```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense, Activation, BatchNormalization
from keras.layers import Conv1D, Conv2D, Dropout, GRU, LSTM
from keras import backend as K
from sklearn.linear_model import LinearRegression

## loading the dataset
B3 = pd.read_excel(r'C:\Users\admin\Desktop\data\27.07.2019.new data.xlsx',sheet_name='B3')
#B6 = pd.read_excel(r'C:\Users\admin\Desktop\data\27.07.2019.new data.xlsx',sheet_name='B6')
#N3 = pd.read_excel(r'C:\Users\admin\Desktop\data\27.07.2019.new data.xlsx',sheet_name='N3')
#N6 = pd.read_excel(r'C:\Users\admin\Desktop\data\27.07.2019.new data.xlsx',sheet_name='N6')

x_tr = B3.loc[0:377, 0:4]
x_tr = np.array(x_tr)

x_te = B3.loc[378:541, 0:4]
x_te = np.array(x_te)

#from sklearn.utils import shuffle
#x_train, y_train = shuffle(x_train, y_train)
#x_test, y_test = shuffle(x_test, y_test)



x_train = np.zeros((378,10))
```

```
x_test = np.zeros((162,10))
for i in range(368):
    x_train[i,:] = x_tr[i:i+10, 0]

for i in range(10):
    x_train[i+368,:] = x_tr[i+359:i+359+10, 4]



for i in range(152):
    x_test[i,:] = x_te[i:i+10, 0]

for i in range(10):
    x_test[i+152,:] = x_te[i+143:i+143+10, 4]




x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],1)
y_train = B3.loc[0:377, 5]
y_train = y_train.reshape(y_train.shape[0],1,1)



x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],1)
y_test = B3.loc[378:541, 5]
y_test = y_test.reshape(y_test.shape[0],1,1)



## define model
```

```
epochs = 5
batch_size = 8

model = Sequential()
model.add(Conv1D(8, kernel_size=(5,), input_shape=(10,1)))
model.add(Conv1D(8, kernel_size=(6,)))
#model.add(Conv1D(16, kernel_size=(3,)))
model.add(LSTM(9, return_sequences=True))
model.add(LSTM(9, return_sequences=True))
model.add(LSTM(9, return_sequences=True))
model.add(Dense(8))
model.add(Dense(1))
model.summary()

#model.compile(loss='mse',
#              optimizer=optimizers.Adam(lr=0.01, decay=0.01),
#              metrics=['mae'])
#
#history = model.fit(x_train, y_train,
#                    batch_size=batch_size,
#                    epochs=epochs,
#                    verbose=1,
#                    )



from keras.callbacks import Callback

class MyCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
```

```
        lr = self.model.optimizer.lr
        decay = self.model.optimizer.decay
        iterations = self.model.optimizer.iterations
        #lr_with_decay = lr / (1. + decay * K.cast(iterations, K.dtype(decay)))
        #print(K.eval(lr_with_decay))
        print(K.eval(lr))

model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=0.004, decay=0.0),
              metrics=['mae'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          callbacks=[MyCallback()])



model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=0.001, decay=0.0),
              metrics=['mae'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          callbacks=[MyCallback()])
```

```
model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=0.0005, decay=0.0),
              metrics=['mae'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          callbacks=[MyCallback()])



model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=0.0001, decay=0.0),
              metrics=['mae'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          callbacks=[MyCallback()])




#score_train = model.evaluate(x_train, y_train, verbose=0)
#print('Train loss:', score_train[0])
#print('Train accuracy:', score_train[1])
#predict = model.predict(x_train)
```

```
score_test = model.evaluate(x_test, y_test, verbose=0)
print('Train loss:', score_test[0])
print('Train accuracy:', score_test[1])
predict_test = model.predict(x_test)

y_train = np.squeeze(y_train, axis=2)
y_test = np.squeeze(y_test, axis=2)
predict_test = np.squeeze(predict_test, axis=2)

plt.plot(y_test[0:100], 'b')
plt.plot(predict_test[0:100], 'r')
plt.grid()
plt.show()



#plt.scatter(y_train)
#plt.plot(predict, color='red')
#plt.show()
error = (predict_test-y_test)
for i in range(162):
    error[i,0] = error[i,0]*error[i,0]

print(np.sqrt(np.sum(error)/162))
```