
الجامعة المصرية للتعلم الإلكتروني الأهلية

Egyptian E-Learning University

Faculty of Computers & Information

# Prediction of cryptocurrency prices



| | |
|---|---|
| **Tasabeh Ahmed Sayed** | **2000980** |
| **Hadeer Abdesallahen Shawky** | **2001606** |
| **Doha Mohamed Abd Elrahman** | **2000533** |
| **Ashraf Hussein Sayed** | **2000767** |
| **Ahmed Hesham Bakr** | **2001174** |
| **Marco Luka Michail** | **2001290** |
| **Abram Nasser Misca** | **2000787** |

**Under supervision**

## DR/ Basem Mohamed

Professor at the Egyptian E-learning university

## Eng/Abdelrahman Younis

Teaching assistant at the Egyptian E-learning university


Bachelor of Science in Computer & Information Technology

# Assiut 2023-2024

## Acknowledgement

First and for most, praises and thanks to the Allah, the Almighty, for his blessing throughout our years in the college and our graduation project to complete this stage of our life successfully.

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

Thanks to Egyptian E-Learning University especially Assiut center for helping us to reach this level of awareness.

Great thanks to **Prof. Dr. Hisham Abdelsalam** head of the Egyptian University E-Learning, and Prof. **Dr. hisham hassan**, Dean of the Faculty of information technology university Egyptian E-Learning, Dr. Kamal Hamza, Program Director of information technology university Egyptian E-Learning.

We would like to express our deep and sincere gratitude to our project supervisor **DR/ Basem Mohamed** for giving us the opportunity to lead us and providing invaluable guidance throughout this project. It was a great privilege and honor to work and study under his guidance. we are extremely grateful for what he has offered us.

We are especially indebted to say many thanks to **Eng/Abdelrahman Younis** for guidance and constant supervision as well as for his patience, friendship, empathy, and great sense of humor. His dynamism, vision, sincerity and motivation have deeply inspired us.

we would also like to extend my sincere thanks to our families and friends for their unwavering love and support. Their encouragement and belief in us have been a constant source of motivation, and we could not have accomplished this without them.

Once again, thank for dedication and commitment to our academic success, and for helping us achieve our goals. We will always be grateful for the role you played in shaping our future.

Finally, we would like to express our gratitude for everyone who helped us during the graduation project.

# Contents

# List of Acronyms or Abbreviations

BTC: Bitcoin.

LTC: Litecoin.

ETH: Ethereum.

XMR: Monero xmr.

ARIMA: AutoRegressive Integrated Moving Average.

SVR: Support Vector Regression.

RBF: Radial Basis Function.

LSTM: Long Short-Term Memory.

GRU: Gated Recurrent Unit.

MAE: Mean Absolute Error.

MSE: Mean Squared Error.

MAPE: Mean Absolute Percentage Error.

RMSE: Root Mean Squared Error.

ML: Machine Learning.

DL: Deep Learning.

VADER: Valence Aware Dictionary and sEntiment Reasoner.

LDA: Linear Discriminant Analysis

# ABSTRACT

Cryptocurrency has recently attracted substantial interest from investors due to its underlying philosophy of decentralization and transparency. Considering cryptocurrency's volatility and unique characteristics, accurate price prediction is essential for developing successful investment strategies. To this end, we decided to make cryptocurrency prediction prices using deep and machine learning models. Cryptocurrencies are becoming increasingly relevant in the financial world and can be considered as an emerging market. The low barrier of entry and high data availability of the cryptocurrency market makes it an excellent subject of study, from which it is possible to derive insights into the behavior of markets through the application of sentiment analysis and machine learning techniques for the challenging task of stock market prediction. While there have been some previous studies, most of them have focused exclusively on the behavior of Bitcoin. In this paper, we propose the usage of common machine learning tools and available social media data for predicting the price movement of the Bitcoin, Ethereum, Monero and Litecoin cryptocurrency market movements. We compare the utilization of Autoregressive integrated moving average (ARIMA), Support vector machine(SVR), Long short term memory (LSTM) and Gated recurrent unit(GRU) using elements from Twitter and market data as input features. The results show that it is possible to predict cryptocurrency markets using machine learning and sentiment analysis, where Twitter data by itself could be used to predict certain cryptocurrencies and that NN outperform the other models. Also The growing popularity of E-commerce, social medias, forums, blogs etc. created a new platform where anyone can discuss and exchange his/her views, ideas , suggestions and experience about any product or services. This trend accumulated a huge amount of user generated data on the web. If this content can be extracted and analyzed properly then it can act as a key factor in decision making. Twitter is one such a platform widely used by people to express their opinions and display sentiments on different occasions. But manual extraction and analysis of this content is an impossible task, as the content is unstructured in nature and it is written in natural language. This situation opened a new area of research called Opinion Mining and Sentiment Analysis. Opinion Mining and Sentiment Analysis is an extension of Data Mining that extracts and analyzes the unstructured data automatically.

# Chapter 1

# Introduction

## 1.1 Introduction

Cryptocurrency, a digital or virtual form of currency secured by cryptography, has revolutionized the financial landscape since the advent of Bitcoin in 2009. Built on blockchain technology, cryptocurrencies offer decentralized and immutable transactions, free from the control of central authorities like governments or banks. Over the years, the cryptocurrency market has expanded exponentially, with thousands of different cryptocurrencies catering to various use cases and industries. This proliferation has sparked immense interest among investors, technologists, and policymakers alike, reshaping how we perceive and interact with money and digital assets in the digital age.

cryptocurrency market is characterized by its inherent volatility, driven by factors such as market sentiment, technological advancements, regulatory developments, and macroeconomic trends. Prices of cryptocurrencies can experience rapid fluctuations, presenting both opportunities for profit and risks of significant losses for investors. While some view cryptocurrencies as a speculative asset class, others embrace them as a means of financial inclusion, decentralization, and innovation. Additionally, the emergence of decentralized finance (DeFi) platforms has further expanded the utility and adoption of cryptocurrencies, enabling activities such as lending, borrowing, and trading without intermediaries. Despite the transformative potential of cryptocurrencies, challenges such as scalability, regulatory uncertainty, security vulnerabilities, and environmental concerns continue to shape the evolution of this nascent industry. As the cryptocurrency ecosystem matures, navigating its complexities requires a nuanced understanding of its technological underpinnings, market dynamics, and regulatory landscape.

## 1.2 History of cryptocurrency

### Inception of an idea

The history of cryptocurrency began long before the first full-fledged cryptocurrency, Bitcoin. In the 1980s and 1990s, scientists and cryptographers worked to create digital currencies that were fully encrypted and secure for use on the internet. One of the first projects that preceded the creation of modern cryptocurrencies was DigiCash, founded by David Chaum in 1989. Although DigiCash was not a huge success, it laid the groundwork for future developments in digital money.

## 1.3 Cryptocurrency Creation

In 1998, computer engineer Wei Dai published a description of “b-money”, an anonymous distributed electronic money system. Shortly thereafter, Nick Szabo created BitGold. Like

bitcoin and other cryptocurrencies created after it, BitGold is a system of electronic currencies in which users are required to perform the function of proving work and posting encrypted solutions.

The first decentralized cryptocurrency, bitcoin, was created in 2009 by a developer under the pseudonym Satoshi Nakamoto. It uses the SHA-256 cryptographic hash function. As of 2023, there are about 19 million bitcoins in circulation. In April 2011, the cryptocurrency Namecoin was created as an attempt to form a decentralized domain name system to make it harder to censor the Internet. Shortly after, in October 2011, the Litecoin cryptocurrency was released. It was the first successful cryptocurrency to use a script as a hash function instead of SHA-256. Another significant cryptocurrency is Peercoin. It was the first to use a hybrid proof-of-work and proof-of-assignment function. Many cryptocurrencies have been created, but few of them have been successful because they did not offer technical innovation.

After 2014, so-called “second generation” cryptocurrencies such as Monero, Ethereum, Dash and NXT emerged. They have advanced features such as address masking, smart contracts, sidechains or assets.

Central bankers argue that the adoption of cryptocurrencies such as bitcoin significantly limits their ability to influence the cost of credit in the economy. They also argue that the more popular cryptocurrency trading becomes, the more consumer confidence in fiat money will erode. According to Gareth Murphy, an official at a central bank, “the widespread use of cryptocurrencies will make it much more difficult for statistical agencies to gather the information on economic activity that governments need to manage the economy.” In his view, virtual currencies represent a new challenge to the important monetary and exchange rate policy functions of central banks.

Jordan Kelly, founder of Robocoin, launched the first bitcoin ATM in the US on 20 February 2014. The device, installed in Austin, Texas, features a document scanner to verify users’ identities. Dogecoin Foundation, a charitable organisation associated with the Dogecoin currency, donated the equivalent of more than US$30,000 in cryptocurrency to support the Jamaican bobsleigh team’s participation in the 2014 Olympic Games in Sochi, Russia.

Since the mid-2010s, a number of countries around the world have begun to recognise cryptocurrencies as legal means of payment on their territories.

## 1.4Cryptocurrencies

# What is Bitcoin?

Bitcoin(BTC) is the world's first successful decentralized cryptocurrency and payment system. By the end of January 2023, Bitcoin's total market capitalization had reached nearly US $ 450 billion, accounting for approximately 42% of the market share. Additionally, the Bitcoin futures contract (BTC) was successfully launched by the Chicago Mercantile Exchange (CME) on December 18, 2017, making a new era for Bitcoin trading. Numerous studies have highlighted the benefits of using Bitcoin, including improving investors' risk–return profile, offering potential safe-haven characteristics, and enabling decentralized payment and transaction facilitation. Bitcoin is design is public, nobody owns or controls Bitcoin and every one can take part.

Through many of its unique properties, Bitcoin allows exciting uses that could not be covered by any previous payment system.

It uses peer-to-peer technology to operate without the need for a central authority behind it. Bitcoin transactions are registered on open-source software. Bitcoin uses blockchain technology to ensure transactions are secure and censorship-resistance.

A blockchain is a distributed ledger, or a shared database that, in BTC's case, anyone can access to verify transactions.

While anyone can access these transactions, Bitcoin works through pseudonymous addresses. This means that while anyone can see the transaction occurred – meaning address A sent BTC to address B – often only the sender and receiver know who's behind each address.

Blockchains are essentially built through blocks of data chained together – forming a chain of blocks – with each new block building on the previous one. Transactions are verified by validators, which on the Bitcoin network are called miners.

These use specialized hardware to "mine" blocks and add them to the blockchain by solving complex mathematical problems.

Miners are rewarded through a set BTC reward included in each block, called the coinbase reward, and with the transaction fees attached to the transactions included in the blocks they mine.

Data stored in blocks is encrypted through Bitcoin's SHA-256 hashing algorithm.

Bitcoin's supply is limited to 21 million coins, and each block is added to the network every 10 minutes. The timing of each block is kept stable through a difficulty adjustment mechanism,

while BTC's inflation is controlled by code, with the reward in each block halving every 210,000, or roughly every four years.

Each Bitcoin is divisible to eight decimal places, with the smallest unit being known as a satoshi – one satoshi is 0.00000001 BTC. The cryptocurrency could be made divisible into even more decimal places in the future.

## Who Created Bitcoin?

Bitcoin was created by Satoshi Nakamoto, a pseudonymous entity who built upon previous work to outline the technology behind the cryptocurrency in a 2008 white paper titled: "Bitcoin: A Peer-to-Peer Electronic Cash System."

It's known that Nakamoto registered the Bitcoin.org domain in August 2008, before announcing the whitepaper to a Cryptography Mailing List in October of that year.

Bitcoin's first block – the genesis block – was mined on January 3, 2009. Nakamoto added to it the text: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks," as a reference to the 2008 financial crisis and central banks' response to it.

The first Bitcoin transaction was made on January 12, involving Nakamoto and Hal Finney, a cypherpunk that worked with the PGP Corporation developing a leading encryption product.

## How Do You Use Bitcoin?

Bitcoin was initially designed as a peer-to-peer payment method. As interest around it grew and its value increased, its use cases grew as well. Because of Bitcoin's open-source approach, competition from other cryptocurrencies grew as well.

To use Bitcoin, a wallet is necessary. Bitcoin wallets work as digital "bank accounts" that can only be controlled by the entity behind them. When a wallet is created, two keys are generated: a public and a private key.

- Public keys are addresses used to send and receive payments. They're akin to a bank account number.
- Private keys are akin to the password protecting a bank account, and anyone who controls the private key to a wallet controls the wallet. As there is no central authority on the Bitcoin network, if a private key is lost, the coins on that wallet are lost.

Bitcoin is used for a number of purposes. Some people use it for everyday transactions, while others prefer to use BTC as a store of value, making it an alternative to gold. Others simply invest, trade, and speculate using the cryptocurrency.

## Why Does Bitcoin Have Value?

Bitcoin's high value is determined by a number of factors. The cryptocurrency was the first to solve the Byzantine Generals' problem, bringing trust to a decentralized system. As the system is decentralized and is governed by code, its fixed and predictable monetary policy cannot be changed unless there's consensus to do so.

Bitcoin uses open-source code and is built on top of a transparent network, making it possible for anyone to independently verify its security, its activity, and the balances of specific accounts on the blockchain.

Miners use tremendous amounts of energy to support Bitcoin's encrypted network, forcing potential attackers to require impossible amounts of energy to do anything to it. The network's uptime since inception is above 99.987%, making it more reliable than traditional payment networks.

Moreover, anyone can create a Bitcoin wallet and start using the network, making it open to anyone in the world regardless of their financial conditions. Bitcoin is an unencodable network that allows for fast peer-to-peer transactions throughout the world at low transaction fees.

While no single entity controls Bitcoin, everyone can participate in the project by creating new businesses around it, helping develop it, mining it, running a node to help secure and relay transactions, documenting its history, using BTC, or simply talking about it.

## What is Ethereum?

Ethereum(ETH), Founded by Vitalik Buterin and Gavin Wood in 2015, is the second-biggest cryptocurrency by market cap after Bitcoin. But unlike Bitcoin, it wasn't created to be digital money. Instead, Ethereum's founders set out to build a new kind of global, decentralized computing platform that takes the security and openness of blockchain and extends those attributes to a vast range of applications, In other words it can run a wide range of applications. Ethereum developers,
can create complex applications with nearly endless capabilities. Everything from financial tools and games to complex databases are already running on the Ethereum blockchain. To many investors Ethereum's value is based on its flexibility as a platform for issuing stable coins, resulting in a growing user base and growing transaction fees. Today Ethereum's market capitalization represents approximately 20% of the $1.1 trillion global crypto market. Many benefits of blockchain technology apply to Ethereum, including the following:

- Availability: Because Ethereum is decentralized, there is no downtime if a node goes down. Other computing models use centralized servers and can suffer performance issues if interrupted.

- Permissionless: Ethereum is a permissionless block chain, meaning that everyone can participate. This contrasts with permissioned block chains, which are limited to designated participants.

Ethereum is a decentralized, open-source blockchain platform that enables developers to build and deploy smart contracts and decentralized applications (dApps). It runs on a global network of nodes, making it highly secure and resistant to censorship. Ethereum introduced the concept of programmable money, allowing users to interact with the blockchain through self-executing contracts, also known as smart contracts. Ethereum's native currency, Ether (ETH), powers these contracts and facilitates transactions on the network.

## What is ETH and why does it have value?

ETH, or Ether, is the native cryptocurrency of the Ethereum platform. It is used to pay for transaction fees and computational services on the network. ETH also serves as a digital store of value due to its decreasing issuance rate, collateral for crypto loans, and as a payment system in various financial apps built on Ethereum. Many people also view ETH as an investment, similar to Bitcoin and other cryptocurrencies.

## What is Ethereum used for?

Ethereum's primary use is for executing smart contracts, which enable various applications, such as decentralized finance (DeFi), stablecoins, non-fungible tokens (NFTs), and decentralized autonomous organizations (DAOs). These applications are built on top of the Ethereum platform, providing a wide range of financial services, digital asset management, and even tokenization of real-world assets like real estate and art. The platform is also home to thousands of tokens, each with its unique use case and value and hosts internet communities governed by token holders, further expanding the potential of decentralized organizations.

## What are Ethereum tokens?

Tokens are digital assets created and traded on the Ethereum platform. They represent a wide range of use cases, including stablecoins that mirror the value of traditional currencies, governance tokens representing voting power in decentralized organizations, collectible tokens (NFTs) for digital art or unique assets, and various utility tokens with specific functions within their respective ecosystems. Ethereum is home to thousands of tokens, with developers constantly building new tokens that unlock new possibilities and open new markets.

## What are smart contracts?

Smart contracts are self-executing computer programs that live on the Ethereum blockchain. They are triggered by transactions from users or other contracts and enable complex applications and interactions on the platform. Smart contracts form the basis of decentralized apps (dApps) and are immutable, meaning they cannot be altered once deployed. They provide transparency, security, and automation, and are used in various applications like lending platforms, decentralized exchanges, insurance, crowdfunding, and more.

## Who created it?

Ethereum was proposed by Vitalik Buterin in 2013 and officially launched in 2015, following a successful crowdfunding campaign. It has since been developed and maintained by the Ethereum community, consisting of developers, researchers, and enthusiasts worldwide.

## What was The Merge and why was it important?

The Merge was a significant upgrade to the Ethereum network, transitioning it from a proof-of-work (PoW) to a proof-of-stake (PoS) consensus mechanism. This event marked the joining of the original execution layer (Ethereum Mainnet) with its new PoS consensus layer, the Beacon Chain. The Merge played a crucial role in Ethereum's development for several reasons:

- Energy Efficiency: The Merge reduced Ethereum's energy consumption by approximately 99.95%, making it a more sustainable and environmentally friendly blockchain platform.
- Decentralization: Ethereum is decentralized, so there is no interference from third-party cloud providers. It uses blockchain, which enables peer-to-peer transactions. The switch to PoS incentivized more participants to join the network as validators, further decentralizing the Ethereum ecosystem and reducing the risk of centralization that can occur in PoW mining.
- Enhanced Security: Like any decentralized blockchain-based network, Ethereum is designed to be unhackable. Hackers would have to control most of the network nodes to exploit the network. By adopting the PoS consensus mechanism, Ethereum's security was strengthened as it became less vulnerable to potential attacks, such as the 51% attack, which is more common in PoW networks.
- Scalability: The Merge laid the foundation for future upgrades aimed at increasing the scalability of the Ethereum network, allowing it to handle more transactions and support a larger user base.
- Continued Evolution: The Merge represented a major milestone in Ethereum's ongoing development, showcasing the project's adaptability and commitment to growth, innovation, and improvement.

## What is Litecoin?

Litecoin (LTC) is a peer-to-peer cryptocurrency that was set up by Charlie Lee (a former Google employee) in 2011. Litecoin was one of the first “altcoins”—a name given to cryptocurrencies other than Bitcoin. It shares many similarities with bitcoin and is based on bitcoin’s original source code. Similar to Bitcoin, Litecoin utilizes a transparent and publicly accessible ledger to record all transactions. This ensures the integrity and transparency of the network. It was designed to address concerns about centralization and make mining more difficult to dominate for large-scale mining firms. It has a faster transaction processing time than bitcoin and was designed to be a medium for daily transactions. LTC has a maximum supply of 84 million coins. Additionally, Its faster transaction speeds, lower transaction fees, and secure payment network make it an attractive option for everyday transactions.

Litecoin was designed as a "lite version of Bitcoin," with the primary goal of improving upon Bitcoin's perceived shortcomings and to offer faster transaction confirmation times. Litecoin is based on an open-source global payment network that is not controlled by any central authority and uses "scrypt" as a proof of work, which can be decoded with the help of consumer-grade CPUs. Although it is technically similar to Bitcoin, Litecoin has some differences, such as a reduced block generation time of approximately 2.5 minutes (compared to Bitcoin's 10 minutes), a higher maximum number of coins, a different hashing algorithm (scrypt instead of SHA-256), and a slightly modified GUI.

## What is Litecoin (LTC) used for?

Litecoin is used for a variety of purposes, including instant, near-zero cost payments to anyone in the world. It serves as a digital currency for peer-to-peer transactions, online purchases, and transferring funds. Its faster transaction confirmation time makes it attractive for small, everyday transactions as well as larger transfers. Merchants also benefit from the fast confirmation times of Litecoin, which enable them to accept payments more quickly compared to Bitcoin.

## Who created Litecoin (LTC)?

Litecoin was created by Charlie Lee on October 8, 2011. Charlie Lee is a computer scientist with a background in technology and an interest in cryptocurrencies. Prior to creating Litecoin, he worked for Google on projects such as YouTube Mobile and Chrome OS. His intention with Litecoin was to improve upon Bitcoin's limitations and to create a lighter, more efficient version of Bitcoin for everyday use. Lee has been an outspoken advocate for cryptocurrencies and continues to be involved in the Litecoin project, although he announced in December 2017 that he had sold all his Litecoin holdings to avoid any conflict of interest and to focus on Litecoin's development.

## What is Monero and why we use it?

Monero (XMR) is an open source, privacy-oriented cryptocurrency that was introduced with the primary intention of facilitating private and anonymous transactions. Unlike many other digital assets, Monero's design aims to conceal the details of senders and recipients through advanced cryptography, making it challenging to trace payments back to their original source. This emphasis on privacy and security is a fundamental aspect of Monero, with the project striving to provide protection to all users, regardless of their technical competence. Monero seeks to enable

quick and inexpensive payments without the fear of censorship. Monero uses privacy-enhancing technologies to make its users and transactions anonymous and confidential.

The sender, receiver, and amount of every transaction on the Monero block chain are hidden.

Monero's software is programmed to update every six months, a regular schedule that has helped it more aggressively add new features without much controversy.

Monero (XMR) is a secure, private, untraceable digital currency, and a form of decentralized cryptocurrency that is based on cryptographically secure technology. It utilizes a combination of ring signatures, ring confidential transactions and stealth addresses to protect sender and receiver privacy, and employs a proof-of-work consensus algorithm to ensure the security of the network.

Monero (XMR) is listed on a variety of cryptocurrency exchanges and is accepted by many merchants and services.

## What is XMR used for?

Monero (XMR) is used as a medium of exchange between individuals or organizations, allowing for fast and secure money transfers over the internet. It is also used to purchase goods and services, and can be exchanged for other cryptocurrencies or fiat currencies.

## Who created Monero (XMR)?

Monero (XMR) was created by a group of developers in April 2014, led by a pseudonymous developer known as "thankful*for*today". The code was originally a fork of Bytecoin, but was heavily modified to provide additional privacy and security features.

# 1.5 Cryptocurrency price prediction

Cryptocurrency price prediction indeed plays a crucial role in assisting cryptocurrency investors in making informed investment decisions. By analyzing historical price data, market trends, and various technical indicators, predictive models aim to forecast future price movements of cryptocurrencies. These predictions can be invaluable for investors seeking to time their trades effectively, mitigate risks, and potentially maximize profits in a highly volatile market environment. Moreover, accurate price predictions can aid investors in identifying entry and exit points, optimizing portfolio allocations, and implementing trading strategies tailored to their risk tolerance and investment objectives.

Furthermore, beyond its utility for individual investors, cryptocurrency price prediction also serves as a valuable tool for financial researchers and analysts studying the behavior of cryptocurrency markets. By examining the efficacy of different prediction models, evaluating the impact of market factors on price dynamics, and assessing the accuracy of forecasting techniques, researchers contribute to a deeper understanding of the underlying mechanisms driving cryptocurrency markets. This research is instrumental in uncovering patterns, anomalies,

and trends within the market, shedding light on market inefficiencies, investor sentiment, and the interplay between various economic and technological factors.

Overall, cryptocurrency price prediction not only benefits investors seeking to navigate the complexities of the market but also contributes to the advancement of academic research and financial analysis in understanding the dynamics of cryptocurrency markets. Through the development of robust prediction models and empirical studies, stakeholders can gain insights into market behavior, enhance decision-making processes, and contribute to the maturation and evolution of the cryptocurrency ecosystem as a whole.

## 1.6 System development life cycle

Phase 1: Planning

Phase 2: Analysis

Phase 3: Design

Phase 4: Implementation

Phase 6: Testing

Phase 7: Deployment

## Project Plan (Planning):

| Task Name | Duration | Start | Finish |
|---|---|---|---|
| Survey | 14 days | 1/10/2023 | 14/10/2023 |
| Data collection | 2 days | 14/10/2023 | 15/10/2023 |
| Data Pre-Processing | 7 days | 16/10/2023 | 22/10/2023 |
| ARIMA model | 7 days | 22/10/2023 | 28/10/2023 |
| SVR model | 7 days | 22/10/2023 | 28/10/2023 |
| LSTM model (single Feature) | 7 days | 29/10/2023 | 4/11/2023 |
| GRU model(single Feature) | 7 days | 29/10/2023 | 4/11/2023 |
| LSTM model (multi feature) | 7 days | 5/11/2023 | 11/11/2023 |
| GRU model (multi feature) | 7 days | 5/11/2023 | 11/11/2023 |
| Models Evolution | 15 days | 12/11/2023 | 26/11/2023 |
| Diagrams | 7 days | 27/11/2023 | 3/12/2023 |
| Build Website | 30 days | 4/12/2023 | 3/1/2024 |
| Twitter Sentiment Analysis | 14 days | 4/1/2024 | 18/1/2024 |
| Flask API | 14 days | 19/1/2024 | 2/2/2024 |
| Prepare for final discussion | 10 days | 3/2/2024 | 12/2/2024 |

# Chapter 2

# Analysis

## 2.1 Problem statement

- Market Volatility: Cryptocurrency markets are highly volatile, making it challenging for investors to predict price movements and optimize their investment strategies.
  Cryptocurrency markets are renowned for their extreme volatility, characterized by rapid and unpredictable price fluctuations. This volatility stems from various factors, including market sentiment, regulatory developments, technological advancements, and macroeconomic trends. For investors, navigating this volatile landscape can be daunting, as traditional financial analysis methods may struggle to capture the unique dynamics of cryptocurrency markets. The lack of centralized control and the prevalence of speculative trading further exacerbate price volatility, creating both opportunities for substantial gains and risks of significant losses. As a result, investors often find it challenging to predict price movements accurately and devise effective investment strategies to capitalize on market opportunities while managing risks.

- Risk Management: Investors lack effective tools to assess and manage risks associated with cryptocurrency investments, leading to potential financial losses. Despite the growing popularity of cryptocurrencies, investors face significant challenges in assessing and managing the risks inherent in this asset class. Unlike traditional financial assets, cryptocurrencies are subject to a wide range of risks, including market volatility, regulatory uncertainty, cybersecurity threats, and technological vulnerabilities. Moreover, the lack of established risk management frameworks and regulatory oversight further complicates the process of evaluating and mitigating risks associated with cryptocurrency investments. As a result, investors may struggle to identify and address potential risks effectively, increasing the likelihood of experiencing financial losses. Developing robust risk management strategies tailored to the unique characteristics of cryptocurrency markets is essential for investors to safeguard their investments and navigate this rapidly evolving landscape with confidence.

- Decision Support Lack: There is a need for a reliable prediction model to provide investors with timely and accurate information, aiding in decision-making and portfolio management.
  In the rapidly evolving landscape of cryptocurrency markets, investors often grapple with the lack of reliable tools to assist them in making informed decisions. The inherent volatility and complexity of these markets necessitate robust decision support systems that can analyze vast amounts of data and generate actionable insights in real-time.

Traditional financial analysis techniques may prove inadequate in capturing the unique dynamics of cryptocurrency markets, highlighting the need for specialized prediction models.

A reliable prediction model would leverage advanced statistical methods, machine learning algorithms, and sentiment analysis techniques to forecast cryptocurrency price movements with a high degree of accuracy. By analyzing historical price data, market trends, trading volumes, and relevant external factors, such as news sentiment and social media activity, the model can identify patterns and correlations that inform investment decisions.

Moreover, timely and accurate information provided by the prediction model can empower investors to optimize their portfolio management strategies. Whether it's identifying optimal entry and exit points, rebalancing asset allocations, or hedging against potential risks, access to reliable predictive analytics can significantly enhance investors' ability to navigate the volatile cryptocurrency market landscape effectively.

Furthermore, such a prediction model could also serve as a valuable tool for financial institutions, asset managers, and policymakers seeking to understand market dynamics, assess risk exposures, and develop strategic investment approaches in the cryptocurrency space.

In summary, the development of a reliable prediction model represents a critical step towards addressing the decision support lack in cryptocurrency investing. By providing investors with timely and accurate insights, such a model can facilitate informed decision-making, mitigate risks, and ultimately contribute to the long-term success and sustainability of cryptocurrency investments.

## 2.2 Aims and Objectives

- **Provide accurate price forecasting:** Accurate price forecasting is essential for investors to anticipate market movements and make informed decisions. By utilizing advanced analytical techniques such as machine learning algorithms and statistical models, accurate price forecasting tools can analyze historical data, market trends, and relevant external factors to generate predictions with a high degree of precision. These forecasts enable investors to identify potential opportunities for profit and adjust their investment strategies accordingly, thereby maximizing returns and minimizing risks in the volatile cryptocurrency market

- **Enabling informed decision-making:** Informed decision-making is paramount for successful investing in cryptocurrency markets. By providing investors with timely and relevant information, decision support tools empower them to assess market conditions, evaluate potential risks and rewards, and make well-informed investment decisions. These tools leverage sophisticated data analysis techniques to interpret market dynamics, identify trends, and forecast future developments, enabling investors to navigate the complex cryptocurrency landscape with confidence and clarity.

- **Predicting risks:** Predicting risks is crucial for investors to mitigate potential losses and safeguard their investments. Risk prediction models analyze various factors such as market volatility, regulatory developments, cybersecurity threats, and technological vulnerabilities to assess the likelihood and impact of potential risks on cryptocurrency investments. By identifying and quantifying risks in advance, investors can implement risk management strategies such as diversification, hedging, and portfolio rebalancing to minimize exposure to adverse market conditions and protect their capital.

- **Minimizing financial losses:** Minimizing financial losses is a primary objective for investors in the cryptocurrency market. Effective risk management strategies, informed by accurate price forecasting and risk prediction, play a crucial role in mitigating potential losses and preserving investment capital. By identifying and addressing potential risks proactively, investors can minimize the impact of adverse market movements and avoid significant financial losses, thereby safeguarding their wealth and achieving long-term investment objectives.

- **Enhancing transparency and trust:** Enhancing transparency and trust is essential for fostering investor confidence and promoting the long-term sustainability of cryptocurrency markets. Transparent pricing mechanisms, reliable forecasting tools, and comprehensive risk assessment frameworks contribute to greater market transparency and investor trust. By providing investors with access to accurate and timely information, these tools empower them to make informed decisions and participate in cryptocurrency markets with greater confidence, ultimately enhancing market integrity and fostering trust among participants.

## 2.3 Suggested solution

The suggested solution entails developing a sophisticated system based on machine learning and deep learning models to predict cryptocurrency prices accurately. This system would leverage historical price data, market trends, and relevant external factors to generate forecasts with a high degree of precision. By employing advanced analytical techniques, such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, the system can capture complex patterns and correlations in cryptocurrency price movements, enabling more accurate predictions.

In addition to its predictive capabilities, the system would provide a user-friendly website interface to facilitate market decision support. This interface would allow investors to access real-time price forecasts, market insights, and risk assessments, empowering them to make informed investment decisions. Through interactive charts, customizable alerts, and portfolio

management tools, users can navigate the cryptocurrency market landscape with confidence and clarity, optimizing their investment strategies and maximizing returns.

Key features of the website interface may include:

- **Real-time Price Forecasts:** Users can view up-to-date predictions for various cryptocurrencies, including short-term and long-term price projections based on the latest data and model updates.

- **Market Insights:** The system provides valuable insights into market trends, sentiment analysis, and key drivers influencing cryptocurrency prices, helping users understand market dynamics and make informed decisions.

- **Risk Assessment:** Users can assess the potential risks associated with cryptocurrency investments, including market volatility, regulatory developments, and cybersecurity threats, enabling them to manage risk effectively.

- **Customizable Alerts:** Users can set personalized alerts for price movements, market trends, and risk factors, ensuring they stay informed about significant developments and opportunities in the cryptocurrency market.

- **Portfolio Management Tools:** The interface offers tools for tracking and managing cryptocurrency portfolios, including performance analytics, asset allocation recommendations, and rebalancing strategies to optimize investment outcomes.

Overall, the proposed system aims to address the decision support lack in cryptocurrency investing by providing investors with a comprehensive platform for accurate price forecasting, informed decision-making, and risk management. By combining cutting-edge machine learning models with user-friendly interface design, the system empowers users to navigate the complexities of the cryptocurrency market with confidence and transparency, ultimately enhancing their ability to achieve their investment objectives.

# 2.4 Related Work

## 2.4.1On Forecasting Cryptocurrency Prices:

Researchers Kate Murray et al conducted a detailed study on cryptocurrency price forecasting, exploring traditional statistical methods, machine learning models, and deep learning models like LSTM and GRU. The dataset, encompassing Bitcoin (BTC) – Litecoin(LTC) - Ethereum(ETH) – Monero (XMR) and Ripple (XPR) over a five-year period from 2017 to 2022, was sourced from Binance.com and Investing.com. Evaluation using metrics such as Mean absolute percentage error (MAPE), Root mean absolute error(RMSE) and Mean absolute error(MAE) [1].

**Table 2.** The average performance of individual models ranked by RMSE in ascending order.

| Model | RMSE | MAE | MAPE | R2 | Train (s) | Inference (ms) |
|---|---|---|---|---|---|---|
| LSTM | 0.02224 | 0.0173 | 3.862% | 0.735 | 173.765 | 1.862 |
| GRU | 0.02285 | 0.0176 | 3.939% | 0.720 | 254.520 | 1.550 |
| HYBRID | 0.02295 | 0.0177 | 3.959% | 0.717 | 461.967 | 2.383 |
| KNN | 0.02332 | 0.0179 | 4.003% | 0.711 | <0.01 | 0.074 |
| TCN | 0.02334 | 0.0180 | 4.021% | 0.711 | 40.475 | 1.219 |
| ARIMA | 0.02343 | 0.0180 | 4.010% | 0.708 | 4.035 | 0.109 |
| TFT | 0.02353 | 0.0181 | 4.062% | 0.707 | 105.913 | 8.842 |
| RF | 0.02402 | 0.0184 | 4.095% | 0.697 | 2.121 | 0.586 |
| SVR | 0.02452 | 0.0189 | 4.240% | 0.681 | <0.01 | 0.008 |

Figure 2.4.1

This table shows the average performance of each model computed across all cryptos.Models are ranked by RMSE in ascending order.
First, we observe that the models' ranking is consistent across all the accuracy metrics (with very few exceptions). The LSTM exhibits the best performance, with a consistent gap compared to the other models. For each metric, values are quite close because we compute them on the normalised predicted price, and not on the deternded data. The recurrent neural network models occupy the first three positions of the rank, followed by the KNN and the convolutional network approach. Interestingly, ARIMA performs better than TFT, RF, and SVR.

## 2.4.2Comparative study of bitcoin price prediction using SVR and LSTM:

Researchers J. Arumugam et al a study was conducted on predicting Bitcoin prices using two models, one of which is a machine learning model, which is Support Vector Regression (SVR), and the other is a deep learning model, which is Long-Short Term Memory (LSTM). They collected the data set from coindesk.com. The timestamp of the dataset contains one day's data in USD from July 2010 to March 2021. The training/test ratio is split 80:20 for model learning[2].

| Training(%) | Test(%) | Training size | Test size | SVR Normalized MSE | LSTM Normalized MSE |
|---|---|---|---|---|---|
| 40 | 60 | 1549 | 2325 | 6.53971E-05 | 0.003671773 |
| 50 | 50 | 1937 | 1937 | 7.84791E-05 | 0.006783988 |
| 60 | 40 | 2324 | 1550 | 9.67046E-05 | 0.011204557 |
| 70 | 30 | 2711 | 1163 | 0.000125415 | 0.002729209 |
| 80 | 20 | 3099 | 775 | 0.000146273 | 0.001398904 |
| 90 | 10 | 3486 | 388 | 0.000257713 | 0.003696527 |

Figure 2.4.2.1

This table shows that training/testing dataset ratios of LSTM and SVR are 80:20 and 40:60 respectively for optimum models. Applying machine learning algorithms on a dataset in different sizes for training and test purpose is not fixed for better results, 80:20 in particular.

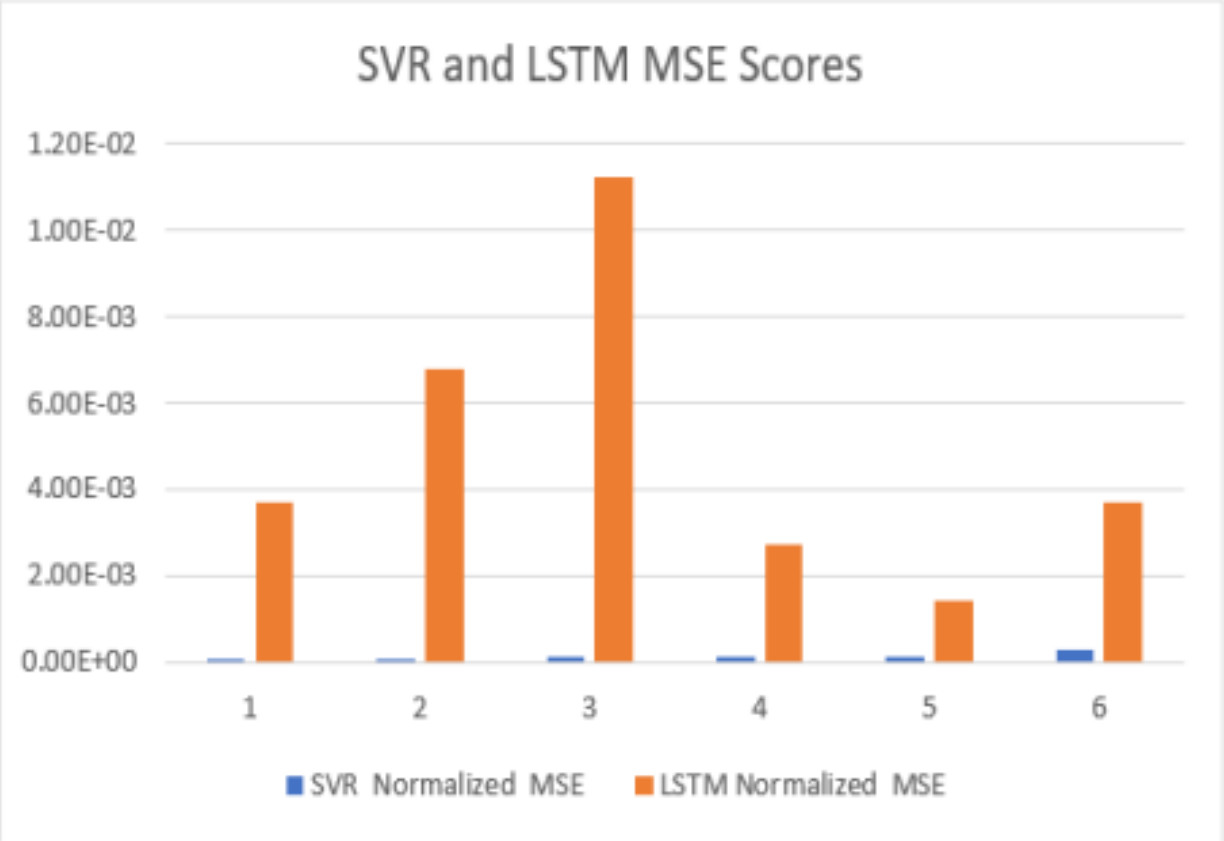



**Fig. 4.** SVR and LSTM MSE Scores

Figure 2.4.2.2

The study on Support Vector Machine, linear regression, and Long Short-Term Memory has been done on the bitcoin dataset and these algorithms help in predicting the prices of bitcoin for future days.

The Long Short-Time Memory (LSTM) provides the best-predicted result.

## 2.4.3FORECASTING CRYPTOCURRENCY PRICES USING DEEP LEARNING: INTEGRATING FINANCIAL, BLOCKCHAIN, AND TEXT DATA:

Vincent Gurgul, Stefan Lessmann and Wolfgang Karl Härdle applied Machine Learning (ML) and Natural Language Processing (NLP) techniques in cryptocurrency price forecasting, specifically Bitcoin (BTC) and Ethereum (ETH). Focusing on news and social media data, primarily from Twitter and Reddit, They analysed the influence of public sentiment on cryptocurrency valuations using advanced deep learning NLP methods. They used a very diverse set of data sources with the time frame of the dataset starting from August 2011 for BTC and August 2015 for ETH [31].

Table 5.2: Average performance by time series model

| Model | Excess profit* | Trades** | AUC ROC** | Accuracy** |
|---|---|---|---|---|
| OLS/Logit | 67.48 % | 52.7 | 0.6782 | 0.8025 |
| XGBoost | 126.12 % | 44.0 | 0.6998 | 0.8057 |
| MLP (FNN) | 138.61 % | 47.4 | 0.6797 | 0.8065 |
| LSTM | 83.88 % | 12.0 | 0.6526 | 0.8028 |
| TFT | 11.13 % | 4.2 | 0.5653 | 0.7971 |

* Profit exceeding buy-and-hold strategy

** All metrics are averages of 7-fold cross-validation and were aggregated across all target variables

Figure 2.4.3

The results show that LSTM displays good prediction ability as the accuracy of LSTM is high.

## 2.5 Data collection

# Dataset:

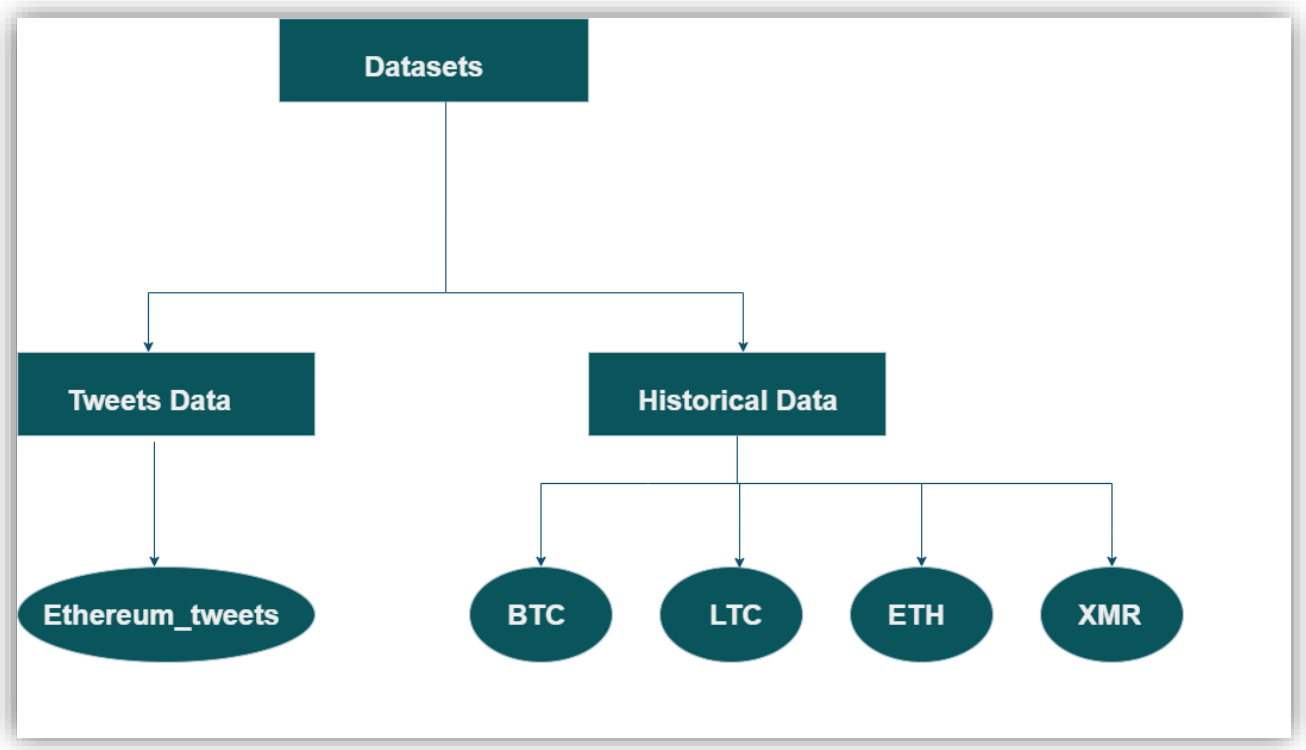



Figure 2.5

We used 5 datasets in this project as follows:

**First:** We gathered Data from Yahoo. Finance and We choose four datasets:

1)Bitcoin (BTC-USD.csv)

2)Litecoin (LTC-USD.csv)

3)Ethereum (ETH-USD.csv)

4)Monero (XMR-USD.csv)

Each dataset contains Columns which contains Date, Open, High, Low, Close, adjust close and Volume

Date: date of the record.

Open: the opening price, the price at which a coin trades at the beginning of the day.

High: the maximum price of the day, the highest price reached on that day.

Low: the minimum price of the day, the lowest price reached on that day.

Close: the closing price, the price at which coin trades at the end of the day.

Adjusted close: is the closing price after adjustments for all applicable splits and dividend distributions.

Volume: the sum of actual trades made during the day.

BTC contains 2480 raw from 1/1/2017 to 10/15/2023.

LTC contains 3316 raw from 9/18/2014 to 10/15/2023.

ETH contains 2146 raw from 12/1/2017 to 10/15/2023.

XMR contains 2167 raw from 11/10/2017 to 10/15/2023.

## Why do we choose yahoo finance?

Yahoo Finance is a popular choice for gathering cryptocurrency price data for several reasons:

- Reliability: Yahoo Finance is a trusted source for financial data, including cryptocurrency prices. Users trust its accuracy and reliability.
- Clean data: Yahoo Finance provide clean data that that is free of error and noise, don't have outliers and don't have redundancy
- Historical Data: Yahoo Finance provides historical data, allowing for comprehensive analysis and back testing of trading strategies.
- Integration: Yahoo Finance integrates well with various analytical tools and platforms, making it easy to incorporate cryptocurrency price data into existing workflows.
- Coverage: While not as extensive as some specialized cryptocurrency platforms, Yahoo Finance still covers a wide range of cryptocurrencies, providing data on many popular coins.

Overall, Yahoo Finance offers a combination of reliability, historical data, accessibility, and coverage that makes it a suitable choice for gathering cryptocurrency price data for dataset creation and analysis.

**Second:** We gathered Data from kaggle.com and we choose two datasets:

2) Bitcoin (Bitcoin Tweets.csv)

**Content:**

The tweets have #Bitcoin and #btc hashtag.. Collection star started on 6/2/2021, with an initial 100,000 tweets, and will continue on a daily basis.

2) Ethereum (Ethereum Tweets.csv)

**Content:**

The tweets have #Ethereum hashtag. Collection started on 01/01/2021.

Information regarding the data:

The data totally consists of 1 lakh+ records with 13 columns.

1- user_name: The name of the user, as they've defined it.
2- user_location: The user-defined location for this account's profile.
3- user_description: The user-defined UTF-8 string describing their account.
4- user_created: Time and date, when the account was created.
5- user_followers: The number of followers an account currently has.
6- user_friends: The number of friends an account currently has.
7- user_favourites: The number of favorites an account currently has
8- user_verified: When true, indicates that the user has a verified account
9- date: UTC time and date when the Tweet was created
10- text: The actual UTF-8 text of the Tweet
11- hashtags: All the other hashtags posted in the tweet along with #Bitcoin & #btc
12- source: Utility used to post the Tweet, Tweets from the Twitter website have a source value - web
13- is_retweet: Indicates whether this Tweet has been Retweeted by the authenticating user.

★ The most important column we used to train the model on is text.

# Chapter 3

# Design

## 3.1 work flow:

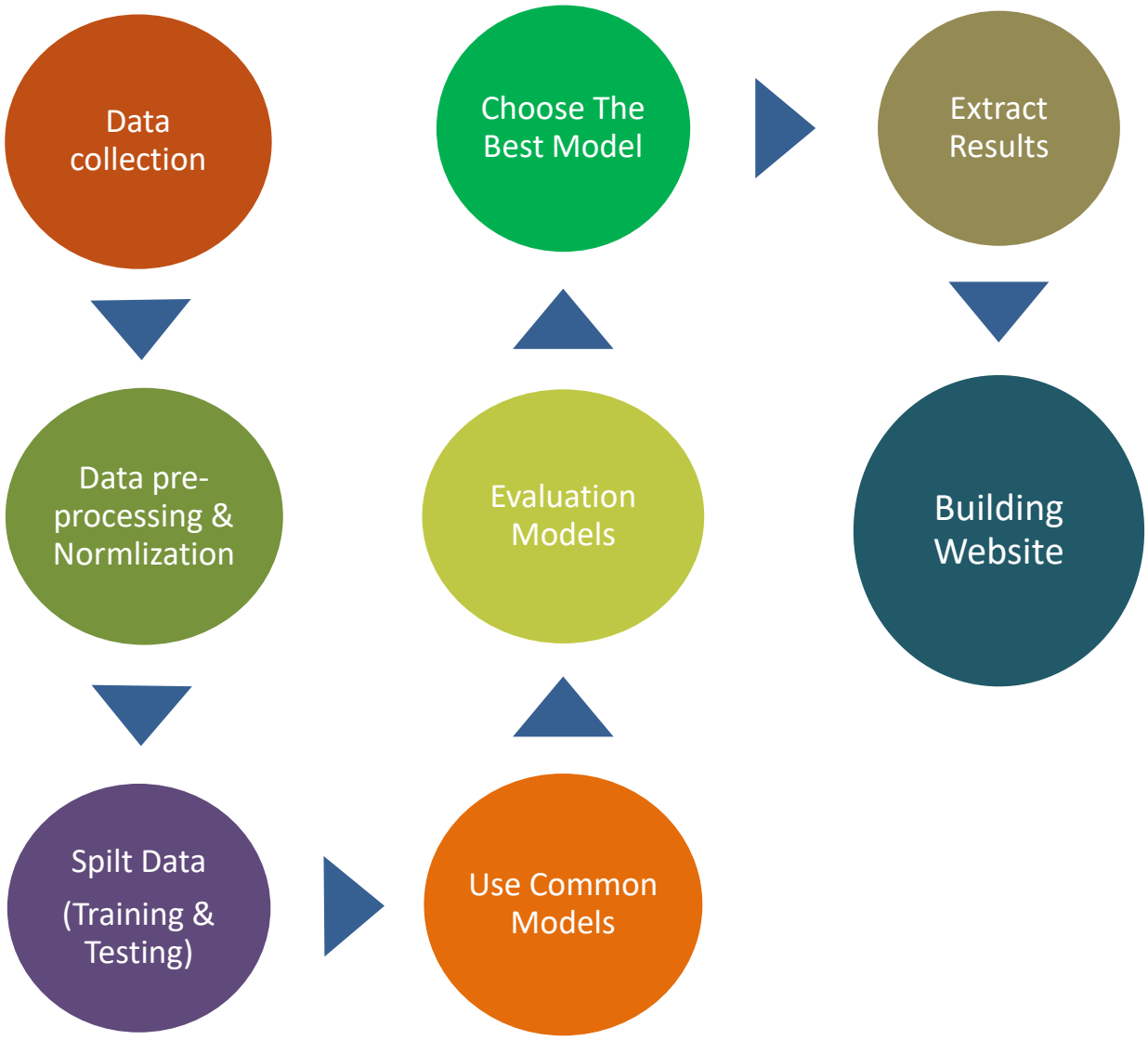



Figure 3.1

## 3.2 Flow chart:

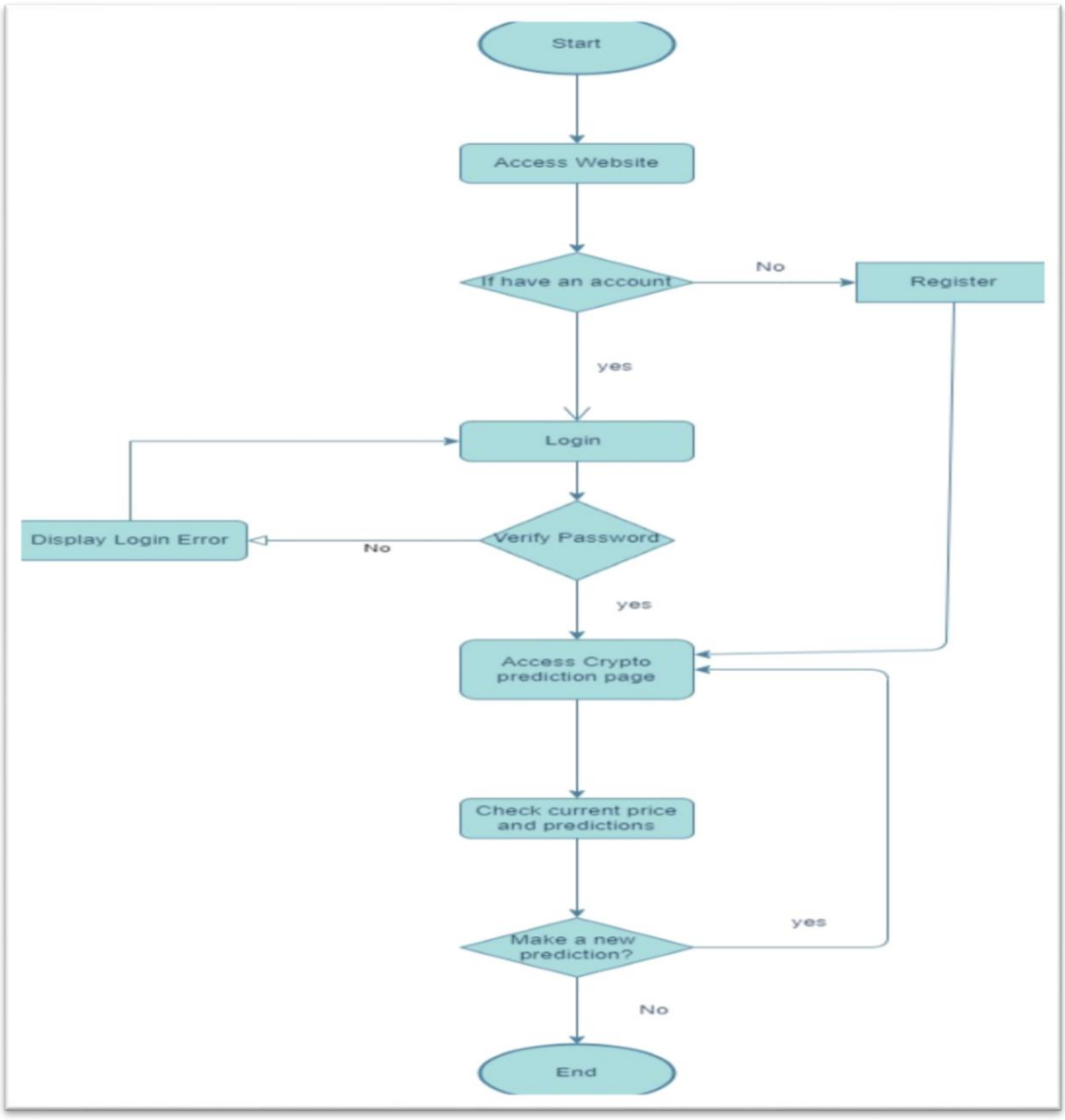



Figure 3.2

A flowchart is one of the most common diagrams used to represent a process or workflow of our system. First the flow chart has a default start point. Second the user will access the website and see home page. Third if the user does not have an account and it's his first time to use our system then he will go to make a registration in sign up page. If the user already has an account, he will go directly to make login in sign in page. Fourth it is a verify password step which means when

the user makes a login the system will check about correctness of user password if it isn't correct the user will go to make a login again or click on forget password button in order to go to update page so user can reset password. If the password is correct then the user can see cryptocurrencies page which contains (BTC, LTC, ETH and XMR) and choose the coin he interested to see its current price and predictions. Finally, after the user see the prediction of coin he chose he will decide if he want to see and make another prediction or it will be the end of the system.

## 3.3 Use Case Diagram

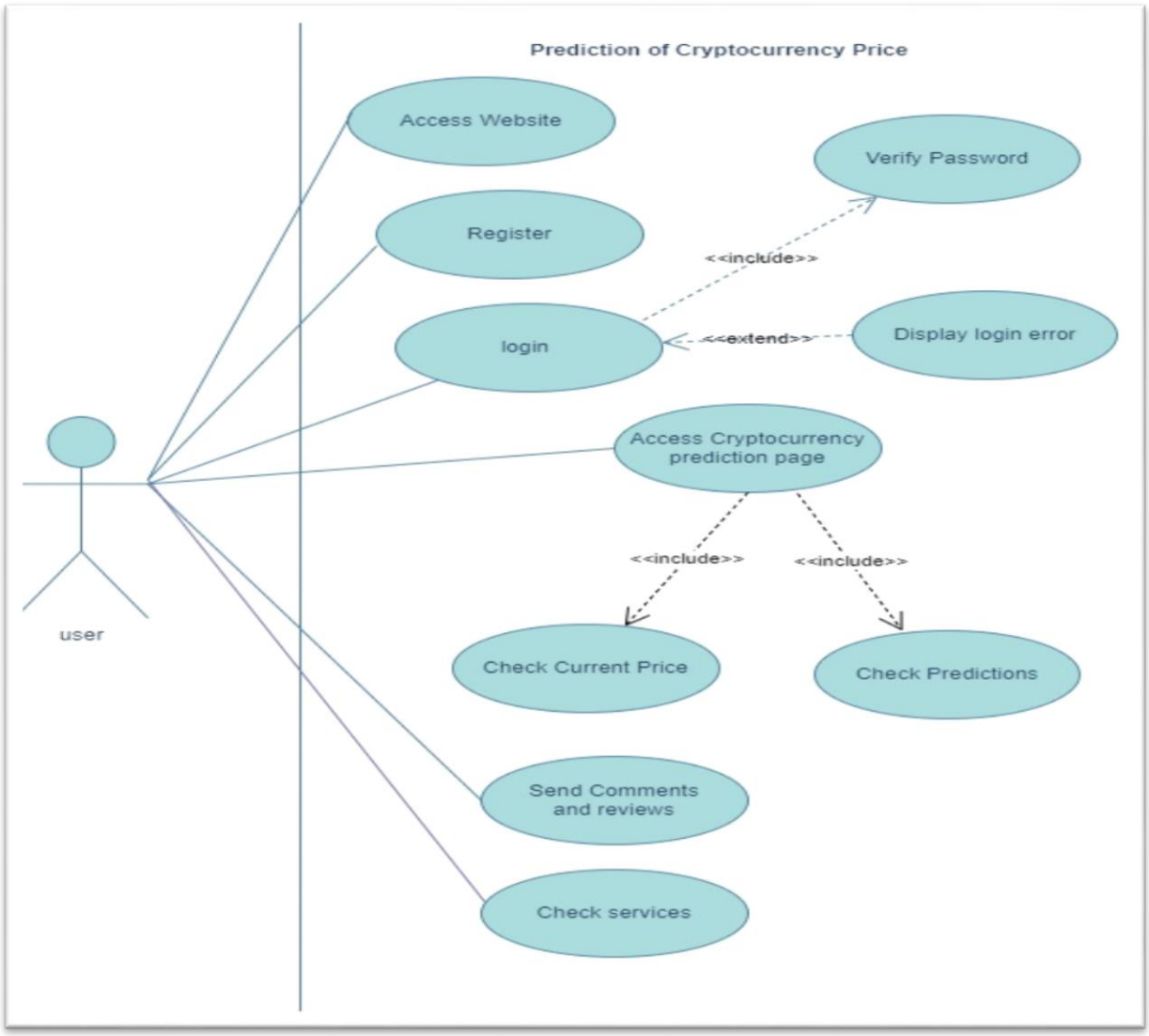



Figure 3.3

The use case diagram shows the different types of users and the tasks they can perform with our website. The primary users are investors in cryptocurrency markets, who interact with our website to know predictions about cryptocurrencies they interested in it.
The use case diagram includes the following use cases:

1. Access website: Users can access and interact with our website by searching about the website.

2. Register: New users can register for an account and creating a username and password.

3. Login: Users can log in to our website using their email and password. Login use case has a include relationship with verify password use case which means that user can't login successfully without verifying his password, also login use case has extend relationship with display login error use case in case user enter an incorrect password.

4. Access Cryptocurrency Prediction Page: the website has 4 pages for cryptocurrencies (BTC, LTC, ETH and XMR). Users can interact with cryptocurrencies pages to see predictions and current prices for each coin.

5. Send comments and review: users can send suggestions, comments, review and opinions through contact us page.

6. check Services: users can check our services though services page.

These use cases represent the primary functions of our system and provide a high- level overview of the user interactions.

## 3.4 Sequence Diagram

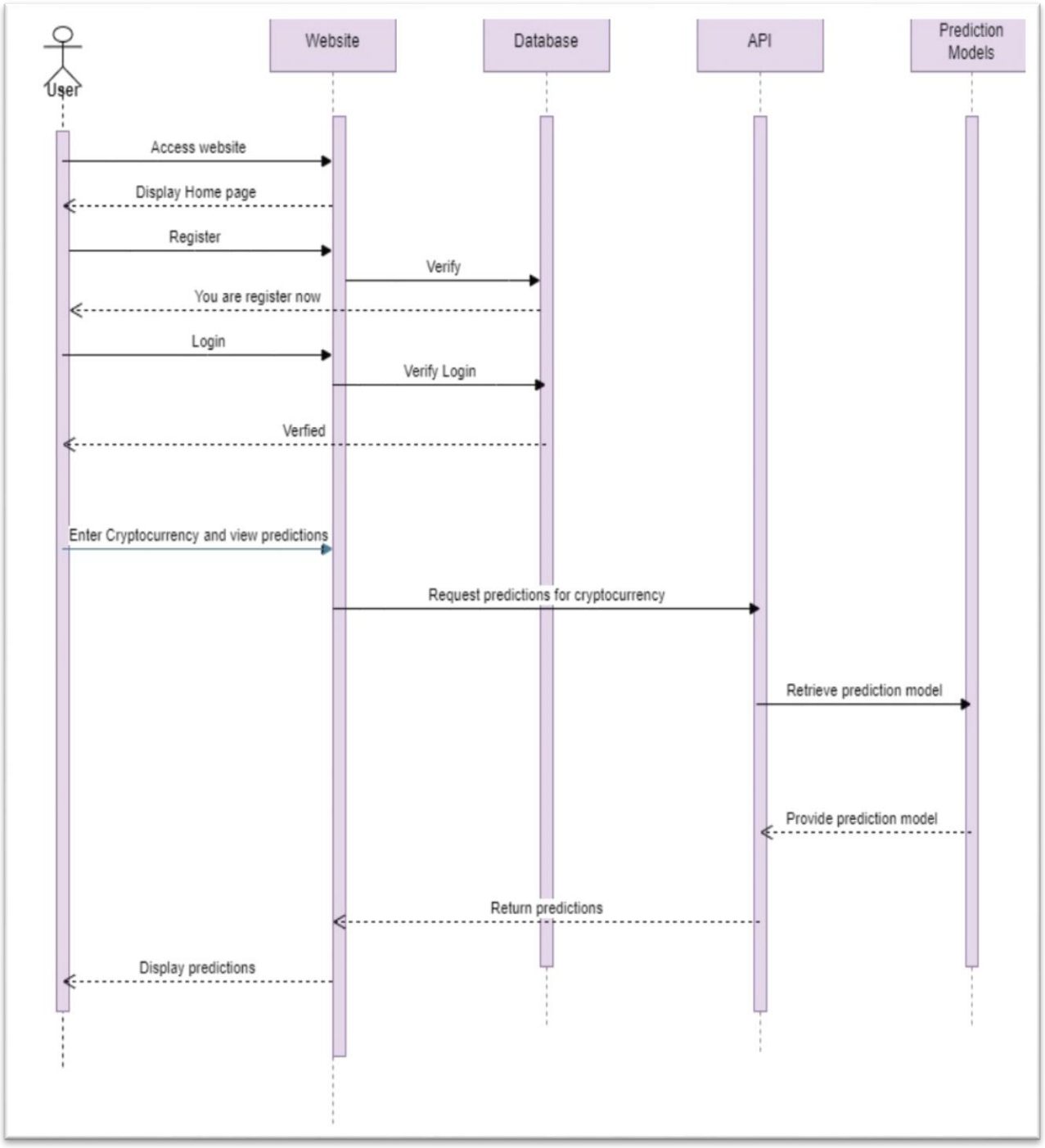



Figure 3.4

The sequence diagram shows the interactions between the different components of the system during a typical user session. Here are the details of each component:

1. User (actor): which represents investors or any person has interest in cryptocurrency market.
2. website (object): which has home page, cryptocurrency prediction page, services and contact us page.

3. Database (object): The backend will store information of users in database for the first time when the user makes a sign up request. Backend sends requests to the database to check user data in case of making a sign in request.

4. API (object): Flask API is used here to connect the website with Predictions Models.

5. Prediction Models (object): The ML and DL models we used.

First, the user will access website after searching about it and the website will display the home page, the user will make a register if he's accessing website for the first time and the website will store the information of users in database through backend. Second, the user can login to website and the backend will request database to check user information to verify his login. Then, user will enter cryptocurrencies prediction page to check and see predictions and the website will request predictions from API which connects to Models and the API will return predictions to the website. Finally, the user can see the predictions to coins he interested to invest on it.

**All these diagrams provide an overall description of our system.**

## 3.5 ERD Diagram

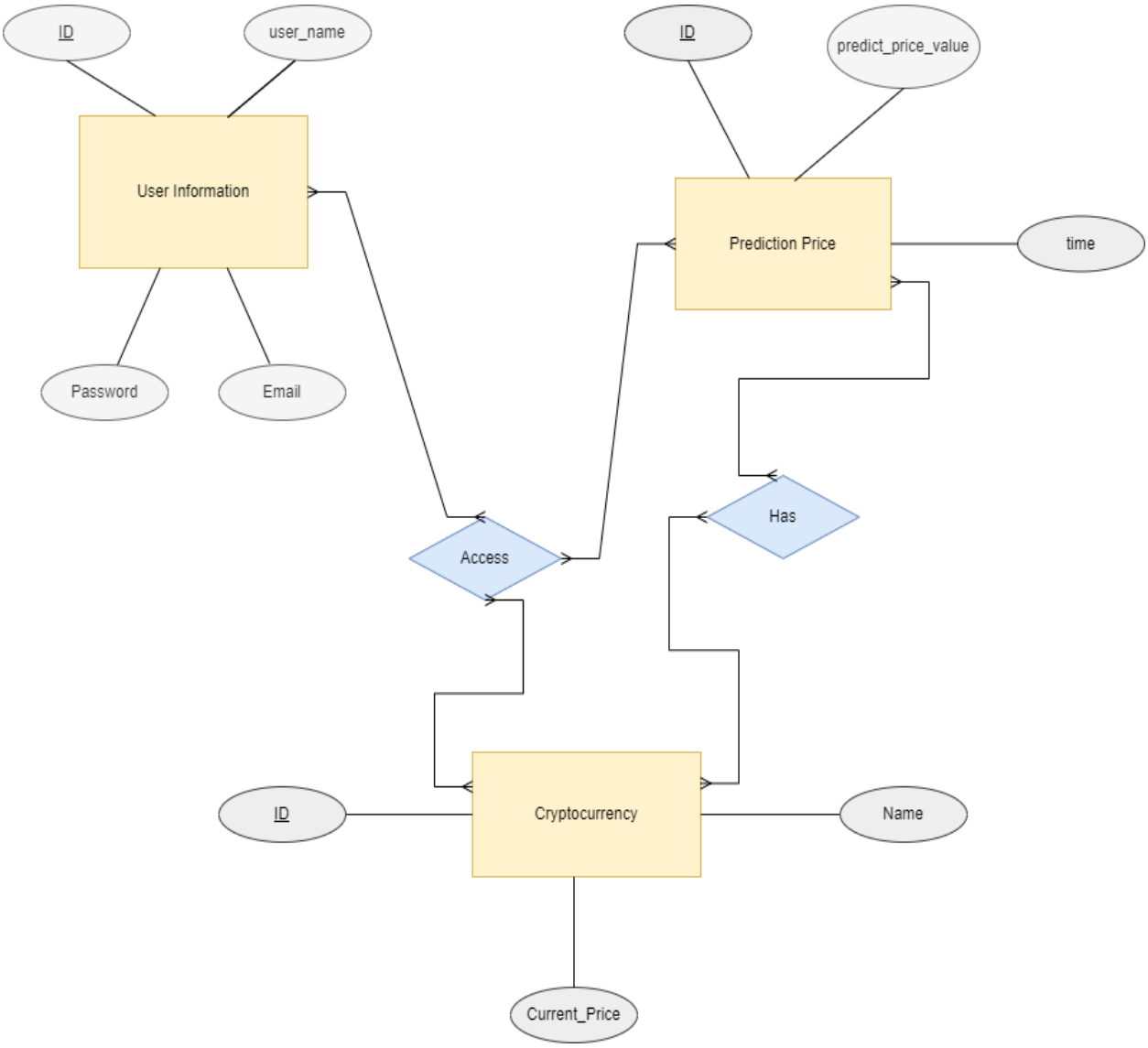



Figure 3.5

## Our ERD:-

We create 3 entities: User Information, Prediction and Cryptocurrency

Each entity of these entities has attributes

The attributes of User Information Entity are:-

- ID
- Name
- Password

- Email

The attributes of Prediction Entity are:-

- predict_price_value
- ID
- Time

The attributes of Cryptocurrency Entity are:-

- ID
- Name
- Current_Price

# Chapter 4

# Implementation

# 4.1 Data preprocessing

Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task. Typical pre-processing step that is widely adopted to enhance learning is data normalisation.

## Normalization:

Normalization is a data preprocessing technique used to transform features in a dataset to a common scale, improving the performance and accuracy of machine learning algorithms. The main goal of normalization is to eliminate the potential biases and distortions caused by the different scales of features. Some common normalization methods include min-max scaling, z-score standardization, and log transformation. And we use min-max scaler in our project.

## MinMax Scaler:

MinMax scaler is a way of data scaling, where the minimum of feature is made equal to zero and the maximum of feature equal to one. MinMax Scaler shrinks the data within the given range, usually of 0 to 1. It transforms data by scaling features to a given range. It scales the values to a specific value range without changing the shape of the original distribution.

## Why do you Use Min-Max Scaling?

Min-Max Scaling is useful when different parts of a design have different ranges or units of measurement. It puts features on the same size so that one feature doesn't stand out more than the others during machine learning training.

## Data preprocessing with MinMaxScaler for historical data :

MinMaxScaler normalizes numeric features to a predefined range, typically 0 to 1, which is essential for fair treatment of features of different scales. In cryptocurrency analysis, MinMaxScaler aligns price data across Bitcoin, Litecoin, Ethereum and Monero to fit the requirements of the models used, while maintaining intrinsic relationships.
MinMaxScaler transforms the data, ensuring each feature aligns within the specified range. This independent scaling of cryptocurrency prices mitigates biases during model training, enhancing predictive accuracy.

## Code:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
```

```
btc[['Close']]=scaler.fit_transform(btc[['Close']])
btcs=btc['Close']
```

## Training and Testing Splitting:

We performed experiments on each dataset/crypto separately, with the following methodology that was the same for all deep learning models. We performed an initial temporal training- test split on each dataset. The first 80% of the data belonged to the training set and the last 20% of the data belonged to the test set.

## Code:

Splitting data in machine learning models:

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Splitting data in deep learning models:

```
def split(size, obs):
    x = []
    y = []

    for i in range(len(obs)-SEQUENCE_SIZE):
        window = obs[i:(i+SEQUENCE_SIZE)]
        after_window = obs[i+SEQUENCE_SIZE]
        window = [[x] for x in window]
        x.append(window)
        y.append(after_window)

    return np.array(x),np.array(y)
```

```
# Split Data........

btc_train_size = int(len(btc_close)*.8 )
btc_train=btc_close[:btc_train_size]
btc_test=btc_close[btc_train_size:]
```

```
SEQUENCE_SIZE=30
btcx_train,btcy_train = split(SEQUENCE_SIZE,btc_train)
btcx_test,btcy_test = split(SEQUENCE_SIZE,btc_test)
```

```
print("Shape of training set: {}".format(btcx_train.shape))
print("Shape of test set: {}".format(btcx_test.shape))
```

```
Shape of training set: (1953, 30, 1, 1)
Shape of test set: (466, 30, 1, 1)
```

## 4.2Comparison Between Single and Multiple Feature

| **Technique** | Single (univariate) Feature Prediction | Multiple ( multivariate) features Prediction |
|---|---|---|
| **Definition** | properties of a single feature are described for a numerous sequence of time periods (time series). The analysis concentrates on the property change of that feature during the whole period of time. a single phenomenon characterized with a single variable (close) is processed in this analysis. | properties of multiple features for different intervals of time are analyzed. This approach allows comparison in change behavior between features. the change analysis can concern the simultaneous changes of several phenomena expressed by multiple variables (Open, Low, High, Close). |
| **Used Models** | Arima, SVR, LSTM and GRU | LSTM and GRU |

## 4.3What Is ARIMA?

ARIMA is a statistical model used to analyze temporal data to predict future values. The model analyzes the past behavior of data to identify trends and patterns, and can be useful in forecasting many phenomena such as industrial production, weather and prices.

### The idea of using ARIMA and how it works?

- The use of artificial intelligence in predicting cryptocurrency prices is based on the analysis of historical data and changing market factors. Mathematical models and machine learning techniques such as artificial neural networks or random trees can be used to analyze this data and generate predictions about future cryptocurrency price movement.
- Temporal Analysis: Temporal data is collected and analyzed to understand trends and patterns.
- Identification of components: Data for the three main components of the ARIMA method are analyzed:
- The first component is Auto regression (AR), which concerns the relationship between previous values and current values.
- The second component is the integral, which relates the time contract to the existing time series by taking the difference between the values.
- The third component is the moving average, which relates to average changes in time tension.
- Determine parameters: Parameters are determined for each creature in the ARIMA method. This includes choosing the autocorrelation (AR), the degree of differentiation (I), and the degree of moving average (MA).
- Model fit: The ARIMA model is customized using the available data and the parameters that have been specified.
- Mean Model Evaluation: Model performance is evaluated using objective criteria such as global error (mean squared error) or correlation (correlation).
- Forecasting: After the specific model is selected, it can be used to make predictions of future future values.
- ARIMA is a powerful tool for analyzing and forecasting temporal data, and can be used in a variety of fields such as finance, forecasting, weather, and industrial production.

## Why did we use the ARIMA model?

The ARIMA model is used in data prediction analysis in cryptocurrencies for several reasons:

- Temporal data: ARIMA relies on analyzing temporal data, and time is an essential element in estimating cryptocurrency forecasts.
- Relative stability: The cryptocurrency market is characterized by high volatility and rapid changes, but ARIMA can be used to identify general trends and the time pattern behind these fluctuations.
- Future Expectations: ARIMA gives an idea of how prices will move in the future, enabling investors and traders to make decisions based on their expectations of the market.
- Simplicity: The ARIMA model is simple and easy to understand, which makes it popular among traders and analysts.

In short, the ARIMA model is used in data forecast analysis in cryptocurrencies to provide insight into future price trends and help traders in making decisions.

**Pros and Cons of ARIMA:**

### Pros:

- Good for short-term forecasting.
- Only needs historical data.
- Models non-stationary data.

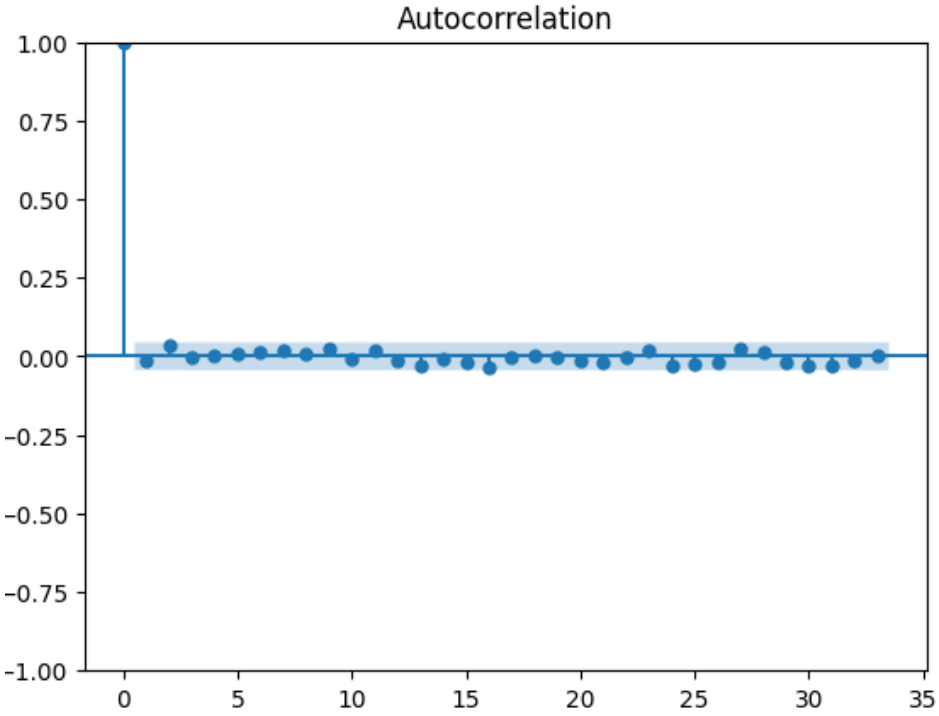
### Cons:

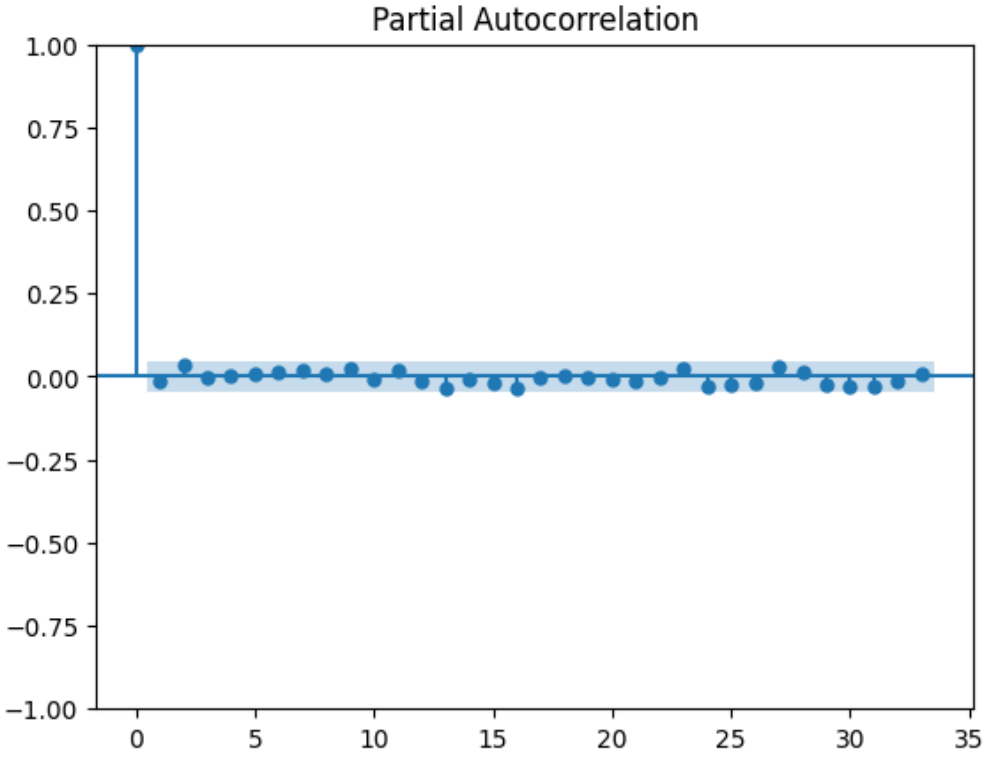
- Not built for long-term forecasting.
- Poor at predicting turning points.
- Computationally expensive.
- Parameters are subjective.

## Snippets of code
## Importing Libraries:

```
import warnings
import statsmodels.api as sm
import matplotlib.pyplot as plt
import itertools
from pandas.core import series
%matplotlib inline
```

```
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
acf_original = plot_acf (btcy_train)
pacf_original = plot_pacf(btcy_train)
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35

1-Imports: The first line imports two specific functions plot_acf and plot_pacf from the statsmodels.graphics.tsaplots module. plot_acf is used to plot the Autocorrelation Function (ACF), while plot_pacf is used to plot the Partial Autocorrelation Function (PACF).

2- Plotting ACF: The second line calls plot_acf(btcy_train). This function calculates and plots the autocorrelation for the given time series. Autocorrelation is a measure of the correlation between a series and a lagged version of itself. It helps identify whether there's a repetitive pattern or trend within the data at specific time lags. A plot is produced, typically displaying lags along the x-axis and the correlation values on the y-axis. It helps to identify patterns like seasonality or other forms of repeated behavior in the data.

3- Plotting PACF: The third line calls plot_pacf(btcy_train). This function calculates and plots the partial autocorrelation for the given time series. Partial autocorrelation is similar to autocorrelation, but it removes the indirect effects of intervening lags. This plot helps to isolate the direct relationship between the series at specific lags. The resulting plot has a similar structure to the ACF plot, with lags on the x-axis and the correlation values on the y-axis, but it provides more precise information about direct relationships between data points at different lags.

```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller( btcy_train )
print ( f'p - value : { adf_test[1]} ')
```

```
p - value : 0.0
```

- **Import Statement:**
  - The first line imports the adfuller function from the statsmodels.tsa.stattools module. The adfuller function is used to perform the Augmented Dickey-Fuller (ADF) test, a statistical test commonly used to check if a time series is stationary.

- **Performing the ADF Test:**
  - This function takes btcy_train as input and returns several statistics related to the Augmented Dickey-Fuller test. The primary purpose of this test is to determine whether a given time series has a unit root (a sign of non-stationarity).

- **The ADF Test and Stationarity:**
  - The ADF test is commonly used to test for stationarity in time series data. A stationary series has statistical properties that do not change over time (like constant mean, variance, and covariance). Non-stationary data can be problematic for time series analysis and forecasting, making it necessary to test and potentially transform data to achieve stationarity.

- **Extracting the p-Value:**

- The third line extracts the p-value from the test result (adf_test[1]). The p-value is a key output from the ADF test, indicating the probability of observing the data if the null hypothesis were true.
- In the ADF test, the null hypothesis is that the series has a unit root (i.e., it is non-stationary), and the alternative hypothesis is that it is stationary. A low p-value suggests that you can reject the null hypothesis, indicating that the series is likely stationary. Conversely, a high p-value indicates that there's insufficient evidence to reject the null hypothesis, suggesting non-stationarity.

```
from statsmodels.tsa.arima.model import ARIMA
model_btc = ARIMA(btcy_train, order=(2,1,0))
model_fit_btc=model_btc.fit()
print(model_fit_btc.summary())
```

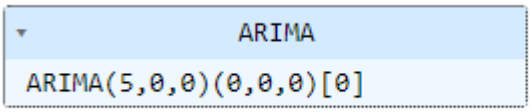
The first line imports the ARIMA class from the statsmodels.tsa.arima.model module.
The second line initializes an ARIMA model with the specified order (2, 1, 0).
ARIMA(btcy_train, order=(2, 1, 0)) creates an ARIMA model using the btcy_train as the time series input, with the given order of (p, d, q):

- p (Autoregressive part): This specifies the number of autoregressive terms. In this case, p=2, meaning the model uses the last two observations to predict the next one.
- d (Integrated part): This represents the number of times the data needs to be differenced to achieve stationarity. In this case, d=1, indicating that the data is differenced once.
- q (Moving Average part): This represents the number of moving average terms. Here, q=0, indicating no moving average component is used.

By specifying these parameters, the ARIMA model structure is defined, allowing for the fitting process to start.

**Fitting the ARIMA Model:** The third line calls fit() on the initialized ARIMA model object (model_btc) to fit the model to the data. The fitting process involves estimating the coefficients and other parameters of the model, aiming to minimize the error between the model's predictions and the actual data. This step generally takes some computational time, depending on the size of the data and the model complexity. It yields a model with fitted parameters that can be used for analysis, forecasting, or further validation.

```
import pmdarima as pm
auto_arima = pm.auto_arima( btcx_train , stepwise = False , seasonal =
False )
auto_arima
```




Using auto_arima: The second line creates an instance of the auto_arima function and assigns it to the variable auto_arima. The auto_arima function is designed to find the best ARIMA model for the given data by exploring different combinations of model parameters and choosing the one with the best performance. The key parameters passed to auto_arima are: btcx_train: the training data, a time series to which you're applying the ARIMA modeling technique.stepwise = False: This parameter indicates that the search for the best ARIMA model should not follow a stepwise approach. Stepwise typically means that the search progresses by adding or removing terms one at a time to optimize the model. seasonal = False: This indicates that the time series is not expected to have a seasonal component, so the search focuses only on non-seasonal ARIMA models.

# The Summary of ARIMA:

# 1-BTC:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 1735 |
| **Model:** | SARIMAX(5, 0, 0) | **Log Likelihood** | -125.360 |
| **Date:** | Wed, 17 Apr 2024 | **AIC** | 262.721 |
| **Time:** | 10:09:55 | **BIC** | 295.473 |
| **Sample:** | 0 - 1735 | **HQIC** | 274.833 |

**Covariance Type:** opg

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.1832 | 0.024 | 7.590 | 0.000 | 0.136 | 0.231 |
| **ar.L2** | 0.1826 | 0.024 | 7.663 | 0.000 | 0.136 | 0.229 |
| **ar.L3** | 0.1294 | 0.026 | 5.023 | 0.000 | 0.079 | 0.180 |
| **ar.L4** | 0.1898 | 0.024 | 7.893 | 0.000 | 0.143 | 0.237 |
| **ar.L5** | 0.1762 | 0.026 | 6.898 | 0.000 | 0.126 | 0.226 |
| **sigma2** | 0.0676 | 0.002 | 29.105 | 0.000 | 0.063 | 0.072 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 3.88 | **Jarque-Bera (JB):** | 193.19 |
| **Prob(Q):** | 0.05 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 0.97 | **Skew:** | 0.82 |
| **Prob(H) (two-sided):** | 0.72 | **Kurtosis:** | 3.03 |

# 2-LTC:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 2320 |
| **Model:** | SARIMAX(5, 0, 0) | **Log Likelihood** | 631.448 |
| **Date:** | Wed, 17 Apr 2024 | **AIC** | -1250.895 |
| **Time:** | 10:12:27 | **BIC** | -1216.399 |
| **Sample:** | 0 - 2320 | **HQIC** | -1238.324 |

**Covariance Type:** opg

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.1508 | 0.020 | 7.526 | 0.000 | 0.112 | 0.190 |
| **ar.L2** | 0.1639 | 0.020 | 8.135 | 0.000 | 0.124 | 0.203 |
| **ar.L3** | 0.1629 | 0.021 | 7.736 | 0.000 | 0.122 | 0.204 |
| **ar.L4** | 0.1872 | 0.020 | 9.354 | 0.000 | 0.148 | 0.226 |
| **ar.L5** | 0.1824 | 0.020 | 9.172 | 0.000 | 0.143 | 0.221 |
| **sigma2** | 0.0340 | 0.001 | 42.814 | 0.000 | 0.032 | 0.036 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 5.72 | **Jarque-Bera (JB):** | 718.72 |
| **Prob(Q):** | 0.02 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.03 | **Skew:** | 1.11 |
| **Prob(H) (two-sided):** | 0.68 | **Kurtosis:** | 4.59 |

# 3-ETH:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 1501 |
| **Model:** | SARIMAX(5, 0, 0) | **Log Likelihood** | -101.938 |
| **Date:** | Wed, 17 Apr 2024 | **AIC** | 215.877 |
| **Time:** | 10:11:07 | **BIC** | 247.760 |
| **Sample:** | 0 - 1501 | **HQIC** | 227.754 |

**Covariance Type:** opg

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.1417 | 0.027 | 5.161 | 0.000 | 0.088 | 0.195 |
| **ar.L2** | 0.1839 | 0.027 | 6.912 | 0.000 | 0.132 | 0.236 |
| **ar.L3** | 0.1928 | 0.026 | 7.427 | 0.000 | 0.142 | 0.244 |
| **ar.L4** | 0.1595 | 0.027 | 5.821 | 0.000 | 0.106 | 0.213 |
| **ar.L5** | 0.1587 | 0.027 | 5.931 | 0.000 | 0.106 | 0.211 |
| **sigma2** | 0.0670 | 0.002 | 27.559 | 0.000 | 0.062 | 0.072 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 2.51 | **Jarque-Bera (JB):** | 160.60 |
| **Prob(Q):** | 0.11 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.07 | **Skew:** | 0.80 |
| **Prob(H) (two-sided):** | 0.44 | **Kurtosis:** | 3.11 |

# 4-XMR:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 1516 |
| **Model:** | SARIMAX(1, 0, 2) | **Log Likelihood** | 431.015 |
| **Date:** | Wed, 17 Apr 2024 | **AIC** | -854.031 |
| **Time:** | 10:13:34 | **BIC** | -832.736 |
| **Sample:** | 0 - 1516 | **HQIC** | -846.102 |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.9997 | 0.000 | 2945.984 | 0.000 | 0.999 | 1.000 |
| **ma.L1** | -1.0045 | 0.026 | -38.570 | 0.000 | -1.055 | -0.953 |
| **ma.L2** | 0.0193 | 0.026 | 0.733 | 0.464 | -0.032 | 0.071 |
| **sigma2** | 0.0331 | 0.001 | 28.784 | 0.000 | 0.031 | 0.035 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.01 | **Jarque-Bera (JB):** | 301.61 |
| **Prob(Q):** | 0.91 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 0.91 | **Skew:** | 0.97 |
| **Prob(H) (two-sided):** | 0.31 | **Kurtosis:** | 4.01 |

## 4.4What is SVR?

Support Vector Regression (SVR). Built on support vector machines for classification, SVR enables both linear and non-linear regression. SVR is a non- parametric methodology and aims to maximize generalization performance when designing regression functions. SVR was applied to a variety of time series tasks such as forecasting warranty claims, predicting blood glucose levels.

## The Idea Behind Support Vector Regression:

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points. SVR seeks to find a hyperplane that best fits the data points in a continuous space. This is achieved by mapping the input variables to a high-dimensional feature space and finding the hyperplane that maximizes the margin (distance) between the hyperplane and the closest data points, while also minimizing the prediction error. The basic objective of the SVR is to find the function which has the most ε deviation from the actual target obtained from all training data, and at the same time the function must be as flat as possible. In other words, the error does not matter, as long as the error is less than epsilon ε.

## The Architecture of an SVR:

**Kernels**: SVR can use different types of kernels, which are functions that determine the similarity between input vectors. A linear kernel is a simple dot product between two input vectors, while a non-linear kernel is a more complex function that can capture more intricate

patterns in the data. The choice of kernel depends on the data's characteristics and the task's complexity.

Major Kernel Functions:

- Linear Kernel: It is used when data is linearly separable.
- Polynomial Kernel: It represents the similarity of vectors in the training set of data in a feature space over polynomials of the original variables used in the kernel.
- Sigmoid Kernel: this function is equivalent to a two-layer, perceptron model of the neural network, which is used as an activation function for artificial neurons.
- Radial Basis Function (RBF) Kernel: The RBF kernel function for two points and computes the similarity or how close they are to each other.

And we choose RBF kernel among the above kernels.

## Why do we choose RBF?

RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution and its flexibility. Also RBF kernel is one of the best kernels which helps us to introduce non-linearity in our model which means we can create Nonlinear Boundaries to classify data.

**Hyper parameters:** SVR has several hyper parameters that you can adjust to control the behavior of the model. For example, the "C" parameter controls the trade-off between the insensitive loss and the sensitive loss. A larger value of "C" means that the model will try to minimize the insensitive loss more, while a smaller value of C means that the model will be more lenient in allowing larger errors.

**Loss Function:** SVR typically uses an epsilon-insensitive loss function that allows for some errors within a defined range (epsilon), and it penalizes errors outside this range more heavily.

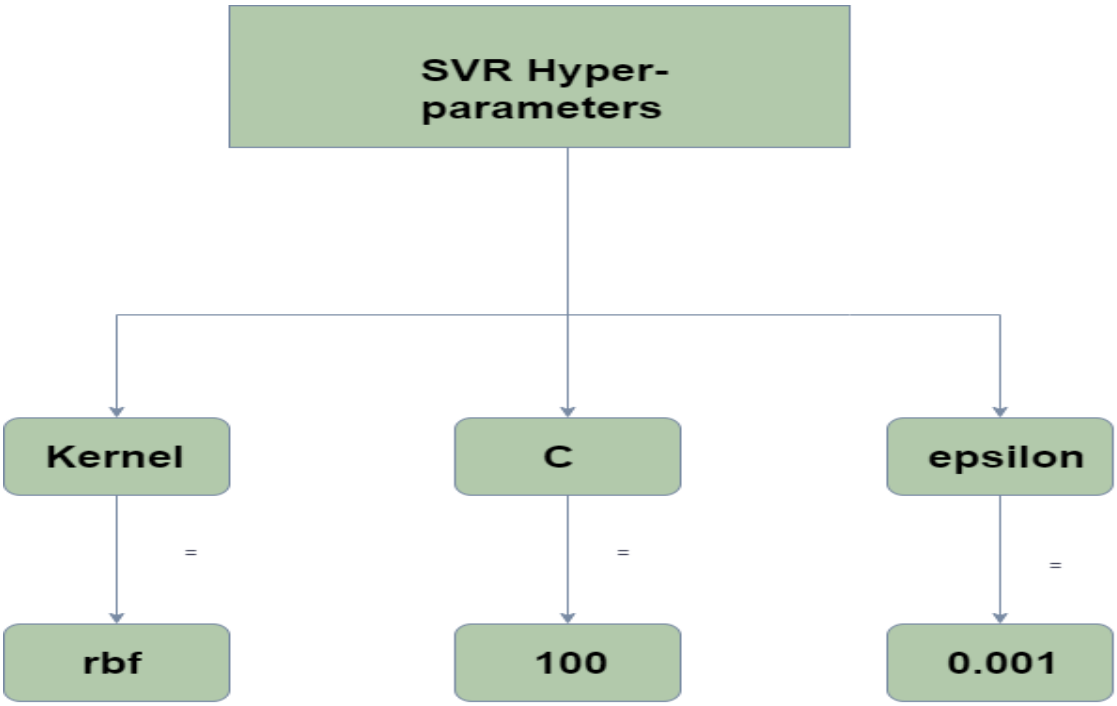



Figure 4.4

## Advantages of Support Vector Regression:

Although Support Vector Regression carries certain advantages that are as mentioned below:

1. Effective in high dimensional spaces: SVR is effective in high dimensional spaces, making it suitable for complex problems with many variables.
2. Robust to outliers: SVR is less sensitive to outliers compared to other regression techniques, making it suitable for datasets with noisy or inconsistent data.
3. Versatile kernel options: SVR allows for the use of different kernel functions, such as linear, polynomial, and radial basis function (RBF), providing flexibility in modeling various types of relationships in the data.
4. Decision model can be easily updated.
5. It has excellent generalization capability, with high prediction accuracy.
6. Its implementation is easy.

# Disadvantages of Support Vector Regression:

Some of the drawbacks faced by Support Vector Machines while handling regression problems are as mentioned below:

1. Complex parameter tuning: SVR requires careful selection of parameters, such as the choice of kernel and regularization parameter, which can be challenging and time-consuming.
2. Memory intensive: SVR can be memory intensive, especially when dealing with large datasets, as it needs to store support vectors and their associated weights.
3. Lack of interpretability: The resulting SVR model may be less interpretable compared to simpler regression models, making it difficult to understand the underlying relationships between variables.
4. They are not suitable for large datasets.
5. In cases where the number of features for each data point exceeds the number of training data samples, the SVR will underperform.
6. The Decision model does not perform very well when the data set has more noise.

## Application of SVR:

- Finance: Predicting stock prices, forecasting market trends, and risk management.
- Text Analysis: Sentiment analysis, document classification, and natural language processing.
- Image and Signal Processing: Object tracking, noise reduction, and signal reconstruction.
- Predictive Maintenance: Anticipating equipment failures, optimizing maintenance schedules, and reducing downtime.

## Why do we use SVR?

Support Vector Regression (SVR) can be a useful tool for cryptocurrency price prediction for several reasons:

- **Non-linearity:** Cryptocurrency price data often exhibit non-linear patterns and complexities. SVR can capture these non-linear relationships between input features (such as historical price data, trading volume, market sentiment, etc.) and output (future price) more effectively than traditional linear regression models.

- **Robustness to outliers:** Cryptocurrency markets can be highly volatile, leading to outliers in the data. SVR is less sensitive to outliers compared to some other regression techniques, making it more robust in handling noisy data.

- **Flexibility:** SVR allows for the use of different kernel functions, such as linear, polynomial, or radial basis function (RBF) kernels, which can capture different types of patterns in the data. This flexibility enables SVR to adapt to various market conditions and price trends observed in cryptocurrency markets.

- **Ability to handle high-dimensional data**: Cryptocurrency price prediction often involves considering multiple input features, such as historical prices, trading volumes, social media sentiment, etc. SVR can handle high-dimensional data efficiently, making it suitable for cryptocurrency price prediction models that incorporate a wide range of features.

- **Generalization:** SVR has excellent generalization capability, with high prediction accuracy. SVR aims to find the optimal hyperplane that maximizes the margin between different classes or, in the case of regression, maximizes the margin between the hyperplane and the data points. This emphasis on maximizing margins can lead to better generalization performance, helping to avoid overfitting and making SVR models more robust when applied to unseen data.

Overall, SVR's ability to capture non-linear relationships, robustness to outliers, flexibility in kernel selection, capability to handle high-dimensional data, and emphasis on generalization make it a useful tool for cryptocurrency price prediction tasks. However, like any predictive modeling technique, its effectiveness also depends on the quality of the data, feature selection, and model tuning.

# The Architecture and Fitting SVR:

## Importing Libraries:

```
#Import Libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import r2_score
from matplotlib import pyplot as plt
```

```
SVRModel = SVR(C = 100 ,epsilon=0.001,kernel = 'rbf') # it also can be :
linear, poly, sigmoid, precomputed
SVRModel.fit(X_train, y_train)
```

```
SVR
SVR(C=100, epsilon=0.001)
```

An instance of the SVR model is created and assigned to the variable SVRModel. The model is initialized with three important parameters:

- C: is the regularization parameter, which controls the trade-off between achieving a low training error and a low complexity model. A higher value of C allows for a more flexible model but may lead to overfitting.
- Epsilon: is the margin of tolerance around the predicted value. It specifies the size of the epsilon-tube within which no penalty is associated with the training points.
- Kernel: specifies the type of kernel function used in the SVR model. In this case, 'rbf' stands for radial basis function, which is a popular choice for non-linear regression problems. Other possible kernel options are 'linear', 'poly', 'sigmoid', or 'precomputed'.

It then trains the SVR model using the fit method. It takes the input features X_train and the corresponding target values y_train as arguments. The model learns from the provided training data to establish the relationships between the input features and the target values.

After executing these two lines of code, SVRModel will be a trained SVR model ready for making predictions on new, unseen data using the predict method.

## 4.5What is LSTM?

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It is designed specifically to overcome the long-term dependency problem faced by recurrent neural networks RNNs (due to the vanishing gradient problem), making it ideal for sequence prediction tasks. A cell state contains information learned in the previous and current time steps. This type of network cell comprises three gates: input gate, output gate, and forget gate, which at every discrete time step add information to the memory cell or remove information from it. Examples of LSTM usage can be found in short-term travel speed prediction, predicting healthcare trajectories from medical records, and forecasting aquifer levels. The model has also been successful for crypto price prediction.

## The Logic Behind LSTM:

The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp. This one cycle of LSTM is considered a single-time step.
These three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or LSTM cell. The first gate is called Forget gate, the second gate is known as the Input gate, and the last one is the Output gate. An LSTM unit that consists of these three gates and a memory cell or LSTM cell can be considered as a layer of neurons in traditional feedforward neural network, with each neuron having a hidden layer and a current state.

## The Architecture of an LSTM:

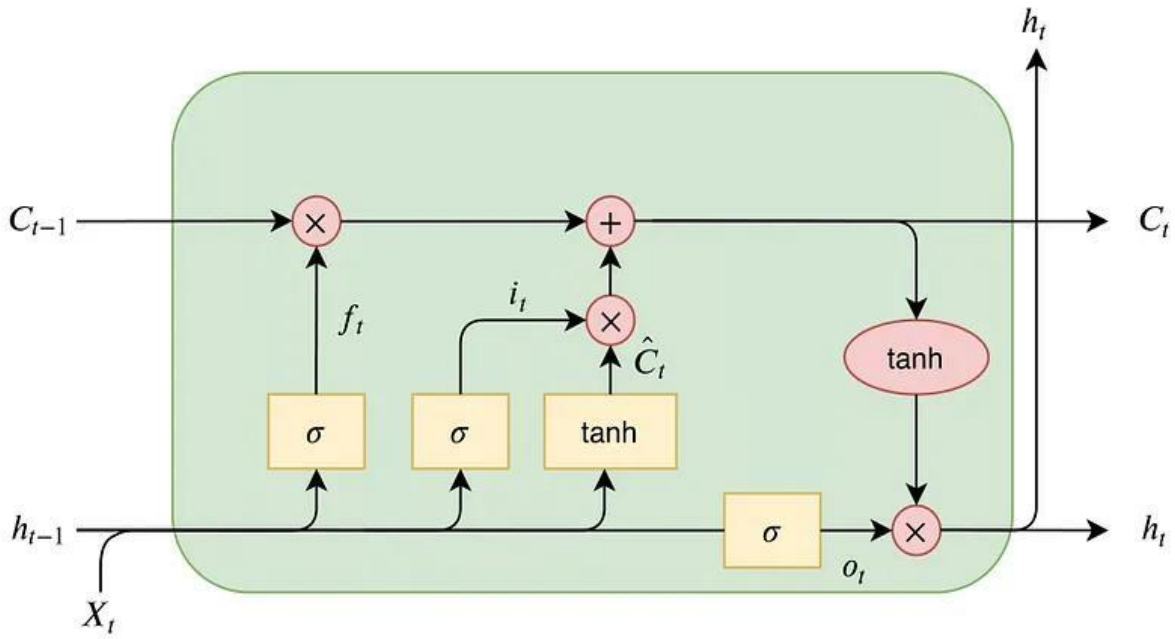



Figure 4.5.1

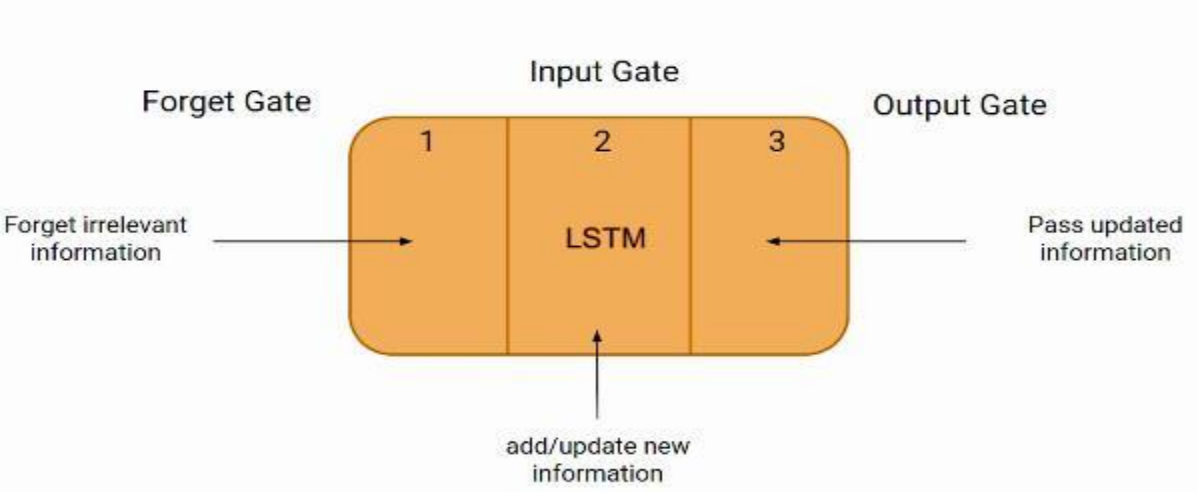



Figure 4.5.2

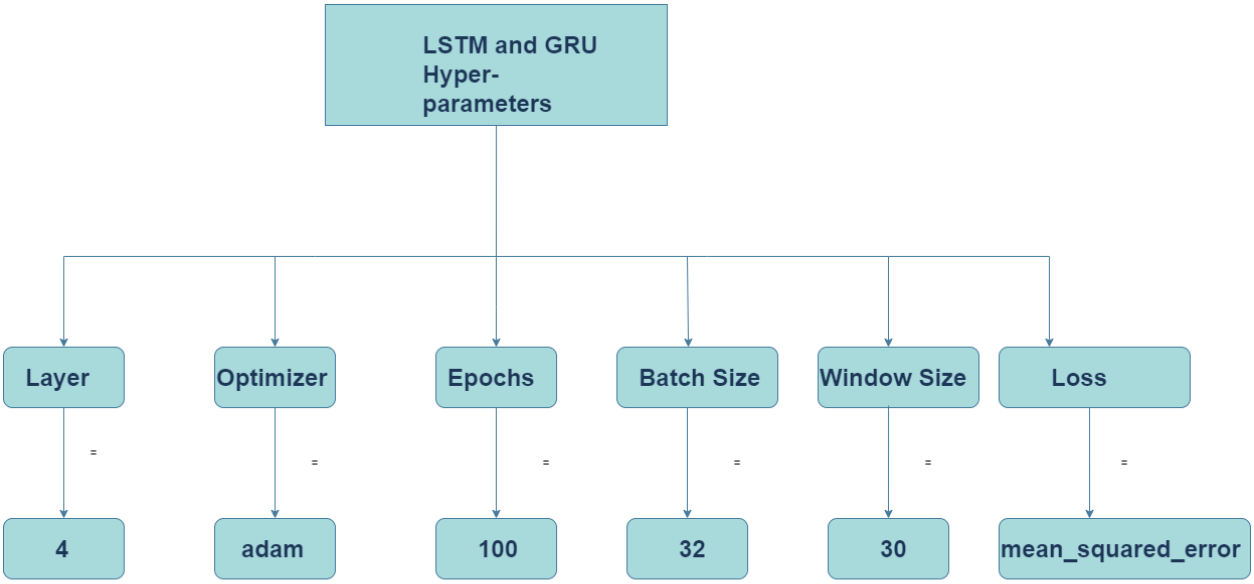



Figure 4.5.3

The LSTM architecture is based on the following key components:
1-Forget Gate: The forget gate determines what information from the previous cell state should be retained and what should be discarded. It allows the LSTM to "forget" irrelevant information.

2-Input Gate: The input gate controls the flow of information into the cell state. It can learn to accept or reject incoming data.

3-Output Gate: The output gate controls the information that is used to produce the output at each time step. It decides what part of the cell state should be revealed to the external world.

4-Hidden State: The hidden state serves as an intermediary between the cell state and the external world. It can selectively remember or forget information from the cell state and produce the output.

5-Cell State: This represents the memory of the LSTM and can store information over long sequences. It can be updated, cleared, or read from at each time step.

Working of an LSTM recurrent unit:
1. Take input the current input, the previous hidden state, and the previous internal cell state.
2. Calculate the values of the four different gates by following the below steps:

- For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
- Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

3. Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then add the two vectors.
4. Calculate the current hidden state by first taking the element-wise hyperbolic tangent of the current internal cell state vector and then performing element-wise multiplication with the output gate.

## The advantages of LSTM:

Long-term dependencies can be captured by LSTM networks. They have a memory cell that is capable of long-term information storage.
In traditional RNNs, there is a problem of vanishing and exploding gradients when models are trained over long sequences. By using a gating mechanism that selectively recalls or forgets information, LSTM networks deal with this problem.
LSTM enables the model to capture and remember the important context, even when there is a significant time gap between relevant events in the sequence. So where understanding context is important, LSTMS are used.

## The disadvantages of LSTM:

Compared to simpler architectures like feed-forward neural networks LSTM networks are computationally more expensive. This can limit their scalability for large-scale datasets or constrained environments.
Training LSTM networks can be more time-consuming compared to simpler models due to their computational complexity. So training LSTMs often requires more data and longer training times to achieve high performance.
Since it is processed word by word in a sequential manner, it is hard to parallelize the work of processing the sentences.

## Applications of LSTM:

Some of the famous applications of LSTM includes:

- **Language Modeling**: LSTMs have been used for natural language processing tasks such as language modeling, machine translation, and text summarization. They can be trained to generate coherent and grammatically correct sentences by learning the dependencies between words in a sentence.

- **Speech Recognition**: LSTMs have been used for speech recognition tasks such as transcribing speech to text and recognizing spoken commands. They can be trained to recognize patterns in speech and match them to the corresponding text.
- **Time Series Forecasting**: LSTMs have been used for time series forecasting tasks such as predicting stock prices, weather, and energy consumption. They can learn patterns in time series data and use them to make predictions about future events.
- **Anomaly Detection**: LSTMs have been used for anomaly detection tasks such as detecting fraud and network intrusion. They can be trained to identify patterns in data that deviate from the norm and flag them as potential anomalies.
- **Recommender Systems**: LSTMs have been used for recommendation tasks such as recommending movies, music, and books. They can learn patterns in user behavior and use them to make personalized recommendations.
- **Video Analysis**: LSTMs have been used for video analysis tasks such as object detection, activity recognition, and action classification. They can be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs), to analyze video data and extract useful information.

## Why do we use LSTM?

LSTM networks are often used in cryptocurrency price prediction because they are well-suited for handling sequences of data, such as historical price data. Cryptocurrency prices exhibit complex patterns and dependencies over time, making them ideal for LSTM's ability to capture long-term dependencies as we discussed before and remember important past information while training. Additionally, LSTM can handle non-linear relationships and adapt to changing market conditions, making them well-suited for cryptocurrency price prediction tasks. Also it can handle input features with different time scales, such as price, volume, and market sentiment, allowing for more comprehensive modeling of the cryptocurrency market dynamics.

### Snippets of code
### Importing Libraries:

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense
from keras.models import load_model
```

```
model=Sequential()
model.add(LSTM(units=30,return_sequences=True,input_shape=(btcx_train.shap
e[1],1)))
model.add(Dropout(0.2))
model.add(LSTM(units=60,return_sequences=True))
model.add(Dropout(0.2))
```

```
model.add(LSTM(units=90,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=120,return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam',loss='mean_squared_error')
model.summary()
```

Line 1: an instance of the Sequential model is created and assigned to the variable model. The Sequential model is a linear stack of layers, allowing for easy construction of deep learning models.

Line 2: adds an LSTM layer to the model. The units parameter specifies the number of memory units or neurons in the LSTM layer.
return_sequences=True indicates that the layer will return sequences as output instead of just the last output.
input_shape defines the shape of the input data, where btcx_train.shape[1] represents the number of time steps in the input sequence, and 1 represents the number of features.

Line 3: adds a Dropout layer to the model. Dropout is a regularization technique that randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting.

The subsequent lines follow a similar pattern of adding LSTM layers with increasing numbers of units and Dropout layers to introduce regularization. The additional LSTM layers allow the model to learn more complex patterns in the data.

Line 10: adds a dense layer with one unit to the model. The Dense layer is a fully connected layer, and in this case, it outputs a single value for the prediction.

Line 11: compiles the model by specifying the optimizer and the loss function to be used during training. In this case, the Adam optimizer is used, which is a popular optimizer for deep learning models. The mean squared error (MSE) loss function is used to measure the difference between the predicted and actual values.

Line 12: prints a summary of the model, displaying the architecture, the number of parameters, and other useful information.

Overall, the code constructs an LSTM model with multiple LSTM layers and Dropout layers for regularization. It is compiled with the Adam optimizer and MSE loss function for training on a time series prediction task.

```
model.fit(btcx_train,btcy_train,epochs=100,batch_size=32)
```

The provided code snippet demonstrates the training of the constructed LSTM model using the fit method.

- btcx_train: This parameter represents the input features of the training dataset. It should be a NumPy array or a TensorFlow tensor containing the input data.
- btcy_train: This parameter represents the target values or labels of the training dataset. It should also be a NumPy array or a TensorFlow tensor containing the target data.
- epochs=100: The epochs parameter specifies the number of times the model will iterate over the entire training dataset. Each epoch consists of forward and backward passes through the network to update the model's weights. In this case, the model will be trained for 100 epochs.
- batch_size=32: The batch_size parameter determines the number of samples that will be propagated through the network at once. It affects the speed of training and memory consumption. The training dataset will be divided into batches, and each batch will contain 32 samples.

During the training process, the model will iterate over the training dataset, processing the input features btcx_train and comparing the predicted values with the target values btcy_train. The model will adjust its weights using the optimization algorithm (in this case, Adam) and the specified loss function (mean squared error). The goal is to minimize the difference between the predicted values and the actual values.

By the end of training, the model's weights will be updated, and it will have learned to make predictions based on the input features. The trained model can then be used to make predictions on new, unseen data using the predict method.

# The Summary of LSTM single Feature Prediction Model:

# 1-BTC:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 30, 30)            3840

 dropout_4 (Dropout)         (None, 30, 30)            0

 lstm_5 (LSTM)               (None, 30, 60)            21840

 dropout_5 (Dropout)         (None, 30, 60)            0

 lstm_6 (LSTM)               (None, 30, 90)            54360

 dropout_6 (Dropout)         (None, 30, 90)            0

 lstm_7 (LSTM)               (None, 120)               101280

 dropout_7 (Dropout)         (None, 120)               0

 dense_1 (Dense)             (None, 1)                 121

=================================================================
Total params: 181441 (708.75 KB)
Trainable params: 181441 (708.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 2- LTC:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_8 (LSTM)               (None, 30, 30)            3840

 dropout_8 (Dropout)         (None, 30, 30)            0

 lstm_9 (LSTM)               (None, 30, 60)            21840

 dropout_9 (Dropout)         (None, 30, 60)            0

 lstm_10 (LSTM)              (None, 90)                54360

 dropout_10 (Dropout)        (None, 90)                0

 dense_2 (Dense)             (None, 1)                 91

=================================================================
Total params: 80131 (313.01 KB)
Trainable params: 80131 (313.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 3-ETH:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_11 (LSTM)              (None, 30, 30)            3840

 dropout_11 (Dropout)        (None, 30, 30)            0

 lstm_12 (LSTM)              (None, 30, 60)            21840

 dropout_12 (Dropout)        (None, 30, 60)            0

 lstm_13 (LSTM)              (None, 90)                54360

 dropout_13 (Dropout)        (None, 90)                0

 dense_3 (Dense)             (None, 1)                 91

=================================================================
Total params: 80131 (313.01 KB)
Trainable params: 80131 (313.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 4-XMR:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_14 (LSTM)              (None, 30, 30)            3840

 dropout_14 (Dropout)        (None, 30, 30)            0

 lstm_15 (LSTM)              (None, 30, 60)            21840

 dropout_15 (Dropout)        (None, 30, 60)            0

 lstm_16 (LSTM)              (None, 90)                54360

 dropout_16 (Dropout)        (None, 90)                0

 dense_4 (Dense)             (None, 1)                 91

=================================================================
Total params: 80131 (313.01 KB)
Trainable params: 80131 (313.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# LSTM Multi Feature Prediction Model:

## Snippets of code:

```
model=Sequential()
model.add(LSTM(units=30,return_sequences=True,input_shape=(btcx_train.shap
e[1],4)))
model.add(Dropout(0.2))
model.add(LSTM(units=60,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=90,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=120,return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam',loss='mean_squared_error')
model.summary()
```

- **Creating a Sequential Model** The code starts by initializing a Keras Sequential model, which is a linear stack of layers. This type of model allows you to add layers one-by-one, defining a straightforward feedforward structure.
- **Adding LSTM Layers** First LSTM Layer: This layer has 30 LSTM units (also known as cells), which are specialized to process sequential data and can capture temporal dependencies. input_shape=(btcx_train.shape[1], 4): This indicates that each input sample is a 2D tensor with a sequence length (btcx_train.shape[1]) and 4 features (Open, low, high and close). return_sequences=True: This specifies that the LSTM layer should return the entire sequence of outputs (one for each time step) to the next layer.
- **Dropout Layer:** model.add(Dropout(0.2)) This layer randomly drops out (sets to zero) 20% of the units during training, helping prevent overfitting by reducing reliance on specific features or neurons. Second LSTM Layer: model.add(LSTM(units=60, return_sequences=True)) This LSTM layer has 60 units and also returns sequences. Third LSTM Layer: model.add(LSTM(units=90, return_sequences=True)) This layer has 90 LSTM units and again returns sequences. Fourth LSTM Layer: model.add(LSTM(units=120, return_sequences=False)) This final LSTM layer has 120 units, but this time return_sequences=False, meaning it outputs only the last time step's result, suitable for a model's final LSTM layer.

**Dense Output Layer**

- model.add(Dense(units=1))
- This adds a dense (fully connected) layer with one output unit, indicating the model predicts a single value. This setup is typical for regression tasks.

**Compiling the Model**

- model.compile(optimizer='adam', loss='mean_squared_error')

- Compiling configures the model for training. Here, the optimizer is 'adam', a popular adaptive learning rate optimizer. The loss function is 'mean_squared_error', which is common for regression tasks.

```
model.fit(btcx_train,btcy_train,epochs=100,batch_size=32)
```

The fit method in Keras is used to train a model on a given dataset. It performs the training by iterating through the data multiple times and updating the model's weights based on the loss function and optimizer specified during compilation.
epochs=100: The epochs parameter specifies how many times the model should go through the entire training dataset during training. In this case, the model is set to train for 100 epochs, which is relatively high, indicating intensive training.
batch_size=32: The batch_size parameter indicates the number of samples used in each training step or batch. Instead of processing the entire dataset at once (which could be memory-intensive), the data is split into smaller batches of 32 samples. This also helps generalize the model better by averaging gradient updates over multiple samples.

# The Summary of LSTM Multi Feature Prediction Model:

# 1-BTC:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 30)            4200

 dropout (Dropout)           (None, 30, 30)            0

 lstm_1 (LSTM)               (None, 30, 60)            21840

 dropout_1 (Dropout)         (None, 30, 60)            0

 lstm_2 (LSTM)               (None, 30, 90)            54360

 dropout_2 (Dropout)         (None, 30, 90)            0

 lstm_3 (LSTM)               (None, 120)               101280

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 181801 (710.16 KB)
Trainable params: 181801 (710.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## 2- LTC:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 30, 30)            4200

 dropout_4 (Dropout)         (None, 30, 30)            0

 lstm_5 (LSTM)               (None, 30, 60)            21840

 dropout_5 (Dropout)         (None, 30, 60)            0

 lstm_6 (LSTM)               (None, 90)                54360

 dropout_6 (Dropout)         (None, 90)                0

 dense_1 (Dense)             (None, 1)                 91

=================================================================
Total params: 80491 (314.42 KB)
Trainable params: 80491 (314.42 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## 3-ETH:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_7 (LSTM)               (None, 30, 30)            4200

 dropout_7 (Dropout)         (None, 30, 30)            0

 lstm_8 (LSTM)               (None, 30, 60)            21840

 dropout_8 (Dropout)         (None, 30, 60)            0

 lstm_9 (LSTM)               (None, 90)                54360

 dropout_9 (Dropout)         (None, 90)                0

 dense_2 (Dense)             (None, 1)                 91

=================================================================
Total params: 80491 (314.42 KB)
Trainable params: 80491 (314.42 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 4-XMR:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_10 (LSTM)              (None, 30, 30)            4200

 dropout_10 (Dropout)        (None, 30, 30)            0

 lstm_11 (LSTM)              (None, 30, 60)            21840

 dropout_11 (Dropout)        (None, 30, 60)            0

 lstm_12 (LSTM)              (None, 90)                54360

 dropout_12 (Dropout)        (None, 90)                0

 dense_3 (Dense)             (None, 1)                 91

=================================================================
Total params: 80491 (314.42 KB)
Trainable params: 80491 (314.42 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## 4.6What is GRU?

The gated recurrent unit (GRU) is a specialized variant of recurrent Neural Network (RNN) developed to tackle the limitations of conventional RNNs, such as the vanishing gradient problem. It was introduced in the year 2014. GRUs have been successful in various applications, including natural language processing, speech recognition, and time series prediction.

## The Logic and Idea Behind GRU:

The GRU presents itself as an innovative solution to the vanishing gradient problem in traditional RNNs. It incorporates gating mechanisms that enable selective information update and resetting in the hidden state. This mechanism empowers the GRU to retain essential information and forget irrelevant data, facilitating the learning of long-term dependencies. The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate. The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU is calculated based on the updated hidden state.

## The Architecture of a GRU:

The architecture of the gated recurrent unit (GRU) is designed with two specific gates - the update gate and the reset gate. Each gate serves a unique purpose, significantly contributing to

the GRU's high efficiency. The reset gate identifies short-term relationships, while the update gate recognizes long-term connections.

The GRU architecture can be illustrated as:

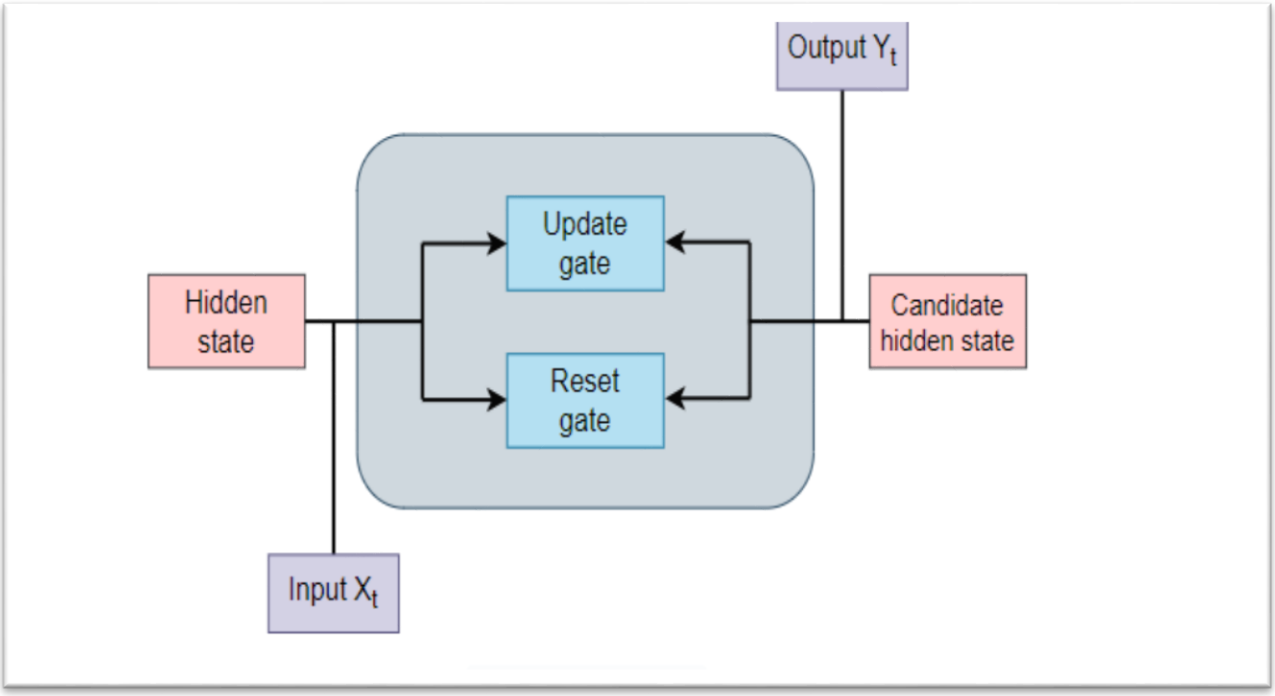



Figure 4.6

**The various components of the architecture are:**

- **Update gate:** The computation of the update gate is the first step in a GRU. It Determines the degree of past information forwarded to the future. It employs the current input and the previous hidden state to decide how much the previous hidden state should be updated. The sigmoid function is used here. It is analogous to the Output Gate in an LSTM recurrent unit.

- **Reset gate:** Decides the amount of past information to discard. The reset gate calculation, similar to the update gate, uses the sigmoid function. It identifies the volume of past information to be discarded. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.

- **Candidate hidden state:** Creates new representations, considering both the input and the past hidden state. After the reset gate computation, a candidate hidden state is computed employing the hyperbolic tangent function (tanh). The value of the reset gate determines the influence of the previous hidden state.

- **Final hidden state:** A blend of the new and old memories governed by the update gate. The final hidden state (also known as the current hidden state) is subsequently defined through linear interpolation. This interpolation involves the previous hidden state and the prospective hidden state, and the update gate influences its degree.

Working of a Gated Recurrent Unit:

- Take input the current input and the previous hidden state as vectors.
- Calculate the values of the three different gates by following the steps given below:

1. For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
2. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

- The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.
- To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

## The advantages of GRU:

- GRU networks are similar to Long Short-Term Memory (LSTM) networks, but with fewer parameters, making them computationally less expensive and faster to train.
- GRU networks can handle long-term dependencies in sequential data by selectively remembering and forgetting previous inputs.
- GRU networks have been shown to perform well on a variety of tasks, including natural language processing, speech recognition, and music generation.

- GRU networks can be used for both sequence-to-sequence and sequence classification tasks.

## The disadvantages of GRU:

- GRU networks may not perform as well as LSTMs on tasks that require modeling very long-term dependencies or complex sequential patterns.
- GRU networks may be more prone to overfitting than LSTMs, especially on smaller datasets.
- GRU networks require careful tuning of hyper parameters, such as the number of hidden units and learning rate, to achieve good performance.
- GRU networks may not be as interpretable as other machine learning models, since the gating mechanism can make it difficult to understand how the network is making predictions.

## Applications of GRU:

GRUs find diverse applications in various domains, including:

- Natural language processing: GRUs excel in tasks such as language modeling, machine translation, sentiment analysis, and text generation.
- Speech recognition: GRUs play a vital role in automatic speech recognition systems for sequence-to-sequence modeling.
- Time series prediction: GRUs are effective in forecasting tasks like stock price prediction, weather forecasting, and demand prediction.

## Why do we use GRU?

Using Gated Recurrent Units (GRUs) in a cryptocurrency price prediction project can be beneficial because GRUs are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data while overcoming some of the limitations of traditional RNNs like vanishing gradients.

Cryptocurrency prices are influenced by a multitude of factors, including market sentiment, trading volumes, news, and historical price patterns. GRUs can effectively learn and remember patterns in such data sequences, making them suitable for predicting cryptocurrency prices.

Additionally, GRUs are computationally efficient compared to other RNN architectures like Long Short-Term Memory (LSTM) networks, which makes them more practical for large-scale datasets like those found in cryptocurrency markets.

However, it's important to note that no single model or algorithm guarantees accurate predictions in the highly volatile and unpredictable cryptocurrency market. It's essential to combine GRUs with other techniques, such as feature engineering, ensemble methods, or external data sources, to build robust prediction models.

## Snippets of code
## Importing Libraries:

```
from keras.models import Sequential
from keras.layers import GRU
from keras.layers import Dropout
from keras.layers import Dense
from keras.models import load_model
from keras.initializers import Orthogonal
import datetime
import time
```

```
model=Sequential()
model.add(GRU(units=30,return_sequences=True,input_shape=(btcx_train.shape
[1],1)))
model.add(Dropout(0.2))
model.add(GRU(units=60,return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(units=90,return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(units=120,return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam',loss='mean_squared_error')
model.summary()
```

The provided code represents the construction and compilation of a Gated Recurrent Unit (GRU) neural network model for a time series prediction task.

Line 1: an instance of the Sequential model is created and assigned to the variable model. The Sequential model is a linear stack of layers, allowing for easy construction of deep learning models.

Line 2: adds a GRU layer to the model. The units parameter specifies the number of memory units or neurons in the GRU layer.
return_sequences=True indicates that the layer will return sequences as output instead of just the last output.
input_shape defines the shape of the input data, where btcx_train.shape[1] represents the number of time steps in the input sequence, and 1 represents the number of features.

Line 3: adds a Dropout layer to the model. Dropout is a regularization technique that randomly sets a fraction of input units to 0 at each update during training, which helps prevent overfitting.

The subsequent lines follow a similar pattern of adding GRU layers with increasing numbers of units and Dropout layers to introduce regularization. The additional GRU layers allow the model to learn more complex patterns in the data.

Line 10: adds a Dense layer with one unit to the model. The Dense layer is a fully connected layer, and in this case, it outputs a single value for the prediction.

Line 11: compiles the model by specifying the optimizer and the loss function to be used during training. In this case, the Adam optimizer is used, which is a popular optimizer for deep learning models. The mean squared error (MSE) loss function is used to measure the difference between the predicted and actual values.

Line 12: prints a summary of the model, displaying the architecture, the number of parameters, and other useful information.

Overall, the code constructs a GRU model with multiple GRU layers and Dropout layers for regularization. It is compiled with the Adam optimizer and MSE loss function for training on a time series prediction task.

```
model.fit(btcx_train,btcy_train,epochs=100,batch_size=32)
```

The provided code snippet demonstrates the training of the constructed GRU model using the fit method.

- btcx_train: This parameter represents the input features of the training dataset. It should be a NumPy array or a TensorFlow tensor containing the input data.
- btcy_train: This parameter represents the target values or labels of the training dataset. It should also be a NumPy array or a TensorFlow tensor containing the target data.
- epochs=100: The epochs parameter specifies the number of times the model will iterate over the entire training dataset. Each epoch consists of forward and backward passes through the network to update the model's weights. In this case, the model will be trained for 100 epochs.
- batch_size=32: The batch_size parameter determines the number of samples that will be propagated through the network at once. It affects the speed of training and memory consumption. The training dataset will be divided into batches, and each batch will contain 32 samples.

During the training process, the model will iterate over the training dataset, processing the input features btcx_train and comparing the predicted values with the target values btcy_train. The model will adjust its weights using the optimization algorithm (in this case, Adam) and the specified loss function (mean squared error). The goal is to minimize the difference between the predicted values and the actual values.

By the end of training, the model's weights will be updated, and it will have learned to make predictions based on the input features. The trained model can then be used to make predictions on new, unseen data using the predict method.

# The Summary of GRU single Feature Prediction Model:

# 1-BTC:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 30)            2970

 dropout (Dropout)           (None, 30, 30)            0

 gru_1 (GRU)                 (None, 30, 60)            16560

 dropout_1 (Dropout)         (None, 30, 60)            0

 gru_2 (GRU)                 (None, 30, 90)            41040

 dropout_2 (Dropout)         (None, 30, 90)            0

 gru_3 (GRU)                 (None, 120)               76320

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 137011 (535.20 KB)
Trainable params: 137011 (535.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## 2- LTC:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 30, 30)            2970

 dropout_4 (Dropout)         (None, 30, 30)            0

 gru_5 (GRU)                 (None, 30, 60)            16560

 dropout_5 (Dropout)         (None, 30, 60)            0

 gru_6 (GRU)                 (None, 90)                41040

 dropout_6 (Dropout)         (None, 90)                0

 dense_1 (Dense)             (None, 1)                 91

=================================================================
Total params: 60661 (236.96 KB)
Trainable params: 60661 (236.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## 3-ETH:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_7 (GRU)                 (None, 30, 30)            2970

 dropout_7 (Dropout)         (None, 30, 30)            0

 gru_8 (GRU)                 (None, 30, 60)            16560

 dropout_8 (Dropout)         (None, 30, 60)            0

 gru_9 (GRU)                 (None, 90)                41040

 dropout_9 (Dropout)         (None, 90)                0

 dense_2 (Dense)             (None, 1)                 91

=================================================================
Total params: 60661 (236.96 KB)
Trainable params: 60661 (236.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 4-XMR:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_10 (GRU)                (None, 30, 30)            2970

 dropout_10 (Dropout)        (None, 30, 30)            0

 gru_11 (GRU)                (None, 30, 60)            16560

 dropout_11 (Dropout)        (None, 30, 60)            0

 gru_12 (GRU)                (None, 90)                41040

 dropout_12 (Dropout)        (None, 90)                0

 dense_3 (Dense)             (None, 1)                 91

=================================================================
Total params: 60661 (236.96 KB)
Trainable params: 60661 (236.96 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# GRU Multi Feature Prediction Model:

## Snippets of code

```
model=Sequential()
model.add(GRU(units=30,return_sequences=True,input_shape=(btcx_train.shape
[1],4)))
model.add(Dropout(0.2))
model.add(GRU(units=60,return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(units=90,return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(units=120,return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam',loss='mean_squared_error')
model.summary()
```

1. **Sequential Model:** Sequential() is a way to create a linear stack of neural network layers in Keras. The layers are added in sequence.
2. **GRU Layer**: GRU stands for Gated Recurrent Unit, a type of recurrent neural network (RNN) that is particularly well-suited for sequence data like time series or text. units=30 defines the number of hidden units or neurons in the layer. return_sequences=True specifies that the layer should return the entire sequence, allowing subsequent layers to work with the full sequence data.
3. input_shape=(btcx_train.shape[1], 4) sets the input shape for the first layer, where the first dimension is the number of time steps, and the second is the number of features 4(open, low, high and).
4. **Dropout Layer:** Dropout(0.2) introduces dropout with a 20% dropout rate, meaning that during training, 20% of the units in the previous layer are randomly set to zero. This helps prevent overfitting by reducing reliance on any individual neuron.
5. **Adding GRU Layers with Increasing Units:** The model adds additional GRU layers with increasing units: 60, 90, and 120. return_sequences=True continues to return the entire sequence until the last GRU layer, where return_sequences=False is used to return only the last output. This is common when the goal is to predict a single output at the end of a sequence.
6. **Dense Layer:** Dense(units=1) is a fully connected layer with one output unit. This is commonly used for regression tasks where the model predicts a single continuous value.
7. **Model Compilation**: model.compile(optimizer='adam', loss='mean_squared_error') specifies the optimizer and loss function for training. The adam optimizer is a popular choice in deep learning, providing a good balance between speed and convergence.

mean_squared_error is a common loss function for regression, measuring the square of the difference between the predicted and actual values.

```
model.fit(btcx_train,btcy_train,epochs=100,batch_size=32)
```

1. model.fit: This method starts the training process for a Keras model. It takes several parameters to configure the training, such as the input data, labels (or target outputs), the number of training iterations (epochs), and the batch size.
2. epochs=100: This parameter specifies the number of times the entire dataset will be passed through the model during training. An epoch represents one complete iteration over the training data. With epochs=100, the model will learn from the dataset 100 times, refining its internal weights with each iteration.
3. batch_size=32: This parameter indicates the number of samples that will be processed together in one step during training. For example, if batch_size is 32, the training data is divided into batches of 32 samples each. This setting influences how the model's internal weights are updated, with smaller batch sizes leading to more updates but potentially noisier gradients, and larger batch sizes being computationally more efficient but possibly less generalizable.

## The Summary of GRU Multi Feature Prediction Model:

# 1-BTC:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 30)            3240

 dropout (Dropout)           (None, 30, 30)            0

 gru_1 (GRU)                 (None, 30, 60)            16560

 dropout_1 (Dropout)         (None, 30, 60)            0

 gru_2 (GRU)                 (None, 30, 90)            41040

 dropout_2 (Dropout)         (None, 30, 90)            0

 gru_3 (GRU)                 (None, 120)               76320

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 137281 (536.25 KB)
Trainable params: 137281 (536.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 2- LTC:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 30, 30)            3240

 dropout_4 (Dropout)         (None, 30, 30)            0

 gru_5 (GRU)                 (None, 30, 60)            16560

 dropout_5 (Dropout)         (None, 30, 60)            0

 gru_6 (GRU)                 (None, 90)                41040

 dropout_6 (Dropout)         (None, 90)                0

 dense_1 (Dense)             (None, 1)                 91

=================================================================
Total params: 60931 (238.01 KB)
Trainable params: 60931 (238.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 3-ETH:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_7 (GRU)                 (None, 30, 30)            3240

 dropout_7 (Dropout)         (None, 30, 30)            0

 gru_8 (GRU)                 (None, 30, 60)            16560

 dropout_8 (Dropout)         (None, 30, 60)            0

 gru_9 (GRU)                 (None, 90)                41040

 dropout_9 (Dropout)         (None, 90)                0

 dense_2 (Dense)             (None, 1)                 91

=================================================================
Total params: 60931 (238.01 KB)
Trainable params: 60931 (238.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 4-XMR:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_10 (GRU)                (None, 30, 30)            3240

 dropout_10 (Dropout)        (None, 30, 30)            0

 gru_11 (GRU)                (None, 30, 60)            16560

 dropout_11 (Dropout)        (None, 30, 60)            0

 gru_12 (GRU)                (None, 90)                41040

 dropout_12 (Dropout)        (None, 90)                0

 dense_3 (Dense)             (None, 1)                 91

=================================================================
Total params: 60931 (238.01 KB)
Trainable params: 60931 (238.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## What is the difference between LSTM and GRU?

The main difference between GRU and LSTM is the way they handle the memory cell state. In LSTM, the memory cell state is maintained separately from the hidden state and is updated using three gates: the input gate, output gate, and forget gate. In GRU, the memory cell state is replaced with a “candidate activation vector,” which is updated using two gates: the reset gate and update gate. LSTM has a cell state and gating mechanism which controls information flow, whereas GRU has a simpler single gate update mechanism. LSTM is more powerful but slower to train, while GRU is simpler and faster.

# 4.7Twitter Sentiment Analysis:

## Twitter:

Twitter is one such a major micro-blogging website, having over 100 million users generating over 500 million tweets every day. With such large audience, Twitter has consistently attracted users to convey their opinions and perspective about any issue, brand, company or any other topic of interest. Due to this reason, Twitter is used as an informative source by many organizations, institutions and companies. On Twitter, users are allowed to share their opinions in the form of tweets, using only 140 characters. This leads to people compacting their statements by using slang, abbreviations, emoticons, short forms etc. Along with this, people convey their opinions by using sarcasm and polysemy. Hence it is justified to term the Twitter language as unstructured.

## Sentiment Analysis:

Sentiment was measured by applying Valence Sentiment Analysis to the text of the cryptocurrency related tweets. Valence quantifies the degree of pleasure or displeasure of an emotional experience.
For the task we used valence aware dictionary and sentiment reasoner (VADER). A sentiment analysis method, specifically designed for social media context. VADER was created from a gold standard sentiment lexicon, that is valence-based and human curated. The result of applying VADER to a tweet text is a vector with a normalized value for the scores: positive sentiment, neutral sentiment, negative sentiment and compound sentiment.
Most work performed on sentiment analysis for financial markets focuses only on the dimensions of valence, mood or calmness, often overlooking the phenomenon of polarization of opinions. For this reason, in a similar fashion as done previously, we calculated a polarization score for each hour of data by applying the geometric mean of the average of the positive sentiment and the negative sentiment of all the tweets that are in the time-step with the intention of using the polarization score as a complementary dimension to emotional valence. Sentiment analysis, a facet of natural language processing, aims to discern and extract subjective information from textual data, unraveling the emotional tone within a string of words. When applied to Twitter, it becomes a powerful tool for gauging sentiments in tweets—categorizing them as positive, negative, or neutral. Its applications span diverse domains, from predicting stock prices and assessing cultural heritage scenarios to monitoring societal health, tracking public opinion on specific topics, and fostering cross-media conversations.
However, conducting sentiment analysis on Twitter poses distinct challenges inherent to the platform. Tweets, characterized by brevity, harbor informal elements like slang and abbreviations, are highly context-dependent, and may incorporate emojis and emoticons. Overcoming these hurdles necessitates the deployment of specialized sentiment analysis models meticulously trained on expansive and diverse Twitter-specific datasets. While our focus lies in exploring sentiment within the Twitter-sphere, this paper confines itself to leveraging well-established libraries tailored for sentiment analysis on this platform, recognizing that delving into novel techniques extends beyond its current scope.

## Sentiment Analysis and Cryptocurrencies:

Sentiment analysis plays an important role in decoding public perception and emotions surrounding cryptocurrencies. By analyzing social media, news, and online discussions, it gauges market sentiment, impacting trading decisions. Positive sentiment often drives price surges, while negative sentiment can trigger selloffs. This intersection of technology and behavioral analysis helps investors navigate the volatile landscape of cryptocurrency markets, enhancing decision-making strategies.

## Applications of Sentiment Analysis:

Social media monitoring: As we all know, social media is taking over the world. More than 55% of customers share their reviews about purchases socially on many social networking sites. It's almost difficult to analyze the reviews manually. Sentiment analysis lets us analyze and derive meaning from them.

Brand monitoring: Brand owners use sentiment analysis tools to keep track of the bad reviews about their brand. They can also use machine learning algorithms to predict outcomes based on the results derived using semantic analysis.

Voice of customer: Various sentiment analysis algorithms let us analyze the voice of the customers, such as the product that are most needed by the customers and also the products that are highly rated, etc. The brand owners can create a personalized customer experience based on these evaluations.

Customer service: Chatbots are a widespread way of delivering good customer service. Using sentiment analysis, you can transfer the chat to a customer service associate whenever needed. Also, you can automate the tasks such as booking a ticket, an appointment for a salon, etc.

Market research: Using sentiment analysis, you can research how well your competitors are growing and what are their positive feedbacks from the customers. You can also analyze the way they deal with their customers. You can, in turn, work on the issues related to your product's failure.

Product Analysis: You can do keyword research to identify the products in demand and the highly rated products. You can also determine what features of a particular product are highly appreciated by the customers or the end users.

# Why do we use Twitter Sentiment Analysis?

Twitter sentiment analysis can be useful in cryptocurrency price prediction for several reasons:

- Crypto Market's Sensitivity to Sentiment: The cryptocurrency market is highly speculative and influenced by public perception and sentiment. News, rumors, and social media trends can cause significant fluctuations in cryptocurrency prices. Analyzing

Twitter sentiment can provide insights into market trends, investor attitudes, and potential price movements.

- Real-Time Insights: Twitter is a fast-paced platform where users share real-time information, opinions, and news. This immediacy makes it a valuable source for tracking market sentiment as it evolves. Monitoring Twitter can help identify emerging trends or shifts in sentiment that may affect cryptocurrency prices.
- Identification of Influencers: Certain individuals or entities on Twitter, like Elon Musk or major cryptocurrency influencers, can have a significant impact on cryptocurrency prices. Sentiment analysis can help identify these key influencers and gauge their impact on market sentiment.
- Market Sentiment Analysis: By analyzing tweets related to specific cryptocurrencies, you can gauge overall market sentiment—whether it's positive, negative, or neutral. This can help predict price movements based on whether sentiment is improving or deteriorating.
- Correlation with Price Movements: Studies have shown that social media sentiment can correlate with cryptocurrency price movements. For example, an increase in positive sentiment might precede a price rally, while a surge in negative sentiment could predict a price drop.
- Event Detection and News Impact: Twitter sentiment analysis can help detect important events, news releases, or major announcements that could influence cryptocurrency prices. This allows traders to react quickly to market-moving news.
- Complementary Data Source: Twitter sentiment analysis can complement other analytical approaches, such as technical analysis, fundamental analysis, and on-chain analysis. It adds a social dimension to price prediction, providing a broader perspective on what might drive market trends.
- Automated Analysis for Large-Scale Data: With machine learning and natural language processing (NLP), it's possible to analyze large volumes of tweets and derive sentiment scores. This can be automated, allowing for continuous monitoring of sentiment trends.

## Snippets of code:
## Importing Libraries:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
```

## Preprocessing for tweets Data:

A tweet contains a lot of opinions about the data which are expressed in different ways by different users. The twitter dataset used in this survey work is already labeled into two classes viz. negative and positive polarity and thus the sentiment analysis of the data becomes easy to observe the effect of various features. The raw data having polarity is highly susceptible to inconsistency and redundancy. Preprocessing of tweet include following points,
Remove all URLs (e.g. www.xyz.com), hash tags (e.g. #topic), targets (@username).
Correct the spellings; sequence of repeated characters is to be handled.
Replace all the emoticons with their sentiment.
Remove all punctuations, symbols, numbers and marks.
Remove Stop Words.
Expand Acronyms (we can use a acronym dictionary).
Remove Non-English Tweets.
Remove Twitter handles from the text.
Remove Extra Spaces.
Remove Short words.
Remove everything other than text.

## Code:

```
import wordcloud

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
tweets = pd.read_csv('/content/Ethereum_tweets.csv')
tweets.head()
```

| | user_name | user_location | user_description | user_created | user_followers | user_friends | user_favourites | user_verified |
|---|---|---|---|---|---|---|---|---|
| 0 | #1 Crypto Currency TRADING | NaN | Profit thought trading, Learn , Trade & Earn$.... | 2021-11-10 10:42:01+00:00 | 4 | 0 | 0 | False |
| 1 | Reyrey | Kuala Lumpur | Artist \| NFT Broker & Collector for AnideaNFT ... | 2011-03-07 07:22:57+00:00 | 343 | 650 | 589 | False |
| 2 | akter jahan | NaN | NaN | 2021-10-28 13:28:05+00:00 | 1 | 2 | 0 | False |
| 3 | riyan | NaN | i always paricipated your project | 2021-09-16 16:59:11+00:00 | 11 | 127 | 99 | False |
| 4 | Michelle Eriksen | NaN | Art Creator, NFT, Crypto | 2021-08-11 06:40:08+00:00 | 522 | 1428 | 16 | False |

| date | text | hashtags | source | is_retweet |
|---|---|---|---|---|
| 2021-11-15 07:26:39+00:00 | Get upto 1000 that's 10x with our premium spot... | NaN | Twitter for Android | False |
| 2021-11-15 07:26:35+00:00 | Adopted a Hypocat #053, "Crazy Rich Cat" 🤑💵💵💵😃... | ['SupportingCreators', 'CryptoArt', 'CryptoArt... | Twitter for iPhone | False |
| 2021-11-15 07:26:32+00:00 | Top 10 Coins by Social Engagement over the las... | ['LunarCrush', 'bitcoin', 'dogecoin'] | Twitter Web App | False |
| 2021-11-15 07:26:30+00:00 | Requesting faucet funds into 0x5278942b39deD8c... | ['Rinkeby', 'Ethereum'] | Twitter Web App | False |
| 2021-11-15 07:26:30+00:00 | New Listing BTS 0.008ETH !! \n#NFTs #nftcolle... | ['NFTs', 'nftcollector', 'NFTCommunity', 'art'... | Twitter Web App | False |

**clean tweets:**

```
# Apostrophe Dictionary
apostrophe_dict = {
"ain't": "am not / are not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is",
"i'd": "I had / I would",
"i'd've": "I would have",
"i'll": "I shall / I will",
"i'll've": "I shall have / I will have",
"i'm": "I am",
"i've": "I have",
"isn't": "is not",
"it'd": "it had / it would",
"it'd've": "it would have",
"it'll": "it shall / it will",
"it'll've": "it shall have / it will have",
"it's": "it has / it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
```

```
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she had / she would",
"she'd've": "she would have",
"she'll": "she shall / she will",
"she'll've": "she shall have / she will have",
"she's": "she has / she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as / so is",
"that'd": "that would / that had",
"that'd've": "that would have",
"that's": "that has / that is",
"there'd": "there had / there would",
"there'd've": "there would have",
"there's": "there has / there is",
"they'd": "they had / they would",
"they'd've": "they would have",
"they'll": "they shall / they will",
"they'll've": "they shall have / they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we had / we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what shall / what will",
"what'll've": "what shall have / what will have",
```

```
"what're": "what are",
"what's": "what has / what is",
"what've": "what have",
"when's": "when has / when is",
"when've": "when have",
"where'd": "where did",
"where's": "where has / where is",
"where've": "where have",
"who'll": "who shall / who will",
"who'll've": "who shall have / who will have",
"who's": "who has / who is",
"who've": "who have",
"why's": "why has / why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you had / you would",
"you'd've": "you would have",
"you'll": "you shall / you will",
"you'll've": "you shall have / you will have",
"you're": "you are",
"you've": "you have"
}

def contx_to_exp(text):
  for key in apostrophe_dict:
    value = apostrophe_dict[key]
    text = text.replace(key, value)
    return text
```

```
# Emotion detection by different symbols
emotion_dict = {
":)": "happy",
```

```
":-)": "happy",
":-]": "happy",
":-3": "happy",
":->": "happy",
"8-)": "happy",
":-}": "happy",
":o)": "happy",
":c)": "happy",
":^)": "happy",
"=]": "happy",
"=)": "happy",
"<3": "happy",
":-(": "sad",
":(": "sad",
":c": "sad",
":<": "sad",
":[": "sad",
">:[": "sad",
":{": "sad",
">:(": "sad",
":-c": "sad",
":-< ": "sad",
":-[": "sad",
":-||": "sad"
}

def emotion_check(text):
  for key in emotion_dict:
    value = emotion_dict[key]
    text = text.replace(key, value)
    return text
```

```
def clean_text(text):
  text = re.sub(r'https?:\/\/\S*'," ", text) # Removing the url from the
text
  text = re.sub(r'@\S+', " ", text) # Removing twitter handles from the
text
  text = re.sub('#'," ", text) # removing # from the data
  text = re.sub(r'RT', "", text) # Removing the Re-tweet mark
  text = re.sub(r"\s+"," ", text)  # Removing Extra Spaces
  text = text.lower()
  return text
```

```
#removes pattern in the input text
import re
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt.lower()
```

```
#removing the twitter handles @user
tweets['clean_tweet'] = np.vectorize(remove_pattern)(tweets['text'],
"@[\w]*")

#using above functions
tweets['clean_tweet'] = tweets['clean_tweet'].apply(lambda x :
clean_text(x))
tweets['clean_tweet'] = tweets['clean_tweet'].apply(lambda x :
contx_to_exp(x))
tweets['clean_tweet'] = tweets['clean_tweet'].apply(lambda x :
emotion_check(x))

#removing special characters, numbers and punctuations
tweets['clean_tweet'] = tweets['clean_tweet'].str.replace("[^a-zA-Z]", "
")


#remove short words
tweets['clean_tweet'] = tweets['clean_tweet'].apply(lambda x: " ".join([w
for w in x.split() if len(w)>3]))

# Removing every thing other than text
tweets['clean_tweet'] = tweets['clean_tweet'].apply( lambda x:
re.sub(r'[^\w\s]',' ',x))  # Replacing Punctuations with space
tweets['clean_tweet'] = tweets['clean_tweet'].apply( lambda x:
re.sub(r'[^a-zA-Z]', ' ', x)) # Raplacing all the things with space other
than text
tweets['clean_tweet'] = tweets['clean_tweet'].apply( lambda x:
re.sub(r"\s+"," ", x)) # Removing extra spaces


#individual words as tokens
tokenized_tweet = tweets['clean_tweet'].apply(lambda x: x.split())
```

```
#stem the words

from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

tokenized_tweet = tokenized_tweet.apply(lambda sentence:
[lemmatizer.lemmatize(stemmer.stem(word)) for word in sentence])



#combine words into single sentence
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = " ".join(tokenized_tweet[i])



tweets['clean_tweet'] = tokenized_tweet
tweets.head()
```

| | user_name | user_location | user_description | user_created | user_followers | user_friends | user_favourites | user_verified |
|---|---|---|---|---|---|---|---|---|
| 0 | #1 Crypto Currency TRADING | NaN | Profit thought trading, Learn , Trade & Earn$.... | 2021-11-10 10:42:01+00:00 | 4 | 0 | 0 | False |
| 1 | Reyrey | Kuala Lumpur | Artist \| NFT Broker & Collector for AnideaNFT ... | 2011-03-07 07:22:57+00:00 | 343 | 650 | 589 | False |
| 2 | akter jahan | NaN | NaN | 2021-10-28 13:28:05+00:00 | 1 | 2 | 0 | False |
| 3 | riyan | NaN | i always paricipated your project | 2021-09-16 16:59:11+00:00 | 11 | 127 | 99 | False |
| 4 | Michelle Eriksen | NaN | Art Creator, NFT, Crypto | 2021-08-11 06:40:08+00:00 | 522 | 1428 | 16 | False |

| date | text | hashtags | source | is_retweet | clean_tweet |
|---|---|---|---|---|---|
| 2021-11-15 07:26:39+00:00 | Get upto 1000 that's 10x with our premium spot... | NaN | Twitter for Android | False | upto that s with premium spot trade signal acc... |
| 2021-11-15 07:26:35+00:00 | Adopted a Hypocat #053, "Crazy Rich Cat" 🤑💵💵💵😍... | ['SupportingCreators', 'CryptoArt', 'CryptoArt... | Twitter for iPhone | False | adopt hypocat crazi rich cat meowww supporting... |
| 2021-11-15 07:26:32+00:00 | Top 10 Coins by Social Engagement over the las... | ['LunarCrush', 'bitcoin', 'dogecoin'] | Twitter Web App | False | coin social engag over last lunarcrush btc bit... |
| 2021-11-15 07:26:30+00:00 | Requesting faucet funds into 0x5278942b39deD8c... | ['Rinkeby', 'Ethereum'] | Twitter Web App | False | request faucet fund into x b ded cbdad e a ec ... |
| 2021-11-15 07:26:30+00:00 | New Listing BTS 0.008ETH !! \n#NFTs #nftcolle... | ['NFTs', 'nftcollector', 'NFTCommunity', 'art'... | Twitter Web App | False | list eth nft nftcollector nftcommun nftdrop op... |

## Polarity and Subjectivity:

In TextBlob analysis, polarity and subjectivity are two key measures that provide insights into the sentiment and nature of a piece of text, such as a sentence or document. In summary, polarity gauges the sentiment expressed in a text, ranging from positive to negative, while subjectivity measures how subjective or opinionated the text is, ranging from objective to subjective. These measures are valuable in sentiment analysis and can provide a quantitative understanding of the emotional tone and nature of textual content. In particular, they are defined as follows:

- Polarity is used to measure the sentiment expressed in a piece of text and ranges from −1 to 1, where −1 represents a completely negative sentiment, 0 indicates a neutral sentiment, and 1 signifies a completely positive sentiment.
- Subjectivity is used to measure how subjective or opinionated a piece of text is. It ranges from 0 to 1, where 0 indicates a highly objective or factual statement, and 1 suggests a highly subjective or opinionated statement. A higher subjectivity score implies a more opinionated or subjective piece of text. A lower subjectivity score indicates a more objective or factual statement.

## Snippets of code:

```
from textblob import TextBlob     # for performing NLP Functions i.e
detection of Polarity and Subjectivity

polarity=[]     #list that contains polarity of tweets
subjectivity=[]    ##list that contains subjectivity of tweets

for i in merge.clean_tweet.values:
    try:
        analysis = TextBlob(i) # [i] records to the first data in dataset
        polarity.append(analysis.sentiment.polarity)
        subjectivity.append(analysis.sentiment.subjectivity)

    except:
        polarity.append(0)
        subjectivity.append(0)



# adding sentiment polarity and subjectivity column to dataframe

merge['polarity'] = polarity
merge['subjectivity'] = subjectivity
merge.head()
```

| | Open | High | Low | Close | clean_tweet | polarity | subjectivity |
|---|---|---|---|---|---|---|---|
| 0 | 445.209015 | 472.609009 | 428.312012 | 466.540009 | upto that s with premium spot trade signal acc... | 0.000000 | 0.000000 |
| 1 | 466.851013 | 476.239014 | 456.653015 | 463.449005 | adopt hypocat crazi rich cat meowww supporting... | 0.375000 | 0.750000 |
| 2 | 463.704987 | 482.813995 | 451.851990 | 465.852997 | coin social engag over last lunarcrush btc bit... | 0.016667 | 0.066667 |
| 3 | 466.053986 | 474.777008 | 453.312012 | 470.204010 | request faucet fund into x b ded cbdad e a ec ... | 0.000000 | 0.000000 |
| 4 | 470.294006 | 473.558014 | 457.660004 | 463.281006 | list eth nft nftcollector nftcommun nftdrop op... | 0.000000 | 0.000000 |

1. Importing TextBlob: from textblob import TextBlob: This line imports the TextBlob library, which is a simple Natural Language Processing (NLP) tool for processing text data. It can be used to analyze sentiment, correct spelling, translate text, and more.
2. Initializing Lists for Polarity and Subjectivity polarity=[]: This initializes an empty list to store the polarity of each tweet. Polarity is a measure of how positive or negative a text is, typically ranging from -1 (most negative) to +1 (most positive). subjectivity=[]: This initializes an empty list to store the subjectivity of each tweet. Subjectivity indicates the degree to which a text is opinion-based (as opposed to fact-based), usually ranging from 0 (most objective) to 1 (most subjective).
3. Looping Through Tweets: for i in merge.clean_tweet.values: This loop iterates over the 'clean_tweet' column in the merge DataFrame. Each iteration processes one tweet, referred to as i.
4. Performing Sentiment Analysis with TextBlob: analysis = TextBlob(i): Creates a TextBlob object from the tweet text (i). TextBlob can then be used to analyze the sentiment of the text. polarity.append(analysis.sentiment.polarity): Extracts the polarity score from the TextBlob object and appends it to the polarity list. subjectivity.append(analysis.sentiment.subjectivity): Similarly, this line extracts the subjectivity score from the TextBlob object and appends it to the subjectivity list.
5. Handling Exceptions: try: ... except: This try-except block is used to handle potential errors when creating a TextBlob object. If an error occurs (e.g., if the tweet is empty or has unexpected characters), the code appends default values (0) to both the polarity and subjectivity lists. Adding Polarity and Subjectivity to the DataFrame merge['polarity'] = polarity: This line adds a new column named 'polarity' to the merge DataFrame, with the values from the polarity list. merge['subjectivity'] = subjectivity: Similarly, this line adds a 'subjectivity' column to the merge DataFrame, with the values from the subjectivity list.

## 4.7.1What is Vader?

Vader (Valence Aware Dictionary and sEntiment Reasoner) is a rule-based sentiment analysis tool that is specifically designed for analyzing social media texts. Vader is a pre-trained sentiment analysis model that provides a sentiment score for a given text. VADER is able to detect the polarity of sentiment (how positive or negative) of a given body of text when the data being analysed is unlabelled. In traditional sentiment analysis, the algorithm is given the opportunity to learn from the labelled training data. VADER results to assess its ability to classify tweets on a five-point scale. Phase four deals with the plotting of positivity, negativity, neutral, compound (overall) score of a tweet produced by the VADER. VADER not only tells about the positivity and negativity scores but also tells about how positive or negative a sentiment is.

**Advantages of VADER for Twitter Sentiment Analysis:**

- Pretrained and Specialized for Social Media: VADER is designed to work with social media texts, making it a good fit for analyzing tweets, which often contain informal language, slang, emojis, and special characters.
- Fast and Easy to Use: It doesn't require extensive training or additional data processing. VADER uses a rule-based approach with a lexicon of sentiment-related words and heuristic rules, making it computationally efficient and quick to implement.
- Handles Emojis and Punctuation: VADER is sensitive to capitalization, punctuation (like exclamation marks), and emojis, providing a more accurate reflection of sentiment in tweets, which often rely on these elements for emotional expression.
- Provides Multiple Sentiment Scores: It gives a compound score indicating overall sentiment and also provides positive, neutral, and negative sentiment scores. This can offer a nuanced view of the sentiment in a tweet.
- Good Performance in Short Texts: Twitter is a platform with a strict character limit, resulting in shorter texts. VADER is effective in handling and extracting sentiment from short texts.

**Disadvantages of VADER for Twitter Sentiment Analysis:**

- Limited Flexibility: VADER relies on a predefined lexicon and heuristic rules, which might not capture emerging slang, new words, or context-specific sentiment accurately. It is less flexible than machine learning-based approaches, which can adapt to changing language trends.
- Less Effective for Complex Language Structures: Tweets can sometimes contain sarcasm, irony, or complex structures that may not be correctly interpreted by VADER's rules-based approach. This can lead to misinterpretation of sentiment.
- No Contextual Understanding: Since VADER is rule-based, it does not understand the broader context in which a tweet is made. This can result in sentiment misjudgments, especially in ambiguous or context-dependent situations.
- Lexicon-Based Limitations: The predefined lexicon may not always cover domain-specific vocabulary or new jargon. This can lead to a lack of sensitivity in recognizing certain sentiments.

# Why do we use VADER?

- **Social Media-Optimized:** VADER is specifically designed to handle the informal language, slang, and emoticons often found in social media platforms like Twitter. This makes it particularly suitable for analyzing tweets, where text can be non-standard and context-specific.

- **Context-Sensitive:** VADER incorporates not just words, but also context, including punctuation, capitalization, and even emojis. This context awareness helps in accurately determining sentiment, a crucial aspect when analyzing informal social media text.
- **Valence Scores:** VADER provides a sentiment "polarity" score that ranges from -1 (most negative) to +1 (most positive), along with compound scores that indicate overall sentiment. This fine-grained scoring system allows for nuanced sentiment analysis, crucial for understanding the diverse range of opinions on Twitter.
- **Speed and Efficiency**: VADER is efficient and can quickly process large volumes of text, making it ideal for real-time sentiment analysis. This is particularly important in cryptocurrency markets, where rapid reactions to sentiment changes are often necessary.
- **Handling Sarcasm and Negations**: VADER is designed to handle negations and sarcasm to some extent, which is common in Twitter language. This ability to understand nuances in text enhances the accuracy of sentiment analysis.
- **Open Source and Readily Available:** VADER is open source and can be integrated with popular programming languages like Python. Its ease of use and compatibility with common data analysis tools make it accessible for a wide range of users, from individual traders to larger research teams.
- **Interpretability:** Unlike some more complex sentiment analysis models, VADER's approach is relatively interpretable. It relies on a lexicon-based method, meaning you can understand why a specific text received a certain sentiment score. This interpretability can be helpful when explaining sentiment-based predictions.
- **Real-Time Monitoring and Alerts:** VADER can be used in real-time monitoring systems, allowing for continuous tracking of cryptocurrency-related sentiment on Twitter. This capability is valuable for traders who need to react quickly to market-moving events.

## Snippets of code:

```
#Create a function to get the sentiment scores (using Sentiment Intensity
Analyzer)
def getSIA(text):
    sia = SentimentIntensityAnalyzer()
    sentiment = sia.polarity_scores(text)
    return sentiment
```

- Sentiment Intensity Analyzer (SIA): SentimentIntensityAnalyzer: This is part of the nltk.sentiment.vader module, a popular sentiment analysis tool in the Natural Language Toolkit (NLTK) library. VADER (Valence Aware Dictionary and sEntiment Reasoner) is designed to detect sentiment in text, including emotional context, emphasizing social media data. It uses a pre-defined lexicon of words and their associated sentiment scores, along with heuristics for analyzing text context, such as punctuation, capitalization, and intensity modifiers like "very."

- Function Definition: def getSIA(text): This line defines a new function named getSIA, which takes one argument text. This argument represents the text data for which you want to determine sentiment.
- Creating the Sentiment Analyzer Instance: sia = SentimentIntensityAnalyzer(): This line initializes a new instance of the SentimentIntensityAnalyzer. It will be used to compute sentiment scores for the given text.
- Calculating Sentiment Scores: sentiment = sia.polarity_scores(text): This line computes the sentiment scores for the provided text using the polarity_scores() method of the SentimentIntensityAnalyzer. The polarity_scores() method returns a dictionary with four sentiment scores: compound: A score ranging from -1 (most negative) to +1 (most positive), representing the overall sentiment of the text. pos: A score indicating the proportion of positive sentiment in the text. neu: A score indicating the proportion of neutral sentiment in the text. neg: A score indicating the proportion of negative sentiment in the text.
- Returning the Sentiment Scores: return sentiment: This line returns the dictionary containing the sentiment scores computed by the SentimentIntensityAnalyzer. This returned dictionary can be used to analyze the sentiment of the given text, determine the overall sentiment, or incorporate these scores into further analysis or visualizations.

```
#Get the sentiment scores
compound = []
neg = []
neu = []
pos = []
SIA = 0
for i in range(0, len(merge['clean_tweet'])):
    SIA = getSIA(merge['clean_tweet'][i])
    compound.append(SIA['compound'])
    neg.append(SIA['neg'])
    neu.append(SIA['neu'])
    pos.append(SIA['pos'])
```

```
#Store the sentiment scores in the data frame
merge['Compound'] =compound
merge['Negative'] =neg
merge['Neutral'] =neu
merge['Positive'] = pos
```

| | Open | High | Low | Close | clean_tweet |
|---|---|---|---|---|---|
| 0 | 445.209015 | 472.609009 | 428.312012 | 466.540009 | upto that s with premium spot trade signal acc... |
| 1 | 466.851013 | 476.239014 | 456.653015 | 463.449005 | adopt hypocat crazi rich cat meowww supporting... |
| 2 | 463.704987 | 482.813995 | 451.851990 | 465.852997 | coin social engag over last lunarcrush btc bit... |
| 3 | 466.053986 | 474.777008 | 453.312012 | 470.204010 | request faucet fund into x b ded cbdad e a ec ... |
| 4 | 470.294006 | 473.558014 | 457.660004 | 463.281006 | list eth nft nftcollector nftcommun nftdrop op... |

| polarity | subjectivity | Compound | Negative | Neutral | Positive |
|---|---|---|---|---|---|
| 0.000000 | 0.000000 | 0.0000 | 0.0 | 1.000 | 0.000 |
| 0.375000 | 0.750000 | 0.6486 | 0.0 | 0.762 | 0.238 |
| 0.016667 | 0.066667 | 0.0000 | 0.0 | 1.000 | 0.000 |
| 0.000000 | 0.000000 | 0.0000 | 0.0 | 1.000 | 0.000 |
| 0.000000 | 0.000000 | 0.0000 | 0.0 | 1.000 | 0.000 |

- Initializing Sentiment Lists: compound = [], neg = [], neu = [], pos = []: These lines initialize four empty lists to store the sentiment scores extracted from each tweet. compound: Will store the compound sentiment score, representing overall sentiment on a scale from -1 (most negative) to +1 (most positive). neg: Will store the negative sentiment score, representing the proportion of negative sentiment in the text. neu: Will store the neutral sentiment score, representing the proportion of neutral sentiment. pos: Will store the positive sentiment score, representing the proportion of positive sentiment.
- Loop Through Cleaned Tweets: for i in range(0, len(merge['clean_tweet'])):: This loop iterates over the index of the clean_tweet column in the merge DataFrame. It uses range(0, len(merge['clean_tweet'])) to iterate from 0 to the total number of tweets in the column. Each iteration i represents a specific tweet's index.
- Extracting Sentiment Scores: SIA = getSIA(merge['clean_tweet'][i]): This line retrieves the text of the tweet at index i from the clean_tweet column and passes it to the getSIA function to calculate the sentiment scores. The result is a dictionary containing the four sentiment scores. compound.append(SIA['compound']): Appends the compound score from the sentiment analysis to the compound list. neg.append(SIA['neg']): Appends the negative sentiment score to the neg list. neu.append(SIA['neu']): Appends the neutral sentiment score to the neu list. pos.append(SIA['pos']): Appends the positive sentiment score to the pos list.

## 4.7.2What is a Decision Tree?

- A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine learning algorithm, which is used for both classification and regression problems. It is one of the very powerful algorithms. And it is also used in Random Forest to train on different subsets of training data, which makes random forest one of the most powerful algorithms in machine learning.

## Advantages of Decision Tree:

- **Interpretability:** One of the significant advantages of decision trees is their interpretability. The decision rules learned by the algorithm are easy to understand and visualize, making it simple to explain to non-technical stakeholders. This transparency is valuable in domains where interpretability is crucial, such as finance or healthcare.
- **Non-linear relationships**: Decision trees can capture non-linear relationships between features and the target variable. Unlike linear models, decision trees can represent complex decision boundaries, making them suitable for datasets with intricate patterns.
- **No feature scaling required**: Decision trees are not sensitive to the scale of features, meaning there's no need for feature scaling (e.g., normalization or standardization) as required by some other algorithms like Support Vector Machines or K-Nearest Neighbors.
- **Handles both numerical and categorical data:** Decision trees can handle both numerical and categorical features without the need for one-hot encoding or other preprocessing techniques. This makes them convenient for datasets with mixed data types.
- **Robust to outliers**: Decision trees are robust to outliers in the data. Since they partition the feature space into regions based on the values of features, outliers tend to have minimal impact on the overall model performance.

## Disadvantages of Decision tree:

- **Overfitting:** Decision trees are prone to overfitting, especially when they grow too deep or when the dataset is noisy. Deep decision trees can memorize the training data, leading to poor generalization on unseen data. Techniques like pruning or limiting the tree depth can mitigate this issue.
- **High variance:** Decision trees have high variance, meaning small changes in the training data can result in significantly different trees. Ensemble methods like Random Forest or Gradient Boosting are often used to reduce variance and improve performance.
- **Instability**: Decision trees are sensitive to small variations in the data, which can lead to different splits and, consequently, different trees. This instability makes them less reliable compared to some other algorithms.
- **Bias towards features with many levels:** Features with a large number of levels (i.e., high cardinality) tend to be favored over features with fewer levels in decision tree splits.

This bias can affect the performance of the model, especially if the high-cardinality features are not truly informative.

- **Difficulty in capturing linear relationships**: Despite being able to capture non-linear relationships, decision trees struggle with capturing linear relationships between features and the target variable. Other algorithms like linear regression may perform better in such cases.

## Why do we use Decision tree?

- **Interpretability and Explainability:**
    1. Clear Structure: Decision trees provide a clear visual representation of how decisions are made, with each node representing a decision point based on a specific feature. This transparency is valuable in the often opaque world of machine learning, helping traders understand why a particular prediction is made.
    2. Insight into Key Factors: Decision trees highlight the most important features or decision points. This can be useful for cryptocurrency price prediction, where you might want to know which aspects of Twitter sentiment (e.g., specific keywords, frequency of tweets, sentiment scores) are most predictive of price movements.
- **Ability to Handle Diverse Data:**
    1. Combination of Features: Decision trees can process different types of data, including numerical, categorical, and binary. This flexibility is useful when integrating Twitter sentiment analysis with other sources of information (like technical indicators or trading volumes).
    2. Complex Decision-Making: The tree structure allows for complex decision rules, making it possible to consider multiple aspects of sentiment and other data simultaneously to make predictions.
- **Robustness to Nonlinear Relationships:**
    1. Nonlinear Patterns: Decision trees can model nonlinear relationships, which is critical in cryptocurrency markets where patterns are not always straightforward. By considering various combinations of features, a decision tree can capture these complex relationships.
    2. Segmented Decisions: Decision trees divide the dataset into segments, each with its own set of rules. This approach is useful in capturing unique market trends or behaviors that might be influenced by different sentiments or events.
- **Handling Outliers and Missing Data:**
    1. Outliers: Decision trees tend to be robust to outliers because they create decision boundaries based on splitting criteria. This is helpful in cryptocurrency markets, which can experience sudden spikes or drops in sentiment and prices.
    2. Missing Data: Decision trees can handle missing data reasonably well by following alternate branches. This flexibility is beneficial when dealing with incomplete Twitter sentiment data.

- **Integration with Ensemble Methods:**
    1. Boosting and Bagging: Decision trees can be used as building blocks for ensemble methods like Random Forests and Gradient Boosting Machines (GBM). These ensemble methods can improve prediction accuracy and stability, providing more robust cryptocurrency price prediction.
    2. Reducing Overfitting: Ensemble methods like Random Forests use multiple decision trees to reduce overfitting, leading to more generalized models. This is crucial when sentiment data might be noisy or biased.

## Snippets of code:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='entropy', max_depth=8,
                                                  min_samples_leaf=10,
                                                  min_samples_split=6,
                                                  random_state=42)
clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_leaf=10,
                       min_samples_split=6, random_state=42)
```

- Importing the Decision Tree Classifier: from sklearn.tree import DecisionTreeClassifier: This line imports the DecisionTreeClassifier class from the Scikit-learn library, which is a widely used library for machine learning in Python. Decision trees are a type of supervised learning algorithm used for classification tasks.
- Creating the Decision Tree Classifier: clf = DecisionTreeClassifier(...): This line creates an instance of the DecisionTreeClassifier with specific hyperparameters to control the tree's behavior. Let's examine these hyperparameters: criterion='entropy': The criterion parameter determines the function used to measure the quality of a split in the decision tree. 'entropy' refers to information gain, a common metric for selecting the best split. Other options include 'gini', which uses the Gini impurity measure. max_depth=8: The max_depth parameter sets the maximum depth of the tree, limiting how many levels the tree can have. A deeper tree might overfit the data, so setting a limit can help avoid overfitting. In this case, the tree will have at most 8 levels. min_samples_leaf=10: The min_samples_leaf parameter specifies the minimum number of samples (data points) required in a leaf node (a terminal node). Setting a higher value can help avoid overfitting by ensuring that the leaves have a certain level of representativity. min_samples_split=6: The min_samples_split parameter specifies the minimum number of samples required to split a node. If the number of samples in a node is less than this threshold, the node won't

be split. This setting also helps to prevent overfitting. random_state=42: The random_state parameter ensures reproducibility by setting the seed for random number generation. This makes the results consistent across different runs.

- Training the Classifier: clf.fit(x_train, y_train): This line trains the decision tree classifier (clf) on the given training data (x_train and y_train). The x_train variable represents the training features (input variables), while y_train represents the target labels (the class that you want to predict). The fit method builds the decision tree based on the training data and the specified hyperparameters, determining the optimal splits and structure of the tree.

## 4.7.3What is Linear Discriminant Analysis?

Linear Discriminant Analysis (LDA), also known as Normal Discriminant Analysis or Discriminant Function Analysis, is a dimensionality reduction technique primarily utilized in supervised classification problems. It facilitates the modeling of distinctions between groups, effectively separating two or more classes. LDA operates by projecting features from a higher-dimensional space into a lower-dimensional one. In machine learning, LDA serves as a supervised learning algorithm specifically designed for classification tasks, aiming to identify a linear combination of features that optimally segregates classes within a dataset. For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping. So, we will keep on increasing the number of features for proper classification.

### Advantages of using LDA:

- It is a simple and computationally efficient algorithm.
- It can work well even when the number of features is much larger than the number of training samples.
- It can handle multi collinearity (correlation between features) in the data.

### Disadvantages of LDA:

- It assumes that the data has a Gaussian distribution, which may not always be the case.
- It assumes that the covariance matrices of the different classes are equal, which may not be true in some datasets.
- It assumes that the data is linearly separable, which may not be the case for some datasets.
- It may not perform well in high-dimensional feature spaces.

### Applications of LDA:

Face Recognition: In the field of Computer Vision, face recognition is a very popular application in which each face is represented by a very large number of pixel values. Linear discriminant analysis (LDA) is used here to reduce the number of features to a more manageable number

before the process of classification. Each of the new dimensions generated is a linear combination of pixel values, which form a template. The linear combinations obtained using Fisher's linear discriminant are called Fisher's faces.
Medical: In this field, Linear discriminant analysis (LDA) is used to classify the patient's disease state as mild, moderate, or severe based on the patient's various parameters and the medical treatment he is going through. This helps the doctors to intensify or reduce the pace of their treatment.
Customer Identification: Suppose we want to identify the type of customers who are most likely to buy a particular product in a shopping mall. By doing a simple question and answers survey, we can gather all the features of the customers. Here, a Linear discriminant analysis will help us to identify and select the features which can describe the characteristics of the group of customers that are most likely to buy that particular product in the shopping mall.

## Why do we use LDA?

1. **Dimensionality Reduction**: LDA is a linear dimensionality reduction technique that aims to project data into a lower-dimensional space while maximizing the separability between different classes. For Twitter sentiment analysis, this can be useful in reducing the noise and focusing on features that most significantly differentiate sentiments.

2. **Class-Based Separation**: Unlike some other dimensionality reduction techniques, LDA explicitly aims to separate data into different classes. This feature is beneficial in sentiment analysis, where the goal is to distinguish between positive, negative, and neutral sentiments. LDA can help in creating clear boundaries among these classes.

3. **Interpretability:** LDA is relatively straightforward to understand and implement. This simplicity can be advantageous in projects where interpretability is crucial, as with sentiment analysis for price prediction. Knowing which features contribute to specific sentiment classes can provide insights into how tweets impact cryptocurrency prices.

4. **Effective for Smaller Datasets**: LDA tends to perform well even with smaller datasets, which can be the case when analyzing specific cryptocurrency-related tweets over a limited time frame. This capability can be advantageous when the dataset might not be as large as needed for more complex models like deep learning.

5. **Faster Training and Computation:** Compared to some complex models, LDA can be faster to train and compute, especially for high-dimensional data like text. If you're processing a large volume of tweets, this efficiency can be crucial for real-time or near-real-time analysis.

6. **Enhanced Feature Selection:** LDA's approach to maximizing variance between classes can help identify the most significant features (words or phrases) that contribute to different sentiments. This approach can improve feature selection in your sentiment analysis pipeline.

6. **Combining with Other Techniques:** LDA can work well in combination with other techniques, like TF-IDF (Term Frequency-Inverse Document Frequency), to extract key

features from text data. This flexibility allows you to design a robust pipeline for sentiment analysis.

### Snippets of code:

```
model = LinearDiscriminantAnalysis().fit(x_train, y_train)
```

- Linear Discriminant Analysis (LDA) is a statistical technique used in supervised learning, particularly for classification problems. It aims to find a linear combination of features that best separates two or more classes. LDA is often used in situations where the classes are reasonably separable with linear boundaries.
- LDA can also be used for dimensionality reduction, similar to Principal Component Analysis (PCA), but it does so while keeping the separability of classes in mind.
- The .fit() Method: The .fit() method is used to train the model on the provided data. In this case, x_train and y_train are the training data and the corresponding labels (or target values), respectively. x_train: This represents the training dataset, typically a 2D array or dataframe where each row is a training sample, and each column is a feature. y_train: This is the set of labels or target values corresponding to the x_train data. The labels indicate the class or category for each training sample.
- Putting It All Together: The code snippet creates an instance of the Linear Discriminant Analysis model and then trains it using the x_train data and the corresponding y_train labels. Once the model is trained, it can be used to make predictions, evaluate performance, or further analyze the feature importance for classification tasks.

## 4.8What is an API?

API is an Application Programming Interface that is a collection of communication protocols and subroutines used by various programs to communicate between them. A programmer can make use of various API tools to make their program easier and simpler. Also, an API facilitates programmers with an efficient way to develop their software programs. Thus api meaning is when an API helps two programs or applications to communicate with each other by providing them with the necessary tools and functions. It takes the request from the user and sends it to the service provider and then again sends the result generated from the service provider to the desired user.
A developer extensively uses APIs in his software to implement various features by using an API call without writing complex codes for the same. API is the connection between two or more applications, via APIs, letting you exchange data. It is a medium through which you can share data and communicate with each other by involving APIs to allow web tools to communicate. Due to the rise in cloud-based products, API integration has become very important. We can create an API for an operating system, database system, hardware system, JavaScript file, or similar object-oriented files. Also, an API is similar to a GUI (Graphical User Interface) with one

major difference. Unlike GUIs, an application program interface helps software developers to access web tools while a GUI helps to make a program easier to understand for users.

### Advantages of APIs:

- Efficiency: API produces efficient, quicker, and more reliable results than the outputs produced by human beings in an organization.
- Flexible delivery of services: API provides fast and flexible delivery of services according to developers' requirements.
- Integration: The best feature of API is that it allows the movement of data between various sites and thus enhances the integrated user experience.
- Automation: As API makes use of robotic computers rather than humans, it produces better and more automated results.
- New functionality: While using API the developers find new tools and functionality for API exchanges.

### Disadvantages of APIs:

- Cost: Developing and implementing API is costly at times and requires high maintenance and support from developers.
- Security issues: Using API adds another layer of surface which is then prone to attacks, and hence the security risk problem is common in APIs.

## 4.9What is Flask?

Flask is an API of Python that allows us to build web applications. It was developed by Armin Ronacher. Flask's framework is more explicit than Django's framework and is also easier to learn because it has less base code to implement a simple web application. Flask Python is based on the WSGI (Web Server Gateway Interface) toolkit and Jinja2 template engine.

Flask is a web framework that allows developers to build lightweight web applications quickly and easily with Flask Libraries. It was developed by Armin Ronacher, leader of the International Group of Python Enthusiasts(POCCO). It is basically based on the WSGI toolkit and Jinja2 templating engine.

Python's Flask micro web framework can be used to build web-based apps for interacting and displaying data-driven content. Although Flask is more frequently associated with web development, it can also be used in data science to create straightforward web interfaces, APIs (Application Programming Interfaces), and visualization tools that enable data scientists and analysts to present their data and analyses in a user-friendly and interactive way.

## Advantages of Flask:

- Scalable: Size is everything, and Flask's status as a micro framework means that you can use it to grow a tech project such as a web app incredibly quickly. If you want to make an app that starts small, but has the potential to grow quickly and in directions you haven't completely worked out yet, then it's an ideal choice. Its simplicity of use and few dependencies enable it to run smoothly even as it scales up and up.
- Easy to negotiate: being able to find your way around easily is key for allowing web developers to concentrate on just coding quickly, without getting bogged down. At its core, the micro framework is easy to understand for web developers, not just saving them time and effort but also giving them more control over their code and what is possible.

- Lightweight: Flask is a lightweight framework with a simple and intuitive design, making it easy to learn and use.
- Flexibility: Flask offers a high level of flexibility, allowing developers to choose libraries and tools that suit their needs.
- Rapid prototyping: Flask provides the ability for rapid development and is suitable for building small projects and prototypes.
- Flask can be used with any database like: SQL and NoSQL and with any Frontend Technology such as React or Angular.
- Flask is great for small to medium projects that do not require the complexity of a large framework.
- Flask Documentation: Following the creator's own theory that "nice documentation design makes you actually write documentation," Flask users will find a healthy number of examples and tips arranged in a structured manner. This encourages developers to use the framework, as they can easily get introduced to the different aspects and capabilities of the tool.

## Disadvantages of Flask:

- Lack of built-in features: Compared to other frameworks, Flask has fewer built-in features and may require additional extension libraries.
- Flask lacks a large toolbox. This means that developers will have to manually add extensions such as libraries. And, if you add a huge number of extensions, it may start to slow down the app itself due to a multitude of requests.

- Lack of standardization: Flask project structures can vary among developers, lacking a unified standardization, which means difficulty to get familiar with a larger Flask app
- Because of the fact that development of a web app using Flask can take a variety of twists and turns, a web developer arriving to the project mid-way through can struggle to come to terms with how it's been designed. The modular nature of the micro framework that we mentioned earlier can come back to haunt coders, who will have to familiarize themselves with each constituent part.

- Maintenance costs: Because it is so versatile in terms of which technologies it can interface with, quite often a company using Flask will incur extra costs of supporting those technologies. For example, if a technology interfacing with your Flask app becomes obsolete or is discontinued, then the company will have to scramble to find a new compatible one. The more complicated the app becomes, the higher the potential maintenance and implementation costs.

## Why do we use Flask?

## There are several reasons why it's commonly chosen for web development projects:

- Web Interface: Flask allows you to create a web interface for your cryptocurrency price prediction model. Users can interact with your model through a web browser, making it more accessible
- Real-Time Updates: With Flask, you can build a dynamic web application that provides real-time updates on cryptocurrency prices. Users can see the latest predictions and adjust their strategies accordingly.
- Scalability: Flask applications can be easily scaled to accommodate a large number of users. This is important in cryptocurrency prediction projects, as there may be significant interest from traders and investors.
- Integration: Flask can easily integrate with other Python libraries and frameworks commonly used in data science and machine learning, such as Pandas, NumPy, and scikit-learn. This makes it easier to incorporate your prediction model into a web application.
- Customization: Flask provides a high degree of customization, allowing you to create a user interface that meets your specific requirements. You can design the layout, style, and functionality of your web application to suit your needs.
- Deployment: Flask applications can be deployed easily on various platforms, including cloud services like AWS, or Google Cloud Platform. This makes it straightforward to make your cryptocurrency prediction model accessible to a wide audience.
- Community and Ecosystem: Flask has a large and active community of developers who contribute plugins, extensions, and tutorials to the Flask ecosystem. This vibrant community ensures that there are plenty of resources available for developers who are building Flask applications, including documentation, tutorials, and third-party libraries.
- Modularity: Flask is built around the concept of modular components called "extensions," which provide additional functionality such as authentication, database integration, and form validation. This modularity allows developers to add or remove features as needed, making it easy to customize and extend Flask applications.

- Simplicity: Flask is designed to be simple and easy to use, making it an excellent choice for beginners and experienced developers alike. Its minimalist approach allows developers to focus on building the features they need without unnecessary complexity.

## Snippets of code:
## Importing Libraries:

```
import pandas as pd
import numpy as np
import pickle
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
from keras.models import Sequential
from keras.layers import LSTM,GRU, Dropout, Dense
from flask import Flask, request, jsonify
from keras.initializers import Orthogonal
from io import BytesIO
from flask import Flask, render_template, send_file
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import json
import yfinance as yf
import cryptocompare
from yahoo_fin.stock_info import get_live_price
import base64
import time
```

**Imports:**
Data handling and numerical computations:

- pandas and numpy for data manipulation and numerical operations.
- Machine Learning:
    1. pickle for saving and loading Python objects.
    2. sklearn.preprocessing.MinMaxScaler for scaling data.
    3. keras.models for loading and defining neural network models.
    4. keras.layers for adding layers to neural networks.
- Web Framework: Flask for creating the web application and handling HTTP requests.
- Plotting: matplotlib.pyplot and FigureCanvasAgg for generating plots.
- Data Retrieval:
    1. yfinance and cryptocompare for fetching cryptocurrency data.
    2. yahoo_fin.stock_info for getting live prices.

- Other utilities:
    1. base64 for encoding images to be sent over the web.
    2. time for time-related operations.
    3. json for working with JSON data.

```
app = Flask(_name_)

def btc_price(coin):
    ticker_symbol = coin+"-USD"
    current_price = get_live_price(ticker_symbol)
    formatted_price = "{:.2f}".format(current_price)

    return formatted_price
```

Initializes the Flask app with the name of the current module. Concatenates the coin symbol with "USD" to form the ticker symbol. Fetches the live price and formats it to two decimal places.

```
# Function to update plot with new data
def update_plot(coin):
    # Retrieve historical price data for the cryptocurrency for the last 30 days
    historical_data = cryptocompare.get_historical_price_day(coin,
currency="USD", limit=7)
    df = pd.DataFrame(historical_data)

    # Extract timestamps and prices
    dates = pd.to_datetime(df['time'], unit='s')  # Convert Unix timestamp to
datetime
    historical_prices = df['close']

    # Plot historical data
    plt.figure(figsize=(11, 6))
    plt.plot(dates, historical_prices, label=f"{coin} - {"USD"} (Past 30 days)",
color='blue')

    # Retrieve real-time price data
    real_time_data = cryptocompare.get_price(coin, currency="USD")
    current_price = real_time_data[coin]["USD"]

    # Add current price to plot
    current_time = pd.Timestamp.now()
    plt.scatter(current_time, current_price, color='red', label=f"{coin} -
{"USD"} (Today)")
```

```
    # Update legend
    plt.legend()

    # Convert plot to bytes
    img = BytesIO()
    plt.savefig(img, format='png')
    img.seek(0)
    plot_data = base64.b64encode(img.getvalue()).decode()

    return plot_data
```

- Fetches historical price data and plots it.
- Adds the current price to the plot.
- Converts the plot to a base64-encoded image.

```
def predict(coin,close_price):

 scaler = MinMaxScaler(feature_range=(0, 1))
 SEQUENCE_SIZE = 30

 if coin=="BTC":
     model = load_model('models/btc.h5')
     model = Sequential()
     model.add(GRU(units=30,return_sequences=True,input_shape=(SEQUENCE_SIZE,1)))
     model.add(Dropout(0.2))
     model.add(GRU(units=60,return_sequences=True))
     model.add(Dropout(0.2))
     model.add(GRU(units=90))
     model.add(Dropout(0.2))
     model.add(Dense(units=1))
 elif coin=="LTC":
     model = load_model('models/ltc.h5')
     model = Sequential()
     model.add(GRU(units=30,return_sequences=True,input_shape=(SEQUENCE_SIZE,1)))
     model.add(Dropout(0.2))
     model.add(GRU(units=60,return_sequences=True))
     model.add(Dropout(0.2))
     model.add(GRU(units=90))
     model.add(Dropout(0.2))
     model.add(Dense(units=1))
```

```
elif coin=="ETH":
    model = load_model('models/eth.h5')
    model = Sequential()
    model.add(GRU(units=30,return_sequences=True,input_shape=(SEQUENCE_SIZE,1)))
    model.add(Dropout(0.2))
    model.add(GRU(units=60,return_sequences=True))
    model.add(Dropout(0.2))
    model.add(GRU(units=90))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
elif coin=="XMR":
    model = load_model('models/xmr.h5')
    model = Sequential()
    model.add(GRU(units=30,return_sequences=True,input_shape=(SEQUENCE_SIZE,1)))
    model.add(Dropout(0.2))
    model.add(GRU(units=60,return_sequences=True))
    model.add(Dropout(0.2))
    model.add(GRU(units=90))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
```

- Defines a model based on the cryptocurrency specified.
- Uses a GRU-based neural network with dropout layers for prediction.
- Scales the input data and predicts future prices.
- Inverses the scaling of the predicted data.

**Functions:**

- btc_price(coin): Fetches the current price of the specified cryptocurrency in USD. Uses yahoo_fin.stock_info.get_live_price.
- update_plot(coin): Retrieves and plots the historical price data of a cryptocurrency for the last 30 days using cryptocompare.get_historical_price_day. Fetches the current price and adds it to the plot. Converts the plot to a base64-encoded PNG image for web display.
- predict(coin, close_price): Loads a pre-trained model specific to the given cryptocurrency (BTC, LTC, ETH, XMR). The model architecture involves stacked GRU (Gated Recurrent Unit) layers with dropout for regularization. Scales the input price data using MinMaxScaler. Predicts future prices using the model. Inverses the scaling of predictions to get the original scale. Returns the predicted prices.

This code combines data retrieval, machine learning, and web application frameworks to create a tool for predicting and visualizing cryptocurrency prices. It handles fetching real-time and historical data, plotting, and running predictions with pre-trained neural network mod

# Chapter 5
# Testing

# 5.1Evaluation Metrics:

**To assess the quality of a model's predictions, we computed the mean absolute error (MAE), mean square error(MSE), mean absolute percentage error (MAPE) and root mean squared error (RMSE).**

## 5.1.1MAE:

Mean Absolute Error (MAE) is a commonly used metric for evaluating the accuracy of a predictive model. It measures the average magnitude of errors in a set of predictions, without considering their direction. It is measured as the average absolute difference between the predicted values and the actual values and is used to assess the effectiveness of models.

## The formula to calculate MAE is:

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|x_t - \hat{x}_t|.$$

Figure 5.1.1

```
from sklearn.metrics import mean_absolute_error
print('MAE =',mean_absolute_error(xmry_test,xmry_pred))
```

## 5.1.2MSE:

Mean Squared Error (MSE) is a commonly used metric to measure the difference between the predicted and actual values of a regression problem. It is calculated by taking the average of the squared differences between the predicted and actual values.

## The formula to calculate MSE is:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(x_t - \hat{x}_t)^2.$$

Figure 5.1.2

```
from sklearn.metrics import mean_squared_error
```

```
print('MSE=',mean_squared_error(xmry_test,xmry_pred))
```

## 5.1.3MAPE:

Mean Absolute Percentage Error (MAPE) measures the average magnitude of error produced by a model, or how far off predictions are on average. is a metric that defines the accuracy of a forecasting method.
It represents the average of the absolute percentage errors of each entry in a dataset to calculate how accurate the forecasted quantities were in comparison with the actual quantities.

## The formula to calculate MAPE is:

$$MAPE = \frac{100}{N}\sum_{t=1}^{N}\frac{|x_t - \hat{x}_t|}{|x_t|}.$$

Figure 5.1.3

```
from sklearn.metrics import mean_absolute_percentage_error
print('MAPE =',mean_absolute_percentage_error(xmry_test,xmry_pred))
```

## 5.1.4RMSE:

Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a predictive model, particularly in regression analysis. RMSE measures the difference between the predicted and actual values of a dataset and provides a measure of how much the predictions deviate from the actual values, on average. Mathematically, it is the standard deviation of the residuals. Residuals represent the distance between the regression line and the data points.

## The formula to calculate RMSE is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(x_t - \hat{x}_t)^2}.$$

Figure 5.1.4

```
#Root Mean Square Erorre
print('RMSE=',np.sqrt(mean_squared_error(xmry_test,xmry_pred)))
```

## Evaluation Matrix for Bitcoin(BTC):

**BTC**

| Error / Model | MAE (BTC) | MSE (BTC) | MAPE (BTC) | RMSE (BTC) |
|---|---|---|---|---|
| LSTM | 0.01085 | 0.00019 | 0.03095 | 0.01380 |
| GRU | 0. 00776 | 0. 00011 | 0. 02269 | 0. 01054 |
| ARIMA | 0.210386 | 0.062159 | 0.989799 | 0.249317 |
| SVR | 0.043688 | 0.004133 | 0.255097 | 0.081429 |

We used 4 Matrices in Evaluation Matrix: Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE)

**In Bitcoin:** MSE was better than Matrix as it had the lowest error rate. In the models, the best of them was GRU, as the error rate in this model was very small compared to other models, followed in second place by LSTM, then SVR, and the last model and the largest in error rate is ARIMA.

## Evaluation Matrix for Ethereum (ETH):

**ETH**

| Error / Model | MAE (ETH) | MSE (ETH) | MAPE (ETH) | RMSE (ETH) |
|---|---|---|---|---|
| LSTM | 0.01002 | 0.00017 | 0.03178 | 0.01309 |
| GRU | 0. 00729 | 0. 00010 | 0. 02365 | 0. 01027 |
| ARIMA | 0.229848 | 0.106178 | 1.013612 | 0.32585 |
| SVR | 0.037183 | 0.004859 | 0.239011 | 0.064295 |

**In Ethereum:** MSE was better than Matrix as it had the lowest error rate. In the models, the best of them was GRU, as the error rate in this model was very small compared to other models, followed in second place by LSTM, then SVR, and the last model and the largest in error rate is ARIMA.

## Evaluation Matrix for Litecoin(LTC):

**LTC**

| Error / Model | MAE (LTC) | MSE (LTC) | MAPE (LTC) | RMSE (LTC) |
|---|---|---|---|---|
| LSTM | 0.00714 | 9.98216e-05 | 0.03581 | 0.00999 |
| GRU | 0. 00606 | 7.84196e -05 | 0. 03027 | 0. 00885 |
| ARIMA | 0.169968 | 0.057039 | 1.199943 | 0.238828 |
| SVR | 0.053655 | 0.010099 | 0.499788 | 0.100495 |

**In Litecoin:** MSE was better than Matrix as it had the lowest error rate. In the models, the best of them was GRU, as the error rate in this model was very small compared to other models, followed in second place by LSTM, then SVR, and the last model and the largest in error rate is ARIMA.

## Evaluation Matrix for Monero (XMR):

**XMR**

| Error / Model | MAE (XMR) | MSE (XMR) | MAPE (XMR) | RMSE (XMR) |
|---|---|---|---|---|
| LSTM | 0.01826 | 0. 00040 | 0. 07031 | 0. 02006 |
| GRU | 0. 00561 | 6.87769e -05 | 0. 02164 | 0. 00829 |
| ARIMA | 0.259147 | 0.101286 | 0.966987 | 0.318254 |
| SVR | 0.047848 | 0.006719 | 0.241991 | 0.081973 |

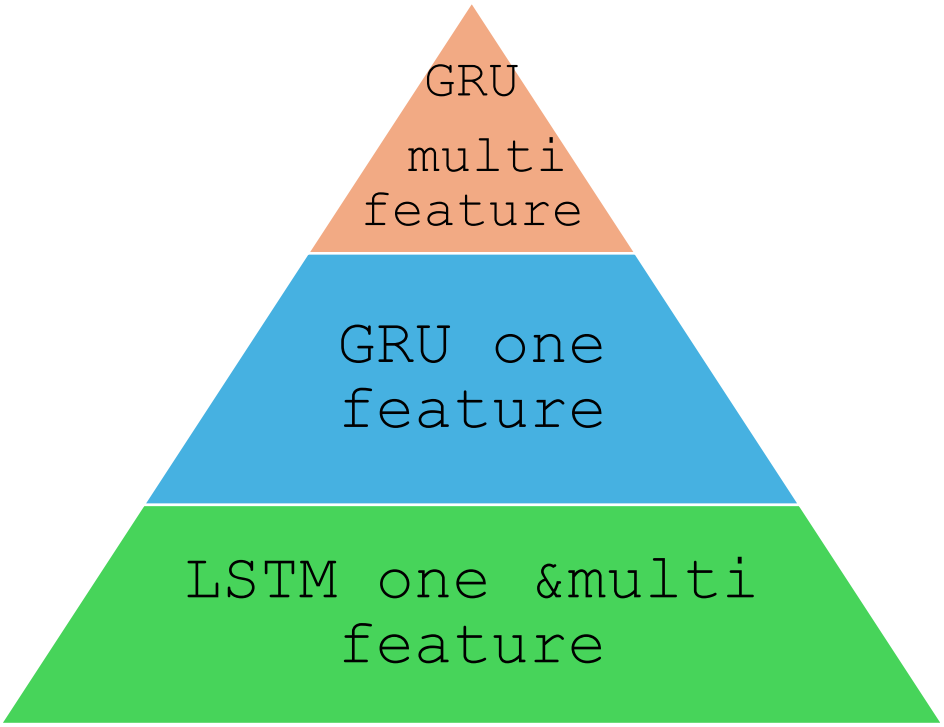
**In Monero:** MSE was better than Matrix as it had the lowest error rate. In the models, the best of them was GRU, as the error rate in this model was very small compared to other models, followed in second place by LSTM, then SVR, and the last model and the largest in error rate is ARIMA.

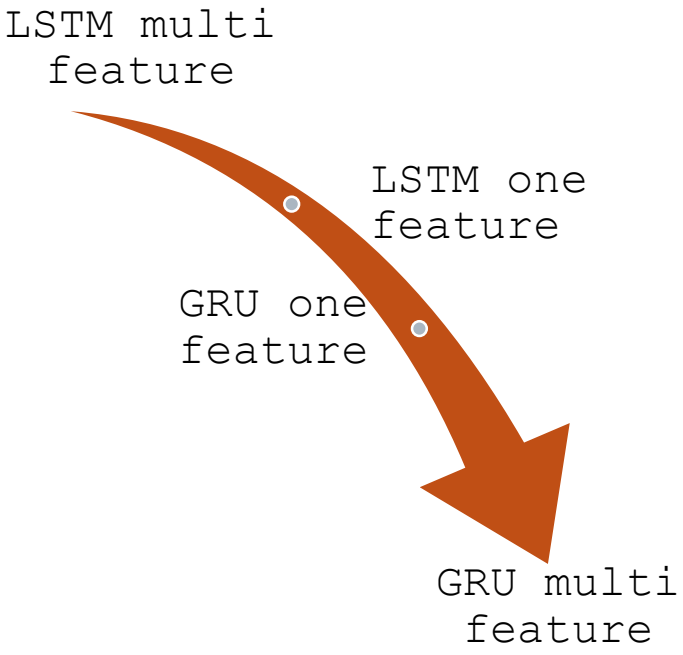
## Evaluation Matrix for Multi features prediction:

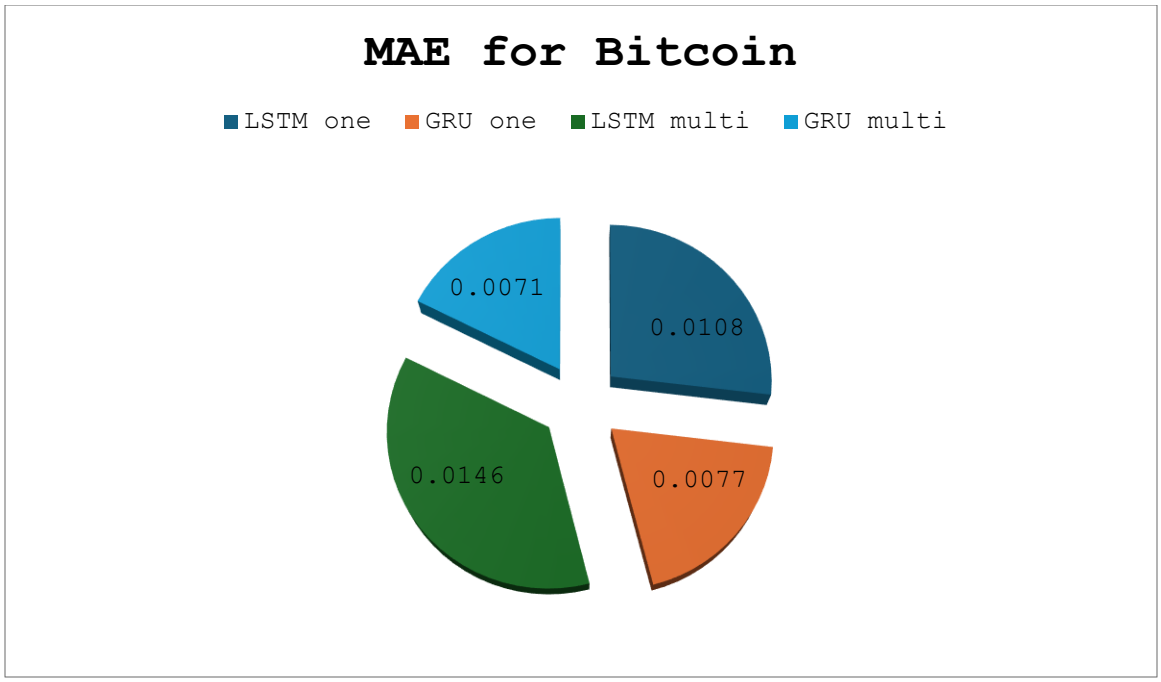
| | Error Metrics | MAE | | | | MSE | | | | MAPE | | | | RMSE | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Coin / Model | BTC | ETH | LTC | XMR | BTC | ETH | LTC | XMR | BTC | ETH | LTC | XMR | BTC | ETH | LTC | XMR |
| One feature | LSTM | 0.0108 | 0.0100 | 0.007 | 0.0182 | 0.0001 | 0.0001 | 9.9821e-05 | 0.0004 | 0.0309 | 0.0317 | 0.0358 | 0.070 | 0.0138 | 0.0130 | 0.0099 | 0.020 |
| | GRU | 0.0077 | 0.0072 | 0.0060 | 0.0056 | 0.0001 | 0.0001 | 7.841e-05 | 6.877e-05 | 0.0226 | 0.0236 | 0.0302 | 0.021 | 0.0105 | 0.0102 | 0.0088 | 0.008 |
| multi features | LSTM | 0.0146 | 0.0213 | 0.008 | 0.006 | 0.0002 | 0.00055 | 0.00012 | 9.51e-05 | 0.0438 | 0.0658 | 0.0450 | 0.026 | 0.0169 | 0.0234 | 0.0113 | 0.009 |
| | GRU | 0.0071 | 0.012 | 0.012 | 0.005 | 0.0001 | 0.002 | 0.002 | 6.60e-05 | 0.021 | 0.038 | 0.0633 | 0.020 | 0.0103 | 0.0149 | 0.0160 | 0.008 |

The table show that the multi feature prediction is better than single feature prediction as it gives less error

- The best model is GRU multi feature prediction


GRU
multi
feature
GRU one
feature
LSTM one &multi
feature


LSTM multi
feature
LSTM one
feature
GRU one
feature
GRU multi
feature




## 5.2Twitter Sentiment Analysis:

The accuracy of Decision Tree is about 97.4

```
accuracy_score(y_test,y_predicted)*100
```

```
97.43589743589743
```

The accuracy of Linear Discriminant Analysis is about 97.7

```
accuracy_score(y_test,predictions)*100
```

```
97.66899766899768
```

# Chapter 6
# Web site & Conclusions

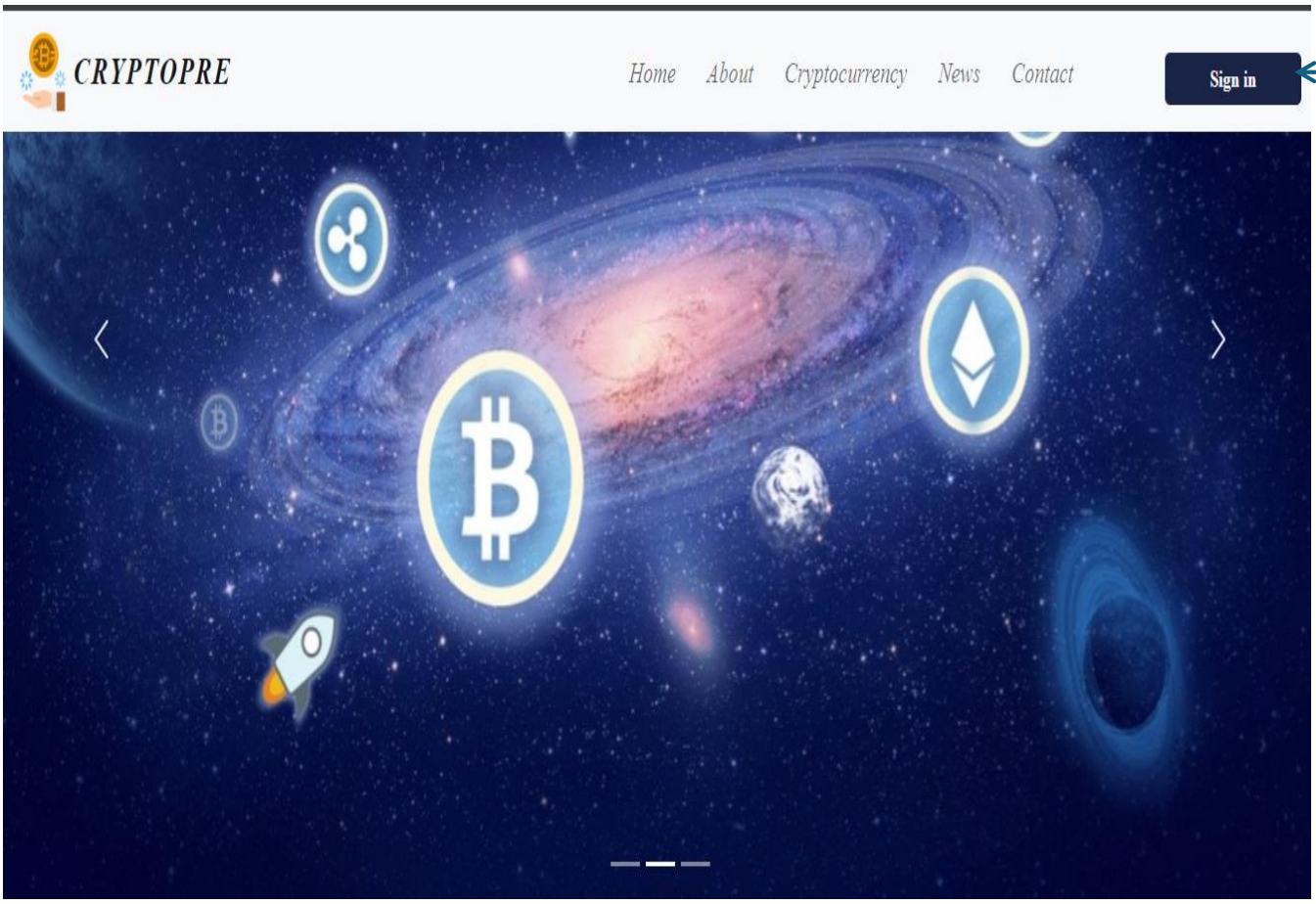
# Web site:

## The importance of the website:

-CryptoPre helps many investors make informed decisions about when to buy or sell cryptocurrencies. Predictions can help them maximize profits and minimize losses.

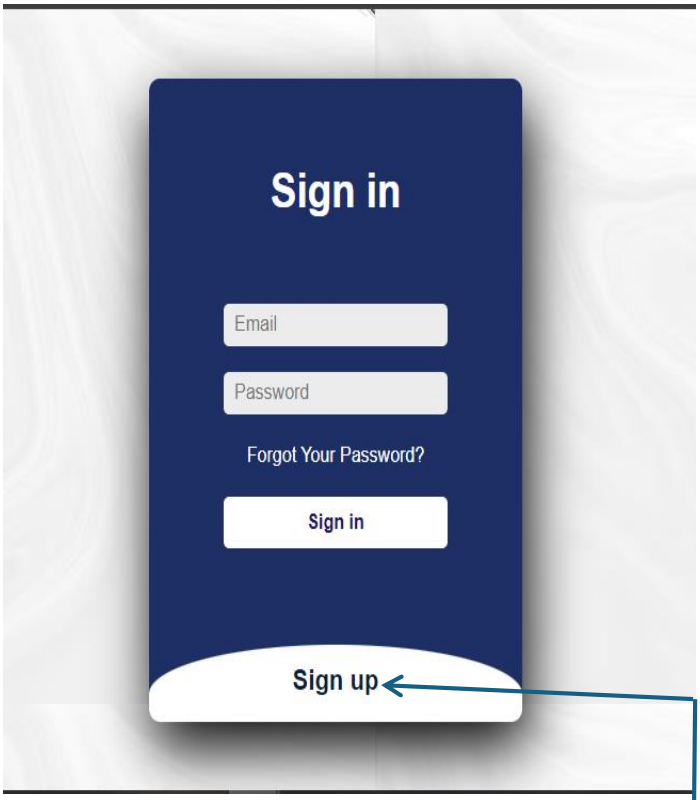
- We used the bootstrap in the front-end and node js in the back-end.

# Design Implementation:

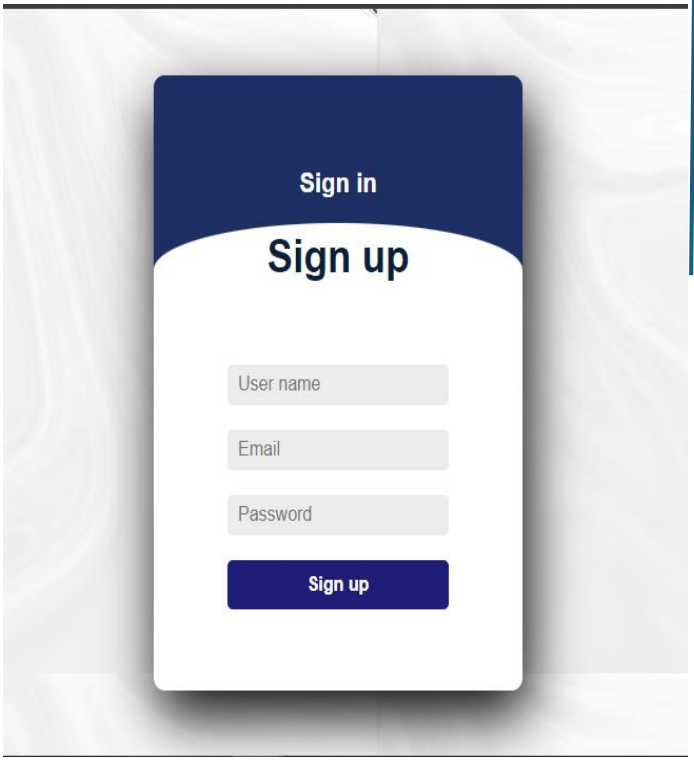



On this page there are some links through which you go to other pages, such as About, Cryptocurrency, Contact and, News.
There is also a Sign in button, when you click on it, you go to the Sign in page.

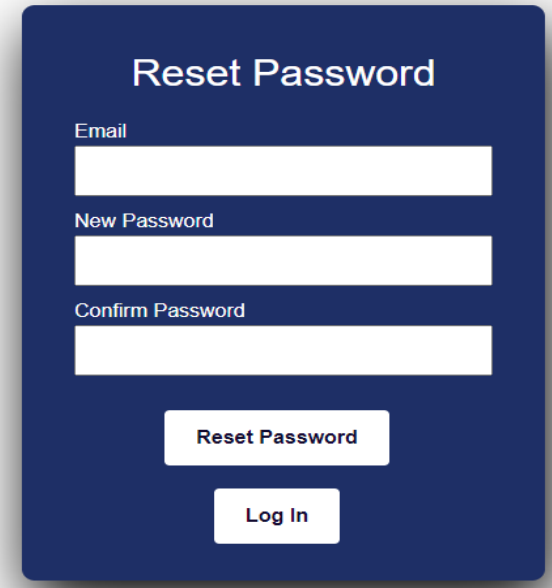
# ***Register pages(signin,signup,Update)***




If the user has logged in before, then he registers his data, which is Email and Password, so that he is allowed to enter the website.




But, if he is entering for the first time, he clicks on Sign up. Data will appear in front of him that he must register in order to be saved in the database.




When clicking on this link (forget your password), the user can change the password from it, writing his email and the new password, then confirming it again, and then clicking on Reset password and it will be changed.

# About page

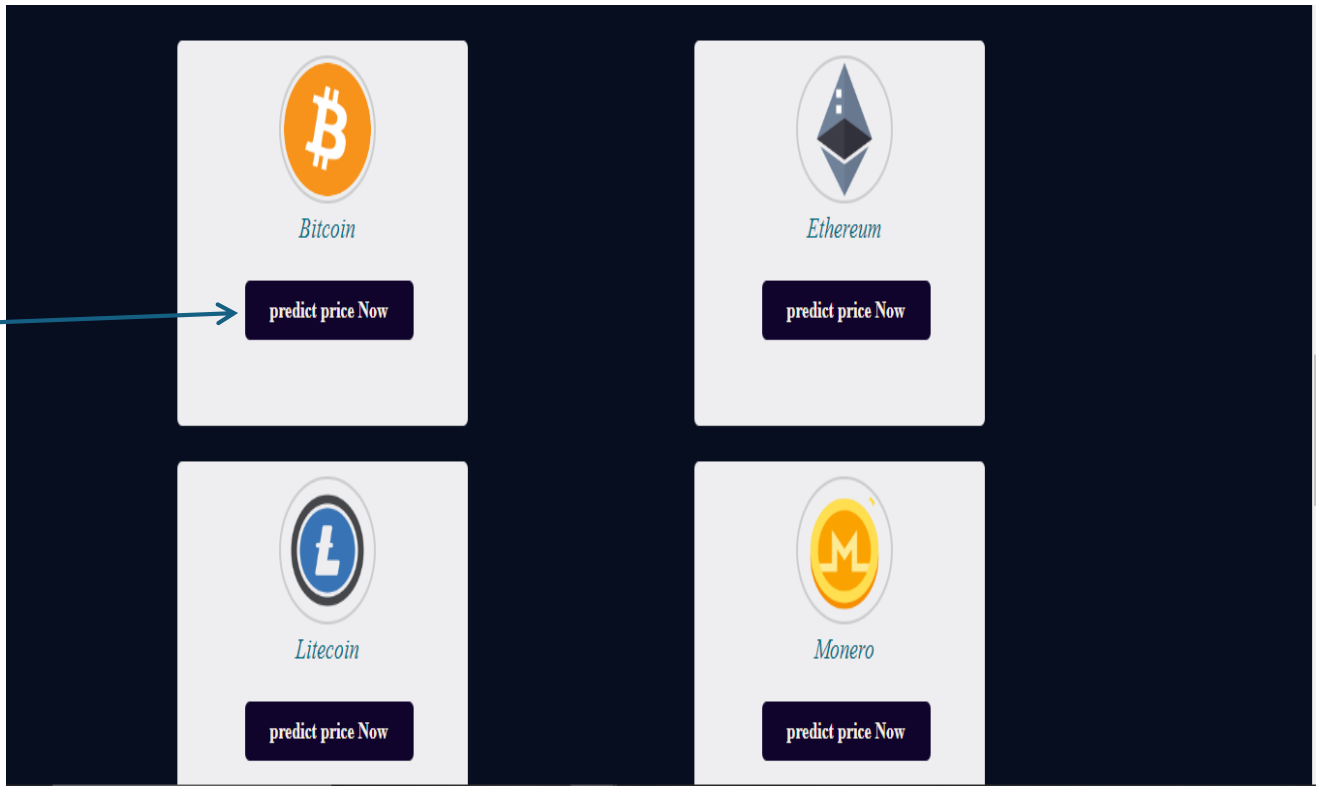
## What is Cryptocurrencies?

A digital currency, such as Bitcoin, that is used as an alternative payment method or speculative investment s any form of currency that exists digitally or virtually and uses cryptography to secure transactions. Cryptocurrencies don t have a central issuing or regulating authority, instead using a decentralized system to record transactions and issue new units.

## Abou Cryptocurrency price prediction

"While cryptocurrency markets are notoriously volatile and unpredictable, many investors and analysts use a variety of methods to attempt price prediction. These methods often include technical analysis, which involves studying past market data and chart patterns to forecast future price movements. Additionally, fundamental analysis examines factors such as adoption rates, regulatory developments, ."

# Cryptocurrency page




On this page there are 4 currency cards, and each card has a picture of the currency, its name, its current price, and there is a button.
When you click this button, you will be taken to the Bitcoin page

- When you press the button on each card (predict price now), you will be moved to each page for that currency.

# ***Bitcoin page***


On this page (on each currency page) there is the current price of the currency, and there are 3 cards, a card that predicts the price after a day, a card that predicts the price after 3 days, and a card that predicts the price after 7 days. There is also a visualization of the currency on the same page.

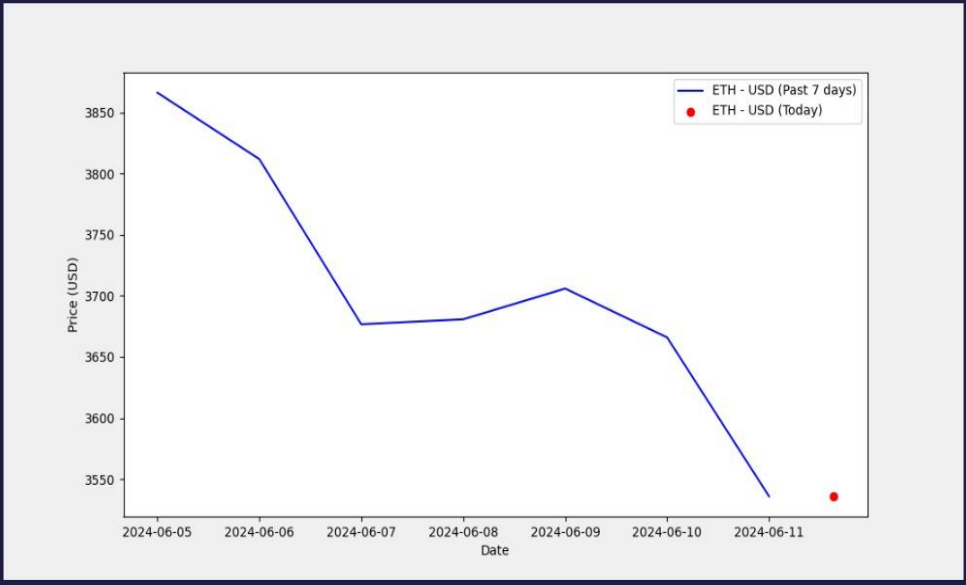
# The Ethereum page

*Ethereum (ETH) USD*

*current price*

*3534.95$*

| Next Day Price | Next 3 Day Price | Next 7 Day Price |
|---|---|---|
| 2879.5283$ | 2878.1946$ | 2877.5981$ |



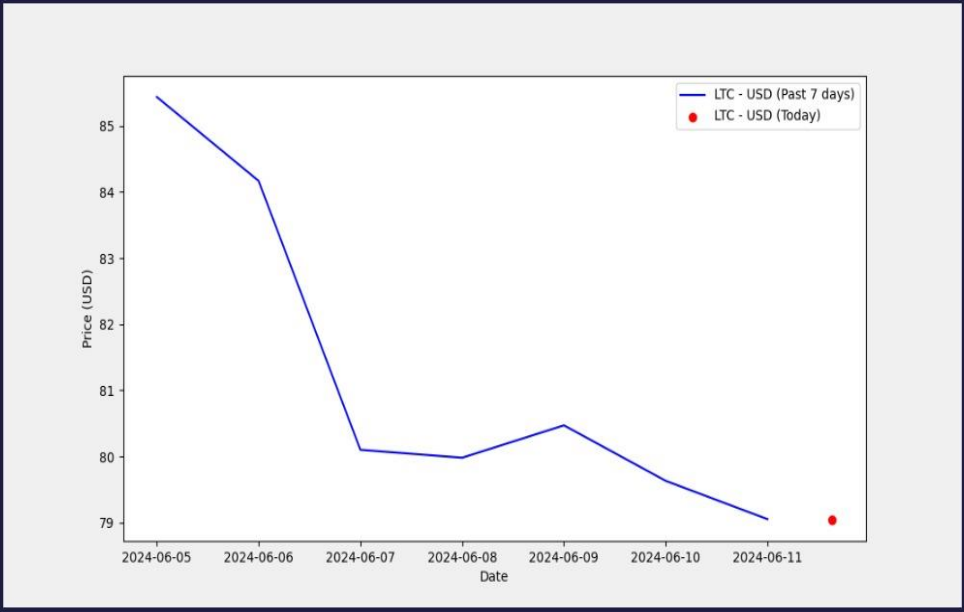
# ***The Litecoin page***

*Litecoin (LTC) USD*

*current price*

*78.94$*

| *Next Day Price* | *Next 3 Day Price* | *Next 7 Day Price* |
|---|---|---|
| *78.727005$* | *78.72443$* | *78.72459$* |



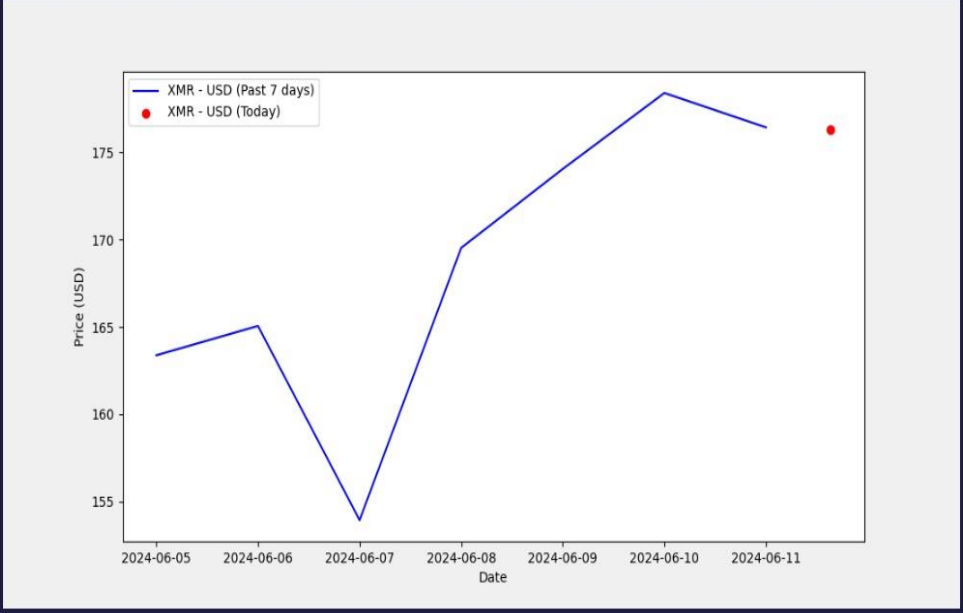
# ***The Monero page***

*Monero (XMR) USD*

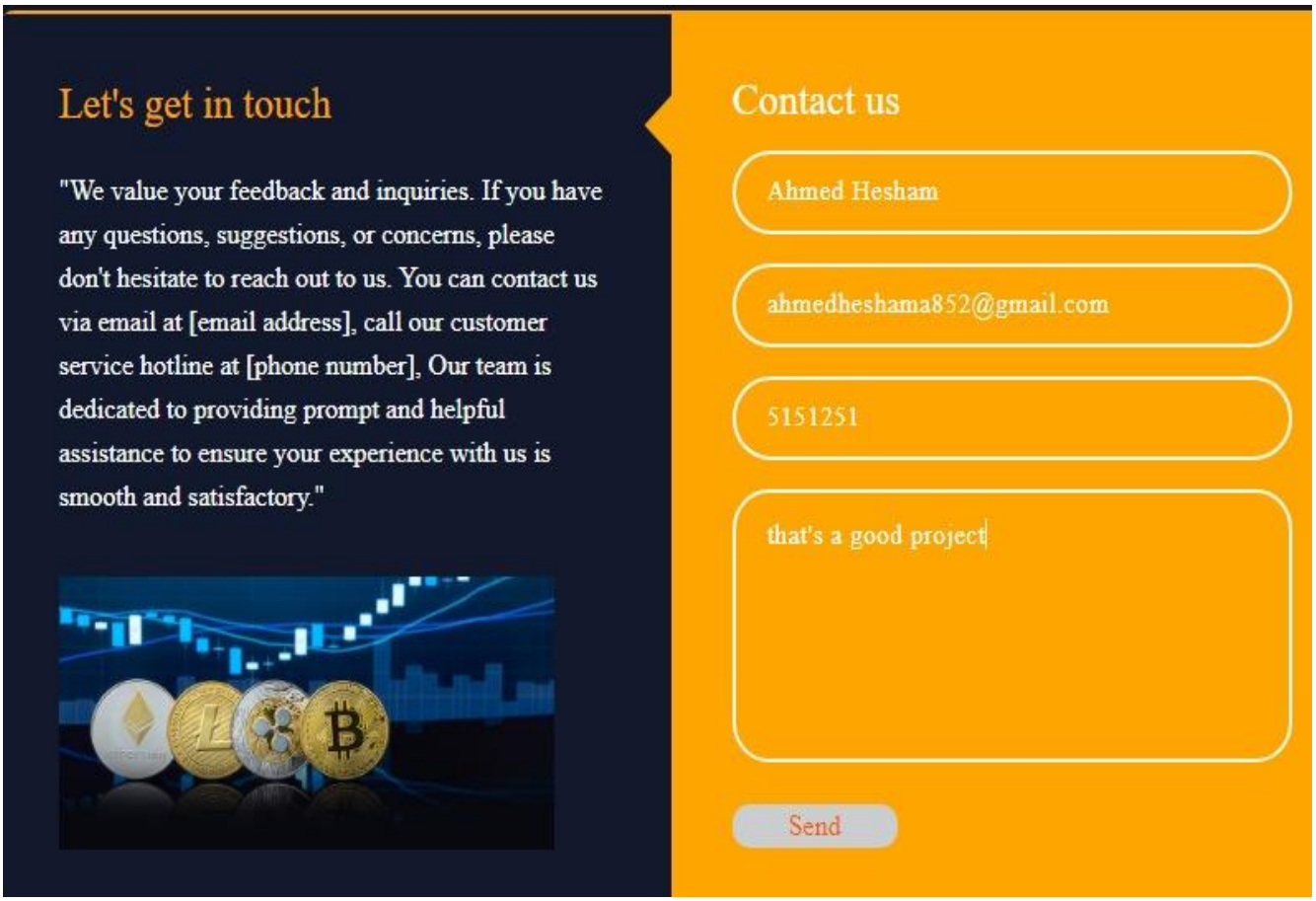
*current price*

*176.05$*

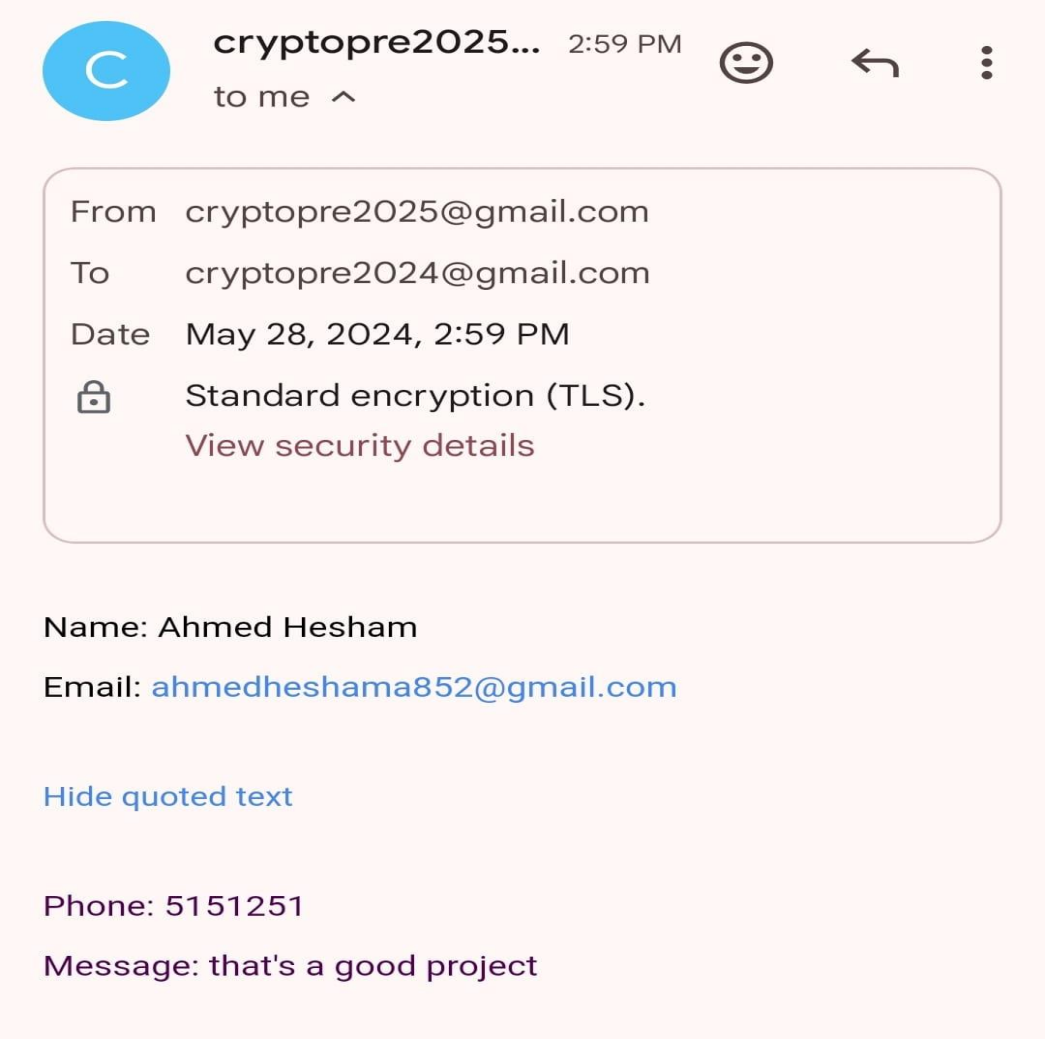
| *Next Day Price* | *Next 3 Day Price* | *Next 7 Day Price* |
|---|---|---|
| *131.9645$* | *131.95515$* | *131.953$* |



# ***Contact page***

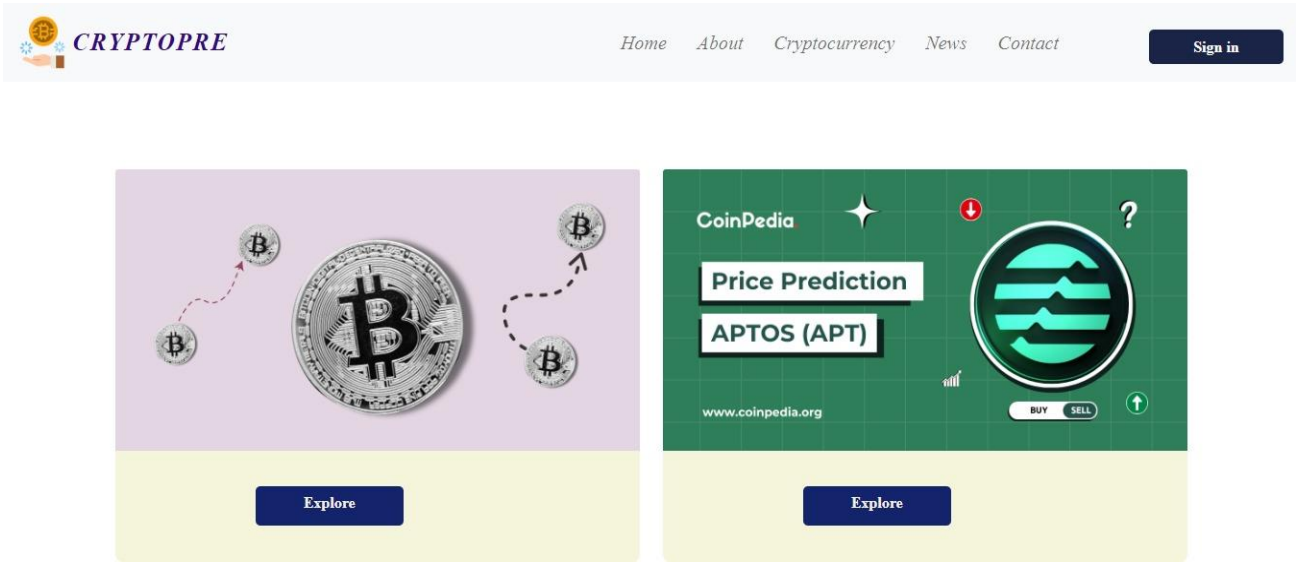


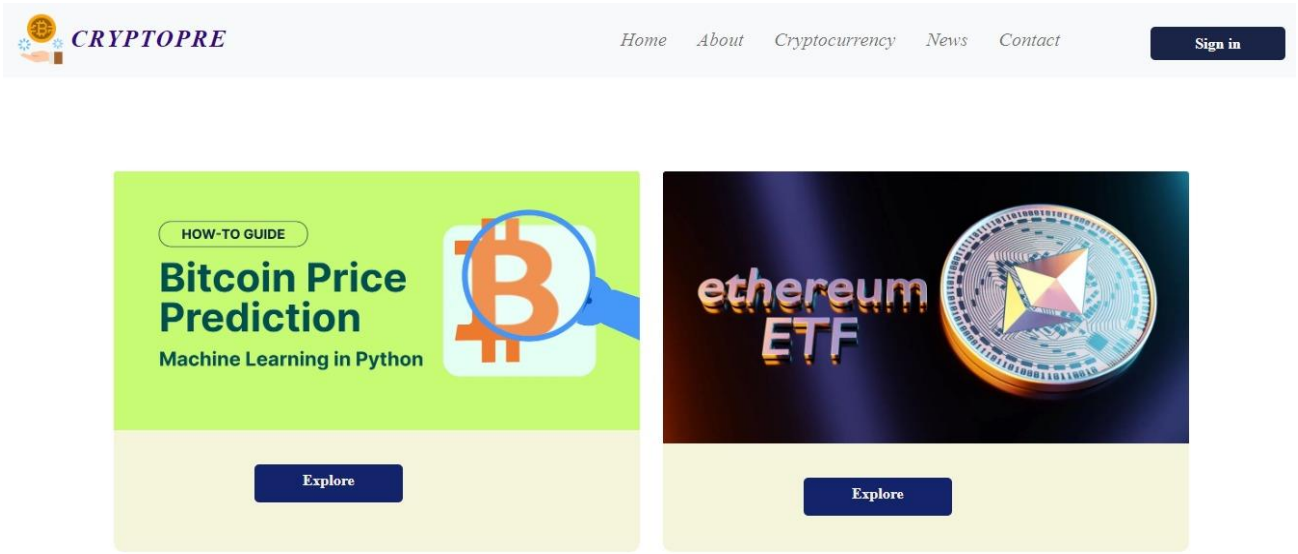

Through this page, the user can send us a review about our website or suggestions that he would like us to add to the website by writing his name, his email, and his phone number, then writing the message he wants to send, and after that he clicks on send, and the message will be sent to our email.




- This picture shows us the arrival of the message that Ahmed Hesham sent to us.

# News page


CRYPTOPRE
Home
About
Cryptocurrency
News
Contact
Sign in
Explore
CoinPedia
Price Prediction
APTOS (APT)
www.coinpedia.org
BUY
SELL
Explore


CRYPTOPRE
Home
About
Cryptocurrency
News
Contact
Sign in
HOW-TO GUIDE
Bitcoin Price Prediction
Machine Learning in Python
Explore
ethereum ETF
Explore


Phone:+1837628791-898939893
Support
Email(FAST):Help@Domain.Com
Business Email:Help@Domain.Com
Company
Home
About Us
Cryptocurrency
Contact Us
News

## 6.2Conclusions

Our team compares deep learning (DL) and machine learning (ML) for forecasting cryptocurrency prices.we use four datasets in this approach which is historical prices from 2014 to 2023 . The datasets are BTC,LTC, ETH and XMR. Results show that, in general, DL approaches are the best models for this task. In particular, the GRU is the best-performing model based on Evaluation matrix we use (MSE, MSE,MAPE and RSME) to evaluate error in each model. In addition we use two techniques one for single feature prediction which depends in close price only and multi feature prediction which depends in four features (open, high, low, close). In general multi feature prediction provide best results. To develop our project we use another technique which is Twitter sentiment analysis. In this technique we use the sentiment of tweets as well as historical data to forecast the price movements of cryptocurrency. Also we use VADER. VADER's is a simple lexicon and rule-based approach which not only gives polarity but also tell how positive or negative a sentence. However, when compared to sophisticated machine learning techniques, the simplicity of VADER carries several advantages. First, it is both quick and computationally economical without sacrificing accuracy. Second, the lexicon and rules used by VADER are directly accessible, not hidden within a machine-access only black-box. we provide a survey and comparative study of existing techniques for opinion mining including machine learning  such as decision tree which has accuracy about 97 and LDA which has accuracy about 98 and lexicon-based approaches such as VADER. We can conclude that more the cleaner data, more accurate results can be obtained. Also we built our website using bootstraps in frontend and node js in backend. Finally we use Flask API to connect models that we use to our website.

# References:

**Data PreProcessing:**


1- Normalization: A Preprocessing Stage by S.Gopal Krishna Patro1, Kishore Kumar sahu **link:** https://arxiv.org/pdf/1503.06462

2- OVERVIEW OF DATA PREPROCESSING by Gurram Bhaskar and Motati Dinesh Reddy e-ISSN: 2582-5208 **Link:** https://www.irjmets.com/uploadedfiles/paper/volume3/issue_3_march_2021/6728/1628083286.pdf


**Cryptocurrency prediction price using ML and DL techniques:**


3- Forecasting Cryptocurrency Prices: A Comparison of Machine Learning, Deep Learning, and Ensembles for Kate Murray , Andrea Rossi , Diego Carraro and Andrea Visentin 2023,5,196-209 .
**Link:** https:// doi.org/10.3390/forecast5010010

4- A comparative study of bitcoin price prediction using SVR and LSTM By J. Arumugam, S.Sabarichvarane and Dr. V. Prasanna Venkatesan.
**Link:** IJCRT2209105.pdf

**5-** A Deep Learning-Based Cryptocurrency Price Prediction Model That Uses On-Chain Data by GYEONGHO KIM, DONG-HYUN SHIN, JAE GYEONG CHOI AND SUNGHOON LIM under Grant 2021R1F1A1046416 and under Grant 1.220083.
**Link:**https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9781377

**6-** Predicting Cryptocurrency Prices with Machine Learning Algorithms: A Comparative Analysis by Harsha Nanda Gudavalli Khetan Venkata Ratnam Kancherla**. Link**: FULLTEXT03.pdf

**7-** A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms by Mohammad J. Hamayel and Amani Yousef Owda
**Link:** https://doi.org/10.3390/ai2040030

**8-** Comparative Study of Bitcoin Price Prediction by Ali Mohammadjafari **Link:** https://arxiv.org/pdf/2405.08089

9- Review of deep learning models for crypto price prediction: implementation and evaluation by Jingyang Wua, Xinyi Zhanga, Fangyixuan Huanga, Haochen Zhoua and Rohtiash Chandraa Link: https://arxiv.org/pdf/2405.11431

10- Prediction of Cryptocurrency Price using Time Series Data and Deep Learning Algorithms byMichael Nair, Mohamed I. Marie, Laila A. Abd-Elmegid (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 14, No. 8, 2023 **Link:** https://ftp.saiconference.com/Downloads/Volume14No8/Paper_37-Prediction_of_Cryptocurrency_Price_using_Time_Series_Data.pdf

11- Time Series Forecasting Based on ARIMA and LSTM by Peiqi Liu Link: https://www.atlantis-press.com/article/125975511.pdf

**Arima :**

12- Time Series Forecasting using ARIMA Model by Hussan Al-Chalabi, Yamur K. Al-Douri and Jan Lundberg **Link:** https://www.diva-portal.org/smash/get/diva2:1266336/FULLTEXT01.pdf

13- The ARIMA versus Artificial Neural Network Modeling by Motaz Khorshid, Assem Tharwat, Amer Bader and Ahmed Omran **Link:** https://ijci.journals.ekb.eg/article_33936_8dd31f2f184453860f32dc8bd08c8417.pdf

14- The mathematical structure of ARIMA models by Robert Nau **Link:** https://people.duke.edu/~rnau/Mathematical_structure_of_ARIMA_models--Robert_Nau.pdf

15- NOTES ON TIME SERIES ANALYSIS ARIMA MODELS AND SIGNAL EXTRACTION by Regina Kaiser and Agustín Maravall **Link:** https://www.bde.es/f/webbde/SES/Secciones/Publicaciones/PublicacionesSeriadas/DocumentosTrabajo/00/Fic/dt0012e.pdf

16- Autoregressive Integrated Moving Average (ARIMA) Modeling by ALBERTO LUCEÑO AND DANIEL PEÑA **Link:** https://halweb.uc3m.es/esp/personal/personas/dpena/publications/ingles/2007Enciclopedia_Luceno_ARIMAmod.pdf

17- CRYPTOCURRENCY PRICE PREDICTION USING ARIMA MODEL by Amritha Subburayan, Kamalnath Sathyamurthy, Sanjay Aravind Loganathan and Yaswanth Kumar Reddy **Link:** https://ap-kamal.github.io/02_Crypto.pdf

**SVR:**

18- Support Vector Regression (SVR) Model for Seasonal Time Series Data by Hanifah Muthiah, Umu Sa'adah and Achmad Efendi **Link:** http://ieomsociety.org/proceedings/2021indonesia/574.pdf

19- Efficient Learning Machines by Mariette Awad and Rahul Khanna **Link:** https://link.springer.com/content/pdf/10.1007/978-1-4302-5990-9.pdf

20- PREDICTING THE PRICE OF CRYPTOCURRENCY USING SUPPORT VECTOR REGRESSION METHODS by Saad Ali. Alahmari ISSN (Online): 2454 -7190 Vol.-15, No.-4, April (2020) pp 313-322 ISSN (Print) 0973-8975 **Link:** https://doi.org/10.26782/jmcms.2020.04.00023

**LSTM:**

21- Understanding LSTM –a tutorial into Long Short-Term Memory Recurrent Neural Networks by Ralf C. Staudemeyer and Eric Rothstein Morris **Link:** https://arxiv.org/pdf/1909.09586

22- LONG SHORT-TERM MEMORY Neural Computation by Sepp Hochreiter and Jürgen Schmidhuber **Link:** https://www.bioinf.jku.at/publications/older/2604.pdf

23- Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network by Alex Sherstinsky **Link:** https://arxiv.org/pdf/1808.03314

24- LSTM-based Stock Prediction Modeling and Analysis by Ruobing Zhang **Link:** https://www.atlantis-press.com/article/125971905.pdf

25- A LSTM-Method for Bitcoin Price Prediction: A Case Study Yahoo Finance Stock Market by Ferdiansyah, Siti Hajar Othman, Raja Zahilah Raja Md Radzi, Deris Stiawan, Yoppy Sazaki and Usman Ependi **Link:** https://rie.binadarma.ac.id/file/conference/a-lstm-method-for-bitcoin-price-prediction-a-case-study-yahoo-finance-stock-market-1639027986.pdf

26- LSTM Based Sentiment Analysis for Cryptocurrency Prediction by Xin Huang, Wenbin Zhang, Xuejiao Tang, Mingli Zhang, Jayachander Surbiryala, Vasileios Iosifidis, Zhen Liu and Ji Zhang **Link:** https://arxiv.org/pdf/2103.14804

27- CRYPTOCURRENCY PRICE PREDICTION USING LSTM Mrs. S. Mounika, B. Aravind, K. Sai Charan and B. Kiran e-ISSN: 2582-5208 **Link:** https://www.irjmets.com/uploadedfiles/paper/issue_2_february_2023/33653/final/fin_irjmets1676988272.pdf

**GRU:**

28- Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks by Rahul Dey and Fathi M. Salem **Link:** https://arxiv.org/pdf/1701.05923

29- Gated Recurrent units (GRU) for Time Series Forecasting in Higher Education by Hassan Bousnguar, Amal Battou and Lotfi Najdi ISSN: 2278-0181 **Link:** https://www.ijert.org/research/gated-recurrent-units-gru-for-time-series-forecasting-in-higher-education-IJERTV12IS030091.pdf

30- DESIGN AND IMPLEMENTATION OF CRYPTOCURRENCY PREDICTION MODEL USING GRU ALGORITHM by Dr.M. Tanooj Kumar, Y. Om Sai , K.Geetardha, P.Naga Sandhya and E.Madhan Mohan Reddy ISSN: 2278-4632 Vol-12 Issue-01 No.01: 2022 **Link:** http://junikhyatjournal.in/no_1_Online_22/73.pdf

**Twitter Sentiment Analysis:**

31- FORECASTING CRYPTOCURRENCY PRICES USING DEEP LEARNING: INTEGRATING FINANCIAL, BLOCKCHAIN, AND TEXT DATA by Vincent Gurgul, Stefan Lessmann and Wolfgang Karl Härdle.
**Link:** 2311.14759.pdf

32- CRYPTOCURRENCY PRICE PREDICTION USING TWITTER SENTIMENT ANALYSIS by Haritha G B and Sahana N B. **Link:** https://arxiv.org/pdf/2303.09397.pdf

33- TWITTER SENTIMENT ANALYSIS USING VADER ON PYTHON by Mrs. K. Bhagya Laxmi, Assistant Professor, Department of Computer Science and Engineering, Matrusri Engineering College, Saidabad, Hyderabad, Telangana. **Link:** JETIR2005456.pdf

34- Sentiment Analysis of Twitter Data: A Survey of Techniques by Vishal A. Kharde and S.S. Sonawane. **Link:** 1601.06971v3.pdf

35- SENTIMENT-DRIVEN CRYPTOCURRENCY PRICE PREDICTION: A MACHINE LEARNING APPROACH UTILIZING HISTORICAL DATA AND SOCIAL MEDIA SENTIMENT ANALYSIS by Saachin Bhatt, Mustansar Ghazanfar, and Mohammad Hossein Amirhosseini **Link:** https://repository.uel.ac.uk/download/e8368d407051a4b5f77120ebcf00062914178b7a3595234e1ce3bfe0b78c2953/989054/SENTIMENT-DRIVEN%20CRYPTOCURRENCY%20PRICE%20PREDICTION.pdf

36- Twitter sentiment Analysis for Bitcoin Price Prediction by Sara Abdali and Ben Hoskins **Link:** https://cs229.stanford.edu/proj2021spr/report2/81988764.pdf

**Node.js:**

37- Beginning Node.js by Basarat Ali Syed **Link:** https://edu.anarcho-copy.org/Programming%20Languages/Node/Beginning%20Nodejs.pdf

38- LEARNING Node.js Link: https://www.irjmets.com/uploadedfiles/paper/volume3/issue_3_march_2021/6728/1628083286.pdf

**Flask:**

39- Flask Web Development by Miguel Grinberg. **Link:** https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf

40- Flask Documentation (1.1.x) Release 1.1.4. **Link:** https://flask.palletsprojects.com/_/downloads/en/1.1.x/pdf/