Trading strategies on python Machine Learning Part

Chenjie LI, ESILV Msc Financial Engineering

October 20, 2024

Abstract

In this paper, we present and implement several advanced trading strategies and volatility prediction models using a range of machine learning techniques, including K-Nearest Neighbors (KNN), DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and Random Forest. These machine learning-driven strategies are designed to identify profitable opportunities by leveraging historical market data and predicting price movements more accurately. We have backtested each strategy using real-world equity data obtained from Yahoo Finance, ensuring practical relevance and performance evaluation. For each strategy, we provide detailed visualizations highlighting the generated long and short signals, accompanied by calculations of the corresponding returns. These visualizations enable an intuitive understanding of how the models perform under various market conditions. Furthermore, the implementation of these strategies, including the prediction of volatility and signal generation, is available in Python, allowing for replication and further experimentation. You can access the complete Python code and detailed analysis at: Chenjie's Github Trading strategies

Introduction

Overview

Machine learning-based trading strategies have become increasingly popular in the financial markets, offering systematic approaches that leverage algorithms and data-driven models to maximize returns and manage risk. These strategies utilize a variety of machine learning techniques, from clustering algorithms like DBSCAN to classification methods like K-Nearest Neighbors (KNN) and ensemble models such as Random Forest. By applying machine learning, traders can uncover patterns in historical data, identify market trends, and generate predictive insights that are difficult to detect through traditional methods.

Having a well-defined machine learning strategy is essential for success in today's fast-moving markets. Machine learning models not only provide a framework for making more objective decisions but also help traders optimize for performance and mitigate human biases. Techniques such as cross-validation, feature engineering, and hyperparameter tuning are crucial in ensuring that these models generalize well to unseen market conditions while maintaining consistency.

Developing and understanding various machine learning-driven trading strategies is vital for adapting to the complexities of modern financial markets. Whether you are an algorithmic trader or a quantitative analyst, a well-crafted strategy powered by machine learning can greatly enhance your ability to interpret large datasets, adjust to evolving market dynamics, and achieve your financial goals. By integrating machine learning models, traders can automate decision-making processes, reduce manual intervention, and improve the probability of profitable trades.

For each machine learning-based strategy, we'll explain its mechanics, the rationale behind its use, and how to backtest and optimize it using Python. Our main goal is to implement these strategies programmatically to enhance our technical knowledge and analytical skills. While we may not apply every backtesting and optimization technique to each model, the hands-on approach with machine learning will undoubtedly deepen your understanding and make the process enjoyable.

Contents

1 Conventional notation of our paper

1.1 variable used

- σ_i is the annualized volatility for a stock i.
- R_i is the average return of a stock.
- R_f is the risk-free rate.
- R_m is the average return of the market.
- $\text{Var}(R_i)$ is the variance of a stock i.
- $\text{Cov}(R_i, R_m)$ is the co-variance between a stock i and the market m.
- ϵ_i is the error term.

1.2 Metric used in our paper

- *Alpha* : Represents the excess return of an investment relative to the return of a benchmark index.
  - Formula: $\alpha = R_i - (R_f + \beta(R_m - R_f))$
  - $\alpha > 0$: Indicates that the investment has outperformed its benchmark, suggesting that the investment manager's strategy has added value.
  - $\alpha < 0$: Indicates that the investment has under-performed its benchmark, suggesting that the investment manager's strategy has detracted value.
  - $\alpha = 0$: Indicates that the investment has performed in line with its benchmark, suggesting that the investment manager's strategy has neither added nor detracted value.

- *Beta* : Measures the volatility or systematic risk of an investment relative to the overall market.
  - Formula: $\beta = \frac{\text{Cov}(R_i, R_m)}{\text{Var}(R_m)}$
  - $\beta > 1$: Indicates that the investment is more volatile than the market, suggesting higher risk and potentially higher returns.
  - $\beta < 1$: Indicates that the investment is less volatile than the market, suggesting lower risk and potentially lower returns.
  - $\beta = 1$: Indicates that the investment's volatility is in line with the market, suggesting average market risk.

- *Correlation* : Measures the degree to which two securities move in relation to each other.
  - Formula: $\rho = \frac{\text{Cov}(R_i, R_m)}{\sigma_i \sigma_m}$
  - $\rho = 1$: Indicates a perfect positive correlation, suggesting that the two securities move in the same direction.
  - $\rho = -1$: Indicates a perfect negative correlation, suggesting that the two securities move in opposite directions.
  - $\rho = 0$: Indicates no correlation, suggesting that the movements of the two securities are unrelated.

- *SharpeRatio* : Measures the risk-adjusted return of an investment.
  - Formula: Sharpe Ratio = $\frac{R_i - R_f}{\sigma_i}$
  - Higher Sharpe Ratio: Indicates better risk-adjusted returns.
  - Lower Sharpe Ratio: Indicates worse risk-adjusted returns.

- *SortinoRatio* : Measures the risk-adjusted return of an investment, similar to the Sharpe Ratio, but only considers downside risk.
  - Formula: Sortino Ratio = $\frac{R_i - R_f}{\sigma_d}$
  - Higher Sortino Ratio: Indicates better risk-adjusted returns with respect to downside risk.
  - Lower Sortino Ratio: Indicates worse risk-adjusted returns with respect to downside risk.
- *UpsideCapture* : Measures a portfolio's performance in up markets relative to a benchmark.
  - Formula: Upside Capture = $\frac{\text{Portfolio Return in Up Markets}}{\text{Benchmark Return in Up Markets}} \times 100$
  - Upside Capture ¿ 100: Indicates the portfolio has outperformed the benchmark in up markets.
  - Upside Capture ¡ 100: Indicates the portfolio has under-performed the benchmark in up markets.
- *DownsideCapture* : Measures a portfolio's performance in down markets relative to a benchmark.
  - Formula: Downside Capture = $\frac{\text{Portfolio Return in Down Markets}}{\text{Benchmark Return in Down Markets}} \times 100$
  - Downside Capture ¡ 100: Indicates the portfolio has outperformed the benchmark in down markets.
  - Downside Capture ¿ 100: Indicates the portfolio has under-performed the benchmark in down markets.
- *AnnualizedReturn* : Measures the geometric average amount of money earned by an investment each year over a given time period.
  - Formula: Annualized Return = $\left(\prod_{t=1}^{T}(1 + R_t)\right)^{\frac{252}{T}} - 1$
  - Higher Annualized Return: Indicates better performance.
  - Lower Annualized Return: Indicates worse performance.
- *CumulativeReturn* : Measures the total change in the value of an investment over a set time period.
  - Formula: Cumulative Return = $\prod_{t=1}^{T}(1 + R_t) - 1$
  - Higher Cumulative Return: Indicates better performance.
  - Lower Cumulative Return: Indicates worse performance.
- *AnnualizedRisk* : Measures the annualized standard deviation of returns, representing the investment's volatility.
  - Formula: Annualized Risk = $\sigma_i \times \sqrt{252}$
  - Higher Annualized Risk: Indicates higher volatility and potential risk.
  - Lower Annualized Risk: Indicates lower volatility and potential risk.
- *MaximumDrawdown* : Measures the largest drop from a peak to a trough of an investment before a new peak is attained.
  - Formula: Maximum Drawdown = $\min\left(\frac{C_t - P_t}{P_t}\right)$ where C_t is the cumulative return at time t and P_t is the peak return before t.
  - Lower Maximum Drawdown: Indicates better performance in avoiding large losses.
  - Higher Maximum Drawdown: Indicates worse performance in avoiding large losses.

2 Global challenges for our trading strategies

All our strategies implemented and back tested with real data from Yahoo Finance face similar challenges listed and each strategy represents specific risks and will be detailed on each section.

1. **Data Quality Issues:**
    - *Incomplete or Incorrect Data*: Stock volatility data from sources like Yahoo Finance or Bloomberg may be incomplete or incorrect, which can affect the accuracy of the low and high volatility portfolios.
2. **Market Efficiency:**
    - *Efficient Market Hypothesis*: According to the efficient market hypothesis, stock prices already reflect all available information. This could limit the effectiveness of the low volatility anomaly strategy, or quaterly published earning as market efficiency might diminish the potential advantage of investing in low volatility stocks.
3. **Market Conditions:**
    - *Market Trends*: Market trends and macroeconomic conditions can affect stock volatility and thereby impact the performance of the low and high volatility portfolios. For example, during periods of high market stress, the performance of low volatility stocks may not align with historical trends.
    - *Sector Rotation*: Shifts in sector performance may influence the volatility of stocks and affect the strategy's effectiveness.
4. **Stock-Specific Factors:**
    - *Company-Specific Events*: Earnings reports, management changes, or other significant company-specific events can cause abrupt changes in stock volatility, affecting portfolio stability.
    - *Sector-Specific Volatility*: Stocks within certain sectors may exhibit higher or lower volatility due to sector-specific factors, which could impact the results of the low volatility strategy.
5. **Behavioral Biases:**
    - *Investor Behavior*: Investor biases, such as overreacting to short-term market movements or news, can affect stock volatility and potentially distort the outcomes of the strategy.
    - *Herding Effect*: The tendency for investors to follow market trends or other investors can impact the volatility and performance of the stocks in the portfolio.

3 Machine Learning - KNN

3.1 KNN - stock trend prediction Strategy

k-Nearest Neighbors (k-NN) is a simple yet powerful machine learning algorithm used for predicting stock trends and prices. By analyzing historical stock data, k-NN identifies the most similar past instances to make predictions about future movements. In trend prediction, the algorithm determines whether the stock will go up or down by examining the majority class among the closest historical data points. For price prediction, it estimates the future stock price by averaging the prices of these nearest neighbors. The effectiveness of k-NN is evaluated by comparing the predicted trends and prices to the actual outcomes, providing insights into its accuracy and precision. k-NN's strength lies in its simplicity and ability to provide intuitive predictions based on historical patterns.

1. **Function:** `loadDataset`
   This function loads data from a CSV file and splits it into training and test datasets based on a given ratio.

   - It fetches data from Yahoo Finance and processes it to be used in the k-nearest neighbors (k-NN) algorithm.
   - The data is randomly split into training and test sets based on the `split` ratio:

   $$\text{split\_ratio} = \frac{\text{number of training examples}}{\text{total number of examples}}$$

   If a random number generated for a data point is less than the `split` ratio, that data point is added to the training set; otherwise, it is added to the test set.

2. **Function:** `euclideanDistance`
   This function calculates the Euclidean distance between two data points.

   - The Euclidean distance between two points $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ in an n-dimensional space is given by:

   $$d(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

   where a_i and b_i are the i^{th} features of points A and B, respectively.
   - This distance measure is used to determine how "close" two instances are in the feature space. A smaller distance indicates that the instances are more similar.

3. **Function:** `getNeighbors`
   This function finds the `k` nearest neighbors of a given test instance.

   - For a given test instance, the Euclidean distance is computed between this instance and every instance in the training set:

   $$d(\text{test\_instance}, \text{training\_instance}_i) = \sqrt{\sum_{j=1}^{n} (\text{test\_instance}[j] - \text{training\_instance}_i[j])^2}$$

   - The distances are sorted in ascending order, and the `k` smallest distances (i.e., the closest instances) are selected as the neighbors:

   $$\text{neighbors} = \text{argmin}_k \; d(\text{test\_instance}, \text{training\_instances})$$

4. **Function:** `getResponse`
   This function determines the majority class (up or down) among the neighbors.

   - A voting mechanism is employed where each neighbor "votes" for its class (up or down).

- The class with the highest number of votes is chosen as the predicted class:

$$\text{predicted\_class} = \text{argmax}_c \sum_{i=1}^{k} 1(\text{neighbor}_i = c)$$

where $1(\cdot)$ is the indicator function, which is 1 if the neighbor belongs to class c (either "up" or "down") and 0 otherwise. The function effectively counts the votes for each class and selects the one with the most votes.

5. **Function:** `getAccuracy`
   This function calculates the accuracy of the predictions.

   - The accuracy is defined as the ratio of the number of correct predictions to the total number of predictions made:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$$

   where a "correct prediction" occurs when the predicted class matches the actual class.

6. **Function:** `change`
   This function determines whether the stock price went up or down.

   - It compares the stock price on the current day (`price_today`) with the previous day (`price_yesterday`) and returns either "up" or "down":

$$\text{change} = \begin{cases} \text{up} & \text{if price}_{\text{today}} > \text{price}_{\text{yesterday}} \\ \text{down} & \text{otherwise} \end{cases}$$

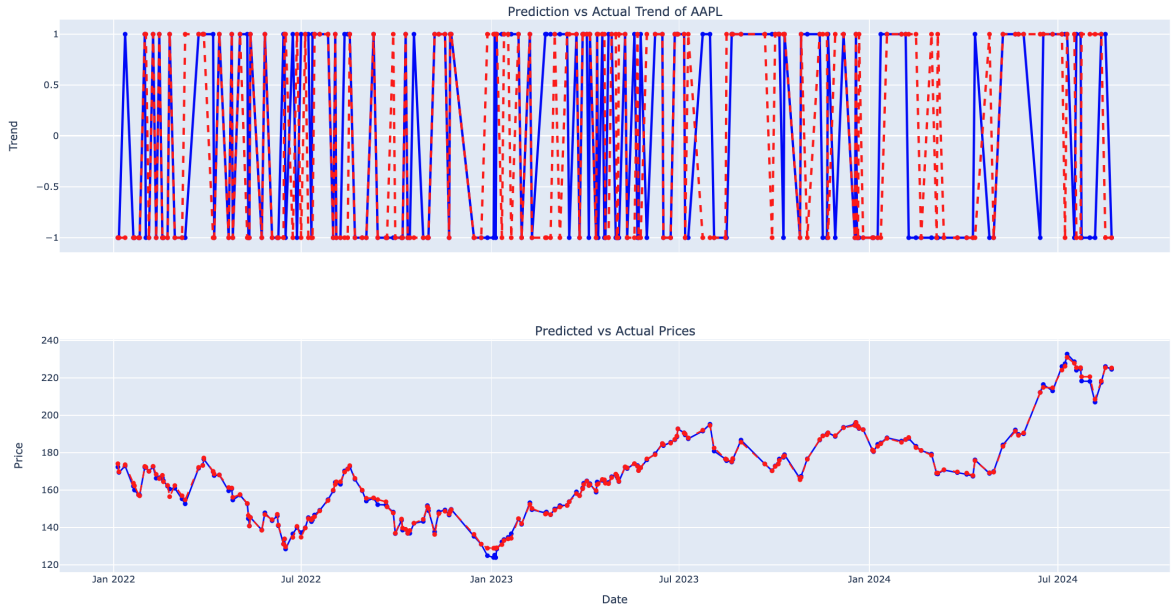
   This function is crucial for labeling the stock price movements, which are later used as the target classes in the k-NN algorithm.

7. **Function:** `predict_and_get_accuracy`
   This function makes predictions and plots the results.

   - For each test instance, the trend (up or down) is predicted based on the majority class of the nearest neighbors, as explained in the `getResponse` function.
   - The predicted price is computed as the average of the closing prices of the nearest neighbors:

$$\text{Predicted Price} = \frac{1}{k} \sum_{i=1}^{k} \text{neighbor}_i[\text{close price}]$$

   - The accuracy of the trend prediction is calculated using the method explained in the `getAccuracy` function.
   - Three subplots are generated:
     - **Stock Prices:** A plot of the actual stock prices over time.
     - **Trends:** A plot comparing the predicted vs actual trends (up or down).
     - **Prices:** A plot comparing the predicted vs actual stock prices.
   - The plots are displayed using the `plotly` library, which allows for interactive exploration of the results.



Results and Discussion

The k-Nearest Neighbors (k-NN) algorithm achieved a 69.52% accuracy in predicting AAPL stock trends using 661 data points (451 for training, 210 for testing). While the model captured some patterns, its reliance on historical data and sensitivity to the choice of k (here $k = 5$) limits its effectiveness, especially in volatile markets.

Despite this moderate success, several limitations are evident:

1. **Scalability:** k-NN is computationally expensive and memory-intensive, which could be problematic for larger datasets.
2. **Dimensionality:** The algorithm struggles as the number of features increases, potentially leading to reduced accuracy.
3. **Sensitivity to Noise:** k-NN is prone to inaccuracies in volatile markets due to its sensitivity to noise and outliers.
4. **Temporal Dependency:** The model does not account for the sequential nature of stock prices, which limits its ability to capture evolving trends.
5. **Overfitting:** With an accuracy of 69.52%, there is a risk of overfitting, particularly with a smaller k, leading to poor performance on new data.

Conclusion

While k-NN achieved a moderate accuracy of 69.52% in predicting AAPL stock trends, its limitations—sensitivity to noise, lack of temporal modeling, and potential for overfitting—suggest that it is better suited as a baseline method. More advanced models that consider market dynamics and temporal dependencies are needed for higher accuracy and reliability in stock market predictions.

3.2 KNN - stock outliers prediction

In this study, we analyze the stock prices of LVMH (Louis Vuitton Moët Hennessy) over the period from January 1, 2021, to December 31, 2023. The primary objective is to detect outliers in the adjusted closing prices using the k-Nearest Neighbors (k-NN) algorithm. Outliers in stock prices may indicate unusual market events, data errors, or other anomalies that deviate from the expected behavior.

Methodology

The steps involved in detecting outliers are as follows:

1. **Data Preprocessing:** The adjusted closing prices were standardized using `StandardScaler` from `scikit-learn`. Standardization transforms the data to have a mean of 0 and a standard deviation of 1, ensuring that the k-NN algorithm operates on data with a uniform scale:

$$z = \frac{x - \mu}{\sigma}$$

   where z is the standardized value, x is the original value, μ is the mean of the dataset, and σ is the standard deviation.

2. **k-NN Model Training:** A k-NN model was trained using the standardized prices. We chose $k = 5$, meaning that each data point is compared with its five nearest neighbors to assess its similarity. The Euclidean distance d between two points x and y in an n-dimensional space is calculated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

   This distance metric is used to determine the "closeness" of data points.

3. **Outlier Detection:** Outliers were identified by calculating the mean distance of each point to its five nearest neighbors. A point was classified as an outlier if its mean distance exceeded a threshold defined as the mean of all distances plus two standard deviations:

$$\text{Threshold} = \mu_d + 2\sigma_d$$

   where μ_d is the mean of the distances and σ_d is the standard deviation of the distances. Points where the mean distance exceeded this threshold were considered outliers.

Results

The k-NN model detected several outliers in the LVMH stock prices during the analyzed period. These outliers are represented as red markers on the price chart, indicating days where the stock price deviated significantly from its typical behavior as compared to neighboring days.

The interactive plot provides a detailed visualization of the stock prices and outliers. By hovering over the points, one can inspect specific dates and prices where anomalies were detected.


```
Number of outliers: 7
Outliers:
Date: 2022-05-20 00:00:00 | Price: 539.0977783203125
Date: 2022-06-16 00:00:00 | Price: 522.4386596679688
Date: 2023-01-11 00:00:00 | Price: 753.1224365234375
Date: 2023-04-24 00:00:00 | Price: 879.6017456054688
Date: 2023-06-30 00:00:00 | Price: 848.15234375
Date: 2023-09-05 00:00:00 | Price: 746.0398559570312
Date: 2023-09-15 00:00:00 | Price: 748.103759765625
```

While zooming-in and using the candlestick, we could have a more precis interpretation of outliers, and we could confirm these outliers detected based on the trend and also the situation.

Discussion

Outlier detection in stock prices is a crucial task for understanding market behavior and identifying potential errors or unusual events. The k-NN method is effective for this purpose because it directly compares each data point to its neighbors, making it sensitive to deviations from the norm.

The effectiveness of k-NN in detecting outliers relies heavily on the choice of parameters. The value of k determines the number of neighbors considered, affecting the sensitivity of the model to local variations in the data. A smaller k might detect more localized anomalies, while a larger k smooths out these variations, potentially missing subtle outliers. Similarly, the threshold, determined by the distance metric, affects the model's ability to distinguish between normal fluctuations and true outliers.

Conclusion

The k-NN algorithm successfully identified outliers in the LVMH stock prices from 2021 to 2023. These outliers can be further analyzed to understand their causes, which may include market anomalies, significant economic events, or data issues. This approach provides a robust method for detecting anomalies in financial data, with potential applications in risk management and automated trading systems.

The incorporation of mathematical techniques, such as standardization and Euclidean distance, enhances the precision of the k-NN model in detecting outliers. Future work may explore optimizing the choice of k and the outlier detection threshold to improve accuracy and applicability across different financial datasets.

3.3 KNN - stock volatility prediction

In this study, we aim to predict and forecast the volatility of stocks price using historical data and a machine learning approach. Volatility, a measure of risk, is calculated as the standard deviation of stock returns over a rolling window. Predicting volatility is of high importance to both traders and risk managers. In this paper, we use the K-Nearest Neighbors (KNN) algorithm to predict future volatility and compare it to the actual observed volatility in the stock market. We also forecast the future volatility for the next 10 days beyond the available data.

Methodology

1. Data Acquisition We use historical stock price data for LVMH (MC.PA) from Yahoo Finance. The data spans from January 1, 2023, to August 5, 2024, with an additional 10 days of data for comparison with predicted and forecasted volatilities. The data is downloaded using the `yfinance` library, and the relevant stock attributes are processed, including adjusted closing price, daily returns, and volatility.

   The volatility (σ) is computed as the rolling standard deviation of daily returns over a window of 21 days:

   $$\sigma_t = \sqrt{\frac{1}{21}\sum_{i=1}^{21}(r_i - \bar{r})^2}$$

   where r_i is the daily return and $\bar{r}$ is the mean return over the window.

2. Feature Engineering To predict future volatility, we employ lagged values of volatility and returns as input features. Specifically:

   $$X = \left[\text{Lagged\_Volatility}_t, \quad \text{Lagged\_Return}_t\right]$$

   The target variable is the future volatility (σ_t), and the model attempts to learn the relationship between the lagged features and the future volatility.

3. Model Training We split the data into training and test sets in an 80:20 ratio. The KNN algorithm is used to predict volatility based on the training set. The model is trained using 5 neighbors ($k = 5$), where the volatility is predicted as the mean value of the 5 nearest neighbors.

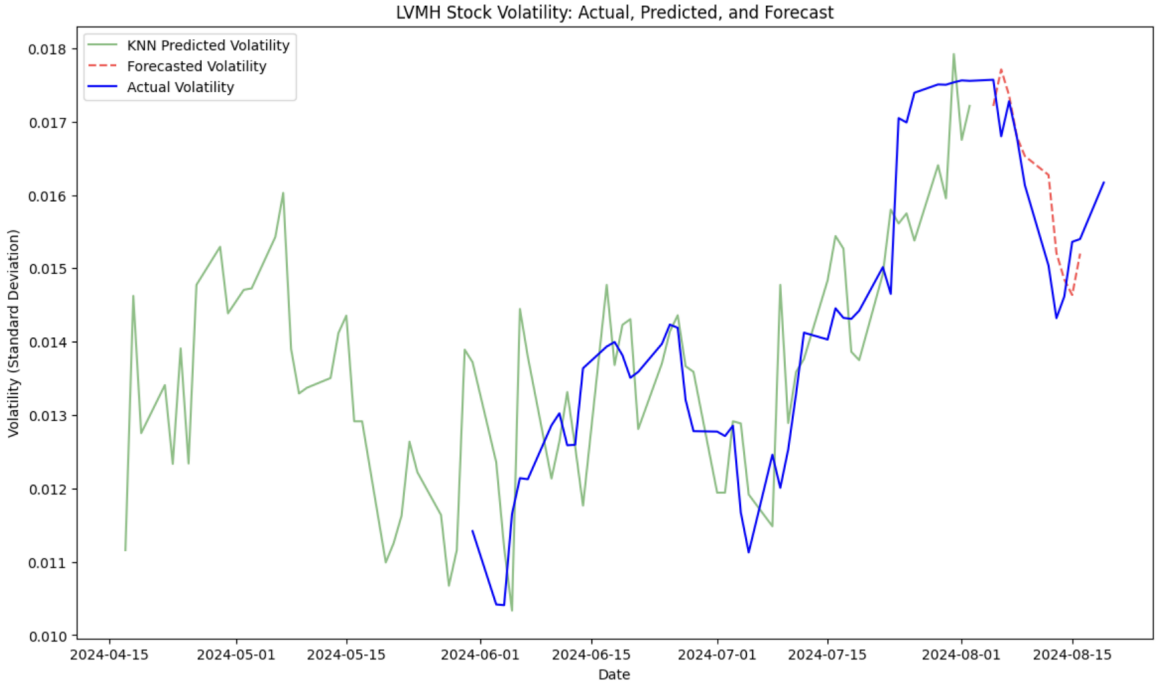
   The Mean Squared Error (MSE) is calculated to evaluate the model's performance:

   $$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{\sigma}_i - \sigma_i)^2$$

   where $\hat{\sigma}_i$ is the predicted volatility, and σ_i is the actual volatility.

4. Forecasting We forecast volatility for the next 10 days by using the most recent lagged values from the test set. At each step, the predicted volatility is fed back into the model to forecast the next day's volatility. Additionally, a small random return is simulated to represent market fluctuations.

Results



- Actual volatility, calculated from true stock prices.
- Predicted volatility using the KNN model on the test set.
- Forecasted volatility for the next 10 business days.

The green curve represents the predicted volatility using KNN, the red dashed line shows the forecasted volatility for the upcoming 10 days, and the blue line is the actual volatility calculated from the stock price data.

Discussion

The KNN model effectively captures the general trend of volatility in the stock market but shows some limitations in accurately predicting sharp changes in volatility, as indicated by the difference between the actual and predicted values. This is due to the lagged nature of the features, which may not fully capture sudden market shifts. Despite this, the model performs reasonably well when the market is stable, as reflected by a relatively low Mean Squared Error (MSE).

The forecasted volatility shows a stable trend, which is expected given the recent historical data. However, the forecast assumes that future volatility will behave similarly to past trends, which might not hold during periods of market instability.

Future improvements could involve experimenting with more complex models, such as ARIMA or LSTM, to capture both trend and seasonality in the volatility series. Additionally, increasing the number of features, such as volume or market indices, may provide more predictive power to the model.

Conclusion

In conclusion, the KNN model provides a good approximation of LVMH's stock volatility, with the ability to forecast future volatility based on historical data. The method is simple to implement but could benefit from enhancements to improve prediction accuracy and to better account for abrupt market changes.

4 Machine Learning – DBSCAN

4.1 DBSCAN - Outliers detection

Introduction

In this paper, we apply the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm to the stock price data of Apple Inc. (AAPL) in order to detect anomalies and outliers in its price movements. DBSCAN is well-suited for detecting outliers, as it identifies regions of high data density and marks points that lie outside of these regions as noise. This makes it particularly useful for spotting unusual price movements that may indicate potential trading signals or abnormal market behavior.

Methodology

Data Acquisition

We collected historical stock price data for AAPL from January 1, 2023, to August 5, 2024, using the `yfinance` Python library. The data includes adjusted closing prices, which were then used to calculate daily returns.

$$\text{Return}_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

where P_t represents the adjusted closing price at time t.

We also calculated the rolling volatility using a 20-day rolling window, which gives us a measure of risk over time. The volatility σ_t was computed as:

$$\sigma_t = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \bar{r})^2}$$

where r_i represents the daily returns over a 20-day window.

DBSCAN Clustering for Anomaly Detection

The feature set used for DBSCAN consists of daily returns and rolling volatility:

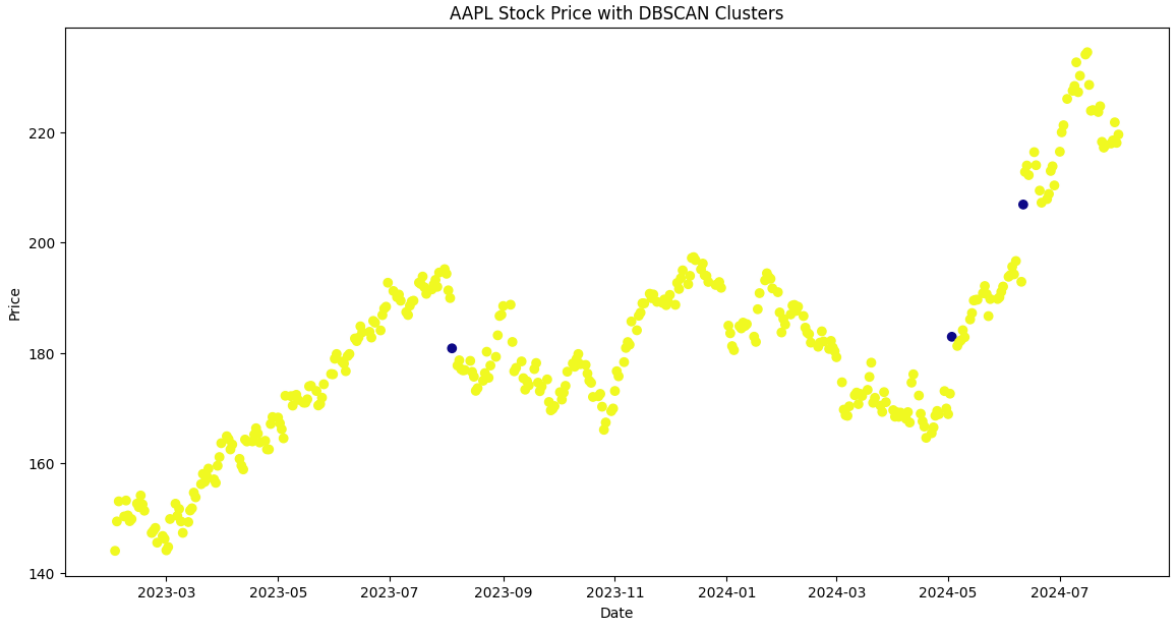
$$X = \begin{bmatrix}\text{Return}_t & \sigma_t\end{bmatrix}$$

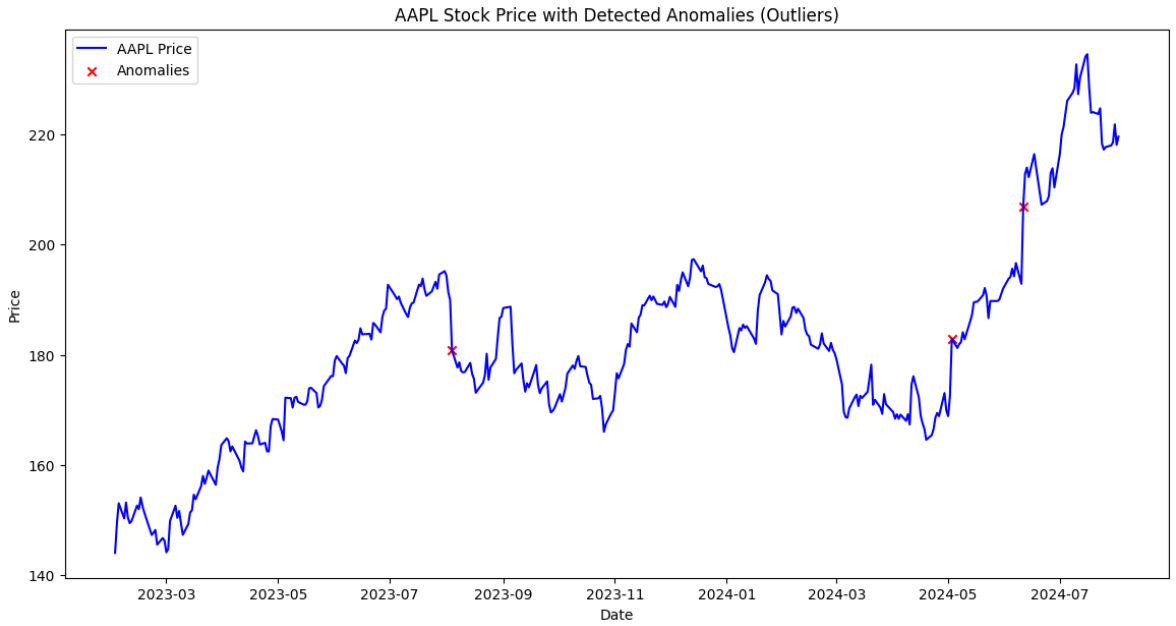
We applied DBSCAN to this dataset with the parameters $\epsilon = 0.01$ and `min_samples` $= 5$. These parameters control the neighborhood radius for identifying clusters and the minimum number of samples required for a point to be considered a core point.

Points classified as noise (assigned a label of -1 by DBSCAN) are considered anomalies in this study.

Visualization

Clustering Results





Discussion

The DBSCAN algorithm successfully identified several anomalies in the stock price data of AAPL over the analyzed period. These anomalies could represent sudden price spikes or drops, potentially offering valuable trading signals. By focusing on clusters of normal price behavior, DBSCAN is able to isolate periods of abnormal activity, which could be used as part of a broader trading strategy.

However, one limitation of this approach is the sensitivity of DBSCAN to the choice of hyperparameters ϵ and `min_samples`. Small changes to these values can lead to different clustering outcomes, which may affect the consistency of anomaly detection. Therefore, careful tuning of these parameters is essential for robust results.

Additionally, while DBSCAN can detect anomalies, it does not provide specific reasons for why a particular data point is classified as an anomaly. In future work, it would be beneficial to investigate the underlying causes of these anomalies, such as macroeconomic factors or company-specific news.

4.2 DBSCAN - clustering with Technical Analysis

The principle behind this strategy relies on a combination of clustering algorithms (DBSCAN : Density-Based Spatial Clustering of Applications with Noise) and technical indicators (RSI, MACD) to identify buy or sell signals. Here's a mathematical explanation of the different components involved in this strategy:

DBSCAN defines two types of points:

- **Core point**: A point is a core point if there are at least min_samples points within a distance ϵ of this point.
- **Border point**: A point that is not a core point but is within a distance ϵ of a core point.

Mathematically, this algorithm helps identify clusters based on the proximity between the technical indicators (RSI, MACD, and moving average).

RSI (Relative Strength Index)

RSI is an oscillator that measures the speed and change of price movements. It is calculated as follows:

$$RSI = 100 - \frac{100}{1 + RS}$$

where

$$RS = \frac{\text{Average of gains over N days}}{\text{Average of losses over N days}}$$

The RSI ranges from 0 to 100. Typically, an RSI above 70 indicates overbought conditions, while an RSI below 30 indicates oversold conditions.

MACD (Moving Average Convergence Divergence)

The MACD is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price:

$$MACD = EMA_{\text{short term}} - EMA_{\text{long term}}$$

Where:

$$EMA_{\text{short term}} = \frac{P_t \cdot \left(\frac{2}{n+1}\right) + EMA_{\text{previous}} \cdot \left(1 - \frac{2}{n+1}\right)}{n}$$

The signal line is another EMA applied to the MACD, usually calculated over 9 days:

$$\text{Signal Line} = EMA(MACD, 9)$$

A bullish crossover (MACD above the signal line) generates a buy signal, and a bearish crossover (MACD below the signal line) generates a sell signal.

Trend Detection with Clusters

We define a buy or sell signal based on the price trend within a given cluster identified by the DBSCAN algorithm. If 5 consecutive points belong to the same cluster and the price trend is upward, a buy signal is generated. Conversely, if the price trend is downward, a sell signal is generated.

Mathematically, for each point i belonging to the same cluster over 5 periods:

- **Upward trend**:

$$\text{Buy Signal} \quad \text{if} \quad P_i > P_{i-4}$$

- **Downward trend**:

$$\text{Sell Signal} \quad \text{if} \quad P_i < P_{i-4}$$

Summary of Operation

- **DBSCAN** groups technical points based on density.
- **RSI** and **MACD** provide insights into relative strength and trend.
- If 5 points belong to the same cluster and prices are rising, a **buy signal** is generated.
- If 5 points belong to the same cluster and prices are falling, a **sell signal** is generated.

These combined approaches provide a comprehensive decision-making framework based on both mathematical signals and the data structure identified through clustering.

Backtest

Discussion

This strategy, combining DBSCAN clustering with RSI and MACD, offers multiple avenues for discussion. The choice of DBSCAN, which doesn't require predefined clusters, and its parameter tuning, is key for handling noisy financial data. RSI helps detect overbought/oversold conditions, while MACD identifies trend reversals, though both have limitations like lagging signals. Combining these with clustering enhances signal accuracy, but the strategy's effectiveness varies across different market regimes, and it's prone to false signals in volatile conditions. Discussions could also focus on computational efficiency, backtesting approaches, performance metrics like Sharpe ratio, and real-world issues such as liquidity, transaction costs, and slippage. Risk management through stop-losses, take-profits, and position sizing is crucial, as is addressing potential pitfalls like overfitting and the strategy's response to extreme market events. Additionally, incorporating other indicators or machine learning models could refine the approach further.

Conclusion

In this study, we applied DBSCAN to AAPL stock price data to detect anomalies. The algorithm was able to classify periods of normal price behavior into clusters and label unusual price movements as anomalies. These anomalies could serve as useful signals for traders looking to exploit sudden price changes. Future improvements could involve tuning the DBSCAN parameters and incorporating additional features such as volume, news sentiment, or external market indicators.

4.3 DBSCAN - clustering using correlation matrix

In financial markets, assets often exhibit similar behaviors, making them candidates for strategies like pair trading. By clustering assets based on their historical price data, we can identify groups of assets that move in a correlated manner. This paper explains how we can use DBSCAN (Density-Based Spatial Clustering of Applications with Noise) along with Principal Component Analysis (PCA) to cluster assets and visualize their relationships.

DBSCAN Clustering

The **DBSCAN** algorithm is a density-based clustering method. It groups data points into clusters based on two parameters: the neighborhood radius ϵ and the minimum number of points required to form a cluster, min_samples. Unlike algorithms like K-means, DBSCAN does not require the number of clusters to be specified in advance. The steps for applying DBSCAN are:

- Calculate the pairwise distances between data points.
- Identify core points (points with at least min_samples points within their ϵ-neighborhood).
- Form clusters by connecting core points and border points that are within the ϵ-distance.
- Points that are not assigned to any cluster are classified as outliers (labelled as -1).

For clustering financial assets, we use a distance metric based on correlations between assets. The correlation matrix $\mathbf{C}$ is computed from the daily returns of the assets:

$$C_{ij} = \frac{\text{cov}(r_i, r_j)}{\sigma_i \sigma_j}$$

where r_i and r_j are the returns of assets i and j, and σ_i and σ_j are the standard deviations of the returns. To apply DBSCAN, we convert the correlation into a distance metric:

$$\text{Distance} = 1 - \text{Correlation}$$

A small distance between two assets indicates a high correlation.

Principal Component Analysis (PCA)

To visualize the clusters identified by DBSCAN, we use **Principal Component Analysis (PCA)** to reduce the dimensionality of the correlation matrix. PCA is a technique that transforms a set of correlated variables into a set of uncorrelated variables, known as principal components, which capture the most variance in the data. Mathematically, PCA involves:

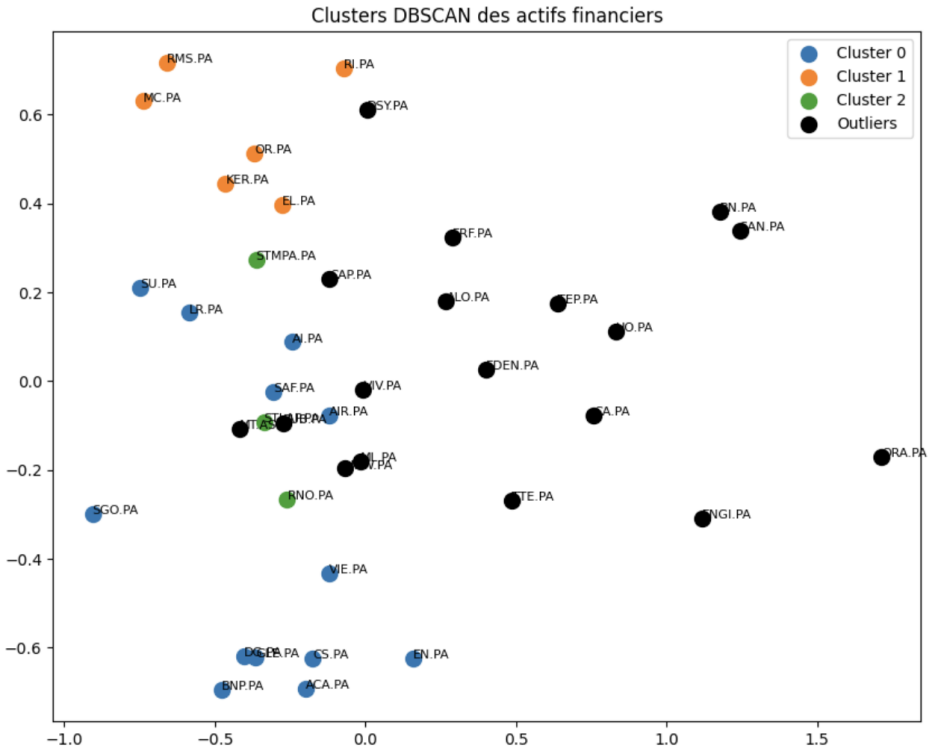
- Computing the covariance matrix $\mathbf{\Sigma}$ of the data.
- Finding the eigenvalues and eigenvectors of $\mathbf{\Sigma}$.
- Projecting the data onto the eigenvectors (principal components) corresponding to the largest eigenvalues.

In our case, we reduce the dimensionality to two or three components to create a 2D or 3D scatter plot for visualization. The assets are then plotted in this reduced space, and points are colored according to their cluster labels from DBSCAN.

Clustering and Visualization Procedure

The steps for finding and visualizing clusters are as follows:

1. **Download historical price data:** We download adjusted closing prices of the assets from Yahoo Finance for a specified period.
2. **Compute daily returns:** The daily returns r_t are computed as the percentage change in price from day t to day $t+1$:
$$r_t = \frac{P_{t+1} - P_t}{P_t}$$
where P_t is the adjusted closing price at time t.
3. **Calculate correlation matrix:** We compute the correlation matrix $\mathbf{C}$ from the returns data to understand the relationships between assets.
4. **Apply DBSCAN:** DBSCAN is applied to the distance matrix derived from the correlation matrix to identify clusters of highly correlated assets.
5. **Dimensionality reduction with PCA:** PCA is used to reduce the dimensionality of the correlation data for visualization.
6. **Plot clusters:** The assets are plotted in 2D (or 3D) space, with different colors representing different clusters. Outliers (assets that don't belong to any cluster) are marked separately.



By using DBSCAN and PCA, we can cluster assets based on their correlation structures and visualize their relationships. This technique can be particularly useful in portfolio management and pair trading strategies, where identifying assets that move together is key. DBSCAN's ability to handle arbitrary-shaped clusters and its robustness to outliers make it an ideal choice for financial market analysis.

5 Machine Learning – Random Forest

5.1 Random Forest - Buy and sell signals prediction

The goal of this process is to predict stock price movements using a machine learning model, specifically Random Forest, and to enrich the predictive model with commonly used technical indicators such as the Moving Average Convergence Divergence (MACD) and Bollinger Bands. These indicators help to capture the market trends and volatility, making the predictions more robust.

Data Collection and Preprocessing

We first retrieve historical stock data using the `yfinance` API, which provides daily prices for a specified stock. The data used includes the adjusted close price (`Adj Close`) of the stock. From this, we calculate daily returns:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

where r_t is the daily return at time t, and P_t and P_{t-1} are the adjusted close prices at time t and $t-1$, respectively.

Technical Indicators

We use the following technical indicators as features for the Random Forest model:

Simple Moving Average (SMA)

The Simple Moving Average (SMA) is calculated as the average of the adjusted closing price over a specified window. We compute two SMAs:

$$\text{SMA}_{20} = \frac{1}{20} \sum_{i=0}^{19} P_{t-i}$$

$$\text{SMA}_{50} = \frac{1}{50} \sum_{i=0}^{49} P_{t-i}$$

Where P_t is the adjusted close price at time t.

Relative Strength Index (RSI)

The RSI is an oscillator that measures the speed and change of price movements. It is calculated as:

$$\text{RSI} = 100 - \left(\frac{100}{1 + \frac{\text{Average Gain over 14 days}}{\text{Average Loss over 14 days}}} \right)$$

Moving Average Convergence Divergence (MACD)

The MACD is a trend-following momentum indicator that shows the relationship between two moving averages of the stock's price. It is computed as the difference between a short-term (12-day) exponential moving average (EMA) and a long-term (26-day) EMA:

$$\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26}$$

We also compute a signal line, which is the 9-day EMA of the MACD:

$$\text{Signal Line} = \text{EMA}_9(\text{MACD})$$

Bollinger Bands

Bollinger Bands consist of a simple moving average and two bands plotted at a distance of two standard deviations from the SMA. The upper and lower Bollinger Bands are computed as follows:

$$\text{Upper Band} = \text{SMA}_{20} + 2 \cdot \sigma_{20}$$

$$\text{Lower Band} = \text{SMA}_{20} - 2 \cdot \sigma_{20}$$

where σ_{20} is the rolling 20-day standard deviation of the stock's price.

Random Forest Model

After calculating these indicators, we use them as features to train the Random Forest model. The target variable is binary, representing whether the stock price will increase (1) or decrease (0) the next day. The dataset is split into training and testing sets, and the model is trained using the training data.

Random Forest is an ensemble method that builds multiple decision trees and combines their predictions. Each decision tree is built by selecting random samples and random subsets of features. The final prediction is the majority vote (for classification) or the average (for regression) of the individual trees' predictions.

The features used in the model are:

- SMA_20
- SMA_50
- RSI
- Returns
- MACD
- MACD Signal Line
- Bollinger Upper Band
- Bollinger Lower Band

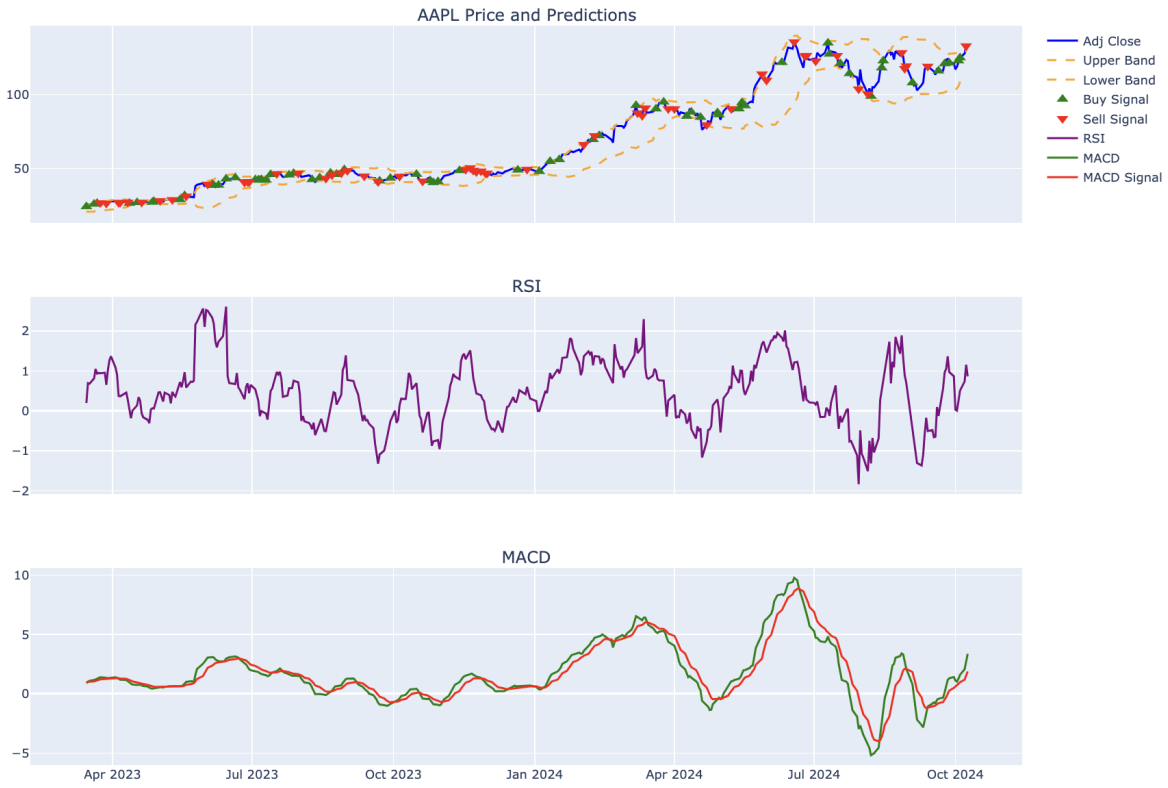
Discussion

However, despite the integration of technical indicators like MACD and Bollinger Bands into the Random Forest model, this approach has several limitations. Firstly, the model is highly dependent on historical data, assuming that past price movements and patterns will persist in the future, which may not always hold true in rapidly changing market conditions. Moreover, the choice of technical indicators and their parameter settings can significantly impact the model's performance, and these parameters may require frequent recalibration to adapt to different market regimes. Additionally, Random Forest, being an ensemble method, may sometimes lack interpretability, making it challenging to understand the rationale behind specific predictions. Another critical consideration is that this strategy does not account for macroeconomic factors, news events, or external shocks, which can greatly influence market behavior. Lastly, overfitting remains a risk, especially if the model is trained on a small dataset or too many features are used, leading to diminished generalization in live market environments. Therefore, continuous evaluation, backtesting, and refinement are essential to ensure robustness and avoid pitfalls in real-world trading.

Backtest

We visualize the stock price movements and trading signals using an interactive plot. The plot includes:

- **Adjusted Close Price**: Displayed as a blue line.
- **Bollinger Bands**: The upper and lower bands are plotted as dashed lines.
- **Buy and Sell Signals**: Marked as green and red triangles based on the model's predictions.
- **RSI and MACD**: Plotted as subplots to observe momentum indicators.



5.2 Random Forest - stock choosing

The algorithm presented aims to select the best stocks from the CAC 40 index based on several commonly used technical indicators in trading, such as Simple Moving Averages (SMA), the Relative Strength Index (RSI), the Moving Average Convergence Divergence (MACD), and Bollinger Bands. The goal is to assign a score to each stock based on these indicators and select those with the highest potential.

Main Steps of the Algorithm

1. **Downloading financial data**: Historical stock data from the CAC 40 is retrieved using the `yfinance` library. The data includes the adjusted closing prices (*Adj Close*) for the period from January 2023 to January 2024.

2. **Calculation of technical indicators**: For each stock, the following technical indicators are calculated:

   - **SMA 50 and SMA 200**: The 50-day and 200-day simple moving averages are calculated to detect medium- and long-term trends.

   $$SMA_{50}(t) = \frac{1}{50} \sum_{i=t-50}^{t} P_{\text{close}}(i), \quad SMA_{200}(t) = \frac{1}{200} \sum_{i=t-200}^{t} P_{\text{close}}(i)$$

   where $P_{\text{close}}(i)$ represents the adjusted closing price at time i.

   - **RSI (14 days)**: The RSI is calculated to measure the strength or weakness of a stock over a given period, typically 14 days.

   $$RSI = 100 - \left(\frac{100}{1 + \frac{\text{average gain over 14 days}}{\text{average loss over 14 days}}} \right)$$

   - **MACD**: The MACD is obtained by subtracting a 26-day exponential moving average (EMA) from a 12-day EMA. A signal line is also calculated by applying a 9-day EMA to the MACD.

   $$\text{MACD}(t) = EMA_{12}(t) - EMA_{26}(t)$$

   - **Bollinger Bands**: Bollinger Bands are calculated around a 20-day SMA, with a standard deviation multiplier (σ) of 2. They measure the volatility of prices.

   $$\text{Bollinger Upper} = SMA_{20} + 2\sigma, \quad \text{Bollinger Lower} = SMA_{20} - 2\sigma$$

3. **Assigning scores**: A score is assigned to each stock based on the following conditions:

   - If the adjusted closing price is above the SMA 50 or SMA 200, a point is awarded.
   - If the RSI is between 30 and 70 (optimal trading zone), a point is awarded.
   - If the MACD is above the signal line (bullish indicator), a point is awarded.
   - If the stock price is near the upper Bollinger Band (indicating an upward trend), a point is awarded.

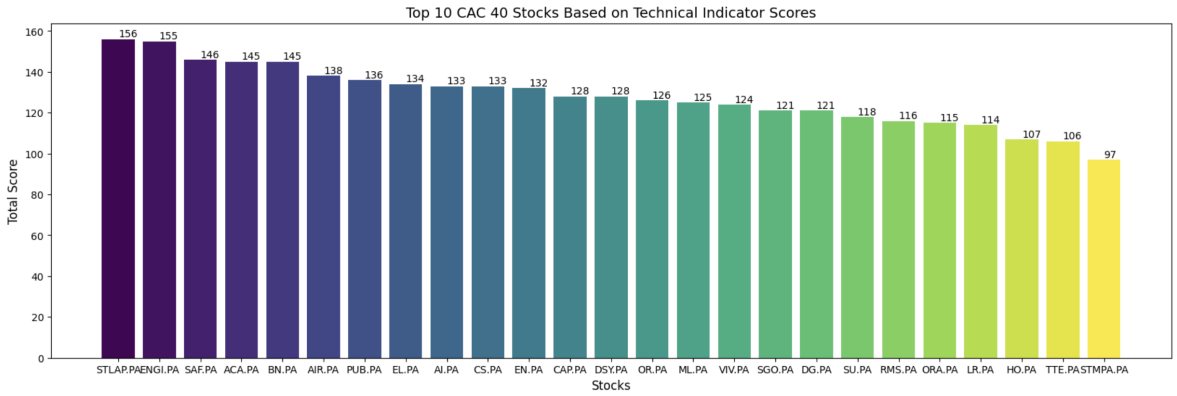
   The total score for a stock is the sum of these points.

   $$\text{Total Score} = \text{SMA Score} + \text{RSI Score} + \text{MACD Score} + \text{Bollinger Score}$$

4. **Selecting the best stocks**: Stocks are ranked based on their total score, and the top 25 are selected.

5. **Graphical visualization**: A bar chart is generated to visualize the scores of the selected stocks. The stocks are represented on the x-axis and the scores on the y-axis. Each bar corresponds to a stock, and its height represents its total score.

Visualization



Conclusion

This process efficiently filters and visualizes CAC 40 stocks based on several technical indicators. The model assigns a score based on each stock's performance relative to the indicators, making it easier to select the best stocks for a portfolio. This method can be further refined by adding additional indicators or adjusting the weightings to fit specific trading strategies.