Portfolio Optimization and Backtesting Using Python: A Pragmatic Approach

Ascalone Sara - 30627
De Col Emilio - 30185
Malamisura Federica - 30828
Radice Jacopo - 30565
Rossi Lisa - 31024

June 3, 2024

Abstract

This paper explores the application of Modern Portfolio Theory (MPT) and Monte Carlo simulations to optimize and backtest a portfolio of various financial assets. By employing Python for data analysis and visualization, we aim to construct a portfolio that maximizes the Sharpe Ratio and adapt it to changing market conditions through rolling window optimization. The analysis integrates data cleaning, correlation analysis, risk-free rate calculation, and practical implications of the results. This study provides a comprehensive examination of the methods used, the execution of the analysis, and the results obtained.

1 Introduction

Portfolio optimization is a cornerstone of modern finance, offering investors a method to allocate assets in a manner that maximizes returns for a given level of risk. Markowitz's Modern Portfolio Theory (MPT) provides the theoretical foundation for this process, emphasizing diversification and quantifying risk through variance. This paper employs MPT, Monte Carlo simulations, and rolling window optimization to analyze and optimize a portfolio of various financial assets. The approach integrates extensive data cleaning and standardization, risk-free rate calculation, and back-testing to ensure robustness and practical applicability.

2 Data Loading and Cleaning

The analysis begins by importing essential libraries and loading of multiple CSV files into pandas DataFrames, each representing distinct financial markets or indices such as commodities, equities, and bonds. This foundational step ensures accurate and structured data import. Consistency is maintained across datasets by standardizing column names and converting numerical columns to a uniform format—replacing commas with periods and converting to numeric types. This meticulous standardization prevents discrepancies during subsequent analytical procedures and calculations.

Listing 1: Data Loading and Cleaning

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.optimize as sco
from typing import Tuple
from numba import njit, prange

# Load data
Energy = pd.read_csv('Commodities/Energy.csv', delimiter=';')
Gold = pd.read_csv('Commodities/UBS␣ETF␣GOLD␣H.␣EUR.csv', delimiter=';'
    )
# Additional data loading lines...

# Rename columns
new_column_names = {
    'Open␣Price': 'Open',
    'High␣Price': 'High',
    'Low␣Price': 'Low',
    'Last␣Price': 'Close',
    'Volume': 'Volume'
}
dataframes = [Energy, Gold, ...]
for df in dataframes:
    df.rename(columns=new_column_names, inplace=True)

# Convert ',' to '.' and convert to numeric
for df in dataframes:
    for col in df.columns:
        df[col] = df[col].astype(str).str.replace(',', '.')
        df[col] = pd.to_numeric(df[col], errors='coerce').round(4)
        df['Volume'] = df['Volume'].fillna(0).astype(int)
```

3 Correlation Analysis

The correlation matrix of the initial 26 ETFs revealed excessive collinearity, which can hinder the effectiveness of the Markowitz model. To address this issue, a correlation analysis was performed, selecting ETFs with correlations below 0.8 while maintaining geographic diversity.

Listing 2: High Correlation based Cleaning

```

def load_data(merged_close_data_path, corr_matrix_path):
    try:
        merged_close_data = pd.read_csv(merged_close_data_path,
            index_col=0)
        corr_matrix = pd.read_csv(corr_matrix_path, index_col=0)
        return merged_close_data, corr_matrix
    except Exception as e:
        print(f"Error␣loading␣data:␣{e}")
        raise

def analyze_and_drop_highly_correlated(corr_matrix, threshold=0.8):
    to_drop = set()
    for i in range(len(corr_matrix.columns)):
        for j in range(i + 1, len(corr_matrix.columns)):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                to_drop.add(corr_matrix.columns[j])

    return list(to_drop)

def create_final_dataframe(merged_close_data, to_drop):
    merged_close_data_final = merged_close_data.drop(columns=to_drop)
    return merged_close_data_final

def main():
    merged_close_data_path = "merged_close_data.csv"
    corr_matrix_path = "corr_matrix.csv"

    merged_close_data, corr_matrix = load_data(merged_close_data_path,
        corr_matrix_path)

    to_drop = analyze_and_drop_highly_correlated(corr_matrix, threshold
        =0.8)

    merged_close_data_final = create_final_dataframe(merged_close_data,
         to_drop)

    merged_close_data_final.to_csv("merged_close_data_final.csv", index
        =True)
    print("Final␣DataFrame␣saved␣to␣'merged_close_data_final.csv'")

if __name__ == "__main__":
    main()
```

4 Risk-Free Rate Calculation

The risk-free rate, essential for calculating the Sharpe Ratio, is derived as a weighted average from multiple countries. This process involves standardizing the risk-free rate data, calculating country-specific and regional means, and applying weights based on each rate's frequency in the dataset for greater accuracy and representation.

Listing 3: Risk-Free calculation process

```
# Loading Risk-Free Rate Data from CSV
Countries_RF = pd.read_csv('Risk_Free.csv', delimiter=';')

# Create an empty list to store the country names
names = []

# Extract the country names
for col in Countries_RF.columns[1:]:
    name = col.split('␣')[0]
    names.append(name)

# Create a list to store the means of the risk-free rates
means = []

# Calculate the mean of the risk-free rates for each country
for col in Countries_RF.columns[1:]:
    mean = Countries_RF[col].mean()
    means.append(mean)

RF_Country_means = pd.DataFrame({'Country':names, 'RF␣Mean':means})

# Drop the Eurozone, UK, China, and Australia from the RF_Country_means
    DataFrame
RF_Country_means = RF_Country_means[~RF_Country_means['Country'].isin([
   'Eurozone','UK','China','Australia'])]

# Concatenate the RF_Country_means DataFrame with the new data frames
RF_Country_means = pd.concat([RF_Country_means, Europe_df,
   Asia_Pacific_df], ignore_index=True)

# Rename the countries
RF_Country_means = RF_Country_means.replace({'Country': {'Europe': '
   Europe', 'China': 'Asia_Pacific', 'US':'USA'}})
----------------------------------------------------------------------
# Calculate the frequency of each region
Country_frequence = name_ETFs['Country'].value_counts()
----------------------------------------------------------------------
# Calculate the weights for each country as the product of the mean
   risk-free rate and the frequency
RF_Country_means['Weights'] = RF_Country_means['RF␣Mean'] *
   RF_Country_means['Frequency']

# Calculate the Risk-Free Rate as the sum of the weights divided by the
    sum of the frequencies
Risk_Free_Rate_calculated = RF_Country_means['Weights'].sum() /
   RF_Country_means['Frequency'].sum()

# Round the Risk-Free Rate to 5 decimal places
Risk_Free_Rate_calculated = round(Risk_Free_Rate_calculated, 5)

# Print the calculated Risk-Free Rate
print("The␣calculated␣Risk-Free␣Rate␣is:", Risk_Free_Rate_calculated)

Output: The calculated Risk-Free Rate is: 0.02073
```

5 Portfolio Optimization

The core of this study involves applying Markowitz Mean-Variance Optimization to construct an optimal portfolio. The optimization seeks to maximize the Sharpe ratio, which is a measure of risk-adjusted return.

The Sharpe ratio S is:

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

where:

- R_f is the risk-free rate
- σ_p is the portfolio standard deviation, calculated as $\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}$
- R_p is the expected return of a portfolio, calculated as $R_p = \sum_{i=1}^{n} w_i R_i$

5.1 Implementation

The implementation of the optimization is as follows:

Listing 4: Portfolio Optimization

```
def portfolio_performance(weights, mean_returns, cov_matrix,
   periods_per_year):
    returns = np.sum(mean_returns * weights) * periods_per_year
    std_dev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) *
        np.sqrt(periods_per_year)
    return std_dev, returns

def negative_sharpe_ratio(weights, mean_returns, cov_matrix,
   risk_free_rate, periods_per_year):
    p_var, p_ret = portfolio_performance(weights, mean_returns,
       cov_matrix, periods_per_year)
    return -(p_ret - risk_free_rate) / p_var

# Constraints and bounds
constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bounds = tuple((0, 1) for asset in range(len(mean_returns)))

# Initial guess
init_guess = len(mean_returns) * [1. / len(mean_returns)]

# Optimization
opt_sharpe = sco.minimize(negative_sharpe_ratio, init_guess, args=(
   mean_returns, cov_matrix, risk_free_rate_periodic, periods_per_year)
   , method='SLSQP', bounds=bounds, constraints=constraints)
optimal_weights = opt_sharpe.x
```

5.2 Monte Carlo Simulation

Monte Carlo simulation visualizes the distribution of possible portfolios by generating random weights and calculating their corresponding returns and risks, demonstrating the superior performance of the implemented optimizer. To enhance execution efficiency, we employed Numba, a Just-in-Time compiler for Python, which compiles computationally intensive functions into machine code. This technique offers smoother and more precise computations compared to alternatives like Cython or PyPy. Additionally, the @njit(parallel=True) decorator enables multi-threading, leveraging the full capabilities of the CPU for improved performance.

Listing 5: Monte Carlo Simulation

```
@njit
def generate_random_weights(num_assets):
    weights = np.random.uniform(0, 1, num_assets)
    return weights / np.sum(weights)

@njit(parallel=True)
def monte_carlo_simulation(num_simulations, num_assets, mean_returns,
   cov_matrix, periods_per_year):
    results = np.zeros((num_simulations, 3))
    for i in prange(num_simulations):
        weights = generate_random_weights(num_assets)
        portfolio_stats = portfolio_performance(weights, mean_returns,
            cov_matrix, periods_per_year)
        sharpe_ratio = (portfolio_stats[1] - risk_free_rate_periodic) /
             portfolio_stats[0]
        results[i, 0] = portfolio_stats[1]
        results[i, 1] = portfolio_stats[0]
        results[i, 2] = sharpe_ratio
    return results

# Number of simulations
num_simulations = 10000000
monte_carlo_results = monte_carlo_simulation(num_simulations, len(
    mean_returns), mean_returns.values, cov_matrix.values,
    periods_per_year)
```

5.3 Results and Discussion

The weight allocation calculated by the optimizer is the following:

```
Energy: -0.062276
Gold:  0.170859
Soft:  0.036288
MSCI J:  0.200501
MSCI Asia: -0.461662
Pacific Stocks:  0.137458
MSCI US:  1.000000
US High Dividend: -0.211941
EUR STOXX 600: -0.349219
EUR GOV 1 3:  0.539994

Expected Return of Optimal Portfolio: 0.1405
Volatility of Optimal Portfolio: 0.0958
Sharpe Ratio of Optimal Portfolio: 1.4497
```

Instead, regarding the chart 1: the efficient frontier chart delineates the trade-off between risk and return for optimal portfolios. Monte Carlo simulation points, dispersed around the efficient frontier, illustrate numerous sub-optimal, randomly generated portfolios. The chart also features the Capital Market Line (CML), which connects the risk-free rate to the optimal portfolio. The optimal portfolio, indicated by a red star, represents the highest Sharpe Ratio. As expected, despite the huge number of simulations of the random weights done by the Monte Carlo, the distance between the cloud of possible portfolios and the efficient frontier/optimal portfolio highlights the importance of portfolio optimization. It demonstrates that randomly selecting portfolio weights is unlikely to result in an efficient or optimal portfolio, emphasizing the value of using optimization techniques like Markowitz mean-variance optimization.

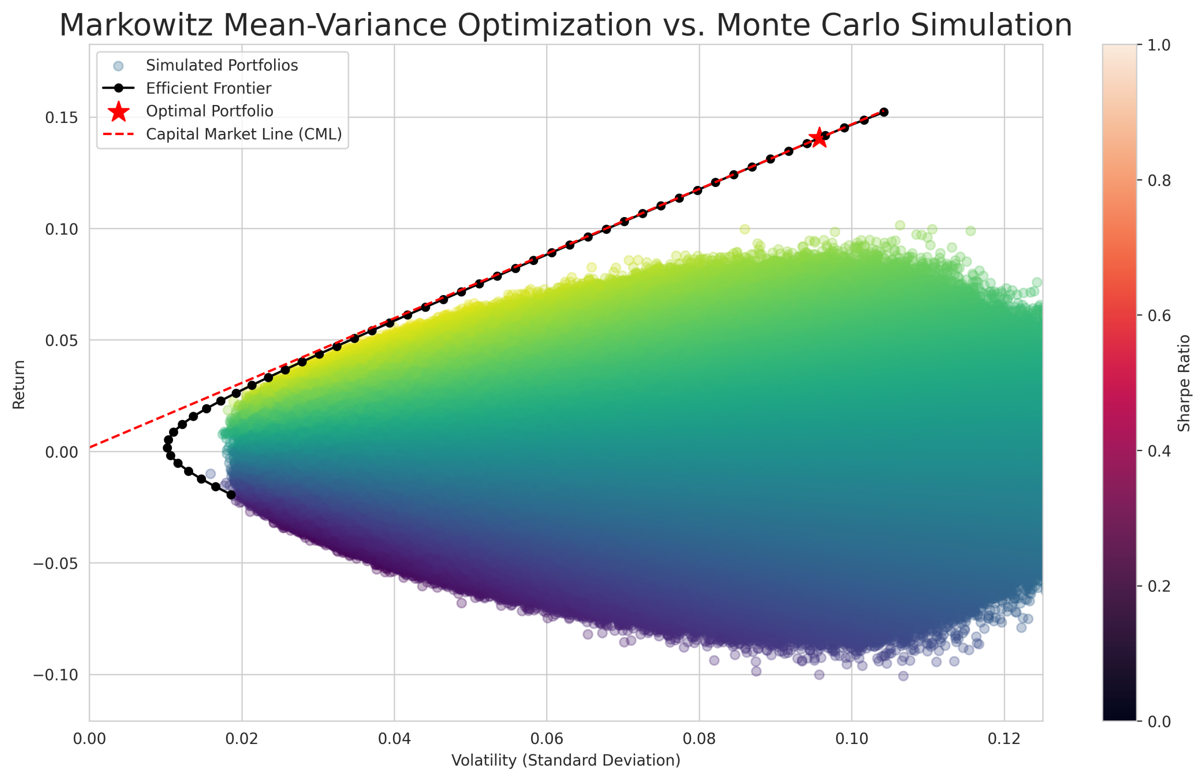

Figure 1: Efficient Frontier and Monte Carlo Simulation Results

6 Rolling Window Optimization and Back-testing

To evaluate the performance of the optimized portfolio over time, we implemented a rolling window optimization and back-testing approach. Rolling window optimization adapts the portfolio to changing market conditions over time. This method involves periodically recalculating the mean returns and covariance matrix within each window and re-optimizing the portfolio weights to maximize the Sharpe Ratio. The rolling window approach ensures that the portfolio remains responsive to market dynamics.

6.1 Implementation

The implementation is carried out as follows:

Listing 6: Rolling Window Optimization and Backtesting

```
# Rolling optimization
window_size = 36  # 36 months
rebalance_period = 1  # Rebalance monthly

def rolling_portfolio_optimization(returns, window_size,
    rebalance_period, optimal_weights):
     num_assets = returns.shape[1]
     portfolio_weights = []
     dates = []
     annualized_returns = []
     annualized_volatility = []
     sharpe_ratio = []
    for i in range(window_size, len(returns), rebalance_period):
        window_returns = returns.iloc[i - window_size:i]
        mean_returns = window_returns.mean()
        cov_matrix = window_returns.cov()

        constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
        bounds = tuple((0, 1) for asset in range(num_assets))
        init_guess = num_assets * [1. / num_assets]

        opt_sharpe = sco.minimize(negative_sharpe_ratio, init_guess,
           args=(mean_returns, cov_matrix, risk_free_rate),
                                  method='SLSQP', bounds=bounds,
                                     constraints=constraints)

        portfolio_weights.append(opt_sharpe.x)
        dates.append(returns.index[i])

        portfolio_returns = (window_returns * opt_sharpe.x).sum(axis=1)
        ann_ret = portfolio_returns.mean() * 12
        ann_vol = portfolio_returns.std() * np.sqrt(12)
        sharpe = (ann_ret - risk_free_rate) / ann_vol

        annualized_returns.append(ann_ret)
        annualized_volatility.append(ann_vol)
        sharpe_ratio.append(sharpe)

    return pd.DataFrame(portfolio_weights, index=dates, columns=returns
       .columns), pd.DataFrame({
        'Annualized␣Returns': annualized_returns,
        'Annualized␣Volatility': annualized_volatility,
        'Sharpe␣Ratio': sharpe_ratio
    }, index=dates)

optimal_weights_over_time, metrics_df = rolling_portfolio_optimization(
   monthly_returns, window_size, rebalance_period, optimal_weights)

# Backtesting
def backtest_rolling_portfolio(returns, optimal_weights):
    optimal_weights = optimal_weights.reindex(returns.index, method='
       ffill')
    portfolio_returns = (returns * optimal_weights).sum(axis=1)
    return portfolio_returns

portfolio_returns = backtest_rolling_portfolio(monthly_returns,
   optimal_weights_over_time)
```

6.2 Results

The analysis reveals Figure 2 shows .. that the rolling portfolio has consistently generated positive returns, with a notable upward trend in cumulative returns. The annualized returns exhibit some fluctuations, indicating periods of higher and lower performance. However, the volatility, represented by the blue line, remains relatively stable, suggesting that the portfolio's risk profile has been well-managed. The Sharpe Ratio, which measures risk-adjusted returns, consistently stays above zero, implying that the portfolio has generated positive returns relative to its risk level.

The rolling optimization approach appears to have effectively adapted to market changes, as evidenced by the portfolio's ability to maintain a consistent level of risk-adjusted performance. This highlights the potential benefits of dynamic portfolio management strategies, which can adapt to evolving market conditions and potentially improve both performance and risk management compared to static portfolios.

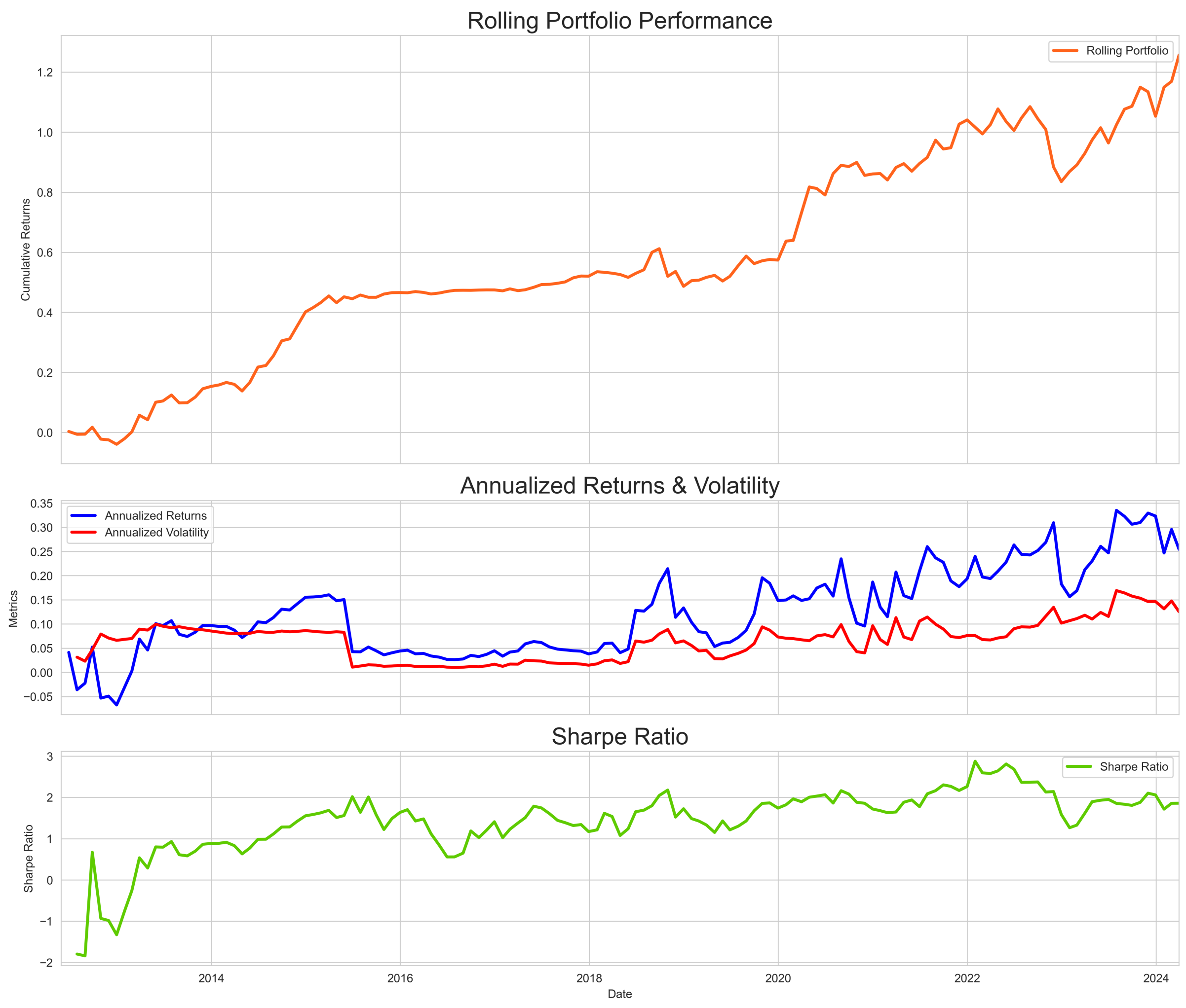

Figure 2: Rolling Portfolio Performance

7 Conclusion

This study demonstrates the application of Modern Portfolio Theory (MPT), Monte Carlo simulations, and rolling window optimization to construct and backtest an optimal portfolio of various financial assets. By integrating extensive data cleaning, correlation analysis, and risk-free rate calculation, the study ensures the robustness and practical applicability of the results. The efficient frontier chart and the rolling window backtesting chart provide valuable visual insights into the trade-offs between risk and return and the effectiveness of the rolling optimization strategy. The efficient frontier chart illustrates the relationship between risk and return for optimal portfolios, while the rolling window backtesting chart highlights the dynamic performance of the portfolio over time.

7.1 Key Observations

- The rolling portfolio optimization strategy shows a steady increase in cumulative returns, suggesting that the approach is effective in capturing market opportunities.
- The annualized returns and volatility plot reveals that the portfolio maintains a relatively stable return profile with manageable risk levels.
- The Sharpe ratio plot indicates that the portfolio achieves a favorable risk-adjusted return, demonstrating the benefits of the optimization process.

7.2 Limitations and Future Work

While the results are promising, there are certain limitations to consider:

- The analysis assumes that historical returns and covariances are indicative of future performance, which may not always hold true.
- The optimization process does not account for transaction costs and other practical constraints that investors may face.
- Future research could examine the effects of varying re-balancing frequencies and window sizes on portfolio performance. Additionally, leveraging Reinforcement Learning (RL) presents a promising avenue. By utilizing historical economic and financial data, an RL agent can be trained to optimize portfolio allocation, aiming to replicate or surpass the Sharpe ratio achieved by traditional methods such as rolling window optimization. The training process would involve the RL agent experimenting with different ETF weight combinations and receiving rewards for achieving or exceeding benchmark Sharpe ratios. Once trained, the RL agent could make forward-looking predictions on ETF prices, thereby identifying optimal portfolio allocations based on anticipated market conditions.

  This RL-based approach has several advantages:

  - It can automate the portfolio optimization process, potentially outperforming manual techniques.
  - The agent's decision-making can adapt to changing market conditions by incorporating economic forecasts
  - The historical validation against a proven optimization method ensures the RL model's effectiveness.

8 References

- Bodie, Zvi, et al. Investments. Thirteenth edition, McGraw Hill LLC,
- Hilpisch, Yves J. Python for Finance : Mastering Data-Driven Finance. O'reilly Media, 2019.