Python Stock Back-tester

CMSC 495 6981

Nancy Isaac

June 14, 2015

Contents

[Overview 3](#_Toc422667840)

[Project Members 3](#_Toc422667841)

[Project Plan 3](#_Toc422667842)

[IT Systems 4](#_Toc422667843)

[Data 5](#_Toc422667844)

[Development Process 5](#_Toc422667845)

[Requirement Specification 5](#_Toc422667846)

[System Specification 5](#_Toc422667847)

[User’s Guide 5](#_Toc422667848)

[Test Plan and Results 5](#_Toc422667849)

[Design and Alternative Designs 5](#_Toc422667850)

[Development History 6](#_Toc422667851)

[Conclusion 6](#_Toc422667852)

# Overview

Many organizations systematically trade U.S. equity (common stock) with daily, weekly, and monthly horizons. The trading of stock with a high level of turnover is a costly endeavor due to transaction costs, taxes, etc., and therefore, those that do partake such activity typically perform extensive research on the trading system. Trading commences only after it has been determined, with high statistical confidence, that the rules that define the system are profitable.

This program will read a series of historical equity price and volume data and then test if any pre-programed rules (indicators), or combination of indicators, predicts future returns with a high statistical significance.

## Project Members

* Baker, Pierce
* Casey, Garrett
* Isaac, Nancy
* Kelley, Justin
* Owens, Chris

# Project Plan

Indicator functions will consume Open, High, Low, Close Prices and Volume (OHLCV) data and return Boolean values. Example indicators include:

NEW\_20\_PERIOD\_CLOSING\_HIGH, which is indicating a closing price higher than all closing prices over the last 19 days

NEW\_52\_WEEK\_LOW, which indicates a closing price lower than all closing prices over the last 52 weeks

The user will choose which indicators to test. If the user selects multiple indicators, the truthfulness of the test will be the intersection of all indicators (logical AND).

The primary part of the program is the marking out forward returns. When indicator(s) return true, the program takes note of the current price and then peaks into the future (mark-out period) and records the future price. For example, the user wants to test the HIGH\_VOLUME indicator function with mark-out periods of 5, 10 and 20 days for AAPL (stock symbol). The log file stored on disk will have similar entries to the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Ticker | Entry Price | Mark-out period | Mark-out price |
| 20150612 | AAPL | 120.28 | 5 | 122.24 |
| 20150612 | AAPL | 120.28 | 10 | 121.18 |
| 20150612 | AAPL | 120.28 | 20 | 119.20 |

Table : Information stored for each event

It will be important to try to keep all events as independent as possible. Therefore, if AAPL hits a new HIGH\_VOLUME in 2 days from now, the program will not produce a new event. If a new event occurs after seven days from now, we will only add a 5-day mark-out row. Therefore, a trade manager will need to keep track of all TICKER/MARKOUT pairs and only record new events if one does not exist.

After the all the trades have been collected, the application will calculate statistics for each mark-out period, allowing the user will determine if indicator has any influence on the direction of prices 5 days, or 10 days or 20 days in the future.

## IT Systems

We will write the program using Python 2.7x, using only the core modules. This minimize the need for users to install any addition Python modules.

The user will interact via the command-line. A configuration file will provide the user specific settings. After the back-test is complete, the program will write the results to a well formatted, easily parsed, human readable, text file.

The program will have a verbose mode, with progress reported to STDOUT. After the back-test is complete, the program writes the results to a well-formatted text file.

We will be using the GitHub.com as our code repository and collaboration tool.

## Data

Historical equity price and volume data will be fetch from finance.yahoo.com via a public API.

## Development Process

Each team member will be assigned one or more of the classes to write and will be tasked with providing documentation and test units for said Classes.

Each team member will code locally and then submit their work to the group via pull requests, whereupon the team will review the code, comment, and then eventually merge into the project’s master branch.

# Requirement Specification

## Functional Requirements

1. The stock back-tester shall use real world data.

We will be using yahoo finance to gather the historical information on a particular ticker symbol. The data that we collect is based on the indicator criteria being triggered. A tester of this program can collect the data from yahoo finance and compare the data to what we collect for this program.

1. The stock back-tester must use well known stock trading indicators.

There are well known indicators that are used in the financial industry. Yahoo finance will give us the OHLCV (open, high, low, close, volume) values. These values can be used in combination with the indicators can be used for back testing.

1. Indicators can be combined for the query.
2. The program must provide a command line interface to the user
3. A configuration file must be used to gather user inputs
4. The following fields must be defined in the configuration file: PORTFOLIO.tickers, STRATEGY.name, and STRATEGY.indicators. All other variables will have default values.
5. User must be able to provide a start date and an end date
6. The user must be able to provide a stock symbol
7. The user must be able to provide a mark-out period of 5, 10 and 20 days
8. All inputs must be logged to STDOUT and to a log file
9. User should have the ability to silence the output to STDOUT
10. The program must output a summary of statistics that can easily be read by humans and machines.
11. The program must output a detailed report of the statistics
12. The program should provide a nice GUI interface to parse the detailed report and provide graphs of the results to the user.
13. The detailed statistics must include forward returns
14. The forward returns should be calculated using the standard formula: log return = ln(end) - ln(start) where ln is the natural log function. *See*  *(Why Log Returns) for info*
15. The detailed statistics should include count, mean and standard deviation
16. The real world data must include the corporate-action adjusted prices.
17. The data source must provide the ability to generate trade-date calendar.
18. The program that evaluates the indicator must return a Boolean value.
19. The program that handles the data must validate the boundaries of the sample period, mark-out periods and maximum indicator look-back period
20. The program responsible for testing the indicators against the historical data must be able to ignore events in overlapping mark-out periods to ensure independent events.

## Non-Functional Requirements

1. The program should be light-weight
2. The program must be able to run on any OS that has python installed.
3. The program must be able to easily reproduce the back-tests with the config files.
4. The program should support multiple independent instantiations of the applications without causing errors.

# System Specification

TBD

# User’s Guide

We provide a preliminary User’s Guide in the distribution’s README.txt file.

# Test Plan and Results

As this program will have many moving parts, it has been parsed into six separate classes. These classes will execute the program, fetch data, compile results, and output the results as requested. The program’s development shall be tested piece meal as time continues.

## Test Case 1:

This test case was simply to ensure that the runBacktester.py file is properly accepting input from the user and storing it properly. While not testable, some classes were imported and classed, although those classes are still being developed**. *Expectations:*** It is expected that the runBacktest.py file will execute without fail. The user will input his queried data and the program will confirm his entries. ***Results***: The test was without flaw and the file was storing user input properly.

## Test Case 2:

During test case two, we shall ensure that the program is capable of compiling without error and outputting an actual data file without actually calculating statistics from data, or pulling data. This test case will ensure the Config Class is configured properly and correctly parsing the command line arguments. The TradeManager Class, and a logger will be incorporated into this test. **Expectations:** It will be imperative that the config class executes without flaw. It is expected that ensure the TradeManager class to execute without flaw may take time. The program should compile without error. **Results**: TBD

## Test Case 3:

This test case will simply begin to pull data from the source and ensure that it is pulling properly. Output will be bypassed and the data of the pull will be simply saved to a txt file. **Expectation:** Ensuring that data is being pulled properly is not expected to be difficult. The data should match exactly as it was pulled. **Results**: TBD

## Test Case 4:

Test case 4 will test against the indicators and the class used to run them. Indicators will be tested and ran against the data by the TradeManager Class. Data markers, which meet the indicators, will be printed to a .txt file and compared against an expected output. **Expectations:**  Either test case 4 or 5 will prove the most difficult. Two very complex classes will be integrated and it is possible that it will take much decoding to get them to run properly. The noted indicators should match what is expected **Results**: TBD

## Test Case 5:

Test case 5 shall test against the analysis class and bring everything together. Results will be compared against a comparable program to ensure data integrity. We shall then output the format into a readable format. **Expectation:** As noted previously, this is expected to be one of the hardest test cases. The mathematics involved in creating statistics and outputs will not be easy. It may take time to resolve errors. The program should compile comparable statistics to open source programs. **Results**: TBD

# Design and Alternative Designs

TBD

# Development History

TBD

# Conclusion

TBD