Crypto Price Predicting LSTM Model

Author:VS001
Date:04-07-2024

1.Introduction

The cryptocurrency market is known for its high volatility and rapid price fluctuations, making it a challenging environment for traders and investors. Traditional financial analysis methods often fall short in predicting the unpredictable nature of crypto assets. To address this challenge, this project aims to build a predictive model that incorporates sentiment analysis of cryptocurrency-related news to enhance the accuracy of price predictions.

Sentiment analysis, a branch of natural language processing (NLP), involves extracting and quantifying the sentiment expressed in textual data. In the context of cryptocurrencies, the sentiment reflected in news articles, headlines, and social media posts can significantly influence market behavior. By analyzing this sentiment, we can gain insights into market trends and investor sentiment, which can, in turn, affect cryptocurrency prices.

In this project, we leverage Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN) specifically designed to handle sequential data and long-term dependencies. LSTM networks are well-suited for time-series forecasting, making them an ideal choice for predicting cryptocurrency prices based on historical data and sentiment analysis.

The project involves several key steps:

1. **Data Collection**: Gathering cryptocurrency-related news headlines, historical cryptocurrency prices, and international currency rates through web scraping techniques using Selenium and BeautifulSoup.
2. **Sentiment Analysis**: Preprocessing and analyzing the collected news headlines to assign sentiment scores, providing a quantitative measure of market sentiment.
3. **Predictive Modeling**: Using LSTM networks to model the relationship between cryptocurrency prices, sentiment scores, and other relevant features to forecast future price movements.

By combining sentiment analysis with advanced machine learning techniques, this project aims to create a robust predictive model that can aid traders and investors in making informed decisions. The integration of market sentiment into the predictive model is expected to provide a more comprehensive understanding of the factors driving cryptocurrency prices, ultimately leading to more accurate and reliable predictions.

This project not only contributes to the field of financial forecasting but also demonstrates the potential of combining NLP and deep learning techniques to tackle complex, real-world

problems. The code, methodology, and results of this project are documented and made available in this GitHub repository, encouraging collaboration and further development within the data science community.

1.1 Problem Statement

The cryptocurrency market is characterized by extreme volatility and rapid price fluctuations, presenting significant challenges for traders and investors. Traditional financial analysis methods often fail to predict these erratic price movements effectively. This unpredictability stems from various factors, including market sentiment, regulatory news, technological advancements, and macroeconomic trends. Market sentiment, particularly, plays a crucial role in influencing cryptocurrency prices. News articles, headlines, and social media posts can sway investor behavior, leading to sharp price movements.

Given this context, the primary problem addressed in this project is:

How can we accurately predict cryptocurrency prices by incorporating market sentiment derived from cryptocurrency-related news?

To tackle this problem, we aim to develop a predictive model that integrates sentiment analysis of news headlines with historical cryptocurrency price data. The objectives are:

1. **Sentiment Analysis**: Extract and quantify sentiment from cryptocurrency-related news articles and headlines.
2. **Predictive Modeling**: Use Long Short-Term Memory (LSTM) networks to analyze the relationship between historical price data and sentiment scores to forecast future cryptocurrency prices.
3. **Feature Integration**: Incorporate additional relevant features such as international currency rates to enhance the predictive power of the model.

The successful completion of this project will result in a predictive model that can provide more accurate forecasts of cryptocurrency prices, thereby assisting traders and investors in making more informed decisions. By leveraging sentiment analysis and advanced machine learning techniques, this project aims to bridge the gap between market sentiment and price prediction in the volatile world of cryptocurrencies.

1.2 Project Scope

This project encompasses the development and implementation of a predictive model for cryptocurrency prices, incorporating sentiment analysis of cryptocurrency-related news. The scope of the project is outlined as follows:

Data Collection

1. **News Headlines**: Collect cryptocurrency-related news articles and headlines from various financial and cryptocurrency news websites using web scraping techniques (Selenium and BeautifulSoup).
2. **Cryptocurrency Prices**: Gather historical and real-time price data for major cryptocurrencies (e.g., Bitcoin, Ethereum) from reputable crypto exchanges and financial platforms.
3. **International Currency Rates**: Obtain exchange rates for major international currencies to consider broader financial market influences.

Data Preprocessing

1. **Text Preprocessing**: Clean and preprocess the collected news headlines to remove noise and prepare the text for sentiment analysis.
2. **Sentiment Analysis**: Used Selenium model for Sentiment Analysis
3. **Time-Series Data Preparation**: Organize and format the historical price data and sentiment scores into a suitable structure for input into the predictive model.

Feature Engineering

1. **Feature Creation**: Create relevant features from the collected data, including:
    - Historical price data of stocks and Crypto Currencies
    - Sentiment scores
    - International currency rates
2. **Feature Scaling**: Normalize or scale the features as necessary to ensure compatibility with the LSTM model.

Model Development

1. **LSTM Network Design**: Design and implement an LSTM network tailored for time-series forecasting.
2. **Model Training**: Train the LSTM model using the prepared dataset, adjusting hyperparameters to optimize performance.
3. **Model Evaluation**: Evaluate the trained model's performance using appropriate metrics (e.g., mean squared error, root mean squared error), and validate the model with a separate test dataset.

Model Integration

1. **Feature Integration**: Integrate additional relevant features, such as international currency rates, into the predictive model to enhance accuracy.
2. **Performance Tuning**: Fine-tune the model based on evaluation results and incorporate feedback for improvement.

Implementation and Testing

1. **Deployment**: Implement the predictive model and test it in a real-time or near real-time environment to evaluate its practical applicability.
2. **Error Analysis**: Conduct error analysis to identify potential areas of improvement and refine the model accordingly.

Documentation and Reporting

1. **Code Documentation**: Document the code, methodology, and processes used in the project to ensure reproducibility and ease of understanding.
2. **Project Report**: Prepare a comprehensive project report detailing the objectives, methodology, results, and conclusions.
3. **GitHub Repository**: Create a GitHub repository to host the project code, documentation, and related resources, encouraging collaboration and further development.

Conclusion and Future Work

1. **Project Summary**: Summarize the project's outcomes, highlighting the effectiveness of combining sentiment analysis with LSTM networks for cryptocurrency price prediction.
2. **Future Enhancements**: Identify potential areas for future enhancements, such as incorporating more data sources, improving sentiment analysis techniques, and exploring other machine learning models.

By defining a clear and comprehensive scope, this project aims to develop a robust and accurate predictive model for cryptocurrency prices, leveraging the power of sentiment analysis and advanced machine learning techniques.

2. System Analysis

2.1. Functional Specifications

This section outlines the functional requirements and specifications of the system, detailing the processes and workflows involved in the project.

2.1.1. Data Workflow and Pipelines

The data workflow and pipelines for this project involve several key steps, from data collection to model deployment. The primary components of the workflow are:

1. **Data Collection Pipeline**
    - **News Scraping**: Use Selenium and BeautifulSoup to scrape cryptocurrency-related news articles and headlines from multiple sources, including reputable financial news websites and cryptocurrency-specific platforms.
    - **Price Data Collection**: Gather historical and real-time price data for major cryptocurrencies from financial APIs and web scraping of crypto exchanges.
    - **Currency Rates Collection**: Obtain international currency rates from reliable financial websites or APIs to consider the broader financial context.
2. **Data Preprocessing Pipeline**
    - **Text Cleaning**: Remove noise, punctuation, stopwords, and perform tokenization on the collected news headlines to prepare the text for sentiment analysis.
    - **Sentiment Analysis Using GEMINI-1.5-PRO-LATEST**: Apply the GEMINI-1.5-PRO-LATEST model to analyze the sentiment of each news headline and assign sentiment scores. This model leverages advanced natural language processing techniques to provide accurate sentiment analysis.
    - **Feature Engineering**: Create relevant features from the collected data, including:
        - Historical cryptocurrency prices
        - Sentiment scores from the GEMINI-1.5-PRO-LATEST model
        - International currency rates
    - **Feature Scaling**: Normalize or scale the features as necessary to ensure compatibility with the LSTM model.
3. **Model Training Pipeline**
    - **Data Splitting**: Split the dataset into training, validation, and test sets to evaluate the model's performance effectively.
    - **LSTM Network Design**: Design and configure the LSTM network architecture for time-series forecasting, considering factors such as the number of layers, number of neurons per layer, activation functions, and dropout rates.
    - **Model Training**: Train the LSTM model using the training dataset and validate its performance on the validation set. Adjust hyperparameters such as learning rate, batch size, and epochs to optimize the model's performance.

4. **Model Evaluation Pipeline**
    - **Performance Metrics**: Evaluate the model using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) to measure its accuracy and robustness.
    - **Validation**: Validate the model on a separate test dataset to ensure its generalizability and effectiveness in real-world scenarios.
5. **Model Deployment Pipeline**
    - **Integration**: Integrate the trained LSTM model into a real-time or near real-time environment for practical use. This may involve deploying the model as a web service or integrating it into a trading platform.
    - **Monitoring and Maintenance**: Continuously monitor the model's performance and update it with new data to maintain accuracy. Implement mechanisms for periodic retraining and performance evaluation.

2.2. System Requirements

The system requirements for this project include both software and hardware components necessary for the successful execution and deployment of the predictive model.

Software Requirements

1. **Programming Languages**
    - Python: For data collection, preprocessing, modeling, and deployment.
2. **Libraries and Frameworks**
    - **Web Scraping**: Selenium, BeautifulSoup
    - **Data Manipulation**: Pandas, NumPy
    - **Natural Language Processing**: GEMINI-1.5-PRO-LATEST
    - **Machine Learning**: pytorch, Scikit-learn
    - **Data Visualization**: Matplotlib, Seaborn
3. **Development Tools**
    - Jupyter Notebook: For interactive data analysis and model development.
    - Integrated Development Environment (IDE): Such as PyCharm or VS Code for code development and debugging.
4. **Version Control**
    - Git: For version control and collaboration.
    - GitHub: For hosting the project repository and documentation.

Hardware Requirements

1. **Processor**: A multi-core processor (e.g., Intel i5/i7 or AMD Ryzen 5/7) for efficient computation.
2. **Memory**: At least 16 GB of RAM to handle large datasets and complex model training processes.
3. **Storage**: SSD with a minimum of 256 GB of storage for fast read/write operations and to store datasets.

4. **Graphics Processing Unit (GPU)**: An optional but recommended GPU (e.g., NVIDIA GTX 1060 or higher) for accelerated deep learning model training.

4. Implementation

4.1. Coding Standard

To ensure consistency, readability, and maintainability of the code throughout the project, the following coding standards will be adhered to:

4.1.1. General Coding Practices

- **Consistent Naming Conventions**: Use meaningful and descriptive names for variables, functions, classes, and modules. Follow the naming conventions:
    - Variables and functions: `snake_case`
    - Classes: `CamelCase`
    - Constants: `UPPER_SNAKE_CASE`
- **Code Documentation**: Include docstrings for all modules, classes, and functions. Use inline comments to explain complex logic or important sections of the code.
- **Error Handling**: Use try-except blocks to handle potential errors gracefully and log meaningful error messages.
- **Modular Code**: Break down the code into reusable functions and classes to avoid repetition and improve readability.
- **Version Control**: Use Git for version control. Commit changes frequently with clear and descriptive commit messages. Use branches for developing new features or fixing bugs and merge them into the main branch after thorough testing.

4.1.2. Python-Specific Standards

- **PEP 8 Compliance**: Follow the PEP 8 style guide for Python code. This includes:
    - Indentation: Use 4 spaces per indentation level.
    - Line Length: Limit lines to a maximum of 79 characters.
    - Blank Lines: Use blank lines to separate top-level function and class definitions and larger blocks of code inside functions.
    - Imports: Group imports in the following order: standard library imports, related third-party imports, and local application/library-specific imports. Use a single import per line.
- **String Formatting**: Use f-strings (formatted string literals) for string formatting.
- **Logging**: Use the `logging` module for logging rather than print statements. Configure different logging levels (DEBUG, INFO, WARNING, ERROR, CRITICAL) appropriately.
- **Environment Variables**: Use environment variables for configuration settings and sensitive information such as API keys.

4.2. Screenshots

4.2.1. Data Collection Pipeline

Crypto Policy Scraping:

Description: A screenshot showing the script running to scrape crypto policy data using BeautifulSoup.

Here we are collecting policy data from a leading crypto related updates Website called

***"the block"** total of 271 pages are there to scrape , so we use a for loop to change the pages and the soup will be collecting the headlines and dates from the website and finally storing it to a pandas dataframe.*

1. *The script imports necessary libraries: `time`, `pandas`, `requests`, and `BeautifulSoup`.*
2. *It defines the base URL and the range of pages to scrape.*
3. *An empty list `data_list` is initialized to store the scraped data.*
4. *The `scrape_page` function constructs the URL for each page, sends a GET request, and parses the HTML content if the request is successful.*
5. *It extracts headlines and publication dates using CSS selectors and appends them to `data_list`.*
6. *A loop iterates through the specified range of pages, calls the `scrape_page` function, and adds a 1-second delay between requests.*
7. *After scraping, a `pandas` DataFrame is created from `data_list`.*
8. *The DataFrame is saved to a CSV file named "scraped_data.csv".*
9. *A completion message is printed to indicate that the data has been saved.*

Scraping Policy Data

```python
import time
import pandas as pd
import requests
from bs4 import BeautifulSoup

# Define the base URL and the range of pages to scrape
base_url = "https://www.theblock.co/category/policy/"
start_page = 1
end_page = 271

# Initialize an empty list to store dictionaries
data_list = []

# Function to scrape data from a single page
def scrape_page(page_number):
    url = f"{base_url}{page_number}"
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        # Find all headline and date elements
        headlines = soup.select('div.collection__feed div.headline span')
        dates = soup.select('div.collection__feed div.meta div.pubDate')
        # Iterate over each headline and date and store in the list
        for headline, date in zip(headlines, dates):
            headline_text = headline.get_text(strip=True)
            date_text = date.get_text(strip=True)
            data_list.append({'headline': headline_text, 'date': date_text})
    else:
        print(f"Failed to fetch page {page_number}")

# Loop through pages and scrape data
for page_number in range(start_page, end_page + 1):
    print(f"Scraping data from page {page_number}...")
    scrape_page(page_number)
    time.sleep(1)  # Add a delay to be polite to the server

# Create DataFrame from the list of dictionaries
df = pd.DataFrame(data_list)

# Save the DataFrame to a CSV file
df.to_csv("scraped_data.csv", index=False)

print("Data saved to scraped_data.csv")
```

Basic Cleaning to remove certain unwanted data from the Date COLUMN.

```
# Assuming df is your existing DataFrame with the "date" column containing dates with "EDT"
df['date'] = df['date'].str.replace(' EDT', '')  # Remove 'EDT'
df['date'] = df['date'].str.replace(' EST', '')  # Remove 'EDT'
df['date'] = df['date'].str.strip()  # Remove leading and trailing whitespace
df['date'] = pd.to_datetime(df['date'], errors='coerce')  # Set errors='coerce' to handle parsing errors

# Print the first few rows of the DataFrame to verify the changes
print(df.head())
```

```
                                     headline                date
0  Hong Kong spot bitcoin ETFs could go live as s... 2024-04-16 02:54:00
1  Lawmakers demand information on CFTC chair's r... 2024-04-15 16:46:00
2  Nebraska man charged for mining $1 million in ... 2024-04-15 14:56:00
3  Potential movement on stablecoin legislation f... 2024-04-15 13:09:00
4  UK to legislate on cryptoasset regulatory fram... 2024-04-15 12:18:00
```

```
df.to_csv("theblock_policy_data.csv", index=False)
```

Result:

Policy News	Date
Hong Kong spot bitcoin ETFs could go live as soon as this month: OSL	16-04-24
Lawmakers demand information on CFTC chair's relationship with FTX founder Sam Bankman-Fried	15-04-24
Nebraska man charged for mining $1 million in cryptocurrencies via 'cryptojacking'	15-04-24
Potential movement on stablecoin legislation following talks with Sen. Schumer: TD Cowen	15-04-24
UK to legislate on cryptoasset regulatory framework by end of July, minister says	15-04-24
Hong Kong's 'spot-on' approval of spot ether ETFs hailed by OSL executive, eyes advantage over US	15-04-24
Hong Kong approves first batch of spot bitcoin, ether ETFs in drive to become crypto hub	15-04-24
IRS anticipates more crypto-related tax crime as filing deadline looms: report	15-04-24
Coinbase files interlocutory appeal in its case against the SEC	12-04-24
Securities law professor: It's time for Uniswap to 'prepare for war'	12-04-24
Ex-security engineer gets three years for first-ever smart contract hacking conviction, stole $12 million in cry	12-04-24
VeChain and Neo rally as proxy bets on Hong Kong ETF approval, analysts say	12-04-24
Hong Kong to approve spot bitcoin, ether ETFs as soon as Monday: Bloomberg	12-04-24
FBI probed bitcoin core developer event linked to Luke Dashjrâ€™s BTC hack: Mike Schmidt	11-04-24
The SEC plans to sue Uniswap, here's what's next	11-04-24
Craig Wright drops defamation appeal in Norway	11-04-24
Former FTX CEO Sam Bankman-Fried appeals conviction and sentence	11-04-24
Bitcoin price gains despite a pullback in global markets as ECB holds rates steady	11-04-24
London mayoral candidate wants to give Â£100 in crypto to every resident	11-04-24
Bitfinex Securities unveils $6.25 million tokenized debt issue for El Salvador airport hotel complex	11-04-24
Uniswap receives SEC lawsuit warning, CLO describes it as 'another abuse of power'	10-04-24
Former FTX exec Ryan Salame's sentencing set for May 28	10-04-24
Taiwanâ€™s ACE Exchange founder among seven indicted in $10.7 million fraud case	10-04-24
Former Ethereum advisor Steven Nerayoff sues US government for $9.6 billion over 'fabricated' charges	09-04-24
VanEck, CoinShares CEOs doubt SEC will approve spot Ethereum ETFs: CNBC	09-04-24

News Headlines Scraping:

Description: A screenshot showing the script running to scrape news headlines using Selenium and BeautifulSoup.

In this code we are scraping 1300 pages , since the size of the pages is a little too big to handle in one go, we are batching the pages to 250 at a time.Here we are also using proxies because sending multiple requests from one server may lead to getting blocked by the website.

1. ***Import Libraries**: The necessary libraries are imported:*
    - `time` *for adding delays between requests.*
    - `pandas` *for creating and saving data to a CSV file.*
    - `requests` *for making HTTP requests to fetch web pages.*
    - `BeautifulSoup` *from the* `bs4` *package for parsing HTML content.*
2. ***Define Base URL and Parameters**:*
    - `base_url` *is the base URL of the website to be scraped.*
    - `total_pages` *is the total number of pages to scrape.*
    - `pages_per_batch` *is the number of pages to scrape in each batch.*
3. ***List of Proxies**: A list of proxy servers is defined to rotate through during the scraping process, helping to avoid IP bans from the target server.*
4. ***Initialize Data Storage**: An empty list* `data_list` *is initialized to store the scraped data.*
5. ***Define Scraping Function**:*
    - `scrape_page(page_number, proxy)`*: This function takes a page number and a proxy as input, constructs the URL for the page, makes a GET request using the specified proxy, and parses the HTML content if the request is successful.*
    - *It extracts headlines and publication dates using CSS selectors and appends them to* `data_list`*.*
6. ***Scraping Loop**:*
    - *The script iterates through batches of pages, defined by* `pages_per_batch`*.*
    - *For each batch, it selects a proxy from the list.*
    - *For each page in the batch, it calls* `scrape_page(page_number, proxy)`*, handles any exceptions, and adds a delay of 15 seconds between requests.*
7. ***Create DataFrame and Save to CSV**:*
    - *After scraping, a* `pandas` *DataFrame is created from* `data_list`*.*
    - *The DataFrame is saved to a CSV file named "news_data2.csv".*
8. ***Completion Message**: The script prints a message indicating that the data has been saved to the CSV file.*

The code aims to collect a large amount of news data from a website while using proxies to distribute the load and avoid detection.

Final Scraper with Proxy and Batching

```python
import time
import pandas as pd
import requests
from bs4 import BeautifulSoup

# Define the base URL and the total number of pages to scrape
base_url = "https://www.theblock.co/latest?start="
total_pages = 1300
pages_per_batch = 250

# List of proxies to rotate through
proxies = [
    {'http': 'http://102.130.125.86:80'},
    {'http': 'http://85.198.13.205:80'},
    {'http': 'http://185.110.189.166:80'},
    {'http': 'http://103.168.254.62:8080'}



    # Add more proxies as needed
]

# Initialize an empty list to store dictionaries
data_list = []

# Function to scrape data from a single page with a specified proxy
def scrape_page(page_number, proxy):
    url = f"{base_url}{page_number}0"
    response = requests.get(url, proxies=proxy)  # Use specified proxy
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        # Find all headline and date elements
        headlines = soup.select('div.collection__feed div.headline span')
        dates = soup.select('div.collection__feed div.meta div.pubDate')
        # Iterate over each headline and date and store in the list
        for headline, date in zip(headlines, dates):
            headline_text = headline.get_text(strip=True)
            date_text = date.get_text(strip=True)
            data_list.append({'headline': headline_text, 'date': date_text})
    else:
        print(f"Failed to fetch page {page_number}")


# Loop through pages and scrape data
for batch_number in range(1,total_pages // pages_per_batch):
    start_page = -150+ batch_number * pages_per_batch +1
    end_page = start_page + pages_per_batch-1
```

```python
# Loop through pages and scrape data
for batch_number in range(1,total_pages // pages_per_batch):
    start_page = -150+ batch_number * pages_per_batch +1
    end_page = start_page + pages_per_batch-1
    print(f"Scraping data from pages {start_page}-{end_page}...")
    # Select proxy for this batch of requests
    proxy = proxies[batch_number % len(proxies)]
    for page_number in range(start_page, end_page + 1):
        try:
            scrape_page(page_number, proxy)
            print(f"Now scraping {page_number}")
        except Exception as e:
            print(f"Error occurred while scraping page {page_number}: {e}")
            break  # Stop scraping if an error occurs
        time.sleep(15)  # Add a delay to be polite to the server

# Create DataFrame from the list of dictionaries
df = pd.DataFrame(data_list)
# Clean and convert the "date" column to datetime format

# Save the DataFrame to a CSV file
df.to_csv("news_data2.csv", index=False)

print("Data saved to news_data2.csv")
```

```
Scraping data from pages 101-350...
Now scraping 101
Now scraping 102
Now scraping 103
Now scraping 104
Now scraping 105
Now scraping 106
Now scraping 107
Now scraping 108
Now scraping 109
Now scraping 110
Now scraping 111
Now scraping 112
Now scraping 113
Now scraping 114
Now scraping 115
Now scraping 116
Now scraping 117
Now scraping 118
Now scraping 119
Now scraping 120
Now scraping 121
Now scraping 122
Now scraping 123
Now scraping 124
```

Result:

News Report	Headline	Date
News Report	Bitcoin Price Analysis: Stagnation in No Manâ€™s Land May Lead to Major Move	27-Sep-18
News Report	Should You Trust the XRP Price Spike?	26-Sep-18
News Report	Bitcoin Fake News LIVE: Anatomy of a Disingenuous Express.co.uk Article	26-Sep-18
News Report	Why is Stellar (XLM) Rising? (Part 3)	26-Sep-18
News Report	Google To Allow Advertising of Regulated Crypto Exchanges	25-Sep-18
News Report	Bitcoin Adoption No Longer A Challenge	25-Sep-18
News Report	Why is Stellar (XLM) Rising? (Part 2)	25-Sep-18
News Report	Juventus to Launch â€˜Official Fan Tokenâ€™ in Q1 2019	24-Sep-18
News Report	Open-Source Operating System Coming to Cryptocurrency Miners	24-Sep-18
News Report	Why is Stellar (XLM) Rising?	24-Sep-18
News Report	Planning to Start Your Own Crypto Startup? Hereâ€™s How!	23-Sep-18
News Report	Can Cryptocurrency Spark a Small Business Boom?	23-Sep-18
News Report	Bitcoin Is Beefing Up Its Privacy Features	22-Sep-18
News Report	2 Ridiculous Targets for Bitcoin Traders This Week	21-Sep-18
News Report	XRP Skyrockets to 2nd Place Following xRapid Development News	21-Sep-18
News Report	Cryptocurrency Market Turns Bullish After Bitcoin ETF Delay	21-Sep-18

Crypto Price Data Collection:

Description: A screenshot of the script collecting cryptocurrency price data using Websraping

Here in this code we are using selenium to perform certain actions inside the page ,

The code is a web scraping script that uses Selenium and BeautifulSoup to extract historical cryptocurrency data from CoinMarketCap and save it to CSV files. Here's a concise description:

1. ***Import Libraries**: The script imports necessary libraries including Selenium, BeautifulSoup, csv, and time.*
2. ***Define Coins and URL**: A list of cryptocurrency coins is defined along with the base URL for historical data on CoinMarketCap.*
3. ***Initialize Variables**: An empty list to track coins that fail during the scraping process is initialized.*
4. ***Set up Selenium**: For each coin, Selenium is used to open the corresponding URL and interact with the web page to set the date range for the historical data.*
5. ***Click Through Date Range**: The script clicks various buttons to set the date range from the past to the present.*
6. ***Scrape Data**: BeautifulSoup is used to parse the page source and extract the data from the table.*
7. ***Save to CSV**: The extracted data is saved to a CSV file named after the coin.*
8. ***Error Handling**: Any coins that encounter errors are added to a list.*
9. ***Close Browser**: The browser is closed after processing each coin.*
10. ***Output**: A list of coins that failed to be processed is printed at the end.*

This script automates the extraction and storage of historical data for multiple cryptocurrencies.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException
from bs4 import BeautifulSoup
import csv
import time

# Define the URL
coins = ['agoras-tokens', 'arbitrum', 'axie-infinity', 'banx', 'basic-attention-token', 'bilshares', 'binance-coin', 'binance-usd', 'bitcoin-bep2', '
not_working_coins = []

for coin in coins:
    url = f"https://coinmarketcap.com/currencies/{coin}/historical-data/"

    try:
        # Set up Selenium
        driver = webdriver.Chrome()
        driver.get(url)
        time.sleep(3)

        # Find and click the first button with the text "Weekly"
        first_button_xpath = "//button[.//span[@class='act-label' and text()='Weekly']]"
        first_button = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.XPATH, first_button_xpath)))
        first_button.click()
        time.sleep(3)

        # Find and click the fourth button (assuming it's the date button)
        date_buttons = driver.find_elements(By.XPATH, "//button[contains(@class, 'BaseButton')]")
        date_button = date_buttons[3]
        date_button.click()
        time.sleep(3)

        for i in range(110):
            third_button = driver.find_element(By.XPATH,'//div[@class="sc-b36483b8-1 hUsaJQ"]//span[@class="icon-Chevron-left "]')
            third_button.click()
        time.sleep(3)

        fourth_button = driver.find_element(By.XPATH, "//div[contains(@class, 'react-datepicker__day react-datepicker__day--001') and text()='1']")
        fourth_button.click()
        time.sleep(3)

        for i in range(122):
            fifth_button = driver.find_element(By.XPATH,'//div[@class="sc-b36483b8-1 hUsaJQ"]//span[@class="icon-Chevron-right "]')
            fifth_button.click()
        time.sleep(3)

        sixth_button = driver.find_element(By.XPATH, "//div[contains(@class, 'react-datepicker__day react-datepicker__day--001') and text()='1']")
        sixth_button.click()
        time.sleep(3)
```

```
        # Click the button using JavaScript
        sev_button= driver.find_element(By.CSS_SELECTOR, "button.sc-2861d03b-0.bcCCXI")
        driver.execute_script("arguments[0].click();", sev_button)
        time.sleep(3)


        max_attempts = 5

        for attempt in range(max_attempts):
            try:
                eighth_button = driver.find_element(By.CSS_SELECTOR, "button.sc-2861d03b-0.iqkKeD")
                driver.execute_script("arguments[0].click();", eighth_button)
                time.sleep(2)
            except NoSuchElementException:
                print("Eighth button not found. Skipping click action.")
                break  # Break out of the loop if the button is not found
        time.sleep(10)
        # Scrape table data
        soup = BeautifulSoup(driver.page_source, 'html.parser')
        table = soup.find('table', class_='sc-14cb040a-3 eGIvUX cmc-table')
        if table:
            rows = table.find_all('tr')

            # Define CSV file name
            csv_file_name = f"{coin}___historical_data.csv"
            # Define column headings
            headings = ['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Market Cap']
           # Write table data to CSV file
            with open(csv_file_name, 'w', newline='', encoding='utf-8') as csvfile:
                csv_writer = csv.writer(csvfile)
                # Write column headings
                csv_writer.writerow(headings)
                for row in rows:
                    cells = row.find_all('td')
                    row_data = [cell.text.strip() for cell in cells]
                    csv_writer.writerow(row_data)
                print(f"Data scraped and saved to '{csv_file_name}'")
        else:
            print("Table not found on the page.")
    except Exception as e:
        print(f"An error occurred while processing {coin}: {e}")
        not_working_coins.append(coin)

    # Close the browser
    driver.quit()

print("Coins with URLs not working:", not_working_coins)
```

Result:

Date	Open	High	Low	Close	Volume	Market Cap
01-May-24	$60,609.50	$60,780.50	$56,555.29	$58,254.01	$48,439,780,271	$1,147,184,157,458
30-Apr-24	$63,839.42	$64,703.33	$59,120.07	$60,636.86	$37,840,840,057	$1,193,544,962,108
29-Apr-24	$63,106.36	$64,174.88	$61,795.46	$63,841.12	$26,635,912,073	$1,257,121,259,561
28-Apr-24	$63,423.51	$64,321.48	$62,793.60	$63,113.23	$17,334,827,993	$1,242,661,116,274
27-Apr-24	$63,750.99	$63,898.36	$62,424.72	$63,419.14	$19,530,783,039	$1,248,878,755,079
26-Apr-24	$64,485.37	$64,789.66	$63,322.40	$63,755.32	$24,139,372,950	$1,255,299,007,880
25-Apr-24	$64,275.02	$65,275.21	$62,783.63	$64,481.71	$32,155,786,816	$1,269,733,248,757
24-Apr-24	$66,408.72	$67,075.37	$63,589.87	$64,276.90	$30,276,655,120	$1,265,561,280,453
23-Apr-24	$66,839.89	$67,199.24	$65,864.87	$66,407.27	$24,310,975,583	$1,307,512,895,862
22-Apr-24	$64,935.63	$67,233.96	$64,548.18	$66,837.68	$28,282,686,673	$1,315,994,975,669
21-Apr-24	$64,992.82	$65,723.24	$64,277.72	$64,926.64	$20,506,644,853	$1,278,296,008,665
20-Apr-24	$63,851.10	$65,442.46	$63,172.40	$64,994.44	$23,097,485,495	$1,279,569,409,422
19-Apr-24	$63,510.75	$65,481.60	$59,651.39	$63,843.57	$49,920,425,401	$1,257,063,743,135
18-Apr-24	$61,275.32	$64,125.69	$60,833.48	$63,512.75	$36,006,307,335	$1,250,301,972,764
17-Apr-24	$63,831.85	$64,486.36	$59,768.59	$61,276.69	$41,915,247,049	$1,206,240,184,791
16-Apr-24	$63,419.30	$64,355.67	$61,716.40	$63,811.86	$42,847,528,078	$1,256,510,316,192
15-Apr-24	$65,739.64	$66,878.65	$62,332.07	$63,426.21	$43,595,917,654	$1,248,326,445,647
14-Apr-24	$63,836.23	$65,824.43	$62,205.85	$65,738.72	$49,084,320,047	$1,293,922,154,521
13-Apr-24	$67,188.38	$67,931.43	$60,919.11	$63,821.47	$52,869,738,185	$1,256,414,309,702
12-Apr-24	$70,061.38	$71,222.74	$65,254.83	$67,195.86	$44,129,299,406	$1,322,318,848,763
11-Apr-24	$70,575.73	$71,256.24	$69,571.81	$70,060.61	$30,153,382,941	$1,378,806,199,022
10-Apr-24	$69,140.24	$71,093.43	$67,503.57	$70,587.88	$38,318,601,774	$1,388,869,502,131
09-Apr-24	$71,632.50	$71,742.51	$68,212.92	$69,139.02	$36,426,900,409	$1,360,554,169,765
08-Apr-24	$69,362.55	$72,715.36	$69,064.24	$71,631.36	$37,261,432,669	$1,409,529,731,788
07-Apr-24	$68,897.11	$70,284.43	$68,851.63	$69,362.55	$21,204,930,369	$1,364,795,785,582

Stock Price Data Collection:

Description: A screenshot of the script collecting Stocks price data using Websraping

This code is a web scraping script that uses Selenium and BeautifulSoup to extract historical stock data from Yahoo Finance for a list of tickers and then combines the data into a single DataFrame. Here's a brief description:

1. ***Import Libraries**: Necessary libraries `pandas`, `selenium`, `BeautifulSoup`, and `time` are imported.*
2. ***Set Up WebDriver**: Selenium WebDriver is initialized to automate browser interactions.*
3. ***Define Tickers**: A list of stock tickers is defined.*
4. ***Initialize Data Storage**: An empty list `all_data` is created to store the data for each ticker.*
5. ***Scraping Loop**: For each ticker in the list:*
    - *The Yahoo Finance historical data URL for the ticker is accessed.*
    - *The script waits for 10 seconds to ensure the page is fully loaded.*
    - *The HTML content of the page is retrieved and parsed using BeautifulSoup.*
    - *The script finds the table container and extracts the table header and row data.*
    - *A DataFrame is created for each ticker's data, and a new column for the ticker symbol is added.*
    - *The DataFrame is appended to the `all_data` list.*
6. ***Error Handling**: Any exceptions during the scraping process are caught and printed.*
7. ***Close WebDriver**: The browser is closed after processing all tickers.*
8. ***Combine DataFrames**: All individual DataFrames are concatenated into a single DataFrame.*
9. ***Print Result**: The final combined DataFrame is printed.*

This script automates the extraction and consolidation of historical stock data for multiple tickers from Yahoo Finance.

```python
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service

from bs4 import BeautifulSoup
import time

# Set up the WebDriver
service = Service()
driver = webdriver.Chrome(service=service)
tickers=['%5EGSPC','%5EIXIC','MSFT','GOOGL','AMZN','NVDA', 'PYPL', 'SQ', 'V', 'MA', 'COIN', 'MSTR', 'RIOT', 'MARA', 'GC=F', 'SI=F']
# Define the URL of the page to scrape


url='https://finance.yahoo.com/quote/MSFT/history/?period1=1420070400&period2=1718101333'
# Navigate to the page

driver.get(url)


# Optionally wait for the page to fully load
time.sleep(5)

# Get the page source and close the WebDriver
html = driver.page_source
driver.quit()

# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(html, 'html.parser')

# Find the table container
table_container = soup.find('div', class_='table-container')

if table_container:
    # Find all rows in the table
    rows = table_container.find_all('tr', class_='svelte-ewueuo')

    # Extract header data
    headers = [th.text.strip() for th in rows[0].find_all('th')]

    # Extract row data
    data = []
    for row in rows[1:]:
        cells = row.find_all('td')
        row_data = [cell.text.strip() for cell in cells]
        data.append(row_data)

    # Create DataFrame
    df = pd.DataFrame(data, columns=headers)
```

Result:

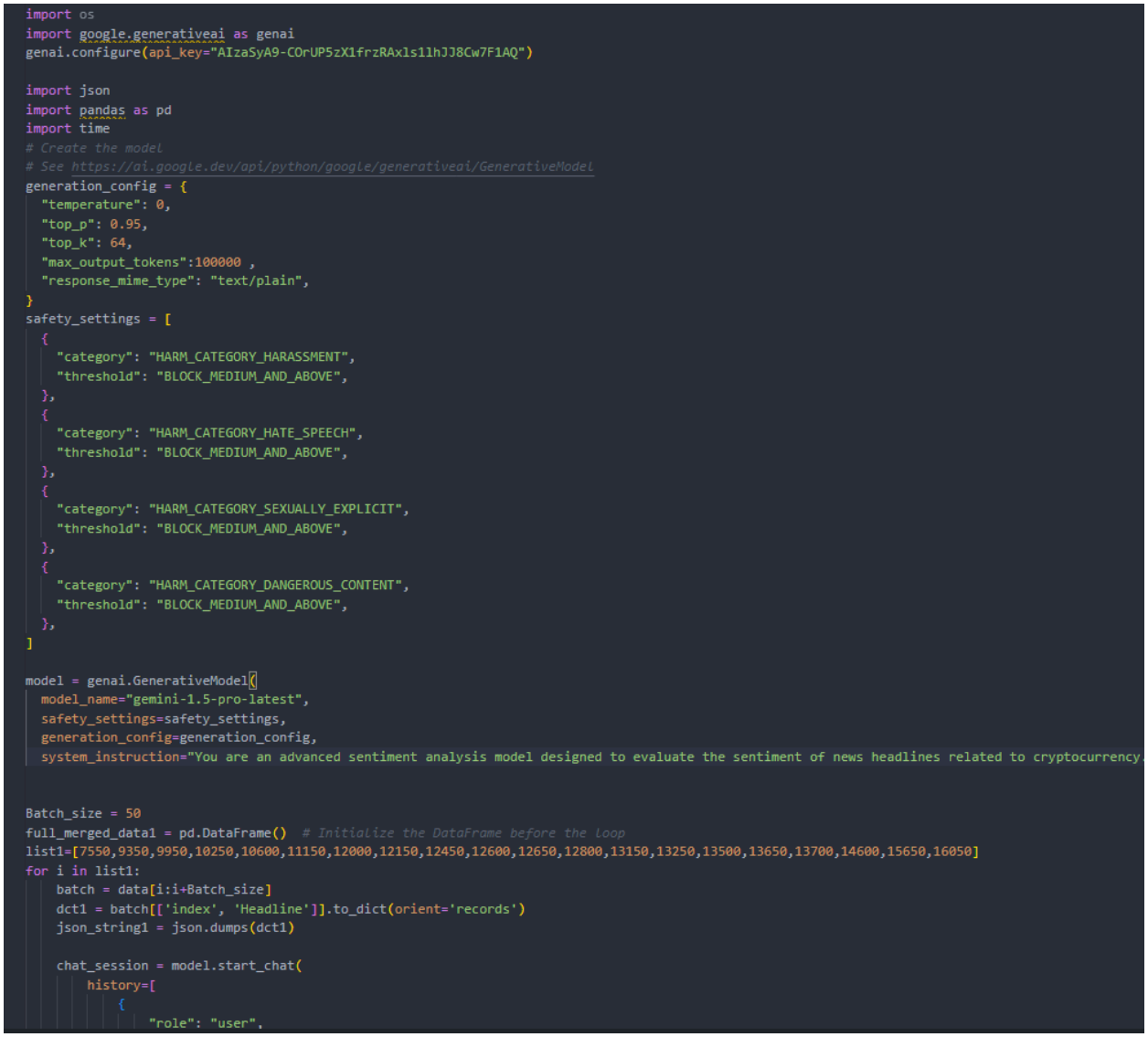
Date	Open	High	Low	Close	Adj Clo	Volume	Ticker
11-Jun-24	2300.00	2314.10	2300.00	2307.50	2307.50	871.00	GC=F
11-Jun-24	29.16	29.51	29.13	29.13	29.13	3.00	SI=F
10-Jun-24	5341.22	5365.79	5331.52	5360.79	5360.79	3622280000.00	%5EGSPC
10-Jun-24	17083.45	17213.45	17057.34	17192.53	17192.53	5207110000.00	%5EIXIC
10-Jun-24	424.70	428.08	423.89	427.87	427.87	14003000.00	MSFT
10-Jun-24	174.97	177.06	172.76	175.01	175.01	23779200.00	GOOGL
10-Jun-24	184.07	187.23	183.79	187.06	187.06	34494500.00	AMZN
10-Jun-24	120.37	123.10	117.01	121.79	121.78	314162700.00	NVDA
10-Jun-24	66.97	67.49	65.80	67.09	67.09	11978800.00	PYPL
10-Jun-24	64.27	64.94	63.52	64.34	64.34	9067100.00	SQ
10-Jun-24	278.14	278.56	273.38	275.04	275.04	5213800.00	V
10-Jun-24	448.42	450.17	444.37	449.25	449.25	1954500.00	MA
10-Jun-24	241.11	253.43	239.53	249.81	249.81	6344800.00	COIN
10-Jun-24	1575.01	1647.76	1560.29	1599.92	1599.92	726500.00	MSTR
10-Jun-24	9.66	10.01	9.47	9.90	9.90	15844500.00	RIOT
10-Jun-24	19.17	19.92	18.78	19.46	19.46	37683300.00	MARA
10-Jun-24	2290.60	2309.30	2290.50	2307.70	2307.70	269.00	GC=F
10-Jun-24	29.59	29.77	29.48	29.77	29.77	10.00	SI=F
07-Jun-24	5343.81	5375.08	5331.33	5346.99	5346.99	3692760000.00	%5EGSPC
07-Jun-24	17124.14	17229.31	17090.04	17133.13	17133.13	4755650000.00	%5EIXIC
07-Jun-24	426.20	426.28	423.00	423.85	423.85	13621700.00	MSFT
07-Jun-24	177.05	177.87	174.30	174.46	174.26	19661400.00	GOOGL
07-Jun-24	184.90	186.29	183.36	184.30	184.30	28021500.00	AMZN
07-Jun-24	119.77	121.69	118.02	120.89	120.88	412386000.00	NVDA
07-Jun-24	66.50	68.30	66.20	67.30	67.30	13195000.00	PYPL

4.2.2. Data Preprocessing Pipeline

Sentiment Analysis:

Description: A screenshot displaying the sentiment analysis script using the GEMINI-1.5-PRO-LATEST model.

This code uses Google's Generative AI to perform sentiment analysis on cryptocurrency news headlines. It processes headlines in batches of 50, sending each batch to the model and then waiting 10 seconds between requests. The model analyzes the sentiment and assigns a score, which is then merged with the original data and saved to a CSV file. If a JSON decoding error occurs, the batch is refreshed and retried.



```python
import os
import google.generativeai as genai
genai.configure(api_key="AIzaSyA9-COrUP5zX1frzRAx1s11hJJ8Cw7F1AQ")

import json
import pandas as pd
import time
# Create the model
# See https://ai.google.dev/api/python/google/generativeai/GenerativeModel
generation_config = {
  "temperature": 0,
  "top_p": 0.95,
  "top_k": 64,
  "max_output_tokens":100000 ,
  "response_mime_type": "text/plain",
}
safety_settings = [
  {
    "category": "HARM_CATEGORY_HARASSMENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
  },
  {
    "category": "HARM_CATEGORY_HATE_SPEECH",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
  },
  {
    "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
  },
  {
    "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE",
  },
]

model = genai.GenerativeModel(
  model_name="gemini-1.5-pro-latest",
  safety_settings=safety_settings,
  generation_config=generation_config,
  system_instruction="You are an advanced sentiment analysis model designed to evaluate the sentiment of news headlines related to cryptocurrency.


Batch_size = 50
full_merged_data1 = pd.DataFrame()  # Initialize the DataFrame before the loop
list1=[7550,9350,9950,10250,10600,11150,12000,12150,12450,12600,12650,12800,13150,13250,13500,13650,13700,14600,15650,16050]
for i in list1:
    batch = data[i:i+Batch_size]
    dct1 = batch[['index', 'Headline']].to_dict(orient='records')
    json_string1 = json.dumps(dct1)

    chat_session = model.start_chat(
        history=[
            {
                "role": "user",
```

```
chat_session = model1.start_chat(
    history=[
        {
            "role": "user",
            "parts": [
                json_string1
            ],
        },
    ]
)
try:
    response1 = chat_session.send_message("INSERT_INPUT_HERE")
    print("Sleeping")
    time.sleep(10)  # Wait for 30 seconds between requests
    print("Woke UP")
    # Clean the data by stripping the backticks and any unwanted characters
    json_data1 = response1.text.strip("`").strip("json")

    # Load the cleaned data and convert to DataFrame
    jsontolist1 = json.loads(json_data1)
    df_sample1 = pd.DataFrame(jsontolist1)

    # Merge the dataframes
    merged_data1 = pd.merge(data, df_sample1, on='index')

    # Update the full merged data DataFrame
    full_merged_data1 = pd.concat([full_merged_data1, merged_data1], ignore_index=True)

    # Save the full merged data to CSV
    full_merged_data1.to_csv(f'D:/DATA SCIENCE/Internship/CrypoDataAnalysis/test/merged_data_till.{i}.csv', index=False)
    print(f"Batch {i} completed.")
except json.JSONDecodeError as e:
    print(f"JSONDecodeError: {e}. Refreshing dct1 to a fresh batch.")
    # Refresh dct1 to a fresh batch
    dct1 = []
    i=i-1
    #Should Have added i=i-50 also
    continue
```

```
Batch 12600 completed.
Sleeping
Woke UP
Batch 12650 completed.
Sleeping
Woke UP
Batch 12800 completed.
Sleeping
Woke UP
Batch 13150 completed.
Sleeping
Woke UP
Batch 13250 completed.
Sleeping
Woke UP
Batch 13500 completed.
Sleeping
Woke UP
Batch 13650 completed.
Sleeping
Woke UP
Batch 13700 completed.
Sleeping
Woke UP
Batch 14600 completed.
Sleeping
Woke UP
Batch 15650 completed.
Sleeping
Woke UP
```

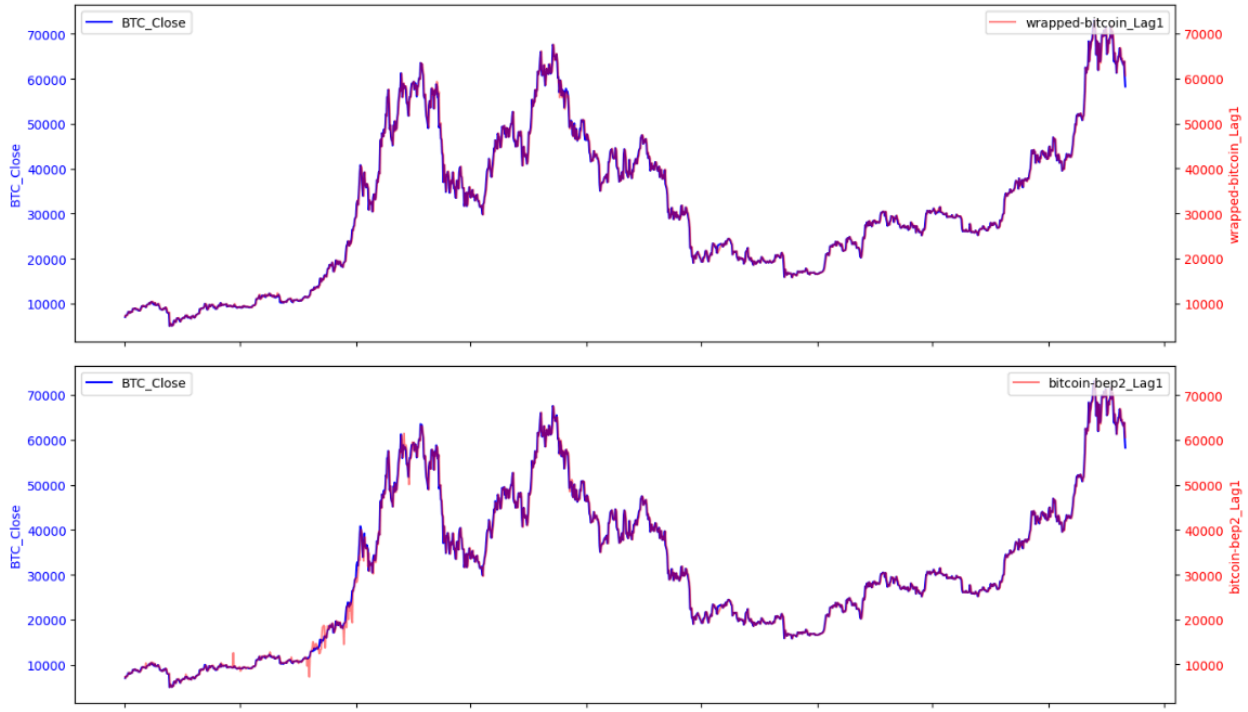
Result:

	index	Headline	Date	pred_score
0	1	UAE Pro League Teams Up With Chiliz to Bring W...	18-Apr-24	0.3
1	2	Polkadot Price Forecast: Will DOT Reach the $5...	18-Apr-24	0.1
2	3	Binance Opts for USDC Reserves While It Seeks ...	18-Apr-24	0.2
3	4	Worldcoin Introduces World Chain Layer-2 Amids...	18-Apr-24	0.0
4	5	BlackRock Was Tipped Off About High Inflation,...	18-Apr-24	0.4
5	6	Is the Bitcoin, Crypto Bull Market Over? Analy...	18-Apr-24	-0.1
6	7	Ripple (XRP) Price Analysis: 13% Recovery or 1...	17-Apr-24	0.0
7	8	This Is How Nearly 1 Billion Polygon (MATIC) W...	17-Apr-24	0.7
8	9	Shiba Inu (SHIB) Price Forecast: Potential Bul...	17-Apr-24	0.3
9	10	Will Pepe (PEPE) Price Mark a New All-Time Hig...	17-Apr-24	0.1

Exploratory Data Analysis:

Description: A screenshot showing the correlation of Features

Here we have found out that wrapped bitcoin and bitcoin bep2 are showing highest correlation with bitcoin close prices , Hence we select these two coins as the feature for our Model.




```
wrapped-bitcoin_Lag1    0.988161
bitcoin-bep2_Lag1       0.988102
Close_Lag1              0.988067
startcoin_Lag1          0.987991
                           ...
ftx-token_Lag1         -0.598334
monero_Lag1            -0.613284
prizm_Lag1             -0.696384
salt_Lag1              -0.748690
htx-token_Lag1         -0.894311
```

4.2.3. Model Training Pipeline

LSTM Model Training:

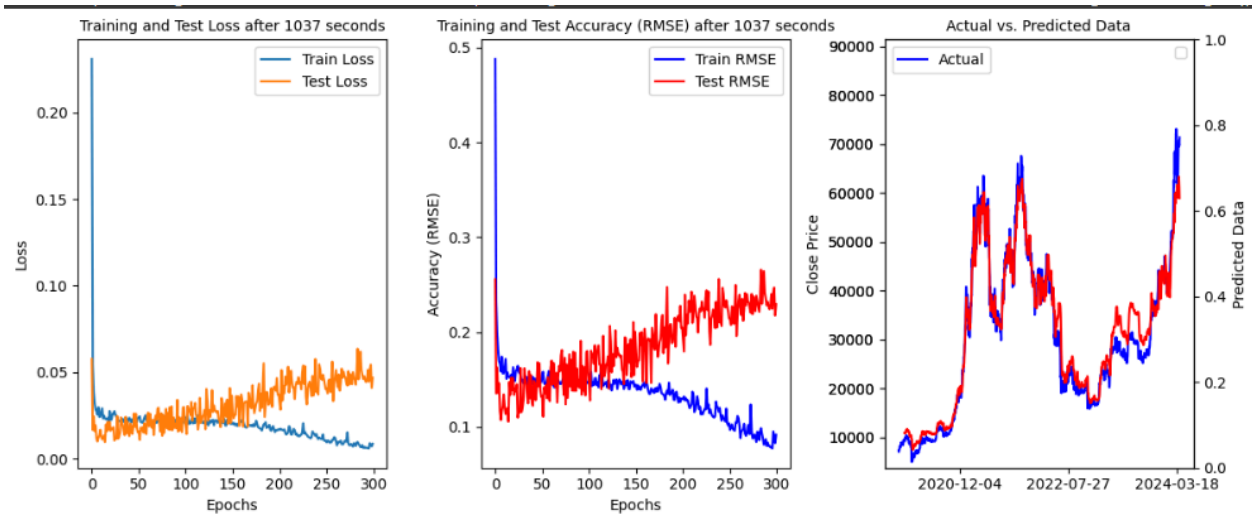
Description: A screenshot showing the LSTM model being trained, including the loss and accuracy metrics.

```
100% ████████████████████ 300/300 [17:16<00:00,  5.13s/it]
Epoch [1/300], Train Loss: 0.2309 | Test Loss: 0.0576 | Train RMSE: 0.4877 | Test RMSE: 0.2556 | Current LR: 0.00100000 | Duration: 4 seconds
Epoch [31/300], Train Loss: 0.0216 | Test Loss: 0.0148 | Train RMSE: 0.1461 | Test RMSE: 0.1282 | Current LR: 0.00100000 | Duration: 110 seconds
Epoch [61/300], Train Loss: 0.0208 | Test Loss: 0.0266 | Train RMSE: 0.1458 | Test RMSE: 0.1699 | Current LR: 0.00100000 | Duration: 211 seconds
Epoch [91/300], Train Loss: 0.0229 | Test Loss: 0.0176 | Train RMSE: 0.1489 | Test RMSE: 0.1411 | Current LR: 0.00100000 | Duration: 312 seconds
Epoch [121/300], Train Loss: 0.0218 | Test Loss: 0.0281 | Train RMSE: 0.1492 | Test RMSE: 0.1744 | Current LR: 0.00100000 | Duration: 413 seconds
Epoch [151/300], Train Loss: 0.0208 | Test Loss: 0.0285 | Train RMSE: 0.1443 | Test RMSE: 0.1786 | Current LR: 0.00100000 | Duration: 516 seconds
Epoch [181/300], Train Loss: 0.0173 | Test Loss: 0.0375 | Train RMSE: 0.1325 | Test RMSE: 0.2048 | Current LR: 0.00100000 | Duration: 618 seconds
Epoch [211/300], Train Loss: 0.0188 | Test Loss: 0.0383 | Train RMSE: 0.1371 | Test RMSE: 0.2097 | Current LR: 0.00100000 | Duration: 719 seconds
Epoch [241/300], Train Loss: 0.0125 | Test Loss: 0.0406 | Train RMSE: 0.1120 | Test RMSE: 0.2149 | Current LR: 0.00100000 | Duration: 820 seconds
Epoch [271/300], Train Loss: 0.0104 | Test Loss: 0.0469 | Train RMSE: 0.1025 | Test RMSE: 0.2314 | Current LR: 0.00100000 | Duration: 920 seconds
Epoch [300/300], Train Loss: 0.0086 | Test Loss: 0.0468 | Train RMSE: 0.0918 | Test RMSE: 0.2293 | Current LR: 0.00100000 | Duration: 1037 seconds
```

4.2.4. Model Evaluation Pipeline

Model Evaluation:

Description: A screenshot of the model evaluation process, displaying the performance metrics such as MSE and RMSE.



5. Conclusion

5.1. Design and Implementation Issues

During the design and implementation of the predictive model for cryptocurrency prices, several challenges and issues were encountered:

1. **Data Quality and Consistency**: Ensuring the quality and consistency of the scraped data was a significant challenge. Inconsistent data formats and missing values required extensive preprocessing and cleaning efforts to make the dataset suitable for modeling.
2. **Sentiment Analysis Accuracy**: Although the GEMINI-1.5-PRO-LATEST model provided robust sentiment analysis, accurately capturing the sentiment from diverse news sources with varying writing styles was challenging. Fine-tuning the sentiment model for better performance on crypto-related news could further improve sentiment accuracy.
3. **Model Overfitting**: The LSTM model showed signs of overfitting, particularly during the early epochs. To mitigate this, techniques such as dropout regularization, early stopping, and careful hyperparameter tuning were employed. Despite these efforts, occasional overfitting required continuous monitoring and adjustment.
4. **Computational Resources**: Training deep learning models like LSTM is computationally intensive. Limited computational resources and hardware constraints sometimes led to longer training times and the need for efficient resource management.
5. **Feature Selection**: Although a large amount of data was collected, only three features—close prices of Bitcoin, Bitcoin-BEP2, and Wrapped Bitcoin—were used for the LSTM model. Other collected features did not show enough correlation with Bitcoin's close price, presenting a challenge in effective feature selection.

5.2. Future Work

Several areas for future work and enhancement have been identified:

1. **Enhanced Sentiment Analysis**: Fine-tune the GEMINI-1.5-PRO-LATEST model specifically for crypto-related news to improve sentiment analysis accuracy. Explore additional sentiment analysis models and techniques to capture more nuanced sentiments.
2. **Extended Data Sources**: Expand the data collection pipeline to include more diverse sources of news and social media sentiment. Incorporating social media data, such as tweets and Reddit posts, could provide a more comprehensive view of market sentiment.
3. **Incorporation of Additional Features**: Further investigate and incorporate additional features that may influence Bitcoin's price. This includes exploring macroeconomic indicators, technical analysis features, and other cryptocurrencies to identify those with meaningful correlations.
4. **Model Optimization**: Experiment with other deep learning architectures, such as Transformer models, to potentially improve prediction accuracy and reduce overfitting.

Conduct hyperparameter optimization using techniques like grid search or Bayesian optimization.

5. **Real-Time Deployment**: Develop a robust real-time deployment pipeline to continuously update the model with new data and predictions. Implement mechanisms for real-time monitoring and alerting to ensure the model's reliability and performance in a live environment.
6. **User Interface Development**: Create a user-friendly interface or dashboard to visualize model predictions and performance metrics. This interface can help traders and investors make informed decisions based on the model's outputs.
7. **Collaboration and Open Source**: Encourage collaboration by making the project open source. Engage with the data science and crypto communities to gather feedback, share insights, and continuously improve the model.

By addressing these design and implementation issues and focusing on future enhancements, the predictive model can be further refined to provide more accurate and reliable forecasts of cryptocurrency prices. This will ultimately help traders and investors navigate the volatile crypto market with greater confidence.