Strategy with 48%/year profit from the stock market

Tags

In addition to my main job in the field of Data, I am also an investor with over 2 years of experience in the Vietnamese stock market.
This time, I will apply my Data knowledge to test an investment strategy. As a result of this research, I identified a stock that aligns with the strategy I tested, yielding a total return of up to 370% and an average annual return of over 48%.

We will test the model on a single stock, then apply it to many stocks and find the best one.
We will then draw conclusions including the strengths and weaknesses of the model, and most importantly, how to use it in practice.

Let's begin my research.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import vnstock
from datetime import datetime
```

First import all the libraries I will be using

```
start_date = "2016-01-01"
end_date = "2023-08-18"
start_datetime = datetime.strptime(start_date, '%Y-%m-%d')
end_datetime = datetime.strptime(end_date, '%Y-%m-%d')
duration = (end_datetime - start_datetime).days
df = vnstock.stock_historical_data("ACB", start_date, end_date, "1D", 'stock')
df.head()
```

Here I use the ‘vnstock’ library, a free library that provides data on the Vietnamese stock market. I use the stock_historical_data function to get the price data of ACB (Asia Commercial Joint Stock Bank) shares for the period 2016-01-01 to 2023-08-18.

	time	open	high	low	close	volume
0	2016-01-04	4480	4480	4430	4450	36439
1	2016-01-05	4430	4430	4360	4390	47676
2	2016-01-06	4390	4410	4360	4410	40589
3	2016-01-07	4360	4390	4290	4320	94027
4	2016-01-08	4290	4320	4290	4320	84809

We will have a dataframe of the following form.

```
features = ['close']
df = df[features]
df.rename(columns={'close': 'price'}, inplace=True)
df.head()
```

For ease of analysis, I removed the time column and kept only the close column (the closing price of the day).

```
plt.figure(figsize=(12,6))
plt.plot(df['price'])
plt.grid(True)
plt.show()
```

Try plotting the line using matplotlib


```
def MA_value(day, index, df):
    MA_v = 0
    for i in range(index-day+1, index+1):
        MA_v = MA_v + df['price'][i]
    MA_v = MA_v/day
    return MA_v
```

I will create a function that calculates the MA value at any position on the price chart.

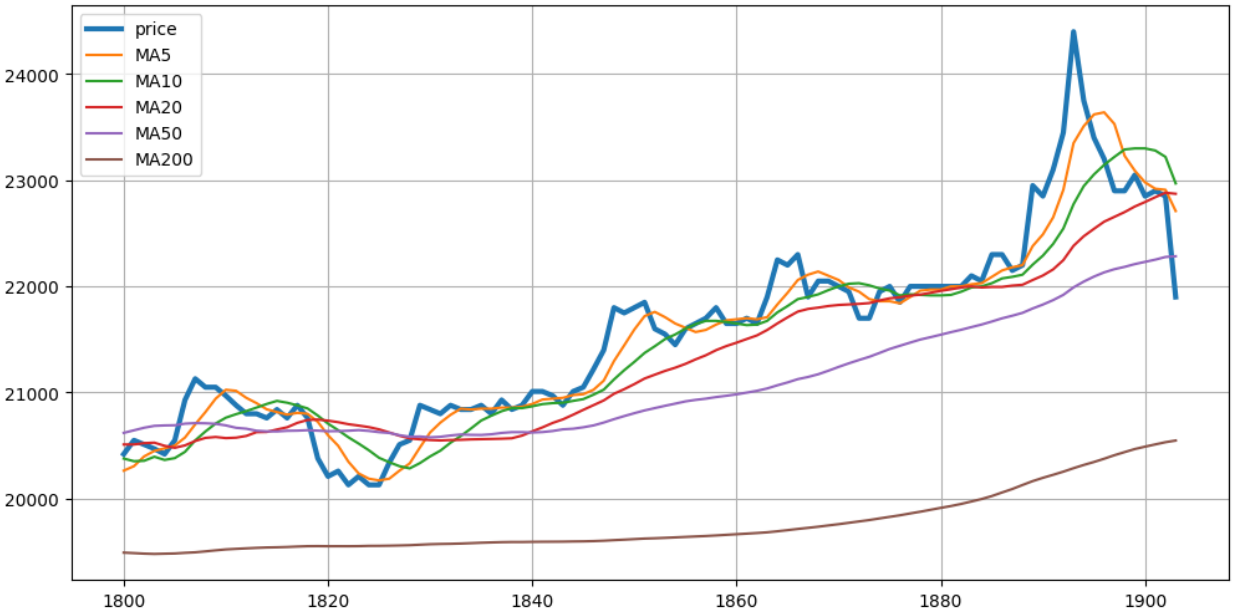
Explain: Moving Average (MA) is the average price line. For example, the MA20 at trading day X is equal to the average of the previous 20 trading days (including day X). Using a combination of MA lines such as MA20, MA50, or MA200 is of great significance in determining the stock's price trend.

```
MA_day_list = [5, 10, 20, 50, 200]
for MA_day in MA_day_list:
    df['MA'+str(MA_day)] = ""
    for index in range(len(df)):
        if index < MA_day:
            df['MA'+str(MA_day)][index] = 0
        else:
            df['MA'+str(MA_day)][index] = MA_value(MA_day, index, df)
```

I will add MA5, 10, 20, 50, and 200 (common MA types in investing) to the data. I add the condition index < MA_day so that if there is a case, the days are not reached enough to calculate the corresponding MA type. For example, this is only 50 days from the first day (in the data set), so MA100 or MA200 cannot be calculated.

```
plt.figure(figsize=(12,6))
plt.plot(df['price'][1800:], label='price', linewidth=3)
for MA_day in MA_day_list:
    plt.plot(df['MA'+str(MA_day)][1800:], label='MA'+str(MA_day))
plt.grid(True)
plt.legend()
plt.show()
```

Let's redraw the graph together. But since there are more MA lines this time, it will be difficult to see the entire data set, so I only plotted the closest few hundred rows for easy observation.



Our chart looks like this.

```
def check_condition_valid(conditions, row):
    for condition in conditions:
        if row['MA'+str(condition)] == 0:
            return False
    return True

def check_conditions_actived(conditions, row):
    for condition in conditions:
        if row['price'] < row['MA'+str(condition)]:
            return False
    return True

def check_conditions_unactived(conditions, row):
    for condition in conditions:
        if row['price'] > row['MA'+str(condition)]:
            return False
    return True
```

Here I created 3 functions, the first function check_condition_valid will ignore MA = 0 values when it encounters the non-computable MA values I explained above.

The second function is check_conditions_actived, it will return True when the conditions when the price is HIGHER than the moving averages I added to the ‘conditions’ array.

The last function is check_conditions_unactived, it will return True when the conditions of check_conditions_actived are no longer true. That is, the price will be LOWER than all the moving averages in the ‘conditions’ array.

```
command = pd.DataFrame({'buy_index':[], 'buy_price':[], 'sell_index':[], 'sell_price':[], 'profit_(%)':[], 'holding_days':[]})
```

Next, I create a new dataframe that stores the commands (transactions when the condition is activated).

```
conditions = [5, 10, 20, 50, 200]
for index, row in df.iterrows():
    if check_condition_valid(conditions, row) == False:
        continue
    elif check_conditions_actived(conditions, row):
        new_command = {'buy_index': index, 'buy_price': row['price']}
        command = pd.concat([command, pd.DataFrame([new_command])], ignore_index=True)
command.head()
```

As I said, I add MA5, 10, 20, 50, and 200 lines to the conditions array. In the loop, every time I realize the conditions are activated, I create a new command.

	buy_index	buy_price	sell_index	sell_price	profit_(%)	holding_days
0	200.0	4390.0	NaN	NaN	NaN	NaN
1	205.0	4390.0	NaN	NaN	NaN	NaN
2	222.0	4320.0	NaN	NaN	NaN	NaN
3	223.0	4360.0	NaN	NaN	NaN	NaN
4	224.0	4390.0	NaN	NaN	NaN	NaN

Our data looks like this, we are done with the buying task, and now we will handle the selling task.

```
command_index = 0
while command_index < len(command):
    for i in range(int(command.iloc[command_index]['buy_index']), len(df)):
        if check_conditions_unactived(conditions, df.iloc[i]):
            buy_price = command.iloc[command_index]['buy_price']
            sell_price = df.iloc[i]['price']
            command.loc[command_index, 'sell_index'] = i
            command.loc[command_index, 'sell_price'] = sell_price
            command.loc[command_index, 'profit_(%)'] = round((sell_price-buy_price)/buy_price*100, 2)
            command.loc[command_index, 'holding_days'] = i - command.iloc[command_index]['buy_index']
            break
    command_index = command_index + 1
command.head(10)
```

I use a while loop to process, every time the check_conditions_unactived function has a value of True, I will save the sell values like sell_index (similar to the sell date), sell_price, and calculate the profit (based on the difference between the buy and sell prices).), and finally holding_days (the number of days holding the stock from the date of purchase to the date of sale).

	buy_index	buy_price	sell_index	sell_price	profit_(%)	holding_days
0	200.0	4390.0	233.0	4170.0	-5.01	33.0
1	205.0	4390.0	233.0	4170.0	-5.01	28.0
2	222.0	4320.0	233.0	4170.0	-3.47	11.0
3	223.0	4360.0	233.0	4170.0	-4.36	10.0
4	224.0	4390.0	233.0	4170.0	-5.01	9.0
5	225.0	4390.0	233.0	4170.0	-5.01	8.0
6	229.0	4470.0	233.0	4170.0	-6.71	4.0
7	230.0	4390.0	233.0	4170.0	-5.01	3.0
8	243.0	4440.0	620.0	9920.0	123.42	377.0
9	244.0	4420.0	620.0	9920.0	124.43	376.0

The first 10 lines of the command should look like this. It is possible to see a purchase repeated over and over again. That is, when activating buy conditions, we only buy once at the signal and sell as soon as there is a sell signal, not buy continuously as above.

```
command = command.drop(command[command.duplicated(subset=['sell_index'])].index)
command = command.reset_index(drop=True)
command.head(20)
```

Handling this is quite simple, we just need to remove the rows that have the same sell_index.

	buy_index	buy_price	sell_index	sell_price	profit_(%)	holding_days
0	200.0	4390.0	233.0	4170.0	-5.01	33.0
1	243.0	4440.0	620.0	9920.0	123.42	377.0
2	652.0	10480.0	694.0	9800.0	-6.49	42.0
3	796.0	9920.0	811.0	9510.0	-4.13	15.0
4	814.0	9740.0	816.0	9550.0	-1.95	2.0
5	878.0	9550.0	893.0	9260.0	-3.04	15.0
6	927.0	9500.0	977.0	9410.0	-0.95	50.0
7	1009.0	9660.0	1018.0	9300.0	-3.73	9.0
8	1021.0	9870.0	1046.0	9090.0	-7.90	25.0
9	1096.0	9580.0	1140.0	8960.0	-6.47	44.0
10	1147.0	9870.0	1534.0	21990.0	122.80	387.0
11	1561.0	22620.0	1566.0	21920.0	-3.09	5.0
12	1753.0	19590.0	NaN	NaN	NaN	NaN

Everything is fine now. But in the last line, from buy_index of 1753 to the last line of the data there is still no sign of selling, so we will assume sell at the price on the last line of the data.

```
last_command_index = len(command) - 1 # we can't use -1 because it will create a new command with index -1
if pd.isna(command.iloc[last_command_index]['sell_index']):
    command.loc[last_command_index, 'sell_index'] = len(df) - 1
    command.loc[last_command_index, 'sell_price'] = df.iloc[-1]['price']
    command.loc[last_command_index, 'profit_(%)'] = round((df.iloc[-1]['price']-command.iloc[last_command_index]
['buy_price'])/command.iloc[last_command_index]['buy_price']*100, 2)
    command.loc[last_command_index, 'holding_days'] = len(df) - 1 - command.iloc[-1]['buy_index']
command.tail()
```

I edited the last line to match the selling hypothesis on the last line. Note that here we are not using index -1 to edit the last line of the ‘command’, because that will create an extra new line.

	buy_index	buy_price	sell_index	sell_price	profit_(%)	holding_days
8	1021.0	9870.0	1046.0	9090.0	-7.90	25.0
9	1096.0	9580.0	1140.0	8960.0	-6.47	44.0
10	1147.0	9870.0	1534.0	21990.0	122.80	387.0
11	1561.0	22620.0	1566.0	21920.0	-3.09	5.0
12	1753.0	19590.0	1903.0	21900.0	11.79	150.0

Everything was fine.

```
df.loc[1561:1566]
```

	price	MA5	MA10	MA20	MA50	MA200
1561	22620	22340.0	22114.0	22055.0	22552.0	22355.45
1562	22350	22332.0	22187.0	22080.0	22540.0	22347.6
1563	22490	22420.0	22274.0	22108.5	22524.0	22339.1
1564	22590	22500.0	22354.0	22142.0	22502.0	22330.6
1565	22420	22494.0	22387.0	22160.5	22484.0	22320.05
1566	21920	22354.0	22347.0	22160.5	22460.6	22307.85

We will try to test the transaction with buy_index of 1561 and sell_index of 1566 (the line has index 11 in the ‘command’). As can be seen in df, when the price crosses all the MA lines right at index 1561 and at index 1566 those conditions are rejected, and there is an immediate sell signal.

```
total_command = len(command)
total_holding_days = int(command['holding_days'].sum())
average_holding_days = round(total_holding_days/total_command, 2)
total_profit = round(command['profit_(%)'].sum(), 2)
average_profit = round(total_profit/total_command, 2)
profit_per_year = round(total_profit/(duration/365), 2)

print("Total of commands: ", total_command, "commands")
print("Total of holding days: ", total_holding_days, "days")
print("Average number of days holding a command: ", average_holding_days, "days")
print("Total profit (%): ", total_profit, "%")
print("Average profit (%) of a command: ", average_profit, "%")
print("Profit (%) per year: ", profit_per_year, '% per year')
```

Assuming every transaction we use the same amount of capital to buy, then the profit will be calculated by summarizing the profit column.

```
Total of commands:  13 commands
Total of holding days:  1154 days
Average number of days holding a command:  88.77 days
Total profit (%):  215.25 %
Average profit (%) of a command:  16.56 %
Profit (%) per year:  28.2 % per year
```

This is the result of our following this MA5, 10, 20, 50 and 200 based trading strategy. The annual return is calculated by taking the total return divided by the duration (distance from the first day to the last day in the data set). We have a total return of 215.25%, equivalent 28.2%/year.

For your comparison, the Vietnamese market in the same period increased by 106%, equivalent to 13.88%/year. I use https://www.tradingview.com/ website for calculation.

It's a high result, but is this the best stock for this tactic?
Each stock in the market often has its own "habit". This simply explains that for each stock, its price will tend to "follow" its own momentum. The implication of this is that there may still be many other stocks that use this strategy better than ACB.

Let's expand the "test zone"!

```
start_date = "2016-01-01"
end_date = "2023-08-18"
start_datetime = datetime.strptime(start_date, '%Y-%m-%d')
end_datetime = datetime.strptime(end_date, '%Y-%m-%d')
duration = (end_datetime - start_datetime).days
industry = vnstock.industry_analysis("ACB", lang='vi').columns
print(industry)
```

We will try with "competitors" in the same industry as ACB, ie the banking industry. This time we continue to use ‘vnstock’ to get the list of these stocks.

```
Index(['ACB', 'VCB', 'BID', 'CTG', 'VPB', 'TCB', 'MBB', 'SSB', 'STB', 'VIB',
       'HDB', 'SHB', 'TPB', 'EIB', 'LPB', 'MSB', 'OCB', 'NAB', 'BAB', 'ABB'],
      dtype='object', name='Mã CP')
```

What we need to do now is turn our code into a function that can be used many times.

```
def strategy_function(stock, start_date, end_date):
    df = vnstock.stock_historical_data(stock, start_date, end_date, "1D", 'stock')
    features = ['close']
    df = df[features]
    df.rename(columns={'close': 'price'}, inplace=True)

    MA_day_list = [5, 10, 20, 50, 200]
    for MA_day in MA_day_list:
        df['MA'+str(MA_day)] = ""
        for index in range(len(df)):
            if index < MA_day:
                df['MA'+str(MA_day)][index] = 0
            else:
                df['MA'+str(MA_day)][index] = MA_value(MA_day, index, df)

    command = pd.DataFrame({'buy_index':[], 'buy_price':[], 'sell_index':[], 'sell_price':[], 'profit_(%)':[], 'holding_days':[]})
    conditions = [5, 10, 20, 50, 200]
    for index, row in df.iterrows():
        if check_condition_valid(conditions, row) == False:
            continue
        elif check_conditions_actived(conditions, row):
            new_command = {'buy_index': index, 'buy_price': row['price']}
            command = pd.concat([command, pd.DataFrame([new_command])], ignore_index=True)

    command_index = 0
    while command_index < len(command):
        for i in range(int(command.iloc[command_index]['buy_index']), len(df)):
            if check_conditions_unactived(conditions, df.iloc[i]):
                buy_price = command.iloc[command_index]['buy_price']
                sell_price = df.iloc[i]['price']
                command.loc[command_index, 'sell_index'] = i
                command.loc[command_index, 'sell_price'] = sell_price
                command.loc[command_index, 'profit_(%)'] = round((sell_price-buy_price)/buy_price*100, 2)
                command.loc[command_index, 'holding_days'] = i - command.iloc[command_index]['buy_index']
                break
        command_index = command_index + 1

    command = command.drop(command[command.duplicated(subset=['sell_index'])].index)
    command = command.reset_index(drop=True)

    last_command_index = len(command) - 1 # we can't use -1 because it will create a new command with index -1
    if pd.isna(command.iloc[last_command_index]['sell_index']):
        command.loc[last_command_index, 'sell_index'] = len(df) - 1
        command.loc[last_command_index, 'sell_price'] = df.iloc[-1]['price']
        command.loc[last_command_index, 'profit_(%)'] = round((df.iloc[-1]['price']-command.iloc[last_command_index]
['buy_price'])/command.iloc[last_command_index]['buy_price']*100, 2)
        command.loc[last_command_index, 'holding_days'] = len(df) - 1 - command.iloc[-1]['buy_index']

    total_command = len(command)
    total_holding_days = int(command['holding_days'].sum())
    average_holding_days = round(total_holding_days/total_command, 2)
    total_profit = round(command['profit_(%)'].sum(), 2)
    average_profit = round(total_profit/total_command, 2)

    result = {'stock': stock, 'total_command': total_command, 'total_holding_days': total_holding_days, 'average_holding_days': average_holding_days,
'total_profit': total_profit, 'average_profit': average_profit}
    return result
```

We simply copy all the code and put it in a function that returns an array containing all the data like stock, total_command, total_holding_days, average_holding_days, total_profit and average_profit.

```
industry_df = pd.DataFrame({'stock': [], 'total_command': [], 'total_holding_days': [], 'average_holding_days': [], 'total_profit': []})
```

Similarly, we will also create industry_df to hold the data returned by our function.

```
for stock in industry:
    result = strategy_function(stock, start_date, end_date)
    industry_df = pd.concat([industry_df, pd.DataFrame([result])], ignore_index = True)

industry_df = industry_df.sort_values(by='total_profit', ascending=False)
industry_df = industry_df.reset_index(drop=True)
```

I use strategy_function to iterate over the list of stocks in 'industry' and store the returned data in 'industry_df'. Then sort descending based on profit.

	stock	total_command	total_holding_days	average_holding_days	total_profit	average_profit
0	SHB	6.0	890.0	148.33	318.93	53.16
1	VIB	10.0	883.0	88.30	277.99	27.80
2	LPB	10.0	656.0	65.60	255.61	25.56
3	MBB	13.0	1000.0	76.92	217.44	16.73
4	ACB	13.0	1154.0	88.77	215.25	16.56
5	TPB	6.0	772.0	128.67	169.29	28.22
6	STB	10.0	989.0	98.90	163.30	16.33
7	VPB	9.0	639.0	71.00	140.45	15.61
8	CTG	12.0	1037.0	86.42	121.23	10.10
9	TCB	8.0	513.0	64.12	103.55	12.94
10	HDB	9.0	634.0	70.44	79.98	8.89
11	VCB	18.0	1207.0	67.06	79.51	4.42
12	BID	16.0	1157.0	72.31	68.12	4.26
13	EIB	17.0	1084.0	63.76	54.41	3.20
14	NAB	2.0	311.0	155.50	41.03	20.52
15	ABB	2.0	185.0	92.50	10.17	5.08
16	BAB	13.0	558.0	42.92	9.37	0.72
17	MSB	2.0	175.0	87.50	9.10	4.55
18	OCB	4.0	191.0	47.75	-7.32	-1.83
19	SSB	6.0	282.0	47.00	-9.39	-1.57

Here are our results. It can be seen that SHB is the best stock in the Banking group when using our strategy. But let's visualize the data to make things easier to observe.

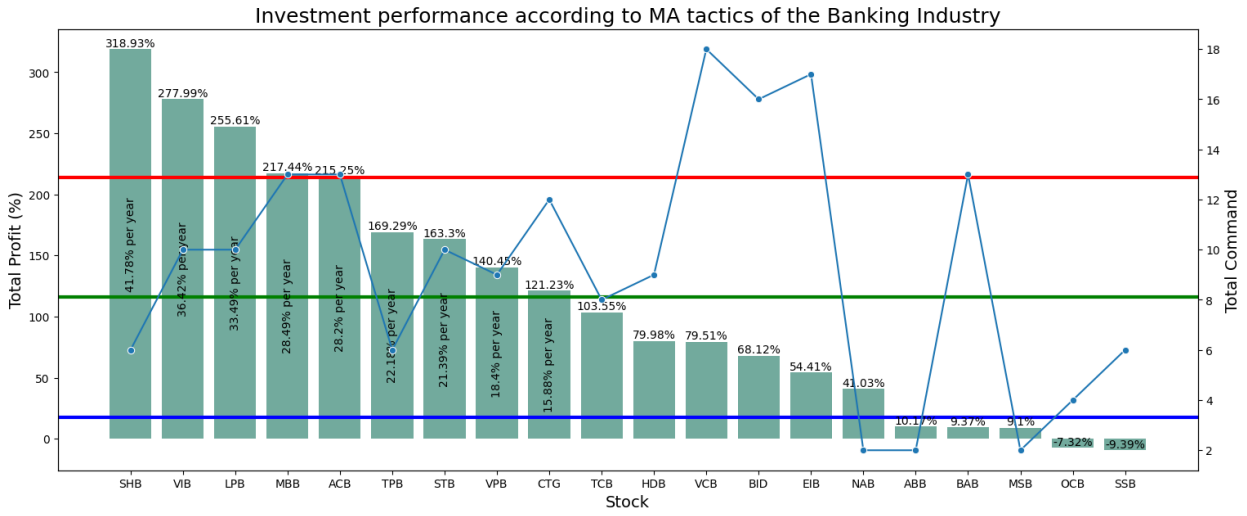
```
def high_low(industry_df, feature):
    np_array = np.array(industry_df[feature].values)
    mean = round(np.mean(np_array), 2)
    high = round(mean + np.std(np_array), 2)
    low = round(mean - np.std(np_array), 2)
    return [high, mean, low]
```

First, we need to create a function that returns High, Mean, and Low values. High will be calculated as Mean +Standard Deviation and vice versa Low = Mean - Standard Deviation. This is useful for comparing samples. Help assess how high the profit is.

```
fig, ax1 = plt.subplots(figsize=(18, 7))

sns.barplot(data=industry_df, x='stock', y='total_profit', color='#69b3a2')
ax1.set_xlabel('Stock', fontsize=14)
ax1.set_ylabel('Total Profit (%)', fontsize=14)

for index, row in industry_df.iterrows():
    plt.text(index, row['total_profit'], str(row['total_profit']) + '%', ha='center', va='bottom')
    if index < 9:
        plt.text(index, row['total_profit'] / 2, str(round(row['total_profit']/(duration/365), 2)) + '% per year', rotation='vertical', ha='center',
va='center')
ax2 = ax1.twinx()
sns.lineplot(data=industry_df, x='stock', y='total_command', marker='o', color='tab:blue', ax=ax2)
ax2.set_ylabel('Total Command', fontsize=14)

high = high_low(industry_df, "total_profit")[0]
mean = high_low(industry_df, "total_profit")[1]
low = high_low(industry_df, "total_profit")[2]

ax1.axhline(y=high, color='red', linestyle='-', linewidth=3, label='high')
ax1.axhline(y=mean, color='green', linestyle='-', linewidth=3, label='mean')
ax1.axhline(y=low, color='blue', linestyle='-', linewidth=3, label='low')

plt.title("Investment performance according to MA tactics of the Banking Industry", fontsize=18)
plt.show()
```

I draw a column chart (representing profit) combined with a line chart (representing the number of orders). Note: The number of orders represents how many times you have traded in total, you will want to trade less with the same profit as trading more.

Finally, I draw 3 more High, Mean, and Low lines on the chart.



Here are our results!

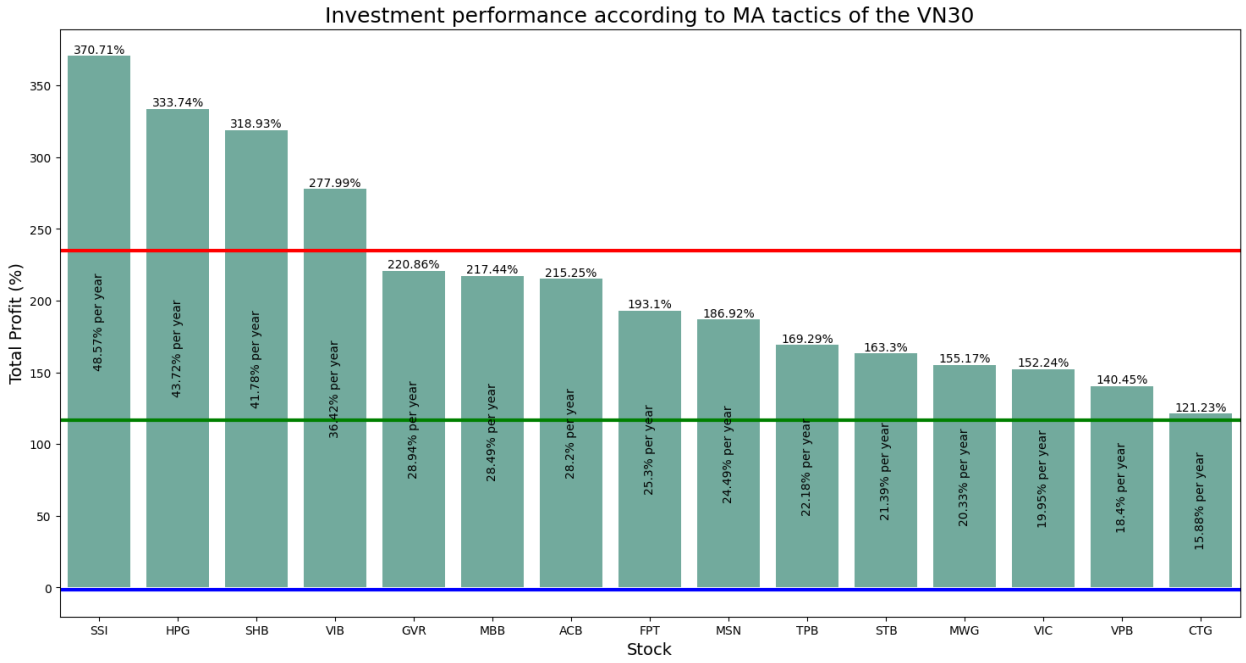
It can be seen that SHB is the best stock using ‘the MA5, 10, 20, 50, 200’ strategy (319%, equivalent to 42%/year) with the lowest trading volume. Besides, VIB, LPB, MBB, and ACB all cross the red High line. The stocks located in the High - Mean region also achieve annual returns of TPB(28.2%/year), STB(21.39%/year), VPB(18.4%/year), and CTG(15.88%/year). higher than the general market (13.88%/year I have proven above via https://www.tradingview.com/).

These are all good investment suggestions with high returns

```
vn30 = ["ACB", "BCM", "BID", "BVH", "CTG", "FPT", "GAS", "GVR", "HDB", "HPG", "MBB", "MSN", "MWG", "PLX", "POW", "SAB", "SHB", "SSB", "SSI", "STB",
        "TCB", "TPB", "VCB", "VHM", "VIB", "VIC", "VJC", "VNM", "VPB", "VRE"]
```

Once again I try to extend this tactic test pattern.
This time I will try it with VN30 (a collection of 30 large-cap stocks, based on the index set by the Ho Chi Minh City Stock Exchange as of August 19, 2023).

You can imagine that in terms of quality, these 30 companies are among the top of the Vietnamese market.



I will only draw the top 15 companies with the highest profit margins.

And we have the first place is SSI with 370% (48.57%/year). With such a huge profit margin, this is clearly a good suggestion if this stock shows signs of buying in the future.

In addition to SSI, it can be seen that all the stocks in this top 15 have a much higher profit margin than the general market.

So what are our conclusions after the research? First, applying the strategy "Buy when the price crosses the MA5, 10, 20, 50, 200 and Sell when these are no longer true" is a tactic that brings good profits.
"Simple is the best" is the right motto, MA is an extremely simple indicator that almost everyone knows. Our job is to apply it properly, comply with the rules of buying and selling, and apply it with the right stocks to make huge profits.
But this research is not without its weaknesses.
Firstly, our data set is not really large. It is necessary to expand to all trades and stocks on the stock exchange, from which to draw a conclusion that the strategy should only be applied to which trades and when to avoid using it.
Second, stocks that have given high yields in the past may not necessarily be the same in the future. The fact that the stock has a familiar momentum can be broken by a factor coming from the business itself, such as

Change of leadership, Change of business model, Influence of macro, .. factors. This can cause the stock's momentum to change suddenly. For example, if a business is predicted to have an extremely bad future, even if buying signals appear, the profit rate is now much lower than before.
So what do we need to do to limit these weaknesses? Obviously we still only use it as a tool to suggest and support investment, DO NOT look at the returns from the past but become greedy to buy and sell regardless. Use this test model to find stocks that react strongly to the strategy and wait for signals to buy from them. Then you need to research more about the nature of the business, from which to make a decision.
That's how to use my research properly.