智慧化企業整合 Project2

利用 LSTM 長短期記憶演算法 - 建立股票預測模型

Group 2

108034532 徐紫芸

108034544 楊雨澄

108034547 梁茲晴

108034554 張佳琳

目錄

一、主題介紹

（一）動機

股價是由幾個因素決定的，歷史股價的評估只是其中的一小部分。因此，預測股市走向行為是一個非常棘手的問題。使用機器學習用來投資已經成為一種很熱門的議題，利用 LSTM 可以儲存自訂股價期數，適合處理時間序列預測問題，而股票市場的數據由於格式完整且非常容易獲得，若要預測股票實際價格是不可能的事，因此我們藉由 LSTM 方法來建立模型預測股票價格是上漲或下跌的趨勢。

（二）目的

一般投資人的買賣直覺就是買低賣高，賺取差價，所以相對其他投資工具來說股票是較容易入門上手，利用 2012-2017 資料預測 2018 的股價，目的是為了判斷股票現行股價的價位是否合理，並描繪出它長遠的發展趨勢，進而提供投資客良好的預測模型。

（三）5W1H

WHAT： 模擬出預測之股價走勢圖

多面向預測，股票是由全球趨勢跟投資客共同影響，我們將分為：

a.多對多，長時間預測，以 6 年預測 6 年(2012-2017)(長期趨勢預測)

b.多對多，以 6 年預測 1 個月/1 年 (未來預測)

c.單點分析，用 6 年預測 1 天(點預測)(單一特徵/多特徵)

WHY : 投資是為了財富持續穩定的增值，而不是指望一夜暴富。投資講究的是計劃性和穩健性，透過穩定的預測模型可以了解股票增長的趨勢。

WHERE : 股票市場，以 APPLE 股票為例。

WHEN : 股票的進場退場時間是非常重要的，正確的邏輯、專業的預測、準確的時機，如果時機不正確，那極有可能陷入死胡同，反而轉盈為虧。

WHO : 任何想投資 APPLE 股票的投資客。投資講究的是合作的雙贏，面對股票的走勢，是否能更給予專業的判斷，是否能夠綜合各方面的因素给予一個合理的分析。

HOW : 以長短期記憶演算法做預測模型，建立低誤差的架構以提供正確的預測，讓投資者能掌握股票漲跌趨勢。

二、資料來源

(一) 研究資料來源

1. APPLE 歷史股價，資料下載於 Yahoo 奇摩股市，如下圖所示

Date	Open	High	Low	Close	Adj Close	Volume
2012/1/3	58.48571	58.92857	58.42857	58.74714	51.11594	75555200
2012/1/4	58.57143	59.24	58.46857	59.06286	51.39065	65005500
2012/1/5	59.27857	59.79286	58.95286	59.71857	51.96119	67817400
2012/1/6	59.96714	60.39286	59.88857	60.34286	52.50438	79573200
2012/1/9	60.78571	61.10714	60.19286	60.24714	52.42109	98506100
2012/1/10	60.84428	60.85714	60.21429	60.46286	52.60879	64549100
2012/1/11	60.38286	60.40714	59.90143	60.36429	52.52303	53771200

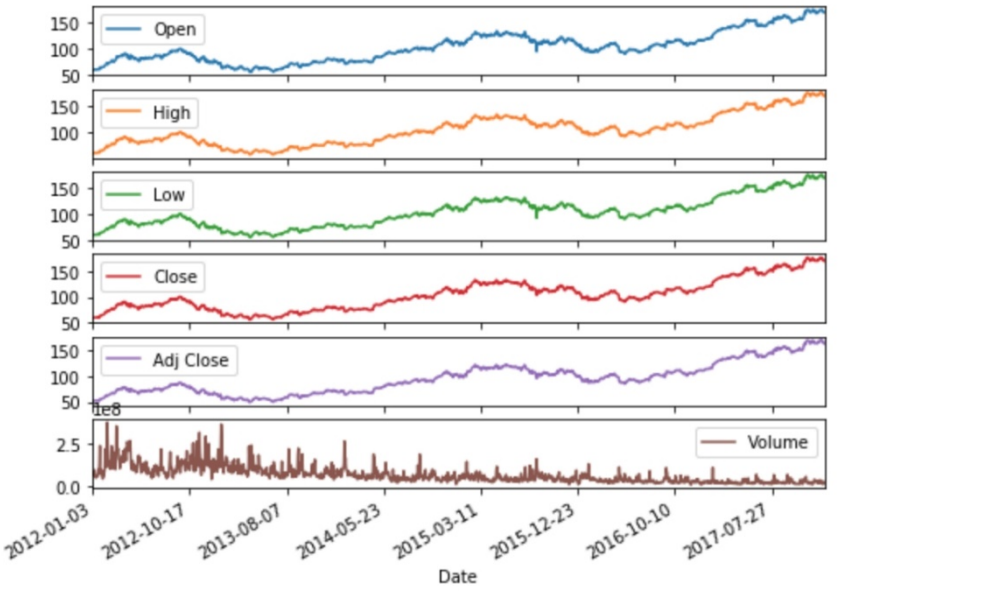
資料走勢圖

```
[26] features_considered = ['Open', 'High', 'Low' ,'Close', 'Adj Close', 'Volume']
```

```
[27] features = df[features_considered]
     features.index = df['Date']
     features.head()
```

```
features.plot(subplots=True)
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fdd9b4bc4e0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fdd9b403668>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fdd9b5e6f98>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fdd963ac438>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fdd96357898>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fdd96389cf8>],
      dtype=object)
```



2. 黃金價，下載於「黃金價格資訊站」，如下圖所示

日期	最新	開市	高	低	成交量	更改%
2017年12月29日	1,362.60	1,362.60	1,362.60	1,362.60	0.02K	0.93%
2017年12月28日	1,350.10	1,346.00	1,346.40	1,343.30	0.13K	0.53%
2017年12月27日	1,343.00	1,341.00	1,341.00	1,341.00	0.00K	0.28%
2017年12月26日	1,339.20	1,337.00	1,337.90	1,337.00	0.00K	4.80%
2017年12月25日	1,277.90	1,278.90	1,279.50	1,277.90	-	-3.95%
2017年12月22日	1,330.50	1,322.00	1,329.10	1,322.00	0.00K	0.64%
2017年12月21日	1,322.00	1,319.50	1,319.50	1,319.50	0.03K	0.07%
2017年12月20日	1,321.10	1,317.90	1,321.80	1,317.90	0.02K	0.43%
2017年12月19日	1,315.50	1,315.50	1,315.50	1,315.50	0.01K	-0.06%

3. 油價，下載於「Investing.com」，如下圖所示

日期	最新	開市	高	低	成交量	更改%
2017年12月29日	60.42	59.91	60.51	59.82	464.48K	0.97%
2017年12月28日	59.84	59.53	59.94	59.44	345.96K	0.34%
2017年12月27日	59.64	59.79	59.93	59.33	402.40K	-0.55%
2017年12月26日	59.97	58.4	60.01	58.32	437.79K	2.36%
2017年12月25日	58.59	58.41	58.62	58.38	-	0.21%
2017年12月22日	58.47	58.21	58.5	57.87	324.55K	0.19%
2017年12月21日	58.36	58.02	58.38	57.63	405.32K	0.46%
2017年12月20日	58.09	57.66	58.12	57.44	420.84K	1.10%
2017年12月19日	57.46	57.3	57.64	57.16	36.29K	0.52%
2017年12月18日	57.16	57.37	57.78	56.82	137.15K	-0.24%

本組討論後認為黃金價、油價對國際股價可能有關係，因此我們加入黃金價跟油價來分析探討是否有影響的趨勢。

（二）資料前處理

A. 多對多，長時間預測，以 6 年預測 6 年(2012-2017)

1. 匯入套件

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

2. 讀取”TSLA_2012-2017.CSV”資料

```
df = pd.read_csv("TSLA_2012-2017.csv")
df
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2012-01-03	28.940001	29.500000	27.650000	28.080000	28.080000	928100
1	2012-01-04	28.209999	28.670000	27.500000	27.709999	27.709999	630100
2	2012-01-05	27.760000	27.930000	26.850000	27.120001	27.120001	1005500
3	2012-01-06	27.200001	27.790001	26.410000	26.910000	26.910000	986300
4	2012-01-09	27.000000	27.490000	26.120001	27.250000	27.250000	897000
...	...	...	...	...	...	...	...
1504	2017-12-22	329.510010	330.920013	324.820007	325.200012	325.200012	4215800
1505	2017-12-26	323.829987	323.940002	316.579987	317.290009	317.290009	4378400
1506	2017-12-27	316.000000	317.679993	310.750000	311.640015	311.640015	4712100
1507	2017-12-28	311.750000	315.820007	309.540009	315.359985	315.359985	4316300
1508	2017-12-29	316.179993	316.410004	310.000000	311.350006	311.350006	3777200

1509 rows × 7 columns

3. 選取要使用的資料

於 1509 筆資料中只選取前 1508 筆(偶數較好分)。由於在選取資料時我們做些抉擇，第一次時將 1508 比分成 52 組資料，每組有 29 筆資料，並拆成 input 跟 output，因此 xraw 20 筆、yraw 9 筆，意思是用過去 20 天預測 9 天(多對多)，得到的結果唯有趨勢但準度並不高。

因此第二次我們將 1508 筆資料拆成 13 組，每組有 116 筆資料，且 13 組中分為 10 個 input、3 個 output，意思是用過去 10 天預測 3 天(多對多)，如此一來得到的結果我們認為是較準確的。

```
[ ]  values = df.iloc[:-1,1:-1] #取1508個而已
     values = values.values
     values
```

```
array([[ 28.940001,  29.5     ,  27.65    ,  28.08    ,  28.08    ],
       [ 28.209999,  28.67    ,  27.5     ,  27.709999,  27.709999],
       [ 27.76    ,  27.93    ,  26.85    ,  27.120001,  27.120001],
       ...,
       [323.829987, 323.940002, 316.579987, 317.290009, 317.290009],
       [316.      , 317.679993, 310.75    , 311.640015, 311.640015],
       [311.75    , 315.820007, 309.540009, 315.359985, 315.359985]])
```

4. Normalization

```
[ ]  #normalization 標準化
     values = (values - values.min()) / (values.max() - values.min())
```

```
[ ]  values
```

```
array([[0.01716762, 0.01869363, 0.01365235, 0.0148241 , 0.0148241 ],
       [0.01517835, 0.01643186, 0.01324359, 0.01381584, 0.01381584],
       [0.0139521 , 0.01441535, 0.01147233, 0.01220809, 0.01220809],
       ...,
       [0.82074829, 0.82104808, 0.8009919 , 0.80292673, 0.80292673],
       [0.79941143, 0.80398944, 0.78510508, 0.78753039, 0.78753039],
       [0.7878301 , 0.79892094, 0.78180783, 0.79766738, 0.79766738]])
```

5. 分資料

LSTM reshape(116, 5(項目）, 10（time-step))

```
## data cleaning
xraw = []
yraw = []
for t in range(116):
  for i in range(13):
    if(i < 10):
      xraw.append(values[t + i])
    else:
      yraw.append(values[t + i])
```

```
[ ]  xraw = np.array(xraw)
     yraw = np.array(yraw)

     xraw = xraw.reshape(116, 5, 10)
     yraw = yraw.reshape(116, 5, 3)

     print(xraw.shape, yraw.shape)
```

```
(116, 5, 10) (116, 5, 3)
```

B. 多對多，以 6 年預測 1 個月/1 年(兩個方式相同僅介紹一次)

1. 匯入套件

```
[ ]  #import the libraries
     import numpy as np
     import matplotlib.pyplot as plt #for drawing
     import pandas as pd
```

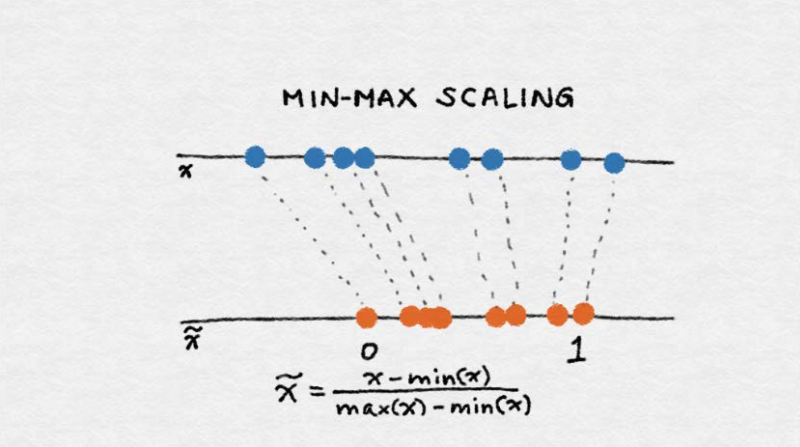
2. 讀取" AAPL_2012-2017.csv" 資料

```
[ ] #import the training set
    dataset_train = pd.read_csv('AAPL_2012-2017.csv') #read training data
    training_set = dataset_train.iloc[:,1:2].values #reach open
    dataset_train.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2012-01-03	58.485714	58.928570	58.428570	58.747143	51.115936	75555200
1	2012-01-04	58.571430	59.240002	58.468571	59.062859	51.390648	65005500
2	2012-01-05	59.278572	59.792858	58.952858	59.718571	51.961189	67817400
3	2012-01-06	59.967144	60.392857	59.888573	60.342857	52.504375	79573200
4	2012-01-09	60.785713	61.107143	60.192856	60.247143	52.421093	98506100

3. Feature scaling，將資料壓縮在 [0,1] 之間：

```
[ ] #Feature Scaling 限定到0-1
    from sklearn import preprocessing
    from sklearn.preprocessing import MinMaxScaler
    sc = MinMaxScaler(feature_range = (0,1))
    training_set_scaled = sc.fit_transform(training_set)
```



4. 準備 training 資料(取預測點前 60 天)

Timesteps 設為 60 ，代表過去 60 天的資訊，嘗試過數值設置太少，將使 RNN 無法學習。

```
[ ] X_train = [] #預測點的前60天資料
    y_train = [] #預測點
    for i in range(60,1509): #1509是訓練集總數
        X_train.append(training_set_scaled[i-60:i,0])
        y_train.append(training_set_scaled[i,0])
    X_train, y_train = np.array(X_train), np.array(y_train) #轉成numpy array的格式, 以利輸入RNN
```

```
[ ] X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

5. Reshape

因為現在 X_train 是 2-dimension，將它 reshape 成 3-dimension: [stock prices, timesteps, indicators]

```
[ ] X_train = [] #預測點的前60天資料
    y_train = [] #預測點
    for i in range(60,1509): #1509是訓練集總數
        X_train.append(training_set_scaled[i-60:i,0])
        y_train.append(training_set_scaled[i,0])
    X_train, y_train = np.array(X_train), np.array(y_train) #轉成numpy array的格式，以利輸入RNN
```

```
[ ] X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

C. 單點分析，用 6 年預測 1 天

1. 匯入套件及資料

```
from __future__ import absolute_import, division, print_function, unicode_literals
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

```
[7] df = pd.read_csv('AAPL_2012-2017.csv')
```

```
[8] df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2012-01-03	58.485714	58.928570	58.428570	58.747143	51.115936	75555200
1	2012-01-04	58.571430	59.240002	58.468571	59.062859	51.390648	65005500
2	2012-01-05	59.278572	59.792858	58.952858	59.718571	51.961189	67817400
3	2012-01-06	59.967144	60.392857	59.888573	60.342857	52.504375	79573200
4	2012-01-09	60.785713	61.107143	60.192856	60.247143	52.421093	98506100

```
[9] df.shape
```

```
(1509, 7)
```

2. 選取資料(單變量/多變量)

➢ 單變量

參數 history_size 是過去信息窗口的大小；target_size 模型在未來需要學會預測的時間，是需要被預測的標籤。

```
[10] def univariate_data(dataset, start_index, end_index, history_size, target_size):
       data = []
       labels = []

       start_index = start_index + history_size
       if end_index is None:
         end_index = len(dataset) - target_size

       for i in range(start_index, end_index):
         indices = range(i-history_size, i)
         # Reshape data from (history_size,) to (history_size, 1)
         data.append(np.reshape(dataset[indices], (history_size, 1)))
         labels.append(dataset[i+target_size])
       return np.array(data), np.array(labels)
```

```
[11] TRAIN_SPLIT = 755
     print(TRAIN_SPLIT)
```

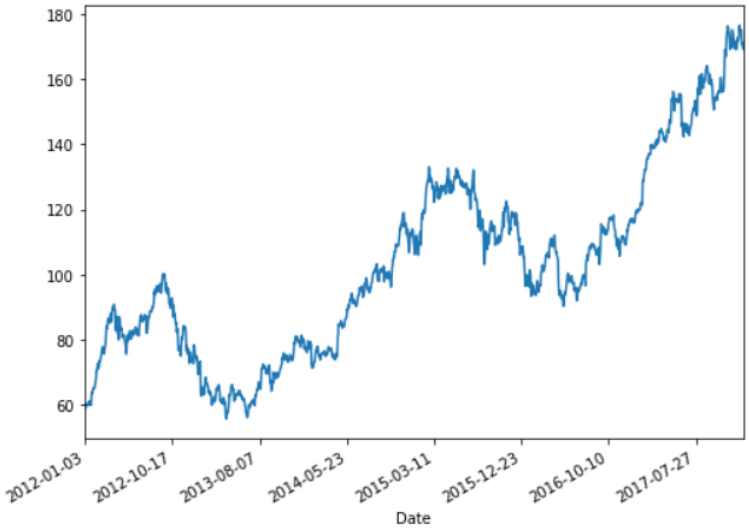
```
uni_data = df['Close']
uni_data.index = df['Date']
uni_data.head()
```

```
Date
2012-01-03    58.747143
2012-01-04    59.062859
2012-01-05    59.718571
2012-01-06    60.342857
2012-01-09    60.247143
Name: Close, dtype: float64
```

“close” 的資料圖形

```
[14] uni_data.plot(subplots=True)
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fa73285a860>],
      dtype=object)
```



- 多變量

```
[30] features_considered = ['Open', 'High', 'Low' ,'Close', 'Adj Close', 'Volume']
```

```
[31] features = df[features_considered]
     features.index = df['Date']
     features.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2012-01-03	58.485714	58.928570	58.428570	58.747143	51.115936	75555200
2012-01-04	58.571430	59.240002	58.468571	59.062859	51.390648	65005500
2012-01-05	59.278572	59.792858	58.952858	59.718571	51.961189	67817400
2012-01-06	59.967144	60.392857	59.888573	60.342857	52.504375	79573200
2012-01-09	60.785713	61.107143	60.192856	60.247143	52.421093	98506100

```
[34] dataset = features.values
     data_mean = dataset[:TRAIN_SPLIT].mean(axis=0)
     data_std = dataset[:TRAIN_SPLIT].std(axis=0)
```

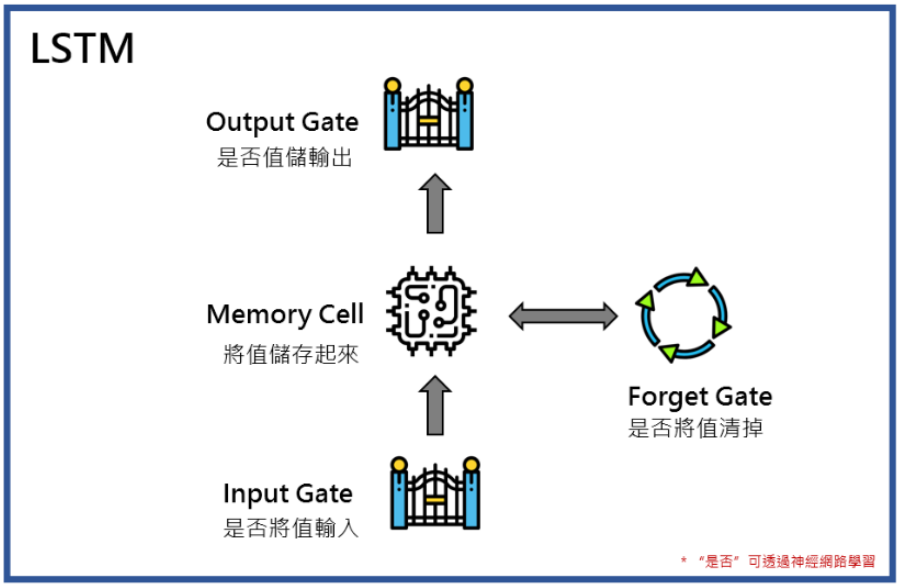
```
[35] dataset = (dataset-data_mean)/data_std
```

```
[36] def multivariate_data(dataset, target, start_index, end_index, history_size,
                           target_size, step, single_step=False):
       data = []
       labels = []

       start_index = start_index + history_size
       if end_index is None:
         end_index = len(dataset) - target_size

       for i in range(start_index, end_index):
         indices = range(i-history_size, i, step)
         data.append(dataset[indices])

         if single_step:
           labels.append(target[i+target_size])
         else:
           labels.append(target[i:i+target_size])

       return np.array(data), np.array(labels)
```

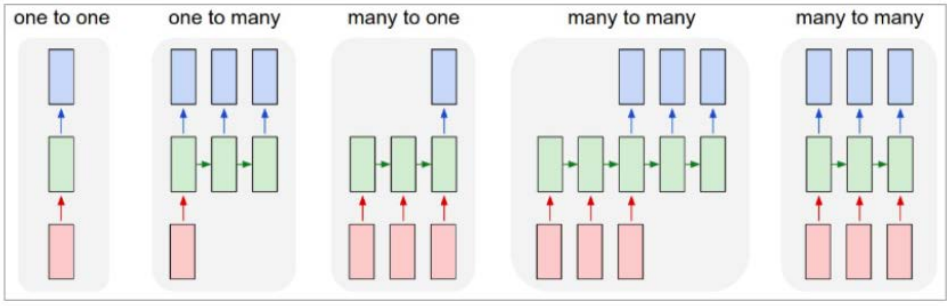
三、模型架構

（一） LSTM 介紹

長短期記憶(Long Short-Term Memory, LSTM)是 RNN 的一種，而其不相同之處在於有了更多的控制單元 input gate、output gate、forget gate 示意圖如下。



而 LSTM model 可分為 One to One、One to many、Many to one 或者 Many to many。如下圖解釋:



本組使用 Many to Many Model

Input:多天股價 (前 10 天)

Output:多天股價(後 3 天)

(二) 模型共分為以下 3 種

A. 多對多，長時間預測，以 6 年預測 6 年(2012-2017)

為了構建 LSTM，我們需要從 Keras 中導入幾個模塊:

Sequential 用於初始化神經網絡

Dense 用於添加密集連接的神經網絡層

LSTM 用於添加長短期內存層

Dropout 用於添加防止過擬合的 dropout 層

```
[ ]  from sklearn.model_selection import train_test_split #分測試跟訓練資料
     from keras.models import Sequential
     from keras.layers import Dense, LSTM, Dropout, Flatten, TimeDistributed
     from keras.optimizers import adam
     from keras.callbacks import ModelCheckpoint
     from keras.losses import mean_absolute_percentage_error
```

資料切割，作 train 跟 test 的切割而 test_size=0.2 ，training data 占 0.8

input 跟 output 都分 train 跟 test

```
[ ]  #作train跟test的切割
     #xraw
     #test_size0.2 trainingdata占0.8 116*0.8是training data
     #input跟output都分train跟test
     X_train, X_test, Y_train, Y_test = train_test_split(xraw, yraw, test_size = 0.2, random_state = 10)
```

LSTM model 一層 100 個單元，也就是輸出空間的維度，設定 input shape training(116,5,10)

return_sequence =True，它決定是否返回輸出序列中的最後一個輸出，還是返回完整的序列

在定義 Dropout 層時，我們指定 0.3，這意味著 30%的層將被刪除。然後，我們添加指定 1 個單元的輸出的 Dense 層

縮放指數線性單元（scaled exponential linear units，selu），經過該激活函數後使得樣本分佈自動歸一化到 0 均值和單位方差

這樣在 activation 的方差過大的時候可以讓它減小，防止了梯度爆炸，但是正半軸坡度簡單的設成了 1 。而 selu 的正半軸大於 1，在方差過小的的時候可以讓它增大，同時防止了梯度消失。這樣激活函數就有一個不動點，網絡深了以後每一層的輸出都是均值為 0 方差為 1。

$$\mathrm{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

```
#Sequential目的變成長條線形狀的model變成有順序性的
#用nn這個東西是有連接性的
model = Sequential()


#LSTM model一層100個單元, input shape trainging data 116,5,10  return_sequence要是true必須要五個五個一起回傳(可能5會不見) 會回傳兩個值
model.add(LSTM(100, input_shape = (5,10),return_sequences=True, recurrent_dropout=0.3))
#model.add(LSTM(100, return_sequences=True))
model.add(LSTM(80, return_sequences=True))
model.add(LSTM(50, return_sequences=True))
#把會被丟棄的cell留0.3下來
model.add(Dropout(0.3))

#timedistributed配合上面的return sequence讓他控制在5
#selu變準的
#可能通過全聯階層之後可能數值會比較好

#model.add(TimeDistributed(Dense(100, activation='selu')))
model.add(TimeDistributed(Dense(50, activation='selu')))
model.add(TimeDistributed(Dense(3, activation='selu')))
```

下圖為可用變數的使用量

```
[ ] model.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 5, 100)            44400
_________________________________________________________________
lstm_8 (LSTM)                (None, 5, 80)             57920
_________________________________________________________________
lstm_9 (LSTM)                (None, 5, 50)             26200
_________________________________________________________________
dropout_3 (Dropout)          (None, 5, 50)             0
_________________________________________________________________
time_distributed_5 (TimeDist (None, 5, 50)             2550
_________________________________________________________________
time_distributed_6 (TimeDist (None, 5, 3)              153
=================================================================
Total params: 131,223
Trainable params: 131,223
Non-trainable params: 0
_________________________________________________________________
```

B. 多對多，以 6 年預測 1 個月

Import Keras

```
[ ] #import the Keras libraries and packages
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from keras.layers import Dropout
    from keras.optimizers import adam
    from keras.callbacks import ModelCheckpoint
    from keras.losses import mean_absolute_percentage_error

    #initialising the RNN
    regressor = Sequential()
```

搭建 LSTM layer

units: 神經元的數目，使用 dropout 設為 0.3，由於第四層 LSTM Layer 即將跟 Ouput Layer 做連接，因此 return_sequences 設為預設值 False （也就是不用寫上 return_sequences）

```
# Adding the first LSTM layer and some Droupout regrlarisation
regressor.add(LSTM(units = 100, return_sequences = True, input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.3))

#Adding a second LSTM layer and some Droupout regularisation
regressor.add(LSTM(units = 100, return_sequences = True))
regressor.add(Dropout(0.3))

#Adding a third LSTM layer and some Droupout regularisation
regressor.add(LSTM(units = 100, return_sequences = True))
regressor.add(Dropout(0.3))

#Adding a fourth LSTM layer and some Droupout regularisation
regressor.add(LSTM(units = 100))
regressor.add(Dropout(0.3))
```

```
[ ] regressor.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 60, 100)           40800
_________________________________________________________________
dropout_5 (Dropout)          (None, 60, 100)           0
_________________________________________________________________
lstm_6 (LSTM)                (None, 60, 100)           80400
_________________________________________________________________
dropout_6 (Dropout)          (None, 60, 100)           0
_________________________________________________________________
lstm_7 (LSTM)                (None, 60, 100)           80400
_________________________________________________________________
dropout_7 (Dropout)          (None, 60, 100)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_8 (Dropout)          (None, 100)               0
=================================================================
Total params: 282,000
Trainable params: 282,000
Non-trainable params: 0
_________________________________________________________________
```

C. 單點分析，用 6 年預測 1 天(單變量/多變量)

➢ 單變量

```
[15] univariate_past_history = 60
     univariate_future_target = 0

     x_train_uni, y_train_uni = univariate_data(uni_data, 0, TRAIN_SPLIT,
                                                univariate_past_history,
                                                univariate_future_target)
     x_val_uni, y_val_uni = univariate_data(uni_data, TRAIN_SPLIT, None,
                                            univariate_past_history,
                                            univariate_future_target)
```

```
[16] print ('Single window of past history')
     print (x_train_uni[0])
     print ('\n Target stock price to predict')
     print (y_train_uni[0])
```

```
Single window of past history
[[-1.07611383]
 [-1.07898735]
 [-1.08958964]
 [-1.04440545]
 [-1.00189681]
 [-0.95373957]
 [-0.87595528]
 [-0.71255888]
 [-0.71008176]
 [-0.61911853]
 [-0.55114428]
 [-0.66796946]
 [-0.62298316]
 [-0.62387487]
 [-0.49773545]
 [-0.51567067]
 [-0.48247628]
 [-0.42282517]
 [-0.38963022]
 [-0.29401018]
 [-0.224351  ]
 [-0.20423657]
 [-0.197201  ]
 [-0.31630485]
 [-0.34504081]
 [-0.34077978]
 [-0.22881047]
 [-0.19729991]
 [-0.12962309]
 [ 0.02990917]
 [ 0.24275079]
 [ 0.20291717]
 [ 0.20301615]
 [ 0.35690034]
 [ 0.40505703]
 [ 0.37077249]
 [ 0.33946099]
 [ 0.3068609 ]
 [ 0.41516401]
 [ 0.48948056]
 [ 0.52059417]]

 Target stock price to predict
0.4437015233244694
```

➢ 多變量

```
[38] past_history = 60
     future_target = 30
     STEP = 1

     x_train_single, y_train_single = multivariate_data(dataset, dataset[:, 1], 0,
                                                        TRAIN_SPLIT, past_history,
                                                        future_target, STEP,
                                                        single_step=True)
     x_val_single, y_val_single = multivariate_data(dataset, dataset[:, 1],
                                                    TRAIN_SPLIT, None, past_history,
                                                    future_target, STEP,
                                                    single_step=True)
```

```
[39] x_val_single.shape

     (664, 60, 6)
```

四、訓練過程

✧ 以下所有訓練模型，為了讓學習率變佳，在優化器的部分(optimizer)我們將 adagrad 改成使用 adam，它有做參數的「**偏離校正**」，使得每一次的學習率都會有個確定的範圍，會讓參數的更新較為平穩。

A. 多對多，長時間預測，以 6 年預測 6 年(2012-2017)

loss 以 MAE 值為基礎，Checkpoint 目的是訓練模型的過程中會有時候有高有低，目的是存在最佳模型

```
#loss 選MAE 跟 MSE（直很小） MAE值比較大比較好看

model.compile(loss = mean_absolute_percentage_error,
              optimizer = adam(lr = 0.001),
             )
```

```
[ ] #checkpoint目的是訓練模型的過程中會有時候有高有低, 目的是存最佳模型
    #用validation loss , 一直存最低的val_loss的model

    checkpoint = ModelCheckpoint('to3.h5', monitor='val_loss', verbose=1, save_best_only=True,mode='min')
    callbacks_list = [checkpoint]

    k = model.fit(X_train ,Y_train, batch_size = 100, epochs = 500,
                  callbacks=[checkpoint],
            verbose = 1,
            validation_data = (X_test, Y_test))

    #MAE百分比轉換
```

```
[ ] Epoch 495/500
92/92 [==============================] - 0s 445us/step - loss: 15.0185 - val_loss: 20.5819

Epoch 00495: val_loss did not improve from 20.44295
Epoch 496/500
92/92 [==============================] - 0s 432us/step - loss: 15.1188 - val_loss: 21.0937

Epoch 00496: val_loss did not improve from 20.44295
Epoch 497/500
92/92 [==============================] - 0s 435us/step - loss: 15.0165 - val_loss: 21.4229

Epoch 00497: val_loss did not improve from 20.44295
Epoch 498/500
92/92 [==============================] - 0s 423us/step - loss: 15.0327 - val_loss: 20.7326

Epoch 00498: val_loss did not improve from 20.44295
Epoch 499/500
92/92 [==============================] - 0s 488us/step - loss: 14.9136 - val_loss: 21.0083

Epoch 00499: val_loss did not improve from 20.44295
Epoch 500/500
92/92 [==============================] - 0s 433us/step - loss: 14.9652 - val_loss: 21.2540

Epoch 00500: val_loss did not improve from 20.44295
```

B. 多對多，以 6 年預測 1 個月/一年。

➢ 一個月

```
[ ] #Adding the output layer
    #output layer units 設為1
    regressor.add(Dense(units = 1))
```

```
[ ] #compiling
    regressor.compile(optimizer='adam', loss = 'mae')

    #train
    history=regressor.fit(X_train, y_train, epochs = 50, batch_size = 32)
```

```
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0195
Epoch 40/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0196
Epoch 41/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0214
Epoch 42/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0211
Epoch 43/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0199
Epoch 44/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0203
Epoch 45/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0203
Epoch 46/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0201
Epoch 47/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0214
Epoch 48/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0208
Epoch 49/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0214
Epoch 50/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0191
```

➢ 一年

```
[ ] #Adding the output layer
    #output layer units 設為1
    regressor.add(Dense(units = 1))
```

```
[ ] #compiling
    regressor.compile(optimizer='adam', loss = 'mae')

    #train
    history=regressor.fit(X_train, y_train, epochs = 50, batch_size = 32)
```

```
Epoch 42/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0260
Epoch 43/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0273
Epoch 44/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0263
Epoch 45/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0280
Epoch 46/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0263
Epoch 47/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0257
Epoch 48/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0263
Epoch 49/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0253
Epoch 50/50
1449/1449 [==============================] - 13s 9ms/step - loss: 0.0253
```

C. 單點分析，用 6 年預測 1 天(單變量/多變量)

➢ 單變量

```
[26] simple_lstm_model = tf.keras.models.Sequential([
        tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),
        tf.keras.layers.Dense(1)
     ])

     tf.keras.optimizers.Adam(lr=0.01)
     simple_lstm_model.compile(optimizer ='adam', loss='mae')
```

```
[27] for x, y in val_univariate.take(1):
        print(simple_lstm_model.predict(x).shape)
```

```
(256, 1)
```

```
EVALUATION_INTERVAL = 200
EPOCHS = 50

simple_lstm_model.fit(train_univariate, epochs=EPOCHS,
                      steps_per_epoch=EVALUATION_INTERVAL,
                      validation_data=val_univariate, validation_steps=50)
```

```
Epoch 45/50
200/200 [==============================] - 2s 8ms/step - loss: 0.0646 - val_loss: 0.5584
Epoch 46/50
200/200 [==============================] - 2s 9ms/step - loss: 0.0646 - val_loss: 0.5594
Epoch 47/50
200/200 [==============================] - 2s 8ms/step - loss: 0.0646 - val_loss: 0.5597
Epoch 48/50
200/200 [==============================] - 2s 8ms/step - loss: 0.0645 - val_loss: 0.5582
Epoch 49/50
200/200 [==============================] - 2s 8ms/step - loss: 0.0646 - val_loss: 0.5586
Epoch 50/50
200/200 [==============================] - 2s 8ms/step - loss: 0.0645 - val_loss: 0.5596
<tensorflow.python.keras.callbacks.History at 0x7fa6d90904e0>
```

- 多變量

```
[40] print ('Single window of past history : {}'.format(x_train_single[0].shape))
```

```
Single window of past history : (60, 6)
```

```
[41] train_data_single = tf.data.Dataset.from_tensor_slices((x_train_single, y_train_single))
     train_data_single = train_data_single.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()

     val_data_single = tf.data.Dataset.from_tensor_slices((x_val_single, y_val_single))
     val_data_single = val_data_single.batch(BATCH_SIZE).repeat()
```

```
[42] single_step_model = tf.keras.models.Sequential()
     single_step_model.add(tf.keras.layers.LSTM(16,
                                                input_shape=x_train_single.shape[-2:],recurrent_dropout=0.3))

     single_step_model.add(tf.keras.layers.Dropout(0.3))
     single_step_model.add(tf.keras.layers.Dense(1))

     single_step_model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss='mae')
```

```
[43] for x, y in val_data_single.take(1):
       print(single_step_model.predict(x).shape)
```

```
(256, 1)
```

```
single_step_history = single_step_model.fit(train_data_single, epochs=25,
                                            steps_per_epoch=200,
                                            validation_data=val_data_single,
                                            validation_steps=50)
```

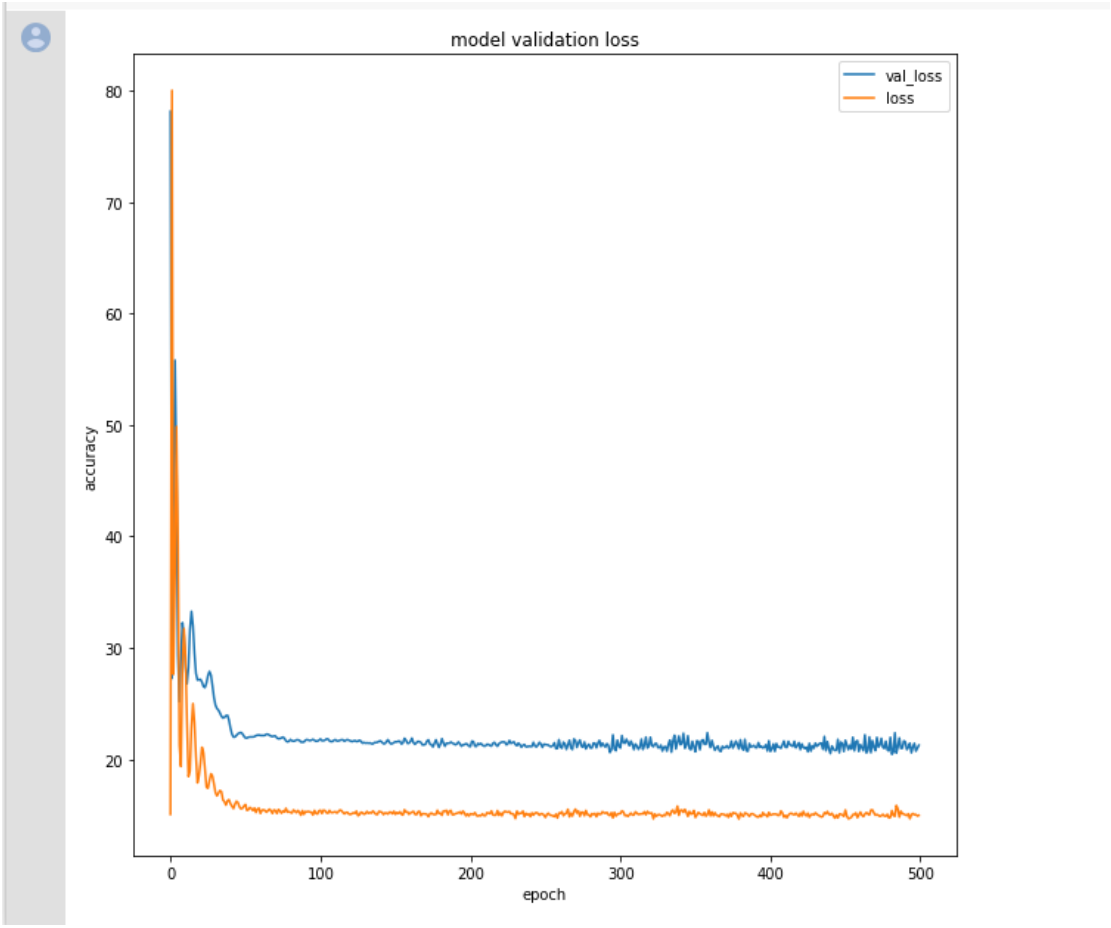
```
200/200 [==============================] - 21s 107ms/step - loss: 0.2284 - val_loss: 1.6190
Epoch 21/25
200/200 [==============================] - 21s 107ms/step - loss: 0.2269 - val_loss: 1.5951
Epoch 22/25
200/200 [==============================] - 22s 110ms/step - loss: 0.2259 - val_loss: 1.5507
Epoch 23/25
200/200 [==============================] - 22s 108ms/step - loss: 0.2254 - val_loss: 1.9494
Epoch 24/25
200/200 [==============================] - 22s 108ms/step - loss: 0.2236 - val_loss: 1.5839
Epoch 25/25
200/200 [==============================] - 22s 108ms/step - loss: 0.2207 - val_loss: 1.8346
```

五、訓練結果

A. 多對多，長時間預測，以 6 年預測 6 年(2012-2017)

印出損失函數圖形

```
[ ]  plt.figure(figsize=(10, 10))
     plt.plot(k.history["val_loss"])
     plt.plot(k.history["loss"])

     plt.title('model validation loss')
     plt.ylabel("accuracy")
     plt.xlabel('epoch')
     plt.legend(['val_loss' , 'loss'], loc = 'best')
     plt.show()
```



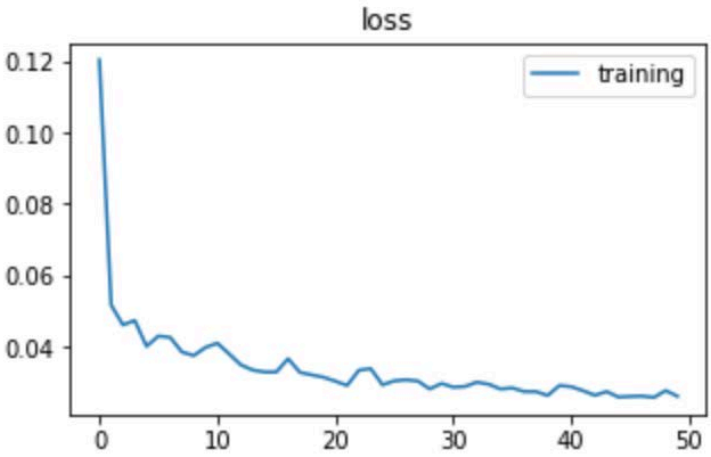
可以看出來有持續下降, 後來有持平無法繼續在收斂，可能是資料極限。

B. 多對多，以 6 年預測 1 個月/一年

➢ 1 個月

印出損失函數

```
[ ]  plt.figure(figsize=(5,3))
     plt.plot(history.epoch,history.history['loss'], label='training')
     plt.title('loss')
     plt.legend(loc='best')
```

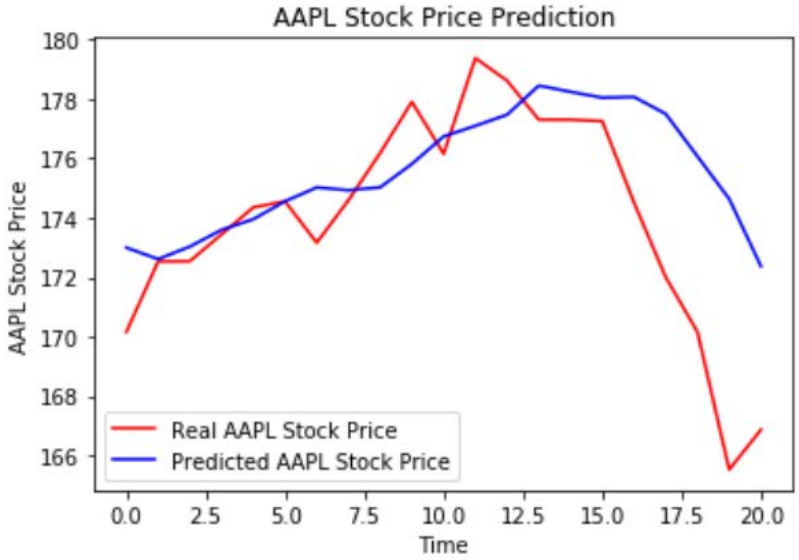


取”open”的資料去做預測分析，印出趨勢走勢圖

```
[ ]  dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']), axis = 0)
     inputs = dataset_total[len(dataset_total)-len(dataset_test)-60:].values
     inputs = inputs.reshape(-1,1)
     inputs = sc.transform(inputs) #feature scaling

     X_test = []
     for i in range (60, 81): #timesteps一樣60, 81等於先前的60天+2018的未來21天資料
       X_test.append(inputs[i-60:i,0])
     X_test = np.array(X_test)
     X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],1)) #reshape 成3-dimension
```

```
[ ]  predicted_stock_price = regressor.predict(X_test)
     predicted_stock_price = sc.inverse_transform(predicted_stock_price) #to get the original scale
```

```
[ ]  #visualising the results
     plt.plot(real_stock_price, color = 'red', label='Real AAPL Stock Price') #red line show the real stock price
     plt.plot(predicted_stock_price, color = 'blue', label='Predicted AAPL Stock Price') #blue line show the predict stock price

     plt.title('AAPL Stock Price Prediction')
     plt.xlabel('Time')
     plt.ylabel('AAPL Stock Price')
     plt.legend()
     plt.show()
```



可以看出趨勢大致符合實際值，但幅度較為平緩。

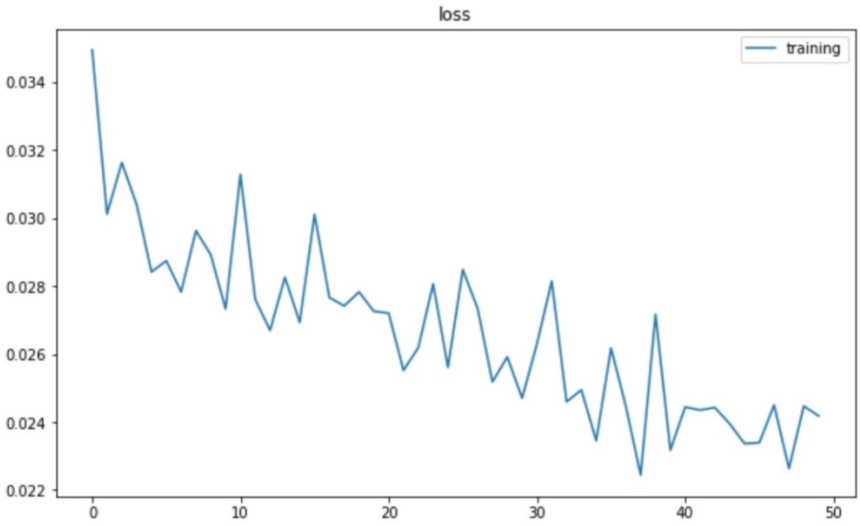
```
a = ((predicted_stock_price)-(real_stock_price))/(real_stock_price)

ACC = np.mean(a, axis=0)

print (abs(ACC))
```

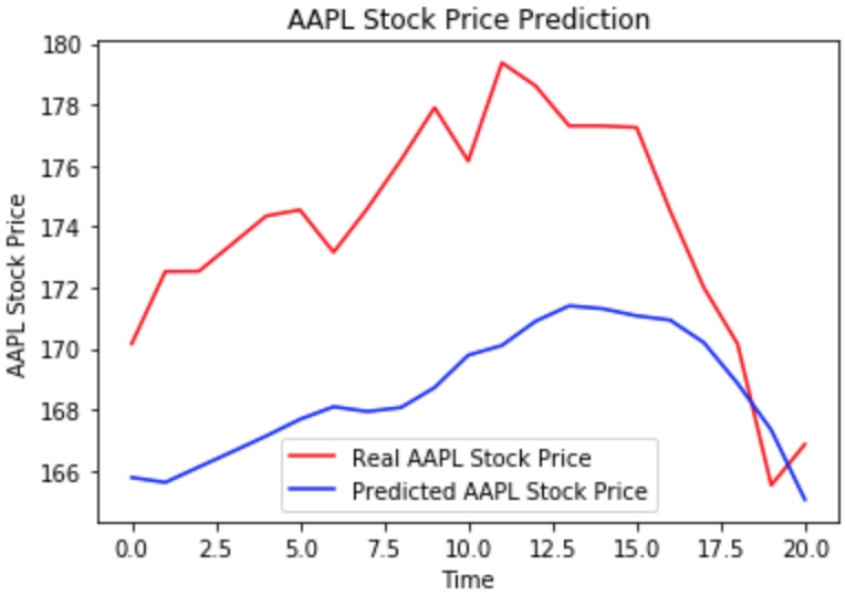
```
[0.00476125]
```

上圖為誤差百分比之計算，我們將數據壓縮至 0-1 之間，因此誤差越小越好。

數值為 0.476125%。

另外，我們亦有針對**金價**是否影響股價去分析一個月的股價影響，如下圖之損失函數及預測分析出趨勢走勢圖





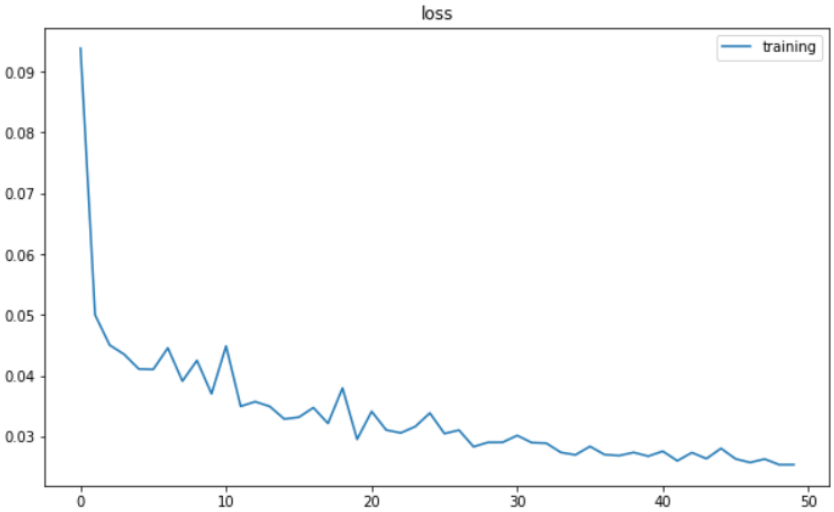
對比有含金價之影響與不包含金價之影響會發現，其實金價對於 APPLE 股票的影響不大，且預測趨勢較不準確。

- 1 年

印出損失函數

```
[ ]  plt.figure(figsize=(10,6))
     plt.plot(history.epoch,history.history['loss'], label='training')
     plt.title('loss')
     plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7fdba935af60>
```



取”open”的資料去做預測分析，印出趨勢走勢圖

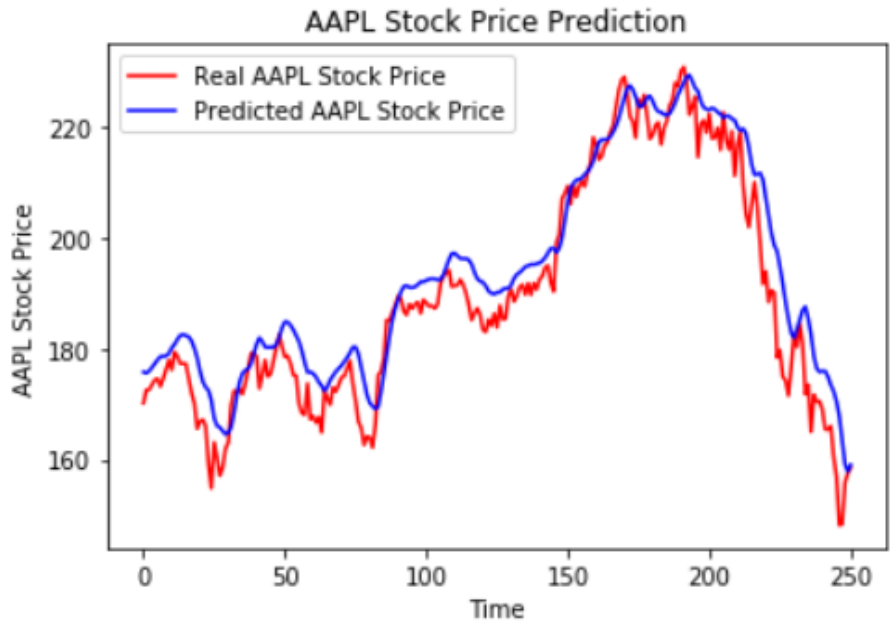
```
[ ] #get 201801 test data
    dataset_test = pd.read_csv('AAPL_2018-2019.csv')
    real_stock_price = dataset_test.iloc[:, 1:2].values
    dataset_test.shape
```

```
(251, 7)
```

```
[ ] dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']), axis = 0)
    inputs = dataset_total[len(dataset_total)-len(dataset_test)-60:].values
    inputs = inputs.reshape(-1,1)
    inputs = sc.transform(inputs) #feature scaling

    X_test = []
    for i in range (60, 311): #timesteps一樣60, 331等於先前的60天+2018的251天資料
      X_test.append(inputs[i-60:i,0])
    X_test = np.array(X_test)
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],1)) #reshape 成3-dimension
```

```
[ ] predicted_stock_price = regressor.predict(X_test)
    predicted_stock_price = sc.inverse_transform(predicted_stock_price) #to get the original scale
```

```
[ ] #visualising the results
    plt.plot(real_stock_price, color = 'red', label='Real AAPL Stock Price') #red line show the real stock price
    plt.plot(predicted_stock_price, color = 'blue', label='Predicted AAPL Stock Price') #blue line show the predict stock price

    plt.title('AAPL Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('AAPL Stock Price')
    plt.legend()
    plt.show()
```



由前所述的一年與一個月的趨勢圖可以清楚知道一年的預測趨勢圖較準確，預測誤差也必較小，因此得知預測時間越長，準確率越高。

```
a = ((predicted_stock_price)-(real_stock_price))/(real_stock_price)

ACC = np.mean(a, axis=0)

print (abs(ACC))
```

```
[0.00040419]
```

上圖為誤差百分比之計算，我們將數據壓縮至 0-1 之間，因此誤差越小越好。

數值為 0.040419%。

C. 單點分析，用 6 年預測 1 天(單變量/多變量)

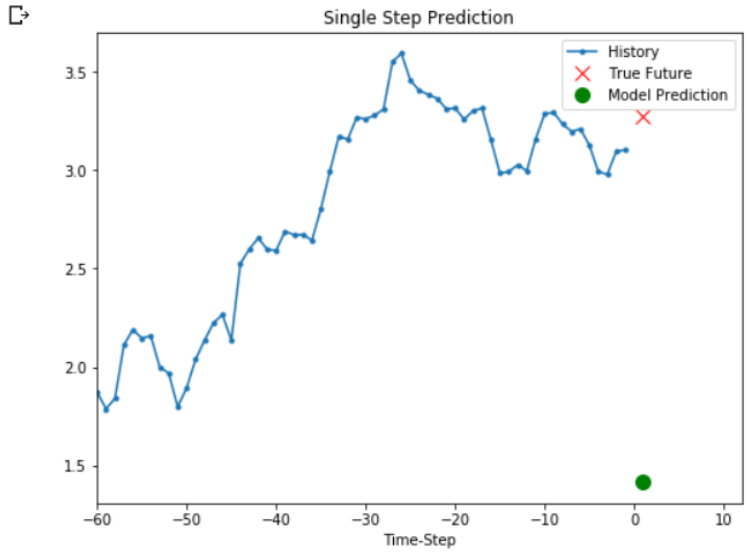
➢ 單變量

```
for x, y in val_univariate.take(1):
  plot = show_plot([x[0].numpy(), y[0].numpy(),
                    simple_lstm_model.predict(x)[0]], 0, 'Simple LSTM model')
  plot.show()
```

可以從圖中看出單一特徵預測單一點誤差極小。

➢ 多變量

```
[44] for x, y in val_data_single.take(1):
       plot = show_plot([x[0][:, 1].numpy(), y[0].numpy(),
                         single_step_model.predict(x)[0]], 1,
                        'Single Step Prediction')
       plot.show()
```



可以從圖中看出多特徵預測單一點的結果還需要調整。

六、結果與未來展望

深度學習肯定是可以用在股票市場的，比如針對某隻股票的新聞情感分析等。本次模型的任務是股票預測，由於股票預測是無法只利用一個模型就能完整的預測，於是我們用了三種模型來建構此預測過程，第一種模型可以用來判斷出股價是可以被用來預測架構合理，且趨勢極為準確，可用於長期趨勢預測；第二種模型屬於預測未來的股價走勢，用於判斷股市上漲或下跌；最後一種用於單一天的股價預測，可以用來產出當天的預測值。此三種模型是相輔相成的，由第一種觀察長期趨勢來決定投資方向，利用第二種模型判斷出漲跌走勢，再由第三個模型的數值來判斷，此走勢和數值對於投資者的投資策略該如何決策。利用視覺化結果可以看出模型的預測表現，如預測的趨勢雖大致上跟真實股價是一致，但預測的較為平滑，因為股市的上下坡動及影響因

素很多，如同前面將金價及油價加入模型做為參考值之一，可以發現金價對於此股的影響較小，甚至有誤判得情形，負面影響較大，而油價的模型對與此股相較起來有正向影響，因此期望在未來於模型中導入多變量分析長期模型，以得到更精確的預測結果。