Python X 金融分析

https://ithelp.ithome.com.tw/users/20103826/ironman/3032?page=1

傳統統計方法請自學，任何問題可討論

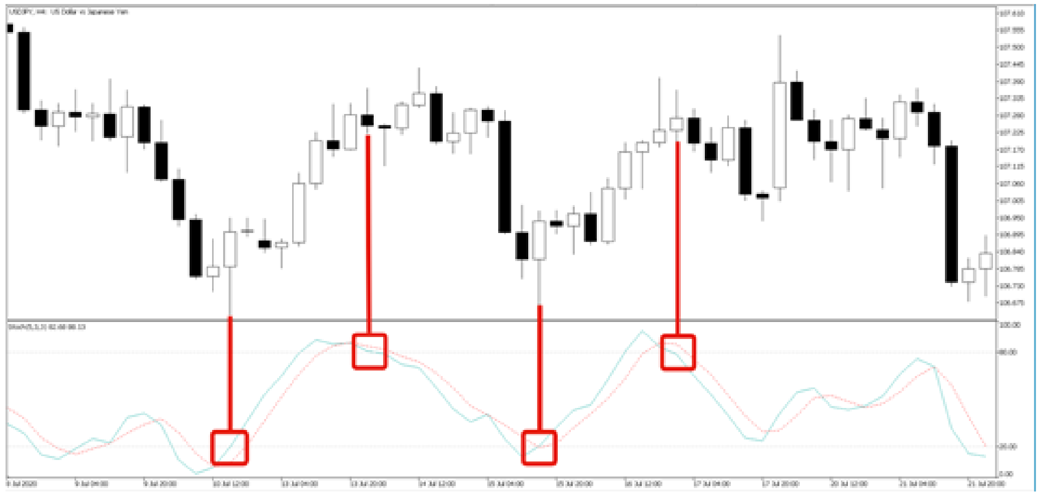
技術分析教學

KD隨機指標的英文為「Stochastic Oscillator」，翻譯為「推算統計學上的指標」。推算統計學聽起來很複雜，講白了就是「以一定期間的最高價與最低價為基準，判斷收盤價的水準」的一種指標。

KD隨機指標（Stochastic Oscillator）中包含了％K、％D、Slow％D三條線，按照組合方式又稱作「快速隨機指標」（％K與％D）以及「慢速隨機指標」（％D與慢速%D）。

分析KD隨機指標時，若低於20％以下判斷為超賣，超過80%以上則為超買。同時，觀察快速變動的部份（快速：％K、慢速：％D），在突破區間時（Zone exit）應該就是逆向操作的有效時機。

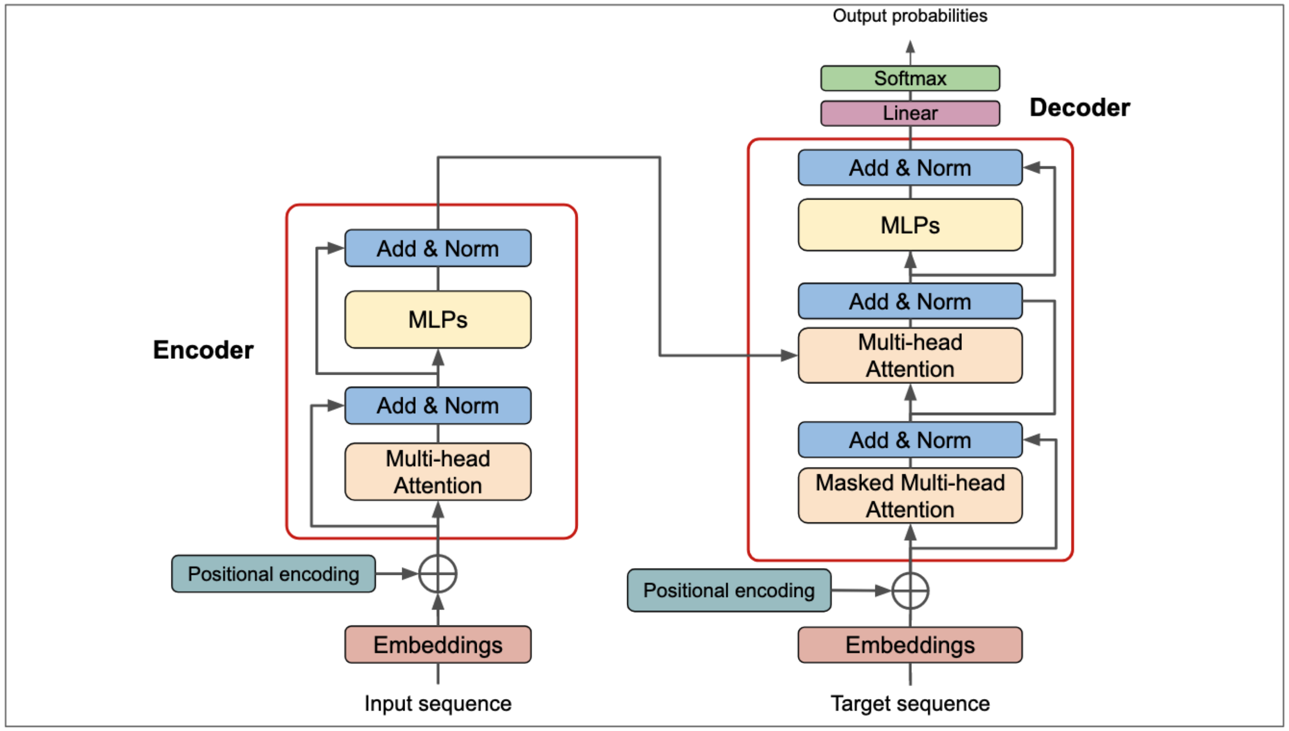
Day 26



高雄大學研究「生成式AI」分析台股漲跌勢準確度達8成

https://udn.com/news/story/6928/8101330

用AI找到最佳進場時間？以Transformer 預測台灣指數期貨上漲與下跌波段實做範例



https://edge.aif.tw/futures/

Need TensorFlow

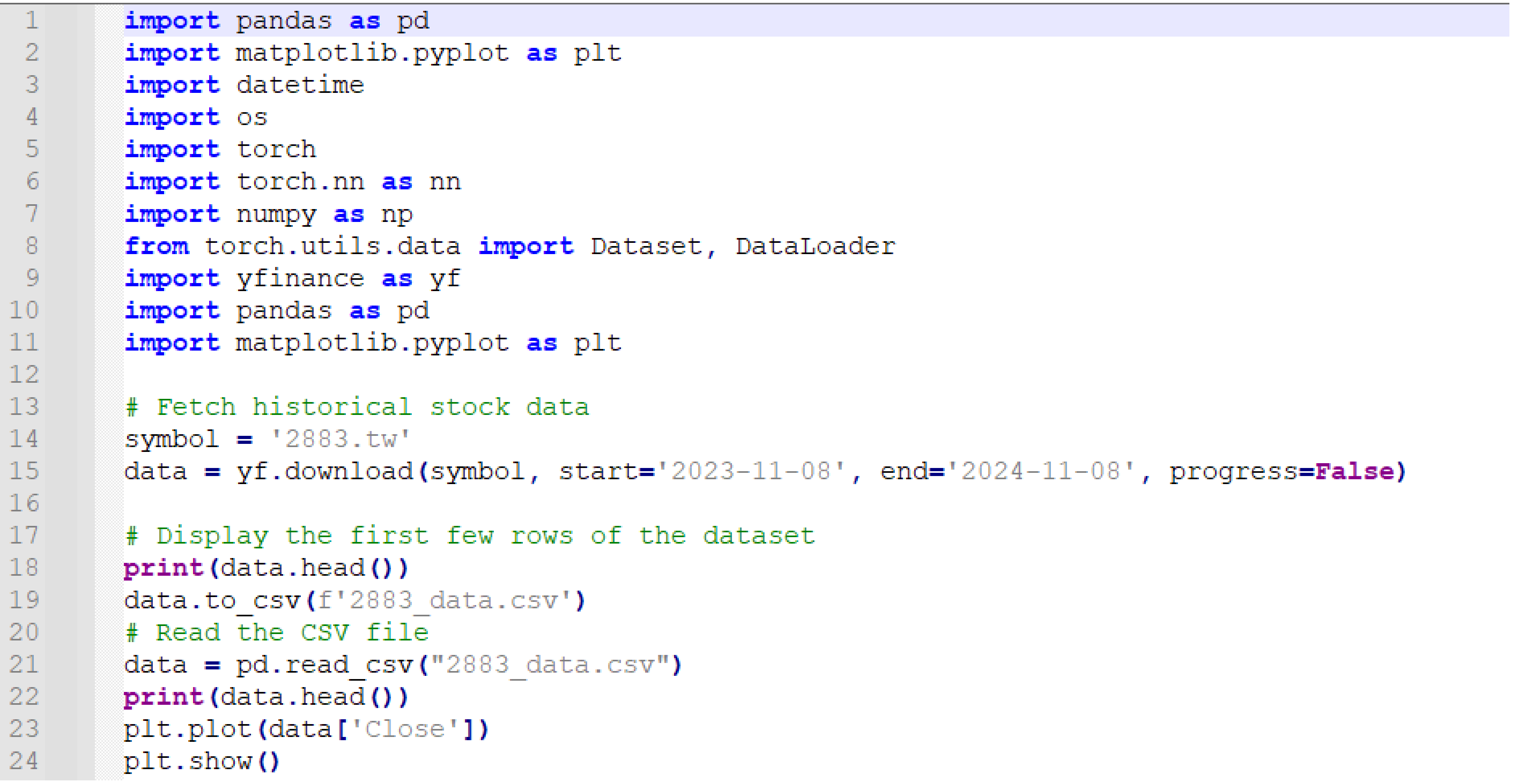
人工智慧股票交易機器人名單中名列前茅的是 交易建議

主要特點:

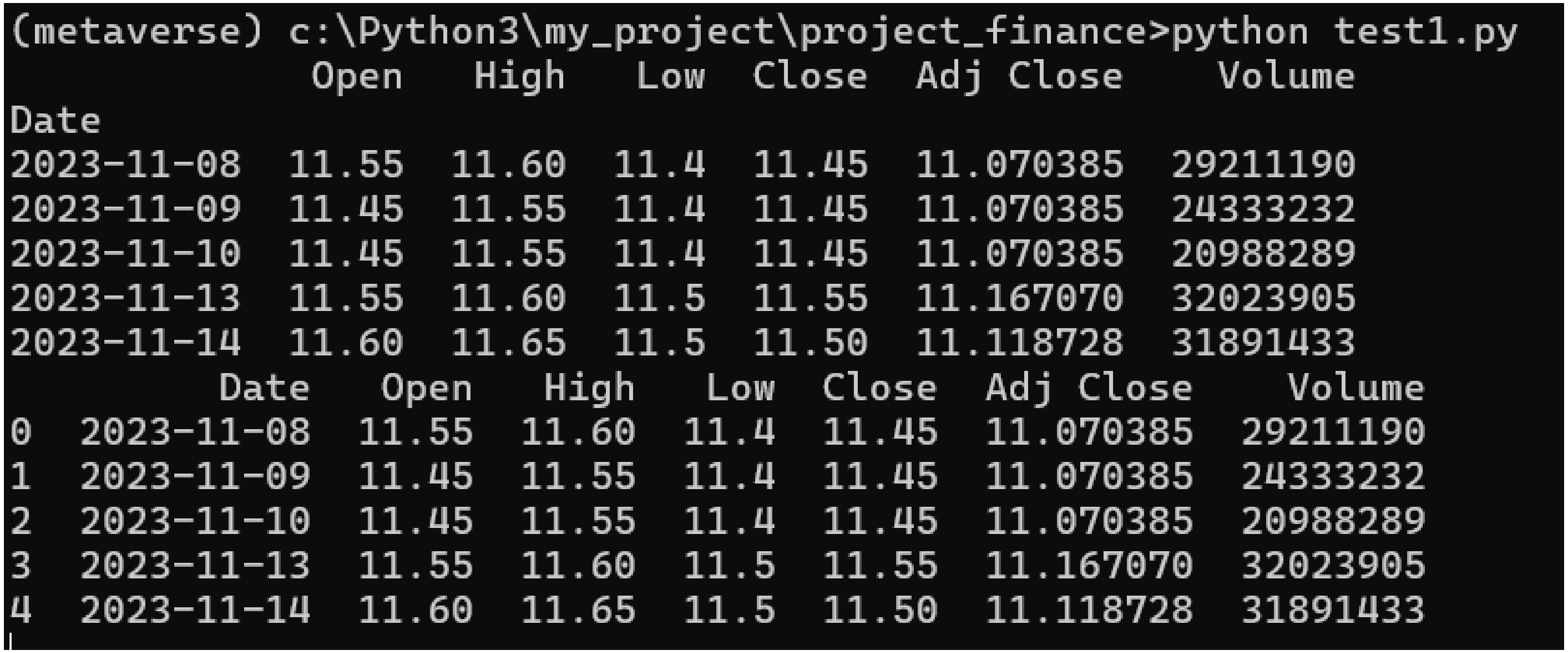
- 人工智能算法
- 模擬訓練
- 進場和出場訊號

LSTM 是個入門案例

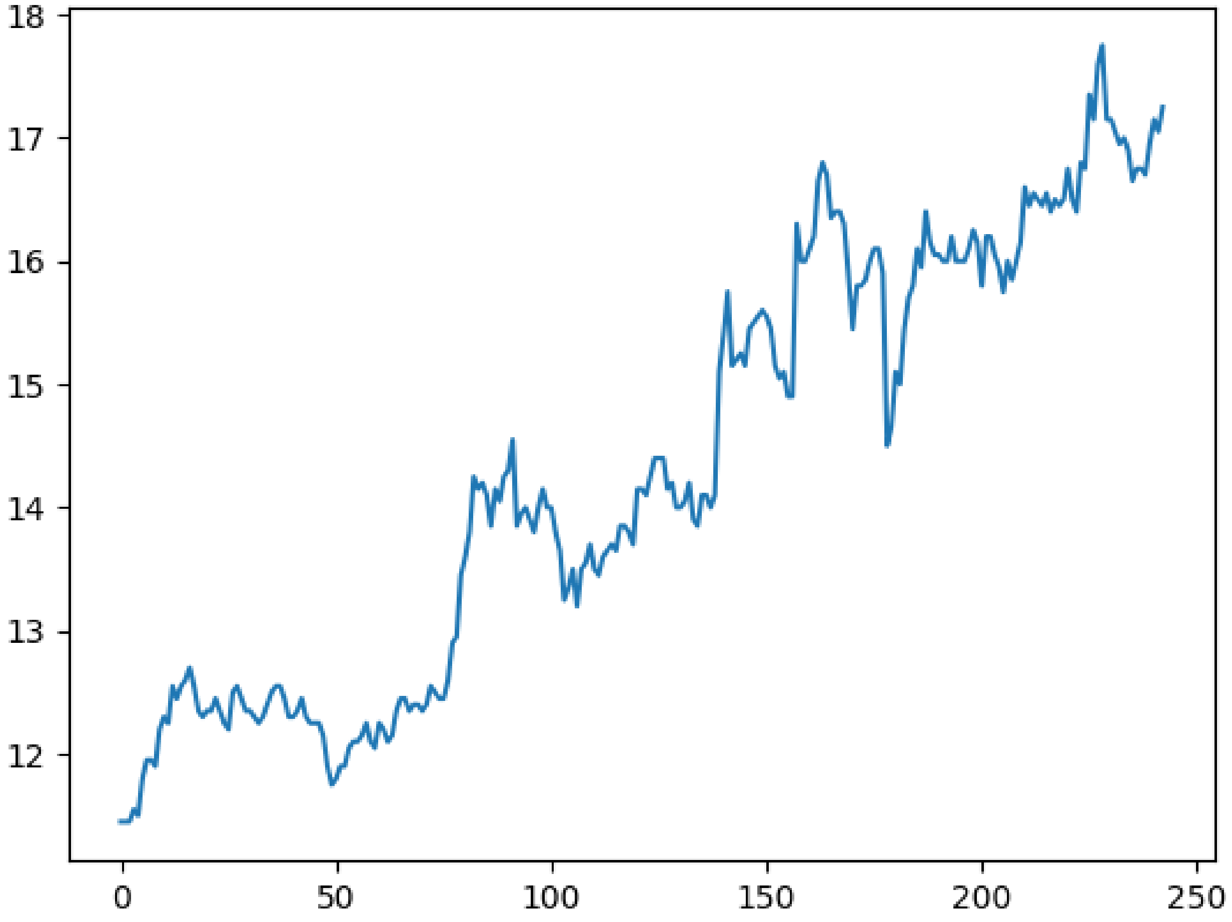
Train_lstm.py
Infer_lstm.py

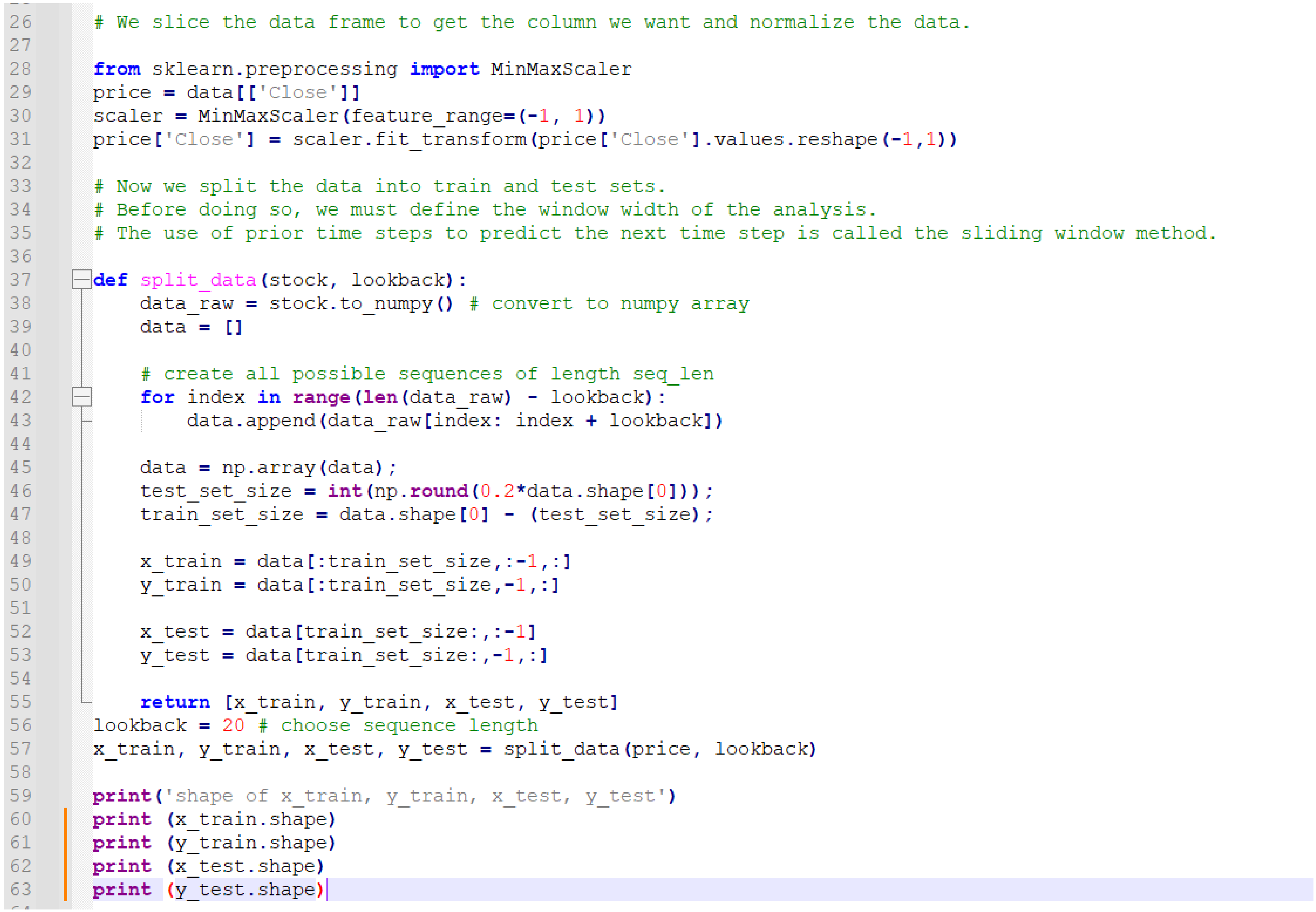


```python
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import os
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Fetch historical stock data
symbol = '2883.tw'
data = yf.download(symbol, start='2023-11-08', end='2024-11-08', progress=False)

# Display the first few rows of the dataset
print(data.head())
data.to_csv(f'2883_data.csv')
# Read the CSV file
data = pd.read_csv("2883_data.csv")
print(data.head())
plt.plot(data['Close'])
plt.show()
```

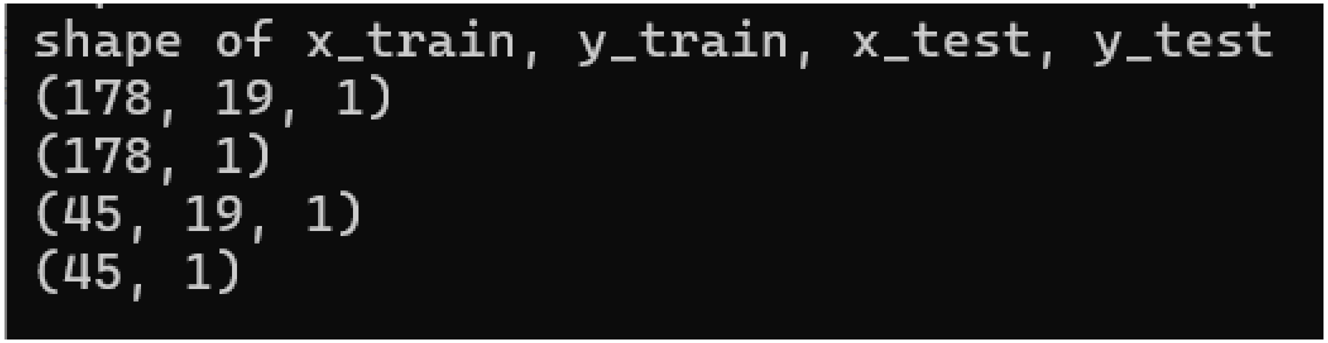


```
(metaverse) c:\Python3\my_project\project_finance>python test1.py
             Open   High   Low  Close  Adj Close    Volume
Date
2023-11-08  11.55  11.60  11.4  11.45  11.070385  29211190
2023-11-09  11.45  11.55  11.4  11.45  11.070385  24333232
2023-11-10  11.45  11.55  11.4  11.45  11.070385  20988289
2023-11-13  11.55  11.60  11.5  11.55  11.167070  32023905
2023-11-14  11.60  11.65  11.5  11.50  11.118728  31891433
         Date   Open   High   Low  Close  Adj Close    Volume
0  2023-11-08  11.55  11.60  11.4  11.45  11.070385  29211190
1  2023-11-09  11.45  11.55  11.4  11.45  11.070385  24333232
2  2023-11-10  11.45  11.55  11.4  11.45  11.070385  20988289
3  2023-11-13  11.55  11.60  11.5  11.55  11.167070  32023905
4  2023-11-14  11.60  11.65  11.5  11.50  11.118728  31891433
```

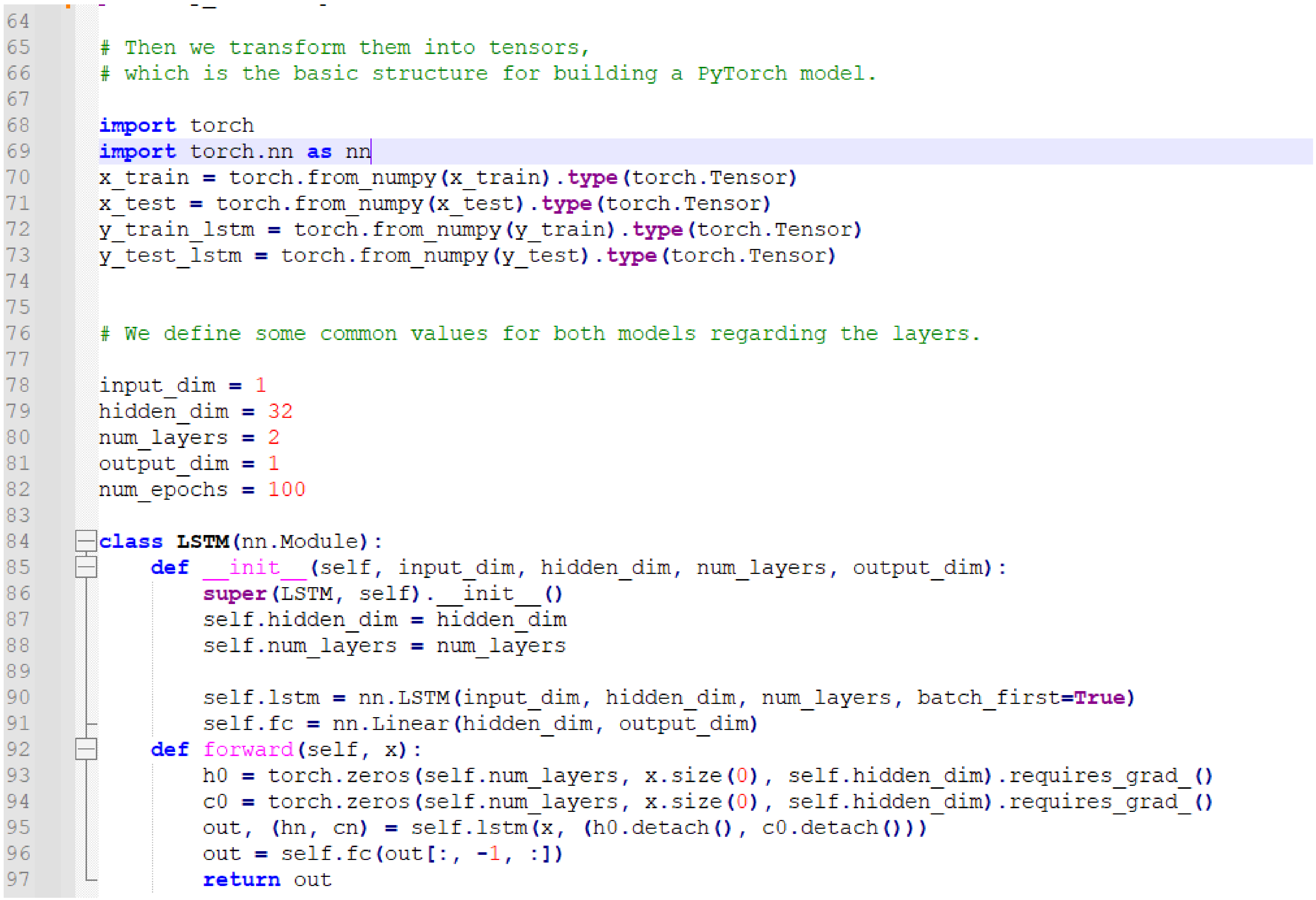

18
17
16
15
14
13
12
0
50
100
150
200
250



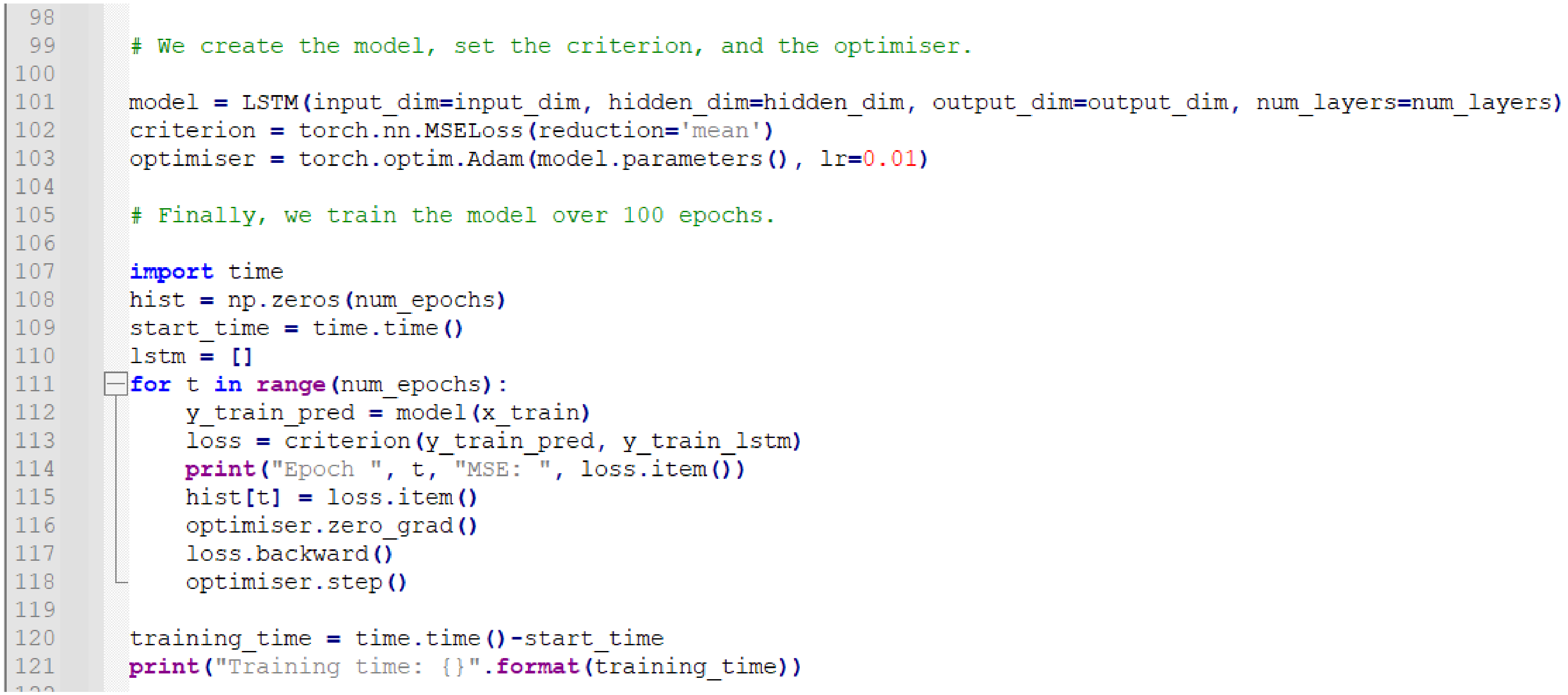
```python
# We slice the data frame to get the column we want and normalize the data.

from sklearn.preprocessing import MinMaxScaler
price = data[['Close']]
scaler = MinMaxScaler(feature_range=(-1, 1))
price['Close'] = scaler.fit_transform(price['Close'].values.reshape(-1,1))

# Now we split the data into train and test sets.
# Before doing so, we must define the window width of the analysis.
# The use of prior time steps to predict the next time step is called the sliding window method.

def split_data(stock, lookback):
    data_raw = stock.to_numpy() # convert to numpy array
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    test_set_size = int(np.round(0.2*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

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

    return [x_train, y_train, x_test, y_test]
lookback = 20 # choose sequence length
x_train, y_train, x_test, y_test = split_data(price, lookback)

print('shape of x_train, y_train, x_test, y_test')
print (x_train.shape)
print (y_train.shape)
print (x_test.shape)
print (y_test.shape)
```



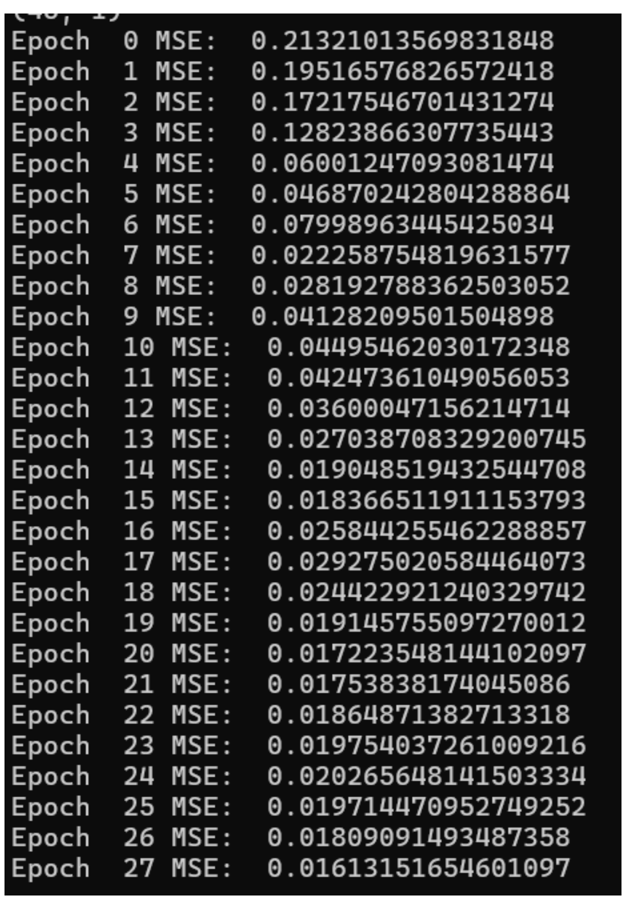
```
shape of x_train, y_train, x_test, y_test
(178, 19, 1)
(178, 1)
(45, 19, 1)
(45, 1)
```



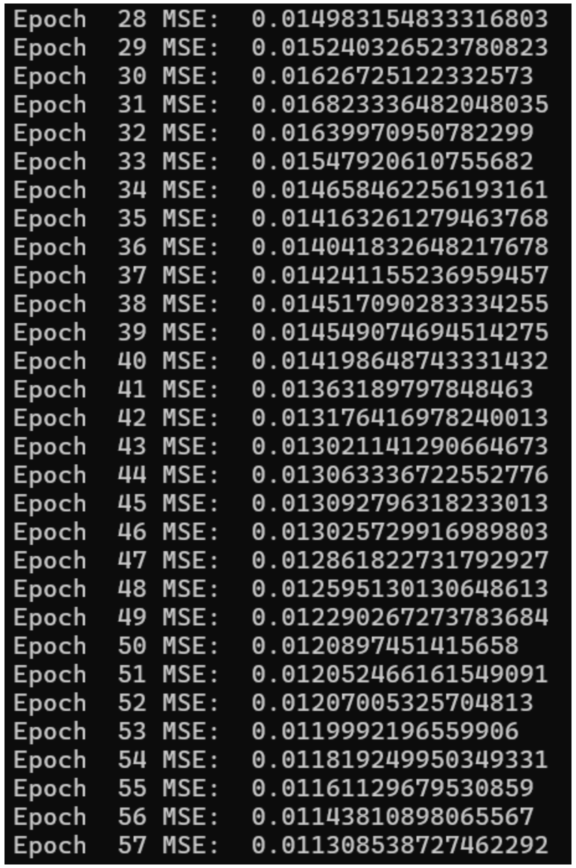
```python

# Then we transform them into tensors,
# which is the basic structure for building a PyTorch model.

import torch
import torch.nn as nn
x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test).type(torch.Tensor)
y_train_lstm = torch.from_numpy(y_train).type(torch.Tensor)
y_test_lstm = torch.from_numpy(y_test).type(torch.Tensor)


# We define some common values for both models regarding the layers.

input_dim = 1
hidden_dim = 32
num_layers = 2
output_dim = 1
num_epochs = 100

class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out
```



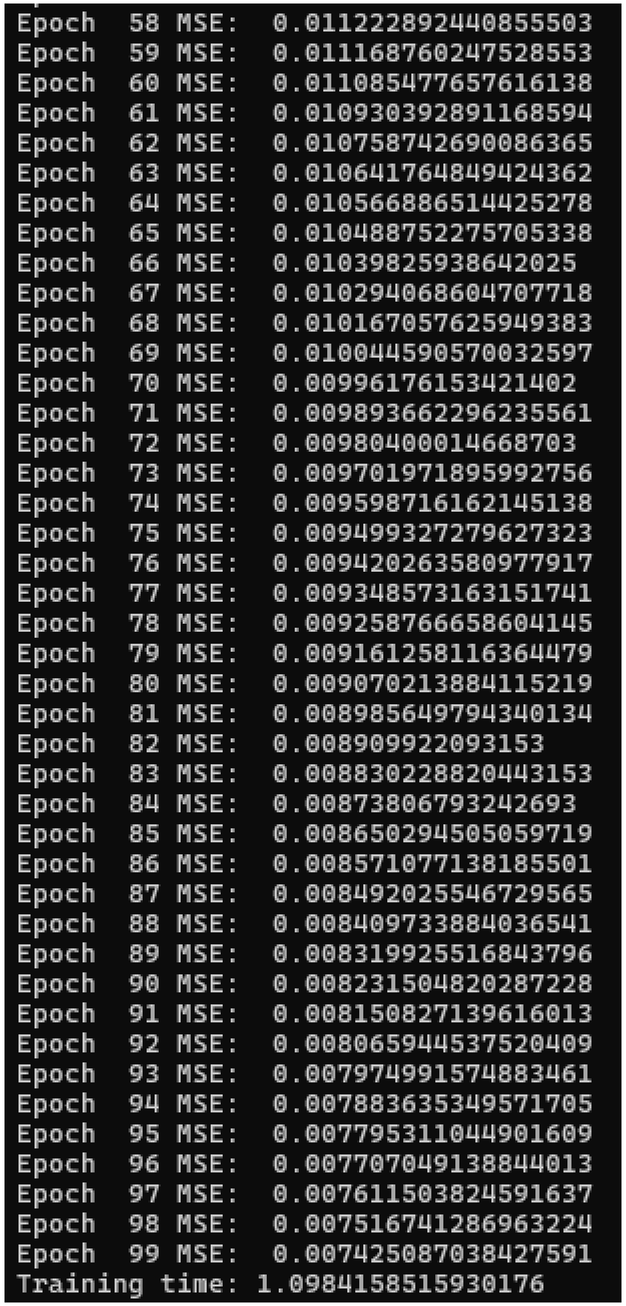
```python

# We create the model, set the criterion, and the optimiser.

model = LSTM(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim, num_layers=num_layers)
criterion = torch.nn.MSELoss(reduction='mean')
optimiser = torch.optim.Adam(model.parameters(), lr=0.01)

# Finally, we train the model over 100 epochs.

import time
hist = np.zeros(num_epochs)
start_time = time.time()
lstm = []
for t in range(num_epochs):
    y_train_pred = model(x_train)
    loss = criterion(y_train_pred, y_train_lstm)
    print("Epoch ", t, "MSE: ", loss.item())
    hist[t] = loss.item()
    optimiser.zero_grad()
    loss.backward()
    optimiser.step()

training_time = time.time()-start_time
print("Training time: {}".format(training_time))
```



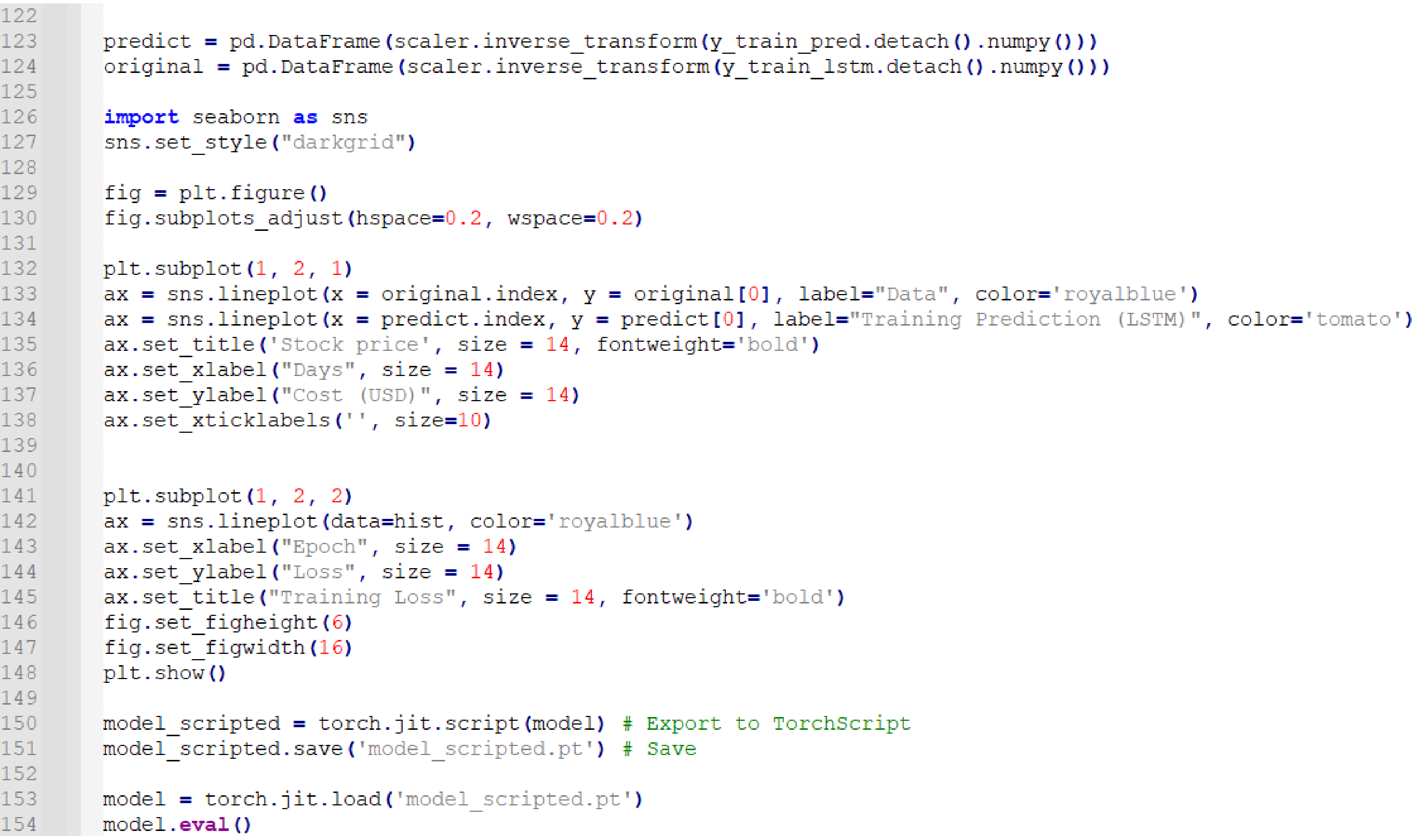
```
(48, 1)
Epoch  0 MSE:  0.21321013569831848
Epoch  1 MSE:  0.19516576826572418
Epoch  2 MSE:  0.17217546701431274
Epoch  3 MSE:  0.12823866307735443
Epoch  4 MSE:  0.06001247093081474
Epoch  5 MSE:  0.046870242804288864
Epoch  6 MSE:  0.07998963445425034
Epoch  7 MSE:  0.022258754819631577
Epoch  8 MSE:  0.028192788362503052
Epoch  9 MSE:  0.04128209501504898
Epoch  10 MSE:  0.04495462030172348
Epoch  11 MSE:  0.04247361049056053
Epoch  12 MSE:  0.03600047156214714
Epoch  13 MSE:  0.027038708329200745
Epoch  14 MSE:  0.019048519432544708
Epoch  15 MSE:  0.018366511911153793
Epoch  16 MSE:  0.025844255462288857
Epoch  17 MSE:  0.029275020584464073
Epoch  18 MSE:  0.024422921240329742
Epoch  19 MSE:  0.019145755097270012
Epoch  20 MSE:  0.017223548144102097
Epoch  21 MSE:  0.01753838174045086
Epoch  22 MSE:  0.01864871382713318
Epoch  23 MSE:  0.019754037261009216
Epoch  24 MSE:  0.020265648141503334
Epoch  25 MSE:  0.019714470952749252
Epoch  26 MSE:  0.01809091493487358
Epoch  27 MSE:  0.01613151654601097
```



```
Epoch  28 MSE:  0.014983154833316803
Epoch  29 MSE:  0.015240326523780823
Epoch  30 MSE:  0.01626725122332573
Epoch  31 MSE:  0.016823336482048035
Epoch  32 MSE:  0.01639970950782299
Epoch  33 MSE:  0.01547920610755682
Epoch  34 MSE:  0.014658462256193161
Epoch  35 MSE:  0.014163261279463768
Epoch  36 MSE:  0.014041832648217678
Epoch  37 MSE:  0.014241155236959457
Epoch  38 MSE:  0.014517090283334255
Epoch  39 MSE:  0.014549074694514275
Epoch  40 MSE:  0.014198648743331432
Epoch  41 MSE:  0.01363189797848463
Epoch  42 MSE:  0.013176416978240013
Epoch  43 MSE:  0.013021141290664673
Epoch  44 MSE:  0.013063336722552776
Epoch  45 MSE:  0.013092796318233013
Epoch  46 MSE:  0.013025729916989803
Epoch  47 MSE:  0.012861822731792927
Epoch  48 MSE:  0.012595130130648613
Epoch  49 MSE:  0.012290267273783684
Epoch  50 MSE:  0.01208974514156558
Epoch  51 MSE:  0.012052466161549091
Epoch  52 MSE:  0.01207005325704813
Epoch  53 MSE:  0.01199921965559906
Epoch  54 MSE:  0.011819249950349331
Epoch  55 MSE:  0.01161129679530859
Epoch  56 MSE:  0.01143810898065567
Epoch  57 MSE:  0.011308538727462292
```



```
Epoch  58 MSE:  0.011222892440855503
Epoch  59 MSE:  0.011168760247528553
Epoch  60 MSE:  0.011085477657616138
Epoch  61 MSE:  0.010930392891168594
Epoch  62 MSE:  0.010758742690086365
Epoch  63 MSE:  0.010641764849424362
Epoch  64 MSE:  0.010566886514425278
Epoch  65 MSE:  0.010488752275705338
Epoch  66 MSE:  0.01039825938642025
Epoch  67 MSE:  0.010294068604707718
Epoch  68 MSE:  0.010167057625949383
Epoch  69 MSE:  0.010044590570032597
Epoch  70 MSE:  0.00996176153421402
Epoch  71 MSE:  0.009893662296235561
Epoch  72 MSE:  0.00980400014668703
Epoch  73 MSE:  0.009701971895992756
Epoch  74 MSE:  0.009598716162145138
Epoch  75 MSE:  0.009499327279627323
Epoch  76 MSE:  0.009420263580977917
Epoch  77 MSE:  0.009348573163151741
Epoch  78 MSE:  0.009258766658604145
Epoch  79 MSE:  0.009161258116364479
Epoch  80 MSE:  0.009070213884115219
Epoch  81 MSE:  0.008985649794340134
Epoch  82 MSE:  0.00890992209315 3
Epoch  83 MSE:  0.008830228820443153
Epoch  84 MSE:  0.00873806793242693
Epoch  85 MSE:  0.008650294505059719
Epoch  86 MSE:  0.008571077138185501
Epoch  87 MSE:  0.008492025546729565
Epoch  88 MSE:  0.008409733884036541
Epoch  89 MSE:  0.008319925516843796
Epoch  90 MSE:  0.008231504820287228
Epoch  91 MSE:  0.008150827139616013
Epoch  92 MSE:  0.008065944537520409
Epoch  93 MSE:  0.007974991574883461
Epoch  94 MSE:  0.007883635349571705
Epoch  95 MSE:  0.007795311044901609
Epoch  96 MSE:  0.007707049138844013
Epoch  97 MSE:  0.007611503824591637
Epoch  98 MSE:  0.007516741286963224
Epoch  99 MSE:  0.007425087038427591
Training time: 1.0984158515930176
```



```

predict = pd.DataFrame(scaler.inverse_transform(y_train_pred.detach().numpy()))
original = pd.DataFrame(scaler.inverse_transform(y_train_lstm.detach().numpy()))

import seaborn as sns
sns.set_style("darkgrid")

fig = plt.figure()
fig.subplots_adjust(hspace=0.2, wspace=0.2)

plt.subplot(1, 2, 1)
ax = sns.lineplot(x = original.index, y = original[0], label="Data", color='royalblue')
ax = sns.lineplot(x = predict.index, y = predict[0], label="Training Prediction (LSTM)", color='tomato')
ax.set_title('Stock price', size = 14, fontweight='bold')
ax.set_xlabel("Days", size = 14)
ax.set_ylabel("Cost (USD)", size = 14)
ax.set_xticklabels('', size=10)


plt.subplot(1, 2, 2)
ax = sns.lineplot(data=hist, color='royalblue')
ax.set_xlabel("Epoch", size = 14)
ax.set_ylabel("Loss", size = 14)
ax.set_title("Training Loss", size = 14, fontweight='bold')
fig.set_figheight(6)
fig.set_figwidth(16)
plt.show()

model_scripted = torch.jit.script(model) # Export to TorchScript
model_scripted.save('model_scripted.pt') # Save

model = torch.jit.load('model_scripted.pt')
model.eval()
```