Calvin Thomas [2/12]

Say a unique china piece is a one-time use case specific model for a targeted market arbitrage data set of uniquely identifying ticker symbols as data objects for each row (highly valuable, fragile outside state). **Without portability, fundamentally non-reusable outside its state!**

OOP: We want to decompose this unique china set (abstracted data object) into a set of data-object code-components (total symbol list df for arb). If it is gilded together with gold, as is with often old china sets for aesthetics (or in this case new introduced code complexity to retain original functionality in model maintenance cost), we need to consider the complex legacy web that holds it together. Now if we scale a modular OOP code, there will be issues as the complex legacy object web grows and mutates.

Functional: We can address modularity and scalability with functional programming. Instead of a unique china set, take a car door as a specific portable model as example. Although complex, a car door or even engine can be broken down into individual object-components with correct schematics. A car door (simpler example), can be broken down up to the rubber lining between the window gap, or car door lining-manufacturers make and material! Once you dive into that granular level of detail, the total picture of the shape (OOP) of the car, and it's individual purpose (rubber-lining, car-door lining to the total car) in functionality (car will: "go", "stop", "accelerate forward"); as the modular code functionality purpose to the total program, is lost to the programmer.

See Toy Example att. (of below) in separate .ipynb file "iso_env_toy_ex.ipynb" in Functional vs. OOP repository. Simplest toy example of ML lab iso. env. processing as bonus.

First create a unique isolated env. for each worker wherein up to drilling down to each iso. env. folder acts as its own lab experiment with assoc. daily json'ed data folder that you then use to separately train test and effectively split your tasks down to; Ex1. One main root node (as its own separate indep. task manager, to further partition ea. ML for data independence from any indiv. ML benefitting in data-dependency that skews results, tight CI and even tinier p-values). Multiprocessing splitting tasks across each ML intelligently partitioning and data handling on its own specific contributory factor alpha. This may require testing several dozen ML, DL, NN, AI, Quantum to find the right mix. Aggregating your Machine Learning abstraction layers of combo functions together, ea. its own strat for a weighting or optimizer blended mix.

Use-Case: Any portable abstracted "data object" [fill your data] of ensemble grouping identifying unique tickers for each strat, portable OOS "data object" testing (ex. simplest granger causality,) to the next strat or market ensemble grouping of assets.

Therefore: a highly-robust python-supported, hybrid-internal "bubble-environment", and measurable standardized reporting ("Firm-wide standard protocol CI/CD best practices & protocols" internal-programming std. protocols guide), can simultaneously handle both functional programming modularity & functionality to scalability; while OOP maintains the shape of the object and total picture for the programmer.