Simple Portfolio Simulation with Python

This article walks through a buy/sell simulation of a single stock. Basically, we go through a constant time period(e.g. we check X days worth of daily prices at a time), and make a buy/sell/no_change decision depending on the movement of the prices observed for that time period. If we notice an uptrend of X days, we purchase 100 stocks and vice versa for selling. For this exercise, an 'uptrend' is constituted as higher highs and higher lows in price; a downtrend is lower lows and lower highs. The price can go down in an uptrend as long as that downturn is higher than any previous downturns. Once we go through the entire price list, we calculate the profit/loss total by going through 'port', short for portfolio. Here is the function.

```
def buy_sell_sim1(AAPL, days, time_length, funds):
    inv_value = AAPL[0]*10000
    profit = 0
    day1 = 0
    funds -= inv_value
    port = {AAPL[0]: 10000}
    day_list = []

    while funds > 0 and days < (len(AAPL)-time_length):
        if find_uptrend(AAPL[day1:days]):
            day_list.append(days)
            inv_value += AAPL[days]*100
            port[AAPL[days]] = 100
            funds -= AAPL[days]*100
            day1 += time_length
            days += time_length

        if find_downtrend(AAPL[day1:days]):

            inv_value -+ AAPL[days]*100
            funds += AAPL[days]*100
            port[AAPL[0]] -= 100
            profit += (AAPL[days]-AAPL[0])*100
            day1 += time_length
            days += time_length
        else:

            day1 += time_length
            days += time_length

    for i in list(port.keys()):
        profit += (AAPL[day1] - i)*port[i]


    return profit, port, day_list
```

The parameters for the 'buy_sell_sim()' function are 'AAPL', which is a list of daily stock prices, 'days' and 'time_length', which are the number of days we are looking at at a time and considering for trend criteria(e.g. if days and time_length are 6, then we are looking for uptrend and downtrend periods of 6 days), and 'funds', which is the amount of cash we have available for investing in Apple. The 'port' variable we initially created is a dictionary that maps a price to the quantity of stock purchased at that price.

The 'while' loop makes sure we do not buy more stocks if we do not have the necessary funds to do so and that we do not look for trends outside of the length of our price list. The while loop exits once we deplete the funds or go through the entire price list.

This is a sample output of the function.

```
>>> buy_sell_sim1(AAPL_close[200:600], 5, 5, 1000000)
(343742.0, {64.27: 4600, 59.31: 100, 60.92: 100, 67.08: 100, 72.56: 100, 69.89:
100, 71.26: 100, 76.22: 100, 80.58: 100, 77.26: 100, 75.59: 100, 75.91: 100, 74.
99: 100, 84.64: 100, 86.64: 100, 92.29: 100, 94.03: 100, 97.67: 100, 97.98: 100,
 102.5: 100, 101.63: 100, 99.81: 100, 105.11: 100, 108.83: 100, 118.62: 100}, [2
5, 35, 45, 55, 80, 95, 125, 145, 160, 175, 190, 225, 235, 245, 260, 275, 290, 30
5, 315, 325, 345, 355, 365, 375])
>>>
```

If we follow the flow of execution, we initially look at the first time interval of prices '[day1:days]'. 'day1' always begins with 0 and 'days' is the time length we want, so the first iteration of the while loop looks at 'AAPL_close[0:5]'. If this interval fits the criteria for an uptrend, we

'buy' 100 stocks of the last day of prices in the trend by adding the price of AAPL_close[5] and quantity 100 to the dictionary 'port'. In addition to updating the portfolio, we add 'days' to the list 'day_list' to keep track of the uptrend days, and subtract the necessary amount from our 'funds' variable since we just made a purchase. (The code I wrote for 'find_uptrend()' is in my other LinkedIn article 'Having fun with Apple stock data' if you are interested).

If the interval constitutes a downtrend, we 'sell' Apple by subtracting 100 stocks from the day 0 price in our portfolio 'port'. We then calculate if there was a loss or profit from the transaction by using the code shown below and by adding it to the total profit variable. Remember that 'days' represents the price apple is *today*.

```
profit += (AAPL[days]-AAPL[0])*100
```

If the interval of daily prices is neither a downtrend nor uptrend, we add the value of 'time_length' to 'day1' and 'days', which simply shifts the interval of time by 'time_length', which in this case is 5. Keep in mind that we have to do this step for the conditional statements of uptrend and downtrend as well; this way the while loop can continue to iterate over multiple intervals.

Finally, the function returns a tuple of size 3. The first item is the profit of our portfolio after 400 days. The second item is the portfolio of

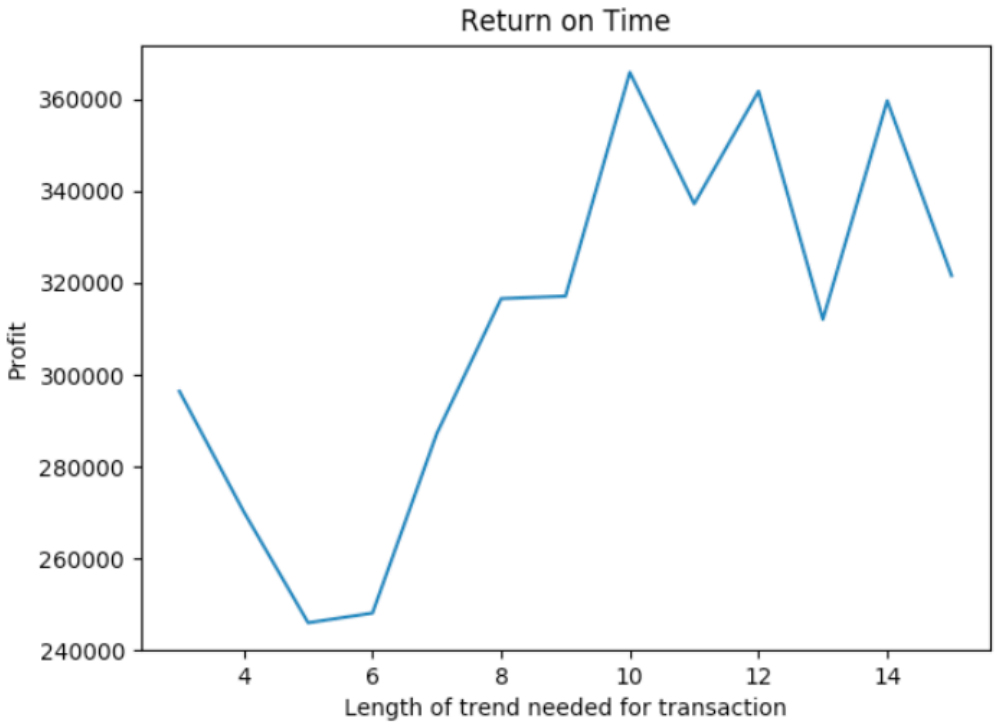
Apple stocks themselves. The keys of the portfolio/dictionary are the bought prices and the values are the quantity of Apple stock we have for that given price. Having a dictionary of price-quantity pairs made calculating the profit easy. We just subtract the difference between bought price and today's price, and multiply by the quantity of stock we have for that given price. The last item of the tuple is a list that shows the end-day mark for an uptrend. This list will prove helpful for plotting the uptrends of the price data.

It would be interesting to see how profit changes when we change the length of a trend needed for a transaction. This function plots a line graph that shows how profit changes with changes in trend-time needed for a buy or sell. Our trend criteria goes from 3 days to 15 days.

```
def return_on_time(AAPL, funds):
    profit = []
    time = list(range(3, 16))
    for i in time:
        profit.append(buy_sell_sim2(AAPL, i, i, funds))
    plt.plot(time, profit)
    plt.show()
```

'buy_sell_sim2()' is exactly the same as the 'buy_sell_sim1()' function used before except the only output is the end profit. We don't need the portfolio nor trend days to consider how profit changes with trend-length criteria for transactions.

This is the output for the list of daily prices we used before. Our funds amount is again $1,000,000.



This graph suggests that, in general, we should wait for a trend period of at least 10 days before making a buy/sell decision; it is better to wait until a trend is solidified through more time. This simulation doesn't consider the transaction costs associated with buying and selling, however, which can take a considerable toll on profit with high activity trading like this. Based from this graph, not only would making moves on shorter trends provide less profit, but it would also rack up more transactional cost because we are buying and selling more frequently.

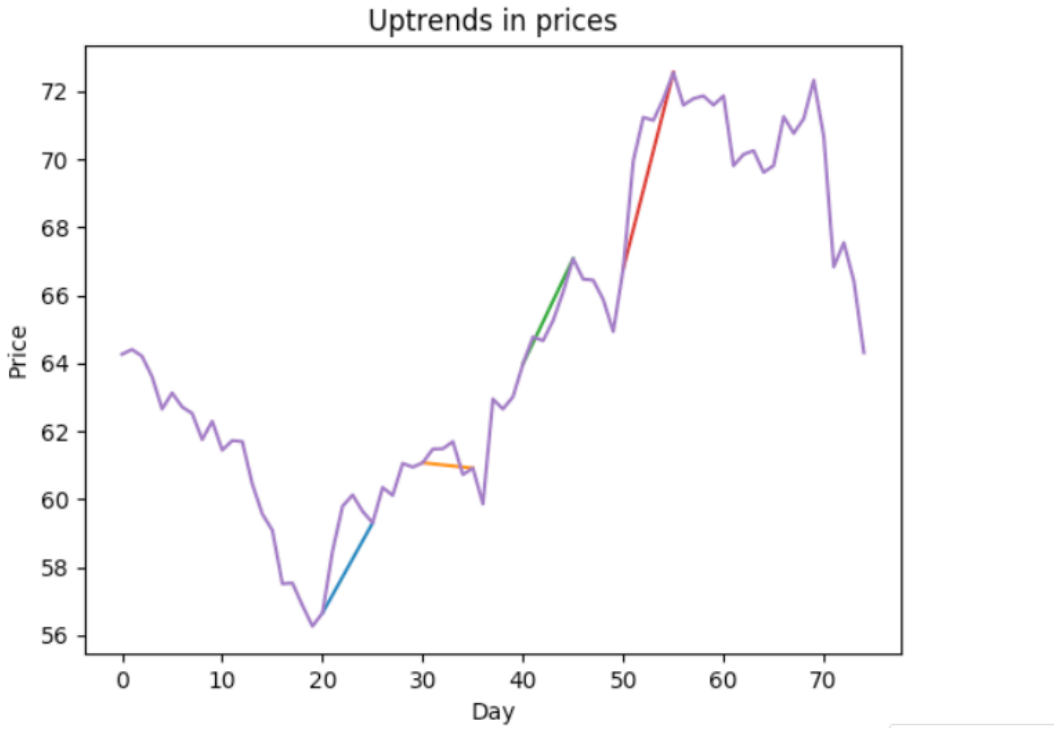
Lastly, to finish up the article, we plot all the uptrends found by the function. Here is the function and its usage.

```
def plot_uptrends(AAPL, days, time_length, funds):
    trend_days = buy_sell_sim3(AAPL, days, time_length, funds)
    trend_ends = []
    for i in trend_days:
        temp = []
        temp.append(i-5)
        temp.append(i)
        trend_ends.append(temp)

    for i in trend_ends:
        plt.plot(i, [AAPL[i[0]], AAPL[i[1]]])
    plt.plot(AAPL)
    plt.show()


plot_uptrends(AAPL_close[200:275], 5, 5, 1000000)
```

We use another 'buy_sell_sim()' function that is a replica of the first one used in the article, the only difference being that the only output of the function is the 'days_list'. This function simply plots the first and last day of a trend against the general prices of the stock. We use only 75 days of prices in this example because it gets harder to see the trends with very long periods of time.



In conclusion, Python is a wonderful tool for examining markets and for coming up with fun ideas to test. I look forward to learning all the things this language can provide.

Best,
Charlie Carrera
Dartmouth College