Python初級數據分析員證書

（六）數據分析 及可視化專案

13. 數據分析專案
Demo2 - Backtesting

Review

- Statistics
- Hypothesis testing
- Algebra
- Linear regression
- Propositional logic
- Python
- R
- SQL

- Pandas, NumPy, SciPy
- Data Visualization, Matplotlib, Seaborn, Plotly
- Dashboard Visualization, Business Intelligence
- Storytelling

Chapter Summary

- Backtesting
- df.apply() a function to DataFrame
- Nested np.where
- df.iterrows
- tqdm

Disclaimer 聲明

我們課程嘅的所有內容僅用于**一般性質嘅信息同教育目的**，不涉及任何特定個人或實體嘅任何情況。 **請勿將任何此類信息或材料解釋為投資、財務、專業或任何其他建議。**

本節數據處理，類同其他Time Series數據處理，或有共通點，學員可舉一反三。

What is Backtesting 回測

Backtesting 回測背後嘅總體思路是透過將其應用於歷史數據來評估使用一些啟發式，或技術指標構建的交易策略嘅性能。

Data and Indicators

財務數據喺啲庫中係自由開放嘅，例如 yFinance and Quandl.

熱門技術指標庫：TA-Lib and Pandas-ta, 提供超過150種分析方法。

在本章中，我哋將發現最簡單嘅一個：移動平均線回測。

Import data and add MA20 and MA60

讓我們由較小嘅數據集開始，添加MA20同MA60。

```
import pandas as pd
import numpy as np
import math
import yfinance as yf
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

df_cola = yf.download('KO', start='2022-04-08',
                      end='2023-01-01', auto_adjust=True)
```

```
[**********************100%**********************]  1 of 1 completed
```

```
for i in [20, 60]:
    df_cola[f"MA_{i}"] = df_cola["Close"].rolling(i).mean()
df_cola = df_cola.dropna()
df_cola.sample(3)
```

Date	Open	High	Low	Close	Volume	MA_20	MA_60
2022-07-08	61.660163	62.041686	61.513425	61.767773	11311500	60.424916	61.720191
2022-09-19	58.522863	59.222496	58.365200	59.114105	12596200	60.770209	61.498024

Plot a general technical chart with plotly

```
fig = make_subplots(rows=2, cols=1, shared_xaxes=True,
                    vertical_spacing=0.01, row_heights=[0.7, 0.3])
fig.add_trace(go.Candlestick(x=df_cola.index,
              open=df_cola['Open'], high=df_cola['High'],
              low=df_cola['Low'], close=df_cola['Close'],
              showlegend=True, name='Close Price'),row=1,col=1 )
fig.add_scatter(x=df_cola.index, y=df_cola['MA_20'], name='MA20')
fig.add_scatter(x=df_cola.index, y=df_cola['MA_60'], name='MA60')
fig.add_trace(go.Bar(x=df_cola.index, y=df_cola.Volume,
                     showlegend=True,name='Volume',
                    marker=dict(color='rgb(125,125,222)')),row=2,col=1)
fig.update_layout(title="CocaCola Share Price (Close) US$",
                  xaxis_rangeslider_visible=False)
fig.show()
```

Plot a general technical chart with plotly

Principal of MA indicator

正如大多數人已經知道的，MA20相對于慢速MA60嘅快速上漲係買入信號，而MA60相對于MA20嘅上漲係賣出信號。 通過這種策略，我們可以使用歷史數據進行回溯測試。

Write a simple function to raise signal

以下function函數將比較兩條平均線。

MA20的數值較高MA60 高，設定signal為 1；反之設定 signal為 - 1。

0表示他們相等。

```
1  def indicator(row): #fastMA, slowMA
2      if row.MA_20 > row.MA_60 :
3          signal = 1
4      elif row.MA_20 < row.MA_60 :
5          signal = -1
6      else:
7          signal = 0
8      return signal
```

Df.apply()

使用函數df.apply（ ）返回pd中嘅信號系列。 我哋可以睇到佢哋何時改變情況。 由-1到1，或由1到- 1。 但係我哋仍然需要比較前一個日期的指標。 因此，我哋仲要再走一步。

```
1 df_cola[['MA_20', 'MA_60']].apply(indicator, axis=1)
```

```
Date
2022-07-06   -1
2022-07-07   -1
2022-07-08   -1
2022-07-11   -1
2022-07-12   -1
             ..
2022-12-23    1
2022-12-27    1
2022-12-28    1
2022-12-29    1
2022-12-30    1
Length: 125, dtype: int64
```

Using nested np.where()

我哋使用np.where() 設置條件以發現更改shifting(previous) 指標. Indicator.shift(1) 表示昨天(T-1) 指標.

```
df_i = pd.DataFrame({"Indicator": df_cola[['MA_20', 'MA_60']].apply(indicator, axis=1) })
np.where((df_i.Indicator==1)&(df_i.Indicator.shift(1)==-1), "B",
        np.where((df_i.Indicator==-1)&(df_i.Indicator.shift(1)==1), "S", "Hold"))
```

```
array(['Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'B', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'S', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'B', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold'], dtype='<U4')
```

```
df_i['Action'] = np.where((df_i.Indicator==1)&(df_i.Indicator.shift(1)==-1), "B",
                 np.where((df_i.Indicator==-1)&(df_i.Indicator.shift(1)==1), "S", "Hold"))
```

Check the Action with chart

```
df_i.loc[df_i['Action']=='B']
```

Date	Indicator	Action
2022-07-26	1	B
2022-11-15	1	B

```
df_i.loc[df_i['Action']=='S']
```

Date	Indicator	Action
2022-09-14	-1	S

Simplify work flow 簡化流程

The Indicator and Action 條件邏輯可以簡化如下。我哋稍後可能會使用此功能嚟測試更多組合. 0 為 hold, 1 為buy, -1 為 sell.

```
fastma = f'MA_20'
slowma = f'MA_60'

def signal(df, fastma, slowma):
    return np.where((df[fastma]>df[slowma])&(df[fastma].shift(1)<df[slowma].shift(1)), 1,
            np.where((df[fastma]<df[slowma])&(df[fastma].shift(1)>df[slowma].shift(1)), -1, 0))
```

```
signal(df_cola, fastma, slowma).tolist()
```

```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
```

DF index warning

```
1 df_strategy = df_cola[['Close']]
```

```
1 df_strategy["Signal"] = signal(df_cola, fastma, slowma).tolist()
```

```
/var/folders/cq/_ztw83fd0vsg5vkyf8hn0zd40000gn/T/ipykernel_31379/2630632433.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

有時，當我們添加一個沒有索引的新列時，它會如上所述警告我們。這是因為 Pandas 混淆了如何正確添加新列。

我們只需要指定為原始 df.index 添加新的Series data.此外，我們還添加了另外兩列 NaN 以供進一步使用。

```
df_strategy.loc[df_strategy.index, ["Signal"]] = signal(df_cola, fastma, slowma).tolist()
```

```
df_strategy.loc[df_strategy.index,['Cash']] = np.NaN
df_strategy.loc[df_strategy.index,['Stock']] = np.NaN
```

```
df_strategy
```

Date	Close	Signal	Cash	Stock
2022-07-06	62.031906	0	NaN	NaN
2022-07-07	61.542770	0	NaN	NaN
2022-07-08	61.767773	0	NaN	NaN
2022-07-11	61.572117	0	NaN	NaN
2022-07-12	61.307983	0	NaN	NaN

用圖表檢查簡化的代碼。

```
# buy signal
df_strategy.loc[df_strategy['Signal']==1]
```

Date	Close	Signal	Cash	Stock
2022-07-26	61.836250	1	NaN	NaN
2022-11-15	59.744759	1	NaN	NaN

```
# sell signal
df_strategy.loc[df_strategy['Signal']==-1]
```

Date	Close	Signal	Cash	Stock
2022-09-14	59.468849	-1	NaN	NaN

避免直接使用loop over DF or Series

Pandas DF 是矢量化的，能夠快速實現。雖然我們需要處理數據行到行，但我們應該優先考慮使用：

1. Built-in function
2. df.iterrows, df.itertuples, or df.items
3. list comprehension

Backtest function

現在我們為我們的strategy_df編寫一個回測函數Backtest function。

還要設置一些變數。

```
def backtest(df, cash=1.0, comm_rate = 0.0, stock = 0.0, rf_rate = 0.01):
    """Backtest given strategy DF"""
    """row[0] is Close price; row[1] is Signal; """
    """Signal=1 means buy; Signal=-1 means sell """
    cash_init = cash
```

Using df.iterrows()

Write a for loop using df.iterrows(), to update buy/sell/hold of stock and cash columns.

```
5     cash_init = cash
6     for i, row  in df.iterrows():
7         # buy signal
8         if row[1]==1:
9             stock = (cash / row[0]) * (1 - comm_rate)
10            comm = cash / row[0] * comm_rate
11            cash = 0
12            df.at[i,'Cash'] = cash
13            df.at[i,'Stock'] = stock
14            print(f"{i}> Bought {stock:.4f}share @${row[0]:.4f}, Comm:{comm:.2f}, Cash Bal.:{cash:.4f}")
15
16        # sell signal
17        elif row[1]==-1:
18            cash = stock * row[0] * (1 - comm_rate)
19            comm = stock * row[0] * comm_rate
20            print(f"{i}> Sold {stock:.4f}share @${row[0]:.4f}, Comm:{comm:.2f}, Cash Bal.:{cash:.4f}")
21            stock = 0
22            df.at[i,'Cash'] = cash
23            df.at[i,'Stock'] = stock
24
25        # hold
26        else:
27            df.at[i,'Cash'] = cash
28            df.at[i,'Stock'] = stock
29
```

i is index, row is row in df

To avoid index warning, use df.at[ ] or df.iat[ ]

Print buy/sell record while signal raised

它可能在最後一天沒有觸發賣出信號。我們需要在到期日結束時強制清算股票兌現。

```
    # force cash out at the end term if still held stock
    if stock >= 0:
        cash = stock * df.iat[-1,0] * (1 - comm_rate)
        comm = stock * df.iat[-1,0] * comm_rate
        print(f"Force selling @${df.iat[-1,0]:.4f} at end term. Cash Bal.:{cash:.4f}")
        stock = 0
        df.at[i,'Cash'] = cash
        df.at[i,'Stock'] = stock

```

Print out summary

最後，我們列印出一份摘要，並返回一份關鍵指標的清單。

```
    # summary output
    simple_rtn = (cash - cash_init)/cash_init
    trading_times = df.loc[(df['Signal']==1) | (df['Signal']==-1)].shape[0]
    holding_days = df[df['Stock']>0].shape[0]
    total_days = df.shape[0]
    df.loc[df.index, ['Amount']] = [x[2] + x[3]*x[0] for x in np.array(df)]
    df.loc[df.index, ['Daily_rtn']] = df['Amount'].pct_change()
    std = df_strategy['Daily_rtn'].std()
    sharpe = (simple_rtn - rf_rate) / std / np.sqrt(total_days) * np.sqrt(250) # annualized sharpe ratio
    print(f"Sharpe:{sharpe:.4f}, Retrun:{simple_rtn*100:.4}%, SD:{std:.5f}")
    print(f"Transaction times:{trading_times}, hold {holding_days}days, total period:{total_days}days")

    return [sharpe, simple_rtn, std, trading_times, holding_days, total_days]
```

Run backtest

運行回測函數，我們得到以下結果。為了實現更多的組合，我們最後需要收集這些關鍵數位。

```
53 backtest(df_strategy)
```

```
2022-07-26 00:00:00> Bought 0.0162share @$61.8363, Comm:0.00, Cash Bal.:0.0000
2022-09-14 00:00:00> Sold 0.0162share @$59.4688, Comm:0.00, Cash Bal.:0.9617
2022-11-15 00:00:00> Bought 0.0161share @$59.7448, Comm:0.00, Cash Bal.:0.0000
Force selling @$63.1258 at end term. Cash Bal.:1.0161
Sharpe:1.2349, Retrun:1.614%, SD:0.00703
Transaction times:3, hold 66days, total period:125days

[1.2349446561111943, 0.016139718112793355, 0.007030981170931356, 3, 66, 125]
```

在我們最終更新后，df_strategy看起來像這樣。對於大量測試，應將其擦除。

```
1 df_strategy.sample(5)
```

Date	Close	Signal	Cash	Stock	Amount	Daily_rtn
2022-10-17	54.876884	0	0.961715	0.000000	0.961715	0.000000
2022-11-04	58.394756	0	0.961715	0.000000	0.961715	0.000000
2022-07-26	61.836250	1	0.000000	0.016172	1.000000	0.000000
2022-12-23	63.334194	0	0.000000	0.016097	1.019494	0.007578
2022-08-05	62.002560	0	0.000000	0.016172	1.002690	-0.004555

Mini Project – Backtest various of MA indicators

In previous slides, we learnt how to perform singles MA test (MA20 MA60) in a short period.

我們是否應該從MA20到MA100的20年價格歷史的組合進行進一步測試？

如果是這樣，您能否根據以前的函數準備偽代碼並起草代碼的結構？

創建更多MA檢驗組合

有數以千計的MA組合. 量化回測的好處是能夠逐個測試它們，並比較結果以進行未來預測。

We first define 3 constant for test.

- Minimum MA: min_MA = 20
- Maximum MA: max_MA = 100
- Interval between MA: interv = 10

  Interval here means the minimum range of two MA, we set 10, so there won't be a test between MA20 and MA25.

Pair up combinations

組合應如下所示：

MA20 to combine MA30,31,32,33,34…MA100

MA21 to combine MA31,32,33,34,35…MA100

…

…

…

MA88 to combine MA98,99,MA100

MA89 to combine MA99,MA100

MA90 to combine MA100

Brainstorm our idea in workflow

- Build function to 生成組合清單
- Build function to 創建信號到標籤 buy/sell/hold action
- 在清單中迴圈所有組合並執行buy/sell/hold test
- 記錄每對測試的每個結果，以便進一步分析

Pair up combination

編寫一個函數來配對所需的組合。有 2556 對 MA。

```python
def pairup(min_MA=20, max_MA=100, interv=10):
    counter = 0
    fast_MA, slow_MA = min_MA, min_MA+interv # initiate combination MAs
    comb_list = []

    for f in range(min_MA, max_MA-interv+1):
        for s in range(f+interv, max_MA+1):
            comb_list.append([f, s])
            slow_MA += 1
            counter += 1

        fast_MA += 1

    print(f"Total combinations: {counter}")
    return comb_list

comb_list = pairup()
```

No need to worry the non-DF for loop. It is fast.

```
Total combinations: 2556
```

Checking pairs

Check the first 7 and last 7 pairs.

```
1 comb_list[0:7]
```

```
[[20, 30], [20, 31], [20, 32], [20, 33], [20, 34], [20, 35], [20, 36]]
```

```
1 comb_list[-7:]
```

```
[[87, 100], [88, 98], [88, 99], [88, 100], [89, 99], [89, 100], [90, 100]]
```

```
1 len(comb_list)
```

```
2556
```

Prepare a larger DF

現在我們需要一個更大的DF 由更多歷史數據組成。

```
df_cola = yf.download('KO', start='2003-01-01',
                      end='2023-01-01', auto_adjust=True)

for i in range(20, 101):
    df_cola[f"MA_{i}"] = df_cola["Close"].rolling(i).mean()
df_cola = df_cola.dropna()
df_cola.sample(3)
```

```
[**********100%**********]  1 of 1 completed
```

Date	Open	High	Low	Close	Volume	MA_20	MA_21	MA_22	MA_23	MA_24
2008-08-20	17.229885	17.289898	17.024580	17.166714	11033200	16.971194	16.952679	16.919336	16.864861	16.820584
2004-07-28	12.449563	12.566233	12.389805	12.429644	25208600	13.994591	14.013026	14.027845	14.037788	14.058048
2016-11-14	33.400263	33.635759	33.172882	33.432743	19514200	34.147764	34.130343	34.117090	34.108167	34.100664

3 rows × 86 columns

```
df_cola.shape
```

```
(4936, 86)
```

tqdm

tqdm 是一個處理反覆運算進度的進度條庫。在運行大量數據時，您希望了解進度並確保系統沒有掛起。

```
pip install tqdm
```

```
1 from tqdm.notebook import tqdm_notebook
```

```
from tqdm.notebook import tqdm_notebook
import time
for i in tqdm_notebook(range(10)):
    time.sleep(0.5)
```

Prepare large testing

修訂了back_test功能

```
# Utilities functions

def back_test(df, cash=1.0, comm_rate = 0.0, stock = 0.0, rf_rate = 0.01):
    """Backtest given strategic DF"""
    """Signal=1 means buy; Signal=-1 means sell """
    """row[0] is Close price; row[1] is Signal; """
    cash_init = cash
    trading_times = 0
    for i, row in df.iterrows():
        # buy signal
        if (row['Signal']==1) and (cash>0):
            stock = (cash * (1 - comm_rate)) / row[0]
            comm = cash * comm_rate
            cash = 0
            df.at[i,'Cash'] = cash
            df.at[i,'Stock'] = stock
            df.at[i,'Comm'] = comm
            trading_times += 1
            #print(f"{i}> Bought {stock:.4f}share @${row[0]:.4f}, Comm:{comm:.2f}, Cash Bal.:{cash:.4f}")

```

Prepare large testing

修訂了back_test功能

```
21          # sell signal
22          elif (row['Signal']==-1) and (stock>0) :
23              cash = stock * row['Close'] * (1 - comm_rate)
24              comm = stock * row['Close'] * comm_rate
25              #print(f"{i}> Sold {stock:.4f}share @${row[0]:.4f}, Comm:{comm:.2f}, Cash Bal.:{cash:.4f}")
26              stock = 0
27              df.at[i,'Cash'] = cash
28              df.at[i,'Stock'] = stock
29              df.at[i,'Comm'] = comm
30              trading_times += 1
31
32          # hold
33          else:
34              df.at[i,'Cash'] = cash
35              df.at[i,'Stock'] = stock
36
37      # force cash out at the end term if still held stock
38      if stock > 0:
```

Prepare large testing

修訂了back_test功能

```
33          else:
34              df.at[i,'Cash'] = cash
35              df.at[i,'Stock'] = stock
36
37      # force cash out at the end term if still held stock
38      if stock > 0:
39          cash = stock * df.iat[-1,0] * (1 - comm_rate)
40          comm = stock * df.iat[-1,0] * comm_rate
41          #print(f"Force selling @${df.iat[-1,0]:.4f} at end term. Cash Bal.:{cash:.4f}")
42          stock = 0
43          df.at[i,'Cash'] = cash
44          df.at[i,'Stock'] = stock
45          df.at[i,'Comm'] = comm
46          trading_times += 1
47
```

Prepare large testing

修訂了back_test功能

```
47      # summary output
48      holding_days = df[df['Stock']>0].shape[0]
49      total_days = df.shape[0]
50      df.loc[df.index, ['Amount']] = [x[2] + x[3]*x[0] for x in np.array(df)]
51      df.loc[df.index, ['Daily_rtn']] = np.log(df['Amount']/df['Amount'].shift(1)) # daily log return
52      std = df['Daily_rtn'].tail(250).std()
53      comm = df['Comm'].sum()
54      simple_rtn = df['Amount'][-1]/df['Amount'][0] - 1
55      sharpe = ((df['Amount'][-1]/df['Amount'][-251]-1-rf_rate)/np.sqrt(250))/std  # twailing 250 days' sharpe ratio
56
57      return [sharpe, simple_rtn, std, trading_times, holding_days, total_days, comm]
58
```

Prepare large testing

創建 DF 以存儲測試結果

```
# create df to store each comparason data
result = pd.DataFrame(columns=['Pairs','Sharpe','Simple_rtn','STD','Trading_times',
                               'Holding_days','Total_days','Comm'])
result = result.set_index(['Pairs'])

```

Prepare large testing

用於測試的循環組合清單

```
for c in tqdm_notebook(comb_list):
    # create df for temp input signal data for comparason each time
    try:
        del df_strategic
    except:
        pass
    df_strategic = df_ko[['Close']]

    # run each pair of test and build buy/sell/hold signal
    df_strategic.loc[df_strategic.index, ["Signal"]] = signal(df_ko, c[0], c[1]).tolist()

    # set record for Cash and Stock
    df_strategic.loc[df_strategic.index,['Cash']] = np.NaN
    df_strategic.loc[df_strategic.index,['Stock']] = np.NaN
    df_strategic.loc[df_strategic.index,['Comm']] = 0

    # run test and get result
    ma_test = back_test(df_strategic)

    # keep result of sharpe, simple_rtn, std, trading_times, holding_days, total_days
    result.loc[f"MA_{c[0]} MA_{c[1]}"] = ma_test
```

Run the test

該測試可能需要大約一小時. 確保您的電腦保持處理模式。並運行代碼以另存為CSV.

```
21
22      # run test and get result
23      ma_test = back_test(df_strategic)
24
25      # keep result of sharpe, simple_rtn, std, trading_times, holding_days, total_days
26      result.loc[f"MA_{c[0]} MA_{c[1]}"] = ma_test
27
28  result.sample(5)
```

92% 2342/2556 [56:34<05:13, 1.47s/it]

Result

一個小時后，我們得到了這個結果 DF。

您應該先將其保存為 csv，然後可能還會備份結果。

如:

result_BAK = result

```
1 result.to_csv("result.csv")
```

```
1 result
```

Pairs	Sharpe	Simple_rtn	STD	Trading_times	Holding_days	Total_days	Comm
MA_20 MA_30	-2.054741	0.022897	0.009683	218.0	2916.0	4936.0	0.0
MA_20 MA_31	-2.519368	0.148676	0.009687	202.0	2934.0	4936.0	0.0
MA_20 MA_32	2.194858	0.102078	0.009887	188.0	2954.0	4936.0	0.0
MA_20 MA_33	3.365354	0.194135	0.009951	194.0	2967.0	4936.0	0.0
MA_20 MA_34	3.692148	0.286114	0.009982	176.0	2978.0	4936.0	0.0
...	...	...	...	...	...	...	...
MA_88 MA_99	8.915629	1.060845	0.010395	74.0	3290.0	4936.0	0.0
MA_88 MA_100	6.611618	1.143889	0.010287	68.0	3285.0	4936.0	0.0
MA_89 MA_99	6.376118	1.100504	0.010289	78.0	3295.0	4936.0	0.0
MA_89 MA_100	6.766223	1.083716	0.010288	76.0	3291.0	4936.0	0.0
MA_90 MA_100	6.838000	1.143076	0.010290	78.0	3297.0	4936.0	0.0

2556 rows × 7 columns

Analyse the result

```
1 result.nlargest(5, 'Sharpe')
```

Pairs	Sharpe	Simple_rtn	STD	Trading_times	Holding_days	Total_days	Comm
MA_85 MA_98	1.199657	1.644795	0.010513	72.0	3272.0	4936.0	0.0
MA_86 MA_97	1.085775	1.869401	0.010561	82.0	3279.0	4936.0	0.0

```
1 result.nlargest(5, 'Simple_rtn')
```

Pairs	Sharpe	Simple_rtn	STD	Trading_times	Holding_days	Total_days	Comm
MA_74 MA_85	0.286327	2.510830	0.010384	100.0	3158.0	4936.0	0.0
MA_72 MA_85	0.506195	2.440178	0.010326	84.0	3148.0	4936.0	0.0

注意：

- Sharpe & STD are trailing 250 days ratio.
- Simple_rtn is ratio for full period
- Trading_times is transaction times within test
- Holding_days is stock on-hand days
- Comm is commission fee, we may set rate in back_test()

```
1 result.nsmallest(5, 'Holding_days')
```

Pairs	Sharpe	Simple_rtn	STD	Trading_times	Holding_days	Total_days	Comm
MA_20 MA_30	-0.129047	0.022897	0.009751	218.0	2916.0	4936.0	0.0
MA_21 MA_31	-0.021034	0.117413	0.009858	204.0	2922.0	4936.0	0.0
MA_20 MA_31	-0.158230	0.148676	0.009755	202.0	2934.0	4936.0	0.0
MA_22 MA_32	-0.001489	0.038748	0.009854	204.0	2937.0	4936.0	0.0
MA_23 MA_33	0.158140	0.163594	0.010055	190.0	2951.0	4936.0	0.0

資產增長

20年資產（長揸）增長率為408.9%，而回測中的最佳回報只是251%. 策略需要調整。

```
1 print("Asset simple return: ", df_ko['Close'][-1]/df_ko['Close'][0]-1)
```

```
Asset simple return:  4.0891343237864675
```

```
1 df_ko.loc[df_ko.index, ['Daily_rtn']] = np.log(df_ko['Close']/df_ko['Close'].shift(1))
```

```
1 # twailing 250 days' KO's sharpe ratio
2 print("CocaCola 250Days Sharpe Ratio: ",
3       ((df_ko['Close'][-1]/df_ko['Close'][-251]-1)/np.sqrt(250))/df_ko['Daily_rtn'].tail(250).std() )
```

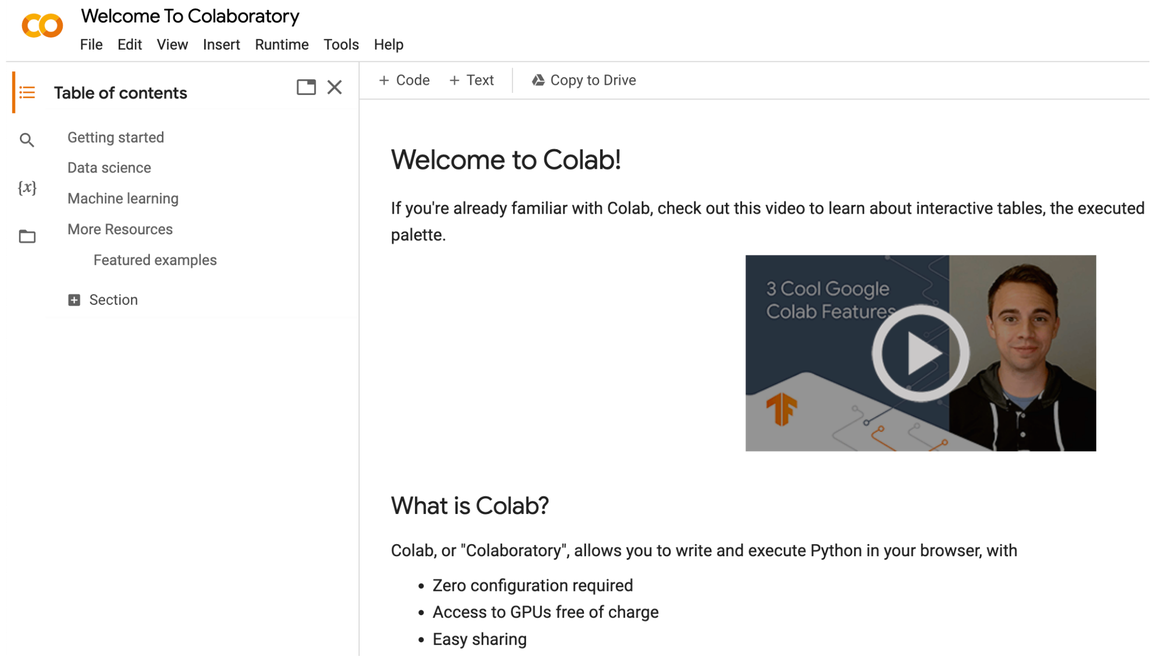
```
CocaCola 250Days Sharpe Ratio:  0.5293804803002438
```

根據結果，您能否在測試中列出更多批論，並指出一些修改方案？

Google Colab

Google Colab 是用於計算的雲基礎機器。您可以將代碼放在那裡運行以釋放桌面電腦記憶體資源。

https://colab.research.google.com/



Google Colab

The Colab computing speed might be faster than a 16GB RAM desktop. And it offer handy IDE environment for team programming.

```
    # set record for Cash and Stock
    df_strategic.loc[df_strategic.index,['Cash']] = np.NaN
    df_strategic.loc[df_strategic.index,['Stock']] = np.NaN
    df_strategic.loc[df_strategic.index,['Comm']] = 0

    # run test and get result
    ma_test = back_test(df_strategic)

    # keep result of sharpe, simple_rtn, std, trading_times, holding_days, total_days
    result.loc[f"MA_{c[0]} MA_{c[1]}"] = ma_test

result.sample(5)
```

```
100% 71/71 [00:00<00:00, 2257.17it/s]
Total combinations: 2556
62% 1581/2556 [19:46<13:28, 1.21it/s]
```

Academic Resources

Technical Analysis Lib - https://github.com/TA-Lib/ta-lib-python

Google Colab - https://colab.research.google.com/

Google Scholar - https://scholar.google.com/

Research Gate - https://www.researchgate.net/

JSTOR - https://www.jstor.org/

Research SPJ - https://spj.science.org/

APA 6th Edition - https://libguides.library.cityu.edu.hk/citing/apa