HW-3 Tutorial: Backtesting a Trading Strategy using S&P 500 Industrials Stocks

March 4, 2025

1 Learning Outcomes

After completing this tutorial, you will be able to:

- Load and filter financial data for stocks in the Industrials sector.
- Apply a non-traditional forecasting method (ETS with an AAN model) to predict stock prices.
- Choose stocks with the best forecast performance based on Mean Squared Error (MSE).
- Create trading indicators using the trend (slope) derived from a rolling forecast.
- Develop and apply trading rules for both long and short positions.
- Simulate daily trading under portfolio constraints.
- Evaluate the trading strategy with multiple performance metrics.

2 Homework Requirements

Our backtesting strategy must meet the following criteria:

a) Use a time-series forecasting method (other than regression or auto-regressive functions). We use an ETS model with an AAN specification.

b) Run the backtest over the period from 1/1/2023 to 12/31/2024, while using enough historical data prior to 1/1/2023 for model initialization.

c) Include both long and short positions, trading daily (entering positions at the open and exiting at the close).

d) Filter predictions based on a threshold return level and confidence (we filter by selecting stocks with lower forecast error).

e) Begin with $100,000 in cash and weight investments according to predictability, imposing constraints like maximum trade per position.

f) Evaluate performance with metrics such as: number of long trades, percentage of winning long trades, average long trade return, number of short trades, percentage of winning short trades, average short trade return, Sharpe ratio, cumulative portfolio return, maximum drawdown (percentage and period), and overall percentage of winning trades.

This tutorial explains, step by step, how to build a backtesting framework for a trading strategy focused on Industrials stocks from the S&P 500. Instead of using typical regression or autoregressive methods, we use an ETS model with an AAN specification to generate forecasts. The key idea is to use the slope of the forecasted time series as an indicator of trend direction. Stocks with better forecast performance (lower MSE) are then selected, and based on the signal (positive slope for long, negative for short), trades are simulated daily under strict capital constraints. Finally, we evaluate the strategy using comprehensive performance metrics.

3 Data Preparation and Stock Filtering (Addresses b and e)

In this section, we set up our R environment, load the necessary libraries, and prepare our stock data. We ensure that our data is clean and contains sufficient historical observations (including extra days before 1/1/2023) for initializing our forecasts.

Listing 1: Data Preparation and Stock Filtering

```r
# Load required libraries
library(tidyverse)
library(forecast)
library(rstudioapi)
library(furrr)          # Parallel processing

plan(multisession, workers = 12)

```

```r
9  current_path <- rstudioapi::getActiveDocumentContext()$path
10 setwd(dirname(current_path))
11 rm(list = ls())
12 options(scipen = 999)
13 
14 # Define the training and validation periods
15 train_start <- as_date("2021-01-01")
16 valid_start <- as_date("2023-01-01")
17 valid_end   <- as_date("2024-12-31")
18 
19 stocks <- read_csv("stocks.csv") |>
20   filter(sector == "Industrials" & date >= train_start - 40 & date <=
        valid_end) |>
21   arrange(ticker, date) |>
22   select(ticker:close)
23 
24 # Remove tickers with data issues
25 excluded_tickers <- c("IR", "AXON", "BLDR")
26 stocks <- stocks |> filter(!ticker %in% excluded_tickers)
```

Explanation:

- **Lines 1–5:** We load essential libraries. `tidyverse` provides powerful data manipulation tools, `forecast` lets us perform time-series forecasting with ETS models, `rstudioapi` is used for dynamically setting our working directory, and `furrr` allows for parallel processing.

- **Line 7:** We set up parallel processing with 12 workers, which helps speed up repetitive tasks like forecasting over many stocks.

- **Lines 9–12:** The working directory is set based on the active document's location; we clear the workspace to avoid variable conflicts and adjust number formatting for readability.

- **Lines 15–17:** Here we define our training period starting in 2021 and our backtesting period from January 1, 2023, to December 31, 2024. We include an extra 40 days of data before the training period to properly initialize our rolling window forecasts.

- **Lines 19–22:** We load the stock data from a CSV file, filter to include only stocks in the Industrials sector, arrange the data by ticker and date, and select only the necessary columns.

- **Lines 25-26:** Specific tickers known to have issues (such as duplicate dates or missing data) are removed from the dataset to ensure a smooth analysis.

4 Part 1: Filtering Stocks by Prediction Performance (Addresses a)

This function uses an ETS model with an AAN specification to generate forecasts on a rolling basis. We calculate the Mean Squared Error (MSE) for each stock and select the top 10 stocks with the lowest errors. This ensures that our analysis focuses on the most predictable stocks.

Listing 2: Filtering Stocks Based on Forecast Error

```r
filter_stocks <- function(stocks, top_k) {
  tickers <- stocks$ticker |> unique()  # Extract unique tickers.

  errors <- tickers |> future_map_dbl(function(t) {
    stock <- stocks |> filter(ticker == t & date < valid_start)
    if (nrow(stock) < 20) return(Inf)

    stock$pred <- NA
    window <- 20

    # Rolling window forecasting loop
    for (i in window:nrow(stock)) {
      s <- i - window + 1

      roll_ts <- stock[s:i, ] |>
        select(close) |>
        ts(start = c(1, 1), end = c(1, window), frequency = window)

      model <- ets(roll_ts, model = "AAN")  # Fit an ETS model using the
          AAN specification.
      stock$pred[i] <- as.double(forecast(model, h = 1)$mean)
    }

    print(t)
    mse <- mean((stock$pred - stock$close)^2, na.rm = TRUE)  # Calculate
        the Mean Squared Error (MSE).
    return(mse)
  })

  sorted_by_err <- as_tibble(cbind(tickers, errors)) |> arrange(errors)

  # Select the top-k stocks with the lowest MSE
  best_stocks <- sorted_by_err[1:top_k, 1]

  filtered_stocks <- stocks |>
34      filter(ticker %in% best_stocks$tickers) |>
35      drop_na()
36
37    return(filtered_stocks)
38  }
```

Explanation:

- **Lines 1–2:** The function `filter_stocks` is defined to take the stock data and the number of top stocks (`top_k`) to keep. It starts by extracting all unique tickers.
- **Line 4-5:** For each ticker, using parallel processing, we filter data to include only dates before the backtesting period—this forms our training data.
- **Line 6:** We check if the stock has fewer than 20 observations; if so, we return an infinite error to exclude that stock.
- **Line 8:** A new column `pred` is initialized to hold forecasted closing prices.
- **Line 9:** The rolling window length is set to 20 days.
- **Lines 12–20:** A loop iterates over the training data once the window is full:
  - The starting index for the current window is computed.
  - The closing prices within this window are converted into a time series.
  - An ETS model (with an AAN specification) is fitted to the data.
  - A one-step-ahead forecast is produced and stored in the `pred` column.
- **Line 23:** The ticker symbol is printed to provide progress feedback.
- **Line 24-25:** The MSE between the forecasted and actual closing prices is calculated.
- **Lines 29–37:** All ticker errors are combined, sorted, and the top `top_k` tickers with the lowest MSE are selected. The original dataset is then filtered to include only these tickers, and any rows with missing data are removed.
- AAN stands for:
  - A(Additive Error): This means that the model assumes the errors (residuals) are additive, i.e., the difference between the predicted and actual values is constant across time.

  - A (Additive Trend): This means the trend (the direction in which the data is moving, such as increasing or decreasing) is assumed to be additive, i.e., the trend changes by a constant amount over time.
  - N (No Seasonality): This indicates that the model assumes there is no seasonal component in the data, meaning there are no recurring patterns over a specific time period (like daily, weekly, or yearly cycles).

- Why choose AAN:
  - Noise in the data is constant over time
  - Data has a linear trend over time
  - Data does not show any cyclical or periodic patterns

- What is within ETS(Exponential Smoothing State Space Model):
  - Error (also called residual or noise) – captures the random fluctuations.
  - Trend – captures the long-term direction (increase or decrease) in the data.
  - Seasonality – captures recurring patterns at fixed periods (e.g., daily, monthly, yearly).

- Why choose ETS:
  - Don't need complex transformations or multiple steps to preprocess your data, making it user-friendly for beginners
  - The key feature of ETS models is their exponential weighting, where more recent observations are given higher importance when making predictions.
  - ETS models are particularly well-suited for short-term forecasting, where the recent past is a strong indicator of future behavior.

5 Part 2: Generating Trading Indicators (Addresses c and d)

This function calculates a trend indicator (the slope) from a rolling ETS forecast. The slope indicates the trend direction: a positive slope implies an upward trend (suggesting a long position), while a negative slope implies a downward trend (suggesting a short position). Although the homework mentions filtering based on prediction confidence, we use forecast error from the previous step as our proxy.

Listing 3: Generating Indicators from Rolling ETS Model

```r
1  gen_indicators <- function(stocks, window) {
2    tickers <- stocks$ticker |> unique()
3  
4    tickers |> map_dfr(function(t) {
5      stock <- stocks |> filter(ticker == t)
6      trade_days <- stock |> filter(date >= valid_start) |> nrow()
7  
8      start <- nrow(stock) - trade_days - window + 1
9      end <- nrow(stock)
10     stock <- stock[start:end, ]
11 
12     stock$slope <- 0
13 
14     cat(paste("Generating␣Indicators␣for:", t, "\n"))
15 
16     for (i in window:nrow(stock)) {
17       s <- i - window + 1
18 
19       roll_ts <- stock[s:i, ] |>
20         select(close) |>
21         ts(start = 1, frequency = window)
22 
23       if (length(roll_ts) < window) next
24 
25       model <- ets(roll_ts, model = "AAN")  # Fit an ETS model using the
             AAN specification.
26       stock$slope[i] <- model$par["b"]
27     }
28 
29     return(stock)
30   })
31 }
```

Explanation:

- **Lines 1–2:** The function `gen_indicators` is defined to take in the stock data and a rolling window length, and then extracts all unique tickers.
- **Line 4:** Each ticker's data is processed in turn.
- **Line 5:** Data for the current ticker is filtered from the complete dataset.
- **Line 6:** The number of trading days during the backtesting period is determined.

- **Line 8-9:** The starting and end index is calculated to ensure that the subset of data covers both the rolling window and the backtest period.
- **Line 10:** The dataset is sliced to include only the necessary observations.
- **Line 12:** A new column `slope` is initialized to store the computed trend indicator.
- **Line 14:** A progress message is printed to indicate that indicator generation is underway for the current ticker.
- **Lines 16–25:** A loop iterates over the observations starting when the window is full:
  - The starting index of the current window is calculated.
  - The closing prices for that window are transformed into a time series.
  - If the window is incomplete, the iteration is skipped.
  - An ETS model is fitted, and the slope parameter is extracted and stored.
- **Line 26:** The modified data, now with a slope indicator, is returned for further processing.

6 Part 3: Setting Trading Rules (Addresses c and e)

This function converts the computed slope into trading signals. A positive slope creates a long signal, while a negative slope creates a short signal. Based on these signals, buy and sell prices are determined (enter at open for long, at close for short, and vice versa for exiting), and the returns for each trade are calculated.

Listing 4: Defining Trading Rules and Calculating Returns

```r
set_rules <- function(stocks) {
  tickers <- stocks$ticker |> unique()

  tickers |> map_dfr(function(t) {
    stock <- stocks |> filter(ticker == t)

    stock$long  <- stock$slope > 0   # Long signal when the slope is positive.
    stock$short <- stock$slope < 0   # Short signal when the slope is negative.
    stock$trade <- stock$long | stock$short

    # Determine the buy price based on the trade type
12      stock$buy <- ifelse(stock$long, stock$open,
13                             ifelse(stock$short, stock$close, NA))  # Buy at
                                  open for long, close for short.
14
15      # Determine the sell price based on the trade type
16      stock$sell <- ifelse(stock$long, stock$close,
17                              ifelse(stock$short, stock$open, NA))  # Sell at
                                   close for long, open for short.
18
19      stock$return <- ifelse(stock$trade,
20                                  1 + (stock$sell - stock$buy) / stock$buy, 1)
21
22      return(stock)
23    })
24  }
```

Explanation:

- **Lines 1–2:** The function `set_rules` is defined and begins by extracting unique tickers from the dataset.
- **Line 4:** Each ticker is processed individually.
- **Line 5:** Data for the current ticker is filtered.
- **Line 7** A binary long signal is created if the slope is greater than zero.
- **Line 8:** Similarly, a binary short signal is generated if the slope is less than zero.
- **Line 9:** A combined trade signal is produced that flags any day with either a long or short signal.
- **Line 12-13:** The buy price is determined: if the signal is long, the stock is bought at the open price; if short, it is bought at the close price.
- **Line 16-17:** The sell price is then determined: for long trades, the stock is sold at the close price; for short trades, it is sold at the open price.
- **Line 19-20:** The return is calculated as the percentage change between the sell and buy prices. If no trade occurs, the return is set to 1 (indicating no change).
- **Line 22:** The modified data for the current ticker is returned, now containing the trading signals and trade returns.

7 Part 4: Simulating the Backtest (Addresses e)

This function simulates the trading process over a specified number of days. Starting with $100,000, capital is allocated based on the absolute value of the slope (indicating signal strength) while capping each trade with a maximum amount. The portfolio equity is updated each day, and the overall return percentage is calculated at the end.

Listing 5: Simulating Daily Trades and Calculating Portfolio Return

```r
apply_trades <- function(stocks, days, initial_equity, max_trade) {
  tickers <- stocks$ticker |> unique()
  num_tickers <- length(tickers)

  slopes <- matrix(data = 0, nrow = days, ncol = num_tickers)

  for (t in 1:num_tickers) {
    stock <- stocks |> filter(ticker == tickers[t])
    for (i in 20:days) {
      slopes[i, t] <- abs(stock$slope[i])
    }
  }

  slopes[is.na(slopes)] <- 0
  totals <- rowSums(slopes)

  weights <- slopes / totals
  weights[is.nan(weights)] <- 0
  weights[is.na(weights)] <- 0

  # Initialize matrices to track investment amounts and revenue per day
  invested <- matrix(data = 0, nrow = days, ncol = num_tickers)
  revenue <- matrix(data = 0, nrow = days, ncol = num_tickers)
  end_equity <- initial_equity

  for (period in 20:days) {
    for (t in 1:num_tickers) {
      stock <- stocks |> filter(ticker == tickers[t])

      if (is.na(stock$return[period])) next

      trade_amount <- min(weights[period, t] * end_equity, max_trade)

      invested[period, t] <- trade_amount
      revenue[period, t] <- trade_amount * stock$return[period]

37        end_equity <- end_equity - trade_amount + revenue[period, t]
38      }
39    }
40
41    # Calculate overall return percentage relative to initial equity
42    ret_pct <- (end_equity - initial_equity) / initial_equity * 100
43    return(ret_pct)
44  }
```

Explanation:

- **Lines 1–3:** The function `apply_trades` is defined with parameters for the dataset, number of trading days, initial equity, and maximum trade size. We begin by extracting unique tickers and determining how many there are.
- **Lines 5–10:** A matrix called `slopes` is created to store the absolute slope values for each stock on each trading day. We loop through each ticker and, starting from day 20 (to ensure our rolling window has been fully populated), store the absolute value of the slope.
- **Lines 14–15:** Any missing values in the slopes matrix are replaced with zero, and then the total signal for each day is computed by summing the slopes.
- **Lines 17–19:** Trading weights are calculated by normalizing each stock's slope by the daily total. Any resulting NaN or NA values are replaced with zero to ensure clean calculations.
- **Lines 22–24:** Two matrices, `invested` and `revenue`, are initialized to record the amount invested and the revenue earned per stock per day. The starting portfolio equity is set to the initial cash value.
- **Lines 26–37:** A loop simulates trading for each day (starting from day 20). For each ticker:
  - If there is no valid trade signal, the iteration is skipped.
  - The trade amount is calculated as a fraction of the current equity (proportional to the weight), but capped at the maximum trade amount.
  - The invested amount and the revenue (based on the stock's return) are recorded.
  - The portfolio equity is updated by subtracting the investment and adding the revenue.
- **Lines 42–43:** The overall return percentage is calculated relative to the initial equity and returned.

8 Part 5: Evaluating Strategy Performance (Addresses f)

This function calculates and displays the performance metrics of our trading strategy. It computes the number and percentage of winning long and short trades, average trade returns, daily Sharpe ratio, cumulative portfolio return, maximum drawdown, and overall winning percentage.

Listing 6: Evaluating Performance Metrics

```r
eval_strategy <- function(portfolio) {

  portfolio <- portfolio |> arrange(date)

  portfolio$cumreturn <- cumprod(portfolio$return)
  portfolio$maxreturn <- cummax(portfolio$cumreturn)

  # Define a daily risk-free return (assuming an annual rate of 4%)
  risk_free_return <- 1 + 0.04 / 365

  daily_sharpe <- mean(portfolio$return - risk_free_return, na.rm = TRUE) /
                  sd(portfolio$return, na.rm = TRUE)

  performance_summary <- tibble(
    "Long␣Trades" = sum(portfolio$long, na.rm = TRUE),
    "%␣Winning␣Long␣Trades" = mean(portfolio$return[portfolio$long] > 1, na.rm = TRUE) * 100,
    "Avg␣Long␣Trade␣Return" = mean(portfolio$return[portfolio$long], na.rm = TRUE),

    "Short␣Trades" = sum(portfolio$short, na.rm = TRUE),
    "%␣Winning␣Short␣Trades" = mean(portfolio$return[portfolio$short] > 1, na.rm = TRUE) * 100,
    "Avg␣Short␣Trade␣Return" = mean(portfolio$return[portfolio$short], na.rm = TRUE),

    "Daily␣Sharpe␣Ratio" = daily_sharpe,
    "Cumulative␣Portfolio␣Return" = tail(portfolio$cumreturn, 1),

    "Max␣Drawdown␣%" = min(portfolio$cumreturn / portfolio$maxreturn - 1, na.rm = TRUE) * 100,

    # Compute overall percentage of winning trades
29      "Overall␣%␣Winning␣Trades" = mean(portfolio$return > 1, na.rm = TRUE)
            * 100
30    )
31
32    glimpse(performance_summary)
33  }
```

Explanation:

- **Lines 1–3:** The function `eval_strategy` starts by sorting the portfolio data by date.
- **Line 5:** Cumulative returns are computed by taking the product of the daily returns.
- **Line 6:** The maximum cumulative return reached up to each day is calculated; this is used for determining drawdowns.
- **Lines 9-12:** A daily risk-free return is defined (assuming a 4% annual rate), and the daily Sharpe ratio is computed to assess risk-adjusted performance.
- **Lines 14–29:** A summary table is created using `tibble` to capture key performance metrics, including:
  - Total number of long and short trades.
  - The percentage of winning trades for both long and short positions.
  - The average return for each type of trade.
  - The daily Sharpe ratio.
  - The final cumulative portfolio return.
  - The maximum drawdown percentage.
  - The overall winning percentage across all trades.
- **Line 32:** The summary is displayed using `glimpse`, providing a concise, human-readable overview.

9 Running the Backtest Pipeline (Addresses b, c, d, e, f)

In this final section, all functions are chained together to execute the complete backtesting process. The portfolio is built by filtering stocks, generating trading indicators, and setting trading rules. We then simulate trades and evaluate the overall performance. This section also sets the initial parameters, such as $100,000 starting capital and the maximum trade size.

Listing 7: Backtest Pipeline Execution

```
1  portfolio <- stocks |>
2    filter_stocks(10) |>      # Select top 10 stocks
3    gen_indicators(20) |>
4    set_rules()
5
6  # Define backtest parameters
7  max_trade <- 10000
8  initial_equity <- 100000
9  trade_days <- 250
10
11 ret_pct <- apply_trades(portfolio, trade_days, initial_equity, max_trade)
12 print(ret_pct)
13
14 eval_strategy(portfolio)
```

Metric	Value
Long Trades	2444
% Winning Long Trades	52.33%
Avg Long Trade Return	1.0012
Short Trades	1749
% Winning Short Trades	53.40%
Avg Short Trade Return	1.0011
Daily Sharpe Ratio	0.07497
Cumulative Portfolio Return	85.55%
Max Drawdown %	-31.68%
Overall % Winning Trades	50.49%

Table 1: Evaluation of Trading Strategy

Explanation:

- **Lines 1–4:** The portfolio is constructed by chaining the functions:
  1. `filter_stocks(10)` selects the top 10 stocks based on forecast error, ensuring we only analyze the most predictable stocks.
  2. `gen_indicators(20)` computes the slope (trend indicator) from a 20-day rolling window, which forms the basis for our trading signals.

3. `set_rules()` converts the slope into actionable trading signals (long/short) and computes the returns for each trade.

  This sequence addresses requirements (a) through (d).

- **Lines 7–9:** Trading parameters are defined:
  - `max_trade` imposes a cap on the amount invested in any single trade.
  - `initial_equity` sets our starting capital at $100,000.
  - `trade_days` defines the total number of days over which the backtest is run.
- **Line 11:** The `apply_trades` function is called to simulate the daily trading process, updating the portfolio equity and computing the overall return percentage.
- **Line 12:** The overall return percentage is printed, giving a quick summary of the strategy's performance.
- **Line 14:** Finally, `eval_strategy` is executed to calculate and display a detailed set of performance metrics, fulfilling requirement (f).

Now we move on to the final part for the tutorial to

10 Conclusion

This tutorial provides a comprehensive walkthrough of constructing a backtesting framework for a trading strategy on S&P 500 Industrials stocks. Our approach:

1. Uses an ETS model with an AAN specification to forecast stock prices (addressing requirement a).
2. Employs a backtest period from 1/1/2023 to 12/31/2024 with sufficient historical data for proper initialization (addressing requirement b).
3. Generates both long and short signals based on a computed trend (slope) from rolling forecasts and executes daily trades at the open and close (addressing requirement c).
4. Filters stocks by selecting those with lower forecast errors as a proxy for confidence (addressing requirement d).
5. Starts with $100,000 in cash and imposes constraints (e.g., maximum trade amount) to guide capital allocation based on predictability (addressing requirement e).

6. Evaluates the strategy using detailed performance metrics, including win percentages, average returns, Sharpe ratio, cumulative return, and drawdown (addressing requirement f).