A REPORT

ON

STOCK PORTFOLIO CREATION USING MOVING AVERAGE CROSSOVER AND MEAN VARIANCE OPTIMISATION BASED ON TIME SERIES DATA

BY

Apoorva Gulati (2021B3A70934P)

Saksham Verma (2021A7PS2414P)

AT

WILLIAM O'NEIL INDIA

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June, 2023)

A REPORT

ON

STOCK PORTFOLIO CREATION USING MOVING AVERAGE CROSSOVER AND MEAN VARIANCE OPTIMISATION BASED ON TIME SERIES DATA

BY

Apoorva Gulati (2021B3A70934P)

M.Sc. Economics and B.E. Computer Science

and

Saksham Verma (2021A7PS2414P)

B.E. Computer Science

Prepared in partial fulfillment of the

Practice School-I Course Nos.

BITS F221/BITS F231/BITS F241

AT

WILLIAM O’NEIL INDIA

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June, 2023)

Acknowledgements

We would like to express our gratitude to the Practice School Division for giving us the opportunity to undertake the Practice School - I course. Your efforts in coordinating and organizing the course across all campuses and your commitment to providing practical exposure and bridging the gap between academia and industry are commendable. This course has taught us to apply our academic knowledge and gain practical insights into the industry.

Our Professor in Charge, Dr. Arfat Ahmad Sofi, has been an amazing support system. We would like to extend our gratitude to him. Your vast knowledge helped us students truly understand the aim of Practice School - I. Your guidance and support have been instrumental in shaping our understanding and progress during this period. We are grateful for the time you dedicated to assessing our progress, providing constructive feedback, and fostering a positive learning environment. A huge thanks to our Co-Instructor, Mr. Ankush Soni, for always being available for all our queries and supporting us through the learning process.

We would like to thank the PS - I station, William O'Neil India, and all its employees who have welcomed us into their workplace, shared their knowledge, and allowed us to contribute to their projects. We are grateful for their support, guidance, and willingness to impart their expertise, which has been crucial in enhancing our understanding of the practical aspects of our field. We would like to specially thank our team manager, Mr. Ravi Kongara, for being a great mentor throughout. You have always been approachable and supportive of all our doubts and questions.

Lastly, we would like to thank our fellow batchmates for their collaboration, camaraderie, and shared learning experience. The opportunity to work alongside such talented peers has been both inspiring and rewarding.

The experiences and skills we have gained will undoubtedly shape our future endeavors. Thank you once again for the invaluable opportunity and for contributing to our personal and professional growth.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

Practice School Division

Station: William O'Neil India

Duration: 46 days

Date of Start: 5th June 2023

Date of Submission: 28th June 2023

Title of the Project: STOCK PORTFOLIO CREATION USING MOVING AVERAGE CROSSOVER AND MEAN VARIANCE OPTIMISATION BASED ON TIME SERIES DATA

Student Details: 1. Apoorva Gulati
2021B3A70934P
M.Sc. Economics and B.E. Computer Science

2. Saksham Verma
2021A7PS2414P
B.E. Computer Science

Name(s) and designation(s) of the experts:

1. Mr. Ravi Theja Kongara (Financial Risk Manager)
2. Mr. Pramod Gopal (IT Head)
3. Ms. Akhila Arjunam (HR Manager)

Name of the PS Faculty: Prof. Arfat Ahmad Sofi

Key Words: stock portfolio, moving average crossover, mean variance optimization, Backtrader, CVXOPT, risk analysis, historical performance, asset allocation.

Abstract:

This report aims to provide an overview of my learning experiences as part of the Practice School - I programme at William O'Neil India. It outlines the knowledge gained about technical indicators used in algorithmic trading strategies.

The study analyzes a 10-year time series data set of 25 stock prices in CSV format, utilizing Backtrader and CVXOPT. The objective is to construct an efficient portfolio that maximizes returns and minimizes risks. Moving average crossover identifies buy/sell signals, while mean variance optimization optimizes asset allocation based on expected returns and risks.

Saksham

[illegible]

Signature of Students | Signature of PS Faculty

Date: June 28, 2023 | Date:

Table of Contents

STOCK PORTFOLIO CREATION USING MOVING AVERAGE CROSSOVER AND MEAN VARIANCE OPTIMISATION BASED ON TIME SERIES DATA

Acknowledgements 3
Introduction 7
Concepts used in the project 8
Moving Average Crossover
Mean Variance Optimization
Backtrader
Code 9
Results 17
Further Work 23
Conclusions 24
Glossary 24

Introduction

This report presents an analysis of stock portfolio creation using moving average crossover and mean variance optimization techniques based on time series data. The study utilizes a dataset consisting of 25 stock prices over a period of 10 years, provided in a CSV format. The analysis involves employing two key tools: Backtrader, a popular open-source trading framework, and CVXOPT, a library for convex optimization.

The objective of this study is to construct an efficient stock portfolio that maximizes returns while minimizing risks. To achieve this, we explore the application of moving average crossover and mean variance optimization methods. Moving average crossover is a commonly used technical analysis technique that helps identify potential buy or sell signals by comparing short-term and long-term moving averages. Mean variance optimization, on the other hand, is a mathematical approach that aims to find the optimal asset allocation by considering the trade-off between expected returns and risks.

Throughout this report, we delve into the implementation of these techniques using the provided time series data. We analyze the historical stock prices and employ backtesting strategies using Backtrader to evaluate the performance of different portfolio compositions. Additionally, we utilize CVXOPT to optimize the allocation of assets within the portfolio based on mean variance optimization principles.

The findings of this report provide insights into the effectiveness of moving average crossover and mean variance optimization methods in constructing a robust and efficient stock portfolio. The analysis takes into account both historical performance and risk factors, offering valuable information for investors seeking to make informed decisions.

<u>Concepts used in the project</u>

1. **Moving Average Crossover:** Moving average crossover is a popular technical analysis technique used in financial markets to generate trading signals. It involves comparing different moving averages of price data, such as a shorter-term and longer-term moving average. When the shorter-term moving average crosses above the longer-term moving average, it indicates a potential buy signal, while a crossover below suggests a potential sell signal. This approach helps traders confirm trends, determine entry and exit points, and manage risk by smoothing out price fluctuations and capturing underlying market trends.

2. **Mean Variance Optimisation:** Mean variance optimization is a widely used technique in finance for constructing efficient investment portfolios. By analyzing the covariance matrix of asset returns, it helps investors find the optimal allocation of assets that maximizes expected returns for a given level of risk or minimizes risk for a given level of expected returns. This technique allows for diversification across assets with different risk-return characteristics, aiding in the management of portfolio risk and potential enhancement of returns. The expected return of the portfolio is calculated as the weighted sum of the expected returns of individual assets. The portfolio variance is determined by considering the covariance between assets and their respective weights. MVO involves solving an optimization problem with constraints on the sum of weights and a target expected return. CVXOPT, a python library, is used to achieve this via code and assign optimized weights to the stocks.

3. **Backtrader:** Backtrader is a trading framework that facilitates the development and backtesting of trading strategies. It provides a comprehensive set of tools and functionalities for implementing and evaluating trading ideas using historical market data. With Backtrader, traders can define their trading rules and indicators, simulate trades, and assess the performance of their strategies based on historical data. It supports various asset classes and trading instruments, making it versatile for different markets. Backtrader also offers features like portfolio management, risk assessment, and performance analysis, allowing traders to gain insights into the profitability and risk profile of their strategies.

Code

The code implements a backtesting framework for a mean-variance portfolio optimization strategy. It preprocesses stock price data and selects stocks based on the condition that the closing price is greater than the 200-day simple moving average (SMA). It defines a strategy class (MyStrategy) with methods for filtering stocks, performing mean-variance optimization, and executing buy/sell orders. The next method is called for each trading bar, updating the current month and executing orders if the closing price exceeds the 200-day SMA. The code also sets up the backtesting environment and includes functionalities for plotting results and generating performance reports using the quantstats library.

```
import backtrader as bt
import backtrader.analyzers as btanalyzers
import matplotlib
import matplotlib.pyplot as plt
import yfinance as yf
import pandas as pd
import numpy as np
from cvxopt import matrix, solvers
from datetime import datetime
import quantstats as qstats
%matplotlib inline
```

These lines import the required libraries and modules for the code, including Backtrader for backtesting, Pandas for data manipulation, NumPy for numerical operations, Matplotlib for plotting, CVXOPT for optimization, datetime for working with dates, and QuantStats for performance analysis.

```
df = pd.read_csv('prices.csv')
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
```

These lines read a CSV file named 'prices.csv' into a Pandas DataFrame and convert the 'date' column to a datetime format.

```
stock_start_date = df['date'].min()
```

```
stock_end_date = df['date'].max()
```

These lines find the minimum and maximum dates in the 'date' column of the DataFrame and assign them to variables stock_start_date and stock_end_date, respectively.

```
starting_dict = {}
tickers = df['ticker'].unique()
dates = pd.date_range(start=stock_start_date,
end=stock_end_date, freq='D')
index = pd.MultiIndex.from_product([tickers, dates],
names=['ticker', 'date'])
dummy_df = pd.DataFrame(index=index).reset_index()
```

These lines create an empty dictionary starting_dict to hold the starting dates for all stocks. They also create a dummy_df DataFrame that contains all combinations of tickers and dates, using the MultiIndex to set the ticker and date as index levels.

```
for ticker in tickers:
    starting_dict[ticker] = df[df['ticker'] ==
ticker]['date'].min()

for ticker in tickers:
    starting_dict[ticker] = starting_dict[ticker] +
pd.Timedelta(days=200)
```

These lines populate the starting_dict dictionary with the starting date of each stock. The first loop finds the minimum date for each ticker in the original DataFrame and assigns it to the corresponding key in the dictionary. The second loop adds 200 days to each starting date.

```
merged_df = pd.merge(dummy_df, df, on=['ticker', 'date'],
how='left')
merged_df[['close', 'open', 'high', 'low']] =
merged_df.groupby('ticker')[['close', 'open', 'high',
'low']].fillna(method='bfill')
```

```
df = merged_df
```

These lines merge the dummy_df DataFrame with the original DataFrame (df) on the 'ticker' and 'date' columns, filling missing values in the 'close', 'open', 'high', and 'low' columns with the next available value within each group of 'ticker'. The resulting DataFrame is then assigned back to df.

```
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
df = df.set_index('date')
```

These lines convert the 'date' column of the DataFrame to a datetime object (which is redundant since it was already converted earlier), and then set the 'date' column as the index of the DataFrame.

```
class MyStrategy(bt.Strategy):
    params = (
        ('sma_period', 200),
        ('lookback_period', 500),
        ('rebalance_day', 1),
        ('max_weight', 0.95),
        ('days', 5)
    )
```

This line defines the MyStrategy class, which is derived from the bt.Strategy class in the Backtrader library. The params tuple defines the default parameter values for the strategy. Each parameter represents a specific aspect of the strategy and can be customized when creating an instance of the strategy.

```
def __init__(self):
    self.first_trading_day = False
    self.current_month = None
    self.prev_month = None
    self.has_position = False
    self.start_date = starting_dict
    self.sma_200 = [bt.ind.SMA(data,
period=self.params.sma_period) for data in self.datas]
    self.selected_stocks = []
    self.selected_data = []
    self.rebalance_dates = set()  # Store rebalance dates
    self.weights = np.zeros(len(self.datas))
```

In the __init__ method, several attributes are initialized.

```
def filter_stocks(self):
    self.selected_data = []
    for i, d in enumerate(self.datas):
        if pd.Timestamp(d.datetime.date(0)) > 
self.start_date[d._name]:
            if self.datas[i].close[0] > self.sma_200[i][0]:
                self.selected_data.append(d._name)
            elif self.datas[i].close[0] < self.sma_200[i][0]:
                pass

    self.selected_stocks = [d for d in self.datas if d._name in 
self.selected_data]

    if len(self.selected_stocks) > 0:
        self.get_returns(self.selected_stocks)
    for stock in self.selected_stocks:
        first_day_of_month = 
stock.datetime.date().replace(day=1)
        self.rebalance_dates.add(first_day_of_month)
```

The filter_stocks method in the MyStrategy class filters stocks based on specific conditions. It iterates over each data feed, checks if the current bar's date is greater than the start date for that data feed, and compares the current closing price with the corresponding SMA value. If the closing price is greater, the data feed is added to the selected stocks list (self.selected_stocks), and the first day of each month for the selected stocks is added to the set of rebalance dates (self.rebalance_dates).

```
def mean_variance_optimization(self, returns):
    n = returns.shape[1]
    Sigma = np.cov(returns.T)
    if Sigma.shape == ():
        Sigma = np.array([[Sigma]])
    P = matrix(Sigma)
    q = matrix(np.zeros((n, 1)))
    G = matrix(-np.eye(n))
    h = matrix(np.zeros((n, 1)))
    A = matrix(np.ones((1, n)))
    b = matrix(np.array([1.]))
    solvers.options['show_progress'] = False
    sol = solvers.qp(P, q, G, h, A, b)
    return np.array(sol['x']).flatten()
```

The mean_variance_optimization method takes a matrix of returns as input and performs mean-variance optimization to find the optimal weights for the selected stocks. It calculates the covariance matrix (Sigma) of the returns and constructs matrices (P, q, G, h, A, b) required for the quadratic programming problem. The solvers.qp function from the cvxopt library is used to solve the optimization problem, and the resulting weights are returned.

```
def get_returns(self, selected_stocks):
    if len(self.datas[0]) < self.params.lookback_period:
        self.weights = np.zeros(len(self.selected_data))
        return

    n_portfolios = len(self.selected_data)
    if n_portfolios == 0:
        return
    if n_portfolios == 1:
        self.weights = [1]
        return

    prices = np.zeros((self.params.lookback_period,
len(selected_stocks)))
    returns = np.zeros((self.params.lookback_period,
len(selected_stocks)))

    for i, d in enumerate(selected_stocks):
        prices[:, i] =
d.close.get(size=self.params.lookback_period)
        returns[1:, i] = np.diff(prices[:, i]) / prices[:-1, i]

    self.weights = self.mean_variance_optimization(returns)

    self.place_order(selected_stocks)
```

The get_returns method in the MyStrategy class calculates the returns for the selected stocks. It first checks if the lookback period is greater than the length of the first data feed. If not, it initializes arrays to store prices and returns for the selected stocks. It then iterates over the selected stocks, retrieves their closing prices over the lookback period, and calculates the corresponding returns. The method calls the mean_variance_optimization method with the calculated returns and assigns the resulting weights to self.weights. Finally, it calls the place_order method with the selected stocks as an argument.

```
def place_order(self, selected_stocks):
    weights = np.array(self.weights) * self.params.max_weight

    # Selling the stocks not selected
    for i, d in enumerate(self.datas):
        if d not in selected_stocks:
            self.close(data=d)
    for i, d in enumerate(selected_stocks):
        self.order_target_percent(data=d, target=weights[i])
    self.weights -= weights
```

The place_order method in the MyStrategy class executes buy and sell orders for the selected stocks. It calculates the target weights by multiplying the normalized weights with the maximum allowed weight. It sells stocks that are not selected by calling self.close(data=d) for each non-selected data feed. Then, it buys the selected stocks by calling self.order_target_percent(data=d, target=weights[i]) with the corresponding target weight. It handles phased buying of stocks due to an upper limit on the weights by subtracting the applied weights from self.weights.

```
def residual_buy(self):
    weights = np.array(self.weights) * self.params.max_weight
    for i, d in enumerate(self.selected_stocks):
        current_value = self.broker.get_value([d])
        additional_value = weights[i] * self.broker.get_value()
        target_value = current_value + additional_value
        self.order_target_value(data=d, target=target_value)
    self.weights -= weights
```

The residual_buy method in the MyStrategy class executes residual buy orders for the selected stocks. It calculates the target weights by multiplying the weights with the maximum allowed weight. Then, for each selected stock, it calculates the current value and additional value to be invested. The target value is obtained by adding the current value and additional value. Finally, it executes the buy orders using self.order_target_value(data=d, target=target_value) and subtracts the applied weights from self.weights

```
def next(self):
    self.current_month = self.data.datetime.date().month
    if self.prev_month is None or self.current_month !=
self.prev_month:
        self.first_trading_day = True
    else:
        self.first_trading_day = False
    if self.first_trading_day:
        self.timer = self.params.days
        self.filter_stocks()

    if self.timer > 0 and self.first_trading_day == False:
        self.residual_buy()
        self.timer -= 1
    self.prev_month = self.current_month
```

The next method in the MyStrategy class is called for each new trading bar. It updates the current month based on the date of the current bar and determines if it's the first trading day of the month. It initializes a timer and calls the filter_stocks method if it's the first trading day. On subsequent trading days, if the timer is greater than 0, it executes the residual_buy method. The prev_month is updated with the current_month for future comparisons.

```
def stop(self):
    for i, d in enumerate(self.datas):
        self.close(data=d)
```

The stop method is called when the backtest is stopped or finished. It iterates over all data feeds (self.datas) and executes a close order for each one using self.close(data=d). This ensures that any remaining positions are closed before ending the backtest.

```
cerebro = bt.Cerebro()
cerebro.addstrategy(MyStrategy)
```

These lines create a Cerebro instance, which is the main object used to run the backtest. The strategy class MyStrategy is added to the Cerebro instance using the addstrategy method.

```
for ticker in df['ticker'].unique():
    stock_df = df[df['ticker'] == ticker]
    datafeed = bt.feeds.PandasData(dataname=stock_df, ...)
    cerebro.adddata(datafeed, name=ticker)
```

These lines loop over each unique ticker in the 'ticker' column of the DataFrame. For each ticker, a new DataFrame stock_df is created containing rows with only that ticker. Then, a PandasData data feed is created using the stock_df and added to the Cerebro instance with a name equal to the ticker.

```
cerebro.broker.setcash(100000.00)
cerebro.broker.setcommission(commission=0.001)
cerebro.addanalyzer(bt.analyzers.PyFolio, _name='pyfolio')
```

These lines set the initial cash amount for the broker, set the commission for trades, and add the PyFolio analyzer to the Cerebro instance. The PyFolio analyzer is used to generate performance statistics and plots using the pyfolio library.

```
strategy_run = cerebro.run()
print('Final Portfolio Value:', cerebro.broker.getvalue())
```

```
cerebro.plot(volume=False, iplot=False)
```

These lines run the backtest by calling the run method on the Cerebro instance, which executes the strategy. The result is stored in the strategy_run variable. The final portfolio value is printed, and the backtest plot is generated using the plot method of the Cerebro instance.

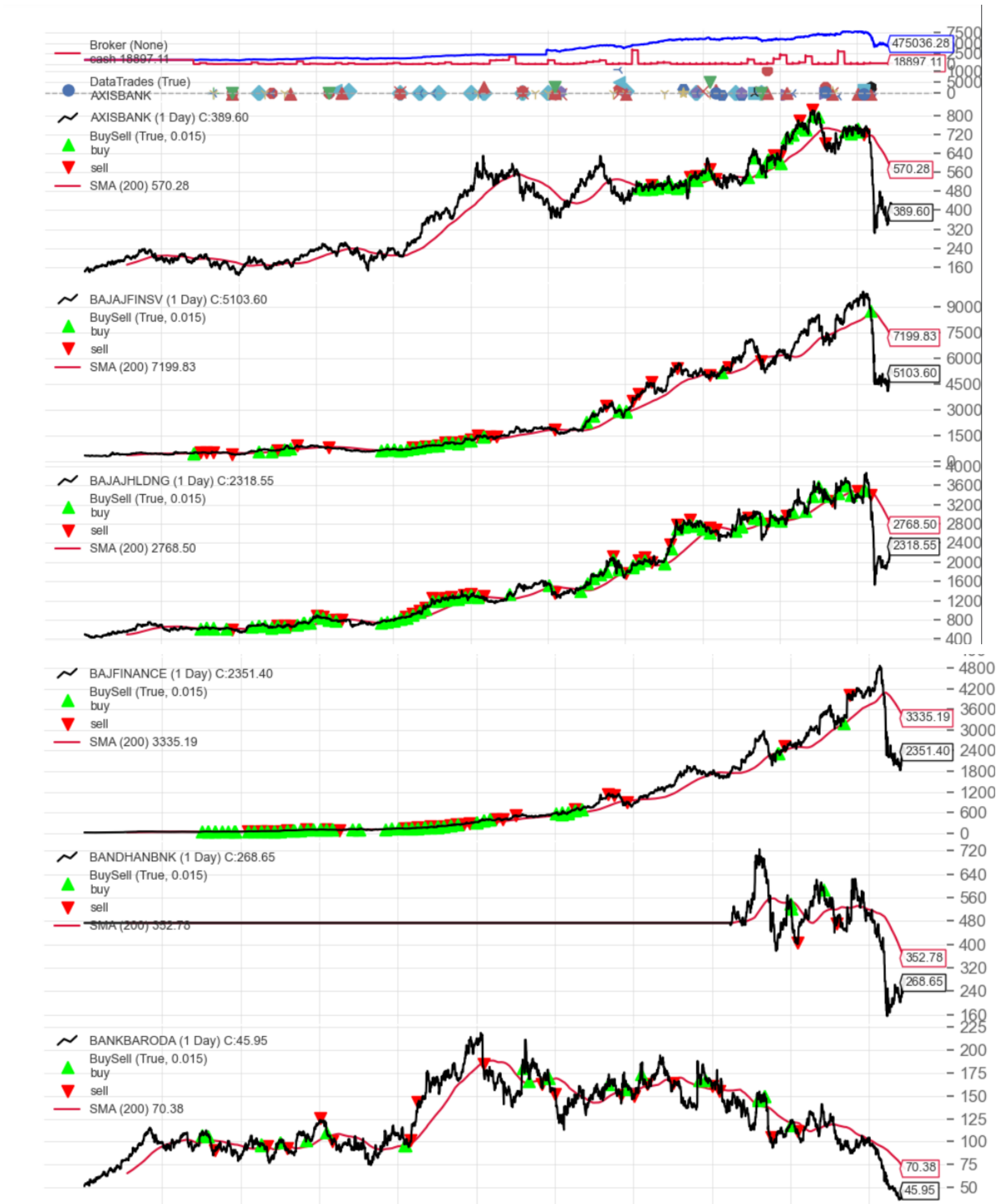
```
pyfolio_analyzer =
strategy_run[0].analyzers.getbyname('pyfolio')
returns, positions, transactions, gross_lev =
pyfolio_analyzer.get_pf_items()
returns_df = pd.DataFrame(returns)
returns_df.index = returns_df.index.tz_convert(None)
qs.reports.html(returns_df['return'], output='report.html')
qs.plots.snapshot(returns_df['return'], title='Strategy
Performance')
```

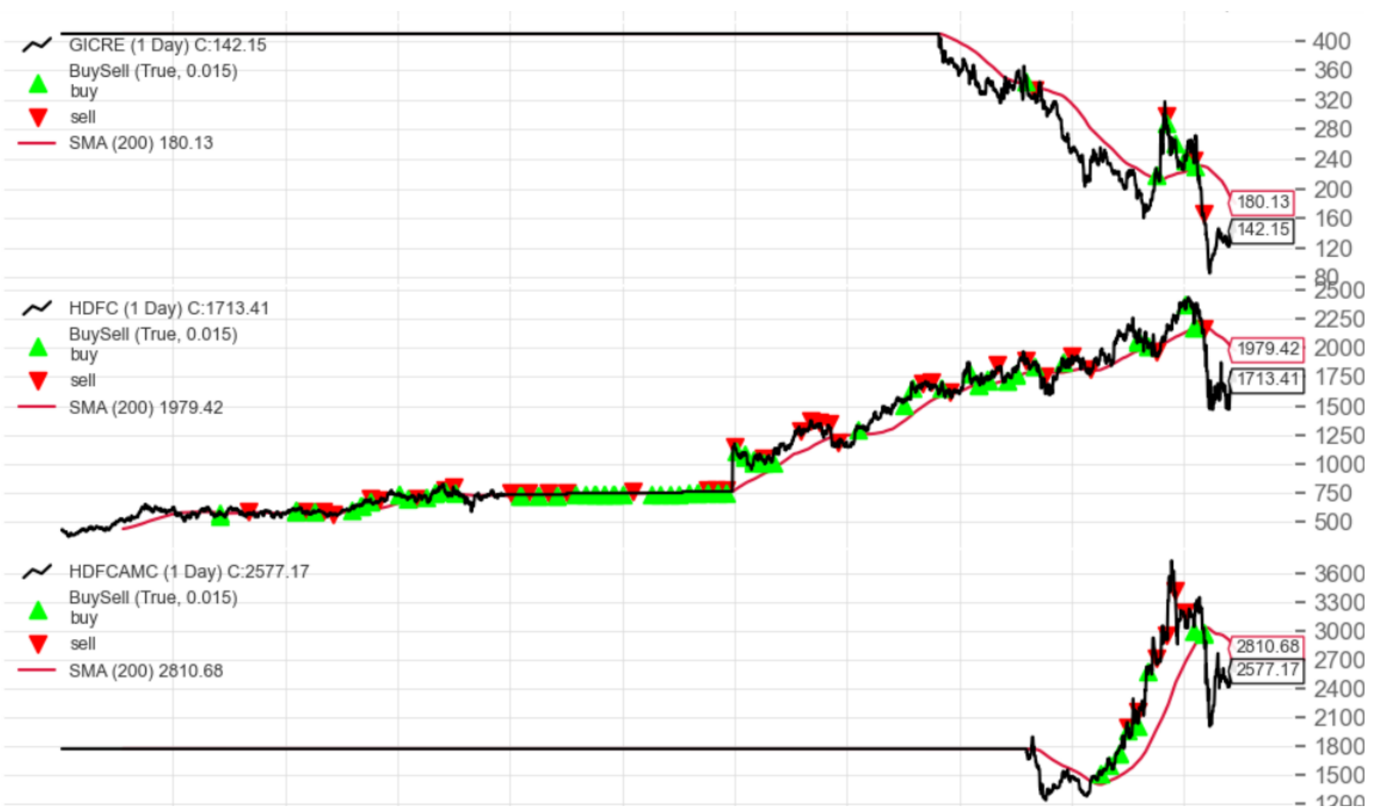
These lines retrieve the pyfolio analyzer from the first strategy run, extract the returns, positions, transactions, and gross leverage from the analyzer. The returns are converted to a DataFrame and the index is reformatted. Then, the quantstats library is used to generate an HTML performance report and plot the strategy performance snapshot.

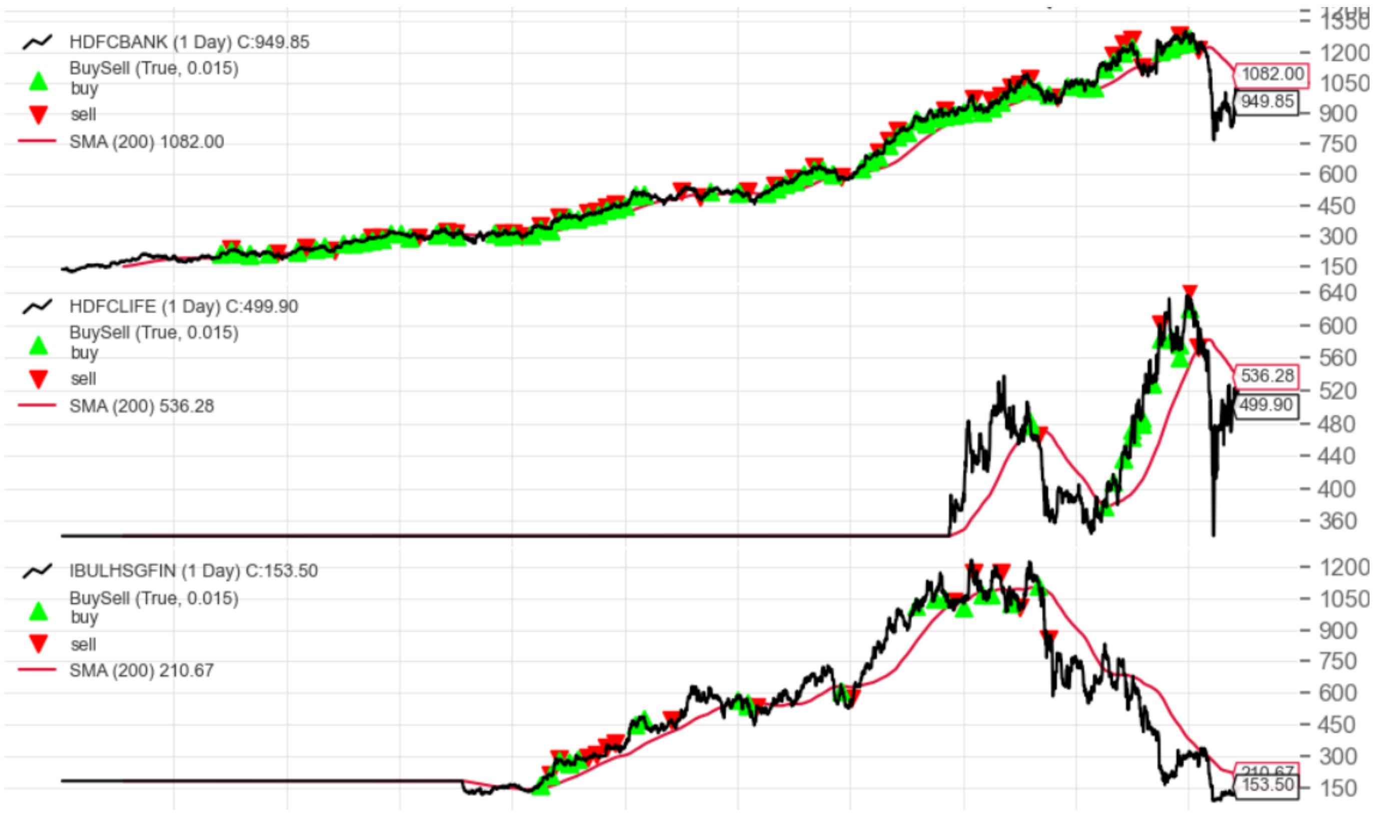
Results

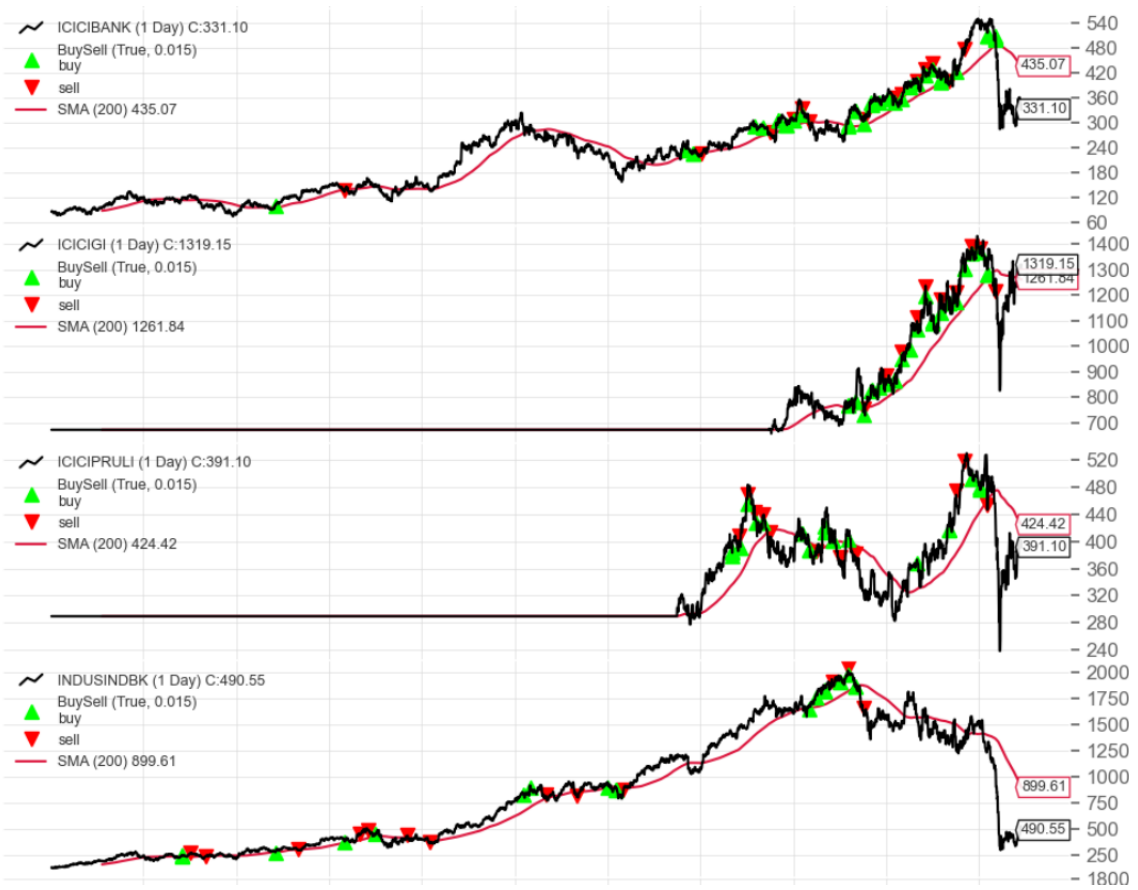
On running the code, we get a Final Portfolio Value of **475036.28** which is about **375% return** on our initial investment.

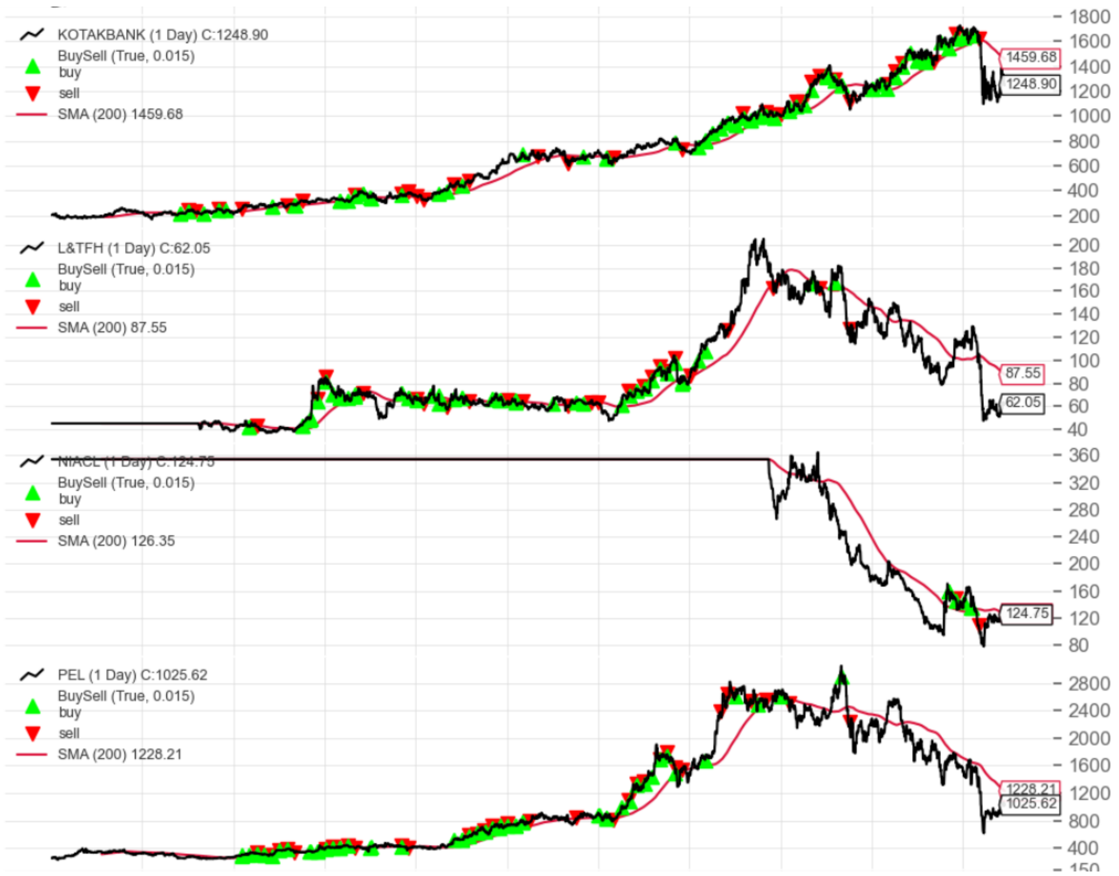
Below is the visualization of our portfolio along with the 25 original stocks, where green triangles indicate a buy signal and red triangles indicate a sell signal.

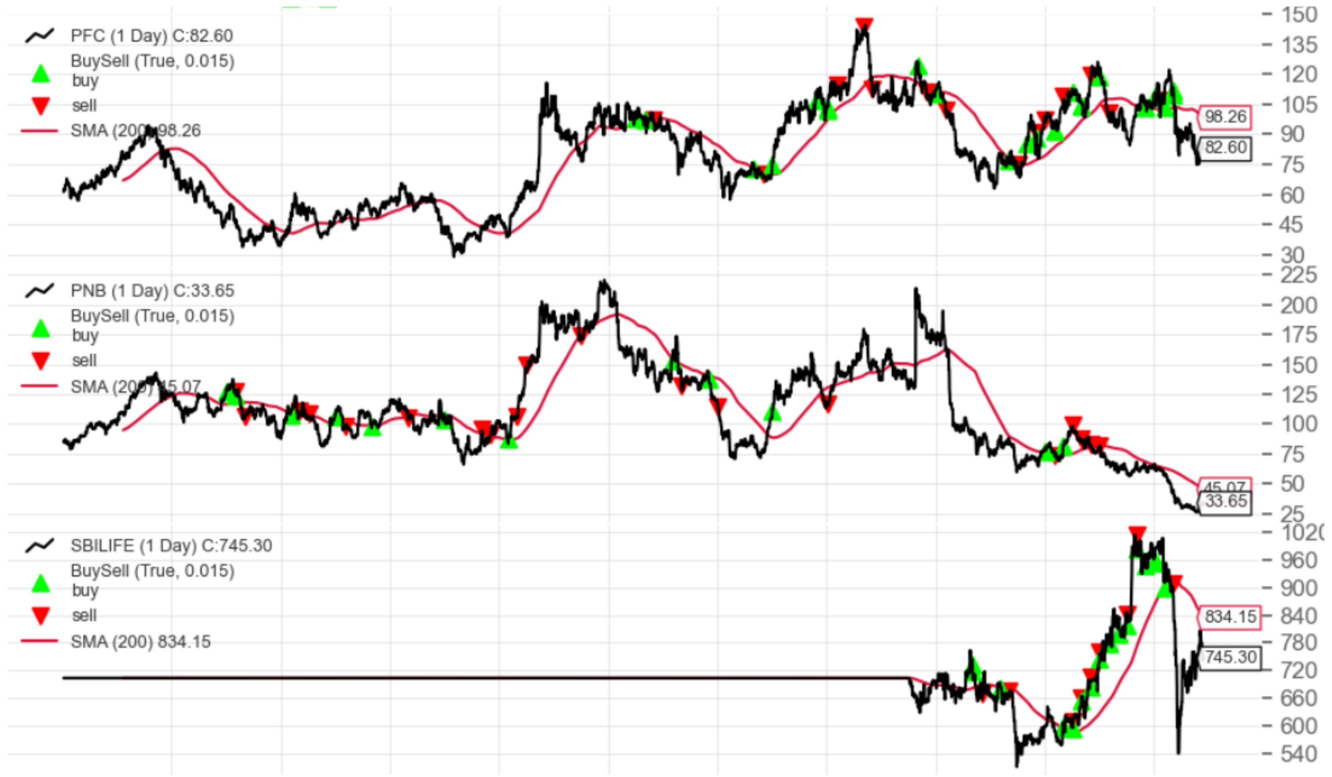

Broker (None)
cash 18897.11
475036.28
18897.11
DataTrades (True)
AXISBANK
AXISBANK (1 Day) C:389.60
BuySell (True, 0.015)
buy
sell
SMA (200) 570.28
570.28
389.60
BAJAJFINSV (1 Day) C:5103.60
BuySell (True, 0.015)
buy
sell
SMA (200) 7199.83
7199.83
5103.60
BAJAJHLDNG (1 Day) C:2318.55
BuySell (True, 0.015)
buy
sell
SMA (200) 2768.50
2768.50
2318.55
BAJFINANCE (1 Day) C:2351.40
BuySell (True, 0.015)
buy
sell
SMA (200) 3335.19
3335.19
2351.40
BANDHANBNK (1 Day) C:268.65
BuySell (True, 0.015)
buy
sell
SMA (200) 352.78
352.78
268.65
BANKBARODA (1 Day) C:45.95
BuySell (True, 0.015)
buy
sell
SMA (200) 70.38
70.38
45.95

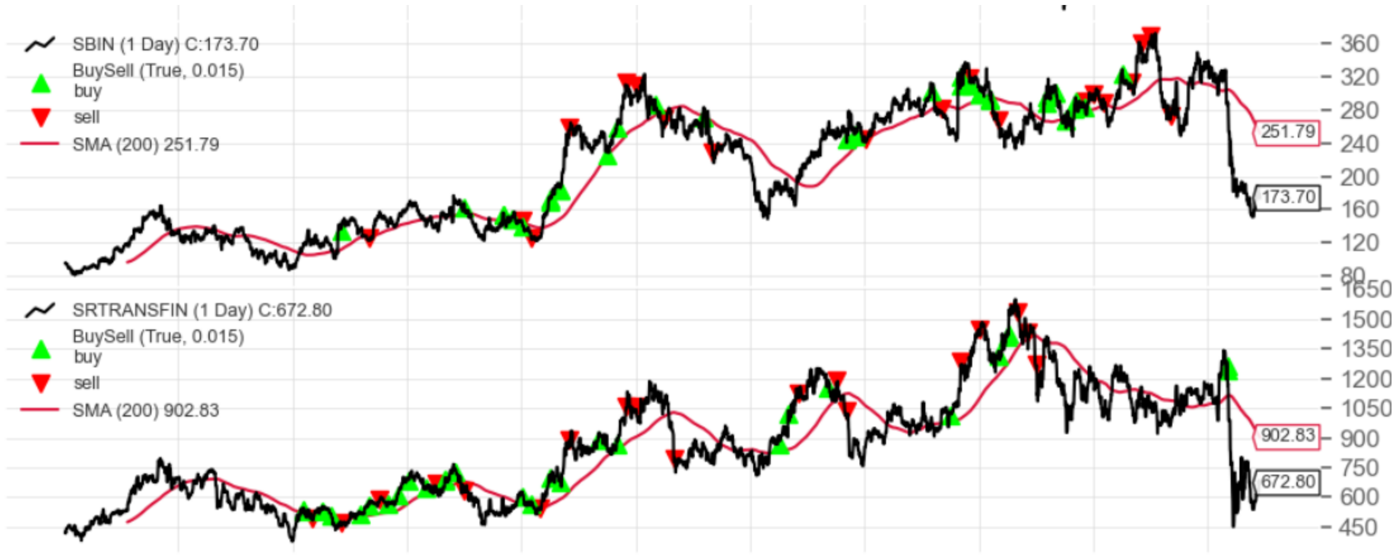

GICRE (1 Day) C:142.15
BuySell (True, 0.015)
buy
sell
SMA (200) 180.13
180.13
142.15
400
360
320
280
240
200
160
120
80
HDFC (1 Day) C:1713.41
BuySell (True, 0.015)
buy
sell
SMA (200) 1979.42
1979.42
1713.41
2500
2250
2000
1750
1500
1250
1000
750
500
HDFCAMC (1 Day) C:2577.17
BuySell (True, 0.015)
buy
sell
SMA (200) 2810.68
2810.68
2577.17
3600
3300
3000
2700
2400
2100
1800
1500
1200


HDFCBANK (1 Day) C:949.85
BuySell (True, 0.015)
buy
sell
SMA (200) 1082.00
1082.00
949.85
1350
1200
1050
900
750
600
450
300
150
HDFCLIFE (1 Day) C:499.90
BuySell (True, 0.015)
buy
sell
SMA (200) 536.28
536.28
499.90
640
600
560
520
480
440
400
360
IBULHSGFIN (1 Day) C:153.50
BuySell (True, 0.015)
buy
sell
SMA (200) 210.67
210.67
153.50
1200
1050
900
750
600
450
300
150


ICICIBANK (1 Day) C:331.10
BuySell (True, 0.015)
buy
sell
SMA (200) 435.07
435.07
331.10
540
480
420
360
300
240
180
120
60
ICICIGI (1 Day) C:1319.15
BuySell (True, 0.015)
buy
sell
SMA (200) 1261.84
1319.15
1261.84
1400
1300
1200
1100
1000
900
800
700
ICICIPRULI (1 Day) C:391.10
BuySell (True, 0.015)
buy
sell
SMA (200) 424.42
424.42
391.10
520
480
440
400
360
320
280
240
INDUSINDBK (1 Day) C:490.55
BuySell (True, 0.015)
buy
sell
SMA (200) 899.61
899.61
490.55
2000
1750
1500
1250
1000
750
500
250

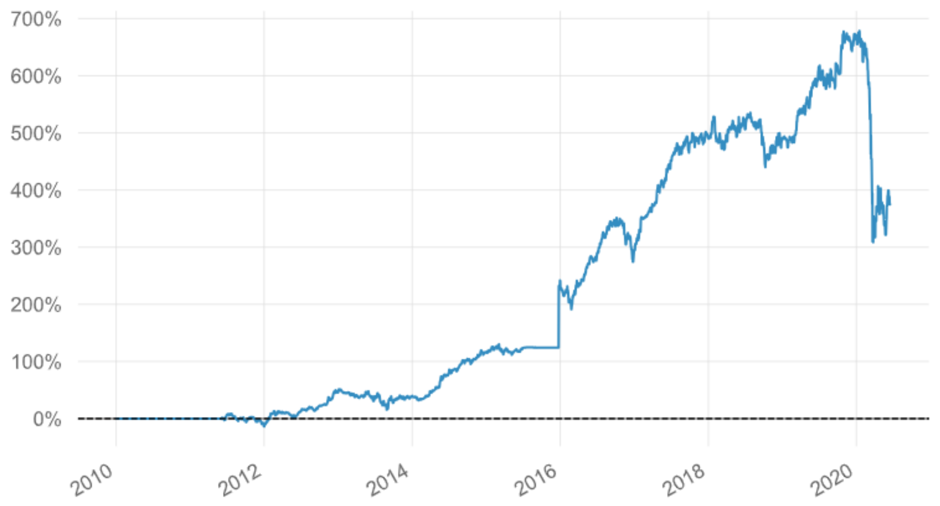

KOTAKBANK (1 Day) C:1248.90
BuySell (True, 0.015)
buy
sell
SMA (200) 1459.68
1459.68
1248.90
1800
1600
1400
1200
1000
800
600
400
200
L&TFH (1 Day) C:62.05
BuySell (True, 0.015)
buy
sell
SMA (200) 87.55
87.55
62.05
200
180
160
140
120
100
80
60
40
NIACL (1 Day) C:124.75
BuySell (True, 0.015)
buy
sell
SMA (200) 126.35
124.75
360
320
280
240
200
160
120
80
PEL (1 Day) C:1025.62
BuySell (True, 0.015)
buy
sell
SMA (200) 1228.21
1228.21
1025.62
2800
2400
2000
1600
1200
800
400



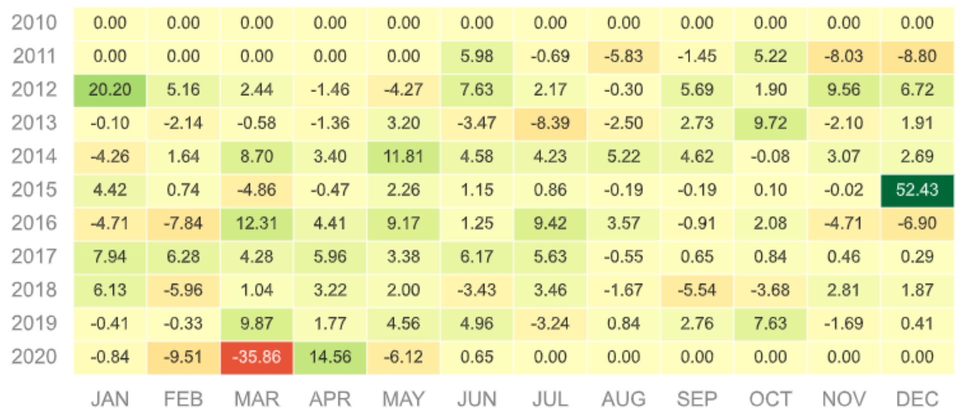


Below is the analysis of the strategy generated by QuantStats:

Cumulative Returns

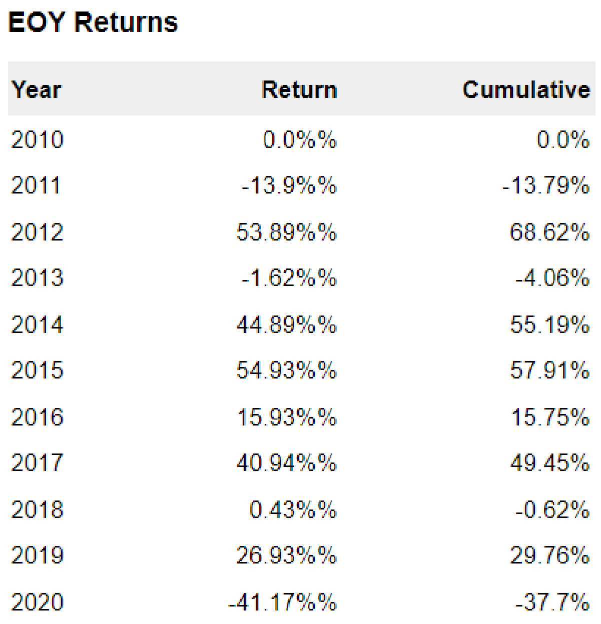


Monthly Returns (%)



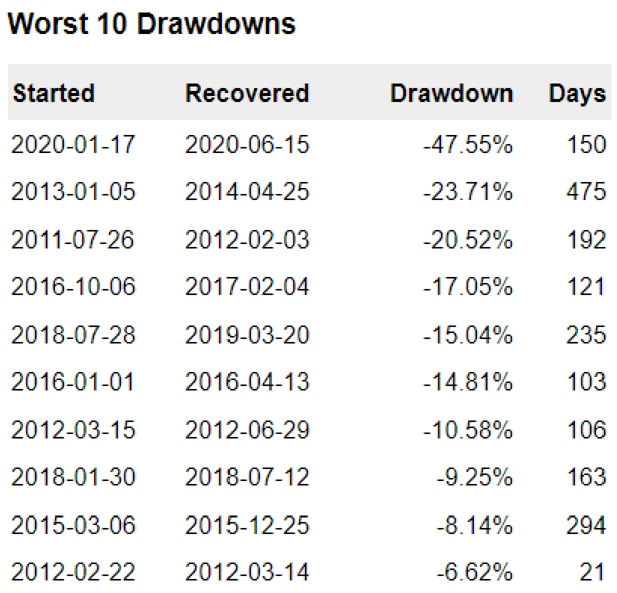
	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
2010	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2011	0.00	0.00	0.00	0.00	0.00	5.98	-0.69	-5.83	-1.45	5.22	-8.03	-8.80
2012	20.20	5.16	2.44	-1.46	-4.27	7.63	2.17	-0.30	5.69	1.90	9.56	6.72
2013	-0.10	-2.14	-0.58	-1.36	3.20	-3.47	-8.39	-2.50	2.73	9.72	-2.10	1.91
2014	-4.26	1.64	8.70	3.40	11.81	4.58	4.23	5.22	4.62	-0.08	3.07	2.69
2015	4.42	0.74	-4.86	-0.47	2.26	1.15	0.86	-0.19	-0.19	0.10	-0.02	52.43
2016	-4.71	-7.84	12.31	4.41	9.17	1.25	9.42	3.57	-0.91	2.08	-4.71	-6.90
2017	7.94	6.28	4.28	5.96	3.38	6.17	5.63	-0.55	0.65	0.84	0.46	0.29
2018	6.13	-5.96	1.04	3.22	2.00	-3.43	3.46	-1.67	-5.54	-3.68	2.81	1.87
2019	-0.41	-0.33	9.87	1.77	4.56	4.96	-3.24	0.84	2.76	7.63	-1.69	0.41
2020	-0.84	-9.51	-35.86	14.56	-6.12	0.65	0.00	0.00	0.00	0.00	0.00	0.00

Metric	Strategy
Risk-Free Rate	0.0%
Time in Market	61.0%
Max Drawdown	-47.55%
Longest DD Days	475
Volatility (ann.)	19.2%
Calmar	0.34
Skew	15.66
Kurtosis	673.92



EOY Returns

Year	Return	Cumulative
2010	0.0%%	0.0%
2011	-13.9%%	-13.79%
2012	53.89%%	68.62%
2013	-1.62%%	-4.06%
2014	44.89%%	55.19%
2015	54.93%%	57.91%
2016	15.93%%	15.75%
2017	40.94%%	49.45%
2018	0.43%%	-0.62%
2019	26.93%%	29.76%
2020	-41.17%%	-37.7%



Worst 10 Drawdowns

Started	Recovered	Drawdown	Days
2020-01-17	2020-06-15	-47.55%	150
2013-01-05	2014-04-25	-23.71%	475
2011-07-26	2012-02-03	-20.52%	192
2016-10-06	2017-02-04	-17.05%	121
2018-07-28	2019-03-20	-15.04%	235
2016-01-01	2016-04-13	-14.81%	103
2012-03-15	2012-06-29	-10.58%	106
2018-01-30	2018-07-12	-9.25%	163
2015-03-06	2015-12-25	-8.14%	294
2012-02-22	2012-03-14	-6.62%	21

It is evident from these charts that the portfolio performance declined sharply during the COVID-19 period, however the optimisation of weights has ensured that this is the best performing portfolio according to the risk-reward ratio.

Further Work:

We were expected to make a few modifications in the code such as changing the lookback period, changing the rebalance frequency, optimising the moving average parameter to check the effect of these changes on the portfolio returns. This was our project for the first month

For the second month, we have been given a new project in which we have to work on fixed income securities and yield curve inversion to generate signals in QQQ, S&P 500 funds and equity, commodities and currencies to generate an optimal portfolio using backtrader. This project is slightly more challenging and will require us to go through research papers on the topic.

Conclusion

In conclusion, this project allowed us to gain practical knowledge and skills in various aspects of stock portfolio creation and analysis. We learned how to use moving average crossover as a tool for identifying stocks with positive momentum. Additionally, we became familiar with mean variance optimization and its application in constructing optimal portfolios. By working with the Backtrader library, we gained hands-on experience in backtesting strategies and executing trades based on predefined rules. The project also introduced us to other important libraries such as Pandas, NumPy, and Matplotlib for data manipulation and visualization. Finally, we learned how to analyze portfolio performance using PyFolio and QuantStats, enabling us to evaluate key metrics and make informed investment decisions. Overall, this project provided us with valuable insights and enhanced our skills in technical analysis, portfolio optimization, backtesting, data manipulation, and performance analysis in the context of stock portfolio creation.

Glossary

Moving Average Crossover: A technical analysis technique that involves plotting two moving averages of different time periods on a price chart. The crossover of these moving averages is used as a signal to identify changes in trend direction.

Mean Variance Optimization (MVO): A mathematical framework used to construct an optimal portfolio by considering the trade-off between expected returns and portfolio risk. MVO aims to find the allocation of assets that maximizes expected returns for a given level of risk or minimizes risk for a given level of expected returns.

Backtrader: An open-source Python framework for backtesting trading strategies. It provides a flexible and efficient environment for simulating and evaluating the performance of trading strategies on historical data.

Pandas: A popular Python library for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data, including time series data, which is commonly used in financial analysis.

NumPy: A fundamental Python library for numerical computing. It provides a powerful array data structure and a wide range of mathematical functions, making it useful for various calculations and operations in quantitative finance.

Matplotlib: A data visualization library in Python. It provides a wide range of functions for creating various types of plots and charts, allowing for the graphical representation of financial data and analysis results.

PyFolio: A Python library that integrates with the Backtrader framework to provide portfolio performance analysis and evaluation. It offers tools for risk and return analysis, position tracking, and generating performance reports.

QuantStats: A Python library for quantitative financial analysis. It provides a comprehensive set of functions and metrics for analyzing and evaluating the performance of investment strategies, including measures of risk, return, and portfolio optimization.

CSV (Comma-Separated Values): A file format used for storing tabular data, where each value is separated by a comma. CSV files are commonly used for importing and exporting data in various applications, including financial data analysis.

Portfolio Optimization: The process of constructing an investment portfolio that aims to achieve the best possible trade-off between risk and return. It involves selecting an appropriate mix of assets based on their historical performance, correlations, and other relevant factors to maximize the portfolio's expected returns while minimizing its risk.

Time Series Data: Data that is collected and recorded over a sequence of equally spaced time intervals. In the context of this project, time series data refers to historical stock prices and other related information recorded at regular intervals over a specific period.

Commission: A fee charged by a brokerage or financial institution for executing trades on behalf of investors. It is typically calculated as a percentage of the trade value and represents a transaction cost incurred when buying or selling assets.

Performance Analysis: The evaluation and assessment of the performance of an investment strategy or portfolio. It involves analyzing various metrics and indicators, such as returns, risk measures, drawdowns, and other performance statistics, to gauge the effectiveness and profitability of the strategy or portfolio.

Portfolio Value: The total value of an investment portfolio, including the combined value of all assets held within the portfolio. It represents the net worth of the portfolio and is calculated by multiplying the number of shares or units held for each asset by their respective prices and summing the values.

Rebalance: The process of adjusting the composition of a portfolio by buying or selling assets to maintain a desired target allocation. Rebalancing is typically performed periodically to ensure that the portfolio remains aligned with the investment objectives and desired risk-return profile.

<u>Strategy Performance:</u> The assessment of how well a trading strategy or investment approach has performed over a given period. It involves analyzing key performance metrics, such as returns, risk-adjusted returns, volatility, Sharpe ratio, and other