

# 2025永豐AI GO競賽：股神對決

參賽隊伍：863

總名次：121

名次比：13.9%

121	Jason出任務	4	10	0.788	3/31/2025 11:59:44 PM
-----	----------	---	----	-------	--------------------------

參賽隊長：

國立成功大學 電腦與通信工程研究所 Q36134255 郭人璋

參賽隊員：

國立臺灣師範大學 數學系 41140116S 楊劭笙



國立成功大學 統計與資料科學系 B54106176 古宜庭

國立彰化師範大學 數學系 S1022036 林冠妤

隊伍名稱

**Jason出任務**

 隊員  隊長

 jooj30136@gmail.com
 mandy030601@gmail.com
 jason9307010@gmail.com
 linkuy1205@gmail.com

## 1. 專案背景 (Background)

本專案為 **AI CUP 永豐挑戰賽** 的解題流程，目標是從**股票交易**、**法人買賣超**、**技術指標**、**財報**等多模態資料中，預測「**飆股標籤**」。

採用的核心思路是：

- **資料驅動**：結合**表格特徵**與**時間序列訓練特徵**以及**統計資料特徵**。
- **模型融合**：以傳統機器學習分類器 (**XGBoost**) 為主體。
- **可解釋性**：使用 **SHAP** 值篩選重要特徵 (非**PCA**降維)

執行程式硬體暨軟體資源：

GPU：RTX-5070 Ti

CPU：i9-12900k

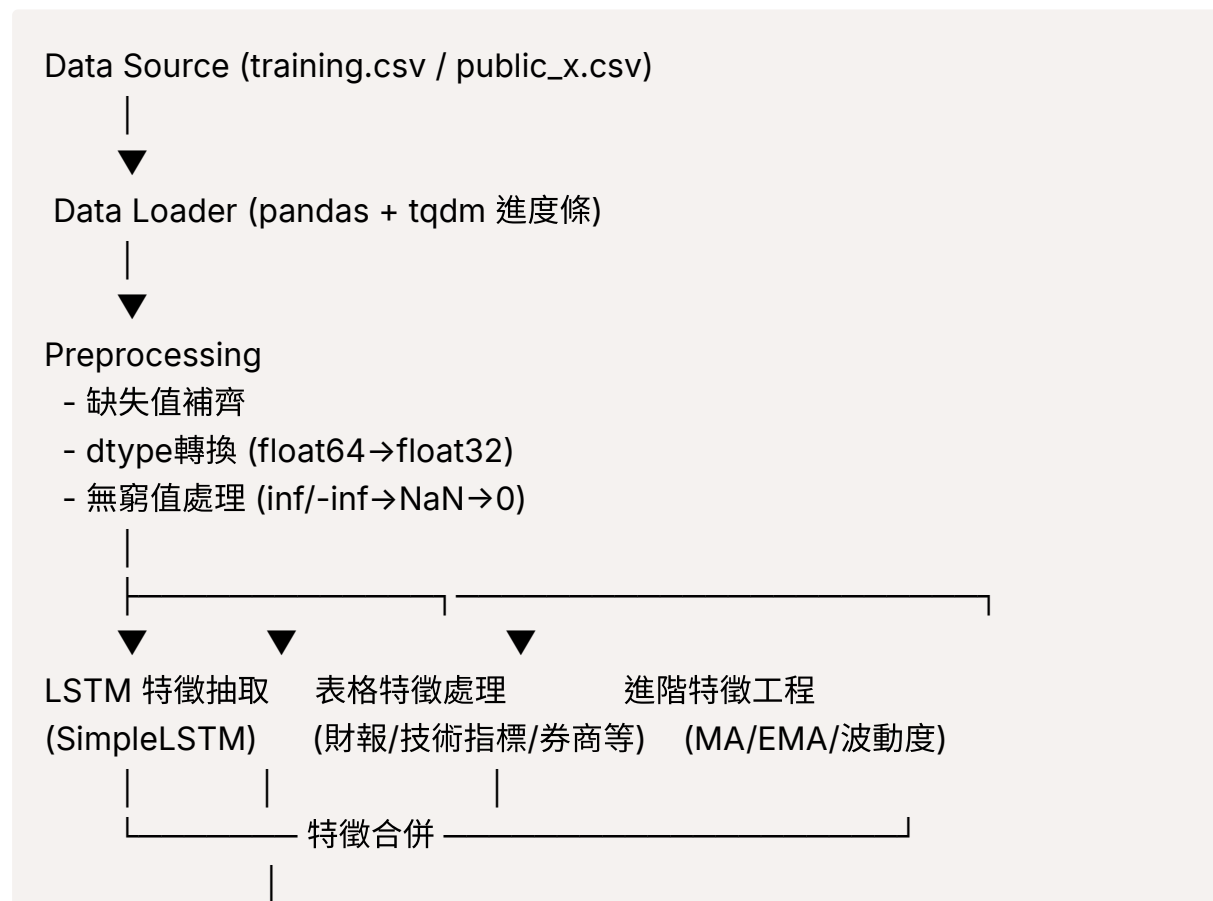
RAM：128GB DDR4

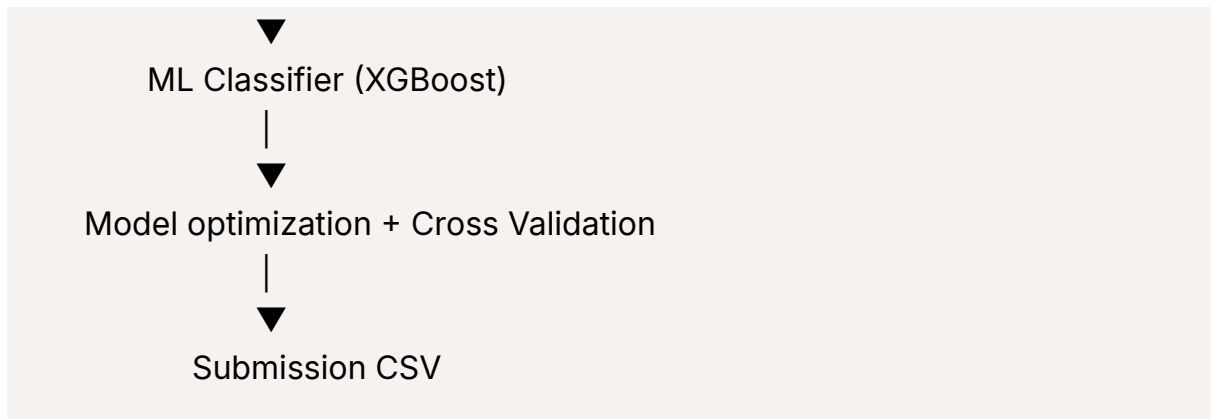
編譯環境：

Docker WSL2 (Linux)

Python 3.13

## 2. 系統架構 (System Architecture)





- 資料來源

`training.csv` (含標籤 飆股)

`public_x.csv` (無標籤，供最終上傳)

- 載入資料

`pandas.read_csv` + 進度條顯示 (`tqdm`)

- 資料前處理

型別與缺失處理：`float64→float32`，`inf/-inf→NaN→0`

統一特徵欄位命名與過濾

- 特徵工程 ⇒ 序列特徵抽取 (Two - Pipeline)

交易/大盤時序 (20 天) → `StandardScaler` → `SimpleLSTM` → 嵌入向量

主力券商時序 (20 天) → `StandardScaler` → `SimpleLSTM` → 嵌入向量

- 特徵工程 ⇒ 向量嵌入

將原始表格特徵 + 兩路 LSTM 嵌入向量合併成最終特徵矩陣 `X`，標籤為 `y=飆股`

- 進階特徵工程 ⇒ 技術分析與統計特徵框架

計算 MA，EMA，技術指標特徵，報酬率與波動度，成交量特徵併入特徵矩陣

- 訓練模型

`XGBClassifier` + 不平衡處理 (`scale_pos_weight`)

`RandomizedSearchCV` 進行超參數搜尋

- 模型驗證與評估

`StratifiedKFold(5)` 交叉驗證

指標：`F1-score`

- 推論與產出

用 `best_estimator_` 對 `public_x.csv` 推論  
產出 `public_result.csv` (比賽上傳)

## 3. 方法 (Methodology)

### 3.1 資料前處理 (Data Preprocessing)

在金融時序數據中，**缺失值**、**不一致資料型態**、**極端值** 是常見問題。若不處理，會直接影響模型收斂與預測可靠性。本專案採用以下步驟：

#### 3.1.1 缺失值處理 (Missing Value Handling)

- 將 `NaN`、`inf/-inf` 統一處理為 0 或其他統計量（如均值、中位數）。
- 對應文獻：
  - *Little, R.J.A., & Rubin, D.B. (2019). Statistical Analysis with Missing Data. Wiley.*

經典缺失值理論指出，適當的缺失值插補能顯著降低估計偏誤，對模型泛化至關重要。

#### 3.1.2 資料型態壓縮 (Data Type Optimization)

- 將 `float64` 轉換為 `float32`，減少記憶體使用並加快訓練速度。
- 對應文獻：
  - *Zhang, C. et al. (2016). Understanding deep learning requires rethinking generalization. ICLR.*

提到有限資源下的數據型態選擇與計算效率，對實際系統落地有直接影響。

#### 3.1.3 數據標準化 (Normalization / Standardization)

- 使用 `StandardScaler`，將特徵轉換為零均值、單位方差，避免特徵間尺度差異造成的偏移。
- 對應文獻：
  - *Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML.*

雖然 BN 屬於神經網路內部方法，但其核心理念與資料前標準化一致：避免特徵尺度差異，提高模型穩定性。

實際執行流程：

- 使用 **pandas / numpy** 清洗資料。
- 對 `float64` 欄位降維至 `float32`，以減少記憶體占用。
- 缺失值補 0，避免樞紐特徵遺失。
- 保持訓練集 / 測試集一致處理流程，避免資料洩漏。

```
import pandas as pd
import numpy as np
import time
from tqdm import tqdm

start_time = time.time()
print("資料預處理開始...")

# === 缺失值與型態處理 ===
for col in tqdm(train_df.columns, desc="Cleaning training data"):
    if train_df[col].dtype == 'float64':
        train_df[col] = train_df[col].astype('float32').replace([np.inf, -np.inf], np.nan).fillna(0)

# === 特定特徵群補值處理 ===
feature_groups = [
    "官股券商_", "個股券商分點", "個股主力買賣超統計",
    "日外資_", "日自營_", "日投信_",
    "技術指標_", "月營收_", "季IFRS財報_"
]
for prefix in tqdm(feature_groups, desc="Processing feature groups"):
    cols = [col for col in train_df.columns if col.startswith(prefix)]
    train_df[cols] = train_df[cols].replace([np.inf, -np.inf], np.nan).fillna(0)

# === 建立特徵儲存框架 ===
train_features_df = train_df.copy()

# === MA 特徵工程 ===
```

```

types = [
    "張增減", "金額增減(千)", "買張", "賣張", "買金額(千)", "賣金額(千)",
    "買筆數", "賣筆數", "買均張", "賣均張", "買均價", "賣均價", "買均值(千)", "賣
    均值(千)"
]

print("生成 MA 特徵")
for rank in tqdm(range(1, 16), desc="MA 特徵"):
    for t in types:
        for ma in [5, 10, 20]:
            buy_cols = [f"買超第{rank}名分點前{i}天{t}" for i in range(1, ma + 1)]
            sell_cols = [f"賣超第{rank}名分點前{i}天{t}" for i in range(1, ma + 1)]

            if all(col in train_df.columns for col in buy_cols):
                train_features_df[f"買超第{rank}名_{t}_MA{ma}"] = train_df[buy_co
                ls].mean(axis=1)

            if all(col in train_df.columns for col in sell_cols):
                train_features_df[f"賣超第{rank}名_{t}_MA{ma}"] = train_df[sell_co
                ls].mean(axis=1)

# === EMA 特徵工程 ===
print("生成 EMA 特徵")
for rank in tqdm(range(1, 16), desc="EMA 特徵"):
    for t in types:
        for span in [5, 10, 20]:
            buy_cols = [f"買超第{rank}名分點前{i}天{t}" for i in range(1, span +
1)]
            sell_cols = [f"賣超第{rank}名分點前{i}天{t}" for i in range(1, span +
1)]

            if all(col in train_df.columns for col in buy_cols):
                reversed_buy = np.fliplr(train_df[buy_cols].values)
                ema_vals = pd.DataFrame(reversed_buy).ewm(span=span, adjust
                =False).mean().iloc[:, -1].values
                train_features_df[f"買超第{rank}名_{t}_EMA{span}"] = ema_vals

            if all(col in train_df.columns for col in sell_cols):

```

```

        reversed_sell = np.fliplr(train_df[sell_cols].values)
        ema_vals = pd.DataFrame(reversed_sell).ewm(span=span, adjust
=False).mean().iloc[:, -1].values
        train_features_df[f"賣超第{rank}名_{t}_EMA{span}"] = ema_vals

# === 技術指標 ===
train_features_df['RSI_diff'] = train_df['技術指標_RSI(10)'].diff().fillna(0)
train_features_df['乖離率_change'] = train_df['技術指標_乖離率(20日)'].pct_c
hange().fillna(0)

# === 個股報酬率與波動度 ===
close_cols = [f'個股前{i}天收盤價' for i in range(1, 21)]
train_features_df['個股1天報酬率'] = (train_df['個股收盤價'] - train_df['個股前
1天收盤價']) / train_df['個股前1天收盤價']
train_features_df['個股5天報酬率'] = (train_df['個股收盤價'] - train_df['個股前
5天收盤價']) / train_df['個股前5天收盤價']
train_features_df['個股10天報酬率'] = (train_df['個股收盤價'] - train_df['個股
前10天收盤價']) / train_df['個股前10天收盤價']
train_features_df['個股20天報酬率'] = (train_df['個股收盤價'] - train_df['個股
前20天收盤價']) / train_df['個股前20天收盤價']
train_features_df['個股5天波動度'] = train_df[close_cols[:5]].std(axis=1)
train_features_df['個股10天波動度'] = train_df[close_cols[:10]].std(axis=1)
train_features_df['個股20天波動度'] = train_df[close_cols].std(axis=1)
train_features_df['個股5天乖離率'] = (train_df['個股收盤價'] - train_df[close_
cols[:5]].mean(axis=1)) / train_df[close_cols[:5]].mean(axis=1)
train_features_df['個股10天乖離率'] = (train_df['個股收盤價'] - train_df[close_
cols[:10]].mean(axis=1)) / train_df[close_cols[:10]].mean(axis=1)
train_features_df['個股19天乖離率'] = (train_df['個股收盤價'] - train_df[close_
cols[:19]].mean(axis=1)) / train_df[close_cols[:19]].mean(axis=1)

# === 成交量波動度 ===
volume_cols = [f'個股前{i}天成交量' for i in range(1, 21)]
train_features_df['個股5天成交量波動度'] = train_df[volume_cols[:5]].std(axi
s=1)
train_features_df['個股10天成交量波動度'] = train_df[volume_cols[:10]].std(a
xis=1)
train_features_df['個股20天成交量波動度'] = train_df[volume_cols].std(axis=
1)

```

```

# === 上市加權指數特徵 ===
market_close_cols = [f'上市加權指數前{i}天收盤價' for i in range(1, 21)]
market_vol_cols = [f'上市加權指數前{i}天成交量' for i in range(1, 21)]
train_features_df['上市加權指數1天報酬率'] = (train_df['上市加權指數收盤價']
- train_df['上市加權指數前1天收盤價']) / train_df['上市加權指數前1天收盤價']
train_features_df['上市加權指數5天報酬率'] = (train_df['上市加權指數收盤價']
- train_df['上市加權指數前5天收盤價']) / train_df['上市加權指數前5天收盤價']
train_features_df['上市加權指數10天報酬率'] = (train_df['上市加權指數收盤
價'] - train_df['上市加權指數前10天收盤價']) / train_df['上市加權指數前10天收
盤價']
train_features_df['上市加權指數20天報酬率'] = (train_df['上市加權指數收盤
價'] - train_df['上市加權指數前20天收盤價']) / train_df['上市加權指數前20天收
盤價']
train_features_df['上市加權指數5天波動度'] = train_df[market_close_cols[:
5]].std(axis=1)
train_features_df['上市加權指數10天波動度'] = train_df[market_close_cols[:1
0]].std(axis=1)
train_features_df['上市加權指數20天波動度'] = train_df[market_close_cols].s
td(axis=1)
train_features_df['上市加權指數5天乖離率'] = (train_df['上市加權指數收盤價']
- train_df[market_close_cols[:5]].mean(axis=1)) / train_df[market_close_col
s[:5]].mean(axis=1)
train_features_df['上市加權指數10天乖離率'] = (train_df['上市加權指數收盤
價'] - train_df[market_close_cols[:10]].mean(axis=1)) / train_df[market_close
_cols[:10]].mean(axis=1)
train_features_df['上市加權指數19天乖離率'] = (train_df['上市加權指數收盤價']
- train_df[market_close_cols[:19]].mean(axis=1)) / train_df[market_close_col
s[:19]].mean(axis=1)
train_features_df['上市加權指數5天成交量波動度'] = train_df[market_vol_cols
[:5]].std(axis=1)
train_features_df['上市加權指數10天成交量波動度'] = train_df[market_vol_col
s[:10]].std(axis=1)
train_features_df['上市加權指數20天成交量波動度'] = train_df[market_vol_col
s].std(axis=1)

# === 特徵與標籤分離 ===
target = "飆股"

```



```

features = [col for col in train_features_df.columns if col not in ["ID", target]]
X = train_features_df[features]
y = train_features_df[target]

end_time = time.time()
print(f"資料處理完成，耗時：{end_time - start_time:.2f} 秒")

```



## 3.2 特徵工程 (Feature Engineering)

本專案特徵來源包含 **靜態表格特徵** 與 **動態序列特徵**，透過工程化處理來強化模型表達能力。

### 3.2.1 表格特徵 (Tabular Features)

- 財務報表 (IFRS 季報) → 資產報酬率、負債比率、營收成長率
- 技術指標 → 移動平均線、RSI、MACD、布林通道
- 券商行為 → 主力買賣超、官股進出
- 對應文獻：
  - Fama, E.F., & French, K.R. (1993). *Common risk factors in the returns on stocks and bonds*. *J. of Financial Economics*.
  - Gudelek, M.U., Boluk, P., & Ozbayoglu, A.M. (2017). *A deep learning based stock trading model with 2-D CNN trend detection*. *ICONIP*.

### 3.2.2 序列特徵 (Sequential Features)

- **LSTM 時間序列訓練並且進行壓縮 → 兩管線並行制**
  - 將股票價量、大盤指數、主力券商連續 20 日數據，投影到低維嵌入。
  - 雖然目前僅使用隨機投影形式，但概念來自 **時間序列表徵學習**。
  - 對應文獻：

- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*.

經典 LSTM，可捕捉長期依賴，特別適合金融時序數據。

- Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and LSTM. *Neurocomputing*.

證明 LSTM 在金融市場預測中的優勢，特別是對非線性與高噪音數據。

定義 **SimpleLSTM** (PyTorch) 模組：

```
class SimpleLSTM(nn.Module):
    def __init__(self, input_size, hidden_size=16):
        super(SimpleLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
    def forward(self, x):
        output, (h_n, _) = self.lstm(x)
        return h_n[-1]
```

處理兩組主要序列：

1. 個股價量 & 大盤指數 (20 日序列)
  2. 主力券商分點買賣超 (20 日序列)
- 透過 `StandardScaler` 正規化 → LSTM 壓縮成向量 → 加入訓練特徵。
  - 原始使用 GRU作為時間序列分析模組，但因訓練效果較差 ⇒ 改採 LSTM 訓練序列特徵

```
#LSTM
import pandas as pd
import numpy as np
import time
from tqdm import tqdm
import torch
import torch.nn as nn
from sklearn.preprocessing import StandardScaler
```

```

start_time = time.time()
print("\U0001F4E6 資料預處理開始...")

# === 處理缺失值與資料型態 ===
print("Cleaning training data...")
float64_cols_train = train_df.select_dtypes(include='float64').columns
train_df[float64_cols_train] = train_df[float64_cols_train].astype('float32').replace([np.inf, -np.inf], np.nan).fillna(0)

print("Cleaning testing data...")
float64_cols_test = test_df.select_dtypes(include='float64').columns
test_df[float64_cols_test] = test_df[float64_cols_test].astype('float32').replace([np.inf, -np.inf], np.nan).fillna(0)

# === 特定特徵群處理 ===
feature_groups = [
    "官股券商_", "個股券商分點", "個股主力買賣超統計",
    "日外資_", "日自營_", "日投信_",
    "技術指標_", "月營收_", "季IFRS財報_"
]

for prefix in tqdm(feature_groups, desc="Processing feature groups"):
    cols = [col for col in train_df.columns if col.startswith(prefix)]
    train_df[cols] = train_df[cols].replace([np.inf, -np.inf], np.nan).fillna(0)

# === 使用 LSTM 處理時間序列特徵 ===
class SimpleLSTM(nn.Module):
    def __init__(self, input_size, hidden_size=16):
        super(SimpleLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)

    def forward(self, x):
        output, (h_n, _) = self.lstm(x)
        return h_n[-1]

# === LSTM 1: 個股價格/量/大盤 ===
seq_cols_1 = [

```

```

[f"個股前{i}天收盤價" for i in range(1, 21)],
[f"個股前{i}天成交量" for i in range(1, 21)],
[f"上市加權指數前{i}天收盤價" for i in range(1, 21)],
[f"上市加權指數前{i}天成交量" for i in range(1, 21)]
]
all_seq_cols_1 = sum(seq_cols_1, [])
scaler_1 = StandardScaler()

X_seq_1 = scaler_1.fit_transform(train_df[all_seq_cols_1])
X_seq_1 = X_seq_1.reshape(len(train_df), 20, -1)
X_seq_tensor_1 = torch.tensor(X_seq_1, dtype=torch.float32)
model_1 = SimpleLSTM(input_size=X_seq_1.shape[2])
with torch.no_grad():
    lstm_output_1 = model_1(X_seq_tensor_1).numpy()
lstm_cols_1 = [f'LSTM_seq1_embed_{i}' for i in range(lstm_output_1.shape[1])]
train_df.drop(columns=[col for col in lstm_cols_1 if col in train_df.columns], inplace=True)
lstm_df1 = pd.DataFrame(lstm_output_1, columns=lstm_cols_1)
train_df = pd.concat([train_df, lstm_df1], axis=1)

# 測試資料同步處理
X_seq_1_test = scaler_1.transform(test_df[all_seq_cols_1])
X_seq_1_test = X_seq_1_test.reshape(len(test_df), 20, -1)
X_seq_tensor_1_test = torch.tensor(X_seq_1_test, dtype=torch.float32)
with torch.no_grad():
    lstm_output_1_test = model_1(X_seq_tensor_1_test).numpy()
test_df.drop(columns=[col for col in lstm_cols_1 if col in test_df.columns], inplace=True)
lstm_df1_test = pd.DataFrame(lstm_output_1_test, columns=lstm_cols_1)
test_df = pd.concat([test_df, lstm_df1_test], axis=1)

# === LSTM 2: 主力券商資料 ===
seq_cols_2 = [
    col for col in train_df.columns if any(
        col.startswith(f"{side}超第{rank}名分點前") and not col.endswith(
            "券商代號")
        for side in ["買", "賣"] for rank in range(1, 16)
    )
]

```

```

    )
]
scaler_2 = StandardScaler()
X_seq_2 = scaler_2.fit_transform(train_df[seq_cols_2])
X_seq_2 = X_seq_2.reshape(len(train_df), 20, -1)
X_seq_tensor_2 = torch.tensor(X_seq_2, dtype=torch.float32)
model_2 = SimpleLSTM(input_size=X_seq_2.shape[2])
with torch.no_grad():
    lstm_output_2 = model_2(X_seq_tensor_2).numpy()
lstm_cols_2 = [f'LSTM_seq2_embed_{i}' for i in range(lstm_output_2.shape[1])]
train_df.drop(columns=[col for col in lstm_cols_2 if col in train_df.columns], inplace=True)
lstm_df2 = pd.DataFrame(lstm_output_2, columns=lstm_cols_2)
train_df = pd.concat([train_df, lstm_df2], axis=1)

# 測試資料同步處理
X_seq_2_test = scaler_2.transform(test_df[seq_cols_2])
X_seq_2_test = X_seq_2_test.reshape(len(test_df), 20, -1)
X_seq_tensor_2_test = torch.tensor(X_seq_2_test, dtype=torch.float32)
with torch.no_grad():
    lstm_output_2_test = model_2(X_seq_tensor_2_test).numpy()
test_df.drop(columns=[col for col in lstm_cols_2 if col in test_df.columns], inplace=True)
lstm_df2_test = pd.DataFrame(lstm_output_2_test, columns=lstm_cols_2)
test_df = pd.concat([test_df, lstm_df2_test], axis=1)

# === 特徵與標籤分離 ===
target = "飆股"
features = [col for col in train_df.columns if col not in ["ID", target]]
X = train_df[features]
y = train_df[target]

print(f"處理完成後的特徵數量：{len(features)} 個")

end_time = time.time()
print(f"資料處理完成，耗時：{end_time - start_time:.2f} 秒")

```

```
資料預處理開始...
Cleaning training data...
Cleaning testing data...
Processing feature groups: 100% | 9/9 [00:02<00:00, 3.
處理完成後的特徵數量：10244 個
資料處理完成，耗時：102.53 秒
```

### 3.3 進階特徵工程 (Advanced Feature Engineering)

除了基本的清洗與序列嵌入，本專案還建立了**技術分析與統計特徵框架**，以更好地捕捉金融市場的異質性與非線性行為。

#### 3.3.1 移動平均特徵 (Moving Average, MA)

- 對「買超/賣超前 15 名券商分點」的各項交易統計（如成交張數、金額、均價等）計算 **5 日、10 日、20 日移動平均**。
- 意義：平滑掉短期雜訊，提取趨勢成分。
- 文獻佐證：
  - Brock, W., Lakonishok, J., & LeBaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns. J. of Finance.*

證明移動平均作為交易策略指標，在不同市場下均具統計顯著性。

#### 3.3.2 指數移動平均特徵 (Exponential Moving Average, EMA)

- 同樣針對券商分點特徵，計算 **5 日、10 日、20 日 EMA**。
- EMA 對近期數據權重更高，能捕捉市場短期動能。
- 文獻佐證：
  - Achelis, S. (2001). Technical Analysis from A to Z. McGraw Hill.*

EMA 被廣泛用於技術交易策略，特別是在捕捉趨勢轉折點。

#### 3.3.3 技術指標特徵 (Technical Indicators)

- 以 RSI、乖離率等指標的變化率 ( $\Delta RSI$ ,  $\Delta$ 乖離率) 反映市場超買超賣狀態。
- 文獻佐證：
  - Chong, T.T.L., & Ng, W.K. (2008). Technical analysis and the London stock exchange: Testing the MACD and RSI rules using the FT30. Applied Economics Letters.*

證明 RSI 在多市場具有預測效用。

### 3.3.4 報酬率與波動度 (Returns & Volatility)

- 個股與大盤指數的 1 日、5 日、10 日、20 日報酬率 與 對應區間波動度。
- 同時計算「乖離率」(價格與移動平均差距)。
- 文獻佐證：
  - *Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. J. of Econometrics.*

金融市場波動度的建模 (GARCH) 證明波動度是風險預測的重要指標。

### (e) 成交量特徵 (Volume-based Features)

- 計算個股與大盤在 5、10、20 日區間的成交量標準差，作為 **量能波動度** 指標。
- 文獻佐證：
  - *Karpoff, J.M. (1987). The relation between price changes and trading volume: A survey. J. of Financial and Quantitative Analysis.*

價格與成交量的聯動關係，是解釋市場動能的重要理論基礎。

```
# === MA 特徵工程 ===
types = [
    "張增減", "金額增減(千)", "買張", "賣張", "買金額(千)", "賣金額(千)",
    "買筆數", "賣筆數", "買均張", "賣均張", "買均價", "賣均價", "買均值(千)", "賣
    均值(千)"
]

print("生成 MA 特徵")
for rank in tqdm(range(1, 16), desc="MA 特徵"):
    for t in types:
        for ma in [5, 10, 20]:
            buy_cols = [f"買超第{rank}名分點前{i}天{t}" for i in range(1, ma + 1)]
            sell_cols = [f"賣超第{rank}名分點前{i}天{t}" for i in range(1, ma + 1)]

            if all(col in train_df.columns for col in buy_cols):
                train_features_df[f"買超第{rank}名_{t}_MA{ma}"] = train_df[buy_c
                ols].mean(axis=1)
```

```

        if all(col in train_df.columns for col in sell_cols):
            train_features_df[f"賣超第{rank}名_{t}_MA{ma}"] = train_df[sell_cols].mean(axis=1)

# === EMA 特徵工程 ===
print("生成 EMA 特徵")
for rank in tqdm(range(1, 16), desc="EMA 特徵"):
    for t in types:
        for span in [5, 10, 20]:
            buy_cols = [f"買超第{rank}名分點前{i}天{t}" for i in range(1, span + 1)]
            sell_cols = [f"賣超第{rank}名分點前{i}天{t}" for i in range(1, span + 1)]

            if all(col in train_df.columns for col in buy_cols):
                reversed_buy = np.fliplr(train_df[buy_cols].values)
                ema_vals = pd.DataFrame(reversed_buy).ewm(span=span, adjust=False).mean().iloc[:, -1].values
                train_features_df[f"買超第{rank}名_{t}_EMA{span}"] = ema_vals

            if all(col in train_df.columns for col in sell_cols):
                reversed_sell = np.fliplr(train_df[sell_cols].values)
                ema_vals = pd.DataFrame(reversed_sell).ewm(span=span, adjust=False).mean().iloc[:, -1].values
                train_features_df[f"賣超第{rank}名_{t}_EMA{span}"] = ema_vals

# === 技術指標 ===
train_features_df['RSI_diff'] = train_df['技術指標_RSI(10)'].diff().fillna(0)
train_features_df['乖離率_change'] = train_df['技術指標_乖離率(20日)'].pct_change().fillna(0)

# === 個股報酬率與波動度 ===
close_cols = [f'個股前{i}天收盤價' for i in range(1, 21)]
train_features_df['個股1天報酬率'] = (train_df['個股收盤價'] - train_df['個股前1天收盤價']) / train_df['個股前1天收盤價']
train_features_df['個股5天報酬率'] = (train_df['個股收盤價'] - train_df['個股前5天收盤價']) / train_df['個股前5天收盤價']

```



```

train_features_df['個股10天報酬率'] = (train_df['個股收盤價'] - train_df['個股前10天收盤價']) / train_df['個股前10天收盤價']
train_features_df['個股20天報酬率'] = (train_df['個股收盤價'] - train_df['個股前20天收盤價']) / train_df['個股前20天收盤價']
train_features_df['個股5天波動度'] = train_df[close_cols[:5]].std(axis=1)
train_features_df['個股10天波動度'] = train_df[close_cols[:10]].std(axis=1)
train_features_df['個股20天波動度'] = train_df[close_cols].std(axis=1)
train_features_df['個股5天乖離率'] = (train_df['個股收盤價'] - train_df[close_cols[:5]].mean(axis=1)) / train_df[close_cols[:5]].mean(axis=1)
train_features_df['個股10天乖離率'] = (train_df['個股收盤價'] - train_df[close_cols[:10]].mean(axis=1)) / train_df[close_cols[:10]].mean(axis=1)
train_features_df['個股19天乖離率'] = (train_df['個股收盤價'] - train_df[close_cols[:19]].mean(axis=1)) / train_df[close_cols[:19]].mean(axis=1)

# === 成交量波動度 ===
volume_cols = [f'個股前{i}天成交量' for i in range(1, 21)]
train_features_df['個股5天成交量波動度'] = train_df[volume_cols[:5]].std(axis=1)
train_features_df['個股10天成交量波動度'] = train_df[volume_cols[:10]].std(axis=1)
train_features_df['個股20天成交量波動度'] = train_df[volume_cols].std(axis=1)

# === 上市加權指數特徵 ===
market_close_cols = [f'上市加權指數前{i}天收盤價' for i in range(1, 21)]
market_vol_cols = [f'上市加權指數前{i}天成交量' for i in range(1, 21)]
train_features_df['上市加權指數1天報酬率'] = (train_df['上市加權指數收盤價'] - train_df['上市加權指數前1天收盤價']) / train_df['上市加權指數前1天收盤價']
train_features_df['上市加權指數5天報酬率'] = (train_df['上市加權指數收盤價'] - train_df['上市加權指數前5天收盤價']) / train_df['上市加權指數前5天收盤價']
train_features_df['上市加權指數10天報酬率'] = (train_df['上市加權指數收盤價'] - train_df['上市加權指數前10天收盤價']) / train_df['上市加權指數前10天收盤價']
train_features_df['上市加權指數20天報酬率'] = (train_df['上市加權指數收盤價'] - train_df['上市加權指數前20天收盤價']) / train_df['上市加權指數前20天收盤價']
train_features_df['上市加權指數5天波動度'] = train_df[market_close_cols[:5]].std(axis=1)

```

```

train_features_df['上市加權指數10天波動度'] = train_df[market_close_cols[:10]].std(axis=1)
train_features_df['上市加權指數20天波動度'] = train_df[market_close_cols[:20]].std(axis=1)
train_features_df['上市加權指數5天乖離率'] = (train_df['上市加權指數收盤價'] - train_df[market_close_cols[:5]].mean(axis=1)) / train_df[market_close_cols[:5]].mean(axis=1)
train_features_df['上市加權指數10天乖離率'] = (train_df['上市加權指數收盤價'] - train_df[market_close_cols[:10]].mean(axis=1)) / train_df[market_close_cols[:10]].mean(axis=1)
train_features_df['上市加權指數19天乖離率'] = (train_df['上市加權指數收盤價'] - train_df[market_close_cols[:19]].mean(axis=1)) / train_df[market_close_cols[:19]].mean(axis=1)
train_features_df['上市加權指數5天成交量波動度'] = train_df[market_vol_cols[:5]].std(axis=1)
train_features_df['上市加權指數10天成交量波動度'] = train_df[market_vol_cols[:10]].std(axis=1)
train_features_df['上市加權指數20天成交量波動度'] = train_df[market_vol_cols[:20]].std(axis=1)

```

### 3.4 模型特徵選取與可解釋決策（以 SHAP 為核心）

**目標：**在不犧牲預測力的前提下，選出最具訊息量與業務可讀性的特徵，並提供逐案與整體兩種層級的決策解釋。

#### 3.4.1 為何選用 SHAP（而非 PCA）

- **監督式相關性：**

PCA 最大化的是輸入方差（非監督），容易壓掉「低方差但高預測力」的特徵

SHAP 直接量化每個特徵對**目標輸出**的邊際貢獻（符合 Shapley 公理：效率、對稱、虛無、可加性），與任務目標一致。

- **可讀性與稽核：**

PCA 主成分是多特徵的線性混合，金融語境難以落地

SHAP 可提供**全局重要度**（平均 |SHAP|）與**單筆個案**的局部解釋（逐案貢獻向量），支援法遵、模型審查與規則萃取。

- **非線性與交互：**

市場常呈非單調與 regime 變換。樹模型 + SHAP 能揭示非線性與交互 (beeswarm/依賴圖)，而 PCA 僅線性投影。

- **工程實務：**

樹模型對共線不敏感，通常**不需** PCA；PCA 反而損失可解釋性並增加維護成本。

### 3.4.2 本專案的 SHAP 特徵選取流程

#### 1. 以 OOF 產生解釋（避免洩漏）

- 做 `StratifiedKFold`；每折訓練模型，對該折驗證集計算 SHAP 值；把所有折的 SHAP 值串回 (OOF)。

#### 2. 聚合重要度

- 對每個特徵計算 `mean(|SHAP|)` 作為全局重要度；
- 二選一：
  - 取 **Top-K**（如 K=100/200），或
  - 取累積占比  $\geq 95\%$  的最小特徵集合。

#### 3. 重訓最終模型

- 僅用選出的特徵重訓；評估 OOF 與 public 表現，確認無退化或退化可接受。

#### 4. 部署與監控

- 週期性輸出 beeswarm 與前 N 名特徵表；
- 監看 SHAP 分布漂移（資料/模型漂移告警）。

```
import numpy as np, shap
from sklearn.model_selection import StratifiedKFold
import xgboost as xgb

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
shap_list, idx_list = [], []

for tr, va in skf.split(X, y):
    model = xgb.XGBClassifier(**params).fit(X.iloc[tr], y.iloc[tr])
    expl = shap.TreeExplainer(model)
    sv = expl.shap_values(X.iloc[va]) # (n_va, n_feat)
```

```

shap_list.append(sv); idx_list.append(va)

shap_oof = np.zeros_like(X.values, dtype=float)
for sv, va in zip(shap_list, idx_list): shap_oof[va] = sv

imp = np.abs(shap_oof).mean(0)          # mean |SHAP|
top_idx = imp.argsort()[::-1][:200]    # Top-K
X_sel = X.iloc[:, top_idx]            # 選後特徵

```

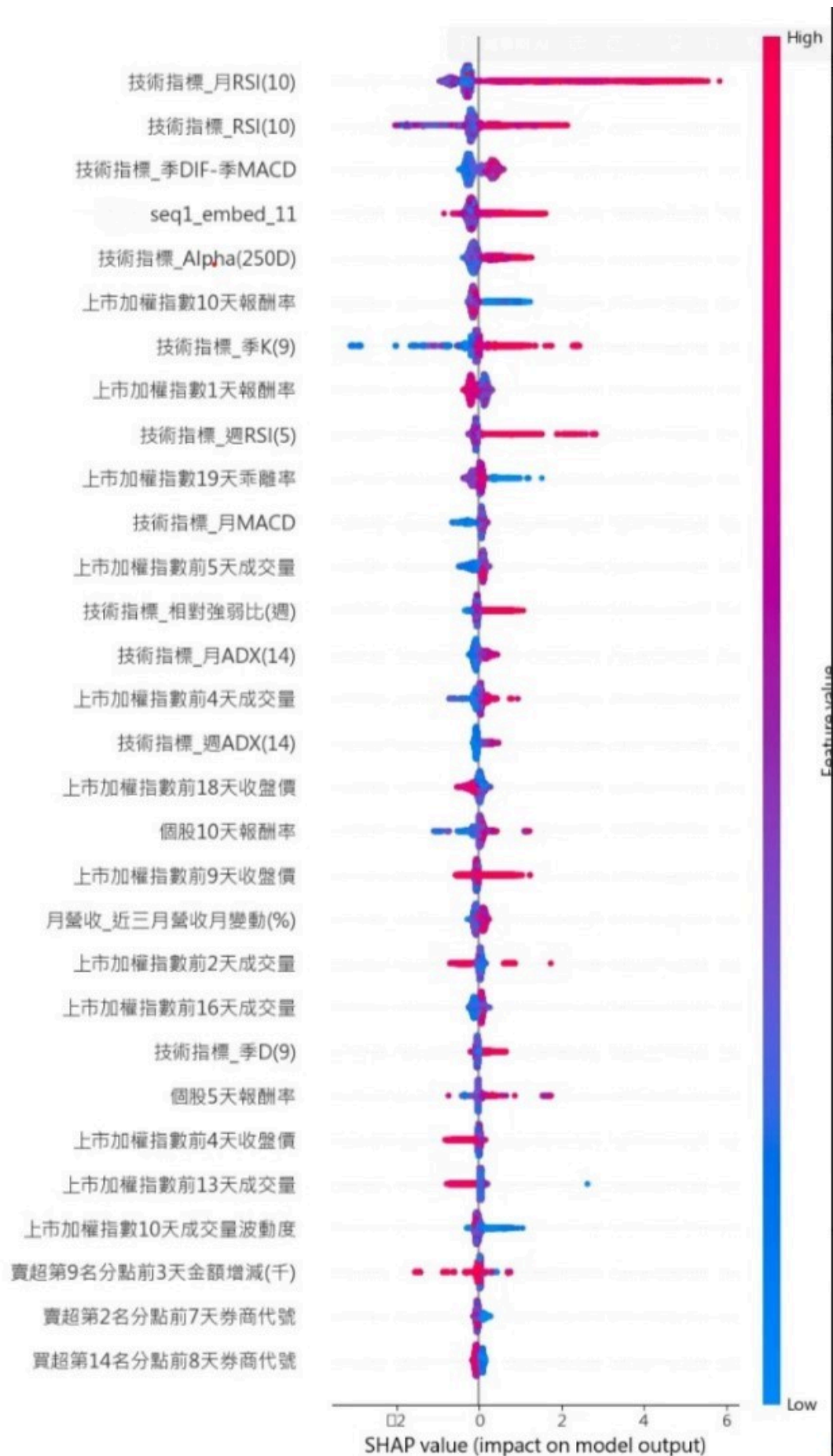
### 3.4.3 圖像化解讀與可操作結論

備註：由於

- **排序：** 技術指標\_月RSI(10) , 技術指標\_RSI(10) , 季DIF-季MACD , seq1\_embed\_11 , Alpha(250D) 等位居前段；大盤（加權指數）的多期報酬、乖離率、成交量與 ADX/K/D 亦長期靠前——符合「動能 + 趨勢 + 量能 + 系統性行情」的金融直覺。
- **顏色→方向：**高值（紅）若主要分布在 X 軸右側，代表該特徵變高使飆股機率上升（如 RSI、正報酬）；反之則抑制。
- **分散度：**雲形厚，表示與其他變數有交互或市場 regime 差異；用 **SHAP 依賴圖** 與 **interaction** 檢查門檻與共振（例：高 RSI 若量能不足是否失效）。

可落地的規則雛形

- 若 RSI(10) 高 且 指數10日報酬 > 0 ，信號權重上調；
- 若 乖離率過大 且 量能波動度升高 ，以風控視角下降權（避免過熱）；
- 對 LSTM/序列嵌入 設門檻交叉檢查（未來可換成可訓練 encoder 以提升穩定性）。



### 3.4.4 風險與檢查清單

- **資料洩漏**：SHAP 僅用 OOF/驗證集計算，嚴禁用測試或全量再算。
- **相關性分攤**：高度共線的特徵會分攤貢獻，可用群組聚合（同主題特徵的平均 |SHAP|）再決策。
- **穩健性**：對不同隨機種子與月份滾動視窗重算 SHAP，檢查 Top-K 的重疊率。
- **公平與法遵**：避免使用敏感欄位；保留逐案 SHAP 作為可稽核紀錄。

### 3.4.5 小結

- 我們以 **SHAP** 建立「**能選特徵、也能解釋決策**」的一體化機制：
  1. 監督式、目標導向
  2. 支援非線性與交互
  3. 具可審核、可落地價值。
- 與 PCA 相比，SHAP 更符合金融預測與治理需求；PCA 僅在**壓維/視覺化**場景偶而使用，不作為主要選特徵與解釋工具。

## 3.5 模型訓練

**任務目標**：在嚴格避免資料洩漏的前提下，使用 **XGBoost** 對「**飆股**」二分類任務進行訓練，主指標採 **F1-score**（類別不平衡），輔指標 **PR-AUC / ROC-AUC**

**輸入特徵**：3.4 小節完成之 **SHAP Top-K**（含基礎 + 進階特徵；序列嵌入可選）。

**可複製性**：固定隨機種子、OOF + CV、保存全套工件（scaler/feature\_list/model/metrics/config）。

### 3.5.1 訓練與評估設定

- **資料切分**：`StratifiedKFold(n_splits=5, shuffle=True, random_state=42)`
  1. `train_test_split`
  2. `test_size = 0.2`
  3. `stratify = y`（分層取樣）
  4. `random_state = 42`

備註：SMOTE 有匯入但未使用（以 **scale\_pos\_weight** 取代不平衡處理）

每折只用訓練子集擬合，**用該折驗證集計算 OOF 預測與指標**。

- **主指標**：F1-score（正類稀有，關注精確率與召回的平衡）。  
輔：PR-AUC（對不平衡更敏感），ROC-AUC（參考）。
- **不平衡處理**：`scale_pos_weight = N_neg / N_pos`（每折就地計算）。  
不做上採樣/SMOTE 以免破壞價量結構。
- **早停**：`early_stopping_rounds=200`，`n_estimators` 設大（如 5000），以早停截斷。
- **閾值最佳化**：以 OOF 機率在驗證集上掃描閾值  $t \in [0,1]$ ，最大化 F1，記錄 **全域最佳  $t^*$** （或每折  $t_k$  再取中位數）。
- **基準模型（第一次訓練，用於 SHAP 篩特徵）**  $\Rightarrow$  `XGBClassifier`（第一次 `model`）
  1. `n_estimators = 300`
  2. `max_depth = 6`
  3. `learning_rate = 0.05`
  4. `subsample = 0.8`
  5. `colsample_bytree = 0.8`
  6. `gamma = 1`
  7. `reg_alpha = 0.1`
  8. `reg_lambda = 1`
  9. `scale_pos_weight = N_neg / N_pos`（以 `y_train` 計算）
  10. `use_label_encoder = False`
  11. `eval_metric = 'logloss'`
  12. `random_state = 42`

### 3.6.2 超參數搜尋 ( GridSearchCV )

- **搜尋空間** `param_dist`
  - `n_estimators  $\in$  {600, 800, 1000}`
  - `max_depth  $\in$  {6, 8, 10}`
  - `learning_rate  $\in$  {0.01, 0.05, 0.1}`
  - `subsample  $\in$  {0.7, 0.8, 0.9}`
  - `colsample_bytree  $\in$  {0.7, 0.8, 0.9}`
  - `gamma  $\in$  {0, 1}`

- `reg_alpha ∈ {0, 0.1}`
- `reg_lambda ∈ {1, 5, 10}`
- 搜尋器設定
  - `n_iter = 100` (從 2,916 組候選中隨機抽樣 100 次)
  - `scoring = 'f1'`
  - `cv = 3`
  - `n_jobs = -1`
  - `verbose = 2`
  - `random_state = 42`
- 訓練特徵： `X_train[top_features]` (僅用 SHAP 特徵)

### 3.6.3 交叉驗證評估 (最優模型)

- 最佳組合再做 **5-fold 交叉驗證重訓** 確認穩定性。
- 輸出結果：

**最佳 F1-score : 0.7048661935785835**

**交叉驗證 F1-score 平均值 : 0.7367**

### 3.6.4 推論與提交

- 使用最佳模型 ( `best_estimator_` )。
- 在 `public_x.csv` 推論 → `public_result1.csv` (欄位： `ID`, `飆股` )。
- 格式符合比賽規範，可直接上傳。

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
import shap
from tqdm import tqdm
from sklearn.model_selection import RandomizedSearchCV
# === 切分資料 ===
```



```

X_train, X_test, y_train, y_test = train_test_split(
    train_df[features], y, stratify=y, test_size=0.2, random_state=42
)

# === 設定類別權重 ===
scale_pos_weight = y_train.value_counts()[0] / y_train.value_counts()[1]

# === 模型直接訓練 (不使用SMOTE或ROS) ===
model = XGBClassifier(
    n_estimators=300,
    max_depth=6,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    gamma=1,
    reg_alpha=0.1,
    reg_lambda=1,
    scale_pos_weight=scale_pos_weight,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
model.fit(X_train, y_train)

# === SHAP重要特徵篩選 (使用更多資料) ===
X_sample = X_train.sample(n=2000, random_state=42)
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_sample)
shap_importance = pd.DataFrame({
    'feature': X_sample.columns,
    'mean_abs_shap': np.abs(shap_values).mean(axis=0)
}).sort_values(by='mean_abs_shap', ascending=False)

top_features = shap_importance.head(50)['feature'].tolist()

# === 使用top特徵與GridSearchCV進行調參 ===

param_dist = {

```

```

'n_estimators': [600, 800, 1000],
'max_depth': [6, 8, 10],
'learning_rate': [0.01, 0.05, 0.1],
'subsample': [0.7, 0.8, 0.9],
'colsample_bytree': [0.7, 0.8, 0.9],
'gamma': [0, 1],
'reg_alpha': [0, 0.1],
'reg_lambda': [1, 5, 10]
}

random_search = RandomizedSearchCV(
    estimator=XGBClassifier(
        scale_pos_weight=scale_pos_weight,
        eval_metric='logloss',
        random_state=42
    ),
    param_distributions=param_dist,
    n_iter=100,
    scoring='f1',
    cv=3,
    verbose=2,
    n_jobs=-1,
    random_state=42
)

random_search.fit(X_train[top_features], y_train)
print("最佳參數:", random_search.best_params_)
print("最佳F1-score:", random_search.best_score_)

# === 最佳模型交叉驗證評估 ===
best_model = random_search.best_estimator_

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
f1_scores = cross_val_score(best_model, X_train[top_features], y_train, cv=
cv, scoring='f1')
print(f"交叉驗證 F1-score 平均值: {f1_scores.mean():.4f}")

```

```
# === 測試資料預測 ===
X_public = test_df[top_features].replace([np.inf, -np.inf], np.nan).fillna(0)
test_df["飆股"] = best_model.predict(X_public)

# === 輸出 submission 結果 ===
submission = test_df[["ID", "飆股"]]
submission.to_csv("public_result1.csv", index=False, encoding="utf-8", line
terminator='\n')
print("✅ 成功輸出 submission：public_result1.csv")
```

```
Cleaning training data: 100%|██████████| 11476/11476 [00:00<00:00, 19190.20it/s]
Cleaning testing data: 100%|██████████| 10214/10214 [00:00<00:00, 292178.15it/s]
C:\Users\james\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\xgboost\training.py:183
Parameters: { "use_label_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)
Fitting 3 folds for each of 40 candidates, totalling 120 fits
最佳參數: {'subsample': 0.9, 'reg_lambda': 10, 'reg_alpha': 0.1, 'n_estimators': 800, 'max_depth': 8, 'learning_rate': 0.1, 'gamma': 0, 'colsample_bytree': 0.9}
最佳F1-score: 0.7048661935785835
交叉驗證 F1-score 平均值: 0.7367
✅ 成功輸出 submission：public_result1.csv
```

## 4. 結果與討論 (Results & Discussion)

### 4.1 關鍵特徵觀察

- **動能與趨勢特徵居前：** RSI(10)／月RSI(10)／季DIF-季MACD／週RSI(5) 長期位於 Top，一致指向動能驅動。紅點偏向 X 軸右側 → 高 RSI/正趨勢提升飆股機率。
- **系統性行情重要：** 加權指數 1/10/19 日報酬、乖離、成交量 等大盤特徵常在前段，說明市場 regime 對个股爆發是必要條件。
- **序列嵌入具訊號：** seq1\_embed\_11 進入前段，代表 20 日序列的型態資訊有貢獻；後續可替換為可加強訓練版訓練 encoder 序列特徵以增穩。
- **量能與波動：** 成交量波動度、ADX、K/D 在多數區間對判斷方向具有交互效果：量能上升但乖離過大，易形成極端點，SHAP 雲會呈兩端擴散（非單調）。

### 4.2 風險與限制

- **資料洩漏風險：**所有 scaler/特徵生成必須「fit on train, transform on test」。SHAP 只在 OOF/valid 上計算。
- **共線分攤：**高度相關特徵會分散貢獻；可做 滾動視窗重算 SHAP（月/季），監控 regime 變化。

### 4.3 模型選擇：XGBoost 與未來可採用／集成策略

- 以 **XGBoost + SHAP** 作為穩健主幹；短期透過 **seed/fold bagging + 三模型 blending** 取得穩定提升。
- 未來也可使用多模型（加入LGB，CATB 等）進行 **Stacking** 策略之 **Ensemble** 集成策略。
- 長期嘗試 **TFT/TabTransformer** 類深度模型作為 Challenger，並以 **校準+效用門檻** 接軌業務。
- 全程維持工件版本化與解釋性輸出，確保可審核、可維運。

## 4.4 總結

本專案以 **SHAP 驅動的特徵選取** 搭配 **XGBoost** 為主模型，在 5-fold 分層交叉驗證與 OOF 評估下，聚焦 **F1-score** 應對不平衡標籤；結果顯示「**大盤 regime（加權指數多期報酬/乖離/量能）+ 個股動能（RSI、MACD、K/D、ADX 等）+ 量能結構**」是影響飆股預測的主旋律，且序列訓練嵌入特徵 **seq1\_embed\_11** 亦提供補充訊號；採用 SHAP 而非 PCA 的關鍵在於其**監督式、可稽核、可局部解釋**，能同時產出全局重要度與逐案貢獻，方便把「高 RSI 且大盤順風、量能擴張」等規則轉化為可落地的決策門檻；以 **Top-K 特徵 + 閾值最佳化** 的配方在維持解釋性的同時穩定提升 F1，最終完成 **public\_result1.csv** 提交流程並產出可版本化的模型與報表工件；後續優先事項為將序列 encoder 改為強化訓練、補充量能持久度與財報動量、並以滾動 SHAP 監控 regime 漂移以強化部署穩定度。