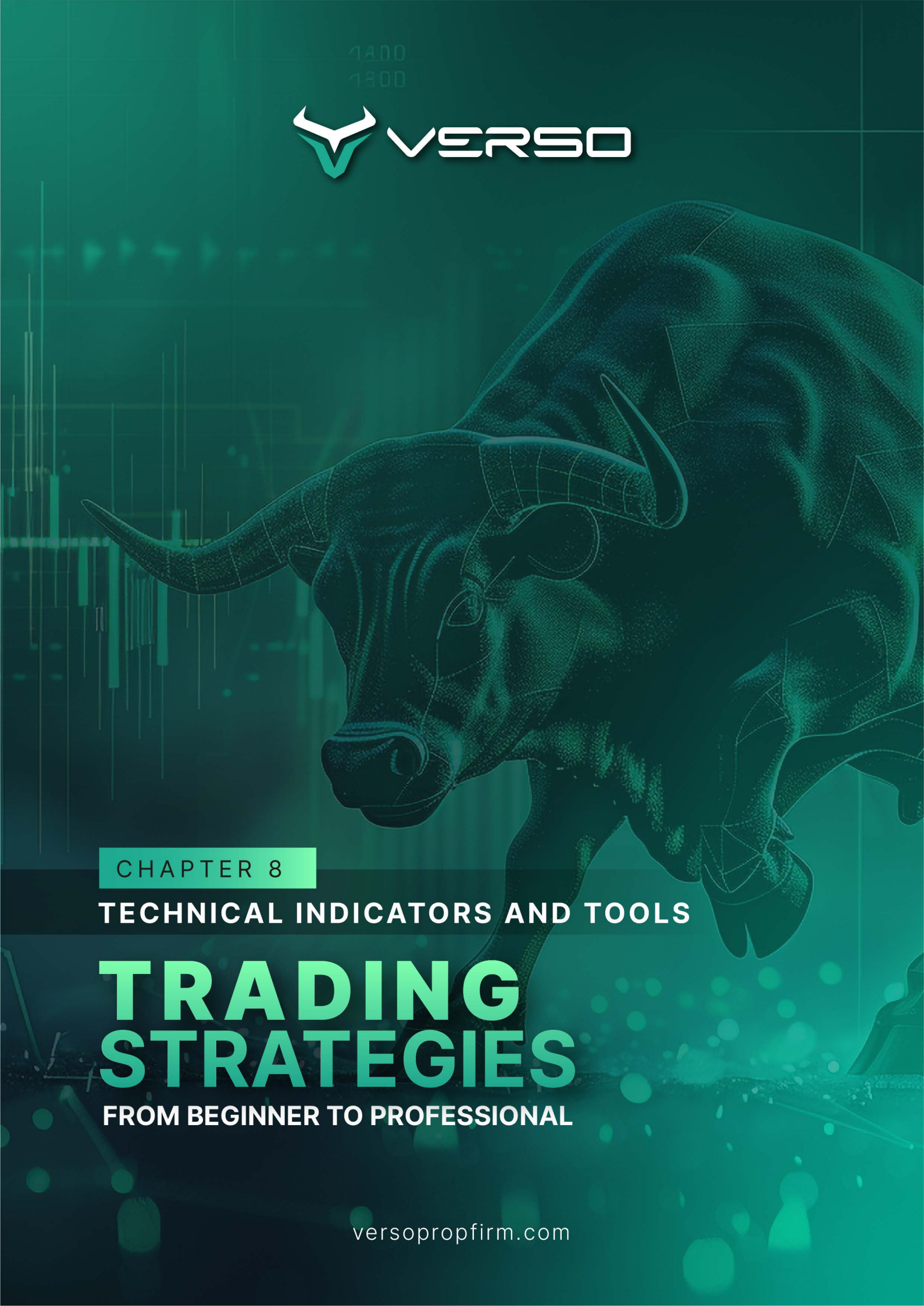

CHAPTER 8

## TECHNICAL INDICATORS AND TOOLS

# TRADING STRATEGIES

## FROM BEGINNER TO PROFESSIONAL

versopropfirm.com

# Technical Indicators and Tools

***Technical indicators are like the instrumentation of a pilot's flight: they do not replace skill and experience, but they provide crucial information to navigate in difficult conditions. The key is not to use all indicators, but to master perfectly a select set.***

**The Pyramid of Indicators**

**Level 1: Price Indicators (Foundation)**

- Pure Price Action
- Support & Resistance
- Trend Lines

**Level 2: Volume Indicators**

Volume Profile
On-Balance Volume (OBV)
Volume-Weighted Average Price (VWAP)

**Level 3: Momentum Indicators**

RSI, MACD, Stochastic
Rate of Change (ROC)
Williams %R

**Level 4: Volatility Indicators**

- Bollinger Bands
- Average True Range (ATR)
- Keltner Channels

**Level 5: Composite Indicators**

- Ichimoku Cloud
- Custom Multi-Factor Indicators

**Price Indicators:** The Base of Everything

**Price Action:** The Language of Price

Price Action is like reading the body language of the market. Each Japanese candle tells a story about the battle between buyers and sellers.

**Master Japanese Candle Patterns:**

**1. Doji Family:**

python

```
def identify_doji_patterns(ohlc_data):
    """
    Identifies doji patterns and their variations
    """
    o, h, l, c = ohlc_data['Open'], ohlc_data['High'], ohlc_data['Low'], ohlc_data['Close']
    body_size = abs(c - o)
    total_range = h - l
    upper_shadow = h - max(o, c)
    lower_shadow = min(o, c) - l

    # Criteria for different types of doji
    is_doji = body_size <= (total_range * 0.1)
    is_dragonfly = is_doji and (lower_shadow >= total_range * 0.6) and (upper_shadow
<= total_range * 0.1)
    is_gravestone = is_doji and (upper_shadow >= total_range * 0.6) and
(lower_shadow <= total_range * 0.1)
    is_long_legged = is_doji and (upper_shadow >= total_range * 0.3) and
(lower_shadow >= total_range * 0.3)

    return {
        'doji': is_doji,
        'dragonfly_doji': is_dragonfly,
        'gravestone_doji': is_gravestone,
        'long_legged_doji': is_long_legged
    }
```

2. **Engulfing Patterns:** The engulfing candles are like a change of guard in the market: the new dominant force literally "devours" the previous one.

python

```
def bullish_engulfing(prev_candle, current_candle):
    """
    Identifies bullish engulfing pattern
    """
    # Previous candle must be bearish
    prev_bearish = prev_candle['Close'] < prev_candle['Open']

    # Current candle must be bullish
    current_bullish = current_candle['Close'] > current_candle['Open']

    # Current candle must completely engulf the previous one
    engulfs_body = (current_candle['Open'] < prev_candle['Close'] and
                    current_candle['Close'] > prev_candle['Open'])

    # Volume must be higher than the average
    volume_confirmation = current_candle['Volume'] > current_candle['Volume_MA_20']

    return prev_bearish and current_bullish and engulfs_body and volume_confirmation
```

python

```
class SupportResistanceDetector:
    def __init__(self, lookback_period=50, min_touches=3, zone_threshold=0.002):
        self.lookback_period = lookback_period
        self.min_touches = min_touches
        self.zone_threshold = zone_threshold

    def find_pivot_points(self, price_data):
        """
        Finds significant pivot points in the price
        """
        highs = price_data['High']
        lows = price_data['Low']

        # High pivots (resistance candidates)
        pivot_highs = []
        for i in range(2, len(highs) - 2):
            if (highs[i] > highs[i-1] and highs[i] > highs[i-2] and
                highs[i] > highs[i+1] and highs[i] > highs[i+2]):
```

```
                pivot_highs.append({'price': highs[i], 'index': i})

        # Low pivots (support candidates)
        pivot_lows = []
        for i in range(2, len(lows) - 2):
            if (lows[i] < lows[i-1] and lows[i] < lows[i-2] and
                lows[i] < lows[i+1] and lows[i] < lows[i+2]):
                pivot_lows.append({'price': lows[i], 'index': i})
        return pivot_highs, pivot_lows

    def cluster_levels(self, pivots):
        """
        Groups nearby pivots into support/resistance zones
        """
        if not pivots:
            return []
        clusters = []
        for pivot in pivots:
            price = pivot['price']

            # Look for existing cluster
            assigned = False
            for cluster in clusters:
                if abs(price - cluster['avg_price']) / cluster['avg_price'] <=
self.zone_threshold:
                    cluster['prices'].append(price)
                    cluster['touches'] += 1
                    cluster['avg_price'] = sum(cluster['prices']) / len(cluster['prices'])
                    assigned = True
                    break

            # Create new cluster if not assigned
            if not assigned:
                clusters.append({
                    'prices': [price],
                    'touches': 1,
                    'avg_price': price,
                    'strength': 1
                })

        # Filter clusters with sufficient touches
```

```
        significant_levels = [cluster for cluster in clusters if cluster['touches'] >=
self.min_touches]
        return sorted(significant_levels, key=lambda x: x['touches'], reverse=True)
```

**Volume Indicators: The Fuel of Movement**

**Volume Profile: The Digital Footprint of the Market**

The Volume Profile shows us where traders really put their money, revealing the most important levels of the market.

```
python
class VolumeProfileAnalyzer:
    def __init__(self, price_levels=100):
        self.price_levels = price_levels

    def calculate_volume_profile(self, ohlc_data):
        """
        Calculates the volume profile to identify POC and Value Area
        """
        high_price = ohlc_data['High'].max()
        low_price = ohlc_data['Low'].min()
        price_range = high_price - low_price
        tick_size = price_range / self.price_levels

        # Create price bins
        price_bins = {}
        for i in range(self.price_levels):
            price_level = low_price + (i * tick_size)
            price_bins[round(price_level, 4)] = 0

        # Distribute volume into bins
        for idx, row in ohlc_data.iterrows():
            candle_range = row['High'] - row['Low']
            if candle_range == 0:
                continue

            # Distribute volume proportionally across the candle's range
            for price_level in price_bins.keys():
```

```
            if row['Low'] <= price_level <= row['High']:
                weight = 1 - abs(price_level - ((row['High'] + row['Low']) / 2)) /
(candle_range / 2)
                price_bins[price_level] += row['Volume'] * weight

    # Find POC (Point of Control)
    poc_price = max(price_bins, key=price_bins.get)
    poc_volume = price_bins[poc_price]

    # Calculate Value Area (70% of total volume)
    total_volume = sum(price_bins.values())
    target_volume = total_volume * 0.70

    # Expand from POC until reaching 70% of volume
    sorted_levels = sorted(price_bins.items(), key=lambda x: x[1], reverse=True)
    value_area_volume = poc_volume
    value_area_levels = [poc_price]
    for price, volume in sorted_levels[1:]:
        if value_area_volume >= target_volume:
            break
        value_area_levels.append(price)
        value_area_volume += volume

    vah = max(value_area_levels)  # Value Area High
    val = min(value_area_levels)  # Value Area Low
    return {
        'poc': poc_price,
        'vah': vah,
        'val': val,
        'volume_profile': price_bins
          }
```

**VWAP: The Fair Price of the Day** The VWAP (Volume Weighted Average Price) is like the "fair price" of the day, calculated considering the volume in each transaction.

python

```
def calculate_vwap_bands(ohlc_data, num_std=2):
    """
    Calculates VWAP and its standard deviation bands
    """
    # Basic VWAP
```

```
    typical_price = (ohlc_data['High'] + ohlc_data['Low'] + ohlc_data['Close']) / 3
    cumulative_tpv = (typical_price * ohlc_data['Volume']).cumsum()
    cumulative_volume = ohlc_data['Volume'].cumsum()
    vwap = cumulative_tpv / cumulative_volume

    # Volume-weighted variance
    squared_diff = (typical_price - vwap) ** 2
    cumulative_squared_diff = (squared_diff * ohlc_data['Volume']).cumsum()
    vwap_variance = cumulative_squared_diff / cumulative_volume
    vwap_std = np.sqrt(vwap_variance)

    # Bands
    upper_band_1 = vwap + (vwap_std * 1)
    upper_band_2 = vwap + (vwap_std * num_std)
    lower_band_1 = vwap - (vwap_std * 1)
    lower_band_2 = vwap - (vwap_std * num_std)
    return {
        'vwap': vwap,
        'upper_1': upper_band_1,
        'upper_2': upper_band_2,
        'lower_1': lower_band_1,
        'lower_2': lower_band_2,
        'std_dev': vwap_std
    }
```

**Advanced RSI: Beyond 70/30**

The traditional RSI is just the beginning. Advanced RSI includes divergences, failure swing patterns, and range analysis.

python

```
class AdvancedRSI:
    def __init__(self, period=14, overbought=70, oversold=30):
        self.period = period
        self.overbought = overbought
        self.oversold = oversold
```

```
def calculate_rsi(self, prices):
    """
    Calculates traditional RSI
    """
    deltas = prices.diff()
    gains = deltas.where(deltas > 0, 0)
    losses = -deltas.where(deltas < 0, 0)
    avg_gains = gains.rolling(window=self.period).mean()
    avg_losses = losses.rolling(window=self.period).mean()
    rs = avg_gains / avg_losses
    rsi = 100 - (100 / (1 + rs))
    return rsi

def detect_divergences(self, prices, rsi_values, lookback=20):
    """
    Detects divergences between price and RSI
    """
    divergences = []
    for i in range(lookback, len(prices) - lookback):
        # Look for peaks and troughs in price
        price_peak = (prices[i] > prices[i-1] and prices[i] > prices[i+1])
        price_trough = (prices[i] < prices[i-1] and prices[i] < prices[i+1])

        if price_peak:
            # Look for previous peak to compare
            for j in range(i - lookback, i - 2):
                if prices[j] > prices[j-1] and prices[j] > prices[j+1]:
                    # Compare price peaks vs RSI
                    if prices[i] > prices[j] and rsi_values[i] < rsi_values[j]:
                        divergences.append({
                            'type': 'bearish',
                            'price_indices': (j, i),
                            'strength': abs(rsi_values[i] - rsi_values[j])
                        })
                        break
```

```
            if price_trough:
                # Similar for troughs (bullish divergence)
                for j in range(i - lookback, i - 2):
                    if prices[j] < prices[j-1] and prices[j] < prices[j+1]:
                        if prices[i] < prices[j] and rsi_values[i] > rsi_values[j]:
                            divergences.append({
                                'type': 'bullish',
                                'price_indices': (j, i),
                                'strength': abs(rsi_values[i] - rsi_values[j])
                            })
                            break
        return divergences

    def failure_swings(self, rsi_values):
        """
        Identifies failure swings - powerful reversal signals
        """
        swings = []
        for i in range(4, len(rsi_values)):
            # Bullish failure swing: RSI makes a higher low than previous < 30
            if (rsi_values[i-2] < 30 and
                rsi_values[i] < 30 and
                rsi_values[i] > rsi_values[i-2] and
                rsi_values[i-1] > rsi_values[i] and
                rsi_values[i-1] > rsi_values[i-3]):
                swings.append({
                    'type': 'bullish_failure_swing',
                    'index': i,
                    'strength': 30 - min(rsi_values[i-2], rsi_values[i])
                })
            # Bearish failure swing: RSI makes a lower high than previous > 70
            elif (rsi_values[i-2] > 70 and
                  rsi_values[i] > 70 and
                  rsi_values[i] < rsi_values[i-2] and
                  rsi_values[i-1] < rsi_values[i] and
                  rsi_values[i-1] < rsi_values[i-3]):
                swings.append({
```

```
                'type': 'bearish_failure_swing',
                'index': i,
                'strength': min(rsi_values[i-2], rsi_values[i]) - 70
            })
    return swings
```

**MACD: The Ultimate Momentum Indicator**

python

```
class MACDAnalyzer:
    def __init__(self, fast_period=12, slow_period=26, signal_period=9):
        self.fast_period = fast_period
        self.slow_period = slow_period
        self.signal_period = signal_period

    def calculate_macd(self, prices):
        """
        Calculates MACD line, Signal line, and Histogram
        """
        # EMAs
        ema_fast = prices.ewm(span=self.fast_period).mean()
        ema_slow = prices.ewm(span=self.slow_period).mean()
        # MACD Line
        macd_line = ema_fast - ema_slow
        # Signal Line
        signal_line = macd_line.ewm(span=self.signal_period).mean()
        # Histogram
        histogram = macd_line - signal_line
        return {
            'macd': macd_line,
            'signal': signal_line,
            'histogram': histogram
        }

    def detect_macd_signals(self, macd_data):
        """
```

```
    Detects traditional and advanced MACD signals
    """
    macd = macd_data['macd']
    signal = macd_data['signal']
    histogram = macd_data['histogram']
    signals = []
    for i in range(1, len(macd)):
        # Bullish crossover
        if macd[i] > signal[i] and macd[i-1] <= signal[i-1]:
            signals.append({
                'type': 'bullish_crossover',
                'index': i,
                'strength': abs(macd[i] - signal[i])
            })
        # Bearish crossover
        elif macd[i] < signal[i] and macd[i-1] >= signal[i-1]:
            signals.append({
                'type': 'bearish_crossover',
                'index': i,
                'strength': abs(macd[i] - signal[i])
            })
        # Zero line cross
        elif macd[i] > 0 and macd[i-1] <= 0:
            signals.append({
                'type': 'bullish_zero_cross',
                'index': i,
                'strength': macd[i]
            })
        elif macd[i] < 0 and macd[i-1] >= 0:
            signals.append({
                'type': 'bearish_zero_cross',
                'index': i,
                'strength': abs(macd[i])
            })
        # Histogram divergence (early warning)
        if i >= 3:
            if (histogram[i] > histogram[i-1] > histogram[i-2] and
```

```
            histogram[i-2] < 0):
            signals.append({
                'type': 'bullish_histogram_divergence',
                'index': i,
                'strength': abs(histogram[i-2])
            })
    return signals
```

**Volatility Indicators: Measuring Uncertainty Advanced Bollinger Bands**

python

```
class BollingerBandsAdvanced:
    def __init__(self, period=20, std_dev=2.0):
        self.period = period
        self.std_dev = std_dev

    def calculate_bands(self, prices):
        """
        Calculates traditional Bollinger Bands
        """
        sma = prices.rolling(window=self.period).mean()
        std = prices.rolling(window=self.period).std()
        upper_band = sma + (std * self.std_dev)
        lower_band = sma - (std * self.std_dev)
        return {
            'sma': sma,
            'upper': upper_band,
            'lower': lower_band,
            'width': upper_band - lower_band,
            'percent_b': (prices - lower_band) / (upper_band - lower_band)
        }

    def squeeze_detection(self, bb_data, lookback=20):
        """
        Detects Bollinger Band Squeeze - low volatility before breakouts
        """
```

```
    width = bb_data['width']
    squeeze_signals = []
    for i in range(lookback, len(width)):
        current_width = width[i]
        avg_width = width[i-lookback:i].mean()
        min_width = width[i-lookback:i].min()

        # Squeeze: current width is the minimum of the period
        if current_width == min_width and current_width < avg_width * 0.7:
            squeeze_signals.append({
                'index': i,
                'width': current_width,
                'avg_width': avg_width,
                'intensity': avg_width / current_width
            })
    return squeeze_signals

def bb_percent_analysis(self, bb_data):
    """
    Analyzes %B to identify extreme conditions
    """
    percent_b = bb_data['percent_b']
    analysis = []
    for i in range(1, len(percent_b)):
        current_b = percent_b[i]
        prev_b = percent_b[i-1]

        # Walking the bands - trending conditions
        if current_b > 0.8 and prev_b > 0.8:
            analysis.append({
                'type': 'walking_upper_band',
                'index': i,
                'strength': current_b
            })
        elif current_b < 0.2 and prev_b < 0.2:
            analysis.append({
                'type': 'walking_lower_band',
```

```
                'index': i,
                'strength': 1 - current_b
            })

        # Reversal signals
        elif current_b > 1.0 and prev_b <= 1.0:
            analysis.append({
                'type': 'breakout_upper',
                'index': i,
                'strength': current_b - 1.0
            })
        elif current_b < 0.0 and prev_b >= 0.0:
            analysis.append({
                'type': 'breakdown_lower',
                'index': i,
                'strength': abs(current_b)
            })
    return analysis
```

**Advanced Trading Tools Market Microstructure Analysis**

```
python
class MarketMicrostructure:
    def __init__(self):
        pass

    def calculate_bid_ask_spread(self, level1_data):
        """
        Analyzes bid-ask spreads to assess liquidity
        """
        spreads = level1_data['ask'] - level1_data['bid']
        relative_spreads = spreads / level1_data['mid_price']
        return {
            'absolute_spread': spreads,
            'relative_spread': relative_spreads,
```

```
            'avg_spread': spreads.mean(),
            'spread_volatility': spreads.std()
        }

    def order_flow_imbalance(self, level2_data):
        """
        Calculates order flow imbalance
        """
        bid_volume = level2_data['bid_volume'].sum()
        ask_volume = level2_data['ask_volume'].sum()
        total_volume = bid_volume + ask_volume
        imbalance = (bid_volume - ask_volume) / total_volume if total_volume > 0 else 0
        return {
            'imbalance': imbalance,
            'bid_volume': bid_volume,
            'ask_volume': ask_volume,
            'interpretation': 'bullish' if imbalance > 0.1 else 'bearish' if imbalance < -0.1 else
'neutral'
        }

    def tick_by_tick_analysis(self, tick_data):
        """
        Analyzes tick-by-tick data to detect institutional activity
        """
        # Classify ticks as buyer/seller initiated
        tick_data['trade_type'] = 'unknown'
        for i in range(len(tick_data)):
            price = tick_data.iloc[i]['price']
            bid = tick_data.iloc[i]['bid']
            ask = tick_data.iloc[i]['ask']
            if price >= ask:
                tick_data.iloc[i, tick_data.columns.get_loc('trade_type')] = 'buyer_initiated'
            elif price <= bid:
                tick_data.iloc[i, tick_data.columns.get_loc('trade_type')] = 'seller_initiated'
            else:
                tick_data.iloc[i, tick_data.columns.get_loc('trade_type')] = 'inside_spread'
```

```
        # Aggregate analysis
        buyer_volume = tick_data[tick_data['trade_type'] ==
'buyer_initiated']['volume'].sum()
        seller_volume = tick_data[tick_data['trade_type'] ==
'seller_initiated']['volume'].sum()
        net_flow = buyer_volume - seller_volume
        return {
            'buyer_volume': buyer_volume,
            'seller_volume': seller_volume,
            'net_flow': net_flow,
            'flow_ratio': buyer_volume / seller_volume if seller_volume > 0 else float('inf')
        }
```

**Sentiment Analysis Tools**

python

```
class MarketSentimentAnalyzer:
    def __init__(self):
        self.vix_levels = {
            'low': 15,
            'normal': 25,
            'high': 35,
            'extreme': 50
        }

    def vix_sentiment(self, vix_value):
        """
        Interprets VIX levels for sentiment
        """
        if vix_value < self.vix_levels['low']:
            return {
                'sentiment': 'extremely_bullish',
                'signal': 'caution_complacency',
                'contrarian_view': 'potential_top'
            }
        elif vix_value < self.vix_levels['normal']:
```

```
            return {
                'sentiment': 'bullish',
                'signal': 'low_fear',
                'contrarian_view': 'neutral'
            }
        elif vix_value < self.vix_levels['high']:
            return {
                'sentiment': 'neutral_bearish',
                'signal': 'elevated_concern',
                'contrarian_view': 'neutral'
            }
        elif vix_value < self.vix_levels['extreme']:
            return {
                'sentiment': 'fearful',
                'signal': 'high_fear',
                'contrarian_view': 'buying_opportunity'
            }
        else:
            return {
                'sentiment': 'panic',
                'signal': 'extreme_fear',
                'contrarian_view': 'strong_buy_contrarian'
            }

    def put_call_ratio(self, options_data):
        """
        Calculates and interprets Put/Call ratio
        """
        put_volume = options_data['put_volume'].sum()
        call_volume = options_data['call_volume'].sum()
        pc_ratio = put_volume / call_volume if call_volume > 0 else float('inf')

        # Contrarian interpretation
        if pc_ratio > 1.2:
            sentiment = 'extremely_bearish'
            contrarian_signal = 'bullish_contrarian'
        elif pc_ratio > 0.9:
```

```
    sentiment = 'bearish'
    contrarian_signal = 'neutral_bullish'
elif pc_ratio > 0.7:
    sentiment = 'neutral'
    contrarian_signal = 'neutral'
else:
    sentiment = 'bullish'
    contrarian_signal = 'bearish_contrarian'
return {
    'pc_ratio': pc_ratio,
    'sentiment': sentiment,
    'contrarian_signal': contrarian_signal
}
```



versopropfirm.com