FTX Quant Zone Competition

FTX is a new exchange to me, however once I saw that it had a quant zone. I had to get involved.

Sunday, 19. April 2020 07:45pm he quantzone allows you to define rules and then have them execute upon the markets in real time.

The only problem for me, is that there is no way to back test it.

Luckily for me, Pyalgotrade comes to the rescue. This will allow us to effectively back test our strategies upon the market data and see how they would have performed.

(This is not investment advice purely an intellectual challenge!)

Python Rule Design and Optimisation

Our simple rule will involve just buying btc, using limit orders. (this is not a complex strategy, however its more than adequately illustrates how powerful this approach could be.

As the goal for myself is simply gather more BTC, we will not implement selling simply, optimal placement of limit orers to maximise our BTC balance over 1m windows.

OHLC Gathering

As an example, we will gather the OHLCV for BTC/USD in 1m candles.

We will gather the data directly from FTX, using the fantastic CCXT library as so;

``` import pandas as pd import ccxt import datetime

exchange = ccxt.ftx()

def gather_data(): data = exchange.fetch_ohlcv("BTC/USD") df = pd.DataFrame(data) df.columns = (["Date Time", "Open", "High", "Low", "Close", "Volume"]) def parse_dates(ts): return datetime.datetime.fromtimestamp(ts/1000.0) df["Date Time"] = df["Date Time"].apply(parse_dates) df.to_csv("sampledata.csv")

def main(): gather_data()

if **name** == "**main**": main()

```

Coding the Strategy

Effectively, we want to buy as many BTC as possible, at the best price possible.

So, we will check how much of an offset from the price will give us the best return on aquiring more BTC.

Luckily, Pyalgotrade handles a lot of this for us.

So, we can first do some thing like this;

``` from **future** import print_function

from pyalgotrade import strategy from pyalgotrade.barfeed import quandlfeed, csvfeed from pyalgotrade.technical import ma

class Accumulator(strategy.BacktestingStrategy): def **init**(self, feed, instrument, buy_offset, buy_percent): super(Accumulator, self).**init**(feed, 10000) self.__position = None self.__instrument = instrument # We'll use adjusted close values instead of regular close values.

```
    # self.setUseAdjustedValues(False)
    self.__sma = ma.SMA(feed[instrument].getPriceDataSeries(), 60)
    self.offset = buy_offset
    self.buy_percent = buy_percent

def onEnterOk(self, position):
    execInfo = position.getEntryOrder().getExecutionInfo()
    self.info("BUY at $%.2f" % (execInfo.getPrice()))

def onEnterCanceled(self, position):
    self.__position = None

def onExitOk(self, position):
    execInfo = position.getExitOrder().getExecutionInfo()
    self.info("SELL at $%.2f" % (execInfo.getPrice()))
    self.__position = None

def onExitCanceled(self, position):
    # If the exit was canceled, re-submit it.
    self.__position.exitMarket()

def onBars(self, bars):
    # Wait for enough bars to be available to calculate a SMA.
    # print(bars)
    bar = bars[self.__instrument]
   # self.info(bar.getClose())
   # self.info(self.__sma[-1])

    if self.__sma[-1] is None:
        return

    bar = bars[self.__instrument]
    # If a position was not opened, check if we should enter a
    # long position.
    shares = (self.getBroker().getCash() / bars[self.__instrument].getPrice())

    if self.__position is None:
        if (bar.getPrice() * (1 + self.offset) < self.__sma[-1]):
            # Enter a buy market order. The order is good till canceled.
            self.__position = self.enterLong(self.__instrument, shares, True)
```
```

```
    # Check if we have to exit the position.
    elif not self.__position.exitActive():
        if (bar.getPrice() * (1 - self.offset) > self.__sma[-1]):
            # Enter a buy market order. The order is good till canceled.
            self.__position.exitMarket()


def getSMA(self):
    return self.__sma
```

Effectively, we are looking to buy with a certain amount of our cash balance, when the current price of BTC is a certain % away from the simple moving average.

Backesting

Now we have effectively coded our strategy, it is time to see what it looks like!;

In order to do that, we will run the backtesting code set up as so;

```
# python3 strategy.py

from pyalgotrade import strategy from pyalgotrade.bar import Frequency from pyalgotrade.barfeed import csvfeed from pyalgotrade.stratanalyzer import returns from pyalgotrade.stratanalyzer import trades

from pyalgotrade import plotter

from accumulator_strategy import Accumulator

def main(): feed = csvfeed.GenericBarFeed(frequency=Frequency.MINUTE) feed.addBarsFromCSV("ETH", "sampledata.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = Accumulator(feed, "ETH", buy_offset=0.0024, buy_percent=0.1)
# myStrategy.run()

returnsAnalyzer = returns.Returns()
myStrategy.attachAnalyzer(returnsAnalyzer)
tradesAnalyzer = trades.Trades()
myStrategy.attachAnalyzer(tradesAnalyzer)

# Run the strategy.

plt = plotter.StrategyPlotter(myStrategy)
# Include the SMA in the instrument's subplot to get it displayed along with the closing prices.
plt.getInstrumentSubplot("ETH").addDataSeries("SMA", myStrategy.getSMA())
# Plot the simple returns on each bar.
plt.getOrCreateSubplot("returns").addDataSeries("Simple returns", returnsAnalyzer.getReturns())

# Run the strategy.
myStrategy.run()
myStrategy.info("Final portfolio value: $%.2f" % myStrategy.getResult())

# Plot the strategy.
plt.plot()

print("Final portfolio value: $%.2f" % myStrategy.getResult())

if __name__ == "__main__": main()
```

We can see that it looks pretty good actually!

However, you will notice that we could probably do better with our market entry.

Optimisation

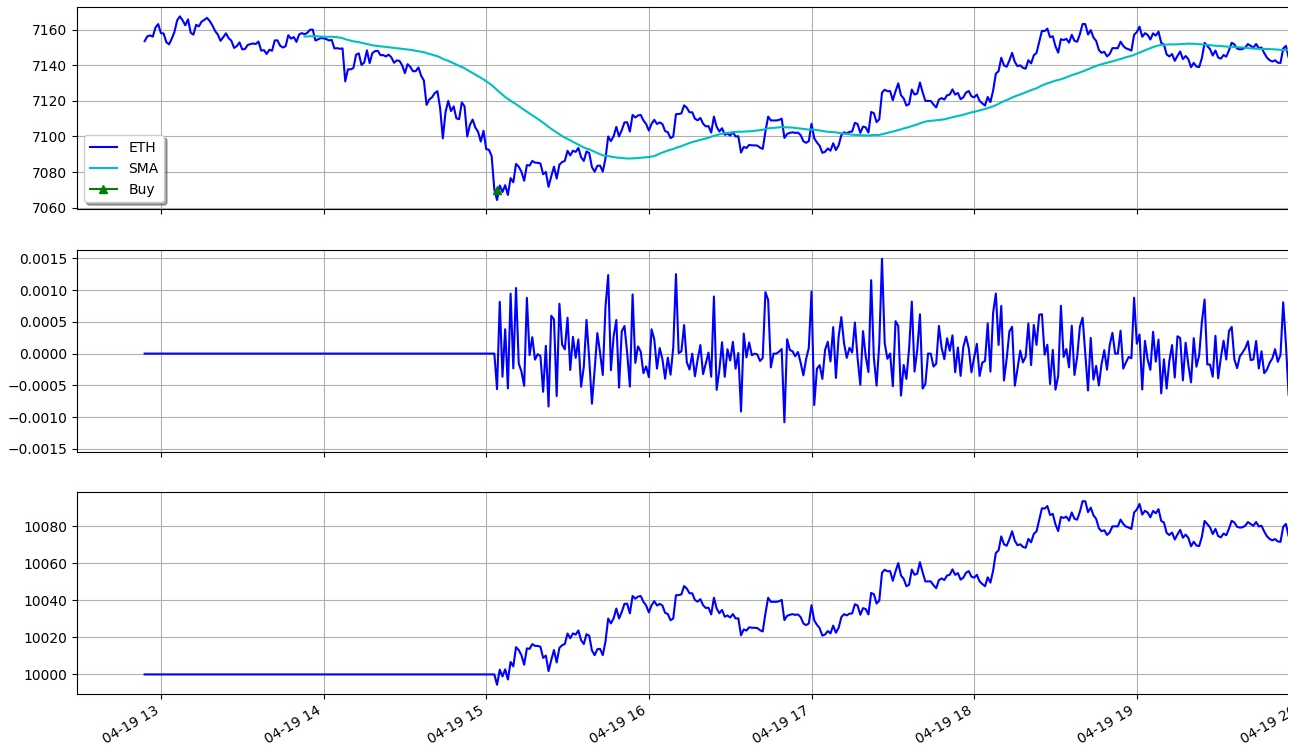
To optimise the strategy, we can again use py algo trade as so;

``` python3 optimiser.py

``` Unfortunately, this will spit out a load of debugging issues, however, if you comb through the output, you will find better parameters as so;

2020-04-19 21:46:46,993 pyalgotrade.optimizer.xmlrpcserver [INFO] Best result so far 10000 with parameters ('BTC', 0.008, 0.49)

Plugging these paramters into out original plotter, we see that;




The best results are obviously buying the very dip...

Now that we have this rule, we can actually go and implement it in FTX as a patient rule to just sit and wait until these conditions are met.

# FTX Quantzone Rule

FTX make this ridiculously easy to do as so;

# References


CCXT

PyAlgoTrade

FTX Exchange