Hello Teams

We are providing participants **4 in-house strategy codes** alongside a **comprehensive quant approaches document** that creates a powerful bridge between theoretical knowledge and practical application.

By analyzing each line of code, we encourage students to **first go through all 4 strategy codes line by line, carefully analyzing the logic behind each decision, understanding why specific indicators were chosen, and recognizing their role in trade execution and risk management.**

Each strategy **employs a unique blend of technical indicators—ranging from trend-following methods like Heiken Ashi and Supertrend to advanced statistical techniques such as Kalman filters and cross-asset Z-score analysis.** This hands-on exposure allows students to understand the rationale behind signal generation, risk management, and trade execution.

The quant approaches document enhances the learning experience by contextualizing methodologies within broader market dynamics, **enabling students to refine, optimize, and innovate on existing strategies while also building their own from scratch using advanced concepts.**

Roadmap Ahead

1. Learn Practical Implementation: The 4 strategy codes give students a hands-on look at how a trading idea evolves from concept to executable logic. They can dissect the structure, syntax, and flow of professional-grade code, which is a critical skill when translating their own ideas into something actionable.

2. Spark Creativity and Innovation: By studying the in-house strategies, students can take inspiration and tweak them—adjusting parameters, combining elements from different strategies, or even hybridizing them with their own ideas. This encourages creative problem-solving, a key trait for success in quant trading.

3. Reverse-Engineer Thinking with Codes: Alternatively, students can start with the four codes to trace the line of thinking behind each strategy—why certain indicators or rules were chosen—and then cross-reference the quant approaches document to deepen their understanding. This helps them see how abstract concepts translate into working models.

4. Develop a Structured Quantitative Mindset: Exposure to in-house strategies gives participants a glimpse into industry standards. By analyzing how the in-house strategies were built, they can identify patterns of logic and reasoning, then use the document to expand that framework into new, original strategies.
They can use the **quant approaches document** to explore more complex approaches there—and apply these lessons to refine the provided codes or build their own unique logics from scratch.

Quant Approaches Document

https://docs.google.com/document/d/17G44VbkKvXkNyz4RmJzPRic3aL4vyHZjVA3J0IxQCJQ/edit?tab=t.0#heading=h.8lmk8h76k3zv

To maximize learning and efficiency, we encourage teams of 3-4 members to divide tasks strategically by forming sub-teams, ensuring a thorough exploration of the Quant Approaches Document. Each sub-team should focus on different sections, analyzing the key methodologies, statistical techniques, and trading concepts outlined in the document.

The next step is to identify concepts from the document that aligns with your team's trading goals. By leveraging these insights, teams can refine existing strategies or develop entirely new models tailored to market dynamics.

Vector platform to Create & Code-

Register yourself with Vector **- vector.untrade.io**

Docs to understand functionality of Vector - https://docs-quant.untrade.io/Vector.html#

User Guide to Vector (Algorithmic Trading Platform)

This guide will walk you through the steps of using the Vector platform to create, code, and run your own trading algorithm (algo). The guide covers:

1. Logging into Vector
2. Connecting to Telegram
3. Creating a Strategy
4. Implementing Your Strategy in `main.py`
5. Code Structure Requirements
6. Basic Price Data, Signals, and Trade Types
7. Example Strategy Implementation
8. Paper Testing Your Algorithm
9. Understanding Classes in Python
10. Running Your Algo in Live Environment

BTC Algorithm 1

```
import talib as ta
import pandas as pd

def __process_data(df, atr_period=10):
    # Heiken Ashi Candles
    df["ha_close"] = (df["open"] + df["high"] + df["low"] + df["close"]) / 4
    df["ha_open"] = (df["open"].shift(1) + df["close"].shift(1)) / 2
    df["ha_high"] = df[["high", "low", "ha_open", "ha_close"]].max(axis=1)
    df["ha_low"] = df[["high", "low", "ha_open", "ha_close"]].min(axis=1)
```

```
    # Initialize the position and signal columns
    df["Position"] = 0
    df["signals"] = 0

    # caclulate the required indicators
    df["adx"] = ta.ADX(df["high"], df["low"], df["close"], timeperiod=15)
    df["atr"] = ta.ATR(df["high"], df["low"], df["close"], atr_period)
    df["DailyReturns"] = df["close"].pct_change()
    df["WeeklyReturns"] = df["DailyReturns"].rolling(5).sum()

    return df


# Strat Logic
def __strat(df, atr_multiplier=4):
    for i in range(1, len(df) - 1):
        # Get the previous and current values of the indicators and candle
        prev_ha_close = df.loc[i - 1, "ha_close"]
        curr_ha_close = df.loc[i, "ha_close"]
        prev_ha_open = df.loc[i - 1, "ha_open"]
        curr_ha_open = df.loc[i, "ha_open"]
        curr_atr = df.loc[i, "atr"]

        # long condition
        if (
            prev_ha_close > prev_ha_open
            and curr_ha_close > curr_ha_open
            and df["adx"][i] < 25
        ) and df["Position"][i] != 1:
            # Enter a long position and set the signal to 1
            df.loc[i + 1, "Position"] = 1
            df.loc[i, "signals"] = df.loc[i + 1, "Position"] - df.loc[i, "Position"]
            # Set the initial stop loss to the current low minus the ATR multiplied by the multiplier
            stop_loss = df.loc[i, "high"] - curr_atr * (atr_multiplier + 1)

        # short condition
        elif (
            prev_ha_close > prev_ha_open
            and curr_ha_close < curr_ha_open
            and df["adx"][i] < 25
        ) and df["Position"][i] != -1:
            # Enter a short position and set the signal to -1
            df.loc[i + 1, "Position"] = -1
            df.loc[i, "signals"] = df.loc[i + 1, "Position"] - df.loc[i, "Position"]
```

```
            # Set the initial stop loss to the current high plus the ATR multiplied by the multiplier
            stop_loss = df.loc[i, "low"] + curr_atr * (atr_multiplier - 1)

        # secondary long condition
        elif (
            df["adx"][i] > 60 and df["Position"][i] != 1 and df["WeeklyReturns"][i] < 0
        ):
            df.loc[i + 1, "Position"] = 1
            df.loc[i, "signals"] = df.loc[i + 1, "Position"] - df.loc[i, "Position"]
            # Set the initial stop loss to the current low minus the ATR multiplied by the multiplier
            stop_loss = df.loc[i, "high"] - curr_atr * (atr_multiplier + 1)

        # secondary short condition
        elif (
            df["adx"][i] > 60 and df["Position"][i] != -1 and df["WeeklyReturns"][i] > 0
        ):
            # Enter a short position and set the signal to -1
            df.loc[i + 1, "Position"] = -1
            df.loc[i, "signals"] = df.loc[i + 1, "Position"] - df.loc[i, "Position"]
            # Set the initial stop loss to the current high plus the ATR multiplied by the multiplier
            stop_loss = df.loc[i, "low"] + curr_atr * (atr_multiplier - 1)

        # stoploss implementation
        else:
            # If the current position is long
            if df.loc[i, "Position"] == 1:
                # Carry over the position
                df.loc[i + 1, "Position"] = 1
                # Check if the current low is below the stop loss
                if df.loc[i, "low"] < stop_loss:
                    # Exit the position and set the signal to -1
                    df.loc[i + 1, "Position"] = 0
                    df.loc[i, "signals"] = -1

                # Otherwise, update the stop loss to the maximum of the previous stop loss and the
current high minus the ATR multiplied by the multiplier
                else:
                    stop_loss = max(
                        stop_loss, df.loc[i, "high"] - curr_atr * (atr_multiplier + 1)
                    )
            # If the current position is short
            elif df.loc[i, "Position"] == -1:
                # Carry over the position
                df.loc[i + 1, "Position"] = -1
```

```
            # Check if the current high is above the stop loss
            if df.loc[i, "high"] > stop_loss:
                # Exit the position and set the signal to 1
                df.loc[i + 1, "Position"] = 0
                df.loc[i, "signals"] = 1

            # Otherwise, update the stop loss to the minimum of the previous stop loss and the
current low plus the ATR multiplied by the multiplier
            else:
                stop_loss = min(
                    stop_loss, df.loc[i, "low"] + curr_atr * (atr_multiplier - 1)
                )

        else:
            df.loc[i + 1, "Position"] = 0

    return df
```

BTC Algorithm 2

```
import pandas as pd
import numpy as np

def __calculate_heikin_ashi(df):

    df = df.copy()

    df["HA_Close"] = (df["open"] + df["high"] + df["low"] + df["close"]) / 4

    df["HA_Open"] = df["HA_Close"].copy()

    df.at[0, "HA_Open"] = df.at[0, "open"]

    for i in range(1, len(df)):
        df.at[i, "HA_Open"] = (df.at[i - 1, "HA_Open"] + df.at[i - 1, "HA_Close"]) / 2

    df["HA_High"] = df[["HA_Open", "HA_Close", "high"]].max(axis=1)
    df["HA_Low"] = df[["HA_Open", "HA_Close", "low"]].min(axis=1)
```

```python
    return df


def __calculate_atr(df, period=80):
    """Calculates Average True Range (ATR)."""
    df["refC1"] = df["HA_Close"].shift(1)

    df["TR1"] = df["HA_High"] - df["HA_Low"]
    df["TR2"] = (df["HA_High"] - df["refC1"]).abs()
    df["TR3"] = (df["HA_Low"] - df["refC1"]).abs()
    df["true_range"] = df[["TR1", "TR2", "TR3"]].max(axis=1)

    df["ATR"] = np.nan
    first_atr_value = df["true_range"].iloc[1 : period + 1].mean()
    df.loc[period, "ATR"] = first_atr_value

    for i in range(period + 1, len(df)):
        df.loc[i, "ATR"] = (
            df["ATR"].iloc[i - 1] * (period - 1) + df["true_range"].iloc[i]
        ) / period

    return df["ATR"]


def __calculate_percent_rank(series, periods=80):
    percent_ranks = pd.Series(index=series.index, dtype=float)

    for i in range(periods - 1, len(series)):
        count = 0

        for j in range(1, periods + 1):
            if i - j >= 0 and series.iloc[i] > series.iloc[i - j]:
                count += 1
        # Calculate the percent rank
        percent_ranks.iloc[i] = 100.0 * count / periods

    return percent_ranks


def __calculate_supertrend(df, atr_period=80):
    df = df.copy()
    iatr = __calculate_atr(df, atr_period)

    df["factor"] = np.where(df["pct_ATR"] >= 26, 7.5, 8)
```

```
    df["basic_upper"] = (df["HA_High"] + df["HA_Low"]) / 2 + (df["factor"] * iatr)
    df["basic_lower"] = (df["HA_High"] + df["HA_Low"]) / 2 - (df["factor"] * iatr)

    # Initialize the trend Series
    trend = pd.Series(1, index=df.index)
    trendup = pd.Series(np.nan, index=df.index)
    trenddown = pd.Series(np.nan, index=df.index)

    for i in range(1, len(df)):
        trend[i] = (
            1
            if df["HA_Close"].iloc[i] > df["basic_upper"].iloc[i - 1]
            else (
                -1
                if df["HA_Close"].iloc[i] < df["basic_lower"].iloc[i - 1]
                else trend[i - 1]
            )
        )
        df.loc[i, "basic_lower"] = (
            max(df["basic_lower"].iloc[i], df["basic_lower"].iloc[i - 1])
            if trend[i] == 1
            else df["basic_lower"].iloc[i]
        )
        df.loc[i, "basic_upper"] = (
            min(df["basic_upper"].iloc[i], df["basic_upper"].iloc[i - 1])
            if trend[i] == -1
            else df["basic_upper"].iloc[i]
        )
        trendup[i] = df["basic_lower"].iloc[i] if trend[i] == 1 else np.nan
        trenddown[i] = df["basic_upper"].iloc[i] if trend[i] == -1 else np.nan

    df["trend"] = trend
    df["trendup"] = trendup
    df["trenddown"] = trenddown

    return df


def __calculate_trigger_lines(df, x=18, y=105):
    df["Trig_line_long"] = (
        df["HA_Close"].rolling(window=x, min_periods=1).min().shift(1)
    )
    df["Trig_line_short"] = (
        df["HA_Close"].rolling(window=y, min_periods=1).max().shift(1)
```

```python
    )
    return df


def __process_data(df):

    df = __calculate_heikin_ashi(df)

    df["ATR"] = __calculate_atr(df)
    df["pct_ATR"] = __calculate_percent_rank(df["ATR"])
    df = __calculate_supertrend(df)
    df = __calculate_trigger_lines(df)
    return df


def __strat(df):
    # Assume 'Trig_line_short' and 'Trig_line_long' already exist in the DataFrame
    # Initialize the signal columns with 0s
    df["buy"] = 0
    df["sell"] = 0
    df["short"] = 0
    df["cover"] = 0

    # Initialize a variable to track the last signal type
    last_signal = None
    df["trade_type"] = "hold"

    # Generate buy/sell/short/cover signals based on trend changes and trigger lines
    for i in range(1, len(df)):
        Short_trig = df.loc[i, "HA_Close"] > df.loc[i, "Trig_line_short"]
        Long_trig = df.loc[i, "HA_Close"] < df.loc[i, "Trig_line_long"]

        # Entering a Long position when conditions are met
        if (
            df.loc[i, "trend"] == 1
            and Long_trig
            and last_signal != "buy"
            and last_signal != "short"
        ):
            df.loc[i, "buy"] = 1
            df.loc[i, "tradeType"] = TradeType.LONG.value
            last_signal = "buy"

        # Entering a Short position when conditions are met
```

```
        elif (
            df.loc[i, "trend"] == -1
            and Short_trig
            and last_signal != "short"
            and last_signal != "buy"
        ):
            df.loc[i, "short"] = 1
            df.loc[i, "tradeType"] = TradeType.SHORT.value
            last_signal = "short"

        # Covering a Short position when conditions are met
        elif (
            df.loc[i - 1, "trend"] == -1
            and df.loc[i, "trend"] == 1
            and last_signal != "cover"
            and last_signal == "short"
        ):
            df.loc[i, "cover"] = 1
            df.loc[i, "tradeType"] = TradeType.CLOSE.value
            last_signal = "cover"

        # Selling a Long position when conditions are met
        elif (
            df.loc[i - 1, "trend"] == 1
            and df.loc[i, "trend"] == -1
            and last_signal != "sell"
            and last_signal == "buy"
        ):
            df.loc[i, "sell"] = 1
            df.loc[i, "tradeType"] = TradeType.CLOSE.value
            last_signal = "sell"

    # Combine signals into a single column
    df["signals"] = df["buy"] - df["sell"] - df["short"] + df["cover"]

    return df
```

ETH Algorithm 1

```
import uuid
 # ALL your imports here
```

```python
import pandas as pd
import numpy as np
import ta
from ta.volume import OnBalanceVolumeIndicator
from ta.trend import EMAIndicator


def process_data(data):
    # Calculate OBV and OBV Oscillator
    obv_indicator = OnBalanceVolumeIndicator(close=data['close'], volume=data['volume'])
    data['obv'] = obv_indicator.on_balance_volume()
    window = 20
    # Calculate the OBV Oscillator as the difference between OBV and its EMA
    short_ema_length = 20
    ema_obv = EMAIndicator(close=data['obv'], window=short_ema_length)
    data['obv_ema'] = ema_obv.ema_indicator()
    data['obv_osc'] = data['obv'] - data['obv_ema']
    # Calculate Long-Term EMA for trend confirmation
    long_ema_length = 65
    long_ema = EMAIndicator(close=data['close'], window=long_ema_length)
    data['ma_long_term'] = long_ema.ema_indicator()
    return data


 # -------STRATEGY LOGIC--------#
def strat(data):
    data['signals'] = 0  # 1 for buy, -1 for sell, 0 for hold
    data["tradeType"] = "hold"
    position_opened = None  # Track whether a position is open
    entry_price = None  # Track entry price for calculating SL and TP
    sl = None  # Stop-loss level
    long_ema_length = 65
    # Generate buy/sell signals based on OBV Oscillator and EMA-based trend confirmation
    for i in range(long_ema_length, len(data)):
        # Check conditions for a buy signal
        if position_opened is None:
            #Long entry
            if data['close'].iloc[i] > data['ma_long_term'].iloc[i]:
                # Buy condition 1: OBV Oscillator is positive and percentage increase > 10%
                if data['obv_osc'].iloc[i] > 0 and data['obv_osc'].pct_change().iloc[i] > 0.10:
                    data.at[i,'signals'] = 1  # Buy signal
                    data.at[i,'tradeType'] = TradeType.LONG.value
                    position_opened = 1  # Open position
                    entry_price = data['close'][i]
```

```
            sl = entry_price * 0.95  # Set 5% stop-loss level

        # Check conditions for a sell signal if a position is open
        elif position_opened == 1:
            # Sell condition: OBV Oscillator crosses below zero
            if (data['obv_osc'].iloc[i] < 0 < data['obv_osc'].iloc[i - 1] and
                  data['close'].iloc[i] < data['ma_long_term'].iloc[i]):
                data.at[i,'signals'] = -1  # Sell signal to close position
                data.at[i,'tradeType'] = TradeType.CLOSE.value
                position_opened = None  # Close position
                entry_price = None  # Reset entry price
                sl = None  # Reset stop-loss

            elif (data['close'].iloc[i] <= sl):
                data.at[i,'signals'] = -1  # Sell signal to close position
                data.at[i,'tradeType'] = TradeType.CLOSE.value
                position_opened = None  # Close position
                entry_price = None  # Reset entry price
                sl = None  # Reset stop-loss
    return data
```

ETH Algorithm 2 - Based on Btc (lead) and Eth (lag) correlation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas_ta as ta
from pprint import pprint

def apply_kalman_filter(data, prediction_variance=1e-5, observation_variance=0.01,
                        new_column_name="close"):
    data = data.copy()

    estimated_price = data["close"].values[0]
    estimated_error = 1.0       # Initial estimation error
    kalman_prices = []          # Store the filtered prices

    for price in data["close"].values:
        # Kalman filter calculations remain the same
        estimated_error += prediction_variance
        kalman_gain = estimated_error / (estimated_error + observation_variance)
```

```
        estimated_price += kalman_gain * (price - estimated_price)
        estimated_error = (1 - kalman_gain) * estimated_error
        kalman_prices.append(estimated_price)

    data.loc[:, "kalman"] = kalman_prices

    # Rename columns
    if new_column_name == "close":
        data = data.rename(columns={"close": "raw_close", "kalman": new_column_name})
    else:
        data = data.rename(columns={"kalman": new_column_name})

    return data

def hawkes_process(data, kappa):
    assert(kappa > 0.0)
    alpha = np.exp(-kappa)
    arr = data.to_numpy()
    output = np.full(len(data), np.nan)
    mean_arrival_rate = data.rolling(window=168, min_periods=1).mean()

    for i in range(1, len(data)):
        if np.isnan(output[i - 1]):
            output[i] = mean_arrival_rate.iloc[i] + arr[i]
        else:
            output[i] = output[i - 1] * alpha + mean_arrival_rate.iloc[i] + arr[i]

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

def vol_signal(close, vol_hawkes, lookback):
    signal = np.zeros(len(close))
    q05 = vol_hawkes.rolling(lookback).quantile(0.05)
    q95 = vol_hawkes.rolling(lookback).quantile(0.95)

    last_below = -1
    curr_sig = 0

    for i in range(len(signal)):
        if vol_hawkes.iloc[i] < q05.iloc[i]:
            last_below = i
            curr_sig = 0

        if i == 0:
            prev_vol = np.nan
```

```
        else:
            prev_vol = vol_hawkes.iloc[i - 1]

        if (vol_hawkes.iloc[i] > q95.iloc[i] and
            (np.isnan(prev_vol) or vol_hawkes.iloc[i - 1] <= q95.iloc[i - 1]) and
            last_below > 0):

            change = close.iloc[i] - close.iloc[last_below]
            if change > 0.0:
                curr_sig = 1
            else:
                curr_sig = -1
        signal[i] = curr_sig

    return signal

def hawkes_process_v_spikes(data, factor, kappa, lookback):
    data = data.copy()

    lookback = lookback * factor
    data['datetime'] = data['datetime'].astype('datetime64[s]')
    data = data.set_index('datetime')
    norm_lookback = 336 * factor

    data.loc[:, 'atr'] = ta.atr(np.log(data['high']), np.log(data['low']), np.log(data['close']),
norm_lookback)
    data.loc[:, 'norm_range'] = (np.log(data['high']) - np.log(data['low'])) / data['atr']

    data.loc[:, 'v_hawk'] = hawkes_process(data['norm_range'], kappa)
    data.loc[:, 'v_sig'] = vol_signal(data['close'], data['v_hawk'], lookback)
    data.reset_index(inplace=True)
    return data['v_sig']

def calculate_bollinger_bands(data, factor, window=20, num_sd=2):
    window = window * factor
    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, factor, short_window=12, long_window=26, signal_window=9):
    short_window = short_window * factor
    long_window = long_window * factor
```

```python
    signal_window = signal_window * factor
    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, factor, k_window=14, d_window=3):
    k_window = int(k_window * factor)
    d_window = int(d_window * factor)
    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, factor, window=14):
    window = window * factor
    tr1 = data['high'] - data['low']
    tr2 = (data['high'] - data['close'].shift(2)).abs()
    tr3 = (data['low'] - data['close'].shift(2)).abs()
    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 calculate_rsi(data, factor, window=14):
    window = window * factor
    return ta.rsi(data['close'], length=window)

def adx(data, factor, window=14):
    adx_period = 7 * factor
    adx_threshold = 10 * factor  # ADX threshold for filtering weak trends
```

```
    data = data.copy()

    # Step 1: Calculate the True Range (TR), +DM, -DM
    data.loc[:, 'TR'] = np.maximum(data['high'] - data['low'],
                            np.maximum(abs(data['high'] - data['close'].shift(1)),
                                    abs(data['low'] - data['close'].shift(1))))
    data.loc[:, '+DM'] = np.where(
        (data['high'] - data['high'].shift(1)) > (data['low'].shift(1) - data['low']),
        data['high'] - data['high'].shift(1), 0)
    data.loc[:, '+DM'] = np.where(data['+DM'] < 0, 0, data['+DM'])
    data.loc[:, '-DM'] = np.where(
        (data['low'].shift(1) - data['low']) > (data['high'] - data['high'].shift(1)),
        data['low'].shift(1) - data['low'], 0)
    data.loc[:, '-DM'] = np.where(data['-DM'] < 0, 0, data['-DM'])

    # Step 2: Calculate smoothed TR, +DI, and -DI
    data.loc[:, 'TR_smooth'] = data['TR'].rolling(window=adx_period).sum()
    data.loc[:, '+DI'] = 100 * (data['+DM'].rolling(window=adx_period).sum() / data['TR_smooth'])
    data.loc[:, '-DI'] = 100 * (data['-DM'].rolling(window=adx_period).sum() / data['TR_smooth'])

    # Step 3: Calculate the ADX
    data.loc[:, 'DX'] = 100 * (abs(data['+DI'] - data['-DI']) / (data['+DI'] + data['-DI']))
    data.loc[:, 'ADX'] = data['DX'].rolling(window=adx_period).mean()

    # Step 4: Apply ADX filter to avoid trades in low-trend markets
    data.loc[:, 'adx_trade'] = np.where(data['ADX'] >= adx_threshold, 1, 0)
    return data['adx_trade']

def process_helper(data):
    data = data.copy()

    data = apply_kalman_filter(data)
    data = apply_kalman_filter(data, prediction_variance=1e-7, observation_variance=1,
new_column_name="regime_analysis_close")
    factor = 4

    data.loc[:, 'BB_upper'], data.loc[:, 'BB_lower'] = calculate_bollinger_bands(data, factor)
    data.loc[:, 'MACD_line'], data.loc[:, 'MACD_signal'] = calculate_macd(data, int(factor/2))
    data.loc[:, 'Stoch_K'], data.loc[:, 'Stoch_D'] = calculate_stochastic(data, factor)
    data.loc[:, 'ATR'] = calculate_atr(data, factor)
    data.loc[:, 'OBV'] = calculate_obv(data)
    data.loc[:, 'OBV_MA'] = data['OBV'].rolling(window=20*factor).mean()
    data.loc[:, 'vol_ma'] = data['volume'].rolling(window=20*factor).mean()
    data.loc[:, 'RSI'] = calculate_rsi(data, factor)
```

```python
    data.loc[:, 'SMA'] = data['close'].rolling(window=50).mean()
    data.loc[:, 'SMA_long'] = data['close'].rolling(window=200).mean()
    data.loc[:, 'SMA_2000'] = data['regime_analysis_close'].rolling(window=2000).mean()
    data.loc[:, 'SMA_1000'] = data['regime_analysis_close'].rolling(window=1000).mean()
    data.loc[:, 'v_signal'] = hawkes_process_v_spikes(data, factor, kappa=0.75, lookback=168)
    data.loc[:, 'adx_trade'] = adx(data, factor)
    return data

def is_bullish_market(data, i):
    if (1.025 * data.loc[i, 'SMA_2000'] < data.loc[i, 'regime_analysis_close'] and
        1.025 * data.loc[i, 'SMA_2000'] < data.loc[i, 'SMA_1000']):
        data.loc[i, 'is_bullish'] = True
    else:
        data.loc[i, 'is_bullish'] = False
    return data.loc[i, 'is_bullish']

def strat_helper(data):
    data = data.copy()

    # Set parameters
    atr_multiplier = 1.5
    profit_target_multiplier = 2
    percent_stop_loss = 0.3
    factor = 4

    # Initialize columns using .loc
    data.loc[:, 'signal'] = 0  # Initialize signal column
    data.loc[:, 'stop_loss'] = np.nan  # Initialize stop-loss column
    data.loc[:, 'take_profit'] = np.nan  # Initialize take-profit column
    data.loc[:, 'trade_type'] = None  # Initialize trade type column
    data.loc[:, 'is_bullish'] = False
    prev = 0 # Initialize previous signal
    in_position = 0  # Position flag
    current_stop_loss = np.nan  # Initialize stop-loss variable
    trade_open = False  # Initialize trade flag
    current_take_profit = np.nan # Initialize take-profit variable

    long_open = 0  # Initialize long trade flag
    long_close = 0  # Initialize long trade flag
    short_open = 0  # Initialize short trade flag
    short_close = 0  # Initialize short trade flag
    long_stop_loss = 0  # Initialize long stop-loss flag
    short_stop_loss = 0  # Initialize short stop-loss flag
```

```
    # Signal logic
    for i in range(0, len(data), factor):
        if i >= len(data):
            break  # Prevent index out of range
        idx = data.index[i]

        # Update 'is_bullish' status
        is_bullish = is_bullish_market(data, i)

        # Buy condition:
        if ((is_bullish) or
           (data.loc[idx, 'MACD_line'] > data.loc[idx, 'MACD_signal'] and
            data.loc[idx, 'close'] > data.loc[idx, 'SMA_long'] and
            data.loc[idx, 'OBV'] > data.loc[idx, 'OBV_MA'] and
            data.loc[idx, 'adx_trade'] == 1 and
            data.loc[idx, 'volume'] > data.loc[idx, 'vol_ma'] and
            data.loc[idx, 'v_signal'] == 1)
            ):  # Adjust Hawkes process condition

            if not trade_open:
                prev = in_position
                in_position = 1
                data.loc[idx, 'signal'] = 1
                data.loc[idx, 'trade_type'] = 'long'
                trade_open = True
                long_open += 1

                current_stop_loss = data.loc[idx, 'close'] - (atr_multiplier * data.loc[idx, 'ATR'])
                current_stop_loss = max(data.loc[idx, 'close'] - percent_stop_loss * data.loc[idx,
'close'], current_stop_loss)
                current_take_profit = data.loc[idx, 'close'] + (profit_target_multiplier * data.loc[idx,
'ATR'])
                data.loc[idx, 'stop_loss'] = current_stop_loss  # Store stop-loss in DataFrame
                data.loc[idx, 'take_profit'] = current_take_profit

            elif prev == -1:
                prev = in_position
                in_position = 1
                data.loc[idx, 'signal'] = 2
                data.loc[idx, 'trade_type'] = 'short_reversal'
                long_open += 1
                short_close += 1

                current_stop_loss = data.loc[idx, 'close'] - (atr_multiplier * data.loc[idx, 'ATR'])
```

```
            current_stop_loss = max(data.loc[idx, 'close'] - percent_stop_loss * data.loc[idx,
'close'], current_stop_loss)
            current_take_profit = data.loc[idx, 'close'] + (profit_target_multiplier * data.loc[idx,
'ATR'])
            data.loc[idx, 'stop_loss'] = current_stop_loss  # Store stop-loss in DataFrame
            data.loc[idx, 'take_profit'] = current_take_profit

      # Sell condition:
      elif (
          data.loc[idx, 'MACD_line'] < 1.5 * data.loc[idx, 'MACD_signal'] and
          1.1 * data.loc[idx, 'SMA_long'] > data.loc[idx, 'close'] and
          data.loc[idx, 'OBV'] < data.loc[idx, 'OBV_MA'] and
          data.loc[idx, 'adx_trade'] == 1 and
          data.loc[idx, 'volume'] > data.loc[idx, 'vol_ma'] and
          data.loc[idx, 'v_signal'] == -1
          ):  # Adjust Hawkes process condition

         if not trade_open:
            data.loc[idx, 'signal'] = -1
            data.loc[idx, 'trade_type'] = 'short'
            prev = in_position
            in_position = -1
            trade_open = True
            short_open += 1

            current_stop_loss = data.loc[idx, 'close'] + (atr_multiplier * data.loc[idx, 'ATR'])  # Set
initial stop-loss
            current_stop_loss = min(data.loc[idx, 'close'] + percent_stop_loss * data.loc[idx,
'close'], current_stop_loss)
            data.loc[idx, 'stop_loss'] = current_stop_loss  # Store stop-loss in DataFrame
            current_take_profit = data.loc[idx, 'close'] - (profit_target_multiplier * data.loc[idx,
'ATR'])
            data.loc[idx, 'take_profit'] = current_take_profit

         elif prev == 1:
            data.loc[idx, 'signal'] = -2
            data.loc[idx, 'trade_type'] = 'long_reversal'
            prev = in_position
            in_position = -1
            short_open += 1
            long_close += 1

            current_stop_loss = data.loc[idx, 'close'] + (atr_multiplier * data.loc[idx, 'ATR'])  # Set
initial stop-loss
```

```
                current_stop_loss = min(data.loc[idx, 'close'] + percent_stop_loss * data.loc[idx, 'close'], current_stop_loss)
                data.loc[idx, 'stop_loss'] = current_stop_loss  # Store stop-loss in DataFrame
                current_take_profit = data.loc[idx, 'close'] - (profit_target_multiplier * data.loc[idx, 'ATR'])
                data.loc[idx, 'take_profit'] = current_take_profit

        elif trade_open and data.loc[idx, 'volume'] < 0.1 * data.loc[data.index[i-1], 'vol_ma']:  # If in a long position
            if in_position == 1:
                data.loc[idx, 'signal'] = -1
                data.loc[idx, 'trade_type'] = 'close'
                in_position = 0  # Reset position
                trade_open = False
            else:
                data.loc[idx, 'signal'] = 1
                data.loc[idx, 'trade_type'] = 'close'
                in_position = 0  # Reset position
                trade_open = False

        # Stop-loss logic
        if data.loc[idx, 'signal'] == 0 and in_position == 1 and trade_open:  # If in a long position
            # Update stop-loss to trail if price moves up
            current_stop_loss = max(current_stop_loss, data.loc[idx, 'close'] - (atr_multiplier * data.loc[idx, 'ATR']))
            current_stop_loss = max(data.loc[idx, 'close'] - percent_stop_loss * data.loc[idx, 'close'], current_stop_loss)
            data.loc[idx, 'stop_loss'] = current_stop_loss
            data.loc[idx, 'take_profit'] = current_take_profit

            # If stop-loss is hit, close the long position
            if data.loc[idx, 'close'] <= current_stop_loss or data.loc[idx, 'close'] >= current_take_profit:
                long_stop_loss += 1
                data.loc[idx, 'signal'] = -1  # Close long position
                data.loc[idx, 'trade_type'] = 'close'
                in_position = 0  # Reset position
                current_stop_loss = np.nan  # Reset stop-loss
                current_take_profit = np.nan
                trade_open = False

        elif data.loc[idx, 'signal'] == 0 and in_position == -1 and trade_open:  # If in a short position
            # Update stop-loss to trail if price moves down
```

```
            current_stop_loss = min(current_stop_loss, data.loc[idx, 'close'] + (atr_multiplier *
data.loc[idx, 'ATR']))
            current_stop_loss = min(data.loc[idx, 'close'] + percent_stop_loss * data.loc[idx, 'close'],
current_stop_loss)
            data.loc[idx, 'stop_loss'] = current_stop_loss
            data.loc[idx, 'take_profit'] = current_take_profit

            # If stop-loss is hit, close the short position
            if (data.loc[idx, 'close'] > current_stop_loss or data.loc[idx, 'close'] <=
current_take_profit):
                short_stop_loss += 1
                data.loc[idx, 'signal'] = 1  # Close short position
                data.loc[idx, 'trade_type'] = 'close'
                in_position = 0  # Reset position
                current_stop_loss = np.nan  # Reset stop-loss
                current_take_profit = np.nan
                trade_open = False

    # Rename columns
    data = data.rename(columns={"close": "kal_close", "raw_close": "close", 'signal' : 'signals'})
    return data


def process_data(btc_data, eth_data):
    btc_data = btc_data.tail(len(eth_data)).copy()
    btc_data.index = eth_data.index
    btc_data = process_helper(btc_data)
    eth_data = process_helper(eth_data)
    return btc_data, eth_data

def strat(btc_data, eth_data):
    btc_data = strat_helper(btc_data)
    eth_data = strat_helper(eth_data)

    eth_data['signals'] = btc_data['signals'].shift(1).fillna(0).astype(int)
    eth_data['trade_type'] = btc_data['trade_type'].shift(1).fillna('')

    return eth_data
```

Question:- How do I ensure that my strategy is not having a look ahead bias?

Frontlook Bias Test

To ensure your strategy does not rely on future data (frontlook bias), implement the following test: Test Procedure

1. **Pick a Signal:**
   - Select a random signal from the log file generated by your algorithm.
   - For example, if you are backtesting from 2020-01-01 to 2023-12-31 and a signal is generated at 2023-01-01 11:00:00, pick this signal.
2. **Remove Future Data:**
   -In the OHLCV data, remove all data after 2023-01-01 11:00:00. -This ensures only data from 2020-01-01 to 2023-01-01 11:00:00 is available for the test.
3. **Re-run the Algorithm:**
   - Backtest the strategy using the truncated data.
4. **Verify the Signal:**

> - Check if the same signal is still generated at 2023-01-01 11:00:00.
> - If the signal is not generated, your strategy may have a frontlook bias.

By removing future data, this test ensures that the algorithm is not relying on information that would not have been available at the time of the signal.

---

7. How does commission accounting take place in the sdk library at Vector?

Answer:- So for every trade (both long and square off or short and square off included) - the fee is 0.15% irrespective of the Profit/loss booked in that trade.

For every trade - you are shelling out 1.5$ dollar (0.15% of 1000 dollars in static approach). Now let' say you have 2000 trades then = 2000*1.5$ amounting to 3000$ (final outgoing charges including commission + slippages).

Gross profit - commissions = Net profit

Question:- Could you provide a detailed overview of one of the in-house strategies that demonstrates profitability and effective risk management?

Specifically, how does this strategy's performance fluctuate on a quarterly basis, and what insights can we get from its results?

Answer:- This is the distribution of the quarterly results of our strategy called Omega which works on a 1 hour timeframe /frequency and is built ***on the foundation of Hawkes Process.***

It's a pure mathematical genius that precisely manages its drawdowns even in non-trendy markets , has good time to recovery rate , comes with good risk-reward between average win and average loss and yields more than 150% returns annually with less than 30% drawdowns in compounding.

In last 16 quarters spanning from 1st Jan 2020 to 1st Jan 2024:-

1. Only 1 quarter is in loss

2. Profitable in 15/16 quarters

3. Able to beat buy and hold returns i.e benchmark returns of BTC in 12/16 quarters

4. Average win rate is close to 60 percent in every quarter

Initial Balance	Final Balance	Profit(%)	Benchmark(%)	Benchmark Beaten?	From	To	Total Trades	Long Trades	Short Trades	Win Rate	Maximum Holding Time	Average Holding Time	Sharpe Ratio
1000	2006.99	100.70	-12.14	Yes	2020-01-01	2020-04-01	4	1	3	75.00	10 days 13:00:00	6 days 00:45:00	12.303185
1000	1131.42	13.14	44.79	No	2020-04-01	2020-07-01	8	5	3	50.00	9 days 05:00:00	3 days 06:07:30	5.740338
1000	1643.57	64.36	18.22	Yes	2020-07-01	2020-10-01	4	2	2	75.00	9 days 22:00:00	4 days 23:30:00	15.251524
1000	2295.59	129.56	169.01	No	2020-10-01	2021-01-01	7	5	2	85.71	8 days 20:00:00	4 days 12:34:17.142857142	29.205306
1000	1465.59	46.56	104.39	No	2021-01-01	2021-04-01	8	4	4	62.50	7 days 17:00:00	4 days 06:37:30	5.971744
1000	1711.23	71.12	-41.46	Yes	2021-04-01	2021-07-01	6	2	4	66.67	18 days 23:00:00	8 days 09:10:00	8.119172
1000	1628.62	62.86	25.79	Yes	2021-07-01	2021-10-01	6	4	2	83.33	9 days 22:00:00	3 days 08:50:00	23.678489
1000	1525.21	52.52	6.82	Yes	2021-10-01	2022-01-01	7	3	4	71.43	11 days 06:00:00	4 days 05:00:00	10.678369
1000	1313.61	31.36	-2.38	Yes	2022-01-01	2022-04-01	8	3	5	62.50	7 days 07:00:00	2 days 19:37:30	11.898458
1000	1207.63	20.76	-55.45	Yes	2022-04-01	2022-07-01	8	1	7	50.00	11 days 10:00:00	4 days 10:52:30	3.900777
1000	1090.60	9.06	-4.42	Yes	2022-07-01	2022-10-01	6	1	5	66.67	7 days 15:00:00	3 days 04:40:00	6.669147
1000	1162.24	16.22	-14.78	Yes	2022-10-01	2023-01-01	7	4	3	57.14	8 days 18:00:00	3 days 15:08:34.285714285	4.878081
1000	1739.94	73.99	72.02	Yes	2023-01-01	2023-04-01	5	3	2	60.00	11 days 11:00:00	7 days 02:36:00	10.329314
1000	1470.20	47.02	7.14	Yes	2023-04-01	2023-07-01	9	5	4	55.56	7 days 16:00:00	3 days 11:53:20	9.940330
1000	836.47	-16.35	-11.43	No	2023-07-01	2023-10-01	8	2	6	12.50	5 days 08:00:00	3 days 10:00:00	-2.951709
1000	1834.83	83.48	57.56	Yes	2023-10-01	2024-01-01	5	4	1	60.00	11 days 07:00:00	6 days 19:48:00	11.951905

Why are benchmark returns important ?

The buy and hold returns/benchmark returns in quarters of BTC USDT refers to purchasing BTC on the first day of quarter and holding it till the last day of quarter, regardless of market fluctuations. They serve as a standard to measure the performance of various investment strategies.

›

Any trading strategy should aim to outperform benchmark returns for several reasons:-

1. Value Addition: A successful strategy that consistently beats the benchmark adds value to the investor's portfolio.

2. Opportunity Cost: Failing to outperform the benchmark means missing out on potential gains that could be achieved through other investment options.

3. Credibility and Trust: Strategies that consistently deliver alpha (excess return over the benchmark) build credibility and trust with investors.

Example of Omega Strategy Performance

Consider the performance of the Omega strategy in the last quarter of 2023, specifically from *October 1, 2023, to January 1, 2024. During this period, the benchmark returns for Bitcoin were 57.56%.*

In contrast, the Omega strategy achieved a profit of 83.48%. This results in an alpha of 25.92 %,

Alpha = Strategy Return - Benchmark Return

Alpha = 83.48% - 57.56% = 25.92%

This example illustrates the effectiveness of the Omega strategy in outperforming the Bitcoin benchmark, demonstrating its potential as a viable trading strategy.

SMART EXIT MANAGEMENTS

```
import pandas as pd
import numpy as np
import talib as ta

# Identify Early Pullbacks
```

```python
def identify_early_pullbacks(df, momentum_period=5, threshold=0.005, trend_period_short=10,
trend_period_long=20):
    """
    Detects early pullbacks in trending markets using momentum and moving averages.

    Parameters:
        df (DataFrame): DataFrame with 'close', 'high', 'low' columns
        momentum_period (int): Period for momentum calculation
        threshold (float): Retracement threshold as a percentage
        trend_period_short (int): Short-term SMA period
        trend_period_long (int): Long-term SMA period

    Returns:
        DataFrame: Original DataFrame with pullback flags
    """
    df = df.copy()

    # Compute Momentum (Percentage Change)
    df['momentum'] = df['close'].pct_change(momentum_period)

    # Compute Moving Averages for Trend Confirmation
    df['SMA_short'] = df['close'].rolling(window=trend_period_short).mean()
    df['SMA_long'] = df['close'].rolling(window=trend_period_long).mean()

    # Compute Highest High and Lowest Low for Retracement Detection
    df['highest_high'] = df['high'].rolling(window=5).max()
    df['lowest_low'] = df['low'].rolling(window=5).min()

    # Identify Pullback in an Uptrend (Price Retracing Downward)
    df['pullback_uptrend'] = (
        (df['SMA_short'] > df['SMA_long']) &  # Uptrend condition
        (df['momentum'] < 0) &  # Momentum slowing
        (df['highest_high'] * (1 - threshold) > df['close'])  # Price retraced from high
    )

    # Identify Pullback in a Downtrend (Price Retracing Upward)
    df['pullback_downtrend'] = (
        (df['SMA_short'] < df['SMA_long']) &  # Downtrend condition
        (df['momentum'] > 0) &  # Momentum increasing
        (df['lowest_low'] * (1 + threshold) < df['close'])  # Price retraced from low
    )

    return df
```

```
# Apply Smart Trailing Stop with ATR and Pullback Detection
def apply_smart_trailing_stop(df, atr_multiplier=5):
    """
    Applies a smart trailing stop that adjusts based on ATR and detected pullbacks.

    Parameters:
        df (DataFrame): DataFrame with 'close', 'high', 'low', 'ATR', 'pullback_uptrend',
'pullback_downtrend'
        atr_multiplier (float): Multiplier for ATR-based stop distance

    Returns:
        DataFrame: DataFrame with trailing stop levels
    """
    long_stop_levels = []
    short_stop_levels = []
    active_long_stop = None
    active_short_stop = None

    for i in range(len(df)):
        price = df['close'].iloc[i]
        atr = df['ATR'].iloc[i]
        high = df['high'].iloc[i]
        low = df['low'].iloc[i]

        # Adjust Long Stop During Uptrend Pullback
        if df['pullback_uptrend'].iloc[i]:
            new_long_stop = low - (atr_multiplier * atr)  # Set stop below recent low
            if active_long_stop is None or new_long_stop > active_long_stop:
                active_long_stop = new_long_stop  # Tighten stop only if higher

        # Adjust Short Stop During Downtrend Pullback
        if df['pullback_downtrend'].iloc[i]:
            new_short_stop = high + (atr_multiplier * atr)  # Set stop above recent high
            if active_short_stop is None or new_short_stop < active_short_stop:
                active_short_stop = new_short_stop  # Tighten stop only if lower

        # Append current stop levels (None if no active trade)
        long_stop_levels.append(active_long_stop)
        short_stop_levels.append(active_short_stop)

    df['smart_trailing_stop_long'] = long_stop_levels
    df['smart_trailing_stop_short'] = short_stop_levels
    return df
```

```
# Example Usage
if __name__ == "__main__":
    # Sample DataFrame (replace with your data)
    df = pd.DataFrame({
        'close': np.random.rand(100) * 100 + 1000,
        'high': np.random.rand(100) * 100 + 1050,
        'low': np.random.rand(100) * 100 + 950,
        'open': np.random.rand(100) * 100 + 1000
    })

    # Add ATR (requires high, low, close)
    df['ATR'] = ta.ATR(df['high'], df['low'], df['close'], timeperiod=14)
    df['ATR'] = df['ATR'].fillna(df['ATR'].mean())  # Fill NaN for demo

    # Apply Pullback Detection and Trailing Stop
    df = identify_early_pullbacks(df)
    df = apply_smart_trailing_stop(df)

    print(df[['close', 'pullback_uptrend', 'pullback_downtrend',
              'smart_trailing_stop_long', 'smart_trailing_stop_short']].tail())
```

What’s Unique and Different About This ATR-Based Pullback Detection?

This ATR-based pullback detection method stands out from conventional trailing stop and pullback strategies in several ways:

1. **Integration of Momentum and Trend Context**:

    - **Conventional Approach**: Most trailing stops (e.g., ATR trailing stop, Chandelier Exit) rely solely on price and volatility (ATR) without considering momentum or trend direction explicitly.
    - **This Method**: Combines momentum (pct_change) with dual moving averages (SMA_short and SMA_long) to confirm the trend direction before identifying a pullback. For example, a pullback in an uptrend requires SMA_short > SMA_long and negative momentum, ensuring the retracement occurs within a bullish context.
    - **Uniqueness**: This multi-signal confluence reduces false positives by filtering out pullbacks in non-trending or choppy markets, unlike simpler ATR stops that might trigger in noise.

2. **Dynamic Pullback Threshold**:

    - **Conventional Approach**: Traditional pullback detection often uses fixed percentage retracements (e.g., 5% drop from a high) or Fibonacci levels, which are static and ignore volatility.
    - **This Method**:: Uses a threshold (0.005 or 0.5%) applied to the highest high/lowest low over a short window (5 periods), but pairs it with ATR in the trailing stop logic. The ATR multiplier (atr_multiplier=5) dynamically adjusts the stop distance based on current volatility.
    - **Uniqueness**: The synergy of a percentage-based pullback trigger with an ATR-scaled stop makes the method adaptive to both short-term price action and market volatility, unlike static methods that may over- or under-react.

3. **Early Pullback Detection**:

    - **Conventional Approach**: Many strategies wait for a pullback to complete (e.g., price crossing a moving average) before adjusting stops, potentially missing early exit opportunities.
    - **This Method**:: Identifies pullbacks early by detecting negative momentum in uptrends (or positive in downtrends) before the price fully reverses, then adjusts the stop proactively.
    - **Uniqueness**: This anticipatory approach allows the strategy to tighten stops during the initial stages of a pullback, preserving profits or limiting losses more effectively than reactive methods.

4. **Asymmetric Stop Adjustment**:

    - **Conventional Approach**: Standard trailing stops (e.g., moving the stop a fixed ATR distance below the highest high) update continuously, often leading to premature exits in volatile markets.
    - **This Method**: Only adjusts the stop when a pullback is detected (pullback_uptrend or pullback_downtrend), and only tightens it if the new level is more favorable (e.g., new_long_stop > active_long_stop). Otherwise, it holds the previous level.

      The "smart" part comes from adjusting stops only during pullbacks and ensuring they tighten monotonically (e.g., long stops only move up, short stops only move down). This preserves profits while avoiding premature exits in choppy markets.

    - **Uniqueness**: This conditional, asymmetric adjustment prevents over-tightening during minor fluctuations, balancing risk management with trend-following flexibility.

5. **ATR as a Volatility Anchor**:
    - **Conventional Approach**: ATR is commonly used as a fixed offset (e.g., price - 3 * ATR), applied uniformly across all conditions.
    - **This Method**: Uses ATR as a dynamic multiplier (atr_multiplier * ATR) tied to pullback events, anchoring the stop to the most recent swing point (low or high) rather than the current price.
    - **Uniqueness**: This ties the stop distance to structural price levels (e.g., recent lows in an uptrend pullback) rather than arbitrary price points, making it more context-aware and robust to volatility spikes.

Mathematical Enhancements to Make It Smarter

To elevate this logic into a mathematical advancement let's introduce advanced techniques that enhance adaptability, precision, and robustness. Here are some ideas, each with a clear rationale and potential implementation:

1. Volatility-Adjusted Thresholds (Adaptive Retracement)

- **Why?** A fixed threshold (e.g., 0.005) does not account for changing market conditions. In high-volatility regimes, a 0.5% pullback might be noise, while in low-volatility periods, it's significant.

- So we can think of scaling the threshold dynamically using ATR or Bollinger Band width:

```
df['volatility_factor'] = df['ATR'] / df['close']  # Normalize ATR by price
dynamic_threshold = threshold * (1 + df['volatility_factor'])
df['pullback_uptrend'] = (
    (df['SMA_short'] > df['SMA_long']) &
    (df['momentum'] < 0) &
    (df['highest_high'] * (1 - dynamic_threshold) > df['close'])
)
```

2. Volatility Breakout Filter (Avoid Choppy Markets)

- **Why?** Pullbacks in sideways markets can trigger false exits. A breakout filter ensures stops activate only in trending conditions.

- Use Donchian Channels or ATR breakout:

```
df['donchian_high'] = df['high'].rolling(window=20).max()
```

```
df['donchian_low'] = df['low'].rolling(window=20).min()
df['breakout'] = (df['close'] > df['donchian_high'].shift(1)) | (df['close'] < df['donchian_low'].shift(1))
df['pullback_uptrend'] = df['pullback_uptrend'] & df['breakout'].shift(1)  # Only after breakout
```

3. Hawkes Process Estimates Market Activity

- Instead of static stop-losses, it **adjusts ATR multipliers** dynamically.
- If the Hawkes intensity is **high**, tighter stops prevent excessive losses.
- If **low**, wider stops allow the trend to play out.

Better Adaptation to Market Regimes

- In calm markets, **trailing stops widen**, avoiding unnecessary stop-outs.
- In volatile phases, **stops tighten**, locking in profits earlier.

Improves Risk Management & Trade Efficiency

- Instead of relying purely on ATR, it uses **market microstructure information** (price jump clustering).
- It prevents **overfitting to historical ATR values**, making stops more **adaptive and realistic**.

Test it on your data, tweak parameters, and backtest ruthlessly to unleash its full potential

---