Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Comparison between Gradient Boosting Ensemble and Long Short Term Memory Neural Network for Cryptocurrency Investing

Trabajo Fin de Máster

presentado por: Buftea, Alberto Gabriel

Dirigido por: Fernandez García, Antonio Jesus

Ciudad:Málaga

Fecha: 19 de Septiembre of 2022

Index

Figures Index

Tables Index

Listings

Resumen

En este trabajo revisaremos las técnicas de inteligencia artificial que dieron mejores resultados en los en los artículos de investigación publicados en relación a predecir el precio de las criptomonedas. El propósito de este trabajo es servir como una guía completa para el desarrollo de un algoritmo de previsión de cryptomonedas, comenzando por explicar como funcionan las criptomonedas, después revisaremos las técnicas que han sido usadas por otros investigadores para después elegir las dos con las que obtuvieron los mejores resultados. Luego explicaremos su funcioanmiento y cómo replicarlas en nuestro entorno de trabajo para finalmente realizar prediciones sobre el precio de una criptomoneda y comparar los resultados de ambos modelos. Con el experimento realizado durante esta investigación, veremos que con la técninca de machine learning Gradient Boosting Enssemble obtenemos mejores resultados con menor tiempo de procesamiento en comaparción a la técnica de deep learning Long Short Term Memory.

Palabras Clave: Inteligencia Artificial, Criptmonedas, Inversión

Abstract

In this work we will review the artificial intelligence techniques that achieved best results in state of the art publications about cryptocurrency price prediction. The objective of this dissertation project is to serve as a complete guide for the development of a crypto price prediction algorithm. We will review the basics about cryptocurrencies to understand their working fundamentals, later we will investigate state of the art publications and choose the two techniques that achieved best results in overall, one for machine learning and another one for deep learning. We will explain their logic and how to replicate those models in our working environment. The last stage of this project consists on making predictions for ethereum price to finally compare the results of both models. In the specific experiment done in this project, we will see that the machine learning approach Gradient Boosting Ensemble outputs better results than the deep learning approach Long Short Term Memory.

Palabras Clave: Artificial Intelligence, Cryptocurrency, Investing

Chapter 1

Introduction

The aim of this dissertation project is to compare state of art investing algorithms which were published in scientific papers and used artificial intelligence. In this project we identify what models achieve better results, selecting one from the side of machine learning and one from the side of deep learning to replicate them in our testing environment and personalize them, using technical analysis indicators and social media sentiments as input features besides the usual *open, high, low, close and volume* fundamental indicators.

When it comes to investing in the modern world, there are big players like multi-billion dollar companies and funds trying to make profit. They hire teams of engineers and financial experts who focus at developing boots and auxiliary algorithms to help at decision taking, investing millions on infrastructure and cloud computing facilities which in return grants a computational power non accessible by an individual investor. Even though we could have the feeling that it is impossible to make profit form the market because we have to compete with them, there are factors which are in favor of individuals when taken in consideration. Two of them are most relevant.

- Due to elevated infrastructure and labor costs, those investors needs to make several hundreds of thousands of earning with each investment decision. They will not be interested into assets with low market capitalization or limited growing possibilities, which on the other hand are very good assets for individuals.
- Due to the amount of money they invest and the way financial markets work, big stake investors risk altering the price of the asset they buy on the opposite direction

of their interests, or they risk buying the major part of the company if they decide to invest. This makes it impossible for them to invest in a series of assets which can be very profitable for individuals.

Fang et al. (2015) states that financial assets prices follow the offer and demand rule, they cost as much as someone is willing to pay for it. This factor makes it very difficult for an algorithm to estimate the future price of an asset since it is based on human decisions which are based on emotions and gut feelings, involving their particular vision of the economies, human progression, new technologies development, political measures and hundreds of other factors which are impossible to estimate like their personal interest and the influence of others on them.

Here is where cryptocurrencies comes into place. As Ammous (2018) explains, the cryptocurrency is a recent asset which claims to be a decentralized currency, meaning it is not controlled by a central institution like the central banks. This makes it complicated for governments to regulate, hence, cryptocurrency prices are not influenced by as many factors as traditional investment assets but for the same reason we cannot guarantee that crypotcurrencies offers stability. In fact they do not, as we will later see in this chapter cryptocurrency markets have a very high volatility. Ammous (2018) states that price fluctuations are expected to continue because the only point at which a cryptocurrency could become stable is when it will be owned by a large portion of humanity such that marginal changes does not have high implications in its price. According to the author, this point is unlikely to happen anytime soon, suggesting that the high volatility of cryptocurrencies will continue for a longer period. This makes it questionable that cryptocurrencies will be used as a unit of account for market participants.

The market volatility is the main feature of the cryptocurrency that we are trying to make profit from. When it comes to investing in companies, investors studies their fundamentals like their purpose, who are the directors, how many assets do the company have, where are the headquarters located, what partnerships do they have and so on. What drives people at investing in a company is felling connected with the purpose of that company and being confident that the companies positive impact on society will grow, hence its stock market price.

Cryptocurrencies lack those social and political fundamental drivers. Their main attraction are the story of other becoming millionaires thanks to investing and the sentiment of opposition to traditional central economies. But on the same way that we have seen success stories, we also have seen many failures like the state of El Salavador who adopted the Bitcoin as an official currency just before the price of this asset started to fall sharply. This happens due to the volatility of the cryptocurency market, meaning that the price can sharply increase or sharply fall in a very short period of time. *Figure* 1.1 represents the evolution of the price of ethereum, a cryptocurrency, in United States dollars starting from 2019 till nowadays. We can see that at the end of 2020, the graph has started to develop several upward and downward trend which are characterized by many spikes. This is known in finance as volatility.

Figure 1.1: Ethereum Price Chart since 2019

Source: https://www.coingecko.com/

This volatility is both, an indicator of high risk since the price of your investment can sharply fall in a matter of several days and also a great opportunity to make big profits very quickly if the price of your investment increases sharply. To profit form the volatility, you need to stay aware of the price movements on a daily or even hourly basis. This is something humans are not designed to do and I think nobody really enjoys staying in front of a computer watching graphs, but we do it becasue we do not want to loose the opportunity to invest in cryptocurrencies and make profits form them. This is the reason

I have decided to develop this project, I think the best approach for trading in crypto is to develop an artificial intelligence who never tires of tracking the price movements and it could even take investment desicions for us.

This work intends to be educational, teaching the reader about the whole process of using algorithms at predicting cryptocurrencies prices. We are going to see what are cryptocurrencies and how do they work. We will review what artificial intelligence techniques performed better at investing in crypto and we will select one form deep leaning and another from machine learning, explain how those techniques works and how to replicate them in our working environment using python, to afterwards personalize the models by adding more relevant features to the dataset than just the history of prices and volumes, to finally compare the results of the deep learning and the machine learning models at predicting ethereum price.

Next chapters of this thesis are structured as indicated below:

- **Chapter 2** explains the objective of this project.
- **Chapter 3** reviews the basics about Cryptocurrencies, how are them generated, maintained, and how can we buy and sell them.
- **Chapter 4** covers the state of art investigation papers about crypto investing. We will review the most recent/relevant scientific articles and the artificial intelligence techniques they used.
- **Chapter 5** Covers the project development where we are going to explain the models, the data intake and processing, we will review how to create those models and compare their outputs.
- **Chapter 6** will be the final of this dissertation project where conclusions are drawn and next steps presented.

Chapter 2

Objective

When investigating about algorithmic cryptocurrency investing I detected several deficiencies that I want to address in my work.

- No cryptocurrency investing paper properly explains the working fundamentals of cryptocurrencies.
- Investing publications usually skips the working principles of the models they are using, just briefly stating what kind of techniques they have used.
- Publications does not explain how to create those models, how to prepare the data and how does the model hyperparametes changes affects the output.
- Most of the publications uses as features only basic market information *OHLCV* open, high, low, close prices and volume.

My intention with this project is to offer a complete guide for someone interested into algorithmic investing, covering all the relevant information from the basics of cryptocurrency to setting up a state of the art model personalized and feed with different data sources.

From the completeness point of view, I consider that it is very important for people interested into algorithmic crypto investing to understand how cryptocurrencies are being generated and what are the differences between the different existing digital coins. This helps at understanding the current market value, understanding the risks involved and helps at designing the investment strategy. In this paper I also explain how data should be structured and processed such that it can be used as inputs to the model, therefore,

readers could personalize their own dataset and use different data sources to create their own features. I also cover the model working principles and hyperparameters, sharing the whole process of builting the model such that readers can easily replicate my experiments obtaining similar results and adjusting, modifying or expanding the models to perform their own experiments or use the models for investments in different assets like real estate or stocks.

From the novelty point of view, in this project I am combining social media sentiments from Augemto API togeder with the *OHLCV* price data downloaded form CoinAPI cryptocurrency data API and additional technical analysis indicators which are calculated using the python library Technical Analysis (TA). This gives us the freedom to generate a complete dataset with state of the art features used by professional investors. In addition, contrary to the usual approach of using the whole history of a cryptocurrency price for training purposes, in this project we are focusing on data starting from December 2020 when the cryptocurency market had a very sharp increase and began it volatile track on which we are today. This is a positive aspect of this project since cryptocurrencies are recent so we will not have a long history of data form most of them.

To sum up, you are about to read a cryptocurency investment project where the best performing state of the art deep learning approach is going to be compared with the best state of the art machine learning approach for predicting the price of ethereum. The model creation, working principles, hyperparameters configuration and how do they impact the output will be explained, while the inputs to the models will be expanded form those used in the bibliography by adding social media sentiments indicators and technical analysis indicators.

Chapter 3

Introduction to Cryptocurrency

In this chapter we are going to have an introduction to the world of Cryptocurrencies, briefly explaining how did they emerged, how are generated and maintained. *Section* 2.1 covers the emerging and exchanging of cryptocurrencies while *section* 2.2 coves the basics about blockchain and mining, which are forms to maintain and generate those digital assets.

3.1 Cryptocurrency systems essentials

As Härdle et al. (2020) says, «a cryptocurrency is a digital asset designed to work as a medium of exchange using cryptography to secure transactions, to control the creation of additional units and to verify the transfer of assets»(p. 3). Crypto could eliminate the intermediaries and assure that payments and transactions arrive instantly. This represents a benefit compared to the traditional banking system, where payments or transfers may delay for several days while banks also charge business owners for using their payment solution on the same way as they charge customers fees for cards and accounts.

Berentsen and Schär (2018) has a very good introduction at explaining cryptocurrencies. They start explaining that cash represents a unit of value which is transferred to whoever owns the coin or the note without the need for intermediary. In order to ease the need of physical presence, ideally we would have an electronic object which represents a unit of value and can be transferred to another person via e-mail for example. The problem of this is that electronic objects can be duplicated an infinite number of time at no cost. To

eliminate this issue, electronic payments introduces the figure of a central authority which electronically receives the money form the buyer and sends it to the seller. In order for this system to work, all the parties need to trust the central figure.

To eliminate the need of central authority, a distributed ledger payments system has been created. Berentsen and Schär (2018) uses a very good metaphor to describe this system. On an island called Yap inhabitants used very large stones to make payments. The stones were brought from a place only accessible by boat very far away from the island. Even though every person could go and bring as much stone as they want, the labor and equipment cost protected the island economy from inflation. At one moment, instead of physically moving the stone form one place to another with every transaction, inhabitants decided to virtually transfer the unit of value that the stone represents to another person by letting everybody on the island know that the respective stone is not owned anymore by them, so even though the stone was in the backyard of one person, that stone was not owned by that person and everybody would be aware of it. In that way, stones remained at their original location in the island meanwhile the unit of value was detached form it and virtually circulated trough the Island.

Cryptocurrencies follow the same rule as the Yap Island financial system. They are divisible virtual monetary units with no physical representation whose transactions are tracked in a data file system named blockchain. The blockchain keeps records of all past transactions and also the addition of new units in the system. Each new entry in the blockchain builds up on the previous entry containing the information about the new transaction. The blockchain is a public record, thus, everybody can access it. Every system participant is free to download and manage their own copy of the blokchain being able to apply modifications to it. Blockchain modifications follow a set of established rules, any modification needs to be broadcast to every system participant who needs to give his approval in order for the new transaction to be added to the main copy of the blokchain. This is the way to avoid the need of central authority who is controlling every system movement, similar to Yap island inhabitants who had to notice everybody they do not own that specific stone anymore.

3.2 Blockchain and Mining

Figure 3.1 shows the architecture of a centralized, decentralized and distributed system. As told by Mukhopadhyay et al. (2016), blockchain is often referred as a distributed ledger transactions technology where each transaction is stored into a block that consists on various verified transactions. Each crytocurrency fixes the maximum size of a block, limiting the number of transactions that could be introduced.

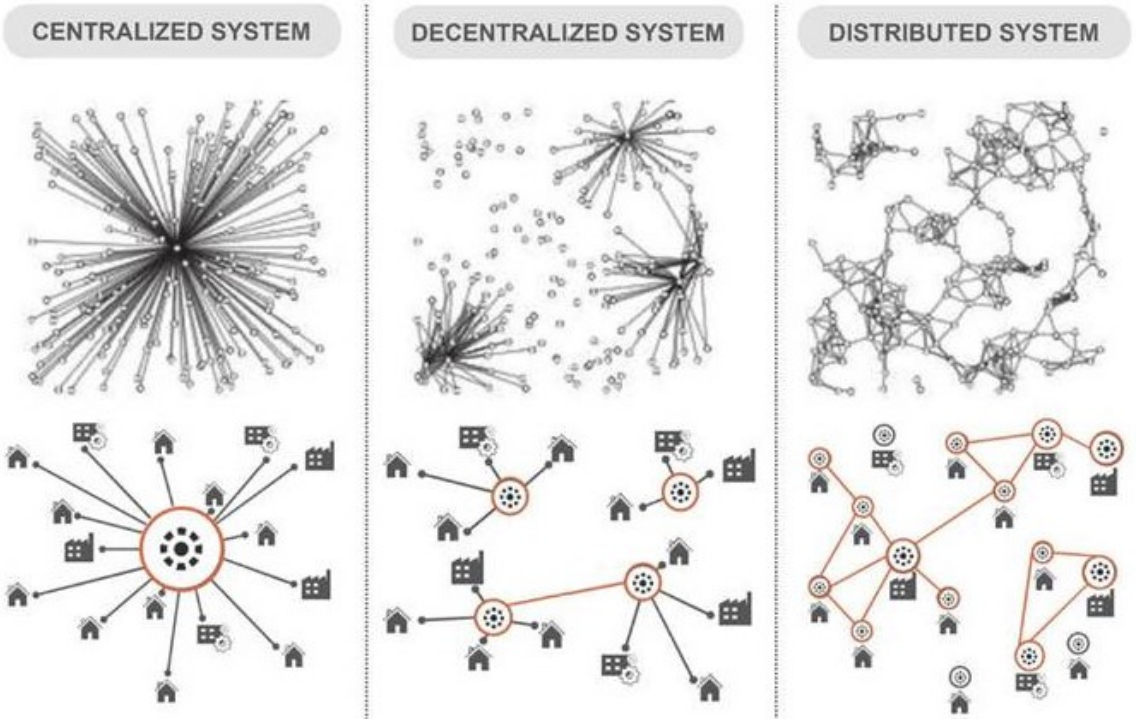


Figure 3.1: Type of networks

Source: https://web.itu.edu.tr/

Yaga et al. (2018) indicates that each blockchain can define its own design for the block structure, but most commonly, blocks are separated between the **block header** and the **block data** which usually include, but not limited to, the following information:

- **Block Header**
  - Block number
  - Previous block hash
  - Current block hash
  - Time stamp

  - Size of the current block
  - Nonce value (used at solving the hash puzzle)

- **Block Data**
  - List of transactions
  - Ledger events
  - Transactions counter
  - Other items defined by the blockchain

Miners use the hash function form a previous block to generate the nonce and the hash function for the following block. Each block in the chain contains the hash digest of previous block headers forming the blockchain as in *figure* 3.2. There is only one path from the first block hash function to the last block hash function, which help blockchains participants to easily detect and reject any alteration in one of the middle blocks, since the path will loose the coherence, and makes it very difficult to change any transaction from the blockchain since it involves creating as many blocks as there currently exists with their hash function related.

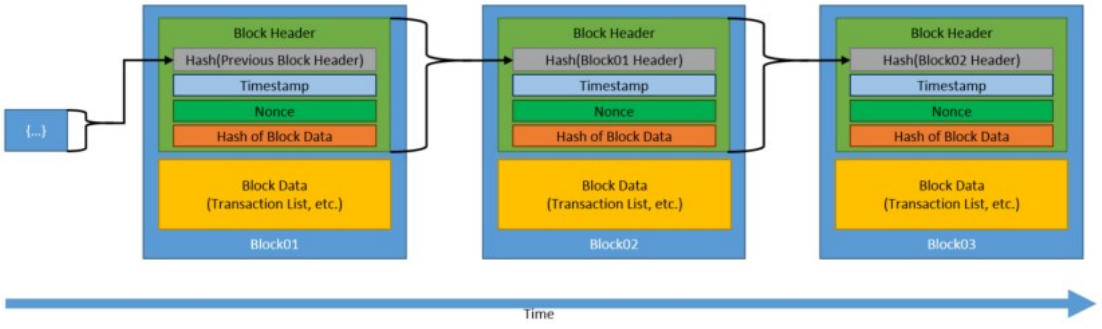


Figure 3.2: Blochain blocks

Source: Yaga et al. (2018)

In order to determine which users can publish a new block to the blockchain, the issuer of the blockchain publishes a consensus model aimed for enabling a group of distrusting users to work together. Each user joining the blockchain network has to agree with the initial state of the system that is published in the first block, named the *"genesis"*. Each new block published in the network, mush be added after the genesis and users should follow the consensus model of the blockchain. The consensus model is designed to decide

which system participant adds the next block to the chain, there are several types of consensus models:

- **Proof of Work (PoW) model**, where the user that publishes the next block is the one who solved a computationally expensive puzzle.
- **Proof of Work (PoS) model**, which awards users that invested the more into the system since they are more likely to want the system to succeed
- **Round Robin model**, where each system participant has its turn for creating a block between a time limit, ensuring there's no node creating the majority of blocks.
- **Proof of Authority/Identity model**, where publishing nodes must have their real world identity proven and verified within the network .
- **Proof of Elapsed Time model**, where nodes in the system are granted a random wait time in which they are idle and can generate and publish one block once that time is over informing all the other nodes. Whenever a block is published, all the nodes wake up form the idle states and a new random waiting time is being given, repeating the process.

Mining is the process of validating transactions in the blockcahin and publishing then in a new block. Miners can check if the transaction funds really belong to the payer since the whole history is available in the blockchain. The miner who generates the new block is the one who has to validate the whole set of transactions happening from the publication of the last block till the publication of the new block he is generating. In order to encourage users to join the network and use their hardware at maintaining the system, a small reward is granted to the miner who has published the block.

Chapter 4

Context and State of Art

In this chapter we will review several recent scientific papers covering cryptocurrency investing. From the information that we are going to extract from them, one machine learning approach and one deep learning approach will be selected to replicate in the *fifth chapter*. We will base our selection on the models that achieved best overall results in the experiments performed by our peers.

In Liew et al. (2019), the authors compare the prediction of over 100 cryptocurrencies using 11 different algorithms. To perform this analysis they first studied the correlation between the most traded criptocurrencies with traditional assets such as the SP 500 index. Not surprisingly, in this analysis they found that cryptocurrencies with larger market capitalization have a bigger correlation with traditional markets assets than those with lower capitalization. Although, an interesting finding from their work is that cryptocurrencies in general have almost no correlation with traditional financial assets, being the overall correlation less than 10%. This supports our statement that cryptocurrency prices do not fluctuates based on the same events as traditional assets.

The investigation realized by Nosratabadi et al. (2020) used the *prisma* approach to find the 57 most relevant articles published in the area of data science applied to the economy. The criteria which they used to structure the results was splitting by the research area and the methodology employed, counting the number of documents published for each group. From their findings, apart form the yearly rise of the published documents as seen in *figure* 4.1, they state that for the research area of investing, 34 articles were focused

in the stock market while 3 articles were focusing on cryptocurrencies. When it comes to compare the methodology used at investing, their conclusion is that the deep learning networks using long short term memory models outperforms other data science techniques.

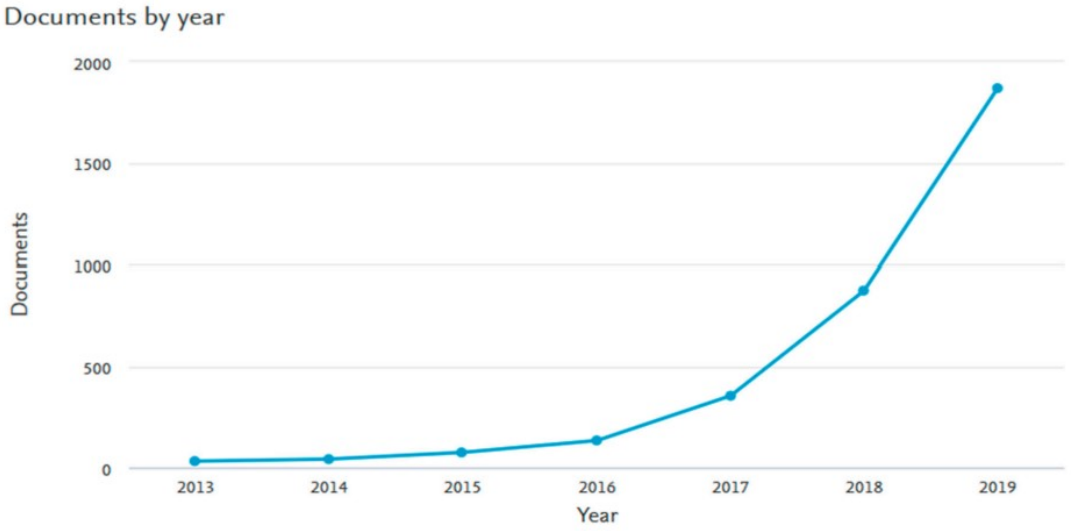


Figure 4.1: Rise in documents about data science in the economy

Source: Nosratabadi et al. (2020)

In the work performed by citeZanin2018, they compared two machine learning techniques based on gradient boosting (GB) decision trees with the deep learning long short term memory (LSTM) neural network at cryptocurrency investing. The study concluded that GB techniques works better when predictions are based on short term windows while LSTM predictions are more accurate when they involves a window of several weeks. An important finding from the study is that all of the models performed better than the simple moving average strategy, which gives purpose to the paper that we currently writing. Their article does not states the models prediction metrics but they do share with us the cumulative return generated over a period of two years for each technique they have used. Seen in *figure* 4.2, model one and model two are different strategies bassed on the gradient boosting technique while the method three is a strategy based on LSTM netowork. The blue line is the simple moving average strategy.

Having seen that for the deep learning approach we have the LSTM network which is said to outperform any other technique, we will lastly review the paper published

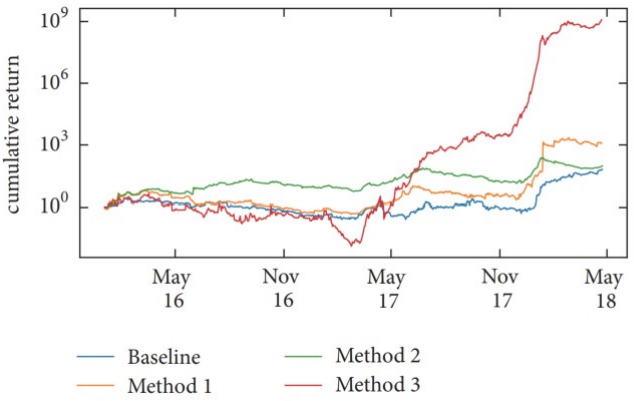


Figure 4.2: Cumulative returns using two Gradient Boosting techniques and one Deep Learning Techniques

Source: Zanin et al. (2018)

by Derbentsev et al. (2021) which focuses specially on machine learning algorithms. In their article, they compared what are said to be the two most powerful machine learning approaches, bagging random forest and gradient boosting ensembles. They predicted the price of two cryptocurrencies for a 95 days period obtaining some surprisingly accurate results. From their findings, the gradient boosting ensemble achieved the best performance, being 10% more accurate than the bagging random forest. See *figure* 4.3 where the blue line represents the real price of ethereum, the dotted red line represents the BRF approach while the dotted gray line represents the GBE approach. We can be impressed on how closely the predicted curves tracked so close the real price, while also seeing how the GBE curve more closely matches the blue one, indicating a better prediction for the GBE technique.

Being our state of art investigation finished, by seeing the increasing number of published articles of data science in the field of the economy, our conclusion is that data science is gaining interest to enter the investment business. Reading the success stories of other investigators which achieved very close predictions to the real price of the assets results and even they even simulated investment condition that generated positive results, gives sense to the investigation that I am doing an expectations that it is going to be helpful for colleagues interested to pursue an investigation in the same area.

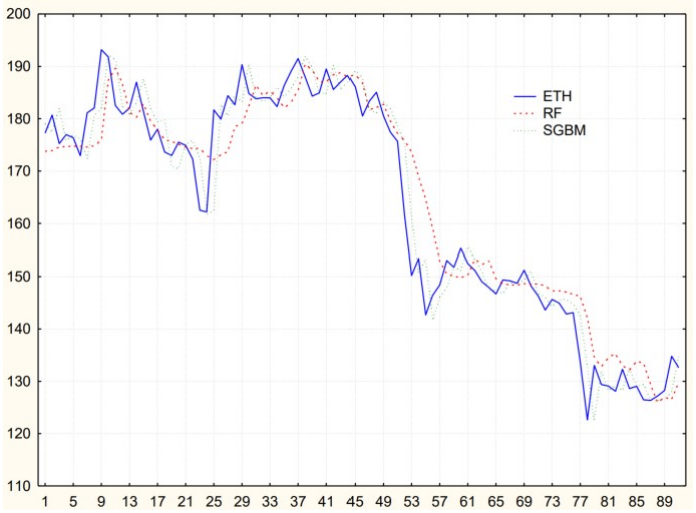


Figure 4.3: Ehtereum price prediction base on BRF and GBE models

Source: Derbentsev et al. (2021)

All of the papers that have been reviewed uses a long timespan of several months or even years for the models to predict, and all of them got better results using the LSTM netowrks for deep learning except the paper published by Derbentsev et al. (2021) which focused specially in machine learning, where they got more accurate results using the gradient boosting technique. Therefore, those two are the models that we are going to replicate in our project development chapter and will predict the price of ethereum to afterwards compare the results achieved by both models and see which one performed better in our test scenario.

Chapter 5

Project Development

After reviewing the state of the art in the previous chapter, the two models that we are going to implement are **Long Short Term Memory (LTSM)** artificial neural network for deep learning and the **Gradient Boosting Ensemble (GBE)** for machine learning.

In this chapter I will cover the data intake and processing, the working fundamentals of both techniques, how to replicate the models in our working environment using python, how to tune the models hyperparameters to obtain the best performance from each model and how to transform the dataset in such a way that is can be used as model inputs. We will finally review the output of each model comparing them based on several metrics to decide what technique performed better at predicting the price of ethereum.

Specific code snippets will be included in this chapter to illustrate how filtering was applied, how the models are created and almost every action that is covered in the paper. However, if you would like to find the whole source code of the project, *appendix B* is the source code for the data retrieval process and dataset processing, *appendix C* is the source code of the gradient boosting technique creation, prediction and metrics while *apendix D* is the LTSM technique creation, prediction and metrics source code.

The project is developed using *python* on the *jupyter notebook* environment. For data processing we used the *pandas* and *numpy* libraries. The information is downloaded form CoinAPI for the *OHLCV* prices, Augmento API for social media sentiment indicators and technical analysis indicators are being calculated through the python technical analysis

(TA) library. The GBE model is imported using the *SkLearn* library while the LTSM neural network is created using the *Keras* framework from the *Tensorflow* library.

5.1 Dataset generation

The first step when planing the design of any machine learning or deep learning project is to determine the information that is needed and explore ways of extracting and processing it.

In the case of this project we are going to use three different data categories as model inputs:

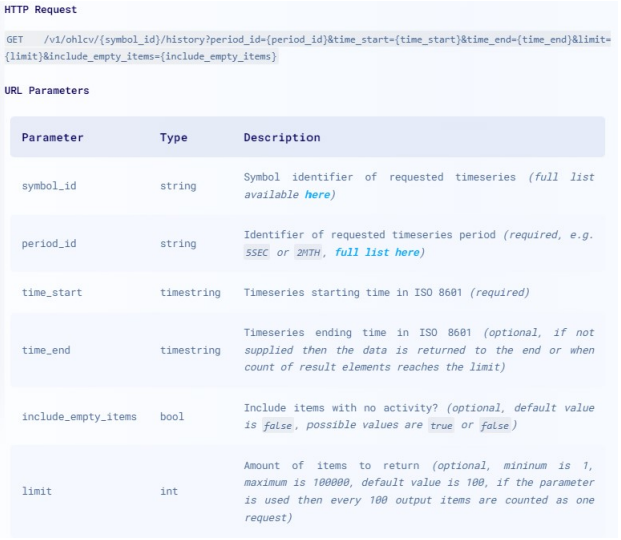
- Cryptocurrency Open - High - Low - Close - Volume (OHLCV) prices
- Cryptocurrency social media sentiment analysis
- Technical analysis indicators

Cryptocurrency prices information are extracted using the online API provided by *CoinAPI*. They offer an online API that returns price information for the cryptocurrency you filter on the request call.

OHLCV stands for **open** as the price of the asset at the beginning of the time period (when the considered period is daily then this refers to the price at the stock market opening hour), **high** as the maximum price the asset reached on the specific time period, **low** as the minimum price the asset reached on the time period, **close** as the price of the asset as the end of the time period and **volume** as the total number of the specific asset traded over the time period.

CoinAPI returns this information as a json object between the time period you choose to extract. The HTTP request for historical data should be configured as shown on *figure* 5.1 where the request parameters are also described.

Form the python code perspective, I request the information using the *request* library extracting the *OHLCV* prices for one day time frame, starting form December 2020 to



HTTP Request

`GET /v1/ohlcv/{symbol_id}/history?period_id={period_id}&time_start={time_start}&time_end={time_end}&limit={limit}&include_empty_items={include_empty_items}`

URL Parameters

Parameter	Type	Description
symbol_id	string	Symbol identifier of requested timeseries *(full list available here)*
period_id	string	Identifier of requested timeseries period *(required, e.g. 5SEC or 2MTH, full list here)*
time_start	timestring	Timeseries starting time in ISO 8601 *(required)*
time_end	timestring	Timeseries ending time in ISO 8601 *(optional, if not supplied then the data is returned to the end or when count of result elements reaches the limit)*
include_empty_items	bool	Include items with no activity? *(optional, default value is false, possible values are true or false)*
limit	int	Amount of items to return *(optional, mininum is 1, maximum is 100000, default value is 100, if the parameter is used then every 100 output items are counted as one request)*

Figure 5.1: OpenAPI historical data GET request

Source: https://www.coinapi.io/

August 2022 and saving the information into a JSON file for backup. The *listing* 5.1 shows the code used at retrieving the OHLCV information from the API.

Listing 5.1: OHLCV price retrieval

```
url = "http://rest.coinapi.io/v1/ohlcv/BINANCE_SPOT_ETH_USDC/..."
headers = {'X-CoinAPI-Key' : 'Your_API_Key'}
response = req.get(url, headers=headers).json()
with open('ETH_USD.json', 'w') as json_file:
    json.dump(response, json_file)
```

As for the starting and ending period I took the decision to filter between those two dates based on two reasons. As can bee seen in *figure* 5.2, starting form December 2020 is when cryptocurrency became a trend that gained people interest and its price shifted form a smooth slowly increase to a very volatile market where prices suffer big drops and raises in a very short period of time. In addition, cryptocurrency market is a recent market where new coins are published on a monthly bases, thus, if we want this project investigation to be useful at investing in other coins different than ethereum, it is helpful that we filtered the retrieval of prices starting at the point where the market gained people interest, since this involves that new coins are published after this date and a proof of this is that we can observe two clearly different market behaviors. A grow in the price is generated when

more people is interested to buy than people interested to sell, the opposites applies when there is a fall of the price. If we can see spikes, this means that peoples interests quickly changes in a short period of time, and this can only happen so sharply when there are other alternatives in the market. What is more, the market data previous December 2020 looks like outlier when comparing it to nowadays market behavior.

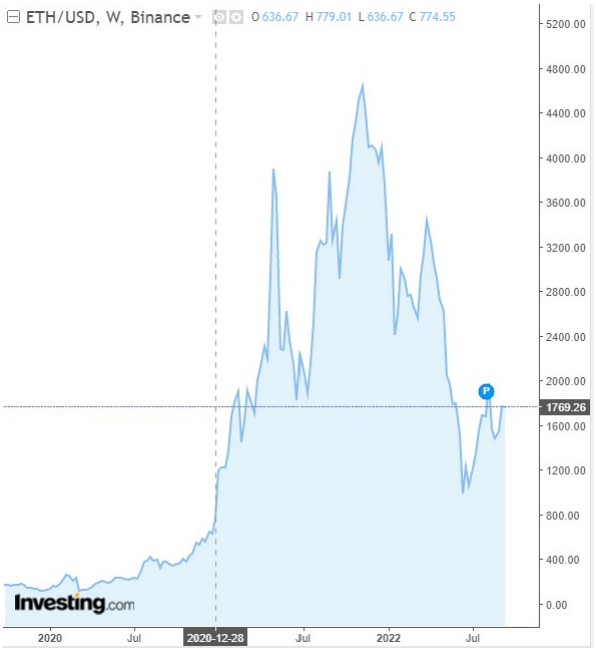


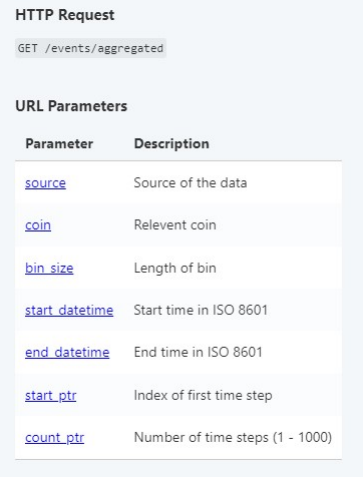
Figure 5.2: Daily ethereum price graph for 3 years period

Source: https://www.investing.com/

As for the chosen end date, the reason is that CoinAPI has a pricing over its online API to consume up to date information, offering free data retrieval for market data which is outdated by more than 30 days from the day you make the request. Therefore, I have chosen the Sunday 7^{th} of August 2022 as the end time and Monday 28^{th} of September as the start date in order to have our datases divisible in whole weeks. You will notice that the request indicates the 8^{th} of august as the end date, but this is due the fact that we will need to shift our target variable one day up for the gradient boosting technique so that we use the previous date indicators to predict the next day close price.

For cryptocurrency social media indicator, I used the Augmento online API that returns the number of news published about a specific cryptocurrency over the indicated time

frame, being the news already classified for many specific sentiments. *Figure* 5.3 shows the needed request parameters while the return of the API is a list of int numbers on each position, where each list position represents a specific sentiment while the int number in that position represents the number of news that have been published in relation with the sentiment. In our case, we are downloading the number of news published on a daily period between December 2020 and August 2022 for consistency with the OHLCV data.



HTTP Request

`GET /events/aggregated`

URL Parameters

Parameter	Description
source	Source of the data
coin	Relevent coin
bin_size	Length of bin
start_datetime	Start time in ISO 8601
end_datetime	End time in ISO 8601
start_ptr	Index of first time step
count_ptr	Number of time steps (1 - 1000)

Figure 5.3: Augmento sentiment data GET request

Source: https://www.augmento.ai/

The python code in listing 5.2 is used at requesting *Augmento* sentiment data, which returns an array of size 93 for each day between the start date and the end date. To filter the most revelant sentiments form the 93 returned, we plot the correlation matrix as in *figure* 5.4.

Listing 5.2: Sentiment API request

```
SentimentUrl = "http://api-dev.augmento.ai/v0.1/events/aggregated"
SentimentParams = {
  "source" : "twitter",
  "coin" : "ethereum",
  "bin_size" : "24H",
  "count_ptr" : 600,
  "start_ptr" : 0,
```

```
    "start_datetime" : "2020-12-27T00:00:00Z",
    "end_datetime" : "2022-08-08T00:00:00Z",
}
SentimentResponse =
    req.get(SentimentUrl, params=SentimentParams).json()
```

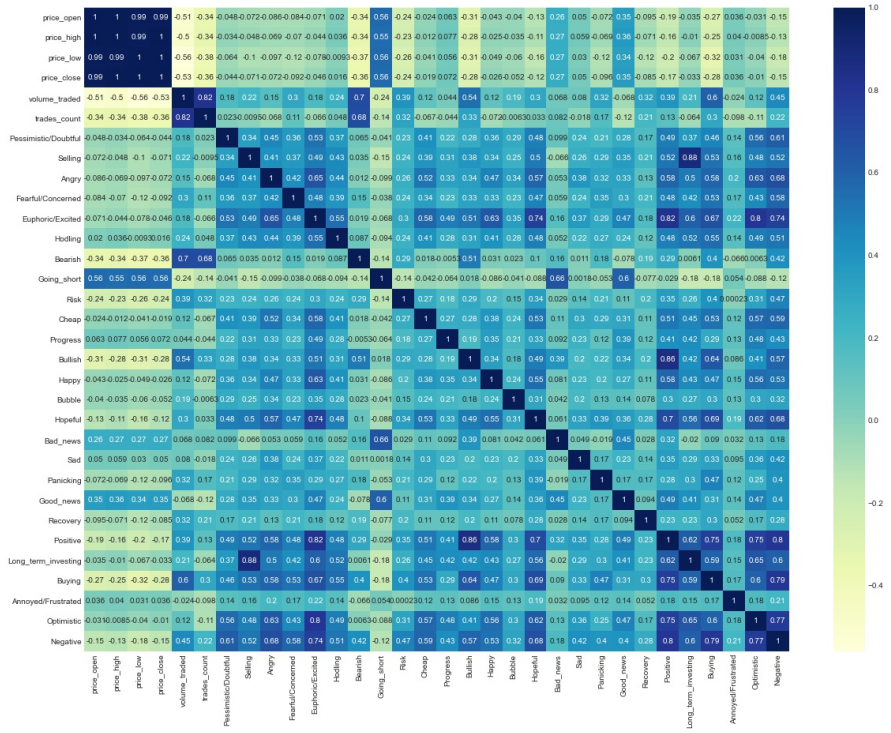


Figure 5.4: Correlation Matrix Prices - Sentimet

Including all the sentiments as features to our model would mean having too many entries data points so we first decide to check the correlation matrix where we see that there are many variables which have almost no correlation with others except themselves. Since we have a linear time series problem, because we would like to predict the price of an asset over time, we filter the data such that we will only use sentiments that are at least 75% correlated to any other variable. The code in lisitng 5.3 was used to perform this filtering, which also eliminates the no correlated features from the dataset.

Listing 5.3: Filtering most correlated sentiments

```
#Filtering for highly correlated features
corelation = dfData.corr()
corelation = corelation[corelation.abs()>0.7]
corelation[corelation == 1] = np.nan

#Dropping all the rows and all the columns with low correlation
corelation.dropna(axis = 0, how='all', inplace = True)
corelation.dropna(axis = 1, how='all', inplace = True)

#Extracting the highly correlated features name
rows = list(corelation.index)
cols = list(corelation.columns)
columns = list(set(rows+cols))

#Limiting our data set just to highly correlated features
dfData = dfData[columns]
```

The last item which we are going to add to our dataset are technical indicators. As Ortu et al. (2022) states, technical indicators are pattern based signals produced by applying mathematical formulas over the *OHLCV* set. We use the python library *Technicall Analysis (TA)* which can easily be installed through pip.

As can be seen in the libraries official *documentation*, we have the possibility to add Momentum, Volume, Volatility and Trend indicators to our dataset as features which will be model inputs. Following the article created by Ortu et al. (2022), I am adding the following technical indicators to our dataset as they proved to have the best results when predicting cryptocurrencies:

- Relative Strength Index (RSI), which compares the magnitude of recent gains and looses to measure the speed and change of prices of an asset.
- Stochastic Oscilator, which compares the closing price of an asset in relation to its higher and lower price.
- Bollinger Bands, which are calculated from the Simple Moving Average (SMA) and

applying two standard deviations (positive and negative) over the SMA.

- Moving Average Convergence Divergence (MACD), which calculates the relationship between two simple moving averages.
- Vortex indicator (VI), which are two oscilators that capture possitive and neggative trends of an asset price.
- Ease of Movement, which relates an asset price change in comparison or its volume.

The code in listings 5.4 adds those features (technical indicators) to our data set.

Listing 5.4: Adding technicall analysis indicators to the dataset

```
rsi ta.momentum.RSIIndicator(dfData["price_close"],
                             window = 7,
                             fillna = False)
dfData["rsi"] = rsi.rsi()

bolinger = ta.volatility.BollingerBands(dfData["price_close"],
                                        window = 7,
                                        window_dev = 1,
                                        fillna = False)
dfData["bolinger_high"] = bolinger.bollinger_hband()
dfData["bolinger_low"] = bolinger.bollinger_lband()
dfData["bolinger_middle"] = bolinger.bollinger_mavg()

macd = ta.trend.MACD(dfData["price_close"],
                             window_slow = 14,
                             window_fast = 4,
                             window_sign = 7,
                             fillna = False)
dfData["macd"] = macd.macd()

dfData["vortex"] =
        ta.trend.vortex_indicator_neg(high=dfData["price_high"],
                                      low=dfData["price_low"],
```

```
                                                close=dfData["price_close"],
                                                window=7,
                                                fillna=False)

stoch =
      ta.momentum.StochasticOscillator(high=dfData["price_high"],
                                                low=dfData["price_low"],
                                                close=dfData["price_close"],
                                                window=7, smooth_window = 3,
                                                fillna = False)
dfData["stoch"] = stoch.stoch()

dfData["movement"] =
                  ta.volume.ease_of_movement(high=dfData["price_high"],
                                                low=dfData["price_low"],
                                                volume = dfData["trades_count"],
                                                window=7,
                                                fillna=False)
```

The end result its a pandas dataframe with dates as index and each of the above explained variables as columns. See *figure* 5.5. The dataset is exported to a csv file which will be later be used and transfomed in such a way that the models can be trained.

```
dfData
```

time_period	volume_traded	Bullish	Positive	Negative	Hopeful	Euphoric/Excited	Long_term_investing	price_open	trades_count	Selling	...	Optimistic
12/28/2020	25763.51754	89	166	122	12	27	79	684.42	15333	35	...	44
12/29/2020	16248.29869	56	144	133	10	13	53	729.02	11362	26	...	35
12/30/2020	14385.47579	66	101	53	7	19	36	731.79	9679	16	...	26
12/31/2020	18938.33743	48	112	89	12	20	39	752.50	9639	11	...	35
01/01/2021	15129.68596	24	83	74	3	16	28	736.90	7544	12	...	28
...	...	...	...	...	...	...	...	...	...	...	...	...
08/03/2022	62869.24340	92	133	103	2	14	28	1631.15	101102	10	...	17
08/04/2022	52963.42630	94	115	84	5	11	21	1618.50	85404	10	...	16
08/05/2022	58457.61080	110	129	85	7	18	26	1608.05	91572	12	...	20
08/06/2022	32432.65890	77	103	82	4	8	18	1737.14	46956	8	...	22
08/07/2022	31401.66210	68	102	79	7	9	14	1691.01	43537	8	...	14

588 rows × 23 columns

Figure 5.5: Data Set in Pandas

5.2 Gradient Boosting Ensemble Model for Time Series Predictions

As stated in Chen and Guestrin (2016), gradient tree boosting is a machine learning technique that achieved state of the art results in many applications. They state that Extreme Gradient Boosting (XGBoosting) was present on 17 out of the 29 winning solutions of Kaggle competition in 2015.

Gradient Boosting refers to a combination of simple individual models which is called an ”ensemble” that usually outputs better result than the ensembled models individually. It relies on the perception that the next model will minimize the prediction error of the previous. To join the models, a differentiable loss function and a gradient descend optimization algorithm is used giving the model its name since the loss gradient is minimized as the models are joined, similar to an artificial neural network. See *figure* 5.6. We call ”weak models” those that are ensembled. At each training iteration, additional weak models are being added to the ensemble.

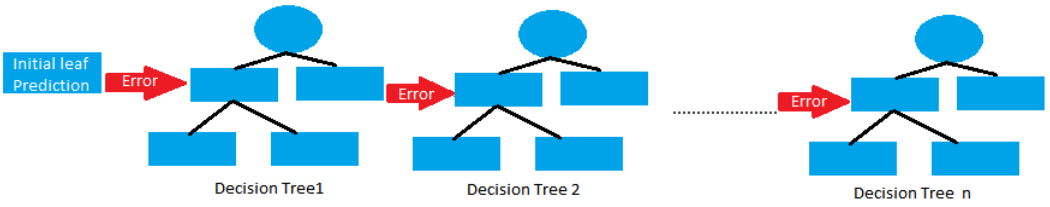


Figure 5.6: Gradient Boosting

Source: Derbentsev et al. (2021)

Those weak models that are being ensembled to form the gradient boosting ensemble are decision trees. While training, new trees are being added to the model without modifying the already existing ones while computing the loss. Those additive models must be added in such a way to reduce the lose of the ensemble.

Decision trees, as their name suggest, are machine learning algorithms whose structure follows a tree like structure where we can differentiate a root node, branches (a subsection of the entire tree), internal nodes and leaf nodes (nodes that do not split) as in *figure* 5.7.

The three starts at the roof node and splits at each internal node based on simple decision which are inferred from the dataset features.

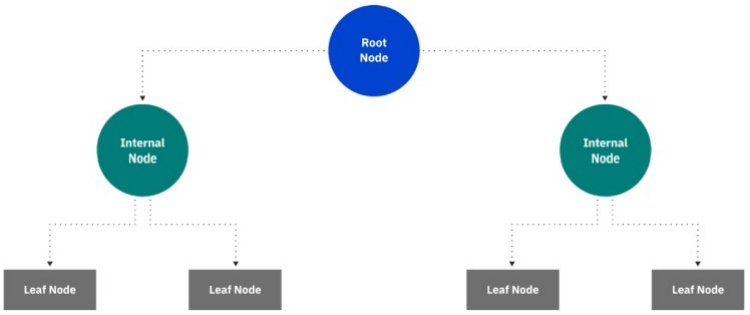


Figure 5.7: Gradient Boosting

Source: www.ibm.com

When an input is given, the tree sorts it down towards some leaf node which is the one providing the output. Wen a tree grows to much, it can result in too little information falling through certain branch which is known as fragmentation. When this occurs, the process of pruning removes the branch form the tree in order to avoid over fitting.

Decision trees models are designed for both classification problems and regression problem so is the gradient boosting technique. In order to use a gradient boosting model for our cryptocurrency prediction we need to transform the data in such a way that we transform our problem to a regression.

This consists on shifting the target variable one cell forward in our dataset, so that we have a new column which is the price that the model has to predict based on the previous time step input. As can be seen on *table* 5.1, we have the target variable at day 1 with all the features that we collected for that specific day. We want the model to predict the next day target, thus, using the features at day 1, the prediction of the model should be the day 2 price which can be found under the ”Shifted Target” column on the day 1 row.

In this way we transform the time series problem into a supervized learning problem where the target variable is the shifted target variable column. Taking the example to our price prediction problem, using the sentiments, OHLCV and technical indicators of yesterday, we want to predict today price. In python, this can be achieved using the

Time	Feature 1	Feature 2	Target	Shifted Target
Day 1	10	40	100	101
Day 2	20	50	101	102
Day 3	30	60	102	103
Day 3	30	60	103	104

Table 5.1: Target shifting for XGBoost time series

.SHIFT() function of the *pandas* library as in the lisitng 5.5

Listing 5.5: Shifting the dataset to create the target for GBE model

```
dfData["next_price_close"] = dfData['price_close'].shift(-1)
print(len(dfData))
dfData[["price_close", "next_price_close"]]
```

The target variable for our model will be the "next_price_close" column, meaning that with the information form day 1, we are trying to predict the price on day 2. First of all we will need to split our dataset between training and testing. I took the last four weeks, 28 days, for testing and the rest of the data for training the model as can be seen in the listing 5.6.

Listing 5.6: Splitting test and training set for GBE

```
prediction_days = 28
df_train = dfData[:len(dfData)-prediction_days]
df_test = dfData[len(dfData)-prediction_days:]
```

To prepare the inputs for our model, we need to split the features form the target. This is done in the lisitng 5.7 where I am getting all the column names from our dataset as a list, removing the target variable name form the list and forming the train features dataset "x_train", the test features dataset "x_test", the train target "y_train" and the test target "y_test".

Listing 5.7: Splitting features from target

```
columns = list(dfData.columns)
target = 'next_price_close'
```

```
columns.remove(target)

x_columns = columns
y_column = [target]

x_train = df_train[x_columns]
y_train = df_train[y_column]

x_test = df_test[x_columns]
y_test = df_test[y_column]
```

Now we are ready to create our model, train it and predict the price of the last 28 days of the dataset which we kept for testing.

For this project I am going to use the Scikit Learn implementation of the gradient boosting algorithm which is available through the open source python library *SkLearn*. We first need to import it from the SkLearn library to our IDE and it can be easily instantiated, trained and used to predict as shown in the listing 5.8.

Listing 5.8: Creating the GBE model

```
import sklearn
from sklearn.ensemble import GradientBoostingRegressor

n_estimators = 500
random_state = 5

model = GradientBoostingRegressor(
            n_estimators=n_estimators,
            random_state=random_state)

model.fit(x_train, np.ravel(y_train))
predictions = model.predict(x_test)
```

Above, we instantiated the model passing only the mandatory hyperparameters, which

means all the other available hyper parameters will be set as default. The library offers many option to personalize our model. We will explore this on the section 5.4.1, where we cover the outputs generated by the Gradient Boosting model. You can check the official documentation for this model in the following link.

5.3 Long Short Term Memory model for time series prediction

Long short term memory artificial neural networks are a type of recurrent neural networks (RNNs) that perform well specially in sequence prediction problems. As stated in Sherstinsky (2020), recurrent neural networks are a class of feed forward neural networks where information cycles through a loop which grants the network with the ability to remember its past inputs and take the next prediction based on the that memory in addition to the input. *Figure* 5.8 represents a recurrent neural network where the learning that the model had with the input X_0 is kept and used when the model predicts has the input X_1, and so on. Therefore, when predicting the value h_t, the model used the input X_t and also the learning it had from the input X_{t-1}.

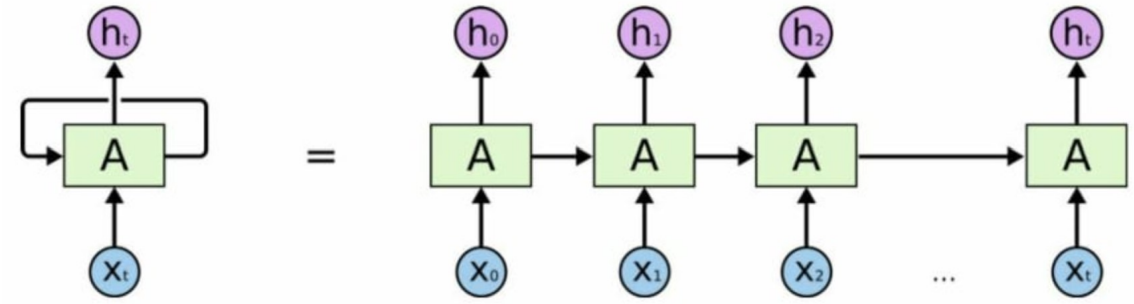


Figure 5.8: Recurent Neural Networks

Source: https://colah.github.io

This learning that the model have form the previous predictions is called **hidden state** or **hidden variable**. The hidden state is updated at each prediction and then used as input for the next prediction, being the variable that gives the concept of memory to the model.

RNNs suffer from the vanishing/exploding gradient problem, which consists on making the error even bigger (exploding) or smaller (vanishing) at the step t when we failed to properly update the weigths at step $t-1$. Long Short Term Memory artificial neural

networks are intended to solve this issue. As can be observed in the *figure* 5.9, long short term memory networks is based on structuring the feed forward network of the RNN in a very specific way, designing it with 3 interacting layers of neurons that are called gates.

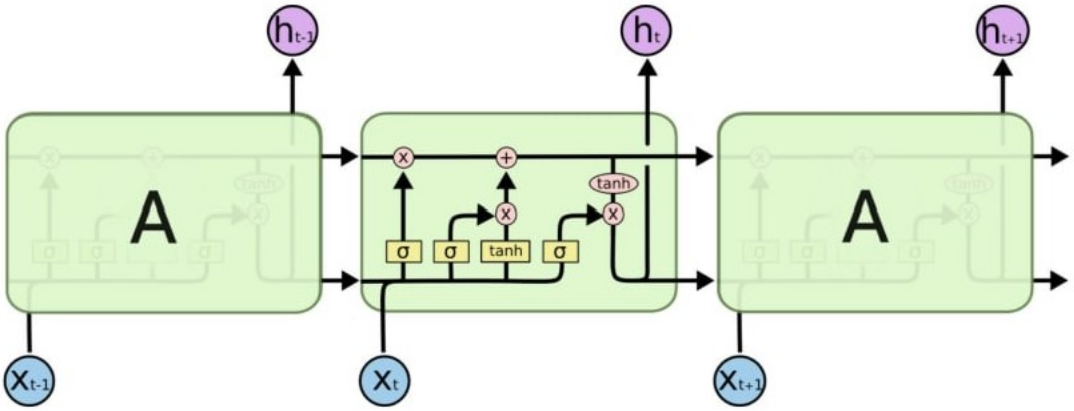


Figure 5.9: Long Short Term Memory Neural Network

Source: https://colah.github.io

LTSMs introduces the concept of the **cell state**, being this a new variable whose function is to decide what information should be propagated to the next predictions. When inputs are being pushed through the LSTM model, it first calculates what information form the cell state variable should be kept, doing this based on the input X_t and the previous hidden state h_{t-1}. This is done by the "forget gate". Afterwards, it decides what information form the actual input should be ignored filtering through the "input get" and uses that information at updating the cell state variable. Finally, it delivers a prediction based on the recently calculated cell state through the "output gate". Those three gates are sigmoid activated, meaning their output are values between 0 and 1, being this the reason LSTM models does not suffer from the gradient vanishing/exploding problems.

The cell state variables gives LSTM the ability to decide what information to store form previous inputs, which improves over time wile training the model and adjusting the weights of the forget layer. As Luo et al. (2021) states, LSTM achieves state of the art results on a large variety of AI problems.

Going straight to our model implementation, the first thing we need to do is to

transform and adjust the data in such a way that we can input the and train the model. LSTM model expects as feature input a three dimensional array with the following structure $[number\_of\_samples, sequence\_length, number\_of\_features]$. The *figure* 5.10 shows a three sequence length LTSM model. Time steps 0 to 2 are used as inputs for the first prediction which corresponds with the target at time step two, on the next iteration, time steps 1 to 3 are used as inputs and the second prediction corresponds with the target at time step three, and so on till we go through all the training samples.

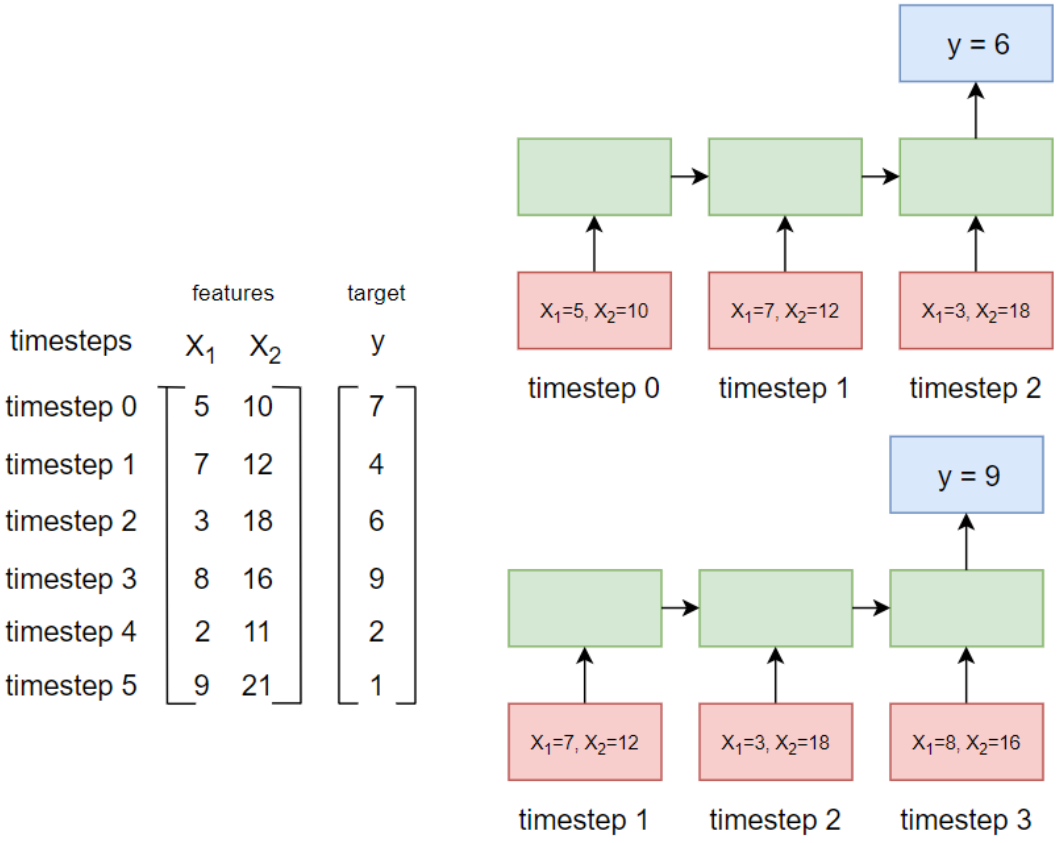


Figure 5.10: LSTM Inputs / Output

Source: https://medium.com/

In our case, we are going to use a 7 days sequence length model, to cover a whole week of data. The first step is to split between the training and testing set, similar to what we did at the gradient boosting model. In order to keep coherence with the GBE model we will use the last 4 weeks for testing but in order to predict the price on the first day of those last 28, the model will need as inputs the features for the 7 days prior this last date,

thus I am adding 7 extra days to the testing set such that the first 7 days information is going to be used at predicting the price for the 1^{st} day of the 28^{th} that we want to predict, see the lisitng 5.9.

Listing 5.9: Splitting test and train dataset for LSTM

```
prediction_days = 28
df_train = dfData[:len(dfData)-prediction_days]

# Adding one week
df_test = dfData[len(dfData)-prediction_days+7:]
```

After splitting the dataset, we need to shape the information in vector of shape $[samples, sequence, features]$ for the features and in the shape $[samples, num\_targets]$ for the target. I did this in several steps, first defining a function which transforms the data set into a list of lists where each inner list represent a feature and contains the 7 days information of it, while for the target variable it just forms an array with the 7 first days shifted as in listing 5.10.

Listing 5.10: Creating training and testing list for LSTM

```
def create_train_test_list(df_train, df_test, target, window):
# Target: Columns name that is the target to predict
# Window: Sequence length of the model

    target_index = df_train.columns.get_loc(target)
    num_cols = len(df_train.columns)
    trainList = []
    testList = []

    for index in range(num_cols):
        inputs = []
        for i in range(0, df_train.shape[0]-window):
            if index == target_index:
                inputs.append(df_train.iloc[i+window, index])
            else:
                inputs.append(df_train.iloc[i:i+window, index])
```

```
        trainList.append(inputs)

    for index in range(num_cols):
        inputs = []
        for i in range(0, df_test.shape[0]-window):
            if index == target_index:
                inputs.append(df_test.iloc[i+window, index])
            else:
                inputs.append(df_test.iloc[i:i+window, index])
        testList.append(inputs)

    return trainList, testList
```

The second step consists on scaling the information of each inner list between a range so the weights of the model can stay at low values, making it more easy for the model to be trained. See liating 5.11

Listing 5.11: Scalling train and test list for LTSM

```
# This process has been done for both, training and testing lists
 for i in range(len(trainList)):
    if i == target_index:
        trainList[i] = scaler_y.fit_transform(trainList[i])
    else:
        trainList[i] = scaler_x.fit_transform(trainList[i])
```

Lastly, I extract the information form the lists and reshaped it as explained above so it can be used at training and testing the model. See listing 5.12

Listing 5.12: Reshaping train and test list for LTSM input

```
# Extracting and reshaping the target variable arrays
y_train = trainList[target_index]
y_test = testList[target_index]
y_train = np.reshape(y_train, (len(y_train), 1))
y_test = np.reshape(y_test, (len(y_test), 1))
```

```
# Forming the features input arrays
del trainList[target_index]
del testList[target_index]

x_train = np.stack(trainList, axis=2)
x_test = np.stack(testList, axis=2)
```

The LTSM model implementation in python can be instantiated through the Keras framework from the TensorFlow library as seen in the lisitng 5.13. At instantiating the models, layers of LTSM units should be added to a sequential model. It is important to figure out that the input layer of the of the LTSM nework will be defined by the "input_shape" parameter which should be coherent with the input vector, more specifically, the input shape of the neural network should be defined as $[num\_sequences, num\_features]$.

Listing 5.13: Creating training and predicting with the LSTM model

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

#Defining the model
model = tf.keras.models.Sequential([
  tf.keras.layers.LSTM(units=22,
                       return_sequences = False,
                       input_shape=(x_train.shape[1],
                       x_train.shape[2])
                       ),
  tf.keras.layers.Dense(22, activation="relu"),
  tf.keras.layers.Dense(1)
])
model.summary()

# Compiling and fitting the model
model.compile(loss='mean_squared_error',
```

```
                optimizer='adam',
                metrics='mean_squared_error')
model.fit(x_train,y_train,
            alidation_split=0.1, epochs=50,
            batch_size=7)

# Making predictions
predictions = model.predict(x_test)
```

In the above code snippet I created just a basic LSTM neural network with 22 different LSTM units of sequence length 7, added a dense hidden layer in the middle such that it processes the 22 different outputs form the 22 LSTM units and finally one dense output layer consisting of 1 neuron which outputs the predicted price. When it comes to deep learning, we have the freedom to include as many layer as we would like consisting of as many neurons as we would like. In addition, the keras framework allows us to personalize each layer hyper-parameter. We will explore this on the section 4.4.3 where we cover the outputs generated by the LSTM model. You can check the official documentation for Keras in the following link.

5.4 Results

Following the explanation of the used models and having shared how to construct them in our working environment, in this section we will cover each model output based on the customization of the hyperparameters to end this chapter comparing the most accurate output achieved for both of the models.

5.4.1 Gradient Boosting Ensemble Prediction Output

Figure 5.11 shows the hyperparameters of the gradient boosting model which are customizable in the library *SkLearn*. The values indicated for each parameter is the default value, meaning it is the value the parameter will be set up to if not specified otherwise.

Just by letting the model parameters as default, the model already outputs an accurate prediction of the ether price for the last 28 days, from 11^{th} of July till the 7^{th} of August. Please note that we haven't used those 28 days information nor to validate or train the

class sklearn.ensemble.**GradientBoostingRegressor**(*, *loss='squared_error', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0*) ¶ [source]

Figure 5.11: Gradient Boosting Regressor Hyper-Parameters

Source: https://scikit-learn.org/

model, thus, the prediction were based on information has not seen before. As seen in *figure* 5.13, using the default configuration we achieved a model accuracy of 0.79% while a mean absolute error of 69.73 USD. This means that in average, predictions differed around 70 dollars from their rel price, which is not a big error since the prices ranges between 1000 and 1800 dollars in the 28 days period we are testing the model.

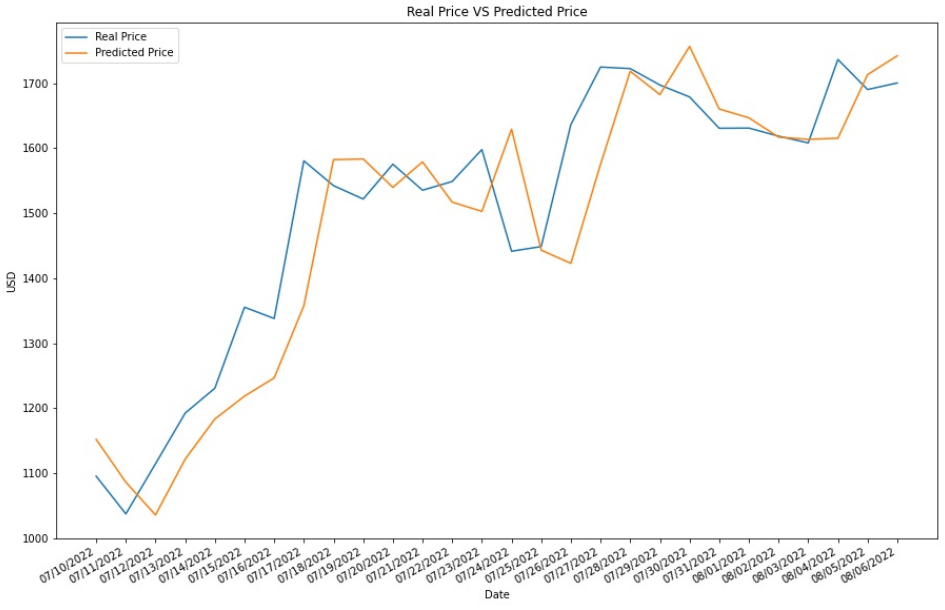


Figure 5.12: Gradient Boosting Output Default Parameters Prediction

```
print("Model Accuracy: %.3f" % model.score(x_test, y_test))
mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
print("The mean absolute error (MAE) on test set: {:.4f}".format(mae))

Model Accuracy: 0.790
The mean absolute error (MAE) on test set: 69.7302
```

Figure 5.13: Gradient Boosting Output Default Parameters Prediction Metrics

As Anghel et al. (2018) covers, the hyperparameters of gradiend boosting enssemble could be divided into three different categories. The following list covers all of them with a short explanation.

- **Three specific parameters** that affects individual trees of the enssemble. The Sklearn model includes the following as most important, *"min_samples_split"*, *"min_samples_leaf"* *"min_weight_fraction_leaf"*, *"max_depth"*, *"max_leaf_nodes"*, *"max_features"*.
- **Boosting Parameters** which affects the boosting operation of the enssemble, the following being the most relevant, *"learning_rate"*, *"n_estimators"*, *"subsample"*.
- **Overall functioning Parameters** which are hyperparameters contributing to the models functioning but not defining its *"loss"*, *"init"*, *"random_state"*, *"warm_start"*, *"presort"*.

As per the learning rate, Islam et al. (2021) states that lower learning rates are preferred since they increases the robustness of the model making it to generalize well, however, this requires highering the number of trees to properly extract all the relation between the input features.

In order to chose a specific number of estimators and number of trees, we will set up the random state of our model in the middle of the price range that the cryptocurrency had over the last 28 days and use the function defined in the listing 5.14 below to print the mean absolute error and the accuracy for a range of estimators and learning rates.

Listing 5.14: Tuning GBE estimators and learning rate

```
def learning_rate_estimators_tunning(learning_rates, estimators):
    for learning_rate in learning_rates:
        for estimator in estimators:
            model = GradientBoostingRegressor(n_estimators=estimator,
                                              random_state = 1400,
                                              learning_rate = learning_rate)
            model.fit(x_train, np.ravel(y_train))
            accuracy = model.score(x_test, y_test)
```

In *figure* 5.14 we can see that the lowest mean absolute error and the highest accuracy was achieved by a learning rate of 0.05 and 1000 number of estimators, thus, we are going to select those values for our final model hyperparameters.

```
LearningRates = [0.1, 0.07, 0.05, 0.03]
Estimators = [300, 500, 700, 1000]
learning_rate_estimators_tunning(LearningRates, Estimators)
```

```
Model Accuracy for 0.1 learning rate and 300 estimators is: 0.793
The mean absolute error (MAE) for 0.1 learning rate and 300 estimators is: 69.7836
Model Accuracy for 0.1 learning rate and 500 estimators is: 0.790
The mean absolute error (MAE) for 0.1 learning rate and 500 estimators is: 70.3836
Model Accuracy for 0.1 learning rate and 700 estimators is: 0.785
The mean absolute error (MAE) for 0.1 learning rate and 700 estimators is: 70.8876
Model Accuracy for 0.1 learning rate and 1000 estimators is: 0.784
The mean absolute error (MAE) for 0.1 learning rate and 1000 estimators is: 70.6453
Model Accuracy for 0.07 learning rate and 300 estimators is: 0.798
The mean absolute error (MAE) for 0.07 learning rate and 300 estimators is: 68.8968
Model Accuracy for 0.07 learning rate and 500 estimators is: 0.805
The mean absolute error (MAE) for 0.07 learning rate and 500 estimators is: 65.9681
Model Accuracy for 0.07 learning rate and 700 estimators is: 0.801
The mean absolute error (MAE) for 0.07 learning rate and 700 estimators is: 66.7967
Model Accuracy for 0.07 learning rate and 1000 estimators is: 0.797
The mean absolute error (MAE) for 0.07 learning rate and 1000 estimators is: 67.0684
Model Accuracy for 0.05 learning rate and 300 estimators is: 0.796
The mean absolute error (MAE) for 0.05 learning rate and 300 estimators is: 68.5455
Model Accuracy for 0.05 learning rate and 500 estimators is: 0.808
The mean absolute error (MAE) for 0.05 learning rate and 500 estimators is: 66.4689
Model Accuracy for 0.05 learning rate and 700 estimators is: 0.814
The mean absolute error (MAE) for 0.05 learning rate and 700 estimators is: 64.0886
Model Accuracy for 0.05 learning rate and 1000 estimators is: 0.816
The mean absolute error (MAE) for 0.05 learning rate and 1000 estimators is: 62.6732
Model Accuracy for 0.03 learning rate and 300 estimators is: 0.806
The mean absolute error (MAE) for 0.03 learning rate and 300 estimators is: 67.5670
Model Accuracy for 0.03 learning rate and 500 estimators is: 0.805
The mean absolute error (MAE) for 0.03 learning rate and 500 estimators is: 67.8261
Model Accuracy for 0.03 learning rate and 700 estimators is: 0.808
The mean absolute error (MAE) for 0.03 learning rate and 700 estimators is: 66.7611
Model Accuracy for 0.03 learning rate and 1000 estimators is: 0.811
The mean absolute error (MAE) for 0.03 learning rate and 1000 estimators is: 66.2812
```

Figure 5.14: Accuracy and MAE at different Learnign Rates and Estimators for GBM

Other model hyperparameters that we are going to estimate are the *"min_samples_split"* which indicates the minimum number of samples required in a node to be considered for splitting, the *"min_sample_leafs"* which indicates the minimum number of samples required in a terminal node and the *"max_depth"* which is the maximum depth allowed in internal trees.

First, we are going to compare the metrics for different "max_depth" variable in order

to set the number for this parameter. A higher depth in internal trees would allow the model to to learn more attributes realted to specific samples, thus we are not going to decrease form the default value which is 3, but instead we will test a range between 1 to 9 with steps of size two. We can see in *figure* 5.15 that increasing or decreasing the value of the default value of "max_depth", which is set to 3, lowers the accuracy of our model, thus, we are keeping this value as our final choice.

```
for maxDepth in range(1, 10, 2):
    model = GradientBoostingRegressor(n_estimators=1000,
                                      random_state = 1400,
                                      learning_rate = 0.05,
                                      max_depth=maxDepth )
    model.fit(x_train, np.ravel(y_train))
    accuracy = model.score(x_test, y_test)
    mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
    print("For a depth of size {}, accuracy is {:.3f} and mae is: {:.4f}".format(maxDepth,accuracy, mae))
```

```
For a depth of size 1, accuracy is 0.784 and mae is: 73.2350
For a depth of size 3, accuracy is 0.816 and mae is: 62.6732
For a depth of size 5, accuracy is 0.675 and mae is: 82.0411
For a depth of size 7, accuracy is 0.559 and mae is: 105.3699
For a depth of size 9, accuracy is 0.487 and mae is: 111.1284
```

Figure 5.15: Accuracy and MAE at different max_depth values

Lastly, we use a function similar to the one the listing above to test model parameters when we change the values for the *"min_samples_split"* and *"min_sample_leafs"* parameters. The default values for them are 2 and 1 respectively, which is the minimum value they could have since a split cannot consist in lower than two elements and a leaf node cannot have less than one element. Looking at the *figure* 5.16 we cans see that increasing any of those two values does not increase the accuracy nor the mean absolute error of our model, thus, we will also keep those variables as default.

Finally, we end up with a gradient boosting model whose parameters are as defined by default by the library, except for the **learning_rate** which is set to 0.05, the *n_estimators* is set to 1000 and the *random_state* which is set to 1400 because this value sits in the middle of the price range that the ether token variates.*Figure* **??** we can see the model achieved an accuracy of more than 80% on our test data while the mean absolute error is slightly higher than 62 USD. Those are very good results considering the price variation of the etherium cryptocurrency over the 28 days of testing had a constant change of value of several times hundreds. *Figure* 5.17 shows the model final prediction, we can see the predicted curve closely follows the real price curve thus I think the model is suitable for designing a trading strategy based on weekly movements.

```
for minSplit in range(2,5):
    for minLeaf in range (1,5):
        model = GradientBoostingRegressor(n_estimators=1000,
                                          random_state = 1400,
                                          learning_rate = 0.05,
                                          min_samples_split=minSplit,
                                          min_samples_leaf=minLeaf)
        model.fit(x_train, np.ravel(y_train))
        accuracy = model.score(x_test, y_test)
        mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
        print("For a min split of size {} and min leaf of size {}, accuracy is {:.3f} and mae
```

```
For a min split of size 2 and min leaf of size 1, accuracy is 0.816 and mae is: 62.6732
For a min split of size 2 and min leaf of size 2, accuracy is 0.801 and mae is: 67.4993
For a min split of size 2 and min leaf of size 3, accuracy is 0.801 and mae is: 64.8576
For a min split of size 2 and min leaf of size 4, accuracy is 0.802 and mae is: 67.4073
For a min split of size 3 and min leaf of size 1, accuracy is 0.801 and mae is: 66.7934
For a min split of size 3 and min leaf of size 2, accuracy is 0.801 and mae is: 67.4993
For a min split of size 3 and min leaf of size 3, accuracy is 0.801 and mae is: 64.8576
For a min split of size 3 and min leaf of size 4, accuracy is 0.802 and mae is: 67.4073
For a min split of size 4 and min leaf of size 1, accuracy is 0.777 and mae is: 70.6186
For a min split of size 4 and min leaf of size 2, accuracy is 0.801 and mae is: 67.4993
For a min split of size 4 and min leaf of size 3, accuracy is 0.801 and mae is: 64.8576
For a min split of size 4 and min leaf of size 4, accuracy is 0.802 and mae is: 67.4073
```

Figure 5.16: Accuracy and MAE at different ”min_samples_split” and ”min_sample_leafs” values

Figure 5.17: Gradient Boosting Ensemble Prediction, using tunned hyper-parameters

5.4.2 Long Short Term Memory Prediction Output

In the case of long short term memory neural networks, the model is defined by adding layers to the *Keras* sequential class instance. The first decision that we have to take is determining how many hidden layers should our LTSM network include. Basing my decision on the work performed by Yadav et al. (2020), I will keep the result to just one hidden layer since as the authors state, "the recommendation here would be to go with one hidden layer for most problems due to better accuracy, easier training and less risk of overfitting. If there is a problem that is very complex and cannot be efficiently modelled with one layer, one may want to go with higher number of layers but such cases are expected to be rare".

There are several hyper-parametrs which we should be considered when designing a LSTM netork, I list below the most relevant which needs to be taken into consideration:

- *Number of hidden layer.*
- *Number of units*, the number of LTSM units in the LTSM layer.
- *Number of units in the dense layer.*
- *Dropout*, which is a layer that randomly deactivates neurons in hidden layer to avoid overfitting.
- *Weight initialization*
- *Activation function*
- *Learning rate*
- *Number of epochs* which represents the number of times that the model is going to be trained using the same dataset.
- *The batch size*, which is the number of samples from the dataset that we are going to input in the model at once.

Creating a suitable deep learning model is more complex than creating a machine learning one mainly because there is not a limit on how many hidden layers the model should have nor the numbers of neurons that each layer should contain. In addition, there are several more customization parameters for each layer apart form the number of

neurons such like choosing different activation functions and deciding weight initialization.

As stated above, our neural network will contain only one LTSM hidden layer and one dense hidden layer which will help at normalizing the size of the inputs towards our ouptut layer which consist on only one neuron that will submit the output, which is the price of the cryptocurrency for the next day of study.

The first parameter we will focus on setting are the number of epochs since we will follow the work done by Vrskova et al. (2021) and utilize the Keras early stopping class which will stop the training process if it detects that the loss of the model is no longer decreasing with each epoch. The early stopping class can be instantiated as in the *lisiting* 5.15 and passed to the model .FIT() function through the CALLBACKS argument.

The class allows the configuration of several parameters such like the value to monitor for deciding to stop training, a minimum delta of the monitored value to consider it as an improvement and the patience which indicates how many epochs will be allowed without improvement before deciding to stop.

The usual parameter to monitor is the validation loss, which means that from our training set we keep a percentage of the samples for validating the model output. In our case, we have a small testing set so if we decide to split a percentage of it for validation we will end up with even less training samples since the validation samples are not used at training and form what I have tested this variable would stop training the model too early, thus, instead of monitoring the validation loss we will monitor the loss of the model.

This is tricky because the loss of the model should always be decreasing since it is fitting the weights to the training dataset features, which could lead to overfitting. In order to avoid this situation, we set a small patience of 3 epochs and we set a delta of 0.1, so if the model loss hasn't decreased more than 0.1 for 3 epochs training will be stopped.

Listing 5.15: LSTM early stopping callback

```
early_stopping = keras.callbacks.EarlyStopping(monitor='loss',
                                               patience=3,
                                               min_delta=0.01)
```

To create the initial model, we minimally need to set the number of units in the LSTM hidden layer and the dense hidden layer of our model. Since we have 22 input features in our dataset, I am initially setting the model dense layers to 22 units, one per feature, and I will double the ammount of units in the LTSM layer to 44 since it has to extract relations form the dataset features.

The initial model prediction can be seen in *figure* 5.18 were we visualize that the predicted price differs form the real price has a big difference form the predicted price except for the upward trend. The mean absolute error of this model is of 200 USD, which is very high.

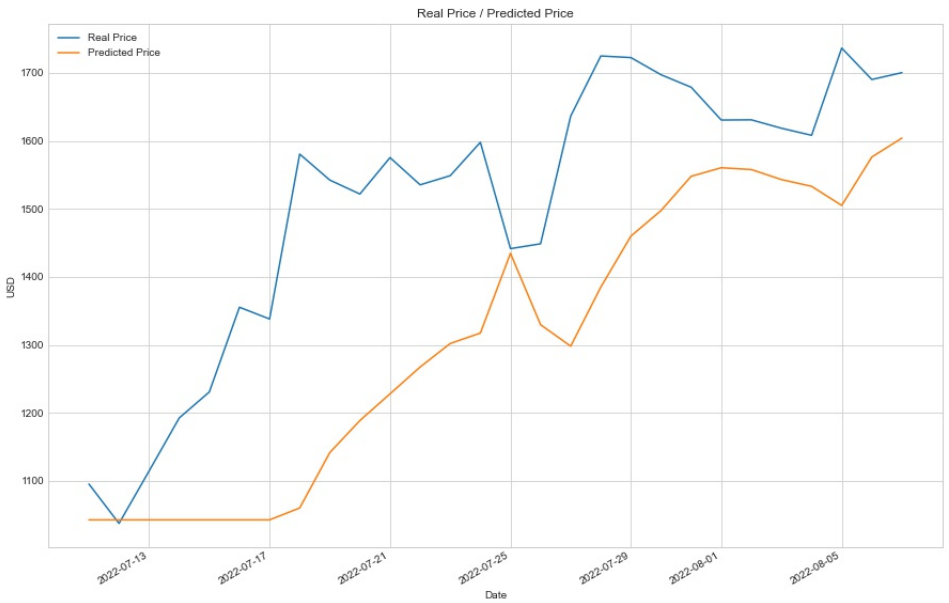


Figure 5.18: LSTM inicial prediction

Looking at the model training process, we saw that because of the early stopping

parameter the model stopped training at 5th epoch out of 50. Therefore we decided to increase the patience of the early stopping argument to 5 epoch and avoid the ussage of a minimum delta. This lead to a 33 epoch training out of 50 and a mean absolute error of 96 USD. A big improvement, the error could even get lower if we increase the patience of the early stopping class, but that would dangerously lead us to over fitting, thus we won't alter that class more.

In order to tune the number of neurons, the strategy which we will follow is the following. We first increase the number of neurons on the LSTM measuring the mean absolute error with a FOR _ IN RANGE () loop, and we will do the same for the number of neurons in the dense layer. At first instance, we will let the number of neurons of the other layer as in the initial model, so we can compare the effects of increasing the number of neurons in each layer starting form the basis of our initial model. The code used for this task can be seen in the *listing* 5.16 and the median absolute error evolution in tables 5.2 and 5.3.

Listing 5.16: For loop for measuring the impact of increasing the num. of neurons

```
for neurons in range(20,500,20):
    model = tf.keras.models.Sequential([
     tf.keras.layers.LSTM(units=neurons, return_sequences = False,
                          input_shape=(x_train.shape[1],
                                       x_train.shape[2])),
    tf.keras.layers.Dense(24, activation="relu"),
    tf.keras.layers.Dense(1)
    ])
    model.compile(loss='mean_squared_error',
                  optimizer='adam', metrics='mae')
    model.fit(x_train,y_train,
              epochs=50, batch_size=2,
              callbacks=[early_stopping], verbose=False)
    predictions = model.predict(x_test)
    unscalledPred = predictions.tolist()
    unscalledPred = scaler_y.inverse_transform(unscalledPred)
    mae = metrics.max_error(unscalledTest, unscalledPredictions)
```

```
    maxError = metrics.mean_absolute_error(unscalledTest,
                                           unscalledPredictions)
    print("METRICS FOR LTSM {} units and Dense 24 units. MAE:
          {:.3f}, MAX: {:.3f}".format(neurons, mae, maxError))
```

LTSM Layer	Dense Layer	MAE (USD)	Max Error (USD)
20	24	170	445
100	24	114	389
140	24	110	396
220	24	95	374
260	24	90	382
320	**24**	**87**	**359**
340	24	124	397
360	24	131	388
380	24	129	403

Table 5.2: Increasing LTSM layer neurons

LTSM Layer	Dense Layer	MAE (USD)	Max Error (USD)
44	20	125	464
44	**40**	**108**	**376**
44	60	124	429
44	80	120	457
44	100	138	431
44	120	137	474

Table 5.3: Increasing dense layer neurons

Comparing both tables, we can see the errors reaches their lowest at 320 layers for the LTSM layer while 40 layer for the dense layer. We did not tested the model metrics combining both values of units in each layer, we cannot assume that the number of neurons obtained above are the most adequate, since their combination could have a different impact on the model output. Following the same strategy as above, we consider a range of neuron values for both layers this time with a lower delta and compare the model metrics

in *table* 5.4.

LTSM Layer	Dense Layer	MAE (USD)	Max Error (USD)
300	36	113	399
300	40	86	355
310	36	96	383
320	36	84	344
320	44	94	355
340	32	86	335
340	32	86	335
350	32	95	400
350	36	95	365
350	**40**	**82**	**341**
350	40	96	351

Table 5.4: Comparing number of neurons impact in both layers

From the results on the different iterations we found that the optimum number of neurons for the LTSM layer is of 350 while for the Dense layer of 40 because it achieves the smallest errors. With this configuration of neurons the model trains for 29 epochs and the predicted output of this middle model, because we will still tune some hyperparameters to reach our final model, can be seen in *figure* 5.19. We visually appreciate that the predicted curve for ether closely match the real price curve when compared with the initial model where we intuitively selected the number of neurons.

Trying to further improve the accuracy of this model, we add a dropout layer between the LTSM and the dense layers with a rate of 0.2, whose function is to randomly set inputs units to 0 preventing over fitting. Another factor which we take into consideration that we have normalized the inputs between 0 and 1 but the activation function of the LTSM neurons as default is the TANH() function whose outputs can be between -1 and 1. Not our inputs nor our outputs could be lower than 0, because the minimum price that an asset can turn is 0, thus, I have also changed the LTSM activation function to the RELU function whose output is just the same as the input but has a less pronounced curve, which means that for a range of inputs the model can extract more relations between the

Figure 5.19: LSTM middle prediction

values since the output of the activation function has less variation based on the input, see *figure* 5.20. In addition, we set the parameter RESTORE_BEST_WEIGTHS form the early stopping class to TRUE so the model get's the weights restored as in the epoch where the loss was the smallest before stopping training.

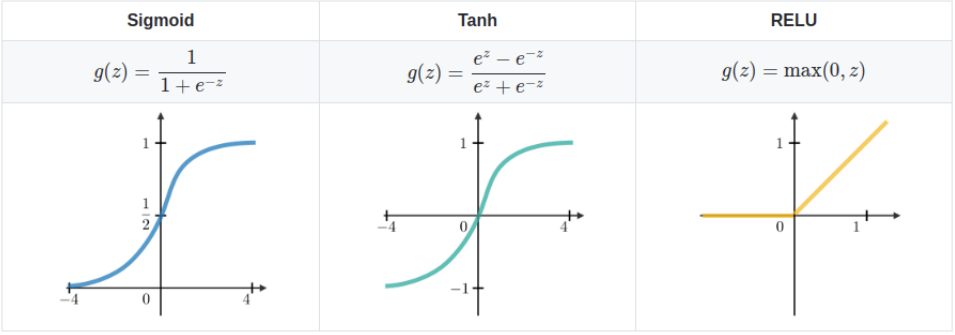


Figure 5.20: Activation Functions Comparison

Source: https://studymachinelearning.com/

The final model that we came out with has the structure shown in *figure* 5.21 and its

output can be seen in *figure* 5.22.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_145 (LSTM)              (None, 350)               522200
_________________________________________________________________
dropout_21 (Dropout)         (None, 350)               0
_________________________________________________________________
dense_288 (Dense)            (None, 40)                14040
_________________________________________________________________
dense_289 (Dense)            (None, 1)                 41
=================================================================
Total params: 536,281
Trainable params: 536,281
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.21: LSTM final model summary

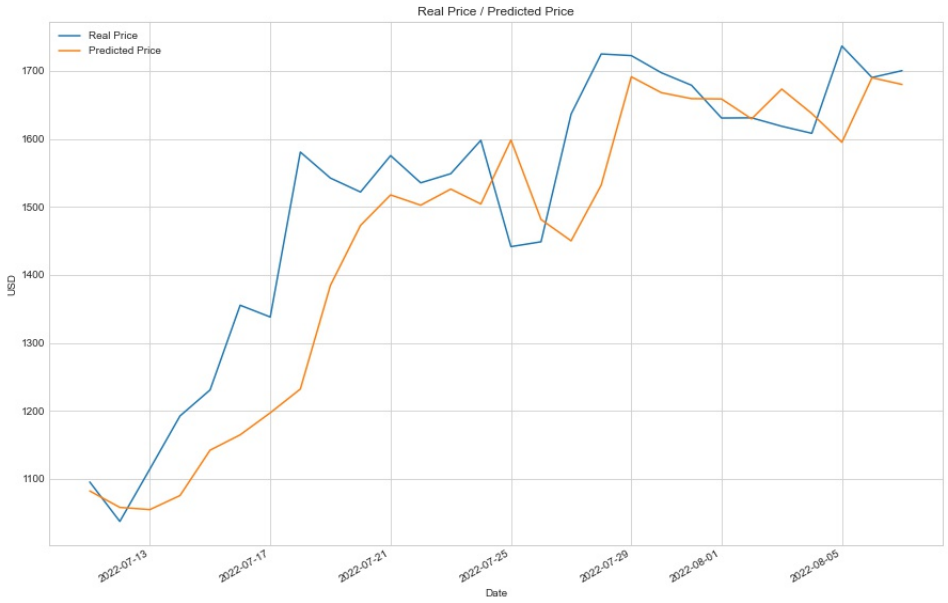


Figure 5.22: LSTM final prediction

We can visually see an improvement between the intial prediction and a small improvement achieved by the dropout layer, the change on the activation functions and the restore weights parameter. Final model prediction achieved a mean absolute error of 79 USD and a maximum error of 344 USDs. Now let us switch to the next section where we compare both outputs.

5.4.3 Output Comparison

The first noticeable aspect it the time difference needed to train one model versus the other. As table 5.5 indicates, it took only 5 seconds and a half to the Grandient Boosting Machine to fit while it took 100 seconds to the LTSM network.

Model	Time for training (seconds)
LTSM	100
GBR	5.5

Table 5.5: Comparing models training time

When it comes to predicting, both models achieved a similar result. In *table* 5.6 we have the real price of the ethereum and the price predicted by each one of the models for the last 10 days of the test set.

Day	Predicted LTSM	Predicted GBR	Real
28 July	1532	1596	1724
29 July	1691	1740	1722
30 July	1668	1690	1697
31 July	1659	1733	1678
1 August	1658	1664	1630
2 August	1629	1639	1631
3 August	1673	1603	1618
4 August	1637	1637	1608
5 August	1595	1623	1736
6 August	1689	1721	1690
7 August	1680	1717	1700

Table 5.6: Comparing predicted vs real prices USD

We can see that both models performed similarly, for example, the 28^{th} of July the real close price was 1724 while both models predicted a close price of the order of 15 hundreds. Looking at *figure* 5.23 where the final prediction of the GBR in green is compared with the final prediction of the LTSM model in orange and the real price in blue, we can see that the green curve follows more closely the blue curve, meaning than from a visual perspective

the machine leanrining technique achieved more accurate results than the deep learning technique even with less computational power required.

Figure 5.23: LSTM final prediction

Let us compare some metrics to quantify the predictions of the models in *table* 5.7. We see that all the GBR model had lower errors than the LTSM model for each one of the considered errors, median absolute, mean absolute and the max error. As expected, the R2 value which measures the variance of the predicted price form the real price, a score of 100% meaning that they are perfectly correlated, is at 80% for the GBR model while only 68% for the LTSM model.

Metric	LTSM Score	GBR Score
Max Error	344 USD	206 USD
Mean Absolute Error	79 USD	63 USD
Median Absolute Error	52 USD	39 USD
R2 Score	0.68	0.8

Table 5.7: Comparing predicted vs real prices USD

Having seen the output achieved by both of the models it is time to switch to the next chapter and draw conclusions and next steps.

Chapter 6

Conclusions and Next Steps

Given our test experiment, the gradient boosting ensemmble achieved better results than the long short term memory network. In fact, the machine learning technique overcome the deep learning approach for all the comparison metrics that we have used, it even did it using less computational power, thus there is no reason in favour of the LTSM that could make us select that model instead of the GBR.

The results were somehow expected, mainly because I have filtered cryptocurrency data since December 2020 till August 2022 using a daily time frame for the OHLCV data, those gave us a dataset of just 588 samples to train an test the model, not enough for a deep learning approach to extract all the needed features from the dataset. As can be seen in *figure* 6.1, machine learning techniques achieves a better performance quicker than the deep learning techniques when it comes to small amount of data while the deep learning approach performance increases and overcoming the ML when big datasets are involved.

However, as we reviewed in in chapter 3 the cryptocurency market had a instantaneous change of behavior starting from December 2020, so I personally think that the decision of filtering data starting from that time period was correct, specially because the results achieved with the Gradient Boosting Ensemble are quite positive, achieving really close price predictions to the real price and because if we want to use this model to predict the price of a newer cryptocurrency there will not be such a long history of data. I strongly believe that a trading strategy that buys and sells several times per week could be designed and it would gave us profits, however, we have to be careful with the model errors such

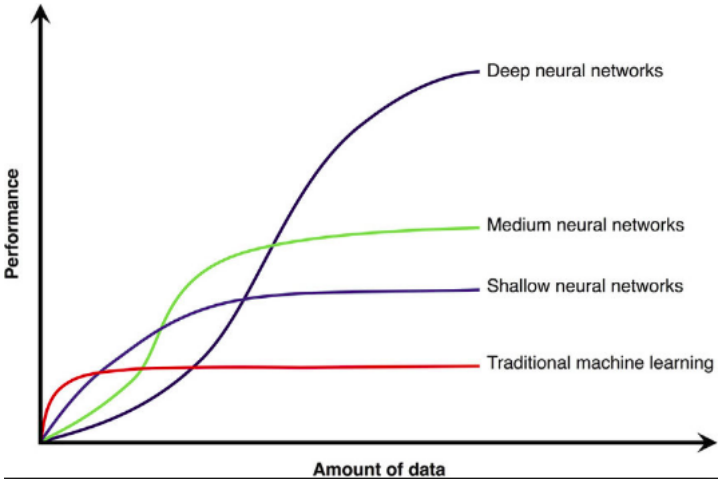


Figure 6.1: ML vs DL performance based on the amount of data

Source: https://medium.com/

as on the 28th of July where the model price prediction was of 1596 USDs while the real price was of 1724 USDs. This is a big price difference between the prediction and the real price, so any trading decision that the model has on that date could lead to the lose of the profits achieved on previous dates, thus, the strategy should be well designed. We have to consider the maximum error of the model and be careful on trading in dates where the expected variation of the cryptocurrency does not correlates with the volatility that the market had, so we ensure we could have possitive outcomes even if the model sometimes commit errors. We also need to take in consideration the fees which the trading broker takes form each buying and selling operation, so, it is not enough to have a model which performs well at predicting the cryptocurrency price but also designing a good strategy according to market conditions and adjusted to your model metrics.

As for the next steps, which are out of the scope of this project, aside form designing a cryptocurrency trading strategy and testing it using a demo account on any online platform, demo account allows you to invest with fake money, I think that the LSTM approach should be further studied and tested with other kind of asset where a longer history of data samples can be downloaded and it is a less volatile market, like the stock market or real estate. As we have seen in *figure* 6.1 deep learning techniques achieve better performance when we have a bigger dataset, thus it will be interesting to compare those models again but using a different type of asset. However, the stock market prices

are affected by many more factors than criptocurrencies, such like companies investement, their clients portfolio, political decisions and many more factors that should be correctly represented in the dataset.

Bibliography

Ammous, S. (2018). Can cryptocurrencies fulfil the functions of money? *The Quarterly Review of Economics and Finance*, 70:38–51.

Anghel, A., Papandreou, N., Parnell, T., De Palma, A., and Pozidis, H. (2018). Benchmarking and optimization of gradient boosting decision tree algorithms.

Berentsen, A. and Schär, F. (2018). A short introduction to the world of cryptocurrencies. *Review*, 100:1–16.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

Derbentsev, V., Babenko, V., Khrustalev, K., Obruch, H., and Khrustalova, S. (2021). Comparative performance of machine learning ensemble algorithms for forecasting cryptocurrency prices. *International Journal of Engineering*, 34(1):140–148.

Fang, L., Ivashina, V., and Lerner, J. (2015). The disintermediation of financial markets: Direct investing in private equity. *Journal of Financial Economics*, 116(1):160–178.

Härdle, W. K., Harvey, C. R., and Reule, R. C. G. (2020). Understanding Cryptocurrencies*. *Journal of Financial Econometrics*, 18(2):181–208.

Islam, S., Sholahuddin, A., and Abdullah, A. (2021). Extreme gradient boosting (xgboost) method in making forecasting application and analysis of usd exchange rates against rupiah. *Journal of Physics: Conference Series*, 1722:012016.

Liew, J., Li, R., Budavari, T., and Sharma, A. (2019). Cryptocurrency investing examined. *The Journal of the British Blockchain Association*, 2:1–12.

Luo, J., Zhang, Z., Fu, Y., and Rao, F. (2021). Time series prediction of covid-19 transmission in america using lstm and xgboost algorithms. *Results in Physics*, 27:104462.

Mukhopadhyay, U., Skjellum, A., Hambolu, O., Oakley, J., Yu, L., and Brooks, R. (2016). A brief survey of cryptocurrency systems. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 745–752.

Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., S. Band, S., Reuter, U., Gama, J., and Gandomi, A. (2020). Data science in economics: Comprehensive review of advanced machine learning and deep learning methods. *Mathematics*, 8:1799.

Ortu, M., Uras, N., Conversano, C., Bartolucci, S., and Destefanis, G. (2022). On technical trading and social media indicators for cryptocurrency price classification through deep learning. *Expert Systems with Applications*, 198:116804.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Vrskova, R., Sykora, P., Kamencay, P., Hudec, R., and Radil, R. (2021). Hyperparameter tuning of convlstm network models. In *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 15–18.

Yadav, A., Jha, C. K., and Sharan, A. (2020). Optimizing lstm for time series prediction in indian stock market. *Procedia Computer Science*, 167:2091–2100. International Conference on Computational Intelligence and Data Science.

Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2018). Blockchain technology overview.

Zanin, M., Alessandretti, L., ElBahrawy, A., Aiello, L. M., and Baronchelli, A. (2018). Anticipating cryptocurrency prices using machine learning. *Complexity*, 2018:8983590.

Appendix A

Investigation Article

Comparison between Gradiend Boosting Enssembler and Long Short Term Memory Neural Networks for Cryptocurrency Investing

Buftea, Alberto Gabriel

Universidad Internacional de la Rioja, Logroño (España)

19 September 2022

ABSTRACT

In this work we will review the artificial intelligence techniques that achieved best results in state of the art publications about cryptocurrency price prediction. The objective of this article is to serve as a guide for the development of a cryptocurrency price prediction algorithm. We will investigate the state of the art publications and choose the two techniques that achieved bets result in overall, one for deep learning and another one for machine learning. We will replicate those models in our working environment to make predictions for the ethereum cryptocurrency price to finally compare the results. We will see that in the specific experiment of this article, the machine learning approach Gradient Boosting Enssemble achieved better results than the deep learning approach long short term memory.

KEY WORDS

Artificial Intelligence, Cryptocurrency, Investing

I. Introduction

In this project we will compare the Gradient Boosting Ensemble machine learning technique with the Long Short Term Memory deep learning technique for cryptocurrency investing. To form the dataset, we request the *OHLCV* open, high, low, close prices and volume form the *CoinApi.io* online API and we request cryptocurrency social media sentiment data using the *Augmento* online API. To include more features to our dataset, we will calculate technical analysis indicators using the *Technical Analysis (TA) library*. The models are going to be created using python, *SkLearn* library for machine learning and *Tesnorflow* for deep learning. We will see that in our experiments the Gradient Boosting Ensemble outperforms the Lonsg Short Term memory network mainly because we have requested ethereum price information starting form December 2020 to august 2022, a decision which is based on the cryptocurrency market change of behavior starting from December 2020.

II. State of art

In [1], the authors compare the prediction of over 100 cryptocurrencies using 11 different algorithms. To perform this analysis they first studied the correlation between the most traded cryptocurrencies and the correlation with traditional assets such as the SP 500 index. Not surprisingly, in this analysis they found that cryptocurrencies with larger market

capitalization have a bigger correlation than those with lower capitalization. An interesting finding which they have discovered is that cryptocurencies have almost no correlation with traditional financial assets, being the correlation less than 0.1 points.

The investigation realized by [2] used the *Prisma* approach to find the 57 most relevant articles published in the area of data science applied to the economy. The criteria which they used to structure the results was splitting by the research area and and the methodology employed. They state that for the research area of investing, 34 articles were focused in the stock market while 3 articles were focusing on cryptocurrencies, finding out that that Long Short Term Memory models outperforms other data science techniques.

From the work performed by citeZanin2018, they compared two machine learning techniques based on Gradient Boosting (GB) decision trees with the deep learning Long short Term Memory (LSTM) neural network for cryptocurrency investing. The study concluded that GB techniques works better when predictions are based on short term windows while LSTM when prediction are more accurate when involves several weeks windows. An important finding from the study is that all of the models performed better than the simple moving average strategy, which is an indicator that gives purpose to the research that we are doing in this article.

In the work performed by [3], they compared what are said to be the two most powerful machine learning approaches that are bagging random forest and gradient boosting ensemble. They predicted the price of two cryptocurrencies for a 95 days periods, obtaining surprisingly accurate results as can be seen in *figure* 1 where the blue line represents the real price of ethereum, the dotted red line represents the BRF approach while the dotted gray line represents the GBE approach.. From their findings, the gradient boosting ensemble achieved the best performance, being 10% more accurate than the bagging random forest.

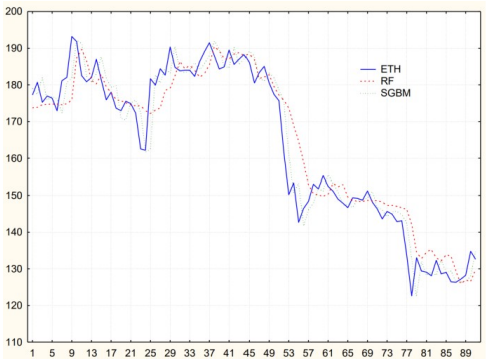


Figure 1: Ehtereum price prediction base on BRF and GBE models

III. Objectives and methodology

The conclusion of the state of the art investigation is that in all of the published articles it is stated that the best performance is achieved using the LSTM networks for deep learning and Gradient Boosting for machine learning, therefore those two models are the ones we are going to compare for our experiment.

Fist we are going to download and prepare the dataset for training the models, getting the cryptocurrency price information and twitter sentiment analysis on a daily basis. We see the correlation between the ethereum price and the proportioned sentiments, filtering for those sentiments that are 75% or more correlated to any of the *OHLCV* indicators. Afterwards, we add the most relevant technical indicators inspiring the choice form the work done by [4], *Relative Strength Index (RSI)*, *Bollinger Bands*, *Moving Average Convergence Divergence (MACD)*, *Vortex indicator (VI)* and *Ease of Movement* to the dataset and we split it in such a way that we left the last four weeks for testing purposes, this is 28 days.

We create the models in python using SkLearn and Tensorflow libraries. We transform the dataset in such a way that with

the GB approach we will use the previous day information to predict the next day close price while for the LTSM approach we will use a 7 steps model where the information of the previous 7 days is used at predicting the 8th day close price. The most relevant model hyperparameters are tunned using iterations and we test both models performance several times till we get the best output.

Our objective is to compare both approaches at predicting the price of a very volatile assets such like the cryptocurrency, and determine if it is feasible or not to develop a trading strategy in this market based on the price predictions performed by an artificial intelligence. The metrics which we are going to use for comparison are the maximum achieved error, the mean absolute error, the median absolute error and the R2 value.

IV. Contribution

When investigating about algorithmic cryptocurrency investing I detected several deficiencies that I want to address with my work.

- No cryptocurrency investing paper properly explains the working fundamentals of cryptocurrencies.
- Investing publications usually skips the working principles of the models they are using, just briefly stating what kind of techniques the model uses.
- Publications does not explain how to create those models, how to prepare the data and how does the model hyperparametes changes affects the output.
- Most of the publications uses as features only the basic market information *OHLCV* open, high, low, close prices and volume.

My intention with this project is to offer a complete guide for someone interested into algorithmic investing where all the relevant information from designing the investment strategy to setting up a state of the art model can be found through this paper. Here I am combining social media sentiments from Augemto API, togeder with the *OHLCV* price data requested form the CoinAPI cryptocurrency data API and additional technical analysis indicators which are calculated using the python library Technicall Analysis (TA). of course not every explanation can be covered in this article, thus I suggest you switch to the main dissertation project paper if you are interested in how the models have been created and how does cryptocurrency works.

V. Results

In this section we will cover each model output based on the customization of the hyperparameters to afterwards compare both models and draw conclusion in the next sections. The explanation of how I created the models and the improvement in the predicted output based on tunning the hyperparameters will be skipped, i will directly present what is the best configuration for each of the models stating briefly how was it achieved to later directly present the predicted graphs.

A Gradient Boosting Ensemble prediction

Table 1 covers the hyperparameters values that we set for the gradient boosting ensemble model so that it achieved the best prediction. We reached those values in several iterations, first setting the learning rate and number of estimators where performance was best, to afterward iterate again filtering for the max depth of the trees to finally iterate again searching the optimum values for the minimum samples split and minimum samples leaf.

Using the parameters above, after training the model the prediction we have for the last 4 weeks of our dataset, form the 11^{th} of July till the 7^{th} of August is shown in figure 2. We can

Parameter	Value
Learning rate	0.05
N. of Estimators	1000
Random State	1400
Max Deepth	3
Min Samples Split	2
Min Samples Leaf	1

Table 1: GBR model parameters

see that the predicted curve closely matches the real price curve, in fact, the prediction of the model has a mean average value of just 62 USDs, pretty god result if you take in consideration that the price of the ethereum in the predicted period variated more than 800 dollars.

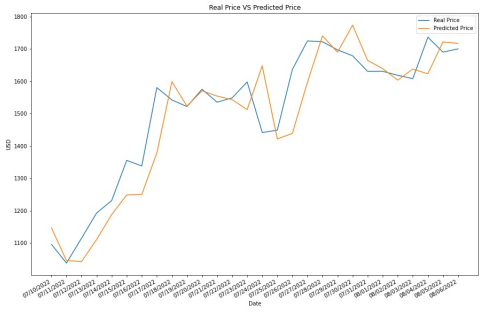


Figure 2: Gradient Boosting Ensemble Prediction

B Long Short Term Memory prediction

Figure 3 is a representation of the final LTSM model that we used at predicting the price of ethereum, the modified hyperparameters of this model are covered in the table 2. To achieve this configuration I inspired my decisions in the information extracted form [5] and [6], were we decided to let just one hidden layer for the LTSM network and adopt an early stopping callback to limit the number of epochs and avoid overfitting when training the model. Afterwards we iterated several times to set the optimum number of neurons in the LTSM layer and the dense layer of the model, to finally slightly increase the performance modifying the activation function of the dense hidden layer to ReLu since predicted prices will never take negative values.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_145 (LSTM)              (None, 350)               522200
_________________________________________________________________
dropout_21 (Dropout)         (None, 350)               0
_________________________________________________________________
dense_288 (Dense)            (None, 40)                14040
_________________________________________________________________
dense_289 (Dense)            (None, 1)                 41
=================================================================
Total params: 536,281
Trainable params: 536,281
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: LSTM final model summary

Parameter	Value
LTSM Units	350
Dropout	0.2
Dense Layer Units	40
Optimizer	Adam
Loss	Mean Sq. Error
Early stopping monitor	Loss Function
Early stopping patience	5
Batch size	2

Table 2: LTSM model parameters

The predicted output achieved with this configuration can be seen in figure 4, where we see that the model predicted price curves follows the trend of the real price curve but it does not follow the spikes and variations as closely as the GBR model. This model achieved a mean average error of 79 USDs, still acceptable when considering that the ethereum price variation in the predicted period was of more than 800 USD.

VI. Results Comparison

The first noticeable aspect it the time difference needed to train one model versus the other. As table 3 indicates, it took only 5 seconds and a half to the Grandient Boosting Machine to fit while it took 100 seconds to the LTSM network.

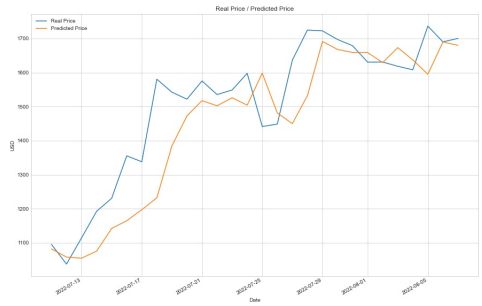


Figure 4: LSTM final prediction

Model	Time for training (seconds)
LTSM	100
GBR	5.5

Table 3: Comparing models training time

Looking at *figure* 5 where the final prediction of the GBR in green is compared with the final prediction of the LTSM model in orange and the real price in blue, we can see that the green curve follows more closely the blue curve, meaning than from a visual perspective the machine learning technique achieved more accurate results than the deep learning technique even with less computational power.

Figure 5: LSTM final prediction

Comparing some metrics to quantify the predictions of the models in *table* 4, we see that the GBR model had lower errors than the LTSM model for each one of the considered errors and the R2 value which measures the variance of the predicted price form the real price, a score of 100% meaning that they are perfectly correlated, is at 80% for the GBR model while only 68% for the LTSM model.

Metric	LTSM	GBR
Max Error	344 USD	206 USD
Mean Abs. Error	79 USD	63 USD
Median Abs. Error	52 USD	39 USD
R2 Score	0.68	0.8

Table 4: Comparing predicted vs real prices USD

VII. Conclusion

The gradient boosting ensemble achieved better results than the long short term memory network. In fact, the machine learning technique overcome the deep learning approach for all the comparison metrics that we have used, it even did it using less computational power thus there is no reason in favor of the LTSM that could make us select that model instead of the GBR.

The results were somehow expected, mainly because I have filtered cryptocurrency data since December 2020 till August 2022 using a daily time frame for the OHLCV data which resulted in a dataset of just 588 samples to train an test the model, not enough for a deep learning approach to extract all the needed features. As can be seen in *figure* 6, machine learning techniques achieves a better performance quicker than the deep learning techniques when it comes to small amount of data while the deep learning approach performance increases overcoming machine learning when big datasets are involved.

However, as I stated in the objective section, the cryptocurency market had a instantaneous change of behavior starting from December 2020 so I think the decision of filtering data starting from that time period was correct, specially because the results achieved with the Gradient Boosting Ensemble are quite

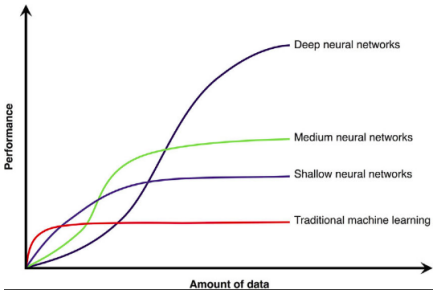


Figure 6: ML vs DL performance based on the amount of data

positive, achieving really close predictions to the real price thus I strongly believe a trading strategy that buys and sells several times per week could be designed and it would gave us profits. being the cryptocurreency market recent, most cryptocurrencies are recently published thus we would not have a long historic datset information for this kinf of assets.

References

[1] Jim Liew, Richard Li, Tamas Budavari, and Avinash Sharma. Cryptocurrency investing examined. *The Journal of the British Blockchain Association*, 2:1–12, 11 2019.

[2] Saeed Nosratabadi, Amir Mosavi, Puhong Duan, Pedram Ghamisi, Ferdinánd Filip, Shahab S. Band, Uwe Reuter, João Gama, and Amir Gandomi. Data science in economics: Comprehensive review of advanced machine learning and deep learning methods. *Mathematics*, 8:1799, 10 2020.

[3] V. Derbentsev, V. Babenko, K. Khrustalev, H. Obruch, and S. Khrustalova. Comparative performance of machine learning ensemble algorithms for forecasting cryptocurrency prices. *International Journal of Engineering*, 34(1):140–148, 2021.

[4] Marco Ortu, Nicola Uras, Claudio Conversano, Silvia Bartolucci, and Giuseppe Destefanis. On technical trading and social media indicators for cryptocurrency price classification through deep learning. *Expert Systems with Applications*, 198:116804, 2022.

[5] Anita Yadav, C K Jha, and Aditi Sharan. Optimizing lstm for time series prediction in indian stock market. *Procedia Computer Science*, 167:2091–2100, 2020. International Conference on Computational Intelligence and Data Science.

[6] Roberta Vrskova, Peter Sykora, Patrik Kamencay, Robert Hudec, and Roman Radil. Hyperparameter tuning of convlstm network models. In *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 15–18, 2021.

Appendix B

Dataset Generating Code

In [72]:

```
import requests as req
import pandas as pd
import numpy as np
import json
import tensorflow as tf
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import seaborn as sb
import ta
```

In [20]:

```
url = "http://rest.coinapi.io/exchangerate/USD?"
headers = {'X-CoinAPI-Key' : '00C44B61-72B2-4A78-A3C9-A22DDBD7BB81'}
response = req.get(url, headers=headers)
```

In [33]:

```
url = "https://rest.coinapi.io/v1/symbols?apikey=00C44B61-72B2-4A78-A3C9-A22DDBD7BB81"
```

In [35]:

```
response = req.get(url).json()
```

In [37]:

```
with open('symbols.json', 'w') as json_file:
    json.dump(response, json_file)
```

In [38]:

```
print(response)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [2]:

```
url = "http://rest.coinapi.io/v1/ohlcv/BINANCE_SPOT_ETH_USDC/history?period_id=1DAY&time_st
headers = {'X-CoinAPI-Key' : '00C44B61-72B2-4A78-A3C9-A22DDBD7BB81'}
```

In [3]:

```
response = req.get(url, headers=headers).json()
```

In [4]:

```
len(response)
```

Out[4]:

```
588
```

In [5]:

```
response
```

Out[5]:

```
[{'time_period_start': '2020-12-28T00:00:00.0000000Z',
  'time_period_end': '2020-12-29T00:00:00.0000000Z',
  'time_open': '2020-12-28T00:00:03.0950000Z',
  'time_close': '2020-12-28T23:59:38.6390000Z',
  'price_open': 684.42,
  'price_high': 746.98,
  'price_low': 680.83,
  'price_close': 729.01,
  'volume_traded': 25763.51754,
  'trades_count': 15333},
 {'time_period_start': '2020-12-29T00:00:00.0000000Z',
  'time_period_end': '2020-12-30T00:00:00.0000000Z',
  'time_open': '2020-12-29T00:00:02.5950000Z',
  'time_close': '2020-12-29T23:59:42.5050000Z',
  'price_open': 729.02,
  'price_high': 739.09,
  'price_low': 687.64,
```

In [6]:

```
with open('ETH_USD.json', 'w') as json_file:
    json.dump(response, json_file)
```

In [7]:

```
df = pd.read_json('ETH_USD.json')
```

In [8]:

```
df
```

Out[8]:

	time_period_start	time_period_end	time_open	time_close	pı
0	2020-12-28T00:00:00.0000000Z	2020-12-29T00:00:00.0000000Z	2020-12-28T00:00:03.0950000Z	2020-12-28T23:59:38.6390000Z	
1	2020-12-29T00:00:00.0000000Z	2020-12-30T00:00:00.0000000Z	2020-12-29T00:00:02.5950000Z	2020-12-29T23:59:42.5050000Z	
2	2020-12-30T00:00:00.0000000Z	2020-12-31T00:00:00.0000000Z	2020-12-30T00:00:01.4280000Z	2020-12-30T23:59:11.5500000Z	
3	2020-12-31T00:00:00.0000000Z	2021-01-01T00:00:00.0000000Z	2020-12-31T00:00:01.7820000Z	2020-12-31T23:58:51.7880000Z	
4	2021-01-01T00:00:00.0000000Z	2021-01-02T00:00:00.0000000Z	2021-01-01T00:00:26.1160000Z	2021-01-01T23:56:57.2780000Z	
...	...	...	...	...	
583	2022-08-03T00:00:00.0000000Z	2022-08-04T00:00:00.0000000Z	2022-08-03T00:00:05.8050000Z	2022-08-03T23:59:59.7240000Z	
584	2022-08-04T00:00:00.0000000Z	2022-08-05T00:00:00.0000000Z	2022-08-04T00:00:01.4310000Z	2022-08-04T23:59:58.9150000Z	
585	2022-08-05T00:00:00.0000000Z	2022-08-06T00:00:00.0000000Z	2022-08-05T00:00:06.6330000Z	2022-08-05T23:59:59.5180000Z	
586	2022-08-06T00:00:00.0000000Z	2022-08-07T00:00:00.0000000Z	2022-08-06T00:00:00.5590000Z	2022-08-06T23:59:55.0270000Z	
587	2022-08-07T00:00:00.0000000Z	2022-08-08T00:00:00.0000000Z	2022-08-07T00:00:03.4570000Z	2022-08-07T23:59:58.7660000Z	

588 rows × 10 columns

In [57]:

```
r = req.request("GET", "http://api-dev.augmento.ai/v0.1/coins")
print(r.content)
```

```
b'["0x","aelf","aion","ardor","ark","augur","bitcoin","cardano","dash","dece
ntraland","decred","digibyte","dogecoin","dragonchain","electroneum","eo
s","ethereum","iota","komodo","lisk","litecoin","monero","nano","nem","ne
o","ontology","polymath","qtum","ravencoin","reddcoin","ripple","siacoin","s
teem","stellar","stratis","syscoin","tether","tron","vechain","verge","walto
nchain","wanchain","zcash","zilliqa","binance_coin","bitcoin_cash","bitcoin_
gold","bitcoin_sv","kucoin_shares","pundi_x"]\n'
```

In [58]:

```
r = req.request("GET", "http://api-dev.augmento.ai/v0.1/topics")
print(r.content)
```

```
b'{"0":"Hacks","1":"Pessimistic/Doubtful","2":"Banks","3":"Selling","4":"Mar
ket_manipulation","5":"(De-)centralisation","6":"Angry","7":"ETF","8":"Lever
age","9":"Bottom","10":"Institutional_money","11":"FOMO","12":"Predictio
n","13":"Adoption","14":"Fearful/Concerned","15":"Portfolio","16":"FUD_them
e","17":"Whitepaper","18":"Announcements","19":"Technical_analysis","20":"Fl
ippening","21":"Community","22":"Investing/Trading","23":"Euphoric/Excite
d","24":"Hodling","25":"ICO","26":"Bearish","27":"Going_short","28":"Uncerta
in","29":"Volume","30":"Risk","31":"Governance","32":"Ban","33":"Cheap","3
4":"Short_term_trading","35":"Fork","36":"Progress","37":"Shilling","38":"Bu
llish","39":"Happy","40":"Bubble","41":"Bots","42":"Hopeful","43":"Bug","4
4":"Open_source","45":"Token_economics","46":"Security","47":"Marketing","4
8":"Bad_news","49":"Due_diligence","50":"Team","51":"Partnerships","52":"Pum
p_and_dump","53":"Sad","54":"Panicking","55":"Listing","56":"Regulation/Poli
tics","57":"Dip","58":"Launch","59":"FOMO_theme","60":"Advice/Support","6
1":"Rebranding","62":"Wallet","63":"Good_news","64":"Problems_and_issues","6
5":"Mining","66":"Waiting","67":"Learning","68":"Scaling","69":"Fees","7
0":"Roadmap","71":"Recovery","72":"Technology","73":"Mistrustful","74":"Mark
etcap","75":"Positive","76":"Tax","77":"Long_term_investing","78":"Strateg
y","79":"Competition","80":"Whales","81":"Correction","82":"Stablecoin","8
3":"Buying","84":"Warning","85":"Annoyed/Frustrated","86":"Price","87":"Use_
case/Applications","88":"Rumor","89":"Scam/Fraud","90":"Airdrop","91":"Optim
istic","92":"Negative"}\n'
```

In [59]:

```
r.json()
```

Out[59]:

```
{'0': 'Hacks',
 '1': 'Pessimistic/Doubtful',
 '2': 'Banks',
 '3': 'Selling',
 '4': 'Market_manipulation',
 '5': '(De-)centralisation',
 '6': 'Angry',
 '7': 'ETF',
 '8': 'Leverage',
 '9': 'Bottom',
 '10': 'Institutional_money',
 '11': 'FOMO',
 '12': 'Prediction',
 '13': 'Adoption',
 '14': 'Fearful/Concerned',
 '15': 'Portfolio',
 '16': 'FUD_theme',
```

In [11]:

```
SentimentUrl = "http://api-dev.augmento.ai/v0.1/events/aggregated"
SentimentParams = {
  "source" : "twitter",
  "coin" : "ethereum",
  "bin_size" : "24H",
  "count_ptr" : 600,
  "start_ptr" : 0,
  "start_datetime" : "2020-12-27T00:00:00Z",
  "end_datetime" : "2022-08-08T00:00:00Z",
}
SentimentResponse = req.get(SentimentUrl, params=SentimentParams).json()
```

In [12]:

```
len(SentimentResponse)
```

Out[12]:

```
589
```

In [10]:

```
SentimentResponse
```

Out[10]:

```
[{'counts': [0,
   3,
   1,
   12,
   2,
   6,
   1,
   0,
   2,
   3,
   0,
   5,
   47,
   25,
   1,
   9,
   0,
```

In [13]:

```
len(SentimentResponse[0]['counts'])
```

Out[13]:

```
93
```

In [14]:

```
remainingList=[]
remainingDates=[]
remainingIndices = [1,3, 6, 14, 23, 24, 26, 27, 30, 33, 36, 38, 39, 40, 42, 48, 53, 54, 63,
for item in SentimentResponse:
    itemTopics = item['counts']
    itemDate = item['datetime']
    remainingTopics = [itemTopics[i] for i in remainingIndices]
    remainingList.append(remainingTopics)
    remainingDates.append(itemDate)
```

In [15]:

```
dfSentiments = pd.DataFrame(remainingList, columns = ['Pessimistic/Doubtful','Selling','Ang
```

In [16]:

```
dfSentiments
```

Out[16]:

	Pessimistic/Doubtful	Selling	Angry	Fearful/Concerned	Euphoric/Excited	Hodling	Bearish
0	12	20	5	3	37	4	9
1	5	35	4	2	27	3	15
2	9	26	7	8	13	1	5
3	3	16	3	0	19	2	5
4	2	11	4	1	20	0	7
...	...	...	...	...	...	...	..
584	3	10	1	5	14	1	26
585	0	10	0	1	11	1	45
586	0	12	2	3	18	2	22
587	5	8	3	4	8	1	9
588	4	8	0	3	9	2	19

589 rows × 26 columns

In [17]:

```
dfSentiments.insert(loc=0, column= "time_period", value=remainingDates)
```

In [18]:

```
dfSentiments.head()
```

Out[18]:

	time_period	Pessimistic/Doubtful	Selling	Angry	Fearful/Concerned	Euphoric/Excited	Hodl
0	2020-12-27T00:00:00Z	12	20	5	3	37	
1	2020-12-28T00:00:00Z	5	35	4	2	27	
2	2020-12-29T00:00:00Z	9	26	7	8	13	
3	2020-12-30T00:00:00Z	3	16	3	0	19	
4	2020-12-31T00:00:00Z	2	11	4	1	20	

5 rows × 27 columns

In [19]:

```
dfSentiments.to_csv('ETH_Sentiment.csv')
```

In [90]:

```
dfETH = pd.read_json('ETH_USD.json')
dfSentiment = pd.read_csv('ETH_Sentiment.csv')
```

In [91]:

```
dfETH["time_period_start"] = pd.to_datetime(dfETH["time_period_start"])
dfETH["time_period_start"] = dfETH.time_period_start.dt.strftime('%m/%d/%Y')
```

In [92]:

```
dfETH.drop(columns=["time_period_end", "time_open", "time_close"], inplace=True)
dfETH.rename(columns={"time_period_start" : "time_period"}, inplace=True)
```

In [93]:

```
dfSentiment["time_period"] = pd.to_datetime(dfSentiment["time_period"])
dfSentiment["time_period"] = dfSentiment.time_period.dt.strftime('%m/%d/%Y')
dfSentiment.drop(columns=["Unnamed: 0"], inplace=True)
```

In [118]:

```
dfData = pd.merge(dfETH, dfSentiment, on="time_period", how="inner")
```

In [95]:

```
dfData.set_index('time_period', inplace = True)
print(len(dfData))
dfData.head()
```

```
588
```

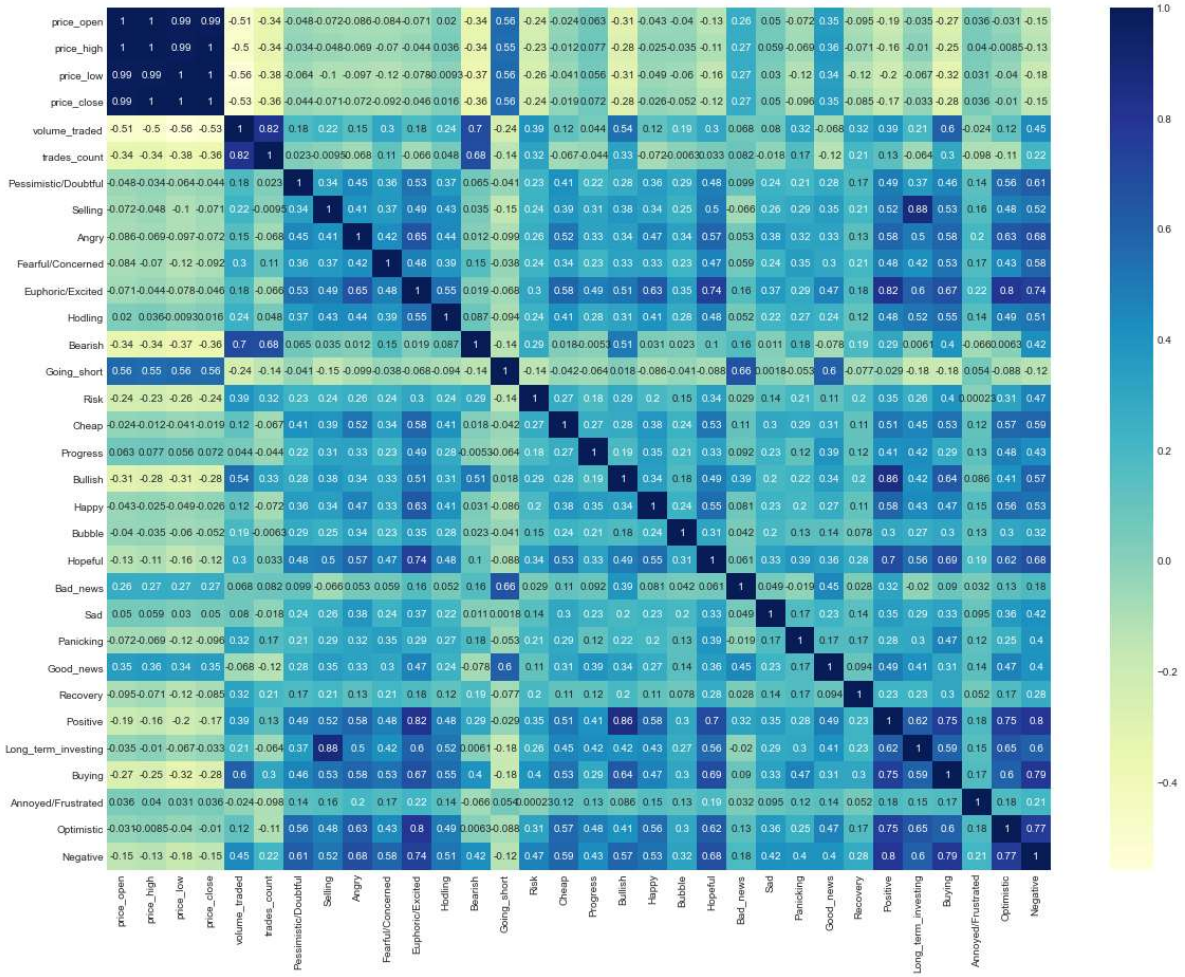
Out[95]:

time_period	price_open	price_high	price_low	price_close	volume_traded	trades_count	Pess
12/28/2020	684.42	746.98	680.83	729.01	25763.51754	15333	
12/29/2020	729.02	739.09	687.64	730.91	16248.29869	11362	
12/30/2020	731.79	758.60	709.92	751.98	14385.47579	9679	
12/31/2020	752.50	756.04	722.00	737.27	18938.33743	9639	
01/01/2021	736.90	750.39	714.86	730.79	15129.68596	7544	

5 rows × 32 columns

In [121]:

```
plt.subplots(figsize=(20,15))
dataplot = sb.heatmap(dfData.corr(), cmap="YlGnBu", annot=True)
plt.savefig('correlationMatix.jpg')
plt.show()
```



In [122]:

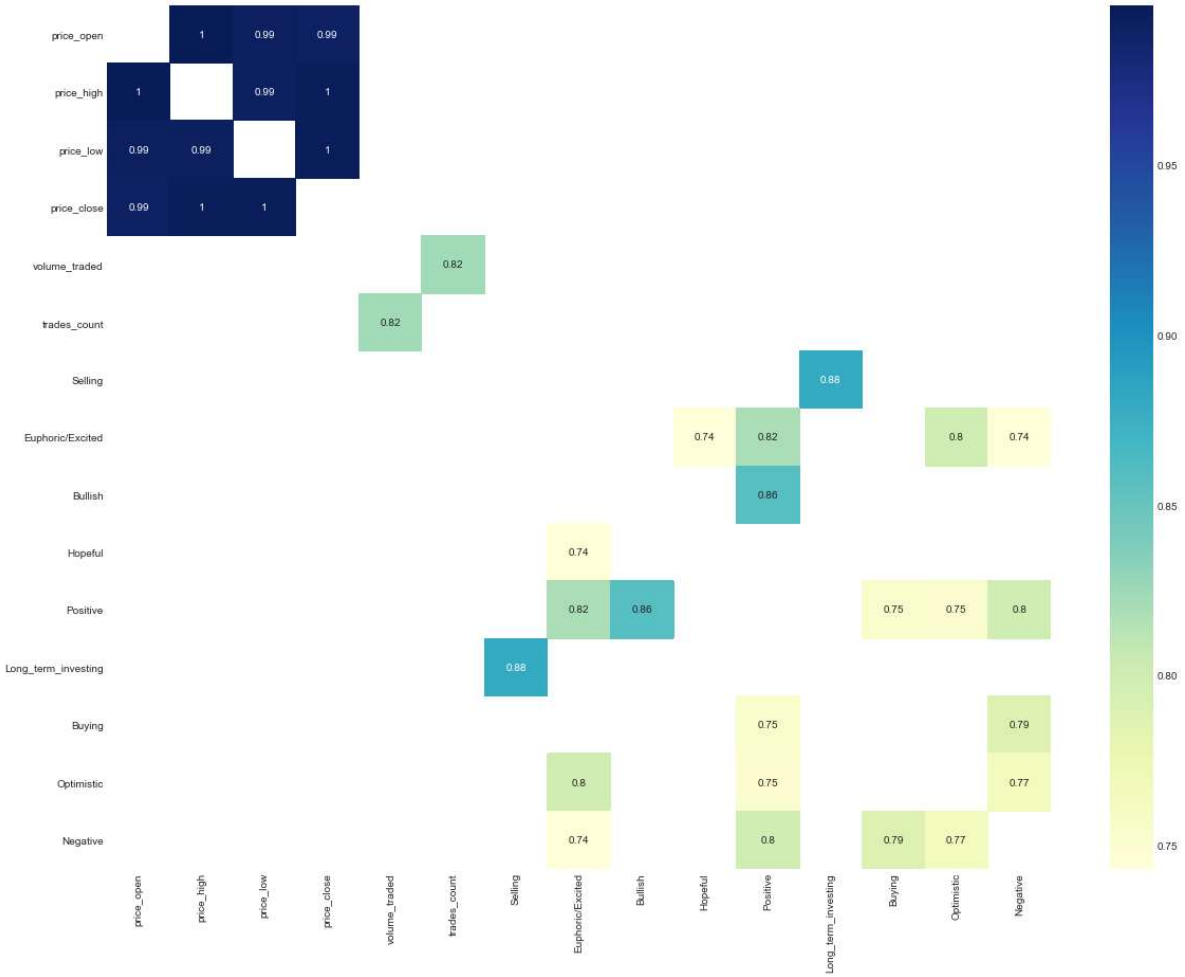
```
corelation = dfData.corr()
corelation = corelation[corelation.abs()>0.7]
corelation[corelation == 1] = np.nan
```

In [123]:

```
corelation.dropna(axis = 0, how='all', inplace = True)
corelation.dropna(axis = 1, how='all', inplace = True)
```

In [124]:

```
plt.subplots(figsize=(20,15))
dataplot = sb.heatmap(corelation, cmap="YlGnBu", annot=True)
plt.show()
```



In [100]:

```
rows = list(corelation.index)
cols = list(corelation.columns)
```

In [101]:

```
columns = list(set(rows+cols))
```

In [102]:

```
dfData = dfData[columns]
```

In [103]:

```
dfData.head()
```

Out[103]:

time_period	volume_traded	Bullish	Positive	Negative	Hopeful	Euphoric/Excited	Long_term_
12/28/2020	25763.51754	89	166	122	12	27	
12/29/2020	16248.29869	56	144	133	10	13	
12/30/2020	14385.47579	66	101	53	7	19	
12/31/2020	18938.33743	48	112	89	12	20	
01/01/2021	15129.68596	24	83	74	3	16	

In [105]:

```
rsi = ta.momentum.RSIIndicator(dfData["price_close"], window = 7, fillna = False)
dfData["rsi"] = rsi.rsi()
```

In [106]:

```
bolinger = ta.volatility.BollingerBands(dfData["price_close"], window = 7, window_dev = 1,
dfData["bolinger_high"] = bolinger.bollinger_hband()
dfData["bolinger_low"] = bolinger.bollinger_lband()
dfData["bolinger_middle"] = bolinger.bollinger_mavg()
```

In [108]:

```
macd = ta.trend.MACD(dfData["price_close"], window_slow = 14, window_fast = 4, window_sign
dfData["macd"] = macd.macd()
```

In [109]:

```
dfData["vortex"] = ta.trend.vortex_indicator_neg(high=dfData["price_high"], low=dfData["pri
```

In [110]:

```
stoch = ta.momentum.StochasticOscillator(high=dfData["price_high"], low=dfData["price_low"]
dfData["stoch"] = stoch.stoch()
```

In [112]:

```
dfData["movement"] = ta.volume.ease_of_movement(high=dfData["price_high"], low=dfData["pric
```

In [113]:

```
dfData
```

Out[113]:

time_period	volume_traded	Bullish	Positive	Negative	Hopeful	Euphoric/Excited	Long_term_
12/28/2020	25763.51754	89	166	122	12	27	
12/29/2020	16248.29869	56	144	133	10	13	
12/30/2020	14385.47579	66	101	53	7	19	
12/31/2020	18938.33743	48	112	89	12	20	
01/01/2021	15129.68596	24	83	74	3	16	
...	...	...	...	...	...	...	
08/03/2022	62869.24340	92	133	103	2	14	
08/04/2022	52963.42630	94	115	84	5	11	
08/05/2022	58457.61080	110	129	85	7	18	
08/06/2022	32432.65890	77	103	82	4	8	
08/07/2022	31401.66210	68	102	79	7	9	

588 rows × 23 columns

In [114]:

```
dfData.to_csv("dfData.csv")
```

In [116]:

```
dfData.to_excel("dfData.xlsx")
```

In [ ]:

```

```

Appendix C

Gradient Boosting Regression Model Code

GradientBoosting

September 21, 2022

[198]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import sklearn
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
import datetime as dt
import timeit
```

[55]:
```
dfData = pd.read_csv("dfData.csv")
dfData.set_index('time_period', inplace = True)
dfData.fillna(value = dfData.mean(), inplace=True)
print(len(dfData))
dfData.head(20)
```

```
588
```

[55]:

time_period	volume_traded	Bullish	Positive	Negative	Hopeful
12/28/2020	25763.51754	89	166	122	12
12/29/2020	16248.29869	56	144	133	10
12/30/2020	14385.47579	66	101	53	7
12/31/2020	18938.33743	48	112	89	12
01/01/2021	15129.68596	24	83	74	3
01/02/2021	26362.64832	75	129	106	10
01/03/2021	86710.73835	169	325	191	27
01/04/2021	88682.01485	120	261	212	9
01/05/2021	48366.89405	105	205	177	11
01/06/2021	51948.05131	106	185	110	14
01/07/2021	50785.01095	86	148	110	16
01/08/2021	55425.91834	77	125	72	4
01/09/2021	29911.63100	77	160	106	15
01/10/2021	57780.80793	86	166	135	17
01/11/2021	140653.08015	84	145	156	16
01/12/2021	57895.78435	69	147	108	15

01/13/2021	37266.08719	60	120	73	14
01/14/2021	48922.97282	71	132	74	12
01/15/2021	58864.46004	73	136	102	17
01/16/2021	45301.65087	88	147	83	12

time_period	Euphoric/Excited	Long_term_investing	price_open	trades_count	\
12/28/2020	27	79	684.42	15333	
12/29/2020	13	53	729.02	11362	
12/30/2020	19	36	731.79	9679	
12/31/2020	20	39	752.50	9639	
01/01/2021	16	28	736.90	7544	
01/02/2021	24	45	731.19	13326	
01/03/2021	88	104	775.24	38600	
01/04/2021	59	100	980.02	50719	
01/05/2021	46	58	1043.99	32159	
01/06/2021	48	86	1103.50	33189	
01/07/2021	28	67	1212.62	29682	
01/08/2021	26	71	1226.49	32663	
01/09/2021	36	51	1217.70	21356	
01/10/2021	27	65	1281.01	35595	
01/11/2021	19	90	1255.39	65237	
01/12/2021	36	47	1087.18	27440	
01/13/2021	15	44	1049.31	20490	
01/14/2021	29	63	1130.00	28730	
01/15/2021	22	68	1231.38	33230	
01/16/2021	22	58	1169.68	29581	

time_period	Selling	…	Optimistic	price_high	rsi	bolinger_high	\
12/28/2020	35	…	44	746.98	52.328735	2748.975368	
12/29/2020	26	…	35	739.09	52.328735	2748.975368	
12/30/2020	16	…	26	758.60	52.328735	2748.975368	
12/31/2020	11	…	35	756.04	52.328735	2748.975368	
01/01/2021	12	…	28	750.39	52.328735	2748.975368	
01/02/2021	19	…	34	788.89	52.328735	2748.975368	
01/03/2021	58	…	42	1021.85	94.762168	860.170556	
01/04/2021	42	…	55	1168.00	95.899571	942.758774	
01/05/2021	19	…	58	1135.42	96.695249	1023.060747	
01/06/2021	32	…	30	1216.65	97.641192	1118.093176	
01/07/2021	35	…	32	1294.91	97.752074	1190.641633	
01/08/2021	28	…	29	1275.00	94.799494	1230.857046	
01/09/2021	18	…	25	1308.26	95.834777	1253.615701	
01/10/2021	26	…	31	1351.50	88.123762	1270.106070	
01/11/2021	38	…	27	1260.12	52.858556	1265.392006	
01/12/2021	19	…	34	1150.70	47.882951	1269.804796	
01/13/2021	15	…	22	1136.92	57.495284	1260.071940	

```
01/14/2021        20  …          29     1246.41  66.915858    1260.930964
01/15/2021        19  …          30     1255.00  58.000137    1252.475597
01/16/2021        19  …          22     1292.45  63.484354    1236.506429

             bolinger_low  bolinger_middle        macd    vortex      stoch  \
time_period
12/28/2020    2506.840499      2627.907933    5.714479  0.974179  56.135943
12/29/2020    2506.840499      2627.907933    5.714479  0.974179  56.135943
12/30/2020    2506.840499      2627.907933    5.714479  0.974179  56.135943
12/31/2020    2506.840499      2627.907933    5.714479  0.974179  56.135943
01/01/2021    2506.840499      2627.907933    5.714479  0.974179  56.135943
01/02/2021    2506.840499      2627.907933    5.714479  0.974179  56.135943
01/03/2021     692.180873       776.175714    5.714479  0.974179  87.299865
01/04/2021     699.075511       820.917143    5.714479  0.456131  73.811308
01/05/2021     724.999253       874.030000    5.714479  0.551201  85.744848
01/06/2021     760.792538       939.442857    5.714479  0.537977  98.648837
01/07/2021     827.795510      1009.218571    5.714479  0.566155  88.068270
01/08/2021     926.414383      1078.635714    5.714479  0.629751  86.475268
01/09/2021    1048.052871      1150.834286    5.714479  0.649060  94.775828
01/10/2021    1110.902502      1190.504286  182.527750  0.767886  79.445523
01/11/2021    1128.256566      1196.824286  119.676974  0.938188  39.963760
01/12/2021    1108.400918      1189.102857   70.534465  0.989638  31.404304
01/13/2021    1094.305203      1177.188571   61.970522  1.054293  49.030578
01/14/2021    1094.517608      1177.724286   81.673511  1.036080  72.355606
01/15/2021    1089.121546      1170.798571   71.237811  1.015032  58.489241
01/16/2021    1090.022143      1163.264286   77.785230  1.011836  71.886750

                movement
time_period
12/28/2020  -2.906701e+06
12/29/2020  -2.445256e+05
12/30/2020   1.050903e+07
12/31/2020   1.680988e+06
01/01/2021  -3.011855e+06
01/02/2021   1.092775e+07
01/03/2021   9.340068e+07
01/04/2021   7.302116e+07
01/05/2021   1.357327e+07
01/06/2021   3.923449e+07
01/07/2021   1.380818e+07
01/08/2021   1.112242e+07
01/09/2021   4.484735e+07
01/10/2021   7.890774e+06
01/11/2021  -9.133647e+07
01/12/2021  -3.535519e+06
01/13/2021  -1.243958e+07
01/14/2021   5.848099e+07
```

```
01/15/2021  -5.630257e+06
01/16/2021   3.111763e+07

[20 rows x 23 columns]
```

[56]:
```
dfData["next_price_close"] = dfData['price_close'].shift(-1)
dfData.drop(dfData.tail(1).index,inplace=True)
print(len(dfData))
dfData[["price_close", "next_price_close"]]
```

```
587
```

[56]:

time_period	price_close	next_price_close
12/28/2020	729.01	730.91
12/29/2020	730.91	751.98
12/30/2020	751.98	737.27
12/31/2020	737.27	730.79
01/01/2021	730.79	774.73
…	…	…
08/02/2022	1631.03	1618.62
08/03/2022	1618.62	1608.23
08/04/2022	1608.23	1736.57
08/05/2022	1736.57	1690.33
08/06/2022	1690.33	1700.39

[587 rows x 2 columns]

[57]:
```
columns = list(dfData.columns)
target = 'next_price_close'
columns.remove(target)

x_columns = columns
y_column = [target]

print(x_columns)
print(y_column)
```

```
['volume_traded', 'Bullish', 'Positive', 'Negative', 'Hopeful',
'Euphoric/Excited', 'Long_term_investing', 'price_open', 'trades_count',
'Selling', 'price_low', 'Buying', 'price_close', 'Optimistic', 'price_high',
'rsi', 'bolinger_high', 'bolinger_low', 'bolinger_middle', 'macd', 'vortex',
'stoch', 'movement']
['next_price_close']
```

[58]:
```
prediction_days = 28
df_train = dfData[:len(dfData)-prediction_days]
df_test = dfData[len(dfData)-prediction_days:]
```

[59]:
```
print(len(df_train))
print(len(df_test))
```

```
559
28
```

[60]:
```
x_train = df_train[x_columns]
y_train = df_train[y_column]

x_test = df_test[x_columns]
y_test = df_test[y_column]
```

[61]:
```
print(len(x_test))
print(len(y_test))
```

```
28
28
```

[191]:
```
n_estimators = 1000
random_state = 1400
learning_rate = 0.05
```

[192]:
```
model = GradientBoostingRegressor(n_estimators=n_estimators,
                                  random_state = random_state,
                                  learning_rate = learning_rate)
```

[201]:
```
start = timeit.default_timer()
model.fit(x_train, np.ravel(y_train))
stop = timeit.default_timer()

print('Time: ', stop - start)
```

```
Time:  5.557609299954493
```

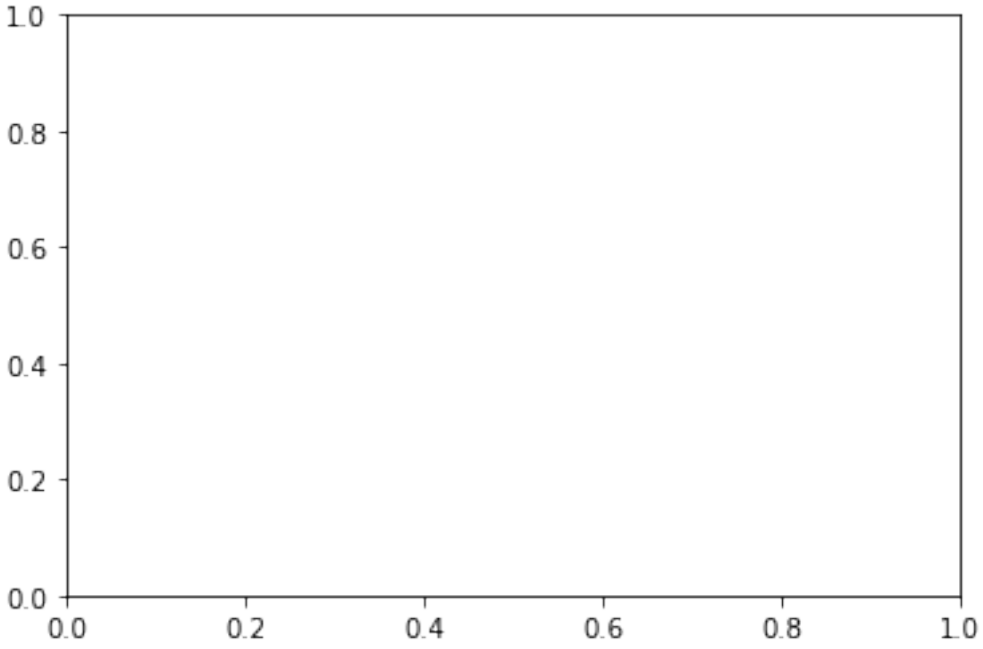
[194]:
```
predictions = model.predict(x_test)
len(predictions)
```

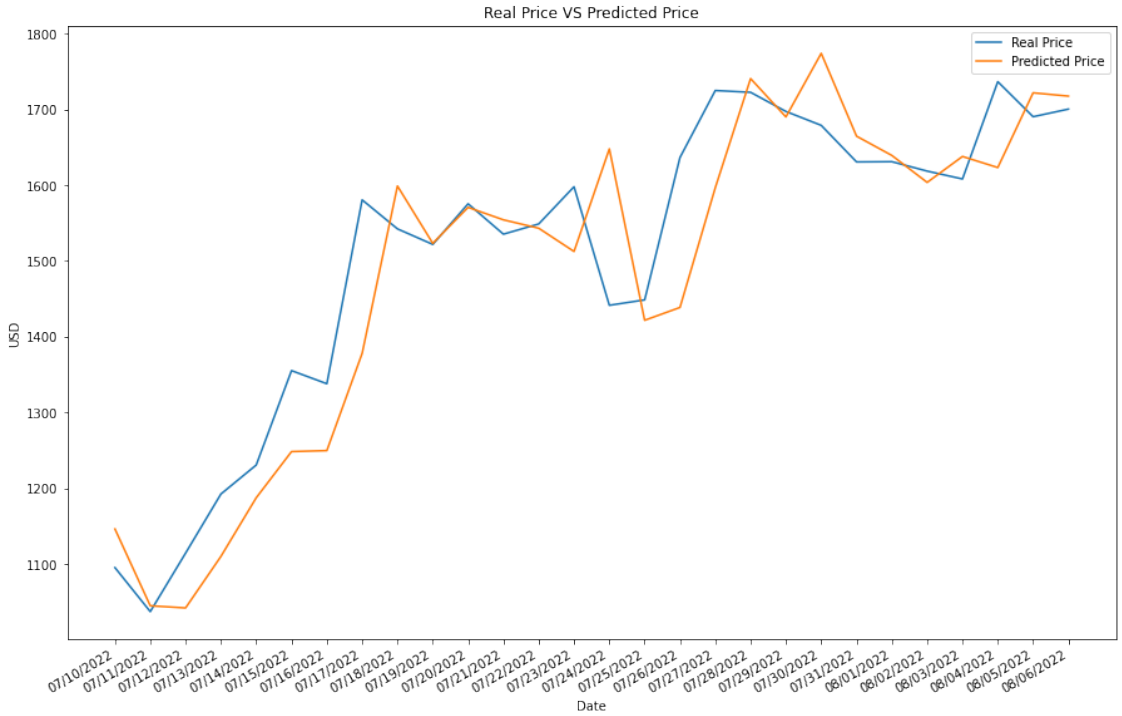
[194]:
```
28
```

[172]:
```
startDate = dt.datetime(2022, 7, 11)
endDate = startDate + dt.timedelta(days=28)
days = mdates.drange(startDate,endDate,dt.timedelta(days=1))
```

[197]:
```
fig = plt.figure()
ax = plt.axes()
plt.subplots(figsize=(15,10))
plt.title("Real Price VS Predicted Price")
plt.xlabel("Date")
```

```
plt.ylabel("USD");
plt.plot(y_test, label="Real Price")
plt.plot(predictions, label="Predicted Price")
plt.gcf().autofmt_xdate()
plt.legend()
plt.savefig('GBRfinalPrediction.jpg')
```





[196]:
```
print("Model Accuracy: %.3f" % model.score(x_test, y_test))
mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
print("The mean absolute error (MAE) on test set: {:.4f}".format(mae))
```

```
Model Accuracy: 0.816
The mean absolute error (MAE) on test set: 62.6732
```

[147]:
```
def learning_rate_estimators_tunning(learning_rates, estimators):
    for learning_rate in learning_rates:
        for estimator in estimators:
            model = GradientBoostingRegressor(n_estimators=estimator,
                                  random_state = 1400,
                                  learning_rate = learning_rate)
            model.fit(x_train, np.ravel(y_train))
            accuracy = model.score(x_test, y_test)
            print("Model Accuracy for {} learning rate and {} estimators is: {:.
 ↪3f}".format(learning_rate, estimator, accuracy))
            mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
            print("The mean absolute error (MAE) for {} learning rate and {}␣
 ↪estimators is: {:.4f}".format(learning_rate, estimator, mae))
```

[148]:
```
LearningRates = [0.1, 0.07, 0.05, 0.03]
Estimators = [300, 500, 700, 1000]
```

```
learning_rate_estimators_tunning(LearningRates, Estimators)
```

```
Model Accuracy for 0.1 learning rate and 300 estimators is: 0.793
The mean absolute error (MAE) for 0.1 learning rate and 300 estimators is:
69.7836
Model Accuracy for 0.1 learning rate and 500 estimators is: 0.790
The mean absolute error (MAE) for 0.1 learning rate and 500 estimators is:
70.3836
Model Accuracy for 0.1 learning rate and 700 estimators is: 0.785
The mean absolute error (MAE) for 0.1 learning rate and 700 estimators is:
70.8876
Model Accuracy for 0.1 learning rate and 1000 estimators is: 0.784
The mean absolute error (MAE) for 0.1 learning rate and 1000 estimators is:
70.6453
Model Accuracy for 0.07 learning rate and 300 estimators is: 0.798
The mean absolute error (MAE) for 0.07 learning rate and 300 estimators is:
68.8968
Model Accuracy for 0.07 learning rate and 500 estimators is: 0.805
The mean absolute error (MAE) for 0.07 learning rate and 500 estimators is:
65.9681
Model Accuracy for 0.07 learning rate and 700 estimators is: 0.801
The mean absolute error (MAE) for 0.07 learning rate and 700 estimators is:
66.7967
Model Accuracy for 0.07 learning rate and 1000 estimators is: 0.797
The mean absolute error (MAE) for 0.07 learning rate and 1000 estimators is:
67.0684
Model Accuracy for 0.05 learning rate and 300 estimators is: 0.796
The mean absolute error (MAE) for 0.05 learning rate and 300 estimators is:
68.5455
Model Accuracy for 0.05 learning rate and 500 estimators is: 0.808
The mean absolute error (MAE) for 0.05 learning rate and 500 estimators is:
66.4689
Model Accuracy for 0.05 learning rate and 700 estimators is: 0.814
The mean absolute error (MAE) for 0.05 learning rate and 700 estimators is:
64.0886
Model Accuracy for 0.05 learning rate and 1000 estimators is: 0.816
The mean absolute error (MAE) for 0.05 learning rate and 1000 estimators is:
62.6732
Model Accuracy for 0.03 learning rate and 300 estimators is: 0.806
The mean absolute error (MAE) for 0.03 learning rate and 300 estimators is:
67.5670
Model Accuracy for 0.03 learning rate and 500 estimators is: 0.805
The mean absolute error (MAE) for 0.03 learning rate and 500 estimators is:
67.8261
Model Accuracy for 0.03 learning rate and 700 estimators is: 0.808
The mean absolute error (MAE) for 0.03 learning rate and 700 estimators is:
66.7611
Model Accuracy for 0.03 learning rate and 1000 estimators is: 0.811
```

```
The mean absolute error (MAE) for 0.03 learning rate and 1000 estimators is:
66.2812
```

[188]:
```
for maxDepth in range(1, 10, 2):
    model = GradientBoostingRegressor(n_estimators=1000,
                                      random_state = 1400,
                                      learning_rate = 0.05,
                                      max_depth=maxDepth )
    model.fit(x_train, np.ravel(y_train))
    accuracy = model.score(x_test, y_test)
    mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
    print("For a depth of size {}, accuracy is {:.3f} and mae is: {:.4f}".
 ↪format(maxDepth,accuracy, mae))
```

```
For a depth of size 1, accuracy is 0.784 and mae is: 73.2350
For a depth of size 3, accuracy is 0.816 and mae is: 62.6732
For a depth of size 5, accuracy is 0.675 and mae is: 82.0411
For a depth of size 7, accuracy is 0.559 and mae is: 105.3699
For a depth of size 9, accuracy is 0.487 and mae is: 111.1284
```

[190]:
```
    for minSplit in range(2,5):
        for minLeaf in range (1,5):
            model = GradientBoostingRegressor(n_estimators=1000,
                                              random_state = 1400,
                                              learning_rate = 0.05,
                                              min_samples_split=minSplit,
                                              min_samples_leaf=minLeaf)
            model.fit(x_train, np.ravel(y_train))
            accuracy = model.score(x_test, y_test)
            mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
            print("For a min split of size {} and min leaf of size {}, accuracy␣
 ↪is {:.3f} and mae is: {:.4f}".format(minSplit, minLeaf, accuracy, mae))
```

```
For a min split of size 2 and min leaf of size 1, accuracy is 0.816 and mae is:
62.6732
For a min split of size 2 and min leaf of size 2, accuracy is 0.801 and mae is:
67.4993
For a min split of size 2 and min leaf of size 3, accuracy is 0.801 and mae is:
64.8576
For a min split of size 2 and min leaf of size 4, accuracy is 0.802 and mae is:
67.4073
For a min split of size 3 and min leaf of size 1, accuracy is 0.801 and mae is:
66.7934
For a min split of size 3 and min leaf of size 2, accuracy is 0.801 and mae is:
67.4993
For a min split of size 3 and min leaf of size 3, accuracy is 0.801 and mae is:
64.8576
For a min split of size 3 and min leaf of size 4, accuracy is 0.802 and mae is:
67.4073
```

```
For a min split of size 4 and min leaf of size 1, accuracy is 0.777 and mae is:
70.6186
For a min split of size 4 and min leaf of size 2, accuracy is 0.801 and mae is:
67.4993
For a min split of size 4 and min leaf of size 3, accuracy is 0.801 and mae is:
64.8576
For a min split of size 4 and min leaf of size 4, accuracy is 0.802 and mae is:
67.4073
```

[ ]:

[202]:
```
predictions
```

[202]:
```
array([1146.51202779, 1045.2202378 , 1042.39338974, 1110.31519168,
       1187.67683071, 1248.61913944, 1249.92473902, 1378.20122727,
       1598.94495569, 1523.31323029, 1570.76073766, 1554.3481477 ,
       1543.26813139, 1512.58719574, 1647.92272569, 1421.82708751,
       1438.75139351, 1596.69601011, 1740.64941409, 1690.07173211,
       1773.94964388, 1664.68187294, 1639.27268087, 1603.70184506,
       1637.9204757 , 1623.29534144, 1721.87130261, 1717.52181195])
```

[204]:
```
dfPred = pd.DataFrame(predictions)
```

[206]:
```
dfPred.to_csv("GBRpredictions.csv")
```

[207]:
```
metrics.max_error(y_test, predictions)
```

[207]:
```
206.31272569161138
```

[208]:
```
metrics.mean_absolute_error(y_test, predictions)
```

[208]:
```
62.6731619501949
```

[209]:
```
metrics.r2_score(y_test, predictions)
```

[209]:
```
0.8160101948962725
```

[210]:
```
metrics.median_absolute_error(y_test, predictions)
```

[210]:
```
38.58252111567401
```

[ ]:

Appendix D

Long Short Term Memory Generation Code

LTSM Model

September 21, 2022

[697]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import keras_tuner as kt
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
plt.style.use('seaborn-whitegrid')
import seaborn as sb
import datetime as dt
import timeit
```

[670]:
```
dfData = pd.read_csv("dfData.csv")
dfData.set_index('time_period', inplace = True)
dfData.fillna(value = 0, inplace=True)
print(len(dfData))
dfData.head(20)
```

```
588
```

[670]:

time_period	volume_traded	Bullish	Positive	Negative	Hopeful
12/28/2020	25763.51754	89	166	122	12
12/29/2020	16248.29869	56	144	133	10
12/30/2020	14385.47579	66	101	53	7
12/31/2020	18938.33743	48	112	89	12
01/01/2021	15129.68596	24	83	74	3
01/02/2021	26362.64832	75	129	106	10
01/03/2021	86710.73835	169	325	191	27
01/04/2021	88682.01485	120	261	212	9
01/05/2021	48366.89405	105	205	177	11
01/06/2021	51948.05131	106	185	110	14
01/07/2021	50785.01095	86	148	110	16

01/08/2021	55425.91834	77	125	72	4
01/09/2021	29911.63100	77	160	106	15
01/10/2021	57780.80793	86	166	135	17
01/11/2021	140653.08015	84	145	156	16
01/12/2021	57895.78435	69	147	108	15
01/13/2021	37266.08719	60	120	73	14
01/14/2021	48922.97282	71	132	74	12
01/15/2021	58864.46004	73	136	102	17
01/16/2021	45301.65087	88	147	83	12

time_period	Euphoric/Excited	Long_term_investing	price_open	trades_count	\
12/28/2020	27	79	684.42	15333	
12/29/2020	13	53	729.02	11362	
12/30/2020	19	36	731.79	9679	
12/31/2020	20	39	752.50	9639	
01/01/2021	16	28	736.90	7544	
01/02/2021	24	45	731.19	13326	
01/03/2021	88	104	775.24	38600	
01/04/2021	59	100	980.02	50719	
01/05/2021	46	58	1043.99	32159	
01/06/2021	48	86	1103.50	33189	
01/07/2021	28	67	1212.62	29682	
01/08/2021	26	71	1226.49	32663	
01/09/2021	36	51	1217.70	21356	
01/10/2021	27	65	1281.01	35595	
01/11/2021	19	90	1255.39	65237	
01/12/2021	36	47	1087.18	27440	
01/13/2021	15	44	1049.31	20490	
01/14/2021	29	63	1130.00	28730	
01/15/2021	22	68	1231.38	33230	
01/16/2021	22	58	1169.68	29581	

time_period	Selling	…	Optimistic	price_high	rsi	bolinger_high	\
12/28/2020	35	…	44	746.98	0.000000	0.000000	
12/29/2020	26	…	35	739.09	0.000000	0.000000	
12/30/2020	16	…	26	758.60	0.000000	0.000000	
12/31/2020	11	…	35	756.04	0.000000	0.000000	
01/01/2021	12	…	28	750.39	0.000000	0.000000	
01/02/2021	19	…	34	788.89	0.000000	0.000000	
01/03/2021	58	…	42	1021.85	94.762168	860.170556	
01/04/2021	42	…	55	1168.00	95.899571	942.758774	
01/05/2021	19	…	58	1135.42	96.695249	1023.060747	
01/06/2021	32	…	30	1216.65	97.641192	1118.093176	
01/07/2021	35	…	32	1294.91	97.752074	1190.641633	
01/08/2021	28	…	29	1275.00	94.799494	1230.857046	

01/09/2021	18	…	25	1308.26	95.834777	1253.615701
01/10/2021	26	…	31	1351.50	88.123762	1270.106070
01/11/2021	38	…	27	1260.12	52.858556	1265.392006
01/12/2021	19	…	34	1150.70	47.882951	1269.804796
01/13/2021	15	…	22	1136.92	57.495284	1260.071940
01/14/2021	20	…	29	1246.41	66.915858	1260.930964
01/15/2021	19	…	30	1255.00	58.000137	1252.475597
01/16/2021	19	…	22	1292.45	63.484354	1236.506429

time_period	bolinger_low	bolinger_middle	macd	vortex	stoch
12/28/2020	0.000000	0.000000	0.000000	0.000000	0.000000
12/29/2020	0.000000	0.000000	0.000000	0.000000	0.000000
12/30/2020	0.000000	0.000000	0.000000	0.000000	0.000000
12/31/2020	0.000000	0.000000	0.000000	0.000000	0.000000
01/01/2021	0.000000	0.000000	0.000000	0.000000	0.000000
01/02/2021	0.000000	0.000000	0.000000	0.000000	0.000000
01/03/2021	692.180873	776.175714	0.000000	0.000000	87.299865
01/04/2021	699.075511	820.917143	0.000000	0.456131	73.811308
01/05/2021	724.999253	874.030000	0.000000	0.551201	85.744848
01/06/2021	760.792538	939.442857	0.000000	0.537977	98.648837
01/07/2021	827.795510	1009.218571	0.000000	0.566155	88.068270
01/08/2021	926.414383	1078.635714	0.000000	0.629751	86.475268
01/09/2021	1048.052871	1150.834286	0.000000	0.649060	94.775828
01/10/2021	1110.902502	1190.504286	182.527750	0.767886	79.445523
01/11/2021	1128.256566	1196.824286	119.676974	0.938188	39.963760
01/12/2021	1108.400918	1189.102857	70.534465	0.989638	31.404304
01/13/2021	1094.305203	1177.188571	61.970522	1.054293	49.030578
01/14/2021	1094.517608	1177.724286	81.673511	1.036080	72.355606
01/15/2021	1089.121546	1170.798571	71.237811	1.015032	58.489241
01/16/2021	1090.022143	1163.264286	77.785230	1.011836	71.886750

time_period	movement
12/28/2020	0.000000e+00
12/29/2020	-2.445256e+05
12/30/2020	1.050903e+07
12/31/2020	1.680988e+06
01/01/2021	-3.011855e+06
01/02/2021	1.092775e+07
01/03/2021	9.340068e+07
01/04/2021	7.302116e+07
01/05/2021	1.357327e+07
01/06/2021	3.923449e+07
01/07/2021	1.380818e+07
01/08/2021	1.112242e+07
01/09/2021	4.484735e+07

```
01/10/2021   7.890774e+06
01/11/2021  -9.133647e+07
01/12/2021  -3.535519e+06
01/13/2021  -1.243958e+07
01/14/2021   5.848099e+07
01/15/2021  -5.630257e+06
01/16/2021   3.111763e+07

[20 rows x 23 columns]
```

[671]:
```
prediction_days = 35 #five weeks in advance (first week will be shifted in test)
df_train = dfData[:len(dfData)-prediction_days+7]# adding one week since in␣
 ↪test input will be shifted
df_test = dfData[len(dfData)-prediction_days:]
```

[672]:
```
print(len(df_train))
print(len(df_test))
```

```
560
35
```

[673]:
```
# Data preprocess
training_set = df_train.to_numpy()
print(len(training_set))
dfData.dtypes
```

```
560
```

[673]:
```
volume_traded          float64
Bullish                  int64
Positive                 int64
Negative                 int64
Hopeful                  int64
Euphoric/Excited         int64
Long_term_investing      int64
price_open             float64
trades_count             int64
Selling                  int64
price_low              float64
Buying                   int64
price_close            float64
Optimistic               int64
price_high             float64
rsi                    float64
bolinger_high          float64
bolinger_low           float64
bolinger_middle        float64
macd                   float64
```

```
vortex                  float64
stoch                   float64
movement                float64
dtype: object
```

[674]:
```
def create_train_test_list(df_train, df_test, target, window):
    target_index = df_train.columns.get_loc(target)
    num_cols = len(df_train.columns)
    trainList = []
    testList = []

    for index in range(num_cols):
        inputs = []
        for i in range(0, df_train.shape[0]-window):
            if index == target_index:
                inputs.append(df_train.iloc[i+window, index])
            else:
                inputs.append(df_train.iloc[i:i+window, index])
        trainList.append(inputs)

    for index in range(num_cols):
        inputs = []
        for i in range(0, df_test.shape[0]-window):
            if index == target_index:
                inputs.append(df_test.iloc[i+window, index])
            else:
                inputs.append(df_test.iloc[i:i+window, index])
        testList.append(inputs)

    return trainList, testList
```

[675]:
```
target = "price_close"
trainList, testList = create_train_test_list(df_train, df_test, target, 7)
```

[676]:
```
target_index = df_train.columns.get_loc(target)
y_train = trainList[target_index]
y_train = np.reshape(y_train, (len(y_train), 1))
y_test = testList[target_index]
y_test = np.reshape(y_test, (len(y_test), 1))

trainList[target_index] = y_train
testList[target_index] = y_test

scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

for i in range(len(trainList)):
```

```
    if i == target_index:
        trainList[i] = scaler_y.fit_transform(trainList[i])
    else:
        trainList[i] = scaler_x.fit_transform(trainList[i])


for i in range(len(testList)):
    if i == target_index:
        testList[i] = scaler_y.fit_transform(testList[i])
    else:
        testList[i] = scaler_x.fit_transform(testList[i])

y_train = trainList[target_index]
y_test = testList[target_index]

print(y_train.shape)
print(y_test.shape)
```

```
(553, 1)
(28, 1)
```

[677]:
```
del trainList[target_index]
del testList[target_index]
```

[678]:
```
x_train = np.stack(trainList, axis=2)
x_test = np.stack(testList, axis=2)
```

[679]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(553, 7, 22)
(28, 7, 22)
```

1 Final Model

[693]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.LSTM(units=350, activation="relu", return_sequences = False,␣
 ↪input_shape=(x_train.shape[1], x_train.shape[2])),
  tf.keras.layers.Dense(40, activation="relu"),
  tf.keras.layers.Dense(1)
])
model.summary()
```

```
Model: "sequential_148"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_149 (LSTM)              (None, 350)               522200
```

```
_________________________________________________________________
dense_296 (Dense)            (None, 40)                14040
_________________________________________________________________
dense_297 (Dense)            (None, 1)                 41
=================================================================
Total params: 536,281
Trainable params: 536,281
Non-trainable params: 0
_________________________________________________________________
```

[694]:
```
model.compile(loss="mean_squared_error", optimizer="adam", metrics="mae")
```

[695]:
```
early_stopping = keras.callbacks.EarlyStopping(monitor='loss',
                                               patience=5,
                                               restore_best_weights=True)
```

[698]:
```
start = timeit.default_timer()
model.fit(x_train,y_train, epochs=50, batch_size=2, callbacks=[early_stopping])
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Epoch 1/50
277/277 [==============================] - 10s 20ms/step - loss: 0.0068 - mae:
0.0608
Epoch 2/50
277/277 [==============================] - 5s 20ms/step - loss: 0.0034 - mae:
0.0459
Epoch 3/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0023 - mae:
0.0362
Epoch 4/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0022 - mae:
0.0355: 0s - loss: 0.0022 - mae: 0.035
Epoch 5/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0028 - mae:
0.0409
Epoch 6/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0022 - mae:
0.0357
Epoch 7/50
277/277 [==============================] - 5s 20ms/step - loss: 0.0019 - mae:
0.0330
Epoch 8/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0020 - mae:
0.0319
Epoch 9/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0022 - mae:
0.0360: 0s - loss: 0.0020
```

```
Epoch 10/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0017 - mae:
0.0305
Epoch 11/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0019 - mae:
0.0320
Epoch 12/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0021 - mae:
0.0347
Epoch 13/50
277/277 [==============================] - 6s 21ms/step - loss: 0.0018 - mae:
0.0319
Epoch 14/50
277/277 [==============================] - 6s 21ms/step - loss: 0.0017 - mae:
0.0306
Epoch 15/50
277/277 [==============================] - 6s 21ms/step - loss: 0.0019 - mae:
0.0320: 1s - loss:
Epoch 16/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0018 - mae:
0.0319
Epoch 17/50
277/277 [==============================] - 6s 20ms/step - loss: 0.0017 - mae:
0.0324
Time:  100.19069839996519
```

[699]:
```
predictions = model.predict(x_test)
unscalledPredictions = predictions.tolist()
unscalledPredictions = scaler_y.inverse_transform(unscalledPredictions)
```

[700]:
```
unscalledPredictions
```

[700]:
```
array([[1082.33618182],
       [1058.12195358],
       [1054.91338861],
       [1075.57613584],
       [1142.28985728],
       [1164.86849206],
       [1197.22817411],
       [1232.17230732],
       [1384.63963462],
       [1472.82010361],
       [1517.64903546],
       [1502.51891679],
       [1526.21327481],
       [1504.307149  ],
       [1598.35407131],
```

```
       [1481.52401148],
       [1450.08609174],
       [1532.09126995],
       [1691.37981332],
       [1668.00483457],
       [1659.21205051],
       [1658.72896072],
       [1629.47048675],
       [1673.31082208],
       [1637.3337584 ],
       [1595.09852784],
       [1689.95950269],
       [1680.08791256]])
```

[701]:
```
unscalledTest = y_test.tolist()
unscalledTest = scaler_y.inverse_transform(unscalledTest)
unscalledTest
```

[701]:
```
array([[1095.55],
       [1037.51],
       [1114.67],
       [1192.6 ],
       [1230.89],
       [1355.44],
       [1338.12],
       [1580.67],
       [1542.41],
       [1521.91],
       [1575.49],
       [1535.43],
       [1548.89],
       [1597.99],
       [1441.61],
       [1448.73],
       [1636.54],
       [1724.91],
       [1722.54],
       [1697.3 ],
       [1678.95],
       [1630.73],
       [1631.03],
       [1618.62],
       [1608.23],
       [1736.57],
       [1690.33],
       [1700.39]])
```

[702]:
```
metrics.max_error(unscalledTest, unscalledPredictions)
```

[702]:
```
348.4976926791669
```

[703]:
```
metrics.mean_absolute_error(unscalledTest, unscalledPredictions)
```

[703]:
```
82.77287272603255
```

[716]:
```
metrics.r2_score(unscalledTest, unscalledPredictions)
```

[716]:
```
0.6800256104079136
```

[717]:
```
metrics.median_absolute_error(unscalledTest, unscalledPredictions)
```

[717]:
```
51.89035923540587
```

[ ]:
```

```

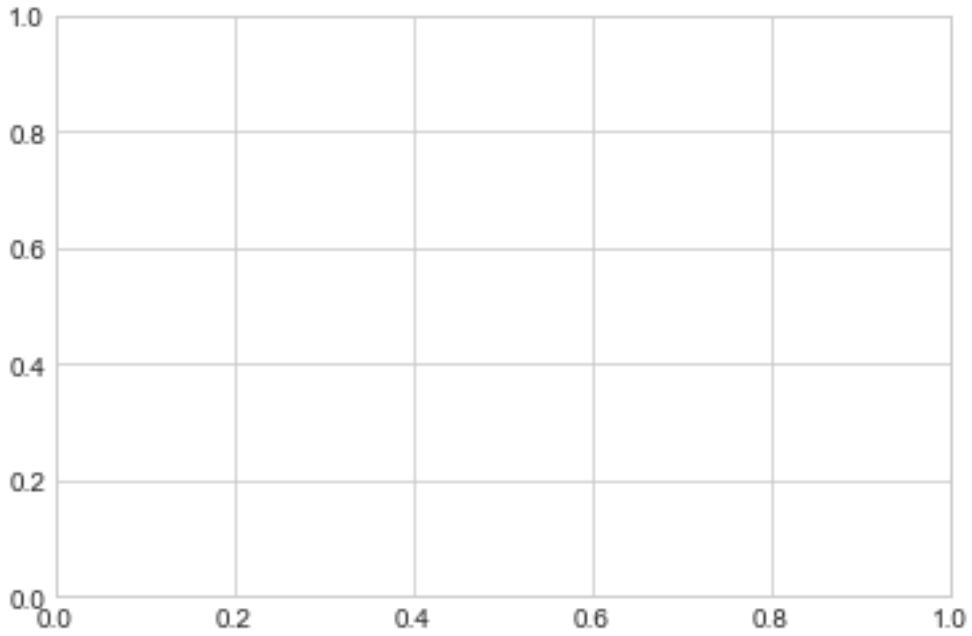
[704]:
```
startDate = dt.datetime(2022, 7, 11)
endDate = startDate + dt.timedelta(days=28)

days = mdates.drange(startDate,endDate,dt.timedelta(days=1))

len(days)
```

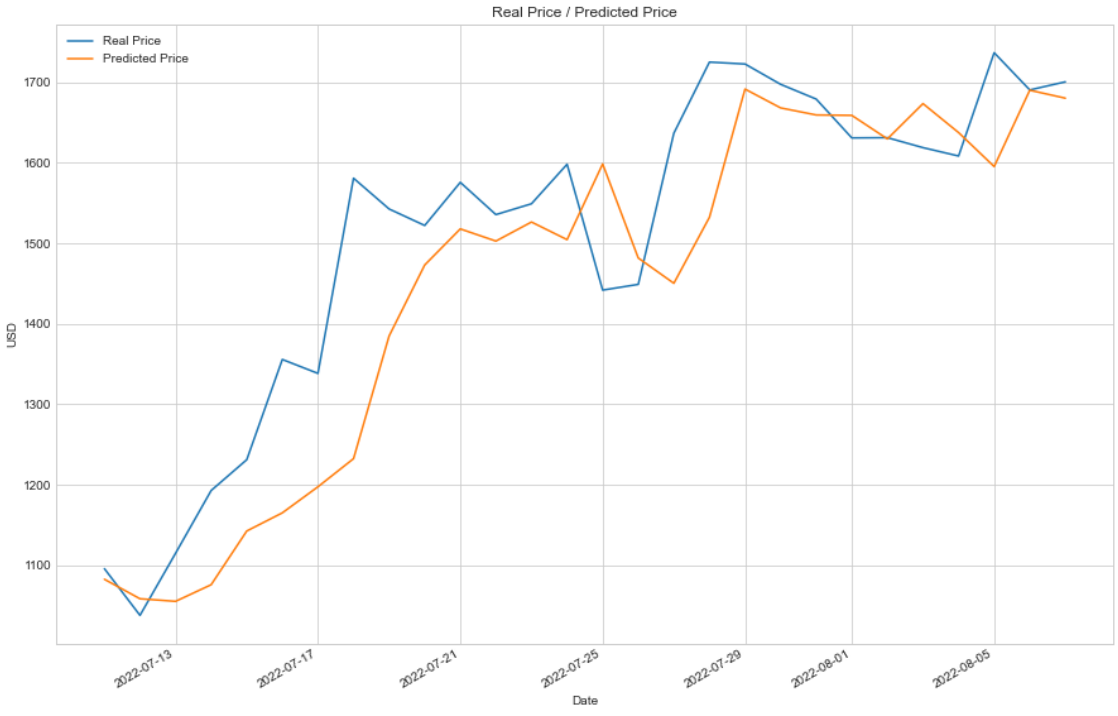
[704]:
```
28
```

[705]:
```
y_pred = np.reshape(unscalledPredictions, 28).tolist()
y = np.reshape(unscalledTest, 28).tolist()
```

[706]:
```
base = dt.datetime(2022, 8, 7)
date_list = [base - dt.timedelta(days=x) for x in range(28)]
date_list.reverse()
```

[707]:
```
fig = plt.figure()
ax = plt.axes()
plt.subplots(figsize=(15,10))
plt.title("Real Price / Predicted Price")
plt.xlabel("Date")
plt.ylabel("USD")
plt.plot(date_list, y, label="Real Price")
plt.plot(date_list, y_pred, label="Predicted Price")
plt.gcf().autofmt_xdate()
plt.legend()
plt.savefig('LTSMfinalPrediction.jpg')
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Real Price / Predicted Price
Real Price
Predicted Price
USD
1700
1600
1500
1400
1300
1200
1100
2022-07-13
2022-07-17
2022-07-21
2022-07-25
2022-07-29
2022-08-01
2022-08-05
Date

2 Tunning

[490]:
```
early_stopping = keras.callbacks.EarlyStopping(monitor='loss',
                                               patience=5
                                               )
for ltsmNeurons in range(300, 360, 10):
    for denseNeurons in range(32,46,4):
        model = tf.keras.models.Sequential([
        tf.keras.layers.LSTM(ltsmNeurons, return_sequences = False,␣
↪input_shape=(x_train.shape[1], x_train.shape[2])),
        tf.keras.layers.Dense(denseNeurons, activation="relu"),
        tf.keras.layers.Dense(1)
        ])
        model.compile(loss='mean_squared_error', optimizer='adam',␣
↪metrics='mae')
        model.fit(x_train,y_train, epochs=50, batch_size=2,␣
↪callbacks=[early_stopping], verbose=False)
        predictions = model.predict(x_test)
        unscalledPredictions = predictions.tolist()
        unscalledPredictions = scaler_y.inverse_transform(unscalledPredictions)
        mae = metrics.max_error(unscalledTest, unscalledPredictions)
        maxError = metrics.mean_absolute_error(unscalledTest,␣
↪unscalledPredictions)
        print("METRICS FOR LTSM {} units and Dense {} units. MAE: {:.3f}, MAX:␣
↪{:.3f}".format(ltsmNeurons, denseNeurons, mae, maxError))
```

```
METRICS FOR LTSM 300 units and Dense 32 units. MAE: 380.647, MAX: 106.204
METRICS FOR LTSM 300 units and Dense 36 units. MAE: 399.914, MAX: 113.663
METRICS FOR LTSM 300 units and Dense 40 units. MAE: 355.099, MAX: 86.212
METRICS FOR LTSM 300 units and Dense 44 units. MAE: 382.334, MAX: 88.810
METRICS FOR LTSM 310 units and Dense 32 units. MAE: 384.698, MAX: 103.801
METRICS FOR LTSM 310 units and Dense 36 units. MAE: 383.343, MAX: 96.201
METRICS FOR LTSM 310 units and Dense 40 units. MAE: 398.718, MAX: 115.347
METRICS FOR LTSM 310 units and Dense 44 units. MAE: 377.412, MAX: 105.532
METRICS FOR LTSM 320 units and Dense 32 units. MAE: 398.778, MAX: 118.450
METRICS FOR LTSM 320 units and Dense 36 units. MAE: 344.263, MAX: 79.605
METRICS FOR LTSM 320 units and Dense 40 units. MAE: 384.947, MAX: 102.230
METRICS FOR LTSM 320 units and Dense 44 units. MAE: 355.056, MAX: 94.162
METRICS FOR LTSM 330 units and Dense 32 units. MAE: 384.411, MAX: 98.642
METRICS FOR LTSM 330 units and Dense 36 units. MAE: 383.901, MAX: 95.608
METRICS FOR LTSM 330 units and Dense 40 units. MAE: 363.394, MAX: 89.063
METRICS FOR LTSM 330 units and Dense 44 units. MAE: 369.064, MAX: 91.664
METRICS FOR LTSM 340 units and Dense 32 units. MAE: 335.949, MAX: 86.910
METRICS FOR LTSM 340 units and Dense 36 units. MAE: 422.149, MAX: 123.094
METRICS FOR LTSM 340 units and Dense 40 units. MAE: 376.697, MAX: 103.289
METRICS FOR LTSM 340 units and Dense 44 units. MAE: 373.124, MAX: 108.734
METRICS FOR LTSM 350 units and Dense 32 units. MAE: 400.355, MAX: 95.081
```

```
METRICS FOR LTSM 350 units and Dense 36 units. MAE: 365.936, MAX: 95.480
METRICS FOR LTSM 350 units and Dense 40 units. MAE: 341.027, MAX: 76.467
METRICS FOR LTSM 350 units and Dense 44 units. MAE: 351.550, MAX: 96.419
```

3 Initial

[477]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.LSTM(units=320, return_sequences = False,␣
 ↪input_shape=(x_train.shape[1], x_train.shape[2])),
  tf.keras.layers.Dense(40, activation="relu"),
  tf.keras.layers.Dense(1)
])
model.summary()
```

```
Model: "sequential_98"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_99 (LSTM)               (None, 320)               439040
_________________________________________________________________
dense_196 (Dense)            (None, 40)                12840
_________________________________________________________________
dense_197 (Dense)            (None, 1)                 41
=================================================================
Total params: 451,921
Trainable params: 451,921
Non-trainable params: 0
_________________________________________________________________
```

[478]:
```
model.compile(loss="mean_squared_error", optimizer="adam",␣
 ↪metrics="mean_absolute_error")
```

[479]:
```
early_stopping = keras.callbacks.EarlyStopping(monitor='loss',
                                               patience=5
                                               )
```

[480]:
```
model.fit(x_train,y_train, epochs=50, batch_size=2, callbacks=[early_stopping])
```

```
Epoch 1/50
277/277 [==============================] - 5s 10ms/step - loss: 0.0095 -
mean_absolute_error: 0.0633
Epoch 2/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0029 -
mean_absolute_error: 0.0410
Epoch 3/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0027 -
mean_absolute_error: 0.0407
Epoch 4/50
```

```
277/277 [==============================] - 3s 10ms/step - loss: 0.0025 -
mean_absolute_error: 0.0381
Epoch 5/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0026 -
mean_absolute_error: 0.0386
Epoch 6/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0024 -
mean_absolute_error: 0.0375
Epoch 7/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0020 -
mean_absolute_error: 0.0338
Epoch 8/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0020 -
mean_absolute_error: 0.0336
Epoch 9/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0021 -
mean_absolute_error: 0.0343
Epoch 10/50
277/277 [==============================] - 3s 10ms/step - loss: 0.0018 -
mean_absolute_error: 0.0329
Epoch 11/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0017 -
mean_absolute_error: 0.0307
Epoch 12/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0017 -
mean_absolute_error: 0.0311
Epoch 13/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0018 -
mean_absolute_error: 0.0319
Epoch 14/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0019 -
mean_absolute_error: 0.0331
Epoch 15/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0019 -
mean_absolute_error: 0.0328
Epoch 16/50
277/277 [==============================] - 3s 11ms/step - loss: 0.0019 -
mean_absolute_error: 0.0326
```

[480]:
```
<tensorflow.python.keras.callbacks.History at 0x1e89c380b20>
```

[481]:
```
predictions = model.predict(x_test)
unscalledPredictions = predictions.tolist()
unscalledPredictions = scaler_y.inverse_transform(unscalledPredictions)
```

[482]:
```
unscalledTest = y_test.tolist()
unscalledTest = scaler_y.inverse_transform(unscalledTest)
```

[483]:
```
metrics.max_error(unscalledTest, unscalledPredictions)
```

[483]:
```
353.2005679684878
```

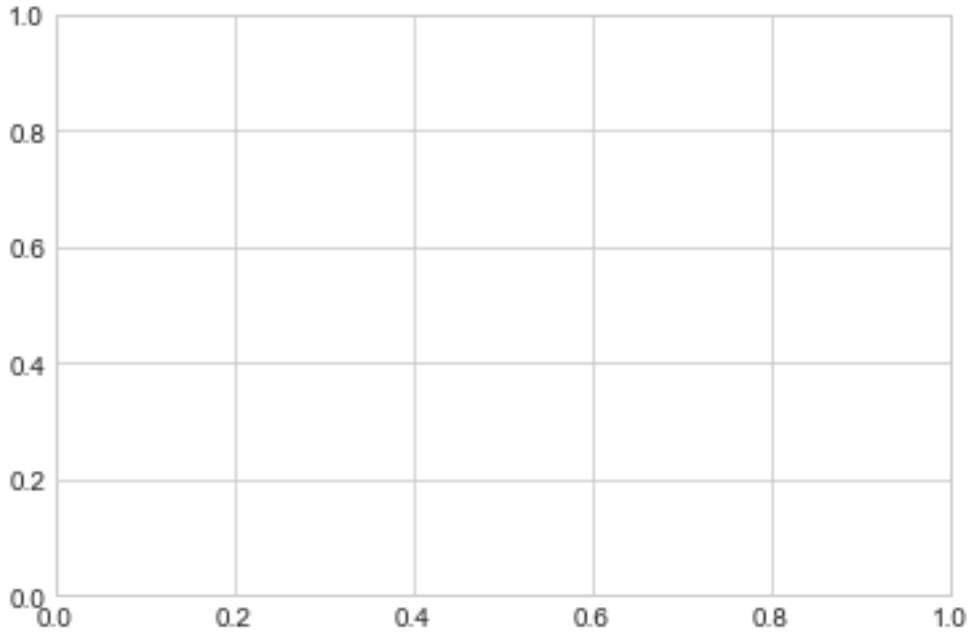
[484]:
```
metrics.mean_absolute_error(unscalledTest, unscalledPredictions)
```

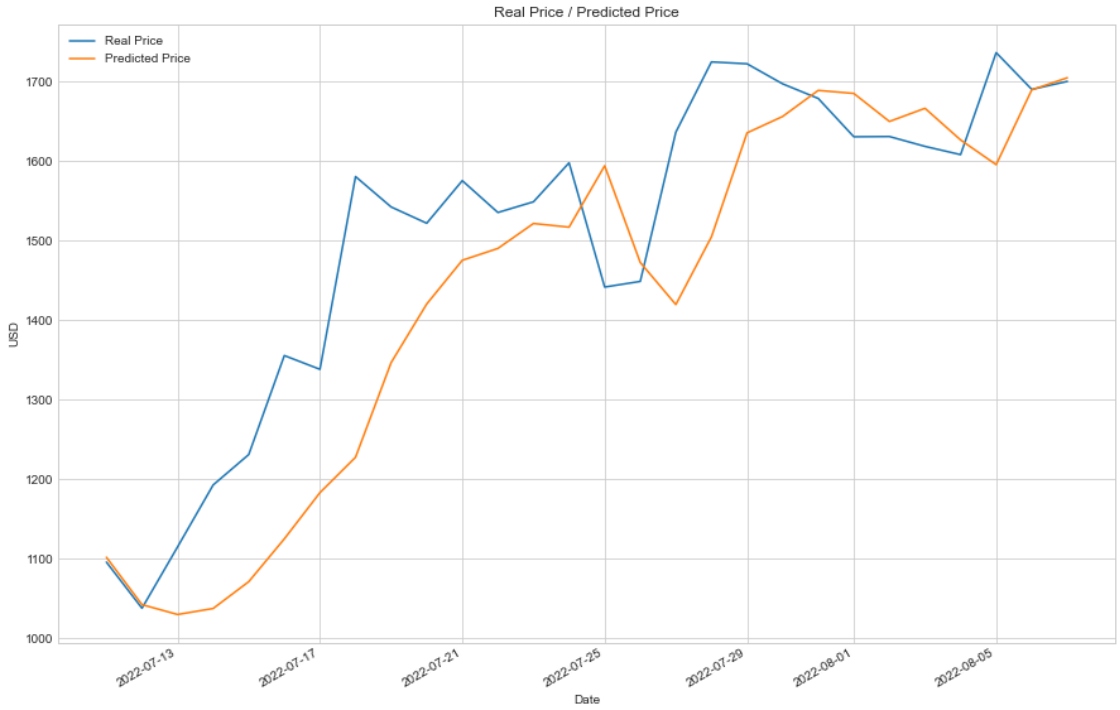
[484]:
```
97.813162180796
```

[485]:
```
y_pred = np.reshape(unscalledPredictions, 28).tolist()
y = np.reshape(unscalledTest, 28).tolist()
```

[486]:
```
base = dt.datetime(2022, 8, 7)
date_list = [base - dt.timedelta(days=x) for x in range(28)]
date_list.reverse()
```

[487]:
```
fig = plt.figure()
ax = plt.axes()
plt.subplots(figsize=(15,10))
plt.title("Real Price / Predicted Price")
plt.xlabel("Date")
plt.ylabel("USD")
plt.plot(date_list, y, label="Real Price")
plt.plot(date_list, y_pred, label="Predicted Price")
plt.gcf().autofmt_xdate()
plt.legend()
plt.savefig('LTSMinitialPrediction.jpg')
```





[ ]:

[708]:
```
gbrPred = pd.read_csv("GBRpredictions.csv")
```

[713]:
```
y_gbr = []
for item in gbrPred["0"]:
    y_gbr.append(item)
```

[710]:
```
y_pred
```

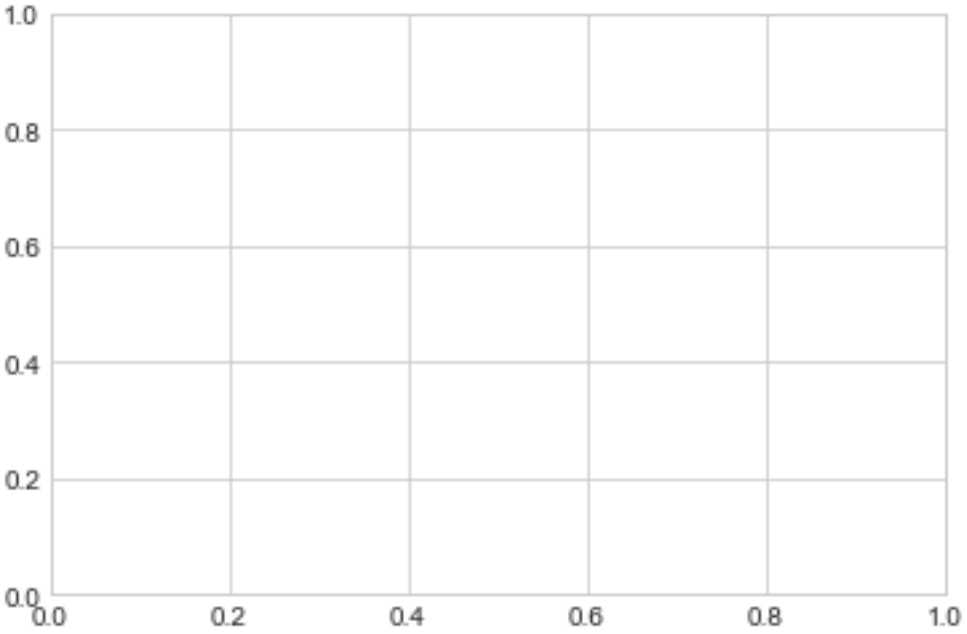
[710]:
```
[1082.336181819439,
 1058.1219535810128,
 1054.9133886072038,
 1075.576135835126,
 1142.289857276678,
 1164.8684920632838,
 1197.2281741058825,
 1232.1723073208332,
 1384.6396346187591,
 1472.8201036059857,
 1517.649035462141,
 1502.5189167892931,
 1526.2132748055458,
 1504.3071489977835,
```

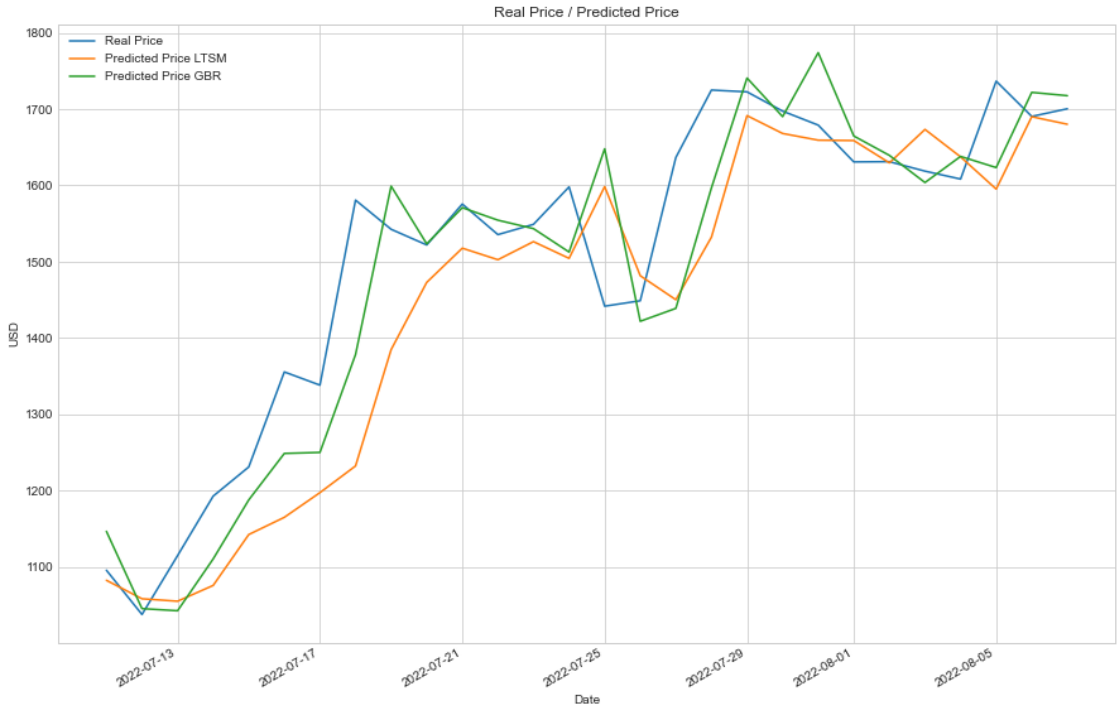
```
        1598.3540713095665,
        1481.5240114808082,
        1450.0860917448997,
        1532.0912699508667,
        1691.3798133206367,
        1668.0048345685004,
        1659.2120505070686,
        1658.7289607238768,
        1629.4704867529867,
        1673.3108220767974,
        1637.333758404255,
        1595.098527843952,
        1689.9595026910304,
        1680.0879125595093]
```

[714]:
```
y_gbr
```

[714]:
```
[1146.5120277905507,
 1045.220237795289,
 1042.3933897391285,
 1110.3151916792938,
 1187.6768307093464,
 1248.619139442523,
 1249.9247390181094,
 1378.2012272726067,
 1598.9449556891682,
 1523.3132302927895,
 1570.7607376599185,
 1554.3481476964907,
 1543.2681313939013,
 1512.5871957376305,
 1647.9227256916113,
 1421.8270875112476,
 1438.7513935086092,
 1596.6960101098455,
 1740.6494140917864,
 1690.0717321146567,
 1773.9496438803862,
 1664.6818729406946,
 1639.2726808709106,
 1603.7018450561193,
 1637.9204757006487,
 1623.295341435199,
 1721.8713026078274,
 1717.5218119454394]
```

[715]:
```
fig = plt.figure()
ax = plt.axes()
plt.subplots(figsize=(15,10))
plt.title("Real Price / Predicted Price")
plt.xlabel("Date")
plt.ylabel("USD")
plt.plot(date_list, y, label="Real Price")
plt.plot(date_list, y_pred, label="Predicted Price LTSM")
plt.plot(date_list, y_gbr, label="Predicted Price GBR")
plt.gcf().autofmt_xdate()
plt.legend()
plt.savefig('ComparisonPrediction.jpg')
```




Real Price / Predicted Price
Real Price
Predicted Price LTSM
Predicted Price GBR
USD
1800
1700
1600
1500
1400
1300
1200
1100
2022-07-13
2022-07-17
2022-07-21
2022-07-25
2022-07-29
2022-08-01
2022-08-05
Date

[ ]: