University of Essex
Department of Mathematical Sciences

MA981: DISSERTATION

Cryptocurrency Price Prediction with Deep Learning

Muhammad Lashan Ali Zahid

Supervisor: Dr. Joe Bailey

August 26, 2022
Colchester

Abstract

Blockchain technology has become increasing popular in the past few years. With the invent of crypto in 2008, the world today is working on the the 4^{th} generation of Blockchain technology. However, the cryptocurrency market is just as unstable as it was in its beginning. It can be affected by many technical, sentimental, and legal factors, so it is highly volatile, uncertain, and unpredictable. These uncertainties makes it hard for the researches to use traditional statistical methods to forecast the crypto market. With the advancements in Deep learning researchers are now considering using deep learning models to predict crypto prices since all the data is available to the public. This study focus on Long short-term memory (LSTM), Gated recurrent units (GRUs) and Bi-Directional LSTM network (BiLSTM) to predict prices of Bitcoin, Ethereum and Cardano. The results shows that the GRU model preforms the best out of the three with the least mean squares errors for all three cryptocurrencies.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Cryptocurrency

The world today has a lot of complex wealth models such as stocks, centralized money, gold reserves, cryptocurrency, etc. In the modern world the concept of money is different. Mostly the wealth travels in just entries on a spreadsheet. After every transaction, the spreadsheets of the account holders are updated which makes the process very simple. However, this form of money has major limitations. One of which is that it is centralized, meaning there is a central authority that regulates it such as a government or central bank. As a result, the government or central bank can print money whenever needed and inflate the money supply on the market (Mittal, Arora, & Bhatia,). The problem with printing money is that when the market is flooded with more money the value of each dollar drops. Up until 2008, all the attempts to create an alternate to the current monetary system were unsuccessful, however in October 2008, a document was published online by a guy with a pseudonym Satoshi Nakamoto (Nakamoto,). The document also called a white paper, suggested a way of creating a system for a decentralized currency called bitcoin. This system claimed to create digital money without the need of a central authority.

At its core, Bitcoin is a transparent ledger without a central authority. Contrary to fiat money, which is centralized and not transparent, a cryptocurrency is a virtual currency based on a ledger which is decentralized and secured by cryptography. There is no one computer

that holds the ledger, with crypto, every computer that participates in the system holds the ledger also known as a Blockchain (Adams, Kewell, & Parry,). The people behind these numerous computers that are constantly recording and updating the transactions made on there ledgers are called cryptocurrency miners. This decentralization makes this concept of currency highly secure as every transaction is shared with everyone in the network. These networked computers add transaction to a shared list of recent transactions, known as a block. Every 10 minutes, the newest block of transactions is added on, or chained, to all the previous blocks (Buterin et al.,). That's how you get a blockchain. To ensure that each block of transactions on the chain is verified, a subset of cryptocurrency's network joins a race to solve a difficult math puzzle. And if they solve it first, their record of the block of transactions becomes the official record. Theyâre rewarded with cryptocurrency itself, and the network gets a new block on the chain. This entire process is known as cryptocurrency mining. The fact that many computers are competing to verify a block ensures that no single computer can monopolize the crypto market. To ensure the competition stays fair and evenly timed, the puzzle becomes harder when more computers join in.

There have been several other cryptocurrencies have emerged since 2008 as an alternative to Bitcoin also known as Altcoins. Some of the famous ones are Ethereum, Dogecoin, Monero, Ripple, Litecoin Steller, etc. This research will particularly focus on Bitcoin (BTC), Ethereum (ETH) and Cardano (ADA).

As of July 2022, there are 20,268 cryptocurrencies in existence with a total market cap of at most 3.1 trillion US dollars. But this all started with Bitcoin, the world's first and most popular cryptocurrency. Bitcoin was made available to public in early 2009, the first 50 bitcoin were created from the mining of the first block called the genesis block. The first transaction was also made with Hal Finney receiving 10 bitcoins from Satoshi Nakamoto. It rose to popularity in 2010 when the first purchase was done using Bitcoin to buy two pizzas. This was a benchmark which proves that Bitcoin, like many other forms of currencies, can be used to purchase merchandise. But it was not like any other currency out there. Although bitcoin is a digital currency, it is powered by an open-source code called the blockchain. To digitally sign a transaction, the bitcoin holder needs to have a Bitcoin wallet which contains a public and private key. To create a signature, a private key and the text from a transaction are fed

Figure 1.1: Bitcoin closing price over the years

into a special cryptographic function. Another function allows other people to check the signature, making sure it was created by the account owner, and that it applies to that specific transaction. Unlike the handwritten version, these signatures can't be copied and reused in the future, as they're unique to each transaction. A digital signature works by utilizing two different but connected keys, a private key to create a signature and a public key that others can use to check it.

The value of bitcoin has grown a lot in the span of a decade. In 2011 bitcoin was worth less than one US dollar which is significantly low compared to the all-time highs of about 68 000 US dollar in the year 2021. However, it is not as simple as it seems. Due to numerous reasons the price of Bitcoin and other cryptocurrencies have always been fluctuation making it an extremely unstable and volatile market. Figure 1.1 illustrated the fluctuations in Bitcoin price in the past 5 years. One of the reasons of the popularity of Bitcoin is its pseudo-anonymous nature. With all the transactions only a public address (public key) is associated with it, which hold no identifying information of the user itself. This anonymity made bitcoin a popular choice for a currency for the darknet to buy and sell illegal commodities. Being the first cryptocurrency, bitcoin got the most attention form financial institutions all around

the world which made bitcoin by the far biggest cryptocurrency in terms of recognition and market cap. As of August 2022, bitcoin dominates the total crypto market cap with a total of almost 400 billion US dollars.

Bitcoin is the basis of many other cryptocurrencies that followed it, Ethereum is one of them. However, Ethereum took the initial idea of bitcoin and applied it to everything, not just money. Ethereum was announced in 2014 in Vitalik Buterin's paper addressing many limitations of bitcoin (Buterin et al.,). Where bitcoin decentralized currency, with Ethereum the idea of decentralized applications emerged. Ethereum is made up of computers all around the world each owned by different people. These computers combine to work like a huge network of computers. This huge network runs applications. Anyone can run applications on it so long as they pay the people providing the computer power. These people are called miners. There miners are like the bitcoin miners, but the difference is Bitcoin miners verify and record Bitcoin transactions in essence they process payments, on the other hand Ethereum miners execute all sorts of programs. These programs can take various forms, they can be complex and ambitious or just a few lines of code. The more complex your program is the more you must pay to run it. Ethereum comes with its own cryptocurrency Ether, which is used to pay to run programs. To run an application using the Ethereum network, thousands of computers all around the world run your application simultaneously you pay a small fee in Ether to the owners of those computers for their trouble. Those fees help incentivize miners to run the network.

One kind of program you can write is called a smart contract. Smart contracts are digital agreements that run on a blockchain. They collapse an agreement and the execution of that agreement into one step. Usually these are two separate steps, step one is sign the contract step two carry out the terms smart contracts take these two steps and combine them into one (Bartoletti & Pompianu,). The decentralized applications often known as 'dapps' can be deployed using these smart contracts. The two use cases of these dapps are called DeFi and NFTs. Ethereum is also a decentralized platform with its own programming language called Solidity and cryptocurrency called Ether. Many old coins that are on the market today such as Zero times and Maker are just Ethereum smart contracts. Ethereum is supported by the non-profit Ethereum foundation and the Ethereum community at large. Developers across

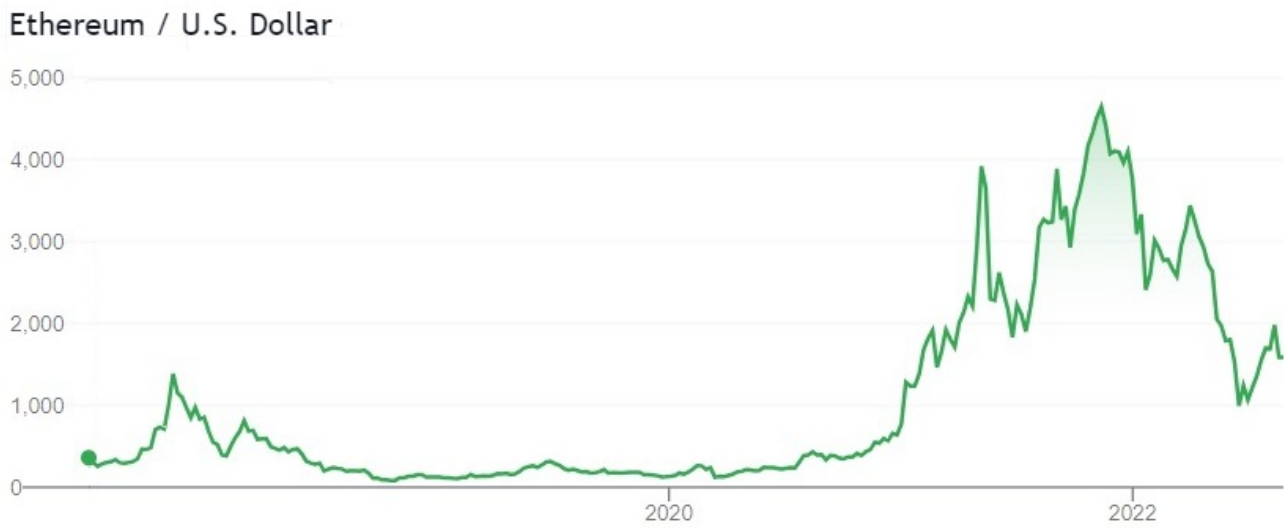

Figure 1.2: Ethereum closing price over the years

the world contribute to improving the product and making the platform more accessible for creators and users.

Since its launch to 2020 a total of 2772 dapps have been launched on the Ethereum network. In 2014 Ether was worth 0.3 US dollars and it gradually increased to 270 US dollars in 2020. The peak value of Ethereum was back in November 2021 which was 4815 US dollars. Figure 1.2 shows the price fluctuations in the Ethereum over the years. Of the total crypto market cap, Ethereum has a market cap of over 185 billion US dollars making it the second most domination crypto currency in the world. Since its launch, many large corporations begun investing in Ethereum. Ethereum also is found in the systems of companies such as Microsoft HSBC and Barclays. As the year progressed Ethereum became increasingly important in large business models. One of the major updates that happened was the update to Ethereum 2.0. Ethereum 2.0 or ETH2 is a set of interconnected upgrades to the Ethereum network that aims at making Ethereum more scalable, more secure, and more sustainable. The previous Ethereum consensus model proof-of-work is a well-known and battle-tested approach to building cryptocurrencies. In proof-of-work miners invest the resources mainly electricity to validate transactions and secure the network. This model requires massive amounts of energy to work properly and protect the network from attacks (Bentov, Gabizon, & Mizrahi,). The proof of stake model tries to address the power consumption concerns by getting rid of the miners completely instead of machines. Securing the network by in-

vesting their resources the proof-of-stake consensus model relies on economic incentives. In proof-of-stake users who want to secure the network stake their Ether and become validators. Each validator is incentivized to validate transactions by receiving, similarly to miners in proof-of-work, both the block reward and the transaction fees. To discourage validators from trying to game the system and validate fraudulent transactions the proof-of-stake system implements a mechanism called slashing, where validators lose part of their staked heath if they decide to act dishonestly. This transition to Ethereum 2.0 did solve many problems like scalability and sustainability for Ethereum, however, it is still soon to tell how this transition will play out.

With Ethereum being categorized as the second generation of blockchain, Cardano is considered to be the third generation. Cardano was launched in 2017 by one of the founders of Ethereum, Charles Hoskinsen. Hoskinson saw that cryptocurrencies would suffer from three main problems these problems are scalability, interoperability, and sustainability (Bartoletti & Pompianu,). To counter these, Hoskinsen created Cardano with more advanced technology. The network is by default built to handle much larger transactions while also offering largely decentralized blockchain ensuring a very high level of security and a low transaction fee. Just like Ether in Ethereum, Ada is the cryptocurrency in the Cardano platform. The Cardano model operates on a model called proof of stake. The Cardano network has validators (individual people, or organizations) that confirm transactions happening on the blockchain. The thing that makes Cardano different from other PoS networks, is Ouroboros (Badertscher, Gaži, Kiayias, Russell, & Zikas,). This is a special type of system made up of epochs (currently, these are periods of 5 days) and slots (1-second intervals). Each slot has a slot leader (who is selected from the transactions validators), and this leader is chosen to confirm transactions. The Ouroboros system allows for the Cardano network to be much more secure than your traditional Proof-of-Stake blockchains, which then allows an increase in scalability. Ecology-wise, Cardanoâs Proof-of-Stake system allows it to be much more eco-friendly than cryptocurrencies such as Bitcoin or Ethereum. Cardano is leaps ahead of the two aforementioned coins, when it comes to energy preservation and ecology concerns. Cardano has a dedicated treasury, itâs filled up with transaction fees, and then those fees are paid out in the form of rewards to anyone who makes a meaningful impact on the ecosystem (developers, scientists, and so on). On top of that, Cardano is actually really big on science

and long-term planning, in general. In 2015, Charles Hoskinson agreed on a collaboration with another co-founder of Ethereum, Jeremy Wood. This collaboration resulted in a brand new company, called IOHK (also known as Input-Output Hong Kong). Cardano is the main point of interest for IOHK, and the company itself is concerned with creating blockchain solutions for both private companies, as well as governmental institutions, worldwide. Cardano also acts as that bridge between two blockchains using a protocol known as the KMZ sidechains protocol. Because of the protocol users are able to trade Cardano coins (or, to be more specific, ADA coins) for other cryptocurrencies.

Cardano has a supply limit of 45 billion Ada available. Cardano's ada crypto token debuted in 2017 at market cap of around 600 million dollars and by 2018 the market cap had blasted up to 10 billion US Dollars. Over that year it would continue to rise to an extremely impressive 33 billion dollars though it eventually dropped back down to 10 billion US dollars. Though since then it has managed to get back up and float around the 50-billion-dollar mark, increasing in 15 billion dollars over the last few months as per digital currency analysis platform Coingecko.com. Figure 1.3 shows the market cap of Cardano over the years. Ada began trading at around two cents a token but since that time it has risen to around one US dollar and fifty cents to one US dollar and sixty cents. Still for most of the coins life it was between 2 cents and 40 cents per token. Ada is just as volatile as most other cryptocurrencies and just like with those other crypto platforms it's more speculation this means that like a lot of cryptocurrencies Ada can be considered quite a risky investment

1.2 Problem Statement

With the gradual advancements in cryptography and other cryptocurrencies, the crypto market became an investment option for many individuals and financial institutions. As the number of users grew the supply and demand grew with them. A cryptocurrencyâs market value is mostly affected by the number of coins in circulation throughout the network. However, finding a trend in the crypto price market is very difficult as the prices fluctuate very often, there fluctuations are termed as market condition (Fang et al.,). In crypto markets, the effects of supply and demand are more exaggerated as there is less liquidity. Experts distinguish three reasons for the heightened volatility of crypto markets. First, thereâs the

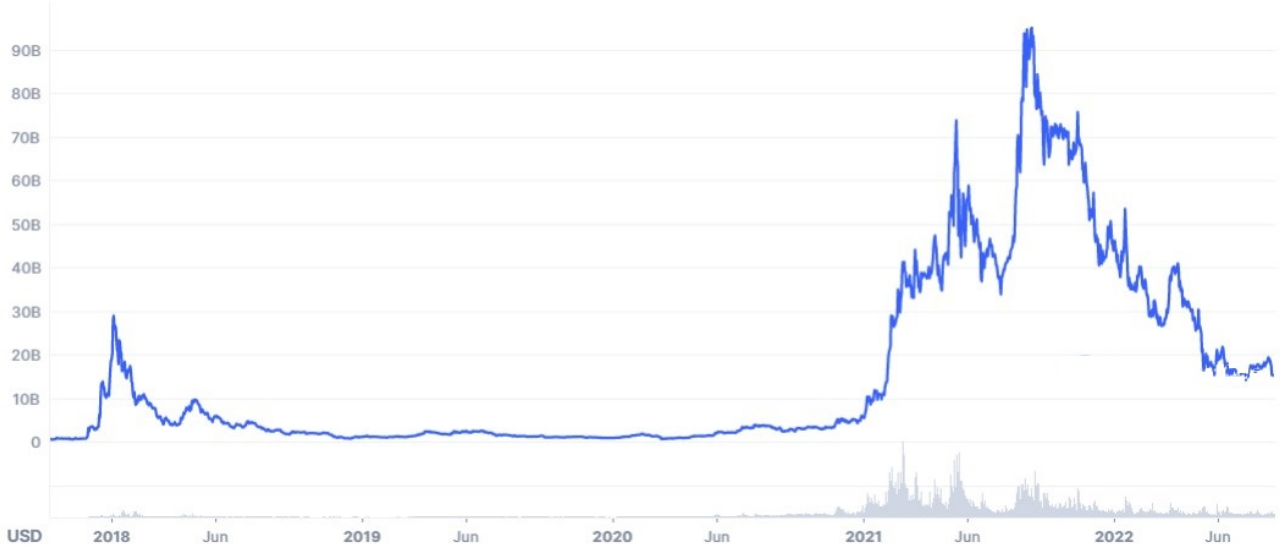

Figure 1.3: Cardano (ADA) market cap chart in billion USD

ever-accelerating news cycles, FUD (that's fear, uncertainty, and doubt) and tweets from crypto influencers that are some of the main reasons for abrupt price changes. A good example is Elon Muskâs tweet from early May of 2021, saying Tesla would no longer accept bitcoin due to energy concerns. Cryptocurrency can also be considered volatile due to its age. For a currency, crypto is still in its startup stage. Because of this there are more and more investors buying and selling every second causing the price fluctuations. Its volatility depends on various other factors such as the number of transactions, mining limits, stock markets, gold prices, social media, and various legal obligations. These fluctuations make the crypto market very unstable and therefore is a risk factor for the investor to look at it as a good investment option. On the other hand, a lot of early investors and traders have already made millions with crypto which, as a result, makes the risk element worth ignoring for some investors and they see the crypto market as an attractive investment opportunity. It is also an important fact that all the data associated with the cryptocurrency is transparent and available to the public. This availability of data and the advancements in Data science and Artificial intelligence have made it possible for data analysts to use machine learning algorithms to predict the prices. In this paper we will be predicting the price of some of the cryptocurrency prices using with the help of machine learning. The cryptocurrency that we will focus of will be Bitcoin (BTC), Ethereum/Ether (ETH) and Cardano/Ada (ADA). The reason for the crypto of choice is that either cryptocurrency represents a generation of the cryptocurrency platform with Bitcoin being the first-generation cryptocurrency and Cardano

as the third generation.

1.3 Organization

This report is divided into five chapters. After **the problem is introduced** in Chapter 1, Chapter 2 will **review** some of the most common methods that have been used in past **literature** to solve the problem. Following that, Chapter 3 will talk about what techniques would be used for **the research** and the process selected to obtain the results. In Chapter 4 the **results** obtained from the research are discuess and finally, Chapter 5 will **conclude** this report by talking the expectations from this project.

CHAPTER 2

Literature Review

The contribution towards the price prediction of cryptocurrency has been immense, but it is still in its early stages. Recent research use a deep learning approach using a deep neural network (DNN), a long short-term memory (LSTM) model, a convolutional neural network and their combination to predict bitcoin price (Ji, Kim, & Im,). In their research they compared the results of deep learning models and found that the LSTM model outperforms the general regression price prediction, however when it comes to price ups and downs classification, the DNN model proves to be the best. In a similar research, McNally et al. (McNally, Roche, & Caton,) used Bayesian optimized recurrent neural network (RNN) and a Long Short Term Memory (LSTM) network to predict Bitcoin price. In their study they concluded that the LSTM model takes more time to train than the RNN model, but the LSTM model achieved a higher classification accuracy of 52% and a RMSE of 8%. As discussed before the cryptocurrency price depends on a lot of different factors, Jay et al. used market data like high and low values, Blockchain data such as transaction per day, hash rate and transaction fee and social media sentiment like google trends and tweet count to predict crypto currency prices (Jay et al.,). Their research used a total of 23 features and to train MLP and LSTM models. According to them MLP and LSTM can effectively capture the non-linear dependencies between the features used. Mittal et al. (Mittal et al.,) also extended the research using Linear regression, polynomial regression, Recurrent Neural Network, and Long Short Term Memory based analysis on tweet volume and Google trends to predict the price of Bitcoin. The accuracy of direction of bitcoin price is predicted with accuracy 77.01%

and 66.66% of polynomial regression with Tweet Volume and Google trends respectively. They concluded that there is a relevant degree of correlation of Google Trends and Tweet volume data with the price of Bitcoin, and with no significant relation with the sentiments of tweets. In addition to this, (Jain, Tripathi, Dwivedi, & Saxena,) also used social media features to forecast price of Bitcoin and Litecoin. In their paper they attempted to predict their future prices using multi-linear regression model. Their method extracts and analyzes tweets that are tagged with the name of cryptocurrencies and explores significant features that are mapped with the concurrent prices of the cryptocurrencies to build prediction curves which can predict the cryptocurrencies prices in the near future. Their multiple linear regression predicts the price of the Bitcoin and Litecoin with R2_score of 44% and 59% respectively.

In a practical analysis on modeling and predicting of the Bitcoin price (Jang & Lee,) used Bayesian neural network (BNN) deals with a number of relevant data features. They used both the traditional determinants of currency markets, such as global macro-economic development and the features endowed from the cryptocurrency making a total 25 explanatory variables as inputs for BNN learning. In another study (Biswas, Pawar, Badole, Galande, & Rathod,) conducted an in-depth analysis of the LSTM process and the GRU to reliably predict the valuation of cryptocurrencies. Their study shows that the proposed GRU and LSTM model is better than the present LSTM model. In the recent years, a study by (Chen, Li, & Sun,) compared more complicated ML approaches, including Logistic regression, quadratic discriminant analysis, XGBoost, RF, SVM, and LSTM, to predict daily bitcoin price. In their paper, the authors used Logistic Regression and Linear Discriminant Analysis for Bitcoin daily price prediction on high-dimensional features with an accuracy of 66% and Random Forest, XGBoost, Quadratic Discriminant Analysis, Support Vector Machine and Long Short-term Memory on Bitcoin 5-minute interval price prediction data which proved to be superior to statistical methods, with accuracy reaching to 67.2%.

In the study of (Pant, Neupane, Poudel, Pokhrel, & Lama,) the sentiment of twitter was used to analyze the price of Bitcoin. The tweets about Bitcoin was classified in the form of positive and negative sentiments. Their RNN model takes into consideration the historical price and the positive and negative sentiment scores and predicts the price of Bitcoin with the accuracy of 77.62% meanwhile the accuracy for sentiment classification of tweets in two class positive

and negative is found to be 81.39%. In terms of a deep learning solution to price prediction, (Spilak,) compared three instants with a Multilayer Perceptron (MLP), a simple Recurrent Neural Network (RNN) and a Long Short-Term Memory (LSTM), to predict cryptocurrency prices. Their study coupled the prediction techniques with supervised learning models to make financial decisions such as trade signals. In their model they used a three-month trading strategy to build a classification model that predicts if the price will go up or down. In their work they also explained the neural network theory and investigate how LSTM networks can outperform, in terms of prediction accuracy, former deep neural network architectures, such as MLP and RNN, on the cryptocurrency market. Another study (Yiying & Yeze,) uses two distinct artificial intelligence frameworks, namely, fully-connected Artificial Neural Network (ANN) and Long-Short-Term-Memory (LSTM) to analyse and predict the price dynamics of Bitcoin, Etherum, and Ripple. This study proves that ANN tends to rely more on long-term history while LSTM tends to rely more on short-term dynamics. According to their study the LSTM models proves to be efficient in utilising useful information hidden in historical memory, however, given enough historical information ANN can achieve a similar accuracy, compared with LSTM.

CHAPTER 3

Methodology

3.1 Data Collection

The data for this study is collected from the historical datasets of Gemini.com and Bitstamp.net, two major resources for historic data on cryptocurrency prices. The data contain the cryptocurrency market exchange and trading features like High, Low, Open, Close, Volume (Crypto), Volume (USD). The timestamp selected for the Date is 24 hours. The raw dataset for BTC consists of all data from 8^{th} of October 2015 to 24^{th} of August 2022, making a total of 60273 observations. The data for ETH consists of all data from 9^{th} of May 2016 to 25^{th} of August 2022, making a total of 55161 observations. Finally the data for ADA consists of all data from 15^{th} of May 2018 to 25^{th} of August 2022, making a total of 37148 observations. All the same datasets contains the same market features. An analysis of the dataset is given in table 3.1 and the description below:

- Date: The data and time each entry collected from crypto market.
- Unix Timestamp: This is the unix timestamp or also known as "Epoch Time". Use this to convert to your local timezone
- Symbol: The symbol of the cyrptocurrency for which the time-series data refers
- Open: This is the opening price of the time period
- High: This is the highest price of the time period

- Low: This is the lowest price of the time period
- Close: This is the closing price of the time period
- Volume (Crypto): This is the volume in the transacted ccy. Ie. For BTC/USD, this is in BTC amount
- Volume USD: This is the volume in the base/converted ccy. Ie. For BTC/USD, this is in USD amount

Variable Name	**Unit**	**Data Type**
Date	dd-mm-yyyy	Date
Open	USD	Number
High	USD	Number
Low	USD	Number
Close	USD	Number
Volume (Crypto)	crypto units	Number
Volume USD	USD	Number

Table 3.1: Description of features

3.2 Data Procesing

Dealing with time series data can sometimes be very problematic because the raw data is very large. Our data is composed of four main components: the trend, the seasonality the cycles that come about and random noise or irregularities. In order to train our models we need to remove noise in data. These noise can be The removal of unwanted data is called preprocessing. In order to remove the irregularities we need to preform data normalization step. This noramlization step will help our data to converge faster. In the raw data there are a number of redundant and repeating data, some data points are outliers and do not represent the correct pattern in the data which results in overfitting. Therefore, it is important to normalize the data and bring all the data features to a common scale so that they are less sensitive to each other. For this study I am using min-max normalization illustrated in

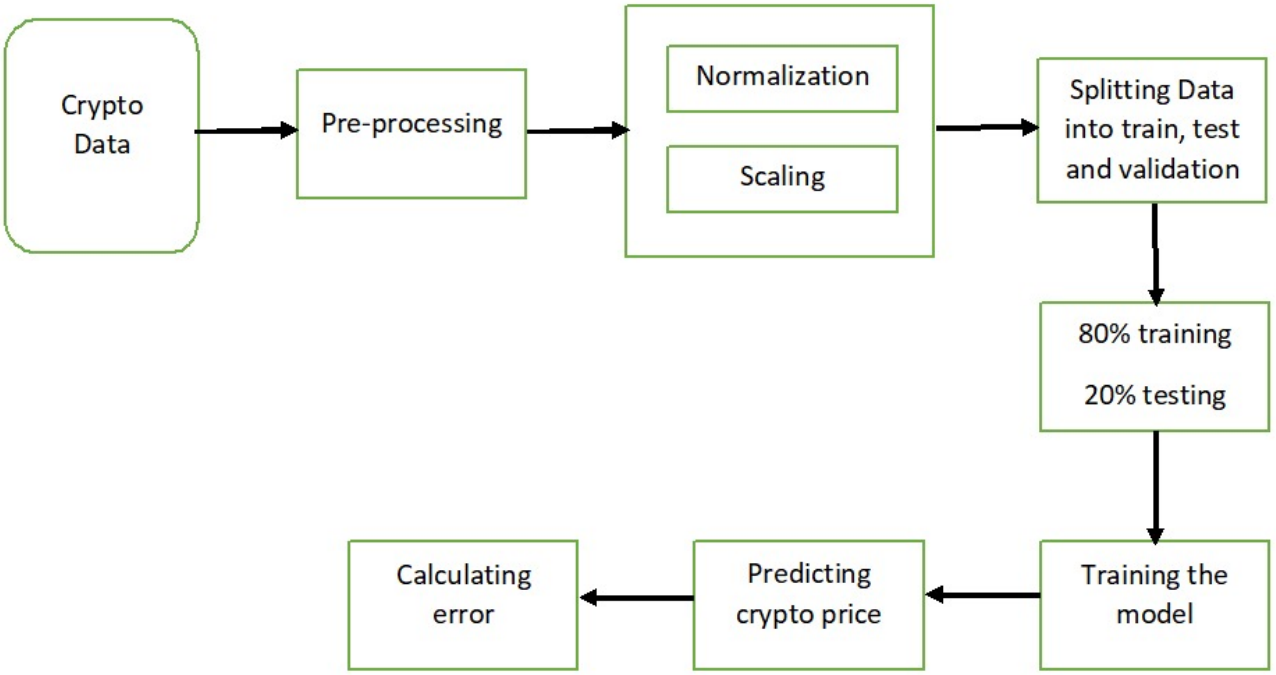

Figure 3.1: Processflow diagram of approach followed

equation 3.2.1.

$$x^{'} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.2.1}$$

where x is an original value, x' is the normalized value.

The next step is to split the data into training and testing sets. For this study we are splitting the data into a 80:20 ratio. In which 80% is for training the model and 20% is for test or validation. Figure 3.1 shows the methodology of this study.

3.3 Prediction models Parameters

I am using keras library in Python to build my LSTM model. The Number of epochs 10 and batch size is 128. The LSTM model consists of an input layer of 120 neurons. For this layer I am using a Leaky Relu activation function and a dropout value of 0.5. Table 3.2 shows the summery for the LSTM model. To stabilizing the learning process and reducing the number of training epochs I am using Batch normalization. Finally a dense layer to get the predictions. The optimizer used in this LSTM model is "Adam" becasue it is a replacement optimization algorithm for stochastic gradient descent for training deep learning models but RMSProp can

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 120)	60960
leaky_re_lu (LeakyReLU)	(None, 120)	0
dropout (Dropout)	(None, 120)	0
batch_normalization (BatchNormalization)	(None, 120)	480
dense (Dense)	(None, 1)	121

Table 3.2: LSTM model summery

also be used to get good results. For the loss function I am using the mean squared error.

Th GRU model the same number of epochs and batch size. The input layer has 50 neurons and the default $tanh$ activation function for the output sequence with a 30% dropout to reduce overfitting to the training data. The output layer is also passed through a rectified linear activation function. Table 3.3 shows the summery for the LSTM model. The optimizer

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 50)	8700
activation (Activation)	(None, 50)	0
dropout_1 (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 1)	51
activation_1 (Activation)	(None, 1)	0

Table 3.3: GRU model summery

used for the GRU is also Adam and mean squared error as loss function.

For the Bi-LSTM the number of epochs are 10 and 128 batch size. The input layer has 128 neurons and the output is pass through a LeakyRelu activation function. The dropout is selected to be 50%. The model uses the AdamOptimizer as its optimization function and the loss function used in this model is mean squared error. The model summary is given in table 3.5.

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 256)	138240
leaky_re_lu_1 (LeakyReLU)	(None, 256)	0
batch_normalization_1 (BatchNormalization)	(None, 256)	1024
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257

Table 3.4: BiLSTM model summery

3.4 Deep Learning Models

This research focus on deep learning techniques to predict the crypto prices. The specific models used are Long short-term memory (LSTM), Gated recurrent units (GRUs) and Bi-Directional LSTM network (BiLSTM).

Long short-term memory (LSTM)

Neural networks are mainly used to learning sequential data. A simple neurat network has an input, hidden and output layer with weight matrices between the layers. In contrast a Recurrent neural network (RNN) has an addition weight matrix that connects hidden state to hidden state at previous time step. The figure 3.2 shows a simple RNN model. The reason for the additional weight matrix is because we want to train every iteration by just the input data, we would also give it the hidden state. That is why we have a recurrent matrix that's connecting the hidden state to itself. The RNN models are good at finding data patterns with the help of input and the previous information, however in practice the modelization is not always accurate. This is because the RNN networks cannot understand the long term dependencies of a data. For example, the context information, that we need to explain the current task, must be find at a very deep time step, rather than the previous one. This problem with the RNNs are referred as the vanishing gradient problem or the exploding gradient. To counter this vanishing gradient problem (Hochreiter & Schmidhuber,) proposed a method of learning to store information over extended time intervals by multiplicative gate units. LSTM model was designed to overcome the long term dependency problem by the help of memory blocks (Graves,). The basic memory block has three NNs interacting with each

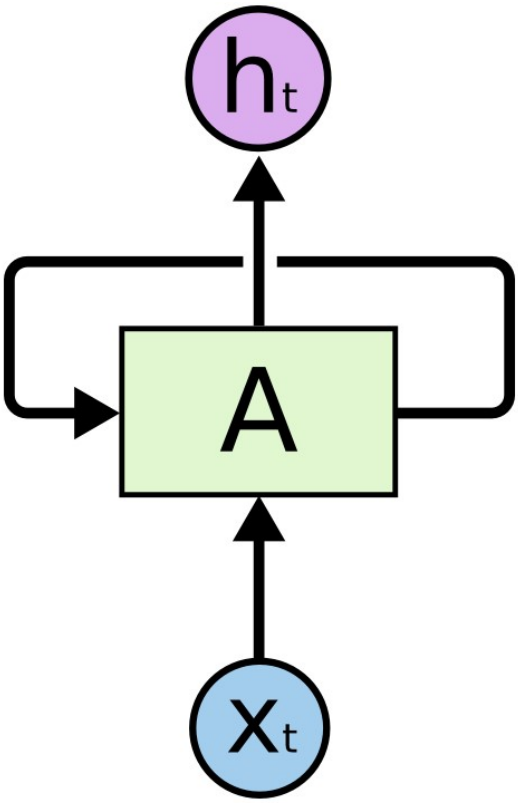

Figure 3.2: Structure of a RNN

other as described below:

1. One is referred as the cell state, which produces linear transformations with constant error flow through the memory block. In order to controll the cell state, gate cells are added to the memory block.

2. A multiplicative input gate unit is introduced to protect the current memory content stored to be perturbed by previous states.

3. Another multiplicative output gate unit is also introduced to protect next units to be perturbed by the currently irrelevant memory content.

In few cases the cell state during learning grow linearly which destroys the purpose of a LSTM network and turn it into an ordinary RNN. To avoid this (Gers, Schmidhuber, & Cummins,) added a forget gate in the memory block. This forget gate with the help of the sigmoid function allows the memory block to reset itself. Figure 3.3 illustrates a simple LSTM memory block used to solve the vanishing gradient problem.

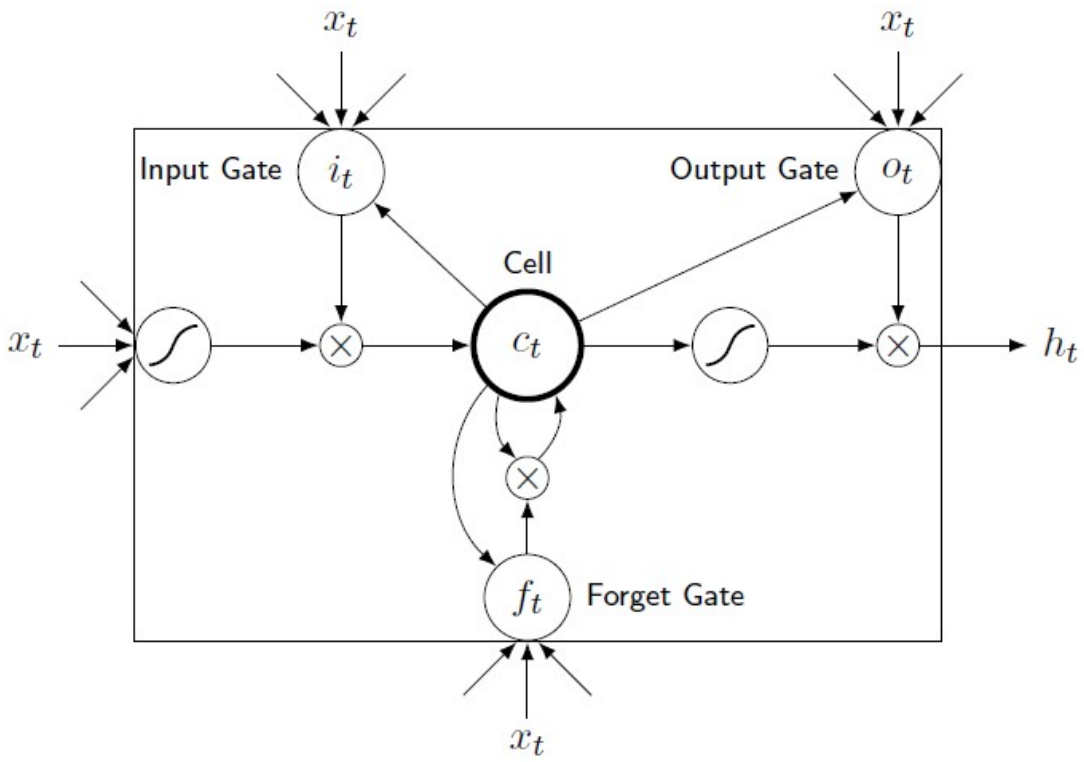

Figure 3.3: A LSTM memory block with input, output and forget gate.

Where x_t is one observation at time t. At every time step t, LSTM decide what information will be removed from the cell state through the decision by a transformation function ($sigmoid$ or $tanh$) in the forget gate layer. The LSTM cell in the figure 3.3 can be described by the following equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.4.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.4.2}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.4.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{3.4.4}$$

$$h_t = o_t \tanh(c_t) \tag{3.4.5}$$

where σ is the logistic $sigmoid$ function, and i, f, o, c are respectively the input gate, forget gate, output gate and memory cell activations. W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{cf}, W_{xc}, W_{hc}, W_{xo}, W_{ho}, W_{co}, which indexes are intuitive, are the input-input gate weight matrix, hidden-input gate weight matrix, etc.

Gated recurrent unit (GRU)

We already know that LSTM is a model to counter the vanishing gradient problem but there is another model that can fix this problem and that model is called GRU. A GRU is also a recurrent neural network similar to LSTM. However, there are a few differences as well. For one GRU has a less complicated structure than LSTM. A GRU two gates, a reset gate and update gate and it lacks the output gate. Unlike LSTM, GRU does not have a cell state, instead it uses hidden states which can combine both long and short term memory. Firgure 3.4 illistrates a simple GRU structure. The hidden layer(s) containing memory cells cover

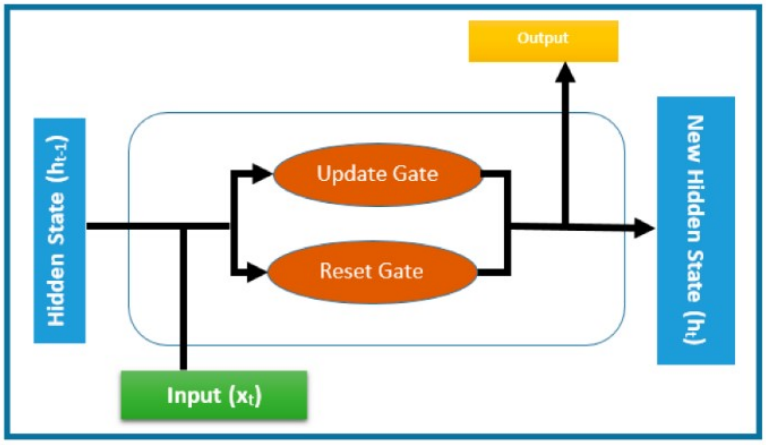

Figure 3.4: Structure of a GRU network.

the main functions of the GRU networks. The reset gate controls the information from the previous timestep to influence the the current information. While the update gate decides whether to ignore the current information or to keep it. In simple terms the function of the update gate is to know how much of the past memory to retain whereas the reset gate is decides how much of the past memory to forget. This makes the gradient propagate reversely and solves the problem with simple RNN. The equations below show how the memory at each hidden state is updated in a GRU network:

1. Update gate z_t is calculated using the equation:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \tag{3.4.6}$$

where, x_t is the input unit which is multiplied with its own weight, $W(z)$. And h_{t-1} is the memory from the previous state $t-1$ which is multiplied with its own weight, $U(z)$.

2. The reset gate r_t is determined by the equation:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \tag{3.4.7}$$

the formula is the same as the update gate where $W(r)$ and $U(r)$ are the weights of the current and previous states respectively.

3. Candidate hidden state h'_t is calculates using the equation:

$$h'_t = \tanh(Wx_t + r_t \odot Uh_t - 1) \tag{3.4.8}$$

where the input x_t is multiplied with a weight W and $h_(t-1)$ with a weight U and an element-wise product is taken between the reset gate r_t and $Uh_(t-1)$ all under a nonlinear activation function $tanh$.

4. The final hidden state h_t is calculated once we have the candidate hidden state using the equation:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{3.4.9}$$

Bidirectional LSTM network (Bi-LSTM)

A BiLSTM model is similar to the LSTM model. However, basic LSTM models use the information from the previous state to influence the output, in BiLSTM, the output at time âtâ is dependent on both, previous and next segments of the sequence not only dependent in a single segment (Ogawa & Hori,). A BiLSTM model is basically two stacked LSTM layers, one with the forward direction and the the other with backward direction. The BiLSTM can consider the past and future information of data simultaneously. The structure of the BiLSTM neural network is illustrated in Figure 3.5.

In a Bi-directional LSTM information not only flows backward to forward but also for-

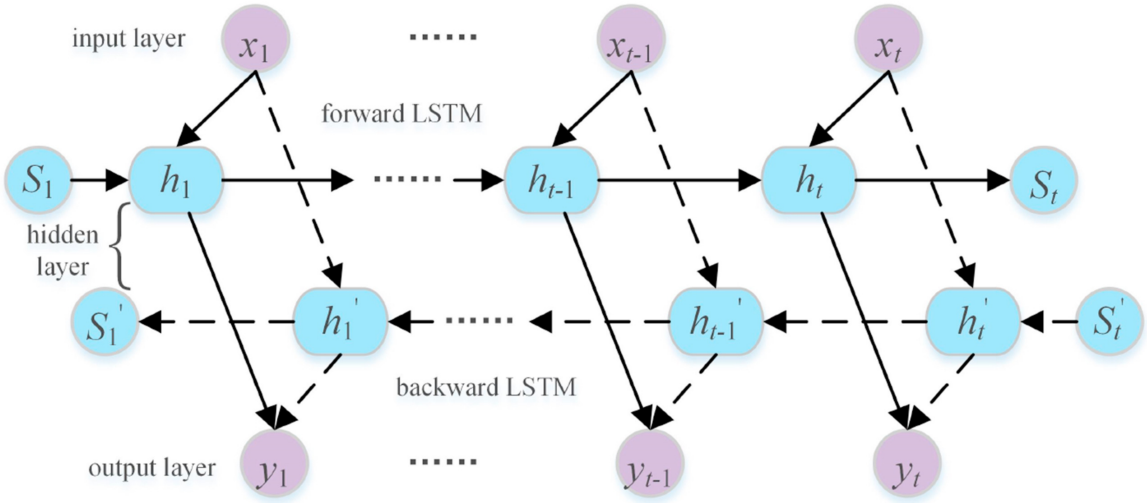

Figure 3.5: Structure of a GRU network.

ward to backward using two hidden states and the two hidden states are connected to get the same output. The hidden layer state H_t of BiLSTM at time t contains forward $\overrightarrow{h_t}$ and backward $\overleftarrow{h_t}$. Below are the equations associated with it:

$$\overrightarrow{h_t} = \overrightarrow{LSTM}(h_{t-1}, x_t, c_{t-1}), t \in [1, T] \tag{3.4.10}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}(h_{t+1}, x_t, c_{t+1}), t \in [T, 1] \tag{3.4.11}$$

$$H_t = [\overrightarrow{h_t}, \overleftarrow{h_t}] \tag{3.4.12}$$

where, T denotes the length of the time series.

3.5 Evaluation

To evaluate the performance of the proposed methods the parameters that I will use are Mean square error (MSE), R2_score, and Mean absolute percentage error (MAPE). These parameters are described below:

1. $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ where N is the number of samples, $\hat{y}_i$ is predicted value and y_i is the real value.

2. $MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}|$ where N is the number of samples, $\hat{y}_i$ is predicted value and y_i is the real value.

3. $R^2 = 1 - RSS/TSS$ where RSS is the residual sum of square and TSS is the total sum of squares.

CHAPTER 4

Discussion and Results

This section provides the results achieved from long short-term memory (LSTM), gated recurrent unit (GRU), and bidirectional LSTM (bi-LSTM) algorithms using three types of popular cryptocurrency: BTC, ETH, and ADA. The comparison between the models are represented in the table 4.1. The results shows that for Bitcoin prediction the GRU model outperforms the LSTM model since the MAPE value of the GRU model is 1.8374% which is significantly less than the 2.6099% of LSTM model and the 11.5204% of Bi-LSTM model. For the R2 score the GRU is still the best with the highest R2 score of 0.9915 as compared to the 0.7815 score of Bi-LSTM. The mean squared error of the GRU is 0.00019 which is the least amongst the other two 0.0003 and 0.005 for the LSTM and Bi-LSTM model respectively. This shows that the GRU model performs the best when it comes to Bitcoin price prediction.

For the prediction of Ethereum (ETH) the LSTM and GRU model performed quite similarly. The MSE for both LSTM and GRU models are the same with 0.000335, however, Bi-LSTM has the highest MSE of 0.0059. Like MSE the R2 score of LSTM and GRU is also same with 0.988 and Bi-LSTM lagging with 0.794 R2 score. The only deciding factor in case of Ethereum is the MAPE which is the least for the LSTM model with 2.837%. For the Ethereum data the LSTM model proved to be the best amongst the three, however, the GRU model is also as efficient.

In case of Cardano (ADA) the results are not as simple as the previous cryptocurrencies. The

mean squared errors of the LSTM and GRU models are very close with 0.000138 and 0.000136 respectively. However, Bi-LSTM has a very high MSE of 0.00197 as compared to the other two models. As similar result is obtained with the R2 scores with 0.975 and 0.976 scores of LSTM and GRU models respectively and a low 0.65 R2 score for the Bi-LSTM model. However, the MAPE is very contradicting to the previous evaluation metrics. The Bi-LSTM model has the lowest MAPE of 2.61% of the other two with LSTM having 3.97% and GRU with 4.6%. According to these results if we consider the MAPE only the Bi-LSTM performs the best, however, if we consider the other two metrics the LSTM and GRU models preformed the better with the GRU model preforming the best.

BITCOIN (BTC)

Model	**MSE**	**R2 score**	**MAPE(%)**
LSTM	0.000304785	0.986915029	2.6099
GRU	0.000197792	0.991508451	1.8374
Bi-LSTM	0.005087385	0.781589509	11.5204

ETHEREUM (ETH)

Model	**MSE**	**R2 score**	**MAPE(%)**
LSTM	0.00033582	0.988390235	2.8372
GRU	0.00033582	0.988390235	3.2461
Bi-LSTM	0.00595131	0.794254627	14.1623

CARDANO (ADA)

Model	**MSE**	**R2 score**	**MAPE(%)**
LSTM	0.0001383	0.975631119	3.9715
GRU	0.000135807	0.976070408	4.6043
Bi-LSTM	0.0019785	0.65137	2.6099

Table 4.1: Model comparison results

In addition to this, if we observe the Figures 4.1, 4.2, and 4.3 the actual and predicted price comparisons are very close to each other for the LSTM and GRU models. The lines

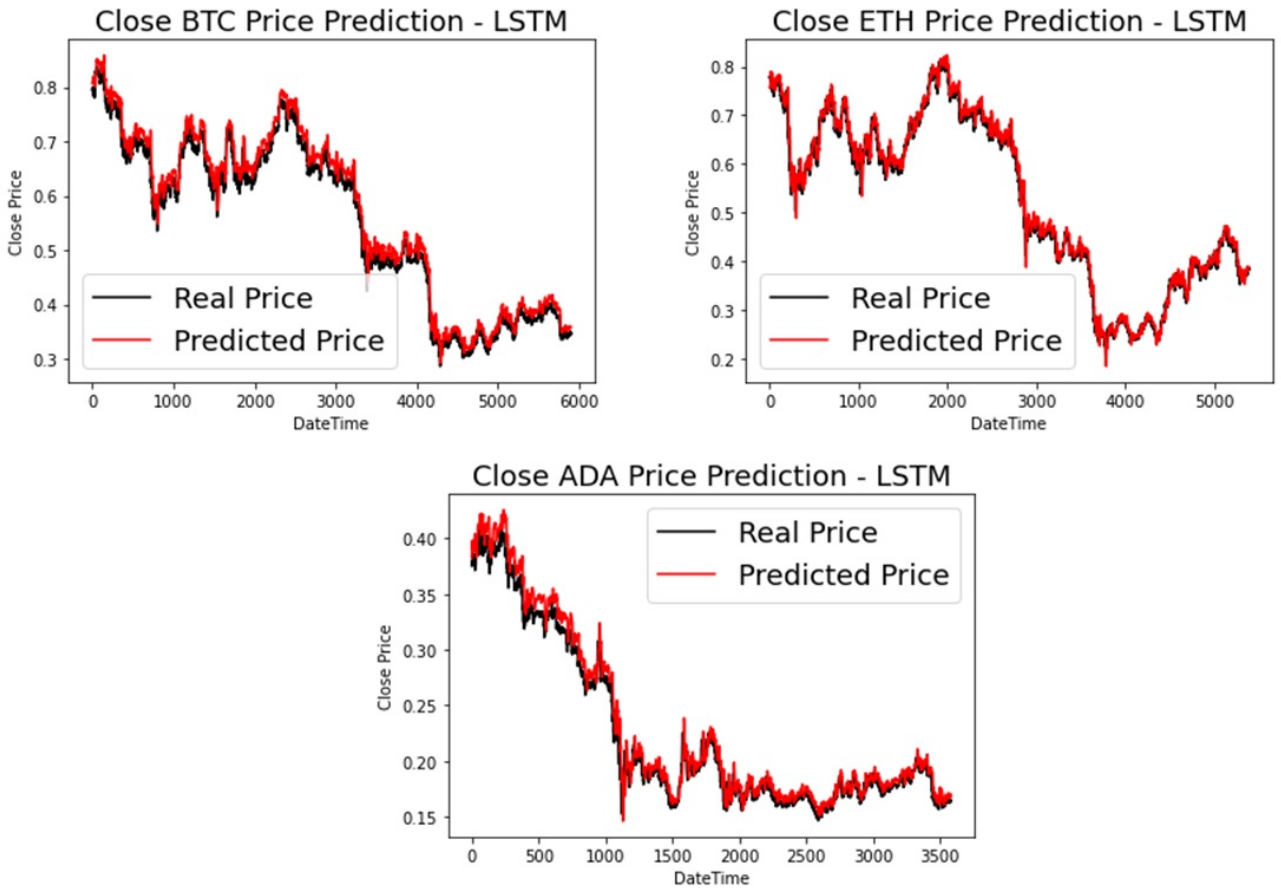

Figure 4.1: Actual and Predicted price comparisons for BTC, ETH and ADA using the LSTM model.

are almost over lapping with one an other in case of all three cryptocurrencies. Simulation results from those models indicate that the forecast result differs from the actual results in a very small magnitude. On the other hand, the Bi-LSTM model forecast's the price of the cryptocurrencies similar to the actual price but the magnitude is larger as compared to the LSTM and GRU models.

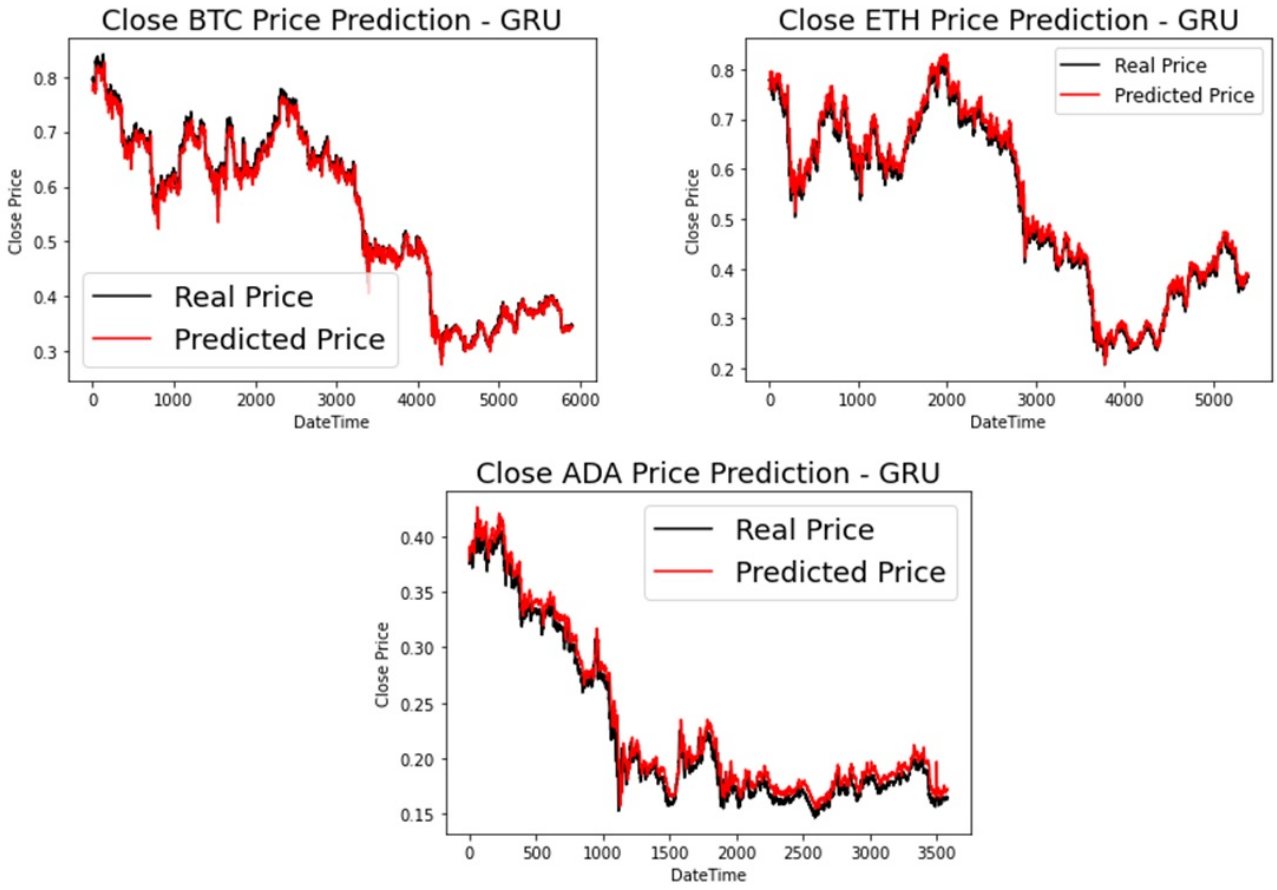

Figure 4.2: Actual and Predicted price comparisons for BTC, ETH and ADA using the GRU model.

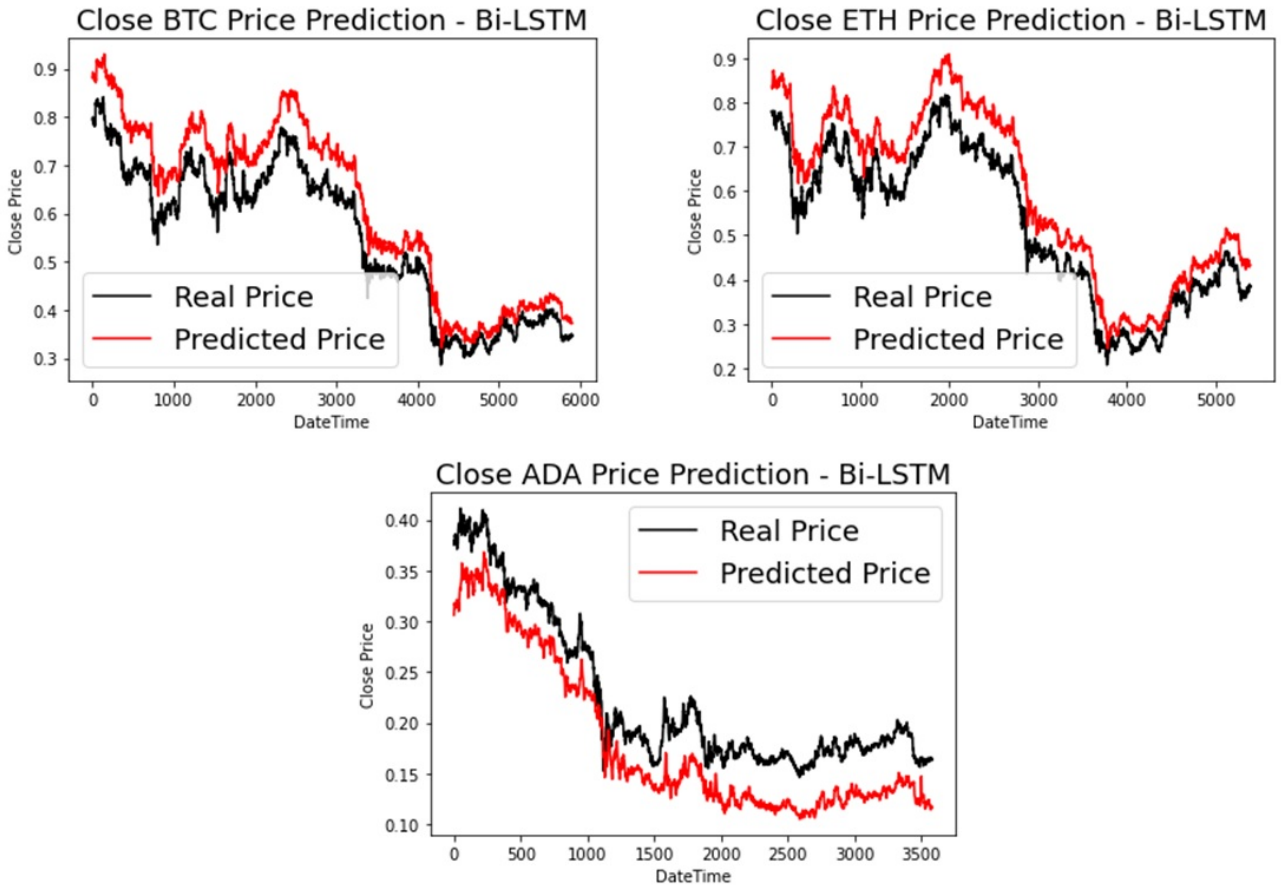

Figure 4.3: Actual and Predicted price comparisons for BTC, ETH and ADA using the Bi-LSTM model.

CHAPTER 5

Conclusion

This paper compares three deep learning models to predict cryptocurrency prices. To compare the models the same inputs were provided to each model and the results were compiled to discover how effectively the models can identify the long and short term patterns within the data. It is known that the market trend of cryptocurrency is very volatile and any prediction is very unlikely to be accurate, however, when dealing with time series data, deep neural networks are very promising. According to the result comparison the GRU and LSTM models performs similar in case of all three cryptocurrencies, however, GRU model preforms the best in most of the case. In this study although, the models does predict accurate results but the predicted prices (red line) are just slightly ahead of the actual prices (black line), which means that the the model is predicting the prices once the actual price is already determined. A better model would be where the red line is leading the black line which would give us a better estimation.

It future work, the models can be further modified with adding more layers to the models and tweaking the model parameters to get different results and further analysing them. Moreover, the data used to train the models can also be modified by concatenating various other features that influence the cryptocurrency market value. Cryptocurrency prices are very unstable at this point in time but in the upcoming years it is believed that the cryptomarket will gradually stabilize and with the advancements in artificial neural networks and machine learning many more data scientists can improve this research.

References

Adams, R., Kewell, B., Parry, G. (2018). Blockchain for good? digital ledger technology and sustainable development goals. In W. Leal Filho, R. W. Marans, J. Callewaert (Eds.), *Handbook of sustainability and social science research* (pp. 127–140). Cham: Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-67122-2_7 doi: 10.1007/978-3-319-67122-2_7

Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V. (2018). Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 acm sigsac conference on computer and communications security* (pp. 913–930).

Bartoletti, M., Pompianu, L. (2017). An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security* (pp. 494–509).

Bentov, I., Gabizon, A., Mizrahi, A. (2016). Cryptocurrencies without proof of work. In *International conference on financial cryptography and data security* (pp. 142–157).

Biswas, S., Pawar, M., Badole, S., Galande, N., Rathod, S. (2021). Cryptocurrency price prediction using neural networks and deep learning. In *2021 7th international conference on advanced computing and communication systems (icaccs)* (Vol. 1, p. 408-413). doi: 10.1109/ICACCS51430.2021.9441872

Buterin, V., et al. (2014). A next-generation smart contract and decentralized application platform ethereum. white paper. *Ethereum Project White Paper*.

Chen, Z., Li, C., Sun, W. (2020). Bitcoin price prediction using machine learning: An approach to sample dimension engineering. *Journal of Computational and Applied Mathematics*, *365*, 112395. doi: https://doi.org/10.1016/j.cam.2019.112395

Fang, F., Ventre, C., Basios, M., Kanthan, L., Martinez-Rego, D., Wu, F., Li, L. (2022). Cryptocurrency trading: a comprehensive survey. *Financial Innovation*, *8*(1), 1–59.

Gers, F. A., Schmidhuber, J., Cummins, F. (2000). Learning to forget: Continual prediction

with lstm. *Neural computation*, *12*(10), 2451–2471.

Graves, A. (2012). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks* (pp. 5–13). Springer.

Hochreiter, S., Schmidhuber, J. (1997, 11). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735

Jain, A., Tripathi, S., Dwivedi, H. D., Saxena, P. (2018). Forecasting price of cryptocurrencies using tweets sentiment analysis. In *2018 eleventh international conference on contemporary computing (ic3)* (p. 1-7). doi: 10.1109/IC3.2018.8530659

Jang, H., Lee, J. (2018). An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information. *IEEE Access*, *6*, 5427-5437. doi: 10.1109/ACCESS.2017.2779181

Jay, P., Kalariya, V., Parmar, P., Tanwar, S., Kumar, N., Alazab, M. (2020). Stochastic neural networks for cryptocurrency price prediction. *IEEE Access*, *8*, 82804-82818. doi: 10.1109/ACCESS.2020.2990659

Ji, S., Kim, J., Im, H. (2019). A comparative study of bitcoin price prediction using deep learning. *Mathematics*, *7*(10). Retrieved from https://www.mdpi.com/2227-7390/7/10/898 doi: 10.3390/math7100898

McNally, S., Roche, J., Caton, S. (2018). Predicting the price of bitcoin using machine learning. In *2018 26th euromicro international conference on parallel, distributed and network-based processing (pdp)* (p. 339-343). doi: 10.1109/PDP2018.2018.00060

Mittal, R., Arora, S., Bhatia, M. (2018). Automated cryptocurrencies prices prediction using machine learning. *Division of Computer Engineering, Netaji Subhas Institute of Technology, India*, *8*, 2229–6956.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.

Ogawa, A., Hori, T. (2017). Error detection and accuracy estimation in automatic speech recognition using deep bidirectional recurrent neural networks. *Speech Communication*, *89*, 70–83.

Pant, D. R., Neupane, P., Poudel, A., Pokhrel, A. K., Lama, B. K. (2018). Recurrent neural network based bitcoin price prediction by twitter sentiment analysis. In *2018 ieee 3rd international conference on computing, communication and security (icccs)* (p. 128-132). doi: 10.1109/CCCS.2018.8586824

Spilak, B. (2018). *Deep neural networks for cryptocurrencies price prediction* (Master's thesis, Humboldt-UniversitÃ¤t zu Berlin, Wirtschaftswissenschaftliche FakultÃ¤t). doi: http://dx.doi.org/10.18452/19249

Yiying, W., Yeze, Z. (2019, March). Cryptocurrency price analysis with artificial intelligence. , 97-101. doi: 10.1109/INFOMAN.2019.8714700

Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# Ignore Warnings
import warnings
from warnings import simplefilter
warnings.filterwarnings("ignore")

# Computational imports
import numpy as np    # Library for n-dimensional arrays
import pandas as pd   # Library for dataframes (structured data)

# Helper imports
import os
import re
import time
import warnings
```

```
from tqdm import tqdm
import datetime as dt
import pandas_datareader as web
from datetime import datetime
import scipy.stats as stats
from pathlib import Path

# ML/DL imports
from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder, LabelEncode
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_probability as tfp
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, RepeatVector, Tim
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Plotting imports
import matplotlib.pyplot as plt
import matplotlib.dates as dates
import seaborn as sns
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode, iplot

get_ipython().run_line_magic('matplotlib', 'inline')
init_notebook_mode(connected=True)

# Set seeds to make the experiment more reproducible.
```

```
from numpy.random import seed
seed(1)

# Allows us to see more information regarding the DataFrame
pd.set_option("display.max_rows", 500)
pd.set_option("display.max_columns", 500)


# # BITCOIN

# In[2]:


url = 'https://www.cryptodatadownload.com/cdd/Gemini_BTCUSD_1h.csv'
dataset = pd.read_csv(url, index_col="date", skiprows=1)
#dataset = pd.read_csv('Gemini_BTCUSD_1h.csv', index_col="date", skiprows=1
dataset.head()


# In[3]:


dataset = dataset.iloc[::-1]

dataset.head()


# In[4]:


df = dataset.reset_index()
df
```

```
# In[5]:


#plt.figure(figsize=(12,6))
#sns.lineplot(x='date', y='close', data=df).set_title("Price of Bitcoin")
df


# In[6]:


df = df[["open", "high", "low", "close","Volume BTC", "Volume USD"]]
#df = df[['Close']]
df.head()
SEQ_LEN = 120
FUTURE_PERIOD = 10

RATIO_TO_PREDICT = "Close"
df


# In[7]:


times = sorted(df.index.values)  # get the times
last_10 = times[-int(0.1*len(times))]
last_20 = times[-int(0.2*len(times))]

test_df = df[(df.index >= last_10)]
validation_df = df[(df.index >= last_20) & (df.index < last_10)]
```

```
train_df = df[(df.index < last_20)]


# In[8]:


train_ = train_df.values
valid_ = validation_df.values
test_ = test_df.values
df


# In[9]:


print("train shape {0}".format(train_.shape))
print("valid shape {0}".format(valid_.shape))
print("test shape {0}".format(test_.shape))


# In[10]:


def line_plot(line1, line2,line3, label1=None, label2=None, label3=None, ti
    fig, ax = plt.subplots(1, figsize=(13, 7))
    ax.plot(line1, label=label1, linewidth=lw)
    ax.plot(line2, label=label2, linewidth=lw)
    ax.plot(line3, label=label3, linewidth=lw)
    ax.set_ylabel('price [USD]', fontsize=14)
    ax.set_title(title, fontsize=16)
    ax.legend(loc='best', fontsize=16);
```

```
# In[11]:


line_plot(train_df['close'], test_df['close'], validation_df['close'], 'tra


# In[12]:


from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_close = MinMaxScaler()


# In[13]:


x = train_[:,3].copy()
scale_close.fit(x.reshape(-1, 1))


# In[14]:


scaler.fit(train_)

train_ = scaler.transform(train_)

valid_ = scaler.transform(valid_)
test_ = scaler.transform(test_)
```

```
# In[15]:


# Now perform exponential moving average smoothing for smooth curve of data
EMA = 0.0
gamma = 0.165
for ti in range(train_.shape[0]):
    EMA = gamma*train_[ti] + (1-gamma)*EMA
    train_[ti] = EMA

# Used for visualization and test purposes
all_mid_data = np.concatenate([train_,valid_, test_],axis=0)


# In[16]:


#data splitting

def split_data(data):
    X = []
    Y = []
    for i in range(SEQ_LEN, len(data)-FUTURE_PERIOD+1):
        X.append(data[i-SEQ_LEN:i])
        Y.append(data[i+(FUTURE_PERIOD-1), 3])
    return np.array(X), np.array(Y)


# In[17]:
```

```
X_train, y_train = split_data(train_)
X_test, y_test = split_data(test_)
X_valid, y_valid = split_data(valid_)


# In[18]:


X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 6))
X_valid = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 6))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 6))


# In[19]:


print("train shape {0}".format(X_train.shape))
print("valid shape {0}".format(X_valid.shape))
print("test shape {0}".format(X_test.shape))


# In[20]:


X_train_2, y_train_2 = split_data(train_)
X_train_2 = np.reshape(X_train_2, (X_train_2.shape[0], X_train_2.shape[1],


# In[21]:


#LSTM model
```

```
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio

# Initialize the RNN
model_LSTM1 = Sequential()
model_LSTM1.add(LSTM(units = 120, input_shape=(120, 6)))
model_LSTM1.add(LeakyReLU(alpha=0.5))
model_LSTM1.add(Dropout(0.5))
model_LSTM1.add(BatchNormalization())
model_LSTM1.add(Dense(1))
model_LSTM1.summary()


# In[22]:


model_LSTM1.compile(optimizer='adam', loss='mean_squared_error', metrics =


# In[23]:


history_LSTM1 = model_LSTM1.fit(X_train, y_train,validation_data=(X_valid,


# In[24]:
```

```
loss = history_LSTM1.history['loss']
val_loss = history_LSTM1.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title("LSTM losses (BTC)")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


# In[25]:


mape = history_LSTM1.history['MAPE']
val_mape = history_LSTM1.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("LSTM MAPE (BTC)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
```

```
plt.legend()

plt.show()


# In[26]:


pred_LSTM1 = model_LSTM1.predict(X_test)


# In[27]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_LSTM1, color = 'red', label = 'Predicted Price')
plt.title('Close BTC Price Prediction - LSTM', fontsize=18)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()


# In[28]:


import math

LSTM1_loss = model_LSTM1.evaluate(X_test, y_test, verbose=1)

#Err_LSTM1 = LSTM1_loss[0]
#Err_LSTM1 = math.sqrt(LSTM1_loss)
```

```
print('\nThe error of the model with 1 layer LSTM is:',LSTM1_loss)


# In[29]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_LSTM1, y_test)
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_LSTM1)
print (MAE,R2)


# In[30]:


#GRU
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,Reshape
from keras.layers import Conv1D, MaxPooling1D, LeakyReLU
from keras.utils import np_utils
from keras.layers import GRU,CuDNNGRU
model_GRU = Sequential()

model_GRU.add(GRU(units=50, input_shape=(120,6),return_sequences=False))
model_GRU.add(Activation('tanh'))
model_GRU.add(Dropout(0.3))
model_GRU.add(Dense(1))
model_GRU.add(Activation('relu'))
model_GRU.compile(loss='mse', optimizer='adam', metrics = ('MAPE'))
model_GRU.summary()
```

```
# In[31]:


history_GRU = model_GRU.fit(X_train, y_train, validation_data=(X_valid, y_v


# In[32]:


loss_GRU = history_GRU.history['loss']
val_loss_GRU = history_GRU.history['val_loss']

epochs = range(len(loss_GRU))

plt.figure()

plt.plot(epochs, loss_GRU, 'b', label='Training loss')
plt.plot(epochs, val_loss_GRU, 'r', label='Validation loss')
plt.title("GRU losses (BTC)")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


# In[33]:


mape = history_GRU.history['MAPE']
val_mape = history_GRU.history['val_MAPE']
```

```
epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("GRU MAPE (BTC)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[ ]:


pred_GRU = model_GRU.predict(X_test)


# In[51]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_GRU, color = 'red', label = 'Predicted Price')
plt.title('Close BTC Price Prediction - GRU', fontsize=18)
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()
```

```
# In[36]:




import math

GRU_loss = model_GRU.evaluate(X_test, y_test, verbose=1)

#Err_GRU = math.sqrt(GRU_loss)
print('\nThe error of the model with a GRU layer is:',GRU_loss)


# In[37]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_GRU, y_test)
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_GRU)
print (MAE,R2)


# In[38]:


#Bi-LSTM
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
```

```
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio
from keras.layers import Bidirectional

num_units = 128
activation_function = 'relu'
optimizer = 'adam'
loss_function = 'mean_squared_error'
batch_size = 128
num_epochs = 10

# Initialize the RNN
regressor2 = Sequential()

# Adding the input layer and the LSTM layer
#regressor2.add(tf.keras.layers.Conv1D(120, 3, activation="relu",input_shap
#regressor2.add(tf.keras.layers.AveragePooling1D(4))
regressor2.add(Bidirectional(LSTM(units = num_units, input_shape=(120, 6)))
regressor2.add(LeakyReLU(alpha= 0.5))
regressor2.add(BatchNormalization())
regressor2.add(Dropout(0.5))
# Adding the output layer
regressor2.add(Dense(units = 1))
# Compiling the RNN
regressor2.compile(optimizer = optimizer, loss = loss_function, metrics = (


# In[39]:


# Using the training set to train the model
history_BiLSTM2 = regressor2.fit(X_train, y_train, validation_data=(X_valid
```

```
# In[40]:


regressor2.summary()


# In[41]:


loss_BiLSTM2 = history_BiLSTM2.history['loss']
val_loss_BiLSTM2 = history_BiLSTM2.history['val_loss']

epochs = range(len(loss_BiLSTM2))

plt.figure()

plt.plot(epochs, loss_BiLSTM2, 'b', label='Training loss')
plt.plot(epochs, val_loss_BiLSTM2, 'r', label='Validation loss')
plt.title("Bi-LSTM losses (BTC)")
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()


# In[42]:


mape = history_BiLSTM2.history['MAPE']
```

```
val_mape = history_BiLSTM2.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("Bi-LSTM MAPE (BTC)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[43]:


pred_BiLSTM2 = regressor2.predict(X_test)


# In[44]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_BiLSTM2, color = 'red', label = 'Predicted Price')
plt.title('Close BTC Price Prediction - Bi-LSTM', fontsize=18)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()
```

```
# In[45]:


import math

BiLSTM2_loss = regressor2.evaluate(X_test, y_test, verbose=1)
print('\nThe error of the model with 1 layer BiLSTM is:',BiLSTM2_loss)


# In[46]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_BiLSTM2, y_test)
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_BiLSTM2)
print (MAE,R2)


# # ethereum


url = 'https://www.cryptodatadownload.com/cdd/Gemini_ETHUSD_1h.csv'
dataset = pd.read_csv(url, index_col="date", skiprows=1)
#dataset = pd.read_csv('Gemini_ETHUSD_1h.csv', index_col="date", skiprows=1
dataset.head()


# In[3]:
```

```
dataset = dataset.iloc[::-1]

dataset.head()


# In[4]:


df = dataset.reset_index()
df


# In[5]:


#plt.figure(figsize=(12,6))
#sns.lineplot(x='date', y='close', data=df).set_title("Price of Bitcoin")
df


# In[6]:


df = df[["open", "high", "low", "close","Volume ETH", "Volume USD"]]
#df = df[['Close']]
df.head()
SEQ_LEN = 120
FUTURE_PERIOD = 10

RATIO_TO_PREDICT = "Close"
df
```

```
# In[7]:


times = sorted(df.index.values)  # get the times
last_10 = times[-int(0.1*len(times))]
last_20 = times[-int(0.2*len(times))]

test_df = df[(df.index >= last_10)]
validation_df = df[(df.index >= last_20) & (df.index < last_10)]
train_df = df[(df.index < last_20)]


# In[8]:


train_ = train_df.values
valid_ = validation_df.values
test_ = test_df.values
df


# In[9]:


print("train shape {0}".format(train_.shape))
print("valid shape {0}".format(valid_.shape))
print("test shape {0}".format(test_.shape))


# In[10]:
```

```
def line_plot(line1, line2,line3, label1=None, label2=None, label3=None, ti
    fig, ax = plt.subplots(1, figsize=(13, 7))
    ax.plot(line1, label=label1, linewidth=lw)
    ax.plot(line2, label=label2, linewidth=lw)
    ax.plot(line3, label=label3, linewidth=lw)
    ax.set_ylabel('Price [USD]', fontsize=14)
    ax.set_xlabel('Hours',)
    ax.set_title(title, fontsize=16)
    ax.legend(loc='best', fontsize=16);


# In[11]:


line_plot(train_df['close'], test_df['close'], validation_df['close'], 'Tra


# In[12]:


from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_close = MinMaxScaler()


# In[13]:


x = train_[:,3].copy()
scale_close.fit(x.reshape(-1, 1))
```

```
# In[14]:


scaler.fit(train_)

train_ = scaler.transform(train_)

valid_ = scaler.transform(valid_)
test_ = scaler.transform(test_)


# In[15]:


# Now perform exponential moving average smoothing for smooth curve of data
EMA = 0.0
gamma = 0.165
for ti in range(train_.shape[0]):
    EMA = gamma*train_[ti] + (1-gamma)*EMA
    train_[ti] = EMA

# Used for visualization and test purposes
all_mid_data = np.concatenate([train_,valid_, test_],axis=0)


# In[16]:


#data splitting
```

```
def split_data(data):
    X = []
    Y = []
    for i in range(SEQ_LEN, len(data)-FUTURE_PERIOD+1):
        X.append(data[i-SEQ_LEN:i])
        Y.append(data[i+(FUTURE_PERIOD-1), 3])
    return np.array(X), np.array(Y)


# In[17]:


X_train, y_train = split_data(train_)
X_test, y_test = split_data(test_)
X_valid, y_valid = split_data(valid_)


# In[18]:


X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 6))
X_valid = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 6))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 6))


# In[19]:


print("train shape {0}".format(X_train.shape))
print("valid shape {0}".format(X_valid.shape))
print("test shape {0}".format(X_test.shape))
```

```
# In[20]:


X_train_2, y_train_2 = split_data(train_)
X_train_2 = np.reshape(X_train_2, (X_train_2.shape[0], X_train_2.shape[1],


# In[21]:


#LSTM model

import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio

# Initialize the RNN
model_LSTM1 = Sequential()
model_LSTM1.add(LSTM(units = 120, input_shape=(120, 6)))
model_LSTM1.add(LeakyReLU(alpha=0.5))
model_LSTM1.add(Dropout(0.5))
model_LSTM1.add(BatchNormalization())
model_LSTM1.add(Dense(1))
model_LSTM1.summary()


# In[22]:
```

```
model_LSTM1.compile(optimizer='adam', loss='mean_squared_error', metrics =


# In[23]:


history_LSTM1 = model_LSTM1.fit(X_train, y_train,validation_data=(X_valid,


# In[24]:


loss = history_LSTM1.history['loss']
val_loss = history_LSTM1.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title("LSTM losses (ETH)")
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()

plt.show()


# In[25]:
```

```
mape = history_LSTM1.history['MAPE']
val_mape = history_LSTM1.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("LSTM MAPE (ETH)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[26]:


pred_LSTM1 = model_LSTM1.predict(X_test)


# In[27]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_LSTM1, color = 'red', label = 'Predicted Price')
plt.title('Close ETH Price Prediction - LSTM', fontsize=18)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
```

```
plt.legend(fontsize=18)
plt.show()


# In[28]:


import math

LSTM1_loss = model_LSTM1.evaluate(X_test, y_test, verbose=1)

#Err_LSTM1 = LSTM1_loss[0]
#Err_LSTM1 = math.sqrt(LSTM1_loss)
print('\nThe error of the model with 1 layer LSTM is:',LSTM1_loss)


# In[29]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_LSTM1, y_test)
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_LSTM1)
print (MAE,R2)


# In[30]:


#GRU
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,Reshape
```

```
from keras.layers import Conv1D, MaxPooling1D, LeakyReLU
from keras.utils import np_utils
from keras.layers import GRU,CuDNNGRU
model_GRU = Sequential()

model_GRU.add(GRU(units=50, input_shape=(120,6),return_sequences=False))
model_GRU.add(Activation('tanh'))
model_GRU.add(Dropout(0.3))
model_GRU.add(Dense(1))
model_GRU.add(Activation('relu'))
model_GRU.compile(loss='mse', optimizer='adam', metrics = ('MAPE'))
model_GRU.summary()


# In[31]:


history_GRU = model_GRU.fit(X_train, y_train, validation_data=(X_valid, y_v


# In[32]:


loss_GRU = history_GRU.history['loss']
val_loss_GRU = history_GRU.history['val_loss']

epochs = range(len(loss_GRU))

plt.figure()

plt.plot(epochs, loss_GRU, 'b', label='Training loss')
plt.plot(epochs, val_loss_GRU, 'r', label='Validation loss')
```

```
plt.title("GRU losses (ETH)")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


# In[33]:


mape = history_GRU.history['MAPE']
val_mape = history_GRU.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("GRU MAPE (ETH)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[34]:


pred_GRU = model_GRU.predict(X_test)
```

```
# In[35]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_GRU, color = 'red', label = 'Predicted Price')
plt.title('Close ETH Price Prediction - GRU', fontsize=18)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=12)
plt.show()


# In[36]:




import math

GRU_loss = model_GRU.evaluate(X_test, y_test, verbose=1)

#Err_GRU = math.sqrt(GRU_loss)
print('\nThe error of the model with a GRU layer is:',GRU_loss)


# In[37]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_LSTM1, y_test)
```

```
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_LSTM1)
print (MAE,R2)


# In[38]:


#Bi-LSTM
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio
from keras.layers import Bidirectional

num_units = 128
activation_function = 'relu'
optimizer = 'adam'
loss_function = 'mean_squared_error'
batch_size = 128
num_epochs = 10

# Initialize the RNN
regressor2 = Sequential()

# Adding the input layer and the LSTM layer
#regressor2.add(tf.keras.layers.Conv1D(120, 3, activation="relu",input_shap
#regressor2.add(tf.keras.layers.AveragePooling1D(4))
regressor2.add(Bidirectional(LSTM(units = num_units, input_shape=(120, 6)))
regressor2.add(LeakyReLU(alpha= 0.5))
regressor2.add(BatchNormalization())
```

```
regressor2.add(Dropout(0.5))
# Adding the output layer
regressor2.add(Dense(units = 1))
# Compiling the RNN
regressor2.compile(optimizer = optimizer, loss = loss_function, metrics = (


# In[39]:


# Using the training set to train the model
history_BiLSTM2 = regressor2.fit(X_train, y_train, validation_data=(X_valid


# In[40]:


regressor2.summary()


# In[41]:


loss_BiLSTM2 = history_BiLSTM2.history['loss']
val_loss_BiLSTM2 = history_BiLSTM2.history['val_loss']

epochs = range(len(loss_BiLSTM2))

plt.figure()

plt.plot(epochs, loss_BiLSTM2, 'b', label='Training loss')
plt.plot(epochs, val_loss_BiLSTM2, 'r', label='Validation loss')
```

```
plt.title("Bi-LSTM losses (ETH)")
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()


# In[42]:


mape = history_BiLSTM2.history['MAPE']
val_mape = history_BiLSTM2.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("Bi-LSTM MAPE (ETH)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[43]:


pred_BiLSTM2 = regressor2.predict(X_test)
```

```
# In[44]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_BiLSTM2, color = 'red', label = 'Predicted Price')
plt.title('Close ETH Price Prediction - Bi-LSTM', fontsize=18)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()


# In[45]:


import math

BiLSTM2_loss = regressor2.evaluate(X_test, y_test, verbose=1)
print('\nThe error of the model with 1 layer BiLSTM is:',BiLSTM2_loss)


# In[46]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_BiLSTM2, y_test)
from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_BiLSTM2)
print (MAE,R2)
```

```
# # Cardano


url = 'https://www.cryptodatadownload.com/cdd/Exmo_ADAUSD_1h.csv'
dataset = pd.read_csv(url, index_col="date", skiprows=1)
#dataset = pd.read_csv('Gemini_BTCUSD_1h.csv', index_col="date", skiprows=1
dataset.head()


# In[3]:


dataset = dataset.iloc[::-1]

dataset.head()


# In[4]:


df = dataset.reset_index()
df


# In[5]:


#plt.figure(figsize=(10,5))
#sns.lineplot(x='date', y='close', data=df).set_title("Price of ADA")
```

```
# In[6]:


df = df[["open", "high", "low", "close","Volume ADA", "Volume USD"]]
#df = df[['Close']]
df.head()
SEQ_LEN = 120
FUTURE_PERIOD = 10

RATIO_TO_PREDICT = "Close"
df


# In[7]:


times = sorted(df.index.values)  # get the times
last_10 = times[-int(0.1*len(times))]
last_20 = times[-int(0.2*len(times))]

test_df = df[(df.index >= last_10)]
validation_df = df[(df.index >= last_20) & (df.index < last_10)]
train_df = df[(df.index < last_20)]


# In[8]:


train_ = train_df.values
valid_ = validation_df.values
```

```
test_ = test_df.values
valid_.shape


# In[9]:


print("train shape {0}".format(train_.shape))
print("valid shape {0}".format(valid_.shape))
print("test shape {0}".format(test_.shape))


# In[10]:


def line_plot(line1, line2,line3, label1=None, label2=None, label3=None, ti
    fig, ax = plt.subplots(1, figsize=(13, 7))
    ax.plot(line1, label=label1, linewidth=lw)
    ax.plot(line2, label=label2, linewidth=lw)
    ax.plot(line3, label=label3, linewidth=lw)
    ax.set_ylabel('Price [USD]', fontsize=14)
    ax.set_xlabel('Hours',)
    ax.set_title(title, fontsize=16)
    ax.legend(loc='best', fontsize=16);


# In[11]:


line_plot(train_df['close'], test_df['close'], validation_df['close'], 'Tra
```

```
# In[12]:


from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_close = MinMaxScaler()


# In[13]:


x = train_[:,3].copy()
scale_close.fit(x.reshape(-1, 1))


# In[14]:


scaler.fit(train_)

train_ = scaler.transform(train_)

valid_ = scaler.transform(valid_)
test_ = scaler.transform(test_)


# In[15]:


# Now perform exponential moving average smoothing for smooth curve of data
EMA = 0.0
gamma = 0.165
```

```
for ti in range(train_.shape[0]):
    EMA = gamma*train_[ti] + (1-gamma)*EMA
    train_[ti] = EMA

# Used for visualization and test purposes
all_mid_data = np.concatenate([train_,valid_, test_],axis=0)


# In[16]:


#data splitting

def split_data(data):
    X = []
    Y = []
    for i in range(SEQ_LEN, len(data)-FUTURE_PERIOD+1):
        X.append(data[i-SEQ_LEN:i])
        Y.append(data[i+(FUTURE_PERIOD-1), 3])
    return np.array(X), np.array(Y)


# In[17]:


X_train, y_train = split_data(train_)
X_test, y_test = split_data(test_)
X_valid, y_valid = split_data(valid_)


# In[18]:
```

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 6))
X_valid = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 6))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 6))


# In[19]:


print("train shape {0}".format(X_train.shape))
print("valid shape {0}".format(X_valid.shape))
print("test shape {0}".format(X_test.shape))


# In[20]:


X_train_2, y_train_2 = split_data(train_)
X_train_2 = np.reshape(X_train_2, (X_train_2.shape[0], X_train_2.shape[1],


# In[21]:


#LSTM model

import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio
```

```
# Initialize the RNN
model_LSTM1 = Sequential()
model_LSTM1.add(LSTM(units = 120, input_shape=(120, 6)))
model_LSTM1.add(LeakyReLU(alpha=0.5))
model_LSTM1.add(Dropout(0.5))
model_LSTM1.add(BatchNormalization())
model_LSTM1.add(Dense(1))
model_LSTM1.summary()


# In[22]:


model_LSTM1.compile(optimizer='adam', loss='mean_squared_error', metrics =


# In[23]:


history_LSTM1 = model_LSTM1.fit(X_train, y_train,validation_data=(X_valid,


# In[24]:


loss = history_LSTM1.history['loss']
val_loss = history_LSTM1.history['val_loss']

epochs = range(len(loss))

plt.figure()
```

```
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title("LSTM losses (ADA)")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


# In[25]:


mape = history_LSTM1.history['MAPE']
val_mape = history_LSTM1.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("LSTM MAPE (ADA)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[26]:
```

```
pred_LSTM1 = model_LSTM1.predict(X_test)


# In[27]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_LSTM1, color = 'red', label = 'Predicted Price')
plt.title('Close ADA Price Prediction - LSTM', fontsize=18)
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()


# In[28]:


import math

LSTM1_loss = model_LSTM1.evaluate(X_test, y_test, verbose=1)

#Err_LSTM1 = LSTM1_loss[0]
#Err_LSTM1 = math.sqrt(LSTM1_loss)
print('\nThe error of the model with 1 layer LSTM is:',LSTM1_loss)


# In[29]:
```

```
from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_LSTM1, y_test)
MAE


# In[30]:


from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_LSTM1)
R2


# In[31]:


#GRU
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,Reshape
from keras.layers import Conv1D, MaxPooling1D, LeakyReLU
from keras.utils import np_utils
from keras.layers import GRU,CuDNNGRU
model_GRU = Sequential()

model_GRU.add(GRU(units=50, input_shape=(120,6),return_sequences=False))
model_GRU.add(Activation('tanh'))
model_GRU.add(Dropout(0.3))
model_GRU.add(Dense(1))
model_GRU.add(Activation('relu'))
model_GRU.compile(loss='mse', optimizer='adam', metrics = ('MAPE'))
model_GRU.summary()
```

```
# In[32]:


history_GRU = model_GRU.fit(X_train, y_train, validation_data=(X_valid, y_v


# In[33]:


loss_GRU = history_GRU.history['loss']
val_loss_GRU = history_GRU.history['val_loss']

epochs = range(len(loss_GRU))

plt.figure()

plt.plot(epochs, loss_GRU, 'b', label='Training loss')
plt.plot(epochs, val_loss_GRU, 'r', label='Validation loss')
plt.title("GRU losses (ADA)")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()


# In[34]:


mape = history_GRU.history['MAPE']
val_mape = history_GRU.history['val_MAPE']
```

```
epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("GRU MAPE (ADA)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[35]:


pred_GRU = model_GRU.predict(X_test)


# In[36]:


plt.plot(y_test, color = 'black', label = 'Real Price')
plt.plot(pred_GRU, color = 'red', label = 'Predicted Price')
plt.title('Close ADA Price Prediction - GRU', fontsize=18)
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()
```

```
# In[37]:




import math

GRU_loss = model_GRU.evaluate(X_test, y_test, verbose=1)

#Err_GRU = math.sqrt(GRU_loss)
print('\nThe error of the model with a GRU layer is:',GRU_loss)


# In[38]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_GRU, y_test)
MAE


# In[39]:


from sklearn.metrics import r2_score
R2=r2_score(y_test, pred_GRU)
R2


# In[40]:
```

```
#Bi-LSTM
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout, BatchNormalizatio
from keras.layers import Bidirectional

num_units = 128
activation_function = 'relu'
optimizer = 'adam'
loss_function = 'mean_squared_error'
batch_size = 128
num_epochs = 10

# Initialize the RNN
regressor2 = Sequential()

# Adding the input layer and the LSTM layer
#regressor2.add(tf.keras.layers.Conv1D(120, 3, activation="relu",input_shap
#regressor2.add(tf.keras.layers.AveragePooling1D(4))
regressor2.add(Bidirectional(LSTM(units = num_units, input_shape=(120, 6)))
regressor2.add(LeakyReLU(alpha= 0.5))
regressor2.add(BatchNormalization())
regressor2.add(Dropout(0.5))
# Adding the output layer
regressor2.add(Dense(units = 1))
# Compiling the RNN
regressor2.compile(optimizer = optimizer, loss = loss_function, metrics = (
```

```
# In[41]:


# Using the training set to train the model
history_BiLSTM2 = regressor2.fit(X_train, y_train, validation_data=(X_valid


# In[42]:


regressor2.summary()


# In[43]:


loss_BiLSTM2 = history_BiLSTM2.history['loss']
val_loss_BiLSTM2 = history_BiLSTM2.history['val_loss']

epochs = range(len(loss_BiLSTM2))

plt.figure()

plt.plot(epochs, loss_BiLSTM2, 'b', label='Training loss')
plt.plot(epochs, val_loss_BiLSTM2, 'r', label='Validation loss')
plt.title("Bi-LSTM losses (ADA)")
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()
```

```
# In[44]:


mape = history_BiLSTM2.history['MAPE']
val_mape = history_BiLSTM2.history['val_MAPE']

epochs = range(len(mape))

plt.figure()

plt.plot(epochs, mape, 'b', label='Training MAPE')
plt.plot(epochs, val_mape, 'r', label='Validation MAPE')
plt.title("Bi-LSTM MAPE (ADA)")
plt.xlabel('Epochs')
plt.ylabel('MAPE')
plt.legend()

plt.show()


# In[45]:


pred_BiLSTM2 = regressor2.predict(X_test)


# In[50]:


plt.plot(y_test, color = 'black', label = 'Real Price')
```

```
plt.plot(pred_BiLSTM2, color = 'red', label = 'Predicted Price')
plt.title('Close ADA Price Prediction - Bi-LSTM', fontsize=18)
#plt.xticks(range(0,df.shape[0],50),df['Date'].loc[::50],rotation=45)
plt.xlabel('DateTime')
plt.ylabel('Close Price')
plt.legend(fontsize=18)
plt.show()


# In[47]:


import math

BiLSTM2_loss = regressor2.evaluate(X_test, y_test, verbose=1)

#Err_BiLSTM2 = math.sqrt(BiLSTM2_loss)
print('\nThe error of the model with 1 layer BiLSTM is:',BiLSTM2_loss)


# In[48]:


from sklearn.metrics import mean_squared_error
MAE=mean_squared_error(pred_BiLSTM2, y_test)
MAE


# In[49]:


from sklearn.metrics import r2_score
```

```
R2=r2_score(y_test, pred_BiLSTM2)
R2
```