

CS971: AI for Finance Assignment 2

Stewart Macfarlane, Vladimir Lenkov, Alvee Kabir

11-04-2025

Project Background

Asset Selection

The initial assets were gathered using the S&P 500 index, a stock market index that tracks the performance of 500 of the largest trading companies in the United States. In addition to having an extensive collection of assets, this index represents a wide range of sectors including but not limited to technology, healthcare and finance. This serves as a solid foundation for selecting a significant asset for the project.

```
assets <- tq_index("SP500") # Load 500 assets from S&P
```

Furthermore, the daily returns for each asset are retrieved to calculate the Sharpe ratio.

```
load_daily_returns <- function(asset_symbols, startDate, endDate) { removed_assets <- c()
  assets_train <- lapply(asset_symbols, function(sym) {
    tryCatch(dailyReturn(getSymbols(sym, from = startDate, to = endDate, auto.assign = FALSE)),
      error = function(e) {removed_assets <- append(removed_assets, sym); NULL}))
  asset_symbols <- setdiff(asset_symbols, removed_assets)
  df <- setNames(do.call(merge, c(assets_train, all = T)), asset_symbols)
  df <- na.omit(df); df <- df[, colSums(is.na(df)) < nrow(df)]; return(df)}
```

The start and end date for the period to be used to make next-day predictions has been set to two months. This is so that enough data is present to reflect vital patterns to make predictions, however, not a long enough time period whereby the large quantity of historic data will negatively skew results.

```
asset_symbols <- assets$symbol; startDate <- "2024-08-01"; endDate <- "2024-12-31"
df <- load_daily_returns(asset_symbols, startDate, endDate)
```

```
calc_sharpe_ratio <- function(returns, rf_rate) {mean_return <- mean(returns); risk <- sd(returns)
  sharpe_ratio <- ((mean_return - rf_rate) / risk) * sqrt(252); return(sharpe_ratio)}
```

The performance of all 500 assets is evaluated and compared to one another based on their Sharpe ratios. The Sharpe ratio serves as a valuable tool for measuring investment prospects for a specific asset as it enables the comparison of the expected return for the level of risk being taken (risk-adjusted return). In this case, a risk-free rate is dynamically retrieved and used within the Sharpe ratio calculation for each asset.

$$S_a = \frac{E[R_a - R_b]}{\sigma_a}$$

Where : S_a = Sharpe Ratio E = Expected Return

R_a = Asset Return R_b = Risk Free Rate σ_a = Asset Risk

```
rf_rate <- as.numeric(last(getSymbols("DGS3M0", src = "FRED", auto.assign = FALSE)))/100 /252
best_res <- calc_sharpe_ratio(df[, 1], rf_rate); best_asset <- NULL
for (col in colnames(df)) { curr_sharpe <- calc_sharpe_ratio(df[, col], rf_rate)
  if (curr_sharpe > best_res) { best_res <- curr_sharpe; best_asset <- col}}
```

Once all assets have been compared, the best-performing asset is selected to be used to make next-day predictions in alignment with a comprehensive trading rule. All relevant data is then retrieved, this includes opening, high, low and closing prices.

```
best_asset_data <- getSymbols(best_asset, from = startDate, to = endDate, auto.assign = FALSE)
```

Data Preprocessing

```
best_asset_data$RSI = rsi; best_asset_data$MACD = macd
best_asset_data$Volume_MA = volume_ma; best_asset_data = na.omit(best_asset_data)
```

```
data <- data.frame(best_asset_data[,1:9])
min_max_normalize <- function(x) {(x - min(x)) / (max(x) - min(x))}
data_scaled <- as.data.frame(lapply(data, min_max_normalize))
```

```
train_test_split <- function(asset, seq_length, target_feature, test_size = 0.2) {
  asset_matrix <- as.matrix(asset)
  num_seq <- nrow(asset_matrix) - seq_length + 1; num_features <- ncol(asset_matrix)
  seq_data <- array(dim = c(num_seq, seq_length, num_features))
  for (index in 1:(nrow(asset_matrix) - seq_length + 1)) {
    seq_data[index, , ] <- asset_matrix[index:(index + seq_length - 1), ]
  }
  test_set_size <- round(test_size * nrow(seq_data)); train_set_size <- nrow(seq_data) - test_set_size
  x_train <- seq_data[1:train_set_size, 1:(seq_length - 1), , drop = FALSE]
  y_train <- seq_data[1:train_set_size, seq_length, target_feature, drop = FALSE]
  x_test <- seq_data[(train_set_size + 1):nrow(seq_data), 1:(seq_length - 1), , drop = FALSE]
  y_test <- seq_data[(train_set_size + 1):nrow(seq_data), seq_length, target_feature, drop = FALSE]
  return(list(x_train = x_train, y_train = y_train, x_test = x_test, y_test = y_test))}
```

```
open <- paste(best_asset, "Open", sep = "."); close <- paste(best_asset, "Close", sep = ".")
high <- paste(best_asset, "High", sep = "."); low <- paste(best_asset, "Low", sep = ".")
rsi = "RSI"; macd = "MACD"; volume_ma = "Volume_MA"; seq_length <- 8
features <- data_scaled[, c(open, high, low, close, macd, volume_ma)]
split_data <- train_test_split(features, seq_length, target_feature=4)
x_train <- split_data$x_train; y_train <- split_data$y_train
x_test <- split_data$x_test; y_test <- split_data$y_test
```

```
split_validation <- function(x, y, valid_prop = 0.2) { total <- dim(x)[1]
  valid_size <- round(valid_prop * total); train_size <- total - valid_size
  x_train_tune <- x[1:train_size, , , drop = FALSE]
  x_val <- x[(train_size + 1):total, , , drop = FALSE]; y <- as.matrix(y)
```

```

y_train_tune <- y[1:train_size, , drop = FALSE]
y_val <- y[(train_size + 1):total, , drop = FALSE]
return(list(x_train_tune = x_train_tune, y_train_tune = y_train_tune,
           x_val = x_val, y_val = y_val))}
split_data <- split_validation(x_train, y_train, valid_prop = 0.2)
x_train_tune <- split_data$x_train_tune; y_train_tune <- split_data$y_train_tune
x_val <- split_data$x_val; y_val <- split_data$y_val

```

Optimising LSTM Parameters

The LSTM parameters are optimised using two techniques: grid search and genetic algorithms. This was done to compare the results from utilising traditional versus evolutionary approaches and conclude the pros and cons of each. Furthermore, the optimised parameters identified from this process are used by the LSTM to make predictions in conjunction with the proposed trading rule.

```

tune_lstm <- function(learningrate, hidden_dim, num_layers, numepochs, batch_size) {
  model <- trainr( Y = y_train_tune, X = x_train_tune, learningrate = learningrate,
                  hidden_dim = hidden_dim, num_layers = num_layers, numepochs = numepochs,
                  network_type = "lstm", seq_to_seq_unsync = TRUE, batch_size = batch_size)
  predictions <- predictr(model, x_val)
  mse <- mean((predictions - y_val)^2, na.rm = TRUE); return(mse)}

```

Grid Search

Grid search is a traditional approach to identifying optimal hyperparameter values for machine learning models. In this approach, the key hyperparameters to be tested are listed inside a vector, which the algorithm then systematically iterates over each combination and records the result. In this case, the mean squared error (MSE) is used on validation data to determine the current performance.

```

lr_vals <- c(0.001, 0.005, 0.01); hd_vals <- c(8, 16, 32, 64, 128) # Grid parameters
nl_vals <- c(1, 2, 3); ne_vals <- c(50, 100, 150, 200); bs_vals <- c(8, 16, 32, 64)

```

```

run_grid_search <- function(lr_vals, hd_vals, nl_vals, ne_vals, bs_vals){
  for (lr in lr_vals) {for (hd in hd_vals) {for (nl in nl_vals) {
    for (ne in ne_vals) {for (bs in bs_vals) { current_mse <- tune_lstm(lr,hd,nl,ne,bs)
      log_results(lr, hd, nl, ne, bs, current_mse)}}}}}
#run_grid_search(lr_vals, hd_vals, nl_vals, ne_vals, bs_vals)
#best_params_GS <- results[which.min(results$mse), ]

```

Genetic Algorithm

A genetic algorithm is an evolutionary process that mimics natural selection and genetics. This algorithm has been used to identify optimal hyperparameters within specified ranges (lower and upper). This implementation has a maximum of 100 iterations and will stop executing if the fitness does not improve after 20 iterations. The fitness is determined using the fitness function which evaluates performance against the MSE value.

```

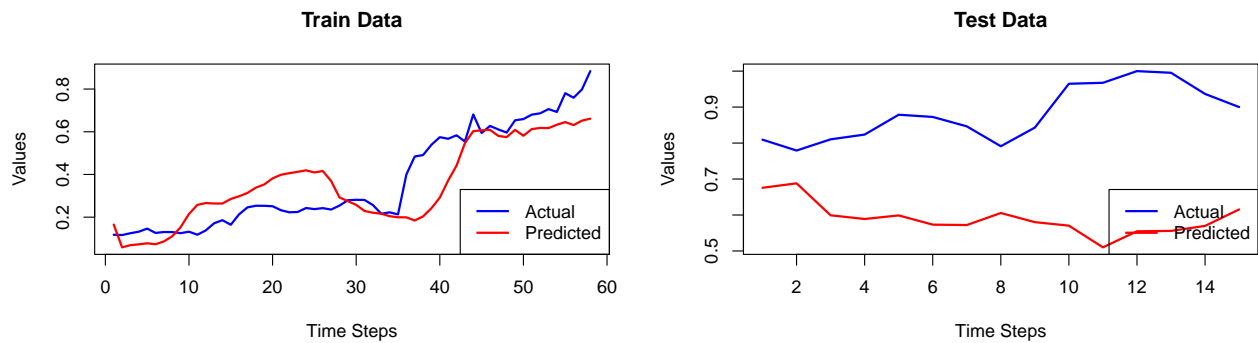
fitness_function <- function(params) {
  lr <- params[1]; hd <- round(params[2]); nl <- round(params[3])
  ne <- round(params[4]); bs <- round(params[5])
  mse <- tune_lstm(lr, hd, nl, ne, bs); return(-mse)}
run_ga <- function(){ ga_result <- ga(type = "real-valued",fitness = fitness_function,
  lower = c(0.0001, 8, 1, 50, 8),upper = c(0.01, 128, 3, 200, 64),
  popSize = 20,maxiter = 100,run = 20); return(ga_result)}
#ga_result <- run_ga(); best_params_GA <- ga_result@solution

```

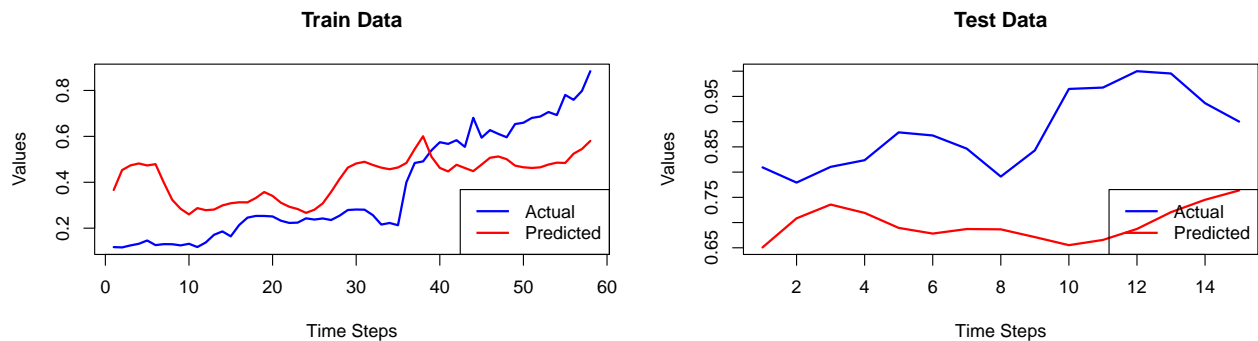
Optimisation Comparisons

Through experimenting with both of the above approaches key benefits and downfalls of each have been identified. First, Grid search is strictly limited to searching the specified hyperparameters whereas the GA solution can navigate the search space more effectively only being restricted to lower and upper bounds. Furthermore, both algorithms are computationally expensive, although, genetic algorithms have an edge as they can effectively terminate execution if the performance has not improved over a specified number of iterations, whereas grid search must evaluate all combinations. Finally, this difference between the two approaches is what sets them apart as a GA can get stuck in a local maximum and never converge to the optimal solution, on the other hand, grid search will evaluate all provided combinations guaranteeing the most optimal from the provided is found. Overall, both methods gain a similar performance using MSE.

Grid Search Optimised Paramaters



Genetic Algorithm Optimised Paramaters



Trading

```

starting_funds = 10000
investment = starting_funds
shares = 0

inverse_scale <- function(scaled_value, unscaled_min, unscaled_max) {
  scaled_value * (unscaled_max - unscaled_min) + unscaled_min
}

predictions_scaled = predictr(lstm_GS, x_test)
unscaled_min_close = min(data[, paste(best_asset, "Close", sep = ".")])
unscaled_max_close = max(data[, paste(best_asset, "Close", sep = ".")])

predictions_unscaled = inverse_scale(predictions_scaled, unscaled_min_close, unscaled_max_close)
actual_unscaled = inverse_scale(y_test, unscaled_min_close, unscaled_max_close)
#predictions_unscaled
#actual_unscaled

```

```

trading_rule = data.frame(
  Date = index(tail(best_asset_data, nrow(y_test))),
  actual_price = rep(NA, nrow(y_test)),
  predicted_price = rep(NA, nrow(y_test)),
  action = character(nrow(y_test)),
  asset_value = numeric(nrow(y_test)),
  shares_held = numeric(nrow(y_test))
)

trading_rule$asset_value[1] = investment
trading_rule$shares_held[1] = shares
trading_rule$actual_price = actual_unscaled
trading_rule$predicted_price = predictions_unscaled

threshold_buy = 0.05
threshold_sell = -0.05

for(i in 1:nrow(trading_rule)){
  if(i>1){
    investment = trading_rule$asset_value[i-1]
    shares = trading_rule$shares_held[i-1]
  }
  current_price = trading_rule$actual_price[i]
  predicted_price = trading_rule$predicted_price[i]
  action = "HOLD"

  if(!is.na(predicted_price) && !is.na(current_price)){
    predicted_change_percentage = (predicted_price - current_price) / current_price
    if (predicted_change_percentage > threshold_buy && investment > 0) {
      action = "BUY"
      buy_quantity = floor(investment / current_price)
      shares = shares + buy_quantity
      investment = investment - (buy_quantity * current_price)
    } else if (predicted_change_percentage < threshold_sell && shares > 0) {
      action = "SELL"
      sell_value = shares * current_price
    }
  }
}

```

```

        investment = investment + sell_value
        shares = 0
    }
}

trading_rule$action[i] = action
trading_rule$asset_value[i] = investment + (shares * current_price)
trading_rule$shares_held[i] = shares
}

```

```

trading_rule = data.frame(
  Date = index(tail(best_asset_data, nrow(y_test))),
  actual_price = rep(NA, nrow(y_test)),
  predicted_price = rep(NA, nrow(y_test)),
  action = character(nrow(y_test)),
  asset_value = numeric(nrow(y_test)),
  shares_held = numeric(nrow(y_test))
)

trading_rule$asset_value[1] = investment
trading_rule$shares_held[1] = shares
trading_rule$actual_price = actual_unscaled
trading_rule$predicted_price = predictions_unscaled

threshold_buy = 0.01
threshold_sell = -0.01

next_day_action = character(nrow(trading_rule))
next_day_action[1] = "HOLD"

for(i in 1:(nrow(trading_rule) - 1)){
  current_price = trading_rule$actual_price[i]
  predicted_price = trading_rule$predicted_price[i]
  action = "HOLD"

  if(!is.na(predicted_price) && !is.na(current_price)){
    predicted_change_percentage = (predicted_price - current_price) / current_price
    if(predicted_change_percentage > threshold_buy){
      action = "BUY"
    } else if(predicted_change_percentage < threshold_sell){
      action = "SELL"
    } else if(predicted_change_percentage < threshold_buy && predicted_change_percentage > threshold_sell){
      action = "HOLD"
    }
  }
  next_day_action[i + 1] = action
}

for(i in 1:nrow(trading_rule)){
  if(i > 1){
    investment = trading_rule$asset_value[i-1]
    shares = trading_rule$shares_held[i-1]
  }
}

```

```

trade_action = next_day_action[i]
current_price = trading_rule$actual_price[i]

if(trade_action == "BUY" && investment > 0){
  buy_quantity = floor(investment / current_price)
  shares = shares + buy_quantity
  investment = investment - (buy_quantity * current_price)
} else if(trade_action == "SELL" && shares > 0){
  sell_value = shares * current_price
  investment = investment + sell_value
  shares = 0
}

trading_rule$action[i] = trade_action
trading_rule$asset_value[i] = investment + (shares * current_price)
trading_rule$shares_held[i] = shares
}

```

```

final_asset_value = tail(trading_rule$asset_value, 1)
initial_investment = starting_funds
profit_loss = final_asset_value - initial_investment
roi = (profit_loss / initial_investment) * 100

cat("\nFinal Asset Value: $", round(final_asset_value, 2), "\n")

```

```

##
## Final Asset Value: $ 10000

```

```

cat("Profit/Loss: $", round(profit_loss, 2), "\n")

```

```

## Profit/Loss: $ 0

```

```

cat("Return on Investment (ROI): ", round(roi, 2), "%\n")

```

```

## Return on Investment (ROI): 0 %

```

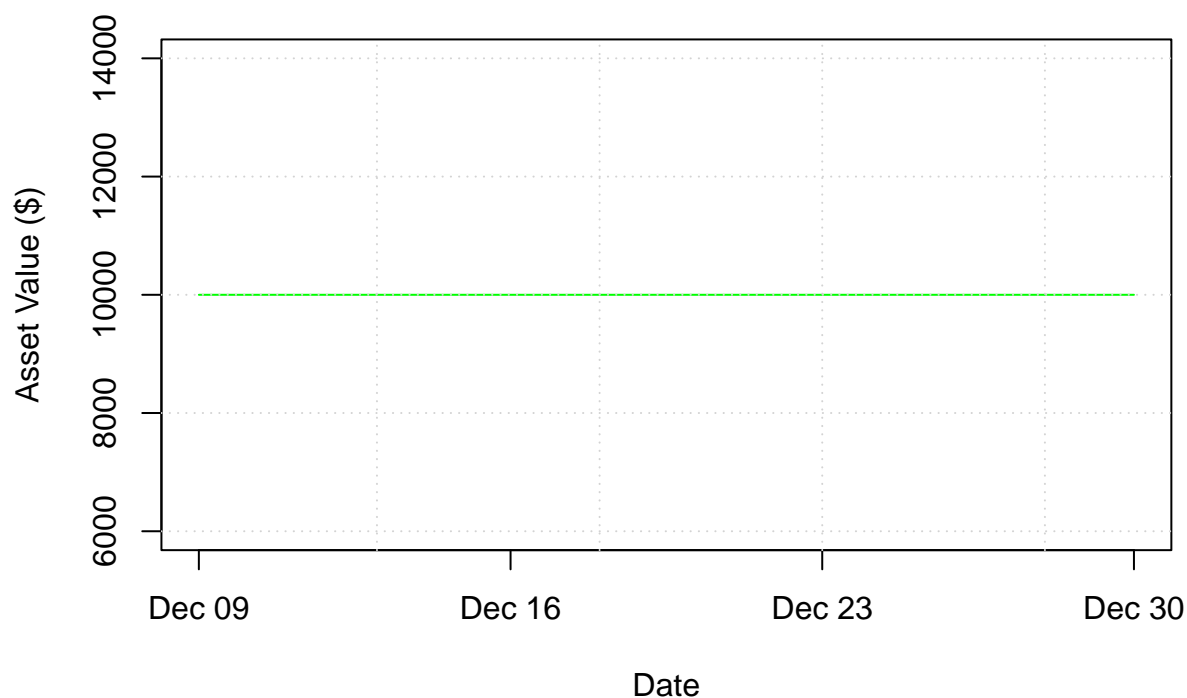
```

plot_trading_simulation = function(trade_log) {
  plot(trading_rule$Date, trading_rule$asset_value, type = "l", col = "green",
        xlab = "Date", ylab = "Asset Value ($)",
        main = "Trading Strategy Performance")
  grid()
}

plot_trading_simulation(trade_log)

```

Trading Strategy Performance



```
print(trading_rule)
```

##	Date	actual_price	predicted_price	action	asset_value	shares_held
## 1	2024-12-09	72.46	65.49831	HOLD	10000	0
## 2	2024-12-10	70.89	66.14440	SELL	10000	0
## 3	2024-12-11	72.51	61.51616	SELL	10000	0
## 4	2024-12-12	73.20	60.98446	SELL	10000	0
## 5	2024-12-13	76.07	61.50168	SELL	10000	0
## 6	2024-12-16	75.75	60.16416	SELL	10000	0
## 7	2024-12-17	74.39	60.10675	SELL	10000	0
## 8	2024-12-18	71.51	61.84878	SELL	10000	0
## 9	2024-12-19	74.21	60.52815	SELL	10000	0
## 10	2024-12-20	80.55	60.01973	SELL	10000	0
## 11	2024-12-23	80.69	56.88198	SELL	10000	0
## 12	2024-12-24	82.38	59.20349	SELL	10000	0
## 13	2024-12-26	82.14	59.26135	SELL	10000	0
## 14	2024-12-27	79.08	59.98507	SELL	10000	0
## 15	2024-12-30	77.18	62.36329	SELL	10000	0

```
#Revised Dual-Indicator Trading Strategy
```

```
threshold_buy <- 0.005           # Predicted change > 0.5%
threshold_sell <- -0.005         # Predicted change < -0.5%
oversold_threshold <- 70         # For a BUY, require RSI < 70
overbought_threshold <- 30      # For a SELL, require RSI > 30
```

```
# Reinitialize simulation variables
```

```
investment_dual <- 10000
shares_dual <- 0
```



```

# Build the trading log for the dual-indicator strategy
trading_rule_dual <- data.frame(
  Date = index(tail(best_asset_data, nrow(y_test))),
  actual_price = as.numeric(actual_unscaled),
  predicted_price = as.numeric(predictions_unscaled),
  RSI = as.numeric(tail(best_asset_data$RSI, nrow(y_test))),
  action = character(nrow(y_test)),
  asset_value = numeric(nrow(y_test)),
  shares_held = numeric(nrow(y_test))
)

trading_rule_dual$asset_value[1] <- investment_dual
trading_rule_dual$shares_held[1] <- shares_dual

# Simulation loop with debug prints for the first few iterations
for (i in 1:nrow(trading_rule_dual)) {
  if (i > 1) {
    investment_dual <- trading_rule_dual$asset_value[i - 1]
    shares_dual <- trading_rule_dual$shares_held[i - 1]
  }
  current_price <- trading_rule_dual$actual_price[i]
  predicted_price <- trading_rule_dual$predicted_price[i]
  current_rsi <- trading_rule_dual$RSI[i]
  action <- "HOLD"

  if (!is.na(predicted_price) && !is.na(current_price) && !is.na(current_rsi)) {
    predicted_change_percentage <- (predicted_price - current_price) / current_price
    if (predicted_change_percentage > threshold_buy && current_rsi < oversold_threshold && investment_dual > 0) {
      action <- "BUY"
      buy_quantity <- floor(investment_dual / current_price)
      shares_dual <- shares_dual + buy_quantity
      investment_dual <- investment_dual - (buy_quantity * current_price)
    } else if (predicted_change_percentage < threshold_sell && current_rsi > overbought_threshold && shares_dual > 0) {
      action <- "SELL"
      sell_value <- shares_dual * current_price
      investment_dual <- investment_dual + sell_value
      shares_dual <- 0
    }
  }

  trading_rule_dual$action[i] <- action
  trading_rule_dual$asset_value[i] <- investment_dual + (shares_dual * current_price)
  trading_rule_dual$shares_held[i] <- shares_dual
}

# Calculate final performance metrics
final_asset_value <- tail(trading_rule_dual$asset_value, 1)
profit_loss <- final_asset_value - 10000
roi <- (profit_loss / 10000) * 100

# Print results
cat("\nFinal Asset Value: $", round(final_asset_value, 2), "\n")

```

```
##
## Final Asset Value: $ 10000

cat("Profit/Loss: $", round(profit_loss, 2), "\n")

## Profit/Loss: $ 0

cat("Return on Investment (ROI):", round(roi, 2), "%\n")

## Return on Investment (ROI): 0 %

# Print the full table
print(trading_rule_dual)
```

```
##      Date actual_price predicted_price      RSI action asset_value
## 1  2024-12-09      72.46      65.49831 69.07233  HOLD      10000
## 2  2024-12-10      70.89      66.14440 65.25666  HOLD      10000
## 3  2024-12-11      72.51      61.51616 67.26606  HOLD      10000
## 4  2024-12-12      73.20      60.98446 68.11199  HOLD      10000
## 5  2024-12-13      76.07      61.50168 71.42037  HOLD      10000
## 6  2024-12-16      75.75      60.16416 70.54158  HOLD      10000
## 7  2024-12-17      74.39      60.10675 66.78071  HOLD      10000
## 8  2024-12-18      71.51      61.84878 59.54137  HOLD      10000
## 9  2024-12-19      74.21      60.52815 63.53262  HOLD      10000
## 10 2024-12-20      80.55      60.01973 70.81358  HOLD      10000
## 11 2024-12-23      80.69      56.88198 70.95150  HOLD      10000
## 12 2024-12-24      82.38      59.20349 72.63272  HOLD      10000
## 13 2024-12-26      82.14      59.26135 71.99547  HOLD      10000
## 14 2024-12-27      79.08      59.98507 64.25471  HOLD      10000
## 15 2024-12-30      77.18      62.36329 59.94499  HOLD      10000
##      shares_held
## 1              0
## 2              0
## 3              0
## 4              0
## 5              0
## 6              0
## 7              0
## 8              0
## 9              0
## 10             0
## 11             0
## 12             0
## 13             0
## 14             0
## 15             0
```

```
# Plot the performance of the dual-indicator trading strategy
plot_dual <- function(trade_log) {
  plot(trade_log$Date, trade_log$asset_value, type = "l", col = "purple",
       xlab = "Date", ylab = "Asset Value ($)"),
       main = "Dual-Indicator Strategy Performance")
}
```

```
    grid()
  }
  plot_dual(trading_rule_dual)
```

