# Quant Data Engineer - Data Exercise

Brendan Dowling

## Preface

The task, despite its simplicity has considerable depth and various considerations. Below, I've outlined my approach and considerations relating to the engineering of the application, the process of cleaning the raw price data, and the computation of the historical volatility. In each, there are numerous further avenues for improvement which I discuss below. In particular, with more time and know-how, the estimation of the historical volatility likely can be improved considerably.

## Engineering

Roughly, the project is structured as such:

- `data`: Templated directories where data is dumped, stored, etc. (can imagine as an S3 bucket).
- `logs`: Directory for storing logs (formatted as JSON).
- `scripts`: Batch scripts to be run one-off for fully processing old data.
- `tests`: PyTest test modules.
- `volatility_estimator`: Core application code with:
  - `cleaner.py`: Functions for cleaning the price data.
  - `config.py`: Global config, configurable by the `.env` file or setting environment variables in the script call (or otherwise).
  - `estimator.py`: Various methods for estimating the historical volatility (implemented as subclasses of some ABC; equally could have used Protocols).
  - `logger.py`: Code for setting up the global logger.
  - `process.py`: Core application code used in `app.py` or the scripts.
- `app.py`: WatchDog app for listening on the `data/load` directory for new price data.

I broke the problem into two parts:

- Batch processing of old data
- Processing of new data

For the former, I setup two separate scripts. The first loads the price data for each stock, cleans it (see next section for details), and then saves the data as a parquet, sharded on the dates of the trades. This sharding enables efficient access for processing new data, as will be described shortly. The second script computes the rolling historical volatility for the all the stocks (see last section for details). The decision to split this batch processing into two scripts is sensible since in principle we might want to run parallel operations on the cleaned prices after they're processed (e.g. some other batch script). Both scripts work in a sequential fashion but one could parallelize them reasonably easily.

For the latter part, I setup a simple WatchDog app for listening on the `data/load` directory and processes any files dumped into it. The handling of new price data is reasonably straightforward, applying similar

cleaning steps. If there is no stock split for the stock on the new date, saving the new price data is trivial and doesn't require accessing the previous days data thanks to the sharding scheme. However, when there is a stock split, we do load back in the old data and adjust the price accordingly. I mocked stock splits as just some global dictionary. Obviously, one would likely have access to some other more principled look-up or data-frame. The code is designed such that swapping out to using such a lookup is relatively straight-forward.

The code itself is generally fairly self-explanatory. With more time or in a production environment I'd write full docstrings for each class and function, and better READMEs. Similarly, for the sake of speed, I mostly engineered the application in an interactive fashion rather than in a TDD approach which I'd usually apply. I added in a trivial `pytest` module and test function just to demonstrate structurally how and where I would add tests. The project is setup as a simple `poetry` package and I used `ruff` for keeping consistent formatting and for basic linting. In a production environment one would likely incorporate these into a CI/CD pipeline (e.g. via GitHub actions, etc.). To product-ize the application, it would likely make sense to Docker-ize the application.

## Data Processing

The nature of the data made processing/cleaning the data non-trivial. I read a good bit on some existing, principled approaches in the academic literature on ultra-high-frequency trade data cleaning. Several papers pointed to the methodology of "Realised Kernels in Practice: Trades and Quotes" (Barndorff-Nielsen et al., 2008), which I employed. Roughly, the approach then to cleaning was as follows:

- Filter out any trades occurring outside trading hours (8:00 to 16:30).
- Filter out any zero prices.
- Combine identical timestamps via the median.
- Remove entries where the price deviated by more than 10 mean absolute deviations from a rolling centered mean (excluding the observation itself) of 50 observations.

Additionally I added a date column (extracted from the `ts` column; could have used file name). I also added handling for stock splits, ensuring alignment between all old prices with ones post splits (i.e. treating latest as the base).

There's obviously endless improvements that could be made; there are several interesting research papers applying different means, most focusing on outlier removal. Another approach to outlier removal I prefer is using a double MAD approach; I used this previously on eCommerce data in my previous job. It's nice since it works well with asymetric data and multimodalities. I decided to keep things simple with the MAD approach above since double MAD would likely require a decent amount of analysis and tuning before applying to tick level data (market-micro-structure noise starts coming into play here, discussed in next section).

Regardless, the results are accpetable with no immediately obvious data issues. See Figure 1 for the final cleaned stock prices.

## Volatility Estimation

This is probably the more interesting aspect of the task due to the access to trade-level data. Standard methods involve working just on daily-level data (e.g. just access to the open, close, high and low). For example, I implemented the most simple historical volatility estimation method by just taking the (cleaned) close prices for each day and looking at the standard deviation. Even though this is a pretty poor estimator, it's useful as a baseline for comparing other, more complex methods. A simple extension on this I didn't implement might be just applying this procedure on tick-level data (though I'm unsure this is well principled). I did also implement the so called Yang-Zhang estimator which, loosely, works

as a weighted average between the intraday and interday volatility, but uses just open and close prices (i.e. inefficiently uses our data).

Indeed, these two distinct forms of volatility are what add complication to the task. For instance, a stock could be extremely volatile during the day but always closes about where it opens, and open where it closes. In such a case, the stock would have high intraday but low interday volatility. Obviously the converse can hold (and is probably the more common scenario).

One approach, building on the assumption the price follows in log an Ito process:

$$\log S_t = \log S_0 + \int_0^t \mu_u du + \int_0^t \sigma_u dW_u, \quad t \in [0, T]$$

to volatility estimation is to estimate the integrated variance (and hence vol) via the realised variance:

$$RV_n = \sum_{t \in \{t_1, \dots, t_T\}} (\log S_i - \log S_{t_T})^2$$

(which estimates the integrated variance of the asset price in the absence of MMS, giving some nice limit theorem results providing confidence intervals), and averaging over the 30 days (rolling, in our case) giving the ARV. However, this will severely underestimate the total volatility since it doesn't consider interday spot changes. The naieve adjustment I implemented was to take the sequential returns over the full 30-day rolling window (i.e. including over-night/weekend returns). This isn't very well principled, however, and perhaps a better approach would be a manual handling/weighting of the computed intraday volatility and interday changes. Regardless, the approach works reasonably well and aligns with the other two higher-level approaches; see Figure 2.

However, there's almost certainly some non-trivial market microstructure noise due to things like price dicreteness and bid-ask bounce. Essentially, the price observations are themselves noisy (in a fashion we can't easily process/clean away). There are a few methods I encountered in a module I sat in on; in particular, the non-parametric realised-kernel estimator method (Barndorff-Nielsen et al., 2008) is quite popular and effective for handling the autocorrelations that may be present:

$$RK_{n,H} := RV_n + 2 \sum_{h=1}^{H} k\left(\frac{h-1}{H}\right) \gamma_n(h)$$

where $k : [0, 1] \to \mathbb{R}$ is a kernel satisfying $k(0) = 1$ and $k(1) = 0$; the Parzen kernel function is a standard choice. I also looked at some more recent work here which uses point processes and splines for better intraday estimation which was pretty intresting. With a bit more time it would be interesting to implement these and benchmark them against the simple methods.

My research at the moment is centered around sequential Monte Carlo which is an adept method at solving Bayesian filtering problems of which this task can be framed as an example. The simple example might be the stochastic volatility model:

$$\begin{aligned} X_t \mid X_{t-1} &\sim \mathcal{N}(\mu + \rho(x_{t-1} - \mu), \sigma^2) \\ Y_t \mid X_t &\sim \mathcal{N}(0, e^{x_t}) \end{aligned}$$

(with $X_t$ the unobserved true volatility and $Y_t$ the log-returns) on which we could run SMC-squared or Particle-Marginal-Metropolis-Hastings to infer the parameters (either rolling or over full full history). Since we're in a Bayesian setting, the MMS issue is dealt with implicitly by infering the the true state/volatility. Furthermore, we can model the state dynamics however we want (not necessarily Gaussian), including incorporating jumps/regime-switching, and accounting for over-night/weekend changes. It'd be interesting see how such an estimator might perform, though obviously there'd be a decent amount of modelling work involved hence why I left just this sketch concept. Additionally, such SMC methods are generally quite computationally expensive which might be a consideration.

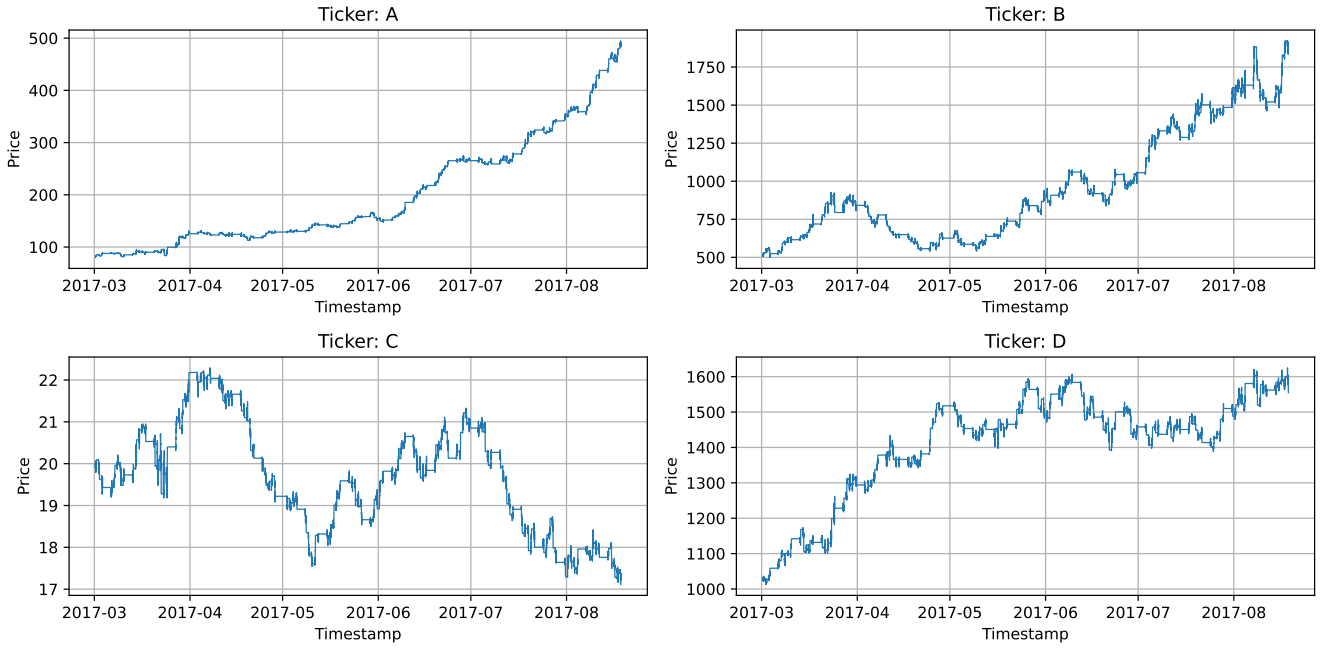


Figure 1: Cleaned stock prices


Figure 2: Volatility over time by implemented method