

Algo Strategy Pack Guide

This guide provides an overview of each trading strategy included in the package, with descriptions, parameters, and usage examples.

Table of Contents

1. [AdaptiveKalmanFilterStrategy](#) [\(adaptive_kalman_filter_strategy.py\)](#)
2. [AdaptiveMAVolatilityStrategy](#) [\(adaptive_ma_volatility_strategy.py\)](#)
3. [ADXADXR Bollinger Bands Strategy](#) [\(adx_adxrbollinger_bands_pctstrategy.py\)](#)
4. [ADXA OBollinger Bands Strategy](#) [\(adx_aobollinger_bands_pctstrategy.py\)](#)
5. [ADXTrendStrengthWithFilters](#) [\(adx_trend_filter.py\)](#)
6. [BB Mean Reversion](#) [\(bb_mean_reversion.py\)](#)
7. [Bollinger RSI Reversion](#) [\(bollinger_rsi_reversion.py\)](#)
8. [Decision Tree EMA Crossover ML Strategy](#) [\(decision_tree_ema_crossover_ml_strategy.py\)](#)
9. [Dema Cross Strategy](#) [\(dema_cross_strategy.py\)](#)
10. [Dual MA Cross ADX Filter](#) [\(dual_ma_cross_adx_filter.py\)](#)
11. [Dual Regime Weekly Strategy](#) [\(dual_regime_weekly_strategy.py\)](#)
12. [Dynamic Exit Trix Strategy](#) [\(dynamic_exit_trix_strategy.py\)](#)
13. [EMA Cross Strategy](#) [\(ema_cross_strategy.py\)](#)
14. [EMA Cross With SMA Filter Strategy](#) [\(ema_cross_with_sma_filter_strategy.py\)](#)
15. [EMA MACD ADX Strategy](#) [\(ema_macd_adx_strategy.py\)](#)
16. [Enhanced BB Mean Reversion](#) [\(enhanced_bb_mean_reversion.py\)](#)
17. [Enhanced Trix Strategy](#) [\(enhanced_trix_strategy.py\)](#)
18. [Ensemble Strategy With Fixed Weights](#) [\(ensemble_strategy_with_fixed_weights.py\)](#)
19. [Guppy MMA Strategy](#) [\(guppy_mma_strategy.py\)](#)
20. [Hilbert RSI Strategy](#) [\(hilbert_rsi_strategy.py\)](#)
21. [Hilbert Trend Strategy](#) [\(hilbert_trend_strategy.py\)](#)
22. [HMM Trend Regime Strategy](#) [\(hmm_trend_regime_strategy.py\)](#)
23. [Hurst Filtered Trend Strategy](#) [\(hurst_filtered_trend_strategy.py\)](#)
24. [Isolation Forest Strategy](#) [\(isolation_forest_strategy.py\)](#)
25. [Keltner ADX Strategy](#) [\(keltner_adx_strategy.py\)](#)
26. [MA Bounce Strategy](#) [\(ma_bounce_strategy.py\)](#)
27. [MA Distance Strategy](#) [\(ma_distance_strategy.py\)](#)
28. [MA Ribbon Pullback Strategy](#) [\(ma_ribbon_pullback_strategy.py\)](#)
29. [MA Slope Strategy](#) [\(ma_slope_strategy.py\)](#)

- 30. [MACDStrategy](#) [\(macd_strategy.py\)](#)
 - 31. [MovingAverageChannelStrategy](#) [\(moving_average_channel_strategy.py\)](#)
 - 32. [RsiMeanReversionFiltered](#) [\(rsi_mean_reversion_filtered.py\)](#)
 - 33. [TrendFilterEmaCrossStrategy](#) [\(trend_filter_ema_cross_strategy.py\)](#)
 - 34. [TripleEmaCrossStrategy](#) [\(triple_ema_cross_strategy.py\)](#)
-

1. AdaptiveKalmanFilterStrategy (adaptive_kalman_filter_strategy.py)

This strategy implements a trading system based on an Adaptive Kalman Filter. The filter estimates the underlying price trend (level) and its velocity. Crucially, it adapts its process noise (Q) and measurement noise (R) parameters based on market volatility, typically calculated using the standard deviation of log returns. Trades are often generated based on the estimated velocity of the price: buying when velocity turns positive and selling/shorting when velocity turns negative. This strategy aims to be responsive to changing market conditions.

Note: This strategy utilizes a custom `LogReturns` indicator (defined within the same file or a utility file) to calculate log returns for the volatility measurement.

Parameters

- `'vol_period', 20` : Lookback period for volatility calculation (e.g., standard deviation of log returns).
- `'delta', 1e-4` : A small number used in initializing or adapting the process noise covariance matrix Q, particularly for its diagonal elements.
- `'R_base', 0.1` : Base value for the measurement noise variance (R).
- `'R_scale', 1.0` : Scaling factor for adapting R based on market volatility.
- `'Q_scale_factor', 0.5` : Scaling factor for adapting Q based on market volatility.
- `'initial_cov', 1.0` : Initial covariance estimate for the state covariance matrix P.
- `'printlog', False` : Boolean to enable/disable printing of trade logs and other messages.

Usage Example

```
import backtrader as bt
# Assuming LogReturns indicator is defined in the same file or imported
from strategy_package_no_numbers.adaptive_kalman_filter_strategy import
AdaptiveKalmanFilterStrategy, LogReturns

cerebro = bt.Cerebro()
```

```
cerebro.addstrategy(AdaptiveKalmanFilterStrategy,
                    vol_period=20, R_base=0.05, printlog=True)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

2. AdaptiveMAVolatilityStrategy (adaptive_ma_volatility_strategy.py)

Strategy using KAMA (Kaufman's Adaptive Moving Average) with volatility bands and a trailing stop-loss. It trades pullbacks to KAMA or breakouts from volatility bands, exiting on a KAMA cross in the opposite direction or when a trailing stop-loss is hit. The adaptive nature of KAMA allows it to respond to changing market volatility.

Parameters

- 'kama_period', 20 : Period for KAMA calculation.
- 'vol_band_period', 20 : Period for volatility calculation (e.g., ATR) used for bands.
- 'vol_band_multiplier', 2.0 : Multiplier for the volatility bands.
- 'trailing_stop_atr_period', 14 : ATR period for the trailing stop-loss.
- 'trailing_stop_atr_multiplier', 3.0 : ATR multiplier for the trailing stop-loss.
- 'enable_shorting', True : Whether to allow short trades.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.adaptive_ma_volatility_strategy import
AdaptiveMAVolatilityStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(AdaptiveMAVolatilityStrategy, kama_period=25,
vol_band_multiplier=2.5)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

3. ADXADXR Bollinger Bands Strategy (adx_adxrbollinger_bands_pctstrategy.py)

This strategy likely combines ADX (Average Directional Index) and ADXR (Average Directional Index Rating) for trend strength assessment with Bollinger Bands (potentially using the %B indicator) for entry and exit signals. Trades might be taken when price interacts with the Bollinger Bands, confirmed by sufficient trend strength indicated by ADX/ADXR. The "pct" in the filename might refer to the Bollinger Bands %B indicator or percentage-based stops/targets.

(The original class name was generic "Strategy". A more descriptive name like ADXADXR Bollinger Bands Strategy is assumed here.)

Parameters

(Specific parameters are not provided in the snippet. Common parameters would include:)

- 'adx_period', 14 : Period for ADX and ADXR calculation.
- 'adx_threshold', 20 : ADX level to confirm trend strength.
- 'bb_period', 20 : Period for Bollinger Bands.
- 'bb_devfactor', 2.0 : Standard deviation factor for Bollinger Bands.
- 'pctb_lookback', 1 : (If %B is used) Lookback for %B signals.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the class name inside the file is 'Strategy' as per original
# snippet
from strategy_package_no_numbers.adx_adxr__bollinger_bands_pct__strategy
import Strategy as ADXADXR Bollinger Bands Strategy

cerebro = bt.Cerebro()
cerebro.addstrategy(ADXADXR Bollinger Bands Strategy, adx_period=14,
bb_period=20)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

4. ADXA0BollingerBandsStrategy (adx_aobollinger_bands_pctstrategy.py)

This strategy likely uses ADX for trend strength, the Awesome Oscillator (AO) for momentum, and Bollinger Bands (possibly %B) for entry/exit points or volatility assessment. Entries could be triggered by Bollinger Band interactions when ADX confirms a trend and AO indicates momentum in the direction of the trade.

(The original class name was generic "Strategy". A more descriptive name like ADXA0BollingerBandsStrategy is assumed here.)

Parameters

(Specific parameters are not provided in the snippet. Common parameters would include:)

- 'adx_period', 14 : Period for ADX.
- 'adx_threshold', 20 : ADX level for trend confirmation.
- 'ao_short_period', 5 : Short period for Awesome Oscillator.
- 'ao_long_period', 34 : Long period for Awesome Oscillator.
- 'bb_period', 20 : Period for Bollinger Bands.
- 'bb_devfactor', 2.0 : Standard deviation factor for Bollinger Bands.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the class name inside the file is 'Strategy' as per original
# snippet
from strategy_package_no_numbers.adx_ao__bollinger_bands_pct__strategy import
Strategy as ADXA0BollingerBandsStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(ADXA0BollingerBandsStrategy, adx_threshold=25)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

5. ADXTrendStrengthWithFilters (adx_trend_filter.py)

This strategy focuses on utilizing the ADX (Average Directional Index) to gauge trend strength. It likely acts as a filter for entries generated by other conditions (e.g., price patterns, other indicators not explicitly named in the class name) or generates signals directly when ADX crosses certain thresholds, possibly in conjunction with DI+ / DI- crossovers. The "WithFilters" suggests additional criteria are applied.

Parameters

- 'adx_period', 14 : The lookback period for calculating ADX, DI+ and DI-.
- 'adx_threshold', 20 : ADX level above which a strong trend is considered present.
- 'di_threshold', 0 : (Potentially) A threshold for DI+ / DI- difference or levels.
- 'filter_period_1', 50 : (Example) Period for an additional filter (e.g., a long-term MA).
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.adx_trend_filter import
ADXTrendStrengthWithFilters

cerebro = bt.Cerebro()
cerebro.addstrategy(ADXTrendStrengthWithFilters, adx_period=14,
adx_threshold=25)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

6. BBMeanReversion (bb_mean_reversion.py)

Implements a Bollinger Bands Mean Reversion strategy. This strategy typically enters long when the price touches or crosses below the lower Bollinger Band (expecting a reversion towards the mean/middle band) and enters short when the price touches or crosses above the upper Bollinger Band. Exits often occur when the price reaches the middle band or the opposite band.

Parameters

- 'bb_period', 20 : The lookback period for calculating the Bollinger Bands (SMA and standard deviation).

- 'bb_devfactor', 2.0 : The number of standard deviations for the upper and lower bands.
- 'exit_on_middle', True : Whether to exit trades when price crosses the middle band.
- 'enable_shorting', True : Whether to allow short trades.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.bb_mean_reversion import BBMeanReversion

cerebro = bt.Cerebro()
cerebro.addstrategy(BBMeanReversion, bb_period=20, bb_devfactor=2.0)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

7. BollingerRSIReversion (bollinger_rsi_reversion.py)

This is a mean reversion strategy that combines Bollinger Bands and the Relative Strength Index (RSI).

It enters long positions when the price touches or crosses below the lower Bollinger Band and the RSI indicates an oversold condition.

Conversely, it enters short positions when the price touches or crosses above the upper Bollinger Band and the RSI indicates an overbought condition.

Exits are typically triggered when the price crosses back over the middle Bollinger Band. An ATR-based stop loss is also mentioned, adding a risk management layer.

Parameters

- 'bb_period', 20 : Period for Bollinger Bands calculation.
- 'bb_devfactor', 2.0 : Standard deviation multiplier for Bollinger Bands.
- 'rsi_period', 14 : Period for RSI calculation.
- 'rsi_oversold', 30 : RSI level below which the asset is considered oversold.
- 'rsi_overbought', 70 : RSI level above which the asset is considered overbought.
- 'atr_period', 14 : Period for ATR calculation (for stop-loss).
- 'atr_multiplier', 2.0 : Multiplier for ATR-based stop-loss.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.bollinger_rsi_reversion import
BollingerRSIReversion

cerebro = bt.Cerebro()
cerebro.addstrategy(BollingerRSIReversion, rsi_oversold=25, rsi_overbought=75)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

8. DecisionTreeEMACrossoverMLStrategy (decision_tree_ema__crossover_ml_strategy.py)

This strategy combines a classical technical analysis signal, the Exponential Moving Average (EMA) crossover, with a machine learning model, specifically a Decision Tree. The EMA crossover likely provides initial potential trade signals (e.g., fast EMA crossing above slow EMA for a long signal). These signals are then likely fed as features (possibly along with other market data) into a pre-trained Decision Tree model, which makes the final decision to enter or ignore the trade. The "lookback_period" parameter suggests features for the model are generated from recent historical data.

Parameters

- 'fast_ema_period', 10 : Period for the fast EMA.
- 'slow_ema_period', 30 : Period for the slow EMA.
- 'lookback_period', 24 : Number of past periods used to generate features for the Decision Tree model.
- 'model_path', 'path/to/decision_tree_model.joblib' : Path to the pre-trained Decision Tree model file.
- 'feature_list', ['feature1', 'feature2'] : List of features used by the model.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.decision_tree_ema__crossover_ml_strategy
```



```
import DecisionTreeEMACrossoverMLStrategy

cerebro = bt.Cerebro()
# Ensure the model path and other ML-specific parameters are correctly set
cerebro.addstrategy(DecisionTreeEMACrossoverMLStrategy,
                    lookback_period=24,
                    model_path='path/to/your_model.pkl')

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

9. DemaCrossStrategy (dema_cross_strategy.py)

This strategy utilizes the crossover of two Double Exponential Moving Averages (DEMAs) to generate trading signals. A DEMA responds faster to price changes than a simple or exponential moving average. A long signal is typically generated when a faster DEMA crosses above a slower DEMA, and a short signal when the faster DEMA crosses below the slower DEMA.

Parameters

- 'fast_dema_period', 12 : Period for the faster DEMA.
- 'slow_dema_period', 26 : Period for the slower DEMA.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.dema_cross_strategy import DemaCrossStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(DemaCrossStrategy, fast_dema_period=10,
                  slow_dema_period=21)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

10. DualMaCrossAdxFilter (dual_ma_cross_adx_filter.py)

Implements a Dual Moving Average (MA) Crossover strategy, where trading signals are filtered by the ADX (Average Directional Index).

Entry Conditions:

- Long: A fast MA crosses above a slow MA, AND the ADX is above a specified threshold (indicating sufficient trend strength).
- Short: A fast MA crosses below a slow MA, AND the ADX is above the threshold.

Exit Conditions (Simple):

- Exit Long: The fast MA crosses back below the slow MA.
- Exit Short: The fast MA crosses back above the slow MA.

This aims to improve the reliability of MA crossover signals by only trading during trending market conditions.

Parameters

- 'fast_ma_period', 9 : Period for the fast moving average.
- 'slow_ma_period', 21 : Period for the slow moving average.
- 'ma_type', 'SMA' : Type of Moving Average to use (e.g., 'SMA', 'EMA').
- 'adx_period', 14 : Period for ADX calculation.
- 'adx_threshold', 20 : Minimum ADX value to confirm a trend and allow trades.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.dual_ma_cross_adx_filter import
DualMaCrossAdxFilter

cerebro = bt.Cerebro()
cerebro.addstrategy(DualMaCrossAdxFilter, fast_ma_period=10,
slow_ma_period=30, adx_threshold=25)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

11. DualRegimeWeeklyStrategy (dual_regime_weekly_strategy.py)

This strategy adapts its trading approach by identifying market regimes (e.g., trending or ranging/mean-reverting) typically on a weekly basis. It might use indicators like ADX or volatility measures on a weekly timeframe to determine the current regime. Once the regime is identified, it applies a suitable sub-strategy: a trend-following system during trending periods and a mean-reversion system during ranging periods.

Parameters

- 'adx_period', 14 : (If ADX is used for regime detection) Period for ADX on the weekly timeframe.
- 'adx_threshold', 25 : ADX threshold to differentiate between trending and ranging regimes.
- 'volatility_period', 20 : (If volatility is used) Period for volatility calculation.
- 'volatility_threshold_low', 0.01 : Volatility threshold.
- # Trend-following sub-strategy parameters (e.g., MA periods)
- 'trend_ma_fast', 10
- 'trend_ma_slow', 30
- # Mean-reversion sub-strategy parameters (e.g., BB periods, RSI levels)
- 'mr_bb_period', 20
- 'mr_rsi_period', 14
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.dual_regime_weekly_strategy import
DualRegimeWeeklyStrategy

cerebro = bt.Cerebro()
# Data feed should ideally be daily for the strategy to resample to weekly for
# regime detection
# cerebro.resampleddata(data, timeframe=bt.TimeFrame.Weeks) # Or handle
# resampling within strategy

cerebro.addstrategy(DualRegimeWeeklyStrategy, adx_period=10, adx_threshold=20)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
```

```
# cerebro.adddata(data)
# cerebro.run()
```

12. DynamicExitTrixStrategy (dynamic_exit_trix_strategy.py)

This strategy is an enhanced version of a TRIX-based system, primarily focusing on dynamic exit mechanisms.

Entry signals are likely derived from TRIX (Triple Smoothed Exponential Moving Average) crossovers with its signal line, potentially with additional confirmations (as suggested by "EnhancedTrixStrategy" mentioned in the description).

The key feature is its exit logic, which replaces simpler exits with an ATR (Average True Range) based trailing stop-loss. It might also incorporate an optional ATR-based profit target and/or a time-based stop (exiting a trade after a certain number of bars).

Parameters

(The snippet indicates parameters are inherited or similar to an `EnhancedTrixStrategy`. Common TRIX and dynamic exit parameters would be:)

- `# --- Entry Params (from a base TRIX strategy) ---`
- `'trix_period', 15` : Period for the TRIX calculation.
- `'trix_signal_period', 9` : Period for the TRIX signal line.
- `# --- Dynamic Exit Params ---`
- `'atr_period_stop', 14` : ATR period for the trailing stop-loss.
- `'atr_multiplier_stop', 3.0` : ATR multiplier for the trailing stop-loss.
- `'use_profit_target', False` : Boolean to enable ATR-based profit target.
- `'atr_period_target', 14` : ATR period for profit target.
- `'atr_multiplier_target', 5.0` : ATR multiplier for profit target.
- `'use_time_stop', False` : Boolean to enable time-based stop.
- `'time_stopBars', 50` : Number of bars for the time stop.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.dynamic_exit_trix_strategy import
DynamicExitTrixStrategy
```

```
cerebro = bt.Cerebro()
cerebro.addstrategy(DynamicExitTrixStrategy, trix_period=15,
atr_multiplier_stop=2.5)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

13. EmaCrossStrategy (ema_cross_strategy.py)

A straightforward trend-following strategy based on the crossover of two Exponential Moving Averages (EMAs).

A long signal is generated when a shorter-period (fast) EMA crosses above a longer-period (slow) EMA.

A short signal is generated when the fast EMA crosses below the slow EMA.

Exits usually occur on the reverse crossover.

Parameters

- 'fast_ema_period', 9 : Period for the fast Exponential Moving Average.
- 'slow_ema_period', 21 : Period for the slow Exponential Moving Average.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ema_cross_strategy import EmaCrossStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(EmaCrossStrategy, fast_ema_period=10, slow_ema_period=30)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

14. EmaCrossWithSmaFilterStrategy (ema_cross_with_sma_filter_strategy.py)

This strategy uses an Exponential Moving Average (EMA) crossover for primary buy/sell signals, but these signals are filtered by a longer-term Simple Moving Average (SMA). For example, a long signal from an EMA crossover (fast EMA crosses above slow EMA) would only be taken if the price is also above the long-term SMA, indicating the overall trend is bullish. Similarly, short signals would only be taken if the price is below the SMA. This helps to avoid taking signals against the dominant market trend.

Parameters

- 'fast_ema_period', 9 : Period for the fast EMA.
- 'slow_ema_period', 21 : Period for the slow EMA (forms the crossover with the fast EMA).
- 'sma_filter_period', 200 : Period for the long-term SMA trend filter.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ema_cross_with_sma_filter_strategy import
EmaCrossWithSmaFilterStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(EmaCrossWithSmaFilterStrategy, fast_ema_period=12,
slow_ema_period=26, sma_filter_period=100)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

15. EmaMacdAdxStrategy (ema_macd_adx_strategy.py)

This is a multi-filter trend-following strategy. It uses an EMA (Exponential Moving Average) crossover for the initial signal direction. This signal is then confirmed by:

1. MACD (Moving Average Convergence Divergence): e.g., MACD line must be above its signal line for a long trade.
 2. ADX (Average Directional Index): ADX must be above a certain threshold, indicating sufficient trend strength to support the trade.
- Exits can be triggered by a reversal in the EMA crossover, a MACD reversal, or a trailing stop-loss.

Parameters

- 'ema_fast_period', 12 : Period for the fast EMA.
- 'ema_slow_period', 26 : Period for the slow EMA.
- 'macd_fast_period', 12 : Fast period for MACD.
- 'macd_slow_period', 26 : Slow period for MACD.
- 'macd_signal_period', 9 : Signal line period for MACD.
- 'adx_period', 14 : Period for ADX.
- 'adx_threshold', 20 : Minimum ADX value to validate a trend.
- 'atr_period_stop', 14 : (If using ATR trailing stop) Period for ATR.
- 'atr_multiplier_stop', 3.0 : Multiplier for ATR trailing stop.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ema_macd_adx_strategy import
EmaMacdAdxStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(EmaMacdAdxStrategy, adx_threshold=25)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

16. EnhancedBBMeanReversion (enhanced_bb_mean_reversion.py)

An enhanced version of the Bollinger Bands Mean Reversion strategy. It adds extra filters to improve the quality of entry signals.

Entry Conditions for Long:

- Price closes below the lower Bollinger Band.
- ADX is below a specified `adx_threshold` (to confirm a sideways or non-trending market, where mean reversion is more likely to succeed).
- RSI is below `rsi_lower` (confirming oversold conditions).

Entry Conditions for Short (if enabled):

- Price closes above the upper Bollinger Band.
- ADX is below `adx_threshold`.
- RSI is above `rsi_upper` (confirming overbought conditions).

Exit Conditions:

- Typically, close long when the price crosses back above the middle Bollinger Band.
- Close short when the price crosses back below the middle Bollinger Band.

Parameters

- `'bb_period', 20` : Period for Bollinger Bands.
- `'bb_devfactor', 2.0` : Standard deviation factor for Bollinger Bands.
- `'adx_period', 14` : Period for ADX.
- `'adx_threshold', 25` : ADX value below which the market is considered ranging.
- `'rsi_period', 14` : Period for RSI.
- `'rsi_lower', 30` : RSI oversold threshold.
- `'rsi_upper', 70` : RSI overbought threshold.
- `'enable_shorting', True` : Whether to allow short trades.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.enhanced_bb_mean_reversion import
EnhancedBBMeanReversion

cerebro = bt.Cerebro()
cerebro.addstrategy(EnhancedBBMeanReversion, adx_threshold=20, rsi_lower=25,
rsi_upper=75)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

17. EnhancedTrixStrategy (enhanced_trix_strategy.py)

An enhanced strategy based on the TRIX (Triple Smoothed Exponential Moving Average) indicator. It typically uses the crossover of the TRIX line and its signal line for primary entry/exit signals. The "enhanced" aspect comes from incorporating several confirmation filters:

1. SMA Trend Filter: Price must be on the correct side of a long-term SMA (e.g., above for longs).
2. SMA Slope Confirmation: The slope of the long-term SMA should confirm the trend direction.
3. N-Period High/Low Breakout Confirmation: Entry might require a breakout of a recent high (for longs) or low (for shorts).
4. TRIX Slope Threshold Filter: The TRIX line itself must have a slope exceeding a certain threshold, indicating strong momentum.

Parameters

- 'trix_period', 15 : Period for TRIX calculation.
- 'trix_signal_period', 9 : Period for TRIX signal line.
- 'sma_filter_period', 200 : Period for the long-term SMA trend filter.
- 'sma_slope_period', 10 : Period over which to calculate the SMA slope.
- 'breakout_period', 20 : Lookback period for N-period high/low breakout confirmation.
- 'trix_slope_threshold', 0.001 : Minimum slope for the TRIX line to confirm momentum.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.enhanced_trix_strategy import
EnhancedTrixStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(EnhancedTrixStrategy, trix_period=15,
sma_filter_period=100)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

18. EnsembleStrategyWithFixedWeights (ensemble_strategy_with_fixed_weights.py)

This strategy implements an ensemble trading approach. It combines signals from multiple, distinct sub-models (or sub-strategies). The final trading decision (e.g., buy, sell, hold, or position size) is determined by aggregating the signals from these sub-models using pre-

defined, fixed weights. These weights are passed as parameters. The assumption is that these fixed weights are determined externally, perhaps through prior optimization or performance analysis, and are updated periodically if needed.

Parameters

(Parameters will be a combination of controls for the ensemble logic and parameters for each sub-model included in the ensemble.)

- # --- Sub-Model 1 (e.g., SMA Crossover) Parameters ---
- 'submodel1_sma_fast_p', 10
- 'submodel1_sma_slow_p', 30
- 'submodel1_weight', 0.4
- # --- Sub-Model 2 (e.g., RSI Mean Reversion) Parameters ---
- 'submodel2_rsi_period', 14
- 'submodel2_rsi_oversold', 30
- 'submodel2_weight', 0.3
- # --- Sub-Model 3 (e.g., MACD) Parameters ---
- 'submodel3_macd_fast', 12
- 'submodel3_macd_slow', 26
- 'submodel3_weight', 0.3
- 'threshold_buy', 0.5 : Aggregate weighted signal threshold to trigger a buy.
- 'threshold_sell', -0.5 : Aggregate weighted signal threshold to trigger a sell/short.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ensemble_strategy_with_fixed_weights import
EnsembleStrategyWithFixedWeights

cerebro = bt.Cerebro()
cerebro.addstrategy(EnsembleStrategyWithFixedWeights,
                    submodel1_sma_fast_p=10, submodel1_weight=0.5,
                    submodel2_rsi_period=14, submodel2_weight=0.5)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

19. GuppyMMAStrategy (guppy_mma_strategy.py)

This strategy is based on the Guppy Multiple Moving Average (GMMA) indicator. The GMMA consists of two groups of Exponential Moving Averages (EMAs): a short-term group and a long-term group.

- The short-term group (e.g., 3, 5, 8, 10, 12, 15 periods) reflects the sentiment of short-term traders.
- The long-term group (e.g., 30, 35, 40, 45, 50, 60 periods) reflects the sentiment of long-term investors.

Trading signals are typically generated when:

- The two groups of MAs cross.
- The short-term MAs show compression then expansion (indicating a potential breakout).
- Price pulls back to and finds support/resistance at one of the groups.

The "Slope" class name in the original snippet might refer to an auxiliary indicator used within the Guppy strategy or could be the strategy class name itself if it focuses on specific slope aspects of the Guppy MAs. Assuming `GuppyMMAStrategy` as a more descriptive name for the overall strategy.

Parameters

- `'short_periods', (3, 5, 8, 10, 12, 15)` : Tuple of periods for the short-term EMAs.
- `'long_periods', (30, 35, 40, 45, 50, 60)` : Tuple of periods for the long-term EMAs.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the main strategy class is named GuppyMMAStrategy,
# or if 'Slope' is the actual strategy class name from the file:
from strategy_package_no_numbers.guppy_mma_strategy import Slope as
GuppyMMAStrategy # Or actual strategy class name

cerebro = bt.Cerebro()
cerebro.addstrategy(GuppyMMAStrategy,
                    short_periods=(3, 5, 8, 10, 12, 15),
                    long_periods=(30, 35, 40, 45, 50, 60))

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

20. HilbertRsiStrategy (hilbert_rsi_strategy.py)

This strategy combines signals from the Hilbert Transform's Sine Wave indicator with several filters: RSI, ATR (Average True Range), and ADX (Average Directional Index).

The Hilbert Transform Sine Wave helps identify cycle modes and potential turning points. A crossover of the Sine and LeadSine waves might generate primary signals. These signals are then likely filtered:

- RSI: To confirm overbought/oversold conditions or momentum.
- ATR: Possibly for volatility-based stops or profit targets, or as a volatility filter.
- ADX: To ensure trades are taken in trending markets if it's a trend-following component, or to identify ranging markets if it's a cycle-based component.

It also includes a percentage-based Trailing Stop Loss for risk management.

Parameters

- 'hilbert_period', 20 : (Typical, but not specified) Period for Hilbert Transform calculations if applicable.
- 'rsi_period', 14 : Period for the RSI.
- 'rsi_oversold', 30 : RSI oversold level.
- 'rsi_overbought', 70 : RSI overbought level.
- 'atr_period', 14 : Period for ATR.
- 'adx_period', 14 : Period for ADX.
- 'adx_threshold', 20 : ADX threshold for trend confirmation.
- 'trailing_stop_pct', 0.02 : Percentage for the trailing stop-loss (e.g., 2%).
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.hilbert_rsi_strategy import
HilbertRsiStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(HilbertRsiStrategy, rsi_period=10, trailing_stop_pct=0.03)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

21. HilbertTrendStrategy (hilbert_trend_strategy.py)

This strategy utilizes the Hilbert Transform to identify the dominant market cycle and distinguish between trend and cycle modes. When a trend mode is detected by the Hilbert Transform (e.g., using the "Dominant Cycle Period" or "Trend vs Cycle Mode" output of the transform), the strategy likely initiates trades in the direction of the identified trend. It might use the Sine Wave or Instantaneous Trendline components of the Hilbert Transform for entry and exit signals within the established trend mode.

Parameters

(Specific parameters are not provided. Common parameters for a Hilbert Transform based strategy include:)

- 'period', 20 : (If applicable to a specific smoothing or lookback within the Hilbert calculations, though often parameterless for the core transform).
- 'price_type', 'close' : The price (e.g., close, hlc3) used for Hilbert Transform input.
- 'trend_confirmation_indicator', None : (Optional) Another indicator to confirm Hilbert trend signals.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.hilbert_trend_strategy import
HilbertTrendStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(HilbertTrendStrategy) # Potentially parameter-less or with
minimal params

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

22. HMMTrendRegimeStrategy (hmm_trend_regime_strategy.py)

This strategy employs a Hidden Markov Model (HMM) to identify underlying market regimes or states (e.g., bullish trend, bearish trend, sideways/volatile). The HMM is trained on historical market data (like price changes, volatility) to learn these hidden states. Once the current market regime is predicted by the HMM, the strategy adapts its trading logic accordingly. For example, it might use trend-following tactics in a bullish or bearish regime and mean-reverting tactics or stay out during a sideways regime. The class name `HMMData` is unusual for a strategy; it might be a wrapper or the actual strategy class. Assuming it's the strategy class here.

Parameters

- `'n_states', 3` : Number of hidden states for the HMM (e.g., bull, bear, range).
- `'features', ['returns', 'volatility']` : Market features used to train the HMM and predict states.
- `'training_period', 252` : Length of historical data used to train/retrain the HMM.
- `'retrain_interval', 60` : How often to retrain the HMM (in bars).
- `'model_path', None` : Path to a pre-trained HMM model; if None, it's trained dynamically.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming HMMData is the strategy class name from the file
from strategy_package_no_numbers.hmm_trend_regime_strategy import HMMData as
HMMTrendRegimeStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(HMMTrendRegimeStrategy, n_states=3, training_period=200)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

23. HurstFilteredTrendStrategy (hurst_filtered_trend_strategy.py)

This strategy likely combines a primary trend-following mechanism (e.g., moving average crossover, breakout) with a filter based on the Hurst Exponent. The Hurst Exponent measures the long-term memory or persistence of a time series.

- $H > 0.5$ suggests a trending (persistent) series.
- $H < 0.5$ suggests a mean-reverting (anti-persistent) series.
- $H = 0.5$ suggests a random walk.

The strategy would use the calculated Hurst Exponent to confirm if the market is currently trending before taking signals from its primary trend indicators. This aims to improve the performance of trend-following systems by avoiding trades in choppy or mean-reverting markets. The `HurstExponentIndicator` class mentioned in the snippet is likely used internally by this strategy.

Parameters

- `# --- Hurst Exponent Parameters ---`
- `'hurst_window', 100` : The lookback window for calculating the Hurst Exponent.
- `'hurst_threshold_trend', 0.55` : Minimum Hurst value to confirm a trending market.
- `# --- Primary Trend Strategy Parameters (e.g., EMA Crossover) ---`
- `'fast_ema_period', 10` : Period for fast EMA.
- `'slow_ema_period', 30` : Period for slow EMA.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the strategy class is HurstFilteredTrendStrategy,
# and it uses HurstExponentIndicator internally.
from strategy_package_no_numbers.hurst_filtered_trend_strategy import
HurstFilteredTrendStrategy # Or actual strategy class

cerebro = bt.Cerebro()
cerebro.addstrategy(HurstFilteredTrendStrategy, hurst_window=100,
hurst_threshold_trend=0.52)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

(Note: The snippet referred to `HurstExponentIndicator` as the class being added. If `HurstExponentIndicator` is the strategy itself, it would likely only output the Hurst value or trade based on its raw thresholds, which is less common than using it as a filter.)

24. IsolationForestStrategy (isolation_forest_strategy.py)

This strategy utilizes an Isolation Forest machine learning algorithm for anomaly detection in financial time series data. Anomalies detected by the Isolation Forest (data points that are easily "isolated") are interpreted as potential trading opportunities or signals for unusual market behavior. For instance, a sharp, anomalous price drop might be flagged as a buying opportunity (if considered an overreaction) or a signal to exit. The strategy requires a pre-trained Isolation Forest model or trains one dynamically on historical data. The class name `AnomalyDetectionModel` in the snippet might be the strategy class or a helper; assuming it's the strategy.

Parameters

- `'n_estimators', 100` : Number of base estimators (trees) in the Isolation Forest.
- `'contamination', 'auto' or float (e.g., 0.01)` : The expected proportion of outliers in the data set.
- `'lookback_period', 50` : Number of bars of historical data to use for feature engineering or model (re)training.
- `'feature_list', ['returns', 'volatility']` : List of features for the model.
- `'historical_data_path', 'path/to/training_data.csv'` : Path to data for initial model training (if not training dynamically on the main data feed).
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming AnomalyDetectionModel is the strategy class name from the file
from strategy_package_no_numbers.isolation_forest_strategy import
AnomalyDetectionModel as IsolationForestStrategy

cerebro = bt.Cerebro()
# Ensure model parameters or path to a pre-trained model are set
cerebro.addstrategy(IsolationForestStrategy, n_estimators=100,
contamination=0.02)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```


25. KeltnerADXStrategy (keltner_adx_strategy.py)

This strategy combines Keltner Channels with the ADX (Average Directional Index).

Keltner Channels are volatility-based bands placed above and below an Exponential Moving Average (EMA). Trades are often initiated on breakouts above the upper Keltner Channel or below the lower Keltner Channel.

The ADX is used as a trend strength filter. Breakout signals are typically only taken if the ADX indicates that a strong trend is present, aiming to avoid false breakouts in choppy markets.

The `KeltnerChannels` class mentioned in the snippet is likely an indicator used internally by this strategy.

Parameters

- `# --- Keltner Channel Parameters ---`
- `'kc_ema_period', 20` : Period for the central EMA of the Keltner Channels.
- `'kc_atr_period', 10` : Period for the ATR (Average True Range) used to calculate channel width.
- `'kc_atr_multiplier', 2.0` : Multiplier for the ATR to set the channel width.
- `# --- ADX Parameters ---`
- `'adx_period', 14` : Period for ADX calculation.
- `'adx_threshold', 25` : Minimum ADX value to confirm a trend and validate breakouts.
- `'printlog', False` : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the strategy class is KeltnerADXStrategy and uses KeltnerChannels
# indicator internally
from strategy_package_no_numbers.keltner_adx_strategy import
KeltnerADXStrategy # Or actual strategy class name

cerebro = bt.Cerebro()
cerebro.addstrategy(KeltnerADXStrategy, kc_ema_period=20,
kc_atr_multiplier=1.5, adx_threshold=20)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

(Note: The snippet referred to `KeltnerChannels` as the class being added. If `KeltnerChannels` is the strategy itself, its logic would be simpler, focusing just on Keltner interactions.)

26. MaBounceStrategy (ma_bounce_strategy.py)

This strategy is designed to trade "bounces" off a key Moving Average (MA). When the price pulls back to a significant MA (acting as dynamic support in an uptrend or resistance in a downtrend) and then shows signs of rejecting that level and resuming the trend, a trade is initiated. Confirmation might come from price patterns (e.g., a bullish engulfing candle at support) or other short-term indicators.

Parameters

- 'key_ma_period', 50 : The period of the key Moving Average (e.g., 50-day MA).
- 'ma_type', 'SMA' : Type of Moving Average (e.g., 'SMA', 'EMA', 'WMA').
- 'confirmation_lookback', 3 : Number of bars for bounce confirmation (e.g., price closing back above MA after touching it).
- 'rsi_confirm_period', 14 : (Optional) RSI period if used for confirming bounce (e.g., RSI moving out of oversold/overbought).
- 'stop_loss_atr_period', 14 : ATR period for setting stop-loss below/above the bounce point.
- 'stop_loss_atr_mult', 1.5 : ATR multiplier for stop-loss.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ma_bounce_strategy import MaBounceStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(MaBounceStrategy, key_ma_period=50, ma_type='EMA')

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

27. MaDistanceStrategy (ma_distance_strategy.py)

This strategy trades based on the price's distance from a specified Moving Average (MA). It can be implemented as either a mean-reversion or a momentum strategy.

- **Mean Reversion:** If the price moves too far from the MA (e.g., a certain percentage or number of ATRs), the strategy might anticipate a return to the mean and trade accordingly (sell if far above, buy if far below).
- **Momentum:** Alternatively, if the price shows a strong and sustained deviation from the MA, it might be interpreted as the start of a new trend.
The exact logic depends on how "distance" is defined and the thresholds used.

Parameters

- 'ma_period', 20 : Period of the Moving Average.
- 'ma_type', 'SMA' : Type of Moving Average.
- 'distance_metric', 'percent' : How distance is measured ('percent', 'atr', 'points').
- 'threshold_upper', 0.05 : Upper distance threshold for triggering a signal (e.g., 5% above MA).
- 'threshold_lower', -0.05 : Lower distance threshold for triggering a signal (e.g., 5% below MA).
- 'reversion_mode', True : If True, trades for mean reversion; if False, might trade breakouts.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.ma_distance_strategy import
MaDistanceStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(MaDistanceStrategy, ma_period=20, threshold_upper=0.03,
threshold_lower=-0.03)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

28. MaRibbonPullbackStrategy (ma_ribbon_pullback_strategy.py)

This strategy uses a "ribbon" of multiple Moving Averages (typically EMAs with progressively increasing periods) to identify the trend direction and strength. Trades are usually initiated

during pullbacks to this ribbon in an established trend. For example, in an uptrend (ribbon EMAs are stacked bullishly and angling up), a buy signal might occur when the price temporarily dips into or near the ribbon and then shows signs of resuming the uptrend. The "Slope" class name in the original snippet is likely an auxiliary indicator or the strategy's class name focusing on the ribbon's slope.

Parameters

- 'ema_periods', (5, 8, 11, 14, 17, 20) : A tuple defining the periods for the EMAs forming the ribbon.
- 'pullback_depth_factor', 0.5 : (Example) Factor determining how deep into the ribbon a pullback should be.
- 'confirmation_indicator', None : (Optional) An indicator like RSI or a candle pattern for pullback confirmation.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the main strategy class is MaRibbonPullbackStrategy,
# or if 'Slope' is the actual strategy class name from the file:
from strategy_package_no_numbers.ma_ribbon_pullback_strategy import Slope as
MaRibbonPullbackStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(MaRibbonPullbackStrategy, ema_periods=(10, 12, 15, 18, 21,
25))

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

29. MaSlopeStrategy (ma_slope_strategy.py)

This strategy bases its trading decisions on the slope of a Moving Average (MA). A positive slope indicates an uptrend, potentially triggering buy signals, while a negative slope suggests a downtrend, potentially triggering sell/short signals. The steepness of the slope can also be used to gauge the momentum of the trend. Trades might be entered when the slope crosses a certain threshold or changes direction. The class name "Slope" in the snippet likely refers to the strategy class itself or a key indicator it uses.

Parameters

- 'ma_period', 30 : Period of the Moving Average whose slope is analyzed.
- 'ma_type', 'SMA' : Type of Moving Average.
- 'slope_lookback', 1 : Number of bars to calculate the MA's slope (e.g., difference between MA now and MA 1 bar ago).
- 'slope_threshold_long', 0.001 : Minimum positive slope to consider an uptrend for a long entry.
- 'slope_threshold_short', -0.001 : Maximum negative slope (minimum absolute negative value) for a short entry.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming the class name from the file is 'Slope'
from strategy_package_no_numbers.ma_slope_strategy import Slope as
MaSlopeStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(MaSlopeStrategy, ma_period=50,
slope_threshold_long=0.0005)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

30. MACDStrategy (macd_strategy.py)

This strategy employs the Moving Average Convergence Divergence (MACD) indicator for trading signals. Common signals include:

1. **MACD Line / Signal Line Crossover:** A buy signal when the MACD line crosses above its signal line; a sell signal when it crosses below.
2. **Zero Line Crossover:** A buy signal when the MACD line crosses above the zero line (indicating increasing bullish momentum); a sell signal when it crosses below (increasing bearish momentum).
3. **Divergence:** Bullish divergence (price makes lower lows, MACD makes higher lows) or bearish divergence (price makes higher highs, MACD makes lower highs).

Parameters

- 'macd_fast_period', 12 : Period for the fast EMA in MACD calculation (often referred to as me1 or period_me1).
- 'macd_slow_period', 26 : Period for the slow EMA in MACD calculation (often referred to as me2 or period_me2).
- 'macd_signal_period', 9 : Period for the signal line EMA (often referred to as sig or period_signal).
- 'trade_on_signal_cross', True : Boolean to trade on MACD/Signal line crossovers.
- 'trade_on_zero_cross', False : Boolean to trade on MACD zero line crossovers.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.macd_strategy import MACDStrategy

cerebro = bt.Cerebro()
# Parameters in backtrader's MACD indicator are often period_me1, period_me2,
period_signal
cerebro.addstrategy(MACDStrategy, macd_fast_period=12, macd_slow_period=26,
macd_signal_period=9)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

31. MovingAverageChannelStrategy (moving_average_channel_strategy.py)

This strategy uses a channel constructed around one or more Moving Averages (MAs). The channel boundaries can be created by:

1. Offsetting a single MA by a fixed percentage or a multiple of ATR (Average True Range).
2. Using two MAs (e.g., a high MA and a low MA calculated on high and low prices, or two EMAs with a spread).

Trades can be initiated on breakouts from the channel (momentum) or reversions from the channel boundaries back towards the central MA (mean reversion).

Parameters

- 'ma_period', 30 : Period for the central Moving Average.
- 'ma_type', 'SMA' : Type of Moving Average.
- 'channel_width_metric', 'percent' : How channel width is determined ('percent', 'atr', 'fixed_points').
- 'channel_width_value', 0.02 : Value for the chosen metric (e.g., 2% or 2 ATRs).
- 'trade_on_breakout', True : If True, trades breakouts; if False, might trade reversions.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.moving_average_channel_strategy import
MovingAverageChannelStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(MovingAverageChannelStrategy, ma_period=20,
channel_width_metric='atr', channel_width_value=1.5)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

32. RsiMeanReversionFiltered (rsi_mean_reversion_filtered.py)

This strategy implements a mean-reversion approach using the Relative Strength Index (RSI), but with additional trend filters to enhance signal quality.

The core idea is to buy when RSI indicates oversold conditions (e.g., $RSI < 30$) and sell/short when RSI indicates overbought conditions (e.g., $RSI > 70$).

The "filtered" aspect means these RSI signals are only acted upon if one or more trend filters confirm that the market is not strongly trending against the potential reversion trade (e.g., price might be above a long-term MA for long entries, or ADX might be low indicating a ranging market).

Parameters

- 'rsi_period', 14 : Lookback period for RSI calculation.

- 'rsi_oversold', 30 : RSI level below which an asset is considered oversold.
- 'rsi_overbought', 70 : RSI level above which an asset is considered overbought.
- 'trend_filter_ma_period', 200 : (Example filter) Period for a long-term MA trend filter.
- 'adx_filter_period', 14 : (Example filter) Period for ADX if used as a range filter.
- 'adx_filter_threshold', 25 : ADX value below which market is considered ranging.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
# Assuming class name in file is RsiStrategyWithTrendFilters as per snippet
from strategy_package_no_numbers.rsi_mean_reversion_filtered import
RsiStrategyWithTrendFilters as RsiMeanReversionFiltered

cerebro = bt.Cerebro()
cerebro.addstrategy(RsiMeanReversionFiltered, rsi_oversold=25,
rsi_overbought=75, trend_filter_ma_period=100)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

33. TrendFilterEmaCrossStrategy (trend_filter_ema_cross_strategy.py)

This strategy is based on an Exponential Moving Average (EMA) crossover for generating buy and sell signals. However, to improve reliability, these crossover signals are subjected to an additional, explicit trend filter. This filter could be a longer-term moving average, ADX, or another indicator that helps confirm the direction and strength of the prevailing market trend. Only EMA crossover signals that align with the broader trend indicated by the filter are taken.

Parameters

- 'fast_ema_period', 9 : Period for the fast EMA.
- 'slow_ema_period', 21 : Period for the slow EMA (crossover with fast EMA).
- 'trend_filter_type', 'SMA' : Type of trend filter (e.g., 'SMA', 'ADX', 'SuperTrend').
- 'trend_filter_period', 100 : Period for the trend filter indicator.
- 'adx_threshold_filter', 20 : (If ADX is the filter) ADX threshold.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.trend_filter_ema_cross_strategy import
TrendFilterEmaCrossStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(TrendFilterEmaCrossStrategy, fast_ema_period=10,
slow_ema_period=30, trend_filter_type='SMA', trend_filter_period=150)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```

34. TripleEmaCrossStrategy (triple_ema_cross_strategy.py)

This strategy employs three Exponential Moving Averages (EMAs) with different periods: a fastest, a medium, and a slowest EMA.

Trading signals are generated based on the crossovers and relative positions of these three EMAs. A common approach:

- **Long Entry:** Fastest EMA crosses above the medium EMA, AND both are above the slowest EMA (confirming an established uptrend). Or, medium EMA crosses above slowest EMA as a primary trend signal, with faster EMA used for refined entry.
- **Short Entry:** Fastest EMA crosses below the medium EMA, AND both are below the slowest EMA.

This provides more confirmation and potentially smoother signals compared to a dual EMA crossover.

Parameters

- 'fastest_ema_period', 5 : Period for the fastest EMA.
- 'medium_ema_period', 10 : Period for the medium EMA.
- 'slowest_ema_period', 20 : Period for the slowest EMA.
- 'printlog', False : Enable/disable trade logging.

Usage Example

```
import backtrader as bt
from strategy_package_no_numbers.triple_ema_cross_strategy import
TripleEmaCrossStrategy

cerebro = bt.Cerebro()
cerebro.addstrategy(TripleEmaCrossStrategy, fastest_ema_period=5,
medium_ema_period=13, slowest_ema_period=26)

# data = bt.feeds.SomeDataFeed(...) # Replace with your actual data feed
# cerebro.adddata(data)
# cerebro.run()
```