Library of Composable Base Strategies¶

This tutorial will show how to reuse composable base trading strategies that are part of *backtesting.py* software distribution. It is, henceforth, assumed you're already familiar with basic package usage.

We'll extend the same moving average cross-over strategy as in Quick Start User Guide, but we'll rewrite it as a vectorized signal strategy and add trailing stop-loss.

Again, we'll use our helper moving average function.

In [1]:

```
from backtesting.test import SMA
```

Loading BokehJS ...

Part of this software distribution is `backtesting.lib` module that contains various reusable utilities for strategy development. Some of those utilities are composable base strategies we can extend and build upon.

We import and extend two of those strategies here:

- `SignalStrategy` which decides upon a single signal vector whether to buy into a position, akin to vectorized backtesting engines, and
- `TrailingStrategy` which automatically trails the current price with a stop-loss order some multiple of average true range (ATR) away.

In [2]:

```
import pandas as pd from backtesting.lib import SignalStrategy, TrailingStrategy class
SmaCross(SignalStrategy, TrailingStrategy): n1 = 10 n2 = 25 def init(self): # In init() and in next() it
is important to call the # super method to properly initialize the parent classes super().init() #
Precompute the two moving averages sma1 = self.I(SMA, self.data.Close, self.n1) sma2 = self.I(SMA,
self.data.Close, self.n2) # Where sma1 crosses sma2 upwards. Diff gives us [-1,0, *1*] signal =
(pd.Series(sma1) > sma2).astype(int).diff().fillna(0) signal = signal.replace(-1, 0) # Upwards/long only
# Use 95% of available liquidity (at the time) on each order. # (Leaving a value of 1. would instead buy
a single share.) entry_size = signal * .95 # Set order entry sizes using the method provided by #
`SignalStrategy`. See the docs. self.set_signal(entry_size=entry_size) # Set trailing stop-loss to 2x
ATR using # the method provided by `TrailingStrategy` self.set_trailing_sl(2)
```

Note, since the strategies in `lib` may require their own intialization and next-tick logic, be sure to **always call `super().init()` and `super().next()` in your overridden methods**.

Let's see how the example strategy fares on historical Google data.

In [3]:

```
from backtesting import Backtest from backtesting.test import GOOG bt = Backtest(GOOG, SmaCross,
commission=.002) bt.run() bt.plot()
```

Out[3]:

GridPlot(id = 'p1398', …)

Notice how managing risk with a trailing stop-loss secures our gains and limits our losses.

For other strategies of the sort, and other reusable utilities in general, see ***backtesting.lib* module reference**.

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