**ML Model Update – Variational Autoencoder (VAE)**

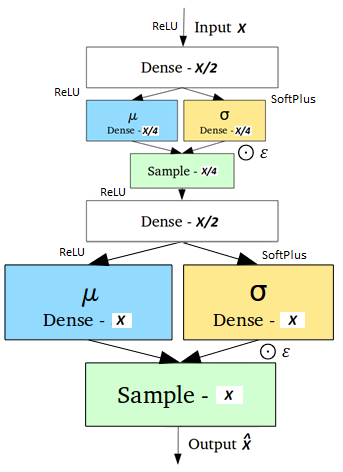
**Purpose**

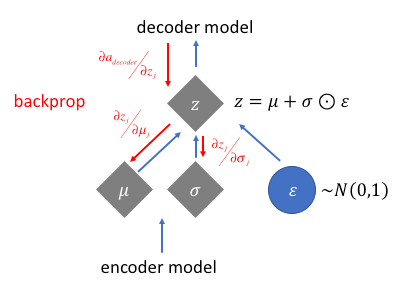
Publicly available financial data are scarce and in small numbers. This applies especially to historical price data, which are also low in frequency and breadth. This is a major disadvantage for training machine learning models for generating investment signals. Models trained on historical price data run the risk of overfitting the limited amount of data available based on past time-series behavior for only one path, even though future price actions may not behave the same way.

To solve this problem, the goal of the variational autoencoder (VAE) is to use an ML model to generate time-series data for any asset on a multitude of paths that vary slightly from the actual historical path, while maintaining the same similarities of characteristics. These generated paths must be representative of possible price actions for a given asset that are genuinely feasible. Ideally, the VAE will generate thousands of paths that can be used to train an accurately fit ML investments model.

Characteristics used to benchmark the generated paths will be discussed in this report. *Further suggestions for measurements are welcome.*

**Method**

The current VAE model is made up of a neural network that takes in a certain number of assets as input nodes, ***X***. The data is compressed through a hidden layer of ***X****/2* nodes, then down to a latent space layer, ***z***, of ***X****/4* hidden nodes. The mean and standard deviation is sampled from the latent space ***z***, and using the reparameterization trick, the data is decompressed from the latent space through another hidden layer of ***X****/2* nodes, to varied output features of ***X*** nodes with mean and standard deviations characteristic of the input features.



**Analysis**

TensorFlow ‘dense’ layers are used to initialize the hidden layers and perform matrix multiplication with the weights. Initializing each layer as TensorFlow variables individually did not produce favorable results for unknown reasons – one possible explanation is due to the way values are stored in the variables.

The historical price data is first converted to daily returns to normalize across the dataset. The outputs are in returns, which are converted back to prices by chain-multiplying with the price on the first date within the training set.

NaN data causes problems to the model – The entire dataset is truncated to a date such that all features have price data available starting at that point.

Previous iterations of the model used features from various random asset classes, which produced outputs with characteristics unlike their corresponding inputs for many assets. This could be caused by similarities in behavior between certain assets in the same class that were unaccounted for. The current model tests for this, with four models being created separately that splits input features by asset classes: Currency, commodities, stock index, and public S&P500 equities. Outlier features from previous iterations have also been removed due to vastly differentiated behavior, such as the Chinese Yuan (partially pegged to a basket of international currencies) and the Hong Kong Dollar (pegged to US Dollar).





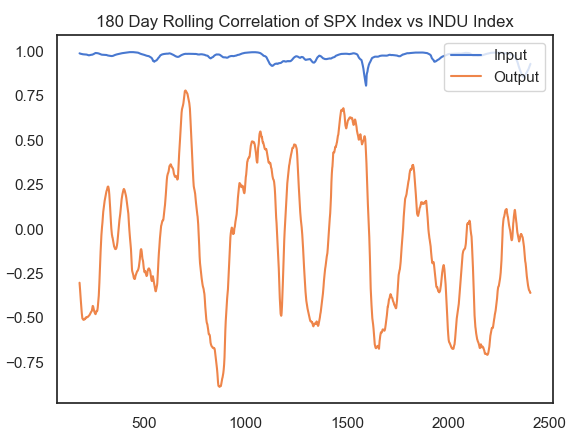


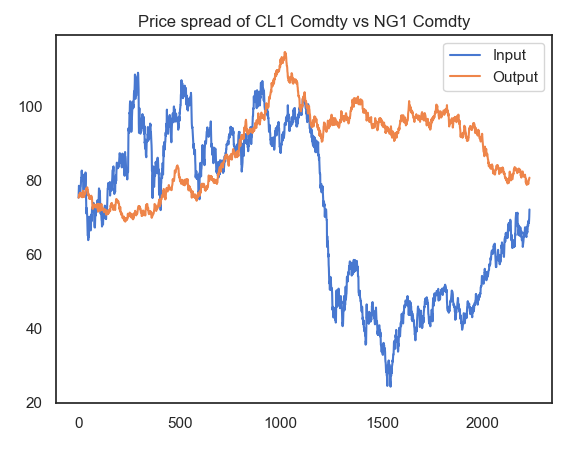


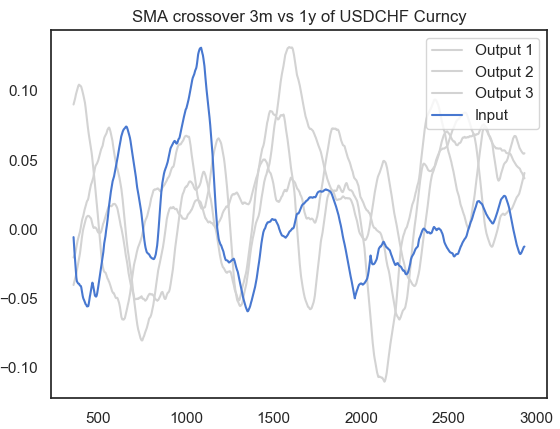
**Next steps**

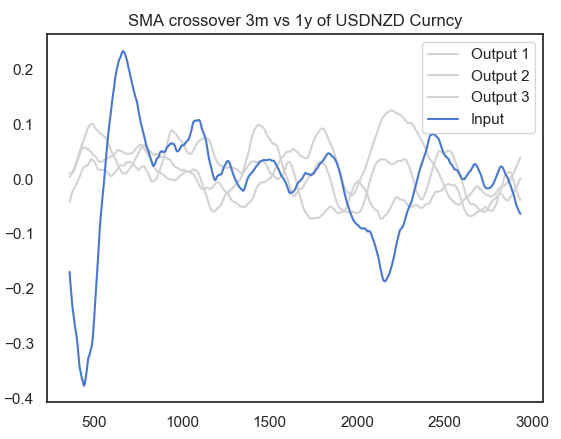
* Look into how correlation of features in the VAE affects the outputs (e.g. similarity of market cap, industry, region in equities; correlation of indices)
* Look into implementing VAE with latent constraints, tuned so the model produces desirable outputs – This may fix the variablility of the distribution of outputs
* Try stacking the separate VAE into one big VAE (questionable)
* Check KL Divergence for distribution of output vs input
* Check autocorrelation and heteroskedasticity
* Brainstorm more methods to monitor the quality of the outputs

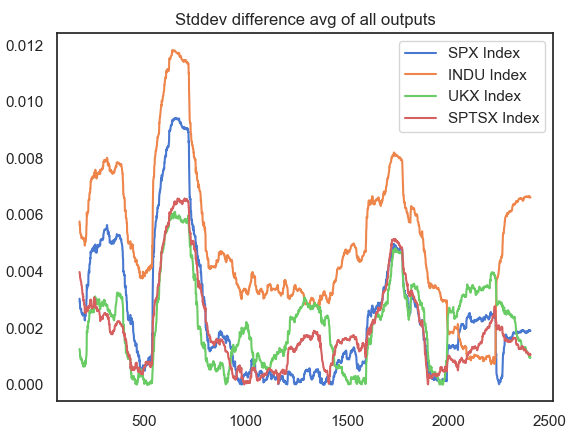
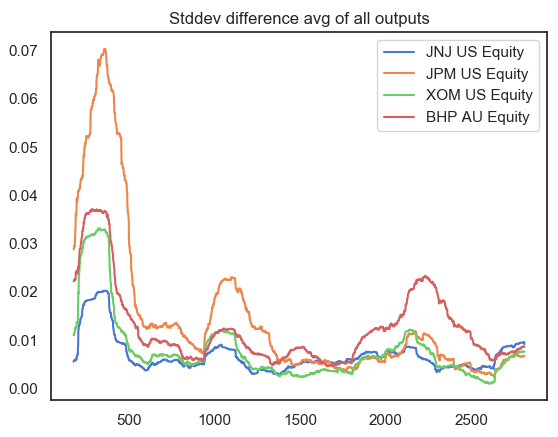
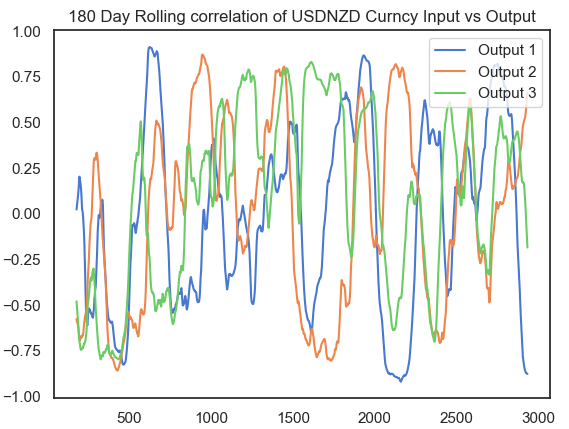
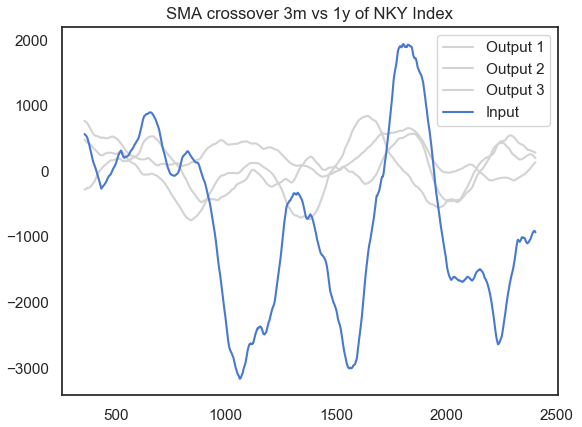
**Testing results**

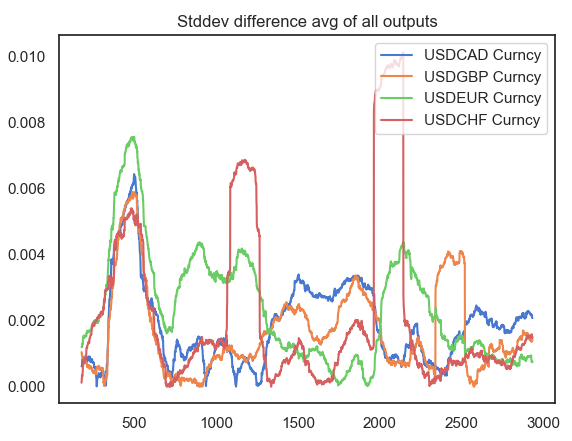
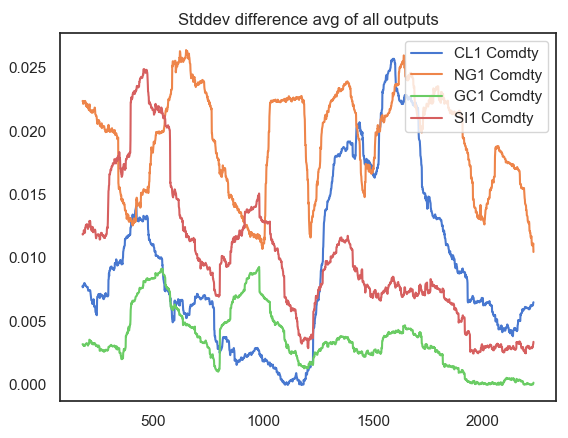


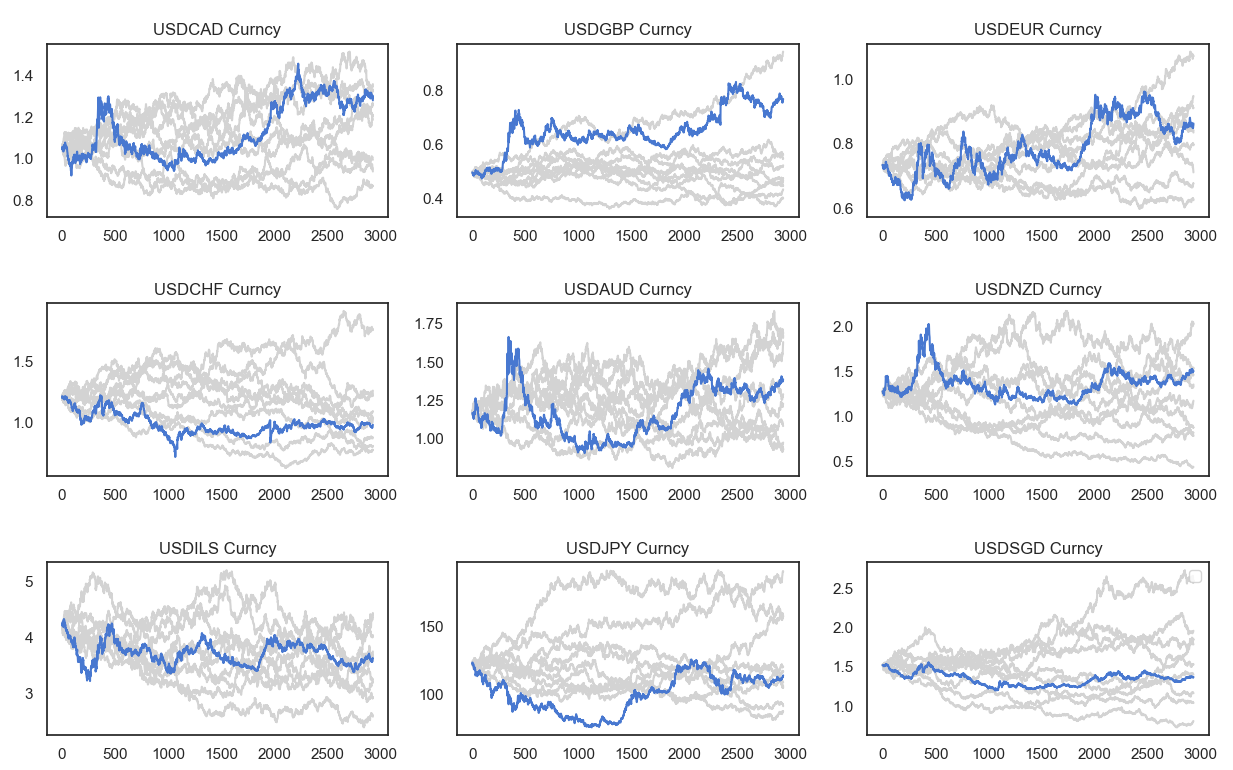


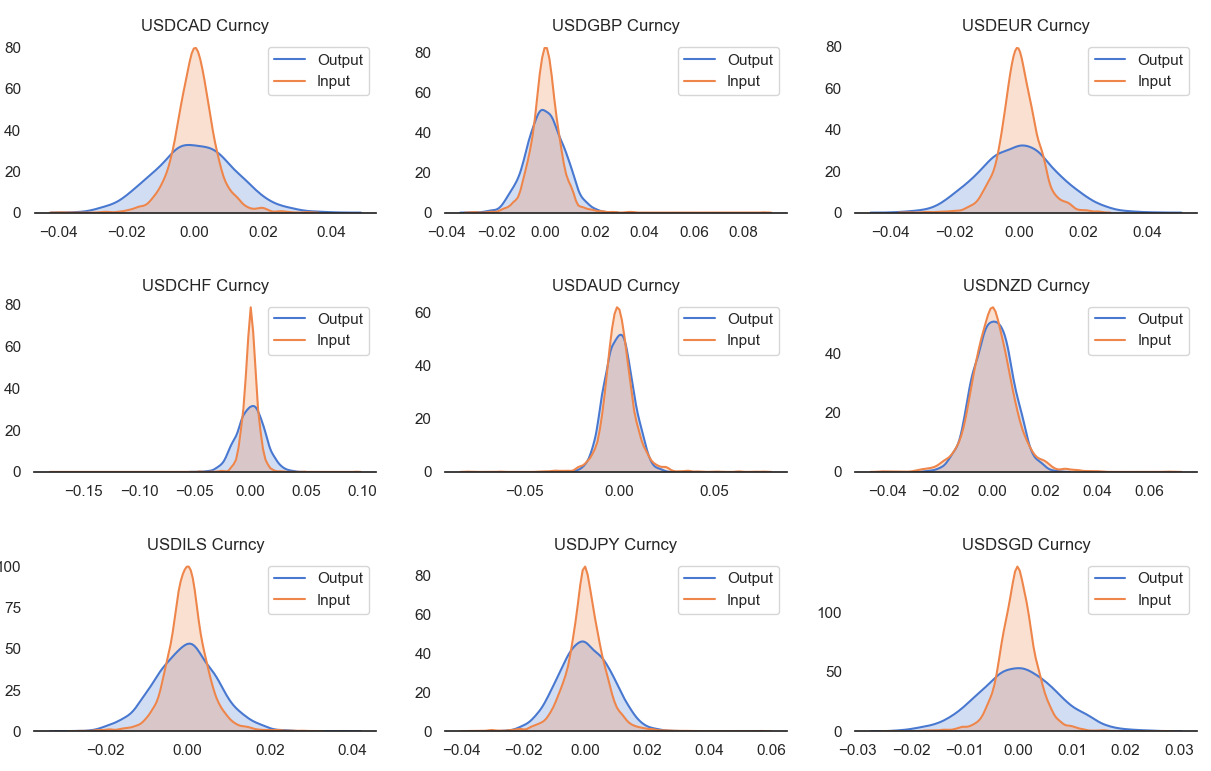


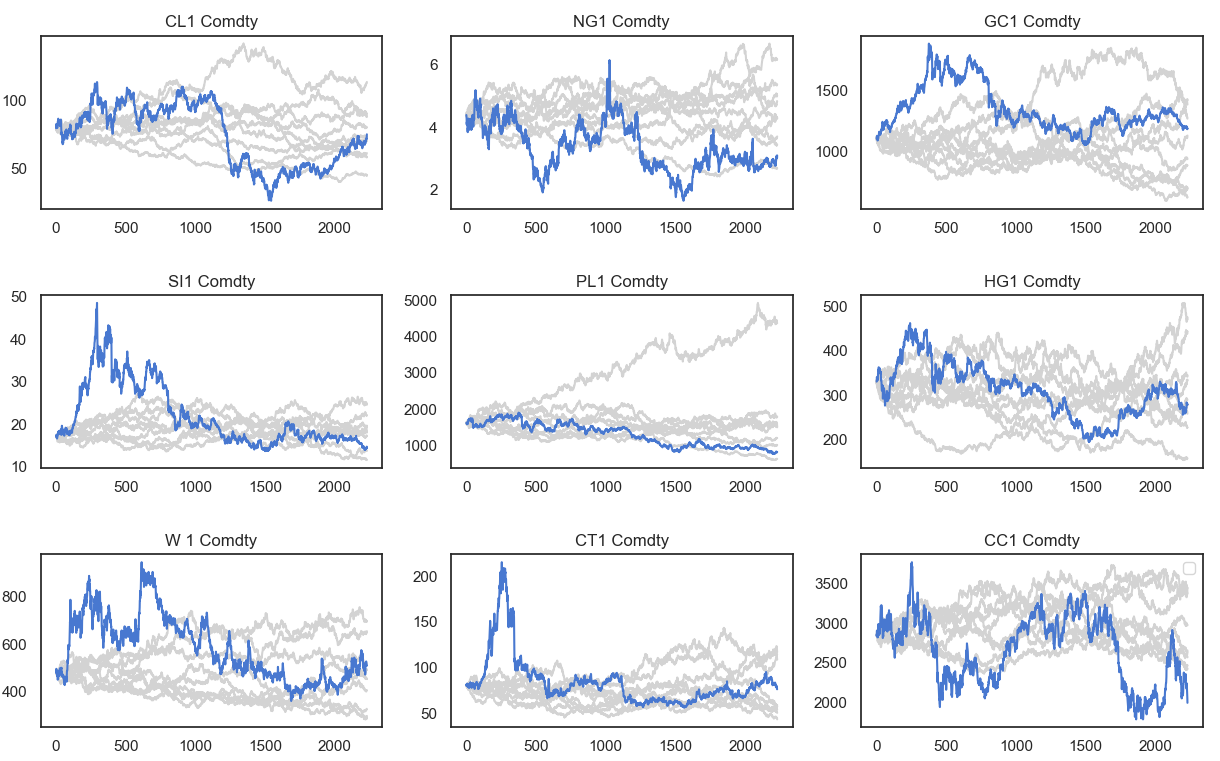


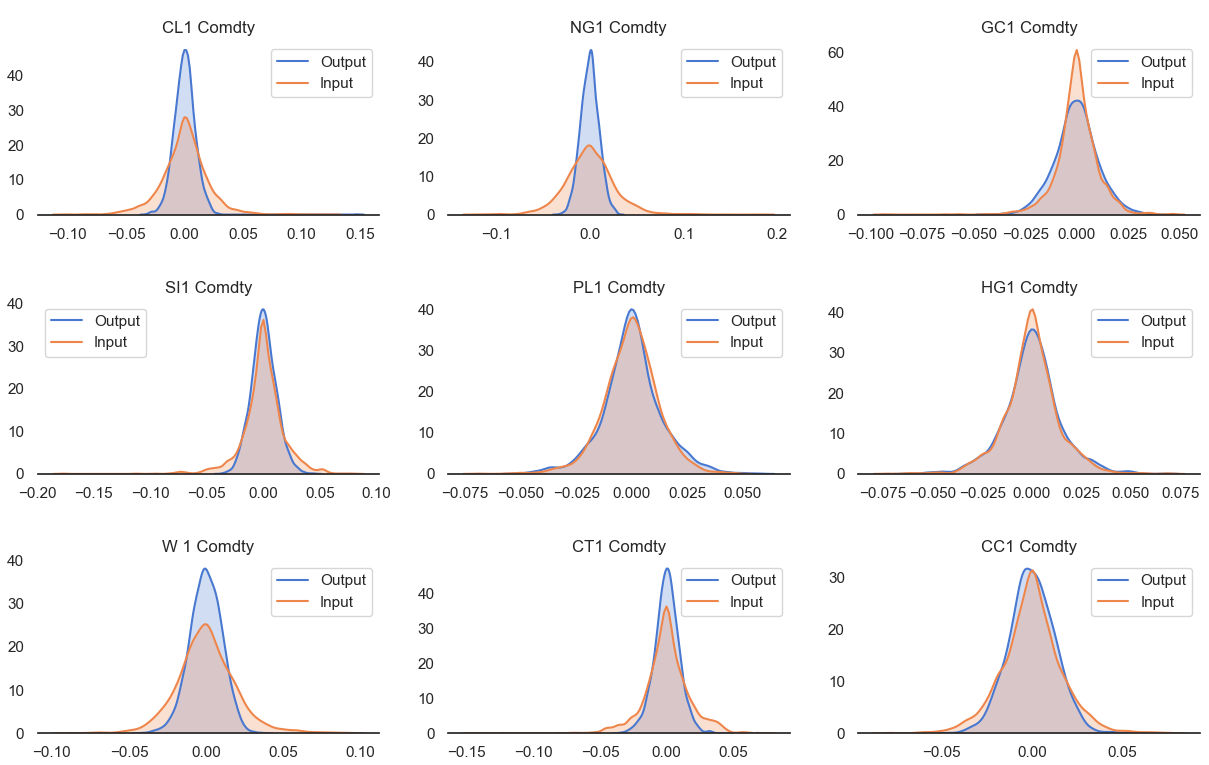


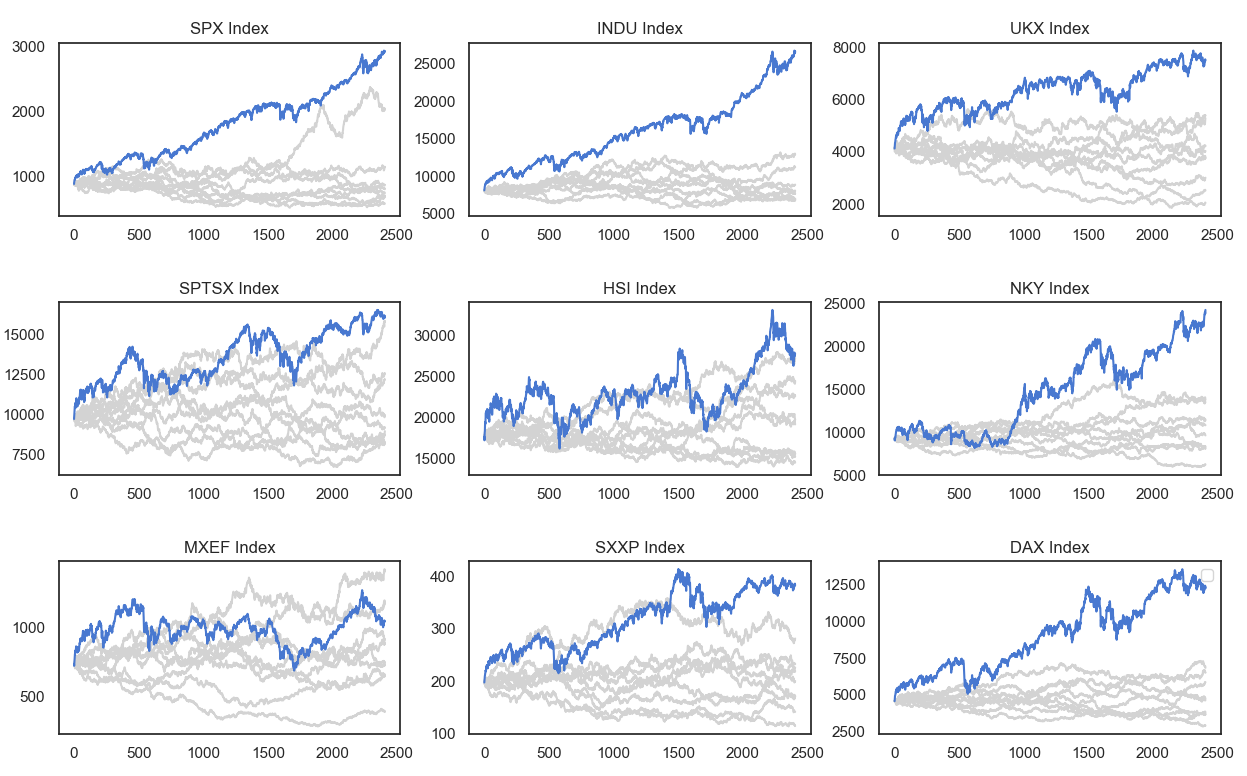


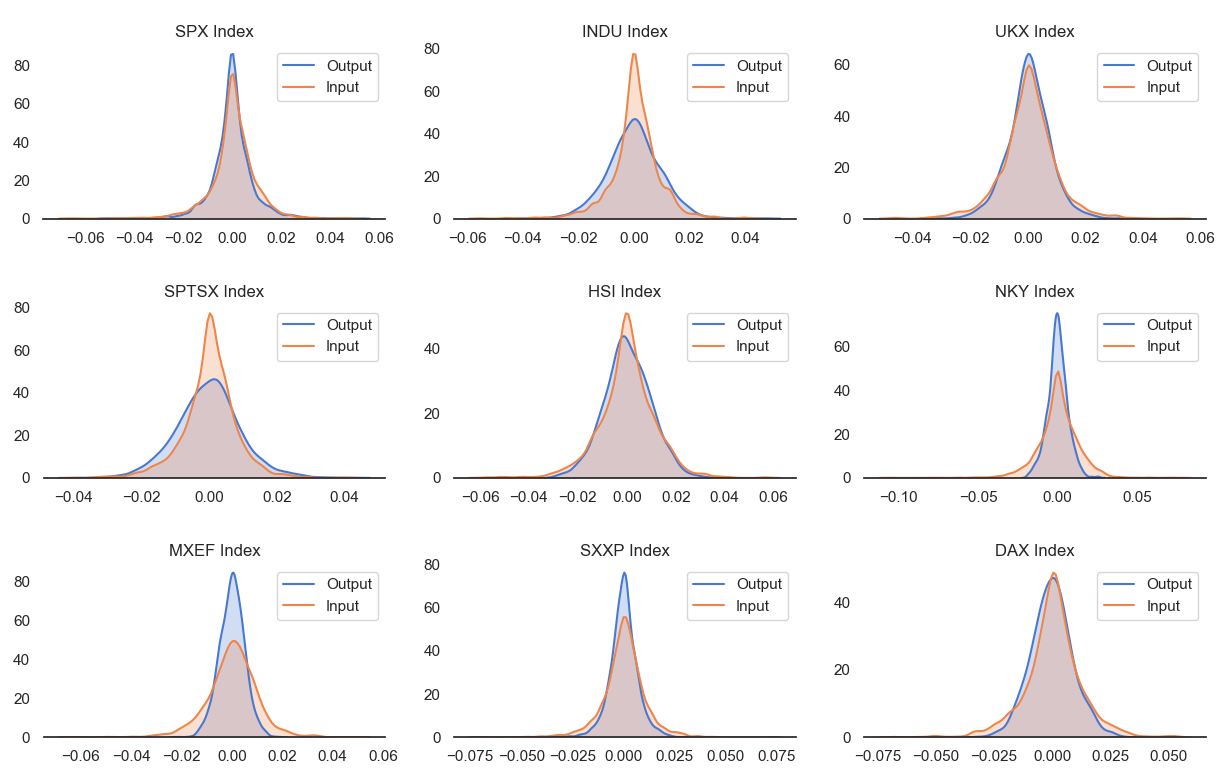


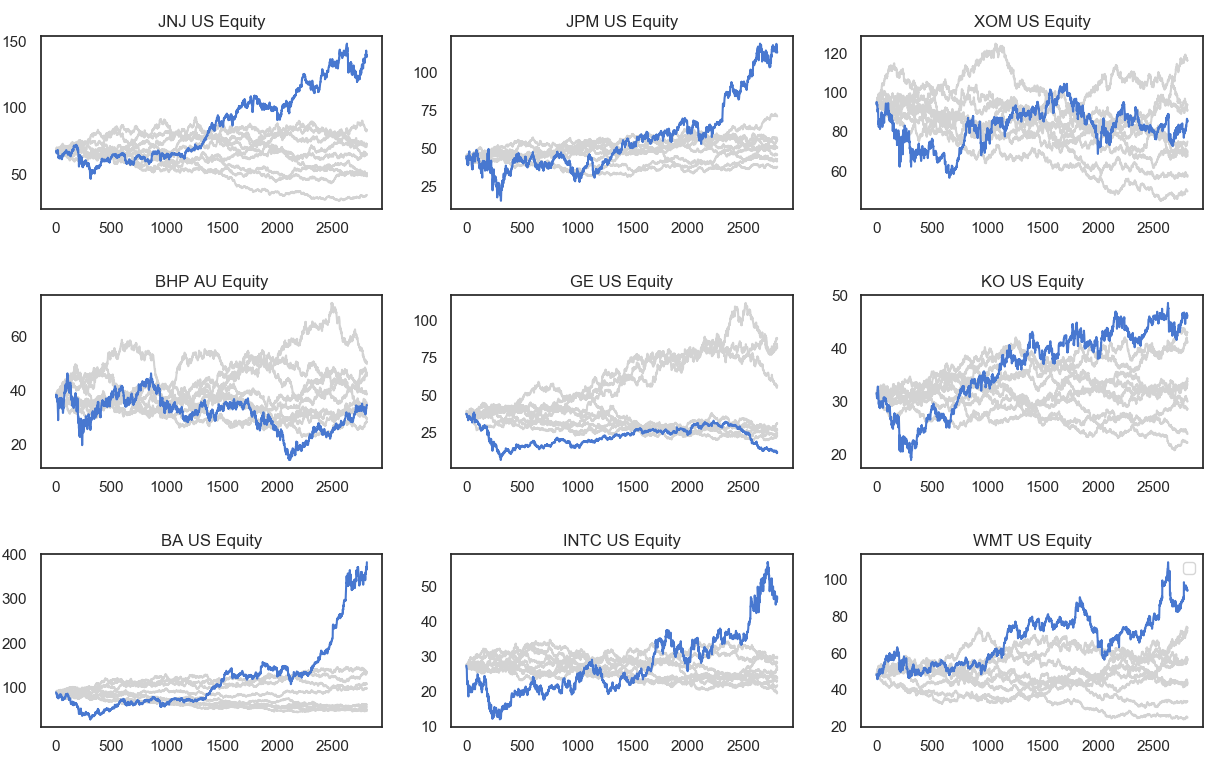




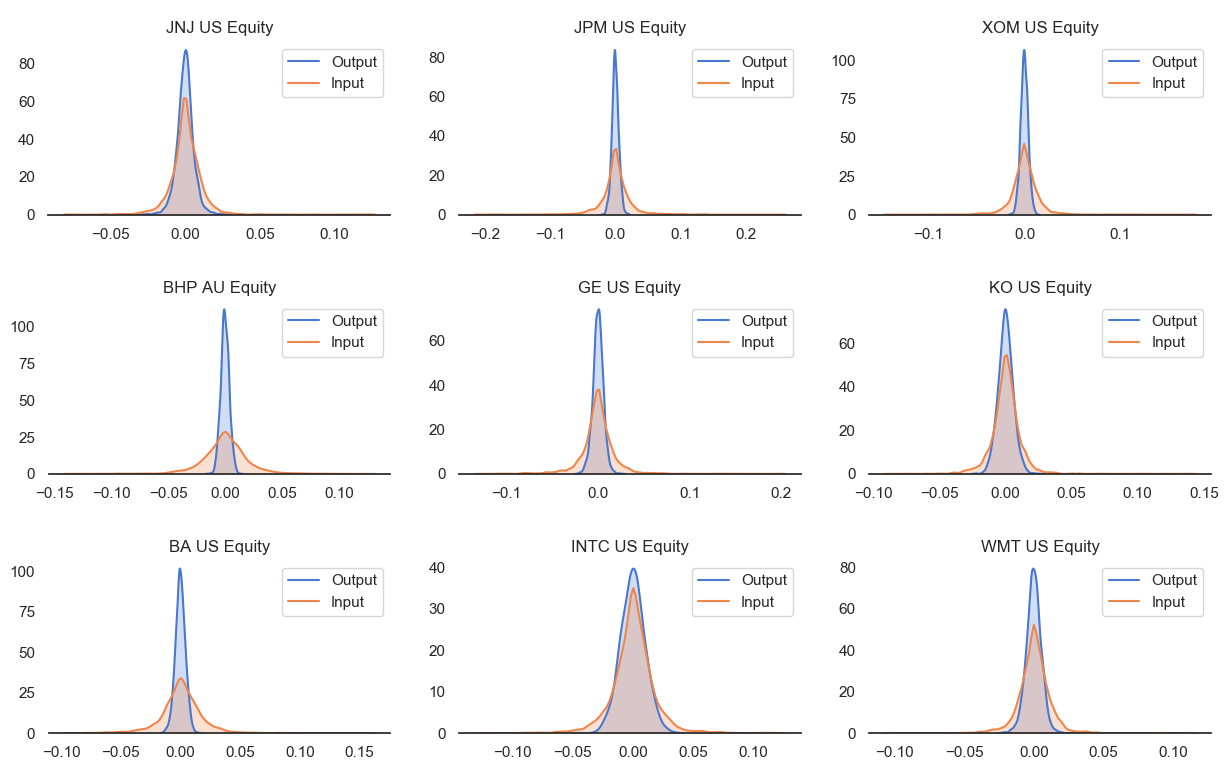




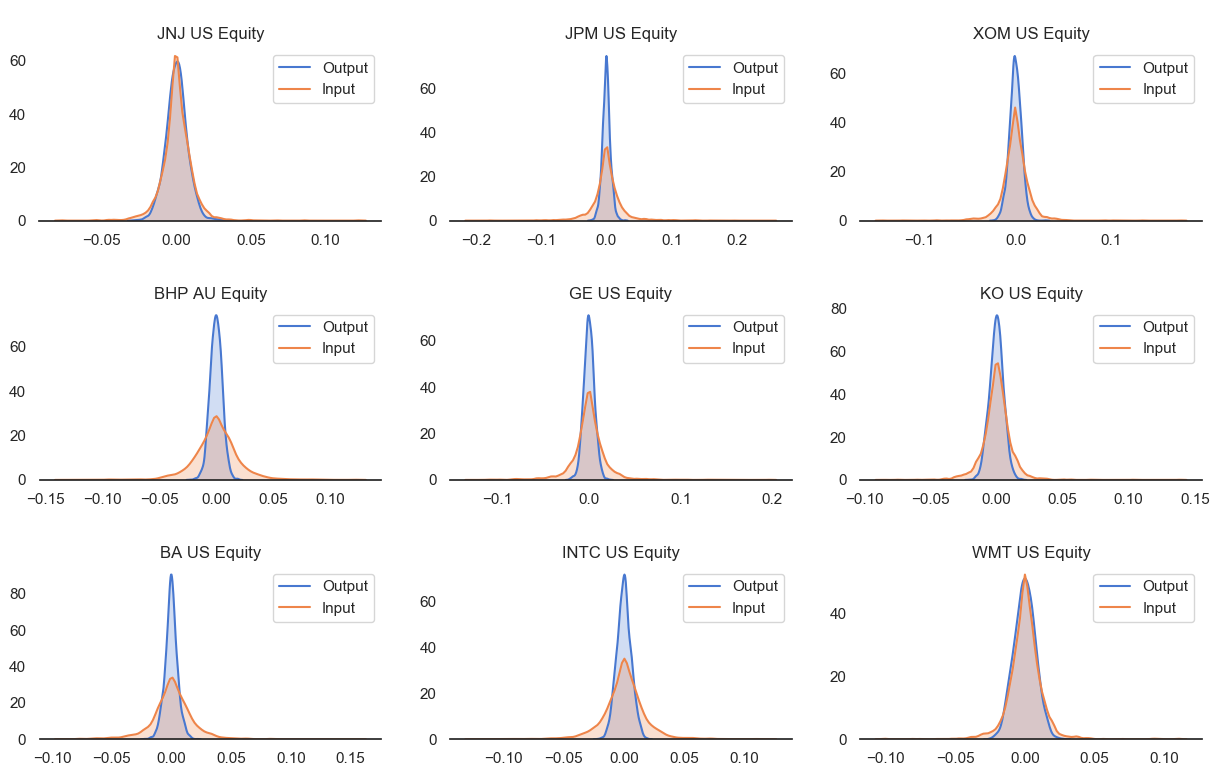




Iteration 1:



Iteration 2:



Iteration 3:

