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:

- **2017-06-20**
  - Update: All leverage references set to 2.0.
  - Transaction cost requirement added.

- 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.
- Q: Do we have to reject the order if the stock is not trading on that day? A: Yes.
- Q: Is it true that the orders might not be in order in the csv? A: Yes.
- Q: Are we supposed to sort the date column after? A: Yes.
- Q: If we were to sort, we can assume that multiple orders on the same date won't have any side-effect if they are executed in random orders? A: Yes.
- Q: Should we assume the prices stay the same the whole day? A: Assume execution prices are adjusted close.
- Q: So buying any stocks won't change our portfolio's value (but selling will), correct? A: If you mark the value of the portfolio to the market as of close. Neither buying or selling will affect the value.
- Q: Do we return portvals daily for the entire periods including weekends, holidays and non-trading days? A: Only consider trading days. This should happen for you automatically when using get_data() from util.py
- Q: It seems like my avg_daily_returns and std are off compared to the wiki's because of this. If not, do we count trading days only for when list of symbols (or SPY) are traded A: Assume the market was open iff SPY was traded.

Template

Instructions:

- Make sure your copy of the course repo is up to date.
- 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: Transaction Costs (up to 95% deduction if not implemented)

Note: We strongly encourage you to get the basic simulator working before you implement the transaction cost and leverage components of this project. Ok, now on to transaction costs.

Transaction costs are an important consideration for investing strategy. Transaction costs include things like commissions, slippage, market impact and tax considerations. High transaction costs encourage less frequent trading, and accordingly a search for strategies that pay out over longer periods of time rather than just intraday or over several days. For this project we will consider two components of transaction cost: Commissions and market impact:

- Commissions: For each trade that you execute, charge a commission of $9.95. Treat that as a deduction from your cash balance.
- Market impact: For each trade that you execute, assume that the stock price moves 50 basis points (0.5%) against you. So if you are buying, assume the price goes up 50 bps before your purchase. Similarly, if selling, assume the price drops 50 bps before the sale. For simplicity treat the market impact penalty as a deduction from your cash balance.

Both of these penalties should be applied for EACH transaction, meaning that a complete entry and exit will cost 2 * $9.95, plus 0.5% of the entry price and 0.5% of the exit price.

Apply these penalties BEFORE you calculate leverage.

Part 3: 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 higher 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(abs(all stock positions))) / (sum(all stock positions) + cash)
```

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.

Part 3: Implement author() function (0%)

You should implement a function called `author()` that returns your Georgia Tech user ID as a string. This is the ID you use to log into t-square. It is not your 9 digit student number. Here is an example of how you might implement author():

```
    def author():
        return 'tb34' # replace tb34 with your Georgia Tech username.
```

And here's an example of how it could be called from a testing program:

```
   import marketsim as ms
   print ms.author()
```

Check the template code for examples. We are adding those to the repo now, but it might not be there if you check right away. Implementing this method correctly does not provide any points, but there will be a penalty for not implementing it.

Orders files to run your code on

Example orders files are available in the orders subdirectory.

Here are some additional test cases: testcases_mc2p1.zip

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     997495.775
2011-01-06     997090.775
2011-01-07    1000660.775
2011-01-10    1010125.775
2011-01-11    1008910.775
2011-01-12    1013065.775
2011-01-13    1014940.775
2011-01-14    1019125.775
2011-01-18    1007425.775
2011-01-19    1004725.775
2011-01-20     993036.375
```

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: 2011-01-10 to 2011-12-20

Sharpe Ratio of Fund: -1.00015025363
Sharpe Ratio of $SPX: 0.0183389807443

Cumulative Return of Fund: -0.00447059537671
Cumulative Return of $SPX: -0.0224059854302

Standard Deviation of Fund: 0.00678073274458
Standard Deviation of $SPX: 0.0149716091522

Average Daily Return of Fund: -0.000427210193308
Average Daily Return of $SPX: 1.7295909534e-05

Final Portfolio Value: 993036.375
```

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: 0.997654521878
Sharpe Ratio of $SPX: 0.0183389807443

Cumulative Return of Fund: 0.108872698544
Cumulative Return of $SPX: -0.0224059854302

Standard Deviation of Fund: 0.00730509916835
Standard Deviation of $SPX: 0.0149716091522

Average Daily Return of Fund: 0.000459098655493
Average Daily Return of $SPX: 1.7295909534e-05

Final Portfolio Value: 1106025.8065
```

`orders2.csv`

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

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

Sharpe Ratio of Fund: 0.552604907987
Sharpe Ratio of $SPX: 0.0183389807443

Cumulative Return of Fund: 0.0538411196951
Cumulative Return of $SPX: -0.0224059854302

Standard Deviation of Fund: 0.00728172910323
Standard Deviation of $SPX: 0.0149716091522

Average Daily Return of Fund: 0.000253483085898
Average Daily Return of $SPX: 1.7295909534e-05

Final Portfolio Value: 1051088.0915
```

Hints & resources

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

Hint, use code like this to read in the orders file:

```
orders_df = pandas.read_csv(orders_file, index_col='Date', parse_dates=True, na_values=['nan'])
```

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

Out of a total of 100 points:

- Basic simulator: 95: 10 test cases: We will test your code against 10 cases (9.5 points per case). Points per case are allocated as follows:
  - 2.0: Correct number of days reported in the dataframe (should be the number of trading days between the start date and end date, inclusive).
  - 5.0: Correct portfolio value on the last day +-0.1%
  - 1.4: Correct Sharpe Ratio +-0.1%
  - 1.0: Correct average daily return +-0.1%
- Transaction costs: Correct execution transaction costs are considered as part of the basic execution of the simulator, so up to 95% may be deducted if costs are not implemented correctly.
- Leverage: 5: 5 test cases (1 point per case). Points per case are allocated as follows:
  - 1.0: Correct portfolio value on the last day +-0.1%
- author() method not implemented -20%

Test Cases

Here are the test cases we used while grading. These are updated each semester, and released after grading.

- MC2-Project-1-Test-Cases-spr2016

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).

- 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 stock 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.
- To read in orders files pandas.read_csv() is allowed.

Prohibited:

- Global variables.
- Reading in data by any means other than the functions in util.py or pandas.read_csv()
- 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).

Legacy

MC2-Project-1-Legacy