Universiti Tunku Abdul Rahman Sungai Long

Lee Kong Chian Faculty of Engineering and Science

UCEM 3544 Case Study on Investment and Trading

May 2023 Trimester

Assignment 2

Group Report Writing

No	Name	ID	Email	Programme
1	Ng Wen Kang	2100068	kangwen177@1utar.my	FM
2	Lim Zhi Yuan	2102125	2102125@1utar.my	FM
3	Lim Sheng Hong	1904621	limsh2001.lsh@1utar.my	FM

Table of Contents

PART 2A: Research and Implementation 1

Trader F: Ng Wen Kang – Fibonacci Retracement 1

1. Introduction 1
2. Trading strategy 2
3. Implementation with Python 5
4. Conclusion 10

Trader B: Lim Zhi Yuan – Bollinger Channel Breakout Strategy 11

1. Introduction: 11
2. Trading Strategy: 11
3. Implementation: 13

Risk Controller: Lim Sheng Hong 19

1. Profit & loss and Compound Annual Growth Rate 19
2. Stop-Loss 22

Part 2B: Result and Discussion 24

1. Introduction 24

2. Result on maximum profit and loss and the live P&L on simulator data and test data 25

3. Performance comparison with strategy in Project 1 30

4. Discussion on the practicability of the designed strategy 31

5. Effects of price slippage to the designed strategy 34

PART 2A: Research and Implementation

Trader F: Ng Wen Kang – Fibonacci Retracement

1. Introduction

In project 2, I combined Fibonacci Retracement and Triple Moving Average Strategy together. Different from the implemented Fibonacci Retracement trading strategy only in Project 1 , Triple Moving Average can be used as an identified trend before entry or exit to increase the performance of trading. The indicators are integrated into this trading strategy: Fibonacci Levels indicator from project 1 and Simple Moving Average(SMA) which is new for our group and ATR (Average True Range) as a cut loss indicator.

The Simple Moving Average (SMA) is a technical indicator that can be used to determine whether an asset's price will continue or reverse a bull or bear trend. It calculates the average of a range of prices over a specified number of periods, which is known as the 'lookback' period. The formula to calculate the SMA is as follows:

$$Simple\ Moving\ Average\ = \frac{\sum_{i=0}^{N} \mathrm{P}_i}{N}$$

- P = The price of an asset at lookback period n
- N = The number of lookback period

Fibonacci Level is used to determine the potential supported line or resistance line, whatever up or down trend. This method utilizes the Fibonacci ratios to identify key price levels where markets tend to experience temporary reversals before resuming their original trend. This method takes the difference of two extreme points from historical price and multiples Fibonacci ratios to get. The formula to calculate Fibonacci levels is as follows:

$$Fibonacci\ \ levels\ = \ \mathrm{Highest}\ + (\mathrm{Highest}\ -\ \mathrm{Lowest}\,) \times\ \mathrm{Fibonacci\ ratios}$$

- Highest = Highest price in the lookback n days
- Lowest = Lowest price in the lookback n days
- Fibonacci ratios= 0, 0.236, 0.382, 0.5, 0.618, 0.728 and 1 , either one

**** Note the lookback days depend on user choice

2. Trading strategy

In this project, I've combined Fibonacci Retracement and Triple Simple Moving Averages Strategy (TMA) to create a comprehensive trading strategy. The primary concept behind this strategy is to use TMA for entry and exit signals, while Fibonacci retracement levels help determine the optimal price points for entry and exit.

Uptrend

Entry Long Signal

To identify an uptrend, I've established two conditions:

1. **MA Short > MA Mid and MA Short continues increasing for three consecutive days:** This condition checks if the short-term moving average is consistently higher than the middle-term moving average and has been on an upward trajectory for three days.

2. **MA Short > MA Mid > MA Long for the Past Three Days, and Candlestick Above MA Short:** This condition is considered as big uptrend when the short-term moving average, middle and long continuous increase with parallel in the past three days, and the current candlestick above the MA Short.

If the first condition is met, an entry-long signal is determined by Fibonacci retracement levels. For example, **if the price is around Fibonacci levels 0.5 and 0.618 when the low of the yesterday candlestick hits Fibonacci Level 0.5, then an entry-long signal is generated in next day**. If the second condition is met, an entry-long signal just directly appears without relying on Fibonacci retracement. Above these cases, the entry long order price is set as the opening price of the next trading day.

Exit Long Signal

The exit long signal is generated after entry long and when any of the following four conditions are met:

1. **MA Mid > MA Short and MA Short continue decreasing for three consecutive days:** This suggests a potential trend reversal. The short-term moving average is consistently lower than the middle-term moving average and has been on an downward for three days

2. **Candlestick Above Fibonacci Level 1.618**: This is an extreme situation indicating a price surge.

3. **Candlestick Below MA Short**: This may indicate a change in trend direction because the price suddenly left with average price in the past

4. **Cut Loss Condition**: The cut loss price is determined using the ATR (Average True Range) indicator. An exit long signal is triggered if the traded price is smaller than the 2 times of ATR

All exit long order price is set as the opening price of the next trading day, but cut loss condition is immediately sold with market price.

Downtrend

Entry Short Signal

For identifying a downtrend, the conditions are as follows:

1. **MA Mid > MA Short and MA Short Continues Decreasing for three consecutive days**: This condition checks if the short-term moving average is consistently lower than the middle-term moving average and has been declining for three consecutive days.

2. **MA Long > MA Mid > MA Short for the past three days, and Candlestick Below MA Short**: This condition is considered a **big downtrend**

If the first condition is met, an entry short signal is determined by Fibonacci retracement levels. For example, if the stock price hovers around Fibonacci levels of 0.382 and 0.236, while the high of the yesterday candlestick hits Fibonacci Level 0.382, then appear entry short signal in tommorow. When the second condition is met, an entry short signal appears without relying on Fibonacci retracement. In this case, the entry short order price is set as the opening price of the next trading day.

Exit Short Signal

The exit short signal is generated when any of the following four conditions are met:

1. **MA Short > MA Mid**: This suggests a potential trend reversal.

2. **Candlestick Below Fibonacci Level -0.236:** This is an extreme situation of price drop.

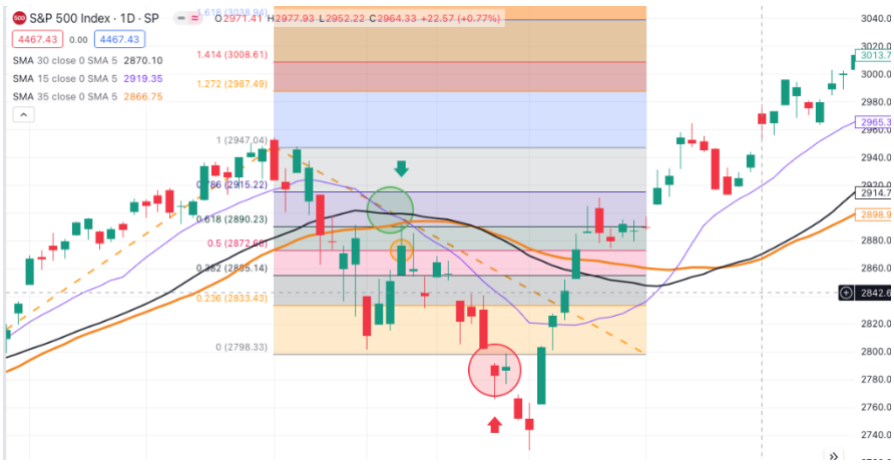
3. **Candlestick Above MA Short**: This may indicate a change in trend direction because the current price is left with the average price in the past.

4. **Cut Loss Condition:** The cut loss price is determined using the ATR indicator. An exit short signal is triggered if the traded price is bigger than 2 times of ATR

All exit short order price is set as the opening price of the next trading day, but cut loss condition is immediately sold with market price.

Example case of trading

In the left bubble, the entry signal appears when condition 2 for entry long is met. In the right bubble, the exit long signal triggers due to condition 3, where the candlestick below the MA Short.



In the left green bubble, the entry short signal appears as condition 1 from entry short is met, and orange bubble, it's confirmed when the candlestick hits Fibonacci Level 0.5. On the right of red bubble, the exit short signal triggers because the candlestick is approaching Fibonacci Level -0.236, meeting condition 2 from exit short.

3. Implementation with Python

The mentioned trading strategy is implemented in Python as required in Project 2, which is continuous in Project 1.

In Project 2, much like implementation in Project 1, we continue to collect date and price data as the simulator data streams in. This data is crucial for multiple aspects of our trading strategy, such as constructing candlestick and computing three indicators: the Simple Moving Average (SMA), Fibonacci Retracement levels, and the Average True Range (ATR).

All the functions responsible for calculating these indicators are located within the **`main()`** function. However indicator calculation functions are located within the **`main()`** function, but they are executed under a conditional statement called **"isnewcandle."** This condition ensures that the indicators are calculated only when the candlestick is fully formed

To calculate the Average True Range (ATR), the **`atr(n)`** function, where 'n' represents the parameter used for this calculation. For computing the Simple Moving Average (SMA), I employ the **`calculate_ma(day1, day2, day3)`** function, with the period for SMA calculations stored in the global variable 'day_ma.'For determining Fibonacci levels, I use the **`fibonacci(n)`** function, with 'n' serving as the parameter.

```
def atr(n):
    h = [row[2] for row in data["candles"]][:-1]
    l = [row[3] for row in data["candles"]][:-1]
    c = [row[4] for row in data["candles"]][:-1]
    data["tr"].append(None)
    data["atr"].append(None)
    size_candles = len(data["candles"])
    d = [row[0] for row in data["candles"]][-1]

    if size_candles >2:

        tr = max(h[-1]-l[-1],abs(h[-1]-c[-2]),abs(l[-1]-c[-2]))
        data["tr"][-1]=tr


        if len(data["tr"]) == n+2:

            atr = sum(data["tr"][-n:])/n
            data["atr"][-1]=atr


        if len(data["tr"]) > n+2:
            atr = (data["atr"][-2]*(n-1)+data["tr"][-1])/n
            data["atr"][-1] = atr
```

Figure 1 function calculate ATR

```
def calculate_ma(day_1,day_2,day_3):
    c = [row[4] for row in data["candles"]][:-1]
    #print(c)
    data["ma_mid"].append(None)
    data["ma_long"].append(None)
    data["ma_short"].append(None)

    if len(c)> day_1:
        ma_1= sum(c[-day_1-1:-1])/day_1
        data["ma_short"][-1]=ma_1

    if len(c) > day_2 :
        ma_1= sum(c[-day_1-1:-1])/day_1
        ma_2= sum(c[-day_2-1:-1])/day_2
        data["ma_short"][-1]=ma_1
        data["ma_mid"][-1]=ma_2


    if len(c)> day_3:
        ma_1= sum(c[-day_1-1:-1])/day_1
        ma_2= sum(c[-day_2-1:-1])/day_2
        ma_3= sum(c[-day_3-1:-1])/day_3
        data["ma_short"][-1]=ma_1
        data["ma_mid"][-1]=ma_2
        data["ma_long"][-1]=ma_3
```

Figure 2 function calculate SMA

The **`strategy(day, price, conn)`** function plays a central role in this implementation as it manages all trading transactions. However, this function is placed within a loop called **(`for d, p`),** because of limit order types and implementing stop-loss conditions. These elements necessitate that consider the currently price of trading, which are not after candlestick is constructed.

Since using limit order type, there may be instances where the submitted order price and actual transacted price. I have the `**record_order(conn)**` function, which is responsible for storing the actual prices and date of transcation are executed. This function also used for computing total realised profit after stimulation.

```
def main():
    conn = connect_brokerage(instrument,0.03)
    for d, p in conn.data_stream():
        collect_price(d, p)
        #visualise_candlesticks()
        is_new_candle = make_candlesticks_day()
        strategy(d,p,conn)
        #cancel_order(conn)
        #print(d, p)
        record_order(conn)
        calculate_live_realised_profit()
        calculate_live_unrealised_profit()

        if is_new_candle:
            data_creation() # Append None in data
            atr(14)
            calculate_ma(day_ma[0], day_ma[1], day_ma[2])
            fibonacci(day_test[0])


    conn.logout()
    calculate_profit(conn)
    calculate_win_rate()
```

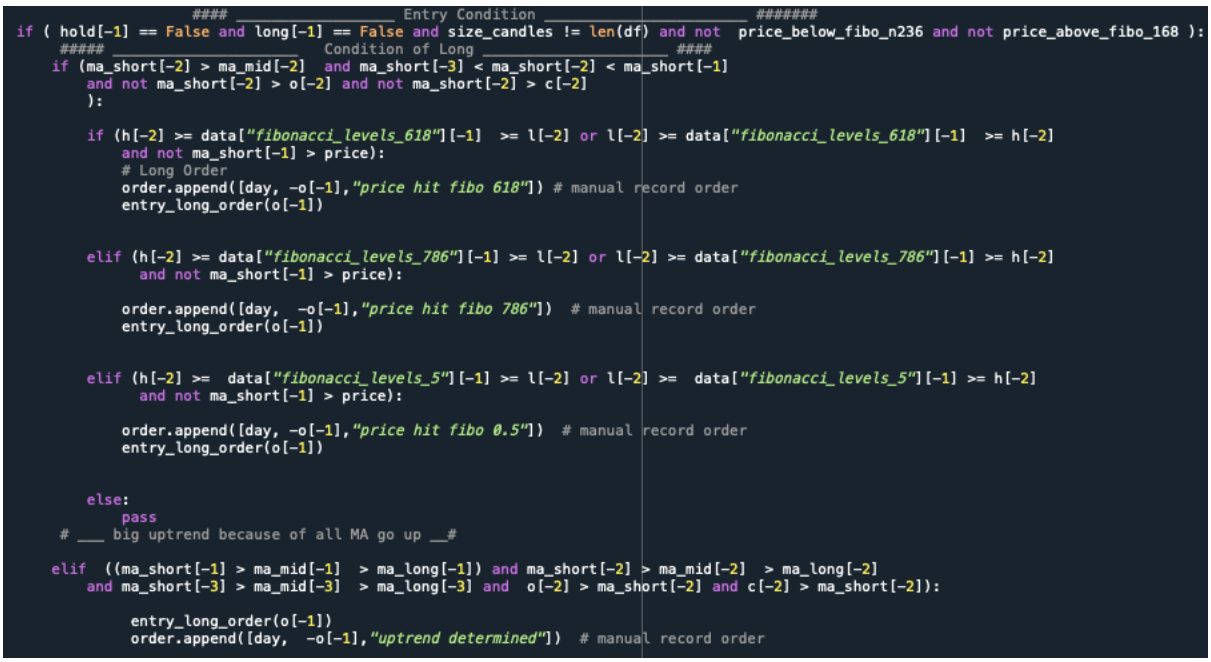
Figure 3 main()

In the **strategy (day, price, conn)** method, I've implemented a comprehensive trading strategy involving four distinct conditional statements to identify entry long, entry short, exit long, and exit short conditions. Prior to entering a trade, I've introduced two variables, namely **"hold"** and **"long,"** both of which default to **False**. Furthermore, when event that either of the two entry conditions is met, such as in the case of an entry long trade, the **"Hold"** variable is set to True, while the **"Long"** variable is also set to True. Conversely, for entry short trades, the "**Hold**" variable is set to True, but the **"Long"** variable is set to False.

For the entry long and entry short conditions, I have designed two separates if statements, as previously I mentioned in trading strategy part. To make a clearly trading activities, I've established a global variable named **data["order"]** to store date, submission order price and reason to trade. Within this variable, I diligently record order-specific details using the **order.append([day, price, reason])** code snippet. This allows me to track critical trade information, including the date of execution, the rationale behind each trade, and the specific price at which orders were placed. It can compute unrealised total profit.

To reduce the frequency of daily transactions, I have settled of conditions within the entry section of the code. Normally, I place entry conditions without (and not) exit conditions in entry section, avoid the possibility of both entry and exit conditions to occur same time. For exit section part also do like same ways.

The initial line of below codes, "**size_candle != len(df)"** serves to prevent entries on the last day of the duration. Furthermore, I've included conditions such as The "**price_below_fibo_n236"** and "**price_above_fibo_168"** to restrain entry when prices surpass the extreme levels of Fibonacci Levels -0.236 and 1.618, because these are considered extreme price locations. The **h[-2]** and **c[-2]** are used to reference the highest and lowest prices of the candlestick from the previous day, respectively.



```
                ####  _________________ Entry Condition _____________________ #######
if ( hold[-1] == False and long[-1] == False and size_candles != len(df) and not  price_below_fibo_n236 and not price_above_fibo_168 ):
     #####  _____________  Condition of Long _____________ ####
    if (ma_short[-2] > ma_mid[-2]  and ma_short[-3] < ma_short[-2] < ma_short[-1]
        and not ma_short[-2] > o[-2] and not ma_short[-2] > c[-2]
        ):

        if (h[-2] >= data["fibonacci_levels_618"][-1]  >= l[-2] or l[-2] >= data["fibonacci_levels_618"][-1]  >= h[-2]
            and not ma_short[-1] > price):
            # Long Order
            order.append([day, -o[-1],"price hit fibo 618"]) # manual record order
            entry_long_order(o[-1])


        elif (h[-2] >= data["fibonacci_levels_786"][-1] >= l[-2] or l[-2] >= data["fibonacci_levels_786"][-1] >= h[-2]
              and not ma_short[-1] > price):

            order.append([day,  -o[-1],"price hit fibo 786"])  # manual record order
            entry_long_order(o[-1])


        elif (h[-2] >=  data["fibonacci_levels_5"][-1] >= l[-2] or l[-2] >=  data["fibonacci_levels_5"][-1] >= h[-2]
              and not ma_short[-1] > price):

            order.append([day, -o[-1],"price hit fibo 0.5"])  # manual record order
            entry_long_order(o[-1])


        else:
            pass
     # ___ big uptrend because of all MA go up __#

    elif  ((ma_short[-1] > ma_mid[-1]  > ma_long[-1]) and ma_short[-2] > ma_mid[-2]  > ma_long[-2]
        and ma_short[-3] > ma_mid[-3]  > ma_long[-3] and  o[-2] > ma_short[-2] and c[-2] > ma_short[-2]):

             entry_long_order(o[-1])
             order.append([day,  -o[-1],"uptrend determined"])  # manual record order
```

Figure 4 Entry long condition statement

To execute trading orders, `**entry_long_order(price)**` for initiating buy orders and `**entry_short_order(price**)` for entering short positions, with the "**price**" parameter representing the order submission price. To maintain proper trade tracking, I append the values of "**Hold**" and "**Long**" variables because the position changes after an order is submitted, and these variables can guide whether to enter or exit in trading. When both **"Hold" and "Long" variables are True,** the code proceeds to execute the **exit long** statement, while if "**Hold"** remains **True but "Long" is False,** it triggers the **exit short statement.**

```
def entry_long_order(p):
    data["entry_long"][-1]=p
    long.append(True)
    hold.append(True)
    conn.submit_order(
        price=p,
        lot_size=1,
        order_type="Limit",
        action="Buy")

def entry_short_order(p):

    data["entry_short"][-1]=p
    long.append(False)
    hold.append(True)
    conn.submit_order(
        price=p,
        lot_size=1,
        order_type="Limit",
        action="Sell")
```

Figure 5 entry_long_order() and entry_short_order()

For exit short section, there are four conditional statements that reference the conditions have mentioned in the trading strategy part. Interest things, the code structure for exit long and exit short is quite similar. In the exit long section, there still have 4 conditional statements that reference in exit long in trading strategy part is mentioned.

When the "**Hold**" variables is true but "**Long**" variables is false, it execute exit short. The global variable **data["cut_loss_short"]** is used to store the price at which plan to cut losses. The **exit_short_order(price)** function is responsible for initiating sell orders, and the variable "price" represents the order submission price. However , under the cut loss condition statement, the submit price is not use next day opening market price, so “price” variable is set as p (currently price) from (for d,p) while loop .

```
def exit_long_order(price):

    p = price
    data["exit_long"][-1] = p
    long.append(False)
    hold.append(False)
    conn.submit_order(
        price=p,
        lot_size=1,
        order_type="Limit",action="Sell")

def exit_short_order(price):

    p = price
    data["exit_short"][-1] = p
    hold.append(False)
    long.append(False)
    conn.submit_order(
        price=p,
        lot_size=1,
        order_type="Limit",
        action="Buy")  # Buy or Sell
```

Figure 6 exit_long_order(price) and exit_short_order(price)

Whenever any of the four exit short conditions are met, both the "**Hold**" and "**Long**" variables are reset to **False**. This will close our position and allows the trader to return to the entry **condition** statement. It means re-entry into the market.

```
 ####  ____________________ Exit Short  ____________________####
elif hold[-1] == True and long[-1] == False :
    data["cut_loss_short"][-1] = order[-1][-2] + 2 * atr
    cut_loss_short = data["cut_loss_short"][-1]
    ma_mid_bigger_ma_short = (ma_short[-1] < ma_mid[-1] and  ma_short[-2] < ma_mid[-2] and ma_short[-3] > ma_short[-2] > ma_short[-1])


    #_____ MA short > MA mid continuous 2 days _______#
    if ma_short[-2] > ma_mid[-2] and ma_short[-3] > ma_mid[-3]:

        exit_short_order(o[-1])
        order.append([day, -o[-1],'buyback with MA cross'])



    #_____ candlestick below MA shorts _______#
    elif c[-2] > ma_short[-2] and  o[-2] > ma_short[-2] :

        exit_short_order(o[-1])
        order.append([day, -o[-1],'buyback with candle below MA'])


    #_____ candlestick below ibonacci(day, 0)_______#
    elif o[-2] < data["fibonacci_levels_n236"][-1] or c[-2] < data["fibonacci_levels_n236"][-1]:

        exit_short_order(o[-1])
        order.append([day, -o[-1],'buyback with candle below fibo -0.236'])

    elif price > cut_loss_short:

         exit_short_order(price)
         order.append([day, -price,'cut loss short'])
```

Figure 7 Exit short condition statement

On the final day of the trading duration, when the "Hold" variable is true , I have to close all position. To achieve this, submitted order price is next day's opening market price and the order type is set to "market." This approach ensures that the transaction is successfully executed.

```
 #________  Last day to cut off  _____________#
if hold[-1] == True  and long[-1] == False and size_candles == len(df) and len(conn.get_transactions(True)[0])==0:

    p= o[-1]
    data["exit_short"][-1]=p
    order.append([day,-p,"cut off short position"])
    hold.append(False)
    long.append(False)
    conn.submit_order(
        price= o[-1],
        lot_size=1,
        order_type="Market",
        action="Buy"  # Buy or Sell
        )

if hold[-1] == True  and long[-1] == True and size_candles == len(df) and len(conn.get_transactions(True)[0])==0:

    p= o[-1]
    data["exit_long"][-1]=p
    order.append([day,p,"cut off long position"])
    hold.append(False)
    long.append(False)
    conn.submit_order(
                    price= o[-1],
                    lot_size=1,
                     order_type="Market",
                    action="Sell"  # Buy or Sell
                    )
```

Figure 8 statement for cut off position

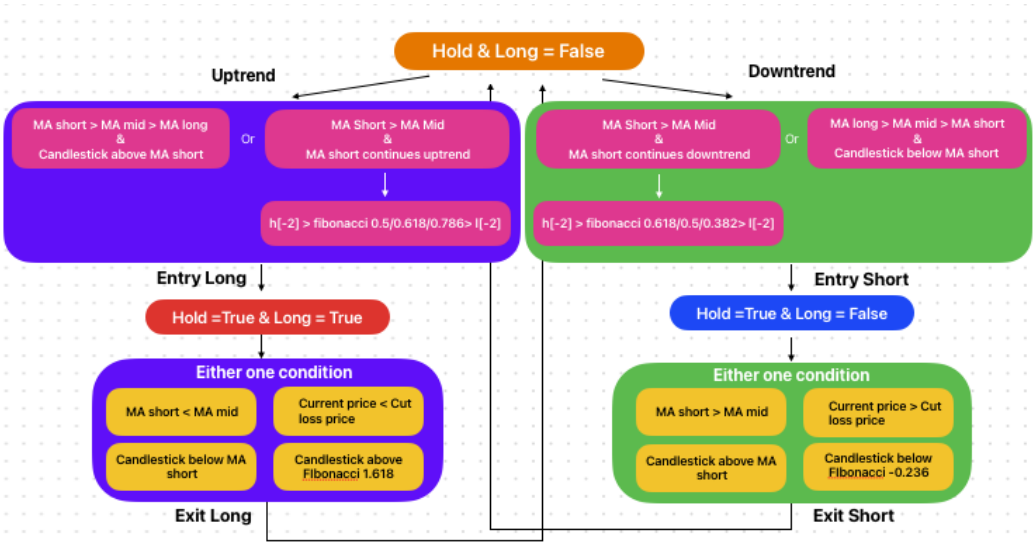
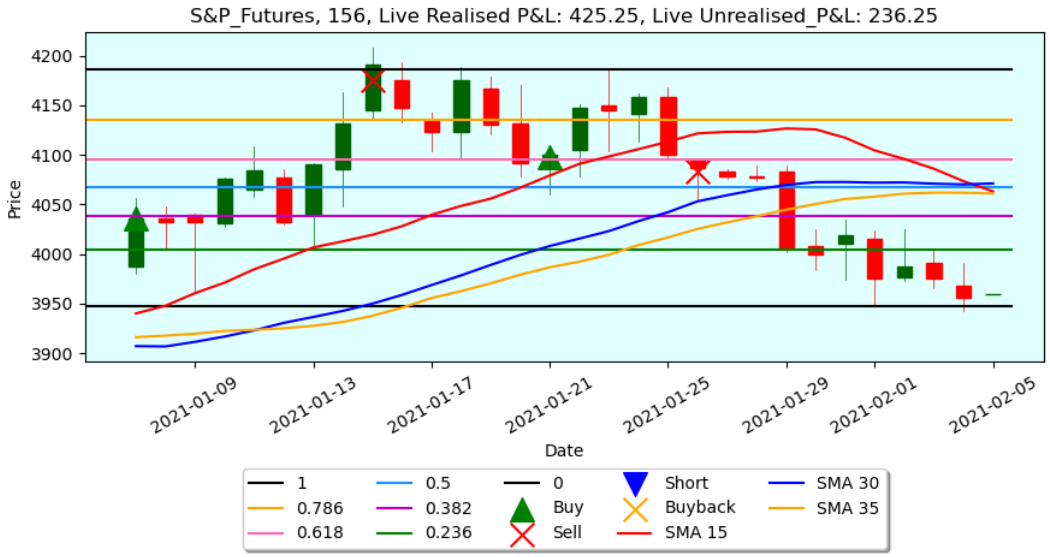


Figure 9 Flowchart of strategy

4. Conclusion

In conclusion, the trading strategy combined TMA stratrgy and Fibonacci Retracement is successfully implemented in python and visualization of the entry, exit , and stop loss exit on the candlestick chart. The performance of the combination trading strategy is also exceeding our expectation especially with the use of the limit order and compare with Project 1 is perform better. I can say the trading strategy combined TMA and Fibonacci Retracement is indicating trading strategy able to work well.



Trader B: Lim Zhi Yuan – Bollinger Channel Breakout Strategy

1. Introduction:

In project 2, the trading strategy that I chose is Bollinger Channel Breakout strategy. Technical analysis chart indicators called Bollinger Bands are commonly employed by traders in various markets. Bollinger Bands may be used for a variety of purposes, including trend tracking, identifying overbought and oversold levels, and keeping an eye out for breakouts. Bollinger bands are used as trading tools to determine entry and exit positions for a trade. The indicator emphasizes volatility and pricing. It is a rather straightforward trading tool that is highly well-liked by both professional and individual investors. Below is the calculation for Bollinger Bands and Simple Moving Average:

$$\text{SMA} = (\text{Sum of Closing Prices for N Periods}) / N$$

$$\text{Upper Bollinger Band} = \text{SMA}(\text{Close},n) + m*\sigma[\text{Close},n]$$

$$\text{Lower Bollinger Band} = \text{SMA}(\text{Close},n) - m*\sigma[\text{Close},n]$$

Where:

$\text{SMA}=\text{Simple Moving average}$

$n=20$ (Number of days in smoothing period)

$m=2$ (Number of standard deviations)

$\sigma[\text{Close},n]=\text{Standard Deviation over last n periods of Close Price}$

2. Trading Strategy:

I begin by setting up the Bollinger Bands indicator with parameters in order to apply a Bollinger Bands trading strategy efficiently. Among these often-used settings is "Period (n)," which denotes the number of periods for the simple moving average. Normally 20 will be chosen as the value, so I chose n=20, but it can be changed corresponding to preferred trading timeframe. The "Standard Deviation Multiplier (m)" also has a significant impact on how wide the bands are. The outside bands will be two standard deviations distant from the moving average if the value is 2, which is normal. If required, change the value to reflect tastes and market circumstances.

Two significant trading indications produced by the Bollinger Bands are the "Bollinger Band Squeeze" and "Price Touching Bands." When the price starts to constrict inside the area

between the upper and lower Bollinger Bands, this is known as a Bollinger Band squeeze. This implies little volatility and frequently implies a significant price breakout. This squeeze is frequently used by traders as a possible entry signal. When the price meets or crosses one of the outside bands, another signal is generated. Moving averages, whether exponential or simple, can also be used to determine the general trend's direction. Meanwhile, I chose simple moving average by using close price.

Consider opening long (buy) positions when getting a bullish signal, such as a Bollinger Band squeeze followed by a price touch of the lower band, and when support from different signals is in line with this signal. On the other hand, when receiving confirmation from additional indicators and receiving a bearish signal such as a Bollinger Band squeeze followed by a price contact of the upper band, consider taking a short position (selling).

Appropriate risk management approaches must be used in order to reduce risk. Always use take-profit orders to protect gains and stop-loss orders to minimize possible losses. Based on different tolerance for risk and the current market conditions, these levels can be altered.

It's crucial to constantly monitor the positions. As the market changes, modify the stop-loss and take-profit orders, and think about using the following stops to lock in gain as the price rises in the direction of self-choice. Whether the departure entails achieving a set profit objective, getting a signal that contradicts the entrance, or noticing price movement that suggests a probable reversal, be ready with a well-defined exit strategy.

For long positions, an "exit" signal is generated when the most recent trading signal suggests entering a long position, and the closing price of the preceding candle exceeds the value of the Simple Moving Average (SMA) line. This exit decision marks the end of the long trade. The closing price of the most recent candle is then used as the exit price, which is then recorded. Calculating the profits or losses from the deal depends on the recorded exit price. Executing a market order to sell is a key component of our long position exit strategy since it guarantees a quick and effective close of the position.

Conversely, for short positions, an "exit" signal is generated when the initial trading signal suggests entering a short position, and the closing price of the previous candle falls below the value of the SMA line. This "exit" signal signifies the conclusion of the short trade. Similar to long positions, record the exit price, which corresponds to the closing price of the most recent candle. This recorded exit price is indispensable for evaluating the performance of the trade.

The exit strategy for short positions involves initiating a market order to buy, facilitating a swift and effective closure of the short position.

3. Implementation:

```
def connect_brokerage(instrument,s):
    username, password = "angry_bird", "0000000"
    exchange = "BDM"
    conn = brokerage.login("www.bursa.com", username, password, speed=s)
    conn.connect(exchange, instrument)
    return conn
```

Connecting to a brokerage platform for trading purposes is made easier by the function connect_brokerage, which is defined in the Python code that is given. It requires the two parameters instrument (which represents the trading financial instrument) and s (usually the connection speed). The function connects to the trading platform, gives login information (username and password), specifies the trading exchange, and then returns a connection object. The usage of this item is possible in later trade operations. When utilising the code in a real trading environment, it is crucial to swap out the placeholder login credentials with actual ones.

```
def collect_price(d, p):
    data['dates'].append(d)
    data['prices'].append(p)

def make_candlesticks():
    if data["candle_duration_unit"] == "day":
        make_candlesticks_day()
    else:
        raise "Error: only minute or day is allowed"
```

The ‘def collect_price(d, p)’ and def make_candlesticks()’ same as part. For saving price information, the “collect_price” method is necessary. It adds a timestamp ('d') and price ('p') to 'data['dates']' and 'data['prices']' as arguments. The trading strategy's visualisation and analysis are built around this data. Based on the selected time unit, the'make_candlesticks' function creates candlestick data. For daily intervals, it calls "make_candlesticks_day()" if the unit is "day". To preserve uniformity in trade analysis and visualisation, an error is thrown for any additional units.

```
def make_candlesticks_day():
    d = data['dates'][-1]
    p = data['prices'][-1]
    candles = data['candles']

    ddate = d.date()
    dtime = datetime.time(0, 0, 0)
    d0 = datetime.datetime.combine(ddate, dtime)

    if len(candles) == 0:
        candles.append([d0, p, p, p, p])

    d_start = candles[-1][0]
    if d - d_start < datetime.timedelta(days=data['candle_duration']):
        d0, o, h, l, c = candles[-1]

        c = p
        h = max(h, p)
        l = min(l, p)

        candles[-1] = [d0, o, h, l, c]
    else:
        candles.append([d0, p, p, p, p])
```

The ‘def make_candlesticks_day()’ also same as part 1. The command "make_candlesticks_day" generates data for daily interval candlesticks. If the list is empty, it creates a new candle and starts it with the current price. It either makes a new candle, as shown by a "new_candle" message or changes the high and low prices within the allotted time.

```
def calculate_sma(input_data, sma_period):
    return sum(price / sma_period for price in input_data[-sma_period:])

def make_sma(data):
    if len(data['sma_line']) < len(data['candles']):
        data['sma_line'].append(None)
    assert len(data['sma_line']) == len(data['candles']), "make_sma: wrong data.sma length"

    if len(data['candles']) >= 20:
        input_data = [c[-1] for c in data['candles']]
        sma_line = calculate_sma(input_data, sma_period=20)
        data['sma_line'][-1] = sma_line
```

Two functions in the code are used to compute and update a Simple Moving Average (SMA) within a data structure known as "data." Input_data, which provides a series of numerical data points, and sma_period, an integer denoting the SMA period, are the two arguments required by the first function, "calculate_sma." For the provided data series over the chosen time, this function calculates the SMA and provides the outcome. The second function, "make_sma," is used to update and maintain the SMA line inside the technical analysis data structure. The SMA line's length is compared to the length of the candle data ('data['candles']') and, if necessary, 'None' values are added to the SMA line to guarantee that both lists have the same length. Additionally, it determines if the candle data length is equal to or larger than 20, which is a typical SMA period. If this criterion is satisfied, the function takes the most recent 20 data points from the candle data, which generally reflect closing prices. The SMA is calculated using the 'calculate_sma' function with this extracted data and a 20-period SMA.

The function then inserts the computed SMA value into the final element of the SMA line within the 'data' structure. In order to do technical analysis, which is a typical method for spotting price patterns and smoothing price data in the financial markets, it is essential to keep the SMA line up to date.

```
def calculate_upper_bollinger_band(input_data, sma_period, num_std_dev):
    sma = sum(price / sma_period for price in input_data[-sma_period:])
    std_dev = np.std(input_data[-sma_period:])
    ubb = sma + (num_std_dev * std_dev)
    return ubb

def make_upper_bollinger_band(data, num_std_dev=2):
    if len(data['ubb_line']) < len(data['candles']):
        data['ubb_line'].append(None)
    assert len(data['ubb_line']) == len(data['candles']), "make_upper_bollinger_band: wrong data.ubb length"

    if len(data['candles']) >= 20:
        input_data = [c[-1] for c in data['candles']]
        ubb_value = calculate_upper_bollinger_band(input_data, sma_period=20, num_std_dev=num_std_dev)
        data['ubb_line'][-1] = ubb_value
```

The provided Python code contains two functions for computing and updating the Upper Bollinger Band (UBB) within a data structure named "data." The first function, 'calculate_upper_bollinger_band,' calculates the UBB based on input data, a SMA period, and the number of standard deviations. It computes the SMA and multiplies it by the standard deviation to derive the UBB. The second function, 'make_upper_bollinger_band,' ensures that the UBB line within 'data' is updated appropriately. It aligns the UBB line's length with the candle data, adds 'None' values as needed, and computes the UBB using a default of 2 standard deviations if the candle data has a length of 20 or more. These functions are crucial for maintaining an up-to-date UBB line, which is integral to Bollinger Bands, a widely used tool for assessing price volatility and identifying potential trading opportunities in financial markets.

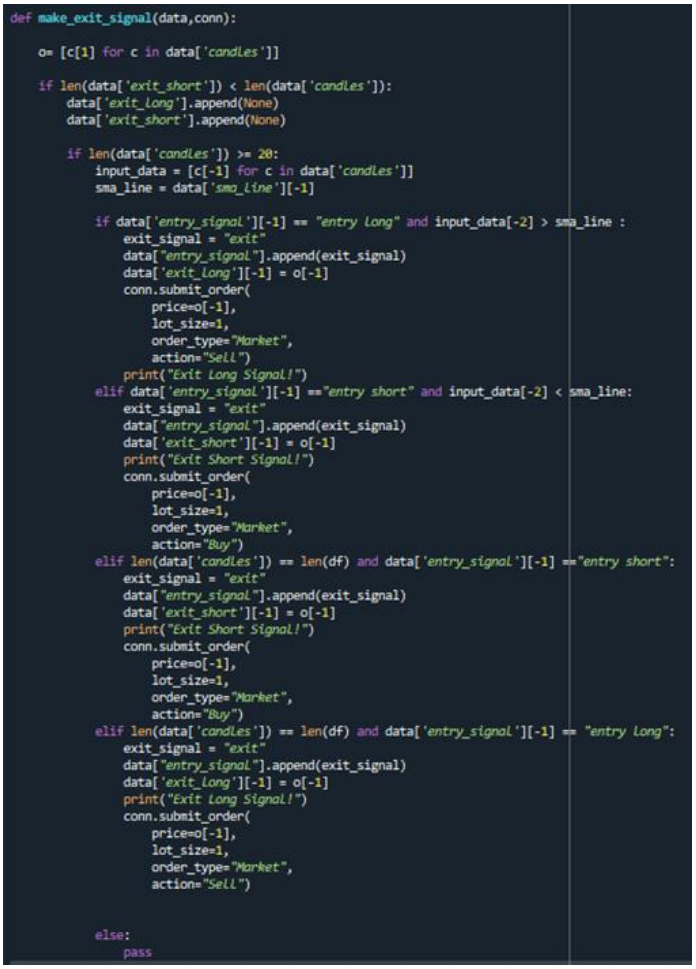
```
def calculate_lower_bollinger_band(input_data, sma_period, num_std_dev):
    sma = sum(price / sma_period for price in input_data[-sma_period:])
    std_dev = np.std(input_data[-sma_period:])
    lbb = sma - (num_std_dev * std_dev)
    return lbb

def make_lower_bollinger_band(data, num_std_dev=2):
    if len(data['lbb_line']) < len(data['candles']):
        data['lbb_line'].append(None)
    assert len(data['lbb_line']) == len(data['candles']), "make_lower_bollinger_band: wrong data.lbb length"

    if len(data['candles']) >= 20:
        input_data = [c[-1] for c in data['candles']]
        lbb_value = calculate_lower_bollinger_band(input_data, sma_period=20, num_std_dev=num_std_dev)
        data['lbb_line'][-1] = lbb_value
```

The provided Python code includes two functions for computing and updating the Lower Bollinger Band (LBB) within a data structure called "data." The first function, 'calculate_lower_bollinger_band,' calculates the LBB based on input data, an SMA period, and the number of standard deviations. It derives the LBB by first calculating the SMA within the specified period, then determining the standard deviation, and subtracting a multiple of the standard deviation from the SMA. The second function, 'make_lower_bollinger_band,' ensures that the LBB line within 'data' is updated appropriately. It aligns the LBB line's length with the

candle data, adds 'None' values as needed, and computes the LBB using a default of 2 standard deviations if the candle data has a length of 20 or more. These functions are vital for maintaining an up-to-date LBB line, which is a crucial component of the Bollinger Bands indicator. The LBB, when combined with the Upper Bollinger Band (UBB), facilitates the assessment of price volatility and the identification of trading opportunities in financial markets.

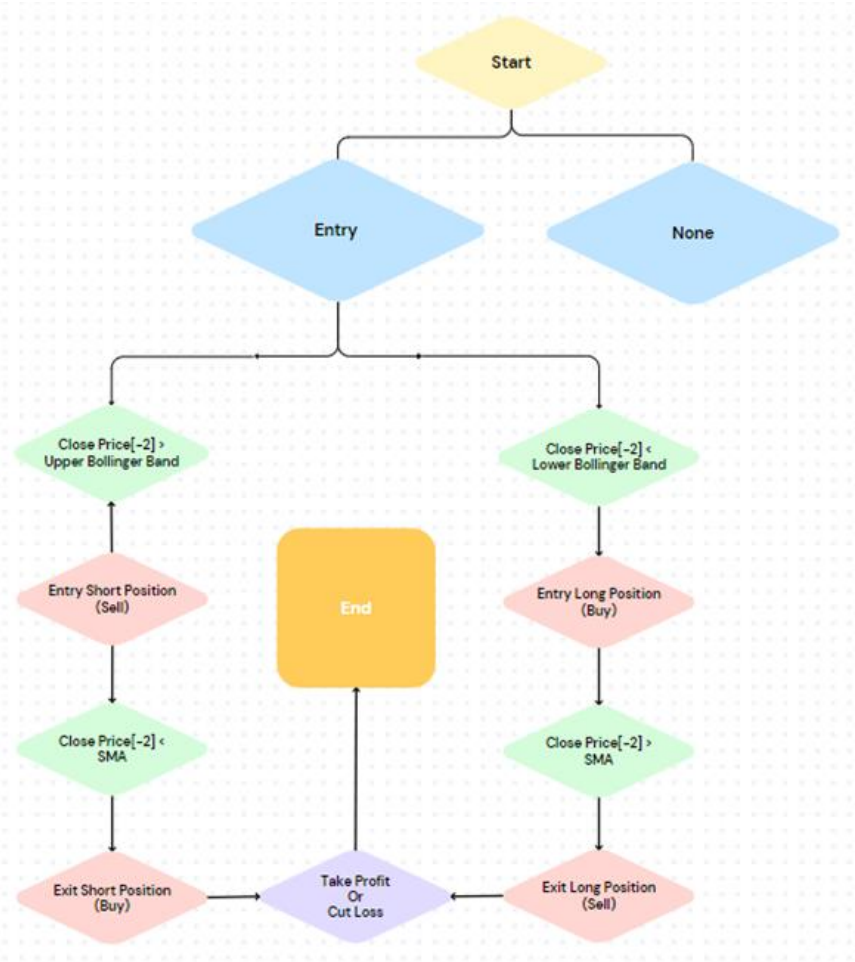
```
def make_entry_signal(data,conn):

    o= [c[1] for c in data['candles']]

    if len(data['entry_Long']) < len(data['candles']):
        data['entry_Long'].append(None)
        data["entry_short"].append(None)

        if len(data['candles']) >= 21:
            input_data = [c[-1] for c in data['candles']] #close price
            ubb_line = data['ubb_line'][-2]
            lbb_line = data['lbb_line'][-2]


            if (data["entry_signal"][-1] == "exit" and input_data[-2] < lbb_line and len(data["candles"]) != len(df)):
                entry_signal = "entry Long"
                data["entry_signal"].append(entry_signal)
                data['entry_Long'][-1] =o[-1]
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Buy")
                print("Entry Long Signal!")


            elif (data["entry_signal"][-1] == "exit" and input_data[-2] > ubb_line and len(data["candles"]) != len(df)):
                entry_signal = "entry short"
                data["entry_signal"].append(entry_signal)
                data['entry_short'][-1] = o[-1]
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Sell")
                print("Entry Short Signal!")

            else:
                pass
```

The Python code features a "make_entry_signal" function for generating entry signals based on candlestick data within trading strategies. It takes two key inputs: "data" (likely containing strategy details) and "conn" (representing a brokerage platform connection). The function extracts opening prices ("o") from the "candles" data in "data" and ensures that the lengths of the "entry_long" and "entry_short" lists match the "candles" data length. If the "candles" data has a length of at least 21, a prerequisite for generating entry signals, and the previous signal was "exit," it generates an "entry long" signal when the previous candle's closing price is below the Lower Bollinger Band ("lbb_line") and the "candles" data length differs from "df." This signal is recorded in "entry_signal" with the entry price in "entry_long," and a buy trade is executed through "conn." Similarly, it generates an "entry short" signal if the previous signal was "exit," the previous candle's closing price surpasses the Upper Bollinger Band ("ubb_line"), and the "candles" data length doesn't match "df." This signal is noted in "entry_signal," with the entry price in "entry_short," and a sell trade is executed via "conn." In summary, this code aids in a trading strategy utilizing candlestick data and Bollinger Bands to generate entry signals, triggering buy or sell orders through the brokerage connection, with added print statements for signal indication.



```
def make_exit_signal(data,conn):

    o= [c[1] for c in data['candles']]

    if len(data['exit_short']) < len(data['candles']):
        data['exit_long'].append(None)
        data['exit_short'].append(None)

        if len(data['candles']) >= 20:
            input_data = [c[-1] for c in data['candles']]
            sma_line = data['sma_line'][-1]

            if data['entry_signal'][-1] == "entry long" and input_data[-2] > sma_line :
                exit_signal = "exit"
                data["entry_signal"].append(exit_signal)
                data['exit_long'][-1] = o[-1]
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Sell")
                print("Exit Long Signal!")
            elif data['entry_signal'][-1] =="entry short" and input_data[-2] < sma_line:
                exit_signal = "exit"
                data["entry_signal"].append(exit_signal)
                data['exit_short'][-1] = o[-1]
                print("Exit Short Signal!")
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Buy")
            elif len(data['candles']) == len(df) and data['entry_signal'][-1] =="entry short":
                exit_signal = "exit"
                data["entry_signal"].append(exit_signal)
                data['exit_short'][-1] = o[-1]
                print("Exit Short Signal!")
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Buy")
            elif len(data['candles']) == len(df) and data['entry_signal'][-1] == "entry long":
                exit_signal = "exit"
                data["entry_signal"].append(exit_signal)
                data['exit_long'][-1] = o[-1]
                print("Exit Long Signal!")
                conn.submit_order(
                    price=o[-1],
                    lot_size=1,
                    order_type="Market",
                    action="Sell")


            else:
                pass
```

The "make_exit_signal" function is essential for creating exit signals. It accepts the arguments "data" (which contains information about the trading strategy) and "conn" (a brokerage connection). By adding "None" values where necessary, the function makes sure that the "exit_short" and "candles" data lengths are aligned. Additionally, it verifies that the "candles" data length, which is necessary for exit signals, is at least 20. Exit signals are produced using the prior "entry_signal." For instance, a "exit" signal would be generated if the previous signal was "entry long" and the closing price of the previous candle was higher than the SMA line. In contrast, if the closing price dropped below the SMA line and the prior signal was "entry short," it results in a "exit" signal. Exit criteria are also taken into account when the "candles" data length meets the "df". Finally, a "pass" statement manages other circumstances. In conclusion, this code section makes it easier to generate exit signals for a trading system that uses candlestick data and SMAs. Through the brokerage link, these signals trigger market orders for selling or purchasing, along with corresponding print statements for notice.



Flowchart of Strategy

In conclusion, the Python implementation of the Bollinger Band Trading Strategy is effective, and the entry and exit signals are visualized well in the candlestick chart (Shown as below). Particularly when using limited orders, the Bollinger Bands Trading Strategy outperforms the Project 1 indicator.

Visualization of BB result

Risk Controller: Lim Sheng Hong

1. Profit & loss and Compound Annual Growth Rate

```
def calculate_live_realised_profit_and_cagr():


    if len(data['transaction_entry_exit']) == 0:

        data["cagr"].append(0)
        data['profit'].append(0)
        data['unrealised_profit'].append(0)
        data['transaction_entry_exit'].append([0, None, data['profit'][-1]])
        data["live_realised_p_l"] = data['profit'][-1]
        data["profit_per_day"].append(0)

    else:

        if len(data['transaction']) == 0:

            data["cagr"].append(0)
            data['profit'].append(0)
            data['unrealised_profit'].append(0)
            data['transaction_entry_exit'].append([0, None, data['profit'][-1]])
            data["live_realised_p_l"] = data['profit'][-1]
            data["profit_per_day"].append(0)
```

Calculate_live_realised_profit_and_cagr is a function defined in the supplied code. This function appears to be a component of a larger programme that maintains financial transaction data and computes various profit and compound annual growth rate (CAGR) indicators for a portfolio or investment plan.

The function begins by determining the length of the 'transaction_entry_exit' list in the ‘data’ data structure. If this list is empty, no transactions have occurred, and the code initialises several critical data fields, including 'cagr', 'profit', 'unrealised_profit', 'transaction_entry_exit', 'live_realised_p_l', and 'profit_per_day'. It initialises 'cagr' and 'profit_per_day' to zero, sets 'profit' and 'unrealised_profit' to zero, and inserts a placeholder transaction with a profit value of 0 to 'transaction_entry_exit'. The 'live_realised_p_l' variable is set to the last profit value in the 'profit' list, which is still 0.

If there have been some transactions (the 'transaction' list is not empty), the code will proceed to handle various circumstances based on the number of transactions and their kind

(buy or sell). The code determines whether the most recent action was a purchase or a sale based on the parity of the number of transactions.

```
if len(data['transaction']) % 2 == 1:


    if sum(sublist[0] for sublist in data['transaction_entry_exit']) == 0:

        if data['transaction'][-1]['action'] == 'Buy':

            data["cagr"].append(0)
            data['initial_price'] = data['transaction'][-1]['transac_price']
            unrealised_profit = sum(data['profit']) -data['initial_price'] + data['prices'][-1]
            profit_per_day = unrealised_profit - sum(data['profit'])
            data['profit'].append(0)
            data['unrealised_profit'].append(unrealised_profit)
            data['transaction_entry_exit'].append([-1, data['transaction'][-1]['action'], data['unrealised_profit'][-1]])
            data["live_unrealised_p_l"] = data['unrealised_profit'][-1]
            data["profit_per_day"].append(profit_per_day)



        if data['transaction'][-1]['action'] == 'Sell':

            data["cagr"].append(0)
            data['initial_price'] = data['transaction'][-1]['transac_price']
            unrealised_profit = sum(data['profit']) + data['initial_price'] - data['prices'][-1]
            profit_per_day = unrealised_profit - sum(data['profit'])
            data['profit'].append(0)
            data['unrealised_profit'].append(unrealised_profit)
            data['transaction_entry_exit'].append([1, data['transaction'][-1]['action'], data['unrealised_profit'][-1]])
            data["live_unrealised_p_l"] = data['unrealised_profit'][-1]
            data["profit_per_day"].append(profit_per_day)
```

If the most recent action was a ‘buy’ and the overall sum of activities in 'transaction_entry_exit' is zero, the portfolio is in an open position. The code computes unrealized profit and profit per day and updates the relevant data columns. It also changes the value of 'live_unrealised_p_l' to the unrealized profit value.

If the last action was a ‘sell’ and the total sum of activities in 'transaction_entry_exit' is 0, it calculates unrealized profit, profit per day, and updates the data fields in the same way.

If the 'transaction_entry_exit' sum is greater than zero, the portfolio is closed. It calculates unrealized profit, profit per day, and changes the required columns in this situation.

```
if len(data['transaction']) % 2 == 0:

    # for buy
    if sum(sublist[0] for sublist in data['transaction_entry_exit']) == -1:

        data["cagr"].append(0)
        unrealised_profit = sum(data['profit']) -data['initial_price'] + data['transaction'][-1]['transac_price']
        profit_per_day = unrealised_profit - sum(data['profit'])
        data['unrealised_profit'].append(unrealised_profit)
        data['transaction_entry_exit'].append([1, data['transaction'][-1]['action'], data['unrealised_profit'][-1]])
        data['profit'].append(unrealised_profit - sum(data['profit']))
        data["live_realised_p_l"] = sum(data['profit'])
        data['profit_after_each_transaction'].append(data["live_realised_p_l"])
        data["live_unrealised_p_l"] = data['unrealised_profit'][-1]
        data["profit_per_day"].append(profit_per_day)


        num_days = (data['transaction'][-1]['transac_timestamp'] - data['transaction'][-2]['transac_timestamp']).days

        if num_days ==0:

            num_days = 1


        data['num_days'].append(num_days)
        N = data['num_days'][-1]/252
        cagr = ((data['x'][-1]+data['x'][-2])/abs(data['x'][-2]) + 1)**(1/N) -1

        data['cagr'][-1] =cagr
        data['each_cagr'].append(cagr)
```

If there are an even number of transactions (indicating a closed position), it calculates realised profit based on the initial price, updates appropriate data, and calculates CAGR based on the time passed between the last two transactions.

```
if len(data['transaction']) % 2 == 1:


    if sum(sublist[0] for sublist in data['transaction_entry_exit']) == 0:

        if data['transaction'][-1]['action'] == 'Buy':

            data["cagr"].append(0)
            data['initial_price'] = data['transaction'][-1]['transac_price']
            unrealised_profit = sum(data['profit']) -data['initial_price'] + data['prices'][-1]
            profit_per_day = unrealised_profit - sum(data['profit'])
            data['profit'].append(0)
            data['unrealised_profit'].append(unrealised_profit)
            data['transaction_entry_exit'].append([-1, data['transaction'][-1]['action'], data['unrealised_profit'][-1]])
            data["live_unrealised_p_l"] = data['unrealised_profit'][-1]
            data["profit_per_day"].append(profit_per_day)



        if data['transaction'][-1]['action'] == 'Sell':

            data["cagr"].append(0)
            data['initial_price'] = data['transaction'][-1]['transac_price']
            unrealised_profit = sum(data['profit']) + data['initial_price'] - data['prices'][-1]
            profit_per_day = unrealised_profit - sum(data['profit'])
            data['profit'].append(0)
            data['unrealised_profit'].append(unrealised_profit)
            data['transaction_entry_exit'].append([1, data['transaction'][-1]['action'], data['unrealised_profit'][-1]])
            data["live_unrealised_p_l"] = data['unrealised_profit'][-1]
            data["profit_per_day"].append(profit_per_day)
```

If the number of transactions is odd (indicating an open position), it computes the unrealized profit and profit per day and updates the data fields.

Based on the transactions recorded in the 'data' structure, the code appears to keep several financial metrics such as 'cagr,' 'profit,' 'unrealised_profit,' and 'profit_per_day'. It handles various scenarios based on the number and kind of transactions, tracking both realised and unrealized profits and calculating CAGR where relevant. It is crucial to note, however, that without a complete grasp of the 'data' structure and how it is managed, it is difficult to provide a complete comprehension of the code's functionality and validity.

2. Stop-Loss

Average True Range (ATR) stop-loss strategy is a risk management technique commonly used in trading to limit potential losses. The Average True Range measures market volatility, helping traders set stop-loss levels that are adjusted to current market conditions. ATR stop-loss has been implemented for both Fibonacci Retracement and TMA strategy, and Bollinger Channel Breakout Strategy.
To implement this strategy, first calculate the first ATR over a specific period, typically 14 periods, using the following formula:

$$n = 14$$
$$ATR = (Sum\ of\ the\ True\ Range\ for\ the\ last\ n\ periods) / n$$

The True Range is the maximum of the following formula:

$$TR = Max[(High - Low), Abs(High - Previous\ Close), Abs(Low - Previous\ Close)]$$

Once you have calculated the first ATR, the next ATR can be calculated using the following formula:

$$n = 14$$
$$ATR = (Previous\ ATR * (n - 1) + TR) / n$$

Once you have calculated the ATR, you can set your stop-loss level at 2 times the ATR below the current market price for a long position or 2 times the ATR above the current market price for a short position. Here's the formula:

$$Stop - Loss\ for\ Long\ Position\ =\ Current\ Market\ Price\ -\ (2\ *\ ATR)$$
$$Stop - Loss\ for\ Short\ Position\ =\ Current\ Market\ Price\ +\ (2\ *\ ATR)$$

Utilizing a 2*ATR stop-loss technique provides a dynamic method for effectively managing risk, as it adjusts the stop-loss level based on the current market volatility. In situations characterized by greater volatility, the Average True Range (ATR) will display larger values, necessitating the use of wider stop-loss levels. Conversely, in markets characterized by lower volatility, the ATR will exhibit smaller values, necessitating the use of tighter stop-loss levels. This method aids traders in effectively managing their risk exposure in relation to current market conditions, a crucial aspect of attaining success in trading.

Part 2B: Result and Discussion

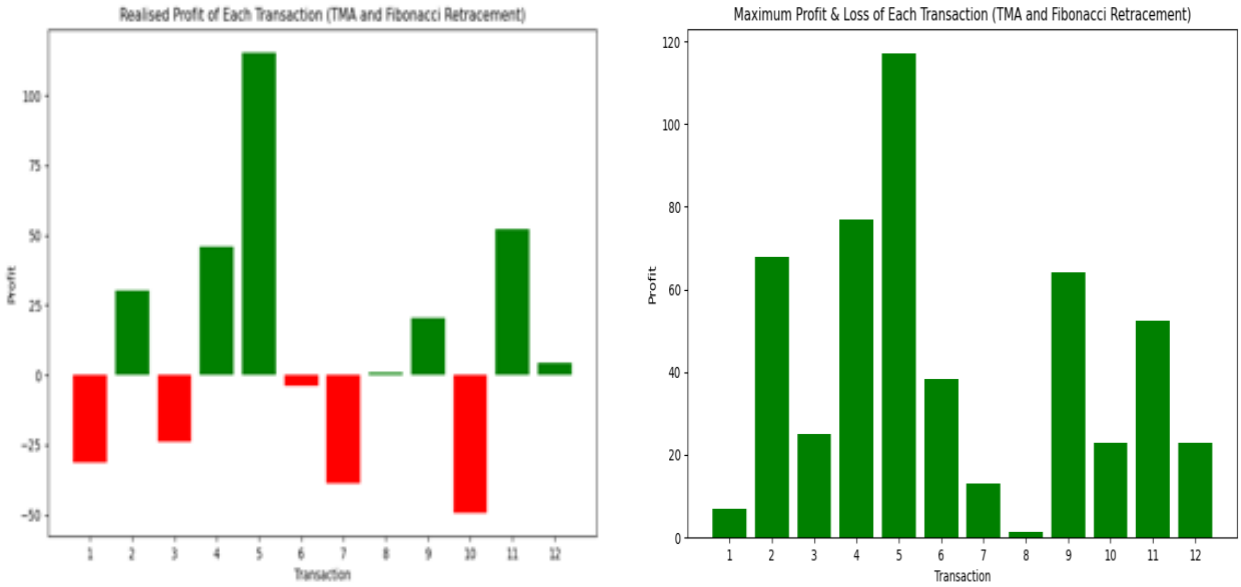
1. Introduction

In Project 2, our group has implemented a trading strategy that combines Fibonacci Retracement with the Triple Moving Average (TMA) strategy and the Bollinger Channel Breakout strategy. The Fibonacci Retracement with TMA Strategy indicate by Fibonacci levels and Simple Moving Averages (SMA). For the Bollinger Channel Breakout strategy., we employ the Upper Bollinger Band, Lower Bollinger Band, and the Simple Moving Average (SMA).

In our trading approach, we exclusively use limit orders to execute trades. To assess the performance of these strategies, we have conducted extensive testing over a one-year period, from August 5, 2022, to August 5, 2023. Our analysis on three financial instruments: Gold spot / USD (XAUUSD), British Pound / U.S. Dollar exchange rate (GBP-USD), and the Standard and Poor's 500 index (S&P Futures).

2. Result on maximum profit and loss and the live P&L on simulator data and test data

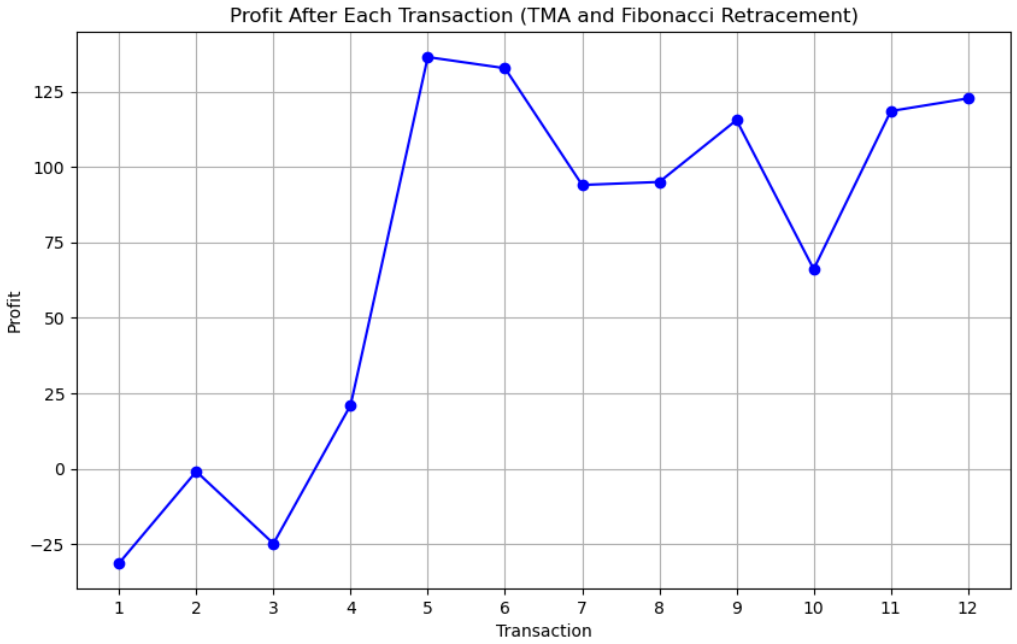
Result of Fibonacci Retracement and TMA Strategy



The diagrams above display the results of the Fibonacci Retracement and TMA strategy for Realised Profit of Each Transaction and Maximum Profit & Loss of Each Transaction. The "XAU_USD" instrument was used to obtain all of these results. Each transaction's realized proceeds exhibit a mixture of positive and negative outcomes. Notably, Transaction 5 generated the greatest realised profit of $115,4, signifying a successful trade. Transaction 11 is also noteworthy for its substantial profit of $52.4. In contrast, Transaction 10 had the largest realised loss of -$49.4, indicating a difficult transaction. This mean we have make sure our principal after trade should more than $49.4 to prevent margin call.

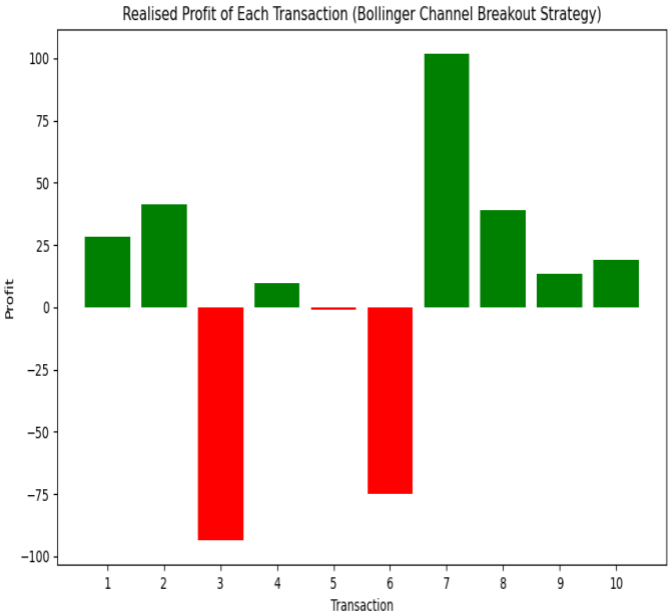
Interesting patterns emerge when we compare these realised profits to the utmost profits and losses for each transaction. Transaction 5, which had the highest realised profit, also had a substantial maximum profit of $117.1, demonstrating that this transaction was maximally profitable according to both Fibonacci Retracement and TMA strategies. In contrast, Transaction 10, which experienced the largest realised loss of -$49.4, generated a maximum profit of $22.8. This indicates that the trade ultimately resulted in a loss, although a 22.8-point profit is possible. Transaction 2 also stands out due to its comparatively realised profit of 30,4 but maximum profit of $67,8, indicating that it could have been even more profitable had it been held for a longer period of time.

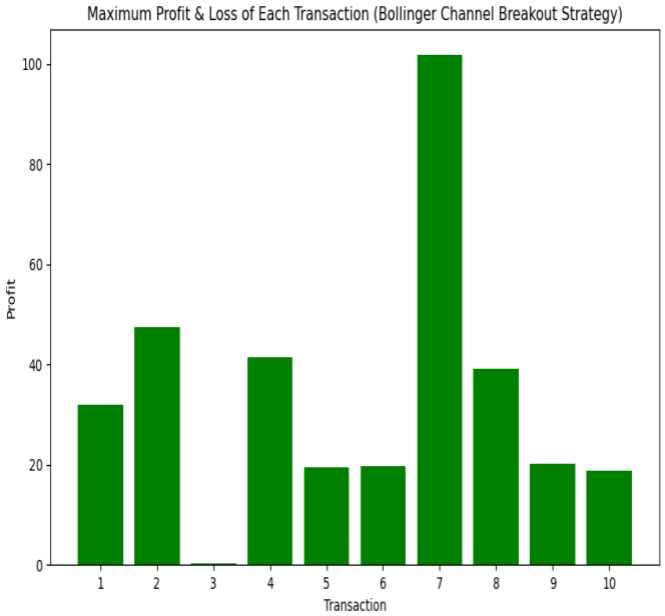
Overall, these comparisons indicate that the Fibonacci Retracement and TMA strategy align reasonably well with the realised gains and losses, with a few exceptions where trades had the potential for more substantial gains or lesser losses



The graphical depiction of profits following each transaction using the Fibonacci Retracement and Triple Moving Average (TMA) strategy with the "XAU_USD" instrument reveals a compelling narrative of trading performance. Initially, the strategy faced adversity, with Transaction 1 resulting in a substantial loss of -$31.3. However, a pivotal turning point emerged with Transaction 4, propelling the total profit from $21.1 to $136.5, marking the most significant improvement, primarily driven by a substantial 10% surge in gold prices over just two weeks in November 2022. However, post Transaction 5, the total profit did not see substantial growth, and it some transaction's profit declined. This phase can be attributed to a gold price downtrend and a period of reduced price volatility from May to August 2023

Result of Bollinger Channel Breakout

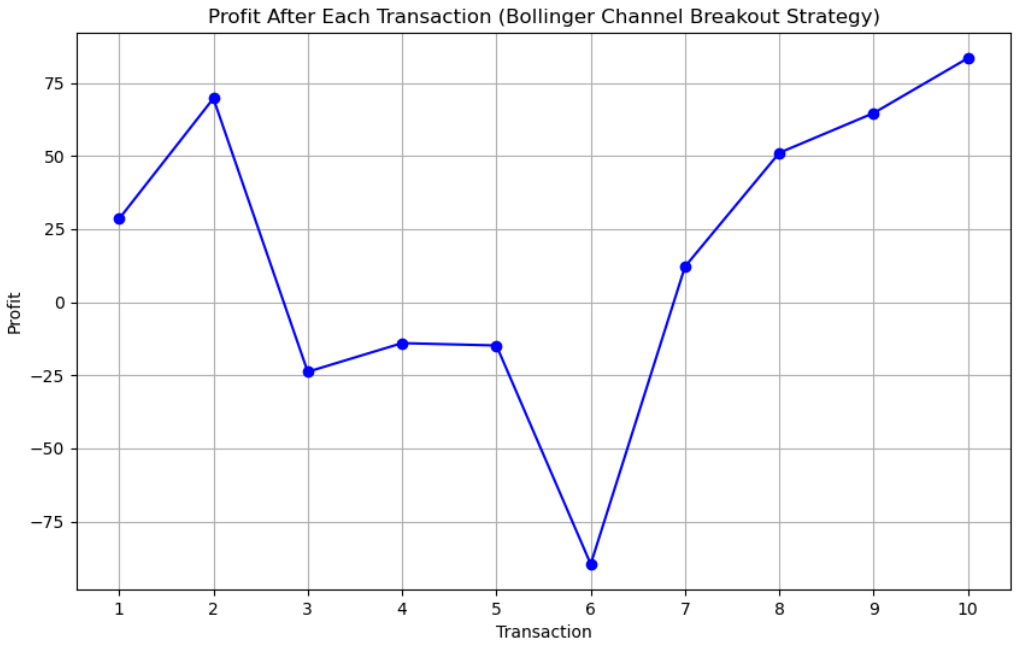




The graphs above display the results of the Bollinger Channel Breakout Strategy for Realised Profit of Each Transaction and Maximum Profit & Loss of Each Transaction. The "XAU_USD" instrument was used to obtain all of these results. To determine the effectiveness of the Bollinger Channel Breakout Strategy, it is essential to compare realised profits to the utmost profit and loss potential for each transaction.

The analysis of individual transactions reveals noteworthy patterns in their outcomes. The first transaction yielded a realized profit of $28.5, slightly below its maximum profit potential of $32.0. Conversely, Transaction 3 incurred a realized loss of -$93.5, falling far short of its potential maximum profit of $0. This highlights a significant risk in Transaction 3 that wasn't effectively managed. Notably, Transaction 7 stands out as a remarkable success, as it achieved its utmost profit potential of 101.8%, showcasing a perfectly executed trade with minimal room for error. Transactions 2, 4, 8, 9, and 10 also produced favourable results, with realized profits closely with their maximum profit potentials. However, Transaction 6 incurred losses exceeding its maximum gain potential. These insights underscore the importance of risk management, as losses like the one in Transaction 3 underscore the need more than $93.5 for capital to prevent margin calls.

Overall, it appears that the Bollinger Channel Breakout Strategy has conflicting results. Some transactions were implemented successfully, maximising profit potential, whereas others involved substantial risks and resulted in substantial losses. It is essential to conduct additional analysis of the strategy, identify patterns or factors contributing to these outcomes, and possibly refine the approach to improve its overall performance and risk management.



The above graphs illustrate the Profit after Each Transaction obtained using the Bollinger Channel Breakout Strategy. The "XAU_USD" instrument is used to obtain the results. The ten transactions analysed disclose a mixed financial performance. Over the duration of these transactions, the total profit experienced fluctuations, with both gains and losses.

Transaction 1 kicked off with a profit of $28.5, initially showing profitability. However, the subsequent Transactions 2 through 6 resulted in consecutive losses, accumulating to -$150.7, effectively nullifying the initial gains due to a misjudgement in identifying the gold trend in November 2022. Transaction 7 marked a turning point with a gain of $12.2, signalling the start of a recovery phase. This positive momentum continued with Transactions 8, 9, and 10, collectively amassing a total profit of $83.5. These subsequent transactions not only offset the prior losses but also generated a substantial profit. This shift can be attributed to significant fluctuations in gold prices, which dropped by around 10% and subsequently rose by a similar

margin in January to March 2023, highlighting the influence of market dynamics on trading outcomes. The final profit is $ 83.5.

The issue of realized losses exceeding the maximum gain potentials in our trading strategy. This situation can arise during highly volatile market conditions, especially when big news events trigger intense emotional responses from market participants. In such scenarios, prices can experience extreme fluctuations, which may trigger our cut loss conditions, leading to an exit from the market. However, it's important to acknowledge that, there can be still profitable opportunities .

3. Performance comparison with strategy in Project 1

Comparison of strategy performance with projects 1 and 2

Project 2						
	Fibonacci Retracement (14) + TMA Strategy (15, 30, 35)			Bollinger Channel Breakout strategy.		
Instrument	Total Profit After Stimulation	Win Rate	number of transaction	Total Profit After Stimulation	Win Rate	number of transaction
XAU_USD	-0.97%	50.00%	24	0.64 %	60%	13
S&P_Futures	26.50%	81.82%	22	13.82386	85.74%	14
GBP-USD	12.55%	77.78%	36	5.71%	60%	10

Project 1						
	Fibonacci Retracement			Turtle Trading		
Instrument	Total Profit After Stimulation	Win Rate	number of transaction	Total Profit After Stimulation	Win Rate	number of transaction
XAU_USD	-7.57%	33.33%	9	-2.95%	28.57%	14
S&P_Futures	9.09%	57.89%	19	5.81%	30.77%	13
GBP-USD	2.25%	46.67%	15	-10.41%	21.74%	23

From the data presented in the table, it has proof that Project 2 outperforms Project 1 in several significant ways. The initial principal calculated is the price we first transaction. There are several reasons we assumed these:

1) **Limit Order Type**:

Project 2 uses limit order types, a strategy that aims to secure more favourable transaction prices by actively seeking the best available price for each trade. It will bring benefits when a trade. When comparing results between limit and market order types, it becomes apparent that limit orders yield higher profits.

2) **Implementation of Cut Loss Conditions**:

In Project 1, the turtle trading strategy did not set cut loss conditions. When losses occurred, the strategy remained in the market, waiting for the next trading signal before exiting. In contrast, Project 2, particularly the Fibonacci Retracement has integrated cut loss conditions. While overall performance is not good, it at least limits the extent of losses.

3) **Combination of Trading Strategies**:

Project 2 benefits from a combination of two strategies, namely Fibonacci Retracement and TMA. This combination proves to be more effective. In Project 1, Fibonacci Retracement's performance suffered because the strategy couldn't identify trends before entering the market. Lack of trend identification occasionally resulted in entering long positions during downtrends due to meet entry-long conditions.

4. Discussion on the practicability of the designed strategy

Furthermore, we try to identify the parameters can maximize profitability when employing the combination of Fibonacci Retracement with TMA and the Bollinger Channel Breakout strategy. In the Fibonacci Retracement with TMA combination, we experimented with two sets of parameters: the first set included SMAs of 7, 14, and 30 days, with a Fibonacci Retracement lookback of 14 days, while the second set is SMAs of 14, 30, and 45 days, with a Fibonacci Retracement lookback of 30 days. As part of the Bollinger Channel Breakout strategy testing, we examined to use 1 time and 2 times the standard deviation for calculating the bands to test performance.

Additionally, we are interested in determining which financial instruments were best suited for each trading strategy. Testing the practicality of the Fibonacci Retracement with TMA Strategy and the Bollinger Channel Breakout Strategy, we have established two distinct sets of variables for experimentation.

1. Fibonacci Retracement with TMA Strategy:

Set 1: Short MA: 7 days, Middle MA: 14 days, Long MA: 35 days,
and Fibonacci Retracement lookback period: 14 days.

Set 2: Short MA: 15 days, Middle MA: 30 days, Long MA: 45 days,
and Fibonacci Retracement lookback period: 30 days.

2. Bollinger Channel Breakout Strategy:

Set 1 : 1 times the standard deviation.

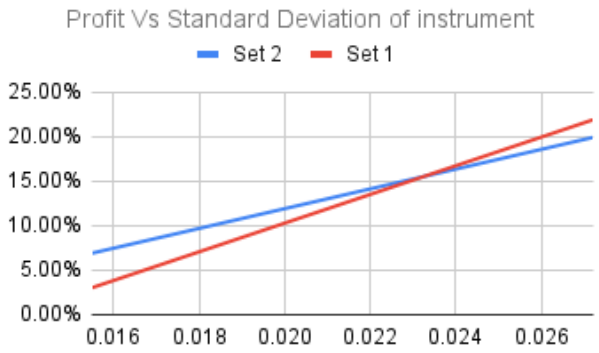
Set 2 : 2 times the standard deviation.

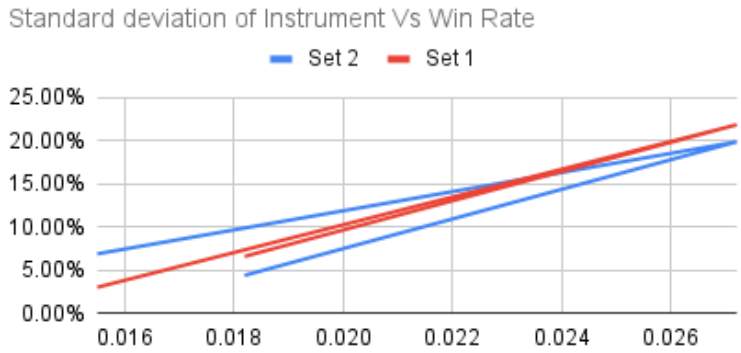
Fibonacci Retracement and TMA strategy

	Fibonacci Retracement (14) + TMA Strategy (7, 14, 35)			Fibonacci Retracement(30) + TMA Strategy (15, 30, 45)			
Instrument	Total Profit After Stimulation	Win Rate	number of transaction	Total Profit After Stimulation	Win Rate	number of transaction	Monkey Profit
XAU_USD	6.58%	41.18%	34	4.37%	50.00%	40	10.52%
S&P_Futures	21.87%	78.57%	28	19.89%	80.00%	20	11.26%
GBP-USD	2.99%	64.29%	28	6.87%	60.00%	20	4.50%

From the table, the combination trading strategy is not incurring losses. However, we cannot determine it was good trading strategy. Profitability alone doesn't provide the complete picture because we must consider the initial capital invested and opportunity costs.

Comparatively, when we test the performance of the "XAU_USD" and "GBP-USD" instruments in set 1 of testing, the strategy may not be advantageous. This is evident when compared to "monkey profit". "Monkey profit" approach where traders buy at the start of the trading duration and sell at the end. The lower performance in these instruments, particularly the win rate below 50% in "XAU_USD. A win rate below 50% implies that half or fewer of the transactions are profitable, leading to additional costs in terms of brokerage fees and missed opportunities for better trades. Therefore, while profitability is important, it should be evaluated the overall effectiveness of a trading strategy.



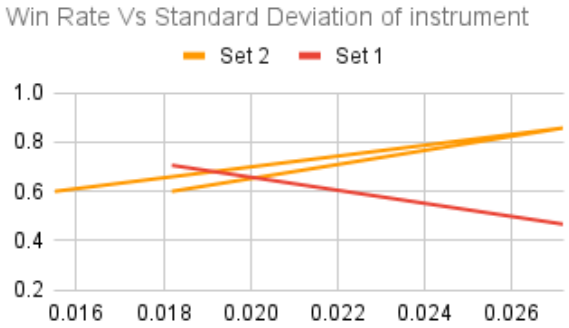


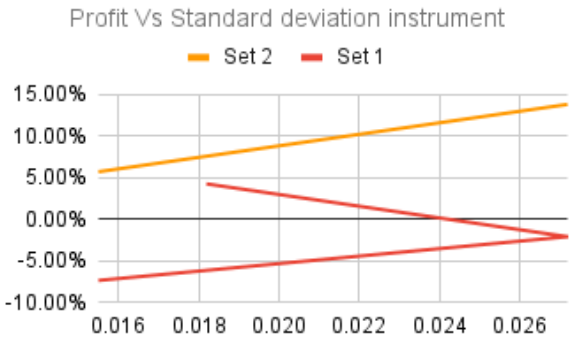
To gain insights into the relationship between financial instruments, we have employed statistical measures such as the standard deviation of instruments and have analysed their correlation with our trading strategy. However, from the graph, it is apparent that discernible linear relationships between profit and fluctuation of the instrument but the win rate vs. fluctuation of the instrument shows a critical point to reverse. However, the results may be due to a lack of observable datasets. It seems that more data may be required to draw conclusive insights.

Bollinger Channel Breakout strategy

	Bollinger Channel Breakout strategy (1 of standard deviation)			Bollinger Channel Breakout Band strategy (2 of standard deviation)		
Instrument	Total Profit After Stimulation	Win Rate	number of transaction	Total Profit After Stimulation	Win Rate	number of transaction
XAU_USD	4.25%	53.85 %	13	0.64 %	60%	10
S&P_Futures	-2.12%	70.59%	17	13.82%	85.74%	14
GBP-USD	-7.39%	46.67 %	15	5.71%	60%	10

Based on the data in the table, using 1 time the standard deviation does not yield favorable results when compared to using 2 times the standard deviation. This discrepancy in performance might be by the fluctuation of financial instruments. According to our observation, we've noticed a trend where increasing the standard deviation leads to a decrease in both the win rate and profit. However, it is occurred in test set 1 only. We hypothesize that using 1 time the standard deviation narrows the Bollinger Channel Breakout strategy, potentially causing the indicators to lose their predictive effectiveness.





In conclusion, the S&P 500 futures deliver the highest win rates and profit margins across both trading strategies. In the second position, Forex trading with GBP-USD, demonstrating competitive performance. However, the lowest-performing asset appears to be Gold (XAU-USD). The reason of gold performed badly are hard to research and obverse. Our assumption is that commodities like Gold are inherently complex due to many unpredictable factors such as weather events and geopolitical unrest, which impacts on prices. In contrast, equities (S&P 500) and Forex (GBP-USD) may exhibit more stable performance because they are influenced to a lesser variable and are more closely tied to economic and policy-related factors.

5. Effects of price slippage to the designed strategy

Price slippage will have an impact on Entry and Exit Points. Slippage might result in the execution of the transactions at less favourable prices than anticipated. This implies that when using the trading strategy approach, the entry and exit positions can differ from the levels determined using the indicator. Slippage might result in greater long-position entrance prices and lower short-position exit prices. Besides, slippage might make the plan less profitable. The potential profit can be lower than expected if starting a long position at a higher price than anticipated or ending a short position at a lower price than anticipated. This is particularly important when dealing with illiquid assets or in volatile markets. In addition, losses will also be increased. If the approach has slippage, the losses may rise. There is a chance that slippage will occur when executing stop-loss orders, causing to sustain more losses than anticipated. Consequently, trading accounts may see a bigger drop.

To reduce the consequences of slippage, traders frequently use risk management tactics like as establishing stop-loss and take-profit levels, placing limit orders, and trading during periods of greater liquidity. Making wise trading selections by testing the approach with a realistic slippage model and keeping an eye on slippage in real time is a way to mitigate the effects of slippage. Slippage must be taken into account when assessing the possible profitability and risk of the trading plan.

In our project, we encountered price slippage that affected our profit. Through our observations, we discerned that limit order types outperformed market order types. There are two primary reasons for this superiority. Firstly, limit order types aim to secure more favourable transaction prices, actively seeking the best available price for the trade. Secondly, limits are instrumental in mitigating the problem of price slippage, as they allow us to set a specific price at which we are willing to execute the trade.

Below table is a comparison of limit order types and market order types under Fibonacci Retracement and TMA strategy.

	XAU_USD	S&P_Futures	GBP-USD
	Total Profit After Stimulation	Total Profit After Stimulation	Total Profit After Stimulation
Limit order type	30.7	997.58	0.137838
Market order type	-134.67	856.61	0.017709