Stock Pattern Detection Algorithms Documentation

This document provides a comprehensive overview of all algorithms implemented in the Stock Pattern Detector system for identifying the four main chart patterns: Head and Shoulders (HNS), Cup and Handle (CH), Double Top (DT), and Double Bottom (DB).

Table of Contents

Core Infrastructure

Data Processing Pipeline

1. **Data Loading**: Uses yfinance to fetch OHLCV data
2. **Swing Point Identification**: Multiple methods for identifying pivot points
3. **Pattern Recognition**: Specific algorithms for each pattern type
4. **Volume Confirmation**: Volume analysis for pattern validation
5. **Breakout Detection**: Confirmation of pattern completion

Key Components

- **Timeframe Support**: 1d to 5y with configurable parameters
- **Detection Modes**: Strict and lenient configurations
- **Multi-Symbol Processing**: Batch processing capabilities

- **Visualization**: Matplotlib and Plotly chart generation

Swing Point Detection

1. Rolling Window Method

```
def find_swing_points(data, N_bars=20):
    data['is_swing_high'] = (data['High'] == data['High'].rolling(window=N_bars*2+1, center=True
    data['is_swing_low'] = (data['Low'] == data['Low'].rolling(window=N_bars*2+1, center=True).r
```

- **Algorithm**: Identifies peaks/troughs as local maxima/minima within a rolling window
- **Window Size**: 2*N_bars + 1 (default: 41 periods)
- **Advantages**: Simple, reliable for most timeframes
- **Use Case**: Primary method for pattern detection

2. ZigZag Method

```
def compute_zigzag(data, pct=0.03):
    # Algorithm identifies significant price movements > pct threshold
    for i in range(1, n):
        change = (closes[i] - last_extreme_price) / last_extreme_price
        if last_type in ('unknown', 'low') and change >= pct:
            is_high[i] = True  # New high
        elif last_type in ('unknown', 'high') and change <= -pct:
            is_low[i] = True   # New low
```

- **Algorithm**: Tracks significant price reversals exceeding percentage threshold
- **Threshold**: 3% default price change requirement
- **Advantages**: Filters out noise, focuses on meaningful moves
- **Use Case**: Alternative method for volatile markets

3. Williams Fractals Method

```
def compute_fractals(data, left=2, right=2):
    for i in range(left, n-right):
        if highs[i] == max(highs[i-left:i+right+1]):
            is_high[i] = True  # Fractal high
        if lows[i] == min(lows[i-left:i+right+1]):
            is_low[i] = True   # Fractal low
```

- **Algorithm**: Identifies fractals where current bar is highest/lowest in surrounding bars
- **Window**: left + right + 1 bars (default: 5 bars)
- **Advantages**: Standard technical analysis approach
- **Use Case**: Traditional fractal-based analysis

Dynamic N_bars Calculation

```
def compute_dynamic_nbars(data, base=20, min_bars=8, max_bars=60):
    atr = compute_atr(data, period=14)
    recent_atr = atr.iloc[-1]
    median_atr = np.nanmedian(atr)
    scale = recent_atr / median_atr
    nbars = int(max(min_bars, min(max_bars, round(base * scale))))
```

- **Purpose**: Adjusts swing detection sensitivity based on market volatility
- **Method**: Uses Average True Range (ATR) to scale window size
- **Range**: 8-60 bars based on volatility conditions

Pattern Detection Algorithms

Head and Shoulders (HNS)

Algorithm Overview

The Head and Shoulders pattern consists of three peaks with the middle peak (head) being the highest, flanked by two roughly equal peaks (shoulders).

Detection Process

1. Swing Point Sequence Validation

```
# Requires: High, Low, High, Low, High sequence
if not (data['is_swing_high'][p1_idx] and data['is_swing_low'][t1_idx] and
        data['is_swing_high'][p2_idx] and data['is_swing_low'][t2_idx] and
        data['is_swing_high'][p3_idx]):
    continue
```

2. Head Dominance Check

```
# Head must be highest peak and exceed shoulders by minimum percentage
higher_shoulder = max(p1_high, p3_high)
if not (p2_high > p1_high and p2_high > p3_high):
    continue
if not (p2_high > higher_shoulder * (1 + config['HEAD_OVER_SHOULDER_PCT'])):
    continue
```

- **Strict Mode**: 7% minimum head elevation
- **Lenient Mode**: 3% minimum head elevation

3. Shoulder Symmetry Analysis

```
# Shoulders must be reasonably similar in height
if abs(p1_high - p3_high) / max(p1_high, p3_high) > config['SHOULDER_TOL']:
    continue
```

- **Strict Mode**: 10% maximum shoulder difference
- **Lenient Mode**: 15% maximum shoulder difference

4. Neckline Analysis

```
# Calculate neckline slope and angle
slope = (neckline_y[1] - neckline_y[0]) / (neckline_x[1] - neckline_x[0])
angle_deg = abs(np.degrees(np.arctan(slope)))
if angle_deg > config['MAX_NECKLINE_ANGLE_DEG']:
    continue
```

- **Purpose**: Ensures neckline isn't too steep (unrealistic pattern)

- **Strict Mode**: Maximum 25° angle
- **Lenient Mode**: Maximum 45° angle

5. Volume Pattern Validation

```
# Volume should decline from left shoulder to head to right shoulder
vol_ls = data['Volume'].iloc[p1_idx:t1_idx+1].mean()  # Left shoulder
vol_h = data['Volume'].iloc[t1_idx:t2_idx+1].mean()   # Head
vol_rs = data['Volume'].iloc[t2_idx:p3_idx+1].mean()  # Right shoulder

if not (vol_ls > vol_h * config['VOLUME_TREND_TOL'] and
        vol_h > vol_rs * config['VOLUME_TREND_TOL']):
    continue
```

- **Pattern**: Decreasing volume through pattern formation
- **Validation**: Each phase should have lower volume than previous

6. Breakout Confirmation

```
# Look for neckline break with volume spike
for j in range(p3_idx + 1, min(len(data), p3_idx + 90)):
    neckline_price_at_j = slope * date_j.toordinal() + intercept
    if close_j < neckline_price_at_j:  # Neckline break
        if vol_j > avg_volume_pattern * config['VOLUME_SPIKE_MULT']:
            breakout_confirmed = True
```

- **Requirement**: Price must break below neckline
- **Volume Confirmation**: Breakout volume must exceed pattern average by multiplier
- **Timeframe**: 90-day maximum lookout period

Configuration Parameters

Parameter	Strict Mode	Lenient Mode	Description
SHOULDER_TOL	0.10	0.15	Maximum shoulder height difference
HEAD_OVER_SHOULDER_PCT	0.07	0.03	Minimum head elevation over shoulders

Parameter	Strict Mode	Lenient Mode	Description
MAX_NECKLINE_ANGLE_DEG	25	45	Maximum neckline slope angle
VOLUME_TREND_TOL	0.95	0.85	Volume decline tolerance
VOLUME_SPIKE_MULT	1.75	1.25	Breakout volume multiplier

Cup and Handle (CH)

Algorithm Overview

The Cup and Handle pattern consists of a U-shaped "cup" followed by a smaller downward "handle" before an upward breakout.

Detection Process

1. Cup Formation Identification

```
# Find two high points that could form cup rims
for i in range(len(high_indices) - 1):
    left_rim_idx = high_indices.iloc[i]['index']
    for j in range(i + 1, len(high_indices)):
        right_rim_idx = high_indices.iloc[j]['index']
```

2. Cup Duration Validation

```
cup_duration = (right_rim_date - left_rim_date).days
if cup_duration < GLOBAL_CONFIG['CUP_DURATION_MIN'] or
   cup_duration > GLOBAL_CONFIG['CUP_DURATION_MAX']:
    continue
```

- **Minimum**: 30 days
- **Maximum**: 365 days

3. Cup Depth Analysis

```
cup_bottom_idx = cup_section['Low'].idxmin()
cup_depth = (left_rim_price - cup_bottom_low) / left_rim_price
if cup_depth < config['CUP_DEPTH_MIN'] or cup_depth > config['CUP_DEPTH_MAX']:
    continue
```

- **Strict Mode**: 15-40% depth
- **Lenient Mode**: 8-60% depth

4. Cup Symmetry (U-Shape) Enforcement

```
# Check for U-shape rather than V-shape
left_side_duration = (cup_bottom_date - left_rim_date).days
right_side_duration = (right_rim_date - cup_bottom_date).days
symmetry_ratio = min(left_side_duration, right_side_duration) / max(left_side_duration, right_s
```

```
# Enforce minimum duration for each side (no sharp V)
if left_side_duration < 0.3 * total_cup_duration or
   right_side_duration < 0.3 * total_cup_duration:
    continue
```

- **Purpose**: Ensures gradual rounding, not sharp reversal
- **Requirement**: Each side must be at least 30% of total duration

5. Rim Height Similarity

```
rim_difference = abs(left_rim_high - right_rim_high) / max(left_rim_high, right_rim_high)
if rim_difference > 0.05:
    continue
```

- **Tolerance**: Maximum 5% difference between rim heights

6. Handle Formation Detection

```
for k in range(right_rim_idx + 1, min(len(data), right_rim_idx + config['HANDLE_DURATION_MAX'])):
    handle_depth = (right_rim_high - current_low) / right_rim_high
    handle_duration = (current_date - right_rim_date).days

    # Handle constraints
    if handle_depth > config['HANDLE_DEPTH_MAX']:
        break  # Too deep
    if handle_duration > config['HANDLE_DURATION_MAX']:
        break  # Too long

    # Must not retrace more than 1/3 of cup height
    if (right_rim_high - current_low) > (cup_height / 3):
        break
```

7. Breakout Confirmation

```
# Look for breakout above right rim with volume
for m in range(k + 1, min(len(data), k + 30)):
    if data.loc[m, 'Close'] > right_rim_high:
        if breakout_volume > avg_pattern_volume * config['VOLUME_SPIKE_MULT']:
            handle_found = True
```

Configuration Parameters

Parameter	Strict Mode	Lenient Mode	Description
CUP_DEPTH_MIN	0.15	0.08	Minimum cup depth percentage
CUP_DEPTH_MAX	0.40	0.60	Maximum cup depth percentage
HANDLE_DEPTH_MAX	0.20	0.35	Maximum handle retracement
CUP_SYMMETRY_TOL	0.25	0.50	Cup symmetry tolerance
HANDLE_DURATION_MIN	7	3	Minimum handle duration (days)
HANDLE_DURATION_MAX	60	90	Maximum handle duration (days)

Parameter	Strict Mode	Lenient Mode	Description
VOLUME_DECLINE_PCT	0.85	0.70	Volume decline during formation
VOLUME_SPIKE_MULT	1.75	1.25	Breakout volume multiplier

Double Top (DT)

Algorithm Overview

Double Top pattern consists of two peaks at approximately the same level, separated by a valley, indicating potential trend reversal.

Detection Process

1. Swing Sequence Validation

```
# Requires: High, Low, High sequence
if (data.get('is_swing_high', False)[idx1] and
    data.get('is_swing_low', False)[idx_mid] and
    data.get('is_swing_high', False)[idx2]):
```

2. Temporal Spacing Check

```
spacing = (p2_date - p1_date).days
if spacing < config['MIN_SPACING_DAYS'] or spacing > config['MAX_SPACING_DAYS']:
    continue
```

- **Strict Mode**: 20-90 days between peaks
- **Lenient Mode**: 10-150 days between peaks

3. Peak Similarity Analysis

```
# Peaks must be at similar levels
if abs(p1_high - p2_high) / max(p1_high, p2_high) > config['PEAK_SIMILARITY_TOL']:
    continue
```

- **Tolerance**: Maximum 6% difference between peaks (both modes)

4. Prominence Validation

```
# Peaks must be significant relative to valley
prominence1 = (p1_high - t_low) / t_low
prominence2 = (p2_high - t_low) / t_low
if prominence1 < config['MIN_PROMINENCE_PCT'] or prominence2 < config['MIN_PROMINENCE_PCT']:
    continue
```

- **Strict Mode**: Minimum 6% prominence
- **Lenient Mode**: Minimum 3% prominence

5. Neckline Breakout Detection

```
neckline_level = t_low  # Valley low becomes neckline
for j in range(idx2 + 1, min(len(data), idx2 + 90)):
    if data['Close'].iloc[j] < neckline_level * (1 - config['NECKLINE_TOLERANCE']):
        # Check for volume confirmation
        if vol_j > avg_vol * config['VOLUME_SPIKE_MULT']:
            breakout_idx = j
```

Configuration Parameters

Parameter	Strict Mode	Lenient Mode	Description
PEAK_SIMILARITY_TOL	0.06	0.06	Maximum peak height difference
MIN_PROMINENCE_PCT	0.06	0.03	Minimum peak prominence
MAX_SPACING_DAYS	90	150	Maximum days between peaks
MIN_SPACING_DAYS	20	10	Minimum days between peaks
NECKLINE_TOLERANCE	0.015	0.03	Neckline break tolerance
VOLUME_SPIKE_MULT	1.75	1.25	Breakout volume multiplier

Double Bottom (DB)

Algorithm Overview

Double Bottom pattern consists of two troughs at approximately the same level, separated by a peak, indicating potential upward reversal.

Detection Process

1. Swing Sequence Validation

```
# Requires: Low, High, Low sequence
if (data.get('is_swing_low', False)[idx1] and
    data.get('is_swing_high', False)[idx_mid] and
    data.get('is_swing_low', False)[idx2]):
```

2. Temporal and Similarity Checks

```
# Same spacing and similarity logic as Double Top
spacing = (p2_date - p1_date).days
if spacing < config['MIN_SPACING_DAYS'] or spacing > config['MAX_SPACING_DAYS']:
    continue

# Trough similarity
if abs(p1_low - p2_low) / max(p1_low, p2_low) > config['PEAK_SIMILARITY_TOL']:
    continue
```

3. Prominence Analysis

```
# Troughs must be significant relative to peak
prominence1 = (t_high - p1_low) / t_high
prominence2 = (t_high - p2_low) / t_high
if prominence1 < config['MIN_PROMINENCE_PCT'] or prominence2 < config['MIN_PROMINENCE_PCT']:
    continue
```

4. Neckline Breakout (Upward)

```
neckline_level = t_high  # Peak high becomes neckline
for j in range(idx2 + 1, min(len(data), idx2 + 90)):
    if data['Close'].iloc[j] > neckline_level * (1 + config['NECKLINE_TOLERANCE']):
        # Volume confirmation for upward break
        if vol_j > avg_vol * config['VOLUME_SPIKE_MULT']:
            breakout_idx = j
```

Volume Analysis

Volume Pattern Requirements

Head and Shoulders

- **Pattern**: Declining volume through formation (LS > H > RS)
- **Breakout**: High volume on neckline break

Cup and Handle

- **Cup Formation**: Generally declining volume during formation
- **Handle**: Lower volume during handle formation
- **Breakout**: Volume spike on rim break

Double Patterns

- **Formation**: Can vary, but typically lower on second peak/trough
- **Breakout**: High volume confirmation required

Volume Calculation Methods

```
# Average volume during pattern
avg_volume_pattern = data.iloc[start_idx:end_idx]['Volume'].mean()

# Volume spike detection
if current_volume > avg_volume_pattern * VOLUME_SPIKE_MULT:
    volume_confirmed = True
```

Trend Analysis

Preceding Trend Check

```
def check_preceding_trend(data, pattern_start_index, trend_type='up',
                          lookback_period=90, min_change_percent=0.15):
    start_pos = max(0, pattern_start_index - lookback_period)
    lookback_data = data.iloc[start_pos: pattern_start_index]

    start_price = lookback_data['Close'].iloc[0]
    if trend_type == 'up':
        end_price = data.iloc[pattern_start_index]['High']
        return end_price > start_price * (1 + min_change_percent)
    else:  # down
        end_price = data.iloc[pattern_start_index]['Low']
        return end_price < start_price * (1 - min_change_percent)
```

Trend Requirements by Pattern

- **Head and Shoulders**: Requires preceding uptrend (15% minimum over 90 days)
- **Cup and Handle**: Requires preceding uptrend
- **Double Top**: Requires preceding uptrend
- **Double Bottom**: Requires preceding downtrend

Global Configuration

Base Parameters

```
GLOBAL_CONFIG = {
    'BASE_NBARS': 20,           # Default swing detection window
    'MIN_NBARS': 8,             # Minimum swing window
    'MAX_NBARS': 60,            # Maximum swing window
    'ZIGZAG_PCT': 0.03,         # ZigZag percentage threshold
    'MIN_TREND_PCT': 0.15,      # Minimum preceding trend
    'CUP_DURATION_MIN': 30,     # Minimum cup duration
    'CUP_DURATION_MAX': 365,    # Maximum cup duration
}
```

Detection Modes

- **Strict Mode**: Conservative parameters, higher quality patterns
- **Lenient Mode**: Relaxed parameters, more pattern detection

Algorithm Performance Characteristics

Computational Complexity

- **Swing Point Detection**: O(n) where n = data points
- **Pattern Detection**: O(s^2) where s = swing points
- **Overall**: O(n + s^2), typically O(n) for reasonable datasets

Memory Usage

- **Data Storage**: Original OHLCV + swing point flags
- **Pattern Storage**: Compact pattern metadata
- **Visualization**: Optional chart generation

Accuracy Considerations

- **False Positives**: Reduced through strict volume and trend validation
- **False Negatives**: Balanced through lenient mode options
- **Market Conditions**: Performance varies with volatility and trending

Usage Examples

Basic Pattern Detection

```
# Load and prepare data
data = load_data(symbol, start_date, end_date)
data = generate_swing_flags(data, method='rolling', N_bars=20)

# Detect patterns
hns_patterns = detect_head_and_shoulders(data, HNS_CONFIG['strict'])
ch_patterns = detect_cup_and_handle(data, CH_CONFIG['strict'])
dt_db_patterns = detect_double_patterns(data, DT_CONFIG['strict'], 'both')
```

Configuration Customization

```
# Custom configuration
custom_config = HNS_CONFIG['strict'].copy()
custom_config['SHOULDER_TOL'] = 0.08  # Stricter shoulder similarity
custom_config['VOLUME_SPIKE_MULT'] = 2.0  # Higher volume requirement

patterns = detect_head_and_shoulders(data, custom_config)
```

This documentation provides a comprehensive overview of all algorithms used in the stock pattern detection system. Each algorithm is designed with configurable parameters to balance between pattern accuracy and detection frequency, allowing users to adjust sensitivity based on their specific requirements and market conditions.