



Algorithmic trading

Introduction to Computation Finance

นายอิทธิเดช นามเหลา 6510503905

เสนอ

ดร. ยอดเยี่ยม ทิพย์สุวรรณ

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์ ปการศึกษา
2567 มหาวิทยาลัยเกษตรศาสตร์


Final project – Introduction to Computational Finance

1. เลือกหุ้น, options, commodity, หรือ ETF 5 ตัว หรือหากมี financial instrument ในใจตัวใดที่อยากเทรด สามารถเลือกมาได้

เลือก options มาดังนี้

- 1).Amazon
- 2).Apple
- 3).Netflix
- 4).Google
- 5).Tesla

```
[43] 1 import yfinance as yf
      2 import pandas as pd
      3 from datetime import datetime
      4 import numpy as np
      5
      6 tickers = ["AAPL", "AMZN", "NFLX", "DSNY", "TSLA"]
      7
      8 initial_train_start = '2018-11-01'
      9 initial_train_end = '2023-11-01'
     10 test_start = '2023-11-01'
     11 test_end = '2024-11-01'
     12
     13 # Fetch historical adjusted close prices for the training period (5 years)
     14 data_5y = yf.download(tickers, start=initial_train_start, end=initial_train_end)['Adj Close']
     15 returns_5y = data_5y.pct_change().dropna()
     16
     17 # Fetch historical adjusted close prices for the testing period (1 year)
     18 data_1y = yf.download(tickers, start=test_start, end=test_end)['Adj Close']
     19 daily_returns_1y = data_1y.pct_change().dropna()
     20
```

 [*****100%*****] 5 of 5 completed
[*****100%*****] 5 of 5 completed

2. นำข้อมูล historical data ย้อยหลัง 6 ปีจนถึงย้อนหลัง 1 ปี จาก IBKR หรือ Yahoo finance มาใช้เพื่อลองทำ Portfolio optimization ให้ลองใช้

2.a Mean-variance optimization

จากการคำนวณโดยใช้ข้อมูลจากเมื่อ 6 ปีก่อนมาจนถึง 1 ปีก่อนแล้วนำ Expected return และ Covariance มาคำนวณ Mean-variance optimization ดัง code Output:

```
Optimal Weights (Mean-Variance): [0.59838, 0.0, 0.0, 0.0, 0.40162]
Expected Portfolio Return (Mean-Variance): 37.67%
Expected Portfolio Risk (Mean-Variance): 39.77%
```

ซึ่ง Weight ไม่กระจายเลยและก็มี Expected Return และ Risk ที่สูงมาก

```
0 รันแล้ว
1 from pypfopt import EfficientFrontier, risk_models, expected_returns
2
3 mu = expected_returns.mean_historical_return(data_5y)
4 covariance = risk_models.sample_cov(data_5y)
5 # print(covariance)
6 # cov_matrix = returns_5y.cov()
7 # print(cov_matrix)
8
9 ef = EfficientFrontier(mu, covariance)
10 weights = ef.max_sharpe()
11 cleaned_weights = ef.clean_weights()
12
13 expected_performance = ef.portfolio_performance()
14 weights_list = [cleaned_weights[ticker] for ticker in tickers]
15
16 print("Optimal Weights (Mean-Variance):", weights_list)
17 print(f"Expected Portfolio Return (Mean-Variance): {expected_performance[0]:.2%}")
18 print(f"Expected Portfolio Risk (Mean-Variance): {expected_performance[1]:.2%}")
19 optimal_weights_mvo = weights_list

Optimal Weights (Mean-Variance): [0.59838, 0.0, 0.0, 0.0, 0.40162]
Expected Portfolio Return (Mean-Variance): 37.67%
Expected Portfolio Risk (Mean-Variance): 39.77%
```

2.a.i Balanced

เป็นการลดค่า Risk และ Return เพื่อลดความเสี่ยง

```
Balanced Optimal Weights for Target Risk: [0.42607, 0.10168, 0.04374, 0.42744, 0.00106]
Expected Portfolio Return (Risk Target): 20.24%
Expected Portfolio Risk (Risk Target): 29.50%
```

```
[65] 1 target_risk = 0.295
      2
      3 # Optimize for minimum volatility at the target risk level
      4 ef = EfficientFrontier(mu, covariance)
      5 weights = ef.efficient_risk(target_risk)
      6 cleaned_weights = ef.clean_weights()
      7
      8 # Calculate expected portfolio performance
      9 expected_performance = ef.portfolio_performance()
     10
     11 # Format weights as a list
     12 weights_list = [cleaned_weights[ticker] for ticker in tickers]
     13
     14 print("Balanced Optimal Weights for Target Risk:", weights_list)
     15 print(f"Expected Portfolio Return (Risk Target): {expected_performance[0]:.2%}")
     16 print(f"Expected Portfolio Risk (Risk Target): {expected_performance[1]:.2%}")
     17 # optimal_weights_mvo = weights_list
```

2.b Sharpe Ratio Optimization

คำนวณโดยใช้ข้อมูลจากเมื่อ 6 ปีก่อนมาจนถึง 1 ปีก่อนได้ mean_returns และ covariance มาใช้คำนวณต่อ โดยให้ Minimum allocation อยู่ที่ 10%

Output:

```
Optimal Weights (Sharpe Ratio): [0.1 0.1 0.1 0.1 0.6]
Expected Portfolio Return (Sharpe Ratio): 0.19%
Expected Portfolio Risk (Sharpe Ratio): 2.93%
Sharpe Ratio: -0.62
```

```
1 from scipy.optimize import minimize
2
3 # Calculate mean returns and covariance matrix from the past data
4 mean_returns = returns_5y.mean()
5 cov_matrix = returns_5y.cov()
6
7 risk_free_rate = 0.02
8
9 def portfolio_performance(weights, mean_returns, cov_matrix):
10     portfolio_return = np.sum(mean_returns * weights)
11     portfolio_std_dev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
12     return portfolio_return, portfolio_std_dev
13
14 # Function to minimize the negative Sharpe Ratio
15 def neg_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate):
16     portfolio_return, portfolio_risk = portfolio_performance(weights, mean_returns, cov_matrix)
17     sharpe_ratio = (portfolio_return - risk_free_rate) / portfolio_risk
18     return -sharpe_ratio
19
20 num_assets = len(mean_returns)
21 initial_weights = np.array([1 / num_assets] * num_assets)
22 bounds = [(0.1, 1.0)] * num_assets
23 constraints = [
24     {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}
25 ]
26 # Optimize Sharpe Ratio
27 args = (mean_returns, cov_matrix, risk_free_rate)
28 result = minimize(neg_sharpe_ratio, initial_weights, args=args, method='SLSQP', bounds=bounds, constraints=constraints)
29
30 # Get optimal weights
31 optimal_weights_sharpe = result.x
32
33 opt_return_sharpe, opt_risk_sharpe = portfolio_performance(optimal_weights_sharpe, mean_returns, cov_matrix)
34 opt_sharpe_ratio = (opt_return_sharpe - risk_free_rate) / opt_risk_sharpe
35
36 print("Optimal Weights (Sharpe Ratio):", optimal_weights_sharpe)
37 print(f"Expected Portfolio Return (Sharpe Ratio): {opt_return_sharpe:.2%}")
38 print(f"Expected Portfolio Risk (Sharpe Ratio): {opt_risk_sharpe:.2%}")
39 print(f"Sharpe Ratio: {opt_sharpe_ratio:.2f}")
```

```
Optimal Weights (Sharpe Ratio): [0.1 0.1 0.6 0.1 0.1]
Expected Portfolio Return (Sharpe Ratio): 0.19%
Expected Portfolio Risk (Sharpe Ratio): 4.21%
Sharpe Ratio: -0.43
```

2.c Black-Litterman Optimization

เช่นเดิม Cov มาจากข้อมูลเก่า 5 ปี และกำหนดมุมมองตามเทรนที่เริ่มเมื่อ 1 ปี ก่อนโดยประมาณ

Output:

Optimal Weights (Black-Litterman): OrderedDict([('AAPL', 0.3629391436948078), ('AMZN', 0.0), ('GOOGL', 0.393704286816254), ('NFLX', 0.0), ('TSLA', 0.2433565694889383)])

Expected Portfolio Return (Black-Litterman): 11.03%

Expected Portfolio Risk (Black-Litterman): 2.10%

```
[62] 1 from pyfpopt import BlackLittermanModel
      2
      3 cov_matrix = returns_5y.cov()
      4
      5 risk_free_rate = 0.02
      6 expected_market_return = 0.08
      7
      8 # Fetch beta values for the tickers
      9 betas = {}
     10 for ticker in tickers:
     11     stock = yf.Ticker(ticker)
     12     betas[ticker] = stock.info['beta']
     13
     14 # Market-implied prior returns (using CAPM model)
     15 market_prior = np.array([risk_free_rate + beta * (expected_market_return - risk_free_rate) for beta in betas.values()])
     16
     17 views = {"AAPL": 0.04, "GOOGL": 0.02, "AMZN": 0.03, "TSLA": 0.03, "NFLX": 0.03}
     18
     19 # Set up the Black-Litterman model
     20 bl = BlackLittermanModel(cov_matrix=cov_matrix, pi=market_prior, absolute_views=views)
     21 bl_adjusted_returns = bl.bl_returns()
     22
     23 # Optimize the portfolio with Black-Litterman adjusted returns
     24 ef = EfficientFrontier(bl_adjusted_returns, cov_matrix)
     25 bl_weights = ef.max_sharpe()
     26
     27 print("Optimal Weights (Black-Litterman):", bl_weights)
     28
     29 def portfolio_performance(weights, mean_returns, cov_matrix):
     30     weights_array = np.array(list(weights.values()))
     31     portfolio_return = np.sum(mean_returns * weights_array)
     32     portfolio_std_dev = np.sqrt(np.dot(weights_array.T, np.dot(cov_matrix, weights_array)))
     33     return portfolio_return, portfolio_std_dev
     34
     35 opt_return_bl, opt_risk_bl = portfolio_performance(bl_weights, market_prior, cov_matrix)
     36 print(f"Expected Portfolio Return (Black-Litterman): {opt_return_bl:.2%}")
     37 print(f"Expected Portfolio Risk (Black-Litterman): {opt_risk_bl:.2%}")
```

2. Performance

ได้ผลลัพธ์ต่างๆดังนี้

Mean-Variance Optimization Metrics:

Final Portfolio Value: \$131,219.44

Profit: \$31,219.44

Percentage Increase: 31.22%

Sharpe Ratio: 0.89

Rolling Volatility: 0.40

Maximum Drawdown: -0.28

Value at Risk (VaR 95%): -0.03

Conditional Value at Risk (CVaR 95%): -0.04

Sharpe Ratio Optimization Metrics:

Final Portfolio Value: \$135,555.20

Profit: \$35,555.20

Percentage Increase: 35.56%

Sharpe Ratio: 0.86

Rolling Volatility: 0.56

Maximum Drawdown: -0.28

Value at Risk (VaR 95%): -0.03

Conditional Value at Risk (CVaR 95%): -0.05

Black-Litterman Optimization Metrics:

Final Portfolio Value: \$134,728.98

Profit: \$34,728.98

Percentage Increase: 34.73%

Sharpe Ratio: 1.22

Rolling Volatility: 0.27

Maximum Drawdown: -0.17

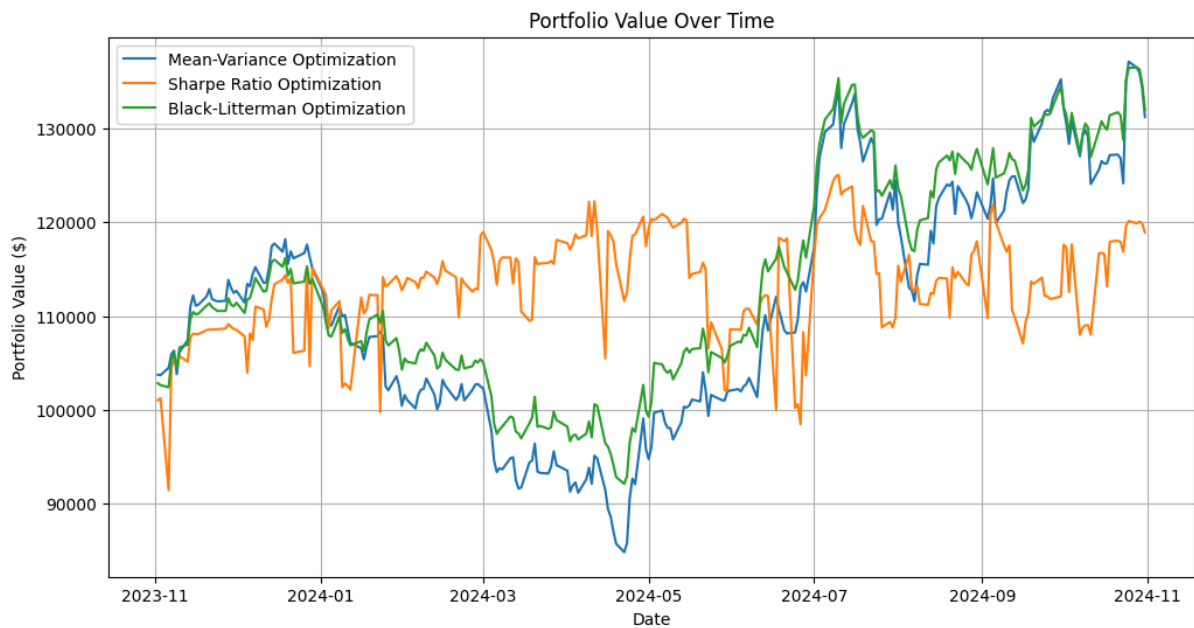
Value at Risk (VaR 95%): -0.02

Conditional Value at Risk (CVaR 95%): -0.03

ซึ่งค่าที่มีการขีดเส้นใต้คือค่าที่เป็นค่าที่ดีที่สุด

สรุปได้ว่า Black-Litterman Optimization มี Performance โดยรวมดีที่สุดโดยมี Mean-Variance Optimization รองลงมาแต่ในแง่ของ Profit ถือว่าสูสีกันมากแต่ Black-Litterman นั้นมี Sharpe Ratio มากที่สุดได้เกิน1ซึ่งหมายถึงว่าการลงทุนโดยใช้วิธีนี้นั้นคุ้มค่าต่อความเสี่ยงที่มี และ Rolling Volatility ที่น้อยที่สุดนั้นก็ค่อนข้างที่จะ

stable และก็ง่ายต่อการคาดเดา และก็มี Maximum Drawdown(ราคาจุดต่ำสุดเมื่อเทียบกับจุดสูงสุด) เป็นอันดับ 2 VaR และ CVaR น้อยที่สุด



3.

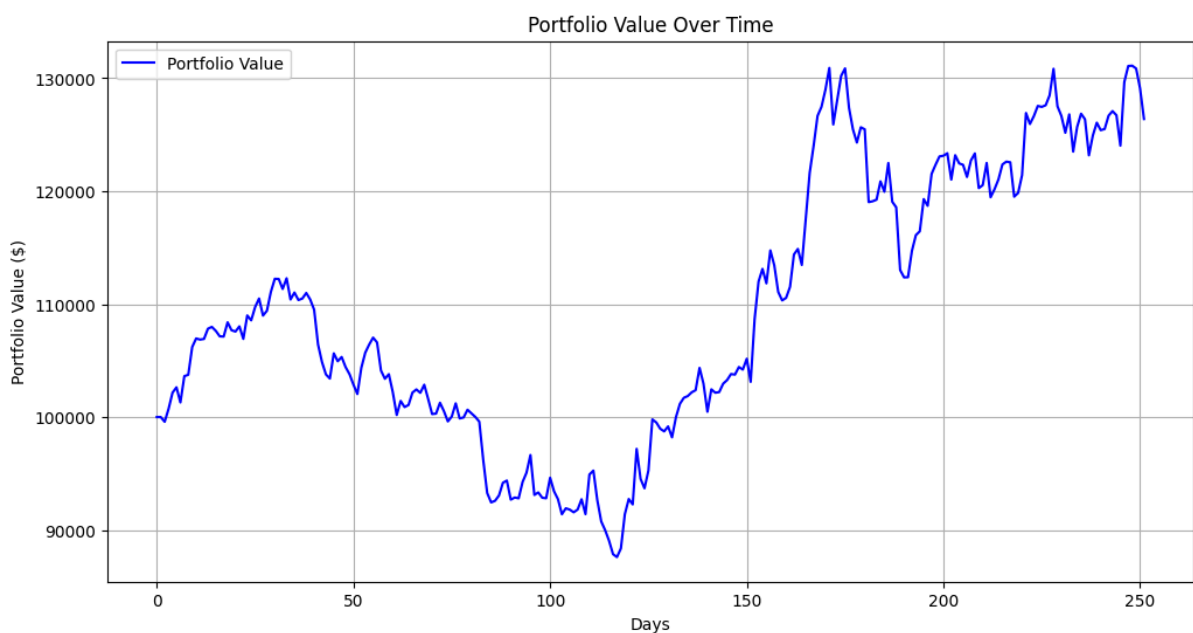
3.a. ใช้ข้อมูลย้อนหลัง 6 ปี จนถึงก่อนหน้า 1 ปี เพื่อ train model ทำ backtest ของ strategies แล้วใช้โมเดลหรือ strategies ที่ได้ มาทดสอบกับข้อมูลย้อนหลัง 1 ปีจนถึงปัจจุบัน ที่เดียว แล้ววิเคราะห์ performance เทียบกับ buy-and-hold

ผมใช้ Random Forest Strategy ซึ่งเป็น Machine learning โดยใช้ Random Forest Regressing Model และใช้ Backtest ในการ simulation 1ปี

Output:

```
➡ [ *****100%***** ]  
[ *****100%***** ]  
Start Portfolio Value: $100000  
Final Portfolio Value: $126394.36  
Profits: 26.39%  
Final Portfolio Value: 126.39%  
Sharpe Ratio: 1.07  
Rolling Volatility: 0.24  
Maximum Drawdown: 0.22  
Value at Risk (VaR 95%): -0.03  
Conditional Value at Risk (CVaR 95%): -0.03
```

จะได้ Performance ในเชิงของ Profits ถือว่าใกล้เคียงกับ buy-and-hold แต่น้อยกว่านิดหน่อยส่วนค่าอื่นๆก็ถือว่าค่อนข้างดี Sharpe ratio สูง Rolling Volatility น้อย



3.b b. ใช้ข้อมูลย้อนหลัง 6 ปี จนถึงก่อนหน้า 1 ปี เพื่อ train model ทำ backtest ของ strategies แล้วใช้โมเดลหรือ strategies ที่ได้ มาทดสอบกับข้อมูล 12 เดือนย้อนหลัง เป็นเวลา 1 เดือน แล้ว ปรับ เทรน model ใหม่ โดยใช้ข้อมูลย้อนหลังที่ขยับมาข้างหน้า 1 เดือน เพื่อเทรน model และปรับ strategies ใหม่ ทดสอบกับ ข้อมูลย้อนหลังเดือนที่ 11 ขยับการดึงข้อมูลเช่นนี้ไปเรื่อยๆ จนถึงปัจจุบัน ให้วิเคราะห์ performance เทียบกับ a. และ buy-and-hold

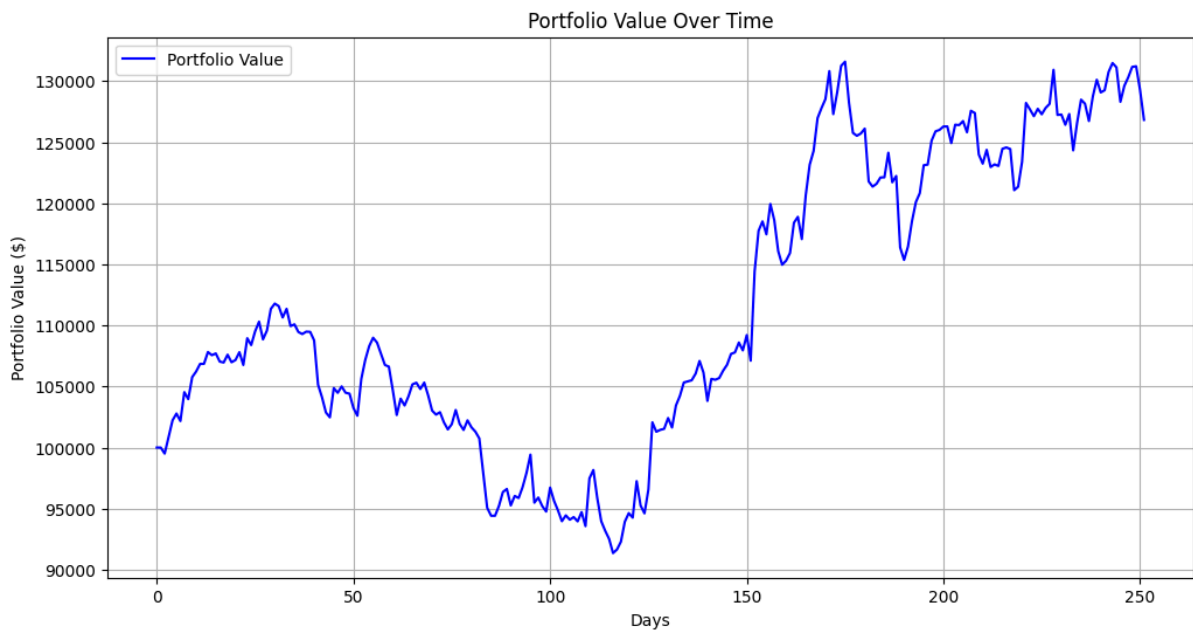
ผมใช้ Random Forest Strategy เช่นเดิมแต่จะทำการ Train ใหม่ทุกๆครั้งตอนจบ เดือนและปรับ Weight ของ Strategy ใหม่ทุกๆเดือน

```
[*****100%*****] 5 of 5 completed
Start Portfolio Value: $100000
Final Portfolio Value: $126826.02
Profits: 26.83%
Sharpe Ratio: 1.17
Rolling Volatility: 0.22
Maximum Drawdown: 0.18
Value at Risk (VaR 95%): -0.02
Conditional Value at Risk (CVaR 95%): -0.03

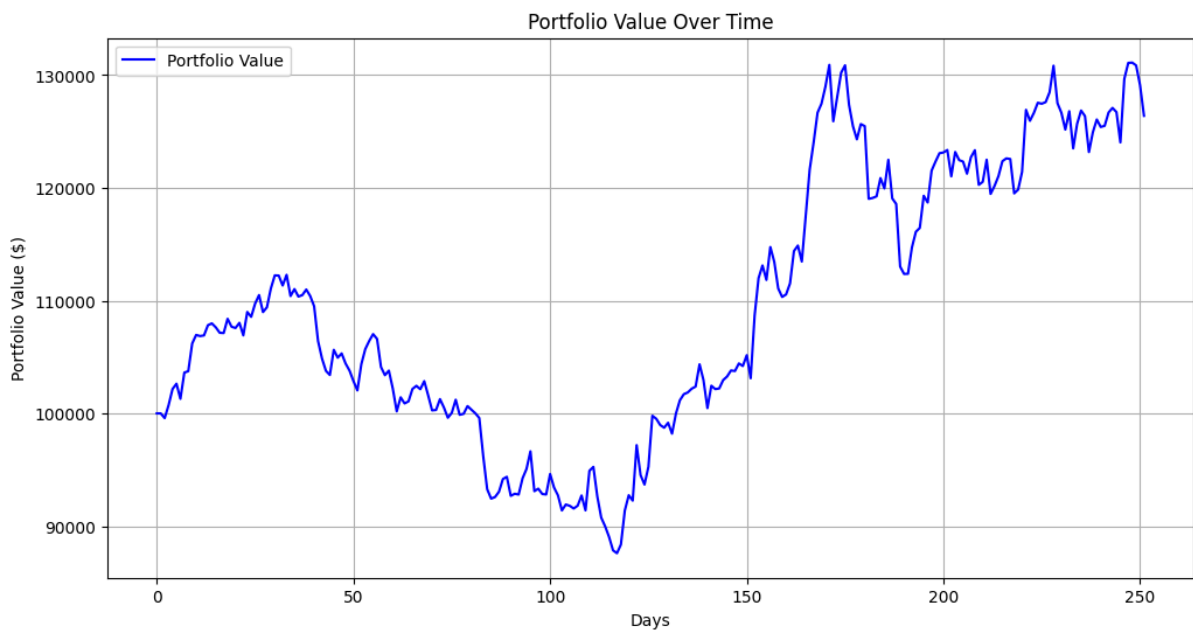
Portfolio Value at the end of 2023-11: $107166.90
Portfolio Value at the end of 2023-12: $108775.98
Portfolio Value at the end of 2024-01: $102663.87
Portfolio Value at the end of 2024-02: $101302.98
Portfolio Value at the end of 2024-03: $95637.99
Portfolio Value at the end of 2024-04: $95257.02
Portfolio Value at the end of 2024-05: $106783.07
Portfolio Value at the end of 2024-06: $117065.22
Portfolio Value at the end of 2024-07: $124134.81
Portfolio Value at the end of 2024-08: $127385.34
Portfolio Value at the end of 2024-09: $130922.94
Portfolio Value at the end of 2024-10: $126826.02
```

ได้ Final Portfolio Value: \$126826.02 ซึ่งมีค่ามากกว่า 3.a เพียงนิดหน่อยและ
ก็ค่าต่างๆพวก Sharpe Ratio, Rolling Volatility, Maximum Drawdown, VaR และ
CVaR นั้นเหนือกว่าหรือเท่ากับ 3.a ทุกอย่างเลย แต่เมื่อเทียบกับ buy-and-hold
Black-Litterman Optimization แล้วถือว่าใกล้เคียงกันมากขึ้นหรือบาง matrices ก็มีค่า
เท่ากันเลย

กราฟจาก 3.b



กราฟจาก 3.a



จะเห็นได้ชัดเลยว่ากราฟของ ทั้ง 3.a และ 3.b นั้นมีความใกล้เคียงกันสูงมากแต่เมื่อดูดีๆแล้วจะเห็นว่ากราฟของ 3.b จะค่อนข้าง smooth มากกว่า

สรุปแล้ว Performance ของ 3.b นั้นสูงกว่า 3.a เกือบทุกด้านแต่ก็ยัง
Performance น้อยกว่าหรือเท่ากับ buy-and-hold Black-Litterman ในบางด้าน ดังนั้น
buy-and-hold Black-Litterman จึงเหมาะสมกับสถานการณ์นี้มากกว่า 3.b

ลิงค์ Colab ข้อ 1,2

<https://colab.research.google.com/drive/1kK9xXdWkpQOCTK8AEADbhlAwMJc3VEvy?usp=sharing>

ลิงค์ Colab ข้อ 3

https://colab.research.google.com/drive/1qraR_VpN6k0ZiXwKpHdos6IE7zpkRRVA?usp=sharing