BLUTA Momentum Algo

BTC Perpetual Future Contract

LAWRENCE CHANG

EXECUTIVE SUMMARY

STRATEGY CONCEPT

Bluta Momentum Algo:

The box theory breakout approach to momentum trading posits that when price data consistently trades within a well-defined "box" or range—captured by recent minimum and maximum levels—an eventual breach of those boundaries signals a potential momentum-driven move. Conceptually, the strategy identifies the prevailing "box" (the highest and lowest prices over a set lookback period), then classifies a break above the upper threshold as a bullish initiation (betting on further upside), and a break below the lower threshold as a bearish indication (anticipating continued downside). This framework couples simple boundary logic with market psychology: persistent confinement and subsequent release generate a "pressure cooker" effect, so that once the price escapes its prior box, momentum—fueled by stop triggers, trend-following orders, and renewed sentiment—may carry it substantially farther, aligning with core momentum principles.

1min K-Line Plot

STATISTICAL ANALYSIS

Target Result for Momentum Trading Algo :

- Beta > 0
- P-Value < 0.05
- High R-Square

Linear Regression Model

$$\mathrm{ret}t + lag \;=\; \alpha \;+\; \beta\,\mathrm{ret}t \;+\; \epsilon_{t+lag}$$

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               ret_next   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     4251.
Date:                Fri, 17 Jan 2025   Prob (F-statistic):               0.00
Time:                        20:08:04   Log-Likelihood:             1.3764e+07
No. Observations:             2515440   AIC:                        -2.753e+07
Df Residuals:                 2515438   BIC:                        -2.753e+07
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.436e-06   6.41e-07      2.239      0.025    1.79e-07    2.69e-06
ret            0.0411      0.001     65.200      0.000       0.040       0.042
==============================================================================
Omnibus:                 11240433.514   Durbin-Watson:                    1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):   1136249419441478.250
Skew:                         145.078   Prob(JB):                          0.00
Kurtosis:                  104122.912   Cond. No.                          982.
==============================================================================
```

Improvement???

Multivariate Regression Model

$$\mathrm{ret}t + lag \;=\; \alpha \;+\; \beta_1\,\mathrm{ret}t \;+\; \beta_2\,X_t^{(1)} \;+\; \beta_3\,X_t^{(2)} \;+\; \cdots \;+\; \epsilon_{t+lag}$$

Auto-Regression Regression Model

$$\mathrm{ret}t \;=\; \alpha + \sum i = 1^p \beta_i\,\mathrm{ret}_{t-i} + \epsilon_t$$

Logistic Regression Model

$$\mathrm{Prob}(\mathrm{ret}t + 1 > 0) \;=\; \sigma(\alpha + \beta_1\,\mathrm{ret}t + \beta_2\,X_t^{(1)} + \ldots)$$

TRADING ALGO/SIGNAL

Rolling Min Return

$$\min\{x_1, \ldots, x_n\} = \text{smallest value among } x_1, \ldots, x_n.$$

Rolling Max Return

$$\max\{x_1, \ldots, x_n\} = \text{largest value among } x_1, \ldots, x_n.$$

Min-Max Indicator

$$\text{MinMaxIndicator}(t) = \frac{P_t - \min_t}{\max_t - \min_t}.$$

Rolling Close Price

$$\min_t = \min\{P_{t-w+1}, \ldots, P_t\}, \qquad \max_t = \max\{P_{t-w+1}, \ldots, P_t\}.$$

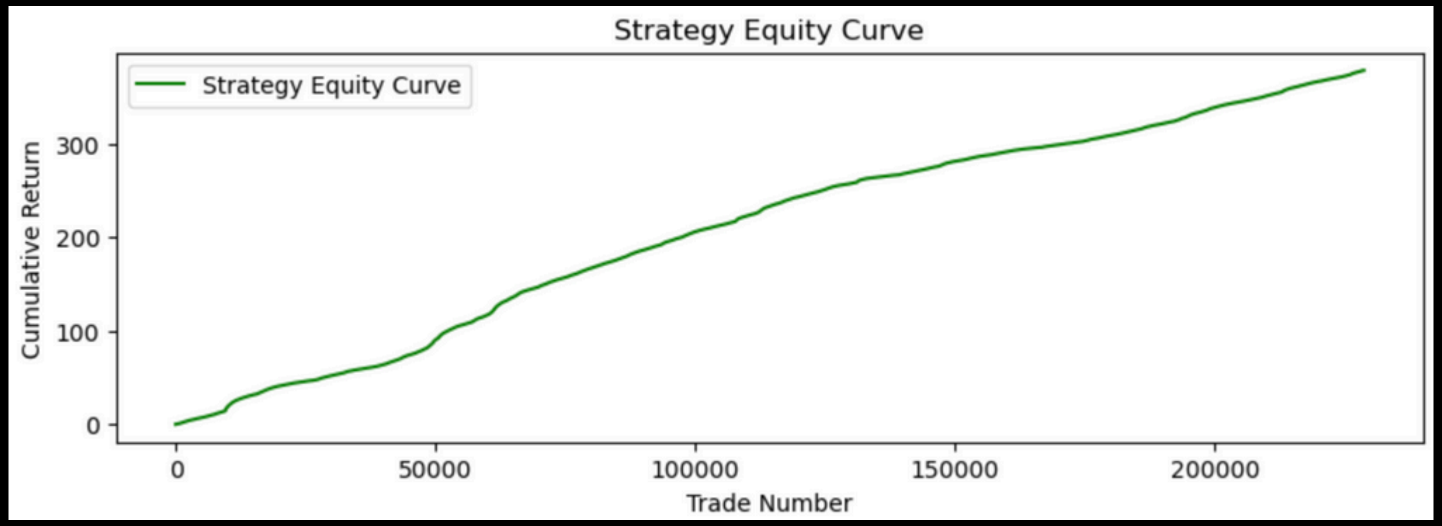
Trading Signal

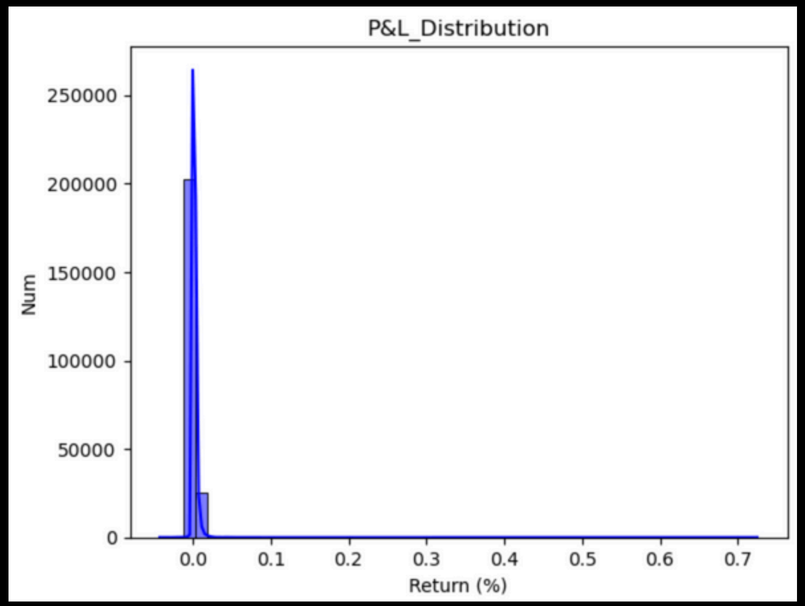
$$S(t) = \begin{cases} +1, & \text{if } I(t) \geq T_{\text{high}}, \\ -1, & \text{if } I(t) \leq T_{\text{low}}, \\ 0, & \text{otherwise.} \end{cases}$$

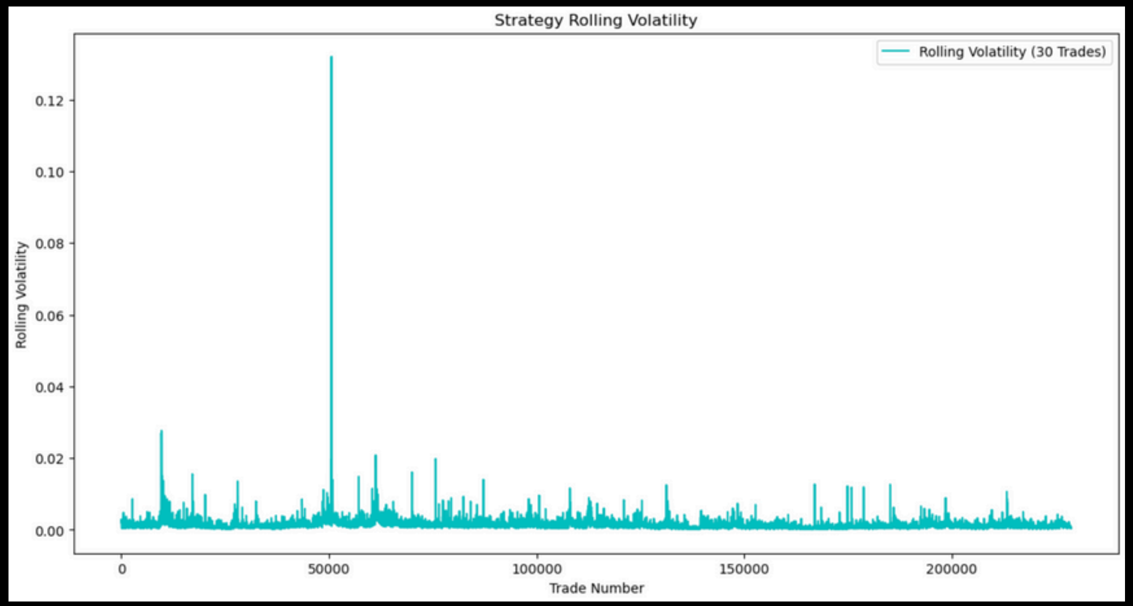
```python
# Min-Max Indicator
def calculate_min_max_indicator(btc, window):
    btc = np.array(btc)
    min_vals = pd.Series(btc).rolling(window=window).min().to_numpy()
    max_vals = pd.Series(btc).rolling(window=window).max().to_numpy()
    # DropNa
    range_vals = max_vals - min_vals
    range_vals[range_vals == 0] = np.nan

    mm = (btc - min_vals) / (max_vals - min_vals)
    return mm
```

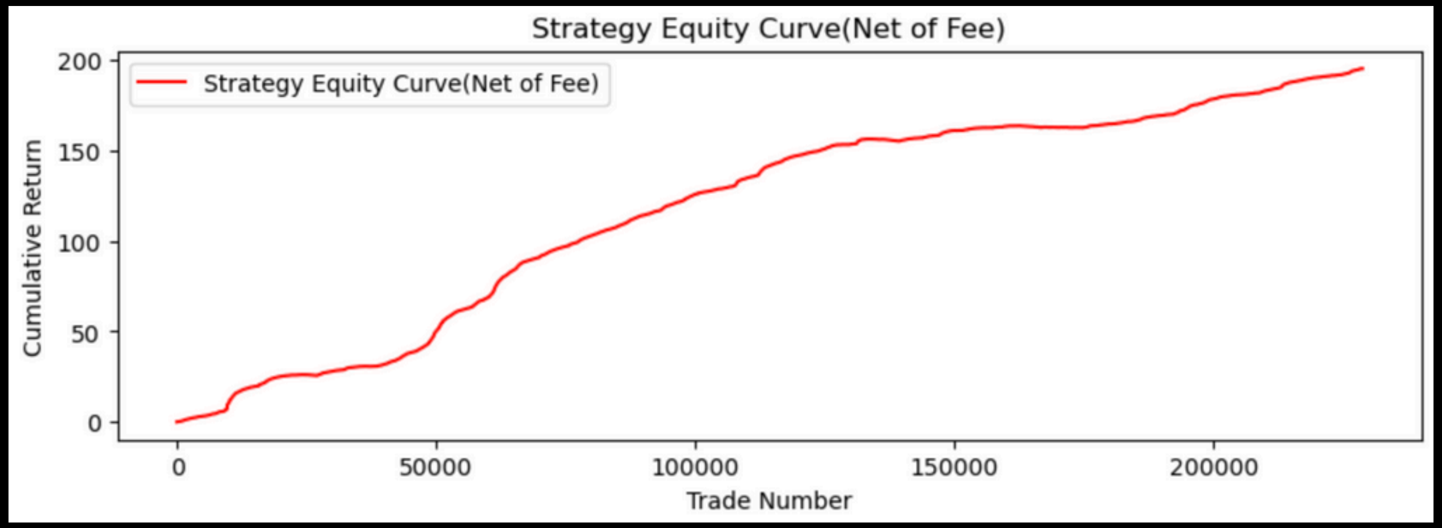
```python
# Trading Algo
def generate_signals(min_max, threshold_high=0.8, threshold_low=0.2):
    signals = np.zeros(len(min_max))
    signals[min_max > threshold_high] = 1  # Long Position
    signals[min_max < threshold_low] = -1  # Short Position
    return signals
```

BACKTEST RESULT









Strategy Summary:

- Total Ret: 37,809.78%
- Total Ret(After Fee): 19,510.66%
- Benchmark Ret(Buy & Hold): 1,175.24%
- Trades: 228739
- Avg Trade(Mean): 0.1653%
- Win Rate: 63.44%
- Max Loss: -4.2757%
- Taker Fee: 0.04%

APPENDIX

```python
def simulate_trading(btc, signals, stop_loss_factor=0.02, take_profit_factor=0.04,
                     max_holding_time=10, trading_fee=0.0004):
    returns = []
    position_open = False
    entry_price = None
    holding_time = 0
    num_trades = 0  # To keep track of the number of trades

    for i in range(len(btc)):  # Corrected from 'prices' to 'btc'
        if signals[i] != 0 and not position_open:
            position_open = True
            entry_price = btc[i]
            take_profit = entry_price * (1 + take_profit_factor * signals[i])
            stop_loss = entry_price * (1 - stop_loss_factor * signals[i])
            holding_time = 0

        elif position_open:
            holding_time += 1
            if signals[i] == 1:  # Long Position
                if btc[i] >= take_profit or btc[i] <= stop_loss or holding_time >= max_holding_time:
                    trade_return = (btc[i] - entry_price) / entry_price
                    trade_return -= 2 * trading_fee  # Subtract trading fees
                    returns.append(trade_return)
                    num_trades += 1
                    position_open = False
            elif signals[i] == -1:  # Short Position
                if btc[i] <= take_profit or btc[i] >= stop_loss or holding_time >= max_holding_time:
                    trade_return = (entry_price - btc[i]) / entry_price
                    trade_return -= 2 * trading_fee  # Subtract trading fees
                    returns.append(trade_return)
                    num_trades += 1
                    position_open = False

    return returns, num_trades
```

```python
# Back-Test Strategy
def run_strategy(
    btc, window, threshold_high=0.8, threshold_low=0.2,
    stop_loss_factor=0.02, take_profit_factor=0.04,
    max_holding_time=10, trading_fee=0.0004
):
    mm = calculate_min_max_indicator(btc, window)
    signals = generate_signals(mm, threshold_high, threshold_low)
    returns, num_trades = simulate_trading(
        btc, signals, stop_loss_factor, take_profit_factor,
        max_holding_time, trading_fee
    )
    total_return = sum(returns)
    return total_return, num_trades

# Define Parameters
window = 15
threshold_high = 0.8
threshold_low = 0.2
stop_loss_factor = 0.02
take_profit_factor = 0.04
max_holding_time = 10
trading_fee = 0.0004
btc = btc_df['close'].values

# Run the Strategy
total_return, num_trades = run_strategy(
    btc, window, threshold_high, threshold_low,
    stop_loss_factor, take_profit_factor,
    max_holding_time, trading_fee
)

print(f"Strategy's Return after Fees: {total_return:.2%}")
print(f"Number of Trades Executed: {num_trades}")
```

```python
# Calculate rolling volatility (standard deviation)
rolling_volatility = returns_series.rolling(window=rolling_window).std()

# Calculate Win Rate
winning_trades = returns_series[returns_series > 0]
losing_trades = returns_series[returns_series <= 0]
win_rate = (len(winning_trades) / len(returns_series)) * 100  # Percentage
```

IMPROVEMENT?

- Out-Sample Backtest: ETH, SOL, etc
- Statistical Analysis Model
- Money Management