ANALYSIS OF HOUSEHOLD ELECTRIC POWER CONSUMPTION

CAPSTONE FINAL REPORT

Submitted By:
Himaja Gaddam, M13500741
MS Business Analytics, University of Cincinnati

First Reader: Dr. Yichen Qin
Second Reader: Dr. Dungang Liu

ABSTRACT:

The energy sector has been an important driver of industrial growth over the past century, providing fuel to power the rest of the economy. Many things in our life starting from lighting our rooms to operating a heavy machinery runs on electricity. Nowadays all the countries are concerned about providing sufficient energy to the consumers as well as optimizing the total demand of energy consumed. Since a significant part of that energy is consumed by household sector, the optimal consumption of the energy at home is of great importance. To better regulate the production of energy it is important to understand the energy needs of this sector which can be done by analyzing the past energy consumption data. This report presents the analysis of electric power consumption data of a household collected every minute for 4 years. The given data is analyzed by aggregating the data over hour. Different models are trained on the aggregated training datasets using the approaches VAR and LSTM.

Table of Contents

INTRODUCTION:

Electric energy is an essential ingredient in economic growth for improving the quality of human life. Approximately 30% of this energy is consumed by households. Energy consumption by the households is usually because of cooking, water heating, space heating and some electric appliances like Refrigerator, Washing machine and Dryer. This project is to analyze past electric power consumption by a household and built a model which can be used to forecast future energy requirements.

DATASET BACKGROUND:

- The Household Power Consumption dataset is a multivariate time series dataset
- It describes the electricity consumption for a household located in Sceaux (7km of Paris, France)
- This dataset contains 2075259 observations and 9 variables (including date and time)
- The data was collected between December 2006 and November 2010 every minute

ATTRIBUTE INFORMATION:

1. ***Date****:* Date in format dd/mm/yyyy
2. ***Time****:* time in format hh:mm:ss
3. ***Global_active_power****:* household global minute-averaged active power (in kilowatt)
4. ***Global_reactive_power****:* household global minute-averaged reactive power (in kilowatt)
5. ***Voltage***: minute-averaged voltage (in volt)
6. ***Global_intensity***: household global minute-averaged current intensity (in ampere)
7. ***Sub_metering_1***: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen (hot plates are not electric, but gas powered).
8. ***Sub_metering_2***: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
9. ***Sub_metering_3***: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

SOURCE:

https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption

Original Source:

Georges Hebrail (georges.hebrail **'@'** edf.fr), Senior Researcher, EDF R&D, Clamart, France
Alice Berard, TELECOM ParisTech Master of Engineering Internship at EDF R&D, Clamart, France

EXPLORATORY DATA ANALYSIS:

DATA UNDERSTANDING:

Global active energy is the real power consumed by the household, whereas the global reactive energy is the unused power in the power lines.

Sub_metering_1, Sub_metering_2, Sub_metering_3 is the energy consumed by specific units in the household. There is some energy consumption which is not included in any of these three measurements.

The remaining energy consumption (let's say Sub_metering_4) can be calculated using below formula:

*Sub_metering_4=(global_active_power*1000/60)–(Sub_metering_1+Sub_metering_2+Sub_metering_3)*

(Converted global_active_power from kilowatt and watt-hour)

DATA CLEANING:

1. Converted the dataset to time series type by taking date and time variables to be the time index
2. Added Sub_metering_4 variable to the data set using above formula
3. This dataset has 25979 null values (?) which is about 1.25% of the data.
4. Converted all the variables to float type and replaced all the null(?) values with nan
5. Filled all the nan values with the value at the same time one day ago

DATA VISUALIZATION:

After data cleaning, the final dataset has 2075259 observations and 8 variables with index as datetime. Summary statistics and the boxplots of the variables of this dataset is as follows:

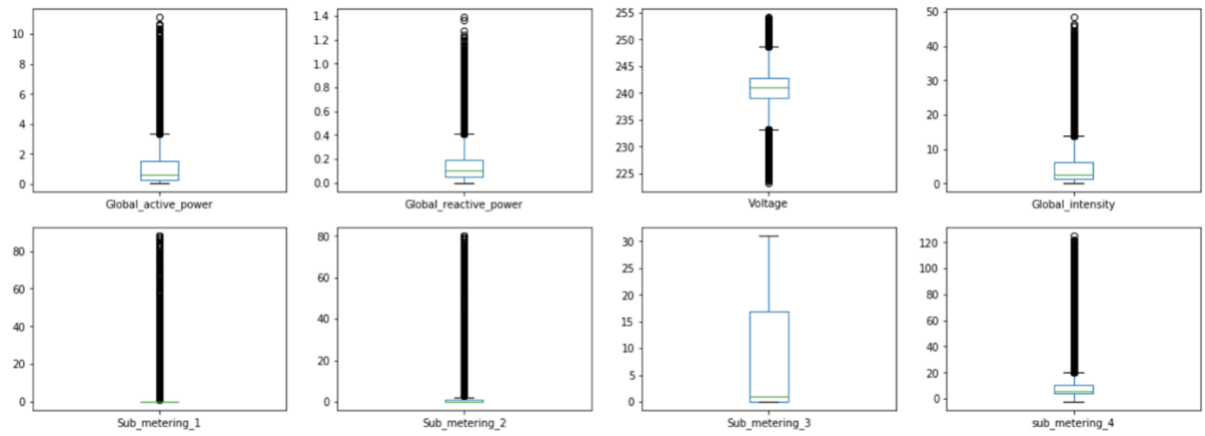
Table 1: Summary statistics of the dataset

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	sub_metering_4
count	2075259.000	2075259.000	2075259.000	2075259.000	2075259.000	2075259.000	2075259.000	2075259.000
mean	1.090	0.124	243.432	4.620	1.118	1.291	6.449	9.299
std	1.054	0.113	4.148	4.430	6.093	5.734	8.457	9.558
min	0.076	0.000	223.200	0.200	0.000	0.000	0.000	-2.400
25%	0.308	0.048	238.990	1.400	0.000	0.000	0.000	3.800
50%	0.602	0.100	241.000	2.600	0.000	0.000	1.000	5.500
75%	1.526	0.194	242.870	6.400	0.000	1.000	17.000	10.367
max	11.122	1.390	254.150	48.400	88.000	80.000	31.000	124.833

Inference:

From the above table, we can say that almost all the variables have outliers

Chart 1: Boxplots of all variables of the dataset

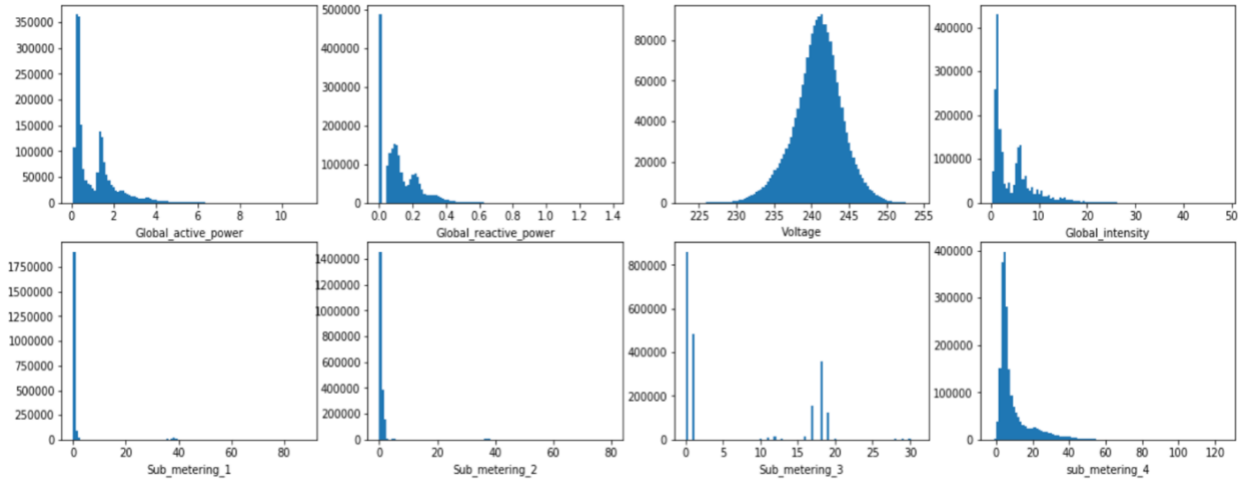


Inference:

- Above chart shows that all the variables except Sub_metering_3 has outliers same as
- This inference is same as the inference from the summary statistics.

Next to understand the distributions of these variables, below histograms are plotted.

Chart 2: Histograms of all variables of the dataset

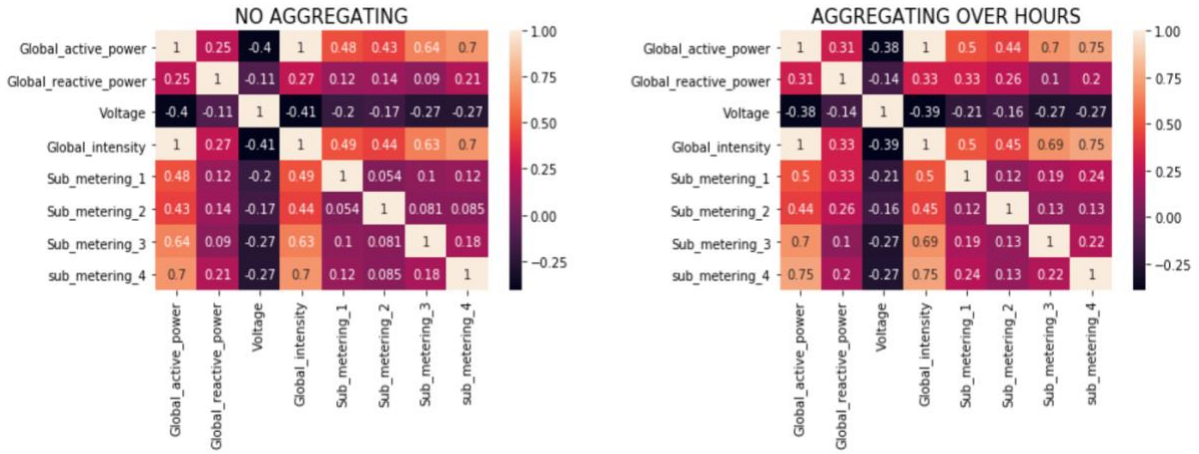


Inference:

- We can see that the variables except Voltage are all left skewed towards small values.
- Distribution of voltage data is strongly Gaussian i.e., Normal.
- The distribution of global_active_power , global_reactive_power and global_intensity appears to be bi-modal i.e., they have two mean groups of observations

Now, going ahead to understand the correlation between the variables by calculating correlation matrices and plotting the heat maps for the original data and the data aggregated over hour.

Chart 3: Heatmaps of correlation matrices

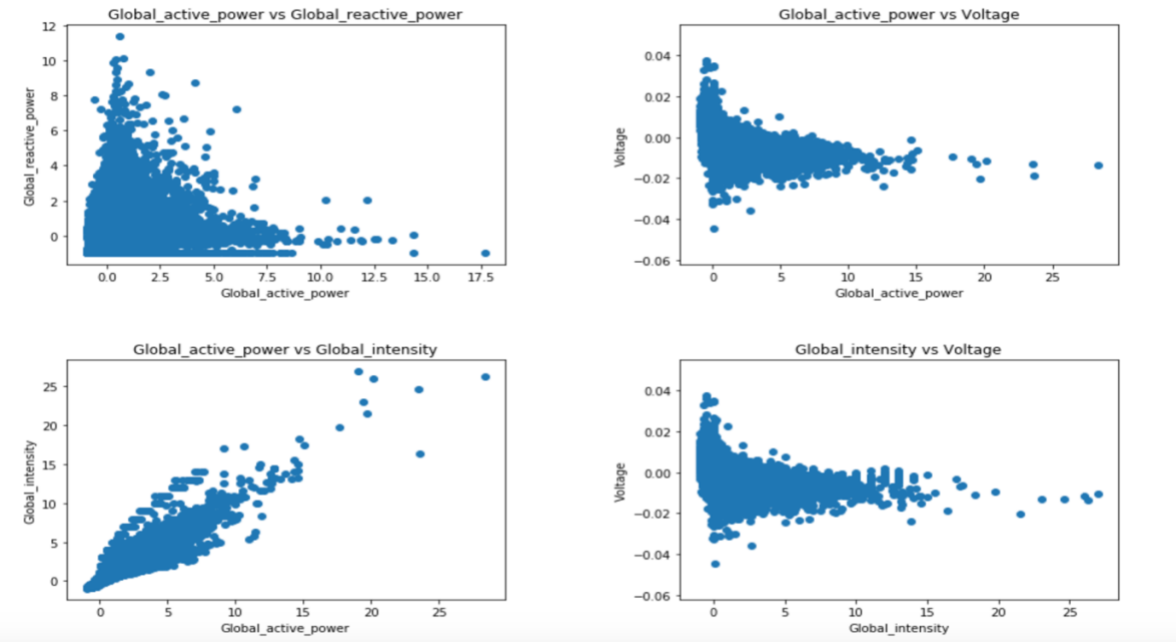


Inference:

- Above maps show that global_active_power and global_intensity have strong correlation.
- Also, global_active_power and sub_metering_4 have strong correlation and this correlation increased when the data is aggregated over hour.

Also, to understand the relationship between change in global_active_power with the change in other global parameters in detail, scatter plots are used.

Chart 4: Scatterplots between change in global variables



Inference:

- From the above scatter plots, we can say that the change in the global_active_power has strong correlation with the change in global_intensity.
- Change in the global_active_power doesn't have strong correlation with the change in other global variables.
- We can also see that the relation between change in voltage with change in global_active_power is almost same as the change in voltage with change in global_intensity. This behavior is expected as there is very strong correlation between global_active_power and global_intensity.
- we can observe that global_active_power and global_reactive power has almost no correlation.

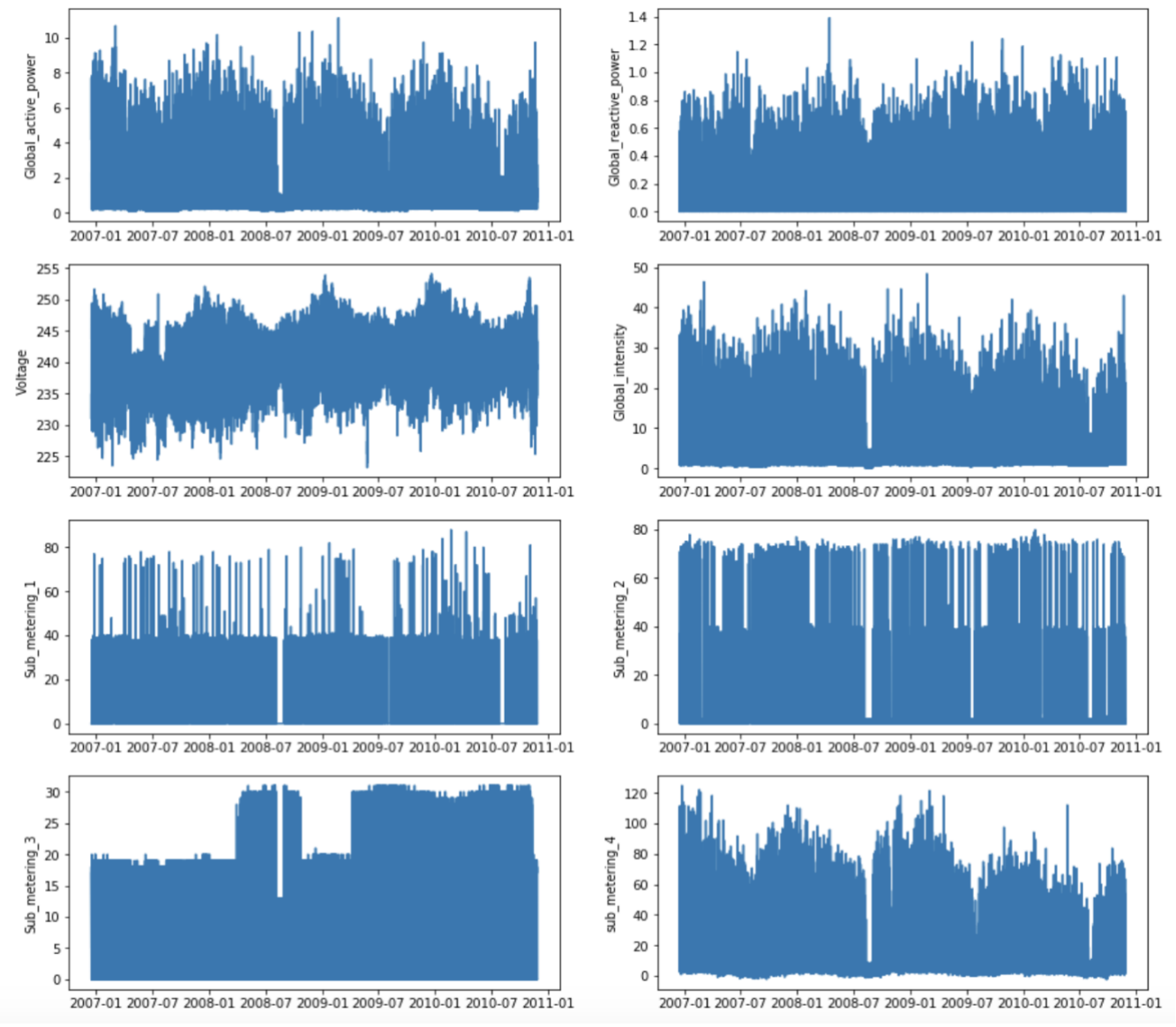
PATTERN RECOGNITION:

To identify the patterns like seasonality and trend in the given variables:

1. All the variables are plotted against time
2. Means of the variables aggregated over hours are plotted against time

1. ***Variables plotted against time gives the high-level view of four years of one-minute observations***

Chart 5: Variables plotted against time

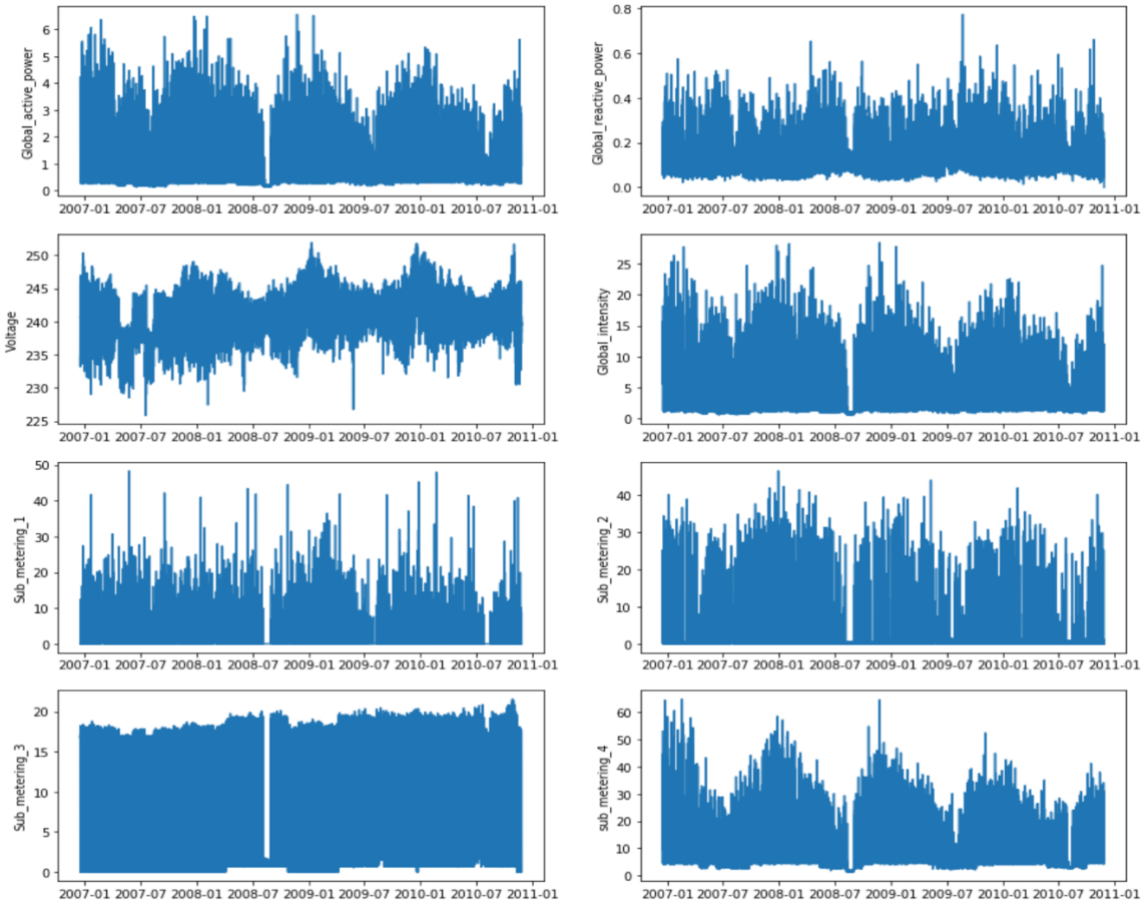


Inference:

- We can see seasonality in variables global_active_power, global_intensity and voltage.
- Variable global_reactive_power has no seasonality nor trend.
- Sub_metering_1 and sub_metering_2 also has slight seasonal trend.
- Sub_metering_3 has suddenly increased in 2008 for a season and stayed high from 2009. This doesn't correspond to any seasonality or trend.
- Sub_metering_4 also shows some seasonality also it shows decreasing trend towards the years
- There are some spikes in almost all the variables, which might correspond to weekends.

2. ***Means of the variables aggregated over hours plotted against time gives the detailed trend of each variable for the aggregated data.***

Chart 6: Means of all the variables aggregated over hours



Inference:

From the above plots, we can say that the structure of all the variable plots are same as the plots with original data.

DATA MODELING:

Below models built on this data and these models can be used to predict the future energy need:

1. VAR Model built using data aggregation over 'Hour'
2. LSTM Model built using data aggregation over 'Hour'

For the above models, the data used is aggregated data not the original data. Here aggregating the data implies changing the frequency of the time series which generates a unique sampling distribution based on the actual data. The original data frequency is 'Minute' whereas the frequency used for the aggregation is 'Hour'. The groupby function used with aggregation is mean which gives the average of the values in a particular period.

Original data has 2075259 observations whereas the aggregated data over 'Hour' has only 34589 observations. When aggregated over 'Hour', number of data points reduced drastically but the structures of both the series remained same and the computation time reduced significantly. So, to build the models the data aggregated over hour is used instead of the original data.

Also, to get some additional perspective and insight into the data, models can be built on the data aggregated over different frequencies as aggregating over different frequencies can give different structures to the data. Additionally, as the frequency becomes wider the computation time decreases so based on the end user requirement a model can be built on the appropriately aggregated data.

For each of the above-mentioned models, data modeling is done as follows:

I. Original data is first aggregated over hours and then split into training and testing data points.
II. Model is built on the training data and a forecast is made for the duration of the testing period.
III. Mean Squared Error (MSE) of the testing data (Global_active_power) is calculated using forecasted values using the built model and the testing data.
IV. Model is now trained on the entire data and this model can be used to predict future energy need.

VAR MODEL BUILT WITH DATA AGGREGATED OVER 'HOUR':

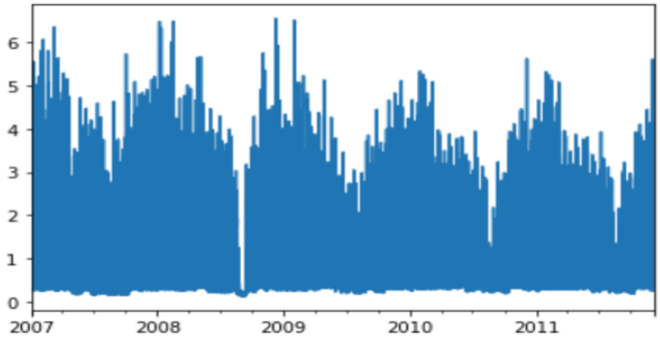
The aggregated data (with mean) is first checked for seasonality and found that the variables 'Global_active_power', 'Global_intensity', 'Voltage', 'Sub_metering_2' and 'sub_metering_4' have high seasonality. Then these variables are seasonally adjusted.

The seasonally adjusted data is then tested for stationarity. All the variables except 'sub_metering_4' are stationary. This variable is made stationary using first order difference.

Then the data is split, trained on two-thirds data and tested on one-thirds data for which the calculated MSE is 0.8838. As the MSE is high, we can say that the model doesn't have good predicting power.

Now the model is trained on the entire data and the 8760 (365*24) values of the variable 'Global_active_power' are forecasted (for 1 year), readjusted for seasonality and plotted below.

Chart 7: Global_active_power aggregated over hour and the forecasted values



From the above VAR models, we can say that the forecasting power of the models decreased as the data resampling duration decreased from month to hour.

But if we need to forecast the future need on daily basis, VAR is not efficient. So, let us build a neural network using LSTM and see the forecasting power of the LSTM model built with aggregated data.

LSTM MODEL BUILT WITH DATA AGGREGATED OVER 'HOUR':

Before jumping into the LSTM modeling, let us have a look at LSTM structure and its advantages.

LONG SHORT-TERM MEMORY (LSTM):

Usually Recurrent Neural Networks (RNNs), a class of artificial neural networks, are used for modeling sequence data such as time series or natural language. Connections between RNN units form a directed graph along a sequence which allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This feature makes them applicable to tasks involving sequences such as time series.

Though RNNs are powerful and can achieve state of the art performance, they suffer from short-term memory which makes them inefficient in modeling long sequences as they will have hard time carrying information from earlier steps to later ones i.e., RNN's may leave out some important information from earlier steps. Short-term memory in RNNs is essentially due to vanishing gradient during back propagation through time. So, as gradient becomes smaller learning rate of the RNN layers becomes smaller and eventually they stop learning which makes RNN forget what it has seen in longer sequences.

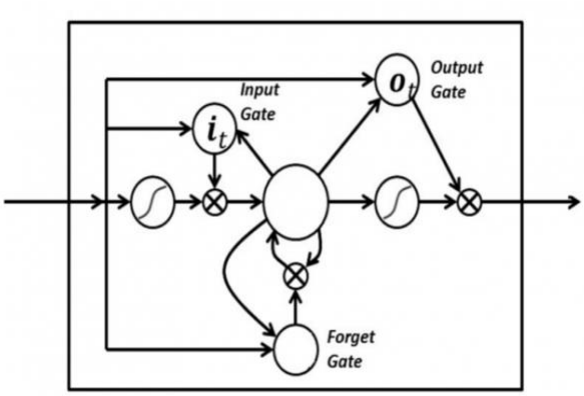
LSTMs (Long Short-Term Memory), a subset of RNNs, were created as a method to mitigate short-term memory problem of RNNs. They have internal mechanisms called gates, which are also neural networks, that can regulate the flow of information by learning which data in a sequence is important to keep or throw away and passing relevant information down the long chain of sequences to make predictions. These gates, implemented with element-wise multiplication by sigmoids, are analog in the range of 0-1 and this(analog) character of the gates has an advantage of being differentiable making them suitable for back propagation.

Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.

LSTM ARCHITECTURE:

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell. These units of an LSTM are used as building units for the layers of an RNN, often called an LSTM network. Below is the LSTM unit structure.

Figure 1: LSTM unit structure



FORGET GATE:

Forget gate decides what information should be thrown away or kept. Information from the previous hidden state and the current input is passed as input through the sigmoid function of the forget gate and the output of this function is value between 0 and 1. Now, the network forgets the information with output value closer to 0 and keeps the information with output value closer to 1.

INPUT GATE:

Input gate updates the cell state by taking information from previous hidden state and current input into its sigmoid function as its input and giving a value between 0 and 1 as output. Output value closer to zero implies the information is not important and the output value closer to 1 implies that the information is important. Cell state is updated to new values by pointwise multiplying its current values by the forget vector and then pointwise adding the output from input gate. This gives the new cell state.

OUTPUT GATE:

The output gate decides what the next hidden state should be. Hidden state contains information on previous input, and it is also used for predictions. previous hidden state and the current input are passed as inputs into a sigmoid function whose output lies between 0 and 1. This output is multiplied with the output from a tanh function whose input is the new cell state and this result is saved as the next hidden state. This new hidden state and the new cell state is then carried over to the next step.

MODELING:

Now, let us look into the modeling part of LSTM. This method is implemented using the “LSTM()” function from “keras” library in python. The model is initiated with the method sequential().

First, the original data is aggregated over hour using mean. Then this data is scaled to have all the values between 0 & 1 so that the neural network convergence is quicker and easier.

Finally, the data is reshaped so that each row will have all the variables at time t-1 along with the variable 'global_active_power' value at time t. All the values at time t-1 are the predictors i.e., 7 X variables and the value at time t is the response variable i.e., y variable.

The final reshaped dataset has 34588 rows and 8 columns. The final reshaped data looks as follows:

Table 2: Reshaped data for LSTM model

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var1(t)
1	0.637	0.296	0.338	0.631	0.000	0.011	0.782	0.545
2	0.545	0.103	0.336	0.541	0.000	0.145	0.783	0.509
3	0.509	0.110	0.284	0.502	0.000	0.031	0.774	0.489
4	0.489	0.097	0.316	0.481	0.000	0.000	0.779	0.456
5	0.456	0.099	0.434	0.450	0.000	0.009	0.799	0.323

Here the dataset is essentially reshaped into a supervised learning problem and the model is trained to predict the energy needed ('global_active_power') at the current hour (t) i.e., var1(t) given the energy used at the prior time step i.e., var1(t-1) to var7(t-1). So, this model is trained like a predicting model which predicts response variable, var1(t) based on the predictor variables, var1(t-1), .. ,var7(t-1). Testing on this model is also done in the similar approach.

Now, the data is split, trained on 75% data using LSTM function having 100 neurons in the first layer.

Dense units i.e., dimensionality of the output space is taken as 1 and drop out of 20% is used to handle the overfitting issue. 'adam' stochastic gradient optimizer is used so that the network weights are updated for each iteration.

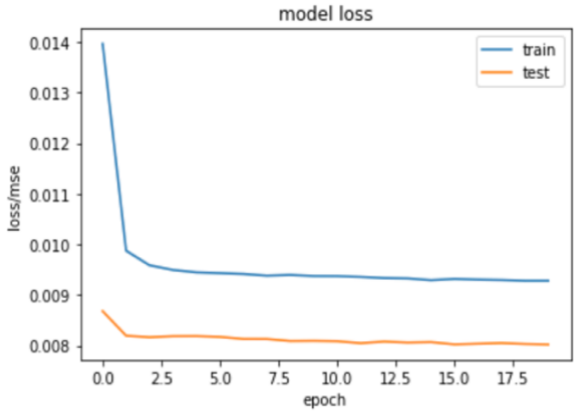
Model is tuned for different epochs and batch sizes and MSE for the testing data is calculated for each of those combinations.

Table 3: MSE for different epochs and batch sizes

	10	15	20	25
70	0.335	0.334	0.332	0.332
80	0.335	0.335	0.336	0.336
90	0.337	0.340	0.342	0.343
100	0.339	0.348	0.351	0.350

From the above we can say that 70 is the optimal batch size and 25/20 is the optimal number of epochs required to get minimal MSE. For batch size = 70 and epochs = 20 (to decrease processing time), loss i.e., MSE is plotted for each epoch in the below chart.

Chart 8: MSE vs Epoch for both training and testing data



RESULTS:

MSE of the testing data is used as the performance metric to compare the models built on the data aggregated over hours. Below table shows the MSE of all the models.

Table 4: MSE of different models

Model	MSE
VAR Model built with data aggregated over 'Hour'	0.8838
LSTM Model built with data aggregated over 'Hour'	0.332

To get detailed predictions like hourly, LSTM model is better than the VAR model.

CONCLUSION:

For the data aggregated over hour, MSE of LSTM model is significantly lower than the MSE of VAR models which makes the LSTM model better for predicting future needs. The final chosen model is approximately 70% efficient in predicting the power needed based on the electric power used in previous time frames.

This prediction can be used in many ways. First of all, consumers can control their consumption and manage their bills efficiently. Secondly, the power grid and power sales companies can meet demands by loading optimal energy to the substations and charging for the electric units. Finally, government can plan different schemes for different group of consumers based on the predictions.

FUTURE SCOPE:

Above LSTM model can be further improved by including CNN (Convolutional Neural Network) and drop out layers. Including the CNN to this model increases efficiency and decreases the computation time.

REFERENCES:

I. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
II. https://www.kaggle.com/uciml/electric-power-consumption-data-set
III. https://machinelearningmastery.com/how-to-load-and-explore-household-electricity-usage-data/
IV. Gers F.A., Eck D., Schmidhuber J. (2002) Applying LSTM to Time Series Predictable Through Time-Window Approaches. In: Tagliaferri R., Marinaro M. (eds) Neural Nets WIRN Vietri-01. Perspectives in Neural Computing. Springer, London
V. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
VI. https://stackabuse.com/time-series-analysis-with-lstm-using-pythons-keras-library/#:~:text=LSTM%20(Long%20Short%2DTerm%20Memory,used%20for%20time%20series%20analysis.
VII. Fouladgar, N.; Främling, K. A Novel LSTM for Multivariate Time Series with Massive Missingness. *Sensors* **2020**, *20*, 2832.
VIII. M. Nelson, T. Hill, W. Remus, and M. O'Connor, "Time series forecasting using neural networks: should the data be deseasonalized first?" J. Forecast., vol. 18, no. 5, pp. 359–367, 1999.
IX.] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," Eur. J. Oper. Res., vol. 160, no. 2, pp. 501–514, 2005.
X. Khodabakhsh A., Ari I., Bakır M., Alagoz S.M. (2020) Forecasting Multivariate Time-Series Data Using LSTM and Mini-Batches. In: Bohlouli M., Sadeghi Bigham B., Narimani Z., Vasighi M., Ansari E. (eds) Data Science: From Research to Application. CiDaS 2019. Lecture Notes on Data Engineering and Communications Technologies, vol 45. Springer, Cham

APPENDIX:

PYTHON CODE:

```
#Import libraries for visualization and data analysis

import sys
import numpy as np
from scipy.stats import randint
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import nan
from numpy import isnan
```

In [ ]:

```
# Import data modeling library - methods - machine learning

from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics
from sklearn.metrics import mean_squared_error,r2_score
```

In [ ]:

```
pip install keras
```

In [ ]:

```
pip install tensorflow
```

In [ ]:

```
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
import itertools
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import Dropout
```

In [ ]:

```
# Function to fill missing values with a value at the same time one day ago
def fill_missing(values):
    one_day = 60 * 24
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if isnan(values[row, col]):
                values[row, col] = values[row - one_day, col]
```

In [ ]:

```
# load all data
data = pd.read_csv('household_power_consumption.txt', sep=';', header=0, low_memory=F
alse, infer_datetime_format=True, parse_dates={'datetime':[0,1]}, index_col=['datetim
e'])

# Replace the missing values with nan
data.replace('?', nan, inplace=True)

# convert datatype of all variables to float
data = data.astype('float32')

## finding all columns that have missing values:
droping_list_all=[]
for j in range(0,7):
```

```
    if not data.iloc[:, j].notnull().all():
        droping_list_all.append(j)
droping_list_all
```

In [ ]:

```
#Counting the missing values
count=0
for i in data.isnull().sum(axis=1):
    if i>0:
        count=count+1
print('missing values: ', count, count*100/2075259,'%')
```

In [ ]:

```
# fill missing values using the pre-defined function
fill_missing(data.values)

# adding a new column for remainder of sub metering
values = data.values
data['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4] + values[:,5] + val
ues[:,6])

# save updated dataset
data.to_csv('household_power_consumption.csv')
```

In [ ]:

```
data.shape
```

In [ ]:

```
data.head(5)
```

In [ ]:

```
data.info()
```

In [ ]:

```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
data.describe()
```

In [ ]:

```
#Visualization:
#Check the distributions of variables:
# histogram plot for each variable
plt.figure(figsize=(20, 30))
for i in range(len(data.columns)):
    plt.subplot(len(data.columns), 4, i+1)
    name = data.columns[i]
    data[name].hist(bins=100)
    plt.xlabel(name)
    plt.grid(False)
plt.show();
```

In [ ]:

```
#Check for the outliers , mean and median comparison
plt.figure(figsize=(20, 30))
for i in range(len(data.columns)):
    plt.subplot(len(data.columns), 4, i+1)
```

```
    name = data.columns[i]
    data.boxplot([name])
    plt.grid(False)
plt.show()
```

In [ ]:

```
#Correlations
# calculate the correlation matrices
corr = data.corr()
corr_H = data.resample('H').mean().corr()

# plot the heatmaps
fig,ax = plt.subplots(2,1, figsize=(15, 10))
plt.subplot(221)
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,annot=True)
plt.title('NO AGGREGATING', size=15)
plt.subplot(222)
sns.heatmap(corr_H,
        xticklabels=corr_H.columns,
        yticklabels=corr_H.columns,annot=True)
plt.title('AGGREGATING OVER HOURS',size=15)

plt.subplots_adjust(hspace=0.75,
                    wspace=0.5);
```

In [ ]:

```
#Plot the correlation between some variables
fig,ax = plt.subplots(2,2, figsize=(15, 10))
data_updated=data.pct_change()
ax[0,0].scatter('Global_active_power','Global_reactive_power', data=data_updated)
ax[0,0].set_title('Global_active_power vs Global_reactive_power')
ax[0,0].set_xlabel('Global_active_power')
ax[0,0].set_ylabel('Global_reactive_power')
ax[0,1].scatter('Global_active_power','Voltage', data=data_updated)
ax[0,1].set_title('Global_active_power vs Voltage')
ax[0,1].set_xlabel('Global_active_power')
ax[0,1].set_ylabel('Voltage')
ax[1,0].scatter('Global_active_power','Global_intensity', data=data_updated)
ax[1,0].set_title('Global_active_power vs Global_intensity')
ax[1,0].set_xlabel('Global_active_power')
ax[1,0].set_ylabel('Global_intensity')
ax[1,1].scatter('Global_intensity','Voltage', data=data_updated)
ax[1,1].set_title('Global_intensity vs Voltage')
ax[1,1].set_xlabel('Global_intensity')
ax[1,1].set_ylabel('Voltage')
plt.subplots_adjust(hspace=0.4,
                    wspace=0.4);
```

In [ ]:

```
#Plot the variables across datetime
plt.figure(figsize=(15, 30))
for i in range(len(data.columns)):
    plt.subplot(len(data.columns), 2, i+1)
    name = data.columns[i]
    plt.plot(data[name])
    plt.ylabel(name)
    plt.grid(False)
plt.show();
```

In [ ]:

```
#Plot the variables across datetime - resampled over hour
data1=data.resample('H').mean()
plt.figure(figsize=(15, 30))
for i in range(len(data1.columns)):
    plt.subplot(len(data1.columns), 2, i+1)
    name = data1.columns[i]
    plt.plot(data1[name])
    plt.ylabel(name)
    plt.grid(False)
plt.show();
```

In [ ]:

```
from statsmodels.tsa.stattools import adfuller
#Function for stationarity check
def adf_test(ts, signif=0.05):
    dftest = adfuller(ts, autolag='AIC')
    adf = pd.Series(dftest[0:4], index=['Test Statistic','p-value','# Lags','# Observ
ations'])
    for key,value in dftest[4].items():
       adf['Critical Value (%s)'%key] = value
    #print (adf)

    p = adf['p-value']
    if p <= signif:
        print(f" Series is Stationary")
        return True
    else:
        print(f" Series is Non-Stationary")
        return False
```

In [ ]:

```
#Function to make stationary
def difference(dataset, interval=1):
    diff = list()
    diff.append(0)
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
```

```
    return diff
```

In [ ]:

```
data_resampled=data.resample('H').mean()
data_resampled.shape
```

In [ ]:

```
data_resampled=data.resample('H').mean()
data_adjusted=data_resampled[8760:]
print(data_adjusted.shape)

#Adjusting seasonality
hourly_mean = data_resampled['Global_active_power']
diff = list()
for i in range(8760, len(hourly_mean)):
    value = hourly_mean[i] - hourly_mean[i - 8760]
    diff.append(value)
print(len(diff))
data_adjusted['Global_active_power']=diff
plt.plot(data_adjusted['Global_active_power'])
plt.show();

hourly_mean = data_resampled['Voltage']
diff = list()
for i in range(8760, len(hourly_mean)):
    value = hourly_mean[i] - hourly_mean[i - 8760]
    diff.append(value)
data_adjusted['Voltage']=diff
plt.plot(data_adjusted['Voltage'])
plt.show();

hourly_mean = data_resampled['Global_intensity']
diff = list()
for i in range(8760, len(hourly_mean)):
    value = hourly_mean[i] - hourly_mean[i - 8760]
    diff.append(value)
data_adjusted['Global_intensity']=diff
plt.plot(data_adjusted['Global_intensity'])
plt.show();

hourly_mean = data_resampled['Sub_metering_2']
diff = list()
for i in range(8760, len(hourly_mean)):
    value = hourly_mean[i] - hourly_mean[i - 8760]
    diff.append(value)
data_adjusted['Sub_metering_2']=diff
plt.plot(data_adjusted['Sub_metering_2'])
plt.show();
```

```
hourly_mean = data_resampled['sub_metering_4']
diff = list()
for i in range(8760, len(hourly_mean)):
    value = hourly_mean[i] - hourly_mean[i - 8760]
    diff.append(value)
data_adjusted['sub_metering_4']=diff
plt.plot(data_adjusted['sub_metering_4'])
plt.show();
```

In [ ]:

```
for col in data_adjusted.columns:
    print("Testing column is:",col)
    stat_model=adf_test(data_adjusted[col],0.05)
    while stat_model==False:
        data_adjusted[col] = difference(data_adjusted[col])
        stat_model=adf_test(data_adjusted[col],0.05)

ncount = data_adjusted.shape[0]
print(ncount)

# split into train and test sets
values = data_adjusted.values
n_train_time = round(ncount/3)
print(n_train_time)
train = values[:n_train_time, :]
test = values[n_train_time:, :]
n=test.shape[0]

#Train the model and forecast global values
from statsmodels.tsa.api import VAR
model = VAR(train)
model_fit = model.fit()
pred = model_fit.forecast(model_fit.y, steps=n)


result_pred=[]
for i in range(0,n):
    result_pred.append(pred[i][0])
result_pred

result_actual=test[:,0]
print("MSE",mean_squared_error(result_actual, result_pred))
```

In [ ]:

```
pred_count=365*24
model = VAR(values)
model_fit_final = model.fit()
pred_final = model_fit_final.forecast(model_fit_final.y, steps=pred_count)
print(pred_final.shape)
```

```
var_pred=pd.DataFrame(pred_final)
var_pred.columns=data.columns
j=0
global_values=[]
init_values = data_resampled.values
for i in range(0,len(init_values)):
    global_values.append(init_values[i][0])
print(len(global_values))

#Readjust the seasonality
hourly_mean = data_resampled['Global_active_power']
print(hourly_mean.shape)
diff = list()
j=0
```

In [ ]:

```
for i in range(25829,34589):
    global_values.append(pred_final[j][0] + hourly_mean[i])
    j=j+1

ts = pd.Series(global_values,
               index=pd.period_range('12/31/2006', periods=len(global_values),freq='H'
))
ts.plot();
```

In [ ]:

```
#Function to reshape data - LSTM
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]
    dff = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(dff.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(dff.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, i)) 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 [ ]:

```
## resampling of data over hour
df=data
df=df.drop(columns="sub_metering_4")
df_resample = df.resample('h').mean()
df_resample.shape
```

In [ ]:

```
## * Note: I scale all features in range of [0,1].

## If you would like to train based on the resampled data (over hour), then used belo
w
values = df_resample.values

scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled.shape
```

In [ ]:

```
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
```

In [ ]:

```
# drop columns we don't want to predict
reframed.drop(reframed.columns[[8,9,10,11,12,13]], axis=1, inplace=True)
print(reframed.head())#Split the data to train & test datasets
```

In [ ]:

```
values = reframed.values

n_train_time = 365*24*3
train = values[:n_train_time, :]
test = values[n_train_time:, :]
##test = values[n_train_time:n_test_time, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
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]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
# We reshaped the input into the 3D format as expected by LSTMs, namely [samples, tim
esteps, features].

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=70, validation_data=(test
_X, test_y), verbose=2, shuffle=False)

# 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 MSE
mse = (mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % mse)
```

In [ ]:

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