Portfolio optimization : Markowitz's Mean-Variance Optimization technique using Python_Pyportfolioopt package

Yassine HALLAL

Master's graduate in Applied Finance and Banking at the Cadi Ayyad university, Marrakech

Introduction :

In this work, we have conducted portfolio optimization using the mean-variance optimization technique implemented through the powerful pyportfolioopt library in **Python**.

Mean-variance optimization is based on **Harry Markowitz**'s 1952 classic paper, which spearheaded the transformation of portfolio management from an art into a science. The key insight is that by combining assets with different expected returns and volatilities, one can decide on a mathematically optimal allocation.

The primary objective of this work is to **construct and analyze three distinct portfolios** with varying optimization goals.

- **Minimum Volatility** : The first portfolio aimed to minimize volatility, emphasizing stability and capital preservation.
- **Unconstrained Maximum Sharpe ratio** : The second portfolio, unconstrained in its approach, focused on maximizing the Sharpe ratio, which balances risk and reward.
- **Constrained Max Sharpe ratio Portfolio (L2 Regularization)** : Finally, the third portfolio incorporated L2 regularization as a constraint, striving to strike a balance between risk management and performance optimization.

To evaluate the effectiveness of these portfolios, we plotted the efficient frontier, which represents the set of portfolios that offer the highest expected return for each level of risk. This visualization provides valuable insights into the risk-return trade-offs and allows for a comprehensive comparison of the three optimized portfolios.

The analysis and comparison of these three portfolios on the efficient frontier enable us to understand the impact of different optimization strategies on risk, return, and the overall portfolio composition.

Data informations :

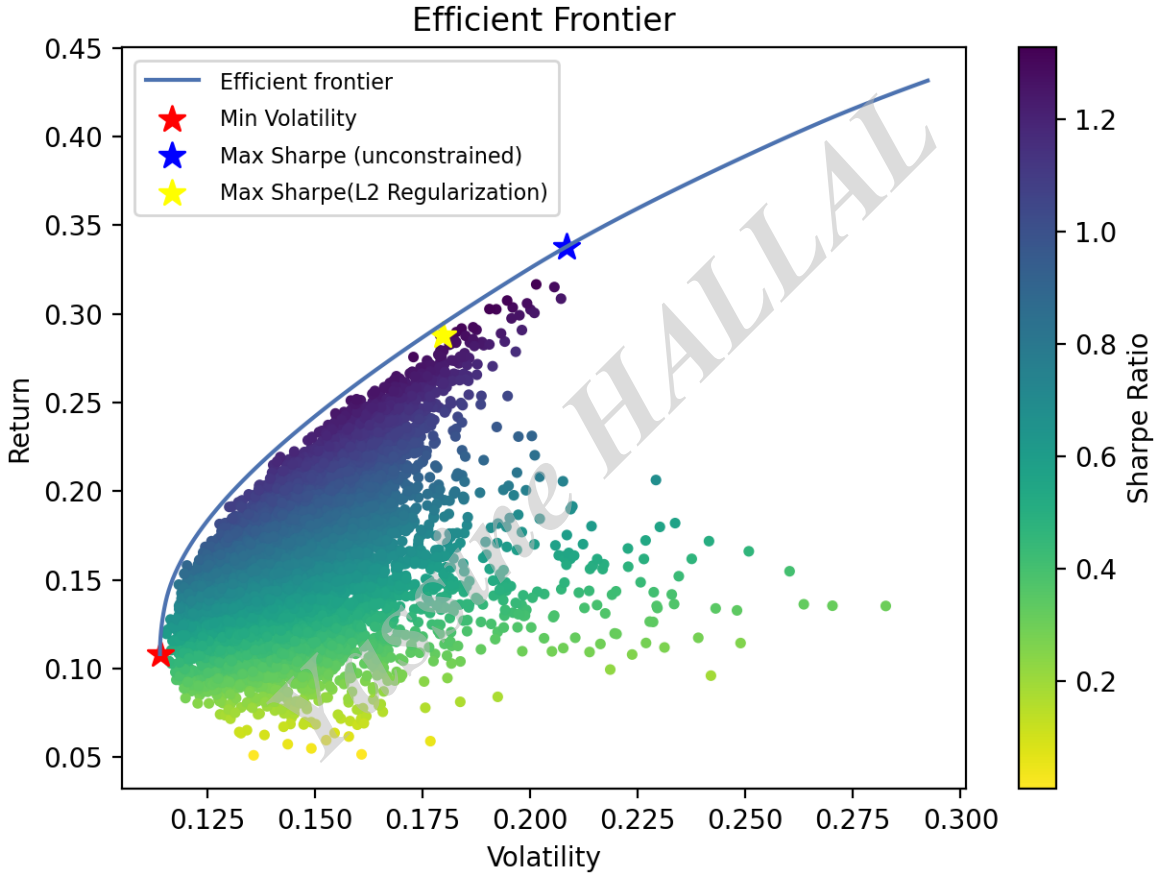
The financial data used corresponds to 10 US companies operating in different sectors, The date ranges from **01 January 2015** to **01 January 2020.** :

- **Apple Inc. (AAPL)** - Technology sector
- **Johnson & Johnson (JNJ)** - Healthcare sector
- **Procter & Gamble Co. (PG)** - Consumer goods sector
- **JPMorgan Chase & Co. (JPM)** - Financial sector
- **Exxon Mobil Corporation (XOM)** - Oil and gas sector

- **Amazon.com Inc. (AMZN)** - E-commerce/Technology sector
- **Coca-Cola Company (KO)** - Beverage sector
- **Microsoft Corporation (MSFT)** - Technology sector
- **Barrick Gold Corporation (GOLD)** - Gold mining sector
- **Chevron Corporation (CVX)** - Oil and gas sector

we are going to import the data from Yahoo Finance using the corresponding python package (yfinance), and we are going to retrieve the **Adjusted Close** price as it takes into consideration dividends and stock splits.

Expected result



Importing libraries

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

from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import expected_returns, risk_models
from pypfopt.risk_models import sample_cov
from pypfopt import plotting
from pypfopt.objective_functions import L2_reg

import warnings
warnings.filterwarnings('ignore')
```

Importing data


```
# Importing prices data

tickers = ['AAPL', 'JNJ', 'PG', 'JPM', 'XOM', 'AMZN', 'KO', 'MSFT', 'GOLD', 'CVX']
start_date = '2015-01-01'
end_date = '2020-01-01'

# Downloading data from YF
data = yf.download(tickers = tickers, start = start_date, end = end_date)
```

```
# Retrieve the 'Adj Close' column
# Takes into consideration dividends, stock splits.

prices = data['Adj Close']

# Setting 'Date' as the index
prices = pd.DataFrame(prices)
prices.reset_index(inplace = True)
prices.set_index('Date', inplace = True)
```

Calculating returns for each asset:

```
returns = np.log(prices / prices.shift(1))
returns
```

```
##                  AAPL      AMZN       CVX  ...      MSFT        PG       XOM
## Date                                       ...
## 2015-01-02       NaN       NaN       NaN  ...       NaN       NaN       NaN
## 2015-01-05 -0.028577 -0.020731 -0.040792  ... -0.009239 -0.004766 -0.027743
## 2015-01-06  0.000094 -0.023098 -0.000463  ... -0.014786 -0.004566 -0.005330
```

```
## 2015-01-07  0.013925  0.010544 -0.000834  ...  0.012625  0.005232  0.010081
## 2015-01-08  0.037702  0.006813  0.022625  ...  0.028994  0.011371  0.016508
## ...              ...       ...       ...  ...       ...       ...       ...
## 2019-12-24  0.000950 -0.002116  0.000083  ... -0.000190  0.002559 -0.003849
## 2019-12-26  0.019646  0.043506  0.002158  ...  0.008163  0.000000  0.001570
## 2019-12-27 -0.000379  0.000551 -0.002491  ...  0.001826  0.006924 -0.003428
## 2019-12-30  0.005917 -0.012328 -0.003748  ... -0.008656 -0.012931 -0.005884
## 2019-12-31  0.007280  0.000514  0.005492  ...  0.000698  0.003449  0.004308
##
## [1258 rows x 10 columns]
```

Prices graph :

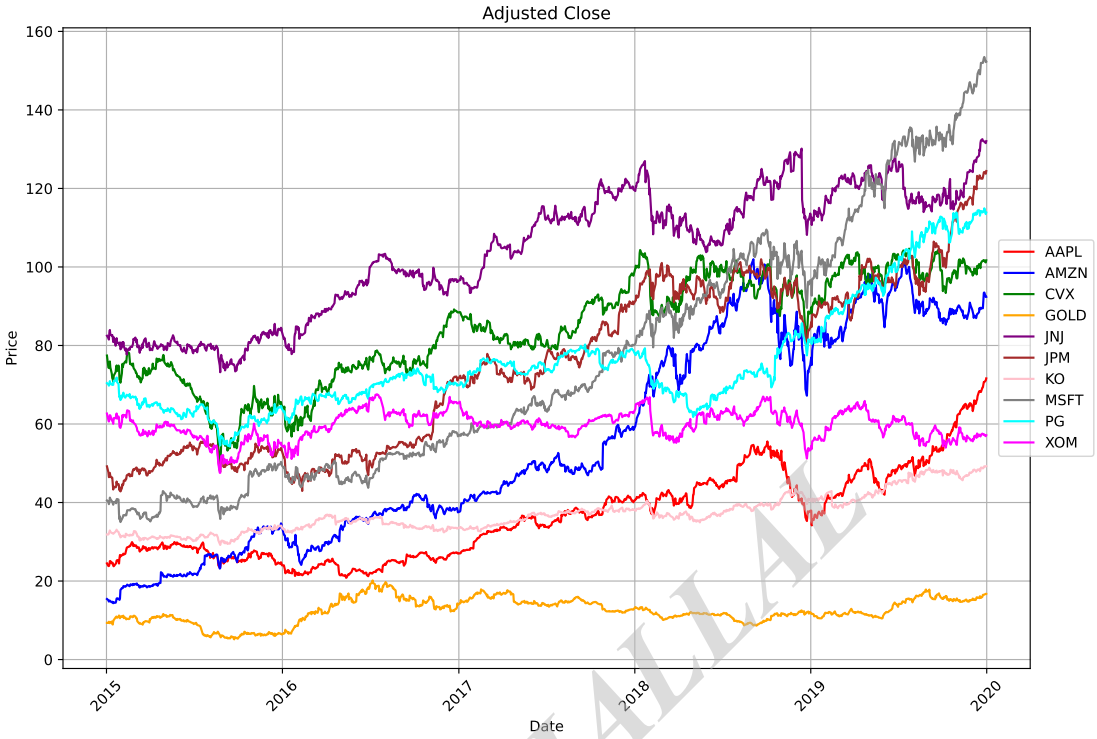
```
plt.figure(figsize=(12, 8))

colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'pink', 'gray', 'cyan', 'magenta']

for i, column in enumerate(prices.columns):
    plt.plot(prices.index, prices[column], label=column, color = colors[i])

plt.title('Adjusted Close')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='center left', bbox_to_anchor=(0.96, 0.5))
plt.xticks(rotation = 45)
```

```
plt.grid(True)
plt.show()
```



Assigning random, equal weights for each asset

```
weights = np.array([0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1])
```

Calculate expected returns

We are using the **mean historical return Method**.

Other methods :

- Exponentially-weighted mean of (daily) historical returns

- Returns estimate using the Capital Asset Pricing Model

```
mu = expected_returns.mean_historical_return(prices, compounding=True, frequency=252)
```

Calculate the risk Model

We are going to use the **Sample Annualized Covariance Matrix**

Other risk models :

- **Covariance Shrinkage Ledoit-Wolf.**
- **semicovariance matrix**

- **exponentially-weighted covariance matrix**
- **Oracle Approximating Shrinkage estimate**

```
S = sample_cov(prices, frequency=252)
```

Calculating the annual returns of the portfolio

$$R_p = \sum_{i=1}^{n} W_i Ri$$

Other important mathematical formulas :

- Variance :

$$s^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}$$

- Standard Deviation :

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}}$$

- Covariance :

$$cov_{i,j} = \frac{\sum_{i=1,j=1}^{N}(x_i - \bar{x_i})(x_j - \bar{x_j})}{N-1}$$

- Correlation :

$$\rho(R_i, Rj) = \frac{COV(R_i, Rj)}{\sigma_i \sigma_j}$$

Portfolio Variance

$$\sigma^2(Rp) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j COV(R_i, R_j)$$

Portfolio standard deviation

$$\sigma(R_p) = \sqrt{\sigma^2(R_p)}$$

Risk Free rate is 5%

Sharpe ratio :

$$Sharpe = \frac{R_p - R_f}{\sigma_p}$$

with : R_p : Portfolio expected return, R_f : Risk free rate, σ_p : Portfolio volatility/Standard deviation.

```
# Expected Annual return
port_annual_ret = round(np.sum(returns.mean() * weights) * 252, ndigits = 4)

# Portfolio Variance
port_variance = round(np.dot(weights.T, np.dot(S, weights)), ndigits=4)

# Portfolio volatility / Risk
port_volatility = round(np.sqrt(port_variance), ndigits= 4)

# Sharpe Ratio (rf = 5%)
risk_free_rate = 0.05
sharpe_ratio = str(round((port_annual_ret - risk_free_rate)/port_volatility, 3))
```

Initial portfolio metrics

```
##                    Metric   Value
## 0  Expected annual return  14.56%
## 1       Annual Volatility  13.34%
## 2            Sharpe Ratio   0.717
```

Minimum Volatility Portfolio

```
# Minimum Volatility Portfolio

ef = EfficientFrontier(mu, S)
weights = ef.min_volatility()
cleaned_min_vol = ef.clean_weights()

min_vol_perf = ef.portfolio_performance(verbose=True, risk_free_rate=0.05)
```

```
## Expected annual return: 10.8%
## Annual volatility: 11.4%
## Sharpe Ratio: 0.51
```


Maximum Sharpe Portfolio

```
# Maximum Sharpe ratio with NO Constraints

ef = EfficientFrontier(mu, S)
weights = ef.max_sharpe(risk_free_rate=0.05)
cleaned_no_const = ef.clean_weights()

max_s = ef.portfolio_performance(verbose=True, risk_free_rate=0.05)
```

```
## Expected annual return: 33.8%
## Annual volatility: 20.8%
## Sharpe Ratio: 1.38
```

L2 regularization portfolio (Constrained)

```
# Max Sharpe Ratio with constraints :
# 1) All weights should add up to 1
# 2) L2_regularization in order to reduce the zero weights

ef = EfficientFrontier(mu, S)
ef.add_objective(L2_reg, gamma=2)

ef.add_constraint(lambda w : w[0] + w[1] + w[2] + w[3] + w[4] + w[5] + w[6] + w[7] + w[8] + w[9] == 1)
ef.add_constraint(lambda w : w >= 0)

weights = ef.max_sharpe(risk_free_rate=0.05)
cleaned_L2_reg = ef.clean_weights()

L2_perf = ef.portfolio_performance(verbose=True, risk_free_rate=0.05)
```

```
## Expected annual return: 28.8%
## Annual volatility: 18.0%
## Sharpe Ratio: 1.32
```

The efficient Frontier

```
# Define the EfficientFrontier object
ef = EfficientFrontier(mu, S)
ef_constr = EfficientFrontier(mu, S, weight_bounds=(0,1))

# Portfolio that minimizes volatility
ef_min_volatility = ef.deepcopy()
weights_min_volatility = ef_min_volatility.min_volatility()

# Max Sharpe ratio Portfolio
ef_no_const = ef.deepcopy()
weights_no_const = ef_no_const.max_sharpe(risk_free_rate=0.05)

# L2 regularization portfolio
ef_L2 = ef_constr.deepcopy()
ef_L2.add_objective(L2_reg, gamma=2)
ef_L2.add_constraint(lambda w : w[0] + w[1] + w[2] + w[3] + w[4] + w[5] + w[6] + w[7] + w[8] + w[9] == 
ef_L2.add_constraint(lambda w : w >= 0)
weights_L2 = ef_L2.max_sharpe(risk_free_rate=risk_free_rate)
```

```
# Plot the efficient frontier with the portfolios
fig, ax = plt.subplots()
plotting.plot_efficient_frontier(ef, ax=ax, show_assets=False)

# Generate random portfolios
n_samples = 10000
w_random = np.random.dirichlet(np.ones(ef.n_assets), n_samples)
rets_random = w_random.dot(ef.expected_returns)
```

```
stds_random = np.sqrt(np.diag(w_random @ ef.cov_matrix @ w_random.T))
sharpes_random = (rets_random - risk_free_rate) / stds_random
sc = ax.scatter(stds_random, rets_random, marker=".", c=sharpes_random, cmap="viridis_r")

# Plot the portfolio that minimizes volatility
ret_vol, std_vol, _ = ef_min_volatility.portfolio_performance()
ax.scatter(std_vol, ret_vol, marker="*", s=100, c="r", label="Min Volatility")
```

```
# Plot the max sharpe ratio portfolio
ret_no_const, std_no_const, _ = ef_no_const.portfolio_performance()
ax.scatter(std_no_const, ret_no_const, marker="*", s=100, c="blue", label="Max Sharpe (unconstrained)")
```

```
# Plot the max sharpe ratio portfolio with L2 regularization
ret_L2, std_L2, _ = ef_L2.portfolio_performance()
ax.scatter(std_L2, ret_L2, marker="*", s=100, c="Yellow", label="Max Sharpe(L2 Regularization)")
```

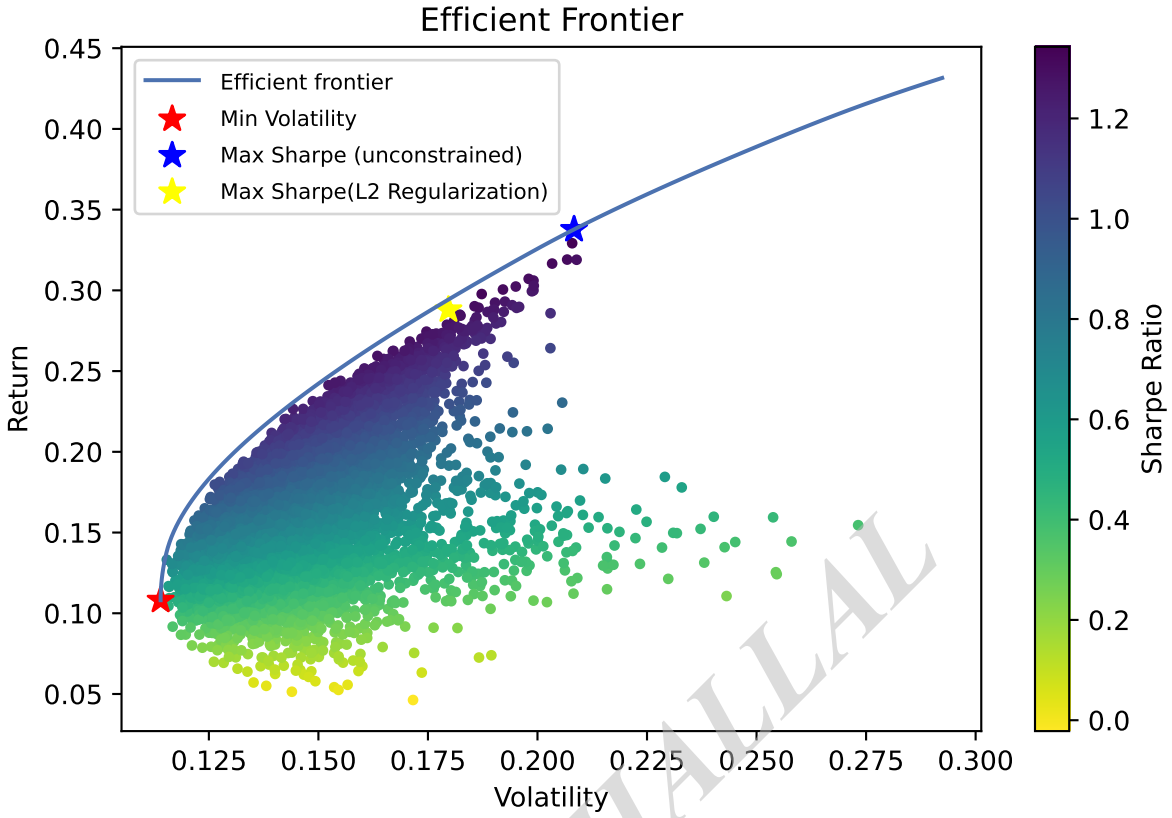
```
# Output
cbar = plt.colorbar(sc)
cbar.set_label('Sharpe Ratio')

ax.set_title("Efficient Frontier")
```

```
legend = ax.legend()

for text in legend.get_texts(): # Reducing the size of the legend
    text.set_fontsize(8)

plt.tight_layout()
plt.show()
```



Recap :

Ticker	Initial_portfolio	Min_Volatility	Max Sharpe	L2 Regularization
AAPL	0.1	0.03361	0	0.15684
AMZN	0.1	0.01409	0.5168	0.32453
CVX	0.1	0	0	0
GOLD	0.1	0.06439	0.08078	0.06288
JNJ	0.1	0.20249	0	0.03816
JPM	0.1	0.09089	0.17737	0.12983
KO	0.1	0.35107	0	0.03287
MSFT	0.1	0	0.22505	0.21376
PG	0.1	0.16546	0	0.04114
XOM	0.1	0.078	0	0
Expected Annual Return	14.56%	10.8%	33.8%	28.8%
Annual volatility	13.34%	11.4%	20.8%	18.0%
Sharpe Ratio	0.717	0.51	1.38	1.32

Comments :

Based on these results, we can conclude that :

- **Comment 1**: The Min_Volatility portfolio allocates a non-zero weight to AAPL, AMZN, GOLD, JNJ, JPM, MSFT, and PG. With only 2 zero allocations for CVX and MSFT respectively, which makes it a more **diversified portfolio** than the **Max Sharpe portfolio**. Therefore, it reduce the impact of any individual security's performance on the overall portfolio.
- **Comment 2**: The Max Sharpe portfolio has **zero weights** for AAPL, CVX, JNJ, KO, PG and XOM. It appears to have a **higher allocation** to AMZN (51.68%), MSFT (22.5%) and JPM (17.73%) compared to the Min_Volatility portfolio. Therefore, it increases the impact of any individual security's performance on the overall portfolio **(more concentrated).**
- **Comment 3**: The L2 regularization portfolio demonstrates a notable improvement by reducing the number of zero weights (only 2 zero weights), which indicates a more balanced allocation of investments across the portfolio. This regularization technique penalizes **extreme weightings**, encouraging a more even distribution of investments across the portfolio. This can lead to improved risk management and stability while still allowing for the potential for attractive returns.

Future Works :

In this work, we worked with the Mean-Variance optimization technique. Note that there are several other optimization techniques used in portfolio management beyond mean-variance optimization. for example :

- The Black-Litterman (BL) model

- Hierarchical Risk Parity

In future work, we plan to expand our portfolio optimization projects to include the **hierarchical risk parity model** and the **Black-Litterman model**, as well as applying the optimization techniques without the use of any external library.

References :

Biblio

- Stewart, S. D., Piros, C. D., & Heisler, J. C. (2019). Portfolio Management: Theory and Practice. John Wiley & Sons.
- Brugière, P. (2020). Quantitative Portfolio Management with applications in Python. In Springer texts in business and economics. Springer International Publishing.
- PyPortfolioOpt documentation.

Web :

- Various YouTube channels.
- GitHub Repos.
- And of course, StackOverflow.