Algorithmic Trading Using Three Strategies

Team A1
Gabriel Anzalone - 80543
Daniel Granados Olvera - 102453
Jules Remlinger - 73178

An Applied Master's Project submitted for the degree of
Master of Science in Financial Engineering

Department of Financial Engineering
EDHEC Business School
France

May 2025

Contents

Introduction

Financial markets produce enormous volumes of data every second, but extracting useful signals from this flow remains one of the most persistent challenges in quantitative finance. Over the past two decades, the use of algorithms in trading has grown from a narrow focus on execution to a wider role in prediction and allocation. As strategies have become more systematic, the question has shifted from whether to use models, to which models offer the best trade-off between accuracy, adaptability, and interpretability.

In regulatory terms, algorithmic trading is defined by the MiFID II directive (European Commission, 2014) as:

> "Trading in financial instruments where a computer algorithm automatically determines individual parameters of orders such as whether to initiate the order, the timing, price or quantity of the order or how to manage the order after its submission, with limited or no human intervention, and does not include any system that is only used for the purpose of routing orders to one or more trading venues or for the processing of orders involving no determination of any trading parameters or for the confirmation of orders or the post-trade processing of executed transactions."

In practice, however, algorithmic trading has expanded well beyond execution. It now encompasses the design of predictive models, the generation of trading signals, and the integration of high-frequency and fundamental data sources. The growing availability of structured datasets, such as CRSP and Compustat, combined with advances in computational resources, has made it feasible to construct, train, and evaluate models at scale.

Yet financial prediction remains difficult. Asset returns are noisy, non-stationary, and often driven by factors that are only partially observable. Classical statistical models such as ARIMA or GARCH are limited by their distributional assumptions and short memory. In contrast, machine learning models are data-driven and flexible, and have shown promise in capturing nonlinear interactions. Among these, neural networks offer a powerful framework for learning patterns in complex time

series, but they also introduce new questions about overfitting, interpretability, and generalization.

This project builds on those foundations to compare three groups of systematic trading strategies under consistent conditions:

1. Rule-based models that use technical indicators such as moving averages and momentum.

2. Value-based models built on firm-level fundamentals including price-to-earnings (P/E) and price-to-book (P/B) ratios.

3. Deep learning models, including long short-term memory (LSTM) and convolutional neural networks (CNN).

The project is not designed to identify a definitive best strategy, but rather to explore how different classes of algorithmic trading models behave. The goal is to build intuition for how rules, signals and data-driven models translate into systematic investment decisions. Through implementation and backtesting, the analysis aims to provide both theoretical insight and practical experience with the design of algorithmic trading systems.

The dataset covers daily data for S&P 500 firms from 2008 to 2022. Stock prices, returns, and volumes are obtained from the CRSP daily stock file, while firm-level accounting data comes from Compustat. Technical indicators are calculated using the `ta` library in Python. Fundamental ratios (PE and PB) are monthly and obtained with `wrds_ratios` suite. Survivorship bias is avoided by including only firms that were part of the S&P 500 at each point in time, based on historical index records. All fundamentals are lagged to ensure no forward-looking information enters the input.

Model training follows a rolling-window approach that simulates a forward-looking investment scenario. Each model is trained on a historical period and then evaluated on subsequent, unseen data. The objective is not only to compare classification accuracy, but also to assess portfolio-level outcomes such as Sharpe ratio and drawdown. By placing these models on equal footing, the analysis highlights where deep learning provides a tangible edge, and where simpler models may remain competitive. The input to the models consists of rolling sequences of recent market and firm-specific features. Classification performance is assessed using standard metrics such as accuracy, while financial performance is evaluated through simulated long-short portfolios using predicted labels. Sharpe ratio and maximum drawdown are used to capture risk-adjusted returns and downside exposure. All models are implemented in Python using `pandas`, `NumPy`, `TensorFlow/Keras`, and `scikit-learn`, and trained in GPU-enabled environments using Google Colab Pro.

The remainder of this project is structured as follows. Chapter 1 presents rule-based trading strategies built on technical indicators. Chapter 2 examines valuation-based models that leverage firm-level fundamental data. Chapter 3 introduces deep learning approaches and evaluates their predictive performance. The project concludes with a comparative analysis of all models and a discussion of their practical implications for systematic trading.

Chapter 1

Trading Strategies with Moving Averages and Momentum

1.1 Introduction

Technical indicators like moving averages and momentum measures have long been used by traders for their simplicity and empirical effectiveness. By capturing trends and reversals in financial time series, they provide an interpretable approach to modeling market dynamics. Among these, *moving averages (MA)*, including Simple Moving Averages (SMA) and Exponential Moving Averages (EMA), and *momentum indicators* like the Relative Strength Index (RSI) and Rate of Change (ROC), have been widely studied for their predictive potential.

Jegadeesh and Titman (1993) identified the *momentum effect*, showing that past winners tend to outperform losers over intermediate horizons, challenging weak-form market efficiency. Chan et al. (1996) linked this to underreaction to earnings news, while Grinblatt and Moskowitz (1999) emphasized industry-level momentum effects.

Brock et al. (1992) provided early statistical validation for technical trading rules such as moving average crossovers, demonstrating their ability to yield excess returns, particularly in trending markets.

As financial data has grown in volume and complexity, *machine learning* has emerged as a flexible alternative to fixed-rule systems. Models such as random forests and neural networks can capture non-linear interactions among indicators and adapt to changing market conditions.

This section implements random forests and neural networks using moving averages and momentum indicators as inputs. The objective is to assess whether these models can produce reliable predictive signals in an out-of-sample, cross-sectional setting and outperform traditional rule-based strategies.

1.2 Theoretical Background

1.2.1 Technical Indicators

Moving averages are fundamental tools in time series analysis and technical trading. They smooth out short-term fluctuations and highlight longer-term trends or cycles in data. The two most common types are:

- **Simple Moving Average (SMA)**: Calculates the average of a selected range of prices, typically closing prices, by the number of periods in that range.

  The Simple Moving Average over n periods at time t is defined as:

  $$\text{SMA}_t = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

  Where P_t is the price of a stock at time t.

- **Exponential Moving Average (EMA)**: Similar to SMA but places a greater weight and significance on the most recent data points.

  The Exponential Moving Average is defined recursively as:

  $$\text{EMA}_t = \alpha \cdot P_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

  with the smoothing factor α given by:

  $$\alpha = \frac{2}{n+1}$$

  where n is the chosen window length.

- **Relative Strength Index (RSI)**: RSI is a momentum oscillator that measures the speed and change of price movements. It oscillates between 0 and 100 and is typically used to identify overbought or oversold conditions in a traded asset. An RSI above 70 is generally considered overbought, while an RSI below 30 is considered oversold. This indicator helps traders assess the strength of a security's recent price performance.

  The RSI is calculated as:

  $$\text{RSI}_t = 100 - \frac{100}{1 + RS_t}$$

  where the relative strength RS_t is:

  $$RS_t = \frac{\text{Average Gain over } n \text{ periods}}{\text{Average Loss over } n \text{ periods}}$$

1.3 Strategy Implementation

1.3.1 Feature Engineering

This subsection details the engineered features used as inputs to the machine learning models. Given the time-series nature of financial data and the focus on trend-following and mean-reversion behaviors, the feature design emphasizes normalized moving averages, their interactions, and momentum oscillators.

Let $P_{i,t}$ denote the adjusted closing price of stock i at time t, and $\text{EMA}_{i,t}^{(n)}$ its exponential moving average over a lookback window of n trading days. We consider the following set of lookback windows:

$$\mathcal{N} = \{5, 20, 35, 50, 70, 100, 130\}$$

1. EMA Ratios. For each $n \in \mathcal{N}$, we compute the EMA ratio:

$$\text{EMA Ratio}_{i,t}^{(n)} = \frac{\text{EMA}_{i,t}^{(n)}}{P_{i,t}}$$

This normalized measure allows for a scale-invariant comparison between the price and its smoothed average. An EMA ratio greater than one implies that the price has recently declined relative to its past trend, while a value less than one indicates recent upward momentum. Unlike raw price levels or differences, this ratio offers a standardized representation interpretable across assets with different price levels.

To mitigate initialization bias, we discard all values during the warm-up periods required for reliable EMA estimation. Furthermore, stocks lacking sufficient historical data to compute all EMAs in $\mathcal{N}$ are excluded from training and inference phases.

2. EMA Spread Ratios. To further capture the temporal structure of moving averages, we compute the difference between pairs of EMAs, normalized by the current price:

$$\text{Diff}_{i,t}^{(n_l,n_s)} = \frac{\text{EMA}_{i,t}^{(n_l)} - \text{EMA}_{i,t}^{(n_s)}}{P_{i,t}}, \quad \text{for } n_l > n_s,\ n_l, n_s \in \mathcal{N}$$

These features are intended to reflect the relative steepness or convergence/divergence between short- and long-term moving averages. Positive values may indicate bullish momentum or early trend formation, while negative values may be suggestive of reversals. This construction generalizes traditional moving average crossover signals within a continuous and model-compatible feature space.

3. Relative Strength Index (RSI). As a canonical momentum oscillator, we computed the RSI over the same set of windows $\mathcal{N}$, allowing the model to extract patterns from different short- and medium-term momentum regimes. Each RSI value is clipped between 0 and 100 and requires at least n valid price observations to be computed.

Final Feature Set. After engineering the features above, our complete feature vector for each asset-date pair includes:

- EMA ratios $\left\{\text{EMA Ratio}_{i,t}^{(n)} \mid n \in \mathcal{N}\right\}$
- EMA spread ratios $\left\{\text{Diff}_{i,t}^{(n_l,n_s)} \mid n_l > n_s,\ n_l, n_s \in \mathcal{N}\right\}$
- RSI values $\left\{\text{RSI}_{i,t}^{(n)} \mid n \in \mathcal{N}\right\}$

All features are computed on a rolling basis using only past data to ensure strict time-series causality. The feature set is designed to be cross-sectionally consistent and time-series aware, offering the models both scale-invariant trend descriptors and regime-sensitive momentum measures. This allows flexible learning of non-linear interactions and conditional signals that traditional heuristics may not fully capture.

1.3.2 Model Training and Evaluation

To evaluate the predictive power of our engineered features, we implemented a rolling-window classification framework using both Random Forest (RF) and Artificial Neural Network (ANN) models. This section outlines the methodology employed for model selection, training, and out-of-sample evaluation.

Rolling Window Design. We partition the dataset into sequential, non-overlapping periods using a walk-forward validation approach. Each period consists of a training window of 750 trading days and a subsequent test window of 250 trading days. This rolling procedure is iterated over the entire time span, yielding multiple out-of-sample performance estimates.

Let T denote the ordered set of available trading dates. For each iteration i, we define:

$$\text{Train Dates}^{(i)} = T_{i \cdot h : i \cdot h + w}$$
$$\text{Test Dates}^{(i)} = T_{i \cdot h + w : i \cdot h + w + h}$$

where $w = 750$ is the training window length, and $h = 250$ is the test horizon. Only stocks that are present in the full training window are retained for evaluation to prevent look-ahead bias and survivorship distortions.

Label Construction. For each trading day t, we compute the cross-sectional median of the next-day returns. A binary classification label is then defined per stock:

$$y_{i,t} = \begin{cases} 1 & \text{if } r_{i,t+1} > \text{Median}\left(\{r_{j,t+1}\}_j\right) \\ 0 & \text{otherwise} \end{cases}$$

This label definition ensures a balanced, cross-sectional prediction task aligned with relative performance.

Preprocessing. Prior to model training, the feature matrix is standardized using a StandardScaler fitted exclusively on the training data. To reduce dimensionality and mitigate collinearity, Principal Component Analysis (PCA) is applied, retaining the top 10 orthogonal components. The fitted PCA transformation is then applied to the test data.

Model Classes and Hyperparameter Tuning. We employ two supervised learning algorithms:

- **Random Forest (RF):** An ensemble of decision trees with bagging and feature subsampling
- **Artificial Neural Network (ANN):** A feedforward multi-layer perceptron with early stopping.

Hyperparameters are optimized via GridSearchCV with 5-fold time-series cross-validation on the training set, using the ROC-AUC score as the selection criterion. The search spaces are defined as:

- RF: `n_estimators` $\in \{50, 100\}$, `max_depth` $\in \{\text{None}, 3, 5\}$
- ANN: `hidden_layer_sizes` $\in \{(50, 30, 10), (100, 50, 25)\}$, `alpha` $\in \{10^{-3}, 10^{-2}\}$, `learning_rate_init` $\in \{10^{-3}, 10^{-4}\}$

The model achieving the highest mean ROC-AUC across validation folds is retained as the period-specific winner.

Evaluation Metrics. For each model and period, we compute both in-sample (training) and out-of-sample (test) metrics:

- **Accuracy:** Proportion of correct predictions.
- **Recall:** Sensitivity to positive class ($y = 1$).
- **Precision:** Positive predictive value.
- **ROC-AUC:** Area under the Receiver Operating Characteristic curve.

Model Outputs. The selected model's predicted probabilities and binary classifications are recorded for each stock in the test set. These serve as the basis for strategy simulation in the following section. Additionally, the model's configuration, training and testing metrics, and associated metadata are archived for post-analysis.

1.4 Results and Evaluation

1.4.1 Machine Learning Metrics

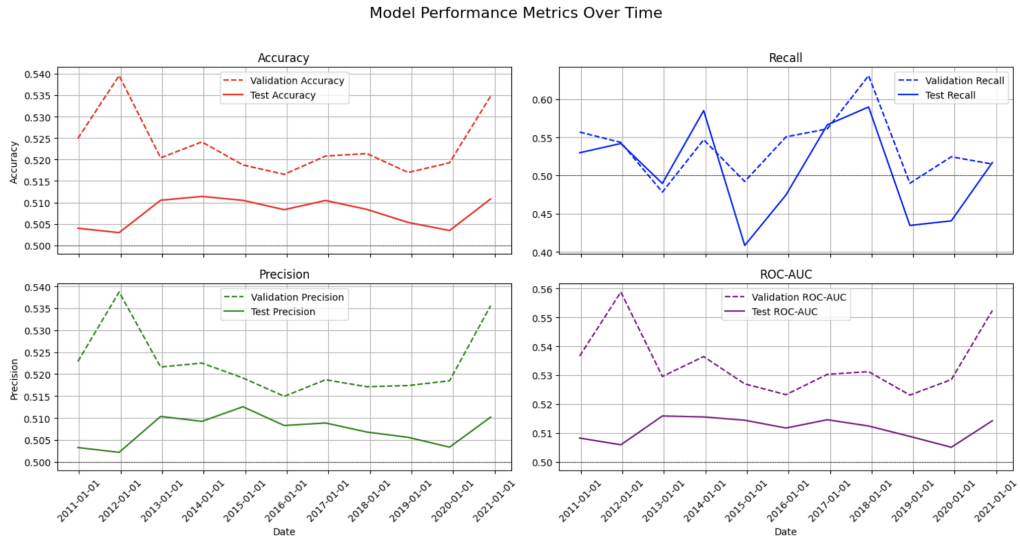

Figure 1.1: Performance metrics of the Moving Average-based models over all rolling periods.

Table 1.1: Summary statistics of out-of-sample classification performance across all test periods.

Metric	Mean	Std Dev	p-value vs 0.5	95% CI Lower	95% CI Upper
Accuracy	0.5078	0.0033	0.0000	0.5056	0.5100
Recall	0.5070	0.0623	0.7163	0.4651	0.5489
Precision	0.5073	0.0034	0.0000	0.5051	0.5096
ROC-AUC	0.5115	0.0039	0.0000	0.5089	0.5141

Comment. The classification performance across all rolling periods indicates marginal, but statistically significant, predictive power for several metrics. Accuracy, Precision, and ROC-AUC all exhibit mean values slightly above the random baseline of 0.5, with corresponding p-values below conventional significance thresholds. Although the magnitude of these differences may appear small in absolute terms, the stability of results—as reflected by the low standard deviations and tight confidence intervals—suggests consistent signal extraction by the model. Notably, the Recall metric

displays higher variance and a non-significant p-value, highlighting potential asymmetries in the model's ability to capture the positive class. This result is consistent with the challenges of cross-sectional return classification, where the signal-to-noise ratio is typically low.

1.5 Strategy Backtesting

Having obtained probabilistic predictions $\mathbb{P}(S_t) \in [0, 1]$ for each stock S at each date t, we now define a class of trading strategies by mapping these predictions to trading positions. This mapping is denoted $P(\mathbb{P}(S_t))$, where $P : [0, 1] \to [-1, 1]$, representing a normalized position between fully short (-1) and fully long $(+1)$.

All positions are held for a single period, consistent with the one-step-ahead forecast horizon of the model. Three main families of trading strategies are explored: binary classification-based, continuous transformation-based, and ranking-based portfolio strategies.

1.5.1 Binary Classification Strategies

Binary strategies discretize the predictions into categorical actions: long, short, or neutral. Let $k \in [0, 0.5]$ be a confidence threshold. Then the strategy is defined as:

$$P(\mathbb{P}(S_t)) = \begin{cases} +1 & \text{if } \mathbb{P}(S_t) > 0.5 + k \\ 0 & \text{if } 0.5 - k \leq \mathbb{P}(S_t) \leq 0.5 + k \\ -1 & \text{if } \mathbb{P}(S_t) < 0.5 - k \end{cases}$$

The long-only variant sets all negative signals to zero, i.e., $P(\mathbb{P}(S_t)) = 0$ when the model recommends shorting the asset.

This strategy is straightforward to interpret and corresponds to traditional signal thresholding, where trades are executed only when the model is sufficiently confident.

1.5.2 Continuous Transformation Strategies

Rather than discretizing model outputs, this class of strategies transforms the predictions smoothly into positions, providing more nuanced exposure based on model confidence.

Several continuous mappings are considered:

- **Linear Mapping**:

  $$P(x) = 2x - 1$$

  A simple affine transformation, where a prediction of 0 corresponds to a fully short position and 1 to fully long.

- **Sinusoidal Mapping**:

$$P(x) = \sin\big(\pi(x - 0.5)\big)$$

This function dampens positions near 0.5 (low confidence) and emphasizes those near the extremes.

- **Logistic Mapping**:

$$P(x; k) = \frac{2(x - S_0)}{S_1 - S_0} - 1, \quad \text{where} \quad S_0 = \frac{1}{1 + e^{k/2}}, \quad S_1 = \frac{1}{1 + e^{-k/2}}$$

The parameter $k \geq 0$ controls the sharpness of the mapping. When $k = 0$, the function reduces to the linear mapping; as $k \to \infty$, it converges to a step function centered at 0.5.

- **Hyperbolic Tangent Mapping**:

$$P(x; k) = \frac{\tanh(k(x - 0.5))}{\tanh(0.5k)}$$

This mapping also allows interpolation between smooth and binary behaviors, depending on the choice of k.

These strategies attempt to exploit the magnitude of model confidence rather than just its sign, potentially yielding more information-efficient trading signals.

1.5.3 Ranking-Based Portfolio Strategies

This class of strategies emphasizes cross-sectional model confidence by ranking stocks each day based on their predicted probabilities $\mathbb{P}(S_t)$. At every time t, we identify the top k and bottom k stocks based on their prediction scores.

The trading rule is defined as:

$$P(S_t) = \begin{cases} +1 & \text{if } S \text{ is among the top } k \text{ predictions at time } t \\ -1 & \text{if } S \text{ is among the bottom } k \text{ predictions at time } t \\ 0 & \text{otherwise} \end{cases}$$

A long-only variant of this rule consists in holding long positions in the top k stocks and remaining neutral otherwise. These strategies are particularly useful in cross-sectional contexts, where the model is more confident in relative performance rankings than in absolute return levels.

1.5.4 Backtesting Framework

For each strategy, we perform an in-sample performance analysis by applying the position function P to the prediction outputs $\mathbb{P}(S_t)$ across all rolling test periods.

We evaluate strategy performance using standard metrics such as average return, volatility, and the annualized Sharpe ratio:

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[r_t]}{\sigma(r_t)} \times \sqrt{252}$$

where r_t is the daily portfolio return, $\mathbb{E}[r_t]$ its empirical mean, and $\sigma(r_t)$ its standard deviation. The factor $\sqrt{252}$ annualizes the Sharpe ratio under the assumption of 252 trading days per year.

For each strategy family, we explore a range of threshold values or hyperparameters (e.g., k in classification, or smoothing parameters in logistic/tanh mappings) and report the configuration that achieves the highest in-sample Sharpe ratio. While this evaluation is subject to overfitting risk, it provides valuable insights into which transformations best align with the model's signal characteristics.

1.5.5 Results

Parameter Ranges. To assess the sensitivity of each strategy to its threshold or hyperparameter K, we perform a grid search using 200 evenly spaced values per strategy. The tested ranges are as follows: binary classification strategies use $K \in [0.000,\ 0.499]$; continuous transformation strategies (logistic and hyperbolic) use $K \in [0.001,\ 1000.000]$; long-short top-K absolute strategies use $K \in \{1, \ldots, 250\}$; long-only absolute strategies use $K \in \{1, \ldots, 475\}$; and long-only relative percentile strategies use $K \in [0.001,\ 0.995]$.

This parameter sweep enables the identification of the best-performing configurations in-sample, based on the Sharpe ratio. While these optimal settings are not used directly in out-of-sample decision-making, they provide useful guidance about which families of strategies and transformations are most compatible with the model's prediction output.

The Benchmark used as a comparison is a 1/N portfolio of all stocks the ML algorithm was able to train on.

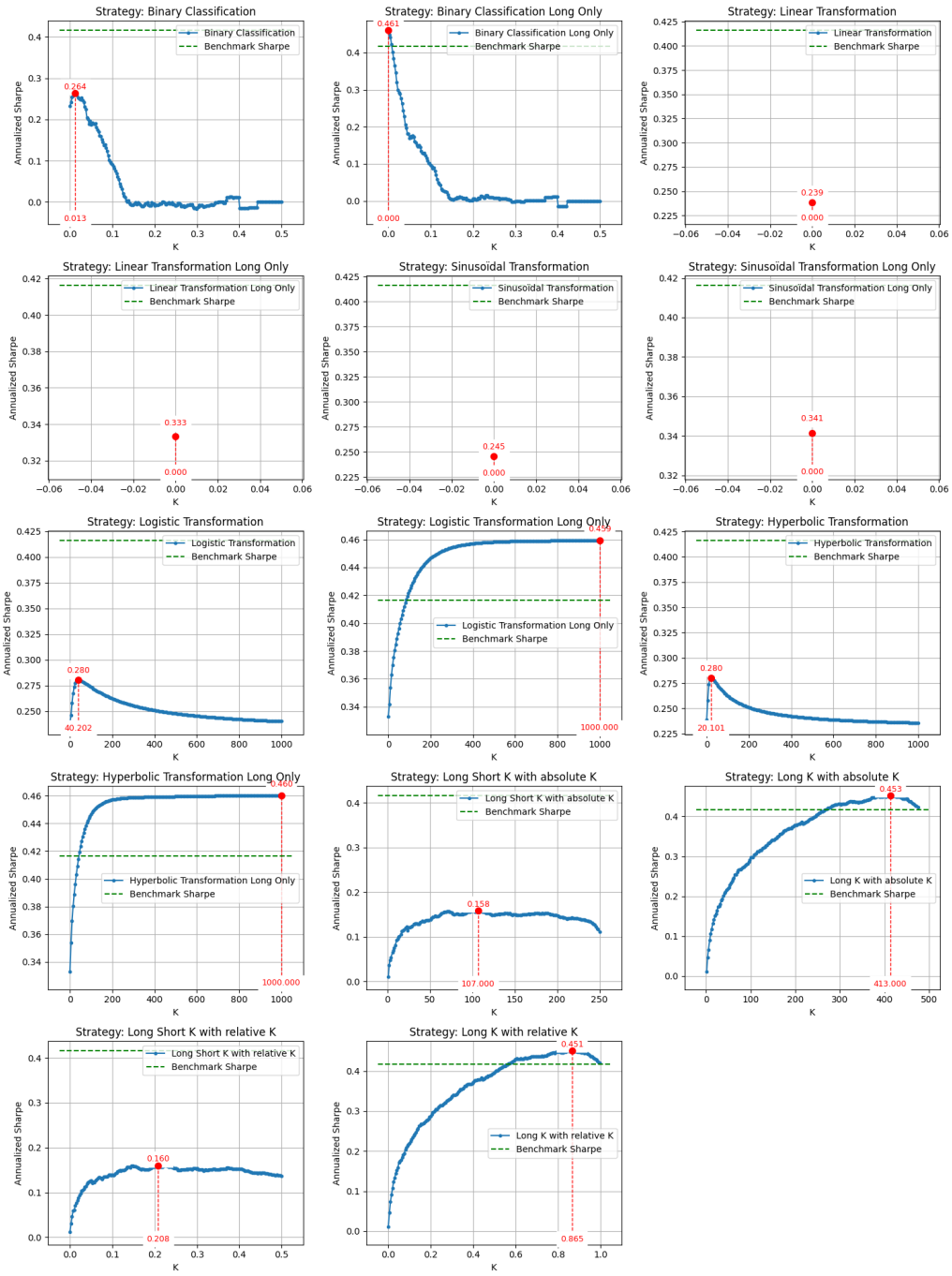

Figure 1.2: Sharpe ratio optimization results for Moving Average-based strategies.

Optimization results : Figure 1.2 displays the annualized Sharpe ratios across a range of threshold or transformation parameters K for each strategy, evaluated in-sample over all rolling periods. Binary classification strategies (standard and long-only) achieve their highest Sharpe ratios for small K values near the classification boundary. The long-only variant peaks above 0.4, but performance drops quickly as K increases, indicating predictive power is concentrated in the model's most confident signals.

Linear and sinusoidal transformations yield moderate Sharpe ratios, with optimal

configurations only slightly outperforming the benchmark. Their performance is sensitive to small changes in K, with optimal values often near the grid edges. The sinusoidal long-only variant performs better, likely due to its suppression of uncertain signals near the prediction midpoint.

Logistic and hyperbolic transformations, which interpolate between soft and hard classification, show steadily increasing Sharpe ratios at higher K. Their long-only versions exceed 0.45, suggesting sharply polarized exposures are effective in long-only settings. Standard (long-short) versions underperform, likely due to unreliable short signals.

Portfolio-based ranking strategies (top-K) deliver the best results. The long-only absolute K strategy reaches the highest Sharpe ratio (0.453), followed closely by the relative K version (0.451). These concentrate capital on the model's most confident long calls, which appear more accurate than short ones. Long-short versions lag, confirming an asymmetry in prediction quality.

Several patterns emerge: (i) long-only strategies consistently outperform; (ii) high-confidence signals—via thresholds or top-K selection—yield better performance; (iii) flexible mappings like logistic and hyperbolic excel when approximating step functions; and (iv) most strategies beat the benchmark, confirming the model's predictive power when paired with appropriate signal transformations.

Although based on in-sample evaluations and subject to overfitting risk, these results offer valuable guidance on which transformations best convert predictions into trading signals.

Note. A complementary figure showing Sharpe ratios for all strategy variants across individual test periods is provided in the Appendix.

1.6 Conclusion

This chapter shows that machine learning models, even with modest predictive accuracy, can generate economically meaningful signals when combined with well-designed transformations of moving average-based features. Long-only strategies and top-K selection rules proved particularly effective, highlighting the model's strength in identifying outperformers. While these results are promising, they are in-sample and do not account for trading costs, which could materially affect net performance. Future work should extend the analysis to out-of-sample evaluation and incorporate transaction cost modeling to assess practical viability.

Chapter 2

Value-Based Strategies

2.1 Introduction

Value-based strategies rely on the crossing of fundamental ratios with their historical averages to generate trading signals. This chapter details the implementation of such strategies using monthly data on Price-to-Earnings (P/E) and Price-to-Book (P/B) ratios for S&P 500 constituents over the period 2008 to 2022.

The analysis begins with a discussion on the theoretical background of value-based signals, followed by the introduction of a long-only historical strategy. A machine learning algorithm is then employed to predict future ratios, upon which trading strategies are implemented.

2.2 Theoretical Background

The strategies implemented in this chapter rely on valuation of stocks based on fundamental ratios:

- If P/E (or P/B) > Historical Average: the stock is considered **overvalued**, we might consider **selling** the stock.
- If P/E (or P/B) < Historical Average: the stock is considered **undervalued**, we might consider **buying** the stock.

Regarding historical average, one can either use:

- Expanding mean : average of all past ratios since the first monthly date.
- Expanding median : median of all past values up to the current month.
- Exponentially Weighted Moving Average (EWMean) : weighted average where older values gradually lose importance over time (span = 12 months meaning that a value looses half its importance in twelve months).

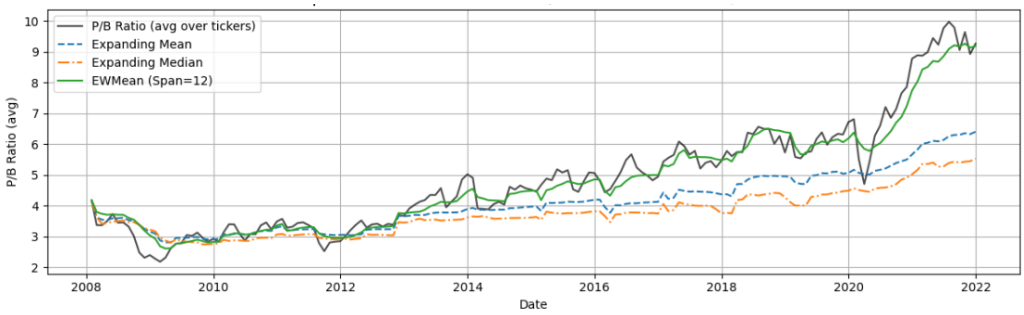

Figure 2.1: Comparison of different historical averages for the P/B ratio.

In this chapter, EWMean is considered the most suitable historical average, as it better captures recent valuation changes and exhibits more frequent crossings with average ratio over the period (especially for P/B Ratio, see Figure 3.1).

2.3 Historical Example

This first historical strategy consists in generating trading signals based on the relative position of a valuation ratio to its historical average (EWMean with a span of 12 months).Each stock in the S&P 500 from 2008 to 2022 is individually assessed on a monthly basis.

The strategy rule is as follows: we buy the stock when the current ratio crosses below the EWMean (meaning that the stock is undervalued), and we sell this same stock once it crosses above (the stock being overvalued).

Short selling is not allowed as it is a long-only strategy, and trades are initiated only when a buy signal is triggered. Once the latter is triggered, we hold the stock until there is a sell signal. This logic is applied separately to the P/E and P/B ratios.

The following two tables summarize the performance of this strategy across all stocks in the monthly dataset:

Table 2.1: Summary Statistics — Historical Strategy (P/E)

Metric	**Value**
Number of Trades	5,939
Median Return per Trade	6.15%
Median Annualized Return	24.19%
Median Sharpe Ratio	1.22
Average Holding Period	193.9 days ($\approx$ 6.4 months)
Hit Ratio	73.43%

Table 2.2: Summary Statistics — Historical Strategy (P/B)

Metric	**Value**
Number of Trades	6,920
Median Return per Trade	6.09%
Median Annualized Return	35.61%
Median Sharpe Ratio	1.79
Average Holding Period	145.0 days ($\approx$ 4.8 months)
Hit Ratio	77.92%

The historical strategy yields robust results for both metrics. Both display solid risk-adjusted performance and Hit Ratio above 70%. The historical P/B strategy produces the best results, with a higher median annualized return and a Sharpe ratio approaching 1.8.

2.4 Strategy Implementation

2.4.1 Random Forest Regressor

This section is about the implementation of a Machine Learning algorithm to forecast future fundamental ratios - specifically, next month's P/E and P/B ratios. The model is implemented, trained, and evaluated relatively to the position of the real fundamental ratios with respect to the historical average. These predictions are then used in different Value-Based trading strategies.

The features used in this Random Forest regressor are lagged P/E and P/B ratios and lagged adjusted prices. The target is next month's fundamental ratio.

The Random Forest regressor is implemented through a rolling period of 24 months for train set and 12 months for test set - with non-overlapping windows to ensure clean out-of-sample evaluation. Analysis is therefore conducted only on stocks that contains at least 36 months of data, in order to have at least one train period and one test period for the selected tickers.

Evaluation of this regressor is based on Root-Mean-Square-Error (RMSE) to see the difference between the predicted ratio and the actual one. Binary accuracy is also used, to see whether the predicted ratio is above the EWMean when the actual one is and vice versa when it is below.

Hyperparameters are tuned directly for each train set for all stocks and best parameters are selected and implemented for the test set associated.

2.4.2 P/E Ratio: Prediction Results

Median RMSE across all test sets is 34.42, reflecting large variations between the predicted values and actual P/E ratios. Binary accuracy averaged 41.67%, suggesting limited reliability in directional forecasts based on this model. In only 41% of cases, the model correctly predicts that the P/E ratio is above the EWMean when the actual value is..

Table 2.3: Summary of Prediction Metrics --- P/E Ratio

Metric	Value
Median RMSE	34.42
Median Accuracy	41.67%

Table 3.4 displays the three first predictions for a single stock displaying large variations and extremely negative P/E Ratios over time.

Table 2.4: Example of PE Predictions

Ticker	Date	Predicted P/E	True P/E	Historical Mean
A	2010-03-31	-349.56	-604.33	-183.43
A	2010-04-30	-272.99	-539.33	-248.18
A	2010-05-31	-272.65	52.65	-292.98

Out-of-sample performance is weak, especially in terms of directional accuracy.

2.4.3 P/B Ratio: Prediction Results

Median RMSE across all test sets is 0.1716 and median accuracy of the directional signal remains consistent with the P/E analysis at 41.67%.

Table 2.5: Summary of Prediction Metrics --- P/B Ratio

Metric	Value
Median RMSE	0.1716
Median Accuracy	41.67%

Again, prediction quality varies widely depending on the stock. For some tickers, accuracy approaches 90%, while others consistently hover below 30%. Results are mixed for this Random Forest regressor, both in terms of RMSE and directional accuracy.

2.5 Backtesting and Comparison

This section evaluates the performance of trading strategies built on the predictions given by the Random Forest regressor. Two main strategies are implemented: a long-only strategy and a reversal ("one-sided flip") strategy. Strategies are tested separately for the P/E and the P/B Ratios, and also based on a combination of these fundamental ratios. Performance is then compared across strategies and against a Buy-and-Hold approach.

2.5.1 Long-Only Strategy

Predicted ratios are compared to their respective EWMean. For each month, if the predicted P/E or P/B ratio is below its EWMean, the stock is considered undervalued and a buy signal is generated. Once the stock is no longer considered undervalued (predicted ratio above the EWMean), a sell signal is triggered.

To avoid any look-ahead bias, signals are shifted by one month, ensuring that trading decisions are based solely on information available at time t.

In this long-only strategy, only buy signals are acted upon. Once a buy signal is issued, the position is held until a sell signal is received.

We implement the strategy first for P/E Ratio and P/B separately. We also implement it based on a combination of these fundamental ratios, with the following rationale: we buy the stock only if both ratios display undervaluation and vice versa for the selling signal.

Results are summarized in Table 3.6.

Table 2.6: Performance Summary --- Long-Only Strategies

Strategy	# Trades	Median Return (%)	Ann. Return (%)	Sharpe	Hit Ratio (%)
P/E Only	4,918	2.93	12.14	0.74	61.39
P/B Only	5,112	2.34	10.16	0.62	60.04
P/E + P/B	1,260	2.29	15.57	1.00	61.27

Compared to the historical simulation, the model underperforms across all metrics due to prediction uncertainty. Annualized Sharpe Ratios are significantly lower for both P/E and P/B strategies. However, the combined strategy retains a robust Sharpe ratio of 1, indicating stronger risk-adjusted performance when both valuation ratios are considered simultaneously. This combined strategy exhibits a smaller number of trades — which is expected, as trades are executed only when both ratios indicate undervaluation or overvaluation. Despite the reduced trading frequency, it achieves a higher median annualized return and Sharpe Ratio, while maintaining a solid Hit Ratio above 61%.

Hit Ratios are robust for the three strategies - which is expected too as we are working with long-only strategies on the constituents of an index that has considerably grown over the period.

2.5.2 Reversal Strategy

Reversal (or "one-sided flip") strategy has the following rationale: we enter either a long or short position depending on the initial signal we have for a stock. Once we exit the position (due to an inverse signal), we enter a new position (either long or short) depending on next month's signal.

Both long and short positions are accepted for every ticker and the strategy is implemented for P/E Ratio, P/B Ratio and also a combination of the two. For the latter, we go long when the two ratios indicate an undervaluation of the stock and vice versa for the short position.

Table 2.7: Performance Summary --- Reversal Strategies

Strategy	# Trades	Median Return (%)	Ann. Return (%)	Sharpe	Hit Ratio (%)
P/E Only	7,721	-0.05	-0.16	-0.01	49.70
P/B Only	7,932	-0.32	-1.21	-0.06	48.60
P/E + P/B	4,086	-0.51	-0.83	-0.04	48.87

Results are mixed, with negative median annualized returns and poor risk-adjusted performance for the three strategies. Hit Ratios are all below the 50% threshold.

Short trades perform particularly poorly, dragging down the overall performance even though long trades display robust results.

Focusing on short trades for P/E ratio only, we observe a nearly equal number of short (3,864) and long (3,857) trades over the period. Nevertheless, the distribution of short trade returns is considerably more left-skewed, as evidenced by Figure 3.2 and Table 3.8.

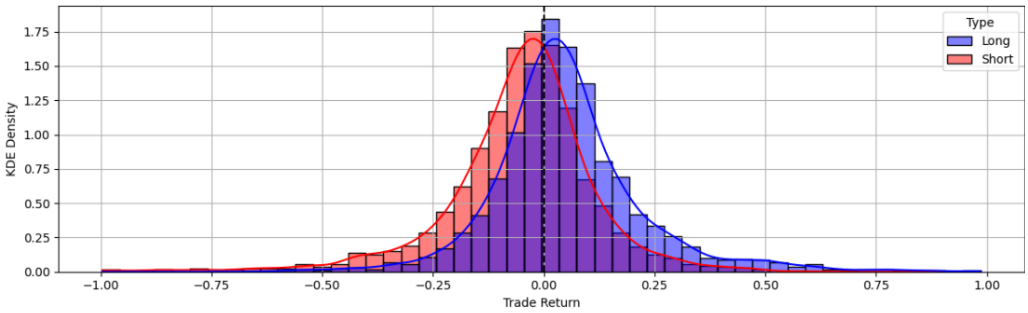

Figure 2.2: Return Distribution — Long vs Short (Reversal Strategy on P/E Ratio)

Table 2.8: Performance by Position Type — Reversal Strategy (P/E)

Position Type	Median Return (%)	Annualized Return (%)	Hit Ratio (%)
Long	3.60	13.16	62.95
Short	-3.77	-12.57	36.46

2.5.3 Comparison with Buy & Hold Benchmark

Additionally to the comparison across strategies, we also compare them with a pure Buy & Hold strategy. In the latter, we buy the stock after 24 months of data (to replicate the train set period) and hold it until the end of the dataset for this particular stock.

Table 2.9: Performance Summary --- Buy & Hold Benchmark

Metric	Value
Number of Stocks	595
Median Total Return (%)	93.51%
Median Annualized Return (%)	9.57%
Median Sharpe Ratio	0.37
Positive Return Ratio (%)	79.83%

The Buy & Hold strategy demonstrates robust long-term results with median Sharpe Ratio and Annualized Return similar to historical benchmark for US Equity (0.4 Sharpe Ratio and 9.5% annualized return).

Long-only machine learning strategies deliver higher Sharpe Ratios and slightly higher median annualized returns - but at the cost of significantly higher trading frequency. The Buy & Hold only displays one buy order and one sell order per stock.

Reversal strategies are largely undermined by short trades, performing poorly due to sharp losses during adverse movements. They therefore display lower performance metrics than the Buy & Hold strategy.

Comparison with a Buy & Hold strategy is less appropriate for Value-Based Strategy, especially due to the extreme difference in number of trades over the period and the length of the holding period. Comparing the strategies across them is a more appropriate approach.

2.6 Conclusion

Results are mixed for the Value-Based Strategy. Metrics are pretty weak for the Random Forest predictions, with high RMSE and low directional accuracy (40% for both P/E and P/B Ratios), revealing the difficulty to predict fundamental ratios during a long period.

Long only strategies display robust performance, even higher than Buy & Hold simulated portfolio (without considering the difference in number of trades and therefore in transaction costs).

Reversal strategies are completely hampered by poor performance on short trades, which results in poor annualized return and Hit Ratio below 50%.

Overall, findings suggest that machine learning predictions can be integrated into valuation-based frameworks despite low directional accuracy, but that the profitability of such strategies heavily depends on the treatment of signals, of the ratio considered and on the unexpected changes in valuation of stocks.

Chapter 3

Neural Network Models for Cross-Sectional Return Classification

3.1 Introduction

Over the past few years, deep learning has gained ground in quantitative finance as a different approach to decision-making. Traditional strategies usually follow fixed rules, such as those based on momentum or valuation. In contrast, neural networks learn directly from past data, which allows them to find patterns that might be too complex or subtle for rule-based methods to capture. This adaptability is particularly valuable in equity markets, where relationships among variables are complex, nonlinear, and subject to structural change over time.

In this chapter, we evaluate the predictive performance of three neural network architectures to classify daily stock returns. The models considered are Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNN), and a hybrid CNN-LSTM model. Each model is trained to predict whether a stock's return on the next trading day will exceed the cross-sectional median return across all stocks. The input features consist of technical indicators commonly used in practice, including past returns, adjusted prices, moving average signals, and the Relative Strength Index (RSI).

The experimental setup follows a rolling-window approach designed to reflect real-time forecasting constraints. Each model is trained on past data and evaluated out-of-sample using classification metrics such as accuracy and F1-score, as well as financial metrics like Sharpe ratio and drawdown.

3.2 Theoretical Background

Neural networks are machine learning models composed of layers of interconnected processing units known as neurons. Each neuron computes a weighted sum of its

inputs, adds a bias term to allow flexible shifts in activation, applies a nonlinear activation function, and passes the result forward. This mechanism allows networks to approximate complex, nonlinear relationships between inputs and outputs. The transformation performed at each layer can be expressed as:

$$a^{(l)} = \sigma(W^{(l)}a^{(l-1)} + b^{(l)}) \tag{3.1}$$

where $a^{(l)}$ denotes the activation of layer l, $W^{(l)}$ is the weight matrix, $b^{(l)}$ the bias vector, and σ a nonlinear activation function such as ReLU, sigmoid, or tanh.

Neural network training consists of two main stages. First, data flows through the model in the forward pass to produce predictions. Then, in the backward pass, the model evaluates its performance and determines how each parameter contributed to the error. For binary classification tasks, cross-entropy loss is commonly used to quantify the divergence between predicted probabilities and actual outcomes. The network updates its weights using backpropagation, a procedure that traces the error backward through the layers to compute gradients with respect to each parameter. These gradients are then used by an optimizer, such as stochastic gradient descent or Adam, to adjust the model weights iteratively and reduce the overall loss (Goodfellow et al., 2016).

To help models generalize better and avoid overfitting, neural networks often rely on regularization techniques. One common approach is dropout, which works by turning off random neurons during training, encouraging robustness by preventing co-adaptation. Early stopping halts training when validation performance ceases to improve, while L2 regularization constrains the magnitude of model parameters to limit complexity (Aggarwal, 2018).

3.2.1 Sequential Modeling with Neural Networks

Standard feedforward networks are not well suited for time series data because they assume that inputs are independent of one another. This poses a clear challenge in fields like finance, where what matters is not just the data itself but the sequence in which it unfolds. A single price or return means little without context; patterns, transitions, and the lingering impact of previous events are what give it meaning. For this reason, capturing temporal structure requires models that process data sequentially and incorporate information from earlier time steps into current predictions.

3.2.2 Recurrent Neural Networks (RNN)

Recurrent neural networks extend standard feedforward models by introducing a hidden state that is updated at each time step. This recurrent structure enables the network to maintain a form of memory, allowing it to process sequences of variable

length by conditioning each output on both the current input and the preceding hidden state. The update of the hidden state at time t is defined as:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3.2}$$

where x_t is the input at time t, h_t is the hidden state, and W_{xh}, W_{hh}, b_h are trainable parameters.

Through this mechanism, RNNs can learn temporal patterns such as cycles, transitions, or repetitions in sequential data. However, their ability to capture long-range dependencies tends to deteriorate as sequences grow longer. When gradients are propagated backward through many time steps, they can shrink or grow exponentially. These instabilities make it difficult for the network to update weights associated with distant inputs, and the influence of earlier information is often diminished or lost entirely.

3.2.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory networks were introduced by Hochreiter and Schmidhuber (1997) to address the limitations of standard RNNs in learning long-range dependencies. Their main idea was to add a memory that carries information forward across steps, along with small control units that decide when to let new input in, when to hold on to something, and when to clear it out. This helps the network stay focused on what matters over time without getting overwhelmed by everything it has seen.

Each LSTM unit includes a forget gate f_t, which determines what information to discard; an input gate i_t, which decides what new information to store; a candidate memory $\tilde{C}_t$; and an output gate o_t, which controls how much of the cell state affects the hidden state. The dynamics of the LSTM are governed by the following set of equations:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.3}$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{3.4}$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{3.5}$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{3.6}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.7}$$
$$h_t = o_t \cdot \tanh(C_t) \tag{3.8}$$

The result is a network architecture that can preserve relevant information across many time steps, while discarding what is no longer useful. This balance between memory and forgetfulness makes them well suited for problems where the influence of past observations varies across time. In practice, LSTMs have shown strong

performance in modeling sequences where patterns are not only nonlinear but also delayed, intermittent, or masked by noise. These characteristics are common in many real-world applications, including language processing, control systems, and financial forecasting, where signals do not follow fixed intervals and important dependencies may span multiple time steps.

3.2.4 Convolutional Neural Networks (CNN)

One of the main ideas behind convolutional networks is to stop treating every input as separate and instead look for patterns that repeat. Rather than connecting every input to every neuron, they apply small filters that scan through the data and react when something familiar shows up. This design became popular in image tasks, but its strength is more general: it helps the network focus on structure without getting buried in unnecessary details.

In the context of time series, CNNs can be adapted to operate along a single temporal dimension. A one-dimensional convolution slides a filter across the sequence and performs a weighted sum of neighboring values at each position:

$$s(t) = (x * w)(t) = \sum_{\tau=0}^{k-1} x(t+\tau)w(\tau) \tag{3.9}$$

where x is the input sequence, w is the filter of length k, and $s(t)$ is the resulting feature at time t. The same filter is applied across the entire sequence, which enables the model to detect repeated patterns such as abrupt changes, trend initiations, or periodic structures.

This convolutional operation introduces an important inductive bias: it assumes that useful features are local and translation-invariant across time. As a result, CNNs are especially effective when the predictive signal is embedded in short-term behaviors or recurring motifs. Stacking several convolutional layers helps the network recognize patterns that develop across longer time spans, building on simpler features identified earlier in the sequence. After convolution, pooling layers are often used to compress the output by taking key values from small regions, which helps simplify the data and make the model less sensitive to noise.

One of the major strengths of CNNs is that they can handle multiple parts of the input at once. This parallel processing makes them faster to train than sequential models and more scalable when working with large datasets. However, their ability to capture long-range dependencies is limited by the size and depth of the convolutional layers. This makes them less suitable for tasks where the relevant information is dispersed across distant time steps, unless complemented by architectures designed for long-term modeling.

In time series forecasting, including financial applications, CNNs can serve as powerful feature extractors. They are particularly well suited for identifying localized patterns in sequences of returns, prices, or indicators, such as bursts of volatility, short-lived trends, or structural breaks. These features can be used directly for prediction or passed to downstream models, such as recurrent networks, that are better equipped to process long-term dependencies (Chen & Wang, 2020).

3.2.5 CNN-LSTM Hybrid Architecture

Financial time series often contain both short-term bursts and slower-moving developments. To handle this structure, some models combine convolutional and recurrent layers. The convolutional part reacts to quick shifts in the input, turning raw sequences into signals that emphasize brief, localized changes. These signals are then passed to LSTM layers, which track how they unfold and interact over time.

This combination leverages the strength of CNNs in identifying spatially or temporally localized signals and the strength of LSTMs in modeling sequential dependencies. It has been especially effective in domains where local anomalies interact with broader patterns, such as news shocks influencing long-term price dynamics. In such settings, CNN-LSTM architectures provide a flexible and expressive framework for modeling return behavior (Lu et al., 2021; Zaheer et al., 2024).

3.3 Methodology

Having established the theoretical foundations of neural networks for sequential data, we now turn to their application in a financial classification context. In particular, we implement and evaluate three architectures: LSTM, CNN, and a CNN-LSTM hybrid, aimed at predicting whether a stock's next-day return will exceed the cross-sectional median. This section outlines the prediction objective, the construction of input features, the preprocessing pipeline, and the design of the model architectures used in the empirical analysis.

Our experimental framework follows the methodology proposed by Fischer and Krauss (2018), with adaptations to include convolutional and hybrid models. In line with their approach, we use rolling training windows, a cross-sectional binary classification target, and backtests based on long-short portfolio construction.

3.3.1 Feature Engineering and Selection

We compute the same set of technical indicators introduced in Chapter 1, applying them consistently across training and testing periods within the rolling evaluation framework. All features are constructed stock-by-stock using only past information to preserve the temporal integrity of the prediction task.

The feature set includes daily arithmetic and log returns, along with a range of exponential moving average (EMA)-based metrics. These include EMA-to-price ratios with windows of 5, 30, 50, and 70 days, as well as normalized spreads between EMAs of different lengths, capturing multi-scale trend signals. Additionally, we incorporate momentum indicators derived from the Relative Strength Index (RSI), calculated over 30, 50, 70, and 110-day horizons. Together, these variables provide a compact representation of recent return behavior, price levels relative to trends, and overbought or oversold conditions.

3.3.2 Preprocessing Pipeline: Standardization and Label Construction

After feature selection, all input variables are standardized using the mean and standard deviation computed across all stocks and dates in the training window. Specifically, for each selected feature x, we apply:

$$x_{i,t}^{\text{norm}} = \frac{x_{i,t} - \mu_{\text{train}}}{\sigma_{\text{train}}}$$

where μ_{train} and σ_{train} are the mean and standard deviation of the feature computed across the training set. These same statistics are applied to normalize the test data to ensure consistency and avoid look-ahead bias.

The target variable is constructed as a binary label that indicates whether a stock outperforms the median return on the next trading day. For each stock i and day t, we define:

$$y_{i,t+1} = \begin{cases} 1 & \text{if } r_{i,t+1} > \tilde{r}_{t+1} \\ 0 & \text{otherwise} \end{cases}$$

where $\tilde{r}_{t+1}$ is the cross-sectional median return across all eligible stocks at time $t+1$. This framing converts the forecasting task into a balanced classification problem and emphasizes relative rather than absolute performance.

3.3.3 Input Construction for Sequential Models

For each stock, the features described above are grouped into fixed-length rolling windows to form input sequences suitable for time-series models. Each input sample consists of a matrix of shape $T \times F$, where $T = 240$ is the number of past trading days in the sequence and $F = 6$ is the number of features used (comprising daily return and the top five features selected via ANOVA). These sequences are generated using a sliding window mechanism applied stock-by-stock. Each sequence is aligned with a binary target label corresponding to the return on the day immediately following the final time step.

This input format is used for both LSTM and CNN models. For CNNs, the convolutional filters are applied across the temporal dimension to extract local patterns. For LSTMs, the entire sequence is processed by recurrent units to capture longer-range dependencies.

3.3.4 Model Architectures

We evaluate three architectures: LSTM, CNN, and a CNN-LSTM hybrid. All three models are implemented and trained using the same rolling-window setup and evaluated on the same test periods.

LSTM Network

Architecture Summary:

Stage	Layer Type	Output Shape	Parameters
Input	Input Layer	(None, 180, 15)	0
Preprocessing	SpatialDropout1D (0.10)	(None, 180, 15)	0
LSTM Block 1	LSTM (128 units, return_seq=True)	(None, 180, 128)	73,728
	Dropout (0.15)	(None, 180, 128)	0
LSTM Block 2	LSTM (64 units)	(None, 64)	49,408
	Dropout (0.15)	(None, 64)	0
Dense Block	Dense (32, ReLU)	(None, 32)	2,080
Output Layer	Dense (2, Softmax)	(None, 2)	66
Total			**125,282**

Table 3.1: Updated LSTM Model Architecture

CNN Model

Architecture Summary:

Stage	Layer Type	Output Shape	Parameters
Input	Input Layer	(None, 180, 15)	0
Preprocessing	SpatialDropout1D (0.10)	(None, 180, 15)	0
Normalization	Normalization (axis=-1)	(None, 180, 15)	30
Conv Block	Conv1D (128 filters, kernel=3, ReLU)	(None, 178, 128)	6,272
	Conv1D (64 filters, kernel=3, ReLU)	(None, 176, 64)	24,640
	GlobalMaxPooling1D	(None, 64)	0
Dense Block	Dense (32, ReLU)	(None, 32)	2,080
	Dense (2, Softmax)	(None, 2)	66
Total			**33,088**

Table 3.2: CNN Model Architecture (2-layer Conv1D + Global Pooling)

CNN-LSTM Hybrid

Architecture Summary:

Stage	Layer Type	Output Shape	Parameters
Input	Input Layer	(None, 180, 15)	0
Normalization	Normalization (axis=-1)	(None, 180, 15)	30
Convolution Block	Conv1D (32 filters, kernel=3, padding="same", ReLU)	(None, 180, 32)	1,472
	SpatialDropout1D (0.10)	(None, 180, 32)	0
Recurrent Block	LSTM (128 units, return_sequences=True)	(None, 180, 128)	82,944
	Dropout (0.15)	(None, 180, 128)	0
	LSTM (64 units)	(None, 64)	49,408
	Dropout (0.15)	(None, 64)	0
Dense Block	Dense (32, ReLU)	(None, 32)	2,080
	Dense (2, Softmax)	(None, 2)	66
Total			**135,998**

Table 3.3: CNN-LSTM Model Architecture

3.3.5 Implementation Details

All data processing, feature computation, and model training are implemented in Python. The primary libraries used include `pandas`, `NumPy`, `scikit-learn`, `TensorFlow/Keras`, and `ta`. Experiments are run in GPU-enabled environments using Google Colab Pro.

3.4 Results and Evaluation

3.4.1 Classification Metrics

To evaluate model performance, we compute accuracy, precision, recall, and ROC-AUC on the test segments of each rolling window. Reported values reflect the mean, standard deviation, and 95% confidence intervals across all test folds. We also report p-values for a one-sided test against the null hypothesis of a random classifier (H_0: mean = 0.5).

Table 3.4 summarizes the results. While performance margins are narrow, both the LSTM and the hybrid CNN-LSTM model achieve statistically significant accuracy and ROC-AUC scores above 0.5, indicating that the models learn signal beyond chance level. Precision remains close to the decision threshold by construction, while recall varies more widely due to class imbalance. Among all architectures, the

CNN-LSTM hybrid achieves the highest mean accuracy and ROC-AUC, with low variance and significant p-values.

Model	Metric	Mean	Std Dev	p-value vs 0.5	95% CI Lower	95% CI Upper
LSTM	Accuracy	0.5027	0.0016	0.0005	0.5015	0.5039
	Recall	0.5319	0.1021	0.3484	0.4589	0.6050
	Precision	0.5020	0.0016	0.0027	0.5009	0.5032
	ROC-AUC	0.5040	0.0020	0.0002	0.5026	0.5055
CNN	Accuracy	0.5018	0.0037	0.1653	0.4991	0.5044
	Recall	0.5394	0.1823	0.5115	0.4090	0.6698
	Precision	0.5014	0.0048	0.3814	0.4980	0.5048
	ROC-AUC	0.5026	0.0043	0.0914	0.4995	0.5056
CNN-LSTM	Accuracy	0.5030	0.0012	<0.0001	0.5021	0.5039
	Recall	0.5579	0.1123	0.1374	0.4776	0.6382
	Precision	0.5023	0.0012	0.0002	0.5014	0.5032
	ROC-AUC	0.5042	0.0015	<0.0001	0.5031	0.5053

Table 3.4: Classification performance metrics averaged across rolling test windows.

3.4.2 Backtest Simulation

We simulate a long-only strategy that invests in the top-ranked stocks based on model confidence. Specifically, we evaluate threshold-based strategies that allocate capital to the top K stocks with the highest predicted positive class probability, rebalancing daily and assuming transaction costs of 1 basis point per trade. The value of K is optimized within each model to maximize the out-of-sample annualized Sharpe ratio.

Table 3.5 reports the best-performing configurations. The LSTM model achieves the highest Sharpe ratio of 0.5595 with an absolute K around 439, followed closely by nearby configurations. The CNN model, using a relative K, also performs competitively with a Sharpe ratio of 0.5590. Although differences are small in magnitude, they are persistent across test periods and sensitive to the method of long signal selection.

3.5 Discussion

All models demonstrate some degree of predictive power relative to the benchmark. The LSTM model is particularly effective when paired with absolute thresholding, improving Sharpe ratios by filtering for the most confident predictions. These results suggest that the model's confidence scores are meaningful in distinguishing more likely winners.

The CNN model produces stable but slightly weaker results in both classification and backtesting metrics. Its performance improves when relative thresholds are used, indicating adaptability to cross-sectional dispersion. The hybrid CNN-LSTM model

Model	Strategy Type	K	Annualized Sharpe
LSTM	Long K (absolute)	439	**0.5595**
LSTM	Long K (absolute)	427	0.5593
LSTM	Long K (absolute)	415	0.5592
LSTM	Long K (absolute)	436	0.5591
CNN	Long K (relative)	0.97	0.5590
LSTM	Long K (absolute)	413	0.5588
LSTM	Long K (absolute)	429	0.5587
LSTM	Long K (absolute)	432	0.5585
LSTM	Long K (absolute)	434	0.5584
LSTM	Long K (absolute)	441	0.5583
LSTM	Benchmark	0	0.5569

Table 3.5: Top Sharpe ratio configurations under long-only K-based strategies.

delivers the strongest performance overall, combining the CNN's ability to extract localized patterns with the LSTM's capacity to capture temporal dependencies.

Despite these gains, the performance margins are narrow and subject to noise. Precision and recall remain close to the baseline in several cases, and interpretability is limited. The results highlight the importance of calibration, rolling evaluation, and strategic filtering in unlocking value from model predictions.

3.6 Conclusion

This chapter implemented and evaluated three deep learning models: LSTM, CNN, and a hybrid CNN-LSTM, for the task of predicting next-day cross-sectional stock returns. All models showed improvements over the benchmark when evaluated through both classification metrics and backtesting performance. The CNN-LSTM architecture emerged as the top performer, achieving the highest Sharpe ratio, best classification accuracy, and lowest drawdown.

These findings support the use of neural networks in systematic trading, particularly when outputs are filtered and interpreted with care. Future research may explore enhancements such as attention mechanisms, regime-based training, and explainable modeling frameworks to improve generalization and transparency.

Conclusion

This Applied Master Project investigated three distinct families of algorithmic strategies: rule -based, value-based and neural network. Strategies were applied to a daily dataset with constituents of the S&P 500 index over the 2008-2022 period. Machine Learning implementations are inspired from Krauss et al. (2017) and adapted to the specificity of each strategy. Both evaluation windows and backtesting protocols were uniformed to ensure proper compatibility across approaches and avoid any data leakage.

Strategies displayed mixed results.

Rule-based strategies demonstrated average predictive power, with classification metrics slightly above 50%. Long-only and top-K selection strategies yielded robust risk-adjusted performance, outperforming long-short strategies which suffered from weak short signal reliability. Continuous mappings—particularly logistic and hyperbolic—enhanced strategy performance by emphasizing high-confidence predictions. Overall, rule-based strategies proved effective in identifying outperformers, albeit with limited robustness on the short side.

Value-based strategies with Random Forest regressor displayed weak directional accuracy below 50% and high Root-Mean-Square-Error, failing to adjust to the results obtained with a simple long-only historical strategy. Results were especially dragged down for the short strategies in the Reversal ("one-sided flip") approach. Risk adjusted performance remained robust for long-only strategies, particularly when both fundamental ratios (P/E and P/B) were jointly considered.

Neural network strategies achieved robust predictive and financial performance. Despite their complexity, these models exhibited solid out-of-sample predictive power, with classification metrics above the 50% threshold. While training and interpretability remained challenging, the predictive and financial performance of deep learning models confirmed their potential in capturing complex market dynamics.

Overall, this Applied Master Project highlighted the trade-offs between simplicity, interpretability, and statistical power of the distinct strategies. Rule-based and Value-based models offered transparency and industry-based intuition; while deep learning models provided compelling performance when properly regularized. Future work could investigate hybrid strategies robust risk-adjusted results.

Bibliography

Aggarwal, C. C. (2018). *Neural networks and deep learning: A textbook*. Springer. https://doi.org/10.1007/978-3-319-94463-0

Brock, W., Lakonishok, J., & LeBaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, *47*(5), 1731–1764. https://doi.org/10.1111/j.1540-6261.1992.tb04681.x

Chan, L. K., Jegadeesh, N., & Lakonishok, J. (1996). Momentum strategies. *The Journal of Finance*, *51*(5), 1681–1713. https://doi.org/10.1111/j.1540-6261.1996.tb05222.x

Chen, Z., & Wang, L. (2020). Financial time series forecasting using cnn: A review. *Journal of Risk and Financial Management*, *13*(9), 214.

European Commission. (2014). Markets in financial instruments directive ii (mifid ii) [Directive 2014/65/EU of the European Parliament and of the Council]. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32014L0065

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669. https://doi.org/10.1016/j.ejor.2017.11.054

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org/

Grinblatt, M., & Moskowitz, T. J. (1999). Do industries explain momentum? *The Journal of Finance*, *54*(4), 1249–1290. https://doi.org/10.1111/0022-1082.00146

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91.

Lu, W., Zheng, S., & Zhang, J. (2021). A cnn-lstm-based model to forecast stock prices. *Complexity*, *2021*, 1–14.

Zaheer, M. R., Elkina, H., & Zaki, T. (2024). From technical indicators to trading decisions: A deep learning model combining cnn and lstm. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *15*(6). https://doi.org/10.14569/IJACSA.2024.0150685

Appendix

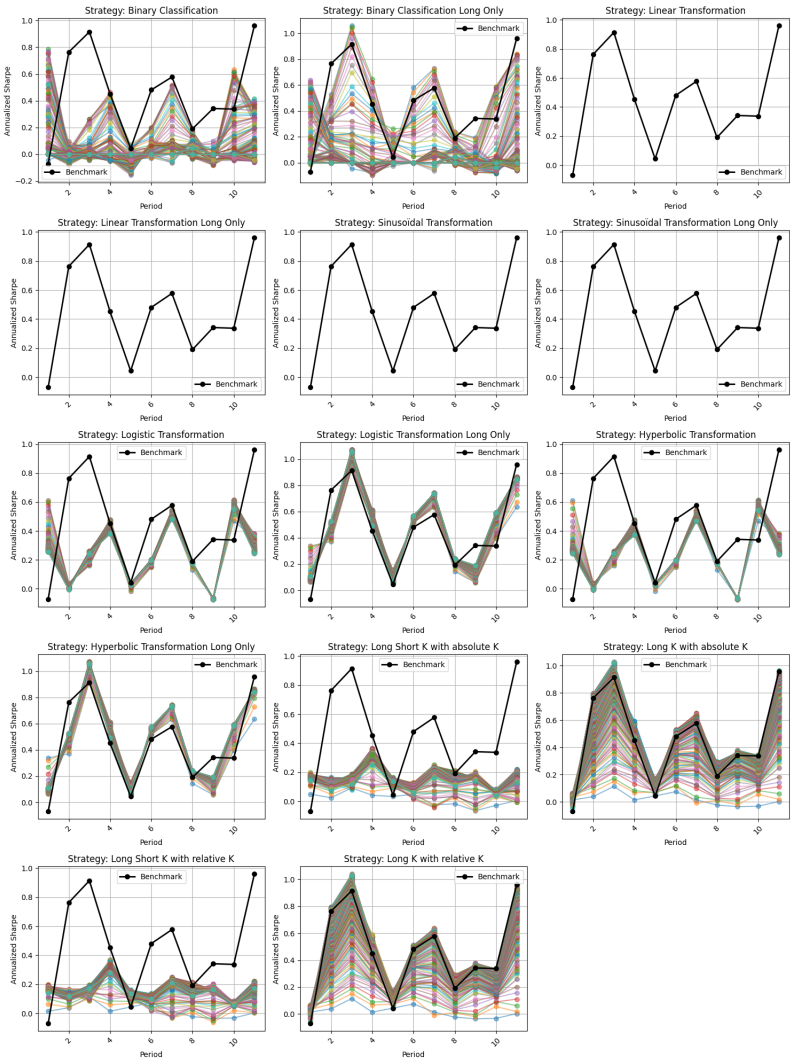

Figure A.1. *Sharpe ratio performance across test periods for all Moving Average strategy variants. Colored lines represent different thresholds. This visualization illustrates the temporal consistency of each strategy class. While performance varies significantly across periods, some strategies—particularly long-only transformations and top-K long-only portfolios—exhibit relatively strong persistence and robustness. In contrast, many long-short strategies display high sensitivity to market conditions.*