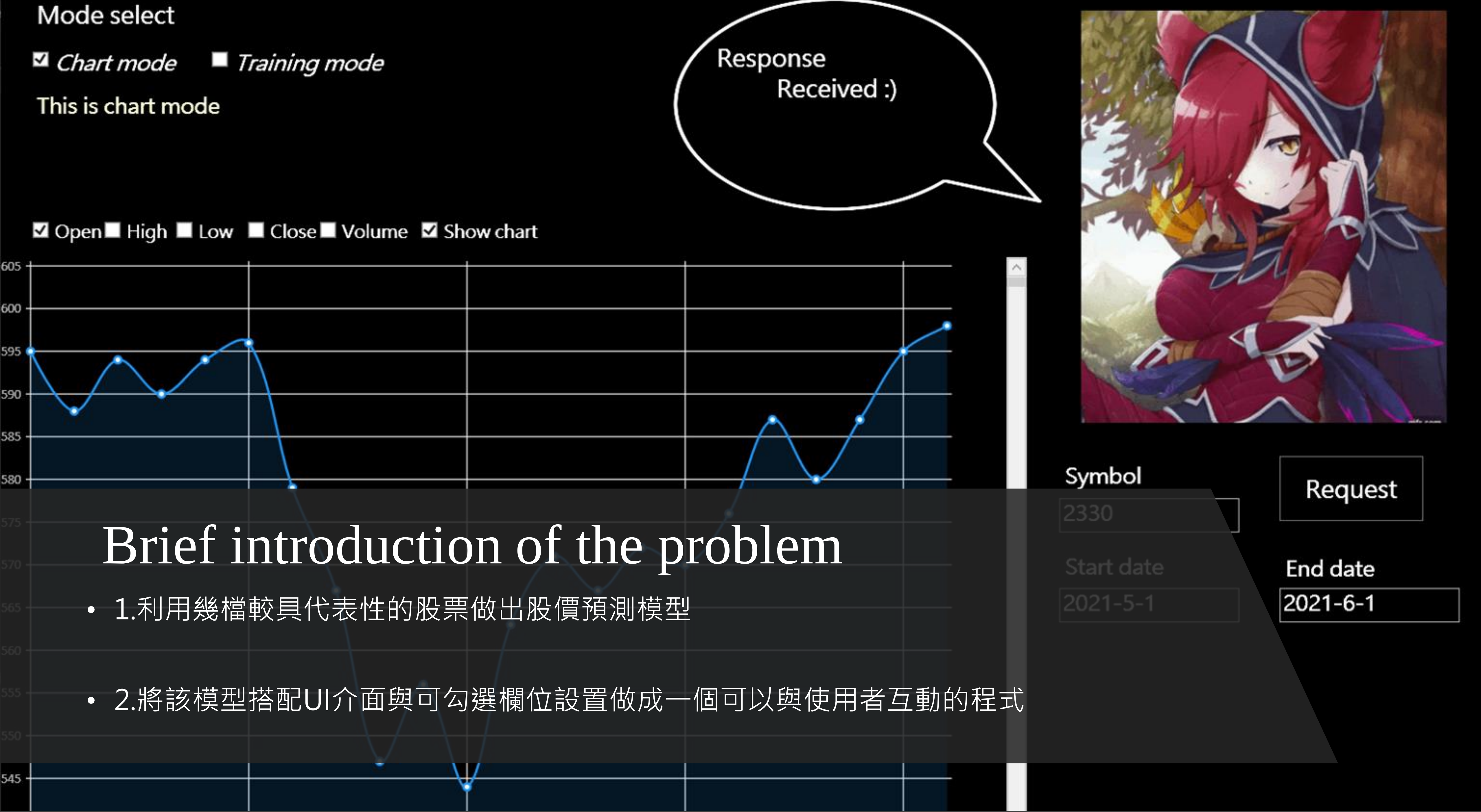
# 見微知著—讓python成為你的股票理專

組名:教父
組員:
統計系112林家同
資訊系112莊上緣
資訊系112李培綸




# Brief introduction of the problem

- 1.利用幾檔較具代表性的股票做出股價預測模型
- 2.將該模型搭配UI介面與可勾選欄位設置做成一個可以與使用者互動的程式


# Data description and preprocessing (1)

- 資料來源網站:FinMind

In [273]:
```python
url = "https://api.finmindtrade.com/api/v4/login"
account = {
    "user_id": "user_id",
    "password": "password",
}
token = requests.post(url, data=account).json()['token']

url = "https://api.finmindtrade.com/api/v4/data"
parameter = {
    "dataset": "TaiwanStockPrice",
    "data_id": "2330",
    "start_date": "2016-06-04",
    "end_date": "2021-06-03",
    "token":token
}
data = pd.DataFrame(requests.get(url, params=parameter).json()["data"])
data['average'] = data["Trading_money"]/data["Trading_Volume"]
data
```

Out[273]:

| | date | stock_id | Trading_Volume | Trading_money | open | max | min | close | spread | Trading_turnover | average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2016-06-04 | 2330 | 2631269 | 420392883 | 160.0 | 160.0 | 159.5 | 160.0 | -0.5 | 1428 | 159.768113 |
| **1** | 2016-06-06 | 2330 | 28898951 | 4651663660 | 161.0 | 162.0 | 160.0 | 161.0 | 1.0 | 8667 | 160.963063 |
| **2** | 2016-06-07 | 2330 | 47573770 | 7726094806 | 161.5 | 163.5 | 161.0 | 162.0 | 1.0 | 12269 | 162.402408 |
| **3** | 2016-06-08 | 2330 | 45587946 | 7522713617 | 164.0 | 166.0 | 163.5 | 165.5 | 3.5 | 15173 | 165.015410 |

## Data description and preprocessing (2)

- Web scraping+將資料表格化

# Data description and preprocessing (3)

- 多次資料拓展(更多特徵值)

In [277]:
```
#將多張dataframe合併
data=data.join(data2)
data=data.join(data3)
data=data.join(data4)
data=data.join(data5)
data
```

Out[277]:

| | 日期 | 股票編號 | 當日成交量 | 當日成交金額 | 開盤價 | 最高價 | 最低價 | 收盤價 | 買賣價差 | 周轉率 | ... | 昨日融卷 | 融資變化 | 融券變化 | 殖利率 | 本益比 | 價淨值比 | 大盤融資餘額(百萬) | 大盤融卷量 | 大盤指數 | 大盤 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2016-06-04 | 2330 | 2631269 | 420392883 | 160.0 | 160.0 | 159.5 | 160.0 | -0.5 | 1428 | ... | 3812 | -26 | -73 | 3.75 | 14.20 | 3.24 | 12994.9943 | 360288 | 8591.57 | 4 |
| **1** | 2016-06-06 | 2330 | 28898951 | 4651663660 | 161.0 | 162.0 | 160.0 | 161.0 | 1.0 | 8667 | ... | 3739 | -260 | 35 | 3.73 | 14.29 | 3.26 | 12967.5393 | 368731 | 8597.11 | 5 |
| **2** | 2016-06-07 | 2330 | 47573770 | 7726094806 | 161.5 | 163.5 | 161.0 | 162.0 | 1.0 | 12269 | ... | 3774 | -173 | 487 | 3.70 | 14.37 | 3.28 | 12935.2080 | 379136 | 8679.90 | 82 |
| **3** | 2016-06-08 | 2330 | 45587946 | 7522713617 | 164.0 | 166.0 | 163.5 | 165.5 | 3.5 | 15173 | ... | 4261 | -436 | 62 | 3.63 | 14.69 | 3.35 | 12945.7301 | 378743 | 8715.48 | 35 |
| **4** | 2016-06-13 | 2330 | 54689228 | 8843806836 | 161.0 | 162.5 | 161.0 | 162.0 | -3.5 | 14466 | ... | 4323 | 44 | -1046 | 3.70 | 14.37 | 3.28 | 12925.4617 | 355334 | 8536.22 | -179 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1217** | 2021-05-28 | 2330 | 30720737 | 18082265480 | 587.0 | 592.0 | 582.0 | 590.0 | 8.0 | 31581 | ... | 425 | 118 | 2 | 1.69 | 28.30 | 7.89 | 23336.5365 | 556327 | 16870.86 | 269 |
| **1218** | 2021-05-31 | 2330 | 31557426 | 18751716595 | 595.0 | 597.0 | 590.0 | 597.0 | 7.0 | 29798 | ... | 427 | -967 | -43 | 1.68 | 28.63 | 7.98 | 23404.7805 | 551036 | 17068.43 | 197 |
| **1219** | 2021-06-01 | 2330 | 18405285 | 10985893229 | 598.0 | 599.0 | 595.0 | 598.0 | 1.0 | 20318 | ... | 384 | -377 | -7 | 1.67 | 28.68 | 8.00 | 24012.6099 | 563095 | 17162.38 | 93 |
| **1220** | 2021-06-02 | 2330 | 22416789 | 13362065937 | 600.0 | 600.0 | 593.0 | 595.0 | -3.0 | 25170 | ... | 377 | 34 | -12 | 1.68 | 28.54 | 7.96 | 24049.8196 | 550856 | 17165.04 | 2 |
| **1221** | 2021-06-03 | 2330 | 31703679 | 18939839664 | 600.0 | 600.0 | 596.0 | 596.0 | 1.0 | 20749 | ... | 365 | 401 | -40 | 1.68 | 28.59 | 7.97 | 24595.8470 | 559529 | 17246.16 | 81 |

1222 rows × 24 columns

Insights discovered from the data

```
In [303]: df.columns=["日期","股票編號","當日成交量","當日成交金額","開盤價","最高價","最低價","收盤價","買賣價差","周轉率","平均成交價","當日融資"
          df=df.drop("股票編號",axis=1)
          df.set_index(["日期"], inplace=True)
          df.insert(loc=0,column="price",value=df["收盤價"].tolist())
          df=df.drop("收盤價",axis=1)
```

```
In [304]: for column in df:
              print(column,":",end="")
              print(df[column].rolling(10).corr(df['price']).mean(),"& ",end="")
              print(df[column].rolling(20).corr(df['price']).mean(),"& ",end="")
              print(df[column].rolling(30).corr(df['price']).mean(),"& ",end="")
              print(df[column].rolling(50).corr(df['price']).mean())

price :1.0000000000009488 & 1.0000000000005163 & 0.999999999999705 & 1.0000000000002032
當日成交量 :-0.060926112156154684 & -0.05296856382912049 & -0.05544127975642681 & -0.04084434167185033
當日成交金額 :-0.023481383742197468 & -0.0034604126050957534 & 0.004525258374345535 & 0.04058623762819695
開盤價 :0.7734322229160839 & 0.8761190336873248 & 0.9156180808073966 & 0.9507631112995036
最高價 :0.897879147235532 & 0.9470361082897873 & 0.9641822082122997 & 0.9788268477013771
最低價 :0.9072244607151286 & 0.9512475508678778 & 0.9666060376034861 & 0.9800555556959423
買賣價差 :0.4481587280886025 & 0.3388784759028325 & 0.28144584272453527 & 0.21574714804626507
周轉率 :-0.049475140709937 & -0.03937077724579707 & -0.03761868327797625 & -0.014568058668363274
平均成交價 :0.9516157832970108 & 0.9747048528102245 & 0.9825495316338504 & 0.9895924037662988
當日融資 :-0.6108921637783956 & -0.5553553371507433 & -0.5104394506348009 & -0.4440866989736907
昨日融資 :-0.33014910292490435 & -0.40943110762830154 & -0.419215521125947 & -0.3999626081316952
當日融卷 :0.5106305586403519 & 0.4517465352706382 & 0.399471632716045 & 0.3389441184408549
昨日融卷 :0.28708088791636915 & 0.3434750500779888 & 0.3370167418569067 & 0.3124360801711942
融卷變化 :-0.32810124701578475 & -0.22020093084227513 & -0.1607550886785136 & -0.09489854198443934
融資變化 :0.2727589878694939 & 0.15210300217203415 & 0.09614529663327576 & 0.05731807517399776
殖利率 :-0.9643120513726858 & -0.948918411884437 & -0.9317468736153713 & -0.9032579832880135
本益比 :0.9416340366213949 & 0.9090629559835036 & 0.8883808943355906 & 0.8541619155738183
股價淨值比 :0.928429893071861 & 0.8920178720670082 & 0.8786244095530218 & 0.8700805943871476
大盤融資餘額(百萬) :0.11743061089911969 & 0.21285919211105592 & 0.2748794726793536 & 0.33806427752749146
大盤融卷量 :0.23525911802303023 & 0.24314460241251976 & 0.23718698839769253 & 0.21838111514182487
大盤指數 :0.8251026948714778 & 0.8366805187418708 & 0.8344626848624905 & 0.8137610730217018
大盤漲跌 :0.38396996118059395 & 0.2893974317442247 & 0.24042742628246785 & 0.17805969964280277
```

```
In [305]: df_new=df.drop(['平均成交價',"當日成交量","當日成交金額","周轉率","當日融卷","昨日融卷","當日融資","昨日融資","殖利率"],axis=1)
          df_new
```

Out[305]:

| price | 開盤價 | 最高價 | 最低價 | 買賣價差 | 融卷變化 | 融資變化 | 本益比 | 股價淨值比 | 大盤融資餘額(百萬) | 大盤融卷量 | 大盤指數 | 大盤漲跌 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

- 計算各個股市資料特徵值與股價(收盤價)的相關係數，並drop掉其中低度相關與負相關的特徵值，去蕪存菁後留下可用資料表格。
- (Which attributes have higher correlation with the prediction target?)

# Methodology details (1)

```
In [40]: from sklearn.preprocessing import MinMaxScaler
         values = df_new.values
         scaler = MinMaxScaler(feature_range=(0, 1))
         scaled = scaler.fit_transform(values)
         print(scaled)
         scaled.shape[0]

[[0.00869565 0.0107949  0.00764818 ... 0.17531891 0.01452332 0.47088767]
 [0.01062802 0.01275761 0.01147228 ... 0.18363432 0.01512964 0.47178086]
 [0.01256039 0.01373896 0.01434034 ... 0.19388207 0.02419057 0.52365922]
 ...
 [0.84927536 0.87438665 0.84894837 ... 0.36300702 0.95284463 0.46984675]
 [0.85120773 0.87438665 0.84894837 ... 0.37154895 0.96172279 0.52253771]
 [0.84927536 0.85672228 0.83938815 ...        nan 0.95091512 0.40174338]]

Out[40]: 1223

In [41]: print(scaled)

[[0.00869565 0.0107949  0.00764818 ... 0.17531891 0.01452332 0.47088767]
 [0.01062802 0.01275761 0.01147228 ... 0.18363432 0.01512964 0.47178086]
 [0.01256039 0.01373896 0.01434034 ... 0.19388207 0.02419057 0.52365922]
 ...
 [0.84927536 0.87438665 0.84894837 ... 0.36300702 0.95284463 0.46984675]
 [0.85120773 0.87438665 0.84894837 ... 0.37154895 0.96172279 0.52253771]
 [0.84927536 0.85672228 0.83938815 ...        nan 0.95091512 0.40174338]]
```

- Data Normalization(為了套入後面的演算法模型)

```
In [269]: #將時間序列轉換為監督式學習
from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# specify the number of lag hours
n_days = 10 #可再調整
n_features = df_new.shape[1]
# frame as supervised learning
reframed = series_to_supervised(scaled, n_days, 1)
```

# Methodology details (2)

- 藉由將輸入的資料像量與predict輸出組成之training data 去跑監督式學習，讓其建立一個learning model

```
n_features = df_new.shape[1]
# frame as supervised learning
reframed = series_to_supervised(scaled, n_days, 1)
reframed
```

Out[269]:

| | var1(t-10) | var2(t-10) | var3(t-10) | var4(t-10) | var5(t-10) | var6(t-10) | var7(t-10) | var8(t-10) | var9(t-10) | var10(t-10) | ... | var4(t) | var5(t) | var6(t) | var7(t) | var8(t) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 0.026388 | 0.023669 | 0.025725 | 0.026611 | 0.515873 | 0.453699 | 0.562206 | 0.000000 | 0.041131 | 0.216743 | ... | 0.028011 | 0.505291 | 0.455038 | 0.562020 | 0.000000 |
| **11** | 0.024113 | 0.025944 | 0.025280 | 0.026144 | 0.489418 | 0.454455 | 0.561376 | 0.000000 | 0.035990 | 0.215251 | ... | 0.028945 | 0.518519 | 0.457133 | 0.562621 | 0.000000 |
| **12** | 0.025023 | 0.024579 | 0.024390 | 0.026611 | 0.507937 | 0.455503 | 0.561584 | 0.000000 | 0.038560 | 0.213494 | ... | 0.029879 | 0.492063 | 0.459653 | 0.561408 | 0.000000 |
| **13** | 0.027753 | 0.024124 | 0.027061 | 0.027077 | 0.518519 | 0.453978 | 0.562071 | 0.000000 | 0.043702 | 0.214065 | ... | 0.026611 | 0.481481 | 0.456775 | 0.560910 | 0.000000 |
| **14** | 0.025023 | 0.026855 | 0.026616 | 0.027077 | 0.486772 | 0.456404 | 0.561262 | 0.000000 | 0.038560 | 0.212964 | ... | 0.025210 | 0.505291 | 0.456219 | 0.561335 | 0.000000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1218** | 0.491356 | 0.508421 | 0.541570 | 0.506069 | 0.132275 | 0.173031 | 0.564260 | 0.009748 | 0.455013 | 0.676792 | ... | 0.818861 | 0.619048 | 0.399112 | 0.564219 | 0.015243 |
| **1219** | 0.548681 | 0.533910 | 0.536229 | 0.525677 | 0.835979 | 0.624185 | 0.596668 | 0.010721 | 0.516710 | 0.685867 | ... | 0.833800 | 1.000000 | 0.942124 | 0.597819 | 0.016685 |
| **1220** | 0.611465 | 0.575785 | 0.597650 | 0.590103 | 0.867725 | 0.578202 | 0.610289 | 0.011784 | 0.583548 | 0.698734 | ... | 0.929972 | 0.634921 | 0.141793 | 0.539654 | 0.017081 |
| **1221** | 0.617834 | 0.611288 | 0.650169 | 0.617180 | 0.539683 | 0.317767 | 0.574665 | 0.011892 | 0.591260 | 0.687531 | ... | 0.967320 | 0.952381 | 0.588040 | 0.577736 | 0.018396 |
| **1222** | 0.687898 | 0.654074 | 0.672423 | 0.664799 | 0.910053 | 0.720631 | 0.595568 | 0.013081 | 0.665810 | 0.706106 | ... | 1.000000 | 0.370370 | 0.337364 | 0.557196 | 0.018000 |

1213 rows × 143 columns

In [299]:

```
#用處理過後的二維數據進行train test切分
values = reframed.values
n_train_day = 1000
```

# Methodology details(2): DataFrame Output

- 將時間序列資料轉為監督式學習後的資料表格
- time shift

# Methodology details (3):Using LSTM

```
In [297]: #Design LSTM model
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout,BatchNormalization

model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
print(model.summary())

# fit network
history = model.fit(train_X, train_y, epochs=150, batch_size=256,validation_data=(test_X, test_y), ver
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_days*n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -12:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
```

- 利用LSTM來train出預測模型，並輔以細部參數微調測試

# Methodology details (4):Using stacked LSTM

```
In [300]: ###stacked LSTM

model=Sequential()
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

# fit network
history = model.fit(train_X, train_y, epochs=150, validation_data=(test_X, test_y), verbose=0, sl
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_days*n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -12:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -12:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
```

- 神經網路的深度(堆疊多層)使其較易在具有廣泛挑戰性的預測問題中取得成功。

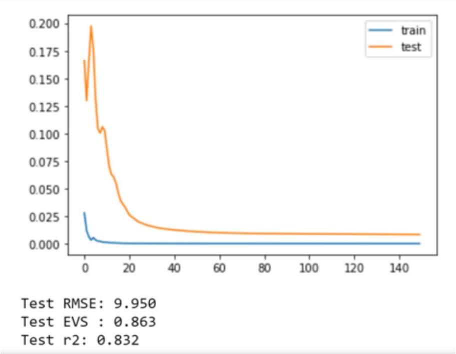
# Evaluation and Results(1)

資料切分:Train/test (訓練與驗證)、X/y (變量與target)

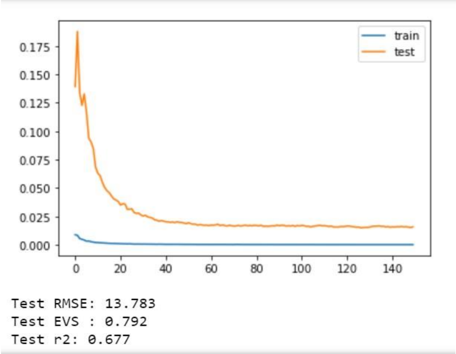
```
In [299]: #用處理過後的二維數據進行train test切分
values = reframed.values
n_train_day = 1000
train = values[:n_train_day, :]
test = values[n_train_day:, :]

#split into input and outputs
n_obs = n_days * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]

#reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_days, n_features))
test_X = test_X.reshape((test_X.shape[0], n_days, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

(1000, 10, 13) (1000,) (213, 10, 13) (213,)
```

切分資料比為4:1

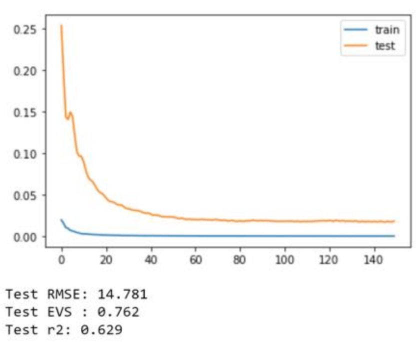
# Evaluation and Results(2):dropout size



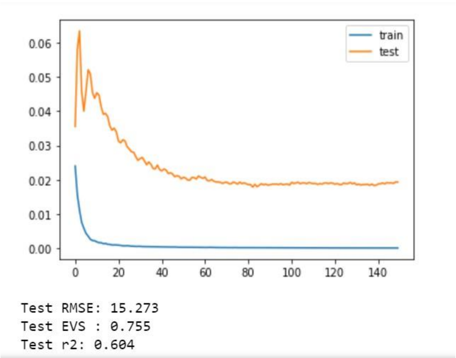

Dropout size=0.0



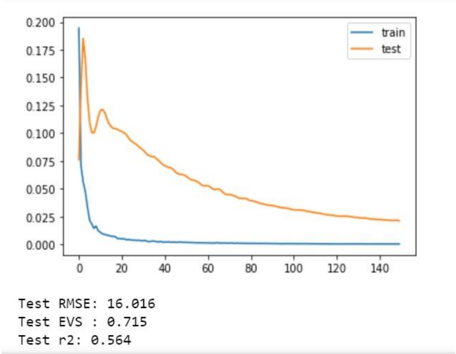

Dropout size=0.2



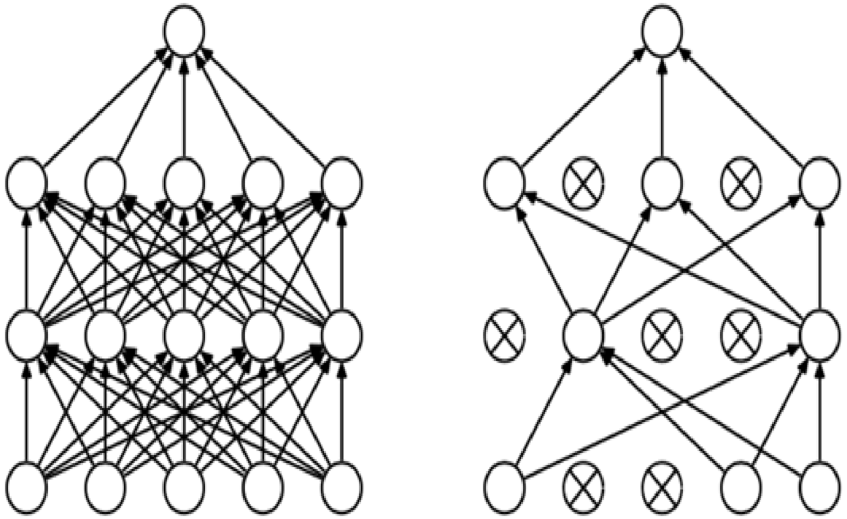

Dropout size=0.4




Dropout size=0.6




Dropout size=0.8



| Dropout / Test | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 |
|---|---|---|---|---|---|
| RMSE | 9.950 | 13.783 | 14.781 | 15.273 | 16.016 |
| EVS | 0.863 | 0.792 | 0.762 | 0.755 | 0.715 |
| r2 | 0.832 | 0.677 | 0.629 | 0.604 | 0.564 |

# Evaluation and Results(3):plot&evaluation

```
def confusion_matrix_list(real,predict):
    real_tendency=[]
    predicted_tendency=[]

    for i in range(1,len(real)):
        if(real[i]-real[i-1]>0):
            real_tendency.append(1)
        if(real[i]-real[i-1]==0):
            real_tendency.append(2)
        if(real[i]-real[i-1]<0):
            real_tendency.append(0)
    for i in range(1,len(predict)):
        if(predict[i]-predict[i-1]>0):
            predicted_tendency.append(1)
        if(predict[i]-predict[i-1]==0):
            pedicted_tendency.append(2)
        if(predict[i]-predict[i-1]<0):
            predicted_tendency.append(0)

    return real_tendency,predicted_tendency
```

自定義副程式:計算股價真實值與預測值之漲跌 “ 趨勢 ”

將趨勢list匯入sklearn之confusion_matrix並得出其各項預測精準度評比

```
real,predict=confusion_matrix_list(inv_y,inv_yhat)

from sklearn.metrics import confusion_matrix
matrix=confusion_matrix(real,predict,labels=[1,0,2])
print(matrix)

from sklearn.metrics import classification_report
report=classification_report(real,predict,labels=[1,0,2])
print(report)
```

```
[[93 39  0]
 [42 28  0]
 [ 5  5  0]]
              precision    recall  f1-score   support

           1       0.66      0.70      0.68       132
           0       0.39      0.40      0.39        70
           2       0.00      0.00      0.00        10

    accuracy                           0.57       212
   macro avg       0.35      0.37      0.36       212
weighted avg       0.54      0.57      0.56       212
```


趨勢圖(黑線為真實值，綠線為預測值)

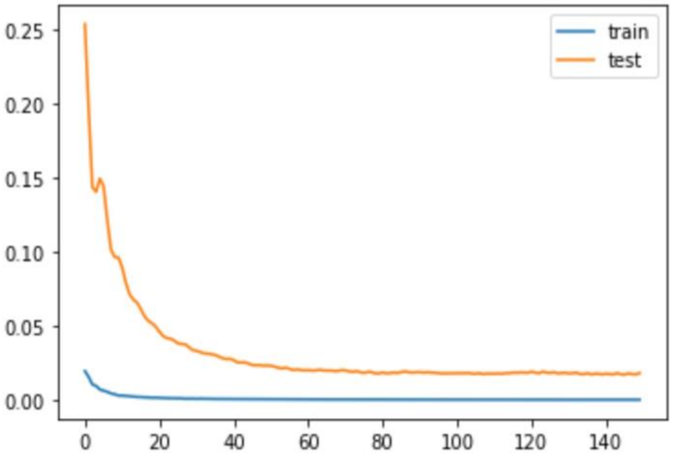
# Evaluation and Results(4):Loss Function

MAE ( Mean-Absolute Error )

MSE ( Mean-Square Error )

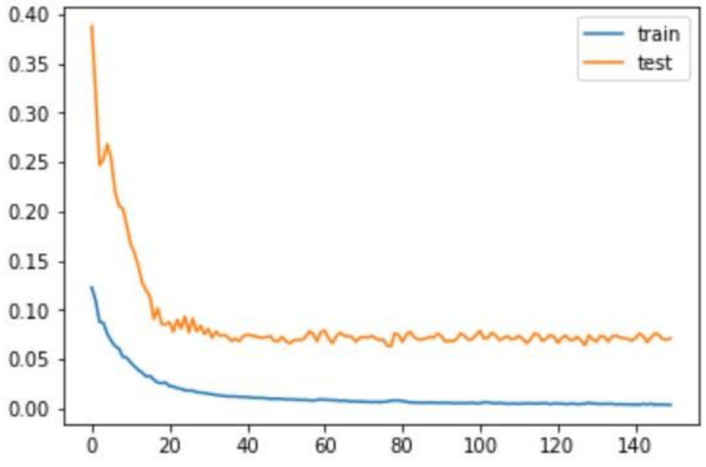
$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y-\hat{y}\right)^{2}}_{\text{The square of the difference between actual and predicted}}$$

$$MAE = \overbrace{\boxed{\frac{1}{n}}}^{\text{Divide by the total number of data points}}\underset{\text{Sum of}}{\Sigma}\underbrace{\left|\overbrace{\boxed{y}}^{\text{Actual output value}}-\overbrace{\boxed{\hat{y}}}^{\text{Predicted output value}}\right|}_{\text{The absolute value of the residual}}$$




Test RMSE: 14.781
Test EVS : 0.762
Test r2: 0.629

MSE

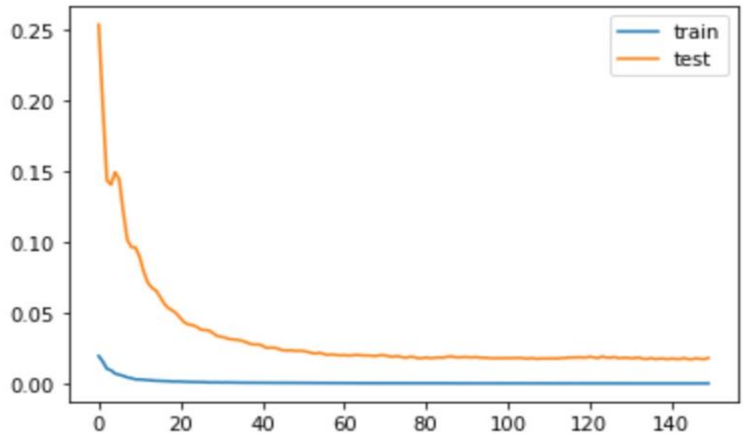



Test MAE: 7.837
Test EVS : 0.803
Test r2: 0.701

MAE

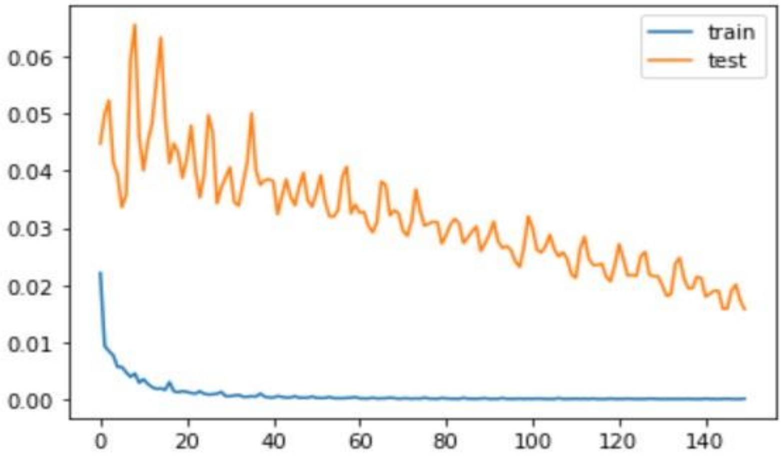
| L-func<br>Test | MSE | MAE |
|---|---|---|
| RMSE/MAE | 14.781 | 7.837 |
| EVS | 0.762 | 0.803 |
| r2 | 0.629 | 0.701 |

# Evaluation and Results(5):optimizer



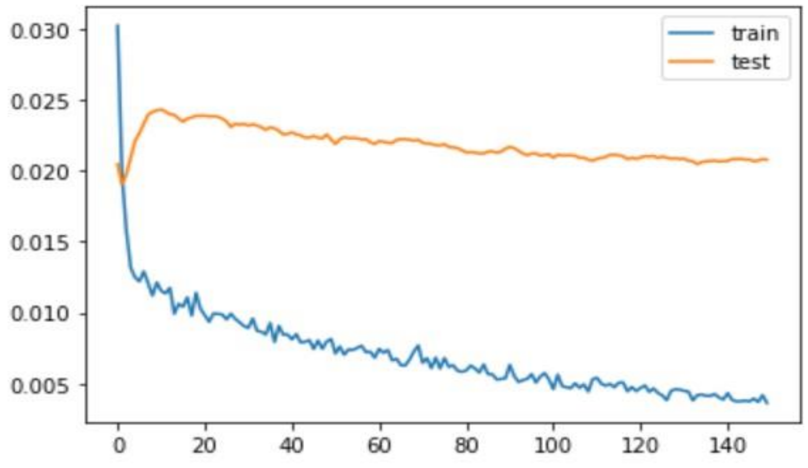

Test RMSE: 14.781
Test EVS : 0.762
Test r2: 0.629

adam




Test RMSE: 13.796
Test EVS : 0.785
Test r2: 0.677

RMSprop




Test RMSE: 15.848
Test EVS : 0.695
Test r2: 0.573

SGD

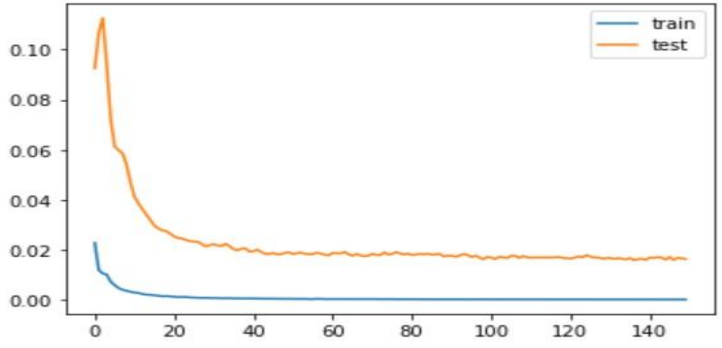
| Test \ Optimizer | adam | RMSprop | SGD |
|---|---|---|---|
| RMSE | 14.781 | 13.796 | 15.848 |
| EVS | 0.762 | 0.785 | 0.695 |
| r2 | 0.629 | 0.677 | 0.573 |

# Evaluation and Results(6):stacked LSTM

## One stack

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm_133 (LSTM) | (None, 50) | 12800 |
| dropout_76 (Dropout) | (None, 50) | 0 |
| dense_103 (Dense) | (None, 1) | 51 |

Total params: 12,851
Trainable params: 12,851
Non-trainable params: 0



Test RMSE: 14.033
Test EVS : 0.792
Test r2: 0.665

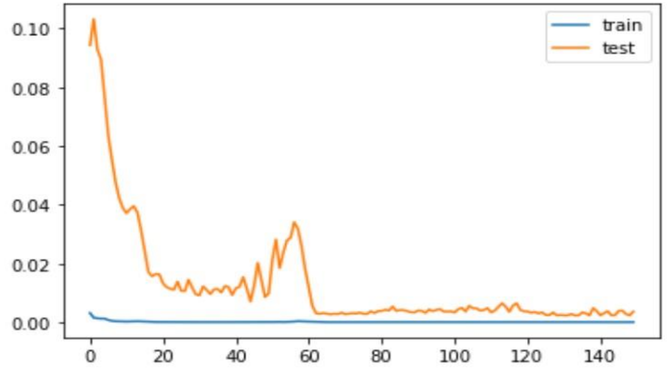
多做一次LSTM之層級堆疊

```
###stacked LSTM

model=Sequential()
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(50,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.4))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()
```

## Two stack

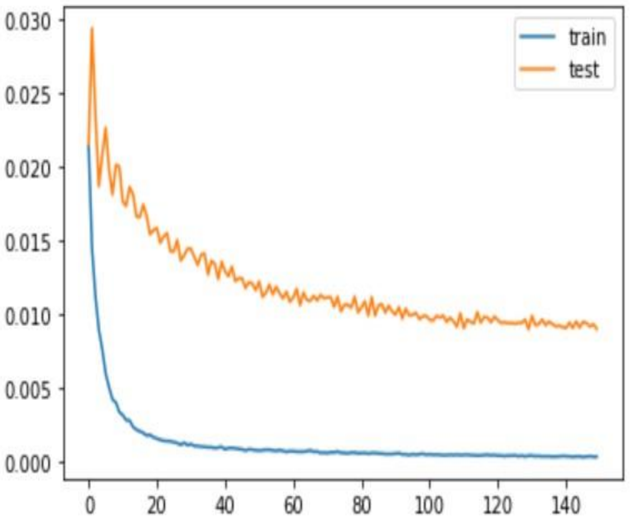
| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm_136 (LSTM) | (None, 10, 50) | 12800 |
| dropout_79 (Dropout) | (None, 10, 50) | 0 |
| lstm_137 (LSTM) | (None, 50) | 20200 |
| dropout_80 (Dropout) | (None, 50) | 0 |
| dense_105 (Dense) | (None, 1) | 51 |

Total params: 33,051
Trainable params: 33,051
Non-trainable params: 0



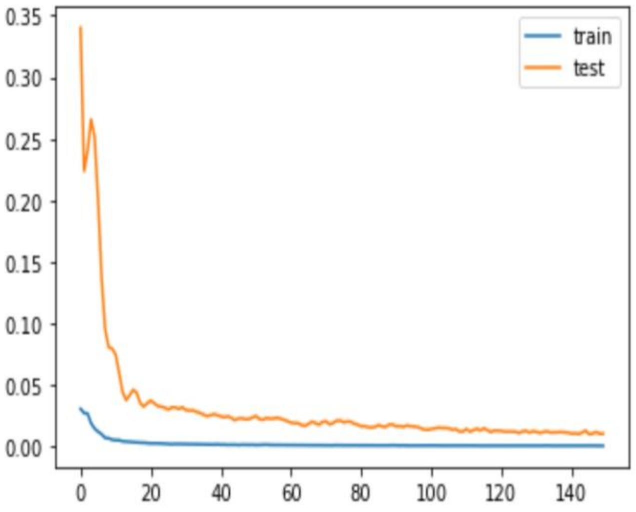
Test RMSE: 6.613
Test EVS : 0.936
Test r2: 0.926

# Conclusions and novelty(1):company



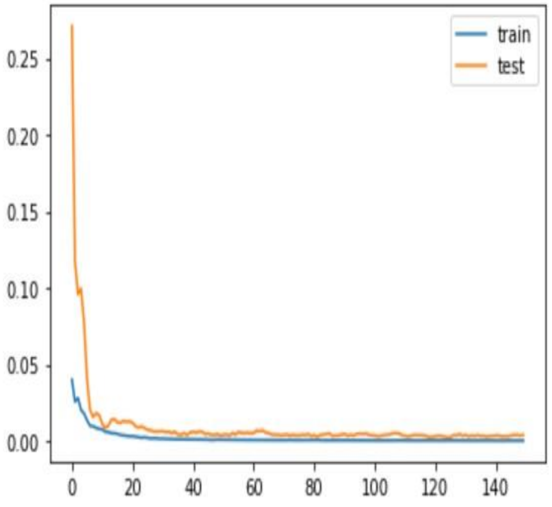

```
Test RMSE: 2.509
Test EVS : 0.852
Test r2: 0.830
```

中鋼



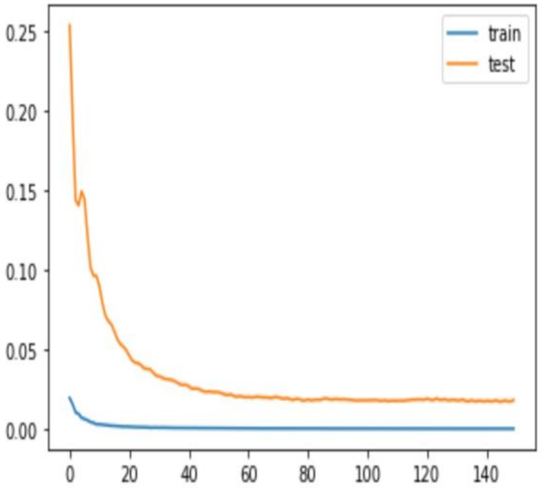

```
Test RMSE: 52.894
Test EVS : 0.906
Test r2: 0.539
```

台積電




```
Test RMSE: 63.237
Test EVS : 0.898
Test r2: 0.814
```

聯發科




```
Test RMSE: 14.781
Test EVS : 0.762
Test r2: 0.629
```

長榮




# Conclusions and novelty(2)

1.監督式學習之標籤:以10天為step size(是否可以更進一步嘗試)

```
In [18]: training_set_scaled_x

Out[18]: array([[0.0107949 ],
               [0.01275761],
               [0.01373896],
               ...,
               [0.87438665],
               [0.87438665],
               [0.85672228]])
```

2.多重共線性:

現實生活中，很難找到一組互不相關，又對因變數y產生主要影響的變數。

# The contribution of each team member

| 組員名稱 | 貢獻 |
|---|---|
| 統計系112林家同 | 1.使用網路爬蟲自FinMind獲取Dataset並找出較佳的特徵值<br>2.測試不同的LSTM函式參數並嘗試找出最佳組合<br>3.測試數據&圖表之統整 |
| 資訊系112莊上緣 | 1.Final project 投影片製作<br>2.建構自定義副程式並將部分參數微調測試模塊化<br>3.深度學習之概念查詢並彙整 |
| 資訊系112李培綸 | 1.建構UI圖像化使用者介面<br>2.LSTM 框架想法提供與觀念釋疑<br>3.將最終程式碼進行優化並封包 |