from cloudquant.interfaces import Strategy

import talib

import numpy as np

###############################################################################################

# CQ Lite Basic Strategy.

# This public strategy is an example for running a basic momentum based strategy.

# It looks at the closing price for four consecutive 1 minute bars. If they are increasing

# over the period of 6 minutes then the assumption is that it will continue to increase.

#

# There is logic to check to prevent too much loss (stop) and logic to take profits.

# no position is held for more than 15 minutes

#

# This script does produce an output file.

# Warning: Backtesting over multiple days will cause the file to be overwritten.

#

# Pay attention to the \_\_init\_\_ function to see variable names and to change settings.

#

##############################################################################################

class strategy1(Strategy):

########################################################################################

#

# high level strategy - start/finish

#

########################################################################################

# called when the strategy starts (aka before anything else)

@classmethod

def on\_strategy\_start(cls, md, service, account):

cls.symbol\_list = ["AAPL", "MSFT", "AMZN", "FB", "JNJ", "XOM", "BRK.B", "JPM", "GOOGL", "GOOG", "WFC", "BAC", "T", "PG","GE", "CVX", "PFE", "C", "CMCSA", "V", "UNH", "PM", "VZ", "KO", "HD", "MRK", "PEP", "INTC", "CSCO","DIS", "ORCL", "MO", "BA", "AMGN", "MCD", "IBM", "MA", "MMM", "MDT", "WMT", "ABBV", "CELG", "HON","AVGO", "NVDA", "GILD", "SLB", "BMY", "UTX", "ABT", "AGN", "UNP", "SBUX", "USB", "TXN", "NFLX","GS", "ACN", "CVS", "LLY", "TWX", "QCOM", "UPS", "NKE", "CHTR", "ADBE", "LMT", "BLK", "FDS", "TMO","NEE", "MS", "CB", "WBA", "MDLZ", "CAT", "COST", "PYPL", "LOW", "CL", "AXP", "BIIB", "PNC", "CRM", "MET","DUK", "AMT", "AIG", "BK", "GD", "EOG", "COP", "FDX", "KHC", "SCHW", "AET", "MON", "SPG", "AMAT","DHR", "ANTM", "RTN", "GM", "D", "PRU", "CSX", "ADP", "SO", "OXY", "BDX", "NOC", "SYK", "CI", "TJX","ITW", "ATVI", "KMB", "F", "COF", "MMC", "CME", "CTSH", "VRTX", "JCI", "CCI", "SPGI", "ICE", "REGN","KMI", "PX", "EMR", "BBT", "DE", "ESRX", "DAL", "BSX", "HAL", "EBAY", "PSX", "EA", "AON", "LUV", "ETN","MCK", "TRV", "EXC", "ISRG", "PCG", "ECL", "EQIX", "AEP", "HUM", "NSC", "INTU", "STT", "STZ", "ALL","PLD", "HPQ", "MAR", "GIS", "PSA", "APD", "MU", "AFL", "ZTS", "TGT", "WM", "VLO", "FIS", "BAX", "ADI","FOXA", "ALXN", "HPE", "MPC", "LYB", "TEL", "SRE", "SHW", "GLW", "CCL", "PXD", "STI", "PPG", "LRCX","WDC", "PGR", "FISV", "HCN", "AVB", "WMB", "PPL", "ZBH", "CMI", "SYY", "EIX", "ILMN", "WY", "NWL", "APC","ED", "AAL", "YUM", "EQR", "CBS", "CAH", "MTB", "DLPH", "EW", "VTR", "PCAR", "DFS", "XEL", "ROP", "ADM","SYF", "INCY", "HCA", "IP", "APH", "PEG", "IR", "EL", "ADSK", "MCO", "AMP", "PH", "KR", "MNST", "SWK","ROST", "ROK", "OKE", "TROW", "FCX", "WEC", "UAL", "KEY", "SWKS", "FITB", "NEM", "HIG", "CXO", "VFC","ES", "DTE", "RCL", "A", "EXPE", "SYMC", "CERN", "NTRS", "MCHP", "WLTW", "DG", "FTV", "APA", "NUE","PAYX", "BXP", "OMC", "TSN", "DLR", "MYL", "PFG", "RHT", "CFG", "COL", "EFX", "RF", "ESS", "DVN", "CLX","K", "ORLY", "DPS", "DLTR", "VMC", "TAP", "LH", "MGM", "BEN", "XLNX", "LNC", "O", "KLAC", "ABC", "MTD","MHK", "HCP", "MSI", "HSY", "DGX", "BLL", "RSG", "IVZ", "HSIC", "BBY", "DISH", "WRK", "IDXX", "AWK","INFO", "GPN", "AZO", "XRAY", "HBAN", "ULTA", "CNC", "MLM", "AME", "WHR", "HRS", "TDG", "SJM", "WAT","AEE", "FE", "GGP", "FOX", "LLL", "ETR", "CAG", "NLSN", "VNO", "HST", "L", "VRSK", "CHD", "ADS", "TXT","CMS", "DOV", "NBL", "ALB", "FTI", "HLT", "FAST", "CMA", "NOV", "CTL", "HOLX", "EMN", "KMX", "DHI","CTXS", "VIAB", "CNP", "GPC", "MAS", "XL", "NTAP", "ALGN", "COO", "HAS", "CTAS", "IT", "COG", "MAA","HES", "CBG", "SNPS", "AMD", "ETFC", "CINF", "ARNC", "EQT", "PRGO", "UNM", "RMD", "ARE", "KSU", "ANSS","ALK", "RJF", "LB", "CA", "WYN", "TSS", "EXPD", "LEN", "DRI", "JNPR", "RE", "MKC", "WYNN", "UDR", "AJG","PNR", "IFF", "MRO", "XYL", "SLG", "AMG", "URI", "FBHS", "FMC", "PKG", "DVA", "CPB", "TIF", "DRE", "UHS","EXR", "LKQ", "PNW", "CMG", "STX", "REG", "FRT", "VAR", "LNT", "CHRW", "ZION", "XEC", "COTY", "SCG","PVH", "BWA", "WU", "HRL", "AYI", "GT", "VRSN", "GWW", "SEE", "QRVO", "CBOE", "IRM", "SNA", "LUK", "IPG","NI", "HOG", "HBI", "NDAQ", "TMK", "MOS", "KIM", "FFIV", "SNI", "AVY", "AAP", "AOS", "JBHT", "AKAM","NRG", "ALLE", "AES", "M", "MAT", "MAC", "PKI", "PHM", "LEG", "AIV", "KSS", "EVHC", "TSCO", "CF", "SRCL","XRX", "FLR", "JEC", "HRB", "FL", "FLS", "PBCT", "GRMN", "HP", "AIZ", "KORS", "RHI", "NFX", "GPS","NWSA", "DISCK", "JWN", "FLIR", "PWR", "CSRA", "TRIP", "SIG", "DISCA", "NAVI", "RL", "RRC", "CHK", "UAA","PDCO", "UA", "AN", "NWS"]

cls.symbol\_list = list(set(cls.symbol\_list)) #convert the list to a set, which removes dupes - dont want dupes, then back to a list.

#cls.buyingpower = 3000000

pass

# called when the strategy finish (aka after everything else has stopped)

@classmethod

def on\_strategy\_finish(cls, md, service, account):

pass

########################################################################################

#

# symbol universe

#

########################################################################################

# note that this doesn't start with "self" because it's a @classmethod

@classmethod

def is\_symbol\_qualified(cls, symbol, md, service, account):

return symbol == 'SPY'

# used to load other symbols data not in is\_symbol\_qualified(). Only used in backtesting

@classmethod

def backtesting\_extra\_symbols(cls, symbol, md, service, account):

return symbol in cls.symbol\_list

########################################################################################

#

# start/finish instance related methods

#

########################################################################################

# used to pass external parameters for each instance (same for values for every instance)

def \_\_init\_\_(self): # , \*\*params - if passing in parameters is desired

self.IsPositionOn = False # do we have a position on?

self.LongQty = 0 # long quantity of our position

self.long\_entry\_price = 0 # estimated price of our position

#self.TargetProfit = 0.06 # target profit

self.HowLongHeld = 0 # How Many Bars the position has been held.

self.filename = "output.csv"

self.sOutString = ""

# called at the beginning of each instance

def on\_start(self, md, order, service, account):

# print "OnStart {0}\t{1}\n".format(service.time\_to\_string(service.system\_time), self.symbol)

# The model requires that we have at least X minutes of bar data prior

# to checking to see if a bear price flip has occurred. Therefore we

# need a variable to track this start time.

self.model\_start = md.market\_open\_time + service.time\_interval(minutes=5)

# if running with an instance per symbol, call when an instance is terminated

def on\_finish(self, md, order, service, account):

pass

########################################################################################

#

# timer method

#

########################################################################################

# called in timer event is received

def on\_timer(self, event, md, order, service, account):

pass

########################################################################################

#

# market data related methods

#

########################################################################################

# called every minute before the first on\_trade of every new minute, or 5 seconds after a new minute starts

def on\_minute\_bar(self, event, md, order, service, account, bar):

#

# don't want to initiate any long positions in the last 5 minutes of the trading day

# as we won't likely have time to trade out of the position for a profit using 1 minute

# bar data.

#

if service.system\_time < md.market\_close\_time - service.time\_interval(minutes=5, seconds=1):

#

# If a position is on we want to check to see if we should take a profit or trade out

# of a losing position.

#

if self.IsPositionOn == True:

# there is a position on, therefore we want to check to see if

# we should realize a profit or stop a loss

bar\_0=bar.minute(start=-200, include\_empty=True)

if len(bar\_0) > 200:

close = bar\_1.close

low=bar\_1.low

vol=bar\_1.volume

vwap=bar\_1.vwap

ema = talib.EMA(close, timeperiod=50)

ema1=ema.iloc[-1]

vma1 = talib.MA(volume, timeperiod=60, matype=0)

vma2 = talib.MA(volume, timeperiod=200, matype=0)

vma3 = talib.MA(volume, timeperiod=100, matype=0)

if low.iloc[-1]<vawp.iloc[-1] or low.iloc[-1]<ema1 or \

vma1.iloc[-1]<vma2.iloc[-1] or vma3.iloc[-1]<vma3.iloc[-2]:

# target profit realized, we want to get out of the position.

print "\texit position now {0}\t{1} entry px at {2} low of bar = {3}\n".format(

service.time\_to\_string(service.system\_time), self.symbol, self.long\_entry\_price, bv\_0[0])

self.IsPositionOn = False

# send order; use a variable to accept the order\_id that order.algo\_buy returns

sell\_order\_id = order.algo\_sell(self.symbol, "market", intent="exit")

self.sOutString = "{0},sellSignal,{1},{2}".format(service.time\_to\_string(service.system\_time),

self.symbol, bv\_0[0])

print self.sOutString

service.write\_file(self.filename, self.sOutString)

# we have to have at least 5 minutes of bar data before we can start checking to see if we want to buy.

else: # position isn't on, therefore check to see if we should buy.

if service.system\_time>service.time(9, 31) and service.system\_time< service.time(11, 30):

bar\_1=bar.minute(start=-200, include\_empty=True)

close = bar\_1.close

low=bar\_1.low

op=bar\_1.open

vol=bar\_1.volume

vwap=bar\_1.vwap

ema = talib.EMA(close, timeperiod=40)

ema1=ema.iloc[-1]

sma=talib.SMA(close, timeperiod=100)

sma1=sma.iloc[-1]

vma1 = talib.MA(volume, timeperiod=60, matype=0)

vma2 = talib.MA(volume, timeperiod=200, matype=0)

vma3 = talib.MA(volume, timeperiod=100, matype=0)

# Check requirements

if len(bar\_1.close) == 200:

if close.iloc[-1]>op.iloc[-1] and low.iloc[-1]>vawp.iloc[-1] and vawp.iloc[-1]>ema1 and \

ema1>sma1 and (vma1.iloc[-1]>vma2.iloc[-1] or vma3.iloc[-1]>vma3.iloc[-2]):

self.IsPositionOn = True

n=100

# send order; use a variable to accept the order\_id that order.algo\_buy returns

order\_id = order.algo\_buy(self.symbol, "market", intent="init", order\_quantity=n)

print("100 shares of " + self.symbol + " have been ordered!\n")

########################################################################

# since on\_trade and on\_fill aren't called in lite, use the close bar px

# as our entry px approximate. We will use this price to check for

# profit taking or for stop loss.

self.long\_entry\_price = close[199];

self.sOutString = "{0},buySignal,{1}".format(

service.time\_to\_string(service.system\_time), self.symbol, close[199])

print self.sOutString

service.write\_file(self.filename, self.sOutString)

else:

####################################################################

# close out of our position at the end of the day because we don't

# want to carry overnight risk.

if self.IsPositionOn == True:

sell\_order\_id = order.algo\_sell(self.symbol, "market", intent="exit")

self.IsPositionOn = False

print "\tExit EOD position now {0} for {1}\n".format(service.time\_to\_string(service.system\_time),

self.symbol)