**The 305 Crypto Forecast App: A Complete Guide**

**Version:** 2.0 (Advanced Data Integration) **Author:** Michel Faleni & Gemini

This document provides a complete, step-by-step guide to building the 305 Crypto Forecast application from a blank slate. It is designed for someone with no prior programming knowledge and includes the full code for every file, detailed instructions for setup, local testing, and deployment.

**Part 1: Introduction & Application Overview**

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

The 305 Crypto Forecast App is a professional-grade, automated financial data analysis platform. At its core, it is designed to gather a massive amount of diverse data for major cryptocurrencies (like Bitcoin, Ethereum, and XRP) and synthesize it into actionable insights. It is a powerful tool for any trader or analyst who wants to move beyond simple price charts and understand the complex forces that drive the crypto market.

**1.2: Core Capabilities**

The application is built on a sophisticated architecture that gives it a unique set of capabilities:

* **Multi-Source Data Aggregation:** The app doesn't just look at one thing. Every day, it automatically connects to a suite of professional data providers to get a 360-degree view of the market:
  + **Yahoo Finance:** For historical price and volume data.
  + **CoinGecko:** For fundamental data like market cap rank, all-time-highs, and community scores.
  + **CoinGlass:** For high-impact futures and derivatives data, including funding rates, open interest, and long/short ratios. This reveals what leveraged traders are betting on.
  + **Santiment:** For deep on-chain and social metrics like the MVRV ratio and daily active addresses, showing the underlying health of the network.
  + **LunarCrush:** For social intelligence data like the Galaxy Score™, which quantifies market hype and social media trends.
  + **NewsAPI:** For the latest global news headlines.
* **Advanced Technical Analysis:** The application automatically calculates a suite of the most important technical indicators, including RSI, MACD, Bollinger Bands, and the Ichimoku Cloud.
* **Dual-Model Forecasting:** It uses two different machine learning models to predict future prices: **Prophet**, a robust time-series model developed by Meta, and a more complex **LSTM (Long Short-Term Memory)** deep learning model.
* **AI-Powered Analyst:** The true "brain" of the application is a call to OpenAI's **GPT-4**. The app gathers all the data for the day—every forecast, indicator, and news headline—and presents it to the AI. The AI is then prompted to act as an expert financial analyst, providing a written summary and a data-driven hypothesis for the day's market movements.
* **Automated Daily Operation:** The entire process is fully automated. The application is deployed on a cloud server (Render) and is triggered by an external service once every 24 hours. It runs, gathers all the new data, and updates its database and dashboard without any human intervention.
* **Persistent Memory & Learning:** Every daily forecast and all associated data are saved to a permanent **Postgres database**. This creates an ever-growing historical record.
* **Human-in-the-Loop Feedback System:** The dashboard includes a unique feedback feature. You, the user, can **"Confirm" or "Deny"** the AI's daily analysis. If you deny it, you can provide a written correction. This feedback is saved to the database alongside the original forecast. Over time, this creates an invaluable, expert-labeled dataset that can be used to **fine-tune** the AI model, effectively teaching it to become a better analyst based on your expert knowledge.
* **Dockerized for Reliability:** The entire application is packaged into a **Docker container**. This is the industry-standard for building reliable software. It guarantees that the application runs in a perfect, clean, and identical environment on your local computer and on the live server, eliminating dependency conflicts and "it works on my machine" problems.

This completes Part 1. In the next part, we will begin the step-by-step process of building the application, starting with the necessary software installation.

**Part 2: Prerequisites (Software Installation)**

Before we write a single line of code, we need to set up your computer with the essential tools that professional developers use. This guide is tailored for a Mac.

**2.1: Install a Code Editor (Visual Studio Code)**

First, you need a place to write and manage your code files. VS Code is the industry standard—it's free, powerful, and makes reading and writing code much easier with features like syntax highlighting.

* **Action:** Go to the official website, download, and install VS Code.
* **Link:** <https://code.visualstudio.com/>

**2.2: Install Homebrew**

Homebrew is a "package manager" for macOS. Think of it as an App Store for powerful command-line tools. It makes installing the rest of our software incredibly simple.

* **Action:** Open your **Terminal** app (you can find it in /Applications/Utilities/ or by searching with Spotlight Cmd + Space). Paste the single command below and press Enter. It will ask for your Mac's login password to proceed.

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

**2.3: Install Git**

Git is the version control system we use to save and manage different versions of our code. It's like a "save" button for your entire project, allowing you to track changes and create different branches (like main and dev).

* **Action:** In the Terminal, now that Homebrew is installed, run:

brew install git

**2.4: Install Docker Desktop**

Docker is the most important tool for ensuring our application works everywhere. It lets us package our app into a "container"—a perfect, clean, miniature computer that has the exact version of Python and all the correct libraries installed. This solves the "it works on my machine but not on the server" problem forever.

* **Action:** Go to the official website, download, and install Docker Desktop for your Mac. Choose the "Apple Silicon" version if you have a newer Mac with an M1/M2/M3 chip, or "Intel Chip" for an older one.
* **Link:** <https://www.docker.com/products/docker-desktop/>
* **Crucial Final Step:** After installing, you must **open the Docker Desktop application** from your Applications folder and let it start up. You will see a small whale icon in your top menu bar when it's running. The docker commands will not work in the terminal until this application is running in the background.

This completes Part 2. In the next part, we will set up the project folder and the Git repository.

**Part 3: Initial Project & Repository Setup**

Now that the software is installed, we will create the project folder and set it up as a Git repository. This is essential for tracking changes and deploying your application.

**3.1: Create the Project Folder**

First, we need a dedicated folder to hold all of our code. We will create this on your Desktop for easy access.

* **Action:** Open your **Terminal** app and run these commands one by one, pressing **Enter** after each line.
* cd ~/Desktop
* mkdir clean-crypto
* cd clean-crypto
* **What this does:** The first command navigates to your Desktop. The second command, mkdir, creates a new folder named clean-crypto. The final command navigates you inside that new folder. Your terminal is now operating from within your project directory.

**3.2: Initialize the Git Repository**

Now, we will turn this regular folder into a Git repository. This creates a hidden .git folder inside clean-crypto that will track all of your file changes. We will also create the two branches we need: main for your stable, live application, and dev for building new features.

* **Action:** Inside the clean-crypto folder, run these commands one by one in your terminal.
* git init
* git branch -m main
* git checkout -b dev
* **What this does:** git init initializes the repository. git branch -m main renames the default branch to main. git checkout -b dev creates a new branch called dev (which is an exact copy of main) and immediately switches you to it. You are now safely on your development branch, ready to start adding files.

**3.3: Connect to GitHub**

The final step is to link your local folder to a remote repository on GitHub. This creates a backup of your code in the cloud and is necessary for deploying to Render.

* **Action:**
  1. Go to [**GitHub.com**](https://github.com) in your web browser and create a **new, empty repository**. Name it something like 305-Crypto-Forecaster. Do **not** initialize it with a README or any other files.
  2. After creating the repository, GitHub will show you a page with commands under the title "...or push an existing repository from the command line".
  3. **Copy and paste** those two lines into your terminal and run them. They will look like this:
  4. git remote add origin https://github.com/your-username/your-repo-name.git
  5. git push -u origin dev
* **What this does:** The first command tells your local repository where its "home" on the internet is. The second command pushes your empty dev branch to GitHub, establishing the connection.

This completes Part 3. In the next part, we will gather all the necessary API keys and create the secret .env file to store them.

**Part 4: Acquiring API Keys & Database URL**

Your application needs several secret keys and a database to function.

**4.1: Create API Accounts**

Your application pulls data from five different professional sources. You need to create an account for each one to get a unique API key, which is like a password for your application.

* **Action:** Go to the website for each service listed below, sign up for an account, and find your API key. Copy each key into a temporary, safe place like a text note.
  + **OpenAI:** For the AI analyst. You will need to add a payment method to access the API.
    - **Link:** <https://platform.openai.com/>
  + **NewsAPI:** For news headlines.
    - **Link:** <https://newsapi.org/>
  + **CoinGecko:** For fundamental data. The free "Demo" plan is sufficient.
    - **Link:** <https://www.coingecko.com/en/api>
  + **Santiment:** For on-chain and social metrics. The free tier will give you an API key.
    - **Link:** <https://santiment.net/>
  + **LunarCrush:** For social intelligence data. This requires a paid subscription to get the data we need.
    - **Link:** <https://lunarcrush.com/>

**4.2: Create the Database on Render**

Your application needs a permanent place to store its daily forecasts. We will use a free Postgres database hosted on Render.

* **Action:**
  1. Go to your **Render Dashboard**, click **New + > Postgres**.
  2. Name it crypto-forecast-db and create it in the **Free** tier.
  3. Once it's ready, go to its **Connect** tab.
  4. Scroll down and find the **External Database URL**. This is the address your computer will use to talk to the database. **Copy this URL.**

**4.3: Create the .env File**

This file is a critical part of the project. It stores all your secrets for local testing. It is **never** uploaded to GitHub, so your keys remain private.

* **Action:** In your clean-crypto folder, create a new file named exactly .env using VS Code. Paste the following content into it, replacing all the placeholders with your actual keys and the **External Database URL** you just copied from Render.
* # API Keys
* OPENAI\_API\_KEY=sk-YourActualOpenAIKeyHere
* NEWS\_API\_KEY=YourActualNewsAPIKeyHere
* COINGECKO\_API\_KEY=YourActualCoinGeckoKeyHere
* SANTIMENT\_API\_KEY=YourActualSantimentKeyHere
* LUNARCRUSH\_API\_KEY=YourActualLunarCrushKeyHere
* # Database Connection (Use the External URL and add +psycopg and ?sslmode=require)
* DATABASE\_URL="postgresql+psycopg://user:password@host/database\_name?sslmode=require"

**Crucial Formatting Notes:**

* There should be **no spaces** around the = sign.
* There should be **no quotation marks** (" or ') around your keys or the URL.
* For the DATABASE\_URL, make sure you add +psycopg after postgresql:// and ?sslmode=require at the very end.

This completes Part 4. In the next part, we will create the configuration files for Docker and Git.

**Part 5: Project Configuration Files**

Now that you have your secrets, we will create all the configuration files that tell your computer and the Render servers how to build and run the application.

**Action:** For each file below, create a new file with the specified name in your clean-crypto folder using VS Code, and then copy and paste the complete, unabridged code into it.

**5.1: Dockerfile**

**Role:** The recipe for building your application's perfect, isolated environment. It specifies the base operating system, installs necessary tools, copies your code, and sets the final command to run.

# 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"]

**5.2: docker-compose.yml**

**Role:** A simple configuration file that makes it easy to run your Docker container locally. It tells Docker to build from your Dockerfile and to load your secret keys from the .env file.

services:

app:

build: .

image: crypto-forecaster

env\_file:

- .env

**5.3: .dockerignore**

**Role:** Tells Docker which files to ignore when building the container. This keeps the container small, fast, and secure by excluding local secrets, temporary data, and environment folders.

# 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

**5.4: .gitignore**

**Role:** Tells Git which files to ignore so they are never uploaded to your public GitHub repository. This is a critical security step to protect your .env file.

# 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/

**5.5: requirements.txt**

**Role:** Lists all the Python libraries that the project depends on. This file is used by pip to install the correct environment inside the Docker container.

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

**4.6: start.sh**

**Role:** The startup script for the deployment. It ensures that the data generation script runs to completion *before* the web server starts.

#!/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

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

streamlit run dashboard\_app.py

This completes Part 5 of the guide. In the next part, we will create the core Python scripts for the application.

**Part 6: The Core Python Scripts**

Now we will create the Python files that contain the application's logic.

**Action:** For each file below, create a new file with the specified name in your clean-crypto folder using VS Code, and then copy and paste the complete, unabridged code into it.

**6.1: db\_utils.py**

**Role:** This is the database manager. It's the only file that knows how to talk to your Postgres database. It defines the table structure, initializes the database, and handles all saving, loading, and updating of records.

import os

import pandas as pd

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

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

if not DATABASE\_URL:

raise Exception("DATABASE\_URL environment variable is not set.")

engine = create\_engine(DATABASE\_URL)

metadata = MetaData()

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),

# --- Placeholder for Future Data ---

Column('Exchange\_Net\_Flow', Float),

# --- AI Analysis & Feedback ---

Column('analysis\_summary', String),

Column('analysis\_hypothesis', String),

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

Column('user\_feedback', String),

Column('user\_correction', String)

)

def init\_db():

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

try:

inspector = inspect(engine)

if not inspector.has\_table('forecasts'):

print(" [INFO] 'forecasts' table not found. Creating table...")

metadata.create\_all(engine)

print(" [SUCCESS] 'forecasts' table created.")

else:

print(" [INFO] 'forecasts' table already exists.")

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'])

results\_df.to\_sql('forecasts', engine, if\_exists='append', index=False)

print(f" [SUCCESS] Saved {len(results\_df)} 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

This completes Part 6 of the guide. In the next part, we will create the data\_utils.py script, which is the heart of your data aggregation pipeline.

**Part 7: The Data Aggregation Script (data\_utils.py)**

**Role:** This script is the heart of your data collection engine. It contains all the functions needed to call the different external APIs (CoinGecko, CoinGlass, Santiment, LunarCrush) and Yahoo Finance. Its main function, fetch\_data, orchestrates all these calls, calculates all the technical indicators, and combines everything into a single, clean Pandas DataFrame, ready for analysis.

**Action:** Create a new file named data\_utils.py in your clean-crypto folder and paste the complete, unabridged code below into it.

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

# --- API HELPER FUNCTIONS ---

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

"""Fetches futures data for a given symbol from the CoinGlass API."""

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

headers = {'accept': 'application/json'}

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

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

try:

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

oi\_url = f"https://open-api.coinglass.com/public/v2/open\_interest?ex=Binance&symbol={api\_symbol}"

ls\_url = f"https://open-api.coinglass.com/public/v2/long\_short?ex=Binance&symbol={api\_symbol}"

funding\_res = requests.get(funding\_url, headers=headers)

if funding\_res.ok and funding\_res.json().get('data'):

data['funding\_rate'] = funding\_res.json()['data'][0].get('rate', 0.0) \* 100

oi\_res = requests.get(oi\_url, headers=headers)

if oi\_res.ok and oi\_res.json().get('data'):

data['open\_interest'] = oi\_res.json()['data'][0].get('openInterest', 0.0)

ls\_res = requests.get(ls\_url, headers=headers)

if ls\_res.ok and ls\_res.json().get('data'):

data['long\_short\_ratio'] = ls\_res.json()['data'][0].get('longShortRatio', 0.0)

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

return data

except Exception as e:

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

return data

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

"""Fetches on-chain/social data for a given slug directly from the Santiment GraphQL API."""

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()

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

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

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

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

metrics = {

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

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

'daily\_active\_addresses': daa\_data[-1]['value'] if daa\_data 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)

# Technical Indicators

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()

# Integrate ALL Advanced Data Sources

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)

# Add all data points to the DataFrame, ensuring fallbacks are numeric

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['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)

# We don't have a free Glassnode source for this, so we'll add a placeholder

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()

This completes Part 7 of the guide. In the next part, we will create the analytical engine scripts.

**Part 8: The Analytical Engine Scripts**

**Action:** For each file below, create a new file with the specified name in your clean-crypto folder using VS Code, and then copy and paste the complete, unabridged code into it.

**8.1: forecasting.py**

**Role:** This script is your forecasting engine. It contains the functions for both the **Prophet model** (for a robust time-series forecast) and the **LSTM model** (a more complex deep learning model). It takes the historical price data and outputs future price predictions.

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

**8.2: sentiment.py**

**Role:** This script handles all interactions with the NewsAPI. It fetches recent headlines for a given cryptocurrency and then uses a simple GPT-4 prompt to generate a sentiment score from -1.0 (very negative) to 1.0 (very positive). It also returns the top articles for the AI analyst to use.

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, []

**8.3: analyst.py**

**Role:** This script is the AI "brain" of the application. It contains the function that takes all the daily data (prices, forecasts, sentiment, indicators, and news), constructs a detailed prompt for the GPT-4 model, sends the request, and parses the structured JSON analysis that comes back.

import os

import json

import openai

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

"""

Takes a dictionary of the day's data, sends it to the GPT-4 API for analysis,

and returns a structured JSON response with validated URLs.

"""

print(" [INFO] Contacting AI Analyst (GPT-4)...")

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

actual = daily\_briefing\_data.get("actual\_price", 0)

prophet = daily\_briefing\_data.get("prophet\_forecast", 0)

sentiment = daily\_briefing\_data.get("sentiment\_score", 0)

rsi = daily\_briefing\_data.get("rsi", 0)

funding\_rate = daily\_briefing\_data.get("funding\_rate", 0)

headlines = daily\_briefing\_data.get("top\_headlines", [])

delta = actual - prophet

delta\_percent = (delta / actual) \* 100 if actual != 0 else 0

headline\_text = "\n".join([f"- {h['title']}" for h in headlines]) if headlines else "No headlines available."

prompt = f"""

Here is the daily market data for {coin}:

- Actual Closing Price: ${actual:,.2f}

- AI Forecasted Price: ${prophet:,.2f}

- Delta (Actual - Forecast): ${delta:,.2f} ({delta\_percent:.2f}%)

- Market News Sentiment Score: {sentiment:.2f}

- RSI (Relative Strength Index): {rsi:.2f}

- Funding Rate: {funding\_rate:.4f}%

- Top News Headlines Provided:

{headline\_text}

Please act as an expert financial analyst. Your task is to provide a concise, data-driven analysis.

Based \*only\* on the data provided, generate a JSON object with the following three keys:

1. "summary": A one-sentence summary of the day's forecast accuracy.

2. "hypothesis": A 2-3 sentence hypothesis explaining the most likely reason for the delta, citing specific data points (e.g., "negative sentiment", "overbought RSI > 70", "high funding rates") to support your hypothesis.

3. "influential\_headline\_titles": An array of up to 3 strings, where each string is the exact title of a headline from the list provided that you believe was most influential.

"""

try:

client = openai.OpenAI()

completion = client.chat.completions.create(

model="gpt-4-turbo",

messages=[

{"role": "system", "content": "You are a helpful financial analyst that provides structured data in JSON format."},

{"role": "user", "content": prompt}

],

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

temperature=0.2

)

response\_content = completion.choices[0].message.content

analysis\_from\_ai = json.loads(response\_content)

final\_headlines = []

ai\_titles = analysis\_from\_ai.get("influential\_headline\_titles", [])

for title in ai\_titles:

for original\_headline in headlines:

if original\_headline["title"] == title:

final\_headlines.append(original\_headline)

break

analysis\_to\_save = {

"summary": analysis\_from\_ai.get("summary"),

"hypothesis": analysis\_from\_ai.get("hypothesis"),

"news\_links": json.dumps(final\_headlines)

}

print(" [SUCCESS] AI analysis generated and URLs verified.")

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 due to an API error.",

"hypothesis": str(e),

"news\_links": json.dumps([])

}

This completes Part 8 of the guide. In the next part, we will create the final two Python scripts: the main daily\_runner.py that orchestrates everything, and the dashboard\_app.py that displays it all.

**Part 9: The Main Application Scripts**

**Action:** For each file below, create a new file with the specified name in your clean-crypto folder using VS Code, and then copy and paste the complete, unabridged code into it.

**9.1: daily\_runner.py**

**Role:** This is the main orchestrator or "engine" of the application. It runs once per day. Its job is to call all the other modules in the correct order: it initializes the database, calls data\_utils.py to fetch all the raw data, calls forecasting.py and sentiment.py to perform the analysis, calls analyst.py to generate the AI narrative, and finally, calls db\_utils.py to save the complete, final record to your database.

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 ---

# This checks all required keys to ensure the script can run successfully.

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]

# Run forecasts and sentiment analysis

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),

"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),

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

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

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

"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()

**9.2: dashboard\_app.py**

**Role:** This is the main user-facing web application, built with Streamlit. Its only job is to connect to the database, load the latest data, and display all the charts, metrics, and analyses. It also handles the user feedback form.

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

# --- 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

# --- 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'

]

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")

with st.expander("❓ \*\*Explain the Sentiment Score\*\*"):

st.info(

"""

The \*\*Sentiment Score\*\* is an aggregated value from a natural language processing (NLP) analysis of recent news articles.

- \*\*A positive score\*\* (e.g., > 0.10) indicates a \*\*bullish sentiment\*\* towards the cryptocurrency.

- \*\*A negative score\*\* (e.g., < -0.10) indicates a \*\*bearish sentiment\*\*.

- \*\*A score close to zero\*\* indicates a \*\*neutral market sentiment\*\*.

This score is a key fundamental indicator, as news and public opinion often precede price movements.

"""

)

st.header("Daily AI Analysis")

with st.container(border=True):

summary = coin\_forecast.get('analysis\_summary', 'Analysis not available.')

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

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

st.subheader("Today's Summary")

st.markdown(summary)

st.subheader("Analyst's Hypothesis")

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.")

st.header("Professional Grade Market Indicators")

with st.container(border=True):

st.subheader("Futures & Derivatives Data (from CoinGlass)")

cg\_col1, cg\_col2, cg\_col3 = st.columns(3)

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')

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")

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

san\_col1, san\_col2, san\_col3 = st.columns(3)

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')

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")

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")

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:

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

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

st.info(

"""

\*\*Reasoning:\*\* These indicators help identify the current trend and volatility.

- \*\*SMA/EMA:\*\* A price above its moving average suggests an uptrend, while a price below suggests a downtrend. Crossovers can signal a change in trend.

- \*\*Bollinger Bands:\*\* The bands widen during high volatility and narrow during low volatility. A price touching the upper band may suggest it's overbought, while touching the lower band may suggest it's oversold.

"""

)

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

with tech\_col1:

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

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

st.info(

"""

\*\*Reasoning:\*\* RSI measures the speed and change of price movements to identify overbought or oversold conditions.

- \*\*Action:\*\* A reading above 70 suggests the asset may be overbought and due for a correction. A reading below 30 suggests it may be oversold and poised for a rebound.

"""

)

st.subheader("Stochastic Oscillator")

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

st.info(

"""

\*\*Reasoning:\*\* This momentum indicator compares a specific closing price to a range of its prices over time.

- \*\*Action:\*\* Like RSI, readings above 80 indicate overbought conditions, while readings below 20 indicate oversold conditions. Crossovers between the %K and %D lines can also be used as buy or sell signals.

"""

)

with tech\_col2:

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

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

st.info(

"""

\*\*Reasoning:\*\* MACD is a trend-following momentum indicator that shows the relationship between two moving averages.

- \*\*Action:\*\* When the MACD line (blue) crosses above the Signal line (orange), it's a bullish signal, suggesting it may be a good time to buy. When it crosses below, it's a bearish signal.

"""

)

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

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

st.info(

"""

\*\*Reasoning:\*\* OBV uses volume flow to predict price changes. The idea is that volume precedes price.

- \*\*Action:\*\* A rising OBV indicates positive volume pressure that can confirm an uptrend. A falling OBV suggests negative pressure that could signal a downtrend.

"""

)

st.subheader("Ichimoku Cloud")

st.line\_chart(chart\_data[['Ichimoku\_a', 'Ichimoku\_b', 'Close']])

st.info(

"""

\*\*Reasoning:\*\* This is an all-in-one indicator that provides information on support, resistance, trend direction, and momentum.

- \*\*Action:\*\* If the price is above the cloud, the overall trend is considered bullish. If the price is below the cloud, the trend is bearish. The cloud itself also acts as a dynamic zone of support or resistance.

- \*\*Ichimoku Cloud A vs. B:\*\* The cloud is formed by the Senkou Span A (`Ichimoku\_a`) and Senkou Span B (`Ichimoku\_b`). When \*\*A is above B\*\*, the cloud is typically green and signals a \*\*bullish trend\*\*. When \*\*B is above A\*\*, the cloud is red and signals a \*\*bearish trend\*\*. The cloud's thickness indicates the strength of the trend.

"""

)

else:

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

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.")

This completes Part 9 of the guide. In the final part, we will cover how to build and run the application locally, and how to deploy it to the cloud.

**art 10: Building, Running, and Deploying the Application**

Now that all the code and configuration files are in place, this final part of the guide will walk you through building and running the application on your local machine, and then deploying it to the cloud on Render.

**10.1: Building and Running Locally**

This process uses Docker to create a self-contained environment, ensuring that all the libraries from your requirements.txt are installed correctly and that your scripts run in a clean, consistent space.

* **Action:** For each step below, run the command in your terminal from the clean-crypto folder.
  1. **Build the Docker Image:** This command reads your Dockerfile, gathers all your project files, and builds the container image. This process can take several minutes the first time as it downloads the Python base image and installs all the libraries.
  2. docker-compose build
  3. **Run the Application:** Once the build is complete, this command starts your application. It will read your docker-compose.yml file, correctly load your secret keys from the .env file, and start the container.
  4. docker-compose up
  5. **What to Expect:** After running docker-compose up, you will see the log output from your application directly in your terminal.
     + First, you'll see the output from the daily\_runner.py script as it fetches data and saves it to the database. The first time it runs, it will create the database table.
     + After that finishes, you will see the message --- Starting Streamlit dashboard ---.
     + Finally, you'll see a message with a "Local URL", like http://localhost:8501.
  6. **View the Dashboard:** Open your web browser and navigate to the "Local URL" provided in the terminal (usually http://localhost:8501). You should see your full, unabridged dashboard with all the new professional-grade indicators populated with data.

**10.2: Deploying to Render**

Once you have confirmed that the application runs perfectly on your local machine, you are ready to deploy it to the cloud.

* **Action:**
  1. **Push Code:** Push your final, working dev branch to GitHub.
  2. git add .
  3. git commit -m "feat: Finalize Docker setup and all features"
  4. git push origin dev
  5. **Create Render Service:**
     + On Render, create a **New Web Service** and connect your GitHub repo.
     + **Name:** three05-crypto-dev
     + **Branch:** dev
     + **Runtime:** Select **Docker**. The Build/Start commands will disappear, which is correct.
  6. **Add Environment Variables:**
     + Go to the **Environment** tab and add all your API keys and the **Internal Database URL** from your Render Postgres instance.
  7. **Deploy:** Render will automatically deploy the new service. You can monitor the logs to watch the start.sh script run.

**10.3: Setting Up Daily Automation**

The final step is to set up a free cron job to automatically re-deploy your application every 24 hours, ensuring the data is always fresh.

* **Action:**
  1. On your Render Web Service, go to the **Settings** tab and find your **Deploy Hook URL**. Copy it.
  2. Go to a free cron job service like [cron-job.org](https://cron-job.org/).
  3. Create a new cron job that sends a request to your Deploy Hook URL once every day.

This completes the guide. Your application is now fully built, deployed, and automated.