# 人工智慧與物聯網裝置設計

# Stock Agent with LINE Bot

**導師: 郭子仁 助理教授**
**學生:張俊偉 01153023**
**楊承穎 01153029**
**日期: 2025/01/17**

# 01 主題功能說明

- **串接LINE BOT**

  可以直接在Colab上執行程式並透過網址連接連線上LINE機器人，透過在LINE上面簡介的互動畫面來詢問各股的資訊等

- **回傳分析圖**

  當我們輸入關鍵字某股票之圖表時系統會自動判別並根據其股票抓取資訊，並且利用AI產生之計算公式導出其他使用者想要之指數分析，最後繪製出分析圖，並回傳至使用者之LINE中

- **AI Agent**

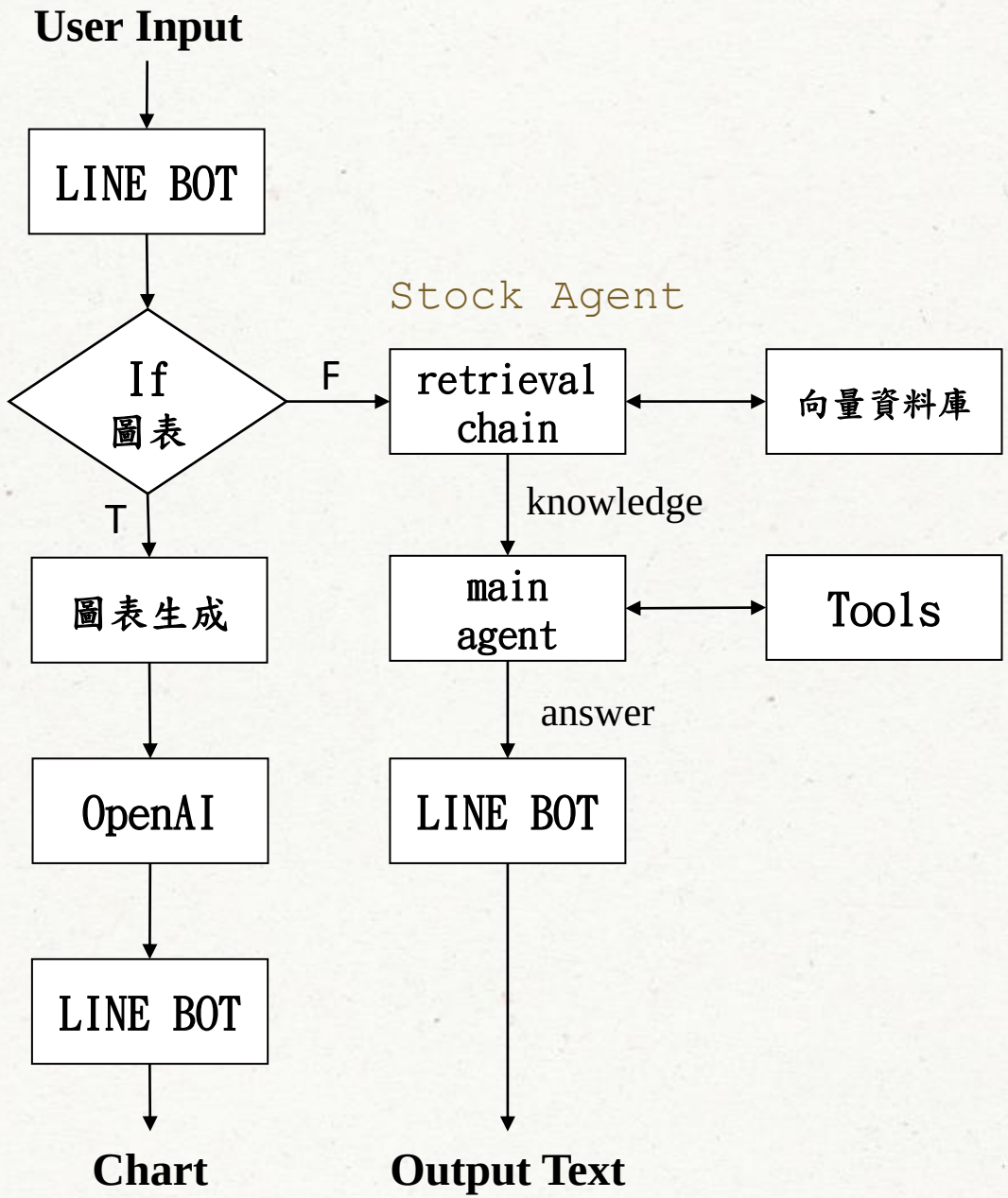
  LINE接收使用者訊息後會先經過比對向量資料庫，找到可用的資訊像是RAG的功能，之後再根據需要加入股票新聞、股票資訊、股價等資訊，之後一同將所有訊息丟給大型語言模型分析就可以改善大型語言模型訓練資料過時的問題

- **股價預測**

  利用 LSTM 預訓練模型進行時間序列分析。輸入為包含「收盤價」欄位的 CSV 檔案，程式會對資料進行縮放與處理，並以最後 60 天的股價作為輸入預測下一天的股價。處理常見錯誤如檔案缺失或欄位不正確確保穩定性與可讀性。

02

# 系統方塊圖


User Input
LINE BOT
Stock Agent
If
圖表
F
retrieval
chain
向量資料庫
knowledge
T
圖表生成
main
agent
Tools
answer
OpenAI
LINE BOT
LINE BOT
Chart
Output Text

# 03 系統操作流程

- 一開始先進到LINE BOT等待輸入訊息
- 當輸入的訊息是像是”2330 MA18圖表”的字樣會進入第一個判別式，呼叫名為plotly_stock的函式，此函式會依據不同之股票代碼搜尋資料並且繪製出不同需求之技術指標之分析圖
- 分析圖會存至stock_technical_chart裡面，再利用ngrok將透過網址將圖片傳送至LINE裡面
- 而函式create_stock_figure是透過我們給的資訊去繪製圖形的主要程式
- 最後回傳圖片地址

```python
if "圖表" in user_message:
    # 移除 "圖表"
    user_message = user_message.replace("圖表", "").strip()

    # 使用空格分開股票代號和技術指標
    parts = user_message.split(" ", 1)    # 只分割第一次出現的空格

    if len(parts) == 2:
        stock_number = parts[0]    # 股票代號
        indicator = parts[1]      # 技術指標

        #傳送股票代碼
        plotly_stock(stock_number, start='2022-01-01', end= None,
                                   indicator=indicator)      #生成圖片
```

```python
111 #  主函式
112 def  plotly_stock(stock_id,  start=None,  end=None,  indicator='MACD'):
113         df  =  download_stock_data(stock_id,  start,  end,  indicator)
114         fig  =  create_stock_figure(stock_id,df)
115         fig.write_image(temp_path)
116         size_change()
117         fig.show()
```

```python
    # 圖片的公開 URL
    image_url = f"{ngrok_url}/{original_image_path}"

    # 回應一張圖片
    image_message = ImageSendMessage(
        original_content_url=
            f"{ngrok_url}/{original_image_path}",    # 圖片完整尺寸 URL
        preview_image_url=
            f"{ngrok_url}/{preview_image_path}"    # 預覽圖片 URL
    )
    api.reply_message(event.reply_token,  image_message)
```

# 03 系統操作流程

- 當LINE BOT訊息不為前面之"圖表"時會進入AI Agent也就是客製化去找答案

```
else:      # 進入AI agent
        reply_text = stock_agent(user_message)
        api.reply_message(
                event.reply_token,
                TextSendMessage(text=reply_text)
        )
```

- 第一步會先經過檢索去搜尋相關文字之消息並彙整成retrieval_output
- 第二步則是結合查到之資訊和輸入之訊息一同傳送給大型語言模型

```
def stock_agent(user_question):
    # 1. 執行檢索 Chain
    retrieval_output = retrieval_chain.invoke({"input": user_question})

    # 2. 主要 Agent 執行
    main_agent_output = main_agent.invoke({
            "input": user_question,
            "knowledge": str(retrieval_output['answer'])
    })

    return main_agent_output['output']
```

- 大型語言模型也會根據需要去執行前面的Tools去幫助大型語言模型解決問題

```
76 # 初始化 Agent
77 main_agent = initialize_agent(
78         tools=main_agent_tools,
79         llm=llm,
80         agent="structured-chat-zero-shot-react-description",
81         verbose=True,
82         handle_parsing_errors=False,
83         input_variables=["user_question"],
84 )
```

# 03 系統操作流程

- 所有Tool，詳細程式碼請參考colab網站

```python
main_agent_tools = [
    Tool(
        name="query_stock",
        func=query_stock,
        description="Queries stock info by name or ID. Returns: 'stock_name', 'stock_id'."
    ),
    Tool(
        name="stock_price",
        func=stock_price,
        description="Fetches historical stock price data. Input: stock ID (str), days (int). Returns: 'stock_price_path'. Use 'extract_stock_price' for price details."
    ),
    Tool(
        name="stock_info",
        func=stock_info,
        description="Fetches stock info. Input: stock ID (str). Returns: 'company_name', 'current_price', 'business_summary', 'market_cap', 'industry', '52_week_high', '52_week_low'"
    ),
    Tool(
        name = "stock_fundamental",
        func= stock_fundamental,
        description= "Fetches fundamental data. Input: stock ID (str). Returns: 'quarterly_dates', 'revenue_growth_rate', 'EPS', 'EPS_quarterly_growth_rate' or None if 大盤."
    ),
    Tool(
        name = "stock_news",
        func= stock_news,
        description= "Fetches news. Input: stock name (str) or '大盤'. Returns: list of news."
    ),
      Tool(
        name="ai_helper",
        func=ai_helper,
        description="Generates and executes Python code on CSV data based on user request. Input: CSV path (str), user message (str). Returns: message with 'stock_price_path'. Then use 'extract_stock_price'
    ),
    Tool(
        name="extract_stock_price",
        func=extract_stock_price,
          description="Extracts stock price details, only the latest 20 records. Input: CSV path (str). Returns: 'dates', 'closing_prices', 'opening_prices', 'highest_prices', 'lowest_prices', 'volume', 'daily_r
    ),
    Tool(
        name="predicted_stock_price",
        func=predicted_stock_price,
        description="Use only if the user asks for a 'prediction' or 'forecast'. Predicts the next day's stock price. Input: stock's CSV path (str). Returns: string with predicted price."
    )
]
```

查詢股名或股號

取得股價表

取得股價資料

取得股票資訊

取得基本面資料

取得新聞資料

- 最後之結果回傳給LINE BOT發送給使用者

```python
else:      # 進入AI agent
        reply_text = stock_agent(user_message)
        api.reply_message(
                event.reply_token,
                TextSendMessage(text=reply_text)
        )
```

# 04 程式說明介紹

爬取股號、股名對照表

```
[8]  1 #  取得全部股票的股號、股名
     2 def  stock_name():
     3     """
     4     Fetches  stock  names  and  IDs  from  the  Taiwan  Stock  Exchange  website  and  saves  it  into  csv.
     5 
     6     Returns:
     7         str:  The  path  to  the  saved  CSV  file.
     8     """
     9     response  =  requests.get('https://isin.twse.com.tw/isin/C_public.jsp?strMode=2')
    10     url_data  =  BeautifulSoup(response.text,  'html.parser')
    11     stock_company  =  url_data.find_all('tr')
    12 
    13     #  資料處理
    14     data  =  [
    15         (row.find_all('td')[0].text.split('\u3000')[0].strip(),
    16             row.find_all('td')[0].text.split('\u3000')[1],
    17             row.find_all('td')[4].text.strip())
    18         for  row  in  stock_company[2:]  if  len(row.find_all('td')[0].text.split('\u3000')[0].strip())  ==  4
    19     ]
    20 
    21     df  =  pd.DataFrame(data,  columns=['股號',  '股名',  '產業別'])
    22     df['股號']  =  df['股號'].astype(str)
    23 
    24     #  存成  CSV  檔案
    25     stock_name_table_path  =  'stock_name_table.csv'
    26     df.to_csv(stock_name_table_path,  index=False,  encoding='utf-8-sig')
    27 
    28     return  stock_name_table_path
```

```
[9]  1 stock_name_table_path  =  stock_name()
```

- 8~11: requests.get從台灣證券交易所的指定網址下載網頁內容，找到所有的 <tr> 標籤
- 14~19: 使用列表推導式處理網頁表格資料，逐行提取有用資訊股號長度為 4（只保留正規股票）
- 21~22: 將資料存成 DataFrame，將「股號」欄位轉成字串型別，避免數字格式化問題
- 24~26: 存成 CSV 檔案，設定存檔路徑為 stock_name_table.csv，不包含索引，並使用 UTF-8 編碼

# 04 程式說明介紹

查詢股名或股號

```
[10]  1 # 查詢股名或股號
      2 def query_stock(query):
      3         """
      4         Queries stock information (name or ID) from a CSV file.
      5
      6         Args:
      7                 query (str): The stock name or ID to search for.
      8
      9         Returns:
     10                 dict: A dictionary containing the stock name and ID, or an error message if not found.
     11                         Example: {'股名': '台積電', '股號': '2330.TW'}
     12                         or {'error': '查無資料'}
     13         """
     14         # 載入 CSV 檔案
     15         df = pd.read_csv('stock_name_table.csv', dtype={'股號': str})
     16
     17         # 查詢條件
     18         result = df[(df['股名'] == query) | (df['股號'] == query)]
     19
     20         # 格式化輸出
     21         if not result.empty:
     22                 stock_info = result.iloc[0]   # 只取第一筆符合的結果
     23                 return {"股名": stock_info['股名'], "股號": f"{stock_info['股號']}.TW"}
     24         else:
     25                 return {"error": "查無資料"}
```

```
[11]  1 query_stock(query = '2330')
```

- 15: 載入 CSV 檔案其中的內容載入為 DataFrame
- 18:查詢是否有符合的股票名稱或股票代號，任何一個條件符合即可
- 21~25: 結果判斷與格式化輸出，判斷查詢結果是否為空，如果有結果取第一筆結果

# 04 程式說明介紹

- 16~17: 解析輸入參數，將 input_dict 拆分為 stock_id（股票代號）和 days（天數）
- 19~20: 搜尋^TWII會比搜尋大盤的結果更好
- 22~23: 設定資料的時間範圍
- 25~29: 使用 yfinance 函式 download 下載股票資料
- 32~35:處理資料的列名
- 38~40: 檢查是否存在名為 stock_price 的資料夾，若不存在則創建
- 43~44:定義 CSV 檔案的儲存路徑，將資料儲存為 UTF-8 編碼的 CSV 檔案

取得股價表

```
[12]
def stock_price(input_dict) -> dict:
    """
    Fetches historical stock price data for a given stock ID and number of days, and saves it as a CSV file.

    After fetching the data, you can use 'extract_stock_price' tool to get stock price information.

    Args:
        input_dict (dict): A dictionary containing 'stock_id' (str) and 'days' (int).
                           Example: {'stock_id': '2330.TW', 'days': 30}

    Returns:
        dict: A dictionary containing the 'stock_price_path' (str) to the saved CSV file.
              Example: {'stock_price_path': 'stock_price/2330.TW.csv'}
              or str if an error occurs.
    """
    stock_id, days = input_dict.split(",")
    days = int(days)

    if stock_id == "大盤":
        stock_id="^TWII"

    end = dt.date.today() # 資料結束時間
    start = end - dt.timedelta(days=days) # 資料開始時間
    # 下載資料
    try:
        df = yf.download(stock_id, start=start, progress=False)

    except:
        return f"無法獲取 {stock_id} 的資訊"

    #更換列名
    if 'Adj Close' in df.columns:
        df.columns = ['調整後收盤價', '收盤價', '最高價', '最低價', '開盤價', '成交量']
    else:
        df.columns = ['收盤價', '最高價', '最低價', '開盤價', '成交量']


    dir_name = "stock_price"
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)

    # 存成 CSV 檔案
    stock_price_path = f'{dir_name}/{stock_id}.csv'
    df.to_csv(stock_price_path, index=True, encoding='utf-8-sig')

    return {"stock_price_path" : f"{stock_price_path}"}
```

```
[13]
stock_price("2330.TW,10")
```

```
{'stock_price_path': 'stock_price/2330.TW.csv'}
```

# 04 程式說明介紹

- 26~27: 使用 pandas 讀取 CSV 檔案，並將 Date 欄位解析為日期型別
- 28: 取出資料集中最後 20 筆資料進行分析
- 32~40: 建立輸出字典，日期轉為字串格式，每日報酬利用 pct_change() 計算收盤價的日變動率
- 43~45: 如果 CSV 檔案中存在其他欄位，則將其動態添加到 data 中
- 48~49: 確保字典中的值以 np.nan 表示空值，以便進行後續分析

取得股價資料

```python
def extract_stock_price(stock_price_path):
    """
    Extracts specific stock price information from a CSV file.

    Args:
        stock_price_path (str): The path to the CSV file containing stock price data.
                                This should be a valid file path to the CSV generated by `stock_price` tool.

    Returns:
        dict or str: If successful, returns a dictionary containing '日期', '收盤價', '開盤價', '最高價', '最低價', '成交量',
                     '每日報酬', '漲跌價差' and other column data. If an error occurs, returns a str error message.
                     Example:
                     {
                         '日期': ['2024-01-01', '2024-01-02'],
                         '收盤價': [100.0, 102.0],
                         '開盤價': [99.0, 101.0],
                         '最高價': [101.0, 103.0],
                         '最低價': [98.0, 100.0],
                         '成交量': [1000, 1200],
                         '每日報酬': [nan, 0.02],
                     }

    """

    try:
        df = pd.read_csv(stock_price_path, parse_dates=['Date'])
        df.set_index('Date', inplace=True) # 確保日期欄位是索引

        # 只選取最後 20 筆資料
        df = df.tail(20)

        data = {
            '日期': df.index.strftime('%Y-%m-%d').tolist(),
            '收盤價': df['收盤價'].tolist(),
            '開盤價': df['開盤價'].tolist(),
            '最高價': df['最高價'].tolist(),
            '最低價': df['最低價'].tolist(),
            '成交量': df['成交量'].tolist(),
            '每日報酬': df['收盤價'].pct_change().tolist(),
        }

        # 動態添加其餘欄位
        for column in df.columns:
            if column not in data:  # 避免重複添加
                data[column] = df[column].tolist()

        # 處理空值
        for key in data:
            data[key] = [np.nan if x is None else x for x in data[key]]

        return data

    except Exception as e:
        return f"[Error] An error occurred: {str(e)}"
```

# 04 程式說明介紹

取得股票資訊

```python
# 股票資訊查詢函數
def stock_info(stock_id = "2330.TW"):
    """
    Fetches general information about a stock.

    Args:
        stock_id (str): The stock ID (e.g., "2330.TW").

    Returns:
        dict: A dictionary containing stock information, or an error message string if the stock information can't be fetched.
              Example:{
                  '公司名稱': info.get('longName', 'N/A'),
                  '現價': info.get('currentPrice', 'N/A'),
                  '企業簡介': info.get('longBusinessSummary', 'N/A'),
                  '市值': info.get('marketCap', 'N/A'),
                  '產業': info.get('industry', 'N/A'),
                  '52週高點': info.get('fiftyTwoWeekHigh', 'N/A'),
                  '52週低點': info.get('fiftyTwoWeekLow', 'N/A')
              }
    """
    try:
        stock = yf.Ticker(stock_id)
        info = stock.info
        return {
            '公司名稱': info.get('longName', 'N/A'),
            '現價': info.get('currentPrice', 'N/A'),
            '企業簡介': info.get('longBusinessSummary', 'N/A'),
            '市值': info.get('marketCap', 'N/A'),
            '產業': info.get('industry', 'N/A'),
            '52週高點': info.get('fiftyTwoWeekHigh', 'N/A'),
            '52週低點': info.get('fiftyTwoWeekLow', 'N/A')
        }
    except:
        return f"無法獲取 {stock_id} 的資訊"
```

```python
[18] stock_info(stock_id = "2330.TW")
```

- 2: 函式定義，預設參數為 "2330.TW“
- 22~23:使用 yfinance.Ticker 建立與指定股票的連結。stock.info 包含股票的完整資訊（以字典形式返回）。
- 24~32: 使用 dict.get() 方法，提取 info 中的目標欄位，若目標欄位不存在，預設返回 'N/A'

# 04 程式說明介紹

- 18~19: 如果 stock_id 是 "大盤"，函式直接返回 None。
- 22: 使用 yfinance.Ticker 來取得指定股票的資料。
- 27~30: 提取每季的營收數據（Total Revenue），並計算每季的營收成長率。使用 pct_change 計算相對變化百分比
- 33~34: 提取每季的每股盈餘數據
- 38~40: 計算每股盈餘的季增率，即每季 EPS 的變化率
- 43~44:提取每季的日期，並將其轉換為 YYYY-MM-DD 格式
- 47~52: 創建一個包含日期、營收成長率、EPS 和 EPS 季增率的字典，並將其返回。 確保各個資料欄位的長度一致

取得基本面資料

[19]

```python
def stock_fundamental(stock_id="2330.TW"):
    """
    Fetches fundamental data for a stock (Revenue growth rate, EPS, EPS quarterly growth rate).

    Args:
        stock_id (str): The stock ID (e.g., "2330.TW").

    Returns:
        dict or None: A dictionary containing fundamental data, or None if the input is '大盤'.
            Example:{
                "季日期": dates[:len(quarterly_revenue_growth)],
                "營收成長率": quarterly_revenue_growth.tolist(),
                "EPS": quarterly_eps[:len(quarterly_revenue_growth)].tolist(),
                "EPS 季增率": quarterly_eps_growth[:len(quarterly_revenue_growth)].tolist(),
            }
        or str if an error occurs
    """
    if stock_id == "大盤":
        return None

    try:
        stock = yf.Ticker(stock_id)
    except:
        return f"無法獲取 {stock_id} 的資訊"

    # 營收成長率
    quarterly_revenue = stock.quarterly_financials.loc["Total Revenue"].dropna()
    quarterly_revenue_growth = np.round(
        quarterly_revenue.pct_change(-1, fill_method=None).dropna().tolist(), 2
    )

    # 每季 EPS
    quarterly_eps = np.round(
        stock.quarterly_financials.loc["Basic EPS"].dropna().tolist(), 2
    )

    # EPS 季增率
    quarterly_eps_growth = np.round(
        stock.quarterly_financials.loc["Basic EPS"].pct_change(-1, fill_method=None).dropna().tolist(), 2
    )

    # 轉換日期
    dates = [
        date.strftime("%Y-%m-%d") for date in stock.quarterly_financials.columns
    ]

    data = {
        "季日期": dates[:len(quarterly_revenue_growth)],
        "營收成長率": quarterly_revenue_growth.tolist(),
        "EPS": quarterly_eps[:len(quarterly_revenue_growth)].tolist(),
        "EPS 季增率": quarterly_eps_growth[:len(quarterly_revenue_growth)].tolist(),
    }

    return data
```

# 04 程式說明介紹

取得新聞資料

```python
# 新聞資料
def stock_news(stock_name="大盤"):
    """
    Fetches news articles related to a specific stock.

    Args:
        stock_name (str): The name of the stock or '大盤' to get market news.

    Returns:
        list: A list of lists, where each inner list contains [stock name, formatted_date, title, news_content]
    """
    if stock_name == "大盤":
        stock_name = "台股 -盤中速報"

    data = []
    # 取得 Json 格式資料
    json_data = requests.get(f'https://ess.api.cnyes.com/ess/api/v1/news/keyword?q={stock_name}&limit=5&page=1').json()

    # 依照格式擷取資料
    items = json_data['data']['items']

    for item in items:
        # 檢查 item["category"] 是否是列表，且包含至少一個元素
        if not item.get("category") or not isinstance(item["category"], list) or not item["category"]:
            continue

        # 確保 category[0] 是字典，且包含 "name" 鍵
        if not isinstance(item["category"][0], dict) or "name" not in item["category"][0]:
            continue

        if str(item["category"][0]["name"]) != "台股新聞":
            continue

        # 網址、標題和日期
        news_id = item["newsId"]
        title = item["title"]
        publish_at = item["publishAt"]

        # 使用 UTC 時間格式
        utc_time = dt.datetime.utcfromtimestamp(publish_at)
        formatted_date = utc_time.strftime('%Y-%m-%d')

        # 前往網址擷取內容
        url_content = requests.get(f'https://news.cnyes.com/news/id/{news_id}').content
        soup = BeautifulSoup(url_content, 'html.parser')
        p_elements = soup.find_all('p')

        # 提取段落內容
        p = ''
        for paragraph in p_elements[4:]:
            clean_text = paragraph.get_text(strip=True)
            p += clean_text

        # 移除關鍵字後的內容
        keywords = r'(按讚|訂閱|https|上一篇|下一篇|立即加入|LINE|老師)'
        p = re.split(keywords, p, maxsplit=1)[0].strip()

        # 清理重複換行
        p = re.sub(r'\n{2,}', '\n', p)    # 將多個換行合併為一個

        # 格式化數據顯示（如去掉過多小數點或雜訊）
        p = re.sub(r'(\d+\.\d{2})\d+', r'\1', p)    # 保留小數點後兩位

        # 移除一些無用的段落內容
        p = re.sub(r'(《.*?》|\=+|\-+)', '', p)    # 移除分隔線和標題
        p = p.strip()    # 清除首尾多餘的空格或換行

        # 添加整理後的內容到資料中
        data.append([stock_name, formatted_date, title, p])

    return data
```

- 17:使用 requests.get 向 cnyes 的 API 發送請求，並返回 JSON 格式的資料。
- 24~32: 從抓取的資料中，篩選出符合「台股新聞」或「台股公告」類別的新聞項目。
- 35~37: 提取每條新聞的 ID、標題及發佈時間。
- 40~41: 將新聞的發佈時間從 UTC 時間戳轉換為 YYYY-MM-DD 格式
- 44~52: 使用 requests.get 獲取每條新聞的詳細頁面內容，並使用 BeautifulSoup 解析 HTML 結構。提取所有 <p> 標籤中的文字，從第 5 個段落開始（通常前 4 個段落是無關內容）。
- 55~66: 使用正則表達式清除無關內容（例如社交媒體按鈕、廣告等），合併多個換行，移除無用的分隔符號
- 69: 將整理後的數據（股票名稱、日期、標題、內容）儲存到 data 列表中

# 04 程式說明介紹

- 第一個extract_code函式使用正則表達式（Regex）查找字串中的程式碼區塊，格式是以「python」開始，並以「」結束。 透過 RegexParser 來提取「python」和「」之間的內容，這部分就是 Python 程式碼。 將程式碼中的「python」和「」符號去除，只保留實際的 Python 程式碼。 如果成功提取程式碼，返回乾淨的 Python 程式碼。如果提取不到程式碼，則返回「No valid code found.」。

- 第二個execute_gen_code，exec(code_str, globals()) 被用來執行傳入的 code_str。這會在全局範疇內執行程式碼，因此可以對外部變數如 df進行操作。程式碼中包含對 DataFrame df 進行的操作，並將其處理結果賦值給新的變數new_df。該程式碼應該有一個名為 calculate(df) 的函式，這個函式會根據需求處理 DataFrame。

AI Helper to generate code

```python
# 定義程式碼提取函數
def extract_code(response):
    """
    Extracts code from a string response, specifically from a markdown code block.

    Args:
        response (str): The string containing the markdown code block.

    Returns:
        str: The extracted code or 'No valid code found.' if no code block is present
             or an error message string if extraction fails.
    """
    try:
        # 使用 RegexParser 從 llm 的輸出中提取程式碼
        code_extractor = RegexParser(
            regex=r"```python\n(.*?)\n```", output_keys=["code"], default_output_key="code"
        )
        extracted_code = code_extractor.parse(response.content)
        if "code" in extracted_code and extracted_code["code"]:
            code = extracted_code["code"].strip()
            code = code.replace('```python', '').replace('```', '')
            return code
        else:
            return "No valid code found."
    except Exception as e:
        return f"Code extraction error: {str(e)}"

# 執行生成程式碼
def execute_gen_code(code_str, df):
    """
    Executes generated code on a given DataFrame.

    Args:
        code_str (str): The code to execute.
        df (pd.DataFrame): The DataFrame to process.

    Returns:
        pd.DataFrame or str: The processed DataFrame or an error message string if code execution fails.
    """
    try:
        exec(code_str, globals())
        new_df = calculate(df)
        return new_df
    except Exception as e:
        return f"Error during code execution: {str(e)}"
```

# 04 程式說明介紹

- 2~17: 模板指示生成的程式碼將包含： 函數名稱 calculate(df) 只使用提供的 DataFrame 欄位 返回處理後的 DataFrame 以 def calculate(df): 開頭，並且不包含導入語句 只生成程式碼，不附加說明或解釋
- 20~23: 這個函式接收一個字典input_dict並處理用戶輸入的字符串
- 25~30: prompt將處理過的輸入傳遞給 PromptTemplate，用來生成程式碼。 llm用來執行生成的程式碼並返回結果。

```
[ ]
# 定義 Prompt (用於生成程式碼)
prompt = PromptTemplate(
        input_variables=["user_msg", "columns"],
        template=(
                "You are a professional Python code generation assistant.\n"
                "Your task is to generate Python code based on specific user requirements.\n"
                "You are provided with a dataframe (df) with the columns {columns}. "
                "Generate a Python function named 'calculate(df)' that processes the dataframe "
                "according to the user requirements: {user_msg}.\n"
                "The code should:\n"
                "- Only use the given columns from the dataframe.\n"
                "- Return the processed dataframe.\n"
                "- Start with 'def calculate(df):' and only include the function definition code. Do
                "- Enclose your code with ```python ```\n\n"
                "Strictly provide the Python code only. No explanations."
        )
)

# 建立 RunnableSequence (用於生成程式碼的 chain)
def split_input(input_dict):
        input_str = input_dict["input"]
        user_msg, columns = input_str.split(", columns: ", 1)
        return {"user_msg": user_msg, "columns": columns}

code_gen_chain = RunnableSequence(
        RunnablePassthrough() |
        RunnableLambda(split_input) |
        prompt |
        llm
)
```

# 04 程式說明介紹

- 這裡主要就是串連前面所定義的函式，全部串在一起，讓使用者輸入之指令得到結果

```
[ ]
# ai_helper 函數 (使用 LangChain)
def ai_helper(input_str):
    """
    Generates Python code using an LLM based on user message and DataFrame columns loaded from a CSV file,
    and then executes it to update the CSV.

    After generating the code, you can use 'extract_stock_price' tool to get stock price information.

    Args:
        stock_price_path (str): The path to the CSV file containing stock price data,
                                which should contain columns that the generated code will use.
        user_msg (str): The user's request for code generation.

    Returns:
        str: If successful, return a message containing the new stock_price_path,
             example: "New data is created. {\"stock_price_path\" : \"{stock_price_path}\"}"
             or an error message if code generation or execution fails.

    Raises:
        FileNotFoundError: If the provided file path does not exist.
        Exception: If any other error occurs during the process.
    """
    try:
        stock_price_path, user_msg = input_str.split(",")
        df = pd.read_csv(stock_price_path)
        df_columns = df.columns.tolist()
        input_data = {"input": f"{user_msg}, columns: {', '.join(df_columns)}"}
        response = code_gen_chain.invoke(input_data)
        code_str = extract_code(response)
        df = execute_gen_code(code_str,df)
        df.to_csv(stock_price_path, index=False, encoding='utf-8-sig', float_format="%.2f")
        return f"New data is created. \"stock_price_path\" : \"{stock_price_path}\""
    except Exception as e:
        return f"Error in ai_helper: {str(e)}"
```

# 04 程式說明介紹

- 我們已在Kaggle上訓練好預測模型，data是抓取十年的股價資訊來訓練，所以一開始先git clone
- 20~21: 從指定的 stock_price_path 路徑讀取 CSV 文件，並提取出收盤價，這一列包含歷史股價數據。
- 24~27: 讀取的股價數據會進行標準化處理。MinMaxScaler 用來將數據縮放到 0 和 1 之間，以便 LSTM 模型能夠更好地處理。
- 29~30: 只使用最近 60 天的股價數據來預測下一天的股價。將這些數據重塑為適合 LSTM 模型的形狀 (1, 60, 1)，即 1 個樣本、60 個時間步長和 1 個特徵。
- 33~34: loaded_model 是已經訓練好的 LSTM 模型，通過 .predict() 方法來預測下一天的股價。

LSTM預測模型

```
[ ]  1 !rm -R /content/Stock_Analysis
     2 !git clone https://github.com/steak0069/Stock_Analysis/
```

```
rm: cannot remove '/content/Stock_Analysis': No such file or directory
Cloning into 'Stock_Analysis'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 26 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (26/26), 29.68 MiB | 11.86 MiB/s, done.
Resolving deltas: 100% (3/3), done.
```

```
[ ]  1 import tensorflow as tf
     2 from tensorflow.keras.models import load_model
     3 from sklearn.preprocessing import MinMaxScaler
     4
     5 model_path = "/content/Stock_Analysis/predictions_model/trained_model.keras"
     6
     7 # 載入儲存的模型
     8 loaded_model = load_model(model_path)
```

```
[ ]  1 def predicted_stock_price(stock_price_path):
     2     """
     3     Predicts the next day's stock price using a pre-trained LSTM model.
     4
     5     Args:
     6         stock_price_path (str): The path to the CSV file containing historical stock price data,
     7                                 which should contain a '收盤價' column.
     8
     9     Returns:
    10         str: A string containing the predicted next day's stock price, formatted to two decimal places.
    11              Example: "預測的明天股價: 123.45"
    12         or str if an error occurs
    13
    14     Raises:
    15         FileNotFoundError: If the provided file path does not exist.
    16         KeyError: If the CSV file does not contain a '收盤價' column.
    17         Exception: If any other error occurs during the prediction process.
    18     """
    19     try:
    20         df = pd.read_csv(stock_price_path)
    21         data = df['收盤價']
    22
    23         # 將 Series 轉換為 2 維陣列
    24         data_reshaped = data.values.reshape(-1, 1)
    25
    26         scaler = MinMaxScaler()
    27         scaled_data = scaler.fit_transform(data_reshaped)
    28
    29         Xlstm = scaled_data[-60:]
    30         Xlstm = Xlstm.reshape(1, Xlstm.shape[0], 1)      # Xlstm.shape = (1, 60, 1)
    31
    32         # 假設 loaded_model 已經被定義
    33         predicted_price = loaded_model.predict(Xlstm, verbose=0)
    34         predicted_price = scaler.inverse_transform(predicted_price)
    35
    36         return f"預測的明天股價: {predicted_price[0][0]:.2f}"
    37
    38     except FileNotFoundError:
    39         return f"[Error] No such file or directory: {stock_price_path}"
    40     except KeyError:
    41         return f"[Error] CSV file does not contain a '收盤價' column."
    42     except Exception as e:
    43         return f"[Error] An error occurred: {str(e)}"
```

# 04 程式說明介紹

## RAG

### 加載向量資料

```
[ ]  1 vector_db_path = "/content/Stock_Analysis/vectordb/stockvector.db"
```

```
[ ]  1 embedding_model = HuggingFaceEmbeddings(model_name="BAAI/bge-m3",
     2 encode_kwargs={"normalize_embeddings": True})
```

顯示隱藏的輸出內容

```
[ ]  1 # 加載向量資料
     2 vectordb = FAISS.load_local(vector_db_path, embedding_model, allow_dangerous_deserialization=True)
     3 # Create retriever for later use
     4 retriever = vectordb.as_retriever()
```

## Retrieval_chain

```
[33]  1 from langchain_core.prompts import ChatPromptTemplate
      2 from langchain.chains.combine_documents import create_stuff_documents_chain
      3 from langchain.chains import create_retrieval_chain
```

```
[34]  1 template = """
      2 You are a Q&A chat bot.
      3 Use the given context only, answer the question.
      4
      5 <context>
      6 {context}
      7 </context>
      8
      9 Question: {input}
     10 """
     11
     12 # Create a prompt template
     13 prompt = ChatPromptTemplate.from_template(template)
     14 doc_chain = create_stuff_documents_chain(llm, prompt)
     15 retrieval_chain = create_retrieval_chain(retriever, doc_chain)
```

- 加載已經保存的向量資料庫，並使用它來創建一個檢索器 (retriever)。用於基於向量搜索的任務，比如在大量文檔中查找最相關的信息，幫助我們解決問題。
- 向量資料庫 (FAISS) 用於儲存大量經過向量化的文本數據，使得查詢能夠快速地與資料庫進行相似度比較。
- 檢索器 (Retriever) 是一個中介，它負責處理查詢並返回與資料庫中向量最接近的結果。這對於知識檢索系統或基於文本的搜尋系統非常有用。

# 04 程式說明介紹

- 與前面都相像，構建模板後丟給大型語言模型讓他去回答問題，這邊加上Tools幫助語言模型去尋找需要的資源找尋我們所撰寫之函式，可以更精準且有效的回答問題

```python
def stock_agent(user_question):
    # 1. 執行檢索 Chain
    retrieval_output = retrieval_chain.invoke({"input": user_question})

    # 2. 主要 Agent 執行
    main_agent_output = main_agent.invoke({
        "input": user_question,
        "knowledge": str(retrieval_output['answer'])
    })

    return main_agent_output['output']
```

Agent

```python
# 主要 Agent 的 Prompt 和初始化
main_agent_prompt = PromptTemplate(
    input_variables=["user_question", "knowledge"],
    template=(
        "You are a financial analyst agent. Your task is to analyze the user's question about the stock market,you can explain the data in a professional and understandable
        "You should follow the following keys:\n"
        "- `user_question`: The original user question.\n"
        "- `knowledge`: Relevant knowledge retrieved from the vector database, if available.\n"
        "- `stock_id`: The stock ID (e.g., '2330.TW') if the question is about a specific stock. Otherwise, leave it as an empty string.\n"
        "- `stock_name`: The stock name (e.g., '台積電') if the question is about a specific stock. Otherwise, leave it as an empty string.\n"
        "- `days`: The number of days of stock data needed. If not explicitly stated in the question, or not inferable, set it to 30.\n"
        "- `stock_price_path`: The path to the CSV file with stock price data, if the 'stock_price' tool is called, otherwise leave it as an empty string\n"
        "- `stock_data`: A dictionary containing data fetched from the tools:\n"
        "    - `stock_price`: Stock price data if needed, otherwise an empty dictionary.\n"
        "    - `stock_info`: Stock information if needed, otherwise an empty dictionary.\n"
        "    - `stock_fundamental`: Fundamental stock data if needed, otherwise an empty dictionary.\n"
        "    - `stock_news`: News about the stock if needed, otherwise an empty string.\n"
        "- `code_str`: The code generated by ai_helper if needed, otherwise an empty string.\n"
        "Your process should follow these steps:\n"
        "1. Analyze the `user_question` and the `knowledge` to understand the user's needs.\n"
        "2. Use tools to retrieve the required stock data. Always use 'query_stock' to get stock_id first if user question related to a specific stock.\n"
        "3. If the user question explicitly asks for a 'prediction' or 'forecast', and you have 60 days stock price data and the stock price path, then use `predicted_stock
        "4. If user need the technical indicators data such as 'MA'、'MACD'、'RSI'... first you need to perpare 60 days stock price data from 'stock_price', then call the `s
        "5. Summarize all the information and put it in analysis, final response your analysis and the answer\n"
        "6. If the user question does not refer to any specific stocks, leave `stock_id` and `stock_name` as empty string, `days` can be set as default 30 days, and do no
        "User question: {user_question}\n"
        "knowledge: {knowledge}\n"
    )
)


# 主要 Agent 工具列表
main_agent_tools = [
    Tool(
        name="query_stock",
        func=query_stock,
        description="Queries stock info by name or ID. Returns: 'stock_name', 'stock_id'."
    ),
    Tool(
        name="stock_price",
        func=stock_price,
        description="Fetches historical stock price data. Input: stock ID (str), days (int). Returns: 'stock_price_path'. Use 'extract_stock_price' for price details."
    ),
    Tool(
        name="stock_info",
        func=stock_info,
        description="Fetches stock info. Input: stock ID (str). Returns: 'company_name', 'current_price', 'business_summary', 'market_cap', 'industry', '52_week_high', '52_week_low'"
    ),
    Tool(
        name = "stock_fundamental",
        func= stock_fundamental,
        description= "Fetches fundamental data. Input: stock ID (str). Returns: 'quarterly_dates', 'revenue_growth_rate', 'EPS', 'EPS_quarterly_growth_rate' or None if 大盤."
    ),
    Tool(
        name = "stock_news",
        func= stock_news,
        description= "Fetches news. Input: stock name (str) or '大盤'. Returns: list of news."
    ),
```

# 04 程式說明介紹

- 3~5: 建立存放圖片的資料夾和圖片之名稱
- 8~9: 存兩種尺寸，一個是原始尺寸，一個是預覽尺寸
- 11~25: size_change函式主要功能是調整圖片之大小和存成JPG檔案

- 2: download_stock_data函式會下載資料並讓AI計算指標
- 3~7: 將今天設為最後一天，前365天設為第一天，如此就可以設定區間是1年
- 12~13: 利用openai_helper計算指標
- 16~20: 把AI生成出來的計算程式設為局部變數，並執行df

繪製圖片

```
[ ]  1 import plotly
     2 import kaleido
```

```
[ ]  1 image_dir = 'stock_technical_chart'
     2 if not os.path.exists(image_dir):
     3         os.makedirs(image_dir)
```

```
[ ]  1 from PIL import Image
     2
     3 temp_path = f"{image_dir}/temp.png"
     4 original_image_path = f"{image_dir}/stock_chart.jpg"    # 請需換為你的圖片路徑
     5 preview_image_path = f"{image_dir}/stock_chart_preview.jpg"    # 輸出的圖片檔案名
     6
     7 # 設定目標尺寸
     8 original_image_size = (1200, 800)
     9 preview_image_size = (240, 240)
    10
    11 def size_change():
    12     # 開啟圖片並調整大小
    13     try:
    14         with Image.open(temp_path) as img:
    15             # 如果圖片有透明度，先轉換為 RGB 模式
    16             if img.mode in ("RGBA", "P"):    # 檢查是否帶透明通道
    17                 img = img.convert("RGB")
    18
    19             # 調整大小
    20             original_image = img.resize(original_image_size, Image.Resampling.LANCZOS)
    21             preview_image = img.resize(preview_image_size, Image.Resampling.LANCZOS)
    22
    23             # 儲存圖片為 JPG 格式
    24             original_image.save(original_image_path, format="JPEG")
    25             preview_image.save(preview_image_path, format="JPEG")
    26     except Exception as e:
    27         print(f"處理圖片時發生錯誤：{e}")
```

```
[ ]  1 # 下載資料並讓 AI 計算指標
     2 def download_stock_data(stock_id, start=None, end=None, indicator='MACD'):
     3     stock_id = f"{stock_id}.tw"
     4     if not end:
     5         end = dt.date.today()
     6     if not start:
     7         start = end - dt.timedelta(days=365)
     8     # 從 yf 下載資料
     9     df = yf.download(stock_id, start=start, end=end).reset_index()
    10
    11     # AI 計算技術指標
    12     code_str = openai_helper(df, f"計算{indicator}")
    13     print(code_str)
    14
    15     # 將 exec 生成的 calculate 設為局部變數
    16     local_vars = {}
    17     exec(code_str, globals(), local_vars)
    18     calculate = local_vars['calculate']
    19
    20     df = calculate(df)
    21
    22     # 資料處理
    23     bk_df = df.reset_index()
    24     bk_df.index = bk_df["Date"].dt.strftime('%Y-%m-%d')
    25
    26     return bk_df
    27
```

# 04 程式說明介紹

- 最後來到LINE BOT這邊，58行上面和105行下面的程式基本上都在處理如何與LINE做連線不是我們的重點

- 62: 接收使用者訊息
- 65: 如果訊息裡面有”圖表”這兩個字就進入繪製圖表
- 70: 為了要提取訊息要先做分割文字，以空格做分隔
- 72~74: 前面是股票代號，空格後是技術指標
- 77~78: 呼叫繪圖函式並給她股票代號和技術指標
- 83~87: 回傳圖片
- 90~95: 假使使用者輸入錯誤格式跳出錯誤訊息
- 97~104: 如果沒有”圖表”這兩個字進入agent所以呼叫stock_agent，最後輸出回傳

```
58 def handle_message(event):
59     """
60     處理來自用戶的訊息事件。
61     """
62     user_message = event.message.text.strip()
63
64     try:
65         if "圖表" in user_message:
66             # 移除 "圖表"
67             user_message = user_message.replace("圖表", "").strip()
68
69             # 使用空格分開股票代號和技術指標
70             parts = user_message.split(" ", 1)  # 只分割第一次出現的空格
71
72             if len(parts) == 2:
73                 stock_number = parts[0]  # 股票代號
74                 indicator = parts[1]  # 技術指標
75
76                 #傳送股票代碼
77                 plotly_stock(stock_number, start='2022-01-01', end= None,
78                              indicator=indicator)  #生成圖片
79                 # 圖片的公開 URL
80                 image_url = f"{ngrok_url}/{original_image_path}"
81
82                 # 回應一張圖片
83                 image_message = ImageSendMessage(
84                     original_content_url=
85                         f"{ngrok_url}/{original_image_path}",  # 圖片完整尺寸 URL
86                     preview_image_url=
87                         f"{ngrok_url}/{preview_image_path}"  # 預覽圖片 URL
88                 )
89                 api.reply_message(event.reply_token, image_message)
90             else:
91                 reply_text = "格式錯誤，正確格式:2330(空格)13MA圖表"
92                 api.reply_message(
93                     event.reply_token,
94                     TextSendMessage(text=reply_text)
95                 )
96
97         else:  # 進入AI agent
98             print(user_message)
99             reply_text = stock_agent(user_message)
100            print(user_message)
101            api.reply_message(
102                event.reply_token,
103                TextSendMessage(text=reply_text)
104            )
105    except Exception as e:
```

# 05 功能測試說明

- 輸入:台積電是做什麼的

1.Agent使用” query_stock”取得對應的股號

```
{
  "action": "query_stock",
  "action_input": "台積電"
}
```

Observation: {'股名': '台積電', '股號': '2330.TW'}

2.Agent使用” stock_info”取得企業簡介

```
{
  "action": "stock_info",
  "action_input": "2330.TW"
}
```

Observation: {'公司名稱': 'Taiwan Semiconductor Manufacturing Company Limited', '現價': 1120.0, '企業簡介': 'Taiwan Semiconductor Manufacturing C

3.Agent分析取得的資料並輸出

Action:

```
{
  "action": "Final Answer",
  "action_input": "台積電是一家半導體製造公司，主要業務包括晶圓製造、封裝、測試和銷售積體電路和其他半導體元件。其產品廣泛應用於高性能計
}
```

> Finished chain.

'台積電是一家半導體製造公司，主要業務包括晶圓製造、封裝、測試和銷售積體電路和其他半導體元件。其產品廣泛應用於高性能計算、智能手機、物聯網

# 06 除錯過程說明

- **stock_price 函數**

  一開始設計的stock_price是兩個輸入 def stock_price(stock_id= “2330.TW", days = 10):
  分別輸入股號及所需的天數，但Agent都只會輸入單一字串，修改很多次Prompt都無法正確執行函數

  - **修改stock_price:** 將stock_price改成單一字串輸入，再使用 split 將參數分開

```
def stock_price(input_dict):
        stock_id, days = input_dict.split(",")
        days = int(days)
```

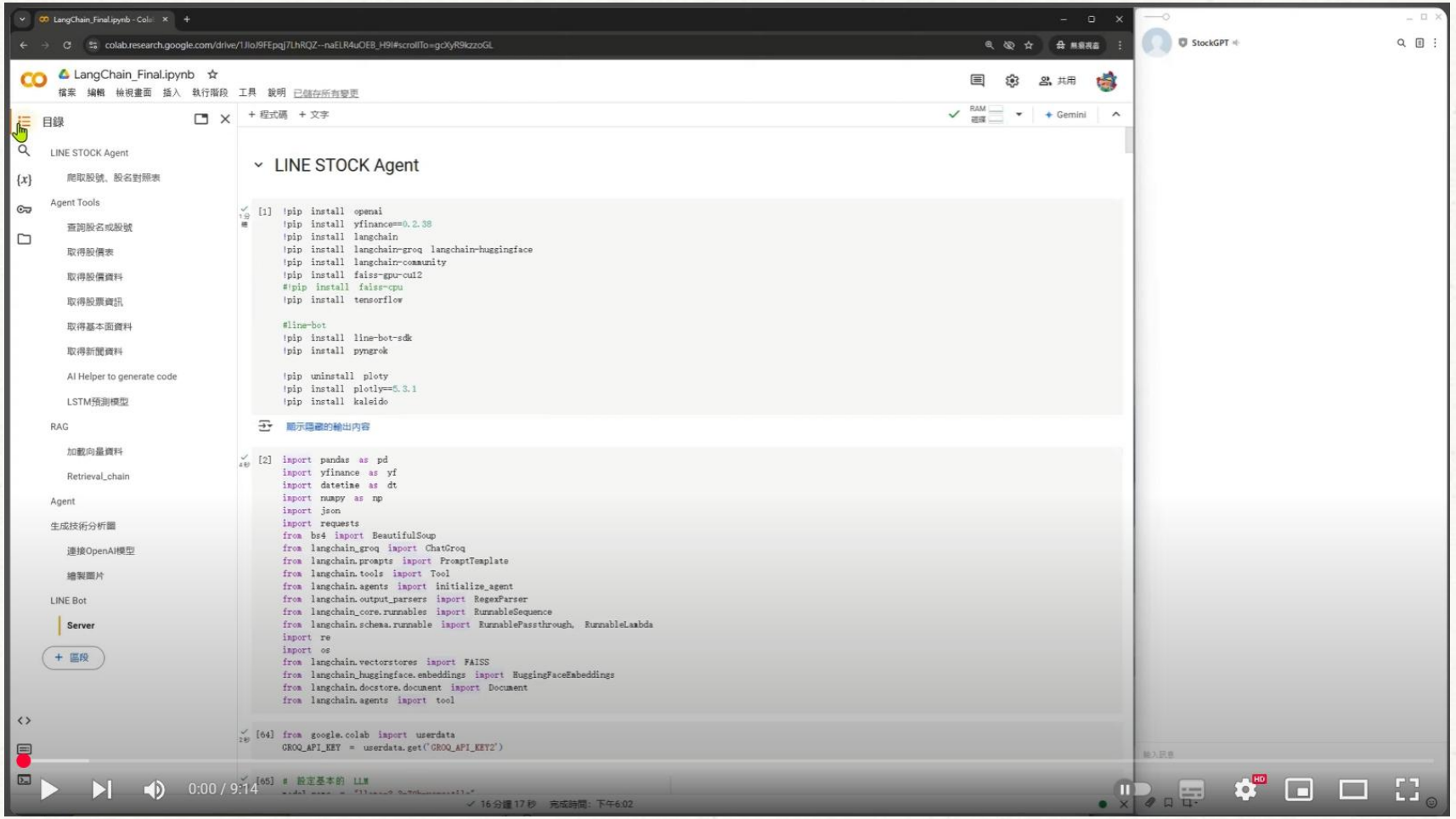
  - **修改Prompt及tool description :**使 Agent 可以固定輸入格式，使用逗號將兩個參數隔開

```
Action:
```
    {
        "action": "stock_price",
        "action_input": "2317.TW, 30"
    }
```
```

  成功執行函數

# 07 Demo影片

https://www.youtube.com/watch?v=wrj2J9nlBuE



# 08 結論及修課心得

我們最後成功做出了可以在LINE上使用的股票Agent，除了可以取得有關股票的資料外，還可以自動下載股票數據，並將股價轉換成需要的指標，畫出K線圖。我主要負責的部分是Stock Agent這個分支，從RAG的Vectordb到LSTM模型的訓練和將整體組合在一起的LangChain，這次的實作整合了非常多程式，學到很多東西。其中我覺得最困難的部分就是LangChain， LangChain算是一個比較新的東西，我看網路上的教學大部分都是示範官方的範例，而且他的結構很簡化有時候用一個(|)運算符號就把程式串在一起了，直接看程式碼都看不懂他在寫什麼。還好有GPT和Gemini2.0的幫助我才能順利實現我的目標。這次的實作也讓我知道什麼是Python的版本地獄，我只是把Colab重新啟動之前重來沒報錯的地方就報錯了，還好只要安裝別的數據包就修復了。

這學期的課程主要都圍繞在AI這一塊，使用的範例或模型都是較近期的，學到的都不是平常電機系課程會教的，我也感受到現在AI的強大與目前AI發展真的非常快速，從之前的生成式AI我都覺得這是很新的東西了，到現在已經是Agent的時代了。不學新東西的很快就要被淘汰了。

# 09 網址

Colab: https://colab.research.google.com/drive/1JIoJ9FEpqj7LhRQZ--naELR4uOEB_H9I#scrollTo=0aNxRglPlqoM

Youtube: https://www.youtube.com/watch?v=wrj2J9nlBuE

LSTM model: https://www.kaggle.com/code/steak0069/simple-lstm

RAG Vectordb: https://www.kaggle.com/code/steak0069/langchain-rag-stockbook