Project Reflection: Crypto Market Sentiment Chatbot Assistant

How We Selected Our Datasets

For this project, we set out to build a chatbot that could help users understand the emotional and financial context of the cryptocurrency market by combining two types of data: public sentiment and historical price movements. We were inspired by the idea that market behavior, particularly in crypto, is heavily influenced by emotion. So we wanted our chatbot to help users interpret whether a given day showed fear or greed in the market — and whether those feelings aligned with how prices actually moved.

We selected the Fear and Greed Index from alternative.me as our public API source. It offered daily sentiment scores derived from multiple indicators like volatility, trading volume, and social media activity. The score was easy to understand and extremely relevant for understanding short-term market psychology.

For our local dataset, we pulled minute-level historical Bitcoin OHLCV data (Open, High, Low, Close, Volume) from Kaggle. This gave us raw, high-resolution price data for cryptocurrencies. We chose this dataset because it allowed us to aggregate daily trends ourselves and fully control the data transformation process. It also paired well with the Fear & Greed Index, which is released daily — enabling us to correlate sentiment with price action.

Challenges and Problems Faced

One of the first challenges we encountered was aligning timestamp granularity. Our price data was captured by the minute, while our sentiment data was daily. To resolve this, we transformed timestamps using .dt.floor('d') and performed daily aggregation of OHLCV metrics (first open, last close, min low, max high, volume sum). Once floored, both datasets could be merged on a shared date column.

One of the biggest challenges we encountered was related to data alignment and granularity. The Fear and Greed Index was structured with one entry per day, whereas our crypto price data had timestamps down to the minute. To resolve this, we implemented a transformation step in our ETL pipeline to floor timestamps to daily intervals using .dt.floor('d'). This allowed us to reliably merge both datasets on a common "date" column.

Integrating the Google Gemini LLM took some trial and error. We needed to write a clear prompt and convert our DataFrame to a string so Gemini could understand it. Once that was set up, the model was able to give helpful answers based on our processed data.

Lastly, deploying the chatbot on Google Cloud Platform presented some learning curves, especially when it came to exposing the Flask app publicly via a VM instance. Troubleshooting firewalls, port access, and deployment configs on GCP required patience, but ultimately helped us gain practical experience with cloud infrastructure.

Key Learnings and Discoveries

This project taught us how to bring together multiple areas of development — including data processing, backend design, cloud deployment, and AI integration — into one cohesive system. Through our work on the ETL pipeline in etl.py, we learned how to extract data from both an API and a local CSV file, transform and clean it into a usable format, and save it for further use. We also learned how to deal with real-world data issues like timestamp mismatches and how to aggregate minute-level price data into daily summaries.

Building the backend with Flask gave us experience in designing an API that could receive user input, call other Python modules, and return a structured JSON response. It also gave us a better understanding of how to build error handling into endpoints and prepare them for deployment. Hosting the chatbot on a Google Cloud Platform virtual machine was especially valuable, as it introduced us to the process of configuring cloud infrastructure, opening firewall ports, and making a web app publicly accessible.

One of the most interesting parts of the project was integrating the Gemini LLM. We created a clear prompt and passed our dataset to the model in string format, along with the user's question. Once everything was connected, we were able to receive clear, context-aware responses based on the latest data.

What We Would Add With More Time

With more time, we would have explored a few additional features to enhance the functionality and usefulness of the chatbot. One idea was to integrate the project with Discord, so that users could interact with the bot in real time through a familiar messaging platform. This would expand accessibility and make the bot feel more interactive and available on demand.

Another improvement we considered was adding visual outputs, such as simple charts showing sentiment trends or price movements over time. This would help users better understand patterns instead of relying solely on text-based responses. Additionally, we thought about expanding the scope of the chatbot by allowing users to select different cryptocurrencies, compare sentiment across coins, explore multi-day trend analysis, incorporating lagged sentiment and show users how fear or greed on a given day might correlate with price action the following day. This could potentially help users identify short-term predictive patterns. Another feature we considered was user customization, allowing them to ask about specific date ranges or view mini time-series visualizations.

Overall, there are many directions we could take the project in the future, and the codebase we built is flexible enough to support continued development and new features.