Backtesting

Package structure

- `data` - contains the input data in adjusted format
- `results` - results are saved there by default
- `visualization` - contains visualization scripts
- `src` - source code of the engine
- `mean_reversion` - Mean Reversion strategy
- `trend_following_style_1` - Trend Following strategy with "style 1" TP/SLs
- `trend_following_style_2` - Trend Following strategy with "style 2" TP/SLs

Installation

Since the source is not pushed to the PIP you need to install from the local directory. It is `src` in the package' root.

Linux:

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install ./src
python3 -m pip install -r requirements.txt
```

Windows:

```
py -m venv venv
venv\Scripts\activate
py -m pip install ./src
py -m pip install -r requirements.txt
```

Usage

To run backtesting

```
python3 trend_following_style_2.py
```

To show the trades (first-trade-id:last-trade-id) on a chart

```
python3 visualization/display_trend_following_style2.py --trades 0:20
```

Indicators

- MA - Moving Average
- EMA - Exponential Moving Average
- MACD - Moving Average Convergence Divergence
- BBANDS - Bollinger Bands
- RSI - Relative Strength Index
- ATR - Average True Range
- ADX - Average Directional Movement Index
- DMI - Directional Movement Indicator
- PIVOT - Pivot Points (Traditional, Fibonacci or Classic)

- KELTNER_CHANNEL - Keltner Channel (EMA & ATR algorithm)

Core concepts

How it works

In the `FuturesStrategy` class you're required to implement `tick` method and provide the `indicators` and `candle_timeframes` fields. Since it is unpredictable when would the user code access upsampled OHLCV or indicators (possibly in each `tick` method call), you specify this ahead of time so engine would do its routine to keep the values always ready.

Then it starts with prefetching. In order for indicators to have the values already computed by the start of the backtesting we pull some data first. Then the engine will iterate over the rest of the data, update some internal buffers, review opened orders and call the `tick` method of your strategy where its logic is defined. The `tick` method is called each time a new candle closes and your strategy operates just before the moment when the next candle comes up.

Broker

Broker provides orders management in a simulated market environment. The broker executes/activates orders whose conditions fits the market. Supports Market, Limit, Take Profit, Take Profit Limit, Stop Loss, Stop Loss Limit orders. Order execution policy of builtin broker:

- **Market orders**: All market orders will be executed when a new candle closes. The price of execution is the candle's OPEN price.
- **Limit orders**: Limit order will be executed at the limit price or better: lower or equal price for BUY orders and higher or equal price for SELL orders. First, the `order_price` of each order will be compared to the OPEN price of a new candle:

  ```
  BUY orders will be executed if `order_price` >= OPEN.
  SELL orders will be executed if `order_price` <= OPEN.
  ```
  Then, remaining BUYs will be compared to LOW,
  and remaining SELLs - to HIGH.
  Fill price is the first price that matched limit price.
  ```

- **Take Profit/Stop Loss orders**: TP/SL orders will be activated if the `trigger_price` is within the price bounds of a candle. This check is performed in two steps:

  ```
  1) For each order: activate if trigger_price == OPEN
  2) For each order: activate if LOW <= trigger_price <= HIGH
  ```

  When a TP/SL order is triggered, it will be treated as a market or limit order, depending on whether `order_price` is set for the order.

Limit, Take Profit and Stop Loss orders are reviewed in the order of their submission (oldest first).

Analyser
  ```

Indicators calculation. See list of supported indicators.

Candles

Provides the last candle representation for various timeframes. It is useful for checking properties of a candle on one timeframe (H1, for example), while having data on another (for instance, M1).

DataProvider

Provides candles in historical order. `DataProvider` is an iterable object that can be created for specific date range (since, until); Yields OHLCV candle during iteration.

BacktestingResult

Provides export to CSV and stats such as Win Rate, Profit/Loss, Average Profit, Best/Worst deal, etc. The algorithm currently used for futures strategies only produce the correct result if SELL order always follows only one BUY. It just so happens that is the case for the strategies implemented.

Prefetching

Indicators require a certain amount of data to get a correct result. For example, to calculate the SMA (simple moving average) with a period of 9, 9 values are required. So, the strategy will get the wrong result of the SMA indicator, until all 9 candles are accumulated.

In order for the strategy to get the correct values right from the start, prefetching of market data is used. You can configure this behavior by choosing from the following options and passing it as `prefetch_option` argument to the `run_backtest` function:

- **PREFETCH_UNTIL** (default) - prefetch the required amount of data until `since` date; the amount of data required is calculated automatically. In this case, the strategy backtesting will be started from the `since` date. This option is convenient when market data is requested from the exchange API, because situations when the exchange does not have enough historical data are quite rare.
- **PREFETCH_SINCE** - prefetch the required amount of data from `since` date; the amount of data required for this is calculated automatically. In this case, the strategy backtesting will be launched starting from the dynamically calculated date. This option may be useful when working with a CSV file when you are not sure if it contains enough data before the `since` date.
- **PREFETCH_NONE** - Do not prefetch data.

Where `since` date is the value of the argument `since` passed to the `run_backtest` function.

Tests

The tests for indicators, candles upsampling, balance operations, orders submission/execution/cancellation are shipped in the `src/tests` folder. `pytest` package is required. For instance, to run a test for futures execution:

```
pytest src/tests/broker_test/futures_execution_test.py
```

Limitations

- Splitted positions won't work with the profit estimation algorithm and with the shipped broker (partially). That's not the case with the implemented strategies, anyway, but if you consider to use orders grid be warned that's not supported by the current `FuturesBroker` implementation.

Troubleshooting

- Indicators mismatch. It should match. Keltner Channels and ATR may produce a slightly (<0.1 diff.) different results, this is because on charts like TradingView the indicator values for historical data are computed using the *established* values (all candles are closed already). This is not the case if we are to compute, say, ATR with 5min time frame on 1min data. The tests for indicators are in `src/tests/analyser_test` folder (pytest needed). Feel free to contact me if you encounter such error anyway.
- Weird trade/order dates. It was unclear to me what tz does the sample MNQ data have and I could not identify it simply comparing with the usual trading hours in US timezones. By default the engine assumes it is in UTC.
- `backintime.data.csv.DateNotFound` when executing visualization scripts. Decrease the first trade id in range (e.g. change `--trades 25:45` --> `--trades 20:45`)