REAL-TIME TRANSPARENT CRYPTO ANALYTICS PLATEFORM

Urs A. Hurni
University of Lausanne
urs.hurni@unil.ch

1 ABSTRACT

This report outlines the development of a web-based application designed for real-time sentiment analysis in the cryptocurrency market. Utilizing a user-friendly interface, the platform equips users with dynamic insights necessary for informed decision-making in a volatile trading environment. It is built on a modular architecture using Flask, a Python-based framework, ensuring ease of maintenance and scalability. Key features include real-time data fetching from various APIs, sophisticated prediction models, and effective caching mechanisms to enhance performance and responsiveness. The application integrates advanced analytical tools and ensures transparency by making all data and methods accessible to users.

2 INTRODUCTION

Cryptocurrency, especially with the rise of web3, aims to empower users by promoting transparency. In the ever-changing and unpredictable world of cryptocurrency, understanding market sentiment can provide valuable insights. It helps to gauge the emotions and psychology of market participants, which often influence price movements and trends. Unfortunately, many existing sentiment analysis tools are not very clear or reliable. For example, the popular fear and greed index does not display the algorithm or weighting used.

Transparency is particularly important in sentiment analysis for cryptocurrency markets. These markets are decentralized and known to be highly influenced by hype and panic, making them sensitive to changes in trader sentiment. Analysing sentiment in traditional stock markets is already challenging, and it's even more difficult in the more volatile cryptocurrency market. Many current tools lack transparency in their methods and don't provide access to the raw data used. This makes their insights less useful.

The usefulness of such applications goes beyond just individual traders or market analysts. In today's financial analysis, traditional models often fail to consider the irrational and emotional factors that drive market movements. Sentiment analysis tools can help fill this gap. By including sentiment data, financial models can become more robust and offer predictions that are more in line with real-world outcomes. For institutions, this means better risk management and more informed strategic decisions.

Furthermore, the significance of sentiment analysis is expected to grow as the cryptocurrency market evolves and more institutions get involved. As these markets integrate more with traditional financial systems, there will be a higher demand for advanced, transparent, and effective analytical tools. Sentiment analysis will play a key role in meeting this demand, helping both individual and institutional investors make better decisions based on a clearer understanding of market emotions and psychology.

3 DESCRIPTION

3.1 Research Question

The central research question of this project is: *How can programming tools be effectively utilized to develop an application for cryptocurrency sentiment analysis?*

3.2 Literature Review and Theoretical Background

The importance of this question is evident given the increasing complexity of financial markets and the crucial role of technology in financial analysis. Richard L. Peterson discusses how market sentiment affects financial markets, emphasizing the need for real-time data processing and natural language processing (NLP) tools to understand market trends [1]. Peterson's discussion highlights the essential need for strong web applications that can efficiently inform the user.

In the field of software engineering and web development, there are many strategies for improving web applications. One important strategy is the use of microservices architecture. This approach allows developers to build applications in a modular way, where different parts of the application, such as data fetching, processing, and user management, are created and deployed independently [2]. This method is particularly useful for financial analysis tools, which need to combine various data streams and processing functions in a clear and maintainable way.

Another key strategy is the use of caching mechanisms and session management techniques to enhance the performance of web applications. When a web application experiences heavy user traffic or a large influx of data, it can become slow or unresponsive. Caching solutions help by temporarily storing frequently accessed data, which can significantly reduce load times and lessen the strain on servers [3]. This is

especially important in cryptocurrency analysis, where users need quick access to the latest market data.

3.3 Objective

The main objective of this project is to develop a structured and flexible platform for investors. It will display relevant market information and include some prediction model, which uses the collected and standardized data to provide real-time insights and predictions about the cryptocurrency market. The platform is intended to have a user-friendly interface, making it easy for both beginners and experienced investors to understand and use the sentiment analysis to make informed decisions.

However, due to the lack of free APIs to get reliable news, the focus of the project will load the data once and use it for the subsequent analysis to avoid surpassing the free tier limit. This base system will also use additional free APIs to increase the data pool, recognizing that these sources may not always be as reliable or comprehensive as paid APIs, but it can still provide valuable guidance. If wanted, it can be easily enhance by just implementing new classes for new APIs and transforming the receiving data into the same format as the others.

To ensure the platform remains adaptable and doesn't require a complete overhaul in the future, it will be built with modular components using object-oriented programming (OOP). This approach allows for straightforward integration of new data sources by managing various data formats and making them compatible with the existing system. As access to better data sources is gained, they can be incorporated with minimal effort.

The project also focuses on increasing transparency in how market analyses are conducted. Documentation will be provided at each stage of the data processing and analysis, as well as complete docstrings for the methods used. The app will be hosted on GitHub for ease of access, clearly showing users the origins and reliability of the information they receive. This clarity will help foster trust and provide users with the confidence to rely on the platform.

3.4 Scope

This project is streamlined due to the restricted availability of free and reliable data sources. The primary goal is to develop a basic platform template by leveraging a few APIs to create a model that acts as a preliminary prototype for future enhancements. This initial version will display its capabilities with real market data and use the predictive models to generate valuable insights. This phase aims to validate the concept's efficacy and lay a solid foundation for potential scaling as more data becomes available or if there is further investment in the project.

Furthermore, the project encompasses the creation of an intuitive web interface. This interface is intended to enable easy access and navigation of the platform’s features, ensuring user-friendly operation.

By focusing on a basic structure and simple APIs, the project remains manageable. This focused approach ensures that the platform's essential components are solid and well-built before additional features are considered.

4 METHODOLOGY

4.1 Application Architecture

The application's architecture leverages Flask, a Python-based micro web framework, to handle HTTP requests and serve dynamic content. Flask was chosen for its simplicity and efficiency in small-scale projects, making it ideal for real-time data processing and response. Flask routes handle web requests and responses. They manage user sessions, from submissions, and redirect behaviours. The application structure is divided into several key modules, each responsible for distinct aspects of the program. Multiple API modules are used to fetch news and cryptocurrency data. Visualization and Analysis are regrouped in one module to generate plots and forecasts. The sentiment module is responsible to apply the FinBERT model in order to create a score on specific news data.

4.2 Real Time Data Fetching

To thoroughly encompass the diverse and nuanced landscape of cryptocurrency news, it is needed to gather information from a broad and varied dataset. Therefore, five specialized crypto news APIs have been selected as the primary sources.

This includes NewsAPI, CryptoPanicAPI, CryptoDataFetcher, CryptoNewsAPI, and SeekingAlphaNewsAPI.

- NewsAPI: Provides broad news coverage including general updates in the cryptocurrency sector.
- SeekingAlphaNewsAPI: Known for its detailed financial analysis, this API provides insights focused on cryptocurrency market movements and significant financial trends that could impact the cryptocurrency landscape.
- CryptoPanicAPI, CryptoDataFetcher, and CryptoNewsAPI: These APIs have been chosen for their availability and to enrich the dataset with targeted cryptocurrency news and timely updates, ensuring a comprehensive collection of information directly relevant to the crypto markets..

To enhance this global perspective, adding an economic perspective is interesting. Therefore, USEconomyAPI will be used for this macro economic perspective

- USEconomyAPI: Offers news articles focused on economic aspects that might impact or reflect the state of the broader economic markets.

Furthermore, one crypto price API will be used for gathering real time price data.

- Coinkranking API, provides real-time cryptocurrency pricing as well as historical pricing.

Each APIs has it's own modules that are used to fetch and retrieve news articles related to cryptocurrencies and economic context for adaptability and ease to use. They provide a rich and up-to-date dataset of textual content which is then analyzed for sentiment.

4.3 Prediction Model

Although the detailed prediction model is addressed in a complementary report, a brief overview is provided here.

It consist of :

- Studying the relationships between different market variables
- Using AutoRegressive Integrated Moving Average models for time series forecasting
- Implementing machine learning models to predict future price movements based on the fetched data.
- Utilizing those models to judge predictive accuracy by considering different metrics

These models help in understanding how sentiment analysis can predict future price movements, providing a comprehensive analytical tool for users.

4.4 Dashboard

The project's focus is on creating an intuitive and user-friendly dashboard.

The dashboard allows users to:

- Choose between different input parameters like time period (daily or monthly) and category (e.g., meme coins, DeFi projects, layer-1 solutions, …).
- Select the prediction model (linear regression, random forest, svr, gbm) developed in parallel in the complementary project.
- See the result in a comprehensive manner

5 IMPLEMENTATION

5.1 Program Implementation

The app has the following structure

1) Main page, *index.html*

2) Dashboard, *dashboard.html*

   a. Graphical visualisation of sentiment and price

   b. Arima forecast

   c. Lag correlations plots and models metrics

   d. Average predictions

The core functionality of the platform is contained within a Python script, *app.py*, which serves as the backend of the application. Several routes are set up that handle different aspects of user interaction and data processing.

1) The main page serves as the application's primary interface where users begin their interaction. The index route in the code serves as the entry point for the data. When a user accesses the base URL, they are greeted with a form on the *index.html* page, which allows them to specify their preferences for the data analysis. The form includes three dropdown menus that allow users to select the cryptocurrency category, the time frame for the data (days back), and the type of model for the analysis (e.g., linear, random forest). Once the form is submitted using a POST request, the user's choices are saved to the session, and they are redirected to the */dashboard* route. This setup is intended for personalizing the user experience by tailoring the data fetched and analysed to their specific interests and need. In addition to the form, the main page displays real-time cryptocurrency prices in a banner at the top, providing users with immediate market updates.
2) The dashboard route is where the bulk of data processing and user interaction takes place. Upon reaching this route, either through a POST request (with updated settings from the user) or a GET request (using existing session data), several operations occur:
   a. Data Fetching: Depending on the *days_back* and category specified, the appropriate news and price data are fetched. This might involve calling external APIs or retrieving cached data if available.
   b. Data Processing: The fetched data undergoes sentiment analysis and any necessary preprocessing. This include cleaning data, standardizing formats, and calculating sentiment scores.
   c. Visualization Generation: Using the processed data, various visualizations are generated to illustrate trends, correlations, or forecasts related to prices and sentiment. It displays a combined plot of sentiment analysis results and cryptocurrency price movements. These visualizations help users

visually comprehend the data. The ARIMA forecast is also presented in a plot format, providing predictions that help users anticipate potential market trends. Different lags can be selected here through a dropdown menu that shows the different lags options. This will refresh the updated page.

d. Data Display: The route then renders the *dashboard.html* template, displaying the fetched news in a table format, alongside the visualizations and sentiment analysis results. It also handles user interactions for further refining the analysis, such as changing the lag or the model type for predictions.

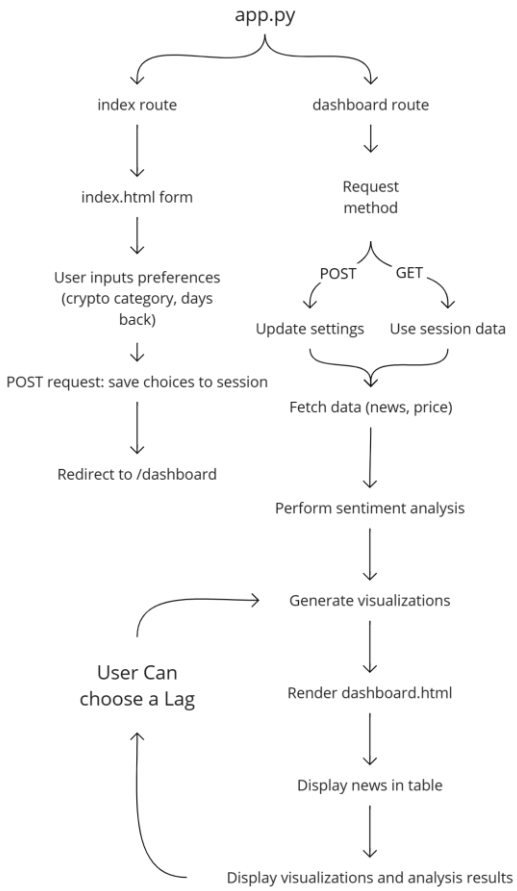

Fig. 1 : App Code's Structure

5.2 Modularity

To make the application modular, it was broken down by functionalities into separate, independent parts. Each part, or module, has a specific job, which makes the code easier to manage and more flexible.

API Handler Modules: These include *NewsAPI, USEconomyAPI, SeekingAlphaNewsAPI, CryptoPanicAPI, CryptoNewsAPI, and CryptoDataFetcher* classes. Each one is responsible for getting data from a specific source. They handle everything related to API requests, like authentication, dealing with errors, and fetching data.

DataHandler: This module cleans and standardizes the data. It makes sure that dates and formats are consistent across different datasets, so the data is ready for analysis.

SentimentAnalyzer: This module analyzes the sentiment of the news data. It uses natural language processing (NLP) to examine the text and assign sentiment scores, which are important for further analysis.

Visualizations: This module creates all the charts and graphs. It takes the processed data and turns it into visual formats that are easy for users to understand.

The connections between these modules are designed to keep them independent. Each module has a set of functions that other parts of the application can use. These functions are clear and well-documented through extended docstrings. The methods in each class are designed to do specific tasks related to that class. This means the internal workings of one class don't interfere with others.

Environment Variables are used for API keys and other sensitive information. This way, these details are not hard-coded into the modules and can be easily changed without modifying the code.

5.3 Data Fetching and Processing

In order to fetch real time data it is needed to have several APIs that gather information and retrieve it into the right format for further use.

Each dataset fetched from API will need to be transformed to match the same format and to allow further usage. Indeed, every API outputs it's data in different format. The *DataHandler* class (hd) plays a central role in managing the various tasks associated with data retrieval, processing, and storage. This section provides an overview of the data processing activities. All dates and header needs to be in the same formats. Thus, standardization is needed. Crucial to maintain consistency.

1) The datasets obtained from the APIs are merged into a single DataFrame. This consolidation is necessary to combine varied data points into a unified format for comprehensive analysis.
2) The merged data is then cleaned, this involves processing duplicates by comparing text fields across news articles
3) Cryptocurrency price data is transformed to align with news data as the merge occurs via the ‘date’.
4) Price data is scaled between 0 and 1 to match the scale of sentiment scores derived from news articles, facilitating comparative analysis.
5) Using resampling techniques, daily closing prices are computed by taking the last recorded price for each day, which is at 12PM.
6) Price changes are calculated as daily percentages to analyse the relationship between market movements and news sentiment.
7) To study the effect of news sentiment on subsequent price movements, the daily price change data is shifted backward by one day (shift(-1)), aligning it with the corresponding day’s sentiment data.
8) The processed price data and sentiment scores are merged based on date indices. This merged dataset includes key variables such as normalized prices, the following time period price change, and average time period sentiment. This combination is critical for examining the interplay between market sentiment derived from news and actual market performance.
9) The final dataset is reviewed for any missing values resulting from non-overlapping dates or the data shifting process. Rows containing NaN values are carefully handled to ensure the dataset's completeness and readiness for robust analysis.

The flowchart (fig. 2) outlines a the approach to data fetching using APIs. The process begins by loading necessary environment variables, such as API keys, to set up secure API communications. Next, an API class is initialized to configure endpoints and parameters. The system then makes requests to the specified APIs, iterating through each if multiple are involved. Upon receiving data, it processes the responses, checking for any errors. Errors are handled by appropriate measures like retrying requests or logging issues, whereas successful responses move to the next stage. The relevant data from these responses is extracted and organized into a DataFrame, a structured format suitable for analysis. If data from different sources are involved, these DataFrames are combined to consolidate the information.

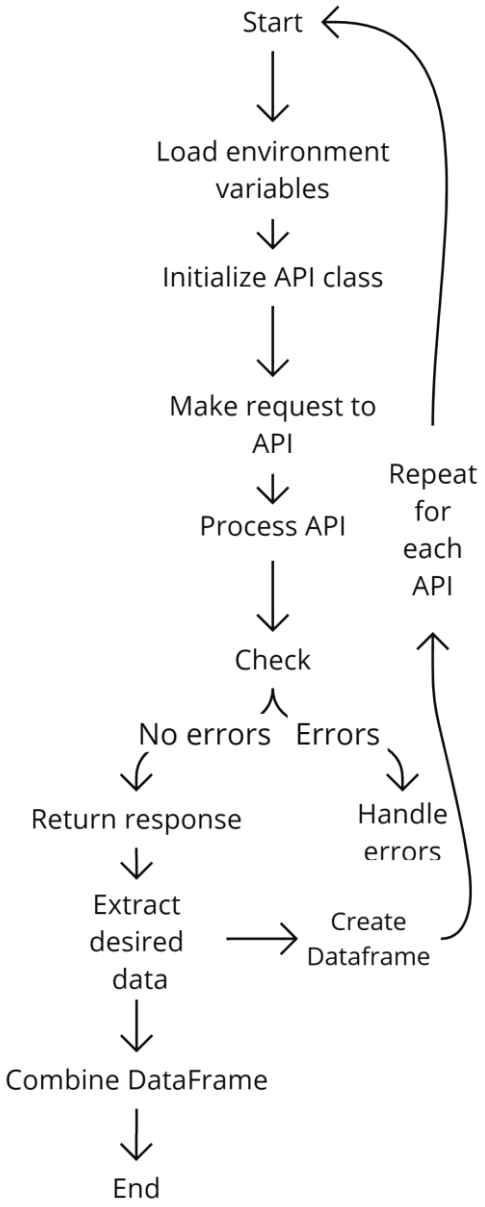

Fig. 2: API functionality Flowchart

5.4 Visualization and Response

The Visualizations class generates interactive charts using Matplotlib and integrates them into the Flask application via HTML templates. These visualizations dynamically represent the correlation between market sentiment and cryptocurrency prices, helping users make informed decisions.

This work with the *for_web* parameter when it is set to *True* in Python plotting functions, it modifies the output for web integration. Normally, plotting libraries like matplotlib display plots interactively or inline within a Jupyter notebook. For web applications, however, plots need to be in a format suitable for embedding into webpages. Setting *for_web* to *True* leads to the plot being rendered into a BytesIO buffer in SVG format, which is chosen for its scalability and high-quality

rendering on web pages. The SVG data is then converted to a string, which can be easily embedded into HTML or transmitted over the web, replacing the typical direct display of the plot. Moreover, data may be serialized into formats like JSON, including URLs for SVG images or data points that can be used by web clients for rendering or processing. For instance, in the *plot_normalized_price_and_sentiment* method with *for_web* enabled, the plot is saved as SVG and the content is extracted as a string for direct use in web HTML or as part of a web response. Similarly, the *analysis* method generates plots, converts them to SVG, and forms URLs for inclusion in web pages, while also preparing other statistical results in a web-friendly serialized format.

5.5 Session Management

Flask's session management capabilities are used to store user preferences and states across multiple views, enhancing personalized user interactions without sacrificing performance. Flask uses a client-side session storage mechanism, where the session data is stored in the user's browser as a cookie. During the initialization of the Flask app in app.py, a secret key is configured using *app.secret_key = os.urandom(24)* to cryptographically sign the session cookies, safeguarding against data tampering. When a user submits their preferences through the form on the *index.html* page, these choices are captured and stored in the session using code such as *session['days_back'] = request.form['days_back'].* This stored data is accessible across all routes within the application and persists until the session cookie expires or is explicitly cleared, thus facilitating a consistent and customized user experience. In the dashboard route, the application accesses these session variables to fetch relevant data, perform analysis, and tailor visualizations to the user's preferences, for example, using *days_back = int(session.get('days_back', 1))* to retrieve the *'days_back*' value from the session.

5.6 Caching Mechanism

The application uses the *flask_caching* module to add caching, which improves performance by saving the results of time-consuming tasks like fetching data and analyzing sentiment. This makes the app faster and eases the load on the server, ensuring it stays responsive when users revisit results.

The cache is set up using the Cache class from *flask_caching*. It is configured to use "SimpleCache," an in-memory cache that's good for single-process setups. The cache timeout is set to 86400 seconds (24 hours), meaning the cached data expires after one day unless it's updated. This configuration is defined in the app's settings (app.config).

Functions such as *fetch_news* and *perform_sentiment_analysis* are marked with the @cache.cached decorator, which automatically saves their outputs. The cache key for each function is created using helper functions (*make_news_cache_key* and *make_sentiment_analysis_cache_key*). These keys ensure each unique request has its own cache entry.

The cache keys are dynamically generated to reflect the specific parameters of each request, such as the number of days back or the category of news. This granularity ensures that users receive data that is both up-to-date and relevant to their specific query, without unnecessary data recomputation.

By caching the results of data-intensive operations, the application minimizes the number of times these operations need to be performed. This speeds up response times for end-users but also reduces the load on the server and external APIs. Since the app uses free tier API, as already discussed, reducing the frequency of API calls can also help avoid hitting usage limits or, in the case of paid APIs, reduce costs.

6 CODE MAINTENANCE

6.1 Github

The app uses Git as a version control system, enabling developers to track changes, revert to previous versions of the code, and manage multiple development branches efficiently. Git ensures that the development process is smooth, and changes can be integrated seamlessly. Each feature and bug fix is can be developed in its own branch, and merged into the main branch upon completion.

6.2 Error Handling and Logging

Error handling mechanisms ensure the application remains operational even when external APIs fail or return unexpected responses. Logging is extensively used to monitor application performance and debug issues in real-time.

6.3 Unit Testing

The app.config['TESTING'] setting enables test mode in Flask, simplifying test execution and environment configuration. It is implemented in the code so that when set to *True* it does not make API calls but use previous dataset that were called and stored. This allow for easy debugging as often the APIs have free tier with limit rate.

6.4 Update

As briefly mentioned earlier, this project is a template project which uses a specific set of free APIs. However, it is built with expansion in mind. Therefore, it can be easily updated to include further new API that might provide additional informations. Indeed, adding a modules for a

specific API is quite easy the only to be sure of is that the returned dataframe matches the format of what is needed for the further analysis.

7 RESULT

The home page of the app (fig 3) is easy to understand and user-friendly. It starts with a brief introduction and shows a Data Submission Form right on the front page, which is simple to use. This allows users to quickly and easily enter their information. If users need help, there's extra documentation they can refer to. The top of the page features a moving banner that shows cryptocurrency prices, much like those found in a stock trading room. Here the user can selected the time frequency and the category as shown in figure 4.

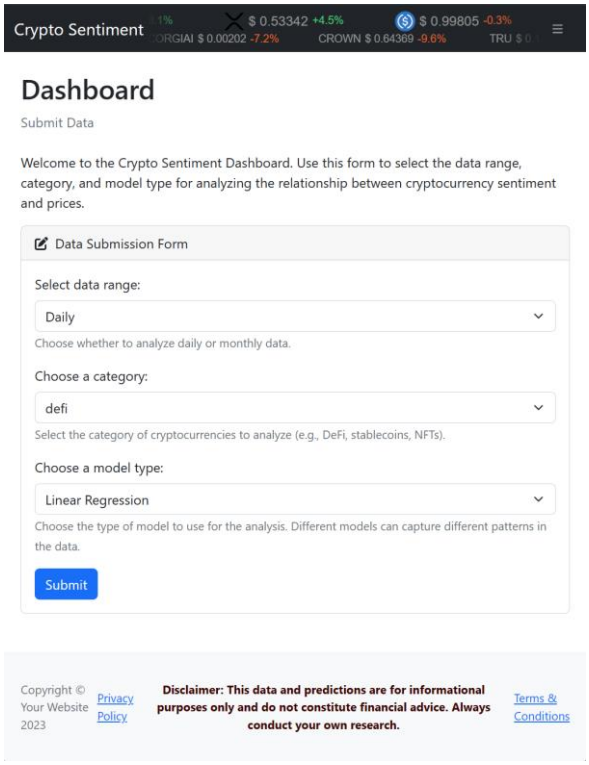

Fig. 3. : Main Page

Upon form submission, the user is taken to the dashboard page, which initially presents a plot of the price and sentiment analysis based on the selected time frequency and category (Figure 5). This visualization offers an immediate insight into the data trends within the chosen parameters.

This allows the user to have a first glance of how the data behaves in this current category and timestamp.

Further down, a scatter plot visualizes the relationship between price changes and lagged sentiment (Figure 6). Users can interact with this plot by selecting different lags to examine how the

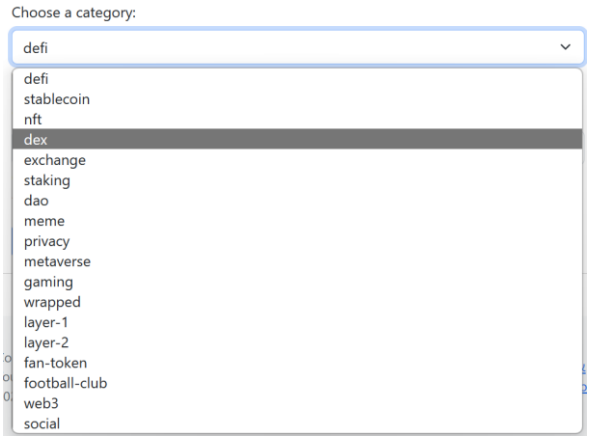

Fig 4. : Form Selection Example

data relationships vary with changes in lag, enhancing understanding of the impact of sentiment on price. The information are shown in a transparent manner so the user can judge by himself the reliability of the results.

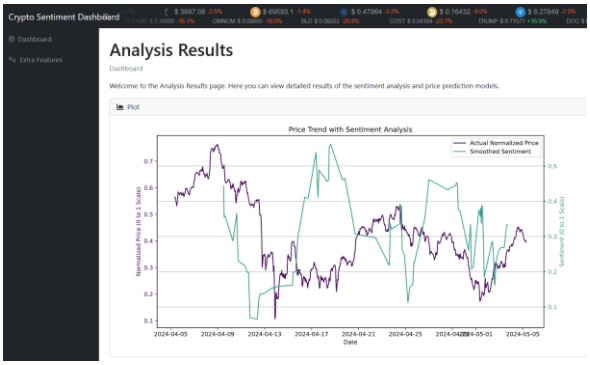

Fig 5. Dashboard

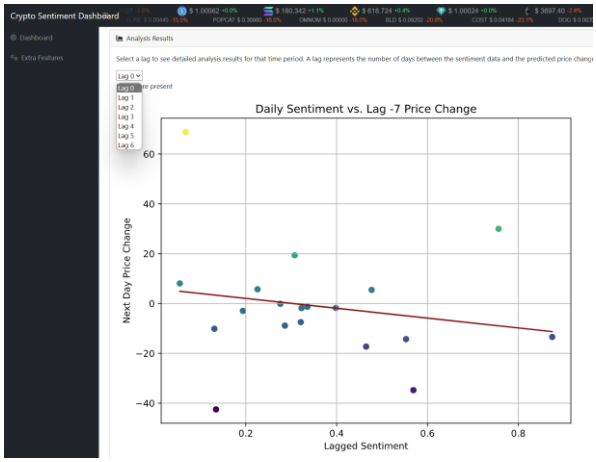

Fig 6. : Scatter Plot

Below this, the page displays several metrics that enrich the analysis, including correlation coefficients, average price changes on days with

high versus low sentiment, R-squared values, and Root Mean Square Error (RMSE), all illustrated in Figure 7. These metrics provide a comprehensive overview of the dataset's performance and its analytical reliability.

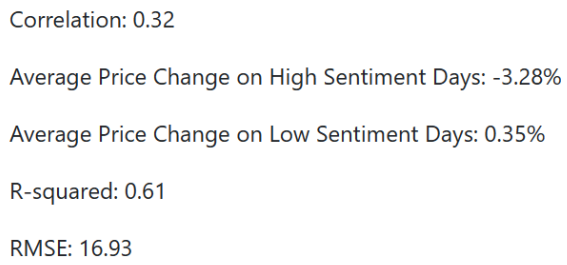

Fig 7. : Metrics

Further down, predictions specific to the chosen lag are shown, alongside an overall average prediction across all lags, offering forecasts based on the sentiment analysis (fig 8).

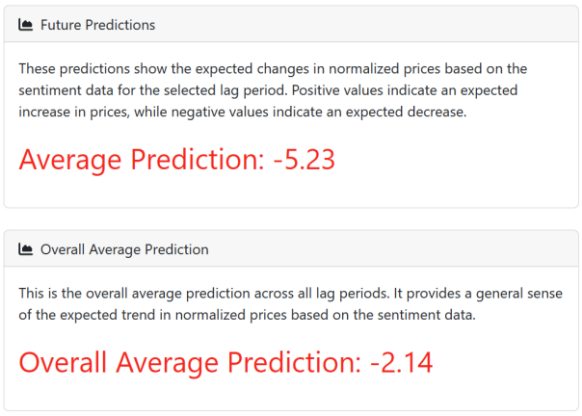

Fig 8. : Predictions

Descending further the user will have a glance on the ARIMA model implemented on the price data. This can serve as a benchmark fit a rather conservative model as even if the ARIMA is in auto mode it will almost always make a mean forecast as price data are very unpredictable and following a random walk. As shown in the figure 9, average predictions as well as confidence intervals are demonstrated there so the user know what to expect as a benchmark to compare the performance of the other metrics

Finally at the bottom of the page the data use is shown in a table format. As shown in figure 8 this further increase the transparency to see on what data the model worked on (fig 10).

Fig 9. : ARIMA

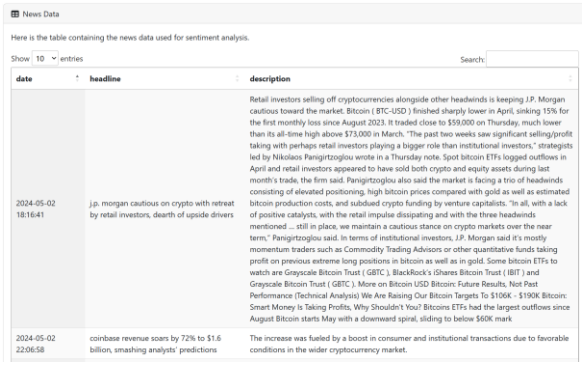

Fig 10. : News Table

8 CONCLUSION

The project describes the development of the web-based application designed to provide transparent, real-time sentiment analysis tailored to the cryptocurrency market. This platform offers a user-friendly interface that enables users to make informed decisions based on comprehensive market insights. The application leverages a modular design and a robust set of APIs to ensure data freshness and usability. It supports real-time data fetching and incorporates advanced prediction models to deliver dynamic insights crucial for users in the volatile cryptocurrency market.

A significant aspect of the application is its modular architecture, which simplifies ongoing maintenance and future updates. This design allows easy integration of new functionalities or improvements based on user feedback and changing market conditions. The report highlights the platform's capability to handle initial computations and unique, uncached queries slowly, which is a critical point for optimizing performance.

Testing the applications reveals the use of Flask, a Python-based micro web framework, which facilitates the handling of HTTP requests and serves dynamic content effectively. This choice supports the application's need for real-time data processing and response, making it suitable for

the fast-paced environment of cryptocurrency trading.

The application can however experience slow response times during first-time computations or when dealing with unique queries that haven't been cached.

Looking forward, there are several areas for potential development. One possibility is transforming the current modules into microservices, enhancing scalability and flexibility. This change would support separate deployment and better resource management, aligning the platform with modern, cloud-native practices to boost performance and reliability. Additionally, improving caching strategies could reduce load times, and introducing asynchronous processing would allow tasks to run in the background, thereby speeding up user interactions [4].

9 APPENDIX

9.1 References

[1] L. Chappex, "Interview with Richard Peterson, CEO of MarketPsych," Swissquote, [Online]. Available: https://en.swissquote.lu/international-investing/investing-ideas/interview-richard-peterson-ceo-marketpsych. [Accessed: May 21, 2024].

[2] Johns Hopkins University, "Cloud-native Architecture and Microservices - 605.702," [Online]. Available: https://ep.jhu.edu/courses/605702-cloud-native-architecture-and-microservices/.

[3] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching strategy for Web application – a systematic literature review," International Journal of Web Information Systems, [Online]. Available: https://doi.org/10.1108/IJWIS-06-2020-0032.

[4] S. Hauck, "Asynchronous Design Methodologies: An Overview," University of Washington, [Online]. Available: https://people.ece.uw.edu/hauck/publications/AsynchArt.pdf

9.2 Helper Tools

Chat-GPT

Co-pilot