In [1]:
```
## These are the liberaries used to run LSTM Model.
## Keras lib provides accruate results in Machine Learning. LSTM is the predictive model
## MinMaxScaler is for transform the attribute values in between 0 and 1
## RMSE is used for evaluate the error

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [2]:
```
## Reading the household Power consumption file, which is already cleaned with missing values.
## Choosing index column as date_time because it is Time series data set
## Its is having dates so parse dates is true
## Its a large file so low memory is false

df = pd.read_csv('Preprocessed data.csv', parse_dates= True, index_col = 'date_time', low_memory = Fa
lse )
```

In [3]:
```
## head displays the of the data set.

df.head()
```

Out[3]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
date_time							
2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	
2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	
2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	
2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	
2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	

In [4]:
```
df.tail()
```

Out[4]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
date_time							
2010-11-26 20:58:00	0.946	0.0	240.43	4.0	0.0	0.0	
2010-11-26 20:59:00	0.944	0.0	240.00	4.0	0.0	0.0	
2010-11-26 21:00:00	0.938	0.0	239.82	3.8	0.0	0.0	
2010-11-26 21:01:00	0.934	0.0	239.70	3.8	0.0	0.0	
2010-11-26 21:02:00	0.932	0.0	239.55	3.8	0.0	0.0	

In [5]:
```
df.tail()
```

Out[5]:

date_time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2010-11-26 20:58:00	0.946	0.0	240.43	4.0	0.0	0.0	
2010-11-26 20:59:00	0.944	0.0	240.00	4.0	0.0	0.0	
2010-11-26 21:00:00	0.938	0.0	239.82	3.8	0.0	0.0	
2010-11-26 21:01:00	0.934	0.0	239.70	3.8	0.0	0.0	
2010-11-26 21:02:00	0.932	0.0	239.55	3.8	0.0	0.0	

In [6]:
```
df.shape
```

Out[6]: (2075259, 7)

Preparing LSTM MODEL

In [7]:
```
## This involves framing the dataset as supervised learning problem and normalizing the input variabl
es
## Framing the supervised learning problem as predicting the power consumption at current hour (t)
## Predicting the power consumption for the next hour based on the power consumption over last 24 hrs
```

In [8]:
```
## Convert series to supervised learning

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()

    ## input sequence (t-n,.....t-1)

    for i in range(n_in, 0, -1):
        cols.append(df.shift(-i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

    ## forecast sequence (t, t+1,....t+n)

    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i==0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1)) for j in range(n_vars)]

    ## put it all together

        agg = pd.concat(cols, axis=1)
        agg.columns = names

    ## Drop rows with NAN values

        if dropnan:
            agg.dropna(inplace=True)
        return agg
```

In [9]:
```
df_resample = df.resample('h').mean()
df_resample.shape
```

Out[9]: (34951, 7)

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

Out[10]:

date_time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	
2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	
2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	
2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	
2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	

In [11]:
```
df.tail()
```

Out[11]:

date_time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2010-11-26 20:58:00	0.946	0.0	240.43	4.0	0.0	0.0	
2010-11-26 20:59:00	0.944	0.0	240.00	4.0	0.0	0.0	
2010-11-26 21:00:00	0.938	0.0	239.82	3.8	0.0	0.0	
2010-11-26 21:01:00	0.934	0.0	239.70	3.8	0.0	0.0	
2010-11-26 21:02:00	0.932	0.0	239.55	3.8	0.0	0.0	

In [12]:
```
## values returns a list of all the values in the data frame
values = df_resample.values
```

In [13]:
```
## Normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [14]:
```
## calculating scaler using fit_transform fn for values and assigning it as scaled
scaled = scaler.fit_transform(values)
```

In [15]:
```
## Called series_to_supervised fn to make it as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
```

In [16]:
```
## Droping the other columns because we are Predicting only for Global active power
reframed.drop(reframed.columns[[8,9,10,11,12,13]], axis=1, inplace=True)
```

In [17]:
```
## Reframed head
## This prints the first five rows of transformed dataset. We can see the 8 input variables( input se
ries )
## The 1 output variable at the current hour that is var1(t)

reframed.head()
```

Out[17]:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var1(t)
0	0.545045	0.103358	0.335501	0.541487	0.0	0.144652	0.782676	0.636816
1	0.509006	0.110073	0.283802	0.502152	0.0	0.030869	0.774169	0.545045
2	0.488550	0.096987	0.315987	0.481110	0.0	0.000000	0.778809	0.509006
3	0.455597	0.099010	0.434417	0.449904	0.0	0.008973	0.798917	0.488550
4	0.322555	0.072536	0.495847	0.323529	0.0	0.002872	0.205723	0.455597

Define and Fit model

In [18]:
```
## Fit an LSTM model on the multivariate input dat.
## First, Splitting the prepared data sets into train and test sets.
## To speed up the training model, fit the model on the first year of data,then evaluate on the remai
ning 3 years of data.
## Splits the dataset into train and test tests, then splits the train and test sets into input and o
utput variables.
## Then input(X) are reshaped into 3D format expected by LSTMs namley [samples,timesteps,features]
```

In [37]:
```
## split into train and test sets

values = reframed.values
n_train_hours = 365*24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
```

In [38]:
```
## split into input and outputs

train_x, train_y = train[:, :-1], train[:, -1]
test_x, test_y = test[:, :-1], test[:, -1]
```

In [39]:
```
## reshape input to be 3D

train_x = train_x.reshape((train_x.shape[0], 1, train_x.shape[1]))
test_x = test_x.reshape((test_x.shape[0], 1, test_x.shape[1]))
```

In [40]:
```
## Prints the shape of input of the train and test input and output about 9k hours for training and 2
6k hours for testing.

print(train_x.shape, train_y.shape,test_x.shape, test_y.shape )
```

```
(8760, 1, 7) (8760,) (25816, 1, 7) (25816,)
```

In [41]:
```
test_y
```

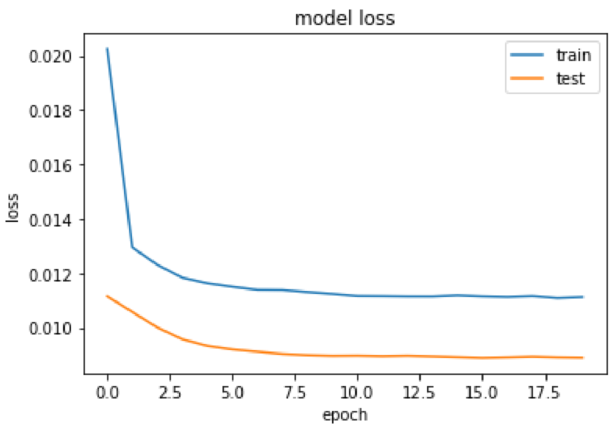
Out[41]:
```
array([0.36525355, 0.38881696, 0.42106517, ..., 0.25313834, 0.15394933,
       0.03053403])
```

Fit our LSTM Model

In [42]:
```
## Define LSTM with 100 neurons in the first hidden layer and 1 neuron in the output layer for predic
ting power consumption.
## The input shape will 1 time step with 8 features
## Using Mean Absolute error(MAE) loss function and the efficient adam version stochastic gradient de
scent
## The model will be fit for 20 training epchos with a batch size of 70.
## Tracking the training and test loss during training by setting the validation_data argument in the
fit() fn.
## Plotting the training and test loss are plotted
```

In [43]:
```
## design network
model = Sequential()
model.add(LSTM(100, input_shape=(train_x.shape[1], train_x.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')


# fit network
history = model.fit(train_x, train_y, epochs=20, batch_size=80, validation_data=(test_x, test_y), ver
bose=2, shuffle=False)

# plot history
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```

```
Epoch 1/20
110/110 - 2s - loss: 0.0203 - val_loss: 0.0112
Epoch 2/20
110/110 - 1s - loss: 0.0130 - val_loss: 0.0106
Epoch 3/20
110/110 - 1s - loss: 0.0123 - val_loss: 0.0100
Epoch 4/20
110/110 - 1s - loss: 0.0118 - val_loss: 0.0096
Epoch 5/20
110/110 - 1s - loss: 0.0116 - val_loss: 0.0093
Epoch 6/20
110/110 - 1s - loss: 0.0115 - val_loss: 0.0092
Epoch 7/20
110/110 - 1s - loss: 0.0114 - val_loss: 0.0091
Epoch 8/20
110/110 - 1s - loss: 0.0114 - val_loss: 0.0090
Epoch 9/20
110/110 - 1s - loss: 0.0113 - val_loss: 0.0090
Epoch 10/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0090
Epoch 11/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0090
Epoch 12/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0090
Epoch 13/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0090
Epoch 14/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0089
Epoch 15/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0089
Epoch 16/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0089
Epoch 17/20
110/110 - 1s - loss: 0.0111 - val_loss: 0.0089
Epoch 18/20
110/110 - 1s - loss: 0.0112 - val_loss: 0.0089
Epoch 19/20
110/110 - 1s - loss: 0.0111 - val_loss: 0.0089
Epoch 20/20
110/110 - 1s - loss: 0.0111 - val_loss: 0.0089
```



Evaluate Model

After the model is fit, we can forecast the entire test dataset

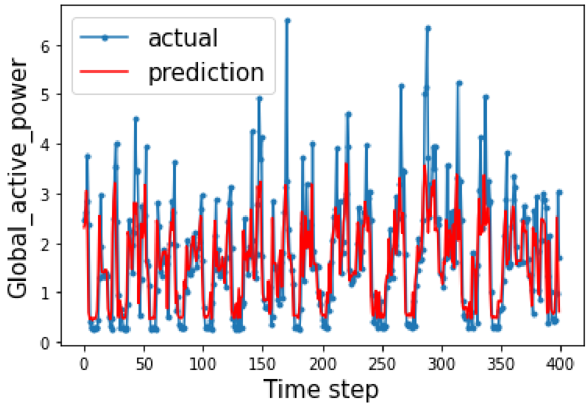
In [44]:
```
## Combine the forecast with the test data set and invert scaling.
## Invert scaling on the test dataset with the exxpected power consumption numbers.
## With forecasts and actual values in their scale, Calcuation of an error is made.
## For this we calculate RMSE( Root mean squared error) that gives error in the same units as the var
iable itself.
```

In [45]:
```
# make a prediction
yhat = model.predict(test_x)
test_x = test_x.reshape((test_x.shape[0], 7))

# invert scaling for forecast
inv_yhat = np.concatenate((yhat, test_x[:, -6:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = np.concatenate((test_y, test_x[:, -6:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

# calculate RMSE
rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.607
```

Plotting Actual vs Predicted

In [46]:
```
px=[x for x in range(400)]
plt.plot(px, inv_y[:400], marker='.', label="actual")
plt.plot(px, inv_yhat[:400], 'r', label="prediction")
plt.ylabel('Global_active_power', size=15)
plt.xlabel('Time step', size=15)
plt.legend(fontsize=15)
plt.show()
```



In [50]:
```
from sklearn import metrics
print('LSTM :\n')
print('Mean Absolute Error:',metrics.mean_absolute_error(inv_y,inv_yhat))
print('Mean Squared Error:',metrics.mean_squared_error(inv_y,inv_yhat))
print('Root Mean Squared Error:',np.sqrt(metrics.mean_squared_error(inv_y,inv_yhat)))
```

```
LSTM :

Mean Absolute Error: 0.4360111388962357
Mean Squared Error: 0.3687112006912978
Root Mean Squared Error: 0.6072159423889477
```

In [51]:
```
df_actual = pd.DataFrame({'Actual': inv_y})
```

In [52]:
```
predict_ser=pd.Series(inv_yhat.reshape(25816,))
```

In [53]:
```
df1_result=pd.concat([df_actual,predict_ser],axis=1)
```

In [54]:
```
df1_result
```

Out[54]:

	Actual	0
0	2.474967	2.310585
1	2.626633	2.427415
2	2.834200	3.047720
3	3.767167	2.176725
4	2.361233	0.590739
...	...	...
25811	2.280867	2.025790
25812	2.391967	1.606829
25813	1.753333	1.167933
25814	1.114900	0.476604
25815	0.320533	0.765389

25816 rows × 2 columns

In [56]:
```
df1_result.columns=['Actual','Predicted']
df1_result
```

Out[56]:

	Actual	Predicted
0	2.474967	2.310585
1	2.626633	2.427415
2	2.834200	3.047720
3	3.767167	2.176725
4	2.361233	0.590739
...	...	...
25811	2.280867	2.025790
25812	2.391967	1.606829
25813	1.753333	1.167933
25814	1.114900	0.476604
25815	0.320533	0.765389

25816 rows × 2 columns

In [57]:
```
df1_result.to_csv('Actaul and predicted numerical values for an hour prediction.csv')
```

In [ ]: