Trend Following with Pullback Entry in Currencies

IMPORTANT UPDATE

- We are removing the min_days_held requirement. This requirement makes it very difficult if not impossible to vectorize. So now the only requirement for changing positions is that there is a new market with a stronger trend, and that market is in a pullback.
- It is important, however, to learn how to implement a min_days_held requirement using Pandas. This is a common aspect of many trading strategies because it reduces "churning" and thereby minimizes transaction costs.
- The test will have a min_days_held requirement, so you are encouraged to practice implementing this with Pandas.

The Strategy

- In this project you will alternate positions in currencies based on which currency has the strongest trend. You will always have a position in exactly one market at a given time – the others will have a position of zero. A long position will be taken if the strongest trend is an uptrend, and a short position will be taken if the strongest trend is a downtrend.
- You will time the entry by waiting for a pullback, or a slight reversal, in the trend. To minimize transaction costs and avoid churning, you will also wait at least some minimum numbers of days between trades.
- There are three parameters: look_back, num_stdevs, and min_days_held.
- First, compute the standard deviation of returns over the past look_back number of trading days. Call this Vol. You can include the closing price of the current trading day.
    - Note – see below for instruction on the correct way to calculate returns for futures contracts. Also, returns are computed with closing prices – you will not need Open, High, or Low prices.
- Next, compute the percent change (same calculation as return) from look_back trading days ago to today. Call this Pct Change.
- Compute the trend score by dividing Pct Change by Vol.
- Initially, take a position in the market with the largest magnitude of the trend score. The sign of the position is the same as the sign of the trend score. The size of the position is discussed below.
- Hold this position until the following three criteria are all met:
    - A new market has a stronger trend
    - At least min_days_held has passed since the last trade
    - There is a pullback in the new market with the stronger trend
- Pullback will be defined as there being a one-day return in the direction opposite the trend of at least num_stdevs multiplied by Vol.

- Finally, when a position is taken, the size of the position is determined by the below formula. This position size does not change until you liquidate your position in that market.
- Size_i = 10 * (1 / (Vol_i * Mult_i)) / sum_j(1 / (Vol_j * Mult_j)). This is the size for market i. Sum_j is the sum over all markets, so j=1,…,6. Mult_i is the multiplier in the contract specs file for market i and Vol_i is the Vol calculation for market i.
- This is called "Even Vol" position sizing. You want each market to have roughly an equal contribution to your risk. And so you are setting the position size such that the expected volatility for each market is the same for all markets. I explain further in my Portfolio Optimization video on YouTube. If you included correlations as well it would be "Risk Parity", but even vol should be complex enough for this project.

Your Task

- Use the parameters look_back=252, num_stdevs=0.5, min_days_held=5, and market_list=['EC', 'BP', 'JY', 'SF', 'CD', 'AD']. The tickers correspond to:
    - Euro (EC)
    - British Pound (BP)
    - Japanese Yen (JY)
    - Swiss Franc (SF)
    - Canadian Dollar (CD)
    - Australian Dollar (AD)
- Your first task is to load the csv's into pandas and combine the dataframe for each market into one dataframe consisting of the back-adjusted close and spliced close for each market (see Returns Calculation section below), and with Date set as the index. Use an outer join to combine them and forward fill prices for any resulting NaN's. The pandas functions pd.merge, pd.concat, and Series.ffill will be helpful here.
- Then you'll code up your vectorized trade logic function (see below).
- Run the strategy from 2007 onward. The look back period should be used on data just prior to the start of 2007, so that there could be a trade on the first trading day of 2007.
- Save your dataframe as 'output.csv'.
- Have your script print the sum of the 'Total P&L' column.

Trade Logic Function

- The bulk of the coding will be within the trade_logic function. It should read exactly trade_logic(df, specs, look_back, num_stdevs, min_days_held, market_list). df refers to the merged dataframe, specs refers to the contract_specs.csv that should be read into pandas as a dataframe.
- The function should return a dataframe with 'Date' set as the index and with the following columns for each market M: '{M} Price', '{M} Return', '{M} PctChng', '{M}

Vol', '{M} Trend', '{M} Position', '{M} P&L'. For price, have that be the back-adjusted closing price. Finally, include a column 'Total P&L' where you sum the P&L's for each currency. You're free to include other columns as well if you like.

- For each market, the P&L for today is tomorrow's back-adjusted close minus today's back-adjusted close, multiplied by the Multiplier in contract_specs.csv and multiplied by our position. Finally, transaction costs are subtracted whenever we make a trade, meaning our position changes, as described below in the Transaction Costs section.
- For example, if our EC Position is +2, and tomorrow's close is $0.0010 higher than today's close, then our EC P&L is 2 * $0.0010 * 125,000 = $250. If our AD Position is -3, and tomorrow's close is $0.0020 lower than today's close, then our AD P&L is -3 * -$0.0020 * 100,000 = $600.

Vectorization of Trade Logic Function

- After a trade logic function has been verified, we will run the function tens to hundreds of thousands of times using different combinations of parameters. It is important not just to write the function accurately, but also efficiently.
- For loops in Python are extremely slow compared to other languages. Looping through each day in the dataframe is very slow. Using pandas and numpy vectorized operations speed things up dramatically.
- This strategy is nearly completely vectorizable. As such, **there should be no for loops present in your code**. The one exception is that your code can loop through the strings in market_list. For those not familiar with pandas and numpy, this will probably be the most difficult part of the project. Below are some very helpful pandas as numpy functions to get you started:
    - Pandas: Series.rolling, Series.ffill, Series.shift
    - Numpy: numpy.where (extremely useful for vectorizing conditional statements), numpy.nan (then do Series.ffill)
- I would strongly recommend coding the trade_logic function in a for loop to get started and make sure you understand the algorithm and that it is doing what it's supposed to be doing. Then recode it using just vectorized operations. You can compare the output to check your work.

Transaction Costs

- In any trading strategy you **must** include transaction costs.
- For this strategy we will assume one tick of slippage for each contract traded. This means every trade we make we automatically start out at a loss equal to the market's tick_size multiplied by its multiplier. For example, every trade in EC incurs a loss equal to $0.00005 * 125,000 = $6.25.
- In addition to slippage, also include exchange fees, commissions, and NFA fees in your transaction costs calculation.

- For clarity, the total transaction cost is the sum of the slippage, exchange fees, commissions, and NFA fees. We incur this transaction cost every time we make a trade.
- We multiply this transaction cost by the change in our position. It's okay if the change in the position is not an integer.
- The details of these fees, multipliers, and tick sizes are all in the contract_specs.csv file.

Returns Calculation

- Calculating returns for futures contracts is not as simple as calculating returns for stocks. For stocks, it is simply the new price minus the old price, all divided by the old price. For futures this does not work so easily because of the fact that there is no one continuous future, but rather multiple different futures contracts with different expiries.
- For example, if one futures contract expired and you switch to the next one, there could be a gap between the two, so if you naively calculated percent change as above, it would be showing a large change even if the market itself didn't even move.
- So you create "back-adjusted" continuous timeseries where all of these gaps are removed. But now the price level is wrong because you've subtracted the gap. So you can't compute percent changes on this back-adjusted timeseries either because while the change in price will always be correct, the price level you are dividing by is wrong.
- The solution is to use both the back-adjusted timeseries and the "spliced" timeseries (where all the months are concatenated together, keeping the price level correct but leaving in the erroneous gaps). The back-adjusted has correct price changes, and the spliced has correct price levels, so the correct percent change formula is the back-adjusted price today minus the back-adjusted price yesterday all divided by the spliced price yesterday.
- The back-adjusted timeseries end in 'CCB', and the spliced timeseries end in 'CCS' (for example, there is 'EC_CCS.csv' and 'EC_CCB.csv'). The column names are Date, Open, High, Low, Close, Volume, Open Interest.
- More information on back-adjusted and spliced futures timeseries here: https://www.premiumdata.net/support/futurescontinuous.php

Submission

- You have one week to complete the project. Email me a python script firstname_lastname.py with your solution.

How I Plan to Evaluate This

- **The number one priority is that the output of the code is 100% accurate.**

- If you are coding up a strategy, and there is some edge-case you missed in your algorithm, and we send the trade_logic function through the backtester and optimizer and into the portfolio, then we could lose money live trading a strategy that doesn't actually have any backtested support. Garbage in, garbage out. We will do every possible check we can before actually live trading the strategy in order to minimize this risk, but even if we catch it near the end of the backtesting process that is still a waste of time and resources. So accuracy is of paramount importance. Your output should **exactly** match my output.
- The vectorization of the trade_logic function is also very important, but it is definitely a secondary consideration. A submission that is vectorized but has inaccuracies will be judged more harshly than a submission that is not vectorized but is 100% accurate.

The Exam

- Following successful completion of the project, we will move on to an onsite interview consisting of a short (20-30 minute) behavioral interview and an exam.
- The exam will be exactly the same format as this assignment, but with an easier strategy to implement. It will be an hour and a half long, but if you really understood every part of the first assignment, you should be able to code it up in the first 30 minutes. The next 60 minutes is there for debugging and testing the algorithm. The exam will be graded the same way the assignment is – accuracy is the number one priority.
- You can bring anything you like to the exam, but you will not have access to the internet. If you're new to Pandas, it might be worth downloading the documentation to your laptop.
- My office is located downtown at 401 N Michigan Ave, Suite 1255. Sign in with the front desk and then head on up.