**Andrew Wu**

**Code Structure:**

The flow of this calculation goes through several steps:

1. Calculation Configuration: We passes configuration to instantiate the calculation, in this project we are passing in request date and frequency mapping. This would be easy to update in case we need to change the mapping in future.
2. Pre-Processing: This step is to parse the files, do some necessary validation (In the project we checked if there is any missed mapping from file), missing value handling etc. Basically in this step we want to get a df that is ready for process. In this project this step is done in get\_merge\_df method. If we want to extend the functionality in terms of parsing, such as column format convert, handling different formats of date, we only need to make change in this method.
3. Processing and get base level data: This calculation has 3 aggregation levels – Tranche level, Fund level and Summary level. At this step we generate the tranche level base. Saved the output in the calculation. (For purpose of caching in future, also all next level calculations will use this saved result as input).

At this step we used an infinite generator to create all the cash flow dates for all the tranches. It shows amount received, amount remain and receive date for each tranche. The logic about base calculation is in get\_tranche\_data method. So when we call the **evaluate** method from the calculation instance, it will do the get\_merge\_df and get\_tranche\_data for us, returns a tranche level df and saved the calculation result.

1. Calculate time to liquidity: I built this as property in this calc. In case we want to derive other statistics from the base calculation that is not related with liquidity calculation, we can simply write the logic in another method or property. So each calculation is independent with each other (but they share the same base data). In this project I am calculating the weighted average time to liquidity at fund level and at portfolio level.

To make sure we always have same API, although portfolio level liquidity is only a number, I am returning it as a separate df, so a calculation always returns a df.

Another good thing about making this as a property is lazy evaluation, for example if we only need to derive base level data, we do not need to go through all the steps for liquidity calculation.

1. Visualization: Usually the backend part should stops at Step 4, it always returns a df and we can serialize it and pass it to a message broker and send to BI tools or to store or to other applications.

I believe the visualization part should be done separately with the backend calculation, so I did not include it in the calculation class. In this project I built some simple BI at tranche, fund and portfolio levels using a facet grid.

**Assumption of Calculation:**

Two assumption about the dataset, which should be clarified and checked:

1. Gate for Fund 5 is blank, I assume it means 1, or better to check if this is a defect from the dataset.
2. Gate for Fund 10 is 33%, I assume it is 33% exactly instead of assume it is 1/3. As the result in this calculation for Fund 10 we need 4 withdraws, 33% for first 3 withdraws and 1% for the last withdraw.

**Executive Summary:**

Portfolio level time to liquidation is 507 days. For a given calculation based on request date of 2017, 5, 31. We can see the bigger the fund name number, the longer it takes to liquidity.

**Can do better:**

1. A separate Test file should be built, to perform each individual unit test for the functions, generator built in this calculation and maybe integration test as a whole. We can patch the input using a context manager.
2. Maybe should build a base class for calculation just to define a common API, and make every calculation inherit from it. For example every calculation should have evaluation method to generate base data set, we can do something at class level such as to perform run-time check, load dataset from cache and save calculation result.
3. This write up should be shorter.