

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



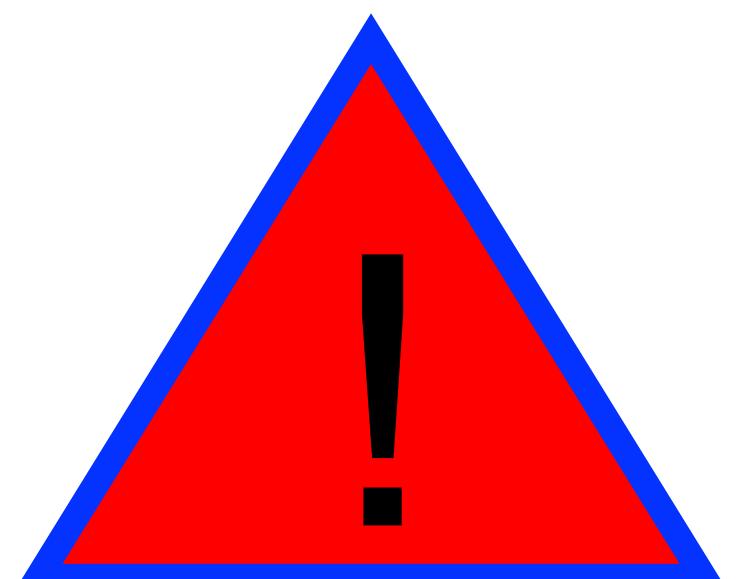
# 13. 數據分析專案 Data Analysis Project – Demo2

## Chapter Summary

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

# Disclaimer

All content on from our course is for informational and educational purposes of a general nature only, and does not address any circumstances of any particular individual or entity. Do not construe any such information or material as investment, financial, professional or any other advice.



# What is backtesting

The general idea behind **backtesting** is to evaluate the performance of a trading strategy—built using some heuristics or technical indicators—by applying it to historical data.

# Data and Indicators

Finance data is open freely in some library such as [yFinance](#) and [Quandl](#).

Popular Technical Indicator library: TA-Lib and Pandas-ta, provide more 150 kinds of analysis methods.

In this chapter, we will discover the most simple one: moving average backtesting.

# Import data and add MA20 and MA60

```

1 import pandas as pd
2 import numpy as np
3 import math
4 import yfinance as yf
5 import plotly.express as px
6 import plotly.graph_objects as go
7 from plotly.subplots import make_subplots
8
9 df_cola = yf.download('KO', start='2022-04-08',
10                         end='2023-01-01', auto_adjust=True)

```

Let's start with smaller dataset and add MA20 and MA60.

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

```

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

```

	Open	High	Low	Close	Volume	MA_20	MA_60
Date							
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

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

# Plot a general technical chart with plotly

CocaCola Share Price (Close) US\$



# Principal of MA indicator

As most of you already knew, the fast MA20 raise over slow MA60 is a buy signal, whereas MA60 raise over MA20 is a sell signal. With this strategy, we can backtest with historic data.

CocaCola Share Price (Close) US\$



# Write a simple function to raise signal

This function will compare the two MA. Return 1 of MA20 is higher, -1 of MA60 is higher. 0 means they are equal.

```
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()

Use function df.apply() to return the signal in pd.Series. We can see when they change situation. From -1 to 1, or from 1 to -1. BUT we still need to compare the previous date's indicators. So we shall do one more step.

```
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()

We use np.where() to set condition to discover the change from shifting(previous) indicators. Indicator.shift(1) means the yesterday(T-1) Indicator.

```

1 df_i = pd.DataFrame({"Indicator": df_col1[['MA_20', 'MA_60']].apply(indicator, axis=1) })
2 np.where((df_i.Indicator==1)&(df_i.Indicator.shift(1)==-1), "B",
3           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', 'S', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'B', 'Hold', 'Hold',
       'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold', 'Hold'],
      dtype='<U4')

```

```

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

```

CocaCola Share Price (Close) US\$

Check the Action with chart



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

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

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

	Indicator	Action
Date		
2022-09-14	-1	S

# Simplify work flow

The Indicator and Action condition logics could be simplified as follow. We may use this function to test many more combination later. 0 means hold, 1 buy, -1 sell.

```
1 fastma = f'MA_20'  
2 slowma = f'MA_60'  
3  
4 def signal(df, fastma, slowma):  
5     return np.where((df[fastma]>df[slowma])&(df[fastma].shift(1)<df[slowma].shift(1)), 1,  
6                     np.where((df[fastma]<df[slowma])&(df[fastma].shift(1)>df[slowma].shift(1)), -1, 0))
```

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

```
[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/_z.../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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

Sometimes while we add a new column without index, it warns us as the above. It is because Pandas confuse how properly to add the new column.

We just need to specify as the original df.index to add new Series data. Furthermore we add two more columns with NaN for further use.

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

```
1 df_strategy.loc[df_strategy.index, ['Cash']] = np.Nan  
2 df_strategy.loc[df_strategy.index, ['Stock']] = np.Nan
```

```
1 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

Check the simplified code with chart.

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

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

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

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



# Avoid directly use loop over DF or Series

Pandas DF is vectorized and able to implement fast. While we need to handle data rows to rows, we should prioritize to use:

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

# Backtest function

Now we write a backtest function for our strategy\_df.

Also set up some variables.

```
1 def backtest(df, cash=1.0, comm_rate = 0.0, stock = 0.0, rf_rate = 0.01):
2     """Backtest given strategy DF"""
3     """row[0] is Close price; row[1] is Signal; """
4     """Signal=1 means buy; Signal=-1 means sell """
5     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

It is possible that it didn't trigger sell signal at the period last day. We need to force liquidate stock to cash at the end of date.

```
--  
30 # force cash out at the end term if still held stock  
31 if stock >= 0:  
32     cash = stock * df.iat[-1,0] * (1 - comm_rate)  
33     comm = stock * df.iat[-1,0] * comm_rate  
34     print(f"Force selling @${df.iat[-1,0]:.4f} at end term. Cash Bal.:{cash:.4f}")  
35     stock = 0  
36     df.at[i,'Cash'] = cash  
37     df.at[i,'Stock'] = stock  
38
```

# Print out summary

Finally we print out a summary and return a list of key figures.

```
39 # summary output
40 simple_rtn = (cash - cash_init)/cash_init
41 trading_times = df.loc[(df['Signal']==1) | (df['Signal']==-1)].shape[0]
42 holding_days = df[df['Stock']>0].shape[0]
43 total_days = df.shape[0]
44 df.loc[df.index, ['Amount']] = [x[2] + x[3]*x[0] for x in np.array(df)]
45 df.loc[df.index, ['Daily rtn']] = df['Amount'].pct_change()
46 std = df_strategy['Daily rtn'].std()
47 sharpe = (simple_rtn - rf_rate) / std / np.sqrt(total_days) * np.sqrt(250) # annualized sharpe ratio
48 print(f"Sharpe:{sharpe:.4f}, Retrun:{simple_rtn*100:.4}%, SD:{std:.5f}")
49 print(f"Transaction times:{trading_times}, hold {holding_days}days, total period:{total_days}days")
50
51 return [sharpe, simple_rtn, std, trading_times, holding_days, total_days]
```

# Run backtest

Run backtest function, we get the following result. To implement more combinations, we need to gather those key numbers at last.

```
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]
```

The df\_strategy looks like this after our final update. It should be erased for large amount of testing.

```
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.

Shall we perform further test from combination of MA20 to MA100 of 20 years prices history?

If so, can you prepare pseudo code and draft the structure of the code, based on previous functions?

## Create more MA test combinations

There are thousands of combinations between each length of MA. The benefit of quant backtesting is able to test them one by one and compare the result for future forecasting.

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

The combinations should be like this:

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 generate a combination list
- Build function to create signal to label buy/sell/hold action
- Loop all combination in list and perform the buy/sell/hold test
- Record every result of each pair test for further analysis

# Pair up combination

Write a function to pair up the needed combination. There are 2556 pairs MA.

```
1 def pairup(min_MA=20, max_MA=100, interv=10):
2     counter = 0
3     fast_MA, slow_MA = min_MA, min_MA+interv # initiate combination MAs
4     comb_list = []
5
6     for f in range(min_MA, max_MA-interv+1):
7         for s in range(f+interv, max_MA+1):
8             comb_list.append([f, s])
9             slow_MA += 1
10            counter += 1
11
12            fast_MA += 1
13
14    print(f"Total combinations: {counter}")
15    return comb_list
16
17 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

```

1 df_cola = yf.download('KO', start='2003-01-01',
2                         end='2023-01-01', auto_adjust=True)
3
4 for i in range(20, 101):
5     df_cola[f"MA_{i}"] = df_cola["Close"].rolling(i).mean()
6 df_cola = df_cola.dropna()
7 df_cola.sample(3)

```

Now we need a larger  
DF consists of more  
historic data.

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

	Open	High	Low	Close	Volume	MA_20	MA_21	MA_22	MA_23	MA_24
Date										
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 x 86 columns

```

1 df_cola.shape
(4936, 86)

```

# tqdm

**tqdm** is a progress bar library that handle iteration progress. In running large amount of data, you would like to know the progress and make sure system is not hang.

```
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)
```

100%  10/10 [00:05<00:00, 1.93it/s]

# Prepare large testing

## Revised back\_test function

```
1 # Utilities functions
2
3 def back_test(df, cash=1.0, comm_rate = 0.0, stock = 0.0, rf_rate = 0.01):
4     """Backtest given strategic DF"""
5     """Signal=1 means buy; Signal=-1 means sell """
6     """row[0] is Close price; row[1] is Signal; """
7     cash_init = cash
8     trading_times = 0
9     for i, row in df.iterrows():
10         # buy signal
11         if (row['Signal']==1) and (cash>0):
12             stock = (cash * (1 - comm_rate)) / row[0]
13             comm = cash * comm_rate
14             cash = 0
15             df.at[i,'Cash'] = cash
16             df.at[i,'Stock'] = stock
17             df.at[i,'Comm'] = comm
18             trading_times += 1
19             #print(f"{i}> Bought {stock:.4f}share @${row[0]:.4f}, Comm:{comm:.2f}, Cash Bal.:{cash:.4f}")
20
```

# Prepare large testing

## Revised back\_test function

```
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

## Revised back\_test function

```
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

## Revised back\_test function

```
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

Create a DF to store testing result

```
1 # create df to store each comparason data
2 result = pd.DataFrame(columns=['Pairs', 'Sharpe', 'Simple_rtn', 'STD', 'Trading_times',
3                           'Holding_days', 'Total_days', 'Comm'])
4 result = result.set_index(['Pairs'])
5
```

# Prepare large testing

## Loop combination list for test

```
6  for c in tqdm_notebook(comb_list):
7      # create df for temp input signal data for comparason each time
8      try:
9          del df_strategic
10     except:
11         pass
12     df_strategic = df_ko[['Close']]
13
14     # run each pair of test and build buy/sell/hold signal
15     df_strategic.loc[df_strategic.index, ["Signal"]] = signal(df_ko, c[0], c[1]).tolist()
16
17     # set record for Cash and Stock
18     df_strategic.loc[df_strategic.index, ['Cash']] = np.NaN
19     df_strategic.loc[df_strategic.index, ['Stock']] = np.NaN
20     df_strategic.loc[df_strategic.index, ['Comm']] = 0
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
```

# Run the test

The test could take around **an hour**. Make sure your computer stay in processing mode. And run the code to **save as 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&lt;05:13, 1.47s/it]

# Result

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

After an hour, we have  
this result DF.

You should save it to csv  
first and probably make a  
backup copy of result.

Such as:  
`result_BAK = result`

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

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

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

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

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

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

## Note:

- 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()

# Asset growth

The asset growth rate in 20 years is 408.9%, whereas the best return in backtest is 251% only. The strategy need to be adjusted.

```
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() )
```

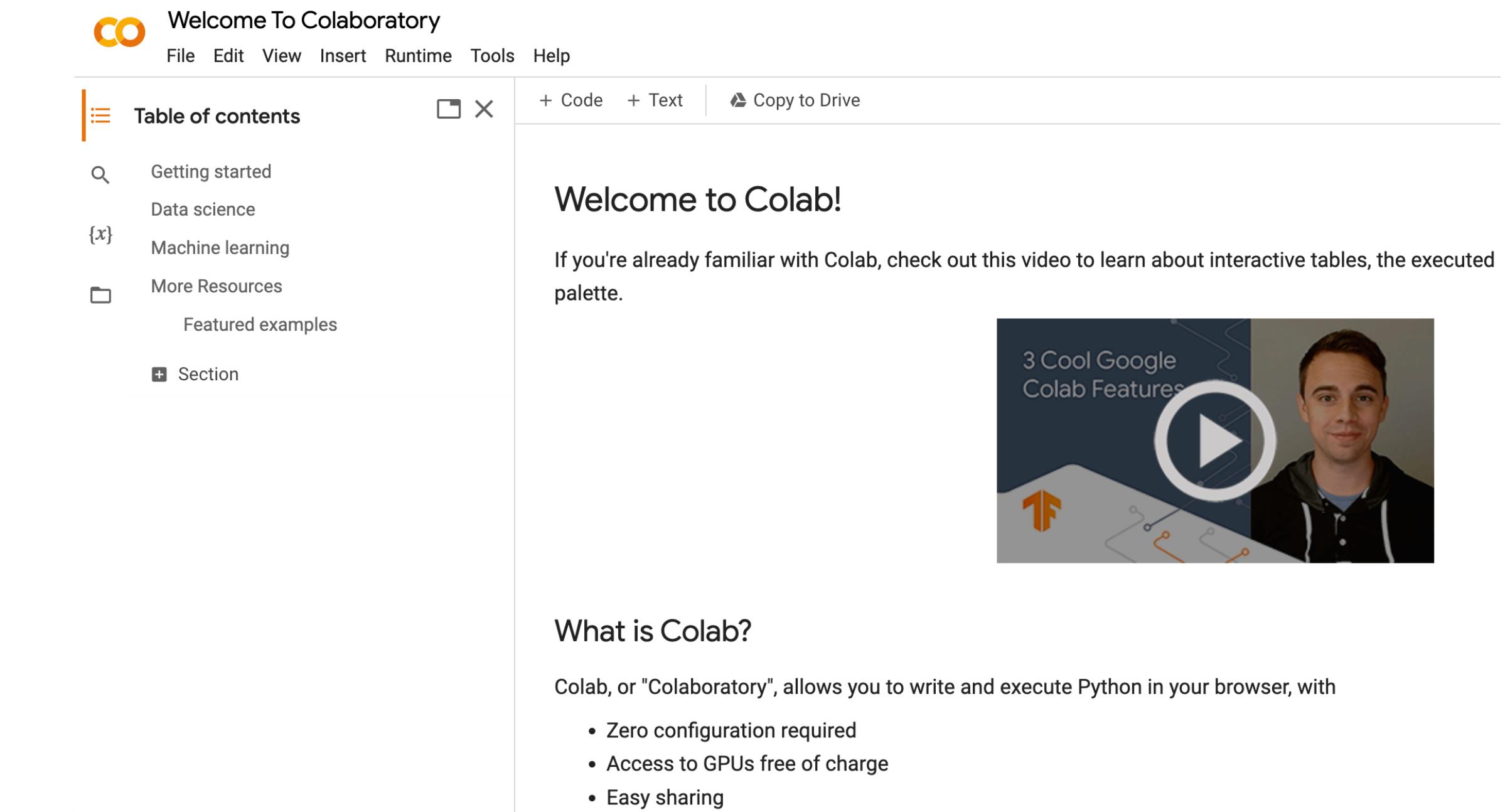
```
CocaCola 250Days Sharpe Ratio: 0.5293804803002438
```

Based on the result, could you list out more critiques in the testing and point out some modification?

# Google Colab

Google **Colab** is a cloud base machine for computing. You may place your code there running to free your local memory resource.

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



The screenshot shows the Google Colab interface. At the top, there's a navigation bar with the title "Welcome To Colaboratory" and menu items: File, Edit, View, Insert, Runtime, Tools, Help. Below the navigation bar is a toolbar with buttons for "+ Code", "+ Text", and "Copy to Drive". On the left, there's a sidebar titled "Table of contents" containing links to "Getting started", "Data science", "Machine learning", "More Resources", "Featured examples", and "Section". The main content area has a heading "Welcome to Colab!" and a sub-section "What is Colab?". It describes Colab as a platform for writing and executing Python in a browser, highlighting features like zero configuration, GPU access, and easy sharing. A video thumbnail titled "3 Cool Google Colab Features" is also visible.

# 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 6<sup>th</sup> Edition - <https://libguides.library.cityu.edu.hk/citing/apa>

