



PROJECT 5: MARKETSIM



Table of Contents

- Overview
- About the Project
- Your Implementation
- Contents of Report
- Testing Recommendations

Submission Requirements

Grading Information

Development Guidelines

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.

1 OVERVIEW

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 to Gradescope SUBMISSION. There is no report associated with this project.

1.1 Learning Objectives

The specific learning objectives for this assignment are focused on the following areas:

- **Market Simulation:** Develop a market simulator. Variations of this market simulator will play a role in future projects.
- **Transaction Costs:** Develop an understanding of market costs and how they affect the value of a portfolio.

2 ABOUT THE PROJECT

In this project, you will leverage some of the work completed in project 2 to build a market simulator, as described below.

3 YOUR IMPLEMENTATION

Before the deadline, make sure to pre-validate your submission using Gradescope TESTING. Once you are satisfied with the results in testing, submit the code to Gradescope SUBMISSION. Only code submitted to Gradescope SUBMISSION will be graded. If you submit your code to Gradescope TESTING and have not also submitted your code to Gradescope SUBMISSION, you will receive a zero (0).

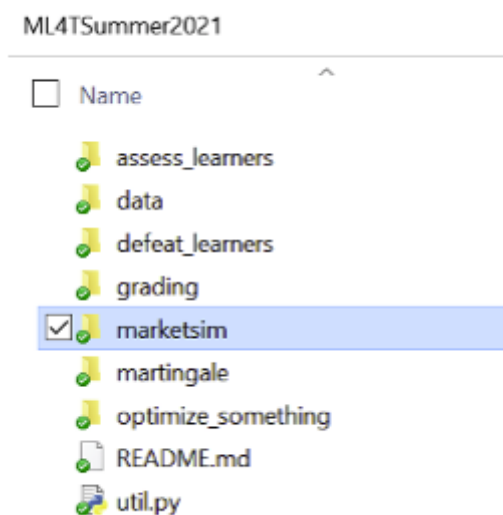
3.1 Getting Started

You will be given a starter framework to make it easier to get started on the project and focus on the concepts involved. This framework assumes you have already set up the [local environment](#) and [ML4T Software](#). The framework for Project 5 can be obtained from: [Marketsim_2022Summer.zip](#).

Extract its contents into the base directory (e.g., ML4T_2022Summer). This will add a new folder called "marketsim" to the course directory structure.

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 provided in the file.
- `orders` directory
 - A directory containing several order files can be used with this project.



3.2 Part 1: Implement the 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:

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

marketsim_api.py hosted with ❤ by GitHub

[view raw](#)

The start date and end date of the simulation are the first and last dates with orders in the `orders_file`. (Note: The orders may not appear in sequential order in the 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:

1	Date,Symbol,Order,Shares
2	2008-12-3,AAPL,BUY,130
3	2008-12-8,AAPL,SELL,130
4	2008-12-5,IBM,BUY,50

marketsim_example_1 hosted with ❤ by GitHub
[view raw](#)

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:

1	2008-12-3 1000000
2	2008-12-4 999418.90
3	2008-12-5 999754.30
4	...

marketsim_example_2 hosted with ❤ by GitHub
[view raw](#)

3.2.1 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.

3.2.2 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 analyze and confirm 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

3.3 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.

Transaction costs are an important consideration of an 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 (e.g., rises 50 bps) before your purchase. Similarly, if selling, assume the price drops (e.g., falls 50 bps) before the sale. For simplicity treat the market impact penalty as a deduction from your cash balance.

Commissions and Market Impact 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.

3.4 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()`:

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

marketsim_author_implementation.py hosted with ❤ by GitHub

[view raw](#)

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

```
1 import marketsim as ms
2
3 print(ms.author())
```

marketsim_author_call.py hosted with ❤ by GitHub

[view raw](#)

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

3.5 Technical Requirements

The following technical requirements apply to this assignment

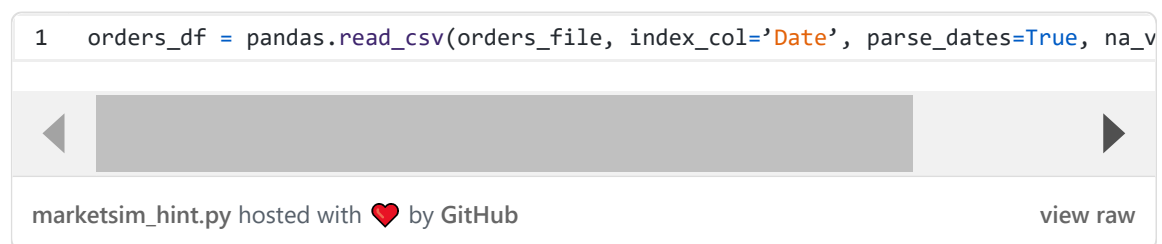
1. When utilizing any of the example order files, the code must run in less than 10 seconds per test case.
2. **Use only the functions in util.py to read stock data.** Only use the API methods provided in that file. Please note that util.py is considered part of the environment and should not be moved, modified, or copied. For grading, we will use our own unmodified version.
3. You should **use pandas' read_csv function to read the orders files.**
4. The "secret" regarding leverage and a "secret date" discussed in the YouTube lecture do not apply and should be ignored.
5. **The Sharpe ratio uses the sample standard deviation.**

3.6 Hints & Suggestions

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

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

```
1 orders_df = pandas.read_csv(orders_file, index_col='Date', parse_dates=True, na_v
```



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

4 CONTENTS OF REPORT

There is no report associated with this assignment.

5 TESTING RECOMMENDATIONS

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 install confidence that the code will run properly when submitted for grading and will produce the required 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:

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

`grade_marketsim` hosted with ❤ by GitHub

[view raw](#)

In addition to testing on your local machine, you are encouraged to submit your files to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. No credit will be given for coding assignments that do not pass this pre-validation. **Gradescope TESTING does not grade your assignment.** The Gradescope TESTING script is not a complete test suite and does not match the more stringent private grader that is used in Gradescope SUBMISSION. Thus, the maximum Gradescope TESTING score of 87, while instructional, does not represent the minimum score one can expect when the assignment is graded using the private grading script. You are encouraged to develop additional tests to ensure that all project requirements are met.

You are allowed **unlimited** resubmissions to Gradescope **TESTING**. Please refer to the [Gradescope Instructions](#) for more information.

5.1 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](#)

5.2 Short Code Check Example

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


```
1 start_val = 1000000
```

`short_example_1` hosted with ❤ by GitHub

[view raw](#)


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

1	Date,Symbol,Order,Shares
2	2011-01-05,AAPL,BUY,1500
3	2011-01-20,AAPL,SELL,1500

short_example_2 hosted with  by GitHub [view raw](#)

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

1	2011-01-05	997495.775
2	2011-01-06	997090.775
3	2011-01-07	1000660.775
4	2011-01-10	1010125.775
5	2011-01-11	1008910.775
6	2011-01-12	1013065.775
7	2011-01-13	1014940.775
8	2011-01-14	1019125.775
9	2011-01-18	1007425.775
10	2011-01-19	1004725.775
11	2011-01-20	993036.375

short_exmample_3 hosted with  by GitHub [view raw](#)

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

1		AAPL
2	2011-01-05	332.57
3	2011-01-06	332.30
4	2011-01-07	334.68
5	2011-01-10	340.99
6	2011-01-11	340.18
7	2011-01-12	342.95
8	2011-01-13	344.20
9	2011-01-14	346.99
10	2011-01-18	339.19
11	2011-01-19	337.39
12	2011-01-20	331.26

short_example_4 hosted with  by GitHub [view raw](#)

The full results:

1	Data Range: 2011-01-05 00:00:00 to 2011-01-20 00:00:00
2	Sharpe Ratio of Fund: -1.00015025363
3	Sharpe Ratio of \$SPX: 0.882168679776
4	Cumulative Return of Fund: -0.00447059537671
5	Cumulative Return of \$SPX: 0.00289841448894
6	Standard Deviation of Fund: 0.00678073274458
7	Standard Deviation of \$SPX: 0.00544933521991
8	Average Daily Return of Fund: -0.000427210193308
9	Average Daily Return of \$SPX: 0.000302827205547
10	Final Portfolio Value: 993036.375

short_example_5 hosted with ❤️ by GitHub

[view raw](#)

5.3 More Comprehensive Examples

orders.csv

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

1	Data Range: 2011-01-10 00:00:00 to 2011-12-20 00:00:00
2	Sharpe Ratio of Fund: 0.997654521878
3	Sharpe Ratio of \$SPX: 0.0183389807443
4	Cumulative Return of Fund: 0.108872698544
5	Cumulative Return of \$SPX: -0.0224059854302
6	Standard Deviation of Fund: 0.00730509916835
7	Standard Deviation of \$SPX: 0.0149716091522
8	Average Daily Return of Fund: 0.000459098655493
9	Average Daily Return of \$SPX: 1.7295909534e-05
10	Final Portfolio Value: 1106025.8065

comprehensive_1 hosted with ❤️ by GitHub

[view raw](#)

orders2.csv

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

1	Data Range: 2011-01-14 00:00:00 to 2011-12-14 00:00:00
2	Sharpe Ratio of Fund: 0.552604907987
3	Sharpe Ratio of \$SPX: -0.177203019906
4	Cumulative Return of Fund: 0.0538411196951
5	Cumulative Return of \$SPX: -0.0629581516192
6	Standard Deviation of Fund: 0.00728172910323

7	Standard Deviation of \$SPX: 0.0150564855724
8	Average Daily Return of Fund: 0.000253483085898
9	Average Daily Return of \$SPX: -0.000168071648902
10	Final Portfolio Value: 1051088.0915
comprehensive_2 hosted with ❤️ by GitHub view raw	

6 SUBMISSION REQUIREMENTS

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.

Assignment due dates in your time zone can be found by looking at the Project in the Assignment menu item in Canvas (ensure your Canvas time zone settings are set up properly). This date is 23:59 AOE converted to your time zone. Late submissions are allowed for a penalty. The times and penalties are as follows:

- -10% Late Penalty: +1 Hour late: submitted by 00:59 AOE (next day)
- -25% Late Penalty: +12 Hours Late: submitted by 11:59 AOE (next day)
- -50% Late Penalty: +24 Hours Late: submitted by 23:59 AOE (next day)
- -100% Late Penalty: > 24+ Late: submitted after 23:59 AOE (next day)

Assignments received after Monday at 23:59 AOE (even if only by a few seconds) are 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](#).

6.1 Report Submission

There is no report submission associated with this assignment.

6.3 Code Submission

This class uses Gradescope, a server-side auto-grader, to evaluate your code submission. No credit will be given for code that does not run in this environment and students are encouraged to leverage Gradescope TESTING prior to submitting an assignment for grading. **Only code submitted to Gradescope SUBMISSION will be graded. If you submit your code to Gradescope TESTING and have not also submitted your code to Gradescope SUBMISSION, you will receive a zero (0).**

Please submit the following file to Gradescope **SUBMISSION**:

marketsim.py

Do not submit any other files.

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

7 GRADING INFORMATION

The submitted code (which is worth 100% of your grade) is run as a batch job after the project deadline. The code represents 100% of the assignment grade and will be graded using a rubric design to mirror the implementation details above. Deductions will be applied for unmet implementation requirements or code that fails to run.

We do not provide an explicit set timeline for returning grades, except that all assignments and exams will be graded before the institute deadline (end of the term). As will be the case throughout the term, the grading team will work as quickly as possible to provide project feedback and grades.

Once grades are released, any grade-related matters must follow the [Assignment Follow-Up guidelines and process](#) alone. Regrading will only be undertaken in cases where there has been a genuine error or misunderstanding. Please note that requests will be denied if they are not submitted using the Summer 2022 form or do not fall within the timeframes specified on the [Assignment Follow-Up](#) page.

7.1 Grading Rubric

7.1.1 Report [0 points]

There is no report associated with this project.

7.1.2 Code

There is no separate code section associated with this project.

7.1.3 Auto-Grader (Private Grading Script) [100 points]

We will test your code against the following cases:

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 $\pm 0.1\%$
- 1.0: Correct Sharpe Ratio $\pm 0.1\%$

- 1.0: Correct average daily return $\pm 0.1\%$

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.

8 DEVELOPMENT GUIDELINES (ALLOWED & PROHIBITED)

See the [Course Development Recommendations, Guidelines, and Rules](#) for the complete list of requirements applicable to all course assignments. **The Project Technical Requirements are grouped into three sections: Always Allowed, Prohibited with Some Exceptions, and Always Prohibited.**

The following exemptions to the Course Development Recommendations, Guidelines, and Rules apply to this project:

- To read in 'orders' files, you may use the `pandas.read_csv()` function.
- Watermarked charts may be shared in the dedicated discussion forum thread alone.
- Code may not take longer than 10 seconds per test case to run.