Crypto Trading Analysis

Alan Matthew

2025-04-16

Contents

Executive Summary

This study investigates trade direction and market impact in a live cryptocurrency strategy. While trade direction is moderately predictable, the analysis finds no strong evidence of market impact. Instead, it reveals a behavioral pattern, particularly in Sell timing, that could be refined to improve the strategy's execution and decision-making.

Background and Motivation

This project examines how a cryptocurrency trading strategy decides when to buy or sell, and whether those decisions influence market prices. Using detailed trading (`fills_data`) and market (`market_data`) data, the study aims to identify what drives these decisions and their broader market impact.

The analysis focuses on two research questions:

1. Trading behavior: Can I predict whether the next trade will be a Buy or Sell? I approach this as a classification problem, modeling trade direction and examining which features most influence the strategy's decisions.
2. Market Impact: Do these trades affect prices? Using permutation tests, I assess whether trades, especially those that provide liquidity, lead to meaningful price changes.

This project contributes to ongoing efforts to better understand how algorithmic trading strategies work and their effects on financial markets. Informed by my internship experience, I apply a structured, data-driven approach to examine decision drivers and potential market influence.

Key Findings

1. Buy and Sell trades were found to be moderately predictable using engineered features. While logistic regression was the most accurate model (66.3 percent accuracy, 66.8 percent balanced accuracy, Kappa 0.33), it showed a clear bias toward predicting Sell trades (classification bias: 0.318, F1 score: 0.609). Gradient boosting performed slightly worse in accuracy (65.4 percent) but achieved the highest F1 score (0.669) and the lowest classification bias (0.047), making it the most balanced model overall. Across all models, `deviation_from_mean_balance_lag1_diff` consistently ranked as the most important feature. Tree-based models also highlighted `trade_pnl_lag1`, `fill_qty`, and `volatility_lag1` as key predictors, while downweighting price features like `fill_prc` and `deviation_from_mean_mid_price_lag1`. SVM and KNN emphasized similar signals but underperformed in accuracy. These results suggest that trade direction is better explained by features related to recent trade profitability, inventory changes, market indicators, and order size than by short-term price signals.
2. Maker trades are followed by significant upward price movement (mean = 0.525, $p < 0.001$), while Taker trades show no effect (mean = –0.406, $p = 0.131$). This is driven entirely by Maker Sell trades (mean = 0.800, $p < 0.001$); Maker Buy trades are not significant ($p = 0.080$). The price moves against the trade, not with it, indicating no market impact. Instead, the results suggest poor timing on Maker Sell trades, where the strategy exits just before prices rise.

Contribution to the Discussion

This study contributes to ongoing discussions on algorithmic trading by showing that:

1. Feature engineering had a greater impact than model choice in predicting trade direction, highlighting the importance of domain-specific inputs and supporting a shift toward data-driven approaches guided by market insight and domain knowledge.
2. This study extends the discussion on algorithmic trading fairness by showing that neither Maker nor Taker trades exhibit evidence of market impact. Instead, the observed price movements following Maker trades reflect poor Sell timing. This shifts the focus away from fairness concerns toward opportunities for improving execution and decision-making logic.

Call to Action

This study shows that the strategy behavior is only moderately predictable. Furthermore, while the results show that the trading strategy does not exhibit market impact, Maker Sell trades often occur just before prices rise, suggesting issues with execution timing rather than price influence.

Considering these findings, two recommendations emerge for improving the strategy:

1. Incorporate richer market context into the dataset: Practitioners should consider expanding the dataset with additional external signals, such as market indexes, more detailed order book data, or macroeconomic indicators. Collecting more detailed order book, trade, and market data will support the development of more expressive features and enable more accurate modeling of the strategy's behavior.
2. Focus on execution quality, especially for Maker Sell decisions: The analysis highlights that Maker Sell trades are often poorly timed, with prices rising shortly after the strategy exits. This suggests that the

logic used to determine when to exit positions requires closer examination. Enhancing execution timing could lead to better responses to market conditions and greater profitability.

Limitations

- This study is based on a limited set of features. Without richer market context such as order book depth or external signals, the analysis may not fully capture the factors behind trade decisions.
- The analysis focuses on a single strategy operating in one market. Results may not generalize to other conditions. Comparing this strategy across multiple markets could offer further insight or reveal consistent patterns.

Future Work

- Expanding the dataset to include order book snapshots, queue position, and broader market indicators would support more effective feature engineering. This could uncover deeper patterns behind trade behavior.
- Future work could explore models that better capture dynamic, non-linear, and temporal effects, such as attention-based networks or sequence models. These approaches may provide deeper insights into the decision-making process behind trade direction by accounting for how recent market history influences strategy behavior.
- ICE (Individual Conditional Expectation) and PDP (Partial Dependence Plot) methods can help interpret how each feature influences model predictions. Using these tools across all predictors verifies expected behavior and strengthens the interpretability of the models.
- Given that Maker Sell trades are followed by consistent price increases, future work should develop tools to evaluate and monitor the timing of trade decisions. Rather than market impact, the results suggest poor Sell-side timing. Developing diagnostic tools to flag missed opportunities could help refine the strategy and reduce inefficiencies.
- Evaluate the strategy across multiple markets to assess generalizability and identify consistent behavioral patterns beyond the current single-market analysis.

Exploratory Data Analysis

I will begin my investigation by loading the dataset and inspecting it for missing values. Then, I will lay the groundwork and compute two important features for our subsequent modelling and analysis: mid price and profit and loss (PnL).

Data Inspection

I first convert some features in `fills_data` and `market_data` to factors to support proper modeling and analysis.

```
## --- Summary of fills_data ---

## tibble [1,123 x 13] (S3: tbl_df/tbl/data.frame)
##  $ order_id        :integer64 [1:1123] 670003026938216 670003026940777 670003026941465 6700030269416
##  $ side            : Factor w/ 2 levels "B","S": 2 1 1 1 2 1 2 2 1 1 ...
##  $ fill_prc        : num [1:1123] 1938 1921 1914 1917 1921 ...
##  $ fill_qty        : num [1:1123] 0.069 0.0264 0.0707 0.1719 0.1719 ...
##  $ liquidity       : Factor w/ 2 levels "Maker","Taker": 1 1 1 2 1 1 1 1 1 2 ...
##  $ fee             : num [1:1123] 0 0 0 0.000305 0 ...
##  $ fee_ccy         : Factor w/ 1 level "bnb": 1 1 1 1 1 1 1 1 1 1 ...
##  $ fee_ccy_usd_rate: num [1:1123] 237 237 237 237 237 ...
##  $ fill_id         :integer64 [1:1123] 1688172365615000000 1688194379383000000 1688203192806000000 16
```

```
##  $ symbol          : Factor w/ 1 level "eth_usdc": 1 1 1 1 1 1 1 1 1 1 ...
##  $ exch            : Factor w/ 1 level "binance": 1 1 1 1 1 1 1 1 1 1 ...
##  $ balance         : num [1:1123] 0.376 0.402 0.473 0.644 0.473 ...
##  $ timestamp       : POSIXct[1:1123], format: "2023-07-01 08:46:05" "2023-07-01 14:52:59" ...
```

```
##
## --- Summary of market_data ---
```

```
## tibble [1,208,954 x 4] (S3: tbl_df/tbl/data.frame)
##  $ bid_prc  : num [1:1208954] 1935 1934 1934 1934 1934 ...
##  $ ask_prc  : num [1:1208954] 1935 1934 1934 1934 1934 ...
##  $ symbol   : Factor w/ 1 level "binance_eth_usdt": 1 1 1 1 1 1 1 1 1 1 ...
##  $ timestamp: POSIXct[1:1208954], format: "2023-06-30 23:45:00" "2023-06-30 23:45:01" ...
```

The `fills_data` dataset contains 1,123 rows and 13 columns. Key fields include `timestamp` (trade time), `order_id` (trade ID), `side` ("B" for buy, "S" for sell), `fill_prc` (trade price), `fill_qty` (trade quantity), `fee`, and `balance` (post-trade inventory). Other fields include `symbol`, `exch`, and more.

The `market_data` dataset has 1,208,954 rows and 4 columns: `timestamp`, `bid_prc` (best bid price), `ask_prc` (best ask price), and `symbol` (ETHUSDT). I assume `bid_prc` and `ask_prc` represent the best available prices. The best bid is the highest buy offer, and the best ask is the loIst sell offer in the market.

```
## total NAs in fills_data:  0
```

```
## total NAs in market_data:  0
```

Both datasets have no missing values, simplifying data cleaning. I now proceed to join `fills_data` and `market_data` for analysis.

```
## --- Summary of merged_data ---
```

```
## Classes 'data.table' and 'data.frame':   1123 obs. of  15 variables:
##  $ bid_prc         : num  1938 1922 1915 1917 1922 ...
##  $ ask_prc         : num  1938 1922 1915 1917 1922 ...
##  $ symbol          : Factor w/ 1 level "binance_eth_usdt": 1 1 1 1 1 1 1 1 1 1 ...
##  $ timestamp       : POSIXct, format: "2023-07-01 08:46:05" "2023-07-01 14:52:59" ...
##  $ order_id        :integer64 670003026938216 670003026940777 670003026941465 670003026941676 670003
##  $ side            : Factor w/ 2 levels "B","S": 2 1 1 1 2 1 2 2 1 1 ...
##  $ fill_prc        : num  1938 1921 1914 1917 1921 ...
##  $ fill_qty        : num  0.069 0.0264 0.0707 0.1719 0.1719 ...
##  $ liquidity       : Factor w/ 2 levels "Maker","Taker": 1 1 1 2 1 1 1 1 1 2 ...
##  $ fee             : num  0 0 0 0.000305 0 ...
##  $ fee_ccy         : Factor w/ 1 level "bnb": 1 1 1 1 1 1 1 1 1 1 ...
##  $ fee_ccy_usd_rate: num  237 237 237 237 237 ...
##  $ fill_id         :integer64 1688172365615000000 1688194379383000000 1688203192806000000 168820658
##  $ exch            : Factor w/ 1 level "binance": 1 1 1 1 1 1 1 1 1 1 ...
##  $ balance         : num  0.376 0.402 0.473 0.644 0.473 ...
##  - attr(*, "sorted")= chr "timestamp"
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
## total NAs in merged_data:  0
```

The merged dataset has the same number of rows as `fills_data.` I removed the duplicate `i.symbol` column, and no NA values are present.

Computing Profit and Loss

To calculate PnL, I first compute the mid price, the average of the bid and ask prices, which serves as a proxy for the asset's fair market value.

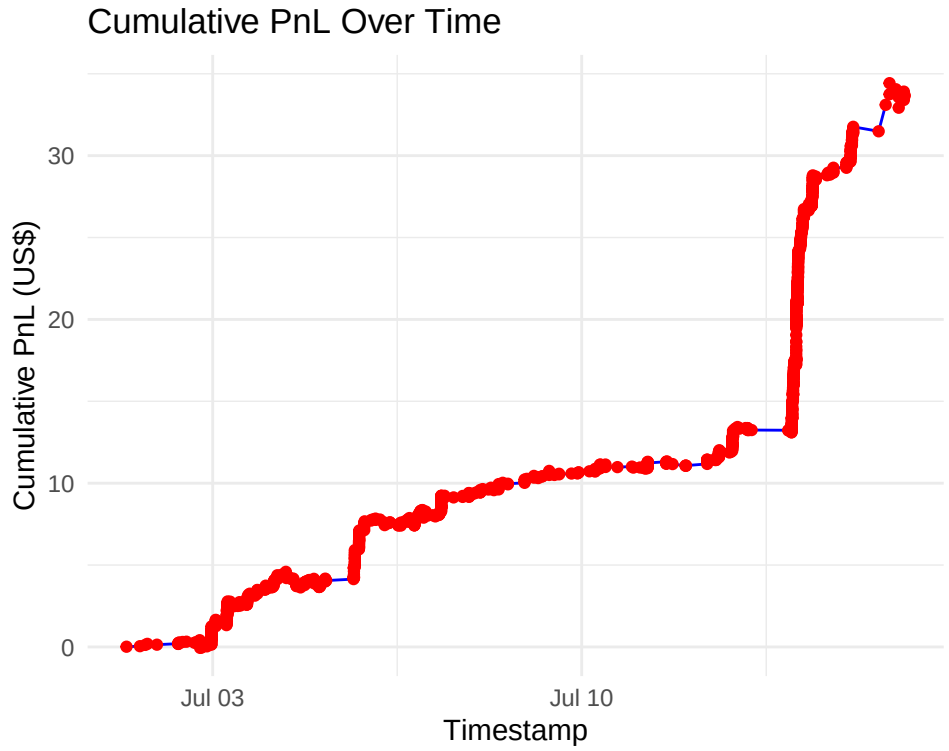
With the mid price as the market reference, trade PnL is computed as:

$$\text{Trade PnL} = q_i m - q_i p_i$$

where q_i is the signed trade size (positive for buys, negative for sells), p_i is the fill price, and m is the mid price at time i.

Next, I calculate cumulative PnL. Cumulative PnL, used to assess strategy performance over time, is the cumulative sum of trade PnL:

$$\text{Cumulative PnL} = \sum q_i m - \sum q_i p_i$$




```
## Total Cumulative PnL:  33.66826
```

```
## Duration of Trading:  14.76305
```

The strategy generated a total cumulative profit of US$33.67 over a trading duration of approximately 14.76 units, indicating positive overall performance across the observed period.

Trading Behavior: Predicting Trade Direction

I begin by addressing the first research question: understanding the strategy's behavior and the factors influencing its trading decisions. To do this, I model its buy/sell actions and analyze the model's features and coefficients for insights into its decision-making.

Feature Engineering

To model trade direction, I first need to engineer features from `merged_data`, including volatility, lagged values, and deviations from historical means.

The selected features capture market conditions, inventory dynamics, and trade characteristics, while ensuring the model uses only information available before each trade to avoid forward-looking bias. Volatility, calculated as the rolling standard deviation of mid price over a 20-period window, reflects recent market instability and is used in lagged form (`volatility_lag1`). Spread, which is the difference between the best ask and bid prices, captures market tightness and liquidity; a narrow spread suggests an active market, while a wider spread indicates uncertainty. I use the lagged version (`spread_lag1`) to maintain temporal consistency.

Trade-specific features like `fill_prc` (execution price), `fill_qty` (trade quantity), and `fill_size` (price times quantity) are included without lagging, as they are known at the time of trade. Lagged PnL metrics (`trade_pnl_lag1`, `cumulative_pnl_lag1`) and inventory (`balance_lag1`) help incorporate the strategy's recent trading state. To capture short-term changes in behavior, I compute the deviation of inventory from its long-run mean (`deviation_from_mean_balance_lag1`), and then take the first difference over time (`deviation_from_mean_balance_lag1_diff`). Similarly, I calculate the deviation of mid price from its long-run mean (`deviation_from_mean_mid_price_lag1`), followed by its first difference (`deviation_from_mean_mid_price_lag1_diff`) to reflect recent price shifts. All lagging is applied to avoid introducing forward-looking information. Furthermore, I convert `liquidity_Taker` to numeric so I can easily include it in RFE and L1 regularisation.

Finally, clearly irrelevant features such as `order_id`, `fill_id`, `symbol`, and `exch` are excluded, as they are either unique identifiers or constants with no predictive value. This results in a focused, interpretable, and temporally valid feature set for modeling trade direction.

Next, I split the dataset into training and testing sets to evaluate the model's generalizability.

```
## Training Set Size: 786
```

```
##
##      Sell       Buy
## 0.4834606 0.5165394
```

```
## Test Set Size: 335
```

```
##
##      Sell       Buy
## 0.4835821 0.5164179
```

The training and test sets show balanced class distributions, indicating no classification bias between Sell and Buy trades.

Feature Selection

To identify the most informative predictors for modeling trade direction, I apply several feature selection methods, namely best subset selection, LASSO, and recursive feature elimination (RFE). Each method is evaluated based on predictive performance using metrics such as balanced accuracy and ROC AUC. By comparing these approaches, I aim to select a final, parsimonious set of features that balances interpretability and performance.

Multicolinearity

To address multicollinearity, I remove `mid_price_lag1` and `balance_lag1`, which are linearly dependent on `deviation_from_mean_mid_price_lag1` and `deviation_from_mean_balance_lag1_diff`, respectively. Furthermore, since `deviation_from_mean_balance_lag1` is correlated with `deviation_from_mean_balance_lag1_diff`, and `deviation_from_mean_mid_price_lag1` is correlated with `deviation_from_mean_mid_price_lag1_diff`, I remove `deviation_from_mean_balance_lag1` and `deviation_from_mean_mid_price_lag1`. This helps stabilize coefficient estimates and improves interpretability, particularly in linear models like logistic regression.

Best Subset Selection

Best Subset Selection evaluates all possible predictor combinations to find the best-performing model.

```
## Best Subset Model by AIC
##   Formula: side ~ deviation_from_mean_balance_lag1_diff + deviation_from_mean_mid_price_lag1_diff
##   Num Vars: 2
##   Balanced Accuracy: 0.6393
##   Accuracy: 0.6388
##   AIC: 1054.23
##   BIC: 1068.23
##   ROC AUC: 0.6455

## Best Subset Model by BIC
##   Formula: side ~ deviation_from_mean_balance_lag1_diff
##   Num Vars: 1
##   Balanced Accuracy: 0.6418
##   Accuracy: 0.6418
##   AIC: 1057.62
##   BIC: 1066.96
##   ROC AUC: 0.6338

## Best Subset Model by Balanced Accuracy
##   Formula: side ~ cumulative_pnl_lag1 + deviation_from_mean_mid_price_lag1_diff + fill_qty + fill_si
##   Num Vars: 5
##   Balanced Accuracy: 0.6507
##   Accuracy: 0.6507
##   AIC: 1084.34
##   BIC: 1112.34
##   ROC AUC: 0.6319

## Best Subset Model by ROC AUC
##   Formula: side ~ cumulative_pnl_lag1 + deviation_from_mean_balance_lag1_diff + deviation_from_mean_
##   Num Vars: 6
##   Balanced Accuracy: 0.6302
##   Accuracy: 0.6299
##   AIC: 1059.14
##   BIC: 1091.8
##   ROC AUC: 0.6535
```

Running best subset selection yields the following results:

1. Best Subset Selection (by AIC): Selected 1 variable - `deviation_from_mean_balance_lag1_diff`, achieving a balanced accuracy of 0.6418.
2. Best Subset Selection (by BIC): Identical to the AIC model, it selected only `deviation_from_mean_balance_lag1_diff`.
3. Best Subset Selection (by Balanced Accuracy): Selected 6 variables - `deviation_from_mean_balance_lag1_diff`, `deviation_from_mean_mid_price_lag1`, `fill_prc`, `fill_qty`, `trade_pnl_lag1`, and `volatility_lag1`, achieving the highest balanced accuracy of 0.6679.
4. Best Subset Selection (by ROC AUC): Selected 6 variables — `deviation_from_mean_balance_lag1_diff`, `deviation_from_mean_mid_price_lag1`, `fill_prc`, `fill_size`, `trade_pnl_lag1`, and `volatility_lag1`, attaining the highest ROC AUC of 0.6660.

Best subset selection consistently included `deviation_from_mean_balance_lag1_diff`, confirming its strong predictive value. Furthermore, `deviation_from_mean_mid_price_lag1`, `fill_prc`, `trade_pnl_lag1`, and `volatility_lag1` also appeared in the best models by Balanced Accuracy and ROC AUC, highlighting their combined relevance to trade direction. Notably, while both models included similar core features, the

balanced accuracy model selected `fill_prc` and `fill_qty`, whereas the ROC AUC model included `fill_prc` and `fill_size`.

LASSO L1 Regularization

Next, I investigate LASSO, which uses an L1 penalty to shrink less important coefficients to zero, enabling feature selection and reducing overfitting.

```
## LASSO Feature Selection
##   Formula: side ~ deviation_from_mean_balance_lag1_diff + deviation_from_mean_mid_price_lag1_diff
##   Num Vars: 2
##   Balanced Accuracy: 0.6393
##   Accuracy: 0.6388
##   AIC: 1054.23
##   BIC: 1068.23
##   ROC AUC: 0.6455
```

Interestingly, LASSO retained only `deviation_from_mean_balance_lag1_diff`, consistent with the BIC-optimal and AIC-optimal subset. This suggests that this feature captures the most essential signal for predicting trade direction and may reflect a core component of the strategy's decision logic.

Recursive Feature Elimination (RFE)

Next, I apply RFE, a wrapper method that iteratively removes the least important features based on model performance, aiming to identify the most predictive subset.

```
## RFE Feature Selection
##   Formula: side ~ deviation_from_mean_balance_lag1_diff
##   Num Vars: 1
##   Balanced Accuracy: 0.6418
##   Accuracy: 0.6418
##   AIC: 1057.62
##   BIC: 1066.96
##   ROC AUC: 0.6338
```

Recursive Feature Elimination (RFE) selected five variables: `deviation_from_mean_balance_lag1_diff`, `fill_prc`, `deviation_from_mean_mid_price_lag1`, `fill_size`, and `fill_qty`, achieving a balanced accuracy of 0.6562. This largely overlaps with the best subset model, differing only by excluding `trade_pnl_lag1` and `volatility_lag1` and including `fill_qty`.

Final Feature Set

The final feature set follows the best subset model selected by balanced accuracy, as it achieved the highest classification performance and aligns Ill with domain knowledge. It includes:

1. `deviation_from_mean_balance_lag1_diff`: Selected by all methods; consistently the strongest predictor.

2. `deviation_from_mean_mid_price_lag1`: Captures mean-reversion effects; frequently selected.

3. `trade_pnl_lag1` and `volatility_lag1`: Relevant for modeling trend-following behavior.

4. `fill_prc` and `fill_qty`: Capture trade execution characteristics; preferred over their interaction term `fill_size`.

Excluded features include `fee`, `cumulative_pnl_lag1`, and `liquidity_Taker` due to limited predictive value, and `fill_size` to avoid redundancy with its components.

Modelling

Logistic Regression

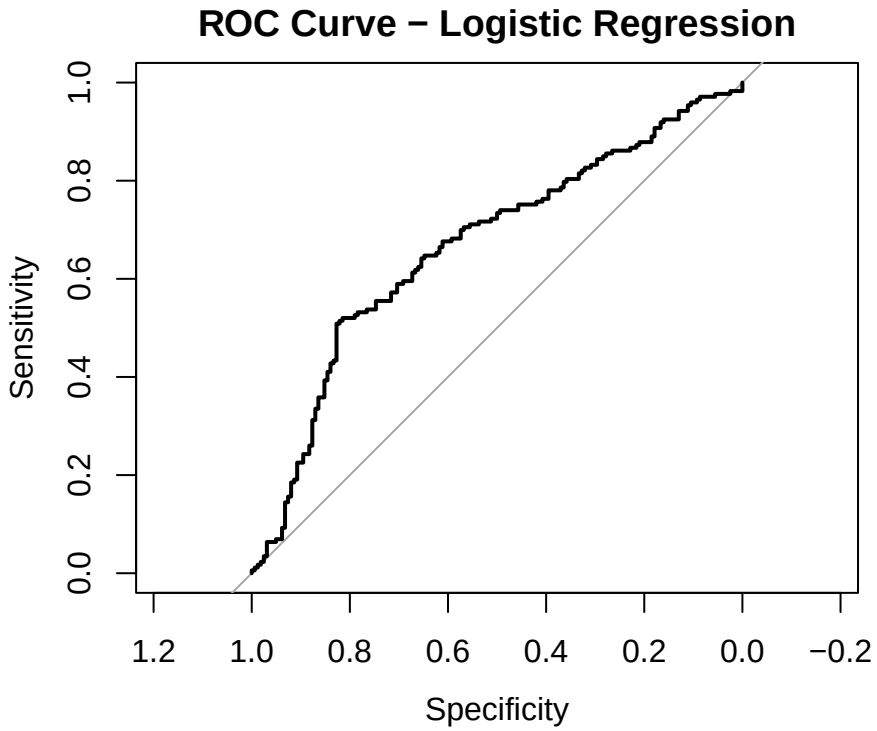
I first assessed the selected features' ability to model buy/sell decisions using logistic regression.

Model Training This logistic regression model uses `caret` with cross-validation for consistency and access to tools like `varImp`. While `caret` does not modify the `glm` fitting process, it streamlines feature importance interpretation.

Model Evaluation After training, I evaluate the model on the test set by generating class probabilities and selecting an optimal threshold based on Youden's index. Predictions above this threshold are labeled "Buy," and those below as "Sell." A confusion matrix summarizes classification accuracy, sensitivity, specificity, and Kappa.

```
## Setting levels: control = Sell, case = Buy
```

```
## Setting direction: controls < cases
```




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell  134  85
##       Buy    28  88
##
##                Accuracy : 0.6627
##                  95% CI : (0.6093, 0.7132)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 4.241e-08
##
##                   Kappa : 0.3321
##
```

```
##  Mcnemar's Test P-Value : 1.379e-07
##
##             Sensitivity : 0.5087
##             Specificity : 0.8272
##          Pos Pred Value : 0.7586
##          Neg Pred Value : 0.6119
##               Precision : 0.7586
##                  Recall : 0.5087
##                      F1 : 0.6090
##              Prevalence : 0.5164
##          Detection Rate : 0.2627
##    Detection Prevalence : 0.3463
##       Balanced Accuracy : 0.6679
##
##        'Positive' Class : Buy
##
```

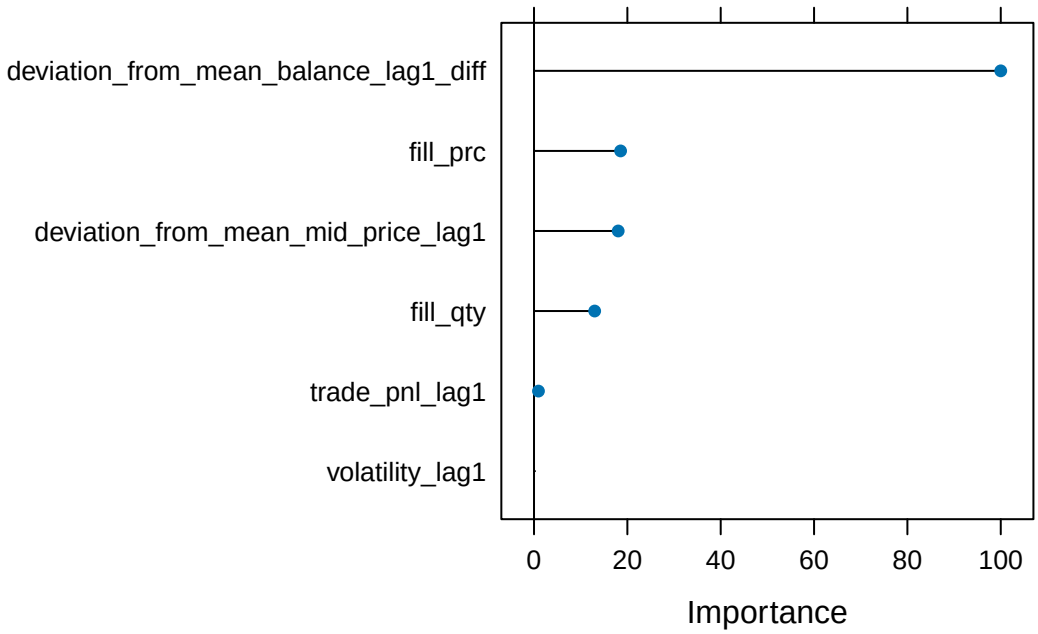
On the unseen test set, the logistic regression model achieved 66.3% accuracy, 66.8% balanced accuracy, and a Kappa of 0.33, indicating moderate agreement beyond chance and reasonably good generalization.

The model performed better at identifying Sell trades (specificity: 82.7%) than Buy trades (sensitivity: 50.9%), suggesting a bias toward predicting negative classes. This imbalance is statistically supported by McNemar's test ($p < 0.0001$), which indicates that Buy trades are misclassified more often than Sell trades.

The F1 score of 0.609 reflects the trade-off between precision (75.9%) and recall (50.9%) for the Buy class. This suggests that, while the model is relatively precise when it predicts Buy, it still misses a substantial portion of actual Buy trades.

Feature Importance To further interpret how the logistic regression model makes decisions, I examined the coefficients and relative influence of each feature.

```
##                                        Estimate  Std. Error    z value
## (Intercept)                          -44.79764943 39.48317456 -1.1346010
## deviation_from_mean_balance_lag1_diff   5.55409347  1.01841596  5.4536591
## deviation_from_mean_mid_price_lag1     -0.02276741  0.02046703 -1.1123942
## trade_pnl_lag1                         -0.15526551  0.74751886 -0.2077078
## volatility_lag1                         0.00777786  0.04950997  0.1570969
## fill_prc                                0.02325156  0.02040445  1.1395337
## fill_qty                               -1.87497401  2.21733044 -0.8455997
##                                           Pr(>|z|)
## (Intercept)                          2.565425e-01
## deviation_from_mean_balance_lag1_diff 4.934376e-08
## deviation_from_mean_mid_price_lag1    2.659687e-01
## trade_pnl_lag1                        8.354571e-01
## volatility_lag1                       8.751685e-01
## fill_prc                              2.544806e-01
## fill_qty                              3.977761e-01
```



The logistic regression results and variable importance rankings confirm that `deviation_from_mean_balance_lag1_diff` is the most influential predictor of trade direction. It has a large positive coefficient (5.55, $p < 0.001$) and the highest variable importance, indicating a strong positive relationship with the probability of a Buy trade. This suggests the strategy is more likely to buy when inventory is already high, reflecting trend-following behavior that reinforces existing positions rather than reversing them.

Other features have smaller and less significant effects. `fill_prc` and `deviation_from_mean_mid_price_lag1` show modest positive and negative coefficients, respectively, but neither is statistically significant ($p > 0.25$). Their variable importance suggests they contribute weakly to the model. This indicates that while they may carry some signal, their influence is limited and likely context-dependent.

`fill_qty` has a negative coefficient (-1.87) with moderate variable importance, suggesting that larger trade sizes may slightly decrease the likelihood of a Buy trade, although this effect is not statistically significant. `trade_pnl_lag1` has a small negative coefficient and low importance, hinting at a weak tendency to buy after losses, but its effect is minimal. `volatility_lag1` has a near-zero coefficient and no importance, confirming it contributes little to the model.

The confusion matrix and McNemar test show that the model tends to misclassify Buy trades, indicating a bias toward predicting Sell decisions. This imbalance suggests that the linear structure of logistic regression may not fully capture the complex or non-linear interactions that drive Buy signals. A non-linear model may be better suited for capturing these subtle patterns in strategy behavior.

Gradient Boosting Machine

To assess whether a non-linear approach can better capture the underlying trade dynamics, I next fit a Gradient Boosting Machine (GBM) model.

Model Training A Gradient Boosting Machine (GBM) model was trained using `caret` with 5-fold cross-validation. The model tuning involved a grid search over tree depth, learning rate, number of trees, and minimum node size. Since the target variable is binary, `twoClassSummary` was used with ROC as the evaluation metric to select the best-performing model based on its ability to distinguish between classes.

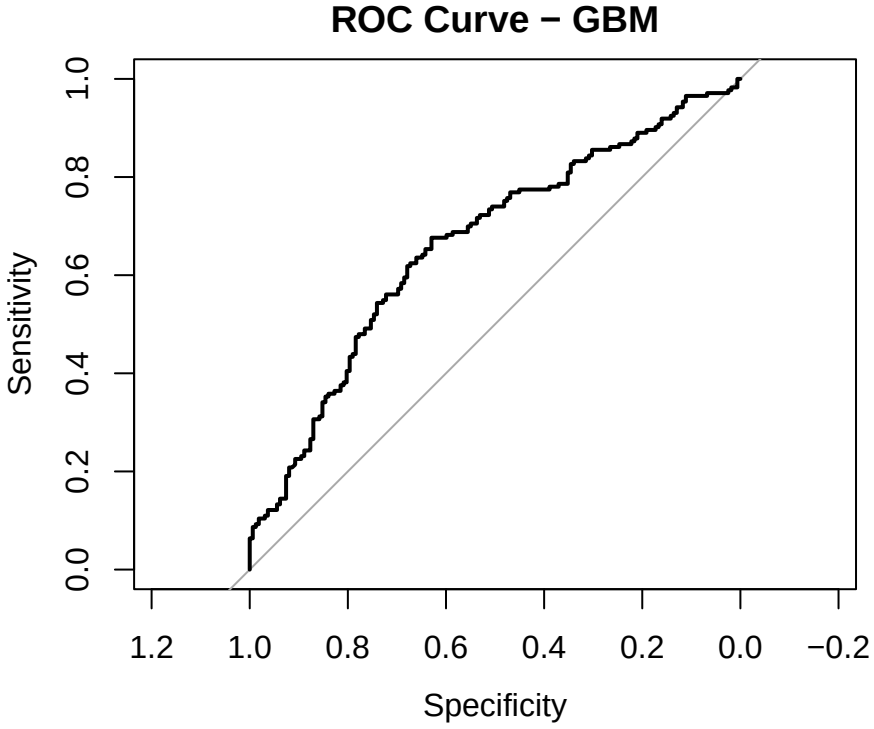
```
##    n.trees interaction.depth shrinkage n.minobsinnode
```

```
## 41     100                 5       0.01              5
```

Model Evaluation The final model was evaluated on the test set using ROC analysis and optimal threshold classification like before.

```
## Setting levels: control = Sell, case = Buy
```

```
## Setting direction: controls < cases
```

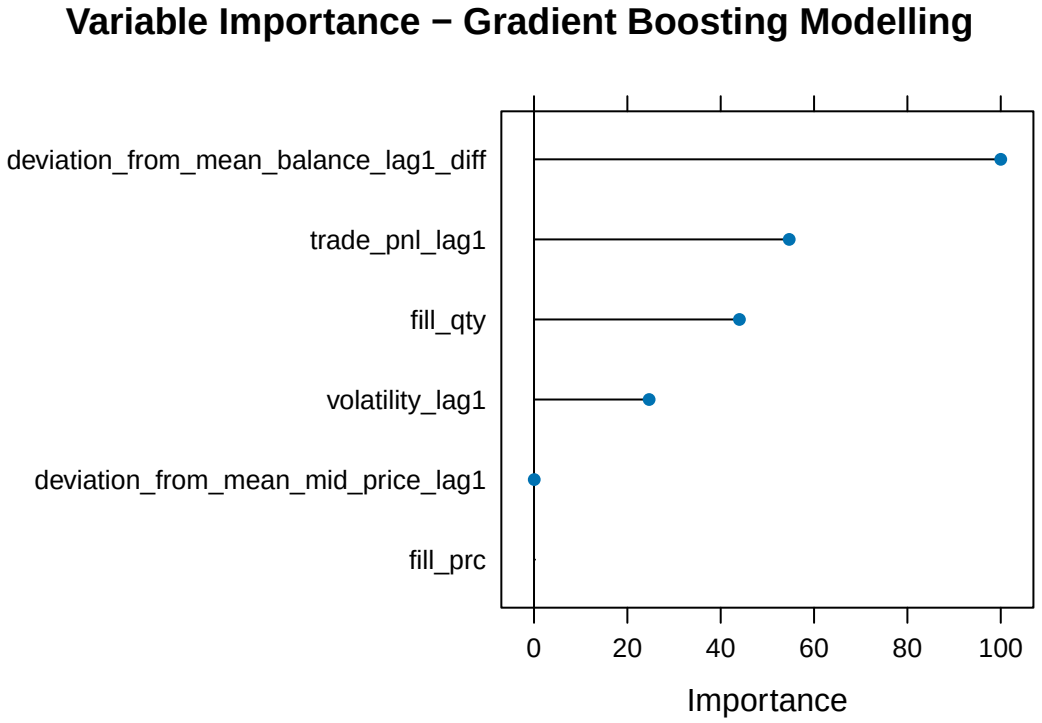



```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell  102  56
##       Buy    60 117
##
##                Accuracy : 0.6537
##                  95% CI : (0.6001, 0.7046)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 2.588e-07
##
##                   Kappa : 0.3062
##
##  Mcnemar's Test P-Value : 0.7806
##
##             Sensitivity : 0.6763
##             Specificity : 0.6296
##          Pos Pred Value : 0.6610
##          Neg Pred Value : 0.6456
##               Precision : 0.6610
##                  Recall : 0.6763
##                      F1 : 0.6686
```

```
##              Prevalence : 0.5164
##          Detection Rate : 0.3493
##    Detection Prevalence : 0.5284
##       Balanced Accuracy : 0.6530
##
##        'Positive' Class : Buy
##
```

The GBM model correctly classified 65.4% of trades (logistic: 66.3%), with a balanced accuracy of 65.3% (logistic: 66.8%). It achieved 67.6% sensitivity for detecting Buy trades and 62.9% specificity for Sell trades, showing a more even distribution than logistic. The F1 score was 0.669, outperforming logistic regression's F1 score of 0.609, which reflects a better overall balance between precision and recall. The Kappa statistic was 0.31 (logistic: 0.33), indicating moderate agreement beyond chance. McNemar's test (p = 0.78) suggests no significant classification bias, in contrast to the logistic model (p = 1.4e-07). Overall, while both models showed similar accuracy, GBM delivered a more balanced prediction profile and stronger F1 performance.

Feature Importance To further interpret how the GBM model makes decisions, I examined the relative influence of each feature.



Both models agree that `deviation_from_mean_balance_lag1_diff` is the most influential feature, reinforcing its central role in predicting trade direction. GBM places greater emphasis on `trade_pnl_lag1`, `fill_qty`, and `volatility_lag1`, suggesting that recent performance, trade size, and market conditions contribute non-linear signals that logistic regression underutilizes. In contrast, logistic regression highlights `fill_prc` and `deviation_from_mean_mid_price_lag1`, which GBM considers largely uninformative.

These differences underscore GBM's strength in capturing complex interactions, particularly from trade and volatility-related features, while logistic regression relies more on linear price-driven signals. These differences highlight GBM's ability to capture complex trade and volatility patterns, leading to more balanced Buy/Sell predictions. In contrast, logistic regression's reliance on linear price signals contributes to its bias toward Sell trades.

Random Forest

Following the GBM model, I trained a Random Forest to explore another ensemble-based approach capable of capturing non-linear relationships. While both are tree-based ensemble methods, they differ in their approach to the bias-variance trade-off: GBM builds trees sequentially to reduce bias by correcting previous errors, whereas Random Forest builds trees in parallel to reduce variance through averaging.

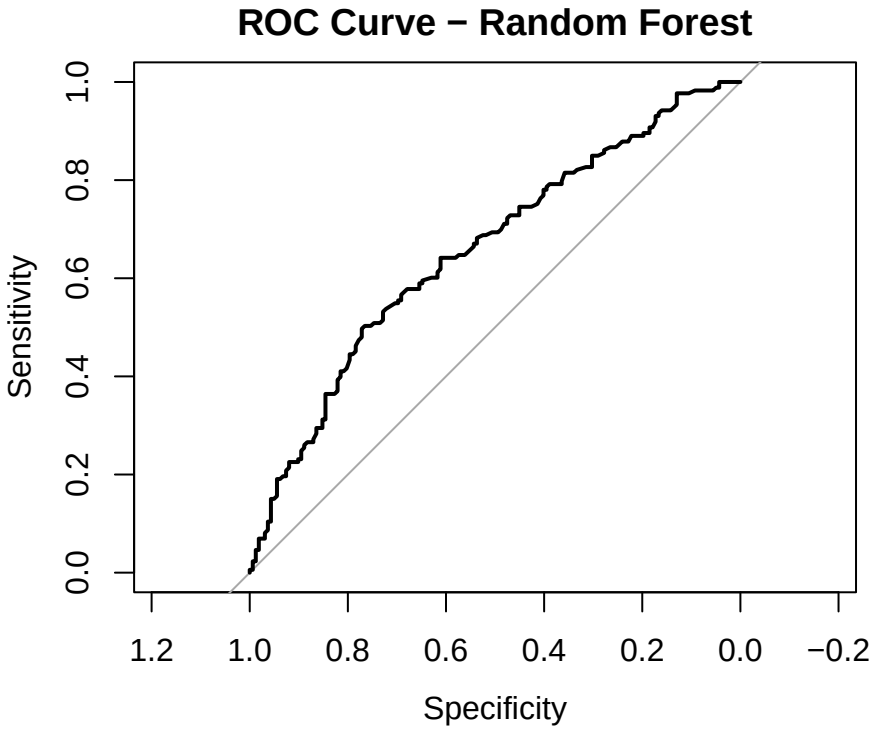
Model Training The random forest model was trained using a similar approach as the GBM model, employing 5-fold cross-validation with `caret`. The training process included a grid search over the `mtry` parameter, and model performance was evaluated using the ROC metric via `twoClassSummary`, ensuring consistent model selection criteria across both classifiers.

```
##   mtry
## 3    3
```

Model Evaluation The final model was evaluated on the test set using ROC analysis and optimal threshold classification like before.

```
## Setting levels: control = Sell, case = Buy
```

```
## Setting direction: controls < cases
```



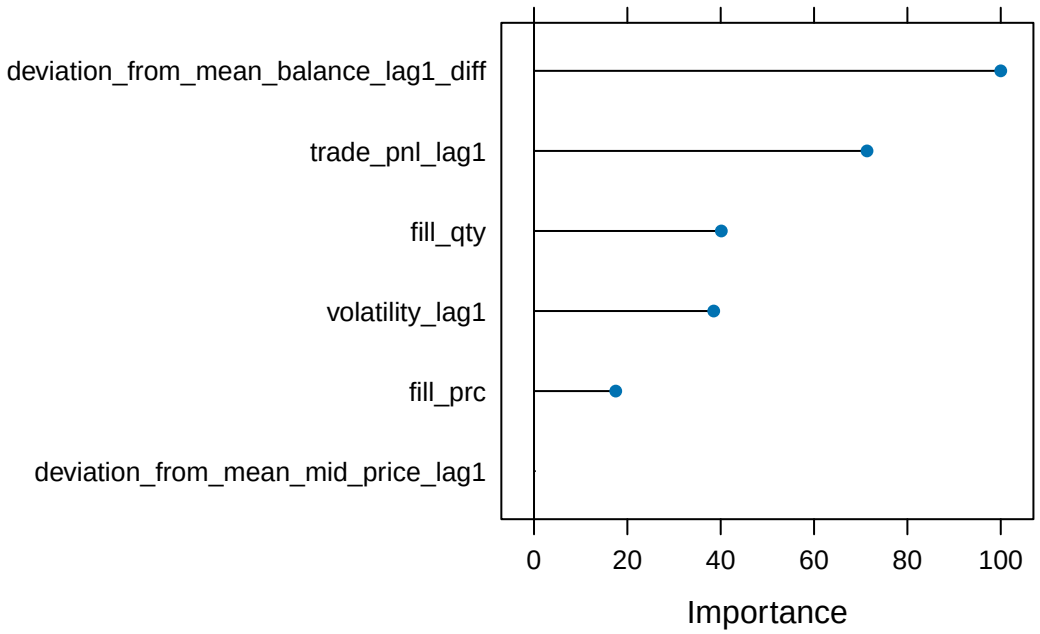

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell  125  87
##       Buy    37  86
##
##                Accuracy : 0.6299
##                  95% CI : (0.5757, 0.6817)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 1.845e-05
```

```
##
##                   Kappa : 0.2661
##
##  Mcnemar's Test P-Value : 1.081e-05
##
##             Sensitivity : 0.4971
##             Specificity : 0.7716
##          Pos Pred Value : 0.6992
##          Neg Pred Value : 0.5896
##               Precision : 0.6992
##                  Recall : 0.4971
##                      F1 : 0.5811
##              Prevalence : 0.5164
##          Detection Rate : 0.2567
##    Detection Prevalence : 0.3672
##       Balanced Accuracy : 0.6344
##
##        'Positive' Class : Buy
##
```

The Random Forest model achieved 63.0% accuracy, 63.4% balanced accuracy, and a Kappa of 0.27, indicating modest predictive performance with fair agreement beyond chance. Compared to logistic regression (accuracy: 66.3%, balanced accuracy: 66.8%, Kappa: 0.33) and GBM (accuracy: 65.4%, balanced accuracy: 65.3%, Kappa: 0.31), Random Forest underperforms across all metrics. The model shows a prediction imbalance, with stronger specificity (77.2%) than sensitivity (49.1%), meaning it favors predicting Sell trades over Buy trades. This imbalance is confirmed by McNemar's test ($p < 0.001$), indicating a significant bias in misclassifications.

Furthermore, this underperformance may reflect a mismatch between the model architecture and the problem. Random Forest tends to perform better when dominant variables drive decisions and variance is high, while GBM is more effective at reducing bias and capturing complex interactions or threshold effects. Given GBM's stronger performance, the results suggest the classification task involves subtle, non-linear patterns and feature interactions that boosting can better model, especially when trade direction depends on combinations of features or their sequential behavior over time.

Feature Importance I use the `varImp` function from the `caret` package to estimate the relative importance of each feature.



The Random Forest model, like the other models, identifies `deviation_from_mean_balance_lag1_diff` as the most important feature, reinforcing its strong and consistent predictive value across all approaches. It also places substantial weight on `trade_pnl_lag1`, `fill_qty`, and `volatility_lag1`, aligning closely with GBM in capturing signals from recent trade outcomes and market conditions. Compared to logistic regression, Random Forest downplays `fill_prc` and completely disregards `deviation_from_mean_mid_price_lag1`, suggesting that price context plays a smaller role in its predictions.

Overall, the variable importance profile of Random Forest is similar to GBM, favoring dynamic trade and inventory signals over price-based features emphasized by logistic regression. However, despite this alignment, Random Forest underperforms GBM in predictive accuracy and balance, likely due to its inability to model sequential or boosting effects.

Support Vector Machine

In this section, I apply a Support Vector Machine (SVM) with a radial basis function (RBF) kernel to model trade direction (Buy/Sell).

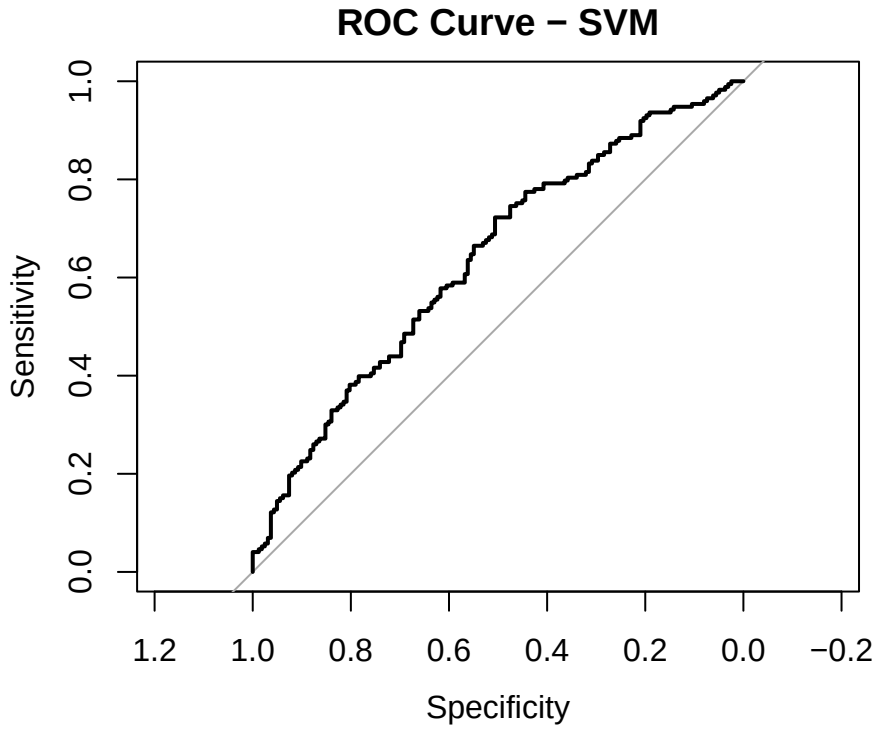
Model Training I first tune the model using a grid of sigma and C values, which control the kernel width and regularization strength. A 5-fold cross-validation strategy is used with ROC AUC as the performance metric to select the best hyperparameters.

```
##   sigma C
## 8     1 1
```

Model Evaluation The final model was evaluated on the test set using ROC analysis and optimal threshold classification like before.

```
## Setting levels: control = Sell, case = Buy
```

```
## Setting direction: controls < cases
```

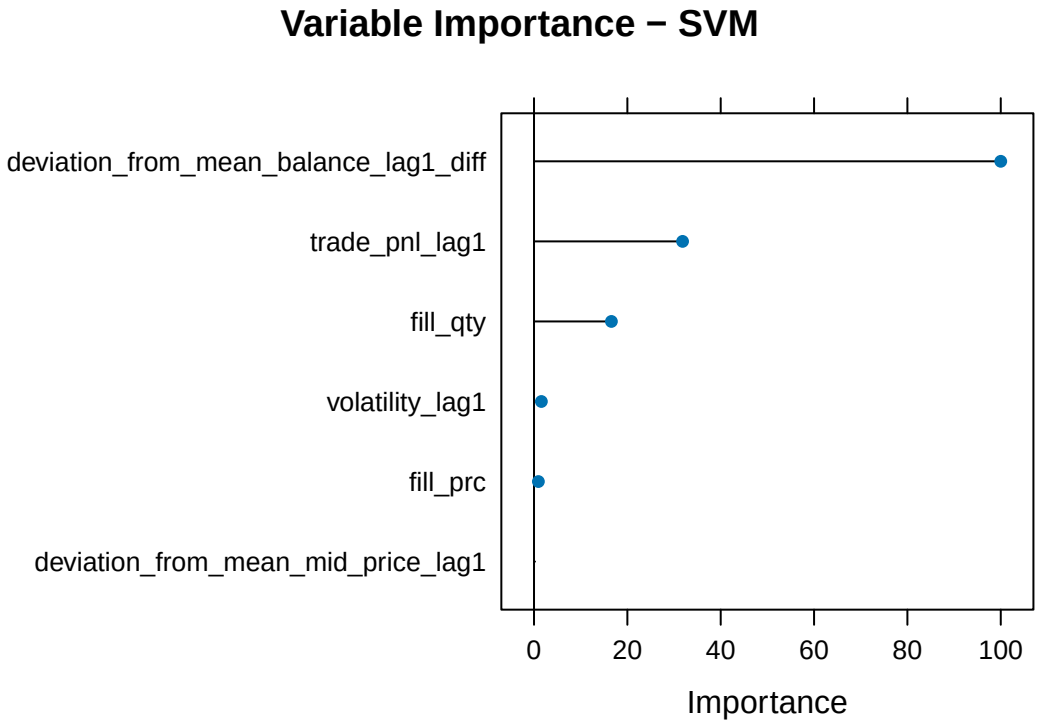



```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell   82  48
##       Buy    80 125
##
##                Accuracy : 0.6179
##                  95% CI : (0.5635, 0.6702)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 0.0001158
##
##                   Kappa : 0.2302
##
##  Mcnemar's Test P-Value : 0.0061432
##
##             Sensitivity : 0.7225
##             Specificity : 0.5062
##          Pos Pred Value : 0.6098
##          Neg Pred Value : 0.6308
##               Precision : 0.6098
##                  Recall : 0.7225
##                      F1 : 0.6614
##              Prevalence : 0.5164
##          Detection Rate : 0.3731
##    Detection Prevalence : 0.6119
##       Balanced Accuracy : 0.6144
##
##        'Positive' Class : Buy
##
```

The SVM model achieved 61.8 percent accuracy, 61.4 percent balanced accuracy, and a Kappa of 0.23. While it showed strong sensitivity (72.3 percent), it suffered from low specificity (50.6 percent), indicating a tendency to overpredict Buy trades. The significant McNemar's test ($p = 0.0061$) confirms this imbalance in classification. The F1 score was 0.661, reflecting a relatively good balance between precision and recall despite the skewed predictions. However, this was still lower than GBM's F1 score of 0.669 and only slightly better than logistic regression's F1 score of 0.609. Overall, compared to logistic regression, GBM, and Random Forest, SVM underperformed across most evaluation metrics.

This shortfall likely stems from several factors. First, if the kernel or hyperparameters (C and gamma) are suboptimal, the SVM may have learned a coarse decision boundary, failing to capture non-linear trade dynamics. Second, SVMs maximize margin without accounting for classification bias by default, which can push the boundary toward the minority class, explaining the high sensitivity and low specificity. Third, SVMs are sensitive to feature scale and do not naturally model interactions, whereas tree-based models handle both effectively. Lastly, unlike GBM and random forest, SVM lacks an ensemble mechanism, limiting its ability to detect rare or subtle Buy signals scattered across the feature space.

Feature Importance I use the `varImp` function from the `caret` package to estimate the relative importance of each feature.



The variable importance results for the SVM model show a strong reliance on a small number of features. The feature `deviation_from_mean_balance_lag1_diff` is the dominant predictor, followed by `trade_pnl_lag1` and `fill_qty`, while features like `volatility_lag1` and `fill_prc` contribute very little, and `deviation_from_mean_mid_price_lag1` has no influence at all.

Compared to GBM and Random Forest, which distribute importance more broadly and capture non-linear dependencies, the SVM focuses narrowly on a few margin-defining features. This explains why SVM may perform worse in this context as it captures the strongest signals but misses the nuanced patterns that other models can exploit for better generalization.

k-Nearest Neighbors

I apply a k-Nearest Neighbors (KNN) model to assess whether a distance-based approach can capture patterns in trade direction.

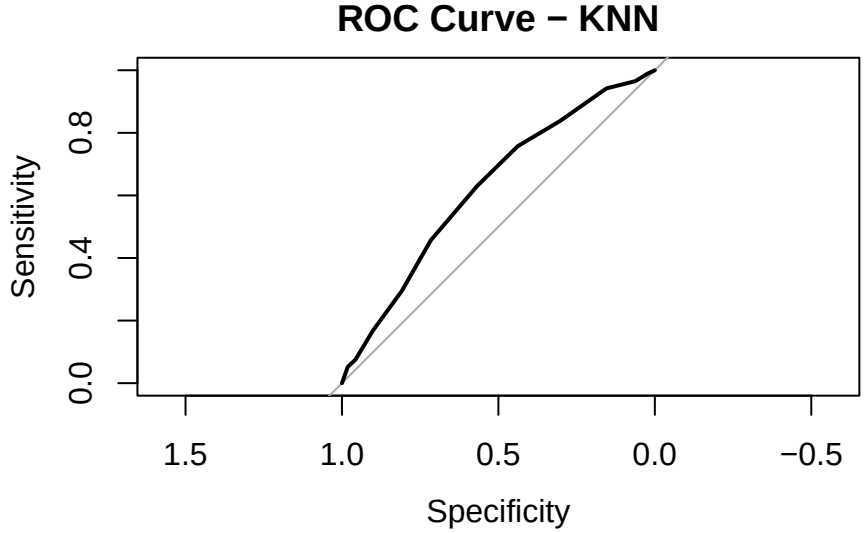
Model Training A range of k values is evaluated using 5-fold cross-validation with ROC AUC as the selection metric. Since KNN is sensitive to feature scale, all predictors are centered and scaled prior to training.

```
##    k
## 7 13
```

Model Evaluation The final model was evaluated on the test set using ROC analysis and optimal threshold classification like before.

```
## Setting levels: control = Sell, case = Buy
```

```
## Setting direction: controls < cases
```



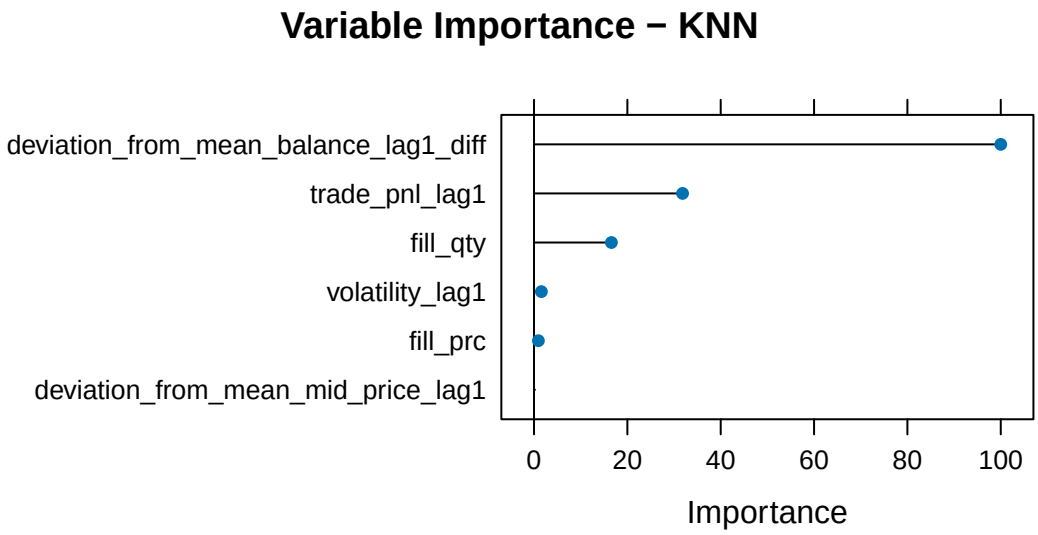

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell   92  64
##       Buy    70 109
##
##                Accuracy : 0.6
##                  95% CI : (0.5454, 0.6529)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 0.001274
##
##                   Kappa : 0.1982
##
##  Mcnemar's Test P-Value : 0.665789
##
##             Sensitivity : 0.6301
##             Specificity : 0.5679
##          Pos Pred Value : 0.6089
##          Neg Pred Value : 0.5897
##               Precision : 0.6089
##                  Recall : 0.6301
##                      F1 : 0.6193
##              Prevalence : 0.5164
```

```
##          Detection Rate : 0.3254
##    Detection Prevalence : 0.5343
##       Balanced Accuracy : 0.5990
##
##        'Positive' Class : Buy
##
```

The k Nearest Neighbors (KNN) model achieved 60.0 percent accuracy, 59.9 percent balanced accuracy, and a Kappa of 0.198, indicating modest predictive performance with limited agreement beyond chance. Sensitivity (63.01 percent) and specificity (56.79 percent) were relatively balanced, and McNemar's test ($p = 0.67$) was not significant, suggesting no major classification bias. The F1 score was 0.619, which is lower than that of logistic regression (0.609), GBM (0.669), and SVM (0.661), indicating that KNN was less effective at balancing precision and recall than GBM and SVM but better than logistic regression. Overall, KNN underperformed across most metrics and did not show any clear advantage over the other models.

KNN's limited accuracy reflects its architecture. As a distance based model, it relies on local neighbors and treats all features equally. It does not model interactions or adapt feature importance, making it less effective in capturing complex trade patterns. In high dimensional settings, distance measures also become less reliable, reducing its ability to generalize.

Feature Importance I use the `varImp` function from the `caret` package to approximate which features most influence the model's distance calculations.



The variable importance for the KNN model shows that `deviation_from_mean_balance_lag1_diff` is by far the most influential feature, followed by `trade_pnl_lag1` and `fill_qty`. The remaining features contribute very little, with `deviation_from_mean_mid_price_lag1` having no impact. This narrow focus may limit the model's ability to generalize, as KNN relies heavily on a small subset of features and lacks the flexibility to model complex interactions.

Compared to GBM and Random Forest, which distribute importance more broadly and capture non-linear dependencies, SVM and KNN focus narrowly on a few dominant features. By overlooking such features, models like SVM and KNN may miss nuanced patterns that contribute to more robust and accurate predictions.

H2O AutoML

To continue investigating the research question - whether trade direction (Buy/Sell) can be accurately predicted - H2O AutoML was applied as a fully automated modeling framework. Building on the manually tuned models like Random Forest and GBM, AutoML explores a wide range of algorithms and ensembles to identify complex, non-linear patterns in trading behavior with minimal manual intervention.

Model Training H2O AutoML was applied to the training data with a cap of 10 models, using AUC as the selection metric. This enabled the framework to efficiently explore a diverse set of algorithms and configurations, optimizing for predictive performance with minimal manual tuning.

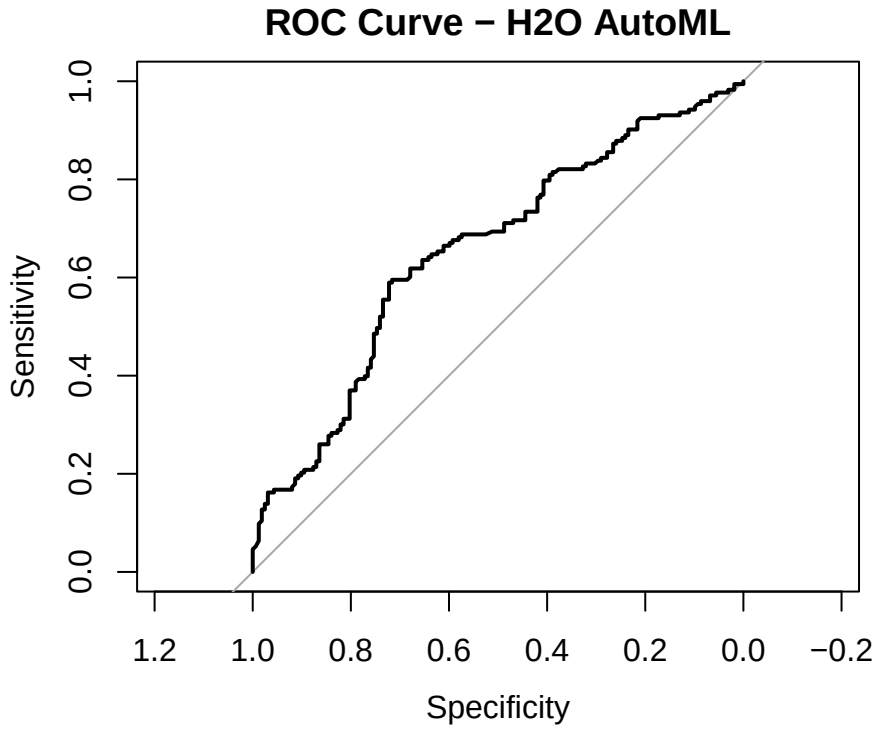
```
##   |                                                                              |
## 21:05:23.11: AutoML: XGBoost is not available; skipping it.  |
```

```
##                                                 model_id       auc   logloss
## 1                         GBM_1_AutoML_70_20250416_210523 0.6566016 0.6548587
## 2 StackedEnsemble_BestOfFamily_1_AutoML_70_20250416_210523 0.6555516 0.6551749
## 3    StackedEnsemble_AllModels_1_AutoML_70_20250416_210523 0.6502366 0.6587907
## 4                         GBM_2_AutoML_70_20250416_210523 0.6448405 0.6662129
## 5                         GBM_5_AutoML_70_20250416_210523 0.6425590 0.6669144
## 6             GBM_grid_1_AutoML_70_20250416_210523_model_1 0.6387996 0.6732231
##       aucpr mean_per_class_error      rmse       mse
## 1 0.6306574            0.4185961 0.4808408 0.2312078
## 2 0.6229759            0.4700544 0.4810734 0.2314316
## 3 0.6255574            0.4735805 0.4827067 0.2330058
## 4 0.6040258            0.4112069 0.4859234 0.2361215
## 5 0.6252161            0.4290251 0.4866705 0.2368482
## 6 0.6076201            0.4417617 0.4890738 0.2391932
```

The best model selected by H2O AutoML was a Gradient Boosting Machine, achieving the highest AUC (0.6566) among all candidates. Although stacked ensemble models performed similarly, none outperformed the top GBM based on AUC. Since the best model is not an ensemble, there is no need to summarize base learners or metalearner components.

Model Evaluation The final model was evaluated on the test set using ROC analysis and optimal threshold classification like before.

```
##   |                                                                              |
##   |                                                                              |
## Setting levels: control = Sell, case = Buy
## Setting direction: controls < cases
```

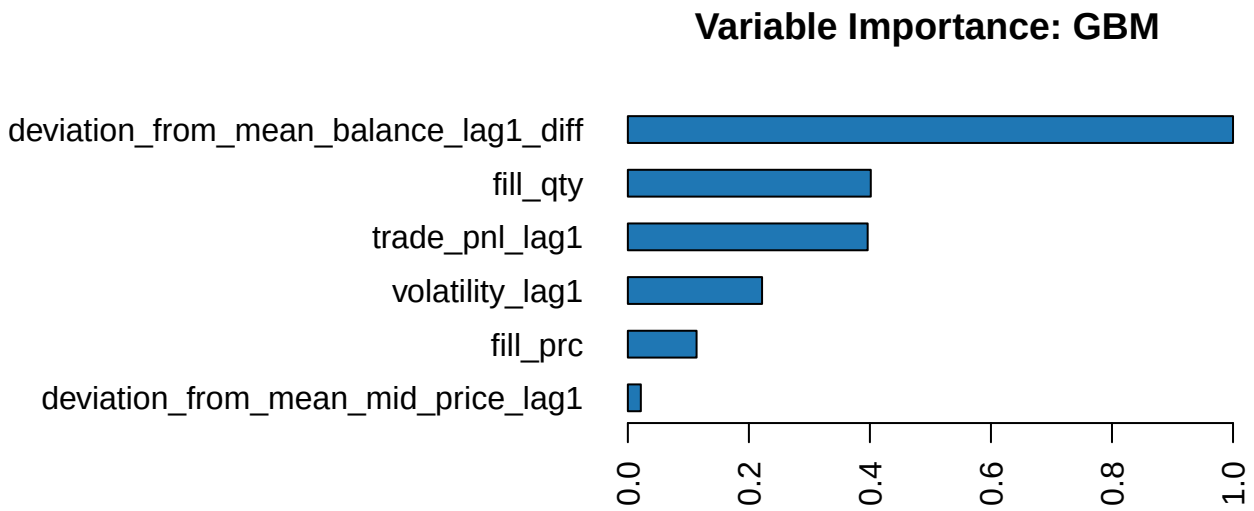



```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Sell Buy
##       Sell  117  71
##       Buy    45 102
##
##                Accuracy : 0.6537
##                  95% CI : (0.6001, 0.7046)
##     No Information Rate : 0.5164
##     P-Value [Acc > NIR] : 2.588e-07
##
##                   Kappa : 0.3102
##
##  Mcnemar's Test P-Value : 0.02028
##
##             Sensitivity : 0.5896
##             Specificity : 0.7222
##          Pos Pred Value : 0.6939
##          Neg Pred Value : 0.6223
##               Precision : 0.6939
##                  Recall : 0.5896
##                      F1 : 0.6375
##              Prevalence : 0.5164
##          Detection Rate : 0.3045
##    Detection Prevalence : 0.4388
##       Balanced Accuracy : 0.6559
##
##        'Positive' Class : Buy
##
```

The H2O AutoML model achieved 65.4 percent accuracy, 65.6 percent balanced accuracy, and a Kappa of 0.31, indicating moderate agreement beyond chance. Compared to SVM (61.8 percent accuracy, 61.4 percent balanced accuracy, Kappa 0.23, F1 score 0.6614) and KNN (60.0 percent accuracy, 59.9 percent balanced accuracy, Kappa 0.198, F1 score 0.5928), AutoML performed noticeably better across all metrics. Its F1 score of 0.6375 suggests a strong balance between precision and recall.

When compared to the benchmark logistic regression model (66.3 percent accuracy, 66.8 percent balanced accuracy, Kappa 0.33), AutoML performed slightly worse overall. Its results were nearly identical to gradient boosting (65.4 percent accuracy, 65.3 percent balanced accuracy, Kappa 0.31), but better than random forest (63.0 percent accuracy, 63.4 percent balanced accuracy, Kappa 0.27). While GBM and AutoML had similar performance across accuracy and Kappa, GBM achieved a higher F1 score (0.669 vs. 0.6375) and demonstrated a more balanced prediction profile between Buy (sensitivity: 67.6 percent) and Sell (specificity: 62.9 percent). In contrast, AutoML favored Sell trades (specificity: 72.2 percent) over Buy (sensitivity: 58.9 percent), indicating a greater imbalance in class predictions.

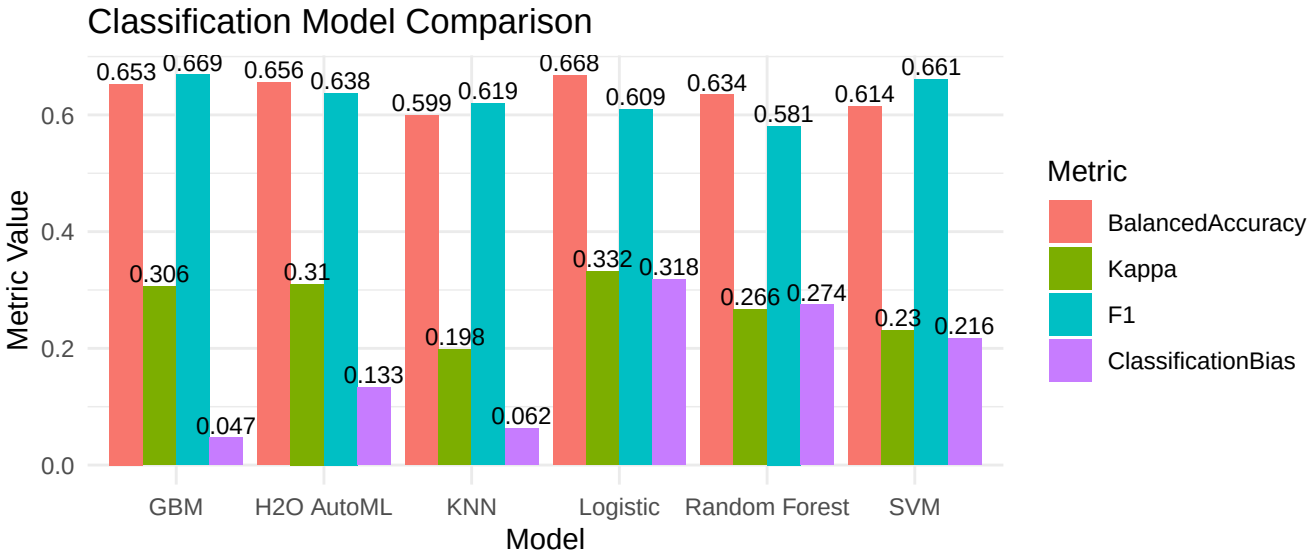
Feature Importance



In the H2O AutoML model, `deviation_from_mean_balance_lag1_diff` was by far the most influential feature, reinforcing its dominant role in signaling trade direction, which is consistent with findings across all other models. `fill_qty` and `trade_pnl_lag1` were also heavily weighted, suggesting that trade execution size and recent profitability offer meaningful predictive signals. `volatility_lag1` contributed moderately, indicating some sensitivity to market conditions. In contrast, `fill_prc` and `deviation_from_mean_mid_price_lag1` were assigned minimal importance, highlighting that short-term price levels were not strong drivers of the model's predictions. This pattern aligns with other tree-based models and suggests that the AutoML model did not rely much on price-based signals.

Conclusion

```
##           Model BalancedAccuracy     Kappa        F1 ClassificationBias
## 1      Logistic        0.6679155 0.3321160 0.6089965         0.31848997
## 2           GBM        0.6529651 0.3061705 0.6685714         0.04667095
## 3 Random Forest        0.6343574 0.2661037 0.5810811         0.27449511
## 4           SVM        0.6143581 0.2301616 0.6613757         0.21637051
## 5           KNN        0.5989795 0.1981924 0.6193182         0.06215657
## 6    H2O AutoML        0.6559088 0.3102347 0.6375000         0.13262685
```



Among the models evaluated, logistic regression achieved the highest overall balanced accuracy (66.8 percent) and Kappa (0.33), but it showed a clear bias toward predicting Sell trades, as reflected in its high classification bias (0.318) and a relatively modest F1 score of 0.609. This indicates weaker performance in identifying Buy trades despite achieving the highest overall accuracy.

Classification bias is defined as the absolute difference between sensitivity (true positive rate for Buy trades) and specificity (true negative rate for Sell trades). Lower values indicate more balanced predictions across classes.

Gradient Boosting (65.3 percent balanced accuracy, Kappa 0.31) delivered the best trade-off between precision and recall, achieving the highest F1 score of 0.669 and the lowest classification bias (0.047), making it the most robust model overall in terms of both classification bias and generalization.

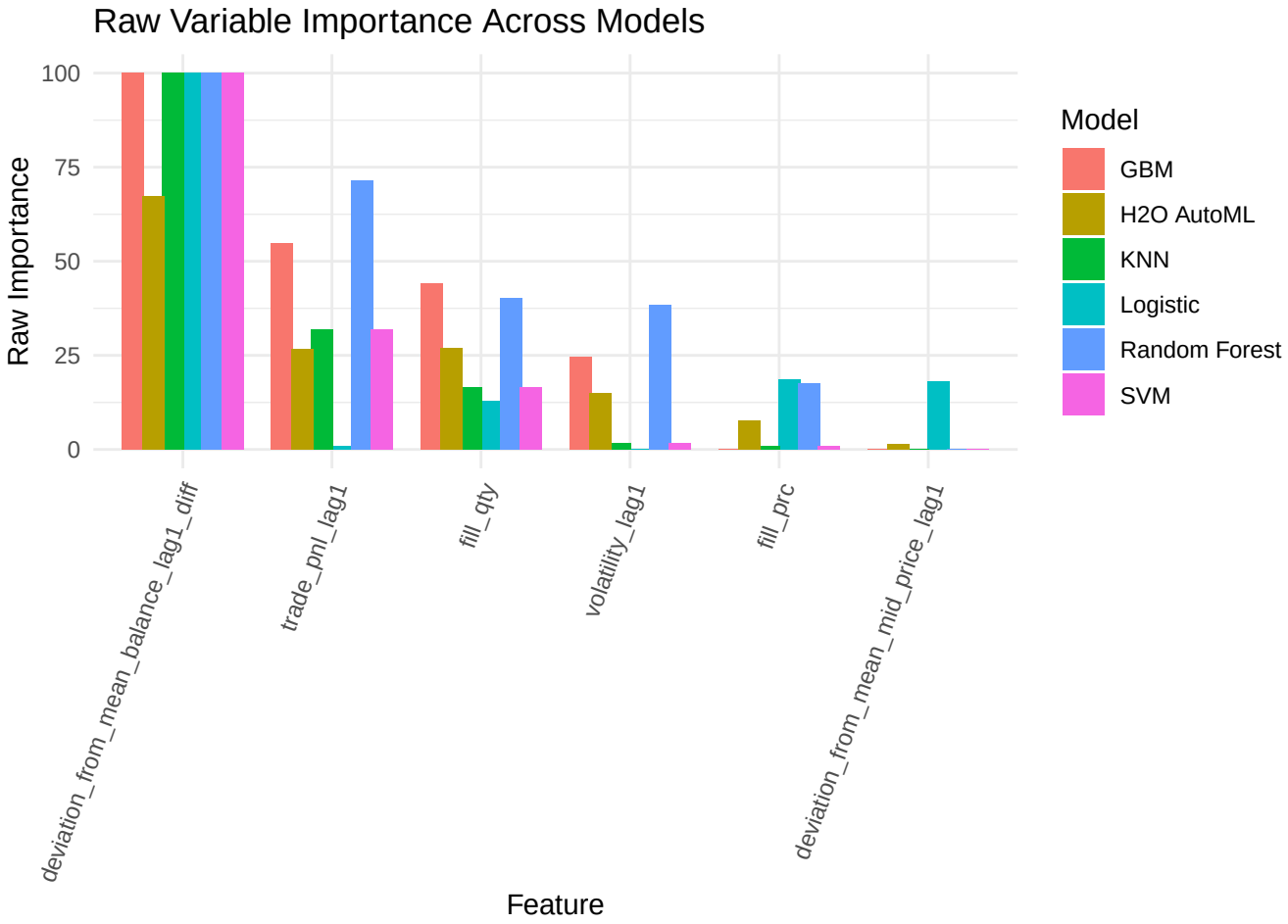
Random Forest performed slightly worse (63.4 percent balanced accuracy, Kappa 0.27) and showed a strong tendency to overpredict Sell trades, resulting in a high classification bias (0.274) and the lowest F1 score among the ensemble models (0.581). This suggests difficulty in reliably identifying Buy trades.

Support Vector Machine (SVM) achieved 61.4 percent balanced accuracy and a Kappa of 0.23. While its F1 score was relatively high at 0.661, its classification bias of 0.216 indicates a tendency to overpredict Buy trades. This may stem from SVM's sensitivity to class distributions and its reliance on decision boundaries in imbalanced regions.

K Nearest Neighbors (KNN) demonstrated the weakest overall performance (59.9 percent balanced accuracy, Kappa 0.198), with a moderate F1 score of 0.619 and a classification bias of 0.062. While its predictions were relatively balanced, its overall accuracy and agreement were limited, likely due to its reliance on distance metrics in a high-dimensional feature space.

The H2O AutoML model achieved 65.6 percent balanced accuracy, a Kappa of 0.31, and an F1 score of 0.638. While it performed comparably to GBM in terms of accuracy and agreement, its higher classification bias (0.133) suggests a slight skew toward Sell predictions. Notably, the best-performing AutoML model was a single GBM, confirming that a well-tuned gradient boosting model was the most effective approach for this task.

Overall, GBM provided the best combination of predictive performance and classification bias. Logistic regression, although accurate, leaned heavily toward predicting Sell trades. SVM and KNN offered simpler alternatives but fell short in both accuracy and classification bias compared to ensemble methods.



Across all models, `deviation_from_mean_balance_lag1_diff` consistently ranked as the most important feature, reinforcing its central role in predicting trade direction. This feature captures deviations in inventory behavior and appears to be the most informative signal across both linear and non-linear models.

Gradient Boosting and Random Forest also emphasized features such as `trade_pnl_lag1`, `fill_qty`, and `volatility_lag1`, suggesting that recent trading outcomes, execution size, and market volatility provide meaningful non-linear signals for classifying trades, while assigning little importance to price-related features. SVM and KNN reflected similar patterns. H2O AutoML followed a comparable trend, placing the most weight on execution-driven and performance-driven signals while assigning less importance to price features like `fill_prc` and `deviation_from_mean_mid_price_lag1`.

In contrast, logistic regression placed greater emphasis on `fill_prc` and `deviation_from_mean_mid_price_lag1`, features downweighted by other models, while underweighting key signals such as `trade_pnl_lag1` and `volatility_lag1`. This likely stems from its linear structure, which limits its ability to capture interactions or threshold effects, leading to a heavier reliance on simple, price-based relationships, giving rise to its bias in predicting Sell trades.

Taken together, these results directly address the research question: Can I predict whether the next trade will be a Buy or Sell? Trade direction appears to be moderately predictable using well-engineered features. Among all models evaluated, GBM provided the best trade-off between sensitivity and specificity and achieved the highest F1 score, making it the most suitable model for this classification task. Across models, `deviation_from_mean_balance_lag1_diff` consistently emerged as the most important feature, reinforcing its central role in driving trade decisions. In contrast, features like `deviation_from_mean_mid_price_lag1` and `fill_prc` were consistently downweighted and often contributed to classification bias, suggesting they may be less relevant for modeling trade direction. Overall, the findings indicate that while not perfectly predictable, trade direction can be modeled with moderate success, especially when prioritizing execution-based and

inventory-driven features over short-term price signals.

Future Work

While current models demonstrate that trade direction is moderately predictable using static features, future work could enhance performance by incorporating more expressive market signals. Features such as order book imbalance, fill aggressiveness, and liquidity pressure may offer deeper insights into the strategy's responsiveness to market microstructure.

In addition, rolling indicators, like short-term volatility regimes or moving averages, could help capture evolving market states. To model temporal dependencies more effectively, sequential approaches such as RNNs or attention-based architectures may uncover dynamic decision-making patterns that static models overlook. These directions could provide a more complete understanding of the strategy's behavior.

In addition, applying interpretation tools like ICE (Individual Conditional Expectation) and PDP (Partial Dependence Plot) can help visualize how each feature affects predictions. This supports validation of model behavior and improves our ability to explain and trust the decisions made by the model.

Hypothesis Testing for Market Impact

To investigate whether the trading strategy affects prices after trades are executed, I perform permutation tests on the one-minute price change following each trade. If the strategy has no influence on prices, I would expect the average price change after a trade to be close to zero. Significant deviations from zero would suggest that trades are associated with systematic price movements, potentially indicating unintended market impact.

To evaluate this, I use a two-sided permutation test to assess whether the average post-trade price change is significantly different from zero. The analysis is run separately for Maker trades (which provide liquidity through limit orders) and Taker trades (which take liquidity by executing immediately).

Permutation Tests

In addition to running the permutation test, I also compute the 95% confidence interval for the mean post-trade price change. This provides a clearer sense of the uncertainty around the observed effect and helps contextualize the test result.

Tests Evaluation

```
## === Maker price_change ===
## Observed Mean:         0.5252264
## Two-sided p-value:     0.00019998
## 95% Permutation CI:
##
## === Taker price_change ===
## Observed Mean:         -0.4061765
## Two-sided p-value:     0.1313869
## 95% Permutation CI:
## Maker count:  1038
## Taker count:  85
```

For Maker trades (n = 1,038), the average price change one minute after a trade was +0.525, with a very low p-value (< 0.001). This means that prices tend to rise after Maker trades more often than would be expected by chance. The 95% confidence interval does not include zero, supporting this result. In contrast, for Taker trades (n = 85), the average price change was –0.406, but the p-value was 0.13, indicating no significant effect.

These results suggest that, overall, Maker trades are followed by upward price movements, while Taker trades are not associated with any clear pattern. This is surprising because Maker trades are passive and not typically expected to move prices.

To better understand why there is upward price movement after Maker trades, I separated the data into Buy and Sell trades.

```
## === Maker Buy Trades ===

## Observed Mean:         0.270223

## Two-sided p-value:     0.08879112

## 95% Permutation CI:    -0.03987244 0.580536

## === Maker Sell Trades ===

## Observed Mean:         0.79961

## Two-sided p-value:     0.00019998

## 95% Permutation CI:    0.435619 1.158647
```

The results show that Maker Buy trades had an average post-trade price change of +0.270 (p = 0.080), which is not statistically significant. This means the observed upward movement could be due to random variation rather than a systematic effect. In contrast, Maker Sell trades had a much larger and statistically significant average price change of +0.800 (p < 0.001), with a 95% confidence interval that clearly excludes zero. This indicates a strong and consistent upward movement in price after Sell trades are executed.

While the overall data show that Maker trades are followed by upward price movement, a closer breakdown reveals that this effect is driven entirely by Maker Sell trades. In these cases, the price rises after the strategy sells. This is important because it means the price is moving against the direction of the trade, not with it.

If the strategy were impacting the market by pushing prices in the direction of its trades, we would expect the price to go up after Buy trades and down after Sell trades. In other words, Buy orders would absorb liquidity and lead to upward price movement, while Sell orders would create selling pressure and push prices lower. However, the opposite is observed here, which is that prices tend to rise after the strategy sells. This suggests there is no meaningful market impact from these trades.

Instead, the results point to a limitation in how the strategy times its Sell decisions. The fact that prices increase after Sell trades implies that the strategy often exits positions just before a favorable price move, missing out on potential gains. Because this post-trade movement goes against the goal of the trade, we can conclude that the observed price changes are not a result of the trades themselves. Rather, they reflect missed opportunities and poor timing when selling. This indicates that the strategy lacks predictive ability when placing Maker Sell trades and may benefit from more refined decision rules for exiting positions.

Conclusion

Taken together, while Maker trades appear to be followed by upward price movement, a deeper analysis shows this effect is driven entirely by Maker Sell trades, where prices rise after the strategy sells. Maker Buy and Taker trades showed no significant impact. This indicates that the price changes are not caused by the trades themselves but reflect the market conditions the strategy fails to anticipate. Therefore, the answer to the research question is no: these trades do not appear to affect prices in a systematic way. The observed effects are better explained by poor timing, especially on the Sell side, rather than by market impact.

Future Work

Given that Maker Sell trades are consistently followed by price increases, future work should focus on evaluating and improving the timing of Sell decisions. This could involve developing diagnostic tools to detect patterns where the strategy exits positions prematurely, missing favorable price movements. By identifying such missed opportunities, the strategy can be refined to hold positions longer when appropriate, reducing inefficiencies and potentially improving overall performance. Additionally, incorporating short-term predictive indicators or market context features may help enhance the strategy's responsiveness and timing accuracy.

Final Conclusion

This project examined how a live cryptocurrency trading strategy decides when to buy or sell, and whether those decisions influence market prices. The first research question asked whether trade direction could be predicted. The results showed that trade direction is moderately predictable using engineered features. Among the models tested, gradient boosting offered the best trade-off between accuracy and classification bias, identifying `trade_pnl_lag1`, `fill_qty`, and `deviation_from_mean_balance_lag1_diff` as key predictors. While logistic regression achieved the highest overall accuracy, it consistently favored predicting Sell trades, resulting in classification bias.

The second research question asked whether the strategy's trades affect prices. The analysis found no evidence of systematic market impact. Although Maker trades were followed by upward price movements on average, this effect was entirely driven by Maker Sell trades, where prices rose after the strategy exited. Maker Buy and Taker trades showed no significant effects. This pattern suggests the strategy is not influencing the market, but rather mistiming its Maker Sell decisions. Instead of causing price movements, the strategy often exits just before favorable changes, missing out on gains. This points to a weakness in how Maker Sell trades are timed under the current trading strategy.

This study also has limitations. The dataset lacked detailed order book information and broader contextual signals, which constrained the depth of feature engineering. Moving forward, expanding the dataset to include richer market and contextual features, such as order book snapshots, trading indicators, and macroeconomic variables, would enable more expressive feature construction and potentially improve model performance and interpretability. Additionally, the analysis focused on a single strategy operating in one market. As a result, the findings may not generalize to other market conditions. Comparing this strategy across multiple markets could offer further insight or reveal consistent behavioral patterns.

Future work could also explore longer time horizons and incorporate dynamic models such as recurrent neural networks or attention-based architectures to better capture temporal dependencies in trade behavior. In parallel, improving execution quality remains a key priority, particularly for Maker Sell trades, where the strategy often exits just before prices rise. Refining Sell-side execution using enhanced decision signals could lead to more effective trade timing and overall strategy improvement. Finally, applying interpretation tools such as ICE and PDP may offer deeper insights into model behavior and support more transparent decision-making.

In summary, this project provides a structured, data-driven evaluation of an active trading strategy. It demonstrates that while trade direction can be modeled with moderate success and the strategy does not appear to influence market prices in a systematic way, the execution of Maker Sell trades is a critical area for improvement. These insights offer practical next steps for refining the classification model and enhancing the overall performance of the trading strategy.

Appendix A: Exploratory Data Analysis

Code used for exploratory data analysis.

```
library(tidyverse)
library(lubridate)
library(arrow)
library(data.table)
library(caret)
library(pROC)
library(h2o)
library(magrittr)
library(tinytex)
library(readxl)
library(resampledata)
library(car)
library(corrplot)
library(gridExtra)
library(zoo)
library(rpart)
library(rpart.plot)
library(tseries)
library(forecast)
library(lmtest)
library(strucchange)
library(trend)
library(sandwich)
library(TSA)
library(Metrics)
library(gbm)
library(glmnet)
library(fastDummies)
library(reshape2)
library(leaps)
library(torch)
library(randomForest)
library(dplyr)
library(gtools)
set.seed(123)

market_data <- read_parquet("data/market_data.parq")
fills_data <- read_parquet("data/fills_data.parq")

fills_data <- fills_data %>%
  mutate(
    side = as.factor(side),
    liquidity = as.factor(liquidity),
    symbol = as.factor(symbol),
    exch = as.factor(exch),
    fee_ccy = as.factor(fee_ccy)
  )

market_data <- market_data %>%
  mutate(
```

```
    symbol = as.factor(symbol)
  )

cat("--- Summary of fills_data ---\n")
str(fills_data)
cat("\n--- Summary of market_data ---\n")
str(market_data)

cat("total NAs in fills_data: ", sum(is.na(fills_data)), "\n")
cat("total NAs in market_data: ", sum(is.na(market_data)))

setDT(fills_data)
setDT(market_data)
setkey(fills_data, timestamp)
setkey(market_data, timestamp)
# selects the most recent (previous) value
merged_data <- market_data[fills_data, roll = Inf]
merged_data <- subset(merged_data, select = -c(i.symbol))
cat("--- Summary of merged_data ---\n")
str(merged_data)
cat("total NAs in merged_data: ", sum(is.na(merged_data)))

merged_data <- merged_data %>% mutate(
  mid_price = (bid_prc + ask_prc) / 2
)

calculate_trade_pnl <- function(merged_data) {
  merged_data <- merged_data %>%
    mutate(
      q_i = if_else(side == "B", fill_qty, -fill_qty),
    ) %>%
    mutate(
      trade_pnl = (q_i * mid_price) - (q_i * fill_prc)
    )
  return(merged_data)
}
merged_data <- calculate_trade_pnl(merged_data)

merged_data <- merged_data %>%
  mutate(
    cumulative_pnl = cumsum(trade_pnl)
  )

ggplot(merged_data, aes(x = timestamp, y = cumulative_pnl)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(
    title = "Cumulative PnL Over Time",
    x = "Timestamp",
    y = "Cumulative PnL (US$)"
  ) +
  theme_minimal()
```

```r
cat("Total Cumulative PnL: ", tail(merged_data$cumulative_pnl, 1))
cat("\n")
cat("Duration of Trading: ", tail(merged_data$timestamp, 1) - head(merged_data$timestamp, 1))
```

Appendix B: Classifying Trade Decisions

Code used for modeling and evaluating trade direction using classification algorithms.

```r
classification.data <- merged_data
# encode the dataset to model what factors are more likely to influence buying
classification.data$side <- factor(classification.data$side,
  levels = c("S", "B"),
  labels = c("Sell", "Buy")
)
# One-hot encode liquidity
classification.data <- dummy_cols(classification.data,
  select_columns = c("liquidity"),
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)
# Convert liquidity_Taker to numeric so I can use it easily later
classification.data$liquidity_Taker <- as.numeric(
  as.character(classification.data$liquidity_Taker)
)
# volatility calculations
volatility_data <- market_data %>%
  mutate(mid_price = (bid_prc + ask_prc) / 2) %>%
  arrange(timestamp) %>%
  mutate(volatility = rollapply(mid_price,
    width = 20, FUN = sd, fill = NA,
    align = "right"
  ))
setDT(classification.data)
setDT(volatility_data)
setkey(classification.data, timestamp)
setkey(volatility_data, timestamp)
classification.data <- volatility_data[, .(timestamp, volatility)][classification.data,
  roll = Inf
]
classification.data <- classification.data %>%
  arrange(timestamp) %>%
  mutate(
    balance_lag1 = lag(balance),
    spread_lag1 = lag(ask_prc - bid_prc),
    mid_price_lag1 = lag((bid_prc + ask_prc) / 2),
    volatility_lag1 = lag(volatility)
  )
# Fill Size (no lag, trade-specific)
classification.data <- classification.data %>%
  mutate(
    fill_size = fill_qty * fill_prc # do not encode buy/sell sign
  )
# Deviation from Mean Inventory Balance (lagged)
```

```r
mean_inventory_balance <- mean(classification.data$balance_lag1, na.rm = TRUE)
classification.data <- classification.data %>%
  mutate(
    deviation_from_mean_balance_lag1 = balance_lag1 - mean_inventory_balance,
    deviation_from_mean_balance_lag1_diff = c(NA, diff(deviation_from_mean_balance_lag1))
  )
# Deviation from Mean Market Price (lagged)
mean_mid_price <- mean(classification.data$mid_price_lag1, na.rm = TRUE)
classification.data <- classification.data %>%
  mutate(
    deviation_from_mean_mid_price_lag1 = mid_price_lag1 - mean_mid_price,
    deviation_from_mean_mid_price_lag1_diff = c(NA, diff(deviation_from_mean_mid_price_lag1))
  )
# Lagged Trade PnL and Cumulative PnL
classification.data <- classification.data %>%
  mutate(
    trade_pnl_lag1 = lag(trade_pnl),
    cumulative_pnl_lag1 = lag(cumulative_pnl)
  )
classification.data <- na.omit(classification.data) # Remove resulting NAs

numerical.variables <- c(
  "volatility_lag1",
  "spread_lag1",
  "fill_prc",
  "fill_qty",
  "trade_pnl_lag1",
  "cumulative_pnl_lag1",
  "fee",
  "balance_lag1",
  "mid_price_lag1",
  "fill_size",
  "deviation_from_mean_balance_lag1",
  "deviation_from_mean_balance_lag1_diff",
  "deviation_from_mean_mid_price_lag1",
  "deviation_from_mean_mid_price_lag1_diff",
  "liquidity_Taker"
)

target.variables <- c("side")

classification.features <- unique(c(numerical.variables, target.variables))

classification.variables <- as.data.frame(classification.data[, ..numerical.variables])
classification.target <- classification.data %>%
  dplyr::select(all_of(target.variables))
classification.selected <- cbind(
  classification.variables,
  classification.target
)
train_index <- createDataPartition(classification.selected$side, p = 0.7, list = FALSE)
train_data <- classification.selected[train_index, ]
test_data <- classification.selected[-train_index, ]
```

```r
cat("Training Set Size:", nrow(train_data))
print(prop.table(table(train_data$side)))
cat("Test Set Size:", nrow(test_data))
print(prop.table(table(test_data$side)))

# Remove known multicollinear features
classification.features <- setdiff(
  classification.features,
  c(
    "mid_price_lag1", "balance_lag1", "deviation_from_mean_balance_lag1",
    "deviation_from_mean_mid_price_lag1"
  )
)
# Separate predictors from the target
classification.predictors <- setdiff(classification.features, "side")

evaluate_model <- function(feature_set, method_name = NA) {
  formula_str <- paste("side ~", paste(feature_set, collapse = " + "))
  formula_obj <- as.formula(formula_str)
  model <- glm(formula_obj, data = train_data, family = "binomial")
  probs <- predict(model, newdata = test_data, type = "response")
  roc_obj <- suppressMessages(roc(test_data$side, probs))
  auc_val <- as.numeric(roc_obj$auc)
  opt_thresh <- coords(roc_obj, x = "best", ret = "threshold", transpose = FALSE)
  preds <- ifelse(probs > opt_thresh$threshold, "Buy", "Sell")
  preds <- factor(preds, levels = c("Sell", "Buy"))
  cm <- confusionMatrix(preds, test_data$side)
  return(data.frame(
    subset = paste(feature_set, collapse = " + "),
    num_vars = length(feature_set),
    aic = AIC(model),
    bic = BIC(model),
    accuracy = cm$overall["Accuracy"],
    balanced_accuracy = cm$byClass["Balanced Accuracy"],
    roc_auc = auc_val,
    stringsAsFactors = FALSE
  ))
}
best.subset.results <- data.frame(
  subset = character(),
  num_vars = integer(),
  aic = numeric(),
  bic = numeric(),
  accuracy = numeric(),
  balanced_accuracy = numeric(),
  roc_auc = numeric(),
  stringsAsFactors = FALSE
)
# Exhaustive subset search using combinations
for (k in 1:length(classification.predictors)) {
  combos <- combinations(
    n = length(classification.predictors), r = k,
    v = classification.predictors
```

```
  )
  for (i in 1:nrow(combos)) {
    vars <- combos[i, ]
    result_row <- evaluate_model(vars)
    best.subset.results <- rbind(best.subset.results, result_row)
  }
}

best_by_aic <- best.subset.results %>%
  arrange(aic) %>%
  slice(1)
best_by_bic <- best.subset.results %>%
  arrange(bic) %>%
  slice(1)
best_by_balacc <- best.subset.results %>%
  arrange(desc(balanced_accuracy)) %>%
  slice(1)
best_by_auc <- best.subset.results %>%
  arrange(desc(roc_auc)) %>%
  slice(1)

print_model_summary <- function(title, row) {
  cat(paste0(title, "\n"))
  cat("  Formula: side ~", row$subset, "\n")
  cat("  Num Vars:", row$num_vars, "\n")
  cat("  Balanced Accuracy:", round(row$balanced_accuracy, 4), "\n")
  cat("  Accuracy:", round(row$accuracy, 4), "\n")
  cat("  AIC:", round(row$aic, 2), "\n")
  cat("  BIC:", round(row$bic, 2), "\n")
  cat("  ROC AUC:", round(row$roc_auc, 4), "\n\n")
}

print_model_summary("Best Subset Model by AIC", best_by_aic)
print_model_summary("Best Subset Model by BIC", best_by_bic)
print_model_summary("Best Subset Model by Balanced Accuracy", best_by_balacc)
print_model_summary("Best Subset Model by ROC AUC", best_by_auc)

x_logistic <- as.matrix(train_data[, classification.predictors])
y_logistic <- train_data$side

lasso_logistic <- cv.glmnet(
  x = x_logistic, y = y_logistic,
  family = "binomial", alpha = 1, nfolds = 10
)
best_lambda_logistic <- lasso_logistic$lambda.min
lasso_coef <- coef(lasso_logistic, s = best_lambda_logistic)
selected_lasso_features <- rownames(lasso_coef)[which(lasso_coef != 0)]
selected_lasso_features <- setdiff(selected_lasso_features, "(Intercept)")
lasso_summary <- evaluate_model(selected_lasso_features, "LASSO")
print_model_summary("LASSO Feature Selection", lasso_summary)

rfe_ctrl <- rfeControl(functions = caretFuncs, method = "cv", number = 10)
rfe_model <- suppressWarnings(
```

```
  rfe(
    x = train_data[, classification.predictors],
    y = train_data$side,
    sizes = 1:length(classification.predictors),
    rfeControl = rfe_ctrl,
    method = "glm",
    family = "binomial"
  )
)
selected_rfe_features <- predictors(rfe_model)
rfe_summary <- evaluate_model(selected_rfe_features, "RFE")
print_model_summary("RFE Feature Selection", rfe_summary)

classification.final.feature.set <- c(
  "deviation_from_mean_balance_lag1_diff",
  "deviation_from_mean_mid_price_lag1",
  "trade_pnl_lag1",
  "volatility_lag1",
  "fill_prc",
  "fill_qty"
)

# Cross-validated logistic regression model using caret
logit_ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)
logit_model_cv <- train(
  as.formula(paste("side ~", paste(classification.final.feature.set, collapse = " + "))),
  data = train_data,
  method = "glm",
  family = binomial,
  metric = "Accuracy",
  trControl = logit_ctrl
)

logit_test_probs <- predict(logit_model_cv, newdata = test_data, type = "prob")[, "Buy"]
roc_logit <- roc(response = test_data$side, predictor = logit_test_probs)
plot(roc_logit, main = "ROC Curve - Logistic Regression")
logit_opt_coords <- coords(
  roc_logit,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)
logit_optimal_threshold <- logit_opt_coords$threshold
logit_test_preds <- ifelse(logit_test_probs > logit_optimal_threshold, "Buy", "Sell")
logit_test_preds <- factor(logit_test_preds, levels = c("Sell", "Buy"))
cm_logit <- confusionMatrix(logit_test_preds, test_data$side,
  positive = "Buy", mode = "everything"
)
```

```
cm_logit

coef(summary(logit_model_cv$finalModel))
logit_varimp <- varImp(logit_model_cv)
plot(logit_varimp, main = "Variable Importance - Logistic Regression")

gbm_grid <- expand.grid(
  n.trees = seq(100, 1000, by = 100),
  interaction.depth = c(1, 3, 5),
  shrinkage = c(0.01, 0.05, 0.1),
  n.minobsinnode = c(5, 10)
)
gbm_ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  verboseIter = FALSE
)
gbm_caret_model <- train(
  x = train_data[, classification.final.feature.set],
  y = train_data$side,
  method = "gbm",
  trControl = gbm_ctrl,
  tuneGrid = gbm_grid,
  metric = "ROC",
  verbose = FALSE
)
gbm_caret_model$bestTune

gbm_probs_test <- predict(gbm_caret_model,
  newdata = test_data[, classification.final.feature.set],
  type = "prob"
)[, "Buy"]
roc_gbm <- roc(response = test_data$side, predictor = gbm_probs_test)
plot(roc_gbm, main = "ROC Curve - GBM")
opt_coords_gbm <- coords(
  roc_gbm,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)
optimal_threshold_gbm <- opt_coords_gbm$threshold
gbm_preds_class <- ifelse(gbm_probs_test > optimal_threshold_gbm, "Buy", "Sell")
gbm_preds_class <- factor(gbm_preds_class, levels = c("Sell", "Buy"))
cm_gbm <- confusionMatrix(gbm_preds_class, test_data$side,
  positive = "Buy", mode = "everything"
)
cm_gbm

gbm_varimp <- varImp(gbm_caret_model)
plot(gbm_varimp, main = "Variable Importance - Gradient Boosting Modelling")
```

```
rf_grid <- expand.grid(mtry = 1:length(classification.final.feature.set))
rf_ctrl <- trainControl(
  method = "cv", # choose CV instead of OOB for accuracy
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  verboseIter = FALSE
)
rf_model_caret <- train(
  x = train_data[, classification.final.feature.set],
  y = train_data$side,
  method = "rf",
  metric = "ROC",
  tuneGrid = rf_grid,
  trControl = rf_ctrl
)
rf_model_caret$bestTune

rf_probs <- predict(rf_model_caret,
  newdata = test_data[, classification.final.feature.set],
  type = "prob"
)[, "Buy"]

roc_rf <- roc(response = test_data$side, predictor = rf_probs)
plot(roc_rf, main = "ROC Curve - Random Forest")

opt_coords_rf <- coords(
  roc_rf,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)

optimal_threshold_rf <- opt_coords_rf$threshold

rf_preds_class <- ifelse(rf_probs > optimal_threshold_rf, "Buy", "Sell")
rf_preds_class <- factor(rf_preds_class, levels = c("Sell", "Buy"))

cm_rf <- confusionMatrix(rf_preds_class, test_data$side,
  positive = "Buy", mode = "everything"
)
cm_rf

rf_varimp <- varImp(rf_model_caret)
plot(rf_varimp, main = "Variable Importance - Random Forest")

svm_grid <- expand.grid(
  sigma = c(0.01, 0.1, 1), # RBF kernel width
  C     = c(0.1, 1, 10) # Regularization cost
)
svm_ctrl <- trainControl(
  method = "cv",
  number = 5,
```

```
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  verboseIter = FALSE
)
svm_model_caret <- train(
  x = train_data[, classification.final.feature.set],
  y = train_data$side,
  method = "svmRadial",
  metric = "ROC",
  tuneGrid = svm_grid,
  trControl = svm_ctrl
)
svm_model_caret$bestTune

svm_probs_test <- predict(
  svm_model_caret,
  newdata = test_data[, classification.final.feature.set],
  type = "prob"
)[, "Buy"]
roc_svm <- roc(response = test_data$side, predictor = svm_probs_test)
plot(roc_svm, main = "ROC Curve - SVM")
# Identify the optimal classification threshold (e.g., Youden's index):
opt_coords_svm <- coords(
  roc_svm,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)
optimal_threshold_svm <- opt_coords_svm$threshold
svm_preds_class <- ifelse(svm_probs_test > optimal_threshold_svm, "Buy", "Sell")
svm_preds_class <- factor(svm_preds_class, levels = c("Sell", "Buy"))
cm_svm <- confusionMatrix(svm_preds_class, test_data$side,
  positive = "Buy", mode = "everything"
)
cm_svm

svm_varimp <- varImp(svm_model_caret)
plot(svm_varimp, main = "Variable Importance - SVM")

# Define hyperparameter grid for KNN:
knn_grid <- expand.grid(k = seq(1, 15, by = 2))
knn_ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  verboseIter = FALSE
)
knn_model_caret <- train(
  x = train_data[, classification.final.feature.set],
  y = train_data$side,
  method = "knn",
  metric = "ROC",
```

```
  tuneGrid = knn_grid,
  trControl = knn_ctrl,
  preProcess = c("center", "scale") # standardize features for distance-based methods
)
knn_model_caret$bestTune

knn_probs_test <- predict(
  knn_model_caret,
  newdata = test_data[, classification.final.feature.set],
  type = "prob"
)[, "Buy"]
roc_knn <- roc(response = test_data$side, predictor = knn_probs_test)
plot(roc_knn, main = "ROC Curve - KNN")
opt_coords_knn <- coords(
  roc_knn,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)
optimal_threshold_knn <- opt_coords_knn$threshold
knn_preds_class <- ifelse(knn_probs_test > optimal_threshold_knn, "Buy", "Sell")
knn_preds_class <- factor(knn_preds_class, levels = c("Sell", "Buy"))
cm_knn <- confusionMatrix(knn_preds_class, test_data$side,
  positive = "Buy", mode = "everything"
)
cm_knn

knn_varimp <- varImp(knn_model_caret)
plot(knn_varimp, main = "Variable Importance - KNN")

h2o.init(nthreads = -1, max_mem_size = "4G")
train_h2o <- as.h2o(train_data[, c(classification.final.feature.set, "side")])
test_h2o <- as.h2o(test_data[, c(classification.final.feature.set, "side")])

train_h2o$side <- as.factor(train_h2o$side)
test_h2o$side <- as.factor(test_h2o$side)

x <- classification.final.feature.set
y <- "side"

aml <- h2o.automl(
  x = x,
  y = y,
  training_frame = train_h2o,
  max_models = 10,
  seed = 123,
  sort_metric = "auc"
)
lb <- aml@leaderboard
head(lb)

test_h2o <- as.h2o(test_data[, c(classification.final.feature.set, "side")])
aml_probs <- as.vector(h2o.predict(aml@leader, test_h2o)[, "Buy"])
```

```r
roc_aml <- roc(response = test_data$side, predictor = aml_probs)
plot(roc_aml, main = "ROC Curve - H2O AutoML")
opt_coords_aml <- coords(
  roc_aml,
  x = "best",
  ret = c("threshold", "sensitivity", "specificity", "accuracy"),
  transpose = FALSE
)
optimal_threshold_aml <- opt_coords_aml$threshold
aml_preds_class <- ifelse(aml_probs > optimal_threshold_aml, "Buy", "Sell")
aml_preds_class <- factor(aml_preds_class, levels = c("Sell", "Buy"))
cm_aml <- confusionMatrix(aml_preds_class, test_data$side,
  positive = "Buy", mode = "everything"
)
cm_aml

h2o.varimp_plot(aml@leader)

model_names <- c("Logistic", "GBM", "Random Forest", "SVM", "KNN", "H2O AutoML")
balanced_accuracy_values <- c(
  cm_logit$byClass["Balanced Accuracy"],
  cm_gbm$byClass["Balanced Accuracy"],
  cm_rf$byClass["Balanced Accuracy"],
  cm_svm$byClass["Balanced Accuracy"],
  cm_knn$byClass["Balanced Accuracy"],
  cm_aml$byClass["Balanced Accuracy"]
)
f1_values <- c(
  cm_logit$byClass["F1"],
  cm_gbm$byClass["F1"],
  cm_rf$byClass["F1"],
  cm_svm$byClass["F1"],
  cm_knn$byClass["F1"],
  cm_aml$byClass["F1"]
)
kappa_values <- c(
  cm_logit$overall["Kappa"],
  cm_gbm$overall["Kappa"],
  cm_rf$overall["Kappa"],
  cm_svm$overall["Kappa"],
  cm_knn$overall["Kappa"],
  cm_aml$overall["Kappa"]
)
class_balance_values <- c(
  abs(cm_logit$byClass["Sensitivity"] - cm_logit$byClass["Specificity"]),
  abs(cm_gbm$byClass["Sensitivity"] - cm_gbm$byClass["Specificity"]),
  abs(cm_rf$byClass["Sensitivity"] - cm_rf$byClass["Specificity"]),
  abs(cm_svm$byClass["Sensitivity"] - cm_svm$byClass["Specificity"]),
  abs(cm_knn$byClass["Sensitivity"] - cm_knn$byClass["Specificity"]),
  abs(cm_aml$byClass["Sensitivity"] - cm_aml$byClass["Specificity"])
)
model_metrics <- data.frame(
  Model = model_names,
```

```
  BalancedAccuracy = balanced_accuracy_values,
  Kappa = kappa_values,
  F1 = f1_values,
  ClassificationBias = class_balance_values
)
model_metrics
metrics_long <- melt(model_metrics, id.vars = "Model")
ggplot(metrics_long, aes(x = Model, y = value, fill = variable)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.9)) +
  geom_text(aes(label = round(value, 3)),
    position = position_dodge(width = 0.9),
    vjust = -0.25, size = 3
  ) +
  labs(
    title = "Classification Model Comparison",
    y = "Metric Value", x = "Model", fill = "Metric"
  ) +
  theme_minimal()

logit_varimp_df <- varImp(logit_model_cv)$importance %>%
  mutate(Feature = rownames(.), Model = "Logistic") %>%
  rename(Importance = Overall)
gbm_varimp_df <- varImp(gbm_caret_model)$importance %>%
  mutate(Feature = rownames(.), Model = "GBM") %>%
  rename(Importance = Overall)
rf_varimp_df <- varImp(rf_model_caret)$importance %>%
  mutate(Feature = rownames(.), Model = "Random Forest") %>%
  rename(Importance = Overall)
svm_raw <- varImp(svm_model_caret)$importance
svm_varimp_df <- svm_raw %>%
  mutate(
    Feature = rownames(.),
    Importance = rowMeans(select(., Sell, Buy))
  ) %>%
  select(Feature, Importance) %>%
  mutate(Model = "SVM")
# averaging across Sell and Buy classes
knn_raw <- varImp(knn_model_caret)$importance
knn_varimp_df <- knn_raw %>%
  mutate(
    Feature = rownames(.),
    Importance = rowMeans(select(., Sell, Buy))
  ) %>%
  select(Feature, Importance) %>%
  mutate(Model = "KNN")
h2o_varimp_df <- as.data.frame(h2o.varimp(aml@leader)) %>%
  mutate(
    Feature = variable,
    Importance = as.numeric(relative_importance),
    Model = "H2O AutoML"
  ) %>%
  select(Feature, Importance, Model)
varimp_combined <- bind_rows(
```

```
  logit_varimp_df,
  gbm_varimp_df,
  rf_varimp_df,
  svm_varimp_df,
  knn_varimp_df,
  h2o_varimp_df
)
ggplot(varimp_combined, aes(x = reorder(Feature, -Importance), y = Importance, fill = Model)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8)) +
  labs(
    title = "Raw Variable Importance Across Models",
    x = "Feature", y = "Raw Importance",
    fill = "Model"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 70, hjust = 1))
```

Appendix C: Trade Impact

Code used for testing whether trades influence post-trade price movements using permutation tests.

```
market_data <- market_data %>%
  mutate(mid_price = (ask_prc + bid_prc) / 2)
get_closest_mid_price <- function(time, market_data) {
  closest_idx <- which.min(abs(market_data$timestamp - time))
  market_data$mid_price[closest_idx]
}
fill_data <- fills_data %>%
  rowwise() %>%
  mutate(
    fill_mid_price = get_closest_mid_price(timestamp, market_data),
    mid_price_1min = get_closest_mid_price(timestamp + 60, market_data),
    price_change   = mid_price_1min - fill_mid_price
  ) %>%
  ungroup() %>%
  filter(!is.na(price_change)) # remove rows with missing data
maker_data <- fill_data %>% filter(liquidity == "Maker")
taker_data <- fill_data %>% filter(liquidity == "Taker")

# H0: mean(x) = 0 HA: mean(x) != 0 (two-sided)
permutation_test <- function(x, R = 1e4, conf = 0.95) {
  n <- length(x)
  obs_stat <- mean(x)
  # permutation distribution under H0
  perm_stats <- replicate(R, mean(sample(c(-1, 1), n, replace = TRUE) * x))
  ## two-sided p-value
  p_val <- 2 * min(
    (sum(perm_stats >= obs_stat) + 1) / (R + 1),
    (sum(perm_stats <= obs_stat) + 1) / (R + 1)
  )
  # bootstrap CI for the mean
  boot_stats <- replicate(R, mean(sample(x, n, replace = TRUE)))
  alpha <- 1 - conf
```

```
  ci <- quantile(boot_stats, c(alpha / 2, 1 - alpha / 2))
  list(
    obs_stat = obs_stat,
    p_val = p_val,
    boot_95CI = ci
  )
}

maker_pricechange_res <- permutation_test(maker_data$price_change)
cat("=== Maker price_change ===\n")
cat("Observed Mean:        ", maker_pricechange_res$obs_stat, "\n")
cat("Two-sided p-value:    ", maker_pricechange_res$p_val, "\n")
cat("95% Permutation CI:   ", maker_pricechange_res$perm_95CI, "\n")

taker_pricechange_res <- permutation_test(taker_data$price_change)
cat("\n=== Taker price_change ===\n")
cat("Observed Mean:        ", taker_pricechange_res$obs_stat, "\n")
cat("Two-sided p-value:    ", taker_pricechange_res$p_val, "\n")
cat("95% Permutation CI:   ", taker_pricechange_res$perm_95CI, "\n")
cat("\n")
cat("Maker count: ", nrow(merged_data[merged_data$liquidity == "Maker", ]))
cat("\n")
cat("Taker count: ", nrow(merged_data[merged_data$liquidity == "Taker", ]))

maker_buy_data <- subset(maker_data, side == "B")
maker_sell_data <- subset(maker_data, side == "S")
maker_buy_res <- permutation_test(maker_buy_data$price_change)
maker_sell_res <- permutation_test(maker_sell_data$price_change)

cat("=== Maker Buy Trades ===\n")
cat("Observed Mean:        ", maker_buy_res$obs_stat, "\n")
cat("Two-sided p-value:    ", maker_buy_res$p_val, "\n")
cat("95% Permutation CI:   ", maker_buy_res$boot_95CI, "\n\n")

cat("=== Maker Sell Trades ===\n")
cat("Observed Mean:        ", maker_sell_res$obs_stat, "\n")
cat("Two-sided p-value:    ", maker_sell_res$p_val, "\n")
cat("95% Permutation CI:   ", maker_sell_res$boot_95CI, "\n")
```