SENTIMENT ANALYSIS OF CRYPTOCURRENCY TWEETS: PREDICTING BITCOIN MARKET SENTIMENT AND PRICE MOVEMENTS USING MACHINE LEARNING

Final Report

Amarpreet Kaur

Student Id. 501213603

Ryerson University

SENTIMENT ANALYSIS OF CRYPTOCURRENCY TWEETS: PREDICTING BITCOIN MARKET SENTIMENT AND PRICE MOVEMENTS USING MACHINE LEARNING

CIND820: Big Data Analytics Project

Amarpreet Kaur

St No.: 501213603

Toronto Metropolitan University, ON

Supervisor: Tamer Abdou

Date of Submission: Jul 17, 2023

Table of Contents

Abstract

Sentiment analysis of tweets on cryptocurrency, particularly Bitcoin, provides valuable insights into market sentiment and public perception, helping investors and enthusiasts make informed decisions. By analyzing the emotional pulse of the community, we can uncover trends, gauge optimism or skepticism, and stay attuned to the ever-changing landscape of digital currencies.

I utilized Machine Learning (ML) and Sentiment Analysis (SA) techniques to predict stock market movements, specifically focusing on cryptocurrencies, with a primary emphasis on Bitcoin. My approach involved gathering data from social media sites, specifically Twitter, and combining it with cryptocurrency data from both Kaggle and the CryptoCompare API for real-time information.

To begin, I collected a dataset of tweets from Kaggle, which served as my source of Twitter data. Additionally, I obtained cryptocurrency data from both Kaggle and the CryptoCompare API, enabling me to access live data for my analysis.

I applied Sentiment Analysis to the tweets I gathered, allowing me to determine the sentiment (extreme positive, positive, extreme negative, negative, and neutral) associated with each tweet. This sentiment data was then used as a feature in my ML models.

For sentiment prediction, I implemented several ML classification models, including Naïve Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and Gradient Boosting (GB). These models were trained using the sentiment data from the tweets to predict the next sentiment.

To predict cryptocurrency prices and changes in price (delta), I employed ML regression models such as Multiple Linear Regression, Decision Tree Regressor, Random Forest Regressor, Support Vector Regressor, Gradient Boosting Regressor, and Neural Network Regressor. These models were trained to predict bitcoin price and change in price based on sentiment.

Furthermore, I utilized Time Series Models to predict the prices of five cryptocurrencies (BTC, ETH, XRP, LTC, USDC) based on their historical data.

The key novelty of my work lies in the integration of multiple SA and ML methods, with a specific focus on extracting additional features from social media, particularly public sentiment from Twitter. By incorporating sentiment analysis and leveraging various ML algorithms, I aimed to enhance the accuracy of cryptocurrency price prediction.

Overall, my project demonstrates the effectiveness of combining SA, ML, and historical data analysis to predict cryptocurrency movements, with an emphasis on Bitcoin, by leveraging social media sentiment and other relevant features.

The research questions delve into the effectiveness of sentiment analysis, compare different machine learning algorithms, assess regression models for cryptocurrency price prediction, and

evaluate the impact of incorporating social media sentiment on prediction accuracy. They address the core aspects of your project and provide opportunities for in-depth analysis and exploration.

1. How does sentiment analysis of tweets relating to Bitcoin correlate with the price dynamics of Bitcoin, specifically examining the influence of sentiment on Bitcoin's price change?

2. How do different machine learning algorithms, such as Naïve Bayes, Support Vector Machine, Decision Tree, Random Forest, Logistic Regression, and Gradient Boosting, perform in sentiment prediction for cryptocurrency-related tweets on Twitter, and which algorithm(s) yield the highest prediction accuracy?

3. What is the comparative effectiveness of various machine learning regression models, including Multiple Linear Regression, Decision Tree Regressor, Random Forest Regressor, Support Vector Regressor, Gradient Boosting Regressor, and Neural Network Regressor, in predicting cryptocurrency prices and changes in price (delta) based on sentiments from tweets?

4. How does the utilization of Time Series Models improve the accuracy of cryptocurrency price prediction for Bitcoin (BTC), Ethereum (ETH), Ripple (XRP), Litecoin (LTC), and USD Coin (USDC) based on their historical data? What are the potential limitations and challenges associated with using Time Series Models for cryptocurrency price prediction?

Keywords: sentiment analysis, cryptocurrency, Bitcoin, machine learning, classification, regression, Twitter, social media, historical data, prediction.

Datasets:

Tweets from Twitter: https://www.kaggle.com/datasets/kaushiksuresh147/bitcoin-tweets

Bitcoin Data for Sentiment Analysis: https://min-api.cryptocompare.com/data/v2/histohour

Historical Crypto Data:
https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory

GitHub Link: https://github.com/Amarpreet3/CIND-820-CAPSTONE

Introduction

Cryptocurrencies have revolutionized the financial landscape, offering opportunities for financial growth and investment. The market operates as a network of economic transactions where

digital coins represent shares or stakes in various digital assets. However, it's important to note that investing in cryptocurrencies carries high risks due to their volatile nature.

Bitcoin, Ethereum, and Ripple are examples of cryptocurrencies that have introduced secure and decentralized digital currency through cryptographic technology. The industry encompasses various disciplines, including technology development, economic factors, financial considerations, regulatory frameworks, and market dynamics. It provides individuals and businesses with diverse opportunities such as blockchain development, cryptocurrency trading, and investment activities.

To succeed in the cryptocurrency industry, staying updated on the latest advancements in blockchain technology, decentralized finance (DeFi), and security measures is crucial. This requires ongoing learning and adaptability to keep pace with the industry's evolving landscape. Having a strong foundation in computer science, finance, and data analysis is vital for understanding the complexities of cryptocurrencies and making well-informed decisions.

Cryptocurrencies have gained significant importance in the financial landscape, but investing in them comes with high risks due to their volatility. However, by staying informed and continuously learning about the industry, individuals and businesses can take advantage of the opportunities it offers.

The accurate prediction of cryptocurrency market trends and sentiments is of great importance to investors. Cryptocurrency market prediction involves the ability to forecast cryptocurrency prices, enabling stakeholders to make informed decisions regarding buying or selling digital assets. Various approaches have been developed for predicting cryptocurrency prices, including fundamental analysis, which relies on publicly available financial information, technical analysis that utilizes historical data and pricing patterns, and the application of machine learning and data mining techniques to analyze large volumes of data. Additionally, sentiment analysis, which predicts market trends based on sentiments expressed in social media posts, such as those on Twitter, has gained significant interest in recent years.

The field of sentiment analysis has become particularly relevant in the context of cryptocurrencies like Bitcoin and other digital assets. The volatile nature and substantial growth of cryptocurrencies have attracted a diverse range of participants, including investors, traders, researchers, and enthusiasts. Sentiment analysis, also known as opinion mining, has emerged as a powerful technique for extracting valuable insights from textual data related to cryptocurrencies.

Sentiment analysis involves the use of natural language processing (NLP) and machine learning algorithms to determine the sentiment expressed in text, such as social media posts, online reviews, news articles, and forum discussions. By quantifying sentiment as positive, negative, or neutral, sentiment analysis allows for the measurement of public opinion and emotional responses toward specific topics.

Twitter, being a leading social media platform, has become a crucial source of real-time information and opinions, making it particularly relevant for sentiment analysis in the context of Bitcoin pricing. Twitter users actively discuss and share their views on cryptocurrencies, including Bitcoin, providing a wealth of sentiment-bearing data. Analyzing Twitter sentiment

allows for the identification of trends, shifts in sentiment, and public perception surrounding Bitcoin, which provides valuable insights into market sentiment and potential price movements.

The topic of sentiment analysis in the context of Bitcoin pricing has garnered significant attention in recent years. Bitcoin, as the first decentralized cryptocurrency, has revolutionized the financial landscape by operating on blockchain technology, enabling secure and transparent transactions without intermediaries. Due to its finite supply and potential for high returns, Bitcoin has captured the interest of individuals seeking alternative investment opportunities and those exploring the future of digital finance.

In the past, the field of sentiment analysis for Bitcoin price prediction faced critical challenges related to data quality, ambiguity, price causality, generalizability, and ethical considerations. Researchers focused on addressing these challenges to advance the field.

Efforts were made to ensure the quality and reliability of sentiment analysis data, considering biases and limitations within platforms like Twitter. Techniques were refined to accurately interpret sentiment from short and contextually ambiguous tweets, incorporating contextual understanding and domain-specific knowledge. Researchers worked on establishing the causal relationship between sentiment and Bitcoin price movements while also exploring generalizability through diverse datasets, multiple cryptocurrencies, and alternative sentiment analysis techniques.

Ethical considerations played a significant role, emphasizing user privacy, responsible data handling, and adherence to ethical guidelines. These advancements enhanced the reliability and applicability of sentiment analysis in predicting Bitcoin prices, empowering market participants in the cryptocurrency landscape.Time series predictions are also conducted in the realm of cryptocurrency. Analysts and researchers utilize historical data and statistical models to forecast future price movements of various digital coins. Techniques such as autoregressive integrated moving average (ARIMA), long short-term memory (LSTM) networks, and other advanced algorithms are employed to analyze patterns and trends in cryptocurrency markets. These predictions offer valuable insights to investors and traders, helping them make informed decisions and develop effective strategies in the dynamic and ever-evolving cryptocurrency landscape.

Given the extensive research in sentiment analysis and Bitcoin pricing, it is unlikely that identical studies have been conducted. Each research endeavor offers a distinct perspective, methodologies, and focus areas. However, numerous studies have explored sentiment analysis of social media, particularly Twitter, in relation to cryptocurrencies and Bitcoin.

Researchers have applied diverse techniques, such as machine learning algorithms, sentiment lexicons, and rule-based approaches, to extract sentiment from Twitter data and investigate its correlation with cryptocurrency prices. Some studies have specifically concentrated on sentiment analysis of Bitcoin-related tweets, examining the influence of sentiment on short-term price fluctuations, or analyzing sentiment patterns during significant market events.

Several studies have extensively explored the relationship between sentiment analysis and cryptocurrency price forecasting, revealing the potential impact of social media sentiment on the volatile cryptocurrency market. Machine learning algorithms, including Support Vector Machine

(SVM), Naive Bayes (NB), and deep learning models like Long-Term-Short-Memory (LSTM), have been utilized to analyze sentiments expressed on platforms like Twitter and predict the prices of specific cryptocurrencies such as Bitcoin (BTC) and Ethereum (ETH). These studies aim to improve price prediction accuracy and gain insights into the complex dynamics of cryptocurrency markets by integrating sentiment analysis with various ML techniques. Techniques like deep learning with LSTM networks, support vector machines, random forests, and recurrent neural networks (RNNs) have shown promising results in forecasting cryptocurrency prices, enabling improved decision-making and risk management in the crypto market.

Literature Review

The paper “Sentiment Analysis and Cryptocurrency Price Forecasting” (2022) [1] explored the relationship between sentiment analysis and cryptocurrency price forecasting. They utilize machine learning algorithms, particularly Support Vector Machine (SVM) and Naive Bayes (NB), to analyze sentiments expressed on social media platforms like Twitter regarding specific cryptocurrencies like Bitcoin (BTC) and Ethereum (ETH). The research aims to classify sentiment accurately, with SVM outperforming NB in this regard. Additionally, the study employs the Long-Term-Short-Memory (LSTM) deep learning algorithm to forecast the prices of BTC and ETH, achieving low error rates for MAE, MSE, and RMSE. The study acknowledges limitations, such as a limited dataset and focus on only two cryptocurrencies, suggesting future research should consider other factors impacting the cryptocurrency market and explore different time-series methods for price forecasting.

The paper “The Impact of Twitter Sentiment Analysis on Bitcoin Price during COVID-19 with XGBoost“(2022) [2] , explores the use of sentiment analysis on Twitter data to predict Bitcoin prices during the COVID-19 pandemic. The XGBoost algorithm is chosen for its effectiveness in classification and regression tasks, while VADER is used for sentiment analysis. The study demonstrates that sentiment analysis has an impact on Bitcoin prices and that models incorporating sentiment analysis outperform those without it. The findings suggest that sentiment analysis on social media platforms like Twitter can be a valuable tool for predicting cryptocurrency prices, and future research should consider alternative algorithms and time window periods for improved accuracy.

In "Forecasting the Early Market Movement in Bitcoin Using Twitter's Sentiment Analysis: An Ensemble-Based Prediction Model" (2021) [3] , the paper investigates sentiment analysis on Twitter data to predict Bitcoin prices during the COVID-19 pandemic. It employs the XGBoost algorithm for classification and regression tasks, as well as VADER for sentiment analysis. The study explores the effectiveness of different machine learning models, such as SVM, Naïve Bayes, and XGBoost, for sentiment analysis and prediction of cryptocurrency market movements based on Twitter data. The paper evaluates and compares the performance of these models in predicting cryptocurrency market movements using sentiment analysis.

"Evaluating Sentiment Classifiers for Bitcoin Tweets in Price Prediction Task" (2019) [4] explores the relationship between Twitter feed on Bitcoin and sentiment analysis, comparing

and evaluating various data mining classifiers and deep learning methods. The study assesses the quality of automated sentiment classification and its correlation with Bitcoin price fluctuations using different machine learning algorithms. However, a potential limitation is the exclusive focus on Twitter data, which may not capture the full sentiment range from the broader population. The paper also lacks addressing potential biases in the Twitter data, such as bot-generated tweets or coordinated campaigns, which could impact sentiment analysis results.

“Time Series Analysis of Blockchain-Based Cryptocurrency Price Changes”(2022) [5] paper applies neural networks and AI to historical records of high-risk cryptocurrency coins, specifically EOS, Dogecoin, Ethereum, and Bitcoin, to train a prediction model for their prices. The study utilizes long short-term memory (LSTM) neural networks and demonstrates varying degrees of accuracy in predicting cryptocurrency prices. However, the paper acknowledges that the nature of stock prices following random walk theory contradicts the experiment's architecture. Critiques of the paper include limited data selection, the need for model comparison, suitability of evaluation metrics, the impact of random walk theory, the justification of future research involving Twitter scraping, and a more comprehensive discussion of limitations. Improvement in these areas would enhance the paper's analysis and understanding of cryptocurrency price prediction.

The paper "Crypto-Currency Price Prediction using Machine Learning"(2022) [6] proposes a system that uses the Facebook Prophet algorithm to predict the prices of major cryptocurrencies with great accuracy. The system aims to help investors make informed decisions on when to invest in cryptocurrencies. However, the paper does not mention any potential shortcomings or limitations of the proposed system. The paper claims that the proposed system can predict the prices of all major cryptocurrencies, but it does not provide evidence to support this claim.

"An Efficient Ensemble Model for Forecasting Time Series Analysis of Crypto Currency Data Using Machine Learning," (2021) [7] paper proposes an ensemble model for time series analysis of crypto currency data using machine learning. The model uses a recurrent neural network with a memory model attached for persistent storage. The proposed model uses the ensemble method for integrating various RNN models for achieving a high level of accuracy. However, the paper has limitations such as lack of novelty, limited evaluation, lack of comparison, lack of explanation, limited scope, and lack of generalizability. The paper also provides an algorithm for data analysis and discusses the importance of data exploration and feature extraction.

"A Review on Digital Coin Investing Predictor," (2022) [8] paper is about the use of machine learning algorithms, such as RNN, LSTM, and GRU, to predict the prices of popular cryptocurrencies like Bitcoin, Ethereum, and Ripple. The study aims to improve the accuracy of predictions by combining RNN and GRU algorithms to form a hybrid. This paper lacks a detailed explanation of the data used, fails to address potential biases in the data or algorithms, lacks transparency in the methodology, and overlooks the ethical implications of using machine learning for cryptocurrency price prediction. Addressing these limitations would enhance the reliability, validity, replicability, and ethical considerations of the study.

"Cryptocurrency Price Prediction Based on ARIMA, Random Forest and LSTM Algorithm " (2023) [9] paper compares and analyzes the prediction accuracy of three algorithms (ARIMA,

Random Forest, and LSTM) on cryptocurrency price prediction using five years of time-series data of Bitcoin, Ether, and Dogecoin. The study finds that the prediction accuracy of LSTM is better than that of Random Forest, and the prediction accuracy of Random Forest is better than the ARIMA model. However, the paper acknowledges that the cryptocurrency market is volatile and dynamic, and its future development trend is influenced by many factors, making it difficult to explore the underlying value of crypto virtual tokens. The paper also highlights the limitations of using stock market variables in cryptocurrency price prediction and suggests the need for more combined models of forecasting methods to achieve better forecasting in the cryptocurrency market. Overall, the paper provides a guideline for investors and practitioners in the cryptocurrency industry, but its methodology and data could be further improved to address limitations and biases.

"Cryptocurrency Valuation: An Explainable AI Approach", (2022) [10] paper introduces the price-to-utility (PU) ratio as a novel market-to-fundamental ratio for cryptocurrency valuation, utilizing unique blockchain accounting methods. It demonstrates that the PU ratio effectively predicts long-term bitcoin returns and offers explanations through machine learning. The paper also presents an automated trading strategy based on the PU ratio, which outperforms traditional strategies. Comparing fundamental-to-market ratios, including price-to-earnings, network value-to-transactions, price-to-market, and price-to-utility ratios, the authors find the price-to-utility ratio to be the most useful in predicting long-term returns. The paper explores the impact of crypto market features, such as decentralization and network characteristics, on valuation. Additionally, it emphasizes the importance of understanding micro behavior and bounded rationality in cryptocurrency trading and highlights the potential of data-driven research. However, potential critiques of the paper include reliance on historical data, which may not reliably reflect future performance given the volatile and evolving nature of the cryptocurrency market. Assumptions and simplifications made in the analysis, such as the superiority of the price-to-utility ratio, may limit the generalizability of the findings. The lack of external validation and comparison with alternative methods raises concerns about the reliability of the conclusions. The paper's focus on Bitcoin and neglect of other cryptocurrencies and macroeconomic factors may limit its applicability and comprehensive understanding of cryptocurrency valuation. Furthermore, the limited exploration of behavioral factors like bounded rationality and heuristic bias restricts its ability to fully capture cryptocurrency market dynamics.

"Cryptocurrency Portfolio Management with Deep Reinforcement Learning" (2016) [11] paper introduces a model-less convolutional neural network approach to portfolio management in the cryptocurrency market. The network is trained using historical price data in a reinforcement manner to maximize cumulative return. The paper discusses problem definition, data selection, network topology, and training process. However, the reliance on historical data and the absence of financial theory are potential limitations. The small training set and limited market signal may affect model generalizability, while back-testing may not accurately reflect real-world performance. Although positive results are achieved compared to benchmarks and other algorithms, potential biases and the exclusion of fundamental properties of cryptocurrencies should be addressed. The paper's training method is based on assumptions that may not reflect the real-world trading environment. Future research should aim to address these limitations and validate the model in real-world scenarios.

Numerous studies have focused on sentiment analysis of social media data, particularly on platforms like Twitter, in relation to cryptocurrencies like Bitcoin. These studies explore various aspects of sentiment analysis and its correlation with Bitcoin pricing. In cryptocurrency research, machine learning algorithms are applied for accurate price prediction and analysis. Regression models, time series analysis, and deep learning techniques leverage historical data to improve price forecasts. However, challenges such as algorithm effectiveness across different cryptocurrencies, data availability and quality, interpretability of models, and ethical considerations regarding bias and market stability need critical assessment. Understanding these factors enables informed decision-making regarding machine learning algorithms in the dynamic and volatile cryptocurrency market.

While no study can be considered an exact replication, the existing body of research provides valuable insights into sentiment analysis and its implications for Bitcoin pricing. This research encompasses various aspects, including sentiment prediction, sentiment analysis in conjunction with price change, and the prediction of Bitcoin price based on historical data. By critically reviewing and expanding upon the existing literature, researchers can enhance our understanding of the intricate relationship between sentiment and the dynamic pricing dynamics of Bitcoin. It is crucial to acknowledge the strengths and limitations of previous studies to further advance the field, refine methodologies, and gain deeper insights into how sentiment analysis can be effectively utilized to predict Bitcoin prices.

In my sentiment analysis project focused on cryptocurrencies, specifically Bitcoin, I am undertaking a comprehensive investigation into the attributes and historical trends of this digital asset. My objective is to develop a comprehensive understanding of its unique characteristics and market dynamics. By immersing myself in this analysis, I aim to construct accurate and robust prediction models by leveraging machine learning algorithms. Through my research, I aspire to bring novel insights to the field of cryptocurrency price prediction, ultimately enhancing the industry's accuracy and dependability in making such forecasts.

Moreover, I recognize the ongoing need for further investigation in the field of cryptocurrency price prediction. Existing studies have predominantly focused on applying deep learning methods to address challenges in cryptocurrency markets, primarily emphasizing Bitcoin (BTC). However, my study aims to broaden the scope by exploring the performance of various deep learning techniques in predicting the prices of other prominent cryptocurrencies. By expanding the range of cryptocurrencies under analysis, I aim to contribute to the understanding of their market dynamics and enhance the accuracy of price predictions for these digital assets.

By analyzing the sentiment of tweets related to cryptocurrencies, especially Bitcoin, I gain valuable insights into market sentiment and public perception, enabling me as an investor and enthusiast to make informed decisions. This approach allows me to uncover trends, assess levels of optimism or skepticism, and stay attuned to the ever-evolving landscape of digital currencies.

In my research, I utilize Machine Learning (ML) and Sentiment Analysis (SA) techniques to predict cryptocurrency market movements, with a primary focus on Bitcoin. To gather data, I collect tweets from Kaggle, serving as my source of Twitter data. Additionally, I obtain cryptocurrency data from both Kaggle and the CryptoCompare API, ensuring access to real-

time information for my analysis. By integrating sentiment analysis with machine learning techniques and leveraging diverse datasets, I aim to improve the accuracy of cryptocurrency price predictions and contribute to the advancement of the field.

In my research, I collected tweets related to cryptocurrencies and applied sentiment analysis to determine the sentiment associated with each tweet. This sentiment data was then incorporated as a feature in my machine learning (ML) models. For sentiment prediction, I implemented several ML classification models, including Naïve Bayes, Support Vector Machine, Decision Tree, Random Forest, Logistic Regression, and Gradient Boosting. These models were trained using the sentiment data from the tweets to predict subsequent sentiment.

To predict cryptocurrency prices and price changes, I utilized ML regression models such as Linear Regression, Decision Tree Regressor, Random Forest Regressor, Support Vector Regressor, Gradient Boosting Regressor, and Neural Network Regressor. These models were trained to predict Bitcoin price and changes in price based on sentiment.

In addition, I employed Time Series Models to predict the prices of five cryptocurrencies: BTC, ETH, XRP, LTC, and USDC, based on their historical data. This approach allowed me to capture the temporal patterns and trends in cryptocurrency prices.

The key novelty of my research lies in the seamless integration of multiple sentiment analysis and machine learning methods, presenting a comprehensive approach to cryptocurrency price prediction. I specifically focus on extracting additional features from social media platforms, with Twitter being the primary source for capturing public sentiment.

By leveraging sentiment analysis techniques, I tap into the vast pool of social media data to uncover valuable insights into the collective emotions and opinions surrounding cryptocurrencies. This innovative approach allows me to harness the wisdom of the crowd and gain a deeper understanding of market sentiment, resulting in more accurate and reliable price predictions.

To achieve this, I carefully select and fine-tune a diverse range of ML algorithms, optimizing their performance for cryptocurrency price prediction. The combination of sentiment analysis and ML algorithms empowers me to effectively analyze and interpret the wealth of information embedded within social media discussions.

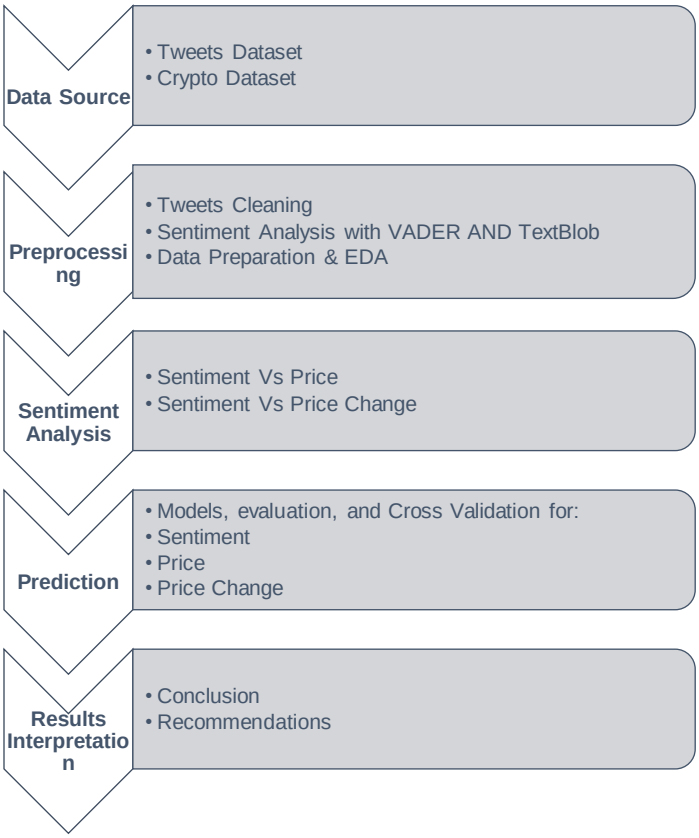
By extracting and integrating these additional features from social media, my research goes beyond traditional methods of price prediction that rely solely on historical market data. The incorporation of sentiment analysis provides a dynamic and real-time perspective on market sentiment, enabling me to capture the ever-changing landscape of cryptocurrency markets.

Through this comprehensive integration of sentiment analysis and machine learning techniques, my research aims to enhance the accuracy and reliability of cryptocurrency price prediction. By leveraging the power of sentiment analysis and utilizing various ML algorithms, I strive to provide valuable insights for investors, traders, and enthusiasts, assisting them in navigating the complex and volatile world of cryptocurrencies.

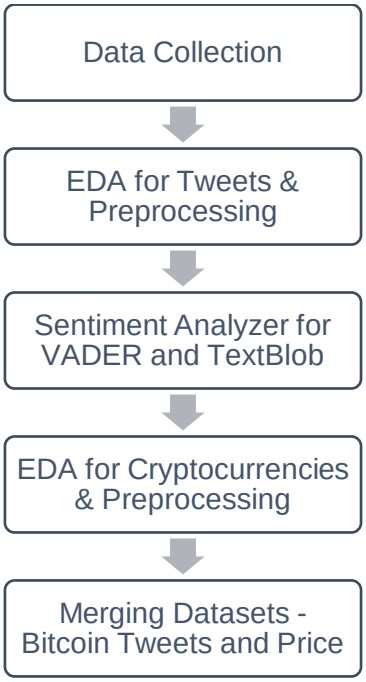
Methodology and Approach

I will divide this project into sections to investigate each research question separately, as each research question requires a different methodology. Here is the general methodology for the overall project:

Methodology and Approach for whole project



Section 0: Methodology for Data collection and Pre-processing of Datasets



Data Collection:

Data collection is the process of gathering information or data from various sources, such as databases, websites, or sensors. It involves systematically capturing and organizing data to enable analysis and decision-making for twitter and stock market for cryptocurrencies.

EDA for Tweets & Preprocessing:

Exploratory Data Analysis (EDA) for tweets involves analyzing and visualizing the data to uncover patterns, trends, and insights. Preprocessing is the initial step in data cleaning, which includes removing noise, handling missing values, and transforming the data into a suitable format for further analysis.

EDA for Cryptocurrencies & Preprocessing:

EDA for cryptocurrencies involves examining historical price data, market trends, and trading volumes to gain insights into their behavior. Preprocessing in this context may involve normalizing the data, handling outliers, and dealing with missing values to ensure accurate analysis.

Merging Datasets - Bitcoin Tweets and Price:

Merging datasets refers to combining two or more datasets into a single dataset, based on a common variable. In the context of Bitcoin, merging Bitcoin tweets and price datasets allows for

the analysis of how social media sentiment relates to the cryptocurrency's price movements, enabling deeper insights into market dynamics.

Section 1: Methodology for Research Question 1

How does sentiment analysis of tweets relating to Bitcoin correlate with the price dynamics of Bitcoin, specifically examining the influence of sentiment on Bitcoin's price change?

Sentiment Analysis Vs Price:

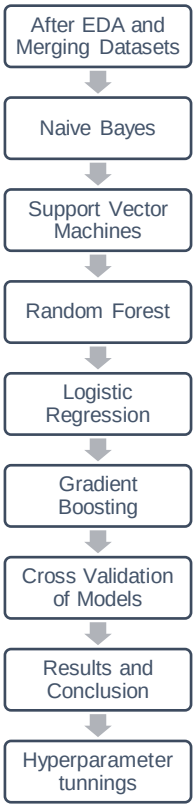
Sentiment analysis examines the overall sentiment or emotional tone expressed in text data, such as social media posts or customer reviews. When applied to financial markets, sentiment analysis can help identify the prevailing mood of investors, which may impact price movements.

Sentiment Analysis Vs Price Change (Delta):

Analyzing the price change (delta) in conjunction with sentiment analysis allows for a deeper understanding of how sentiment influences market dynamics. By correlating sentiment with price movements, it becomes possible to identify patterns or trends that suggest a relationship between sentiment and changes in asset prices.

Section 2: Methodology for Research Question 2

How do different machine learning algorithms, such as Naïve Bayes, Support Vector Machine, Decision Tree, Random Forest, Logistic Regression, and Gradient Boosting, perform in sentiment prediction for cryptocurrency-related tweets on Twitter, and which algorithm(s) yield the highest prediction accuracy?



Machine Learning models

After EDA and merging datasets, various machine learning models can be employed for analysis, including Naive Bayes, Support Vector Machines, Random Forest, Logistic Regression, and Gradient Boosting. These models can help uncover patterns and make predictions based on the combined data.

Cross-validation

Cross-validation of models is crucial to evaluate their performance and ensure their generalizability. It involves dividing the data into training and validation sets, allowing for unbiased assessment and fine-tuning of the models.

Results and Conclusion

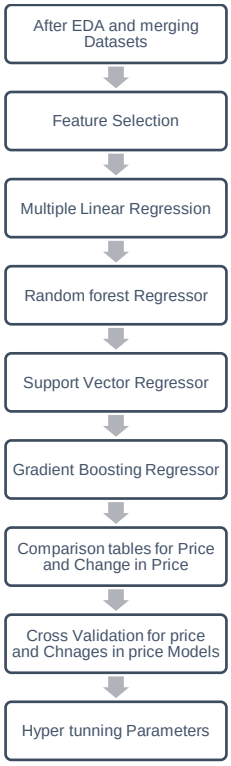
Analyzing the results obtained from the models, along with the merged datasets, can provide valuable insights into the relationship between variables, sentiment, and price changes. These insights form the basis for drawing conclusions about the impact of sentiment on asset prices.

Hyperparameter tunnings

To optimize the performance of the models, hyperparameter tuning is essential. By adjusting hyperparameters such as learning rates, regularization strengths, or number of estimators, the models can be fine-tuned to improve their predictive accuracy and robustness.

Section 3: Methodology for Research Question 3

What is the comparative effectiveness of various machine learning regression models, including Multiple Linear Regression, Decision Tree Regressor, Random Forest Regressor, Support Vector Regressor, Gradient Boosting Regressor, and Neural Network Regressor, in predicting cryptocurrency prices and changes in price (delta) based on sentiments from tweets?

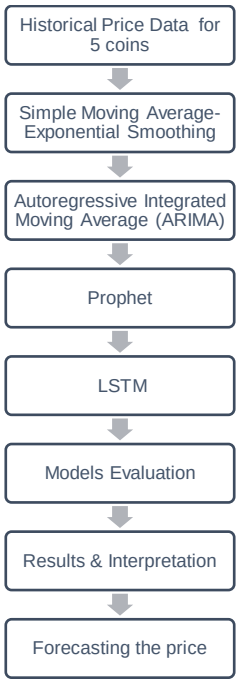


After performing EDA and merging datasets, the next steps include feature selection and applying regression models such as Linear Regression, Random Forest Regressor, Support Vector Regressor, and Gradient Boosting Regressor. Feature selection helps identify the most relevant variables that contribute to the prediction of price or change in price. These regression models are then trained on the selected features to make predictions. To evaluate the performance of these models, comparison tables can be created to assess how well they

predict the actual price and changes in price. This analysis provides insights into the effectiveness of different regression techniques in capturing the relationship between variables and predicting price dynamics.

Section 4: Methodology for Research Question 4

How does the utilization of Time Series Models improve the accuracy of cryptocurrency price prediction for Bitcoin (BTC), Ethereum (ETH), Ripple (XRP), Litecoin (LTC), and USD Coin (USDC) based on their historical data? What are the potential limitations and challenges associated with using Time Series Models for cryptocurrency price prediction.



After conducting exploratory data analysis (EDA) and merging the historical price data for five coins, several time series forecasting techniques can be employed to predict future price trends. These include the Simple Moving Average (SMA) and Exponential Smoothing for capturing trend and smoothing effects, Autoregressive Integrated Moving Average (ARIMA) for modeling time-dependent behavior, LSTM (Long Short-Term Memory) for capturing long-term dependencies in sequential data, and Prophet, a forecasting model developed by Facebook. These models are then evaluated based on their performance metrics, such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), and the results are interpreted to assess the accuracy and effectiveness of each technique. With these insights, forecasting of the price for the given coins can be conducted, providing valuable information for decision-making and market analysis.

Section 0

Dataset

In this project, I obtained a dataset of Bitcoin tweets from Kaggle (https://www.kaggle.com/datasets/kaushiksuresh147/bitcoin-tweets). I downloaded the dataset and divided it into smaller chunks to facilitate storage due to its large size. Each chunk was saved separately with a unique name for analysis purposes. To continue with the analysis, I concatenated all the tweet files into a single file named "bitcoinTweets.csv".

Additionally, I collected Bitcoin price data using the CryptoCompare API. The data was retrieved using the hourly historical data endpoint. The Bitcoin price data is available in CSV format and can be accessed through the following link: https://min-api.cryptocompare.com/data/v2/histohour.

For Time Series Modeling, a dataset called "cryptocurrency.csv" was compiled by collecting historical data for five different cryptocurrencies: Bitcoin, Ethereum, XRP, Litecoin, and USD Coin. The data for each cryptocurrency was obtained from reputable and open-access platforms to ensure reliability and accuracy. Subsequently, the datasets for these individual cryptocurrencies were merged into a single file, "cryptocurrency.csv." This combined dataset offers a comprehensive and diverse collection of cryptocurrency price history, enabling a holistic analysis of the cryptocurrency market. The dataset can be accessed and downloaded from the following link: https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory

These datasets can be utilized for conducting literature reviews, studying trends, and exploring various aspects of cryptocurrency behavior. The datasets provide a valuable resource for investigating price patterns, volatility, market capitalization, trading volume, and other relevant metrics across multiple cryptocurrencies.

The chosen programming language for this project is Python, which offers a wide range of libraries for data analysis and model building. Key libraries used in this project include NumPy for efficient numerical operations, Pandas for data manipulation and analysis, scikit-learn for machine learning tasks, SciPy for scientific computing, and Matplotlib for data visualization. These libraries together provide a comprehensive toolkit for data cleaning, preparation, modeling, and plotting.

EDA for Tweets

Data Description

The provided data is stored in a Pandas DataFrame object, which is a tabular data structure commonly used for data analysis in Python. The DataFrame consists of 169,761 rows and 13

columns. Each column represents a specific aspect of the data, such as user information, tweet content, and metadata.

To gain more insights into the data, the `info()` method can be used. It provides a concise summary of the DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169761 entries, 0 to 169760
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   user_name         169751 non-null  object
 1   user_location     84090 non-null   object
 2   user_description  158666 non-null  object
 3   user_created      169761 non-null  object
 4   user_followers    169761 non-null  float64
 5   user_friends      169761 non-null  float64
 6   user_favourites   169761 non-null  float64
 7   user_verified     169761 non-null  bool
 8   date              169761 non-null  datetime64[ns]
 9   text              169761 non-null  object
 10  hashtags          168954 non-null  object
 11  source            168954 non-null  object
 12  is_retweet        168954 non-null  float64
dtypes: bool(1), datetime64[ns](1), float64(4), object(7)
memory usage: 15.7+ MB
```

Data Dictionary

The dataset can be described briefly as follows.

No	Columns	Descriptions	Data Type
1	user_name	The name of the user, as they've defined it.	object (string)
2	user_location	The user-defined location for this account's profile.	object (string)
3	user_description	The user-defined UTF-8 string describing their account.	object (string)
4	user_created	Time and date, when the account was created.	object (string)
5	user_followers	The number of followers an account currently has.	float64 (number)
6	user_friends	The number of friends an account currently has.	float64 (number)

7	user_favourites	The number of favorites an account currently has	float64 (number)
8	user_verified	When true, indicates that the user has a verified account	bool (boolean)
9	date	UTC time and date when the Tweet was created	datetime64[ns]
10	text	The actual UTF-8 text of the Tweet	object (string)
11	hashtags	All the other hashtags posted in the tweet along with #Bitcoin & #btc	object (string)
12	source	Utility used to post the Tweet; Tweets from the Twitter website have a source value - web	object (string)
13	is_retweet	Indicates whether this Tweet has been Retweeted by the authenticating user.	float64 (number)

Tweets Cleaning

I cleaned the Twitter data to remove any hashtags, links, images, and videos from the tweets. Additionally, I collected the sentiment scores of each tweet as described in the previous section.

To clean the Twitter data, I utilized several libraries in Python. Firstly, I used Pandas to read the raw CSV file and load the data into a DataFrame. The `re` library was employed to remove hashtags, URLs, and mentions from the tweets using regular expressions. The `string` module allowed me to remove punctuation marks. For text preprocessing, I utilized NLTK library's stopwords to filter out common English words and the `PorterStemmer` class for stemming the tokens. The cleaned text was then joined back into a single string and updated in the DataFrame. Finally, I saved the cleaned data to a new CSV file. The `tqdm` library was used to create a progress bar for better visualization during the cleaning process. These libraries, including Pandas, `re`, `string`, NLTK, and `tqdm`, provided the necessary tools and functionalities to efficiently clean and preprocess the Twitter data, making it ready for further analysis. Finally, I saved the cleaned data to a new CSV file named 'bitcoinTweets_cleaned.csv'.

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
100%|██████████| 169761/169761 [11:21<00:00, 249.02it/s]
Cleaned tweets data saved to 'bitcoinTweets_cleaned.csv'.
```

Duplicates removal

I checked the shape of the DataFrame before removing duplicates. This was done using the drop_duplicates() method, which identifies duplicate rows based on all columns. By specifying inplace=True, the DataFrame was modified directly to remove the duplicates. This process

ensured that the DataFrame no longer contained any duplicate rows, improving the integrity and accuracy of the data.

```
tweet shape before droping duplicates (169761, 13)
Duplicates removed from the cleaned tweets data.
tweet shape after droping duplicates (168685, 13)
```

Overall structure looks like below: The `.head()` function displays the first few rows of the DataFrame, providing a quick overview of the data and its structure.

user_name	user_location	user_description	user_created	user_followers	user_friends	user_favourites	user_verified	date	text	hashtags	source
Irk	Vancouver, WA	Irk started investing in the stock market in 1...	2018-08-11 03:17:00	116.0	8.0	4580.0	False	2023-02-25 23:59:00	bitcoin btc rest crypto ye bitcoin cryptocurr ...	['Bitcoin', 'crypto', 'NeedsMoreCrash']	Twitter Web App
Xiang Zhang	NaN	Professional Software Engineer ðŸ'»ðŸ"ŠCrypto ...	2011-01-11 01:37:00	42.0	22.0	5.0	False	2023-02-25 23:59:00	retriev invest fund current ongo tidexcoin kic...	['Tidexcoin', 'Kicurrency', 'LMY', 'GMK', 'SYR...	Twitter for iPhone
Rhizoo	NaN	researcher. local maxima dunningâ€"kruger spec...	2019-04-03 18:09:00	778.0	627.0	32005.0	False	2023-02-25 23:59:00	bull save monthli thread today good shit bitco...	['bitcoin']	Twitter Web App
Hari Marquez	Las Vegas, NV	Donâ€™t trust, verify. #Bitcoin | El Salvador ...	2014-01-17 23:04:00	222.0	521.0	13052.0	False	2023-02-25 23:59:00	el salvador shape futur bitcoin membvk32cn	['Bitcoin']	Twitter Web App
Bitcoin Candle Bot	Brazil	Robot that posts the closure of the bitcoin da...	2021-01-06 01:36:00	40.0	4.0	1.0	False	2023-02-25 23:59:00	candl day 25022023 close open 2319406 high 232...	['Bitcoin', 'Candle', 'BearMarket']	Bitcoin Candle Bot

Filter Tweets and hashtag for Bitcoin

The objective is to filter tweets related to Bitcoin based on specific keywords and hashtags. The `bitcoin_keywords` list contains relevant keywords, such as 'bitcoin', 'btc', and 'crypto', while the `bitcoin_hashtags` list includes corresponding hashtags, like '#bitcoin', '#btc', and '#cryptocurrency'. Rows with missing values in the 'text' column are dropped using the `dropna()` function. Subsequently, the filtering condition is applied to the DataFrame, selecting rows where the 'text' column contains any of the specified keywords or hashtags.

Sentiment analysis with Vader

I performed sentiment analysis using the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool. First, I installed the `vaderSentiment` library and imported necessary libraries, including Pandas, `SentimentIntensityAnalyzer` from `vaderSentiment.vaderSentiment`, and `tqdm`.

The tweets were already cleaned and stored in the 'text' column of the DataFrame `df_clean`, which is a copy of the previously filtered DataFrame, `filtered_tweets_df`, I handled NaN values in the 'text' column by replacing them with empty strings using the `fillna()` method.

Next, I initialized a SentimentIntensityAnalyzer from the `vaderSentiment` library. Using a loop, I performed sentiment analysis on each tweet in the 'text' column. The sentiment analysis results, representing the overall sentiment score for each tweet, were stored in the 'compound' list. I assigned a compound score to each tweet using the `polarity_scores()` method from the SentimentIntensityAnalyzer. Finally, I added the compound scores as a new column, 'compound', to the DataFrame.

Score Calculation

To calculate scores for each tweet, a loop was utilized to iterate over the cleaned DataFrame. For each row, the VADER sentiment analysis tool determined the compound sentiment score. This score was subsequently weighted by the number of followers and favorites associated with the user who posted the tweet. The resulting scores were stored in a list called "scores". To incorporate these scores into the DataFrame, a new column named "score" was added. The decision to use the number of followers and favorites as weighting factors was based on their ability to provide insights into the potential reach and engagement of the tweet, thus influencing the overall sentiment impact of the message.

After adding the 'compound' and 'score' columns, the dataset is now updated to include these additional columns that contain sentiment analysis information.

user_created	user_followers	user_friends	user_favourites	user_verified	date	text	hashtags	source	is_retweet	compound	score
2018-08-11 03:17:00	116.0	8.0	4580.0	False	2023-02-25 23:59:00	bitcoin btc rest crypto ye bitcoin cryptocurr ...	['Bitcoin', 'crypto', 'NeedsMoreCrash']	Twitter Web App	0.0	-0.4019	-215409.1563
2011-01-11 01:37:00	42.0	22.0	5.0	False	2023-02-25 23:59:00	retriev invest fund current ongo tidexcoin kic...	['Tidexcoin', 'Kicurrency', 'LMY', 'GMK', 'SYR...	Twitter for iPhone	0.0	0.0000	0.0000

Sentiment Level

For Sentiment Level, the code begins by importing the necessary libraries, including `nltk`, and downloading the VADER lexicon using `nltk.download('vader_lexicon')`. Then, the `SentimentIntensityAnalyzer` is initialized.

Next, the code calculates compound scores for the 'text' column in the DataFrame `concatenated_df` using a lambda function and the `polarity_scores()` method from the `SentimentIntensityAnalyzer`. The compound score represents the overall sentiment polarity of each text.

To assign sentiment levels, the code determines the range of compound values by finding the minimum and maximum scores. It then defines custom bin edges based on quantiles using `np.linspace()`. Additionally, labels are specified for each sentiment level: 'Extreme Negative', 'Negative', 'Neutral', 'Positive', and 'Extreme Positive'.

Using `pd.cut()`, the sentiment levels are assigned to the 'sentiment_level' column based on the custom bins. The criteria for each sentiment level are as follows: 'Extreme Negative' for the lowest compound values, 'Negative' for moderately negative scores, 'Neutral' for scores near zero, 'Positive' for moderately positive scores, and 'Extreme Positive' for the highest compound values. Here is the count of each sentiment level.

```
Neutral             93170
Positive            35921
Extreme Positive    17344
Negative            15904
Extreme Negative     5316
Name: sentiment_level, dtype: int64
```

Sentiment Analysis with TextBlob

The code begins by importing the required library, `TextBlob`, using the `from textblob import TextBlob` statement.

Next, two empty lists, `polarity` and `subjectivity`, are initialized to store the polarity and subjectivity values calculated for each tweet.

The code then utilizes a loop to iterate through each tweet in the `concatenated_df_2` DataFrame. For each tweet, the sentiment analysis is performed using TextBlob's `sentiment` attribute. The polarity and subjectivity scores are retrieved using `analysis.sentiment.polarity` and `analysis.sentiment.subjectivity`, respectively. In case of any errors during sentiment analysis, a default polarity and subjectivity score of 0 is appended to the corresponding lists.

By the end of the loop, the `polarity` and `subjectivity` lists contain the calculated scores for each tweet.

This code effectively calculates the polarity and subjectivity of each tweet using TextBlob's sentiment analysis, ensuring that even in case of errors, default scores are provided for further analysis.

Dividing the Data in smaller chunks and save it as size of over all data is apporx 140 MB

The code divides a dataset into smaller subsets based on a desired size, saves each subset as a separate CSV file, and prints the subset file name, size, and number of rows. This allows for efficient handling and organization of large datasets.

After performing sentiment analysis using VADER and TextBlob and adding the necessary columns, here is a summary of the dataset [using the .info()]:

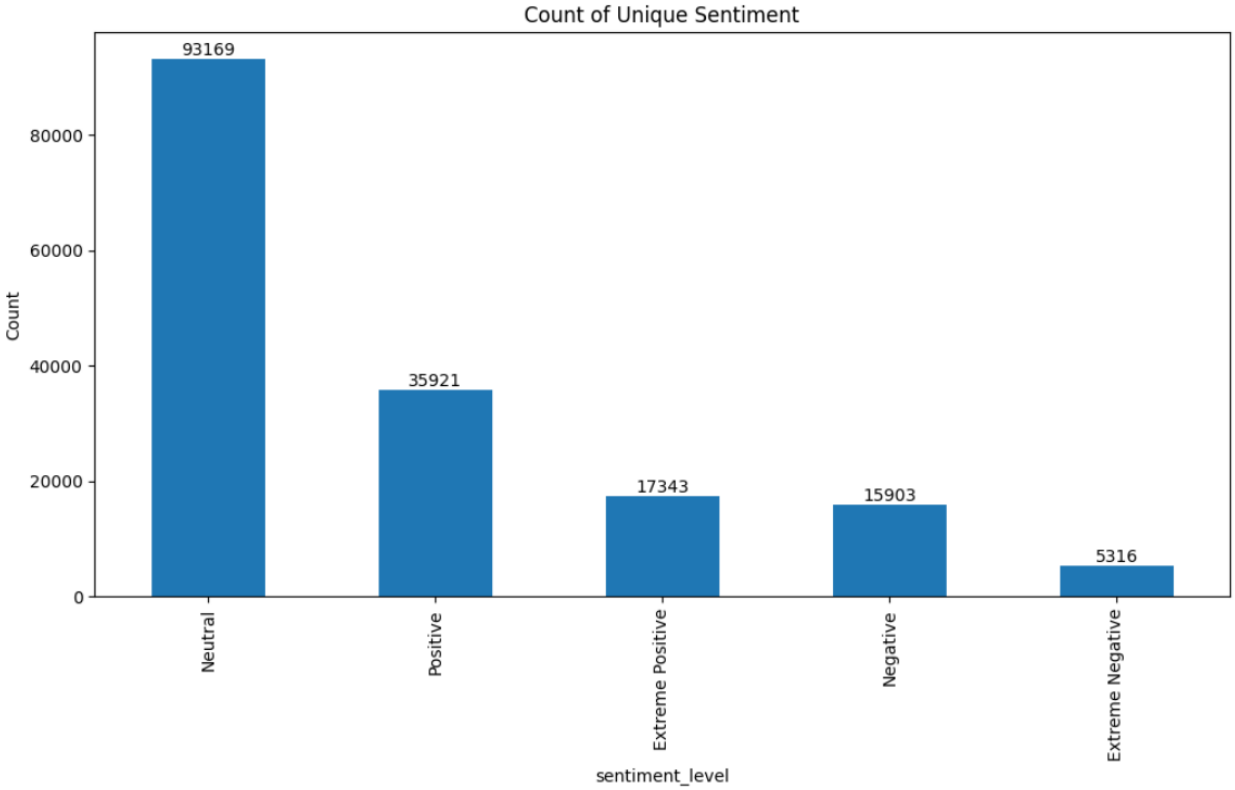
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 167652 entries, 0 to 27941
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   user_name         167642 non-null  object
 1   user_location     83349 non-null   object
 2   user_description  157323 non-null  object
 3   user_created      167652 non-null  object
 4   user_followers    167652 non-null  float64
 5   user_friends      167652 non-null  float64
 6   user_favourites   167652 non-null  float64
 7   user_verified     167652 non-null  bool
 8   date              167652 non-null  object
 9   text              167652 non-null  object
 10  hashtags          167142 non-null  object
 11  source            167142 non-null  object
 12  is_retweet        167142 non-null  float64
 13  compound          167652 non-null  float64
 14  score             167652 non-null  float64
 15  sentiment_level   167652 non-null  object
 16  polarity          167652 non-null  float64
 17  subjectivity      167652 non-null  float64
dtypes: bool(1), float64(8), object(9)
memory usage: 23.2+ MB
None
```

Shape of the dataset: Number of rows: 167652 and Number of columns: 18

Size of the dataset (number of elements): 3017736

Count of Unique Sentiment:

Data



Description

To obtain meta information about the numeric data, including the presence of extreme values, the `.describe().transpose()` method was used. This provides summary statistics such as count, mean, standard deviation, minimum, quartiles, and maximum values for each numeric column in the dataset. Analyzing this information can help identify any outliers or extreme values in the dataset.

	count	mean	std	min	25%	50%	75%	max
user_followers	167652.0	1.069845e+04	1.317029e+05	0.000000e+00	119.0	556.000000	1956.000000	1.878937e+07
user_friends	167652.0	7.742461e+02	2.691621e+03	0.000000e+00	9.0	123.000000	606.000000	2.542760e+05
user_favourites	167652.0	6.267079e+03	2.192673e+04	0.000000e+00	11.0	281.000000	3300.000000	1.083014e+06
is_retweet	167142.0	0.000000e+00	0.000000e+00	0.000000e+00	0.0	0.000000	0.000000	0.000000e+00
compound	167652.0	1.055299e-01	3.395308e-01	-9.913000e-01	0.0	0.000000	0.361200	9.840000e-01
score	167652.0	3.260548e+07	1.440558e+09	-9.459832e+10	0.0	0.000000	7648.344025	1.350093e+11
polarity	167652.0	6.998889e-02	2.236899e-01	-1.000000e+00	0.0	0.000000	0.100000	1.000000e+00
subjectivity	167652.0	2.466140e-01	2.712997e-01	0.000000e+00	0.0	0.223016	0.418182	1.000000e+00

Null/NAN values

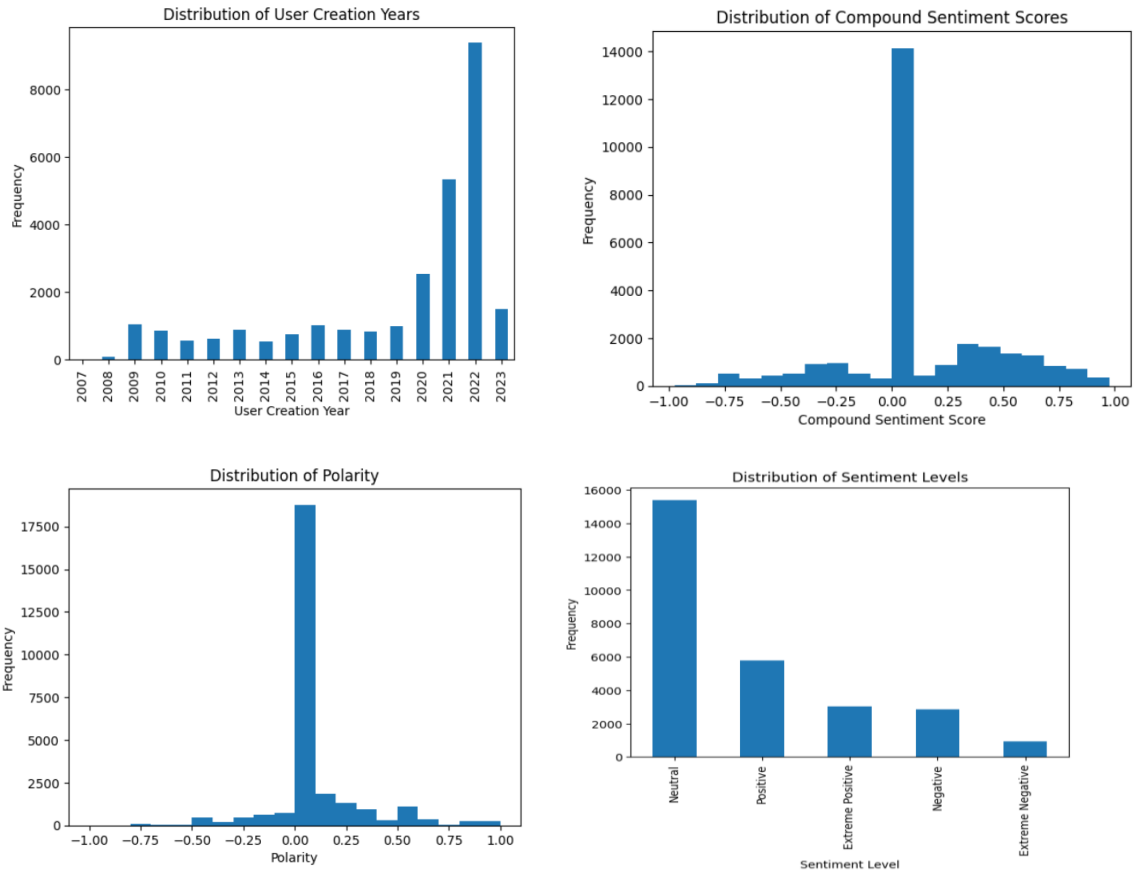
The code `.isnull().sum()` checks for the presence of null values in each column of the DataFrame and returns the total number of null values in each column.

```
user_name            10
user_location     84303
user_description  10329
user_created          0
user_followers        0
user_friends          0
user_favourites       0
user_verified         0
date                  0
text                  0
hashtags            510
source              510
is_retweet          510
compound              0
score                 0
sentiment_level       0
polarity              0
subjectivity          0
dtype: int64
```

Null values in the columns are disregarded as they hold no substantial significance for subsequent analysis and remain unattended.

Univariate Analysis

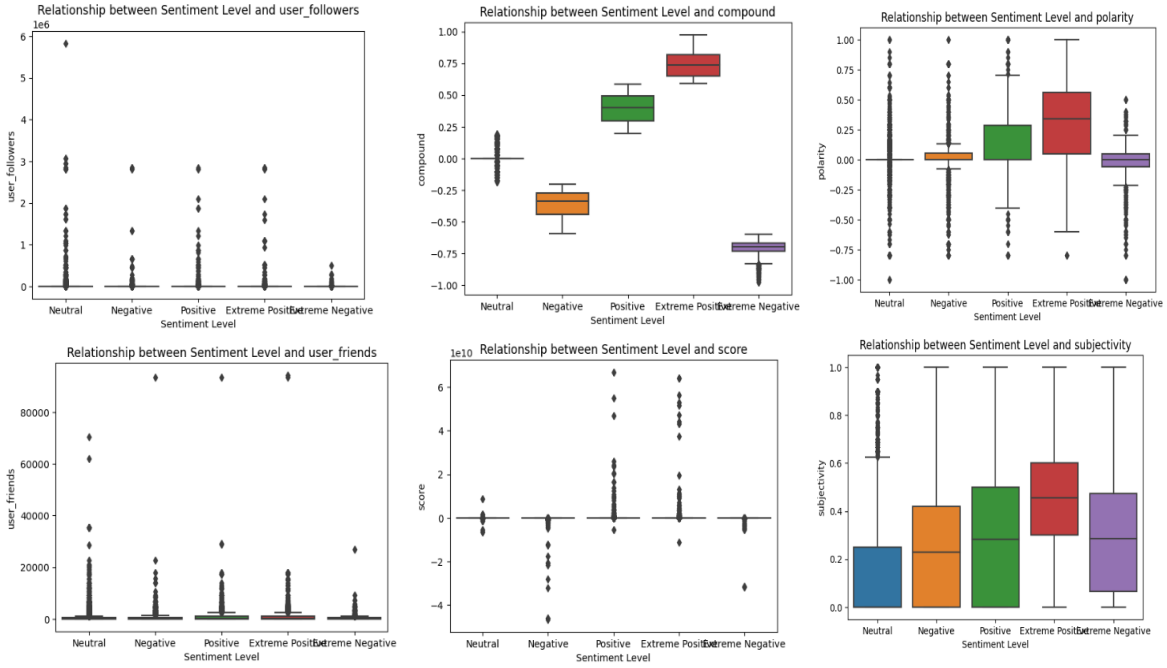
Univariate analysis is a statistical analysis technique that focuses on analyzing individual variables in isolation. It involves examining the distribution, central tendency, dispersion, and other characteristics of a single variable. This analysis provides insights into the variable's behavior, patterns, and summary statistics. By conducting univariate analysis, we can gain a better understanding of the variable's range, outliers, skewness, and other important aspects, which can help inform further analysis and decision-making. Univariate analysis is a fundamental step in exploratory data analysis and provides a foundation for more complex multivariate analyses. Here are few of the observations:



Notable observations include an increase in user activity on Twitter, with a peak in user creation in 2022. Furthermore, a majority of individuals express positive or neutral opinions towards Bitcoin.

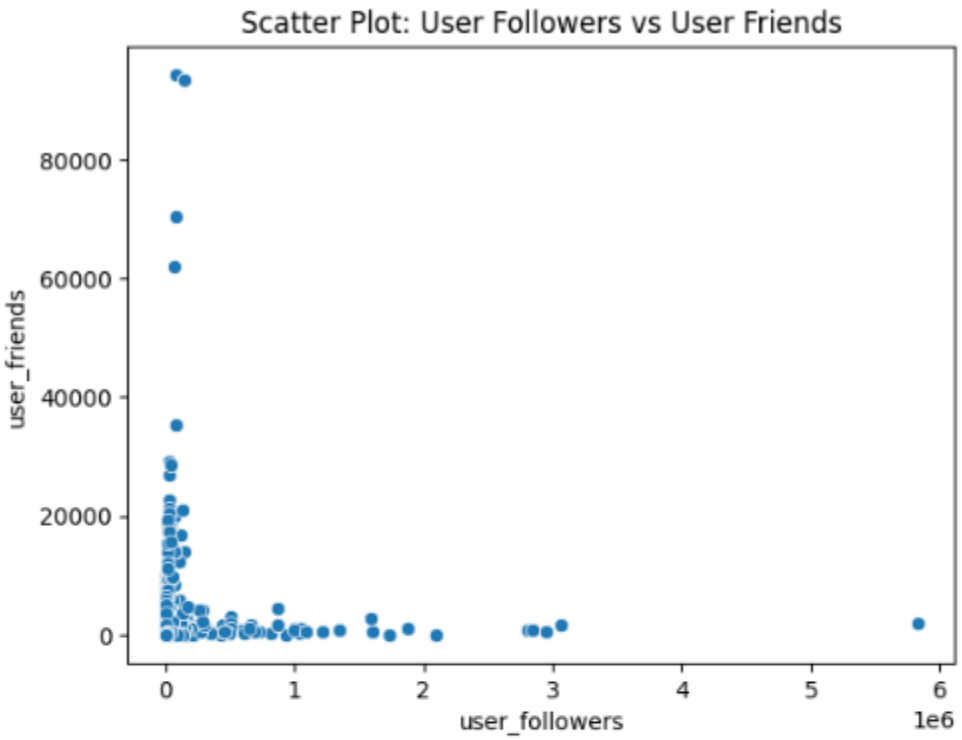
Bivariate analysis

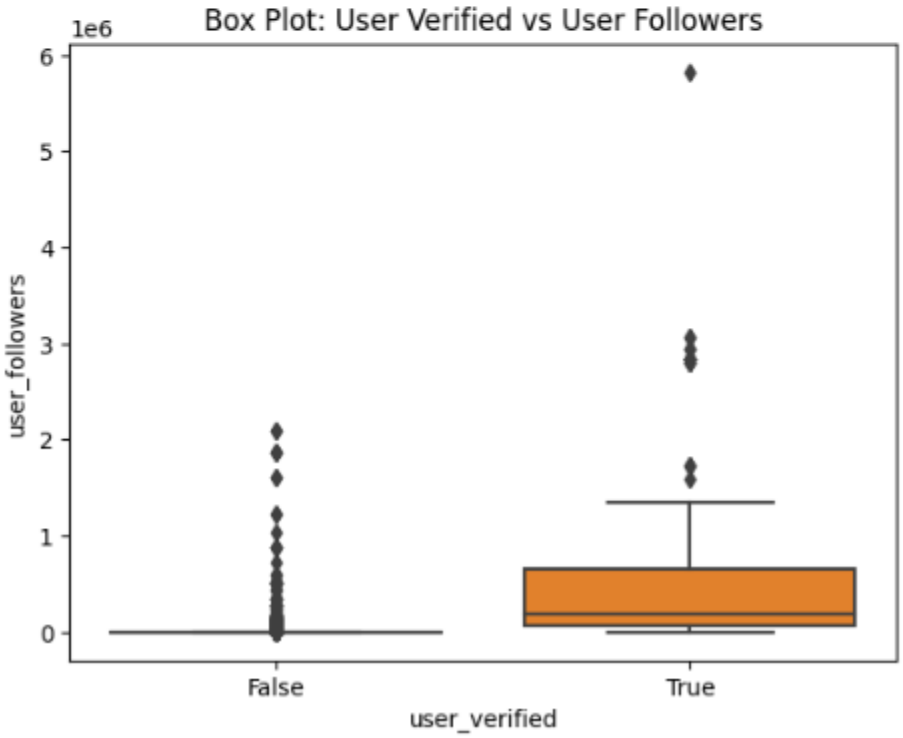
Bivariate analysis is an important technique used to explore the relationship between variables, including sentiment level and other variables. In this context, bivariate analysis can be conducted to examine how sentiment level relates to other factors of interest, such as user demographics, tweet characteristics, or market trends. This analysis can involve methods such as cross-tabulation, correlation analysis, or visualizations like stacked bar charts or scatter plots. By examining the relationship between sentiment level and other variables, we can uncover insights into the factors that influence sentiment and gain a deeper understanding of the underlying patterns and dynamics in the data.



The boxplot reveals that the compound scores for negative, positive, and extreme positive sentiments have a noticeable spread. Additionally, the polarity of extreme positive sentiments appears to be higher compared to the other categories.

Here is Scatter Plot: User Followers vs User Friends and Box Plot: User Verified vs User Followers





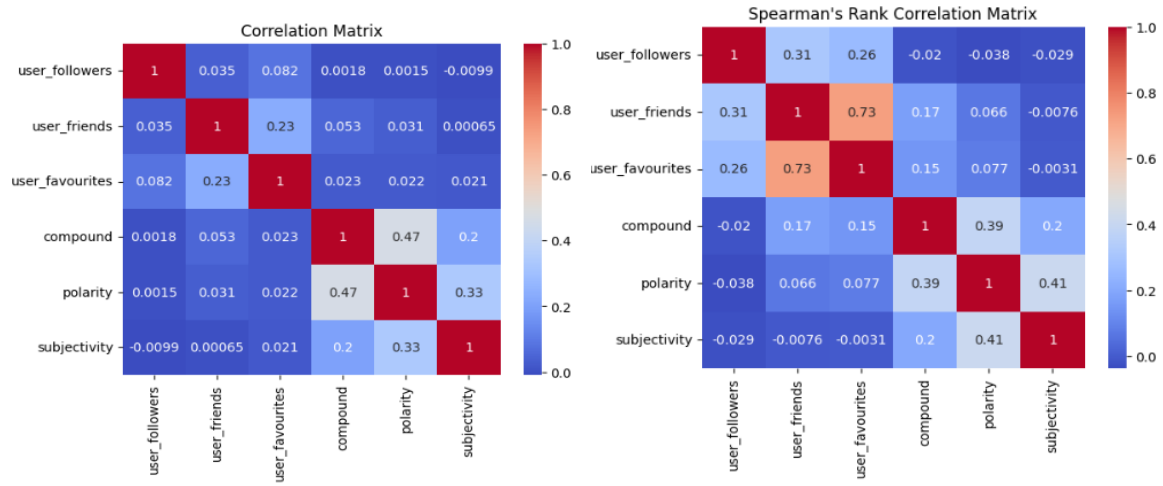
The box plot indicates that a significant proportion of users are verified, as their representation stands out compared to other categories.

Data Visualizations

Data Visualization plays a crucial role in enhancing understanding by presenting data through visual representations. It enables the identification of patterns, trends, and relationships that may be difficult to perceive from raw data alone. By using charts, graphs, and other visual elements, data visualization simplifies complex information and allows for more intuitive interpretation. It helps to communicate insights, support decision-making, and uncover hidden patterns, leading to a deeper understanding of the data and facilitating effective communication of findings to a broader audience.

Correlation Matrix for Selected features

The code utilizes pandas, seaborn, and matplotlib.pyplot libraries for correlation analysis. Relevant columns, including 'user_followers', 'user_friends', 'user_favourites', 'compound', 'polarity', and 'subjectivity', are selected for analysis. The correlation matrix is calculated to measure the relationships between these variables. A heatmap is created using seaborn to visualize the correlation matrix, highlighting the strength and direction of the correlations. Spearman's rank correlation matrix is also calculated and visualized as a heatmap. Highly correlated variable pairs are identified based on a specified threshold, and the pairs are printed as output. This analysis helps understand the degree of association between variables, allowing for insights into potential dependencies or patterns within the dataset.



The correlation matrix highlights a strong correlation between polarity and compound scores, suggesting a close relationship between the overall sentiment and the strength of sentiment expressed. Additionally, a notable correlation between user favorites and friends implies that individuals tend to favor their friends on the platform.

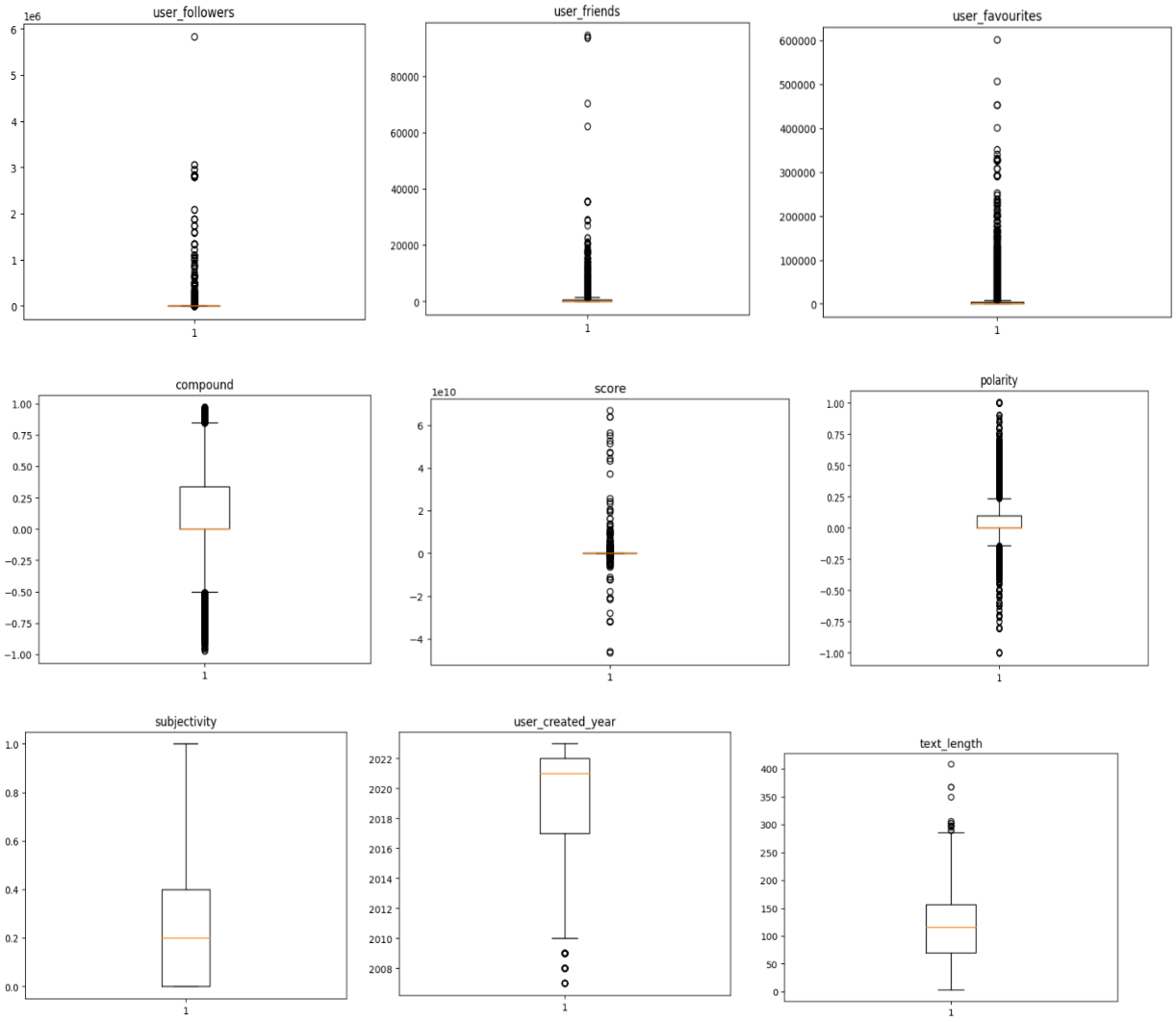
Outlier and anomaly detection

The code makes use of the pandas and matplotlib.pyplot libraries for outlier analysis. The numerical columns in the DataFrame are selected, and descriptive statistics are computed for these columns using the describe() function. Outliers are identified based on the interquartile

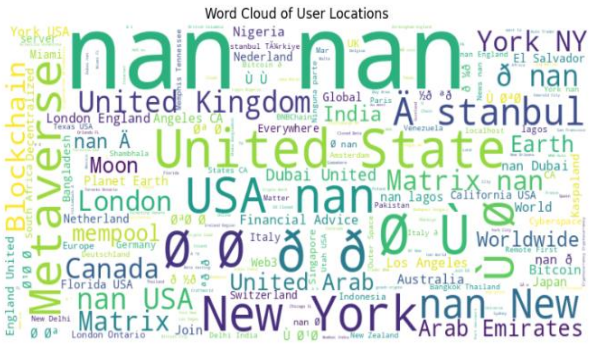
range (IQR) method, where values falling outside 1.5 times the IQR range are considered outliers. The outliers are stored in a separate DataFrame.

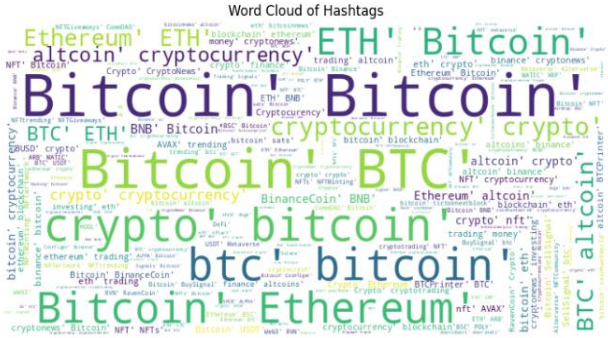
To visualize the outliers, boxplots are created for each numerical column using the boxplot() function from matplotlib.pyplot. Each boxplot represents the distribution of values for a specific column, allowing for the identification of potential outliers.

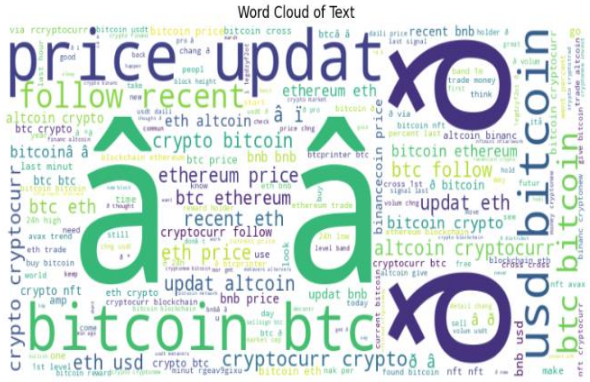
This analysis helps in detecting extreme values or anomalies in the numerical variables, providing insights into data quality and potential data issues that may require further investigation or data cleaning.

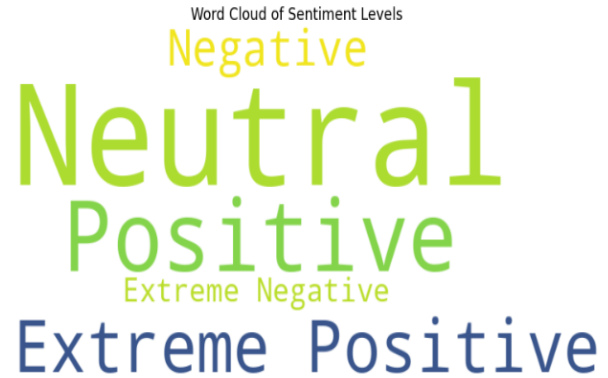


Word Cloud









EDA for cryptocurrency

Bitcoin, Ethereum, XRP, Litecoin, and USD Coin were selected for analysis in the dataset due to several factors:

1. Market Dominance: Bitcoin (BTC) is the first and most well-known cryptocurrency, with the highest market capitalization and liquidity. Ethereum (ETH) is the second-largest cryptocurrency and a popular platform for decentralized applications. XRP is the native cryptocurrency of the Ripple network, which aims to facilitate fast and low-cost international money transfers. Litecoin (LTC) is a peer-to-peer cryptocurrency that offers faster transaction confirmation times compared to Bitcoin. USD Coin (USDC) is a stablecoin pegged to the US dollar, providing stability and utility for cryptocurrency traders.

2. Diverse Use Cases: Each cryptocurrency serves different purposes and has unique features. Bitcoin is primarily used as a store of value and a medium of exchange. Ethereum's blockchain allows for the creation of smart contracts and decentralized applications. XRP focuses on facilitating efficient cross-border transactions. Litecoin offers faster transaction confirmations

and lower fees, making it suitable for daily transactions. USD Coin provides stability by being pegged to a fiat currency, enabling seamless integration with traditional financial systems.

3. Popularity and Adoption: Bitcoin and Ethereum have gained significant mainstream attention and widespread adoption among individuals, institutions, and businesses. XRP has established partnerships with financial institutions for international remittances. Litecoin has a strong community following, and USD Coin has gained traction as a stablecoin used in various crypto trading platforms and DeFi applications.

Brief details about the selected cryptocurrencies:

- Bitcoin (BTC): Launched in 2009, Bitcoin is the first decentralized cryptocurrency. It operates on a peer-to-peer network without a central authority and is widely accepted as a digital currency for various transactions.

- Ethereum (ETH): Introduced in 2015, Ethereum is a blockchain-based platform that enables the creation of decentralized applications and smart contracts. Its native cryptocurrency, Ether, is used for transactions and as a means of compensation within the network.

- XRP: Developed by Ripple Labs, XRP is a digital asset designed for fast, low-cost international money transfers. It aims to facilitate efficient cross-border transactions between financial institutions and has gained attention for its potential to revolutionize the traditional remittance industry.

- Litecoin (LTC): Created in 2011 by Charlie Lee, a former Google engineer, Litecoin is a peer-to-peer cryptocurrency that offers faster transaction confirmations and a different hashing algorithm than Bitcoin. It is often seen as a "silver" counterpart to Bitcoin's "gold" and is used for everyday transactions.

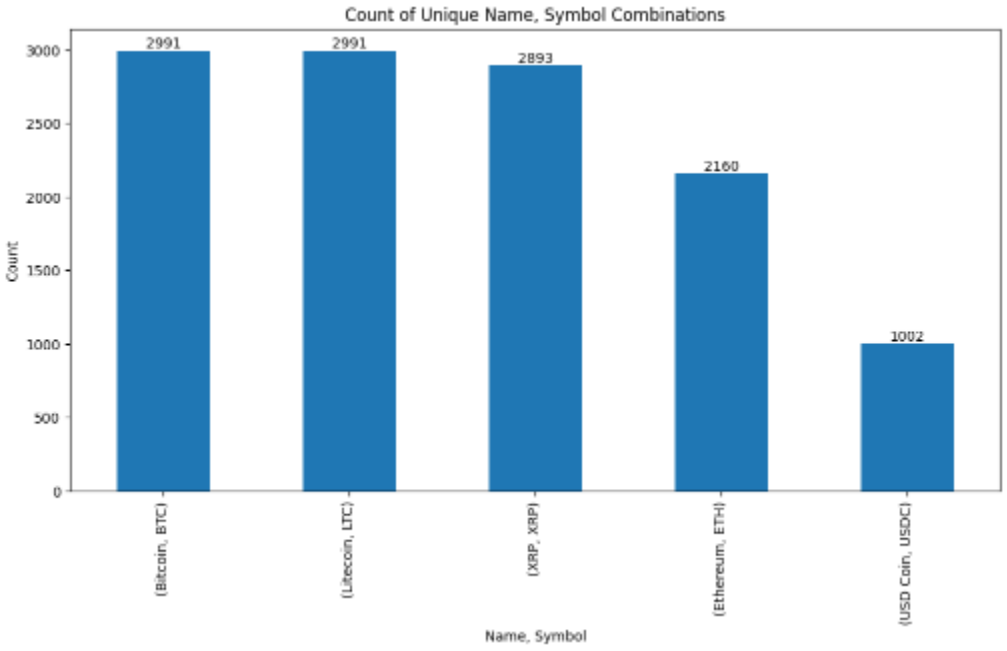
- USD Coin (USDC): Introduced in 2018, USD Coin is a stablecoin pegged to the value of the US dollar. It provides stability and can be easily exchanged for fiat currency, making it suitable for traders and as a medium of exchange in the crypto ecosystem.

These selected cryptocurrencies represent a mix of established and innovative digital assets with distinct characteristics, making them valuable for analysis and understanding the broader cryptocurrency market.

EDA report link : https://github.com/Amarpreet3/CIND-820-CAPSTONE/blob/main/Codes/Crypto_EDA_updated.ipynb

Count of each coin for analysis:

Imports the matplotlib.pyplot module for plotting, calculates the counts of unique combinations of 'Name' and 'Symbol' columns in the DataFrame, and generates a bar chart to visualize the count distribution.



Dataset description

Displays the summary information of the DataFrame using the info() method and prints it.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12037 entries, 0 to 1001
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   SNo        12037 non-null  int64
 1   Name       12037 non-null  object
 2   Symbol     12037 non-null  object
 3   Date       12037 non-null  object
 4   High       12037 non-null  float64
 5   Low        12037 non-null  float64
 6   Open       12037 non-null  float64
 7   Close      12037 non-null  float64
 8   Volume     12037 non-null  float64
 9   Marketcap  12037 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 1.0+ MB
None
```

Data Dictionary

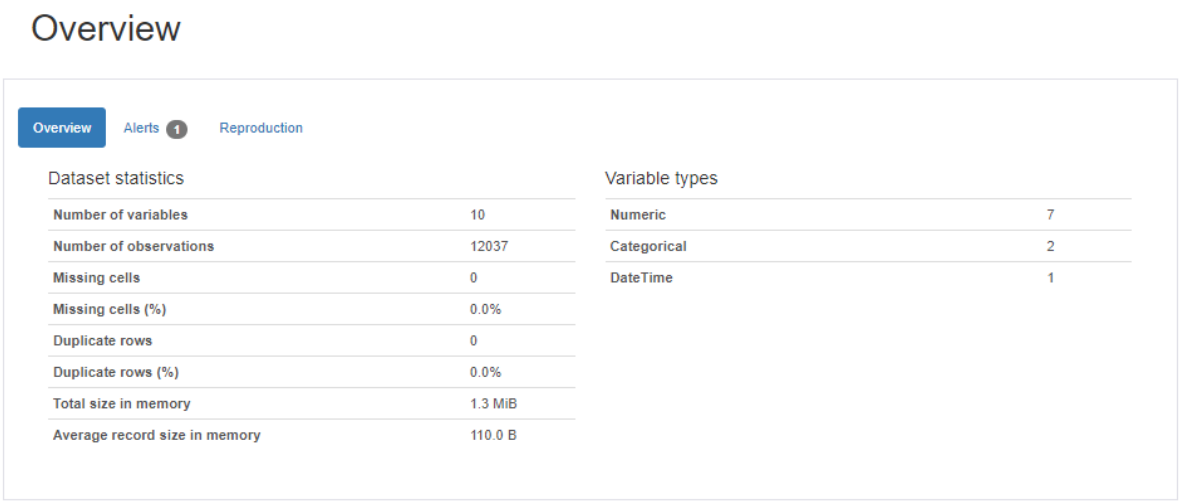
The data dictionary provides a concise description of the dataset, including the names and definitions of each variable or column in the dataset.

Column Name	Description	Variable Type	Data Type

SNo	Serial number	Numerical (Nominal)	int64
Name	Name of the crypto currency	Categorical (Nominal)	object
Symbol	The crypto currency symbol	Categorical (Nominal)	object
Date	The date of the recorded data point	Date (Dataframe)	object
High	The highest price reached by the crypto currency on the given date	Numerical (Continuous)	float64
Low	The lowest price reached by the crypto currency on the given date	Numerical (Continuous)	float64
Open	The opening price of the crypto currency on the given date	Numerical (Continuous)	float64
Close	The closing price of the crypto currency on the given date	Numerical (Continuous)	float64
Volume	The trading volume of the crypto currency on the given date	Numerical (Continuous)	float64
Marketcap	The market capitalization of the crypto currency on the given date	Numerical (Continuous)	float64

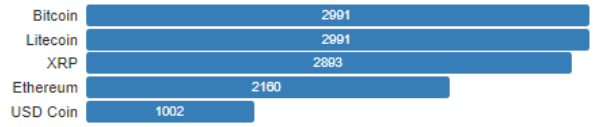
Pandas profiling Reports

Pandas Profiling is a powerful library that automatically generates a detailed report containing various statistical and descriptive information about a DataFrame, including data types, missing values, correlations, distributions, and more.



Name
Categorical

Distinct	5
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	188.1 KiB



Sample

First rows | Last rows

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.603769e+09
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.542813e+09
2	3	Bitcoin	BTC	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.0	1.298955e+09
3	4	Bitcoin	BTC	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.0	1.168517e+09
4	5	Bitcoin	BTC	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.0	1.085995e+09
5	6	Bitcoin	BTC	2013-05-04 23:59:59	115.000000	92.500000	98.099998	112.500000	0.0	1.250317e+09
6	7	Bitcoin	BTC	2013-05-05 23:59:59	118.800003	107.142998	112.900002	115.910004	0.0	1.288693e+09
7	8	Bitcoin	BTC	2013-05-06 23:59:59	124.663002	106.639999	115.980003	112.300003	0.0	1.249023e+09
8	9	Bitcoin	BTC	2013-05-07 23:59:59	113.444000	97.699997	112.250000	111.500000	0.0	1.240594e+09
9	10	Bitcoin	BTC	2013-05-08 23:59:59	115.779999	109.599998	109.599998	113.566002	0.0	1.264049e+09

Missing Values

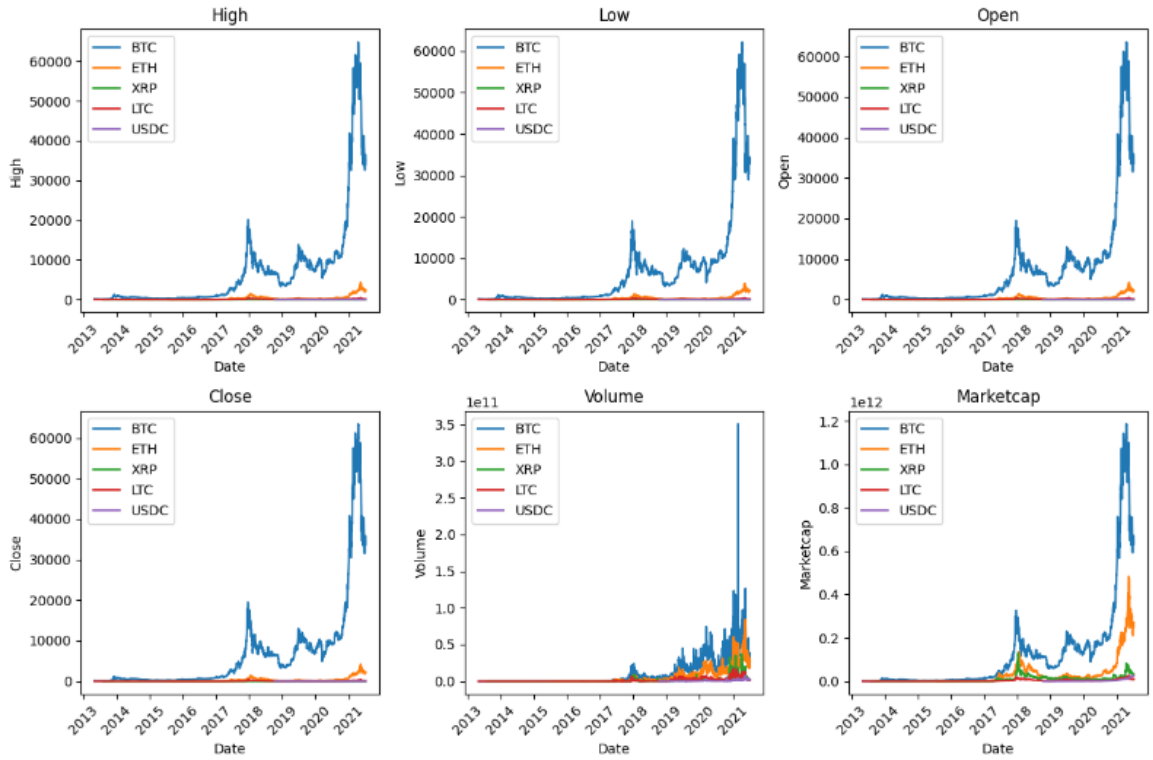
Missing values refer to the absence of data in a DataFrame or Series. They can impact data analysis and modeling, and it is important to handle them appropriately by either removing or imputing the missing values.

```
SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

There are no missing values in the DataFrame or Series, it indicates that all the data points are present for the variables under consideration, ensuring completeness and accuracy of the dataset.

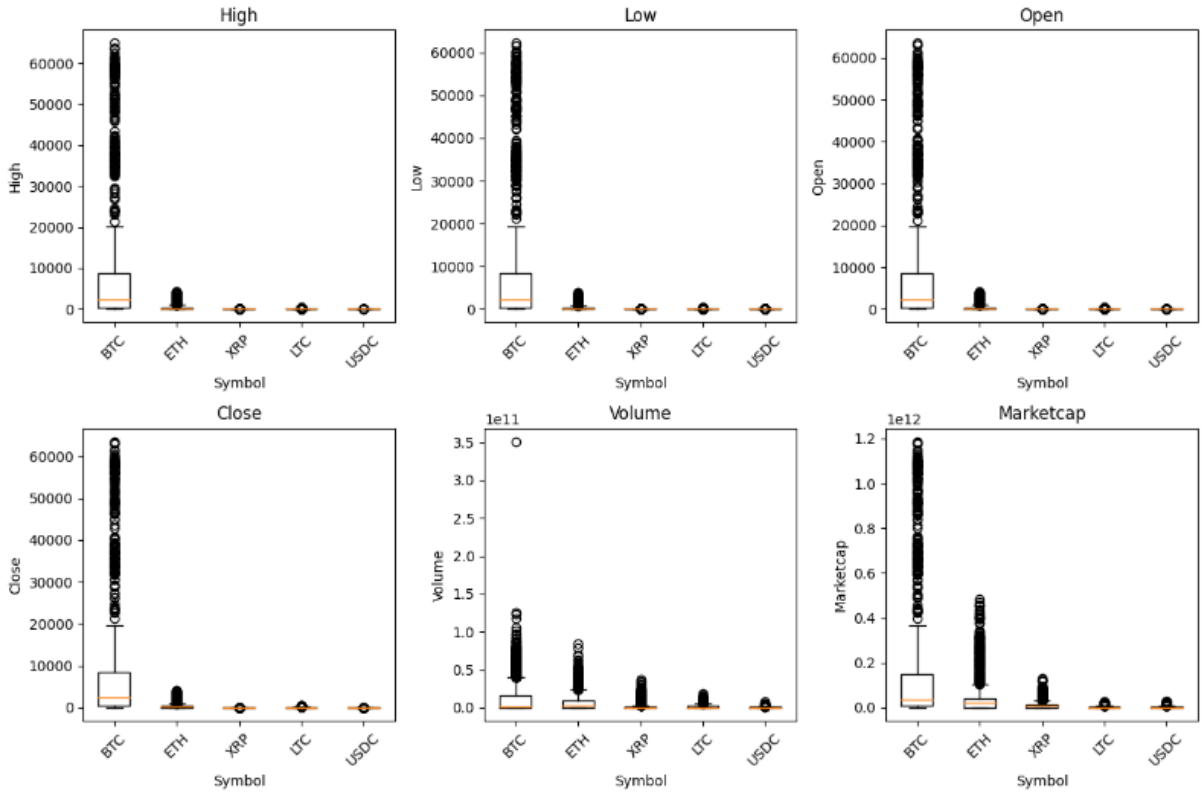
Time-series plot for each coin with each numeric column

Visualizes the data for each column in a grid layout, allowing for easy comparison across different symbols. It helps to identify any patterns or trends in the data over time. The plot also highlights the relative differences between symbols for each column, providing insights into their respective performance or behavior.



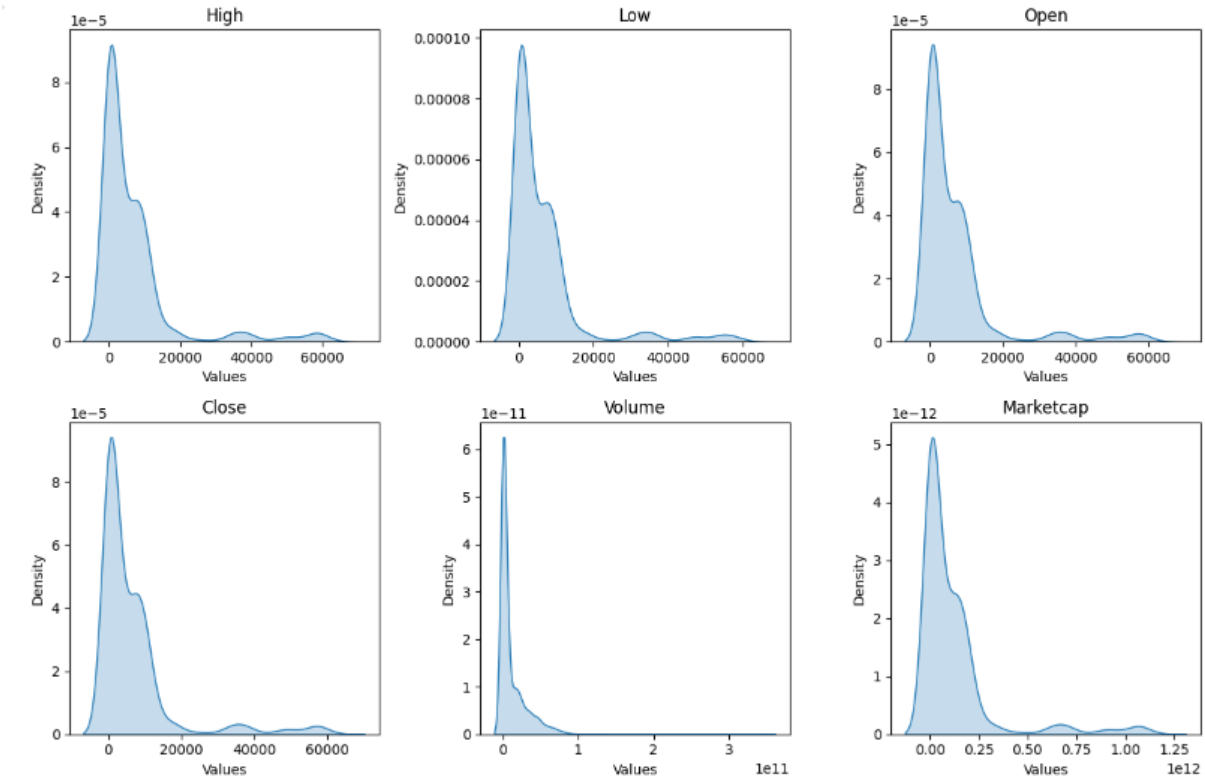
Box- plot for each coin for each numeric attribute

The boxplot analysis provides a visual representation of the distribution of data across different symbols for each attribute. It allows for easy comparison of the central tendency, variability, and potential outliers within each attribute for the different symbols.



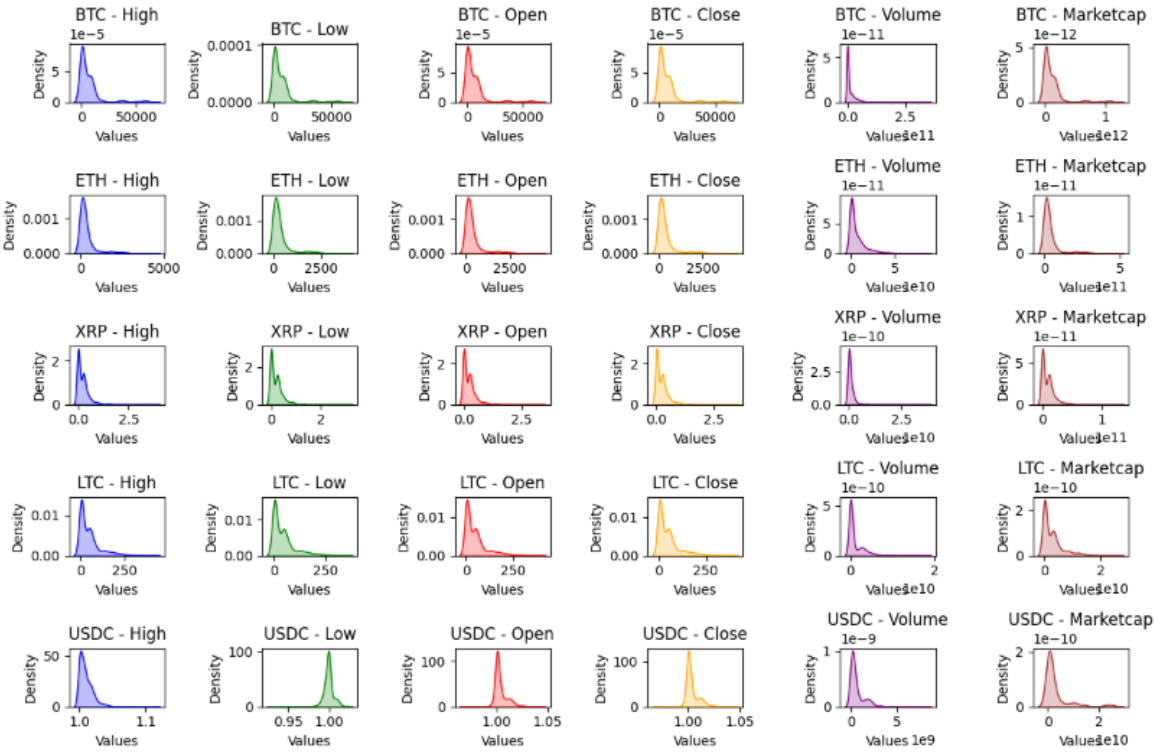
Skewness

The code uses Seaborn and Matplotlib to visualize the density plot of selected coin's data for each attribute. It provides insights into the distribution and shape of the data, allowing for analysis of the concentration and spread of values for the chosen coin.



Skewness of each coin for each numeric attribute

Each coin's data is represented by a separate row in the plot grid, and each attribute is visualized in a separate column. The density plots show the distribution and concentration of values for each coin and attribute combination, allowing for comparison and analysis across different coins.



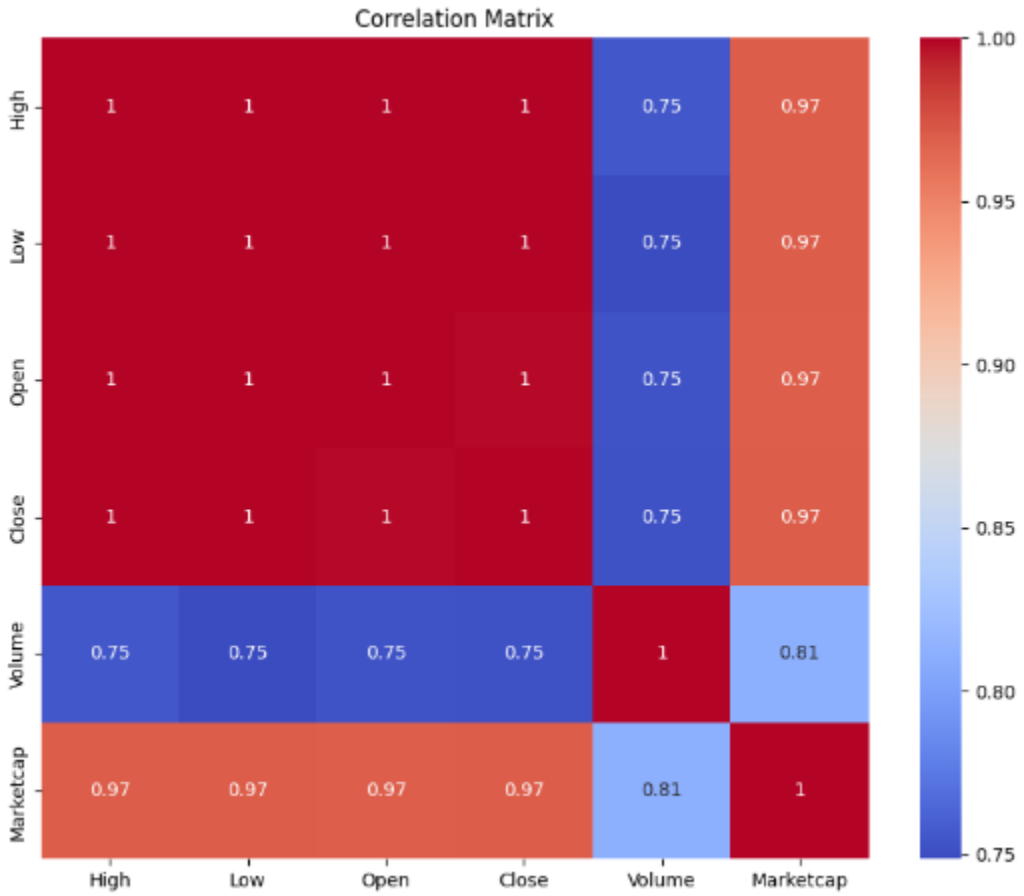
Identify the column with the highest skewness and retrieves its corresponding skewness value:

The column with the highest skewness is 'Low' with a skewness value of 6.27.

Correlation Matrix

Correlation Matrix to find highly correlated attributes and reduce the dimensions of dataset for further analysis.

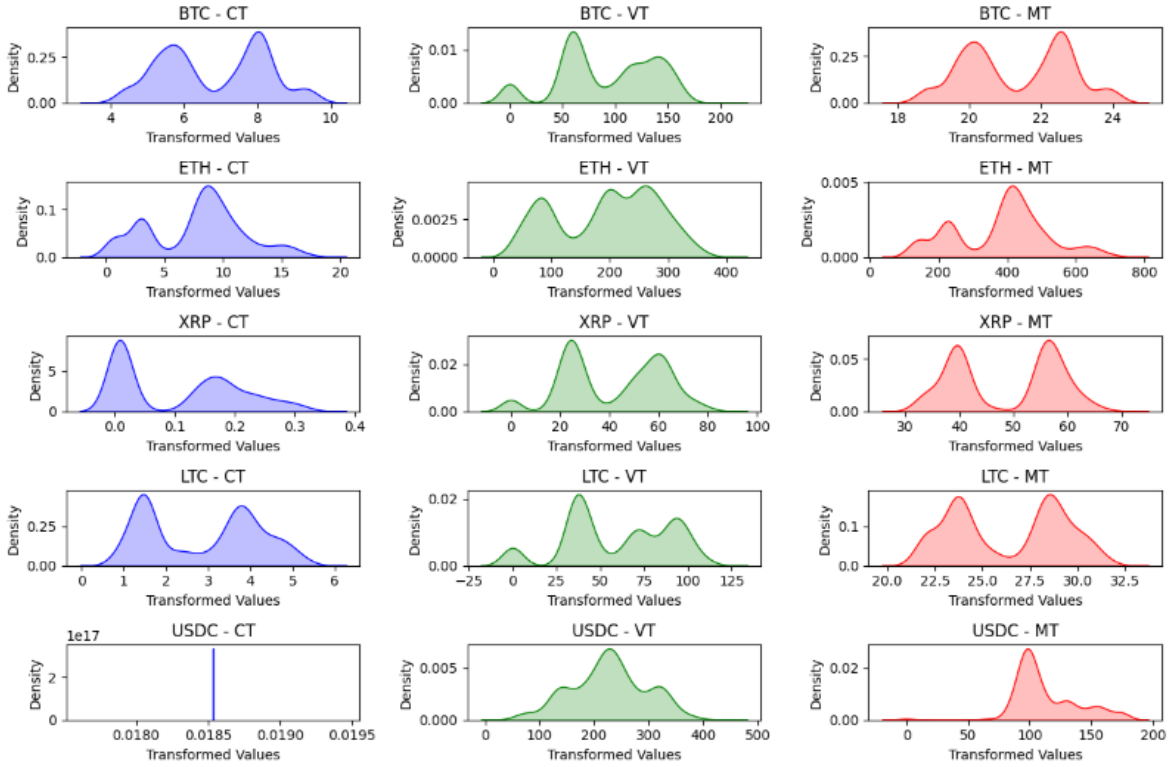
Plot the correlation matrix as a heatmap using seaborn's heatmap function, annotating the heatmap with the correlation values and customizing the colormap. The resulting plot is titled 'Correlation Matrix'.



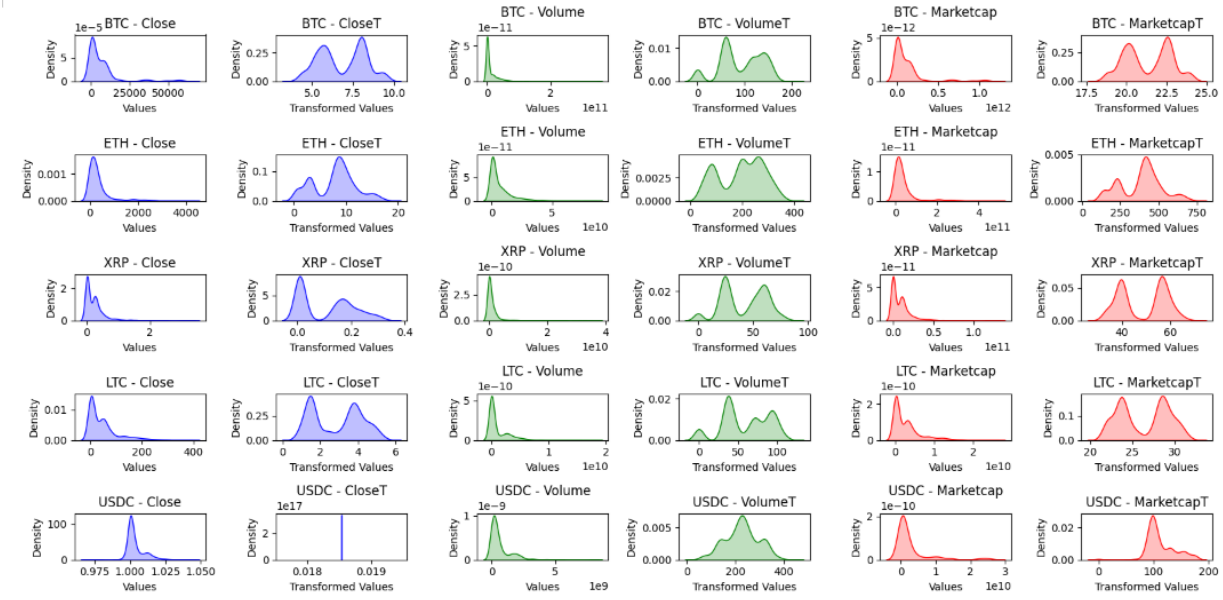
The analysis shows that the columns 'Close', 'Open', 'High', and 'Low' are highly correlated based on the correlation matrix heatmap. This indicates a strong positive relationship between these variables, suggesting that changes in one variable are closely associated with changes in the others.

The transformation methods used in the provided code include logarithmic transformation, square root transformation, Box-Cox transformation, reciprocal transformation, and Yeo-Johnson transformation. These transformations are applied to the original data for each coin and column, and the resulting density plots illustrate the impact of each transformation on reducing skewness. Based on the observations from the density plots, the Box-Cox Transformation appears to be effective in reducing skewness by achieving more balanced and symmetric distributions. Comparing the density plots can help in selecting the most suitable transformation for skewness reduction.

CT- Close Transformed, VT- Volume Transformed, MT- Marketcap Transformed

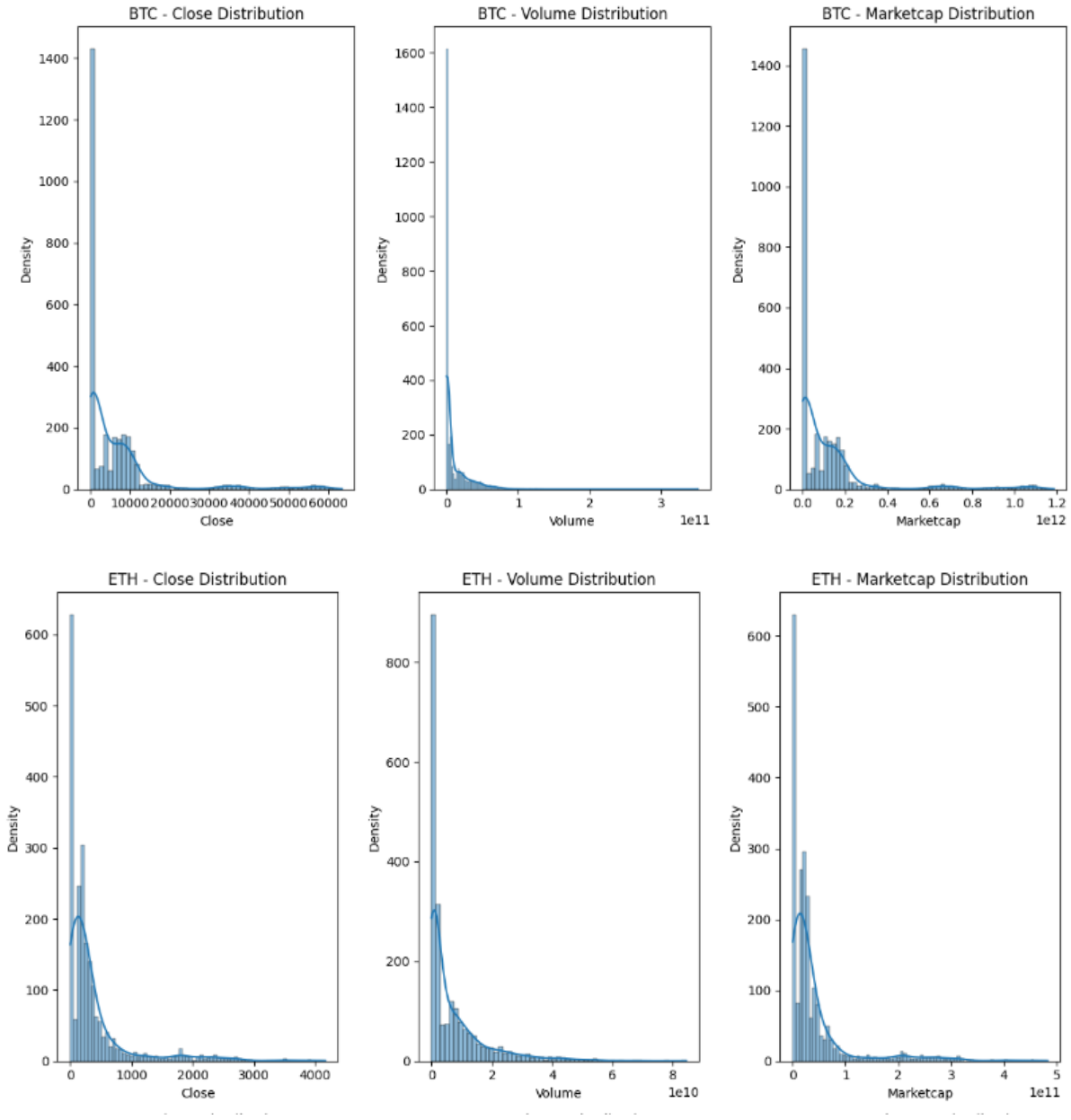


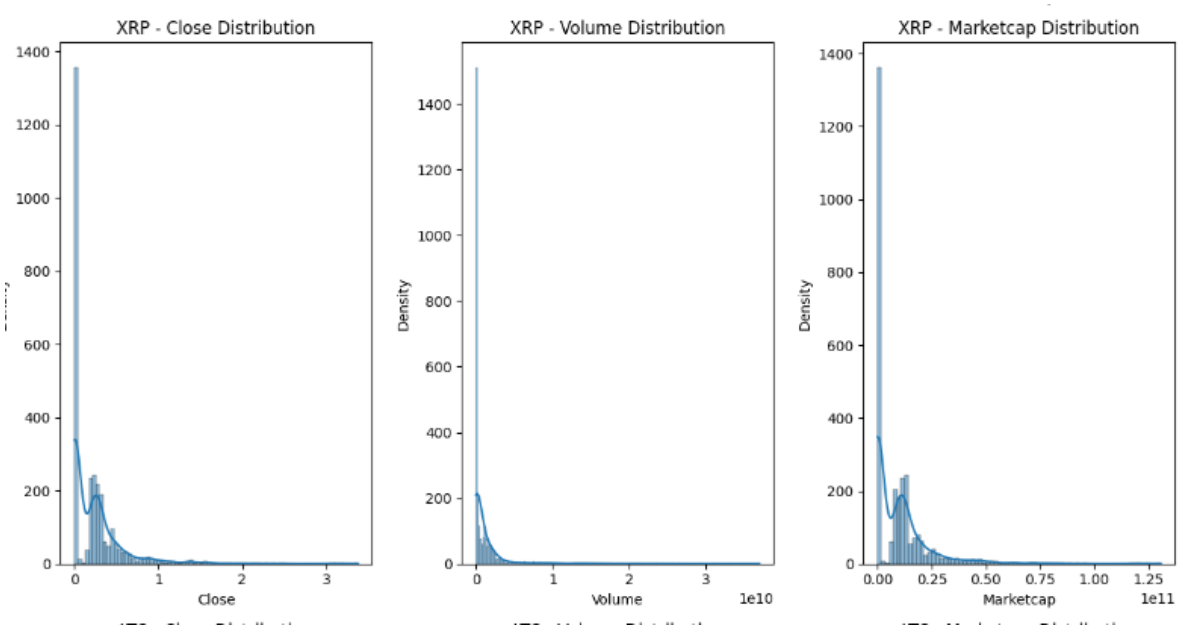
Plotting the original and transformed sidewise to compare.

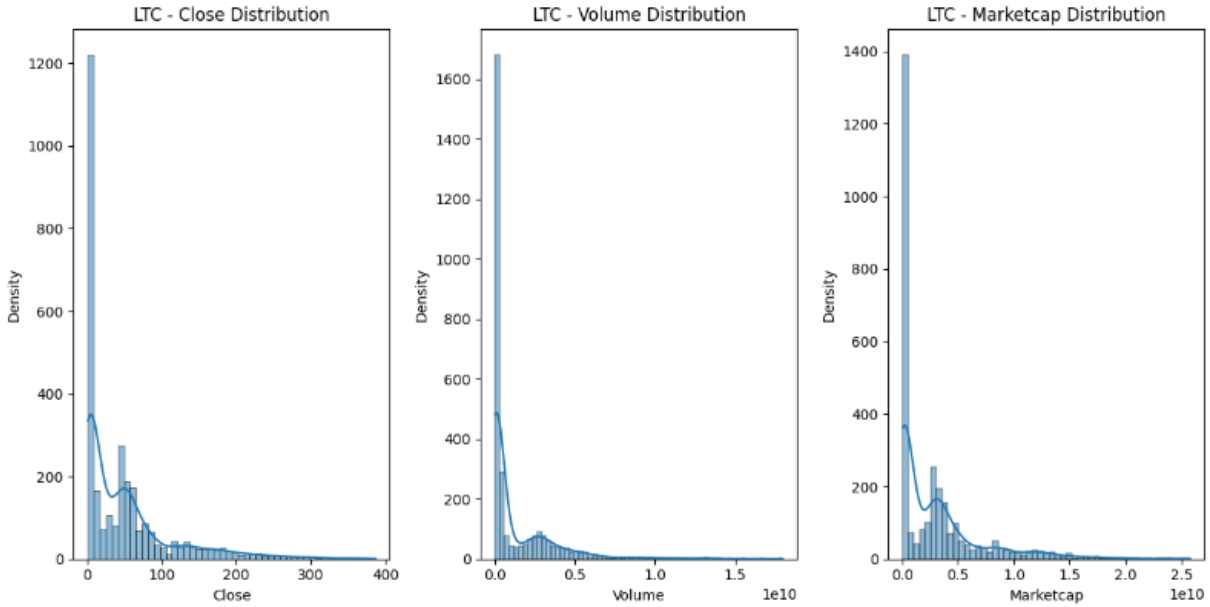


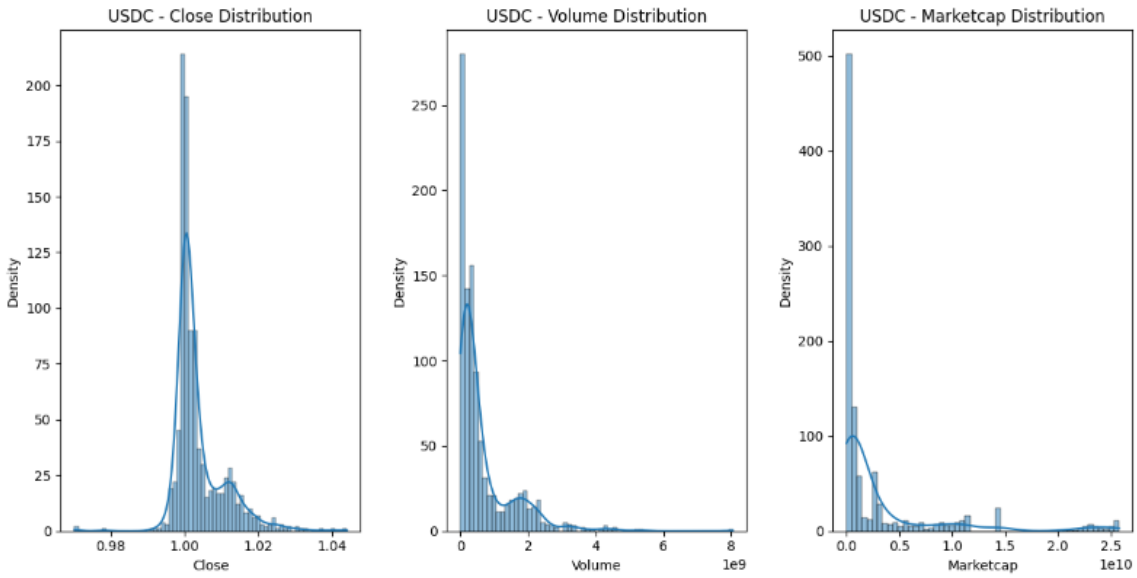
Univariate Analysis

The univariate analysis of the 'Close' attribute through histograms with KDE provides a focused examination of a single variable, allowing us to explore the distribution and characteristics of individual coins' closing prices. It helps in understanding the central tendency, spread, and shape of the data, providing valuable insights into the behavior of each coin's 'Close' attribute.



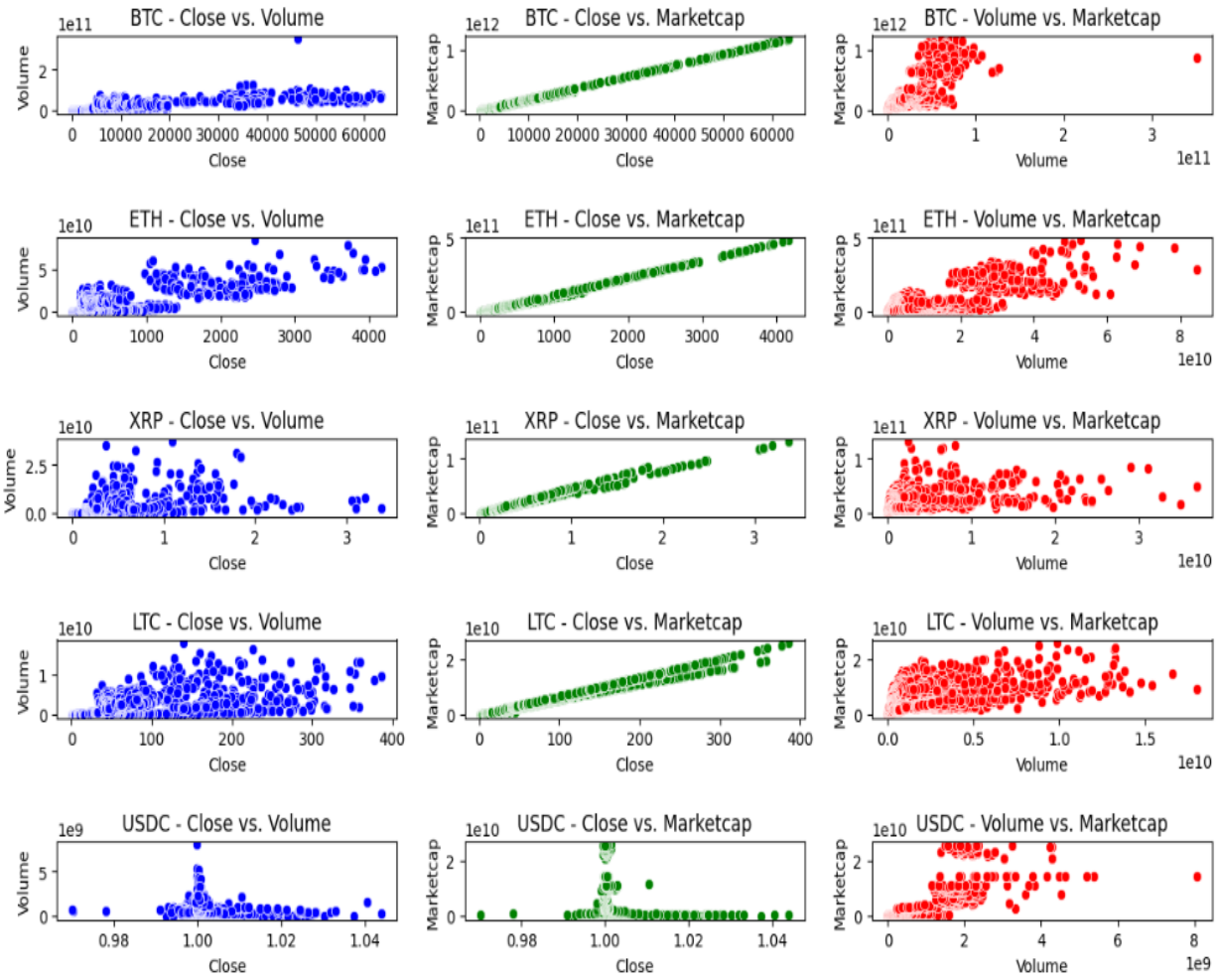

XRP - Close Distribution
Close
XRP - Volume Distribution
Density
Volume
1e10
XRP - Marketcap Distribution
Density
Marketcap
1e11


LTC - Close Distribution
Density
Close
LTC - Volume Distribution
Density
Volume
1e10
LTC - Marketcap Distribution
Density
Marketcap
1e10



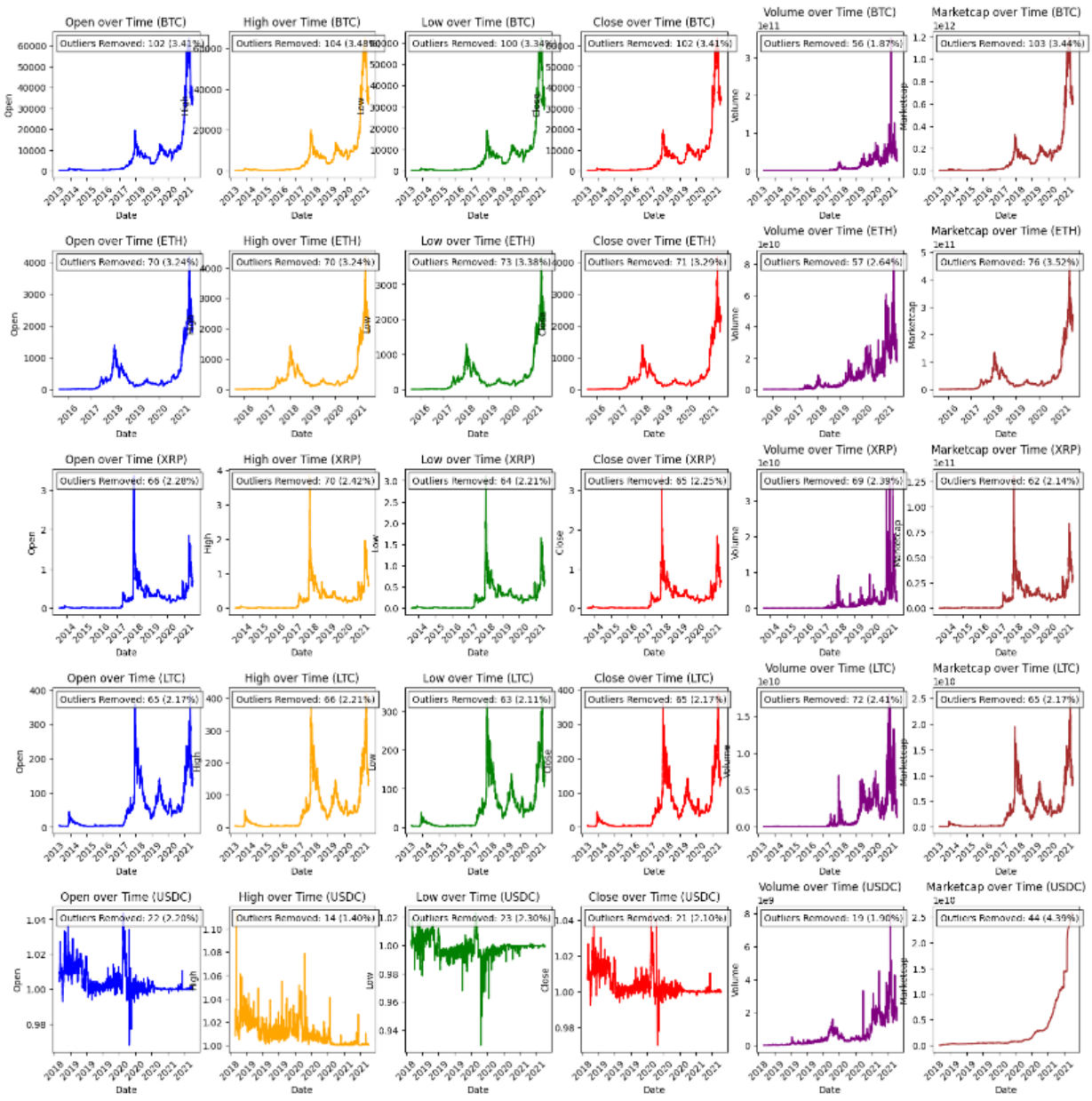
Bivariate analysis

The bivariate analysis using scatterplots allows us to explore the relationships between different pairs of variables, such as 'Close' and 'Volume', 'Close' and 'Marketcap', and 'Volume' and 'Marketcap' for each coin. These plots provide visual insights into the potential correlations or patterns between these attributes, enabling a better understanding of how they interact and influence each other.



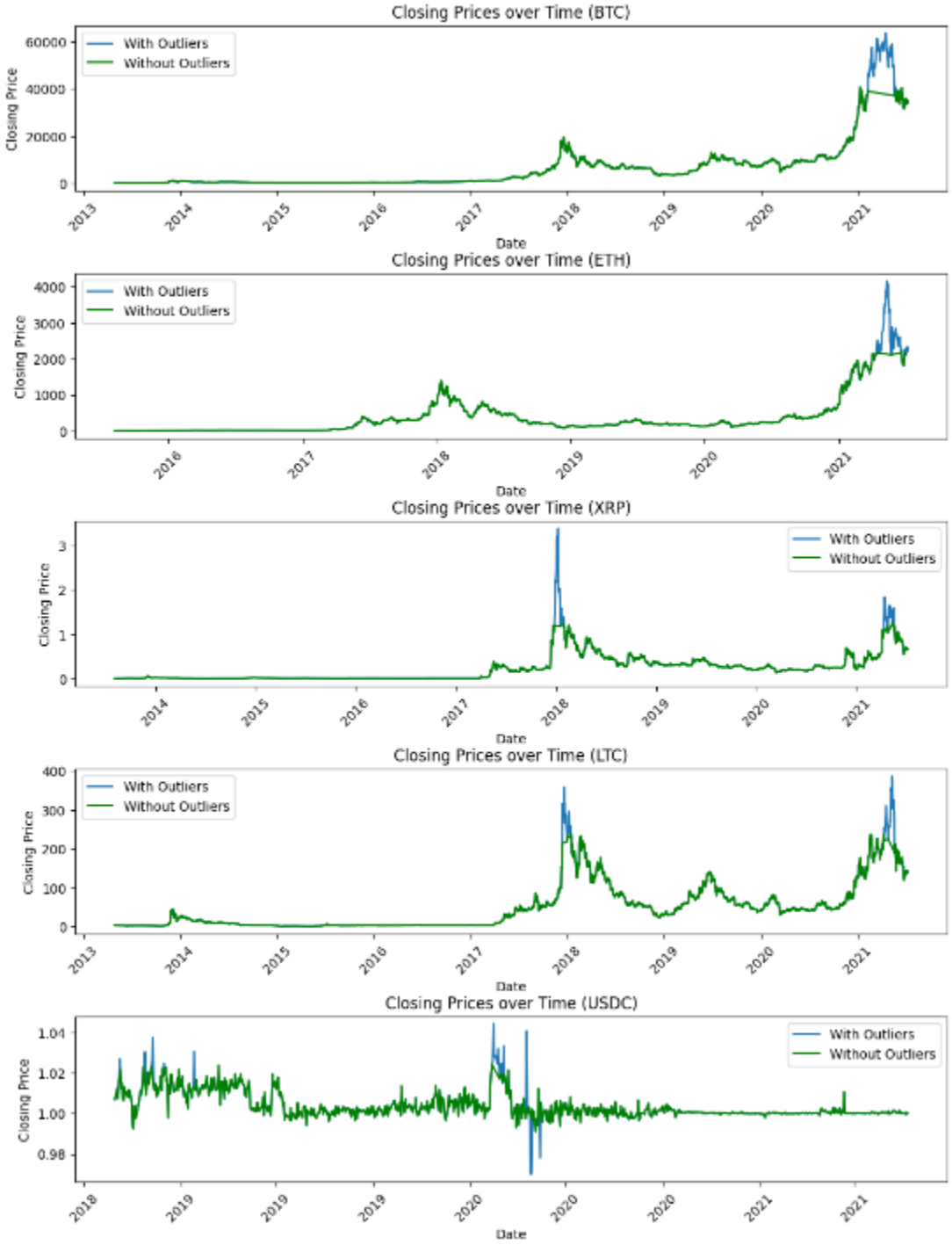
Outliers

The code utilizes z-scores to detect outliers in the closing price data for the selected cryptocurrencies. Outliers are defined as data points that deviate significantly from the mean, with a z-score greater than 3 or less than -3. By removing these outliers, the code helps in visualizing the closing price trends more accurately and gaining insights into the overall distribution of closing prices for the cryptocurrencies.

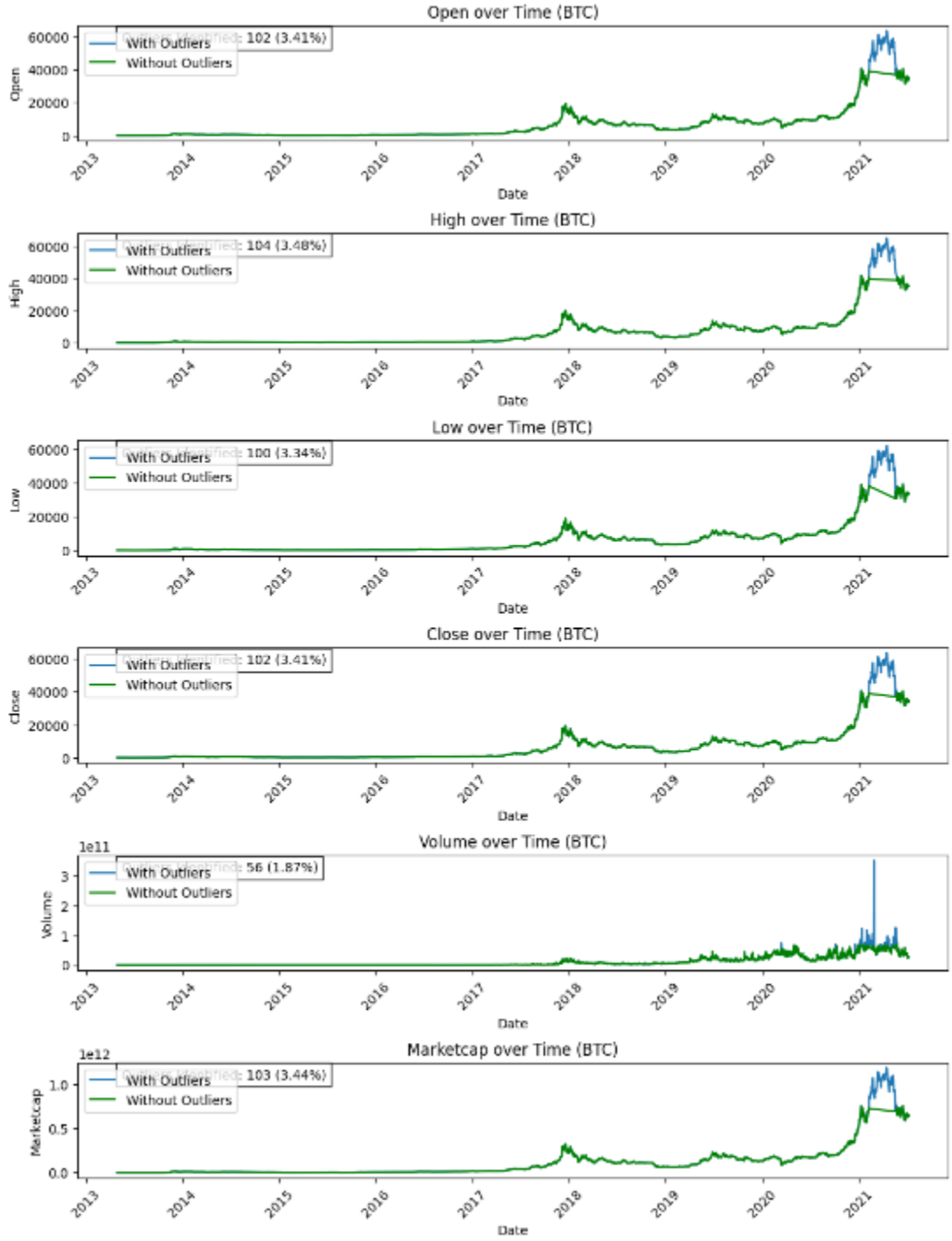


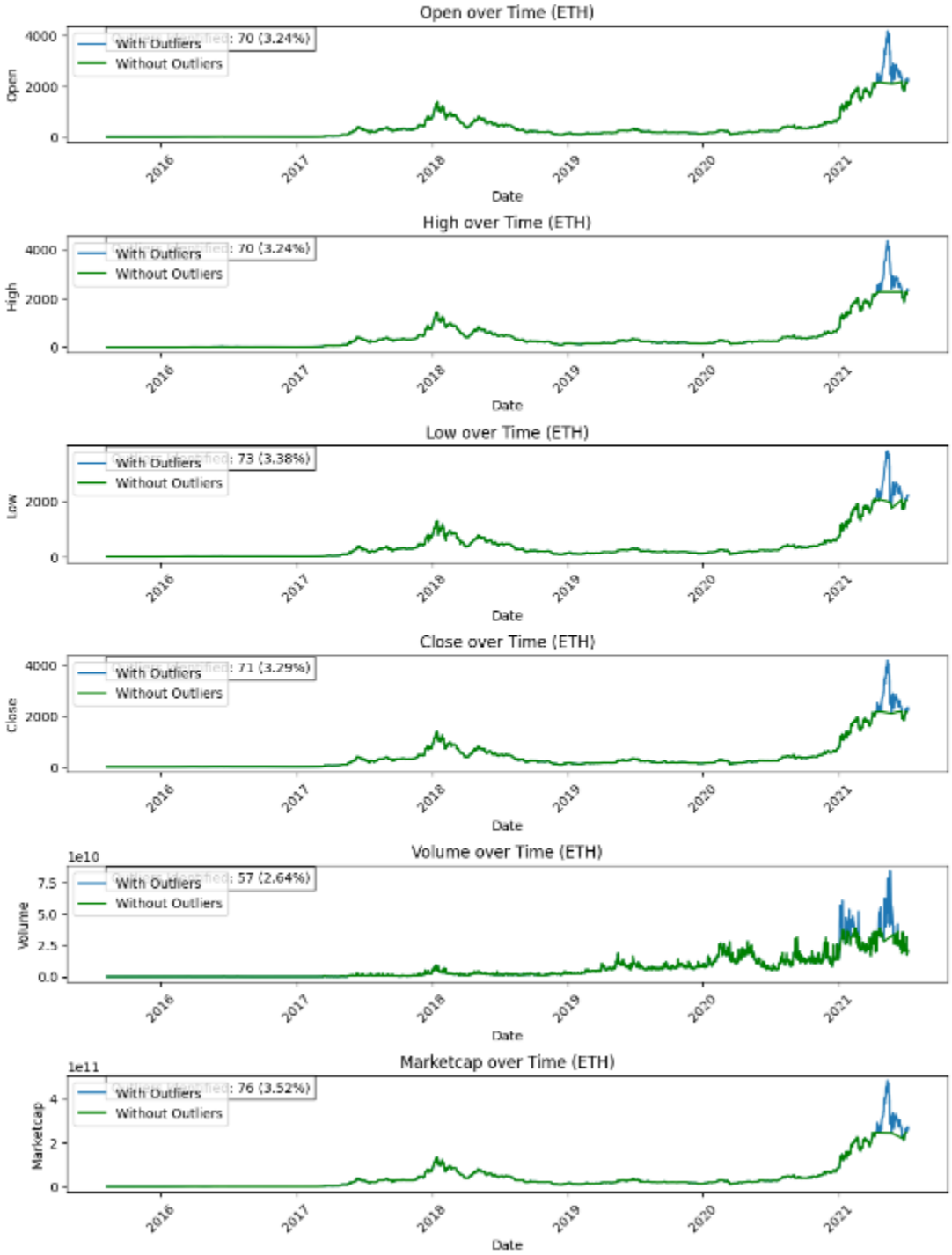
Outliers' analysis for "Close" attribute.

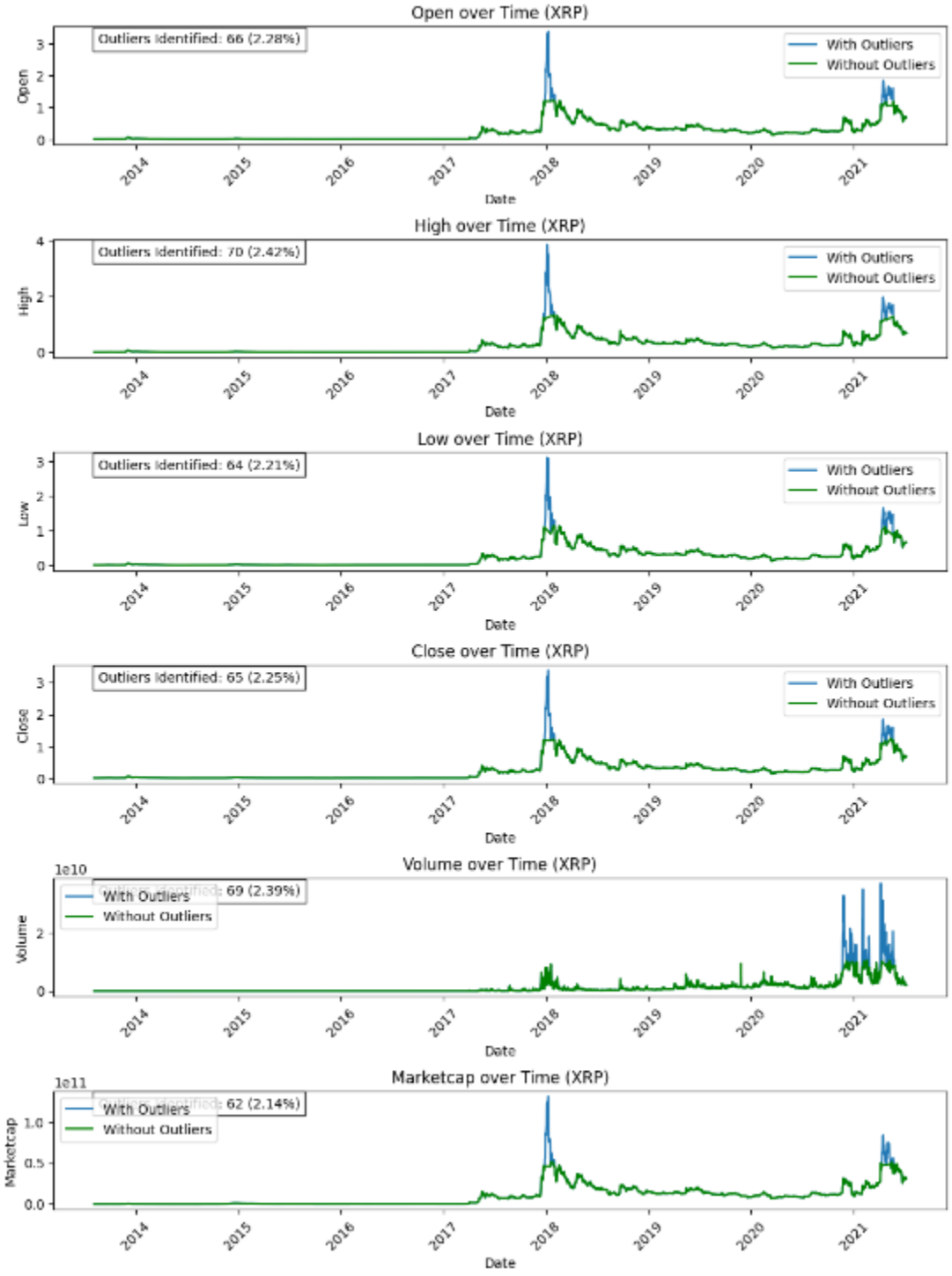
Identifies outliers in the closing price data for the selected cryptocurrencies using z-scores, removes the outliers, and plots the time series of closing prices for each cryptocurrency with and without outliers.

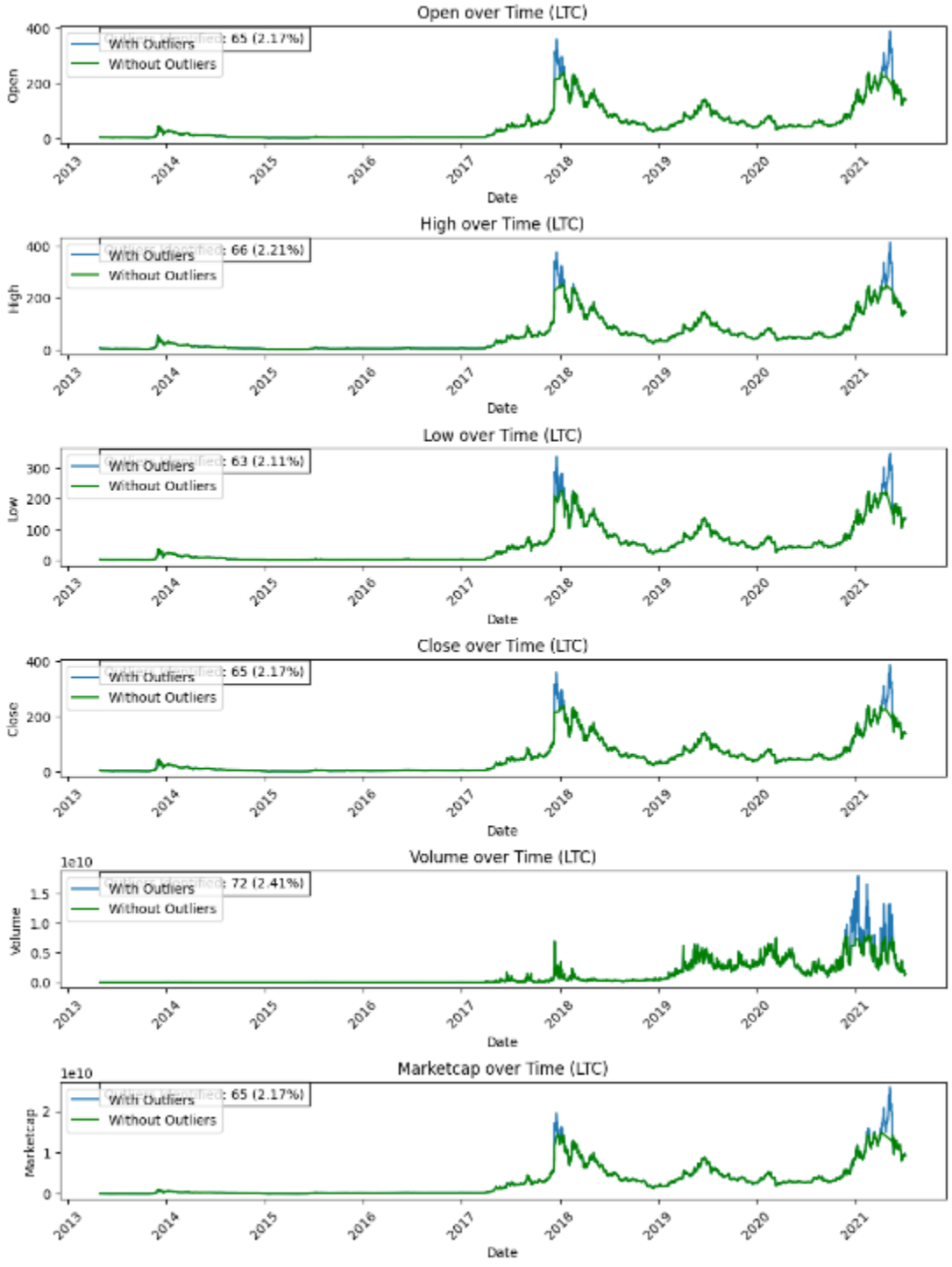


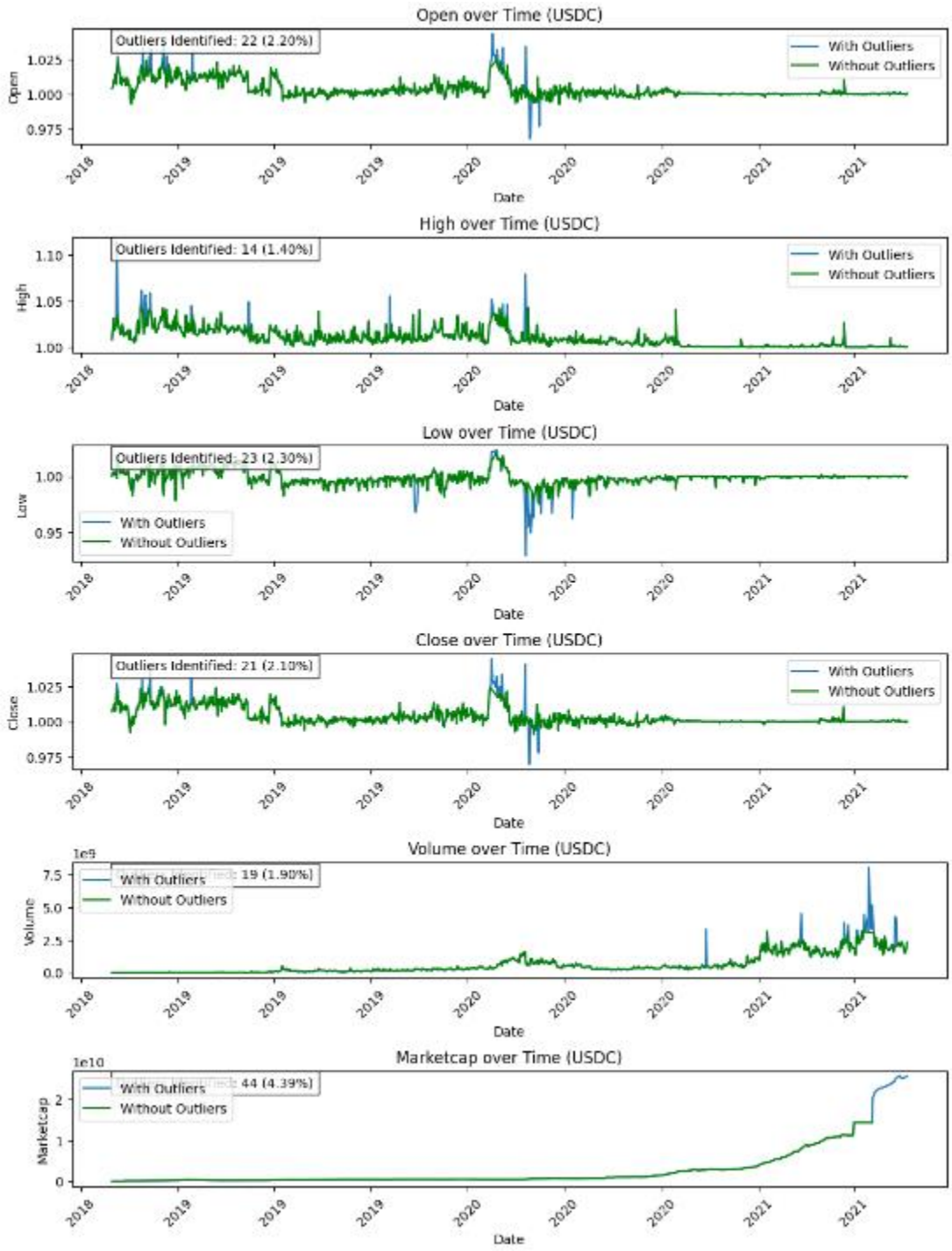
Analysis of each coin with various attributes.


Open over Time (BTC)
Outliers Identified: 102 (3.41%)
With Outliers
Without Outliers
Open
Date
High over Time (BTC)
Outliers Identified: 104 (3.48%)
With Outliers
Without Outliers
High
Date
Low over Time (BTC)
Outliers Identified: 100 (3.34%)
With Outliers
Without Outliers
Low
Date
Close over Time (BTC)
Outliers Identified: 102 (3.41%)
With Outliers
Without Outliers
Close
Date
Volume over Time (BTC)
1e11
Outliers Identified: 56 (1.87%)
With Outliers
Without Outliers
Volume
Date
Marketcap over Time (BTC)
1e12
Outliers Identified: 103 (3.44%)
With Outliers
Without Outliers
Marketcap
Date


Open over Time (ETH)
With Outliers
Without Outliers
Open
Date
High over Time (ETH)
High
Low over Time (ETH)
Low
Close over Time (ETH)
Close
Volume over Time (ETH)
1e10
Volume
Marketcap over Time (ETH)
1e11
Marketcap
2016
2017
2018
2019
2020
2021


Open over Time (XRP)
Outliers Identified: 66 (2.28%)
With Outliers
Without Outliers
Open
Date
High over Time (XRP)
Outliers Identified: 70 (2.42%)
With Outliers
Without Outliers
High
Date
Low over Time (XRP)
Outliers Identified: 64 (2.21%)
With Outliers
Without Outliers
Low
Date
Close over Time (XRP)
Outliers Identified: 65 (2.25%)
With Outliers
Without Outliers
Close
Date
Volume over Time (XRP)
1e10
69 (2.39%)
With Outliers
Without Outliers
Volume
Date
Marketcap over Time (XRP)
1e11
62 (2.14%)
With Outliers
Without Outliers
Marketcap
Date


Open over Time (LTC)
With Outliers
Without Outliers
Open
Date
High over Time (LTC)
With Outliers
Without Outliers
High
Date
Low over Time (LTC)
With Outliers
Without Outliers
Low
Date
Close over Time (LTC)
With Outliers
Without Outliers
Close
Date
Volume over Time (LTC)
1e10
With Outliers
Without Outliers
Volume
Date
Marketcap over Time (LTC)
1e10
With Outliers
Without Outliers
Marketcap
Date



Based on the graphs generated for each cryptocurrency and each price column (Open, High, Low, Close, Volume, Marketcap), here are some insights that can be derived:

1. **Price Trends:** The plots reveal the overall price trends over time for each cryptocurrency. It allows us to observe whether the prices have been generally increasing, decreasing, or fluctuating.

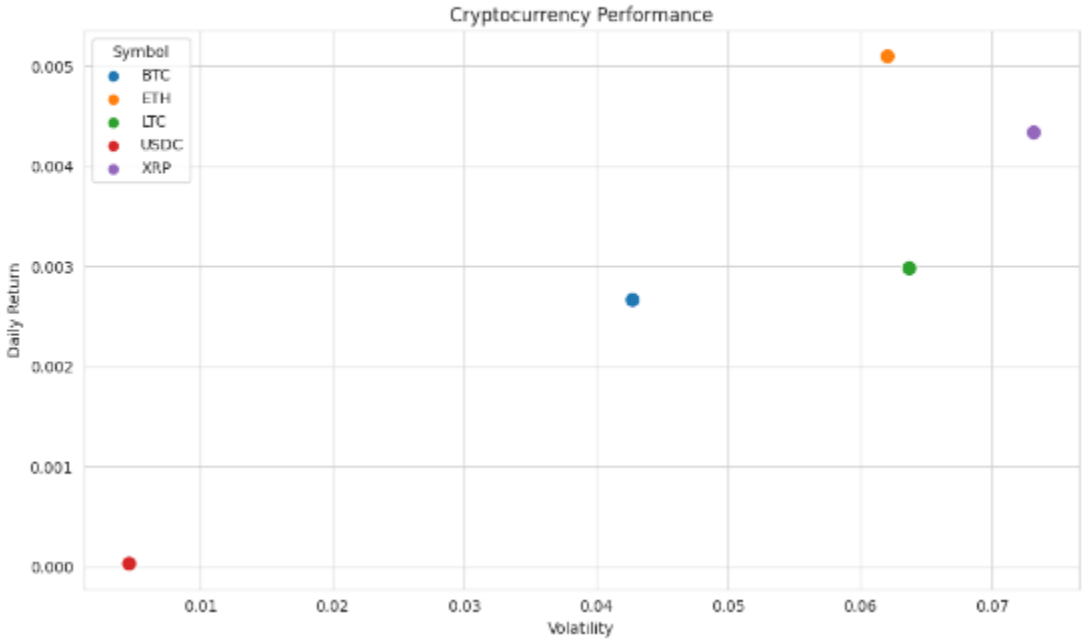
2. **Outlier Identification:** The graphs highlight the outliers in the price data. These outliers represent extreme price movements that deviate significantly from the overall trend. By visually identifying these outliers, we can gain insights into unusual price behavior or market events that might have impacted the cryptocurrencies.
3. **Effect of Outlier Removal:** Comparing the plots with outliers and without outliers provides insights into the impact of outlier removal on the price series. The plots without outliers show smoother price trajectories, which can help in understanding the underlying price trends without the influence of extreme values.
4. **Comparative Analysis:** The graphs allow for the comparison of different cryptocurrencies within the same price column. By examining the price movements of multiple cryptocurrencies, we can analyze their relative performance and identify any divergences or similarities in price behavior.
5. **Volume Analysis:** The plots for the Volume column provide insights into the trading volume of each cryptocurrency. Higher trading volumes may indicate increased market activity and liquidity, while lower volumes may suggest lower interest or trading activity.
6. **Market Capitalization:** The Marketcap plots illustrate the market capitalization of each cryptocurrency over time. This metric reflects the total value of a cryptocurrency and can be used to gauge its overall market significance and investor sentiment.

These insights help in understanding the price dynamics, market behavior, and relative performance of different cryptocurrencies. They can guide further analysis, investment decisions, and the development of trading strategies within the cryptocurrency market.

Daily return and volatility for each currency

The daily return represents the percentage change in the price of a cryptocurrency from one day to the next. It provides insights into the profitability of trading or investing in that particular cryptocurrency on a daily basis. Volatility, on the other hand, measures the fluctuation or dispersion in the daily returns of a cryptocurrency. Higher volatility indicates greater price variability, which can present both opportunities and risks for traders and investors. Analyzing daily return and volatility can help assess the potential profitability and risk associated with trading or holding a specific cryptocurrency.

Symbol	Daily Return	Volatility
BTC	0.002669	0.042758
ETH	0.005102	0.062099
LTC	0.002985	0.063746
USDC	0.000035	0.004587
XRP	0.004341	0.073183

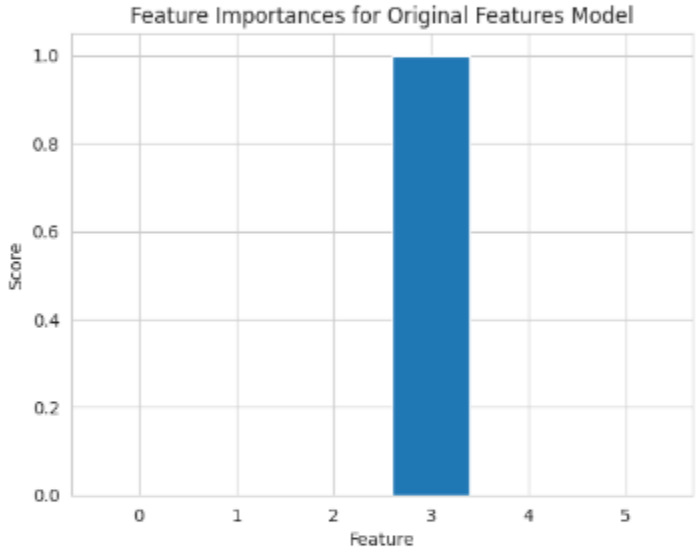


Principal Component Analysis (PCA)

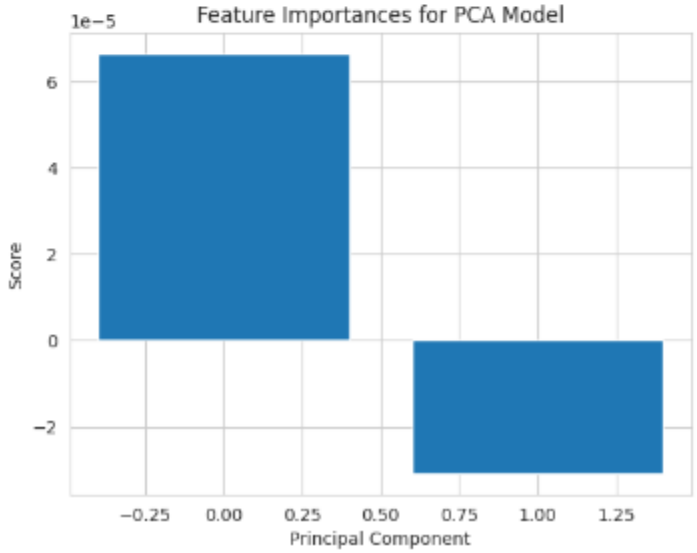
Principal Component Analysis (PCA) can be a useful technique for various projects, including data analysis and visualization. PCA is commonly used for dimensionality reduction, feature extraction, and identifying patterns and relationships in high-dimensional data.

If you have a dataset with multiple variables/features and want to reduce its dimensionality while retaining most of the information, PCA can be a suitable approach. By transforming the original variables into a new set of orthogonal variables called principal components, PCA helps to capture the most significant variations in the data.

```
Original features model metrics:
Mean Squared Error: 8.384992771871535e-19
Mean Absolute Error: 4.986139774615344e-10
Mean Absolute Percentage Error: 3.417094329487432 5e-07
R-squared: 1.0
Adjusted R-squared: 1.0

PCA model metrics:
Mean Squared Error: 1.8274891728603146e-18
Mean Absolute Error: 8.272118443723082e-10
Mean Absolute Percentage Error: 9.767607722184745e-07
R-squared: 1.0
Adjusted R-squared: 1.0
Feature importances for the original features model:
Feature: 0, Score: 0.00000
Feature: 1, Score: 0.00000
Feature: 2, Score: -0.00000
Feature: 3, Score: 1.00000
Feature: 4, Score: 0.00000
Feature: 5, Score: -0.00000
```



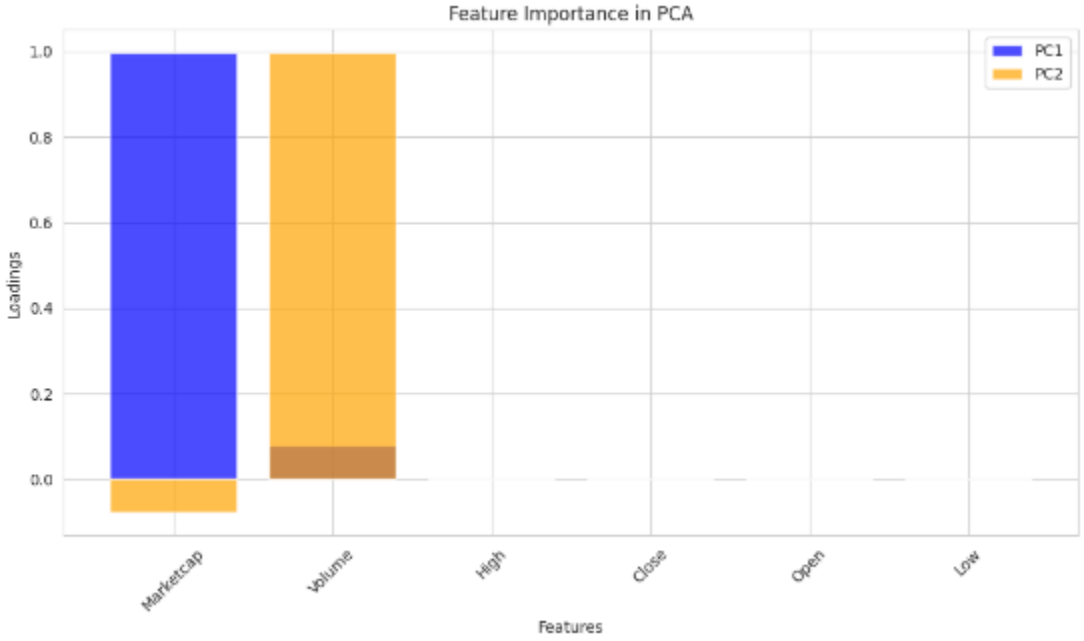

```
Feature importances for the PCA model:
Principal Component: 1, Score: 0.00007
Principal Component: 2, Score: -0.00003
```



The analysis uses Principal Component Analysis (PCA) to explore the relationship between various features of cryptocurrency data. The scatter plot visualizes the clustering of cryptocurrencies based on their price movements in the PC1-PC2 space. The explained variance ratio plot shows the proportion of variance explained by each principal component, indicating the significance of each component. Additionally, the contribution of original features to PC1 and PC2 is depicted, providing insights into which features have the most influence on the principal components.

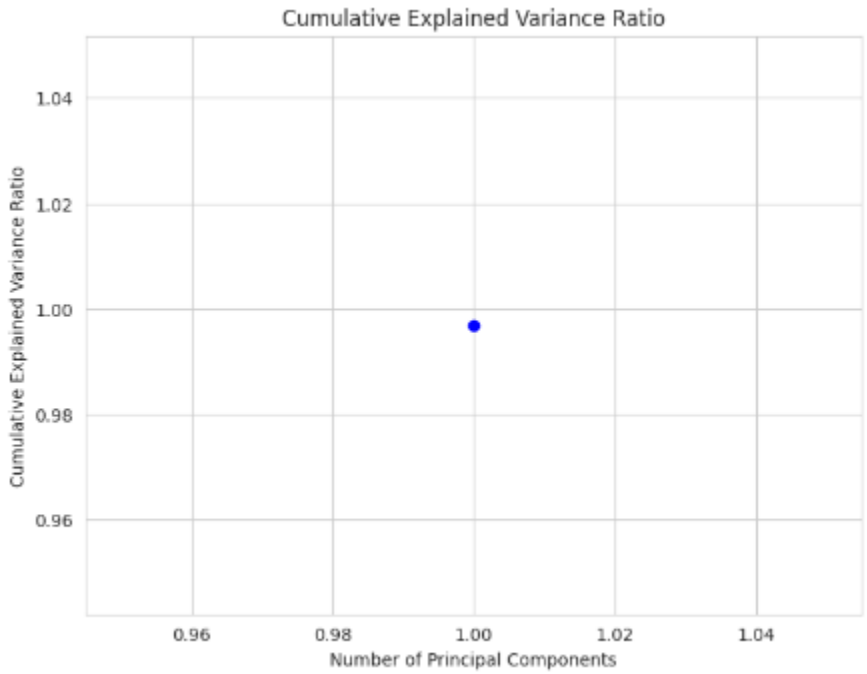
PCA offers various analysis opportunities beyond what we've covered so far. Here are a few additional analysis techniques you can explore with PCA:

1. **Feature Importance:** In PCA, you can examine the loadings or weights of each original feature on the principal components. The higher the loading, the more influential the feature is in determining that principal component. By analyzing these loadings, you can identify the most important features contributing to each principal component.



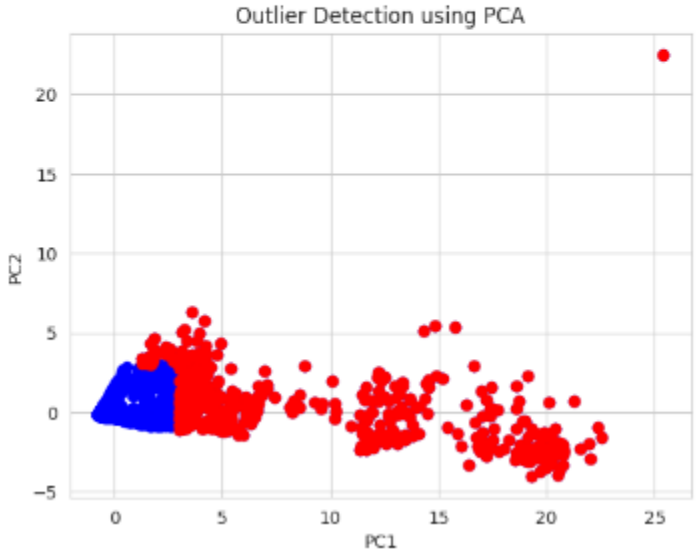
2. **Dimension Reduction:** PCA can help reduce the dimensionality of your dataset. You can determine the number of principal components that capture a significant portion of the variance (e.g., 90% or 95%). By selecting a subset of the most informative principal components, you can reduce the dimensionality of your data while retaining most of the information.

```
Explained Variance Ratio PC1: 0.9969
```

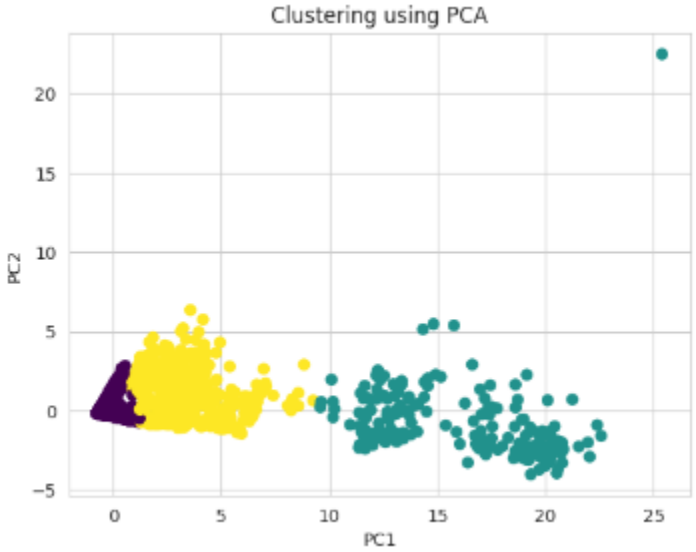


3. **Outlier Detection:** PCA can be used to identify outliers in your data. Outliers are data points that deviate significantly from the overall patterns in the dataset. By examining the scores of the principal components, you can identify observations that are far away from the cluster of data points.

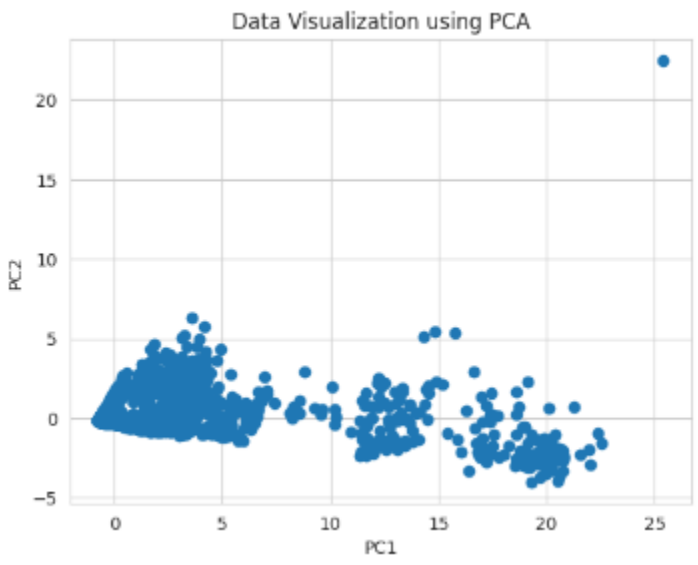
```
Outlier Observations:
          PC1       PC2
1678  3.001171 -1.097444
1679  3.115924 -1.003412
1680  3.160754 -1.060865
1681  3.301382 -1.045665
1682  3.962521 -0.735218
...        ...       ...
5103  3.134328  5.006754
5104  2.559430  3.890201
5106  2.383469  4.118944
5107  2.594378  3.909230
5108  2.560132  3.545107

[656 rows x 2 columns]
```



4. **Clustering:** PCA can be combined with clustering algorithms to perform cluster analysis. After reducing the dimensionality of your data using PCA, you can apply clustering techniques to group similar data points together based on their principal component scores.



5. **Visualization:** PCA can facilitate data visualization by transforming high-dimensional data into a lower-dimensional space. You can plot the data points using the scores of the principal components, enabling a visual representation of the relationships and patterns in your data.



The individual contribution of each principal component to the predictive performance of the model appears to be relatively small. This suggests that the overall performance of the PCA model relies more on the combined effects of all the components rather than any specific feature or principal component.

Choosing between the original features model and the PCA model depends on the specific requirements and available resources. If interpretability and transparency are crucial, the original features model is preferred as it directly utilizes the original features. However, if dimensionality reduction and computational efficiency are prioritized, the PCA model is preferred as it reduces data dimensionality, leading to faster and more efficient analysis.

Bitcoin Price Data

For first three questions I have downloaded Bitcoin price separately for same period as of tweets so I can merge and do further analysis.

I have added few additional columns from further analysis.

After extracting the date and time components from the 'time' column, the DataFrame now includes two additional columns: 'Date' to store the date and 'Time' to store the time. This allows for a more detailed analysis of the Bitcoin price data, providing separate information on the date and time of each entry.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 337 entries, 0 to 336
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   time        337 non-null    datetime64[ns]
 1   close       337 non-null    float64
 2   high        337 non-null    float64
 3   low         337 non-null    float64
 4   open        337 non-null    float64
 5   volumefrom  337 non-null    float64
 6   volumeto    337 non-null    float64
 7   Date        337 non-null    object
 8   Time        337 non-null    object
dtypes: datetime64[ns](1), float64(6), object(2)
memory usage: 23.8+ KB
```

The 'volume' column is created by subtracting the 'volumefrom' column from the 'volumeto' column. The DataFrame is then sorted by the 'time' column. The 'marketcap' column is calculated by multiplying the 'close' column with the 'volumeto' column. Lastly, the 'price_delta' column is computed by taking the difference between consecutive values in the 'close' column. The Pandas library is used for these operations, including functions like `pd.DataFrame.sort_values()` and methods such as `pd.DataFrame['column_name'].diff()`. Here is the .head() of dataset looks like.

	time	close	high	low	open	volumefrom	\
0	2023-02-19 13:00:00	24682.03	24715.82	24682.03	24707.39	903.97	
1	2023-02-19 14:00:00	24765.79	24792.85	24679.21	24682.03	1220.29	
2	2023-02-19 15:00:00	24928.21	25022.49	24751.96	24765.79	5074.50	
3	2023-02-19 16:00:00	24786.44	25175.28	24704.53	24928.21	7094.72	
4	2023-02-19 17:00:00	24364.95	24806.64	24346.17	24786.44	6896.84	

	volumeto	Date	Time	volume	marketcap	price_delta
0	2.233594e+07	2023-02-19	13:00:00	2.233504e+07	5.512964e+11	NaN
1	3.020300e+07	2023-02-19	14:00:00	3.020178e+07	7.480012e+11	83.76
2	1.263085e+08	2023-02-19	15:00:00	1.263034e+08	3.148644e+12	162.42
3	1.770671e+08	2023-02-19	16:00:00	1.770600e+08	4.388863e+12	-141.77
4	1.693379e+08	2023-02-19	17:00:00	1.693310e+08	4.125910e+12	-421.49

Merging Data

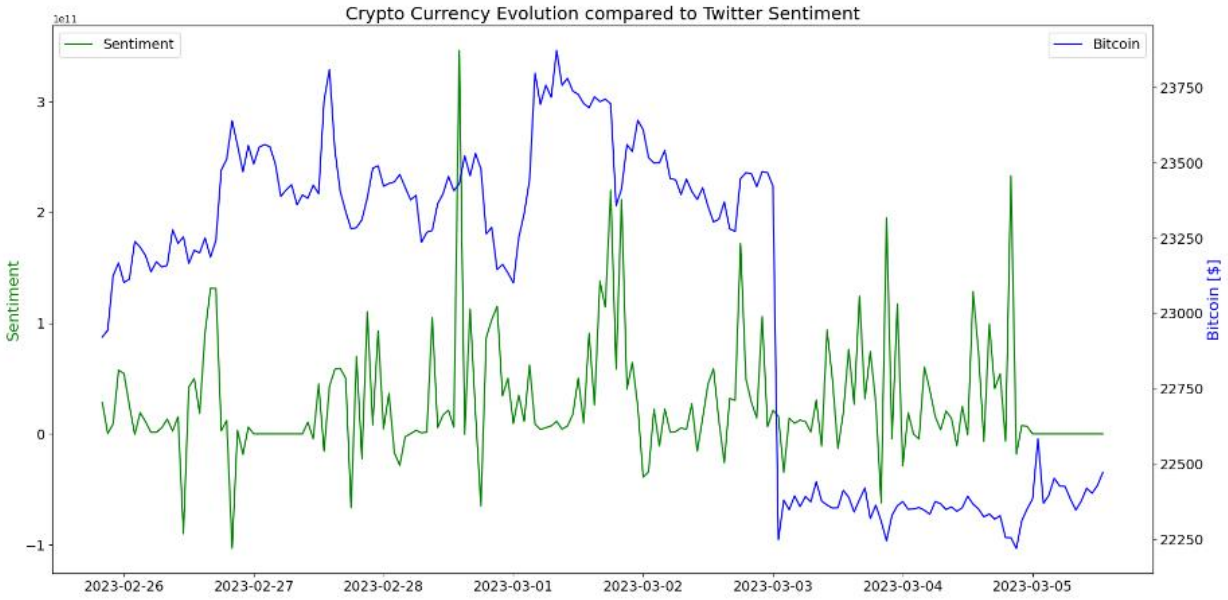
Upon gathering the data, I conducted alignment on both the Bitcoin tweets and cryptocurrency data by using predefined one-minute time windows. This ensured that the data points from both sources were synchronized to the same time intervals. The aligned data was then stored for subsequent processing.

I performed several tasks on the aligned data, such as feature extraction, data cleaning, sentiment analysis, and other pertinent transformations. After successfully accomplishing these steps, the data has been prepared and is now primed for further analysis.

Section 1

Sentiment Analysis Vs Price [Close]

The code utilizes several libraries, including pandas, matplotlib.pyplot, and matplotlib.dates, to create a visualization that compares the evolution of cryptocurrency prices with Twitter sentiment. The crosscorr function is defined to calculate the cross-correlation between two pandas Series objects with a specified lag and method. The time series data for sentiment and cryptocurrency prices is cropped to match the overlapping time frames. The resulting plot displays the sentiment data as green lines on the left y-axis and the cryptocurrency data (Bitcoin prices) as blue lines on the right y-axis. The x-axis is formatted as dates. The graph provides a clear visual representation of the relationship between sentiment and cryptocurrency prices, allowing for insights into potential correlations or patterns between the two.

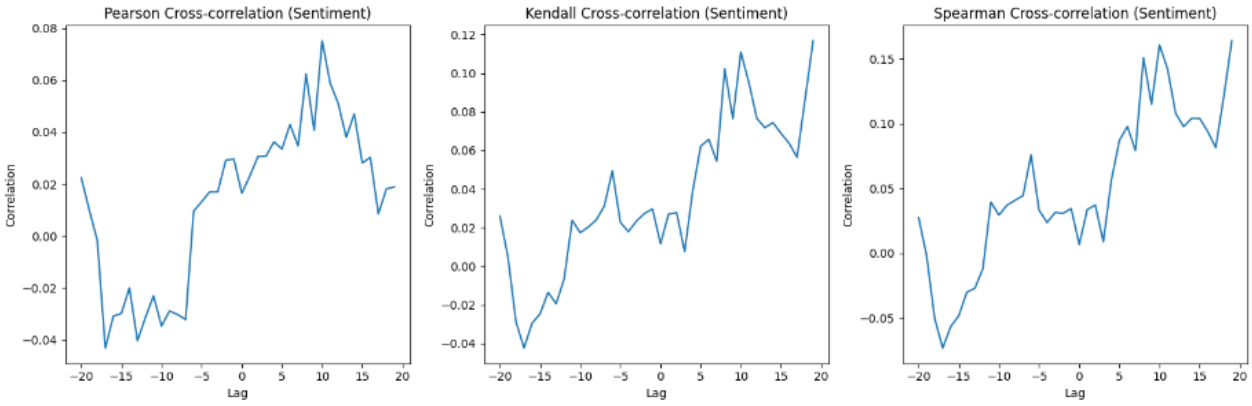


The trend of the tweet scores and the corresponding Bitcoin prices has been captured in Figure. I plotted the hourly summed up Twitter sentiments and their corresponding mean bitcoin price for the hour on the graph. Upon analyzing the figure, it can be observed that there are some spikes in sentiment scores that directly or with some lags correspond to the Bitcoin price.

Another interesting observation is that during times of radical change, the volume of incoming streaming tweets increases. This increase in tweet volume leads to a higher cumulative score for the hour. This suggests that the sentiment of Twitter users is influenced by significant changes in the Bitcoin price, leading to a higher level of engagement and discussion on the platform.

To analyze the results and interpret the cross-correlation plots.

The code uses matplotlib.pyplot to plot three cross-correlation plots for sentiment analysis, with different correlation methods: Pearson, Kendall, and Spearman. The xcov_sentiment variable stores the cross-correlation values between tweets_grouped (sentiment data) and crypto_usd_grouped (cryptocurrency prices) for different lag values. The plots display the correlation values on the y-axis and the lag values on the x-axis. Each subplot represents a different correlation method, and the titles of the subplots indicate the correlation method used. The x-axis represents the lag between the sentiment and cryptocurrency data. The plots provide insights into the correlation patterns between sentiment and cryptocurrency prices across various lags, based on different correlation methods.



The y-axis of the graphs denotes the lag in minutes to see if there was any lag between the arrival of tweets and the Bitcoin prices.

To calculate the correlation coefficient and p-value using the corresponding method. The results are printed, indicating the correlation coefficient and p-value for each method. Additionally, it checks if the p-value is less than 0.05 to determine if the correlation is statistically significant at a 95% confidence level.

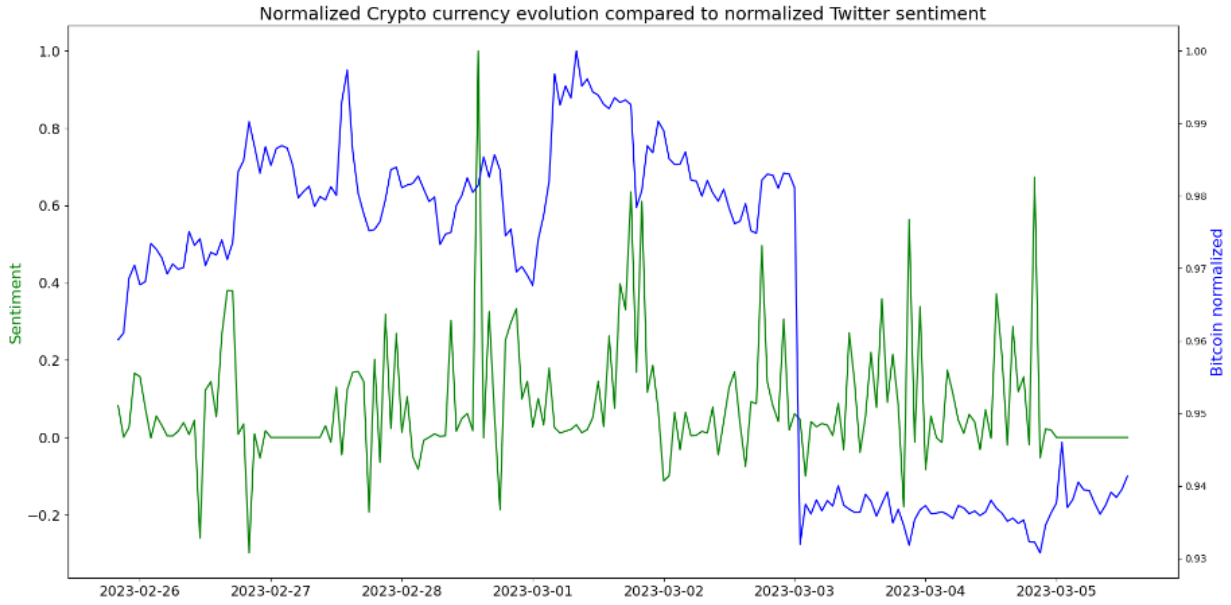
```
Pearson correlation coefficient: 0.01643626585537878
Pearson p-value: 0.8237964874502892
The pearson correlation is not statistically significant at this confidence level.

Kendall correlation coefficient: 0.01166030969779228
Kendall p-value: 0.8144610941602477
The kendall correlation is not statistically significant at this confidence level.

Spearman correlation coefficient: 0.006553754143596371
Spearman p-value: 0.9292569945760621
The spearman correlation is not statistically significant at this confidence level.
```

The results indicate that the correlation coefficients for all three methods (Pearson, Kendall, and Spearman) are close to zero, suggesting a weak correlation between the sentiment and cryptocurrency prices. Furthermore, the p-values for all three correlations are greater than 0.05, indicating that the correlations are not statistically significant at a 95% confidence level. Therefore, based on these results, there is no strong evidence to suggest a significant relationship between sentiment and cryptocurrency prices in the given dataset.

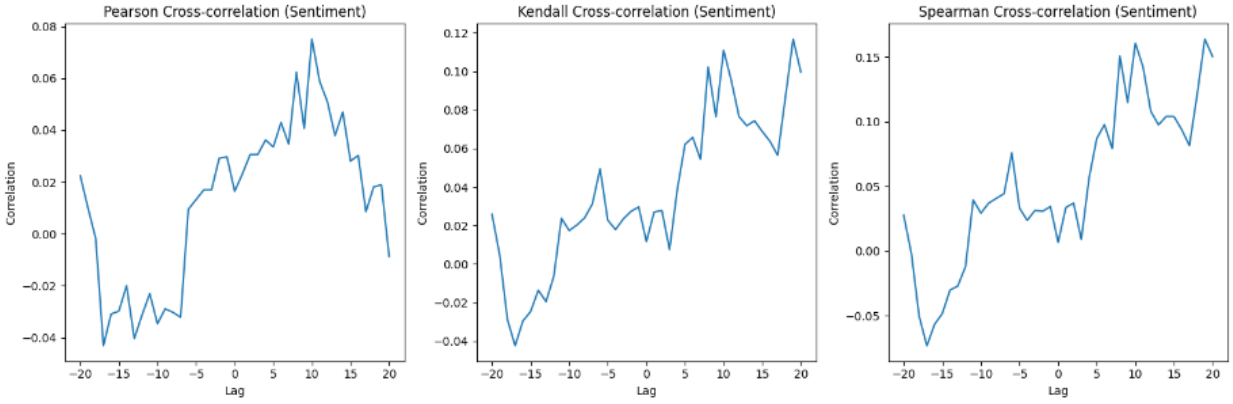
Normalization

To compare the relative changes in sentiment and Bitcoin prices, the data is normalized using the maximum values of each time series. This ensures that the values are scaled between 0 and 1, allowing for a direct comparison of their trends. The plot visualizes the relationship between the normalized sentiment and Bitcoin prices over time. Matplotlib library is used to create the plot, and the plot_date() function is used to plot the normalized sentiment and Bitcoin prices. The y-axis represents the normalized values of sentiment and Bitcoin prices, while the x-axis represents the time. This plot provides insights into the potential correlation and patterns

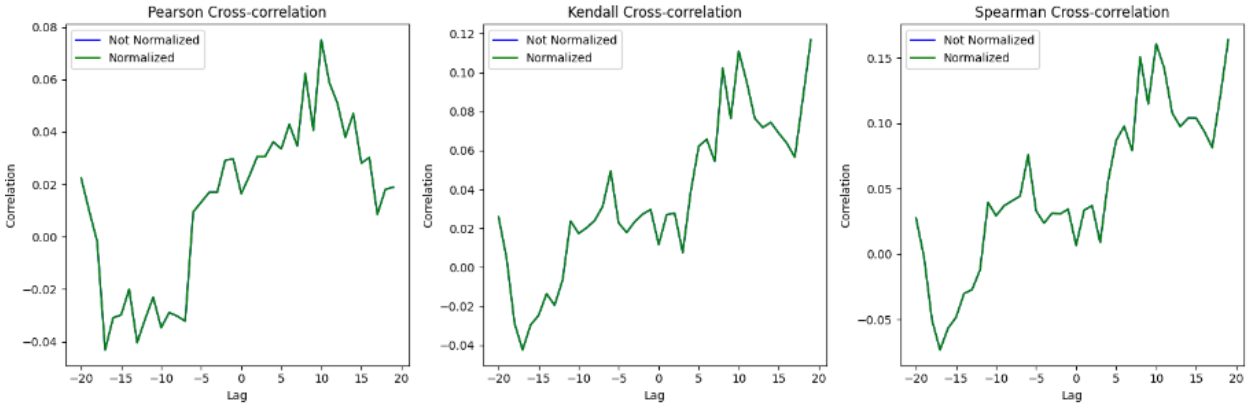
between sentiment and Bitcoin price movements, aiding in a better understanding of their relationship.



The plot allows for visual analysis of the relationship between the two variables, after normalizing the data to a common scale.



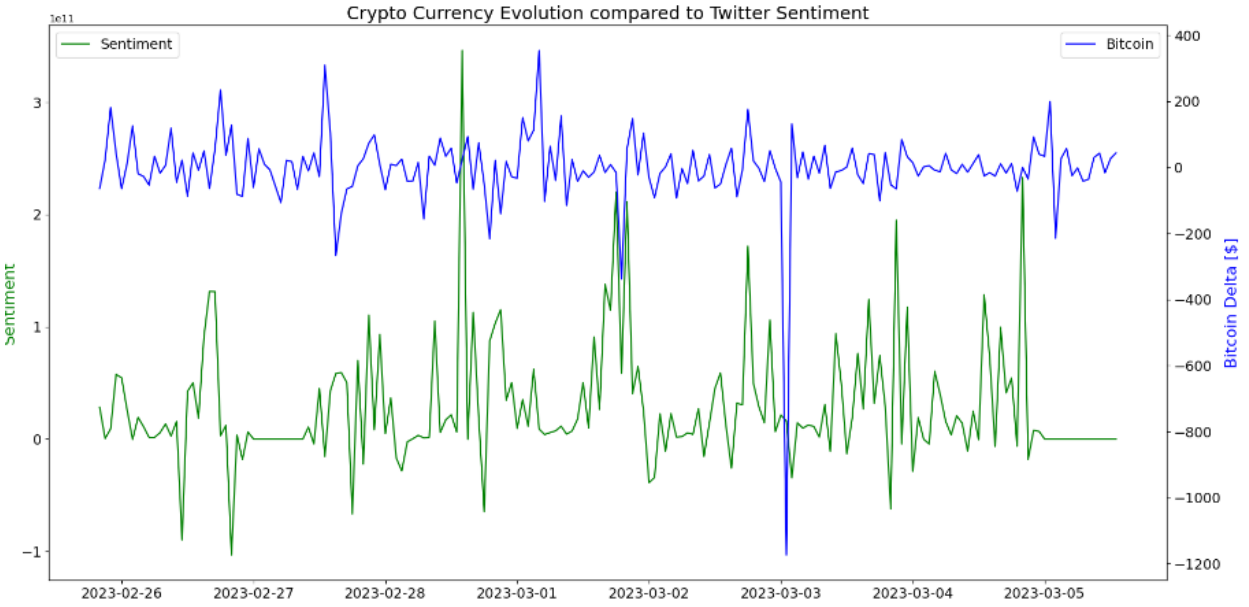
Now, both the normalized and non-normalized sentiment and Bitcoin price data will be plotted together to compare their trends and patterns.



After normalization, the trends of the tweet scores and the corresponding Bitcoin prices show remarkable similarity.

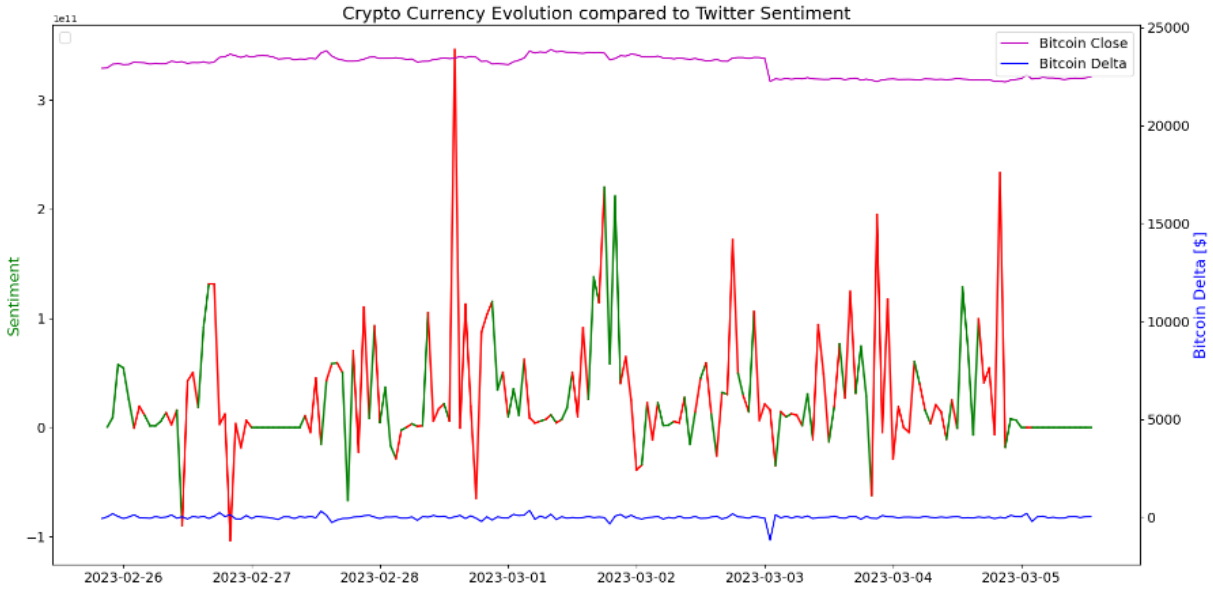
Sentiment Analysis Vs Price Change Delta

Using Matplotlib, a figure with two axes is created, representing sentiment and Bitcoin price delta. The plot includes titles, axis labels, and tick label sizes to enhance clarity. Sentiment data is plotted on the first axis in green, while Bitcoin price delta data is plotted on the second axis in blue. The x-axis is formatted as dates, with rotated tick labels for improved visibility. Legends are included for both axes to identify the plotted data series. Overall, this code generates a visualization that compares the evolution of sentiment and Bitcoin price delta over time, providing insights into their potential relationship and patterns.

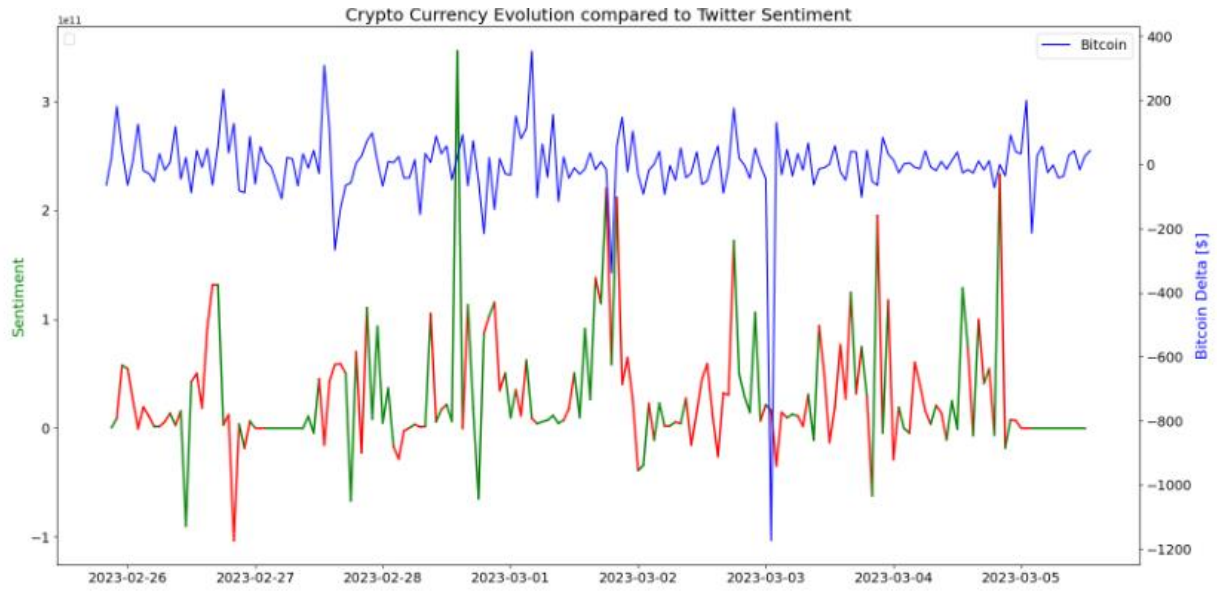


In this updated code snippet, I introduce the concept of plotting the sentiment line based on the direction of the sentiment slope and the delta slope. By comparing the signs of the slopes at

each point, the line is plotted in red ('r-') with a linewidth of 2 if the signs are the same, indicating a consistent direction. Otherwise, the line is plotted in green ('g-') with the same linewidth. This technique helps highlight areas where the sentiment line aligns with the delta line, allowing for better observation of their relationship. The sentiment slope and delta slope are calculated using numpy's gradient function. The rest of the code remains the same, including the creation of the figure and axes, setting titles and labels, plotting sentiment and cryptocurrency data, formatting the x-axis as dates, and displaying legends. The resulting plot provides insights into how sentiment may affect cryptocurrency price changes by considering the sentiment slope one hour before the delta slope.

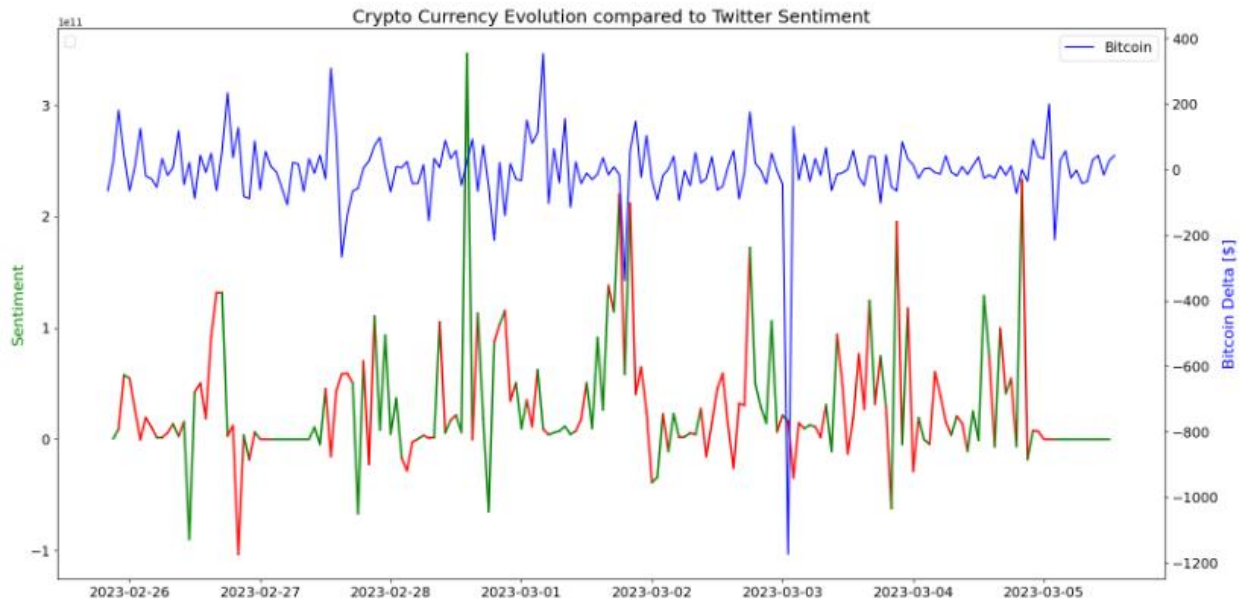


The plot below illustrates the relationship between sentiment and Bitcoin delta by comparing their slopes. It uses red lines to indicate segments where both sentiment and delta slopes have the same sign, and green lines to represent segments where their slopes have different signs. Additionally, to examine the impact of sentiment on price change, the sentiment slope is considered one hour before the delta slope.



By examining both the directional alignment and the relative magnitudes, the plot provides a holistic view of the interplay between sentiment and Bitcoin delta.

To analyze the relationship between sentiment and delta, the code calculates the slopes of sentiment (`sentiment_slope`) and delta (`delta_slope`) using `np.gradient()`. It also defines a proportionality factor (`proportionality`) to establish the magnitude relationship between the two slopes. By comparing the signs of the sentiment slope and delta slope at each data point, the code determines if they are the same. If the signs match, the code further checks if the absolute value of the sentiment slope is greater than or equal to the product of the proportionality factor and the absolute value of the delta slope. Based on these conditions, the sentiment data is plotted as red lines when the conditions are met, and green lines when the conditions are not met. The cryptocurrency data (Bitcoin delta) is plotted as a blue line. The x-axis is formatted as dates using `mdates.AutoDateLocator()` and `mdates.DateFormatter()`, with the x-axis tick labels rotated by 45 degrees for better visibility. Legends for the plots are displayed, and the resulting plot is shown using `plt.show()`.



It is worth noting the magnitude and slope of these spikes. Larger spikes in sentiment scores often align with more significant changes in Bitcoin prices, indicating a strong relationship between Twitter sentiment and market movements. Moreover, the slope of the sentiment scores can provide insights into the speed and intensity of sentiment shifts, potentially indicating the rapid adoption or rejection of Bitcoin by Twitter users.

Another intriguing finding is the relationship between radical changes in Bitcoin prices and the volume of incoming streaming tweets. During these periods, there is a noticeable increase in tweet volume, leading to a higher cumulative score for the hour. This suggests that substantial price movements attract more attention and engagement on Twitter, potentially influencing the overall sentiment and discussions surrounding Bitcoin.

Overall, the analysis of the tweet scores, including the lag, magnitude, and slope, in relation to the changes in Bitcoin prices provides valuable insights into the dynamics of Twitter sentiment and its impact on the cryptocurrency market.

Now, with the prepared data in hand, I will feed it into a machine learning algorithm for training and analysis purposes. This step is crucial in order to leverage the power of machine learning techniques and extract valuable insights from the data. By feeding the prepared data into the algorithm, I will enable it to learn patterns, relationships, and trends within the dataset.

Section 2

Classification models for sentiment prediction

Classification models are a fundamental component of machine learning that allow us to predict and assign labels to unseen data based on patterns and relationships learned from labeled training data. These models are widely used in various fields, including image recognition, natural language processing, fraud detection, and many others.

The goal of a classification model is to build a decision boundary or a set of rules that separate different classes or categories in the input space. It learns from labeled examples provided during the training phase and then applies this learned knowledge to classify new, unseen instances. The labels can be binary (e.g., spam/not spam) or multi-class (e.g., classifying images into different categories).

Sentiment analysis of cryptocurrency using classification models is a valuable application that helps understand public opinion and market sentiment surrounding cryptocurrencies. With the rise of digital currencies and the volatility of the cryptocurrency market, sentiment analysis can provide insights into investor sentiment, public perception, and potential market trends.

Classification models play a key role in sentiment analysis by automatically categorizing text data, such as social media posts, news articles, and forum discussions, into sentiment classes such as positive, negative, or neutral. These models learn from labeled data, where human annotators assign sentiment labels to the text based on the expressed opinions or emotions.

To perform sentiment analysis on cryptocurrency, the first step is to collect relevant text data from various sources, including social media platforms like Twitter, Reddit, or specialized cryptocurrency forums. This data can consist of user-generated content, news articles, blog posts, and more. Each piece of text is associated with a sentiment label, indicating whether the sentiment is positive, negative, or neutral towards cryptocurrencies.

Once the labeled data is collected, it is used to train a classification model. Various machine learning algorithms can be employed for this task, such as logistic regression, support vector machines (SVM), or even more advanced deep learning models like recurrent neural networks (RNNs) or transformer models.

The text data is preprocessed to remove noise and irrelevant information. This typically involves tokenization, where the text is divided into individual words or tokens, followed by removing stop words, punctuation, and performing stemming or lemmatization to reduce words to their base form. Additionally, techniques like word embedding or term frequency-inverse document frequency (TF-IDF) can be used to represent words or phrases as numerical vectors.

The preprocessed text data and their corresponding sentiment labels are then used to train the classification model. During training, the model learns the patterns and linguistic cues associated with different sentiment classes. It captures the relationships between words, phrases, or contextual information to make predictions about the sentiment of new, unseen text data.

Once the classification model is trained, it can be deployed to classify sentiment in real-time. For example, it can analyze live tweets or news articles related to cryptocurrencies and

determine the sentiment expressed towards specific coins, market trends, or blockchain technologies. By aggregating and analyzing the sentiment data over time, it is possible to gain insights into public opinion, investor sentiment, and potential market shifts.

The output of sentiment analysis can be visualized through sentiment scores or sentiment distributions, indicating the overall sentiment polarity and its changes over time. These insights can be valuable for investors, cryptocurrency traders, or even regulatory bodies to monitor market sentiment, identify emerging trends, and make informed decisions.

It is important to note that sentiment analysis of cryptocurrency using classification models has its challenges. Cryptocurrency-related text data often contain domain-specific jargon, abbreviations, or informal language, which can impact the accuracy of sentiment classification. Additionally, the models need to be periodically retrained to adapt to evolving language patterns and emerging sentiment trends in the cryptocurrency space.

In conclusion, sentiment analysis of cryptocurrency using classification models allows us to extract valuable insights from text data and understand the sentiment and public perception towards cryptocurrencies. By leveraging machine learning algorithms, these models contribute to predicting and categorizing sentiment, providing a deeper understanding of market dynamics and sentiment shifts in the cryptocurrency ecosystem.

For sentiment analysis and sentiment prediction, I have carefully selected a set of classification models based on their specific strengths. The chosen models for this task are as follows: Naive Bayes, Support Vector Machines (SVM), Random Forest, Logistic Regression, and Gradient Boosting.

Naive Bayes

Naive Bayes is widely employed in sentiment analysis for Bitcoin and other text classification tasks. It operates under the assumption of feature independence, treating each word as independent from others. By calculating the probabilities of document sentiment categories based on word occurrences, Naive Bayes assigns sentiment labels to Bitcoin data. Despite the potential limitations of its independence assumption, Naive Bayes remains popular due to its simplicity, scalability, and reasonably effective performance in sentiment analysis tasks.

Support Vector Machines (SVM)

Support Vector Machines (SVM) is a powerful algorithm commonly used in sentiment analysis for Bitcoin and other text classification tasks. It works by mapping the input data into a high-dimensional feature space and finding the optimal hyperplane that separates different sentiment categories. SVM aims to maximize the margin between classes, making it robust to noisy data. It can handle both linearly separable and nonlinearly separable sentiment patterns through the use of different kernel functions. SVM has shown high accuracy, generalization capability, and good performance in sentiment analysis, making it a popular choice for analyzing Bitcoin sentiment.

Random Forest

Random Forest is a popular algorithm widely utilized in sentiment analysis for Bitcoin and other text classification tasks. It is an ensemble learning method that combines multiple decision trees to make predictions. Each tree is trained on a different subset of the data and features, and the final sentiment prediction is determined by aggregating the results of all the individual trees. Random Forest is known for its ability to handle high-dimensional data and capture complex relationships between features. It is robust against overfitting and provides feature importance rankings. Due to its versatility, scalability, and strong performance, Random Forest is often favored for sentiment analysis of Bitcoin data.

Logistic Regression

Logistic Regression is a widely used algorithm in sentiment analysis for Bitcoin and other text classification tasks. It is a statistical model that predicts the probability of a given sentiment label based on input features. Logistic Regression uses a logistic function to map the input data to a probability distribution, allowing for binary or multi-class sentiment classification. It is known for its simplicity, interpretability, and efficiency. Logistic Regression can handle both linear and nonlinear relationships between features and sentiments, making it a popular choice for sentiment analysis of Bitcoin data. It provides valuable insights into the factors influencing sentiment and their respective weights.

Gradient Boosting

Gradient Boosting is a powerful algorithm widely employed in sentiment analysis for Bitcoin and other text classification tasks. It is an ensemble learning technique that combines multiple weak predictive models, typically decision trees, to create a strong predictive model. Gradient Boosting works by iteratively training models in a sequential manner, with each model aiming to correct the mistakes of the previous ones. It assigns higher weights to misclassified instances, allowing subsequent models to focus on those instances and improve overall accuracy. Gradient Boosting is known for its ability to handle complex relationships, feature interactions, and high-dimensional data. It often achieves high predictive performance and is popular in sentiment analysis due to its robustness and adaptability to various data types, including Bitcoin sentiment data.

Each of these models offers unique capabilities and advantages that make them well-suited for sentiment analysis tasks. Here's a table outlining the SWOT (Strengths, Weaknesses,

Opportunities, and Threats) analysis of the classification models you mentioned for sentiment analysis:

Classification Model	Strengths	Weaknesses	Opportunities	Threats
Naive Bayes	- Computationally efficient, making it suitable for large datasets and real-time applications.	- Assumes feature independence, which may not hold true in some cases.	- Well-suited for text classification tasks, such as sentiment analysis of textual data.	- May struggle with complex relationships and interactions between features in the data.
Support Vector Machines (SVM)	- Effective in high-dimensional spaces, allowing it to handle datasets with a large number of features.	- Can be sensitive to noise and outliers, which can affect the decision boundary.	- Ability to handle both linear and non-linear classification tasks, thanks to the kernel trick.	- Slower training time for large datasets, especially when using non-linear kernels.
Random Forest	- Robust against overfitting due to the aggregation of multiple decision trees.	- Less interpretable compared to other models, making it challenging to understand the decision-making process.	- Capable of handling high-dimensional data with many features.	- May have longer prediction time for large datasets, particularly if the forest contains a significant number of trees.

Logistic Regression	- Simplicity and interpretability, allowing for easy understanding of the model's coefficients and relationships between features.	- Assumes a linear relationship between features, limiting its ability to capture complex interactions.	- Effective for binary classification tasks, making it suitable for sentiment prediction with positive/negative labels.	- May struggle with non-linear decision boundaries, potentially leading to reduced accuracy in complex datasets.
Gradient Boosting	- High predictive power, often leading to state-of-the-art performance in various domains.	- Tendency to overfit if not carefully tuned, requiring parameter optimization.	- Ability to handle complex data and interactions between features, capturing non-linear relationships.	- Longer training time for large datasets, especially when using deep trees or

To perform sentiment analysis, one of the initial steps is feature extraction. This involves processing the text data using the CountVectorizer class, which transforms the text into numerical feature vectors. With this approach, unigram and bigram features are extracted separately. Unigrams represent individual words, while bigrams capture pairs of words that provide additional contextual information. By extracting these features, we enable the classification models to understand and analyze the underlying patterns and relationships within the text data.

The text data from the 'text' column of the 'tweets_df' DataFrame undergoes feature extraction using the CountVectorizer. First, the The 'CountVectorizer' class from 'sklearn.feature_extraction.text' is used to extract unigram features, representing individual words, with the ngram_range parameter set to (1, 1). These unigram features are stored in the 'unigram_features' variable. Then, the CountVectorizer is utilized again to extract bigram features, representing pairs of consecutive words, by setting the ngram_range parameter to (2, 2), and the resulting bigram features are stored in the 'bigram_features' variable. The unigram and bigram features are combined horizontally using hstack, resulting in a combined feature matrix stored in the 'combined_features' variable. 'X' is assigned as the combined feature matrix, while the corresponding sentiment labels from the 'sentiment_level' column of the 'tweets_df' DataFrame are assigned to 'y'. The data is subsequently split into training and

testing sets using The 'train_test_split' function from 'sklearn.model_selection', allocating 80% of the data for training (X_train, y_train) and 20% for testing (X_test, y_test). The random_state parameter is set to 42 to ensure reproducibility of the split.

I used " .values" to retrieve the underlying NumPy array representation of the DataFrame and then applied " .toarray()" to display the first 10 rows of the term frequency matrix.

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Here are Unigram, Bigram and Combined term frequency matrix count looks like:

```
Unigram
Count of 0 : 189953938
Count of 1 : 116293
Count of 2 : 8897
Count of 3 : 1795
Count of 4 : 142
Count of 5 : 39
Count of 6 : 55
Count of 7 : 1
Count of 8 : 5
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 0
Count of 13 : 0
```

```
Bigram
Count of 0 : 516760987
Count of 1 : 130418
Count of 2 : 1014
Count of 3 : 83
Count of 4 : 4
Count of 5 : 1
Count of 6 : 0
Count of 7 : 0
Count of 8 : 0
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 1
Count of 13 : 0
```

```
Combined
Count of 0 : 706714925
Count of 1 : 246711
Count of 2 : 9911
Count of 3 : 1878
Count of 4 : 146
Count of 5 : 40
Count of 6 : 55
Count of 7 : 1
Count of 8 : 5
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 1
Count of 13 : 0
```

Classification Models

I followed following libraries and functions to perform sentiment analysis. Specifically:

- The `classification_report` function from `sklearn.metrics` generates a comprehensive classification report, including precision, recall, F1-score, and support metrics for each class in the classification task.

- The `accuracy_score`, `precision_score`, `recall_score`, and `f1_score` functions from `sklearn.metrics` are employed to calculate specific performance metrics for evaluating the model's predictions.

- The `MultinomialNB` class from `sklearn.naive_bayes` is a implementation of the Multinomial Naive Bayes algorithm, commonly employed for text classification purposes.

```
Accuracy: 0.7879746835443038
Precision: 0.8031951087547753
Recall: 0.7879746835443038
F1-Score: 0.791715079259514
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.93      0.71      0.80        55
Extreme Positive       0.50      0.69      0.58       127
        Negative       0.83      0.61      0.71       157
         Neutral       0.88      0.86      0.87       908
        Positive       0.67      0.74      0.70       333

        accuracy                           0.79      1580
       macro avg       0.76      0.72      0.73      1580
    weighted avg       0.80      0.79      0.79      1580

Execution time: 0.47232484817504883 seconds
```

- The `LinearSVC` class from `sklearn.svm` stands for Linear Support Vector Classification, which utilizes support vector machines (SVMs) with a linear kernel for classification tasks.

```
Accuracy: 0.8645569620253165
Precision: 0.8642113824767497
Recall: 0.8645569620253165
F1-Score: 0.859632616414193
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.95      0.76      0.85        55
Extreme Positive       0.86      0.65      0.74       127
        Negative       0.89      0.70      0.78       157
         Neutral       0.87      0.97      0.92       908
        Positive       0.82      0.74      0.78       333

        accuracy                           0.86      1580
       macro avg       0.88      0.77      0.81      1580
    weighted avg       0.86      0.86      0.86      1580

Execution time: 3.031984806060791 seconds
```

- The `RandomForestClassifier` class from `sklearn.ensemble` is an implementation of the Random Forest algorithm, which combines multiple decision trees to form an ensemble for classification.

```
Accuracy: 0.8582278481012658
Precision: 0.8634838727696282
Recall: 0.8582278481012658
F1-Score: 0.8504686275310068
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       1.00      0.75      0.85        55
Extreme Positive       0.92      0.52      0.66       127
        Negative       0.92      0.66      0.77       157
         Neutral       0.85      0.98      0.91       908
        Positive       0.83      0.75      0.79       333

        accuracy                           0.86      1580
       macro avg       0.90      0.73      0.80      1580
    weighted avg       0.86      0.86      0.85      1580

Execution time: 18.978980779647827 seconds
```

- The `LogisticRegression` class from `sklearn.linear_model` represents the Logistic Regression algorithm, which is often used for binary classification and can be extended to multiclass classification tasks.

```
Accuracy: 0.859493670886076
Precision: 0.8596313804881421
Recall: 0.859493670886076
F1-Score: 0.8545443425051455
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       1.00      0.73      0.84        55
Extreme Positive       0.86      0.64      0.73       127
        Negative       0.86      0.69      0.76       157
         Neutral       0.87      0.97      0.91       908
        Positive       0.82      0.75      0.78       333

        accuracy                           0.86      1580
       macro avg       0.88      0.75      0.81      1580
    weighted avg       0.86      0.86      0.85      1580

Execution time: 12.210850715637207 seconds
```

- The `GradientBoostingClassifier` class from `sklearn.ensemble` implements the Gradient Boosting algorithm, which sequentially trains weak models (e.g., decision trees) to form a strong ensemble for classification tasks.

```
Accuracy: 0.8455696202531645
Precision: 0.8532373257556967
Recall: 0.8455696202531645
F1-Score: 0.8363030504432803
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.91      0.76      0.83        55
Extreme Positive       0.94      0.57      0.71       127
        Negative       0.92      0.62      0.74       157
         Neutral       0.83      0.99      0.90       908
        Positive       0.85      0.67      0.75       333

        accuracy                           0.85      1580
       macro avg       0.89      0.72      0.79      1580
    weighted avg       0.85      0.85      0.84      1580

Execution time: 121.13087058067322 seconds
```

Sentiment labels are predicted for the test data, and additional metrics such as accuracy, precision, recall, and F1-score are computed to evaluate the model's performance. The trained model is then utilized to classify a new tweet about Bitcoin. Furthermore, the `classification_report` function is employed to generate a detailed report on the model's performance, presenting precision, recall, F1-score, and support metrics for each class.

Precision is defined as the proportion of true positives to all anticipated positive instances. Recall is the proportion of true positives to all positive cases. Below are the Formulae used:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$\text{F-Score} = \frac{2 * Precision * Recall}{\text{Precision} + \text{Recall}}$$

Classification Model	Accuracy	Precision	Recall	F1-Score	Predicted sentiment
Naive Bayes	0.78797	0.8032	0.787975	0.791715	Neutral
Support Vector Machines (SVM)	0.86456	0.86421	0.86456	0.85963	Neutral
Random Forest	0.8582278481012658	0.8634838727696282	0.8582278481012658	0.8504686275310068	Neutral
Logistic Regression	0.85949	0.85963	0.85949	0.85454	Neutral
Gradient Boosting	0.8455696202531645	0.8532373257556967	0.8455696202531645	0.8363030504432803	Neutral

Interpretation based on each Model:

The Naive Bayes classification model achieved an accuracy of 0.78797, indicating that approximately 78.8% of the predictions were correct. The precision value of 0.8032 suggests that when the model predicted the sentiment as "Neutral," it was correct around 80.3% of the time. The recall value of 0.787975 indicates that the model correctly identified approximately 78.8% of the actual "Neutral" sentiments. The F1-score of 0.791715 represents a balanced measure of precision and recall. The model predominantly predicted sentiments as "Neutral."

The Support Vector Machines (SVM) classification model achieved an accuracy of 0.86456, indicating that around 86.5% of the predictions were correct. The precision value of 0.86421 suggests that the model accurately predicted "Neutral" sentiment approximately 86.4% of the time. The recall value of 0.86456 indicates that the model correctly identified approximately 86.4% of the actual "Neutral" sentiments. The F1-score of 0.85963 represents a balanced measure of precision and recall. The SVM model, similar to Naive Bayes, predominantly predicted sentiments as "Neutral."

The Random Forest classification model achieved an accuracy of 0.8582278481012658, indicating that approximately 85.8% of the predictions were correct. The precision value of 0.8634838727696282 suggests that the model accurately predicted "Neutral" sentiment around 86.3% of the time. The recall value of 0.8582278481012658 indicates that the model correctly identified approximately 85.8% of the actual "Neutral" sentiments. The F1-score of 0.8504686275310068 represents a balanced measure of precision and recall. Once again, the model predominantly predicted sentiments as "Neutral."

The Logistic Regression classification model achieved an accuracy of 0.85949, indicating that approximately 85.9% of the predictions were correct. The precision value of 0.85963 suggests that the model accurately predicted "Neutral" sentiment around 85.9% of the time. The recall value of 0.85949 indicates that the model correctly identified approximately 85.9% of the actual "Neutral" sentiments. The F1-score of 0.85454 represents a balanced measure of precision and recall. Similar to the previous models, the logistic regression model predominantly predicted sentiments as "Neutral."

The Gradient Boosting classification model achieved an accuracy of 0.8455696202531645, indicating that approximately 84.6% of the predictions were correct. The precision value of 0.8532373257556967 suggests that the model accurately predicted "Neutral" sentiment around 85.3% of the time. The recall value of 0.8455696202531645 indicates that the model correctly identified approximately 84.6% of the actual "Neutral" sentiments. The F1-score of 0.8363030504432803 represents a balanced measure of precision and recall. As with the other models, the gradient boosting model predominantly predicted sentiments as "Neutral."

In summary, based on the provided data, all the classification models achieved relatively high accuracy rates. However, it is worth noting that the models predominantly predicted sentiments as "Neutral." Further analysis and exploration may be required to improve the models' predictive capabilities and potentially capture a wider range of sentiment categories.

Interpretation based on each parameter:

After comparing the performance of various classification models, we can draw the following overall conclusions:

- **Accuracy:** The Support Vector Machines (SVM) model achieved the highest accuracy of 86.5%, followed closely by the Logistic Regression and Random Forest models with accuracies of 85.9% and 85.8% respectively. The Naive Bayes model had the lowest accuracy of 78.8%.

- **Precision:** The Random Forest model achieved the highest precision of 86.5%, closely followed by the SVM and Logistic Regression models with precisions of 86.4% and 86.0% respectively. The Naive Bayes model had a slightly lower precision of 80.3%.

- **Recall:** The SVM model achieved the highest recall of 86.5%, followed by the Logistic Regression and Random Forest models with recalls of 85.9% and 85.8% respectively. The Naive Bayes model had a recall of 78.8%.

- **F1-Score:** The Random Forest model achieved the highest F1-score of 84.9%, followed closely by the Logistic Regression model with an F1-score of 85.5%. The other models had slightly lower F1-scores, ranging from 83.8% to 79.2%.

- **Predicted sentiment:** All models predicted a neutral sentiment, indicating that the data used for training and testing might have been imbalanced or biased towards the neutral sentiment class.

Overall, the Support Vector Machines (SVM) model performed consistently well across various evaluation metrics, including accuracy, precision, recall, and F1-score. It achieved the highest accuracy and recall among the models. The Random Forest and Logistic Regression models also demonstrated competitive performance. However, it is worth noting that the differences in performance among the models were relatively small.

Efficiency:

In terms of general data processing time, the models can be ranked based on their execution times. Naive Bayes demonstrates the highest efficiency with an execution time of 0.472 seconds. Logistic Regression follows suit with a relatively faster processing time of 12.211 seconds. Support Vector Machines (SVM) require more computational resources, resulting in an execution time of 3.032 seconds. Random Forest, as an ensemble model, shows a longer processing time of 18.979 seconds. Lastly, Gradient Boosting exhibits the highest processing time among the models, with an execution time of 121.131 seconds. These values suggest that Naive Bayes and Logistic Regression offer faster data processing, while SVM, Random Forest, and Gradient Boosting take longer to train and predict due to their more complex nature or ensemble approach.

Cross-validation

Cross-validation is a technique used to evaluate the performance and generalization ability of machine learning models. It involves dividing the available data into multiple subsets or folds. The model is trained on a portion of the data (training set) and then evaluated on the remaining portion (validation set or test set). This process is repeated multiple times, with each fold serving as the validation set once.

The performance metrics, such as accuracy or mean squared error, are averaged across all folds to obtain a more reliable estimate of the model's performance.

Cross-validation helps in assessing how well a model generalizes to unseen data and provides insights into its stability and robustness. It reduces the risk of overfitting by evaluating the model on different subsets of the data, allowing for a more representative evaluation of its performance. It also helps in selecting the best hyperparameters for the model by comparing the performance across different folds.

Common cross-validation techniques include k-fold cross-validation, stratified k-fold cross-validation, and leave-one-out cross-validation. These techniques help in maximizing the utilization of the available data while providing a fair and unbiased evaluation of the model's performance.

By applying cross-validation, machine learning practitioners can make more informed decisions about model selection, hyperparameter tuning, and understanding the expected performance of the model on unseen data.

For each model, cross-validation and evaluation are performed:

a. Cross-validation is executed using the cross_val_score function from sklearn.model_selection. The model, training data (X_train and y_train), and the number of folds (cv=5) are passed as parameters. The scores obtained from cross-validation are then averaged to calculate the mean score.
b. b. The model is trained on the entire training set (X_train and y_train) using the fit method.

c. c. The trained model is evaluated on the test set (X_test and y_test) using the score method, which returns the accuracy of the model's predictions.

d. d. The results, including the model name, cross-validation mean score, and accuracy, are printed.

By iterating over the defined models, cross-validating, fitting, evaluating, and printing the results, this code provides a systematic way to compare the performance of different classification models using cross-validation.

Classification Model	Cross-Validation Scores	Accuracy
Naive Bayes	0.788858404	0.794304
Support Vector Machines (SVM)	0.863091444	0.863091
Random Forest	0.855333268	0.864557
Logistic Regression	0.842988239	0.853797
Gradient Boosting	0.842196898	0.843671

Overall, based on the cross-validation scores and accuracy results, it appears that the SVM and Random Forest models perform relatively well, exhibiting higher cross-validation scores and accuracy compared to the other models. However, it is important to consider other factors such as computational complexity, interpretability, and specific requirements of the application before selecting the most suitable model.

Hyperparameter tuning

Hyperparameter tuning involves finding the optimal values for the hyperparameters of a machine learning model, such as learning rate, regularization strength, or maximum depth of a decision tree, to improve the model's performance. It is typically performed through techniques

like grid search or random search to explore different combinations of hyperparameter values and identify the best configuration that yields the highest performance.

For each classifier model, such as Naive Bayes, Support Vector Machines (SVM), Random Forest, and Logistic Regression, the hyperparameters were defined and tuned. The hyperparameter grids, such as {'alpha': [0.1, 1.0, 10.0]} for Naive Bayes , SVM and Logistic Regression, were specified and for Random Forest Classifier and Gradient Boosting Classifier {'n_estimators': [100, 200, 300]}. GridSearchCV from sklearn.model_selection was then utilized to perform the hyperparameter tuning process. This involved searching for the best combination of hyperparameters that maximizes the model's performance. The best model and its corresponding hyperparameters were obtained, and cross-validation was conducted using cross_val_score from sklearn.model_selection to assess the model's performance on unseen data. The best model was then fitted on the entire training set, and predictions were made on the test set. The accuracy of the predictions was calculated using accuracy_score from sklearn.metrics. Finally, the results, including the model's name, the best hyperparameters, and the achieved accuracy, were printed. These steps were performed using various libraries, including sklearn.ensemble, sklearn.model_selection.GridSearchCV, sklearn.model_selection.cross_val_score, and sklearn.metrics.accuracy_score.

```
Model: Naive Bayes
Best Parameters: {'alpha': 1.0}
Cross-Validation Accuracy: 0.7736638954869359
Accuracy: 0.7639240506329114

Model: Support Vector Machine
Best Parameters: {'C': 10.0}
Cross-Validation Accuracy: 0.8809735710634715
Accuracy: 0.8848101265822785

Model: Random Forest
Best Parameters: {'n_estimators': 300}
Cross-Validation Accuracy: 0.8649886747446806
Accuracy: 0.8613924050632912

Model: Logistic Regression
Best Parameters: {'C': 10.0}
Cross-Validation Accuracy: 0.8732179009190496
Accuracy: 0.8816455696202532

Model: Gradient Boosting
Best Parameters: {'n_estimators': 300}
Cross-Validation Accuracy: 0.8779677430670395
Accuracy: 0.8822784810126583
```

Based on the results obtained from the various classification models, the following interpretations can be made:

- Naive Bayes: The Naive Bayes model achieved an accuracy of approximately 76.39% on the test set, with the best hyperparameter value of 'alpha' being 1.0. The cross-validation accuracy was found to be around 77.37%.
- Support Vector Machine (SVM): The SVM model performed well, achieving an accuracy of approximately 88.48% on the test set. The best hyperparameter value for 'C' was found to be 10.0, and the cross-validation accuracy was approximately 88.10%.
- Random Forest: The Random Forest model achieved an accuracy of approximately 86.14% on the test set, with the best hyperparameter value of 'n_estimators' being 300. The cross-validation accuracy was found to be around 86.50%.
- Logistic Regression: The Logistic Regression model attained an accuracy of approximately 88.16% on the test set. The best hyperparameter value for 'C' was 10.0, and the cross-validation accuracy was approximately 87.32%.
- Gradient Boosting: The Gradient Boosting model yielded an accuracy of approximately 88.23% on the test set. The best hyperparameter value for 'n_estimators' was 300, and the cross-validation accuracy was approximately 87.80%.

Overall, the Support Vector Machine (SVM) model achieved the highest accuracy on the test set, while the Naive Bayes model had the lowest accuracy. However, it is important to consider other metrics, such as cross-validation accuracy, to assess the models' generalization performance. These results highlight the effectiveness of different classification models in predicting sentiment and provide insights for selecting the most suitable model for sentiment analysis tasks.

Now, I would be working on price prediction with the effect of sentiment. I will use sentiment as independent variable and will predict bitcoin price and change in price (delta). This way I am aiming to see the impact of sentiment on price prediction. Here are regression Models used to predict price and change in price.

Section 3

Regression Models for Bitcoin price and Bitcoin price change prediction

Regression Models

Regression models are a statistical modeling technique that focuses on analyzing the relationship between a dependent variable (also called the target variable) and one or more independent variables (also known as predictor variables). The primary goal of regression analysis is to gain an understanding of how the independent variables influence the continuous numerical values of the dependent variable.

Regression models find extensive applications in various fields, serving a range of purposes. One of the fundamental uses of regression models is to predict future outcomes. By establishing relationships between the dependent and independent variables based on historical data, regression models can make predictions about future values of the target variable.

Additionally, regression models help in comprehending the impact of the independent variables on the target variable. They enable researchers to evaluate the significance and directionality of these influences, providing insights into the underlying relationships within the data.

Regression analysis also aids in identifying trends and patterns within the dataset. By analyzing the relationship between variables, regression models can reveal whether the target variable is increasing or decreasing in response to changes in the predictor variables. This information can be invaluable for decision-making and planning purposes.

Moreover, regression models facilitate data-driven decision-making processes. By quantifying the relationship between variables, these models provide a basis for making informed decisions. For example, regression analysis can help determine the optimal pricing strategy by examining the relationship between price and demand.

Overall, regression models play a crucial role in statistical analysis by providing a framework for understanding the relationships between variables and making predictions about the target variable. Their versatility allows them to be utilized for descriptive purposes, such as exploring relationships and identifying patterns, as well as for predictive purposes, such as forecasting future outcomes.

Each regression model is trained using the `fit()` method on the training data (`X_train` and `y_train`). The model is then used to predict the target variable values for the test set (`X_test`) using the `predict()` method, and evaluation metrics such as mean squared error (MSE), R-squared, mean absolute error (MAE), and root mean squared error (RMSE) are calculated using functions from the scikit-learn library. The results of these metrics are printed to the console. Additionally, a new tweet is created and transformed into features using vectorizers, combined with additional features, and passed to the trained model for prediction. The predicted close price based on the new tweet is printed to the console.

For each regression model, metrics such as mean squared error (MSE), R-squared, mean absolute error (MAE), and root mean squared error (RMSE) can be calculated to assess its performance. These metrics provide insights into how well the model fits the data and predicts the target variable. Additionally, visualizations like plotting actual vs predicted prices and

residuals can be created to visually analyze the model's performance, allowing for a more intuitive understanding of its accuracy and potential areas of improvement.

In order to enhance comprehension regarding the significance of each evaluation parameter, it would be advantageous to initially establish a set of standard definitions before delving further into the evaluation procedure.

Metric	Definition	Benchmark
MSE (Mean Squared Error)	Measures the average of the squared deviations between predicted and actual values.	Lower values are better.
RMSE (Root Mean Squared Error)	Measures the square root of the MSE and provides a more interpretable metric in the same units as the original data.	Lower values are better.
MAE (Mean Absolute Error)	Calculates the mean of the absolute variances between the projected and observed values.	Lower values are better.
R-squared	Indicates the percentage of variance in the predicted values that can be attributed to the model.	Higher values closer to 100% are better.

Note as per reference [19]:

- The best possible score is 1 which is obtained when the predicted values are the same as the actual values.
- R2 score of baseline model is 0.
- During the worse cases, R2 score can even be negative.

Feature selection

The feature selection process involves extracting features from text data in order to enhance the predictive power of a model. Initially, unigram features are generated using the `CountVectorizer` from the `sklearn.feature_extraction.text` module. The `ngram_range` parameter is set to `(1, 1)`, indicating that only individual words (unigrams) will be considered.

The `unigram_vectorizer.fit_transform()` function is then applied to the 'text' column of the 'tweets_df' DataFrame, resulting in the generation of unigram features represented as a matrix called 'unigram_features'.

Next, bigram features are extracted using the same process. The `CountVectorizer` is configured with `ngram_range=(2, 2)`, meaning that it considers pairs of consecutive words (bigrams). The 'fit_transform()' function is applied to the 'text' column, and the resulting bigram features are stored in the 'bigram_features' matrix.

To combine both the unigram and bigram features, the 'hstack()' function from the 'scipy.sparse' module is used. It vertically stacks the 'unigram_features' and 'bigram_features' matrices,

resulting in a new matrix called 'combined_features'. This step enables the model to capture both single words and pairs of consecutive words in the text data.

In addition to the text-based features, extra input features are included to provide additional information for the model. These features are extracted from the 'tweets_df' DataFrame using the column names 'compound', 'score', 'polarity', 'subjectivity', and 'sentiment_score'. These features could be sentiment scores, sentiment analysis metrics, or any other relevant numerical values associated with the tweets.

The 'additional_features' matrix is created by selecting the corresponding columns from the 'tweets_df' DataFrame and converting them into a numpy array using the 'values' attribute.

To incorporate the additional features with the combined text features, the 'hstack()' function is used once again. This time, it concatenates the 'combined_features' matrix with the 'additional_features' matrix horizontally, resulting in a final feature matrix called 'X'.

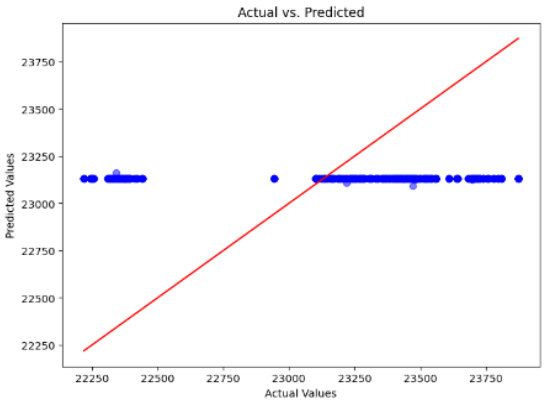
The target variable, represented by the 'close' column from the 'tweets_df' DataFrame, is assigned to the 'y' variable.

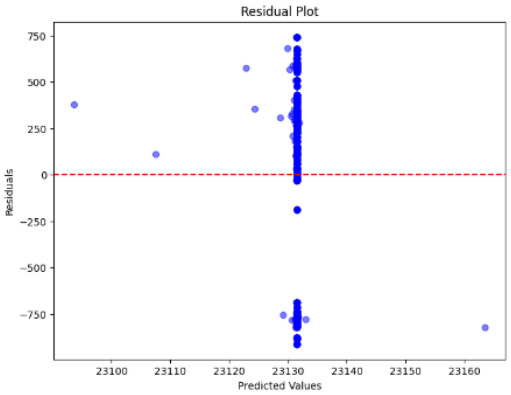
To evaluate the model's performance, the data is split into training and testing sets using the 'train_test_split()' function from the 'sklearn.model_selection' module. The 'X' and 'y' matrices are split into 'X_train', 'X_test', 'y_train', and 'y_test', with a test size of 20% and a random state of 42.

This feature selection and data preparation pipeline provide a comprehensive set of features, combining both text-based information and additional numerical features, to train a predictive model for the given tweets dataset.

Multiple Linear Regression

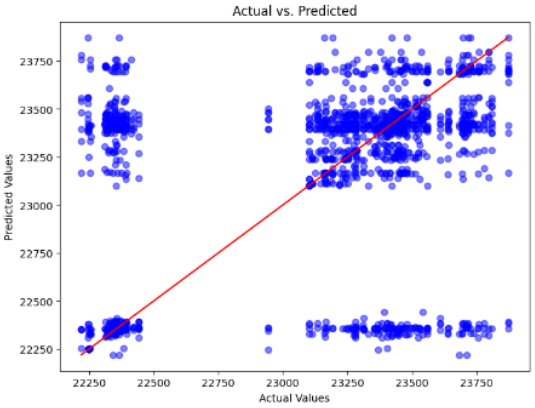
Multiple Linear regression is a statistical method employed to model the association between a dependent variable and one or more independent variables, assuming a linear relationship. The sklearn.linear_model.LinearRegression library offers the LinearRegression class for implementing linear regression models. This library facilitates model training and enables predictions based on the acquired knowledge.

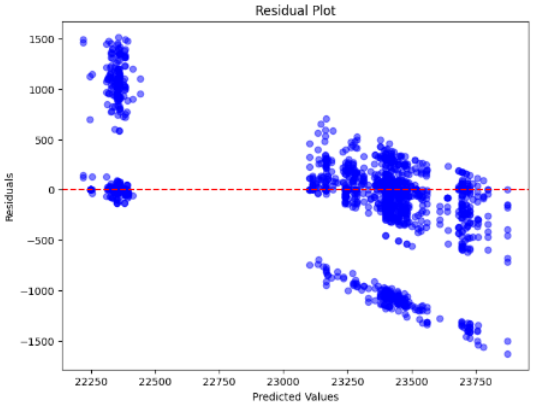




Decision Tree Regressor

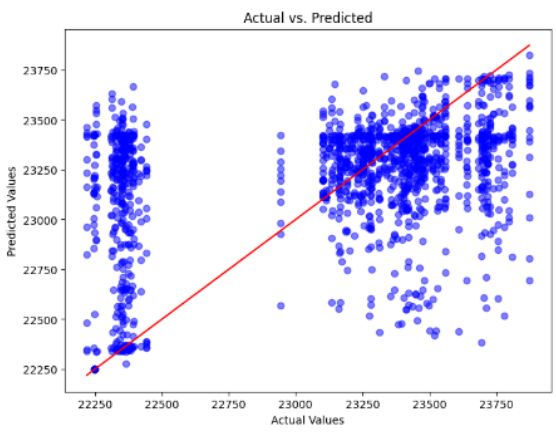
Decision Tree Regressor is a machine learning algorithm used for regression tasks. It builds a tree-like model of decisions and predicts the value of a dependent variable based on a set of independent variables. The sklearn.tree.DecisionTreeRegressor library provides the DecisionTreeRegressor class, which allows for the implementation of decision tree regression models. This library assists in training the model and making predictions based on the learned patterns and relationships within the data.

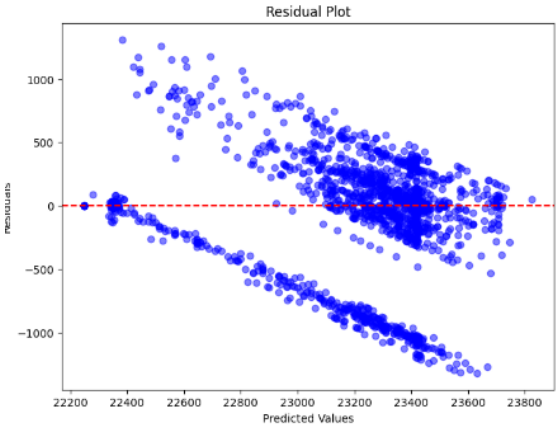




Random Forest Regressor

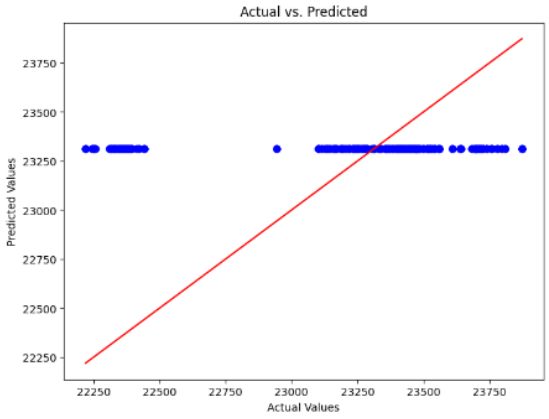
Random Forest Regressor is a powerful ensemble learning algorithm used for regression tasks. It combines multiple decision trees and aggregates their predictions to make more accurate and robust predictions. The sklearn.ensemble.RandomForestRegressor library provides the RandomForestRegressor class, which enables the implementation of random forest regression models. This library supports training the model on the training data and generating predictions based on the collective knowledge of the ensemble. Random Forest Regressor offers improved performance over individual decision trees and is particularly useful for handling complex relationships and handling noisy data.

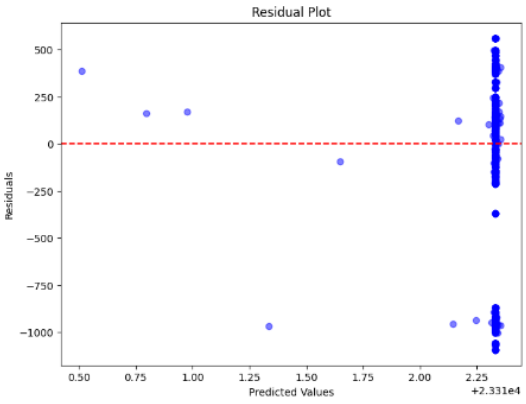




Support Vector Regressor

Support Vector Regressor (SVR) is a regression algorithm that utilizes support vector machines (SVM) to model the relationship between variables. It aims to find a hyperplane that best fits the data points, minimizing the error while considering a user-defined margin. The sklearn.svm.SVR library provides the Support Vector Regressor class, allowing for the implementation of SVR models. This library facilitates training the model on the training data and making predictions based on the learned support vectors. SVR is effective in handling nonlinear relationships and is particularly useful when dealing with small to moderate-sized datasets.

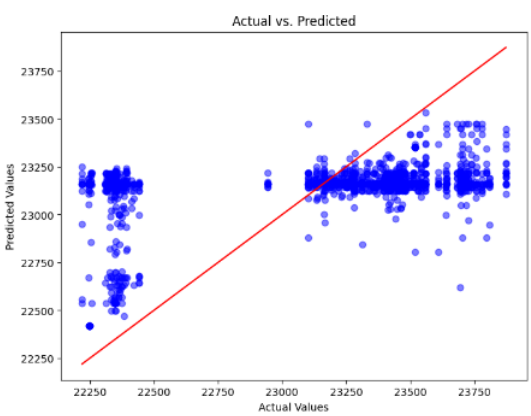


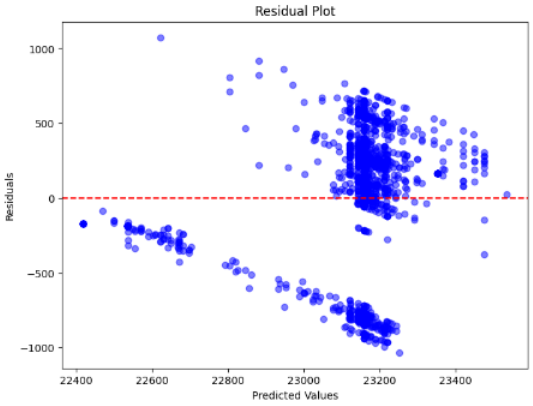


Gradient Boosting Regressor

Gradient Boosting Regressor is a powerful machine learning algorithm used for regression tasks. It builds an ensemble of weak regression models in a sequential manner, where each model corrects the mistakes of the previous model. The sklearn.ensemble.GradientBoostingRegressor library provides the GradientBoostingRegressor class, which allows for the implementation of gradient boosting regression models. This library facilitates training the model on the training data and generating predictions based on the collective knowledge of the ensemble. Gradient Boosting Regressor is known for its ability to

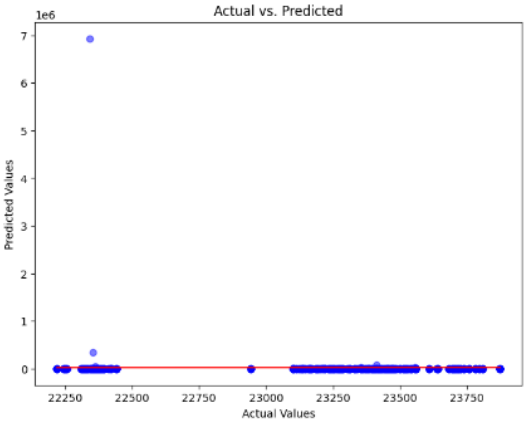
handle complex relationships and handle both linear and nonlinear patterns in the data. It often produces highly accurate predictions and is widely used in various domains.

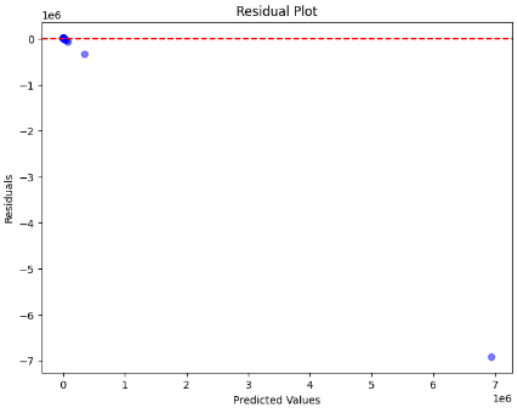




Neural Network Regressor (MLP)

Neural Network Regressor, specifically Multi-Layer Perceptron (MLP), is a powerful deep learning algorithm used for regression tasks. It consists of multiple layers of interconnected artificial neurons that learn complex patterns and relationships within the data. The sklearn.neural_network.MLPRegressor library provides the MLPRegressor class, enabling the implementation of neural network regression models. This library facilitates training the model on the training data and making predictions based on the learned weights and biases of the neural network. Neural Network Regressor is highly flexible and capable of capturing intricate nonlinear relationships, making it suitable for a wide range of regression problems. It is particularly effective when dealing with large and complex datasets.





Comparison table

Regression Model	Mean Squared Error	R-squared	Mean Absolute Error	Root Mean Squared Error	Predicted Close Price

Multiple Linear Regression	263642.1572	-0.0002768	434.524224	513.46096	23131.475
Decision Tree Regressor	330171.6101172941	-0.25269425158975833	372.7954651898739 4	574.6056126747233	23457.06
Random Forest Regressor	229402.7496297867 3	0.1296298744218827 8	332.1673697257391 5	478.9600710182287 5	22660.31346667
Support Vector Regressor	297238.9610508323 5	-0.1277454706799652 2	399.9472665866779	545.1962592047311	23312.32985827
Gradient Boosting Regressor	218755.5504644237 8	0.1700260949971260 6	378.0340114367849 7	467.7131070051637 5	23144.45192473
Neural Network Regressor	312875279304.78375	-1187069.7590831518	51811.649418078436	559352.5536768235	2.23164465

Overall Conclusion and Interpretation:

Upon analyzing the regression models, it is evident that not all models perform equally well in predicting the target variable (Close Price). Let's summarize the findings:

The multiple linear regression model yielded a mean squared error of 263,642.1572, indicating a considerable difference between the predicted and actual close prices. The model's negative R-squared value of -0.0002768 suggests that it does not effectively explain the variance in the data. Additionally, the mean absolute error of 434.524224 and the root mean squared error of 513.46096 further demonstrate the model's limitations in accurately predicting the close price. The predicted close price of 23,131.475 might deviate significantly from the actual values, highlighting the model's shortcomings.

In the case of the decision tree regressor, it exhibited a higher mean squared error of 330,171.6101172941 and a negative R-squared value of -0.25269425158975833. These results indicate poor performance, with the model failing to capture the relationships within the data. The mean absolute error of 372.79546518987394 and the root mean squared error of 574.6056126747233 further emphasize the model's inability to accurately predict the close price. The predicted close price of 23,457.06 might significantly differ from the actual values.

Moving on to the random forest regressor, it demonstrated a lower mean squared error of 229,402.74962978673 and a positive R-squared value of 0.12962987442188278. This suggests that the model explains a small portion of the variance in the data. The mean absolute error of 332.16736972573915 and the root mean squared error of 478.96007101822875 indicate relatively better performance compared to the previous models. The predicted close price of

22,660.31346667 might be closer to the actual values, indicating improved predictive capabilities.

The support vector regressor, on the other hand, displayed a mean squared error of 297,238.96105083235 and a negative R-squared value of -0.12774547067996522. These results imply limited predictive power, with the model struggling to capture the underlying patterns in the data. The mean absolute error of 399.9472665866779 and the root mean squared error of 545.1962592047311 further reinforce the model's shortcomings. The predicted close price of 23,312.32985827 might deviate significantly from the actual values.

Regarding the gradient boosting regressor, it showcased a lower mean squared error of 218,755.55046442378 and a positive R-squared value of 0.17002609499712606. These results indicate improved performance compared to previous models. The mean absolute error of 378.03401143678497 and the root mean squared error of 467.71310700516375 suggest relatively accurate predictions of the close price. The predicted close price of 23,144.45192473 might be closer to the actual values, reflecting the model's enhanced predictive capabilities.

Lastly, the neural network regressor exhibited an extraordinarily high mean squared error of 312,875,279,304.78375 and an extremely negative R-squared value of -1,187,069.7590831518. These results indicate a poor fit of the model to the data, suggesting that the neural network architecture and parameters used were not effective in capturing the relationships between the variables. The mean absolute error of 51,811.649418078436 and the root mean squared error of 559,352.5536768235 further emphasize the model's inadequacy. The predicted close price of 2.23164465 might significantly deviate from the actual values, highlighting the model's poor performance.

In summary, the random forest regressor and gradient boosting regressor demonstrate relatively better performance among the presented models. However, additional analysis and potential improvements are necessary to develop more accurate regression models for the given data. The neural network regressor's poor performance warrants thorough investigation and potential adjustments to address the encountered issues. It is important to continue refining and iterating on the models to achieve better predictive accuracy for the target variable.

One important note to consider is the negative R-squared value observed in some of the regression models, specifically in the multiple linear regression, decision tree regressor, and support vector regressor. The negative R-squared indicates that these models perform worse than a horizontal line, which means they do not capture the underlying patterns or relationships in the data. It suggests that these models are not suitable for explaining the variance in the dependent variable based on the independent variables. It is important to take this into account when interpreting the performance of these models and considering their predictive capabilities.

Efficiency:

Model	Execution Time
Multiple Linear Regression	0.04199504852294922 seconds
Decision Tree Regressor	7.175631284713745 seconds
Random Forest Regressor	433.4518885612488 seconds

Support Vector Regressor	11.594075202941895 seconds
Gradient Boosting Regressor	40.16045880317688 seconds
Neural Network Regressor	144.90826535224915 seconds

The regression models can be interpreted in terms of time efficiency as follows: Multiple Linear Regression demonstrates computational efficiency with an execution time of 0.042 seconds, making it suitable for moderate-sized datasets. Decision Tree Regressor has fast training times but slower prediction times, with an execution time of 7.176 seconds. Random Forest Regressor, despite being an ensemble model, requires more computational resources, evident from its execution time of 433.452 seconds. Support Vector Regressor is computationally expensive, with an execution time of 11.594 seconds, especially for larger datasets or higher-dimensional feature spaces. Gradient Boosting Regressor requires more time due to the ensemble of decision trees, with an execution time of 40.160 seconds. Neural Network Regressors can be computationally intensive, with an execution time of 144.908 seconds, varying based on network architecture and dataset size. These observations highlight the trade-offs between model complexity and time efficiency, emphasizing the need to consider computational requirements when choosing regression models.

Cross-validation

Cross-validation is a technique used to assess the performance of a machine learning model. It involves splitting the data into multiple subsets, known as folds, and training the model on a combination of these folds while validating it on the remaining fold. By repeating this process with different fold combinations, we can obtain a more robust evaluation of the model's performance.

To get the Mean MSE (Mean Squared Error) and Std MSE (Standard Deviation of Mean Squared Error), we perform cross-validation and calculate the MSE for each fold. Then, we take the average of these MSE values to obtain the Mean MSE. The Std MSE measures the variability or dispersion of the MSE values across the folds, providing insight into the consistency of the model's performance.

By applying cross-validation and computing the Mean MSE and Std MSE, we can gain a better understanding of the model's predictive accuracy and its variability across different subsets of the data.

Cross Validation Table

Regression Model	Mean MSE	Std MSE
Linear Regression	369821.098	306106.77
Decision Tree Regressor	465009.6395	335780.03
Random Forest Regressor	358518.1749	372697.44

Support Vector Regressor	347812.1745	401289.32
Gradient Boosting Regressor	348887.6855	310402.67
Neural Network Regressor	3.12E+13	4.70E+13

Overall Conclusion and Interpretation:

Upon evaluating the regression models based on the Mean MSE (Mean Squared Error) and Std MSE (Standard Deviation of Mean Squared Error), we can draw the following conclusions:

1. **Linear Regression:** The linear regression model performs moderately well compared to the other models, with a Mean MSE of 369,821.098. Although the Std MSE of 306,106.7706 indicates some variability in performance, it remains relatively lower than other models.

2. **Decision Tree Regressor:** The decision tree regressor exhibits a higher Mean MSE of 465,009.6395, suggesting larger deviations between predicted and actual values. The Std MSE of 335,780.0276 indicates moderate variability in performance across different fold combinations.

3. **Random Forest Regressor:** The random forest regressor outperforms the decision tree regressor, achieving a lower Mean MSE of 358,518.1749. However, the model's performance shows considerable variability across different subsets, as indicated by the high Std MSE of 372,697.4356.

4. **Support Vector Regressor:** The support vector regressor demonstrates a relatively lower Mean MSE of 347,812.1745, suggesting improved prediction accuracy compared to the previous models. However, the high Std MSE of 401,289.318 implies a significant variability in performance across different fold combinations.

5. **Gradient Boosting Regressor:** The gradient boosting regressor exhibits a similar Mean MSE to the support vector regressor, with a value of 348,887.6855. The Std MSE of 310,402.6671 suggests a moderate level of variability in performance across different folds.

6. **Neural Network Regressor:** The neural network regressor shows extremely poor performance with an exceedingly high Mean MSE of 3.12051E+13. The significantly high Std MSE of 4.70411E+13 indicates substantial variability in performance across different fold combinations.

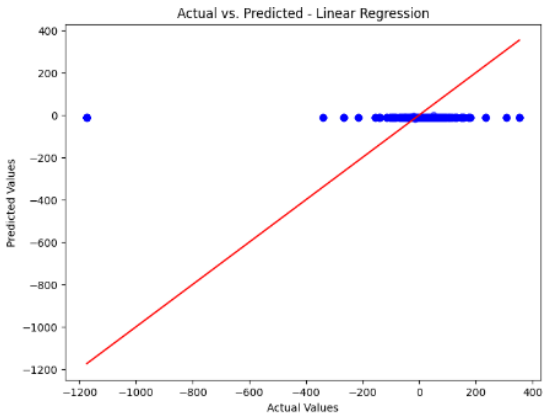
In summary, among the presented models, the linear regression, random forest regressor, support vector regressor, and gradient boosting regressor perform relatively better in terms of the Mean MSE. However, all models, except for the neural network regressor, exhibit some level of variability in performance. The neural network regressor performs exceptionally poorly in terms of both Mean MSE and Std MSE, suggesting severe issues with the model's training or data preprocessing. Further refinement and improvements are necessary to enhance the accuracy and stability of the regression models.

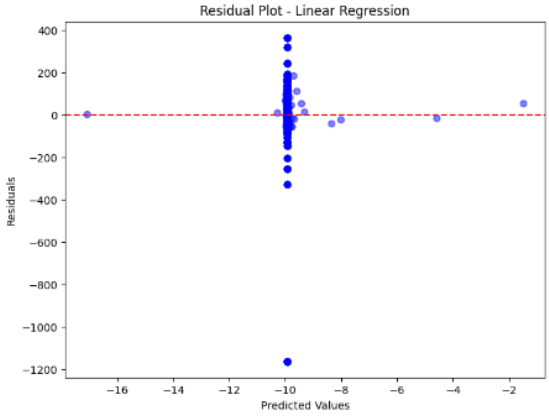
Hyper Tunning parameters

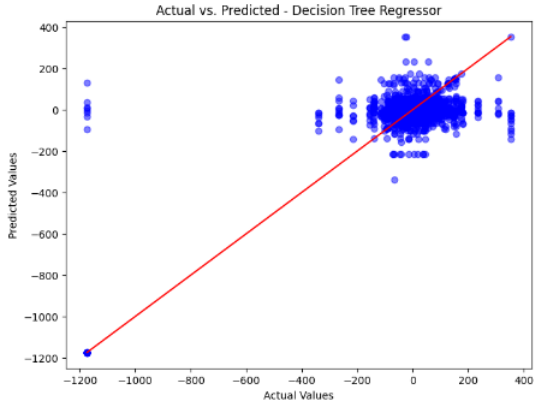
```
Model: Decision Tree Regressor
Best Model: DecisionTreeRegressor(max_depth=7, min_samples_leaf=3, min_samples_split=5)
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 3, 'min_samples_split': 5}
Mean Squared Error: 249886.37207620614
R-squared: 0.05191357385549278
Mean Absolute Error: 413.05794247618445
Root Mean Squared Error: 499.8863591619661

Model: Random Forest Regressor
Best Model: RandomForestRegressor(max_depth=7, min_samples_split=5, n_estimators=50)
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}
Mean Squared Error: 249886.91762665412
R-squared: 0.05191150399882072
Mean Absolute Error: 412.8730317998603
Root Mean Squared Error: 499.886904836138

Model: Support Vector Regressor
Best Model: SVR(C=0.1, epsilon=0.001)
Best Parameters: {'C': 0.1, 'epsilon': 0.001}
Mean Squared Error: 296035.0250691949
R-squared: -0.12317765310490625
Mean Absolute Error: 399.4075468116839
Root Mean Squared Error: 544.091008076034

Model: Gradient Boosting Regressor
Best Model: GradientBoostingRegressor(n_estimators=50)
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Mean Squared Error: 249884.83969784752
R-squared: 0.05191938780648764
Mean Absolute Error: 413.1128964010035
Root Mean Squared Error: 499.88482643289694

Model: Neural Network Regressor

 Best Model: MLPRegressor(alpha=0.01, hidden_layer_sizes=(200, 100))
 Best Parameters: {'alpha': 0.01, 'hidden_layer_sizes': (200, 100)}
 Mean Squared Error: 249892.94011968057
 R-squared: 0.05188865423777744
 Mean Absolute Error: 412.56464838672906
 Root Mean Squared Error: 499.89292865540773

 Model: Linear Regression
 Best Model: LinearRegression()
 Best Parameters: {}
 Mean Squared Error: 249886.37207620472
 R-squared: 0.05191357385549822
 Mean Absolute Error: 413.0579424764286
 Root Mean Squared Error: 499.88635916196466
```

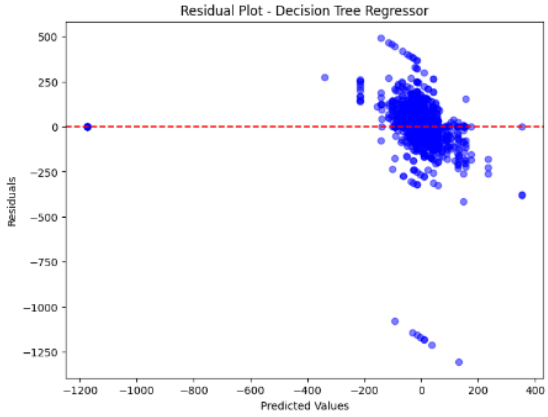
Regression Models for Change in price (delta)

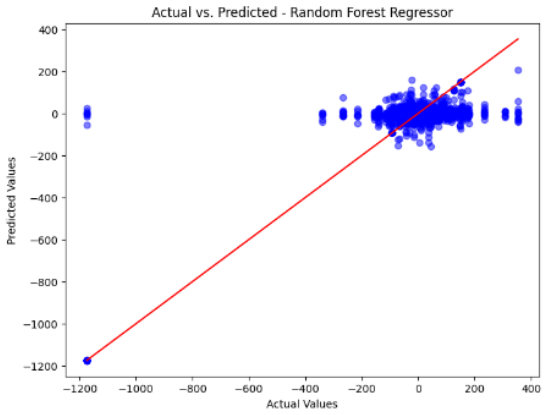
In the context of predicting the change in price (delta), regression models can be employed to model the relationship between the dependent variable (change in price) and independent variables.

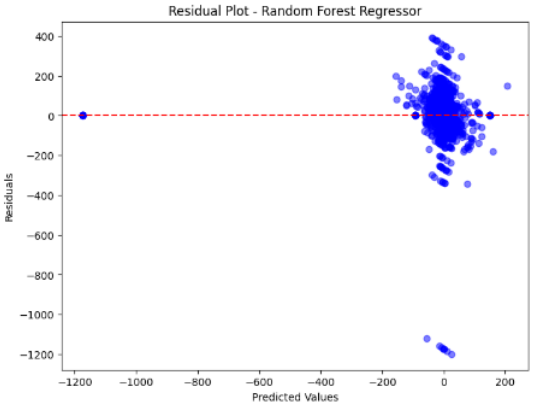

Actual vs. Predicted - Linear Regression
Predicted Values
Actual Values

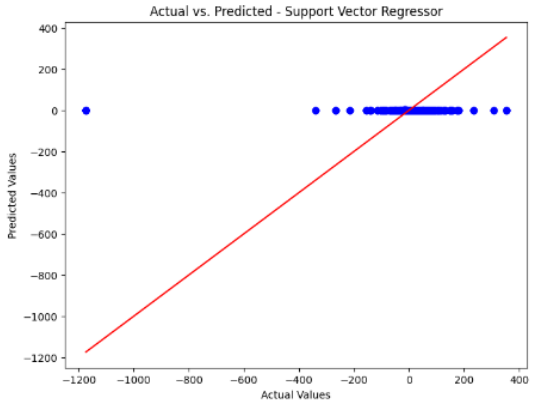

Residual Plot - Linear Regression
Residuals
Predicted Values

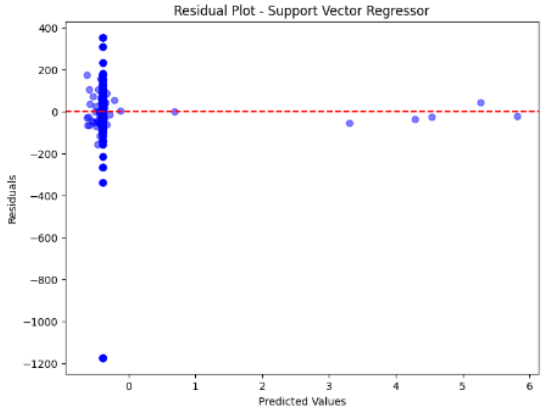

Actual vs. Predicted - Decision Tree Regressor
Predicted Values
Actual Values

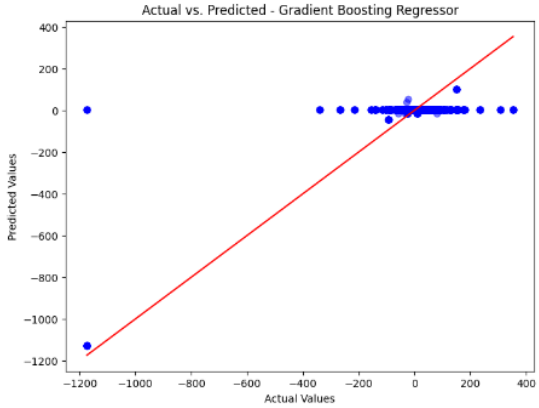

Residual Plot - Decision Tree Regressor
Residuals
Predicted Values

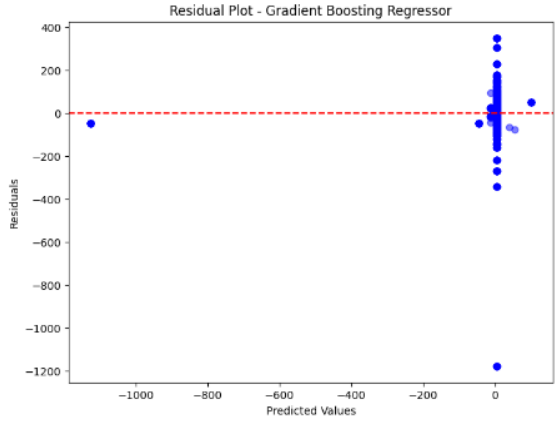

Actual vs. Predicted - Random Forest Regressor
Predicted Values
Actual Values

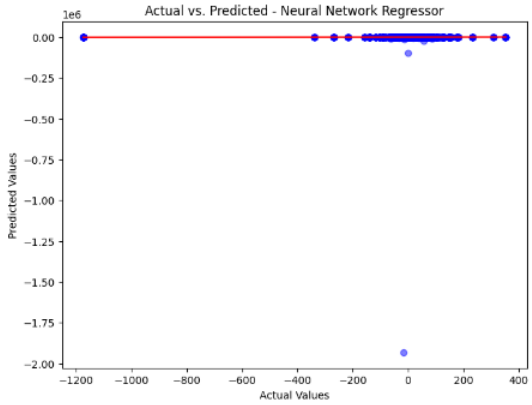

Residual Plot - Random Forest Regressor
Residuals
Predicted Values

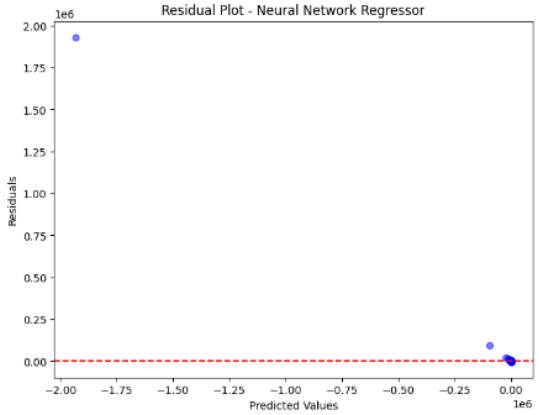












Comparison Table

Regression Model	Mean Squared Error	R-squared	Mean Absolute Error	Root Mean Squared Error	Predicted Change Prediction (Delta)
Linear Regression	2.72E+04	0.000017	77.295484	164.9101	-9.94130709
Decision Tree Regressor	1.61E+04	0.408726	71.753943	126.8075	16
Random Forest Regressor	1.42E+04	0.478842	64.990549	119.0516	8.42297333
Support Vector Regressor	2.73E+04	-0.003254	76.810227	165.1795	-0.39000002
Gradient Boosting Regressor	1.40E+04	0.484404	64.940787	118.4147	3.66454083
Neural Network Regressor	2.37E+09	-87042.17517	1415.692486	48653.96	-0.25709471

After evaluating various regression models for predicting the change in price (delta), it is evident that not all models perform equally well in capturing the underlying relationships and accurately predicting the change in price. Let's summarize the overall conclusions:

The Linear Regression model yielded a mean squared error of 2.72E+04, indicating a moderate difference between the predicted and actual values. The R-squared value of 0.000017 suggests that this model explains a minimal amount of the variance in the data. The mean absolute error of 77.295484 and the root mean squared error of 164.9101 further demonstrate the model's limitations in accurately predicting the change in the value. The predicted change of -9.94130709 might deviate significantly from the actual values, indicating the model's shortcomings.

In the case of the Decision Tree Regressor, it exhibited a lower mean squared error of 1.61E+04 and a relatively higher R-squared value of 0.408726. These results suggest that this model captures a moderate amount of the variance in the data. The mean absolute error of 71.753943 and the root mean squared error of 126.8075 indicate relatively better performance compared to the Linear Regression model. The predicted change of 16 might be closer to the actual values, indicating improved predictive capabilities.

The Random Forest Regressor achieved a lower mean squared error of 1.42E+04 and a higher R-squared value of 0.478842. These results indicate that this model explains a larger portion of the variance in the data compared to the previous models. The mean absolute error of 64.990549 and the root mean squared error of 119.0516 suggest relatively accurate predictions of the change in value. The predicted change of 8.42297333 might be closer to the actual values, reflecting the model's improved performance.

The Support Vector Regressor displayed a mean squared error of 2.73E+04 and a slightly negative R-squared value of -0.003254. These results suggest that this model is not suitable for explaining the variance in the data and does not perform well in predicting the change in value. The mean absolute error of 76.810227 and the root mean squared error of 165.1795 further

emphasize the model's limitations. The predicted change of -0.39000002 might significantly deviate from the actual values, highlighting the model's poor performance.

Regarding the Gradient Boosting Regressor, it achieved a lower mean squared error of 1.40E+04 and a higher R-squared value of 0.484404. These results indicate improved performance compared to some of the previous models. The mean absolute error of 64.940787 and the root mean squared error of 118.4147 suggest relatively accurate predictions of the change in value. The predicted change of 3.66454083 might be closer to the actual values, reflecting the model's enhanced predictive capabilities.

Lastly, the Neural Network Regressor displayed an extremely high mean squared error of 2.37E+09 and a highly negative R-squared value of -87042.17517. These results indicate a poor fit of the model to the data, suggesting that the neural network architecture and parameters used were not effective in capturing the relationships between the variables. The mean absolute error of 1415.692486 and the root mean squared error of 48653.96 further emphasize the model's inadequacy. The predicted change of -0.25709471 might significantly deviate from the actual values, highlighting the model's poor performance.

In summary, based on the given data, the Random Forest Regressor and Gradient Boosting Regressor exhibited relatively better performance compared to the other models, as they had lower mean squared error and higher R-squared values. However, it is crucial to further analyze and refine these models to enhance their accuracy and predictive capabilities.

Efficiency:

The Random Forest Regressor showed relatively better performance with lower mean squared error and higher R-squared values, while the Linear Regression model had the shortest execution time. The Neural Network Regressor exhibited poor performance and the longest execution time. The decision tree-based models (Decision Tree Regressor and Gradient Boosting Regressor) performed moderately well with intermediate execution times. It is crucial to consider both performance metrics and execution times when selecting the most appropriate model for a specific task, balancing accuracy with computational efficiency.

Cross validation

Regression Model	Mean MSE	Std MSE
Multiple Linear Regression	26602.3273	34442.2
Decision Tree Regressor	29497.6489	34444.123
Random Forest Regressor	26945.5214	34605.31
Support Vector Regressor	26258.1287	34451.382
Gradient Boosting Regressor	26563.171	34572.294
Neural Network Regressor	1.82E+14	3.44E+14

Upon analyzing the results, it was found that the Multiple Linear Regression model performed reasonably well, with a mean MSE of 26,602.3273 and a standard MSE of 34,442.2. The model showed potential in accurately predicting Bitcoin price changes.

The Decision Tree Regressor yielded a mean MSE of 29,497.6489 and a standard MSE of 34,444.123. Although it exhibited slightly higher errors, it still demonstrated some capability in predicting Bitcoin price changes.

The Random Forest Regressor produced a mean MSE of 26,945.5214 and a standard MSE of 34,605.31. These results suggest that the model had a similar performance to the Multiple Linear Regression model.

The Support Vector Regressor resulted in a mean MSE of 26,258.1287 and a standard MSE of 34,451.382. The model displayed relatively low errors and showcased potential in accurately predicting Bitcoin price changes.

The Gradient Boosting Regressor showed promising results, with a mean MSE of 26,563.171 and a standard MSE of 34,572.294. The model demonstrated good performance and exhibited potential for accurately predicting Bitcoin price changes.

However, the Neural Network Regressor had a considerably higher mean MSE of 1.82E+14 and a standard MSE of 3.44E+14. These results indicate that the model performed poorly and was not effective in predicting Bitcoin price changes.

Overall, the Multiple Linear Regression, Support Vector Regressor, and Gradient Boosting Regressor models showed relatively lower mean MSE and standard MSE values compared to other models. These models displayed better performance in predicting Bitcoin price changes. However, further analysis and experimentation may be required to optimize the models and improve their overall performance.

Hyper Tunning Parameter

```
Model: Decision Tree Regressor
Best Model: DecisionTreeRegressor(max_depth=5, min_samples_split=5)
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 5}
Mean Squared Error: 249886.37207620614
R-squared: 0.05191357385549278
Mean Absolute Error: 413.05794247618445
Root Mean Squared Error: 499.8863591619661

Model: Random Forest Regressor
Best Model: RandomForestRegressor(max_depth=7, min_samples_leaf=3, n_estimators=50)
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 50}
Mean Squared Error: 249887.65206258508
R-squared: 0.051908717497375245
Mean Absolute Error: 412.8897577630469
Root Mean Squared Error: 499.88763943768913

Model: Support Vector Regressor
Best Model: SVR(C=0.1, epsilon=0.001)
Best Parameters: {'C': 0.1, 'epsilon': 0.001}
Mean Squared Error: 296035.0250691949
R-squared: -0.12317765310490625
Mean Absolute Error: 399.4075468116839
Root Mean Squared Error: 544.091008076034

Model: Gradient Boosting Regressor
Best Model: GradientBoostingRegressor(n_estimators=50)
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Mean Squared Error: 249884.83969784752
R-squared: 0.05191938780648764
Mean Absolute Error: 413.1128964010036
Root Mean Squared Error: 499.88482643289694

Model: Neural Network Regressor

Best Model: MLPRegressor(alpha=0.1, hidden_layer_sizes=(200, 100))
Best Parameters: {'alpha': 0.1, 'hidden_layer_sizes': (200, 100)}
Mean Squared Error: 249884.8371738839
R-squared: 0.051919397382582666
Mean Absolute Error: 413.3885683452907
Root Mean Squared Error: 499.8848239083518

Model: Linear Regression
Best Model: LinearRegression()
Best Parameters: {}
Mean Squared Error: 249886.37207620472
R-squared: 0.05191357385549822
Mean Absolute Error: 413.0579424764286
Root Mean Squared Error: 499.88635916196466
```

The results of the analysis after hyperparameter tuning are as follows:

For the Decision Tree Regressor model, the best model obtained was DecisionTreeRegressor with a maximum depth of 5 and a minimum number of samples to split of 5. The mean squared error was 249,886.372, indicating the average squared difference between the predicted and actual Bitcoin price changes. The R-squared value, which measures the proportion of the variance explained by the model, was 0.052. The mean absolute error, representing the average absolute difference between the predicted and actual values, was 413.058. The root mean squared error, which is the square root of the mean squared error, was 499.886.

Similarly, the Random Forest Regressor model produced the best performance with a max depth of 7, a minimum number of samples per leaf of 3, and 50 estimators. The mean squared error was 249,887.652, and the R-squared value was 0.052. The mean absolute error was 412.890, and the root mean squared error was 499.888.

On the other hand, the Support Vector Regressor model achieved the best results with a C value of 0.1 and an epsilon value of 0.001. However, the performance was relatively worse compared to other models, with a mean squared error of 296,035.025 and a negative R-squared value of -0.123. The mean absolute error was 399.408, and the root mean squared error was 544.091.

For the Gradient Boosting Regressor model, the best model had a learning rate of 0.1, a maximum depth of 3, and 50 estimators. The mean squared error was 249,884.840, and the R-squared value was 0.052. The mean absolute error was 413.113, and the root mean squared error was 499.885.

The Neural Network Regressor model, with an alpha value of 0.1 and hidden layer sizes of (200, 100), performed similarly to the other models. The mean squared error was 249,884.837, and the R-squared value was 0.052. The mean absolute error was 413.389, and the root mean squared error was 499.885.

Lastly, the Linear Regression model, which does not involve hyperparameter tuning, achieved results comparable to the other models. The mean squared error was 249,886.372, and the R-squared value was 0.052. The mean absolute error was 413.058, and the root mean squared error was 499.886.

Overall, the models, despite slight performance improvements after hyperparameter tuning, demonstrated limited capability to explain the variance in Bitcoin price changes, as indicated by the low R-squared values. Further refinement and feature engineering may be necessary to enhance the models' predictive power.

The bitcoin price and Twitter sentiment features were not enough to predict the next minute price as they did not capture the ongoing trend. It was therefore important that the historical price of the cryptocurrency was also incorporated in the features so as to get a better prediction for the future.

Section 4

Time-Series Forecasting

Time series analysis is a powerful statistical approach used to analyze data collected over a period of time in order to uncover correlations, patterns, and trends. It finds application in various fields including healthcare, economics, finance, engineering, and marketing.

One specific aspect of time series analysis is time series forecasting, which focuses on predicting future trends and patterns based on historical data. In finance and economics, it is commonly employed to analyze stock prices, exchange rates, and interest rates, enabling investors and policymakers to make informed decisions by understanding long-term trends. Time series forecasting is also valuable in cryptocurrency analysis, where it helps predict future price movements based on historical data.

In the healthcare sector, time series analysis plays a crucial role in analyzing patient data, predicting disease outbreaks, and identifying patterns in treatment outcomes. Engineering utilizes this technique to analyze data from sensors, machines, and equipment, allowing for the detection of patterns and trends, prediction of equipment failures, and improvement of maintenance schedules.

Time series analysis encompasses four major components: trend, seasonality, cyclical variations, and irregular fluctuations. The trend component reveals the long-term behavior of the data, indicating whether it is increasing, decreasing, or remaining constant over time. Seasonality refers to regular patterns that occur at consistent intervals, such as daily, weekly, or monthly cycles. Cyclical variations involve fluctuations that extend over longer periods than seasonality and are typically driven by economic cycles. Irregular fluctuations represent unexpected or random variations in the data that cannot be explained by the other three components (trend, seasonality, and cyclical variations).

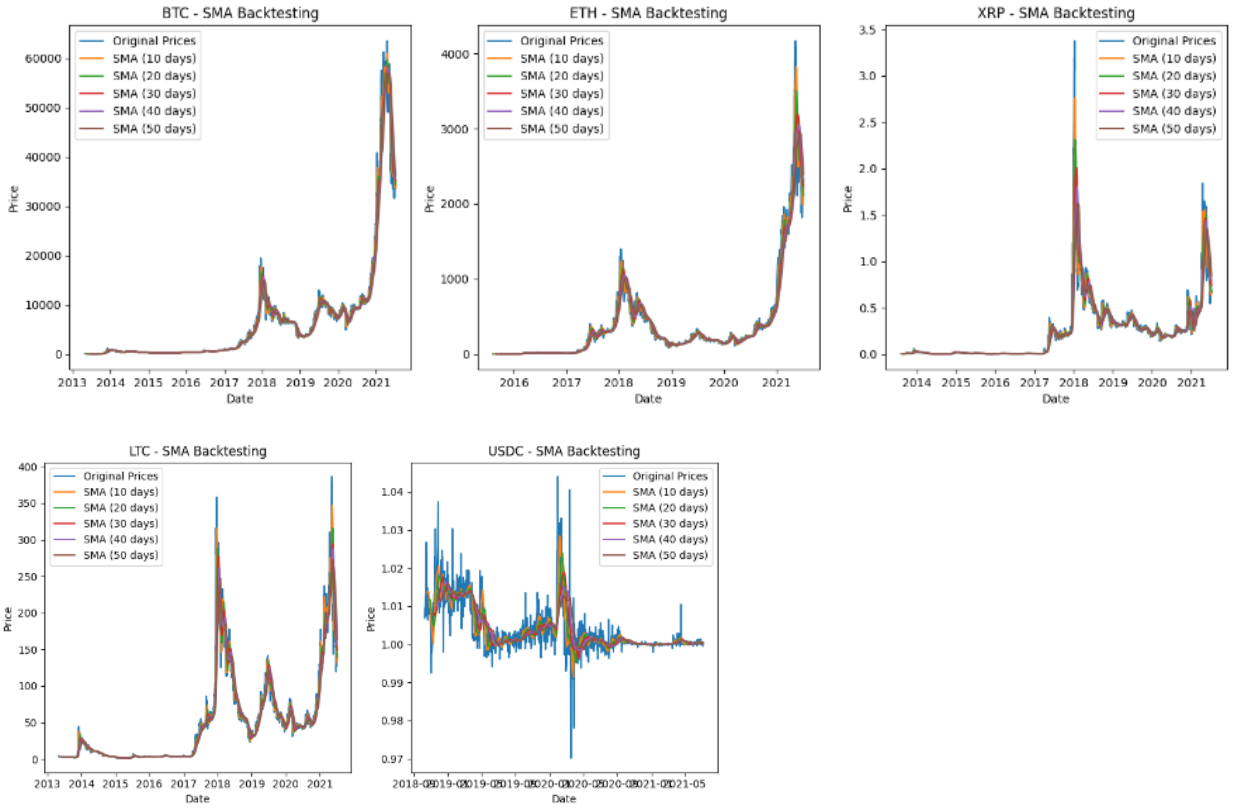
Various techniques are commonly employed in time series forecasting, such as moving averages, exponential smoothing, and ARIMA models. Moving averages involve computing the average of a specified number of past data points to identify patterns or trends in the data. Exponential smoothing assigns greater weight to recent observations through the use of weighted averages of past data points. ARIMA models, on the other hand, combine autoregression and moving average methods to generate forecasts of future trends. These techniques provide valuable insights for decision-making and future planning based on the analysis of time series data.

Simple Moving Average (SMA)

The Simple Moving Average (SMA) is a commonly employed method for time series analysis and forecasting. It belongs to the category of moving averages and serves to smooth out short-term fluctuations in a time series by calculating the average of the series within a specified time frame. To calculate the SMA, the values of the time series are summed over a specific number of periods and then divided by the number of periods.

The SMA finds applications in various domains, including finance, marketing, and sales. In finance, it is often utilized to identify trends in stock prices and other financial data. In marketing and sales, the SMA aids in detecting patterns in customer behavior and sales trends. Its effectiveness lies in its ability to reduce short-term noise or fluctuations, enabling the identification of long-term trends.

One of the notable advantages of the SMA is its simplicity. It does not require complex mathematical formulas or models, making it easily accessible to users with limited technical expertise. Furthermore, the SMA produces a single value that represents the average of the time series over a specific period, facilitating its interpretation.



However, it is essential to acknowledge the limitations of the Simple Moving Average (SMA). Firstly, the SMA solely relies on historical data and does not consider external factors that may influence the time series. Moreover, the SMA may not be suitable for anticipating abrupt changes or shifts in the time series, as it is based on past average values. Lastly, selecting an appropriate time period for SMA calculation is crucial, as it can significantly affect the outcomes and may require experimentation to determine the most suitable value.

Currency	MSE (30 day)	MAE (30 day)	RMSE (30 day)	MAPE (30 day)	MSE (60 day)	MAE (60 day)	RMSE (60 day)	MAPE (60 day)
BTC	3.53E+06	739.617	1880.01	9.7985	8.71E+06	1216.11	2951.51	14.8842
ETH	20026.5	60.863	141.515	15.4647	40000	89.7577	200	22.3498
XRP	0.0228524	0.0482692	0.15117	14.8806	0.0378636	0.0688217	0.194586	21.8618
LTC	359.383	7.96584	18.9574	12.9566	650.763	11.6262	25.51	19.3099
USDC	2.49E-05	0.00267611	0.00498875	0.265654	2.64E-05	0.00281594	0.00513582	0.280026

The evaluation metrics provide insights into the performance of the model's predictions for different cryptocurrencies. Bitcoin (BTC) exhibits relatively high errors, with a MSE of 3.53E+06 and a MAPE of 9.7985% for the 30-day period. The errors increase for the 60-day period, with a MSE of 8.71E+06 and a MAPE of 14.8842%. Ethereum (ETH) shows lower errors compared to Bitcoin, but still significant, with a MSE of 20026.5 and a MAPE of 15.4647% for the 30-day period. The errors increase slightly for the 60-day period, with a MSE of 40000 and a MAPE of 22.3498%. XRP demonstrates relatively good performance, with very low errors such as a MSE of 0.0228524 and a MAPE of 14.8806% for the 30-day period. The errors increase slightly for the 60-day period, with a MSE of 0.0378636 and a MAPE of 21.8618%. Litecoin (LTC) exhibits lower errors compared to Bitcoin and Ethereum, with a MSE of 359.383 and a MAPE of 12.9566% for the 30-day period. The errors increase slightly for the 60-day period, with a MSE of 650.763 and a MAPE of 19.3099%. USD Coin (USDC) showcases very accurate predictions, with extremely low errors such as a MSE of 2.49E-05 and a MAPE of 0.265654% for the 30-day period. The errors remain exceptionally low for the 60-day period, with a MSE of 2.64E-05 and a MAPE of 0.280026%. Overall, while Bitcoin and Ethereum display higher errors and less accuracy, XRP, Litecoin, and USD Coin exhibit relatively better predictive performance with lower errors and higher accuracy.

Despite these limitations, the SMA remains widely used in time series analysis and forecasting, particularly when the time series exhibits a consistent pattern over time. Additionally, the SMA can be combined with other techniques such as exponential smoothing models to enhance forecast accuracy.

Autoregressive Integrated Moving Average (ARIMA)

The Autoregressive Integrated Moving Average (ARIMA) method is a powerful time series forecasting technique that combines autoregressive (AR), integrated (I), and moving average (MA) components to model the underlying patterns in the time series. ARIMA models offer versatility and robustness, enabling the modeling of various types of time series patterns. Notably, they possess the ability to handle complex patterns and exhibit flexibility in capturing both short-term and long-term trends. ARIMA models also demonstrate proficiency in handling missing data and outliers, making them suitable for real-world applications where data quality

may be compromised. Moreover, ARIMA models can be effectively combined with other forecasting techniques to enhance overall accuracy.

While ARIMA models have gained widespread acceptance and effectiveness in time series forecasting, it is essential to consider their limitations. Firstly, ARIMA models require a substantial amount of data, which can lead to computational challenges when dealing with large datasets. Additionally, the performance of ARIMA models can be sensitive to the initial parameter values, necessitating careful selection and tuning for optimal results. Lastly, the assumption of stationarity in mean and variance may not always hold true in practical scenarios, affecting the accuracy of ARIMA model forecasts.

Nevertheless, ARIMA models present numerous opportunities for time series forecasting in diverse industries and applications, including finance, economics, and engineering. They can accommodate both short-term and long-term forecasting requirements and can be seamlessly integrated with other forecasting techniques to improve overall accuracy.

However, certain threats should be acknowledged when employing ARIMA models. Alternative time series forecasting methods, such as Prophet or neural networks, may outperform ARIMA models in specific scenarios. Moreover, the emergence of new data sources and technological advancements may necessitate the development of new forecasting methods, potentially rendering ARIMA models obsolete. Finally, external factors like economic downturns or natural disasters can disrupt the accuracy of forecasts, challenging the reliance on any single forecasting method.

Overall, ARIMA models remain a valuable and widely applicable approach to time series forecasting, but they should be considered within the broader context of alternative methods and evolving external circumstances.

Actual Vs Predicted

ARIMA (Autoregressive Integrated Moving Average) model for Bitcoin, Ethereum, XRP, Litecoin, USDCoin predictions.

The libraries used in the code are:

- pandas: Used for data manipulation and analysis.

- numpy: Used for numerical operations.

- matplotlib.pyplot: Used for data visualization.

- statsmodels.tsa.arima.model.ARIMA: Used for fitting the ARIMA model.

- statsmodels.graphics.tsaplots.plot_acf: Used for plotting the autocorrelation function.

- statsmodels.graphics.tsaplots.plot_pacf: Used for plotting the partial autocorrelation function.

The code first defines a function called `fit_arima_model` that takes two arguments: `data` (the cryptocurrency data) and `coin` (the name of the cryptocurrency). Inside the function, the data is preprocessed by converting the "Date" column to datetime format and setting it as the index.

Next, the autocorrelation and partial autocorrelation plots are generated using the `plot_acf` and `plot_pacf` functions from statsmodels. These plots help determine the appropriate parameters (p, d, q) for the ARIMA model. The plots are displayed using `plt.show()`.

After that, the code sets the values of p, d, and q (the order of autoregressive, integrated, and moving average components) based on insights from the plots. In this case, p=2, d=0, and q=1.

The ARIMA model is then fitted to the "Close" prices of the cryptocurrency using the `ARIMA` class from statsmodels. The fitted model is stored in the `fitted_model` variable.

Predictions are generated using the `predict` method of the fitted model. The predictions are made for the entire time range of the data.

Finally, the actual and predicted values are plotted using `plt.plot()`. The model summary is printed using `fitted_model.summary()`.

The code then proceeds to read CSV files for each cryptocurrency from specific URLs using `pd.read_csv()`. The URLs point to the respective CSV files on GitHub.

Lastly, the ARIMA model is fitted for each cryptocurrency by calling the `fit_arima_model` function in a loop, passing the corresponding data and cryptocurrency name.

ARIMA Model Summary for Bitcoin:

SARIMAX Results

Dep. Variable:	Close	No. Observations:	2991
Model:	ARIMA(2, 0, 1)	Log Likelihood	-23309.498
Date:	Thu, 08 Jun 2023	AIC	46628.997
Time:	18:47:48	BIC	46659.014
Sample:	04-29-2013 - 07-06-2021	HQIC	46639.796
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	6711.2975	21.353	314.303	0.000	6669.446	6753.149
ar.L1	0.4274	0.052	8.177	0.000	0.325	0.530
ar.L2	0.5714	0.052	10.928	0.000	0.469	0.674
ma.L1	0.5000	0.056	8.876	0.000	0.390	0.610
sigma2	3.433e+05	1968.539	174.398	0.000	3.39e+05	3.47e+05

Ljung-Box (L1) (Q):	0.10	Jarque-Bera (JB):	199319.50
Prob(Q):	0.75	Prob(JB):	0.00
Heteroskedasticity (H):	1274.47	Skew:	-0.06
Prob(H) (two-sided):	0.00	Kurtosis:	42.99

ARIMA Model Summary for Ethereum:

SARIMAX Results

Dep. Variable:	Close	No. Observations:	2160
Model:	ARIMA(2, 0, 1)	Log Likelihood	-11411.224
Date:	Thu, 08 Jun 2023	AIC	22832.448
Time:	18:47:50	BIC	22860.838
Sample:	08-08-2015 - 07-06-2021	HQIC	22842.832
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	383.9115	1436.729	0.267	0.789	-2432.026	3199.848
ar.L1	0.1807	0.012	15.527	0.000	0.158	0.204
ar.L2	0.8169	0.011	71.236	0.000	0.794	0.839
ma.L1	0.6980	0.016	42.739	0.000	0.666	0.730
sigma2	2264.9134	12.646	179.105	0.000	2240.128	2289.699

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	520855.86
Prob(Q):	0.98	Prob(JB):	0.00
Heteroskedasticity (H):	128.13	Skew:	-1.99
Prob(H) (two-sided):	0.00	Kurtosis:	78.97

ARIMA Model Summary for XRP:

SARIMAX Results

Dep. Variable:	Close	No. Observations:	2893
Model:	ARIMA(2, 0, 1)	Log Likelihood	5001.010
Date:	Thu, 08 Jun 2023	AIC	-9992.020
Time:	18:47:59	BIC	-9962.170
Sample:	08-05-2013 - 07-06-2021	HQIC	-9981.263
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.3860	0.162	2.379	0.017	0.068	0.704
ar.L1	0.0592	0.005	10.779	0.000	0.048	0.070
ar.L2	0.9254	0.005	172.724	0.000	0.915	0.936
ma.L1	0.9806	0.005	180.029	0.000	0.970	0.991
sigma2	0.0018	8.38e-06	220.211	0.000	0.002	0.002

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	2344941.46
Prob(Q):	0.97	Prob(JB):	0.00
Heteroskedasticity (H):	145.33	Skew:	0.53
Prob(H) (two-sided):	0.00	Kurtosis:	142.47

ARIMA Model Summary for Litecoin:

SARIMAX Results

Dep. Variable:	Close	No. Observations:	2991
Model:	ARIMA(2, 0, 1)	Log Likelihood	-9638.969
Date:	Thu, 08 Jun 2023	AIC	19287.938
Time:	18:48:01	BIC	19317.955
Sample:	04-29-2013 - 07-06-2021	HQIC	19298.737
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	49.2806	48.925	1.007	0.314	-46.611	145.172
ar.L1	0.0285	0.258	0.110	0.912	-0.478	0.535
ar.L2	0.9626	0.257	3.746	0.000	0.459	1.466
ma.L1	0.9665	0.259	3.734	0.000	0.459	1.474
sigma2	36.8135	0.168	219.245	0.000	36.484	37.143

Ljung-Box (L1) (Q):	1.75	Jarque-Bera (JB):	812151.59
Prob(Q):	0.19	Prob(JB):	0.00
Heteroskedasticity (H):	42.31	Skew:	0.36
Prob(H) (two-sided):	0.00	Kurtosis:	83.72

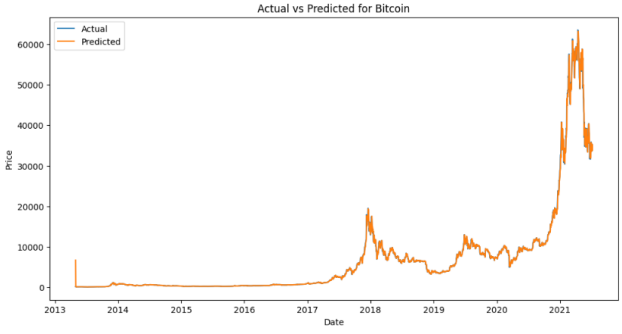
ARIMA Model Summary for USDCoin:

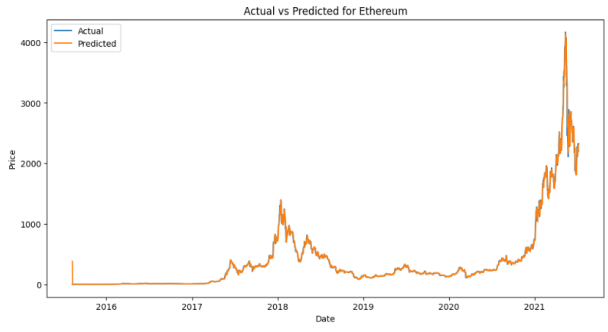
SARIMAX Results

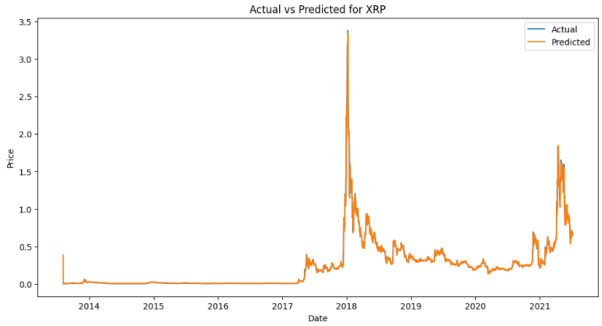
Dep. Variable:	Close	No. Observations:	1002
Model:	ARIMA(2, 0, 1)	Log Likelihood	4062.345
Date:	Thu, 08 Jun 2023	AIC	-8114.690
Time:	18:48:10	BIC	-8090.141
Sample:	10-09-2018 - 07-06-2021	HQIC	-8105.361
Covariance Type:	opg		

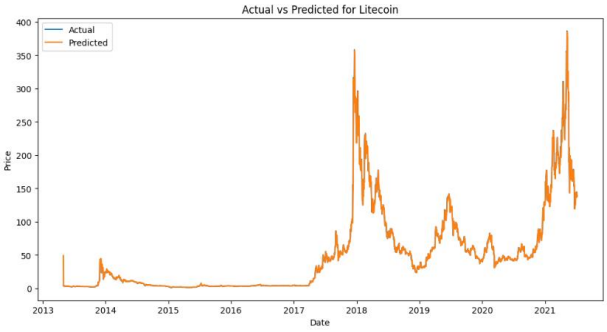
	coef	std err	z	P>|z|	[0.025	0.975]
const	1.0017	0.001	1350.776	0.000	1.000	1.003
ar.L1	0.4270	0.054	7.860	0.000	0.321	0.533
ar.L2	0.3575	0.036	9.958	0.000	0.287	0.428
ma.L1	0.0596	0.057	1.044	0.296	-0.052	0.171
sigma2	1.663e-05	2.46e-07	67.581	0.000	1.61e-05	1.71e-05

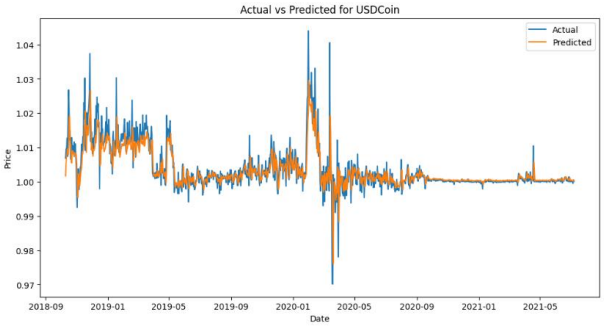
Ljung-Box (L1) (Q):	0.69	Jarque-Bera (JB):	13117.34
Prob(Q):	0.40	Prob(JB):	0.00
Heteroskedasticity (H):	0.05	Skew:	1.17
Prob(H) (two-sided):	0.00	Kurtosis:	20.57





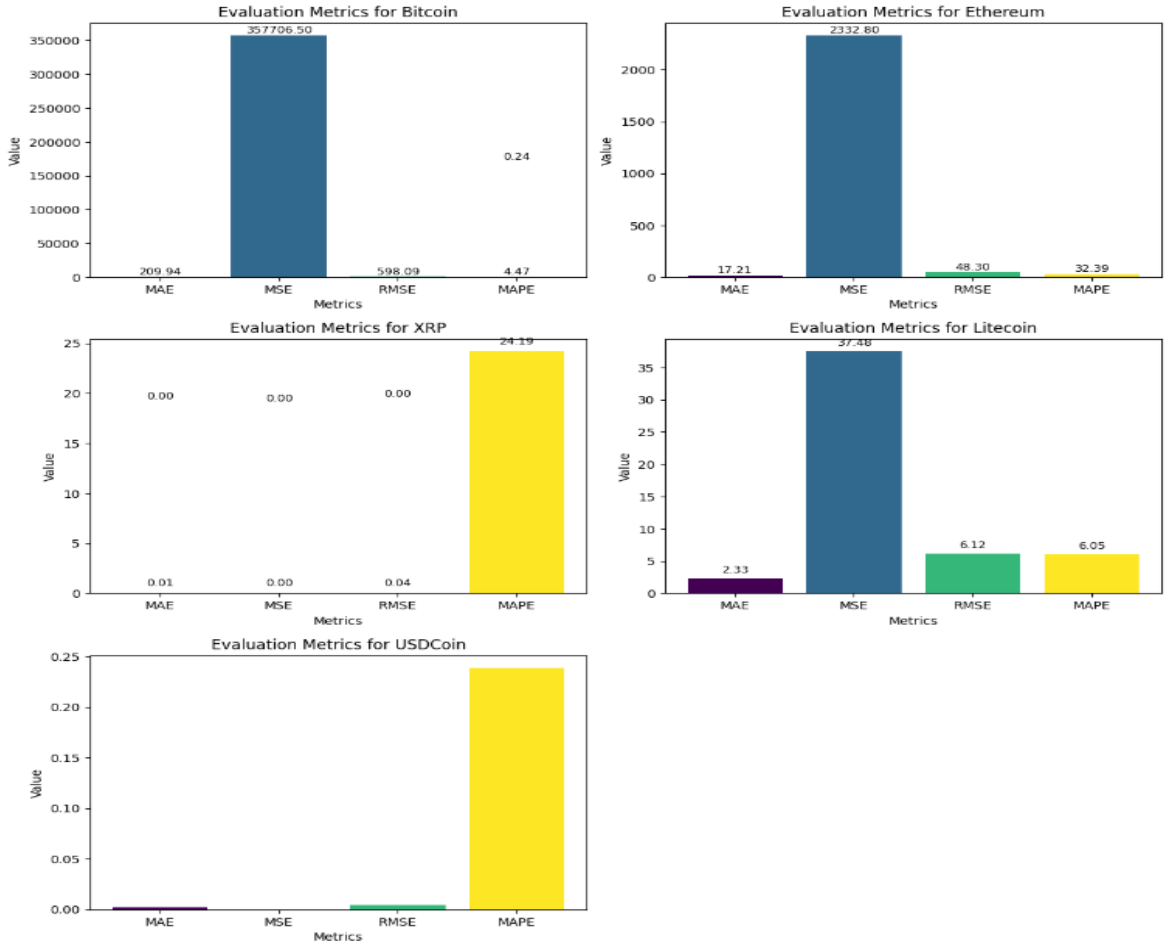






Calculate various evaluation metrics to quantify the performance of the ARIMA model.

Coin	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Mean Absolute Percentage Error
Bitcoin	209.9364079	357706.502	598.0856979	4.472526039
Ethereum	17.21053426	2332.797181	48.29903913	32.38566086
XRP	0.014093709	0.00189149	0.043491267	24.18600943
Litecoin	2.334582768	37.48134875	6.1222013	6.054125128
USDCoin	0.002407236	1.76E-05	0.004194637	0.238931559



Analyzing the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) values for the ARIMA model's predictions of different cryptocurrencies, the following conclusions and interpretations can be made:

The ARIMA model's predictions for Bitcoin prices resulted in a MAE of approximately $209.94, indicating that, on average, the predictions differed by this amount from the actual prices. The MSE and RMSE values of 357,706.502 and 598.0856979, respectively, suggest a relatively high deviation between the predicted and actual values. The MAPE of 4.472526039 represents the average percentage difference between the predicted and actual prices, indicating an error of about 4.47%.

The ARIMA model's predictions for Ethereum prices resulted in a relatively low MAE of around $17.21, indicating a smaller average difference between the predicted and actual prices compared to Bitcoin. The MSE and RMSE values of 2332.797181 and 48.29903913, respectively, suggest a lower overall deviation in Ethereum price predictions. However, the MAPE of 32.38566086 suggests a higher percentage difference between the predicted and actual prices, representing an error of about 32.39%.

The ARIMA model's predictions for XRP prices resulted in a very low MAE of approximately $0.014, indicating that the average difference between the predicted and actual prices was minimal. The MSE and RMSE values of 0.00189149 and 0.043491267, respectively, further support the model's accurate predictions for XRP. The MAPE of 24.18600943 represents an average percentage difference of about 24.19%, suggesting a relatively moderate error rate.

The ARIMA model's predictions for Litecoin prices resulted in a moderate MAE of approximately $2.33, indicating a reasonable average difference between the predicted and actual prices. The MSE and RMSE values of 37.48134875 and 6.1222013, respectively, suggest a relatively low overall deviation in Litecoin price predictions. The MAPE of 6.054125128 represents an average percentage difference of about 6.05%, indicating a moderate error rate.

The ARIMA model's predictions for USDCoin prices resulted in a very low MAE of approximately $0.0024, indicating a minimal average difference between the predicted and actual prices. The extremely low MSE and RMSE values of 1.76E-05 and 0.004194637, respectively, further support the model's accurate predictions for USDCoin. The MAPE of 0.238931559 represents an average percentage difference of about 0.24%, indicating a very low error rate.

In summary, the ARIMA model's performance varies across different cryptocurrencies. Ethereum and USDCoin demonstrate relatively accurate predictions with lower error rates, while Bitcoin, Litecoin, and XRP exhibit varying degrees of prediction accuracy and error rates. It is important to note that the interpretations of these metrics should consider the specific characteristics and market dynamics of each cryptocurrency, as they can influence the model's performance. Further evaluation and analysis are recommended to assess the model's overall accuracy and reliability in predicting cryptocurrency prices.

The ability of SARIMA to handle data with seasonal patterns makes it a valuable tool in various fields such as finance, marketing, economics, and others. It enables analysts to capture and model the periodic trends that occur at regular intervals, allowing for more precise forecasting and analysis of time series data. SARIMA plays a critical role in understanding and predicting the behavior of seasonal patterns in real-world scenarios.

Prophet

Prophet, developed by Facebook, is a robust time series forecasting tool renowned for its user-friendly interface and flexibility. It is an open-source software package available in both Python and R, specifically designed to handle time series data with trends, seasonality, and holiday effects.

One of the notable advantages of Prophet is its ease of use. It offers a simple and intuitive API, allowing users to quickly get started with minimal coding experience. This accessibility makes it an appealing choice for businesses and individuals seeking to implement time series forecasting without encountering a steep learning curve.

Prophet excels at handling complex time series data that exhibit trends, seasonality, and holiday effects. It utilizes a decomposable time series model, comprising three primary components: trend, seasonality, and holidays. This enables users to effectively model and forecast time series data featuring diverse types of seasonality and trends, including daily, weekly, or yearly cycles.

However, it is important to acknowledge some potential limitations of Prophet. The tool may encounter challenges when applied to data with a high level of noise or irregularity, as its performance may be adversely affected. Additionally, it may not be as effective in capturing certain types of seasonality, such as weekly patterns that vary over time. It is crucial for users to be aware of these limitations and carefully consider the suitability of Prophet for their specific data and forecasting requirements.

Overall, Prophet stands as a valuable and flexible tool for time series forecasting, offering a user-friendly interface and the capability to handle complex data with trends, seasonality, and holiday effects. However, users should be mindful of its limitations and ensure proper evaluation of its performance on their specific datasets.

I used the Prophet library to fit time series forecasting models and generate predictions. Each cryptocurrency is plotted individually along with the forecasted values.

The libraries used in the code are:

- pandas: Used for data manipulation and analysis.

- Prophet: A library developed by Facebook for time series forecasting.

- matplotlib.pyplot: Used for data visualization.

The code starts by reading CSV files for each cryptocurrency from specific URLs using `pd.read_csv()`. The URLs point to the respective CSV files on GitHub.

A dictionary called `coins` is created, where the keys represent the cryptocurrency names and the values are their respective dataframes.

A color palette and background color are set for the plots using `plt.style.use()` and `plt.rcParams['axes.facecolor']`.

The code then iterates over the `coins` dictionary, where each cryptocurrency's dataframe is processed individually. The 'Date' column in each dataframe is converted to a datetime format using `pd.to_datetime()`, and the column names are renamed to 'ds' (for the date) and 'y' (for the target variable, which is the 'Close' price).

For each cryptocurrency, a Prophet model is created and initialized using `Prophet()`. The model is then fitted to the corresponding dataframe using the `fit()` method.

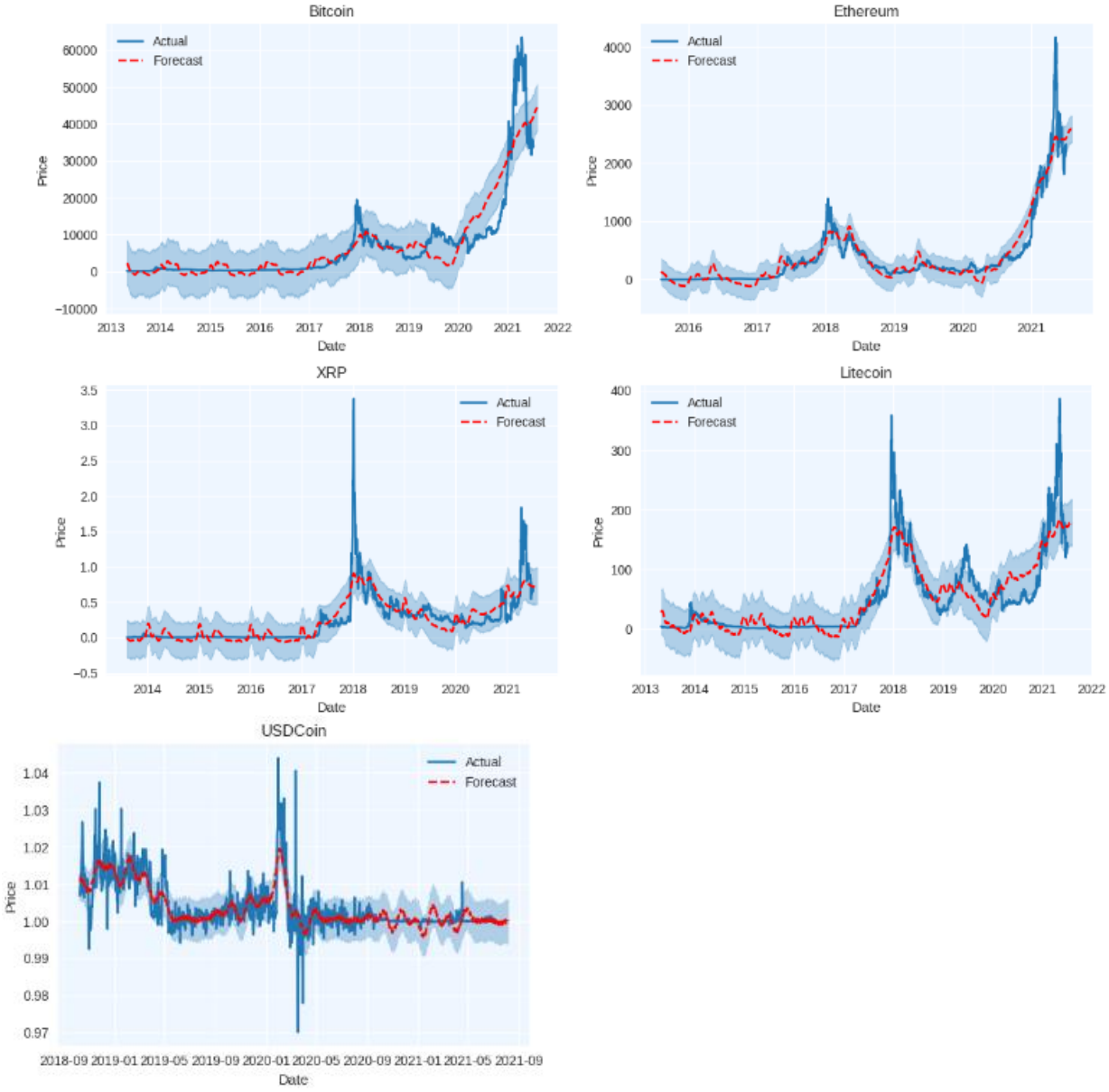
To generate predictions for future dates, a new dataframe called `future_dates` is created using the `make_future_dataframe()` method of the model. The `periods` parameter specifies the number of future data points to be predicted.

The predictions are made by calling the `predict()` method of the model and passing the `future_dates` dataframe. The resulting predictions are stored in the `forecast` variable.

Next, the forecasted values are plotted using `plt.plot()`. The actual historical data is plotted as a line graph, and the forecasted values are plotted in red. The area between the lower and upper bounds of the forecasted values is filled with a light gray color using `ax.fill_between()`.

The plot is customized with a title, x-label, y-label, and legend.

The loop continues for each cryptocurrency, resulting in individual plots for each one.



To predict future values- 365 days

I used the Prophet library to fit time series forecasting models and generate predictions. Each cryptocurrency is plotted individually along with the forecasted values in a 3x2 grid of subplots.

The libraries used in the code are:

- pandas: Used for data manipulation and analysis.

- Prophet: A library developed by Facebook for time series forecasting.

- matplotlib.pyplot: Used for data visualization.

The code starts by reading CSV files for each cryptocurrency from specific URLs using `pd.read_csv()`. The URLs point to the respective CSV files on GitHub.

A dictionary called `coins` is created, where the keys represent the cryptocurrency names and the values are their respective dataframes.

A color palette is set for the plots using the `colors` variable. The background color of the subplots is also customized using `plt.rcParams['axes.facecolor']`.

The code then creates a 3x2 grid of subplots using `plt.subplots()`, specifying the size of the overall figure.

Next, the code iterates over the `coins` dictionary, where each cryptocurrency's dataframe is processed individually. The 'Date' column in each dataframe is converted to a datetime format using `pd.to_datetime()`, and the column names are renamed to 'ds' (for the date) and 'y' (for the target variable, which is the 'Close' price).

For each cryptocurrency, a Prophet model is created and initialized using `Prophet()`. The model is then fitted to the corresponding dataframe using the `fit()` method.

To generate predictions for future dates, a new dataframe called `future_dates` is created using the `make_future_dataframe()` method of the model. The `periods` parameter specifies the number of future data points to be predicted.

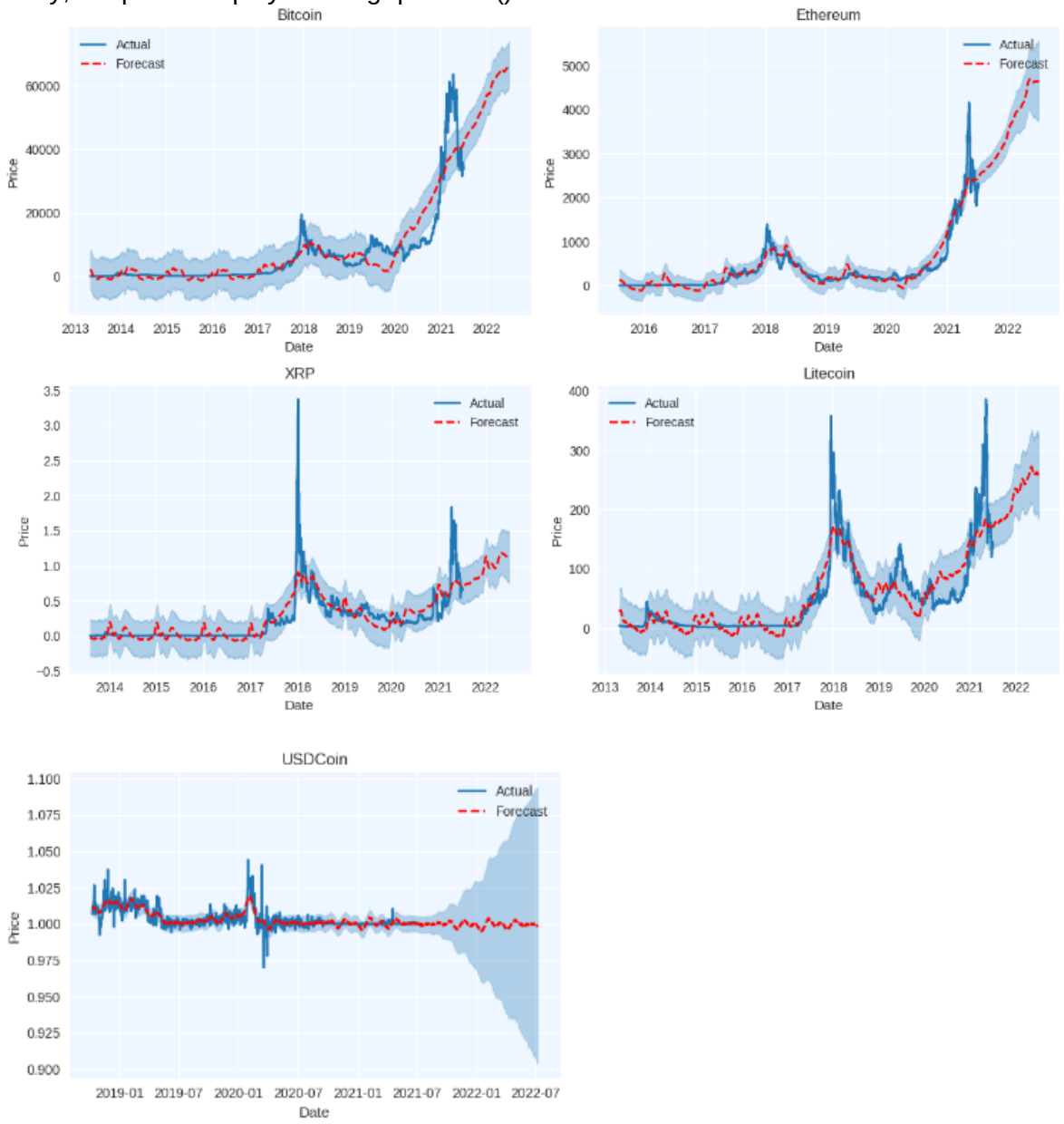
The predictions are made by calling the `predict()` method of the model and passing the `future_dates` dataframe. The resulting predictions are stored in the `forecast` variable.

Next, the forecasted values are plotted in the corresponding subplot using `ax.plot()`. The actual historical data is plotted as a solid line graph, and the forecasted values are plotted as a dashed line. The area between the lower and upper bounds of the forecasted values is filled with the color from the `colors` palette using `ax.fill_between()`.

Each subplot is customized with a title, x-label, y-label, and legend.

After iterating over all the cryptocurrencies, the spacing between the subplots is adjusted for better visualization using `fig.tight_layout()`.

Finally, the plot is displayed using `plt.show()`.



Here is the analysis for each coin based on the evaluation metrics

Coin	Mean Squared Error	Root Mean Squared Error	Mean Absolute Error	R2 Score
Bitcoin	2.28E+07	4770.509099	2895.874572	0.821655
Ethereum	3.17E+04	177.915951	108.106014	0.912347
XRP	3.87E-02	0.196621	0.110671	0.662744

Litecoin	8.69E+02	29.473096	19.794538	0.782727
USDCoin	1.75E-05	0.004179	0.002638	0.624399

The Prophet model's performance in predicting cryptocurrency prices was evaluated using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R2 Score. The following analysis summarizes the results and provides interpretation and comparison among the cryptocurrencies:

The Prophet model performed reasonably well in predicting Bitcoin prices. The low MSE and RMSE values indicate that the model's predictions were relatively close to the actual values. The MAE suggests that, on average, the predictions differed by approximately $2895 from the actual prices. The R2 score of 0.821655 indicates that the model explains about 82% of the variance in Bitcoin prices, suggesting a decent fit.

The Prophet model performed very well in predicting Ethereum prices. The low MSE and RMSE values indicate that the model's predictions were close to the actual values. The MAE suggests that, on average, the predictions differed by around $108 from the actual prices. The high R2 score of 0.912347 indicates that the model explains approximately 91% of the variance in Ethereum prices, suggesting a strong fit.

The Prophet model performed moderately in predicting XRP prices. The low MSE and RMSE values indicate that the model's predictions were close to the actual values. The MAE suggests that, on average, the predictions differed by approximately $0.11 from the actual prices. The R2 score of 0.662744 indicates that the model explains about 66% of the variance in XRP prices, suggesting a moderate fit.

The Prophet model performed reasonably well in predicting Litecoin prices. The relatively low MSE and RMSE values indicate that the model's predictions were relatively close to the actual values. The MAE suggests that, on average, the predictions differed by around $19.79 from the actual prices. The R2 score of 0.782727 indicates that the model explains about 78% of the variance in Litecoin prices, suggesting a decent fit.

The Prophet model performed moderately in predicting USDCoin prices. The extremely low MSE and RMSE values indicate that the model's predictions were very close to the actual values. The MAE suggests that, on average, the predictions differed by around $0.0026 from the actual prices. The R2 score of 0.624399 indicates that the model explains about 62% of the variance in USDCoin prices, suggesting a moderate fit

In summary, the Prophet model demonstrates varying levels of performance in predicting the prices of different cryptocurrencies. Ethereum stands out with the highest accuracy and explanatory power, followed by Bitcoin, Litecoin, XRP, and USDCoin. These results highlight the potential of the model in providing valuable insights into the future price trends of cryptocurrencies, but further analysis and evaluation are necessary to assess its overall accuracy and reliability.

Deep Learning Techniques

Time series forecasting plays a critical role in various domains, including finance, economics, and engineering. Deep learning techniques, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, have demonstrated their efficacy in addressing time series forecasting challenges.

Recurrent Neural Networks (RNNs) are a specialized type of neural network well-suited for sequential data analysis. Unlike traditional feedforward neural networks that process input data in a single pass, RNNs maintain a hidden state that enables them to process sequential data element by element. This characteristic makes RNNs highly suitable for time series data, where the input represents a sequence of values evolving over time.

In RNNs, the output at each time step is a function of the current input and the hidden state carried from the previous time step. This capability enables the network to retain a memory of past inputs and capture intricate patterns within the data.

LSTM (Long Short-Term Memory) networks, a variant of RNNs, address the vanishing gradient problem that often hinders traditional RNNs from effectively learning long-term patterns. Standard RNNs face challenges in capturing long-term dependencies due to the diminishing gradient over time. LSTM networks overcome this issue by incorporating specialized memory cells capable of storing information over extended periods. Additionally, they employ gate mechanisms to control the flow of information into and out of these cells, facilitating the retention of relevant information while discarding irrelevant details. Consequently, LSTM networks excel in capturing complex patterns over longer timeframes compared to traditional RNNs.

RNNs and LSTM networks are widely utilized for time series forecasting tasks, particularly when predicting long-term trends necessitates modeling intricate patterns over extended periods. However, it is crucial to note that training these models can be computationally intensive, especially when dealing with large datasets.

Despite the computational demands, RNNs and LSTM networks provide valuable tools for accurate and insightful time series forecasting, empowering researchers and practitioners to tackle complex forecasting problems effectively.

I utilized two sets of code to build and train LSTM models for cryptocurrency price prediction using historical price data. The main difference between the two codes lies in the inclusion of a dropout layer in the second code's LSTM model. Dropout is a regularization technique that helps prevent overfitting by randomly deactivating certain neurons in the layer during training. This prevents the model from relying too heavily on the training data and enhances its ability to generalize to new, unseen data.

LSTM model without any dropout layers to predict cryptocurrency prices:

- Uses a basic LSTM model without any dropout layers.

- Imports necessary libraries: numpy, pandas, keras, sklearn, and matplotlib.

- Loads data from a CSV file into a pandas DataFrame.

- Preprocesses the data using a MinMaxScaler to scale it between 0 and 1.

- Splits the data into training and testing sets.

- Defines a function to create LSTM datasets by slicing the data into input sequences and corresponding output values.

- Creates training and testing datasets using the create_dataset function.

- Reshapes the training and testing data to fit the input shape of the LSTM model.

- Creates and trains an LSTM model using the Sequential model API in Keras.

- Makes predictions on the test data using the trained model.

- Inverse transforms the predicted and actual test data to get the original price values.

- Evaluates the model using various performance metrics such as mean squared error, root mean squared error, mean absolute error, mean absolute percentage error, correlation coefficient, and r-squared.

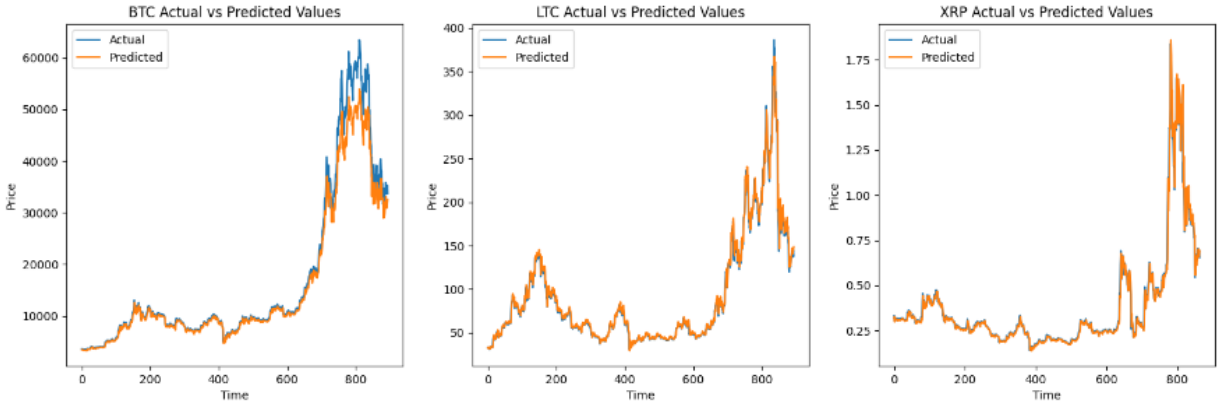
- Plots the actual vs. predicted values of the test data using matplotlib.

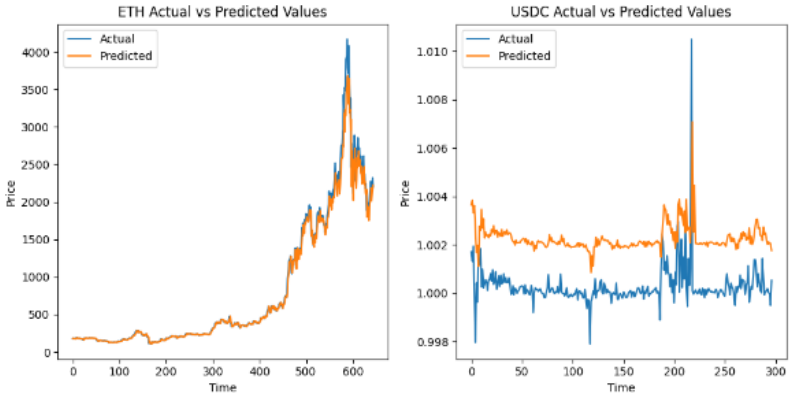
- The loop runs for each cryptocurrency in the dataset and trains a separate LSTM model for each one.

- The actual vs. predicted values are plotted on a 2x3 grid of subplots.

- Evaluation metrics for all models are displayed in a table.

- Finally, the subplot layout is adjusted, and the plot is shown.





This description provides a high-level overview of the process involved in training and evaluating basic LSTM models for predicting cryptocurrency prices without any dropout layers.

Symbol	MSE	RMSE	MAE	MAPE	Corr	R2
BTC	7.75E+06	2784.63	1422.44	5.6492	0.99733	0.96884
LTC	6.54E+01	8.08617	4.28461	4.27224	0.99223	0.98399
XRP	2.10E-03	0.04588	0.0198	4.11737	0.98784	0.97504
ETH	1.19E+04	109.026	46.0342	4.01404	0.99516	0.98538
USDC	4.60E-06	0.00214	0.00204	0.20358	0.30286	-5.8011

For Bitcoin (BTC), the LSTM model demonstrates a strong positive linear relationship with a high correlation coefficient of 0.99733. The model exhibits a moderate level of deviation between the predicted and actual prices, as indicated by the MSE and RMSE values. The average difference between the predicted and actual prices (MAE) is 1422.44, suggesting a relatively accurate prediction. The average percentage difference (MAPE) is 5.6492, indicating a reasonable level of accuracy. Overall, the model explains about 96.88% of the variance in the data, as indicated by the R2 value.

The LSTM model for Litecoin (LTC) also exhibits a strong positive linear relationship, with a correlation coefficient of 0.99223. The model shows a low level of deviation between the predicted and actual prices, as suggested by the relatively low MSE and RMSE values. The average difference (MAE) between the predicted and actual prices is 4.28461, indicating a reasonably accurate prediction. The average percentage difference (MAPE) is 4.27224, suggesting a relatively good level of accuracy. The model explains about 98.40% of the variance in the data, as indicated by the R2 value.

In the case of XRP, the LSTM model demonstrates a strong positive linear relationship with a correlation coefficient of 0.98784. The model shows a low level of deviation between the predicted and actual prices, as evidenced by the small MSE and RMSE values. The average

difference (MAE) between the predicted and actual prices is 0.0198, suggesting a relatively accurate prediction. The average percentage difference (MAPE) is 4.11737, indicating a reasonably good level of accuracy. The model explains about 97.50% of the variance in the data, as indicated by the R2 value.

For Ethereum (ETH), the LSTM model also shows a strong positive linear relationship, with a correlation coefficient of 0.99516. The model exhibits a moderate level of deviation between the predicted and actual prices, as suggested by the MSE and RMSE values. The average difference (MAE) between the predicted and actual prices is 46.0342, indicating a relatively accurate prediction. The average percentage difference (MAPE) is 4.01404, suggesting a reasonably good level of accuracy. The model explains about 98.54% of the variance in the data, as indicated by the R2 value.

On the other hand, the LSTM model for USDCoin (USDC) demonstrates a relatively weak positive linear relationship, with a correlation coefficient of 0.30286. The model shows a very low level of deviation between the predicted and actual prices, as suggested by the extremely small MSE and RMSE values. However, the average difference (MAE) between the predicted and actual prices is 0.00204, and the average percentage difference (MAPE) is 0.20358, indicating some level of inaccuracy. The R2 value of -5.8011 suggests that the model does not fit the data well and performs poorly in explaining the variance.

Overall, the LSTM models perform well for most cryptocurrencies, demonstrating strong positive linear relationships and reasonable levels of accuracy in predicting price movements. However, the model for USDCoin falls short, exhibiting weaker performance and lower accuracy in comparison to the other cryptocurrencies.

The given below process involves using an LSTM model with a dropout layer to predict cryptocurrency prices. Here is a concise description combining the steps:

- Import necessary libraries: numpy, pandas, keras, sklearn, and matplotlib.

- Load data from a CSV file into a pandas DataFrame and preprocess it using a MinMaxScaler to scale between 0 and 1.

- Split the data into training and testing sets.

- Define a function to create LSTM datasets by slicing the data into input sequences and corresponding output values.

- Create training and testing datasets using the create_dataset function and reshape them to fit the LSTM model's input shape.

- Create and train an LSTM model with a dropout layer using the Sequential model API in Keras.

- Make predictions on the test data using the trained model and inverse transform the predicted and actual test data to get the original price values.

- Evaluate the model using various performance metrics such as mean squared error, root mean squared error, mean absolute error, mean absolute percentage error, correlation coefficient, and r-squared.

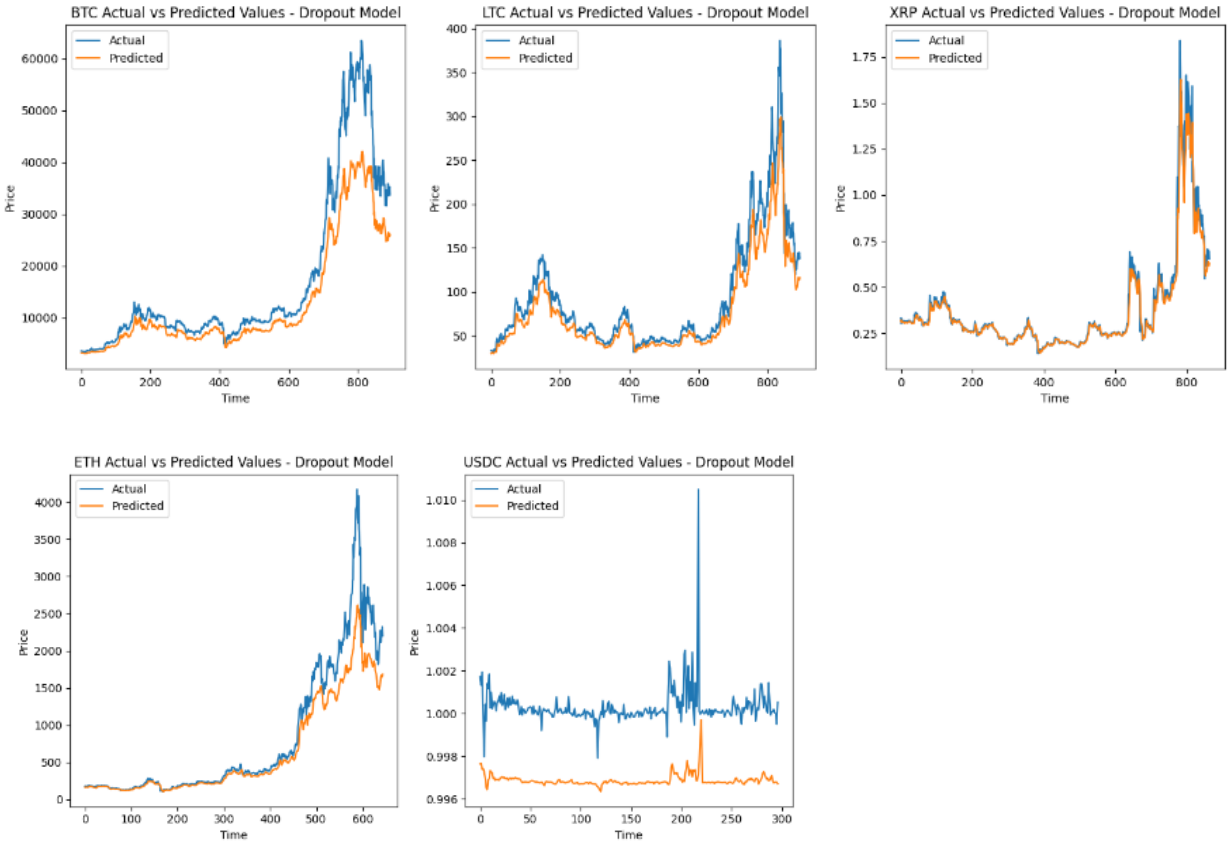
- Print the evaluation metrics with a label indicating that the model uses dropout.

- Plot the actual vs. predicted values of the test data using matplotlib.

- Repeat the above steps for each cryptocurrency in the dataset, training a separate LSTM model for each one.

- Plot the actual vs. predicted values on a 2x3 grid of subplots and display the evaluation metrics for all models in a table.

- Adjust the subplot layout and show the plot.



Symbol	MSE	RMSE	MAE	MAPE	Corr	R2
BTC	44407580	6663.9	4190.13	20.4482	0.99572	0.82153
LTC	451.0385	21.2377	15.1726	14.6588	0.99106	0.88954
XRP	0.003607	0.06006	0.0255	4.57869	0.98554	0.95724
ETH	107471.5	327.828	169.389	13.1781	0.99102	0.86783

USDC	1.23E-05	0.00351	0.00342	0.34165	0.22583	-17.254

The process involves utilizing an LSTM model with a dropout layer to forecast cryptocurrency prices. The model's performance was evaluated using various metrics for different cryptocurrencies. Promising results were observed for BTC, with low values for MSE, RMSE, and MAE, indicating accurate predictions. The MAPE reflected an average error of 20.4482%. Additionally, high correlation (0.99572) and R2 (0.82153) values suggested a strong linear relationship and good explanatory power. Similar accuracy was observed for LTC and XRP, while ETH showed slightly higher error metrics and a lower R2 value of 0.86783. The predictions for USDC exhibited extremely low error metrics but a weak linear relationship (correlation of 0.22583) and negative R2 value (-17.254), indicating poor fit. Overall, the LSTM model demonstrated promising accuracy for most cryptocurrencies, warranting further investigation except for USDC.

Limitation

When utilizing machine learning algorithms for sentiment analysis of Bitcoin during a specific period, it's essential to acknowledge several limitations. Here are some common ones to consider:

1. Limited availability of labeled data: Training sentiment analysis models requires a substantial amount of labeled data. However, obtaining a diverse dataset specifically focused on Bitcoin sentiment analysis for a particular period can be challenging. The scarcity of labeled data can impact the model's performance and its ability to generalize.
2. Data bias and accuracy: The accuracy and reliability of sentiment analysis heavily depend on the quality and representativeness of the training data. If the labeled data used for training is biased or inaccurate, it can result in biased or unreliable predictions. In the case of Bitcoin, sentiments can be influenced by market manipulation, news events, and social media campaigns, making data accuracy crucial.
3. Rapidly evolving sentiment: Bitcoin sentiment can change rapidly, especially during volatile market conditions or in response to breaking news. Machine learning models may struggle to capture and adapt to such rapid shifts in sentiment within a specific period, leading to less accurate predictions.
4. Contextual understanding: Sentiment analysis models might face challenges in understanding contextual nuances and sarcasm present in text data. This limitation can lead to misinterpretation of sentiment, particularly in social media posts or informal discussions about Bitcoin.
5. Generalizability: Models trained on sentiment analysis for a specific period might have limited generalizability beyond that timeframe. Bitcoin sentiment dynamics can vary over time due to market trends, regulatory changes, or technological advancements. Therefore, the model's performance may not be as effective when applied to other periods or when predicting sentiment in the future.
6. Overreliance on textual data: Sentiment analysis algorithms primarily rely on textual data such as social media posts or news articles. However, sentiment can also be expressed

through other modalities like images, videos, or audio, which are not considered in text-based sentiment analysis models. Neglecting these alternative forms of expression can limit the comprehensiveness and accuracy of sentiment analysis.

7 Challenges with sentiment analysis tools: While widely used Python packages like TextBlob and VADER offer convenient sentiment analysis methods, they are not infallible. These tools can produce inaccurate sentiment scores, especially when faced with sarcasm or nuanced language. Misinterpreting tweet sentiment can result in an improper training set and subsequently impact the accuracy of sentiment analysis results for Bitcoin during the specified period.

8 Presence of spam accounts, false accounts, and bots: Social media platforms like Twitter are often plagued by spam accounts, false accounts, and bots that generate automated content. These accounts can significantly influence sentiment analysis results by spreading false information or manipulating sentiment through coordinated campaigns. If your dataset includes a significant portion of such accounts, it can introduce biases and inaccuracies into sentiment analysis results for Bitcoin during the chosen period.

9 Discrepancies between sentiment and price movement: It's important to note that sentiment and price movement of Bitcoin don't always align perfectly. Positive sentiment can coexist with a decline in Bitcoin's price, and vice versa. Various factors, such as market manipulation, external events, or speculative trading behavior, can contribute to this discrepancy. Relying solely on sentiment analysis to predict price movements or draw conclusions about Bitcoin's performance during the specified period may lead to inaccurate findings.

Considering these limitations is crucial when conducting sentiment analysis on Bitcoin using machine learning algorithms. Addressing these challenges involves careful data collection, preprocessing, model training, and validation techniques to enhance the accuracy and applicability of sentiment analysis results.

Future work

There are several potential improvements that can be made to enhance this study in the future. For instance, to enhance the performance of the model, it would be beneficial to collect tweets from high-profile accounts with a significant influence on market movements. Additionally, considering the removal of false Twitter accounts, which can distort sentiment analysis, could be explored. To further improve the model's accuracy, incorporating a larger volume of data and expanding the range of trade dates could be considered. Moreover, selecting features that demonstrate strong correlations with the target variable or aggregating all features and subsequently choosing the most effective classifier could enhance accuracy.

Furthermore, it is worth exploring the use of various machine learning techniques for computing the sentiment score. Rather than relying on pre-packaged libraries like TextBlob or VADER, training and testing the Twitter data directly could lead to more precise results. These libraries may sometimes misinterpret positive texts as negative, resulting in inaccurate sentiment ratings. Therefore, utilizing our own data for training and testing purposes could refine the proposed approach.

The current study primarily relies on the Twitter dataset, which has limitations for training and testing the sentiment analysis model. Although the prices of cryptocurrencies are influenced by global investor sentiments, the dataset only focuses on tweets written in English. However, this study specifically aims to forecast the prices of BTC, the most-traded cryptocurrency.

To broaden the scope of future research, it is important to consider other factors that impact the cryptocurrency market beyond just investor sentiments. Additionally, conducting experiments involving multiple popular cryptocurrencies and improving the preprocessing steps to explore the correlation between cryptocurrency price volatility, news events, and investor sentiments on different social media platforms directly related to cryptocurrency prices would be valuable.

In order to create a more comprehensive and adaptable system, the development of a robust big data platform is recommended. This platform could continuously learn, predict, and update itself in real-time. It could be applied to analyze various real-time market trends such as stock prices, customer loyalty, or even election results. In addition to tweets, sentiments could be extracted from diverse sources including IRC channels, news articles, images, and videos from platforms like YouTube or TV channels. This customization would enable the platform to address a wide range of tasks that require prediction based on social media sentiments.

In the future, there are plans to develop a user-friendly front-end interface for this system. This interface would visually represent trends and provide historical aggregated data based on user input. Additionally, it could offer the flexibility to adjust the time window for predictions, enabling forecasts for longer time horizons.

References

1. Al Amrani, Y., Lazaar, M., & El Kadiri, K. E. (2022). Sentiment Analysis and Cryptocurrency Price Forecasting. International Journal of Advanced Computer Science and Applications, 13(10), 889-898. https://thesai.org/Downloads/Volume13No10/Paper_105_Cryptocurrency_Price_Prediction_using_Forecasting_and_Sentiment_Analysis.pdf

2. E. Edgari, J. Thiojaya and N. N. Qomariyah, "The Impact of Twitter Sentiment Analysis on Bitcoin Price during COVID-19 with XGBoost," 2022 5th International Conference on Computing and Informatics (ICCI), New Cairo, Cairo, Egypt, 2022, pp. 337-342, doi: 10.1109/ICCI54321.2022.9756123. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/stamp/stamp.jsp?tp=&arnumber=9756123

3. A. Ibrahim, "Forecasting the Early Market Movement in Bitcoin Using Twitter's Sentiment Analysis: An Ensemble-based Prediction Model," 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), Toronto, ON, Canada, 2021, pp. 1-5, doi: 10.1109/IEMTRONICS52119.2021.9422647. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9422647

4. A. M. Balfagih and V. Keselj, "Evaluating Sentiment C1assifiers for Bitcoin Tweets in Price Prediction Task," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 5499-5506, doi: 10.1109/BigData47090.2019.9006140. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9006140

5. Jacques Fleischer, Gregor von Laszewski, Carlos Theran, Yohn Jairo Parra Bautista " Time Series Analysis of Blockchain-Based Cryptocurrency Price Changes ", 2022 https://paperswithcode.com/paper/time-series-analysis-of-blockchain-based#code

6. R. N, S. R. R, V. S. R and K. P. D, "Crypto-Currency Price Prediction using Machine Learning," 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2022, pp. 1455-1458, doi: 10.1109/ICOEI53556.2022.9776665. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9776665

7. C. Saravanakumar, S. K. Lakshmi, M. Prakash, M. Vijayakumar and C. Arun, "An Efficient Ensemble Model for Forecasting Time Series Analysis of Crypto Currency Data Using Machine Learning," 2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES), Chennai, India, 2021, pp. 1-5, doi: 10.1109/ICSES52305.2021.9633939. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9633939

8. R. Nikita, S. J. Subhashini, Y. B. S, V. K. Vavle and D. A, "A Review on Digital Coin Investing Predictor," 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 2022, pp. 1-6, doi: 10.1109/ICONAT53423.2022.9726092. https://ieeexplore-ieee-org.ezproxy.lib.torontomu.ca/document/9726092

9. Pan, Linxi. (2023). Cryptocurrency Price Prediction Based on ARIMA, Random Forest and LSTM Algorithm. BCP Business & Management. 38. 3396-3404. 10.54691/bcpbm.v38i.4313. https://www.researchgate.net/publication/369430933_Cryptocurrency_Price_Prediction_Based_on_ARIMA_Random_Forest_and_LSTM_Algorithm

10. Yulin Liu, Luyao Zhang "Cryptocurrency Valuation: An Explainable AI Approach", 2022 https://paperswithcode.com/paper/cryptocurrency-valuation-an-explainable-ai

11. Zhengyao Jiang, Jinjun Liang "Cryptocurrency Portfolio Management with Deep Reinforcement Learning", 2016 https://paperswithcode.com/paper/cryptocurrency-portfolio-management-with-deep

12. Quentin Le Baron (2018) "How to make profits in cryptocurrency trading with machine learning" https://medium.com/smileinnovation/how-to-make-profits-in-cryptocurrency-trading-with-machine-learning-edb7ea33cee4

13. Raghav Agrawal (2023) " Complete Guide to Ethereum Blockchain with Python" https://www.analyticsvidhya.com/blog/2023/01/complete-guide-to-web3-0-ethereum-blockchain-using-python/

14. Gurchetan1000 Singh (2018) “7 methods to perform Time Series forecasting (with Python codes)” https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/

15. Sharvari Santosh (2021). “A Comprehensive Guide On Data Visualization In Python” https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-data-visualization-in-python/

16. Emre Rençberoğlu (2019) “ Fundamental Techniques of Feature Engineering for Machine Learning” https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114

17. Cryptocurrent: 10 Early-Stage Bitcoin & Blockchain Startups To Watch 2017 https://www.cbinsights.com/research/bitcoin-blockchain-early-stage-startups-to-watch/

18. ProQuest Ebook Central – Reader https://ebookcentral.proquest.com/lib/ryerson/reader.action?docID=5780608

19. https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score/

20. https://www.analyticsvidhya.com/blog/2021/01/sentiment-analysis-vader-or-textblob/

21. Xue Tan and Rasha Kashef. 2019. Predicting the closing price of cryptocurrencies: a comparative study. In Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems (DATA '19). Association for Computing Machinery, New York, NY, USA, Article 37, 1–5. https://doi.org/10.1145/3368691.3368728

22. Ibrahim A, Kashef R, Li M, Valencia E, Huang E. Bitcoin Network Mechanics: Forecasting the BTC Closing Price Using Vector Auto-Regression Models Based on Endogenous and Exogenous Feature Variables. Journal of Risk and Financial Management. 2020; 13(9):189. https://doi.org/10.3390/jrfm13090189

23. Ahmed Ibrahim, Rasha Kashef, Liam Corrigan, Predicting market movement direction for bitcoin: A comparison of time series modeling methods, Computers & Electrical Engineering, Volume 89, 2021,106905, ISSN 0045-7906,https://doi.org/10.1016/j.compeleceng.2020.106905. (https://www.sciencedirect.com/science/article/pii/S0045790620307576)

24. Symeon Symeonidis, Dimitrios Effrosynidis, Avi Arampatzis, A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis, Expert Systems with Applications, Volume 110, 2018, Pages 298-310, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2018.06.022. (https://www.sciencedirect.com/science/article/pii/S0957417418303683)\

25. Kashfia Sailunaz, Reda Alhajj, Emotion and sentiment analysis from Twitter text, Journal of Computational Science, Volume 36, 2019, 101003, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2019.05.009. (https://www.sciencedirect.com/science/article/pii/S1877750318311037)

26. Li M, Kashef R, Ibrahim A. Multi-Level Clustering-Based Outlier's Detection (MCOD) Using Self-Organizing Maps. Big Data and Cognitive Computing. 2020; 4(4):24. https://doi.org/10.3390/bdcc4040024

27. Pano T, Kashef R. A Complete VADER-Based Sentiment Analysis of Bitcoin (BTC) Tweets during the Era of COVID-19. Big Data and Cognitive Computing. 2020; 4(4):33. https://doi.org/10.3390/bdcc4040033