CS7646 ML4T
Machine Learning
for Trading

# PROJECT 5: MARKETSIM

## REVISIONS

This assignment is subject to change up until 3 weeks prior to the due date. We do not anticipate changes; any changes will be logged in this section.

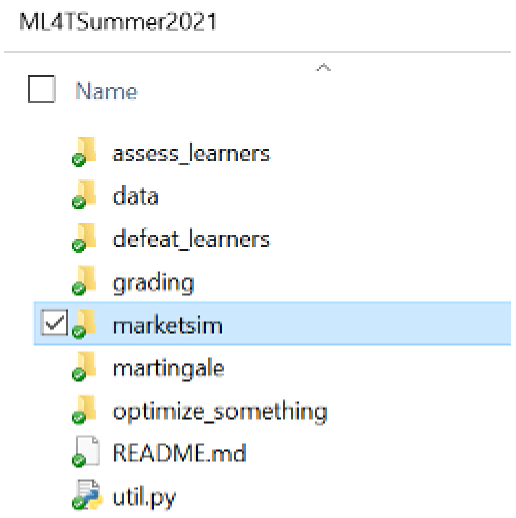
## ABOUT THE PROJECT

In this project, you will create a market simulator that accepts trading orders and keeps track of a portfolio's value over time. It also then assesses the performance of that portfolio. You will submit the code for the project in Gradescope SUBMISSION. There is no report associated with this project.

## STARTER CODE

To make it easier to get started on the project and focus on the concepts involved, you will be given a starter framework. This framework assumes you have already set up the local environment and ML4T Software. The framework for Project 5 can be obtained from: Marketsim_2021Summer.zip.

Extract its contents into the base directory (e.g., ML4T_2021Summer). This will add a new folder called "marketsim" to the course directly structure.


ML4TSummer2021

Name

assess_learners
data
defeat_learners
grading
marketsim
martingale
optimize_something
README.md
util.py

Within the marketsim folder are one directory and two files:

- grade_marketsim.py
  The local grading / pre-validation script. This is the same script that will be executed in the Gradescope TESTING Environment
- marketsim.py
  Student implementations will replace your compute_portvals() function within this file, modifying or replacing the existing stub code the is provided in the file.
- orders directory
  A director that contains several order files that can be used with this project.

## TASK & REQUIREMENTS

### Part 1: Basic simulator (90 points)

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 = 10000
  # 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 arguments are as follows:

- orders_file is the name of a file from which to read orders, and
- start_val is the starting value of the portfolio (initial cash available)
- commission is the fixed amount in dollars charged for each transaction (both entry and exit)
- impact is the amount the price moves against the trader compared to the historical data at each transaction. Impact of 0.01 in the API corresponds to an impact of 1%.

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 999418.90
2008-12-5 999754.30
...
```

### 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 a 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 (10 points)

***Note:*** We strongly encourage you to get the basic simulator working before you implement the transaction cost and market impact 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 payout 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 according to the parameter sent. Treat that as a deduction from your cash balance.
- Market impact: For each trade that you execute, assume that the stock price moves against you according to the impact parameter. So, if you are buying, assume the price goes up 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 penalties should be applied for **EACH** transaction, for instance, depending on the parameter settings, a complete entry and exit will cost 2 * $9.95, plus 0.5% of the entry price and 0.5% of the exit price.

## Part 3: Implement author() function (deduction if not implemented)

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 Canvas. 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())
```

Implementing this method correctly does not provide any points, but there will be a penalty for not implementing it.

## TECHNICAL REQUIREMENTS

Your project must be coded in Python 3.6. and run in the Gradescope SUBMISSION environment.

When utilizing any of the example order files, the code must run in less than 10 seconds per test case.

Use only the functions in util.py to read in stock data. Only use the API methods provided in that file. Please note that util.py is considered part of the environment and should not be moved, modified, or copied. For grading, we will use our own unmodified version

You should use pandas' read_csv function to read the orders files.

The "secret" regarding leverage and a "secret date" discussed in the YouTube lecture do not apply and should be ignored.

Note: The Sharpe ratio uses the sample standard deviation.

## HINTS & RESOURCES

Here is a video outlining an approach to solving this problem [YouTube video].

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.

## CONTENTS OF REPORT

There is no report associated with this assignment.

## TESTING

To test your code, you can modify the provided test_code() function in the marketsim.py file. You are encouraged to perform any unit tests necessary to instill confidence that the generated datasets will produce the desired learner results.

Additionally, we have provided the grade_marketsim.py file that can be used for your tests. This file is the same script that will be run when the code is submitted to Gradescope TESTING. This file is not a complete test suite and does not match the more stringent private grader that is used in Gradescope SUBMISSION. To run and test that the file will run from within the marketsim directory, use the command:

```
PYTHONPATH=../:. python grade_marketsim.py
```

You are encouraged to submit your files to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. Gradescope TESTING does not grade your assignment. No credit will be given for coding assignment fails in Gradescope SUBMISSION that has also failed to pass this pre-validation in Gradescope TESTING.

You are allowed unlimited resubmissions to Gradescope TESTING. Please refer to the Gradescope Instructions for more information.

## Additional Example Tests/Checks (Orders File Code Check Examples)

Example orders files are available in the orders subdirectory. You can use these for additional testing.

Here are some additional test cases:

- testcases_mc2p1.zip
- additional_orders.zip

## Short Code Check Example

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: 2011-01-05 00:00:00 to 2011-01-20 00:00:00  Sharpe Ratio of Fun
Sharpe Ratio of $SPX: 0.882168679776  Cumulative Return of Fund: -0.0044705
Cumulative Return of $SPX: 0.00289841448894  Standard Deviation of Fund: 0.
Standard Deviation of $SPX: 0.00544933521991  Average Daily Return of Fund:
Average Daily Return of $SPX: 0.000302827205547  Final Portfolio Value: 993
```

## 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 00:00:00 to 2011-12-20 00:00:00  Sharpe Ratio of Fun
Sharpe Ratio of $SPX: 0.0183389807443  Cumulative Return of Fund: 0.1088726
Cumulative Return of $SPX: -0.0224059854302  Standard Deviation of Fund: 0.
Standard Deviation of $SPX: 0.0149716091522  Average Daily Return of Fund:
Average Daily Return of $SPX: 1.7295909534e-05  Final Portfolio Value: 1106
```

### 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 00:00:00 to 2011-12-14 00:00:00  Sharpe Ratio of Fun
Sharpe Ratio of $SPX: -0.177203019906  Cumulative Return of Fund: 0.0538411
Cumulative Return of $SPX: -0.0629581516192  Standard Deviation of Fund: 0.
Standard Deviation of $SPX: 0.0150564855724  Average Daily Return of Fund:
Average Daily Return of $SPX: -0.000168071648902  Final Portfolio Value: 10
```

## SUBMISSION INSTRUCTIONS

**This is an individual assignment**. All work you submit should be your own. Make sure to cite any sources you reference and use quotes and in-line citations to mark any direct quotes. Citations within the code should be captured as comments.

Late work is not accepted without advanced agreement except in cases of medical or family emergencies. In the case of such an emergency, please contact the Dean of Students.

### Report Submission

There is no report submission associated with this assignment.

### Code Submission

This class uses Gradescope, a server-side autograder, to evaluate your code submission. No credit will be given for code that does not run in the Gradescope SUBMISSION environment. Students are encouraged to leverage Gradescope TESTING prior to submitting an assignment for grading.

Please submit the following file to Gradescope SUBMISSION:

- **marketsim.py**

Do not submit any other files. Note: Only the compute_portvals() function will be tested.

You are allowed a MAXIMUM of three (3) code submissions to Gradescope SUBMISSION.

## RUBRIC

### Report

- No Report

### Code

- See Auto-Grader

### Auto-Grader [100 points]

- Basic simulator: (90 points) 10 test cases: We will test your code against 10 cases (9 points per case) without transaction costs. 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%
  - 1.0: Correct Sharpe Ratio +-0.1%
  - 1.0: Correct average daily return +-0.1%

- Transaction costs: (10 points) 5 test cases: We will test your code against 5 cases as follows:
  - 2.0: Two test cases that evaluate commissions only (impact = 0)
  - 2.0: Two test cases that evaluate impact only (commission = 0)
  - 1.0: One test case with non-zero commission and impact.

## ALLOWED

- Your code may use the standard Python libraries, NumPy, SciPy, matplotlib and Pandas libraries. Be sure you are using the correct versions as stated on the ML4T Local Environment Setup
- Code provided by the instructor or is allowed by the instructor to be shared.
- To read in 'orders' files, you may use the pandas.read_csv() function.
- Code that displays "warning" messages to the terminal or console.
- Students are allowed to share statistics from the *additional orders files* provided in the project wiki (above) in the Students Results thread alone.
- Unladen African swallows.

## PROHIBITED

- No readings of stock files by any other means other than the functions provided in the util.py file. The 'orders' files can be read using pandas.read_csv().
- You may not use any libraries not listed in the "allowed" section above.
- You may not use any code you did not write yourself.
- Code that takes longer than 10 seconds per test case to run.
- You may not use any classes (other than Random) that create their own instance variables for later use (e.g., learners like kdtree).
- The compute_portvals() function may not send any output to the screen/console/terminal.
- Code must not create extra directories.
- Code must not use of global variables.
- Code must not use absolute import statements, such as: *from folder_name import optimization*, where "folder_name" is the path/name of a folder or directory.
- You must **not** share tables and charts.