**305 Crypto Forecaster: The Complete Guide for Dummies (Mac Edition)**

Welcome! This guide will walk you through every step of setting up the 305 Crypto Forecast App on your Mac, from installing the necessary tools to deploying your own live version on the internet. No programming experience is required. We will explain everything as we go.

**What is the 305 Crypto Forecast App?**

At its heart, the **305 Crypto Forecast App** is your personal, automated crypto analyst. Think of it as a sophisticated tool that does four key things every single day:

1. **Gathers Data:** It connects to multiple professional sources online to collect a massive amount of data: historical prices, current market sentiment from news articles, on-chain metrics (what's happening on the blockchain), and complex derivatives data (what professional traders are doing).
2. **Analyzes and Predicts:** It uses this data to run powerful machine learning models that forecast future prices for major cryptocurrencies like Bitcoin, Ethereum, and XRP.
3. **Writes a Report:** It feeds all this information to a powerful AI (GPT-4) and instructs it to act like an expert financial analyst. The AI then writes a detailed, human-readable report with bullish signals, bearish signals, and a final hypothesis on where the market is headed.
4. **Creates a Dashboard:** Finally, it presents all this information—the charts, the forecasts, and the AI's written report—on a clean, interactive web dashboard that you can access from any device.

By the end of this guide, you will have your own private, running version of this powerful tool.

**Chapter 1: Getting Your Tools Ready**

Before we can build anything, we need to install the necessary tools on your Mac. We'll use a program called **Homebrew** to make this easy. Think of Homebrew as an "App Store for developers"—it lets you install powerful tools with simple commands.

**Step 1: Install Homebrew**

First, we need to open the **Terminal**. The Terminal is an application on your Mac that lets you give it direct text commands.

1. Press **Cmd + Space** to open Spotlight Search.
2. Type Terminal and press **Enter**.
3. A white or black window will appear. Copy the entire command below, paste it into the Terminal, and press **Enter**.

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

The installation will start. It might ask for your computer's password and ask you to press **Enter** to continue. This process can take several minutes.

**Step 2: Install Git**

**Git** is like a "save button for your code's history." It's a system that tracks every change you make, allowing you to go back to previous versions if you make a mistake. It's essential for managing any software project.

In the same Terminal window, run this command:

brew install git

**Step 3: Install a Code Editor (Visual Studio Code)**

While you can write code in any text editor, a specialized code editor makes the process much easier with features like syntax highlighting and error checking. We recommend **Visual Studio Code (VS Code)**, which is free and the most popular choice.

* **Download and install VS Code from the official website:** <https://code.visualstudio.com/>

**Step 4: Install Docker Desktop**

**Docker** is a program that creates "mini-computers" called **containers** on your Mac. Our app will run inside these containers. This is incredibly useful because it guarantees that the app works in the exact same way on your computer as it does online, preventing "it works on my machine" problems.

* **Download and install Docker Desktop from the official website:** <https://www.docker.com/products/docker-desktop/>

After installing, open the Docker Desktop application and let it start up. You'll see a whale icon in your top menu bar when it's ready.

**Chapter 2: Setting Up the Project**

Now that our tools are ready, let's set up the project folder and get the application's code.

**Step 1: Create the Project Folder**

In your Terminal, we'll create a folder for our project and move into it. Run these two commands one by one:

mkdir clean-crypto

cd clean-crypto

**Step 2: Get the Application Code**

We will use Git to download (or "clone") the application's code from its central repository on GitHub.

git clone https://github.com/mfaleni/305-Crypto-Forecaster.git .

*(Note the period . at the end—it's important!)*

This command copies all the necessary files into your clean-crypto folder.

**Step 3: Create the Secrets File (.env)**

The application needs several secret API keys to connect to its data sources. We will store these in a special file called .env that is kept private on your computer and is ignored by Git.

1. In your Terminal, run this command to create and open the file in a simple text editor called **Nano**:
2. nano .env
3. Your Terminal will now look like a text editor. Copy the entire block of text below and paste it into the Nano editor.
4. # API KEYS
5. OPENAI\_API\_KEY=YOUR\_KEY\_HERE
6. NEWS\_API\_KEY=YOUR\_KEY\_HERE
7. COINGECKO\_API\_KEY=YOUR\_KEY\_HERE
8. LUNARCRUSH\_API\_KEY=YOUR\_KEY\_HERE
9. SANTIMENT\_API\_KEY=YOUR\_KEY\_HERE
10. COINGLASS\_API\_KEY=YOUR\_KEY\_HERE
11. # LOCAL DOCKER DATABASE CREDENTIALS
12. DB\_NAME=crypto\_db
13. DB\_USER=crypto\_user
14. DB\_PASSWORD=supersecretpassword
15. DB\_HOST=db
16. DB\_PORT=5432
17. Now, you need to replace each YOUR\_KEY\_HERE with the actual API key from each service. You will need to sign up for a free account on each of these websites to get your key:
    * **OpenAI:** <https://platform.openai.com/api-keys>
    * **NewsAPI:** <https://newsapi.org/>
    * **CoinGecko:** <https://www.coingecko.com/en/api>
    * **LunarCrush:** <https://lunarcrush.com/developers/api>
    * **Santiment:** <https://app.santiment.net/account>
    * **CoinGlass:** [https://coinglass.github.io/API-docs/](https://www.google.com/search?q=https://coinglass.github.io/API-docs/)
18. Once you've replaced all the keys, save and exit Nano by pressing **Ctrl + X**, then **Y**, then **Enter**.

**Chapter 3: Running the App Locally**

With the project set up, we can now run the entire application on your local machine using Docker.

**Step 1: Start the Application**

In your Terminal, from inside the clean-crypto folder, run this single command. This tells Docker to build your application, start the database, and run the main script.

docker-compose up --build

This process will take a long time, especially the first time you run it, as it needs to download and install all the dependencies. You will see a huge amount of text scrolling in your Terminal. This is normal!

**Step 2: View Your Dashboard**

Once the process is complete, you will see messages from Streamlit near the end of the log, like this:

You can now view your Streamlit app in your browser.

URL: http://0.0.0.0:8501

Open your web browser (like Chrome or Safari) and go to the following address:

**http://localhost:8501**

You should now see your live, fully functional 305 Crypto Forecast dashboard, running privately on your own computer!

**Chapter 4: Deploying to the Internet (Render)**

Running the app locally is great for testing, but if you want to be able to access your dashboard from anywhere or share it with others, you need to deploy it online. We will use a free service called **Render**.

**Step 1: Create a Render Account**

* Go to <https://render.com/> and sign up for a free account. It's best to sign up using your GitHub account to make connecting your code easier.

**Step 2: Create the Online Database**

1. From the main Render dashboard, click **"New +"** and select **"PostgreSQL"**.
2. Give your database a unique name (e.g., 305-crypto-db).
3. Ensure you select the **"Free"** plan.
4. Click **"Create Database"**.
5. Wait for the database to be created. Once it's ready, go to its "Info" page and copy the **"Internal Connection URL"**. We will need this in the next step.

**Step 3: Create the Web Application**

1. From the main Render dashboard, click **"New +"** and select **"Web Service"**.
2. Connect your GitHub account and select your 305-Crypto-Forecaster repository.
3. Give your application a unique name (e.g., my-crypto-forecaster).
4. Under **"Root Directory"**, make sure it's set to the base of your project.
5. Under **"Start Command"**, enter: bash start.sh
6. Ensure you select the **"Free"** instance type.
7. Click the **"Advanced"** button.
8. Click **"Add Environment Variable"**. You will now add every single key from your local .env file here.
   * Add a variable named DATABASE\_URL and paste the Internal Connection URL you copied from your Render database.
   * Add variables for OPENAI\_API\_KEY, NEWS\_API\_KEY, etc., and paste in your secret keys.
9. Click **"Create Web Service"**.

Render will now automatically deploy your application. The first time it runs, it will connect to your new Render database, create all the necessary tables, and populate it with the first day's analysis. Once the deployment is "Live," you can click the URL at the top of the page to see your dashboard running on the internet!

Congratulations! You have successfully built and deployed the 305 Crypto Forecast App.

305 Crypto Forecaster Phase 2

Of course. Here is a complete, exportable summary of the entire 305 Crypto Forecast App, including its architecture, all final scripts, and a guide for local testing and deployment.

You can copy and paste this entire response into a document for your records.

**305 Crypto Forecast App: Project Summary**

The 305 Crypto Forecast App is a comprehensive, multi-layered application designed to fetch, analyze, and forecast cryptocurrency data. It leverages a suite of professional data sources, machine learning models, and a powerful AI analyst to produce detailed daily reports, which are then displayed on an interactive web dashboard. The entire application is containerized with Docker for consistent development and deployment.

**Application Flow**

The application runs as an automated sequence:

1. **Data Fetching:** Gathers historical prices, on-chain metrics, social sentiment, and derivatives data from multiple APIs.
2. **Analysis & Forecasting:** Calculates technical indicators and runs Prophet and LSTM models to generate price forecasts.
3. **AI Synthesis:** Sends a comprehensive data briefing to an AI analyst (GPT-4) to generate a qualitative report, including bullish/bearish cases and a final hypothesis.
4. **Data Persistence:** Saves all final results to a PostgreSQL database.
5. **Visualization:** Launches a Streamlit web dashboard to display all the data and analysis.

**## Snapshot Contents**

I have saved the following files as our official, stable baseline:

**Application Code**

* daily\_runner.py (The main orchestrator)
* data\_utils.py (The complete data pipeline)
* db\_utils.py (The final database schema and functions)
* analyst.py (The strong AI analyst with the preferred prompt)
* sentiment.py (Sentiment analysis module)
* forecasting.py (Prophet and LSTM models)
* dashboard\_app.py (The fully-featured dashboard)
* chart\_analyst.py (Dynamic chart interpretation)
* test\_coinglass.py
* test\_coingecko.py
* test\_santiment.py

**Configuration & Environment**

* docker-compose.yml
* Dockerfile
* start.sh
* .env (Structure and required keys)
* .gitignore
* .dockerignore
* requirements.txt

**Data Structure & Examples**

* forecast\_results.csv
* BTC-USD\_data.csv
* ETH-USD\_data.csv
* XRP-USD\_data.csv

**Core Components and File Structure**

Here are the final, definitive scripts that make up your application.

**1. Configuration Files**

**.env (Template)**

*Stores secret keys and configuration variables.*

# API KEYS

OPENAI\_API\_KEY=sk-proj-MTFhvOBWHFVaq-4BQBjIRbJ8HOSd86gSfjJFQJMob-aMqsdZwWBoN7DPtcOz4LV\_Wjtx5e5CH3T3BlbkFJylBQBbwxRRisFH2NCLhX3-p-3TxTWnE223eUg-VLlrBlBJJXJ4\_xap\_9COg11Q6DV\_54F5Qw8A

NEWS\_API\_KEY=ef06174b0ea443f4b2b18368328540cf

# DATABASE\_URL=postgresql+psycop://crypto\_forecast\_db\_f894\_user:x6eG4FJniUTJvH4cJJXADpJjmWBw7EHk@dpg-d2du5pumcj7s73dbkdv0-a.ohio-postgres.render.com/crypto\_forecast\_db\_f894

COINGECKO\_API\_KEY=CG-cAteQjGtbCKzgEfcsgVRjz5G

LUNARCRUSH\_API\_KEY=g9e5oyvvnr8h0kt0gdrlq2jjn7dme7cl8vxkhu3zf9

SANTIMENT\_API\_KEY=jqseson62geevsyl\_u7i3s44gk7xvaass

COINGLASS\_API\_KEY=acca976f86f842019b3f6d766034f529

# PostgreSQL Database Credentials

DB\_NAME=crypto\_db

DB\_USER=crypto\_user

DB\_PASSWORD=supersecretpassword

DB\_HOST=db

DB\_PORT=5432

**.gitignore**

*Tells Git which files to ignore.*

# Python artifacts

\_\_pycache\_\_/

\*.pyc

# Virtual Environment

venv/

# Environment variables

.env

# Data files generated by the script

data/

forecast\_results.csv

# macOS specific files

.DS\_Store

# IDE / Editor specific files

.vscode/

.idea/

**requirements.txt**

*Lists all Python dependencies.*

absl-py==2.3.1

altair==5.5.0

annotated-types==0.7.0

anyio==4.9.0

astunparse==1.6.3

attrs==25.3.0

beautifulsoup4==4.13.4

blinker==1.9.0

cachetools==6.1.0

certifi==2025.7.14

cffi==1.17.1

charset-normalizer==3.4.2

click==8.2.1

cmdstanpy==1.2.5

contourpy==1.3.2

curl\_cffi==0.12.0

cycler==0.12.1

distro==1.9.0

exceptiongroup==1.3.0

flatbuffers==25.2.10

fonttools==4.59.0

frozendict==2.4.6

gast==0.6.0

gitdb==4.0.12

GitPython==3.1.44

google-pasta==0.2.0

grpcio==1.73.1

h11==0.16.0

h5py==3.14.0

holidays==0.76

httpcore==1.0.9

httpx==0.28.1

idna==3.10

importlib\_resources==6.5.2

Jinja2==3.1.6

jiter==0.10.0

joblib==1.5.1

jsonschema==4.24.1

jsonschema-specifications==2025.4.1

keras==3.10.0

kiwisolver==1.4.8

libclang==18.1.1

Markdown==3.8.2

markdown-it-py==3.0.0

MarkupSafe==3.0.2

matplotlib==3.10.3

mdurl==0.1.2

ml-dtypes==0.3.2

multitasking==0.0.12

namex==0.1.0

narwhals==1.47.0

numpy==1.26.4

openai==1.97.0

opt\_einsum==3.4.0

optree==0.16.0

packaging==25.0

pandas==2.3.1

peewee==3.18.2

pillow==11.3.0

platformdirs==4.3.8

plotly==6.2.0

prophet==1.1.7

protobuf==4.25.8

pyarrow==21.0.0

pycparser==2.22

pydantic==2.11.7

pydantic\_core==2.33.2

pydeck==0.9.1

Pygments==2.19.2

pyparsing==3.2.3

python-dateutil==2.9.0.post0

python-dotenv==1.1.1

pytz==2025.2

referencing==0.36.2

requests==2.32.4

rich==14.0.0

rpds-py

scikit-learn==1.7.0

scipy==1.15.3

six==1.17.0

smmap==5.0.2

sniffio==1.3.1

soupsieve==2.7

stanio

streamlit==1.47.0

ta==0.11.0

tenacity==9.1.2

tensorboard==2.16.2

tensorboard-data-server==0.7.2

tensorflow-cpu

tensorflow-io-gcs-filesystem==0.37.1

termcolor==3.1.0

threadpoolctl==3.6.0

toml==0.10.2

tornado==6.5.1

tqdm==4.67.1

typing-inspection==0.4.1

typing\_extensions==4.14.1

tzdata==2025.2

urllib3==2.5.0

websockets==15.0.1

Werkzeug==3.1.3

wrapt==1.17.2

yfinance==0.2.65

SQLAlchemy

psycopg[binary]

psycopg2-binary

lunarcrush

**2. Docker & Orchestration**

**Dockerfile**

*Blueprint for building the application's Docker container.*

Dockerfile

# Use an official Python runtime as a parent image

FROM python:3.10-slim

# Set the working directory in the container

WORKDIR /app

# Set environment variables to prevent Python from writing .pyc files

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

# Install system dependencies that some Python packages might need

RUN apt-get update && apt-get install -y \

build-essential \

curl \

&& rm -rf /var/lib/apt/lists/\*

# Copy the requirements file into the container

COPY requirements.txt .

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application's code into the container

COPY . .

# Make the startup script executable

RUN chmod +x ./start.sh

# Command to run when the container launches

CMD ["bash", "./start.sh"]

**docker-compose.yml**

*Defines and runs the multi-container Docker application.*

YAML

# Use version 3.8 of the Docker Compose file format for modern features.

version: "3.8"

# Define the services (containers) that make up your application.

services:

# 1. The Database Service

db:

image: postgres:14-alpine # Use a specific, lightweight version of PostgreSQL.

container\_name: crypto\_db\_container

environment:

# These variables are passed from your .env file to configure the database.

POSTGRES\_USER: ${DB\_USER}

POSTGRES\_PASSWORD: ${DB\_PASSWORD}

POSTGRES\_DB: ${DB\_NAME}

ports:

# Map port 5432 on your Mac to port 5432 inside the container.

# This lets you connect to the database from your local machine.

- "5432:5432"

volumes:

# This creates a persistent storage volume named 'postgres\_data'.

# It ensures your database data is saved even if the container is removed.

- postgres\_data:/var/lib/postgresql/data

# 2. Your Application Service

app:

build: . # Build the app's Docker image from the Dockerfile in the current directory.

image: crypto-forecaster

container\_name: crypto\_forecaster\_app

depends\_on:

# This tells Docker to start the 'db' service BEFORE starting the 'app' service.

# It ensures the database is ready to accept connections.

- db

env\_file:

# Load the environment variables from your .env file into the app container.

- .env

ports:

# Map port 8501 on your computer (localhost) to port 8501 inside the container.

- "8501:8501"

dns:

# Add a public DNS server to resolve network issues inside the container.

- 8.8.8.8

# Define the persistent volume used by the 'db' service.

volumes:

postgres\_data:

**start.sh**

*The startup script for the app container.*

#!/bin/bash

# Exit immediately if a command exits with a non-zero status.

set -e

# First, run the data generation script

echo "--- Running daily analysis script ---"

python daily\_runner.py

# Then, start the streamlit dashboard on all network interfaces

echo "--- Starting Streamlit dashboard ---"

streamlit run dashboard\_app.py --server.port 8501 --server.address 0.0.0.0

**.dockerignore**

# Git files

.git

.gitignore

# Docker files

Dockerfile

.dockerignore

docker-compose.yml

# Virtual environment

venv/

# Local secrets and data

.env

data/

forecast\_results.csv

# macOS specific files

.DS\_Store

# Python cache

\_\_pycache\_\_/

\*.pyc

**3. Application Logic**

**data\_utils.py**

*Fetches all data from all external APIs.*

import pandas as pd

import yfinance as yf

import requests

import os

from ta.momentum import RSIIndicator, StochasticOscillator

from ta.trend import MACD, SMAIndicator, EMAIndicator, IchimokuIndicator

from ta.volatility import BollingerBands

from ta.volume import OnBalanceVolumeIndicator

import numpy as np

from datetime import datetime, timedelta

import time

# --- API HELPER FUNCTIONS ---

def fetch\_coinglass\_data(symbol: str) -> dict:

"""Fetches advanced futures data from CoinGlass using the definitive, proven endpoint."""

print(f" [INFO] Fetching futures data for {symbol} from CoinGlass (Hobbyist Tier)...")

coinglass\_api\_key = os.getenv("COINGLASS\_API\_KEY")

if not coinglass\_api\_key:

print(" [WARN] COINGLASS\_API\_KEY not found. Skipping.")

return {}

headers = {

'accept': 'application/json',

'coinglassSecret': coinglass\_api\_key

}

api\_symbol = symbol.replace("-USD", "")

data = {'funding\_rate': 0.0, 'open\_interest': 0.0, 'long\_short\_ratio': 0.0, 'futures\_volume\_24h': 0.0}

try:

# Definitive v2 endpoint that provides all data in one call

url = f"https://open-api.coinglass.com/public/v2/perpetual\_market?ex=Binance&symbol={api\_symbol}"

response = requests.get(url, headers=headers)

if response.ok and response.json().get('data'):

# The response contains data for all exchanges; we need to filter for the specific symbol on Binance

market\_data\_list = response.json()['data'].get(api\_symbol, [])

binance\_data = next((item for item in market\_data\_list if item.get("exchangeName") == "Binance"), None)

if binance\_data:

data['funding\_rate'] = binance\_data.get('rate', 0.0) \* 100

data['open\_interest'] = binance\_data.get('openInterest', 0.0)

data['futures\_volume\_24h'] = binance\_data.get('totalVolUsd', 0.0)

# Calculate the Long/Short Ratio

long\_rate = binance\_data.get('longRate', 0.0)

short\_rate = binance\_data.get('shortRate', 1.0) # Default to 1 to avoid division by zero

data['long\_short\_ratio'] = long\_rate / short\_rate if short\_rate > 0 else 0

print(" [SUCCESS] Futures data fetched.")

else:

print(f" [WARN] Binance data for symbol {api\_symbol} not found in CoinGlass response.")

else:

print(f" [WARN] CoinGlass request failed. Status: {response.status\_code}, Response: {response.text[:100]}")

return data

except Exception as e:

print(f" [ERROR] A critical error occurred while fetching CoinGlass data: {e}")

return {}

def fetch\_cryptoquant\_data(symbol: str) -> dict:

"""Placeholder for fetching advanced on-chain data like Exchange Supply Ratio (ESR)."""

print(f" [INFO] Fetching advanced on-chain data for {symbol} from CryptoQuant...")

# To implement: Add CRYPTOQUANT\_API\_KEY to .env and make the API call here.

print(" [SUCCESS] CryptoQuant data fetched (placeholder).")

return {'exchange\_supply\_ratio': 0.0}

def fetch\_santiment\_data(slug: str) -> dict:

"""Fetches on-chain/social data with robust handling for null values."""

print(f" [INFO] Fetching on-chain/social data for {slug} from Santiment...")

api\_key = os.getenv("SANTIMENT\_API\_KEY")

if not api\_key:

print(" [WARN] SANTIMENT\_API\_KEY not found. Skipping.")

return {}

query = f"""

query {{

mvrv: getMetric(metric: "mvrv\_usd") {{ timeseriesData(slug: "{slug}", from: "utc\_now-2d", to: "utc\_now", interval: "1d") {{ value }} }}

social\_dominance: getMetric(metric: "social\_dominance\_total") {{ timeseriesData(slug: "{slug}", from: "utc\_now-2d", to: "utc\_now", interval: "1d") {{ value }} }}

daa: getMetric(metric: "daily\_active\_addresses") {{ timeseriesData(slug: "{slug}", from: "utc\_now-2d", to: "utc\_now", interval: "1d") {{ value }} }}

}}

"""

try:

response = requests.post('https://api.santiment.net/graphql', json={'query': query}, headers={'Authorization': f'Apikey {api\_key}'})

response.raise\_for\_status()

json\_data = response.json()

mvrv\_data = json\_data.get('data', {}).get('mvrv', {}).get('timeseriesData', [])

social\_data = json\_data.get('data', {}).get('social\_dominance', {}).get('timeseriesData', [])

daa\_data = json\_data.get('data', {}).get('daa', {}).get('timeseriesData', [])

metrics = {

'mvrv\_usd': mvrv\_data[-1]['value'] if mvrv\_data and mvrv\_data[-1] else 0.0,

'social\_dominance': social\_data[-1]['value'] if social\_data and social\_data[-1] else 0.0,

'daily\_active\_addresses': daa\_data[-1]['value'] if daa\_data and daa\_data[-1] else 0.0

}

print(" [SUCCESS] Santiment data fetched.")

return metrics

except Exception as e:

print(f" [WARN] Could not fetch Santiment data: {e}")

return {}

def fetch\_lunarcrush\_data(symbol: str) -> dict:

"""Fetches social intelligence for a given symbol directly from the LunarCrush API v4."""

print(f" [INFO] Fetching social intelligence for {symbol} from LunarCrush...")

api\_key = os.getenv("LUNARCRUSH\_API\_KEY")

if not api\_key:

print(" [WARN] LUNARCRUSH\_API\_KEY not found. Skipping.")

return {}

api\_symbol = symbol.replace("-USD", "")

url = f"https://lunarcrush.com/api4/public/coins/{api\_symbol}/v1"

headers = {'Authorization': f'Bearer {api\_key}'}

try:

response = requests.get(url, headers=headers)

response.raise\_for\_status()

data = response.json().get('data', {})

metrics = {

'galaxy\_score': data.get('galaxy\_score', 0.0),

'alt\_rank': data.get('alt\_rank', 0)

}

print(" [SUCCESS] LunarCrush data fetched.")

return metrics

except Exception as e:

print(f" [WARN] Could not fetch LunarCrush data: {e}")

return {}

def fetch\_coingecko\_data(coin\_id: str) -> dict:

"""Fetches fundamental and market data from the CoinGecko API."""

print(f" [INFO] Fetching CoinGecko data for {coin\_id}...")

api\_key = os.getenv("COINGECKO\_API\_KEY")

url = f"https://api.coingecko.com/api/v3/coins/{coin\_id}"

params = {'x\_cg\_demo\_api\_key': api\_key}

try:

response = requests.get(url, params=params)

response.raise\_for\_status()

data = response.json()

metrics = {

'market\_cap\_rank': data.get('market\_cap\_rank', 0),

'ath\_usd': data.get('market\_data', {}).get('ath', {}).get('usd', 0),

'total\_volume': data.get('market\_data', {}).get('total\_volume', {}).get('usd', 0),

'circulating\_supply': data.get('market\_data', {}).get('circulating\_supply', 0),

'community\_score': data.get('community\_score', 0),

'developer\_score': data.get('developer\_score', 0),

'sentiment\_up\_percentage': data.get('sentiment\_votes\_up\_percentage', 0)

}

print(" [SUCCESS] CoinGecko data fetched.")

return metrics

except requests.exceptions.RequestException as e:

print(f" [WARN] Could not fetch CoinGecko data for {coin\_id}: {e}")

return {}

def fetch\_data(coin: str) -> pd.DataFrame:

"""

Fetches historical data, calculates technical indicators, and enriches

it with data from all integrated professional sources.

"""

coingecko\_map = {"BTC-USD": "bitcoin", "ETH-USD": "ethereum", "XRP-USD": "ripple"}

santiment\_slug = coingecko\_map.get(coin)

print(f" [INFO] Fetching 180 days of historical data for {coin}...")

try:

df = yf.download(tickers=coin, period="180d", interval="1d", progress=False, auto\_adjust=False)

if df.empty: return pd.DataFrame()

if isinstance(df.columns, pd.MultiIndex): df.columns = df.columns.get\_level\_values(0)

print(" [INFO] Calculating technical indicators...")

df['SMA'] = SMAIndicator(close=df['Close'], window=20).sma\_indicator()

df['EMA'] = EMAIndicator(close=df['Close'], window=20).ema\_indicator()

df['RSI'] = RSIIndicator(close=df['Close']).rsi()

macd = MACD(close=df['Close'])

df['MACD'] = macd.macd(); df['MACD\_Signal'] = macd.macd\_signal()

bollinger = BollingerBands(close=df['Close'], window=20, window\_dev=2)

df['BB\_High'] = bollinger.bollinger\_hband(); df['BB\_Low'] = bollinger.bollinger\_lband()

stoch = StochasticOscillator(high=df['High'], low=df['Low'], close=df['Close'])

df['Stoch\_k'] = stoch.stoch(); df['Stoch\_d'] = stoch.stoch\_signal()

df['OBV'] = OnBalanceVolumeIndicator(close=df['Close'], volume=df['Volume']).on\_balance\_volume()

ichimoku = IchimokuIndicator(high=df['High'], low=df['Low'])

df['Ichimoku\_a'] = ichimoku.ichimoku\_a(); df['Ichimoku\_b'] = ichimoku.ichimoku\_b()

cg\_data = fetch\_coingecko\_data(santiment\_slug)

futures\_data = fetch\_coinglass\_data(coin)

santiment\_data = fetch\_santiment\_data(santiment\_slug)

lunar\_data = fetch\_lunarcrush\_data(coin)

cryptoquant\_data = fetch\_cryptoquant\_data(coin)

df['Market\_Cap\_Rank'] = cg\_data.get('market\_cap\_rank', 0)

df['All\_Time\_High\_Real'] = cg\_data.get('ath\_usd', 0.0)

df['Transaction\_Volume\_24h'] = cg\_data.get('total\_volume', 0.0)

df['Circulating\_Supply'] = cg\_data.get('circulating\_supply', 0.0)

df['Community\_Score'] = cg\_data.get('community\_score', 0.0)

df['Developer\_Score'] = cg\_data.get('developer\_score', 0.0)

df['Sentiment\_Up\_Percentage'] = cg\_data.get('sentiment\_up\_percentage', 0.0)

df['Funding\_Rate'] = futures\_data.get('funding\_rate', 0.0)

df['Open\_Interest'] = futures\_data.get('open\_interest', 0.0)

df['Long\_Short\_Ratio'] = futures\_data.get('long\_short\_ratio', 0.0)

df['Futures\_Volume\_24h'] = futures\_data.get('futures\_volume\_24h', 0.0)

df['MVRV\_Ratio'] = santiment\_data.get('mvrv\_usd', 0.0)

df['Social\_Dominance'] = santiment\_data.get('social\_dominance', 0.0)

df['Daily\_Active\_Addresses'] = santiment\_data.get('daily\_active\_addresses', 0.0)

df['Galaxy\_Score'] = lunar\_data.get('galaxy\_score', 0.0)

df['Alt\_Rank'] = lunar\_data.get('alt\_rank', 0)

df['Exchange\_Supply\_Ratio'] = cryptoquant\_data.get('exchange\_supply\_ratio', 0.0)

df['Exchange\_Net\_Flow'] = 0.0

df.dropna(inplace=True)

print(f" [SUCCESS] Data processing complete for {coin}.")

return df

except Exception as e:

print(f" [ERROR] An error occurred in fetch\_data for {coin}: {e}")

return pd.DataFrame()

**db\_utils.py**

*Manages all database interactions.*

import os

import pandas as pd

from sqlalchemy import create\_engine, text, inspect, Table, Column, MetaData, Integer, String, Float, DateTime

# --- Intelligent Database Connection Logic ---

DATABASE\_URL = os.getenv("DATABASE\_URL")

engine = None

if DATABASE\_URL:

print(" [INFO] Connecting to cloud database (Render)...")

engine = create\_engine(DATABASE\_URL)

else:

print(" [INFO] Connecting to local database (Docker)...")

db\_user = os.getenv("DB\_USER")

db\_pass = os.getenv("DB\_PASSWORD")

db\_host = os.getenv("DB\_HOST")

db\_port = os.getenv("DB\_PORT")

db\_name = os.getenv("DB\_NAME")

if not all([db\_user, db\_pass, db\_host, db\_port, db\_name]):

raise Exception("Missing one or more required local database environment variables (DB\_USER, DB\_PASSWORD, etc.).")

local\_db\_url = f"postgresql+psycopg://{db\_user}:{db\_pass}@{db\_host}:{db\_port}/{db\_name}"

engine = create\_engine(local\_db\_url)

metadata = MetaData()

# --- MODIFIED: Added new columns to the table definition ---

forecasts\_table = Table('forecasts', metadata,

Column('id', Integer, primary\_key=True, autoincrement=True),

Column('Date', DateTime, nullable=False),

Column('Coin', String, nullable=False),

Column('Actual\_Price', Float),

Column('Prophet\_Forecast', Float),

Column('LSTM\_Forecast', Float),

Column('Sentiment\_Score', Float),

Column('RSI', Float),

Column('MACD', Float),

Column('All\_Time\_High', Float),

Column('High\_Forecast\_5\_Day', String),

# --- CoinGlass Futures Data ---

Column('Funding\_Rate', Float),

Column('Open\_Interest', Float),

Column('Long\_Short\_Ratio', Float),

# --- Santiment On-Chain/Social Data ---

Column('MVRV\_Ratio', Float),

Column('Social\_Dominance', Float),

Column('Daily\_Active\_Addresses', Float),

# --- LunarCrush Social Data ---

Column('Galaxy\_Score', Float),

Column('Alt\_Rank', Float),

# --- AI Analysis & Feedback ---

Column('analysis\_summary', String),

Column('analysis\_hypothesis', String),

Column('analysis\_news\_links', String),

Column('report\_title', String),

Column('report\_recap', String),

Column('report\_bullish', String),

Column('report\_bearish', String),

Column('report\_hypothesis', String),

# --- START: ADDED NEW ADVANCED METRIC COLUMNS ---

Column('Leverage\_Ratio', Float),

Column('Futures\_Volume\_24h', Float),

Column('Exchange\_Supply\_Ratio', Float),

# --- END: ADDED NEW ADVANCED METRIC COLUMNS ---

Column('user\_feedback', String),

Column('user\_correction', String)

)

def init\_db():

print(" [INFO] Initializing database...")

try:

metadata.create\_all(engine, checkfirst=True)

print(" [SUCCESS] Database initialized and schema verified.")

except Exception as e:

print(f"❌ [ERROR] Could not initialize database: {e}")

raise

def save\_forecast\_results(results\_df: pd.DataFrame):

print(" [INFO] Saving forecast results to the database...")

try:

results\_df['Date'] = pd.to\_datetime(results\_df['Date'])

# Drop columns that are not in the table schema to prevent errors

inspector = inspect(engine)

table\_columns = [c['name'] for c in inspector.get\_columns('forecasts')]

df\_to\_save = results\_df[[col for col in results\_df.columns if col in table\_columns]]

df\_to\_save.to\_sql('forecasts', engine, if\_exists='append', index=False)

print(f" [SUCCESS] Saved {len(df\_to\_save)} new records to the database.")

except Exception as e:

print(f"❌ [ERROR] Could not save results to database: {e}")

raise

def load\_forecast\_results() -> pd.DataFrame:

print(" [INFO] Loading forecast results from the database...")

try:

query = text("SELECT \* FROM forecasts ORDER BY \"Date\" DESC, id DESC")

with engine.connect() as connection:

df = pd.read\_sql\_query(query, connection)

print(f" [SUCCESS] Loaded {len(df)} records from the database.")

return df

except Exception as e:

print(f"❌ [ERROR] Could not load results from database: {e}")

return pd.DataFrame()

def update\_feedback(record\_id: int, feedback: str, correction: str = ""):

print(f" [INFO] Updating feedback for record ID: {record\_id}...")

try:

with engine.connect() as connection:

stmt = text(

"UPDATE forecasts SET user\_feedback = :feedback, user\_correction = :correction WHERE id = :id"

)

connection.execute(stmt, {"feedback": feedback, "correction": correction, "id": record\_id})

connection.commit()

print(" [SUCCESS] Feedback updated in the database.")

return True

except Exception as e:

print(f"❌ [ERROR] Could not update feedback in database: {e}")

return False

**analyst.py**

*The AI Brain: generates the qualitative report.*

import os

import json

import openai

try:

client = openai.OpenAI()

except openai.OpenAIError:

print("❌ [FATAL] OpenAI API key not configured. Please check your .env file.")

client = None

def get\_daily\_analysis(daily\_briefing\_data: dict) -> dict:

if not client:

return { "summary": "AI analysis failed.", "hypothesis": "Configuration error.", "news\_links": "[]" }

coin\_name = daily\_briefing\_data.get("coin\_name", "the asset")

print(f" [INFO] Briefing AI Analyst for comprehensive report on {coin\_name}...")

# --- START: UPGRADED SYSTEM PROMPT ---

system\_prompt = """

You are an expert crypto market analyst writing a daily briefing. Your tone is objective, data-driven, and insightful. Your task is to synthesize a comprehensive set of market data into a multi-part report.

You MUST provide your response in a single, valid JSON object with the following five keys:

1. "title": A compelling, news-style headline for today's analysis.

2. "price\_action\_recap": A 1-2 sentence summary of the recent price action.

3. "bullish\_case": A markdown-formatted string. Detail the bullish signals. For each point, start with a bolded title (e.g., "\*\*On-Chain Strength\*\*"), cite specific metrics (e.g., MVRV Ratio, Daily Active Addresses, positive Exchange Supply Ratio), and explain the positive implication.

4. "bearish\_case": A markdown-formatted string. Detail the bearish signals. Follow the same format, using bolded titles (e.g., "\*\*Overheated Derivatives Market\*\*") and citing specific metrics (e.g., High Leverage Ratio, Funding Rates).

5. "analyst\_hypothesis": A concluding 2-3 sentence paragraph. Synthesize the conflicting bullish and bearish cases, giving special weight to derivatives and on-chain data, to form a primary, forward-looking hypothesis.

"""

# --- END: UPGRADED SYSTEM PROMPT ---

user\_prompt = f"""

Generate a comprehensive market analysis report for {coin\_name} based on the following data.

Directly cite the data points in your analysis, especially the advanced metrics.

```json

{json.dumps(daily\_briefing\_data, indent=2)}

```

"""

try:

completion = client.chat.completions.create(

model="gpt-4-turbo",

messages=[

{"role": "system", "content": system\_prompt},

{"role": "user", "content": user\_prompt}

],

response\_format={"type": "json\_object"},

temperature=0.4

)

analysis\_from\_ai = json.loads(completion.choices[0].message.content)

summary\_for\_db = f"### Bullish Case\n{analysis\_from\_ai.get('bullish\_case', '')}\n\n### Bearish Case\n{analysis\_from\_ai.get('bearish\_case', '')}"

analysis\_to\_save = {

"summary": summary\_for\_db,

"hypothesis": analysis\_from\_ai.get("analyst\_hypothesis", "No hypothesis generated."),

"news\_links": json.dumps(daily\_briefing\_data.get("top\_headlines", [])),

"report\_title": analysis\_from\_ai.get("title", "Daily Analysis"),

"report\_recap": analysis\_from\_ai.get("price\_action\_recap", ""),

"report\_bullish": analysis\_from\_ai.get("bullish\_case", ""),

"report\_bearish": analysis\_from\_ai.get("bearish\_case", ""),

"report\_hypothesis": analysis\_from\_ai.get("analyst\_hypothesis", "")

}

print(f" [SUCCESS] Comprehensive AI analysis for {coin\_name} generated.")

return analysis\_to\_save

except Exception as e:

print(f"❌ [ERROR] AI Analyst API call failed: {e}")

return {

"summary": "AI analysis could not be generated.", "hypothesis": str(e), "news\_links": "[]",

"report\_title": "Analysis Failed", "report\_recap": "", "report\_bullish": "", "report\_bearish": "", "report\_hypothesis": ""

}

**daily\_runner.py**

*The main orchestrator script.*

import pandas as pd

from datetime import datetime

import numpy as np

import os

import json

import openai

from frozendict import frozendict

from dotenv import load\_dotenv

# --- Load Environment and Keys ---

load\_dotenv()

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

news\_api\_key = os.getenv("NEWS\_API\_KEY")

# --- Robust Key Check ---

required\_keys = [

"OPENAI\_API\_KEY", "NEWS\_API\_KEY", "SANTIMENT\_API\_KEY",

"LUNARCRUSH\_API\_KEY", "COINGECKO\_API\_KEY"

]

missing\_keys = [key for key in required\_keys if not os.getenv(key)]

if missing\_keys:

print(f"❌ [FATAL] The following required API keys are missing: {', '.join(missing\_keys)}")

exit(1)

# --- Module Imports ---

try:

from data\_utils import fetch\_data

from forecasting import prophet\_forecast, lstm\_forecast, prophet\_forecast\_highs

from sentiment import get\_news\_sentiment

from db\_utils import init\_db, save\_forecast\_results

from analyst import get\_daily\_analysis

except ImportError as e:

print(f"❌ [FATAL] Failed to import a required module: {e}. Exiting.")

exit(1)

# --- Configuration ---

COINS = {"BTC-USD": "Bitcoin", "ETH-USD": "Ethereum", "XRP-USD": "XRP"}

DATA\_DIR = os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), 'data')

def default\_json\_serializer(obj):

if isinstance(obj, pd.Timestamp): return obj.isoformat()

if isinstance(obj, frozendict): return dict(obj)

raise TypeError(f"Object of type {obj.\_\_class\_\_.\_\_name\_\_} is not JSON serializable")

def run\_daily\_analysis():

print("✅ [START] Kicking off daily crypto forecasting run...")

init\_db()

today = datetime.today().strftime("%Y-%m-%d")

os.makedirs(DATA\_DIR, exist\_ok=True)

all\_results = []

for ticker, name in COINS.items():

print(f"\nProcessing {ticker} ({name})...")

try:

market\_data = fetch\_data(ticker)

if market\_data.empty or len(market\_data) < 61:

print(f" [WARN] Insufficient data for {ticker}. Skipping.")

continue

detailed\_data\_path = os.path.join(DATA\_DIR, f"{ticker}\_data.csv")

market\_data.to\_csv(detailed\_data\_path)

latest\_data = market\_data.iloc[-1]

prophet\_price = prophet\_forecast(market\_data.copy())

lstm\_price = lstm\_forecast(market\_data.copy())

high\_forecasts\_list = prophet\_forecast\_highs(market\_data.copy(), periods=5)

sentiment\_score, top\_headlines = get\_news\_sentiment(coin\_ticker=ticker, coin\_name=name, api\_key=news\_api\_key)

# Prepare a comprehensive briefing for the AI Analyst

daily\_briefing\_data = {

"coin\_name": name,

"actual\_price": latest\_data.get("Close", 0.0),

"prophet\_forecast": prophet\_price,

"sentiment\_score": sentiment\_score,

"rsi": latest\_data.get("RSI", 0.0),

"macd": latest\_data.get("MACD", 0.0),

"funding\_rate": latest\_data.get("Funding\_Rate", 0.0),

"open\_interest": latest\_data.get("Open\_Interest", 0.0),

"long\_short\_ratio": latest\_data.get("Long\_Short\_Ratio", 0.0),

"mvrv\_ratio": latest\_data.get("MVRV\_Ratio", 0.0),

"social\_dominance": latest\_data.get("Social\_Dominance", 0.0),

"daily\_active\_addresses": latest\_data.get("Daily\_Active\_Addresses", 0.0),

"galaxy\_score": latest\_data.get("Galaxy\_Score", 0.0),

"alt\_rank": latest\_data.get("Alt\_Rank", 0.0),

# START: ADDED NEW DATA TO BRIEFING

"leverage\_ratio": latest\_data.get("Leverage\_Ratio", 0.0),

"futures\_volume\_24h": latest\_data.get("Futures\_Volume\_24h", 0.0),

"exchange\_supply\_ratio": latest\_data.get("Exchange\_Supply\_Ratio", 0.0),

# END: ADDED NEW DATA TO BRIEFING

"top\_headlines": top\_headlines

}

analysis\_results = get\_daily\_analysis(daily\_briefing\_data)

# Assemble the final, complete record for the database

result = {

"Date": today,

"Coin": ticker,

"Actual\_Price": latest\_data.get("Close", 0.0),

"Prophet\_Forecast": prophet\_price,

"LSTM\_Forecast": lstm\_price,

"Sentiment\_Score": sentiment\_score,

"RSI": latest\_data.get("RSI", 0.0),

"MACD": latest\_data.get("MACD", 0.0),

"All\_Time\_High": latest\_data.get("All\_Time\_High\_Real", 0.0),

"High\_Forecast\_5\_Day": json.dumps(high\_forecasts\_list, default=default\_json\_serializer),

"Funding\_Rate": latest\_data.get("Funding\_Rate", 0.0),

"Open\_Interest": latest\_data.get("Open\_Interest", 0.0),

"Long\_Short\_Ratio": latest\_data.get("Long\_Short\_Ratio", 0.0),

"MVRV\_Ratio": latest\_data.get("MVRV\_Ratio", 0.0),

"Social\_Dominance": latest\_data.get("Social\_Dominance", 0.0),

"Daily\_Active\_Addresses": latest\_data.get("Daily\_Active\_Addresses", 0.0),

"Galaxy\_Score": latest\_data.get("Galaxy\_Score", 0.0),

"Alt\_Rank": latest\_data.get("Alt\_Rank", 0.0),

"Exchange\_Net\_Flow": latest\_data.get("Exchange\_Net\_Flow", 0.0),

# START: ADDED NEW DATA TO SAVE

"Leverage\_Ratio": latest\_data.get("Leverage\_Ratio", 0.0),

"Futures\_Volume\_24h": latest\_data.get("Futures\_Volume\_24h", 0.0),

"Exchange\_Supply\_Ratio": latest\_data.get("Exchange\_Supply\_Ratio", 0.0),

# END: ADDED NEW DATA TO SAVE

"analysis\_summary": analysis\_results.get("summary"),

"analysis\_hypothesis": analysis\_results.get("hypothesis"),

"analysis\_news\_links": analysis\_results.get("news\_links"),

"report\_title": analysis\_results.get("report\_title"),

"report\_recap": analysis\_results.get("report\_recap"),

"report\_bullish": analysis\_results.get("report\_bullish"),

"report\_bearish": analysis\_results.get("report\_bearish"),

"report\_hypothesis": analysis\_results.get("report\_hypothesis"),

"user\_feedback": None,

"user\_correction": None

}

all\_results.append(result)

except Exception as e:

print(f"❌ [ERROR] An unexpected error occurred while processing {ticker}: {e}")

continue

print("\n✅ [FINISH] Daily processing complete.")

if all\_results:

results\_df = pd.DataFrame(all\_results)

save\_forecast\_results(results\_df)

else:

print("\n[WARN] No results were generated. Database not updated.")

if \_\_name\_\_ == "\_\_main\_\_":

run\_daily\_analysis()

**dashboard\_app.py**

from dotenv import load\_dotenv

load\_dotenv() # Load variables from .env file FIRST

import streamlit as st

import pandas as pd

import os

import json

import numpy as np

from db\_utils import load\_forecast\_results, update\_feedback

# START: ADDED NEW IMPORT

from chart\_analyst import analyze\_bollinger\_bands, analyze\_rsi

# END: ADDED NEW IMPORT

# --- Page Configuration ---

st.set\_page\_config(page\_title="305 Crypto Forecast", page\_icon="📈", layout="wide")

# --- Robust Local Path Configuration ---

SCRIPT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

DATA\_DIR = os.path.join(SCRIPT\_DIR, 'data')

# --- Caching Functions ---

@st.cache\_data(ttl=3600)

def get\_historical\_data():

"""Loads all forecast data from the database."""

return load\_forecast\_results()

@st.cache\_data(ttl=3600)

def load\_chart\_data(ticker):

"""Loads the detailed indicator data from the local CSV file."""

file\_path = os.path.join(DATA\_DIR, f"{ticker}\_data.csv")

if os.path.exists(file\_path):

try:

return pd.read\_csv(file\_path, index\_col=0, parse\_dates=True)

except Exception:

return None

return None

# START: ADDED NEW FIBONACCI CALCULATION FUNCTION

def calculate\_fibonacci\_levels(df: pd.DataFrame):

"""Calculates Fibonacci retracement levels for the given data."""

if df.empty:

return {}

highest\_high = df['High'].max()

lowest\_low = df['Low'].min()

price\_range = highest\_high - lowest\_low

levels = {

"Level 0% (High)": highest\_high,

"Level 23.6%": highest\_high - (price\_range \* 0.236),

"Level 38.2%": highest\_high - (price\_range \* 0.382),

"Level 50%": highest\_high - (price\_range \* 0.5),

"Level 61.8%": highest\_high - (price\_range \* 0.618),

"Level 100% (Low)": lowest\_low,

}

return levels

# END: ADDED NEW FIBONACCI CALCULATION FUNCTION

# --- Main Application ---

st.title("📈 305 Crypto Forecast Dashboard")

st.markdown("An automated forecasting and sentiment analysis system for major cryptocurrencies.")

historical\_df = get\_historical\_data()

if historical\_df.empty:

st.error("🚨 No forecast data found in the database. The daily analysis may not have run yet.")

st.stop()

# --- Data Preparation ---

latest\_date = historical\_df['Date'].max()

latest\_forecast\_df = historical\_df[historical\_df['Date'] == latest\_date].copy()

# --- Utility Function ---

def format\_numeric\_columns(df):

formatted\_df = df.copy()

numeric\_cols = [

'Actual\_Price', 'Prophet\_Forecast', 'LSTM\_Forecast', 'All\_Time\_High', 'RSI', 'MACD',

'High', 'Low', 'Close', 'Open', 'Volume', 'SMA', 'EMA',

'BB\_High', 'BB\_Low', 'Stoch\_k', 'Stoch\_d', 'OBV',

'Ichimoku\_a', 'Ichimoku\_b', 'Transaction\_Volume\_24h', 'Circulating\_Supply',

'Market\_Cap\_Rank', 'Community\_Score', 'Developer\_Score', 'Sentiment\_Up\_Percentage',

'Forecasted High', 'Funding\_Rate', 'Open\_Interest', 'Long\_Short\_Ratio',

'MVRV\_Ratio', 'Social\_Dominance', 'Daily\_Active\_Addresses',

'Galaxy\_Score', 'Alt\_Rank',

# START: ADDED NEW COLS FOR FORMATTING

'Leverage\_Ratio', 'Futures\_Volume\_24h', 'Exchange\_Supply\_Ratio'

# END: ADDED NEW COLS FOR FORMATTING

]

for col in numeric\_cols:

if col in formatted\_df.columns:

try:

formatted\_df[col] = formatted\_df[col].apply(lambda x: f"{x:,.2f}" if pd.notna(x) and isinstance(x, (int, float)) else x)

except (ValueError, TypeError):

pass

return formatted\_df

# --- Sidebar & Main Content ---

st.sidebar.header("Dashboard Options")

selected\_coin = st.sidebar.selectbox("Select a Cryptocurrency", latest\_forecast\_df['Coin'].unique())

chart\_data = load\_chart\_data(selected\_coin)

coin\_forecast = latest\_forecast\_df[latest\_forecast\_df['Coin'] == selected\_coin].iloc[0]

# --- Main Page Layout ---

st.header(f"Today's Overview for {selected\_coin}")

col1, col2, col3 = st.columns(3)

actual\_price = pd.to\_numeric(coin\_forecast['Actual\_Price'], errors='coerce')

all\_time\_high = pd.to\_numeric(coin\_forecast['All\_Time\_High'], errors='coerce')

sentiment\_score = pd.to\_numeric(coin\_forecast['Sentiment\_Score'], errors='coerce')

col1.metric("Actual Price", f"${actual\_price:,.2f}" if pd.notna(actual\_price) else "N/A")

col2.metric("All-Time High", f"${all\_time\_high:,.2f}" if pd.notna(all\_time\_high) else "N/A")

col3.metric("Sentiment Score", f"{sentiment\_score:.2f}" if pd.notna(sentiment\_score) else "N/A")

# --- START: UPGRADED AI ANALYSIS SECTION ---

st.header("Daily AI Analyst Report")

with st.container(border=True):

title = coin\_forecast.get('report\_title', 'Analysis not available.')

recap = coin\_forecast.get('report\_recap', '')

bullish\_case = coin\_forecast.get('report\_bullish', 'Bullish case not available.')

bearish\_case = coin\_forecast.get('report\_bearish', 'Bearish case not available.')

hypothesis = coin\_forecast.get('report\_hypothesis', 'Hypothesis not available.')

news\_links\_json = coin\_forecast.get('analysis\_news\_links', '[]')

st.subheader(title)

st.caption(recap)

st.markdown("---")

col\_bull, col\_bear = st.columns(2)

with col\_bull:

st.markdown("#### Bullish Case 🐂")

st.markdown(bullish\_case)

with col\_bear:

st.markdown("#### Bearish Case 🐻")

st.markdown(bearish\_case)

st.markdown("---")

st.subheader("Analyst's Final Hypothesis")

with st.container(border=True):

st.markdown(hypothesis)

st.subheader("Influential News")

try:

news\_links = json.loads(news\_links\_json) if pd.notna(news\_links\_json) else []

if news\_links:

for item in news\_links:

st.markdown(f"- [{item['title']}]({item['url']})")

else:

st.write("No specific news articles were identified as highly influential today.")

except (json.JSONDecodeError, TypeError):

st.write("Could not parse news links.")

st.markdown("---")

st.subheader("Provide Feedback on this Analysis")

record\_id = coin\_forecast['id']

current\_feedback = coin\_forecast.get('user\_feedback')

if pd.notna(current\_feedback):

st.success(f"Feedback previously saved: \*\*{current\_feedback}\*\*")

col\_confirm, col\_deny = st.columns(2)

with col\_confirm:

if st.button("Confirm Analysis ✅", key=f"confirm\_{record\_id}"):

if update\_feedback(record\_id, "Confirmed"):

st.toast("Feedback 'Confirmed' saved!", icon="🎉")

st.rerun()

else:

st.error("Failed to save feedback.")

with col\_deny:

if st.button("Deny Analysis ❌", key=f"deny\_{record\_id}"):

st.session\_state[f'deny\_clicked\_{record\_id}'] = True

if st.session\_state.get(f'deny\_clicked\_{record\_id}'):

with st.form(key=f"correction\_form\_{record\_id}"):

correction\_text = st.text\_area("What was wrong with the analysis? Please provide your correction.")

submitted = st.form\_submit\_button("Submit Correction")

if submitted:

if update\_feedback(record\_id, "Denied", correction\_text):

st.toast("Correction saved! Thank you.", icon="🙌")

st.session\_state[f'deny\_clicked\_{record\_id}'] = False

st.rerun()

else:

st.error("Failed to save correction.")

# --- END: UPGRADED AI ANALYSIS SECTION ---

st.header("Professional Grade Market Indicators")

with st.container(border=True):

st.subheader("Futures & Derivatives Data")

# START: MODIFIED TO INCLUDE NEW METRICS

cg\_col1, cg\_col2, cg\_col3, cg\_col4 = st.columns(4)

funding\_rate = pd.to\_numeric(coin\_forecast.get('Funding\_Rate'), errors='coerce')

open\_interest = pd.to\_numeric(coin\_forecast.get('Open\_Interest'), errors='coerce')

long\_short\_ratio = pd.to\_numeric(coin\_forecast.get('Long\_Short\_Ratio'), errors='coerce')

futures\_volume = pd.to\_numeric(coin\_forecast.get('Futures\_Volume\_24h'), errors='coerce')

cg\_col1.metric("Funding Rate", f"{funding\_rate:.4f}%" if pd.notna(funding\_rate) else "N/A")

cg\_col2.metric("Open Interest", f"${open\_interest:,.0f}" if pd.notna(open\_interest) else "N/A")

cg\_col3.metric("Long/Short Ratio", f"{long\_short\_ratio:.2f}" if pd.notna(long\_short\_ratio) else "N/A")

cg\_col4.metric("Futures Volume (24h)", f"${futures\_volume:,.0f}" if pd.notna(futures\_volume) else "N/A")

# END: MODIFIED TO INCLUDE NEW METRICS

st.subheader("On-Chain & Social Metrics")

# START: MODIFIED TO INCLUDE NEW METRICS

san\_col1, san\_col2, san\_col3, san\_col4 = st.columns(4)

mvrv = pd.to\_numeric(coin\_forecast.get('MVRV\_Ratio'), errors='coerce')

social\_dom = pd.to\_numeric(coin\_forecast.get('Social\_Dominance'), errors='coerce')

daa = pd.to\_numeric(coin\_forecast.get('Daily\_Active\_Addresses'), errors='coerce')

esr = pd.to\_numeric(coin\_forecast.get('Exchange\_Supply\_Ratio'), errors='coerce')

san\_col1.metric("MVRV Ratio", f"{mvrv:.2f}" if pd.notna(mvrv) else "N/A")

san\_col2.metric("Social Dominance", f"{social\_dom:.2f}%" if pd.notna(social\_dom) else "N/A")

san\_col3.metric("Daily Active Addresses", f"{daa:,.0f}" if pd.notna(daa) else "N/A")

san\_col4.metric("Exchange Supply Ratio", f"{esr:.2f}" if pd.notna(esr) else "N/A (Placeholder)")

# END: MODIFIED TO INCLUDE NEW METRICS

st.subheader("Social Intelligence (from LunarCrush)")

lc\_col1, lc\_col2 = st.columns(2)

galaxy\_score = pd.to\_numeric(coin\_forecast.get('Galaxy\_Score'), errors='coerce')

alt\_rank = pd.to\_numeric(coin\_forecast.get('Alt\_Rank'), errors='coerce')

lc\_col1.metric("Galaxy Score™", f"{galaxy\_score:.1f}/100" if pd.notna(galaxy\_score) else "N/A")

lc\_col2.metric("AltRank™", f"#{alt\_rank:.0f}" if pd.notna(alt\_rank) else "N/A")

st.header("5-Day High Forecast vs. Historical Highs")

# This section remains unchanged

if chart\_data is not None and 'High\_Forecast\_5\_Day' in coin\_forecast and pd.notna(coin\_forecast['High\_Forecast\_5\_Day']):

historical\_highs = chart\_data[['High']].tail(5)

historical\_highs.index = historical\_highs.index.strftime('%Y-%m-%d')

try:

forecast\_data = json.loads(str(coin\_forecast['High\_Forecast\_5\_Day']))

if forecast\_data:

forecast\_df\_highs = pd.DataFrame(forecast\_data)

forecast\_df\_highs['ds'] = pd.to\_datetime(forecast\_df\_highs['ds']).dt.strftime('%Y-%m-%d')

forecast\_df\_highs = forecast\_df\_highs.rename(columns={'ds': 'Date', 'yhat': 'Forecasted High'}).set\_index('Date')

combined\_df = pd.concat([historical\_highs, forecast\_df\_highs['Forecasted High']], axis=1)

st.bar\_chart(combined\_df)

st.dataframe(format\_numeric\_columns(combined\_df.reset\_index()))

else:

st.warning("Forecast data is available but empty.")

except (json.JSONDecodeError, TypeError):

st.error("Could not parse the 5-day forecast data.")

else:

st.warning(f"Could not load 5-day forecast data for {selected\_coin}.")

st.header(f"Technical Indicators for {selected\_coin}")

if chart\_data is not None:

# START: UPGRADED BOLLINGER BANDS SECTION

st.subheader("Price, Moving Averages, & Bollinger Bands")

st.line\_chart(chart\_data[['Close', 'SMA', 'EMA', 'BB\_High', 'BB\_Low']])

st.info(f"\*\*Analysis:\*\* {analyze\_bollinger\_bands(chart\_data)}") # DYNAMIC ANALYSIS

# END: UPGRADED BOLLINGER BANDS SECTION

tech\_col1, tech\_col2 = st.columns(2)

with tech\_col1:

# START: UPGRADED RSI SECTION

st.subheader("RSI (Relative Strength Index)")

st.line\_chart(chart\_data['RSI'])

st.info(f"\*\*Analysis:\*\* {analyze\_rsi(chart\_data)}") # DYNAMIC ANALYSIS

# END: UPGRADED RSI SECTION

st.subheader("Stochastic Oscillator")

st.line\_chart(chart\_data[['Stoch\_k', 'Stoch\_d']])

# Placeholder for dynamic Stochastic analysis

st.info("\*\*Analysis:\*\* Readings above 80 indicate overbought conditions, while below 20 indicate oversold.")

with tech\_col2:

st.subheader("MACD (Moving Average Convergence Divergence)")

st.line\_chart(chart\_data[['MACD', 'MACD\_Signal']])

# Placeholder for dynamic MACD analysis

st.info("\*\*Analysis:\*\* When the MACD line crosses above the Signal line, it's a bullish signal.")

st.subheader("OBV (On-Balance Volume)")

st.line\_chart(chart\_data['OBV'])

# Placeholder for dynamic OBV analysis

st.info("\*\*Analysis:\*\* A rising OBV indicates positive volume pressure that can confirm an uptrend.")

# START: UPGRADED PRICE CHART WITH FIBONACCI LEVELS

st.subheader("Price Chart with Fibonacci Levels")

fib\_levels = calculate\_fibonacci\_levels(chart\_data)

# Create a new DataFrame for plotting that includes the price and Fib levels

plot\_df = chart\_data[['Close']].copy()

for name, level in fib\_levels.items():

plot\_df[name] = level

st.line\_chart(plot\_df)

st.info(f"""

\*\*Analysis:\*\* The Fibonacci levels are key potential areas of support and resistance.

The market will often see price react around these levels.

Currently, the key support is at the \*\*{list(fib\_levels.keys())[4]}\*\* (${fib\_levels[list(fib\_levels.keys())[4]]:,.2f}) and

the key resistance is at the \*\*{list(fib\_levels.keys())[1]}\*\* (${fib\_levels[list(fib\_levels.keys())[1]]:,.2f}).

""")

# END: UPGRADED PRICE CHART WITH FIBONACCI LEVELS

else:

st.warning(f"Could not load technical indicator data for {selected\_coin}.")

# ... (The "On-Chain & Fundamental Indicators" and "Raw Data Viewer" sections remain unchanged) ...

st.header(f"On-Chain & Fundamental Indicators for {selected\_coin}")

if chart\_data is not None:

st.subheader("On-Chain & Market Indicators (from CoinGecko)")

st.info(

"""

\*\*Reasoning:\*\* These metrics provide a direct view of a blockchain's recent market activity. (Source: CoinGecko)

- \*\*Transaction Volume (24h):\*\* The total value in USD of all transactions for this asset in the last 24 hours. High volume can help confirm the strength of a price trend.

- \*\*Circulating Supply:\*\* The number of coins that are publicly available and circulating in the market. This is a key metric for calculating market capitalization and assessing scarcity.

"""

)

onchain\_col1, onchain\_col2 = st.columns(2)

with onchain\_col1:

st.subheader("Transaction Volume (24h)")

if 'Transaction\_Volume\_24h' in chart\_data.columns:

latest\_volume = chart\_data['Transaction\_Volume\_24h'].iloc[-1]

st.metric("Volume (USD)", f"${latest\_volume:,.2f}")

else:

st.metric("Volume (USD)", "N/A")

with onchain\_col2:

st.subheader("Circulating Supply")

if 'Circulating\_Supply' in chart\_data.columns:

latest\_supply = chart\_data['Circulating\_Supply'].iloc[-1]

st.metric("Supply", f"{latest\_supply:,.0f} {selected\_coin.split('-')[0]}")

else:

st.metric("Supply", "N/A")

st.subheader("Fundamental Indicators (from CoinGecko)")

st.info(

"""

\*\*Reasoning:\*\* These scores assess the long-term viability, community health, and development activity of a project. (Source: CoinGecko)

- \*\*Market Cap Rank:\*\* The project's rank relative to all other cryptocurrencies by market capitalization.

- \*\*Community Score:\*\* A score based on social media activity.

- \*\*Developer Score:\*\* A score based on GitHub activity.

- \*\*Sentiment:\*\* The percentage of users who voted "Good" on CoinGecko.

"""

)

latest\_fundamentals = chart\_data.iloc[-1]

fund\_col1, fund\_col2, fund\_col3, fund\_col4 = st.columns(4)

fund\_col1.metric("Market Cap Rank", f"#{latest\_fundamentals.get('Market\_Cap\_Rank', 0):.0f}")

fund\_col2.metric("Community Score", f"{latest\_fundamentals.get('Community\_Score', 0):.1f}")

fund\_col3.metric("Developer Score", f"{latest\_fundamentals.get('Developer\_Score', 0):.1f}")

fund\_col4.metric("Sentiment (Up %)", f"{latest\_fundamentals.get('Sentiment\_Up\_Percentage', 0):.1f}%")

else:

st.warning(f"Could not load on-chain or fundamental data for {selected\_coin}.")

st.header("Raw Data Viewer")

st.subheader("Full Historical Forecast Data")

st.dataframe(format\_numeric\_columns(historical\_df))

st.subheader(f"Full Daily Indicator Data for {selected\_coin}")

if chart\_data is not None:

st.dataframe(format\_numeric\_columns(chart\_data))

st.sidebar.markdown("---")

st.sidebar.info("This is for educational purposes only and is not financial advice.")

sentiment.py

(Sentiment analysis module)

import os

import requests

import openai

import re

from datetime import datetime, timedelta

def get\_news\_sentiment(coin\_ticker: str, coin\_name: str, api\_key: str) -> tuple:

"""

Fetches recent news, returns a sentiment score from GPT, and the top articles.

Returns:

tuple: A (score, articles\_list) tuple.

"""

print(f" [INFO] Starting sentiment analysis for {coin\_ticker}...")

try:

if not api\_key:

print(" [WARN] NewsAPI key was not provided. Skipping.")

return 0.0, []

from\_date = (datetime.now() - timedelta(days=3)).strftime('%Y-%m-%d')

url = (f'https://newsapi.org/v2/everything?q={coin\_name}&from={from\_date}&sortBy=publishedAt&language=en&apiKey={api\_key}')

response = requests.get(url)

response.raise\_for\_status()

articles = response.json().get("articles", [])

if not articles:

print(f" [WARN] No recent news articles found for {coin\_name}.")

return 0.0, []

top\_articles = [{"title": a['title'], "url": a['url']} for a in articles[:5]]

headlines\_for\_analysis = [f"Title: {a['title']}. Desc: {a.get('description', '')}" for a in articles[:10]]

news\_text = "\n".join(headlines\_for\_analysis)

client = openai.OpenAI()

completion = client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are a financial sentiment analyst. Based on the news headlines, provide a single sentiment score from -1.0 to 1.0. Respond with only the numerical score."},

{"role": "user", "content": f"Analyze sentiment for {coin\_name} from these articles:\n\n{news\_text}"}

],

temperature=0.0,

max\_tokens=10

)

content = completion.choices[0].message.content

match = re.search(r"(-?\d+\.?\d\*)", content)

if match:

score = float(match.group(0))

final\_score = max(-1.0, min(1.0, score))

print(f" [SUCCESS] Sentiment analysis complete. Score: {final\_score:.2f}")

return final\_score, top\_articles

else:

print(f" [WARN] Could not parse sentiment score from OpenAI response: '{content}'")

return 0.0, top\_articles

except Exception as e:

print(f" [ERROR] Sentiment analysis error: {e}")

return 0.0, []

forecasting.py

(Prophet and LSTM models)

import pandas as pd

import numpy as np

from prophet import Prophet

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM

from tensorflow.keras.optimizers import Adam

def prophet\_forecast(df: pd.DataFrame) -> float:

if df.empty: return np.nan

print(" [INFO] Starting Prophet 'Close' price forecast...")

try:

prophet\_df = pd.DataFrame({'ds': df.index, 'y': df['Close'].values})

model = Prophet(daily\_seasonality=True)

model.fit(prophet\_df)

future = model.make\_future\_dataframe(periods=1)

forecast = model.predict(future)

predicted\_price = forecast.iloc[-1]['yhat']

print(f" [SUCCESS] Prophet 'Close' forecast complete. Predicted: {predicted\_price:.2f}")

return float(predicted\_price)

except Exception as e:

print(f" [ERROR] Prophet 'Close' forecasting error: {e}")

return np.nan

def prophet\_forecast\_highs(df: pd.DataFrame, periods: int = 5) -> list:

if df.empty: return []

print(f" [INFO] Starting Prophet {periods}-day 'High' price forecast...")

try:

prophet\_df = pd.DataFrame({'ds': df.index, 'y': df['High'].values})

model = Prophet(daily\_seasonality=True)

model.fit(prophet\_df)

future = model.make\_future\_dataframe(periods=periods)

forecast = model.predict(future)

future\_forecasts = forecast.iloc[-periods:]

predicted\_highs = future\_forecasts[['ds', 'yhat']].to\_dict('records')

print(f" [SUCCESS] Prophet {periods}-day 'High' forecast complete.")

return predicted\_highs

except Exception as e:

print(f" [ERROR] Prophet 'High' forecasting error: {e}")

return []

def lstm\_forecast(df: pd.DataFrame, look\_back\_period: int = 60) -> float:

if df.empty or len(df) <= look\_back\_period: return np.nan

print(" [INFO] Starting LSTM 'Close' price forecast...")

try:

data = df[['Close']].copy()

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(data)

X\_train, y\_train = [], []

for i in range(look\_back\_period, len(scaled\_data)):

X\_train.append(scaled\_data[i-look\_back\_period:i, 0])

y\_train.append(scaled\_data[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

model = Sequential([

LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),

LSTM(50, return\_sequences=False),

Dense(25),

Dense(1)

])

model.compile(optimizer=Adam(), loss='mean\_squared\_error')

model.fit(X\_train, y\_train, batch\_size=16, epochs=5, verbose=0)

last\_sequence = scaled\_data[-look\_back\_period:]

last\_sequence = np.reshape(last\_sequence, (1, look\_back\_period, 1))

predicted\_price\_scaled = model.predict(last\_sequence, verbose=0)

predicted\_price = scaler.inverse\_transform(predicted\_price\_scaled)

print(f" [SUCCESS] LSTM 'Close' forecast complete. Predicted: {predicted\_price[0][0]:,.2f}")

return float(predicted\_price[0][0])

except Exception as e:

print(f" [ERROR] LSTM forecasting error: {e}")

return np.nan

chart\_analyst.py

(Dynamic chart interpretation)

import pandas as pd

def analyze\_bollinger\_bands(df: pd.DataFrame) -> str:

"""

Analyzes the latest Bollinger Bands data and returns a text interpretation.

"""

if df.empty or not all(k in df.columns for k in ['Close', 'BB\_High', 'BB\_Low']):

return "Bollinger Bands data is not available for analysis."

latest = df.iloc[-1]

close = latest['Close']

bb\_high = latest['BB\_High']

bb\_low = latest['BB\_Low']

analysis = f"The bands are currently between ${bb\_low:,.2f} and ${bb\_high:,.2f}. "

if close >= bb\_high:

analysis += f"The current price of \*\*${close:,.2f} is touching or above the upper band\*\*, suggesting the asset may be \*\*overbought\*\* and could be due for a short-term price correction."

elif close <= bb\_low:

analysis += f"The current price of \*\*${close:,.2f} is touching or below the lower band\*\*, suggesting the asset may be \*\*oversold\*\* and could be poised for a rebound."

else:

analysis += f"The current price of \*\*${close:,.2f} is trading within the bands\*\*, which does not indicate an immediate overbought or oversold condition."

return analysis

def analyze\_rsi(df: pd.DataFrame) -> str:

"""

Analyzes the latest RSI data and returns a text interpretation.

"""

if df.empty or 'RSI' not in df.columns:

return "RSI data is not available for analysis."

latest\_rsi = df['RSI'].iloc[-1]

analysis = f"The current RSI is \*\*{latest\_rsi:.2f}\*\*. "

if latest\_rsi >= 70:

analysis += "A value \*\*above 70\*\* indicates the asset is likely \*\*overbought\*\*, which can signal a potential price pullback."

elif latest\_rsi <= 30:

analysis += "A value \*\*below 30\*\* indicates the asset is likely \*\*oversold\*\*, which can signal a potential price rebound."

else:

analysis += "This value is in the \*\*neutral zone\*\*, suggesting the market is not currently in an extreme overbought or oversold condition."

return analysis

# We will add more functions here for MACD, Stochastics, etc. in the future.

* test\_coinglass.py
* import os
* import requests
* import json
* from dotenv import load\_dotenv
* # Load environment variables from .env file
* load\_dotenv()
* COINGLASS\_API\_KEY = os.getenv("COINGLASS\_API\_KEY")
* # --- MODIFIED: List of coins to test ---
* COINS\_TO\_TEST = ["BTC", "ETH", "XRP"]
* if not COINGLASS\_API\_KEY:
* print("❌ COINGLASS\_API\_KEY not found in .env file. Exiting.")
* exit()
* print("--- Definitive CoinGlass API Multi-Coin Test ---")
* print(f"Using Key: ...{COINGLASS\_API\_KEY[-4:]}")
* # --- Loop through each coin ---
* for symbol in COINS\_TO\_TEST:
* headers = {
* 'accept': 'application/json',
* 'coinglassSecret': COINGLASS\_API\_KEY
* }
* url = f"https://open-api.coinglass.com/public/v2/perpetual\_market?ex=Binance&symbol={symbol}"
* print(f"\n{'='\*10} Testing: {symbol} {'='\*10}")
* print(f"URL: {url}")
* try:
* response = requests.get(url, headers=headers)
* print(f"Status Code: {response.status\_code}")
* if response.ok:
* json\_data = response.json()
* print("\n--- Parsing Response ---")
* if json\_data.get('success') and json\_data.get('data'):
* market\_data\_list = json\_data['data'].get(symbol, [])
* binance\_data = next((item for item in market\_data\_list if item.get("exchangeName") == "Binance"), None)
* if binance\_data:
* print("✅ Successfully found data for Binance exchange.")
* funding\_rate = binance\_data.get('rate', 0.0) \* 100
* open\_interest = binance\_data.get('openInterest', 0.0)
* futures\_volume = binance\_data.get('totalVolUsd', 0.0)
* long\_rate = binance\_data.get('longRate', 0.0)
* short\_rate = binance\_data.get('shortRate', 1.0)
* long\_short\_ratio = long\_rate / short\_rate if short\_rate > 0 else 0
* print("\n--- Extracted Metrics ---")
* print(f"Funding Rate: {funding\_rate:.4f}%")
* print(f"Open Interest: ${open\_interest:,.2f}")
* print(f"Futures Volume (24h): ${futures\_volume:,.2f}")
* print(f"Long/Short Ratio: {long\_short\_ratio:.2f}")
* if funding\_rate == 0 and open\_interest == 0 and futures\_volume == 0 and long\_short\_ratio == 0:
* print("\n⚠️ WARNING: All extracted values are zero.")
* else:
* print("\n🎉 SUCCESS: All key metrics were extracted successfully.")
* else:
* print(f"❌ ERROR: Could not find 'Binance' data for {symbol} within the API response.")
* else:
* print(f"❌ ERROR: API response for {symbol} indicates failure or is empty.")
* print("Full Response:", json.dumps(json\_data, indent=2))
* else:
* print(f"❌ ERROR: API call for {symbol} failed.")
* print("Response Text:", response.text)
* except Exception as e:
* print(f"An error occurred for {symbol}: {e}")
* test\_coingecko.py
* import os
* import requests
* import json
* from dotenv import load\_dotenv
* # Load environment variables from .env file
* load\_dotenv()
* COINGECKO\_API\_KEY = os.getenv("COINGECKO\_API\_KEY")
* COIN\_ID = "bitcoin" # We'll test with bitcoin
* if not COINGECKO\_API\_KEY:
* print("❌ COINGECKO\_API\_KEY not found in .env file. Exiting.")
* exit()
* print(f"--- Definitive CoinGecko API Test for {COIN\_ID} ---")
* print(f"Using Key: ...{COINGECKO\_API\_KEY[-4:]}")
* url = f"https://api.coingecko.com/api/v3/coins/{COIN\_ID}"
* params = {'x\_cg\_demo\_api\_key': COINGECKO\_API\_KEY}
* try:
* print("\n--- Testing Endpoint: CoinGecko Coins ---")
* response = requests.get(url, params=params)
* print(f"Status Code: {response.status\_code}")
* if response.ok:
* json\_data = response.json()
* print("\n--- Parsing Response ---")
* community\_score = json\_data.get('community\_score', "Not Available")
* developer\_score = json\_data.get('developer\_score', "Not Available")
* print("\n--- Extracted Metrics ---")
* print(f"Community Score: {community\_score}")
* print(f"Developer Score: {developer\_score}")
* if community\_score != "Not Available" and developer\_score != "Not Available":
* print("\n🎉 SUCCESS: All key metrics were extracted successfully.")
* else:
* print("\n⚠️ WARNING: One or more metrics were not available.")
* print("Full Response (first 500 chars):", str(json\_data)[:500])
* else:
* print(f"❌ ERROR: API call failed.")
* print("Response Text:", response.text)
* except Exception as e:
* print(f"An error occurred: {e}")
* test\_santiment.py
* import os
* import requests
* import json
* from dotenv import load\_dotenv
* # Load environment variables from .env file
* load\_dotenv()
* SANTIMENT\_API\_KEY = os.getenv("SANTIMENT\_API\_KEY")
* SLUG = "bitcoin" # We'll test with bitcoin
* if not SANTIMENT\_API\_KEY:
* print("❌ SANTIMENT\_API\_KEY not found in .env file. Exiting.")
* exit()
* print(f"--- Definitive Santiment API Test for {SLUG} ---")
* print(f"Using Key: ...{SANTIMENT\_API\_KEY[-4:]}")
* # This is the GraphQL query to get the metrics we need
* query = f"""
* query {{
* mvrv: getMetric(metric: "mvrv\_usd") {{
* timeseriesData(slug: "{SLUG}", from: "utc\_now-2d", to: "utc\_now", interval: "1d") {{ value }}
* }}
* social\_dominance: getMetric(metric: "social\_dominance\_total") {{
* timeseriesData(slug: "{SLUG}", from: "utc\_now-2d", to: "utc\_now", interval: "1d") {{ value }}
* }}
* }}
* """
* try:
* print("\n--- Testing Endpoint: Santiment GraphQL ---")
* response = requests.post('https://api.santiment.net/graphql',
* json={'query': query},
* headers={'Authorization': f'Apikey {SANTIMENT\_API\_KEY}'})
* print(f"Status Code: {response.status\_code}")
* if response.ok:
* json\_data = response.json()
* print("\n--- Parsing Response ---")
* if json\_data.get('data'):
* mvrv\_data = json\_data['data'].get('mvrv', {}).get('timeseriesData', [])
* social\_data = json\_data['data'].get('social\_dominance', {}).get('timeseriesData', [])
* mvrv\_value = mvrv\_data[-1]['value'] if mvrv\_data and mvrv\_data[-1] else "Not Available"
* social\_value = social\_data[-1]['value'] if social\_data and social\_data[-1] else "Not Available"
* print("\n--- Extracted Metrics ---")
* print(f"MVRV Ratio: {mvrv\_value}")
* print(f"Social Dominance: {social\_value}")
* if mvrv\_value != "Not Available" and social\_value != "Not Available":
* print("\n🎉 SUCCESS: All key metrics were extracted successfully.")
* else:
* print("\n⚠️ WARNING: One or more metrics were not available. This is likely a Santiment plan limitation.")
* print("Full Response:", json.dumps(json\_data, indent=2))
* else:
* print("❌ ERROR: API response did not contain a 'data' key.")
* print("Full Response:", json.dumps(json\_data, indent=2))
* else:
* print(f"❌ ERROR: API call failed.")
* print("Response Text:", response.text)
* except Exception as e:
* print(f"An error occurred: {e}")

**Local Development Workflow**

1. **Configure .env:** Create a .env file and fill in all API keys. For local testing, comment out the DATABASE\_URL line.
2. **Reset Environment:** Run docker-compose down -v to ensure a clean slate.
3. **Run Analysis:** Run docker-compose run --build --rm app python daily\_runner.py to build the app and populate the local database.
4. **Launch Dashboard:** Run docker-compose up --build to start all services and view the dashboard at http://localhost:8501.

**Deployment to Render**

1. **Configure Environment:** On your Render application service, go to the "Environment" tab and add all the required API keys and the production DATABASE\_URL as environment variables.
2. **Deploy Code:** Merge your dev branch into main and run git push origin main.
3. **Reset Database:** On the Render dashboard for your PostgreSQL service, go to "Settings" and reset the database to apply the final schema.
4. **Redeploy App:** Manually redeploy your application service on Render one last time to run the startup script against the fresh database.