MONASH University

Group 2: Stock Portfolio Optimisation

Name	Contribution	Responsibilities
Jiazhe Han	10%	Data cleaning, presentation, report
Mehak Dhingra	20%	Executive Summary, Absolute static weights (random and optimisation), Presentation, Report
Ruchir Mandar Kajrolkar	22.5%	Data cleaning, Breakout strategy using Absolute Static weights: Equal weights,Strategy development, implementation, feature optimization, research, gridsearch, coding, Presentation, Report
Salihah Nisha	20%	Relative static weights, Presentation, Conclusive, Report
Simin Liang	27.5%	Data cleaning, Absolute Static Weights(random and optimisation), relative static weights(random and optimisation), dynamic weights (Machine learning- Supportive Vector Regression & standard optimisation), Presentation, Report

Table of Contents

Executive Summary

Project Brief

Stocks represent ownership in companies that give shareholders voting rights including any claim on the company's earnings and assets. They are traded on stock exchanges where investors buy and sell them to capitalise on their return potential, primarily through two avenues: capital appreciation and dividends. However, it is important to note that stocks are also inclined to face large risks that come present in the economy and other factors. Incorporating stocks into investment portfolios provides growth opportunities and diversification benefits. Therefore, it is important to understand the nature of stocks as this will allow investors to build an effective portfolio.

By analysing market trends and economic indicators, investors can make informed decisions to maximise returns while decreasing risks. Investors are more likely to implement various optimisation strategies that will manage risk and increase the overall return value. Throughout this project, different portfolio optimisation strategies are explored that will allow investors to use a portfolio that will generate a high return value which will ultimately lead to a higher sharpe ratio. These strategies include absolute static weights, relative static weights, dynamic weights using SVR and equal weights using the breakout strategy.

Final Conclusions

The final analysis displayed Support Vector Regression (SVR) as my most effective strategy that produced a high return value compared to other methods like breakout strategy. Although the breakout strategy demonstrated more proficiency in short-term predictions, SVR is still chosen as the preferred choice for long-term forecasting. This is because it remained consistent throughout the entire timeframe and actively displayed an actual mean Sharpe ratio of 2.92.

The consistent outperformance of SVR against traditional methods, with an actual mean Sharpe ratio of 2.92 compared to the often low ratios of conventional approaches, commonly around or below 2, can be seen. The efficiency of SVR in providing more accurate and reliable forecasts for portfolio optimisation is illustrated by this significant performance gap. In addition, SVR demonstrates resilience across different market conditions which makes it more suitable for volatile (risk) environments. Traditional methods may struggle to adapt to sudden market shifts and may only be consistent with a linear model. SVR overcomes this issue and adjusts accordingly to display accurate predictions

Introduction

Main Objective

The primary objective of this study is to determine the optimal stock portfolio configuration using the Sharpe ratio, a widely recognized metric for evaluating risk-adjusted returns. This objective will be explored through three distinct scenarios: static weights (absolute), static weights (relative), and dynamic weights. The goal is to analyse and compare these scenarios to identify which strategy provides the best risk-adjusted returns.

Background

The dataset used in this study comprises historical stock prices from the S&P 500,The Standard & Poor's 500 Index which is a hallmark of the financial markets and a cornerstone of modern finance. Introduced in 1957, the S&P 500 represents the 500 largest publicly traded companies in the United States, encompassing a broad spectrum of industries and sectors. The dataset includes daily closing prices over an extensive period, sourced from reliable financial databases.

The Sharpe ratio is employed to measure the risk-adjusted return of each investment. It is calculated by dividing the difference between the investment's return and the risk-free rate by the standard deviation of the investment's returns. This ratio provides a standardised measure of the excess return per unit of risk, allowing for comparison across different investments and strategies. A higher Sharpe ratio indicates better risk-adjusted performance.

Key challenges in this study include handling missing data, accurately calculating returns, and optimising portfolio weights under various constraints. Missing data can arise from different sources, such as stock market holidays, weekends, and data gaps. To address this, the study employs methods like forward filling and backward filling to interpolate missing values.

Additionally, any remaining gaps are removed to ensure a complete dataset for analysis. Optimising portfolio weights under various constraints is another critical challenge. The study explores different optimization techniques, such as random generation of portfolios, standard optimization methods, and advanced machine learning models like Support Vector Regression (SVR). Each method involves setting specific constraints to ensure a fully invested portfolio with weights summing to one. By comparing the performance of different optimization techniques, the study aims to identify the most effective approach for maximising the Sharpe ratio.

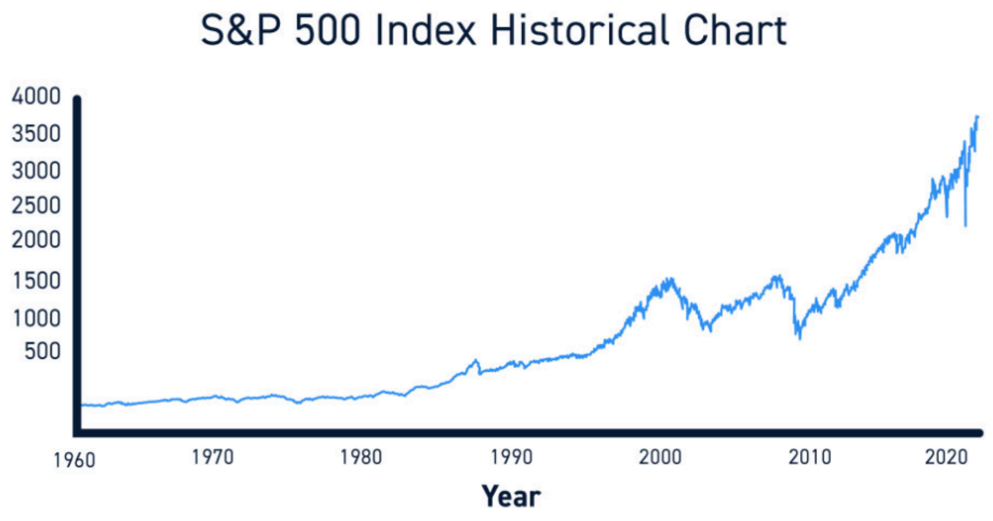


Figure 1.1 Historical chart of S&P 500 (j87as6. 2021),

Hypothesis

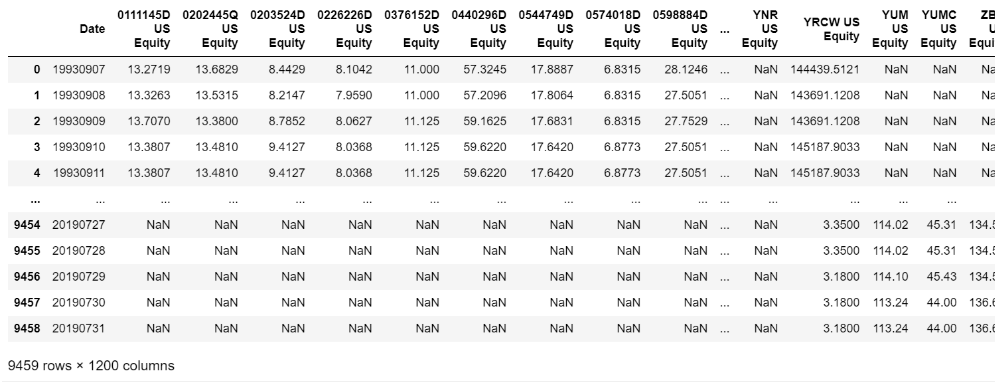
The hypothesis of this study is that employing Support Vector Regression (SVR) in the dynamic weights scenario will yield a higher Sharpe ratio and better overall portfolio performance compared to static weight strategies. By adjusting investments based on recent stock performance using methods like simple moving averages and machine learning models, we expect to achieve superior risk-adjusted returns.

To test this hypothesis, the study will implement SVR models for predicting stock returns and volatility. The predictions from these models will be used to adjust the weights of the stocks in the portfolio dynamically. The performance of the SVR-based dynamic portfolio will be compared to that of the static weight strategies in terms of the Sharpe ratio.

Data Quality and Preprocessing

Preprocessing for Absolute Static, Relative Static and Dynamic Weights

Figure 2.1 shows that after we display the csv file directly, we find that the data has 9459 rows by 1200 columns, so we decide to process the data.

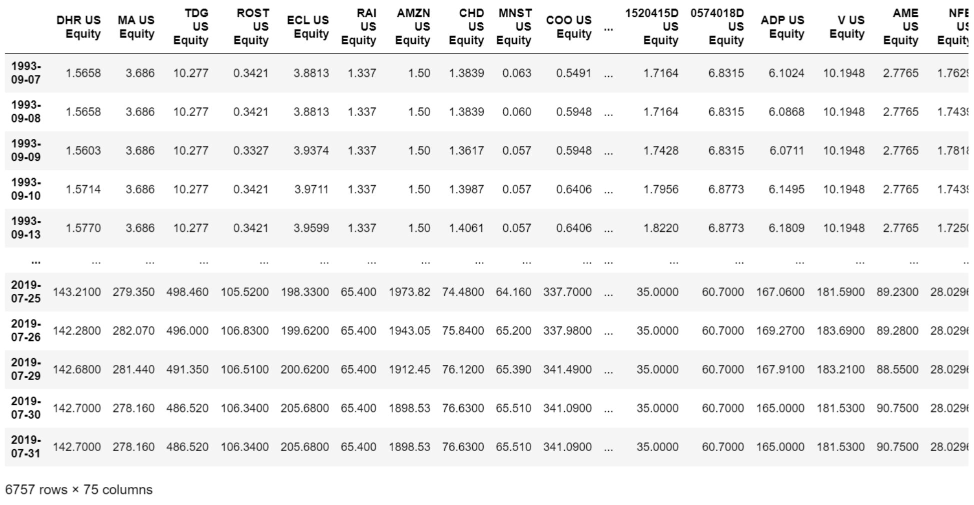


	Date	0111145D US Equity	0202445Q US Equity	0203524D US Equity	0226226D US Equity	0376152D US Equity	0440296D US Equity	0544749D US Equity	0574018D US Equity	0598884D US Equity	...	YNR US Equity	YRCW US Equity	YUM US Equity	YUMC US Equity	ZB U Equi
0	19930907	13.2719	13.6829	8.4429	8.1042	11.000	57.3245	17.8887	6.8315	28.1246	...	NaN	144439.5121	NaN	NaN	Na
1	19930908	13.3263	13.5315	8.2147	7.9590	11.000	57.2096	17.8064	6.8315	27.5051	...	NaN	143691.1208	NaN	NaN	Na
2	19930909	13.7070	13.3800	8.7852	8.0627	11.125	59.1625	17.6831	6.8315	27.7529	...	NaN	143691.1208	NaN	NaN	Na
3	19930910	13.3807	13.4810	9.4127	8.0368	11.125	59.6220	17.6420	6.8773	27.5051	...	NaN	145187.9033	NaN	NaN	Na
4	19930911	13.3807	13.4810	9.4127	8.0368	11.125	59.6220	17.6420	6.8773	27.5051	...	NaN	145187.9033	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9454	20190727	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	3.3500	114.02	45.31	134.
9455	20190728	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	3.3500	114.02	45.31	134.
9456	20190729	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	3.1800	114.10	45.43	134.
9457	20190730	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	3.1800	113.24	44.00	136.
9458	20190731	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	3.1800	113.24	44.00	136.

9459 rows × 1200 columns

Figure 2.1 Original Dataframe for stocks

Figure 2.2 shows the resulting image after we processed the data. Firstly the data was loaded and the 'Date' column was converted to datetime format and then set to index. This will allow us to process the time series data appropriately. Missing values are managed using forward fill and backward fill techniques, and if there is still a NaN value, we delete the row to ensure that no gaps remain. The top 75 stocks were selected based on the highest Sharpe ratio and used as the base data for the portfolio optimisation analysis. The data has 6757 rows by 75 columns



	DHR US Equity	MA US Equity	TDG US Equity	ROST US Equity	ECL US Equity	RAI US Equity	AMZN US Equity	CHD US Equity	MNST US Equity	COO US Equity	...	1520415D US Equity	0574018D US Equity	ADP US Equity	V US Equity	AME US Equity	NFE US Equity
1993-09-07	1.5658	3.686	10.277	0.3421	3.8813	1.337	1.50	1.3839	0.063	0.5491	...	1.7164	6.8315	6.1024	10.1948	2.7765	1.762
1993-09-08	1.5658	3.686	10.277	0.3421	3.8813	1.337	1.50	1.3839	0.060	0.5948	...	1.7164	6.8315	6.0868	10.1948	2.7765	1.743
1993-09-09	1.5603	3.686	10.277	0.3327	3.9374	1.337	1.50	1.3617	0.057	0.5948	...	1.7428	6.8315	6.0711	10.1948	2.7765	1.781
1993-09-10	1.5714	3.686	10.277	0.3421	3.9711	1.337	1.50	1.3987	0.057	0.6406	...	1.7956	6.8773	6.1495	10.1948	2.7765	1.743
1993-09-13	1.5770	3.686	10.277	0.3421	3.9599	1.337	1.50	1.4061	0.057	0.6406	...	1.8220	6.8773	6.1809	10.1948	2.7765	1.725
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
2019-07-25	143.2100	279.350	498.460	105.5200	198.3300	65.400	1973.82	74.4800	64.160	337.7000	...	35.0000	60.7000	167.0600	181.5900	89.2300	28.029
2019-07-26	142.2800	282.070	496.000	106.8300	199.6200	65.400	1943.05	75.8400	65.200	337.9800	...	35.0000	60.7000	169.2700	183.6900	89.2800	28.029
2019-07-29	142.6800	281.440	491.350	106.5100	200.6200	65.400	1912.45	76.1200	65.390	341.4900	...	35.0000	60.7000	167.9100	183.2100	88.5500	28.029
2019-07-30	142.7000	278.160	486.520	106.3400	205.6800	65.400	1898.53	76.6300	65.510	341.0900	...	35.0000	60.7000	165.0000	181.5300	90.7500	28.029
2019-07-31	142.7000	278.160	486.520	106.3400	205.6800	65.400	1898.53	76.6300	65.510	341.0900	...	35.0000	60.7000	165.0000	181.5300	90.7500	28.029

6757 rows × 75 columns

Figure 2.2 Preprocessed Dataframe

Preprocessing for Breakout Strategy

The code for the preprocessing is illustrated in Figure 5.1. The initial dataset, adjprice.csv, contains adjusted closing prices for various stocks, with each stock represented as a separate column and dates as rows. The first step is to read this data and format the dates correctly. To ensure that the dataset includes all trading days, a complete date range is created. This step is crucial for consistent time series analysis and to handle missing dates. The data is reshaped using the melt function to transform the ticker names into a single column. This format is suitable for group-based operations, such as calculating moving averages and returns. Next, the daily percentage returns are calculated for each stock. The pct_change method computes the percentage change between the current day and the next day, thus generating a dataframe where we can compare individual stocks.

Exploratory Data Analysis

Absolute Static Weights

In this project, Absolute static weights are applied by first establishing fixed quantities of each stock at the beginning and fixed throughout the period. They provide a more straightforward and reliable approach for managing investments. By controlling the fixed allocations over time, this strategy displayed its accuracy and consistency and was successful in generating high sharpe ratio calculation. Using this approach, random generation and standard optimisation techniques were employed to enhance portfolio performance.

Random Generation

With the random generation approach, this strategy commenced by first generating 1000 random portfolios by utilising the historical data given at the beginning. Each portfolio underwent an evaluation process wherein the expected annualised return, volatility (risk), and Sharpe ratio are calculated. Daily returns were computed using the pct.change() method, and volatility was determined by calculating the covariance matrix of the returns. By multiplying this by 252 and taking the square root of the dot product of the portfolio weights, volatility was obtained. These daily returns and volatility were then aggregated to determine the portfolio's overall expected annualised return and expected annualised volatility. Using the sharpe ratio calculation, the overall annualised expected sharpe ratio was also calculated. These findings are analysed using a line graph as depicted in Figure 3.1

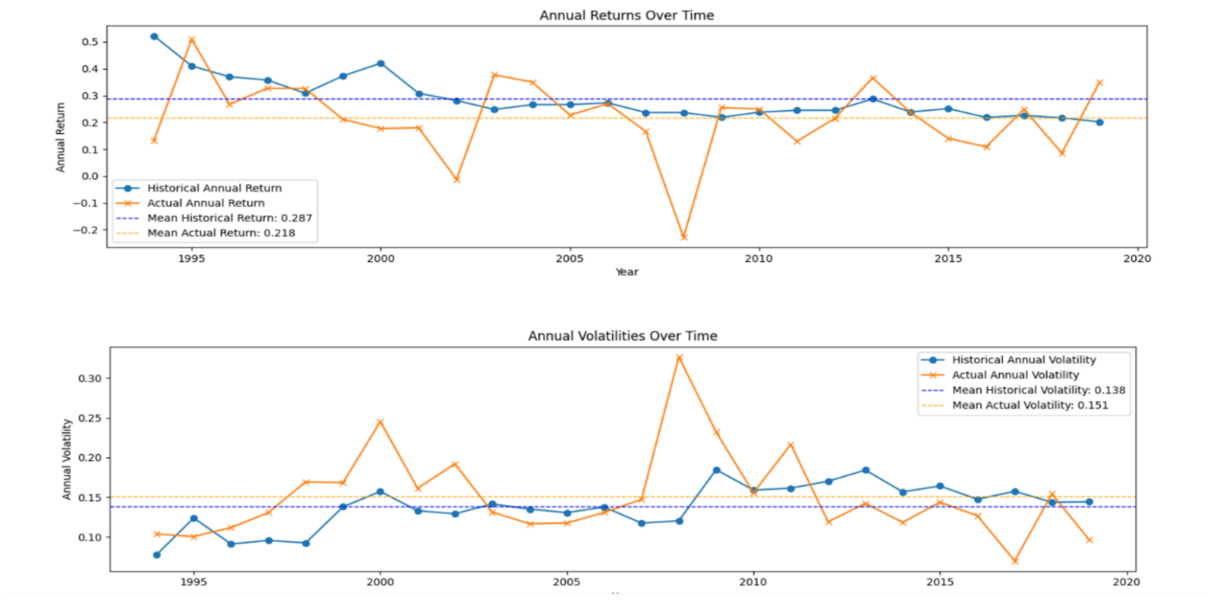


Figure 3.1 The annualised return and annualised volatility for the historical and actual data

In Figure 3.1, the mean historical return stands at 0.287, contrasting with the mean annual return of 0.218. This indicates a tendency for returns to revert to the long-term average over time. The significant drops observed during 2000-2002 and 2008 showcase the repercussions of the dot-com bubble burst and financial crisis, respectively. Furthermore, the mean historical volatility is 0.141, contrasting with the mean actual volatility of 0.153. Notably, volatilities exhibit an inverse relationship with returns. As returns increase over time, volatilities tend to decrease, and vice versa, illustrating their dynamic interplay in investment dynamics

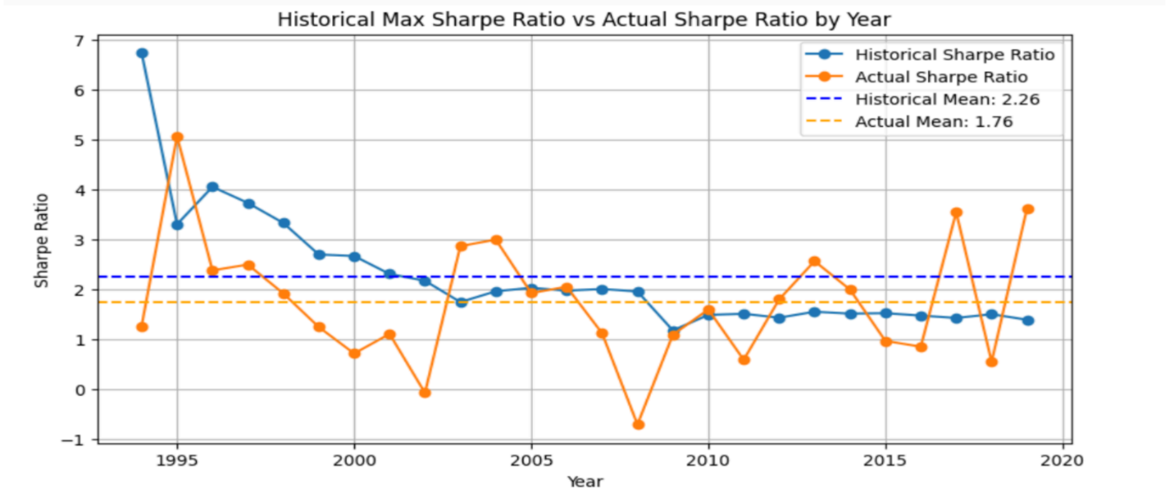


Figure 3.2 The annualised historical sharpe ratio and annualised actual sharpe ratio for random generation

As depicted in Figure 3.2, the historical maximum Sharpe Ratio peaks around 1999 and 2009, with lowest points near 2003 and post-2015. The actual sharpe ratio is lower than the historical. This could be due to changing market conditions and other real-world factors.

Standard Optimisation

In the standard optimisation approach using absolute weights, the portfolio performance is optimised by minimising the negative sharpe ratio set by the objective function. A constraint is set that mandates the sum of the portfolio weights to be 1. Then, with the help of scipy.optimize.minimize function with the SLSQP (Sequential Least Squares Programming) method, an optimal weight for the portfolio is identified. With this process, the aim is to minimise the negative sharpe ratio while adhering to the constraint which ultimately optimises the overall portfolio. After initialising this process, the annual returns, annual volatilities and the annual sharpe ratio is calculated using the optimised weights and historical returns. Furthermore, this result is displayed in figure 3.3 using a line graph

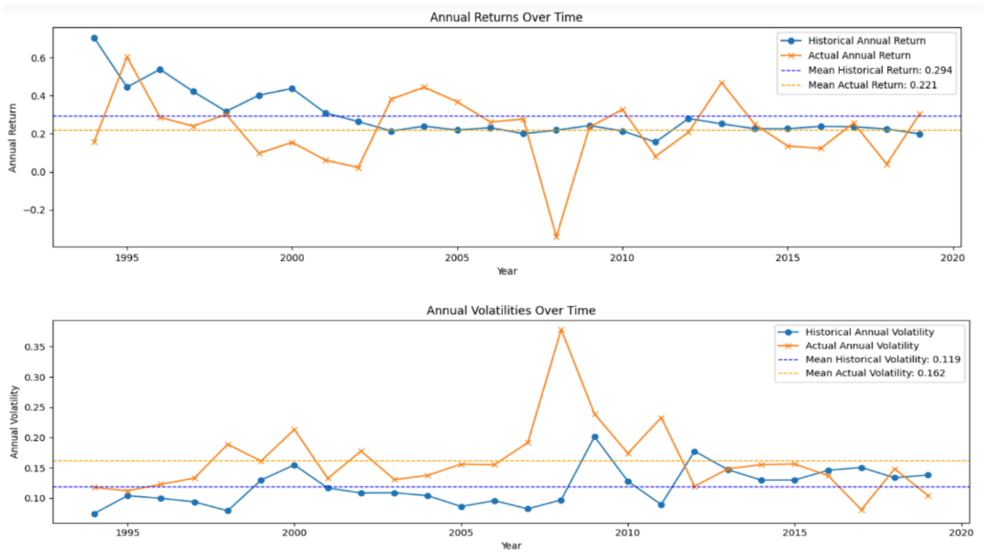


Figure 3.3 The annualised return and annualised volatility for the historical and actual data

In Figure 3.3, the mean annual return is 0.294 whereas the mean historical annual return is 0.221. The comparison between the two showcase significant fluctuations in historical performance, with the mean actual return generally tracking close to the historical average. Similarly, the comparison of mean historical volatility (0.119) and mean actual volatility (0.162) showcase the actual Volatility to align more closely with the historical pattern. This method aims to balance risk and return, with the SLSQP method potentially providing more precise results.

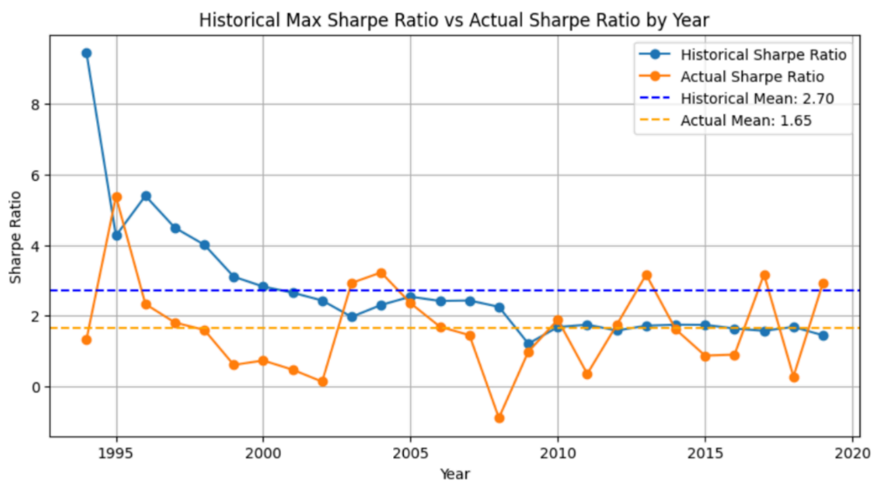


Figure 3.4 The annualised historical sharpe ratio and annualised actual sharpe ratio for standard optimisation

Figure 3.4 illustrates the annual historical max sharpe ratio and the actual sharpe ratio that is calculated using the standard optimisation method. The annual mean historical max sharpe ratio is 2.70 whereas the annual actual sharpe ratio is 1.65. This suggests that there were some underperforming market conditions that led to decrease in the sharpe ratio of the optimised portfolio.

Furthermore, the overall sharpe ratio is 1.76 for random and 1.65 for optimisation.
Despite a slightly lower Sharpe ratio (1.65), the optimisation approach with SLSQP remains preferable due to its consistency. It offers a systematic method for finding optimal portfolio weights, particularly suited for scalability and complex optimization problems compared to random generation.

Relative Static Weights

Relative static weight is a strategy with a fixed distribution of investment across the different assets in the portfolio and has a periodic readjustment that is also fixed for each year. An initial investment of a hundred thousand dollars and ten thousand dollars of monthly investment is chosen for demonstration for this project.

Random Generation

The first approach for relative static weight is random generation where five thousand portfolios were randomly generated. For each portfolio, the portfolio return, volatility and Sharpe ratio were calculated. Portfolio return is calculated as the weighted sum of the mean returns of the individual assets. Volatility is calculated as the standard deviation of the returns and the Sharpe ratio as mentioned earlier is calculated as the risk-free rate subtracted from the average return of the portfolio and the difference is divided by the standard deviation of portfolio returns. A yearly loop is then set for each year and for each year the historical stock data from previous years is used to form the portfolio and stocks are selected if their last-day price of the previous year is greater than their mean price of the previous year. Daily returns and covariance matrix are calculated for the stock selected. Daily returns are calculated using the pct. change() method in percent and covariance matrix is calculated to see how two stocks are correlated. Positive shows that the two stocks tend to move in the same direction while negative shows that the two stocks move the opposite way. Optimal weights are then determined and the portfolio's performance is stimulated for the current year with monthly investment decided in the beginning. For each, annual returns, volatility, Sharpe ratio and profit are calculated and stored.

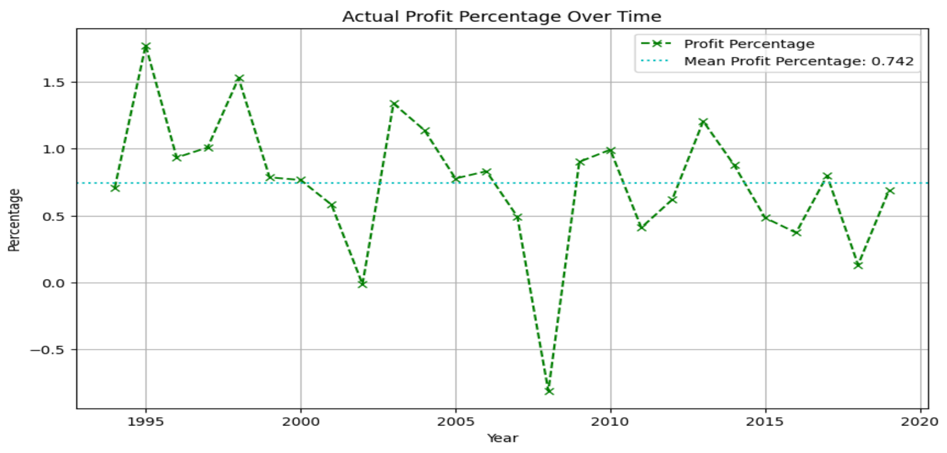


Figure 4.1 Actual Profit Percentage Over Time For Random Generation

Profit percentage measures the profit earned from individual stocks or the portfolio relative to their revenue indicating profitability efficiency. Figure 4.1 illustrates the profit percentage over time together with the mean profit percentage. Initial values showed slight profits each year but percentages were relatively low ranging from 0.17% to 1.75%. This shows modest growth of investment as the return did not outperform the additional investments although the portfolio increased.

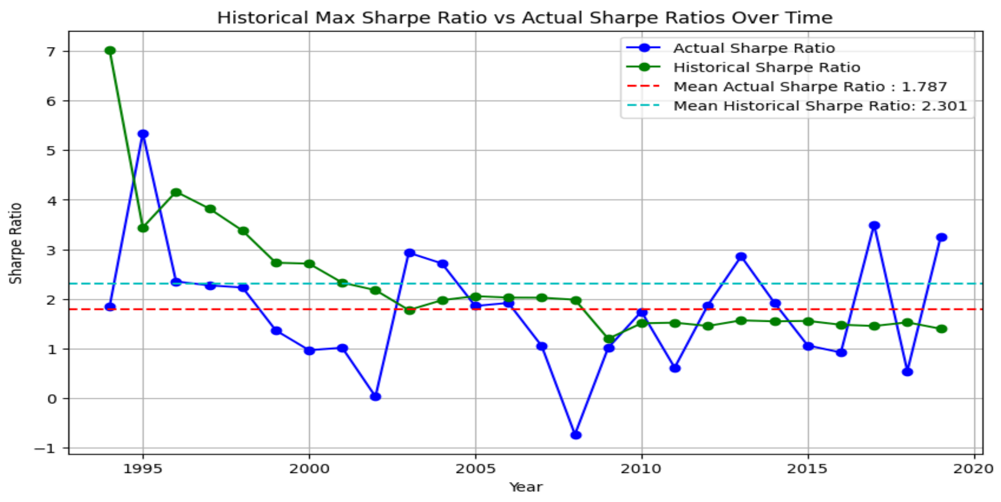


Figure 4.2 Max Sharpe Ratio and Actual Sharpe Ratios Over Time Plotted with Mean Average Sharpe Ratio and Mean Historical Sharpe Ratio for Random Generation

Figure 4.2 shows the historical max Sharpe ratio and the actual Sharpe ratio over time together with the mean for both the Sharpe ratios. Historical Sharpe ratios were consistently higher than actual Sharpe ratios. This indicates over-optimistic forecasts and a lower actual Sharpe ratio than the historical Sharpe ratio could also reflect market unpredictability.

Standard Optimisation

Standard optimization uses mathematical techniques to find the optimal weights for each asset in the portfolio. The same process as random generation is then conducted by calculating the iterating through each year and using the previous year's data to optimise the portfolio. The average price for the previous year is then calculated and stocks with greater prices from the previous day than the previous year's average price are the data. The portfolio is then optimised using the functions neg_sharpe_ratio which computes negative Sharpe ratio for optimization, optimize_portfolio which uses minimise from scipy.optimize with constraints for weight sum = 1 and bounds [0, 1] and SLSQP method, an inbuilt method used to solve non-linear optimisation problem. Monthly investments are then stimulated using the optimised weights and historical returns. Annualised average return volatility, Sharpe ratio and profit percentage are computed.

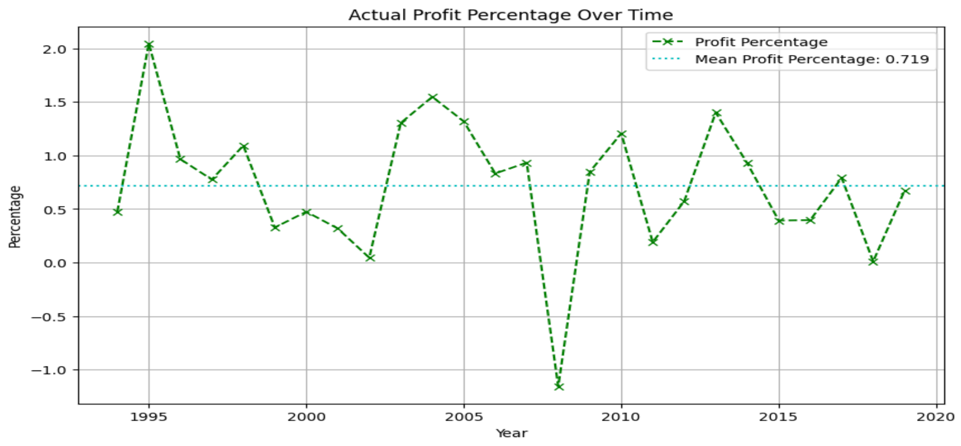


Figure 4.3 Actual Profit Percentage Over Time For Standard Optimisation

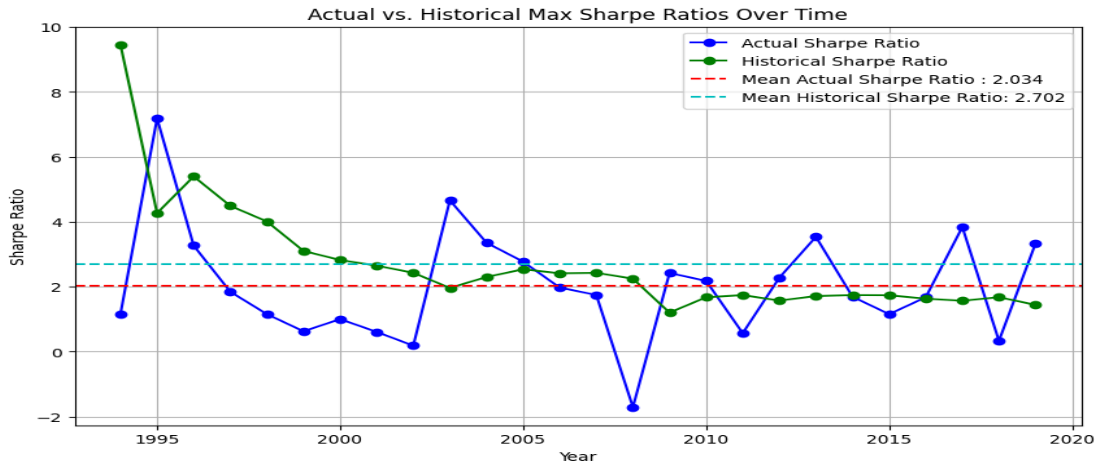


Figure 4.4 Max Sharpe Ratio and Actual Sharpe Ratios Over Time Plotted with Mean Average Sharpe Ratio and Mean Historical Sharpe Ratio for Standard Optimisation

Figure 4.3 demonstrates the profit percentage over time for the standard optimisation method and it shows that the Profit percentage is mostly low ranging from 0.05% to 2.04% with only a few years achieving over 1%. Figure 4.4 shows the max historical Sharpe ratio and actual Sharpe ratio together with the average of both and it can be seen that Historical max Sharpe ratios are often higher than actual values and range from slight gains to modest increases in the final portfolio value. This indicates the market will change over time and the classification strategy cannot fit the change well.

It can be observed that the mean historical max Sharpe ratio of 2.034 and mean actual Sharpe ratio of 2.702 for the standard optimization is higher than the mean historical max Sharpe ratio of 1.787 and mean actual Sharpe ratio of 2.301 for random generation, making it a better method between the two. Though the mean profit percentage for standard optimization is higher, it is not significant enough to support random generation as a better method. This is also true as standard optimisation is more stable and uses mathematical techniques to create portfolios instead of randomly generating them.

Data modelling

Equal Weights: Break out strategy

This section discusses the strategy, development, optimization and implementation of the breakout strategy. The breakout strategy makes use of equal weights ensuring that stocks in the portfolio have the same initial value relative to other stocks in the portfolio. The development of this model was inspired from the book: A beginners guide to the stock market. (2019)

Strategy Implementation (shown in figure 5.1)

1. Calculating Moving Averages (Yassinehammou27, n.d.):
   Short-term (5-day) and long-term (15-day) moving averages are computed for each ticker. These moving averages help smooth out price data, highlighting trends and potential breakout points.

2. Generating Buy and Sell Signals (Yassinehammou27, n.d.):
   Buy signals are generated when the price exceeds both the short-term and long-term moving averages whereas sell signals are triggered when the price falls below both moving averages. Positions are determined based on these signals, where a buy signal indicates entering a long position and a sell signal indicates exiting the position. The process of generating these signals is illustrated in figure 5.2, which is key in showing how to leverage the breakout strategy.

3. Maintaining Positions and Computing Returns:
   Holding positions are calculated by forward-filling the positions, ensuring continuity between signals. Strategy returns are calculated by multiplying holding positions with log returns.

4. Implementing Stop-Loss and Take-Profit Mechanisms:
   To optimise the strategy,we used exhaustive grid search to find the optimal parameters to maximise return (3.2. Tuning the Hyper-Parameters of an Estimator — Scikit-Learn 0.22 Documentation, 2012). Therefore using the grid search results the stop-loss is set at -3% and a take-profit at 100%, capping losses and gains respectively. These mechanisms help protect against significant losses during market downturns and lock in profits during uptrends.

Behind the Success of Breakout Strategies

Breakout strategies capitalise on market momentum (2019). A breakout often signals the start of a new trend driven by shifts in market sentiment, leading to extended price movements and significant gains. These levels are critical in technical analysis. A breakout above resistance or below support indicates a change in supply-demand dynamics, often resulting in strong price movements as new trends emerge (2019). Breakouts are typically accompanied by increased trading volume, confirming the breakout's validity and reducing the likelihood of false signals. This volume surge indicates strong investor interest and participation, further driving the trend. Financial markets exhibit trends due to factors such as economic fundamentals, investor behaviour, and market cycles (2019). Breakout strategies leverage these trends, allowing traders to profit from sustained price movements. Since we only had access to closing price data this helps us generate and analyse long-term trends of several stocks in our dataset thus making the breakout strategy even more effective due to the availability of data

Results and Analysis

Pre-Optimization Results:

The initial implementation of the breakout strategy yielded an annualised expected return of 5.15%, with volatility at 14.52%, and a Sharpe ratio of 0.35. The cumulative returns graph as shown in figure 5.3 in the appendix, indicates significant gains over time but also highlights periods of substantial drawdowns, particularly during financial crises. We can see several trends illustrated in figure 5.3, with the model performing well during growing or stable market periods but suffered heavy losses during major downturns such as the Dot-com Bubble Burst (2000-2002), September 11 Attacks (2001), the 2002 Stock Market Downturn, the Financial Crisis of 2007-2008, and the 2018 Stock Market Correction. These events underscore the need for mechanisms to mitigate losses during turbulent market conditions.

Post-Optimization Results:

Optimising the stop-loss and take-profit parameters significantly improved the strategy's performance. The annualised expected return increased to 38.22%, with reduced volatility of 12.15%, resulting in a much higher Sharpe ratio of 3.14. Implementing these mechanisms greatly enhanced the strategy's robustness and overall profitability, this illustrates the cumulative capability of this model as once the losses are smoothed over the cumulative gains rise exponentially leading to higher model performance. A representative of this performance is figure 5.4 showing the yearly returns of the optimised breakout strategy with an intriguing feature being the gain that we observe immediately after periods of financial turmoil. This is likely due to the breakout strategy making the maximal use of market correction and uptrend by identifying long-periods of bull markets.

Dynamic Weights: Using Machine Learning (SVR)

This section discusses our approach to solving dynamic weights using machine learning methods. Dynamic weights are defined as weights that are a function of recent stock performances.

We employ Support Vector Regression (SVR) models on historical stock prices to predict future returns and volatility. These predictions are used to optimise a portfolio to maximise the Sharpe ratio and evaluate the actual Sharpe ratio of the portfolio.

SVR is a type of Support Vector Machine (SVM) used for regression tasks. It works by finding a function that approximates target values within a specified margin of tolerance. Financial time series data, such as stock returns and volatility, often exhibit nonlinear patterns. SVR, particularly with an RBF (Radial Basis Function) kernel, can effectively model these complex relationships, making it suitable for predicting financial metrics (Ghanbari & Arian, 2019).

Details of Process: SVR in Dynamic Weights

For each stock, daily returns are calculated using the percentage change of stock prices (Figure 7.1). The rolling standard deviation of returns over a 30-day window represents the stock's volatility. Features for model training are prepared by calculating rolling statistics (mean, standard deviation, max, min) of returns and volatility over multiple windows (90, 180, and 360 days). These features capture different time horizons of stock behaviour, providing the model with comprehensive information. Standardisation ensures that each feature contributes equally to the model's predictions.

After doing features engineering, Figure 7.2 shows that TimeSeriesSplit is used to preserve the temporal order when splitting data into training and testing sets, avoiding data leakage. The script uses TimeSeriesSplit to create five splits for cross-validation, ensuring that each training set precedes the test set. For each cross-validation split, an SVR model with an RBF kernel is trained on the prepared features. The model is evaluated using Mean Squared Error (MSE) to measure prediction accuracy. The model with the lowest MSE for both returns and volatility for each stock is selected and stored along with the corresponding feature scalers for future predictions.

To minimise the negative Sharpe ratio, the optimize_portfolio function is used to find the optimal weights for each stock in the portfolio. Constraints ensure that the sum of portfolio weights is 1 (fully invested) and each weight is between 0 and 1. Sequential Least Squares Programming (SLSQP) is employed to solve constrained optimization problems.

As for calculation of Actual Sharpe Ratio, the portfolio's performance is evaluated by: calculating daily returns based on historical data and optimised weights; computing the mean and standard deviation of daily returns; annualizing these metrics to determine the annualised return and volatility; calculating the Actual Sharpe ratio, which indicates the portfolio's risk-adjusted return.

The Figure 7.3 indicates significant fluctuations in both predicted and actual Sharpe ratios from 1999 to 2017. Some periods exhibit high spikes in the predicted Sharpe ratio, indicating that the model sometimes predicts very high returns relative to risk.

Result of SVR in Dynamic Weights

Most traditional methods, such as random generation and standard optimization, achieve actual Sharpe ratios close to but below 2. The SVR model, despite some differences, consistently outperforms these traditional methods, achieving an actual mean Sharpe ratio of 2.92. It is a wonderful value of sharpe ratio in real word.

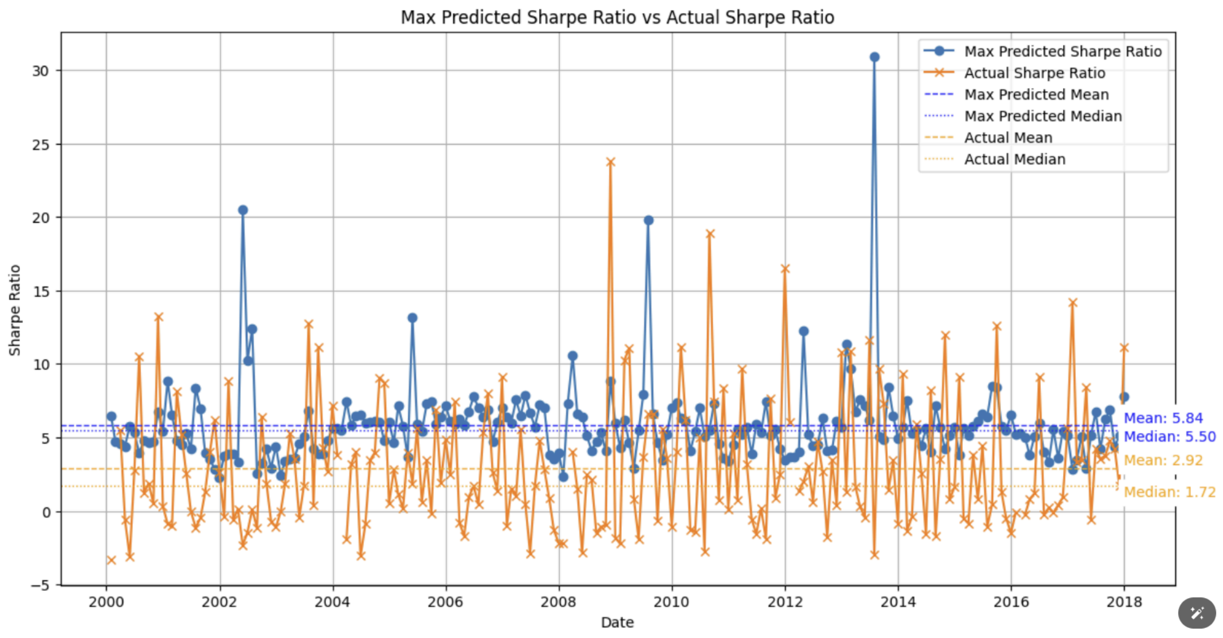


Figure 7.3: max predicted sharpe ratio and actual sharpe ratio from support vector regression

Conclusion

Performance Comparison

The analysis conducted shows that Support Vector Regression(SVR) is the best method for the prediction of the Sharpe ratios and is more effective than the breakout strategy that was also used to make predictions. Breakout strategy has a better sharpe ratio and is better for predicting short term while SVR does a better job predicting the long term SVR also outperformed the traditional methods such as static absolute weight and static relative weight analysis. This can be concluded as the traditional methods for both random generation and standard optimisation achieved an actual Sharpe ratio below or around 2 while the SVR model despite some differences consistently outperformed achieving an actual mean Sharpe ratio of 2.92.

Benefits of SVR in modelling

The SVR model developed to predict the Sharpe ratios can assist in better risk-adjusted returns allowing the interested parties to make informed decisions when allocating assets which enhances portfolio performance. The dynamic weights can be adjusted based on SVR to ensure portfolio performance remains resilient and adaptive to market changes ultimately managing the risk for the portfolio. This makes SVR a valuable tool for long-term investment strategy.

Limitations and Data Considerations

Despite the strengths of SVR for predicting, there are some limitations. The predictions are conducted using the top 75 stocks this could show biases as the best stocks are used to predict the future prices of stocks. This could result in over-optimistic forecasts. Additionally, there was limited data available in the dataset. The dataset only consisted of daily closing prices for each stock for over 16 years. Data such as volume, opening prices and returns on stocks would have allowed further analysis and underprovided a more comprehensive understanding. More diverse dataset could help improve the accuracy and robustness of predictive models.

Reference List

3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.22 documentation. (2012). Scikit-Learn.org.
https://scikit-learn.org/stable/modules/grid_search.html

Algorithmic-Trading/strategy_break_out_moving_avg.ipynb at master. yassinehammou27/Algorithmic-Trading. (n.d.). GitHub. Retrieved May 25, 2024, from https://github.com/yassinehammou27/Algorithmic-Trading/blob/master/strategy_break_out_moving_avg.ipynb

Ghanbari, M., & Arian, H. (2019). Forecasting Stock Market with Support Vector Regression and Butterfly Optimization Algorithm. arXiv.
https://doi.org/10.48550/arXiv.1905.11462

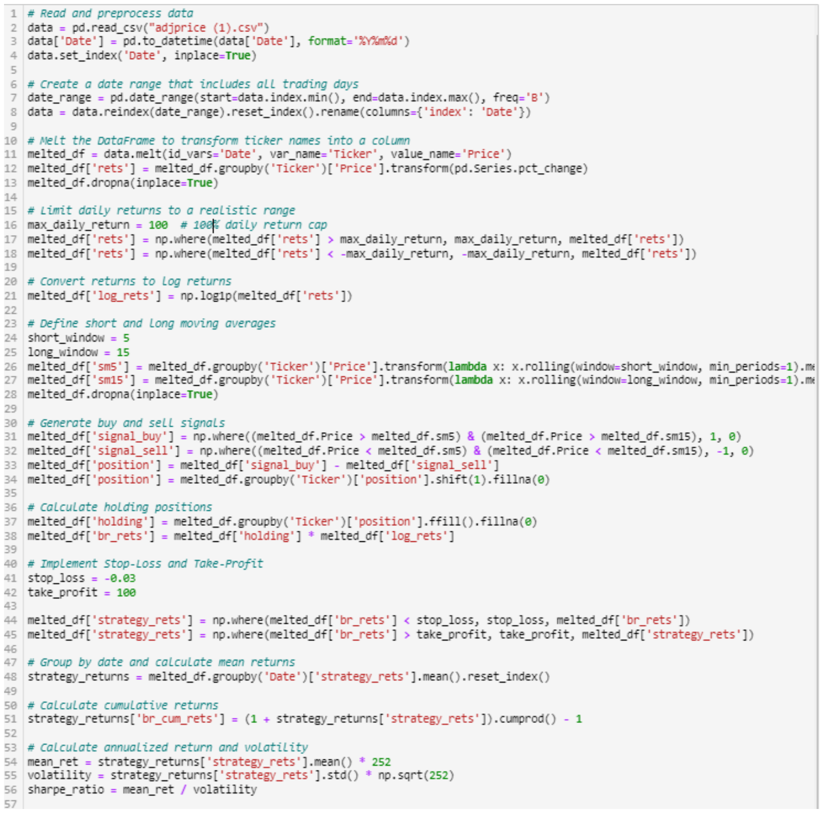
j87as6. (2021, February 12). The S&P 500 - A Complete Guide for Active Traders. CenterPoint Securities. https://centerpointsecurities.com/sp-500/

Kratter, M. R. (2019). A beginner's guide to the stock market. Trader University.

S&P Dow Jones Indices. (2024). S&P 500® - S&P Dow Jones Indices.
https://www.spglobal.com/spdji/en/indices/equity/sp-500/#overview

The Anatomy of Trading Breakouts. (2020). Investopedia.
https://www.investopedia.com/articles/trading/08/trading-breakouts.asp

Appendix



```python
# Read and preprocess data
data = pd.read_csv("adjprice (1).csv")
data['Date'] = pd.to_datetime(data['Date'], format='%Y%m%d')
data.set_index('Date', inplace=True)

# Create a date range that includes all trading days
date_range = pd.date_range(start=data.index.min(), end=data.index.max(), freq='B')
data = data.reindex(date_range).reset_index().rename(columns={'index': 'Date'})

# Melt the DataFrame to transform ticker names into a column
melted_df = data.melt(id_vars='Date', var_name='Ticker', value_name='Price')
melted_df['rets'] = melted_df.groupby('Ticker')['Price'].transform(pd.Series.pct_change)
melted_df.dropna(inplace=True)

# Limit daily returns to a realistic range
max_daily_return = 100  # 100% daily return cap
melted_df['rets'] = np.where(melted_df['rets'] > max_daily_return, max_daily_return, melted_df['rets'])
melted_df['rets'] = np.where(melted_df['rets'] < -max_daily_return, -max_daily_return, melted_df['rets'])

# Convert returns to log returns
melted_df['log_rets'] = np.log1p(melted_df['rets'])

# Define short and long moving averages
short_window = 5
long_window = 15
melted_df['sm5'] = melted_df.groupby('Ticker')['Price'].transform(lambda x: x.rolling(window=short_window, min_periods=1).m
melted_df['sm15'] = melted_df.groupby('Ticker')['Price'].transform(lambda x: x.rolling(window=long_window, min_periods=1).m
melted_df.dropna(inplace=True)

# Generate buy and sell signals
melted_df['signal_buy'] = np.where((melted_df.Price > melted_df.sm5) & (melted_df.Price > melted_df.sm15), 1, 0)
melted_df['signal_sell'] = np.where((melted_df.Price < melted_df.sm5) & (melted_df.Price < melted_df.sm15), -1, 0)
melted_df['position'] = melted_df['signal_buy'] - melted_df['signal_sell']
melted_df['position'] = melted_df.groupby('Ticker')['position'].shift(1).fillna(0)

# Calculate holding positions
melted_df['holding'] = melted_df.groupby('Ticker')['position'].ffill().fillna(0)
melted_df['br_rets'] = melted_df['holding'] * melted_df['log_rets']

# Implement Stop-Loss and Take-Profit
stop_loss = -0.03
take_profit = 100

melted_df['strategy_rets'] = np.where(melted_df['br_rets'] < stop_loss, stop_loss, melted_df['br_rets'])
melted_df['strategy_rets'] = np.where(melted_df['br_rets'] > take_profit, take_profit, melted_df['strategy_rets'])

# Group by date and calculate mean returns
strategy_returns = melted_df.groupby('Date')['strategy_rets'].mean().reset_index()

# Calculate cumulative returns
strategy_returns['br_cum_rets'] = (1 + strategy_returns['strategy_rets']).cumprod() - 1

# Calculate annualized return and volatility
mean_ret = strategy_returns['strategy_rets'].mean() * 252
volatility = strategy_returns['strategy_rets'].std() * np.sqrt(252)
sharpe_ratio = mean_ret / volatility
```

Figure 5.1: Code for breakout strategy

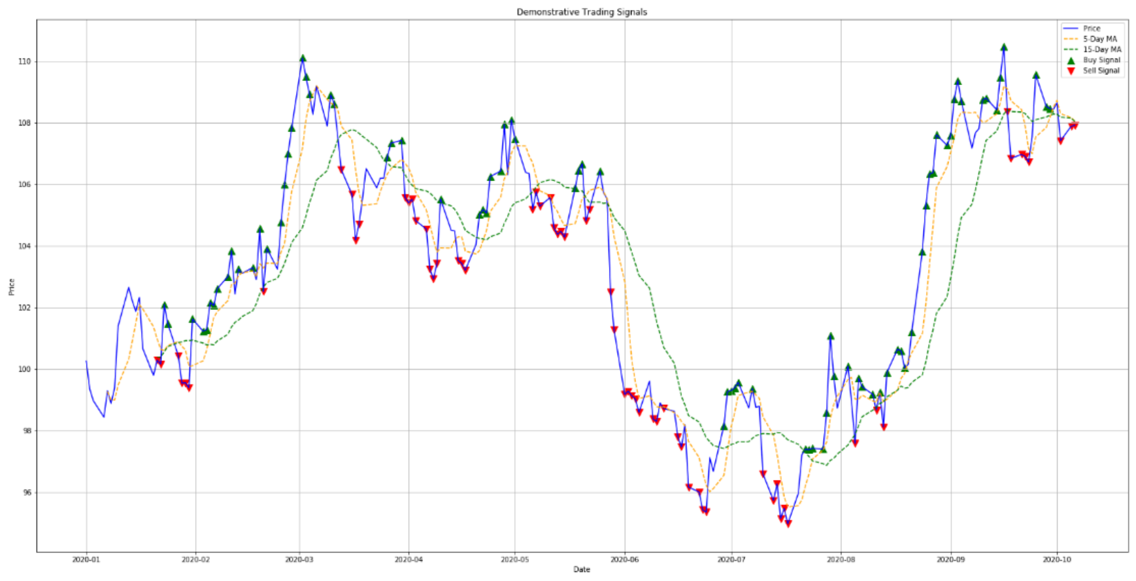


Figure 5.2: illustrating of trading signals generated on the dataset

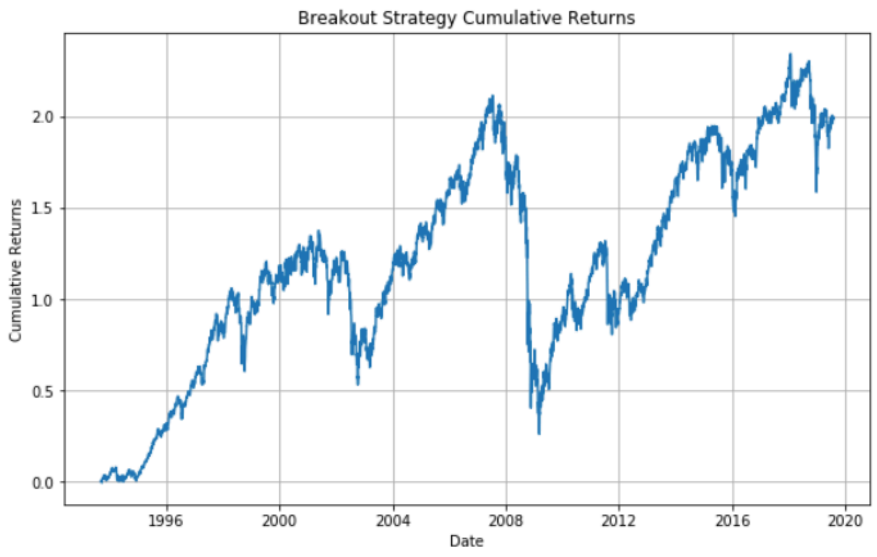


Figure 5.3: Cumulative gains of breakout strategy, highlighting overall model performance

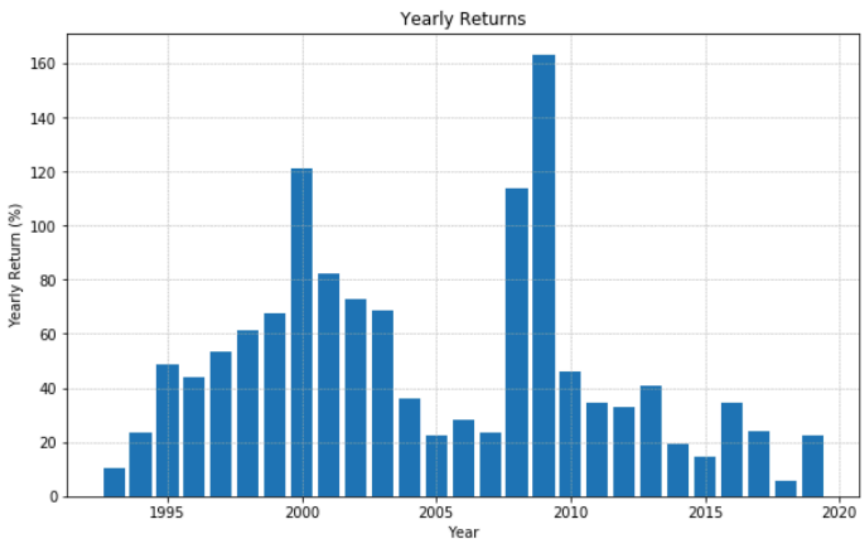
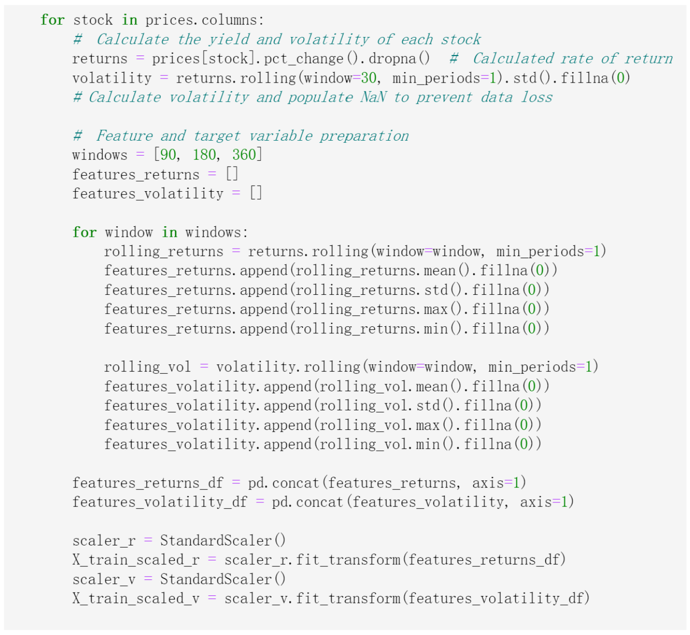
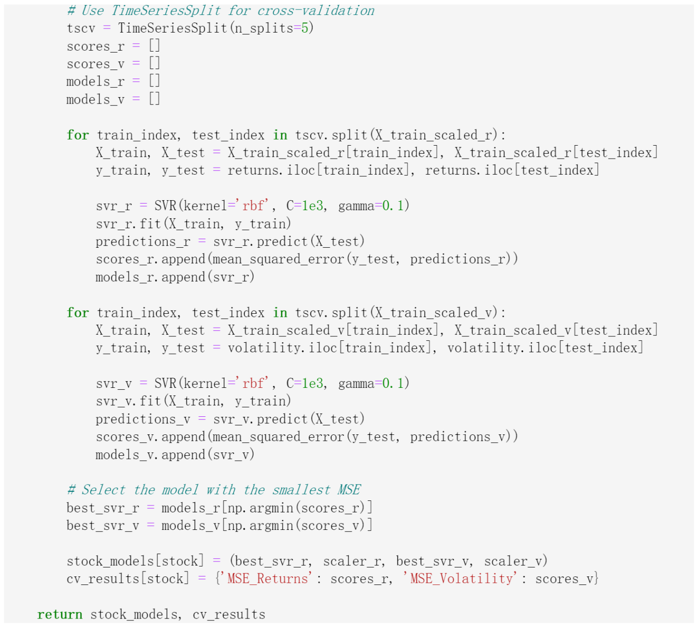


Figure 5.4: yearly gains for optimised model



```
for stock in prices.columns:
    #  Calculate the yield and volatility of each stock
    returns = prices[stock].pct_change().dropna()   #  Calculated rate of return
    volatility = returns.rolling(window=30, min_periods=1).std().fillna(0)
    # Calculate volatility and populate NaN to prevent data loss

    #  Feature and target variable preparation
    windows = [90, 180, 360]
    features_returns = []
    features_volatility = []

    for window in windows:
        rolling_returns = returns.rolling(window=window, min_periods=1)
        features_returns.append(rolling_returns.mean().fillna(0))
        features_returns.append(rolling_returns.std().fillna(0))
        features_returns.append(rolling_returns.max().fillna(0))
        features_returns.append(rolling_returns.min().fillna(0))

        rolling_vol = volatility.rolling(window=window, min_periods=1)
        features_volatility.append(rolling_vol.mean().fillna(0))
        features_volatility.append(rolling_vol.std().fillna(0))
        features_volatility.append(rolling_vol.max().fillna(0))
        features_volatility.append(rolling_vol.min().fillna(0))

    features_returns_df = pd.concat(features_returns, axis=1)
    features_volatility_df = pd.concat(features_volatility, axis=1)

    scaler_r = StandardScaler()
    X_train_scaled_r = scaler_r.fit_transform(features_returns_df)
    scaler_v = StandardScaler()
    X_train_scaled_v = scaler_v.fit_transform(features_volatility_df)
```

Figure 7.1: first part code about features engineering of support vector regression



```python
        # Use TimeSeriesSplit for cross-validation
        tscv = TimeSeriesSplit(n_splits=5)
        scores_r = []
        scores_v = []
        models_r = []
        models_v = []

        for train_index, test_index in tscv.split(X_train_scaled_r):
            X_train, X_test = X_train_scaled_r[train_index], X_train_scaled_r[test_index]
            y_train, y_test = returns.iloc[train_index], returns.iloc[test_index]

            svr_r = SVR(kernel='rbf', C=1e3, gamma=0.1)
            svr_r.fit(X_train, y_train)
            predictions_r = svr_r.predict(X_test)
            scores_r.append(mean_squared_error(y_test, predictions_r))
            models_r.append(svr_r)

        for train_index, test_index in tscv.split(X_train_scaled_v):
            X_train, X_test = X_train_scaled_v[train_index], X_train_scaled_v[test_index]
            y_train, y_test = volatility.iloc[train_index], volatility.iloc[test_index]

            svr_v = SVR(kernel='rbf', C=1e3, gamma=0.1)
            svr_v.fit(X_train, y_train)
            predictions_v = svr_v.predict(X_test)
            scores_v.append(mean_squared_error(y_test, predictions_v))
            models_v.append(svr_v)

        # Select the model with the smallest MSE
        best_svr_r = models_r[np.argmin(scores_r)]
        best_svr_v = models_v[np.argmin(scores_v)]

        stock_models[stock] = (best_svr_r, scaler_r, best_svr_v, scaler_v)
        cv_results[stock] = {'MSE_Returns': scores_r, 'MSE_Volatility': scores_v}

    return stock_models, cv_results
```

Figure 7.2: second part code of support vector regression