BASICS #101

Signal Generation Structure - These simple yet effective conditions can be incorporated in any algorithm to create some really sound long/short conditions.

A. Long Signals:-

1. Heikin-Ashi Confirmation:
   - Look for "open = low" in Heikin-Ashi candles.
   - Confirm with a Bullish MACD Crossover
   - Ensure the price is near a **61.8% Fibonacci Retracement Level

2. ATR Volatility Check
   - Calculate the ATR. Set a threshold (e.g., ATR > 1.5% of the price).
   - Ensure volatility is sufficient to support the trade.

3. Volume Confirmation:
   - Check for higher-than-average volume (e.g., 1.5x the average volume over the last 20 periods).

B. Short Signals:-

1.Heikin-Ashi Confirmation
   - Look for "open = high" in Heikin-Ashi candles.
   - Confirm with a **Bearish MACD Crossover**.
   - Ensure the price is near a **61.8% Fibonacci Retracement Level**.

2. ATR Volatility Check:
   - Calculate the ATR. Set a threshold (e.g., ATR > 1.5% of the price).
   - Ensure volatility is sufficient to support the trade.

3. Volume Confirmation:
   - Check for increasing volume (e.g., 1.5x the average volume over the last 20 periods).

A. Exit Strategies for Long Positions:

1. Profit Target:
   - Set an initial profit target at a multiple of the ATR (e.g., 1.5 * ATR from entry point).
   - Optionally, target a key Fibonacci extension level (e.g., 161.8% of the retracement).

2. Trailing Stop-Loss:
   - Use a trailing stop based on a percentage of the ATR (e.g., trailing stop at 0.5 * ATR).
   - Adjust the stop-loss as the price moves favorably.

3. Volume Decrease Signal:
   - Exit if volume decreases significantly (e.g., below average volume) after reaching the profit target.

B. Exit Strategies for Short Positions:

1. Profit Target:
   - Set an initial profit target at a multiple of the ATR (e.g., 1.5 * ATR from entry point).
   - Optionally, target a key Fibonacci extension level.

2. Trailing Stop-Loss:
   - Use a trailing stop based on a percentage of the ATR (e.g., trailing stop at 0.5 * ATR).
   - Adjust the stop-loss as the price moves favorably.

3. Volume Increase Signal:
   - Exit if volume increases significantly (e.g., above average volume) after reaching the profit target.

Data Denoising Approaches:-

Drawdowns are the heartbeat of trading – a rhythmic reminder that success is not a straight line but a series of ups and downs. Embrace this rhythm. However while embracing it , you should also brainstorm as to why strats usually end up in drawdowns with skewed time to recovery rate.

From our own experience in algorithmic trading especially when you are processing signals on the most brutal and notoriously volatile asset like BTC - we have found out one major reason as to why this happens.

Signals which are generated in noise or signals from noisy price data are the major pushers for creating drawdowns. Therefore to identify meaningful patterns and signals which truly represent the underlying trend is the need of the hour and to pull this off precisely - **you should aim at first denoising the markets .**

1.HA Candles: Smooth out price fluctuations, revealing underlying trends amidst market noise, aiding in clearer decision-making.

A.https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/heikin-ashi-technique/ B. https://osf.io/preprints/africarxiv/phejd

C.https://www.researchgate.net/publication/361590865_Study_of_Heiken_Ashi_Candlestick_for_Noise_Reduction

D.https://www.researchgate.net/publication/361590865_Study_of_Heiken_Ashi_Candlestick_for_Noise_Reduction

2. Kalman Filters: Dynamic algorithms adept at separating signal from noise, Kalman filters excel in denoising financial data, providing cleaner inputs for analysis and trading strategies.
A. https://arxiv.org/ftp/arxiv/papers/1808/1808.03297.pdf

B.https://digitalcommons.sacredheart.edu/cgi/viewcontent.cgi?article=1025&context=wcob_theses#:~:text=The%20pairs%20trading%20strategy%20using,potential%20for%20higher%20returns%20exists.

C.https://tng-daryl.medium.com/implementing-the-kalman-filter-on-stock-data-1dce3a192a93 D. https://github.com/pykalman/pykalman

D. **"Application of Kalman Filter in Financial Time Series"** by S. S. C. Tso and J. C. K. Yu - This book discusses how Kalman Filters can be effectively applied in financial markets

Go through this Youtube Link for enhanced understanding of Kalmans-
https://www.youtube.com/results?search_query=kalman+filter+finance

---

Youtube Links:-

Attaching here a list of some cool Youtube Links that pertains to Quantitative Finance and would help you to develop a line of thinking for making sound algorithms.

We would arrange them in order of relevance and utility :-

1. www.youtube.com/@intothecryptoverse
A no nonsense channel that integrates mathematics with bitcoin and opens up your psyche to a lot of possibilities in trading. His videos on Heikin Ashi candles are really great and can be used as a great fundamental bedrock in your algorithm rather than relying on traditional candlesticks.

2. https://youtu.be/rlSkRMmycWo?si=8o3vjkyptvVKwxq1
Pick lots of ideas and explain them briefly. There are lots of indicators and their applications nestled in the playlist of this channel.

3. youtube.com/CodeTradingCafe
A pure coder by heart with a knack for markets. Has some really cool stuff in his playlists.

Research Areas

1. Stochastic Control Theory

- **Details**:

Dynamic Programming: Introduce *concepts like Bellman's equation* and how they apply to trading, particularly in the context of optimal trading solutions.

Link to doc for Bellman's Equation-
https://docs.google.com/document/d/1dJC3iZipjjdWxAIR05o6w1jj4yevO_4eqeHNksjRnvQ/edit?tab=t.0

Youtube Link for *Bellman's equation* - https://www.youtube.com/watch?v=9JZID-h6ZJ0

Risk Management: Discuss how traders can formulate strategies that not only maximize returns but also incorporate risk constraints, adjusting their portfolios dynamically.

Books:

- **"Stochastic Calculus for Finance II: Continuous-Time Models"** by Steven E. Shreve - This book provides a thorough understanding of stochastic processes and control theory in the context of finance.

Online Resources:

https://www.youtube.com/results?search_query=stochastic+control+theory

2. Markov Switching Models

- **Details**:
    - **Regime Identification**: Discuss techniques for estimating the parameters of the Markov process, *such as the Expectation-Maximization (EM) algorithm*.

    - **Applications**: Explore how to identify different market regimes (e.g., high volatility vs. low volatility) and how traders can adjust their strategies accordingly, like switching from trend-following in bullish markets to more conservative approaches in bearish conditions.

Resources:-

1. Youtube Link-
https://www.youtube.com/results?search_query=markov+switching+models+financial+time+series

2. "Markov-Switching Models for Forecasting Financial Time Series" by Chuliang Liao, Jian Chen, and Chunhua Zhang - This book discusses applications in financial forecasting, including regime identification.

3. "Time Series Analysis" by James D. Hamilton - A comprehensive text covering Markov Switching models and their application in econometrics.

Learning from Example

Here is the core logical part behind indicators used , their utility and the mathematics behind them one of the algorithms which did quite good in Inter-IIT last year:-

This has been provided as a reference to this year's inter-iit participants to think out of the box , study relevance of indicators used and gather inspiration from the hard work done by this particular team in Inter-IIT last year at Madras.

One should strive to add genius to this minimum expectation that we have set for all the team members. A performance like this should act as your starting point

and all teams should strive to build and create value over & above this performance.

Long Conditions

Long Entry Conditions:

1. **RSI Smoothed Value > 75:**
    - **Reasoning:** A high RSI (over 75) indicates that the asset is in overbought territory, suggesting strong upward momentum. However, it's essential to confirm this momentum with additional indicators.

    - **Mathematical Logic:** The RSI is a momentum oscillator that measures the speed and change of price movements. A smoothed RSI reduces noise, allowing for a clearer trend signal.

2. **Fast Volume SMA > Slow Volume SMA:**
    - **Reasoning:** This condition indicates increasing volume, which typically confirms the strength of a price move. When fast volume averages exceed slow ones, it suggests that recent buying interest is stronger than the historical average.

    - **Mathematical Logic:** Simple Moving Averages (SMA) smooth out price data over time, providing a clearer view of volume trends. The crossover indicates a shift in market sentiment.

3. **Fast EMA > Slow EMA, Fastest EMA > Fast EMA:**
    - **Reasoning:** This series of exponential moving averages (EMAs) signals a strong bullish trend. The hierarchy of EMAs indicates that not only is the price increasing, but the rate of increase is accelerating.

    - **Mathematical Logic:** EMAs give more weight to recent prices, allowing for a quicker response to price changes compared to SMAs.

4. **Aroon Up < Aroon Down:**

- **Reasoning:** This might seem counterintuitive for a long entry condition. However, it suggests that while the current trend is down, a reversal may be imminent, especially if combined with other bullish signals.

- **Mathematical Logic:** The Aroon indicator measures the time since the highest and lowest prices in a given period. Aroon Up < Aroon Down indicates potential exhaustion in the downtrend.

5. **Action:** If all conditions are met, a long position is opened.

Long Exit Conditions:

1. **RSI < 30:**
   - **Reasoning:** An RSI below 30 typically indicates oversold conditions, suggesting that the asset may be undervalued and could rebound. This serves as a trigger to close the long position.

   - **Mathematical Logic:** The RSI's typical range is from 0 to 100, and crossing below 30 may suggest a trend reversal.

2. **Volatility Index > Volatility Index Moving Average:**
   - **Reasoning:** An increasing volatility index indicates rising market uncertainty or potential price swings, which can be a sign to exit positions to avoid losses.

   - **Mathematical Logic:** This condition uses the concept of volatility to gauge risk. If current volatility exceeds historical averages, it may signal an unfavorable environment for holding long positions.

3. **Time-Based Long Exit (576 hours):**
   - **Reasoning:** Time-based exits prevent positions from being held indefinitely and manage risk over time.

   - **Mathematical Logic:** This is a fixed timeframe that helps maintain discipline in trading.

Short Conditions

Short Entry Conditions:

1. **Fast EMA < Slow EMA, Fastest EMA < Fast EMA:**
    - **Reasoning:** A hierarchy of declining EMAs indicates a strong bearish trend, suggesting that further declines are likely.

    - **Mathematical Logic:** The same logic applies as in the long conditions; falling EMAs indicate that recent prices are lower and reflect bearish sentiment.

2. **VWAP Crossover from Below to Above Fastest EMA:**
    - **Reasoning:** This crossover suggests that the average price traded (VWAP) has increased and may indicate a potential reversal or weakening of the bearish trend.

    - **Mathematical Logic:** VWAP represents the average price a security has traded at throughout the day, based on both volume and price. A crossover can signal momentum shifts.

3. **RSI Smoothed Value < 30:**
    - **Reasoning:** A low RSI indicates oversold conditions, which might suggest that a bounce could happen, making it a strategic time to enter a short position before a potential reversal.

    - **Mathematical Logic:** A low RSI can indicate a high probability of a price reversal, creating opportunities to profit from a downward price movement.

4. **Aroon Down < Aroon Up:**
    - **Reasoning:** This condition suggests that the downward momentum is weakening and could indicate an impending reversal, making it a suitable time to enter a short position.

- **Mathematical Logic:** Just as in the long conditions, the relationship between Aroon Up and Down signals potential changes in trend dynamics.

5. **Action:** If all conditions are met, a short position is opened.

Short Exit Conditions:

1. **VWAP > Fastest EMA:**
   - **Reasoning:** If the VWAP exceeds the fastest EMA, it indicates that the average price is now above the short-term trend, suggesting the trend may be reversing or weakening.

   - **Mathematical Logic:** Similar to the entry condition, this crossover reflects potential shifts in market sentiment.

2. **Position Held for More than 8 Hours:**
   - **Reasoning:** This condition allows for a controlled exit after a reasonable duration, reducing the risk of prolonged exposure to unfavorable movements.

   - **Mathematical Logic:** It enforces discipline in trading, ensuring positions are actively managed rather than left open indefinitely.

Step-by-Step Guide to Building an Algorithm with Indicators and Hawkes Process:-

Step 1: Choose Indicators

Select five indicators with unique mathematical logic. Here are some suggestions:

1. Bollinger Bands (BB)

- Logic: BB consists of a middle band (SMA) and two outer bands that are standard deviations away from the SMA. It captures volatility and price levels.

- Usage: Price touching the upper band suggests overbought conditions; touching the lower band suggests oversold.

2. Moving Average Convergence Divergence (MACD)

- Logic: MACD is derived from the difference between a short-term EMA and a long-term EMA. The MACD line and signal line crossover generates buy/sell signals.

- Usage: A bullish signal occurs when the MACD line crosses above the signal line, and vice versa for a bearish signal.

3. Stochastic Oscillator

- Logic: This momentum indicator compares a particular closing price of a security to a range of its prices over a specified period.

- Usage: Values above 80 indicate overbought, while values below 20 indicate oversold conditions.

4. Average True Range (ATR)

- Logic: ATR measures market volatility by decomposing the entire range of an asset for that period. It uses the maximum of three calculations: current high minus current low, and the absolute values of current high minus previous close and current low minus previous close.

- Usage: Higher ATR values indicate higher volatility, which can affect entry and exit points.

5. On-Balance Volume (OBV)

-Logic: OBV uses volume flow to predict changes in stock price. If a security's price rises, all of the volume is considered up-volume; if the price falls, all the volume is down-volume.

-Usage: Increasing OBV confirms an uptrend, while decreasing OBV confirms a downtrend.

Step 2: Define the Hawkes Process

Yes, the Hawkes process is a type of point process that models events occurring in time, where the occurrence of one event increases the probability of subsequent events. It's particularly useful in areas where events are clustered or exhibit self-excitation, meaning that past events influence future ones.

Key Features of the Hawkes Process

1. **Self-Excitation**: The occurrence of an event can increase the likelihood of subsequent events happening in the near future. For example, a sudden price spike might trigger more trades or volatility.
2. **Conditional Intensity**: The process is characterized by a conditional intensity function that defines the rate at which events are expected to occur, given the history of past events.
3. **Exponential Decay**: Typically, the influence of past events decreases over time, often modeled with an exponential decay function.

Go for this Youtube Video to understand the concept better-

https://www.youtube.com/watch?v=wdsiZBIhAFw&t=20s

- Integrate Hawkes Process: Use the Hawkes process to model the influence of past events (like significant price movements) on future volume. The formula typically looks like:

$$
\lambda(t) = \mu + \sum_{t_i < t} \alpha e^{-\beta (t - t_i)}
$$

Where:
- $\lambda(t)$ is the intensity function,
- μ is the baseline intensity,
- α is the influence of past events,
- β controls how quickly the influence decays.

Here is the code block for Hawkes Process:-

```
def hawkes_process(data: pd.Series, kappa: float):

assert(kappa > 0.0)

alpha = np.exp(-kappa)
```

```
arr = data.to_numpy()

output = np.zeros(len(data))

output[:] = np.nan

for i in range(1, len(data)):

if np.isnan(output[i - 1]):

output[i] = arr[i]

else: output[i] = output[i - 1] * alpha + arr[i]

return pd.Series(output, index=data.index) * kappa
```

Explanation:

- This function implements a Hawkes process, which models how past events influence future occurrences in a time series (here, possibly price changes).
- It calculates an intensity function based on the previous values in the series, enabling predictions of future volume changes influenced by historical data.

Step 3: Create Indicator Conditions

Define the conditions for the chosen indicators to generate signals:

1. Bollinger Bands:
   - Buy when the price touches the lower band and the closing price is above the lower band.
   - Sell when the price touches the upper band and the closing price is below the upper band.

2. MACD:
   - Buy when the MACD line crosses above the signal line.
   - Sell when the MACD line crosses below the signal line.

3. Stochastic Oscillator:
   - Buy when the stochastic value is below 20 and crosses above that level.
   - Sell when it’s above 80 and crosses below.

4. ATR:
   - Use ATR to set stop-loss and take-profit levels. For example, a stop-loss could be set at 1.5 times the ATR below the entry price for long positions.

5. OBV:
   - Confirm buy signals with increasing OBV and sell signals with decreasing OBV.

Step 4: Combine Conditions with Hawkes Process

1. Generate Volume Signal:
   - Apply the Hawkes process to the price changes to predict future volume spikes.
   - For example, if a significant volume spike is predicted, check if other indicators align for a potential trade.

2. Signal Generation Logic:
   - If a volume spike occurs (from the Hawkes process) and:
     - The BB conditions align (price at the lower band for buy),
     - The MACD shows a bullish crossover,
     - The Stochastic is crossing above 20,
     - ATR confirms the volatility is appropriate for entry,
     - OBV is increasing,
   - Then trigger a buy signal.

   - Conversely, apply the reverse logic for sell signals.

Here's how you can modify the `process_data` function:

`process_data` Function with Hawkes Process Integration

```
python
import pandas as pd
import numpy as np
import pandas_ta as ta

def calculate_bollinger_bands(data, window=20, num_sd=2):
    sma = data['close'].rolling(window).mean()
    rolling_std = data['close'].rolling(window).std()
    upper_band = sma + (rolling_std * num_sd)
    lower_band = sma - (rolling_std * num_sd)
    return upper_band, lower_band

def calculate_macd(data, short_window=12, long_window=26, signal_window=9):
    exp1 = data['close'].ewm(span=short_window, adjust=False).mean()
    exp2 = data['close'].ewm(span=long_window, adjust=False).mean()
    macd_line = exp1 - exp2
    signal_line = macd_line.ewm(span=signal_window, adjust=False).mean()
    return macd_line, signal_line

def calculate_stochastic(data, k_window=14, d_window=3):
    lowest_low = data['low'].rolling(window=k_window).min()
    highest_high = data['high'].rolling(window=k_window).max()
    k_value = 100 * ((data['close'] - lowest_low) / (highest_high - lowest_low))
    d_value = k_value.rolling(window=d_window).mean()
    return k_value, d_value

def calculate_atr(data, window=14):
    tr1 = data['high'] - data['low']
    tr2 = abs(data['high'] - data['close'].shift(1))
    tr3 = abs(data['low'] - data['close'].shift(1))
    tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
    atr = tr.rolling(window).mean()
    return atr

def calculate_obv(data):
    obv = np.zeros(len(data))
    for i in range(1, len(data)):
        if data['close'].iloc[i] > data['close'].iloc[i - 1]:
            obv[i] = obv[i - 1] + data['volume'].iloc[i]
        elif data['close'].iloc[i] < data['close'].iloc[i - 1]:
            obv[i] = obv[i - 1] - data['volume'].iloc[i]
```

```
        else:
            obv[i] = obv[i - 1]
    return pd.Series(obv, index=data.index)

def hawkes_process(data: pd.Series, kappa: float):
    assert(kappa > 0.0)
    alpha = np.exp(-kappa)
    arr = data.to_numpy()
    output = np.zeros(len(data))
    output[:] = np.nan
    for i in range(1, len(data)):
        if np.isnan(output[i - 1]):
            output[i] = arr[i]
        else:
            output[i] = output[i - 1] * alpha + arr[i]
    return pd.Series(output, index=data.index) * kappa

def process_data(data):
    """Process data to include selected indicators and integrate with the Hawkes process."""

    # Calculate indicators
    data['BB_upper'], data['BB_lower'] = calculate_bollinger_bands(data)
    data['MACD_line'], data['MACD_signal'] = calculate_macd(data)
    data['Stoch_K'], data['Stoch_D'] = calculate_stochastic(data)
    data['ATR'] = calculate_atr(data)
    data['OBV'] = calculate_obv(data)

    # Normalize and use a relevant column for the Hawkes process
    # For example, we can use the ATR to assess volatility
    data['norm_volatility'] = (data['ATR'] - data['ATR'].mean()) / data['ATR'].std()
    data['v_hawk'] = hawkes_process(data['norm_volatility'], kappa=3)

    # Generate signals based on indicators and Hawkes process
    data['signal'] = 0  # Initialize signal column

    # Example signal logic (customize as needed)
    for i in range(len(data)):
        # Example buy condition:
        if (data['close'].iloc[i] < data['BB_lower'].iloc[i] and
            data['MACD_line'].iloc[i] > data['MACD_signal'].iloc[i] and
            data['Stoch_K'].iloc[i] < 20 and
            data['v_hawk'].iloc[i] > 0):  # Adjust Hawkes process condition
            data['signal'].iloc[i] = 1  # Buy signal
```

```
        # Example sell condition:
        elif (data['close'].iloc[i] > data['BB_upper'].iloc[i] and
              data['MACD_line'].iloc[i] < data['MACD_signal'].iloc[i] and
              data['Stoch_K'].iloc[i] > 80 and
              data['v_hawk'].iloc[i] < 0):  # Adjust Hawkes process condition
            data['signal'].iloc[i] = -1  # Sell signal

    return data
```

Explanation of Changes

1. Indicator Calculations:
- The functions for calculating Bollinger Bands, MACD, Stochastic Oscillator, ATR, and OBV remain unchanged.

2. Hawkes Process Integration:
- The `hawkes_process` function is applied to a normalized version of the ATR to create a Hawkes process output, `v_hawk`, that reflects volatility.

3. Signal Generation Logic:
- A new `signal` column is initialized in the DataFrame.

- Example conditions for buy and sell signals have been added. These conditions take into account the indicators and the output from the Hawkes process:

  - Buy Condition: The price is below the lower Bollinger Band, MACD line is above the signal line, Stochastic %K is below 20, and the Hawkes process output is positive.

  - Sell Condition: The price is above the upper Bollinger Band, MACD line is below the signal line, Stochastic %K is above 80, and the Hawkes process output is negative.

Adx and its relevance:-

```
def calculate_adxr(adx, lookback):
    adxr = (adx + adx.shift(lookback)) / 2
```

```
    return adxr

def calculate_adx(data, lookback):
    high = data['high']
    low = data['low']
    close = data['close']

    plus_dm = high.diff()
    minus_dm = low.diff()

    plus_dm[plus_dm < 0] = 0
    minus_dm[minus_dm > 0] = 0

    tr1 = high - low
    tr2 = abs(high - close.shift(1))
    tr3 = abs(low - close.shift(1))
    tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)

    atr = tr.rolling(lookback).mean()

    plus_di = 100 * (plus_dm.ewm(alpha=1/20).mean() / atr)
    minus_di = abs(100 * (minus_dm.ewm(alpha=1/20).mean() / atr))

    dx = (abs(plus_di - minus_di) / abs(plus_di + minus_di)) * 100
    adx = dx.rolling(window=lookback).mean()

    return plus_di, minus_di, adx
```

This code block implements the Average Directional Index (ADX) and its derivative, the Average Directional Index Rating (ADXR). Both are widely used in trading algorithms to assess market trends and momentum.

Here's a detailed explanation of each function and its relevance:

1. `calculate_adx(data, lookback)`

Purpose:
This function calculates the ADX and its components, which help traders identify whether a market is trending or ranging.

Parameters:

- `data`: A DataFrame containing historical price data, specifically high, low, and close prices.
- `lookback`: The period over which the ADX and its components are calculated.

Functionality:
- Directional Movement (DM) Calculation:
  - `plus_dm` and `minus_dm` are calculated as the difference between the current and previous highs/lows. Positive values indicate upward movement, while negative values are set to zero.
  - Negative values of `minus_dm` are set to zero, as they indicate no downward movement.

- True Range (TR):
  - The true range is calculated using three different methods:
    1. Current high minus current low.
    2. Absolute difference between the current high and the previous close.
    3. Absolute difference between the current low and the previous close.
  - The maximum of these three values for each time period gives the true range.

- Average True Range (ATR):
  - The ATR is the rolling mean of the true range over the specified lookback period, indicating market volatility.

- Directional Index (DI):
  - `plus_di` and `minus_di` are calculated to represent the strength of upward and downward movements relative to the ATR.

- Directional Movement Index (DX):
  - The DX measures the difference between `plus_di` and `minus_di`, normalized by the sum of these two. It quantifies the trend strength.

- Average Directional Index (ADX):
  - Finally, the ADX is calculated as the rolling mean of the DX, which indicates the strength of the trend without considering its direction.

Relevance in Trading:
- Trend Strength Identification: The ADX provides a numerical value indicating whether a trend is strong (typically above 25) or weak (below 20). This helps traders decide whether to enter or exit positions based on the market's trend state.

- Signal Generation: Traders may use crossovers of the `plus_di` and `minus_di` to generate buy/sell signals. For instance, a buy signal may be generated when `plus_di` crosses above `minus_di`, indicating a bullish trend.

2. `calculate_adxr (adx, lookback)`

Purpose:

This function calculates the ADXR, which is a smoothed version of the ADX. It helps in reducing noise in the ADX values, providing a clearer picture of trend strength.

Parameters:
- `adx`: The ADX values calculated from the previous function.
- `lookback`: The period over which the ADXR is calculated.

Functionality:
- The ADXR is computed as the average of the current ADX and the ADX from `lookback` periods ago, providing a smoother indicator.

Relevance in Trading:
- Noise Reduction: The ADXR helps mitigate the volatility of the ADX, making it easier for traders to interpret trends.

- Confirmation of Trends: A rising ADXR indicates strengthening trends, while a falling ADXR can suggest weakening trends, aiding in more informed decision-making.

Overall Significance in Trading Algorithms:
- Signal Clarity: By using ADX and ADXR, traders can filter out false signals, focusing on genuine trends.
- Risk Management: Understanding the strength of trends helps in setting stop-loss orders and determining position sizing.

- Strategic Entry/Exit: These indicators assist in identifying optimal entry and exit points based on market momentum, enhancing the performance of trading strategies.

In summary, the ADX and ADXR are powerful tools in technical analysis, providing traders with valuable insights into market conditions and helping to produce high-quality trading signals.

Also when you are catching trend - you also want to ride only those which has strength
So why not bring ADX into the picture:-

```
def include_adx(df , di_period , adx_period):
df['adx'] = ta.ADX(df['ha_high'] , df['ha_low'] , df['ha_close'] , timeperiod=adx_period)
df['+di'] = ta.PLUS_DI(df['ha_high'] , df['ha_low'] , df['ha_close'] , timeperiod= di_period)
df['-di'] = ta.MINUS_DI(df['ha_high'] , df['ha_low'] , df['ha_close'] , timeperiod= di_period)

return df
```

This is the denoised version of ADX. Works really great for some of our in-house algos.

SLOPE OF INDICATORS AND THEIR RELEVANCE:-

Understanding Lagging Indicators

Lagging indicators, such as moving averages (MA), Relative Strength Index (RSI), and MACD, are based on historical price data. While they can confirm trends and provide signals, their reactive nature means they may generate signals after significant price movements have already occurred, potentially leading to missed opportunities or increased risk.

Incorporating Slope of Indicators

The slope of an indicator quantifies how quickly it is changing over time, which can help traders identify the strength and direction of a trend. Here's how you can leverage this concept:

1. **Calculate the Slope:**
   - The slope can be calculated by taking the difference between the current value of the indicator and its value at a previous point in time (e.g., one or several periods back).
   - For instance, for a moving average:
     $$\text{Slope} = \text{MA}_t - \text{MA}_{t-n}$$
     where n is the number of periods back you want to look.
2. **Assess Rate of Change:**
   - A positive slope indicates that the indicator is increasing, which may signal a strengthening trend, while a negative slope indicates a decreasing indicator, suggesting a weakening trend.
   - You can also calculate the percentage change to quantify the rate of change:
     $$\text{Rate of Change} = \frac{\text{Slope}}{\text{MA}_{t-n}} \times 100$$

3. **Create Long/Short Signals:**
    - **Long Signal:** When the slope of the indicator becomes positive (indicating increasing momentum) and the indicator itself is above a certain threshold (e.g., above a moving average), you could consider entering a long position.
    - **Short Signal:** Conversely, when the slope is negative (indicating decreasing momentum) and the indicator falls below a certain threshold, you could consider entering a short position.
4. **Combine with Other Indicators:**
    - Use the slope in conjunction with other indicators for confirmation. For example, if the slope of the RSI is increasing while the price is above a certain moving average, it could strengthen the case for a long position.
5. **Backtest the Strategy:**
    - Before implementation, backtest this approach using historical data to evaluate its effectiveness. Analyze the results based on your desired performance metrics.

Benefits of Using Slopes

- **Timely Signals:** By assessing the slope, you can generate more timely trading signals, potentially allowing you to capitalize on price movements more effectively.
- **Trend Strength Assessment:** Slope provides insight into the strength of a trend, helping you avoid trades in weak or sideways markets.
- **Risk Management:** Incorporating slope can enhance your risk management strategy by helping you identify when to exit positions more effectively.

BENFORD'S LAW ANALYSIS IN ALGORITHMIC TRADING:-

Vishal_03 01/18/2024 2:15 PM
Gone through the Benford's law which was quite surprising to know
Sure it would be great to have discussions on these ideas, which could be potentially incorporated in building strategy

HarshB 01/18/2024 2:39 PM
When observing percentage changes in the closing prices of BTCUSDT every 15mins/30mins or 1 hour, you can apply Benford's Law to the leading digits of these percentage changes to identify potential deviations from the expected distribution.

Let's say you've gathered the percentage changes for several 15-minute candlesticks in sequential order and extracted their leading digits. For instance:

1.23%
0.89%
2.56%
0.47%
3.78%
0.12%
4.91%
1.68%

Then, you'd extract the leading digit from each percentage change:

1
8
2
4
3
1
4
1

Next, compare the frequency of occurrence of each leading digit (1-9) with the expected Benford's Law distribution. In Benford's Law, the expected percentage of occurrences for each leading digit is approximately:

1: 30.1%
2: 17.6%
3: 12.5%
4: 9.7%
5: 7.9%
6: 6.7%
7: 5.8%
8: 5.1%
9: 4.6%

By comparing the observed frequency of leading digits in your data with these expected percentages, you can detect any significant deviations. If, for instance, you find that the occurrences of a specific leading digit consistently deviate significantly from Benford's Law (e.g., a notably higher frequency for a particular digit), it might indicate anomalies or irregularities in the data.

When applying Benford's Law to leading digits, it's a common practice to exclude leading zeros from the analysis. As per law , numbers start naturally from 1 and thus we have to avoid zero.

Therefore when calculating the distribution of leading digits, exclude numbers that start with zero. For example, if you have a percentage change of 0.75%, consider only the digit 7 for Benford's Law analysis or if the actual percentage change is 0.81%, consider the next digit after zero (in this case, 8) for Benford's Law analysis

is this aspect clear to you ? If yes , we can proceed further into this
👍 6

Applying Benford’s Through Volume:-

To incorporate Benford's Law in the analysis of BTCUSDT trading volume for filtering out fake volume and creating effective long/short signals, you can follow these steps:

Step 1: Collect Volume Data

Gather historical trading volume data for BTCUSDT at your desired time intervals (15 minutes, 30 minutes, 1 hour). Ensure that the data is clean and free from outliers or errors.

Step 2: Calculate Percentage Changes

Calculate the percentage changes in volume over your chosen intervals. For example, if the volume at time t is V_t and at time $t+1$ is V_{t+1}, the percentage change is:

$$\text{Percentage Change} = \left(\frac{V_{t+1} - V_t}{V_t} \right) \times 100$$

Step 3: Extract Leading Digits

For each percentage change calculated, extract the leading digit. Exclude leading zeros as per Benford's Law. For instance, if you get a percentage change of 0.45%, consider only the leading digit 4.

Step 4: Analyze Frequency Distribution

Compile the leading digits from all your calculated percentage changes and analyze their frequency of occurrence. Create a frequency table for the leading digits (1-9).

Step 5: Compare with Benford's Distribution

Compare the observed frequency of leading digits against the expected frequencies from Benford's Law. This can be done visually (e.g., through a bar chart) or statistically (using a Chi-squared test to check for significant deviations).

Compare the observed frequency of leading digits against the expected frequencies from Benford's Law. This can be done visually (e.g., through a bar chart) or statistically (using a Chi-squared test to check for significant deviations).

Step 6: Identify Anomalies

Look for significant deviations from the expected distribution. A consistent over-representation of certain leading digits (especially lower digits) might indicate manipulative trading practices, such as wash trading or fake volume.

Step 7: Create Volume-Based Signals

Use the insights gained from your Benford analysis to inform your trading strategy:

- **Volume Increase Signals**: If you observe a normal distribution with occasional spikes in legitimate volume (following Benford's expectations), this could signal a potential uptrend. Consider entering long positions if volume spikes correlate with upward price movement.
- **Volume Decrease or Anomalies**: If certain leading digits suggest fake volume (e.g., many leading ones or twos), it might indicate that recent volume spikes are not legitimate. Use this information to avoid entering long positions or consider shorting the asset, especially if the price trend contradicts the volume signals.

Step 8: Validate with Additional Indicators

Combine your findings with other technical indicators or analysis methods (like RSI, MACD, or support/resistance levels) to validate your trading signals further. This can enhance the reliability of your long/short signals.

Step 9: Continuous Monitoring and Adjustment

Regularly monitor the leading digit analysis and volume trends. Adjust your strategy based on new data and observed patterns. Market dynamics can change, and your approach should be flexible enough to adapt.

UTILITY OF MACD AND ITS HISTOGRAM:-

1. Adaptive Histogram Smoothing

Methodology:-

1. **Weighted Moving Average:**
   Instead of treating all histogram values equally, a weighted moving average assigns different weights to the values based on their distance from the mean. Closer values receive higher weights, while those further away receive lower weights.

- Gaussian Smoothing:

A Gaussian filter can be applied to smooth the histogram data. This filter assigns weights according to the Gaussian distribution, which reduces the influence of outliers and captures trends more effectively.

Implementation:
- Calculate the mean of the histogram.
- Determine weights based on the distance from the mean (e.g., using a Gaussian function).
- Apply the weighted average or Gaussian filter to the histogram values.

Signal Generation:
- When the smoothed histogram crosses a certain threshold (e.g., zero or a moving average), it can generate long or short signals based on momentum shifts.

2. Z-Score Analysis

Methodology:

Z-Score Calculation:
The Z-score indicates how many standard deviations a data point is from the mean. It's calculated as:

$$Z = \frac{(X - \mu)}{\sigma}$$

where X is the histogram value, μ is the mean of the histogram, and σ is the standard deviation.

Implementation:
- Compute the mean and standard deviation of the histogram.
- Calculate the Z-score for each histogram value.
- Define thresholds for entry signals (e.g., +2 for long, -2 for short).

Signal Generation:
- A Z-score above +2 indicates a strong bullish signal, while below -2 indicates a strong bearish signal.

3. Fourier Transform

Methodology:

Frequency Analysis:
The Fourier Transform decomposes a signal into its constituent frequencies. It allows you to identify cycles and trends in the histogram data.

Discrete Fourier Transform (DFT): Use the DFT or Fast Fourier Transform (FFT) algorithms to convert the time-domain signal (histogram) into the frequency domain.

Implementation:
- Apply the FFT to the histogram data.
- Analyze the resulting frequency spectrum to identify dominant frequencies.

Signal Generation:
- If a dominant frequency shifts significantly, it may indicate a change in market trends. For example, if a previously dominant bullish frequency weakens, it might suggest a reversal.

4. Entropy-Based Measures

Methodology:

Entropy Calculation:
Entropy quantifies the amount of uncertainty or disorder in a dataset. For a histogram, it can be calculated using Shannon entropy:

$$Z = \frac{(X - \mu)}{\sigma}$$

where X is the histogram value, μ is the mean of the histogram, and σ is the standard deviation.

Implementation:
- Normalize the histogram values to compute probabilities.
- Calculate the entropy based on the normalized values.

Signal Generation:
A high entropy value indicates a more random or uncertain market, while a low entropy value indicates a more stable or trending market. Changes in entropy can signal potential reversals or continuations.

STOP LOSS PLACEMENTS USING Average True Range (ATR)

ATR Percentile Ranking:

- Calculate the percentile rank of the current ATR compared to historical values. Use this percentile to adjust the stop loss multiplier. For instance, if the ATR is in the 75th percentile, you might use a higher multiplier to account for increased volatility.

Step-by-Step Process

1. **Gather Historical ATR Data**:
   - Collect historical ATR values for your asset over a specified period (e.g., 100 days). You can calculate ATR using high, low, and close prices.
2. **Calculate the Percentile Rank**:
   - For the current ATR value, calculate its percentile rank among the historical ATR values. This involves:
     - **Sorting Historical ATR Values**: Arrange the historical ATR values in ascending order.
     - **Finding the Current ATR's Position**: Determine how many historical ATR values are less than the current ATR.
     - **Calculating the Percentile**: Use the formula:

$$\text{Percentile Rank} = \left(\frac{\text{Number of Values Below Current ATR}}{\text{Total Number of Historical Values}} \right) \times 100$$

3. **Set Multiplier Based on Percentile**:
   - Define thresholds for the percentile ranks to determine the stop loss multiplier. For example:
     - **0-25th Percentile**: Use a lower multiplier (e.g., 1.0)
     - **26-50th Percentile**: Moderate multiplier (e.g., 1.5)
     - **51-75th Percentile**: Higher multiplier (e.g., 2.0)
     - **76-100th Percentile**: Highest multiplier (e.g., 2.5)
   - Adjust these ranges based on your strategy and risk tolerance.

ATR Divergence:

- Analyze the relationship between price movements and ATR trends. For example, if prices are making new highs but ATR is declining, consider tightening stop losses, as this could indicate weakening momentum.

Step-by-Step Process

1. **Collect Price and ATR Data**:
   - Gather historical price data (highs, lows, closes) and calculate the ATR over your desired time frame (e.g., 14 days).
2. **Identify Price Trends**:
   - Analyze recent price movements to identify trends. This includes looking for:
     - New highs: Determine if the price is making new highs compared to previous highs.
     - New lows: Check for new lows if you're analyzing a downtrend.
3. **Examine ATR Trends**:
   - Track the ATR over the same period to assess its trend:
     - Calculate the moving average of the ATR (e.g., 14-day MA) to smooth out volatility.
     - Determine if the ATR is rising, falling, or flat over the same timeframe.
4. **Look for Divergence**:
   - Identify divergences between price and ATR:
     - **Bullish Divergence**: Price makes a new high while ATR is declining. This suggests weakening momentum despite higher prices.
     - **Bearish Divergence**: Price makes a new low while ATR is rising. This indicates increasing volatility during a downtrend, suggesting potential reversal or instability.

5. **Decide on Stop Loss Adjustments**:
   - Based on the identified divergences:
     - **For Bullish Divergence**:
       - Tighten your stop loss to protect against potential reversals. Set the stop loss closer to the entry point or the last swing low.
     - **For Bearish Divergence**:
       - You might consider widening your stop loss to give the position more room to breathe or tightening it further if you anticipate increased volatility.

ATR Ratio Analysis:

- Compare the current ATR to the average ATR over a longer time frame. A rising ATR compared to the historical average could signal increasing volatility, prompting a wider stop loss. Conversely, a declining ATR might suggest tightening.

Step-by-Step Process

1. **Gather Data**:
   - Collect historical price data for your asset to calculate ATR. You'll need high, low, and close prices over a specified period (e.g., 14 days for current ATR, 100 days for historical average).
2. **Calculate Current ATR**:
   - Calculate the ATR for your chosen short-term period (e.g., 14 days) using the standard ATR formula, which considers the true range (high - low, high - previous close, low - previous close).
3. **Calculate Historical Average ATR**:
   - Calculate the average ATR over a longer time frame (e.g., 100 days):

$$\text{Average ATR} = \frac{\sum \text{ATR over last 100 days}}{100}$$

4. **Compute the ATR Ratio**:
   - Calculate the ATR Ratio by comparing the current ATR to the historical average ATR:

$$\text{ATR Ratio} = \frac{\text{Current ATR}}{\text{Average ATR}}$$

5. **Set Thresholds for Stop Loss Adjustment**:
   - Define thresholds for the ATR Ratio to determine how to adjust your stop loss. For example:
     - **ATR Ratio < 0.75**: Suggests low volatility. Consider tightening the stop loss.
     - **ATR Ratio between 0.75 and 1.25**: Suggests normal volatility. Keep the stop loss as is.
     - **ATR Ratio > 1.25**: Indicates high volatility. Consider widening the stop loss.
6. **Calculate Stop Loss**:
   - Depending on the ATR Ratio, calculate the stop loss:
     - If tightening:

$$\text{Stop Loss} = \text{Entry Price} - (\text{Current ATR} \times 1.0) \quad \text{(for example)}$$

     - If maintaining:

$$\text{Stop Loss} = \text{Entry Price} - (\text{Current ATR} \times 1.5)$$

     - If widening:

$$\text{Stop Loss} = \text{Entry Price} - (\text{Current ATR} \times 2.0)$$

ATR Multiplier:

- The stop loss can be set at a multiple of the ATR from the entry point. For example, using 1.5x or 2x ATR allows for adjustments based on current volatility.

Determine Entry Point:

- Decide on the entry price for your trade (e.g., the price at which you buy or sell the asset).

Select an ATR Multiplier:

- Choose an appropriate ATR multiplier based on your risk tolerance and market conditions. Common multipliers are 1.5x, 2x, or even higher for volatile assets.

Calculate Stop Loss Level:

- Depending on whether you are taking a long or short position, calculate the stop loss:
- **For Long Positions**:

$$\text{Stop Loss} = \text{Entry Price} - (\text{ATR} \times \text{Multiplier})$$

- **For Short Positions**:

$$\text{Stop Loss} = \text{Entry Price} + (\text{ATR} \times \text{Multiplier})$$

Implement the Stop Loss:

- Set your stop loss order at the calculated level in your trading platform. This ensures that your position is protected according to the volatility of the asset.

Dynamic ATR Scaling:

- Instead of a fixed multiplier, dynamically adjust the multiplier based on market conditions (e.g., using a higher multiplier during high volatility and a lower one during low volatility).

Determine Volatility Conditions:

- Establish criteria to categorize market volatility. This can be based on historical ATR values or a comparison to a longer-term average ATR. For example:
  - **Low Volatility**: ATR < 0.5 (or below a certain percentile).
  - **Normal Volatility**: ATR between 0.5 and 1.0.
  - **High Volatility**: ATR > 1.0 (or above a certain percentile).

Set Dynamic Multipliers:

- Assign multipliers based on the determined volatility conditions. For example:
  - **Low Volatility**: Multiplier = 1.0
  - **Normal Volatility**: Multiplier = 1.5
  - **High Volatility**: Multiplier = 2.0 or higher

Determine Entry Price:

- Identify the entry price for your trade (the price at which you buy or sell the asset).

Calculate Stop Loss Level:

- Using the current ATR and the dynamically assigned multiplier, calculate the stop loss:
- **For Long Positions**:

$$\text{Stop Loss} = \text{Entry Price} - (\text{ATR} \times \text{Dynamic Multiplier})$$

- **For Short Positions**:

$$\text{Stop Loss} = \text{Entry Price} + (\text{ATR} \times \text{Dynamic Multiplier})$$

ATR Bands:

- Create upper and lower bands around a moving average using ATR. For instance, a stop loss could be placed just outside the lower ATR band, providing a buffer against noise while still reacting to price movements.

Calculate Moving Average:

- Choose a moving average type (e.g., simple moving average (SMA), exponential moving average (EMA)) and calculate it over a specific period (e.g., 20 days).

$$\text{Moving Average} = \frac{\sum \text{Closing Prices}}{\text{Number of periods}} \quad (\text{e.g., 20 days})$$

Determine ATR Bands:

- Create upper and lower ATR bands around the moving average:
- **Upper Band**:

$$\text{Upper Band} = \text{Moving Average} + (ATR \times \text{Multiplier})$$

- **Lower Band**:

$$\text{Lower Band} = \text{Moving Average} - (ATR \times \text{Multiplier})$$

- Choose a suitable multiplier based on your trading strategy (e.g., 1.5, 2.0).

Set Stop Loss Level:

- For long positions, consider placing the stop loss just outside the lower ATR band:

$$\text{Stop Loss} = \text{Lower Band}$$

- For short positions, you can place the stop loss just outside the upper ATR band:

$$\text{Stop Loss} = \text{Upper Band}$$

ATR-Based Trailing Stops:

- Implement a trailing stop loss that adjusts based on the ATR as the trade progresses.

LONG CASE:-

You can always trail the stop losses as per given calculation below on closing of timestamps. Always update trailing stop loss on closing of candle.

Similar reasoning is applied in case of **short case:-**

. Mathematical Framework

- **ATR Calculation**:

$$\text{ATR} = \frac{1}{n}\sum_{i=1}^{n}\text{TR}_i$$

Where TR_i is the true range for each period, calculated as:

$$\text{TR}_i = \max(\text{High}_i - \text{Low}_i, |\text{High}_i - \text{Close}_{i-1}|, |\text{Low}_i - \text{Close}_{i-1}|)$$

- **Initial Trailing Stop**:

$$\text{Trailing Stop} = \text{Entry Price} - (\text{ATR} \times \text{Multiplier})$$

- **Dynamic Adjustment**: If the highest price since entry is updated, the trailing stop is recalculated:

$$\text{Trailing Stop} = \text{Highest Price Since Entry} - (\text{ATR} \times \text{Multiplier})$$

Correlation Between BTC and ETH for Algorithmic Trading

1. Pearson Correlation Coefficient

Measures the strength and direction of the linear relationship between two variables. In the context of cryptocurrency trading, it helps traders understand how closely the price movements of BTC and ETH are related.

Interpretation can be of three types:-

1. Perfect positive correlation (as BTC moves up, ETH moves up).
2. Perfect negative correlation (as BTC moves up, ETH moves down).
3. No correlation.

Trading Example: Imagine you find a Pearson correlation of 0.85 between BTC and ETH over the last 90 days. This strong positive correlation suggests that when BTC experiences a price increase, ETH is likely to follow. If BTC breaks above a significant resistance level, you might consider entering a long position in ETH as well, anticipating a similar upward movement.

Trading Logic Based on Correlation and Indicators

Compute the rolling Pearson correlation coefficient over a specified window (e.g., 30 days) to capture dynamic relationships. We have taken 30 days as an example , this number could be different for different teams.

- **Entry Signal**:
    - If the rolling Pearson correlation rt > 0.7 and BTC/USDT is above its 50-day moving average(bullish) while ETH/USDT RSI is below 30(oversold), consider entering a long position in ETH/USDT.

    - Conversely, if rt < −0.7 and BTC/USDT is below its 50-day moving average (bearish) while ETH/USDT RSI is above 70(overbought), consider entering a short position in ETH/USDT.

We have shown the widely used 50 moving average and RSI in example above for your clear understanding .

All Teams can go ahead to opt their own unique indicators and logics for creating long/short conditions and then finely integrate it with correlation mechanisms to produce best alpha producing signals on the dataset.

Regularly reassess the correlation coefficient and adjust trading strategies accordingly. If the correlation changes significantly, be prepared to recalibrate entry and exit criteria.

2. Cross-Correlation Analysis

Cross-correlation analyzes the relationship between two time series at different lags. This is particularly useful for understanding if movements in one cryptocurrency predict movements in another over time.

- **Calculation:** Cross-correlation is calculated by shifting one time series (e.g., ETH) over another (e.g., BTC) and measuring the correlation at each lag.

- **Interpretation:** Positive lags indicate that ETH movements precede BTC, while negative lags indicate the opposite.

Trading Example: Suppose you perform cross-correlation analysis and find that ETH price movements lead BTC price movements by 3 days. If ETH is showing bullish signs today, you might prepare to enter a long position in BTC in three days, based on the expectation that BTC will follow suit.

Trading Logic Based on Cross-Correlation and Indicators

1. Compute the cross-correlation function (CCF) between BTC and ETH prices at various lags.

2. Analyze the cross-correlation results to identify significant lags. Positive lags suggest that ETH movements precede BTC movements, while negative lags suggest the opposite.
For example, if the highest positive correlation occurs at lag +3 days, it indicates that ETH price movements lead BTC by three days and vice versa.

This also indicates that lagged values of ETH can also be an additional feature in your analysis. For instance, if ETH's price shows a significant upward movement three days before BTC's price moves, use ETH’s lagged price as a predictor for BTC.

3. If cross-correlation analysis indicates that ETH leads BTC by +3 days and ETH shows bullish indicators (like a bullish MACD crossover or crossing above its 50-day moving average), prepare to enter a long position in BTC three days later.

Similarly if ETH shows bearish signs and leads BTC with a significant negative correlation, consider entering a short position in BTC.

3. Advanced Trading Strategy Using Dynamic Time Warping (DTW)

It measures the similarity between two time series that may vary in speed or timing. The DTW distance is calculated by constructing a cost matrix where each cell represents the distance between elements of the two series, followed by a recursive calculation to find the optimal path.

Interpretation: A lower DTW distance indicates that the two time series are more similar, while a higher distance suggests less similarity.

Trading Example: If you use DTW to analyze BTC and ETH prices during a volatile market period and find a low DTW distance, you might conclude that their price movements are closely related. This could lead you to create a trading strategy where you take a position in BTC whenever ETH shows specific volatility patterns.

Trading Logic Based on DTW

1. Compute the DTW distance between BTC and ETH price movements over a specified time window (e.g., hourly data over the last 30 or may be 60 timestamps).

2. A low DTW distance indicates a high degree of similarity in price movements, suggesting that the series move together over time.
Use threshold values (e.g., DTW distance < threshold) to determine when to trigger trading signals.

3. Use the Average True Range (ATR) to gauge volatility in both BTC and ETH. If both assets exhibit rising volatility concurrently, it may signal an impending price movement.

Track MACD crossovers in both BTC and ETH to confirm trends. A bullish crossover in ETH could signal a corresponding movement in BTC if DTW distance is low.

4. If the DTW distance between BTC and ETH is below a defined threshold (indicating strong correlation) and ETH shows bullish indicators (like crossing above the 50-day moving average), enter a long position in BTC.

Conversely, if DTW indicates strong correlation but ETH shows bearish signs (e.g., bearish MACD crossover), consider entering a short position in BTC.

4. Advanced Trading Strategy Using Mutual Information

It quantifies the amount of information obtained about one variable through another, capturing both linear and non-linear relationships. It provides insight into how much knowing the state of one variable reduces uncertainty about another.

The formula for mutual information between two random variables X (BTC) and Y (ETH) is:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \left(\frac{P(x,y)}{P(x)P(y)} \right)$$

Here, $P(x,y)$ is the joint probability distribution of X and Y, while $P(x)$ and $P(y)$ are the marginal probability distributions.

Interpreting Mutual Information Results

1. A high mutual information value indicates a strong dependency between BTC and ETH. This suggests that movements in one asset can predict movements in the other.

2. Set a threshold (e.g., I(X;Y) > threshold to determine when to trigger trading signals based on the calculated mutual information.

- **Cointegration Test**: Conduct a cointegration test (like the Engle-Granger test) to determine if BTC and ETH prices share a long-term equilibrium relationship, enhancing the understanding of their dependency.

- **Technical Indicators**:
    - **Moving Averages**: Use short-term and long-term moving averages (e.g., 50-day and 200-day) to confirm trend direction.
    - **Volume Analysis**: Analyze trading volume spikes in both BTC and ETH. Increasing volume during price movements can signal stronger trends.

Trading Logic Based on Mutual Information

- **Entry Signal**:
    - If mutual information I(X;Y) indicates strong dependency and BTC shows bullish signs (e.g., crossing above its 50-day moving average), consider entering long positions in both BTC and ETH.

    - If mutual information is high but BTC shows bearish indicators (e.g., falling below its moving average), consider entering short positions in both assets.

- **Divergence Analysis**: Look for divergence between the two assets. If BTC moves higher while ETH lags, assess whether this divergence is sustainable or likely to converge.

YouTube Link to widen your thinking ability on correlation:-

https://www.youtube.com/watch?v=n2mY86S01fg

CUSUM (Cumulative Sum) Trading

What is CUSUM?

CUSUM (Cumulative Sum) is a way of tracking changes in a number (like price) over time. It's like keeping track of how much the number has moved from a starting point.

How Does It Work?

1. **Step 1: Find the Difference**
   For each new price you see, **subtract** the starting value (the average or μ_0) from it.

$$d_i = x_i - \mu_0$$

   Where:

   - x_i is the price at time **i**.
   - μ_0 is the starting average price (like the middle point).
   - d_i is how much the price has moved from the starting point.

2. **Step 2: Add Up the Differences**
   After finding the difference for each price, **add them up**.

$$S_m = \sum_{i=1}^{m} (x_i - \mu_0)$$

   So, at any time **m**, the CUSUM is just the sum of all the changes (positive or negative) from the starting point.

Why is it Useful?

- If the sum keeps getting **bigger** or **smaller** over time, it shows that the prices are **shifting** in one direction.
- This helps traders see when there's a **big change** in the market.

Example:
Let's say the price starts at $100, and every hour it goes up by $2:

- **Hour 1**: 102 → $d_1 = 102 - 100 = +2$
- **Hour 2**: 104 → $d_2 = 104 - 100 = +4$
- **Hour 3**: 106 → $d_3 = 106 - 100 = +6$

CUSUM at Hour 3:
$S_3 = 2 + 4 + 6 = 12$

Since the sum keeps going up, it shows prices are **increasing**. This can help us **spot a trend** early.

It tracks cumulative deviations from a reference value, where consistent positive or negative deviations indicate potential market shifts. The deviation between each observed value x_i (e.g., price) and the reference value μ_0 (mean of the process) is calculated as:

$$d_i = x_i - \mu_0$$

The cumulative sum at time i is computed by summing these deviations up to that point. This method helps to identify significant shifts in market conditions when the deviations consistently move in one direction.

To improve the accuracy and robustness of **CUSUM for trend detection, we can refine the reference value (or the baseline level) by applying various filtering techniques:-**

These techniques help smooth out noise, adapt to market conditions, and make the algorithm more responsive to real trends.

1.

Kalman Filter

What it is: The Kalman filter is an **optimal estimation technique** that helps estimate the true state of a system in the presence of noise. It's widely used in tracking and prediction.

How it helps: By treating price data as noisy observations, the Kalman filter generates an **estimated reference value** (or baseline) that accounts for **both the trend and noise**, making it more adaptive and accurate.

Mathematics:
The Kalman filter works recursively, updating the estimate based on the observed data and the **uncertainty of the data**.

Why use it: **Adaptively adjusts** to both **short-term fluctuations** and **long-term trends**, making it perfect for **dynamic markets** where trends change often.

2. CUSUM with Nonlinear Reference Values (Polyfit or Spline)

- **Concept**: Instead of using linear reference values (like moving averages), use **polynomial fits** or **splines** to model the reference value. These methods allow you to fit a **nonlinear trend** to price data, which can capture more complex behaviors.
- **How it works**: Fit a **polynomial** or a **spline curve** (e.g., cubic spline) to historical price data and use this as your reference value for CUSUM. These approaches can account for **curved trends** in price data, which traditional moving averages might miss.
- Most traditional methods assume a **linear reference value**, but markets often exhibit nonlinear behavior (e.g., parabolic moves, exponential growth). This approach adds **flexibility** to adapt to **complex patterns**.

Example: Use **least-squares polynomial fitting** to determine the reference value at each time step, allowing the algorithm to **fit the market's true nonlinear trend**.

Python Code Implementation:-

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

class CUSUMNonlinear:
    def __init__(self, price_data, threshold=5, window_size=50, poly_degree=3):
        """
        Initializes the CUSUM Nonlinear model.

        :param price_data: List or numpy array of historical price data.
        :param threshold: CUSUM threshold for detecting significant deviations.
        :param window_size: The size of the window for polynomial fitting.
        :param poly_degree: Degree of the polynomial for fitting (default is cubic).
        """
        self.price_data = np.array(price_data)
        self.threshold = threshold
        self.window_size = window_size
        self.poly_degree = poly_degree

        # Prepare for CUSUM calculation
        self.cusum_up = np.zeros(len(self.price_data))
        self.cusum_down = np.zeros(len(self.price_data))
        self.reference_values = np.zeros(len(self.price_data))

        self._fit_reference_values()

    def _fit_polynomial(self, x, y, degree):
        """
        Fits a polynomial of given degree to the data points.

        :param x: X-values (time or indices)
        :param y: Y-values (price data)
        :param degree: Degree of the polynomial.
        :return: Polynomial function (coefficients).
        """
        coeffs = np.polyfit(x, y, degree)
        poly_func = np.poly1d(coeffs)
        return poly_func
```

```
def _fit_spline(self, x, y):
    """
    Fits a cubic spline to the data points.

    :param x: X-values (time or indices)
    :param y: Y-values (price data)
    :return: CubicSpline function.
    """
    spline = CubicSpline(x, y, bc_type='natural')
    return spline

def _fit_reference_values(self):
    """
    Fit reference values using polynomial and spline for the price data.
    """
    # Generate an index array corresponding to price data
    x = np.arange(len(self.price_data))

    for i in range(self.window_size, len(self.price_data)):
        window_x = x[i - self.window_size:i]
        window_y = self.price_data[i - self.window_size:i]

        # Fit a polynomial to the window of data
        poly_func = self._fit_polynomial(window_x, window_y, self.poly_degree)
        poly_reference = poly_func(i)
        self.reference_values[i] = poly_reference

        # Fit a cubic spline to the window of data
        spline_func = self._fit_spline(window_x, window_y)
        spline_reference = spline_func(i)
        self.reference_values[i] = spline_reference  # Overwrite with spline fit

def _compute_cusum(self):
    """
    Computes the CUSUM based on the reference values (either polynomial or spline).
    """
    for i in range(1, len(self.price_data)):
        deviation = self.price_data[i] - self.reference_values[i]

        if deviation > 0:
```

```
            self.cusum_up[i] = max(0, self.cusum_up[i-1] + deviation)
            self.cusum_down[i] = 0
        else:
            self.cusum_down[i] = min(0, self.cusum_down[i-1] + deviation)
            self.cusum_up[i] = 0

    def detect_anomalies(self):
        """
        Detects anomalies based on the CUSUM threshold.

        :return: A list of anomaly indices where CUSUM crosses the threshold.
        """
        anomalies = []
        for i in range(len(self.price_data)):
            if abs(self.cusum_up[i]) > self.threshold or abs(self.cusum_down[i]) >
self.threshold:
                anomalies.append(i)
        return anomalies

    def plot_results(self):
        """
        Visualize the price data, reference values, and CUSUM results.
        """
        plt.figure(figsize=(14, 7))

        # Plot price data and reference values
        plt.subplot(2, 1, 1)
        plt.plot(self.price_data, label='Price Data', color='blue', alpha=0.7)
        plt.plot(self.reference_values, label='Fitted Reference Values', color='red',
linestyle='--')
        plt.title('Price Data and Fitted Reference (Polynomial/Spline)')
        plt.legend()

        # Plot CUSUM
        plt.subplot(2, 1, 2)
        plt.plot(self.cusum_up, label='CUSUM Up', color='green')
        plt.plot(self.cusum_down, label='CUSUM Down', color='orange')
        plt.axhline(y=self.threshold, color='black', linestyle='--', label='Threshold')
        plt.axhline(y=-self.threshold, color='black', linestyle='--')
        plt.title('CUSUM Anomaly Detection')
```

```
        plt.legend()

        plt.tight_layout()
        plt.show()

# Example usage
if __name__ == "__main__":
    # Simulated price data (e.g., daily stock prices)
    np.random.seed(42)
    price_data = np.cumsum(np.random.randn(200)) + 100  # Cumulative sum to
simulate price data with noise

    # Initialize and run the CUSUM with nonlinear reference values
    cusum_model = CUSUMNonlinear(price_data, threshold=5, window_size=50,
poly_degree=3)
    cusum_model._compute_cusum()

    # Detect anomalies
    anomalies = cusum_model.detect_anomalies()
    print(f"Anomalies detected at indices: {anomalies}")

    # Visualize the results
    cusum_model.plot_results()
```

3. CUSUM with Multi-Asset Correlations (Cross-Market Influence between BTC and ETH)

Explanation of Key Components:

1. **Multi-Asset Correlation:**
    - **This method uses the rolling correlation between the two assets (e.g., BTC and ETH) to adjust the reference value for the first asset based on its relationship with the second asset. The rolling correlation is computed over a sliding window of `window_size` data points.**
    - **If the correlation is above the specified `correlation_threshold`, the reference value for the first asset is adjusted by adding the**

scaled difference between the two assets.

2. **Polynomial Fitting:**
    - **Polynomial regression (using `np.polyfit`) is applied to the adjusted price data, which incorporates the correlation between the assets. The degree of the polynomial (`poly_degree`) can be adjusted based on the complexity of the market trends you're modeling.**

3. **CUSUM Calculation:**
    - **Standard CUSUM logic is applied using the reference values derived from the multi-asset correlation. CUSUM accumulates positive or negative deviations from the reference value, and if the cumulative sum exceeds a certain threshold (`threshold`), it indicates an anomaly.**

4. **Anomaly Detection:**
    - **Anomalies are detected when the CUSUM value (either up or down) crosses the threshold, indicating a significant deviation from the expected behavior of the asset.**

Python Code Implementation:-

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.interpolate import CubicSpline


class CUSUMMultiAsset:

    def __init__(self, price_data_1, price_data_2, threshold=5, window_size=50, poly_degree=3, correlation_threshold=0.7):

        """
```

```
    Initializes the CUSUM with multi-asset correlations.

    :param price_data_1: Price data for the first asset (e.g., BTC).
    :param price_data_2: Price data for the second asset (e.g., ETH).
    :param threshold: CUSUM threshold for detecting significant deviations.
    :param window_size: The size of the window for correlation-based reference fitting.
    :param poly_degree: Degree of the polynomial for fitting (default is cubic).
    :param correlation_threshold: Minimum correlation required to adjust the reference value.
    """
    self.price_data_1 = np.array(price_data_1)
    self.price_data_2 = np.array(price_data_2)
    self.threshold = threshold
    self.window_size = window_size
    self.poly_degree = poly_degree
    self.correlation_threshold = correlation_threshold

    # Prepare for CUSUM calculation
    self.cusum_up = np.zeros(len(self.price_data_1))
    self.cusum_down = np.zeros(len(self.price_data_1))
    self.reference_values = np.zeros(len(self.price_data_1))

    self._fit_reference_values()
```

```
def _compute_correlation(self, x, y):
    """
    Compute the rolling correlation between two assets over a sliding window.

    :param x: Price data for the first asset (e.g., BTC).
    :param y: Price data for the second asset (e.g., ETH).
    :return: A list of correlation values.
    """
    correlations = np.zeros(len(x) - self.window_size)
    for i in range(self.window_size, len(x)):
        window_x = x[i - self.window_size:i]
        window_y = y[i - self.window_size:i]
        correlation = np.corrcoef(window_x, window_y)[0, 1]
        correlations[i - self.window_size] = correlation
    return correlations

def _fit_polynomial(self, x, y, degree):
    """
    Fits a polynomial of given degree to the data points.

    :param x: X-values (time or indices)
    :param y: Y-values (price data)
    :param degree: Degree of the polynomial.
```

```
        :return: Polynomial function (coefficients).
        """
        coeffs = np.polyfit(x, y, degree)
        poly_func = np.poly1d(coeffs)
        return poly_func

    def _fit_reference_values(self):
        """
        Fits reference values using polynomial fitting, adjusted for the correlation between
the assets.
        """
        # Generate an index array corresponding to price data
        x = np.arange(len(self.price_data_1))

        # Compute rolling correlation between assets
        correlation_values = self._compute_correlation(self.price_data_1,
self.price_data_2)

        for i in range(self.window_size, len(self.price_data_1)):
            window_x = x[i - self.window_size:i]
            window_y_1 = self.price_data_1[i - self.window_size:i]
            window_y_2 = self.price_data_2[i - self.window_size:i]
            correlation = correlation_values[i - self.window_size]
```

```
        # If correlation is above the threshold, adjust the reference value for Asset 1 using Asset 2
        if abs(correlation) >= self.correlation_threshold:
            adjusted_window_y = window_y_1 + correlation * (window_y_2 - window_y_1)
        else:
            adjusted_window_y = window_y_1  # If correlation is low, use Asset 1's own price

        # Fit a polynomial to the adjusted price data
        poly_func = self._fit_polynomial(window_x, adjusted_window_y, self.poly_degree)
        poly_reference = poly_func(i)
        self.reference_values[i] = poly_reference

    def _compute_cusum(self):
        """
        Computes the CUSUM based on the reference values (adjusted for correlation).
        """
        for i in range(1, len(self.price_data_1)):
            deviation = self.price_data_1[i] - self.reference_values[i]

            if deviation > 0:
                self.cusum_up[i] = max(0, self.cusum_up[i-1] + deviation)
                self.cusum_down[i] = 0
```

```
            else:
                self.cusum_down[i] = min(0, self.cusum_down[i-1] + deviation)
                self.cusum_up[i] = 0

    def detect_anomalies(self):
        """
        Detects anomalies based on the CUSUM threshold.

        :return: A list of anomaly indices where CUSUM crosses the threshold.
        """
        anomalies = []
        for i in range(len(self.price_data_1)):
            if abs(self.cusum_up[i]) > self.threshold or abs(self.cusum_down[i]) >
self.threshold:
                anomalies.append(i)
        return anomalies

    def plot_results(self):
        """
        Visualize the price data, reference values, and CUSUM results.
        """
        plt.figure(figsize=(14, 7))

        # Plot price data and reference values
```

```
        plt.subplot(2, 1, 1)
        plt.plot(self.price_data_1, label='Asset 1 Price (e.g., BTC)', color='blue', alpha=0.7)
        plt.plot(self.reference_values, label='Adjusted Reference Values (CUSUM)',
color='red', linestyle='--')
        plt.title('Asset Price and Adjusted Reference Values')
        plt.legend()

        # Plot CUSUM
        plt.subplot(2, 1, 2)
        plt.plot(self.cusum_up, label='CUSUM Up', color='green')
        plt.plot(self.cusum_down, label='CUSUM Down', color='orange')
        plt.axhline(y=self.threshold, color='black', linestyle='--', label='Threshold')
        plt.axhline(y=-self.threshold, color='black', linestyle='--')
        plt.title('CUSUM Anomaly Detection (Up/Down)')
        plt.legend()

        plt.tight_layout()
        plt.show()



# Example usage
if __name__ == "__main__":
    # Simulated price data for two assets (e.g., BTC and ETH)
    np.random.seed(42)
```

```
    price_data_btc = np.cumsum(np.random.randn(200)) + 100  # Simulate BTC price data with noise

    price_data_eth = np.cumsum(np.random.randn(200)) + 50   # Simulate ETH price data with noise



    # Initialize and run the CUSUM model with multi-asset correlation

    cusum_model = CUSUMMultiAsset(price_data_btc, price_data_eth, threshold=5, window_size=50, poly_degree=3)

    cusum_model._compute_cusum()



    # Detect anomalies

    anomalies = cusum_model.detect_anomalies()

    print(f"Anomalies detected at indices: {anomalies}")



    # Visualize the results

    cusum_model.plot_results()
```

Customization:

Correlation Threshold: Adjust `correlation_threshold` to determine when to adjust the reference value based on the relationship between the assets. Default is `0.7`.

Window Size: Modify `window_size` to define how many data points are used for calculating correlation and fitting the polynomial. Default is `50`.

Polynomial Degree: Change `poly_degree` to control the flexibility of the reference value model (e.g., linear, quadratic, cubic).

V-mask and **Dynamic Volatility Threshold for CUSUM:-**

V-mask: This will act as a dynamic envelope around the CUSUM chart, helping to identify when the CUSUM values exceed acceptable bounds (the arms of the V-mask). It flags potential regime shifts or deviations when these bounds are violated.

Dynamic Volatility Threshold: Instead of using a fixed threshold for anomaly detection, we'll dynamically adjust the CUSUM threshold (k) based on market volatility. The threshold will be based on the rolling standard deviation of the price (volatility) over a defined window, adjusted by a factor (δ).

Steps to Integrate V-mask and Dynamic Volatility Threshold:

1. **Rolling Volatility**: Calculate the rolling standard deviation (σ) of the asset's price over a sliding window. This will give us an adaptive measure of market volatility.

2. **Dynamic Threshold Calculation**: Using the volatility, compute a dynamic threshold $k = \delta * \sigma$. This ensures that during periods of high volatility, the threshold will increase, and during periods of low volatility, the threshold will decrease.

3. **V-mask Construction**: Calculate the arms of the V-mask around the CUSUM plot. These will be based on the moving average of the asset prices or an estimated baseline. The arms of the V-mask will widen as the volatility increases, and contract as volatility decreases.

4. **CUSUM Logic**: The CUSUM calculation will compare the asset price against the dynamic threshold, and if the CUSUM exceeds the arms of the V-mask, an anomaly or regime shift will be detected.

Python code for implementing this approach:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class CUSUMWithVMaskAndDynamicThreshold:
    def __init__(self, price_data, window_size=50, delta=1.5, initial_threshold=5):
        """
        Initializes the CUSUM with V-mask and dynamic volatility threshold.

        :param price_data: List or numpy array of historical price data.
        :param window_size: The size of the window for rolling calculations.
        :param delta: Constant factor for dynamic threshold adjustment.
        :param initial_threshold: Initial CUSUM threshold for detecting significant
deviations.
        """
        self.price_data = np.array(price_data)
        self.window_size = window_size
        self.delta = delta
        self.initial_threshold = initial_threshold

        # Prepare for CUSUM calculation
        self.cusum_up = np.zeros(len(self.price_data))
        self.cusum_down = np.zeros(len(self.price_data))
        self.reference_values = np.zeros(len(self.price_data))
        self.thresholds = np.zeros(len(self.price_data))

        self._fit_reference_values()
        self._compute_dynamic_threshold()
        self._compute_cusum()

    def _fit_reference_values(self):
        """
        Fits reference values using a simple moving average (SMA).
        """
        self.reference_values =
pd.Series(self.price_data).rolling(window=self.window_size).mean().to_numpy()
        self.reference_values[:self.window_size] =
self.reference_values[self.window_size]

    def _compute_dynamic_threshold(self):
```

```
        """
        Computes the dynamic volatility threshold based on rolling standard
deviation.
        """
        rolling_std_dev =
pd.Series(self.price_data).rolling(window=self.window_size).std().to_numpy()
        self.thresholds = self.delta * rolling_std_dev
        self.thresholds[:self.window_size] = self.initial_threshold

    def _compute_cusum(self):
        """
        Computes the CUSUM based on the reference values and dynamic
thresholds.
        """
        for i in range(1, len(self.price_data)):
            deviation = self.price_data[i] - self.reference_values[i]

            if deviation > 0:
                self.cusum_up[i] = max(0, self.cusum_up[i-1] + deviation -
self.thresholds[i])
                self.cusum_down[i] = 0
            else:
                self.cusum_down[i] = min(0, self.cusum_down[i-1] + deviation +
self.thresholds[i])
                self.cusum_up[i] = 0

    def detect_anomalies(self):
        """
        Detects anomalies based on the CUSUM and V-mask approach.

        :return: A list of anomaly indices where CUSUM crosses the dynamic
thresholds.
        """
        anomalies = []
        for i in range(len(self.price_data)):
            if abs(self.cusum_up[i]) > self.thresholds[i] or abs(self.cusum_down[i]) >
self.thresholds[i]:
                anomalies.append(i)
        return anomalies
```

```
    def plot_results(self):
        """
        Visualize the price data, reference values, CUSUM results, and V-mask.
        """
        plt.figure(figsize=(14, 10))

        # Plot price data and reference values
        plt.subplot(3, 1, 1)
        plt.plot(self.price_data, label='Price Data', color='blue', alpha=0.7)
        plt.plot(self.reference_values, label='Reference Values (SMA)', color='red',
linestyle='--')
        plt.title('Price Data and Reference Values')
        plt.legend()

        # Plot dynamic thresholds
        plt.subplot(3, 1, 2)
        plt.plot(self.thresholds, label='Dynamic Threshold', color='purple',
linestyle='--')
        plt.title('Dynamic Volatility Threshold')
        plt.legend()

        # Plot CUSUM with V-mask
        plt.subplot(3, 1, 3)
        plt.plot(self.cusum_up, label='CUSUM Up', color='green')
        plt.plot(self.cusum_down, label='CUSUM Down', color='orange')
        plt.axhline(y=0, color='black', linestyle='--')
        for i in range(len(self.price_data)):
            plt.vlines(i, self.cusum_up[i] - self.thresholds[i], self.cusum_up[i] +
self.thresholds[i], colors='grey', alpha=0.5)
        plt.title('CUSUM with V-mask and Dynamic Volatility Threshold')
        plt.legend()

        plt.tight_layout()
        plt.show()


# Example usage
if __name__ == "__main__":
    # Simulated price data
    np.random.seed(42)
```

```
    price_data = np.cumsum(np.random.randn(200)) + 100  # Simulate price data with noise

    # Initialize and run the CUSUM model with V-mask and dynamic threshold
    cusum_model = CUSUMWithVMaskAndDynamicThreshold(price_data, window_size=50, delta=1.5, initial_threshold=5)

    # Detect anomalies
    anomalies = cusum_model.detect_anomalies()
    print(f"Anomalies detected at indices: {anomalies}")

    # Visualize the results
    cusum_model.plot_results()
```

Kelly Criterion: A Quantitative Approach for Alpha Trading Algorithms

The **Kelly Criterion** is a mathematical formula used to determine the optimal size of a series of bets or trades in order to maximize long-term growth of wealth. Developed by John L. Kelly Jr. in 1956, it is widely used in calculated gambling and financial markets, particularly in risk management and position sizing.

For **quantitative traders**, the Kelly Criterion can be used to optimize position sizing in trading algorithms, balancing risk and reward, with the goal of maximizing the compound growth of capital.

Kelly Formula:

The Kelly Criterion formula for the optimal bet size is:

$$f^* = \frac{p}{l} - \frac{1-p}{u}$$

Where:

- f^* is the fraction of your current portfolio to bet or invest.
- p is the probability of a win (or a profitable trade).
- l is the payoff for a loss (usually expressed as a negative value of the potential loss).
- u is the payoff for a win (the potential profit).

In trading, it's more commonly expressed as:

$$f^* = \frac{E[R]}{Var[R]}$$

Where:

- f^* is the optimal fraction of the capital to invest.
- $E[R]$ is the expected return of the asset (or trade).
- $Var[R]$ is the variance (or risk) of the asset's return.

Mathematical and Statistical Underpinnings:

1. **Expected Return**: $E[R] = p \cdot u - (1 - p) \cdot l$
   - p is the probability of a successful outcome.
   - u is the potential profit from a successful trade.
   - l is the potential loss from an unsuccessful trade.
2. **Risk or Variance**: The variance $Var[R]$ quantifies the risk of a trade. A high variance means high risk, and a low variance means lower risk. Variance is typically calculated from the distribution of asset returns.

$$Var[R] = p \cdot (u - E[R])^2 + (1 - p) \cdot (l - E[R])^2$$

3. **Maximizing Logarithmic Growth**: Kelly Criterion maximizes the long-term growth of wealth by choosing a position size that maximizes the **logarithmic growth** of the portfolio. The Kelly Criterion's optimal bet size f^* ensures that the geometric growth of capital is maximized over time.
4. **Logarithmic Utility**: The Kelly strategy is tied to **log utility**, meaning it seeks to maximize the logarithm of wealth. This is optimal under the assumption of compound returns, as maximizing logarithmic utility results in the highest possible long-term growth of wealth.

Using the Kelly Criterion in Trading Algorithms:

1. Expected Return Calculation (Alpha Generation):

In quantitative trading, the expected return E[R] can be derived from factors such as:

- **Historical Returns**: Statistical analysis of past price movements.
- **Factor Models**: Expected returns from factor exposure (e.g., value, momentum, size, or quality factors).
- **Machine Learning Models**: Predictive models that generate future return estimates.

For example:

E[R] = Expected Return from Model

2. Risk/Volatility (Variance):

The variance of returns can be derived from historical volatility or predictive volatility models:

- **Historical Volatility**: Calculated using the standard deviation of asset returns over a time window.

- **GARCH Models**: Statistical models that forecast time-varying volatility, allowing for dynamic risk adjustment.

- **Monte Carlo Simulations**: Can be used to model potential future return distributions and their variance.

For example:

Var[R]=Variance of returns from historical data or model predictionVar[R]=Variance of returns from historical data or model prediction

3. Optimal Position Size:

Once the expected return E[R] and variance Var[R] are calculated, the optimal position size f∗ is determined using the Kelly formula:

$$f^* = \frac{E[R]}{Var[R]}$$

The algorithm can dynamically adjust the position size based on these parameters. For example, when volatility (variance) is high, the algorithm will scale back position sizes, reducing risk exposure.

We are providing the **dynamic position sizing functionality in SDK.**

Key Considerations:

1. **Overfitting Risk**:

   The Kelly Criterion assumes that historical data is a good predictor of future returns and volatility. In reality, overfitting can lead to poor out-of-sample performance. Traders should incorporate **regularization techniques** or **robust models** to mitigate this.

2. **Fractional Kelly**:

   To reduce the risk of large drawdowns, many traders use a **Fractional Kelly** approach, where the position size is scaled down to a fraction of the optimal Kelly

bet (e.g., 50% of the calculated optimal size). This helps reduce the risk of large losses while still benefiting from the growth maximization property of the Kelly Criterion.

3. **Sharpe Ratio Maximization**:

   The Kelly Criterion is closely related to the **Sharpe Ratio**, which also aims to maximize return per unit of risk. Traders can use the Kelly position sizing formula in tandem with Sharpe Ratio optimization to refine their trading strategies.

Enhancing Indicator Signals with Logarithmic Transformations, Oscillations, and the Hawks Process

Financial and trading indicators are pivotal in decision-making for market entries and exits. However, raw indicator values can sometimes lead to noise or suboptimal signals. Applying mathematical transformations such as logarithmic scaling, oscillation functions, or combining these methods with the Hawks process can refine these signals, yielding better insights. This document explores these techniques and their application.

1. Logarithmic Transformations

Logarithmic transformations scale data logarithmically, compressing higher values while expanding lower ones. This transformation is particularly useful when dealing with data that exhibits exponential growth or has a wide range of values.

Application in Indicators

- **Reduce Noise:** Indicators with high volatility often produce exaggerated spikes. Applying a log function to their values smoothens these spikes, making patterns more discernible.

- **Enhance Proportional Relationships:** Logarithmic scaling can reveal proportional relationships between price movements and indicator levels, improving interpretability.

- **Example:** For an RSI (Relative Strength Index) that often swings rapidly, applying log(1 + RSI) can stabilize its movements without altering its relative trends.

2. Oscillation Techniques

Oscillation involves periodic fluctuation or transformation of values around a mean, often through trigonometric functions like sine or cosine. It is useful for smoothing or re-aligning raw data to highlight cyclic trends.

Application in Indicators

- **Highlight Cyclic Behavior:** Many financial instruments follow cycles. Oscillation helps emphasize these cycles, aligning indicators with market rhythms.

- **Normalize Data:** By bounding values (e.g., between -1 and 1 for sine waves), oscillation ensures uniformity, reducing outliers' impact.

- **Example:** Transforming an EMA (Exponential Moving Average) using a sine function, sin(EMA), can emphasize periodicity in price movements.

3. Combining Logarithms and Oscillations

A combination of logarithmic scaling and oscillation brings the benefits of both transformations. Logarithms stabilize large fluctuations, while oscillation captures underlying cycles. Together, they refine signals by balancing stability and periodicity.

Application in Indicators

- **Smooth Cycles with Reduced Bias:** For an indicator like MACD (Moving Average Convergence Divergence), a combined transformation, such as sin(log(1 + MACD)), can stabilize large swings and emphasize turning points.

- **Enhance Trend Reversals:** These combined transformations often reveal subtle trend reversals that raw values might obscure.

4. Integrating the Hawks Process

The Hawks process, a statistical model for capturing self-exciting events, is useful for detecting bursts of activity in financial data. It models event probabilities as a function of prior occurrences, making it highly effective for identifying clustering behavior in markets.

Application in Indicators

- **Dynamic Entry/Exit Points:** By applying the Hawks process to transformed indicator values, traders can identify clusters of signal activity that precede major market movements.

- **Refinement of Noise Filtering:** When combined with logarithmic and oscillatory transformations, the Hawks process can adaptively filter noise and focus on statistically significant signal clusters.

● **Example:** If an oscillator-transformed RSI is fed into a Hawks model, the resulting probabilities highlight periods of sustained overbought or oversold conditions.

5. Practical Implementation Steps

1. **Choose Indicators:** Select indicators relevant to your strategy (e.g., RSI, MACD, EMA).

2. **Apply Transformations:** Use logarithmic scaling, oscillation, or a combination on the raw indicator values.

3. **Integrate Hawks Process:** Feed the transformed values into a Hawks process to model and detect signal clusters.

4. **Backtest and Optimize:** Evaluate the modified indicators against historical data, adjusting parameters to maximize performance.

5. **Deploy in Real-Time:** Implement the refined signals in a live trading environment, monitoring performance closely.

6. Benefits and Considerations

- Enhanced signal clarity and reduced noise.
- Improved detection of entry and exit points.
- Better alignment with market dynamics and cycles.

Considerations

● **Complexity:** Combining transformations and statistical models requires computational resources and expertise.

● **Parameter Sensitivity:** Fine-tuning parameters is essential to avoid overfitting or misinterpretation.

● **Market-Specific Behavior:** The effectiveness of these methods may vary across markets and asset classes.

Hurst Exponent: A Quantitative Approach

The **Hurst exponent** can be derived through various methods, with the **Rescaled Range Analysis (R/S analysis)** and **DFA (Detrended Fluctuation Analysis)** being two of the most common techniques. The formula for the Hurst exponent in the context of a time series (xt) is typically expressed as:

$$R/S = \frac{\text{Range of cumulative deviations from the mean}}{\text{Standard deviation of the series}}$$

Where:

- **Range** is the difference between the maximum and minimum values of the cumulative sum of deviations.
- **Standard deviation** is the variability of the time series.

The relationship between the **R/S ratio** and the size of the time window N follows a power-law:

$$R/S \sim N^H$$

Here, the exponent H describes the time series' behavior:

- $H = 0.5$: **Random Walk** (no autocorrelation, similar to Brownian motion).
- $H < 0.5$: **Mean-Reverting** (the series tends to return to the mean).
- $H > 0.5$: **Trending** (the series exhibits a tendency to follow a trend).

Using the Hurst Exponent in Trading Algorithms:

1. Identifying Market Regimes:

- **Trending Regimes:** If the Hurst exponent H>0.5, the market is likely in a persistent (trending) state. This suggests that momentum strategies could be effective, as trends are expected to continue. Use strategies that capitalize on momentum, such as **moving averages**, **trend-following models** or **mathematical combinations of indicators** that exploit the persistence of trends.

- **Mean-Reverting Regimes:** If H<0.5, the market exhibits mean-reverting behavior. In such cases, mean-reversion strategies may perform better.

- **Random Walk:** If H=0.5 , the series is a random walk, indicating no established trend or mean reversion. In such regimes, traditional trend-following and mean-reverting strategies would likely fail.

Portfolio Optimization:

By measuring the Hurst exponent of different assets or asset classes, traders can optimize portfolio construction **by allocating more capital to assets that show persistent trends (H>0.5H>0.5) and less to mean-reverting assets (H<0.5)**

We are providing the **dynamic position sizing functionality in SDK.**

Lead_Lag_Signal_Generator

The `LeadLagSignalGenerator` class is designed to generate trading signals for Ethereum (ETH) by analyzing the lead-lag relationship between Bitcoin (BTC) and Ethereum.

1. Lead-Lag Relationship

- **Core Strength:** The code explicitly models BTC's influence on ETH using lagged correlations. This aligns with the observed behavior in crypto markets, where BTC often leads ETH due to BTC's dominant market role.

- **Advantage:** Identifying and using the optimal lag ensures that the strategy dynamically adapts to changing lead-lag relationships.

---

2. Momentum and Volatility Thresholds

- **Core Strength:** BTC momentum and volatility thresholds act as filters to avoid reacting to minor, noisy price movements. This ensures only significant trends are acted upon.
- **Advantage:** Prevents overtrading in choppy or low-volatility markets, enhancing strategy robustness.

---

3. Base System Integration

- Core Strength: The integration of a pre-existing base system (e.g., trend, volatility, and correlation metrics) adds additional layers of confirmation for trading signals.
- Advantage: This multi-factor approach reduces the risk of false signals and improves overall signal reliability.

---

4. Dynamic Adjustments

- Core Strength: Position sizes and stop-loss/take-profit levels are dynamically adjusted based on BTC momentum strength. This risk management framework is critical in highly volatile markets like cryptocurrencies.
- Advantage: Scaling positions and stops ensures that the strategy is conservative during high uncertainty and aggressive during strong trends, maximizing risk-adjusted returns.

---

5. Rolling Statistics

- Core Strength: Rolling calculations for momentum, volatility, and Hurst exponent capture short-term changes in market dynamics.
- Advantage: Ensures that the strategy remains responsive to real-time shifts in market behavior.

Objective: Generate long/short signals for ETH based on BTC momentum and volatility, combined with base system statistics.

1. **Calculate Returns:**

- Logarithmic returns for ETH and BTC are computed using:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

2. **Find Optimal Lag:**
   - **Calls `find_optimal_lag` to determine the best lag where BTC leads ETH.**

3. **Base Statistics:**
   - **Fetches additional statistics (e.g., ETH trend, volatility, correlation) from the `base_system`.**

4. **Lagged BTC Features:**
   - **Momentum: Rolling mean of BTC returns over `lead_window`, lagged by the optimal lag.**
   - **Volatility: Rolling standard deviation of BTC returns over `lead_window`, lagged by the optimal lag.**

5. **Generate Signals:**
   - **BTC Momentum Signal:**
     - **If BTC momentum exceeds \text{signal_threshold} \times \text{BTC Volatility}, a long signal is generated (+1).**
     - **If it falls below -\text{signal_threshold} \times \text{BTC Volatility} , a short signal is generated (-1).**
     - **Otherwise, no signal (0).**

   - **Combine BTC Signal with Base Conditions:**
     - **Long Conditions:**
       - **High ETH-BTC correlation (>0.7).**
       - **ETH volatility is higher than BTC volatility.**
       - **ETH trend (Hurst exponent) indicating a trending market (>0.6).**
       - **BTC momentum positive.**

     - **Short Conditions:**
       - **Low ETH-BTC correlation (<0.3).**
       - **ETH volatility is significantly higher than BTC volatility (1.5×).**
       - **ETH trend indicating mean reversion (<0.4).**
       - **BTC momentum negative.**

Method: `calculate_enhanced_position_sizes`

Objective: Scale position sizes based on BTC momentum strength.

Logic:

1. Retrieve signals and BTC momentum/volatility from `lead_lag_stats`.
2. Calculate the **momentum strength**:

$$\text{Momentum Strength} = \frac{|\text{BTC Momentum}|}{\text{BTC Volatility}}$$

3. Scale the strength between 0.1 and 1.0 to avoid extreme position sizes.
4. Adjust the base system's position sizes by the scaled momentum strength.

Supporting Logic

1. Lead-Lag Analysis

- BTC's movements often precede ETH's due to market dominance, making BTC an effective leading indicator.

2. Rolling Metrics

- Rolling averages for momentum and volatility capture short-term trends.
- Hurst exponent distinguishes trending (>0.5) from mean-reverting (<0.5) behavior.

3. Dynamic Adjustments

- Position sizes, stops, and targets are dynamically adapted to BTC momentum, reducing risk during high volatility and enhancing profits during strong trends.

Code Snippet:-

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
```

```
from scipy import stats

class LeadLagSignalGenerator:
    def __init__(self, base_system: StatisticalTradingSystem,
                 lead_window: int = 12,  # Hours for lead-lag relationship
                 signal_threshold: float = 0.6):
        self.base_system = base_system
        self.lead_window = lead_window
        self.signal_threshold = signal_threshold

    def find_optimal_lag(self, btc_returns: np.array,
                         eth_returns: np.array,
                         max_lag: int = 24) -> int:
        """
        Find optimal lag between BTC and ETH movements
        """
        correlations = []
        for lag in range(1, max_lag + 1):
            corr = np.corrcoef(btc_returns[:-lag], eth_returns[lag:])[0, 1]
            correlations.append(corr)

        return np.argmax(correlations) + 1

    def generate_lead_lag_signals(self, eth_prices: np.array,
                                  btc_prices: np.array,
                                  volumes: Dict[str, np.array] = None) -> Dict[str, np.array]:
        """
        Generate trading signals based on BTC's leading indicators
        """
        # Calculate returns
        eth_returns = np.diff(np.log(eth_prices))
        btc_returns = np.diff(np.log(btc_prices))

        # Find optimal lag
        opt_lag = self.find_optimal_lag(btc_returns, eth_returns)

        # Get base statistics
        base_stats = self.base_system.generate_trading_signals(eth_prices, btc_prices)

        # Calculate lagged features from BTC
        btc_mom = pd.Series(btc_returns).rolling(self.lead_window).mean().shift(opt_lag)
        btc_vol = pd.Series(btc_returns).rolling(self.lead_window).std().shift(opt_lag)

        # Generate lead-lag signals
```

```
    signals = np.zeros_like(eth_returns)

    for i in range(len(signals)):
        if i < max(self.lead_window, opt_lag):
            continue

        # BTC momentum signals
        btc_signal = 1 if btc_mom[i] > self.signal_threshold * btc_vol[i] else \
                -1 if btc_mom[i] < -self.signal_threshold * btc_vol[i] else 0

        # Combine with base system conditions
        if btc_signal > 0:
            # Long conditions with BTC confirmation
            long_conditions = [
                base_stats['correlation']['correlation'][i] > 0.7,  # High correlation
                base_stats['eth_vol']['volatility'][i] > base_stats['btc_vol']['volatility'][i],
                base_stats['eth_trend']['hurst'][i] > 0.6,  # Trending market
                btc_mom[i] > 0  # BTC showing upward momentum
            ]

            if all(long_conditions):
                signals[i] = 1

        elif btc_signal < 0:
            # Short conditions with BTC confirmation
            short_conditions = [
                base_stats['correlation']['correlation'][i] < 0.3,  # Decorrelation
                base_stats['eth_vol']['volatility'][i] > base_stats['btc_vol']['volatility'][i] * 1.5,
                base_stats['eth_trend']['hurst'][i] < 0.4,  # Mean-reverting
                btc_mom[i] < 0  # BTC showing downward momentum
            ]

            if all(short_conditions):
                signals[i] = -1

    return {
        'signals': signals,
        'base_stats': base_stats,
        'btc_momentum': btc_mom,
        'btc_volatility': btc_vol,
        'optimal_lag': opt_lag
    }

def calculate_enhanced_position_sizes(self,
```

```
                              lead_lag_stats: Dict[str, np.array],
                              max_pos: float = 1.0) -> np.array:
    """
    Calculate position sizes incorporating lead-lag relationship
    """
    signals = lead_lag_stats['signals']
    base_stats = lead_lag_stats['base_stats']

    # Get base position sizes
    base_sizes = self.base_system.calculate_position_sizes(
        signals, base_stats, max_pos)

    # Adjust based on BTC momentum strength
    btc_mom_strength = np.abs(lead_lag_stats['btc_momentum']) / \
                       lead_lag_stats['btc_volatility']
    mom_scale = np.clip(btc_mom_strength, 0.1, 1.0)

    # Scale positions
    adjusted_sizes = base_sizes * mom_scale

    return adjusted_sizes

def calculate_enhanced_stops(self,
                             lead_lag_stats: Dict[str, np.array],
                             base_tp: float = 0.03,
                             base_sl: float = 0.02) -> Tuple[np.array, np.array]:
    """
    Calculate dynamic stops incorporating BTC information
    """
    # Get base stop levels
    base_tp, base_sl = self.base_system.calculate_dynamic_stops(
        lead_lag_stats['base_stats'], base_tp, base_sl)

    # Adjust based on BTC momentum
    btc_mom_strength = np.abs(lead_lag_stats['btc_momentum']) / \
                       lead_lag_stats['btc_volatility']

    # Widen stops when BTC momentum is strong
    tp_adj = np.clip(1 + btc_mom_strength, 1.0, 2.0)
    sl_adj = np.clip(1 - btc_mom_strength * 0.5, 0.5, 1.0)

    return base_tp * tp_adj, base_sl * sl_adj
```

Potential Drawbacks:-

1. Lag Optimization Limitations

- **Weakness:** The optimal lag is determined solely by maximizing correlation. This assumes that the lead-lag relationship is linear and stationary, which may not always hold true.

- **Alpha Decay Risk:** Non-linear or time-varying lead-lag relationships could result in suboptimal lag selection, reducing signal accuracy.

2. Fixed Signal Thresholds

- **Weakness**: The fixed momentum threshold (`signal_threshold = 0.6`) may not generalize well across different market conditions or timeframes.

- **Alpha Decay Risk**: During periods of extreme volatility or low liquidity, fixed thresholds may fail to adapt, leading to missed opportunities or increased false signals.

Improvements:-

1. Adaptive Quantile-Based Thresholds

- **How It Works**:
  - Calculate rolling quantiles (e.g., 70th percentile for high correlation, 30th percentile for low correlation) based on historical data.
  - Use these quantiles as thresholds for signal generation instead of fixed values.
- **Statistical Significance**:
  - Ensures that thresholds adapt to recent market conditions, reducing the risk of overfitting to past data.
  - Quantile-based methods are statistically robust in capturing distribution shifts over time.

---

2. Kernel Density Estimation (KDE)

- **How It Works**:
    - Estimate the probability density function of indicators (e.g., correlation, momentum) using KDE.
    - Identify natural thresholds based on density peaks or inflection points.
- **Statistical Significance**:
    - Accounts for multimodal distributions, enabling better distinction between market regimes.
    - Reduces noise from arbitrary thresholds, aligning thresholds with the actual data distribution.

---

3. Non-Linear Correlation Analysis

- **Methods**:
    - **Copula Functions**:
        - Model joint distributions of BTC and ETH returns to capture tail dependencies and non-linear relationships.
    - **Mutual Information**:
        - Quantify the shared information between BTC and ETH returns to detect complex dependencies.

ATR-Differential Early Exit approach

The core idea of the ATR-Differential Early Exit approach is rooted in the assumption that:

- **Volatility Regime Changes**: Financial markets are often driven by shifts in volatility regimes. A sharp rise in the ATR differential (i.e., the change in ATR between periods) is a reliable indicator of a sudden increase in market activity or uncertainty.

- **Dynamic Nature of Volatility**: Sudden increases or decreases in ATR indicate transitions between trending, consolidating, or chaotic market states. By observing these shifts, traders can preemptively manage risks or capitalize on potential opportunities.

- **Volatility Rate of Change**: Unlike absolute ATR values, the rate of change offers an earlier warning system for market transitions. It allows traders to respond before volatility stabilizes, avoiding unnecessary drawdowns or missed opportunities.

How the System Becomes Dynamic

- **Adaptability Over Static Rules**:
    - Instead of reacting to **fixed ATR thresholds, the system observes how ATR is evolving (i.e., the differential)**. For instance, a sharp increase in ATR from one period to the next (e.g., a 20% rise) signals increasing market volatility, regardless of the absolute ATR level.

    - This makes the strategy dynamic because it adjusts to current market conditions rather than adhering to predefined values.

- **Sensitivity to Market Conditions**:
    - If the market is calm, even a small ATR differential might indicate an impending shift. In contrast, during high-volatility periods, only significant ATR changes might matter.

    - By incorporating differential logic, the system ensures responsiveness without being overly reactive.

---

3. Conditions for Long/Short Trades and Exits Based on ATR Differential

Long Trade Conditions

1. **Entry:**
    - ATR Differential (ΔATR) **shows a decreasing trend** for the last 3 periods (dont go with 3 , try to find the best optimized value) , indicating a stabilization of volatility or the start of a calmer upward trend.

    - Price action confirms bullish momentum (your indicators or combination of indicators with volume indicators for confirmation suggest bullish signal)

      You can pair ATR differential signals with trend indicators like ADX, RSI, or MACD for directional confirmation , just a suggestion.

2. Exit:

- ATR Differential spikes upward by more than a pre-defined percentage (e.g., 25-30%) within a single period.
- This indicates a sudden increase in volatility, potentially signaling a reversal or consolidation phase.

Short Trade Conditions

1. **Entry:**
   - ATR Differential **shows an increasing trend** for the last 3 periods (indicating a building bearish momentum or breakdown).
   - Price action confirms bearish momentum ((your indicators or combination of indicators with volume indicators for confirmation suggest bullish signal).
2. **Exit:**
   - ATR Differential decreases sharply by more than a pre-defined percentage (e.g., a drop of 20% or more).
   - This suggests weakening of bearish momentum and potential consolidation or trend reversal.

Dynamic Thresholds:

- Use **rolling averages or standard deviations of ATR differential to set adaptive thresholds** for identifying significant changes.