**幫我解釋程式碼**

好的，這段程式碼的目的是從臺灣證券交易所 (TWSE) 下載特定日期的股票交易資料，並將其處理成一個 pandas DataFrame。

以下是各個部分的解釋：

download\_twse*\_csv(date: str, type\_*: str = "ALL") -> str 函式：

這個函式負責從 TWSE 的網站下載 CSV 格式的每日交易資料。

date 參數是你想要下載資料的日期，格式為 YYYYMMDD。

type\_ 參數是股票的類型，預設是 "ALL" (所有股票)。

它建構一個 URL，然後使用 requests 函式庫發送 GET 請求。

response.encoding = 'cp950' 設定編碼以正確處理中文。

如果下載失敗 (HTTP 狀態碼不是 200)，會拋出一個例外。

成功下載後，會印出 "✅ Download complete" 並回傳原始的 CSV 文字內容。

extract\_csv\_lines(raw\_text: str) -> list 函式：

這個函式將原始的 CSV 文字內容分割成一行一行的列表。

它會移除空白行和不包含逗號的行 (這些通常是檔案開頭或結尾的說明文字)。

印出提取到的非空白行數，並回傳清理過的行列表。

find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list) 函式：

這個函式在清理過的行列表中尋找資料的起始位置。

它以 "證券代號" 作為關鍵字來尋找表頭 (header)。

找到表頭後，會將表頭行和其後的所有行 (資料行) 分開。

如果找不到表頭，會拋出一個例外。

印出找到表頭的行數，並回傳表頭行和資料行的列表。

clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame 函式：

這個函式負責將表頭行和資料行轉換成一個乾淨的 pandas DataFrame。

它使用 csv.reader 來正確解析 CSV 格式的行。

在處理資料行時，它會移除儲存格開頭的 =" 和結尾的 "，並去除前後的空白字元，以確保資料是乾淨的。

最後，使用清理過的資料建立一個 pandas DataFrame。

印出建立好的 DataFrame 的形狀 (列數和欄數)，並回傳 DataFrame。

範例使用部分 (# 🎯 Example usage)：

設定了一個範例日期 date = "20250702"。

依序呼叫上面定義的四個函式：下載原始文字、提取行、尋找資料區段、清理資料並建立 DataFrame。

最後使用 df.head() 顯示 DataFrame 的前幾行。

總結來說，這段程式碼提供了一個自動化下載和初步處理 TWSE 每日交易資料的流程。

**請使用python去完成 合併6/30～7/4的股票資料 每日收盤行情(全部) 成為一張資料表 (Data Frame) 使用 Colab 上傳 colab 分享連結 使用vs code 繳交 HackMD 連結，要包含程式碼及結果 請上傳 聊天機器人的對話紀錄分享連結 請使用df.info() 給整理完成 df 資訊**

**好的，這是一個多步驟的任務。我將為您制定一個計畫來完成這個任務，包括下載多天的資料、合併、以及顯示 DataFrame 的資訊。**

以下是計畫步驟：

下載多日資料: 迭代指定日期範圍，使用已定義的 download\_twse\_csv、extract\_csv\_lines、find\_data\_section 和 clean\_data 函式下載並處理每一天的資料。

合併資料: 將多天的 DataFrame 合併成一個單一的 DataFrame。

新增日期欄位: 在合併後的 DataFrame 中新增一個欄位，標示每筆資料的日期。

顯示 DataFrame 資訊: 使用 df.info() 顯示合併後 DataFrame 的資訊。

Finish task: 整理程式碼和結果，準備分享。

下載多日資料: 迭代指定日期範圍 (20240630 ~ 20240704)，使用已定義的函式下載並處理每一天的 TWSE 股票資料。

合併資料: 將下載的每日 DataFrame 儲存到一個列表中，然後使用 pandas 的 concat 函式將它們合併成一個單一的 DataFrame。

新增日期欄位: 在合併後的 DataFrame 中新增一個名為 '日期' 的欄位，用來記錄每筆資料的原始日期。

顯示 dataframe 資訊: 使用 df.info() 方法顯示合併後 DataFrame 的欄位資訊、資料類型和記憶體使用情況。

Finish task: 整理並提供完成後的程式碼和結果的分享連結。

生成中...

Preparing

下載多日資料

合併資料

新增日期欄位

顯示 dataframe 資訊

Finish task

**請說明這段程式碼：**

這行程式碼的作用是將原始下載的 CSV 文字 (raw\_text) 處理成一個乾淨的行列表 (lines)。

詳細解釋如下：

raw\_text.split('\n'): 這部分會將原始文字按照換行符號 (\n) 分割成一個列表，列表中的每個元素代表原始文字中的一行。

[... for line in ...]: 這是一個列表推導式 (list comprehension)，它會遍歷 raw\_text.split('\n') 生成的每一行。

line.strip(): 對每一行使用 .strip() 方法，這會移除該行開頭和結尾的空白字元 (包括空格、換行符、製表符等)。

if line.strip() and ',' in line: 這是一個過濾條件，只有滿足這個條件的行才會被包含在最終的 lines 列表中。

line.strip(): 檢查去除空白字元後的行是否為非空字串。這可以排除原始文字中的空白行。

',' in line: 檢查去除空白字元後的行是否包含逗號。TWSE 下載的 CSV 檔案在資料開始之前會有幾行說明文字，這些文字通常不包含逗號，通過這個條件可以過濾掉這些非資料行。

總之，這行程式碼有效地清除了原始 CSV 文字中的空白行和開頭的說明文字，只保留了看似是實際資料的行。

def find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list): 我想要知道這段函式程式碼的分解意義

好的，這個 find\_data\_section 函式的目的是在從 TWSE 下載並初步處理過的行列表 (lines) 中，找到實際的資料部分，並將表頭行和資料行分開。

以下是這段函式的詳細分解說明：

def find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list)::

定義了一個名為 find\_data\_section 的函式。

它接受兩個參數：

lines: list: 這是輸入的參數，預期是一個包含字串的列表，每一字串代表 CSV 檔案中的一行。

keyword: str = "證券代號": 這是另一個輸入參數，指定用來尋找資料區段起始的關鍵字。預設值是 "證券代號"，因為在 TWSE 的 CSV 檔案中，資料的表頭通常以這個詞開始。

-> (list, list): 這是函式的型別提示 (type hint)，表示這個函式預期會回傳一個包含兩個列表的元組 (tuple)，第一個列表是表頭行，第二個列表是資料行。

header\_idx = None:

初始化一個變數 header\_idx 為 None。這個變數將用來儲存找到表頭行的索引位置。

for idx, line in enumerate(lines)::

這是一個 for 迴圈，用於遍歷輸入的 lines 列表。

enumerate(lines) 會同時提供每個元素的索引 (idx) 和元素本身 (line)。

if keyword in line::

在迴圈的每次迭代中，檢查當前行 (line) 是否包含指定的 keyword (預設是 "證券代號")。

header\_idx = idx:

如果當前行包含關鍵字，則將當前行的索引 (idx) 賦值給 header\_idx。這表示我們找到了表頭行的位置。

break:

一旦找到表頭行，就使用 break 語句立即退出迴圈，因為我們只需要找到第一個包含關鍵字的行作為表頭。

if header\_idx is None::

在迴圈結束後，檢查 header\_idx 是否仍然是 None。

如果 header\_idx 是 None，表示在整個 lines 列表中都沒有找到包含指定關鍵字的行。

raise Exception("Could not find header row"):

如果在上一步檢查中發現沒有找到表頭行，則拋出一個 Exception，並附帶錯誤訊息 "Could not find header row"，告訴使用者無法定位資料區段。

header\_line = lines[header\_idx]:

如果找到了表頭行 (即 header\_idx 不是 None)，就使用 header\_idx 從 lines 列表中取出表頭行，並賦值給 header\_line 變數。

data\_lines = lines[header\_idx + 1:]:

使用切片 (slicing) 的方式，從 lines 列表中取出表頭行之後的所有行。header\_idx + 1 表示從表頭行的下一行開始，: 表示直到列表的末尾。這些行被賦值給 data\_lines 變數。

print(f"✅ Found header at line {header\_idx}"):

印出一條成功訊息，表明找到了表頭行，並顯示其索引位置。

return header\_line, data\_lines:

最後，函式回傳一個元組，其中包含 header\_line (單一個字串) 和 data\_lines (一個字串列表)。

總之，這個函式的核心功能就是通過尋找一個特定的關鍵字來自動定位 CSV 資料的起始點，並將表頭和實際的資料記錄分開，為後續的資料處理做準備。

**請說明這段程式碼：**

好的，這段程式碼定義了幾個函式，用於從臺灣證券交易所 (TWSE) 下載每日收盤行情數據，並將其處理成一個 pandas DataFrame。

以下是各個部分的詳細解釋：

Import 函式庫:

import requests: 導入 requests 函式庫，用於發送 HTTP 請求來下載網頁內容。

import csv: 導入 csv 函式庫，用於解析 CSV 格式的數據。

import pandas as pd: 導入 pandas 函式庫，並將其別名為 pd，用於數據處理和分析，特別是 DataFrame 的操作。

download\_twse*\_csv(date: str, type\_*: str = "ALL") -> str 函式:

功能: 下載指定日期和類型的 TWSE 每日 CSV 數據。

參數:

date: 字串，格式為 YYYYMMDD，表示要下載數據的日期。

type\_: 字串，表示股票類型，預設為 "ALL" (全部)。

回傳值: 字串，原始的 CSV 文本內容。

內部工作:

構建 TWSE 數據下載的 URL。

使用 requests.get() 發送 GET 請求下載數據。

設定響應編碼為 cp950，以正確處理中文。

檢查 HTTP 狀態碼，如果不是 200 (成功)，則拋出例外。

成功下載後，印出完成訊息並回傳數據文本。

extract\_csv\_lines(raw\_text: str) -> list 函式:

功能: 清理原始 CSV 文本，提取有效的數據行。

參數: raw\_text: 字串，download\_twse\_csv 下載的原始 CSV 文本。

回傳值: 列表，包含清理後的數據行字串。

內部工作:

將原始文本按換行符分割成行。

使用列表推導式過濾掉空白行和不包含逗號的行 (通常是檔案開頭的說明文字)。

印出提取到的非空白行數。

find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list) 函式:

功能: 在數據行列表中找到實際數據開始的位置 (表頭行)。

參數:

lines: 列表，extract\_csv\_lines 提取的數據行列表。

keyword: 字串，用於尋找表頭的關鍵字，預設為 "證券代號"。

回傳值: 元組，包含表頭行字串和資料行列表。

內部工作:

遍歷數據行，尋找包含指定關鍵字的行。

找到後，將該行視為表頭行，其後的行視為資料行。

如果找不到表頭行，則拋出例外。

印出找到表頭行的索引位置。

clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame 函式:

功能: 將表頭行和資料行轉換為乾淨的 pandas DataFrame。

參數:

header\_line: 字串，表頭行。

data\_lines: 列表，資料行。

回傳值: pandas DataFrame，包含清理後的數據。

內部工作:

使用 csv.reader 解析表頭行和資料行。

遍歷資料行，對每個單元格進行清理，移除 =", " 等可能影響數據類型的字元並去除前後空白。

使用清理後的數據和表頭創建一個 pandas DataFrame。

印出創建好的 DataFrame 的形狀。

範例使用部分:

設定了一個範例日期 date = "20250702"。

依序呼叫上述四個函式，下載、提取、尋找數據區段並清理數據，最終生成一個 pandas DataFrame 儲存到 df 變數中。

最後使用 df.head() 顯示 DataFrame 的前五行，方便快速查看數據內容。

總的來說，這段程式碼提供了一套完整的流程，用於自動化從 TWSE 網站獲取、清理和初步處理每日股票交易數據。

**import requests import csv import pandas as pd def download\_twse*\_csv(date: str, type\_*: str = "ALL") -> str: """ Download TWSE daily CSV data as text. :param date: Date string in YYYYMMDD :param type\_: TWSE type, default "ALL" :return: CSV raw text """ url = f"https://www.twse.com.tw/rwd/zh/afterTrading/MI*\_INDEX?date={date}&type={type\_*}&response=csv" response = requests.get(url) response.encoding = 'cp950' if response.status\_code != 200: raise Exception(f"Failed to download: {response.status\_code}") print("✅ Download complete") return response.text 幫我逐行解釋**

好的，這段程式碼定義了一個名為 download\_twse\_csv 的函式，用於從臺灣證券交易所 (TWSE) 的網站下載特定日期的每日收盤行情 CSV 數據。

以下是逐行解釋：

import requests: 這一行導入了 Python 的 requests 函式庫，這個函式庫是用來方便地發送 HTTP 請求的，例如從網路上獲取數據。

import csv: 這一行導入了 Python 的 csv 模組，這個模組提供了讀取和寫入 CSV (Comma Separated Values) 格式檔案的功能。雖然在這個特定的函式中沒有直接用到 csv 模組的功能，但在整個筆記本的其他部分可能會用到，例如解析下載下來的 CSV 數據。

import pandas as pd: 這一行導入了 pandas 函式庫，並將其別名為 pd。pandas 是一個強大的數據處理和分析工具，特別是它提供了 DataFrame 這個數據結構，非常適合處理表格型的數據。同樣地，雖然在這個函式本身沒有用到 pandas，但在後續處理下載數據時會用到。

def download\_twse*\_csv(date: str, type\_*: str = "ALL") -> str:: 這一行定義了一個函式，名為 download\_twse\_csv。

date: str: 指定函式接受一個名為 date 的參數，並且預期這個參數是一個字串 (string)。

type*\_: str = "ALL": 指定函式接受一個名為 type\_* 的參數，預期這個參數是一個字串，並且如果呼叫函式時沒有提供 type\_ 參數，它的預設值是 "ALL"。

-> str: 這是函式的型別提示 (type hint)，表示這個函式預期會回傳一個字串。

""" ... """: 這是一個多行字串，用作函式的說明文件 (docstring)。它解釋了函式的功能、參數和回傳值。

Download TWSE daily CSV data as text.: 說明函式的功能是下載 TWSE 每日 CSV 數據的文本內容。

:param date: Date string in YYYYMMDD: 說明 date 參數的用途和格式。

:param type*\_: TWSE type, default "ALL": 說明 type\_* 參數的用途和預設值。

:return: CSV raw text: 說明函式的回傳值是什麼。

url = f"https://www.twse.com.tw/rwd/zh/afterTrading/MI*\_INDEX?date={date}&type={type\_*}&response=csv": 這一行構建了要下載數據的網址 (URL)。它使用了 F-string (格式化字串字面值)，將 date 和 type\_ 參數的值嵌入到網址中。這是 TWSE 提供數據下載的特定網址格式。

response = requests.get(url): 這一行使用 requests.get() 函式向上面構建的 URL 發送一個 HTTP GET 請求。請求的結果 (包括狀態碼、回傳的數據等) 被儲存在 response 變數中。

response.encoding = 'cp950': 這一行設定了 response 的編碼為 cp950。這是因為 TWSE 的網站可能使用這種編碼來顯示中文，設定正確的編碼可以確保回傳的文本內容能夠被正確地解碼，避免亂碼。

if response.status\_code != 200:: 這一行檢查 HTTP 請求的狀態碼 (response.status\_code) 是否不等於 200。狀態碼 200 表示請求成功。如果狀態碼不是 200，表示下載過程中發生了問題 (例如頁面不存在、伺服器錯誤等)。

raise Exception(f"Failed to download: {response.status\_code}"): 如果狀態碼不等於 200，這一行就會拋出一個 Exception (例外或錯誤)。例外訊息會包含失敗的原因和 HTTP 狀態碼。這會中斷程式的執行並提示錯誤。

print("✅ Download complete"): 如果狀態碼是 200 (下載成功)，這一行會執行，並印出一個帶有勾號的訊息 "✅ Download complete"，表示下載已完成。

return response.text: 這一行回傳 response.text，也就是從網址下載到的原始文本內容。這個文本內容就是 CSV 格式的數據。

總結來說，download\_twse\_csv 函式的作用就是根據提供的日期和類型，從 TWSE 網站下載原始的 CSV 數據文本，並在下載成功後回傳該文本，若失敗則拋出錯誤。

import requests import csv import pandas as pd def download\_twse*\_csv(date: str, type\_*: str = "ALL") -> str: """ Download TWSE daily CSV data as text. :param date: Date string in YYYYMMDD :param type\_: TWSE type, default "ALL" :return: CSV raw text """ url = f"https://www.twse.com.tw/rwd/zh/afterTrading/MI*\_INDEX?date={date}&type={type\_*}&response=csv" response = requests.get(url) response.encoding = 'cp950' if response.status\_code != 200: raise Exception(f"Failed to download: {response.status\_code}") print("✅ Download complete") return response.text 再幫我逐行解釋程式碼

很抱歉，我發生錯誤，請再試一次。

**def extract\_csv\_lines(raw\_text: str) -> list: # Clean raw text into lines, removing empty and irrelevant ones. lines = [line.strip() for line in raw\_text.split('\n') if line.strip() and ',' in line] print(f"✅ Extracted {len(lines)} non-empty lines") return lines def find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list): # Find header row and data rows starting from the keyword. header\_idx = None for idx, line in enumerate(lines): if keyword in line: header\_idx = idx break if header\_idx is None: raise Exception("Could not find header row") header\_line = lines[header\_idx] data\_lines = lines[header\_idx + 1:] print(f"✅ Found header at line {header\_idx}") return header\_line, data\_lines 幫我逐行解釋程式碼**

好的，這段程式碼包含了兩個函式：extract\_csv\_lines 和 find\_data\_section。它們的作用是處理從 TWSE 下載的原始 CSV 文本，提取出實際的數據行，並找出數據的表頭和內容。

我們逐行解釋這兩個函式：

函式一：extract\_csv\_lines

def extract\_csv\_lines(raw\_text: str) -> list:: 這一行定義了一個名為 extract\_csv\_lines 的函式。

raw\_text: str: 指定函式接受一個名為 raw\_text 的參數，預期它是一個字串，代表原始的 CSV 文本內容。

-> list: 型別提示，表示函式預期回傳一個列表。

# Clean raw text into lines, removing empty and irrelevant ones.: 這是一行註解，說明這個函式的功能是將原始文本清理成行，移除空白和不相關的行。

lines = [line.strip() for raw\_line in raw\_text.split('\n') if raw\_line.strip() and ',' in raw\_line]: 這一行是核心的列表推導式 (list comprehension)，用於處理原始文本並生成清理後的行列表。

raw\_text.split('\n'): 將原始文本 raw\_text 按換行符 \n 分割成一個原始行的列表。

for raw\_line in ...: 遍歷這個原始行的列表，將每個原始行賦值給變數 raw\_line。

if raw\_line.strip() and ',' in raw\_line: 這是過濾條件。只有當原始行滿足這兩個條件時，才會對其進行處理並包含在最終的 lines 列表中：

raw\_line.strip(): 檢查去除前後空白字元後的 raw\_line 是否為非空字串。這會過濾掉完全空白的行。

',' in raw\_line: 檢查去除前後空白字元後的 raw\_line 是否包含逗號。這可以過濾掉 CSV 檔案開頭不包含數據的說明行。

line.strip(): 如果原始行滿足過濾條件，則對其再次使用 .strip() 移除前後的空白字元，並將結果作為一個元素添加到最終的 lines 列表中。 (注意這裡的變數名變成了 line，雖然在列表推導式內部可以和外部的 raw\_line 不同，但功能上是對滿足條件的 raw\_line 進行 strip() 操作)。

print(f"✅ Extracted {len(lines)} non-empty lines"): 這一行印出一條訊息，顯示成功提取了多少條非空白且包含逗號的行。len(lines) 獲取 lines 列表中的元素數量。

return lines: 函式回傳最終清理後的行列表 lines。

函式二：find\_data\_section

def find\_data\_section(lines: list, keyword: str = "證券代號") -> (list, list):: 這一行定義了一個名為 find\_data\_section 的函式。

lines: list: 參數，預期是一個列表，包含 extract\_csv\_lines 處理後的行字串。

keyword: str = "證券代號": 參數，預期是一個字串，用於尋找表頭的關鍵字，預設值為 "證券代號"。

-> (list, list): 型別提示，表示函式預期回傳一個包含兩個列表的元組，第一個列表是表頭行，第二個列表是資料行。 (注意，雖然型別提示是 (list, list)，但實際上回傳的是一個字串和一個列表的元組，這是一個小的不一致，但函式功能是正確的)。

# Find header row and data rows starting from the keyword.: 註解，說明函式功能是尋找表頭行和從關鍵字開始的資料行。

header\_idx = None: 初始化一個變數 header\_idx 為 None，用於儲存找到的表頭行的索引位置。

for idx, line in enumerate(lines):: 遍歷輸入的 lines 列表，同時獲取每個元素的索引 (idx) 和值 (line)。

if keyword in line:: 檢查當前行 line 是否包含指定的 keyword。

header\_idx = idx: 如果找到了包含關鍵字的行，將其索引 idx 賦值給 header\_idx。

break: 找到表頭行後立即退出迴圈，因為我們只需要第一個匹配的行作為表頭。

if header\_idx is None:: 檢查迴圈結束後 header\_idx 是否仍然是 None (表示沒有找到表頭行)。

raise Exception("Could not find header row"): 如果沒有找到表頭行，則拋出一個例外。

header\_line = lines[header\_idx]: 使用找到的索引 header\_idx 從 lines 列表中取出表頭行，儲存在 header\_line 變數中。

data\_lines = lines[header\_idx + 1:]: 使用切片操作，從 lines 列表中取出 header\_idx 後面的所有行，儲存在 data\_lines 列表中。這些是實際的數據行。

print(f"✅ Found header at line {header\_idx}"): 印出一條訊息，顯示找到表頭行的索引位置。

return header\_line, data\_lines: 函式回傳一個元組，包含表頭行字串 (header\_line) 和資料行列表 (data\_lines)。

這兩個函式協同工作，先清理原始文本，然後定位數據區段，為後續將數據轉換為 DataFrame 做準備。

def clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame: """ Convert header and data lines to a cleaned DataFrame. """ header = next(csv.reader([header\_line])) rows = list(csv.reader(data\_lines)) cleaned\_rows = [] for row in rows: cleaned\_row = [ cell.replace('="', '').replace('"', '').strip() if cell.startswith('="') else cell.replace('"', '').strip() for cell in row ] cleaned\_rows.append(cleaned\_row) df = pd.DataFrame(cleaned\_rows, columns=header) print(f"✅ Created DataFrame with shape {df.shape}") return df 幫我逐行解釋程式碼

好的，這段程式碼定義了一個名為 clean\_data 的函式，它的作用是將從 TWSE 下載並經過前兩步處理的表頭行和資料行列表，轉換成一個乾淨的 pandas DataFrame。

以下是逐行解釋：

def clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame:: 這一行定義了函式 clean\_data。

header\_line: str: 參數，預期是一個字串，代表表頭行。

data\_lines: list: 參數，預期是一個列表，每個元素是一個字串，代表資料行。

-> pd.DataFrame: 型別提示，表示函式預期回傳一個 pandas DataFrame 物件。

""" ... """: 函式的說明文件 (docstring)，說明函式的功能是將表頭和資料行轉換為清理過的 DataFrame。

header = next(csv.reader([header\_line])): 這一行處理表頭行。

[header\_line]: 將單個表頭行字串放入一個列表中。

csv.reader(...): 使用 csv.reader 來解析這個列表中的字串，它會根據 CSV 的規則 (例如逗號分隔、雙引號處理等) 將一行分割成欄位的列表。

next(...): csv.reader 回傳的是一個迭代器，next() 函式從這個迭代器中取出第一項，也就是解析後的表頭欄位列表。這個列表被賦值給 header 變數，作為 DataFrame 的欄位名稱。

rows = list(csv.reader(data\_lines)): 這一行處理資料行列表。

csv.reader(data\_lines): 使用 csv.reader 來解析 data\_lines 列表中的每一個字串。它會逐行解析，並根據 CSV 規則將每行分割成欄位的列表。

list(...): 將 csv.reader 回傳的迭代器轉換成一個列表。這個列表 rows 包含了所有資料行，每一行都是一個欄位值的列表。

cleaned\_rows = []: 初始化一個空的列表 cleaned\_rows，用於儲存清理後的資料行。

for row in rows:: 遍歷 rows 列表中每一行 (每個 row 是一個欄位值的列表)。

cleaned\_row = [...]: 這是一個列表推導式，用於處理當前行的每一個欄位。

for cell in row: 遍歷當前行 row 中的每一個欄位值 (cell)。

cell.replace('="', '').replace('"', '').strip(): 對每一個欄位值進行字串替換和去除空白操作。

cell.replace('="', ''): 替換掉開頭的 ="。TWSE 的數據中，有時候數值會被這樣包裹，例如 ="123.45"，這樣做是為了確保數據被視為文本而不是數字，但我們需要移除它以便後續轉換為數字。

.replace('"', ''): 替換掉所有的雙引號 "。

.strip(): 移除字串開頭和結尾的空白字元。

if cell.startswith('="') else cell.replace('"', '').strip(): 這是一個條件表達式。

如果 cell 以 =" 開頭，則執行 cell.replace('="', '').replace('"', '').strip()。

否則 (即 cell 不以 =" 開頭)，則執行 cell.replace('"', '').strip()。

這樣做的目的是確保無論數據是否以 =" 開頭，都能正確移除雙引號和空白字元。

處理後的每一個欄位值組成一個新的列表 cleaned\_row。

cleaned\_rows.append(cleaned\_row): 將處理好的 cleaned\_row (單一資料行) 添加到 cleaned\_rows 列表中。

df = pd.DataFrame(cleaned\_rows, columns=header): 使用清理後的資料行列表 cleaned\_rows 和表頭列表 header 來創建一個 pandas DataFrame，並將結果賦值給 df 變數。

print(f"✅ Created DataFrame with shape {df.shape}"): 印出一條訊息，顯示成功創建了 DataFrame，並顯示其形狀 (列數, 欄數)。df.shape 是一個元組，包含 DataFrame 的維度。

return df: 函式回傳最終創建並清理好的 pandas DataFrame。

總結來說，clean\_data 函式接收解析後的表頭和資料行，對每個數據單元格進行清理以移除不必要的字元，最後將這些清理後的數據組織成一個易於使用的 pandas DataFrame。

def find\_data\_section(lines: list, keyword: str = "證券代號") -> (str, list): # Find header row and data rows starting from the keyword. header\_idx = None for idx, line in enumerate(lines): if keyword in line: header\_idx = idx break if header\_idx is None: raise Exception("Could not find header row") header\_line = lines[header\_idx] data\_lines = lines[header\_idx + 1:] print(f"✅ Found header at line {header\_idx}") return header\_line, data\_lines 這段程式碼有問題嗎?

這段程式碼從功能上來看是沒有問題的，它成功地實現了從清理過的行列表中找到包含特定關鍵字的行作為表頭，並將其後的行作為資料行的目的。

然而，在型別提示 (type hint) 的部分有一點小小的不一致：

函式的定義是 def find\_data\_section(...) -> (str, list):，這表示函式預期回傳一個包含一個字串和一個列表的元組。

函式的回傳語句是 return header\_line, data\_lines，其中 header\_line 確實是一個字串，而 data\_lines 是一個列表。

所以，雖然型別提示寫的是 (list, list)，但實際回傳的型別是 (str, list)，這與函式定義上的型別提示 -> (str, list) 是一致的。

總結來說，這段程式碼的功能是正確的，並且型別提示也是正確的。沒有發現明顯的問題。

您是否有在執行這段程式碼時遇到任何錯誤或非預期的行為呢？如果您有遇到問題，請提供詳細的錯誤訊息或描述，我可以幫您進一步分析。

所以，雖然型別提示寫的是 (list, list) 我不太了解這句話的意思

抱歉，我的解釋可能造成了混淆。

您是對的，在我的前一個回答中，我說「雖然型別提示寫的是 (list, list)」，這句話是錯誤的。

正確的型別提示應該是 -> (str, list)，這表示函式預期回傳一個元組 (tuple)，這個元組包含兩個元素：第一個元素是一個字串 (str)，第二個元素是一個列表 (list)。

我之前的解釋中，錯誤地將 (str, list) 說成了 (list, list)。

所以，這段程式碼的型別提示 -> (str, list) 是正確的，它準確地描述了函式 return header\_line, data\_lines 所回傳的數據結構：header\_line 是一個字串，data\_lines 是一個列表，它們被組合成一個元組 (header\_line, data\_lines)。

感謝您的提醒，我為我的錯誤感到抱歉。

你可以讀取到我頁面的所有程式碼嗎?

是的，我可以讀取到您 Colab 頁面中所有的程式碼儲存格 (code cells)。

這有助於我理解您目前的進度、已定義的函式和變數，這樣我才能根據您的需求生成或修改程式碼。

接下來，我可以繼續執行之前規劃的步驟，來完成合併多日股票資料的任務。

好的，我們現在來合併多日股票資料 在合併後的表格，應該要多出日期的欄位，史資料做出區別

好的，我們來執行計畫中的前三個步驟：下載指定日期範圍內的每日資料，合併成一個 DataFrame，並新增一個日期欄位來區別不同日期的資料。

我會使用 2024 年 6 月 30 日至 2024 年 7 月 4 日這個日期範圍。

好的，資料已經成功下載並合併，並且加入了日期欄位。

接下來，我們將使用 df\_combined.info() 來顯示合併後 DataFrame 的資訊，包括欄位名稱、非空值數量、資料型別以及記憶體使用情況。這對了解資料的整體結構很有幫助。

好的，我們已經成功下載了 2024 年 6 月 30 日至 2024 年 7 月 4 日的臺灣證券交易所每日收盤行情資料，合併到一個 pandas DataFrame df\_combined 中，並且使用 df\_combined.info() 顯示了它的資訊。

根據您的要求，現在需要提供相關的分享連結：

Colab 分享連結: 您可以從瀏覽器頂部的分享按鈕獲取 Colab 筆記本的分享連結。請確保分享設定允許我查看。

HackMD 連結 (如果使用 VS Code): 如果您是在 VS Code 中完成的，請將程式碼和執行結果整理到 HackMD 文件中，並提供該文件的分享連結。

聊天機器人對話紀錄分享連結: 您可以在這個聊天介面中找到分享對話紀錄的選項，並提供相應的連結。

請提供這些連結以便我確認任務的最終完成狀態。

任務已完成。

from datetime import date, timedelta start\_date = date(2024, 6, 30) end\_date = date(2024, 7, 4) delta = timedelta(days=1) all\_dates = [] current\_date = start\_date while current\_date <= end\_date: all\_dates.append(current\_date.strftime("%Y%m%d")) current\_date += delta print(f"將處理以下日期：{all\_dates}") df\_list = [] for current\_date\_str in all\_dates: print(f"\n正在處理日期：{current\_date\_str}") try: raw\_text = download\_twse\_csv(current\_date\_str) lines = extract\_csv\_lines(raw\_text) header\_line, data\_lines = find\_data\_section(lines) df\_day = clean\_data(header\_line, data\_lines) # Add date column df\_day['日期'] = current\_date\_str df\_list.append(df\_day) except Exception as e: print(f"處理日期 {current\_date\_str} 時發生錯誤: {e}") # Concatenate all dataframes if df\_list: df\_combined = pd.concat(df\_list, ignore\_index=True) print("\n✅ 所有日期資料已合併。") else: df\_combined = pd.DataFrame() print("\n⚠️ 沒有成功下載任何日期資料，合併結果為空 DataFrame。") 請幫我逐行解釋

好的，這段程式碼負責處理下載 2024 年 6 月 30 日至 2024 年 7 月 4 日期間的 TWSE 每日數據，將其處理並合併到一個 DataFrame 中，並在過程中加入日期欄位。

以下是逐行解釋：

from datetime import date, timedelta: 這一行從 Python 的 datetime 模組中導入了 date 和 timedelta 這兩個物件。date 用於表示日期，timedelta 用於表示時間間隔，方便進行日期計算。

start\_date = date(2024, 6, 30): 創建一個 date 物件，表示開始日期 2024 年 6 月 30 日，並賦值給 start\_date 變數。

end\_date = date(2024, 7, 4): 創建一個 date 物件，表示結束日期 2024 年 7 月 4 日，並賦值給 end\_date 變數。

delta = timedelta(days=1): 創建一個 timedelta 物件，表示一天的時間間隔，並賦值給 delta 變數。

all\_dates = []: 初始化一個空的列表 all\_dates，用於儲存日期範圍內的每一個日期的字串 (格式為 YYYYMMDD)。

current\_date = start\_date: 將當前處理的日期初始化為 start\_date。

while current\_date <= end\_date:: 這是一個 while 迴圈，只要 current\_date 小於或等於 end\_date，就繼續執行迴圈內的程式碼。這用於遍歷從開始日期到結束日期的所有日期。

all\_dates.append(current\_date.strftime("%Y%m%d")): 在每次迴圈中，將當前日期 current\_date 格式化為 YYYYMMDD 格式的字串 (strftime("%Y%m%d"))，並添加到 all\_dates 列表中。

current\_date += delta: 將當前日期 current\_date 增加一天 (即加上 delta)，準備處理下一個日期。

print(f"將處理以下日期：{all\_dates}"): 在遍歷完所有日期後，印出將要處理的日期列表。

df\_list = []: 初始化一個空的列表 df\_list，用於儲存每個日期下載並處理後的 DataFrame。

for current\_date\_str in all\_dates:: 這是一個 for 迴圈，遍歷 all\_dates 列表中的每一個日期字串。

print(f"\n正在處理日期：{current\_date\_str}"): 在處理每個日期之前，印出當前正在處理的日期。

try:: 開始一個 try 區塊，用於捕獲在下載和處理數據過程中可能發生的錯誤。

raw\_text = download\_twse\_csv(current\_date\_str): 呼叫之前定義的 download\_twse\_csv 函式，下載當前日期的原始 CSV 文本。

lines = extract\_csv\_lines(raw\_text): 呼叫 extract\_csv\_lines 函式，清理原始文本並提取數據行。

header\_line, data\_lines = find\_data\_section(lines): 呼叫 find\_data\_section 函式，從清理後的行中找出表頭行和資料行。

df\_day = clean\_data(header\_line, data\_lines): 呼叫 clean\_data 函式，將表頭行和資料行轉換為一個 pandas DataFrame，並賦值給 df\_day 變數。

df\_day['日期'] = current\_date\_str: 在當前日期的 DataFrame df\_day 中新增一個名為 '日期' 的欄位，並將當前日期的字串賦值給這個欄位的每一列。這用於標示每一筆資料的來源日期。

df\_list.append(df\_day): 將處理好並新增了日期欄位的 df\_day 添加到 df\_list 列表中。

except Exception as e:: 如果在 try 區塊中發生任何類型的例外 (錯誤)，則執行 except 區塊中的程式碼。

print(f"處理日期 {current\_date\_str} 時發生錯誤: {e}"): 在發生錯誤時，印出錯誤訊息，包括發生錯誤的日期和錯誤的具體內容。這樣可以知道哪些日期的數據未能成功下載或處理。

# Concatenate all dataframes: 註解，說明接下來的程式碼用於合併所有 DataFrame。

if df\_list:: 檢查 df\_list 列表是否非空。如果非空，表示至少有一個日期的數據成功處理並添加到列表中。

df\_combined = pd.concat(df\_list, ignore\_index=True): 如果 df\_list 非空，使用 pd.concat() 函式將 df\_list 中的所有 DataFrame 合併成一個單一的 DataFrame。ignore\_index=True 參數會重新生成合併後 DataFrame 的索引。

print("\n✅ 所有日期資料已合併。"): 印出成功合併所有資料的訊息。

else:: 如果 df\_list 是空的 (表示沒有任何日期的數據成功下載或處理)。

df\_combined = pd.DataFrame(): 創建一個空的 pandas DataFrame，並賦值給 df\_combined。

print("\n⚠️ 沒有成功下載任何日期資料，合併結果為空 DataFrame。"): 印出警告訊息，表示沒有成功下載任何數據，合併結果為空 DataFrame。

這段程式碼通過迴圈處理多個日期，利用之前定義的輔助函式下載、清理和轉換數據，並將每日的數據合併到一個大的 DataFrame 中，同時記錄每筆數據的日期。

df.info() 是什麼意思

df.info() 是 pandas DataFrame 提供的一個非常有用的方法，它的作用是快速查看 DataFrame 的概況資訊。

當您在一個 DataFrame 物件上呼叫 info() 方法時，它會印出以下資訊：

DataFrame 的類別: 顯示這個物件是 pandas.core.frame.DataFrame 類型。

索引資訊 (Index): 顯示 DataFrame 的索引類型和範圍 (例如：RangeIndex: 155629 entries, 0 to 155628)。這表示您的 DataFrame 有 155629 列，索引從 0 到 155628。

欄位資訊 (Data columns):

欄位數量: 顯示 DataFrame 有多少個欄位 (例如：Data columns (total 18 columns):)。

欄位列表: 逐一列出每個欄位的資訊：

#: 欄位的序號 (從 0 開始)。

Column: 欄位的名稱。

Non-Null Count: 該欄位中非空值的數量。這可以幫助您快速了解哪些欄位包含缺失值 (Null values)。

Dtype: 該欄位的數據類型 (例如：object, int64, float64, datetime64 等)。object 通常表示該欄位包含混合的數據類型，或者主要包含字串。

數據類型統計: 顯示 DataFrame 中不同數據類型的欄位數量 (例如：dtypes: object(18))。

記憶體使用情況: 估計 DataFrame 佔用的記憶體大小 (例如：memory usage: 21.4+ MB)。這對於處理大型數據集時很有用。

簡而言之，df.info() 就像是 DataFrame 的一個「健康檢查報告」，讓您可以快速地了解數據的結構、 completeness (是否有缺失值) 和數據類型，而無需查看實際的數據內容。這在進行數據清洗和預處理之前非常重要。

from datetime import date, timedelta start\_date = date(2024, 6, 30) end\_date = date(2024, 7, 4) delta = timedelta(days=1) all\_dates = [] current\_date = start\_date while current\_date <= end\_date: all\_dates.append(current\_date.strftime("%Y%m%d")) current\_date += delta print(f"將處理以下日期：{all\_dates}") 我想要把日期這欄的資訊加在證券名稱後面

很抱歉，我發生錯誤，請再試一次。

from datetime import date, timedelta start\_date = date(2024, 6, 30) end\_date = date(2024, 7, 4) delta = timedelta(days=1) all\_dates = [] current\_date = start\_date while current\_date <= end\_date: all\_dates.append(current\_date.strftime("%Y%m%d")) current\_date += delta print(f"將處理以下日期：{all\_dates}") 我想要把日期這欄的資訊加在證券名稱後面

好的，您想要將 '日期' 欄位移動到 '證券名稱' 欄位的後面。我會修改合併資料的程式碼，在將每日資料加入列表之前，調整欄位的順序。

好的，我已經修改了程式碼，現在 '日期' 欄位會在處理每個日期的 DataFrame 時就被放到 '證券名稱' 欄位的後面。

接下來，我們再次執行 df\_combined.info() 來確認欄位的順序以及合併後 DataFrame 的資訊。

好的，我們已經成功下載了指定日期範圍內的 TWSE 每日收盤行情資料，合併到一個 DataFrame df\_combined 中，並且確保 '日期' 欄位在 '證券名稱' 欄位之後，最後也顯示了合併後 DataFrame 的資訊。

根據您的要求，現在需要提供相關的分享連結：

Colab 分享連結: 您可以從瀏覽器頂部的分享按鈕獲取 Colab 筆記本的分享連結。請確保分享設定允許我查看。

HackMD 連結 (如果使用 VS Code): 如果您是在 VS Code 中完成的，請將程式碼和執行結果整理到 HackMD 文件中，並提供該文件的分享連結。

聊天機器人對話紀錄分享連結: 您可以在這個聊天介面中找到分享對話紀錄的選項，並提供相應的連結。

請提供這些連結以便我確認任務的最終完成狀態。

任務已完成。

df\_list = [] for current\_date\_str in all\_dates: print(f"\n正在處理日期：{current\_date\_str}") try: raw\_text = download\_twse\_csv(current\_date\_str) lines = extract\_csv\_lines(raw\_text) header\_line, data\_lines = find\_data\_section(lines) df\_day = clean\_data(header\_line, data\_lines) df\_day['日期'] = current\_date\_str # Add date column cols = df\_day.columns.tolist() # Reorder columns to place '日期' after '證券名稱' cols.insert(cols.index('證券名稱') + 1, cols.pop(cols.index('日期'))) df\_day = df\_day[cols] df\_list.append(df\_day) except Exception as e: print(f"處理日期 {current\_date\_str} 時發生錯誤: {e}") 幫我解釋一下

好的，這段程式碼是您之前執行的合併多日資料迴圈中的核心部分，它負責處理每一天的資料：從下載、清理到加入日期並調整欄位順序。

以下是逐行解釋：

df\_list = []: 初始化一個空的列表 df\_list。這個列表將用來存放每一個成功處理的日期的 DataFrame。

for current\_date\_str in all\_dates:: 這是一個 for 迴圈，它會遍歷之前生成的 all\_dates 列表中的每一個日期字串 (例如 "20240630", "20240701" 等)。在每一次迴圈中，當前的日期字串會被賦值給 current\_date\_str。

print(f"\n正在處理日期：{current\_date\_str}"): 在開始處理每個日期之前，印出一個換行符 \n 和一條訊息，告訴使用者當前正在處理哪個日期的資料。

try:: 開始一個 try 區塊。這個區塊內的程式碼是我們預期可能會發生錯誤的地方 (例如下載失敗、解析失敗等)。如果 try 區塊中的任何程式碼拋出例外 (Exception)，程式執行會跳到對應的 except 區塊。

raw\_text = download\_twse\_csv(current\_date\_str): 呼叫之前定義好的 download\_twse\_csv 函式，使用當前的日期字串 current\_date\_str 下載原始的 CSV 文字內容。下載的結果儲存在 raw\_text 變數中。

lines = extract\_csv\_lines(raw\_text): 呼叫 extract\_csv\_lines 函式，將下載的原始文本 raw\_text 清理並分割成一個列表 lines，其中包含有效的數據行。

header\_line, data\_lines = find\_data\_section(lines): 呼叫 find\_data\_section 函式，從 lines 列表中找到表頭行 (header\_line) 和實際的資料行列表 (data\_lines)。

df\_day = clean\_data(header\_line, data\_lines): 呼叫 clean\_data 函式，使用找到的表頭行和資料行創建一個乾淨的 pandas DataFrame，並將結果賦值給 df\_day 變數。這個 DataFrame 包含了當前日期下載的所有股票交易數據。

df\_day['日期'] = current\_date\_str # Add date column: 在剛剛創建的 df\_day DataFrame 中新增一個名為 '日期' 的欄位。這個欄位的每一列都會被賦予當前的日期字串 current\_date\_str。這樣做是為了在合併多天的數據後，還能知道每一筆交易記錄是屬於哪一天的。

cols = df\_day.columns.tolist() # Reorder columns to place '日期' after '證券名稱': 獲取 df\_day DataFrame 的所有欄位名稱，將它們作為一個列表儲存在 cols 變數中。

cols.insert(cols.index('證券名稱') + 1, cols.pop(cols.index('日期'))): 這一行進行欄位順序的調整。

cols.index('證券名稱'): 找到 '證券名稱' 欄位在 cols 列表中的索引位置。

cols.index('證券名稱') + 1: 計算 '證券名稱' 欄位後面一個位置的索引。

cols.pop(cols.index('日期')): 從 cols 列表中移除 '日期' 欄位，並回傳被移除的元素 (也就是 '日期' 這個字串)。

cols.insert(..., ...): 將剛剛移除的 '日期' 字串插入到計算好的 '證券名稱' 欄位後面的位置。這樣就完成了將 '日期' 欄位移動到 '證券名稱' 後面的操作。

df\_day = df\_day[cols]: 使用重新排序後的欄位列表 cols 來選擇並重新排列 df\_day DataFrame 的欄位。這會生成一個新的 DataFrame，其中的欄位順序已經按照 cols 列表的要求排列。將這個新的 DataFrame 再次賦值給 df\_day。

df\_list.append(df\_day): 將處理好、加入了日期並調整好欄位順序的當前日期的 DataFrame df\_day 添加到 df\_list 列表中。

except Exception as e:: 如果在 try 區塊內的任何地方發生了例外 (錯誤)，程式執行會跳到這裡。變數 e 會儲存發生的例外物件，包含錯誤的詳細資訊。

print(f"處理日期 {current\_date\_str} 時發生錯誤: {e}"): 在發生錯誤時，印出一個錯誤訊息，說明在哪個日期處理時發生了什麼錯誤。

# Concatenate all dataframes: 這是一行註解，說明接下來的程式碼用於合併所有 DataFrame。

if df\_list:: 檢查 df\_list 列表是否非空。如果它包含任何 DataFrame，則條件為真。

df\_combined = pd.concat(df\_list, ignore\_index=True): 如果 df\_list 非空 (表示至少有一個日期的數據成功處理)，則使用 pd.concat() 函式將列表 df\_list 中的所有 DataFrame 按列方向 (預設是列方向，axis=0) 合併成一個大的 DataFrame。ignore\_index=True 會為合併後的 DataFrame 生成一個新的、從 0 開始的連續索引。合併結果儲存在 df\_combined 變數中。

print("\n✅ 所有日期資料已合併。"): 如果成功合併，印出完成訊息。

else:: 如果 df\_list 是空的 (表示所有日期的下載或處理都失敗了)。

df\_combined = pd.DataFrame(): 創建一個空的 pandas DataFrame，並賦值給 df\_combined。

print("\n⚠️ 沒有成功下載任何日期資料，合併結果為空 DataFrame。"): 印出警告訊息，說明沒有成功下載任何數據，合併結果是一個空的 DataFrame。

總而言之，這段程式碼通過一個迴圈自動處理指定日期範圍內的每一天數據，使用之前定義的函式完成下載、清理和解析，並在將數據加入最終合併列表之前，為每筆數據標記日期並調整欄位順序。最後，將所有成功處理的日期的 DataFrame 合併成一個大的 DataFrame。

哈囉，我想要把原本csv檔案裡，備註以後(含)的資料都剔除

好的，您想要移除 CSV 檔案中那些備註或說明性的資料列，只保留實際的交易數據。這些備註通常在檔案的尾部。

我會修改 clean\_data 函式，在轉換成 DataFrame 之前，過濾掉這些包含備註的資料行。一個簡單的方法是檢查資料行的第一個欄位 (證券代號) 是否只包含數字，因為實際的證券代號是數字，而備註行的第一個欄位通常是說明文字。

好的，我已經修改了 clean\_data 函式，讓它在處理資料時會過濾掉第一個欄位不是數字的行。

現在，我們需要重新執行下載和合併資料的程式碼，讓新的 clean\_data 函式生效。執行完成後，我們會再次使用 df\_combined.info() 來確認合併後的 DataFrame 中是否已經移除了那些備註行。

你已停止這則回覆

date = "20250702" raw\_text = download\_twse\_csv(date) lines = extract\_csv\_lines(raw\_text) header\_line, data\_lines = find\_data\_section(lines) df = clean\_data(header\_line, data\_lines) df.head().tail() 在這邊，我該如何同時輸出head()和tail()

好的，您想要在同一個輸出中看到 DataFrame 的開頭幾行 (head()) 和結尾幾行 (tail())。

使用 df.head().tail() 是不正確的，因為 df.head() 已經只包含開頭的幾行，再對這幾行取 tail() 可能會得到空結果或非預期的輸出。

在 Colab 中，您可以使用 display() 函式來顯示多個變數或表達式的輸出。

我將修改您的程式碼，使用 display() 來同時顯示 df.head() 和 df.tail()。

好的，現在您可以在同一個輸出中看到 DataFrame 的開頭和結尾了。

您之前也已經成功地將多日的股票資料合併到了 df\_combined 這個 DataFrame 中，並且修改了 clean\_data 函式來過濾掉備註行。

接下來，您可能想要查看一下合併後的 df\_combined 這個 DataFrame 的開頭和結尾，來確認資料的結構和內容是否符合預期。

好的，我們已經成功下載了 2024 年 6 月 30 日至 2024 年 7 月 4 日期間的臺灣證券交易所每日收盤行情資料，將其合併到一個乾淨的 pandas DataFrame df\_combined 中，並且移除了備註行。我們也查看了合併後 DataFrame 的開頭和結尾，以及它的資訊。

根據您最初的要求，現在需要提供相關的分享連結：

Colab 分享連結: 您可以從瀏覽器頂部的分享按鈕獲取 Colab 筆記本的分享連結。請確保分享設定允許我查看。

HackMD 連結 (如果使用 VS Code): 如果您是在 VS Code 中完成的，請將程式碼和執行結果整理到 HackMD 文件中，並提供該文件的分享連結。

聊天機器人對話紀錄分享連結: 您可以在這個聊天介面中找到分享對話紀錄的選項，並提供相應的連結。

請提供這些連結以便我確認任務的最終完成狀態。

如果您還有其他的分析或處理需求，也請隨時告訴我！

任務已完成。

為什麼最後印出的dataframe是從台灣水泥開始(股票)，而非證券?

您好，您看到的 DataFrame 輸出是從 '台泥' 開始，是因為 '台泥' (證券代號 1101) 是實際數據中的第一筆股票記錄。

在顯示 DataFrame 時，第一行通常是欄位名稱 (column headers)，例如 '證券代號'、'證券名稱'、'日期' 等。這些欄位名稱用來描述每一列數據代表的意義。

而緊隨其後的行才是實際的數據記錄 (data rows)。在您下載的 TWSE 每日收盤行情資料中，第一筆數據記錄的證券代號是 '1101'，對應的證券名稱是 '台泥'。

所以，您看到的輸出是正確的，它顯示了 DataFrame 的欄位名稱，以及從第一筆數據記錄 ('台泥') 開始的內容。'證券代號' 和 '證券名稱' 是欄位的標題，而不是第一筆數據本身。

希望這樣解釋有幫助！如果您還有其他問題，請隨時提出。

舊式我想要證券的資料一起讀入，所以在判斷形成乾淨的dataframe過程似乎出現問題

好的，我了解了。您是希望在生成乾淨的 DataFrame 時，不要過濾掉那些包含「證券」或其他非數字開頭的資料行，這樣您可以看到原始 CSV 中的所有內容，包括表頭和備註。

目前的 clean\_data 函式會檢查每一行的第一個欄位是否為數字，如果不是，就跳過該行。這就是為什麼您看不到「證券」等非股票數據的行。

為了達成您的目標，我會修改 clean\_data 函式，移除這個根據第一個欄位是否為數字來過濾行的邏輯。這樣，函式就會將所有從 data\_lines 傳入的行都嘗試轉換為 DataFrame 的一部分。

請注意，移除這個過濾後，DataFrame 中將會包含表頭以下的所有行，包括實際數據、備註和可能的其他非數據行。您可能需要後續步驟來處理或識別這些非數據行。

def clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame: """ Convert header and data lines to a cleaned DataFrame. Removes specific characters but includes all provided data lines. """ header = next(csv.reader([header\_line])) rows = list(csv.reader(data\_lines)) cleaned\_rows = [] for row in rows: # Removed filtering based on the first column being a digit cleaned\_row = [ cell.replace('="', '').replace('"', '').strip() if cell.startswith('="') else cell.replace('"', '').strip() for cell in row ] # Ensure the cleaned row has the same number of columns as the header if len(cleaned\_row) == len(header): cleaned\_rows.append(cleaned\_row) else: print(f"⚠️ Skipping row due to column count mismatch: {row}") # Optional: print skipped rows df = pd.DataFrame(cleaned\_rows, columns=header) print(f"✅ Created DataFrame with shape {df.shape}") return df 包我逐行解釋

好的，這段程式碼是修改後的 clean\_data 函式，它現在會將所有傳入的資料行都嘗試轉換為 DataFrame，同時進行字元清理。

以下是逐行解釋：

def clean\_data(header\_line: str, data\_lines: list) -> pd.DataFrame:: 這一行定義了函式 clean\_data，參數和回傳值型別提示與之前相同。

header\_line: str: 參數，預期是一個字串，代表表頭行。

data\_lines: list: 參數，預期是一個列表，每個元素是一個字串，代表資料行。

-> pd.DataFrame: 型別提示，表示函式預期回傳一個 pandas DataFrame 物件。

""" ... """: 函式的說明文件 (docstring)。更新後的說明指出函式會移除特定字元，但包含所有提供的資料行。

header = next(csv.reader([header\_line])): 這一行與之前相同，處理表頭行，使用 csv.reader 解析後，取出第一個結果作為 DataFrame 的欄位名稱列表，儲存在 header 變數中。

rows = list(csv.reader(data\_lines)): 這一行也與之前相同，使用 csv.reader 解析 data\_lines 列表中的每一個字串 (每一行)，將結果轉換為一個列表 rows。rows 是一個列表的列表，外層列表代表行，內層列表代表該行的欄位值。

cleaned\_rows = []: 初始化一個空的列表 cleaned\_rows，用於儲存清理後的資料行。

for row in rows:: 遍歷 rows 列表中每一行 (每個 row 是一個欄位值的列表)。

# Removed filtering based on the first column being a digit: 這是一行註解，說明之前根據第一個欄位是否為數字進行過濾的程式碼已經被移除。

cleaned\_row = [...]: 這是一個列表推導式，用於處理當前行的每一個欄位。

for cell in row: 遍歷當前行 row 中的每一個欄位值 (cell)。

cell.replace('="', '').replace('"', '').strip(): 對每一個欄位值進行字串替換和去除空白操作。

cell.replace('="', ''): 替換掉開頭的 ="。

.replace('"', ''): 替換掉所有的雙引號 "。

.strip(): 移除字串開頭和結尾的空白字元。

if cell.startswith('="') else cell.replace('"', '').strip(): 這是一個條件表達式，與之前相同，用於處理可能以 =" 開頭的欄位值，確保移除雙引號和空白。

處理後的每一個欄位值組成一個新的列表 cleaned\_row。

# Ensure the cleaned row has the same number of columns as the header: 註解，說明接下來的檢查是為了確保資料行的欄位數量與表頭一致。

if len(cleaned\_row) == len(header):: 檢查清理後的當前行 cleaned\_row 的欄位數量是否與表頭 header 的欄位數量相同。

cleaned\_rows.append(cleaned\_row): 如果欄位數量匹配，則將清理後的 cleaned\_row 添加到 cleaned\_rows 列表中。這些行將被用來創建 DataFrame。

else:: 如果欄位數量不匹配。

print(f"⚠️ Skipping row due to column count mismatch: {row}") # Optional: print skipped rows: 印出一條警告訊息，說明由於欄位數量不匹配而跳過了這一行，並顯示原始的 row 內容。這有助於除錯，了解哪些行因為格式問題被忽略了。

df = pd.DataFrame(cleaned\_rows, columns=header): 使用符合欄位數量匹配的 cleaned\_rows 列表和表頭列表 header 來創建一個 pandas DataFrame，並將結果賦值給 df 變數。

print(f"✅ Created DataFrame with shape {df.shape}"): 印出一條訊息，顯示成功創建了 DataFrame，並顯示其形狀 (列數, 欄數)。

return df: 函式回傳最終創建並清理好的 pandas DataFrame。

與之前版本的主要區別在於，這個版本不再檢查第一欄是否為數字來過濾行，而是嘗試處理所有傳入的 data\_lines，只會因為欄位數量與表頭不匹配而跳過某些行。這讓您可以將原始 CSV 中表頭後的所有行都讀入 DataFrame 中。

if len(cleaned\_row) == len(header): cleaned\_rows.append(cleaned\_row) else: #print(f"⚠️ Skipping row due to column count mismatch: {row}") # Optional: print skipped rows 如果不匹配就直接跳過，省去需要「移除這個過濾後，DataFrame 中將會包含表頭以下的所有行，包括實際數據、備註和可能的其他非數據行。您可能需要後續步驟來處理或識別這些非數據行。」

您說得很對！

修改後的 clean\_data 函式中保留了這個條件：

if len(cleaned\_row) == len(header):

    cleaned\_rows.append(cleaned\_row)

else:

    # Skipping row due to column count mismatch

    pass # 或 print 警告

這個 if len(cleaned\_row) == len(header): 的檢查確實是一種過濾。它會確保只有欄位數量與表頭一致的行才會被添加到 cleaned\_rows 列表中，進而包含在最終的 DataFrame 中。

對於 TWSE 提供的 CSV 檔案，許多位於資料區段之後的備註行或說明行，它們的結構可能與實際的數據行不同，導致欄位數量不匹配。因此，這個根據欄位數量進行的過濾，確實可以幫助您排除掉一部分格式不符的非數據行，包括一些備註。

這與之前基於「第一個欄位是否為數字」的過濾方式不同，但都能起到排除部分非數據行的作用。

您觀察得很準確，依靠欄位數量的匹配來過濾，在某些情況下，確實可以減少需要手動處理或識別的非數據行。

不過請注意，如果某些備註行或非數據行恰好也包含與表頭相同數量的欄位，那麼這個過濾條件就無法將它們排除掉。在這種情況下，您仍然需要在生成 DataFrame 後，根據其他特徵 (例如欄位的內容是否為數字或特定文字) 來進一步篩選或清理。

但對於大多數 TWSE 的備註行，欄位數量不匹配是一個常見的特徵，所以您現在的程式碼應該已經能排除掉大部分的備註了。

您的理解是正確的，依靠這個欄位數量匹配的過濾，可以省去處理所有表頭後行的步驟，只保留結構上與數據行更相似的行。

請上傳 聊天機器人的對話紀錄分享連結 我該如何上傳這份連天記錄連結?

好的，很高興為您提供聊天記錄的分享連結。

在 Google Colab 的聊天介面中，您應該可以在介面的上方或側邊找到一個選項來分享對話記錄。

通常這個選項會是一個按鈕或圖標，點擊後會生成一個分享連結，您可以複製這個連結提供給我。

具體的操作步驟可能會因為 Colab 介面的更新而略有不同，您可以尋找類似「分享」、「Share」、「連結」等字樣的按鈕或菜單選項。

如果您找到了分享選項並生成了連結，請直接將連結貼到聊天框中即可。