Machine Learning For Finance
Portfolio Optimization for Risk-Adjusted Returns

M2 Quantitative Finance - Université Paris-Saclay

AISSA Rayen, FANKAM Jean-Marie, PATARIN Etienne, SOUMAH Mohamed Lamine

May 2024

Contents

1 Context

In the field of finance, portfolio optimization is a critical task that aims to balance risk and return to achieve the best possible investment outcome. Our project "Portfolio Optimization for Risk-Adjusted Returns", focuses on designing and implementing an AI-guided algorithm to determine the optimal portfolio within a 10-minute time frame for a given risk tolerance.

The primary objective of this project is to construct a portfolio that maximizes returns while adhering to a specified level of volatility. This is particularly useful in the context of automated trading systems where frequent adjustments based on the latest market data are necessary. By forecasting the optimal portfolio at regular intervals, investors can dynamically adjust their asset allocations to respond to real-time market conditions and varying risk preferences.

To achieve this, we developed a comprehensive portfolio optimization framework that integrates both traditional financial theories and advanced AI techniques. The framework consists of several key components:

1. **Data Preprocessing**: We merged and processed initial data from order books and trading books, created new features such as mid-price, spread, log returns, and volatility, and ensured data consistency through statistical summaries.
2. **Portfolio Optimization Strategy**: Using foundational concepts such as log returns and volatility, we applied the Markowitz Mean-Variance Optimization approach to establish a baseline for portfolio construction.
3. **AI integration for 10-Minute Horizon Optimal Portfolio**: We enhanced our strategy by incorporating AI models, specifically linear regression and XGBoost, to forecast the optimal portfolio over a 10-minute horizon. This involved creating lagged features, preprocessing data for modeling, and validating the models.

Throughout the project, we experimented with various methods to refine our approach. We analyzed what worked well and what didn't, understanding the limitations and subtle aspects of our code. This iterative process was crucial in developing a smart and effective solution for real-time portfolio optimization.

In the following sections, we will delve into each component in detail, discussing the methods and techniques employed, the challenges encountered, and the insights gained from our efforts.

2 Data Preprocessing

Data preprocessing is a crucial step in any machine learning project. It involves preparing and cleaning the data to ensure that it is suitable for modeling. For this project, we combined data from the order book and trading book, created new features, and ensured data consistency.

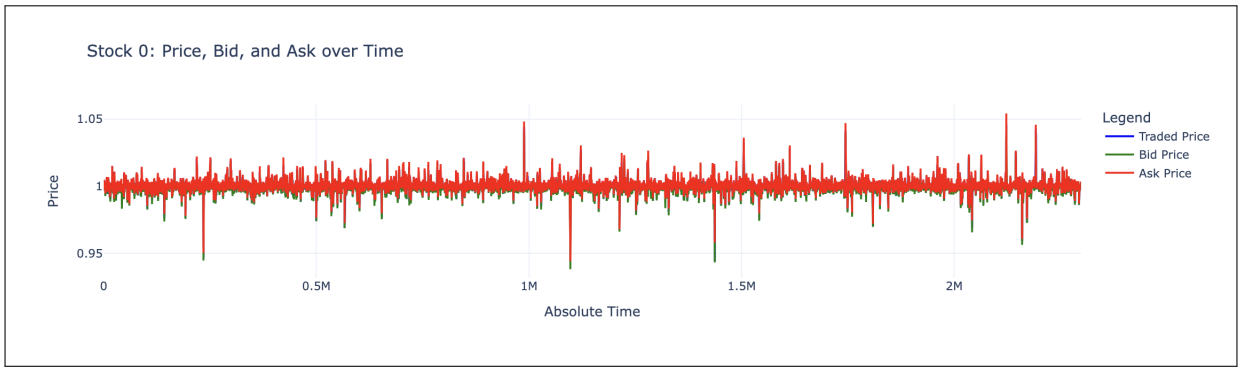
2.1 Merging Initial Data: Order Book and Trading Book

We began by reading the order book and trading book data for both the training and test sets. The order book data provides detailed information about the prices and sizes of buy and sell orders, while the trading book data includes details of executed trades.
We decided to merge both datasets in order to ensured that we had a comprehensive dataset that included both order and trade information for each interval.

$$\mathbf{D} := \mathbf{book} \bigcup_{(\text{stock\_id, time\_id, seconds\_in\_buckets})} \mathbf{trade}$$

where the union denotes the left join operation on the specified keys.
This resulted in a dataset of shape (38,382,740 × 14), combining the initial book (167,253,292 × 11) and trade (38,382,744 × 6) datasets.
The columns includes:

- `stock_id`: Stock identifier (112 stocks).
- `time_id`: Time or session identifier.
- `seconds_in_bucket`: Second within each time bucket (approximately 600 seconds per bucket)
- `bid_price_1`, `ask_price_1`, `bid_price_2`, `ask_price_2`: Bid and ask prices of first and second order.
- `bid_size_1`, `ask_size_1`, `bid_size_2`, `ask_size_2`: Sizes of best and second best bid and ask orders.
- `price`: Trade price.
- `size`: Trade size.
- `order_count`: Number of orders.



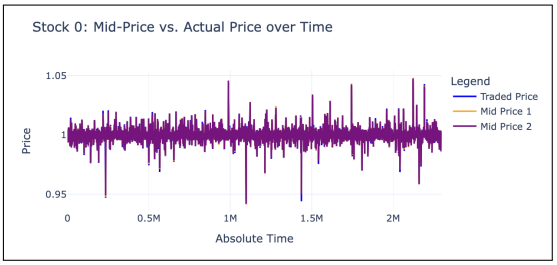
Above we can see the price movements of a specific stock over time, showcasing the traded price, bid price, and ask price. Throughout the timeline, the ask price consistently lies above the bid price, creating a spread. The traded price fluctuates between the bid and ask prices, reflecting the execution of trades within this range. The volatility and density of price changes are visible, with occasional spikes indicating significant price movements or trades.

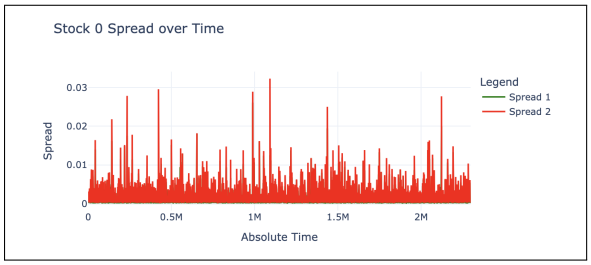
2.2 New Features Creation: Mid-Price, Spread, Log Returns, and Volatility

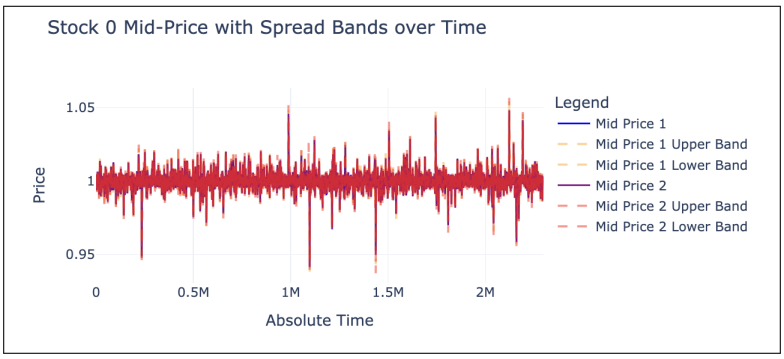
Next, we created several new features that are essential for portfolio optimization:

- `mid_Price_[1/2]`: Average of the bid and ask prices.
- `spread[1/2]`: Difference between the ask price and the bid price.
- `log_returns`: Logarithm of the ratio of consecutive prices.
- `volatility`: Standard deviation of log returns over a rolling window.

The plots below offer insights of a specific stock behavior over time, focusing on mid-price (left graph), actual price, and spread (right graph).
Both mid-prices closely follow the traded price, highlighting that the mid-price is a good approximation of the actual trading price. There are minor deviations, but overall, the mid-prices provide a stable reference point for the actual trading activity.
Regarding the spreads, the right plot shows periods of varying spread sizes, with occasional spikes. These spikes can indicate periods of lower liquidity or higher market volatility, where the difference between the highest price a buyer is willing to pay and the lowest price a seller is willing to accept increases.







2.2.1 Log Returns and Volatility

Log returns and volatility are key metrics for analyzing financial time series data.

Handling Outliers in Price

We captured outliers in the price data using the z-score method and replaced them with the corresponding lower or upper bounds. The z-score method formula is:

$$Z = \frac{X - \mu}{\sigma}$$

where X is the price, μ is the mean price, and σ is the standard deviation. Prices beyond $\mu \pm 3\sigma$ are considered outliers and are replaced by the bounds:

- Lower Bound $= \mu - 3\sigma$
- Upper Bound $= \mu + 3\sigma$

Log Returns

The log return of a price series is given bu the formula:

$$p_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

where P_t is the price at time t and P_{t-1} is the price at the previous time step.

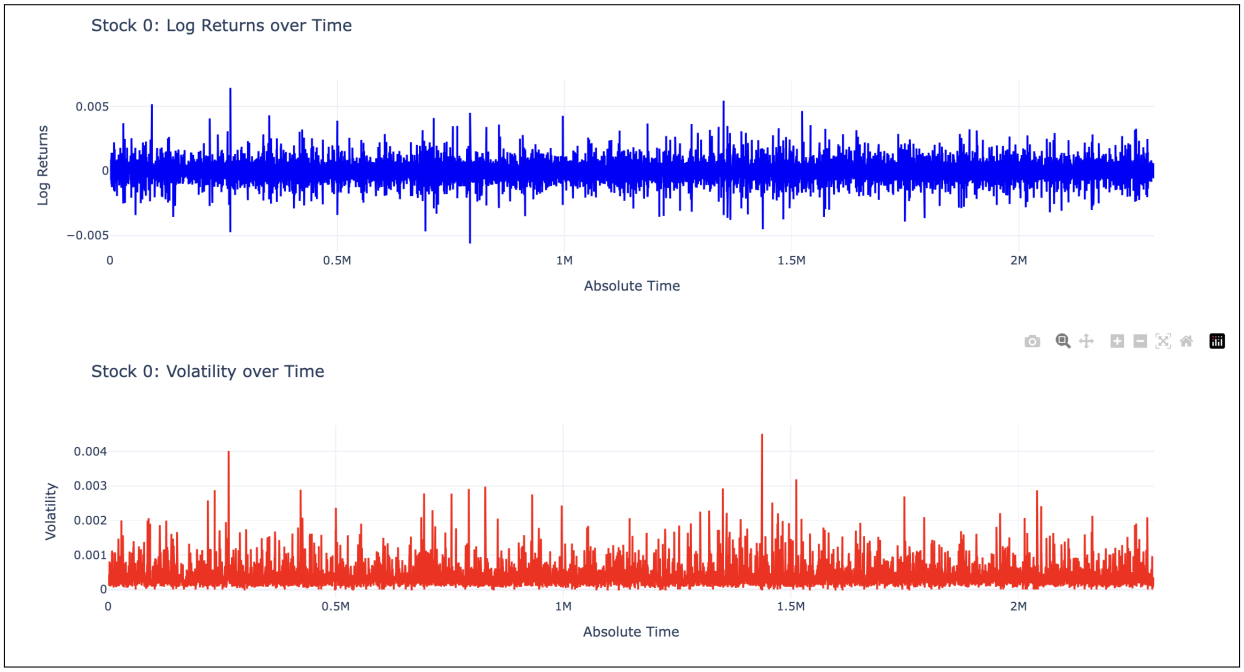
Volatility

Volatility is typically measured as the standard deviation of log returns over a specified window. For a rolling window of size n, the volatility at time t is:

$$\sigma_t = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(p_{t-i} - \overline{p_t})^2}$$

where $\overline{p_t}$ is the mean of log return over the window.

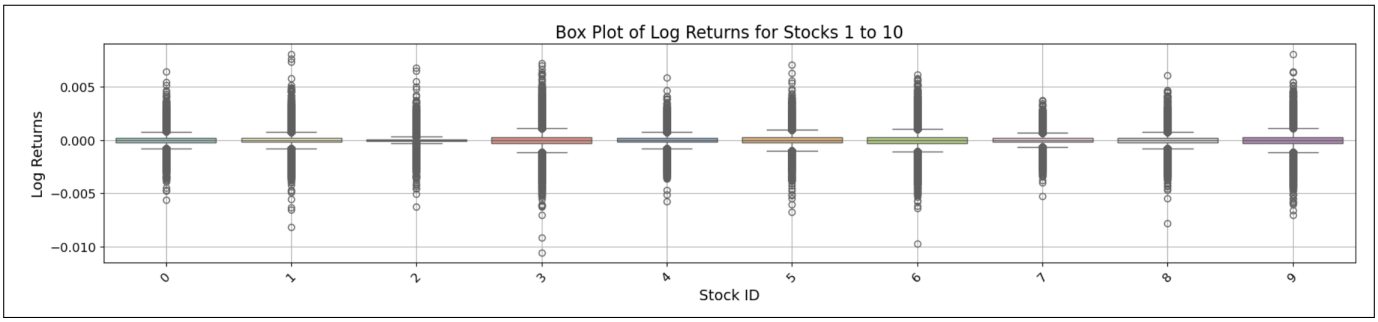
The graphs below show log returns (top plot) and volatility (bottom plot) over time.
The log returns (blue line) fluctuate around zero, indicating the relative changes in price. The spikes represent significant price movements within short time intervals, both upwards and downwards. The plot also shows a high frequency of small fluctuations with occasional larger changes, typical of high-frequency trading data.
The volatility (red line) measures the dispersion of log returns. Periods of high volatility correspond to times of larger and more frequent log return spikes. The volatility plot indicates the risk or uncertainty associated with the stock's price changes over time.



2.3 Statistics Summary and Data Consistency

To ensure data consistency, we checked for null values and duplicates rows, and summarized the statistics of the log returns for each stock.
We also include a field `session_id` and `absolute_time` in order consider each session as a separate period.

Below box plots that we generated for some specific stocks. We can see that the log returns for each stock are centered around zero, indicating that price changes are symmetrically distributed around the mean. There are several outliers for each stock, which are significant deviations from the typical log returns. These could indicate periods of high volatility or unusual market activity. The IQR and the overall range of log returns are relatively small, suggesting that most price changes are minor with occasional larger movements.



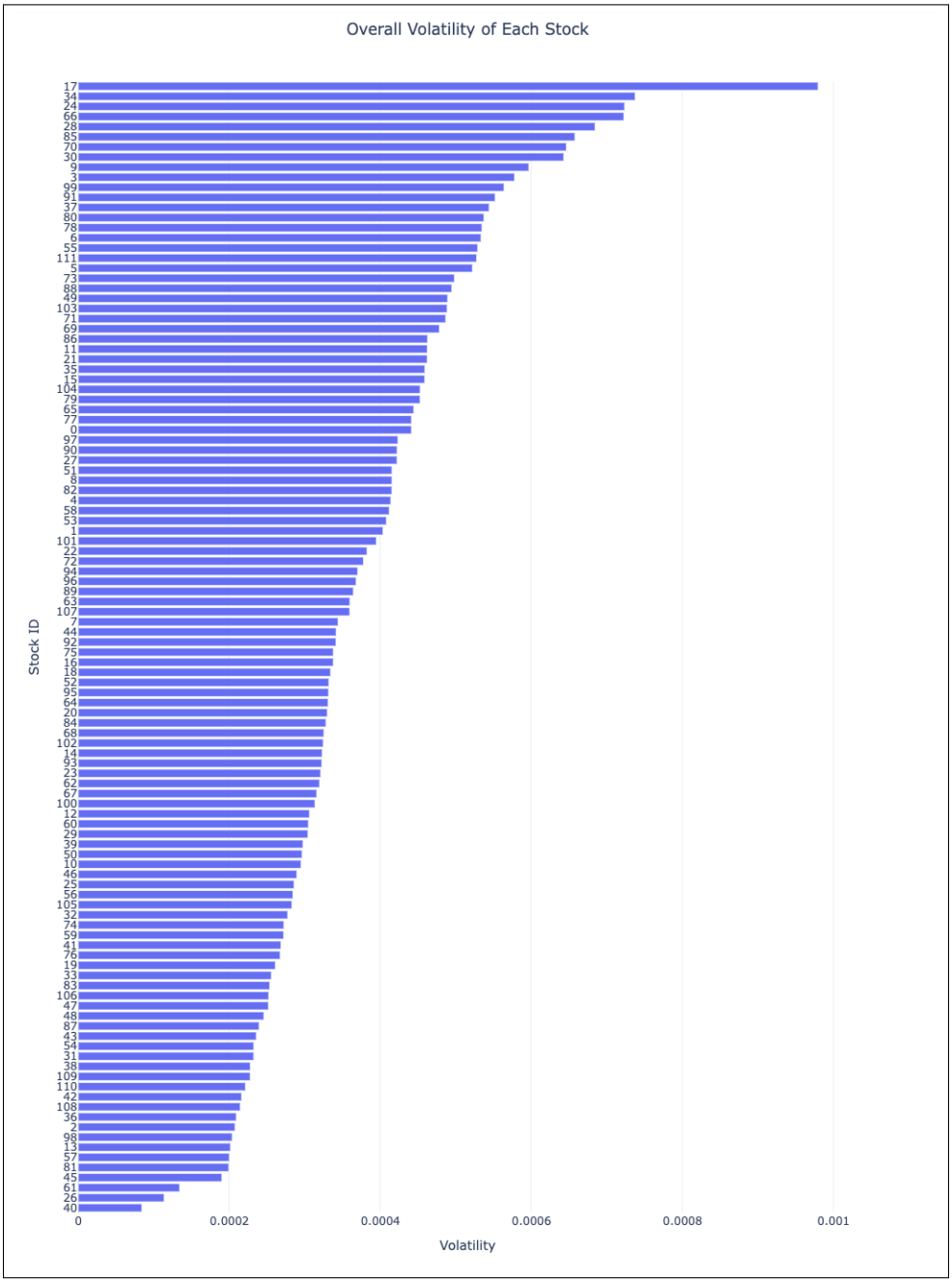
Through these steps, we ensured that our dataset was clean and ready for the subsequent modeling phase. This thorough preprocessing was essential to handle the intricacies of high-frequency trading data and to create reliable features for our portfolio optimization models.

3 Portfolio Optimization Strategy

In this section, we aim to develop a framework that constructs an optimal portfolio, maximizing returns while adhering to a specified level of volatility. This involves leveraging traditional financial theories and advanced machine learning techniques.

3.1 Log Returns and Volatility

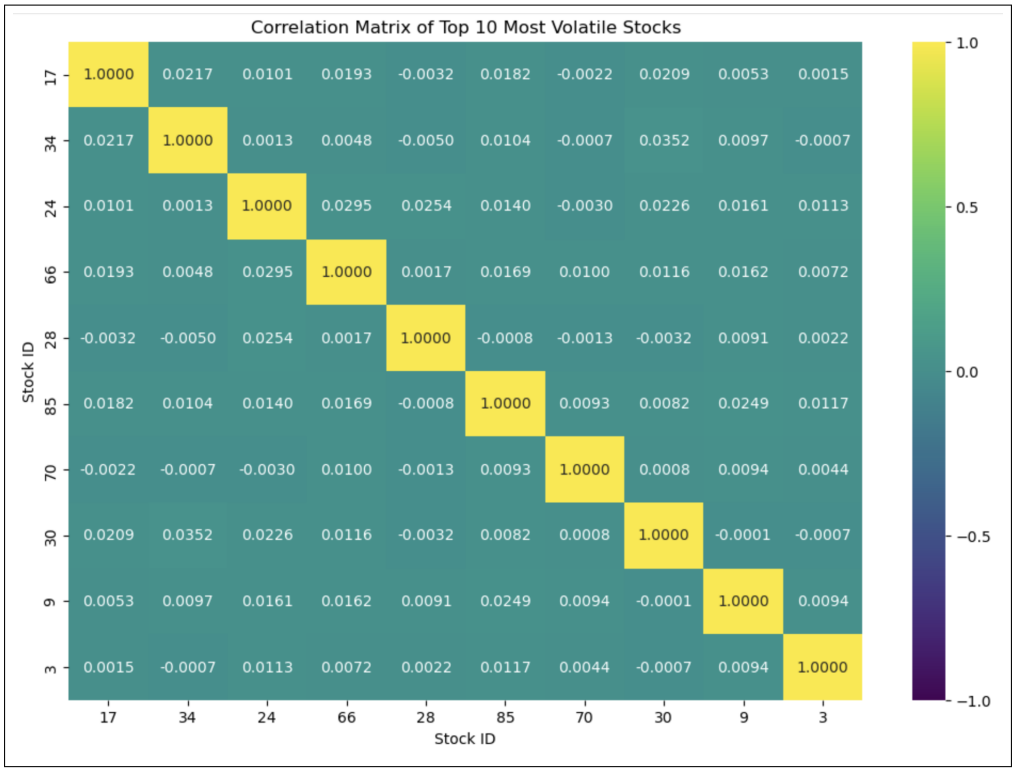
As introduced earlier, log returns and volatility are fundamental metrics for evaluating the performance and risk of financial assets. In our optimization strategy, these metrics help quantify expected returns and risk levels for different assets in the portfolio.



The bar plot above illustrates the overall volatility of each stock, ranked from highest to lowest. The stocks exhibit varying levels of volatility, with the most volatile stock (Stock 17) having significantly higher volatility compared to the least volatile ones. This visualization allows for quick comparison between stocks, identifying which stocks are riskier due to higher volatility.

Correlation of Log Returns

We computed the correlation matrix of log returns to understand the relationship between different stocks.



The plot above represents the correlation matrix of Top 10 most volatile stocks. It helps identify the degree of linear relationship between the log returns of different stocks.

Most pairs of stocks have low or near-zero correlation, indicating that their price movements are largely independent. Low correlation between stocks suggests potential benefits from diversification, as combining uncorrelated assets can reduce overall portfolio risk.

Annualized Mean Returns and Covariance Matrix

To annualize the mean returns and compute the covariance matrix, we used an annual factor based on our data frequency using the following formula:

$$\textbf{Annual Factor} = 252 \times 6.5 \times 6$$

We assumed there are 252 trading days per year, 6.5 trading hours per day, and data sampled every 10 minutes (6 ten-minutes intervals per hour).

We computed the Annualized Means Returns as follows:

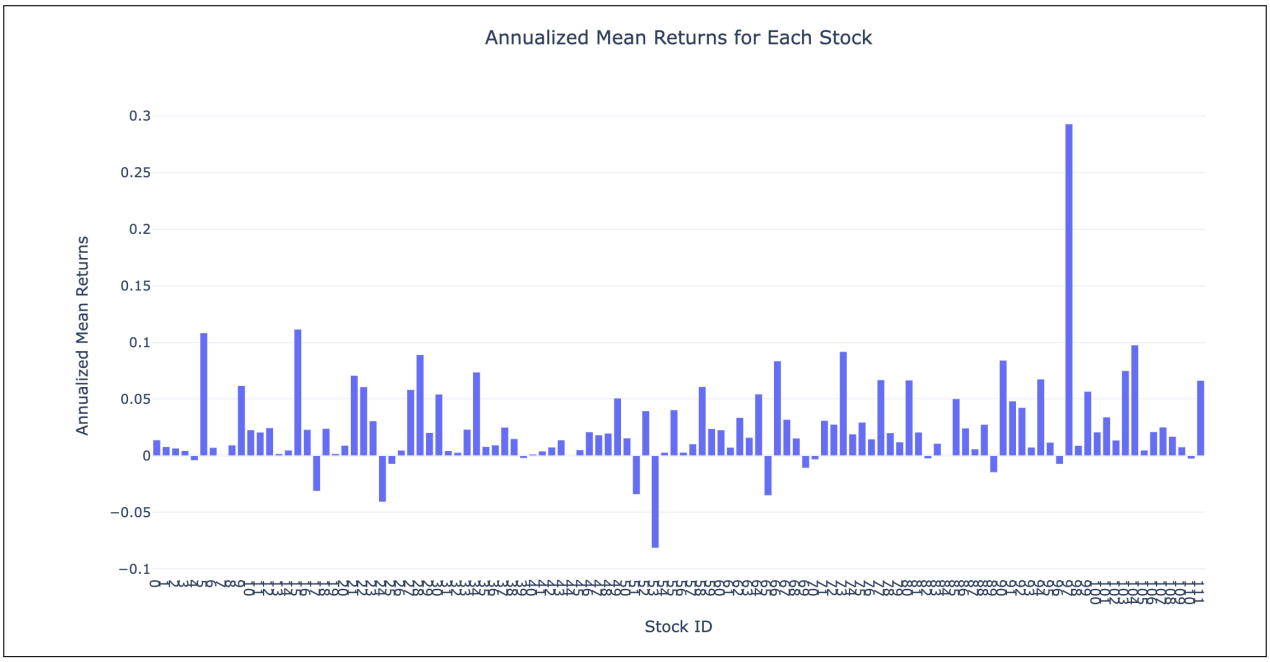
$$\textbf{Annualized Mean Return}_i = \mu_i \times \text{Annual Factor}$$

where μ_i is the mean log return of stock i.

The covariance matrix was scaled by the annual factor to reflect annualized variances and covariances:

$$\textbf{Annualized Covariance}(x_i, x_j) = \text{Cov}(x_i, x_j) \times \text{Annual Factor}$$

The plot below shows the annualized mean returns for each stock. Stocks with higher returns are potential candidates for investment, though risk (volatility) must also be considered. The returns vary widely, with some stocks showing significant positive or negative returns, highlighting the importance of stock selection in portfolio optimization.



3.2 Markowitz Mean-Variance Optimization

The foundation of our portfolio optimization strategy is the Markowitz Mean-Variance Optimization, a classical method in modern portfolio theory. This approach aims to find the optimal portfolio that offers the highest expected return for a given level of risk (volatility) or equivalently, the lowest risk for a given level of expected return.

Expected Return of the Portfolio:

$$\mu_p = \sum_{i=1}^{n} \omega_i \mu_i$$

where μ_p is the expected return of the portfolio, ω_i is the weight of asset i in the portfolio, and μ_i is the expected return of asset i.

Variance of the Portfolio:

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \omega_i \omega_j \sigma_{ij}$$

where σ_p^2 is the variance of the portfolio, ω_i and ω_j are the weights of assets i and j, and σ_{ij} is the covariance between the returns of assets i and j.

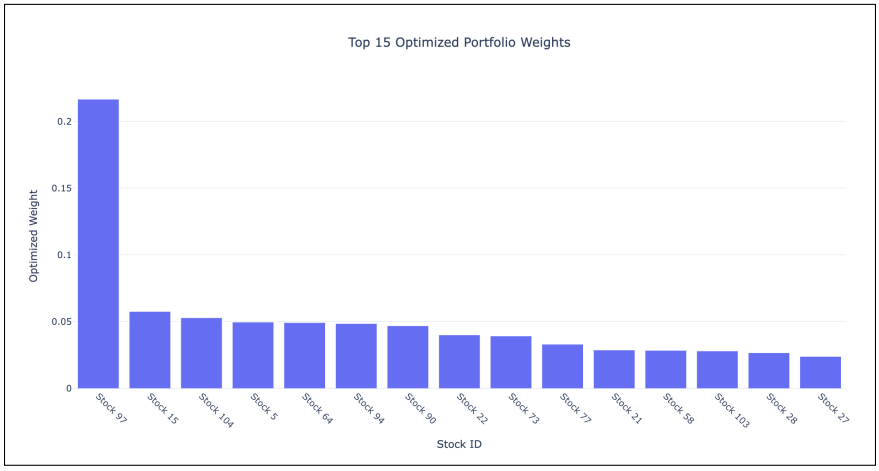
Optimization Objective: The goal is to maximize the Sharpe Ratio, defined as:

$$\textbf{Sharpe Ratio} = \frac{\mu_p - r_f}{\sigma_p}$$

where r_f is the risk-free rate (set to 2% in our simulations).

Alternatively, the optimization can focus on minimizing the portfolio variance for a given level of expected return or maximizing the expected return for a given level of risk.

After performing the optimization, we obtain the optimal weights for each asset in the portfolio as we can see on the plot below, which represents the optimized weights for the top 15 assets in the portfolio:

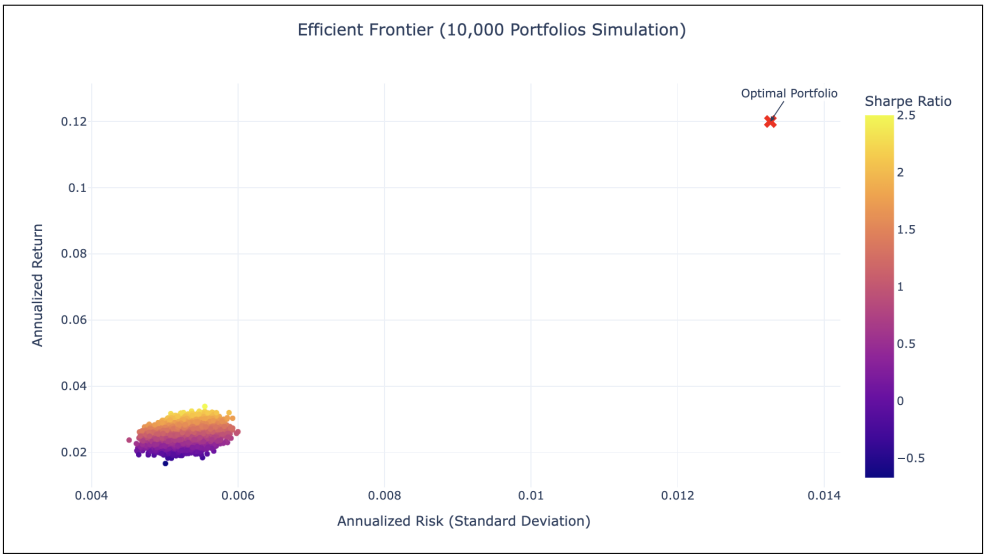


The optimized portfolio heavily weights Stock 97, which indicates it has the most attractive risk-return profile based on our optimization criteria, which is consistent with the previous graph showing that this stock has the highest annual returns.
While Stock 97 has the highest weight, the remaining weights are distributed among other stocks, providing a degree of diversification.
The total weight for the top 15 assets is approximately 0.769, indicating these assets make up a significant portion of the portfolio.

Regarding the portfolio performance metrics, including standard deviation (risk), returns, and Sharpe ratio:

- **Standard Deviation**: The optimal portfolio has a standard deviation of 0.0133, which reflects the portfolio's risk.
- **Returns**: The portfolio is expected to yield an annual return of 0.1200, or 12%.
- **Sharpe Ratio**: The Sharpe ratio of 7.54 indicates a highly efficient portfolio, offering high returns relative to its risk.

The efficient plot below displays 10,000 simulated portfolios, highlighting the optimal portfolio:



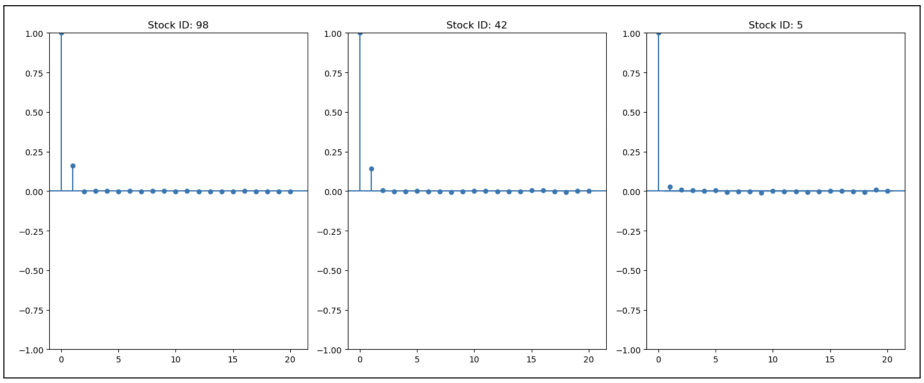
The colored scatter points represent the risk-return combinations of 10,000 simulated portfolios. The color gradient indicates the Sharpe ratio, with warmer colors representing higher Sharpe ratios.
The red cross marks the optimal portfolio, which lies on the far right of the frontier, demonstrating the highest Sharpe ratio.
The efficient frontier illustrates the trade-off between risk and return. Portfolios on the frontier offer the best possible returns for a given level of risk.

4 AI Integration for 10-Minute Horizon Optimal Portfolio

In this section, we used AI techniques to forecast and optimize the portfolio for a 10-minute horizon. The goal is to leverage machine learning models to predict short-term price movements and dynamically adjust the portfolio to maximize returns while maintaining a specified level of risk.

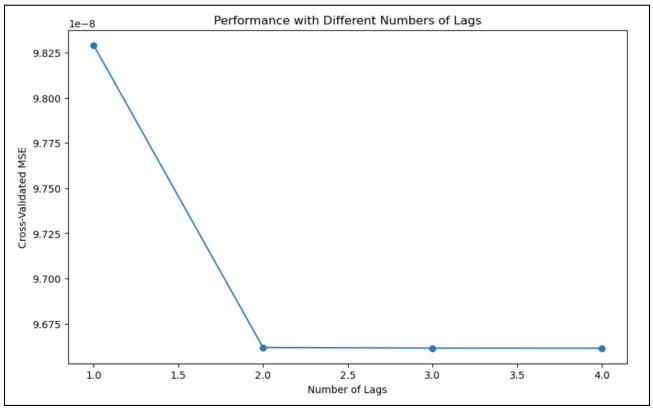
4.1 Lagged Features Creation

To capture the temporal dependencies in the data, we created lagged features for each stock. Lagged features are past values of the variables used as predictors in our models. These features help in predicting future price movements based on historical data.
To determine the optimal number of lags, we computed the autocorrelation for each stock and aggregated the results. We implemented a function to compute significant lags for each stock based on a threshold for autocorrelation.



The graph above show the autocorrelation for some specific stocks of the portfolio. We can see that there are negligible autocorrelation at higher lags than the lag 1.

We computed the average and median number of significant lags across all stocks in order to aggregate all stocks results. The average optimal lag number is 1, but we didn't stop there and evaluated the number of lags from a range of 1 to 5 using cross-validation and plotted the Mean Squared Error (MSE) against the number of lags. The results are shown below:



The plot shows the cross-validated MSE for different numbers of lags. We observed that the MSE decreases significantly when increasing the number of lags up to 2, after which it stabilizes. Based on the MSE plot, we determined that using 2 lags provides a balance between model complexity and prediction accuracy.

4.2 Preprocessing for Modeling

With the large volume of data, we faced significant challenges related to **time and memory complexity**. To address these, we avoided for-loops where possible, preferring vectorized operations, and accelerated processes using `Numba`. Despite these improvements, the data size remained a challenge, necessitating the creation of a sampling function to manage the workload effectively.

Sampling Function

To efficiently handle the large dataset, we implemented a stratified sampling function. This function ensures that each stock and session's identifier is proportionally represented without shuffling the data, maintaining the temporal order.

Train Test Split Function for Time Series

We also implemented a custom train-test split function to preserve the temporal order and stock's identifier dependencies. This approach ensures that the training and test sets are sequential and consistent with the structure of the data.

Scaling the Data

Scaling is a crucial preprocessing step in machine learning, especially for models sensitive to the scale of input features like linear regression and gradient boosting algorithms. We standardized the features to have a mean of zero and a standard deviation of one. This ensures that all features contribute equally to the model and improves the convergence speed of gradient-based optimization algorithms.

$$z = x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

where X is the original feature value, μ is the mean of the feature, and σ is the standard deviation of the feature.

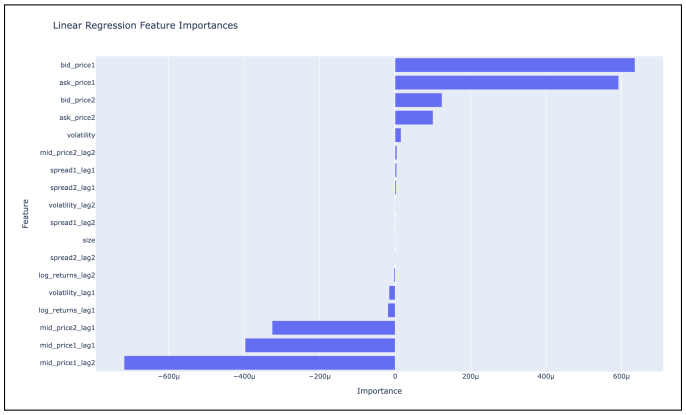
4.3 AI Models

To predict the log-returns of stocks, we employed two different models: Linear Regression and XGBoost. Each model was chosen for its unique strengths and capabilities in capturing the underlying patterns in financial time series data.

4.3.1 Linear Regression

Linear Regression was chosen as a baseline model due to its simplicity and interpretability. It assumes a linear relationship between the input features and the target variable (log-returns), making it easy to understand and implement. Although it may not capture complex patterns in the data, it provides a good benchmark for evaluating the performance of more sophisticated models.

The graph below shows the feature importances identified by the Linear Regression model:

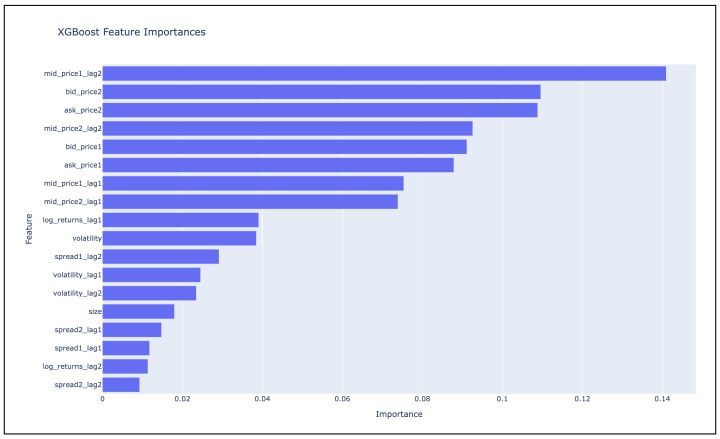


The most significant features includes the best orders for bid and ask price (`bid_price1`, `ask_price1`), for positive weights, and the mid prices for best orders for negative weight.
The Mean Squared Error (MSE) for the Linear Regression model on the test set is 7.476×10^{-8}.

4.3.2 XGBoost

XGBoost is a powerful gradient boosting algorithm, was employed to capture more complex patterns in the data. It is known for its efficiency and performance in regression tasks. XGBoost can handle non-linear relationships and interactions between features, making it suitable for financial time series data where such patterns are common.

The graph below shows the feature importances identified by the XGBoost model:



The top features for the XGBoost model include the mid price (`mid_price1_lag2`), the bid and ask price for second best orders (`bid_price2`, `ask_price2`). The diverse set of important features, including lags and spreads, shows that XGBoost effectively captures complex patterns and relationships in the data, enhancing its predictive power. The MSE for the XGBoost model on the test set is 4.282×10^{-8}.

4.4 Validation and Testing

To validate the models, we used metrics such as Mean Squared Error (MSE) to evaluate the prediction accuracy. The models' performance on the test set was crucial to ensure that they generalize well to unseen data.

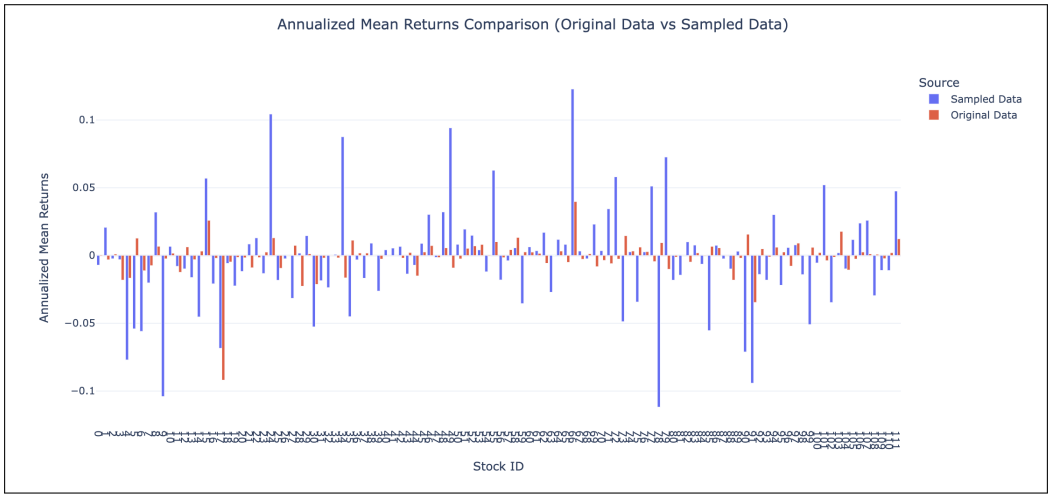
The following graph compares the performance of the Linear Regression and XGBoost models using the test MSE:

- Linear regression: Cross-Validation MSE is 7.665×10^{-8}, and test MSE is 7.476×10^{-8}.
- XGBoost: Cross-Validation MSE is 4.531×10^{-8}, and test MSE is 4.282×10^{-8}.

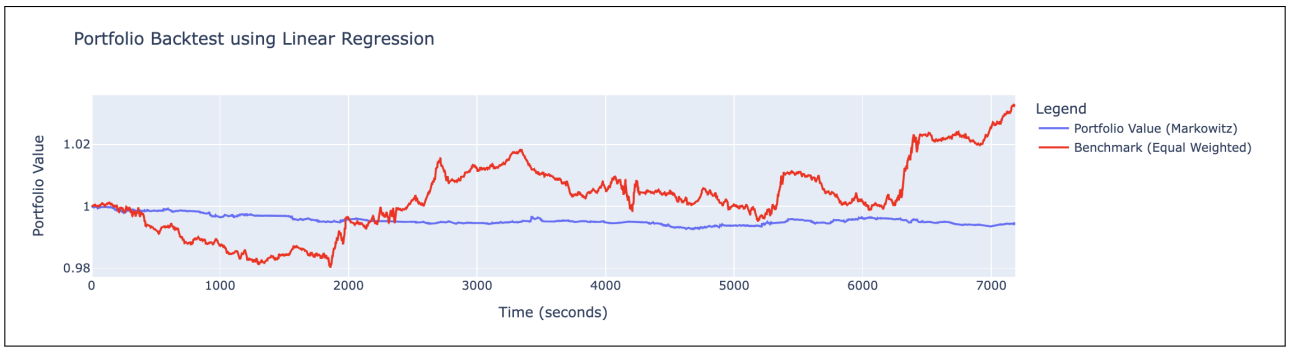
XGBoost outperforms Linear Regression significantly, with lower MSE values in both cross-validation and test sets. This demonstrates XGBoost's superior ability to capture complex relationships and provide more accurate predictions.

Backtesting the Portfolio

We implemented a function to backtest our portfolio based on the predicted returns of both Linear Regression and XGBoost model. We computed on the sample data the mean returns as well as the covariance matrix and compared them on the entire dataset as we can see below:

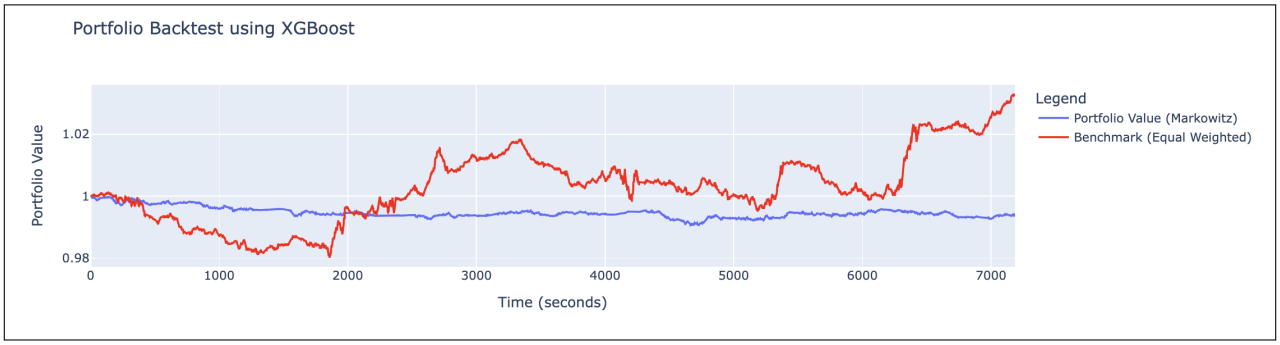


We computed a benchmark based on the initial dataset that is equal-weighted returns. This portfolio gives the same importance to each stock in the portfolio.



The graph above shows the portfolio value of the linear regression using the Markowitz strategy against the Equal Weighted portfolio value at each time. We can see that the the benchmark outperforms almost all the time the linear regression portfolio. However, while the benchmark seems to be volatile over time, the portfolio is more constant over time which makes it less risky even if portfolio values turn progressively more negative over time.

Below the portfolio value trained using the XGBoost model:



The curves of the linear regression and XGBoost models are quite similar because both models effectively

capture the underlying patterns in the high-frequency financial data, leading to comparable portfolio values. Despite XGBoost's superior predictive power indicated by its lower MSE, the incremental benefits over linear regression in the context of this dataset are marginal, resulting in similar performance for both models in the portfolio backtest.

4.5 Forecasting Optimal Portfolio for different risk levels

Using the predictions from the AI models, we dynamically adjusted the portfolio weights to optimize for the 10-minute horizon. This involved rebalancing the portfolio based on the forecasted log returns, while adhering to the specified risk tolerance.

To proceed, we have implemented functions to rebalance the portfolio by maximizing the Sharpe Ratio as a function of tolerance to a certain level of risk, to predict the optimal portfolio and evaluate the effectiveness and stability of the strategy over time.

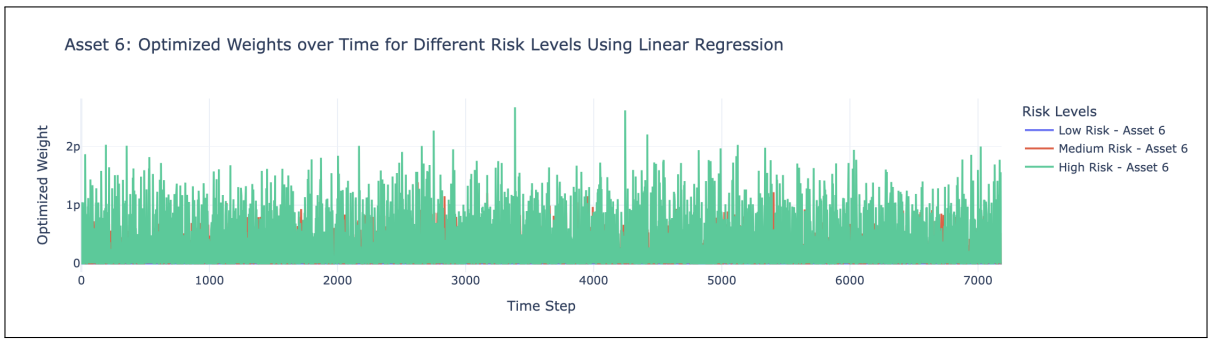
We have defined 3 key levels/threshold of risk:

- **Low Risk Level**: 1% volatility.
- **Medium Risk Level**: 15% volatility.
- **High Risk Level**: 30% volatility.

The table below shows the prediction results for the different risk levels using the linear regression model:

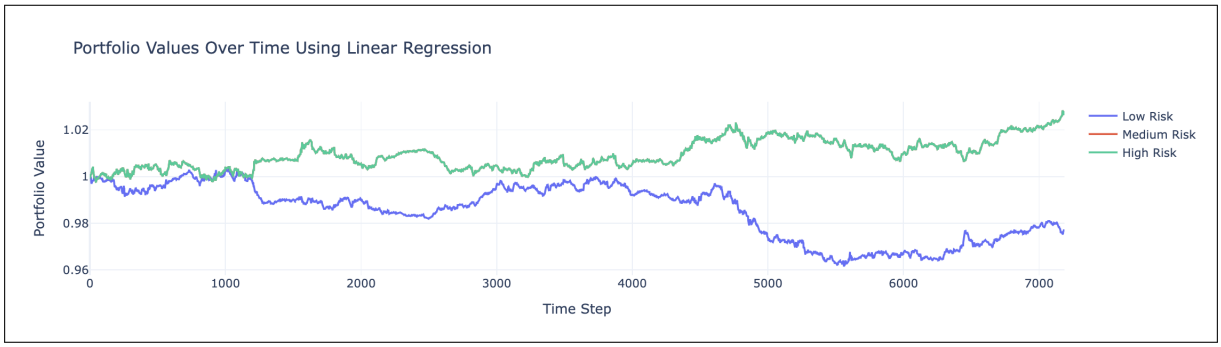
	mean_change	std_change	proportion_stable
Low Risk	1.740472e-08	5.427892e-07	0.999443
Medium Risk	1.272774e-10	9.829166e-11	1.000000
High Risk	1.335353e-10	9.463410e-11	1.000000

We can see that the portfolio seems to be stable over time regardless of the level of risk. Regarding the mean and the standard deviation of the change, changes are more frequent in the low-risk portfolio, which may reflect the need to reallocate the portfolio more often in order to maintain low risk.



The graph above shows the optimized weights over time for different risk levels using the linear regression model. As seen, higher optimized weights are consistently associated with higher risk levels. This reflects the model's allocation strategy where assets with higher expected returns (and hence higher risk) receive greater weights in the portfolio for higher risk tolerance levels. The graph indicates a more aggressive allocation for high-risk levels, while low and medium risk levels show more conservative and stable weight distributions over time.

The graph below represents the portfolio values over time using the linear regression model:

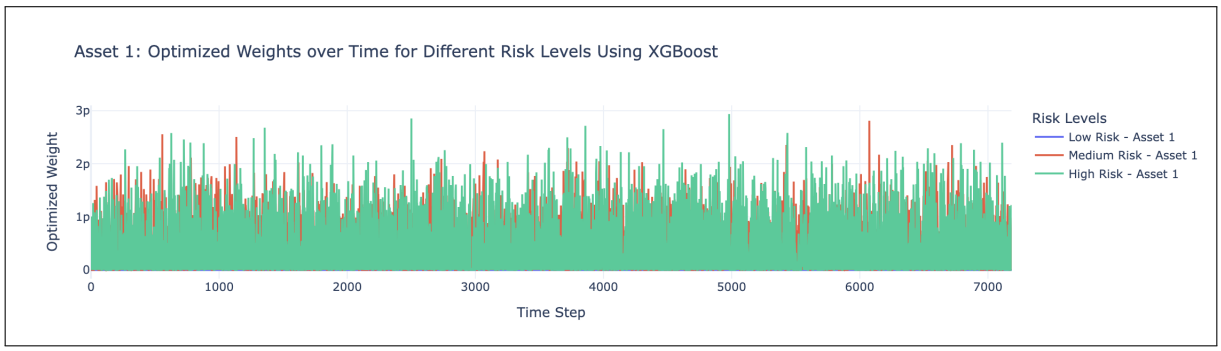


The graph above shows the portfolio values over time for different risk levels using the linear regression model. Higher risk portfolios generally yield higher returns, as indicated by the green line, while lower risk portfolios show less volatility but also lower returns, as seen with the blue line. This demonstrates the trade-off between risk and return in portfolio optimization.

The same conclusions can be drawn for the XGBoost model, whose simulation plots are shown below:

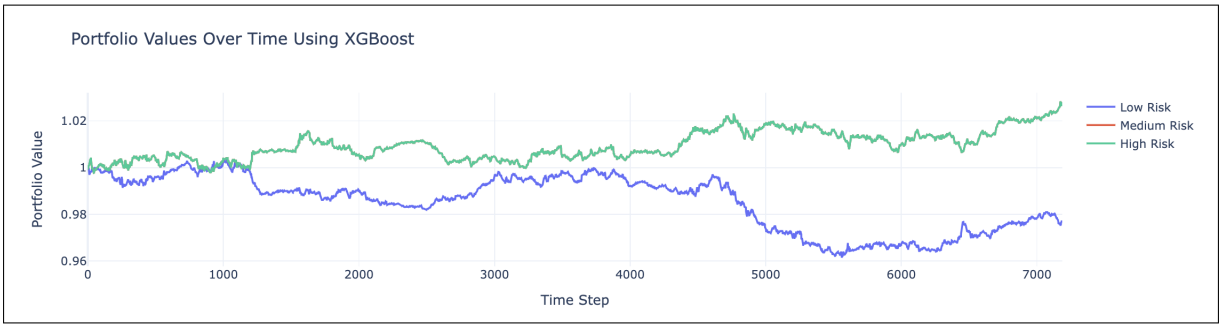
	mean_change	std_change	proportion_stable
Low Risk	1.034814e-08	4.541394e-07	0.999722
Medium Risk	1.278199e-10	9.731926e-11	1.000000
High Risk	1.309393e-10	9.365848e-11	1.000000

The table above presents the mean change, standard deviation of change, and proportion of stability for the optimized portfolio weights using XGBoost across different risk levels. The high risk portfolio exhibits a slightly higher mean change, indicating more frequent adjustments in weights, while the low risk portfolio shows slightly less frequent adjustments. However, all portfolios demonstrate a high proportion of stability, with values very close to 1. This indicates that XGBoost maintained stable portfolio weights over time despite the varying risk levels.



The graph above shows the optimized weights for Asset 1 over time for different risk levels using the XGBoost model. As expected, higher risk levels are associated with more significant weight allocations, demonstrating the model's responsiveness to risk tolerance.

The graph below displays the portfolio values over time for different risk levels using XGBoost. Similar to the linear regression model, higher risk portfolios generally yield higher returns, evidenced by the green line, while lower risk portfolios exhibit more stability but lower returns, as indicated by the blue line.



5 Conclusion

Finally, this project has enabled us to appreciate the importance and application of machine learning in finance, particularly in asset management through portfolio optimization. We have seen that while Machine Learning alone is not sufficient to dynamically allocate a portfolio, it serves as a powerful tool when used in modern strategies such as Markowitz.

Predicting the optimal portfolio at short and regular intervals enables the portfolio to be realigned with increasingly up-to-date market data. We implemented two Machine Learning models: a simple yet effective linear regression model serving as a benchmark, and XGBoost, a gradient boosting model particularly effective in handling our high-frequency data. Although XGBoost achieved a lower Mean Squared Error (MSE), both models produced similar results due to the high efficiency of linear regression in capturing linear relationships in the data and the marginal gains from more complex models when dealing with high-frequency financial data.

One of the main challenges in this project was the time and memory complexity due to the large dataset size. Writing highly optimized code was crucial to avoid redundant calculations and efficiently manage memory and processing time.

Future perspectives for this project could include the implementation of other portfolio optimization strategies for comparative analysis. Given the limitations of our computational resources, further hyperparameter tuning of models, particularly XGBoost, could enhance performance. Additionally, incorporating textual market data using Natural Language Processing (NLP) and sentiment analysis could provide more comprehensive market movement predictions.