NOVA IMS Information Management School

MEGI

Master Degree Program in

Statistics and Information Management

Pairs Trading Strategy with a Machine Learning Approach

A comparison between Traditional Statistical Models and LSTM Neural Networks

Riccardo Andrea Gurzu

Master Thesis

presented as partial requirement for obtaining the Master Degree in Statistics and Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Contents

1 Introduction **1**

2 Literature review **2**
2.1 The CAPM . . . 2
2.2 Market Neutral Strategy . . . 3
2.3 Efficient Market Hypothesis . . . 3
2.3.1 Reversal Effect . . . 4
2.4 Pairs Trading . . . 4
2.5 Time Series . . . 5
2.5.1 Autoregressive Process . . . 5
2.5.2 Stationarity . . . 6
2.5.3 Moving Average Process . . . 9
2.5.4 Autoregressive moving-average Process . . . 9
2.5.5 Order of Integration . . . 9
2.5.6 Cointegration . . . 10
2.6 Neural Networks . . . 13
2.6.1 Network Composition . . . 13
2.6.2 Activation Functions and the Universal Approximation Theorem . . . 14
2.6.3 Loss Function . . . 14
2.6.4 Learning Process . . . 15
2.6.5 Backpropagation Algorithm . . . 16
2.6.6 Practical Issues in Learning . . . 16
2.6.7 Training Algorithms . . . 17
2.6.8 Hyperparameters . . . 18
2.6.9 Recurrent Neural Networks . . . 19
2.6.10 Long-Short Term Memory Networks . . . 20

3 Methodology **22**
3.1 Overview . . . 22
3.2 Data Collection and Preparation . . . 23
3.3 Pairs Selection Using Cointegration Analysis . . . 25
3.4 Spread Modeling . . . 26
3.4.1 ARMA Model . . . 27
3.4.2 LSTM Neural Network . . . 27
3.5 Training and Validation . . . 29
3.6 Backtesting the Trading Strategy . . . 30

4 Results **32**
4.1 Analysis of Model Performance During Market Turmoil 33
4.2 Risk Management Analysis 35

5 Discussion **36**
5.1 Model Performance and Market Conditions 36
5.2 Limitations, Delimitation and Further Research 36
5.3 Conclusion 38

Bibliography **39**

List of Figures

2.1 Stationary and Non-Stationary Time Series 7
2.2 Cointegrated Time Series 11
2.3 Example of underfitting and overifitting 16
2.4 LSTM neural network architecture (Zhang et al. 2021) 20

3.1 Adjusted Closing Prices 24
3.2 DJIA Cumulative Returns 25
3.3 Spread percentage change distribution 27
3.4 Data partitions periods 29
3.5 Cumulative returns 31

List of Tables

2.1 ADF Test Results for Series A and Series B 8
2.2 Cointegration Test Results 12
2.3 ADF Test Results for AAPL, MSFT, and the Spread 12

4.1 Yearly performance Metrics of Index, ARMA, and LSTM Models 33
4.2 Average performance Metrics of Index, ARMA, and LSTM Models 35

Abstract

Pairs trading, a statistical arbitrage strategy, capitalizes on the mean-reverting behavior of a pair of securities exhibiting similar market dynamics. This study delves into enhancing the pairs trading strategy within the Dow Jones Industrial Average Index by comparing the effectiveness of traditional statistical methods and Long-Short Term Memory (LSTM) Neural Networks in forecasting spread movements. The research adopts a bottom-up approach, employing the Engle-Granger Test for cointegration to identify potential pairs within the index. Subsequently, the ARMA Model, alongside LSTM Neural Networks, are utilized to forecast spread behavior. The comparison of these methods aims to discern whether their application leads to improved performance compared to merely tracking the index. The analysis unveils the predictive capabilities, advantages, and limitations of each method, shedding light on their efficacy in enhancing the pairs trading strategy within the Dow Jones Industrial Average.

Resumo

O pairs trading, uma estratégia de arbitragem estatística, se aproveita do comportamento de reversão à média de um par de valores mobiliários que exibem dinâmicas de mercado similares. Este estudo se aprofunda em aprimorar a estratégia de pairs trading dentro do Índice Dow Jones Industrial Average, comparando a eficácia de métodos estatísticos tradicionais e Long-Short Term Memory (LSTM) Neural Network na previsão dos movimentos de spread. A pesquisa adota uma abordagem de baixo para cima, utilizando o Teste de Engle-Granger para cointegração a fim de identificar pares potenciais dentro do índice. Posteriormente, o Modelo ARMA, juntamente com LSTM Neural Network, são utilizados para prever o comportamento do spread. A comparação desses métodos visa discernir se sua aplicação leva a um desempenho melhorado em comparação a apenas acompanhar o índice. A análise revela as capacidades preditivas, vantagens e limitações de cada método, lançando luz sobre sua eficácia em aprimorar a estratégia de pairs trading dentro do Dow Jones Industrial Average.

Keywords

Pairs Trading; Dow Jones Industrial Average (DJIA); ARMA Model; LSTM Neural Networks; Statistical Arbitrage; Time Series Analysis

Chapter 1

Introduction

Pairs trading is a statistical arbitrage trading strategy that involves a pair of securities that shows a similar behaviour in the market. In this case, it could be assumed that the spread between the two securities follows a mean-reverting process, which means that any deviations from the spread's mean will ultimately disappear. Once identified the pair, when this similar behaviour becomes weaker and the price of one asset increases while the other one decrease, a market opportunity is created. We then buy the comparatively inexpensive stocks and sell the relatively costly assets, and waiting for the spread to return to its mean level to profit (Vidyamurthy, 2004). This strategy brings three fundamental advantages: the short and the long position for the same value allows to build a self-financing portfolio; the strategy is neutral against the market dynamics; and lastly, while is difficult to predict the value of an asset, it's feasible to forecast the relative difference of the two asset prices as proofed in Bossaerts and Green (1989) and Jagannathan and Viswanathan (1988) within an equilibrium asset-pricing framework with nonstationary common factors.For this last reason, a key role of the pairs trading strategy is the selection of the mean-reverting process used to describe the trend of the spread so that entering and exiting trading signals can be developed from that model. The main object of this work is to deepen the pairs trading strategy in the context of the Dow Jones Industrial Average Index, investigating whether applying traditional statistical methods and Long-Short Term Memory Neural Networks for forecasting the spread can lead to a better performance compared to simply tracking the Index, which will be used as a benchmark, and which method performs better in this context.

In this research, a bottom-up approach will be used to identify potential pairs within the Dow Jones Industrial Average Index utilizing the Engle-Granger Test for cointegration to evaluate the relationship between these pairs. The spread, instead, will be forecasted using both traditional statistical methods (in this particular research will be used the ARMA Model) and LSTM Neural Networks, which captured an increasing attention towards its ability to understand complex pattern and adapt to changing market conditions to predict future market trend, deepen in the work of Flori and Regoli (2021) [16] in the context of pairs trading. With this study will be presented the results obtained from ARMA and LSTM Neural Network to analyse their respective predictive capabilities with their potential advantages and limitation regarding the performance of the pairs trading strategy when applied to the DJIA.

Chapter 2

Literature review

To understand how the Pairs Trading Strategy is located in the economical theory, is important to fix two topic: How the Market Neutral Strategy works and the Efficient Market Hypothesis.

The Market Neutral Strategy relies on an impactful model in finance which formalized the notion of a market portfolio: the Capital Asset Pricing Model.

2.1 The CAPM

Firstly proposed by William T. Sharpe (1964), the model introduced the notion of *beta*, attempting to explain asset returns as an aggregate sum of component returns, separating it in two components: one is the systematic component (or market component) and the other is the residual component (or nonsystematic component):

$$r_p = r_f + \beta(r_m - r_f), \tag{2.1}$$

where r_p is the return on the asset, r_f represents the risk-free rate, r_m is the return on the market portfolio. In the empirical version it's included also an error term ϵ_p, representing the unexplained return on the portfolio. If we consider the equation in geometrical terms, we can see the β as the slope of the line (also called Security Market Line (SML)), representing the leverage number of the number of the asset return over the market return.

For this reason is possible to estimate the value of beta as the slope of the regression line between market returns and the asset returns using the standard regression formula:

$$\beta = \frac{\text{cov}(r_p r_m)}{\text{var}(r_m)} \tag{2.2}$$

Based on the definition of beta, we can then imply that when the market goes up, we can typically expect the price of all securities to go up with it, assuming that beta is positive. However, beta can also be negative if the covariance between the asset returns and the market returns is negative, indicating that the asset moves in the opposite direction to the market. Thus, a positive return for the market usually implies a positive return for the asset, that is, the sum of the market component and the residual component is positive. If the residual component of the asset return is small, as we expect it to be, then the positive return in the asset is explained almost completely by its market component.

Therefore, a positive return in the market portfolio and the asset implies a positive market component of the return and, by implication, a positive value for beta. Therefore, we can expect all assets to typically have positive values for their betas. (Vidyamurthy (2004))

2.2 Market Neutral Strategy

Definition 1 (Market Neutral Strategies) *They are strategies that are neutral to market returns, that is, the return from the strategy is uncorrelated with the market return.*

Based on this definition we can imply that these strategies perform consistently regardless the market movements achieved trading market neutral portfolios.

In the CAPM context, these particular type of portfolios are characterized by a beta equals to zero, which based on the Equation 2.1, it means that the return of the portfolio is determined only by the residual component θ_p, thus being uncorrelated with market returns. For this reason a zero beta portfolio qualifies as a market neutral portfolio.

Since the consensus expectation or mean on the residual return is zero, it is reasonable to expect a strong mean-reverting behavior (value oscillates back and forth about the mean value) of the residual time series. This mean-reverting behavior can then be exploited in the process of return prediction, leading to trading signals that constitute the trading strategy.

2.3 Efficient Market Hypothesis

The Efficient Market Hypothesis (EMH) theory is a fundamental hypothesis in the context of financial economics. It states that markets are efficient, meaning that the price of assets traded in the market reflects and incorporates all relevant available information (Fama, 1970). Therefore, the historical performance of stock prices cannot be used to predict future performance. Stock market movements would be solely determined by future and unpredictable information. It also follows that stock returns are memoryless stochastic processes, and prices react immediately when new information becomes available to the market. The perspective of systematically "beating the market" is not realistic, as stocks are traded at their correct (fair) value.

Over time, investors and researchers have widely contested the efficient market theory both theoretically and empirically, demonstrating that the market is imperfect because in reality, there are phenomena, defined as market anomalies, through which predictions can be made even in highly liquid markets. For example, behavioral economists attribute these imperfections to a combination of cognitive biases, such as overconfidence, exaggerated reactions, or other irrational human behaviors in the process of information analysis (Jegadeesh and Titman, 1995). The combination of these factors can make price movements predictable. If the market is imperfect, in some situations, it is possible to take advantage of market vulnerabilities to achieve abnormal positive returns.

2.3.1 Reversal Effect

One of the most studied phenomena along this "market imperfections" (Fama (1965), Jegadeesh (1990), and Jegadeesh and Titman (1993)) is the *reversal effect*, according to which stocks that have had poor recent performance are likely to experience a reversal in price trend in the short term, and vice versa. In this sense, past market trends can influence market participants' expectations. The main explanation provided for this phenomenon is that stock prices react excessively to new information about companies, leading to temporary mispricing and subsequent price correction in the opposite direction (Jegadeesh and Titman, 1995). There is also another explanation related to liquidity effects, which suggests that profits from these reversals are compensation for market makers in exchange for providing liquidity and assuming inventory risk (Avramov et al., 2006).

The common denominator of the two motivations for the reversal effect is that past trends can influence market participants' behavior and make future trends predictable. This allows for the creation of contrarian strategies, characterized by buying and selling in opposition to the stock's current trend. In other words, a trader is inclined to sell stocks that have had good performance and buy stocks that have performed poorly, expecting a reversal in trend.

2.4 Pairs Trading

Pairs trading is a trading strategy that involves matching a long position with a short position in two stocks that exhibit a high correlation. This market-neutral strategy enables the operator to profit from any market condition — whether it's upward, downward, or stagnant — by exploiting the concept of mean reversion. The strategy was generally attributed to a Morgan Stanley trader, Nunzio Tartaglia, during the 1980s. Tartaglia and his team, which included mathematicians, physicists, and computational analysts, developed this quantitative and automated arbitrage strategy by trading stocks in pairs (Vidyamurthy, 2004).

In pairs trading, pairs of similar stocks, typically those with parallel past performance, are identified. The strategy capitalizes on situations where the stocks of the pair show opposite market trends. Specifically, opportunities are sought when this similarity weakens—when the price of one stock increases while the other decreases. The trader then buys the stock that has decreased in value and sells short the stock that has increased. Divergences within the pair can be triggered by various factors, such as temporary changes in supply and demand, significant buy or sell orders for one of the stocks, or reactions to important news affecting one of the companies.

Assuming that future performances will mirror past ones, the "incorrect" relative price is viewed as temporary. At the moment when the prices converge again, a profit is realized. This occurrence is interpreted as a reversion to the mean of the relative price.

The advantages of this strategy are threefold:

- The long and short positions in the pair, being of the same value, allow for the construction of a portfolio that is essentially self-financing.
- The portfolio remains neutral to overall market dynamics; should both

securities in the pair experience the same positive or negative shock, the resultant losses and gains would offset each other.

- While predicting the prices of individual stocks can be challenging, there is strong evidence to suggest that forecasting the relative price difference between a pair is feasible.

For these reasons, pairs trading has been extensively studied over time by researchers and professionals who continue to develop numerous techniques based on the idea of exploiting the relative value of two correlated assets.

2.5 Time Series

A time series can be defined as the set of values that a variable Y assumes over time. Considering a time period T, the time series Y can be defined as the set of values $Y = \{Y_1, Y_2, Y_3, ..., Y_T\}$, so we can denote the value in a specific time t as Y_t with $t = 0, 1, 2, 3, ..., T$.

Given a moment t, the value of Y in the previous period is called the first lag and is indicated by Y_{t-1}, and similarly, the j-th lag is the value taken by Y j moments before, namely Y_{t-j}. Finally, the first difference is defined as the change in the value of Y between moment t and moment $t-1$, that is, $\Delta Y_t = Y_t - Y_{t-1}$.

These values may be derived from a fixed deterministic formula, in which case they are referred to as a deterministic time series. Alternately, the value may be obtained by drawing a sample from a probability distribution, in which case they may be termed as probabilistic or stochastic time series.

Given this set of values overtime is possible to measure the dependency of Y in two different close moments thought a measure called *autocorrelation*, and can be defined to any lag:

$$\rho_j = \text{corr}(Y_t, Y_{t-j}) = \frac{\text{cov}(Y_t, Y_{t-j})}{\sqrt{\text{var}(Y_t)\text{var}(Y_{t-j})}}, \quad j = 1, 2, ..., T. \tag{2.3}$$

This quantity expresses how much a value Y_t of the series is influenced by the value of the series j moments earlier, and whether this influence is on average positive or negative.

2.5.1 Autoregressive Process

Autocorrelation forms the basis of the concept of autoregression, which is a model that relates a time series with its past values in order to make predictions about future values. The simplest autoregressive (AR) model is called the first-order model (denoted $AR(1)$), and it involves performing ordinary least squares (OLS) regression of the values Y_t of the series on the lagged values by one time step, Y_{t-1}. It's termed autoregressive because the regression occurs on its own lagged values. In formula:

$$Y_t = \beta Y_{t-1} + u_t \tag{2.4}$$

where β is the parameter of the model, and u_t is an error term.

However, the $AR(1)$ model only uses one lagged value and therefore risks disregarding potentially interesting information from further moments in the past. This issue can be easily addressed by generalizing the model to include p lagged values. The latter is defined as an autoregressive model of order p and denoted as $AR(p)$. In formula:

$$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + ... + \beta_p Y_{t-p} + u_t = \sum_{i=1}^{p} \beta_i Y_{t-i} + u_t \tag{2.5}$$

where u_t represents error terms satisfying $E(u_t \mid Y_{t-1}, Y_{t-2}, ...) = 0$ for every t, and are not serially correlated with each other. The process $\{u_t\}$, additionally, is i.i.d. with mean 0 and constant variance equal to σ_u^2.

2.5.2 Stationarity

Considering a stochastic process that can describe our time series, if the future behaves in the same way as the past, historical data can be used to predict future data. However, if the future significantly differs from the past, historical relationships would no longer be reliable guides for the future. This concept is expressed by the definition of *stationarity*.

Definition 2 (Stationarity) *A time series is stationary if its probability distribution does not change over time, meaning that the joint distribution of* $(Y_{s+1}, Y_{s+2}, ..., Y_{s+\tau})$ *does not depend on* s *for every value of* τ*; otherwise,* Y_t *is called non-stationary."*

Dickey-Fuller Test

To identify a stochastic trend in a time series is possible to rely on quantitative statistical procedures to test the hypothesis of a random trend against the alternative hypothesis that the trend is not present. The primary test used for this purpose, and the one used in this work, is the Dickey-Fuller test (Dickey and Fuller, 1979), which is widely acknowledged in econometric literature for its robustness in detecting unit roots in autoregressive models.

This test is used for $AR(1)$ models and tests the presence or absence of a unit root in the model. More specifically, it tests:

$$H_0 : \beta_1 = 1 \quad \text{vs} \quad H_1 : \beta_1 < 1 \quad \text{in} \quad Y_t = \beta_0 + \beta_1 Y_{t-1} + u_t.$$

The test can be rewritten by subtracting Y_{t-1} from both sides of the equation on the right, obtaining:

$$H_0 : \delta = 1 \quad \text{vs} \quad H_1 : \delta < 1 \quad \text{in} \quad \Delta Y_t = \beta_0 + \delta Y_{t-1} + u_t,$$

where $\delta = \beta_1 - 1$.

This test can also be extended to $AR(p)$ models. In this case, it is called the Augmented Dickey-Fuller test, ADF, and tests the hypothesis:

$$\begin{aligned} &H_0 : \delta = 0 \quad \text{vs} \quad H_1 : \delta < 0 \quad \text{in} \\ &\Delta Y_t = \beta_0 + \delta Y_{t-1} + \gamma_1 \Delta Y_{t-1} + \gamma_2 \Delta Y_{t-2} + .. + \gamma_p \Delta Y_{t-p} + u_t \end{aligned} \tag{2.6}$$

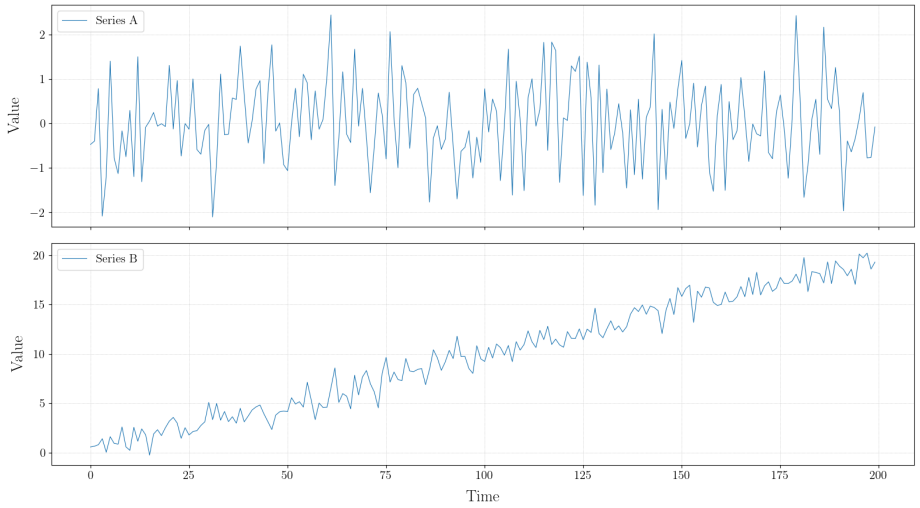

Figure 2.1: Stationary and Non-Stationary Time Series

In the example shown in Figure 2.1, two time series are generated and analyzed for stationarity. The first time series, Series A, is generated using white noise, where each data point is drawn independently from a normal distribution with a mean of 0 and a standard deviation of 1. This series exhibits random fluctuations around a constant mean value of 0 and consistent variance over time, which is characteristic of white noise.

The second time series, Series B, is generated with a non-stationary trend, where each data point is drawn from a normal distribution with a mean that increases linearly over time (specifically, with a mean of $t \times 0.1$ for time step t) and a standard deviation of 1. This introduces a trend in the data, making it non-stationary.

To statistically confirm the stationarity of these series, the Augmented Dickey-Fuller (ADF) test is performed on both. The ADF test checks for the presence of a unit root, with the null hypothesis (H_0) being that the time series is non-stationary (i.e., it has a unit root), and the alternative hypothesis (H_1) being that the time series is stationary.

For Series A, the ADF test produces an ADF statistic of approximately -7.88 and a p-value of 4.779×10^{-12}. The critical values for different confidence levels are as follows: -3.464 for 1%, -2.876 for 5%, and -2.575 for 10%. Given this significantly negative value and p-value < 0.05, the null hypothesis is rejected, concluding that Series A is indeed stationary. This result is expected, given the nature of white noise, as it inherently has properties of stationarity, with its mean and variance being constant over time. Thus, the statistical test corroborates the visual and theoretical understanding of Series A.

For Series B, the ADF test produces an ADF statistic of approximately -0.02 and a p-value of 0.957. The critical values for different confidence levels are as follows: -3.466 for 1%, -2.877 for 5%, and -2.575 for 10%. Given that the test statistic is much less negative than the critical values and p-value > 0.05, the null hypothesis cannot be rejected. This indicates that Series B is not stationary,

which aligns with the fact that Series B was generated with an increasing trend over time.

Metric	**Series A**	**Series B**
ADF Statistic	-7.8783	-0.0210
p-value	4.7791e-12	0.9568
Critical Values		
1%	-3.464	-3.466
5%	-2.876	-2.877
10%	-2.575	-2.575

Table 2.1: ADF Test Results for Series A and Series B

Phillips-Perron Test

An alternative to the Dickey-Fuller tests is the Phillips-Perron (PP) test, which is an advanced method used to detect unit roots in time series data, extending the capabilities of the traditional Dickey-Fuller tests. Developed to accommodate a wider range of autocorrelation and heteroscedasticity, the PP test enhances the testing procedure without altering the fundamental structure of the model. Like the Dickey-Fuller approach, the Phillips-Perron test analyzes whether the coefficient associated with the lagged level of a series in a regression model is zero, which would indicate a unit root and, consequently, non-stationarity.

The core regression equation utilized in the PP test remains straightforward:

$$\Delta Y_t = \alpha + \beta Y_{t-1} + \epsilon_t,$$

where ΔY_t is the first difference of the series Y_t, α is a constant, β is the coefficient on the lagged level of the series, and ϵ_t is the error term. However, what distinguishes the Phillips-Perron test is its method of adjusting the t-statistics used to test the null hypothesis of $\beta = 0$. This adjustment is crucial as it accounts for any serial correlation and heteroscedasticity in the residuals, making the test more robust against variations in error structures.

By using a nonparametric method to estimate the spectrum at zero frequency of the residuals (also known as the long-run variance) the PP test modifies the conventional test statistics. This technique ensures that the statistical properties of the error terms do not unduly influence the outcome of the test, thereby providing a more reliable measure of unit root presence.

The Phillips-Perron test offers distinct advantages, particularly in its ability to handle complex error structures without the need for specifying the number of lags in the model, as is required by the Augmented Dickey-Fuller test. This simplification in model specification, combined with its robustness to various forms of error term behaviors, makes the PP test especially valuable in econometric analyses where the error terms are suspected of having non-constant variance or intricate autocorrelation patterns.

However, the test is not without limitations. It may suffer from size distortions if the residuals include significant moving average components, and its power can be compromised in the presence of structural breaks within the series. These challenges suggest that while the Phillips-Perron test is a powerful

tool for detecting unit roots, it should be applied with consideration of potential biases and, where appropriate, supplemented with additional testing to confirm findings.

2.5.3 Moving Average Process

Rather than using past values of the forecast variable in a regression, like in the Autoregressive Process, a Moving Average process uses past forecast errors in a regression-like model.

The Moving Average Process of order q, noted as $MA(q)$, is described by the following equation:

$$Y_t = u_t + \theta_1 u_{t-1} + ... + \theta_q u_{t-q} = \sum_{i=1}^{q} \theta_i u_{t-i} + u_t, \tag{2.7}$$

where $\theta_1, ..., \theta_q$ are the parameters of the model and $u_t, u_{t-1}, ..., u_{t-q}$ are the errors term where the process $\{u_t\}$ is i.i.d. with mean 0 and constant variance equal to σ_u^2. Since the MA is a linear combination of past u_t and, additionally, the autocovariance function of the MA process decreases to zero as the time lag between observations increases, these combined properties ensure that the statistical characteristics of the process do not change over time, leading to stationarity. Thus, the moving average process is always stationary, facilitating its analysis and modeling in time series studies.

2.5.4 Autoregressive moving-average Process

Combining the Autoregressive (AR) process and the Moving Average (MA) process, we obtain a new process, called $ARMA$, which provides a description of a weakly stationary stochastic process in terms of two polynomials (the first one for the AR and the second for the MA):

$$Y_t = \sum_{i=1}^{p} \beta_i Y_{t-i} + \sum_{i=1}^{q} \theta_i u_{t-i} + u_t \tag{2.8}$$

An important requirement is that the time series $\{Y_t\}$ is stationary. Stationarity is a crucial condition for the ARMA model because it ensures the validity of model assumptions, maintains stable statistical properties, facilitates parameter interpretation, and enhances model performance. Therefore, ensuring stationarity in the time series data is typically a prerequisite for using the ARMA model effectively. For a more detailed discussion on ARMA models, see Hamilton (1994).

2.5.5 Order of Integration

Since in reality time series have generally a more regular trend, meaning that they don't vary so much from one moment to another, is possible to build a model to describe this type of time series where the first difference follows a random walk, that is:

$$\Delta Y_t = \beta_0 + \Delta Y_{t-1} + u_t, \tag{2.9}$$

where u_t is i.i.d. and such that $E(u_t \mid Y_{t-1}, Y_{t-2}, ...) = 0$ and non-autocorrelated. If ΔY_t follows a random walk, then its difference, called the second difference and denoted as $\Delta^2 Y_t$, is stationary. At this point, it is useful to introduce additional terminology to distinguish between different stochastic trends. A series that follows a random walk, whose difference is stationary, is defined as integrated of order 1, or $I(1)$. A series whose first difference follows a random walk is defined as integrated of order 2, or $I(2)$, and so forth. If the series does not exhibit any stochastic trend, i.e., it is stationary, it is defined as integrated of order 0, or $I(0)$. The order of integration represents the number of times the series needs to be differenced to become stationary.

2.5.6 Cointegration

It can happen in reality that two different time series share the same random trend, causing them to move together, i.e., in the same direction over a large time interval, but their difference may not appear to have any kind of trend. When this characteristic is observed, the two time series are said to be *cointegrated*.

Definition 3 (Cointegration) *Given k time series $\left\{Y^{(1)}, Y^{(2)}, ..., Y^{(k)}\right\}$, all integrated of order d, these are called cointegrated if there exist k coefficients $\theta^{(1)}, \theta^{(2)}, ..., \theta^{(k)}$ such that $\theta^{(1)}Y^{(1)}, \theta^{(2)}Y^{(2)}, ..., \theta^{(k)}Y^{(k)}$ is integrated of order less than d. In this case, $\theta^{(1)}, ..., \theta^{(k)}$ are called cointegration coefficients.*

Cointegration can be used as an alternative method to calculating differences of a series to remove a common stochastic trend from at least two series. This renders the linear combination stationary and therefore usable for regression analysis.

If we are given two series that are deemed nonstationary but are cointegreted, their difference will be a stationary process. An approach to provide a theoretical basis for this relationship is attributed to Stock and Watson, called the *common trends model*. The primary idea of the common trends model is that of a time series being expressed as a simple sum of two component time series: a stationary component and a nonstationary component. If two series are cointegrated, then the cointegrating linear composition acts to nullify the nonstationary components, leaving only the stationary components. To see what we mean, consider two time series:

$$y_t = n_{y_t} + \varepsilon_{y_t}$$
$$z_t = n_{z_t} + \varepsilon_{z_t}$$

where n_{y_t} and n_{z_t} are the random walk (nonstationary) components of the two time series, and ε_{y_t} and ε_{z_t} are the stationary components of the time series. Also, let the linear combination $y_t - \gamma z_t$ be the cointegrating combination that results in a stationary time series. Expanding the linear combination and rearranging some terms, we have:

$$y_t - \gamma z_t = (n_{y_t} - \gamma n_{z_t}) + (\varepsilon_{y_t} - \gamma \varepsilon_{z_t}) \tag{2.10}$$

If the combination in Equation 2.10 must be stationary, the nonstationary component must be zero, implying that $n_{y_t} = \gamma n_{z_t}$, or the trend component of one series must be a scalar multiple of the trend component in the other series.

Therefore, for two series to be cointegrated, the trends must be identical up to a scalar.

To be able to identify two cointegreted time series is possible perform some statistical tests, where the most popular one is the Engle-Granger Test.

The Engle-Granger test (Engle and Granger, 1987) consists of two different steps: in the first step, the cointegration vector is estimated through the following OLS regression:

$$Y_t^{(1)} = \alpha + \theta Y_t^{(2)} + u_t \tag{2.11}$$

Equation 2.11 constitutes a model called a distributed lag model, where the current value of the dependent variable is based on the current value and the lagged values of an explanatory variable.

In the second step, an ADF test for unit root is performed on the residuals of regression 2.11. If the residuals are stationary, it means that the two time series share the same stochastic trend, which has been eliminated through the difference $Y_t^{(1)} - \theta Y_t^{(2)}$, and thus they are cointegrated.

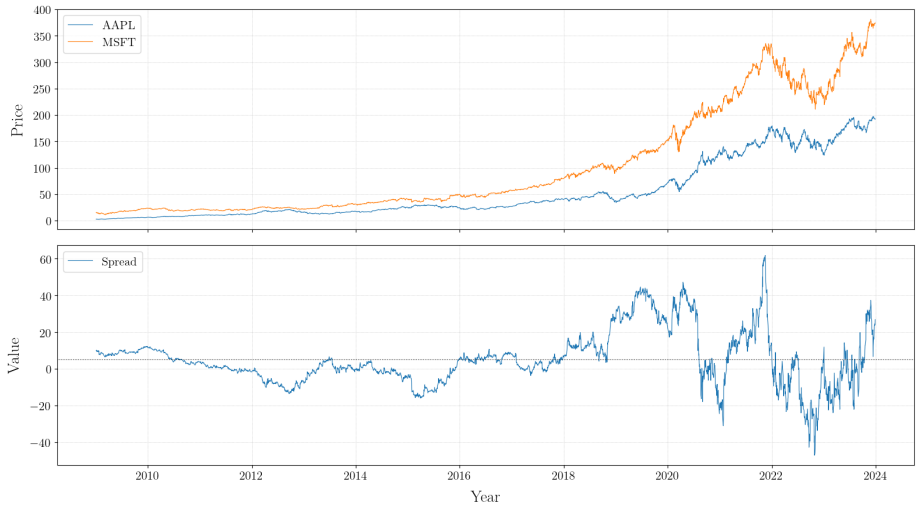

Figure 2.2: Cointegrated Time Series

In the example shown in Figure 2.2, two time series are analyzed: the first time series, represented by the adjusted closing prices of Apple Inc. (AAPL), and the second time series, represented by the adjusted closing prices of Microsoft Corp. (MSFT). These series are downloaded for the period from January 1, 2009, to January 1, 2024.

The spread between the prices of AAPL and MSFT is calculated using linear regression, where the slope (beta) of the regression is used to create a linear combination of the two series. This ensures that both time series move together over time due to the shared random trend, but the difference between them is driven by the stationary noise component and does not exhibit any trending behavior.

To statistically confirm the cointegration, the Engle-Granger test is performed. In this case, the cointegration test produces a test statistic of -3.68 and

a p-value of 0.019. The critical values at the 1%, 5%, and 10% confidence levels are -3.90, -3.34, and -3.05, respectively. Given that the test statistic is more negative than the critical value at the 5% level, and p-value < 0.05, the null hypothesis of no cointegration is rejected. This result indicates that the time series are indeed cointegrated. The test statistic being more negative than the critical value at the 5% level and the low p-value provide strong evidence that the two time series share the same stochastic trend, confirming their cointegration.

Metric	**Value**
Engle-Granger Statistic	-3.6819
p-value	0.0193
Critical Values	
1%	-3.8993
5%	-3.3377
10%	-3.0456

Table 2.2: Cointegration Test Results

As proof of the common trends model, the ADF test is performed on the AAPL series, MSFT series, and the spread of the two series. The results of the ADF tests are presented in Tables 2.3. For the AAPL and MSFT series, the ADF test statistics are approximately 1.9929 with p-values of 0.9987. These high p-values indicate that the null hypothesis of non-stationarity cannot be rejected for either series, implying that both the AAPL and MSFT time series are non-stationary.

However, when performing the ADF test on the spread between AAPL and MSFT, the test yields an ADF statistic of approximately -3.5846 with a p-value of 0.0061. This low p-value indicates that the null hypothesis of non-stationarity is rejected, and we conclude that the spread of the two series is stationary. The critical values for the ADF test at the 1%, 5%, and 10% confidence levels are -3.432, -2.862, and -2.567, respectively. The ADF statistic for the spread is more negative than all these critical values, further confirming the stationarity of the linear combination.

These results demonstrate that while the individual time series (AAPL and MSFT) are non-stationary, their spread is stationary, thus providing strong evidence for cointegration and supporting the common trends model.

Metric	**AAPL**	**MSFT**	**Spread**
ADF Statistic	1.9929	1.9929	-3.5846
p-value	0.9987	0.9987	0.0061
Critical Values			
1%	-3.432	-3.432	-3.432
5%	-2.862	-2.862	-2.862
10%	-2.567	-2.567	-2.567

Table 2.3: ADF Test Results for AAPL, MSFT, and the Spread

2.6 Neural Networks

Artificial Neural Networks (ANN), commonly referred to as Neural Networks (NN), are sophisticated machine learning models designed to identify patterns and make predictions based on data. These networks are comprised of *layers* of interconnected computational units, known as *neurons* or *nodes*, linked by weighted connections. Each connection carries a weight that is adjusted during the learning process to minimize prediction errors.

A neural network processes input data by passing it through these layers, using the weighted connections to transform the input at each stage. The learning process, known as supervised learning, involves modifying these connection weights to reduce the discrepancy between the predicted and actual outputs. Mathematically, the objective of a neural network is to approximate an unknown function f^* with a model $y = f(x; w)$, where x represents the input data and w denotes the weights or parameters that the network adjusts to best fit the function f^*.

Neural networks and artificial intelligence are becoming increasingly integral to various fields, from finance to healthcare, due to their ability to handle large datasets and uncover intricate relationships within the data. This rapid advancement has also sparked discussions about the ethical implications and potential risks associated with these technologies.

The following sections delve into the primary components and functions of these computational models, with a particular focus on the application of long-short term memory networks in trading strategies. The primary references for this discussion are the comprehensive works by Goodfellow et al. (2016) and Aggarwal (2018), which provide an in-depth exploration of the subject.

2.6.1 Network Composition

Neural networks are powerful tools capable of modeling complex functions. The simplest form of a neural network is known as a *perceptron*, which consists of a single computational unit. In this model, the input data is mapped directly to an output using an activation function, described by the equation:

$$x \to \Phi(w'x + b),$$

where w is the vector of weights, b is the bias term that shifts the input data, and Φ is the activation function.

In practice, neural networks are composed of many such units, interconnected in various ways. The strength of these connections is governed by the weights w. More complex networks are created by increasing the number of neurons within a layer or by adding more layers. Neurons are organized in a layered structure, with input and output layers separated by intermediate layers known as *hidden layers* and the number of hidden layers defines the network's depth.

A common type of neural network is the *feed-forward neural network*, where connections between the nodes do not form cycles. In this architecture, the information moves in one direction only, from the input nodes, through the hidden nodes, and finally to the output nodes. Each layer in a feed-forward network is fully connected to the next layer, meaning each neuron in one layer is connected to every neuron in the next layer. This structure allows the network

to learn complex representations of the data by gradually transforming the input data layer by layer through non-linear activation functions.

2.6.2 Activation Functions and the Universal Approximation Theorem

Activation functions introduce non-linearities into a neural network, enabling it to learn complex patterns in data. Typically, all neurons in a layer use the same activation function, though different layers may use different functions.

There are various types of activation functions, each with distinct characteristics, chosen based on the specific application. Three activation functions are usually utilized: sigmoid, tanh, and softmax.

The sigmoid function is defined as:

$$\Phi(x) = \frac{1}{1 + \exp(-x)} \tag{2.12}$$

Its output range is (0, 1), which is useful for interpreting output values as probabilities.

The tanh function:

$$\Phi(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{2.13}$$

resembles the sigmoid function but returns values in the range (-1, 1), making it suitable for scenarios where results can be positive or negative.

Lastly, the softmax function is defined as:

$$\Phi(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{k} \exp(x_j)} \tag{2.14}$$

where x_i represents the output value of the i-th node in the final hidden layer, and k is the total number of nodes in the last layer. The softmax function is typically used in the output layer of networks aimed at solving categorical classification problems, where each input is assigned a probability of belonging to one of k classes. In addition to distributing probabilities among various classes, the softmax function assigns larger weights to models with smaller forecasting errors, with the weights decaying exponentially the larger the error, as discussed by Bravo et al. (2021) and Ashofteh et al. (2022).

2.6.3 Loss Function

In mathematical optimization and statistics, the loss function is a crucial component used to measure the discrepancy between an estimated value and the true value. In the context of neural networks, it serves to gauge the accuracy of the approximation of the target function f^* by the function f generated by the network. The choice of this function is critical and highly dependent on the specific application.

In this work, the chosen loss function is the mean squared error (MSE), defined as:

$$L = \frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{y}_j)^2,$$

where y_j is the true value, $\hat{y}_j$ is the predicted value, and N is the number of data points. The mean squared error calculates the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value. This loss function is particularly advantageous for its sensitivity to large errors ensures that significant deviations from actual values are penalized more heavily, which is crucial in time series forecasting where such deviations can be particularly detrimental. This property helps the model to focus on minimizing larger errors, thereby improving overall accuracy. Additionally, the smooth and continuous gradient provided by MSE facilitates efficient training of LSTM networks by offering a clear direction for optimization algorithms like gradient descent, leading to better convergence during training.

Furthermore, MSE is widely recognized and commonly used in regression tasks, where the objective is to predict continuous values, making it a natural fit for time series forecasting problems. Its widespread use and acceptance in the field underscore its reliability and effectiveness in measuring model performance. Thus, the combination of simplicity, sensitivity to large errors, smooth gradient, and common use in regression tasks makes MSE a strong choice for time series analysis with LSTM neural networks, contributing to its popularity and effectiveness in producing accurate and reliable forecasts.

2.6.4 Learning Process

The effectiveness of neural networks is largely due to the learning process, during which the network learns to recognize the features in the input data x that determine the response y. The objective is to minimize the loss function by adjusting all parameters w in the network, which connect the various neurons. This process starts by dividing the dataset into a training set and a test set. The training set is used to calibrate the parameters, while the test set evaluates the network's approximation quality once trained. This evaluation helps assess the network's predictive performance on data not used during the learning process. The optimization problem to be solved is:

$$\hat{w} = \arg\min_{w \in W} L(\{f(w, x_i)\}_{i=1}^{M}, \{f^*(x_i)\}_{i=1}^{M}),$$

where L is the chosen loss function, W represents the parameter space, and M is the number of data points in the training set. Often, the high number of layers and nodes in the network, along with non-linear activation functions, make the loss function a complex composition of non-linear functions, resulting in a high-dimensional, highly non-linear optimization problem, which is challenging to solve using standard gradient descent methods.

To compute the gradient of this composition of functions, an algorithm known as *backpropagation* was developed. This algorithm leverages the chain rule from differential calculus to calculate the gradient. Specifically, it computes the gradient as a sum of the products of local gradients across all possible paths from a neuron to the output. Although the number of paths grows exponentially with the number of layers, this computation can be efficiently managed using a technique called *dynamic programming*.

2.6.5 Backpropagation Algorithm

The backpropagation algorithm, or backward propagation, uses dynamic programming and consists of two phases: forward and backward. In the forward phase, training set data are propagated through the network from input to output. Activation values for each hidden layer are computed using the current weights. The output values are then compared to the true values using the loss function, and the derivatives of the loss function with respect to the output values are calculated.

In the backward phase, the gradients of the loss with respect to all network weights are computed using the chain rule. Starting with the gradient of the loss with respect to the output of the last layer, the gradients with respect to the input of the last layer and the weights connecting different layers are calculated in reverse order, back to the first layer. Once the gradients for all weights are obtained, they are updated following the gradient descent method. The backpropagation algorithm is essential for training neural networks.

It's worth noting that learning typically uses all training set data only once, meaning each data point is propagated through the network and used for gradient estimation at a single moment. Often, multiple iterations of this algorithm are necessary to ensure the network learns all important features in the input data. Each of these iterations is called an epoch.

2.6.6 Practical Issues in Learning

Despite the high reputation of neural networks, many challenges remain, especially during the learning phase. One significant issue is underfitting and overfitting the training data. Underfitting occurs when the model is too simple to capture all relevant features in the data, which can usually be addressed by increasing the network's depth or the number of nodes. Overfitting happens when the neural network is overly complex, resulting in excellent predictive performance on the training data but poor performance on unseen data (test set). This occurs because the network learns random features specific to the training data that do not generalize to new data. An example of overfitting, underfitting, and a good fit for a complex model is shown in Figure 2.3

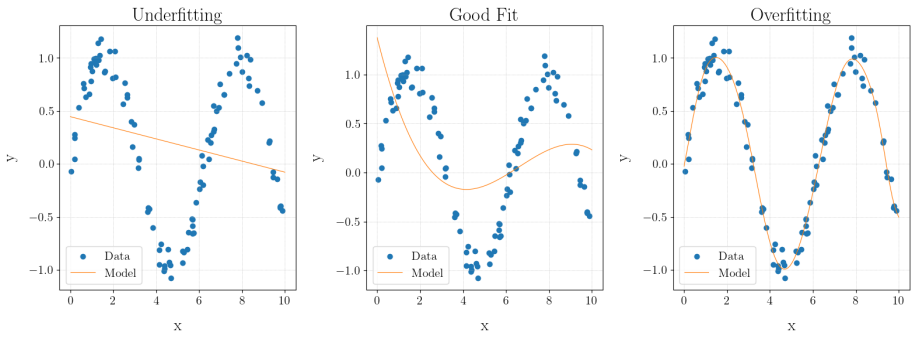

Figure 2.3: Example of underfitting and overifitting

To address overfitting in neural networks, various regularization techniques

have been developed. The two main techniques are penalty-based and early stopping. The first involves modifying the loss function by adding penalties or other constraints to encourage simpler models. The second involves terminating the learning process early, before achieving the optimal solution on the training data. Specifically, a small percentage of the training data, called the validation set, is set aside and not used for training. Instead, it is used to evaluate the error between the true value and the value predicted by the network using the loss function. At the end of each epoch, the network's performance on both the training and validation sets is analyzed. The learning process stops at the epoch where the error on the validation set begins to increase, indicating the onset of overfitting. This technique has the advantage of preventing early overfitting by monitoring performance on a separate validation set.

As previously mentioned, increasing the network's depth enhances its power and predictive capabilities but also leads to instability in weight updates during backpropagation. If the number of layers is very high, weight updates in the initial layers can be either very small (vanishing gradient) or very large (exploding gradient). This occurs due to the results of the chain rule products, which can exponentially increase or decrease along the paths. Consider a network composed of many layers, each with one node. The derivative with respect to the weights in the initial layers is computed as the product of all derivatives with respect to the weights of the successive layers. Consequently, if the expected value of these local derivatives is greater or less than one, the product of the derivatives increases or decreases exponentially, causing excessively large or small weight updates.

In this work, two different activation functions (sigmoid and tanh) are used in the hidden layers, both of which contribute to the vanishing gradient problem: both have small derivative values in saturation regions, with maximum values of 0.25 and 1, respectively.

2.6.7 Training Algorithms

As previously mentioned, the learning algorithm for neural networks is based on minimizing the loss function $L(w, x_i)$, and to achieve this, the gradient descent technique is used, which recursively updates the parameters w according to the equation:

$$w_{t+1} = w_t - \alpha \nabla L_t,$$

where α represents the update step, known as the *learning rate*, and all data in the training set are used to evaluate the loss. This procedure ensures that the weights are updated in the direction opposite to the gradient, i.e., the direction of maximum decrease of the loss function relative to the parameter space. This direction is optimal only for infinitesimal steps; in reality, it can lead to wrong directions or continuous zigzagging. To solve this issue, a step that decreases iteration by iteration, as the algorithm approaches the minimum of the function, is usually proposed.

The classical version of gradient descent described above is rarely used in practice for two reasons: firstly, calculating the gradients for the entire training set is often impossible due to machine memory constraints. Secondly, the steepest descent algorithm does not necessarily converge to a global minimum but converges to a stationary point that generally corresponds to a non-optimal local

minimum. Several solutions exist to address these problems. One such solution is stochastic gradient descent, which evaluates the loss, calculates the gradient, and updates the weights for each data point rather than the entire training set. This significantly reduces memory consumption and the time required for the algorithm to converge to a minimum point, but at the same time, the direction of weight updates is highly variable due to the strong approximation in gradient calculation.

Another alternative is *mini-batch gradient descent*, which involves dividing the training set into several groups, called *mini-batches*, and approximating the true gradient, i.e., calculated on the entire training set, with one calculated on a single batch. In formulas:

$$w_{t+1} = w_t - \alpha \sum_{j \in B} \nabla L_j,$$

where $\{j_1, j_2, \ldots, j_m\}$ are the indices belonging to the current batch B, and L_j indicates the loss evaluated on the j-th data point. This solution often proves to be the best compromise between stability, algorithm speed, and memory usage.

Over time, several adjustments to these algorithms have been developed to avoid numerous local minima and accelerate the learning process, which can be very long. They mainly fall into two categories: momentum-based techniques, which increase the learning rate in directions pointing toward the optimal solution, i.e., those directions that remain consistent between updates; and parameter-specific learning rate techniques, which assign different update steps to different parameters depending on the magnitude of the partial derivatives. The idea is that directions with large partial derivatives tend to oscillate, while those with small partial derivatives tend to remain consistent. Therefore, it is useful to decrease the learning rates for the former and increase them for the latter.

The method used in this work is Adam (Kingma and Ba, 2014), which the authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- The Adaptive Gradient Algorithm (AdaGrad) maintains an individual learning rate for each parameter, enhancing performance on problems with sparse gradients, such as those encountered in natural language processing and computer vision.
- Root Mean Square Propagation (RMSProp) adapts learning rates per parameter based on the recent average magnitudes of the gradients for each weight. This approach is particularly effective for online learning and non-stationary problems, such as those involving noisy data.

2.6.8 Hyperparameters

As discussed, neural networks, through the training process, can assign values that minimize the loss function to the weights associated with the network's connection density. However, there are several parameters, called hyperparameters, that the network cannot choose independently and must be set externally. The term hyperparameter refers to all parameters regulating the model's structure, such as the learning rate and the number of nodes in a layer, fundamentally

different from the weights in the various network layers. Therefore, there is a two-level organization of parameters in neural networks: in the first, hyperparameter values are chosen, and only then are the weights optimized through the backpropagation algorithm.

Hyperparameter selection can be done in two different ways: they can be set manually, or automated tuning procedures can be used to test various combinations. The two most commonly used tuning algorithms are grid search and random search. Grid search involves selecting a set of values for each hyperparameter and testing all possible combinations, choosing the optimal one, i.e., the one for which the network shows the best performance. The problem with this technique is that the number of combinations can be very high, requiring excessive computational cost. Random search tests only some of the possible combinations randomly, reducing computation time.

Hyperparameter tuning is crucial for using neural networks, as it determines the network's accuracy in performing its task. For this reason, the two techniques mentioned can be improved in several ways to reduce computational cost and increase efficiency.

2.6.9 Recurrent Neural Networks

Feed-forward neural networks, described in 2.6.1, have proven capable of achieving exceptional results in many practical cases. They are designed for multidimensional data with independent features, treating each observation as uncorrelated with the others. However, some datasets, such as time series and texts, present sequential dependencies and are not suitable for analysis by these networks because they do not consider "context." The need to model such data led to the introduction of a different neural network model, called *Recurrent Neural Networks* (RNNs), which aim to encode sequential and temporal relationships. The single data point x analyzed by an RNN can be represented as a time series: it consists of a sequence of observations $\{x_t\}$. Each observation is an essential part of the information, but if analyzed separately from the others, it overlooks all context-derived information. The peculiarity of recurrent neural networks is having feedback connections that allow them to account for the context.

The simplest RNN, introduced in Elman (1990), consists of a hidden layer and an output layer. The peculiarity of this type of network is the self-loop of the hidden layer h, which enables it to utilize knowledge of the state at the previous time step. An RNN can be extended through a temporal layer network resembling a standard feed-forward network, where each layer represents a different time instant. The hidden layer state is thus described by the relation:

$$h_t = \Phi(x_t, h_{t-1}; w).$$

It is important to note that the network receives a different input x_t at each time instant, specific to that time instant; the hidden layer state is influenced by its previous value and the input data, and consequently, the output also depends on time. However, the weight matrices and activation function are the same at every time instant, meaning they do not depend on time.

The output is defined as a function of the state h_t, i.e., $y_t = g(h_t)$, but thanks to the recursive nature of the hidden layer, it can be rewritten as a function of the input data alone:

$$y_t = F_t(x_1, x_2, \ldots, x_t).$$

Thus, the function $F_t(\cdot)$ varies with the value of t, although its relationship with the immediately preceding state remains the same.

More specifically, a vanilla RNN with a hidden layer and an output layer is described by the following equations, presented here in vector form:

$$h_t = \Phi(Wh_{t-1} + Ux_t + b_h), \quad t = 1, \ldots, T,$$

$$y_t = \Phi_{\text{output}}(Vh_t + b_y).$$

Suppose p is the dimension of the internal state, d_0 is the dimension of the input data, and d_1 is the output dimension; then U is a weight matrix of size (p, d_0), W is a matrix of size (p, p), V is a matrix of size (d_1, p), and b_h and b_y are bias vectors of dimensions p and d, respectively, for the two activation functions Φ and Φ_{output}.

2.6.10 Long-Short Term Memory Networks

A more effective method for addressing the gradient problem is to introduce an internal memory into the recurrent network to bring greater stability to the network states and gradient updates. To this end, Long-Short Term Memory (LSTM) networks were introduced (Hochreiter and Schmidhuber, 1997). The architecture of these networks is much more complex than that of a vanilla RNN.

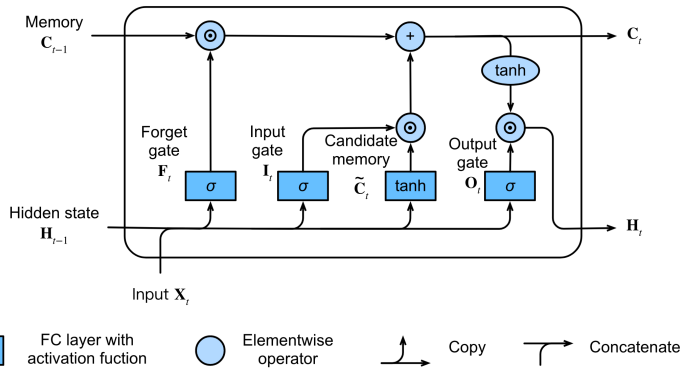

Figure 2.4: LSTM neural network architecture (Zhang et al. 2021)

In these networks, a long-term memory C_t, called the cell state, is present, which serves to retain information from previous states. This memory is updated over time by the algorithm in a delicate manner to ensure greater persistence in information storage, thus avoiding instabilities arising from the vanishing/exploding gradient problem. Three gated units F_t, I_t, and O_t have also been added, referred to as the forget gate, input gate, and output gate, respectively. The first two regulate the flow of information into the cell state by forgetting and adding new information at each time step. The third gate modifies the state H_t of the hidden layer, which then influences the output values. The equations governing these dynamics, assuming the network solves a classification problem with K different classes, are:

$$F_t = \sigma(W_f H_{t-1} + U_f X_t + b_f),$$

$$
\begin{aligned}
I_t &= \sigma(W_i H_{t-1} + U_i X_t + b_i), \\
O_t &= \sigma(W_o H_{t-1} + U_o X_t + b_o), \\
C_t &= F_t \otimes C_{t-1} + I_t \otimes \tanh(W_c H_{t-1} + U_c X_t + b_c), \quad t = 1, \ldots, T, \\
H_t &= \tanh(C_t) \otimes O_t, \quad t = 1, \ldots, T, \\
\hat{y}_t &= \text{softmax}(V H_t + b_y),
\end{aligned}
$$

where σ represents the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\otimes$ denotes element-wise multiplication of two vectors. The matrices W_j, for $j \in \{f, i, o, c\}$, have dimensions (p, p), where p is the number of nodes in the LSTM layer, and the matrices U_j, for $j \in \{f, i, o, c\}$, have dimensions (p, d), where d is the input data vector dimension. Finally, the matrix V has dimensions (K, p). Consequently, the total number of parameters the model must estimate is $4(p(d+1) + p^2) + K(p+1)$, as there are four copies of the triplet (W, U, b) plus the parameters for the classification response in K classes.

Now, the meaning of the various components of the network can be better understood. The three gates F, I, O, since the sigmoid function assumes values in $(0, 1)$, regulate whether, and how much, the information in each neuron can be forgotten by the cell state: $F_t \otimes C_{t-1}$, added to the cell state: $I_t \otimes \tanh(\cdot)$, or flow from the cell state to the hidden state: $\tanh(\cdot) \otimes O_t$. The cell state, where the recurrent nature of the network resides, stores the most important information, which is then used by the hidden layer to decide what to transmit to the output layer. Furthermore, the additive nature of the cell state updates helps avoid the vanishing-gradient problem.

The LSTM neural network, with its characteristics suited for analyzing time series, will be used throughout this work.

Chapter 3

Methodology

3.1 Overview

This chapter details the comprehensive methodology employed to construct and evaluate the pairs trading strategy. The approach is grounded in a robust integration of theoretical concepts such as cointegration and time series modeling, alongside practical implementation using both traditional statistical methods and advanced Long-Short Term Memory (LSTM) Neural Networks using the open source softwere Python 3.9.18 on a Apple M1 8-core CPU 3.2GHz, 16 GB RAM.

The pairs trading strategy capitalizes on the mean-reverting behavior observed between pairs of securities. The core premise is that the spread between the prices of two cointegrated assets will revert to its historical mean over time. This expectation forms the basis for making trading decisions—buying the underperforming asset and selling the outperforming one when deviations from the mean spread are detected.

To implement this strategy, the methodology follows several critical steps:

- **Data Collection and Preparation:** Historical price data for the constituents of the Dow Jones Industrial Average is collected from reliable financial databases. This data is then cleaned and preprocessed to ensure accuracy and consistency. Preprocessing steps include handling missing values, adjusting for corporate actions, and normalizing the data to facilitate analysis.

- **Pairs Selection Using Cointegration Analysis:** Potential pairs are identified using the Engle-Granger Test for cointegration. This test determines whether a linear combination of two non-stationary time series is stationary, indicating a long-term equilibrium relationship. The test involves performing an Ordinary Least Squares (OLS) regression of one asset's price on another and conducting the Augmented Dickey-Fuller (ADF) test on the residuals of the regression to check for stationarity. Pairs with stationary residuals are considered cointegrated and suitable for the trading strategy.

- **Modeling and Forecasting the Spread:** The spread, defined as the price difference between two cointegrated assets, is modeled to predict its

future behavior. Two primary modeling approaches are employed:

  - ARMA (Autoregressive Moving Average) model
  - Long-Short Term Memory (LSTM) Neural Networks

- **Training and Validation of Models:** The models are trained on a designated training dataset and validated on a separate test dataset. The training process involves using a loss function (MSE loss) to measure the discrepancy between predicted and actual values, employing the Adam optimizer to minimize the loss function, and implementing early stopping to prevent overfitting.

- **Backtesting the Trading Strategy:** The strategy is rigorously backtested over historical data to evaluate its performance. This involves defining entry and exit signals based on deviations from the mean spread, and calculating performance metrics such as cumulative returns, volatility, Sharpe ratio.

By integrating rigorous statistical analysis and advanced neural network techniques, this methodology aims to enhance the robustness and profitability of the pairs trading strategy within the Dow Jones Industrial Average Index. The subsequent sections provide detailed insights into each step of the methodology, from data collection to model validation and backtesting.

3.2 Data Collection and Preparation

The cornerstone of this research is the meticulous collection and preparation of historical stock price data for the Dow Jones Industrial Average (DJIA) components collected using a combination of web scraping and using the `yfinance` library, which provides an easy-to-use interface for fetching financial data from the Yahoo Finance API. The DJIA is a prominent stock market index comprising 30 significant publicly traded companies in the United States, reflecting the overall health of the US stock market. For this analysis, historical data was gathered over a 15-year period, from January 1, 2009, to January 1, 2024, which encompasses various market cycles like the market fluctuations due to the COVID-19 pandemic.

The initial step in the data collection process involved obtaining the list of current DJIA components. This was achieved through web scraping the relevant Wikipedia page, which provided the most up-to-date and accurate list of the DJIA companies. Using the BeautifulSoup library in Python, the table containing the ticker symbols was identified and parsed, ensuring that the list of companies was correctly extracted.

Once the ticker symbols were acquired, the historical adjusted closing prices for each DJIA component were downloaded using the Yahoo Finance API. This reliable source provided the necessary historical data for the specified date range, allowing for a comprehensive analysis that spans significant market events and trends.

Data preparation is a critical phase in ensuring the dataset's integrity and reliability. The raw data often contains imperfections that need to be addressed before any meaningful analysis can be conducted. An initial inspection of the

dataset revealed the presence of missing values in the Dow Inc. stock, since it started being part of the DJIA only on 2nd April 2019[1], which were handled by removing the ticker.

To gain a preliminary understanding of the data, the adjusted closing prices of each DJIA component were plotted over time. This visualization helped identify any anomalies or outliers and provided an initial sense of the price movements of individual stocks relative to the broader market. Such visual inspections are invaluable for detecting patterns and trends that might not be immediately apparent from raw data. On top of this, the adjusted closing prices ensures that the impact of dividends, stock splits, and other corporate actions are appropriately reflected in the price data, providing a true representation of the stock's value over time. This adjustment allows for more reliable and meaningful analysis, leading to better-informed investment decisions.

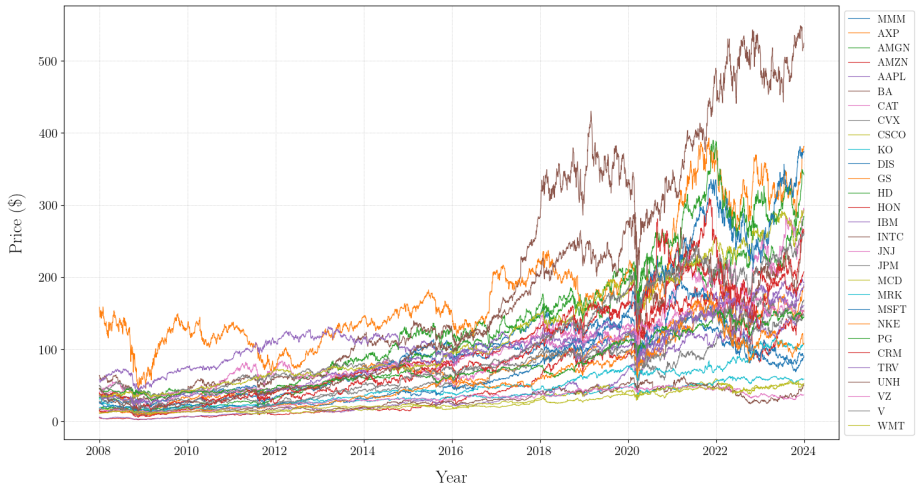

Figure 3.1: Adjusted Closing Prices

Additionally, to benchmark the performance of individual stocks against the overall market, the adjusted closing prices of the DJIA index itself were downloaded and analyzed. The cumulative returns of the DJIA index were plotted in Figure 3.2, offering a clear visual representation of the market's performance over the selected period. This comparison is essential to understand how individual stocks perform in the context of the overall market trends.

Through these meticulous steps of data collection and preparation, a clean and comprehensive dataset was established. This dataset forms the robust foundation for the subsequent analysis and modeling in this study. By ensuring high-quality data, the findings and conclusions derived from this research are grounded in accurate and reliable information, thereby enhancing the credibility and validity of the study.

[1]This change occurred following the split of DowDuPont into three separate entities: Dow Inc., DuPont, and Corteva. Dow Inc. was added to the DJIA as part of this restructuring.

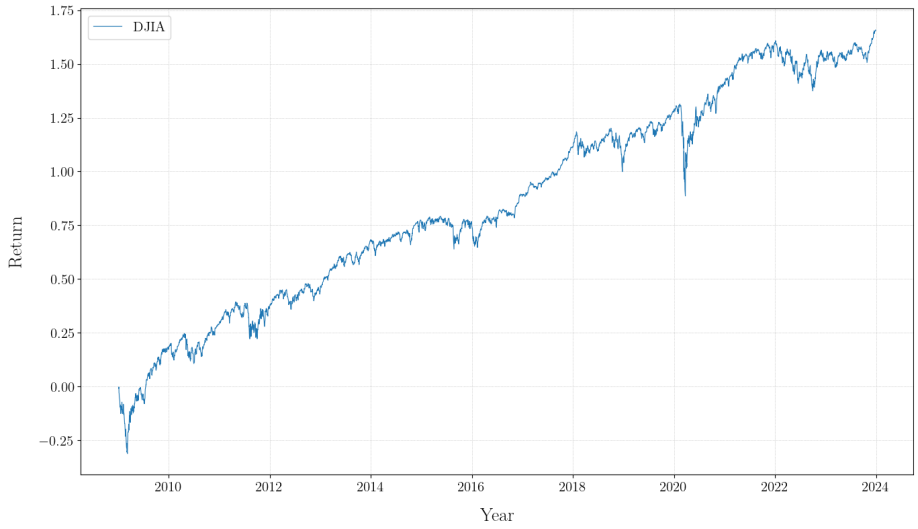

Figure 3.2: DJIA Cumulative Returns

3.3 Pairs Selection Using Cointegration Analysis

The selection of pairs for trading is a critical step in pairs trading strategy, where the goal is to identify pairs of stocks whose prices move together over time, maintaining a stable, long-term relationship. This work employs cointegration analysis to identify such pairs among the Dow Jones Industrial Average (DJIA) components.

To perform the cointegration analysis, the adjusted closing prices of the DJIA components, collected from January 1, 2009, to January 1, 2024, were used. The analysis was conducted using the Engle-Granger two-step method, implemented in Python through the function `coint` from the `statsmodels` package.

Initially, all possible pairs of stocks were generated using the combinations of the DJIA tickers. This resulted in a comprehensive list of stock pairs to be tested for cointegration.

One consideration in such studies is *survivorship bias*, which occurs when only the stocks that have remained in the index until the end of the study period are analyzed, while those that were dropped are ignored. This bias can lead to overly optimistic conclusions because it excludes companies that failed or were removed from the index, potentially skewing the results.

However, for this study, survivorship bias is not a critical concern as the primary objective is to compare the predictive performance and strategy performance of different models in identifying and trading on stable relationships between pairs of stocks within the DJIA. By focusing on the relative behavior of stock pairs rather than the absolute behavior of the DJIA Index, the study leverages the stable and significant components of it to ensure consistent and robust analysis.

The cointegration analysis identifies pairs that exhibit significant long-term

relationships, providing a reliable basis for evaluating the effectiveness of the pairs trading strategy.

The cointegration test was performed for each pair using the Engle-Granger method. Specifically, the Augmented Dickey-Fuller (ADF) test was applied to the residuals obtained from the linear regression of the two stock prices. If the p-value of the ADF test was below 0.05, the pair was considered cointegrated, indicating a statistically significant long-term equilibrium relationship between the two stocks.

This methodology enabled the identification of pairs of stocks within the DJIA that maintained a stable, long-term relationship during the training period. These cointegrated pairs were then used to test and validate the pairs trading strategy in the subsequent analysis, leveraging their stable relationships to generate trading signals and potential profit opportunities.

3.4 Spread Modeling

The spread modeling process is a crucial component in pairs trading strategy, aimed at quantifying the price relationship between two cointegrated stocks. By understanding and modeling the spread, traders can make informed decisions about entering and exiting positions based on the divergence and convergence of stock prices. In this study, we implemented spread modeling through the calculation of a spread series and the application of forecasting techniques.

The initial step in spread modeling involves calculating the spread between two assets, A and B. This is done by performing a linear regression of the price series of asset B on asset A, which provides a hedge ratio, β. The spread is then defined as the difference between the price of asset B and β times the price of asset A. Mathematically, the spread S_t at time t is given by:

$$S_t = P_{B,t} - \beta P_{A,t}$$

where $P_{B,t}$ and $P_{A,t}$ are the prices of asset B and asset A at time t, respectively, and β is the slope coefficient obtained from the regression.

To ensure the spread series is stationary, it is normalized by subtracting the mean and dividing by the standard deviation:

$$\text{Normalized Spread}_t = \frac{S_t - \mu_S}{\sigma_S}$$

where μ_S and σ_S are the mean and standard deviation of the spread series, respectively. This normalization process ensures that the spread has a zero mean and unit variance, facilitating easier identification of deviations from the mean.

Once the spread series is calculated and normalized, trigger levels for trading signals are set based on the percentage changes in the spread. The long and short triggers are defined as the 90th and 10th percentiles of the percentage change in the spread, respectively. These trigger levels indicate significant deviations from the mean spread, prompting potential trading opportunities (Figure 3.3).

To forecast future spread values, two modeling approaches were employed: the AutoRegressive Moving Average (ARMA) model and the Long Short-Term Memory (LSTM) neural network model.

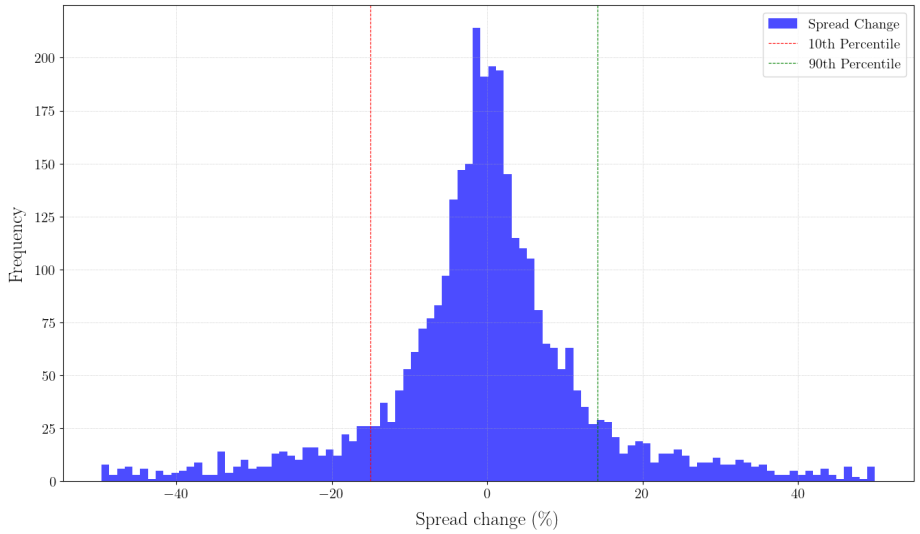

Figure 3.3: Spread percentage change distribution

3.4.1 ARMA Model

The ARMA model is a popular time series forecasting method that captures linear dependencies in the data. The model parameters are chosen based on historical spread data, and the fitted model is used to predict future spread values. The ARMA(1,1) model used in this study is represented as:

$$S_t = \phi_1 S_{t-1} + \theta_1 \epsilon_{t-1} + \epsilon_t$$

where S_t is the spread at time t, ϕ_1 is the autoregressive parameter, θ_1 is the moving average parameter, and ϵ_t is the white noise error term.

The choice of using the ARMA(1,1) model was influenced by the need for a balance between model simplicity and computational efficiency. Therefore, the ARMA(1,1) model was selected as a practical compromise, providing a straightforward and efficient approach while still capturing the essential linear dependencies in the spread data.

To implement it in the Python enviorment, the `ARIMA` class from the `statsmodels` library was used with `order=(1,0,1)`, indicating one autoregressive term and one moving average term.

3.4.2 LSTM Neural Network

The LSTM model, a type of recurrent neural network (RNN), is used for its ability to capture long-term dependencies and non-linear patterns in the time series data. The LSTM network is trained on historical spread data, and its performance is evaluated on a separate test set. The LSTM model's architecture includes input, forget, and output gates, which help in retaining relevant information and discarding irrelevant information over time.

In this work, the neural network structure was chosen with careful consideration of the trade-off between model complexity and computational efficiency. The LSTM model consists of a single LSTM layer containing 4 units, followed by a dense layer with a single neuron to produce the forecasted spread value. The decision to use a single LSTM layer is influenced by several factors:

- **Simplicity and Interpretability:** A single LSTM layer helps maintain simplicity and interpretability of the model. While deeper networks can capture more complex patterns, they also introduce additional complexity and the risk of overfitting, especially with limited data.
- **Computational Efficiency:** Training deep neural networks requires substantial computational resources. Given the constraints of the available hardware (Apple M1 8-core CPU, 16 GB RAM), a single LSTM layer ensures that the model can be trained efficiently within a reasonable time frame without exhausting computational resources.
- **Adequate Performance:** For many time series forecasting tasks, a single LSTM layer can effectively capture essential patterns and dependencies. The inclusion of 4 units within this layer allows the model to learn sufficient temporal relationships in the spread data without unnecessary complexity.

The process begins with the normalization of the training and test data using the `MinMaxScaler` from the `sklearn.preprocessing` package. A sliding window approach is used to create datasets for the LSTM model, where each input sequence consists of the previous 5 time steps of the spread, and the target is the spread at the next time step. This window size is chosen to provide the model with enough historical context to make accurate predictions.

The LSTM model is constructed using the `Keras` library, which is part of `TensorFlow`. The network architecture includes a single LSTM layer with 4 units, followed by a dense layer with a single neuron. The model is compiled with the mean squared error loss function and the Adam optimizer, which are also part of the `Keras` library, known for their efficiency in training deep learning models.

The model is trained for 100 epochs with a batch size of 1, which allows the network to update its weights frequently, thereby improving its ability to capture fine-grained patterns in the data. Frequent updates are particularly beneficial for time series data, where patterns can be subtle and nuanced.

For each time step in the test data, the LSTM model generates a forecast, which is then compared to the observed spread value. The results, including the forecasted and observed spreads, are stored along with relevant metadata.

The forecasted spread values from the LSTM model are used to determine trading positions. If the predicted spread exceeds the long trigger, a long position is initiated, anticipating a convergence back to the mean. Conversely, if the predicted spread falls below the short trigger, a short position is taken. Positions are adjusted based on the changes in the predicted spread and the triggers.

This comprehensive spread modeling approach, incorporating both linear and non-linear forecasting techniques, provides a robust framework for pairs

trading. By accurately modeling the spread and identifying significant deviations, traders can capitalize on mean-reverting behavior, thereby enhancing the profitability of the pairs trading strategy.

3.5 Training and Validation

Considering multiple partitions of the dataset is particularly enticing, as analyzing the trading results under different conditions enhances confidence in the statistical significance of the results. Unlike classical Machine Learning problems where k-fold cross-validation is typically employed, time series data pose unique challenges. Traditional cross-validation methods are not suitable for time series due to the risk of look-ahead bias, which can occur when future data is used to train the model.

To address this issue, a sequential moving training and test window approach was utilized. This method ensures that each model is trained and tested with data available up to the specific historical date, thus avoiding look-ahead bias. While forward chaining could also be applied, it results in variable training and test period sizes, potentially affecting the consistency of the results. The fixed-size window approach used here maintains consistency in the data sizes across all partitions.

For this reason, as illustrated in Figure 3.4, historical data was divided into rolling windows, each comprising a four-year training period followed by a one-year testing period. This method ensured that the models were validated across different market conditions, enhancing also their robustness.

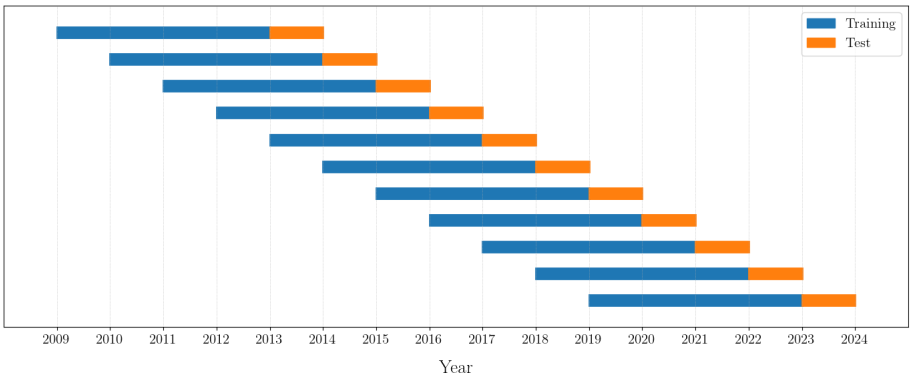

Figure 3.4: Data partitions periods

For each partition, the cointegrated pairs are found through the methodology explained in Section 3.3. The spread between each pair, already calculated and normalized, served as the basis for setting long and short trading triggers. These triggers were used to generate trading signals during the testing period.

Both the forecasting models were trained on the spread series from the training period and tested on the subsequent year.

During the testing phase, the forecasted spread values from the models were used to determine trading positions. These positions were dynamically adjusted

based on the predicted changes in the spread relative to the established triggers.

This rolling window approach provided several benefits. By continuously updating the training and testing periods, the models could adapt to changes in market dynamics, ensuring that they remained relevant and effective over time. This method also allowed for a more comprehensive evaluation of the strategy's performance, as it was tested across multiple market cycles and conditions.

Moreover, the use of a rolling window approach mitigated the risk of overfitting, a common problem in financial modeling. By constantly training and validating the models on different segments of the data, the likelihood of the models being too closely tailored to a specific period was reduced. This increased the generalizability and reliability of the models when applied to unseen data.

The results from each partition were aggregated to assess the overall performance and robustness of the strategy. This aggregate analysis provided insights into the strategy's consistency and stability, highlighting its potential for real-world application. The dynamic nature of this approach ensured that the models could effectively respond to evolving market conditions, making the pairs trading strategy both adaptable and resilient.

3.6 Backtesting the Trading Strategy

The backtesting process involves applying the pairs trading strategy to historical data to evaluate its performance and profitability. In this study, both ARMA and LSTM models were used within a comprehensive framework for this purpose where the historical data, covering the period from January 1, 2009, to January 1, 2024, was used for training and validating the models.

The backtesting process began with the creation of instances for both ARMA and LSTM models, and the data was dynamically partitioned into training and testing sets through a rolling window approach. For each partition, the spread values between pairs of stocks were forecasted using the ARMA model to capture linear dependencies and the LSTM model to identify non-linear patterns. These models were trained on the spread series from the training period and then used to forecast the spread during the subsequent testing period.

Trading positions were determined based on the forecasted spread values. The strategy monitored the percentage change between the spread at the current time and the predicted spread at the next time-step. When the absolute value of the predicted change exceeded a predefined threshold, a position was initiated, anticipating an abrupt movement in the spread from which the investor could benefit. Specifically, the true and predicted values for the spread at time t were defined as S_t and S_t^*, respectively. The predicted change was calculated as:

$$\Delta_{t+1} = \frac{S_{t+1}^* - S_t}{S_t} \times 100$$

The conditions for setting positions were:

$$\text{Market conditions} : \begin{cases} \text{if } \Delta_{t+1} \geq \alpha_L, & \text{open long position} \\ \text{if } \Delta_{t+1} \leq \alpha_S, & \text{open short position} \\ \text{otherwise}, & \text{remain outside market} \end{cases}$$

where α_L and α_S were the top and bottom quintiles, respectively. Positions were maintained as long as the predicted spread direction persisted and were closed when the direction shifted.

In contrast to some research that aims for a dollar-neutral approach, which involve investing \$1 in both long and short positions, this study follows the cointegration ratio β between the two securities as the spread is defined as $S_t = Y_t - \beta X_t$ affecting the way positions are set. Consequently, the amount invested in X is β times the amount invested in Y.

Additionally, transaction costs were incorporated into the strategy to assure that the results presented in this work could reflect as much as possible real-word application. A transaction cost of 0.1% was applied when entering a new position, and an additional 1% cost was applied for short selling.

Figure 3.5: Cumulative returns

The cumulative returns of the ARMA and LSTM strategies were compared to the cumulative returns of the DJIA index (Figure 3.5), highlighting the relative performance of the pairs trading strategy against the broader market. This approach demonstrated the robustness and adaptability of the pairs trading strategy, effectively responding to different market conditions while considering transaction costs for a realistic evaluation of its practical applicability.

Chapter 4

Results

The strategy's return, volatility, and Sharpe ratio were calculated to assess its risk-adjusted performance for each year of testing. These metrics are crucial for understanding the profitability, risk, and efficiency of the trading strategy.

The annualized volatility σ_a indicates the strategy's risk by measuring the variability of returns. Volatility is a key aspect of financial performance as it reflects the degree of uncertainty or risk associated with the strategy's returns. High volatility indicates a high level of risk, as returns can vary significantly from the mean. The annualized volatility considering 252 trading days per year, was calculated as:

$$\sigma_a = \sigma_d \times \sqrt{252} \tag{4.1}$$

where σ_d is the standard deviation of the daily returns. This formula scales the daily volatility to an annual level, providing a comprehensive view of the strategy's risk over a year.

The Sharpe ratio (Sharpe, 1966) provides a measure of risk-adjusted return, which is essential for evaluating the efficiency of the strategy in generating returns relative to the risk taken. It is a widely used metric in finance to compare the performance of investment strategies. The Sharpe ratio was determined using the formula:

$$SR = \frac{R_a - r_f}{\sigma_a}, \tag{4.2}$$

where R_a is the yearly return, r_f is the risk-free rate that was approximated to 0 since for comparing purposes doesn't impact the results.

The Sharpe ratio considers both the return and the risk of the strategy, offering a single value that balances these two critical aspects. A higher Sharpe ratio indicates a more favorable risk-adjusted return, meaning the strategy generates higher returns for each unit of risk taken.

The backtesting results demonstrated that both the ARMA and LSTM models could effectively capture and exploit the mean-reverting behavior of cointegrated pairs, resulting in profitable trading strategies. The ARMA model, which captures linear dependencies in the spread data, provided a solid baseline for performance. However, the LSTM model, with its ability to capture non-linear patterns, showed superior performance in terms of risk-adjusted returns compared to the ARMA model. The LSTM model's advantage lies in its capacity

to learn complex relationships and dynamics in the data, enabling it to predict more accurately and adapt to market changes more effectively.

Overall, the application of these metrics in backtesting the trading strategy provides a comprehensive evaluation of its performance, highlighting its strengths and areas for improvement. The LSTM model's superior Sharpe ratio underscores its effectiveness in delivering higher returns per unit of risk, making it a compelling choice for implementing pairs trading strategies in dynamic financial markets.

Year	Index			ARMA			LSTM		
	Return	Volatility	SR	Return	Volatility	SR	Return	Volatility	SR
2013	0.2359	0.0993	2.2755	0.0223	0.2068	0.0595	0.2083	0.3791	0.5232
2014	0.0752	0.1089	0.5985	0.2270	0.2373	0.9141	-0.1553	0.3387	-0.4880
2015	-0.0223	0.1546	-0.2092	0.5866	0.3941	1.4630	1.2116	0.4684	2.5653
2016	0.1342	0.1262	0.9836	0.2031	0.3684	0.5243	0.1133	0.7359	0.1404
2017	0.2508	0.0662	3.6359	-0.0449	0.1005	-0.5459	0.1197	0.2985	0.3674
2018	-0.0563	0.1796	-0.3692	-0.0071	0.4479	-0.0381	0.5308	0.5803	0.8975
2019	0.2234	0.1246	1.7129	0.2968	0.1628	1.7618	0.3430	0.7067	0.4713
2020	0.0725	0.3704	0.1686	-0.0924	0.4735	-0.2163	-0.5440	0.8790	-0.6303
2021	0.1873	0.1244	1.4251	2.0089	0.7813	2.5585	-0.1701	0.5746	-0.3134
2022	-0.0878	0.1984	-0.4930	-0.3946	0.4007	-1.0097	1.4629	0.7502	1.9367
2023	0.1370	0.1140	1.1142	-0.1497	0.1391	-1.1478	-0.0805	0.3247	-0.2788

Table 4.1: Yearly performance Metrics of Index, ARMA, and LSTM Models

4.1 Analysis of Model Performance During Market Turmoil

The performance metrics for the Index, ARMA, and LSTM models from 2013 to 2023 reveal significant fluctuations, particularly during periods of market turmoil.

From 2013 to 2014, both the ARMA and LSTM models demonstrated stable performance, effectively capturing market dynamics and generating profitable trading strategies. For example, in 2013, the ARMA model achieved an annual return of 2.23% with a Sharpe ratio of 0.0595, while the LSTM model reported an annual return of 20.83% with a Sharpe ratio of 0.5232. However, starting in mid-2014, a series of global events significantly impacted market conditions, challenging the models' effectiveness.

The oil price crash beginning in mid-2014 caused substantial volatility in global markets. The sharp decline in oil prices led to increased uncertainty and instability, particularly affecting sectors tied to energy and commodities. This sudden market shift disrupted the historical patterns the models relied upon, leading to difficulties in accurately predicting returns. Consequently, in 2015, the ARMA model recorded a negative annual return of -2.23% with a Sharpe ratio of -0.2092, and the LSTM model, despite achieving high returns, exhibited increased volatility and a high Sharpe ratio, indicating a higher risk profile.

Simultaneously, the lingering effects of the European Sovereign Debt Crisis continued to impact markets. Uncertainty about the financial stability of European countries introduced additional volatility and risk, further complicating the models' predictive capabilities. In 2015, this ongoing crisis contributed to the overall challenging market environment.

China's economic slowdown during this period added another layer of complexity. As China's growth decelerated, global trade and market sentiment were adversely affected. The slowdown led to increased market volatility, which the ARMA and LSTM models struggled to capture accurately, given their reliance on historical data that did not account for these new dynamics.

The Federal Reserve's rate hike in 2015 introduced further market adjustments. Anticipation of the rate hike, followed by its implementation, caused shifts in investor behavior and market dynamics. Changes in interest rates affect discount rates and valuations, leading to an environment of uncertainty and adjustment. Models trained on data from previous lower-rate environments found it challenging to navigate these changes effectively.

The COVID-19 pandemic in 2020 marked another significant period of market disruption. The pandemic caused unprecedented volatility, with rapid and severe market fluctuations. The ARMA model experienced a negative annual return of -9.24% and a Sharpe ratio of -0.2163, indicating its failure to adapt to the highly volatile environment. Similarly, the LSTM model's performance plummeted, with an annual return of -54.40% and a Sharpe ratio of -0.6303. These sharp declines can be attributed to the models' inability to cope with the unprecedented volatility and uncertainty brought about by the pandemic.

In 2021, there was a brief recovery. The ARMA model achieved a notable annual return of 200.89% and a Sharpe ratio of 2.5585, suggesting a temporary adaptation to the post-pandemic market conditions. However, the LSTM model continued to struggle, with a negative return of -17.01% and ongoing difficulty in adjusting to the new market dynamics.

The subsequent year, 2022, saw another significant decline in performance, particularly for the ARMA model, which recorded an annual return of -39.46% and a Sharpe ratio of -1.0097. Conversely, the LSTM model exhibited a remarkable annual return of 146.29% but with high volatility, resulting in a Sharpe ratio of 1.9367. This indicates that while the LSTM model managed to achieve high returns, it did so with considerable risk, reflecting its struggle to stabilize in the volatile market conditions.

By 2023, the instability persisted, with the ARMA model showing a negative return of -14.97% and the LSTM model exhibiting slight negative returns of -8.05%. This continued underperformance highlights the ongoing challenges faced by both models in adapting to the post-COVID-19 market environment.

In summary, the general reasons for the observed drops in performance of the ARMA and LSTM models include increased market volatility, structural breaks, and significant shifts in market dynamics due to external macroeconomic events. These factors disrupted the historical patterns that the models depended on, highlighting the need for more robust and adaptive modeling approaches. This analysis underscores the importance of incorporating resilience and flexibility into pairs trading strategies, particularly when applied to the components of the DJIA Index, to better withstand future market disruptions and maintain performance stability.

4.2 Risk Management Analysis

Strategy	Return	Volatility	Sharpe Ratio
Index	10.45%	0.1515	0.9857
ARMA	24.15%	0.3375	0.3930
LSTM	27.53%	0.5487	0.4719

Table 4.2: Average performance Metrics of Index, ARMA, and LSTM Models

From a risk management perspective, the annualized volatility and Sharpe ratio provide crucial insights into the performance and risk characteristics of the trading models under consideration. Volatility measures the degree of variation in returns, with higher values indicating greater uncertainty and potential risk. In this context, considering the average of each metric shown in Table 4.2, the Index model exhibits the lowest annualized volatility at 0.1515, suggesting that it offers the most stable returns among the three models. On the other hand, the LSTM model presents the highest volatility at 0.5487, signifying more pronounced fluctuations in its returns.

While higher volatility typically implies higher risk, it is essential to evaluate the risk-adjusted performance using the Sharpe ratio. The Sharpe ratio assesses the amount of excess return generated per unit of risk, with a higher ratio indicating a more favorable risk-adjusted return. This metric is instrumental in determining how effectively a strategy balances return and risk.

In this analysis, the Index model, which represents a passive investment strategy, demonstrates a Sharpe ratio of 0.9857, indicating that it delivers a respectable return for its level of risk. As passive investment strategies aim to replicate the performance of a benchmark index, they do not require frequent trading or extensive analysis, leading to lower risk and lower costs. The consistent, stable returns of the Index model reflect this approach, making it an attractive option for risk-averse investors seeking reliable, long-term growth.

Conversely, active investment strategies, such as those employing the ARMA and LSTM models, aim to outperform the market through various techniques, including forecasting and trading on predicted relationships between assets. Indeed, the ARMA model, achieving a higher annualized return of 24.15%, has a lower Sharpe ratio of 0.3930. This suggests that while the ARMA model can generate substantial returns, it does so with a proportionally higher level of risk, making it less efficient in terms of risk-adjusted performance.

In contrast, the LSTM model, with the highest annualized return of 27.53%, also exhibits the highest volatility. However, its Sharpe ratio of 0.4719 is significantly better than that of the ARMA model, indicating the LSTM model delivers higher returns more efficiently relative to its risk. This reflects the model's ability to capture non-linear relationships and adapt to market dynamics effectively.

The superior Sharpe ratio of the LSTM model highlights its effectiveness in delivering higher returns per unit of risk. This advantage comes from the model's capacity to learn and leverage complex patterns within the data, enabling accurate predictions and swift adaptation to changing conditions. Therefore, the LSTM model is ideal for pairs trading strategies in dynamic financial markets.

Chapter 5

Discussion

The primary objective of this study was to explore the efficacy of pairs trading strategies within the Dow Jones Industrial Average (DJIA) by comparing traditional statistical methods, such as the ARMA model, with advanced neural network techniques like Long-Short Term Memory (LSTM) neural networks. The results of the research demonstrate significant insights into the performance and risk characteristics of these models across various market conditions, from 2009 to 2024.

5.1 Model Performance and Market Conditions

The study found that both ARMA and LSTM models were effective in capturing mean-reverting behaviors of cointegrated stock pairs during stable market periods, generating profitable trading signals. However, during times of heightened market volatility and turmoil, such as the oil price crash in 2014-2015 and the COVID-19 pandemic in 2020, the models faced challenges. The ARMA model, which relies on linear dependencies, struggled to adapt to sudden market shifts, leading to poorer returns and lower Sharpe ratios. Conversely, the LSTM model, with its capacity to learn complex non-linear patterns, showed a degree of resilience and adaptability, although it too experienced difficulties during extreme market disruptions.

The LSTM model's superior risk-adjusted returns, indicated by higher Sharpe ratios, highlight its ability to generate higher returns per unit of risk compared to the ARMA model. This is particularly valuable in dynamic market environments where capturing intricate patterns can significantly enhance trading strategies.

5.2 Limitations, Delimitation and Further Research

A critical limitation of this study is the computational complexity and resource intensity associated with training LSTM models and performing optimization techniques for both ARMA models and LSTM Neural Networks. Neural networks, particularly those as sophisticated as LSTMs, require substantial com-

putational power for effective training and validation (Bengio et al., 1994). This constraint limited the scope of our analysis and necessitated significant time and resource investment. As a result, the study's findings are primarily confined to the DJIA components, which may not fully represent broader market dynamics. Future research should expand the dataset to include a wider range of stocks across different indices to enhance the generalizability of the results (Han et al., 2011).

Moreover, the high computational demands of LSTM models highlight the need for more efficient neural network architectures or hybrid models that combine the strengths of both statistical and machine learning approaches. This could help mitigate the resource constraints while maintaining or even enhancing model performance. Addressing the computational challenges of LSTM models requires a multifaceted approach. One promising strategy involves developing hybrid models that leverage the complementary strengths of ARMA and LSTM techniques (Zhang, 2003). By delegating linear aspects of the data to the ARMA model, the LSTM can focus on capturing more complex, non-linear relationships, thereby reducing the overall computational load. This method not only optimizes resource utilization but also enhances predictive accuracy by combining the best of both methodologies.

Optimizing the LSTM architecture itself is another crucial step. Fine-tuning the number of layers and neurons, and incorporating dropout layers to prevent overfitting, can significantly improve efficiency (Hinton et al., 2012). For instance, reducing the complexity of the LSTM by limiting the number of layers or neurons can decrease training times without a substantial loss in model performance. Additionally, implementing dimensionality reduction techniques, such as Principal Component Analysis (PCA), prior to training can help manage high-dimensional data more effectively, further streamlining the computational process (Goodfellow et al., 2016).

Hyperparameter optimization is essential for both LSTM and ARMA models to achieve optimal performance. For LSTM models, techniques such as grid search or Bayesian optimization can be employed to fine-tune hyperparameters like the number of layers, number of neurons in each layer, learning rate, batch size, and dropout rate (Bergstra and Bengio, 2012; Snoek et al., 2012). These methods involve systematically exploring a range of hyperparameter values to identify the combination that minimizes forecast error and maximizes model efficiency. Similarly, for ARMA models, hyperparameter optimization focuses on selecting the appropriate order of the autoregressive and moving average components, ensuring the model is well-calibrated to the data. This process involves evaluating different combinations of AR and MA orders to identify the best fit for the time series data. Hyperparameter optimization for both LSTM and ARMA models can also involve adjusting the time steps used in the sliding window approach and the thresholds for entering and exiting trades. Fine-tuning these hyperparameters can enhance the model's ability to generate accurate and timely trading signals, ultimately improving the overall performance of the pairs trading strategy.

Utilizing advanced computational frameworks and cloud-based platforms offers another practical solution. Distributed computing frameworks like Apache Spark or Dask enable parallel processing of large datasets, thus accelerating the training process (Zaharia et al., 2012). Cloud-based machine learning platforms such as Google Cloud AI, Amazon SageMaker, or Microsoft Azure

ML provide scalable resources that can handle the extensive computational demands of LSTM models (Kraska et al., 2013). These platforms offer high-performance computing capabilities, which can be particularly beneficial for extensive datasets and complex model architectures.

Efficient data handling techniques, including batch processing and data augmentation, also play a vital role in optimizing model training. Processing data in manageable batches rather than in its entirety can enhance computational efficiency and allow for the handling of larger datasets. Moreover, data augmentation techniques can increase the diversity of the training dataset without necessitating additional data collection, thereby improving model robustness.

In addition to addressing the challenges of LSTM models, it is crucial to consider enhancements for the ARMA model, particularly through the Box-Jenkins methodology. The Box-Jenkins methodology provides a systematic approach to identifying, estimating, and diagnosing ARMA models (Box and Jenkins, 1970). By carefully examining the autocorrelation function (ACF) and partial autocorrelation function (PACF), the appropriate order of the AR and MA components can be identified, improving the model's accuracy.

Incorporating seasonality into the ARMA model is also essential for capturing patterns that occur at regular intervals. By extending the ARMA framework to a Seasonal ARIMA (SARIMA) model, we can account for seasonal effects, thereby improving forecast accuracy for time series data exhibiting seasonal trends (Hyndman and Athanasopoulos, 2018).

5.3 Conclusion

This study underscores the potential of advanced neural networks like LSTM in enhancing pairs trading strategies, particularly in volatile and dynamic market environments. The empirical evidence highlights the superiority of LSTM models in delivering higher risk-adjusted returns compared to traditional ARMA models, especially during periods of market instability. The LSTM's ability to capture complex, non-linear patterns in financial time series provides a significant edge in adapting to rapid market shifts and anomalies.

However, the research also brings to light the substantial computational challenges associated with training and deploying LSTM models. The need for significant computational resources and the complexity of hyperparameter tuning present barriers to the widespread adoption of these models in real-world trading applications. Addressing these challenges through hybrid models, more efficient neural network architectures, and advanced optimization techniques is essential for practical implementation.

In conclusion, while LSTM models present promising advancements for pairs trading strategies, their practical application requires overcoming significant computational and resource challenges. By combining traditional statistical methods with innovative neural network approaches and continuously adapting to market changes, future research can develop more resilient, efficient, and profitable trading strategies. This integrated approach will not only optimize resource utilization but also enhance the predictive capabilities and adaptability of trading models in diverse market conditions.

Bibliography

[1] Ashofteh, A., Bravo, J. M., & Ayuso, M. (2022). A New Ensemble Learning Strategy for Panel Time-Series Forecasting with Applications to Tracking Respiratory Disease Excess Mortality during the COVID-19 Pandemic. *Applied Soft Computing, Volume 128*, Article 109422, October 2022.

[2] Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer.

[3] Avramov, D., Chordia, T., Jostova, G., & Philipov, A. (2006). Momentum and Credit Rating. *Journal of Finance, 62*(5), 2503-2520.

[4] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5*(2), 157-166.

[5] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281-305.

[6] Bossaerts, P., & Green, R. (1989). A General Equilibrium Model of Changing Risk Premia: Theory and Evidence. *Review of Financial Studies, 2*, 467–493.

[7] Box, G. E. P., & Jenkins, G. M. (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day.

[8] Bravo, J. M. (2022). Pricing Participating Longevity-Linked Life Annuities: A Bayesian Model Ensemble Approach. *European Actuarial Journal, 12*, 125–159.

[9] Bravo, J. M., Ayuso, M., Holzmann, R., & Palmer, E. (2021). Addressing the Life Expectancy Gap in Pension Policy. *Insurance: Mathematics and Economics, 99*, 200–221.

[10] Brockwell, P.J., & Davis, R.A. (2016). *Introduction to Time Series and Forecasting*. Switzerland: Springer.

[11] Dickey, D. A., & Fuller, W. A. (1979). Distribution of the Estimators for Autoregressive Time Series with a Unit Root. *Journal of the American Statistical Association, 74*(366), 427-431.

[12] Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*(2), 179-211.

[13] Engle, R. F., & Granger, C. W. J. (1987). Co-integration and Error Correction: Representation, Estimation, and Testing. *Econometrica, 55*(2), 251-276.

[14] Fama, E. F. (1965). The Behavior of Stock Market Prices. *Journal of Business, 38*(1), 34-105.

[15] Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance, 25*(2), 383-417.

[16] Flori, A., & Regoli, D. (2021). Revealing Pairs-trading opportunities with long short-term memory networks. *European Journal of Operational Research, 295(2)*, 772–791.

[17] Gatev, E., Goetzmann, W. N., & Rouwenhorst, K. G. (2006). Pairs trading: Performance of a relative-value arbitrage rule. *The Review of Financial Studies, 19*(3), 797–827.

[18] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[19] Hamilton, J. D. (1994). *Time Series Analysis*. Princeton University Press.

[20] Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Elsevier.

[21] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[22] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

[23] Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

[24] Jagannathan, R., & Viswanathan, S. (1988). Linear Factor Pricing, Term Structure of Interest Rates and the Small Firm Anomaly. Working Paper 57, Northwestern University.

[25] Jegadeesh, N. (1990). Evidence of Predictable Behavior of Security Returns. *The Journal of Finance, 45*(3), 881-898.

[26] 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.

[27] Jegadeesh, N., & Titman, S. (1995). Overreaction, Delayed Reaction, and Contrarian Profits. *Review of Financial Studies, 8*(4), 973-993.

[28] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

[29] Kraska, T., Talwalkar, A., Duchi, J., Griffith, R., Franklin, M. J., & Jordan, M. I. (2013). MLbase: A distributed machine-learning system. In CIDR (Vol. 1, pp. 2-1).

[30] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems, 25.

[31] Sarmento, M. S., & Horta, N. (2020). A Machine Learning based Pairs Trading Investment Strategy. Springer.

[32] Sharpe, W. F. (1966). Mutual Fund Performance. *Journal of Business, 39*(1), 119-138.

[33] Vidyamurthy, G. (2004). *Pairs Trading: Quantitative Methods and Analysis*. John Wiley and Sons.

[34] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12) (pp. 15-28).

[35] Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing, 50*, 159-175.

[36] Zhang, A., Lipton, Z., Li, M., & Smola, A. (2021). Dive into Deep Learning. *arXiv:2106.11342*.

[37] Zhang, M., Tang, X., Zhao, S., Wang, W., & Zhao, Y. (2022). Statistical Arbitrage with Momentum Using Machine Learning. *Procedia Computer Science, 202*, 194–202.

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa