Time Series Analysis Project: Precious Metals

Justin Chan, Ameer Dharamshi, Vivian Ngo

Abstract

A stable index is a basket of assets used to represent the current state of the market. In order to accurately depict the overall health of the market, it is important that the index has minimal variance so that it is not prone to excessive fluctuations or risk. In this report we construct a stable minimum variance index using a basket of currencies, cryptocurrencies, and precious metals. The index is updated on a quarterly basis as more data is crystallized. Prior to constructing the index, we begin by pre-processing the data, exploring the data to understand its structure, and analysing the assets individually and as a collection from a statistical perspective. Using our findings, we optimize the weights of the basket to minimize the variance of the index. This process involves the covariance matrix for the group of assets. This construction of the covariance matrix is rooted in the findings of the exploratory and preliminary analysis. Finally, we consider the predictive power of ARIMA models as applied to individual assets and we explore the idea of whether the random walk hypothesis holds true.

Data Preparation: Cleaning, Imputation, and Conversions

Perhaps the most critical step in the data analysis process is the data processing phase. Without a solid foundation to work with, any analysis performed is suspicious at best. With financial market data, the market exchanges produce open, high, low, close prices on each trading day. For the purposes of this project, we decided to use the closing price of the assets as they most accurately reflect the market sentiment for the trading day. In addition, certain assets only offered (XDR-USD, BTC, ETH, etc) close prices and thus for consistency, close prices are the best choice.

However, closing price data was not available for all Dates. Therefore, we need to consider which data points to remove or impute. After some conceptual considerations, it was decided to only keep data for the days that the US markets were active. As the markets are closed on weekends and holidays, there is a consistent pattern of missingness among the exchange traded data. If we were to impute these days, we would likely introduce significant serial correlation. Furthermore, the market is not open on weekends and holidays so, in a sense, weekends and holidays do not exist in the trading universe and thus it would not make sense to impute prices for these days.

Any other missing data points were imputed using the most recent available price as under the random walk hypothesis, the prior day is the best indicator of the next day. Note that the impact of this decision should be quite small because the number of imputed data points is very small relative to the size of the dataset. Finally, there was one specific instance of seemingly non-random missingness. There was one week in May 2018 where the XDR-USD data was missing. Given that this was an isolated incident, we once again applied our roll forward of the immediately preceding trading day.

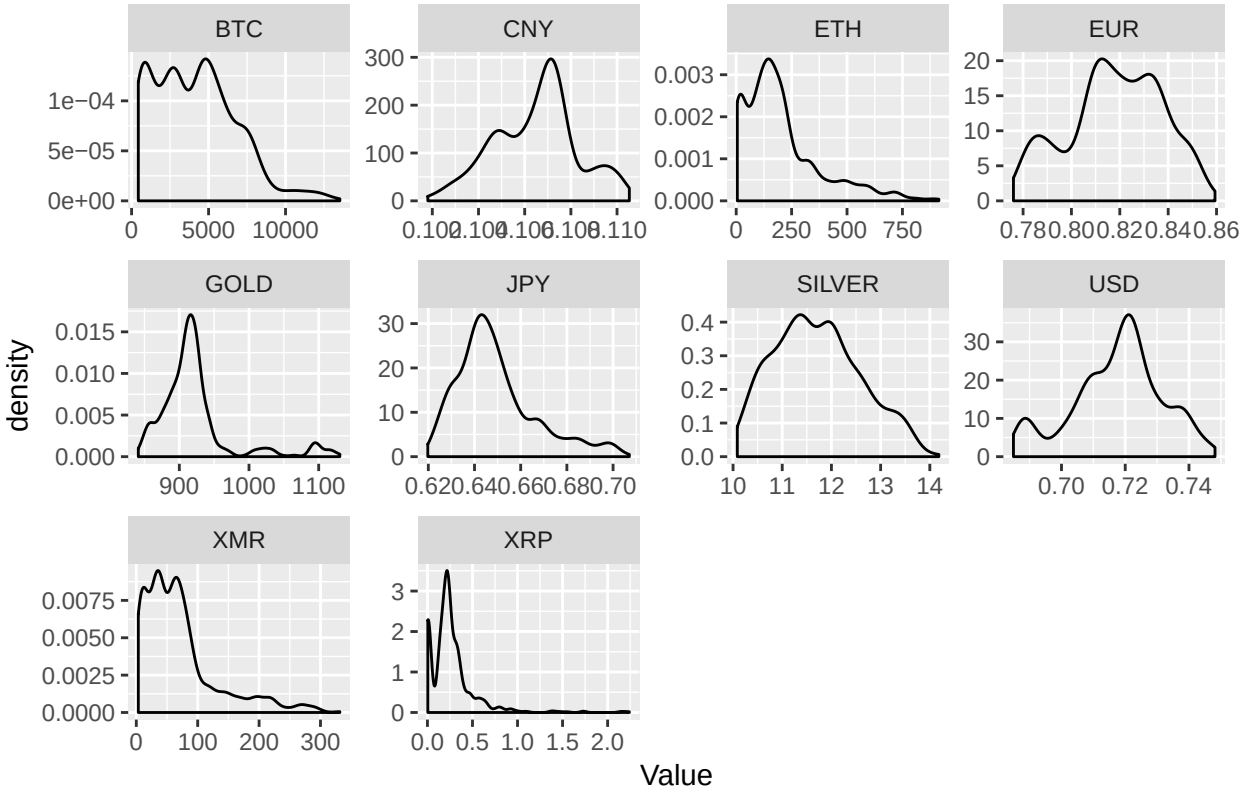
The other major step in the data processing component was to convert the data to XDR. We first checked to confirm that all assets were in units of USD. This was the case except for the CNY-USD data, which was reversed. We converted the data into units of XDR by multiplication (or division) with the XDR-USD column. In addition, a USD column was created by effectively multiplying the XDR-USD column by 1 (i.e. the column is already USDs in units of XDR).

Univariate Analysis

Before creating the index, it was important to explore the structure of the data and detect the properties that would be useful for our analysis. The first step in the EDA process was to simply investigate the raw prices of each asset. In order to see the distribution of the prices, we first plotted the kernel density for the prices of every asset.

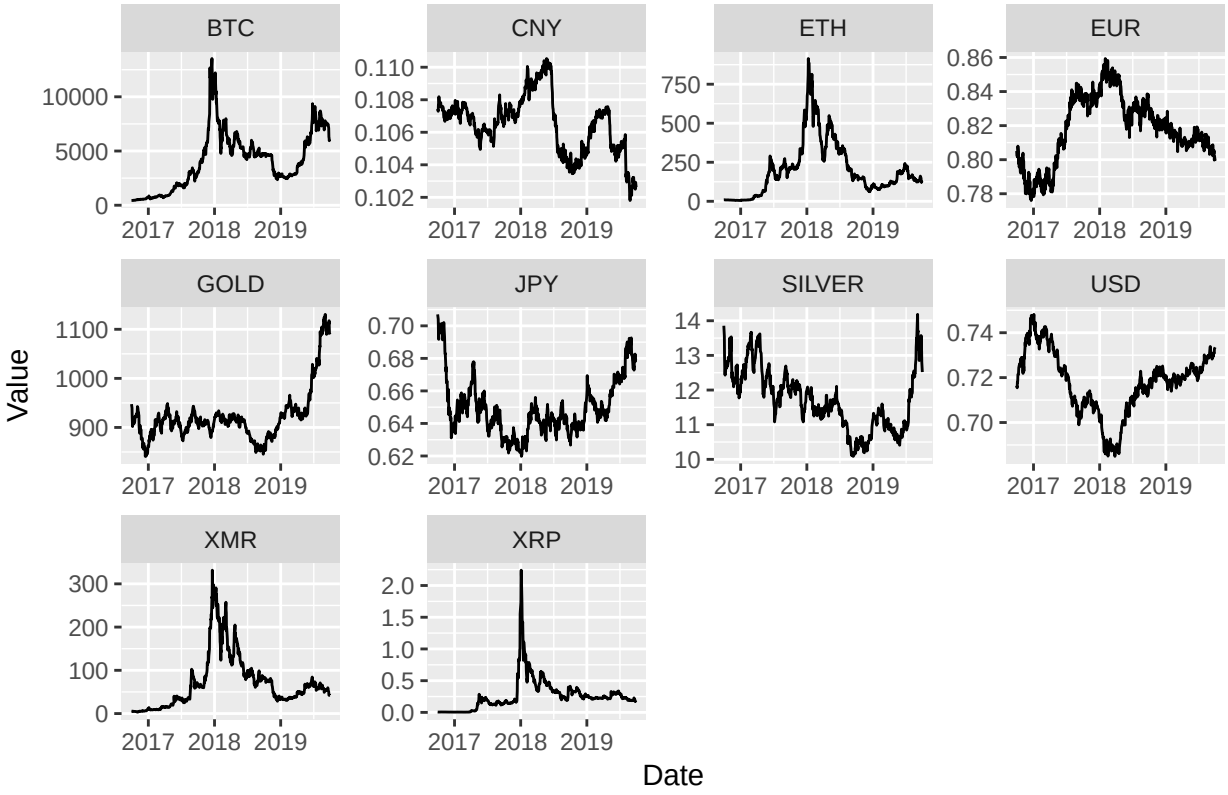
As we can see below, the distribution of the raw prices is either multimodal or skewed, or a combination of both, for every asset. As such, it is not likely that the prices of any of these assets follow a normal or t distribution.

Figure 1: Kernel Density Plots of Raw Price Data



We also produced the time series plots of the raw prices to see if they were stationary for any asset. As we can see in the time series plots below, the raw prices do not appear to be stationary for any of the assets. One main issue that indicates this is that there does not appear to be a common mean price that the time series returns to.

Figure 2: Time Series Plot of Raw Prices



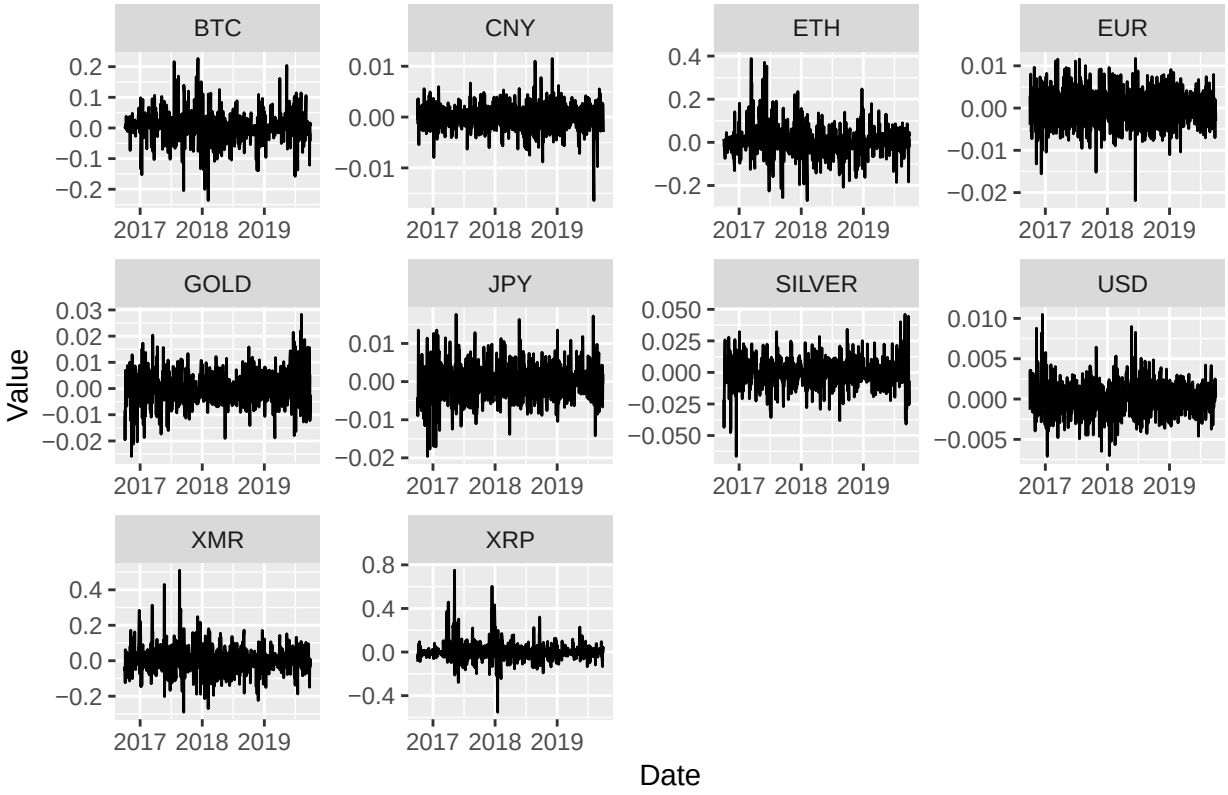
To formally test whether these time series are stationary, we also conducted a KPSS test for each asset. Table 1 below is consistent with our observations from the figure above. Because the KPSS test yielded p-Values of less than 0.05 for every asset's price series, there is evidence against the null hypotheses that the series are each stationary around a mean or linear trend. As such, we tried various transformations of the data in an attempt to find stationary series. One common method of introducing stationarity that we applied is the differencing operator. In addition, we also transformed the raw price data with the logarithm function as the concavity of the transformation is useful for spacing out data points that are grouped together and pushing Values together when they are very spaced out. Because of this concavity, we also expect that log returns will exhibit both of the useful properties of stationarity and symmetry.

Table 1: P-Values from the KPSS Test for different data transformations

	BTC	ETH	EUR	GOLD	JPY	SILVER	CNY	USD	XMR	XRP
Prices	0.01	0.010	0.010	0.01	0.010	0.010	0.01	0.01	0.01	0.010
Diff in Prices	0.01	0.100	0.100	0.10	0.064	0.081	0.10	0.10	0.10	0.100
Log Prices	0.01	0.010	0.010	0.01	0.010	0.010	0.01	0.01	0.01	0.010
Net Returns	0.01	0.092	0.020	0.10	0.078	0.092	0.10	0.10	0.10	0.040
Gross Returns	0.01	0.092	0.020	0.10	0.078	0.092	0.10	0.10	0.10	0.040
Log Returns	0.01	0.098	0.028	0.10	0.078	0.089	0.10	0.10	0.10	0.065

In Table 1 above, we see that the transformations were mostly successful as now the log return data does not reject the stationary null hypothesis except for the bitcoin and euro data. This conclusion is supported in the time series plots below as we can see that the log returns do indeed appear more stationary than the raw prices. However, the log returns of bitcoin do appear to have significant periods of volatility clustering and the euro data appears to have a non-constant variance.

Figure 3: Time Series Plot of Log Returns



The boxplots and density plots below also support the conclusion that the log returns are more symmetric than the raw prices. After constructing time series plots, boxplots, and kernel density plots for the net returns, gross returns, and differences in prices, we found that the log returns exhibited the most desireable properties.

Figure 4: Boxplots of Log Return Data

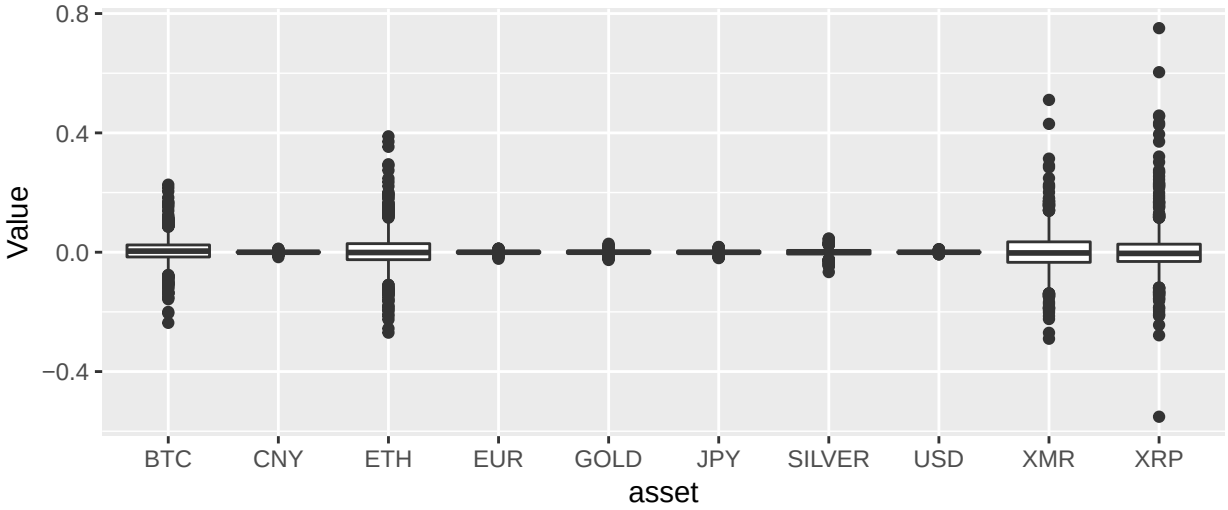
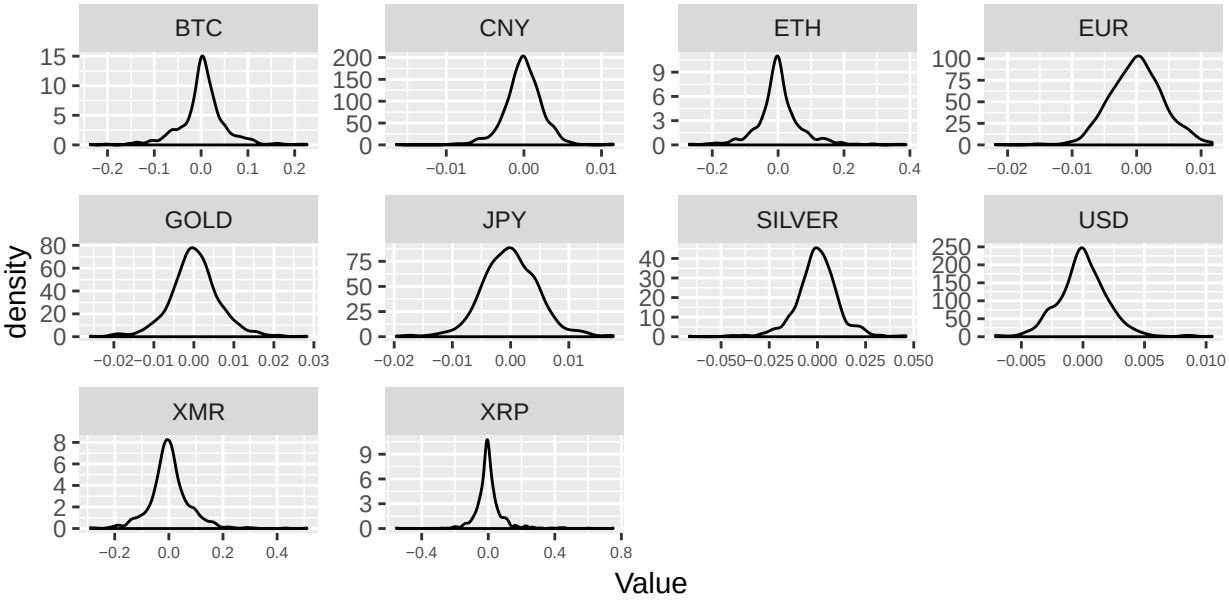
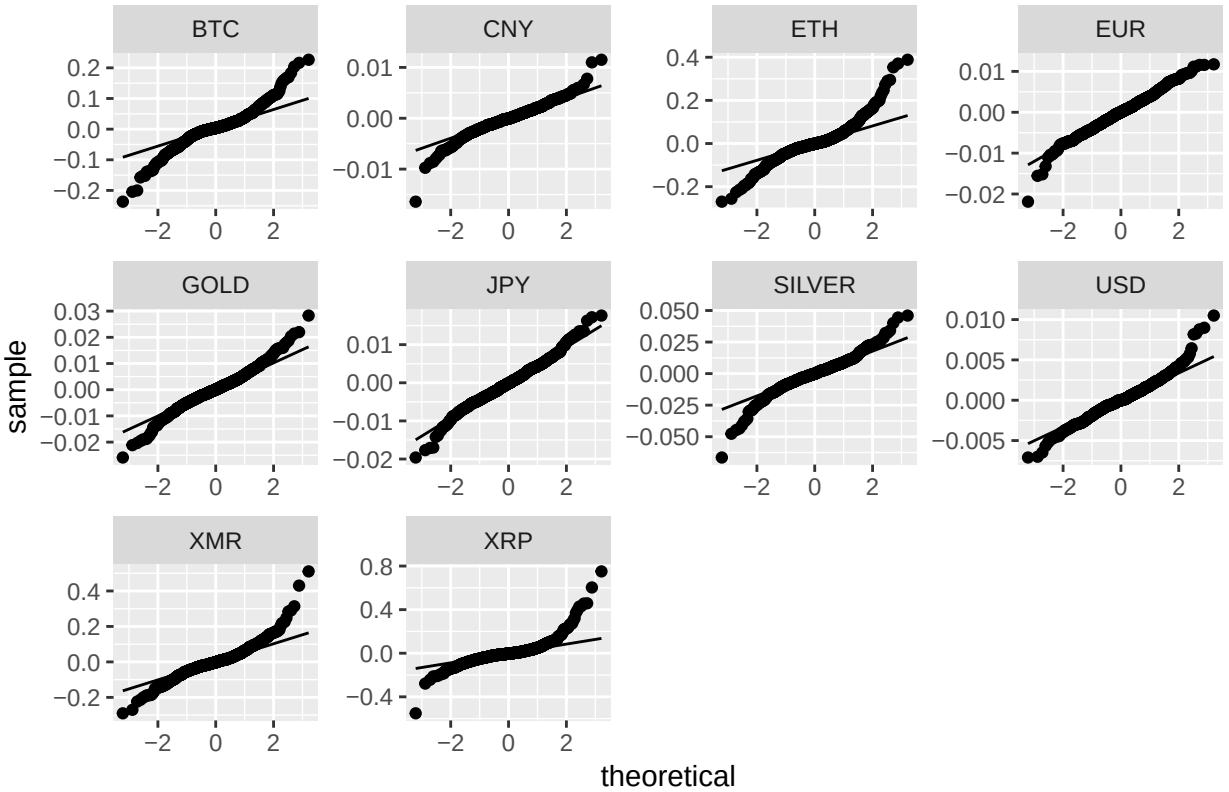


Figure 5: Kernel Density Plots of Log Return Data



Finally, it would be helpful to investigate what the distribution of the log returns is. Below is the collection of normal quantile-quantile plots for the log returns. Clearly, there is some indication of tail heaviness compared to the normal distribution.

Figure 6: Quantile−Quantile Plots of Log Return Data



The Shapiro-Wilks test for normality, when conducted on the log returns, also yields p-values that are much smaller than 0.05. This suggests that there is significant evidence to reject the null hypothesis that the log returns of the assets are normally distributed.

Table 2: P-Values from the Shapiro Wilks Test for Log Return Data

Asset	P-Value
BTC	3.00476578155941e-17
ETH	2.84873610700972e-20
EUR	7.41974546492912e-05
GOLD	1.99875251980354e-08
JPY	9.22703160339301e-06
SILVER	5.47333946607765e-14
CNY	3.35680350345856e-13
USD	5.13773867753304e-10
XMR	1.93024686892347e-17
XRP	6.36674506423801e-30

As we can see from the table above, there is some concave-convex pattern in most of the QQ plots. This is an indication that the log returns are distributed with heavier tails than the normal distribution. To capture this tail heaviness, we decided to fit t distributions to the data instead. Below are the sets of estimated parameters, AIC, BIC, and log likelihood values that are produced when the log returns are fitted to the standardized t distribution and the skewed t distribution.

Table 3: Parameter Estimates, AIC, BIC, and Log Likelihood for Log Returns fitted to Standardized t Distribution

Asset	mean	sd	df	AIC	BIC	Log Likelihood
BTC	0.005	0.122	2.099	-2567.276	-2553.38	1286.638
ETH	-0.001	0.49	2.01	-2095.43	-2081.534	1050.715
EUR	0	0.004	11.972	-6172.409	-6158.513	3089.204
GOLD	0	0.006	4.704	-5593.928	-5580.032	2799.964
JPY	0	0.005	7.629	-5969.042	-5955.146	2987.521
SILVER	0	0.012	3.682	-4751.079	-4737.183	2378.539
CNY	0	0.003	4	-7039.763	-7025.867	3522.881
USD	0	0.002	4	-7281.957	-7268.061	3643.979
XMR	-0.001	0.087	2.823	-1910.081	-1896.185	958.041
XRP	-0.004	1.682	2.001	-2001.309	-1987.413	1003.655

Table 4: Parameter Estimates, AIC, BIC, and Log Likelihood for Log Returns fitted to Skewed t Distribution

Asset	mean	sd	df	epsilon	AIC	BIC	Log Likelihood
BTC	0.003	0.118	2.108	0.967	-2565.948	-2547.42	1286.974
ETH	0.003	0.285	2.032	1.089	-2097.589	-2079.061	1052.794
EUR	0	0.005	4	1	-6150.937	-6132.409	3079.468
GOLD	0	0.007	4	1	-5591.085	-5572.557	2799.543
JPY	0	0.005	4	1	-5956.747	-5938.219	2982.374
SILVER	0	0.012	3.696	0.957	-4749.855	-4731.327	2378.927
CNY	0	0.003	4	1	-7037.789	-7019.261	3522.894
USD	0	0.002	4	1	-7279.975	-7261.447	3643.988
XMR	0.003	0.087	2.821	1.091	-1912.017	-1893.489	960.008
XRP	0.002	0.642	2.007	1.11	-2005.574	-1987.046	1006.787

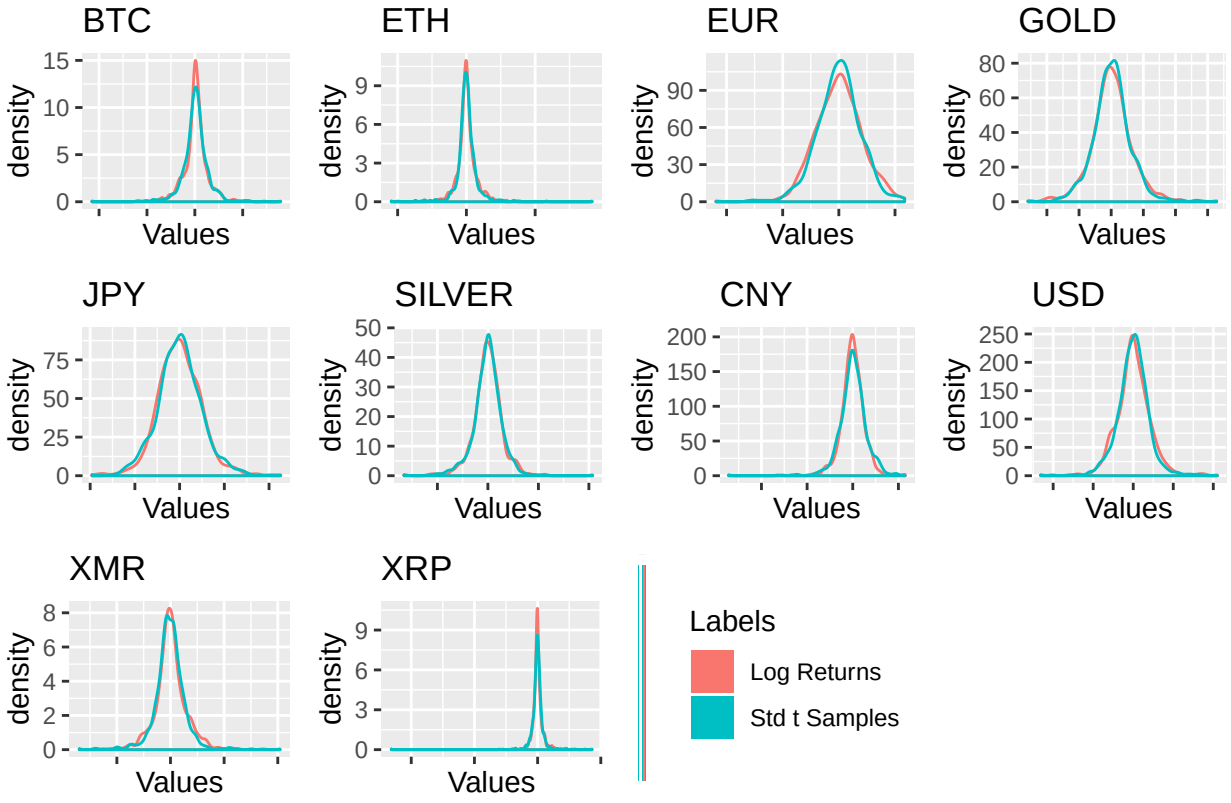
The AIC, BIC, and log likelihood values for both the standardized t distribution and skewed t distribution are quite similar. For many assets, there is no clear choice between the two models. For example, Etherium has a lower AIC under the skewed t distribution but also a lower BIC under the standardized student's t distribution. Therefore, if we were to pick a model based on a better AIC model, we would choose the skewed

t distribution, but we would choose the standardized t distribution if we were to pick a model based on BIC. The log likelihoods are also similar for both models, so there is no clear advantage or disadvantage to using either.

Thus, for all of the assets, both the standardized t distribution and skewed t distribution perform similarly well. Based on AIC, BIC, and the log likelihood, there is no clear advantage of using one model over the other. However, one important consideration is parsimony. We prefer models which are parsimonious and are able to achieve a desired level of explanatory or predictive power while using as few predictors as possible. Therefore, in this case, we would prefer to use the standardized student's t distribution which requires one less parameter.

In the figure below, we have sampled some data that follow the distributions estimated above. As we can see, the sampled data from the standardized t distributions do indeed fit the log returns quite nicely.

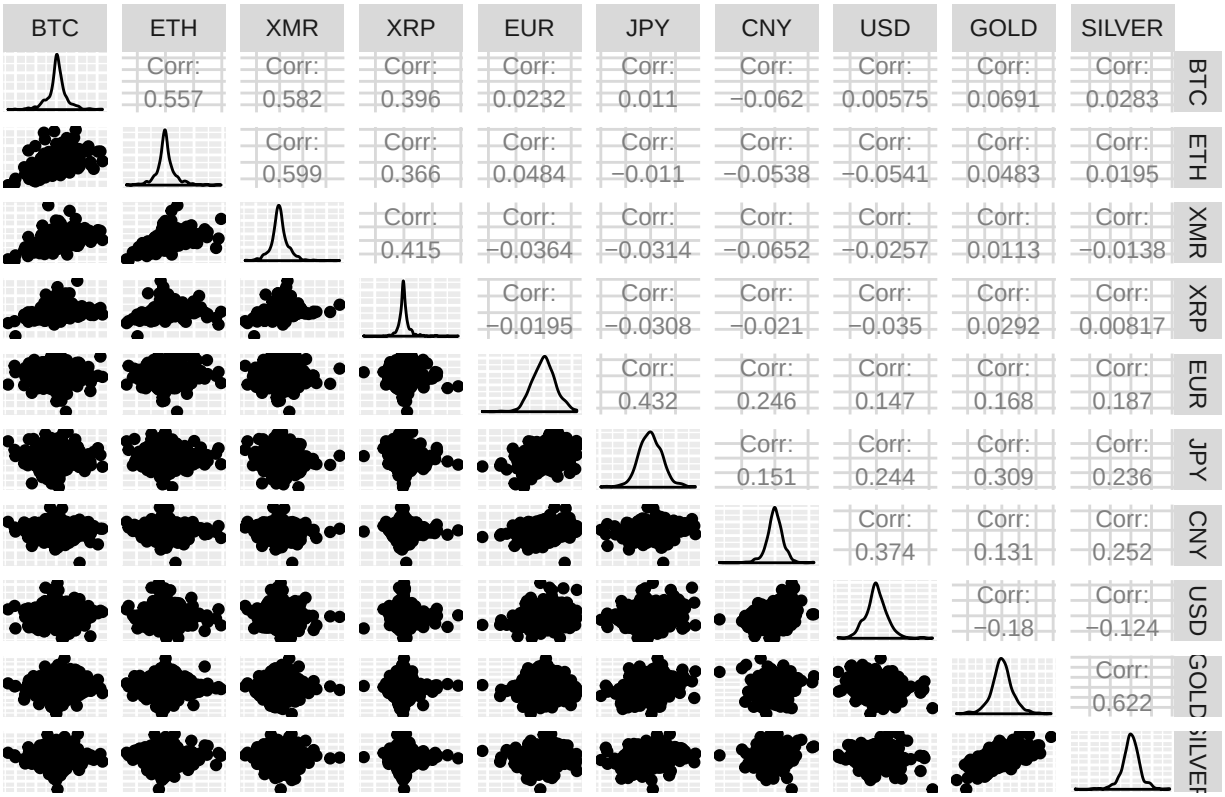
Figure 7: Estimated t distributions for Log Returns



Multivariate Analysis

Recall that in the Univariate Analysis phase, we found that the raw price data was clearly not generated from a normal or t distribution given the presence of multimodality among other noisy features. However, the t distribution represented the log transform of the gross returns (log returns) of each asset fairly well. It is natural to next turn to multivariate models to attempt to capture any relationships between the assets. To motivate this analysis, we look at the correlations between the assets.

Figure 8: Correlation between Assets



In the above plot we have plotted pairwise scatterplots of the log returns to search for any meaningful correlation values. We can see that certain pairs such as gold and silver have high correlation values of 0.622 though many are rather insignificant. The following plots present a much more important characteristic of the log returns:

Figure 9: Prices Correlation

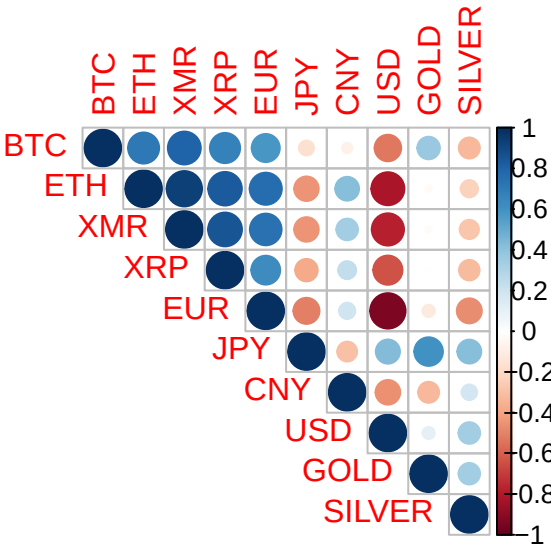
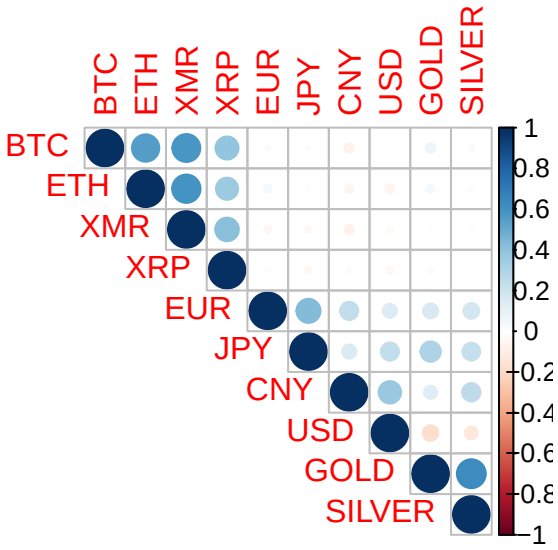


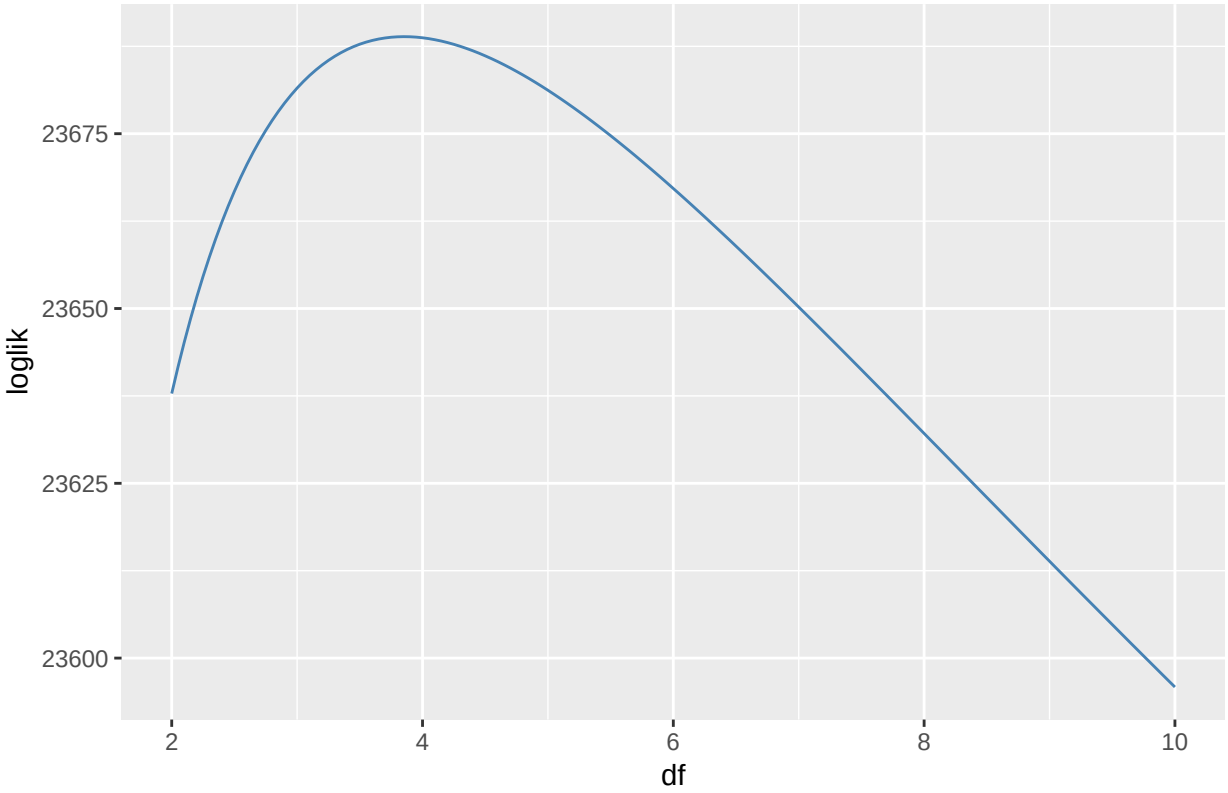
Figure 10: Log Returns Correlation



The difference between the price data and the log return data is quite striking. The price data seems to imply most of the market is moving together, other than the assets that are commonly considered safe, such as JPY. On the contrary, the log return correlations paint a different picture. There is a clear separation between the cryptocurrencies and the other assets. Had we only considered distinct univariate models, we would have had no chance of capturing this structure. These findings motivate the search for a representative multivariate model for the log return data.

First considering the entire log returns dataset, we search for the parameters of a multivariate t distribution by examining the profile likelihood of the degrees of freedom parameter and selecting the maximum. As seen in the plot below, the value of $\nu = 3.85$ is optimal.

Figure 11: Log Returns Multivariate t Profile Likelihood



To verify that this model performs better than similar models, we can compare the AIC value at $\nu = 3.85$ of -47245.76 to AIC values of a normal and skewed t distribution. The normal distribution's AIC value of -45913.77 is significantly larger than that of the t distribution indicating that it does not fit the model as well as the t distribution. Moreover, the skewed-t has an AIC value of -47059.66 which, while better than the normal, suffers from the increased parameter count and is a less parsimonious model.

Following our analysis, we have a better method of constructing a covariance matrix for the assets under investigation. As the t distribution models the log returns better than the normal distribution, generating a covariance matrix under the multivariate t distribution for each quarter end using the prior 6 months of data offers a better alternative to the sample covariance matrix for index construction.

Building the Index

Figure 12: Index Values for Ideal Weighting

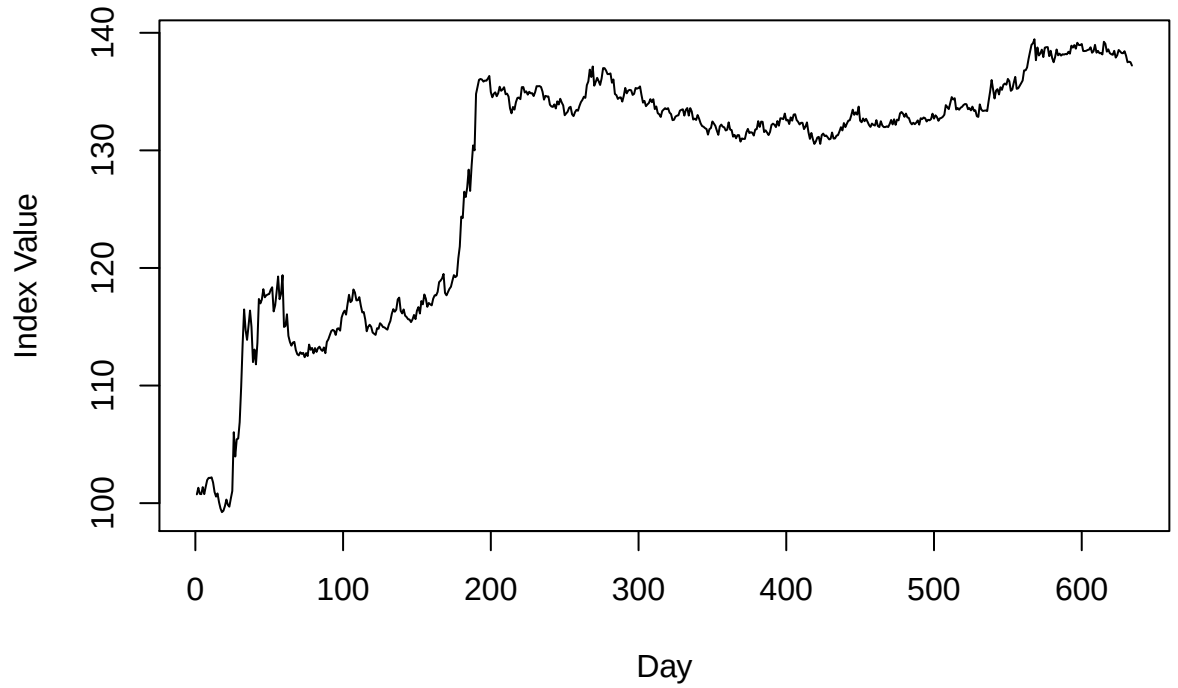
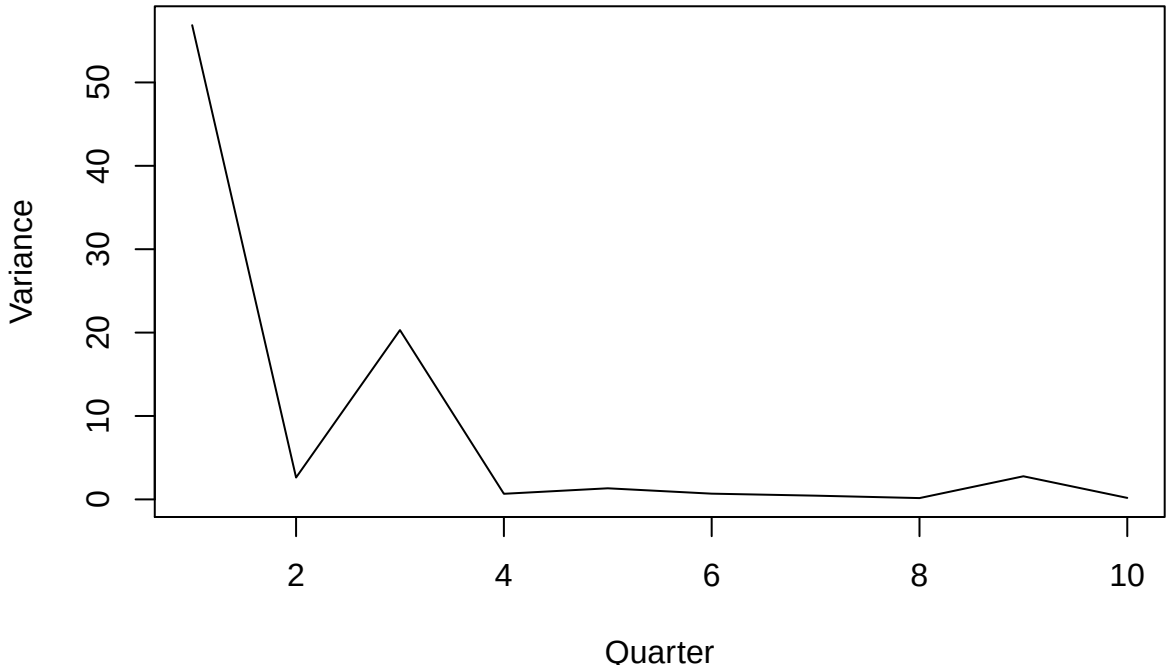


Figure 13: Variances for Ideal Weighting



Having completed the exploratory and distribution modelling analyses, we can now move on to the index construction compoenent of this analysis. Our first objective is to construct an index encompassing the currency, cryptocurrency, and precious metals assets that minimizes variance. This will then act as an

indicator of the overall performance of the market.

Framing the Optimization Problem as a Quadratic Program

Since each asset has non-zero correlation with the others, it is not enough to consider their variances separately. Instead, we must consider the covariance matrix. In particular, we would like to exploit negative correlations in order to reduce the variance. Each asset is a correlated random variable, and the index is the sum of weighted correlated random variables. Given some covariance matrix (to be described later in this section), the variance of the weighted sum is:

$$Var(S) = w^T \Sigma w$$

where the weighted sum itself is:

$$S = w^T A$$

where A is the vector of correlated items (e.g. log returns). We were also given some additional constraints, namely that each weight must be between 1% and 25%, and the weights must sum to 1. Therefore, the problem is:

$$min_w w^T \Sigma w$$

$$\text{st } \Sigma_{n=1}^{N} w_n = 1, w_n \geq 0.01, w_n \leq 0.25 \ \forall n \in 1, ..., N$$

This is a quadratic program and so we used the quadprog library in R in order to find the weights for each quarter. The rest of this section discusses various ways we tried to build the covariance matrix from which we then optimized the weights.

Covariance of Prices

The index value is defined as:

$$I(t) = \Sigma_{n=1}^{N} w_n^i \frac{I(T_i)}{A_n(T_i)} A_n(t) = \Sigma_{n=1}^{N} w_n^i k_n A_n(t)$$

where the index $I(T_i)$ an the asset value $A_n(T_i)$ are effectively extra weights (called k_n for short). The item whose covariance we are seeking is therefore:

$$\{k_n A_n(t)\}$$

And we thus require the covariance matrix:

$$C(n, m) = Cov(k_n A_n(t), k_m A_m(t))$$

This method also allows us to compute the "ideal" weights by using the true asset prices in the next quarter instead of attempting to predict the covariance in the next quarter. These ideal weights gave us the ideal index with which to compare all the other methods we tried. The ideal weights are weights that still meet the original constraints (e.g. between 1% and 25%), but cheat by using the true covariance and asset prices within each quarter. We expect that each method will yield a variance bounded below by the ideal variance.

Since we will not know the true covariance during each quarter (because it is in the future), we use the sample covariance from the past 6 months as our estimate. Other estimates of the future covariance follow.

Covariance of Log-Returns

Instead of attempting to estimate the covariance of the weighted asset prices directly, we instead try to estimate the covariance of the log returns. As before, the log-returns are calculated for each asset over the past 6 months, and the covariance of the log-returns in the future is estimated as the sample covariance of the log-returns over the past 6 months.

Covariance of Log-Returns from Fitted t-distribution

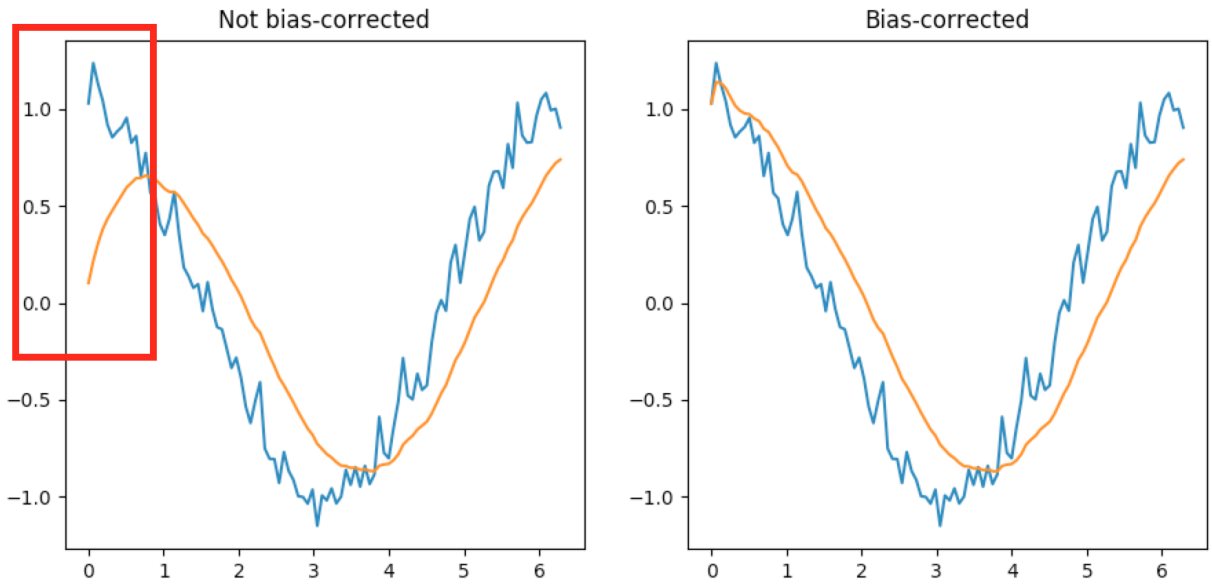
As discussed under the Multivariate Analysis section, the naive historical 6-month sample covariance matrix is a model-free Monte Carlo estimate, but the t-distribution fits the log returns well. Under this approach, we fit a multivariate t-distribution using the past 6 months of data and use the MLE of the covariance matrix as the input into the optimization problem.

Bias-Corrected Exponential Smoothing

The sample covariance of returns (or prices) from the past 6 months may not be the best estimate of the covariance in the next quarter. We therefore attempt to use an exponentially-weighted moving-average estimate (Ruppert & Matteson; Ch14: GARCH Models) in order to best capture any volatility clustering that may occur. If high volatility occurred too far in the past, EWMA will forget about it. If high volatility occurred recently, then EWMA ensures that it will have a greater effect on future predictions.

One problem with EWMA is that because the initial value of the output is zero, early estimates are biased toward zero. A visualization of the problem is shown below:

Figure 14: Bias Correction



We therefore modified Ruppert & Matteson's EWMA estimate using bias-correction. Without bias-correction, the EWMA update for the covariance would be:

$$\hat{\mu}_N = \alpha\hat{\mu}_{N-1} + (1-\alpha)x_N \ , \ \hat{V}_N = \alpha\hat{V}_{N-1} + (1-\alpha)x_N x_N^T$$

$$\hat{\Sigma}_N = \hat{V}_N - \hat{\mu}_N\hat{\mu}_N^T$$

With bias-correction, we add 2 additional updates for the first and second moments and estimate the covariance as follows:

$$\hat{\mu}'_N = \frac{1}{1-\alpha^N}\hat{\mu}_N \ , \ \hat{V}'_N = \frac{1}{1-\alpha^N}\hat{V}_N$$

$$\hat{\Sigma}_N = \hat{V}'_N - \hat{\mu}'_N(\hat{\mu}'_N)^T$$

In the below results, we tested the bias-corrected EWMA of both weighted asset prices and of log-returns for our estimate of the covariance in the next quarter as input into the quadratic program.

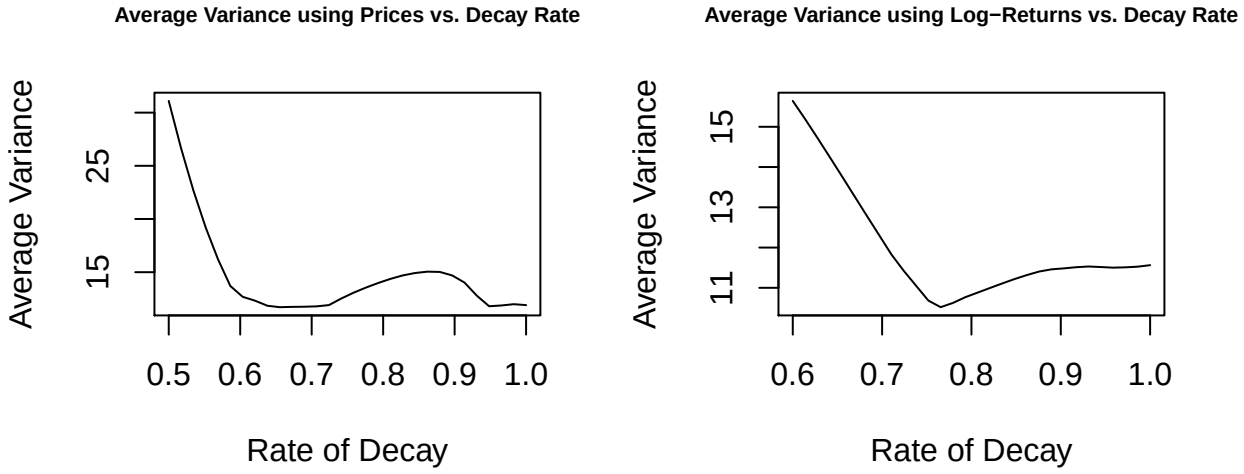
EWMA Decay Rate

The decay rate for EWMA (denoted by α above) is a hyperparameter that can be optimized. We used a grid search to find the best decay rate over all quarters by choosing the one that yielded the minimum average variance in each quarter.

Ideally, we would have used a separate validation set to pick the best decay rate, but only a limited number of quarters were available in the dataset.

For prices, the best decay rate was found to be 0.6551693. For log-returns, the best decay rate was found to be 0.7655131.

Figure 15: Optimal Rate of Decay

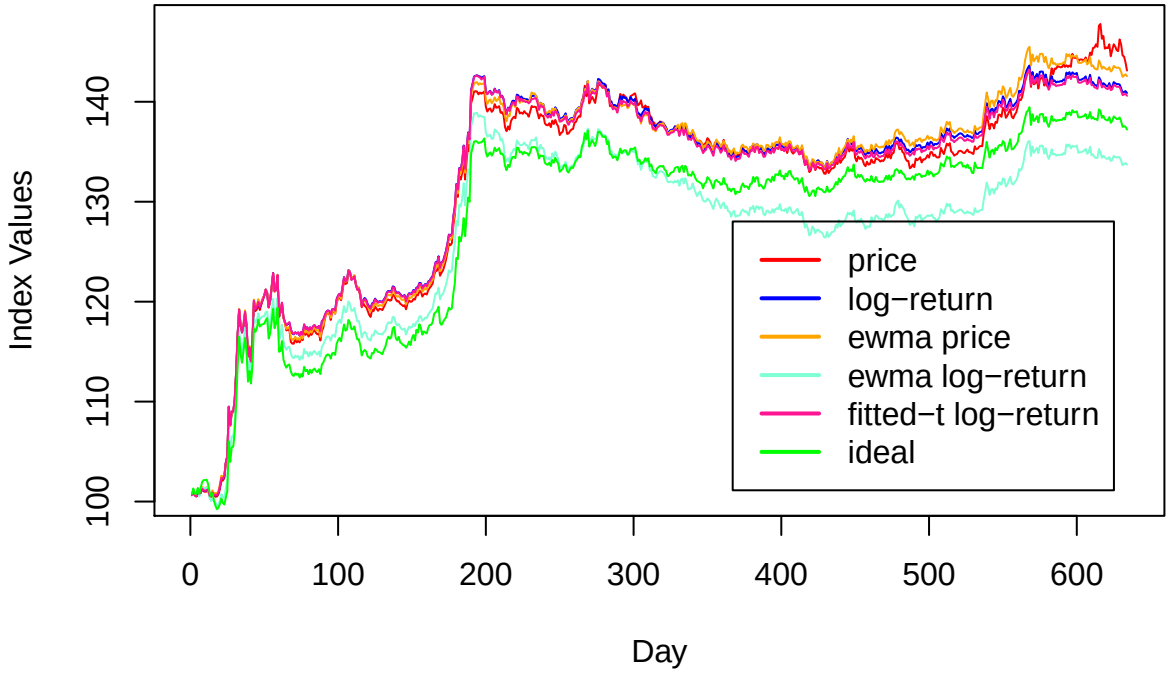


The relatively small values for the decay rate indicate that forgetting about earlier risk measurements was useful, and that more recent volatility measurements matter more. This makes sense in light of volatility clustering.

Index Performance Results

The chart below shows the index value per unit time for each method we tested compared with the ideal weightings.

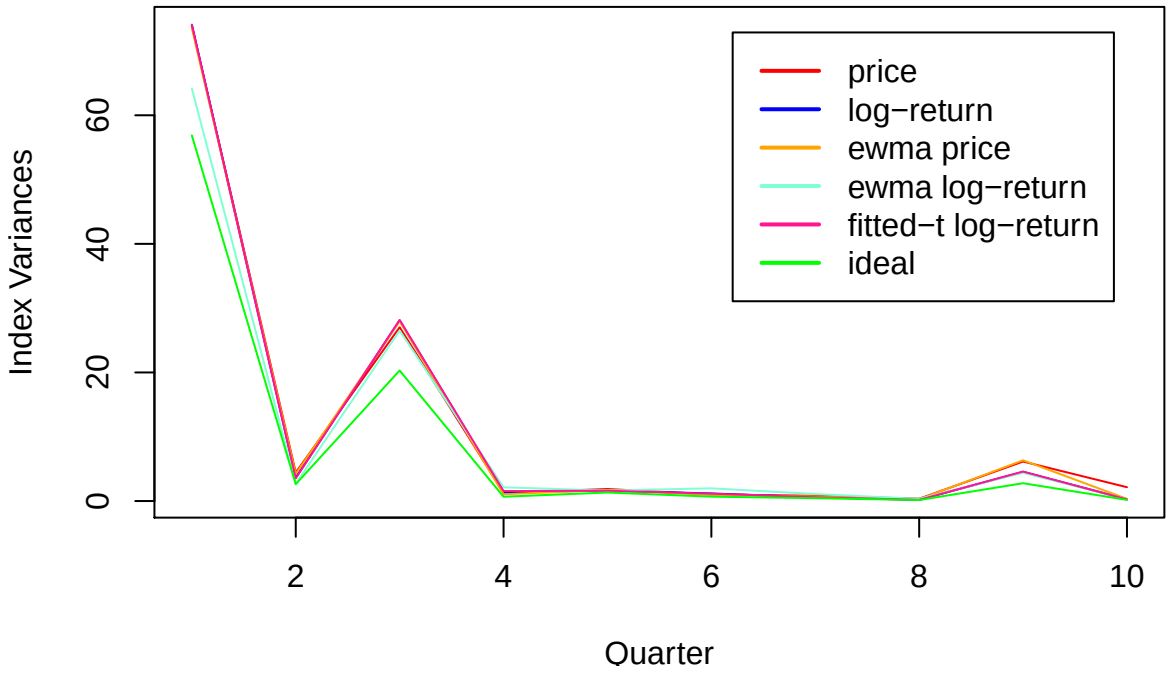
Figure 16: Index Values Comparison



We see that all indexes take on a similar shape, but the ideal weightings yield an index with generally lower values than the others.

The chart below plots the variance per quarter for each of the weighting schemes.

Figure 17: Index Variance per Quarter Comparison



One can see that the ideal weights do indeed lower bound all the other methods. Using the log-returns (both the standard estimate and EWMA) are a close tie with each other, but smaller than using prices, especially in the last few quarters.

The table below summarizes the average variance per quarter for each of the methods discussed above.

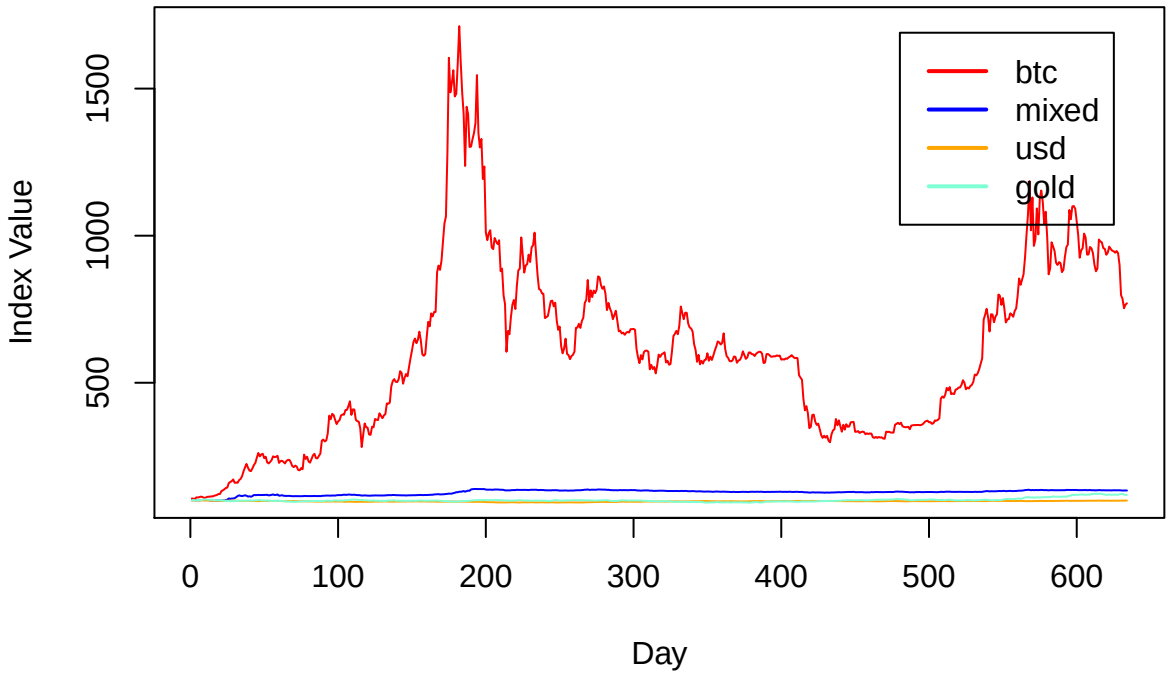
Table 5: Comparison of Methods

Method	Average Variance Per Quarter
Price	11.90243
Price EWMA	11.7104
Log-Return	11.56243
Log-Return t-dist	11.56862
Log-Return EWMA	10.51929
Ideal	8.602228

We now compare our minimum variance index (Log-Return EWMA) with single asset indices (BTC, USD, and Gold). Each of the single assets had the highest weights (or were tied for the highest weights) for the quarter after March 31, 2017.

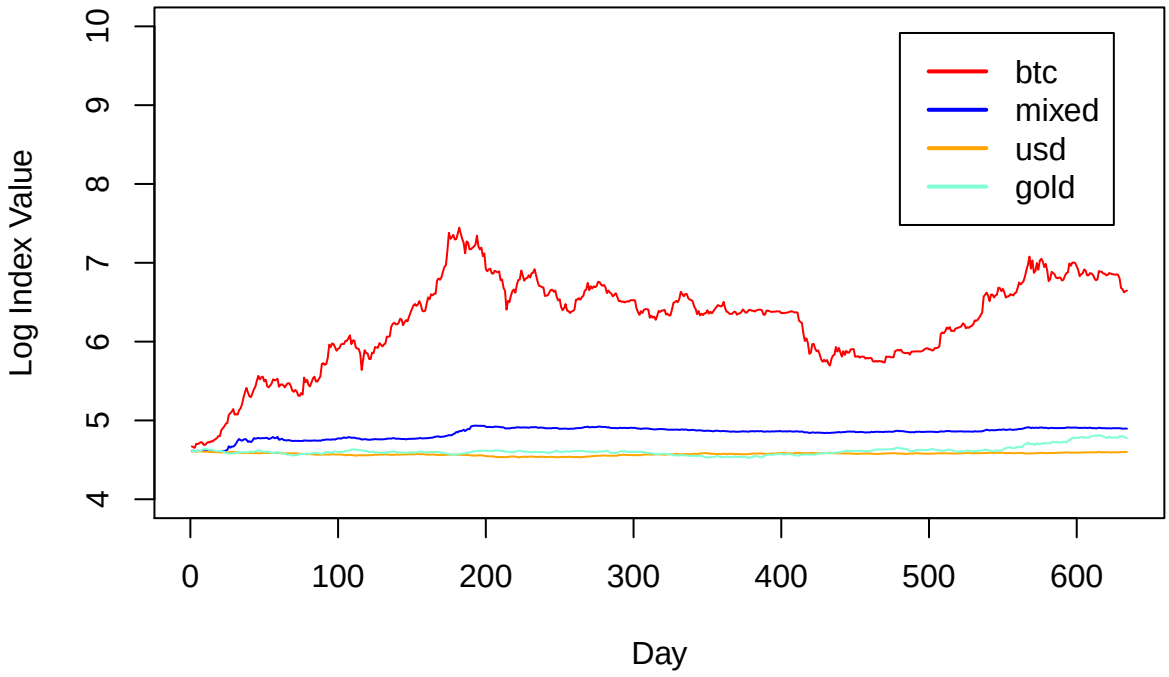
The index values are shown in the chart below:

Figure 18: Our index values vs Single asset index



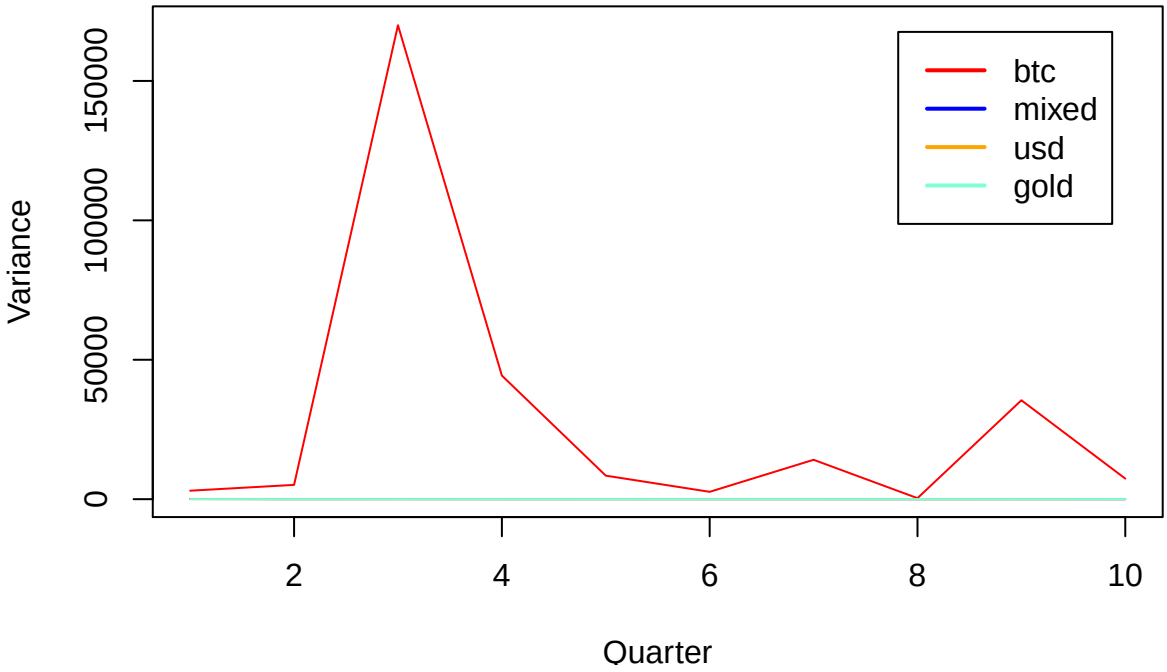
Because the index values for BTC-only are disproportionately large, it is useful to consider the log of the index values as well:

Figure 19: Log of our index values vs Single asset index



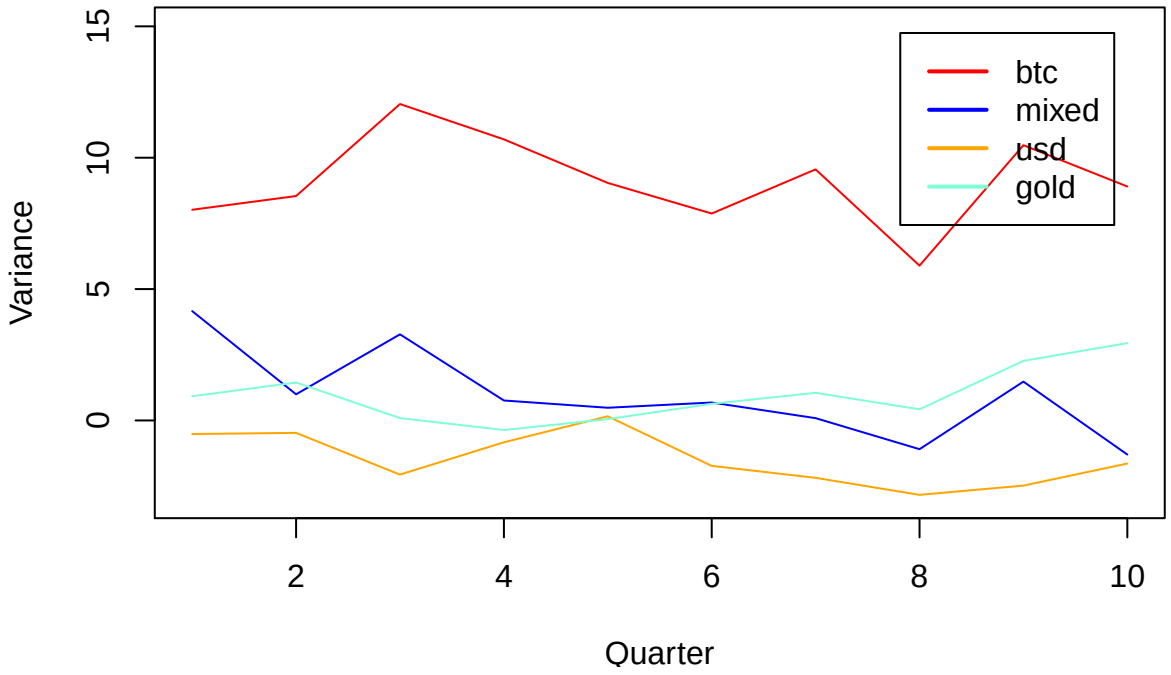
The variance per quarter of our index compared with single asset indices are shown below:

Figure 20: Our index variance vs. Single asset variance



Again, since the variance of the BTC-only index is disproportionately high, we show the log-variances also:

Figure 21: Log of our index variance vs. Single asset variance



Our index has a smaller variance than gold in some quarters, and is bounded below only by USD.

Overall, it is encouraging to see that the index benefitted from the jump in price in cryptocurrency in late 2017, while not suffering from any sharp drop-off when cryptocurrency plunged in early 2018. Instead, all the indices we created maintained their values after the cryptocurrency jump with relatively low volatility.

Time Series & Forecasting

Having constructed a minimum variance index to represent the current state of the market at any given point in time, we are now interested in seeing if we can develop models to forecast the future performance of each asset. During the EDA phase, we identified the log-returns as the most symmetric and seemingly stationary transformation of the data. If this is the case, we can begin by attempting to model the log-prices as an ARIMA(p,1,q) process. It is often floated around that market prices cannot be predicted and are simply a random walk. If this is the case, we should expect that an ARIMA(0,1,0) process we be the best performing model. The objective of this section will be to examine the validity of the random walk hypothesis by fitting optimal ARIMA models to each asset. To test the predictive power of such a model, we will remove the last quarter from the dataset and fit the model with the remainder of the data. Then, we will examine how well the final quarter can be estimated.

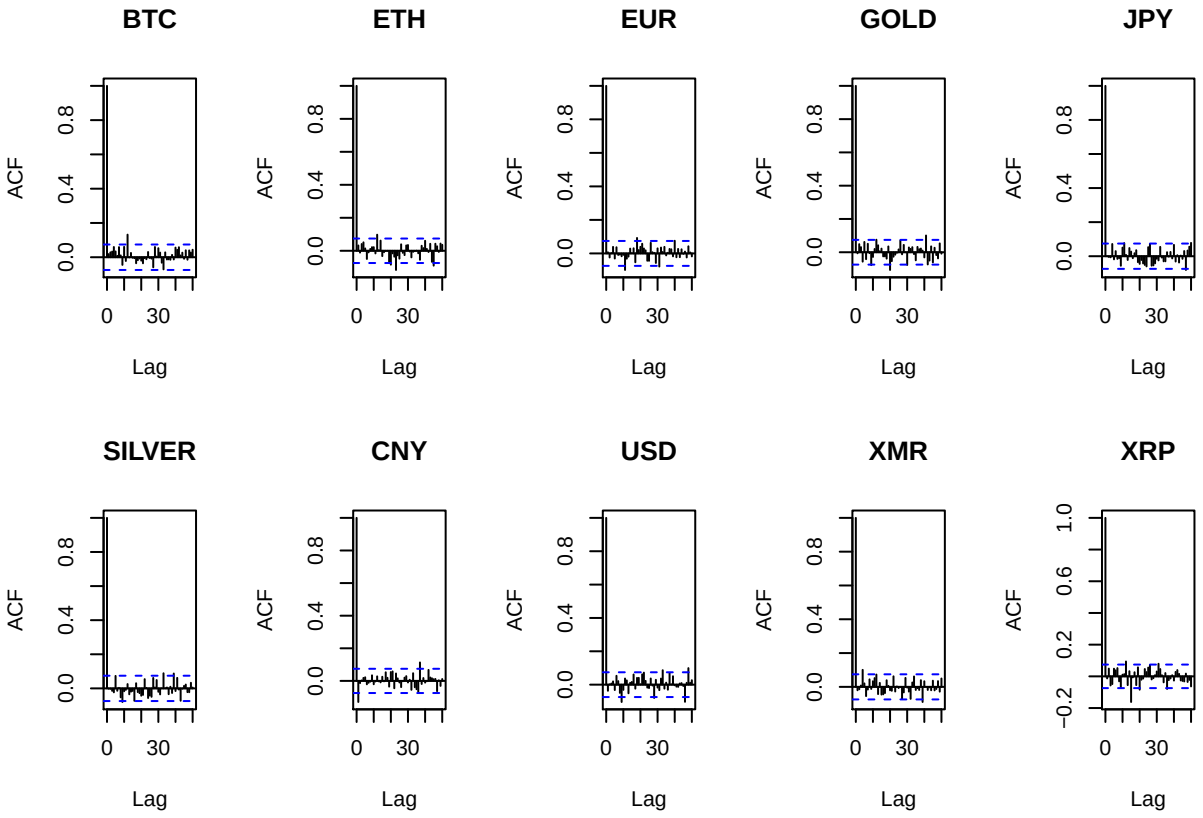
The following table displays the optimal ARIMA models chosen by minimizing the BIC. We decided to use BIC instead of AIC as the difference in AIC between models is quite small and thus to choose the most parsimonious model, we introduce the penalty for excessive parameters through the BIC.

Interestingly, most of the models produced are very low parameter models, with ARIMA(0,1,0) being the most frequent. However, assets such as silver produce higher parameter models indicating that the time series is not simply a random walk. However, before drawing any conclusions, we must examine the time series and autocorrelation plots of the residuals to determine if there is any indication that the residuals are not stationary. In the interest of space, only the ACF plots are displayed below.

Table 6: Optimal Arima Models

	p	d	q	BIC
BTC	0	1	0	-2200.438
ETH	0	2	1	-1689.836
EUR	0	1	1	-5712.552
GOLD	0	1	0	-5148.268
JPY	1	1	0	-5455.909
SILVER	2	1	1	-4317.230
CNY	0	1	0	-6424.862
USD	0	1	0	-6597.729
XMR	0	1	0	-1598.493
XRP	1	1	1	-1374.934

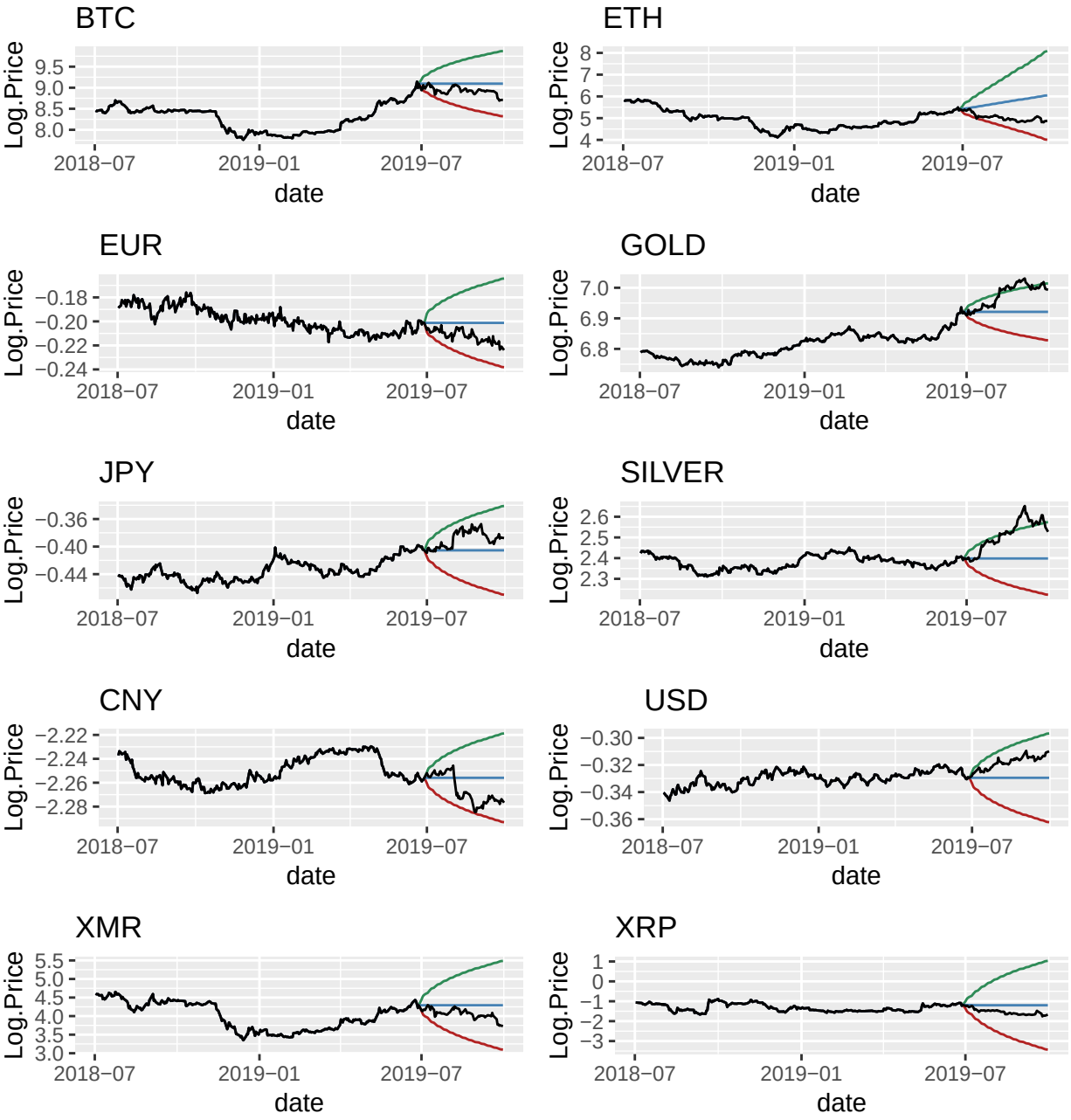
Figure 22: Residual ACF Plots



We can clearly see that there is no consistent pattern of significant autocorrelation in the residuals. This is supported by the Box-Ljung tests performed. For each asset, we tested the lag parameter equal to 5, 10, and 25. Each instance did not reject the null hypothesis for lag values of 5 and 10 though the ethereum did detect the presence of minor autocorrelation in the lag 25 instance and the xrp data did find autocorrelation in the lag 25 case. For xrp, we can see the negative spike in the ACF plot at the lag 15 mark. As this is not a consistent pattern of autocorrelation and we know that the cryptocurrency is very noisy and definitely not stationary, we do not take the minor presence of autocorrelation as invalidating the models. As a whole, the ARIMA models do appear to fit well.

Finally, to test the predictive power of these models, we have forecasted the final quarter of the dataset along with 95% confidence intervals. The forecast and confidence intervals are then imposed on the time series plots to gauge how well the forecasts performed.

Figure 23: Log Price Forecasts



In summary, the actual forecasts (blue line) are not necessarily accurate. Note that the random walk models produce a constant forecast given that the expected value is 0. However, in general, the 95% confidence intervals do seem to contain the actual log-prices aside from the extremely large positive fluctuations in the precious metals data. While the gold model was a random walk and would not have benefited much from seeing the additional data, the silver model was more complex and had not been fit with any strong positive differences data. Thus, it is possible that the data used to fit the silver model was not representative of the complete behaviour of the asset. For the more stable assets, the forecasts do perform well and the actual log-prices are often very close to the predicted log-prices. Given the results of the forecast, even models that did not employ the random walk model performed well, indicating that there may be more than random behaviour driving market prices.

Conclusion

Through the analysis of the market data and construction of a stable index, a few important conclusions have been identified. First of all, we have reinforced the notion that while raw market price data is far too noisy to analyse from a time series perspective, the log-transform and differencing operators together are able to consistently induce symmetry and stationarity. We also reinforced the conclusion that the t-distribution's heavy-tails model market log-returns quite well. Extending this conclusion to the multivariate domain, the multivariate t-distribution not only captures the heavy-tailed aspects, but also considers the dependency structure observed in the data. This observation was then employed in the index construction component. We attempted a variety of methods of constructing a covariance matrix to insert into the minimum variance objective and found that log-return EWMA was optimal (excluding the true future covariance matrix). Finally, we posed the question of whether market data was simply a random walk or if there was some underlying structure in each asset that could be exploited for forecasting purposes. While many of the assets were best modelled by a random walk, we did find that certain assets such as Silver and Ripple did have statistically significant parameters. We found that the actual market behaviour in the final quarter of the dataset consistently fell within the 95% confidence interval of the ARIMA forecasts. This suggests that there is reason to believe that these models do have the capability to model the financial markets.

R Code Used

```
library(MASS)
library(dplyr)
library(fGarch)
library(tidyverse)
library(ggplot2)
library(GGally)
library(xts)
library(forecast)
library(grid)
library(gridExtra)
library(corrplot)
library(RColorBrewer)
library(dplyr)
library(mnormt)
library(sn)
library(knitr)
library(kableExtra)
```

```
### DATAFRAMES
# raw data - close prices
prices<-read.csv("datav2.csv") ## DATAFRAME TO USE
prices <- prices %>% mutate(Date = as.Date(as.character(prices$Date),
                                           format = "%Y-%m-%d")) #Date type
# setup
allcolnames<-colnames(prices)
Date<- prices$Date[-1]
Pt<- prices[-1,-1]
Ptminus1<-prices[-(length(Date)+1),-1]
# differences in prices
diffPrices <- cbind(Date, Pt-Ptminus1)
diffPrices$Label = "diffPrice"
# log prices
logPrices<-cbind(Date = prices$Date,log(prices[,2:11]))
logPrices$Label = "logPrice"
# net returns
netReturns<- cbind(Date, (Pt-Ptminus1)/Ptminus1) # (Pt-Pt-1)/(Pt-1)
# gross returns
grossReturns<-cbind(Date, Pt/Ptminus1)
grossReturns$Label = "grossReturn"
# log returns
logReturns<-cbind(Date, log(Pt/Ptminus1))
logReturns$Label = "logReturn"

prices$Label = "price"
netReturns$Label ="netReturn"
grossReturns$Label = "grossReturn"
logReturns$Label = "logReturn"

### FUNCTION TO FLATTEN THE DATAFRAMES
flatten <- function(dataframe){
  flattened_df <- rbind(setNames(dataframe[,c(1,2,12)],c("Date","Value","Label")),
```

```
                    setNames(dataframe[,c(1,3,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,4,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,5,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,6,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,7,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,8,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,9,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,10,12)],c("Date","Value","Label")),
                    setNames(dataframe[,c(1,11,12)],c("Date","Value","Label"))
  )
  flattened_df$asset<- rep(allcolnames[2:11], each = nrow(flattened_df)/10)
  return (flattened_df)
}
### FLATTENED DATAFRAMES
flatprices<- flatten(prices)
flatdiffPrices<-flatten(diffPrices)
flatlogPrices<-flatten(logPrices)
flatnetReturns<-flatten(netReturns)
flatgrossReturns<-flatten(grossReturns)
flatlogReturns<-flatten(logReturns)
```

```
ggplot(flatprices, mapping=aes(x=Value))+geom_density() +
  labs(title = "Figure 1: Kernel Density Plots of Raw Price Data") +
  facet_wrap(~asset, scales='free')
```

```
# Time Series Plots
ggplot(flatprices, mapping=aes(x=Date, y = Value))+geom_line() +
  facet_wrap(~asset, scales="free") +
  labs(title = "Figure 2: Time Series Plot of Raw Prices")
```

```
# Unit Root Tests for all of the data
library(kableExtra)
library(tseries)
# KPSS
unitroot_table<-matrix(rep(NA,60),nrow=6) # one row per data set
# prices
for (i in 2:11){
  kps<- kpss.test(prices[,(i-1)])
  unitroot_table[1, (i-1)] = round(kps$p.value,3)
}
for (i in 2:11){
  kps<- kpss.test(diffPrices[,(i-1)])
  unitroot_table[2, (i-1)] = round(kps$p.value,3)
}
for (i in 2:11){
  kps<- kpss.test(logPrices[,(i-1)])
  unitroot_table[3, (i-1)] = round(kps$p.value,3)
}
for (i in 2:11){
  kps<- kpss.test(netReturns[,(i-1)])
  unitroot_table[4, (i-1)] = round(kps$p.value,3)
}
for (i in 2:11){
  kps<- kpss.test(grossReturns[,(i-1)])
```

```
  unitroot_table[5, (i-1)] = round(kps$p.value,3)
}
for (i in 2:11){
  kps<- kpss.test(logReturns[,(i-1)])
  unitroot_table[6, (i-1)] = round(kps$p.value,3)
}
unitroot_table<-data.frame(unitroot_table)
rownames(unitroot_table)<-c("Prices","Diff in Prices", "Log Prices",
                            "Net Returns","Gross Returns","Log Returns")
colnames(unitroot_table)<-allcolnames[2:11]

kable(unitroot_table,
      caption ="P-Values from the KPSS Test for different data transformations")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
# Time Series Plots
ggplot(flatlogReturns, mapping=aes(x=Date, y = Value))+geom_line() +
  facet_wrap(~asset, scales="free") +
  labs(title = "Figure 3: Time Series Plot of Log Returns")
```

```
ggplot(flatlogReturns, mapping=aes(x=asset, y = Value))+geom_boxplot()  +
  labs(title = "Figure 4: Boxplots of Log Return Data")
```

```
ggplot(flatlogReturns, mapping=aes(x=Value))+geom_density() +
  labs(title = "Figure 5: Kernel Density Plots of Log Return Data") +
  facet_wrap(~asset, scales='free')  +
  theme(axis.text.x = element_text(size=6))
        #axis.text.x = element_text(angle=90, hjust=1))
```

```
ggplot(flatlogReturns, aes(sample=Value)) +
  stat_qq() +
  stat_qq_line() + facet_wrap(~asset, scales="free") +
  labs(title = "Figure 6: Quantile-Quantile Plots of Log Return Data")
```

```
total_pvals2 <- matrix(rep(NA,20),ncol=2)
for (i in 2:11){
  total_pvals2[(i-1),1]<- allcolnames[i]
  total_pvals2[(i-1),2]<-shapiro.test(logReturns[,i])$p.value
}
total_pvals2<-data.frame(total_pvals2)
colnames(total_pvals2)<-c("Asset","P-Value")

kable(total_pvals2,
      caption ="P-Values from the Shapiro Wilks Test for Log Return Data")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed",
                                      "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
library(fGarch)
std_calc<-function(df){
  total_std_table<- matrix(rep(NA,70),nrow=10)
  total_sstd_table<- matrix(rep(NA,80),nrow=10)
  for (i in 2:11){
    # STD
```

```
    s1<- stdFit(df[,i],hessian=FALSE)
    AIC_std = 2 * s1$objective + 2 * length(s1$par) # add 2*#params
    BIC_std = 2 * s1$objective + log(length(df[,i])) * length(s1$par)
    total_std_table[(i-1),1]<-allcolnames[i]
    total_std_table[(i-1),2]<- round(s1$par[1],3)
    total_std_table[(i-1),3]<- round(s1$par[2],3)
    total_std_table[(i-1),4]<- round(s1$par[3],3)
    total_std_table[(i-1),5]<- round(AIC_std,3)
    total_std_table[(i-1),6]<- round(BIC_std,3)
    total_std_table[(i-1),7]<- round(-s1$objective,3) # log likelihood

    #SSTD
    ss1<- sstdFit(df[,i],hessian=FALSE)
    AIC_sstd = 2 * ss1$minimum + 2 * length(ss1$estimate) # add 2*#params
    BIC_sstd = 2 * ss1$minimum + log(length(df[,i])) * length(ss1$estimate)
    total_sstd_table[(i-1),1]<-allcolnames[i]
    total_sstd_table[(i-1),2]<- round(ss1$estimate[1],3)
    total_sstd_table[(i-1),3]<- round(ss1$estimate[2],3)
    total_sstd_table[(i-1),4]<- round(ss1$estimate[3],3)
    total_sstd_table[(i-1),5]<- round(ss1$estimate[4],3)
    total_sstd_table[(i-1),6]<- round(AIC_sstd,3)
    total_sstd_table[(i-1),7]<- round(BIC_sstd,3)
    total_sstd_table[(i-1),8]<- round(-ss1$minimum,3)
  }
  total_std_table<- data.frame(total_std_table)
  total_sstd_table<- data.frame(total_sstd_table)
  colnames(total_std_table)<-c("Asset", "mean", "sd","df", "AIC","BIC",
                               "Log Likelihood")
  colnames(total_sstd_table)<-c("Asset","mean", "sd","df",
                                "epsilon", "AIC","BIC","Log Likelihood")
  return (list(stdtab = total_std_table, sstdtab = total_sstd_table))
}
```

```
kable(std_calc(logReturns)$stdtab, caption ="Parameter Estimates, AIC, BIC,
      and Log Likelihood for Log Returns fitted to Standardized t Distribution")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed",
                                      "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
kable(std_calc(logReturns)$sstdtab, caption ="Parameter Estimates, AIC, BIC,
      and Log Likelihood for Log Returns fitted to Skewed t Distribution")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed",
                                      "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
df = logReturns

plot_list = list()
outputused <- std_calc(df)$stdtab # the std table #df should be logReturns
outputused$mean<-as.numeric(as.character(outputused$mean))
outputused$sd<-as.numeric(as.character(outputused$sd))
outputused$df<-as.numeric(as.character(outputused$df))
outputused$AIC<-as.numeric(as.character(outputused$AIC))
outputused$BIC<-as.numeric(as.character(outputused$BIC))
```

```
outputused$`Log Likelihood`<-as.numeric(as.character(outputused$`Log Likelihood`))
outputused$mean<-as.numeric(as.character(outputused$mean))

for (i in 2:11){
  data = df[,i]
  #print(summary(f1n))
  samples<- rstd(n=length(data), mean = outputused$mean[(i-1)],
                 sd = outputused$sd[i-1], nu=outputused$df[i-1])
  Labels<-rep(c("Log Returns", "Std t Samples"), each = length(data))
  Values<-c(data,samples)
  table<-cbind(Values=Values,Labels=Labels)
  table<-data.frame(table)
  #table$Values<-as.numeric(table$Values)
  table$Values <- as.numeric(as.character(table$Values))
  temp <- ggplot(table, aes(x=Values, color=Labels)) +
    geom_density(show.legend=FALSE) +ggtitle(allcolnames[i]) +
    theme(#axis.title.x=element_blank(),
        axis.text.x=element_blank()
        )
  plot_list[[i-1]] <-temp
}

#empty dataframe
dat <- data.frame(x=runif(10),y=runif(10))

p <- ggplot(dat, aes(x=x, y=y)) +
        geom_point() +
        scale_x_continuous(expand=c(0,0)) +
        scale_y_continuous(expand=c(0,0))

p<- p + theme(axis.line=element_blank(),axis.text.x=element_blank(),
          axis.text.y=element_blank(),axis.ticks=element_blank(),
          axis.title.x=element_blank(),
          axis.title.y=element_blank())

my_hist <- ggplot(table, aes(Values, fill = Labels)) +
    geom_bar() +theme(axis.line=element_blank(),axis.text.x=element_blank(),
          axis.text.y=element_blank(),axis.ticks=element_blank(),
          axis.title.x=element_blank(),
          axis.title.y=element_blank())
plot_list[[11]] <- my_hist
grid.arrange(grobs=plot_list, ncol = 4,
             top = "Figure 7: Estimated t distributions for Log Returns")
```

```
ggpairs(logReturns, columns=c(2,3,10,11,4,6,8,9,5,7), axisLabels="none", progress=FALSE,
        upper = list(continuous = wrap("cor", size=3), combo = "box_no_facet"),
        title = "Figure 8: Correlation between Assets")
```

```
pricescorr <- cor(prices[,c(2,3,10,11,4,6,8,9,5,7)])
logreturncorr <- cor(logReturns[,c(2,3,10,11,4,6,8,9,5,7)])
par(mfrow=c(1,2))
corrplot(pricescorr, type = "upper", title="Figure 9: Prices Correlation",
         mar=c(0,0.25,1,0.25))
corrplot(logreturncorr, type = "upper", title="Figure 10: Log Returns Correlation",
```

```
        mar=c(0,0.25,1,0.25))
```

```
#Search for the optimal multivariate t parameters
dfs = seq(2,10, 0.05)
n = length(dfs)
loglik = rep(0, n)
for (i in 1:n){
  multi.t = cov.trob(logReturns[,(2:11)], nu=dfs[i])
  loglik[i] = sum(dmt(logReturns[,(2:11)], mean=multi.t$center, S=multi.t$cov,
                      df = dfs[i], log=TRUE))
}
AICt = -2*loglik + 2*(1 + 10 + 10*(10+1)/2)
multi.t.df <- data.frame(df = dfs, Log.Likelihood = loglik, AIC = AICt)
## ^^^I ASSUMED THIS IS AICt. it said AIC = AIC before
ggplot(multi.t.df, aes(df, loglik)) + geom_line(col="steelblue") +
  ggtitle("Figure 11: Log Returns Multivariate t Profile Likelihood")
nuhat <- dfs[loglik == max(loglik)]
multi.t.hat <- cov.trob(logReturns[,(2:11)], nu=nuhat)
mvnorm.loglik <- sum(dmnorm(logReturns[,(2:11)],
                            mean=apply(logReturns[,(2:11)], 2, mean),
                            varcov=cov(logReturns[,(2:11)]), log=TRUE))
AICn <- -2*mvnorm.loglik + 2*(10 + 10*(10+1)/2)
skewt <- mst.mple(y=logReturns[,(2:11)], penalty=NULL)
AICst <- -2 * skewt$logL + 2*(1 + 10 + 10*(10+1)/2)
```

```
#### find_weights.R
library(quadprog)
library(rlist)

##### data source
df = read.csv('datav2.csv')
##### the columns we care about
names = c('BTC', 'ETH', 'XMR', 'XRP', 'EUR', 'JPY', 'USD', 'CNY', 'GOLD', 'SILVER')
data = df[names]
N = length(names) # number of assets
##### setup variables for QP solver
constr_mat = matrix(rep(0, (2*N + 1)*N), 2*N + 1, N)
constr_vec = c(1, rep(0.01, N), rep(-0.25, N))
for (i in 1:N) {
  constr_mat[1,i] = 1 # equality constraint
  constr_mat[1+i,i] = 1
  constr_mat[1+i+N,i] = -1
}
constr_mat = t(constr_mat) # transpose
##### quarter start and end dates

# these are for datav2.csv
q_starts = c(
  '2016-10-03',
  '2017-01-03',
  '2017-04-03',
  '2017-07-03',
  '2017-10-02',
  '2018-01-02',
```

```
  '2018-04-02',
  '2018-07-02',
  '2018-10-01',
  '2019-01-02',
  '2019-04-01',
  '2019-07-01'
)
q_ends = c(
  '2016-12-30',
  '2017-03-31',
  '2017-06-30',
  '2017-09-29',
  '2017-12-29',
  '2018-03-30',
  '2018-06-29',
  '2018-09-28',
  '2018-12-31',
  '2019-03-29',
  '2019-06-28',
  '2019-09-30'
)
##### a function to find all weights in all quarters
find_weights_all <- function(get_weights_func) {
  weights = matrix(rep(0, 100), 10, 10)
  index_values = list()
  all_index_values = c()
  index_variances = rep(0, 10)
  I_end = 100
  for (q in 1:10) {
    result = optimize_quarter(q, I_end, get_weights_func)
    weights[q,] = result$weights
    index_values = list.append(index_values, result$index_values)
    index_variances[q] = result$variance

    # concatenate index values to one big vector
    all_index_values = c(all_index_values, result$index_values)

    # update I_end
    I_end = result$index_values[length(result$index_values)]
  }
  res = list(
    weights=weights,
    index_values=index_values,
    all_index_values=all_index_values,
    index_variances=index_variances)
  return(res)
}
##### get dates for quarter
# returns 4 dates:
#   data_start (start of data gathering period)
#   data_end (end of data gathering period)
#   q_start (start of quarter)
#   q_end (end of quarter)
```

```r
get_indices_for_quarter <- function(q) {
  data_start_date = q_starts[q]
  data_end_date = q_ends[q+1]
  quarter_start_date = q_starts[q+2]
  quarter_end_date = q_ends[q+2]

  d1 = which(df$Date == data_start_date)
  d2 = which(df$Date == data_end_date)
  q1 = which(df$Date == quarter_start_date)
  q2 = which(df$Date == quarter_end_date)
  res = list(data_start=d1, data_end=d2, q_start=q1, q_end=q2)
}
##### a function to find the weights for a given quarter
# Inputs:
#    q (integer representing which quarter)
#    I_end (index value at the end of last quarter)
#    get_weights_func (function to get weights based on a list of prices)
# Outputs:
#    result$
#        - weights
#        - index_values
#        - variance
optimize_quarter <- function(q, I_end, get_weights_func) {
  # Find relevant dates
  # start of 9 mos ago, end of 3 mos ago
  # start of quarter, end of quarter
  date_indices = as.numeric(get_indices_for_quarter(q))
  d1 = date_indices[1]
  d2 = date_indices[2]
  q1 = date_indices[3]
  q2 = date_indices[4]
  # data to use to calculate covariance
  last_two_quarters = df[d1:d2, names]

  # asset values at end of last quarter
  A_end = as.numeric(df[d2, names])

  # get the weights
  w = get_weights_func(last_two_quarters, I_end, A_end)

  # get the data for the given quarter
  q_data = df[q1:q2, names]

  # calculate the index values
  I = 0
  for (i in 1:N) {
    I = I + w[i] * q_data[i] * I_end / A_end[i]
  }
  # make it a vector
  I = as.numeric(unlist(I))

  # calculate the variance
  v = var(I)
```

```
  result = list(weights=w, index_values=I, variance=v)
}

##### get weights using the sample covariance of prices from the last 6 months
get_weights_simple <- function(data, I_end, A_end) {
  # col-wise division
  data_weighted = data
  for (i in 1:N) {
    data_weighted[i] = data[i] * I_end / A_end[i]
  }
  the_cov = cov(data_weighted)
  result = solve.QP(Dmat=2*the_cov, dvec=rep(0, N), Amat=constr_mat, bvec=constr_vec, meq=1)
  # print(result)
  w = result$solution
  return(w)
}
##### get weights using log returns
# note: additional arguments not used
get_weights_log_returns <- function(data, ...) {
  # length of data
  T = dim(data)[1]
  log_returns = matrix(rep(0, (T - 1)*N), T - 1, N)
  for (i in 1:N) {
    log_returns[,i] = diff(as.numeric(unlist(log(data[i]))))
  }
  the_cov = cov(log_returns)
  result = solve.QP(Dmat=2*the_cov, dvec=rep(0, N), Amat=constr_mat, bvec=constr_vec, meq=1)
  w = result$solution
  return(w)
}
##### get EWMA covariance
lambda = 0.99
get_ewma_cov <- function(data) {
  mu = 0
  xxT = 0

  # bias-corrected
  mu_hat = 0
  xxT_hat = 0

  T = dim(data)[1]
  for (t in 1:T) {
    mu = lambda * mu + (1 - lambda) * data[t,]
    mu_hat = mu / (1 - lambda ^ t)

    xxT = lambda * xxT + (1 - lambda) * outer(data[t,], data[t,])
    xxT_hat = xxT / (1 - lambda ^ t)
  }
  c_hat = xxT_hat - outer(mu_hat, mu_hat)
  return(c_hat)
}
##### get weights using EWMA for prices
get_weights_price_ewma <- function(data, I_end, A_end) {
```

```
  T = dim(data)[1]
  data_weighted = matrix(rep(0, T*N), T, N)
  for (i in 1:N) {
    data_weighted[,i] = as.numeric(unlist(data[i] * I_end / A_end[i]))
  }
  the_cov = get_ewma_cov(data_weighted)
  result = solve.QP(Dmat=2*the_cov, dvec=rep(0, N), Amat=constr_mat, bvec=constr_vec, meq=1)
  w = result$solution
  return(w)
}
##### get weights using EWMA for log returns
get_weights_log_ret_ewma <- function(data, ...) {
  T = dim(data)[1]
  log_returns = matrix(rep(0, (T - 1)*N), T - 1, N)
  for (i in 1:N) {
    log_returns[,i] = diff(as.numeric(unlist(log(data[i]))))
  }
  the_cov = get_ewma_cov(log_returns)
  result = solve.QP(Dmat=2*the_cov, dvec=rep(0, N), Amat=constr_mat, bvec=constr_vec, meq=1)
  w = result$solution
  return(w)
}

##### get fitted-t covariance
# currently doesn't work because covariance not symmetric
tcov <- function(data, ...){
  dfs = seq(2,10, 0.05)
  n = length(dfs)
  loglik = rep(0, n)
  model <- NA
  opt <- -Inf
  for (i in 1:n){
    multi.t = cov.trob(data, nu=dfs[i])
    loglik[i] = sum(dmt(data, mean=multi.t$center, S=multi.t$cov, df = dfs[i], log=TRUE))
    # print(loglik[i])
    if (loglik[i] > opt){
      model <- multi.t
      opt <- loglik[i]
    }
  }
  return(matrix(as.numeric(model$cov), 10, 10))
}
get_weights_log_returns_t <- function(data, ...) {
  # length of data
  T = dim(data)[1]
  log_returns = matrix(rep(0, (T - 1)*N), T - 1, N)
  for (i in 1:N) {
    log_returns[,i] = diff(as.numeric(unlist(log(data[i]))))
  }
  the_cov = tcov(log_returns)
  result = solve.QP(Dmat=2*the_cov, dvec=rep(0, N), Amat=constr_mat, bvec=constr_vec, meq=1)
  w = result$solution
  return(w)
```

```
}
```

```
##### test the functions
result = find_weights_all(get_weights_simple)
result2 = find_weights_all(get_weights_log_returns)
###### (code used to find the optimal lambda is below)
lambda = 0.6551693
result3 = find_weights_all(get_weights_price_ewma)
lambda = 0.7655131
result4 = find_weights_all(get_weights_log_ret_ewma)
result_t = find_weights_all(get_weights_log_returns_t)
###### find the ideal weights
find_weights_ideal <- function() {
  weights = matrix(rep(0, 100), 10, 10)
  index_values = list()
  all_index_values = c()
  index_variances = rep(0, 10)
  I_end = 100
  for (q in 1:10) {
    result = optimize_quarter_ideal(q, I_end)
    weights[q,] = result$weights
    index_values = list.append(index_values, result$index_values)
    index_variances[q] = result$variance

    # concatenate index values to one big vector
    all_index_values = c(all_index_values, result$index_values)

    # update I_end
    I_end = result$index_values[length(result$index_values)]
  }
  res = list(
    weights=weights,
    index_values=index_values,
    all_index_values=all_index_values,
    index_variances=index_variances)
  return(res)
}

optimize_quarter_ideal <- function(q, I_end) {
  # Find relevant dates
  # start of 9 mos ago, end of 3 mos ago
  # start of quarter, end of quarter
  date_indices = as.numeric(get_indices_for_quarter(q))
  # d1 = date_indices[1]  # don't need this one
  d2 = date_indices[2]
  q1 = date_indices[3]
  q2 = date_indices[4]

  # get the data for the given quarter
  q_data = df[q1:q2, names]

  # asset values at end of last quarter
  A_end = as.numeric(df[d2, names])
```

```
  # get the weights
  w = get_weights_simple(q_data, I_end, A_end)

  # calculate the index values
  I = 0
  for (i in 1:N) {
    I = I + w[i] * q_data[i] * I_end / A_end[i]
  }
  # make it a vector
  I = as.numeric(unlist(I))

  # calculate the variance
  v = var(I)

  result = list(weights=w, index_values=I, variance=v)
}

result_ideal = find_weights_ideal()
```

```
plot(result_ideal$all_index_values, type='l',
     main="Figure 12: Index Values for Ideal Weighting", xlab="Day", ylab="Index Value")
```

```
plot(result_ideal$index_variances, type='l',
     main='Figure 13: Variances for Ideal Weighting',xlab="Quarter",ylab="Variance")
# mean: 8.602228
```

```
# function to calculate index and its variance in quarter q
# given a single asset n
calculate_index_in_quarter <- function(q, I_end, n) {
  # Find relevant dates
  # start of 9 mos ago, end of 3 mos ago
  # start of quarter, end of quarter
  date_indices = as.numeric(get_indices_for_quarter(q))
  # d1 = date_indices[1]  # don't need this one
  d2 = date_indices[2]
  q1 = date_indices[3]
  q2 = date_indices[4]
  # get the data for the given quarter
  q_data = df[q1:q2, names]
  # asset values at end of last quarter
  A_end = as.numeric(df[d2, names])
  # calculate the index values
  I = q_data[n] * I_end / A_end[n]
  # make it a vector
  I = as.numeric(unlist(I))
  # calculate the variance
  v = var(I)
  result = list(index_values=I, variance=v)
}
# function to get index (and variance per quarter)
# input: n ('index' of the asset)
get_single_asset_index <- function(n) {
  all_index_values = c()
  index_variances = rep(0, 10)
```

```
  I_end = 100

  for (q in 1:10) {
    result = calculate_index_in_quarter(q, I_end, n)

    index_variances[q] = result$variance

    # concatenate index values to one big vector
    all_index_values = c(all_index_values, result$index_values)

    # update I_end
    I_end = result$index_values[length(result$index_values)]
  }
  res = list(
    all_index_values=all_index_values,
    index_variances=index_variances)
  return(res)
}
```

```
#, fig.cap="Figure12"
knitr::include_graphics("bias-correction.png")
```

```
### test different lambda for ewma-price
par(mfrow=c(1,2))
lambdas = seq(0.5, 0.99999, length.out=30)
price_ewma_var = rep(0, length(lambdas))
for (i in 1:length(lambdas)) {
  lambda = lambdas[i]
  r = find_weights_all(get_weights_price_ewma)
  price_ewma_var[i] = mean(r$index_variances)
}
best_lam_price = which.min(price_ewma_var)
#cat("best lambda for ewma price:", lambdas[best_lam_price], "var:", price_ewma_var[best_lam_price], "\
# 0.6551693 --> 11.7104
plot(lambdas, price_ewma_var, type="l", main="Average Variance using Prices vs. Decay Rate",cex.main=0.

### test different lambda for ewma-log return
lambdas = seq(0.6, 0.99999, length.out=30)
logr_ewma_var = rep(0, length(lambdas))
for (i in 1:length(lambdas)) {
  lambda = lambdas[i]
  r = find_weights_all(get_weights_log_ret_ewma)
  logr_ewma_var[i] = mean(r$index_variances)
}
best_lam_logr = which.min(logr_ewma_var)
#cat("best lambda for ewma log return:", lambdas[best_lam_logr], "var:", logr_ewma_var[best_lam_logr],
# 0.7655131 --> 10.51929
plot(lambdas, logr_ewma_var, type='l',
     main='Average Variance using Log-Returns vs. Decay Rate', cex.main=0.7,
     xlab="Rate of Decay", ylab="Average Variance")
```

```
plot(result$all_index_values, type='l', col='red')#, main="index values comparison")
lines(result2$all_index_values, type='l', col='blue')
lines(result3$all_index_values, type='l', col='orange')
```

```
lines(result4$all_index_values, type='l', col='aquamarine')
lines(result_t$all_index_values, type='l', col='deeppink')
lines(result_ideal$all_index_values, type='l', col='green')
legend(
  "bottomright",
  inset=0.05,
  c('price', 'log-return', 'ewma price', 'ewma log-return', 'fitted-t log-return', 'ideal'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine', 'deeppink', 'green'))
```

```
##### plot the variance per quarter for each method
plot(result$index_variances, type='l', col='red')#, main="index variance per quarter comparison")
lines(result2$index_variances, type='l', col='blue')
lines(result3$index_variances, type='l', col='orange')
lines(result4$index_variances, type='l', col='aquamarine')
lines(result_t$index_variances, type='l', col='deeppink')
lines(result_ideal$index_variances, type='l', col='green')
legend(
  "topright",
  inset=0.05,
  c('price', 'log-return', 'ewma price', 'ewma log-return', 'fitted-t log-return', 'ideal'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine', 'deeppink', 'green'))
```

```
table<-matrix(c("Price", "Price EWMA","Log-Return","Log-Return t-dist",
                "Log-Return EWMA", "Ideal", 11.90243, 11.7104, 11.56243,
                11.56862, 10.51929, 8.602228), byrow=FALSE, ncol=2)
table<-data.frame(table)
colnames(table) <- c("Method", "Average Variance Per Quarter")
kable(table, caption ="Comparison of Methods")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
# get results for single assets
btc_result = get_single_asset_index(1)
usd_result = get_single_asset_index(7)
gold_result = get_single_asset_index(9)
# plot the results vs. chosen index
# plot BTC first because it's the largest
plot(btc_result$all_index_values, type='l', col='red',
     main="Figure 18: Our index values vs Single asset index")
lines(result4$all_index_values, type='l', col='blue')
lines(usd_result$all_index_values, type='l', col='orange')
lines(gold_result$all_index_values, type='l', col='aquamarine')
legend(
  "topright",
  inset=0.05,
  c('btc', 'mixed', 'usd', 'gold'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine'))
```

```r
# log of index values since raw values are too wide-ranging
plot(log(btc_result$all_index_values),
  type='l', col='red',
  main="Figure 19: Log of our index values vs Single asset index", ylim=c(4, 10))
lines(log(result4$all_index_values), type='l', col='blue')
lines(log(usd_result$all_index_values), type='l', col='orange')
lines(log(gold_result$all_index_values), type='l', col='aquamarine')
legend(
  "topright",
  inset=0.05,
  c('btc', 'mixed', 'usd', 'gold'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine'))
```

```r
# variance per quarter
plot(btc_result$index_variances, type='l', col='red',
     main="Figure 20: Our index variance vs. Single asset variance")
lines(result4$index_variances, type='l', col='blue')
lines(usd_result$index_variances, type='l', col='orange')
lines(gold_result$index_variances, type='l', col='aquamarine')
legend(
  "topright",
  inset=0.05,
  c('btc', 'mixed', 'usd', 'gold'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine'))
```

```r
plot(log(btc_result$index_variances),
  type='l', col='red',
  main="Figure 21: Log of our index variance vs. Single asset variance", ylim=c(-3, 15))
lines(log(result4$index_variances), type='l', col='blue')
lines(log(usd_result$index_variances), type='l', col='orange')
lines(log(gold_result$index_variances), type='l', col='aquamarine')
legend(
  "topright",
  inset=0.05,
  c('btc', 'mixed', 'usd', 'gold'),
  lwd=2,
  lty=c(1, 1, 1),
  col=c('red', 'blue', 'orange', 'aquamarine'))
```

```r
### Import Data
pmdata <- read.csv("datav2.csv")
#Create log price data
pmdata2 <- pmdata %>%
  mutate_at(dplyr::vars(-Date), log) %>%
  mutate(Date = as.Date(Date, format="%Y-%m-%d"))

#Create log return data
pmdata3 <- pmdata2 %>%
  mutate(BTC = c(0,diff(BTC)),
         ETH = c(0,diff(ETH)),
```

```
          EUR = c(0,diff(EUR)),
          GOLD = c(0,diff(GOLD)),
          JPY = c(0,diff(JPY)),
          SILVER = c(0,diff(SILVER)),
          CNY = c(0,diff(CNY)),
          USD = c(0,diff(USD)),
          XMR = c(0,diff(XMR)),
          XRP = c(0,diff(XRP))) %>%
  dplyr::filter(BTC != 0)
```

```
### Fitting ARIMA Models
#Append a column containing the quarter number to the dataset
quarternum <- function(Date){
  if (Date <= "2016-12-31"){
    1
  } else if (Date <= "2017-03-31"){
    2
  } else if (Date <= "2017-06-30"){
    3
  } else if (Date <= "2017-09-30"){
    4
  } else if (Date <= "2017-12-31"){
    5
  } else if (Date <= "2018-03-31"){
    6
  } else if (Date <= "2018-06-30"){
    7
  } else if (Date <= "2018-09-30"){
    8
  } else if (Date <= "2018-12-31"){
    9
  } else if (Date <= "2019-03-31"){
    10
  } else if (Date <= "2019-06-30"){
    11
  } else {
    12
  }
}
#First remove the final quarter from the log-prices dataset
pmdata5 <- pmdata2 %>%
  rowwise() %>%
  mutate(quarter = quarternum(Date))
pmdata6 <- pmdata5 %>%
  dplyr::filter(quarter == 12) %>%
  dplyr::select(-quarter)
pmdata5 <- pmdata5 %>%
  dplyr::filter(quarter != 12) %>%
  dplyr::select(-quarter)
results <- matrix(0, nrow=10, ncol=4)
for (i in 2:11){
  model <- auto.arima(pmdata5[,i], max.p=5, max.q=5, ic="bic")
  results[i-1,1:3] = arimaorder(model)
  results[i-1,4] = model$bic
```

```
}
rownames(results) <- colnames(pmdata5[,-1])
colnames(results) <- c("p","d","q","BIC")
```

```
results %>% kable(caption="Optimal Arima Models") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

```
par(mfrow=c(2,5))
for (i in 1:10){
  temp <- auto.arima(pmdata5[,i+1], max.p=5, max.q=5, ic="bic")
  acf(temp$residuals, lag.max=50, main=rownames(results)[i])
  Box.test(temp$residuals, lag=5, type="Ljung-Box", fitdf=(results[i,1]+results[i,3]))
}
```

```
#Box-Ljung Tests
#Note, the value of k was manually changed from 2 to 11 as the Box.test function does not
#display results when inside a loop
k = 11
temp <- auto.arima(pmdata5[,k], max.p=5, max.q=5, ic="bic")
Box.test(temp$residuals, lag=5, type="Ljung-Box", fitdf=results[k-1,1]+results[k-1,3])
Box.test(temp$residuals, lag=10, type="Ljung-Box", fitdf=results[k-1,1]+results[k-1,3])
Box.test(temp$residuals, lag=25, type="Ljung-Box", fitdf=results[k-1,1]+results[k-1,3])
```

```
### Forecasting
days <- nrow(pmdata2) - nrow(pmdata5)
lb <- pmdata6
est <- pmdata6
ub <- pmdata6
for (i in 2:11) {
  temp <- auto.arima(pmdata5[,i], max.p=5, max.q=5, ic="bic")
  predtemp <- predict(temp, n.ahead=days, se.fit=TRUE)
  lb[,i] = predtemp$pred - 1.96*predtemp$se
  est[,i] = predtemp$pred
  ub[,i] = predtemp$pred + 1.96*predtemp$se
}
forecastlb <- as.data.frame(pmdata5 %>% bind_rows(lb))
forecastdata <- as.data.frame(pmdata5 %>% bind_rows(est))
forecastub <- as.data.frame(pmdata5 %>% bind_rows(ub))
p <- list()
for (i in 2:11){
  newdf <- data.frame(date=pmdata2$Date, Actual=pmdata2[,i], Forecast=forecastdata[,i],
                      Lower.Bound = forecastlb[,i], Upper.Bound=forecastub[,i]) %>%
    dplyr::filter(date>="2018-07-01") %>%
    gather("Series", "Log.Price", 2:5)
  p[[i-1]] <- ggplot(newdf, aes(x=date, y=Log.Price, color=Series)) +
    geom_line() + theme(legend.position="none") +
    scale_color_manual(values=c("black", "steelblue", "firebrick", "sea green")) +
    aes(group=rev(Series)) +
    ggtitle(colnames(pmdata2)[i])
}
do.call(grid.arrange, c(p, ncol=3, top = "Figure 23: Log Price Forecasts"))
```