Revisiting the Performance of MACD and RSI Oscillators

วัตถุประสงค์ของงานวิจัย

เพื่อทดสอบประสิทธิภาพ และความสามารถในการทำกำไรในการวิเคราะห์การลงทุนทางเทคนิค (Technical Analysis) เปรียบเทียบกับกลยุทธ์การลงทุนแบบซื้อและถือ (Buy and Hold)

เพื่อตรวจสอบหาประสิทธิภาพในการทำกำไรจากดัชนีตลาดหุ้นในเครือประเทศแถบเอเชีย ว่าวิธีไหนมีประสิทธิภาพในการทำกำไรที่ดีกว่ากัน โดยใช้การเขียนโปรแกรมไพทอน (Python) เพื่อทดสอบประสิทธิภาพในการลงทุน

Top 10 Asian stock Exchanges

01 **^BSESN**
Bombay Stock Exchange, India

02 **^N225**
Tokyo Stock Exchange, Japan

03 **^HSI**
Hong Kong Stock Exchange, Hong Kong

04 **399001.SZ**
Shenzhen Stock Exchange, China

05 **000001.SS**
Shanghai Stock Exchange, China

06 **^NSEI**
National Stock Exchange, India

07 **^KS11**
Korea Exchange, South Korea

08 **^TWII**
Taiwan Stock Exchange, Taiwan

09 **^STI**
Singapore Exchange, Singapore

10 **^SET.BK**
The Stock Exchange of Thailand, Thailand

Process

กระบวนการจัดการ

ขั้นตอน 1

Import Library &
Import Daily Index price

ขั้นตอน 2

Create Buy and Hold Strategy

ขั้นตอน 3

Create MACD & RSI Strategy

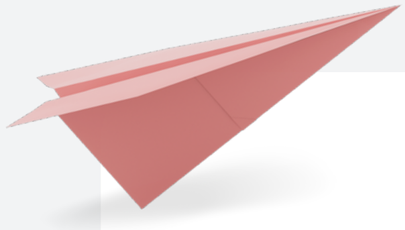
- MACD Rule 1
- MACD Rule 2
- RSI Rule 3
- RSI Rule 4

ขั้นตอน 5

Empirical result

ขั้นตอน 4

Parameter Tuning for Trading Strategy



01 Import Library & Import Daily Index price

Import Library

```
# Import Libraries
import numpy as np
import pandas as pd
from datetime import datetime, timedelta # Import datetime for date handling
import matplotlib.pyplot as plt
from scipy import stats
import yfinance as yf # Import yfinance for downloading financial data

import time # for get runtime
```

Setting Date

```
# Set the end date to September 30, 2023.
end_date = datetime(2023, 9, 30)

# Calculate the start date by subtracting 10 years from the end date.
start_date = end_date - timedelta(days=3652)

print("Start Date is", start_date)
print("End Date is", end_date)
```

```
Start Date is 2013-09-30 00:00:00
End Date is 2023-09-30 00:00:00
```

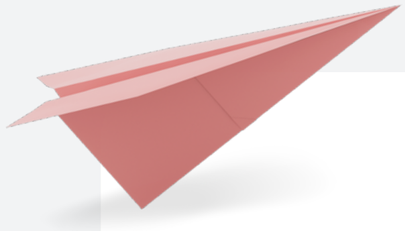
Setting Symbols & Import Daily Index price

```python
# Define a list of stock market INDEX symbols for the top 10 Asian stock exchanges in 2023
index_symbols = [
    "^BSESN", # Bombay Stock Exchange, India
    "^N225", # Tokyo Stock Exchange, Japan
    "^HSI", # Hong Kong Stock Exchange, Hong Kong
    "399001.SZ", # Shenzhen Stock Exchange, China
    "000001.SS", # Shanghai Stock Exchange, China
    "^NSEI", # National Stock Exchange, India
    "^KS11", # Korea Exchange, South Korea
    "^TWII", # Taiwan Stock Exchange, Taiwan
    "^STI", # Singapore Exchange, Singapore
    "^SET.BK" # The Stock Exchange of Thailand, Thailand
]

# Define a list of stock market INDEX Names for the top 10 Asian stock exchanges in 2023.
index_names = [
    "Bombay Stock Exchange, India",
    "Tokyo Stock Exchange, Japan",
    "Hong Kong Stock Exchange, Hong Kong",
    "Shenzhen Stock Exchange, China",
    "Shanghai Stock Exchange, China",
    "National Stock Exchange, India",
    "Korea Exchange, South Korea",
    "Taiwan Stock Exchange, Taiwan",
    "Singapore Exchange, Singapore",
    "The Stock Exchange of Thailand, Thailand"
]
```

```python
# Download the daily prices for the INDEX symbols
def import_financial_data(symbols, price=type_price):
    dict = {}
    for symbol in symbols:
        data = yf.download(symbol, start = start_date, end = end_date)
        dict[symbol] = data[[type_price]]
        print(f"Import {symbol} Index completed")
    return dict

index_price_dict = import_financial_data(index_symbols, price=type_price)
```

	Adj Close
Date	
2013-09-30	19379.769531
2013-10-01	19517.150391
2013-10-03	19902.070312
2013-10-04	19915.949219
2013-10-07	19895.099609

```python
index_price_dict["^BSESN"].head()
```



02 Create Buy and Hold Strategy

```python
def buy_and_hold_log_return(index_price_df,
                            price=type_price):

    # Copy the index price column from the input DataFrame.
    price_df = index_price_df.copy()

    # Calculate the log returns and removing NaN values.
    log_return_df = pd.DataFrame(np.log(price_df[price] / price_df[price].shift(10))).dropna()
    log_return_df.columns = ["log_return"]

    # Merge the calculated log returns back to the original DataFrame based on index
    price_result_df = pd.merge(price_df, log_return_df, how="left", left_index=True, right_index=True)
    price_result_df.dropna(inplace=True)

    return price_result_df
```

```python
# Create an empty dictionary called 'buy_and_hold_dict' to store log returns for different symbols.
buy_and_hold_dict = {}

# Iterate through each symbol and index price.
for symbol, index_price_df in index_price_dict.items():
    # Calculate and store the log returns for the symbol in the 'buy_and_hold_dict' dictionary.
    buy_and_hold_dict[symbol] = buy_and_hold_log_return(index_price_df)
    print(f"Complete calculate log return of '{symbol}'")
```

Date	Adj Close	log_return
2013-10-15	20547.619141	0.058515
2013-10-17	20415.509766	0.045001
2013-10-18	20882.890625	0.048106
2013-10-21	20893.890625	0.047936
2013-10-22	20864.970703	0.047598

```python
buy_and_hold_dict["^BSESN"].head()
```

Create Buy and Hold Strategy

```python
def calculate_buy_and_hold_summary_stat(buy_and_hold_dict,
                                        symbols=index_symbols,
                                        names=index_names):

    # Create an summary stat table as a dictionary and store top 10 Asian stock exchanges symbol.
    summary_stat_table = {
        "Symbol": symbols,  # Column for index symbols
        "Sample Period (2013-2023)": names #  Column for index names
    }

    # Convert the dictionary to a Pandas DataFrame
    summary_stat_table = pd.DataFrame(summary_stat_table)

    # Iterate through the dictionary of index data for each symbol.
    for symbol, buy_and_hold_df in buy_and_hold_dict.items():

        buy_and_hold_df = buy_and_hold_df.dropna()

        # Find the row in the summary table where 'Symbol' matches the current symbol.
        condition = summary_stat_table["Symbol"] == symbol

        # Calculate the mean of log returns and store it in the DataFrame
        summary_stat_table.loc[condition, "Mean"] = round(buy_and_hold_df["log_return"].mean(), 5)

        # Calculate the standard deviation of log returns and store it in the DataFrame
        summary_stat_table.loc[condition, "S.D."] = round(buy_and_hold_df["log_return"].std(), 5)

        # Perform a skewness test and store skewness with ** if p-value of 2-tails < 0.05
        skew = str(round(stats.skew(buy_and_hold_df["log_return"]), 4))
        skew_pvalue = stats.skewtest(buy_and_hold_df["log_return"], alternative="two-sided").pvalue
        summary_stat_table.loc[condition, "Skewness"] = (skew + "**") if (skew_pvalue < 0.05) else (skew)

        # Perform a kurtosis test and store kurtosis with ** if p-value of 2-tails < 0.05
        kurt = str(round(stats.kurtosis(buy_and_hold_df["log_return"]), 4))
        kurt_pvalue = stats.kurtosistest(buy_and_hold_df["log_return"], alternative="two-sided").pvalue
        summary_stat_table.loc[condition, "Kurtosis"] = (kurt + "**") if (kurt_pvalue < 0.05) else (kurt)

    # Set the "Symbol" column to index names.
    summary_stat_table.set_index("Symbol", inplace=True)

    return summary_stat_table
```

Define the function to Create Summary Statistics Table for Buy and Hold Strategy

Symbol	Sample Period (2013–2023)	Mean	S.D.	Skewness	Kurtosis
^BSESN	Bombay Stock Exchange, India	0.00490	0.03417	-1.9983**	16.9082**
^N225	Tokyo Stock Exchange, Japan	0.00342	0.03836	-0.7892**	3.8005**
^HSI	Hong Kong Stock Exchange, Hong Kong	-0.00107	0.04012	-0.2896**	1.7275**
399001.SZ	Shenzhen Stock Exchange, China	0.00065	0.05323	-1.0859**	6.5168**
000001.SS	Shanghai Stock Exchange, China	0.00142	0.04305	-1.1281**	6.916**
^NSEI	National Stock Exchange, India	0.00492	0.03422	-2.0256**	16.9649**
^KS11	Korea Exchange, South Korea	0.00095	0.03294	-1.3682**	13.2264**
^TWII	Taiwan Stock Exchange, Taiwan	0.00283	0.03187	-1.1064**	6.3968**
^STI	Singapore Exchange, Singapore	0.00008	0.02954	-1.5724**	13.5074**
^SET.BK	The Stock Exchange of Thailand, Thailand	0.00023	0.03115	-1.5286**	14.5991**

```
buy_and_hold_summary_stat.to_excel("buyhold.xlsx")
```

03 Create MACD & RSI Strategy

Moving Average Convergence Divergence (MACD)

MACD เป็น Indicators ซึ่งใช้วัดโมเมนตัมและด้วยกลยุทธ์เกี่ยวกับการเคลื่อนที่ตามแนวโน้ม โดยถูกคำนวณจากค่าเฉลี่ยของราคา ซึ่งประกอบด้วย 2 ส่วน คือ

MACD Main

$$MACD_p = EMA_{n1}(p) - EMA_{n2}(p)$$

Signal Line

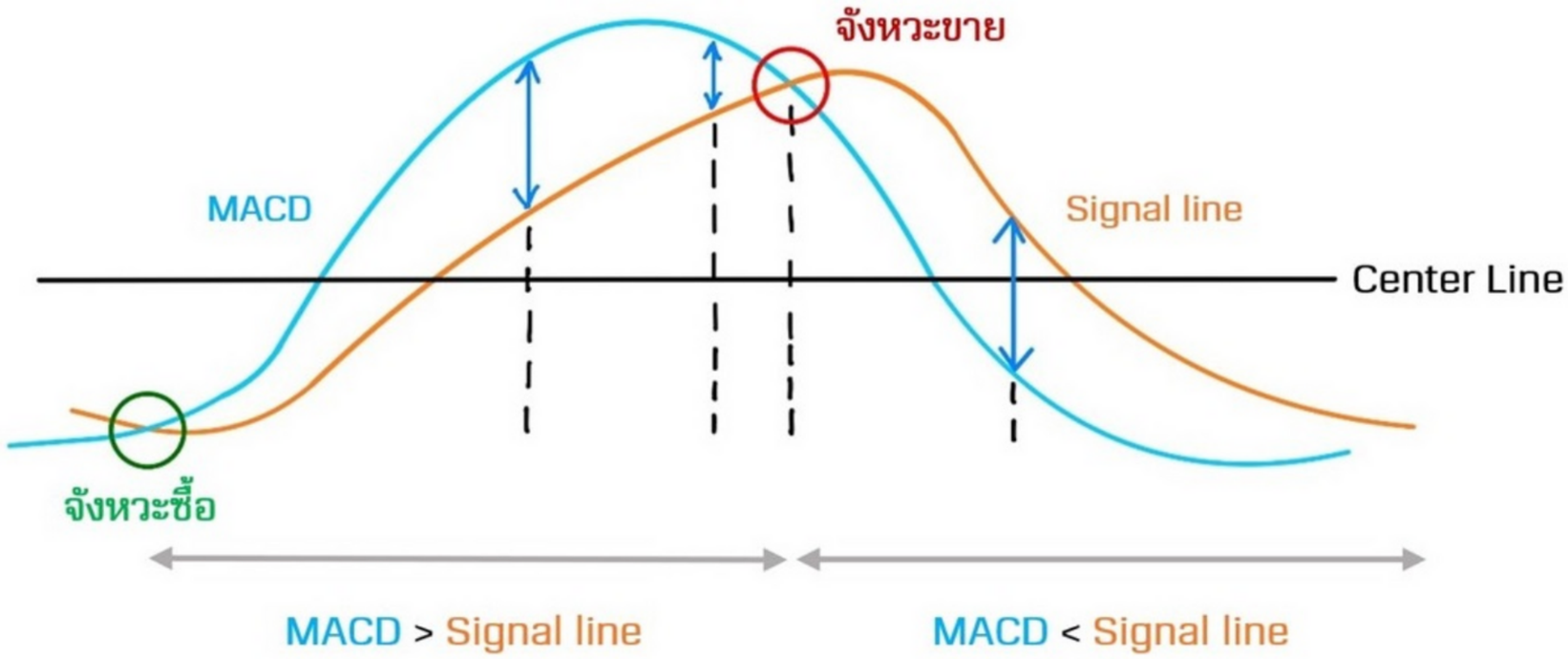
$$S_{MACD} = EMA_n(MACD)$$

MACD = EMA (P,N1) – EMA (P,N2) โดยที่ N1 < N2 หรือสามารถเขียนได้ว่า MACD = Short EMA – Long EMA

$$EMA_t = \frac{x_t + (1+\alpha)\,x_{t-1} + (1-\alpha)^2\,x_{t-2} + \ldots\ldots + (1-\alpha)^t\,x_0}{1 + (1-\alpha) + (1+\alpha)^2 + \ldots\ldots + (1+\alpha)^t}$$

โดยที่ Smoothing $(\alpha) = \frac{2}{(N+1)}$

Moving Average Convergence Divergence (MACD)



MACD Rule 1

Buy Signal (Long Position) : เมื่อ MACD Main ตัดเส้น 0 สูงขึ้นมาและกลับขึ้นไปสูงกว่า

Sell Signal (Short Position) : เมื่อ MACD Main ตัดเส้น 0 ลงมาและอยู่ต่ำกว่า

ซึ่งกฎของ MACD กฎนี้ คือการใช้ MACD (N,M,0) โดยปกติทีนิยมใช้จะเป็น MACD (12,26,0)

MACD Rule 2

Buy Signal (Long Position) : เมื่อ MACD ตัดเส้น Signal Line จากด้านล่างตัดขึ้นมา

Sell Signal (Short Position) : เมื่อ MACD ตัดเส้น Signal Line จากด้านบนลงมา

ซึ่งกฎของ MACD กฎนี้ คือการใช้ MACD (N,M,9) , ซึ่ง Signal Line = EMA(9)

Define the functionto calculate the MACD

```
# Function to calculate the Moving Average Convergence Divergence (MACD)
def calculate_macd(index_price_df,
                   period_short = 12,
                   period_long = 26,
                   signal_line = 0):

    ##########################################################################
```

Calculate MACD from EMA short and EMA long

```
### Calculate MACD from EMA short and EMA long

# Create a copy of the input DataFrame
MACD_df = index_price_df.copy()

# Calculate MACD from EMA short (Shorter-period) and EMA long (Longer-period)
MACD_df["EMA_short"] = index_price_df[type_price].ewm(span=period_short,
                                                      adjust=True,
                                                      min_periods=period_short).mean()
MACD_df["EMA_long"] = index_price_df[type_price].ewm(span=period_long,
                                                     adjust=True,
                                                     min_periods=period_long).mean()

MACD_df["MACD"] = MACD_df["EMA_short"] - MACD_df["EMA_long"]
MACD_df.drop(["EMA_short", "EMA_long"], inplace=True, axis=1)
```

```
### Get Signal of MACD Rule 1 and MACD Rule 2

# MACD Rule 1: MACD crosses zero
if signal_line == 0:
    MACD_cross_zero = MACD_df.copy().dropna()
    MACD_cross_zero = MACD_cross_zero[["MACD"]]

    # Create new column for check previous day
    MACD_cross_zero.loc[:,"MACD_shift"] = MACD_cross_zero["MACD"].shift(1)

    # Update Buy signal: MACD crosses zero from below
    from_below = (MACD_cross_zero["MACD_shift"] < 0) & (MACD_cross_zero["MACD"] >= 0)
    MACD_cross_zero.loc[:,"buy_signal"] = np.where(from_below, 1, 0)

    # Update Sell signal: MACD crosses zero from above
    from_above = (MACD_cross_zero["MACD_shift"] > 0) & (MACD_cross_zero["MACD"] <= 0)
    MACD_cross_zero.loc[:,"sell_signal"] = np.where(from_above, 1, 0)

    # Drop MACD shift and Join table
    MACD_cross_zero.drop(["MACD", "MACD_shift"], axis=1, inplace=True)
    MACD_result_df = pd.merge(MACD_df, MACD_cross_zero, how="left", left_index=True, right_index=True)
```

Rule 1 Get Signal of MACD Rule 1 and MACD Rule 2

```
# MACD Rule 2: MACD crosses the signal_line-day EMA of the MACD
elif signal_line > 0:
    EMA_crossover = MACD_df.copy().dropna()
    EMA_crossover = EMA_crossover[["MACD"]]
    EMA_crossover["EMA_of_MACD"] = EMA_crossover["MACD"].ewm(span=signal_line,
                                                             adjust=True,
                                                             min_periods=signal_line).mean()

    # Create new column for check previous day
    EMA_crossover.loc[:,"MACD_shift"] = EMA_crossover["MACD"].shift(1)
    EMA_crossover.loc[:,"EMA_of_MACD_shift"] = EMA_crossover["EMA_of_MACD"].shift(1)

    # Update Buy signal: MACD crosses EMA of the MACD from below
    from_below = (EMA_crossover["MACD_shift"] < EMA_crossover["EMA_of_MACD_shift"])
                 & (EMA_crossover["MACD"] >= EMA_crossover["EMA_of_MACD"])
    EMA_crossover.loc[:,"buy_signal"] = np.where(from_below, 1, 0)

    # Update Sell signal: MACD crosses EMA of the MACD from above
    from_above = (EMA_crossover["MACD_shift"] > EMA_crossover["EMA_of_MACD_shift"])
                 & (EMA_crossover["MACD"] <= EMA_crossover["EMA_of_MACD"])
    EMA_crossover.loc[:,"sell_signal"] = np.where(from_above, 1, 0)

    # Drop MACD and Join table
    EMA_crossover.drop(["MACD", "MACD_shift", "EMA_of_MACD_shift"], axis=1, inplace=True)
    MACD_result_df = pd.merge(MACD_df, EMA_crossover, how="left", left_index=True, right_index=True)

else:
    raise ValueError("The signal_line must be greater than or equal 0.")
```

Rule 2 Get Signal of MACD Rule 1 and MACD Rule 2

```
calculate_macd(index_price_df=index_price_dict["^BSESN"])
```

Date	Adj Close	MACD	buy_signal	sell_signal
2013-09-30	19379.769531	NaN	NaN	NaN
2013-10-01	19517.150391	NaN	NaN	NaN
2013-10-03	19902.070312	NaN	NaN	NaN
2013-10-04	19915.949219	NaN	NaN	NaN
2013-10-07	19895.099609	NaN	NaN	NaN
...	...	...	...	...
2023-09-25	66023.687500	274.871140	0.0	0.0
2023-09-26	65945.468750	206.763344	0.0	0.0
2023-09-27	66118.687500	164.864278	0.0	0.0
2023-09-28	65508.320312	81.468319	0.0	0.0
2023-09-29	65828.406250	40.735111	0.0	0.0

2456 rows × 4 columns

```
calculate_macd(index_price_df=index_price_dict["^BSESN"], signal_line=9)
```

Date	Adj Close	MACD	EMA_of_MACD	buy_signal	sell_signal
2013-09-30	19379.769531	NaN	NaN	NaN	NaN
2013-10-01	19517.150391	NaN	NaN	NaN	NaN
2013-10-03	19902.070312	NaN	NaN	NaN	NaN
2013-10-04	19915.949219	NaN	NaN	NaN	NaN
2013-10-07	19895.099609	NaN	NaN	NaN	NaN
...	...	...	...	...	...
2023-09-25	66023.687500	274.871140	365.287518	0.0	0.0
2023-09-26	65945.468750	206.763344	333.582683	0.0	0.0
2023-09-27	66118.687500	164.864278	299.839002	0.0	0.0
2023-09-28	65508.320312	81.468319	256.164865	0.0	0.0
2023-09-29	65828.406250	40.735111	213.078915	0.0	0.0

2456 rows × 5 columns

Date	Adj Close	MACD	buy_signal	sell_signal
2013-11-07 00:00:00	20822.76953125	131.745651973426	0.0	0.0
2013-11-08 00:00:00	20666.150390625	106.31284027910078	0.0	0.0
2013-11-11 00:00:00	20490.9609375	72.71612610730153	0.0	0.0
2013-11-12 00:00:00	20281.91015625	30.58747918187146	0.0	0.0
2013-11-13 00:00:00	20194.400390625	-9.011605015275563	0.0	1.0
2013-11-14 00:00:00	20399.419921875	-24.840687552939926	0.0	0.0
2013-11-18 00:00:00	20850.740234375	-3.699468193746725	0.0	0.0
2013-11-19 00:00:00	20890.8203125	15.835540167536237	1.0	0.0
2013-11-20 00:00:00	20635.130859375	11.887080225271347	0.0	0.0
2013-11-21 00:00:00	20229.05078125	-21.730632215902006	0.0	1.0
2013-11-22 00:00:00	20217.390625	-48.65373829422242	0.0	0.0
2013-11-25 00:00:00	20605.080078125	-39.87516999387299	0.0	0.0
2013-11-26 00:00:00	20425.01953125	-46.20322249328092	0.0	0.0
2013-11-27 00:00:00	20420.259765625	-50.98726113168232	0.0	0.0
2013-11-28 00:00:00	20534.91015625	-45.388252288550575	0.0	0.0
2013-11-29 00:00:00	20791.9296875	-20.787296411454008	0.0	0.0
2013-12-02 00:00:00	20898.009765625	6.8644115569441055	1.0	0.0
2013-12-03 00:00:00	20854.919921875	25.113852392812987	0.0	0.0
2013-12-04 00:00:00	20708.7109375	27.8167241061783	0.0	0.0
2013-12-05 00:00:00	20957.810546875	48.909948935823195	0.0	0.0
2013-12-06 00:00:00	20996.529296875	67.87363554299009	0.0	0.0
2013-12-09 00:00:00	21326.419921875	107.60929455354199	0.0	0.0
2013-12-10 00:00:00	21255.259765625	131.951920859432	0.0	0.0
2013-12-11 00:00:00	21171.41015625	142.96179675818348	0.0	0.0
2013-12-12 00:00:00	20925.609375	130.73019905703404	0.0	0.0

```
macd_dict_rule1["^BSESN"]
```

```python
def adjust_signal(indicator_df):

    # Check if there is a buy and sell signal
    # Replace the next 10 rows in a buy_signal and sell_signal columns with 0 (no signal)
    for i in range(0,len(indicator_df)-10):
        if (indicator_df["buy_signal"].iloc[i] == 1) | (indicator_df["sell_signal"].iloc[i] == 1):
            """ Reset buy and sell signals in the next 10 days """
            indicator_df["buy_signal"].iloc[i+1:i+11] = 0
            indicator_df["sell_signal"].iloc[i+1:i+11] = 0

    # Replace the last 10 rows in a buy_signal and sell_signal columns with 0
    indicator_df["buy_signal"].iloc[len(indicator_df)-10:len(indicator_df)] = 0
    indicator_df["sell_signal"].iloc[len(indicator_df)-10:len(indicator_df)] = 0

    return indicator_df
```

Define the function Don't buy/sell for 10 days after the buy/sell signal for Indicator Trading Rules

```python
# Define a function to calculate strategy log returns
def calculate_trading_rules_log_return(indicator_df,
                                       price=type_price):

    # Create a copy of the input DataFrame and remove any rows with missing values
    log_return_df = indicator_df.copy()

    # log_return_df.dropna(inplace=True) # don't use this function because it causes a warning.
    log_return_df = log_return_df.loc[~log_return_df.isna().all(axis=1)] # drop by any column (by .all) isna = True

    # Shift data 10 days
    log_return_df["price_shift"] = log_return_df[price].shift(-10)

    # Calculate Buy and Sell Log Returns
    log_return_df.loc[:,"buy_log_return"] = np.log(log_return_df["price_shift"] / log_return_df[price]) * log_return_df["buy_signal"]
    log_return_df.loc[:,"sell_log_return"] = np.log(log_return_df["price_shift"] / log_return_df[price]) * (-1 * log_return_df["sell_signal"])
    log_return_df = log_return_df[["buy_log_return", "sell_log_return"]]

    # Merge the calculated log returns back to the original DataFrame based on index
    result_df = pd.merge(indicator_df, log_return_df, how="left", left_index=True, right_index=True)

    return result_df
```

Define the function to calculate Log Return for Indicator Trading Rules

Date	Adj Close	MACD	buy_signal	sell_signal	buy_log_return	sell_log_return
2018-04-02 00:00:00	33255.359375	-300.59507934331486	0.0	0.0	0.0	-0.0
2018-04-03 00:00:00	33370.62890625	-292.8641612141073	0.0	0.0	0.0	-0.0
2018-04-04 00:00:00	33019.0703125	-311.4267008047973	0.0	0.0	0.0	-0.0
2018-04-05 00:00:00	33596.80078125	-284.5031769775014	0.0	0.0	0.0	-0.0
2018-04-06 00:00:00	33626.96875	-257.85431385923584	0.0	0.0	0.0	-0.0
2018-04-09 00:00:00	33788.5390625	-221.7839513410654	0.0	0.0	0.0	-0.0
2018-04-10 00:00:00	33880.25	-182.61073636542278	0.0	0.0	0.0	-0.0
2018-04-11 00:00:00	33940.44140625	-143.10051952892536	0.0	0.0	0.0	-0.0
2018-04-12 00:00:00	34101.12890625	-95.93910306084581	0.0	0.0	0.0	-0.0
2018-04-13 00:00:00	34192.6484375	-47.21379203716788	0.0	0.0	0.0	-0.0
2018-04-16 00:00:00	34305.4296875	4.370180271493155	1.0	0.0	0.02461565232581236	-0.0
2018-04-17 00:00:00	34395.05859375	56.6686216021626	0.0	0.0	0.0	-0.0
2018-04-18 00:00:00	34331.6796875	98.1770717780164	0.0	0.0	0.0	-0.0
2018-04-19 00:00:00	34427.2890625	141.8057604153364	0.0	0.0	0.0	-0.0
2018-04-20 00:00:00	34415.578125	179.2859670117541	0.0	0.0	0.0	-0.0
2018-04-23 00:00:00	34450.76953125	214.7146410618443	0.0	0.0	0.0	-0.0
2018-04-24 00:00:00	34616.640625	257.9201341532753	0.0	0.0	0.0	-0.0
2018-04-25 00:00:00	34501.26953125	287.12344808428315	0.0	0.0	0.0	-0.0
2018-04-26 00:00:00	34713.6015625	328.10590782146755	0.0	0.0	0.0	-0.0
2018-04-27 00:00:00	34969.69921875	382.6728651676531	0.0	0.0	0.0	-0.0
2018-04-30 00:00:00	35160.359375	444.3186264607575	0.0	0.0	0.0	-0.0
2018-05-02 00:00:00	35176.421875	499.1943312400763	0.0	0.0	0.0	-0.0
2018-05-03 00:00:00	35103.140625	541.3558774048724	0.0	0.0	0.0	-0.0
2018-05-04 00:00:00	34915.37890625	563.7063071156663	0.0	0.0	-0.0	0.0
2018-05-07 00:00:00	35208.140625	604.008706732202	0.0	0.0	-0.0	0.0

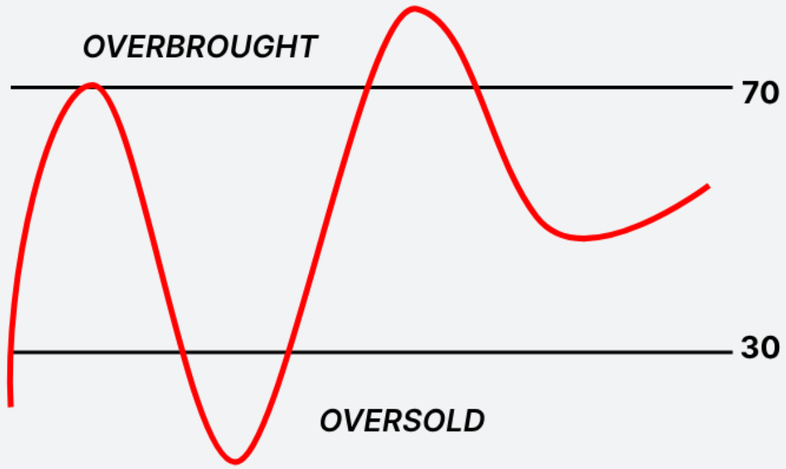

```
macd_dict_rule1["^BSESN"]
```

Relative Strength Index (RSI)

เป็นเครื่องมือที่พัฒนามาจาก Indicator ที่มีชื่อว่า Momentum ซึ่งใช้ในการวิเคราะห์ทางเทคนิคเพื่อวัดขนาดการเปลี่ยนแปลงของราคาล่าสุด และประเมินสภาวะซื้อมากเกินไป (Overbought) หรือขายมากเกินไป (Oversold) ในราคาหุ้นหรือสินทรัพย์อื่น ๆ โดยจะแสดงเป็นกราฟเส้นที่สามารถอ่านค่าได้ตั้งแต่ 0 – 100

$$RSI = 100 - \left(\frac{100}{1+\frac{\text{Average gain}}{\text{Average loss}}}\right)$$

What is Relative Strength Indicator (RSI) ?



RSI Rule 3

Buy Signal (Long Position) : เมื่อเส้น RSI ตัดเส้นกึ่งกลางขึ้นมา (RSI = 50) จากด้านล่าง

Sell Signal (Short Position) : เมื่อเส้น RSI ตัดเส้นกึ่งกลางลงมา (RSI = 50) จากข้างบน

ซึ่งกฎของ RSI นี้จะเป็น RSI (N,50)

RSI Rule 4

Buy Signal (Long Position) : เมื่อค่า RSI มีค่าต่ำกว่า 30 ลงมาในช่วง Oversold และค่า RSI กลับขึ้นไปสูงกว่า 30 (จะเป็นสัญญาของการซื้อ)

Sell Signal (Short Position) : เมื่อค่า RSI มีค่าสูงกว่า 70 ขึ้นไปอยู่ในช่วง Overbought และค่า RSI กลับลงมาต่ำกว่า 70 (จะเป็นสัญญาของการขาย)

ซึ่งกฎของ RSI นี้สามารถเปลี่ยนจำนวนที่คิดตามความเหมาะสม

```python
# Function to calculate the Relative Strength Index (RSI)
def calculate_rsi(index_price_df,
                  period = 14,
                  centerline = True,
                  os = None,
                  ob = None,
                  price = type_price):
```

```python
    if not isinstance(centerline, bool):
        raise ValueError("centerline must be a Boolean value.")

    if ob is not None:
        if not (ob >= 60 and ob <= 90):
            raise ValueError("ob must be a float between 60 and 90.")

    if os is not None:
        if not (os >= 10 and os <= 40):
            raise ValueError("os must be a float between 10 and 40.")

    if centerline is True and (ob is not None or os is not None):
        raise ValueError("You cannot specify centerline along with ob or os.")

    if ob is not None and os is None:
        raise ValueError("If you specify ob, you must also specify os.")

    if os is not None and ob is None:
        raise ValueError("If you specify os, you must also specify ob.")

    if os is not None and ob is not None and centerline is not False:
        raise ValueError("If you specify os and ob, centerline must be False.")
```

Define the function to calculate the RSI

```
### Calculate RSI

# Create a copy of the input DataFrame
RSI_df = index_price_df.copy()

# Calculate the price change between consecutive days
RSI_df["price_change"] = RSI_df[price].diff()
RSI_df.dropna(subset=["price_change"], inplace=True)

# Separate gains and losses based on price changes
RSI_df.loc[:,"gain"] = np.where(RSI_df["price_change"] > 0, RSI_df["price_change"], 0)
RSI_df.loc[:,"loss"] = np.where(RSI_df["price_change"] < 0, -RSI_df["price_change"], 0)

# Calculate average gain and average loss by moving average
RSI_df.loc[:,"avg_gain"] = RSI_df["gain"].rolling(window=period, min_periods=period).mean()
RSI_df.loc[:,"avg_loss"] = RSI_df["loss"].rolling(window=period, min_periods=period).mean()

# Calculate RSI from average gain and average loss for find buy and sell signal
RSI_df.loc[:,"RSI"] = np.where(RSI_df["avg_loss"] != 0,
                               100 - (100 / (1 + (RSI_df["avg_gain"] / RSI_df["avg_loss"]))),
                               100)

# Drop Gain, Loss, Average Gain, and Average Loss columns used for calculations
RSI_df.drop(["price_change", "gain", "loss", "avg_gain", "avg_loss"], inplace=True, axis=1)
```

Calculate RSI

```python
# RSI Rule 3: RSI crosses the centerline
if centerline == True:
    RSI_cross_center = RSI_df.copy().dropna()
    RSI_cross_center = RSI_cross_center[["RSI"]]

    # Create new column for check previous day
    RSI_cross_center.loc[:,"RSI_shift"] = RSI_cross_center["RSI"].shift(1)

    # Update Buy signal: RSI crosses the centerline from below
    from_below = (RSI_cross_center["RSI_shift"] < 50) & (RSI_cross_center["RSI"] >= 50)
    RSI_cross_center.loc[:,"buy_signal"] = np.where(from_below, 1, 0)

    # Update Sell signal: RSI crosses the centerline from above
    from_above = (RSI_cross_center["RSI_shift"] > 50) & (RSI_cross_center["RSI"] <= 50)
    RSI_cross_center.loc[:,"sell_signal"] = np.where(from_above, 1, 0)

    # Drop RSI and Join table
    RSI_cross_center.drop(["RSI", "RSI_shift"], axis=1, inplace=True)
    RSI_result_df = pd.merge(RSI_df, RSI_cross_center, how="left", left_index=True, right_index=True)
```

Rule 3 Get Signal of RSI Rule 3 and RSI Rule 4

```python
# RSI Rule 4: RSI crosses the oversold and overbought
elif centerline == False:
    RSI_cross_osob = RSI_df.copy().dropna()
    RSI_cross_osob = RSI_cross_osob[["RSI"]]

    # Create new column for check previous day
    RSI_cross_osob.loc[:,"RSI_shift"] = RSI_cross_osob["RSI"].shift(1)

    # Update Buy signal: RSI falls below oversold zone (RSI < os value) and rises above os value again
    from_below = (RSI_cross_osob["RSI_shift"] < os) & (RSI_cross_osob["RSI"] >= os)
    RSI_cross_osob.loc[:,"buy_signal"] = np.where(from_below, 1, 0)

    # Update Sell signal: RSI rises abover overbought zone (RSI > ob value) and falls below ob value again
    from_above = (RSI_cross_osob["RSI_shift"] > ob) & (RSI_cross_osob["RSI"] <= ob)
    RSI_cross_osob.loc[:,"sell_signal"] = np.where(from_above, 1, 0)

    # Drop RSI and Join table
    RSI_cross_osob.drop(["RSI", "RSI_shift"], axis=1, inplace=True)
    RSI_result_df = pd.merge(RSI_df, RSI_cross_osob, how="left", left_index=True, right_index=True)

else:
    raise ValueError("Please enter centerline or (os and ob) again")
```

Rule 4 Get Signal of RSI Rule 3 and RSI Rule 4

Date	Adj Close	RSI	buy_signal	sell_signal	buy_log_return	sell_log_return
2013-10-01 00:00:00	19517.150390625	NaN	NaN	NaN	NaN	NaN
2013-10-03 00:00:00	19902.0703125	NaN	NaN	NaN	NaN	NaN
2013-10-04 00:00:00	19915.94921875	NaN	NaN	NaN	NaN	NaN
2013-10-07 00:00:00	19895.099609375	NaN	NaN	NaN	NaN	NaN
2013-10-08 00:00:00	19983.609375	NaN	NaN	NaN	NaN	NaN
2013-10-09 00:00:00	20249.259765625	NaN	NaN	NaN	NaN	NaN
2013-10-10 00:00:00	20272.91015625	NaN	NaN	NaN	NaN	NaN
2013-10-11 00:00:00	20528.58984375	NaN	NaN	NaN	NaN	NaN
2013-10-14 00:00:00	20607.5390625	NaN	NaN	NaN	NaN	NaN
2013-10-15 00:00:00	20547.619140625	NaN	NaN	NaN	NaN	NaN
2013-10-17 00:00:00	20415.509765625	NaN	NaN	NaN	NaN	NaN
2013-10-18 00:00:00	20882.890625	NaN	NaN	NaN	NaN	NaN
2013-10-21 00:00:00	20893.890625	NaN	NaN	NaN	NaN	NaN
2013-10-22 00:00:00	20864.970703125	NaN	NaN	NaN	NaN	NaN
2013-10-23 00:00:00	20767.880859375	NaN	NaN	NaN	NaN	NaN
2013-10-24 00:00:00	20725.4296875	NaN	NaN	NaN	NaN	NaN
2013-10-25 00:00:00	20683.51953125	NaN	NaN	NaN	NaN	NaN
2013-10-28 00:00:00	20570.279296875	NaN	NaN	NaN	NaN	NaN
2013-10-29 00:00:00	20929.009765625	NaN	NaN	NaN	NaN	NaN
2013-10-30 00:00:00	21033.970703125	NaN	NaN	NaN	NaN	NaN
2013-10-31 00:00:00	21164.51953125	NaN	NaN	NaN	NaN	NaN
2013-11-01 00:00:00	21196.810546875	81.43646207214886	0.0	0.0	-0.0	0.0
2013-11-05 00:00:00	20974.7890625	74.5009116712967	0.0	0.0	-0.0	0.0
2013-11-06 00:00:00	20894.939453125	68.59589810298421	0.0	1.0	-0.0	0.032387269486044985
2013-11-07 00:00:00	20822.76953125	66.62132090764071	0.0	0.0	-0.0	0.0

```
rsi_dict_rule4['^BSESN']
```

04 Parameter Tuning for Trading Strategy

```python
# Check if the rule is 1
if rule == 1:
    # Loop through different values for period_short, period_long
    for period_short in range(10, 81):
        for period_long in range(period_short + 10, 151):

            # Calculate MACD and adjust signals
            macd_df = calculate_macd(df, period_short=period_short, period_long=period_long, signal_line=0)
            macd_df = adjust_signal(macd_df)
            macd_df = calculate_trading_rules_log_return(macd_df)

            # Calculate the mean buy and sell log returns
            buy_mean = macd_df.loc[macd_df["buy_log_return"] != 0, "buy_log_return"].mean()
            sell_mean = macd_df.loc[macd_df["sell_log_return"] != 0, "sell_log_return"].mean()
            mean = buy_mean + sell_mean

            buy_count = macd_df.loc[:, "buy_signal"].sum()
            sell_count = macd_df.loc[:, "sell_signal"].sum()

            # Create a parameter list and add it to the list of all parameters
            parameter = [period_short, period_long, 0, mean, buy_count, sell_count]
            all_parameter.append(parameter)

    # Convert the list of each parameters into a DataFrame
    all_parameter_df = pd.DataFrame(all_parameter, columns=["period_short", "period_long", "signal_line", "mean_return", "buy_count", "sell_count"])

    # Find the parameter combination with the highest mean_return
    max_ret = all_parameter_df[(all_parameter_df["buy_count"] > all_parameter_df["buy_count"].mean()) & (all_parameter_df["sell_count"] > all_parameter_df["sell_count"].mean())]
    max_ret = max_ret[max_ret["mean_return"] == max_ret["mean_return"].max()]
    period_short = max_ret["period_short"].values[0]
    period_long = max_ret["period_long"].values[0]
    signal_line = max_ret["signal_line"].values[0]
    mean_return = max_ret["mean_return"].values[0]

    # Return the best parameter combination
    return period_short, period_long, signal_line, mean_return
```

Rule 1

Define the function for Parameter Tuning of each rules

```python
# Check if the rule is 2
elif rule == 2:
    # Loop through different values for period_short, period_long, signal_line
    for period_short in range(10, 81):
        for period_long in range(period_short + 10, 151):
            for signal_line in range(5, 11):

                # Calculate MACD and adjust signals
                macd_df = calculate_macd(df, period_short=period_short, period_long=period_long, signal_line=signal_line)
                macd_df = adjust_signal(macd_df)
                macd_df = calculate_trading_rules_log_return(macd_df)

                # Calculate the mean buy and sell log returns
                buy_mean = macd_df.loc[macd_df["buy_log_return"] != 0, "buy_log_return"].mean()
                sell_mean = macd_df.loc[macd_df["sell_log_return"] != 0, "sell_log_return"].mean()
                mean = buy_mean + sell_mean

                buy_count = macd_df.loc[:, "buy_signal"].sum()
                sell_count = macd_df.loc[:, "sell_signal"].sum()

                # Create a parameter list and add it to the list of all parameters
                parameter = [period_short, period_long, signal_line, mean, buy_count, sell_count]
                all_parameter.append(parameter)

    # Convert the list of each parameters into a DataFrame
    all_parameter_df = pd.DataFrame(all_parameter, columns=["period_short", "period_long", "signal_line", "mean_return", "buy_count", "sell_count"])

    # Find the parameter combination with the highest mean_return
    max_ret = all_parameter_df[(all_parameter_df["buy_count"] > all_parameter_df["buy_count"].mean()) & (all_parameter_df["sell_count"] > all_parameter_df["sell_count"].mean())]
    max_ret = max_ret[max_ret["mean_return"] == max_ret["mean_return"].max()]
    period_short = max_ret["period_short"].values[0]
    period_long = max_ret["period_long"].values[0]
    signal_line = max_ret["signal_line"].values[0]
    mean_return = max_ret["mean_return"].values[0]

    # Return the best parameter combination
    return period_short, period_long, signal_line, mean_return
```

Rule 2

Define the function for Parameter Tuning of each rules

```
# Check if the rule is 3
elif rule == 3:
    # Loop through different values for period
    for period in range(10, 81):

        # Calculate RSI and adjust signals
        rsi_df = calculate_rsi(df, period=period, centerline=True)
        rsi_df = adjust_signal(rsi_df)
        rsi_df = calculate_trading_rules_log_return(rsi_df)

        # Calculate the mean buy and sell log returns
        buy_mean = rsi_df.loc[rsi_df["buy_log_return"] != 0, "buy_log_return"].mean()
        sell_mean = rsi_df.loc[rsi_df["sell_log_return"] != 0, "sell_log_return"].mean()
        mean = buy_mean + sell_mean

        buy_count = rsi_df.loc[:, "buy_signal"].sum()
        sell_count = rsi_df.loc[:, "sell_signal"].sum()

        # Create a parameter list and add it to the list of all parameters
        parameter = [period, mean, buy_count, sell_count]
        all_parameter.append(parameter)

    # Convert the list of each parameters into a DataFrame
    all_parameter_df = pd.DataFrame(all_parameter, columns=["period", "mean_return", "buy_count", "sell_count"])

    # Find the parameter combination with the highest mean_return
    max_ret = all_parameter_df[(all_parameter_df["buy_count"] > all_parameter_df["buy_count"].mean()) & (all_parameter_df["sell_count"] > all_parameter_df["sell_count"].mean())]
    max_ret = max_ret[max_ret["mean_return"] == max_ret["mean_return"].max()]
    period = max_ret["period"].values[0]
    mean_return = max_ret["mean_return"].values[0]

    # Return the best parameter combination
    return period, mean_return
```

Rule 3

Define the function for Parameter Tuning of each rules

```
# Check if the rule is 4
elif rule == 4:
    # Loop through different values for period, os, ob
    for period in range(10, 81):
        for os, ob in zip([20, 30], [80, 70]):

                # Calculate RSI and adjust signals
                rsi_df = calculate_rsi(df, period=period, centerline=False, os=os, ob=ob)
                rsi_df = adjust_signal(rsi_df)
                rsi_df = calculate_trading_rules_log_return(rsi_df)

                # Calculate the mean buy and sell log returns
                buy_mean = rsi_df.loc[rsi_df["buy_log_return"] != 0, "buy_log_return"].mean()
                sell_mean = rsi_df.loc[rsi_df["sell_log_return"] != 0, "sell_log_return"].mean()
                mean = buy_mean + sell_mean

                buy_count = rsi_df.loc[:, "buy_signal"].sum()
                sell_count = rsi_df.loc[:, "sell_signal"].sum()

                # Create a parameter list and add it to the list of all parameters
                parameter = [period, os, ob, mean, buy_count, sell_count]
                all_parameter.append(parameter)

    # Convert the list of each parameters into a DataFrame
    all_parameter_df = pd.DataFrame(all_parameter, columns=["period", "os", "ob", "mean_return", "buy_count", "sell_count"])

    # Find the parameter combination with the highest mean_return
    max_ret = all_parameter_df[(all_parameter_df["buy_count"] > all_parameter_df["buy_count"].mean()) & (all_parameter_df["sell_count"] > all_parameter_df["sell_count"].mean())]
    max_ret = max_ret[max_ret["mean_return"] == max_ret["mean_return"].max()]
    period = max_ret["period"].values[0]
    os = max_ret["os"].values[0]
    ob = max_ret["ob"].values[0]
    mean_return = max_ret["mean_return"].values[0]

    # Return the best parameter combination
    return period, os, ob, mean_return
##########################################################################################################################################################################
```

Rule 4

Define the function for Parameter Tuning of each rules

```python
# Parameter turning for finding the best parameters for MACD strategy
rule = 1

# Create a dictionary to store MACD parameters for different symbols
macd_dict_para_rule1 = {}

# Iterate through symbols and Parameter turning for MACD signal for each index
for symbol, index_price_df in index_price_dict.items():

    # Call a function to find the optimal MACD parameters for the current symbol and rule
    max_period_short, max_period_long, max_signal_line, max_mean_return = parameter_turning(index_price_df, rule = rule)

    # Store the found MACD parameters in the dictionary for the current symbol
    macd_dict_para_rule1[symbol] = [max_period_short, max_period_long, max_signal_line, max_mean_return]
```

Parameter Turning for MACD Rule 1: MACD crosses zero

```python
# NEW After Optimize (Turning for Finding the best) parameter MACD Rule 1
macd_dict_para_rule1 = {
    '^BSESN': [20, 93, 0, 0.052960009697397205],
    '^N225': [17, 148, 0, 0.024195181419448816],
    '^HSI': [28, 112, 0, 0.04224809107221513],
    '399001.SZ': [13, 49, 0, 0.026171408791009945],
    '000001.SS': [20, 31, 0, 0.016107061464924338],
    '^NSEI': [24, 55, 0, 0.05291505421186649],
    '^KS11': [32, 77, 0, 0.026981930673081734],
    '^TWII': [20, 121, 0, 0.02385377768401497],
    '^STI': [11, 91, 0, 0.0151193100655477],
    '^SET.BK': [13, 63, 0, 0.013610101831140904]
}
```

```python
# Set Rule and Strategy
rule = 1

# Create a dictionary to store MACD DataFrames for different symbols
macd_dict_rule1 = {}

for symbol, index_price_df, parameter_list in zip(index_price_dict.keys(), index_price_dict.values(), macd_dict_para_rule1.values()):

    # slice each macd parameter of each index
    max_period_short = parameter_list[0]
    max_period_long = parameter_list[1]
    max_signal_line = parameter_list[2]

    # Iterate through symbols and calculate MACD and Buy/Sell signal for each index
    macd_dict_rule1[symbol] = calculate_macd(index_price_df,
                                             period_short = max_period_short,
                                             period_long = max_period_long,
                                             signal_line = max_signal_line)


    # Iterate through symbols and adjust buy and sell signal for each index
    macd_dict_rule1[symbol] = adjust_signal(macd_dict_rule1[symbol])

    # Iterate through symbols and calculate buy and sell log return for each index
    macd_dict_rule1[symbol] = calculate_trading_rules_log_return(macd_dict_rule1[symbol], price = type_price)

    print(f"Complete to create MACD Rule {rule} as MACD({max_period_short},{max_period_long},{max_signal_line}) of {symbol}")
```

Apply the Best Parameter for MACD Rule 1: MACD crosses zero

```python
# Parameter turning for finding the best parameters for MACD strategy
rule = 2

# Create a dictionary to store MACD parameters for different symbols
macd_dict_para_rule2 = {}

# Iterate through symbols and Parameter turning for MACD signal for each index
for symbol, index_price_df in index_price_dict.items():

    # Call a function to find the optimal MACD parameters for the current symbol and rule
    max_period_short, max_period_long, max_signal_line, max_mean_return = parameter_turning(index_price_df, rule = rule)

    # Store the found MACD parameters in the dictionary for the current symbol
    macd_dict_para_rule2[symbol] = [max_period_short, max_period_long, max_signal_line, max_mean_return]
```

Parameter Turning for MACD Rule 2: MACD crosses the signal_line

```python
# NEW After Optimize (Turning for Finding the best) parameter MACD Rule 2
macd_dict_para_rule2 = {
    '^BSESN': [12, 107, 5, 0.007308],
    '^N225': [21, 42, 10, 0.011959],
    '^HSI': [20, 58, 6, 0.017018],
    '399001.SZ': [26, 57, 10, 0.02051],
    '000001.SS': [50, 61, 6, 0.014628],
    '^NSEI': [25, 135, 5, 0.006364],
    '^KS11': [34, 132, 5, 0.018170],
    '^TWII': [12, 83, 5, 0.013261],
    '^STI': [10, 146, 5, 0.019279],
    '^SET.BK': [35, 48, 8, 0.012857]
}
```

```
# Set Rule and Strategy
rule = 2

# Create a dictionary to store MACD DataFrames for different symbols
macd_dict_rule2 = {}

for symbol, index_price_df, parameter_list in zip(index_price_dict.keys(), index_price_dict.values(), macd_dict_para_rule2.values()):

    # slice each macd parameter of each index
    max_period_short = parameter_list[0]
    max_period_long = parameter_list[1]
    max_signal_line = parameter_list[2]

    # Iterate through symbols and calculate MACD and Buy/Sell signal for each index
    macd_dict_rule2[symbol] = calculate_macd(index_price_df,
                                             period_short = max_period_short,
                                             period_long = max_period_long,
                                             signal_line = max_signal_line)

    # Iterate through symbols and adjust buy and sell signal for each index
    macd_dict_rule2[symbol] = adjust_signal(macd_dict_rule2[symbol])

    # Iterate through symbols and calculate buy and sell log return for each index
    macd_dict_rule2[symbol] = calculate_trading_rules_log_return(macd_dict_rule2[symbol], price = type_price)
```

Apply the Best Parameter for MACD Rule 2: MACD crosses the signal_line

```python
# Parameter turning for finding the best parameters for RSI strategy
rule = 3

# Create a dictionary to store RSI parameters for different symbols
rsi_dict_para_rule3 = {}

# Iterate through symbols and Parameter turning for RSI signal for each index
for symbol, index_price_df in index_price_dict.items():

    # Call a function to find the optimal RSI parameters for the current symbol and rule
    max_period, max_mean_return = parameter_turning(index_price_df, rule = rule)

    # Store the found RSI parameters in the dictionary for the current symbol
    rsi_dict_para_rule3[symbol] = [max_period, max_mean_return]
```

Parameter Turning for RSI Rule 3: RSI crosses the centerline

```python
# NEW After Optimize (Turning for Finding the best) parameter RSI Rule 1
rsi_dict_para_rule3 = {
    '^BSESN': [10, 0.0049265260231308285],
    '^N225': [20, 0.005647426103924104],
    '^HSI': [16, 0.003317851437657518],
    '399001.SZ': [19, 0.007659214290682891],
    '000001.SS': [23, 0.006820912645341845],
    '^NSEI': [10, 0.005485283531483458],
    '^KS11': [36, 0.015939115807670722],
    '^TWII': [16, 0.0045690312045233075],
    '^STI': [10, 0.01053776104541056],
    '^SET.BK': [25, 0.012621796173549256]
}
```

```python
# Set Rule and Strategy
rule = 3
centerline = True

# Create a dictionary to store RSI DataFrames for different symbols
rsi_dict_rule3 = {}

for symbol, index_price_df, parameter_list in zip(index_price_dict.keys(), index_price_dict.values(), rsi_dict_para_rule3.values()):

    # slice each rsi parameter of each index
    max_period = parameter_list[0]

    # Iterate through symbols and calculate RSI and Buy/Sell signal for each index
    rsi_dict_rule3[symbol] = calculate_rsi(index_price_df,
                                           period = max_period,
                                           centerline = centerline)

    # Iterate through symbols and adjust buy and sell signal for each index
    rsi_dict_rule3[symbol] = adjust_signal(rsi_dict_rule3[symbol])

    # Iterate through symbols and calculate buy and sell log return for each index
    rsi_dict_rule3[symbol] = calculate_trading_rules_log_return(rsi_dict_rule3[symbol])

    print(f"Complete to create RSI Rule {rule} as RSI({max_period},50) of {symbol}")
```

Apply the Best Parameter for RSI Rule 3: RSI crosses the centerline

```python
# Parameter turning for finding the best parameters for RSI strategy
rule = 4

# Create a dictionary to store RSI parameters for different symbols
rsi_dict_para_rule4 = {}

# Iterate through symbols and Parameter turning for RSI signal for each index
for symbol, index_price_df in index_price_dict.items():

    # Call a function to find the optimal RSI parameters for the current symbol and rule
    max_period, max_os, max_ob, max_mean_return = parameter_turning(index_price_df, rule = rule)

    # Store the found RSI parameters in the dictionary for the current symbol
    rsi_dict_para_rule4[symbol] = [max_period, max_os, max_ob, max_mean_return]
```

Parameter Turning for RSI Rule 4: RSI crosses the oversold and overbought

```python
# NEW After Optimize (Turning for Finding the best) parameter RSI Rule 2
rsi_dict_para_rule4 = {
    '^BSESN': [22, 30, 70, 0.007419352383202654],
    '^N225': [14, 20, 80, 0.030022706297901203],
    '^HSI': [41, 30, 70, 0.05290384532059271],
    '399001.SZ': [29, 30, 70, 0.021019486941094297],
    '000001.SS': [27, 30, 70, 0.029444337696901933],
    '^NSEI': [33, 30, 70, 0.011402813480866112],
    '^KS11': [20, 20, 80, 0.038879906785413934],
    '^TWII': [34, 30, 70, 0.02716104146169302],
    '^STI': [13, 20, 80, 0.027755139515745205],
    '^SET.BK': [20, 20, 80, 0.010492253968142814]
}
```

```
# Set Rule and Strategy
rule = 4
centerline = False

# Create a dictionary to store RSI DataFrames for different symbols
rsi_dict_rule4 = {}

for symbol, index_price_df, parameter_list in zip(index_price_dict.keys(), index_price_dict.values(), rsi_dict_para_rule4.values()):

    # slice each macd parameter of each index
    max_period = parameter_list[0]
    max_os = parameter_list[1]
    max_ob = parameter_list[2]

    # Iterate through symbols and calculate RSI and Buy/Sell signal for each index
    rsi_dict_rule4[symbol] = calculate_rsi(index_price_df,
                                           period = max_period,
                                           centerline = centerline,
                                           os = max_os,
                                           ob = max_ob,
                                           price=type_price)

    # Iterate through symbols and adjust buy and sell signal for each index
    rsi_dict_rule4[symbol] = adjust_signal(rsi_dict_rule4[symbol])

    # Iterate through symbols and calculate buy and sell log return for each index
    rsi_dict_rule4[symbol] = calculate_trading_rules_log_return(rsi_dict_rule4[symbol])

    print(f"Complete to create RSI Rule {rule} as RSI({max_period},{max_os}/{max_ob}) of {symbol}")
```

Apply the Best Parameter for RSI Rule 4: RSI crosses the oversold and overbought

05 Empirical result

```python
def summary_table_func(buy_and_hold_dict,
                       indicator_dict,
                       parameter_dict,
                       rule=1,
                       symbols=index_symbols,
                       names=index_names):

    # Create an empty summary table as a dictionary.
    summary_table = {"Symbol": symbols,  # Column for index symbols
                     "Sample Period (2013-2023)": names}  # Column for index names
    summary_table = pd.DataFrame(summary_table) # Convert the dictionary to a Pandas DataFrame

    # Iterate through the dictionary of index data for each symbol.
    for symbol, indicator_df, buy_and_hold_df, parameter_list in zip(indicator_dict.keys(), indicator_dict.values(), buy_and_hold_dict.values(), parameter_dict.values()):

        indicator_df = indicator_df.dropna()

        ################################################################################

        # Find the row in the summary table where 'Symbol' matches the current symbol.
        condition = summary_table["Symbol"] == symbol

        ################################################################################

        # slice each technical parameter of each index for add parameter to summary table
        if (rule == 1) | (rule == 2):
            period_short = parameter_list[0]
            period_long = parameter_list[1]
            signal_line = parameter_list[2]
            summary_table.loc[condition, "Parameter"] = "MACD(" + str(period_short) + "," + str(period_long) + "," + str(signal_line) + ")"
        elif (rule == 3):
            period = parameter_list[0]
            summary_table.loc[condition, "Parameter"] = "RSI(" + str(period) + ",50)"
        elif (rule == 4):
            period = parameter_list[0]
            os = parameter_list[1]
            ob = parameter_list[2]
            summary_table.loc[condition, "Parameter"] = "RSI(" + str(period) + "," + str(os) + "/" + str(ob) + ")"
```

Define the function to Create Summary Table for Indicator Trading Rules

```
# Find number of buy and sell signal
summary_table.loc[condition, "N(Buy)"] = indicator_df["buy_signal"].sum()
summary_table.loc[condition, "N(Sell)"] = indicator_df["sell_signal"].sum()

#############################################################################

indicator_buy_mean = indicator_df.loc[indicator_df["buy_log_return"] != 0, "buy_log_return"].mean()

### t-Test two sample assuming unequal variances
### Between average ten-day return of trading rules for buy signal and average ten-day return of Buy and Hold Strategy

t_buy, t_pvalue_buy = stats.ttest_ind(indicator_df.loc[indicator_df["buy_log_return"] != 0, "buy_log_return"],
                                      buy_and_hold_df["log_return"],
                                      equal_var=False,
                                      alternative="greater")

if (t_pvalue_buy < 0.05): # reject Ho at alpha = 0.05
    summary_table.loc[condition, "Buy"] = str(round(indicator_buy_mean, 5)) + "**" + "(" + str(round(t_buy, 3)) + ")"
elif (t_pvalue_buy < 0.10): # reject Ho at alpha = 0.10
    summary_table.loc[condition, "Buy"] = str(round(indicator_buy_mean, 5)) + "*" + "(" + str(round(t_buy, 3)) + ")"
else: # accept Ho
    summary_table.loc[condition, "Buy"] = str(round(indicator_buy_mean, 5)) + "(" + str(round(t_buy, 3)) + ")"

#############################################################################

indicator_sell_mean = indicator_df.loc[indicator_df["sell_log_return"] != 0, "sell_log_return"].mean()

### t-Test two sample assuming unequal variances
### Between average ten-day return of trading rules for sell signal and average ten-day return of Buy and Hold Strategy

t_sell, t_pvalue_sell = stats.ttest_ind(indicator_df.loc[indicator_df["sell_log_return"] != 0, "sell_log_return"],
                                        -1*buy_and_hold_df["log_return"],
                                        equal_var=False,
                                        alternative="greater")

if (t_pvalue_sell < 0.05): # reject Ho at alpha = 0.05
    summary_table.loc[condition, "Sell"] = str(round(indicator_sell_mean, 5)) + "**" + "(" + str(round(t_sell, 3)) + ")"
elif (t_pvalue_sell < 0.10): # reject Ho at alpha = 0.10
    summary_table.loc[condition, "Sell"] = str(round(indicator_sell_mean, 5)) + "*" + "(" + str(round(t_sell, 3)) + ")"
else: # accept Ho
    summary_table.loc[condition, "Sell"] = str(round(indicator_sell_mean, 5)) + "(" + str(round(t_sell, 3)) + ")"
```

Define the function to Create Summary Table for Indicator Trading Rules

```
    ### Find Percentage when ten-days return is profit or better than 0

    summary_table.loc[condition, "Buy>0"] = round(indicator_df.loc[indicator_df["buy_log_return"] > 0, "buy_log_return"].count() / indicator_df.loc[indicator_df["buy_log_return"] != 0, "buy_log_return"].count(), 3)
    summary_table.loc[condition, "Sell>0"] = round(indicator_df.loc[indicator_df["sell_log_return"] > 0, "sell_log_return"].count() / indicator_df.loc[indicator_df["sell_log_return"] != 0, "sell_log_return"].count(), 3)

    ###############################################################################

    indicator_buy_minus_sell_mean = indicator_buy_mean + indicator_sell_mean

    ### t-Test two sample assuming unequal variances
    ### Between average ten-day return of trading rules for buy signal and average ten-day return of trading rules for sell signal

    t_buy_minus_sell, t_pvalue_buy_minus_sell = stats.ttest_ind(indicator_df.loc[indicator_df["buy_log_return"] != 0, "buy_log_return"],
                                                               -indicator_df.loc[indicator_df["sell_log_return"] != 0, "sell_log_return"],
                                                               equal_var=False,
                                                               alternative="greater")

    if (t_pvalue_buy_minus_sell < 0.05): # reject Ho at alpha = 0.05
        summary_table.loc[condition, "Buy+Sell"] = str(round(indicator_buy_minus_sell_mean, 5)) + "**" + "(" + str(round(t_buy_minus_sell, 3)) + ")"
    elif (t_pvalue_buy_minus_sell < 0.10): # reject Ho at alpha = 0.10
        summary_table.loc[condition, "Buy+Sell"] = str(round(indicator_buy_minus_sell_mean, 5)) + "*" + "(" + str(round(t_buy_minus_sell, 3)) + ")"
    else: # accept Ho
        summary_table.loc[condition, "Buy+Sell"] = str(round(indicator_buy_minus_sell_mean, 5)) + "(" + str(round(t_buy_minus_sell, 3)) + ")"

    ###############################################################################

# Set the "Symbol" column to index names from the summary table.
summary_table.set_index("Symbol", inplace=True)

return summary_table
```

Define the function to Create Summary Table for Indicator Trading Rules

Rule 1

```
# Set Rule and Strategy
rule = 1

macd_summary_table_rule1 = summary_table_func(buy_and_hold_dict = buy_and_hold_dict,
                                              indicator_dict = macd_dict_rule1,
                                              parameter_dict = macd_dict_para_rule1,
                                              rule = rule,
                                              symbols = index_symbols,
                                              names = index_names)
macd_summary_table_rule1
```

Symbol	Sample Period (2013-2023)	Parameter	N(Buy)	N(Sell)	Buy	Sell	Buy>0	Sell>0	Buy+Sell
^BSESN	Bombay Stock Exchange, India	MACD(20,93,0)	11.0	11.0	0.01283(0.841)	0.04013**(2.136)	0.727	0.818	0.05296**(2.296)
^N225	Tokyo Stock Exchange, Japan	MACD(17,148,0)	14.0	13.0	0.00546(0.245)	0.01874(1.014)	0.643	0.615	0.0242(1.037)
^HSI	Hong Kong Stock Exchange, Hong Kong	MACD(28,112,0)	12.0	12.0	0.01525**(2.139)	0.02699**(2.002)	0.667	0.667	0.04225**(2.819)
399001.SZ	Shenzhen Stock Exchange, China	MACD(13,49,0)	19.0	21.0	0.01513(1.237)	0.01104*(1.443)	0.526	0.714	0.02617**(1.849)
000001.SS	Shanghai Stock Exchange, China	MACD(20,31,0)	25.0	22.0	0.01006*(1.422)	0.00605(0.966)	0.680	0.500	0.01611*(1.65)
^NSEI	National Stock Exchange, India	MACD(24,55,0)	13.0	12.0	0.01146(0.692)	0.04145**(2.974)	0.692	0.833	0.05292**(2.907)
^KS11	Korea Exchange, South Korea	MACD(32,77,0)	11.0	12.0	0.00154(0.113)	0.02544*(1.716)	0.455	0.833	0.02698*(1.664)
^TWII	Taiwan Stock Exchange, Taiwan	MACD(20,121,0)	10.0	11.0	0.01122(1.168)	0.01264(0.867)	0.800	0.455	0.02385(1.242)
^STI	Singapore Exchange, Singapore	MACD(11,91,0)	19.0	20.0	0.01147**(2.44)	0.00365(0.936)	0.842	0.500	0.01512**(2.487)
^SET.BK	The Stock Exchange of Thailand, Thailand	MACD(13,63,0)	15.0	23.0	0.01712**(2.335)	-0.00351(-0.573)	0.667	0.391	0.01361*(1.481)

* พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ10% ,** พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ5%

Rule 2

```
# Set Rule and Strategy
rule = 2

macd_summary_table_rule2 = summary_table_func(buy_and_hold_dict = buy_and_hold_dict,
                                              indicator_dict = macd_dict_rule2,
                                              parameter_dict = macd_dict_para_rule2,
                                              rule = rule,
                                              symbols = index_symbols,
                                              names = index_names)

macd_summary_table_rule2
```

Symbol	Sample Period (2013-2023)	Parameter	N(Buy)	N(Sell)	Buy	Sell	Buy>0	Sell>0	Buy+Sell
^BSESN	Bombay Stock Exchange, India	MACD(12,107,5)	47.0	63.0	0.01359**(2.023)	-0.00628(-0.348)	0.723	0.365	0.00731(1.268)
^N225	Tokyo Stock Exchange, Japan	MACD(21,42,10)	50.0	45.0	0.00803(0.929)	0.00393(1.2)	0.620	0.422	0.01196*(1.532)
^HSI	Hong Kong Stock Exchange, Hong Kong	MACD(20,58,6)	49.0	57.0	0.01136**(2.371)	0.00566(1.018)	0.633	0.614	0.01702**(2.496)
399001.SZ	Shenzhen Stock Exchange, China	MACD(26,57,10)	38.0	48.0	0.01381*(1.368)	0.0067(0.719)	0.658	0.458	0.02051*(1.47)
000001.SS	Shanghai Stock Exchange, China	MACD(50,61,6)	37.0	40.0	0.00876*(1.576)	0.00587(0.856)	0.676	0.475	0.01463*(1.52)
^NSEI	National Stock Exchange, India	MACD(25,135,5)	38.0	44.0	0.01125(1.169)	-0.00489(0.007)	0.684	0.386	0.00636(0.918)
^KS11	Korea Exchange, South Korea	MACD(34,132,5)	38.0	40.0	0.01019**(2.109)	0.00798**(1.796)	0.605	0.600	0.01817**(2.77)
^TWII	Taiwan Stock Exchange, Taiwan	MACD(12,83,5)	46.0	62.0	0.01323**(2.349)	3e-05(0.735)	0.761	0.419	0.01326**(2.278)
^STI	Singapore Exchange, Singapore	MACD(10,146,5)	56.0	53.0	0.01096**(3.358)	0.00832**(2.172)	0.696	0.698	0.01928**(3.875)
^SET.BK	The Stock Exchange of Thailand, Thailand	MACD(35,48,8)	45.0	40.0	0.00875**(1.871)	0.00411(1.16)	0.622	0.625	0.01286**(2.208)

* พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ10% ,** พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ5%

Rule 3

```
# Set Rule and Strategy
rule = 3

rsi_summary_table_rule3 = summary_table_func(buy_and_hold_dict = buy_and_hold_dict,
                                             indicator_dict = rsi_dict_rule3,
                                             symbols = index_symbols,
                                             parameter_dict = rsi_dict_para_rule3,
                                             rule = rule,
                                             names = index_names)

rsi_summary_table_rule3
```

Symbol	Sample Period (2013-2023)	Parameter	N(Buy)	N(Sell)	Buy	Sell	Buy>0	Sell>0	Buy+Sell
^BSESN	Bombay Stock Exchange, India	RSI(10,50)	42.0	79.0	0.01259*(1.435)	-0.00767(-0.853)	0.667	0.354	0.00493(0.796)
^N225	Tokyo Stock Exchange, Japan	RSI(20,50)	45.0	62.0	0.01216**(2.018)	-0.00651(-0.677)	0.644	0.355	0.00565(0.911)
^HSI	Hong Kong Stock Exchange, Hong Kong	RSI(16,50)	54.0	55.0	0.00198(0.533)	0.00134(0.062)	0.556	0.509	0.00332(0.468)
399001.SZ	Shenzhen Stock Exchange, China	RSI(19,50)	40.0	52.0	0.00097(0.049)	0.00669(0.797)	0.525	0.404	0.00766(0.688)
000001.SS	Shanghai Stock Exchange, China	RSI(23,50)	44.0	42.0	-0.00021(-0.284)	0.00703(1.205)	0.591	0.548	0.00682(0.758)
^NSEI	National Stock Exchange, India	RSI(10,50)	41.0	79.0	0.01319*(1.529)	-0.00771(-0.873)	0.683	0.354	0.00549(0.884)
^KS11	Korea Exchange, South Korea	RSI(36,50)	39.0	35.0	0.00594*(1.324)	0.00999**(1.877)	0.590	0.600	0.01594**(2.317)
^TWII	Taiwan Stock Exchange, Taiwan	RSI(16,50)	41.0	56.0	0.00684(0.784)	-0.00227(0.147)	0.683	0.464	0.00457(0.726)
^STI	Singapore Exchange, Singapore	RSI(10,50)	69.0	62.0	0.00504*(1.605)	0.00549**(1.752)	0.594	0.645	0.01054**(2.418)
^SET.BK	The Stock Exchange of Thailand, Thailand	RSI(25,50)	41.0	39.0	0.00863**(2.163)	0.00399(1.146)	0.634	0.564	0.01262**(2.392)

* พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ10% ,** พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ5%

Rule 4

```
[83] # Set Rule and Strategy
     rule = 4

     rsi_summary_table_rule4 = summary_table_func(buy_and_hold_dict = buy_and_hold_dict,
                                                  indicator_dict = rsi_dict_rule4,
                                                  symbols = index_symbols,
                                                  parameter_dict = rsi_dict_para_rule4,
                                                  rule = rule,
                                                  names = index_names)
     rsi_summary_table_rule4
```

Symbol	Sample Period (2013-2023)	Parameter	N(Buy)	N(Sell)	Buy	Sell	Buy>0	Sell>0	Buy+Sell
^BSESN	Bombay Stock Exchange, India	RSI(22,30/70)	11.0	49.0	0.01759(1.226)	-0.01017(-1.237)	0.727	0.388	0.00742(0.666)
^N225	Tokyo Stock Exchange, Japan	RSI(14,20/80)	11.0	31.0	0.03301**(2.426)	-0.00299(0.064)	0.818	0.452	0.03002**(2.154)
^HSI	Hong Kong Stock Exchange, Hong Kong	RSI(41,30/70)	8.0	14.0	0.04761**(2.218)	0.0053(0.471)	0.750	0.500	0.0529**(2.234)
399001.SZ	Shenzhen Stock Exchange, China	RSI(29,30/70)	12.0	21.0	0.02953*(1.747)	-0.00851(-0.543)	0.667	0.381	0.02102(0.959)
000001.SS	Shanghai Stock Exchange, China	RSI(27,30/70)	10.0	21.0	0.01508**(2.544)	0.01437(1.321)	0.800	0.476	0.02944**(2.257)
^NSEI	National Stock Exchange, India	RSI(33,30/70)	6.0	35.0	0.01971(1.067)	-0.00831(-0.688)	0.833	0.286	0.0114(0.777)
^KS11	Korea Exchange, South Korea	RSI(20,20/80)	7.0	16.0	0.03217**(2.804)	0.00671(1.24)	1.000	0.688	0.03888**(3.061)
^TWII	Taiwan Stock Exchange, Taiwan	RSI(34,30/70)	7.0	32.0	0.03028*(1.793)	-0.00312(-0.052)	0.857	0.438	0.02716*(1.666)
^STI	Singapore Exchange, Singapore	RSI(13,20/80)	26.0	39.0	0.01865**(2.953)	0.00911**(2.193)	0.731	0.641	0.02776**(3.696)
^SET.BK	The Stock Exchange of Thailand, Thailand	RSI(20,20/80)	9.0	15.0	0.007(0.646)	0.00349(0.899)	0.667	0.533	0.01049(0.934)

* พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ10% ,** พารามิเตอร์มีนัยสำคัญอยู่ที่ระดับ5%

Strategy

Buy and Hold

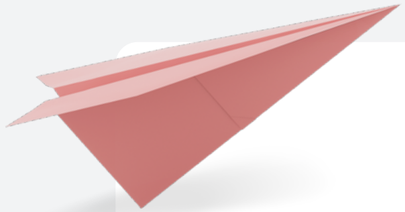
Symbol	Sample Period (2013-2023)	Mean	S.D.	Skewness	Kurtosis
^BSESN	Bombay Stock Exchange, India	0.00490	0.03417	-1.9983**	16.9082**
^N225	Tokyo Stock Exchange, Japan	0.00342	0.03836	-0.7892**	3.8005**
^HSI	Hong Kong Stock Exchange, Hong Kong	-0.00107	0.04012	-0.2896**	1.7275**
399001.SZ	Shenzhen Stock Exchange, China	0.00065	0.05323	-1.0859**	6.5168**
000001.SS	Shanghai Stock Exchange, China	0.00142	0.04305	-1.1281**	6.916**
^NSEI	National Stock Exchange, India	0.00492	0.03422	-2.0256**	16.9649**
^KS11	Korea Exchange, South Korea	0.00095	0.03294	-1.3682**	13.2264**
^TWII	Taiwan Stock Exchange, Taiwan	0.00283	0.03187	-1.1064**	6.3968**
^STI	Singapore Exchange, Singapore	0.00008	0.02954	-1.5724**	13.5074**
^SET.BK	The Stock Exchange of Thailand, Thailand	0.00023	0.03115	-1.5286**	14.5991**

```
buy_and_hold_summary_stat.to_excel("buyhold.xlsx")
```

Conculsion

บทสรุปการวิจัย

พบว่าในตลาด Singapore กลยุทธ์ส่วนใหญ่สามารถทำกำไรได้อย่างมีประสิทธิภาพมากกว่ากลยุทธ์การซื้อและถือ (Buy and Hold) นอกจากนี้ในทุกๆกลยุทธ์จะให้ผลตอบแทนที่ดีกว่าเมื่อเกิดสัญญาณซื้อ อีกทั้งกลยุทธ์ RSI crosses the oversold and overbought สามารถทำกำไรได้มีประสิทธิภาพมากที่สุดในตลาดของ Hong Kong โดยสามารถทำกำไรได้มากถึง 4.76% จึงสามารถสรุปได้ว่าไม่มีตลาดไหนที่กลยุทธ์การซื้อและถือ (Buy and Hold) จะสามารถเอาชนะทุกๆกลยุทธ์ได้



THANK YOU

ขอบคุณมาก! เราหวังว่าคุณได้เรียนรู้สิ่งใหม่แล้ว