MC2-Project-1

From Quantitative Analysis Software Courses

Contents

Overview

In this project you will create a market simulator that accepts trading orders and keeps track of a portfolio's value over time and then assesses the performance of that portfolio.

Updates & FAQs

FAQs:

- Q: What if the portfolio becomes levered after the trades have been entered? A: It is OK if the trades are entered and then later, due to stock price changes, leverage exceeds 2.0.
- Q: Should I allow a partial order to be filled so it gets just right up to 2.0? A: No reject the order entirely.
- Q: What if the portfolio is levered at 2.0 already, should I accept orders that reduce leverage? A: Yes.

- Q: How can I be sure that my function is returning a DataFrame? Q: Check the type of the returned item foo with `type(foo)`

- Q: What if multiple orders arrive on the same day and executing all of them would cause leverage to exceed 2.0? A: This won't happen in our test cases. But if you're still worried, you should reject all orders for that day.

Template

Instructions:

- Download **`mc2_p1.zip`**, unzip inside `ml4t/`
- Implement the `compute_portvals()` function in the file `mc2_p1/marketsim.py`.
- To execute, run **`python marketsim.py`** from `mc2_p1/` directory.

Part 1: Basic simulator (95%)

Your job is to implement your market simulator as a function, `compute_portvals()` that returns a dataframe with one column. You should implement it within the file `marketsim.py`. It should adhere to the following API:

```
def compute_portvals(orders_file = "./orders/orders.csv", start_val = 1000000):
    # TODO: Your code here
    return portvals
```

The start date and end date of the simulation are the first and last dates with orders in the `orders_file`. The `orders_file` argument is the name of a file from which to read orders, and `start_val` is the starting value of the portfolio (initial cash available). Return the result (`portvals`) as a single-column `pandas.DataFrame` (column name does not matter), containing the value of the portfolio for each trading day in the first column from `start_date` to `end_date`, inclusive.

The files containing orders are CSV files with the following columns:

- Date (yyyy-mm-dd)
- Symbol (e.g. AAPL, GOOG)
- Order (BUY or SELL)
- Shares (no. of shares to trade)

For example:

```
Date,Symbol,Order,Shares
2008-12-3,AAPL,BUY,130
2008-12-8,AAPL,SELL,130
2008-12-5,IBM,BUY,50
```

Your simulator should calculate the total value of the portfolio for each day using **adjusted closing prices**. The value for each day is cash plus the current value of equities. The resulting data frame should contain values like this:

```
2008-12-3 1000000
2008-12-4 1000010
2008-12-5 1000250
...
```

How it should work

Your code should keep account of how many shares of each stock are in the portfolio on each day and how much cash is available on each day. Note that negative shares and negative cash are possible. Negative shares mean that the portfolio is in a short position for that stock. Negative cash means that you've borrowed money from the broker.

When a BUY order occurs, you should add the appropriate number of shares to the count for that stock and subtract the appropriate cost of the shares from the cash account. The cost should be determined using the adjusted close price for that stock on that day.

When a SELL order occurs, it works in reverse: You should subtract the number of shares from the count and add to the cash account.

Evaluation

We will evaluate your code by calling `compute_portvals()` with multiple test cases. No other function in your code will be called by us, so do not depend on "main" code being called. Do not depend on global variables.

For debugging purposes, you should write your own additional helper function to call `compute_portvals()` with your own test cases. We suggest that you report the following factors:

- Plot the price history over the trading period.
- Sharpe ratio (Always assume you have 252 trading days in an year. And risk free rate = 0) of the total portfolio
- Cumulative return of the total portfolio
- Standard deviation of daily returns of the total portfolio
- Average daily return of the total portfolio
- Ending value of the portfolio

Part 2: Leverage (5%)

Many brokers allow "leverage" which is to say that you can borrow money from them in order to buy (or sell) more assets. As an example, suppose you deposit $100,000 with your broker; You might then buy $100,000 worth of stocks. At that point you would have a cash position of $0 and a sum of long positions of $100,000. This situation is 1.0 leverage. However, many brokers allow up to 2.0 leverage. So, you could borrow $100,000, to buy more stocks. If you did that, you'd have long positions of $200,000 and a cash position of -$100,000 due to the loan. Here's how to calculate leverage:

```
leverage = (sum(longs) + sum(abs(shorts))) / ((sum(longs) - sum(abs(shorts)) + c
```

Here are a few examples:

- You deposit $100,000, then short $50K worth of stock and buy $50K worth of stock. You would then have $100K of cash, $50K of longs, -$50K of shorts, so your leverage would be 1.0.
- You deposit $100,000 then short $200K worth of stock. You have $300K of cash and -$200K in shorts. So your leverage is 2.0.
- You deposit $100,000 then buy $50K of stock. Your leverage is 0.5.

Your simulator should prohibit trades that would cause portfolio leverage to exceed 2.0.

Orders files to run your code on

Example orders files are available in the orders subdirectory.

Short example to check your code

Here is a very very short example that you can use to check your code. Starting conditions:

```
start_val = 1000000
```

For the orders file `orders-short.csv`, the orders are:

```
Date,Symbol,Order,Shares
2011-01-05,AAPL,BUY,1500
2011-01-20,AAPL,SELL,1500
```

The daily value of the portfolio (spaces added to help things line up):

```
2011-01-05    1000000
```

```
2011-01-06     999595
2011-01-07    1003165
2011-01-10    1012630
2011-01-11    1011415
2011-01-12    1015570
2011-01-13    1017445
2011-01-14    1021630
2011-01-18    1009930
2011-01-19    1007230
2011-01-20     998035
```

For reference, here are the **adjusted close** values for AAPL on the relevant days:

```
              AAPL
2011-01-05  332.57
2011-01-06  332.30
2011-01-07  334.68
2011-01-10  340.99
2011-01-11  340.18
2011-01-12  342.95
2011-01-13  344.20
2011-01-14  346.99
2011-01-18  339.19
2011-01-19  337.39
2011-01-20  331.26
```

The full results:

```
Data Range: 2011-01-05 to 2011-01-20

Sharpe Ratio of Fund: -0.446948390642
Sharpe Ratio of $SPX: 0.882168679776

Cumulative Return of Fund: -0.001965
Cumulative Return of $SPX: 0.00289841448894

Standard Deviation of Fund: 0.00634128215394
Standard Deviation of $SPX: 0.00544933521991

Average Daily Return of Fund: -0.000178539446839
Average Daily Return of $SPX: 0.000302827205547

Final Portfolio Value: 998035.0
```

More comprehensive examples

`orders.csv`

We provide an example, `orders.csv` that you can use to test your code, and compare with others. All of these runs assume a starting portfolio of 1000000 ($1M).

```
Data Range: 2011-01-10 to 2011-12-20

Sharpe Ratio of Fund: 1.21540888742
Sharpe Ratio of $SPX: 0.0183389807443
```

```
Cumulative Return of Fund: 0.13386
Cumulative Return of $SPX: -0.0224059854302

Standard Deviation of Fund: 0.00720514136323
Standard Deviation of $SPX: 0.0149716091522

Average Daily Return of Fund: 0.000551651296638
Average Daily Return of $SPX: 1.7295909534e-05

Final Portfolio Value: 1133860.0
```

`orders2.csv`

The other sample file is `orders2.csv` that you can use to test your code, and compare with others.

```
Data Range: 2011-01-14 to 2011-12-14

Sharpe Ratio of Fund: 0.788982285751
Sharpe Ratio of $SPX: -0.177203019906

Cumulative Return of Fund: 0.0787526
Cumulative Return of $SPX: -0.0629581516192

Standard Deviation of Fund: 0.00711102080156
Standard Deviation of $SPX: 0.0150564855724

Average Daily Return of Fund: 0.000353426354584
Average Daily Return of $SPX: -0.000168071648902

Final Portfolio Value: 1078752.6
```

Hints & resources

Here is a video outlining an approach to solving this problem [youtube video (https://www.youtube.com/watch?v=E1GTOSUoDpE)].

In terms of execution prices, you should assume you get the **adjusted close** price for the day of the trade.

What to turn in

Be sure to follow these instructions diligently!

Via T-Square, submit as attachment (no zip files; refer to schedule for deadline):

- Your code as `marketsim.py` (only the function `compute_portvals()` will be tested)

Unlimited resubmissions are allowed up to the deadline for the project.

Rubric

- Basic simulator: 10 test cases: We will test your code against 10 cases (9.5% per case). Each case will be deemed "correct" if:
  - For each day, abs(reference portval - your portval) < $0.01
- Leverage: 5 test cases (1% per case). Each case will be deemed "correct" if:
  - For each day, abs(reference portval - your portval) < $0.01

Required, Allowed & Prohibited

Required:

- Your project must be coded in Python 2.7.x.
- Your code must run on one of the university-provided computers (e.g. buffet02.cc.gatech.edu), or on one of the provided virtual images.
- When utilizing any of the example orders files code must run in less than 10 seconds on one of the university-provided computers.
- To read in data, use only the functions in util.py

Allowed:

- You can develop your code on your personal machine, but it must also run successfully on one of the university provided machines or virtual images.
- Your code may use standard Python libraries.
- Unladen African swallows.
- You may use the NumPy, SciPy, matplotlib and Pandas libraries. Be sure you are using the correct versions.
- You may reuse sections of code (up to 5 lines) that you collected from other students or the internet.
- Code provided by the instructor, or allowed by the instructor to be shared.

Prohibited:

- Global variables.
- Reading in data by any means other than the functions in util.py.
- Any libraries or modules not listed in the "allowed" section above.
- Any code you did not write yourself (except for the 5 line rule in the "allowed" section).
- Code that takes longer than 10 seconds to run.
- Any Classes (other than Random) that create their own instance variables for later use (e.g., learners like kdtree).

Retrieved from "http://quantsoftware.gatech.edu/index.php?title=MC2-Project-1&oldid=965"

- This page was last modified on 3 February 2016, at 17:13.
- This page has been accessed 7,597 times.