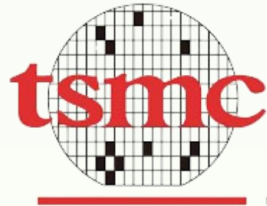


# 以統計方法與AI模型 進行股價分類預測

製作人：林萍珍、黃宥輔、楊育傑

包含程式碼：

- 1.download\_data.py
- 2.feature processing.py
- 3.Min-Max Normalization.py
- 4.predict.py
- 5.lstm\_split\_data.py



## 目標: 預測5日後收盤價的漲跌

### 特徵(x)

股價資料:開盤價、最高價、最低價、收盤價、成交量

前n天的收盤價:前五、十、十五、二十天收盤價

技術指標: MA\_20、RSI\_14、MACD

### 目標(y)

五日後收盤價漲跌, 漲(1)跌(0)

### 運用的模型

SVM、KNN、XGBoost、RF、LSTM、邏輯式迴歸、貝式分類器

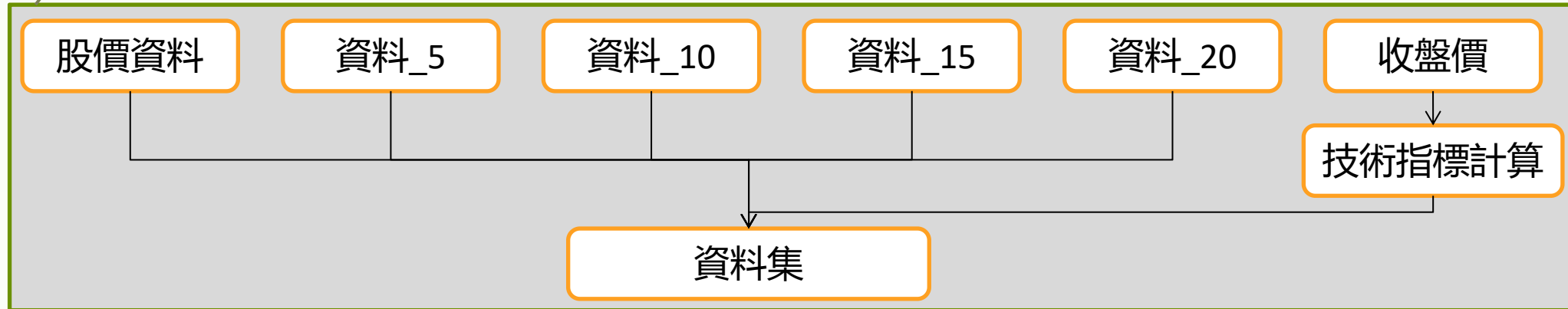
資料期間:2020/01/01-2024/05/01

公司名稱:台灣積體電路製造

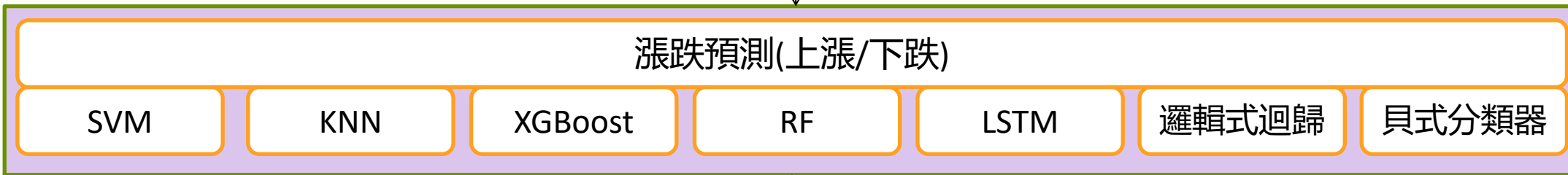
股價代號:2330.TW



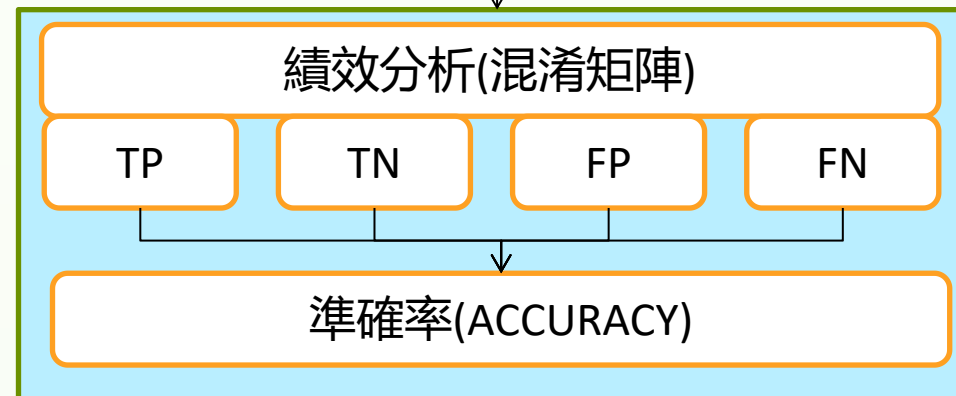
## 流程圖



股價資料:  
開盤價  
最高價  
最低價  
收盤價  
成交量



技術指標:  
MA\_20  
RSI\_14  
MACD



資料\_n:  
前n天的資料  
(收盤價、MA\_20、RSI\_14、MACD)  
(n = 5、10、15、20)

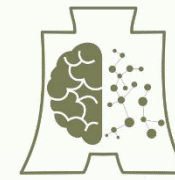
## 績效分析(混淆矩陣)

- **準確率 (Accuracy):** 模型預測正確數量佔整體的比例  $(TP+TN) / (TP+TN+FP+FN)$
- 精確率 (Precision): 模型預測為正(漲)有多少真的為正(漲)  $TP/(TP+FP)$
- 召回率 (Recall): 實際為正的資料有多少被模型預測為正  $TP/(TP+FN)$
- F1分數 (F1 Score): 精確度和召回率的調和平均值  $2 * (Precision * Recall) / (Precision + Recall)$

真實 \ 預測	上漲	下跌
	上漲	下跌
上漲	TP	FP
下跌	FN	TN

涵義

真實 \ 預測	上漲	下跌
	上漲	下跌
上漲	預測正確 true positive(TP)	預測錯誤 false positive(FP)
下跌	預測錯誤 false negative(FN)	預測正確 true negative(TN)



## 安裝套件

- 需要安裝後續會使用到的套件
- 1.yfinance(下載股價資料)
  - 2.talib(計算技術指標)
  - 3.xgboost(xgboost預測模型)
  - 4.tensorflow(LSTM預測模型)

```
File c:\users\ai lab\downloads\股價預測\股價下載.py:8
import yfinance as yf
```

```
ModuleNotFoundError: No module named 'yfinance'
```

安裝yfinance套件

程式碼:**pip install yfinance**

```
File c:\users\ai lab\downloads\股價預測\機器學習預測.py:9
from xgboost import XGBClassifier
```

```
ModuleNotFoundError: No module named 'xgboost'
```

安裝xgboost套件

程式碼:**pip install xgboost**

```
File c:\users\ai lab\downloads\股價預測\特徵整理.py:4
import talib
```

```
ModuleNotFoundError: No module named 'talib'
```

安裝talib套件

程式碼:**pip install ta-lib**

```
File c:\users\ai lab\downloads\股價預測\lstm預測.py:4
from tensorflow.keras.models import Sequential
```

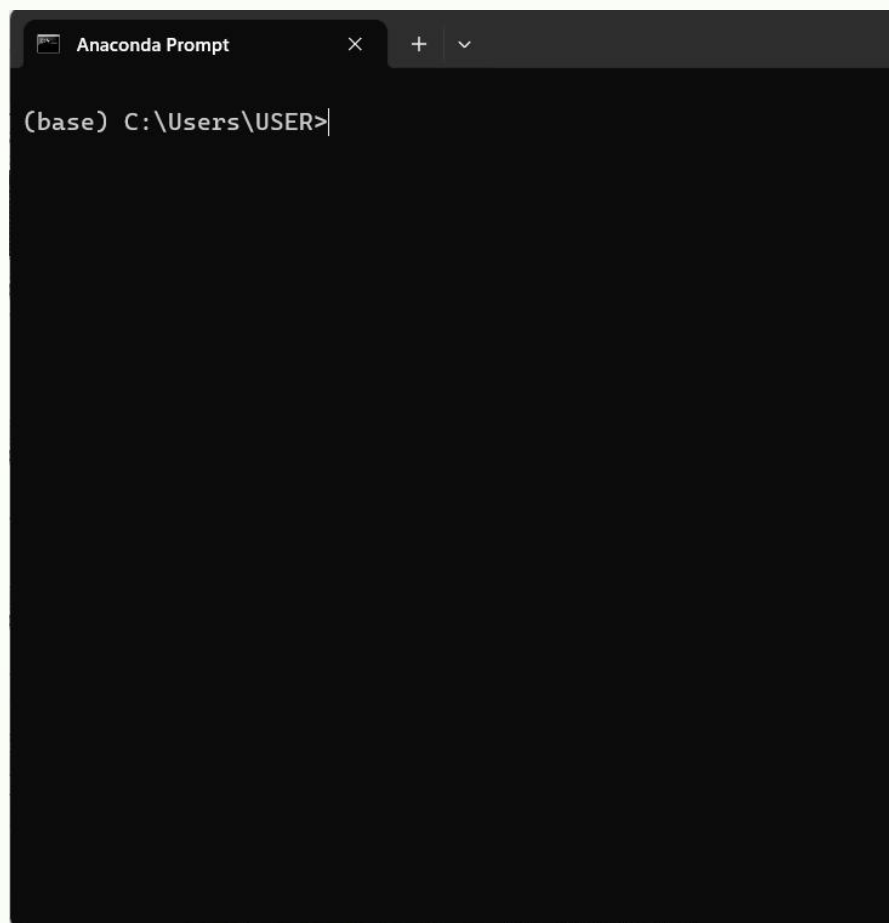
```
ModuleNotFoundError: No module named 'tensorflow'
```

安裝tensorflow套件

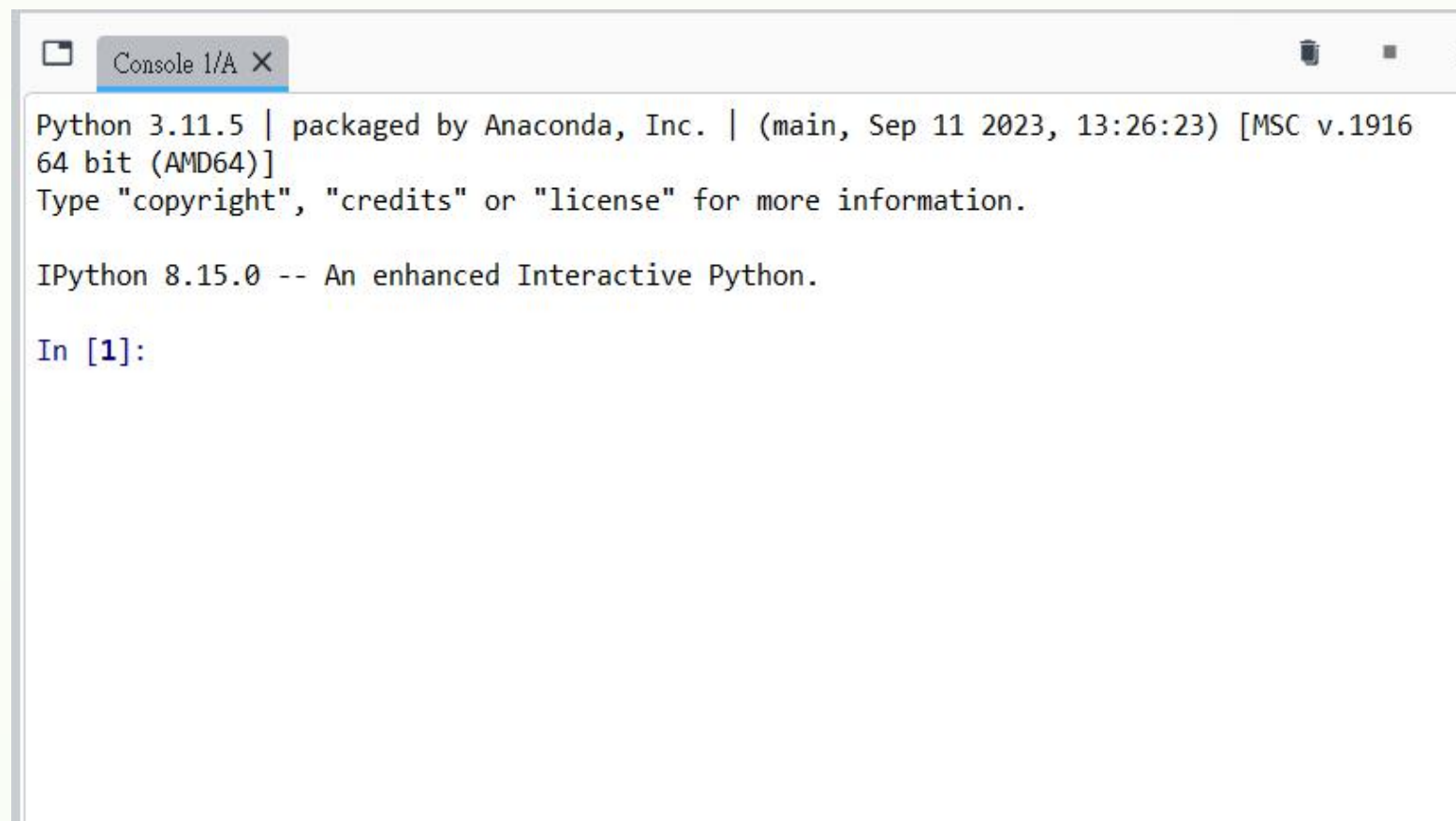
程式碼:**pip install tensorflow**



- 安裝方法一，開啟Anaconda Prompt 輸入程式碼
- 安裝方法二，於spyder 控制台(Console)中輸入程式碼



```
Anaconda Prompt
(base) C:\Users\USER>
```

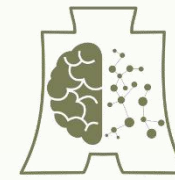


```
Console 1/A X
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.15.0 -- An enhanced Interactive Python.

In [1]:
```

## 安裝套件



AI.FINTECH

AI 金融 科技 中心

- 安裝套件完成後，需要重新從Anaconda Navigator啟動Spyder更新套件

```
\programdata\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance)
(2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in c:\programdata\anaconda3\lib\site-
packages (from pandas>=1.3.0->yfinance) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:
\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-
packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:
\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinar
Requirement already satisfied: certifi>=2017.4.17 in c:
\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2024.6.2)
Note: you may need to restart the kernel to use updated packages.
```

出現提示需重新啟動

開啟Anaconda Navigator



Anaconda Navigator



Spyder

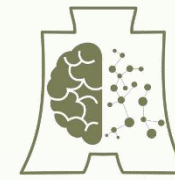
5.4.3

Scientific PYTHON Development  
EnviRonment. Powerful Python IDE with  
advanced editing, interactive testing,  
debugging and introspection features

Launch

重新啟動Spyder

## 安裝套件



- ta-lib安裝失敗處理

```
_ta_lib.c
talib/_ta_lib.c(1223): fatal error C1083: 無法開啟包含檔案: 'ta_libc.h': No such
file or directory
error: command 'C:\\Program Files (x86)\\Microsoft Visual Studio\\2019\\
\\Community\\VC\\Tools\\MSVC\\14.29.30133\\bin\\HostX86\\x64\\cl.exe' failed with
exit code 2
[end of output]
```

```
note: This error originates from a subprocess, and is likely not a problem with
pip.
ERROR: Failed building wheel for ta-lib
ERROR: Could not build wheels for ta-lib, which is required to install
pyproject.toml-based projects
```

spyder 控制台(Console)安裝失敗

```
Kits\\10\\include\\10.0.19041.0\\ucrt" "-IC:\\Program Files (x86)\\Windows talib/_ta_lib.c(1223): fatal error
C1083: 無法開啟包含檔案: 'ta_libc.h': No such file or directory
error: command 'C:\\Program Files (x86)\\Microsoft Visual
Studio\\2019\\Community\\VC\\Tools\\MSVC\\14.29.30133\\bin\\HostX86\\x64\\cl.exe' failed with
exit code 2
[end of output]
```

Anaconda Prompt安裝失敗

```
note: This error originates from a subprocess, and is likely not a problem with pip.
ERROR: Failed building wheel for ta-lib
Failed to build ta-lib
ERROR: Could not build wheels for ta-lib, which is required to install pyproject.toml-based projects
```



- ta-lib安裝失敗處理

1.請下載壓縮檔(TA\_Lib-0.4.28-cp312-cp312-win\_amd64.zip)並解壓縮

下載網址:[https://github.com/TA-Lib/ta-lib-python/files/13259509/TA\\_Lib-0.4.28-cp312-cp312-win\\_amd64.zip](https://github.com/TA-Lib/ta-lib-python/files/13259509/TA_Lib-0.4.28-cp312-cp312-win_amd64.zip)

2.開啟Anaconda Prompt輸入cd 切換到下載路徑 `cd C:\Users\USER\Downloads`

3.安裝下載好的檔案 `pip install TA_Lib-0.4.28-cp312-cp312-win_amd64.whl`

```
(base) C:\Users\AI LAB>cd C:\Users\AI LAB\Downloads
```

切換到下載路徑

```
(base) C:\Users\AI LAB\Downloads>pip install TA_Lib-0.4.28-cp312-cp312-win_amd64.whl
```

安裝檔案

```
Defaulting to user installation because normal site-packages is not writeable
```

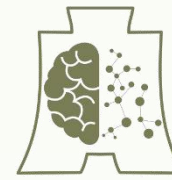
```
Processing c:\users\ai lab\downloads\ta_lib-0.4.28-cp312-cp312-win_amd64.whl
```

```
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from TA-Lib==0.4.28) (1.26.4)
```

```
Installing collected packages: TA-Lib
```

```
Successfully installed TA-Lib-0.4.28
```

完成安裝



## 安裝套件

### • ta-lib安裝失敗處理

不確定存放路徑、檔案名稱可以右鍵到內容中複製

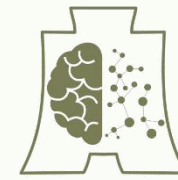
The image shows a Windows file explorer window with a context menu open for a file named 'TA\_Lib-0.4.28-cp312-cp312-win\_amd64.whl'. The menu options include '開啟(O)', '使用 Skype 分享', '以 WinRAR 開啟(W)', '解壓縮檔案(A)...', and '解壓縮至此(X)'. A green callout bubble labeled '解壓縮' points to the '解壓縮檔案(A)...' option. Below the file explorer, a file properties window is open for the same file. The '一般' tab is selected, showing the file name 'TA\_Lib-0.4.28-cp312-cp312-win\_amd64.whl' (highlighted with a red box and a green callout bubble labeled '檔案名稱'), the file type 'WHL 檔案 (.whl)', and the location 'C:\Users\USER\Downloads' (highlighted with a red box and a green callout bubble labeled '檔案儲存路徑'). A green callout bubble labeled '開啟檔案內容' points to the '內容(R)' option in the file explorer's context menu.

解壓縮

檔案名稱

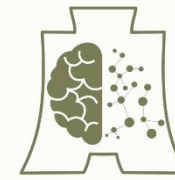
開啟檔案內容

檔案儲存路徑



# 程式撰寫 - 股價下載

- 運用yfinance下載股價資料(開、高、低、收、成交量)
- 將收盤價畫圖呈現



## download\_data.py完整程式碼

```
import yfinance as yf
import matplotlib.pyplot as plt

stock_symbol = '2330.TW' # 輸入股票代號下載股價資料

stock_data = yf.download(stock_symbol, start='2010-01-01', end='2024-05-01') # 獲取特定日期範圍的股票資料

excel_filename = f'{stock_symbol}_stock_data.xlsx' # 將股票資料存儲為 Excel 檔案，以股票代號作為檔案名稱
stock_data.to_excel(excel_filename)

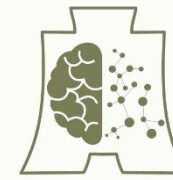
print(f"股票資料已存儲為 '{excel_filename}'")
print(stock_data)

stock_data['Close'].plot() # 用stock_data的CLOSE畫出圖形
plt.xlabel("Date") # x軸的標籤
plt.ylabel("Closing Price") # y軸的標籤
plt.title(f'{stock_symbol} Stock Price') # 圖標題
plt.show() # 顯示圖片
```

**程式位置**

檔案:download\_data.py





## download\_data.py

```
8 import yfinance as yf
9 import matplotlib.pyplot as plt
10
11 stock_symbol = '2330.TW' # 輸入股票代號下載股價資料
12 stock_data = yf.download(stock_symbol, start='2010-01-01', end='2024-05-01') # 獲取股票資料
13
14 excel_filename = f'{stock_symbol}_stock_data.xlsx' # 將股票資料存儲為 Excel 檔案，以股票代號作為檔案名稱
15 stock_data.to_excel(excel_filename)
16
17 print(f"股票資料已存儲為 '{excel_filename}'")
18 print(stock_data)
```

輸入股票代號、開始及  
結束日期進行股價下載

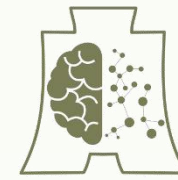
下載好的資料存成  
excel檔，並用股票代  
號命名

```
20 stock_data['Close'].plot() # 用stock_data的CLOSE畫出圖形
21 plt.xlabel("Date") # x軸的標籤
22 plt.ylabel("Closing Price") # y軸的標籤
23 plt.title(f'{stock_symbol} Stock Price') # 圖標題
24
25 plt.show() # 顯示圖片
```

運用收盤價畫圖，設定  
x、y軸的文字及圖表的  
標題

程式位置

檔案:download\_data.py



## 程式撰寫 - 特徵整理

- 運用talib進行技術指標計算(MA\_20、RSI\_14、MACD)
- 取得前5天、前10天、前15天、前20天的資訊(收盤價、MA\_20、RSI\_14、MACD)
- 收盤價計算價格變化，並進行標記(漲:1，跌:0)



## feature processing.py完整程式碼

```
import pandas as pd
import talib

stock_data = pd.read_excel('2330.TW_stock_data.xlsx', index_col='Date') # 讀取股價資料
missing_values = stock_data.isnull().sum() # 檢查每一列是否有空值
print(missing_values)

stock_data.drop(columns=['Adj Close'], inplace=True)
df_close = stock_data['Close']
stock_data['MA_20'] = talib.SMA(df_close, 20) # 計算MA20
stock_data['RSI_14'] = talib.RSI(df_close, 14) # 計算RSI
macd, macdsignal, macdhist = talib.MACD(df_close, fastperiod=12, slowperiod=26, signalperiod=9) # 計算MACD
stock_data['MACD'] = macd # 將MACD計算結果存回資料中

columns_to_shift = ['Close', 'MA_20', 'RSI_14', 'MACD'] # 選取需要進行處理的欄位名稱

for period in range(5, 21, 5): # 運用迴圈帶入前N期收盤價
    for column in columns_to_shift: # 運用迴圈走訪所選的欄位名稱
        stock_data[f'{column}_{period}'] = stock_data[column].shift(period) # 運用.shift()方法取得收盤價
```

程式位置

檔案:feature processing.py



## feature processing.py完整程式碼

```
stock_data['Next_5Day_Return']= stock_data['Close'].diff(5).shift(-5) #計算價格變化
def classify_return(x):
    return 1 if x > 0 else 0 # 標示漲跌，大於0標示為漲(1)，小於0標示為跌(0)

stock_data['LABEL'] = stock_data['Next_5Day_Return'].apply(classify_return) # 創造新的一列LABEL來記錄漲跌

stock_data = stock_data.dropna() # 刪除因技術指標計算出現的空值
stock_data.to_excel("data.xlsx") #將整理好的資料存成excel

ones_count = (stock_data['LABEL'] == 1).sum() # 計算資料為1的數量(將LABEL欄位加總計算1的數量)
zeros_count = len(stock_data) - ones_count # 計算資料為0的數量(全部資料數減上漲數)

print(f"1(上漲): {ones_count}")
print(f"0(下跌): {zeros_count}")
```

**程式位置**

檔案:feature processing.py





## feature processing.py

```
3 import pandas as pd
4 import talib
```

```
6 stock_data = pd.read_excel('2330.TW_stock_data.xlsx', index_col='Date') # 讀取股價資料
7 missing_values = stock_data.isnull().sum() # 檢查每一列是否有空值
8 print(missing_values)
```

讀取下載好的股價資料，  
檢查是否有空值

```
10 stock_data.drop(columns=['Adj Close'], inplace=True)
```

```
11 df_close = stock_data['Close']
```

```
12 stock_data['MA_20'] = talib.SMA(df_close, 20) # 計算MA20
```

```
13 stock_data['RSI_14'] = talib.RSI(df_close, 14) # 計算RSI
```

```
14 macd, macdsignal, macdhist = talib.MACD(df_close, fastperiod=12, slowperiod=26, signalperiod=9) # 計算MACD
```

```
15 stock_data['MACD'] = macd # 將MACD計算結果存回資料中
```

刪除不需要的欄位  
(Adj Close調整後的股價)

```
18 columns_to_shift = ['Close', 'MA_20', 'RSI_14', 'MACD'] # 選取需要進行處理的欄位名稱
```

```
19
20 for period in range(5, 21, 5): # 運用迴圈帶入前N期收盤價
```

```
21     for column in columns_to_shift: # 運用迴圈走訪所選的欄位名稱
```

```
22         stock_data[f'{column}_{period}'] = stock_data[column].shift(period) # 運用.shift()方法取得收盤價
```

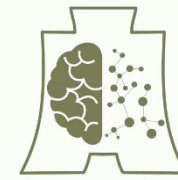
計算技術指標MA\_20、  
RSI\_14、MACD

運用迴圈查詢前5天、  
前10天、前15天、前  
20天的資料

程式位置

檔案:feature processing.py

## feature processing.py



```

24 stock_data['Next_5Day_Return']= stock_data['Close'].diff(5).shift(-5)
25 def classify_return(x):
26     return 1 if x > 0 else 0 # 標示漲跌，大於0標示為漲(1)，小於0標示為跌(0)
27
28 stock_data['LABEL'] = stock_data['Next_5Day_Return'].apply(classify_return)
29
30
31 stock_data = stock_data.dropna() # 刪除因技術指標計算出現的空值
32 stock_data.to_excel("data.xlsx") # 將整理好的資料存成excel

```

計算價格變化，結果往前移動5天

漲跌幅標記，變化大於0  
為漲(1)，反之為跌(0)

刪除因技術指標計算產生  
的空值資料，將資料存成  
excel檔

```

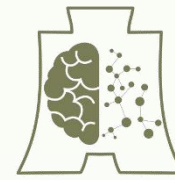
34 ones_count = (stock_data['LABEL'] == 1).sum() # 計算資料為1的數量(將LABEL欄位加總，算1的數量)
35 zeros_count = len(stock_data) - ones_count # 計算資料為0的數量
36
37 print(f"1(上漲): {ones_count}")
38 print(f"0(下跌): {zeros_count}")
39

```

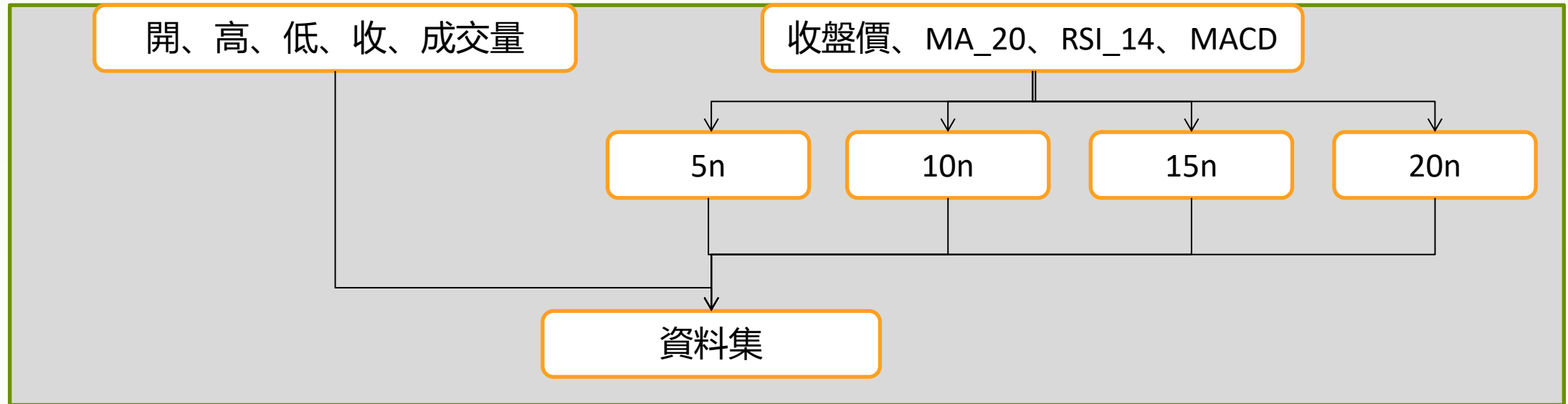
計算上漲(1)跟下跌(1)的數量

程式位置

檔案:feature processing.py



## feature processing



資料名稱	股價資料	技術指標	資料_N	RETURN	LABEL	欄位總數
資料欄位	5	3	$4 * 4 = 16$	1	1	26

**原始資料數**

**1090**

MACD\_20(MACD空33筆, \_20空20筆)

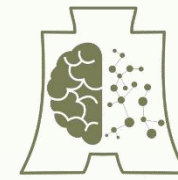
前面空53筆

RETURN

後面空5筆

**剩餘資料列**

**1032(1090-53-5)**

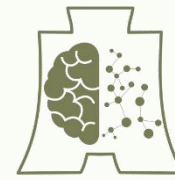


# 程式撰寫 - 正規化

- 將所有資料進行正規劃，使資料縮放於 $[0,1]$ 之間



## Min-Max Normalization.py完整程式碼



```
import pandas as pd

stock_data = pd.read_excel('data.xlsx', index_col='Date')
columns_to_exclude = ['Next_5Day_Return', 'LABEL'] # 需要排除的欄位
columns_to_normalize = stock_data.columns.difference(columns_to_exclude) # 分離不需要正規化的欄位['Next_Day_Return', 'LABEL']

def min_max_normalize(column):
    return (column - column.min()) / (column.max() - column.min()) # 定義 Min-Max 正規化函數進行正規化

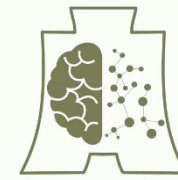
df_normalized = stock_data[columns_to_normalize].apply(min_max_normalize) # 針對需要正規化的欄位進行正規化
df_final = pd.concat([df_normalized, stock_data[columns_to_exclude]], axis=1) # 合併正規化後的數據和不需要正規化的欄位

df_final.to_csv('min_max_normalized_data.csv')
print(df_final)
```

程式位置

檔案:Min-Max Normalization.py

## Min-Max Normalization.py



```
2 import pandas as pd
3
4 stock_data = pd.read_excel('data.xlsx', index_col='Date')
5 columns_to_exclude = ['Next_5Day_Return', 'LABEL'] # 需要排除的欄位
6 columns_to_normalize = stock_data.columns.difference(columns_to_exclude) # 分離不需要正規化的欄位
7
8 def min_max_normalize(column):
9     return (column - column.min()) / (column.max() - column.min()) # 定義正規化函數
10
11 df_normalized = stock_data[columns_to_normalize].apply(min_max_normalize) # 針對需要正規化的欄位進行正規化
12
13 df_final = pd.concat([df_normalized, stock_data[columns_to_exclude]], axis=1) # 合併正規化後的數據和不需要正規化的欄位
14
15 df_final.to_csv('min_max_normalized_data.csv')
16 print(df_final)
```

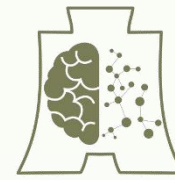
排除不需要正規化的欄位

定義正規化函數，將資料正規化

正規化與無正規化資料合併

程式位置

檔案:Min-Max Normalization.py



## 程式撰寫 - 預測

- 特徵整理(股價資料<開、高、低、收、成交量>、  
技術指標<MA\_20、RSI\_14、MACD>  
前N期<5、10、15、20>資料<收盤價、MA\_20、RSI\_14、MACD>)  
(共24個)
- 標籤整理(漲跌)(1個)
- 資料分割(訓練集、測試集)
- 機器學習預測
- 深度學習預測
- 準確度分析



## predict.py完整程式碼

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,GRU,Bidirectional,Dense,Dropout
from tensorflow.keras.optimizers import Adam
import lstm_split_data

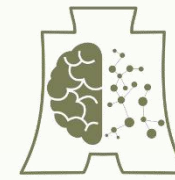
df = pd.read_csv('min_max_normalized_data.csv',index_col='Date')

def split_stock_data(stock_data, label_column, delete_column, test_size=0.3, random_state=42):
    X = stock_data.drop([label_column, delete_column], axis=1).values # X為特徵, 刪除非特徵的欄位
    y = stock_data[label_column].values # y為標籤(LABEL)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state) # 資料分割
    return X_train, X_test, y_train, y_test

label_column = 'LABEL' # 標籤欄位
delete_column = 'Next_5Day_Return' # 刪除的欄位
trainX, testX, trainY, testY = split_stock_data(df, label_column, delete_column)
```



## predict.py完整程式碼

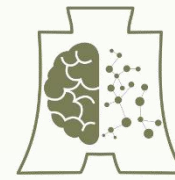


```
model_accuracies = {} # 建立空字典儲存所有測試及準確度

# KNN
KNN = KNeighborsClassifier() # 設定KNN模型
KNN.fit(trainX, trainY) # 模型訓練
train_acc = KNN.score(trainX, trainY) # 訓練集準確度計算
test_acc = KNN.score(testX, testY) # 測試集準確度計算
model_accuracies['KNN'] = test_acc # 將測試集結果儲存到字典中
print('KNN訓練集準確率 %.2f' % train_acc) # 輸出訓練集的準確度
print('KNN驗證集準確率 %.2f' % test_acc) # 輸出驗證集的準確度

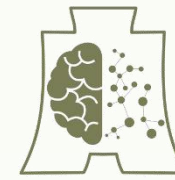
# SVM
SVM = svm.SVC() # 設定模型
SVM.fit(trainX, trainY) # 模型訓練
train_acc = SVM.score(trainX, trainY) # 訓練集準確度計算
test_acc = SVM.score(testX, testY) # 將測試集結果儲存到字典中
model_accuracies['SVM'] = test_acc # 將測試集結果儲存到字典中
print('SVM訓練集準確率 %.2f' % train_acc)
print('SVM驗證集準確率 %.2f' % test_acc)
```

## predict.py完整程式碼



```
# Gaussian Naive Bayes
Bayesian = GaussianNB()
Bayesian.fit(trainX, trainY)
train_acc = Bayesian.score(trainX, trainY) #訓練集準確度計算
test_acc = Bayesian.score(testX, testY) #測試集準確度計算
model_accuracies['GaussianNB'] = test_acc #將測試集結果儲存到字典中
print('Bayes訓練集準確率 %.2f' % train_acc)
print('Bayes測試集準確率 %.2f' % test_acc)

# Logistic Regression
Logistic = LogisticRegression()
Logistic.fit(trainX, trainY)
train_acc = Logistic.score(trainX, trainY) #訓練集準確度計算
test_acc = Logistic.score(testX, testY) #測試集準確度計算
model_accuracies['LogisticRegression'] = test_acc #將測試集結果儲存到字典中
print('LR訓練集準確率 %.2f' % train_acc)
print('LR測試集準確率 %.2f' % test_acc)
```

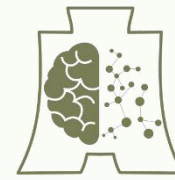


## predict.py完整程式碼

```
# RandomForest
RF = RandomForestClassifier()
RF.fit(trainX, trainY) # 模型訓練
train_acc = RF.score(trainX, trainY) #訓練集準確度計算
test_acc = RF.score(testX, testY) #測試集準確度計算
model_accuracies['RandomForest'] = test_acc #將測試集結果儲存到字典中
print('RF訓練集準確率 %.2f' % train_acc)
print('RF測試集準確率 %.2f' % test_acc)

# XGBoost
Xgboost = XGBClassifier()
Xgboost.fit(trainX, trainY)
train_acc = Xgboost.score(trainX, trainY) #訓練集準確度計算
test_acc = Xgboost.score(testX, testY) #測試集準確度計算
model_accuracies['XGBoost'] = test_acc #將測試集結果儲存到字典中
print('Xgboost訓練集準確率 %.2f' % train_acc)
print('Xgboost測試集準確率 %.2f' % test_acc)
```

## predict.py完整程式碼



```
trainX, testX, trainY, testY = lstm_split_data.split_stock_data(df, label_column, delete_column) #運用lstm預測的資料分割(與機器學習不同)
```

```
# 創建LSTM模型
```

```
model = Sequential()
```

```
model.add(LSTM(units=100, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))
```

```
model.add(Dropout(0.3))
```

```
model.add(LSTM(units=100, return_sequences=False))
```

```
model.add(Dropout(0.3))
```

```
model.add(Dense(units=1, activation='relu'))
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy']) # 編譯模型
```

```
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2) # 訓練模型
```

```
train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0) # 訓練集準確度計算
```

```
test_loss, test_acc = model.evaluate(testX, testY, verbose=0) # 測試集準確度計算
```

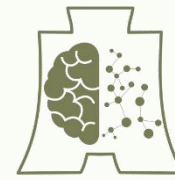
```
model_accuracies['LSTM'] = test_acc #將測試集結果儲存到字典中
```

```
print('LSTM訓練集準確率: %.4f' % train_acc)
```

```
print('LSTM測試集準確率: %.4f' % test_acc)
```



## predict.py完整程式碼

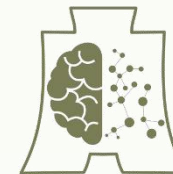


```
# 創建GRU模型(GRU)
model = Sequential()
model.add(GRU(units=100, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(Dropout(0.3))
model.add(GRU(units=100, return_sequences=False))
model.add(Dropout(0.3))
model.add(Dense(units=1, activation='relu'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy'])
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2)

train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0)
test_loss, test_acc = model.evaluate(testX, testY, verbose=0)
model_accuracies['GRU'] = test_acc
print('GRU訓練集準確率: %.2f' % train_acc)
print('GRU測試集準確率: %.2f' % test_acc)
```

## predict.py完整程式碼



```
# 創建BI-LSTM模型(Bidirectional)
model = Sequential()
model.add(Bidirectional(LSTM(units=100, return_sequences=True), input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(units=100, return_sequences=False)))
model.add(Dropout(0.3))
model.add(Dense(units=1, activation='relu'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy'])
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2)

train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0)
test_loss, test_acc = model.evaluate(testX, testY, verbose=0)
model_accuracies['BI-LSTM'] = test_acc
print('BI-LSTM訓練集準確率: %.2f' % train_acc)
print('BI-LSTM測試集準確率: %.2f' % test_acc)

best_model = max(model_accuracies, key=model_accuracies.get) # 找出準確率最高的模型
best_accuracy = model_accuracies[best_model] # 找出準確率最高的模型的名稱
print(f'準確率最高的模型是 {best_model}, 準確率為 %.2f' % best_accuracy)
```

**程式位置**  
檔案:predict.py



## predict.py

載入套件

讀取正規化後的資料

運用函數進行資料拆分，  
同時處理標籤資料跟特  
徵資料

設定標籤欄位跟非特徵欄位

根據標籤跟非特徵欄位進行  
訓練集、測試集資料分割

程式位置  
檔案:predict.py

```
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn import svm
6 from sklearn.ensemble import RandomForestClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.naive_bayes import GaussianNB
9 from xgboost import XGBClassifier
10 from tensorflow.keras.models import Sequential
11 from tensorflow.keras.layers import LSTM, GRU, Bidirectional, Dense, Dropout
12 from tensorflow.keras.optimizers import Adam
13 import lstm_split_data
```

```
15 df = pd.read_csv('min_max_normalized_data.csv', index_col='Date')
```

```
17 def split_stock_data(stock_data, label_column, delete_column, test_size=0.3, random_state=42):
18     X = stock_data.drop([label_column, delete_column], axis=1).values # X為特徵，刪除非特徵的欄位
19     y = stock_data[label_column].values # y為標籤(LABEL)
20     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state) # 資料分割
21     return X_train, X_test, y_train, y_test
```

```
23 label_column = 'LABEL' # 標籤欄位
24 delete_column = 'Next_5Day_Return' # 刪除的欄位
25
26 trainX, testX, trainY, testY = split_stock_data(df, label_column, delete_column)
```



## predict.py

建立空字典儲存預測準確度

```

27 model_accuracies = {} # 建立空字典儲存所有測試及準確度
28
29 # KNN
30 KNN = KNeighborsClassifier() # 設定KNN模型
31 KNN.fit(trainX, trainY) # 模型訓練
32 train_acc = KNN.score(trainX, trainY) # 訓練集準確度計算
33 test_acc = KNN.score(testX, testY) # 測試集準確度計算
34 model_accuracies['KNN'] = test_acc # 將測試集結果儲存到字典中
35
36 print('KNN訓練集準確率 %.2f' % train_acc) # 輸出訓練集的準確度
37 print('KNN測試集準確率 %.2f' % test_acc) # 輸出測試集的準確度
38
39 # SVM
40 SVM = svm.SVC() # 設定模型
41 SVM.fit(trainX, trainY) # 模型訓練
42 train_acc = SVM.score(trainX, trainY) # 訓練集準確度計算
43 test_acc = SVM.score(testX, testY) # 將測試集結果儲存到字典中
44 model_accuracies['SVM'] = test_acc # 將測試集結果儲存到字典中
45
46 print('SVM訓練集準確率 %.2f' % train_acc)
47 print('SVM測試集準確率 %.2f' % test_acc)

```

載入模型並進行訓練

計算訓練集、測試集準確度

將模型的預測結果存於字典中

其他模型一樣步驟

- 1.載入模型進行訓練
- 2.計算訓練集、測試集準確度
- 3.將模型的預測結果存於字典中

程式位置

檔案:predict.py





## predict.py

```
48 # Logistic Regression
49 Logistic = LogisticRegression() # 設定模型
50 Logistic.fit(trainX, trainY) # 模型訓練
51 train_acc = Logistic.score(trainX, trainY) #訓練集準確度計算
52 test_acc = Logistic.score(testX, testY) #測試集準確度計算
53 model_accuracies['LogisticRegression'] = test_acc #將測試集結果儲存到字典中
54 print('LR訓練集準確率 %.2f' % train_acc)
55 print('LR測試集準確率 %.2f' % test_acc)
56
57
58 # Gaussian Naive Bayes
59 Bayesian = GaussianNB() # 設定模型
60 Bayesian.fit(trainX, trainY) # 模型訓練
61 train_acc = Bayesian.score(trainX, trainY) #訓練集準確度計算
62 test_acc = Bayesian.score(testX, testY) #測試集準確度計算
63 model_accuracies['GaussianNB'] = test_acc #將測試集結果儲存到字典中
64 print('Bayes訓練集準確率 %.2f' % train_acc)
65 print('Bayes測試集準確率 %.2f' % test_acc)
```

## 其他模型一樣步驟

- 1.載入模型進行訓練
- 2.計算訓練集、測試集準確度
- 3.將模型的預測結果存於字典中

程式位置  
檔案:predict.py

```
66 # RandomForest
67 RF = RandomForestClassifier()
68 RF.fit(trainX, trainY) # 模型訓練
69 train_acc = RF.score(trainX, trainY) # 訓練集準確度計算
70 test_acc = RF.score(testX, testY) # 測試集準確度計算
71 model_accuracies['RandomForest'] = test_acc # 將測試集結果儲存到字典中
72 print('RF訓練集準確率 %.2f' % train_acc)
73 print('RF測試集準確率 %.2f' % test_acc)
74
75
76 # XGBoost
77 Xgboost = XGBClassifier() # 設定模型
78 Xgboost.fit(trainX, trainY) # 模型訓練
79 train_acc = Xgboost.score(trainX, trainY) # 訓練集準確度計算
80 test_acc = Xgboost.score(testX, testY) # 測試集準確度計算
81 model_accuracies['XGBoost'] = test_acc # 將測試集結果儲存到字典中
82 print('Xgboost訓練集準確率 %.2f' % train_acc)
83 print('Xgboost測試集準確率 %.2f' % test_acc)
84
```

### 其他模型一樣步驟

1. 載入模型進行訓練
2. 計算訓練集、測試集準確度
3. 將模型的預測結果存於字典中



## predict.py

```
84 trainX, testX, trainY, testY = lstm_split_data.split_stock_data(df, label_column, delete_column) #運用lstm
85
86 # 創建LSTM模型
87 model = Sequential()
88 model.add(LSTM(units=100, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))
89 model.add(Dropout(0.3))
90 model.add(LSTM(units=100, return_sequences=False))
91 model.add(Dropout(0.3))
92 model.add(Dense(units=1, activation='relu'))
93
94 model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy']) # 編譯
95 model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2) # 訓練模型
96
97 train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0) # 訓練集準確度計算
98 test_loss, test_acc = model.evaluate(testX, testY, verbose=0) # 測試集準確度計算
99 model accuracies['LSTM'] = test_acc #將測試集結果儲存到字典中
100 print('LSTM訓練集準確率: %.4f' % train_acc)
101 print('LSTM測試集準確率: %.4f' % test_acc)
```

LSTM模型設定

模型訓練

計算訓練集、測試集準確度

將結果存在字典中

程式位置  
檔案:predict.py





## predict.py

```
84 trainX, testX, trainY, testY = lstm_split_data.split_stock_data(df, label_column, delete_column) #運用lstm
85
86 # 創建LSTM模型
87 model = Sequential()
88 model.add(LSTM(units=100, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))
89 model.add(Dropout(0.3))
90 model.add(LSTM(units=100, return_sequences=False))
91 model.add(Dropout(0.3))
92 model.add(Dense(units=1, activation='relu'))
93
94 model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy']) # 編譯
95 model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2) # 訓練模型
96
97 train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0) # 訓練集準確度計算
98 test_loss, test_acc = model.evaluate(testX, testY, verbose=0) # 測試集準確度計算
99 model accuracies['LSTM'] = test_acc #將測試集結果儲存到字典中
100 print('LSTM訓練集準確率: %.4f' % train_acc)
101 print('LSTM測試集準確率: %.4f' % test_acc)
```

LSTM模型設定

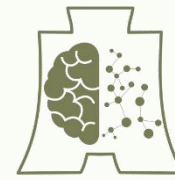
模型訓練

計算訓練集、測試集準確度

將結果存在字典中

程式位置  
檔案:predict.py





```
102 # 創建GRU模型(GRU)
103 model = Sequential()
104 model.add(GRU(units=100, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))
105 model.add(Dropout(0.3))
106 model.add(GRU(units=100, return_sequences=False))
107 model.add(Dropout(0.3))
108 model.add(Dense(units=1, activation='relu'))
109
110 model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy'])
111 model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2)
112
113 train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0)
114 test_loss, test_acc = model.evaluate(testX, testY, verbose=0)
115 model_accuracies['GRU'] = test_acc
116 print('GRU訓練集準確率: %.2f' % train_acc)
117 print('GRU測試集準確率: %.2f' % test_acc)
118
```

gru模型設定

模型訓練

計算訓練集、測試集準確度

將結果存在字典中

```
119 # 創建BI-LSTM模型(Bidirectional)
120
121 model = Sequential()
122 model.add(Bidirectional(LSTM(units=100, return_sequences=True), input_shape=(trainX.shape[1], trainX.shape[2])))
123 model.add(Dropout(0.3))
124 model.add(Bidirectional(LSTM(units=100, return_sequences=False)))
125 model.add(Dropout(0.3))
126 model.add(Dense(units=1, activation='relu'))
127
128 model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy'])
129 model.fit(trainX, trainY, epochs=10, batch_size=32, validation_split=0.2)
130
131 train_loss, train_acc = model.evaluate(trainX, trainY, verbose=0)
132 test_loss, test_acc = model.evaluate(testX, testY, verbose=0)
133 model_accuracies['BI-LSTM'] = test_acc
134 print('BI-LSTM訓練集準確率: %.2f' % train_acc)
135 print('BI-LSTM測試集準確率: %.2f' % test_acc)
136
137
138 best_model = max(model_accuracies, key=model_accuracies.get) # 找出準確率最高的模型
139 best_accuracy = model_accuracies[best_model] # 找出準確率最高的模型的名稱
140 print(f'準確率最高的模型是 {best_model}, 準確率為 %.2f' % best_accuracy)
```

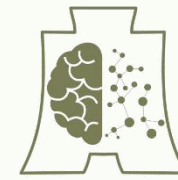
bi-lstm模型設定

模型訓練

將結果存在字典中

計算訓練集、測試集準確度

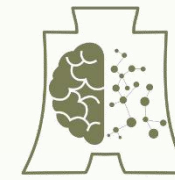
找出準確度最高的模型



# 程式撰寫 - LSTM資料分割

- 特徵整理(股價資料<開、高、低、收、成交量>、  
技術指標<MA\_20、RSI\_14、MACD>  
前N期<5、10、15、20>資料<收盤價、MA\_20、RSI\_14、MACD>)  
(共24個)





## lstm\_split\_data.py完整程式碼

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('min_max_normalized_data.csv', index_col='Date') # 讀取整理好的資料

def split_stock_data(stock_data, label_column, delete_column, time_step=1, test_size=0.3, random_state=42):
    X, y = [], [] # 創建兩個空列表
    scaled_data = stock_data.drop([label_column, delete_column], axis=1).values # 特徵集整理，刪除非特徵的欄位
    labels = stock_data[label_column].values # 標籤集[LABEL]

    for i in range(time_step, len(scaled_data)): # 運用迴圈進行資料處理，根據 time_step 設定的步數
        X.append(scaled_data[i-time_step:i]) # 將過去 time_step 天的特徵數據添加到 X 中
        y.append(labels[i]) # 將當前時間步的標籤數據添加到 y 中

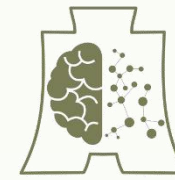
    X = np.array(X) # 轉換資料型態為numpy格式
    y = np.array(y) # 轉換資料型態為numpy格式

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state) # 資料分割成訓練集跟測試集
    return X_train, X_test, y_train, y_test
```

程式位置

檔案:lstm\_split\_data.py





## lstm\_split\_data.py

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
```

載入套件

創建空  
列表

```
4
5 df = pd.read_csv('min_max_normalized_data.csv', index_col='Date') # 讀取整理好的資料
```

```
6
7 def split_stock_data(stock_data, label_column, delete_column, time_step=1, test_size=0.3, random_state=42):
```

```
8     X, y = [], [] # 創建兩個空列表
```

```
9     scaled_data = stock_data.drop([label_column, delete_column], axis=1).values # 特徵集
```

```
10    labels = stock_data[label_column].values # 標籤集[LABEL]
```

建立特徵跟標籤資料

```
11
12     for i in range(time_step, len(scaled_data)): # 運用迴圈
```

```
13         X.append(scaled_data[i-time_step:i]) # 將過去time_step個特徵
```

```
14         y.append(labels[i]) # 將當前時間步的標籤數據添加到y
```

運用迴圈根據time\_step進行資料處理

```
15
16
17     X = np.array(X) # 轉換資料型態為numpy
```

```
18     y = np.array(y) # 轉換資料型態為numpy
```

轉換資料型態為numpy

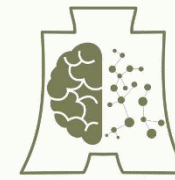
```
19
20     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
```

```
21     return X_train, X_test, y_train, y_test
22
```

資料分割訓練集、測試集

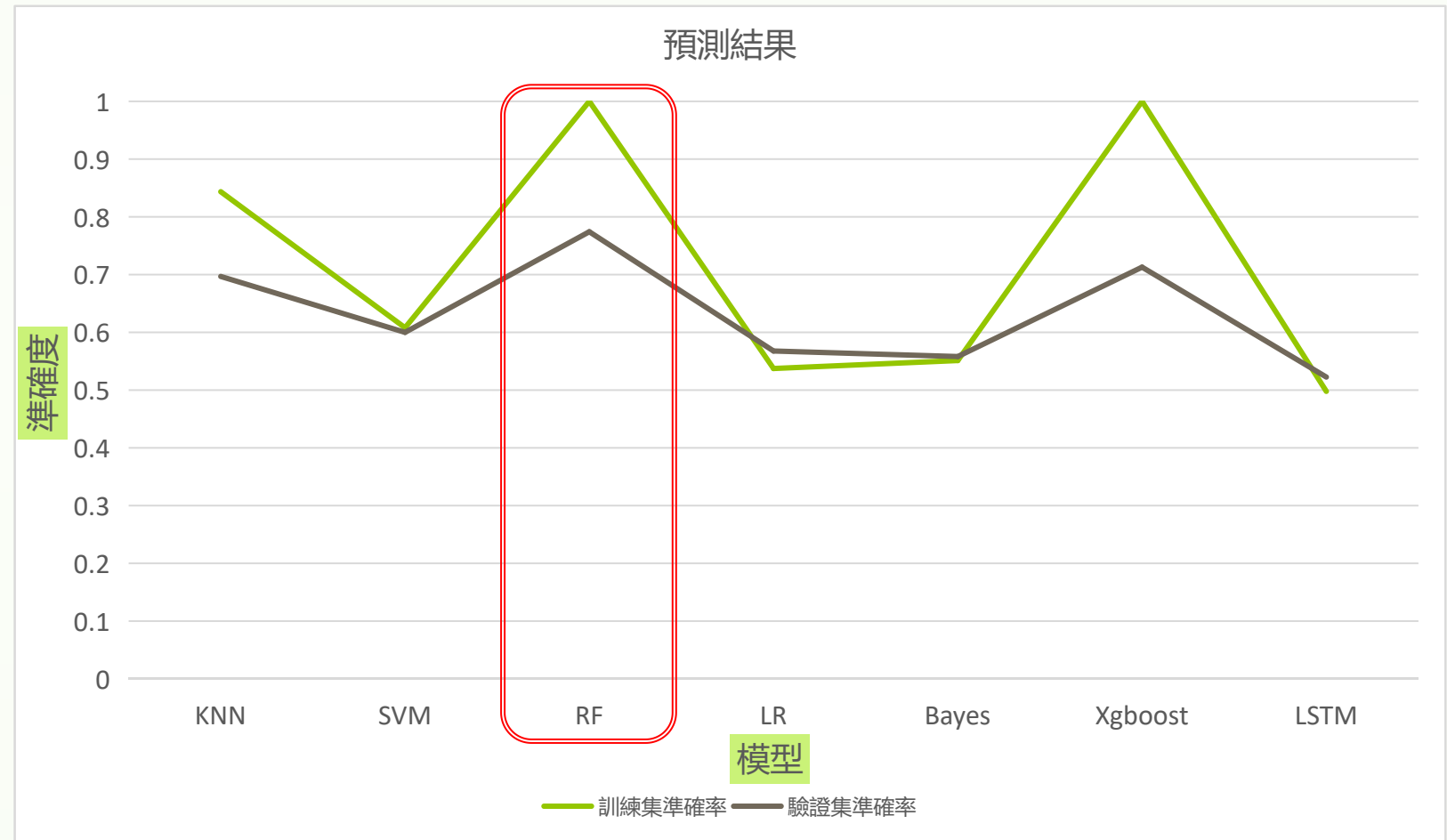
程式位置

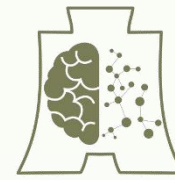
檔案:lstm\_split\_data.py



## 預測結果

KNN訓練集準確率 0.8435  
 KNN測試集準確率 0.6968  
 SVM訓練集準確率 0.6080  
 SVM測試集準確率 0.6000  
**RF訓練集準確率 1.0000**  
**RF測試集準確率 0.7742**  
 LR訓練集準確率 0.5374  
 LR測試集準確率 0.5677  
 Bayes訓練集準確率 0.5512  
 Bayes測試集準確率 0.5581  
 Xgboost訓練集準確率 1.0000  
 Xgboost測試集準確率 0.7129  
 LSTM訓練集準確率: 0.5714  
 LSTM測試集準確率: 0.5161  
 GRU訓練集準確率: 0.57  
 GRU測試集準確率: 0.54  
 BI-LSTM訓練集準確率: 0.59  
 BI-LSTM測試集準確率: 0.52





## 預測結果-0050成分股前五名

商品代碼	商品名稱	商品數量	商品權重
2330	台積電	227214109	55.09
2317	鴻海	107712968	5.42
2454	聯發科	13988559	4.44
2308	台達電	20406682	2.01
2382	廣達	25368186	1.93

股票代號	測試集準確度
2330	準確率最高的模型是 <b>RandomForest</b> ，準確率為 <b>0.7742</b>
2317	準確率最高的模型是 <b>RandomForest</b> ，準確率為 <b>0.7806</b>
2454	準確率最高的模型是 <b>XGBoost</b> ，準確率為 <b>0.7935</b>
2308	準確率最高的模型是 <b>RandomForest</b> ，準確率為 <b>0.7548</b>
2382	準確率最高的模型是 <b>RandomForest</b> ，準確率為 <b>0.8355</b>
0050	準確率最高的模型是 <b>RandomForest</b> ，準確率為 <b>0.8032</b>

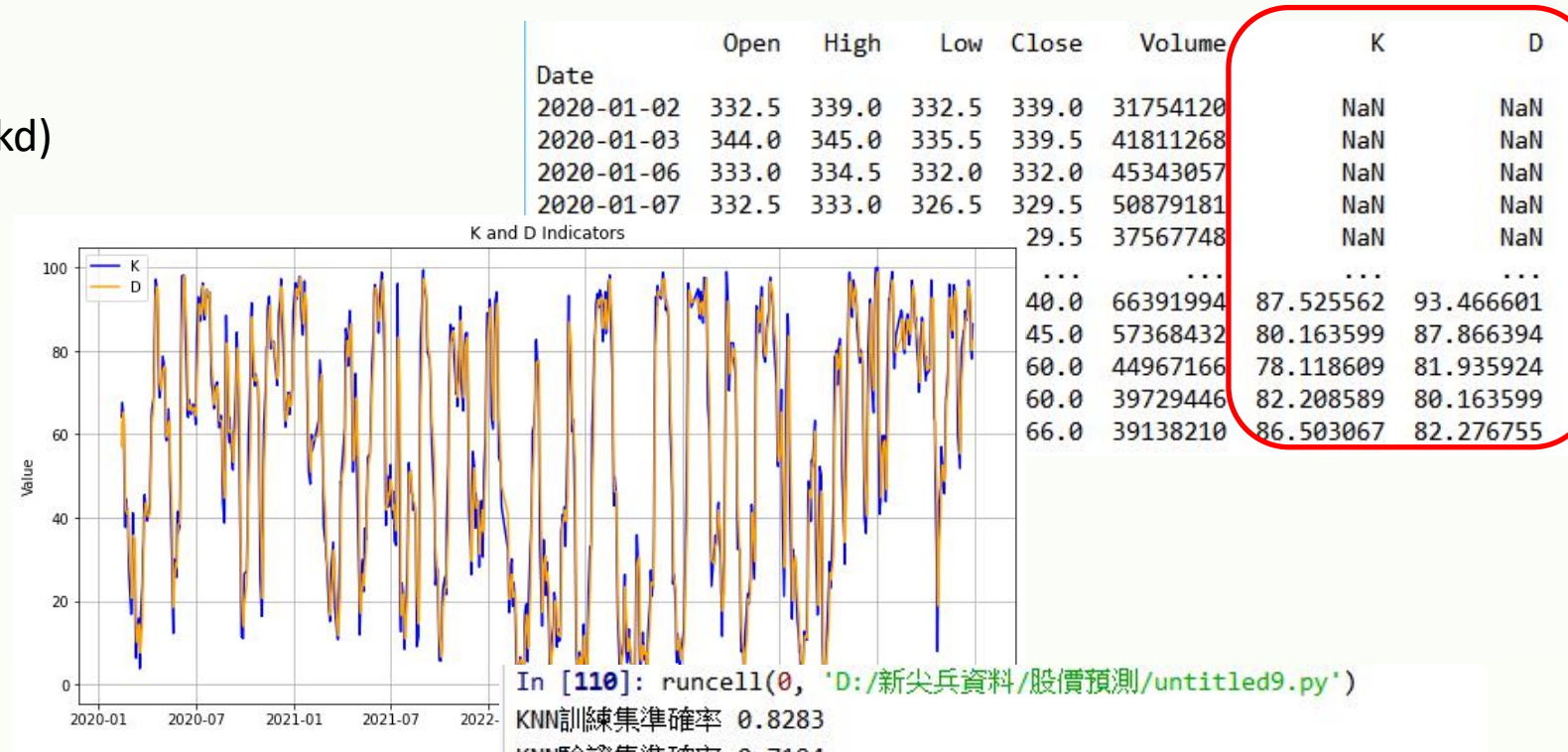


## 隨堂練習

1. 運用malib計算其他技術指標(如kd)

2. 將計算完的技術指標畫圖呈現

3. 加入其他技術指標進行預測，觀察預測準確率上升/下降



```
In [110]: runcell(0, 'D:/新尖兵資料/股價預測/untitled9.py')
```

```
KNN訓練集準確率 0.8283
KNN驗證集準確率 0.7194
SVM訓練集準確率 0.6524
SVM驗證集準確率 0.6161
RF訓練集準確率 1.0000
RF驗證集準確率 0.8419
LR訓練集準確率 0.5789
LR驗證集準確率 0.5484
Bayes訓練集準確率 0.5471
Bayes驗證集準確率 0.5129
Xgboost訓練集準確率 1.0000
Xgboost驗證集準確率 0.7968
```

準確率最高的模型是 RandomForest，準確率為 0.8419





**Thank you.**

