[16]:

```
660
```

In [17]:

```
test_ind=200
```

In [18]:

In [18]:

```
train=df1.iloc[:-test_ind]
test=df1.iloc[-test_ind:]
```

In [19]:

```
len(test)
```

Out[19]:

```
200
```

In [20]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [21]:

```
scaler= MinMaxScaler()
```

In [22]:

```
scaler.fit(train)
```

Out[22]:

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [23]:

```
scaled_train=scaler.transform(train)
scaled_test=scaler.transform(test)
```

In [24]:

```
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
```

```
C:\Users\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: F
utureWarning: 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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: F
utureWarning: 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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: F
```

```
C:\Users\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: F
utureWarning: 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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: F
utureWarning: 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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: F
utureWarning: 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\chumj\Anaconda3\Ben\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: F
utureWarning: 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)])
```

In [25]:

```
length=11
batch_size=1
generator=TimeseriesGenerator(scaled_train,scaled_train,length=length,batch_size=batch_size)
```

In [26]:

```
len(scaled_train)
```

Out[26]:

```
460
```

In [27]:

```
X,y=generator[0]
```

In [28]:

```
y
```

Out[28]:

```
array([[0.01222651, 0.2       , 0.87375   , 0.00763359, 0.        ,
        0.5       , 0.        ]])
```

In [29]:

```
scaled_train.shape
```

Out[29]:

```
(460, 7)
```

In [30]:

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

In [31]:

```
model=Sequential()
```

In [32]:

```
model.add(LSTM(100,input_shape=(length,scaled_train.shape[1])))
model.add(Dense(scaled_train.shape[1]))
model.compile(optimizer='adam',loss='mse')
model.summary()
```

```
WARNING:tensorflow:From C:\Users\chumj\Anaconda3\Ben\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from
tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 100)               43200     
_________________________________________________________________
dense (Dense)                (None, 7)                 707       
=================================================================
Total params: 43,907
Trainable params: 43,907
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```
#Earlystopping
from tensorflow.keras.callbacks import EarlyStopping
```

In [34]:

```
early_stop=EarlyStopping(monitor='val_loss',patience=45)
```

In [35]:

```
validation_generator=TimeseriesGenerator(scaled_test,scaled_test,length=length,batch_size=batch_siz
e)
```

In [36]:

```
model.fit_generator(generator,epochs=100,validation_data=validation_generator,callbacks=[early_stop
])
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\chumj\Anaconda3\Ben\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
449/449 [==============================] - 4s 9ms/step - loss: 0.0335 - val_loss: 0.0177
Epoch 2/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0208 - val_loss: 0.0144
Epoch 3/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0193 - val_loss: 0.0112
Epoch 4/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0174 - val_loss: 0.0149
Epoch 5/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0161 - val_loss: 0.0120
Epoch 6/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0148 - val_loss: 0.0110
Epoch 7/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0144 - val_loss: 0.0099
Epoch 8/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0134 - val_loss: 0.0116
Epoch 9/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0136 - val_loss: 0.0120
Epoch 10/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0132 - val_loss: 0.0104
Epoch 11/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0124 - val_loss: 0.0163
Epoch 12/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0125 - val_loss: 0.0110
Epoch 13/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0123 - val_loss: 0.0102
Epoch 14/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0122 - val_loss: 0.0105
Epoch 15/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0117 - val_loss: 0.0105
Epoch 16/100
Epoch 16/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0116 - val_loss: 0.0116
Epoch 17/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0115 - val_loss: 0.0126
Epoch 18/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0109 - val_loss: 0.0112
Epoch 19/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0110 - val_loss: 0.0104
Epoch 20/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0107 - val_loss: 0.0102
Epoch 21/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0104 - val_loss: 0.0098
Epoch 22/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0101 - val_loss: 0.0093
Epoch 23/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0099 - val_loss: 0.0101
Epoch 24/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0100 - val_loss: 0.0104
Epoch 25/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0095 - val_loss: 0.0099
Epoch 26/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0092 - val_loss: 0.0118
Epoch 27/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0092 - val_loss: 0.0109
Epoch 28/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0090 - val_loss: 0.0117
Epoch 29/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0088 - val_loss: 0.0117
Epoch 30/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0084 - val_loss: 0.0127
Epoch 31/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0083 - val_loss: 0.0106
Epoch 32/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0080 - val_loss: 0.0110
Epoch 33/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0078 - val_loss: 0.0132
Epoch 34/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0078 - val_loss: 0.0108
Epoch 35/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0072 - val_loss: 0.0107
Epoch 36/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0070 - val_loss: 0.0106
Epoch 37/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0071 - val_loss: 0.0107
Epoch 38/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0068 - val_loss: 0.0105
Epoch 39/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0067 - val_loss: 0.0126
Epoch 40/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0066 - val_loss: 0.0123
Epoch 41/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0063 - val_loss: 0.0122
Epoch 42/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0059 - val_loss: 0.0123
Epoch 43/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0057 - val_loss: 0.0118
Epoch 44/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0059 - val_loss: 0.0131
Epoch 45/100
449/449 [==============================] - 3s 8ms/step - loss: 0.0057 - val_loss: 0.0120
Epoch 46/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0055 - val_loss: 0.0128
Epoch 47/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0053 - val_loss: 0.0125
Epoch 48/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0049 - val_loss: 0.0125
Epoch 49/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0052 - val_loss: 0.0134
Epoch 50/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0049 - val_loss: 0.0127
Epoch 51/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0046 - val_loss: 0.0121
Epoch 52/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0047 - val_loss: 0.0136
Epoch 53/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0046 - val_loss: 0.0122
Epoch 54/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0046 - val_loss: 0.0133
```

```
449/449 [==============================] - 4s 8ms/step - loss: 0.0046 - val_loss: 0.0133
Epoch 55/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0045 - val_loss: 0.0132
Epoch 56/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0043 - val_loss: 0.0124
Epoch 57/100
449/449 [==============================] - 4s 8ms/step - loss: 0.0042 - val_loss: 0.0128
Epoch 58/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0043 - val_loss: 0.0138
Epoch 59/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0041 - val_loss: 0.0130
Epoch 60/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0040 - val_loss: 0.0124
Epoch 61/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0037 - val_loss: 0.0119
Epoch 62/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0039 - val_loss: 0.0130
Epoch 63/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0041 - val_loss: 0.0131
Epoch 64/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0037 - val_loss: 0.0123
Epoch 65/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0034 - val_loss: 0.0132
Epoch 66/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0037 - val_loss: 0.0121
Epoch 67/100
449/449 [==============================] - 4s 9ms/step - loss: 0.0034 - val_loss: 0.0124
```

Out[36]:

```
<tensorflow.python.keras.callbacks.History at 0x15c21cb3bc8>
```

In [37]:

```
model.history.history.keys()
```

Out[37]:

```
dict_keys(['loss', 'val_loss'])
```

In [38]:

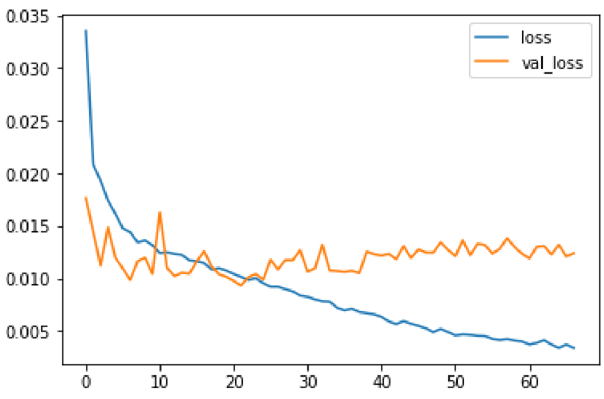
```
losses=pd.DataFrame(model.history.history)
```

In [39]:

```
losses.plot()
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c229a8c48>
```



In [40]:

```
#Evaluate on the Test Data
first_eval_batch=scaled_train[-length:]
```

In [41]:

```
first_eval_batch
```

Out[41]:

```
array([[0.23809524, 0.69795918, 0.62125   , 0.23664122, 0.        ,
        1.        , 0.94736842],
       [0.23712999, 0.69387755, 0.56      , 0.23664122, 0.025     ,
        0.5       , 0.89473684],
       [0.23777349, 0.69795918, 0.6075    , 0.23664122, 0.        ,
        0.5       , 0.94736842],
       [0.23648649, 0.68979592, 0.53875   , 0.22900763, 0.025     ,
        0.5       , 0.89473684],
       [0.23584299, 0.68571429, 0.49125   , 0.22900763, 0.        ,
        0.5       , 0.94736842],
       [0.23648649, 0.68979592, 0.5275    , 0.22900763, 0.025     ,
        0.5       , 0.89473684],
       [0.23777349, 0.70204082, 0.63375   , 0.23664122, 0.        ,
        0.5       , 0.94736842],
       [0.23680824, 0.69387755, 0.5725    , 0.22900763, 0.025     ,
        1.        , 0.89473684],
       [0.23680824, 0.69387755, 0.5825    , 0.22900763, 0.        ,
        0.5       , 0.94736842],
       [0.23745174, 0.69795918, 0.63375   , 0.22900763, 0.        ,
        0.5       , 0.94736842],
       [0.23745174, 0.70204082, 0.64375   , 0.22900763, 0.025     ,
        0.5       , 0.89473684]])
```

In [42]:

```
first_eval_batch=first_eval_batch.reshape((1,length,scaled_train.shape[1]))
```

In [43]:

```
first_eval_batch
```

Out[43]:

```
array([[[0.23809524, 0.69795918, 0.62125   , 0.23664122, 0.        ,
         1.        , 0.94736842],
        [0.23712999, 0.69387755, 0.56      , 0.23664122, 0.025     ,
         0.5       , 0.89473684],
        [0.23777349, 0.69795918, 0.6075    , 0.23664122, 0.        ,
         0.5       , 0.94736842],
        [0.23648649, 0.68979592, 0.53875   , 0.22900763, 0.025     ,
         0.5       , 0.89473684],
        [0.23584299, 0.68571429, 0.49125   , 0.22900763, 0.        ,
         0.5       , 0.94736842],
        [0.23648649, 0.68979592, 0.5275    , 0.22900763, 0.025     ,
         0.5       , 0.89473684],
        [0.23777349, 0.70204082, 0.63375   , 0.23664122, 0.        ,
         0.5       , 0.94736842],
        [0.23680824, 0.69387755, 0.5725    , 0.22900763, 0.025     ,
         1.        , 0.89473684],
        [0.23680824, 0.69387755, 0.5825    , 0.22900763, 0.        ,
         0.5       , 0.94736842],
        [0.23745174, 0.69795918, 0.63375   , 0.22900763, 0.        ,
         0.5       , 0.94736842],
        [0.23745174, 0.70204082, 0.64375   , 0.22900763, 0.025     ,
         0.5       , 0.89473684]]])
```

In [44]:

```
model.predict(first_eval_batch)
```

Out[44]:

```
array([[0.22277883, 0.6795678 , 0.6263764 , 0.21392551, 0.00556113,
        0.5768008 , 0.8857764 ]], dtype=float32)
```

In [45]:

```
scaled_test[0]
```

Out[45]:

```
array([0.23712999, 0.69795918, 0.605     , 0.22900763, 0.        ,
       0.5       , 0.94736842])
```

In [46]:

```
# now put this logic in a for loop to predict the future of the entire test range
n_features=scaled_train.shape[1]   # or 7
test_prediction=[]

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

for i in range(len(test)):
    #get prediction 1 time stamp ahead ([0]) is for grabbing just the number inside
    current_pred=model.predict(current_batch)[0]
    # store prediction
    test_prediction.append(current_pred)
    #update batch to now include prediction and drop first value
    current_batch=ny.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

In [47]:

```
test_prediction
```

Out[47]:

```
[array([0.22277883, 0.6795678 , 0.6263764 , 0.21392551, 0.00556113,
        0.5768008 , 0.8857764 ], dtype=float32),
 array([ 0.19131559,  0.66784406,  0.6359488 ,  0.19293219, -0.0545701 ,
         0.64945114,  0.872379  ], dtype=float32),
 array([ 0.1565428 ,  0.68394566,  0.6758579 ,  0.16726848, -0.12208983,
         0.6848943 ,  0.8765361 ], dtype=float32),
 array([ 0.11465757,  0.69708484,  0.71428466,  0.12140411, -0.18970206,
         0.67495435,  0.8683116 ], dtype=float32),
 array([ 0.03995019,  0.68861854,  0.75298226,  0.03298608, -0.32763174,
         0.5861645 ,  0.84383184], dtype=float32),
 array([-0.04127509,  0.6943927 ,  0.8257568 , -0.0477392 , -0.46288612,
         0.48114514,  0.8254717 ], dtype=float32),
 array([-0.12767443,  0.73050773,  0.945034  , -0.1168526 , -0.57036763,
         0.47746092,  0.7876618 ], dtype=float32),
 array([-0.19228522,  0.7839043 ,  1.0725664 , -0.16463041, -0.6220663 ,
         0.55872977,  0.7272058 ], dtype=float32),
 array([-0.22034241,  0.8047699 ,  1.1545378 , -0.17499343, -0.6068525 ,
         0.6461071 ,  0.6452997 ], dtype=float32),
 array([-0.21139343,  0.82628846,  1.1983749 , -0.16052496, -0.5621867 ,
         0.7285183 ,  0.5605302 ], dtype=float32),
 array([-0.18327464,  0.8496729 ,  1.2116617 , -0.14060792, -0.522484  ,
         0.76902324,  0.4733026 ], dtype=float32),
 array([-0.15043   ,  0.8689288 ,  1.1921139 , -0.11918243, -0.47454295,
         0.76386374,  0.39710444], dtype=float32),
 array([-0.11596902,  0.88110507,  1.1495459 , -0.09147375, -0.4008719 ,
         0.7410063 ,  0.33878863], dtype=float32),
 array([-0.08087216,  0.88383067,  1.0934714 , -0.05945795, -0.31351233,
         0.7274464 ,  0.29446423], dtype=float32),
 array([-0.0473676 ,  0.8856847 ,  1.0309443 , -0.02865169, -0.23219556,
         0.7221368 ,  0.26359755], dtype=float32),
 array([-0.02126922,  0.88673544,  0.9701512 , -0.00451002, -0.17323917,
         0.7228715 ,  0.24420948], dtype=float32),
 array([-0.0068184 ,  0.888427  ,  0.9190018 ,  0.00879005, -0.14541888,
         0.72515035,  0.23504895], dtype=float32),
 array([-0.00368305,  0.89121413,  0.88138497,  0.01046524, -0.14409643,
         0.7167571 ,  0.23395887], dtype=float32),
 array([-0.0088688 ,  0.89306474,  0.8567457 ,  0.00383135, -0.158842  ,
         0.6896694 ,  0.23924212], dtype=float32),
 array([-0.01751097,  0.890235  ,  0.84370804, -0.00430609, -0.17626485,
         0.6451212 ,  0.24940903], dtype=float32),
 array([-0.02712822,  0.88246024,  0.84421706, -0.0103092 , -0.18570682,
         0.60676   ,  0.2617587 ], dtype=float32),
 array([-0.03246666,  0.868073  ,  0.85677874, -0.01120026, -0.18316925,
         0.58897245,  0.27317312], dtype=float32),
 array([-0.03133063,  0.84973353,  0.87447727, -0.0067276 , -0.1717951 ,
         0.592513  ,  0.28315613], dtype=float32),
 array([-2.5814392e-02,  8.3203465e-01,  8.8918412e-01,  8.7484717e-04,
        -1.5931147e-01,  6.0973549e-01,  2.9214767e-01], dtype=float32),
 array([-0.02054894,  0.82229286,  0.8963809 ,  0.00644018, -0.15437564,
         0.62455463,  0.29963514], dtype=float32),
 array([-0.02156053,  0.8237706 ,  0.89581275,  0.00335145, -0.16322619,
         0.63523537,  0.30365092], dtype=float32),
 array([-0.0301698 ,  0.8333583 ,  0.89129055, -0.00941955, -0.18811294,
         0.63161254,  0.30431086], dtype=float32),
 array([-0.04459375,  0.84507626,  0.8870791 , -0.02558758, -0.22091225,
         0.604804  ,  0.30552906], dtype=float32),
 array([-0.06164411,  0.85487795,  0.88880587, -0.04054659, -0.24880394,
         0.5731513 ,  0.30834252], dtype=float32),
 array([-0.07370359,  0.8592434 ,  0.9008342 , -0.04928089, -0.26076895,
         0.55245984,  0.31160653], dtype=float32),
 array([-0.07583484,  0.8576362 ,  0.9195008 , -0.04816092, -0.2526721 ,
         0.5550972 ,  0.3135966 ], dtype=float32),
 array([-0.06726192,  0.8498896 ,  0.93719983, -0.03758451, -0.22968957,
         0.58441764,  0.3132115 ], dtype=float32),
 array([-0.05238953,  0.8397744 ,  0.94597375, -0.02178071, -0.20352569,
         0.62036884,  0.31059188], dtype=float32),
 array([-0.03921844,  0.8355886 ,  0.94492435, -0.00986737, -0.18616387,
         0.64434654,  0.30674434], dtype=float32),
 array([-0.03424151,  0.83785003,  0.93407905, -0.00873867, -0.18498188,
         0.6578337 ,  0.30068988], dtype=float32),
 array([-0.03809685,  0.8458523 ,  0.9189551 , -0.01718481, -0.20058003,
         0.6474464 ,  0.29471585], dtype=float32),
 array([-0.04763786,  0.8552589 ,  0.90485585, -0.02801045, -0.2230657 ,
         0.6151582 ,  0.29257584], dtype=float32),
 array([-0.05948254,  0.86202556,  0.897002  , -0.03722971, -0.24014455,
         0.5813817 ,  0.29370606], dtype=float32),
 array([-0.06682634,  0.8637641 ,  0.90003145, -0.04079951, -0.24247321,
         0.56142014,  0.29687166], dtype=float32),
 array([-0.06515785,  0.85898674,  0.9105158 , -0.03604309, -0.2286304 ,
         0.56724006,  0.29976445], dtype=float32),
 array([-0.05458051,  0.848494  ,  0.9216759 , -0.02409945, -0.20476568,
         0.59608567,  0.3012914 ], dtype=float32),
 array([-0.04013627,  0.8371947 ,  0.92706513, -0.00963203, -0.18220744,
         0.62555933,  0.30202842], dtype=float32),
 array([-0.02986054,  0.8332982 ,  0.9250016 , -0.00134308, -0.17164257,
         0.64150053,  0.3020386 ], dtype=float32),
 array([-0.02837066,  0.8357227 ,  0.9149617 , -0.00381327, -0.17759812,
         0.64879006,  0.29976243], dtype=float32),
 array([-0.03517385,  0.84373873,  0.90246785, -0.0145166 , -0.19805545,
         0.6328494 ,  0.29715645], dtype=float32),
 array([-0.04693377,  0.8521285 ,  0.8921586 , -0.02657825, -0.22243929,
         0.6005631 ,  0.29764277], dtype=float32),
 array([-0.05989242,  0.85831285,  0.89017606, -0.03653751, -0.23941562,
         0.56868786,  0.3008353 ], dtype=float32),
 array([-0.06700443,  0.8584968 ,  0.89808035, -0.0400363 , -0.24117368,
         0.55605626,  0.30456087], dtype=float32),
 array([-0.06442821,  0.85287136,  0.911958  , -0.03509606, -0.22729954,
         0.56905484,  0.3070739 ], dtype=float32),
 array([-0.05374727,  0.8428144 ,  0.92430425, -0.0232506 , -0.20506233,
         0.5997543 ,  0.3081305 ], dtype=float32),
 array([-0.04025079,  0.83366174,  0.9295157 , -0.01001345, -0.18584624,
         0.6272611 ,  0.30806875], dtype=float32),
 array([-0.03212243,  0.8322889 ,  0.926355  , -0.00403085, -0.17903188,
         0.6412885 ,  0.3065858 ], dtype=float32),
 array([-0.03231204,  0.8366014 ,  0.9162338 , -0.00844141, -0.18774813,
         0.64516   ,  0.30322444], dtype=float32),
 array([-0.03988217,  0.8452078 ,  0.9043714 , -0.01917142, -0.20833299,
         0.62511486,  0.30031836], dtype=float32),
 array([-0.05168284,  0.85326385,  0.8952515 , -0.03042059, -0.23020366,
         0.593945  ,  0.30044734], dtype=float32),
 array([-0.06293385,  0.85845757,  0.8951416 , -0.03852778, -0.24280718,
         0.5665983 ,  0.3030606 ], dtype=float32),
 array([-0.06748094,  0.85790205,  0.9036734 , -0.0396689 , -0.24021858,
         0.5594305 ,  0.30595958], dtype=float32),
 array([-0.06255428,  0.85160404,  0.9165946 , -0.03282102, -0.22358304,
         0.57775605,  0.30748075], dtype=float32),
 array([-0.05087665,  0.84172946,  0.92647815, -0.02030538, -0.20125219,
         0.6081005 ,  0.30777383], dtype=float32),
 array([-0.03777865,  0.8342132 ,  0.9294841 , -0.00813061, -0.1841338 ,
     0.6308666 ,  0.30748346], dtype=float32),
array([-0.03150896,  0.8337854 ,  0.92380273, -0.00432828, -0.18012717,
     0.6430769 ,  0.305331  ], dtype=float32),
array([-0.03317032,  0.83886045,  0.9128791 , -0.01016782, -0.19108862,
     0.64118266,  0.30205798], dtype=float32),
array([-0.04134733,  0.8471337 ,  0.9012518 , -0.02062548, -0.21158013,
     0.61804664,  0.3001667 ], dtype=float32),
array([-0.05312751,  0.8543531 ,  0.8936204 , -0.03114302, -0.23133981,
     0.5882518 ,  0.30089092], dtype=float32),
array([-0.06295177,  0.8581254 ,  0.895324  , -0.0378005 , -0.2406776 ,
     0.56548935,  0.3037058 ], dtype=float32),
array([-0.06568071,  0.85642296,  0.9046631 , -0.03731325, -0.23545176,
     0.56278163,  0.3065241 ], dtype=float32),
array([-0.05940805,  0.84931797,  0.91718054, -0.02953044, -0.21798524,
     0.5843835 ,  0.30780983], dtype=float32),
array([-0.04771978,  0.83972806,  0.9256526 , -0.01735179, -0.19726223,
     0.6130698 ,  0.30795658], dtype=float32),
array([-0.03619787,  0.83396256,  0.927304  , -0.00711326, -0.18338332,
     0.63200116,  0.30762452], dtype=float32),
array([-0.0318161 ,  0.83455   ,  0.9207022 , -0.00547708, -0.18268746,
     0.6417391 ,  0.30533418], dtype=float32),
array([-0.03491179,  0.8402439 ,  0.9099767 , -0.01237769, -0.19567284,
     0.6351667 ,  0.3023419 ], dtype=float32),
array([-0.04345373,  0.84801155,  0.8993906 , -0.02244657, -0.21565306,
     0.6102952 ,  0.3012467 ], dtype=float32),
array([-0.05491525,  0.8545797 ,  0.8937291 , -0.03212432, -0.23288968,
     0.5823422 ,  0.30244663], dtype=float32),
array([-0.06320646,  0.8571099 ,  0.89703476, -0.03733356, -0.23906675,
     0.56480944,  0.30513448], dtype=float32),
array([-0.06409851,  0.85441315,  0.9069282 , -0.03539865, -0.23145553,
     0.56756145,  0.30748916], dtype=float32),
array([-0.05682084,  0.847001  ,  0.9184927 , -0.02686075, -0.21372744,
     0.5908643 ,  0.30841964], dtype=float32),
array([-0.04534437,  0.83829266,  0.92532635, -0.01524203, -0.19499296,
     0.6171266 ,  0.30828634], dtype=float32),
array([-0.03553844,  0.83451366,  0.9255661 , -0.00701486, -0.18425885,
     0.63216656,  0.3077433 ], dtype=float32),
array([-0.03280398,  0.8360341 ,  0.91823304, -0.00722846, -0.18631688,
     0.6392857 ,  0.3053348 ], dtype=float32),
array([-0.03691592,  0.8420187 ,  0.90781844, -0.01466321, -0.20036829,
     0.6287663 ,  0.30258888], dtype=float32),
array([-0.04565045,  0.84912235,  0.8982384 , -0.02414598, -0.21901903,
     0.60382324,  0.30198324], dtype=float32),
array([-0.05621547,  0.85497886,  0.89467263, -0.03263425, -0.23335546,
     0.5779229 ,  0.30366218], dtype=float32),
array([-0.06281148,  0.856207  ,  0.89910865, -0.03622114, -0.23641285,
     0.5659395 ,  0.30614913], dtype=float32),
array([-0.06193861,  0.85256946,  0.9090229 , -0.03301186, -0.22686839,
     0.5734414 ,  0.3079626 ], dtype=float32),
array([-0.05399406,  0.84505   ,  0.91922474, -0.02405909, -0.2094172 ,
     0.5969212 ,  0.30862415], dtype=float32),
array([-0.0430476 ,  0.8374727 ,  0.9243512 , -0.01335458, -0.1930072 ,
     0.6200585 ,  0.30834588], dtype=float32),
array([-0.0351121 ,  0.83545214,  0.92318547, -0.00717325, -0.18551001,
     0.63096166,  0.30769563], dtype=float32),
array([-0.0338907 ,  0.8376817 ,  0.9153892 , -0.00896223, -0.18990278,
     0.6355291 ,  0.3053909 ], dtype=float32),
array([-0.03874204,  0.84365034,  0.9055679 , -0.01659879, -0.20436105,
     0.62191385,  0.30300695], dtype=float32),
array([-0.04766317,  0.8500666 ,  0.8971993 , -0.02552439, -0.22134328,
     0.5981878 ,  0.30280584], dtype=float32),
array([-0.05706055,  0.85481024,  0.8956094 , -0.03270123, -0.23283505,
     0.57573843,  0.30472744], dtype=float32),
array([-0.06195468,  0.85496587,  0.90100944, -0.03474484, -0.23326984,
     0.5682603 ,  0.3070268 ], dtype=float32),
array([-0.0596568 ,  0.8506205 ,  0.91074276, -0.03054724, -0.22251967,
     0.57925385,  0.30844015], dtype=float32),
array([-0.05146426,  0.84331805,  0.9195545 , -0.02162562, -0.20597464,
     0.6017063 ,  0.3088628 ], dtype=float32),
array([-0.04137663,  0.83707434,  0.92321754, -0.01215636, -0.19206604,
     0.621219  ,  0.30855682], dtype=float32),
array([-0.03519005,  0.83623964,  0.9207324 , -0.00783323, -0.18739367,
     0.6294198 ,  0.30769998], dtype=float32),
array([-0.03519627,  0.8390802 ,  0.91281855, -0.01075146, -0.19352743,
     0.63084483,  0.3055538 ], dtype=float32),
array([-0.04045215,  0.84475476,  0.9036764 , -0.01817594, -0.20769358,
     0.6152991 ,  0.3037218 ], dtype=float32),
 array([-0.04921648,  0.85055125,  0.89675105, -0.02640656, -0.22271514,
         0.5931828 ,  0.3039027 ], dtype=float32),
 array([-0.05735622,  0.8541817 ,  0.896695  , -0.03227437, -0.2314679 ,
         0.57505596,  0.30586156], dtype=float32),
 array([-0.0606931 ,  0.8535359 ,  0.9026277 , -0.03300514, -0.2298649 ,
         0.5713415 ,  0.30788052], dtype=float32),
 array([-0.05735742,  0.84883654,  0.91187   , -0.02814863, -0.21862033,
         0.5846539 ,  0.30896038], dtype=float32),
 array([-0.0492709 ,  0.8420508 ,  0.9193282 , -0.01965163, -0.20351923,
         0.60542834,  0.3091374 ], dtype=float32),
 array([-0.04029898,  0.8371662 ,  0.9217777 , -0.0115818 , -0.1921551 ,
         0.62117946,  0.3088443 ], dtype=float32),
 array([-0.03561503,  0.8370074 ,  0.9183656 , -0.00885885, -0.18973434,
         0.627725  ,  0.30780223], dtype=float32),
 array([-0.03657498,  0.8402009 ,  0.9106202 , -0.01246087, -0.19698691,
         0.6256215 ,  0.3058347 ], dtype=float32),
 array([-0.04190902,  0.8454116 ,  0.90228844, -0.01931455, -0.21027893,
         0.60914606,  0.3046761 ], dtype=float32),
 array([-0.05021125,  0.8506344 ,  0.8968779 , -0.02671074, -0.22309396,
         0.5891947 ,  0.30523914], dtype=float32),
 array([-0.05706899,  0.8532501 ,  0.8978816 , -0.03134771, -0.22935674,
         0.57539475,  0.30707967], dtype=float32),
 array([-0.05909224,  0.85201746,  0.9038911 , -0.031044  , -0.22632161,
         0.57496065,  0.30876523], dtype=float32),
 array([-0.05511512,  0.84729797,  0.9123918 , -0.02589723, -0.21524984,
         0.5894727 ,  0.30953878], dtype=float32),
 array([-0.04741643,  0.84128296,  0.9185958 , -0.01814707, -0.20201492,
         0.60817766,  0.30949613], dtype=float32),
 array([-0.0397529 ,  0.83769757,  0.92005444, -0.01153772, -0.19316897,
         0.6202092 ,  0.3092276 ], dtype=float32),
 array([-0.03637605,  0.8380272 ,  0.9161384 , -0.01012683, -0.19258383,
         0.6249542 ,  0.3081427 ], dtype=float32),
 array([-0.03799428,  0.8412434 ,  0.90883195, -0.01402317, -0.20026246,
         0.62013847,  0.30639824], dtype=float32),
 array([-0.04354948,  0.8460237 ,  0.9013294 , -0.02045175, -0.21233815,
         0.6046781 ,  0.30552095], dtype=float32),
 array([-0.05100787,  0.8507068 ,  0.8976673 , -0.02687313, -0.22308248,
         0.5864544 ,  0.30644852], dtype=float32),
 array([-0.05661649,  0.8523768 ,  0.8994266 , -0.03030778, -0.22727513,
         0.57696414,  0.30809832], dtype=float32),
 array([-0.05751118,  0.8506771 ,  0.9053774 , -0.02925936, -0.22325194,
         0.57913667,  0.30936456], dtype=float32),
 array([-0.05318092,  0.84612644,  0.91292834, -0.02407683, -0.2126551 ,
         0.5934427 ,  0.30991545], dtype=float32),
 array([-0.04606089,  0.8410127 ,  0.9178926 , -0.01714994, -0.20126274,
         0.6096823 ,  0.30971414], dtype=float32),
 array([-0.03968612,  0.83855814,  0.9184215 , -0.01187921, -0.19460857,
         0.6186106 ,  0.30942956], dtype=float32),
 array([-0.0373053 ,  0.8392206 ,  0.91425216, -0.01141514, -0.19530526,
         0.6217575 ,  0.3083673 ], dtype=float32),
 array([-0.0392559 ,  0.84228855,  0.90748477, -0.01527604, -0.20282266,
         0.6151535 ,  0.3068643 ], dtype=float32),
 array([-0.04472849,  0.8465967 ,  0.90096366, -0.02115589, -0.21340477,
         0.6008781 ,  0.30629668], dtype=float32),
 array([-0.05125283,  0.8504395 ,  0.89850116, -0.02655343, -0.22210649,
         0.5856637 ,  0.30732298], dtype=float32),
 array([-0.0556657 ,  0.85139155,  0.90068483, -0.02895034, -0.22466403,
         0.5793021 ,  0.30875278], dtype=float32),
 array([-0.05570342,  0.84939605,  0.9063586 , -0.02739307, -0.22014329,
         0.58334965,  0.30975398], dtype=float32),
 array([-0.05138655,  0.84520555,  0.91286814, -0.02248823, -0.21056536,
         0.59651136,  0.31014183], dtype=float32),
 array([-0.04509724,  0.8410244 ,  0.916734  , -0.01654848, -0.20115295,
         0.609994  ,  0.3098996 ], dtype=float32),
 array([-0.03998511,  0.83945125,  0.9165951 , -0.01254543, -0.19639993,
         0.6165138 ,  0.30961585], dtype=float32),
 array([-0.03836233,  0.8403163 ,  0.91253734, -0.01269987, -0.1979622 ,
         0.61787575,  0.30869806], dtype=float32),
 array([-0.04053622,  0.84315723,  0.9064554 , -0.01635913, -0.20492074,
         0.61074924,  0.30743486], dtype=float32),
 array([-0.04564176,  0.8469886 ,  0.90115523, -0.0215902 , -0.21390235,
         0.59782827,  0.30718866], dtype=float32),
 array([-0.05118087,  0.84994984,  0.8995769 , -0.02595835, -0.22069666,
         0.58608603,  0.30815238], dtype=float32),
 array([-0.05450867,  0.850361  ,  0.90196383, -0.0275177 , -0.22203839,
         0.5821878 ,  0.30929896], dtype=float32),
 array([-0.05396348,  0.84832174,  0.9071357 , -0.02567897, -0.21746036,
     0.5871136 ,  0.310081  ], dtype=float32),
array([-0.04989471,  0.84466636,  0.9125793 , -0.02128592, -0.20918924,
     0.59872574,  0.31030464], dtype=float32),
array([-0.04458001,  0.84139365,  0.9154757 , -0.01638779, -0.20168838,
     0.6094598 ,  0.31004575], dtype=float32),
array([-0.04057423,  0.84039694,  0.9149195 , -0.01349467, -0.19842494,
     0.61445135,  0.30972233], dtype=float32),
array([-0.03960371,  0.84141356,  0.9111762 , -0.01404988, -0.20049721,
     0.6141379 ,  0.308909  ], dtype=float32),
array([-0.04194325,  0.8440225 ,  0.9058759 , -0.01744433, -0.20674717,
     0.60736084,  0.3078605 ], dtype=float32),
array([-0.0464569 ,  0.8473733 ,  0.90188134, -0.02192921, -0.21422192,
     0.5957926 ,  0.30787006], dtype=float32),
array([-0.05100526,  0.8495504 ,  0.90100634, -0.02532687, -0.2193037 ,
     0.5872508 ,  0.30870968], dtype=float32),
array([-0.05336225,  0.84951806,  0.9034163 , -0.02622794, -0.21967852,
     0.58525705,  0.30952817], dtype=float32),
array([-0.05240577,  0.8475625 ,  0.90789914, -0.02423925, -0.2152088 ,
     0.5902858 ,  0.3100977 ], dtype=float32),
array([-0.04870604,  0.8444799 ,  0.912259  , -0.02037144, -0.20817083,
     0.60020214,  0.3102144 ], dtype=float32),
array([-0.04431595,  0.8420017 ,  0.9142668 , -0.0164264 , -0.20232537,
     0.608563  ,  0.30995074], dtype=float32),
array([-0.04123488,  0.84141344,  0.9133767 , -0.01443042, -0.20020804,
     0.6124109 ,  0.30958995], dtype=float32),
array([-0.04083657,  0.84255385,  0.9099808 , -0.01530427, -0.20256066,
     0.61106294,  0.30883762], dtype=float32),
array([-0.04316302,  0.84493166,  0.9056492 , -0.0183773 , -0.20811734,
     0.6043965 ,  0.30810687], dtype=float32),
array([-0.04707447,  0.84769714,  0.90272784, -0.02210449, -0.21422997,
     0.5947644 ,  0.3082726 ], dtype=float32),
array([-0.05069292,  0.8491749 ,  0.90240765, -0.02468083, -0.21787217,
     0.5887044 ,  0.30893365], dtype=float32),
array([-0.05223463,  0.8488024 ,  0.9047222 , -0.02503393, -0.21746507,
     0.58801585,  0.30952626], dtype=float32),
array([-0.05101612,  0.8469895 ,  0.9084867 , -0.02300117, -0.2132433 ,
     0.5928312 ,  0.3099267 ], dtype=float32),
array([-0.04774928,  0.8444337 ,  0.9118563 , -0.01965634, -0.2073693 ,
     0.6011303 ,  0.30999506], dtype=float32),
array([-0.0441693 ,  0.8425883 ,  0.91313803, -0.01656745, -0.20289832,
     0.6076506 ,  0.30974305], dtype=float32),
array([-0.0418793 ,  0.8423308 ,  0.9120445 , -0.01529799, -0.20170248,
     0.61033344,  0.30938047], dtype=float32),
array([-0.04192268,  0.8435517 ,  0.909042  , -0.01636575, -0.20421427,
     0.60822165,  0.30875152], dtype=float32),
array([-0.04416142,  0.8456993 ,  0.905596  , -0.01909765, -0.20909208,
     0.60195416,  0.30829862], dtype=float32),
array([-0.04748909,  0.8477777 ,  0.90346026, -0.02213115, -0.21393451,
     0.5946016 ,  0.30847767], dtype=float32),
array([-0.05026869,  0.84869736,  0.9035536 , -0.02401798, -0.21643242,
     0.5902105 ,  0.30897748], dtype=float32),
array([-0.05116889,  0.8481228 ,  0.9057287 , -0.02394565, -0.2154941 ,
     0.59040487,  0.309462  ], dtype=float32),
array([-0.04982628,  0.8464637 ,  0.9088628 , -0.02198546, -0.21164495,
     0.5949259 ,  0.3097362 ], dtype=float32),
array([-0.04703318,  0.84441245,  0.91136646, -0.01916342, -0.20688203,
     0.60154456,  0.30978274], dtype=float32),
array([-0.04414951,  0.843043  ,  0.9121258 , -0.01683308, -0.20352942,
     0.60679996,  0.3095551 ], dtype=float32),
array([-0.04253338,  0.8430536 ,  0.9109291 , -0.01611531, -0.20305347,
     0.60832685,  0.30918604], dtype=float32),
array([-0.04289683,  0.8443392 ,  0.9083712 , -0.01726675, -0.20559531,
     0.6056058 ,  0.3087343 ], dtype=float32),
array([-0.044993  ,  0.8462032 ,  0.9056872 , -0.01964839, -0.20977992,
     0.6001854 ,  0.30849537], dtype=float32),
array([-0.04775237,  0.84773195,  0.9041877 , -0.02207337, -0.21352652,
     0.5946932 ,  0.30863947], dtype=float32),
array([-0.04981554,  0.84823996,  0.9045594 , -0.02338298, -0.21511155,
     0.5916704 ,  0.3090103 ], dtype=float32),
array([-0.05022588,  0.84755826,  0.9065422 , -0.02301715, -0.21385774,
     0.592505  ,  0.30938572], dtype=float32),
array([-0.04885346,  0.84606504,  0.90910554, -0.02120098, -0.21044037,
     0.59664655,  0.3095624 ], dtype=float32),
array([-0.04647452,  0.8444    ,  0.910944  , -0.01887808, -0.20661443,
     0.6020246 ,  0.3095698 ], dtype=float32),
array([-0.0442193 ,  0.8434388 ,  0.9112718 , -0.01715475, -0.204207  ,
     0.60589117,  0.30935556], dtype=float32),
array([-0.04315784,  0.84363836,  0.91002035, -0.01683799, -0.20427087,
      0.606499  ,  0.30902678], dtype=float32),
array([-0.04373446,  0.8448215 ,  0.9078077 , -0.01799791, -0.20668226,
      0.603767  ,  0.30865806], dtype=float32),
array([-0.04562567,  0.8463398 ,  0.9057034 , -0.02003225, -0.2101619 ,
      0.5991892 ,  0.30853263], dtype=float32),
array([-0.0478584 ,  0.8474869 ,  0.9047301 , -0.02192171, -0.21300575,
      0.59491307,  0.30870622], dtype=float32),
array([-0.049357  ,  0.84772533,  0.90528786, -0.02277925, -0.2139351 ,
      0.5929432 ,  0.3090223 ], dtype=float32),
array([-0.049436  ,  0.84701735,  0.90705764, -0.02225135, -0.21256319,
      0.59417146,  0.3093113 ], dtype=float32),
array([-0.04812327,  0.8457213 ,  0.9091277 , -0.02063978, -0.2096155 ,
      0.59785575,  0.30943555], dtype=float32),
array([-0.04613233,  0.8444173 ,  0.91046786, -0.01876838, -0.20660192,
      0.6021725 ,  0.30941254], dtype=float32),
array([-0.04440647,  0.84379023,  0.9105195 , -0.01752803, -0.20494133,
      0.6049419 ,  0.30921662], dtype=float32),
array([-0.04375804,  0.844117  ,  0.90931916, -0.01748303, -0.20535296,
      0.6049475 ,  0.30892527], dtype=float32),
array([-0.04444131,  0.8451787 ,  0.90746963, -0.01859082, -0.20754501,
      0.6023599 ,  0.3086357 ], dtype=float32),
array([-0.0460946 ,  0.8464056 ,  0.9058604 , -0.02029294, -0.21038234,
      0.5985518 ,  0.30858117], dtype=float32),
array([-0.04786758,  0.84724456,  0.90526986, -0.02173363, -0.21249163,
      0.59528834,  0.30875188], dtype=float32),
array([-0.04891886,  0.8472987 ,  0.9059156 , -0.0222469 , -0.21293253,
      0.5941131 ,  0.3090114 ], dtype=float32),
array([-0.04877359,  0.84660727,  0.9074508 , -0.02163447, -0.21153748,
      0.5955248 ,  0.3092262 ], dtype=float32),
array([-0.04756168,  0.8455079 ,  0.9090884 , -0.0202341 , -0.20903799,
      0.5987306 ,  0.30930966], dtype=float32),
array([-0.04591945,  0.844508  ,  0.9100268 , -0.0187495 , -0.20669758,
      0.6021348 ,  0.30926624], dtype=float32),
array([-0.04462375,  0.8441296 ,  0.909886  , -0.01788236, -0.20560405,
      0.6040523 ,  0.30909163], dtype=float32),
array([-0.04427591,  0.84452164,  0.9087765 , -0.0180152 , -0.20621806,
      0.6036685 ,  0.30884436], dtype=float32),
array([-0.04499613,  0.84544647,  0.9072546 , -0.0190363 , -0.2081596 ,
      0.6013034 ,  0.30863148], dtype=float32),
array([-0.04640986,  0.8464242 ,  0.9060503 , -0.02043513, -0.21043995,
      0.5981777 ,  0.30862498], dtype=float32),
array([-0.04779799,  0.847007  ,  0.9057318 , -0.02151293, -0.21197468,
      0.59573984,  0.3087799 ], dtype=float32),
array([-0.04850601,  0.84693706,  0.9064063 , -0.02177736, -0.212075  ,
      0.5951387 ,  0.30899352], dtype=float32),
array([-0.04822244,  0.84629166,  0.90771663, -0.02114079, -0.21073273,
      0.59658957,  0.3091517 ], dtype=float32),
array([-0.04713778,  0.84537536,  0.90899336, -0.01994837, -0.20865116,
      0.59932333,  0.30920643], dtype=float32)]
```

In [48]:

```
scaled_test
```

Out[48]:

```
array([[0.23712999, 0.69795918, 0.605     , ..., 0.        , 0.5       ,
        0.94736842],
       [0.22393822, 0.46530612, 0.59375   , ..., 0.025     , 0.5       ,
        0.89473684],
       [0.22168597, 0.42857143, 0.6075    , ..., 0.        , 1.        ,
        0.94736842],
       ...,
       [0.43790219, 0.57142857, 0.54125   , ..., 0.        , 0.        ,
        0.94736842],
       [0.4021879 , 0.59183673, 0.5325    , ..., 0.        , 0.        ,
        0.94736842],
       [0.38416988, 0.72244898, 0.56625   , ..., 0.        , 0.5       ,
        0.89473684]])
```

In [49]:

```
#Inverse transformation and compare
true_predictions=scaler.inverse_transform(test_prediction)
```

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

In [50]:

```
true_predictions
```

Out[50]:

```
array([[ 1.54479319e+00,  3.32988229e-01,  2.39731011e+02, ...,
         2.22445130e-01,  1.15360165e+00,  1.68297516e+01],
       [ 1.34921772e+00,  3.27243588e-01,  2.39807590e+02, ...,
        -2.18280405e+00,  1.29890227e+00,  1.65752011e+01],
       [ 1.13307000e+00,  3.35133371e-01,  2.40126863e+02, ...,
        -4.88359302e+00,  1.36978865e+00,  1.66541854e+01],
       ...,
       [-1.41513337e-01,  4.14999160e-01,  2.41971250e+02, ...,
        -8.48299980e+00,  1.19027746e+00,  5.87087685e+00],
       [-1.39750672e-01,  4.14682914e-01,  2.41981733e+02, ...,
        -8.42930913e+00,  1.19317913e+00,  5.87388247e+00],
       [-1.33008453e-01,  4.14233926e-01,  2.41991947e+02, ...,
        -8.34604621e+00,  1.19864666e+00,  5.87492210e+00]])
```

In [51]:

```
test
```

Out[51]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
Date_Time							
2007-06-30 20:40:00	1.634	0.342	239.56	6.8	0.0	1.0	18.0
2007-06-30 20:41:00	1.552	0.228	239.47	6.4	1.0	1.0	17.0
2007-06-30 20:42:00	1.538	0.210	239.58	6.4	0.0	2.0	18.0
2007-06-30 20:43:00	1.540	0.210	240.01	6.4	1.0	1.0	18.0
2007-06-30 20:44:00	1.530	0.190	240.23	6.4	0.0	1.0	17.0
...	...	...	...	...	...	...	...
2007-06-30 23:55:00	2.880	0.360	239.01	12.0	0.0	0.0	18.0
2007-06-30 23:56:00	2.892	0.358	238.86	12.2	0.0	0.0	17.0
2007-06-30 23:57:00	2.882	0.280	239.05	12.0	0.0	0.0	18.0
2007-06-30 23:58:00	2.660	0.290	238.98	11.2	0.0	0.0	18.0
2007-06-30 23:59:00	2.548	0.354	239.25	10.6	0.0	1.0	17.0

200 rows × 7 columns

In [52]:

```
forecast_index=pd.date_range(start='2007-06-30 23:59:00',periods=200,freq='T')
```

In [53]:

```
forecast_index
```

Out[53]:

```
DatetimeIndex(['2007-06-30 23:59:00', '2007-07-01 00:00:00',
               '2007-07-01 00:01:00', '2007-07-01 00:02:00',
               '2007-07-01 00:03:00', '2007-07-01 00:04:00',
               '2007-07-01 00:05:00', '2007-07-01 00:06:00',
               '2007-07-01 00:07:00', '2007-07-01 00:08:00',
               ...
               '2007-07-01 03:09:00', '2007-07-01 03:10:00',
               '2007-07-01 03:11:00', '2007-07-01 03:12:00',
               '2007-07-01 03:13:00', '2007-07-01 03:14:00',
               '2007-07-01 03:15:00', '2007-07-01 03:16:00',
               '2007-07-01 03:17:00', '2007-07-01 03:18:00'],
              dtype='datetime64[ns]', length=200, freq='T')
```

In [54]:

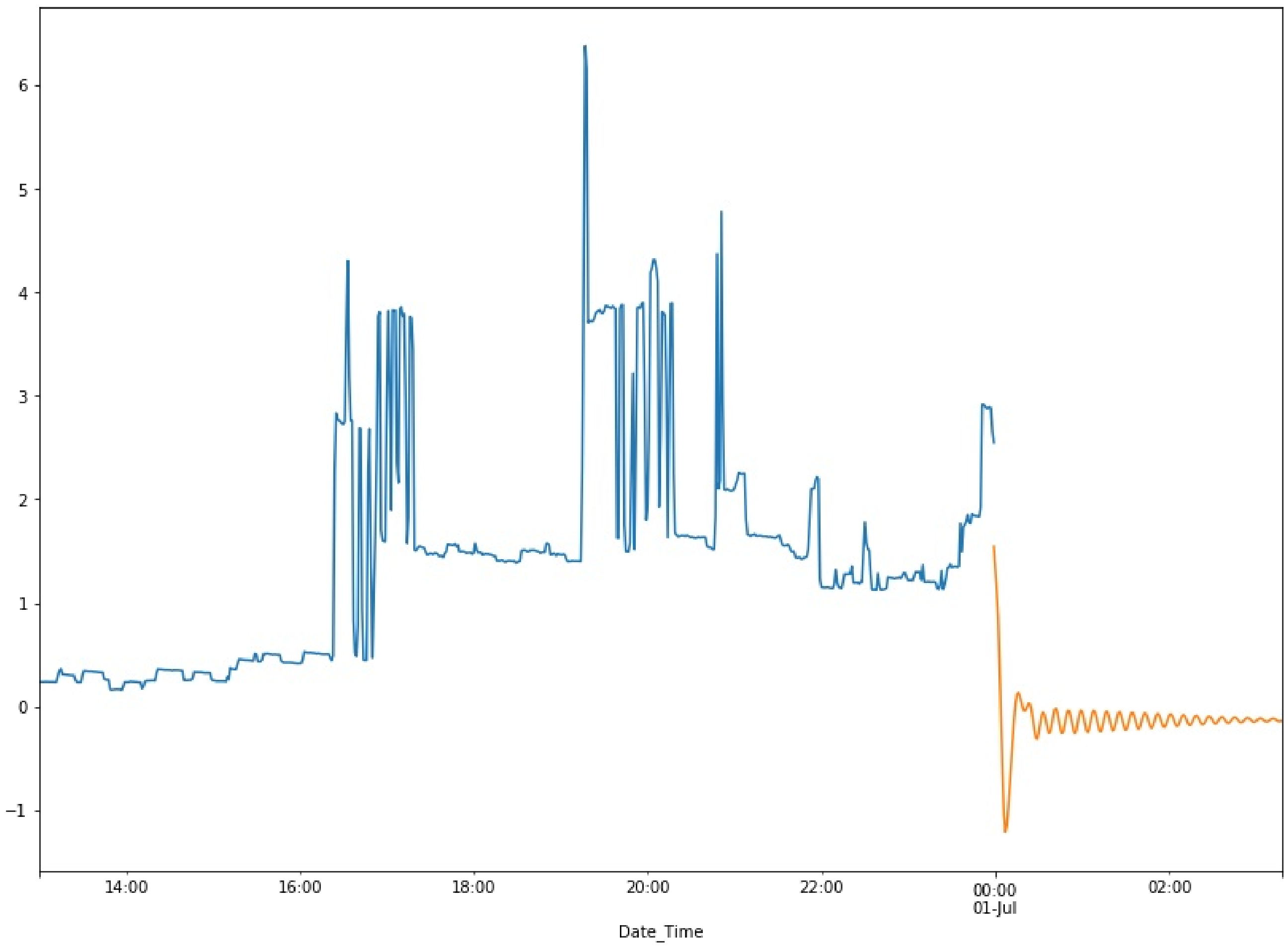
```
true_predictions=pd.DataFrame(data=true_predictions,columns=test.columns,index=forecast_index)
```

In [55]:

```
df1['Global_active_power'].plot(figsize=(14,10))
true_predictions['Global_active_power'].plot()
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c249d7d88>
```

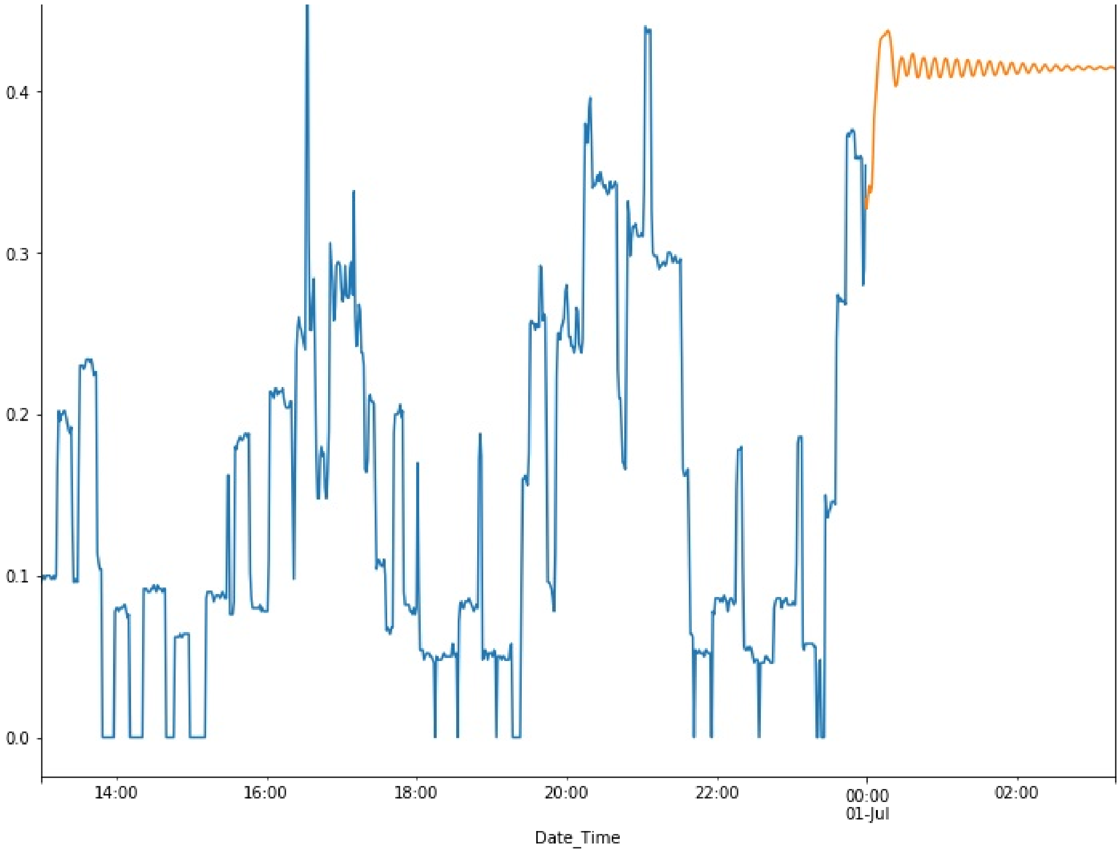


In [56]:

```
df1['Global_reactive_power'].plot(figsize=(12,10),legend=True)
true_predictions['Global_reactive_power'].plot(legend=True)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c24a84248>
```

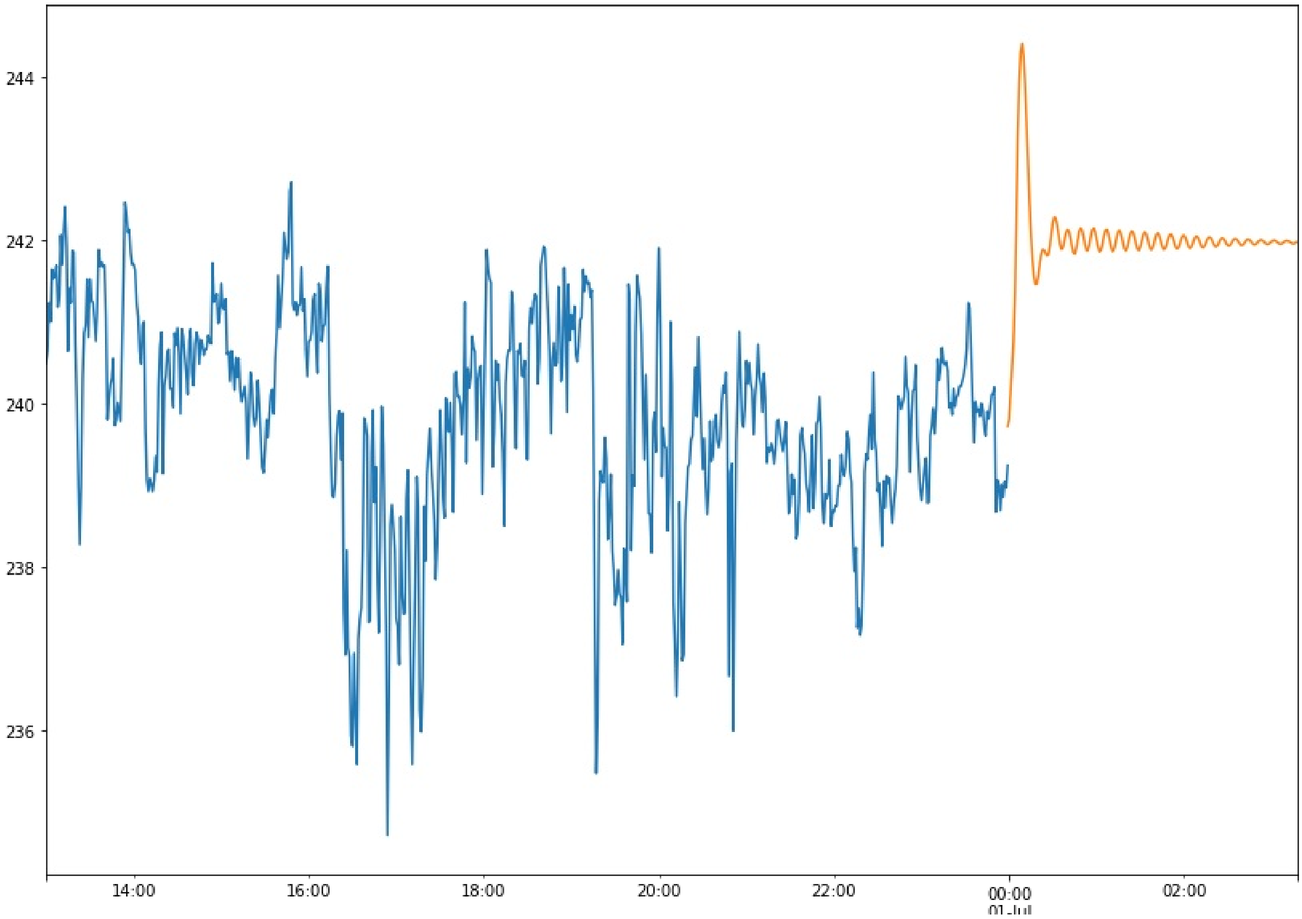


In [57]:

```
df1['Voltage'].plot(figsize=(14,10))
true_predictions['Voltage'].plot()
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c24afcac8>
```



01-Jul

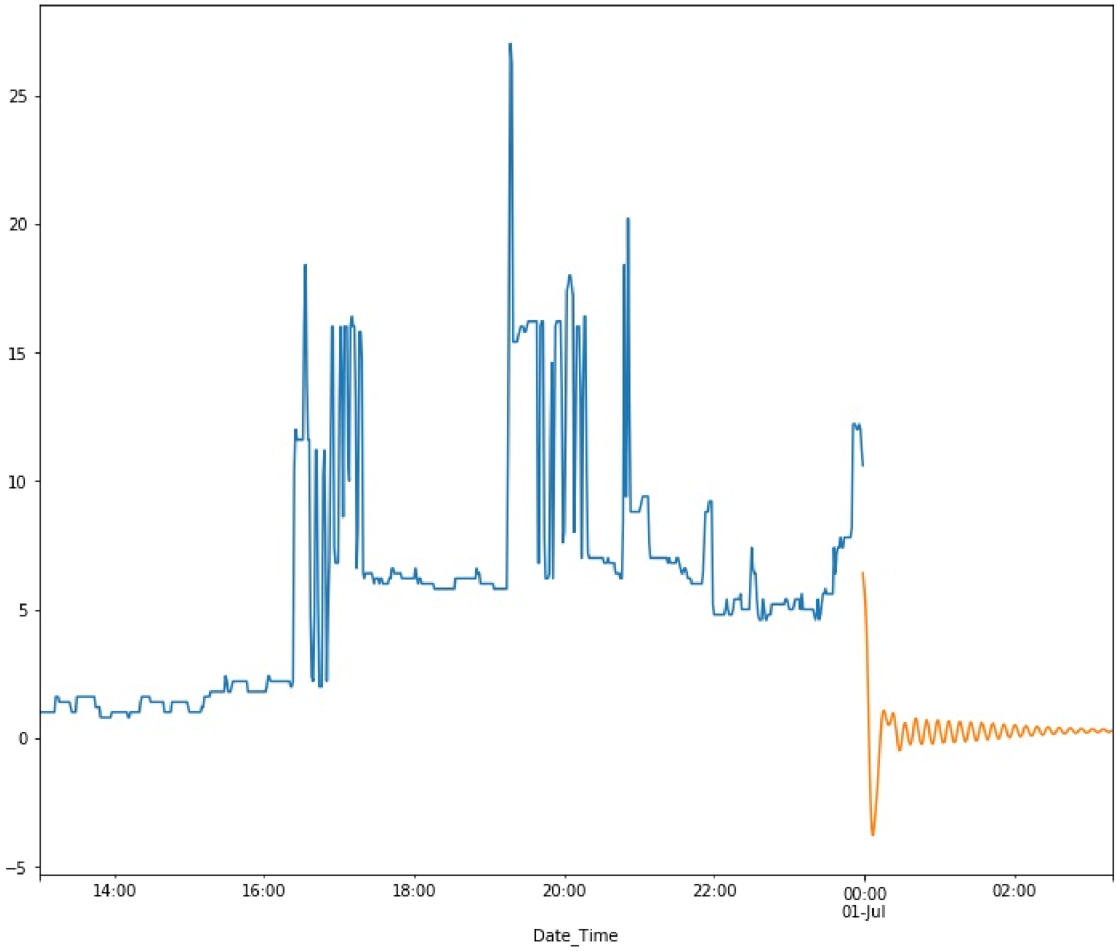
Date_Time

In [58]:

```
df1['Global_intensity'].plot(figsize=(12,10))
true_predictions['Global_intensity'].plot()
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15c24d401c8>
```



In [59]:

```
from sklearn import metrics
```

In [60]:

```
metrics.mean_absolute_error(test,true_predictions)

ny.sqrt(metrics.mean_squared_error(test,true_predictions))
```

Out[60]:

```
5.6012039159631355
```

In [61]:

```
metrics.mean_squared_error(test,true_predictions)
```

Out[61]:

```
31.37348530820076
```

In [62]:

In [62]:

```
metrics.mean_absolute_error(test,true_predictions)
```

Out[62]:

```
4.087946937365575
```

In [ ]:

```

```

In [ ]:

```

```

In [64]:

```
print('MAE:',metrics.mean_absolute_error(test,true_predictions))
print('MSE:',metrics.mean_squared_error(test,true_predictions))
print('RMSE:',ny.sqrt(metrics.mean_squared_error(test,true_predictions)))
```

```
MAE: 4.087946937365575
MSE: 31.37348530820076
RMSE: 5.6012039159631355
```

In [65]:

```
test.describe().transpose()
```

Out[65]:

	count	mean	std	min	25%	50%	75%	max
Global_active_power	200.0	1.59338	0.525927	1.128	1.2235	1.450	1.7685	4.776
Global_reactive_power	200.0	0.16808	0.122290	0.000	0.0575	0.128	0.2940	0.440
Voltage	200.0	239.48560	0.786411	236.000	239.0175	239.565	240.0825	241.240
Global_intensity	200.0	6.67800	2.247481	4.600	5.0000	6.000	7.4000	20.200
Sub_metering_1	200.0	0.01000	0.099748	0.000	0.0000	0.000	0.0000	1.000
Sub_metering_2	200.0	0.35500	0.557250	0.000	0.0000	0.000	1.0000	2.000
Sub_metering_3	200.0	7.74000	8.732335	0.000	0.0000	0.000	17.0000	18.000

In [ ]:

```

```

In [ ]:

```

```