Contents

[Important File Links: 1](#_Toc212413105)

[Project Setup and Installation 1](#_Toc212413106)

[How to Run the Application 2](#_Toc212413107)

[AI Usage Transparency Note: 2](#_Toc212413108)

[Project requirement understanding: 3](#_Toc212413109)

[My understanding for definitions of Concepts used in project: 3](#_Toc212413110)

[System architecture and design and trade-offs: 5](#_Toc212413111)

[Front end visuals selections: 7](#_Toc212413112)

# Important File Links:

Repository Link: <https://github.com/VivekSawale98/QuantStream>

Backend: <https://github.com/VivekSawale98/QuantStream/tree/main/backend>

Frontend: <https://github.com/VivekSawale98/QuantStream/tree/main/frontend/quantstreamui>

Project Architecture Diagram: <https://github.com/VivekSawale98/QuantStream/blob/main/Project%20Documents/Project%20Architecture%20and%20Data%20Flow.drawio.svg>

Project Understanding Flow and Architecture TradeOffs Word Document: <https://github.com/VivekSawale98/QuantStream/blob/main/Project%20Documents/Project%20requirement%20undestanding%2C%20Architecture%20and%20flow.docx>

API Documentation (Swagger UI documentation): Can be accessed on <http://localhost:8000/docs> on project start

# Project Setup and Installation

Prerequisites:

- Python 3.10+ - Used 3.11.2

- Node.js and npm – NodeJS version v22.21.0, NPM version 10.9.4

Backend Setup:

1. Navigate to the `backend` directory: `cd backend`

2. Create and activate a virtual environment:

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

3. Install Python dependencies: `pip install -r requirements.txt`

Frontend Setup:

1. Navigate to the `frontend` directory: `cd frontend`

2. Install Node.js dependencies: `npm install`

3. Build the static frontend files: `npm run build`

4. To run frontend app: `npm run dev`

# How to Run the Application

1. Navigate to the project's **backend** directory.

2. Make sure your Python virtual environment (from the backend setup) is activated.

3. Run the single command:

python app.py

4. The server will start. Open your web browser and go to http://127.0.0.1:8000

The application will start, the data ingestor will begin collecting data in the background, and the dashboard will be ready to use.

# AI Usage Transparency Note:

I utilized a Large Language Model (LM Arena - gemini) as a development assistant throughout this project. Its primary uses included:

Boilerplate Code Generation: Generating initial setup code for FastAPI, React components, and the websocket-client library.

Conceptual Clarification: Acting as a sounding board to refine my understanding of quantitative concepts like the intercept's role in regression and the different definitions of "spread."

Debugging: Helping to diagnose and suggest solutions for common issues like CORS errors, Python ValueError exceptions, and race conditions in React useEffect hooks.

Documentation: Assisting in structuring and wording the README.md file for clarity and professionalism.

The core architectural decisions, feature implementation logic, and final code were directed and written by me, with the AI serving as a productivity and learning tool.

# Project requirement understanding:

1. I have to use provided Binance WebSocket streams emitting tick data
2. Ingest raw tick data from WebSocket and store it . Sampling should be based on selected timeframe
3. Based on sampled data we have to process/analyse that data
   1. Hedge Ratio (via Ordinary Least Squares Regression)
   2. The resulting Spread time series
   3. The Z-Score of the Spread
   4. Stationarity check via Augmented Dickey-Fuller (ADF) Test
   5. Rolling Correlation over a configurable window
4. Create frontend interactive dashboard which will consume the analytics data and show to user in real time
5. we have to allow alert functionality and download option and data upload option

Technical & Design:

1. The backend must be written in Python.
2. The system design must be loosely coupled and extensible, allowing for future additions of new analytics or data sources with minimal rework.
3. The application must be runnable via a single local command and be delivered with clear documentation (README.md) and an architecture diagram.

Based on above understanding I started the development

# My understanding for definitions of Concepts used in project:

Before starting the project I have to understand the trading concepts which are going to be used in the project. Below is my understanding about the concepts.

1. Tick

A tick represents one transaction on Binance. It tells us at what time in the world a certain amount of an asset was bought at what price.

1. Hedge Ratio

The Hedge Ratio is calculated between two related assets.

It answers: "How much of one asset do I need to buy to balance 1 unit of another asset to create a neutral position?"

1. The Spread

The hedge ratio we calculated is used here.

Formula: Spread = Price of Asset A - (Hedge Ratio \* Price of Asset B)

Why we need spread: If we plot two assets, let's say Pepsi and Coke, then we get two separate lines. They are related, so if a factor like the price of sugar rises, then they are both going to rise. But there are other factors which can affect a single asset. As they are related, they are eventually going to come back to their normal price relationship. These sudden fluctuations are what an investor sees as an opportunity to make a profit.

To trade this, we cannot just rely on simple subtraction because a 5-unit difference is not the same when prices are high vs. when they are low. This is not reliable.

Here, the Spread is used, which uses the hedge ratio to create one single line.

Now the scenario is: when the price of one asset goes up, it causes the Spread to vary. Let's say the price of Coke is overpriced; then what the investor does is sell the overpriced Coke and buy Pepsi, because they expect the Spread is going to be 0 again. So when the Spread reaches 0, the investor can close the position and make a profit.

1. Z-Score

What it is: A number that tells us how many standard deviations the Spread is away from its average.

Purpose: It's our main signal. A high Z-score (e.g., > 2) signals that the Spread is unusually high and might be a good time to sell. A low Z-score (e.g., < -2) signals it's unusually low and might be a good time to buy.

1. ADF (Augmented Dickey-Fuller) Test: (This is a check before you start trading)

In a spread strategy, we trade because we expect the spread between two assets is going to come to zero.

The ADF test is performed before trading to check whether that spread is likely going to come to 0 or if it is just going to increase continuously.

If the p-value is LOW (less than 0.05), then the spread is "stationary" (it is likely going to return to its average).

If the p-value is HIGH (more than 0.05), it is "non-stationary" (it is not likely to return).

1. Rolling Correlation: (This is a check to use while the trade is going on)

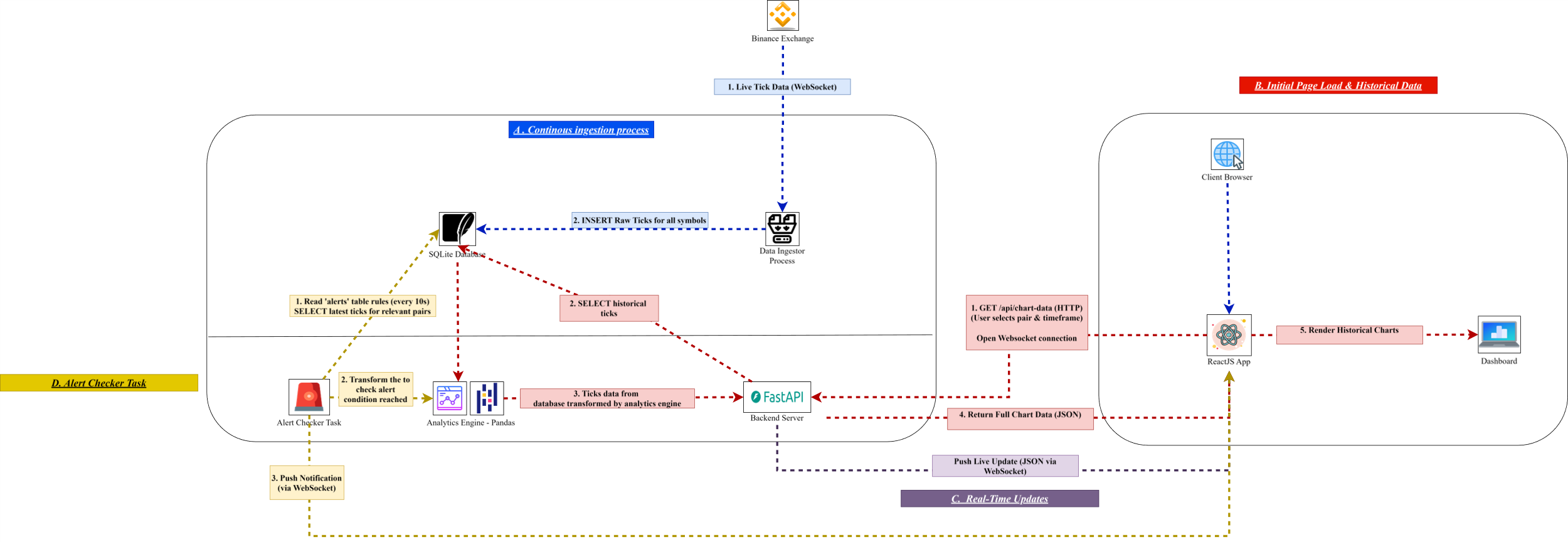
Correlation is a number from -1 to +1 that measures how much two things move together. +1 means they move in perfect lockstep. 0 means there's no connection.

"Rolling" means we calculate this based on a recent window, like the last 60 minutes.

If the correlation is high (for example, above 0.7), it confirms the relationship between the assets is strong and our strategy is still valid.

If the correlation is low (like 0.2), this is a warning that the relationship has weakened, and we should be careful or think about exiting the trade.

# System architecture and design and trade-offs:



We have four main components in the architecture:

1. Data Ingestor: An independent background process responsible solely for data collection.
2. Database
3. Backend API Server
4. Frontend Client

In above as we can see we are using Client-Server Architecture for separation of concerns, principles of loose coupling and scalability.

|  |  |
| --- | --- |
| Component | Technology |
| Backend Framework | FastAPI |
| Frontend Framework | React |
| Charting Library | TradingView Lightweight Charts |
| Database | SQLite |

Reason for selection of technology for component and trade-offs:

1. Backend: FastAPI

Why selected FastAPI instead of a framework like Django:

1. FastAPI is used for high-performance APIs with simple, less complex code. This allows for quick development and debugging while still achieving the required high performance.
2. A framework like Django is not as suitable for this specific case. It requires more boilerplate code, and more code can mean more chances for error. Django is a full-stack tool, and we only need the API part.
3. Our project involves using WebSockets for live updates. FastAPI is built on ASGI (Asynchronous Server Gateway Interface), which is designed to handle many connections and long-running tasks like a WebSocket asynchronously, without getting blocked.
4. Django, on the other hand, was built on WSGI (Web Server Gateway Interface), which is synchronous. To handle a WebSocket in Django, we would need an extra library like Django Channels, adding more complexity.
5. FastAPI's Dependency Injection system allows us to write cleaner and more modular code. For example, I can define the database connection I need in a function's signature, and FastAPI will manage it. This makes it very easy to swap things later, like using a different database for testing.
6. FastAPI also gives us automatic data validation and API documentation for free. By defining the data models with standard Python types, it validates all incoming requests and generates an interactive API doc page. This saves a lot of time and makes the API more robust.

2. Database: SQLite

Why selected SQLite:

1. We selected it because it is a serverless database that runs as a local file on the same machine. This is the most important reason because it allows us to meet the "single-command run" requirement. It requires no extra setup or installation from the user, unlike Docker or a separate PostgreSQL server.
2. The performance is more than enough for this project. Since it's a local file, read/write operations are extremely fast. For the small queries we need to run like getting the last 100 ticks, SQLite with a proper index is incredibly efficient and will not be the bottleneck in our 500ms latency budget.

3. Frontend: React with TradingView Lightweight Charts

1. Using a separate React app creates strong loose coupling. The frontend is its own project. If we have a scenario where we need to add a new chart, we just have to create a new component in the React app and have it use an existing or new API endpoint. We don't need to update the entire application, just the frontend code.
2. This loose coupling works both ways. If we have a scenario where we have to change the backend technology (e.g., from Python to Go), we can easily do it. As long as the new backend provides the same API endpoints, we can just "plug in" the new service and the React frontend will work without any changes.
3. We chose the TradingView Lightweight Charts library specifically because it is a professional, high-performance tool built for financial data. It already has all the components required to show candlestick charts with high-speed interactivity (zooming and panning). This follows the "clarity over complexity" principle by not reinventing a complex component from scratch.

# Front end visuals selections:

Mapping of Chart selection with goals and insights that trader can gain from the chart:

Chart 1: Asset Price Comparison – How are the two assets moving relative to each other? – To identify the major change

Is bitcoin and ethereum moving relative to together. If one is spiked without other then this is our point of interest.

Requirement Goal: To visually confirm the basic relationship between the two selected assets and identify any major structural changes or anomalies at a glance.

Chart Details:

Title: Asset Prices: [Base Symbol] vs. [Hedge Symbol]

Base Asset (Y): A CandlestickSeries (Green/Red) on the right price scale.

Why CandlestickSeries for base asset is we want to track how that base asset is moving with respect to hedge asset. Our focus is so we need it most information.

Hedge Asset (X): A LineSeries (Blue) on the left price scale.

Hedge is for reference. Hence lineseries is enough.

Business & Trading Point of View:

1. Concept
2. It tells trader how two assets value is moving relative to each other
3. Based on data of two related assets we can create baseline or regression line.
4. Regression line:
   1. Regression line tells expected price/predicts of Base asset based on Hedge asset
   2. Regression line is determined based on historical data
   3. Regression line equation: Y(Base asset Price) = a(initial price of base asset when value of hedge asset is 0 / intercept) + b(Hedge ratio) \* X( Hedge asset)
   4. Regression line for making prediction

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | X (BTC Price) | Y (Actual ETH) | Expected ETH Price = 200 + (0.05 \* X) | Difference  (Actual - Expected) |
| 10:01 | $60,000 | **$3,200** | 200 + (0.05 \* 60000) = **$3,200** | $0 |
| 10:02 | $61,000 | **$3,250** | 200 + (0.05 \* 61000) = **$3,250** | $0 |
| 10:03 | $62,000 | **$3,300** | 200 + (0.05 \* 62000) = **$3,300** | $0 |
| 10:04 | $61,500 | **$3,275** | 200 + (0.05 \* 61500) = **$3,275** | $0 |
| 10:05 | $63,000 | **$3,350** | 200 + (0.05 \* 63000) = **$3,350** | $0 |

* 1. our purpose is focus on base asset so we take detail of this and hedge is normal line theer will be regression line so most of the time base will be with regression line regression line is prediction base don historical data so if it not overalaps then this is the point that trader will look for
  2. For above based on available data hedge ratio is determined. Hedge ratio is how much of base asset worth 1 unit of hedge asset. It states what is change in base asset if hedge asset is increased by one unit. Y = a + b where X is 1
  3. Formula to calculate hedge ratio:
     1. Terminologies:

b: This is the Hedge Ratio (the slope we want to find).

Σ (Sigma): This is the symbol for "summation." It means "add up all the values."

X: The price of our Hedge Asset (e.g., BTC).

Y: The price of our Base Asset (e.g., ETH).

X̄ (X-bar): The average (mean) price of X over our historical data.

Ȳ (Y-bar): The average (mean) price of Y over our historical data.

n: The number of data points (candles) in our historical sample.

* + 1. **The Numerator (Top Part):**Σ ( (X - X̄) \* (Y - Ȳ) ) – calulcate X and Y covariance If both are above average, you get a positive number.

If both are below average, you get (negative \* negative) = a positive number.

If one is above and one is below, you get a negative number.

By summing these up, you get a single number that tells you the strength and direction of their joint movement. A large positive number means they tend to move up and down together.

* + 1. **The Denominator (Bottom Part):**Σ ( (X - X̄)² ): This is variance of hedge asset. Square purpose is to determine maximum deviation. So it adds up even if it is negative

1. What decision trader make based on above concept:

Trader views base asset value in normal case it should overlap with regression line. But if there is major change in base asset value then lets say it is increased or decreased then trader can spot from these chart and can proceed for further analysis on other charts to procced for trade.

If increased then can proceed for sell of base asset and buy of under priced hedge asset. Opposite in case if decreased in base asset.

1. How we have implemented regression line:

X\_with\_const\_for\_prediction = sm.add\_constant(aligned\_prices["X"])  
regression\_line = model.predict(X\_with\_const\_for\_prediction)

In above statsmodels of sm we use OLS Ordinary Least Squares regression.

This model contains hedge ratio and intercept and other important calculations.

We pass it hedge price so based on that we get regression value for base asset.

Chart 2: Spread & Z-Score Analysis – What is difference between Actual base asset price and regression line (Expected value of base asset Y) is **shows amount**. And Z-score is standard unit of measure in standard deviation to represent this spread amount.

Requirement Goal: To get identify concrete, statistically significant trading opportunities based on mean-reverting nature of the pair's relationship

Chart Details:

Title: Spread & Z-Score Analysis

Series 1: An AreaSeries for the Spread on the left price scale.

Series 2: A LineSeries for the Z-Score on the right price scale.

Annotations: Horizontal lines at Z-Score levels of +2.0, 0.0, and -2.0.

Series 3: Spread Mean line (Intercept a value)

Business & Trading Point of View:

1. Concept
   1. Spread: It is amount of difference between actual price of base asset and regression value (expected value of base asset): Y – b\*X

Visual Gap = Y\_actual - Y\_expected

Substitute the formula for Y\_expected: Visual Gap = Y\_actual - (a + b\*X)

Visual Gap = (Y\_actual - b\*X) - a

We know that Spread = (Y\_actual - b\*X).

Therefore:

Visual Gap = Spread – a

Spread = Visual Gap + a

Example:

The Goal: I want to predict my Ice Cream Sales (Y) based on the Temperature (X).

|  |  |  |
| --- | --- | --- |
| Day | Temperature(X) | Ice Cream Sales(Y) |
| 1 | 20°C | $110 |
| 2 | 25°C | $135 |
| 3 | 30°C | $160 |

Find the Sensitivity (Hedge Ratio b): I look at my historical data and see that for every 1°C increase, my sales go up by $5. So, my Hedge Ratio b is 5.

Find the Baseline Offset (Intercept a): I notice that simply multiplying the temperature by 5 (b\*X) is not enough. My actual sales are always about $10 higher than that simple calculation. This consistent, historical offset is my Intercept a.

Create the Prediction Model (Regression Line): My full prediction for the "expected" or "fair" price is therefore Expected Sales = a + (b \* X), which is 10 + (5 \* Temperature).

Spot an Anomaly: On a new day, the temperature is 28°C. My model predicts sales of 10 + (5 \* 28) = $150. But I actually sell $165.

The "Visual Gap": The difference between my actual sales and my predicted sales is $165 - $150 = $15. This is the "error" or "residual" of my model.

The "Hedged Spread": My code calculates a different number: Spread = Y - b\*X, which is $165 - (5 \* 28) = $25.

The Connection: The reason my "Hedged Spread" ($25) is different from my "Visual Gap" ($15) is because the Hedged Spread includes the baseline offset a. The relationship is Spread = Visual Gap + a ($25 = $15 + $10).

* 1. Z-Score: It is standard unit of measure for spread represent spread in unit of standard deviation

1. What decision trader make based on above concept:
   1. Based on chart 1 trader decide can have opportunity so based on spread and Z-score value it statistically confirms the opportunity
   2. From graph trader is going to track spread mean line and spread gap.
   3. So, the rule is:

If Z-Score > +2.0 (i.e., spread is significantly higher than its mean) -> SELL the spread (Sell Base, Buy Hedge).

If Z-Score < -2.0 (i.e., spread is significantly lower than its mean) -> BUY the spread (Buy Base, Sell Hedge).

Chart 3: Rolling Correlation

Requirement goal: This value is to ensure that whether the mean-reverting nature of the pair's relationship strong. Will two asset relation will come to normal or will it just keeps on moving away from each other. As trader profit is in it goes away but when it comes back.

Chart Details:

Title: [Window]-Period Rolling Correlation

Series: A single BaselineSeries with base value as 0.7.

Essential Configuration:

Fixed Y-Axis: The Y-axis must be fixed with a range from -1.0 to +1.0.

Threshold Line: A horizontal dashed line at a "healthy" correlation level, y = 0.7.

Business & Trading Point of View:

1. Concept:
   1. Rolling Correlation: whether the relation between base and hedge is strong means will it return to normal expected relation again.

+1.0: Perfect positive correlation. When one asset goes up, the other goes up in perfect lockstep.

0.0: No correlation. Their movements are random and unrelated.

-1.0: Perfect negative correlation. When one goes up, the other goes down.

1. What decision trader make based on above concept:
   1. The trader looks at the Spread & Z-Score Chart and sees a clear entry signal. The Z-Score has just dropped below -2.0. This is a signal to BUY the spread.
   2. The trader looks at the correlation chart and sees the line is high and stable, hovering around 0.85, well above the 0.7 threshold.
   3. The trader proceeds with confidence and executes the trade buy the base asset and sale the hedge asset
   4. Trader will also use this to monitor the position if score is more than 0.7 it keeps when either z-score or spread is reduced or Correlation is low than 0.7 then trader will exit

KPI:

Hedge Ratio:

Strategy: "For every 1 unit of the Base Asset I sell, I must buy hedge\_ratio units of the Hedge Asset."

Example:

The Z-Score hits +2.5 for ETH/BTC.

The hedge\_ratio is 0.06.

The trader decides to short 10 ETH.

To hedge this, they must buy 10 \* 0.06 = 0.6 BTC.

Their final trade is: SELL 10 ETH and BUY 0.6 BTC.

ADF Test p-value:

Based on 0.05 if more then rubber band/stationary spread else non-stationary spread