Quant Developer Assignment

Multi-Timeframe Strategy Execution & Trade Matching

Overview

This project implements a deterministic, rule-based multi-timeframe trading strategy using a clean, modular, class-based Python architecture. The system is designed to ensure strict **execution parity** between backtesting and live trading by employing a single strategy logic source across both environments.

The implementation prioritizes engineering rigor, execution correctness, and reproducibility over strategy profitability.

Strategy Logic

The strategy is a multi-timeframe trend-following and momentum system.

Timeframes and Indicators

- **15-Minute Timeframe:** Primary interval for entry and exit signals.
  - Indicators: RSI(14), SMA(20)
- **1-Hour Timeframe:** Secondary interval for trend confirmation filtering.
  - Indicators: SMA(50)

Trading Rules (Long Only)

- **Entry Rule:** Initiate a BUY order when:
  1. 1H Close $>$ 1H SMA(50) (Trend Confirmation)
  2. 15M RSI $<$ 45 (Mean Reversion Pullback)
  3. System is not currently in a position.
- **Exit Rule:** Liquidate position when:
  1. 15M RSI $>$ 70

Architecture & Implementation

The system architecture enforces a strict separation of concerns to ensure modularity.

File Structure

- `indicators.py`: Core functions for indicator computation.
- `strategy.py`: The **Single Source of Truth** for decision logic.
- `backtest_engine.py`: Adapter for the *backtesting.py* framework.
- `live_trader.py`: Live execution module for Binance Testnet.
- `run_backtest.py`: Runner script for simulations and CSV export.

Single Source of Truth

All signal generation is centralized in `MultiTimeframeStrategy.generate_signal()`. This class is:

- **Stateless:** Does not maintain internal trade state, preventing drift.
- **Framework-Agnostic:** Contains pure decision logic with no dependencies on the backtesting engine or live API.
- **Reused:** Imported unchanged by both the backtester and the live trader.

Backtesting and Trade Logging

The backtesting suite uses `backtesting.py` with a custom adapter (`BacktestStrategy`).

- **Data Slicing:** The adapter ensures the strategy only sees data up to the current bar to prevent look-ahead bias.
- **Logging:** Post-run, trades are extracted and saved to `backtest_trades.csv` with a schema including timestamp, symbol, direction, entry price, and exit price.

Live Trading (Binance Testnet)

Live execution utilizes the *python-binance* REST API. The live trader follows a rigid flow:

1. Fetch latest 15m candles and compute indicators.
2. Resample data to 1H to match the backtest trend filter.
3. Invoke the shared strategy class for signal generation.
4. Execute Market Orders and log results to `live_trades.csv`.

Execution Parity & Validation

Parity is verified by maintaining identical indicator functions, timeframe construction, and signal method calls.

Results Comparison

- **Backtest Trades:** 29 generated.
- **Live Trades:** 0 (during limited runtime window).

Validation: The zero-trade result in the live window is expected due to conservative RSI thresholds and short execution time. The focus remains on the fact that the **logic flow** remained identical across both layers.

Reproducibility

- **Backtest:** Execute `python run_backtest.py`.
- **Live:** Execute `python live_trader.py`.