Multiple Time Frames¶

Best trading strategies that rely on technical analysis might take into account price action on multiple time frames. This tutorial will show how to do that with *backtesting.py*, offloading most of the work to pandas resampling. It is assumed you're already familiar with basic framework usage.

We will put to the test this long-only, supposed 400%-a-year trading strategy, which uses daily and weekly relative strength index (RSI) values and moving averages (MA).

In practice, one should use functions from an indicator library, such as TA-Lib or Tulipy, but among us, let's introduce the two indicators we'll be using.

In [1]:

```
import pandas as pd def SMA(array, n): """Simple moving average""" return
pd.Series(array).rolling(n).mean() def RSI(array, n): """Relative strength index""" # Approximate; good
enough gain = pd.Series(array).diff() loss = gain.copy() gain[gain < 0] = 0 loss[loss > 0] = 0 rs =
gain.ewm(n).mean() / loss.abs().ewm(n).mean() return 100 - 100 / (1 + rs)
```

The strategy roughly goes like this:

Buy a position when:

- weekly RSI(30) $\geq$ daily RSI(30) $>$ 70
- Close $>$ MA(10) $>$ MA(20) $>$ MA(50) $>$ MA(100)

Close the position when:

- Daily close is more than 2% *below* MA(10)
- 8% fixed stop loss is hit

We need to provide bars data in the *lowest time frame* (i.e. daily) and resample it to any higher time frame (i.e. weekly) that our strategy requires.

In [2]:

```
from backtesting import Strategy, Backtest from backtesting.lib import resample_apply class
System(Strategy): d_rsi = 30 # Daily RSI lookback periods w_rsi = 30 # Weekly level = 70 def init(self):
# Compute moving averages the strategy demands self.ma10 = self.I(SMA, self.data.Close, 10) self.ma20 =
self.I(SMA, self.data.Close, 20) self.ma50 = self.I(SMA, self.data.Close, 50) self.ma100 = self.I(SMA,
self.data.Close, 100) # Compute daily RSI(30) self.daily_rsi = self.I(RSI, self.data.Close, self.d_rsi)
# To construct weekly RSI, we can use `resample_apply()` # helper function from the library
self.weekly_rsi = resample_apply( 'W-FRI', RSI, self.data.Close, self.w_rsi) def next(self): price =
self.data.Close[-1] # If we don't already have a position, and # if all conditions are satisfied, enter
long. if (not self.position and self.daily_rsi[-1] > self.level and self.weekly_rsi[-1] > self.level and
self.weekly_rsi[-1] > self.daily_rsi[-1] and self.ma10[-1] > self.ma20[-1] > self.ma50[-1] >
self.ma100[-1] and price > self.ma10[-1]): # Buy at market price on next open, but do # set 8% fixed
stop loss. self.buy(sl=.92 * price) # If the price closes 2% or more below 10-day MA # close the
position, if any. elif price < .98 * self.ma10[-1]: self.position.close()
```

Loading BokehJS ...

Let's see how our strategy fares replayed on nine years of Google stock data.

In [3]:

```
from backtesting.test import GOOG backtest = Backtest(GOOG, System, commission=.002) backtest.run()
```

Out[3]:

```
Start 2004-08-19 00:00:00 End 2013-03-01 00:00:00 Duration 3116 days 00:00:00 Exposure Time [%] 2.79
Equity Final [$] 9939.03 Equity Peak [$] 10940.25 Commissions [$] 156.64 Return [%] -0.61 Buy & Hold
Return [%] 313.30 Return (Ann.) [%] -0.07 Volatility (Ann.) [%] 4.95 CAGR [%] -0.05 Sharpe Ratio -0.01
Sortino Ratio -0.02 Calmar Ratio -0.01 Alpha [%] -7.35 Beta 0.02 Max. Drawdown [%] -10.23 Avg. Drawdown
[%] -9.54 Max. Drawdown Duration 2653 days 00:00:00 Avg. Drawdown Duration 1410 days 00:00:00 # Trades 4
Win Rate [%] 25.00 Best Trade [%] 9.49 Worst Trade [%] -4.66 Avg. Trade [%] -0.12 Max. Trade Duration 35
days 00:00:00 Avg. Trade Duration 21 days 00:00:00 Profit Factor 1.01 Expectancy [%] 0.03 SQN -0.05
Kelly Criterion -0.02 _strategy System _equity_curve Equ... _trades Size EntryBa... dtype: object
```

Meager four trades in the span of nine years and with zero return? How about if we optimize the parameters a bit?

In [4]:

```
%%time backtest.optimize(d_rsi=range(10, 35, 5), w_rsi=range(10, 35, 5), level=range(30, 80, 10))
```

```
CPU times: user 154 ms, sys: 44.1 ms, total: 199 ms Wall time: 4.48 s
```

Out[4]:

```
Start 2004-08-19 00:00:00 End 2013-03-01 00:00:00 Duration 3116 days 00:00:00 Exposure Time [%] 21.55
Equity Final [$] 21903.22 Equity Peak [$] 22500.98 Commissions [$] 1343.28 Return [%] 119.03 Buy & Hold
Return [%] 313.30 Return (Ann.) [%] 9.63 Volatility (Ann.) [%] 13.10 CAGR [%] 6.55 Sharpe Ratio 0.74
Sortino Ratio 1.23 Calmar Ratio 0.48 Alpha [%] 80.61 Beta 0.12 Max. Drawdown [%] -19.93 Avg. Drawdown
[%] -3.88 Max. Drawdown Duration 779 days 00:00:00 Avg. Drawdown Duration 97 days 00:00:00 # Trades 22
Win Rate [%] 63.64 Best Trade [%] 24.83 Worst Trade [%] -6.50 Avg. Trade [%] 3.40 Max. Trade Duration 63
days 00:00:00 Avg. Trade Duration 30 days 00:00:00 Profit Factor 4.33 Expectancy [%] 3.67 SQN 2.24 Kelly
Criterion 0.48 _strategy System(d_rsi=30,... _equity_curve Equ... _trades Size EntryB... dtype: object
```

In [5]:

```
backtest.plot()
```

Out[5]:

GridPlot(id = 'p1468', …)

Better. While the strategy doesn't perform as well as simple buy & hold, it does so with significantly lower exposure (time in market).

In conclusion, to test strategies on multiple time frames, you need to pass in OHLC data in the lowest time frame, then resample it to higher time frames, apply the indicators, then resample back to the lower time frame, filling in the in-betweens. Which is what the function `backtesting.lib.resample_apply()` does for you.

Learn more by exploring further examples or find more framework options in the full API reference.