Edvard Tsymbala Kupchynskyy

A1: Project Write-Up

HULT International Business School

DAT4876 Algorithmic Trading in Python

Dr. Michael Rolleigh

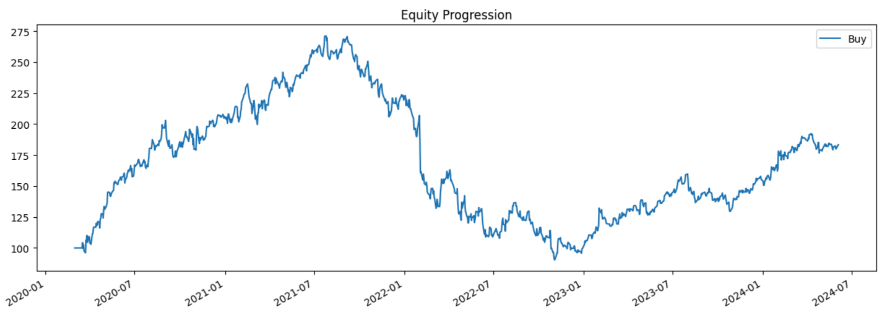
Algorithmic trading, also called Algo trading or black-box trading, indicates how trade orders are executed. This has been present through pre-programmed, automated trading instructions considering variables like time, price, and volume. It has evolved to take advantage of speed and data processing features on the computer over human traders.

Hence, participants in the financial markets look for some edge in their trading imparted by formulating strategies that can predict and exploit future market movements. The onset of computers and the internet has led to the advent of a new style of trading known as algorithmic trading. In other words, using complex math-based models or formulas to make high-speed decisions and transactions in the financial markets is called algorithmic trading. But the movement in the market is based on something other than mathematical formulas: human emotions. It is unpredictable and drives the market.

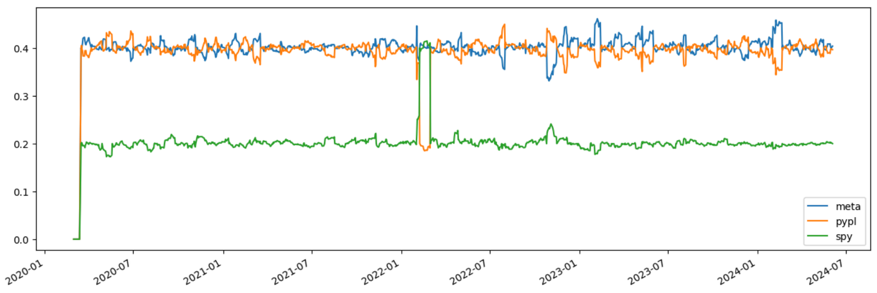
This paper aims to investigate the subtleties of algorithmic trading and some possible strategies. We are going to discuss the underlying algorithms that these systems use, the impact of such an algorithmic trading strategy on market quality, and how it is going to affect the chosen strategy within our portfolio.

In the strategy, we import historical price data for the specified tickers and calculate the risk-free rate. Define and backtest the three strategies, Optimal, which uses Sharpe Ratios and Risk-free rate, EMA cross, and MACD. However, the MACD Strategy performs a grid search to find the best MACD parameters and backtesting the strategy with these parameters.

Optimal Portfolio Strategy



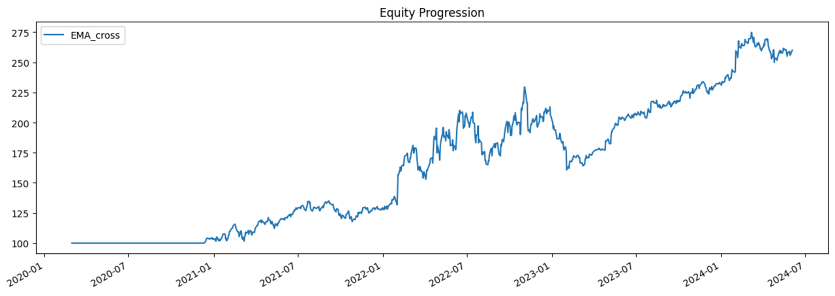
.

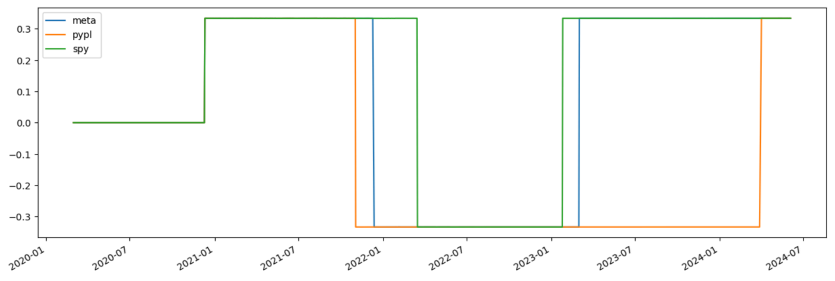


The WeightMeanVar strategy applies the mean-variance optimization technique in a trial for the best asset weights that minimize portfolio risk. The plan was put to the test on three assets: META, PYPL, and SPY. The Sharpe Ratio for these average risk-free rates was used as the starting point of the computations. It employs a three-month lookback period and bounds the weights between 0.05 and 0.35. During this period, the strategy had a 83.33% of Total Return with a CAGR of 15.30%. However, it had a really big drawdown after 2021-08 of -66.66% suggesting that it did not adjust correctly to some of the market trends.

The strategy held an almost constantly ongoing equity curve with reasonable drawdowns. Allocation weights were flexibly changed based on the changing market environment to get the best possible risk-return profile.

EMA Cross Strategy

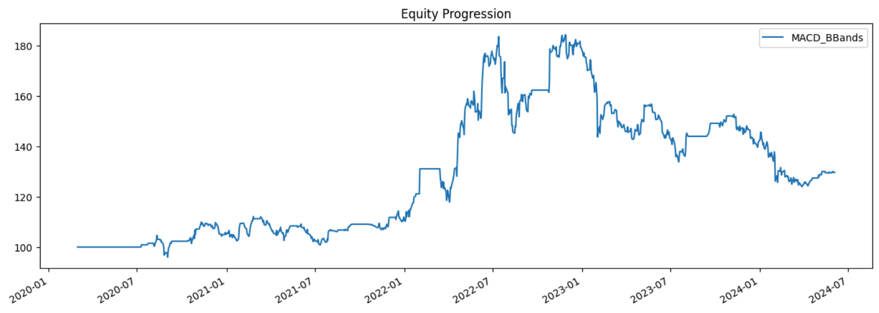


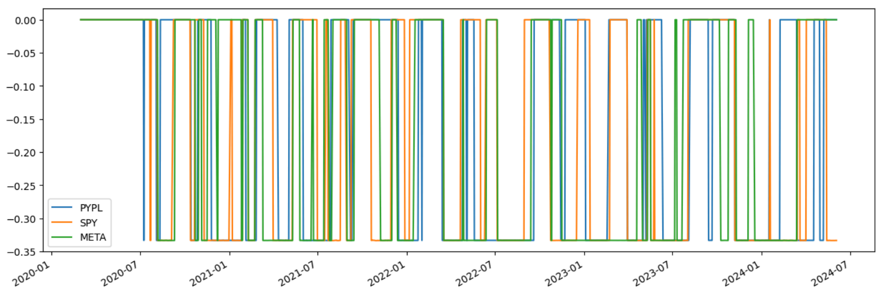


The EMA crossover system uses two exponential moving averages: one near the price and the other far away. A signal is thus received to buy when the first moves above the second. In that way, the system was programmed to compute the 52- and 200-day exponential moving averages to derive signals on the crossover points.

Growth in the posted strategy was steady from early 2020, with an impressive high in early 2022. Drawdowns were experienced over shorter periods, but the long-term trend was upward. The strategy generated fewer trades than the mean-variance strategy, indicating its longer-term focus.
This strategy was selected because its indicators will, therefore, catch medium-term trends, offering opportunities for momentum riding in the market. It has shown strong performance with clear trends but is subject to periods of underperformance during sideways markets.
The EMA cross strategy had the highest Total Return of all the strategies with 160.09%, a CAGR of 25.17% and a drawdown of as low as -30%. This could be due to its relatively low rebalancing and possibly avoiding some market downturns since it focuses on a long term investment strategy rather than looking for short-term trades.

MACD Strategy

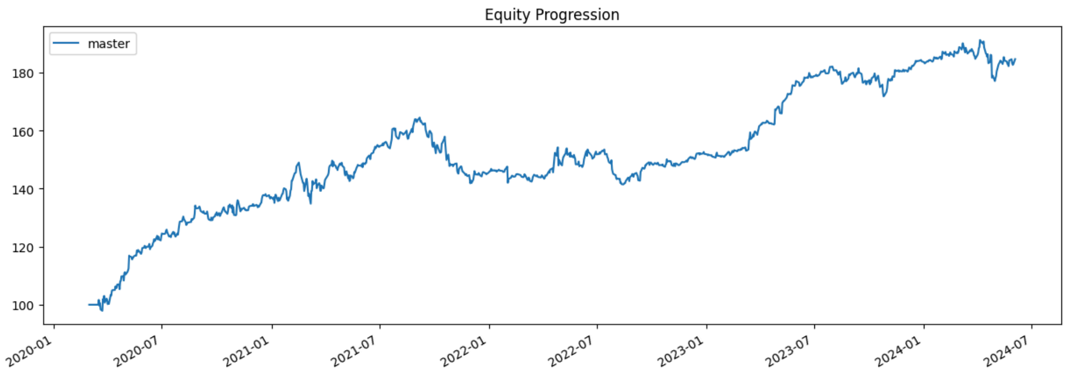


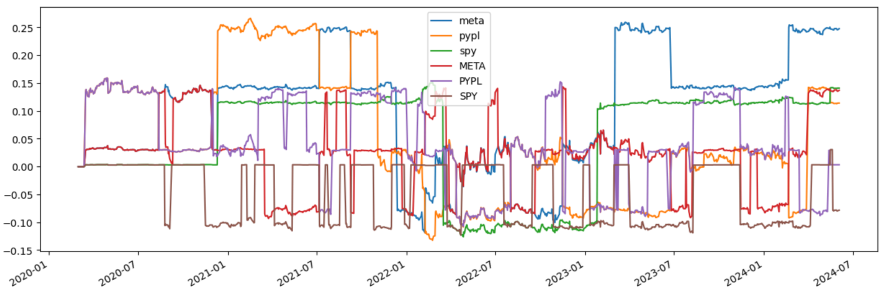


The MACD strategy capitalizes on the difference between the short-term EMA and the long-term EMA to identify trends. This strategy comes with a signal line, an EMA of the MACD line itself. Purchase and sale signals are generated based on the crossover of the MACD line with the signal line. The objective of this strategy was to capture momentum- and trend-following opportunities. The strategy showed moderate growth and several periods

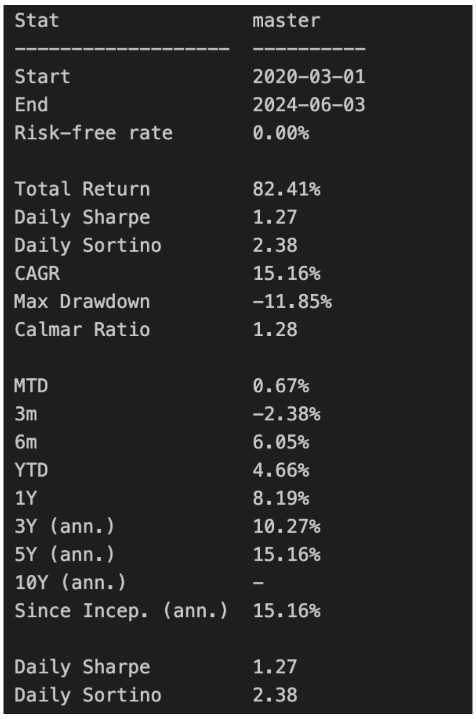
of drawdown. The plan is suitable for trending markets but not as good for ranging markets. That highlights the strategy's sensitivity to market changes since there are ample occasions for "buy" and "sell" signals. This would lead to increased transaction costs and potential overtrading. So it would perform well in strong market trends but badly otherwise.
The MACD strategy had the lowest type of returns, while being the most challenging one to compute. The Total Return of the period is of 29.61% with a CAGR of 6.28% and a Max Drawdown of -32.78%. This might suggest that while having a positive outcome, due to the indicator being of a lagging nature. It may need some more assistance of other formulas or strategies.

Master Strategy





The equity curve shows the equity path of the master strategy between January 2020 and July 2024. In this initial period, equity continuously increases, so the strategy's performance is decent. The equity line does show a noticeable dip across the four months from around 2021-08, which does give the impression of a market downturn or imperfect strategy performance during this period. After the drawdown, a recovery of the equity value is signaled, with the following period being one of steady growth. The flattening of the equity curve at the tail end of the period indicates that either market conditions have stabilized or, at worst, the strategy has begun to lose responsiveness to changes.

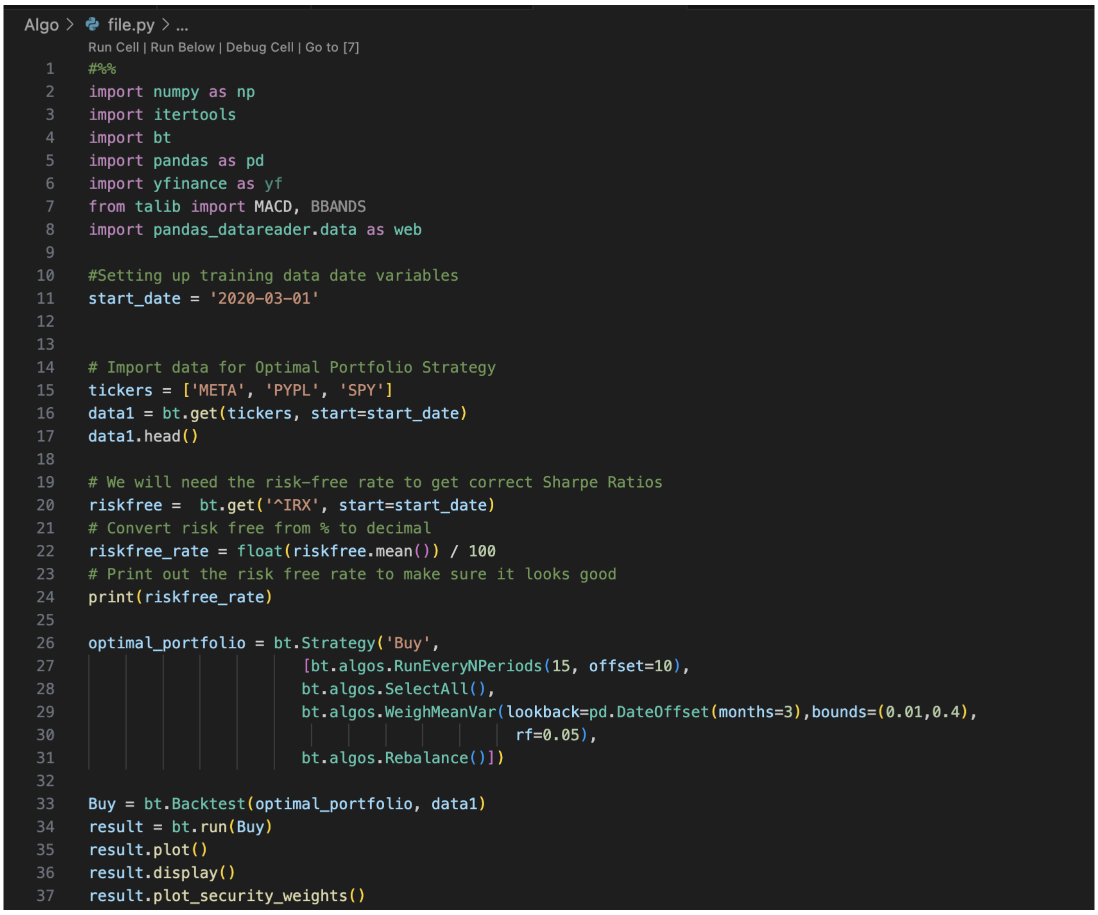


```
Stat                 master
-------------------  ----------
Start                2020-03-01
End                  2024-06-03
Risk-free rate       0.00%

Total Return         82.41%
Daily Sharpe         1.27
Daily Sortino        2.38
CAGR                 15.16%
Max Drawdown         -11.85%
Calmar Ratio         1.28

MTD                  0.67%
3m                   -2.38%
6m                   6.05%
YTD                  4.66%
1Y                   8.19%
3Y (ann.)            10.27%
5Y (ann.)            15.16%
10Y (ann.)           -
Since Incep. (ann.)  15.16%

Daily Sharpe         1.27
Daily Sortino        2.38
```

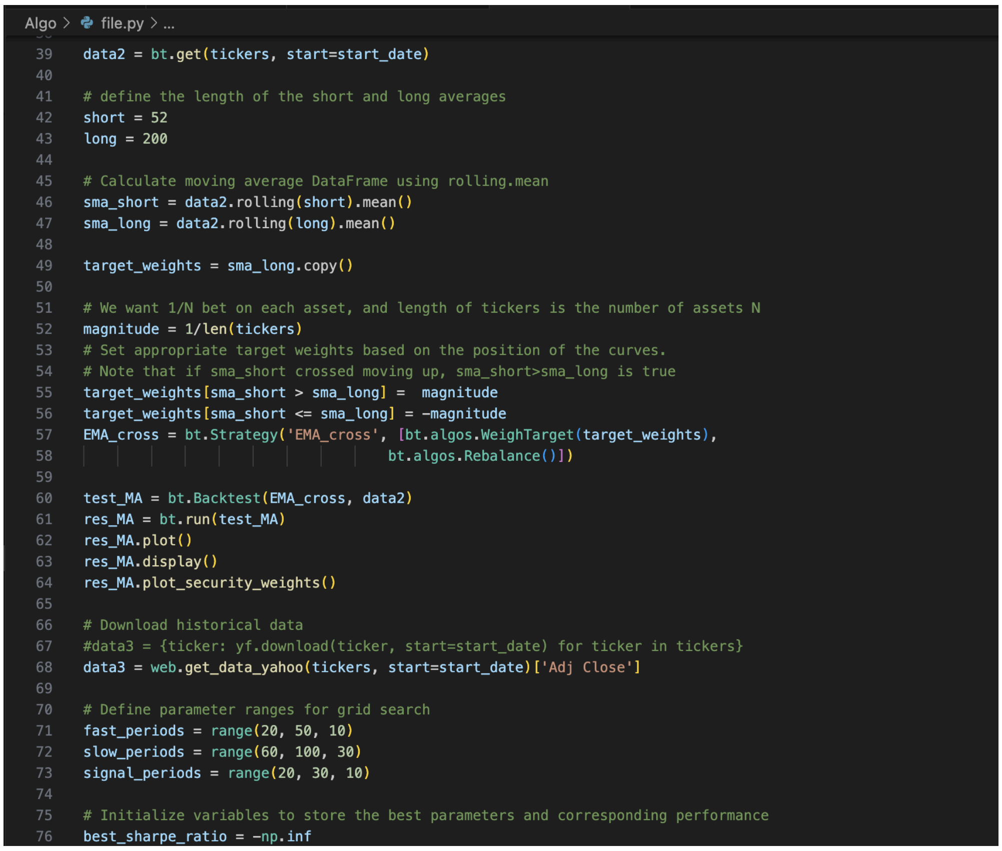
The master strategy shows a strong performance with a spectacular total return and CAGR that produces substantial long-term returns. High Sharpe and Sortino ratios suggest good returns with risk, supported by the modest value of maximum drawdown and high Calmar ratio as an indicator of reasonable effective risk control.

The strategy attained an 82.41% total return over the backtest period. This points to very high growth, which means that the strategy was very effective in capturing profitable opportunities in the market. Therefore, the CAGR of 15.16% shows very effective annualized returns, and the strategy will be pretty attractive for long-term investors. A Sharpe ratio of 1.27 suggests that the strategy has generated good returns relative to the risk taken. Satisfactory means that it gives a better return per unit of risk. The maximum drawdown of -11.85% points to the largest peak-to-trough decline in the value of the portfolio over the sample period. This is not insignificant but relatively modest compared to the overall return, suggesting good risk management.

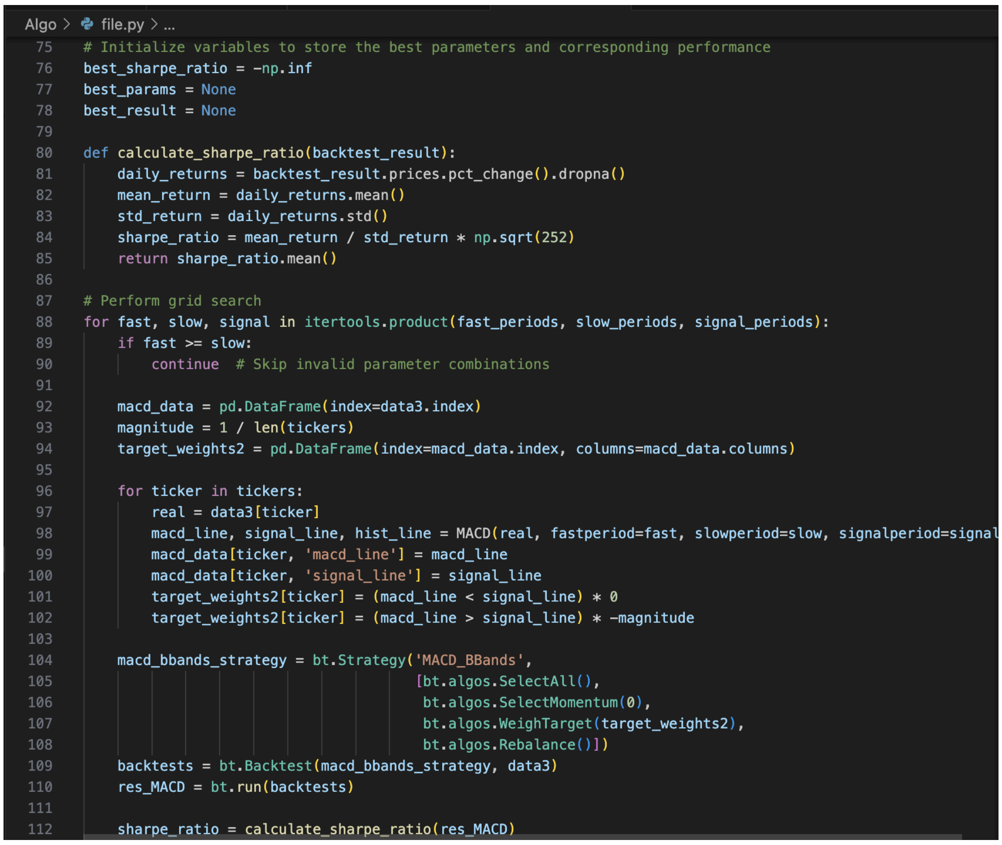
Appendix



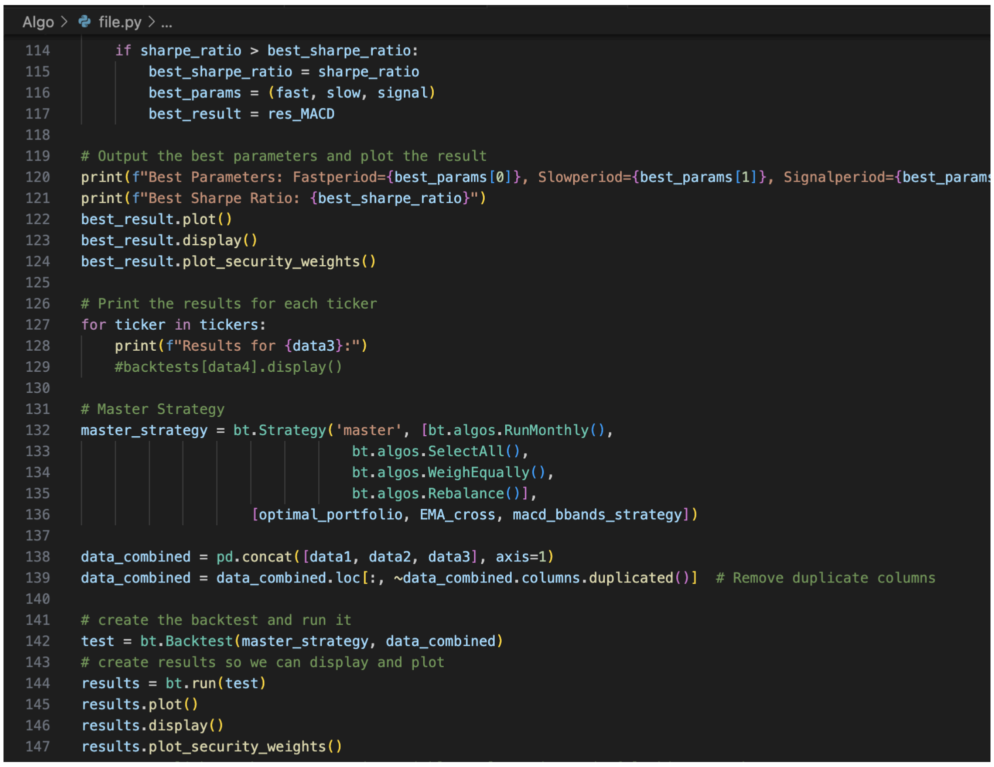
```python
#%%
import numpy as np
import itertools
import bt
import pandas as pd
import yfinance as yf
from talib import MACD, BBANDS
import pandas_datareader.data as web

#Setting up training data date variables
start_date = '2020-03-01'


# Import data for Optimal Portfolio Strategy
tickers = ['META', 'PYPL', 'SPY']
data1 = bt.get(tickers, start=start_date)
data1.head()

# We will need the risk-free rate to get correct Sharpe Ratios
riskfree =  bt.get('^IRX', start=start_date)
# Convert risk free from % to decimal
riskfree_rate = float(riskfree.mean()) / 100
# Print out the risk free rate to make sure it looks good
print(riskfree_rate)

optimal_portfolio = bt.Strategy('Buy',
                       [bt.algos.RunEveryNPeriods(15, offset=10),
                       bt.algos.SelectAll(),
                       bt.algos.WeighMeanVar(lookback=pd.DateOffset(months=3),bounds=(0.01,0.4),
                                             rf=0.05),
                       bt.algos.Rebalance()])

Buy = bt.Backtest(optimal_portfolio, data1)
result = bt.run(Buy)
result.plot()
result.display()
result.plot_security_weights()
```



```python
data2 = bt.get(tickers, start=start_date)

# define the length of the short and long averages
short = 52
long = 200

# Calculate moving average DataFrame using rolling.mean
sma_short = data2.rolling(short).mean()
sma_long = data2.rolling(long).mean()

target_weights = sma_long.copy()

# We want 1/N bet on each asset, and length of tickers is the number of assets N
magnitude = 1/len(tickers)
# Set appropriate target weights based on the position of the curves.
# Note that if sma_short crossed moving up, sma_short>sma_long is true
target_weights[sma_short > sma_long] =  magnitude
target_weights[sma_short <= sma_long] = -magnitude
EMA_cross = bt.Strategy('EMA_cross', [bt.algos.WeighTarget(target_weights),
                                      bt.algos.Rebalance()])

test_MA = bt.Backtest(EMA_cross, data2)
res_MA = bt.run(test_MA)
res_MA.plot()
res_MA.display()
res_MA.plot_security_weights()

# Download historical data
#data3 = {ticker: yf.download(ticker, start=start_date) for ticker in tickers}
data3 = web.get_data_yahoo(tickers, start=start_date)['Adj Close']

# Define parameter ranges for grid search
fast_periods = range(20, 50, 10)
slow_periods = range(60, 100, 30)
signal_periods = range(20, 30, 10)

# Initialize variables to store the best parameters and corresponding performance
best_sharpe_ratio = -np.inf
```



```python
# Initialize variables to store the best parameters and corresponding performance
best_sharpe_ratio = -np.inf
best_params = None
best_result = None

def calculate_sharpe_ratio(backtest_result):
    daily_returns = backtest_result.prices.pct_change().dropna()
    mean_return = daily_returns.mean()
    std_return = daily_returns.std()
    sharpe_ratio = mean_return / std_return * np.sqrt(252)
    return sharpe_ratio.mean()

# Perform grid search
for fast, slow, signal in itertools.product(fast_periods, slow_periods, signal_periods):
    if fast >= slow:
        continue  # Skip invalid parameter combinations

    macd_data = pd.DataFrame(index=data3.index)
    magnitude = 1 / len(tickers)
    target_weights2 = pd.DataFrame(index=macd_data.index, columns=macd_data.columns)

    for ticker in tickers:
        real = data3[ticker]
        macd_line, signal_line, hist_line = MACD(real, fastperiod=fast, slowperiod=slow, signalperiod=signal
        macd_data[ticker, 'macd_line'] = macd_line
        macd_data[ticker, 'signal_line'] = signal_line
        target_weights2[ticker] = (macd_line < signal_line) * 0
        target_weights2[ticker] = (macd_line > signal_line) * -magnitude

    macd_bbands_strategy = bt.Strategy('MACD_BBands',
                                       [bt.algos.SelectAll(),
                                        bt.algos.SelectMomentum(0),
                                        bt.algos.WeighTarget(target_weights2),
                                        bt.algos.Rebalance()])
    backtests = bt.Backtest(macd_bbands_strategy, data3)
    res_MACD = bt.run(backtests)

    sharpe_ratio = calculate_sharpe_ratio(res_MACD)
```



```python
    if sharpe_ratio > best_sharpe_ratio:
        best_sharpe_ratio = sharpe_ratio
        best_params = (fast, slow, signal)
        best_result = res_MACD

# Output the best parameters and plot the result
print(f"Best Parameters: Fastperiod={best_params[0]}, Slowperiod={best_params[1]}, Signalperiod={best_param
print(f"Best Sharpe Ratio: {best_sharpe_ratio}")
best_result.plot()
best_result.display()
best_result.plot_security_weights()

# Print the results for each ticker
for ticker in tickers:
    print(f"Results for {data3}:")
    #backtests[data4].display()

# Master Strategy
master_strategy = bt.Strategy('master', [bt.algos.RunMonthly(),
                                bt.algos.SelectAll(),
                                bt.algos.WeighEqually(),
                                bt.algos.Rebalance()],
                    [optimal_portfolio, EMA_cross, macd_bbands_strategy])

data_combined = pd.concat([data1, data2, data3], axis=1)
data_combined = data_combined.loc[:, ~data_combined.columns.duplicated()]  # Remove duplicate columns

# create the backtest and run it
test = bt.Backtest(master_strategy, data_combined)
# create results so we can display and plot
results = bt.run(test)
results.plot()
results.display()
results.plot_security_weights()
```