```
# 1. Libraries and settings
import pandas as pd
import numpy as np
import math
import sklearn
import sklearn.preprocessing
import datetime
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape, GlobalAveragePooling1D
from keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D
from keras.utils import np_utils
#display parent directory and working directory
print(os.path.dirname(os.getcwd())+':', os.listdir(os.path.dirname(os.getcwd())));
print(os.getcwd()+':', os.listdir(os.getcwd()));
# 2. Analyze data
df = pd.read_csv("../../prices-split-adjusted.csv", index_col = 0)
print(df.info())
print(df.head())
print(df.values.shape)
# number of different stocks
print('\nnumber of different stocks: ', len(list(set(df.symbol))))
print(list(set(df.symbol))[:10])
df.tail()
df.describe()
#3.plot data
plt.figure(figsize=(15, 5));
plt.subplot(1,2,1);
plt.plot(df[df.symbol == 'EQIX'].open.values, color='red', label='open')
plt.plot(df[df.symbol == 'EQIX'].close.values, color='green', label='close')
plt.plot(df[df.symbol == 'EQIX'].low.values, color='blue', label='low')
plt.plot(df[df.symbol == 'EQIX'].high.values, color='black', label='high')
plt.title('stock price')
plt.xlabel('time [days]')
```

```
plt.ylabel('price')
plt.legend(loc='best')
#plt.show()

plt.subplot(1,2,2);
plt.plot(df[df.symbol == 'EQIX'].volume.values, color='black', label='volume')
plt.title('stock volume')
plt.xlabel('time [days]')
plt.ylabel('volume')
plt.legend(loc='best');
# 3. Manipulate data
#- choose a specific stock
#- drop feature: volume
#- normalize stock data
#- create train and test data sets

def feature_normalize(train):
    train_norm = train.apply(lambda x: (x - np.min(x)) / (np.max(x) - np.min(x)))    #
標準化(介於 0~1 之間)
    return train_norm

## 很重要　切割視窗
def create_segments_and_labels(df, time_steps, step):#, label_name):
    """
    This function receives a dataframe and returns the reshaped segments
    of x,y,z acceleration as well as the corresponding labels
    Args:
        df: Dataframe in the expected format
        time_steps: Integer value of the length of a segment that is created
    Returns:
        reshaped_segments
        labels:
    """
#圖畫中的 overlap 越高，代表資料中的相關性越強
#圖中 80 筆資料一次跳 40 筆，代表其並非相關性高
#feature 有四個
    N_FEATURES = 4
#選擇測試切出 20%
```

```
        test_set_size_percentage = 20
        segments = []
        labels = []

        data_raw = df.as_matrix()
#創造時間窗，將所有選擇特徵一起切割視窗
        for i in range(0, len(data_raw) - time_steps, step):#
                segments.append(data_raw[i: i + time_steps])

        segments = np.array(segments);
        test_set_size =
int(np.round(test_set_size_percentage/100*segments.shape[0]));
        train_set_size = segments.shape[0] - (test_set_size);
#以訓練資料占比分割訓練測試集，並以視窗最後一筆資料當作預測值
        x_train = segments[:train_set_size,:-1,:]
        y_train = segments[:train_set_size,-1,:]

#         x_valid = data[train_set_size:train_set_size+valid_set_size,:-1,:]
#         y_valid = data[train_set_size:train_set_size+valid_set_size,-1,:]

        x_test = segments[train_set_size:,:-1,:]
        y_test = segments[train_set_size:,-1,:]

        return [x_train, y_train, x_test, y_test]
#         return [x_train, y_train, x_valid, y_valid, x_test, y_test]


# choose one stock & drop volume
df_stock = df[df.symbol == 'EQIX'].copy()
df_stock.drop(['symbol'],1,inplace=True)
df_stock.drop(['volume'],1,inplace=True)

cols = list(df_stock.columns.values)
print('df_stock.columns.values = ', cols)

# normalize stock
df_stock_norm = df_stock.copy()
df_stock_norm = feature_normalize(df_stock_norm)
```

```
# create train, test data
time_steps = 20# choose sequence length
step = 5
x_train, y_train, x_test, y_test = create_segments_and_labels(df_stock_norm,
time_steps, step)
print('x_train.shape = ',x_train.shape)
print('y_train.shape = ', y_train.shape)
# print('x_valid.shape = ',x_valid.shape)
# print('y_valid.shape = ', y_valid.shape)
print('x_test.shape = ', x_test.shape)
print('y_test.shape = ',y_test.shape)

df_stock_norm.values.shape

plt.plot(df_stock_norm.open.values, color='red', label='open')
plt.plot(df_stock_norm.close.values, color='green', label='close')
plt.plot(df_stock_norm.low.values, color='blue', label='low')
plt.plot(df_stock_norm.high.values, color='black', label='high')
#plt.plot(df_stock_norm.volume.values, color='gray', label='volume')
plt.title('stock')
plt.xlabel('time [days]')
plt.ylabel('normalized price/volume')
plt.legend(loc='best')
plt.show()
#reshape 資料
num_time_periods, num_sensors = x_train.shape[1], x_train.shape[2]
input_shape = (num_time_periods*num_sensors)      ## 80*3   每一筆資料  80(時
間窗) 3 個變數( xyz)
x_train_reshape = x_train.reshape(x_train.shape[0], input_shape).astype('float32')
print(f"x_train_reshape.shape:{x_train_reshape.shape}")
x_test_reshape = x_test.reshape(x_test.shape[0], input_shape).astype('float32')
print(f"x_test_reshape.shape:{x_test_reshape.shape}")
```

#建立模型

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, LSTM, TimeDistributed,
RepeatVector, GRU
from keras.layers import SimpleRNN, Activation, Dense, RNN
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt
from keras.callbacks import ReduceLROnPlateau
%matplotlib inline

## build one To One Model(shape):

model_gru = Sequential()

model_gru.add(GRU(units=128, input_shape=( x_train.shape[1], x_train.shape[2])))
#units 隱藏層神經元個數
model_gru.add(Dropout(0.1))
model_gru.add(Dense(64, activation = 'relu'))
model_gru.add(Dropout(0.1))
model_gru.add(Dense(16, activation = 'relu'))#三層隱藏層
model_gru.add(Dropout(0.1))
model_gru.add(Dense(num_classes, activation='softmax'))
#num_classes 不能改
```

#開始訓練

```
model_gru.compile(loss='MSE',
                  optimizer='adam', metrics=['mse'])
model_gru.summary()


learning_rate_function = ReduceLROnPlateau(monitor='mean_squared_error',
                                           patience=3, #準確率重複 3 次
就要減少
                                           verbose=1,
                                           factor=0.5, #準確率乘上
```

```
factor 設成下一個 learning_rate
                                                    min_lr=0.00001)


print("\n--- Fit the model ---\n")

train_history = model_gru.fit(x=x_train, y= y_train, validation_split=0.18,
epochs=200,
batch_size=10,callbacks=[learning_rate_function],verbose=2)#callbacks=[learning_ra
te_function], verbose=2)


print("\n--- Learning curve of model training ---\n")

get_ipython().magic('matplotlib inline')
## IPython 有一組預先定義好的所謂的魔法函數（Magic Functions），你可以通
過命令列的語法形式來訪問它們。
#繪圖
import matplotlib.pyplot as plt
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title("Train History")
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.show()


show_train_history(train_history, "mean_squared_error", "val_loss") ##  訓練正確
率圖

scorelstm = model_gru.evaluate(x_test, y_test)
print(f"MSE:{scores[0]}")
```