Technical Document

A fully replicable Equity backtesting workflow

Created by Riaz Arbi in fulfilment of the dissertation requirements for an MSc in Data Science at the University of Cape Town

30 October 2018

Contents

Project Purpose

This paper provides a working example of an equity backtest that has been conducted in R according to the principles of transparency and reproducability in research. It provides working code to conduct the full analysis chain of equity backtesting in a way that is totally automated and modifiable. The companion git repository to this paper can be cloned and, with minimal modifications, new backtests can be created that avoid the pitfalls of common statistical biases. It is hoped that this work will make quality research into the cross section of equity returns more accessible to practitioners.

Principles

This project favours transparency and customizability over ease of use. For a sophisticated, easy to use backtesting environment see Zipline or QSTrader.

This project aims to be -
- Totally transparent in the flow and transformation of data
- Low-level in terms of dependencies
- Highly customizable
- Easy to set up in any environment

Intended Audience

This project should be useful to:

- Finance students at all levels wanting to conduct statistically rigorous equity backtests
- Post graduates and academics looking to conduct research on a common platform to facilitate replication and peer review
- Research professionals looking to build out an in-house backtesting environment for proprietary equity investment strategies
- Equity researchers looking for a bridge between Excel-based research and `R` or `python`.

Included Data

This repository **does not include any equity data**, but it does include working scripts to automatically extract data from data vendors, and to save that data on a well-formed way. It is assumed that the user will acquire data using the scripts which have been included in this repository. This will only be possible if a user has access to the relevant data services - namely Bloomberg, DataStream or iNet.

The dimensions directory

The dimensions directory contains dimensions relevant to the backtest. These dimensions are organized on a per-file basis. For instance, the `fundamental_fields.csv` file contains a list of fundamental Bloomberg fields salient to the backtest.

Replication and Extension to other use cases

The only files that change between different backtests are `alorgithm.R`, which houses the portfolio weighting rules, and `parameters.R`, which houses the backtest parameters.

To replicate the results of a backtest using this codebase, simply clone this git repository to your computer, and run the the `trade.R` script.

To replicate the results of this paper on another equity index, change the index parameter in the `parameters.R` file to another index.

To alter the algorithm, modify the `compute_weights` function in the `algorithm.R` script.

Code flow

The scripts in this repository are written in a procedural manner. That is, it is intended that parameters are set prior to execution and that the scripts are executed noninteractively. The code logic flows down each script in a linear manner wherever possible. This style facilitates the auditing of how data is manipulated as it flows through the code logic.

Data Pipeline

The data pipeline scripts can be run as in a standalone fashion. This facilitates spreading scripts across multiple machines, whic can be helpful when data vendor terminals are shared or on different machines. In such instances, the query scripts can be run on each terminal, and the logs can be copied across to a single logfile for downstream processing. Logfiles are timestamped to Unix time, so logs from different machines will sort correctly.

Because each data pipeline script can be run standalone (ie from a clean R environment), they are amenable to scheduling using `cron`, ‘anacron or similar.

Simulation

Effort has been made to keep the simulation scripts as simple as possible, to keep them true to the spirit of this project. Nevertheless, the event-driven nature of the simulation renders

Limiting the number of code files

Wherever possible, the code is kept in a single file. Code is split between several files when the contents of the files do not naturally run as the same execution batch. For instance, in an academic setting, data is often collected from vendors through the use of shared terminals, often situated in a library or laboratory. It does not make sense to include the data collection code with downstream processing code because downstream processing can be done on a another machine at a different time, freeing up the terminal for another user.

Chunking and Parallelization

Researchers are often limited by their computing reosurces. Most of the time, research is conducted on consumer hardware with limited amounts of RAM. To accommodate this, workloads are chunked or run sequentially to limit RAM consumption. This slows down execution time. The worst offending code chinks have been parallelized to mitigate this, and data is stored on disk in `feather` format to speed up disk I/O.

Sequential list of procedures - Backtest Mode

Data Pipeline

1. Index, ticker market, fundamental and meta data is collected from Bloomberg via the R Rblpapi package and saved to a log directory.
2. The logfiles are transformed into three datasets -
    a. Ticker market and fundamental data
    b. Ticker metadata
    c. Monthly index constituent lists

Simulation

3. Backtest parameters are defined
    a. Target index
    b. Date range
    c. Salient ticker metrics
    d. Mode: Backtest or Live (live isn't implemented yet)
4. A portfolio constituent weighting criterion is defined
    a. For instance, "this portfolio will weight the constituents of the index according to their Price to Book ratio."
    b. This weighting can be any mathematical function, and can contain binary statements such as "weight the constituents equally of their price to book ratio is greate than 2, otherwise weight them as zero."
5. A master dataset is loaded into memory
6. The code loops through the following steps in heartbeat increments (typically 10 minutes, but can be set in `parameters.R`) until the backtest date range ends (backtest mode) or never (live mode).
    a. A runtime dataset is created, adjusting for survivorship and look-ahead bias.
    b. The runtime dataset is passed to the `algorithm.R` function, and target portfolio weights are computed.
    c. [Not implemented] Transaction logs and trade histroy is loaded to compute current portfolio positions.
    d. [Not implemented] An order list is generared from the diff of target weights and existing weights.
    e. [Not implemented] Orders are submitted to the trading engine.
    f. [Not implemented] The trading engine reads market data for the tickers and submit trades.

g. [Not implemented] If volume limits are not hit, the trade completes and is appended to the trade history log. Cash movements are appended to the transaction log.

7. [Not implemented] The final trade and transaction logs are used to compute portfolio retuirns. THese are used to compute various risk measures, Sharpe Ratio chief among them.

8. [Not implemented] The risk and return results and various matrices, backtest parameters and algorithm script are bundled into a folder that follows the Data naming conventions above. The folder that can be passed to a report generator for automated reporting.

Data flow

The data query scripts save four kinds of dataset to the `datalog` folder. These are -

1. Index constituents for a certain date
2. Ticker metadata for an arbitrary list of tickers
3. Fundamental data for a ticker for an arbitrary date range
4. Market data for a ticker for an arbitrary date range

The files in the `datalog` directory can be identified for their filename. The `filename` contains the following substrings, where each substring is separated by two underscores (ie `__`)

- substring1: timestamp (integer-represented UNIX epoch time)
- substring2: source (eg. BLOOMBERG, REUTERS)
- substring3: data type (eg. constituent list data, ticker market data, ticker fundamental data, metadata array)
- substring4: data label (eg. 20120101_TOP40; ANG SJ)

A sample filename is `1029384859940__BLOOMBERG__constituent_list_data__20121001_TOP40.csv`

This file naming convention allows the datalog to be searchable by a conbination of substrings. This is leveraged by the dataset builder to transform the datalog into well-formed datasets.

Development Status

Versioning and Support

This project will continually change. A user is advised to compare between commits on github to see how the code changes over time. It is anticipated that this codebase will be altered significantly and rapidly until December 2018; thereafter development will slow. A good way to check the development status is look at the github commit logs.

It is advised that a user of this codebase fork the repository in order to stabilize their codebase at a particular point in time. When sharing your work, you can facilitate peer review and replication by including access to your forked repository.

There is no guaranteed support for this project past December 2018, but users are free to fork and develop it on their own. Wherever possible, the code is written in `R` to ensure compatibility with generally available computer software for the forseeable future.

Project Personnel, Reporting and Responsibility

This codebase is being built out by Riaz Arbi, who can be contacted via riazarbi.github.io. I take no responsibility for any errors contained in this code. A user of this code should verify each line of the codebase to ensure that it operates as expected.

Intellectual Property Statement

This is the intellectual property of Riaz Arbi. This code will be released under a strong copyleft license. A license is included in the repository.

Feature List

Data pipeline

Feature	Planned	Implemented
Specification: Log formats	X	X
Pipeline: Bootstrap regeneration if data lost	X	X
Log Dataset: Keep query logs forever	X	X
Mining: Summary statistics of logfiles		
Pipeline: Accept arbitrary data sources	X	X
Pipeline: Bloomberg query script	X	X
Pipeline: Datastream query script	X	
Pipeline: iNet query script	X	
Pipeline: Random Ticker Dataset Generator		
Pipeline: Data compaction	X	X
Pipeline: Log to dataset	X	X
Specification: Dataset formats	X	X
Master Dataset: Data versioning	X	X
Mining: Summary statistics of datasets		
Pipeline: Join ticker, constituent and metadata into master dataset	X	X
In-memory Dataset: Automate loading	X	X
Scoring Dataset: Compute ticker returns timeseries for scoring		
Scoring Dataset: Compute benchmark returns timeseries for scoring		

Simulation

Feature	Planned	Implemented
Easily switch between backtesting and live trading	X	
Specify generic portfolio weighting rule	X	X
Automatically load master dataset	X	X
Simulate market data stream	X	X
Build trading engine		
Model slippage and trading costs		
Keep track of trades		
Keep track of cash balances		

Feature	Planned	Implemented
Build event-driven while-loop	X	X
Inside loop: dynamically refresh dataset	X	X
Inside loop: Eliminate survivorship bias	X	X
Inside loop: Eliminate look-ahead bias	X	X
Inside loop: Create target portfolio weights	X	X
Inside loop: Track existing portfolio weights		
Inside loop: Compute order list	X	X
Inside loop: Submit orders to trading engine	X	X
Inside loop: Save successful orders to transaction log		
Logging: Create backtest archive directory		
Logging: Save transactions		
Logging: Save trades		
Logging: Save algorithm script		
Logging: Save parameters script		
Evaluation: Compute portfolio returns		
Evaluation: Compare portfolio returns to index		
Evaluation: Generate backtest report		
Reproducibility: Create Dockerfile and docker image to replicate environment		

Requirements Specification

1. Hardware

- My development machine is a modern generic x86 workstation running Ubuntu 16.04 LTS.
- It is conneced to the internet
- It is recommended that you have at least 4gb of RAM for data transformation; large datasets will require significantly more.

2. Software

- At a minimum, you need to have python 3.5, R version 3.4.3 running on a modern x86 linux distribution.
- A Dockerfile and docker image will be made available to facilitate the setup of an environment.

3. Data

- This project presupposes that a user has access to a data vendor subscription. It contains R scripts that can automatically extract relevant data from the Bloomberg BBCOM.exe server. It is anticipated that support will be added for similar automated R-based extraction of DataStream and iNet sources.

4. Maintenance

- The user will have to periodically upload new data if this environment is to be converted into a production trading system. The Excel scrapers can easily be modified to perform incremental additons to `raw` directories but a produciton system should probably talk directly to an API using a native data vendor API.