# INTRODUCTORY NOTE

This prject aims to build out a complete analysis pipeline for equity analysis. It encompasses the full chain from acquisition all the way to visualization.

# To Do

Move the entire workflow from a files-and-folders data structure to a log-centric structure.

All raw data will be saved to a log directory called ‘data’. The actual files in the log directory will be flat data tables; the identifying attributes of the data will be saved to the filename. The filename separator will be two underscores (ie \_\_)

* timestamp (integer-represented UNIX epoch time)
* source (BLOOMBERG, REUTERS)
* data type (constituent list data, ticker market data, ticker fundamental data, metadata array)
* data label (20120101\_TOP40; 20120101\_ANG SJ)

A sample filename is 1029384859940\_\_BLOOMBERG\_\_constituent\_list\_data\_\_20121001\_TOP40

The log will be searchable by filtering the filename.

**A CONCERN** File path length limitations. If I hit this problem, I’ll build a key-value store that relates the timestamp (unique) to the filename and rename all files with just their timestamps and throw the filename in the key-value store. Adds another layer but inevitable. Another option is to move the whole framework (including data) to JSON.

**MILESTONES** The first iteration will just log actual data. Second iteration will log dimensions as well and let us version our dimension queries. This may require addition of another parameter in the filename schema, or it may be rolled into source. Third iteration will log data transformations and aggregation as well. The fundamental principle is that you should be able to replay everything from the log directory.

**OTHER ISSUES** Checksumming, bit rot, indexing, all the database problems. Maybe move to Apache Kafka at some point. But this adds yet more overhead! s

# WHAT FOLLOWS BELOW IS MY OLD TECHNICAL DOCUMENT FOR THE FIRST ITERATION OF THIS PROJECT.

IT IS ALL INCORRECT NOW, BUT SHOULD GIVE YOU AN IDEA OF THE OBJECTIVES OF THE PROJECT ###################################################################### # 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*

## Project Purpose

The purpose of this project is to build out a fully reproducible, semi-automated workflow for the backtesting and exploratory data analysis of equity investment strategies. When complete, it should be possible for a researcher to clone this repository and, with minimal effort, conduct statistically robust investment strategy backtests on any subset of global traded common stock instruments.

At present, it is not common practice to provide code when publishing academic research on the cross section of common stock returns. This leads to considerable confusion when fellow researchers attempt replication of results - which is a key requirement for the advancement of scientific knowledge. This problem is further compounded when replication attempts fail to provide their backtesting codebases - in this ‘double failure’ it is not clear whether the orginal paper or the replicated paper (or both) have made errors in their analysis!

This problem is merely technical; there exist mature, freely available software that facilitates transparency in data manipulation and statistical analysis. This codebase provides a template for glueing together these tools into an end-to-end workflow.

# Description

This is **not** a software package. It is a collection of scripts that collectively comprise a template workflow to conduct statistically rigorous backtesting of equity investment strategies. As such, it does not present itself to a user as a command -option [argument] command-line program or GUI.

Rather, a user manually invokes scripts in sequence as and when appropriate. In its unmodified form, this codebase should provide all that is necessary to perform a simple ‘batteries included’ backtest of an equity investment strategy.

It is anticipated that the user’s operating system will conduct the automated aspects of the workflow using cron and run custom interactive scripts in RStudio Server or Jupyter browser-based environments.

To ensure stability in end-user use, it is recommended that a user forks this codebase at the start of their research project. This will ensure that future changes to this repo do not break functionality that exists at a particular point in time.

**To preserve the principle of reproducibility, any customizations in your fork should be documented as a complete bash, python or R script so that a replicator can easily transform a base repo into your customized, working environment.**

## Principles

This project favours transparency and customizability over ease of use. For a sophisticated, easy to use backtesting environment see [Zipline](https://github.com/quantopian/zipline) or [QSTrader](https://github.com/mhallsmoore/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 template Excel workbooks that contain the necessary VBA code to automatically extract data from data vendors. It is assumed that the user will acquire data using the Excel-VBA based tools 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.

# Documentation

This README.md file contains a broad overview of the philosophy, principles and functionality of this project. The codebase is a sequential series of scripts; all code is clearly commented for transparency.

## Setup

**This codebase is being heavily modified weekly. You should assume that the structure and functionality will vary significantly in the immediate future.**

1. Fork this repo to your own profile. This is so that you can commit to your own repo as you customize your environment down the line.
2. Assuming you have a fully configured environment, create a new user to house all your backtesting research. If you do not have a fully configured environment yet, use my [serversetup](https://github.com/riazarbi/serversetup) scripts to automatically set up a correctly configured server.
3. Clone your newly forked repo into the empty /home/ directory.
4. Navigate to the directory ~/backtest\_workflow/data\_preparation and execute python -m folder\_structure\_builder.py. This will build out the ~/data\_drop and ~/backtest\_data directories with the appropriate directory structure.

The directory structure below is a representation of the /home/<user> directory once python -m folder\_structure\_builder.py has been run.

├── backtest\_workflow  
│ ├── data\_preparation  
│ ├── backtests  
│ ├── visualizations  
│ └── README.md  
├── data  
│ ├── raw\_archives  
│ ├── raw  
| ├── merged  
| ├── clean  
| └── db.sqlite  
└── data\_drop

## Data Flow

This section assumes that you have set up your server using [serversetup](https://github.com/riazarbi/serversetup), cloned the repo and run the folder\_structure\_builder.py script.

### Pre-Backtest Data Flow

All data pre-preparation scripts are housed in the data\_preparation directory. These scripts manipulate data in the workflow\_data directory, and it is expected that the workflow\_data files are used as a ‘cardinal data source’ by downstream backtesting and visualization projects.

The pre-backtest data flow follows a clearly defined, sequential series of steps. In accordance with the [Unix preference for modularity](http://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html), these steps are each contained in a separate script. The scripts are written in the most appropriate language for the task at hand; languages contained in this codebase include python, R and bash.

The data\_preparation directory comes with a set of scripts to automatically pull, merge, clean and commit data from a vendor. Users are encouraged to modify or augment these scripts as they see fit. Users will benefit from adhering to the script file naming convention adopted by this project.

### Script Naming Conventions

One can identify the sequential position of a script in the data\_prepararation directory by the three-digit prefix code of the filename. The first digit refers to a basic step in a data preparation workflow. Basic steps are 0xx: system setup, 1xx: data collection, 2xx: data cleaning, 3xx: data warehousing.

Substeps are context-specific and allow us to break us a monolithic workflow step into subcomponents. In general they are meant to be run one after another. That is, 20x\_example.py is a workflow step prior to 21x\_example.py.

The final digit exists to indicate an logical workflow step that has been broken up for transparency. For example, 210\_Bloomberg\_merge.py is logically part of the same workflow step as 211\_Reuters\_merge.py - that is, it is part of the **merge** substep in the monolithic **data cleaning** step. However, for transparency reasons we break each source up into a separate script. This helps avoid excessively complex scripts and aids in debugging.

┌───── basic workflow step  
│┌──── substep  
││┌─── substep segment number  
│││ ┌─ intuitive description  
000\_description.sh

### Warehousing

This project stores all equity data in a single SQLite file, stored at backtest\_data/db.sqlite. SQLite is [well suited](https://sqlite.org/whentouse.html) to the use case of a central data store for single-user SQL access to a structured dataset.

Individual backtests pull subsets of the central database and analyse the subsets.

### Scheduling

It is anticipated that, once the above workflow has been refined and tweaked, the scripts are run periodically via cron to ensure that new data that enters the system is automatically merged, cleaned and committed. cron jobs should be set using the standard crontab -e command. A sample crontab file is available at backtest\_workflow/data\_preparation/010\_crontab.backup. The `crontab’ file should be clearly commentated and serves as a snapshot of the scheduling of your workflow.

### Backtesting

The backtesting system runs on top of the data warehouse outlined above. Workflow-wise, all processes downstream from the db.sqlite creation step are housed in their own project subdirectory, with backtests going to the ~/backtest\_workflow/backtests/<project\_name> subdirectory and exploratory data analysis and visualizations going to the ~/backtest\_workflow/visualizations/<project\_name>. It is expected that each backtest is unique, so there are no prescriptions on the structure of these directories. Users should include a README.md file in each backtest folder to orient an auditor.

In many cases, an entire backtest could be housed in a Jupyter notebook or Rmd file. This would enable the backtest construction, analysis and report generation to be housed in a single document. We include a sample Rmd backtest at backtest\_workflow/backtests/example.

# 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 August 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 August 2018, but users are free to fork and develop it on their own. Wherever possible, the code is written in R, python and bash to ensure compatibility with generally avaiailable 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.

### Milestones and Project Deliverables

The end goal of this project is to have a complete equity backtesting workflow by August 2018. This project is being built solely by Riaz Arbi.

|  |  |  |
| --- | --- | --- |
| Date | Milestone | Status |
| October 2017 | Set up server with necessary dependencies | Complete |
| November 2017 | Build Bloomberg Excel VBA Workbook to scrape data | Complete |
| January 2018 | Scrape Bloomberg terminal for data | Complete |
| January 2018 | Merge raw files into single csv files | Complete |
| February 2018 | Clean, join and interpolate data | In Progress |
| February 2018 | Transform raw data into Sqlite file | In Progress |
| March 2018 | Build out code to control for backtesting biases (look-ahead, survivorship etc) | Not Started |
| April 2018 | Perform a case study backtest | Not Started |
| May 2018 | Debug, refactor, refine | Not Started |
| June 2018 | Add additional data sources: iNet, Datastream | Not Started |
| July 2018 | Replicate case study backtest on alternative data and compare differential results | Not Started |
| August 2018 | Create second backtest and document workflow steps and benchmark timing | Not Started |

## 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.

1. Software

* At a minimum, you need to have python 3.5, R version 3.4.3 running on a modern x86 linux distribution.
* A complete browser-based software suite can be set up automatically using this repo <https://github.com/riazarbi/serversetup>. The scripts contained in that repo will, among other things, set up RStudio Server, JupyterHub with Python 3.5, NextCloud and secure the server with approproate firewall rules. It can also set up a Tor hidden ssh service, private key access and automated standard user creation.

1. Data

* This project presupposes that a user has access to a data vendor subscription. It contains VBA enabled Excel workbooks that can automatically extract relevant data from the Bloomberg Excel Add-In. It is anticipated that support will be added for similar automated VBA-based extraction of DataStream and iNet sources.

1. 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.