Exploration and analysis of algorithmic trading strategies

Tang Kai Chi (Zepa)

December 28, 2024

0. Declaration and acknowledgements

0.1. Declaration

I hereby declare that this project is my own original work, and all sources of information have been properly acknowledged.

0.2. Acknowledgments

I would like to express my sincere gratitude to Professor Yiu Siu Ming and Dr. Chim Tat Wing for their invaluable guidance, insightful advice, and constant encouragement throughout the course of this research. Their expertise and mentorship have been instrumental in shaping the quality and direction of this work. Their dedication and support have not only been essential to the completion of this project but have also inspired me to strive for excellence and approach future challenges confidently. I am deeply grateful for their patience and encouragement, which have motivated me to push beyond my limits throughout this journey.

Table of Content

1. Abstract

Algorithmic trading has transformed financial markets by leveraging data-driven strategies to exploit inefficiencies and optimize trade executions. This research evaluates the performance of 3 classic strategies — Simple Moving Average (SMA), Momentum, and Mean Reversion — across various market conditions, timeframes, and asset classes, including the S&P 500 Index, gold, and GBP/USD. Assessment on profitability, adaptability and parameter sensitivity is performed with backtesting.

Momentum emerges as the most robust strategy, excelling in trending markets, particularly bear markets with persistent trends and heightened volatility. SMA demonstrates occasional profitability but is highly sensitive to parameter changes, having limited reliability. Mean Reversion underperforms in bull markets but outperforms benchmarks in bear markets by profiting from short-term price reversions and corrections.

This research highlights the need for strategy adaptation to market dynamics and time horizons. Future exploration of leverage trading, hybrid models, advanced risk controls, and a wider variety of asset classes could further improve trading frameworks. These findings offer insights for building resilient and systematic trading strategies in the fast evolving financial markets.

Keywords: Simple Moving Average, Momentum, Mean Reversion, Optimization of parameter, Heatmap, combinations, Timeframes, Bull market, Bear market

2. Introduction

Algorithmic trading, the use of rule-based systems for executing trades, is a cornerstone of modern financial markets. It systematically exploits inefficiencies, reduces behavioral biases, and enhances execution efficiency by data-driven strategies. Its growth has been driven by advancements in technology, high-frequency data, and complex market structures (Hendershott, Jones, & Menkveld, 2011).

This research evaluates the performance of 3 classic algorithmic trading strategies — Simple Moving Average (SMA), Momentum, and Mean Reversion — which represent distinct approaches to price behavior. The SMA strategy is a trend-following strategy that generates signals based on the crossover of shorter and longer term moving averages (Brock, Lakonishok, & LeBaron, 1992). The Momentum strategy capitalizes on trend persistence, making the assumption that assets with strong past performance have high probability to continue performing well in a certain period of time (Jegadeesh & Titman, 1993). In contrast, the Mean Reversion strategy makes the assumption that prices will revert to historical averages in overbought or oversold scenarios (Levy, 1967).

The analysis incorporates diverse types of asset classes, such as equities, commodities and forex, including the S&P 500 Index (SPX), gold (XAU/USD) and GBP/USD currency pair, to capture varying market dynamics and enhance generalizability.

This research aims to evaluate the profitability and returns of SMA, Momentum, and Mean Reversion strategies across various timeframes and market conditions, while identifying optimal parameter combinations and assessing performance sensitivity to parameter changes. It further seeks to compare strategy adaptability during bull and bear markets and test its robustness under different market conditions.

Building on foundational studies in technical trading (Brock et al., 1992), Momentum (Jegadeesh & Titman, 1993), and Mean Reversion (Levy, 1967), this research integrates modern data and computational methods to refine these strategies in contemporary markets. The findings aim to offer insights to support the development of systematic trading frameworks for practitioners and researchers.

3. Literature Review

Algorithmic trading applies quantitative techniques to exploit market inefficiencies. This literature review summarizes the Simple Moving Average (SMA), Momentum, and Mean Reversion strategies' core ideas and performance based on empirical and theoretical evidence.

3.1 Simple Moving Average (SMA)

SMA strategies identify trends by analyzing the crossover of shorter and longer term moving averages. Brock, Lakonishok, and LeBaron (1992) revealed that SMA rules are able to generate excess returns in trending markets by capturing momentum. However, SMA strategy has high sensitivity in parameter settings, limiting its reliability during range-bound or volatile conditions (Lo, Mamaysky, & Wang, 2000). SMA's profitability is also reduced after the emergence of high-frequency trading (HFT) as arbitrage opportunities are eliminated substantially (Hendershott, Jones, & Menkveld, 2011). To remain strategy's effectiveness, adaptive mechanisms and optimization techniques are required for the fluctuating market dynamics.

3.2 Momentum

Momentum strategies exploit trends persistence, making the assumption that assets with strong past performance will continue to outperform in a certain period. Jegadeesh and Titman (1993) illustrated that momentum portfolios are able to generate consistent excess returns over 3 to 12 months, driven by behavioral biases including herding and underreaction (Daniel, Hirshleifer, & Subrahmanyam, 1998). The strategy has been proven effective across various asset classes and market conditions, especially during volatile and trending markets (Moskowitz, Ooi, & Pedersen, 2012). However, the strategy is vulnerable to sharp reversals during market transitions, known as "momentum crashes" (Barroso & Santa-Clara, 2015). To mitigate risks and enhance robustness, risk controls such as volatility scaling are suggested to be incorporated (Baltas & Kosowski, 2020).

3.3 Mean Reversion

Mean reversion makes the assumption that prices will revert to historical averages after significant deviations to profit from overreactions in asset prices. Levy (1967) formalized the concept, and De Bondt and Thaler (1985) provided empirical evidence supporting its profitability in markets prone to overreaction. However, mean reversion strategy may suffer in trendy markets, during which prices deviate further without reversion (Lo & MacKinlay, 1990). Its effectiveness depends substantially on market conditions and precise calibration to avoid false signals.

3.4 Timeframe and Market Conditions

Strategy performance can vary in different market conditions and time frames. Momentum thrives in trendy and highly volatile markets, while mean reversion can perform better in range-bound, low-volatility environments (Asness, Moskowitz, & Pedersen, 2013). Due to high parameter sensitivity, SMA strategy is effective in trends but may lack robustness (Neely et al., 2014). Timeframes can also have direct effects on strategy optimal parameters. Momentum may benefit from smoothed volatility with larger rolling windows in short timeframes, whereas mean reversion may perform better in larger time frames where noise is reduced to a larger extent (Lo et al., 2000).

4. Data and Methodology

4.1 Data

4.1.1. Data Description

This research utilizes high-quality financial market data to rigorously evaluate the performance of algorithmic trading strategies across diverse types of instruments, timeframes, and market conditions. The data are extracted from TradingView, a globally recognized platform for reliable and comprehensive historical market data. The format of the datasets follows a structured convention:

- Dataset Format: "Asset (Broker) - Timeframe - (Starting Time - Ending Time)"
    - Asset: The financial instrument analyzed (e.g. GBPUSD, XAUUSD, SPX).
    - Broker: The brokerage providing the data (e.g. PEPPERSTONE).
    - Timeframe: The frequency of data aggregation (e.g. 5 min, 1 hour, daily).
    - Date Range: Varies per dataset, covering significant and random market periods (e.g. 1928-01-03 to 2024-11-29).

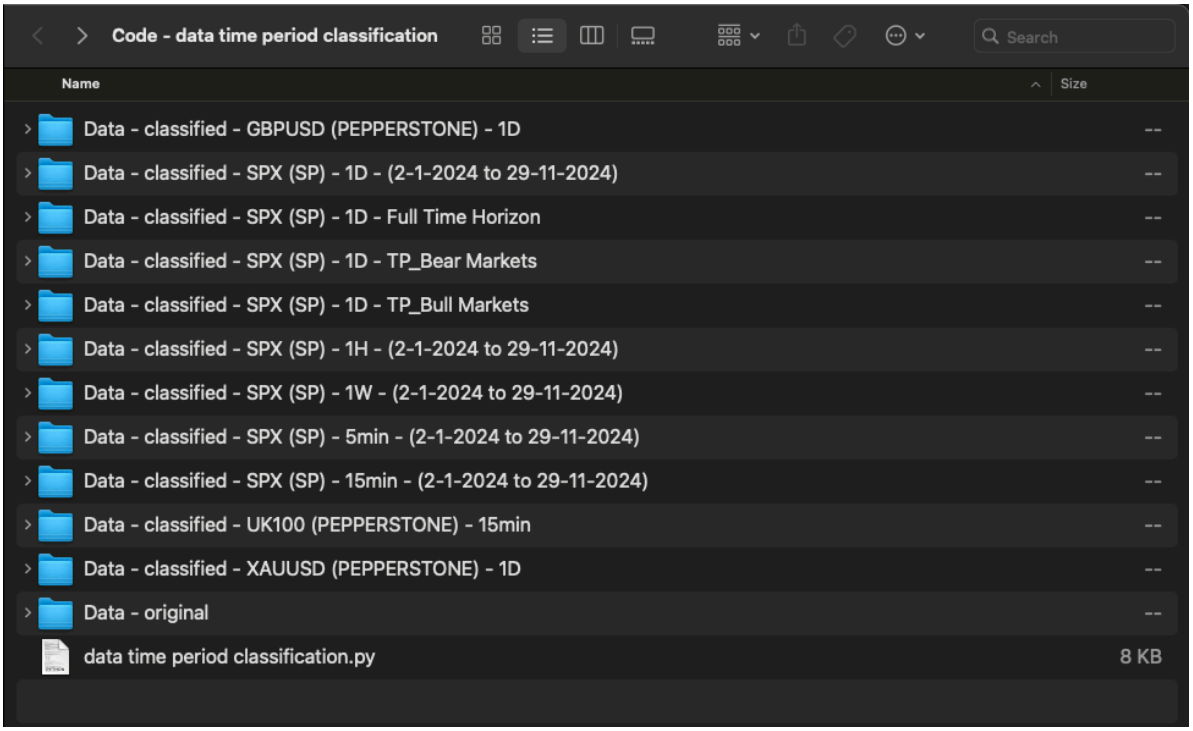
The data is chosen to encompass a wide range of market conditions, including bull and bear markets, ensuring robust evaluation of strategy performance under different market conditions.

4.1.2. Data Preprocessing

To ensure datasets suitability for backtesting, the preprocessing pipeline below is applied:

1. Data ingestion:
    - Data is imported as CSV files via the pandas library in Python.
    - Missing or invalid observations are removed using .dropna() to maintain data integrity.
    - The time column is converted into datetime objects for precise handling of time-series data.
    - The datasets are indexed by time to facilitate efficient slicing and filtering for specific periods.
2. Feature engineering:
    - The closing price is extracted for computing returns and generating signals.
    - Log returns are calculated as $\log(P_t / P_{t-1})$ to normalize price changes and enable comparability across assets.
    - Dynamic computation of rolling window averages and deviations are done in backtesting.
3. Dataset partitioning:

- Each dataset is segmented into specific time ranges for targeted scenario analysis. Examples include:
    - SPX (S&P 500): Weekly data from 1928-01-03 to 2024-11-29 to analyze performance over nearly a century of market cycles.
    - UK100 Index: 15-minute data from 2024-05-01 to 2024-11-29 for intraday strategy assessment.
- Using python code “data time period classification.py” in 4.1.2.1., each partitioned dataset is transformed and saved with a descriptive filename format: "Asset (Broker) - Timeframe - (Starting Time - Ending Time)" from the original less meaningful filename for systematic usage and clarity.
    - eg. SP_SPX, 1D_b2eac.csv → SPX (SP) - 1D - (2007-10-09 - 2009-03-09).csv



4. Incorporating realistic assumptions:
    - Transaction costs:
        - Transaction cost of 0.1% is applied to simulate real life trading scenarios.
    - Initial capital:
        - All strategies are benchmarked with an initial capital of 10,000 USD, ensuring consistent comparisons of performance across different instruments and strategies.

4.1.2.1. data time period classification.py

```
import pandas as pd

import os
```

```
# Get the directory of the current script

script_dir = os.path.dirname(os.path.abspath(__file__))



# Input folder and file name

import_folder = os.path.join(script_dir, "Data - original")



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

# Self-change manually:

# Replace with input file name

import_file_name = "SP_SPX, 1D_b2eac.csv"

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



input_file = os.path.join(import_folder, import_file_name)



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

# Self-change manually:

# Define the time periods

time_periods = [

    ("2007-10-09", "2009-03-09")

    # another example for timeframe smaller than Daily level (eg. 5 min, 1 hour):

    # ("2024-01-02T14:30:00Z", "2024-11-29T18:00:00Z"),

]

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



# Ensure the input file exists

if not os.path.exists(input_file):

    print(f"Error: Input file not found at {input_file}")

    exit(1)
```

```
# Extract the broker, instrument, and timeframe from the input file name
filename_parts = import_file_name.split(", ")
broker_instrument = filename_parts[0]  # Example: "BYBIT_BTCUSDT.P"
timeframe_id = filename_parts[1].split("_")[0]  # Example: "240" from "240_64f8c.csv"

# Map timeframe identifiers to output format
timeframe_mapping = {
   "1S": "1s",    # Second
   "15S": "15s",  # 15 seconds
   "1": "1min",   # 1 minute
   "5": "5min",   # 5 minutes
   "15": "15min", # 15 minutes
   "60": "1H",    # 1 hour
   "240": "4H",   # 4 hours
   "1D": "1D",    # Daily
   "1W": "1W",    # Weekly
   "1M": "1M",    # Monthly
   "12M": "1Y"    # Yearly
}

# Convert timeframe_id to a human-readable format
timeframe = timeframe_mapping.get(timeframe_id, timeframe_id)

# Extract the broker and instrument from the broker_instrument string
broker, instrument = broker_instrument.split("_", 1)  # Example: "BYBIT" and "BTCUSDT.P"

# Create the specific output folder dynamically
```

```
classified_folder = os.path.join(script_dir, f"Data - classified - {instrument} ({broker}) - {timeframe}")

os.makedirs(classified_folder, exist_ok=True)



# Load the CSV file

data = pd.read_csv(input_file)



# Detect time format and normalize to a standard format

if "T" in str(data["time"].iloc[0]):  # Check if the time column contains "T" (hourly or smaller)

   data["time"] = pd.to_datetime(data["time"], format="%Y-%m-%dT%H:%M:%S%z", errors="coerce")

else:  # Assume daily or higher timeframe (YYYY-MM-DD format)

   data["time"] = pd.to_datetime(data["time"], format="%Y-%m-%d", errors="coerce")



# Check for invalid 'time' values and drop them

if data["time"].isnull().any():

   print("Warning: Some rows have invalid 'time' values and will be dropped.")

   data = data.dropna(subset=["time"])  # Drop rows with invalid dates



# Iterate over time periods and split the data

for start_date, end_date in time_periods:

   # Filter rows within the time range

   filtered_data = data[(data["time"] >= start_date) & (data["time"] <= end_date)]



   # Remove problematic characters (e.g., "/") from the start and end dates

   start_date_cleaned = start_date.replace(":", "").replace("/", "").replace("-", "").replace("T", "")

   end_date_cleaned = end_date.replace(":", "").replace("/", "").replace("-", "").replace("T", "")
```

```
    # Format the output file name dynamically based on the extracted details

    output_file_name = f"{instrument} ({broker}) - {timeframe} - ({start_date_cleaned}Z -
{end_date_cleaned}Z).csv"

    output_file_path = os.path.join(classified_folder, output_file_name)



    # Save the filtered data to a new CSV file in the specific output folder

    filtered_data.to_csv(output_file_path, index=False)

    print(f"File saved: {output_file_path}")
```

4.3 Methodology

4.3.1. Backtesting framework - 2-stage streamlined process

The evaluation of various algorithmic trading strategies follows a 2-stage streamlined process:

Stage 1: Illustration phase

- Strategy_Illustration.ipynb: Develop and test strategy logic in Jupyter Notebook detailedly to ensure error-free before transitioning to a scalable framework.

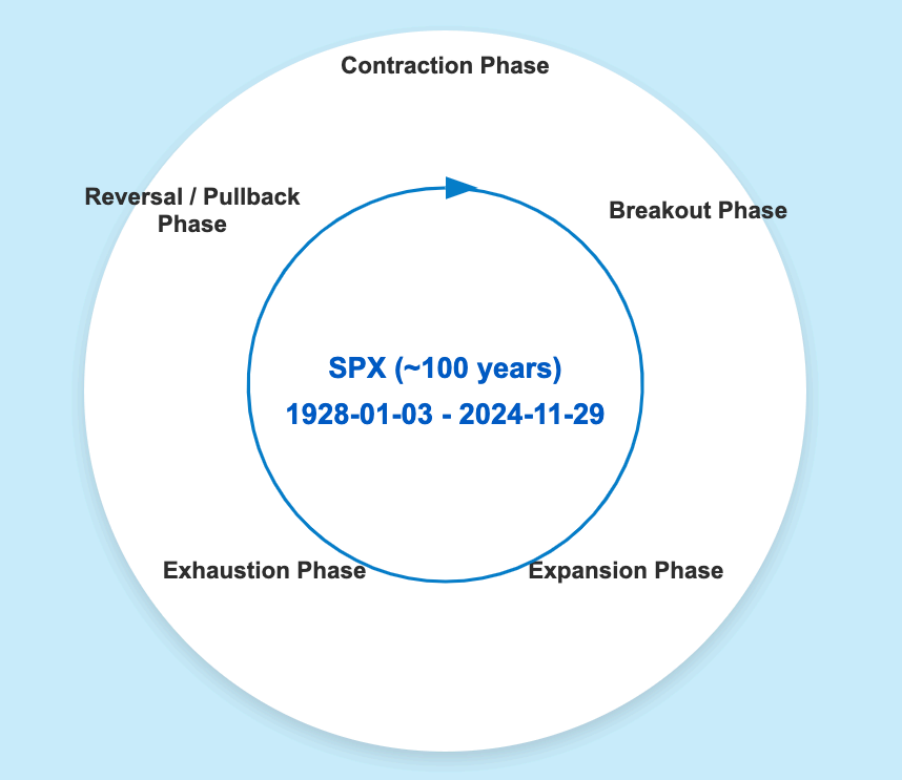
Stage 2: Backtesting phrase

1. Strategy_Backtester.py:
    - Modularize strategies for scalability and reproducibility, mainly incorporating strategies' algorithms.
2. Strategy_CodeRunner.ipynb:
    - Execute strategies with Strategy_Backtester.py, showing analysis of optimal parameter combinations, total performance & outperformance, heatmap for visualising optimization process and return graph compared to benchmark asset return.

4.3.2. Scenario Analysis

The scenario analysis part focuses on evaluating asset performance (S&P 500 index) and practicability of strategies (mainly momentum and mean reversion strategy) across various timeframes and market conditions to derive potential insights. The analysis includes 3 key components:

1. Full historical timespan of S&P 500 Index - (1928-01-03 - 2024-11-29)
    - Strategies are tested using a comprehensive dataset encompassing nearly a century of financial market dynamics, from 1928-01-03 to 2024-11-29.
    - This also helps evaluate strategies overall ultra long-term performance and adaptability since it has gone through the circular economic phrase below:



2. Time frame analysis
    - Different time frames ranging from shorter-term (5 min) to longer-term (weekly) are utilized to find the optimal combination of strategies and time frame based on difference in data granularity in time series and various levels of market noise and trend clarity, deriving insights for strategies' suitability for specific trading horizons.
3. Market conditions comparison - Bull and Bear markets
    - Strategies optimal performance and parameter combinations are compared using selected bull and bear market phases, including major events such as the Dot-Com Bubble (1995-2000), the Global Financial Crisis (2007-2009), and the Post-COVID Bull Market (2020-2022). This helps identify which strategies potentially excel in bull and bear markets.

5. Strategy Design, Implementation, and Analysis

5.1. Simple Moving Average (SMA) Strategy

5.1.1. Strategy overview

The Simple Moving Average (SMA) Strategy is a classic trend-following indicator widely used in technical analysis. It leverages the behavior of moving averages—specifically the relationship between short-term and long-term SMAs—to generate actionable buy and sell signals.

The foundation of the SMA crossover strategy lies in identifying the relationship between two moving averages:

- Short-Term SMA (faster-moving average):
    - Reacts quicker to recent price movements, acting as a more sensitive signal for short-term trends.
- Long-Term SMA (slower-moving average):
    - Reacts slower, smoothing out noise and reflecting the broader trend over a longer time period.

5.1.1.1. Logic of Simple Moving Average (SMA)

- Bullish signal - Golden Cross:
    - When short-term SMA crosses above long-term SMA, it suggests a change in momentum upward, indicating the start of a new bullish trend, signaling a potential long opportunity.
- Bearish signal - Death Cross :
    - When short-term SMA crosses below long-term SMA, it suggests a change in momentum downward, indicating the start of a new bearish trend, signaling a potential short opportunity.
- Trading position(s) is/are opened based on the logic:
    - Define:
        - SMA_1 = $SMA_{\text{number of day\_1}}$
        - SMA_2 = $SMA_{\text{number of day\_2}}$
        - number of day_2 > number of day_1
    - Define market positioning T as:
        - T = 1, if SMA_1 > SMA_2 (long position), 0 otherwise (short position)

5.1.2. Implementation details: SMA_Illustration.ipynb

- For implementation details, SMA_Illustration - GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31).ipynb is used for illustration.
- Here, the data source is GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31).

5.1.2.1. Data import & processing - GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)

```
# Strategy - Simple Moving Averages
```

```
import pandas as pd
data_imported = "GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)"
fn = f'/Users/kctang/Desktop/Algo Trading Project  (S1)/Strategies/Data/{data_imported}.csv'
raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
print(raw_data)
```

```
                open     high      low    close  Volume
time
2010-01-04  1.61223  1.62411  1.60570  1.60893    8994
2010-01-05  1.60890  1.61533  1.59648  1.59866    9744
2010-01-06  1.59863  1.60634  1.59367  1.60082    9544
2010-01-07  1.60068  1.60582  1.58959  1.59256    8362
2010-01-08  1.59254  1.61109  1.59146  1.60211    8592
...             ...      ...      ...      ...     ...
2020-12-24  1.34902  1.36193  1.34886  1.35560  117499
2020-12-28  1.35448  1.35758  1.34290  1.34576  116785
2020-12-29  1.34484  1.35225  1.34422  1.35005  145295
2020-12-30  1.34949  1.36256  1.34893  1.36245  158513
2020-12-31  1.36116  1.36859  1.36040  1.36635  145481

[2854 rows x 5 columns]
```

- First, imports the pandas library, a powerful data manipulation and analysis library in Python.
- Then, import csv file data "GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)", the currency pair of the British Pound and the US Dollar.
- Then, read the data into a pandas DataFrame.
    - index_col=0 sets the 1st column as the index of the DataFrame.
    - parse_dates=True indicates that any date columns will be parsed as datetime objects, which increases easiness of handling dates.
    - .dropna() removes any rows in the DataFrame that contain missing values (NaN).
- Then, print out raw data to check data correctness, output shows there are 2854 rows of data.

```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2854 entries, 2010-01-04 to 2020-12-31
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   open    2854 non-null   float64
 1   high    2854 non-null   float64
 2   low     2854 non-null   float64
 3   close   2854 non-null   float64
 4   Volume  2854 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 133.8 KB
```

- Output a summary of a Pandas DataFrame of raw_data, “2854” among all rows shows the existence of all data because those “2854” matches [2854 rows x 5 columns].

```
data = pd.DataFrame(raw_data['close'])
data
```

time	close
2010-01-04	1.60893
2010-01-05	1.59866
2010-01-06	1.60082
2010-01-07	1.59256
2010-01-08	1.60211
...	...
2020-12-24	1.35560
2020-12-28	1.34576
2020-12-29	1.35005
2020-12-30	1.36245
2020-12-31	1.36635

2854 rows × 1 columns

- Extract the closing price only for further usage.

5.1.2.2. SMA parameters - SMA_1 & SMA_2

```
data['SMA_1'] = data['close'].rolling(42).mean()
data['SMA_2'] = data['close'].rolling(252).mean()
data.tail()
```

time	close	SMA_1	SMA_2
2020-12-24	1.35560	1.328651	1.281870
2020-12-28	1.34576	1.329781	1.282014
2020-12-29	1.35005	1.331144	1.282186
2020-12-30	1.36245	1.332750	1.282408
2020-12-31	1.36635	1.334523	1.282676

- SMA_1 is the 42-period simple moving average, which smooths the closing prices over the last 42 days, which helps identify shorter-term trends.
- SMA_2 is the 252-period simple moving average, which smooths the closing prices over the last 252 days, showing longer-term trends.
- Here, adopted 42 and 252 because of its numerical significance in trading:
    - 42 days is roughly equivalent to 2 months of trading (assuming about 21 trading days per month), which is typically adopted to capture short to medium-term trends.
    - 252 days is typically the total number of trading days in 1 year (excluding weekends and holidays), indicating long-term trends.

```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
from pylab import mpl, plt
# outdated line due to version update
# plt.style.use('seaborn')
plt.style.use('seaborn-v0_8')  # Use the base Seaborn style
mpl.rcParams['savefig.dpi'] = 120
mpl.rcParams['font.family'] = 'serif'
```

```
data.plot(title='GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31) | 42 & 252 days SMAs',
                  figsize=(20, 12));
```

```
# Show plot
# here, need to write plt.show() explicitly to show
plt.show()
```

- Here, plot the graph of SMA_1 and SMA_2, showing how longer-term trend and shorter-term trend reacts and crosses each other throughout the time horizon 2010-01-04 - 2020-12-31.

5.1.2.3. Market Positioning - SMA strategy

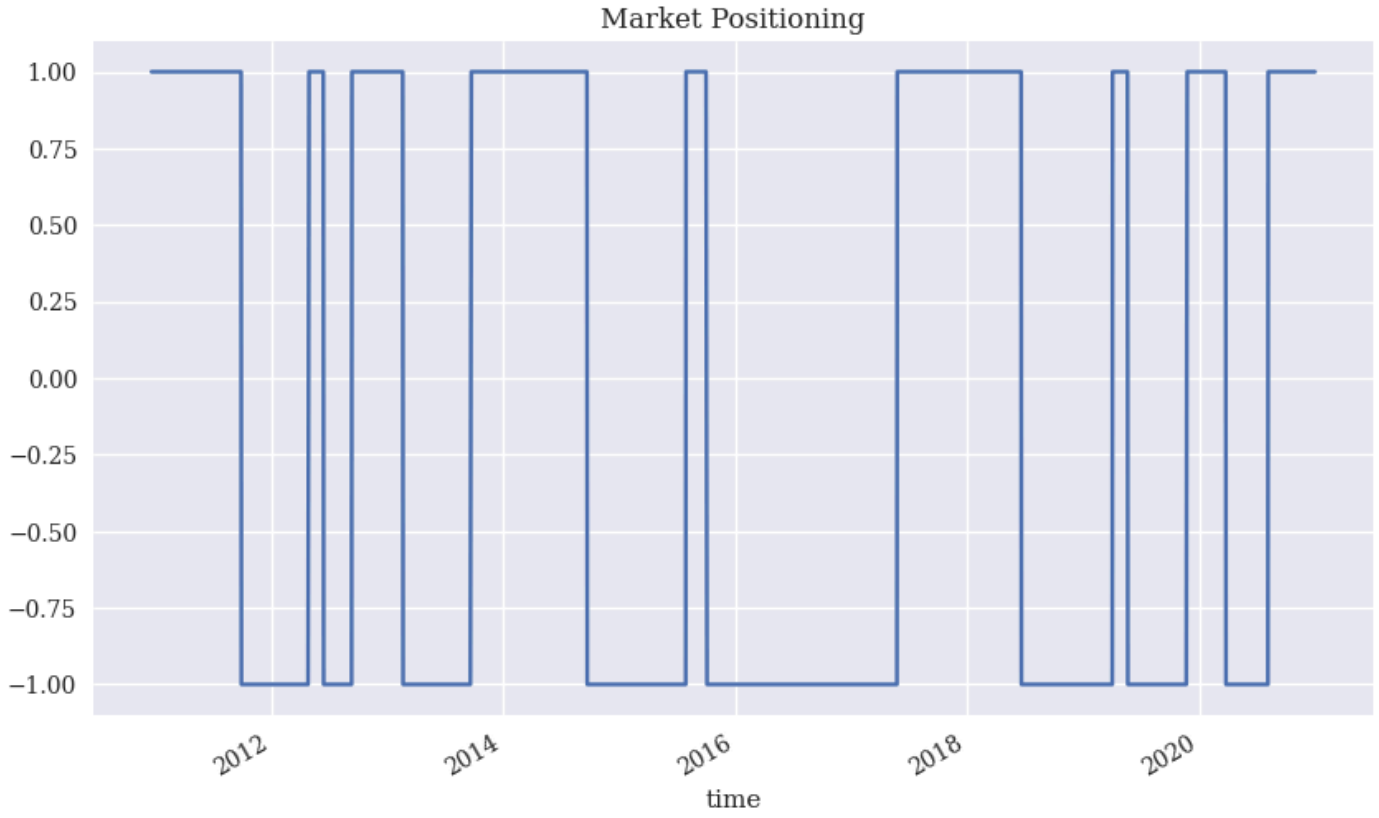
```
import numpy as np
```

```
data['position'] = np.where(data['SMA_1'] > data['SMA_2'],
                                    1, -1)
```

```
data.dropna(inplace=True)
```

```
data['position'].plot(ylim=[-1.1, 1.1],
                      title='Market Positioning',
                      figsize=(10, 6));
```

```
# Show plot
# here, need to write plt.show() explicitly to show
plt.show()
```



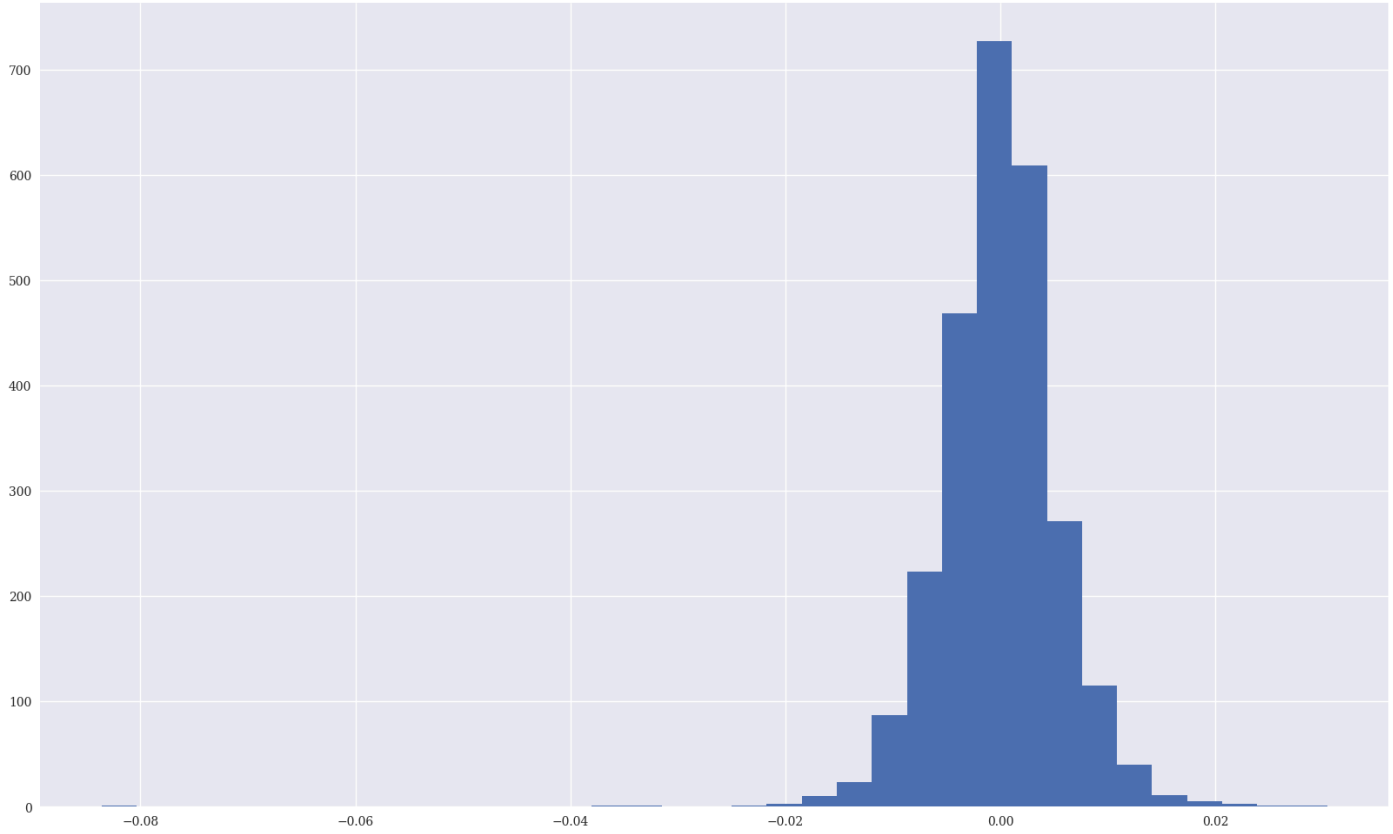
- Here, it opens trading position based on the logic:
  - Let:
    - SMA_1 = SMA_{42}
    - SMA_2 = SMA_{252}
  - Define market positioning T as:
    - T = 1, if SMA_1 > SMA_2 (long position), 0 otherwise (short position)
- A Market Positioning graph is shown, 1.00 & -1.00 means long position and short position respectively in different time. Here the long & shorts positions are quite visually balanced.

5.1.2.4. Histogram - Frequency distribution of log returns

```
data['returns'] = np.log(data['close'] / data['close'].shift(1))
```

```
# Frequency distribution of GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31) log returns
data['returns'].hist(bins=35, figsize=(20, 12));
```

```
plt.show()
```



- Calculates daily log returns by comparing daily closing price and previous day closing price
    - .shift(1) shifts the values in ‘close’ column by one position to previous data and the 1st row will have NaN value since there is no previous day available.
    - The reason for using “log” is because “log” has an addictive property which increases ease to further process data, compared to “non-log” calculation.
- Here, a histogram is plotted to show the frequency distribution of GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31) log returns.

5.1.2.5. Benchmark comparison

```
# Comparing the returns to show that SMA strategy > passive benchmark investment
```

```
data['strategy'] = data['position'].shift(1) * data['returns']
```

```
# Selected data
data
```

time	close	SMA_1	SMA_2	position	returns	strategy
2010-12-21	1.54688	1.584307	1.545193	1	NaN	NaN
2010-12-22	1.53902	1.583506	1.544916	1	-0.005094	-0.005094
2010-12-23	1.54262	1.582501	1.544693	1	0.002336	0.002336
2010-12-24	1.54346	1.581682	1.544466	1	0.000544	0.000544
2010-12-27	1.54167	1.580440	1.544264	1	-0.001160	-0.001160
...	...	...	...	...	...	...
2020-12-24	1.35560	1.328651	1.281870	1	0.004325	0.004325
2020-12-28	1.34576	1.329781	1.282014	1	-0.007285	-0.007285
2020-12-29	1.35005	1.331144	1.282186	1	0.003183	0.003183
2020-12-30	1.36245	1.332750	1.282408	1	0.009143	0.009143
2020-12-31	1.36635	1.334523	1.282676	1	0.002858	0.002858

2603 rows × 6 columns

- Here, “returns’ means ‘asset daily return’ and ‘strategy’ means ‘strategy’s daily return’, which refers to ‘SMA strategy’s daily return’ here. And columns “returns’ and ‘strategy’ are used to compare whether strategy outperforms.

```
# Full data
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(data)
```

```
                close     SMA_1     SMA_2  position   returns  strategy
time
2010-12-21   1.54688  1.584307  1.545193         1       NaN       NaN
2010-12-22   1.53902  1.583506  1.544916         1 -0.005094 -0.005094
2010-12-23   1.54262  1.582501  1.544693         1  0.002336  0.002336
2010-12-24   1.54346  1.581682  1.544466         1  0.000544  0.000544
2010-12-27   1.54167  1.580440  1.544264         1 -0.001160 -0.001160
2010-12-28   1.53642  1.578833  1.544003         1 -0.003411 -0.003411
2010-12-29   1.54967  1.577540  1.543758         1  0.008587  0.008587
2010-12-30   1.54291  1.576095  1.543465         1 -0.004372 -0.004372
2010-12-31   1.55861  1.574866  1.543195         1  0.010124  0.010124
2011-01-03   1.54797  1.572985  1.542855         1 -0.006850 -0.006850
2011-01-04   1.55957  1.571592  1.542591         1  0.007466  0.007466
2011-01-05   1.55039  1.570084  1.542257         1 -0.005904 -0.005904

                ⋮

2011-09-29   1.56021  1.606900  1.606505         1  0.002072  0.002072
2011-09-30   1.55792  1.604854  1.606381        -1 -0.001469 -0.001469
2011-10-03   1.54374  1.602895  1.606154        -1 -0.009144  0.009144
2011-10-04   1.54749  1.600719  1.605950        -1  0.002426 -0.002426

                ⋮

2020-12-28   1.34576  1.329781  1.282014         1 -0.007285 -0.007285
2020-12-29   1.35005  1.331144  1.282186         1  0.003183  0.003183
2020-12-30   1.36245  1.332750  1.282408         1  0.009143  0.009143
2020-12-31   1.36635  1.334523  1.282676         1  0.002858  0.002858
```

- Here, full data is deliberately outputted to check correctness manually.
- For example, after position changes from long to short, from 2011-09-29 to 2011-09-30, in 2011-10-03, ‘returns’ is -0.009144, while ‘strategy’ is +0.009144, indicating short position of SMA is successfully “opened” and SMA strategy is correctly functioning.
- While in 2011-10-04, ‘returns’ was 0.002426, price slightly increased back during short position, so ‘strategy’ has negative value of -0.002426.
- Here, only long and short positions are adopted, there is no neutral position, for simplicity and illustration purposes.

```
# Sum up asset log return values & strategy log return values
data[['returns', 'strategy']].sum()
```

```
returns    -0.124097
strategy   -0.088958
dtype: float64
```

```
# Sum up asset log return values & strategy log return values
# Then, apply exponential function to the sum to find total performance
data[['returns', 'strategy']].sum().apply(np.exp)
```

```
returns     0.883294
strategy    0.914884
dtype: float64
```

```
data[['returns', 'strategy']].cumsum(
                    ).apply(np.exp).plot(figsize=(20, 12));
```

```
plt.show()
```

- Here, a graph is plotted to show comparison between total performance of strategy (SMA) and asset return in different time.
- Visually, in around 2017, SMA strategy outperformed as divergence occurs between ‘strategy’ and ‘returns’, ‘strategy’ is increasing while ‘returns’ is decreasing.
- While in around 2015, divergence occurred again, ‘strategy’ is increasing, while ‘returns’ is decreasing, indicating total performance is getting closer and closer, even when ‘strategy’ is below 1.00, the basis point of initial capital.

5.1.2.6. Annualized mean return & standard deviation

```
# annualized mean return - regular space
data[['returns', 'strategy']].mean() * 252
```

```
returns    -0.012019
strategy   -0.008615
dtype: float64
```

```
# annualized mean return - log space
np.exp(data[['returns', 'strategy']].mean() * 252) - 1
```

```
returns    -0.011947
strategy   -0.008578
dtype: float64
```

- Here shows the annualized mean return of both regular space and log space.
- SMA strategy has higher annualized mean return in both spaces, although still negative.

```
#  annualized standard deviation - regular space
data[['returns', 'strategy']].std() * 252 ** 0.5
```

```
returns     0.088509
strategy    0.088510
dtype: float64
```

```
#  annualized standard deviation - log space
(data[['returns', 'strategy']].apply(np.exp) - 1).std() * 252 ** 0.5
```

```
returns     0.088195
strategy    0.088801
dtype: float64
```

- Here shows the annualized standard deviation of both regular space and log space.
- Both SMA strategy and asset returns have nearly the same annualized standard deviation in both spaces of around 0.088
- Getting annualized standard deviation can be beneficial because it represents volatility of both strategy and asset returns. Further potential research can be done with the help of annualized standard deviation to analyze assets characteristics. Also, a diversified portfolio can be constructed based on the knowing standard deviation of each individual asset as it helps reduce overall portfolio risk.

5.1.2.7. Maximum drawdown & longest drawdown period

```
data['cumret'] = data['strategy'].cumsum().apply(np.exp)
```

```
# Maximum drawdown:
# Def: Maximum drawdown = the largest peak-to-trough decline
# Steps:
#   1) Identify peak value
#   2) Identify lowest value which follows the peak, before a new peak is reached
# Usage: cumulative maximum gross performance as calculated
#        by the cummax() method applied to the gross performance of the strategy


# Longest drawdown period
# Def:
#   Longest drawdown period =  duration that a trade experiences a drawdown
#                              from its peak until it reaches a new peak
#   It measures how long it takes for a trade to recover from a significant loss
# Step:
#   1) Identify a peak's starting time
#   2) When a new peak is achieved, ends, record ending time
#   3) Find the longest duration


data['cummax'] = data['cumret'].cummax()
```

```
data[['cumret', 'cummax']].dropna().plot(figsize=(20, 12));
```

```
plt.show()
```

- ‘cumret’ is the cumulative asset return
- Here, a graph of maximum drawdown and longest drawdown period is plotted.

- Maximum drawdown is the largest peak-to-trough decline
    - Steps to calculate maximum drawdown:
        1. Identify peak value
        2. Identify lowest value which follows the peak, before a new peak is reached
- Longest drawdown period is the duration that a trade experiences a drawdown from its peak until it reaches a new peak. It measures how long it takes for a trade to recover from a significant loss.
    - Steps to calculate longest drawdown period:
        1. Identify a peak's starting time
        2. When a new peak is achieved, ends, record ending time
        3. Find the longest duration
- From the graph, the 2 horizontal green lines are significant long drawdown periods, especially the 1st one.

```
drawdown = data['cummax'] - data['cumret']
```

```
drawdown.max()
```

```
np.float64(0.2750134250918912)
```

```
# Frequent zero-drawdown events signal strong portfolio performance,
# while long periods without them indicate prolonged drawdowns or underperformance

# When the drawdown is 0, it means the asset is at a new cumulative high

# temp: This temporary object stores only those points (dates) where the drawdown equals zero.
# These are the dates when the portfolio reaches a new high.

temp = drawdown[drawdown == 0]
```

```
periods = (temp.index[1:].to_pydatetime() -
           temp.index[:-1].to_pydatetime())
```

```
periods.max()
```

```
datetime.timedelta(days=2453)
```

- Also, frequent zero-drawdown events signal strong portfolio performance, while long periods without them indicate prolonged drawdowns or underperformance.
- When the drawdown is 0, it means the asset is at a new cumulative high
- “Temp” is a temporary object that stores only those points (dates) where the drawdown equals zero. These are the dates when the portfolio reaches a new high.
- Here, the longest drawdown period is 2453 days, which is the longest green horizontal line in the previous drawdown graph, and is indeed a very long period of time.

5.1.3. Parameter Optimization - SMA_1 & SMA_2

- For the parameter optimization part, Backtester.py & Backtester_CodeRunner.ipynb are used.
- Brute force is implemented to find the optimal parameter combinations of SMA_1 & SMA_2.
- Heatmap is generated to show the optimization process.

5.1.3.1. Backtester - SMA_Backtester.py

- Below is the SMA_Backtester.py, with detailed explanations inside the code.
- Here, the data source is EURUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31).

```
# SMA_Backtester

import numpy as np
import pandas as pd
from scipy.optimize import brute  # Optimization of parameters
import matplotlib.pyplot as plt   # Optimization visualization


class SMA_Backtester(object):
   ''' A class designed for backtesting trading strategy based on
       Simple Moving Averages (SMA) in a vectorized manner

   Attributes
   ==========
   asset: str
       symbol of asset
   SMA_1: int
       time window (in no. of candlesticks) for shorter SMA
   SMA_2: int
       time window (in no. of candlesticks) for longer SMA
   start_time: str
       starting date for the backtesting period
   end_time: str
       ending date for the backtesting period

   Methods
   =======
   import_data:
       Loads and prepares the historical price data for backtesting
   set_parameters:
       Updates the SMA parameters (shorter and/or longer) and recalculates the respective SMA
time series
   execute_strategy:
       Executes trading strategy and evaluates its performance
   display_results:
       Creates a plot comparing the strategy's performance to that of the asset
   update_and_run:
```

```
        Updates SMA parameters and computes the negative performance (used for optimization)
    parameters_optimization:
        Uses brute force to optimize the SMA parameters for maximum strategy performance
    heatmap_visualisation:
        Generates a heatmap of strategy performance across various SMA parameter combinations.
    '''

    def __init__(self, asset, SMA_1, SMA_2, start_time, end_time):
        self.asset = asset
        self.SMA_1 = SMA_1
        self.SMA_2 = SMA_2
        self.start_time = start_time
        self.end_time = end_time
        self.results = None
        self.import_data()


    def import_data(self):
        ''' Loads and prepares the historical price data for backtesting
        '''
        # ⭐
        # illustration              = GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)
        # SMA_Backtester_CodeRunner = EURUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)
        data_imported = "EURUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)"
        fn = f'/Users/kctang/Desktop/Algo Trading Project
(S1)/Strategies/Data/{data_imported}.csv'
        raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
        print(raw_data)

        raw_data = pd.DataFrame(raw_data['close'])
        raw_data = raw_data.loc[self.start_time:self.end_time]
        raw_data.rename(columns={'close': 'price'}, inplace=True)

        raw_data['return'] = np.log(raw_data / raw_data.shift(1))
        raw_data['SMA_1'] = raw_data['price'].rolling(self.SMA_1).mean()
        raw_data['SMA_2'] = raw_data['price'].rolling(self.SMA_2).mean()

        self.data = raw_data


    def set_parameters(self, SMA_1=None, SMA_2=None):
        ''' Updates the SMA parameters (shorter and/or longer) and recalculates the respective
SMA time series
        '''
        if SMA_1 is not None:
            self.SMA_1 = SMA_1
            self.data['SMA_1'] = self.data['price'].rolling(
                self.SMA_1).mean()
        if SMA_2 is not None:
            self.SMA_2 = SMA_2
```

```
        self.data['SMA_2'] = self.data['price'].rolling(self.SMA_2).mean()


def execute_strategy(self):
    ''' Executes trading strategy and evaluates its performance
    '''
    data = self.data.copy().dropna()

    data['position'] = np.where(data['SMA_1'] > data['SMA_2'], 1, -1)
    data['strategy'] = data['position'].shift(1) * data['return']
    data.dropna(inplace=True)

    data['cum_returns'] = data['return'].cumsum().apply(np.exp)
    data['cum_strategy'] = data['strategy'].cumsum().apply(np.exp)
    self.results = data

    # Total performance of the strategy
    total_perf = data['cum_strategy'].iloc[-1]

    # out-/underperformance of strategy
    out_perf = total_perf - data['cum_returns'].iloc[-1]

    return round(total_perf, 2), round(out_perf, 2)


def display_results(self):
    ''' Creates a plot comparing the strategy's performance to that of the asset
    '''
    if self.results is None:
        print('No results to plot yet. Run a strategy.')
    title = '%s | SMA_1=%d, SMA_2=%d' % (self.asset,
                                         self.SMA_1, self.SMA_2)
    self.results[['cum_returns', 'cum_strategy']].plot(title=title,
                                                       figsize=(20, 12))
    plt.show()


def update_and_run(self, SMA):
    ''' Updates SMA parameters and computes the negative performance (used for optimization)

    Parameters
    ==========
    SMA: tuple
        SMA parameter tuple
    '''

    if SMA[0] >= SMA[1]:  # Skip invalid combinations
        return np.inf     # Return a very large value to penalize invalid combinations

    self.set_parameters(int(SMA[0]), int(SMA[1]))
```

```
        # Run strategy and return negative performance
        perf = self.execute_strategy()[0]
        print(f"Evaluating SMA_1={SMA[0]}, SMA_2={SMA[1]}: Performance={perf}")

        return -perf    # Return negative performance for minimization,
                        # as brute force method used later in parameters_optimization() is a
minimization algorithm


    def parameters_optimization(self, SMA_1_range, SMA_2_range):
        ''' Uses brute force to optimize the SMA parameters for maximum strategy performance

        Parameters
        ==========
        SMA_1_range, SMA_2_range: tuple
            tuples of the form (start_time, end_time, step size)
        '''

        print("Starting Optimization...")

        opt = brute(self.update_and_run, (SMA_1_range, SMA_2_range), finish=None)

        print("Optimization Completed")
        print(f"Evaluated Parameters: {opt}")

        return opt, -self.update_and_run(opt)    # Return positive performance back after brute
force minimization in update_and_run
                                                 # Negative * Negative = Positive, i.e. -(-perf) =
perf


    def heatmap_visualisation(self, SMA_1_range, SMA_2_range):
        ''' Generates a heatmap of strategy performance across various SMA parameter
combinations.

        Parameters
        ==========
        SMA_1_range, SMA_2_range: tuple
            Tuples of the form (start_time, end_time, step size) for SMA_1 and SMA_2.
        '''
        # Generate parameter ranges
        SMA1_values = range(SMA_1_range[0], SMA_1_range[1], SMA_1_range[2])
        SMA2_values = range(SMA_2_range[0], SMA_2_range[1], SMA_2_range[2])

        # Initialize a grid to store performance results
        results = np.zeros((len(SMA1_values), len(SMA2_values)))

        # Track the best performance and parameters
        best_perf = float('-inf')
```

```
        best_params = None

        # Evaluate all combinations of SMA_1 and SMA_2
        for i, SMA_1 in enumerate(SMA1_values):
            for j, SMA_2 in enumerate(SMA2_values):
                if SMA_1 >= SMA_2:  # Skip invalid combinations
                    results[i, j] = np.nan  # Mark as invalid
                    continue
                self.set_parameters(SMA_1, SMA_2)  # Update SMA parameters
                perf = self.execute_strategy()[0]  # Get Total performance
                results[i, j] = perf  # Store performance

                # Track the best performance
                if perf > best_perf:
                    best_perf = perf
                    best_params = (SMA_1, SMA_2)


        # Plot the heatmap
        plt.figure(figsize=(12, 8))
        im = plt.imshow(results, cmap='viridis', aspect='auto',
                        extent=[SMA_2_range[0], SMA_2_range[1], SMA_1_range[1], SMA_1_range[0]])

        plt.colorbar(im, label='Performance')  # Add color bar
        plt.title(f'Optimization Results\nBest: SMA_1={best_params[0]}, SMA_2={best_params[1]},
Perf={best_perf:.2f}')
        plt.xlabel('SMA_2 (Long)')
        plt.ylabel('SMA_1 (Short)')

        # Mark the best point on the heatmap
        plt.plot(best_params[1], best_params[0], marker='*', color='magenta', markersize=20)

        # Show: "★ = Optimal combination: SMA_1 (short) & SMA_2 (long)" at the top center of the
graph

        # use a font that supports the ★ character
        import matplotlib
        matplotlib.rcParams['font.family'] = 'Arial Unicode MS'  # Use a font that supports
Unicode

        # Add the label above the main title using suptitle
        plt.suptitle(
        "★ = Optimal combination: SMA_1 (short) & SMA_2 (long)",
        fontsize=16, color='magenta', ha='center', x=0.44, y=0.98
        )


        plt.show()
```

5.1.3.2. Backtester_CodeRunner - SMA_Backtester_CodeRunner.ipynb

- Here, in SMA_Backtester_CodeRunner.ipynb, the data source is EURUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31).

```
# SMA_Backtester_CodeRunner
```

```
import SMA_Backtester as SMA
```

```
# Run strategy

from SMA_Backtester import SMA_Backtester

SMA_BT = SMA_Backtester('EURUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31)', 42, 252, '2010-01-04', '2020-12-31')

SMA_BT.execute_strategy()
```

```
              open     high      low    close  Volume
time
2010-01-04  1.43249  1.44555  1.42563  1.44102    5704
2010-01-05  1.44099  1.44832  1.43460  1.43612    6330
2010-01-06  1.43615  1.44343  1.42830  1.44001    6198
2010-01-07  1.44004  1.44459  1.42989  1.43042    6098
2010-01-08  1.43048  1.44386  1.42636  1.44090    6642
...             ...      ...      ...      ...     ...
2020-12-25  1.21837  1.21857  1.21824  1.21840    1854
2020-12-28  1.21842  1.22503  1.21810  1.22093   90962
2020-12-29  1.22135  1.22751  1.22063  1.22499   87708
2020-12-30  1.22501  1.23099  1.22460  1.22899   96234
2020-12-31  1.22919  1.23093  1.22092  1.22172   88307

[2867 rows x 5 columns]
(np.float64(1.58), np.float64(0.64))
```

- For the standard parameter combination SMA_1 = 42 & SMA_2 = 252, total performance is 1.58 & outperformance is 0.64.

5.1.3.2.1. Heatmap - visualization of optimization process - SMA

```
# Visualization of the optimization process
```

```
# Optimizing parameters
optimal_params, optimal_perf = SMA_BT.parameters_optimization((30, 50, 1), (200, 300, 1))
```

```
Starting Optimization...
Evaluating SMA_1=30, SMA_2=200: Performance=1.28
Evaluating SMA_1=30, SMA_2=201: Performance=1.28
Evaluating SMA_1=30, SMA_2=202: Performance=1.26
Evaluating SMA_1=30, SMA_2=203: Performance=1.29
Evaluating SMA_1=30, SMA_2=204: Performance=1.27
Evaluating SMA_1=30, SMA_2=205: Performance=1.31
Evaluating SMA_1=30, SMA_2=206: Performance=1.3
Evaluating SMA_1=30, SMA_2=207: Performance=1.32
Evaluating SMA_1=30, SMA_2=208: Performance=1.31
Evaluating SMA_1=30, SMA_2=209: Performance=1.28
Evaluating SMA_1=30, SMA_2=210: Performance=1.29
Evaluating SMA_1=30, SMA_2=211: Performance=1.29

                        ⋮

Evaluating SMA_1=49, SMA_2=295: Performance=1.43
Evaluating SMA_1=49, SMA_2=296: Performance=1.43
Evaluating SMA_1=49, SMA_2=297: Performance=1.43
Evaluating SMA_1=49, SMA_2=298: Performance=1.41
Evaluating SMA_1=49, SMA_2=299: Performance=1.4
Optimization Completed
Evaluated Parameters: [ 48. 237.]
Evaluating SMA_1=48.0, SMA_2=237.0: Performance=1.67
```

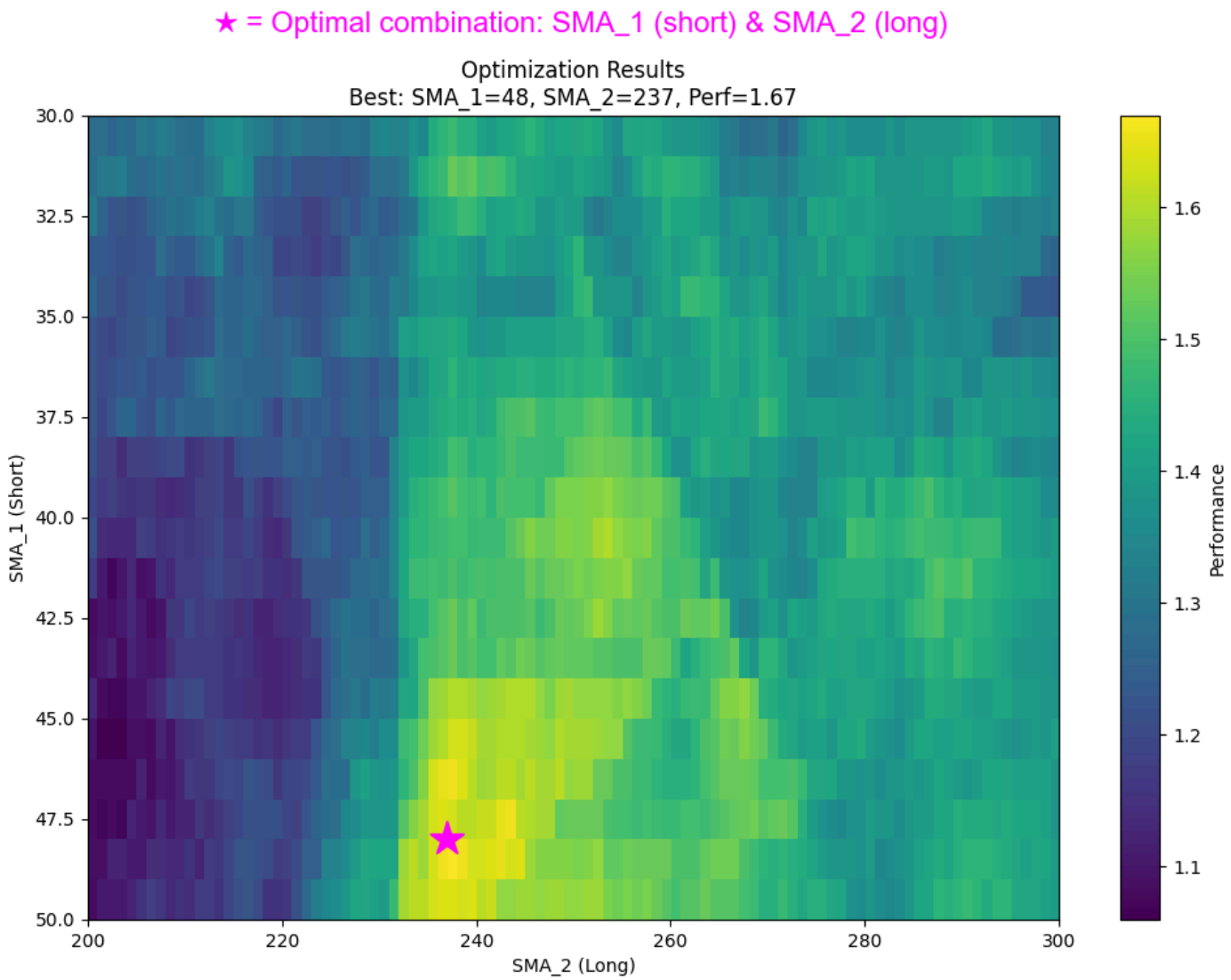
```
# Result - Optimized parameters
print("Optimal Parameters:", optimal_params)
print("Optimal Performance:", optimal_perf)
```

```
Optimal Parameters: [ 48. 237.]
Optimal Performance: 1.67
```

- Here, brute force approach is implemented to find parameter combinations that are potentially more profitable than the standard parameter combination SMA_1 = 42 & SMA_2 = 252.
- Brute force range SMA_1 = (30, 50, 1) & SMA_2 = (200, 300, 1) is implemented for shorter-term period and longer-term period SMA.
- Optimal parameters combinations of SMA_1 & SMA_2 are found to be 48 & 237, with optimal performance being 1.67 (1 is base), showing profitability.

```
# Heatmap
```

```
SMA_BT.heatmap_visualisation((30, 50, 1), (200, 300, 1))
```



- Here, apart from finding the optimal parameter combinations of SMA_1 and SMA_2, this research also generates a heatmap for visualization of the optimizing process.

- This helps more solid understanding of the parameter combinations to see whether the whole surrounding area of the optimal parameter combinations are also as profitable as the optimal one, to prevent sudden outlier.
- Here, the magenta star locates the optimal parameter combinations of SMA_1 and SMA_2 (48 & 237). The surrounding area is also yellow and yellowish green, demonstrating a locally high-performance neighborhood.
- This indicates that the optimal combination is not an isolated outlier, since it resides within a robust region of the parameter space where similar configurations yield comparably strong performance. This result suggests a potential good stability and generalizability of the SMA model's sensitivity to these parameters.

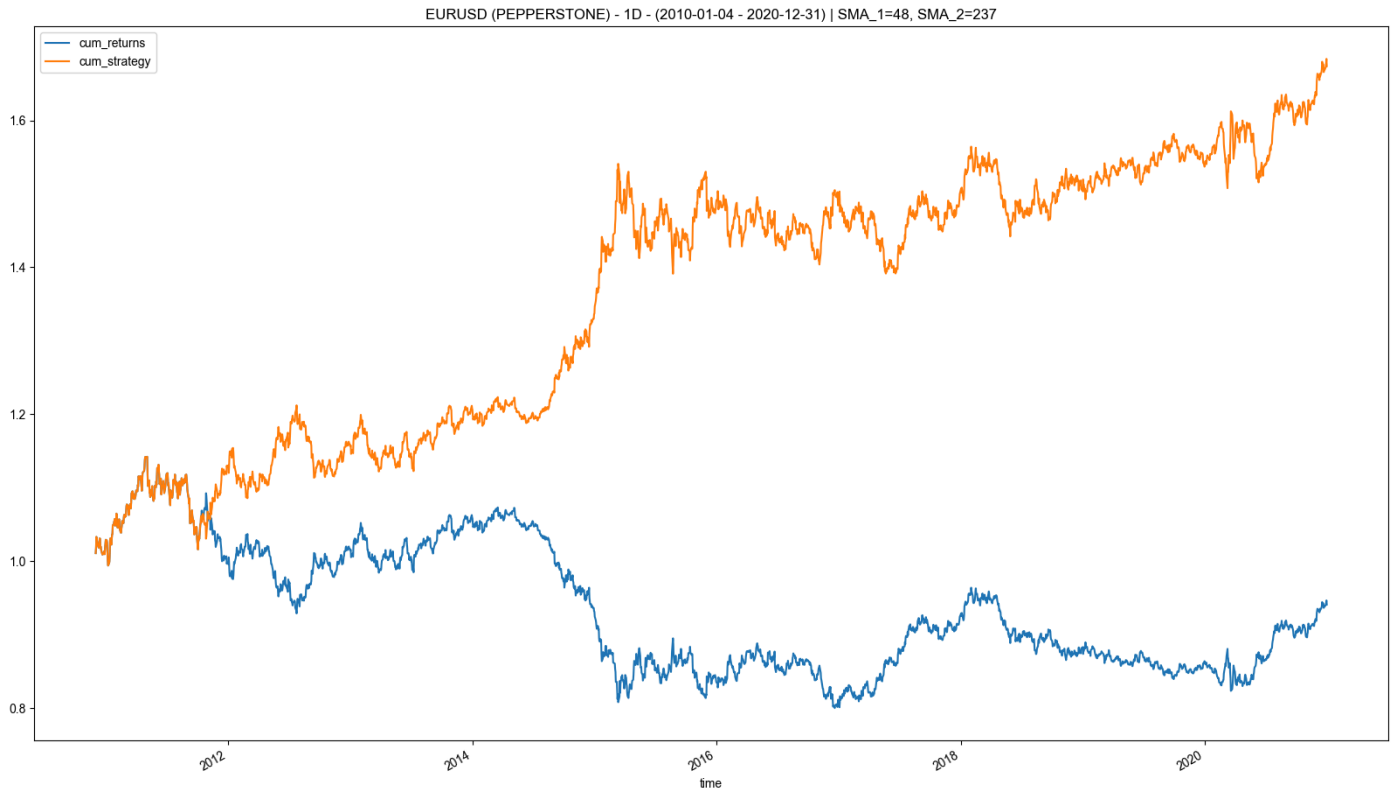
5.1.3.2.2. Optimal scenario performance - SMA

```
# Optimized scenario:
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(48, 237)
print("Total Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
print("Out-Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[1])
```

```
Total Performance for optimized parameters (best parameters): 1.67
Out-Performance for optimized parameters (best parameters): 0.73
```

```
# Plot result - optimized case
SMA_BT.display_results()
```



- Here shows the performance of the optimal parameter combinations of SMA_1 and SMA_2 (48 & 237) .

- Total Performance ('cum_strategy') is 1.67 and Out-Performance ('cum_returns') is 0.73. Not only does it show profitability, but it also demonstrates stronger performance than asset normal return.
- Also, divergence occurred during time around 2014-2015, 'cum_strategy' is increasing while 'cum_returns' is decreasing.

5.1.3.2.3. Double check heatmap correctness

```
# Double check heatmap correctness - by substituting values
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(48, 237)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 1.67
```

- Last, to double check heatmap value correctness, the above code is implemented to double check it. Here shows the consistency with heatmap result.

5.2. Momentum Strategy

5.2.1. Strategy overview

Momentum trading strategy aims to capitalize on the persistence of price trends by taking positions based on the mean of return over a specific rolling window of past n days. The idea and assumption is that if recent returns are positive, the trend will continue upward, and if recent returns are negative, the trend will continue downward.

5.2.1.1 Logic of Momentum

- Define:
    - momentum = mean of last n returns.
        - 'momentum' (or 'mom') is the mean return over a rolling window n.
        - n < number of rows of data in the dataset.
        - n can be an arbitrary value.
- If mean of last n returns > 0 (i.e. momentum > 0), then 'position' ready for next day = +1 (long position).
- If mean of last n returns = 0 (i.e. momentum = 0), then 'position' ready for next day = 0 (neutral position).
- If mean of last n returns < 0 (i.e. momentum < 0), then 'position' ready for next day = -1 (short position).

5.2.2. Implementation details: Momentum_Illustration.ipynb

- For implementation details, Momentum_Illustration - XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31).ipynb is used for illustration.
- Here, the data source is XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31).

5.2.2.1. Data import & processing - XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)

```
# Strategies Based on Momentum
```

```
import pandas as pd
import numpy as np
```

```
data_imported = "XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)"
fn = f'/Users/kctang/Desktop/Algo Trading Project  (S1)/Strategies/Data/{data_imported}.csv'
raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
print(raw_data)
```

```
              open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
```

```
data = pd.DataFrame(raw_data['close'])
data
```

time	close
2010-01-04	1121.05
2010-01-05	1117.65
2010-01-06	1138.35
2010-01-07	1131.45
2010-01-08	1138.25
...	...
2019-12-24	1499.09
2019-12-26	1511.31
2019-12-27	1510.55
2019-12-30	1514.77
2019-12-31	1517.63

2567 rows × 1 columns

- Similar to 5.1.2.1 Data import & processing - GBPUSD (PEPPERSTONE) - 1D - (2010-01-04 - 2020-12-31), import data and retrieve closing price.

```
data.rename(columns={'close': 'price'}, inplace=True)
data
```

time	price
2010-01-04	1121.05
2010-01-05	1117.65
2010-01-06	1138.35
2010-01-07	1131.45
2010-01-08	1138.25
...	...
2019-12-24	1499.09
2019-12-26	1511.31
2019-12-27	1510.55
2019-12-30	1514.77
2019-12-31	1517.63

2567 rows × 1 columns

- Here, rename column ‘close’ to ‘price’, for making column name more intuitive.

```
data['returns'] = np.log(data['price'] / data['price'].shift(1))
data
```

time	price	returns
2010-01-04	1121.05	NaN
2010-01-05	1117.65	-0.003037
2010-01-06	1138.35	0.018352
2010-01-07	1131.45	-0.006080
2010-01-08	1138.25	0.005992
...	...	...
2019-12-24	1499.09	0.008992
2019-12-26	1511.31	0.008119
2019-12-27	1510.55	-0.000503
2019-12-30	1514.77	0.002790
2019-12-31	1517.63	0.001886

2567 rows × 2 columns

- Obtain daily log return.

5.2.2.2. Comparison to benchmark for different Momentum parameters - Momentum rolling window

```
# rolling window = 1 day
```

```
# 'position' = position that ready to open the next day

# np.sign() function gives: 1 if +ve | -1 if -ve | 0 if 0
data['position'] = np.sign(data['returns'])

data['strategy'] = data['position'].shift(1) * data['returns']

data[['returns', 'strategy']].dropna().cumsum(
                    ).apply(np.exp).plot(figsize=(20, 12));
```

- Here, the rolling window is 1 day.
- The code first defines a new column [‘position’] with sign (1 or -1) of relevant log return, the value indicates ‘position’ (market positioning: 1 = long, -1 = short).
- Then, momentum strategy log return is calculated based on ‘position’.

- Logic:
    - Define:
        - momentum = mean of last 1 returns.
            - ‘momentum’ (or ‘mom’) is the mean return over a rolling window 1.
            - 1 < number of rows of data in the dataset = 2567.
    - If mean of last 1 returns > 0 (i.e. momentum > 0), then ‘position’ ready for next day = +1 (long position).
    - If mean of last 1 returns = 0 (i.e. momentum = 0), then ‘position’ ready for next day = 0 (neutral position).

- If mean of last 1 returns < 0 (i.e. momentum < 0), then 'position' ready for next day = -1 (short position).

- Then, a graph is outputted to compare strategy return (momentum strategy) and asset return, here shows underperformance and lack of profitability of strategy if set rolling window = 1 day.

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(data)
```

```
              price   returns  position  strategy
time
2010-01-04  1121.050       NaN       NaN       NaN
2010-01-05  1117.650 -0.003037      -1.0       NaN
2010-01-06  1138.350  0.018352       1.0 -0.018352
2010-01-07  1131.450 -0.006080      -1.0 -0.006080
2010-01-08  1138.250  0.005992       1.0 -0.005992
2010-01-11  1151.300  0.011400       1.0  0.011400
2010-01-12  1128.180 -0.020286      -1.0 -0.020286
2010-01-13  1137.720  0.008421       1.0 -0.008421
2010-01-14  1144.210  0.005688       1.0  0.005688
2010-01-15  1130.980 -0.011630      -1.0 -0.011630
2010-01-18  1133.200  0.001961       1.0 -0.001961
2010-01-19  1138.110  0.004324       1.0  0.004324
2010-01-20  1111.750 -0.023434      -1.0 -0.023434
2010-01-21  1094.230 -0.015884      -1.0  0.015884
2010-01-22  1092.760 -0.001344      -1.0  0.001344
2010-01-25  1097.900  0.004693       1.0 -0.004693
2010-01-26  1097.680 -0.000200      -1.0 -0.000200
2010-01-27  1088.000 -0.008858      -1.0  0.008858
2010-01-28  1085.860 -0.001969      -1.0  0.001969
2010-01-29  1080.670 -0.004791      -1.0  0.004791
2010-02-01  1106.460  0.023585       1.0 -0.023585
2010-02-02  1114.360  0.007115       1.0  0.007115
```

- Column 'returns' 1st row is NaN, because it needs to be based on previous day return performance.
- Column 'position' first 1 rows are NaN, because rolling window = 1 day, need to calculate mean of last 1 returns first before generating daily position.
- Column 'strategy' first 2 rows are NaN, because momentum strategy can only be executed after having 'previous day' position.

```
# rolling window = 3 days
# Implementing a rolling time window,
# the time series momentum strategy can be generalized to more than just the last return.
# For example, the average of the last 3 returns can be used to generate the signal for the positioning.
```

```
# Adopted average of the last 3 returns to decide direction of position

# np.sign() function gives: 1 if mean_last_3_returns = +ve | -1 if if mean_last_3_returns = -ve | 0 if if mean_last_3_returns = 0
data['position'] = np.sign(data['returns'].rolling(3).mean())

data['strategy'] = data['position'].shift(1) * data['returns']

data[['returns', 'strategy']].dropna().cumsum(
                ).apply(np.exp).plot(figsize=(20, 12));
```

- Now, set rolling window = 3, which is more than 1 in the previous example.
- Logic:
    - Define:
        - momentum = mean of last 3 returns.
            - ‘momentum’ (or ‘mom’) is the mean return over a rolling window 3.
            - 3 < number of rows of data in the dataset = 2567.
    - If mean of last 3 returns > 0 (i.e. momentum > 0), then ‘position’ ready for next day = +1 (long position).
    - If mean of last 3 returns = 0 (i.e. momentum = 0), then ‘position’ ready for next day = 0 (neutral position).
    - If mean of last 3 returns < 0 (i.e. momentum < 0), then ‘position’ ready for next day = -1 (short position).
- Here, strategy return outperformed asset return after 2015, showing profitability as well.

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(data)
```

```
              price   returns  position  strategy
time
2010-01-04  1121.050       NaN       NaN       NaN
2010-01-05  1117.650 -0.003037       NaN       NaN
2010-01-06  1138.350  0.018352       NaN       NaN
2010-01-07  1131.450 -0.006080       1.0       NaN
2010-01-08  1138.250  0.005992       1.0  0.005992
2010-01-11  1151.300  0.011400       1.0  0.011400
2010-01-12  1128.180 -0.020286      -1.0 -0.020286
2010-01-13  1137.720  0.008421      -1.0 -0.008421
2010-01-14  1144.210  0.005688      -1.0 -0.005688
2010-01-15  1130.980 -0.011630       1.0  0.011630
2010-01-18  1133.200  0.001961      -1.0  0.001961
2010-01-19  1138.110  0.004324      -1.0 -0.004324
2010-01-20  1111.750 -0.023434      -1.0  0.023434
2010-01-21  1094.230 -0.015884      -1.0  0.015884
2010-01-22  1092.760 -0.001344      -1.0  0.001344
2010-01-25  1097.900  0.004693      -1.0 -0.004693
2010-01-26  1097.680 -0.000200       1.0  0.000200
2010-01-27  1088.000 -0.008858      -1.0 -0.008858
2010-01-28  1085.860 -0.001969      -1.0  0.001969
2010-01-29  1080.670 -0.004791      -1.0  0.004791
2010-02-01  1106.460  0.023585       1.0 -0.023585
```

- Now, check the correctness of the implementation of momentum strategy.
- Column ‘returns’ 1st row is NaN, because need to base on previous day return performance
- Column ‘position’ first 3 rows are NaN, because rolling window = 3 day, need to calculate mean of last 3 returns first before generating daily position.
- Column ‘strategy’ first 4 rows are NaN, because momentum strategy can only be executed after having ‘previous day’ position.

5.2.3. Parameter Optimization - Momentum rolling window

- For the parameter optimization part, Backtester.py & Backtester_CodeRunner.ipynb are used.
- Brute force is implemented to find the optimal parameter Momentum rolling window (‘mom’).
- Heatmap is generated to show the optimization process.
- Here, initial capital and transaction cost are included as well, and is set to be 10000 USD and 0.1% respectively to mimic live trading.

5.2.3.1. Backtester -Momentum_Backtester.py

- Below is the Momentum_Backtester.py, with detailed explanations inside the code.
- Here, same with illustration code, the data source is also XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31).

```
# Momentum_Backtester

import numpy as np
import pandas as pd

# Visualization of optimizing parameters: heatmap_visualisation
import matplotlib.pyplot as plt
import seaborn as sns
```

```
class Momentum_Backtester(object):
    ''' A class designed for backtesting trading strategy based on
        Momentum in a vectorized manner

    Attributes
    ==========
    asset: str
        symbol of asset
    start_time: str
        starting date for the backtesting period
    end_time: str
        ending date for the backtesting period
    capital: int, float
        capital to be invested at the beginning
    t_cost: float
        proportional transaction costs (e.g. 0.1% = 0.001) per trade

    Methods
    =======
    import_data:
        Loads and prepares the historical price data for backtesting
    execute_strategy:
        Executes trading strategy and evaluates its performance
    display_results:
        Creates a plot comparing the strategy's performance to that of the asset
    parameters_optimization:
        Optimizes the momentum parameter by testing a range of values
        and identifying the one with the best performance
    display_results:
        Creates a plot comparing the strategy's performance to that of the asset
parameters_optimization:
                Optimizes the momentum parameter by testing a range of values
                and identifying the one with the best performance
    parameters_optimization:
        Optimizes the momentum parameter by testing a range of values
        and identifying the one with the best performance
    heatmap_visualisation:
        Visualizes the optimization results using a heatmap to highlight
        the best momentum parameter and its corresponding performance
    '''

    def __init__(self, asset, start_time, end_time, capital, t_cost):
        self.asset = asset
        self.start_time = start_time
        self.end_time = end_time
        self.capital = capital
        self.t_cost = t_cost
        self.results = None
```

```
        self.import_data()


    def import_data(self):
        ''' Loads and prepares the historical price data for backtesting
        '''
        # ⭐
        # illustration = XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)
        # CodeRunner   = XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)
        data_imported = "XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)"
        fn = f'/Users/kctang/Desktop/Algo Trading Project
(S1)/Strategies/Data/{data_imported}.csv'
        raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
        print(raw_data)

        raw_data = pd.DataFrame(raw_data['close'])
        raw_data = raw_data.loc[self.start_time:self.end_time]
        raw_data.rename(columns={'close': 'price'}, inplace=True)

        raw_data['return'] = np.log(raw_data / raw_data.shift(1))

        self.data = raw_data


    def execute_strategy(self, momentum=1):
        ''' Executes trading strategy and evaluates its performance
        '''
        self.momentum = momentum

        data = self.data.copy().dropna()

        data['position'] = np.sign(data['return'].rolling(momentum).mean())
        data['strategy'] = data['position'].shift(1) * data['return']

        # determine when a trade takes place
        data.dropna(inplace=True)

        #
        # .diff(): calculates difference between current day value and previous day value
            # If the position changes from +1 to -1, the difference will be -2.
            # If the position remains the same (e.g., +1 to +1), the difference will be 0.
            # eg.
            # Date      position  .diff()
            # 2024-11-20    0       NaN
            # 2024-11-21    +1      +1
            # 2024-11-22    +1      0
            # 2024-11-23    -1      -2
            # 2024-11-24    0       +1
            # 2024-11-25    +1      +1
```

```
        # .fillna(0):
            # .diff() operation leaves the first value as NaN because there is no previous value
to subtract
            # fillna(0) replaces this NaN with 0,
            # treating it as no change in position (since there is no prior position to compare
with)
            # eg.
            # Date      position  .diff()
            # 2024-11-20    0       0 (NaN -> 0)
            # 2024-11-21    +1      +1
            # 2024-11-22    +1      0
            # 2024-11-23    -1      -2
            # 2024-11-24    0       +1
            # 2024-11-25    +1      +1

        # != 0:
            # If True, it means a trade occurred on that day (because the position has changed)
            # Date          position    .diff()     trades (but in separate dataframe)
            # 2024-11-20    0           0           False
            # 2024-11-21    +1          +1          True
            # 2024-11-22    +1          0           False
            # 2024-11-23    -1          -2          True
            # 2024-11-24    0           +1          True
            # 2024-11-25    +1          +1          True
        trades = data['position'].diff().fillna(0) != 0

        # subtract transaction costs from return when trade takes place

        # data['strategy'][trades]:
            # "trades" is used as a boolean mask,
            # to filter the strategy column to include only the rows where trades is True

        # initial:
        # Date          Strategy    Trades
        # 2010-01-08    0.003704    False
        # 2010-01-11    0.014419    False
        # 2010-01-12    -0.022195   True
        # 2010-01-13    -0.009798   True
        # 2010-01-14    0.003901    True

        # filtered series:
        # Date          Strategy
        # 2010-01-12    -0.022195
        # 2010-01-13    -0.009798
        # 2010-01-14    0.003901

        # after "-= self.t_cost":
            # eg. self.t_cost = 0.001  # Transaction cost = 0.1%

        # Date          Strategy
```

```
        # 2010-01-12: -0.023195 (-0.022195 - 0.001 = -0.023195)
        # 2010-01-13: -0.010798 (-0.009798 - 0.001 = -0.010798)
        # 2010-01-14:  0.002901 (0.003901 - 0.001 = 0.002901)

        # Date         Price     Return     Position   Strategy
        # 2010-01-08   1136.10   0.003704   1.0        0.003704 (unchanged, no new transaction
cost occured as position unchanged)
        # 2010-01-11   1152.60   0.014419   1.0        0.014419 (unchanged, no new transaction
cost occured as position unchanged)
        # 2010-01-12   1127.30  -0.022195  -1.0       -0.023195 (updated: -= self.t_cost)
        # 2010-01-13   1138.40   0.009798   1.0       -0.010798 (updated: -= self.t_cost)
        # 2010-01-14   1142.85   0.003901  -1.0        0.002901 (updated: -= self.t_cost)

        #
        # data['strategy'][trades] -= self.t_cost: also works,
        # but will give: FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!

        # data.loc[trades, 'strategy'] -= self.t_cost will modify the strategy column in the rows
where trades is True,
        # directly on the original DataFrame, avoiding the chained assignment issue
        data.loc[trades, 'strategy'] -= self.t_cost

        data['cum_returns'] = self.capital * data['return'].cumsum().apply(np.exp)
        data['cum_strategy'] = self.capital * data['strategy'].cumsum().apply(np.exp)

        self.results = data

        # Total performance of the strategy
        total_perf = self.results['cum_strategy'].iloc[-1]

        # outperformance or underperformance of strategy
        out_perf = total_perf - self.results['cum_returns'].iloc[-1]

        return round(total_perf, 2), round(out_perf, 2)


    def display_results(self):
        ''' Creates a plot comparing the strategy's performance to that of the asset
parameters_optimization:
                Optimizes the momentum parameter by testing a range of values
                and identifying the one with the best performance
        '''
        if self.results is None:
            print('No results to plot yet. Run a strategy.')
        title = '%s | t_cost = %.4f' % (self.asset, self.t_cost)
        self.results[['cum_returns', 'cum_strategy']].plot(title=title,
                                                           figsize=(20, 12))
```

```
    def parameters_optimization(self, momentum_range):
        ''' Optimizes the momentum parameter by testing a range of values
        and identifying the one with the best performance

        Parameters
        ==========
        momentum_range: range
            range of momentum values to test
        '''
        performances = []
        for momentum in momentum_range:
            total_perf, _ = self.execute_strategy(momentum)
            performances.append((momentum, total_perf))

        # Convert results to a DataFrame
        results = pd.DataFrame(performances, columns=['momentum', 'performance'])
        best_momentum = results.loc[results['performance'].idxmax(), 'momentum']
        best_performance = results['performance'].max()

        print(f'Best Momentum: {best_momentum}, Best Performance: {best_performance}\n')
        return results


    def heatmap_visualisation(self, results):
        ''' Visualizes the optimization results using a heatmap to highlight
        the best momentum parameter and its corresponding performance

        Parameters
        ==========
        results: DataFrame
            Contains momentum values and their corresponding performance.
        '''
        # Restructure the data: Performance on x-axis, Momentum on y-axis
        heatmap_data = results.pivot(index="momentum", columns="performance",
values="performance")

        # Find the max performance and its coordinates
        max_index = results['performance'].idxmax()
        max_momentum = results.loc[max_index, 'momentum']
        max_performance = results.loc[max_index, 'performance']

        # Plot the heatmap
        # can change figsize for longer height, eg. 6 -> 100
        #   standard : plt.figure(figsize=(25, 6))
        #   longer   : plt.figure(figsize=(25, 100))
        #   longest  : plt.figure(figsize=(25, 500))
        # ⭐
        plt.figure(figsize=(25, 6))
        ax = sns.heatmap(
            heatmap_data,  # Data for the heatmap
```

```
            cmap="RdYlGn",  # Red for low, green for high
            annot=True,  # Annotate cells with numerical values
            fmt=".2f",  # Format numbers to 2 decimal places
            center=self.capital,  # Dynamically center at the initial capital
            cbar=True,  # Add a color bar

        )

        # Rectangle dimensions (wider for long values)
        rect_x = heatmap_data.columns.get_loc(max_performance) - 0.6  # start_time further left
        rect_y = heatmap_data.index.get_loc(max_momentum) - 0.4
        rect_width = 2.2  # Increased width
        rect_height = 1.8

        # Add a magenta rectangle around the max performance
        rect = plt.Rectangle(
            (rect_x, rect_y),  # Bottom-left corner of the rectangle
            rect_width,  # Width of the rectangle
            rect_height,  # Height of the rectangle
            fill=False, color='magenta', linewidth=3
        )
        ax.add_patch(rect)

        # Add magenta extended lines (excluding any overlap with the rectangle)
        max_x = heatmap_data.columns.get_loc(max_performance) + 0.5
        max_y = heatmap_data.index.get_loc(max_momentum) + 0.5

        # Calculate the gap boundaries (so the lines stop at rectangle edges)
        x_gap_start = max_x - (rect_width / 2)  # Left edge of the rectangle
        x_gap_end = max_x + (rect_width / 2)  # Right edge of the rectangle

        # Horizontal line (left side only, stopping at the rectangle)
        plt.hlines(
            y=max_y, xmin=0, xmax=x_gap_start, color='magenta', linestyle='-', linewidth=1.5,
alpha=0.8
        )

        # Vertical line (only below the rectangle, no overlap inside the rectangle)
        plt.vlines(
            x=max_x, ymin=max_y + (rect_height / 2), ymax=len(heatmap_data.index),
            color='magenta', linestyle='-', linewidth=1.5, alpha=0.8
        )

        # Extend x-axis limits to ensure the rectangle is fully visible
        plt.xlim(-0.5, len(heatmap_data.columns) + 1)  # Add extra space on the right

        # Title and labels
        plt.title("Momentum Optimization Heatmap")
        plt.xlabel("Performance")  # x-axis label
        plt.ylabel("Momentum")  # y-axis label
```

```
    # Show the plot
    plt.show()
```

5.2.3.2. Backtester_CodeRunner - Momentum_Backtester_CodeRunner

5.2.3.2.1. transaction cost = 0

```
# Momentum_Backtester_CodeRunner
```

```
import Momentum_Backtester as Mom
```

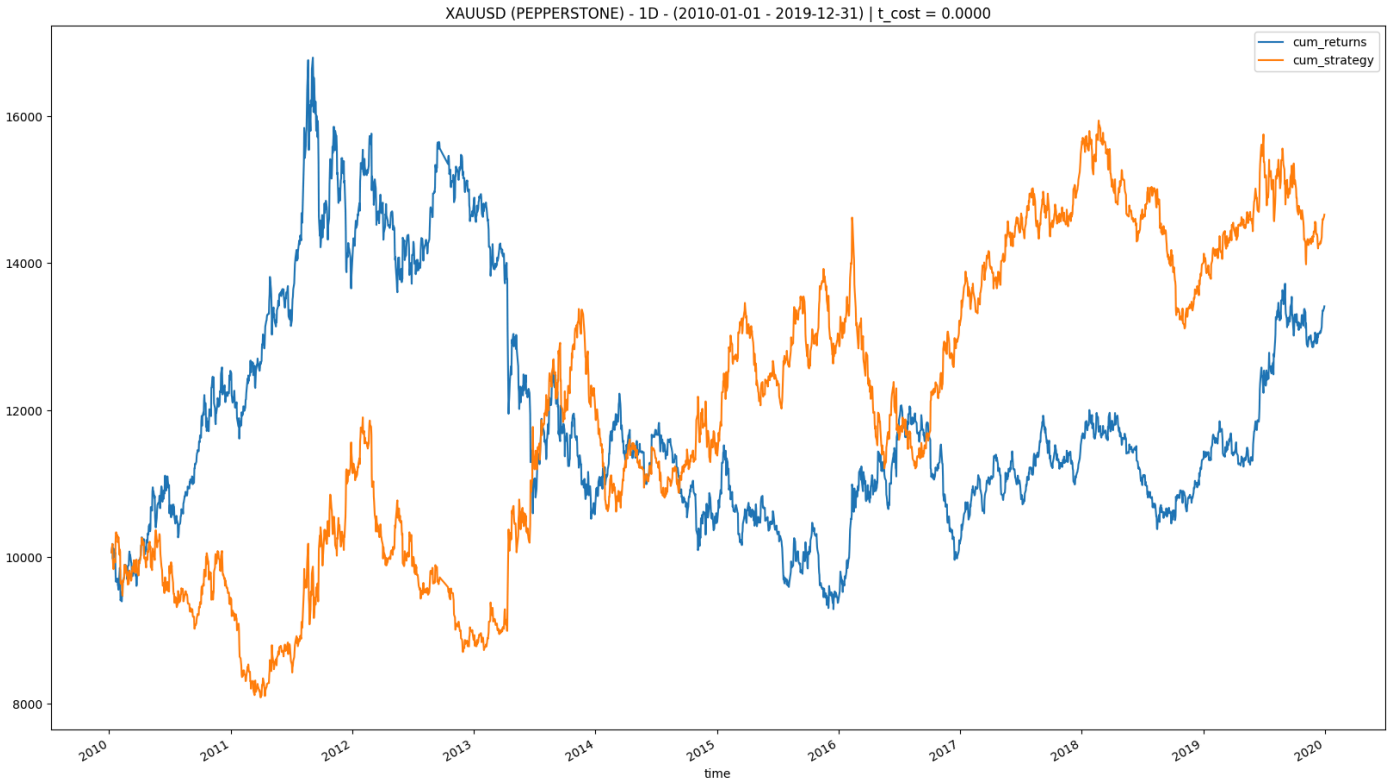
```
MOM_BT = Mom.Momentum_Backtester('XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)', '2010-1-1', '2019-12-31', 10000, 0.0)
```

```
              open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
```

```
MOM_BT.execute_strategy(momentum = 3)
```

```
(np.float64(14662.19), np.float64(1249.05))
```

```
MOM_BT.display_results()
```



- Here, set initial capital 10000 USD and transaction cost = 0.0.

- Total performance (‘cum_strategy’) outperformed asset return in general starting from 2015, and exceeded initial capital 10000 USD in general.

5.2.3.2.2. transaction cost = 0.001 (0.1%)

```
MOM_BT = Mom.Momentum_Backtester('Asset', '2010-1-1', '2019-12-31', 10000, 0.001)
```

```
              open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
```

```
MOM_BT.execute_strategy(momentum=3)
```

```
(np.float64(7398.46), np.float64(-6014.68))
```

```
MOM_BT.display_results()
```

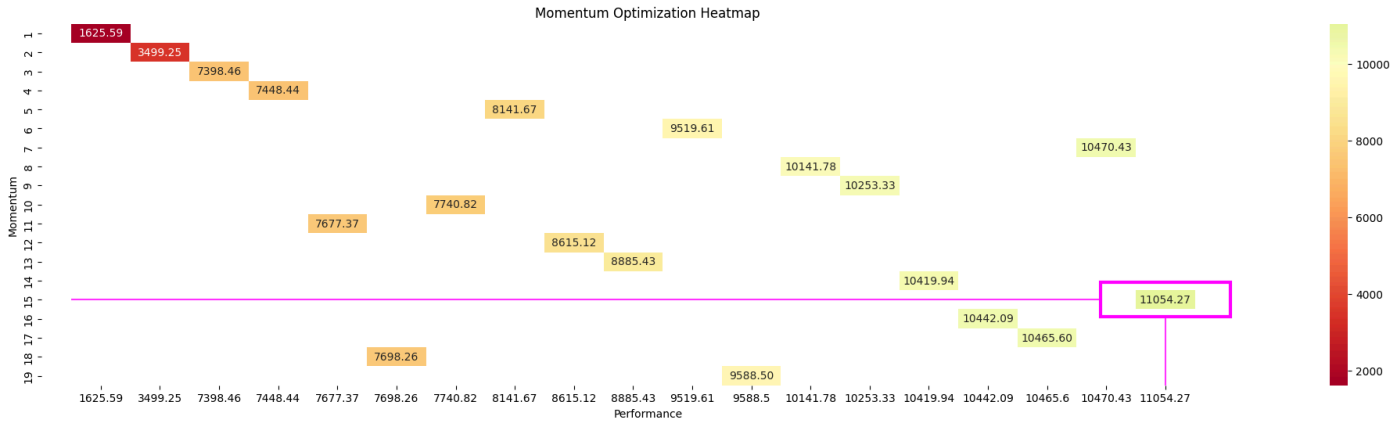
- Here, initial capital 10000 USD is unchanged but the transaction cost is adjusted to 0.001 to mimic real life scenarios.
- Total performance (‘cum_strategy’) undeformed asset return throughout the whole time horizon and is below initial capital 10000 USD in general, highlighting transaction cost’s significant effect of how it can greatly reduce strategy overall returns.

5.2.3.2.3. Heatmap - visualization of optimization process - Momentum

```
# Optimization of parameters: momentum (average of last n returns)
# Visualization of optimization process
```

```
# Import the backtesting module
import Momentum_Backtester as Mom

# Initialize the backtester object
MOM_BT = Mom.Momentum_Backtester('XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)', '2010-1-1', '2019-12-31', 10000, 0.001)

# Momentum = 1-20
# Define the range of momentum values to test
momentum_range = range(1, 20)

# Run the optimization
optimization_results = MOM_BT.parameters_optimization(momentum_range)

# Visualize the optimization process with a heatmap
MOM_BT.heatmap_visualisation(optimization_results)
```

```
              open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
Best Momentum: 15, Best Performance: 11054.27
```



- After understanding the significance of transaction cost, now include transaction cost of 0.1% and initial capital 10000 USD in parameter optimization process as well, to increase robustness.
- Here, set Momentum rolling window range 1 to 20 and implement brute force to find optimal Momentum rolling window (Momentum).
- Optimal Momentum is 15 with performance 11054.27 > 10000 initial capital, showing profitability.

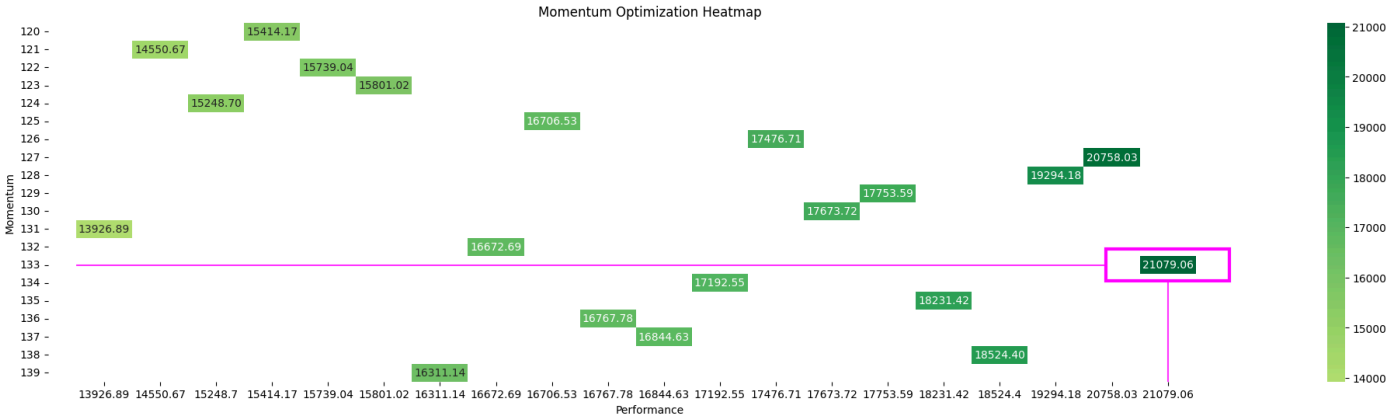
```python
# Momentum = 1-2000
# i.e. when momentum = 2000, use Average(last 2000 returns) to compare and make trading decision
# Screen nearly all possible momentum values to find most optimal value of "Momentum"

# Import the backtesting module
import Momentum_Backtester as Mom

# Initialize the backtester object
MOM_BT = Mom.Momentum_Backtester('XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)', '2010-1-1', '2019-12-31', 10000, 0.001)

# Define the range of momentum values to test
momentum_range = range(1, 2000)

# Run the optimization
optimization_results = MOM_BT.parameters_optimization(momentum_range)

# Visualize the optimization process with a heatmap
MOM_BT.heatmap_visualisation(optimization_results)
```

```
               open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
Best Momentum: 133, Best Performance: 21079.06
```

- Now, set the Momentum rolling window range 1 to 2000 instead of 1 to 20 to find optimal Momentum.
- Optimal momentum is 133, with performance 21079.06 > 10000 and is +110.79% return, showing good profitability considering no leverage is used and gold is a significant asset due to its high trading volume, liquidity and global demand.

```
# Zoom in optimal parameter result of Momentum = 130

# Import the backtesting module
import Momentum_Backtester as Mom

# Initialize the backtester object
MOM_BT = Mom.Momentum_Backtester('XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)', '2010-1-1', '2019-12-31', 10000, 0.001)

# Define the range of momentum values to test
momentum_range = range(120,140)

# Run the optimization
optimization_results = MOM_BT.parameters_optimization(momentum_range)

# Visualize the optimization process with a heatmap
MOM_BT.heatmap_visualisation(optimization_results)
```

```
              open     high      low    close  Volume
time
2010-01-04  1097.45  1124.00  1093.08  1121.05   72388
2010-01-05  1121.20  1128.85  1115.40  1117.65   37326
2010-01-06  1117.35  1140.65  1116.40  1138.35   41250
2010-01-07  1138.50  1139.20  1128.60  1131.45   39638
2010-01-08  1129.15  1139.70  1119.80  1138.25   41170
...             ...      ...      ...      ...     ...
2019-12-24  1485.31  1500.48  1484.38  1499.09   90344
2019-12-26  1499.98  1512.60  1497.39  1511.31  131109
2019-12-27  1511.07  1515.32  1507.35  1510.55  132320
2019-12-30  1511.56  1516.11  1510.79  1514.77  108420
2019-12-31  1514.17  1525.26  1514.17  1517.63  127252

[2567 rows x 5 columns]
Best Momentum: 133, Best Performance: 21079.06
```



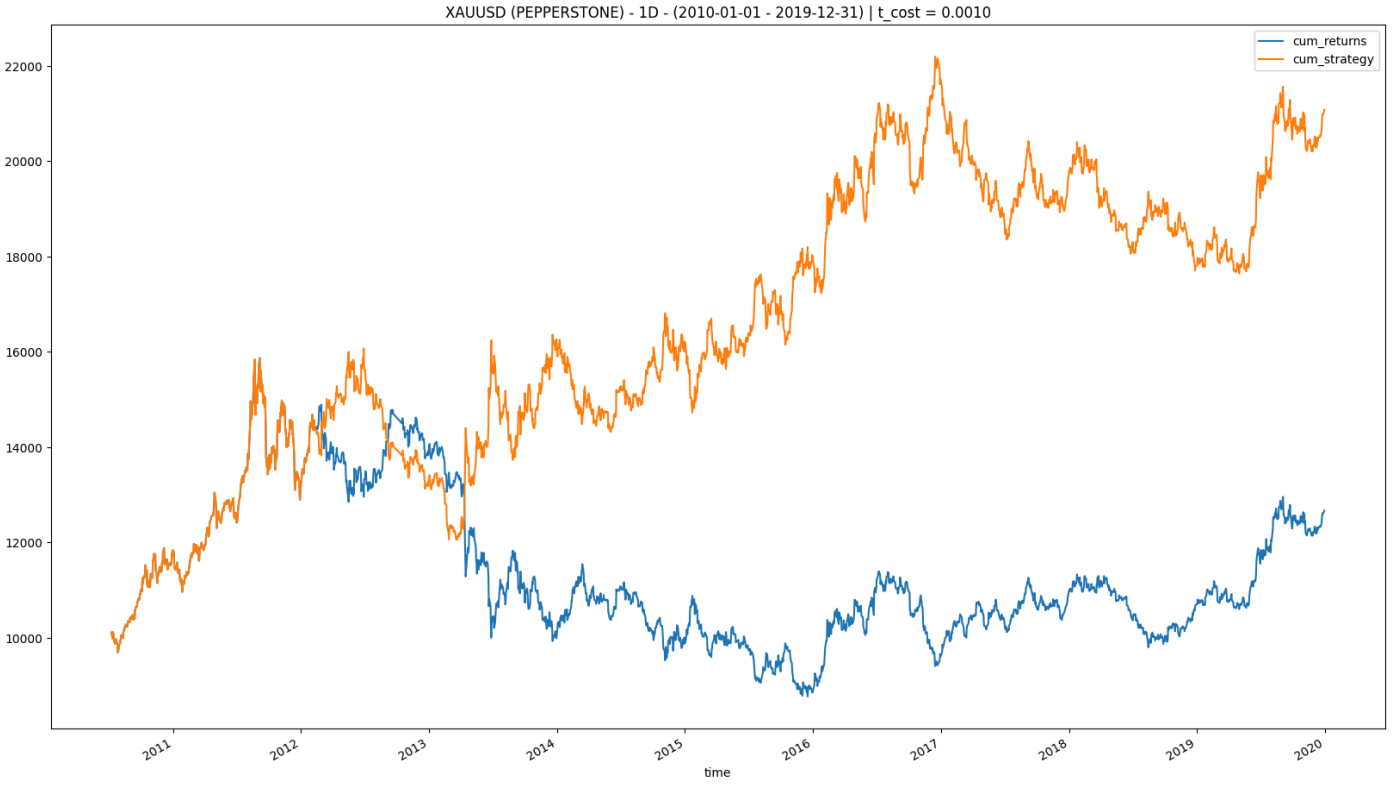
- Since a heatmap of 2000 x 2000 data is vertically too long to be shown on Jupyter Notebook, now zoom in by adjusting the range 120 to 140.
    - To show it, adjust plt.figure(figsize=(25, 6)) to plt.figure(figsize=(25, 500)) (code line 246) in Momentum_Backtester.py heatmap_visualisation method.
- Optimal momentum parameter is automatically pointed by the magenta rectangle.
- The heatmap shows good profitability in general, values surrounding the magenta rectangle also show strong performance as they are all quite dark green color, further underscoring the robustness of the strategy in this situation, implying that momentum strategy is not overly sensitive to minor changes in parameters, making it a potential reliable choice for real money trading.

5.2.3.2.4. Optimal scenario performance - Momentum

```
# Performance - optimized parameters (best parameters)
print("Total Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(133)[0])
print("Out-Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(133)[1])
```

```
Total Performance for optimized parameters (best parameters): 21079.06
Out-Performance for optimized parameters (best parameters): 8410.08
```

```
# Plot result - optimized case
MOM_BT.display_results()
```




```
# Comparison to benchmark:

    # simply investing in the asset (XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)) and do nothing:
        # (data 1st day opening price - data last day closing price) / data 1st day opening price
        # = (1517.63 - 1097.45)/1097.45 = + 38.29 %

    # Momentum strategies (Momentum = 133):
        # = (21079.06-10000)/10000 = + 110.79%
        #                          > + 38.29 %
```

- Here, it shows the outperformance of 8410.08 USD as well apart from the total performance 21079.06 USD from 10000 USD.
- The graph also shows some level of divergence starting from 2013, indicating momentum strategy outperform asset return (gold) in the broker Pepperstone
- And compared to the benchmark of simply investing in the asset and doing nothing, Momentum strategy has + 110.79% return, which is higher than the + 38.29 % of 'simply investing', showing potential profitability of Momentum strategies.

5.3. Mean Reversion Strategy

5.3.1. Strategy overview

Mean reversion strategy is a trading style based on the belief that asset prices tend to oscillate around their historical average (mean) over time. The strategy aims to capitalize on price deviations from the mean by identifying overbought or oversold scenarios. When prices move significantly above or below their average, anticipate a return back to the mean and an open position to profit from the correction.

5.3.1.1. Logic of Mean Reversion

- Define:
    - Mean:
        - Mean = SMA_t
        - $SMA_t = (1/N) * \Sigma (P_{t-i})$ for i = 1 to N
    - Distance from the Mean:
        - $Distance_t = Price_t - SMA_t$
        - It means the current distance between the current price and the mean.
        - A positive distance indicates that the price is above the mean, while a negative distance indicates that it is below the mean.
    - Threshold:
        - Threshold is a pre-set value used to classify significant deviations from the mean.
        - $Upper\_Threshold_t = SMA_t + Threshold$
        - $Lower\_Threshold_t = SMA_t - Threshold$
        - If $Distance_t$ exceeds $Upper\_Threshold_t$, $Price_t$ is considered overbought. Open a short position.
        - If $Distance_t$ falls below $Lower\_Threshold_t$, $Price_t$ is considered oversold. Open a long position..
        - After opening position, if $Distance_t = 0$ ($Price_t = SMA_t$), close position, otherwise, keep position until $Distance_t$ becomes 0.

5.3.2. Implementation details: Mean_Reversion_Illustration.ipynb

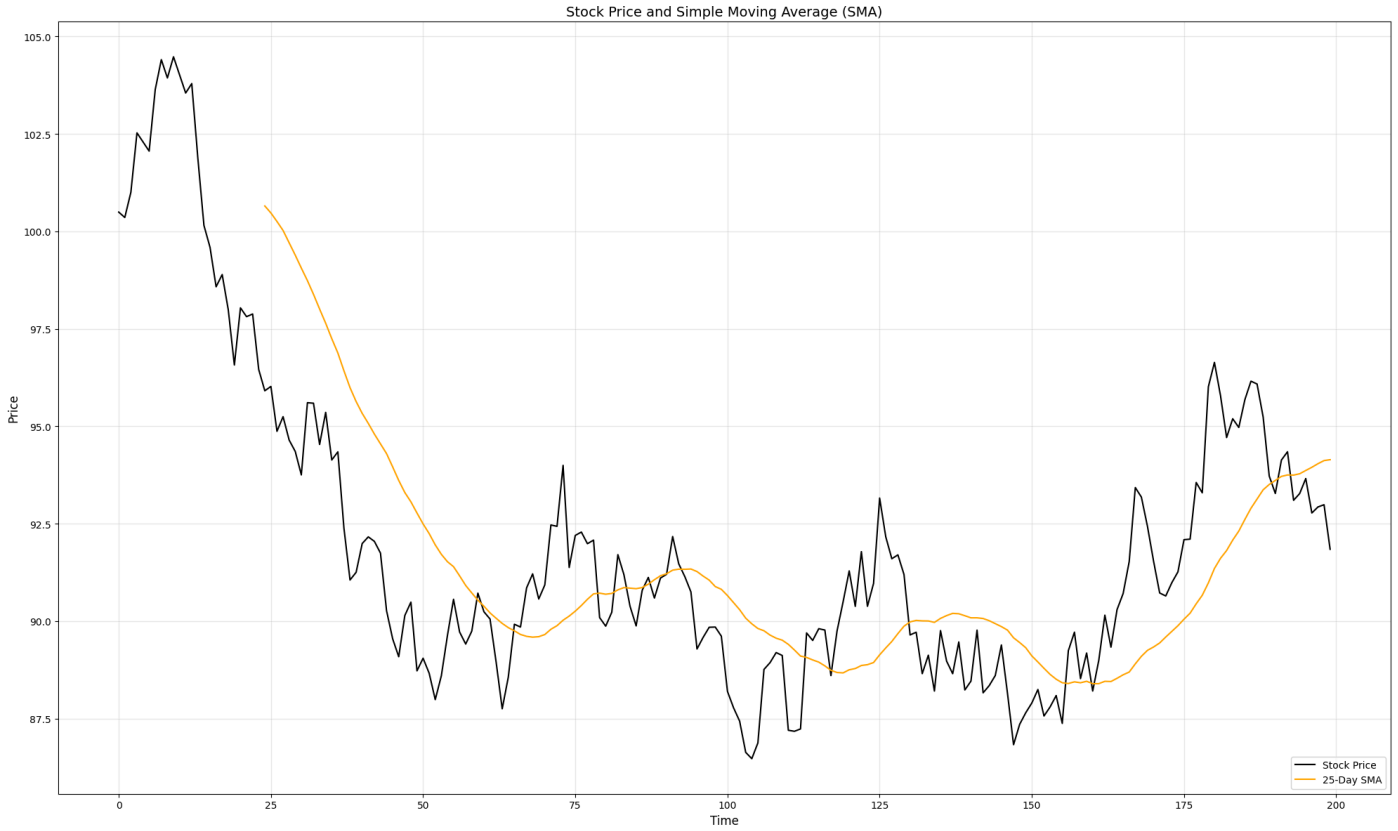
- For implementation details, Mean_Reversion_Illustration - UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ).ipynb is used for illustration.
- Here, the data source is UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ).

- This is 15 min time frame data, instead of previous Daily time frame data, to show a variety of data sources.

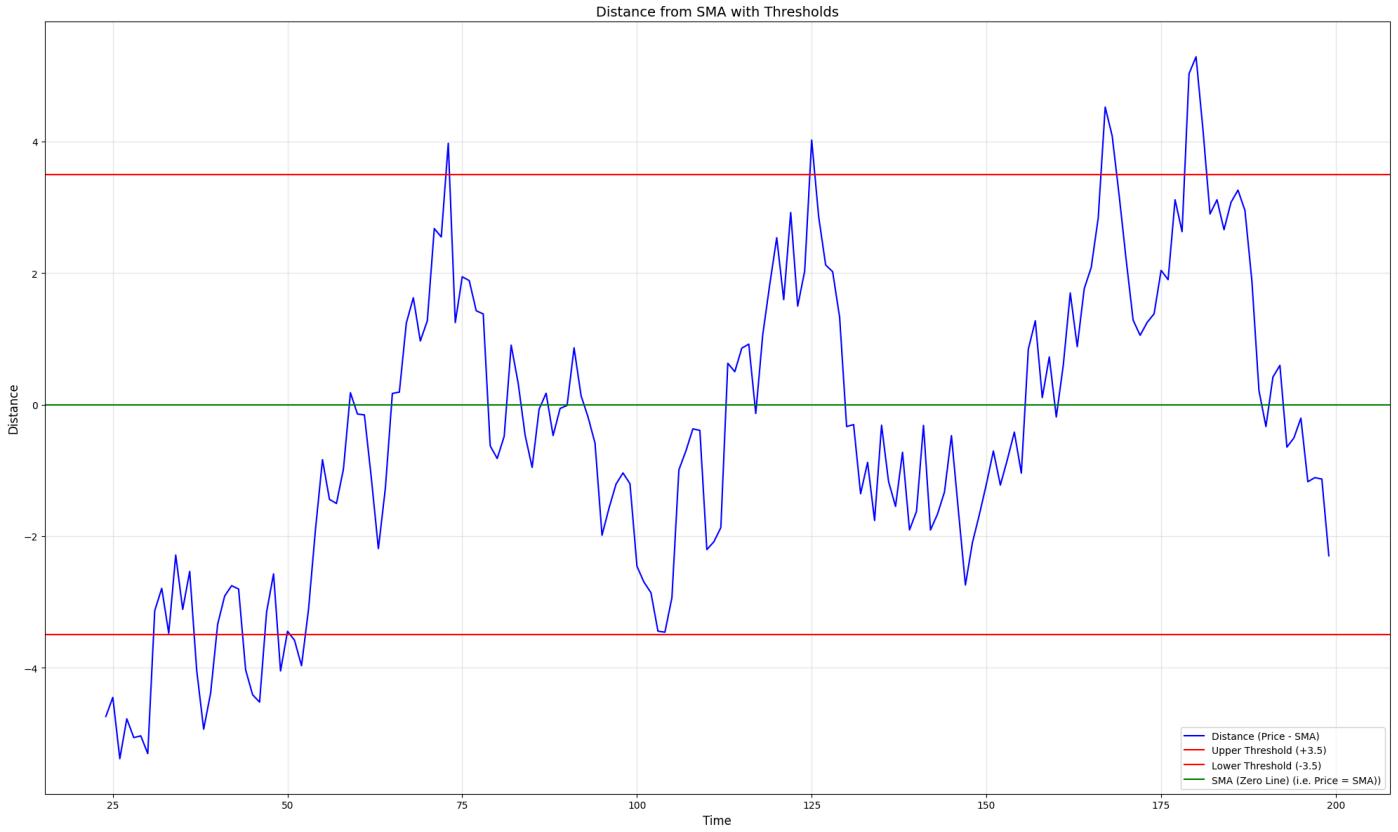
5.3.2.1. Visualization of strategy idea - Mean Reversion

```
# # Strategies Based on Mean_Reversion
```

```
# General idea:
# to define a threshold for the distance between current asset price and  SMA,
# which signals a long or short position

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Simulated Stock Price Data
np.random.seed(42)
price = np.cumsum(np.random.normal(0, 1, 200)) + 100  # Random walk starting at 100
data = pd.DataFrame(price, columns=['price'])

# Calculate SMA
SMA = 25
data['SMA'] = data['price'].rolling(SMA).mean()

# Calculate Distance
data['distance'] = data['price'] - data['SMA']

# Define Thresholds
threshold = 3.5

# Plot 1: Stock Price and SMA
plt.figure(figsize=(20, 12))
plt.plot(data['price'], label='Stock Price', color='black', linewidth=1.5)
plt.plot(data['SMA'], label=f'{SMA}-Day SMA', color='orange', linestyle='-', linewidth=1.5)
plt.title('Stock Price and Simple Moving Average (SMA)', fontsize=14)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Price', fontsize=12)
plt.legend(loc='lower right', fontsize=10)  # Legend in lower right corner
plt.grid(alpha=0.3)
plt.tight_layout()  # Adjust layout
plt.show()

# Plot 2: Distance from SMA with Thresholds
plt.figure(figsize=(20, 12))
plt.plot(data['distance'], label='Distance (Price - SMA)', color='blue', linewidth=1.5)
plt.axhline(threshold, color='red', linestyle='-', label=f'Upper Threshold (+{threshold})')
plt.axhline(-threshold, color='red', linestyle='-', label=f'Lower Threshold (-{threshold})')
plt.axhline(0, color='green', linestyle='-', label='SMA (Zero Line) (i.e. Price = SMA))')
plt.title('Distance from SMA with Thresholds', fontsize=14)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Distance', fontsize=12)
plt.legend(loc='lower right', fontsize=10)  # Legend in lower right corner
plt.grid(alpha=0.3)
plt.tight_layout()  # Adjust layout
plt.show()
```



- Orange line is SMA_t, used as the “mean”.



- Here is the visualization of the strategy idea of mean reversion.

- Description:
    - Red lines are the $Upper\_Threshold_t$ and $Lower\_Threshold_t$ respectively.

    - Blue line is the $Distance_t$ ($Distance_t = Price_t - SMA_t$).
    - Green line is the SMA_t, acting as a zero line here.

- Trading logic:
    - When the blue line touches the red $Upper\_Threshold_t$ line, a short position will be opened.
    - When the blue line touches the red $Lower\_Threshold_t$ line, a long position will be opened.
    - When the blue line touches the green line SMA_t, close position if previously opened a position.

5.3.2.2. Data import & processing - UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ)

- The 15 min UK100 data has the date range from 2024-05-01 00:00:00+00:00 to 2024-11-29 20:45:00+00:00

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
data_imported = "UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ)"
fn = f'/Users/kctang/Desktop/Algo Trading Project  (S1)/Strategies/Data/{data_imported}.csv'
raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
print(raw_data)
```

```
                           open    high     low   close           MA  Volume
time
2024-05-01 00:00:00+00:00  8128.7  8134.0  8125.8  8128.1  8164.353191     690
2024-05-01 00:15:00+00:00  8128.2  8131.7  8125.3  8128.7  8163.442553     536
2024-05-01 00:30:00+00:00  8128.8  8130.0  8126.2  8127.0  8162.523404     286
2024-05-01 00:45:00+00:00  8126.9  8132.7  8124.3  8132.4  8161.538298     344
2024-05-01 01:00:00+00:00  8132.6  8134.9  8130.6  8132.1  8160.546809     380
...                           ...     ...     ...     ...          ...     ...
2024-11-28 17:00:00+00:00  8289.1  8290.1  8287.1  8287.1  8285.661702     272
2024-11-28 17:15:00+00:00  8286.8  8288.6  8286.1  8288.1  8285.627660     192
2024-11-28 17:30:00+00:00  8287.8  8292.6  8287.1  8292.6  8285.661702     172
2024-11-28 17:45:00+00:00  8293.1  8293.8  8290.8  8291.7  8285.638298     272
2024-11-28 18:00:00+00:00  8291.8  8291.8  8290.6  8291.0  8285.623404      72

[11903 rows x 6 columns]
```

```
data = pd.DataFrame(raw_data['close'])
data.rename(columns={'close': 'price'}, inplace=True)
data
```

time	price
2024-05-01 00:00:00+00:00	8128.1
2024-05-01 00:15:00+00:00	8128.7
2024-05-01 00:30:00+00:00	8127.0
2024-05-01 00:45:00+00:00	8132.4
2024-05-01 01:00:00+00:00	8132.1
...	...
2024-11-28 17:00:00+00:00	8287.1
2024-11-28 17:15:00+00:00	8288.1
2024-11-28 17:30:00+00:00	8292.6
2024-11-28 17:45:00+00:00	8291.7
2024-11-28 18:00:00+00:00	8291.0

11903 rows × 1 columns

```
data['returns'] = np.log(data['price'] / data['price'].shift(1))
data
```

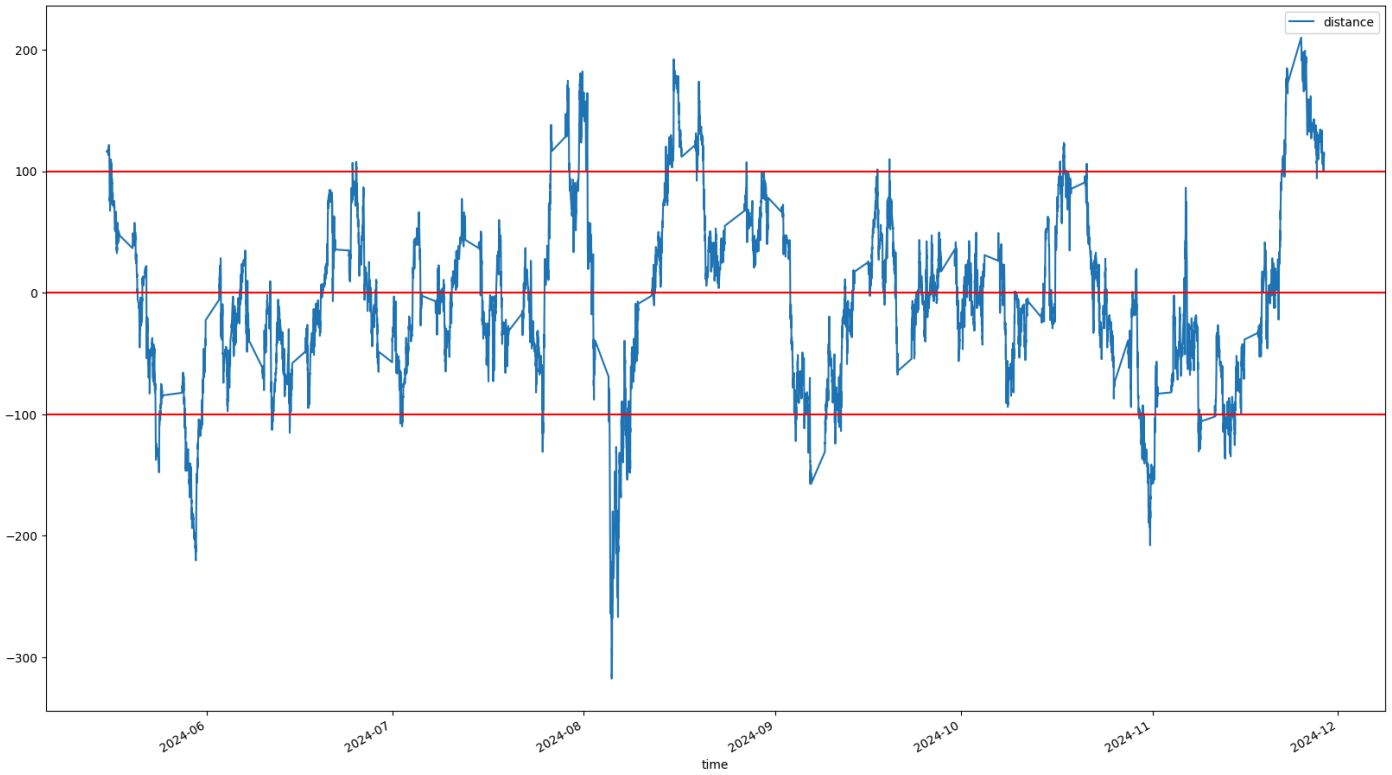
time	price	returns
2024-05-01 00:00:00+00:00	8128.1	NaN
2024-05-01 00:15:00+00:00	8128.7	0.000074
2024-05-01 00:30:00+00:00	8127.0	-0.000209
2024-05-01 00:45:00+00:00	8132.4	0.000664
2024-05-01 01:00:00+00:00	8132.1	-0.000037
...	...	...
2024-11-28 17:00:00+00:00	8287.1	-0.000302
2024-11-28 17:15:00+00:00	8288.1	0.000121
2024-11-28 17:30:00+00:00	8292.6	0.000543
2024-11-28 17:45:00+00:00	8291.7	-0.000109
2024-11-28 18:00:00+00:00	8291.0	-0.000084

11903 rows × 2 columns

5.3.2.3. Mean reversion parameters - SMA & Threshold

```
# Set SMA = 800
SMA = 800
data['SMA'] = data['price'].rolling(SMA).mean()
```

```
# Set threshold = 100
threshold = 100

data['distance'] = data['price'] - data['SMA']

data['distance'].dropna().plot(figsize=(20, 12), legend=True)

plt.axhline(threshold, color='r')
plt.axhline(-threshold, color='r')
plt.axhline(0, color='r');
```



- For the data UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ), here shows the place where positions will be opened and closed, if setting SMA = 800 and threshold = 100.

```
data
```

time	price	returns	SMA	distance
2024-05-01 00:00:00+00:00	8128.1	NaN	NaN	NaN
2024-05-01 00:15:00+00:00	8128.7	0.000074	NaN	NaN
2024-05-01 00:30:00+00:00	8127.0	-0.000209	NaN	NaN
2024-05-01 00:45:00+00:00	8132.4	0.000664	NaN	NaN
2024-05-01 01:00:00+00:00	8132.1	-0.000037	NaN	NaN
...	...	...	...	...
2024-11-28 17:00:00+00:00	8287.1	-0.000302	8176.615375	110.484625
2024-11-28 17:15:00+00:00	8288.1	0.000121	8176.915750	111.184250
2024-11-28 17:30:00+00:00	8292.6	0.000543	8177.213625	115.386375
2024-11-28 17:45:00+00:00	8291.7	-0.000109	8177.507250	114.192750
2024-11-28 18:00:00+00:00	8291.0	-0.000084	8177.795625	113.204375

11903 rows × 4 columns

5.3.2.4. Market positioning - Mean Reversion strategy

```
# short position:
# otherwise NaN
# NaN = keep current position opened unchanged, (except for first 25 values (SMA = 25) default to be NaN)
data['position'] = np.where(data['distance'] > threshold,
                                    -1, np.nan)
```

```
# long position:
# "1, data['position'])" = otherwise, -1 or NaN
# because “otherwise NaN” is set already in above line

data['position'] = np.where(data['distance'] < -threshold,
                                   1, data['position'])
```

```
# if sign of distance value (price - SMA) change,
# go market neutral (closing existing position), i.e. position = 0

# use "data['distance'] * data['distance'].shift(1)" to detect change of sign

    # eg. +ve * +ve = +ve , no need position = 0
    # eg. -ve * -ve = +ve , no need position = 0

    # eg. +ve * -ve = -ve < 0 , need position = 0
    # eg. -ve * +ve = -ve < 0 , need position = 0

# otherwise, remains unchanged

data['position'] = np.where(data['distance'] * data['distance'].shift(1) < 0, 0, data['position'])
```

- Position opening conditions, comparing ‘distance’ with ‘threshold’.

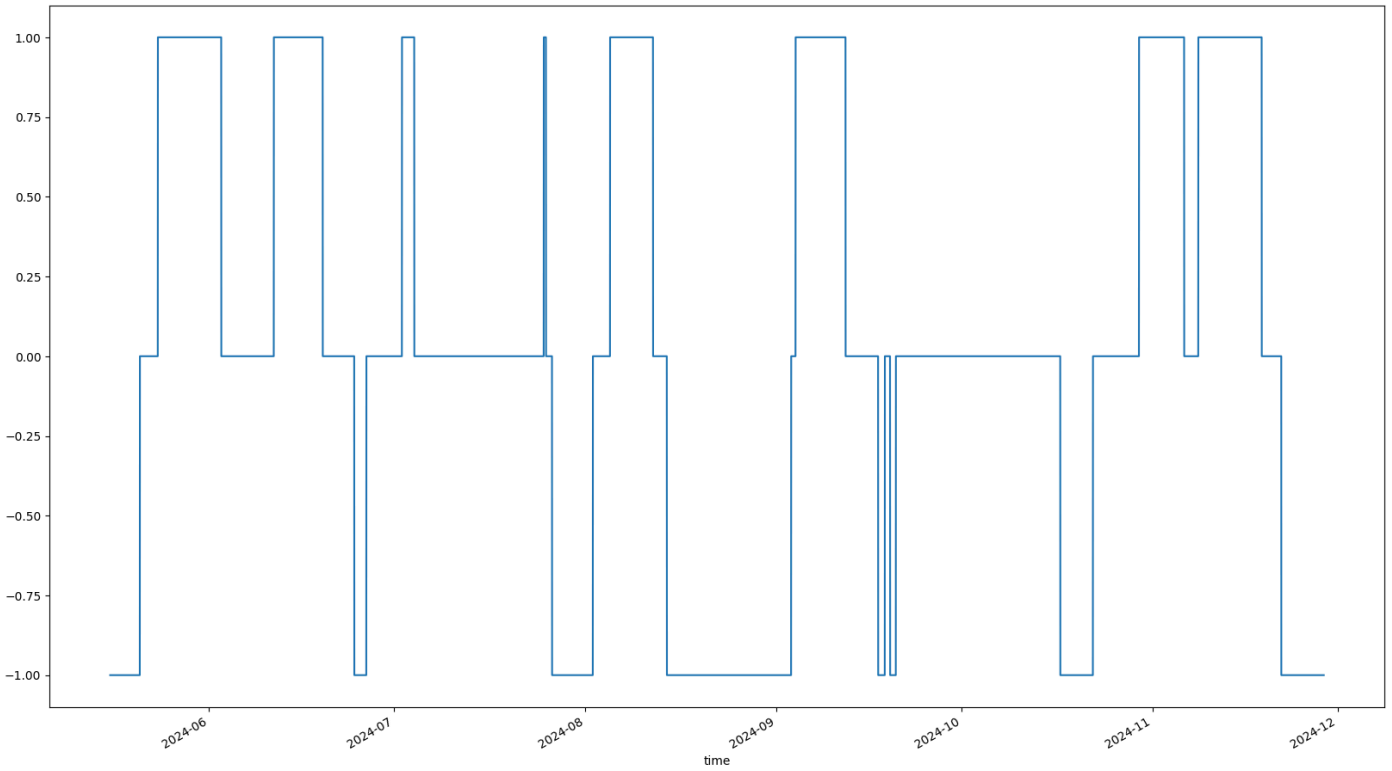
```
# .fillna(0) is to make first n values (if SMA = n) become 0, we dont open position before SMA is established
# as forward filling .ffill() deal with first n values and make it 0

# eg.

# Before ffill() and fillna(0):
#    position
# 0       NaN
# 1      -1.0
# 2       NaN
# 3       NaN
# 4       1.0
# 5       NaN

# After ffill():
#    position
# 0       NaN      # No previous value to fill
# 1      -1.0
# 2      -1.0      # Filled with value from index 1
# 3      -1.0      # Filled with value from index 1
# 4       1.0
# 5       1.0      # Filled with value from index 4

# After ffill().fillna(0):
#    position
# 0       0.0      # Remaining NaN filled with 0
# 1      -1.0
# 2      -1.0
# 3      -1.0
# 4       1.0
# 5       1.0

data['position'] = data['position'].ffill().fillna(0)
```

```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(data)
```

time	price	returns	SMA	distance	position
2024-05-01 00:00:00+00:00	8128.1	NaN	NaN	NaN	0.0
2024-05-01 00:15:00+00:00	8128.7	0.000074	NaN	NaN	0.0
2024-05-01 00:30:00+00:00	8127.0	-0.000209	NaN	NaN	0.0
2024-05-01 00:45:00+00:00	8132.4	0.000664	NaN	NaN	0.0
2024-05-01 01:00:00+00:00	8132.1	-0.000037	NaN	NaN	0.0
2024-05-01 01:15:00+00:00	8134.7	0.000320	NaN	NaN	0.0
2024-05-01 01:30:00+00:00	8131.6	-0.000381	NaN	NaN	0.0
⋮					
2024-05-15 19:15:00+00:00	8441.0	0.000604	NaN	NaN	0.0
2024-05-15 19:30:00+00:00	8438.3	-0.000320	NaN	NaN	0.0
2024-05-15 19:45:00+00:00	8441.3	0.000355	8324.867125	116.432875	-1.0
2024-05-16 00:00:00+00:00	8442.1	0.000095	8325.259625	116.840375	-1.0
2024-05-16 00:15:00+00:00	8440.6	-0.000178	8325.649500	114.950500	-1.0
2024-05-16 00:30:00+00:00	8441.1	0.000059	8326.042125	115.057875	-1.0
⋮					
2024-05-31 19:45:00+00:00	8301.8	0.001422	8324.310375	-22.510375	1.0
2024-06-03 00:00:00+00:00	8318.5	0.002010	8324.170500	-5.670500	1.0
2024-06-03 00:15:00+00:00	8330.7	0.001466	8324.047375	6.652625	0.0
2024-06-03 00:30:00+00:00	8329.0	-0.000204	8323.925500	5.074500	0.0
2024-06-03 00:45:00+00:00	8331.5	0.000300	8323.799875	7.700125	0.0
⋮					
2024-06-11 12:15:00+00:00	8150.1	-0.000589	8244.266875	-94.166875	0.0
2024-06-11 12:30:00+00:00	8146.0	-0.000503	8244.096750	-98.096750	0.0
2024-06-11 12:45:00+00:00	8140.2	-0.000712	8243.929125	-103.729125	1.0
2024-06-11 13:00:00+00:00	8147.6	0.000909	8243.763250	-96.163250	1.0
2024-06-11 13:15:00+00:00	8143.6	-0.000491	8243.588750	-99.988750	1.0

- Here prints out full data of how positions are changed from:
    - 0 to -1 when ‘distance’ crosses +100, from NaN to 116.432875, passing the upper threshold.
    - 1 to 0, when ‘distance’ crosses 0 from -5.670500 to 6.652625, passing SMA.
    - 0 ot 1, when ‘distance’ crosses -100, from -98.096750 to -103.729125, passing the lower threshold.

```
# ylim: y-axis limits of the plot
# range is slightly extended beyond the possible position values (-1, 0, 1) to allow for better visualization

data['position'].iloc[SMA:].plot(ylim=[-1.1, 1.1],
                                 figsize=(20, 12));
```

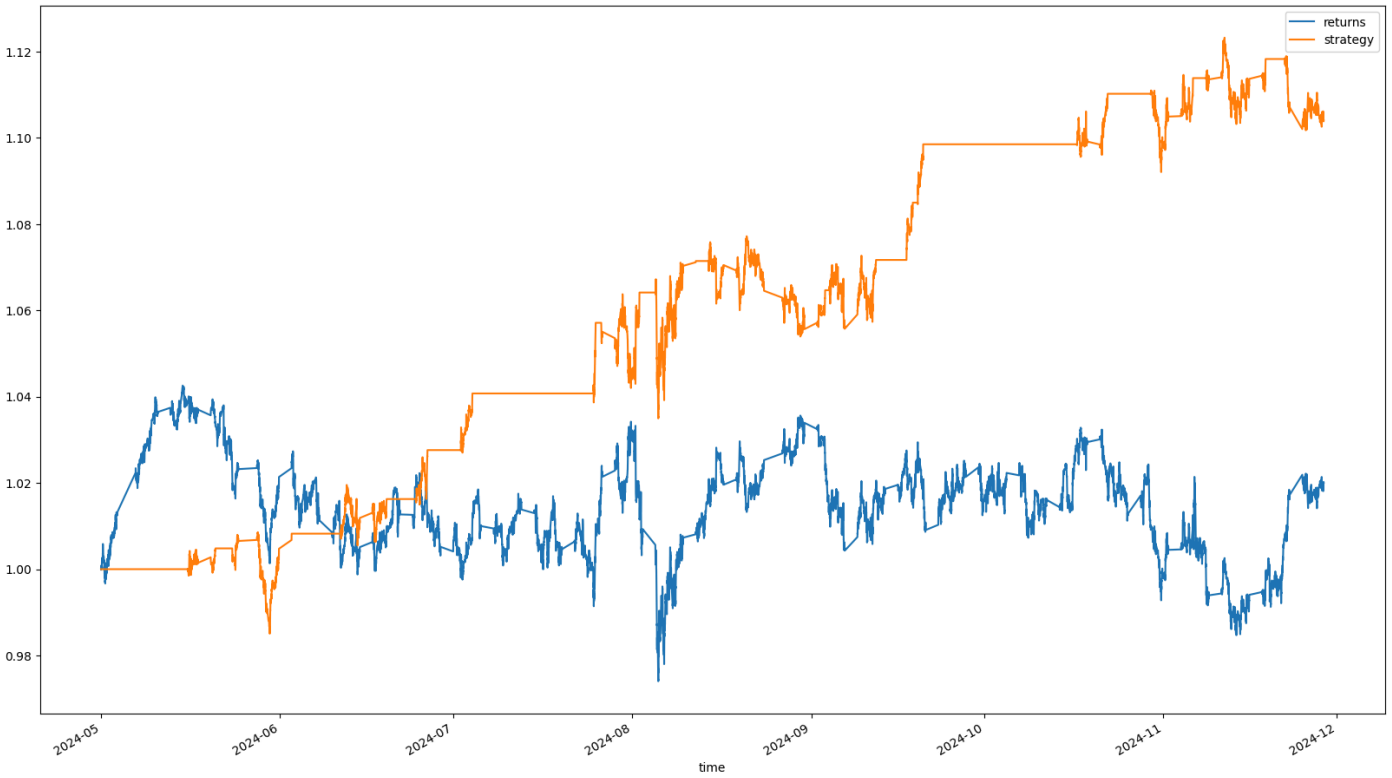


- Shows graph of market positions over time, including -1 (short), 0 (neutral) and 1 (long) throughout May 2024 to Nov 2024.

5.3.2.5. Benchmark comparison

```
data['strategy'] = data['position'].shift(1) * data['returns']
```

```
data[['returns', 'strategy']].dropna().cumsum(
                 ).apply(np.exp).plot(figsize=(20, 12));
```



- Mean reversion strategy starts outperform asset return starting from July 2024 in general and have overall outperformance towards asset normal return.

5.3.3. Parameter Optimization - SMA & Threshold

- For the parameter optimization part, Backtester.py & Backtester_CodeRunner.ipynb are used.
- Brute force is implemented to find the optimal parameter combinations SMA & Threshold, under the circumstances that transaction cost and initial capital are included.
- Heatmap is generated to show the optimization process.

5.3.3.1. Backtester - Mean_Reversion_Backtester.py

- Below is the Mean_Reversion_Backtester.py, with detailed explanations inside the code.
- Here, same with illustration code, the data source is also UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ).

```
# Mean_Reversion_Backtester

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt     # Visualization of optimizing parameters:
heatmap_visualisation
import seaborn as sns


class Mean_Reversion_Backtester(object):
   ''' A class designed for backtesting trading strategy based on
       Mean Reversion in a vectorized manner

   Attributes
   ==========
   asset: str
       symbol of asset
   start_time: str
       starting date for the backtesting period
   end_time: str
       ending date for the backtesting period
   capital: int, float
       capital to be invested at the beginning
   t_cost: float
       proportional transaction costs (e.g. 0.5% = 0.005) per trade

   Methods
   =======
   import_data:
```

```
        Loads and prepares the historical price data for backtesting
    execute_strategy:
        Executes trading strategy and evaluates its performance
    display_results:
        Creates a plot comparing the strategy's performance to that of the asset
parameters_optimization:
            Optimizes the mean reversion parameter by testing a range of values
            and identifying the one with the best performance
    parameters_optimization:
        Optimizes the mean reversion parameter combination (SMA, threshold)
        by testing a range of values and identifying the one with the best performance
    heatmap_visualisation:
        Visualizes the optimization results using a heatmap to highlight
        the best momentum parameter and its corresponding performance
    '''


    def __init__(self, asset, start_time, end_time, capital, t_cost):
        self.asset = asset
        self.start_time = start_time
        self.end_time = end_time
        self.capital = capital
        self.t_cost = t_cost
        self.results = None
        self.import_data()

    def import_data(self):
        # ❗ Need to adjust time:
        #  eg. mrbt = MR.Mean_Reversion_Backtester(
        #              'SPX', '2020-05-05 16:00:00', '2020-05-05 20:00:00', 10000, 0.001)
        ''' Loads and prepares the historical price data for backtesting
        '''

        # ⭐
        # illustration = UK100 (PEPPERSTONE) - 15min - (202405010000000ZZ - 2024112920450ZZ)
        # CodeRunner   = UK100 (PEPPERSTONE) - 15min - (202405010000000ZZ - 2024112920450ZZ)
        data_imported = "SPX (SP) - 1D - (1928-01-03 - 2024-11-29)"
        fn = f'/Users/kctang/Desktop/Algo Trading Project
(S1)/Strategies/Data/{data_imported}.csv'
        raw_data = pd.read_csv(fn, index_col=0, parse_dates=True).dropna()
        print(raw_data)

        raw_data = pd.DataFrame(raw_data['close'])
        # ❗ Need to imput correct starting time and ending time!
        raw_data = raw_data.loc[self.start_time:self.end_time]
        raw_data.rename(columns={'close': 'price'}, inplace=True)

        raw_data['return'] = np.log(raw_data / raw_data.shift(1))

        self.data = raw_data
```

```
   def execute_strategy(self, SMA, threshold):
       ''' Executes trading strategy and evaluates its performance
       '''
       data = self.data.copy().dropna()
       data['sma'] = data['price'].rolling(SMA).mean()
       data['distance'] = data['price'] - data['sma']
       data.dropna(inplace=True)

       # Short signals
       data['position'] = np.where(data['distance'] > threshold,
                                   -1, np.nan)
       # Long signals
       data['position'] = np.where(data['distance'] < -threshold,
                                   1, data['position'])

       # Crossing of current price and SMA (i.e. zero distance)
       data['position'] = np.where(data['distance'] * data['distance'].shift(1) < 0, 0,
data['position'])
       data['position'] = data['position'].ffill().fillna(0)
       data['strategy'] = data['position'].shift(1) * data['return']

       # Decide when to open position (and close position if flipping long/short side)
       trades = data['position'].diff().fillna(0) != 0

       # Subtract transaction costs from return when trade takes place
       data.loc[trades, 'strategy'] -= self.t_cost

       data['cum_returns'] = self.capital * data['return'].cumsum().apply(np.exp)
       data['cum_strategy'] = self.capital * data['strategy'].cumsum().apply(np.exp)

       self.results = data

       # Total performance of the strategy
       total_perf = self.results['cum_strategy'].iloc[-1]

       # Out/under-performance of strategy
       out_perf = total_perf - self.results['cum_returns'].iloc[-1]

       return round(total_perf, 2), round(out_perf, 2)


   def display_results(self):
       ''' Creates a plot comparing the strategy's performance to that of the asset
parameters_optimization:
               Optimizes the mean reversion parameter by testing a range of values
               and identifying the one with the best performance
       '''
       if self.results is None:
```

```
            print('No results to plot yet. Run a strategy.')
            return

        plt.figure(figsize=(20, 12))  # Large figure size for Jupyter Notebook
        title = '%s | t_cost = %.4f' % (self.asset, self.t_cost)
        self.results[['cum_returns', 'cum_strategy']].plot(title=title, figsize=(20, 12),
fontsize=14)
        plt.title(title, fontsize=18)
        plt.xlabel('Date', fontsize=14)
        plt.ylabel('Cumulative Performance', fontsize=14)

        plt.show()


    def parameters_optimization(self, SMA_range, threshold_range):
        ''' Optimizes the mean reversion parameter combination (SMA, threshold)
        by testing a range of values and identifying the one with the best performance

        Parameters
        ==========
        SMA_range: tuple
            Tuple of integers for SMA range (start_time, stop, step).
        threshold_range: tuple
            Tuple of floats for threshold range (start_time, stop, step).

        Returns
        =======
        results: pd.DataFrame
            DataFrame containing the performance for all combinations.
        '''
        SMAs = range(*SMA_range)
        thresholds = np.arange(*threshold_range)
        results = []

        for SMA in SMAs:
            for threshold in thresholds:
                total_perf, out_perf = self.execute_strategy(SMA, threshold)
                cum_returns = self.results['cum_returns'].iloc[-1]  # Get the benchmark
cumulative returns
                results.append((SMA, threshold, total_perf, out_perf, cum_returns))

        # Create a DataFrame with the results
        results_df = pd.DataFrame(results, columns=['SMA', 'Threshold', 'total_perf', 'out_perf',
'cum_returns'])

        self.optimization_results = results_df

        # Find the best combination based on absolute performance (total_perf)
        best_result = results_df.loc[results_df['total_perf'].idxmax()]
        best_SMA = best_result['SMA']
```

```
        best_threshold = best_result['Threshold']
        best_aperf = best_result['total_perf']
        best_operf = best_result['out_perf']
        best_creturns = best_result['cum_returns']

        # Print the best parameters and their performance
        print(f"\nProduct: [{self.asset}]\n")
        print(f"    > Best Parameters:\n")
        print(f"        > SMA: {best_SMA} | Threshold: {best_threshold}\n")
        print(f"        > Strategy's Absolute Performance: {best_aperf:.2f}")
        print(f"        > Strategy's Outperformance: {best_operf:.2f}")
        print(f"        > Benchmark's Cumulative Returns: {best_creturns:.2f}\n\n")

        return results_df


    def heatmap_visualisation(self, metric='total_perf'):
        ''' Visualizes the optimization results using a heatmap to highlight
        the best momentum parameter and its corresponding performance

        Parameters
        ==========
        metric: str
            'total_perf' for absolute performance or 'out_perf' for outperformance.
        '''
        if self.optimization_results is None:
            print('No optimization results to plot yet. Run parameters_optimization().')
            return

        print(f"Plotting heatmap for metric: {metric}")  # Debugging statement

        # Pivot the DataFrame for heatmap plotting
        pivot = self.optimization_results.pivot(index='Threshold', columns='SMA', values=metric)

        plt.figure(figsize=(20, 12))  # Large figure size for Jupyter Notebook
        sns.heatmap(pivot, annot=True, fmt='.2f', cmap='RdYlGn', cbar=True)
        plt.title(f'Optimization Results ({metric})', fontsize=18)
        plt.xlabel('SMA', fontsize=14)
        plt.ylabel('Threshold', fontsize=14)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)


        plt.show()
```

5.3.3.2. Backtester_CodeRunner - Mean_Reversion_CodeRunner - UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ)

```
import Mean_Reversion_Backtester as MR
```

```
# ⭐ Initialize the backtester
MR_BT = MR.Mean_Reversion_Backtester('UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ)',
                                     '2024-05-01T00:00:00Z', '2024-11-29T20:45:00Z', 10000, 0.001)
```

```
                           open    high     low   close           MA  Volume
time
2024-05-01 00:00:00+00:00  8128.7  8134.0  8125.8  8128.1  8164.353191     690
2024-05-01 00:15:00+00:00  8128.2  8131.7  8125.3  8128.7  8163.442553     536
2024-05-01 00:30:00+00:00  8128.8  8130.0  8126.2  8127.0  8162.523404     286
2024-05-01 00:45:00+00:00  8126.9  8132.7  8124.3  8132.4  8161.538298     344
2024-05-01 01:00:00+00:00  8132.6  8134.9  8130.6  8132.1  8160.546809     380
...                           ...     ...     ...     ...          ...     ...
2024-11-28 17:00:00+00:00  8289.1  8290.1  8287.1  8287.1  8285.661702     272
2024-11-28 17:15:00+00:00  8286.8  8288.6  8286.1  8288.1  8285.627660     192
2024-11-28 17:30:00+00:00  8287.8  8292.6  8287.1  8292.6  8285.661702     172
2024-11-28 17:45:00+00:00  8293.1  8293.8  8290.8  8291.7  8285.638298     272
2024-11-28 18:00:00+00:00  8291.8  8291.8  8290.6  8291.0  8285.623404      72

[11903 rows x 6 columns]
```

5.3.3.2.1. Heatmap - visualization of optimization process - Mean Reversion

```
# ⭐ Optimize parameters:
    # Test SMA values from:
    # eg. 1 to 1000 (step 5) and thresholds from 1 to 500 (step 1)

# ❗ SMA_range(>= 1, ...) | 1st value must be >= 1
optimization_results = MR_BT.parameters_optimization(SMA_range=(760, 770, 1), threshold_range=(70, 110, 1))
```

```
Product: [UK100 (PEPPERSTONE) - 15min - (20240501000000ZZ - 20241129204500ZZ)]

    > Best Parameters:

        > SMA: 765.0 | Threshold: 89.0

        > Strategy's Absolute Performance: 11066.73
        > Strategy's Outperformance: 1260.24
        > Benchmark's Cumulative Returns: 9806.50
```

```
# Display the optimization results
print(optimization_results)
```

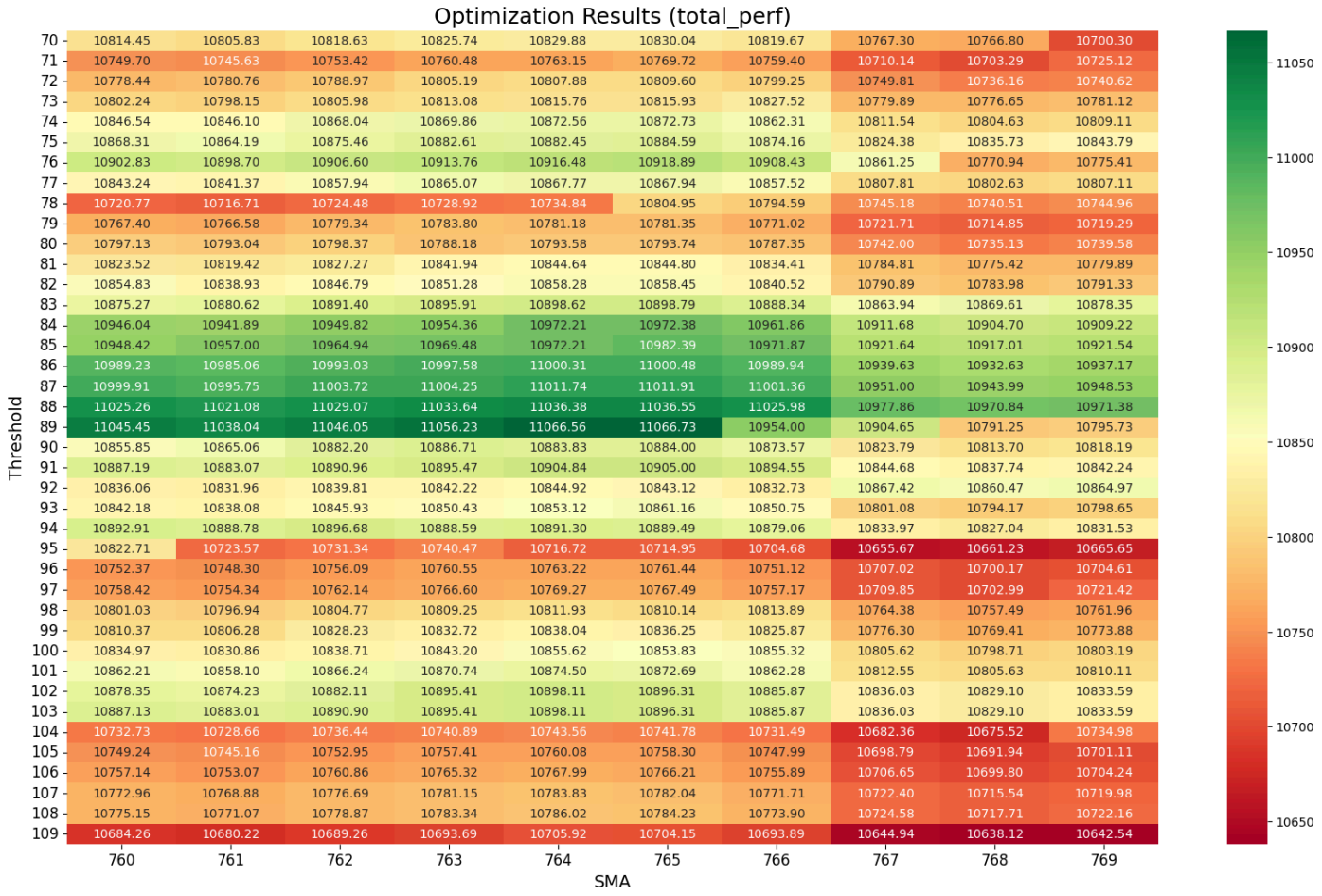
```
     SMA  Threshold  total_perf  out_perf  cum_returns
0    760         70    10814.45    997.15  9817.294827
1    760         71    10749.70    932.40  9817.294827
2    760         72    10778.44    961.14  9817.294827
3    760         73    10802.24    984.95  9817.294827
4    760         74    10846.54   1029.24  9817.294827
..   ...        ...         ...       ...          ...
395  769        105    10701.11    870.31  9830.797870
396  769        106    10704.24    873.44  9830.797870
397  769        107    10719.98    889.19  9830.797870
398  769        108    10722.16    891.36  9830.797870
399  769        109    10642.54    811.74  9830.797870

[400 rows x 5 columns]
```

- Here, SMA_range=(760, 770, 1) & threshold_range=(70, 110, 1) is the final range that is narrowed down to generate a nice-looking heatmap.
- Initially, the range will be much larger and is set to be SMA_range=(1, 1000, 5), threshold_range=(1, 500, 1), with SMA_range step = 5 for each iteration, which helps reduce computational cost and

memory. Then, SMA_range and threshold_range are narrowed down to get the final range SMA_range=(760, 770, 1) & threshold_range=(70, 110, 1) eventually.

```
# Calculate strategy performance in different combinations of parameters

# Plot heatmap of Total Performance (total_perf)
    # After mean-reversion strategy is applied, final portfolio value
    # Calculated as the value of the cumulative performance of the strategy (cstrategy) at the last date in the dataset
    # aperf = self.results['cstrategy'].iloc[-1]

MR_BT.heatmap_visualisation(metric='total_perf')
```

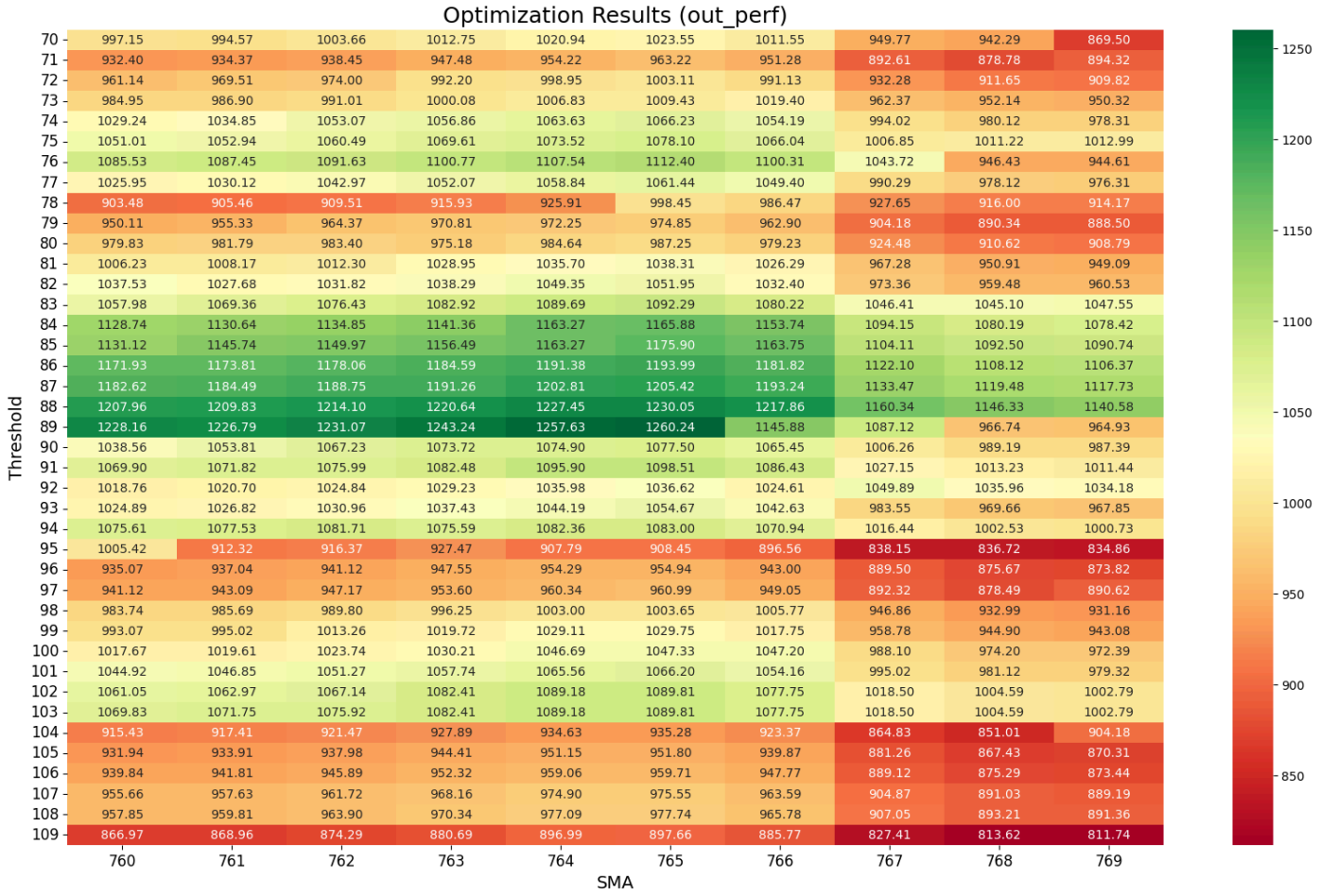
```
Plotting heatmap for metric: total_perf
```



- Total performance heatmap of UK100 in 15 min time frame throughout May 2024 to Nov 2024.
- Here, red area does not mean losing money, it just means comparatively less profitable than the best parameters, since even the darkest red is above 10000 USD initial capital, as shown on the right vertical status bar.

```
# Compare with benchmark

# Plot heatmap of Outperformance (out_perf)
    # Operf = difference between the performance of the strategy and
    #         the performance of simply holding the asset (the benchmark)
    # operf = aperf - self.results['creturns'].iloc[-1]

MR_BT.heatmap_visualisation(metric='out_perf')
```

```
Plotting heatmap for metric: out_perf
```



- Outperformance heatmap of UK100 in 15 min time frame throughout May 2024 to Nov 2024, having exactly the same color as Total performance

5.3.3.2.2. Optimal scenario performance - Mean Reversion

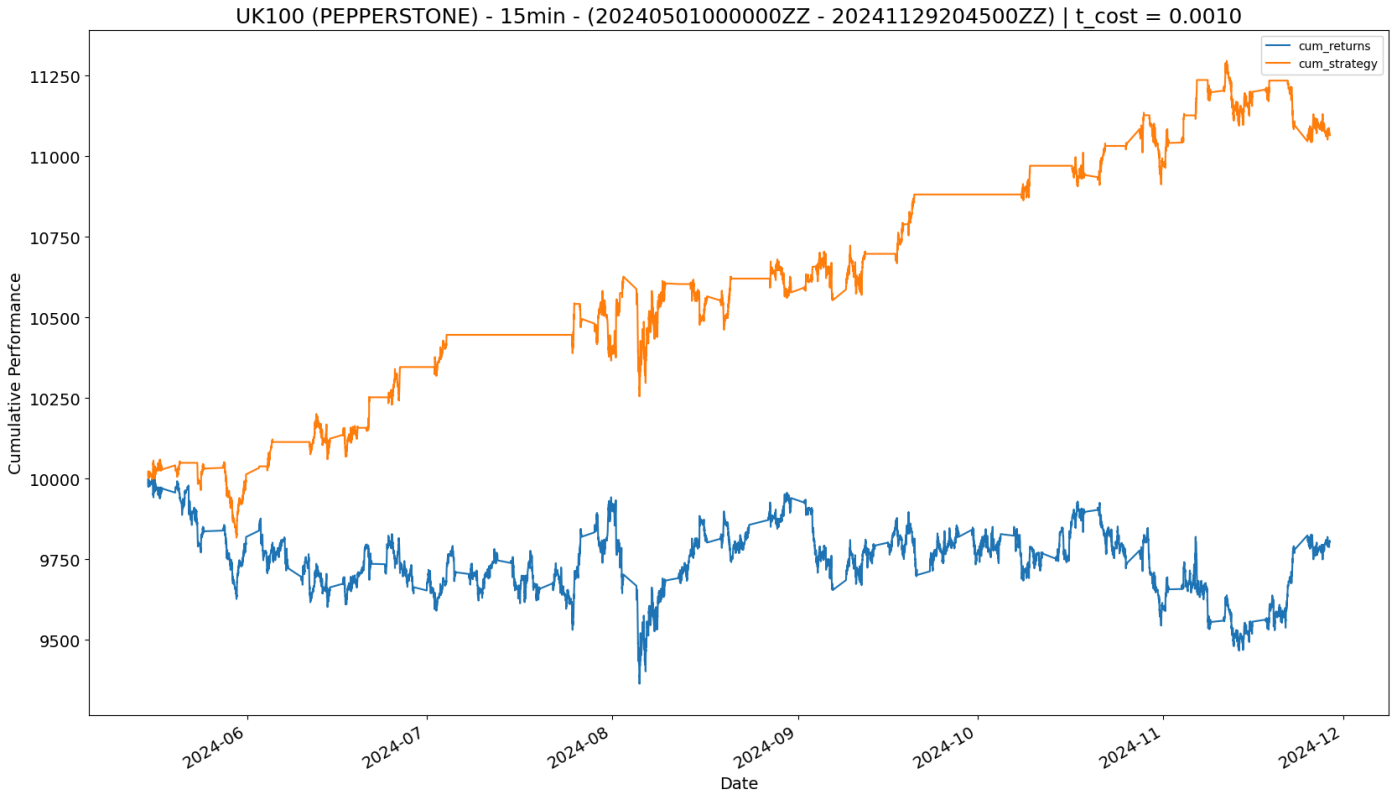
```
MR_BT.execute_strategy(765, 89)
```

```
(np.float64(11066.73), np.float64(1260.24))
```

```
MR_BT.display_results()
```

```
<Figure size 2000x1200 with 0 Axes>
```

- When in optimal parameter combinations (SMA = 765 & Threshold = 89), total performance is 11066.73, with outperformance 1260.24, showing profitability.



- Overall, mean reversion strategy implemented on UK100 outperforms asset normal return and is profitable, in 15 min time frame throughout May 2024 to Nov 2024.

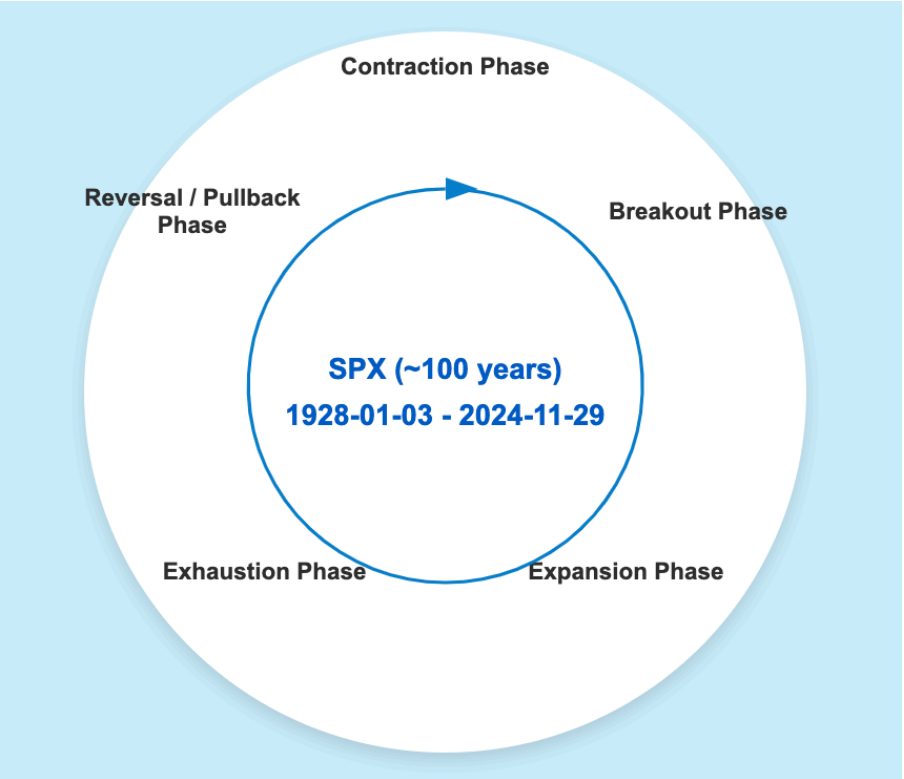
6. Scenario Analysis

Here, in the scenario analysis part, the goal is to find which strategy is the most profitable in each scenario and its effectiveness. In every scenario, each strategy (SMA, Momentum, Mean Reversion) will be executed, mainly data of the S&P 500 index, which is the most significant and representative asset in the world. And the optimal parameters of each strategy in each scenario will be used to compare effectiveness. This part mainly compares Momentum and Mean Reversion strategy, with transaction cost = 0.1%, while SMA will have 0 transaction cost.

6.1 Full historical timespan of S&P 500 Index - (1928-01-03 - 2024-11-29)

- Data: SPX (SP) - 1D - (1928-01-03 - 2024-11-29)
- Time range: 96 years, 10 months, and 26 days (or 35,395 days)

In this part, the full historical timespan of S&P 500 Index from 1928-01-03 to 2024-11-29 is adopted. On TradingView, this is the earlier date that has Daily data, dates prior to 1928-01-03 are all monthly data. The official launching date of S&P 500 is March 4, 1957. Data prior to 1957 are reconstructed data projected by analysts using S&P 90, an index of 90 stocks started in 1926. Since the year 1928 is a key time period in the US just before the Great Depression and the 1929 stock market crash, including date range starting from 1928 allows more comprehensive study of the research. This also helps evaluate strategies overall ultra long-term performance and adaptability since it has gone through the circular economic phrase below:



6.1.1 SMA

- Code: SMA_Backtester_CodeRunner_TP_SPX (SP) - 1D - (1928-01-03 - 2024-11-29).ipynb

```
# SMA_Backtester_CodeRunner
```

```
import SMA_Backtester as SMA
```

```
# Run strategy

from SMA_Backtester import SMA_Backtester

SMA_BT = SMA_Backtester('SPX (SP) - 1D - (1928-01-03 - 2024-11-29)', 42, 252, '1928-01-03', '2024-11-29')

SMA_BT.execute_strategy()
```

```
                  open         high          low        close      Volume
time
1928-01-03    17.760000    17.760000    17.760000    17.760000           0
1928-01-04    17.719999    17.719999    17.719999    17.719999           0
1928-01-05    17.549999    17.549999    17.549999    17.549999           0
1928-01-06    17.660000    17.660000    17.660000    17.660000           0
1928-01-09    17.500000    17.500000    17.500000    17.500000           0
...                 ...          ...          ...          ...         ...
2024-11-22  5944.360000  5972.900000  5944.360000  5969.330000  2510000000
2024-11-25  5992.280000  6020.750000  5963.910000  5987.380000  3643000000
2024-11-26  6001.160000  6025.420000  5992.270000  6021.640000  2319000000
2024-11-27  6014.110000  6020.160000  5984.870000  5998.730000  2024000000
2024-11-29  6003.980000  6044.170000  6003.980000  6032.390000  1382000000

[24378 rows x 5 columns]
(np.float64(356.09), np.float64(113.43))
```

- Classic parameter combination SMA_1 = 42 & SMA_2 = 252 gives total performance of 356.09 and outperformance of 113.43 with initial capital base 1.00 throughout 96 years time horizon.

```
# Optimizing parameters
optimal_params, optimal_perf = SMA_BT.parameters_optimization((1, 1000, 5), (1, 1000, 5))
```

```
Evaluating SMA_1=971, SMA_2=976: Performance=14.22
Evaluating SMA_1=971, SMA_2=981: Performance=17.9
Evaluating SMA_1=971, SMA_2=986: Performance=11.66
Evaluating SMA_1=971, SMA_2=991: Performance=22.38
Evaluating SMA_1=971, SMA_2=996: Performance=21.03
Evaluating SMA_1=976, SMA_2=981: Performance=17.92
Evaluating SMA_1=976, SMA_2=986: Performance=13.58
Evaluating SMA_1=976, SMA_2=991: Performance=22.27
Evaluating SMA_1=976, SMA_2=996: Performance=20.03
Evaluating SMA_1=981, SMA_2=986: Performance=18.45
Evaluating SMA_1=981, SMA_2=991: Performance=15.84
Evaluating SMA_1=981, SMA_2=996: Performance=13.17
Evaluating SMA_1=986, SMA_2=991: Performance=8.88
Evaluating SMA_1=986, SMA_2=996: Performance=9.81
Evaluating SMA_1=991, SMA_2=996: Performance=14.64
Optimization Completed
Evaluated Parameters: [1. 6.]
Evaluating SMA_1=1.0, SMA_2=6.0: Performance=16208.21
```

- Broad search (1, 1000, 5), (1, 1000, 5) gives optimized parameters SMA_1 = 1 & SMA_2 = 6. 1 and 6 are very near values considering the large number of rows of 24378 data.

```
# Optimizing parameters
optimal_params, optimal_perf = SMA_BT.parameters_optimization((1, 10, 1), (1, 15, 1))
```

```
Evaluating SMA_1=7, SMA_2=11: Performance=0.76
Evaluating SMA_1=7, SMA_2=12: Performance=1.14
Evaluating SMA_1=7, SMA_2=13: Performance=1.4
Evaluating SMA_1=7, SMA_2=14: Performance=1.25
Evaluating SMA_1=8, SMA_2=9: Performance=0.72
Evaluating SMA_1=8, SMA_2=10: Performance=0.67
Evaluating SMA_1=8, SMA_2=11: Performance=1.86
Evaluating SMA_1=8, SMA_2=12: Performance=2.04
Evaluating SMA_1=8, SMA_2=13: Performance=4.59
Evaluating SMA_1=8, SMA_2=14: Performance=2.32
Evaluating SMA_1=9, SMA_2=10: Performance=2.67
Evaluating SMA_1=9, SMA_2=11: Performance=1.66
Evaluating SMA_1=9, SMA_2=12: Performance=3.37
Evaluating SMA_1=9, SMA_2=13: Performance=4.78
Evaluating SMA_1=9, SMA_2=14: Performance=4.77
Optimization Completed
Evaluated Parameters: [1. 3.]
Evaluating SMA_1=1.0, SMA_2=3.0: Performance=56716.52
```

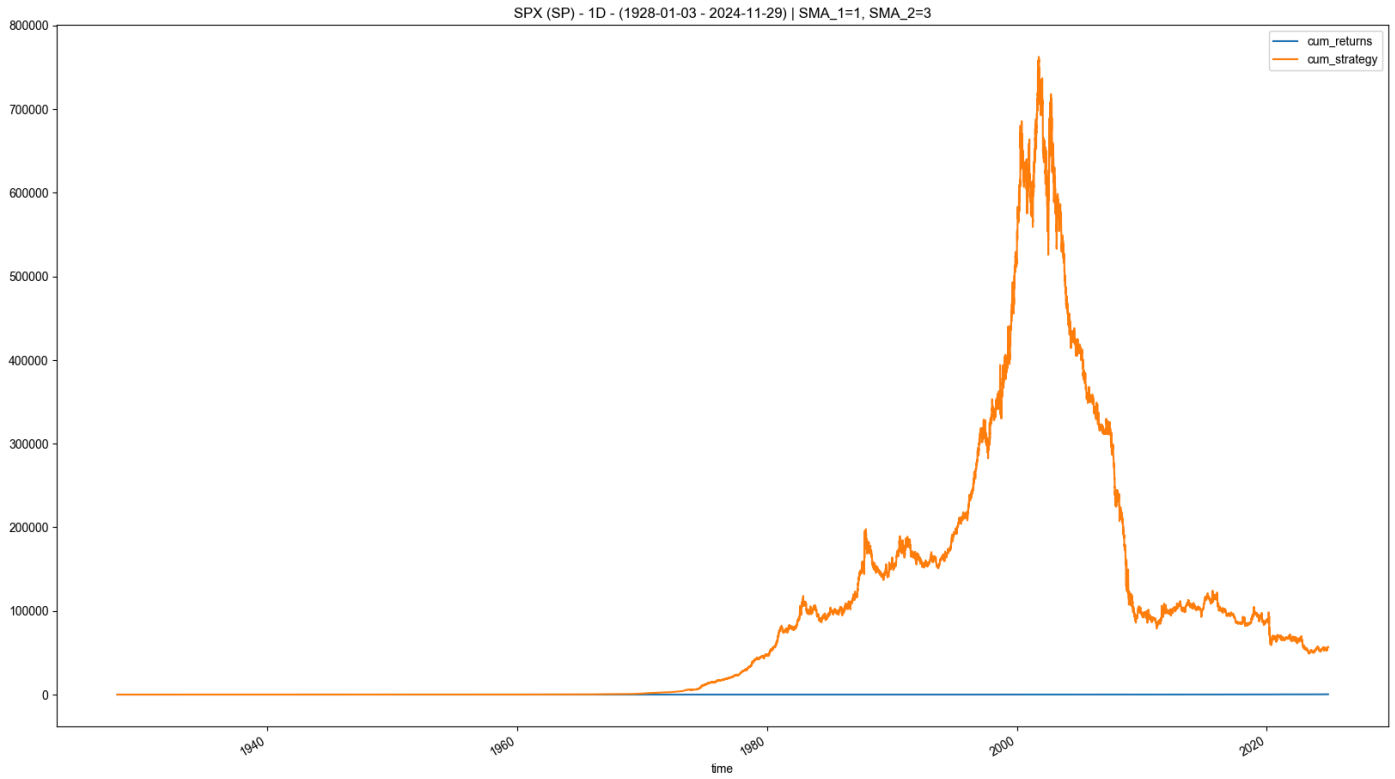
- Focused search (1, 10, 1), (1, 15, 1) gives optimized parameters SMA_1 = 1 & SMA_2 = 3, in which the value is even closer.
- This indicates although the time length of data is very long, approximately 96 years, a very small and close parameter combination of 1 and 3 yields the highest return, having total performance of 56716.52 (+ 5671652 %) (excluding transaction cost).

```
# Optimized scenario:
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(1, 3)
print("Total Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
print("Out-Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[1])
```

```
Total Performance for optimized parameters (best parameters): 56716.52
Out-Performance for optimized parameters (best parameters): 56372.8
```

```
# Plot result - optimized case
SMA_BT.display_results()
```



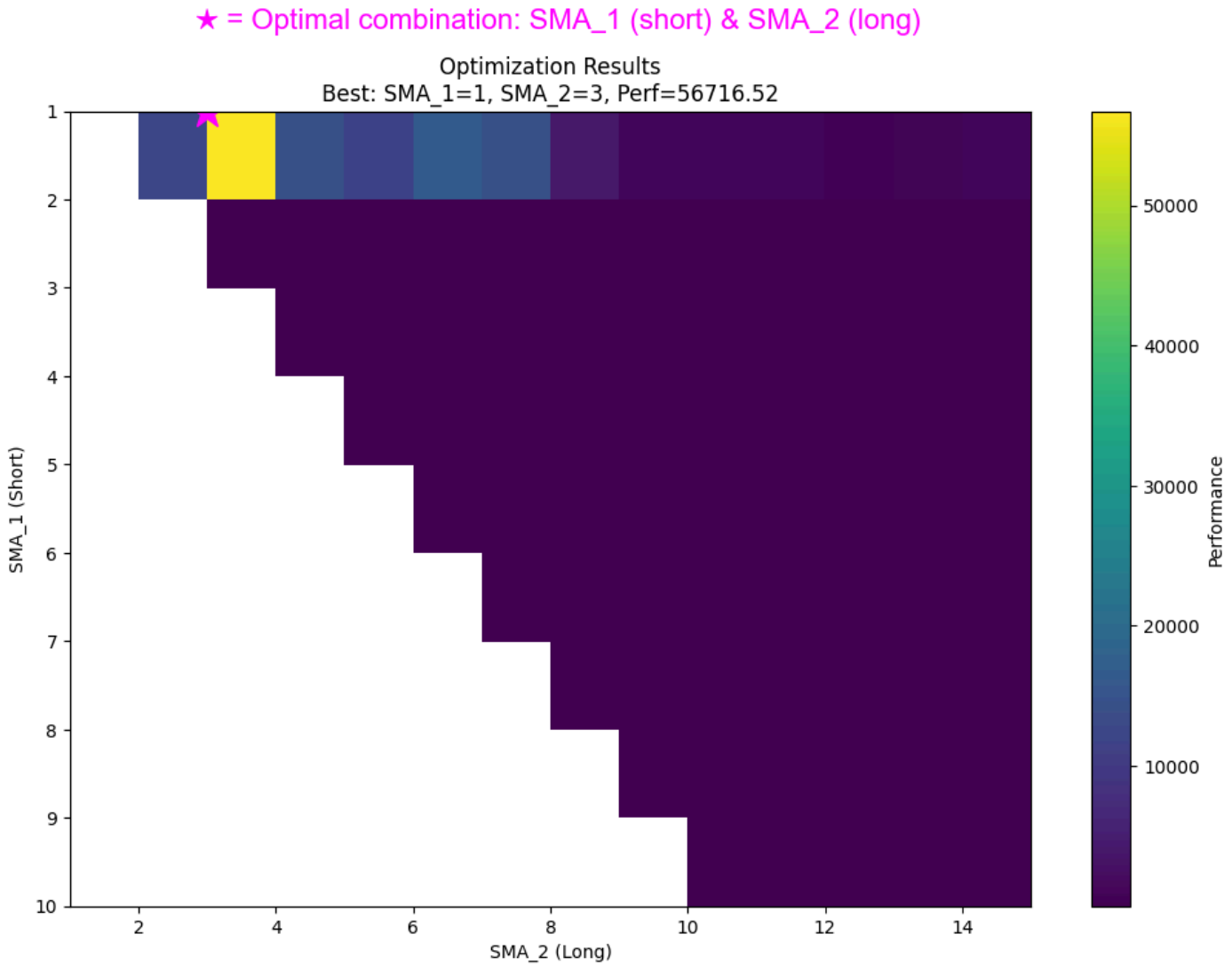
- Excluding transaction cost, total performance is 56716.52 (+ 5,671,652 %), with outperformance 56372.8 (+ 5,637,280 %) throughout 96 years, which is a very substantial return, demonstrating ultra strong profitability.
- Here, starting from 1970, SMA strategy increases exponentially, with the area starting from 1990 having the largest slope till the area around 2005. This indicates SMA strategy with parameter 1 and 3 is the most profitable during the area around 1970 - 2005.
- Potential reason is because the overall market conditions are favorable for trend-following strategies like SMA. From the 1970s to the early 2000s, global markets experienced several long-term upward trends, particularly in equities and commodities, including commodities boom in 1970s (oil crisis, gold bull market) (Hamilton, 1983), U.S. equity bull market (Campbell & Shiller, 1988) and Japanese asset bubble in 1980s (Hoshi & Kashyap, 1999), Dot-com boom (Ofek & Richardson, 2003b) and the emerging markets growth of China and India in 1990s (Rodrik & Subramanian, 2004) and Post-dot-com recovery, commodity supercycle in 2000s (Erten & Ocampo, 2013).

```
# Result - Optimized parameters
print("Optimal Parameters:", optimal_params)
print("Optimal Performance:", optimal_perf)
```

```
Optimal Parameters: [1. 3.]
Optimal Performance: 56716.52
```

```
# Heatmap
```

```
SMA_BT.heatmap_visualisation((1, 10, 1), (1, 15, 1))
```



- Here, since SMA_2 needs to be larger than SMA_1, a blank space exists if the condition SMA_2 > SMA_1 is not satisfied.
- The difference between optimal parameter combination 1 & 3 and its surrounding areas is very large, from ultra profitable to even losing money (lose even without transaction cost).
- This indicates that this optimal parameter combination 1 & 3 is extremely sensitive to even slightest changes of values.

```
# Double check heatmap correctness - by substituting values
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(1, 3)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 56716.52
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(1, 2)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 12663.62
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(1, 4)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 14788.65
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(2, 3)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 9.81
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(3, 5)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 11.77
```

```
# Performance - optimized parameters (best parameters)
SMA_BT.set_parameters(5, 10)
print("Performance for optimized parameters (best parameters):", SMA_BT.execute_strategy()[0])
```

```
Performance for optimized parameters (best parameters): 0.13
```

- To further check for the extreme sensitivity and fragility to value changes from the heatmap, check parameter combinations surrounding 1 & 3, which gives total performance of 56,716.52.
- Parameter combination of 1 & 2 and 1 & 4 gives total performance of 12663.62 and 14788.65 respectively, showing substantial decrease of -77.67% and -73.93% respectively even if the SMA_2 increment is as minimal as +1 or -1.
- While for parameter combinations of 2 & 3, 3 & 5, 5 & 10, it gives total performance of 9.81, 11.77 and 0.13 respectively, which are extremely far away from 56,716.52, highlighting extreme sensitivity of SMA strategy in this scenario.

6.1.2 Momentum

- Code: Momentum_Backtester_CodeRunner_TP_SPX (SP) - 1D - (1928-01-03 - 2024-11-29).ipynb

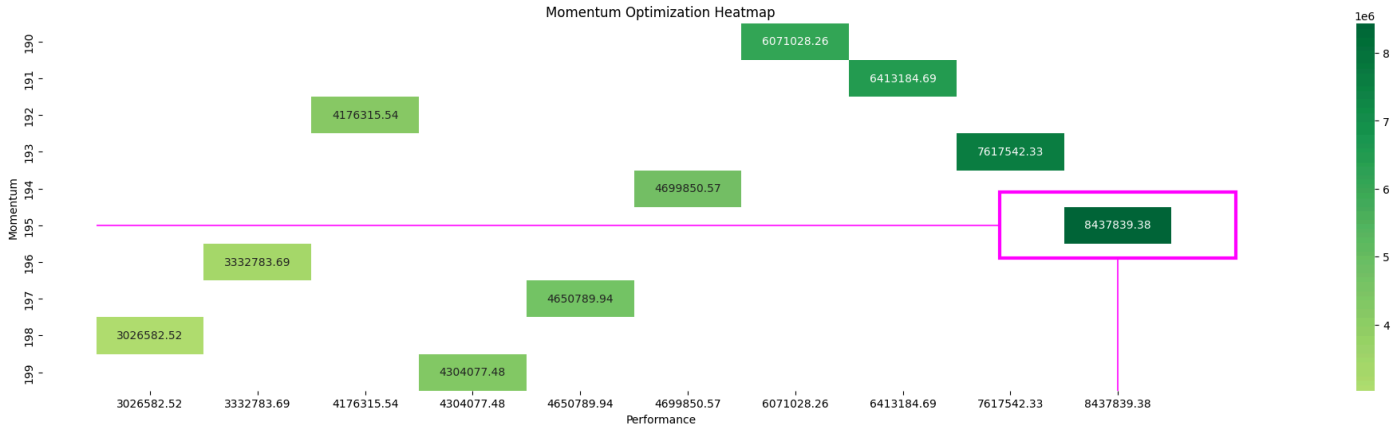
```
# Momentum_Backtester_CodeRunner
```

```
# Optimization of parameters: momentum (average of last n returns)
# Visualization of optimization process
```

```
# Momentum = 1-2000
# i.e. when momentum = 2000, use Average(last 2000 returns) to compare and make trading decision
# Screen nearly all possible momentum values to find most optimal value of "Momentum"

# Import the backtesting module
import Momentum_Backtester as Mom

# Initialize the backtester object
# MOM_BT = Mom.Momentum_Backtester('XAU=', '2010-1-1', '2019-12-31', 10000, 0.001)
MOM_BT = Mom.Momentum_Backtester('SPX (SP) - 1D - (1928-01-03 - 2024-11-29)',
                                 '1928-01-03', '2024-11-29', 10000, 0.001)

# Define the range of momentum values to test
momentum_range = range(1, 2000)

# Run the optimization
optimization_results = MOM_BT.parameters_optimization(momentum_range)

# Visualize the optimization process with a heatmap
MOM_BT.heatmap_visualisation(optimization_results)
```

time	open	high	low	close	Volume
1928-01-03	17.760000	17.760000	17.760000	17.760000	0
1928-01-04	17.719999	17.719999	17.719999	17.719999	0
1928-01-05	17.549999	17.549999	17.549999	17.549999	0
1928-01-06	17.660000	17.660000	17.660000	17.660000	0
1928-01-09	17.500000	17.500000	17.500000	17.500000	0
...	...	...	...	...	...
2024-11-22	5944.360000	5972.900000	5944.360000	5969.330000	2510000000
2024-11-25	5992.280000	6020.750000	5963.910000	5987.380000	3643000000
2024-11-26	6001.160000	6025.420000	5992.270000	6021.640000	2319000000
2024-11-27	6014.110000	6020.160000	5984.870000	5998.730000	2024000000
2024-11-29	6003.980000	6044.170000	6003.980000	6032.390000	1382000000

[24378 rows x 5 columns]
Best Momentum: 195, Best Performance: 8437839.38



```python
# Zoom in optimal parameter result of Momentum

# Import the backtesting module
import Momentum_Backtester as Mom

# Initialize the backtester object
# illustration:
#   MOM_BT = Mom.Momentum_Backtester('XAUUSD (PEPPERSTONE) - 1D - (2010-01-01 - 2019-12-31)', '2010-1-1', '2019-12-31', 10000, 0.001)
MOM_BT = Mom.Momentum_Backtester('SPX (SP) - 1D - (1928-01-03 - 2024-11-29)', '1928-01-03', '2024-11-29', 10000, 0.001)

# Define the range of momentum values to test
momentum_range = range(190, 200)

# Run the optimization
optimization_results = MOM_BT.parameters_optimization(momentum_range)

# Visualize the optimization process with a heatmap
MOM_BT.heatmap_visualisation(optimization_results)
```

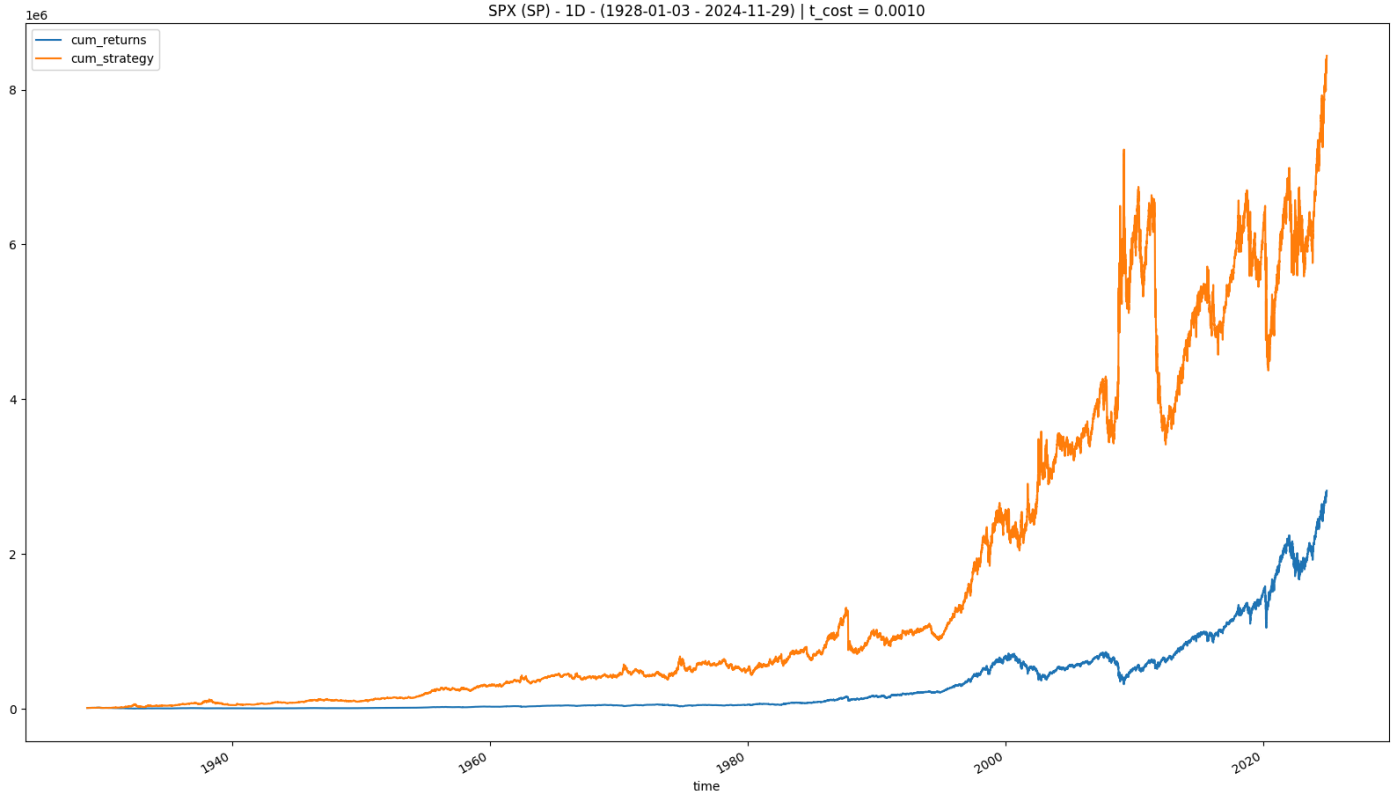
```
                open         high          low        close      Volume
time
1928-01-03    17.760000    17.760000    17.760000    17.760000           0
1928-01-04    17.719999    17.719999    17.719999    17.719999           0
1928-01-05    17.549999    17.549999    17.549999    17.549999           0
1928-01-06    17.660000    17.660000    17.660000    17.660000           0
1928-01-09    17.500000    17.500000    17.500000    17.500000           0
...                 ...          ...          ...          ...         ...
2024-11-22  5944.360000  5972.900000  5944.360000  5969.330000  2510000000
2024-11-25  5992.280000  6020.750000  5963.910000  5987.380000  3643000000
2024-11-26  6001.160000  6025.420000  5992.270000  6021.640000  2319000000
2024-11-27  6014.110000  6020.160000  5984.870000  5998.730000  2024000000
2024-11-29  6003.980000  6044.170000  6003.980000  6032.390000  1382000000

[24378 rows x 5 columns]
Best Momentum: 195, Best Performance: 8437839.38
```

```python
# Performance - optimized parameters (best parameters)
print("Total Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(195)[0])
print("Out-Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(195)[1])
```

```
Total Performance for optimized parameters (best parameters): 8437839.38
Out-Performance for optimized parameters (best parameters): 5620282.17
```

```python
# Plot result - optimized case
MOM_BT.display_results()
```

- Total performance for optimized parameters is 8437839.38 (+ 843,783,938%), with out-performance of 5620282.17 (+ 562,028,217 %), even with transaction cost of 0.1%. This shows strong and promising profitability of momentum strategy throughout the years.



- Momentum strategy outperformed throughout ~96 years time period, with area around 2010 having the strongest divergence visually, showing strongest outperformance period.
- Potential reason is because of post-global financial crisis recovery during 2008-2009 (Gorton & Metrick, 2012). The aftermath of the 2008 financial crisis had created a favorable market environment for momentum strategy. After the substantial price decline during the crisis, the S&P 500 index experienced sustained recoveries (M. Baker & Wurgler, 2013). Investors were eager to chase the overall upward trends, amplifying the momentum effect (Daniel, Hirshleifer, et al., 1998). Central banks, particularly the Federal Reserve, enacted unprecedented monetary easing (e.g. quantitative easing) (Joyce et al., 2012), providing huge liquidity to the market and further fueled momentum-driven trends.

6.1.3 Mean Reversion

- Code: Mean_Reversion_CodeRunner_TP_SPX (SP) - 1D - (1928-01-03 - 2024-11-29).ipynb

```
import Mean_Reversion_Backtester as MR
```

```
# ⭐ Initialize the backtester
MR_BT = MR.Mean_Reversion_Backtester('SPX (SP) - 1D - (1928-01-03 - 2024-11-29)', '1928-01-03', '2024-11-29', 10000, 0.001)
```

```
                open         high          low        close      Volume
time
1928-01-03    17.760000    17.760000    17.760000    17.760000           0
1928-01-04    17.719999    17.719999    17.719999    17.719999           0
1928-01-05    17.549999    17.549999    17.549999    17.549999           0
1928-01-06    17.660000    17.660000    17.660000    17.660000           0
1928-01-09    17.500000    17.500000    17.500000    17.500000           0
...                 ...          ...          ...          ...         ...
2024-11-22  5944.360000  5972.900000  5944.360000  5969.330000  2510000000
2024-11-25  5992.280000  6020.750000  5963.910000  5987.380000  3643000000
2024-11-26  6001.160000  6025.420000  5992.270000  6021.640000  2319000000
2024-11-27  6014.110000  6020.160000  5984.870000  5998.730000  2024000000
2024-11-29  6003.980000  6044.170000  6003.980000  6032.390000  1382000000

[24378 rows x 5 columns]
```

```
# ⭐ Optimize parameters:
    # Test SMA values from:
    # eg. 20 to 50 (step 5) and thresholds from 5 to 10 (step 0.5)

# ❗ SMA_range(>= 1, ...) | 1st value must be >= 1
optimization_results = MR_BT.parameters_optimization(SMA_range=(1, 1000, 5), threshold_range=(1, 500, 3))
```

```
Product: [SPX (SP) - 1D - (1928-01-03 - 2024-11-29)]

    > Best Parameters:

        > SMA: 6.0 | Threshold: 25.0

        > Strategy's Absolute Performance: 27172.12
        > Strategy's Outperformance: -3445706.21
        > Benchmark's Cumulative Returns: 3472878.33
```

- Broad search: SMA_range=(1, 1000, 5), threshold_range=(1, 500, 3)

```
# ⭐ Optimize parameters:
    # Test SMA values from:
    # eg. 20 to 50 (step 5) and thresholds from 5 to 10 (step 0.5)

# ❗ SMA_range(>= 1, ...) | 1st value must be >= 1
optimization_results = MR_BT.parameters_optimization(SMA_range=(1, 15, 1), threshold_range=(15, 35, 0.5))
```

```
Product: [SPX (SP) - 1D - (1928-01-03 - 2024-11-29)]

    > Best Parameters:

        > SMA: 6.0 | Threshold: 25.0

        > Strategy's Absolute Performance: 27172.12
        > Strategy's Outperformance: -3445706.21
        > Benchmark's Cumulative Returns: 3472878.33
```

- Focused search: SMA_range=(1, 15, 1), threshold_range=(15, 35, 0.5)
- Threshold range is iterated with step = 0.5, which is smaller than 1, to increase accuracy.

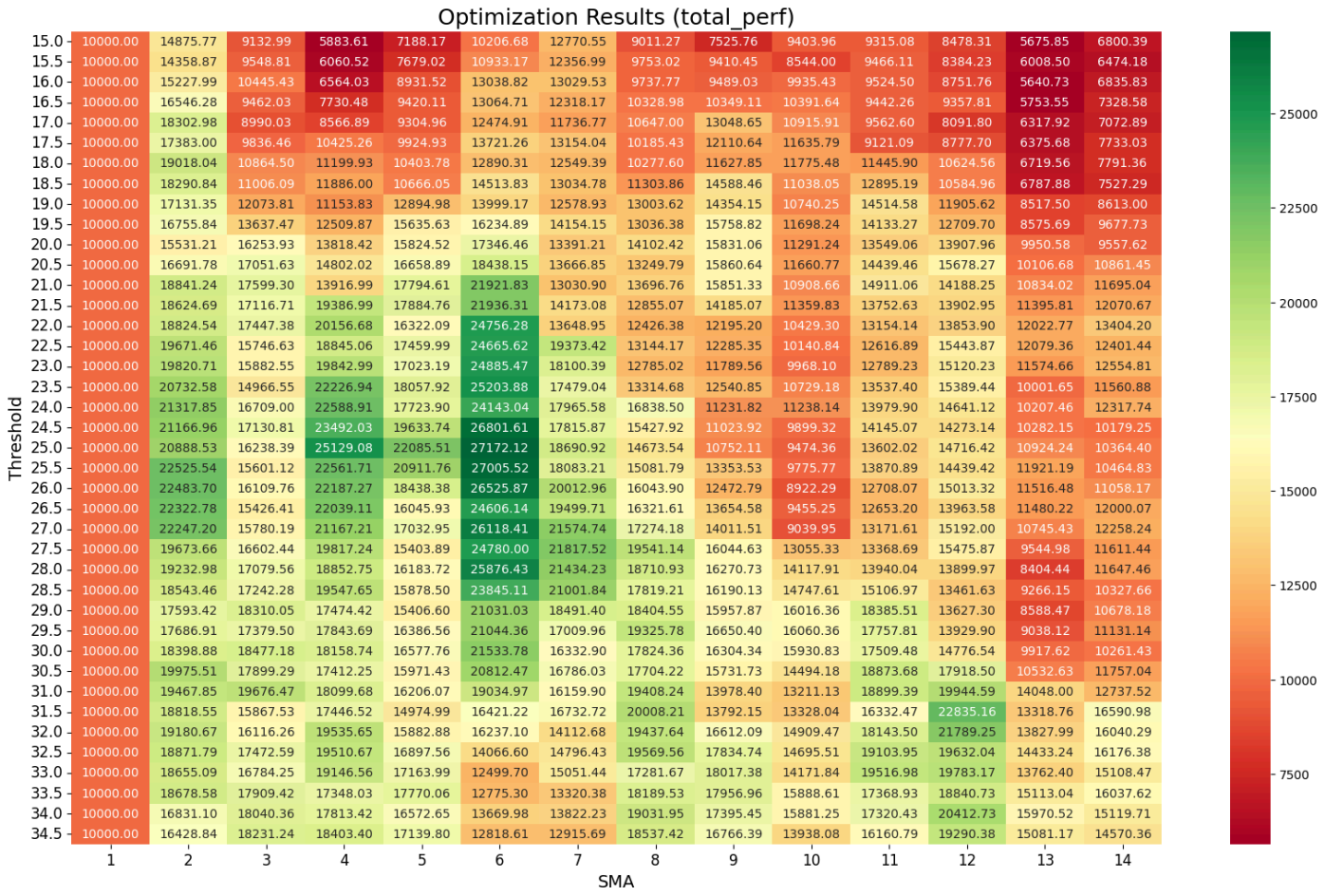
- Here, the optimal parameter combination for SPX is SMA = 6.0 & Threshold = 25.0.
- Total performance (absolute performance) is 27172.12.
- Benchmark's cumulative returns is 3,472,878.33 and outperformance is -3,445,706.21. This means that mean reversion strategy has underperformed significantly throughout the whole Daily time horizon of ~96 years, even though the parameter combination is optimized and has the best performance already.

```
# Display the optimization results
print(optimization_results)
```

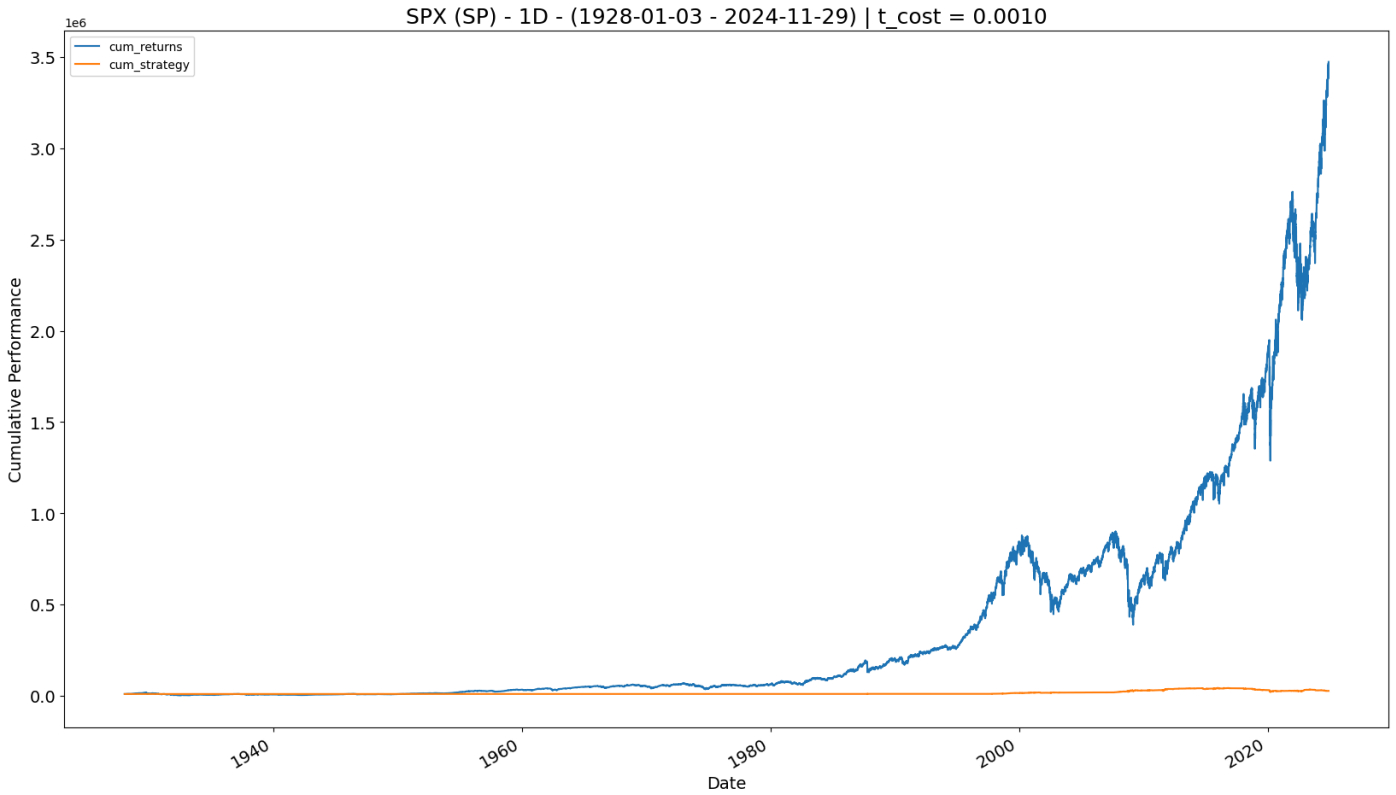
```
     SMA  Threshold  total_perf    out_perf   cum_returns
0      1       15.0    10000.00 -3386615.99  3.396616e+06
1      1       15.5    10000.00 -3386615.99  3.396616e+06
2      1       16.0    10000.00 -3386615.99  3.396616e+06
3      1       16.5    10000.00 -3386615.99  3.396616e+06
4      1       17.0    10000.00 -3386615.99  3.396616e+06
..   ...        ...         ...          ...           ...
555   14       32.5    16176.38 -3434847.65  3.451024e+06
556   14       33.0    15108.47 -3435915.55  3.451024e+06
557   14       33.5    16037.62 -3434986.41  3.451024e+06
558   14       34.0    15119.71 -3435904.31  3.451024e+06
559   14       34.5    14570.36 -3436453.67  3.451024e+06

[560 rows x 5 columns]
```

```
# Calculate strategy performance in different combinations of parameters

# Plot heatmap of Total Performance (total_perf)
    # After mean-reversion strategy is applied, final portfolio value
    # Calculated as the value of the cumulative performance of the strategy (cstrategy) at the last date in the dataset
    # aperf = self.results['cstrategy'].iloc[-1]

MR_BT.heatmap_visualisation(metric='total_perf')
```

```
Plotting heatmap for metric: total_perf
```



- Here, the total performance heatmap has quite evenly distributed values as the changes in color gradient are very smooth and natural, indicating there is no sudden outlier within this set of range.
- This also suggests the effectiveness of mean reversion strategy is very low in this Daily timeframe of 1928-01-03 - 2024-11-29.



- Mean reversion strategy return is so low that the orange curve is flattened out near the area of 0.0, while asset return is increasing exponentially.
- Potential reason: Inefficiency of Mean Reversion over longer timeframes
    1. Data of daily time frame has no mean-reverting characteristics:
        - Over the prolonged time horizon of ~96 years, equity markets exhibit significant upward drift due to economic growth, inflation, and corporate earnings growth. This long-term bullish trend in stocks dominates any potential short-term mean-reverting behavior, making mean reversion strategy less effective.
    2. Mean Reversion may works better on short timeframes
        - Due to the design nature of mean reversion strategy, it is expected to be more effective on intraday, shorter-term time frames (smaller than Daily timeframe level), in which prices oscillate around the short-term mean (e.g., due to noise, liquidity imbalances, or market overreaction).

6.2. Time frame analysis

The selection of time frames in algorithmic trading can be a critical decision that significantly influences the design, performance, and applicability of trading strategies. Time frames ranging from short-term granular intervals (eg. 5 min) to long-term macro intervals (eg. weekly) play a pivotal role in determining trades duration and risk exposure. In this part, the effectiveness of different strategies in different time frames (5 min, 1 hour, Daily, Weekly) will be analysed.

Data will be S&P 500 index with date range 2 Jan 2024 to 29 Nov 2024, from TradingView SPX (SP).

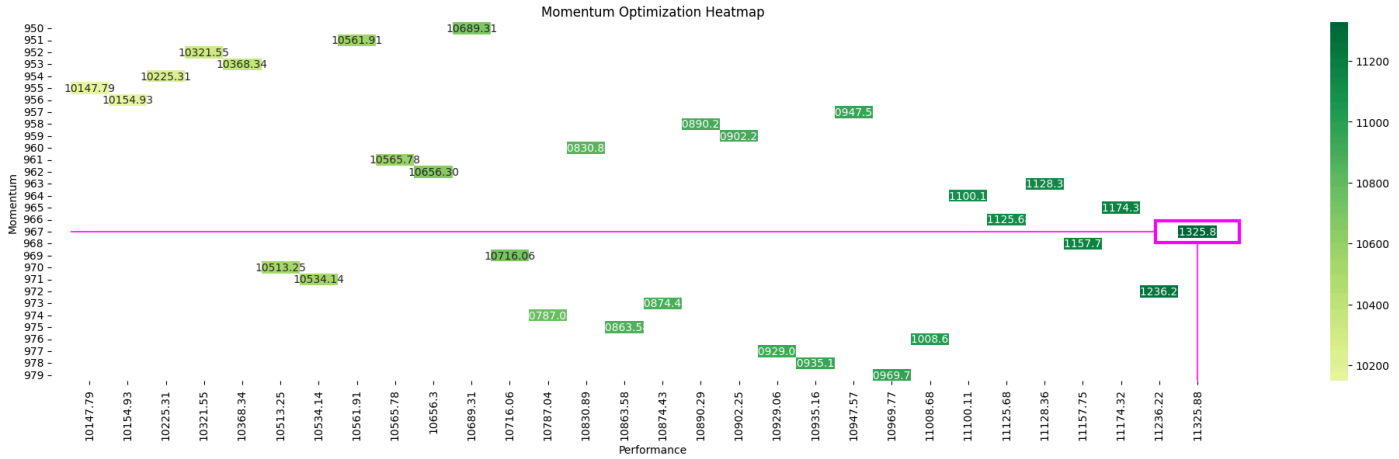
6.2.1. 5 min time frame

The 5 min time frame is characterized by its high granularity, capturing short-term price fluctuations with exceptional precision. Since many traders seek to exploit small and fast price movements nowadays, it is widely used in scalping and high-frequency trading (HFT) strategies. However, the 5 min time frame has high sensitivity to market noise and short-term volatility. Strategies may need to be adjusted to filter out irrelevant fluctuations to generate meaningful signals. Also, the data-intensive nature demands significant computational resources for both backtesting and live trading, but position entry point and exit point can be fine-tuned to minimize delay of signals and response to market changes quicker.

6.2.1.1. Momentum

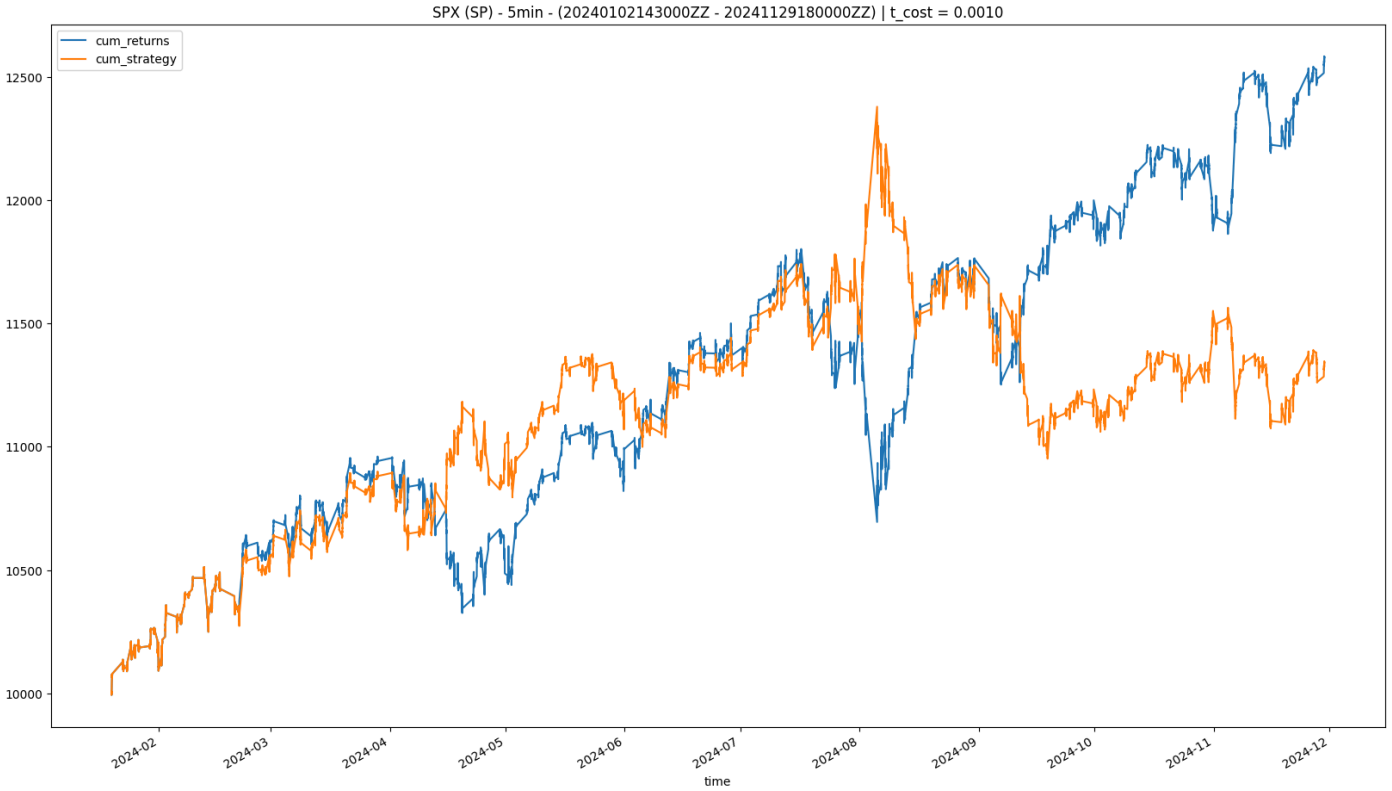
time	open	high	low	close	Volume
2024-01-02 14:30:00+00:00	4745.20	4745.20	4738.63	4743.20	100388539
2024-01-02 14:35:00+00:00	4743.50	4744.00	4736.52	4736.81	38780551
2024-01-02 14:40:00+00:00	4736.57	4737.52	4734.49	4735.30	37744641
2024-01-02 14:45:00+00:00	4735.28	4738.26	4731.04	4731.09	38725836
2024-01-02 14:50:00+00:00	4731.00	4736.08	4730.95	4731.45	35520078
...	...	...	...	...	...
2024-11-29 17:40:00+00:00	6043.01	6043.37	6041.83	6042.34	22645680
2024-11-29 17:45:00+00:00	6042.30	6043.18	6041.67	6042.58	28750220
2024-11-29 17:50:00+00:00	6042.63	6044.17	6038.56	6039.81	47846060
2024-11-29 17:55:00+00:00	6040.01	6040.01	6033.33	6034.10	109717710
2024-11-29 18:00:00+00:00	6033.57	6034.40	6032.39	6032.39	323708400

[18166 rows x 5 columns]
Best Momentum: 967, Best Performance: 11325.88




```
print("Total Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(967)[0])
print("Out-Performance for optimized parameters (best parameters):", MOM_BT.execute_strategy(967)[1])
```

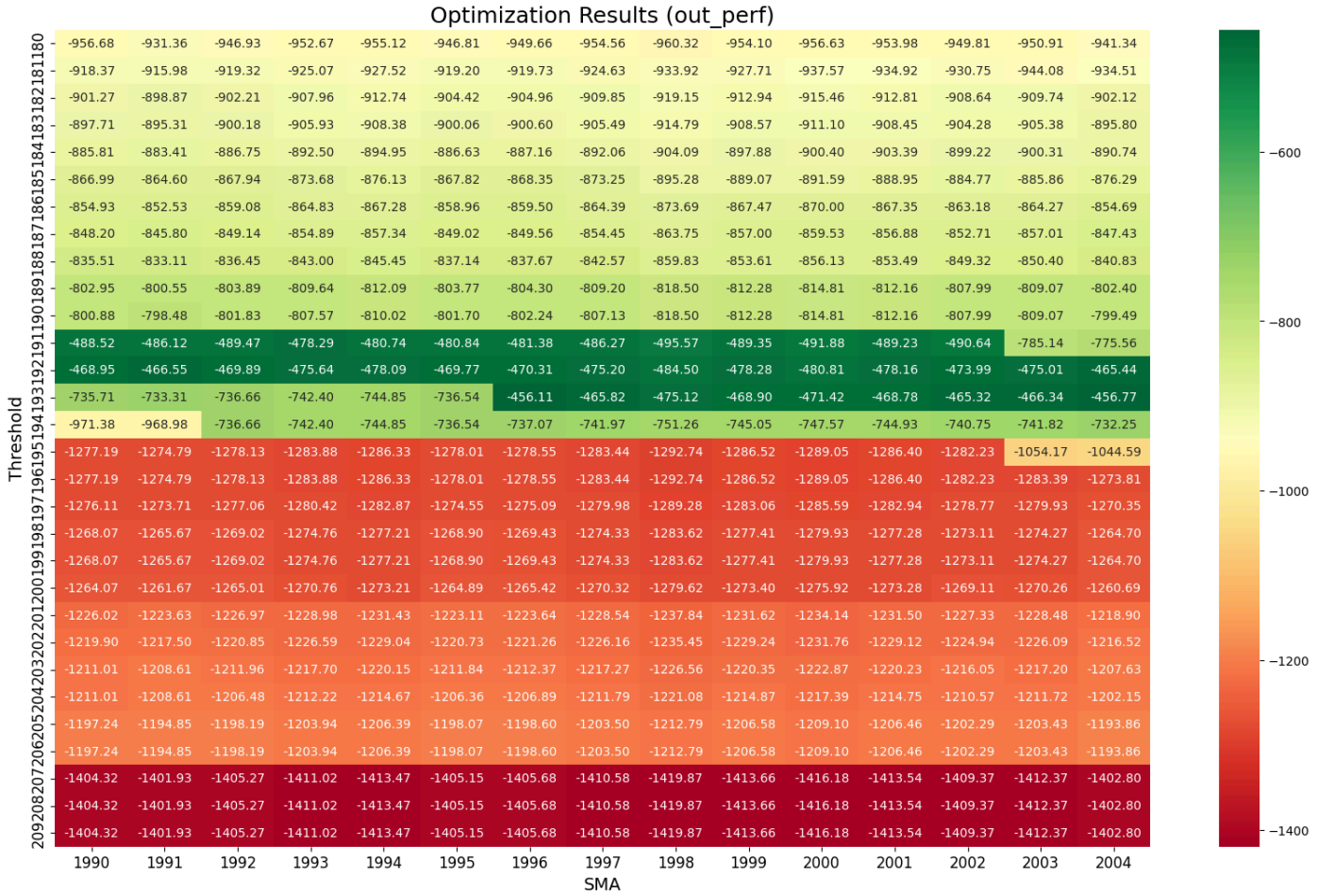
```
Total Performance for optimized parameters (best parameters): 11325.88
Out-Performance for optimized parameters (best parameters): -1236.29
```

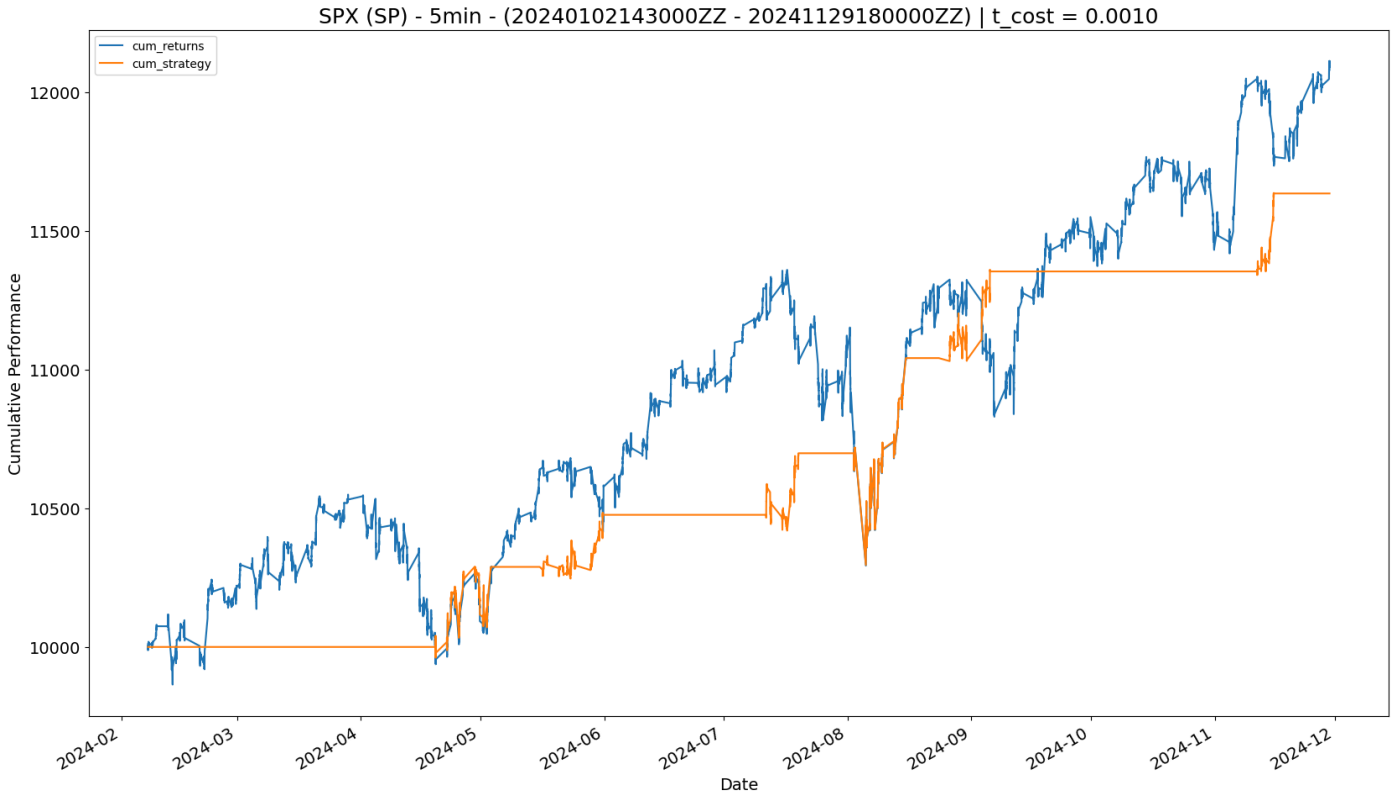


- In the 5 min time frame, even using the best parameter, momentum strategy still underperformed the asset return, although profitable.

6.2.1.2. Mean Reversion

```
Product: [SPX (SP) - 5min - (20240102143000ZZ - 20241129180000ZZ)]

    > Best Parameters:

        > SMA: 1996.0 | Threshold: 193.0

        > Strategy's Absolute Performance: 11634.45
        > Strategy's Outperformance: -456.11
        > Benchmark's Cumulative Returns: 12090.56


Total Performance for optimized parameters (best parameters): 11634.45
Out-Performance for optimized parameters (best parameters): -456.11
```





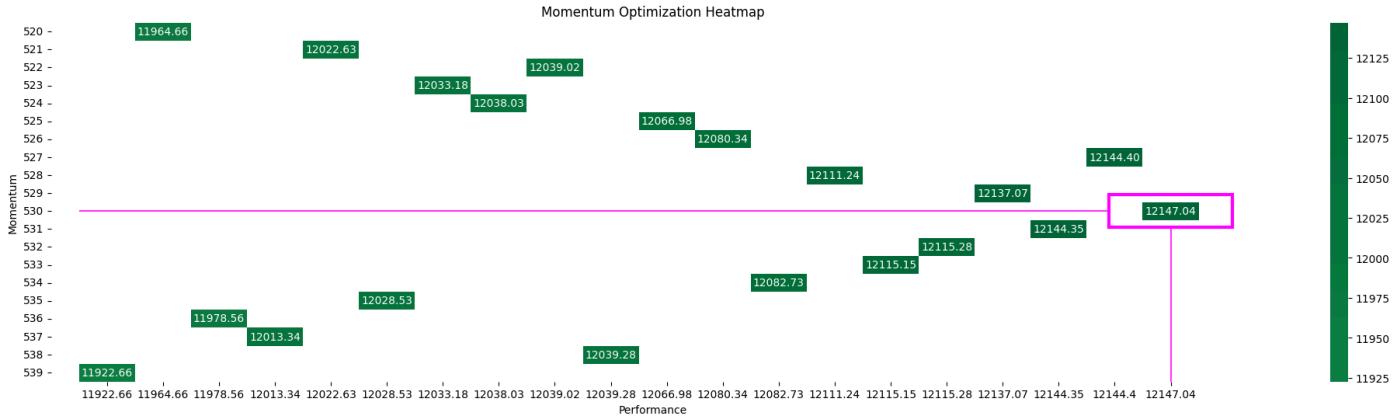
- In the 5 min time frame, same with momentum strategy, even using the best parameter, mean reversion strategy still underperformed the asset return, although profitable.

6.2.2. 1 hour time frame

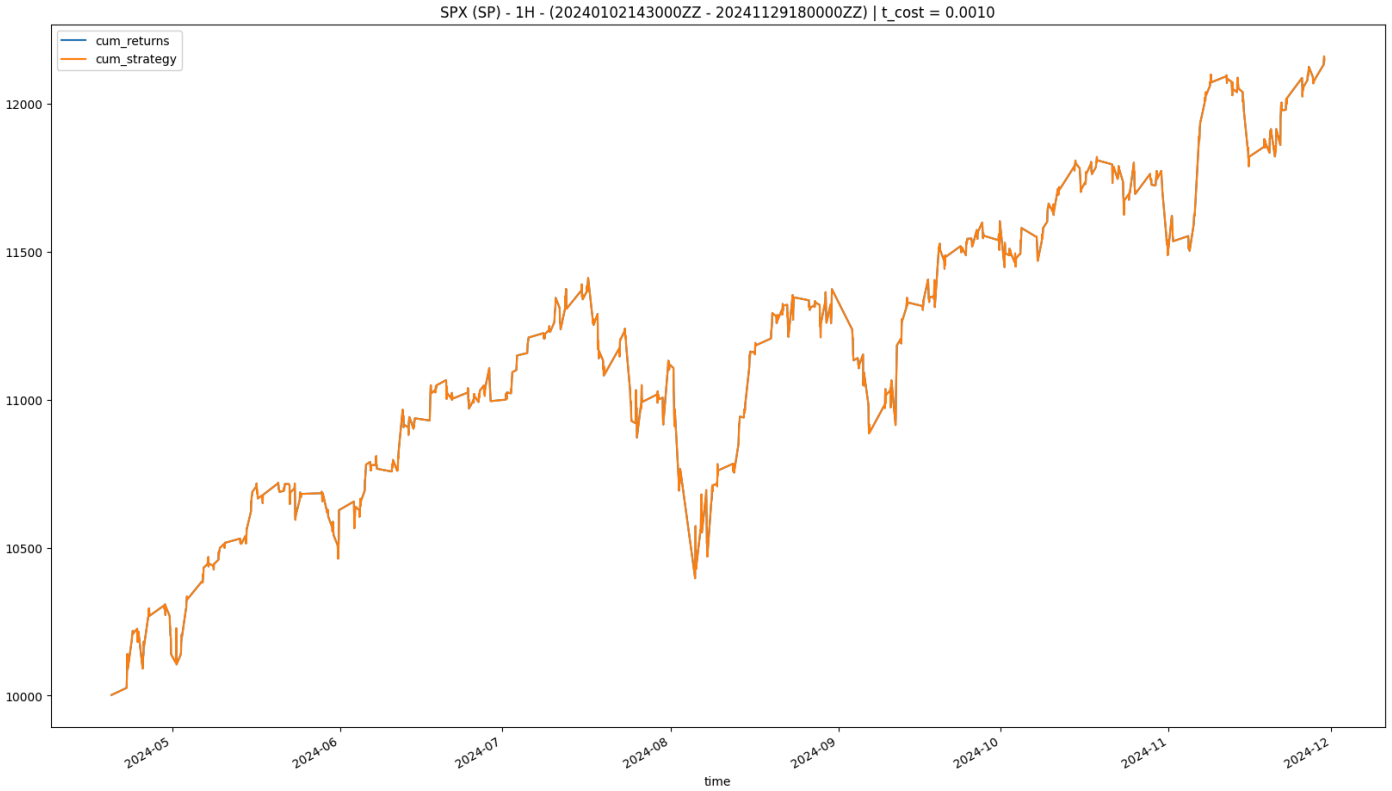
The 1 hour time frame offers a balance between granularity and trend clarity. The timeframe has gained popularity for swing trading and intraday trading nowadays. The 1 hour time frame helps reduce a certain level of market noise compared to lower time frames, such as 5 min time frame, potentially allowing traders to identify patterns, trends, and reversals clearer.

6.2.2.1. Momentum

```
                           open     high      low    close     Volume
time
2024-01-02 14:30:00+00:00  4745.20  4745.20  4730.95  4736.25  460192930
2024-01-02 15:30:00+00:00  4736.11  4742.45  4731.16  4741.11  292380240
2024-01-02 16:30:00+00:00  4740.88  4751.22  4740.88  4747.54  217999400
2024-01-02 17:30:00+00:00  4747.57  4754.33  4740.93  4744.81  185891580
2024-01-02 18:30:00+00:00  4744.82  4747.23  4733.68  4737.01  194987940
...                            ...      ...      ...      ...        ...
2024-11-27 20:30:00+00:00  5997.03  6001.11  5991.48  5998.73  641123020
2024-11-29 14:30:00+00:00  6003.98  6028.51  6003.98  6025.37  384673350
2024-11-29 15:30:00+00:00  6025.33  6035.07  6025.19  6034.29  223282780
2024-11-29 16:30:00+00:00  6034.33  6043.18  6031.45  6039.17  201124140
2024-11-29 17:30:00+00:00  6039.12  6044.17  6032.39  6032.39  573507160

[1611 rows x 5 columns]
Best Momentum: 530, Best Performance: 12147.04


Total Performance for optimized parameters (best parameters): 12147.04
Out-Performance for optimized parameters (best parameters): 0.0
```



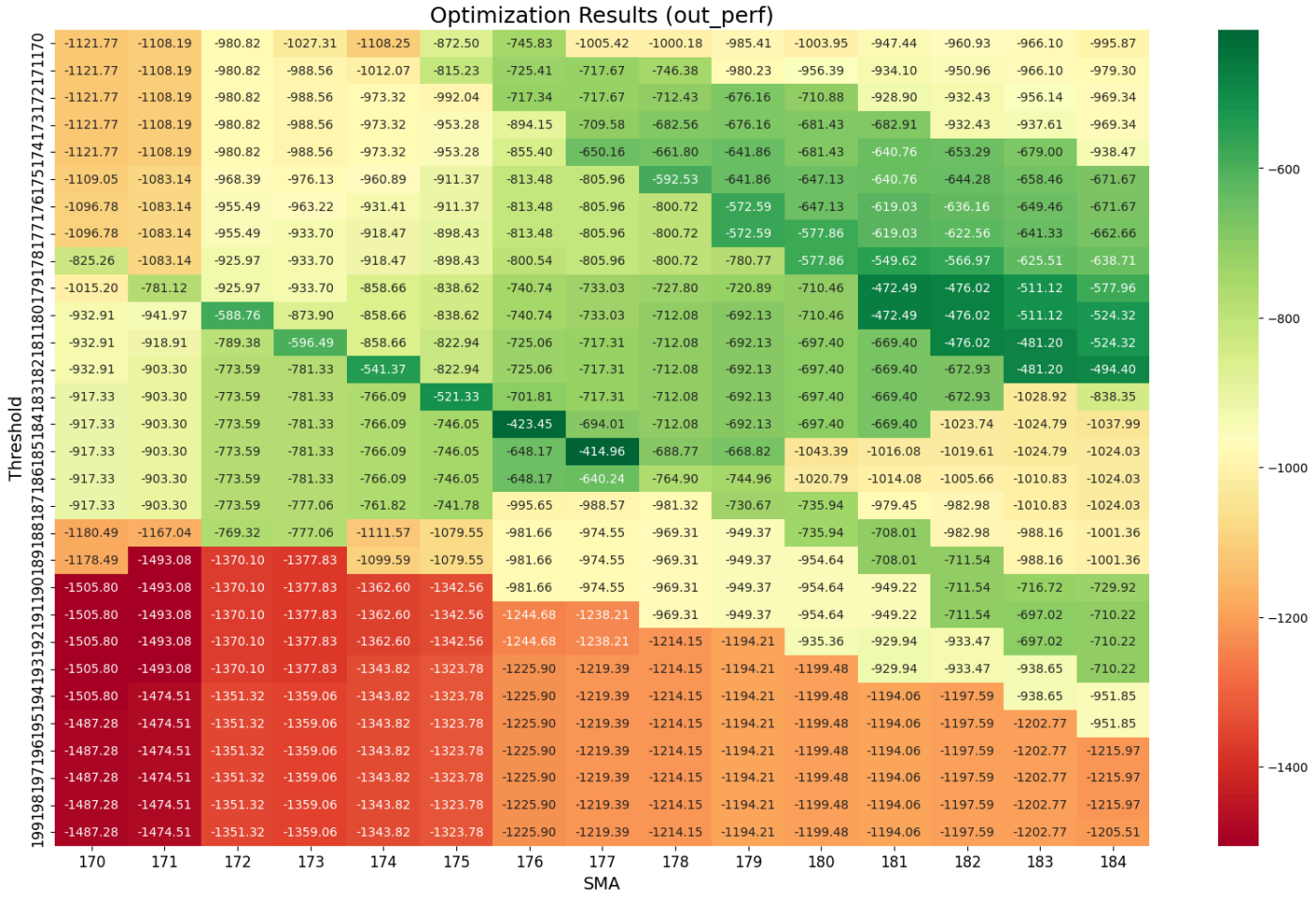
- Here, Outperformance = 0 indicates the potential scenario that only 1 trade is opened and a long position is still being held, as outperformance = 0 and green area with profitability is shown.

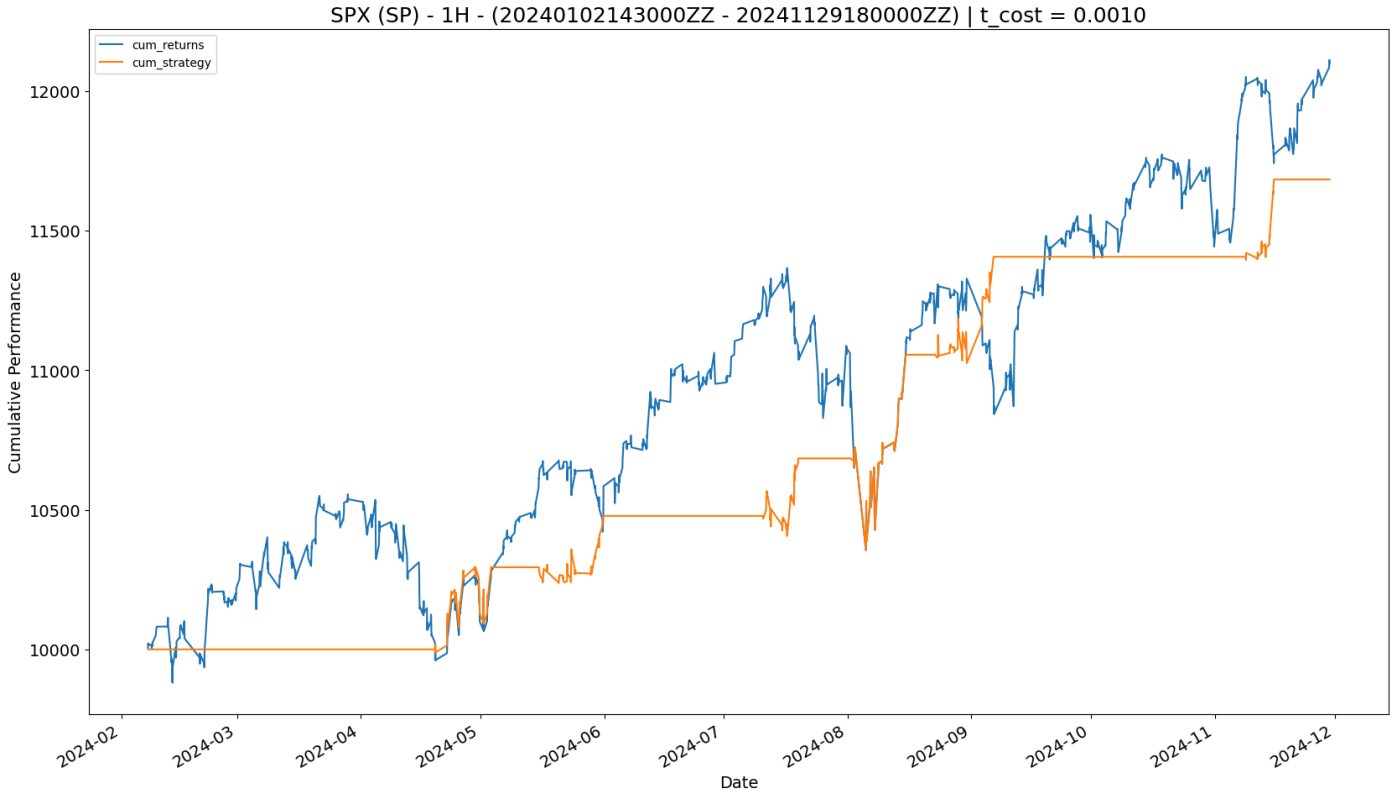


- In 1 hour time frame, the exact overlapping further proves only 1 long position is opened and is ongoing, potentially keep rising with asset return.

6.2.2.2. Mean Reversion

```
Product: [SPX (SP) - 1H - (202401021430000ZZ - 202411291800000ZZ)]

    > Best Parameters:

        > SMA: 177.0 | Threshold: 185.0

        > Strategy's Absolute Performance: 11683.57
        > Strategy's Outperformance: -414.96
        > Benchmark's Cumulative Returns: 12098.53


Total Performance for optimized parameters (best parameters): 11683.57
Out-Performance for optimized parameters (best parameters): -414.96
```





- In 1 hour time frame, mean reversion strategy underperforms all the time in general, although it is profitable.

6.2.3. Daily time frame

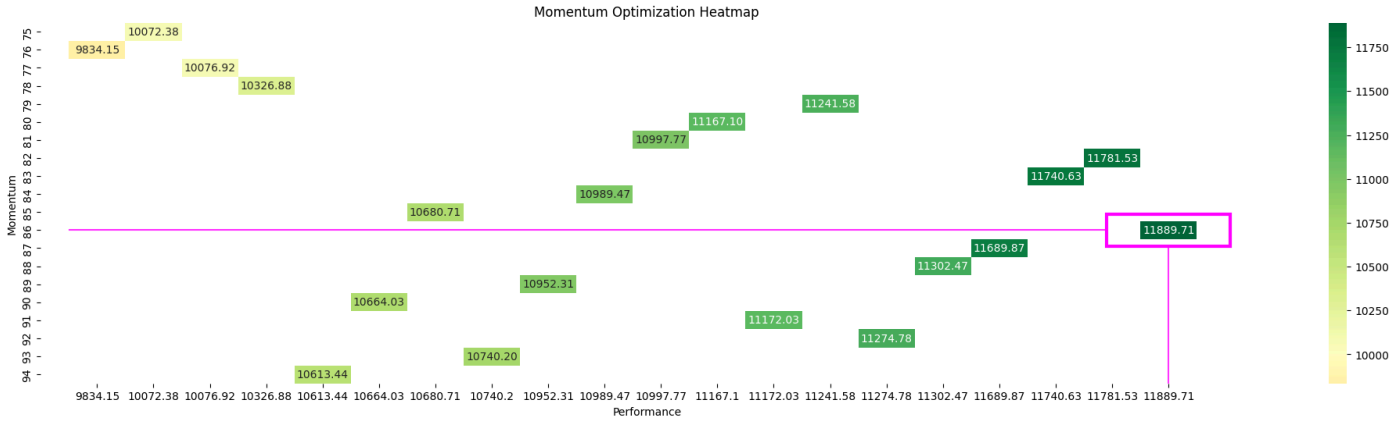
The daily time frame focuses more on macro-level price movements, smoothing out intraday volatility and random fluctuations. It is widely used in both swing and position trading strategies, where trades are typically held for days to weeks. Not only does significant trends and patterns, including breakouts and reversals, can be highlighted in the daily time frame, but it also aligns well with fundamental analysis, such as macroeconomic events and earnings reports.

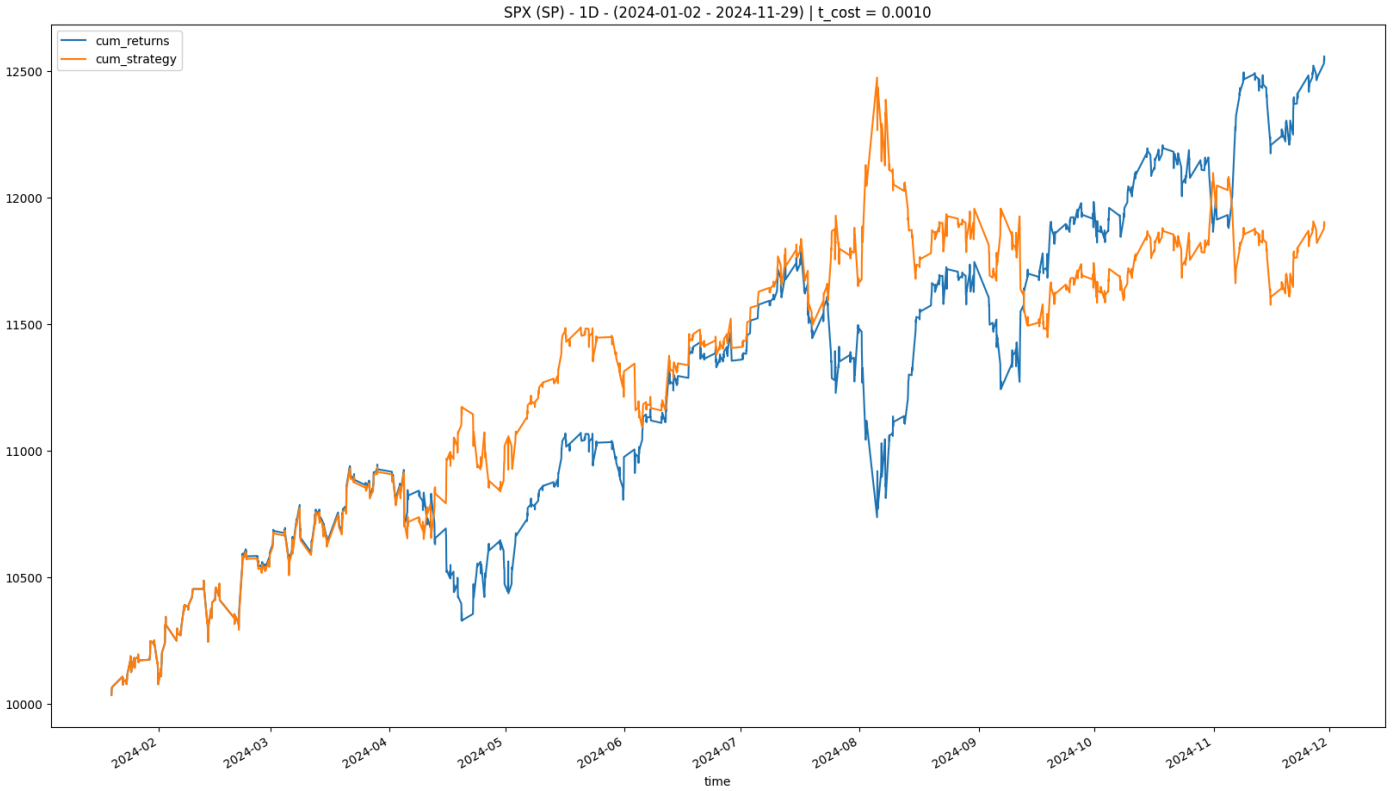
6.2.3.1. Momentum

time	open	high	low	close	Volume
2024-01-02 14:30:00+00:00	4745.20	4745.20	4730.95	4736.25	460192930
2024-01-02 15:30:00+00:00	4736.11	4742.45	4731.16	4741.11	292380240
2024-01-02 16:30:00+00:00	4740.88	4751.22	4740.88	4747.54	217999400
2024-01-02 17:30:00+00:00	4747.57	4754.33	4740.93	4744.81	185891580
2024-01-02 18:30:00+00:00	4744.82	4747.23	4733.68	4737.01	194987940
...	...	...	...	...	...
2024-11-27 20:30:00+00:00	5997.03	6001.11	5991.48	5998.73	641123020
2024-11-29 14:30:00+00:00	6003.98	6028.51	6003.98	6025.37	384673350
2024-11-29 15:30:00+00:00	6025.33	6035.07	6025.19	6034.29	223282780
2024-11-29 16:30:00+00:00	6034.33	6043.18	6031.45	6039.17	201124140
2024-11-29 17:30:00+00:00	6039.12	6044.17	6032.39	6032.39	573507160

[1611 rows x 5 columns]
Best Momentum: 86, Best Performance: 11889.71

Total Performance for optimized parameters (best parameters): 11889.71
Out-Performance for optimized parameters (best parameters): -653.93

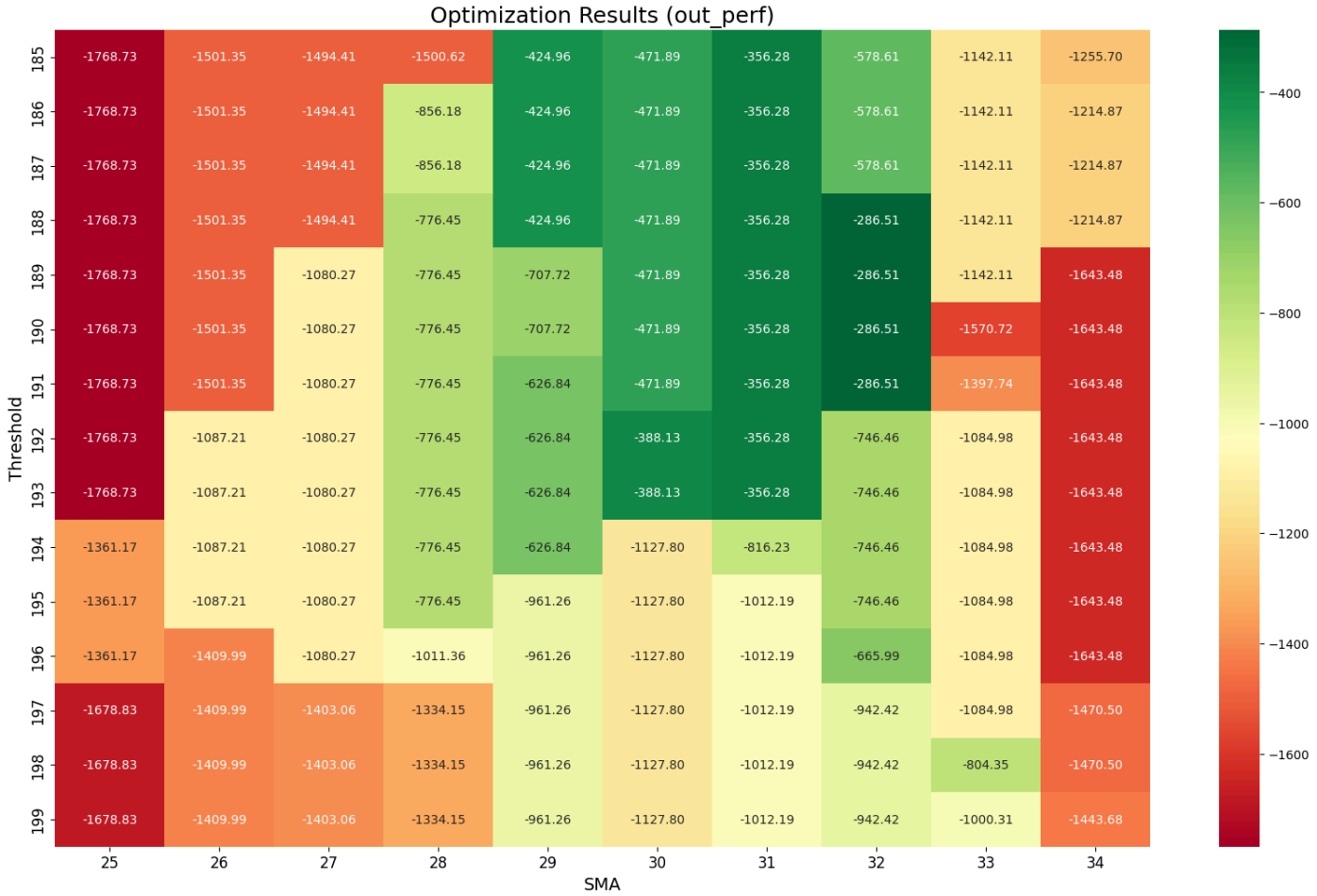


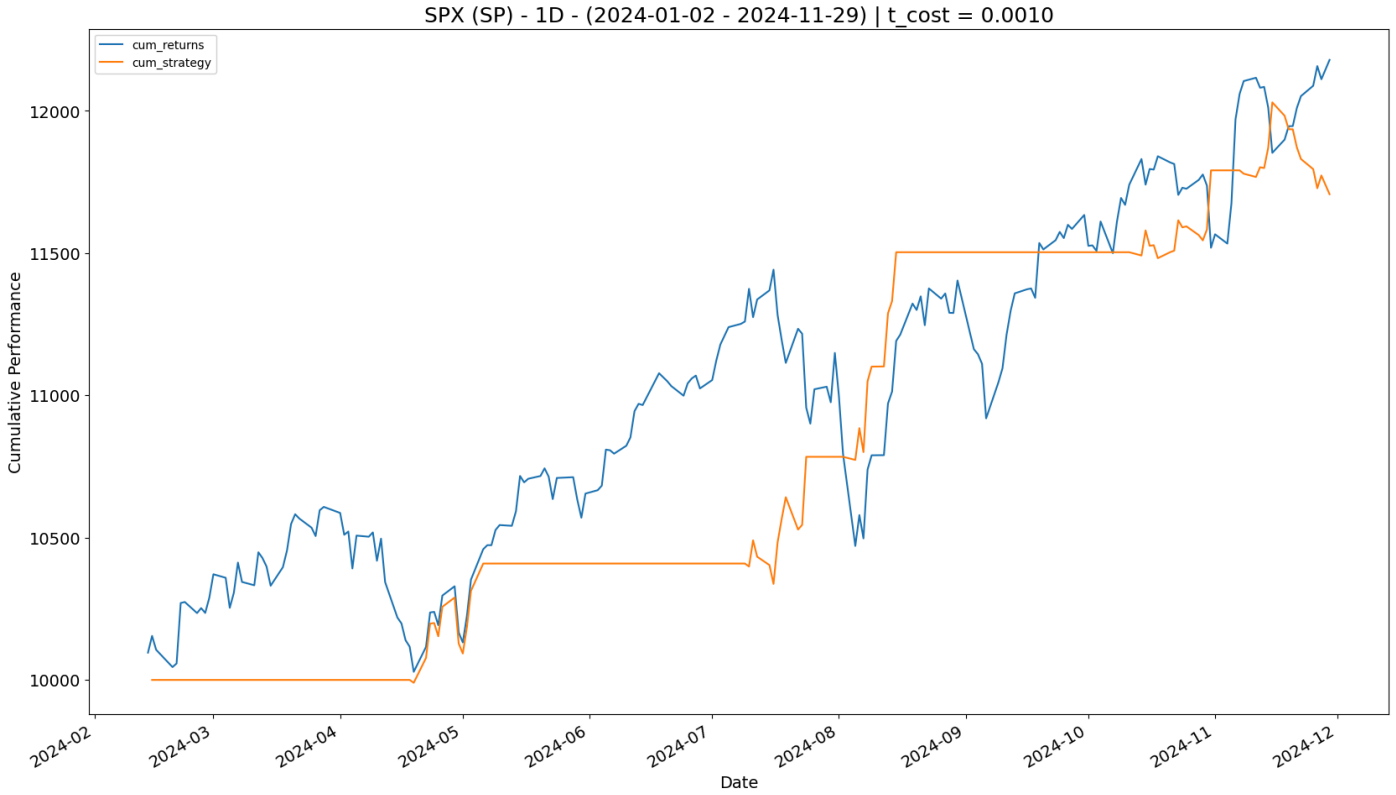


- In the daily time frame, momentum strategy is profitable but underperforms.

6.2.3.2. Mean Reversion

```
Product: [SPX (SP) - 1D - (2024-01-02 - 2024-11-29)]

    > Best Parameters:

        > SMA: 30.0 | Threshold: 192.0

        > Strategy's Absolute Performance: 11790.74
        > Strategy's Outperformance: -388.13
        > Benchmark's Cumulative Returns: 12178.87



Total Performance for optimized parameters (best parameters): 11790.74
Out-Performance for optimized parameters (best parameters): -471.89
```





- In the daily timeframe, mean reversion strategy underperforms in general, except during mid August to mid September in 2024.

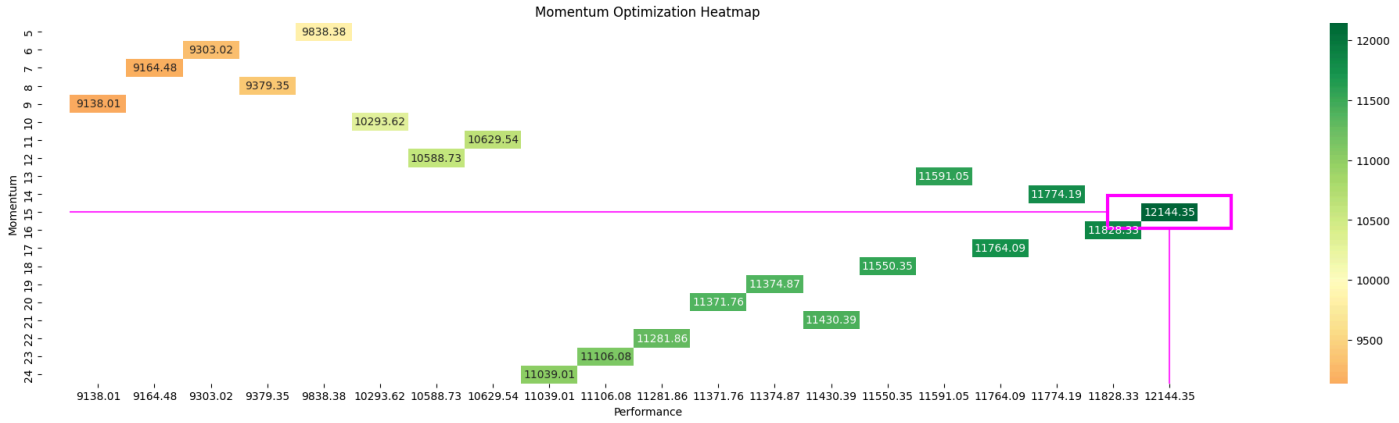
6.2.4. Weekly time frame

The weekly time frame provides a long-term perspective, focusing on structural market trends that develop over weeks or months. The aggregation of weekly level price data eliminates short-term noise and emphasizes stable, directional movement in the market. The weekly time frame is commonly used in position trading and portfolio management strategies, in which trades are aligned with various macroeconomic trends and market structural shifts. Trading frequency is also expected to be very low when using the weekly time frame for trading in general.

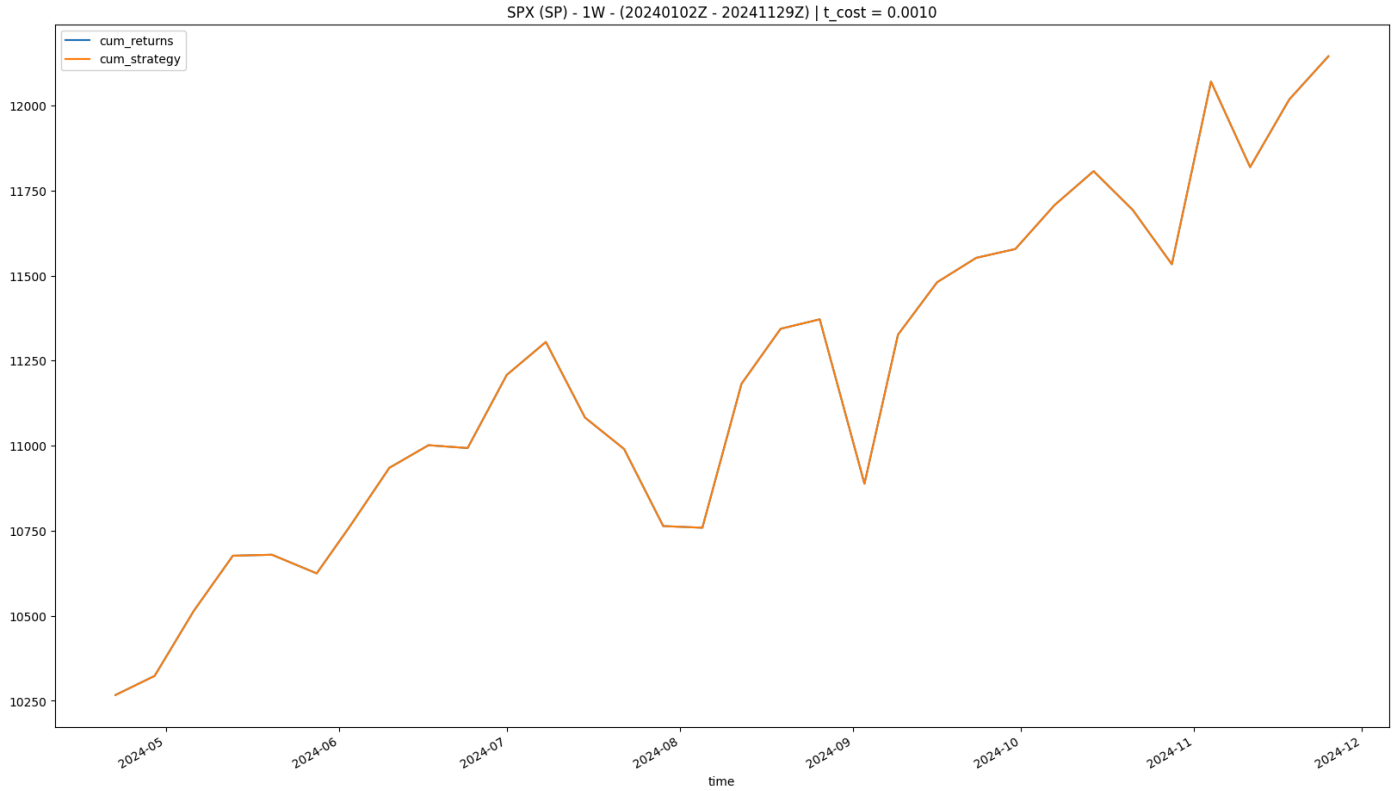
6.2.4.1. Momentum

```
                open     high      low    close       Volume
time
2024-01-02  4745.20  4754.33  4682.11  4697.23   9147000000
2024-01-08  4703.70  4802.40  4699.82  4783.84  11050000000
2024-01-16  4772.35  4842.07  4714.82  4839.82  10014000000

                           ⋮

2024-11-04  5725.15  6012.45  5696.51  5995.53  14370000000
2024-11-11  6008.86  6017.31  5853.01  5870.63  13288000000
2024-11-18  5874.17  5972.90  5855.29  5969.33  12952000000
2024-11-25  5992.28  6044.17  5963.91  6032.39   9368000000
Best Momentum: 15, Best Performance: 12144.35
```

```
Total Performance for optimized parameters (best parameters): 12144.35
Out-Performance for optimized parameters (best parameters): 0.0
```

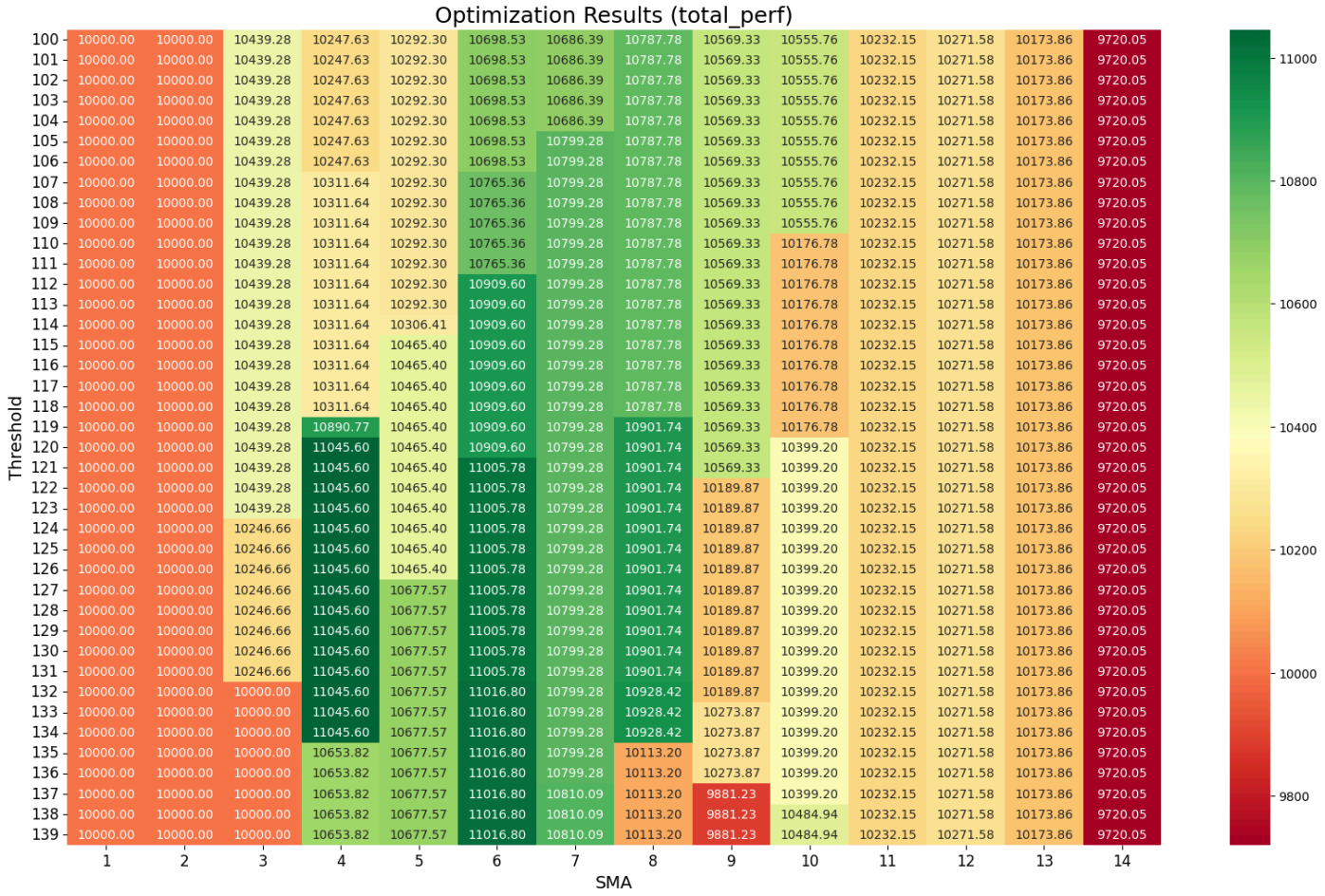


- Here, same with the 1 hour time frame momentum part, outperformance = 0 indicates the potential scenario that only 1 trade is opened and a long position is still being held, as outperformance = 0 and green area with profitability is shown.

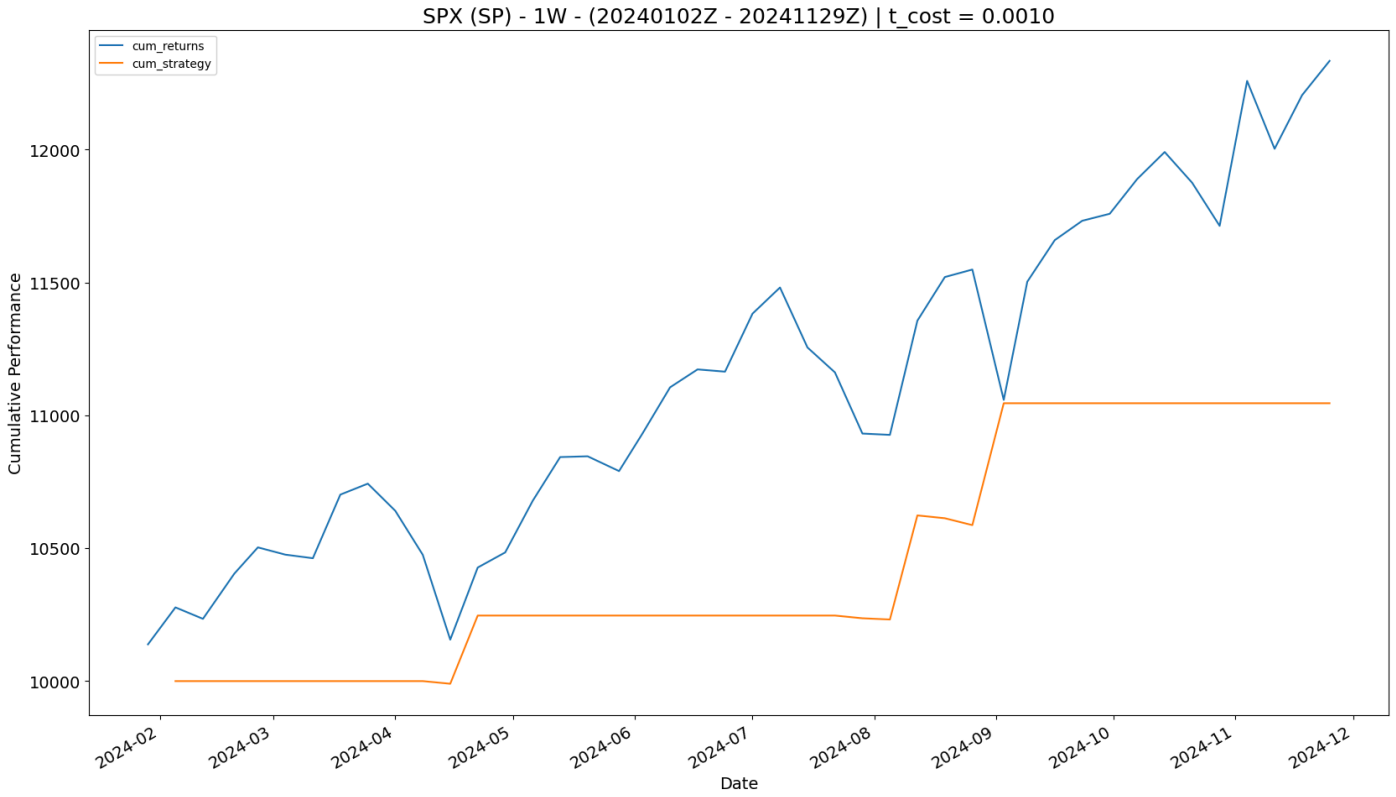


6.2.4.2. Mean Reversion

```
Product: [SPX (SP) - 1W - (20240102Z - 20241129Z)]

    > Best Parameters:

        > SMA: 4.0 | Threshold: 120.0

        > Strategy's Absolute Performance: 11045.60
        > Strategy's Outperformance: -1288.15
        > Benchmark's Cumulative Returns: 12333.75


Total Performance for optimized parameters (best parameters): 11045.6
Out-Performance for optimized parameters (best parameters): -1288.15
```



- In the weekly time frame, from the heatmap, parameter SMA is much more sensitive than that of Threshold, as color gradient changes throughout SMA ranger are much faster than that of Threshold change (which is similar in general), showing high sensitivity of parameter SMA in weekly timeframe.



- In the weekly time frame, although the mean reversion strategy is profitable, it underperforms throughout the whole time series, showing its weak effectiveness in the weekly timeframe.

6.2.5 Summary of data of different time frame

6.2.5.1. Optimal parameters

Strategy / Timeframe	Momentum	Mean Reversion	
5 min	Momentum rolling window = 967	SMA = 1996.0	Threshold = 193.0
1 hour	Momentum rolling window = 530	SMA = 177.0	Threshold = 185.0
Daily	Momentum rolling window = 86	SMA = 30.0	Threshold = 192.0
Weekly	Momentum rolling window = 15	SMA = 4.0	Threshold = 120.0

- For momentum strategy,
  - Shorter timeframes 5 min and 1 hour have higher optimal momentum rolling window values 967 and 530 respectively. The longer momentum rolling window helps smoothing out the increased noise and volatility associated with smaller time intervals. Shorter timeframes capture more frequent price fluctuations, which may often be dominated by market noise,

microstructure effects, or random short-term movements. In order to extract meaningful signals and trends noisy data, a longer rolling window is used to average out the fluctuations.

    - Longer timeframes Daily and Weekly time frames have lower optimal momentum rolling window values of 86 and 15 respectively and are efficient because price movements over longer time horizons are inherently smoother and less affected by noise, allowing a shorter rolling window effectively capturing useful momentum signals.

- For mean reversion strategy,
    - The larger the timeframe, the lower the optimal SMA is in the optimal combination of SMA and Threshold, from 5 min SMA = 1996.0 down to Weekly SMA = 4.0. This is because longer timeframes naturally exhibit smoother price movements due to the aggregation of data over extended periods. On the contrary, in shorter time frames, such as 5 min and 1 hour, price movements are more volatile and noisy, necessitating a longer SMA to establish a more stable reference point for detecting price deviations to generate signals.
    - While for the optimal Threshold, values are close to each other in general, except Weekly Threshold being less (120.0). The similarity of Threshold value is potentially due to the nature of the strategy, as the strategy mainly focuses on identifying price deviations to find potential trading entry point, regardless of timeframe.

6.2.5.2. Parametric sensitivity of optimal parameters

From 6.2.1 to 6.2.4, parametric sensitivity of optimal parameters is captured from heatmap (visualization of optimizing process). And below is the gradient-based sensitivity classification based on the subjective human interpretation. It helps analyze how the values in the heatmap change with respect to the parameters being optimized and focuses on the rate of change (the gradient) across the heatmap.

Strategy / Timeframe	Momentum	Mean Reversion	
5 min	mid	SMA = low	Threshold = high
1 hour	low	SMA = mid	Threshold = high
Daily	mid	SMA = high	Threshold = high
Weekly	high	SMA = mid	Threshold = low

If solely consider parametric sensitivity of optimal parameters:

1. For executing momentum strategy, choose 1 hour timeframe, since the sensitivity of momentum rolling window is the lowest (low).
    - For the 1 hour timeframe, compared to the 5 min time frame, it shows a smoother, slower response to price changes, which potentially helps filtering noise over a medium timeframe, such as short-term small liquidity spikes, increasing signal quality. By filtering out noise, the strategy can better focus on meaningful momentum signals instead of reacting to short-term volatility, such as false break out, which may potentially cause loss as momentum strategy is a trend following strategy, not a counter-trend strategy that caters for false break out.
    - While compared to Daily and Weekly timeframe, not only does it have lower sensitivity, but it also has higher responsiveness to price changes, generating signals in a faster manner to reduce delay reaction to emerging trends and optimal entry time.
    - And the delay can result in the dilemma that there will be less momentum remaining for this trend wave when entering into position late, potentially missing out on an earlier entry point where the risk-to-reward ratio is higher.
2. For executing mean reversion strategy, choose Weekly time frame, since the combination of sensitivity of SMA and Threshold is the lowest.
    - For the Weekly time frame, compared to the shorter time frame (5 min, 1 hour, Daily), it has greater stability on SMA, allowing strategy to be less reliant on hyper-precise parameter optimization. For Threshold, since the weekly time frame focuses on long-term trends and momentum, the thresholds used to trigger entries and exits are less sensitive to small changes. Therefore, a slight increment or decrement in threshold value will not significantly affect the strategy's performance because the signals are based on sustained movements over weeks rather than short-term price fluctuations, demonstrating robustness of stability performance.

6.2.5.3. Total performance

Strategy / Timeframe	Momentum	Mean Reversion
5 min	11325.88	11634.45
1 hour	12147.04	11683.57
Daily	11899.71	11790.74
Weekly	12144.35	11045.60

- Throughout the total performance table, both momentum and mean reversion strategy is profitable, with > 10 % return throughout the year of 2024 (excluding December).

- In general, momentum strategy outperforms mean reversion, with 1 hour and Weekly time frame being the most profitable, having > 20 % return. This indicates that 1 hour and Weekly time frame may suit momentum strategy the most.

6.2.5.4. Outperformance

Strategy / Timeframe	Momentum	Mean Reversion
5 min	-1236.29	-456.11
1 hour	0.0	-414.98
Daily	-653.93	-471.89
Weekly	0.0	-1288.15

- Here, it shows that both momentum and mean reversion strategy underperform benchmark (asset normal return), except for momentum strategy of 1 hour and Weekly time frame.
- In the 1 hour and Weekly time frame, the 0.0 shows that only 1 long position is opened, and the position is not yet closed but still ongoing. In 6.2.2.1 and 6.2.3.1, the 1 hour and Weekly time frame respectively, the optimal momentum rolling windows are 530 and 15 respectively, while the number of rows of data are 1611 and 15 respectively. This means that the trading frequency of momentum strategy in these 2 time frames is very low, as the ratio of "optimal momentum rolling windows / number of rows of data" is high. The ratios are 32.90% and 100% in the 1 hour and Weekly time frame respectively. This also suggests if no leverage is used, the simplest "buy and hold" method can be more profitable than implementing momentum and mean reversion strategy in general.

6.3 Market conditions comparison - Bull and Bear markets

The goal of this section is to assess the performance and effectiveness of different trading strategies (momentum and mean reversion) across different market conditions (bull and bear markets), to select the best strategy to use during bull market and bear market.

Bull market vs Bear market	
Dot-Com Bubble Bull Market (1995-01-04 - 2000-09-10)	Dot-Com Crash Bear Market (2000-09-10 - 2002-10-09)
Post-Global Financial Crisis Bull Market (2009-03-09 - 2020-02-19)	Global Financial Crisis Bear Market (2007-10-09 - 2009-03-09)

Post-COVID Bull Market (2020-03-23 - 2022-01-03)	COVID-19 Crash Bear Market (2020-02-20 - 2020-03-23)
	Inflation and Rate-Hike Bear Market (2022-01-04 - 2022-10-12)

The “Bull market vs Bear market” table above summarizes selected bull and bear markets for analysis and these markets are with high correlations in general, as both timeline and event align, which helps increase comparison robustness.

- Bull market used in comparison are:
    - Dot-Com Bubble Bull Market (Edenfield, 2003)
    - Post-Global Financial Crisis Bull Market (Bekaert et al., 2013)
    - Post-COVID Bull Market (Cheema-Fox et al., 2020)
- Bear market used in comparison are:
    - Dot-Com Crash Bear Market (Ofek & Richardson, 2003),
    - Global Financial Crisis Bear Market (Reinhart & Rogoff, 2009)
    - COVID-19 Crash Bear Market (Baker et al., 2020)
    - Inflation and Rate-Hike Bear Market (Toye, 2021)

Optimal parameters, total performance and outperformance tables will be included throughout this section. Those parameters and performance values are strategies’ optimal solution with best total performance in every single event in bull markets and bear markets, to show the best possible case in every single scenario.

6.3.1 Bull markets analysis

Optimal parameters:

Strategy / Market condition	Momentum	Mean Reversion	
Dot-Com Bubble Bull Market (1995-01-04 - 2000-09-10)	206	SMA = 14	Threshold = 22
Post-Global Financial Crisis Bull Market (2009-03-09 - 2020-02-19)	649	SMA = 59	Threshold = 80
Post-COVID Bull Market (2020-03-23 - 2022-01-03)	68	SMA = 5	Threshold = 50

Total performance:

Strategy / Market condition	Momentum	Mean Reversion
Dot-Com Bubble Bull Market (1995-01-04 - 2000-09-10)	25680.91	20057.65
Post-Global Financial Crisis Bull Market (2009-03-09 - 2020-02-19)	30804.75	16358.86
Post-COVID Bull Market (2020-03-23 - 2022-01-03)	15584.03	14431.85

Outperformance:

Strategy / Market condition	Momentum	Mean Reversion
Dot-Com Bubble Bull Market (1995-01-04 - 2000-09-10)	54.85	-11917.36
Post-Global Financial Crisis Bull Market (2009-03-09 - 2020-02-19)	0.0	-19554.36
Post-COVID Bull Market (2020-03-23 - 2022-01-03)	-125.74	-4441.36

For bull markets,

- In the Total performance table, in all 3 bull market periods, both strategies show profitability, but momentum strategy outperforms mean reversion strategy in general.
- In the Outperformance table, in all 3 bull market periods, momentum strategy has similar performance compared to asset return in general, while mean reversion strategy has significant underperformance compared to asset return.

6.3.1.1. Potential reasons why momentum strategy has similar performance to asset return in bull market:

1. Trend persistence in bull markets
    - Momentum strategy thrives in environments where price trends persist, which is a defining feature of bull markets. Bull markets are characterized by sustained upward price movements driven by investor optimism, economic recovery, or technological innovation (Jegadeesh & Titman, 1993). These trends align directly with the core mechanism of momentum strategy, allowing them to capture the continuation of price increases similar to asset return.
2. Reduced price reversals
    - Momentum strategy is more vulnerable in environments where prices frequently revert to their historical means because such environments deviate strategy core assumption that recent

price trends will continue in the same direction. However, during bull markets, price reversals are relatively less common due to the dominance of upward sentiment and sustained investor optimism (Daniel et al., 1998). Therefore, less losses occurred from reversing price trends, enabling momentum strategy to perform in line with market return.

6.3.1.2. Potential reasons why mean reversion strategy has significant underperformance in bull market:

1. Lack of price reversals
    - The persistence that bull markets are characterized by sustained upward trends in asset prices leads to price continuation rather than reversion, which directly undermines the core principle of mean reversion strategies (De Bondt & Thaler, 1985). Assets that outperform early in the bull market cycle typically continue to outperform as winners. Mean reversion strategy which short these winners experience losses when prices rise further in the bull run.
2. Investors behavioral biases and sentiment
    - In bull markets, investors tend to be overconfident and have the behavioral bias of buying because of "fear of missing out". Investor psychology in bull markets reduces the likelihood of price corrections (Barberis et al., 1998), making market environment unfavorable for mean reversion strategy. Investors tend to overreact to positive news and winners, driving leading asset prices even higher, causing potential loss if executing mean reversion strategy.

6.3.2 Bear markets analysis

Optimal parameters:

Strategy / Market condition	Momentum	Mean Reversion	
Dot-Com Crash Bear Market (2000-09-10 - 2002-10-09)	13	SMA = 64	Threshold = 170
Global Financial Crisis Bear Market (2007-10-09 - 2009-03-09)	153	SMA = 2	Threshold = 1
COVID-19 Crash Bear Market (2020-02-20 - 2020-03-23)	9	SMA = 2	Threshold = 56
Inflation and Rate-Hike Bear Market (2022-01-04 - 2022-10-12)	5	SMA = 13	Threshold = 199

Total performance:

Strategy / Market condition	Momentum	Mean Reversion
Dot-Com Crash Bear Market (2000-09-10 - 2002-10-09)	19316.47	14390.35
Global Financial Crisis Bear Market (2007-10-09 - 2009-03-09)	21087.46	18766.63
COVID-19 Crash Bear Market (2020-02-20 - 2020-03-23)	16345.31	11826.55
Inflation and Rate-Hike Bear Market (2022-01-04 - 2022-10-12)	12592.87	12593.85

Outperformance:

Strategy / Market condition	Momentum	Mean Reversion
Dot-Com Crash Bear Market (2000-09-10 - 2002-10-09)	13989.74	8718.59
Global Financial Crisis Bear Market (2007-10-09 - 2009-03-09)	16345.31	14436.75
COVID-19 Crash Bear Market (2020-02-20 - 2020-03-23)	6842.02	5123.23
Inflation and Rate-Hike Bear Market (2022-01-04 - 2022-10-12)	5003.27	4460.43

For bear markets,

- In the Total performance table, in all 4 bear market periods, both strategies show profitability, but same with the bull market, momentum strategy outperforms mean reversion strategy in general.
- In the Outperformance table, in all 4 bear market periods, both momentum strategy and mean reversion strategy outperform benchmark (asset return).

6.3.2.1. Potential reasons why momentum strategy outperform mean reversion strategy and asset return in bear market:

1. Trend persistence in bear markets
    - Bear markets are characterized by sustained downward trends driven by prolonged negative sentiment and deteriorating economic fundamentals (Jegadeesh & Titman, 1993). Momentum strategy thrives in such market environments by identifying and exploiting continuation of downward price patterns. In contrast, mean reversion strategy may be less profitable due to

the occasional failure of price reversion to historical averages during prolonged downward trends (Asness, Moskowitz, & Pedersen, 2013). The persistence of bearish trends provides a structural advantage to momentum strategy.

2. Increased volatility in bear market strengthens momentum signals
    - Bear markets are typically associated with heightened levels of volatility, which amplify price trends and strengthen momentum signals (Daniel, Jagannathan, & Kim, 2012). The increased volatility allows momentum strategy to grasp trends that are more pronounced and predictable. In contrast, mean reversion strategy may underperform momentum strategy slightly in such volatile market conditions, as the probability of fast price reversion during sharper price movements is lower (De Bondt & Thaler, 1985).
3. Different suitability in different time horizons
    - The inherent nature of momentum strategy is to operate over medium-to longer-term time horizons, aligning well with the prolonged nature of bear markets (Jegadeesh & Titman, 1993). On the contrary, the inherent nature of mean reversion strategy relies more heavily on shorter-term price corrections, which may be less profitable during continuous and steady market down trends (De Bondt & Thaler, 1985). The mismatch of time horizon potentially limits the efficacy and profitability of mean reversion strategy in bear markets.

6.3.2.2. Potential reasons why mean reversion strategy outperform asset return in bear market:

1. Short-term rebounds during oversold periods
    - During bear markets, a phenomenon named "dead cat bounces" often occurs when assets experience sharp sell-offs followed by short-term rebounds (Lo & MacKinlay, 1990), partly due to investors' panic emotions and overreaction, in which corrections typically occur later after the excessive short selling pressure. The inherent nature of mean reversion strategy helps exploiting such rebounds by identifying oversold assets that have higher probability to revert toward their mean value. Compared to the overall downward trend of asset return, the ability to profit from temporary price corrections contribute to the outperformance success of mean reversion strategy.
2. Lower benchmark performance during bear markets
    - During bear markets, the overall benchmark typically suffers significant losses, creating a low baseline for outperformance (Chen et al., 1986). During these market downtrends, mean reversion strategy captures shorter-term gains from price corrections and also helps reduce negative exposure during prolonged downward trends (Levy, 1967). Modest gains or smaller losses can thus be achieved by mean reversion strategy, which contributes to the success of outperformance compared to benchmark asset return. Such relative advantage of mean

reversion strategy becomes more pronounced during severe market decline periods with negative benchmark returns, creating an easier outperformance environment (Campbell et al., 1997).

7. Conclusion

This research evaluated the performance of 3 classic algorithmic trading strategies — Simple Moving Average (SMA), Momentum, and Mean Reversion — across various timeframes and market conditions using rigorous backtesting and parameter optimization.

Momentum strategy is proved to be the most robust and adaptable, excelling in trending markets, particularly during bear markets, where sustained downward trends and heightened volatility amplified its profitability. SMA strategies, while occasionally profitable potentially, seem to be more sensitive to parameter changes, reducing its standalone reliability. Mean Reversion tends to be unprofitable and has losses in bull markets due to the absence of more frequent price corrections, but it performed quite well in bear markets compared to benchmark by capitalizing on shorter-term price reversals.

Time frame selection can potentially influence strategy performance significantly, with optimal parameters balancing noise reduction and responsiveness. Momentum strategy in shorter time frames requires longer rolling windows to smooth out volatility and noise, while longer time frames tend to have shorter rolling windows to capture sustained trends. Mean reversion strategy follows a similar pattern, with shorter time frames requiring larger SMAs to stabilize noisy data, while longer time frames rely on smaller SMAs due to their smoother price trajectories. Threshold values remain consistent across timeframes in general, reflecting strategy's primary focus on detecting price deviations.

Further research can be done by incorporating optimal leverage and advanced risk management techniques, such as position sizing and drawdown controls, to enhance returns while mitigating risks. Exploration of hybrid strategies which combine Momentum and Mean Reversion strategy may potentially balance trend-following and counter-trend dynamics, improving overall profitability and robustness. Expanding the analysis to higher frequency data (e.g. 1 second, 1 min) may generate different insights to intraday trading. Additionally, extending the research to slightly more diverse asset classes, such as cryptocurrencies and fixed income, helps increase strategy generalizability.

Ultimately, the integration of other advanced techniques, hybrid strategies, and broader asset coverage could potentially pave the way for more robust, adaptable, and consistently profitable algorithmic trading frameworks in rapidly evolving market conditions nowadays.

8. References

Asness, C. S., Moskowitz, T. J., & Pedersen, L. H. (2013). Value and momentum everywhere. *The Journal of Finance*, *68*(3), 929–985. https://doi.org/10.1111/jofi.12021

Baker, M., & Wurgler, J. (2013). Behavioral Corporate Finance: an updated survey. In *Handbook of the economics of finance* (pp. 357–424). https://doi.org/10.1016/b978-0-44-453594-8.00005-7

Baker, S., Bloom, N., Davis, S., & Terry, S. (2020). *COVID-Induced economic uncertainty*. https://doi.org/10.3386/w26983

Barroso, P., & Santa-Clara, P. (2014). Momentum has its moments. *Journal of Financial Economics*, *116*(1), 111–120. https://doi.org/10.1016/j.jfineco.2014.11.010

Bekaert, G., Hoerova, M., & Lo Duca, M. (2013). Risk, uncertainty and monetary policy. *Journal of Monetary Economics*, *60*(7), 771–788. https://doi.org/10.1016/j.jmoneco.2013.06.003

Brock, W., Lakonishok, J., & LeBARON, B. (1992a). Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, *47*(5), 1731–1764. https://doi.org/10.1111/j.1540-6261.1992.tb04681.x

Brock, W., Lakonishok, J., & LeBARON, B. (1992b). Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, *47*(5), 1731–1764. https://doi.org/10.1111/j.1540-6261.1992.tb04681.x

Campbell, J. Y., Lo, A. W., & MacKinlay, A. C. (1997). *The econometrics of financial markets*. Princeton University Press.

Campbell, J. Y., & Shiller, R. J. (1988). Stock prices, earnings, and expected dividends. *The Journal of Finance*, *43*(3), 661–676. https://doi.org/10.1111/j.1540-6261.1988.tb04598.x

Cheema-Fox, A., LaPerla, B. R., Serafeim, G., & Wang, H. (2020). Corporate resilience and response during COVID-19. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3578167

Chen, N., Roll, R., & Ross, S. A. (1986). Economic forces and the stock market. *The Journal of Business*, *59*(3), 383. https://doi.org/10.1086/296344

Daniel, K., David, H., & Avanidhar, S. (1998). Investor Psychology and Security Market Under- and Overreactions.&qout; Journal of Finance. *Journal of Finance*, *53*(6), 1839–1885. https://doi.org/10.1111/0022-1082.00077

Daniel, K., Hirshleifer, D., & Subrahmanyam, A. (1998a). Investor psychology and Security Market under‐ and overreactions. *The Journal of Finance*, *53*(6), 1839–1885. https://doi.org/10.1111/0022-1082.00077

Daniel, K., Hirshleifer, D., & Subrahmanyam, A. (1998b). Investor psychology and Security Market under‐ and overreactions. *The Journal of Finance*, *53*(6), 1839–1885. https://doi.org/10.1111/0022-1082.00077

Daniel, K., Jagannathan, R., & Kim, S. (2012). *Tail risk in momentum strategy returns*. https://doi.org/10.3386/w18169

De Bondt, W. F. M., & Thaler, R. (1985a). Does the stock market overreact? *The Journal of Finance*, *40*(3), 793–805. https://doi.org/10.1111/j.1540-6261.1985.tb05004.x

De Bondt, W. F. M., & Thaler, R. (1985b). Does the stock market overreact? *The Journal of Finance*, *40*(3), 793–805. https://doi.org/10.1111/j.1540-6261.1985.tb05004.x

Edenfield, K. A. (2003). 'Irrational Exuberance' Robert J. Shiller, Princeton University Press, Princeton, NJ, 2000, 296 pages, $35. *Journal of Banking & Finance*, *27*(4), 779–782. https://doi.org/10.1016/s0378-4266(02)00285-6

Erten, B., & Ocampo, J. A. (2013). Super cycles of commodity prices since the Mid-Nineteenth century. *World Development*, *44*, 14–30. https://doi.org/10.1016/j.worlddev.2012.11.013

Gorton, G. B., & Metrick, A. (2012). Getting up to speed on the financial crisis: A One-Weekend-Reader's guide. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1974662

Hamilton, J. D. (1983). Oil and the Macroeconomy since World War II. *Journal of Political Economy*, *91*(2), 228–248. https://doi.org/10.1086/261140

Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011a). Does algorithmic trading improve liquidity? *The Journal of Finance*, *66*(1), 1–33. https://doi.org/10.1111/j.1540-6261.2010.01624.x

Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011b). Does algorithmic trading improve liquidity? *The Journal of Finance*, *66*(1), 1–33. https://doi.org/10.1111/j.1540-6261.2010.01624.x

Hilpisch, Y. (2020). *Python for algorithmic trading*. O'Reilly Media.

Hoshi, T., & Kashyap, A. (1999). *The Japanese banking crisis: Where did it come from and how will it end?* https://doi.org/10.3386/w7250

Jegadeesh, N., & Titman, S. (1993a). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x

Jegadeesh, N., & Titman, S. (1993b). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x

Jegadeesh, N., & Titman, S. (1993c). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x

Joyce, M., Miles, D., Scott, A., & Vayanos, D. (2012). Quantitative Easing and Unconventional Monetary Policy – An Introduction. *The Economic Journal*, *122*(564), F271–F288. https://doi.org/10.1111/j.1468-0297.2012.02551.x

Levy, R. A. (1967a). RELATIVE STRENGTH AS a CRITERION FOR INVESTMENT SELECTION. *The Journal of Finance*, *22*(4), 595–610. https://doi.org/10.1111/j.1540-6261.1967.tb00295.x

Levy, R. A. (1967b). RELATIVE STRENGTH AS a CRITERION FOR INVESTMENT SELECTION. *The Journal of Finance*, *22*(4), 595–610. https://doi.org/10.1111/j.1540-6261.1967.tb00295.x

Lo, A. W., & MacKinlay, A. C. (1990). When are contrarian profits due to stock market overreaction? *Review of Financial Studies*, *3*(2), 175–205. https://doi.org/10.1093/rfs/3.2.175

Lo, A. W., Mamaysky, H., & Wang, J. (2000). Foundations of technical analysis: computational algorithms, statistical inference, and empirical implementation. *The Journal of Finance*, *55*(4), 1705–1765. https://doi.org/10.1111/0022-1082.00265

Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2011). Time series momentum. *Journal of Financial Economics*, *104*(2), 228–250. https://doi.org/10.1016/j.jfineco.2011.11.003

Neely, C. J., Rapach, D. E., Tu, J., & Zhou, G. (2014). Forecasting the equity risk premium: The role of technical indicators. *Management Science*, *60*(7), 1772–1791. https://doi.org/10.1287/mnsc.2013.1838

Nick, B., & Robert, K. (2012). Demystifying Time-Series Momentum Strategies: Volatility Estimators, Trading Rules and Pairwise Correlations. *Wiley*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2140091

Ofek, E., & Richardson, M. (2003a). DotCom Mania: The rise and fall of internet stock prices. *The Journal of Finance*, *58*(3), 1113–1137. https://doi.org/10.1111/1540-6261.00560

Ofek, E., & Richardson, M. (2003b). DotCom Mania: The rise and fall of internet stock prices. *The Journal of Finance*, *58*(3), 1113–1137. https://doi.org/10.1111/1540-6261.00560

Reinhart, C. M., & Rogoff, K. S. (2009). This time is different: eight centuries of financial folly. *Economics Books*. https://ideas.repec.org/b/pup/pbooks/8973.html

Rodrik, D., & Subramanian, A. (2004). *From "Hindu Growth" to Productivity Surge: The Mystery of the Indian Growth Transition*. https://doi.org/10.3386/w10376

Toye, J. (2021). The great demographic reversal: ageing societies, waning inequality and an inflation revival. *Contributions to Political Economy*, *40*(1), 91–94. https://doi.org/10.1093/cpe/bzab010