# **Binance WebSocket Price Precision Capture: BTC & ETH - Documentation**

## **1. Introduction and Project Overview**

In the rapidly evolving world of cryptocurrency, access to accurate, real-time market data is paramount for traders, analysts, and developers building financial applications. This project, "Binance WebSocket Price Precision Capture: BTC & ETH," offers a robust and reliable solution for acquiring high-fidelity, real-time cryptocurrency price updates directly from the **Binance WebSocket API**. Beyond mere data collection, the system is designed to persist these critical price points with exceptional precision into a **PostgreSQL database**, a powerful relational database known for its reliability and advanced querying capabilities.

The primary objective is to establish a foundational data pipeline that not only captures every minute price tick but also provides immediate, actionable insights through a powerful querying tool. This tool allows users to retrieve the absolute latest price, historical prices at specific timestamps, and aggregate values such as high and low prices within a defined timeframe. This makes the project an indispensable starting point for anyone aspiring to develop sophisticated real-time crypto analytics dashboards, automated trading bots, or platforms for in-depth historical data analysis.

### **1.1 The Need for High-Precision Price Data**

Cryptocurrency markets operate 24/7 and are characterized by high volatility and rapid price fluctuations. Capturing every price movement, no matter how small, and associating it with a precise timestamp is crucial for:

* **Accurate Analysis:** Understanding true market behavior, identifying trends, and performing accurate backtesting of trading strategies.
* **Algorithmic Trading:** Trading bots rely on the most current and precise price data to execute orders at optimal times and minimize slippage.
* **Regulatory Compliance:** In some financial contexts, detailed historical data with high precision is required for auditing and compliance purposes.
* **Preventing Data Loss:** Relying on aggregated or delayed data can lead to missed opportunities or misinformed decisions. Direct WebSocket feeds provide the raw, unfiltered stream.

### **1.2 Key Capabilities**

This system is engineered with specific capabilities to address the demands of real-time market data:

* **Live Price Ingestion:** The core functionality involves establishing and maintaining a persistent connection to Binance's WebSocket API. This connection allows for the continuous streaming of real-time price updates for specified cryptocurrency pairs, initially focusing on BTCUSDT (Bitcoin vs. Tether) and ETHUSDT (Ethereum vs. Tether). The WebSocket protocol ensures low-latency, full-duplex communication, which is ideal for streaming market data.
* **High-Precision Storage:** Every incoming price tick, along with its exact timestamp, is meticulously stored in a PostgreSQL database. PostgreSQL's robust support for various data types, including NUMERIC for precise price values and TIMESTAMP WITH TIME ZONE for accurate timekeeping, ensures data integrity and precision. This granular storage is vital for any subsequent in-depth analysis.
* **Real-time Analytics:** The project includes a command-line interface (CLI) tool that enables users to query the collected data for immediate insights. This tool demonstrates practical applications of the stored data, allowing users to:
  + Retrieve the **absolute latest recorded price** for any monitored symbol, providing an up-to-the-second market snapshot.
  + Fetch the **price of a symbol at any given historical timestamp**, enabling point-in-time analysis.
  + Calculate and display the **highest and lowest prices** recorded within a specified recent time window (e.g., the last minute, last 5 minutes), offering quick volatility insights.

By combining low-latency data ingestion with robust, high-precision storage and immediate querying capabilities, this project provides a solid foundation for advanced cryptocurrency data applications.

## **2. Technical Architecture and Data Flow**

The system's architecture is designed for simplicity, efficiency, and reliability, ensuring a clear flow of real-time price data from the source to the persistent storage and then to the querying interface.

### **2.1 Architecture Overview**

The core idea is to create a reliable and accurate pipeline for live market data. By consuming data directly from Binance's low-latency WebSocket, we ensure that every price tick is captured. Storing this data in PostgreSQL allows for efficient querying and analysis, enabling both real-time monitoring and historical backtesting.

+---------------------+ +----------------------+ +---------------------+

| Binance WebSocket |----->| Python WebSocket |----->| PostgreSQL Database |

| (Live Price Stream) | | Client (main.py) | | (Prices Table) |

+---------------------+ | (Data Ingestion) | +---------------------+

+----------------------+ ▲

|

|

+----------------------+ +---------------------+

| Python Query Tool |<-----| PostgreSQL Database |

| (queries.py) | | (Prices Table) |

| (Data Analytics) | +---------------------+

+----------------------+

### **2.2 Component Breakdown and Data Flow**

1. **Binance WebSocket API (Live Price Stream):**
   * **Role:** This is the primary data source. Binance provides a WebSocket API that allows clients to subscribe to real-time market data streams, including "mini-ticker" streams that deliver continuous price updates for specific trading pairs. The WebSocket protocol is chosen for its low latency and persistent, full-duplex communication, which is ideal for streaming high-frequency data.
   * **Data Flow:** Price updates for subscribed symbols (BTCUSDT, ETHUSDT) are pushed from Binance's servers to our Python WebSocket client as soon as they occur.
2. **Python WebSocket Client (main.py - Data Ingestion):**
   * **Role:** This Python script acts as the data producer. It establishes and maintains the WebSocket connection to the Binance API. Upon receiving a new price update, it parses the incoming JSON message, extracts the relevant price and timestamp, and then pushes this data to the PostgreSQL database.
   * **Key Operations:**
     + **Connection Management:** Handles WebSocket connection, including subscription to desired market streams.
     + **Data Parsing:** Extracts symbol, price, and event\_time from the raw JSON messages.
     + **Database Insertion:** Calls a function (from db.py) to insert the parsed data into the prices table.
   * **Data Flow:** Parsed price data (symbol, price, timestamp) is sent to the PostgreSQL database for persistence.
3. **PostgreSQL Database (prices Table - Data Storage):**
   * **Role:** This is the central data repository. PostgreSQL is chosen for its robustness, ACID compliance, and excellent support for time-series data. The prices table is specifically designed (as defined in schema.sql) to store each price tick with high precision, including the exact timestamp of the event.
   * **Data Storage:** Each row in the prices table represents a single price update, containing the cryptocurrency symbol, the precise price value (using NUMERIC for accuracy), and the timestamp of the event.
   * **Data Flow:** Receives price data from main.py and stores it. It also serves as the data source for queries.py.
4. **Python Query Tool (queries.py - Data Analytics):**
   * **Role:** This Python script acts as the data consumer/analytics tool. It connects to the PostgreSQL database and executes various SQL queries to retrieve insights from the captured price data.
   * **Key Operations:**
     + **Database Connection:** Establishes a connection to the PostgreSQL database.
     + **Query Execution:** Runs predefined SQL queries to fetch the latest price, historical prices at specific timestamps, and high/low prices within timeframes.
     + **Result Presentation:** Prints the query results to the console in a user-friendly format.
   * **Data Flow:** Sends SQL queries to the PostgreSQL database and receives query results for display.

This architectural design ensures that the data ingestion path is fast and non-blocking, allowing the system to handle high volumes of real-time data. The decoupling of data ingestion from data querying means that analytics can be performed without impacting the live data capture process.

## **3. Tech Stack**

This project is built using a straightforward yet powerful and widely adopted combination of technologies, each selected for its specific strengths in handling real-time data and robust persistence.

* **Python:**
  + **Role:** The primary programming language for the entire project. Python's rich ecosystem of libraries makes it an excellent choice for WebSocket communication, data parsing, and interacting with databases.
  + **Key Libraries Used:**
    - websocket-client: A low-level WebSocket client for Python, enabling direct communication with the Binance WebSocket API.
    - psycopg2-binary: A PostgreSQL adapter for Python, providing the necessary interface to connect to and interact with the PostgreSQL database.
* **PostgreSQL:**
  + **Role:** A powerful, open-source object-relational database system. PostgreSQL is renowned for its reliability, feature robustness, data integrity, and advanced querying capabilities. It is particularly well-suited for storing time-series data like price ticks due to its efficient indexing and support for various data types, including NUMERIC for high-precision decimal values and TIMESTAMP WITH TIME ZONE for accurate time-series management.
  + **Why PostgreSQL?** Its strong ACID compliance ensures data consistency, and its extensibility (e.g., with extensions like TimescaleDB for more advanced time-series features, though not used here) makes it a future-proof choice.
* **pgAdmin (Optional but Recommended):**
  + **Role:** A popular and comprehensive open-source graphical administration and development platform for PostgreSQL. While not strictly required for the system to function, pgAdmin significantly simplifies database management tasks, allowing for visual inspection of data, execution of custom SQL queries, and monitoring of database activity without needing to use the command line.
* **Binance WebSocket API:**
  + **Role:** The official API provided by Binance for real-time market data. It offers a direct, low-latency stream of price updates, order book changes, and other market events.
  + **Why WebSocket?** Unlike traditional REST APIs that require polling, WebSockets provide a persistent, full-duplex connection, meaning data is pushed from the server to the client as soon as it's available. This is crucial for capturing every price tick in a fast-moving market.

This combination of technologies provides a stable, efficient, and scalable foundation for real-time cryptocurrency data capture and analysis.

## **4. Folder Structure**

The project's codebase is organized into a clear and logical folder structure, promoting modularity, readability, and ease of maintenance. Each file serves a distinct purpose within the overall system.

binance\_price\_capture/

├── db.py # Handles PostgreSQL database connection and data insertion logic.

├── main.py # The core script for establishing WebSocket connection and collecting live price data.

├── queries.py # Contains functions and logic for querying the collected price data from the database.

├── schema.sql # SQL script to define and create the 'prices' table in PostgreSQL.

├── README.md # This comprehensive project documentation.

└── summary.md # A brief, high-level summary of the project.

* **db.py**:
  + **Purpose:** This Python module is dedicated to handling all interactions with the PostgreSQL database. It encapsulates the logic for establishing a connection to the database and provides a function for inserting new price data records into the prices table. Separating database logic into this module ensures that main.py and queries.py don't need to directly manage connection details or SQL insertion statements, promoting cleaner code and easier maintenance.
* **main.py**:
  + **Purpose:** This is the heart of the real-time data capture component. It's the core script responsible for initiating and maintaining the WebSocket connection to the Binance API. It defines the logic for subscribing to specific cryptocurrency market streams (BTCUSDT and ETHUSDT), receiving real-time price updates, parsing the incoming JSON messages, and then utilizing the functions from db.py to persist this high-precision price data into the PostgreSQL database. This script is designed to run continuously to ensure uninterrupted data capture.
* **queries.py**:
  + **Purpose:** This Python module provides the analytical capabilities of the project. It contains functions and logic specifically designed for querying the collected price data from the PostgreSQL database. This includes functions to fetch the latest recorded price for a symbol, retrieve historical prices at precise timestamps, and calculate high/low values within specified timeframes. It demonstrates how to extract immediate insights from the stored data.
* **schema.sql**:
  + **Purpose:** This SQL script defines the schema for the prices table in the PostgreSQL database. It specifies the table structure, column names, data types (e.g., NUMERIC for price precision, TIMESTAMP WITH TIME ZONE for accurate time), and any constraints or indexes. This file is executed once during the setup phase to prepare the database for data storage.
* **README.md**:
  + **Purpose:** This comprehensive documentation file provides a detailed overview of the project, its features, architecture, setup instructions, usage guides, and contribution guidelines. (This is the document you are currently reading).
* **summary.md**:
  + **Purpose:** A brief, high-level summary of the project. It offers a quick overview of the system's purpose and key components, useful for a rapid understanding or for presentation purposes.

This organized structure ensures that each part of the system is clearly defined and manageable, contributing to the overall robustness and maintainability of the project.

## **5. Setup Instructions**

To get the Binance price capture system up and running on your local machine, follow these detailed step-by-step instructions.

### **5.1 Prerequisites**

Before you begin the setup process, ensure that you have the following software installed on your system:

* **Python 3.9+:** The project is developed and tested with Python version 3.9 or newer. You can download it from the official Python website.
* **PostgreSQL:** You need a running PostgreSQL database instance. You have several options for installation:
  + **Local Installation:** Install PostgreSQL directly on your operating system (e.g., using Homebrew on macOS, apt on Ubuntu, or the official installer on Windows).
  + **Docker Container:** For a more isolated and reproducible environment, you can run PostgreSQL in a Docker container. A simple docker-compose.yml for PostgreSQL is often available online or can be created easily.
* **pgAdmin (Optional but Recommended):** While not strictly necessary for the system's operation, pgAdmin is a graphical tool that greatly simplifies PostgreSQL database management, allowing you to visually inspect tables, run queries, and manage users.

### **5.2 Step-by-Step Setup Guide**

**Clone the Repository:** Begin by cloning the project's source code from its GitHub repository to your local machine. Open your terminal or command prompt and execute the following commands:  
Bash  
git clone https://github.com/karthikeyapranav/Binance-WebSocket-Price-Precision-Capture-BTC-ETH-.git

cd Binance-WebSocket-Price-Precision-Capture-BTC-ETH-

**Create PostgreSQL Database and User (if needed):** If you don't already have a PostgreSQL database and a dedicated user for this project, you'll need to create them. It's good practice to use specific credentials for each application. You can do this using the psql command-line client (which comes with PostgreSQL) or via pgAdmin.  
**Example using psql:** Open your terminal and connect to your PostgreSQL server (you might need to use sudo -u postgres psql or just psql if your user has direct access). Then run these SQL commands:  
SQL  
CREATE DATABASE crypto;

CREATE USER postgres WITH PASSWORD 'your\_secure\_password'; -- Replace with a strong, unique password

GRANT ALL PRIVILEGES ON DATABASE crypto TO postgres;

\q -- Exit psql

* + **CREATE DATABASE crypto;**: Creates a new database named crypto.
  + **CREATE USER postgres WITH PASSWORD 'your\_secure\_password';**: Creates a new user named postgres (or any name you prefer) with the specified password. **Crucially, replace 'your\_secure\_password' with a strong, unique password.**
  + **GRANT ALL PRIVILEGES ON DATABASE crypto TO postgres;**: Grants all necessary permissions on the crypto database to the newly created user.

**Create the prices Table:** The schema.sql file contains the SQL commands to create the prices table, which is where all the captured cryptocurrency price data will be stored. Execute this script against your newly created crypto database.  
Bash  
# Replace 'your\_password' with the secure password you set in the previous step.

# Adjust '-p 5433' if your PostgreSQL is running on a different port (default is 5432).

psql -U postgres -d crypto -p 5433 -f schema.sql

* + -U postgres: Specifies the PostgreSQL user to connect as.
  + -d crypto: Connects to the crypto database.
  + -p 5433: Specifies the port number. If your PostgreSQL is running on the default port, you can omit this or use 5432.
  + -f schema.sql: Tells psql to execute the SQL commands from the schema.sql file.

**Update Database Credentials:** The Python scripts (db.py and queries.py) need to know your PostgreSQL database credentials to connect. You must manually update these files with the password and port you configured.  
Open db.py and queries.py in a text editor and locate the psycopg2.connect line. Modify it to match your setup:  
Python  
# Example snippet from db.py and queries.py

conn = psycopg2.connect(

dbname="crypto",

user="postgres",

password="your\_secure\_password", # <--- UPDATE THIS with your actual password

host="localhost",

port="5433" # <--- UPDATE THIS if your PostgreSQL port is different

)

* + **Ensure the password and port values are accurate.** Incorrect credentials will prevent the Python scripts from connecting to the database.

**Install Python Dependencies:** Finally, install the necessary Python libraries that the project relies on. These are specified in the requirements.txt implicitly.  
Bash  
pip install websocket-client psycopg2-binary

* + websocket-client: Enables WebSocket communication with Binance.
  + psycopg2-binary: The Python adapter for PostgreSQL.

Once all these steps are completed, your environment will be fully set up, and you'll be ready to run the data capture and querying scripts.

## **6. Running the Scripts**

With the setup complete, you can now start capturing live price data and querying it for insights. You will typically run these components in separate terminal windows.

### **6.1 1. Start WebSocket Data Capture (Producer)**

This script will connect to Binance's WebSocket API and continuously stream real-time price updates, inserting them into your PostgreSQL database.

Open your first terminal window, navigate to the project's root directory, and run:

Bash

python main.py

* **Expected Output:** You will see continuous console output indicating that the WebSocket connection has been established and that price updates for BTCUSDT and ETHUSDT are being received and successfully inserted into the database. This script is designed to run indefinitely to collect data.
* **Keep it Running:** Leave this terminal window open and main.py running to ensure continuous data capture.

### **6.2 2. Query Data from Database (Consumer/Analytics)**

While main.py is capturing data, you can simultaneously run queries.py in a separate terminal to fetch and analyze the collected data in real-time.

Open a **new terminal window**, navigate to the project's root directory, and run:

Bash

python queries.py

* **Expected Output:** The script will execute a set of predefined queries and print the results directly to the console. You will see output similar to:
  + The latest recorded price for BTCUSDT and ETHUSDT.
  + The price of a specific symbol at a historical timestamp (e.g., the price 1 minute ago).
  + The highest and lowest prices recorded for a symbol within a recent time window (e.g., the last 1 minute).
* **Customization:** You can modify the queries.py script to test different timeframes (e.g., last 5 minutes, last 10 minutes) or to query for other symbols if you extend main.py to capture more pairs.

### **6.3 Viewing Data in pgAdmin (Optional)**

For a more visual and interactive way to inspect your captured data and run custom SQL queries, pgAdmin is an excellent tool.

1. **Open pgAdmin:** Launch the pgAdmin application on your system.
2. **Connect to your PostgreSQL server:** If you haven't already, add a new server connection in pgAdmin pointing to your local PostgreSQL instance (using localhost as host, 5433 or your port, and the postgres user credentials).
3. **Navigate to your crypto database:** In the pgAdmin browser tree, expand "Servers" -> "Your Server Name" -> "Databases" -> "crypto".
4. **Explore the prices table:** Continue expanding the tree: "Schemas" -> "public" -> "Tables" -> "prices".
5. **View Data:** Right-click on the prices table and select "View/Edit Data" -> "All Rows". This will open a data grid displaying all the captured price entries, allowing you to scroll through and visually verify the data.

**Run Custom SQL:** To execute your own SQL commands, right-click on the crypto database (or any table within it) and select "Query Tool...". In the query editor, you can type and execute SQL commands like:  
SQL  
-- Get the 10 most recent BTCUSDT prices

SELECT \* FROM prices WHERE symbol = 'BTCUSDT' ORDER BY timestamp DESC LIMIT 10;

-- Find the minimum and maximum price for ETHUSDT in the last 5 minutes

SELECT MIN(price), MAX(price) FROM prices WHERE symbol = 'ETHUSDT' AND timestamp >= NOW() - INTERVAL '5 minutes';

1. This visual interface is invaluable for debugging, exploring data, and developing new analytical queries.

## **7. Potential Use Cases**

This project serves as a robust foundational component that can be integrated into or extended for a wide array of applications within the cryptocurrency domain and beyond. Its ability to capture and query high-precision, real-time market data opens up numerous possibilities:

* **Real-time Price Monitoring Dashboards:** The captured data can feed directly into interactive web dashboards (e.g., built with Streamlit, Dash, Flask/React, or Grafana) to visualize live price movements, candlestick charts, and custom indicators. This provides traders and analysts with immediate insights into market dynamics.
* **Algorithmic Trading & Strategy Testing:** The high-precision live and historical data is invaluable for developing, backtesting, and executing algorithmic trading strategies. Algorithms can consume the real-time stream for trade execution and use the historical data for rigorous performance evaluation.
* **Crypto Analytics Platforms:** This system can be the data backbone for more sophisticated analytical tools. Users could build custom platforms to track market trends, measure volatility, analyze order book depth, identify arbitrage opportunities, or perform correlation studies between different assets.
* **Historical Data Analysis:** Researchers and data scientists can leverage the meticulously captured historical data for in-depth research on past market behavior, identifying patterns, and building predictive models. The precision of the timestamps is crucial for accurate historical reconstruction.
* **Automated Alerting Systems:** By continuously monitoring the incoming price stream and querying the database, you can implement systems that trigger alerts (e.g., via email, SMS, push notifications, or Discord webhooks) when prices cross predefined thresholds, exhibit unusual patterns, or when significant price changes occur within a short period.
* **Market Data Feeds for Other Applications:** The PostgreSQL database can act as a reliable internal market data feed for other applications within an organization, ensuring consistent and high-quality data across various services.
* **Compliance and Auditing:** For financial institutions dealing with crypto assets, maintaining a precise, auditable log of price movements is critical for regulatory compliance and internal auditing purposes.

By providing a solid foundation for data acquisition and storage, this project significantly reduces the barrier to entry for building complex and insightful cryptocurrency-related applications.

## **8. Next Steps & Enhancements**

This project, while robust in its current form, can be extended and enhanced in many exciting ways to add more features, improve resilience, and expand its utility.

* **Support for More Trading Pairs:**
  + **Enhancement:** Easily extend main.py to subscribe to and capture data for additional cryptocurrency symbols beyond BTCUSDT and ETHUSDT (e.g., XRPUSDT, ADAUSDT, SOLUSDT, etc.). This would involve modifying the WebSocket subscription logic and potentially the database schema if different data points are required for new pairs.
* **Build a Dashboard:**
  + **Enhancement:** Integrate the captured data with visualization tools to create an interactive price dashboard.
  + **Options:**
    - **Streamlit/Dash:** For quick and powerful Python-native dashboards.
    - **Grafana:** For robust monitoring and time-series visualization, integrating directly with PostgreSQL.
    - **Flask/React Frontend:** For a custom web application with more control over UI/UX.
* **Implement Advanced Analytics:**
  + **Enhancement:** Add more complex querying and analytical capabilities to queries.py or a new analytics module.
  + **Examples:** Calculate moving averages (SMA, EMA), Bollinger Bands, volatility metrics, relative strength index (RSI), or implement order book depth analysis if full order book streams are captured.
* **Threshold-Based Alerts:**
  + **Enhancement:** Develop a dedicated system to continuously monitor the incoming price data against predefined thresholds.
  + **Functionality:** Trigger alerts (e.g., email, SMS, push notifications, or integration with messaging platforms like Slack/Discord) when prices cross specific levels, exhibit rapid changes, or meet custom criteria.
* **Error Handling & Reconnection Logic:**
  + **Enhancement:** Improve the robustness of the WebSocket client in main.py with more sophisticated error handling.
  + **Functionality:** Implement automatic reconnection logic with exponential backoff in case of network interruptions or API disconnections to ensure continuous and uninterrupted data capture.
* **Containerization (Docker Compose):**
  + **Enhancement:** Package the PostgreSQL database and all Python scripts (main.py, queries.py) into Docker containers.
  + **Benefit:** Orchestrate them using Docker Compose for even easier setup, deployment, and environment consistency across different machines. This would simplify the entire setup process to a single docker-compose up command.
* **Scalability for High Volume:**
  + **Enhancement:** For extremely high-volume data (e.g., capturing many symbols, or full order book data), consider using a dedicated time-series database extension for PostgreSQL like **TimescaleDB** or a dedicated time-series database like InfluxDB.
  + **Further:** Implement a message queue (e.g., Kafka) between the WebSocket client and the database for better decoupling and buffering of data, similar to the previous project.

These enhancements can transform this foundational project into a comprehensive and production-ready market data platform.

## **9. Contributing**

Contributions, issues, and feature requests are highly welcome! This project is open-source, and community involvement is crucial for its growth and improvement. If you have ideas for enhancements, encounter any bugs, or wish to contribute code, please feel free to engage.

### **9.1 How to Contribute**

To contribute to this project, please follow the standard GitHub workflow:

1. **Open an Issue:**
   * If you've found a bug, have a feature idea, or want to propose a significant change, please open a new issue on the GitHub repository's issues page.
   * Provide a clear and detailed description of the problem or feature request. For bugs, include steps to reproduce the issue.
2. **Fork the Repository:**
   * Create a copy of the project in your personal GitHub account by clicking the "Fork" button on the repository page.
3. **Clone Your Fork:**

Clone your forked repository to your local machine:  
Bash  
git clone https://github.com/YOUR\_USERNAME/Binance-WebSocket-Price-Precision-Capture-BTC-ETH-.git

cd Binance-WebSocket-Price-Precision-Capture-BTC-ETH-

* + (Replace YOUR\_USERNAME with your GitHub username).

1. **Create a New Branch:**

Before making any changes, create a new branch for your feature or bug fix:  
Bash  
git checkout -b feature/your-awesome-feature

# Or for a bug fix:

git checkout -b bugfix/fix-issue-number

1. **Make Your Changes:**
   * Implement your feature or fix the bug. Ensure your code adheres to any existing style guidelines and is well-commented.
2. **Test Your Changes:**
   * Thoroughly test your changes to ensure they work as expected and don't introduce new issues.
3. **Commit Your Changes:**

Commit your changes with a clear, concise, and descriptive commit message:  
Bash  
git add .

git commit -m 'Add: Description of your changes'

1. **Push to Your Fork:**

Push your new branch to your forked repository on GitHub:  
Bash  
git push origin feature/your-awesome-feature

1. **Open a Pull Request (PR):**
   * Go to the original repository on GitHub. You should see a prompt to create a Pull Request from your new branch.
   * Provide a clear title and a detailed description of your changes. Reference any related issues (e.g., "Closes #123" if your PR fixes issue #123).

Your contributions are highly valued and help improve this project for everyone!

## **10. License**

This project is licensed under the **MIT License**.

The MIT License is a permissive free software license. It grants broad rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software. The only significant condition is that the original copyright notice and permission notice must be included in all copies or substantial portions of the software.

For the full text of the license, please refer to the LICENSE file in the project's root directory.

## **11. Acknowledgments**

We extend our sincere gratitude and appreciation to the following entities and communities whose work has been instrumental in the development and success of this project:

* **The Binance team:** For providing comprehensive, reliable, and low-latency WebSocket APIs that enable direct access to real-time market data.
* **The PostgreSQL community:** For developing and maintaining a robust, versatile, and highly reliable open-source relational database system that serves as the backbone for our data storage.
* **The developers of websocket-client and psycopg2:** For creating and maintaining their excellent Python libraries, which greatly simplified WebSocket communication and PostgreSQL database interactions, respectively.
* **The broader open-source community:** For continuously inspiring innovation, fostering collaboration, and providing the vast ecosystem of tools and knowledge that make projects like this possible.

Their collective efforts empower developers worldwide and are foundational to modern software development.