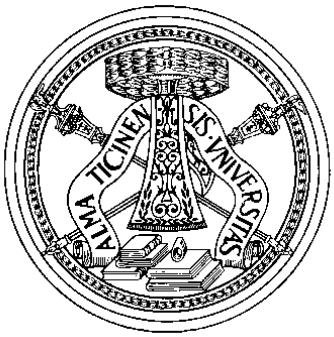

UNIVERSITY OF PAVIA

DEPARTMENT OF ECONOMICS AND MANAGEMENT

Master Program in

International Business and Entrepreneurship

Time series forecasting using statistical modeling and Deep Learning to predict Bitcoin market price

Supervisor: Prof. Paola Cerchiello

Student:

Andrea Valentini

Academic Year 2018-2019

"I always avoid prophesying beforehand, because it is a much better policy to prophesy after the event has already taken place."

SIR WINSTON CHURCHILL, British Prime Minister

UNIVERSITY OF PAVIA

Abstract

Faculty Name
DEPARTMENT OF ECONOMICS AND MANAGEMENT

Time series forecasting using Statistical Modeling and Deep Learning to predict Bitcoin market price

by Valentini Andrea

This research is concerned with predicting the price of Bitcoin using statistical modeling and Deep Learning neural network. The purpose of the research is double-fold. First, we evaluate prediction accuracy of classical statistical time series models and modern deep learning approach to financial time series. The former is taken as comparison baseline, while the research invests more effort to explore Artificial Neural Network model fitting to time series data. Secondly, the research introduces Blockchain network variables for studying the joint relationship with Bitcoin market price to evaluate the existence of some signal. Formation of Blockchain, a core technology of Bitcoin, distinguishes Bitcoin from other fiat currencies and is directly related to Bitcoins supply and demand. The current research study evaluates and characterizes the process of Bitcoin price by modeling and predicting Bitcoin prices using Blockchain information.

ITA: La ricerca riguarda lo studio di serie storiche per predire il futuro prezzo di mercato dei Bitcoin attraverso modelli statistici e di Deep learning. Lo scopo della ricerca riguarda da un lato l'accuratezza con il quale possibile predire il futuro andamento di mercato del prezzo dei Bitcoin usando dati storici, mentre dall'altro volto alla valutazione della relazione del prezzo dei Bitcoin con variabili relative alla Blockchain. Tali variabili vengono studiare attraverso analisi multivariate al fine migliorare la previsione dei Bitcoin.

Contents

List of Figures

Dedicated to my Family and to all the amazing persons I met during this Master's degree

Chapter 1

Introduction to Bitcoins and Blockchain technology

1.1 What are Bitcoins

The definition of Bitcoin dates back to the white paper (Bitcoin: a peer-to-peer electronic cash system) published in 2009 by a group of anonymous people under the name of Satoshi Natatomo [23]. Bitcoin is the first decentralized cryptocurrency. Since its inception has gained a growing attention from media, academics and finance industry due to its nature of combining encryption technology and monetary units. The crypto has achieved great success and represents a fundamental change in financial systems. Bitcoin is the words most valuable cryptocurrency. It is built on a decentralized, peer-to-peer network with the creation of money and transaction management carried out by the member of the network[20].

In the internet of money where those cryptocurrencies like Bitcoin lies, there isn't any central authority controlling the demand and supply for the currency. The transaction of Bitcoin relies on cryptographic proof verified by miners using a distributed open ledger known as Blockchain. This verification takes place in a trustless system with no intermediary required to pass the funds from sender to receiver.

In a few words, Bitcoin stands for an IT innovation based on advancement in peer-to-peer networks and cryptographic protocols which provides many advantages over traditional payment methods including high liquidity,lower transaction costs, and anonymity to name just a few. Time to break down the definition of Bitcoin in all its fundamental parts and define what practically Bitcoin is all about.

Bitcoin is a Cryptocurrency. Bitcoin belongs to the large family of cryptocurrencies, which are based on cryptographic methods of protection. The main promises of the digital currencies trends include a faster, more flexible and innovative payment system. A cryptocurrency uses the power of internet to process transactions. They can be transacted with any outside agents and the governance is decentralized mainly but not necessary due to open-source software. There is no legal entity responsible for the activities, and therefore, they fall outside traditional regulation [5]. In general terms, a cryptocurrency is a peer-to-peer version of electronic cash. It allows online payment to be sent directly from one party to another without going through a financial institution. Cryptocurrency is a subset of digital currency.

Bitcoin is a peer-to-peer payment system to facilitate transactions. In the world of Bitcoin, transactions are executed using a peer-to-peer cryptocurrency protocol without intermediaries. The technology literally disrupts the payment system as we know, because it costs almost nothing to transfer funds. Cryptocurrency technology allows us to include in the economy even those layer of the population without the possibility to access a financial institution (bank account). As internet has democratize information to everyone, so does Bitcoin for the economy.

Bitcoin is a decentralized electronic currency system. Decentralization is the major value achieved by cryptocurrencies as opposed to general fiat currencies being valued by central banks. There is no central authority which issues and regulates the currency. It is not only decentralized but also supposedly fully distributed. That means that every node or computer terminal is connected to each other. Every node can leave and rejoin the network at will and will later accept the longest proof of work known as the blockchain as the authoritative record [5].

Bitcoin uses Cryptography proof to built and maintain trust. The Bitcoin Network uses cryptography to validate transactions during the payment processing and creates transaction blocks [7]. It relies on two cryptographic schemes:

- Digital signatures
- Cryptographic hash function

While digital signatures allow for the exchange of payment instructions between parties involved, the second is used to maintain the discipline when recording transactions to the public ledger (known as Blockchain). Digital signature offers authentication, non-repudiation and Integrity benefits between a sender and a recipient. It includes a public public key cryptography, where a pair of keys (open and private) are generated. All the members of the network can verify that the transaction came from the owner of the

public key, by taking the message, the signature and by running a test algorithm [7]. On the other hand, an hash function is a deterministic function which takes as input a string of arbitrary length (the message m), and returns the string with predetermined length (the hash h). Exchange of Bitcoin in the bitcoin network take place through bitcoin-addresses. The ability to send payments to other bitcoin addresses is controlled by a digital signature, which include a public key and a private key. In particular, every bitcoin address is indexed by a unique public ID, which is an alphanumeric identifier, which corresponds to the public key. The private key controls the bitcoins stored at that address. At any point in time, every bitcoin address is associated with a bitcoin balance, which is public information. Each existing or proposed (broadcasted) transaction can be checked for compliance with the past transaction history, i.e. it is possible to verify that the transferred bitcoin do exist at the corresponding bitcoin address [7].

The transaction processing in the network is based on mechanisms which ensure:

- Distributed verification of each transaction among several network members
- discrete transaction record with respect to time
- linearly ordered transaction
- participant in the network are rewarded for the recording of the transaction in a bitcoin network
- multiple nodes cross-check each transaction

Before a transaction get executed, the Bitcoin protocol verify first the authenticity of the sender of the message. The digital signature scheme ensures that only the owner of the private key for that address may sign a message; secondly, to check whether there are sufficient funds in the address to ensure the completion of the transaction.

The last step of the process consist of updating the blockchain. Validation nodes in the Bitcoin network begin to compete for the opportunity to record a transaction in the Blockchain in the block of the transaction. The block is used to define a complex computing task. The node that first solves this task proceeds to record the transactions on the Blockchain and collects a reward. The task which the competing nodes try to solve is based on the hash function. The first of the competing nodes which will find the right random code, transfers this information to the other participants in the network, and the Blockchain is updated. The implementation of this scheme is the so-called Hashcash - a proof that the system is operating properly (proof-of-work). The nodes that perform the process of the proof-of-work in the Bitcoin network are called miners [7]. To solve these decoding task, miners need computational power, which is measured

by the Hash rate. The Hash rate is the speed at which a computer can complete an operation in the Bitcoin code, while the mining difficulty refers to the level of complexity of the task.

1.2 Blockchain technology

Unlike existing fiat currencies with central banks, Bitcoin aims to achieve complete decentralization. Participants in the Bitcoin market build trust relationships through the formation of Blockchain based on cryptography techniques using hash functions. Blockchain is a broad topics which have led to diverse research interests not only in the field of economics but also in cryptography and machine learning.

Blockchain is a distributed ledger stored locally on the computer hard drive of every user running a full version of the Bitcoin software. The ledger records the history of every transaction sent and confirmed on the Bitcoin Network, including information included as a part of those transactions [30].

Decentralization can be specified by the following ultimate objective:

- Who will maintain and manage the transaction ledge?
- Who will have the right to validate transactions?
- Who will create new Bitcoins?

Blockchain is the enabling technology that can simultaneously achieve these goals.

First, it enable the transfer of money digitally between willing participants without the need of a trusted party, secondly, it establishes the order of transaction to avoid double spending. In its most simple form blockchain is a decentralized ledger. The implications of blockchain however, are far greater than the simplicity its name implies.

Blockchain facilitates the digital transference of value itself. Quoting D A Tapscott:

"The blockchain is an incorruptible digital ledger of economic transactions that can be programmed to record not just financial transactions but virtually everything of value

The blockchain is a chain of data blocks. Each block can be thought of as a page in a ledger. The main difference with a database we currently know lies on the fact that the blockchain is decentralised and updated by the member of the network.

Advanced application of Bitcoin and Blockchain technology have the potential to impact any industry or product line that relies on the storage and verification of information and value. Bitcoin and Blockchain technologys programmable aspects may also facilitate the development of autonomous governance systems, contracts and legal constructs (e.g., smart contract) or the ability of interconnected devices to interact with and even pay each other in the Internet of Things. [30]

1.3 Benefits and Weaknesses of Bitcoin and the Blockchain

Both Bitcoin and Blockchain are two disruptive innovation in the financial industry. At the same time, they bring benefits and risks to the table. This section outlines some of the most well-known benefits and risks.[30]

- Transparency: All Bitcoin Network transactions are cleared in the Blockchain. A complete, auditable and immutable record of all activity exists.
- No risk of chargeback fraud
- Low or no transaction costs: Transaction on the Bitcoin network can be sent with no transaction fees.
- Nearly instantaneous transactions: Bitcoin Network transactions register nearly instantaneously
- Network security: The Bitcoin Network itself is highly secure due cryptography and decentralized Blockchain protocols. Furthermore, there is no single, centralized point of failure
- Protection of financial information: anonymity of the sender and receiver are guaranteed

Relative to these advantages, there are significant weaknesses relating to Bitcoin as a payment system and as an asset.

- Difficult to use: Most software to control, custody, or transact in bitcoin is complex
- Difficult to access: Although the Bitcoin Network is open access and liquid markets, few of the exchanges and services that allow the purchase of bitcoins are regulated
- Lacks protections against mistakes: Unlike traditional electronic payments, Bitcoin transactions cannot be reversed

- Limited retail and institutional adoption

Chapter 2

Statistical Background on Time series analysis and Literature review

2.1 Time series data structure

The recent few years have witnessed the widespread application of statistics and machine learning to derive actionable insights and business value out of data in almost all industrial sectors. Hence, it is becoming imperative for business analysts and data scientist to be able to tackle different types of datasets.[10]

In the field of Data Science many data sets used to conduct machine learning research come in a variety of type. This empirical research deals with time series data, which are composed of a sequence or a series of quantitative observations (data point) about a system or a process made at successive point in time. Most commonly, data are collected at equally spaced points in time.

Whereas some statistical methods can be applied with little or no modification to many different kinds of data sets, the special features of time series data should be exploited.

Data comes in many form. Non-experimental data are not accumulated through controlled experiments about individuals, firms, natural phenomenons, processes or segments of the economy (Non-experimental data are sometimes called observational data, or retrospective data, to emphasize the fact that the researcher is a passive collector of the data). Experimental data are often collected in laboratory environments in the natural sciences, but they are much more difficult to obtain in the social sciences[37]

In addition to the origin of the data source, data may come in under several district category. The following discussion will help the reader understand how time series and non-time series intrinsically differ and hence, it will come in handy to choose the right approach to formulate and solve forecasting problems.

Business analysts and data scientists come across many different types of data in their analytics projects. Most data commonly found in academic and industrial projects can be broadly classified into the following categories:

- Cross-sectional data
- Time series data
- Panel data

Cross-sectional data are composed of sample observations about individuals, households, firms, cities, states, countries, or a variety of other units, collected at one point in time. Those data are theoretically collected by random sampling from the underlying population.[1] As a matter of fact, the order in which observations are listed in the dataset doesn't matter. This fact is a key feature of cross-sectional data. Thus, the collection of random variables are assumed to be *i.i.d.* Although different variables are collected over different periods, time dimension is not taken into account.

Next, we are going to discuss the main data structure studied in the research, that is time series analysis. A time series data set consists of observations collected on a variable or several variables over time. Examples of time series data include stock prices, money supply, consumer price index, gross domestic product, annual homicide rates, and automobile sales figures. Unlikely cross-sectional datasets, the time dimension conveys potential information for business decision maker and policy analyst. Depending on the nature of the variable and its applications, observations can be recorder at different **data frequency**. Sales Volume might be published monthly, or quarterly, stock price index might be recorded using second granularity over days. Supply chain operational data might be inserted in the database one day after the event took place. In such situation, it's important to keep a low latency between the time the event happened and the actual availability of time series data since timeliness of decision could be paramount. On the other extreme, there are several physical processes which generate time series data at fraction of a second.

[1] The goal of mathematical statistics involves learning something about the population of interest from a sample. Under the random sampling assumption, each $\binom{N}{n}$ sample has the same probability of being selected. If $Y_1, Y_2, , Y_n$ are independent random variables with a common probability density function $f(y; u)$, then $\{Y_1, , Y_n\}$ is said to be a random sample from $f(y; \theta)$[8]

One of the key features of time series analysis that make them more difficult to analyze is that the assumption of identically distributed observation of the random variable does not hold. Most time series variables are not independent, but related to historical values. This features addo hurdles and opportunities at the same time. The former due to the fact that many statistical inference techniques, statistical estimation properties and theorems such as independent error with $mean = 0$ and $VAR = \theta$ need to be satisfied. The latter because under some circumstances the analyst can exploit the dependent nature of the data to model the stochastic process that has generated the time series analysis, hence make accurate future prediction out of the training sample.

Finally, we conclude this data structure review defining panel data. A panel data (or longitudinal data) set consists of a time series for each cross-sectional member in the data set. By its nature, it combines features of cross-sectional data (snapshot of variables) and time series data (snapshot taken at different time frame). The same observational unit is observed over a period of time. Clearly, medical research has massive applications margins. For example patients blood value or city's country states pollution can be recorded over a period of time. The use of more than one observation can facilitate causal inference in situations where inferring causality would be very difficult if only a single cross section were available. A second advantage of panel data is that they often allow us to study the importance of lags in behavior or the result of decision making.[37]

2.2 Characteristics of Time Series Data

When a variable is measured sequentially in time at a fixed interval, known as *sampling interval*, the resulting data forms a time series.[21] In order to provide a statistical setting to describe the characteristics of data that seemingly fluctuate in a random fashion over time, we assume a time series can be defined as a collection of random variables indexed according to the order they are obtained in time.[35]. A time series can be expressed as sequence of random variable $x_1, x_2, ..., x_N$, that can generally be written as $\{x_t\}$, referred as a stochastic process. The observed values of a stochastic process are referred as realization of the stochastic process. Observations that have been collected over fixed sampling intervals form a historical time series. From a statistical and stochastic perspective, chapter 4 will treat time series as realization of random variables. When a sequence of random variable is collected at discrete point in time, they referred to *discrete-stochastic process*. Plotting the data should be the first step in any time series analysis tasks. Vision inspection can spot outliers, erroneous values, positive or negative trend, recurrent seasonal fluctuation and main patterns. A *general trend* is a variable's systematic change in a time series that does not appear to be periodic. In

many applications, trend is said to be stochastic as in case of a random walk with drift. Trend extrapolation might account for a unrealistic forecasting estimation if scenarios or sudden future events changes some hypothesis of the phenomenon analysed. A repeating pattern within a fixed period is known as *seasonal fluctuation*. Obviously, this became visible if data have not been aggregated at lower frequency which obfuscate any seasonal variation factors. Sometimes we may claim that are present *cycles behaviour* in a time series that do not correspond to some fixed natural period. By definition, unexpected variation that follow a stochastic process cannot be framed into a linear regression model. The left unexplained variation is also called irreducible error component which doesn't exhibit systematic dependency with the time index. Another important feature of most time series is that observations close together in time tend to be correlated (serially dependent). Much of the methodology in a time series analysis is aimed at explaining this correlation and the main features in the data using appropriate statistical models and descriptive methods. Auto correlation, partial auto correlation and cross-correlations help to find the ideal statistical stochastic model that has generated the time series. In this research, the study of time series analysis will focus on statistical and deep learning modelling fitting for making prediction $\hat{x}_{t+k|t}$ at time t for a future value at time $t+k$ as well as run statistical inference tests. Also, once a good model will be found and fitted to the data, we ideally would disposes of a summary view over the distribution and main characteristics of the time series.

2.2.1 Additive decomposition models

The main objective of time series analysis is to extrapolate the main characteristics of the series to define the related causes that has produced the observed state or to forecast the future state of the process in terms of observable characteristics. Business decision and policy makers might eventually leverage this knowledge to take optimally future decision. Somethings, plotting the series versus the time index reveals a clear structure which perfectly suits the framework of many additive decomposition models. Those simple models leverage the linear and additive assumption to break down the internal structure of a series into its constituent components . Consider for a moment sales figures, unemployment rate, tourism country inflow data or supply chain transportation volume. Even before looking at the raw data, we expect to see some deterministic (not due to randomness) components such as monthly volume increase due to Christmas holidays in supply chain, linear trends due to improved consumption propensity of a nation in some mass item production series or global temperature warming as a consequence of an increase industrialization and consumption of fossil fuel. Under such circumstances

and holding those assumption we would model the series as:

$$x_t = m_t + s_t + z_t \tag{2.1}$$

Or if both trend and seasonality tend to increase or decrease as a function of time, a multiplicative model may be more appropriate:

$$x_t = m_t * s_t + z_t \tag{2.2}$$

where $t = 1, 2, 3...N$, x_t is the observed series.

Trend m_t can be estimated using smoothing techniques which take an average of successive observations centered over a predefined period. In this example we compute the yearly centered moving average at time t

$$\hat{w_t} = \frac{1}{3}\left(w_{t-1} + w_t + w_{t+1}\right) \tag{2.3}$$

where t is the frequency rolling window we decide to compute the mean. If we want to obtained an unbiased estimate of the trend, it is fundamental to pay attention at the rolling window parameter of the formula. A general trend is commonly modeled by setting up the time series as a regression against time. If the series follow a deterministic linear trend over time, detracting the estimated linear regression might result into a detrended time series.

Similarly, seasonal variation s_t is estimated averaging monthly, weekly or at any sampling frequency that have a repetitive pattern. By using the seasonal frequency for the coefficients in the moving average, the procedure generalises for any seasonal frequency. It is common to present economic indicators, such as unemployment percentages, as seasonally adjusted series. This highlights any trend that might otherwise be masked by seasonal variation. Practical techniques to determine seasonality are

- run sequence plot
- seasonal sub series plot
- multiple box-plots

Aggregate the original series over predefined periodicity and use box-plots to estimate whether the distributions is statistically different over several months, weeks, quarters produces seasonal factors which are useful to compute the contribution of each frequency period to level of the time series. Also, their visual outputs have the power to shed lights

on the frequency of the seasonality. Assuming that long run trend has been removed by a trend line, the seasonality model can be expressed as $xt = s_t + y_t$, where the seasonal variation with known periodicity is α. Many realistic models for generating time series assume an underlying signal with some consistent periodic variation, contaminated by adding a random noise. An example of such models is the *harmonic regression model* which attempt to fit the sum of multiple *sin* and *cos* waves.

$$x_t = A\cos(2\pi\upsilon + \phi) + w_t \tag{2.4}$$

If the time series contains any deterministic components, subtracting them would reveals its residual errors z_t that are an estimation of the random stochastic process that has generated the series. In general, a sequence of correlated random variables with mean 0. This type of error is due to lack of information about explanatory variables that can model these variations or due to presence of a random noise. It must be highlighted that it doesn't consist of a realization of the random process, but rather an estimate of that realisation. However, we treat it as a realisation of the random process. State space models features similar models which uses Kalman filters and smoothers to create parametric mathematical decomposition function of future value from the historical ones.

2.2.2 Time series serial correlation and stationarity

As we mentioned earlier, the random component left after seasonal adjusting and removing trends from the series doesn't necessarily result in independent random variables. In many cases, consecutive random variables will be correlated. The correlation structure of a time series model is defined by the correlation function, and we estimate this from the observed time series. As reference for future chapters, in this section, the research introduce theoretical measures used to describe how time series behave.

The expected value, commonly known as *Expectation, E* of a function of a variable is its *mean* value, μ in a population.

$$u_{xt} = E(x_t) = \int_{-\infty}^{+\infty} x f_t(x) dx, \tag{2.5}$$

or in the discrete sampling interval case:

$$\hat{x} = \sum \frac{x_i}{n} \tag{2.6}$$

provided it exists, it defines the first moment of the probability distribution, while the second moment is called the variance σ^2 ,

$$\sigma^2 = \int_{-\infty}^{+\infty} (y - \mu_y)^2 f(y) dy \tag{2.7}$$

or in the discrete sampling interval case:

$$V(x) = E(X - EX)^2 \tag{2.8}$$

$$V(x) = E[(x - \mu)^2] \tag{2.9}$$

defined as the mean of the squared deviation about μ. The standard deviation is σ the square root of the variance.

If there are two variables (x, y) the variance may be generalised to the *Covariance* $\gamma(x, y)$. Covariance is defined by

$$\gamma(x, y) = E[(x - \mu_x)(y - \mu_y)] \tag{2.10}$$

The Covariance is a measure of linear association between two variables. It is important to emphasize that a linear association between variables does not imply causality.[37]

Since the covariance isn't in the same unit of measure of the variable, we use correlation measure to define linear association between pairs of variables (x, y) and is obtained by standardising the covariance by dividing it by the product of the standard deviations of the variables. Correlation takes a value between 1 and +1, with a value of 0 indicating no linear association.

$$\rho(x, y) = \frac{E[(x - \mu_x)(y - \mu_y)]}{\sigma_x \sigma_y} \tag{2.11}$$

A complete description of a time series, observed as a collection of random variables at arbitrary integer points $t_1, t_2, ..., t_n$, for any positive integer n, is provided by the joint distribution function, evaluated as the probability that the values of the series are jointly less than the n constants, $c_1, c_2, ..., c_n$:

$$F(c_1, c_2, ..., c_n) = P(x_{t1} \leq c_1, x_{t2} \leq c_2, ..., x_{tn} \leq c_n). \tag{2.12}$$

Although the joint distribution function describes the data completely, it is an unwieldy tool for displaying and analyzing time series data. The distribution function (reference the joint distribution equation) must be evaluated as a function of n arguments, so any plotting of the corresponding multivariate density functions is virtually impossible.[35].

The lack of independence between two adjacent values x_s and x_t can be assessed numerically, using the notions of autocovariance and autocorrelation. Assuming the variance of x_t is finite, we have the following definition for the ***Autocovariace***:

$$\gamma(s,t) = E[(y_t - \mu)(y_{t+} - \mu)] \tag{2.13}$$

The Autocovariance measures the linear dependence between two points on the same series observed at different times. Very smooth series exhibit autocovariance functions that stay large even when the t and s are far apart, whereas choppy series tend to have autocovariance functions that are nearly zero for large separations. The Autocovariance (reference the autocovariance formula) is the average cross-product relative to the joint distribution $F(x_s; x_t)$.

Similarly to the correlation between two random variables, autocorrelation function ACF measures the linear predictability of the series at time t, say x_k, using only the value at time k. It ranges between $-1 \leq \gamma(s,t) \leq 1$

$$\rho(y) = \frac{Cov(y_t, y_{t+}k)]}{\sigma_t} = \frac{\gamma_k}{\gamma_0} \tag{2.14}$$

Ultimately, two random variables might be related at different time lags. Thus, we defined the **cross-correlation**:

$$\hat{\rho}_y(k) = \frac{Cov_{xy}(h)}{\sqrt{\gamma_x(0)\gamma_y(0)}} \tag{2.15}$$

In the definition above, the autocorrelation and cross-correlation functions may change as one moves along the series because the values depend on both s and t, the location of the points in time. Ideally, we would like the autocorrelation to depends on the separation of x_s and x_t, say, $h = |s - t|$ and not on where the points are located in time. As long as the points are separated by h units, the location of the two points does not matter. This notion, called weak stationarity, when the mean is constant, is fundamental in allowing us to analyze sample time series data when only a single series is available.

So far, we have been defining some descriptive statistics to define the moments and correlation function of a random variable. Although we have not made any special assumptions about the behavior of the time series, many of the preceding examples have hinted that a sort of regularity may exist over time in the behavior of a time series. We introduce the notion of regularity using a concept called **stationarity**.

A **strictly stationary** time series is one for which the probabilistic behavior of every collection of values $\{x_{t1}, x_{t2}, ..., x_{tK}\}$ is identical to that of the time shifted set

$\{x_{t1+h}, x_{t2+h}, ..., x_{tK+h}\}$. That is,

$$P\{x_{t1} \leq c_1, ..., x_{tk} \leq c_k\} = P\{x_{t1+h} \leq c_1, ..., x_{tk+h} \leq c_k\} \tag{2.16}$$

for all $k = 1, 2, ...$, all time points $t_1, t_2, ...t_k$, all numbers $c_1, c_2, ..., c_k$ and all time shift $h = 0, 1, 2...$

If a time series is strictly stationary, then all of the multivariate distribution functions for subsets of variables must agree with their counterparts in the shifted set for all values of the shift parameter h. If the variance function of the process exists, the autocovariance function of the series x_t satisfies:

$$\gamma(s, t) = \gamma(s + h, t + h) \tag{2.17}$$

for all s and t and h. Thus, that means the autocovariance function of the process depends only on the time difference between s and t, and not on the actual times. In some situation, we relax the strong assumption of strict stationarity. Rather than imposing conditions on all possible distributions of a time series, we will impose conditions only on the first two moments of the series.

A **weakly stationary** time series, x_t, is a finite variance process such that

- the mean value function, μ_t is constant and does not depend on time t
- the autocovariance function $\gamma(s, t)$ depends on s and t only through their difference $|s - t|$

Because the mean function $E(x_t) = \mu_t$, of a stationary time series is independent of time t, we say the time series is stationary on the mean

$$\mu_t = \mu \tag{2.18}$$

On the same extent, we assume the model is stationary in the variance when it is contant over all time period:

$$\sigma^2(t) = E[(x_t - \mu)^2] \tag{2.19}$$

In a time series analysis, sequential observations may be correlated. If the correlation is positive, Var(x) will tend to underestimate the population variance in a short series because successive observations tend to be relatively similar.

Also, we study the *second-order property* which include the serial correlation. Let $s = t + h$, where h represents the time shift or lag. Then

$$\gamma(t+h,t) = cov(x_{t+h}, x_t) = cov(x_h, x_0) = \gamma(h,0) \tag{2.20}$$

because the time difference between times $t+h$ and t is the same as the time difference between times h and 0. Thus, the autocovariance (acvf) function of a stationary time series does not depend on the time argument t. The number of time steps between the variables is known as the lag.

The autocovariance function of a stationary time series is defined as:

$$\gamma(h) = cov(x_{t+h}, x_t) = E[(x_{t+h} - \mu)(x_t - \mu)] \tag{2.21}$$

While the autocorrelation (ACF) of stationary time series is defined as:

$$\gamma(h) = \frac{\gamma(t+h,t)}{\sqrt{\gamma(t+h,t+h)\gamma(t,t)}} = \frac{\gamma(h)}{\gamma(0)} \tag{2.22}$$

The concept of weak stationarity forms the basis for much of the analysis performed with time series. The fundamental properties of the mean and autocovariance functions and are satisfied by many theoretical models that appear to generate plausible sample realizations. The correlogram plot is the main plot to evaluate if a time series is a realization of a stationary stochastic process. The main use of the correlogram is to detect autocorrelation in the time series after we have removed an estimate of the trend and seasonal variation. Thus, if a model has accounted for all the serial correlation in the data, the residual series would be serially uncorrelated, so that a correlogram of the residual series would exhibit no obvious patterns.[21]

2.2.3 Basic Stochastic models

An ideal state would be for the time series analyst to develop a mathematical models that could provide a plausible description of the sample data. Temporal ordering of observation adds further issues on the statistical modelling fitting. Instead of independent and identically distributed observation taken from the same population using random sampling, time series observation collected ad adjacent point in time may suffer from serial correlation. OLS estimators are by definition random variables since different sampling distributions from the same population yield different values of the independent and dependent variables. In the following paragraphs, the research tries to give arguments to the following points:

- To deal with randomness in time series data
- To use statistical setting to explain the random fluctuation of the series?

Since the outcomes of these variables are not foreknown, they should clearly be viewed as random variables. We assume a time series can be defined as a collection of random variables indexed according to the order they are obtained in time. A sequence of random variables indexed by time is called a stochastic[2] process for a time series process. The previous definition tell us that a time series is just a realization of a stochastic process, one of many possible outcomes. Using simulation to run a new time series from the same probability distribution would result in a different dataset. Here we are going to discuss the distribution and mathematical statistical properties of some of the most popular stochastic processes that might have generated the Time series. As a general rule of thump, if our model encapsulates most of the deterministic features of the time series, our residual error series should appear to be a realisation of independent random variables from some probability distribution.[21] However, we often find that there is some structure in the residual error series, such as consecutive errors being positively correlated, which we can use to improve our forecasts and make our simulations more realistic. Since we judge a model to be a good fit if its residual error series appears to be a realisation of independent random variables, it seems natural to build models up from a model of independent random variation, known as discrete white noise.

2.2.4 White Noise models

A time series $\{w_t : t = 1, 2, ..., n\}$ is *discrete white noise* if the variables $w_1, w_2, ...w_n$ are independent and identically distributed with a mean of zero. This implies that the variables all have the same variance σ^2 and $Cor(w_i, w_j) = 0$ for all $i \neq j$. If, in addition, the variables also follow a normal distribution ($i.e., w_t \sim N(0, \sigma^2)$) the series is called Gaussian white noise. Second order properties of a white noise series:

$$\mu_w = 0 \tag{2.23}$$

$$\gamma(h) = cov(x_{t+h}, x_t) \begin{cases} \sigma^2 if k = 0 \\ 0 if k \neq 0 \end{cases}$$

A white noise series usually arises as a residual series after fitting an appropriate time series model. If the stochastic behavior of all time series could be explained in terms of the white noise model, classical statistical methods would suffice.

[2]Stochastic is a synonym for random

2.2.5 Random Walk models

Let $\{x_t\}$ be a time series. Then $\{x_t\}$ is a random walk if:

$$x_t = x_{t-1} + w_t \tag{2.24}$$

where $\{w_t\}$ is a white noise series. Random walk second order properties:

$$\begin{cases} \mu_x & 0 \\ \gamma(h) = cov(x_{t+h}, x_t) = t\sigma^2 \end{cases}$$

The covariance is a function of time, so the process is non-stationary. In particular, the variance is $t\sigma$ and so it increases without limit as t increases. It follows that a random walk is only suitable for short term predictions. For large t with k considerably less than t, rho_k is nearly 1. Hence, the correlogram for a random walk is characterised by positive autocorrelation that decay very slowly down from unity. Definitely, stationarity is the main properties of time series data. Thus, the difference operation at different lag can make a series stationary. Differencing the data at time t with it Thevious observation at time $t-1$ can remove possible deterministic trend. Also differentiation turn out to be useful to remove seasonal variation at frequency time t. *Backward shift* operator B is defined

T by

$$B^n x_t = x_{t-1} \tag{2.25}$$

hIt is sometimes called the 'lag operator'. By repeatedly applying B, it follows that:

$$B^n x_t = x_{tn} \tag{2.26}$$

$$x_t = Bx_t + w_t \rightarrow (1-B)x_t = w_t \rightarrow x_t = (1-B)^{-1} w_t \tag{2.27}$$

e first-order differences of a random walk are a white noise series, so the correlogram of the series of differences can be used to assess whether a given series is reasonably modelled as a random walk.

If the time series shows some stochastic trend as many stock prices within the volatility of the financial market, the random walk model can be adapted to allow for this by including a *drift* parameter δ:

$$x_t = x_{t-1} + \delta + w_t \tag{2.28}$$

for $t = 1, 2, ...$, with initial condition $x_0 = 0$, and where w_t is white noise. The constant δ is called drift, and when $\delta = 0$ we return to the classical random walk model. The term random walk comes from the fact that, when $\delta = 0$, the value of the time series at time t is the value of the series at the time $t - 1$ plus a completely random movement determined by w_t. As a result, we might rewrite the formula as a cumulative sum of white noise variables. That is,

$$x_t = \delta t + \sum_{j=1}^{t} w_j \tag{2.29}$$

2.2.6 Autoregressive models

The series $\{x_t\}$ is an autoregressive process of order p, abbreviated to AR(p), if

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + ... + \alpha_p x_{t-p} + w_t \tag{2.30}$$

where $\{w_t\}$ is white noise and $\alpha_p \neq 0$ for an order p process.

The previous equation can be also expressed as a polynomial of order p in terms of the backward shift operator.

$$\theta_p(B)x_t = (1 - \alpha_1 B - \alpha_2 B^2 ... - \alpha_p B^p)x_t = w_t \tag{2.31}$$

The random walk is the special AR(1) with $\delta_1 = 1$. The model is a regression of x_t on past terms from the same series, hence the use of the term "autoregressive".

Second order properties of an AR(p)

$$\begin{cases} \mu_x & 0 \\ \gamma_k & \alpha^k \sigma^2/(1 - \alpha^2) \end{cases}$$

2.2.7 Moving average models

Moving average models belongs to the family of stationary models. Those models are suitable for residual series that contain no obvious trend or seasonal cycles. A time series model $\{x_t\}$ is strictly stationary if the join statistical distribution of $x_{t1}, ..., x_{tn}$ is the same as the join distribution of $x_{t1+m}, ..., x_{tn+m}$ for all $t_1, ..., t_n$ and m, so that the distribution is unchanged after an arbitrary time shift. [21] If a series is not strictly stationary but the mean and variance are constant in time and the autocovariance only depends on the lag, then the series is called second-order stationary. A moving average

(MA) process of order q is a linear combination of the current white noise term and the q most recent past white noise terms:

$$x_t = w_t + \beta_1 w_{t-1} + ... + \beta_q w_{t-q} \tag{2.32}$$

where $\{w_t\}$ is white noise with $\mu = 0$ and variance σ^2. Because MA processes consist of a finite sum of stationary white noise terms, they are stationary and hence have a time-invariant mean and autocovariance. The mean for $\{x_t\}$ is just zero because it is a sum of terms that all have a mean of zero. The variance is $\sigma^2(1 + \beta_1^2 + ... + \beta_q^2$ because each of the white noise terms has the same variance and the terms are mutually independent.

2.3 Literate review

Cryptocurrency price dynamics is inevitably an hot industry to apply machine learning and forecasting algorithms. Although efficient market theory argues that there are not "free lunches" in the financial market, the young history and high volatility of the crypto industry have created lot of expectation around the possibility of high return by trading those assets. The well-known efficient market hypothesis suggests the price of assets such as currencies reflect all available information, and as a result trade at their fair value. Although there is an abundance of data related to Bitcoin and its network, the author argues that not all market participants will utilise all this information effectively and as a result it may not be reflected in the price. This paper aims to take advantage of this assumption through various machine learning methods.[20] The following section isn't aimed at reporting the full list of algorithms applied to the time series domain, but it limits itself to the previous works carried out around Bitcoin price forecasting using statistical modelling and machine learning methodology. The array of research papers reported could be broken down into two main categories. Many research report that Bitcoin price formation follow dynamics that are different from many other fiat currencies. At the same time, they investigate the power and the association influence that some dependent variables might have on the Bitcoin exchange rate fluctuation. Typically, those leverage financial econometric applications to discover meaningful insight in the data. On the other hand, tons of studies have applied supervised algorithms to predict Bitcoin price as continuous regression target variable. Quite popular are also applications of Artificial Neural Network in the Stock Price Performance[38] and Ensemble learning, XBoost, Random Forest particularly optimal to model and capture non-linear functions. Typically, the financial data analyzed in those research are freely sourced from the web through Web API and Scaping. This section presents some related

work in the area of both Bitcoin price prediction. Though, literature on using statistical modelling and machine learning to financial time series is abundant, I report those papers which have studied Bitcoin from a time series forecasting perspective. Numerous studies have been conducted recently on modeling the time series of Bitcoin prices as a new market variable with specific technical rules. Machine learning Bitcoin prediction has been extensively studies in [20]. The author implemented Bayesian optimised recurrent neural network (RNN) and Long short term memory to predict the closing price of Bitcoin in US dollars. Also, the dependent variable was studies in relation with Blockchain and feature engineered variables. he clealry shows performance difference between linear models and non-linear models applied to Bitcoin prediction task. The LSTM achieved the highest classification accuracy of 52% and a RMSE of 8% The popular ARIMA model for time series forecasting was implemented as a comparison to the deep learning models. As expected, the non-linear deep learning methods outperform the ARIMA forecast which performs poorly. Similar variables has been used in [13] to automate a bitcoin trading strategy. Using bitcoin price and payment network over the course of five years, they were able to predict the sign of the daily price change with an accuracy of 98.7%. Price sign prediction was modelled as binomial classification task trough SVM, random forest and binomial GLM. Trying to study the impact of web search media on the Bitcoin trade volume was the purpose of [19]. In this work they studied the existing relationship between Bitcoins trading volumes and the queries volumes of Google search engine. The results demonstrated that search volumes power could anticipate trading volumes of Bitcoin currency. Geourgoula et al. [12] investigated the determinants of the price of Bitcoin while also implementing sentiment analysis using support vector machines. A series of short-run regressions shows that the Twitter sentiment ratio is positively correlated with Bitcoin prices. The short-run analysis also reveals that the number of Wikipedia search queries and the hash rate have a positive effect on the price of Bitcoins. Outstanding results with blockchain network variables were achieved using Bayesian Neural Network in [15]. Selecting the most relevant features from Blockchain information involved in Bitcoins supply and demand, the author trained models to improve the predictive performance of the latest Bitcoin pricing process. According to [25] Bitcoin price major drivers are supply-demand fundamentals, global macro-financial indicators and BitCoins attractiveness for investors. The econometric approach implemented with an Auto-Regressive model to analyse the causality between endogenous time-series had important results in terms of statistical inference. More related to the statistical modelling domains, some studies have been dedicated to determining the factors that drive the price of Bitcoin. [17] is the first research to propose a multivariate approach which focused on the speculative component of the Bitcoin value. In this regard, the numbers of search queries on Google Trends and Wikipedia are used as proxies for investors interest and attention. More specifically, Kristoufek (2013)

employed a bivariate Vecto-AutoRegression (VAR) model for the weekly log-returns of bitcoin prices and Google Trends data. They found that the increased interest in the BitCoin currency measured by the searched terms increases its price. As the interest in the currency increases, the demand increases as well causing the prices to increase. Garcia 2014 [6] quantified four socio-economic signals about Bitcoin from large datasets: price on online exchanges, volume of word-of-mouth communication in online social media, volume of information search and user base growth. By using vector autoregression, we identify two positive feedback loops that lead to price bubbles in the absence of exogenous stimuli. [6] expanded the set of variables which may affect the bitcoin price, considering not only BitCoin attractiveness -measured by Google Trends data-, but also accounting for the impacts of BitCoin supply and demand. Ultimately, It is worth mentioning the study made in [26]. Within the research, the sustainability of Bitcoin as a global currency have been studied using an econometric approach to asses which factors most drives Bitcoin. They can be summarized in: market forces of BitCoin supply and demand, BitCoin attractiveness, and global macroeconomic and financial developments.

Chapter 3

Research Materials, Methodology and Exploratory Data Analysis

3.1 Research Variables

Two separate datasets have been used to statistically modelling the price dynamics of Bitcoin. While the first one is a long daily history exploring blockchain network cryptocurrencies related variables, the second one deals with supply/demand Bitcoin variables observed on the market at high frequency monitored at 10 minutes interval. All the data explored in this research were publicly available on the web at different websites. They were collected using web scaping technique in Python environment.

3.1.1 Bitcoin - Blockchain dataset

Through this datasets I wanted to understand a bit more about the blockchain network which at the same time run alongside the Bitcoin exchange price. The main source of inspiration for this dataset comes from this kernel [34] on the Kaggle website. For getting an answer to those questions, I started collecting data from the web at different sites using Web Scraping techniques in Python.

The following Bitcoin financial market data were collected from coinmarketcap.com at daily frequency from 2013-04-28 to 2019-05-20.

- Date : date of observation
- Open : Opening price on the given day
- High : Highest price on the given day

- Low : Lowest price on the given day
- Close : Closing price on the given day
- Volume : Volume of transactions on the given day
- Market Cap : Market capitalization in USD

Now that we have the price data, I wanted to dig a little more about the factors affecting the price of coins. Thanks to Blockchain Info, I was able to get quite a few parameters on a daily basis from 2013-05-23 to 2019-05-20.

- Date : Date of observation
- btc market price : Average USD market price across major bitcoin exchanges.
- btc total bitcoins : The total number of bitcoins that have already been mined.
- btc market cap : The total USD value of bitcoin supply in circulation.
- btc trade volume : The total USD value of trading volume on major bitcoin exchanges.
- btc blocks size : The total size of all block headers and transactions.
- btc avg block size : The average block size in MB.
- btc n orphaned blocks : The total number of blocks mined but ultimately not attached to the main Bitcoin blockchain.
- btc n transactions per block : The average number of transactions per block.
- btc median confirmation time : The median time for a transaction to be accepted into a mined block.
- btc hash rate : The estimated number of tera hashes per second the Bitcoin network is performing.
- btc difficulty : A relative measure of how difficult it is to find a new block.
- btc miners revenue : Total value of coinbase block rewards and transaction fees paid to miners.
- btc transaction fees : The total value of all transaction fees paid to miners.
- btc cost per transaction percent : miners revenue as percentage of the transaction volume.

- btc cost per transaction : miners revenue divided by the number of transactions.
- btc n unique addresses : The total number of unique addresses used on the Bitcoin blockchain.
- btc n transactions : The number of daily confirmed Bitcoin transactions.
- btc n transactions total : Total number of transactions.
- btc n transactions excluding popular : The total number of Bitcoin transactions, excluding the 100 most popular addresses.
- btc n transactions excluding chains longer than 100 : The total number of Bitcoin transactions per day excluding long transaction chains.
- btc output volume : The total value of all transaction outputs per day.
- btc estimated transaction volume : The total estimated value of transactions on the Bitcoin blockchain.
- btc estimated transaction volume usd : The estimated transaction value in USD value.

3.1.2 Methodology of the research

This research is concerned with predicting the price of Bitcoin using statistical modelling and Deep Learning. The purpose of the research is double-fold. First, we evaluate prediction accuracy of classical statistical time series models and modern deep learning approaches to financial time series. The former is taken as comparison baseline, while the research invests more effort to explore Artificial Neural Network model fitting to time series data. Secondly, we introduce econometric models and methods useful for studying jointly multiple time series. The next chapter considers econometric model that belong to vector or multivariate time series analysis to make inference on lead-lag relationship between blockchain network variable and Bitcoin closing Price. This chapter concludes addressing a statistical exploratory analysis and a review of which metrics are used to evaluate the forecasting accuracy. Then, the rest of the research focuses on time series modelling. Exploration of the dataset will help us understanding statistical properties of our dependent and independent variables, and also to evaluate variable transformation to accommodate forecasting modelling assumptions. Many studies of financial forecasting found in the literature focus on an univariate approach. More often than not, the world around us is more complicated than that. The dataset contains many variables collected at the same time frequency of Bitcoin price to leverage possible

association between them. Linear time series analysis provides a natural framework to study the dynamic structure of financial time series. The application of linear time series models includes stationarity, dynamic dependence, autocorrelation function, modeling and forecasting. The econometric models explored are auto-regressive integrated moving average (ARIMA) without and with seasonal component (SARIMA) and Vector auto-regressive models (VAR) in the multidimensional time series scenario. Those models contribute to statistically infer possible association between Bitcoin price and Blockchain Network variables. Ultimately, chapter 5 extends the forecasting statistical modelling to applied Machine learning and Deep Learning algorithms.Advanced research on State-of-the-art Deep Learning algorithms leads to consider Long-Short-Term Memory (LSTM) (a particular version of Recurrent Neural Network) to forecast financial time series. For this reason, the research reports a comparative forecast accuracy of LSTM against classical statistical modelling to predict Bitcoin price dynamics. Given the complexity of the task, deep learning makes for an interesting technological solution based on its performance in similar areas. Tasks such as natural language processing which are also sequential in nature and have shown promising results. This type of task uses data of a sequential nature and as a result is similar to a price prediction task. [20]. This body of research builds on existing literature in the area which is reported in the bibliography of the research.

3.1.3 Research question

- ***With what accuracy can the direction of the price of Bitcoin be predicted using classical statistical modelling and machine learning?***
- ***What relationship do exist between Bitcoin price Index and Blockchain transaction features? Are they statistically significant predictors for the price of Bitcoin?***
- ***Does any Blockchain network variable contain some signal to improve Bitcoin price prediction***
- ***What are the dynamic properties of those time series? How much does x influence y? What are the contributions of the various lags? Given that this information (the so-called transfer function) is required, parsimonious modeling should follow (see below).***

3.1.4 Purpose

The purpose of this study is to find out with what accuracy the direction of the price of Bitcoin can be predicted using machine learning methods and whether Blockchain network variables are statistically significant predictors. Blockchain technology, first implemented by Satoshi Nakamoto in 2009 as a core component of Bitcoin, is a distributed, public ledger recording transactions. Its usage allows secure peer-to-peer communication by linking blocks containing hash pointers to a previous block, a timestamp, and transaction data. Bitcoin is a decentralized digital currency (cryptocurrency) which leverages the Blockchain to store transactions in a distributed manner in order to mitigate against flaws in the financial industry.[**reference20**] The research built on statistical time series domain to show the potential of time series forecasting. Eventually, it ends up studying forecasting performance of recurrent neural network to financial time series data. Formation of Blockchain, a core technology of Bitcoin, distinguishes Bitcoin from other fiat currencies and is directly related to Bitcoins supply and demand. While much research exists surrounding the use of different machine learning techniques for time series prediction, research in this area relating specifically to Bitcoin is lacking.[20] To the best of our knowledge, in addition to macroeconomic variables, direct use of Blockchain information in ARIMA and Recurrent Neural Network models has not been sufficiently investigated to describe the process of Bitcoin price. To fill this gap, the current study systematically evaluates and characterizes the process of Bitcoin price formation by modeling and predicting Bitcoin prices using historical values and Blockchain network information. Based on our research, few researchers have explored the relationship between Blockchain information and Bitcoin prices. We sought to explore additional features surrounding the Bitcoin network to understand relationships in the problem space, if any, while also exploring multiple machine learning algorithms and prediction methodologies within our research.

3.2 Exploratory Data Analysis and Feature engineering

The scope of this section is to prepare the dataset for analysis carried out in the time series forecasting modeling of this research. A range of time series plotting descriptive, univariate and multivariate statistics across the dataset will contribute to gain a ”feel” regarding our dataset. Particularly, stationary time series properties are intensively investigated across each variable. Their properties form the main assumptions of many time series statistical methods explored. Even though perfect stationarity might be hard to achieve, in the proper part of the analysis I am trying to transform the original raw data to generate stationary time series. The two data source described above requires

different treatments, thus to avoid confusion of which data source the analysis is referring to, I am going to keep them separated. This section assumes that all required data cleaning preprocessing steps have already been taken. First, the research examines the Bitcoin2min dataset and related market demand-supply variables. Then, it moves to the more complex and rich data source about Bitcoin-blockchain variables. The successful outcome of a Data Science process depends on the way the analyst design the data pipeline. Within this framework, both data preprocessing and exploratory data analysis create a foundation to model fitting and model training. All the discussion about the main characteristics of Time series data discussed in chapter 2 finally start show their utility. In fact, we are going to turn into practice many statistical methods described and more. Out of the great deal of steps one could take to run a time series exploratory data analysis, I decide to concentrate on the following key aspect of my time series data:

- Importance of working with Stationary data
- Data visualization of raw data to check the presence of trend, seasonality and cyclical pattern
- Smoothing techniques to estimate any deterministic trend or seasonal fluctuation
- Time Series Decomposition methods
- Time series probability distribution
- Descriptive statistics
- Estimation of the correlation structure: autocorrelation, partial autocorrelation
- Stationarity validation
- Cross Covariance and lagged scatter plots

Eventually, my hopes are to come up with insightful facts and challenging hypothesis to be answered in the modelling part of the research. Well dive into the implementation part of this article soon, but first its important to establish what were aiming to solve.

3.2.1 Importance of working with Stationary data and how to check it

If there are any golden rule in time series statistical modelling, one of them is certainly **stationarity**. Basically, stationarity can be thought as the reason why we conduct an exploratory data analysis on time series data. Stationary is inherently related to

time because it guarantee that a stochastic process maintains its properties over the time. Time related data makes really awkward to work with regression models. First, sequence data cannot be assumed as independent realization of a random variable. Also, Homoskedasticity and uncorrelated error which are two assumptions of unbiased and efficient estimators of regression models coefficients are likely to be false. Plus, if the probability distribution of such random variables should vary through time, we would not be confident to uniformly translate past pattern into the future. For those and more reasons, stationarity is the first step of our journey. Overall, in this section our scope will be restricted to data exploring not to building time series models yet. If the time series is not stationary, first we have to make it stationary. For doing so, we need to remove the trend and seasonality from the data. In a few words, stationarity property is the point of reference for this statistical exploratory data analysis section.

A time series is said to be stationary if its statistical properties such as mean, variance remain constant over time. But why is it important? Most of the time series models work on the assumption that the time series is stationary. Intuitively, we can sat that if a time series has a particular behaviour over time, there is a very high probability that it will follow the same in the future. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.[14] For instance ARIMA models and VAR models assume stationarity in the data to be true. Stationarity is defined using very strict criterion. However, for practical purposes we can assume the series to be stationary if it has constant statistical properties over time:

- The mean of the series should not be a function of time rather should be a constant
- The variance of the series should not a be a function of time. This property is known as homoscedasticity
- The covariance of the i th term and the $(i+m)$ the term should not be a function of time

So why is stationarity so important? Because it is easy to make predictions on a stationary series since we can assume that the future statistical properties will not be different from those currently observed

The reason I took up this section first was that until unless your time series is stationary, you cannot build a time series model. In cases where the stationary criterion are violated, the first prerequisite becomes to stationarize the time series and then try to fit a stochastic model for forecasting.

There are multiple ways of bringing this stationarity out. Exploring data becomes the most important part in a time series model without this exploration, you will not know whether a series is stationary or not.

3.2.2 How to Check Stationarity of a Time Series?

A Time Series is said to be stationary if its statistical properties such as mean, variance remain constant over time.[14] Intuitively, we can say that if a time series has a particular behaviour over time, there is a very high probability that it will follow the same in the future. So, more formally, we can check stationarity using the following:

- Plotting Rolling Statistics: Those moving centered statistics are computed along a moving sliding windows.This is also known as convolving over the time index. That's why the rolling term in the definition. Clearly, those tools shows the evolution of the statistics over time which in case of stationarity assumption is constant and not affected by the time origin.

- Augmented Dickey-Fuller Test: In order to evaluate if a time series follow a non-stationary process we apply the so called Augmented Dickey-Fuller statistical test. The statistical tests for objectively determining whether differencing is required to stationarize a time series are known as unit root tests. Therefore, the null hypothesis of the ADF is:

$$H_{0:Y} = 0 \tag{3.1}$$

  against the alternative hypotehsis:

$$H_{1:}Y < 0 \tag{3.2}$$

  In other words, the null hypothesis is presence of unit root or non-stationarity (the first-order differences x_t of the original series can be expressed as a linear regression model of the previous time index and first-order differences up to a lag of m times indices).Whereas the alternate hypothesis suggests stationarity of the data. The null hypothesis is rejected of the test statistic for Y is less than the critical negative value for the given level of confidence. In case the series is already stationary, the linear regression model represents a random drift for which the change at time index t depends on the previous value and not the previous differences, which are iids for the stationary process.[10]

- Ljung-Box: The Ljung- Box test determines if the observed auto-correlation is statistically significant. The null hypothesis of the Ljung-Box test is that the time

series consist of random variations and lacks predictable autocorrelation while the alternate hypothesis proposes that the observed autocorrelation is not random.

- Residuals white noise: Our task in statistical forecasting terminates when the residuals from the model fit are realization of identically distributed random variable with mean 0 and constant variance. When those conditions are met we say that the residuals error term follow a white noise stochastic process as describe in chapter 2.

3.2.3 How to make a Time Series Stationary?

From the concepts developed on chapter 2 section 2.1, we can finally move on to actually making our series stationary. Always keep in mind that in order to use time series forecasting models, it is necessary to convert any non-stationary series to a stationary series first.

Though stationarity assumption is taken in many time series models, almost none of practical time series are stationary. So statisticians have figured out ways to make series stationary, which well discuss now. Actually, its almost impossible to make a series perfectly stationary, but we try to take it as close as possible.

Lets understand what is making a time series non-stationary. There are 2 major reasons behind non-stationarity of a time series:

- Trend varying mean over time.
- Seasonality variations at specific time-frames.

The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to get a stationary series, so that statistical forecasting techniques can be implemented on this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.[14]

- Smoothing techniques: Applying moving averages at different lags reduces noise and drastically shrink the variability of the data. Sometimes it is useful to overlay a smoothed version of the original data on the original time series plot to help in revealing patterns in the original data. One of the simplest and most widely used is the ordinary or simple moving average. The moving average has less variability than the original observations; in fact, if the variance of an individual observation

y_t is σ^2, then assuming that the observations are uncorrelated the variance of the moving average is:[9]

$$Var(M_T) = Var(\frac{1}{N} \sum_{t=T-N+1}^{N} y_t) = \frac{\sigma^2}{N} \tag{3.3}$$

- Differencing: It is widely use to remove trend and seasonal variation applying the difference operator to the original time series to obtain a new time series, say:

$$x_t = y_t - y_{t-1} = \delta y_t \tag{3.4}$$

  where δ is the backward difference operator. Differencing has two advantages relative to fitting a trend model to the data. First, it does not require estimation of any parameters, so it is a more parsimonious (i.e., simpler) approach[9]; and second, model fitting assumes that the trend is fixed throughout the time series history and will remain so in the (at least immediate) future. Differencing can allow the trend component to change through time.

- Transformation: Taking the logarithm of the raw time series

- Model Time series residuals from mathematical decomposition: it is possible to reduce the standard deviation of the time series removing deterministic trends that cannot be a realization of some random process. A time series r_t is called a white noise if r_t is a sequence of independent and identically distributed random variables with finite mean and variance

If series data contains a trend or a clear periodic constant variation (seasonality), we can fit some type of curve to the data and then model the residual from that fit. The above techniques are intended to generate series with constant location and scale. Although seasonality violates stationarity, this is usually explicitly incorporated into the time series model. It might be the case that after removing the trend, the run sequence plot indicates that the data have a constant location and variance, although the pattern of the residuals shows that the residuals depart from model in a systematic way. If seasonality is present, it must be incorporated into the time series model. In this section, we discuss techniques for detecting seasonality. We defer modeling of seasonality until chapter 4.

- Seasonal subseries plot: This type of plots are useful if we know the period of the seasonality. If it is unknown, autocorrelation plots can be used to determine it.[31] This plot allows you to detect both between group and within group patterns. While the y-axis of the plot shows the response variable, the x-axis contains time

ordered season that could be at each quarter, month, week day, hour , minutes ecc.

- Box Plots: They are a powerful descriptive data visualization technique because convey location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different groups of data (seasons). Hence, a single box-plot is drawn for each seasonal factor of interest showing median, quantile range and distribution of the response variable.

- Autocorrelation plot: Although we defined the population autocorrelation function just for stationary time series data, the sample ACF can be calculated for any time series, including deterministic signals as trend and seasonality. Some results for deterministic signals are helpful for explaining patterns in the ACF that we do not consider realisation of some stationary process[21]. More on the interpretation of the autocorrelation and partial autocorrelation plot in chapter 4 where we explain how those two tools help us determine the best order of the ARIMA process.

- Seasonal differentiation: Difference can also be used to eliminate seasonality. Define a lag d seasonal difference operator as:

$$\delta_d y_t = (1 - B^d) = y_t - y_{td} \tag{3.5}$$

  where d is the seasonality frequency we believe the time series repeat itself over time

Most of the exploratory task we talked about so far deal with univariate time series. However, the datasets shows multiple variables at play, and handling all of them at the same time is where beauty and powerful time series modelling earn his worth. In fact, we investigate which relationships exists among exogenous variables. Lagged Cross-correlations and scatter plots are employed to estimate the predictive effects influence that lagged independent variables have on the dependent variables. En example of cross-correlation is to determine possible leading or lagging relations between two series. If then the model holds, the series x is said to lead y for lag greater than zero. On the other hand, variable scatter plots of dependent and lagged exogenous variables can detect non-linear relationship which cross-correlation wouldn't be capable of. The scatter plot cannot establish a causal relationship between two variables, but it is useful in displaying how the variables have varied together in the historical data set. It is important to note that in statistical terms variable correlations doesn't mean variable causation. In fact, two variable might show high correlation just because they are both trending (spurious regression), though completely far off from any causality relationship. The essential point here is that commonly, meaningless correlations exist between independent pairs

	Open	High	Low	Close	Volume_mil	Market_Cap_mill
mean	2445.790248	2517.224354	2368.416360	2449.176545	2379.785863	40879.727625
std	3361.109444	3482.560353	3216.247109	3362.610026	4267.483664	57433.984136
min	68.500000	74.560000	65.530000	68.430000	2.857830	778.411179
25%	349.070000	358.067500	342.312500	349.595000	29.588200	4806.462365
50%	630.745000	643.425000	619.775000	630.985000	90.856096	9287.685138
75%	3902.902500	3969.290000	3835.862500	3905.770000	3932.480000	67792.486850
max	19475.800000	20089.000000	18974.100000	19497.400000	33167.197581	326502.485530<>

of time series that are themselves autocorrelated.[11] simulation. If two stationary series, xt and yt, are independent of each other (i.e., if values in one series at any time do not provide information about the values in the other series at any time), then it is still possible that the cross-correlation between the series can appear to be significantly nonzero when judged against standard criteria for significance for independent pairs of measurements. [11] Cross-correlation deserve special treatment in time series analysis. The autocorrelation structure of two time series both heavily serial correlated add an element of uncertainty to the correlation interpretation. To solve this issues, Time series that are to be modeled or related to each other are normally molded first to conform within probabilistic limits to weak stationarity. The stationarity of both series of interest is required in order to assess cross-correlation. Our purpose here is to point out that neglecting the consideration of autocorrelation, and relying instead on a simple cross-correlation analysis, may be inadequate and even dangerous in these circumstances, and then to summarize an approach providing a principled analysis.

3.2.4 Exploratory Data Analysis Daily Bitcoin Market price and Blockchain features

Developing a forecasting model should always begin with graphical display and analysis of the available data. Many of the broad general features of a time series can be seen visually. This is not to say that analytical tools are not useful, because they are, but the human eye can be a very sophisticated data analysis tool. To paraphrase the great New York Yankees catcher Yogi Berra, You can observe a lot just by watching.[9]. As a matter of fact, we will heavily use visualization techniques for data communication and story telling. Our exploratory data analysis begins from the Bitcoin daily market price dataset. A popular visualization in financial trading is Candlestick OHLC graph to quantify the daily price fluctuation of the asset. Each bar is built from the daily open, low, close and high price as in 3.1

Across the date range Bitcoin price shows a quite stable volatility until the end of 2016. Throughout the year 2007, its price raised exponentially to 20089.00 USD, its maximum value on the 2017-12-17. As the price increased in 2017, so did the volume. Since the early days of this year, the volume hasn't stopped increasing. The dependent variable

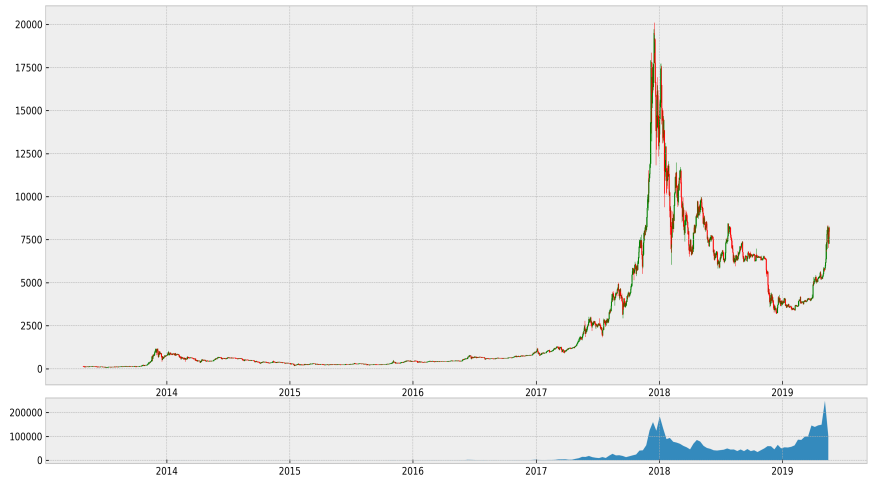

FIGURE 3.1: Bitcoin candlestick graph

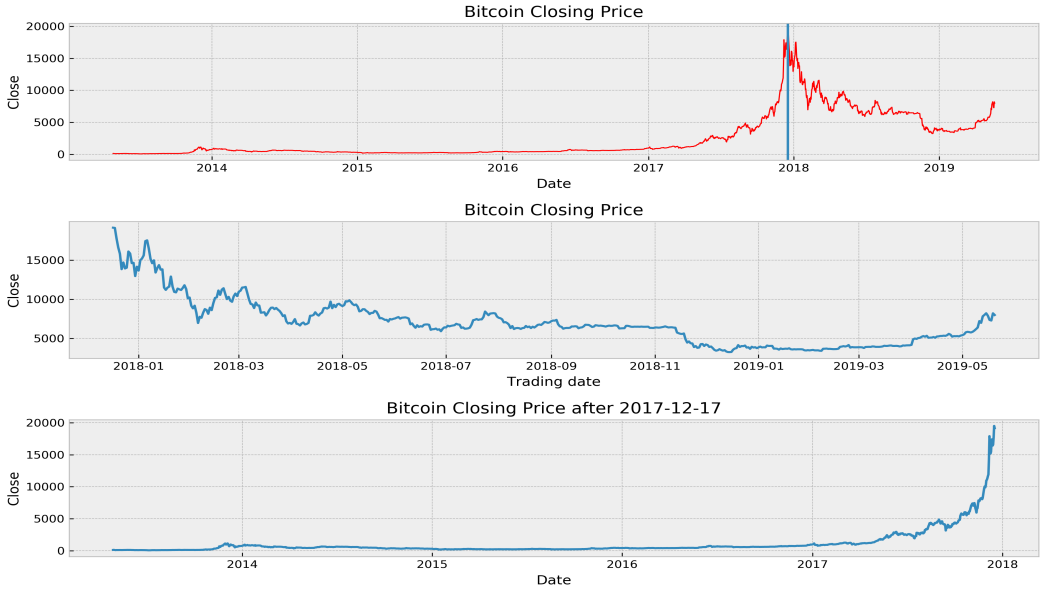

FIGURE 3.2: Bitcoin Clossing Price

of the research is the daily bitcoin closing price in USD. A great deal of this exploratory data analysis aim at transforming this time series into a stationary random variables which has constant properties (mean, variance) and covariance that doesn't depend on the time but on the lag between observations. As we go further in the analysis, we are going to unveil all its statistical probability distribution using the theoretical tools defined in the previous section.

Aggregated statistics group the data by different period to compute the measure, In this figure 3.3 we displayed the time series at different temporal aggregated mean.

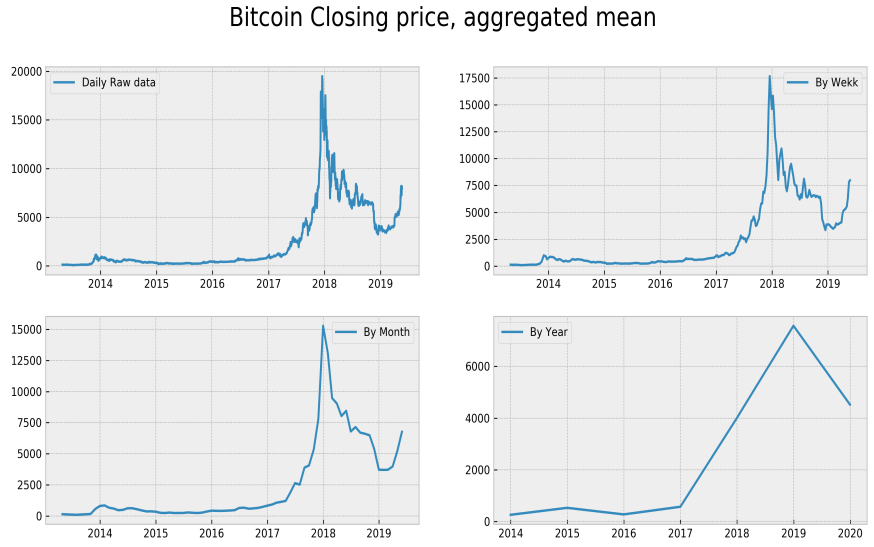

FIGURE 3.3: Btc Close Mean Aggregation

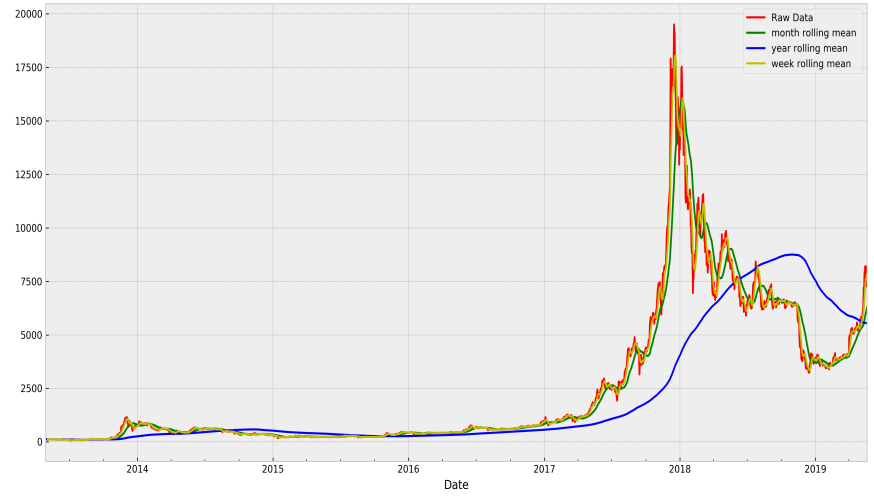

FIGURE 3.4: Btc Close Rolling mean

On the other hand, Smoothing techniques or rolling statistics are quite useful to spot long trend and seasonal variation across the series. 3.4

Clearly, the series is an example of non-stationarity time series. Also, the stochastic behaviour doesn't show any definite deterministic structure such as weekly 3.5 or monthly seasonality 3.6.

Take last weekly closing price as a benchmark. There is no clear day of week effect in the data 3.7.

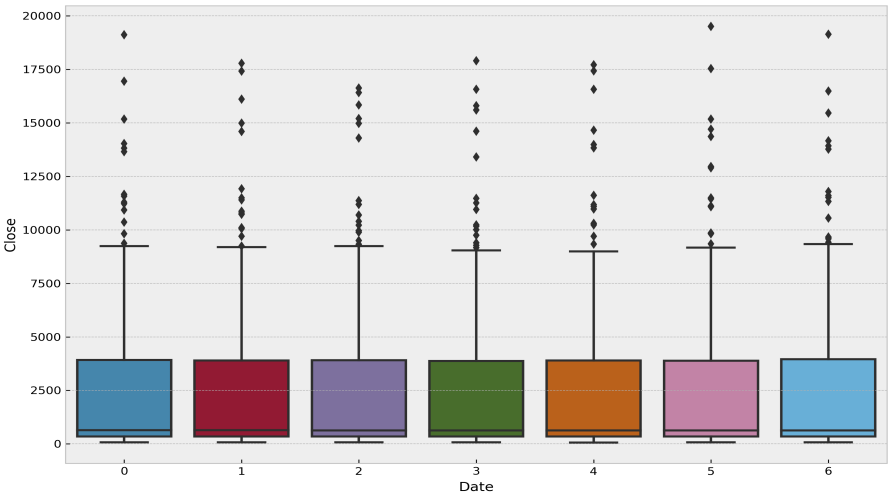

FIGURE 3.5: Weekly Box Plots

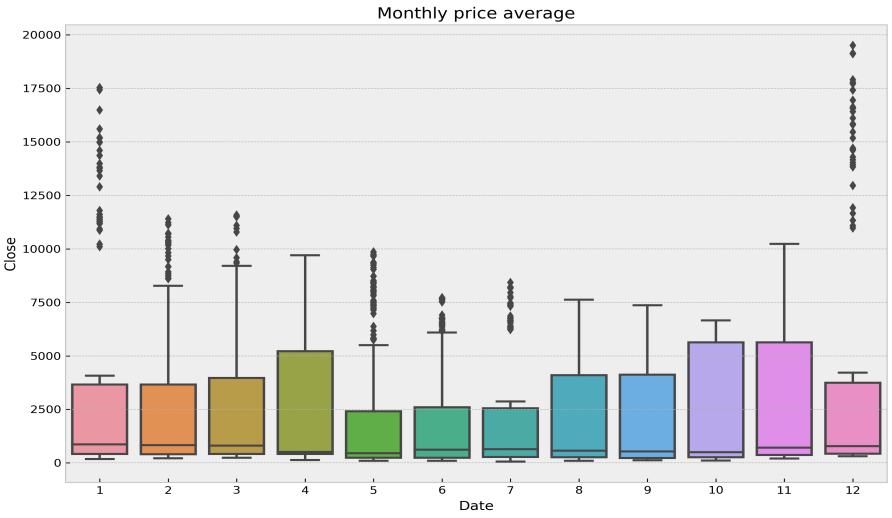

FIGURE 3.6: Monthly Box Plots

(more on seasonality if time left)

As expected trading volume increased with the price. Look at figure 3.8. In the figure is clear that the activity of the market depends on price expectations. Yet, this might reveal the immaturity of the crypto as system of payments because of such volatility due to speculative currency trading.

Now, it is time to tackle the preliminary time series stationarity assumption. Econometric models assumed the data to be stationary. Its presence gives the analyst the guarantee that parameter estimations are unbiased with minimum variance. Also, It

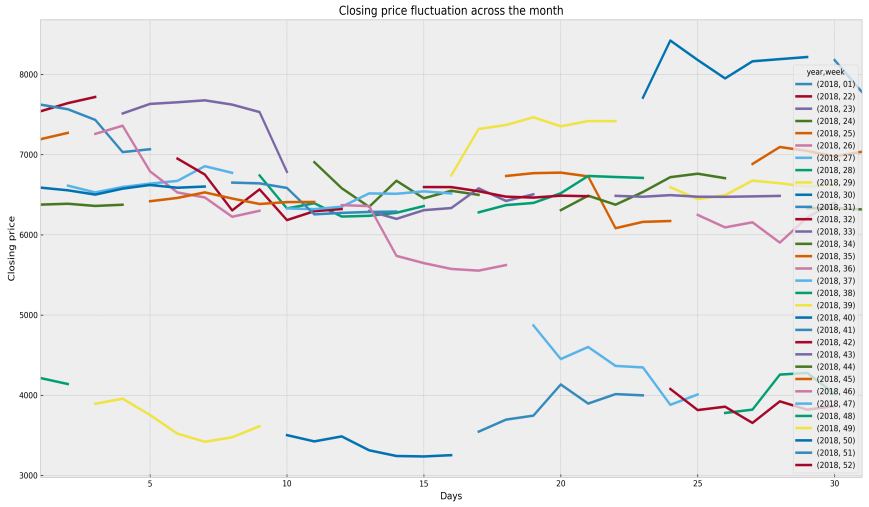

FIGURE 3.7: Weekly Close in 2018

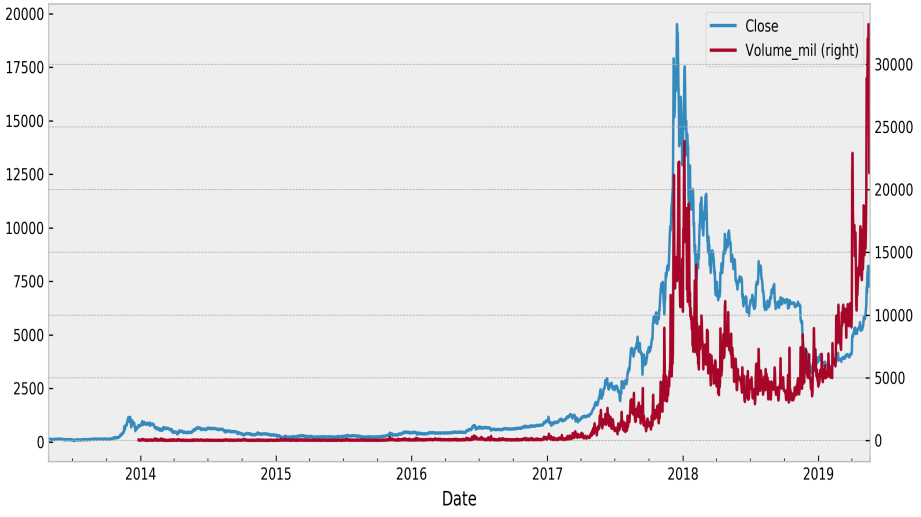

FIGURE 3.8: BTC close Volume plot

allows time series regression analysis to leverages many asymptotic convergence properties and error distribution assumptions used in cross-sectional data for robust statistical inference. Stationarity is inherently related to the time component of time series data. However, while the time component adds additional information, it also makes time series problems more difficult to handle compared to many other prediction tasks. For more reference on stationarity, both the previous section and chapter 2 deep dive into the mathematical formulation.

In the following plot (ref figure) we look at stationarity from different angles. The

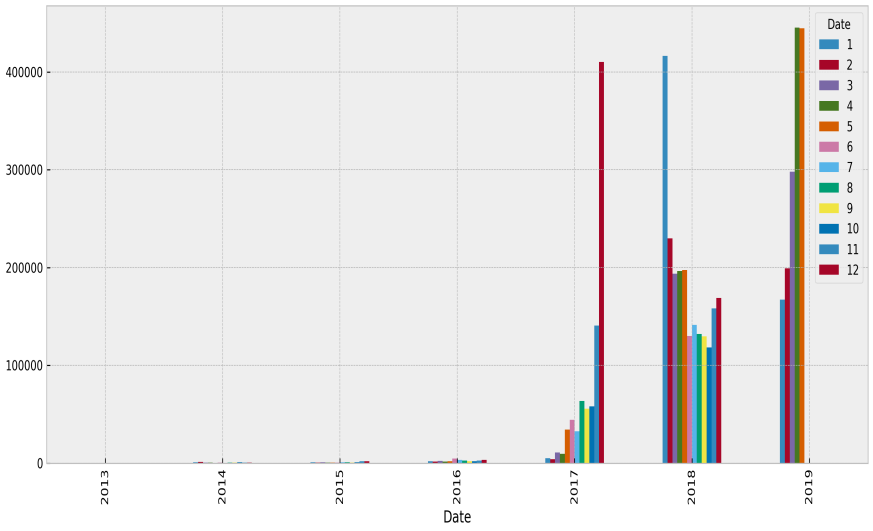

FIGURE 3.9: Histogram of BTC Volume

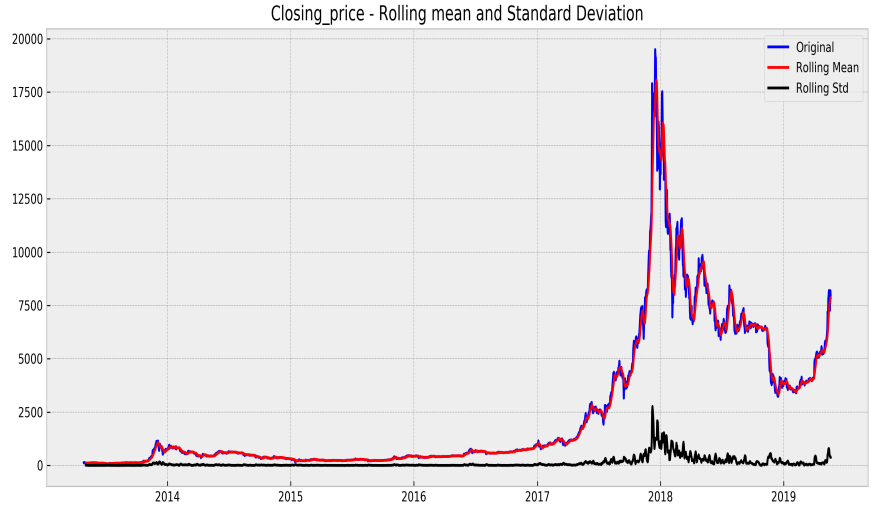

FIGURE 3.10: Visual Test Stationarity BTC close raw data

plots give us a feeling whether the statistical probability distribution measured by the first and second moments are constant over the time period. Below, we run a unit-root test, the Augmented Dickey-Fuller Test to accept or reject the null hypothesis of stationarity according to an alpha test of 0.05%. Ultimately auto-correlation and partial auto-correlation plots estimates the serial correlations structure at different lags. In the next chapter, ACF and PACF will have central role in defining the right order of the ARIMA process of the residual stochastic time series. 3.10

From the above figures and calculated test statistics the data I are actually modeled using

Results of Dickey-Fuller Test:	
Test Statistic	-1.614047
p-value	0.475852
#Lags Used	27.000000
Number of Observations Used	2186.000000
Critical Value (1%)	-3.433345
Critical Value (5%)	-2.862863
Critical Value (10%)	-2.567474

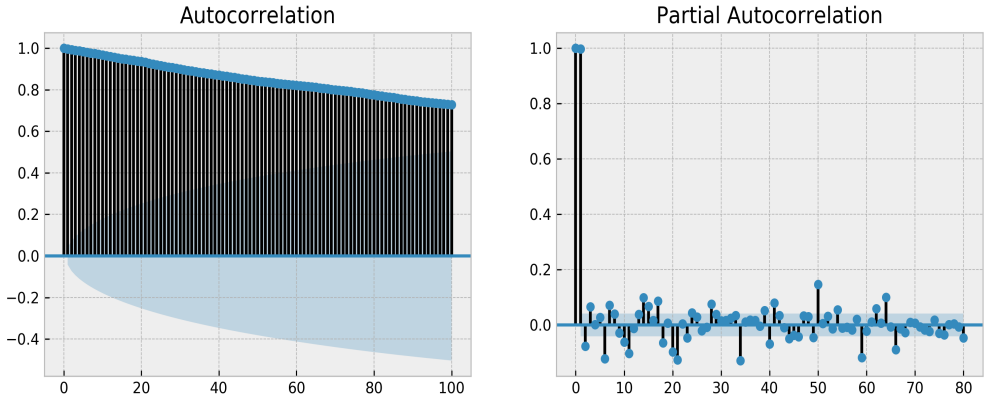

FIGURE 3.11: ACF PACF BTC Close raw data

Results of Dickey-Fuller Test	
p-value	1.815140e-12
#Lags Used	2.700000e+01
Number of Observations Used	2.185000e+03
Critical Value (1%)	-3.433346e+00
Critical Value (5%)	-2.862864e+00
Critical Value (10%)	-2.567475e+00</pre>
Critical Value (10%)	-2.567474</pre>

a random walk process. As the name indicates, a random walk is a completely stochastic process. Due to this, the idea of using historical data as a training set in order to learn the behavior and predict future outcomes is simply not possible. In order to make the time series stationary, several data transformation have been tried to decrease variability. The final stationary time series for statistical modelling was obtained applying a logarithmic scale and the first shift difference to the raw closing data. From the table below at a p-value less than 1% we cannot reject the null hypothesis.

It's now time to turn our attention to the Blockchain features of our datasets. So far, Time series analysis have been studied in its univariate dimension. Multidimensional time series analysis involves exploring existing lead-lag relationship between dependent and independent variables. In practice, my goal is to evaluate if any Blockchain variable rise/fall imply some predictive trading signal. One way to find correlation (linear

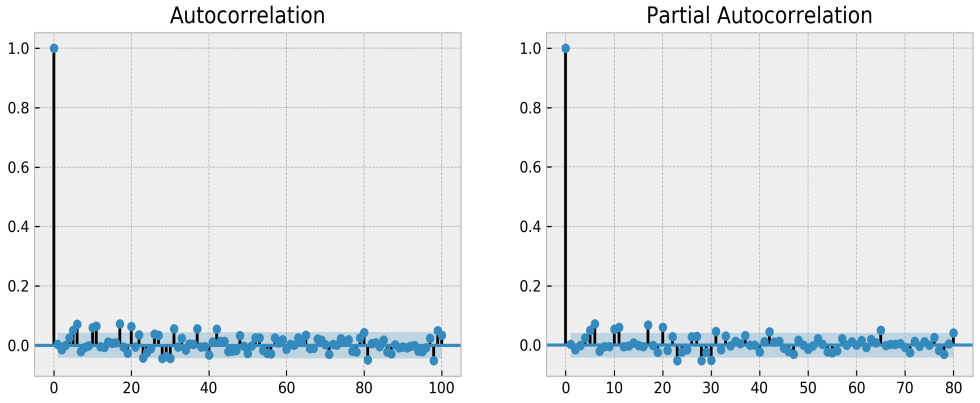

FIGURE 3.12: ACF PACF First difference log transformed Close

association) between timeseries is to look at cross-correlation of the timeseries. Cross-correlation is computed between two timeseries using a lag, so when creating the correlation matrix I will specify the correlation as well as the lag. Before computing the cross correlation, it is important to have wide-sense station (often just called stationary) data. Also, it is important not to confuse correlation with causation or causation with forecasting. A variable x might be useful to forecasting a variable y, but that doesn't mean x caused y. By studying k-dimensional time series jointly, we can assess the temporal and contemporaneous dependence between Bitcoin price and features of the blockchain. An important assumption for cross-correlation estimation in multidimensional time series is at least weak stationarity of each time series. If a k-dimensional time series z_t is weakly stationary the cross-covariance matrix measures the linear dynamic dependence as a function of lag l, not of the time index t Most such series are individually autocorrelated: they do not comprise independent values. Given this situation, an unfounded reliance is often placed on cross-correlation as an indicator of relationships (e.g., referent vs. response, leading vs. following). Such cross-correlations can indicate spurious relationships, because of autocorrelation.[11] Think about taking the correlation of two series (like two different cryptocurrencies) that both trend upward over time. You would get a strong correlation, but this is only because the series increase over time, not because movement in one is associated with movement in the other. This is spurious correlation. In order to take the correlation of two time series, you would want to make them stationary first (by differencing for example). Often (but not always) a strong correlation between two non-stationary time series will disappear after differencing. Given this danger, our approach would be to estimate the autocorrelation structure, the establishment of stationarity, the determination of cross-correlation, the assessment of Grager Casuality. All this steps will make inference on which variables and which lags should be included in vector auto-regressive models fitting in the next chapters as we study the relationships between time series to improve Bitcoin price prediction. As a

matter of fact, blockchain transaction features are considered exogenous predictors such as the series representing the control process(es). Time series that are to be modeled or related to each other are normally molded first to conform within probabilistic limits to weak stationarity.[11].

The dependent variable or target for this correlation analysis was the btc market price expressed as Average USD market price across major bitcoin exchanges. Each variable was tested about stationarity with the popular unit root test (Augmented Dickey-Fuller Test) and The KPSS (Kwiatkowski-Phillips-Schmidt-Shin) statistical test tests. In any case of non-stationarity evidence the variable was transformed to a stationary process taking the first difference of the logarithmic scale. Only for some blockchain transaction features it was necessary to remove the weekly seasonality taking the difference of their observations with their previous week values. Many of the variable in the dataset have high serial correlation with their previous values. This could be the product of random walk processes. The following figures report the correlation structure before and after stationarization.

Features stationarity had the effect to remove the spurious correlation between the bitcoin market price and the blockchain features. The cross-correlation at different lags than $t - 1$ didn't show any statistically significant linear associations. Those results might prove the random walk nature of blockchain network variable and the absence of causality associations with the target variable.

Finally, we performed a Granger Causality test whether there are likely casual relationship between the series. Granger causality assesses whether the relationship between the two series (now again in their original stationarized forms, not in their prewhitened forms) is likely to contain a causal element, considered from a statistical perspective. To be more precise, a variable x can be said to be Granger-causal of variable y if preceding values of x help in the prediction of the current value of y, given all other relevant information. The Null hypothesis is: the series in the second column, does not Granger cause the series in the first. If the P-Values are less than a significance level (0.05) then you reject the null hypothesis and conclude that the said lag of X is indeed useful. A Granger causality assessment confirms that none of these cross-correlations are significant, in the sense of predictive.

From the conclusion above, my hypothesis is that classical time series analysis (historical prices alone) might be insufficient when predicting such volatile markets. Using Machine Learning with variety of predictors on top of ARIMA and Deep Learning model will improve prediction performance significantly.

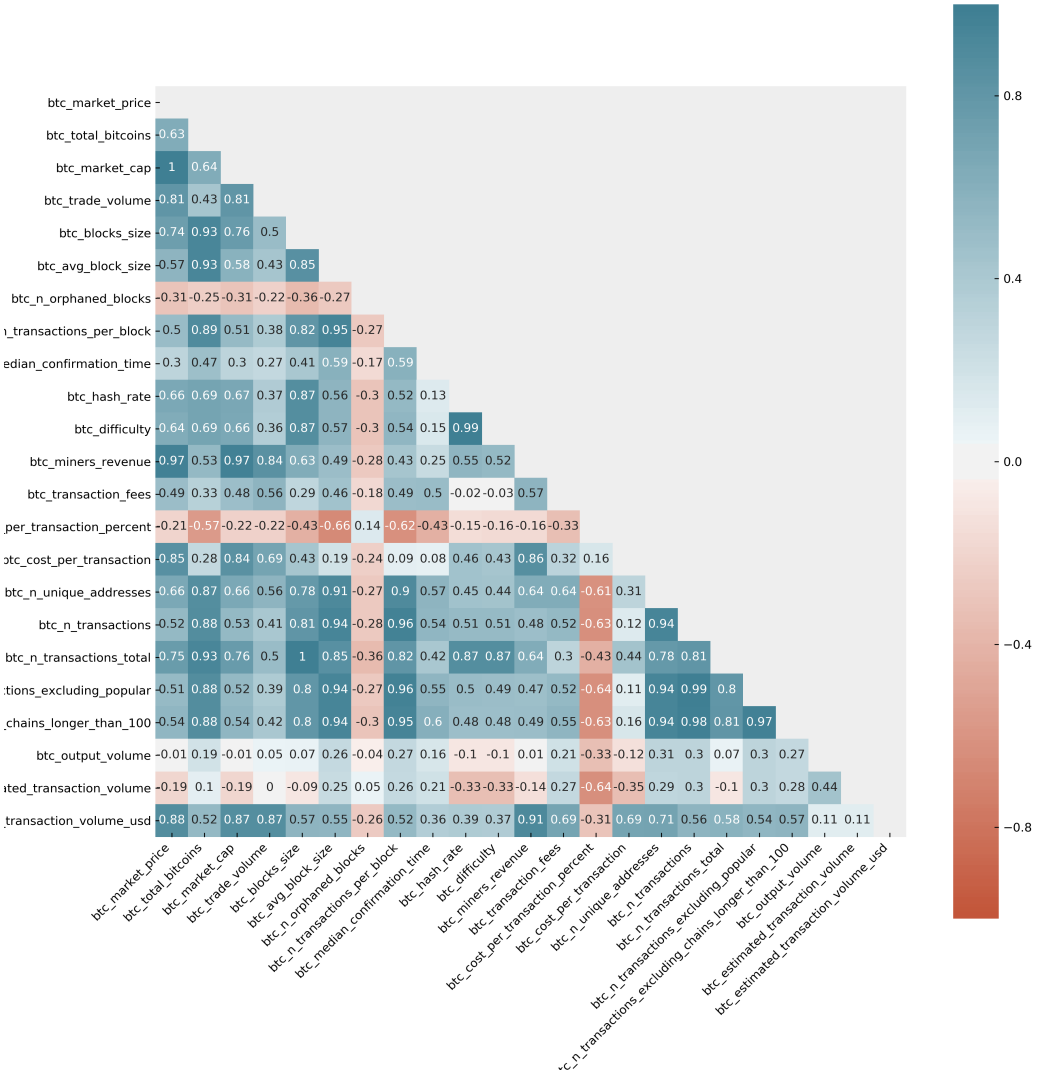

FIGURE 3.13: Lag 0 cross correlation non-stationary features

3.3 Forecast accuracy evaluation

: introduction to metrics and model evaluation:

This research deals with the forecasting process of financial time series. A forecast is a prediction of some future event or events- As such, it involves an element of uncertainty that we are trying to model using statistical and mathematical rigor. The strategic vision of the forecasting project defines the optimal timing for data acquisition, forecast production and model review. Throughout the research, the forecasting modeling are assumed to take a short-medium term. The reason that forecasting is so important is that prediction of future events is a critical input into many types of planning and

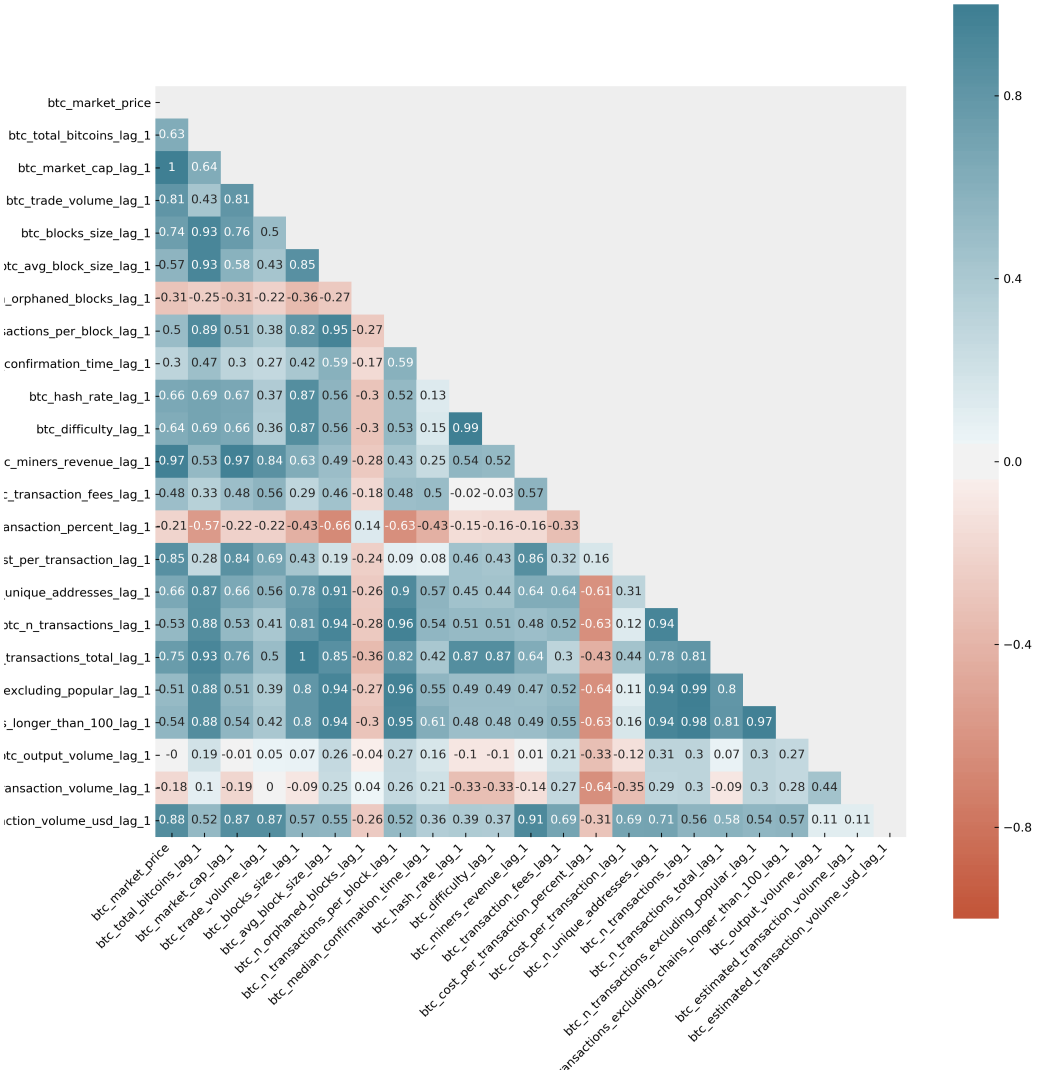

FIGURE 3.14: Lag 1 cross correlation non-stationary features

decision-making processes. Now, it time to define some forecasting terminology as well as statistical metrics to measure the quality of our predictions. A forecasting models summarizes pattern in the data and express a statistical relationship between previous and current values of the variable. Then the model is used to project the patterns in the data into the future[9]. The type of forecast can be important. Forecast interval differs from a point forecast because it includes a lower and upper bounds which the forecast are likely to be. Other important features of the forecasting problem are the forecast horizon and the forecast interval. The forecast horizon is the number of future periods for which forecasts must be produced. The forecast horizon is also often called the forecast lead time. The forecast interval is the frequency with which new forecasts are

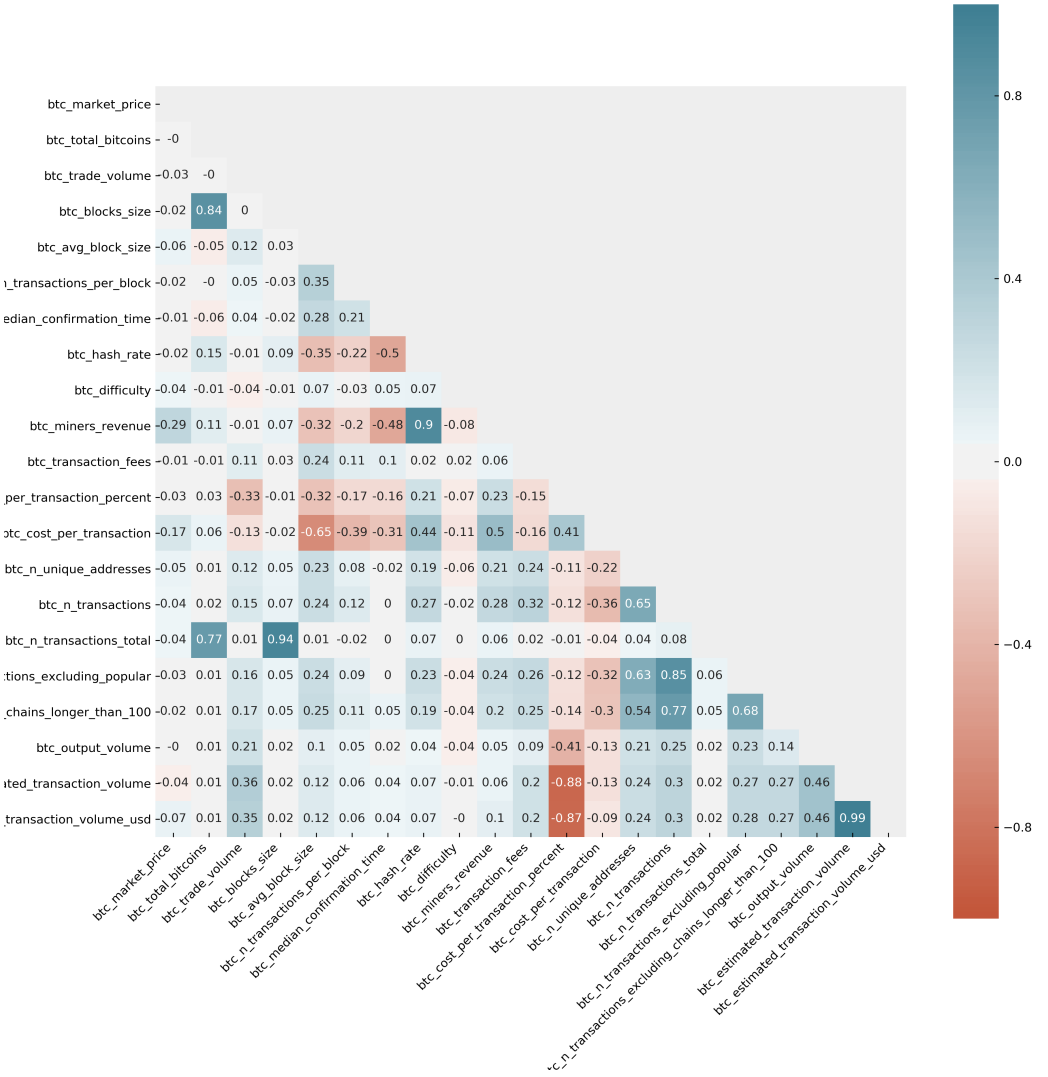

FIGURE 3.15: Lag 1 cross correlation non-stationary features

prepared. Moving to forecasting performance, it is importance to define the methodology followed. Future accuracy is evaluated using out of sample forecast error, to distinguish it from the residuals that arise from a model-fitting process. In an ideal situation, the forecasts would adequately model all of the structure in the data, and the sequence of forecast errors would be structureless (i.i.d. white noise process).[9] There are often several competing models that can be used for forecasting a particular time series. The use of Root Means squared error for continuous target variable offer a powerful metrics to asses the variability in forecast error.

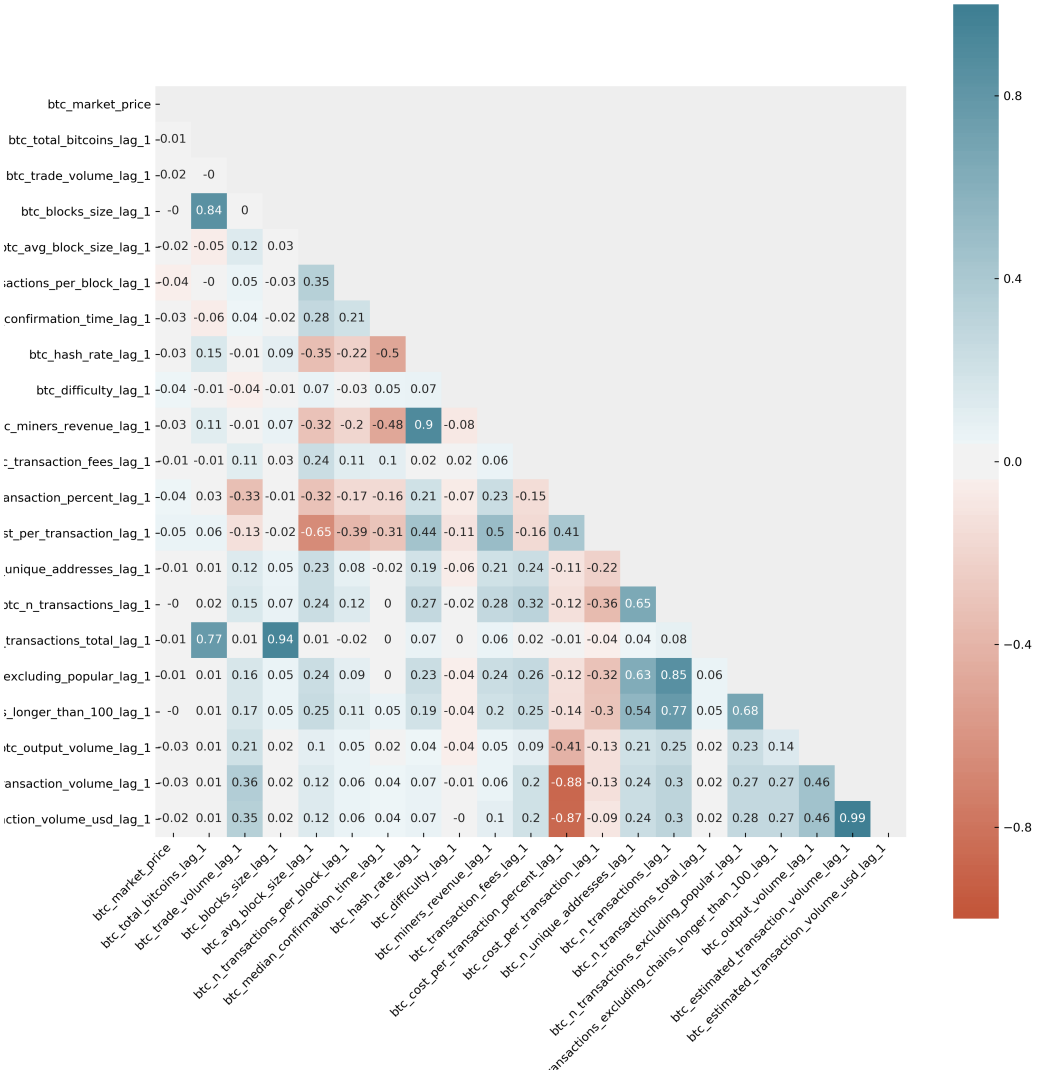

FIGURE 3.16: Lag 1 cross correlation non-stationary features

$$\sigma_K = RMSE = [\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2]^{\frac{1}{2}} \tag{3.6}$$

Optimal model fitting and parameter estimation in the research apply the minimization of the RMSE function evaluated on the test dataset. A standard way to measure this out-of-sample performance is by utilizing some form of data splitting; that is, divide the time series data into two segmentsone for model fitting and the other for performance testing. Sometimes data splitting is called cross-validation. Then, using cross-validation, we will evaluate our chosen loss function for the given model parameters, calculate the gradient, adjust the model parameters, and so on, eventually descending to the global minimum.

You may be asking how to do cross-validation for time series because time series have this temporal structure and one cannot randomly mix values in a fold while preserving this structure. With randomization, all time dependencies between observations will be lost. This is why we will have to use a more tricky approach in optimizing the model parameters. The idea is rather simple – we train our model on a small segment of the time series from the beginning until some t , make predictions for the next t+n steps, and calculate an error. Then, we expand our training sample to $t + n$ value, make predictions from $t + n$ until $t + 2n$, and continue moving our test segment of the time series until we hit the last available observation. As a result, we have as many folds as n will fit between the initial training sample and the last observation.[16].

(picture of cross-validation from https://www.kaggle.com/kashnitsky/topic-9-part-1-time-series-analysis-in-python)

Another statistical metric for model evaluation is the AIC. In statistical time series modelling, the order of the stochastic models is selected chosen using the Akaike Information Criterion (AIC)

$$AIC = -2 * log - likelihood + 2 * numberofparameters \tag{3.7}$$

or

$$AIC = \ln(\frac{\sum_{t=1}^{T} e_t^2}{T} + \frac{2p}{T}) \tag{3.8}$$

The method mle used in the fitting procedure above is based on maximising the likelihood function (the probability of obtaining the data given the model) with respect to the unknown parameters. The model with the smallest AIC is selected as the best fitting statistical model. This criteria penalizes the sum of squared residuals for including additional parameters in the model.

Chapter 4

Time series statistical modeling

4.1 Introduction

The current chapter focuses on using statistical methods for forecasting. The next chapter extend the statistical approach to machine learning methods for forecasting specifically looking into deep learning models. The explicit time series modeling approach to forecasting that we have chosen to emphasize is the auto-regressive integrated moving average (ARIMA) model. Also, we demostrate how ARIMA models can be extended using exogenous variables. Later in the chapter, multidimensional vector time series try to identify and quantify the dynamic association between Bitcoin price and blockchain network features observed at the same time frequency interval.

4.2 ARIMA Models

Exploratory data analysis covered on the previous chapter wasn't the ultimate goal of our research at all. Instead, this chapter focus on a popular statistical modelling techniques for time series analysis, ARIMA models.

When we fit mathematical models to time series data, we refer to the discrepancies between the fitted values, calculated from the model, and the data, as a residual error series. If our model encapsulates most of the deterministic features of the time series, our residual error series should appear to be a realisation of independent random variables from some probability distribution. However, we often find that there is some structure in the residual error series, such as consecutive errors being positively correlated, which we can use to improve our forecasts and make our simulations more realistic. [21].

Arima models who stands for auto-regressive moving average models includes a broad suite of models that model future time series values based on some linear combination of past values. In time series setting, the assumption of independent and identically distributed error (shocks) assumption is quite often violated. Under these circumstances, these class of models take advantages of the serial correlation or dependence at different lags to model the stochastic behaviour of the time series. In these scenarios, auto-regressive models can be very useful as these models adjust immediately using the prior lag values by taking advantage of inherent serial correlation between observations.

This chapter introduces forecasting concepts using auto-regressive models. The auto-regressive model includes auto-regressive terms or moving average terms. Based on the components used, there are multiple approaches that can be used in time series forecasting such as moving average (MA), auto-regressive moving average (ARMA), and auto-regressive integrated moving average (ARIMA). So what exactly is an ARIMA model? ARIMA, short for Auto Regressive Integrated Moving Average is actually a class of models that explains a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. Thus, the ARIMA model combines the power of AR and MA components together. An ARMA(p, q) time series forecasting model incorporates the p^th order AR and q^th order MA model, respectively.

$$x_t = \alpha + \phi X_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - ... - \theta_2 \epsilon_{t-2} + \epsilon_t \quad (4.1)$$

As explored in chapter 3, many variable in our dataset are not stationary because of seasonal effect or trends. In particular, random walks, which characterise many types of series, are non-stationary but can be transformed to a stationary series by first-order differencing. ARIMA, also known as the Box-Jenkins model, is a generalization of the ARMA model by including integrated components. The integrated components are useful when data has non-stationarity, and the integrated part of ARIMA helps in reducing the non-stationarity.

The ARIMA applies differencing on time series one or more times to remove non-stationarity effect. The ARIMA(p, d, q) represent the order for AR, MA, and differencing components. The major difference between ARMA and ARIMA models is the d component, which updates the series on which forecasting model is built. The d component aims to detrend the signal to make it stationary and ARMA model can be

applied to the de-trended dataset. [10]. Again, the assumption of time series stationarity is mandatory to fit those class of statistical models. To summarize, an ARIMA stochastic process can be identified by 3 terms:

- p is the order of the AR term
- q is the order of the MA term
- d is the number of differencing required to make the time series stationary:

In the previous chapter, the concept of differentiation to make a time series stationary was extensively covered. Moving average and auto-regressive stochastic models which were theoretically covered in chapter 2, here are defined in their practical modelling setting. The auto-regressive modelling approach regress the time series on its own lag terms. The AR models are very good in capturing trends as the next time values are predicted based on the prior time values. Thus, AR models are very useful in situations where the next forecasted value is a function of the previous time period. The p_{th} order AR model is denoted by AR(p): that stand for how many lagged term fit into the stochastic model. The moving average models use dependency between residual errors to forecast values in the next time period. The moving average time series model with q_{th} order is represented as MA(q) for how many forecast auto-regressive lagged errors to consider in the model. Both AR and MA order of the ARIMA model can identified looking at the auto-correlation-function and the partial-autocorrelation function. The ACF plot is very helpful in identifying the MA model and its appropriate order as it cuts off after lag q. In real life applications, however, the sample ACF, r (k), will not necessarily be equal to zero after lag q. It is expected to become very small in absolute value after lag q. The x-axis gives the lag (k) and the y-axis gives the autocorrelation (r_k) at each lag. If $\rho_k = 0$, the sampling distribution of r_k is approximately normal, with a mean of $-1/n$ and a variance of $1/n$, the dotted line on the correlogram are drawn at

$$\frac{1}{n} \pm \frac{2}{\sqrt{n}} \tag{4.2}$$

If r_k falls outside these lines, we have evidence against the null hypothesis that $\rho_k = 0$ at the 5% level.

A partial autocorrelation is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed. [3] The PACF at lag k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags. For the PACF of a MA, we would expect the plot to show a strong relationship to the q lag and a trailing off of correlation from the lag onwards.

We know that the ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information. This means we would expect the ACF for the AR(k) time series to be strong to a lag of k and the inertia of that relationship would carry on to subsequent lag values, trailing off at some point as the effect was weakened. While the ACF graph of an auto-regressive model decreases exponentially, the partial autocorrelation of an AR(k) process will be zero for all lags greater than order p. In general, the partial autocorrelation at lag k is the kth coefficient of a fitted AR(k) model; if the underlying process is AR(p), then the coefficients ρ_k will be zero for all k ¿ p. Thus, an AR(p) process has a correlogram of partial auto-correlations that is zero after lag p. Hence, a plot of the estimated partial autocorrelations can be useful when determining the order of a suitable AR process for a time series. [21] The ACF and PACF of an ARMA process are determined by the AR and MA components, respectively. It can therefore be shown that the ACF and PACF of an ARMA(p, q) both exhibit exponential decay and/or damped sinusoid patterns, which makes the identification of the order of the ARMA(p, q) model relatively more difficult.

The ARIMA process can be extended to include seasonal terms, giving a non-stationary seasonal ARIMA (SARIMA) process. Seasonal ARIMA models are powerful tools in the analysis of time series as they are capable of modelling a very wide range of series.

4.3 Time series Model Building

The classical approach for fitting an ARIMA model is to follow the Box-Jenkins Methodology. This is a process that uses time series analysis and diagnostics to discover good parameters for the ARIMA model.

- Plot the sample ACF and PACF of the differenced series
- Specify and fit an ARIMA/ARMA/AR/MA model
- Check goodness of fit
- Generate forecasts

Model identification efforts should start with preliminary efforts in understanding the type of process from which the data is coming and how it is collected. As discussed earlier, both the ACF and PACF can be used a guide for identifying AR or MA model components.

There are several methods such as the methods of moments, maximum likelihood, and least squares that can be employed to estimate the parameters in the tentatively identified model. The best fitting ARMA(p, q) model can be chosen using the smallest AIC either by trying a range of combinations of p and q in the arima function or using a for loop with upper bounds on p and q. In each step of the for loop, the AIC of the fitted model is compared with the currently stored smallest value. If the model is found to be an improvement (i.e., has a smaller AIC value), then the new value and model are stored. To start with, best aic is initialised to infinity (Inf). After the loop is complete, the best model can be found in best order. The order of the process is choosen using the Akaike Information Criterion. The model with the smallest AIC is selected as the best fitting ARIMA models. After a tentative model has been fit to the data, we must examine its adequacy and, if necessary, suggest potential improvements. A satisfactory time series model fit is found only when the residuals don't show any significant temporal structure and looks white noise. White noise is the first Time Series Model we need to understand. By definition a time series that is a white noise process has serially uncorrelated errors and the expected mean of those errors is equal to zero. Another description for serially uncorrelated errors is, independent and identically distributed (i.i.d.). This is important because, if our TSM is appropriate and successful at capturing the underlying process, the residuals of our model will be i.i.d. and resemble a white noise process. Therefore part of TSA is literally trying to fit a model to the time series such that the residual series is indistinguishable from white noise.[27] Thus, if a model has accounted for all the serial correlation in the data, the residual series would be serially uncorrelated, so that the autocorrelation should not differ significantly from zero for all lags greater than one. The process is repeated until either a desirable level of fit is achieved on the in-sample or out-of-sample observations (e.g. training or test datasets).

To start, I decided to see if I could fit an ARIMA to the Bitcoin closing prices. If I found some sort of AR(p) or MA(q) signature in the data, I was then going to attempt to introduce an additional set of exogenous variables (ARIMAX) to see if I could improve upon the fit/predictions from ARIMA. However, I quickly found that ARIMA alone would not be workable in this context, likely due to the many different factors that drive the cryptocurrency market (making a linear fit to autoregressive/error terms insufficient). From a visual inspection, the Bitcoin daily closing was clearly generated by a random walk process. It observed value at time t are totally dependent on the value at lag $t-1$. Taking a Logarithmic transformation to regularize increasing time series variance and applying the first difference produced a stationary time series, while also it contributed to eliminate both the exponential decay and high spike on the ACF and PACF. Nevertheless, we attempt to fit an ARIMA model with parameters' order defined by the smallest AIC on the training data.

	parameters	aic
0	(3, 1, 2)	-7641.891641
1	(2, 1, 2)	-7626.861060
2	(3, 1, 1)	-7626.483859
3	(0, 1, 1)	-7619.459431
4	(1, 1, 0)	-7619.458131
5	(2, 1, 0)	-7617.823133
6	(0, 1, 2)	-7617.805149
7	(3, 1, 0)	-7615.826059
8	(2, 1, 1)	-7615.823718
9	(0, 1, 3)	-7615.813093
10	(1, 0, 0)	-7612.633598
11	(2, 0, 0)	-7610.683276
12	(3, 0, 0)	-7609.047902
13	(0, 0, 2)	8977.200384

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	2213
Model:	ARIMA(3, 1, 2)	Log Likelihood	3826.946
Method:	mle	S.D. of innovations	0.043
Date:	Tue, 04 Jun 2019	AIC	-7641.892
Time:	15:05:54	BIC	-7607.679
Sample:	04-29-2013 - 05-20-2019	HQIC	-7629.394

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.D.Close	0.7340	0.031	23.831	0.000	0.674	0.794
ar.L2.D.Close	-0.9730	0.022	-44.655	0.000	-1.016	-0.930
ar.L3.D.Close	-0.0066	0.023	-0.288	0.774	-0.051	0.038
ma.L1.D.Close	-0.7344	0.022	-32.880	0.000	-0.778	-0.691
ma.L2.D.Close	0.9527	0.017	55.808	0.000	0.919	0.986

FIGURE 4.1: ARIMA(3,1,2) model summary

For a combination of ARIMA parameters combination from 0 to 3, the best chosen model according to AIC was an ARIMA (3,1,2). This model fitting produced used the all time range to estimate model parameters.

The summary attribute that results from the output of SARIMAX returns a significant amount of information, but we'll focus our attention on the table of coefficients. The coef column shows the weight (i.e. importance) of each feature and how each one impacts the time series. The P¿—z— column informs us of the significance of each feature weight. For the third auto-regressive lag of the model the p-value is above the test statistics, so we cannot reject the null hypothesis that the value is different from 0. This parameters might be the result of model overfitting that AIC statistics wasn't able to detect.

ARIMA Model Results

Dep. Variable:	D.Close	No. Observations:	2213
Model:	ARIMA(2, 1, 2)	Log Likelihood	3818.431
Method:	mle	S.D. of innovations	0.043
Date:	Tue, 04 Jun 2019	AIC	-7626.861
Time:	15:08:07	BIC	-7598.351
Sample:	04-29-2013 - 05-20-2019	HQIC	-7616.446

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.D.Close	1.7564	0.211	8.323	0.000	1.343	2.170
ar.L2.D.Close	-0.7838	0.213	-3.685	0.000	-1.201	-0.367
ma.L1.D.Close	-1.7674	0.205	-8.625	0.000	-2.169	-1.366
ma.L2.D.Close	0.8029	0.203	3.948	0.000	0.404	1.202

FIGURE 4.2: ARIMA(2,1,2) model summary

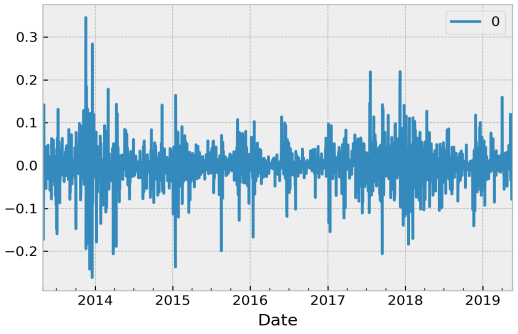

FIGURE 4.3: Arima (2,1,2) residual plot

Quite better result in terms of model summary was obtain from the second best model on the table below. Here, each weight has a p-value lower or close to 0.05, so it is reasonable to retain all of them in our model.

Before evaluating our model prediction performance, it is a good practice to assess model residual distribution and serial correlation structure.

Our primary concern is to ensure that the residuals of our model are uncorrelated and normally distributed with zero-mean. If the seasonal ARIMA model does not satisfy these properties, it is a good indication that it can be further improved.[36] The ACF and PACF are showing no significant autocorrelation. The QQ and Probability Plots show the residuals are approximately normal with heavy tails. However, this model's residuals do not look like white noise. Look at the areas of obvious conditional heteroskedasticity (conditional volatility) that the model has not captured.

We have obtained a model for our time series that can now be used to produce forecasts. We start by comparing predicted values to real values of the time series, which will help us understand the accuracy of our forecasts. Best model performance was evaluated on the validation test of 50 days before making forecast out-of-sample. The training data includes Bitcoin closing price observation from 2013-04-28 to the end of March 2019. The

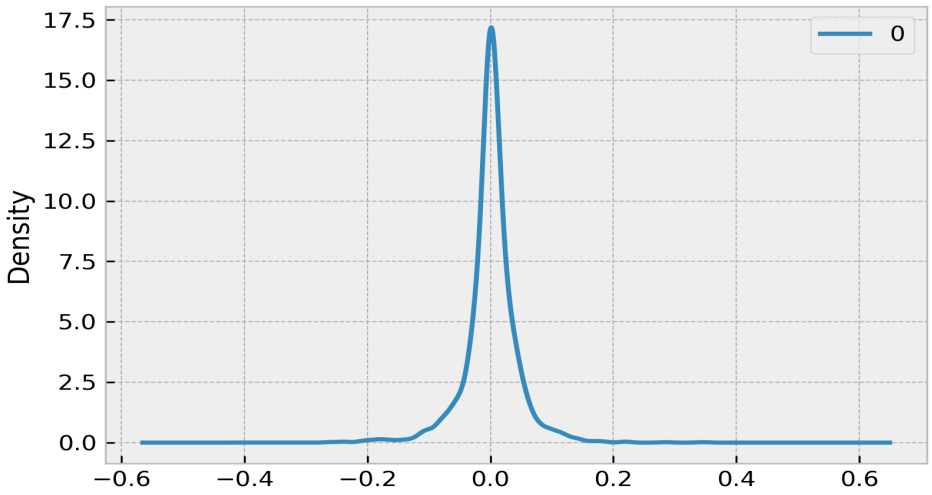

FIGURE 4.4: Arima (2,1,2) residual distribution

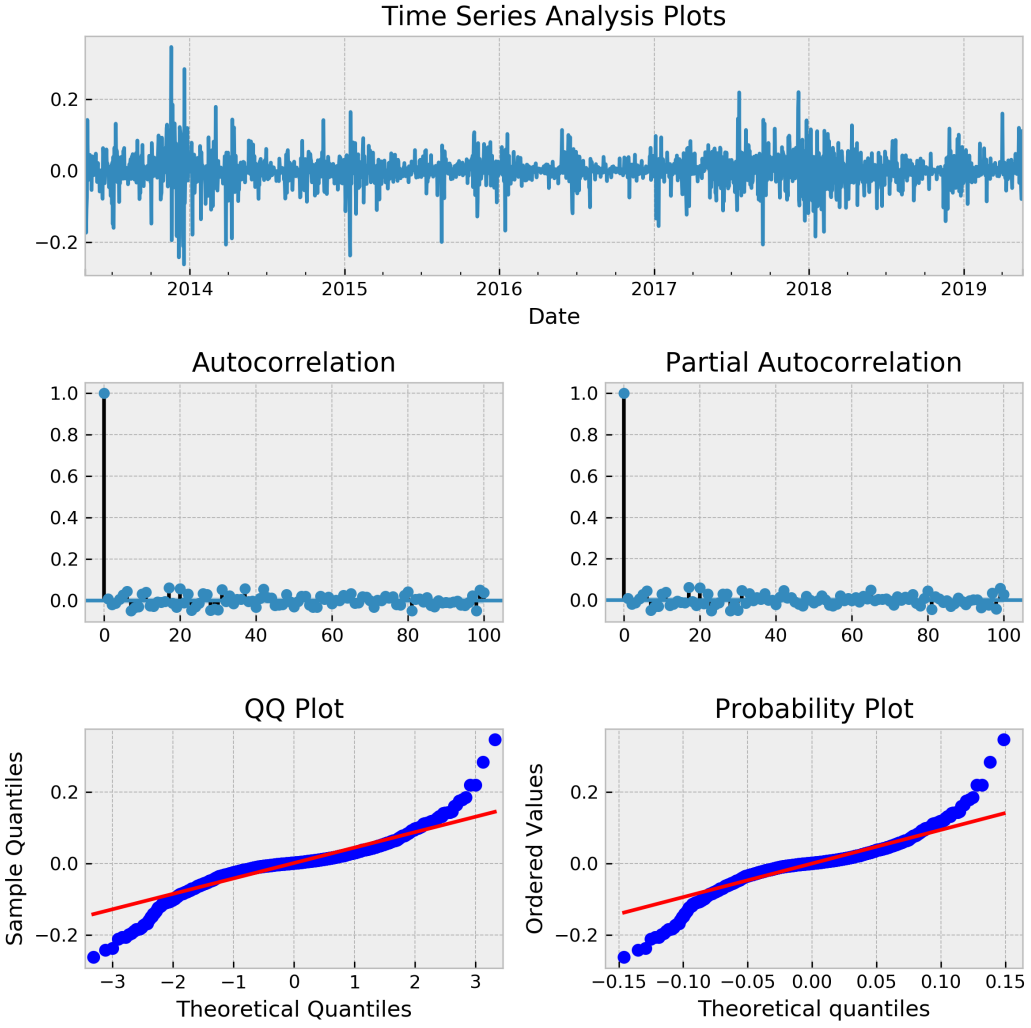

FIGURE 4.5: Arima (2,1,2) residual correlation structure

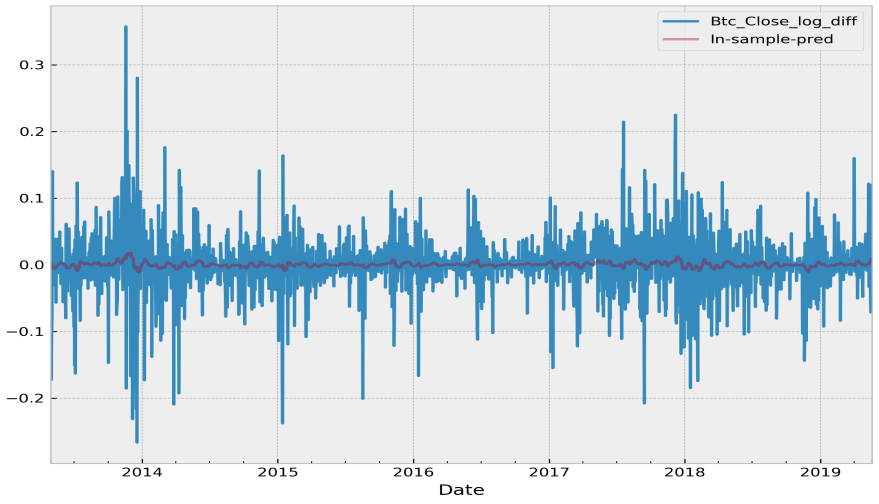

FIGURE 4.6: Arima (2,1,2) In-sample fitting values of first Diff Btc

last two month available on our dataset, April and May 2019 are used as validation test to evaluate the accuracy on the model on data points it didn't have chance to observe. Additionally, I measured a one-step ahead forecast error for each test observation rolling the historical training window.

In-sample prediction of the best model selected clearly overfit the time series. The best prediction of the model use the replicate the value observed at $t-1$. The Figure show the fitted values from February 2019 to the end of our dataset. In fact, when prediction are compared to the lag 1 differenced data, there is a clear modelling difficulties to capture market shocks.

Ultimately, the evaluation of the model on the last 50 days produced a RMSE of 1983.1622 and a MAPE of 27.44 %. Our conclusion is that the univariate approach of the ARIMA models using historical data doesn't produce a good result.

4.4 Vector Multivariate Time Series analysis

One of the major use cases of time series analysis is time series forecasting. In univariate time series forecasting, the statistical relationship is unidirectional in that the forecast variable is influenced by its own lags or the lags of other predictors variables (or features). In dealing with economic variables, often the value of one variable is not only related to its predecessors in time but, in addition, it depends on past values of other variables. [18] Those features are called exogenous variables in multidimensional time series analysis. The observation of their state collected over the same interval of the endogenous variables

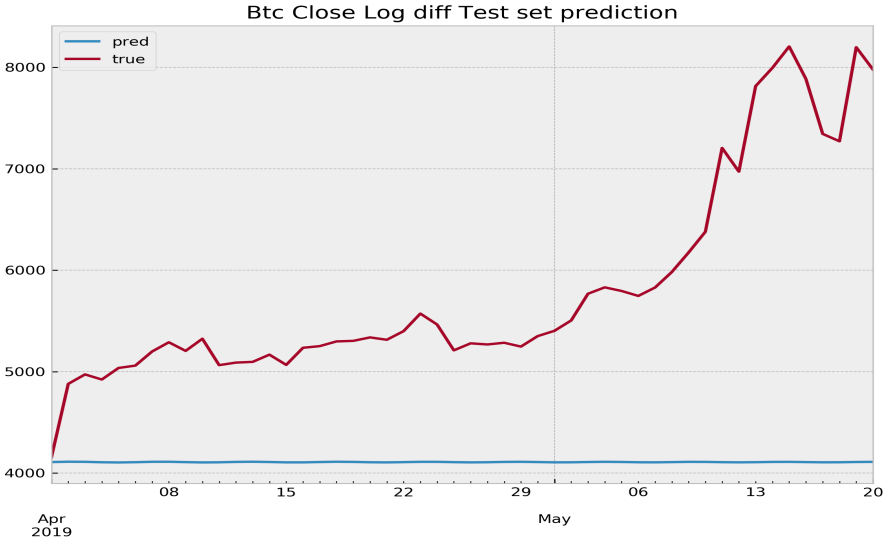

FIGURE 4.7: Arima (2,1,2) Validation Test: Prediction vs Actual values

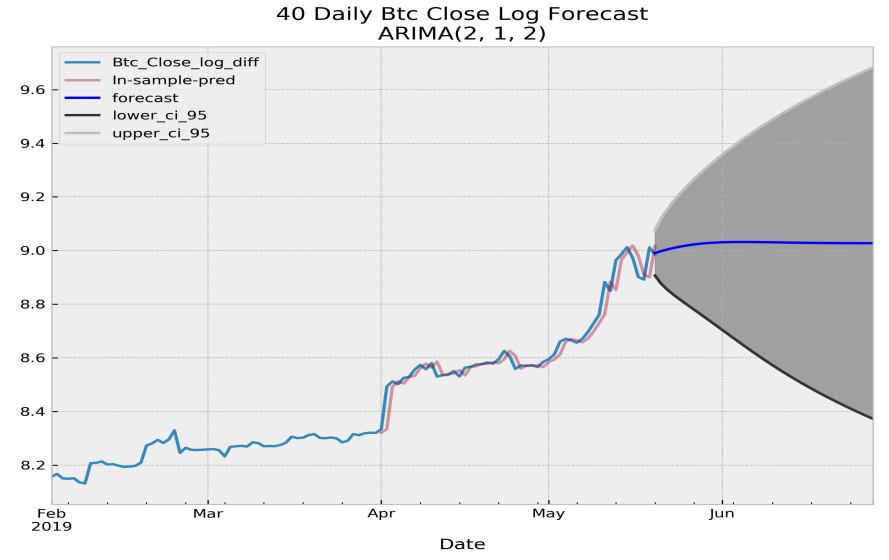

FIGURE 4.8: Arima (2,1,2) Out of sample forecasts

(dependent) can help to improve the forecast accuracy of the model. In other words, denoting the related variables by $y_{1t}, y_{2t}, ..., y_{Kt}$, the forecast of $y_{1,T+h}$ at the end of period T may be of the form:

$$y_{1,\hat{T}+h} = f_1(y_{1,T}, y_{2,T}, ..., y_{K,T}) \tag{4.3}$$

A set of time series $y_{kt}, k = 1, ..., K, t = 1, ..., T$, is called a multiple time series and the previous formula expresses the forecast $\hat{y}_{kT+h}$ as a function of a multiple time series.

In analogy with the univariate case, it is one major objective of multiple time series analysis to determine suitable functions $f_1, ..., f_K$ that may be used to obtain forecasts with good properties for the variables of the system. [18] Thus, obtaining insight into the dynamic structure of a system is a further objective of multiple time series analysis. A time series will be assumed to be generated by a stochastic process.

In this section, I will introduce one of the most commonly used methods for multivariate time series forecasting Vector Auto Regression (VAR). In a VAR model, each variable is a linear function of the past values of itself and the past values of all the other variables.[33] If you want to use VAR models, there are some basic steps good for practice to follow in order to fit multivariate time series

We have already covered in the previous chapter time series visualization and exploratory data analysis. The dependent variable which is here the bitcoin market price as well as all other Bitcoin blockchain network features have been fully investigated to meet the stationarity condition, weak stationarity in this particular case. Stationarity is a mandatory requirement to fit VAR models. In case of coointegration among non-stationary time series, it should be used a different kind of model called vector error correction model that I didn't cover in this research. This kind of model required the series to be stationary, if the series are not stationary, the individual series have to be transformed to be stationary. In fact, it is necessary to difference all variables first as estimation of a model with non-stationary errors is not consistent and can lead to spurious regression. Basically, we applied the first difference to the log of each series to make the series stationary. For all those Blockchain transaction variables showing high serial correlation with their previous week observation, I subtracted the shifted 7th lags on to of the previous transformation. The visualization of the series after transformation looks stationary and the unit root statistical test p-value is far below the 5% threshold. I also looked at the autocorrelation and partial autocorrelation function that now seems a realization of a white noise process. At this point, the dataset was split into a train and validation set estimate and compute the error of our forecasts. It is a common practice to use a rolling window of training and test set in time series analysis. Although, I build the validation set just leaving the last 50 observation out of the training sample. Before showing the results of the fitted VAR model, I introduce some high level mathematical notation to explain Var model equations. The following notation represent the simplest VAR model with two variable equations, but it can easily extended in n multidimensional time series.

Genaral notation for a two series VAR(1) model:

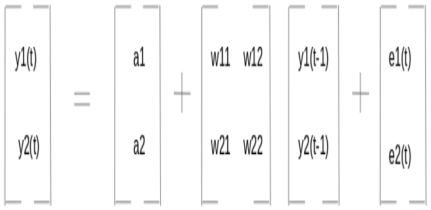

FIGURE 4.9: Vector equation 2 time series VAR(1)

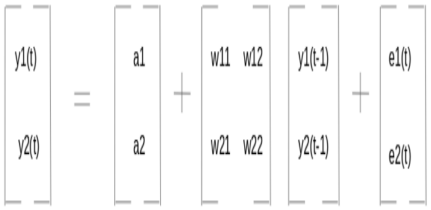

FIGURE 4.10: VAR(p) process with variables $y_1, y_2 ... y_k$.

$$y_{1t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + u_{1,t}$$
$$y_{2t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + u_{2,t}$$

where

- ϵ_{1t} and ϵ_{2t} are white noise processes that may be contemporaneously correlated
- $\phi_{ii,l}$ captures the effect of the l^th lag of the series y_i on itself
- $\phi_{ij,l}$ captures the effect of the l^th lag of the series y_j on y_i

The variance covariance matrix is

$$\begin{Bmatrix} \text{var}(y_1) & \text{cov}(y_1, y_2) \\ \text{cov}(y_2, y_1) & \text{var}(y_2) \end{Bmatrix}$$

From the above equations, it is clear that each variable is using the past values of every variable to make the predictions. Unlike AR, VAR is able to understand and use the relationship between several variables. This is useful for describing the dynamic behavior of the data and also provides better forecasting results

Fitting a VAR model with all the explanatory variable we found that many of the coefficient estimates were not significantly different from zero. This comply with the Granger-casuality test carried out in chapter 3. The best model selected according

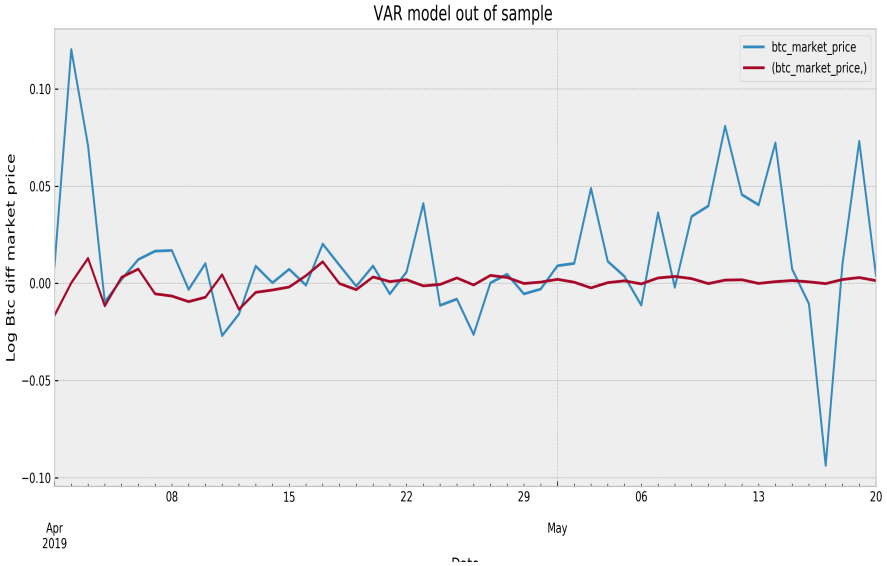

FIGURE 4.11: VAR(13) Out of sample forecast - Log first diff Btc market price

to the AIC had a max-lag of 13 lagged observation for each variables. Here is the performance on the validation. Surprisingly, on the first weeks of the test set, we can notice a quite good performance in terms of positive or negative one-step ahead forecast market return direction. The MAPE for the model, though is too high. Thus, we won't consider this a good prediction model for the Bitcoin market price.

The ultimate model we consider in this statistical time series modeling section is an extension of the ARIMA model studied in the univariate case with exogenous covariates into play. As in the univariate case, multivariate or vector ARIMA models can often be successfully used in forecasting. Many concepts covered in the last section will be directly applicable in the multivariate case as well. Suppose that

$$y_t = a + \phi_1 y_{t-1} + ... + \phi_p y_{t-p} + \omega_t + \theta_1 \omega_{t-1} + ... + \theta_q \omega_{t-q} + \sum_{m=1}^{M} \beta_m X_{m,t} \tag{4.4}$$

where

$$\omega_t \sim N(0, \sigma^2) \forall t \tag{4.5}$$

Suppose that the vector time series $Y_t = (y_{1t}, y_{2t}, ...y_{mt})$ consists of m univariate time series. Then Y_t with finite first and second order moments is said to be weakly stationary

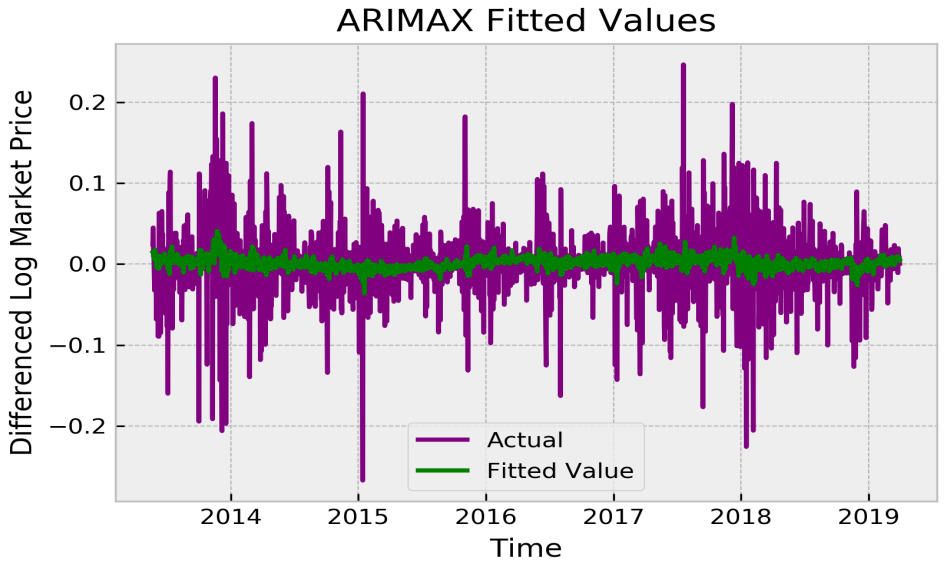

FIGURE 4.12: ARIMAX (6,1,1) Fitted values

if:

$$E(Y_t) = E(Y_{t+s}) = \mu, constant for all s$$
$$Cov(Y_t) = E[(Y_t - \mu)(Y_t - \mu)^{']=\Gamma(0)}$$
$$Cov(Y_t, Y_{t+s}) = \Gamma(s) depends only on s$$

The stationary vector time series can be represented with a vector ARMA model given by:

$$\Phi(B)Y_t = \delta + \Theta(B)\epsilon_t \tag{4.6}$$

As in the univariate case if non-stationary is present, through an appropriate degree of differencing a stationary vector time series may be achieved. Hence the vector ARIMA model can be represented as:

$$\Phi(B)D(B)Y_t = \delta + \Theta(B)\epsilon_t \tag{4.7}$$

Again, a search grid of parameters combination to find the model with the lowest AIC selected an ARIMAX (6,1,1). Results turned out to increase prediction performance by a great extend. Using the following model tested on a left out sample of 50 days reselted in a mean squared error of 19.20% and a RMSE of 1320.44 on the original BTC market price scale.

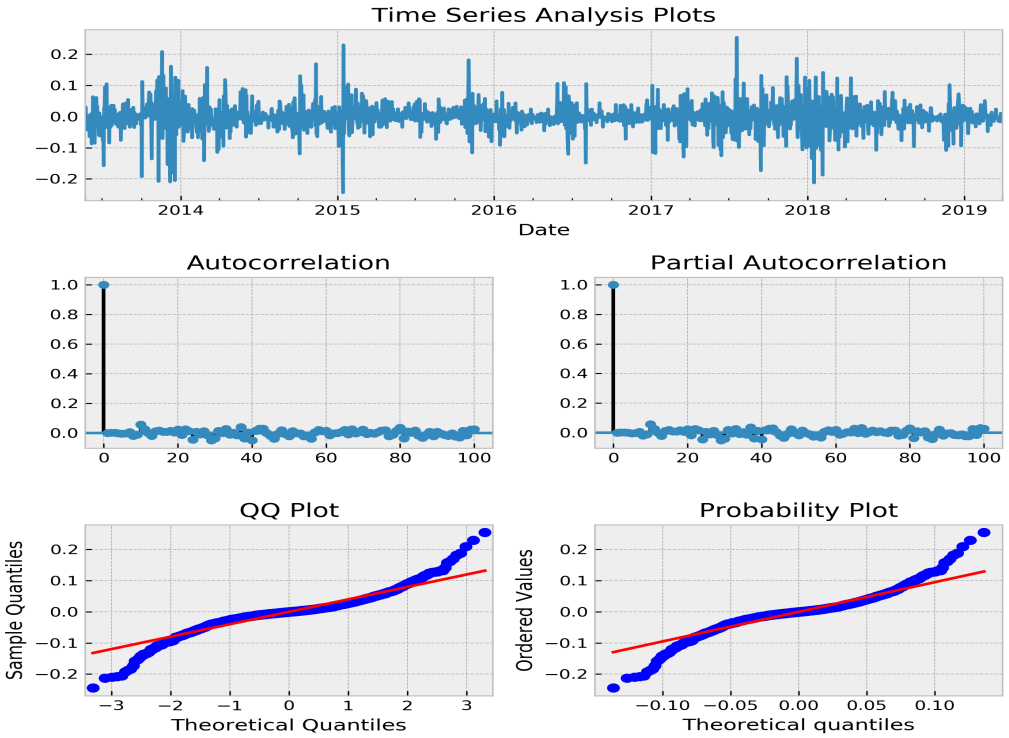

FIGURE 4.13: ARIMAX (6,1,1) residual analysis

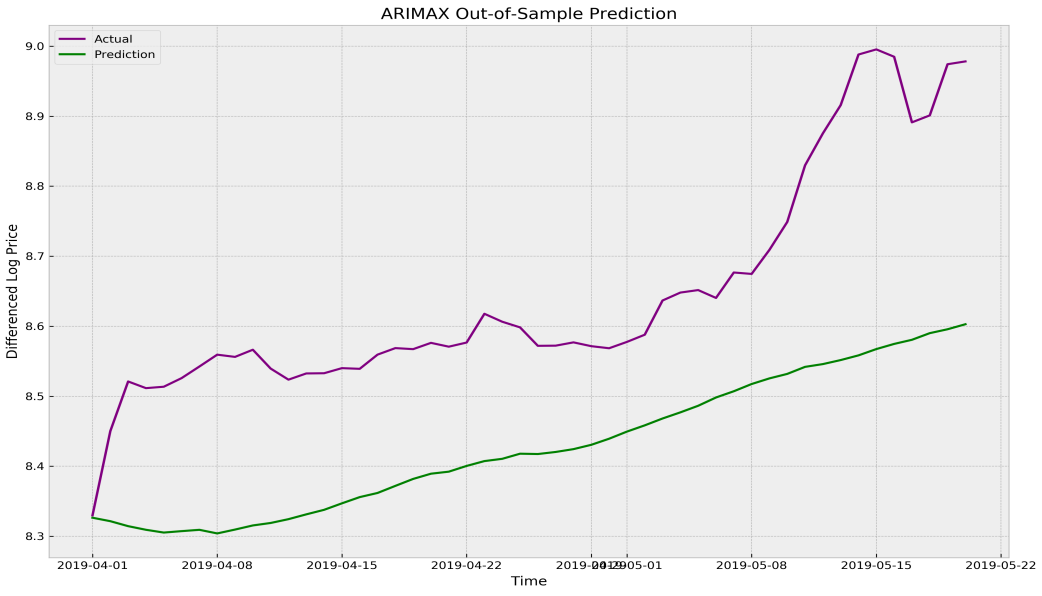

FIGURE 4.14: ARIMA (6,1,1) Validation set performance

Chapter 5

Deep Learning algorithms for Time series forecasting

5.1 Why Deep Learning

So far in this book, we have described traditional statistical methods for time series analysis. In the preceding chapters, we have discussed several methods to forecast the series at a future point in time from observations taken in the past. The idea underlying the linear model can be generalized that the objective of time series forecasting is to develop a function f that predicts x_t in terms of the observations at previous p points of time:[10]

$$x_t = f(x_t{-}_1, x_t{-}_2, ..., x_t{-}_p) \tag{5.1}$$

In this chapter, we will explore Artificial neural network and Deep learning models architecture to estimate function f. After a brief background introduction to neural network structure to model supervised learning task, the rest of the research section is focused on a particular structure of neural network designed to handle sequential data structure: Long-Short-Term-Memory Recurrent Neural network.

The last few years has witnessed an incredible reassurance of neural network algorithms made possible to the availability of abundant training data from digital media and cheaper access to GPU-driven parallel computing. These factors have enabled training neural networks having hundreds of thousands, and in some cases, millions of parameters.[10]

This success is attributed to their ability to learn hierarchical representations, unlike traditional methods that rely upon hand-engineered features. Over the past several years, storage has become more affordable, datasets have grown far larger, and the field of parallel computing has advanced considerably. Deep learning methods, in particular those based on deep belief networks (DNNs), which are greedily built by stacking restricted Boltzmann machines, indicative of the many hidden layers used in these models have demonstrated record-setting results on many important applications [39]. These new developments have also been applied to areas where traditionally statistical machine learning has dominated. One such area is time series forecasting. All the predictive time series models for Bitcoin market price shown in this section are implemented using the Keras API for deep learning. Keras is a high-level API that allows defining different neural network architectures and training them using various kind of gradient-based optimizers.

Before diving into the technicality of artificial neural network structure, I would like to give some background context of where and how Deep learning is a subject of intensive hype these days.

Deep learning is part and fit within the context of Artificial Intelligence and Machine Learning witch constitute a special sub-field. Artificial intelligence can be concisely defined as the effort to automate intellectual tasks normally performed by humans. [4] As such, AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that dont involve any learning.

Our life is full of exceptions, diversity, bias and any sample population referring to whichever phenomenon will have intrinsic variance in it. In this high uncertain context, handcrafted programs will have hard time to define an exhaustive set of explicit rules for manipulating this knowledge. Although symbolic AI proved suitable to solve well-defined, logical problems, such as playing chess, it turned out to be intractable to figure out explicit rules for solving more complex, fuzzy problems, such as image classification, speech recognition, and language translation. A new approach arose to take symbolic AIs place: machine learning. [4]

With machine learning, humans input data as well as the outcome expected from the data, and out come the rules. These rules can then be applied to new unseen data to produce original answers. The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

Machine learning draws on concepts and results from many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory. A machine-learning system is trained rather

than explicitly programmed.

The goal of machine learning is to learn and extract information not obviously known from data to improve the performance on some tasks learning from experience (exposure to training observations). Yet, machine learning is about learning a representation of data. Learning in the field of data science is more precisely expressed as:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.[22]

A machine-learning model transforms its input data into meaningful outputs, a process that is learned from exposure to known examples of inputs and outputs. Therefore, the central problem in machine learning and deep learning is to meaningfully transform data: in other words, to learn useful representations of the input data at handrepresentations that get us closer to the expected output. [4]

The central problem in machine learning and deep learning is to transform input data into meaningful output, a process that is "learned" from exposure to known inputs and outputs: in other words, to learn useful representations of the input data at handrepresentations that get us closer to the expected output values.

All machine learning algorithms consist of some form of automatic learning to turn data into useful information to solve a given task. This representation learning is also defined as searching through a predefined set of operations or space of possibilities using guidance from a feedback signal, called hypothesis space.

Deep learning is a sub-field of machine learning. A mathematical framework for learning representation from data composed by many stacked layers of increasingly meaningful representation. The term "deep" stand for the idea of successive layers of representation. While many machine learning algorithms are designed to learn just only one or two layers of representation (shallow representation), in deep learning these layers representation are learned via models called neural networks structured in literal layers stacked on top of each other.[4]. Metaphorically, deep learning consist of a bunch successive multistage information distillation operation, where information are increasingly purified. In the next section, deep learning models are broken down into their constituent part in order to understand how they work. The question could be rephrased into: how the input-output mapping learning happen inside a deep neural network?

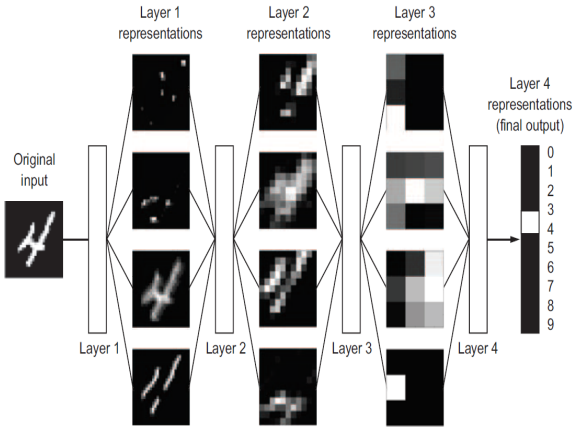

FIGURE 5.1: A deep neural network for digit classification

5.2 Background of Artificial Neural Network

Images are more powerful that thousand of words someone once upon a time said. Visual representation of an ANN structure is a great tool for educational purposes. In order to explain how artificial neural network works, I am going to use a really popular example in the field coming from computer vision about hand-written digits.

What do the representations learned by a deep-learning algorithm look like?

As it is clear from the figure, the network gradually transform the original input image into a more informative representation at each layers. The original images are the inputs of the network and the number each image represents are the labels the network use to learn a data representation. A key element of an ANN structure is a measure of whether the algorithm is doing a good job: This is necessary in order to determine the distance between the algorithms current output and its expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call learning. [4]

Once again you might be wondering how this learning take place through each layer. Well, The specification of what a layer does to its input data is stored in the layers weights, which in essence are a bunch of numbers [4]. A neural network learn by means of each set of weights in each layer, that is basically expressed similarly to a parameterization of the learning process. The goal of a deep learning models is to find a set of weights for all layer in the network such that the network will correctly map example inputs to their associated targets. Clearly, manually searching for the the best combination of thousand weights in the network seems a daunting task, especially since as we will see late on, a small change in one weight affect the behaviour of all others.

To automate this task a neural network define a loss function or objective function that measure how far the output or network prediction is from what should be expected for each set of weights. The loss function takes the predictions of the network and the true target (what you wanted the network to output) and computes a distance score, capturing how well the network has done on this specific example.[4] This feedback signal in terms of score help the network to adjust the network's weights a little in the direction that will decrease the loss function. This is the job of the optimizer which implements the backpropagation algorithm.

At the beginning of the process, weights are assigned randomly to the network and its output is far from its ideal state. As the network manage to see new input examples, weights are adjusted a little in the correct direction and the loss function start decreasing. One sequence of this process is called training loop which repeated for a sufficient number of time (epochs) yields weight values that minimize the loss function. A network with a minimal loss is one for which the outputs are as close as they can be to the targets: a trained network[4]

Understanding deep learning requires familiarity with many simple mathematical concepts: tensors (neurons), tensor operations, input layer, hidden layer, output layer, activation function, objective function, optimizers, differentiation, gradient descent, backpropagation, and so on. My goal in this chapter is to provide the mathematical building blocks as well as an intuitive understanding to the fascinating topic of time series forecasting by deep learning. Ill go into detail about every pieces and clarify whats going on behind the scenes.

5.2.1 Anatomy of a neural network

From the introduction chapter, training a neural network boils down on the availability of the following elements:

- Layers, which are stacked into the network
- Input data and corresponding targets
- The Loss function
- The Optimizer

The following picture visualize the interaction of those element: [4]

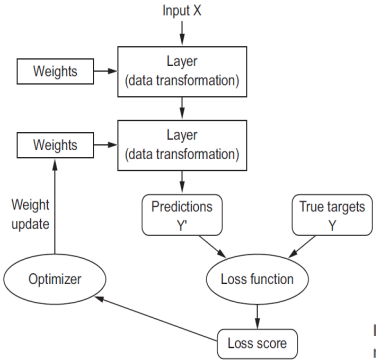

FIGURE 5.2: Relationship between the network, layers, loss function, and optimizer

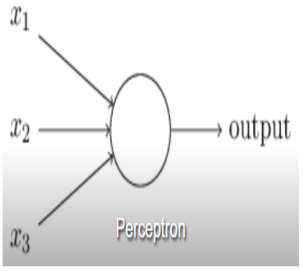

FIGURE 5.3: Perceptron

Let's take a closer look at each elements. The basic block of a neural network is a perceptron. A perceptron can be understood as anything that takes multiple inputs and produces one output.

The above pictures shows how the simple perceptron takes multiple inputs and produce one output. There are several ways of creating an input-output relationship and all of them assume an activation function at each layer.

The Activation Function takes the sum of weighted input $(w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + 1 * b)$ as an argument and return the output of the neuron.

$$a = f(\sum_{n=0}^{N} w_i * x_i + b_i) \tag{5.2}$$

So, the perceptron output is determined by the sum of the dot product between layer's weight and input values. The activation function is mostly used to make a non-linear transformation which allows us to fit nonlinear hypotheses or to estimate the complex functions. There are multiple activation functions, like: Sigmoid, Tanh, ReLu and many other. [28] In programming terms , perceptron are data store in a multidimensional array, called tensor. Tensor are generalization of matrices to an arbitrary number of dimensions in linear algebra, which have 2D tensor. A tensor that contains only one

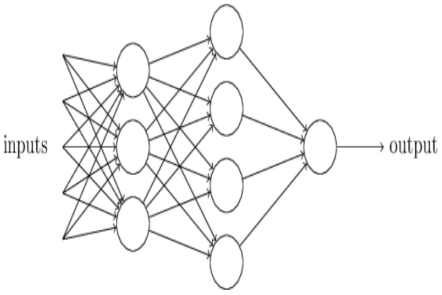

FIGURE 5.4: Multi-layer-Perceptron

number is called a scalar or 0 dimensional [4]. An array of number (1D tensor) is called vector. A 1D tensor is said to have exactly one axis. Much as any computer program can be ultimately reduced to a small set of binary operations on binary inputs (AND, OR, NOR, and so on). All transformations learned by deep neural networks can be reduced to tensor operations (addition, multiplication and so on) applied to tensors of numeric data.[4]

The previous process described to compute the output is known as Forward Propagation. The next step is to estimate the error between the estimated output from the actual one. In neural network term, this error will be propagated back to all layers updating biases and weights in the optimal direction. This step is called "Backpropagation". The weights are updated to minimize the error resulting from each neuron. This one round of forward and back propagation iteration is known as one Epoch training iteration. The simple perceptron model can be generalized to a multi-layer-perceptron model (MPL). An MLP consists of three components: an input layer, a bunch of hidden layers, and an output layer.

In this network, the first column of perceptrons (what we'll call the first layer of perceptrons) is making three very simple decisions, by weighing the input evidence. What about the perceptrons in the second layer? Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making. [24]

Even in a high-dimensional tensor space the network is able to implement a complex geometrical transformation through a long series of simple single steps. The Image above shows just two Hidden layers, but in practice can contain multiple hidden layers. Let's look at the MLP in terms of time series forecasting task.

An input layer represents a vector of regressors, for instance observations from preceding p points in time $[x_{t-1}, x_{t-2}, ..., x_{t}-_{p}$. Input features are fed to the hidden layers that has n neurons, each of which applies a linear transformation and a nonlinear activation to the the input features. The output of a neuron is $g_i = h(w_{ix} + b_i)$, where w_i and b_i are the weights and bias of the linear transformation and h is a nonlinear activation function. The nonlinear activation function enables the neural network to model complex non-linearities of the underlying relations between the regressors and the target variable [10]. Popular activation function are the sigmoid function that squashes any real number to the interval [0,1]:

$$\sigma(z) = \frac{1}{1 + e^{-}z} \tag{5.3}$$

To put it all a little more explicitly, the output of a sigmoid neuron with inputs $x_1, x_2, ,$ weights $w_1, w_2, ,$ and bias b is:

$$\sigma(z) = \frac{1}{1 + e^{\sum_j} w_j * x_j + b_j} \tag{5.4}$$

Another choice of a nonlinear activation function is the tanh function which binds any real number to the interval [-1,1].

$$\sigma(z) = \frac{1 - e^{-}z}{1 + e^{z}} \tag{5.5}$$

The hidden layers of MLP are also called dense, or sometimes fully-connected, layers. Due to the many-to-many connections between the input layer and the first dense layer and between the dense layers themselves, an MLP has an enormous number of trainable weights. In the context of trainig a neural network we haven't discussed yet about the mathematical procedure that happen under the hood during to minimize the error from the loss function. Here, we will look at the most common optimization algorithms implemented by artificial neural network, known as gradient descent.

5.2.2 Gradient-based optimization and Backpropagation

Backpropagation using a gradient descent algorithm can devise a learning algorithm which can automatically tune the weights and biases of a network of artificial neurons without the intervention by a programmer. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks.

The output equation of our neural network is composed of tensors for each layers called weights or trainable parameters. These weights contain the information learned by the network from exposure to training data. First, those weights are filled with random variables (random initiation). Of course, the resulting output would be meaningless (far away from the true target variable). What comes next is to gradually adjust these weights, using a gradient descent based optimization algorithm based on the feedback signal at each epoch. Given an individual weight coefficient in the network, how can you compute whether the coefficient should be increased or decreased, and by how much?

ANN takes advantage of the fact that all operations used in the network are differentiable, and compute the gradient of the loss with regards to the network's coefficients. Since, from calculus the gradient is defined as the direction of the steepest increase of a function, the algorithms moving in the opposite direction is able to decrease the loss function. The concept of derivative in Univariate calculus can be extented to multidimensional variables as well. When the function has just one variable we interpret the derivative as the marginal increase or decrease of y from a small change in x. The absolute value of the derivative (the magnitude of the derivative) tells you how quickly this increase or decrease will happen. The derivative completely describes how $f(x)$ evolves as you change x.

A gradient is the derivative of a tensor operation. Its the generalization of the concept of derivatives to functions of multidimensional inputs: that is, to functions that take tensors as inputs.[4] Take as hypothesis that the first training iteration has weights which we will call W_o. Then the derivative of f in the point W_0 is a tensor $gradient(f)(W0)$ with the same shape as W, where each coefficient gradient$(f)(W0)[i, j]$ indicates the direction and magnitude of the change in loss value you observe when modifying $W0[i, j]$. That tensor $gradient(f)(W0)$ is the gradient of the function $f(W) = loss_value$ in W0. Likewise the univariate case, $gradient(f)(W0)$ can be interpreted as the tensor describing the curvature of $f(W)$ around $W0$, hence with a function $f(W)$ of a tensor, you can reduce $f(W)$ by moving W in the opposite direction from the gradient.

$$W_{i+1} = W_i - step * \alpha(gradient)\frac{\partial L}{\partial W} \tag{5.6}$$

where step or learning rate controls how fast the point descends along the gradient. Gradient descent algorithms work by moving the weights, in iteration i, along their gradient path that express the partial derivative of the loss function L with respect to the weights.

This basic update rule has several variants that impacts the convergence of the algorithm.

- Batch gradient descent Vanilla gradient descent computes the gradient of the cost function to the parameters θ for the entire dataset:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{5.7}$$

  As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory[29]

- Stochastic gradient descent Stochastic gradient descent performs a parameter update for each training example $x^(i)$ and label $y^(i)$.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^(i); y^(i)) \tag{5.8}$$

  Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online[29]

- Mini-batch gradient descent Mini-batch gradient descent performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^(i : i+n); y^(i : i+n)) \tag{5.9}$$

  This way, it a) reduces the variance of the parameter updates, which can lead to more stable convergence.

In deep learning networks the gradient algorithms must be calculated for all weights of the network. In some instances, this might involve a massive computational task. This is exactly where backpropagation algorithm comes in to solve this problem efficiently. In practice a neural network function consists of many operations chained together, each of which has a simple and known derivative. Calculus tells us that a chain of function can be derived applying the chain rule to the computation of the gradient values. During the backward phase we are going on the opposite direction of the Forward phase. In fact Backpropagation starts with the final loss value and works backward from the top layers to the bottom layers, applying the chain rule to compute the contribution that each parameter had in the loss value.[4]

Starting from the loss value from the forward phase the backpropagation algorithm is applied to compute the partial derivations between two nodes connected by an edge. The partial differentiation operator is applied at every node and the partial derivatives

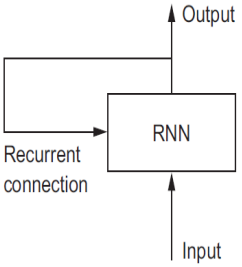

FIGURE 5.5: A single node of RNN

are assigned to the respective edges connecting the downstream node along the computational graph. Following the Chain Rule, the partial derivative $\frac{\nabla L}{\nabla w}$ is computed by multiplying the partial derivatives on all edges connecting the weight node and loss node. If multiple paths exist between a weight node and loss node, the partial derivatives along each path are added to get the total partial derivative of the loss with respect to the weight. [10] The next step is to update the weights using the gradient descent algorithm. The process of iterative weight updates is repeated multiple times.

5.3 Deep Learning for Time series Data: Long Short Term Memory Recurrent Neural Network

So far we have studied how neural networks work and how they are able to fit any representation of data. A key assumption of neural network models is that data instances passed to the models are uncorrelated and independent variables. One problem with this kind of model is that it does not implicitly consider the sequential nature of the time series data where observations have correlation with each other [10].

The correlation in a time series can also be interpreted as the memory that the series carries over itself. In this section, we will discuss recurrent neural networks (RNNs) that are appropriate to fit sequential data. [10]

Humans doesn't start their thinking from scratch every time they need to process new information, but they leverage previous knowledge and the most current state information to take action. Similarly, this is what a recurrent neural network is capable of doing. Past information are allowed to have some persistence in the future. RNN are network with loops in them, allowing information to persist [2].

It processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what is has seen so far [4]. As clear in the figure of a RNN unrolled over time, it takes as input a sequence of vectors having $\hat{x}_t$ timesteps.

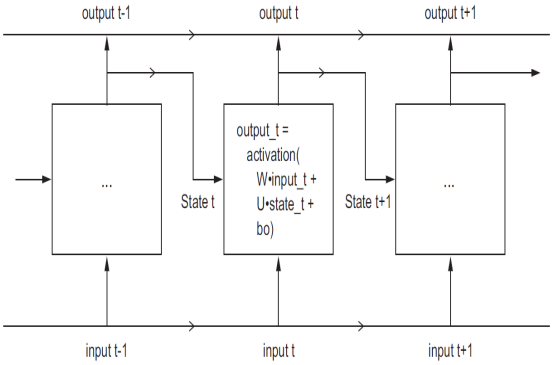

FIGURE 5.6: Unrolled RNN

Next, it loops over timesteps considering its current state at time t and the input at t. The state for the next step become this previous output. This internal state of each timestep carries the memory of the series [10].

For example, this RNN can be used to develop a time series forecasting model where the input series $[x_{t-1}, ..., x_{t-p}]$ is fed to the RNN and the output from the last timestep is the prediction. In computational terms, the transformation of the input and state into an output will be parameterized by two matrices: W and U and a bias vector.

The power of Deep learning comes out when we are stacking many layers on top of each other. In case of RNN, the training of weights in the network is paramount. Successful application of recurrent neural network in language processing, time series forecasting, text generation, image caption used a slight different special variant called Long-short-term-memory (LSTM) which works much better that the standard version.

The difficulty of training a simple RNN stems from the vanishing and exploding gradient that gives erratic results when the network has many hidden layers. As a results, Vanilla RNN have difficulties in learning long-range dependencies. The vanishing or exploding gradient problem arise during the computation of the Backpropagation Through Time (BPTT), a variation of the Backpropagation seen earlier in the scenario of RNN. The problem of vanishing gradients in long-range RNNs is due to the multiplicative terms in the BPTT gradient computations.

The Long Short-Term Memory (LSTM) algorithm was developed by Hochreiter and Schmidhuber in 1997 and it was the culmination of their research on the vanishing gradient problem. Also, in theory a RNNs are absolutely capable of handling such long-term dependencies., but as the gaps between observation input from series grows, RNNs become unable to learn to connect the information [2]. This variants is specifically designed to tackle long term dependencies problem. All recurrent neural network have the form of a chain of repeating modules as seen before.

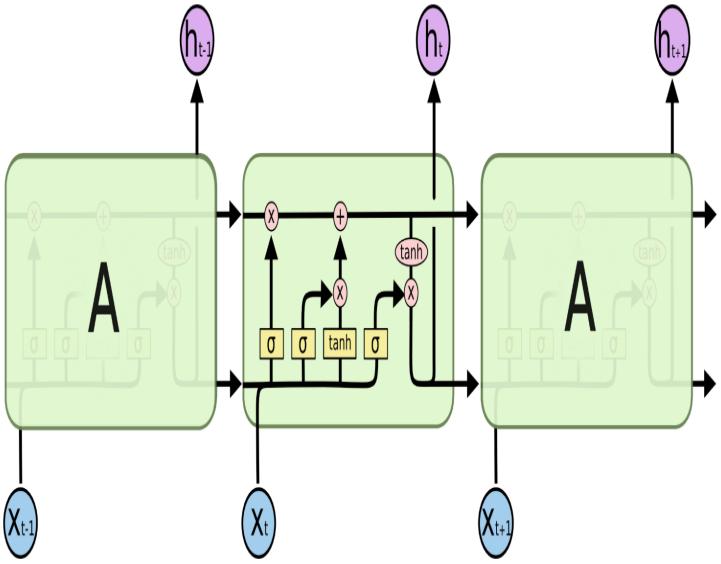

FIGURE 5.7: LSTM interacting internal layers

LSTM adds to the simple RNN a way to carry information across many timesteps. A single internal *tahnlayer* in Vanila RNN is replaced by a different struture made of four internal layers interacting together [2].

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. [2]

5.3.1 The algorithmic structure behind LSTMs

The key component of LSTM is the call state (the horizonal line running on top) acting likewise a conveyor bell. Information can jump onto or off the conveyor at any training time steps. Past information might be stored for later, thus preventing older signals from gradually vanishing during processing [4]. We will call this internal layer as Gates. Gates are a way to optionally let information through. They apply a sigmoid neural net layer and a point wise tensor multiplication.

The first layer is called "forget gate layer".

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{5.10}$$

A sigmoid activation function outputs a number between 0 and 1 looking at h_{t-1} and x_t. A 1 represents completely keep this while a 0 represents completely get rid of this.

In the second step a sigmoid layer, the "input gate layer" decide which value we will update. Following a tahn layer creates a vector of new candidate C_t that could be added to the state. Those steps are cobined to create an update to the state.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{5.11}$$

$$C_t = tahn(W_C * [h_{t-1}, x_t] + b_C) \tag{5.12}$$

At this point, the old call state C_{t-1} get updated into the new cell state C_t. Here it comes the beauty of LSTMs network models. We multiply the old state by f_t, (forgetting the things we decided to forget at the previous step. Then we add $i_t * C_t$. This is the new candidate values, scaled by how much we decided to update each state value.

$$C_t = f_t * C_{t-1} + i_t * C_t \tag{5.13}$$

The final and last step of each internal LSTM layer at each sequential input step dedide what we are going to output. A filtered version of each cell state is computed by a sigmoid of the call state multiplied by a tahn function of the same cell.

$$\sigma_t = \sigma(W_o[h_{t-1}, x_t] + b_o \tag{5.14}$$

$$h_t = \sigma_t * tahn(C_t) \tag{5.15}$$

In essence, all tensor transformation inside a LSTM network we looked so far, are determined by the contents of the weights parameterizing them. The specification of an LSTM cell determines your hypothesis space the space in which youll search for a good model configuration during training but it doesnt determine what the cell does; that is up to the cell weights[4].

5.4 LSTM Model training and forecast

In this section, we will use LSTM Recurrent Neural networks to develop time series forecasting models for Daily Bitcoin market price. My first approach to model Long-Short-Term-memory neural network is an attempt to forecast Daily Bitcoin Closing Price to forecast from its historical value and some featured engineered from daily open-close-high-low. The code I used for this first Deep Learning time series modeling took inspiration from David Sheehan's blog post [32]. In deep learning, the data is typically

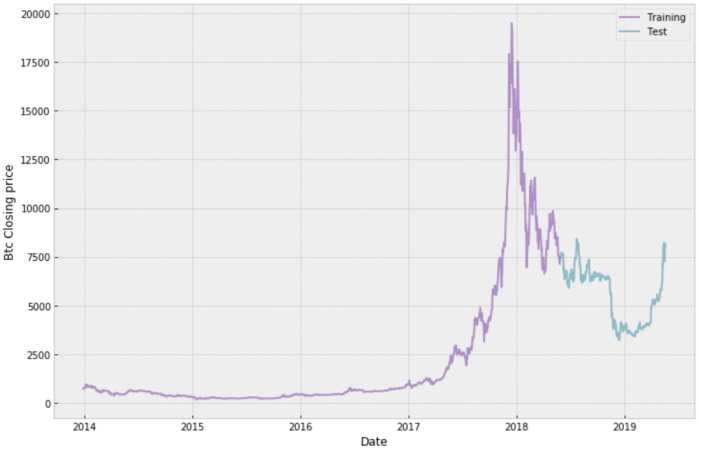

FIGURE 5.8: Train-Test-Split

split into training and test set. The model is built on the training set and only after it is evaluated in previously unseen data. This means computation of the loss function, back propagation and weights updated by a gradient descent algorithm is done on the train set. The validation set is used to evaluate the model and to determine the number of epochs in model training. A common problem which arises quite often in this scenario is overfitting. Increasing the number of epochs will further decrease the loss function on the train set but might not necessarily have the same effect for the validation. The figure below shows the cut split date on $2018 - 06 - 01$.

The input variables we fit into the models are:

- Close-off-high : the gap between the closing price and price high for that day.
- Volatility: the difference between high and low price divided by the opening price.
- Closing price
- Volume

Our model will use previous data to predict the next day's closing price. The model needs to know how many days it will have access to, called windows. Picking a smaller window means that we could fit more windows in our model [32]. Each window size would be an input observation of the LSTM. I have decided to set it to be 10 days to include previous weekly pattern. To help our network during trading, it is a good practice to normalize the data so that our inputs are consistent. The first entry of each window is 0 and all other values represent the change with respect to the first value. Hence, I am predicting price changes, rather than absolute price. Using Keras library Sequential layer staking, I trained a simple neural network with an LSTM hidden layer with 20 neurons. The model also includes more generic neural network features like a

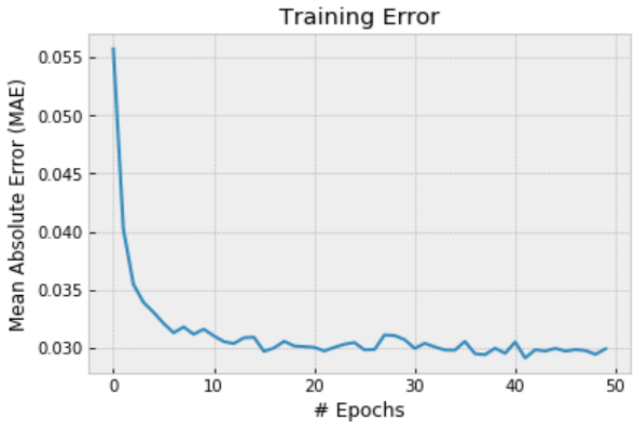

FIGURE 5.9: Training MAE across epochs

FIGURE 5.10: Training prediction vs True Value

dropout factor of 0.25 to avoid overfitting the training dataset. In addition, I used Mean Absolute Error (MAE) as loss function and the Adam as optimiser. For this model I trained the network for 50 epochs using 1 batch size.

The examination of both prediction on the training and test set reveals the fundamental flaw we were expecting from the statistical time series modelling in the previous chapter. The single point forecast we are implementing is almost exclusively using the value of the previous day to predict the next day which is basically what a random walk does. The very low MAE obtain shouldn't misleading us. The shifted predicted price of the one-point prediction are not practically useful for trading.

Also, the prediction of the next 5 days window are never able to track neither the price momentum or the sudden fluctuation. Most of the time, it doesn't deviate from a straight line from the last input price observed.

On the very ultimate attempt to predict the future price of Bitcoin I decided to fit a LSTM neural network with blockchain network variables explored in the previous

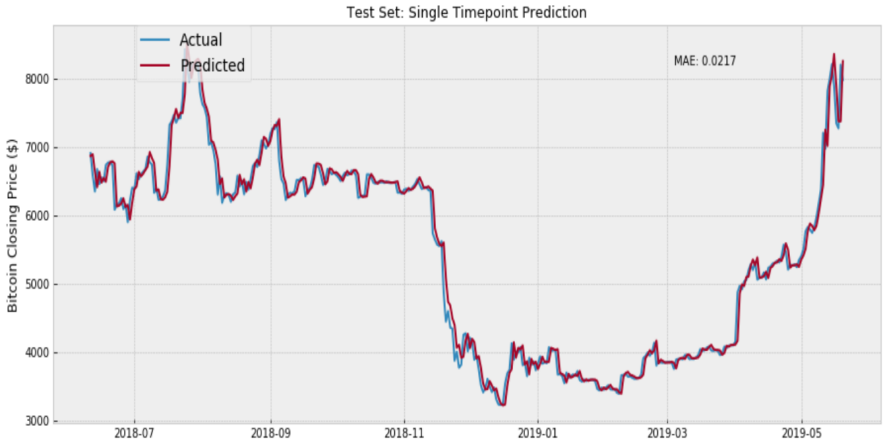

FIGURE 5.11: Test prediction vs True Value

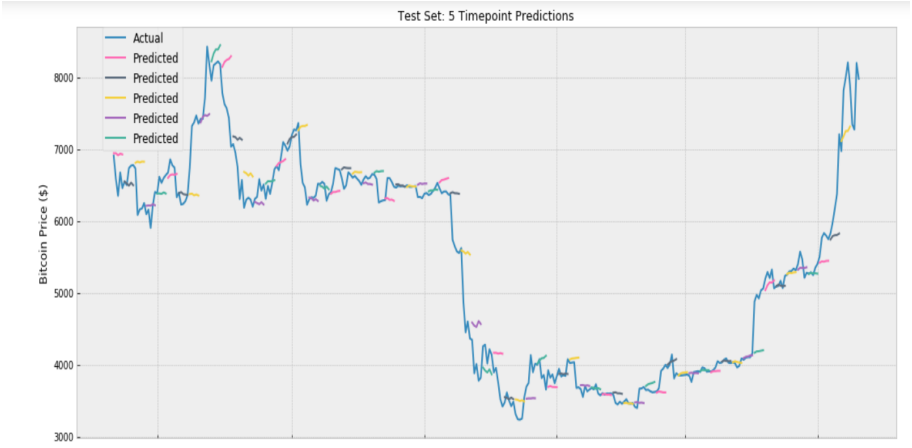

FIGURE 5.12: 5 Days window prediction

chapter. The code for this analysis has been inspired from [1]. Not all blockchain available were used to train the neural network. Out of the all list, variables were selected based in the statistical significance explored during the multivariate ARIMA model fitting in chapter 4. Also, variable selection has the potential to include just only those relevant predictors really useful for Bitcoin Market Price Model purposes. The dependent or target variable of our neural network was the Bitcoin day market price while the daily bitcoin trade volume, bitcoin hash rate, bitcoin miners revenue, bitcoin percent cost per transaction, bitcoin unique address as weel as historical bitcoin price were used to create the LSTM 3 dimensional input sequences. In fact, LSTM variables in the Keras library accept inputs as numpy array of 3 dimensions (number of training sequences, the sequence length or windows size, number of features of each sequence). This time I chose to go for a longer window series of 14 timesteps to help the recurrent neural network model far away time price signal. This forecasting modelling part took a different approach from the one-point step forecast of the previous one that I will call multi-sequence prediction. This method initialize the test window with the test data,

```
config = {}
config["data"] =  {"columns": ["Btc_market_price", "btc_trade_volume", "btc_hash_rate","btc_miners_revenue",
                               "btc_cost_per_transaction_percent", "btc_n_unique_addresses"], "sequence_length": 15,
                   "train_test_split": 0.85, "normalise": True}

config["training"] = {"epochs": 10, "batch_size": 100}

config["model"] = {"loss": "mse",
                   "optimizer": "adam",
                   "save_dir": "saved_models",
                   "layers": [{"type":"lstm","neurons": 32,
                               "input_timesteps": 14,
                               "input_dim" : 6,"return_seq": True},
                              {"type": "dropout","rate": 0.2},
                              {"type": "lstm","neurons": 32,"return_seq": True},
                              {"type":"lstm","neurons":32,"return_seq": False},
                              {"type": "dropout","rate": 0.2},
                              {"type": "dense","neurons": 1,"activation": "linear"}]}
```

FIGURE 5.13: Input LSTM neural network

predicts the next point over that and make a new window with the next point. However, once it reaches a point where the input window is made up fully of past predictions it stops, shifts forward one full window length, resets the window with the true test data, and starts the process again. In essence this gives multiple trend-line like predictions over the test data to be able to analyze how well the model can pick up future momentum trends [1]. Object-oriented-programming in python is a great way to optimize coding. Inputs fed into the network were stored into a dictionary for easy element extraction and reusability.

Figure 5.14 shows the dictionary named "config" which defines the hyperparameters of our neural network. The best network structure was found to be made of 3 hidden LSTM layer with respectively 32, 32 and 32 neurons. Again, I added a Dropout layer at each layer with a rate of 0.2. The objective function to minimize in 10 training epochs using 100 batch side is the mean squared error. After a series of preprocessing function, a "Model Class" object builds our network using parameters from the "config" file. With the data loaded and the model built we can now progress onto training the model with our training train generator function.

Last but not list the multi-sequence function and prediction which predict a sequence of 14 steps before shifting prediction run forward by 14 steps

5.5 Conclusions

The goal of this research was to evaluate several time series forecasting models to cryptocurrency financial data. By now it should be crystal clear that Bitcoin price is not any sort of specific static function which can be mapped. The best property to describe the motion of a Bitcoin price time series would be a random walk. As a stochastic process,

```
class Model():
    """A class for an building and inferencing an lstm model"""

    def __init__(self):
        self.model = Sequential()

    def load_model(self, filepath):
        print('[Model] Loading model from file %s' % filepath)
        self.model = load_model(filepath)

    def build_model(self, configs):
        timer = Timer()
        timer.start()

        for layer in configs['model']['layers']:
            neurons = layer['neurons'] if 'neurons' in layer else None
            dropout_rate = layer['rate'] if 'rate' in layer else None
            activation = layer['activation'] if 'activation' in layer else None
            return_seq = layer['return_seq'] if 'return_seq' in layer else None
            input_timesteps = layer['input_timesteps'] if 'input_timesteps' in layer else None
            input_dim = layer['input_dim'] if 'input_dim' in layer else None

            if layer['type'] == 'dense':
                self.model.add(Dense(neurons, activation=activation))
            if layer['type'] == 'lstm':
                self.model.add(LSTM(neurons, input_shape=(input_timesteps, input_dim), return_sequences=return_seq))
            if layer['type'] == 'dropout':
                self.model.add(Dropout(dropout_rate))

        self.model.compile(loss=configs['model']['loss'], optimizer=configs['model']['optimizer'])

        print('[Model] Model Compiled')
        timer.stop()
```

FIGURE 5.14: Model Class Object

```
def train_generator(self, data_gen, epochs, batch_size, steps_per_epoch, save_dir):
    timer = Timer()
    timer.start()
    print('[Model] Training Started')
    print('[Model] %s epochs, %s batch size, %s batches per epoch' % (epochs, batch_size, steps_per_epoch))

    save_fname = save_dir + '_%s-e%s.h5' % (dt.datetime.now().strftime('%d%m%Y-%H%M%S'), str(epochs))
    callbacks = [
        ModelCheckpoint(filepath=save_fname, monitor='loss', save_best_only=True)
    ]
    self.model.fit_generator(
        data_gen,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        callbacks=callbacks,
        workers=1
    )

    print('[Model] Training Completed. Model saved as %s' % save_fname)
    timer.stop()
```

FIGURE 5.15: Train generator

```
def predict_sequences_multiple(self, data, window_size, prediction_len):
    print('[Model] Predicting Sequences Multiple...')
    prediction_seqs = []
    for i in range(int(len(data)/prediction_len)):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(self.model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-2], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs
```

FIGURE 5.16: Predict Multiple seq function

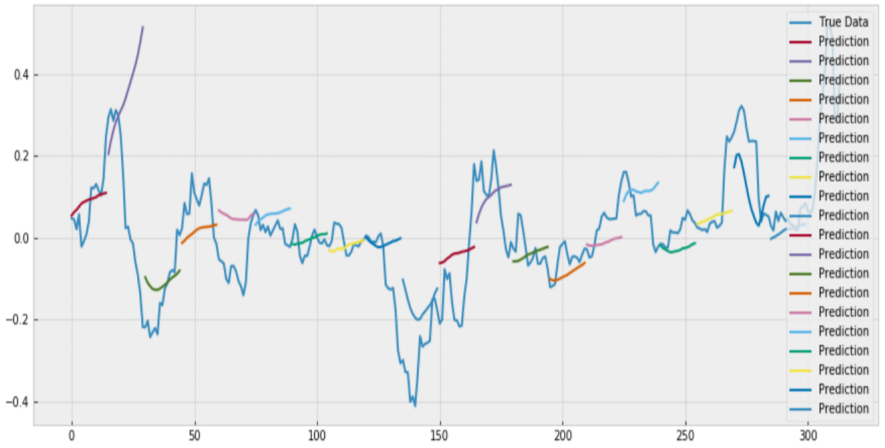

FIGURE 5.17: Train$_g$*enerator*

a true random walk has no predictable patterns and so attempting to model it would be pointless. Fortunately, the LSTM neural networks using Blockchain network variables allowed me to theorize that the time series may well have some kind of hidden pattern. Ultimately, the LSTM managed to learn some signals from our training. The accurate prediction of the price market drop in the last multi out of sample sequence prediction is a starting point for future work. Using more data, as well as optimising network architecture and hyperparameters are a start. In my opinion, however, there is more potential in incorporating data and features that go beyond historic prices alone. Based on the metrics result I reported for each methods, it would not fair comparing Deep Learning models and classical statistical forecasting models such as ARIMA. Although results improved using blockchain variables, still the linear hypothesis space which those algorithm search for a good models might not be sufficient to model high volatile and non-stationary cryptocurrencies. The MAE obtained below 20% on the test set can be considered a good achievement for those algorithms. Even though, a more accurate metrics would use a cross-validation window sliding train split test to evaluate model performance. In general, a great benefit of ARIMA is that is possible to make statistical inference on the models estimates. Also, if model hypothesis are satisfied, those algorithms can output confidence interval predictions, statistical p-value significance for each coefficient and many other statistical metrics. In our case, for many blockchain network variables after removing spurious correlation and running cross-correlation and granger-casuality test, it wasn't possible to argue in any strong and deterministic linear association with historical bitcoin market price. Even with the most difficult forecasting task out there, Deep Learning and specifically Long-Short Term Recurrent Neural Network did a great job. I am really optimist for the fact that on the test set the neural network manage to model non-linearity in the data and at least for short term prediction outperformed ARIMAX models by a great extent. Unfortunately, we are facing a black-box algorithm. Model weights learned by LSTM are as much astonishing as deeply

non-interpretable from human-being. To overcome this issue, Bayesian Neural Network is today the state-of-the art algorithm to perform time series forecasting in the financial market sector.

Bibliography

[1] Jakob Aungiers. *Time series prediction using LSTM Deep Neural Network*. URL: https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks.

[2] Colah's blog. *Understanding LSTM Networks*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[3] Jason Brownlee. *gentle-introduction-autocorrelation-partial-autocorrelation*. URL: https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/.

[4] Franfffdfffdois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018. ISBN: 978-1-4419-7864-6.

[5] David LeeKuo Chuen. *Introduction to Bitcoin*. URL: file:///C:/Users/Andrea/Downloads/Chapter1-DigitalCurrency_DavidLee.pdf.

[6] Pavlin Mavrodiev David Garcia Claudio J. Tessone and Nicolas Perony. *The digital traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin economy*. URL: https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2014.0623.

[7] Erik Nigmatullin Dean Fantazzini. *Everything you always wanted to know about Bitcoin but were afraid to ask*.

[8] Richard L. Scheaffer Dennis D. Wackerly William Mendenhall III. *Mathematical Statistics with Applications, Seventh Edition*. Thomson Brooks/Cole, 2008. ISBN: 978-0-495-38508-0.

[9] MURAT KULAHCI DOUGLAS C. MONTGOMERY CHERYL L. JENNINGS. *Introduction to time series analysis and Forecasting, Second Edition*. Wiley, 2015.

[10] Dr. PKS Prakash Dr. Avishek Pal. *Practical Time Series Analysis*. Packt Publishing Ltd, 2017. ISBN: 978-1-78829-022-7.

[11] Roger T. Dean William T. M. Dunsmuir. *Dangers and uses of cross-correlation in analyzing time series in perception, performance, movement, and neuroscience: The importance of constructing transfer function autoregressive models*. URL: https://arxiv.org/ftp/arxiv/papers/1405/1405.4498.pdf.

[12] Ifigeneia Georgoula. *USING TIME-SERIES AND SENTIMENT ANALYSIS TO DETECT THE DETERMINANTS OF BITCOIN PRICES*. URL: https://pdfs.semanticscholar.org/05f3/7fcb95488402bb9e4d0d5a291e1338de41a6.pdf.

[13] Aojia Zhao Isaac Madan Shaurya Saluja. *Automated Bitcoin Trading via Machine Learning Algorithms*.

[14] Aarshay Jain. *A comprehensive beginner guide to create a Time Series Forecast (with Codes in Python)*. URL: https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/.

[15] HUISU JANG and JAEWOOK LEE. *An Empirical Study on Modeling and Prediction of Bitcoin Prices With Bayesian Neural Networks Based on Blockchain Information*. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8125674.

[16] Yury Kashnitsky. *Topic 9. Part 1. Time series analysis in Python*. URL: https://www.kaggle.com/kashnitsky/topic-9-part-1-time-series-analysis-in-python.

[17] Ladislav Kristoufek. *BitCoin meets Google Trends and Wikipedia: Quantifying the relationship between phenomena of the Internet era*. URL: file:///C:/Users/Andrea/Downloads/Kristoufek2013_BitCoin%5C%20(2).pdf.

[18] Helmut Lfffdfffdtkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2005.

[19] Ilaria Lunesu Martina Matta and Michele Marchesi. *The Predictor Impact of Web Search Media On Bitcoin Trading Volumes*. URL: https://www.bitcoinnews.ch/wp-content/uploads/2015/10/Final.pdf.

[20] Sean McNally. *Predicting the price of Bitcoin using Machine Learning*. URL: http://trap.ncirl.ie/2496/1/seanmcnally.pdf.

[21] Paul S.P. Cowpertwait fffdfffd Andrew V. Metcalfe. *Introductory Time Series with R*. Springer, 2009. ISBN: 978-0-387-88697-8.

[22] Tom Michell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[23] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf.

[24] Micheal Nielsen. *Neural Networks and Deep Learning*.

[25] d'Artis Kancs Pavel Ciaian1 Miroslava Rajcaniova. *The Economics of BitCoin Price Formation*. URL: https://arxiv.org/ftp/arxiv/papers/1405/1405.4498.pdf.

[26] dfffdfffdfffdArtis Kancs Pavel Ciaian1 Miroslava Rajcaniova2. *The digital agenda of virtual currencies: Can BitCoin become a global currency?* URL: https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2014.0623.

[27] Blog post. *Time Series Analysis (TSA) in Python - Linear Models to GARCH*. URL: http://www.blackarbs.com/blog/time-series-analysis-in-python-linear-models-to-garch/11/1/2016.

[28] SUNIL RAY. *Understanding and coding Neural Networks From Scratch in Python and R*. URL: https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/.

[29] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. URL: https://arxiv.org/pdf/1609.04747.pdf.

[30] Kaye scholer. *An introduction to Bitcoin and blockchain technology*. URL: https://files.arnoldporter.com/docs/IntrotoBitcoinandBlockchainTechnology.pdf.

[31] Nist Sematech. *Engineering statistics handbook*. URL: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4431.htm.

[32] David Sheehan. *Predicting Cryptocurrency Prices with Deep Learning*. URL: https://dashee87.github.io/deep%5C%20learning/python/predicting-cryptocurrency-prices-with-deep-learning/.

[33] AISHWARYA SINGH. *A Multivariate Time Series Guide to Forecasting and Modeling (with Python codes)*.

[34] SRK. *Cryptocurrency Historical Prices*. URL: https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory.

[35] Robert H. Shumway fffdfffdfffd David S. Stoffer. *Time Series Analysis and its applications with R examples, Third edition*. Springer, 2011. ISBN: 978-1-4419-7864-6.

[36] Thomas Vincent. *A Guide to Time Series Forecasting with ARIMA in Python 3*. URL: https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3#step-6-%5C%E2%5C%80%5C%94-validating-forecasts.

[37] Jeffrey M. Wooldridge. *Introductory Econometrics: A Modern Approach, Fifth Edition*. South-Western, Cengage Learning, 2013. ISBN: 978-1-111-53104-1.

[38] George Swales Youngohc Yoon. *Predicting Stock Price Performance: A Neural Network Approach.* URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=184055.

[39] John Berkowitz Zachary C. Lipton. *A critical Review Of Recurrent Neural Networks for sequence Learning.* URL: https://arxiv.org/pdf/1506.00019.pdf.