Acknowledgments

First of all, I would like to take this opportunity to sincerely thank my thesis supervisor, Professor Daniele Tonini, for his guidance throughout the realization of this work, his availability and his precious advice. I began working on this thesis during my exchange semester at Georgia Tech, Atlanta, which coincided with one of the most challenging yet transformative periods in my life. I feel blessed to have met the greatest people along this journey, especially Cristiana, my roommate, Pierluigi and Francesco, thanks for making this experience truly unforgettable.

I would like to take a moment to express my heartfelt gratitude to my relatives, particularly my mom, dad, and grandparents, for their support and pride in me from the very beginning, regardless of any circumstances.

To Tommaso, my safe place and my best friend, thanks for reminding me every day what I am fighting for.

Contents

List of Figures

List of Tables

Introduction

Over the last few decades, Time Series Forecasting has gained significant attention across various disciplines, representing a dynamic and continually evolving research area. Particularly in the field of finance, stock prediction has held a prominent position for quite some time. Despite the wide array of mathematical tools available, making accurate predictions remains a challenging task, mainly because of the inherent nature of the stock market, whose movements are noisy, impacted by macroeconomic trends and more often than not, irrational. Volatility itself represents one of the biggest issues: while it can create profitable scenarios, it can also increase the opportunity of incurring in a loss. In financial economics, the Efficient-Market Hypothesis (EMH), first formalized by Eugene Fama in 1970 [1], stands as a fundamental proposition. According to the EMH, stock markets are deemed efficient, signifying their ability to promptly and accurately reflect all available information. Consequently, the hypothesis suggests that it is impossible to consistently outperform the market without accepting increased investment risk. In other words, there is no "money machine" that generates wealth for nothing: if the price of an asset is right, there is no arbitrage opportunities.

EMH implies that if new information arrives randomly in an efficient market, changes in prices caused by such information are random as well. Thus, the price is said to follow a *random walk* [2] [1]:

$$p_{t+1} = p_t + \epsilon_{t+1}$$

where $\epsilon_{t+1} \sim WN(0, \sigma^2)$. The shock is determined by the *white noise*, which is

[1] The term was first coined by a French broker, Jules Regnault, who published a book in 1863, and then by a French mathematician Louis Bachelier, in his dissertation titled "The Theory of Speculation" (1900).

unpredictable, namely the best forecast of tomorrow's price is today's price. Therefore, any change in market price will be accounted for by new information that arrives in a time interval. Even if there is some evidence in favour of EMH, this latter has faced considerable debate and challenges over time, no longer holding the status of indisputable truth. Influential figures like Warren Buffet have also expressed their hesitation towards the hypothesis. Furthermore, the empirical study conducted by Dremen and Berrys [3] presented evidence rejecting at least the strong market efficiency variant of the EMH, claiming that security prices reflect instantly all available information, public or private. Their research revealed that the market tends to overreact to low price-to-earnings (P/E) ratios in the long term, indicating a deviation from the notion of market efficiency. Moreover, esteemed professors Andrew Lo and Craig MacKinlay in "A Non-Random Walk Down Wall Street" [4] provided a witty perspective on Burton Malkiel's renowned book [5], showing that the stock market is somehow predictable.

When it comes to market forecasting, there are two primary schools of thought that hold diverging opinions and methodologies. Fundamental analysis evaluates the value of a company by taking into account both qualitative and quantitative factors within a general framework, and decisions are driven by a simple comparison between the estimated and the real value of the company. Technical analysis instead avoids subjectivity and focuses on trends and recurring patterns by looking at past prices [6], thus being closer to time series forecasting.

With the advent of Artificial Intelligence and its several branches, a third and more recent school of thought has gained prominence in the field of market forecasting. This approach leverages machine learning models as a powerful tool for predicting future market trends. It is essential to recognize the impact of Machine Learning by examining how businesses and financial institutions have embraced these advancements.

In today's landscape, organizations rely on Machine Learning techniques to enhance their decision-making processes and strategically plan for the future. These latter offer valuable insights and enable proactive development strategies. Notably, one of the most significant breakthroughs in the financial sector is algorithmic trading, also known as

automated trading or black-box trading, which has revolutionized the approach to stock prediction. Algorithmic trading serves as the cutting edge of innovation in this domain, providing new avenues for optimizing trading strategies and leveraging the power of machine learning to navigate the complexities of the market. These algorithms possess the capability to process and analyze vast volumes of historical and real-time market data, enabling them to detect valuable patterns, trends, and anomalies that can drive profitable trading opportunities. Machine learning models, in particular, have gathered significant attention due to their adaptive nature: the critical importance of accurate predictions becomes apparent considering the substantial sums of money at stake within milliseconds of trading. However, despite the noticeable results achieved by machine learning in financial forecasting, some companies may exhibit hesitancy in embracing these techniques for various reasons. Firstly, there may exist a lack of understanding or awareness regarding the potential benefits that machine learning can offer. This often leads to scepticism and reluctance, as companies may be uncertain about the return on investment or the feasibility of implementing such complex systems. Furthermore, the integration of machine learning into an organization necessitates substantial changes to existing processes, infrastructure, and a skilled workforce. Historically, many disruptive technologies have faced slow adoption in industries. As clearly outlined in the Hype Cycle (Figure 1), developed by Gartner Inc., after an initial period of inflated expectations, disillusionment often sets in, sometimes impeding the smooth integration of such technologies.

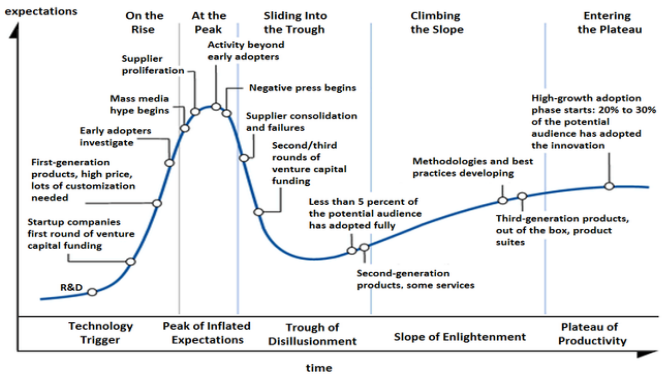

Figure 1: Hype Cycle of a generic technology. Source: [7]

Reaching the plateau of productivity is not an easy task, since each innovation brings a certain degree of risk in the picture. In summary, the decision to incorporate machine learning is a complex one that requires a careful evaluation of potential downsides and rewards, as well as the company's ability to embrace technological innovation. AI-driven technologies are gradually revolutionizing various fields, although their full potential is yet to be fully realized.

Despite all these rapid changes, this thesis focuses on univariate time series forecasting and explores the extent to which markets can be forecasted. After a brief literature review in Chapter 1, some background knowledge is given about time series, classical models and deep learning techniques in Chapters 2, 3, 4 respectively. Then, in Chapter 6, the datasets chosen for this analysis are presented. This study seeks to develop different architectures, a classical Auto Regressive Moving Average method (ARMA) together with a Deep Learning model, Long-Short Term Memory (LSTM) model, on logarithmic returns, in order to play with different granularities. The aim is to discover strengths and pitfalls of these models: choosing the most suitable is a trade-off between performance and complexity while adhering to the *parsimony principle* [2] [8]. In addition to those aforementioned, Prophet library is also included in this work and all the results are displayed in Appendix B.

[2]In economics, the principle is also known as *Occam's Razor* and states that a simpler model with fewer parameters has to be preferred over more complex models with more parameters. The optimal model always involves striking a balance between the goodness of fit of the model and the number of regressors included.

Chapter 1

Literature Review

Forecasting is an ancient problem dating back thousands of years ago, sometimes associated with magical power or inspiration from gods, while sometimes with suspicious activities. Despite its connotation, the ability to foresee the future has always been intriguing to humans for a *plethora* of activities, ranging from medicine to weather, scheduling problems, economics and finance. In some situations, when there is a lack of historical data, or data is not yet available, the only possibility is *judgemental forecasting* [9], though it turns out to be highly subjective and the forecaster has to prove sufficient knowledge in a certain domain. Instead, when data is at hand, quantitative forecasting methods can be resorted. The field of time series forecasting has been a promising research area starting from the early 1980s when the Journal of Forecasting and the International Journal of Forecasting were founded. To give a rough estimate, since then, more than 940 papers about forecasting models were published, most of which have been summarized by authors De Gooijer and Hyndman [10] for the 25th-anniversary occasion, in 2005. However, without a universal consensus on the performance metric to use in evaluating them, establishing a ranking for the most suitable econometric model to apply is almost an impossible task. As a general guideline, Box and Jenkins [11] came up with a three-stage model to develop an ARMA or ARIMA model that best fits the time series data. ARIMA model stands for Auto-Regressive Integrated Moving Average model, as it includes both an Auto-Regressive component and a Moving Average component with differencing. The main benefit of using the ARIMA model is to transform a non-stationary series into a

series without seasonality or trend by applying finite differencing of data points, one or even more times. However, many researchers and statisticians have looked into the limitations of Moving Average models like ARIMA and SARIMA models. The major drawback is that they are regression-based approaches, therefore they are not able to tackle non-linear relationships between parameters. In addition, to have a meaningful insight, certain assumptions about the data must hold, for instance, a constant standard deviation in error terms. Since the advent of Artificial Neural Networks, which will be explored in more detail in the next chapters, the stock prediction problem has gained even more popularity over the last thirty years [12]. The vast majority of the work is about exploring more complex models, such as Convolutional Neural Networks (CNNs), Long-Short Term Memory (LSTM) and Recurrent Neural Networks (RNN). A remarkable advantage of these latter is their capability to handle a massive amount of data, with a faster pre-processing phase and no particular assumption behind it. Wamkaya *et al.* [13] carried out some experiments on three NYSE stocks using ANNs. Subsequently, several other works can be cited regarding the study of CNNs, which achieved good performance results, more specifically S. Chen *et al.* [14] who proposed the use of CNNs to predict Chinese stock market movements, or M. Hiransha *et. al* [15] who showed how NNs of different kinds outperformed ARIMA models on the National Stock Exchange of India and NYSE. In a comparative study on minute-wise stock price data by Gopalakrishnan *et al.* (2017) [16], the LSTM model outperformed a vanilla RNN, which again outperformed simpler linear models. Another fundamental and rigorous paper on six stock indices by J. Yue *et al.* (2017) [17] reached the same conclusion about LSTMs. In line with the aim of this thesis, it is worth mentioning that there is a prominent interest in testing simpler models against state-of-the-art DNN models. In particular, in "Do We Need Deep Learning Models for Time Series Forecasting?" by S. Elsayed *et al.* [18], nine datasets are examined along with eight different DDN models presented at top-level conferences. The results indeed show that a well-engineered Gradient Boosting Regression Tree (GBRT) can compete or even outperform such complex architectures, therefore suggesting not to disregard baseline models. Given the sequential nature of financial data, Transformer

based architectures are picking up pace as well. One of the very first applications aimed at forecasting S&P500 volatility [19]. With the increased computational power and advancements in the field of Artificial Intelligence and its branches, forecasting is no longer limited to past observations only. Natural Language Processing (NLP) is growing rapidly too, thanks to its versatile applicability in several fields, some of which are text summarization, sentiment analysis and text generation. The large amount of data that constantly floods our world can be exploited for many analyses to improve predictions. This is the reason why many researchers are trying to come up with and experiment with hybrid models or combine them to gain a complete understanding of the problem. For instance, Ding *et al.* (2015) [20] proposed a news-based stock prediction framework that uses a deep learning-based sentiment analysis model to extract sentiment information from news articles, and combine it with historical stock prices to make future predictions. The incorporation of textual data significantly improves forecasting accuracy compared to using only financial indicators. Transfer Learning is gaining ground among NLP practitioners, not only for text-related tasks but also for Time Series Classification (TSC) as proposed by H.I. Fawaz *et al.* in "Self- and Transfer-Learning for Time Series Forecasting with Deep Neural Network" (2018) [21].

The main idea behind transfer learning is to leverage the power of pre-trained Deep Neural Networks, in this case, Convolutional Neural Networks, and then transfer the learned weights to a second model, avoiding implementing it from scratch. In such a way, the network has to be fine-tuned on a different dataset, but exploiting what has previously been learnt. While for image and audio data such a method works well, TSC is quite a relatively new area and limited progress has been made so far. One of the main limitations is that time series data is very domain-specific, compared to sequential NLP data, as T. Zhou *et al.* pointed out in his paper "Power Time Series Forecasting by Pretrained LM" [22]. In this latter, cross-domain knowledge is investigated as known to be one of the most challenging tasks, with a particular focus on GPT-2 (Radford *et al.*, 2019) [23], BERT [24] and BEiT [25]. A Frozen Pre-trained Transformer (FPT) is employed, which consists of using a pre-trained Transformer model on image data and then using it

without altering the feed-forward and self-attention layers of the residual block. Overall, the application of such techniques to time series forecasting is a promising field of research, nevertheless, both DL and NLP models often face the problem of limited or insufficient training samples, in contrast to classical econometrics models, which are more robust even working with smaller datasets. Moreover, a relevant issue remains the interpretability and the architecture of the model, this is why we often speak of *black-box* models. In light of this remark, the Temporal Fusion Transformer model (TFT), which leverages the self-attention mechanism to capture the dynamics of multiple time sequences, seems to be a reasonable proxy. In "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting" by B. Lim *et al.* (2019) [26], an alternative to a classical Transformer model is presented, with the ability to capture both temporal and cross-sectional dependencies in the time series data. The self-attention layers are responsible for detecting long-term dependencies, whereas recurrent layers are for local processing. TFT allows programmers to understand how each input feature contributes to the forecast, therefore revealing what is under the hood. Though, the use of dilated convolutions and cross-sectional attention mechanisms, together with several stacked layers, makes this model computationally expensive compared to simpler statistical models. Starting from the basic structure of a Transformer, as designed by Vaswani *et al.* in 2017 [27], another extension is under the attention of scientists, namely the Transformer-XL model [28]. The purpose is to address the problem of context fragmentation with the use of a segment-level recurrence mechanism and a sinusoidal positional encoding scheme. Transformer-XL [29] can learn dependency that is 80% longer than RNNs, 450% longer than vanilla Transformers and it is faster in the valuation phase, in addition, it has no restriction of the input size length.

This literature review is supposed to be a concise summary of the current landscape, however, it is far from being exhaustive due to the continuous proliferation of articles and news. Ultimately, there is still much room for improvement: further investigation is needed to explore the effectiveness of such techniques for time series data and to develop ready-to-use models.

Chapter 2

Basic Concepts of Time Series

2.1 Time Series definition and Properties

A time series can be defined as a collection of sequential data points, where a specific metric, for a singular entity, is measured at periodic time intervals. Depending on the frequency of data acquisition, time intervals can range from yearly to monthly, weekly, daily, hourly, or even seconds. Essentially, a time series offers a temporal perspective on a particular variable, providing insights into how it has evolved over time. As such, given the abundance of this type of data, especially in the *Big Data era*, the need for a systematic forecasting approach is critical. Time series analysis is widely employed in various domains, in particular, to detect patterns, forecast future trends, and make well-informed decisions. In the field of time series analysis, a sequence of data points that describes a solitary variable is referred to as a *univariate* time series. Conversely, if multiple variables are observed at the same time intervals, the resultant data sequence is classified as a *multivariate* time series. A general notation to describe a discrete time series stochastic process is :

$$\{Y_1, Y_2, \ldots, Y_t\} = \{Y_t\}_{t=1}^{\infty}$$

namely, a sequence of random variables indexed by time t, defined on a common probability space. In modelling time series data, the ordering imposed by the time index

is fundamental because we are often interested in capturing the temporal relationships, if any, between the random variables in the stochastic process. A *realization* of the process with T observations is denoted as:

$$\{Y_1 = y_1, Y_2 = y_2, \ldots, Y_t = y_t\} = \{Y_t\}_{t=1}^{T}$$

Strictly related to the notion of stochastic process, we must introduce the concept of stationarity, which is nothing but a form of *'equilibrium'*. Time series stationarity refers to the property of a time series for which the statistical properties such as mean, variance, and autocorrelation remain constant over time. Formally, weakly stationarity implies that:

$$E[y_t] = E[x_{t-h}] = \mu \; \forall h$$

$$Var[y_t] = Var[y_{t-h}] = \sigma^2 \; \forall h$$

$$Cov[y_t, y_{t+h}] = \gamma_h$$

where γ_h exists and is finite. In other words, the process has the same mean and variance at all time points, whereas the covariance does not depend on the location of the points along the time axis. Particularly, a stochastic process is said to be *strictly stationary* if the joint probability distribution function of the observations is independent of the time t for any lag.

Stationarity is an important assumption in many time series models and techniques, furthermore, violations of stationarity can lead to inaccurate results, this is why checking for stationarity is always important. In practice, data showing a crystal clear trend or seasonal patterns are non-stationary by nature. In such cases, some techniques such as differencing and log transformations are resorted to make the series stationarity.

The Augmented Dickey-Fuller (ADF) test, also known as the *Unit Root test*, serves as a confirmatory method for determining whether a time series is stationary or not. In probability theory and statistics, a unit root is a characteristic of certain stochastic processes, which can cause issues in statistical inference involving time series models.

The test is conducted with the following assumptions:

- Null Hypothesis *(H0)*: series is non-stationary or series has a unit root;
- Alternate Hypothesis *(H1)*: series is stationary or series has no unit root.

If the null hypothesis fails to be rejected, the test provides evidence that the series is non-stationary. Rather than manually performing the ADF test, programming languages like R and Python offer useful packages for data analysis. The resulting output includes the p-value, the test statistic value, the number of lags considered in the test, and the critical value cut-offs. Conventional statistical rules are used to determine whether to reject the null hypothesis in favour of the alternative.

2.2 Why do we need time series?

Following this brief introduction, one may reasonably question the necessity of conducting time series analysis. However, in today's world, statisticians and researchers are pretty much occupied with analyzing the patterns behind data. In fact, it is risky to overlook such data because it frequently has substantial value, signifying a strong *competitive advantage* over rivals. The first reason for inspecting time series is that they can help us to predict future trends and behavior based on historical data, as well as to track performance over time. Additionally, scrutinizing this form of data can prove useful in identifying anomalies or unusual occurrences that need further exploration. Lastly, time series data can be used in many fields to evaluate the efficacy of interventions or decisions made over a given period.

2.3 Time Series Components

Time Series are mainly analyzed using statistical tools. Once the data has been collected, the initial step involves carrying out an exploratory analysis to gain deeper insights of its characteristics. A time series is supposed to be affected by four main components:

1. T_t - which is the trend component at time t and reflects the progression of the series. A trend is spotted whenever there is a persistent increasing, decreasing or flat direction in the data;

2. C_t - which is the cyclical component at time t, referring to the irregular, non-repeating fluctuations in a time series that occur over long periods;

3. S_t - which is the seasonal component at time t, suggesting that the data is influenced by seasonal factors. Seasonality occurs over a fixed and known period (e.g., the quarter of the year, the month, or day of the week) and can be caused by extraordinary circumstances (e.g. level of sales during Christmas time);

4. ϵ_t - also called "noise" or residual at time t, which reflects random influences in the model.

Considering the overall effect of these aforementioned components, two types of models are generally used for time series decomposition:

- $y(t) = T_t + C_t + S_t + \epsilon_t$ (Additive model)
- $y(t) = T_t * C_t * S_t * \epsilon_t$ (Multiplicative model)

When selecting the best approach to use, it is important to consider that the additive model is appropriate when the seasonal variation remains constant over time and the components are independent of each other, whereas the multiplicative model is more suitable when the seasonal variation increases over time and the components can affect one another.

2.4 Correlation and Autocorrelation

When working with time series data, it is important to understand clearly two additional concepts: correlation and autocorrelation. Correlation refers to the measure of how two time series variables vary in relation to each other. The Pearson correlation coefficient denoted as ρ, is always a number between -1 and 1. More specifically, when $\rho = 1$, the two

series have a perfect linear relationship with no deviations, therefore they move together, conversely, if $\rho = -1$, they vary in opposite directions, but still with a linear relationship. A low value of correlation is synonymous of a weak association.

Instead, autocorrelation, also known as *serial correlation* refers to the degree of correlation of a single time series with its past values. More often than not, when speaking of autocorrelation we mean a 'lag-one' auto correlation, namely we consider only one day lags. Similarly to correlation, auto correlation can ranges between -1 and 1, therefore it can be either positive or negative. Positive autocorrelation means that the increase observed in a time step leads to a proportionate increase in the lagged time interval. Such observations can be plotted in a smooth curve and, by exploiting a regression line, we observe that a positive error is followed by another positive error and vice versa if it is negative. On the other hand, negative auto correlation implies that an increase observed in a time interval step lead to a proportionate decrease in the lagged time interval. In this scenario a positive error will be followed by a negative one and vice versa and no smooth curve can be fitted, the residuals are scattered. Auto correlation plot can be easily obtained using the corresponding command in Python, moreover if the time series is stationary, the plot will drop to zero relatively quickly, whereas with white-noise data (non-stationary) it will decrease slowly. In order to test for autocorrelation, the Durbin-Watson statistic is generally resorted. Under the null hypothesis, the errors of the least squares regression are serially uncorrelated, whereas the alternative states that they follow a first order auto-regressive process. The test statistic can be computed as follows:

$$DW = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2}$$

where T is the number of observations and e_t is the residual obtained as $e_t = \rho e_{t-1} + \nu_t$. The value of DW always lies between 0 and 4, moreover if T is large, the test statistic can be approximated as $DW = 2(1 - \hat{\rho})$, where $\hat{\rho}$ is the sample autocorrelation. At the threshold value $DW = 2$, meaning that the autocorrelation is zero, instead if $DW < 2$ the autocorrelation is said to be positive, negative if $DW > 2$.

Chapter 3

Classical Approach

3.1 AR(p)

The Auto-Regressive model [1] is a simple, yet powerful, approach to analyzing time series data. Its underlying principle is to exploit past observations of the phenomenon to predict its future values, assuming the data to be stationary. This family of models is essentially a linear regression model where each regressor is a past observation, with its corresponding weight. The variable "p" in AR(p) represents the number of past values or *lags* included in the regression.

Mathematically, it can be expressed as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t = c + \sum_{k=1}^{p} \phi_k y_{t-k} + \epsilon_t$$

where y_t is the future value of the time series, c is the intercept, ϕ_k the single weight and ϵ_t is the error term at each time step. To estimate the set of weights, the method of maximum likelihood estimation is often adopted. The model can be evaluated by analyzing the residuals to ensure that they are uncorrelated and have constant variance.

[1] The term "auto-regressive" is self-explanatory, the model is regressing the variable of interest against its past values.

3.2 MA(q)

The Moving-Average method is another popular model used for estimates. Unlike the AR method, that uses past values of the response variable, the MA model utilizes the past forecast errors or residuals to predict future values. The MA(q) model assumes that the current value of y depends linearly on the q most recent forecast errors, with a constant mean and variance. The q denotes the order of the MA model, which refers to the number of lagged forecast errors used in the model. Mathematically, it can be expressed as:

$$y_t = c + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q} + \epsilon_t = c + \sum_{k=1}^{q} \theta_k\epsilon_{t-k} + \epsilon_t$$

where, again, y_t is the future value of the time series, c is the intercept, θ_k the single weight and ϵ_t is the error term at each time step. The error terms, used as predictors, are also called random shocks and they are assumed to be uncorrelated and to follow a gaussian distribution, with zero mean and constant variance. The Moving Average (MA) model is highly effective in capturing short-term variations, such as sudden shocks or seasonal patterns, in time series data. However, it may not be the most suitable approach for modeling long-term trends, since it relies solely on past forecast errors, rather than the actual time series values.

3.3 ARMA(p,q)

The ARMA(p,q) model is obtained by combining the two models just described above, in fact, is can be written as their simple sum:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i\epsilon_{t-i} + \epsilon_t$$

However, a major flaw of this model is its assumption of *stationarity* in the time series being analyzed, which is frequently not valid in real-world settings. As a result, it is fundamental to ensure the presence of this property. The ARIMA model offers a more

resilient extension of the ARMA model, specifically designed to handle non-stationary time series by utilizing differencing operations.
One technique for promptly identifying stationarity in the data is to examine its decomposition. However, relying solely on graphical analysis can be misleading, therefore, statistical tests are commonly employed for this purpose.

3.4 ARIMA(p,d,q)

ARIMA is an acronym for Auto-Regressive Integrated Moving Average model, which is widely used in statistics and econometrics. It is a generalization of the Auto-Regressive Moving Average (ARMA) model, designed to overcome the limitation that ARMA is only suitable for stationary time series data, by incorporating differencing into the model.
The non-seasonal ARIMA(p,d,q) model is made of the following components:

- Auto-regressive (AR) - a model that employs a variable to regress on its own lagged values. The number of lagged observations, namely the lag order, is denoted as p;
- Integrated (I) - it refers to the degree of differencing applied to the raw data to render the time series stationary. This component is also known as the degree of differencing and is represented by d;
- Moving Average (MA) - a model that employs a variable to regress on residual errors. The order of the moving average is denoted as q.

Since ARIMA incorporates differencing in its model building process, it does not strictly require the empirical data to be stationary. To ensure that ARIMA model works well, the appropriate degree of differencing should be selected, so that time series is transformed to stationary data after being de-trended.

3.5 SARIMA(p,d,q)(P,D,Q,s)

The SARIMA (Seasonal Auto-Regressive Integrated Moving Average) model is an extension of the ARIMA model that incorporates the seasonality component into the

forecasting process. In this case, the model is characterized by four parameters, (p,d,q) for the non-seasonal part and (P,D,Q,s) for the seasonal part. The true novelty is the introduction of the last parameter s, which specify the seasonal length of the data. With its ability to capture both short-term and long-term dependencies, this model provides a flexible but robust framework for forecasting time series data.

3.6 Box-Jenkins Methodology

After a brief overview of the various models, we have to address the concern of how picking up the most appropriate and how to set its order. As anticipated, George Box and Gwilym Jenkins developed a practical method to build the ARIMA model, which best fits the data and satisfies, at the same time, the principle of parsimony. This methodology consists of three steps: *model identification, parameter estimation* and *diagnostic checks*.

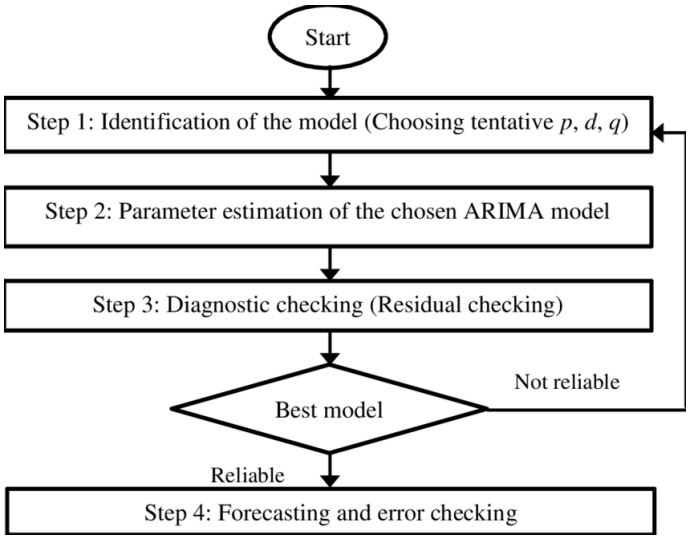

Figure 2: Box-Jenkins methodology. Source: [30]

More in detail, for the first step, it is necessary to check for stationarity of the data by plotting the raw data to get a general idea. There are several tools used for this scope, in particular the autocorrelation function, the ACF plot and the Augmented Dickey-Fuller (ADF) test. If the data appears to be non-stationary, transformations of the data will be carried out. In most cases, first-order differencing is enough to proceed with an ARIMA model. Then with a trial-and-error approach, the best values for p and q

are determined, using some metrics such as the Akaike-Information-Criterion (AIC) and Bayesian Information Criterion (BIC). Lastly, it is a good practice to run some diagnostic checks on the calculated residuals. The model can be considered reliable if the residuals are independent of each other, with constant mean and variance over time, namely, they are *white noise* or *i.i.d.*. Moreover, one can take a closer look at the ACF plot of the residuals or interpret the Jarque-Bera (JB) statistic for the normality test. This procedure is iterated several times (Figure 2) if the best model is not immediately identified.

Chapter 4

Deep Learning Approach

This section provides some introductory information about deep learning history and models, particularly in the context of time series forecasting.

4.1 The Artificial Neuron

Artificial Neural Networks (ANNs) are considered as the building blocks of several architectures in deep learning. As their name suggests, the origin can be traced back to 1943, when Warren McCulloch, a neuroscientist, and Walter Pitts, a logician, published in "A logical calculus of the ideas immanent in nervous activity" [31] the Threshold Logic Unit (TLU), the very first artificial neuron [32], that mimics the functionality the biological counterpart. A simplified version of a biological neuron can be visualized in Figure 3.

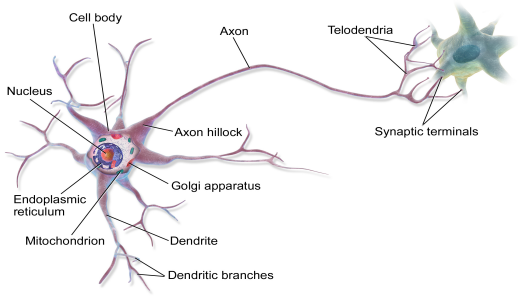

Figure 3: Schematic drawing of a biological neuron. Source: [33]

Essentially, electrical signals travel towards the neuron cell body across several dendrites.

Inside the body, the signals are added together and compared to a threshold value. If the resulting signal is greater than the threshold, the neuron is considered to be firing and a new signal is propagated through the neuron's axon. Conversely, if the signal does not reach the threshold, the neuron remains inactive, as the neuron's activity follows an *all-or-none* principle. Large neural networks can be easily built by tying together several neurons via dendrites and axons.

4.2 The Perceptron

The Perceptron[1], introduced by Frank Rosenblatt in January 1957 at Cornell Aeronautical Laboratory [34], was the first algorithmically represented neural network, inspired by the M-C neuron. The purpose of a Simple Perceptron is to perform binary *classification* tasks. A Simple Perceptron is composed of three main elements: synapses, adders and activation functions. Synapses are the connections between two different neurons of the net, each of which has its corresponding synaptic weight. The term adder represents the linear combination between inputs and their weights. Lastly, activation functions are non-linear functions used to limit the output magnitude. Their role is key they introduce non-linear operations to the chain, thus preventing networks from collapsing into a single neuron. There are several examples of activation functions, including the sigmoid function [2], that maps an input into a real value between 0 and 1, but also a group of *piece wise* linear activation function, like the $ReLU$ function [3] and its derivative *Leaky* $ReLU$, which are indeed the most widespread. From a mathematical standpoint, a Perceptron can be described as:

$$\mathbf{y} = \phi(\sum_{i=1}^{n} w_i x_i + b) = \phi(\mathbf{w^T x} + b)$$

where $\phi(\cdot)$ is a generic activation function, $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, $\mathbf{y}$ the output vector and b is the bias component.

The Perceptron learns from a labelled training dataset and then adjusts the weights of

[1] The first concrete implementation was the Mark I Perceptron machine, designed for image recognition, with an array of 400 photocells.

[2] $y = \sigma(z) = \frac{1}{1+e^{-z}}$

[3] $f(x) = x^+ = max(0, x)$

the inputs during the training process to minimize classification errors. The learning algorithm [4], also known as the Perceptron Learning Rule or the Delta Rule, updates the weights, therefore the decision boundary, based on the previously observed error between the predicted outputs and the true labels (Figure 4). This iterative learning process continues until the perceptron classifies correctly all the available examples.

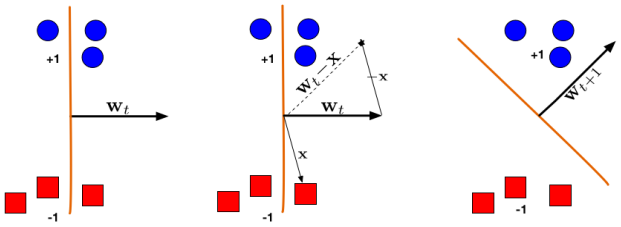

Figure 4: Perceptron learning process. In the left figure, the plane defined by w_t misclassifies one red and one blue point. In the middle figure, the red point x is chosen and used for an update. Because its label is -1 we need to subtract x from w_t. On the right, the updated plane $w_t + 1 = w_t - x$ separates the two classes and the Perceptron algorithm has converged. Source:[35]

At this point, it seems clear that Neural Networks are not just a simple collection of artificial neurons, conversely to make them work, they must be trained: their weights must be tuned accordingly to minimize the loss function. The most common method is *back-propagation*, an algorithm that resorts gradient descent to decide how to modify the weights in order to improve the loss function. While the Perceptron is the basis for most NNs, a Single-Layer Perceptron can only learn linear binary classifiers, therefore it is only able to deal with perfectly separable data. To tackle more complex and high-dimensional problems, such as the XOR logic operator, Multi-layer Perceptron (MLP) is the solution as it introduces non-linearity. In addition to input and output layers, it has also one or more hidden layers in between; it is a feed-forward neural network, which means that the input data is fed through the network in a single direction, from left to right, with no loops or connections. Each neuron in an MLP receives input from the neurons in the previous layer, performs a weighted sum of the inputs, adds a bias term, and then applies an activation function to the result. The output of each neuron is then passed as input

[4]The Perceptron learning algorithm is typically considered an *online* algorithm as the model is continuously updated as new data points are observed, namely it scans the training dataset observation by observation.

to the neurons in the next layer, and so on until the final output is produced. A MLP with a single hidden layer can be represented as:

$$\mathbf{h} = \phi(\mathbf{W}^x x + \mathbf{b}^x)$$

$$\mathbf{y} = \mathbf{W}^y \mathbf{h} + \mathbf{b}^y$$

where $\mathbf{W}^x$ is the weight matrix, used to map the inputs to the hidden layer, and $\mathbf{W}^y$ is used to map the hidden layer to the output layer, the same applies to the bias component b. The weight matrices in MLPs are usually initialized randomly with values in $(0, 1)$, or sampled from a uniform or normal gaussian distribution. Another widely used technique is the Xavier/Glorot initialization [36], commonly adopted together with a *tanh* activation function to fulfil the assumptions behind the mathematical proof. ANNs are nothing but Multi-layer Perceptron, having more than one hidden layer. Choosing the best design of an ANN architecture depends on several factors, namely the type of problem being solved, the size of the dataset, and the computational resources available. In the upcoming sections, we will delve into models specifically designed to handle sequential data, the scope of our research.

4.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were born as an extension of traditional feed-forward neural networks. The need for RNNs arises from the limitations of Multi-layer Perceptrons (MLPs), in fact, unlike RNNs, MLPs cannot model sequences, lack a temporal structure and can only process fixed-size inputs and outputs.

Moreover, while feed-forward networks process input data in a single pass through the network with no memory, RNNs have loops that allow them to maintain an internal state that can represent the context of the input data. A Vanilla Recurrent Neural Network can be formulated as:

$$\mathbf{h}_t = \phi(\mathbf{W}^x \mathbf{x}_t + \mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{b}^h)$$

$$\mathbf{y}_t = \mathbf{W}^y \mathbf{h}_t + \mathbf{b}^y$$

where $\mathbf{W}^x$ is the weight matrix applied to the input, $\mathbf{W}^h$ is used to map values from the previously hidden layer to the next one and $\mathbf{W}^y$ is the weight matrix that maps the values from the current layer to the output layer. The recurrent structure can be better visualized in Figure 5:

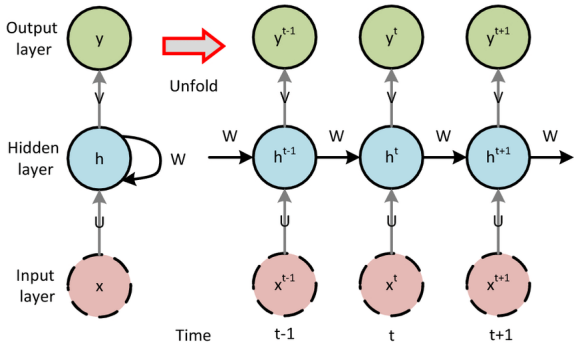

Figure 5: Computational graph of RNN. Source: [37]

One peculiarity of RNNs is that the parameters in the network are shared across time and thus inputs, at each time step, affect the model parameters. Given the different structures, the back-propagation algorithm is no more useful in this context. Starting from the 1980s, Paul Werbos [38] and many other researchers [39], independently introduced the concept of back-propagation through time ($BPTT$), a variant of the back-propagation algorithm used in feed-forward networks. BPTT accounts for the temporal nature of RNNs by unrolling the recurrent connections across time steps, each of which is considered as a separate layer. The gradients of the loss function are computed at each time step and accumulated over time. The process of optimization using BPTT is indeed very expensive as the gradients of the model have to be calculated throughout the sequence. Another downside of this architecture is related to long back-propagation sequences. Moving along the states, the gradients will tend to vanish or even explode as the sequence gets longer. Both of these scenarios may represent a serious threat to the model. In the first case, if the weights are small, the gradients can become exponentially smaller with each layer,

and this leads to slower convergence or perhaps, it can cause the network to stop learning altogether. To address this issue, several techniques are adopted, including:

1. Weight initialization - setting the initial weights in a manner that prevents them from becoming excessively small or large can mitigate the problem of vanishing gradients;

2. Non-saturating activation function - using activation functions that do not saturate [5] for large inputs, such as the Rectified Linear Unit (ReLU), Leaky ReLU, Parametric ReLU (PReLU);

3. Batch normalization - applying batch normalization to the inputs of each layer can help to stabilize the distribution of activation and gradients, making it easier to propagate gradients through the network.

Instead, in the case of exploding gradients, one of the most popular methods is using gradient clipping [40], namely choosing a maximum value that the gradient can attain, or even reducing the learning rate while training the network.

4.4 Long-Short Term Memory

In 1997, to overcome the severe problem of gradients, Sepp Hochreiter and Jürgen Schmidhuber proposed the Long Short-Term memory (LSTM) [41] model as a solution. Compared to a recurrent cell, LSTM features some novelties:

1. memory cell structure - which can store and propagate information over multiple time steps, allowing LSTM to preserve long-term dependencies in the data;

2. gating mechanism - three gates control the flow of information within the network, enabling LSTM to selectively store and retrieve information as needed.

The structure of an LSTM unit is presented in Figure 6:

[5] The term 'saturate' is used to indicate output values that are very close to the upper or lower bound of the activation function.

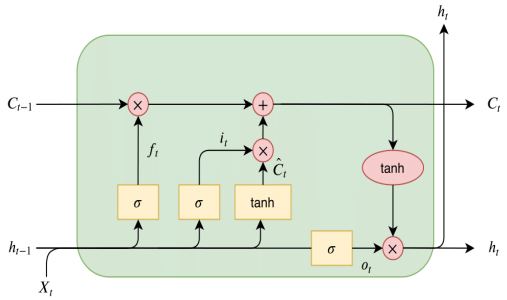

Figure 6: Structure of a LSTM unit. Source: [42]

In the figure above f_t, i_t and o_t are the vectors of the forget gate, input gate and output gate respectively. The input vector is denoted as x_t, whereas h_t is the hidden state vector. Then c_t and $\tilde{c}_t$ are the cell state and the cell input activation vectors. All the gates can be rigorously formulated as follows:

$$f_t = \sigma_g(\mathbf{W^f x_t} + \mathbf{U^f h_{t-1}} + \mathbf{b}^f)$$

$$i_t = \sigma_g(\mathbf{W^i x_t} + \mathbf{U^i h_{t-1}} + \mathbf{b}^i)$$

$$o_t = \sigma_g(\mathbf{W^o x_t} + \mathbf{U^o h_{t-1}} + \mathbf{b}^o)$$

$$\tilde{c}_t = \sigma_c(\mathbf{W^c x_t} + \mathbf{U^c h_{t-1}} + \mathbf{b}^c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$y_t = h_t = o_t \odot \sigma_h(c_t)$$

with $\mathbf{W}$, $\mathbf{U}$ and b being the weight matrices, recurrent weight matrices and bias vector of different gates of the model, indexed by gate type, whereas σ_g, σ_c, σ_h are different activation functions and the $\odot$ is the Hadamard product (or element-wise product).

The input gate determines which elements in the long-term memory get updated by input data from the current time step, the output gate selects long-term memory elements which should be moved to the short-term memory and, lastly, the forget gate is used to decide which long-term memory pieces must be erased, *i.e.* set to zero.

Although LSTM represents a remarkable improvement from RNN, it still has some pitfalls. Among these, it is worth mentioning:

- Computational complexity - LSTMs are computationally more expensive than simpler models like feed-forward neural networks, due to the higher number of parameters and Back-Propagation Through Time algorithm;
- Difficulty in training - LSTMs can be difficult to train, especially if the input sequences are very extended;
- Limited memory - despite their name, LSTMs can still struggle in remembering information over very long sequences. This is because the memory cell can become saturated or forget important information over time.

4.5 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) [43] is a simpler variant of the aforementioned Long-Short Term Memory model. This unit has only two gating mechanisms, the update z_t and reset r_t gates. The update gate combines the functionalities of the forget and input gates. Instead, the reset gate decides how much of the previous information of the previous hidden state should be carried forward to the current state.
The following equations show the computations of all the gates and output vectors:

$$\mathbf{z}_t = \sigma_g(\mathbf{W}^z\mathbf{x}_t + \mathbf{U}^z\mathbf{h}_{t-1} + \mathbf{b}^z)$$

$$\mathbf{r}_t = \sigma_g(\mathbf{W}^r\mathbf{x}_t + \mathbf{U}^r\mathbf{h}_{t-1} + \mathbf{b}^r)$$

$$\mathbf{h}_t = (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \sigma_h(\mathbf{W}^h\mathbf{x}_t + \mathbf{U}^h(r_t \odot h_{t-1}) + \mathbf{b}^h)$$

Specifically, the weight matrices and bias vectors are denoted as $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$, while the activation functions σ_g and σ_h are typically the sigmoid and hyperbolic tangent, respectively. The GRU architecture has gained popularity due to its ability to effectively capture long-range dependencies while mitigating the vanishing gradient problem. Compared to an LSTM model, GRU has a simpler architecture, a flexible memory management mechanism, moreover, it is faster to train and performs relatively

well even with smaller datasets. Ultimately, the choice between LSTM and GRU model depends mostly on the specific task one aims to address [6].

4.6 Transformer and Attention Mechanism

Despite being out of the scope of this work, the latest advancement in sequential data processing has been the development of the Transformer model and its numerous variants. As previously anticipated in Chapter 1, this architecture was introduced by Vaswani *et al.* (2017) in the pioneering paper called *'Attention is all you need'* [27] which has been the starting point of a new era. As one can guess from the title, the authors explained the concept of *attention*, which is indeed employed to overcome some of the sequential models' pitfalls. This concept is closely related to neuroscience too: as our brain, when an image is presented, focuses only on certain regions [7], analogously the network focuses on specific parts of the input sequence or image that are most relevant to the task at hand [44].

The basic idea behind the attention technique proposed by Bahdanau *et al.* (2014) [45] in a machine translation-related paper, is to assign weights to different parts of the input, based on how important they are to the output. These weights are learned during training, and they determine how much attention the network should pay to each part of the input when generating the output. This is achieved by using queries Q, keys K and values V that are compared to each other. The correspondence key-query is drawn from the retrieving mechanism of a simple database, here, however, there is not just a single match, in fact, the query matches against all the keys to a certain degree based on their dot product. This latter is therefore nothing but a similarity score between the queries and the keys. Then, the scores are used to weigh the corresponding values to produce a new attention-weighted representation of the input sequence.

[6] In the last couple of years, many researchers have tried to combine them in a hybrid LSTM-GRU model, thus taking advantage of the strengths of both.

[7] Related to images, *visual attention* mechanism is widely used in computer vision applications, such as object detection, visual tracking and image captioning. By assigning attention weights to different regions, the network knows which part has to focus on, leading to faster training. Saliency maps are the pixel-wise representation of these areas, where brighter colors are used to highlight the most 'salient' portions.

The attention mechanism represents a novelty for a variety of reasons: it can access all states in the input sequence and address the problem of vanishing gradients, it is bidirectional by its nature, therefore sequences can be scanned from start to end and vice versa, moreover, unlike RNNs, it allows for some interpretability thanks to the attention scores.

The Transformer model was originally applied for NLP tasks, such as Text Summarization, Neural Machine Translation (NMT), Question Answering (QA), though Transformed-based models have been deployed also for time series analysis tasks. The model consists of an encoder and a decoder, each of which is structured in many layers, as depicted in Figure 7.

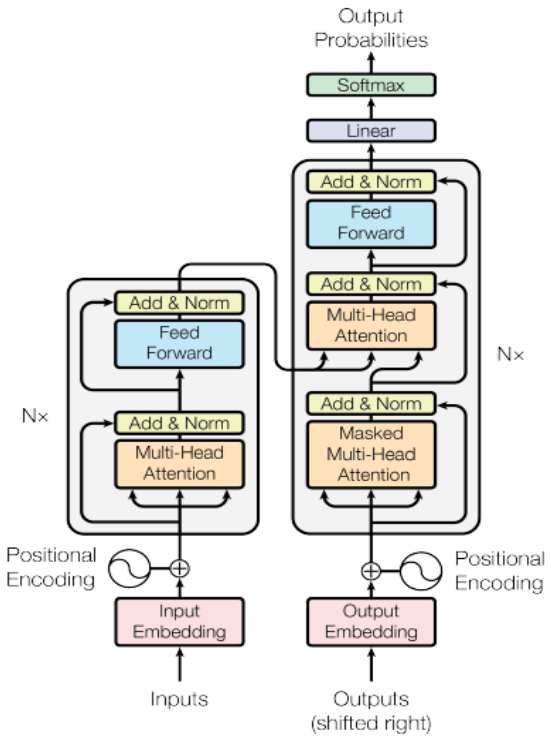

Figure 7: Transformer model architecture. Source: [46]

The encoder takes the input sequence and produces a set of encoded representations, which are then passed to the decoder block. The decoder then generates the output sequence based on these encoded representations. The encoder is made of two main sub-layers, a multi-head self-attention layer, that captures dependencies between different positions in the sequence, and a position-wise fully connected feed-forward layer, that introduces non-

linearity in the model. The decoder is pretty similar, except for a masked self-attention that is first applied, moreover, the multi-head attention in the decoder uses values from both the encoder and decoder. Positional encoding aims at providing more information about the order of the positions in the input sequence, whereas residual connections allow the gradient to flow more easily from one layer to the other. Lastly, the output of the decoder goes through a linear layer followed by a softmax activation function to get the final probabilities over the target variables. Given the query, the key and the values, the attention is computed as:

$$attention(Q, K, V) = softmax\left(\frac{\mathbf{QK^T}}{\sqrt{d_k}}\right)\mathbf{V}$$

where d_k is the hidden dimension of the keys, Q, K, V are the matrices for queries, keys and values, respectively. The square root of d_k as denominator ensures a non-vanishing gradient, as the variance of dot products in the similarity matrix increases as a function of d_k. Compared to RNNs, sequential data is input all at once in a transformer and positional encoding maintains the sequential nature of the data. Moreover, transformers do not have to process the input data from start to end, thus enabling parallelization and lower training time. In the context of time series, a significant contribution is given by the attention mechanism, which allows it to give more importance to certain parts of the inputs, whereas common RNNs or LSTMs do not. In the end, it is interesting to mention that with this new architecture, multi-step-ahead prediction is carried out directly, i.e. forecasts $y_{\{T+h\}}$, with $h \in \{1, 2, ..., H\}$. RNNs are only suitable for one-step-ahead prediction: perhaps one can iterate n times such one-step prediction, though, with this method, errors get propagated, and predictions may be inaccurate.

Chapter 5

Metrics of Evaluation

Selecting the appropriate forecast metric is key when comparing the performance of different models. Depending on the characteristics of the models and tasks, different metrics must be taken into consideration.

5.1 Mean Absolute Error (MSE)

Since we are dealing with a regression problem, therefore with a supervised learning task, the ultimate goal is to minimize the loss function at the optimal point. In particular, Mean Squared Error (MSE) is commonly used and it is computed as the average squared deviations of the forecasted values from the true value of y, namely:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2$$

where y_t it the true value, $\hat{y}_t$ is the predicted value, $(y_t - \hat{y}_t)$ is the error and N indicates the number of observations in the dataset. MSE particularly penalizes extreme errors because of the quadratic dependence the presence of outliers contributes to magnifying its value. Moreover, it is very sensitive to the change in scale and data transformation. Often, a slight variation of the MSE is used, namely the Root Mean Squared Error

(RMSE), whose mathematical formulation goes as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}$$

5.2 Mean Squared Error (MAE)

The Mean Absolute Error (MAE) is the average, in absolute value, of the difference between the model's prediction and the ground truth value:

$$MAE = \frac{1}{N}\sum_{t=1}^{N} \mid (y_t - \hat{y}_t) \mid$$

As the MSE, also the MAE cannot get negative because it considers the residuals in absolute value. The advantage of MAE is that errors are weighted on a linear scale (i.e. no squaring is involved), therefore very bad predictions do not affect too much the overall performance. As a downside, the model may seem accurate, apparently, on the vast majority of data, but in reality, it makes very poor predictions in some other cases. Ideally, when evaluating time series-related models, one aims for lower values of MSE, RMSE and MAE.

5.3 Akaike Information Criterion (AIC)

The Akaike Information Criterion is a statistical measure developed by Hirotugu Akaike in 1974 [47] and it is founded on Information Theory. The AIC is obtained as:

$$AIC = 2k - 2ln(\mathcal{L})$$

where k [1] is the number of regressors included in the model and $\mathcal{L}$ represents the maximum value of the likelihood function of the model. The AIC criterion is derived from the

[1] In the case of an ARIMA model, k is nothing but the sum of the parameters that specify the order.

Kullback-Leibler divergence[2] [48], which measures the information lost when a model is used to approximate the probability distribution of the underlying process. Such a metric penalizes models with many parameters, as the first term in the formula increases proportionally with k. Overall, the AIC aims at striking a balance between the goodness-of-fit and the complexity of the model: a lower value is a synonym for a better model fit. However, AIC per se is not a sufficient measure of model quality and it should be used in conjunction with other considerations when selecting the appropriate model.

[2]Given two discrete probability distributions the KL divergence is computed as $D_{KL}(P||Q) = \sum_i P(i) log_2(\frac{P(i)}{Q(i)})$

Chapter 6

Data Collection

The current and next chapters represent the heart of the empirical analysis. Beginning with the data collection process, details about the data source and granularity are provided, followed by a concise exploratory analysis [1]. This latter aim to enhance the motivations behind model development. The primary objective of this investigation is to explore stock returns predictability using a variety of models, encompassing both traditional econometric models and deep learning approaches.

Data Overview

A quick summary of the main features of the datasets can be visualized in Table 1:

Dataset	Frequency	No. of observations
Nasdaq-100	Minute	32641
AAPL	Daily	3272
WTI Crude Oil	Monthly	433

Table 1: Datasets overview

[1] More details and plots of the exploratory analysis are reported in Appendix A, at the end of this work.

6.1 Nasdaq-100

The first dataset presented for this analysis is the Nasdaq-100 (NDX [2]), perhaps the most symbolic index for innovation and growth in the world of financial markets. It is home to 100-plus of the largest domestic and international non-financial companies based on market capitalization, ranging from technology, the most prominent sector, to health care, utilities and telecommunication (Figure 8), therefore its analysis is crucial for gaining a broader perspective on market conditions. It is worth underlying that there is often confusion between the Nasdaq-100 and the Nasdaq Composite Index, also known as "The Nasdaq." The latter encompasses all Nasdaq domestic stocks listed on the Nasdaq Stock Market.

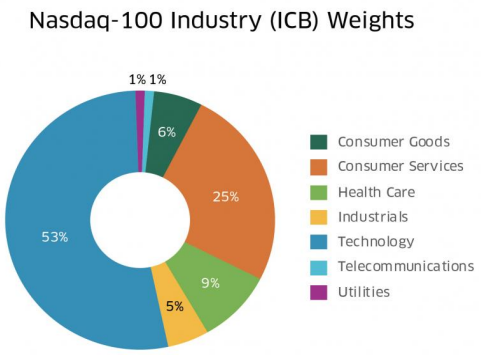

Figure 8: Nasdaq-100 index composition. Source: [49]

Given its significant concentration on technology companies, it provides insights into the health of the tech sector and potential expansion opportunities. Minute-wise data about the index is collected using Refinitiv, a leading platform to access real-time market analytics. The dataset consists of 32641 non-null observations and 9 features, namely Local Date, Local Time, the time indexes, Close, Net, %Chg, Open, Low, High price and Volume. The closing price is further used to compute the log returns, the variable of interest for our models. The time window is approximately 50 trading days, from March until the half of May 2023.

[2]Indicates the corresponding ticker.

6.2 AAPL

Buying the Nasdaq-100 index has increasingly turned into a bet on a six-pack of tech stocks. Apple became the first publicly traded U.S. company to hit a 1 trillion market cap in August 2018 and quite recently, 3 trillion market capitalization. Speaking of the technology sector and given its importance in today's economy, Apple Inc. (AAPL [2]) stock price is investigated. Apple is listed on the NASDAQ-100 exchange, where it encompasses the 12% of shares of the total index. In this case, data is downloaded from Yahoo! Finance [3], just by searching for 'AAPL'. The chosen time window covers approximately 12 years, from 2010 to 2022 and the closing price exhibits an incredible exponential growth throughout the period, with a sharp increase after 2020.

6.3 WTI Crude Oil

To introduce a bit of differentiation in our ideal portfolio and reduce the reliance on the technology sector, we turn our attention to the price of crude oil. Given its significant impact on both political and economic spheres, oil holds a crucial position nowadays, with major oil-producing countries enjoying strategic influence. Overall, the oil price is mainly determined by the amount of demand and supply, but a chaotic component is always at stake. Of course, there are different types of oil, with different peculiarities. West Texas Intermediate (WTI) crude oil, our focus [4], represents the benchmark in the Americas for oil contracts traded on the New York Mercantile Exchange (NYMEX). Given the interest in predicting future oil price variations, lots of research is ongoing, employing both classical and deep learning models [50] [51]. WTI oil price (MCOILWTICO) [2] is gathered from the FRED database, with a monthly frequency, from 1986 to 2022, for a total of 433 non-null observations. As a side note, the unit of measurement for the oil price is the dollars per barrel, not adjusted for seasonality.

[3]Yahoo! Finance is a public domain, authorized by the United States Securities and Exchange Commission (SEC), containing over 40,000 stock quotes.

[4]Instead, Brent Blend is a mixture of crude oil from fifteen different oil fields in the North Sea and is the primary reference in Europe and Africa.

Chapter 7

Modeling and Results

7.1 A premise

The primary objective of investing in financial markets is to achieve profitable returns while managing risks. Common investments involve purchasing assets like stocks, bonds, or deposits and holding them for a certain amount of time. In a few words, profits are realized when the selling price of an asset exceeds its purchase price, taking into account the initial capital, holding period, and price fluctuations. As highlighted in Appendix A, the closing price exhibits substantial variability over time, making it impractical to use for forecasting purposes. The reason behind this is that prices are susceptible to noise and random fluctuations, which may not lead to accurate predictions. Relying solely on the closing price, or even adjusted price, runs the risk of incorporating these random variations, resulting in less precise forecasts. For the scope of this work, the focus will be on *returns*, namely the change in price of an asset over time. Using returns can be beneficial for several reasons:

1. Stationarity - stock prices are often non-stationary, meaning they exhibit trends, seasonality, and other patterns over time. By taking the difference between consecutive prices to calculate residuals (i.e. returns), you can transform the non-stationary series into a stationary one. Stationary time series are generally easier to model and analyze, as they have constant statistical properties over time;

2. Statistical assumptions - many statistical models, including linear regression and auto-regressive models like ARIMA, assume that the residuals are independent and identically distributed (*i.i.d.*) with zero means. By using residuals instead of raw prices, you align the data with these assumptions;

3. Risk measurement: returns are immediately linked to the concepts of risk and volatility. Predicting returns allows for better estimation and forecasting of volatility, which is crucial for risk management and portfolio optimization strategies.

In financial modeling, it is common to replace simple returns with logarithmic returns, because of their nice properties. They are symmetric around zero, meaning that returns of equal magnitude but with opposite signs will cancel each other out, moreover, log returns are time-additive, therefore the total return over an entire period is just the algebraic sum of all the single returns in each sub-period.
Practically speaking, minute, daily and monthly log returns are computed as follows:

$$R_{L,t} = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln(P_t) - \ln(P_{t-1})$$

where $R_{L,t}$ is a shorthand for log returns between time $t-1$ and t, whereas P_t represents the closing price of the stock at time t. It is worth mentioning that logarithmic returns and simple returns are approximately equal. They are linked to one another by the following formula:

$$R_{L,t} = log(1 + R_{S,t})$$

therefore, by considering a first-order Taylor approximation[1], log returns and simple returns are almost identical within the trading day unless big shocks take place.

[1]Note that the Taylor approximation for the function is:

$$log(1+x) \approx log(1+a) + \frac{1}{1+a}x$$

so at the origin, when $a = 0$

$$log(1+x) \approx x$$

.

Figure 9: Simple returns vs. Logarithmic returns. Source: [52]

More specifically, Figure 9 clearly shows that the two returns are very similar in magnitude as long as $R_{S,t} \leq 0.15$ [53]. A large debate about whether to use simple or log returns is going on, mainly because simple returns may be more explicit as they signal what you get in the end. However, for the scope of this analysis, log returns are preferred, especially for some desirable properties, some of which are listed below:

1. Stationarity - values typically fluctuate around a constant level, thus suggesting a constant mean over time;

2. Heavy tails - the distribution of the log returns often shows heavy tails compared to those of a normal distribution. Quantile-Quantile (QQ) plot is generally used for checking normality, together with the Jarque-Bera test [2];

3. Asymmetry - the distribution of return is often negatively skewed, reflecting the fact that the downturns of financial markets are often much steeper than the recoveries. In other words, investors are more sensitive to negative information rather than positive news;

[2]Jarque-Bera test checks whether the skewness and the kurtosis of a statistical distribution match the one of the normal. The statistic is computed as:

$$JB = \frac{n}{6}(S^2 + \frac{1}{4}(K-3)^2)$$

where S is the sample skewness, K is the sample kurtosis and n is the number of observations.

4. Volatility clustering - this term describes the phenomenon where significant price changes, characterized by large absolute values, tend to occur in clusters or groups. Indeed, large price changes tend to be followed by large price changes, and periods of tranquillity alternate with periods of high volatility.

7.2 Data preprocessing

Data preprocessing always plays a crucial role in enhancing the quality of data. In particular, by analyzing these datasets, we spot no missing values or anomalies of any kind. Moreover, as explained above, log returns are used as a regressor of our models. While daily and monthly data exhibit considerable variability over the extensive-time period considered, including multiple years, with high-frequency data is better to discard the first and last hour of trading data. Trading near the bells becomes quite hectic and the level of liquidity, therefore volatility, varies a lot intra-day. Most often than not, when the market opens, exaggerated price movements are observed since traders react to overnight information or adjust their position to take the most out of it. Similarly, when the market approaches closing time, traders may rush to place many orders. To conclude, removing these specific data points may help to eliminate some of the noise and extreme fluctuations [54].

7.3 Rolling window approach

When dealing with time series analysis, several approaches can be resorted to make forecasts, according to the type of data at hand. In this scenario, to compare different granularities of data, a rolling window technique is adopted to make predictions on the test set. In particular, look-back and look-forward variables are defined, so that the model knows how many values to look at, during the training, and how many to predict, respectively. The choice of these values depends on the research question one may want to address, together with the type of data and the amount available. Table 2 shows the window lengths and prediction horizons that are considered for each dataset:

Dataset	Granularity	Look-back	Look-forward
Nasdaq-100	Minute	50 minutes	10 minutes
AAPL	Daily	60 days	7 days
WTI Crude Oil	Monthly	48 months	6 months

Table 2: Summary of the windows lengths

Speaking of low or medium-frequency data, these lengths are quite standard, whereas, with high-frequency data, it could be better to play with them since the more values you include in the training, the better the prediction. Given that is quite difficult to manipulate and execute code on extensive datasets, we try to strike a balance to get a reasonable forecast.

7.4 ARMA implementation

Despite their simplicity, econometrics models are still used nowadays to investigate stock and index behaviour. When it comes to the ARIMA family models, it is important to do some background checks before instantiating the model order and forecast. In fact, as explained in Chapter 7.1, for the purpose of *detrending*, log returns are considered instead of prices. To make sure they are stationary, the Dickey-Fuller test is run on data, confirming that our series has no unit root. Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test and Phillips–Perron (PP) test is further executed, suggesting the same outcome: no need for differencing. Therefore, by this, we can state that the order parameter $d = 0$. Regarding the order of two parameters, p and q, a common practice is by trial-and-error, namely a subset of parameters is used to run the model several times, in the end, the parameters that minimize the AIC score are deemed to be optimal. To do this automatically, Python provides the function auto_arima, which tries multiple combinations of parameters, however, given the data at hand ARMA(1,1) is undoubtedly enough for fitting purposes. More complex models may also be used, though some problems arise quite commonly, for instance, over-fitting, numerical instability and lack of interpretability. According to our initial aim, look back and look forward period are used

to train the ARIMA (1,0,1) model, in particular, inside a for loop, the SARIMAX model is instantiated, then fitted and forecasts are made using the specified look forward variable. In order to plot the results, two lists are created, so that at each iteration forecasts and time indexes are appended. As can be deduced from the plots below, the forecast made by the ARMA is a naive forecast, namely the prediction for the next period is based on the last period and varies around zero.

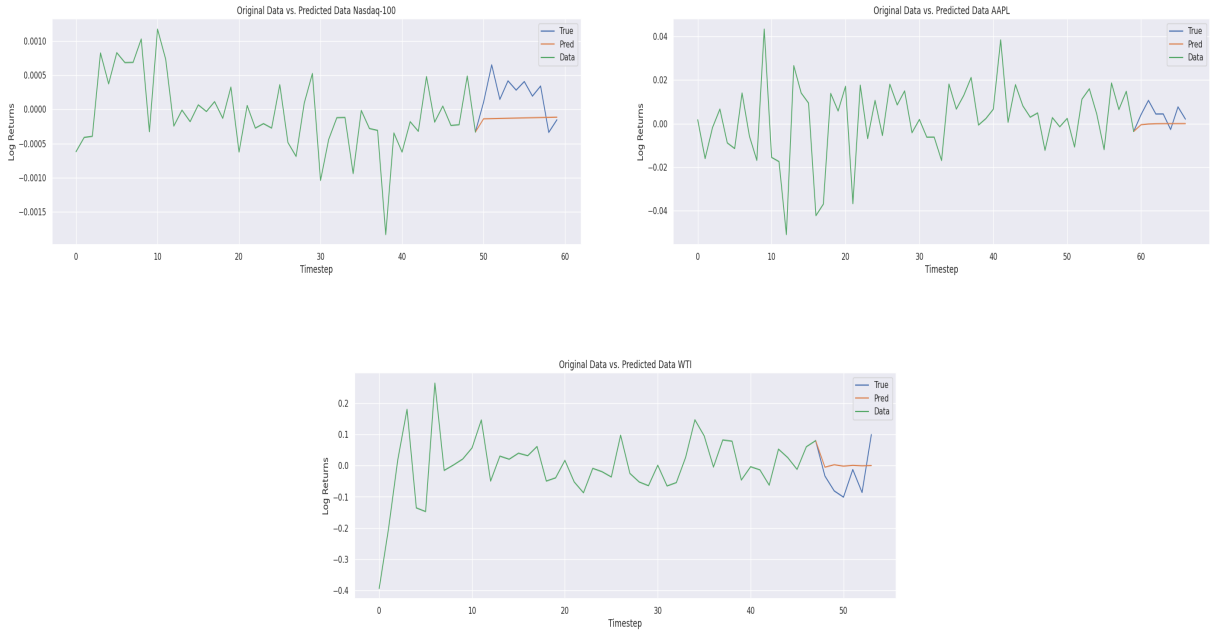

Figure 10: ARMA predictions vs.True data

As one would expect, both the ACF and PACF decline geometrically, in particular, the first plot is the result of the AR process, whereas the second depicts the MA process.

Results

Common evaluation metrics such as MSE and RMSE are employed to compare the performance of ARMA(1,1) in Table 3.

Dataset	MSE	RMSE
Nasdaq-100	1.13724	0.00034
AAPL	0.00054	0.02325
WTI Crude Oil	0.03257	0.18047

Table 3: ARMA model Results: MSE and RMSE

7.5 LSTM implementation

There are several ways to implement an LSTM model using Python, however, the Keras library, a high-level API built on top of TensorFlow is adopted for this work, especially for its easy-to-use interface and scalability while designing the network.

As discussed in Chapter 1, Deep Learning models are capable of dealing with raw data effectively, thus reducing the need for extensive feature engineering. In particular, given that we are dealing with returns and not prices, we do not perform any scaling [3].

The LSTM model is specified by a model builder function, which adds to the Sequential() module an LSTM layer, with a tanh activation function, some Dense and dropout [4] layers. For all the datasets, the Adam [5] optimizer is used, which is one of the current default optimizers in deep learning development. Regarding the loss function, mean squared error loss is selected. The subsequent step is to wrap the LSTM in a KerasRegressor object, a tool to interface Keras models with scikit-learn's cross-validation and hyper-parameter tuning functionalities.

Cross-Validation

When dealing with time series data, the conventional approach of splitting the data into training and test sets may not be enough. Traditional cross-validation techniques are often inadequate for sequential data problems due to the risk of overfitting and the temporal nature of the data. For example, methods like k-fold cross-validation, stratified k-fold cross-validation, and leave-p-out cross-validation are not appropriate as they shuffle the data and do not preserve the temporal coherence between the validation folds and the actual test set.

[3] In other cases, MinMaxScaler() is commonly adopted to re-scale data between 0 and 1, separately in the training and test set, to avoid data leakage. Data leakage refers to a problem where information about the holdout dataset, such as a test or validation dataset, is made available to the model in the training dataset. This leakage is often small and subtle but can have a marked effect on performance.

[4] In Machine Learning, dropout is a regularization technique consisting of ignoring randomly selected neurons. The dropout layer is specified by a rate, which indicates the proportion of 0s and 1s in the instantiated tensor, which is nothing but a mask applied on the previous layer to prevent over-fitting [55].

[5] Adam stands for "Adaptive Moment Estimation" and can be considered as a combination between two other popular optimizers AdaGrad and RMSProp.

To perform cross-validation on the LSTM parameters, BlockTimeSeriesSplit resorts to GridSearch CV, a technique for hyper-parameter tuning that exhaustively searches through a specified grid of hyperparameters to find the best combination of values for a given model. The number of splits can be manually set before instantiating GridSearchCV, as well as the parameters grid and scoring function.

Once the best parameters have been retrieved from cross-validation (Table 4) and stored in an appropriate dictionary, the best model is instantiated using them as arguments.

Dataset	batch_size	drop_rate	epochs	lstm_units
Nasdaq-100	32	0.1	20	64
AAPL	64	0.1	15	128
WTI Crude Oil	32	0.1	15	64

Table 4: GridSearch CV optimal parameters

Results

After completing the training phase, predictions are made using the test data set apart. Finally, standard evaluation metrics like MSE and RMSE are used to measure the discrepancies between predictions and ground truth data (Table 5).

Dataset	MSE	RMSE
Nasdaq-100	1.74077e-07	0.00042
AAPL	0.00042	0.02058
WTI Crude Oil	0.03488	0.18676

Table 5: LSTM Results: MSE and RMSE

Overall, the values of these metrics are pretty low, however, by plotting the results, one can get more easily a sense of what is happening.

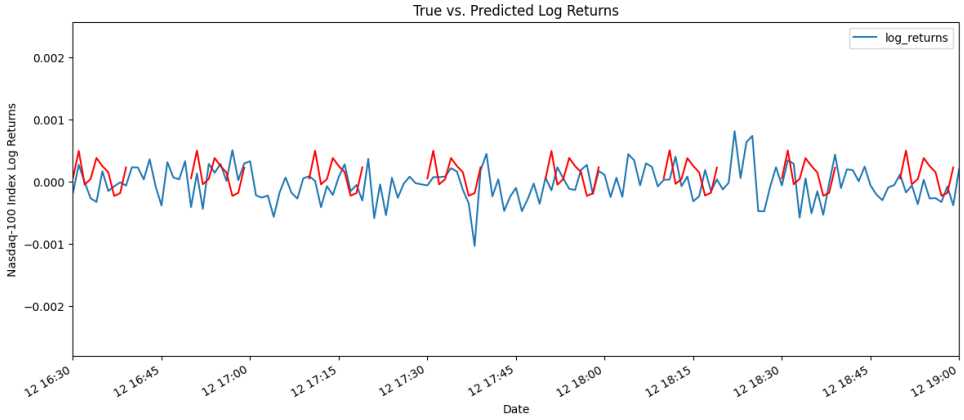

Figure 11: LSTM predictions on Nasdaq-100 data

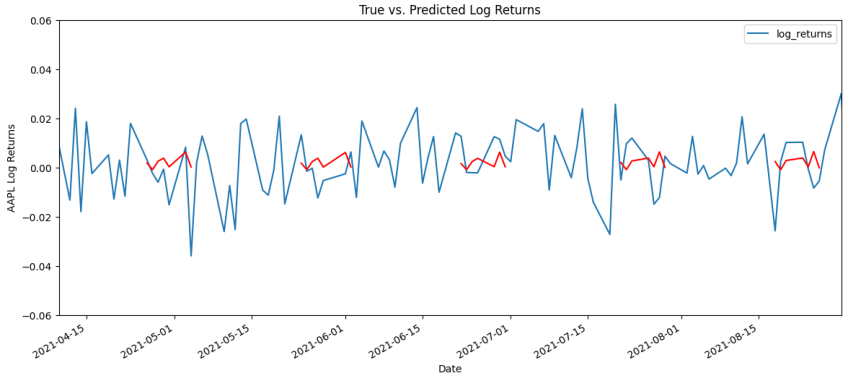

Figure 12: LSTM predictions on AAPL data

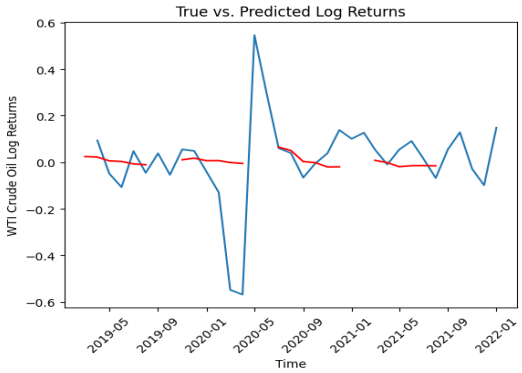

Figure 13: LSTM predictions on WTI data

Based on the plots above, it can be inferred that the LSTM model is capable of capturing the general trend and movement of the returns over time, however, it lacks precision in predicting the exact values. Perhaps, such a model could be better in classification tasks, for instance predicting whether the returns, therefore the prices, go up or down at the next time step.

7.6 FB Prophet

Facebook Prophet, or more simply Prophet, is an open-source library for forecasting tasks, developed by Facebook's data science team in 2017. Prophet is often considered an alternative to ARIMA models, however, to be more precise, it may be helpful when dealing with time-series data that have strong seasonal effects. In fact, among the advantages of this library, we can mention an automatic function that detects seasonality, but also its intuitive interface and flexibility, which allows to programmer to set the desired granularity of data.

The Prophet procedure is nothing but an additive regression model, belonging to the Generalized Additive Model (GAM) family, with four main components [56]:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

where

- g(t) is a piece-wise linear or logistic curve that models trend;
- s(t) models seasonality using the Fourier series and describes how data is affected by seasonal factors such as the time of the year;
- h(t) is the effect of holidays or events that impact business-related time series;
- ϵ_t which represents the error term of the model at each time step t.

To better understand how Prophet works, we have to think of it as a "data analyst-centric" model (Figure 14). The framework is double-sided: on one side model fitting is automated, pretending that the user has no previous statistical knowledge of the data, whereas on the other side, the framework allows the same user to input information based on his personal or industry knowledge [57].

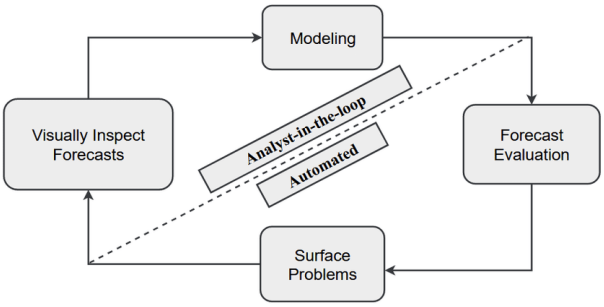

Figure 14: Analyst-in-the-Loop modeling. Source: [57].

Overall, Prophet has proved to be an interesting library in the time series panorama, though its performance is sometimes not as accurate as one may expect. All the results are displayed in the dedicated Appendix B paragraph titled *Prophet Results*.

Chapter 8

Conclusion

After delving into the complexities of exploratory analysis, random movements and patterns inherent in the data, it becomes straightforward that accurately predicting stock future behaviour is not an easy task. Overall, classical models are only able to make naive forecasts on our dataset, whereas the LSTM model can capture some dependencies across time steps, even though they are not so accurate. Based on the observed model performance, we can identify areas for potential future improvement. Specifically, to enhance the results of deep learning models like LSTM, two primary options can be explored: increasing the training data volume and incorporating more complex features. To achieve better outcomes, it is advisable to consider expanding the dataset or adjusting the rolling window length. Accurately predicting log returns often necessitates a substantial historical dataset capable of capturing diverse market conditions and trends. For example, when working with high-frequency data such as minute data, it is common to employ several years of observations to predict just a few minutes into the future. Along with expanding the training data, one could also explore incorporating additional features, or evaluating a portfolio of several stocks, thereby transforming the model into a multivariate one. However, it is important to note that this approach would introduce more complexity and increase the computational cost. Strictly related to model complexity, another option could be to use advanced architectures to unlock additional capabilities in capturing long-term dependencies, handling temporal dynamics, and achieving heightened prediction accuracy, as specified in Chapter 1, for example,

the Transformer and TFT model. Furthermore, considering the model's current lack of precision, it might be advantageous to exploit it for classification tasks. For example, instead of attempting to predict the exact behaviour of stock returns, the model could be trained to classify whether the return for the next period will be positive or negative. Indeed, the accuracy of the model could improve because of the inherent advantages of dealing with binary outcomes, which is easier compared to predicting the exact numerical value. Additionally, it is important to remember that financial data analysis is not limited to just numerical data. Another fashion to boost the model is to include textual data analysis. In recent years, we observe an increased interest in news, tweets, and articles that may significantly impact asset prices. Especially with the rise of social media and the ease of accessing information online, investors and traders are increasingly incorporating news and sentiment analysis into their decision-making processes. One of the most widespread approaches is sentiment analysis, which consists of quantifying the positive, negative, or neutral tone of news articles or tweets. Natural language processing algorithms are often employed to automatically categorize the sentiment of textual data. By examining the sentiment of news or tweets related to a particular asset or company, researchers can identify correlations between sentiment and subsequent price movements.

Lastly, considering the classical model, one may focus the analysis on volatility rather than on prices or returns, as in risk management applications. A commonly used approach is to employ Autoregressive Conditional Heteroscedasticity (ARCH) models [1] or its variants [58] [59] [60]. In particular ARCH(1) and GARCH(1,1) introduce non-linearity and usually provide a very good fit to log return data, higher-order models are rarely employed in experiments.

[1] The ARCH model was introduced by Robert Engle, who won the Nobel Prize in Economics in 2003. Some years later, Engle's PhD student Tim Bollerslev, developed the GARCH model.

Bibliography

[1] Eugene F. Fama. Random walks in stock market prices. *Financial Analysts Journal*, 21 no.5:55–59, 2010.

[2] Random walk hypothesis - Wikipedia, the free encyclopedia, 2023. [Online; accessed 28-May-2023].

[3] David N. Dreman and Michael A. Berry. Overreaction, underreaction, and the low-p/e effect. *Financial Analysts Journal*, 51(4):21–30, 1995.

[4] Andrew W. Lo and A. Craig MacKinlay. *A Non-Random Walk Down Wall Street*. Princeton University Press, 1999.

[5] Burton G. Malkiel. *A random Walk Down Wall Street*. WW Norton, New York, 12th edition, 2019.

[6] A. W. Lo and J. Hasanhodzic. *The Evolution of Technical Analysis: Financial Prediction from Babylonian Tablets to Bloomberg Terminals*. Bloomberg Press, 2010.

[7] Generic hype cycle of a technology. https://commons.wikimedia.org/wiki/File:Hype-Cycle-General.png/media/File:Hype-Cycle-General.png.

[8] Occam's razor - Wikipedia, the free encyclopedia, 2023. [Online; accessed 6-June-2023].

[9] Goodwin P. O'Connor M. Önkal D. Lawrence, M. Judgmental forecasting: A review of progress over the last 25 years. *International Journal of Forecasting*, 22 no.3:493–518, 2006.

[10] Jan G. De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22:443–473, 2006.

[11] Gwilym M. Jenkins George E. P. Box. Time series analysis: Forecasting and control. *Journal of Time Series Analysis*, 37:712, 2016.

[12] Eberhard Schöneburg. Stock price prediction using neural networks: A project report. *Neurocomputing*, 2:17–27, 1990.

[13] B. W. Wanjawa and L. Muchemi. Ann model to predict stock prices at stock exchange markets, 2014.

[14] Sheng Chen and Hongxiang He. Stock prediction using convolutional neural network. *IOP Conference Series: Materials Science and Engineering*, 435:012026, 11 2018.

[15] Hiransha M, E. A Gopalakrishnan, Vijay Menon, and Soman Kp. Nse stock market prediction using deep-learning models. *Procedia Computer Science*, 132:1351–1362, 01 2018.

[16] Sreelekshmy Selvin, R. Vinayakumar, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1643–1647, 2017.

[17] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE*, 12, 07 2017.

[18] Shereen Elsayed, Daniela Thyssens, Ahmed Rashed, Hadi Samer Jomaa, and Lars Schmidt-Thieme. Do we really need deep learning models for time series forecasting?, 2021.

[19] Eduardo Ramos-Pérez, Pablo J. Alonso-González, and José Javier Núñez-Velázquez. Multi-transformer: A new neural network-based architecture for forecasting s&ampp volatility. *Mathematics*, 9(15):1794, jul 2021.

[20] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *International Joint Conference on Artificial Intelligence*, 2015.

[21] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification, 2018.

[22] Tian Zhou, PeiSong Niu, Xue Wang, Liang Sun, and Rong Jin. Power time series forecasting by pretrained lm, 2023.

[23] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[25] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers, 2022.

[26] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2020.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[28] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[29] HuggingFace. Transformer xl. Accessed on Month Day, Year.

[30] Evaluating forecasting method using autoregressive integrated moving average (arima) approach for shariah compliant oil and gas sector in malaysia - scientific figure on researchgate. https://www.researchgate.net/figure/Forecasting-procedure-using-Box-Jenkins-approach_fig1_328630224 [accessed 20 May, 2023].

[31] Frederic B. Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic*, 9(2):49–50, 1944.

[32] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd edition, 2023.

[33] Anatomy of a multipolar neuron. https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Blausen_0657_MultipolarNeuron.png/500px-Blausen_0657_MultipolarNeuron.png.

[34] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[35] Illustration of a perceptron update. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/images/perceptron/PerceptronUpdate.png.

[36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 13–15 May 2010.

[37] Computational graph of a recurrent neural network with one hidden layer. https://www.researchgate.net/publication/342156022/figure/fig4/AS:941464641097770@1601474070772/Computational-graph-of-a-recurrent-neural-network-with-one-hidden-layer-29.png.

[38] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model, January 1988.

[39] Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Syst.*, 3, 1989.

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[42] Lstm cell architecture. https://thorirmar.com/post/insight_into_lstm/featured.png.

[43] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[44] Fatoumata Dama and Christine Sinoquet. Time series analysis and modeling to forecast: a survey, 2021.

[45] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[46] The transformer-model architecture. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[47] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[48] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[49] Nasdaq-100 vs. sp 500. https://www.nasdaq.com/articles/nasdaq-100-vs.-sp-500-2019-03-01-0.

[50] Junhui Guo. Oil price forecast using deep learning and arima. In *2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pages 241–247, 2019.

[51] Y. Jeevan Nagendra Kumar, Partapu Preetham, P. Kiran Varma, P. Rohith, and P. Dilip Kumar. Crude oil price prediction using deep learning. In *2020*

Second International Conference on Inventive Research in Computing Applications (ICIRCA), pages 118–123, 2020.

[52] Eric Kammers Colton Smith. Relationship between simple and log returns. https://quantoisseur.files.wordpress.com/2017/11/fig2.png.

[53] William R. Kinney Michael S. Rozeff. Capital market seasonality: The case of stock returns. *Journal of Financial Economics*, pages 379–402, 1976.

[54] Lee Bohl Randy Frederick. Trading near the bells, what to know about the market's two most volatile trading hours. https://www.schwab.com/learn/story/trading-near-bells [Online; accessed 30-May-2023].

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[56] Md Jamal Ahmed Shohan, Md Omar Faruque, and Simon Y. Foo. Forecasting of electric load using a hybrid lstm-neural prophet model. *Energies*, 15(6):1–0, 2022.

[57] Sean J. Taylor Benjamin Letham. Forecasting at scale. *The American Statistician*, pages 37–45, 2018.

[58] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

[59] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 50(Volume 31, Issue 3):987–1007, 1986.

[60] Embrechts P. Frey R. McNeil, A.J. *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, revised edition edition, 2015.

Appendix A

Exploratory analysis

Nasdaq-100

In data analysis, it is considered good practice to take a quick look at the data and perform some feature engineering to eliminate any duplicated, irrelevant, or missing data observations. The Nasdaq-100 dataset stands out as the largest among the datasets included in this work, with a total of 32,641 minute observations, ranging from March to the half of May 2023. However, after performing calculations for log returns and removing data points, the dataset is reduced to 13,988 observations. In this particular case, the data does not exhibit a clear trend, moreover, minute log returns can be observed in Figure 15.

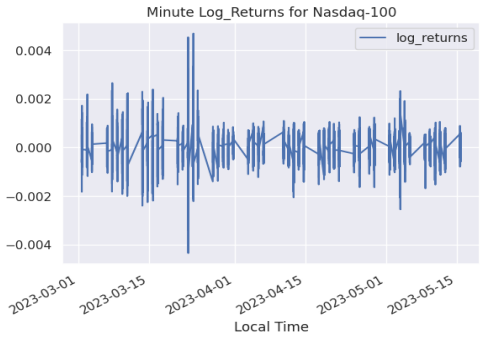

Figure 15: Nasdaq-100 logarithmic returns

Larger fluctuations are recorded at the beginning of April, followed by relatively lower returns for the rest of the month.

AAPL

The AAPL dataset contains data points spanning from April 2010 to December 2022, with a daily frequency. The exploratory analysis begins by examining the overall pattern of the data: more specifically, we can observe a consistent upward trend in the AAPL stock price (Figure 16). This latter undergoes fluctuations influenced by various factors including product launches, financial performance of the company, and market conditions. Additionally, notable price shocks have occurred as a result of events like stock splits and the Covid-19 outbreak, as evidenced in 2020.

As mentioned in previous chapters, an essential step is to clean up the dataset. Given that our focus is on log returns and there is a high degree of collinearity among variables, we will only keep one column.

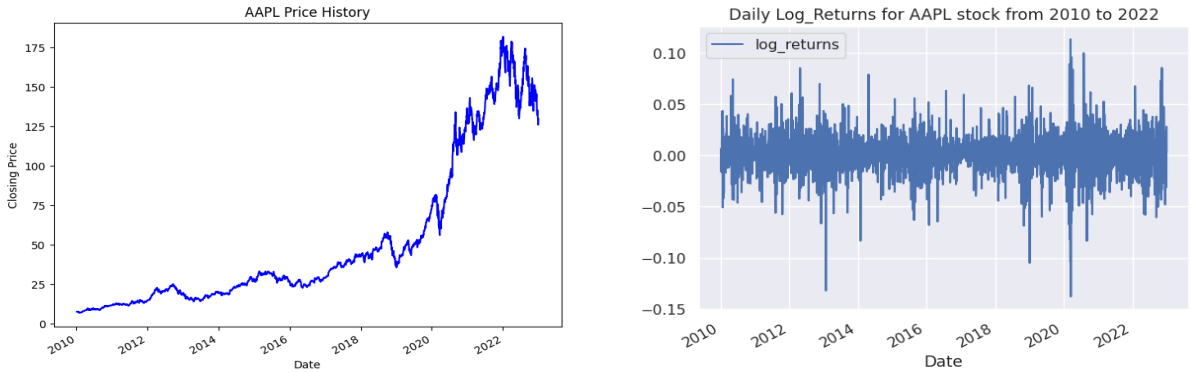

Figure 16: Price history and logarithmic returns for AAPL dataset

Before proceeding with the implementation of classical models, it is crucial to assess the stationarity of the time series. Dickey-Fuller test is performed on daily log returns data: the obtained p-value, which is close to zero, allows us to reject H_0, confirming that the data is stationary as expected. Two additional statistical tests were run in the code, namely Kwiatkowski–Phillips–Schmidt–Shin (KPSS) and Phillips–Perron Test (PP), supporting the previously stated outcome.

WTI Crude Oil

The WTI Oil dataset is made of 433 non-null observations ranging from January 1986 to January 2022, with a monthly frequency. The dataset includes a single column labelled

'MCOILWTICO,' which shows the price at which it closes. By examining Figure 17, we can spot the general trend and fluctuations. Undoubtedly, the major shock is the fall in price due to the economic crisis in 2008, even though, a significant decrease is observed also around 2014. For the scope of the analysis, monthly log returns are then computed, stored in a new dataset column and further plotted.

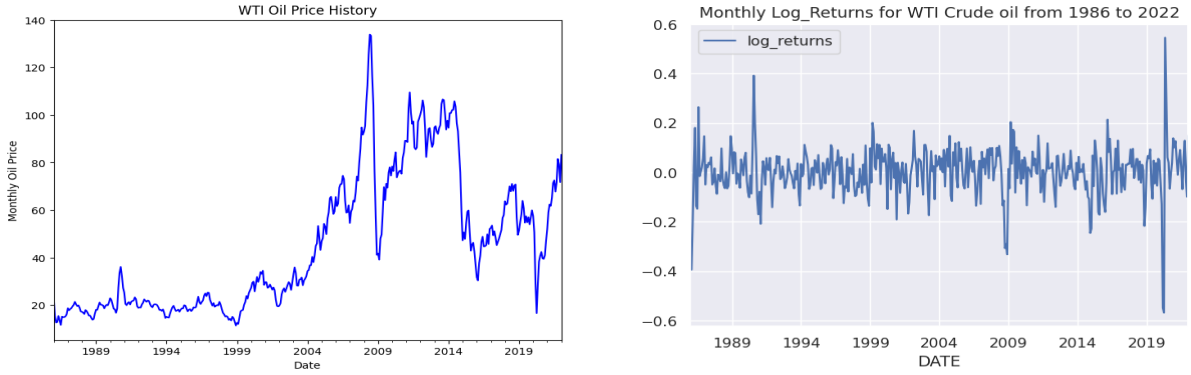

Figure 17: Price history and logarithmic returns for WTI Crude Oil dataset

As expected, the returns reflect the price movements, moreover from the additive decomposition plot, we are guaranteed that there is no general trend but a lot of variability. To check for stationarity, the Dickey-Fuller test is performed confirming the stationarity of data with a p-value of almost zero.

Python Libraries

For the scope of this analysis, several Python libraries have been used to speed up the pre-processing of data, but also the construction of the models, in particular: Pandas, Numpy, Keras, Sklearn, Matplotlib, Seaborn and Prophet.

Code

The full code is available at the following GitHub page. A README file details the structure of the repository, containing the datasets, divided by granularity, and all the printed Colab notebooks.

Appendix B

Prophet Results

This section serves as a continuation of paragraph 7.6 and contains supplementary results of the Prophet implementation. To successfully fit a Prophet model, the first step is to ensure that the dataset is structured according to the prescribed format, including specific column names. Specifically, the column containing the date should be renamed as 'ds', while the column containing the closing price (or log returns), serving as the target variable, should be labeled as 'y'. The dataset, containing only two variables, log returns and adjusted close price, is further divided into a training and testing set. In particular, the shape of the train and test sets are outlined in Table 6:

Data	Train Set	Test Set
AAPL	(2769, 2)	(503, 2)
WTI Crude Oil	(372, 2)	(61, 2)

Table 6: Train-Test sets shape

The model is simply instantiated by calling Prophet() and then fitted on the training data; the .make_future_dataframe command is used to make predictions, according to the desired frequency.

In Figures 18 and 19, Prophet forecasts can be observed for both types of data, together with its metrics in Table 7 and 8. The code provides also interactive plots, which offer a deeper understanding and additional insights.

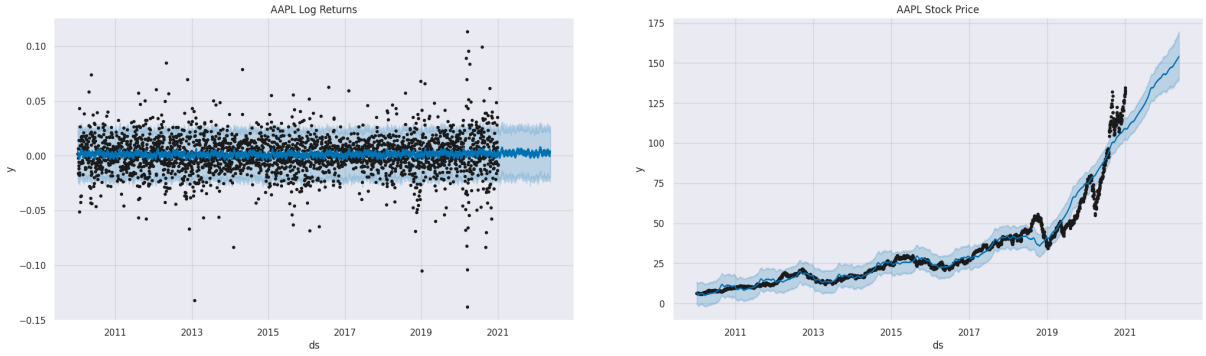

Figure 18: Prophet predictions on AAPL data.

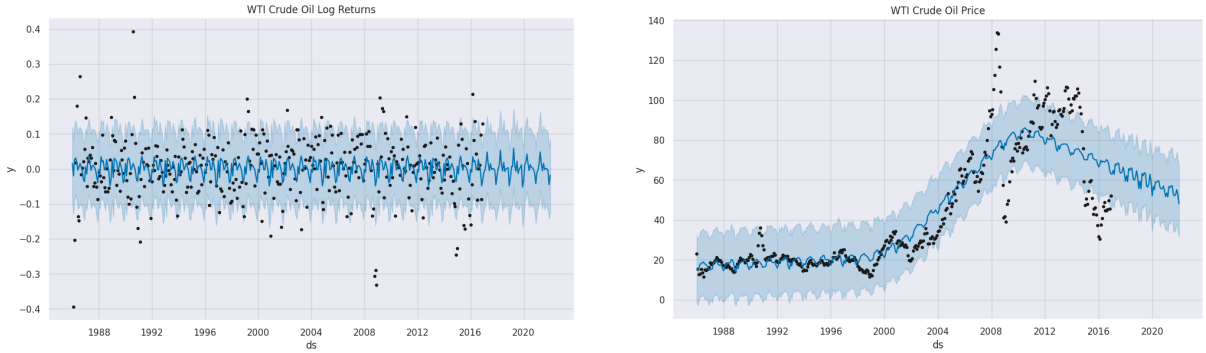

Figure 19: Prophet predictions on WTI Crude oil data.

Dataset	Frequency	RMSE
AAPL	Daily	20.701
WTI Crude Oil	Monthly	14.697

Table 7: Results of Prophet implementation using Adj Close price.

Dataset	Frequency	RMSE
AAPL	Daily	0.01984
WTI Crude Oil	Monthly	0.15059

Table 8: Results of Prophet implementation using Log returns.

Overall, the Prophet model does its best in the presence of seasonality in time series data. The performance metrics are quite good and Prophet can capture the macro trend either in log returns, which vary around zero and adjusted price, though it struggles to deal with unexpected changes due to the economic crisis, shocks or perhaps, the launch of new products in the case of Apple Inc. Moreover, for this analysis, fine-tuning of the model has not been carried out, since it is deemed to be too expensive compared

to the real contribution to the performance of the model. The prophet model has several parameters that one may consider tuning, for instance, changepoint_prior_scale, seasonality_prior_scale, and holidays_prior_scale, however, this requires the collection of extra information about the stock or index under consideration.

To conclude, when it comes to minute-wise data, as this is the case with Nasdaq-100, we decided not to use Prophet as it may not capture all the nuances and dynamics of intra-day trading. Prophet's strength lies in capturing longer-term patterns and seasonality. For achieving higher accuracy and precision in forecasting, it may be necessary to explore alternative models or techniques specifically designed for analyzing high-frequency data. Alternatively, one could aggregate minute-level data into higher frequencies, such as hourly or daily intervals. However, it is important to note that this aggregation process may lead to data shrinkage and potentially result in a loss of meaningful information. Careful consideration should be given to the trade-off between granularity and the potential loss of relevant details when choosing the appropriate data aggregation strategy.