5/24/2025

Individual Assignment: Portfolio Optimization Using Python

BBFE 31382- Computing for Finance

FE/2021/047 - Gamalath K.H.

University of Kelaniya

Faculty of Commerce and Management Studies

Department of Finance

BBFE 31382- Computing for Finance

Individual Assignment: Portfolio Optimization Using Python

UNIVERSITY OF KELANIYA | Faculty of Commerce & Management Studies

Department of Finance

Course code: BBFE 31382

Course Title: Computing for Finance

Semester : 1st Semester

Year : 3rd Year (2025)

Lecturer/(s) : Mr. A.J.P. Samarawickrama

Ms. P. Kethmi

Assignment Topic: Individual Assignment: Portfolio Optimization Using Python

Student Number and Name :

Student Number	**Student Name**
FE/2021/047	Gamalath K.H.

DECLERATION

I certify that this report does not incorporate without acknowledgment, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in the text.

Signature of the Student

Name of Student : Gamalath K.H.

Students/s Number: FE/2021/047

Date : 24/05/2025

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to my Lecturers; Mr. A.J.P. Samarawickrama and Ms. P. Kethmi for all the knowledge bestowed upon me and for guiding me on the preparation of this report. I also thank everyone who extended their support to me in the preparation and completion of this report.

EXECUTIVE SUMMARY

This project implements Modern Portfolio Theory (MPT) to construct an optimized investment portfolio using Python, focusing on risk-adjusted return maximization through Sharpe Ratio analysis. Historical adjusted closing prices for five diversified S&P 500 constituents—JPM, JNJ, DIS, BA, and XOM—were sourced via the yfinance API over a 5-year period. Preprocessing included daily return computation, outlier handling, and exploratory data analysis to establish statistical foundations. Annualized expected returns and the covariance matrix were derived to evaluate inter-asset dependencies and risk dynamics. A single simulated portfolio established baseline metrics for portfolio return and volatility, serving as a precursor to broader stochastic analysis.

A Monte Carlo simulation generated 10,000 random portfolios to construct the efficient frontier, facilitating the identification of both the maximum Sharpe Ratio and minimum volatility portfolios. Subsequently, constrained optimization using Sequential Least Squares Programming (SLSQP) via scipy.optimize was used to refine asset weights under real-world constraints (no short-selling, full capital allocation). The optimized Sharpe-maximizing portfolio delivered an expected annual return of 26.53% with a volatility of 24.72%, while the minimum variance portfolio achieved a lower return of 8.93% at 15.10% volatility. These results demonstrate the robustness of quantitative techniques in portfolio construction and reinforce the trade-off between return-seeking strategies and volatility minimization for different investor risk profiles.

TABLE OF CONTENT

1 INTRODUCTION

The objective of this project was to apply Modern Portfolio Theory (MPT) using Python [1] to construct an optimal investment portfolio. By analyzing historical data, calculating risk-return metrics, and leveraging optimization techniques, we aimed to identify the best portfolio allocations for different investor profiles.

2 PORTFOLIO CONSTRUCTION OVERVIEW

Five diverse stocks were selected from the S&P 500 index: **JPM**, **JNJ**, **DIS**, **BA**, and **XOM**, representing finance, healthcare, entertainment, aerospace(industrials), and energy sectors respectively. Historical adjusted close prices from May 2020 to May 2025 were collected using the yfinance library, and daily returns were calculated using the .pct_change() [2] method.

3 KEY INSIGHTS FROM MONTE CARLO SIMULATION

A Monte Carlo simulation [3] of 10,000 portfolios was performed, varying the asset weights randomly under the constraint that weights sum to 1. For each portfolio, expected return, volatility, and Sharpe ratio (with a 2% risk-free rate) were calculated. The efficient frontier was visualized, clearly highlighting the trade-off between risk and return.

4 OPTIMIZED PORTFOLIO COMPARISON AND EXPLAIN THE CHARACTERISTICS

Using scipy.optimize [4], two optimized portfolios were derived based on Modern Portfolio Theory principles:

- **Maximum Sharpe Ratio Portfolio**: Focused on maximizing risk-adjusted return.
- **Minimum Variance Portfolio**: Aimed to minimize total portfolio risk.

4.1.1 Summary of Results:

Portfolio	Expected Return	Volatility (Risk)	Sharpe Ratio
Max Sharpe Ratio	26.53%	24.72%	0.99
Minimum Variance	8.93%	15.10%	0.46

4.1.2 Portfolio Weights:

Stock	Max Sharpe Ratio	Min Variance
JPM	0.00%	0.00%
JNJ	0.00%	10.00%
DIS	0.00%	71.43%
BA	56.95%	9.52%
XOM	43.05%	9.05%

The **Maximum Sharpe Ratio Portfolio** is characterized by its objective to achieve the highest possible risk-adjusted return. It does this by allocating the majority of its capital to high-performing but volatile assets like BA (Boeing) and XOM (ExxonMobil). This results in a higher expected return (26.53%) with higher volatility (24.72%), making it a more aggressive strategy.

The **Minimum Variance Portfolio** seeks to minimize overall portfolio risk regardless of return. It allocates the largest portion of its weight to DIS (Disney), a relatively stable stock, resulting in a much lower volatility (15.10%) and moderate return (8.93%)**.** This strategy reflects a conservative investment approach**,** prioritizing stability over growth.

5 SUITABILITY FOR INVESTOR TYPES

Each portfolio suits a different investor profile:

- **Max Sharpe Ratio Portfolio**
  - Suitable for **aggressive investors** seeking **maximum return** even at the cost of higher volatility.
  - These investors are willing to take on more risk for potentially higher reward, and are likely to have a **longer investment horizon**.
- **Minimum Variance Portfolio**
  - Ideal for **conservative investors** who prioritize **capital preservation** and **lower volatility**.
  - This is a good fit for **risk-averse individuals** or those approaching retirement who value stability and predictable performance.

6 CHALLENGES ENCOUNTERED

- **Missing or inconsistent data**

Used .dropna() to clean the dataset and ensure accurate return calculations.

- **Understanding annualization formulas**

Reviewed lecture slides and used mean * 252 and cov * 252 for annual returns and risk.

- **Sharpe ratio optimization logic was confusing**

Broke it into small functions and referred to documentation for scipy.optimize.

- **Efficient Frontier plot looked too crowded**

Added color gradients and alpha transparency to improve visualization clarity.

- **Constraint setup in optimization was tricky**

Used a lambda function with 'type': 'eq' [5] to ensure weights sum to 1.

7 KEY LEARNINGS

- Learned how to collect and clean real-world financial data using yfinance.
- Understood how to calculate and interpret returns, volatility, and the Sharpe ratio.
- Discovered how optimization can improve investment decisions significantly.
- Gained practical experience in using scipy.optimize for financial problems.
- Realized the importance of diversification and risk management in portfolios.

8 CONCLUSION

Through systematic data analysis, simulation, and optimization, an efficient portfolio was constructed. The practical exposure to tools like yfinance, numpy, pandas, matplotlib [6], and scipy.optimize has enhanced my understanding of portfolio theory and financial computation, preparing me for real-world financial modeling.

9 REFERENCES

[1] p. organization, "yfinance 0.2.61," [Online]. Available: https://pypi.org/project/yfinance/. [Accessed 17 5 2025].

[2] "Python | Pandas dataframe.pct_change()," GeeksforGeeks, Sanchhaya Education Private Limited, [Online]. Available: https://www.geeksforgeeks.org/python-pandas-dataframe-pct_change/. [Accessed 17 5 2025].

[3] "What is Monte Carlo Simulation?," GeeksforGeeks, Sanchhaya Education Private Limited, [Online]. Available: https://www.geeksforgeeks.org/what-is-monte-carlo-simulation/. [Accessed 14 5 2025].

[4] "Optimization and root finding," The SciPy community, [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html. [Accessed 17 5 2025].

[5] "How to Use Python Lambda Functions," DevCademy Media Inc. DBA Real Python, [Online]. Available: https://realpython.com/python-lambda/. [Accessed 17 5 2025].

[6] "Matplotlib Tutorial," W3Schools, [Online]. Available: https://www.w3schools.com/python/matplotlib_intro.asp. [Accessed 12 5 2025].

10 APPENDIX

10.1 Stock Tickers Used

- **JPM** – JPMorgan Chase & Co. (Finance)
- **JNJ** – Johnson & Johnson (Healthcare)
- **DIS** – Walt Disney Company (Entertainment)
- **BA** – Boeing Company (Industrials)
- **XOM** – ExxonMobil Corporation (Energy)

10.2 Key Python Libraries Used

- yfinance – For downloading historical stock data.
- pandas – For data manipulation and analysis.
- numpy – For numerical calculations and portfolio metrics.
- matplotlib.pyplot – For plotting visualizations.
- scipy.optimize – For portfolio optimization using constraints.

10.3 Assumptions

- **Trading Days Per Year**: 252
- **Risk-Free Rate**: 2% (0.02)
- **Investment Horizon**: 5 years (2020-05-01 to 2025-04-30)
- **Short Selling**: Not allowed (weights constrained to [0, 1])

10.4 Portfolio Optimization Constraints

- All weights sum to **1**: sum(weights) == 1
- No short-selling: weights >= 0 and weights <= 1

10.5 Efficient Frontier Plot Highlights

- **Red Dot**: Portfolio with **maximum Sharpe ratio**
- **Blue Dot**: Portfolio with **minimum volatility**
- **Scatter Points**: 10,000 simulated portfolios via Monte Carlo

10.6 Codes

10.6.1 Task 01

```
# ----------------------#
# Task 1: Data Collection & Preprocessing
# ----------------------#

# Import necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np

# Step 1: Select 5 diverse stocks from major stock market index

# JPMorgan Chase (JPM)- Financial Sector
# Johnson & Johnson (JNJ)- Healthcare Sector
# The Walt Disney Company (DIS)- Entertainment Sector
# Boeing (BA)- Aerospace Sector
# Exxon Mobil (XOM)- Energy Sector

tickers = ['JPM', 'JNJ', 'DIS', 'BA', 'XOM']

# Step 2: Download historical stock data (5 years) with auto_adjust=True
try:
    data = yf.download(tickers, start="2020-05-01", end="2025-05-01", auto_adjust=True)
```

```
except Exception as e:
    print("Error fetching data:", e)
    raise

# Step 3:Directly extract 'Close' prices (already adjusted due to auto_adjust=True)
adj_close = data['Close']

# Step 4: Calculate Daily Returns and drop the first NaN
returns = adj_close.pct_change().dropna()

# Step 5: Display Outputs Nicely
print("\n--- Adjusted Close Prices (Last 5 Rows) ---")
display(adj_close.tail())

print("\n--- Daily Returns (Last 5 Rows) ---")
display(returns.tail())

# Step 6: Summary Info
print(f"\nAdjusted Close Data Shape: {adj_close.shape}")
print(f"Returns Data Shape: {returns.shape}")
print("\nData Info:")
display(adj_close.info())

print("\nBasic Statistics for Adjusted Close Prices:")
display(adj_close.describe())
```

10.6.2 Task 02

```
# ----------------------#
# Task 2: Expected Returns & Risk Calculation
# ----------------------#
```

```
import numpy as np

# Step 1: Calculate Annualized Mean Returns
# 252 trading days per year assumed
mean_returns = returns.mean() * 252

# Step 2: Calculate Annualized Covariance Matrix of returns
cov_matrix = returns.cov() * 252

# Step 3: Simulate a Basic Portfolio with Random Weights (must sum to 1)
weights = np.random.random(len(tickers))
weights /= np.sum(weights)

# Step 4: Calculate Expected Portfolio Return & Volatility
port_return = np.dot(weights, mean_returns)
port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

# Step 5: Display Results in Clean Tables
print(" Annualized Mean Returns (%):")
display(mean_returns.to_frame(name='Annualized Mean Return').style.format("{:.2%}"))

print(" Annualized Covariance Matrix:")
display(cov_matrix.style.format("{:.4f}"))

print(f" Simulated Portfolio Return: {port_return:.2%}")
print(f" Simulated Portfolio Volatility: {port_volatility:.2%}")

print("\n Portfolio Weights Used:")
for ticker, weight in zip(tickers, weights):
    print(f" {ticker}: {weight:.2%}")
```

10.6.3 Task 03

```
# -----------------------------#
# Task 3: Monte Carlo Simulation for Efficient Frontier
# -----------------------------#

import numpy as np
import matplotlib.pyplot as plt

# Use previously defined variables from Task 2:
# - mean_returns
# - cov_matrix
# - tickers

# Step 1: Parameters
num_portfolios = 10000
risk_free_rate = 0.02  # 2% risk-free rate as per assignment

# Step 2: Arrays to store simulation results
results = np.zeros((3, num_portfolios))  # Rows: Return, Volatility, Sharpe Ratio
weights_record = []

# Step 3: Monte Carlo Simulation Loop
for i in range(num_portfolios):
   # Step 3.1: Generate random weights that sum to 1
   weights = np.random.random(len(tickers))
   weights /= np.sum(weights)
   weights_record.append(weights)

   # Step 3.2: Calculate expected return and volatility for the portfolio
```

```
    port_return = np.dot(weights, mean_returns)
    port_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

    # Step 3.3: Calculate Sharpe Ratio
    sharpe_ratio = (port_return - risk_free_rate) / port_volatility

    # Step 3.4: Store results
    results[0, i] = port_return
    results[1, i] = port_volatility
    results[2, i] = sharpe_ratio

# Step 4: Plotting the Efficient Frontier
plt.figure(figsize=(14, 8))
scatter = plt.scatter(results[1, :], results[0, :], c=results[2, :], cmap='viridis', s=10, alpha=0.5)
plt.colorbar(scatter, label='Sharpe Ratio')
plt.title('Monte Carlo Simulation - Efficient Frontier (10,000 Portfolios)')
plt.xlabel('Portfolio Volatility (Risk)')
plt.ylabel('Expected Portfolio Return')
plt.grid(True)

# Step 5: Highlight Max Sharpe Ratio Portfolio
max_sharpe_idx = np.argmax(results[2])
plt.scatter(results[1, max_sharpe_idx], results[0, max_sharpe_idx], c='red', s=100, edgecolors='black', label='Max Sharpe Ratio')

# Step 6: Highlight Min Volatility Portfolio
min_vol_idx = np.argmin(results[1])
plt.scatter(results[1, min_vol_idx], results[0, min_vol_idx], c='blue', s=100, edgecolors='black', label='Min Volatility')


# Step 6.5: Add Efficient Frontier Line
```

```
sorted_indices = np.argsort(results[1])
sorted_vol = results[1][sorted_indices]
sorted_ret = results[0][sorted_indices]

efficient_vol = []
efficient_ret = []
max_ret_so_far = -np.inf

for vol, ret in zip(sorted_vol, sorted_ret):
    if ret > max_ret_so_far:
        efficient_vol.append(vol)
        efficient_ret.append(ret)
        max_ret_so_far = ret

plt.plot(efficient_vol, efficient_ret, 'r--', linewidth=2.5, label='Efficient Frontier')

plt.legend()
plt.show()

# Step 7: display portfolio details
print(f"\nMax Sharpe Ratio Portfolio Metrics:")
print(f"  Return: {results[0, max_sharpe_idx]:.2%}")
print(f"  Volatility: {results[1, max_sharpe_idx]:.2%}")
print(f"  Sharpe Ratio: {results[2, max_sharpe_idx]:.2f}")
print("  Weights:")
for ticker, weight in zip(tickers, weights_record[max_sharpe_idx]):
    print(f"    {ticker}: {weight:.2%}")

print(f"\nMin Volatility Portfolio Metrics:")
print(f"  Return: {results[0, min_vol_idx]:.2%}")
print(f"  Volatility: {results[1, min_vol_idx]:.2%}")
```

```
print(f"  Sharpe Ratio: {results[2, min_vol_idx]:.2f}")
print("  Weights:")
for ticker, weight in zip(tickers, weights_record[min_vol_idx]):
    print(f"    {ticker}: {weight:.2%}")
```

10.6.4 **Task 04**

```
# -----------------------------#
# Task 4: Portfolio Optimization
# -----------------------------#

from scipy.optimize import minimize
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Already defined from Task 1 ,2 & 3
# tickers, mean_returns, cov_matrix, results

risk_free_rate = 0.02  # 2% risk-free rate

# Step 1: Function to Calculate Portfolio Performance
def portfolio_metrics(weights, mean_returns, cov_matrix, risk_free_rate):
    ret = np.dot(weights, mean_returns)
    vol = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    sharpe = (ret - risk_free_rate) / vol
    return ret, vol, sharpe

# Step 2: Objective Function to Maximize Sharpe Ratio (minimize negative Sharpe)
def neg_sharpe_ratio(weights, mean_returns, cov_matrix, risk_free_rate):
    return -portfolio_metrics(weights, mean_returns, cov_matrix, risk_free_rate)[2]
```

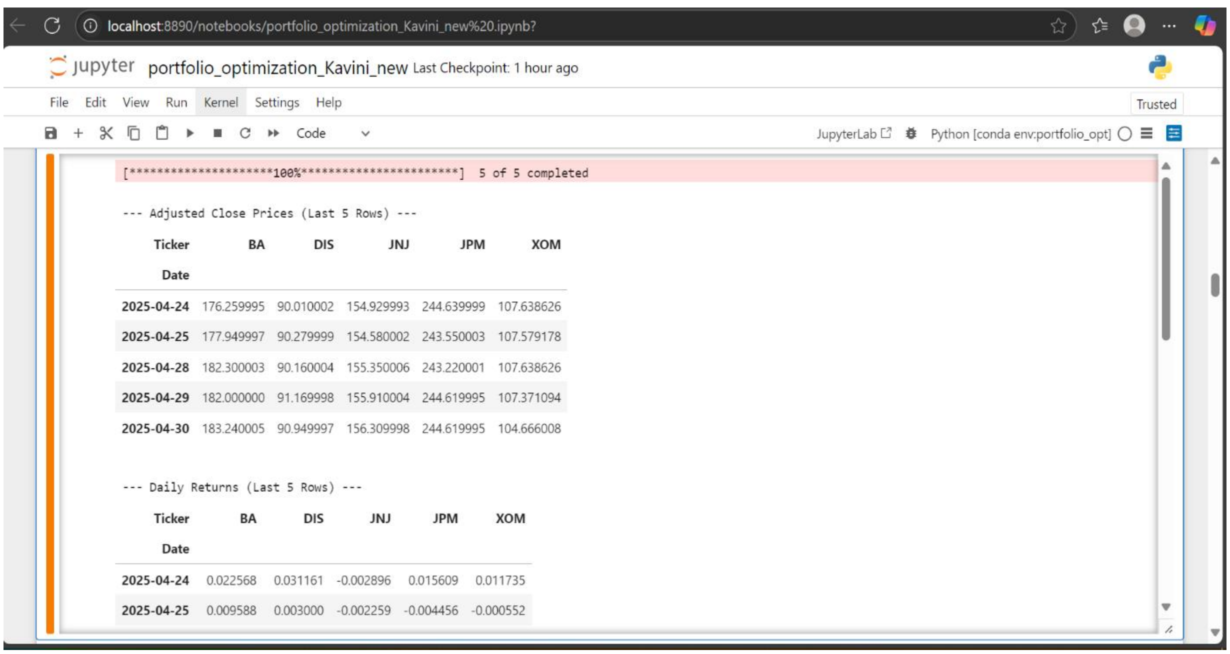
```
# Step 3: Objective Function for Minimum Variance Portfolio
def portfolio_volatility(weights, mean_returns, cov_matrix):
    return portfolio_metrics(weights, mean_returns, cov_matrix, 0)[1]

# Step 4: Constraints & Bounds
constraints = {'type': 'eq', 'fun': lambda x: np.sum(x) - 1}  # Weights must sum to 1
bounds = tuple((0, 1) for _ in tickers)  # No short selling

# Step 5: Initial Guess (Equal allocation)
init_guess = len(tickers) * [1. / len(tickers)]

# Step 6: Optimization for Maximum Sharpe Ratio Portfolio
opt_sharpe = minimize(neg_sharpe_ratio, init_guess,
                      args=(mean_returns, cov_matrix, risk_free_rate),
                      method='SLSQP', bounds=bounds, constraints=constraints)

# Step 7: Optimization for Minimum Variance Portfolio
opt_min_var = minimize(portfolio_volatility, init_guess,
                       args=(mean_returns, cov_matrix),
                       method='SLSQP', bounds=bounds, constraints=constraints)

# Step 8: Get Optimized Metrics
max_sharpe_return, max_sharpe_volatility, max_sharpe_ratio = portfolio_metrics(opt_sharpe.x, mean_returns, cov_matrix, risk_free_rate)
min_var_return, min_var_volatility, min_var_sharpe = portfolio_metrics(opt_min_var.x, mean_returns, cov_matrix, risk_free_rate)

# Step 9: Display Optimized Portfolios
comparison_df = pd.DataFrame({
    'Stock': tickers,
    'Max Sharpe Ratio Weights': opt_sharpe.x,
```

```
    'Min Variance Weights': opt_min_var.x
})

portfolio_summary = pd.DataFrame({
    'Portfolio': ['Max Sharpe Ratio', 'Min Variance'],
    'Expected Return': [max_sharpe_return, min_var_return],
    'Volatility': [max_sharpe_volatility, min_var_volatility],
    'Sharpe Ratio': [max_sharpe_ratio, min_var_sharpe]
})

# Step 10: Display Optimized Portfolios Table
print("\nOptimized Portfolio Allocations:")
display(comparison_df.style.format({
    'Max Sharpe Ratio Weights': "{:.2%}",
    'Min Variance Weights': "{:.2%}"
}))

# Step 11: Display Portfolio Performance Summary Table
print("\nPortfolio Performance Summary:")
display(portfolio_summary.style.format({
    'Expected Return': "{:.2%}",
    'Volatility': "{:.2%}",
    'Sharpe Ratio': "{:.2f}"
}))

# Step 12: Plot Efficient Frontier and mark optimized points
plt.figure(figsize=(12, 8))
plt.scatter(results[1, :], results[0, :], c=results[2, :], cmap='viridis', s=10, alpha=0.5)
plt.colorbar(label='Sharpe Ratio')
plt.scatter(max_sharpe_volatility, max_sharpe_return, c='red', s=100, edgecolors='black',
label='Max Sharpe Ratio')
```

```
plt.scatter(min_var_volatility, min_var_return, c='blue', s=100, edgecolors='black', label='Min Variance')
plt.xlabel('Volatility (Risk)')
plt.ylabel('Expected Return')
plt.title('Optimized Portfolios on Efficient Frontier')
plt.legend()
plt.grid(True)
plt.show()
```

10.7 Outputs (Screenshots)

10.7.1 Task 01


localhost:8890/notebooks/portfolio_optimization_Kavini_new%20.ipynb?

jupyter portfolio_optimization_Kavini_new Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

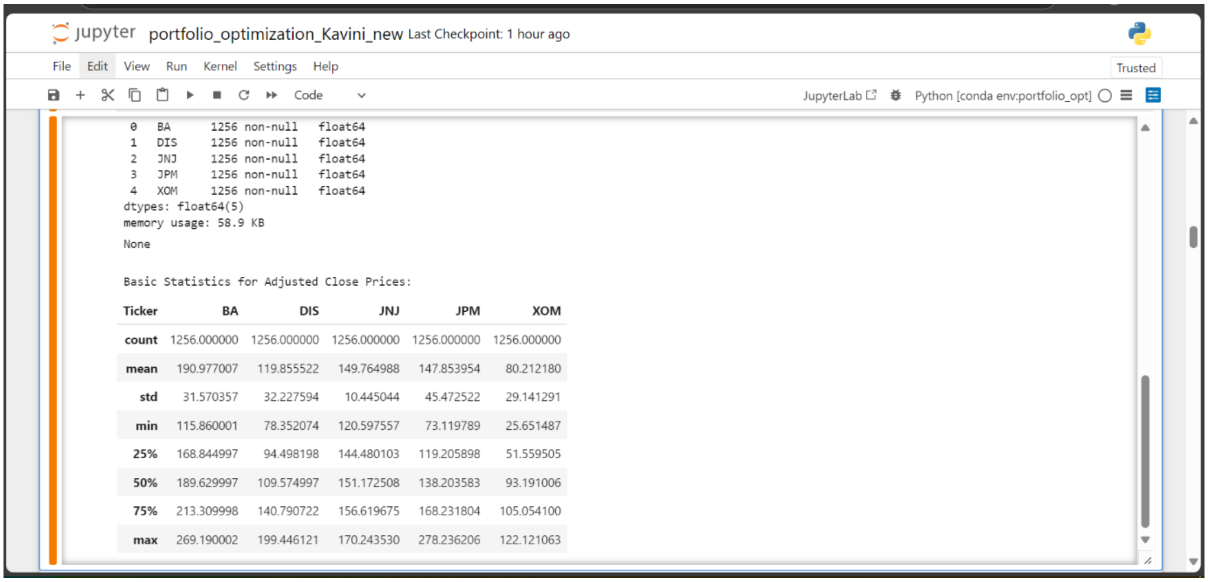
Code JupyterLab Python [conda env:portfolio_opt]

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

--- Adjusted Close Prices (Last 5 Rows) ---
```

Ticker	BA	DIS	JNJ	JPM	XOM
Date					
2025-04-24	176.259995	90.010002	154.929993	244.639999	107.638626
2025-04-25	177.949997	90.279999	154.580002	243.550003	107.579178
2025-04-28	182.300003	90.160004	155.350006	243.220001	107.638626
2025-04-29	182.000000	91.169998	155.910004	244.619995	107.371094
2025-04-30	183.240005	90.949997	156.309998	244.619995	104.666008

```
--- Daily Returns (Last 5 Rows) ---
```

Ticker	BA	DIS	JNJ	JPM	XOM
Date					
2025-04-24	0.022568	0.031161	-0.002896	0.015609	0.011735
2025-04-25	0.009588	0.003000	-0.002259	-0.004456	-0.000552



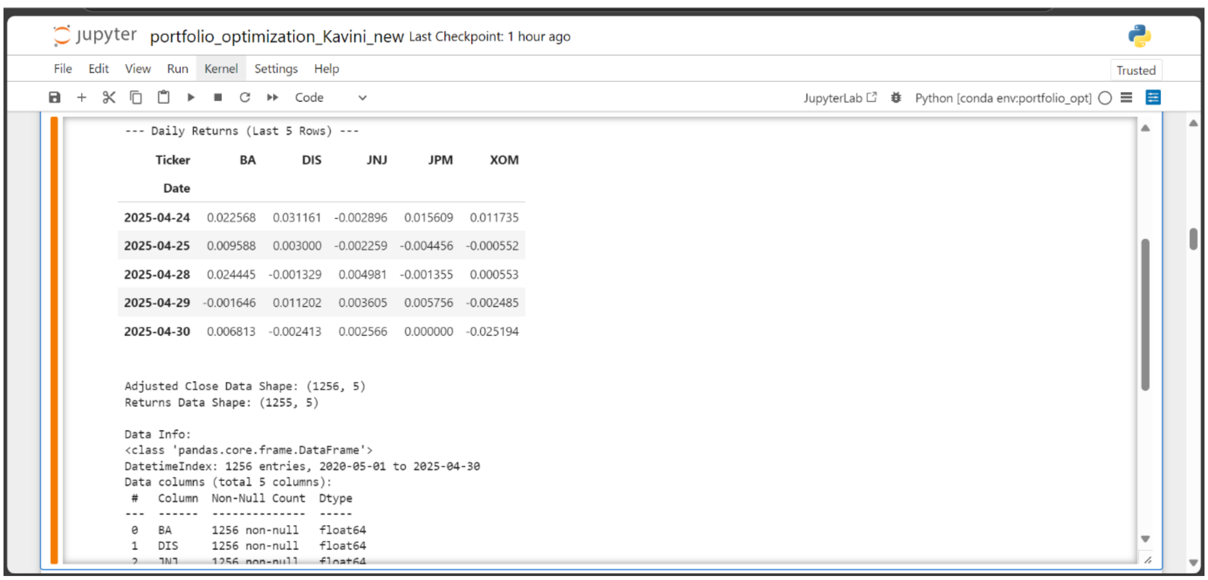
jupyter portfolio_optimization_Kavini_new Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python [conda env:portfolio_opt]

```
 0   BA      1256 non-null   float64
 1   DIS     1256 non-null   float64
 2   JNJ     1256 non-null   float64
 3   JPM     1256 non-null   float64
 4   XOM     1256 non-null   float64
dtypes: float64(5)
memory usage: 58.9 KB
None

Basic Statistics for Adjusted Close Prices:
```

Ticker	BA	DIS	JNJ	JPM	XOM
count	1256.000000	1256.000000	1256.000000	1256.000000	1256.000000
mean	190.977007	119.855522	149.764988	147.853954	80.212180
std	31.570357	32.227594	10.445044	45.472522	29.141291
min	115.860001	78.352074	120.597557	73.119789	25.651487
25%	168.844997	94.498198	144.480103	119.205898	51.559505
50%	189.629997	109.574997	151.172508	138.203583	93.191006
75%	213.309998	140.790722	156.619675	168.231804	105.054100
max	269.190002	199.446121	170.243530	278.236206	122.121063



jupyter portfolio_optimization_Kavini_new Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help — Trusted

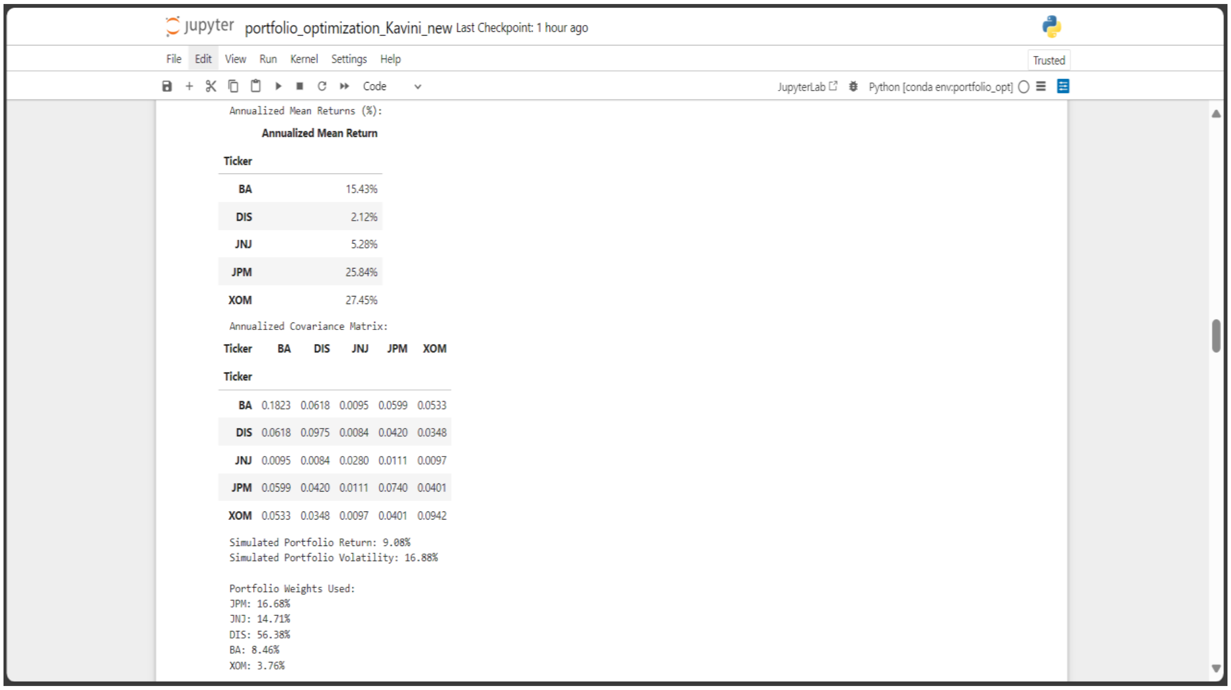
Code — JupyterLab — Python [conda env:portfolio_opt]

```
--- Daily Returns (Last 5 Rows) ---
```

Ticker Date	BA	DIS	JNJ	JPM	XOM
2025-04-24	0.022568	0.031161	-0.002896	0.015609	0.011735
2025-04-25	0.009588	0.003000	-0.002259	-0.004456	-0.000552
2025-04-28	0.024445	-0.001329	0.004981	-0.001355	0.000553
2025-04-29	-0.001646	0.011202	0.003605	0.005756	-0.002485
2025-04-30	0.006813	-0.002413	0.002566	0.000000	-0.025194

```
Adjusted Close Data Shape: (1256, 5)
Returns Data Shape: (1255, 5)

Data Info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1256 entries, 2020-05-01 to 2025-04-30
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   BA      1256 non-null   float64
 1   DIS     1256 non-null   float64
 2   JNJ     1256 non-null   float64
```

10.7.2 Task 02



jupyter portfolio_optimization_Kavini_new Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python [conda env:portfolio_opt]

```
Annualized Mean Returns (%):
```

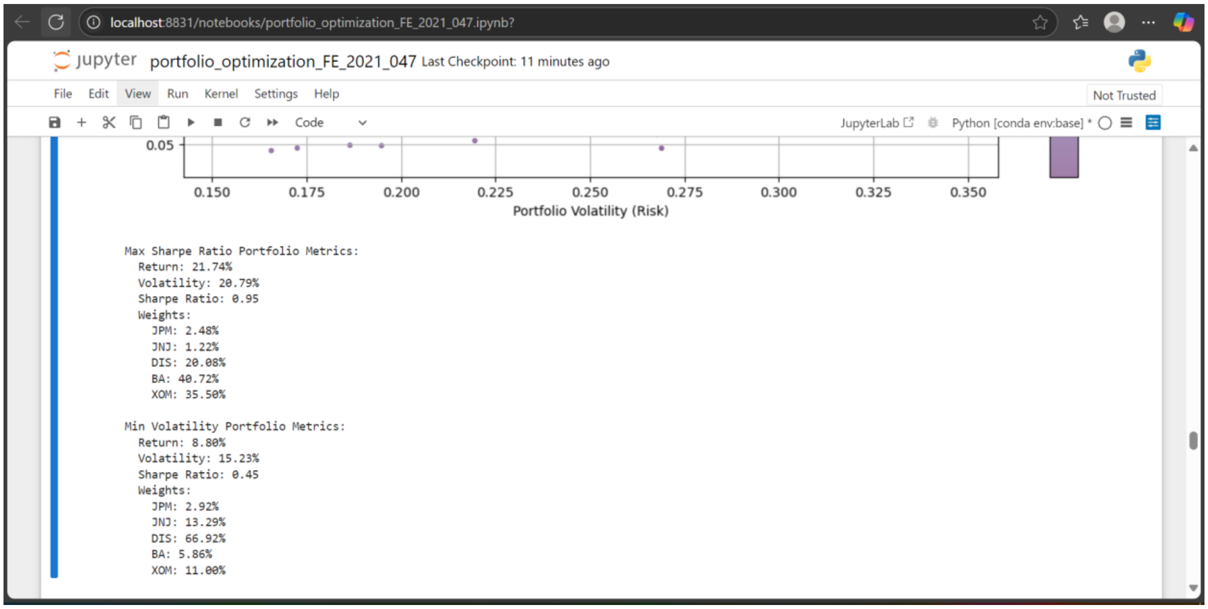
Ticker	Annualized Mean Return
BA	15.43%
DIS	2.12%
JNJ	5.28%
JPM	25.84%
XOM	27.45%

```
Annualized Covariance Matrix:
```

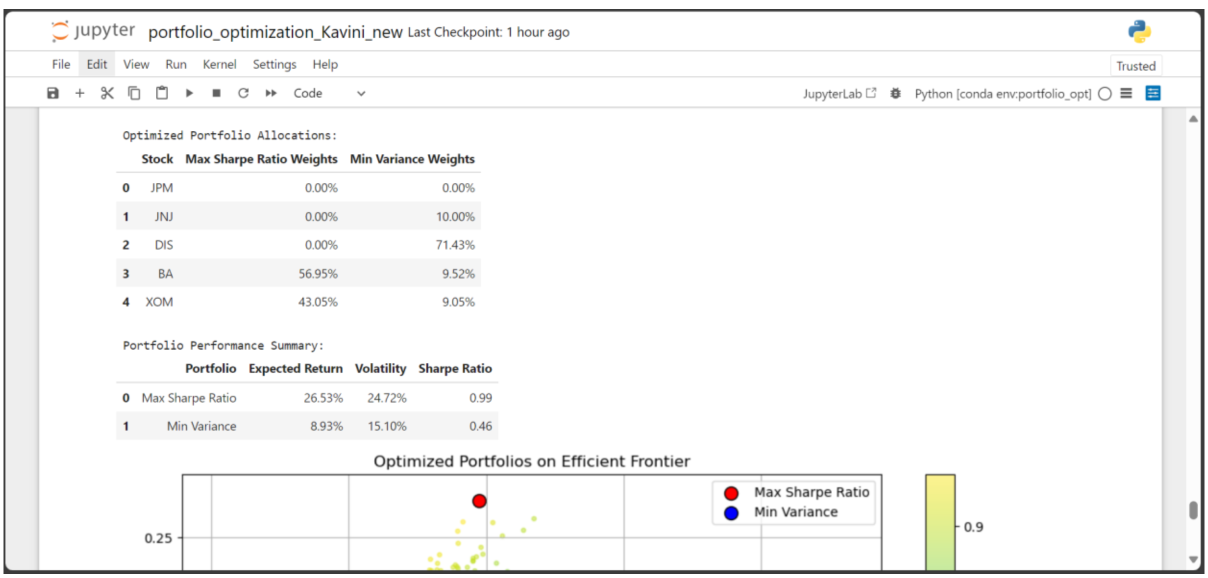
Ticker	BA	DIS	JNJ	JPM	XOM
BA	0.1823	0.0618	0.0095	0.0599	0.0533
DIS	0.0618	0.0975	0.0084	0.0420	0.0348
JNJ	0.0095	0.0084	0.0280	0.0111	0.0097
JPM	0.0599	0.0420	0.0111	0.0740	0.0401
XOM	0.0533	0.0348	0.0097	0.0401	0.0942

```
Simulated Portfolio Return: 9.08%
Simulated Portfolio Volatility: 16.88%

Portfolio Weights Used:
JPM: 16.68%
JNJ: 14.71%
DIS: 56.38%
BA: 8.46%
XOM: 3.76%
```

10.7.3 Task 03



10.7.4 Task 04



	Stock	Max Sharpe Ratio Weights	Min Variance Weights
0	JPM	0.00%	0.00%
1	JNJ	0.00%	10.00%
2	DIS	0.00%	71.43%
3	BA	56.95%	9.52%
4	XOM	43.05%	9.05%

	Portfolio	Expected Return	Volatility	Sharpe Ratio
0	Max Sharpe Ratio	26.53%	24.72%	0.99
1	Min Variance	8.93%	15.10%	0.46


jupyter
portfolio_optimization_Kavini_new Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python [conda env:portfolio_opt]
1 Min Variance 8.93% 15.10% 0.46
Optimized Portfolios on Efficient Frontier
Max Sharpe Ratio
Min Variance
Expected Return
Volatility (Risk)
Sharpe Ratio
0.05
0.10
0.15
0.20
0.25
0.15
0.20
0.25
0.30
0.35
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
[ ]: