Published in Towards Data Science

Jose Manu (CodingFun) Follow
Mar 8, 2020 · 6 min read · Listen

Backtesting with Python

Backtesting Moving Averages

In this post, we will perform backtesting with Python on a simple moving average (MA) strategy. In one of my latest posts, I showed how to compute and plot a moving average strategy using Python. Now, we will learn to simulate how the moving average strategy performs over the last few months by backtesting our algorithm.

Let's first quickly recap what we buil ost. Our model was simple, we built a script to calculate and plot a short moving average (20 days) and long moving average (250 days).

I recommend you have a look at this post to learn more in detail about moving averages and how to build the Python script. In this post, I will only post the code to get the moving averages and the stock prices of the selected stock:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt


def stockpriceanalysis(stock):
    stockprices =
requests.get(f"https://financialmodelingprep.com/api/v3/historical-
price-full/{stock}?serietype=line")
    stockprices = stockprices.json()

#Parse the API response and select only last 1200 days of prices
    stockprices = stockprices['historical'][-1200:]

#Convert from dict to pandas datafram

    stockprices = pd.DataFrame.from_dict(stockprices)
    stockprices = stockprices.set_index('date')
    #20 days to represent the 22 trading days in a month
    stockprices['20d'] = stockprices['close'].rolling(20).mean()
    stockprices['250d'] = stockprices['close'].rolling(250).mean()

stockpriceanalysis('aapl')
```

Backtesting Strategy

To perform the backtesting we will:

- Go long on 100 stocks (i.e. buy 100 stocks), when the **short term moving average crosses above the long term moving average**. This is known as golden cross.

It is a very simple strategy. We will have daily close prices for the selected stock. The strategy could also be used with minutes or hourly data but I will keep it simple and perform the backtesting based on daily data.

Since I do not expect to have many entry points, that is when we buy the stocks, I will ignore the transaction costs for simplicity.

Remember from our previous post, that if we run the script by passing the name of the stock to analyse as an argument, we will get a Pandas DataFrame called ***stockprices*** containing the closing price and moving averages from the last 1,200 days.

```
stockpriceanalysis('aapl')

#Outcome stored in a DataFrame call stockprices
                close    20d              250d
date
2020-02-07       320.03   316.7825         224.76192
2020-02-10       321.55   317.3435         225.36456
2020-02-11       319.61   317.4760         225.96228
2020-02-12       327.20   318.2020         226.58788
```

Backtesting Strategy in Python

To build our backtesting strategy, we will start by creating a list which will contain the profit for each of our long positions.

First (1), we create a new column that will contain *True* for all data points in the data frame where the 20 days moving average cross above the 250 days moving average.

Of course, we are only interested in the first or second day when the crossover happens (i.e. 20 days MA goes over 250 days MA). Therefore, we locate the first or second date (rows) where the crossover happens (2).

As well stated in this article, we will use the two-day rule (ie we start the trade only after it is confirmed by one more day's closing), and will keep the date as the entry point only if

This approach will help us to avoid daily trading noise fluctuations. When this happens, we will have the entry points in the column *firstbuy* where the value equals to True:

```
#Start longposiiton list

longpositions = []
# (1)
stockprices['buy'] =(stockprices['20d'] > stockprices['250d'])
# (2)
stockprices['firstbuy'] =   ((stockprices['buy'] == True) &
(stockprices['buy'].shift(2) == False)& (stockprices['buy'].shift(1)
== True))

# (3) identify the buying points
buyingpoints = np.where(stockprices['firstbuy'] == True)
print(buyingpoints)

#Outcome
(array([ 307,  970, 1026]),)
```

The rule *(stockprices['buy'].shift(2) == False)*, helps us to find out the first date after the crossover has happened.

Now we have in the variable *buyingpoints* (3), the dates where we should enter the market with our long strategy.

Each of the elements in the array *buyingpoints* represent the row where we need to go long. Therefore, we can loop through them to get the close price and buy 100 stocks (4).

Then, we keep the stocks for 20 days (5) and sell the 100 stocks at +20 days close price.

```
#(4)
for item in buyingpoints[0]:
    #close price of the stock when 20MA crosses 250 MA
    close_price = stockprices.iloc[item].close

    #Enter long position by buying 100 stocks
```

```
#Sell 20 days later:
sell = close_price*100

# (6) Calculate profit
profit = sell - long

longpositionsprofit.append(profit)
```

Finally, we calculate the profit and add the result of the strategy to the *longpositionprofit* array (6).

Result of our Strategy

To find out how we did with our strategy, we can print out the long position profit list and calculate the sum:

```
print(longpositionsprofit)
#outcome
(array([ 307,  970, 1026]),)

print(sum(longpositionsprofit))
#outcome
2100.0
```

Great, our backtesting strategy for Apple, show us that over 1,200 days, we entered a long position and sell after 20 days a total of three times.

On the first occasion, we got a profit from $307, on the second occasion, $970 and in the last long position, we amounted to a profit of $1,026. That makes a total of $2,100.

Not bad at all. But what if we just had bought the stock 1,200 days ago and keep until today? We can easily calculate the profit of buying and holding by getting the last available price and the first available price in our *stockprices* DataFrame.


```
#Outcome
15906.999999999996
```

Interestingly, by just holding the stock for 1,200 days, our profit would have been $15,906 plus the annual dividends. Much higher than if we had followed the moving average strategy.

Wrapping Up

We have used a simple strategy of buying the stock when the 20 days MA crosses above the 250 days MA. Then, we kept the stock for 20 days before selling it.

There are other strategies that we may have followed. For instance, we could have bought the stocks when the moving average Crossover took place and kept the stock until the end. Or, we could have just sold the stock if the 250 days moving average crosses below the 20 days moving average.

I will let you know to play around and test these other strategies. Happy to get your feedback on my Twitter account.

See below the whole Python script for backtesting moving average strategies for any company. Just replace ***Apple*** by any other company stockpriceanalysis('aapl')

```
#Moving Averages and backtesting
import requests
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


def stockpriceanalysis(stock):
    stockprices =
requests.get(f"https://financialmodelingprep.com/api/v3/historical-
price-full/{stock}?serietype=line")
    stockprices = stockprices.json()
```


```
    stockprices = pd.DataFrame.from_dict(stockprices)
    stockprices = stockprices.set_index('date')
    #20 days to represent the 22 trading days in a month
    stockprices['20d'] = stockprices['close'].rolling(20).mean()
    stockprices['250d'] = stockprices['close'].rolling(250).mean()

    ###STARTING BACKTESTING STRATEGY
    #Start longposiiton list
    longpositionsprofit = []

    shortpositions = []

    stockprices['buy'] =(stockprices['20d'] > stockprices['250d'])

    #We find the first True since it is first day when the line has
crossed. Also, ensure that we have at least two days where sticoprices
    stockprices['firstbuy'] =   ((stockprices['buy'] == True) &
(stockprices['buy'].shift(2) == False)& (stockprices['buy'].shift(1)
== True))


    buyingpoints = np.where(stockprices['firstbuy'] == True)

    for item in buyingpoints[0]:
        #close price of the stock when MA crosses
        close_price = stockprices.iloc[item].close
        #Enter long position
        long = close_price*100
        sellday = item + 20
        close_price = stockprices.iloc[sellday].close
        #Sell 20 days later:
        sell = close_price*100

        #Calculate profti
        profit = sell - long

        longpositionsprofit.append(profit)

    #Result of Moving Average Strategy of going long
    print(longpositionsprofit )
    print(str(sum(longpositionsprofit)) + ' is the profit of Moving
Averag Strategy')

    #Result of just holding the stock over 1200 days
    firstdayprice = stockprices.iloc[0].close
    lastdayprice = stockprices.iloc[-1].close
```


```
#Change apple for the ticker of any company you want to backtest
stockpriceanalysis('AAPL')
```

Originally published at https://codingandfun.com on March 8, 2020.