LSTM(LONG-SHORT TERM MEMORY) MULTI-STEP TIME SERIES FORECASTING USING MANY-TO-SEQUENCE PREDICTION.

Open “main5” pycharm folder and execute the code by creating new project in your pycharm environiment and copy and paste the code in your new project.

Import pandas text file using pd.read\_csv and give the path of text file where the file it is located in your computer

import pandas as pd  
datase = pd.read\_csv('C:\\Users\\KALYAN\\Downloads\\pv.txt', header=0, low\_memory=False, infer\_datetime\_format=True, parse\_dates={'datetime':[0]}, index\_col=['datetime'])

import libraries required for data preprocessing

from numpy import nan  
from numpy import isnan  
from pandas import read\_csv  
from pandas import to\_numeric

fill the missed values with the same value of the old data given below:

def fill\_missing(values):  
 one\_da = 60 \* 24  
 for ro in range(values.shape[0]):  
 for colu in range(values.shape[1]):  
 if isnan(values[ro, colu]):  
 values[ro, colu] = values[ro - one\_da, colu]

Load the data set convert into numeric and give values to the dataset. and give the path of text file where the file it is located in your computer

datase = read\_csv('C:\\Users\\KALYAN\\Downloads\\pv.txt', header=0, low\_memory=False, infer\_datetime\_format=True,  
 parse\_dates={'datetime': [0]}, index\_col=['datetime'])  
datase.replace('?', nan, inplace=True)  
datase = datase.astype('float32')  
fill\_missing(datase.values)  
values = datase.values  
datase.to\_csv('pv.csv')

convert the text file into .csv file using the following command and make datetime in a single column and add new column

from pandas import read\_csv  
datase = read\_csv('pv.csv', header=0, infer\_datetime\_format=True, parse\_dates=['datetime'], index\_col=['datetime'])  
daily\_group = datase.resample('D')  
daily\_dat = daily\_group.sum()  
print(daily\_dat.shape)  
print(daily\_dat.head())  
daily\_dat.to\_csv('pvr.csv')

import libraries required for data splitting, classification of input and output sequences, model building, multistep forecasting with many-to-many sequences using encoders and decoders.

from math import sqrt  
from numpy import split  
from numpy import array  
from pandas import read\_csv  
from sklearn.metrics import mean\_squared\_error  
from matplotlib import pyplot  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import Flatten  
from keras.layers import LSTM  
from keras.layers import RepeatVector  
from keras.layers import TimeDistributed

split the train and test dataset using the command below

def split\_datase(data):  
 trainx, testy = data[1:-328], data[-328:-6]  
 trainx = array(split(trainx, len(trainx) / 7))  
 testy = array(split(testy, len(testy) / 7))  
 return trainx, testy

Evaluate the forecasts of actual and prediction using the evaluation metrics RMSE,MSE and MAE

def evaluate\_forecast(actua, predicte):  
 score = list()  
 for i in range (actua.shape[1]):  
 mse = mean\_squared\_error(actua[:, i], predicte[:, i])  
 rmse = sqrt(mse)  
 score.append(rmse)  
 s = 0  
 for ro in range(actua.shape[0]):  
 for colu in range(actua.shape[1]):  
 s += (actua[ro, colu] - predicte[ro, colu]) \*\* 2  
 scores = sqrt(s / (actua.shape[0] \* actua.shape[1]))  
 return scores, score

summarizes the scores of the evaluation metrics using the following code

def summarize\_score(name, scores, score):  
 ss\_scores = ', '.join(['%.1f' % s for s in score])  
 print('%s: [%.3f] %s' % (name, scores, ss\_scores))

encoding and decoding the history into multiple inputs and outputs and step over the entire history one time step at a time and generate multistep time series for many-to-many sequences.

def too\_supervised(trainx, N\_input, N\_out=7):  
 data = trainx.reshape((trainx.shape[0] \* trainx.shape[1], trainx.shape[2]))  
 X, y = list(), list()  
 i\_start = 0  
 for \_ in range(len(data)):  
 i\_end = i\_start + N\_input  
 o\_end = i\_end + N\_out  
 if o\_end <= len(data):  
 X.append(data[i\_start:i\_end, :])  
 y.append(data[i\_end:o\_end, 0])  
 i\_start += 1  
 return array(X), array(y)

build the model with performance metrics like verbose, epochs, batch size, and the model sequential with two activation layers ‘relu’ and optimizer = ‘adam’

def build\_model(trainx, N\_input):  
 trainx\_x, trainy\_y = too\_supervised(trainx, N\_input)  
 verbose, epochs, batch\_size = 0, 50, 16  
 N\_timesteps, N\_features, N\_outputs = trainx\_x.shape[1], trainx\_x.shape[2], trainy\_y.shape[1]  
 trainy\_y = trainy\_y.reshape((trainy\_y.shape[0], trainy\_y.shape[1], 1))  
  
 model = Sequential()  
 model.add(LSTM(200, activation='relu', input\_shape=(N\_timesteps, N\_features)))  
 model.add(RepeatVector(N\_outputs))  
 model.add(LSTM(200, activation='relu', return\_sequences=True))  
 model.add(TimeDistributed(Dense(100, activation='relu')))  
 model.add(TimeDistributed(Dense(1)))  
 model.compile(loss='mse', optimizer='adam')  
 model.fit(trainx\_x, trainy\_y, epochs=epochs, batch\_size=batch\_size, verbose=verbose)  
 return model

build the forcast and flatten the data , accumulate all the previous observations for input data and forecast the upcoming week

def forecas(model, histor, N\_input):  
 data = array(histor)  
 data = data.reshape((data.shape[0] \* data.shape[1], data.shape[2]))  
 i\_x = data[-N\_input:, :]  
 i\_x = i\_x.reshape((1, i\_x.shape[0], i\_x.shape[1]))  
 yhat = model.predict(i\_x, verbose=0)  
 yhat = yhat[0]  
 return yhat

evaluate model and store the predictions in the history to make further predictions and evaluate prediction for each day in a week.

def evaluat\_model(trainx, testy, N\_input):  
 model = build\_model(trainx, N\_input)  
 histor = [x for x in trainx]  
 prediction = list()  
 for i in range(len(testy)):  
 yhat\_sequence = forecas(model, histor, N\_input)  
 prediction.append(yhat\_sequence)  
 histor.append(testy[i, :])  
  
 prediction = array(prediction)  
 scores, score = evaluate\_forecast(testy[:, :, 0], prediction)  
 return scores, score

load the new dataset and select the no of inputs and summarize the score and plot the values.

datase = read\_csv('pvr.csv', header=0, infer\_datetime\_format=True, parse\_dates=['datetime'], index\_col=['datetime'])  
trainx, testy = split\_datase(datase.values)  
N\_input =14  
scores, score = evaluat\_model(trainx, testy, N\_input)  
summarize\_score('lstm', scores, score)  
days = ['sun', 'mon', 'tue', 'wed', 'thr', 'fri', 'sat']  
pyplot.plot(days, score, marker='o', label='lstm')  
pyplot.show()

WAIT FOR FIVE TO TEN MINUTES UNTILL THE CODE WILL BE EXECUTING.

the predicted solar energy gas consumption for the next seven days.