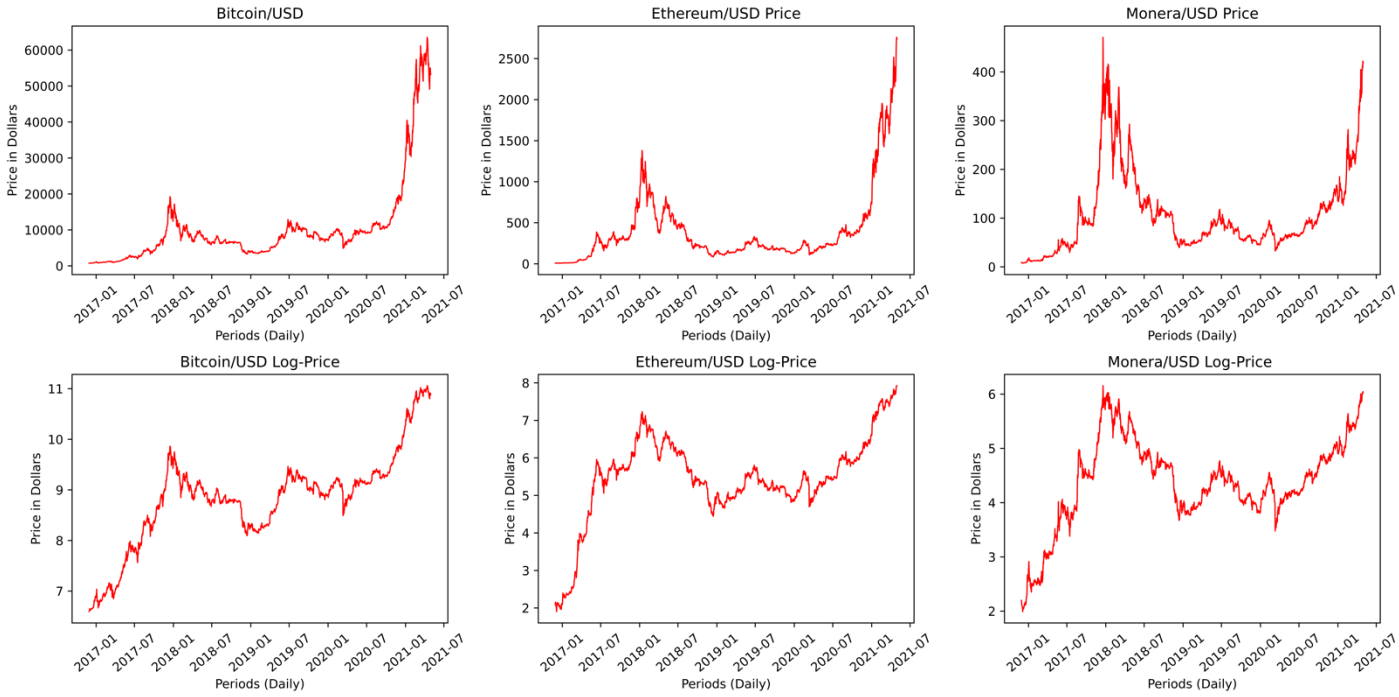
Empirical Methods in Finance – Assignment 2

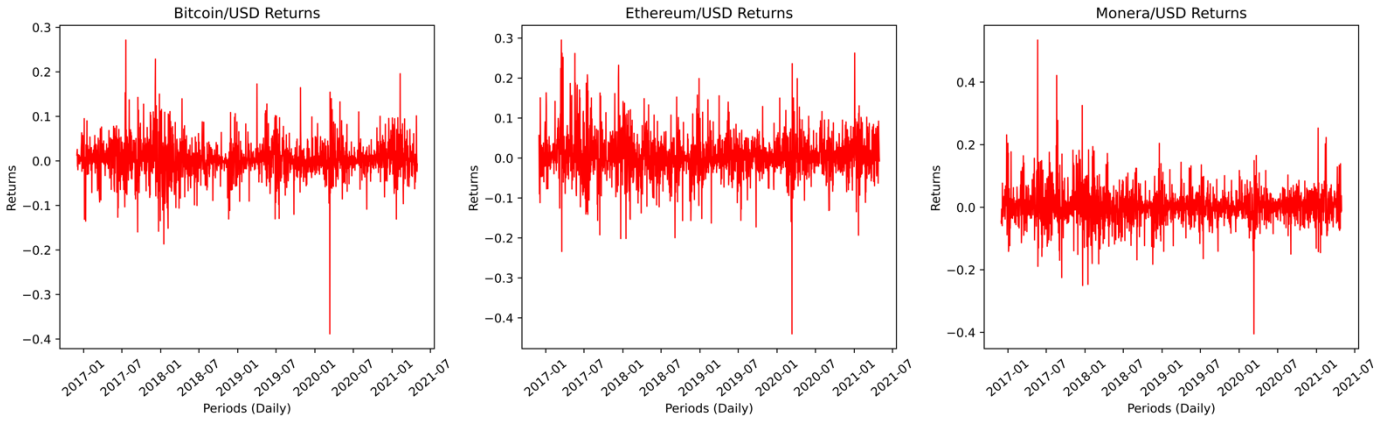
I. Introduction:

What is a cryptocurrency? The question has been asked more and more in recent years due to their popularization and their decidedly extreme crazy growth. A "Crypto" is a digital asset where the transactions are verified and controlled by a decentralized system called the Blockchain, a distributed ledger system enforced by a network of computers, instead of being controlled by a centralized authority. This makes it totally independent of any government manipulation. Secured by a cryptography, it is nearly impossible to counterfeit this asset.

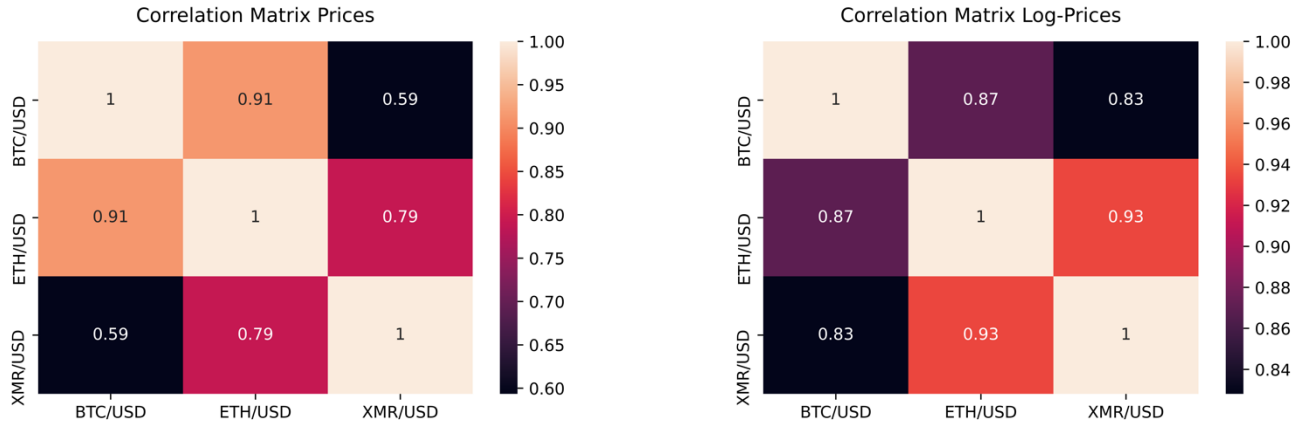
Since several years, their popularization has grown exponentially especially with the most popular of all cryptos: The Bitcoin, created in January 2009 by the pseudonymous Sathoshi Nakamoto. Globally, their prices are experiencing unprecedented growth, attracting the attention of many investors. It is for this reason that we will analyze through this report financial characteristics of three different and well-known cryptocurrencies: Bitcoin (BTC), Ethereum (ETH) and Monero (XMR). Our dataset goes from November 30, 2016 to April 30, 2021.



Log-prices were also computed. Each cryptocurrency saw their prices increased a lot around 2020, months after the beginning of the Covid-19 crisis, taking advantage of high market volatility. They are also considered as hedging instruments against inflation since cryptos are independent from Central Banks. Simple price returns are given below:

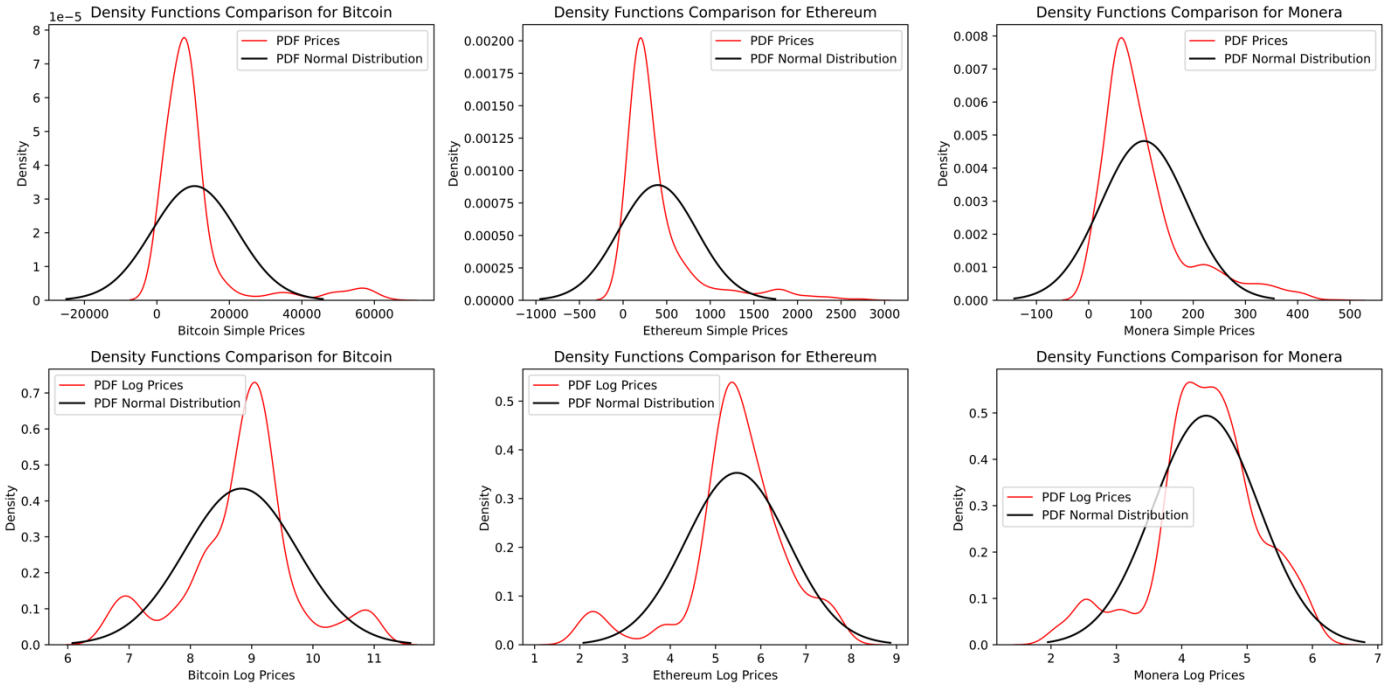


We can also see the relation between each cryptocurrency by computing correlation through the following matrix.



All our cryptocurrency's prices seem to be significantly correlated, especially with Ethereum and Bitcoin (correlation of 0.91). For the log prices, all series are even more correlated between them (correlation of 0.93 for Ethereum and Monera). We can have a look on the statistical and probabilistic properties of each time series by computing their empirical distribution functions. We computed at first respectively the empirical firsts moments for each time series:

		Mean $\hat{\mu}_i$	Volatility $\hat{\sigma}_i$	Skewness $\hat{S}$	Kurtosis $\hat{K}$	Minimum	Maximum
BTC/USD	*Simple*	10407	11798	2.85	10.93	740.0	63537
	Log	8.83	0.92	-0.188	3.63	6.60	11.06
ETH/USD	*Simple*	398.5	449.3	2.54	9.81	6.702	2758.6
	Log	5.47	1.13	-0.932	4.87	1.9	7.92
XMR/USD	*Simple*	106.028	82.808	1.621	5.464	7.323	470.99
	Log	4.37	0.807	-0.502	3.502	1.99	6.155



None of our time series really fit the normal distribution. Simple prices have a positive skewness, meaning that the curve of the density function is left leaning, the main part of the distribution is concentrated in the left part. Kurtosis for simple prices is very high, far superior to three (kurtosis of a normal distribution), meaning that extreme values are more likely to occur. However, concerning the log-prices, the kurtosis of each times series is close to three. Their skewness is negative, so the distribution is right leaning.

Thus, we will study in depth the statistical and econometric character of these time series, and then we will establish a trading strategy to make profit with cryptocurrencies. Finally, at the end of this report we will study the relation between the Bitcoin and several macroeconomic variables by building a Vector Autoregressive model.

II. Critical Values Simulation:

The goal of this part is to test if our time series are stationarity or not. A time series is stationary when its properties like its expected value and standard deviation do not depend on time. If the time series follows trends like constant growth with our cryptocurrencies' price, the process is not stationary. To check if our series are effectively stationary, we can apply the Dickey-Fuller test. We will use log-prices p_t to run the following regression:

$$p_t = \alpha + \beta p_{t-1} + u_t \quad ; \quad p_t = \log(P_t) \quad ; \quad H_0 : \begin{cases} \beta = 1 \\ \alpha = 0 \end{cases} \quad ; \quad H_1 : \beta < 1$$

In this case, there is a constant term. We will test the null hypothesis that the process is non-stationary. Indeed, if $\beta \geq 1$ the process won't be stationary, p_t is explosive and its variance doesn't exist. If it is really the case, p_t will be a random walk as $p_t = p_{t-1} + u_t$. In the other hand, with a $\beta < 1$ the process will be stationary. To run this test, we have first to compute the test-statistic as follow:

$$\tau_\alpha = \frac{\hat{\beta} - 1}{std(\hat{\beta})} = \frac{\hat{\beta} - 1}{[\hat{\sigma}_u^2 / \sum_{t=1}^{T}(p_{t-1} - \bar{p})^2]^{1/2}} \quad ; \quad \hat{\sigma}_u^2 = \frac{1}{T-1}\sum_{t=1}^{T}\left(p_t - \hat{\alpha} - \hat{\beta} p_{t-1}\right)^2$$

The index α means that our model is estimated with a constant term. Then, we reject the null hypothesis if we have $\tau_\alpha > Z$ where Z is the critical value depending on which level (for example: 5%, 10% or 1%) we choose.
Now, if instead of having the precedent model for our test, let's assume that we want to run the following regression:

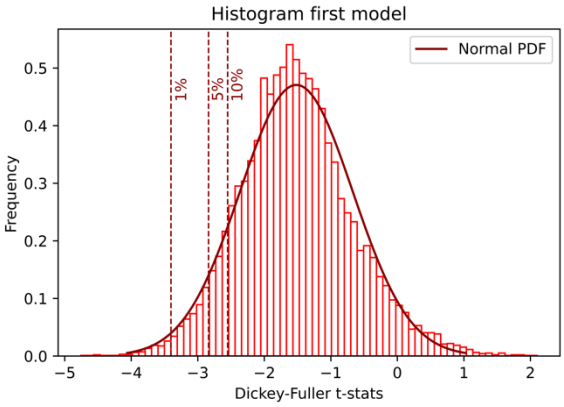
$$r_t = p_t - p_{t-1} = \Delta p_t = \alpha + \beta p_{t-1} + u_t - p_{t-1} = \alpha + (\beta - 1)p_{t-1} + u_t = \alpha + \beta^* p_{t-1} + u_t$$

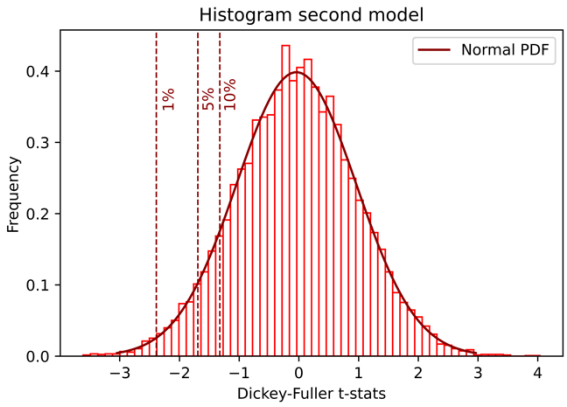
$$\tau_\alpha^* = \frac{\hat{\beta}^* - 0}{std(\hat{\beta}^*)} = \frac{\hat{\beta} - 1}{std(\hat{\beta})} = \tau_\alpha$$

So, in this situation we can rewrite the new null hypothesis as $H_0^* : \beta^* = 0 \Leftrightarrow \beta = 1$ and $\alpha = 0$. Indeed, if the null hypothesis is not rejected, we obtain $p_t - p_{t-1} = u_t \Leftrightarrow p_t = p_{t-1} + u_t$, it corresponds to a Random Walk so the process is not stationary. It is equivalent to the previously defined null hypothesis H_0. From now, we computed our own critical values for our own sample data size T = 1613 daily observations:

1)	*Simulate time series error $\varepsilon_t^{(i)} \sim \mathcal{N}(0,1)$ with $t = 1, \dots, 1613$.*
2)	*Compute random walk process of prices. $p_t^{(i)} = p_{t-1}^{(i)} + \varepsilon_t^{(i)}$.*
3)	*Estimate the $AR(1)$ model: $p_t^{(i)} = \alpha^{(i)} + \beta^{(i)} p_{t-1}^{(i)} + u_t^{(i)}$.*
4)	*Compute the test statistic $t(\beta^{(i)})$ for the null hypothesis from the DF test.*
5)	*Repeat the precedent actions N = 10000 times.*

We simulate Random Walks because they are what characterize log-price time series. Most of the time, log-price processes follow trends. Also, in the Dickey-Fuller test, we assume non-stationarity. We performed the same steps as before by assuming this time that the AR (1) model is $p_t = 0.2 * p_{t-1} + \varepsilon_t$. After computing the several t-stats of the Dickey-Fuller test, we obtain the following histograms, corresponding to the $t(\beta^{(i)})$, and our Critical Values:





The above distribution represents the density of our simulated t-statistics, it indicates where the different values are the most likely to occur. In the first model, we can see that the distribution is positively skewed, it doesn't have a perfect symmetry and is not centered around its mean like the Normal distribution as we can see from the graph.

	Threshold at 1%	Threshold at 5%	Threshold at 10%
First model	-3.4697	-2.86925	-2.55415
Second model	-2.387	-1.6916	-1.3235
Dickey-Fuller	-3.51	-2.89	-2.58

The quantile of 1%, as shown in the graphs, represents the probability that the t-statistics will be lower or equal to the corresponding critical value. It is the same principle for the other quantiles of 5% and 10%.
Concerning the second distribution, a reason that could explain the fact that its shape is approximately a standardized Normal distribution (with a perfect symmetry around zero) is the use in the previous simulation of time series errors ε_t that follows a normal distribution $\mathcal{N}(0,1)$. The second model have a model with a beta of 0.2, already assuming stationarity in our time series. The t-stat in this case was $\tau = (\hat{\beta} - 0.2)/std(\hat{\beta})$.

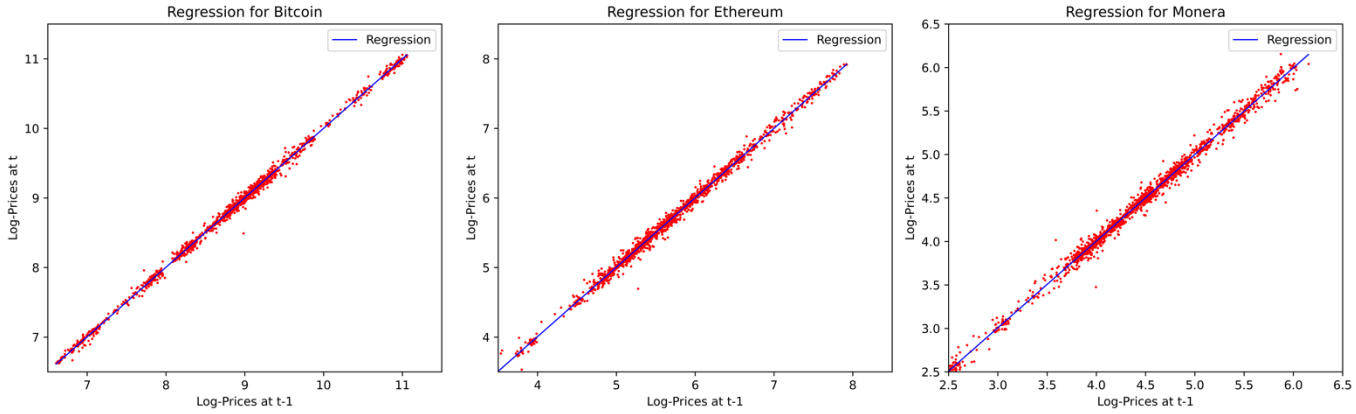
III. Testing for stationarity:

From now we finally turned on our cryptocurrencies. At first, we estimated the parameters β_i for i = BTC, ETH and XMR by performing a linear regression and obtained the following representation.

$$p_{t,i} = \alpha_i + \beta_i * p_{t-1,i} + u_{t,i}$$

	BTC	ETH	XMR
Estimator $\hat{\alpha}_i$	0.0149	0.0174	0.0185
Estimator $\hat{\beta}_i$	0.9986	0.9975	0.9963

For each cryptocurrency, the estimates for the β_i coefficients are all very close to one. As we saw previously, a coefficient equal to one would mean that the process won't be stationary, it would follow a random walk under H_0.



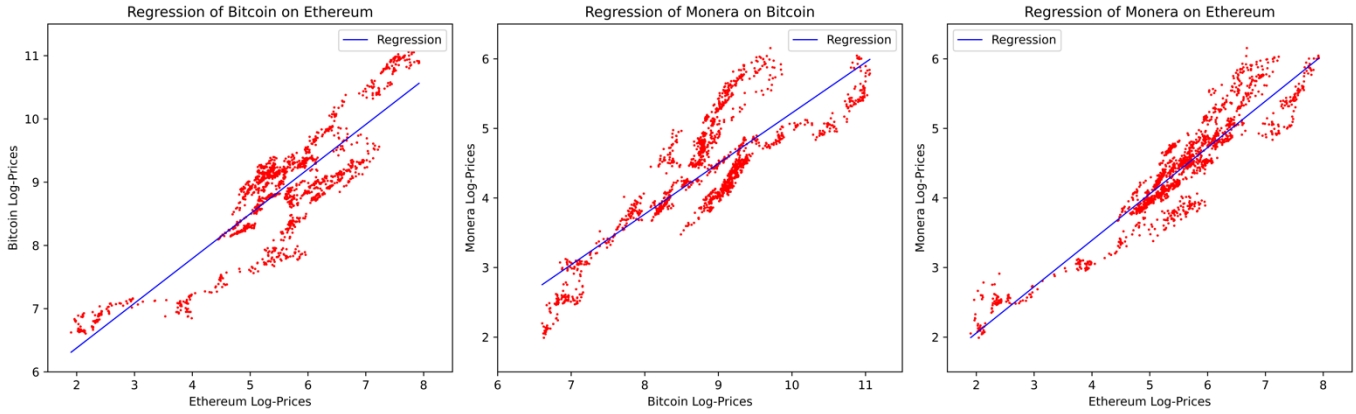
The t-stats τ_α were also computed for each cryptocurrency using the formula that we exposed previously. We obtained -1.186 for the Bitcoin regression, -1.9958 for Ethereum and -1.9925 for Monera. To remind, the critical values that we found before corresponding to the Dickey-Fuller test are -2.55 at the threshold of 10%, -2.869 at the threshold of 5% and -3.47 at 1%. For each time series, the corresponding t-stat is higher than all critical values meaning that the null hypothesis is not rejected. Our time series are in consequence not stationary.
In the case where we would perform a regression between cryptocurrency pairs, so non-stationary variables, we would obtain a spurious regression, meaning that these two variables would be associated but not causally related. In theory, the coefficient b of the regression between two independent stationary series (for example an asset return A and another asset return B) would be equal to zero according to Granger and Newbold. But they reported that $H_0 : \beta = 0$ (which is the true value) would be rejected in 76% of the cases if our series are two independent random walks, so non-stationary. But if we show that variables are cointegrated (if a long-term relation exists between them), the regression of a pair of variables will not be spurious.

IV. Cointegration:

We now turned to the cointegration relationship between the cryptocurrencies of our dataset. We tested for each pair of cryptos if there exist cointegration between them. We run the following regression for pair A and B using log-prices.

$$p_t^{(A)} = a + b * p_t^{(B)} + z_t$$

	BTC-ETH	XMR-BTC	XMR-ETH
Estimator $\hat{a}$	4.965	-2.051	0.7238
Estimator $\hat{b}$	0.7067	0.7272	0.6667

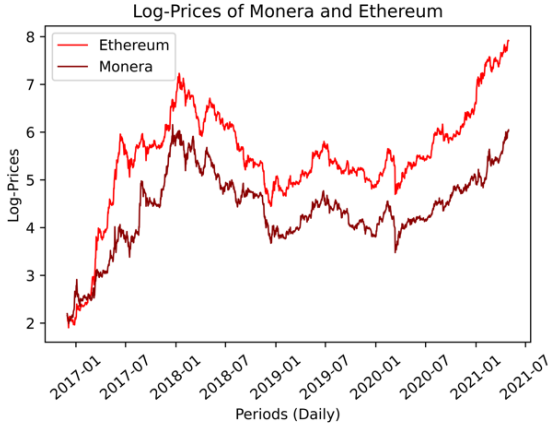


We observe that our b estimates are not equal to zero. As we said before, a regression between two non-stationary variables would lead to a spurious regression where the coefficient is not equal to zero. However, it would not be spurious if we find a cointegration relation between the variables. Then, we used the Dickey-Fuller test to check if the residuals of the regression z_t are stationary. We estimated the regression:

$$\hat{z}_t - \hat{z}_{t-1} = \Delta\hat{z}_t = \alpha + \beta\hat{z}_{t-1} + u_t \quad ; \quad \tau_\alpha = \frac{\hat{\beta} - 0}{std(\hat{\beta})}$$

	BTC-ETH $z_t^{(1)}$	XMR-BTC $z_t^{(2)}$	XMR-ETH $z_t^{(3)}$
Estimator $\hat{\alpha}$	0.000126	0.0004582	0.00000456
Estimator $\hat{\beta}$	-0.00257	-0.0054199	-0. 012
t-statistics τ_α	-1.4036	-2.1777	-3.11728

Our null hypothesis here is that $\beta = 0$ and $\alpha = 0$, so that our equation $\Delta\hat{z}_t$ would be a Random Walk $z_t = z_{t-1} + u_t$. The critical values of our test are -3.07 at the threshold of 10%, -3.37 at the threshold of 5% and -3.96 at 1%. Among all our variables, only the residuals of the regression of Monera on Ethereum are stationary at the threshold of 10%. Residuals $z_t^{(3)}$ are $I(0)$, Monera and Ethereum are both $I(1)$. This leads to the conclusion that these two cryptocurrencies are cointegrated. Cointegration is the measure that analyze the co-movement of two prices time series. It detects the long-term relationship between both variables. It is similar to the principle of correlation, but they are not the same since correlation measures the short-term relationship between the variables. Then, it seems that the log-prices of Monera and Ethereum have a similar movement on the long run, justifying the fact that they are cointegrated:



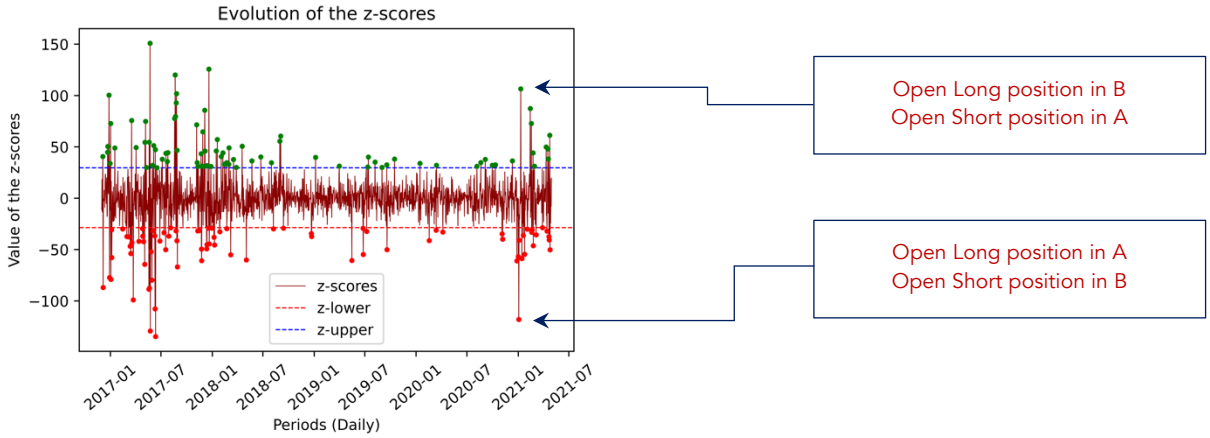
V. Pair Trading:

In this part, we implemented a profitable pair-trading strategy using the pair of cryptocurrencies A and B that are cointegrated (in our case, Monera and Ethereum). The spread between the corresponding log-returns was computed:

$$Spread_t = r_{A,t} - r_{B,t} \quad ; \quad zscore_t = \frac{Spread_t - \overline{Spread}}{\sigma_{Spread}}$$

$$zscore_{upper} = q_{95\%}(zscore) \quad ; \quad zscore_{lower} = q_{5\%}(zscore) \quad ; \quad zscore_{out} = q_{40\%}(zscore)$$

This trading strategy, build by using the different quantiles of z-score, can be summarized by the following scheme:



So, if we didn't open a position before and if our z-score at time t is higher than z-upper, we opened our Long position on Ethereum and Short position on Monera. Then, the positions are closed if on the later our z-score is lower or equal to the value of z-out. In the other case, if our z-score is lower than z-lower, we opened our Long position on Monera and Short position on Ethereum. Then, the positions are closed if on the later our z-score is higher or equal to the value of minus z-out. Concerning our weights, when we open our position at time $t = j$, we have $w_{A,j} = w_{B,j} = 0.5$ since we invest 50% of our wealth W. Then, they change over time with the change of log-prices. Besides, since we short one of the cryptocurrencies, a decrease in the price is wanted to have positive return in this asset. So, we used the formulas by putting a minus in front of the return of the short cryptocurrency (for example the crypto B is short sell):

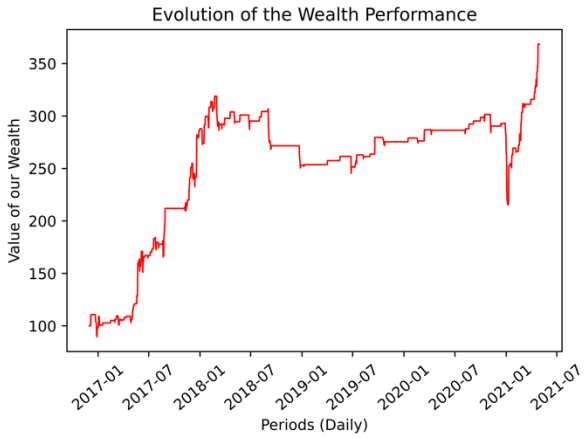
$$w_{A,t} = \frac{Value\ of\ A\ at\ t}{Value\ of\ Portfolio\ at\ t} = \frac{w_{A,t-1} * \exp(r_{A,t})}{w_{A,t-1} * \exp(r_{A,t}) + w_{B,t-1} * \exp(-r_{B,t})}$$

With the log returns $r_{i,t}$ for each cryptocurrency. Then, for the dates after opening a position, the log returns of our portfolio and our new wealth were computed as follow and we repeat these steps until we close the position:

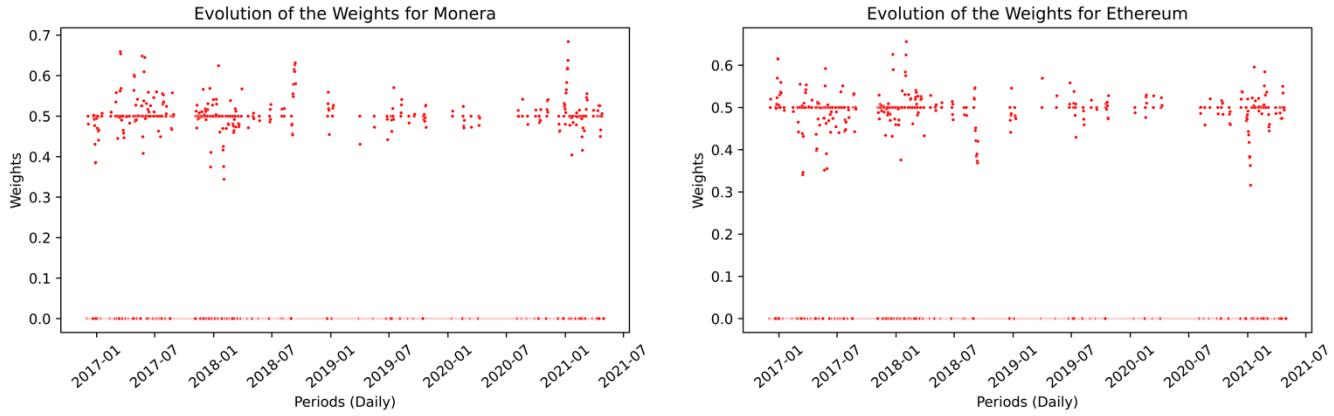
$$r_{p,t} = \log\left[w_{A,t-1} * \exp(r_{A,t}) + w_{B,t-1} * \exp(-r_{B,t})\right] \quad ; \quad W_t = W_{t-1} * \exp(r_{p,t})$$

If the position is close, for the time after it (so with no Long-A or Long-B) our wealth will not change over time $W_t = W_{t-1}$. This strategy relies strongly on cointegration. If two series are cointegrated, there spread mean reverts. When they diverge at some dates, they will be pulled back together. It is the core concept of pair trading. Essentially, it will identify the periods where the difference of prices between the two assets would be larger than it ought to be before decreasing on the later. At this moment, a long position in the undervalued stock and a short position in the overvalued stock should be taken before closing all the positions when the prices difference returns around an expected level. It would lead to profits that are independent from the market trend making the trading strategy a market neutral one.

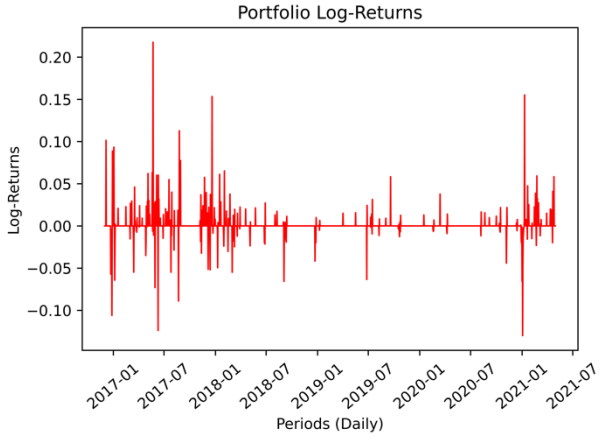
However, this strategy does not exploit a true arbitrage opportunity since it is not guaranteed at 100% that it would lead to strictly positive profits in the future. But we bet that the difference between both time series movement gravitates around the same value so when the difference jumps (so if one price jumps), it will be back very soon to the mean value, identifying if an asset is eventually overvalued or undervalued.



Our strategy seems to be profitable, as the performance of our wealth hit above 350 at the end of the investment period. We can see that even if our wealth is globally increasing through time, sometimes our performance falls like for instance in January 2021. This justifies our previous statement that this strategy does not provide always profits but it can also experience performance fall. The evolution of our weights is given below:



Globally, our weights gravitate around 0.5 for each cryptocurrency. Due to the price change through time, our weights are adjusted at each time t from the period following the opening of the position to the closing of it. We have also the representation of our portfolio return and the statistical characteristics of our performance:

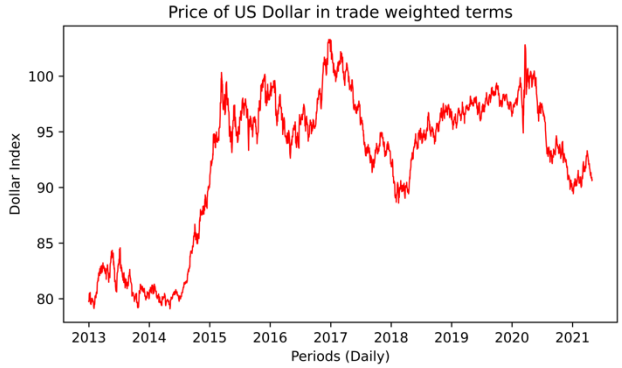


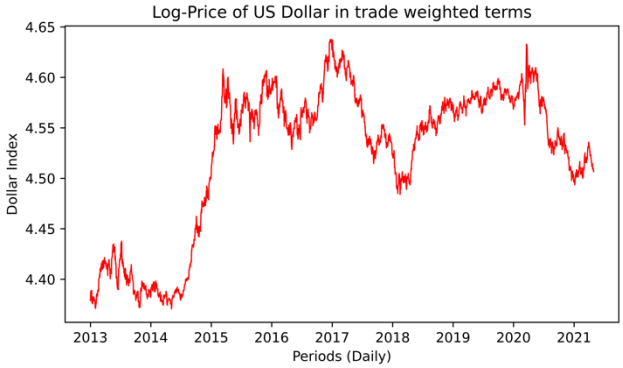
	Mean $\hat{\mu}_i^{Ann}$	Volatility $\hat{\sigma}_i^{Ann}$	Skewness $\hat{S}$	Kurtosis $\hat{K}$	Minimum	Maximum
Wealth Perf	251.59	60.692	-1.41	3.925	89.49	368.45
Portfolio Returns	0.2039	0.2316	3.189	66.39	-0.129	0.218

$$Strategy\ Total\ Return = \frac{W_T - W_0}{W_0} = \frac{368.45 - 100}{100} = 2.6845 \Leftrightarrow 268.45\%$$

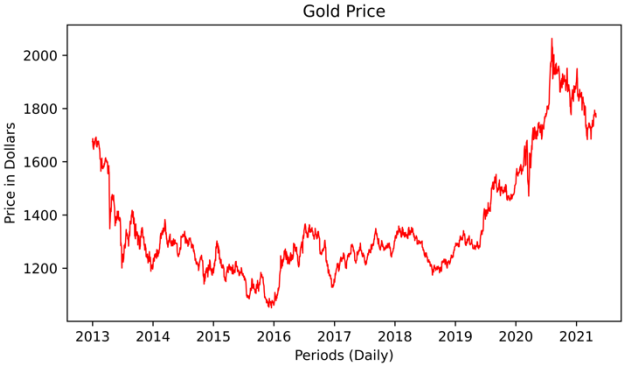
VI. VAR Models:

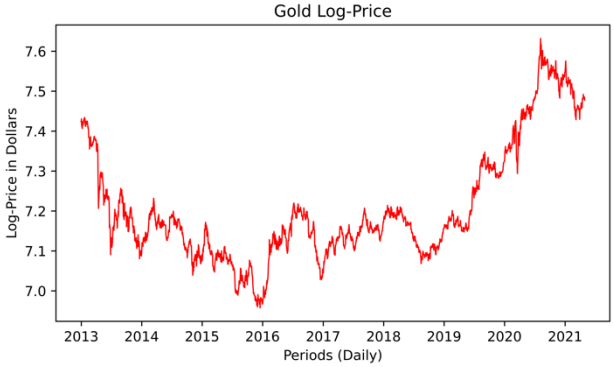
From now, we will analyze the evolution of the Bitcoin price with other macroeconomic and assets variables. Our first time series is the price of US Dollar in trade weighted terms. Known as the "Broad Index", it is an index created by the Federal Reserve system of the United States, or the FED, in 1998 in response to the introduction of the Euro. Its goal is to measure the purchasing value of the US Dollar and the effects of dollar appreciation and depreciation against foreign currencies. This index gives weight to currencies most widely used in international trade.



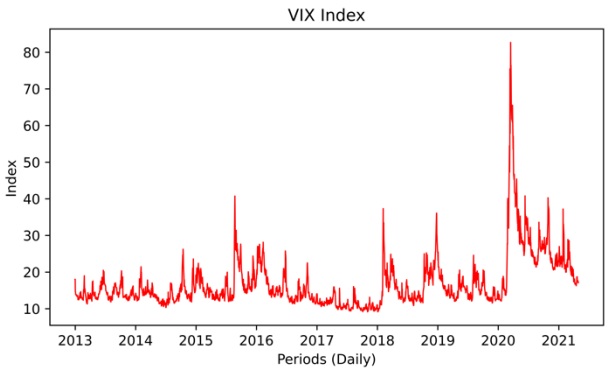


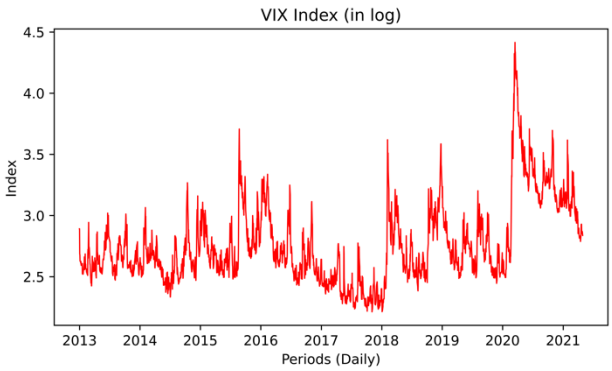
Then, we have the Price of gold. It is a commodity asset and is considered as a hedging instrument during time of crisis. It is interesting to note that the Bitcoin during the last years has been considered as an alternative of gold to hedge against inflation for example. Indeed, like the bitcoin, the quantity of gold on the market is not controlled by central banks and so the inflation linked to gold is limited to quantities mined, which are relatively small compared to the volume already in circulation.



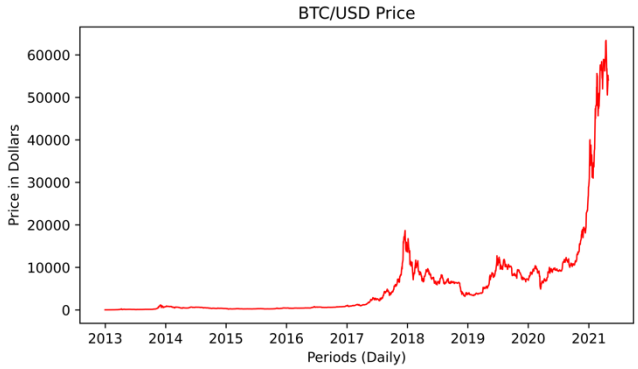


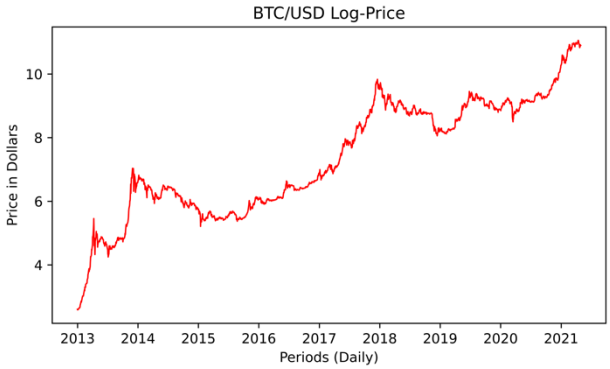
We have also the level of the VIX Index or also called the "Fear Index". It is the most popular measure of stocks market expected volatility based on S&P 500 index options. It is established daily by the Chicago Board Options Exchange (CBOE) and calculated by computing the 30-day expected annual volatility on Call and Put Options in real time. It permits to have a manner to evaluate market sentiment.



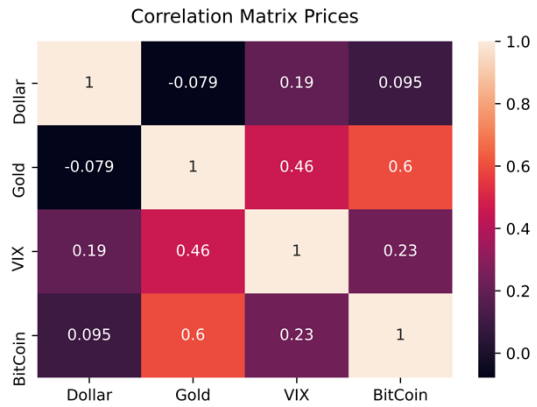


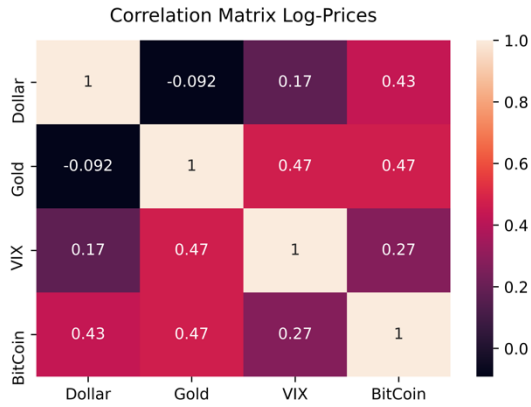
Finally, we have the bitcoin that we presented in the previous part. We will check if the price evolution of the most important current cryptocurrency has an impact on the behavior of our other variables.





We can also have a look on how the time series are correlated between them by computing the correlation matrices:





Not all of our variables are extremely correlated between each other. The most notable correlated series are Gold and Bitcoin (correlation of 0.47 for the log series), VIX and Gold (correlation of 0.47 for the log). To check for stationarity, we used their log-prices $p_t = \log(P_t)$ and we run the Augmented Dickey-Fuller test. Compared to the precedent Dickey-Fuller test where innovations were supposed to be normal i.i.d, the ADF test introduce k lags of Δp_t in the regression to capture the serial correlation in residuals. In general, k is unknown but for financial data we can fix $k = T^{1/4}$. It is possible to introduce additional lags until the residual series is found to be serially uncorrelated. Under the null hypothesis $H_0 : \beta = 1$ and $\alpha = 0$ The t-stat is the same as in the previous Dickey-Fuller test:

$$\tau_\alpha = \frac{\hat{\beta} - 1}{std(\hat{\beta})} \quad ; \quad p_t = \alpha + \beta p_{t-1} + \sum_{i=1}^{k-1} \delta_i \Delta p_{t-i} + \varepsilon_t$$

	Dollar Index	*Gold*	*VIX Index*	*Bitcoin/USD*
t-statistics	-1.969	-1.30	-4.639	-1.596
p-value	0.30	0.629	0.00011	0.486

Critical Values are -3.43 for the threshold of 1%, -2.86 for the threshold of 5% and -2.57 for 10%. All log-prices time series are non-stationary, except for the VIX Index which its t-statistics is lower than all critical values. Furthermore, we decided to compute the KPSS test to complete the precedent statement. It is another of the commonly used test to check for time series stationarity. The null hypothesis H_0 is that the process is stationary around a deterministic trend, it this exact opposite of the Augmented Dickey-Fuller test. We run the following regression model:

$$p_t = \beta' D_t + \mu_t + \eta_t \quad ; \quad \mu_t = \mu_{t-1} + \varepsilon_t \quad ; \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon)$$

With D_t the matrix containing the deterministic components (a constant and/or a trend). The Test Statistic is given by:

$$KPSS = T^{-2} \sum_{t=1}^{T} \frac{\hat{S}_t^2}{\hat{\lambda}^2} \quad ; \quad \hat{S}_t = \sum_{j=1}^{t} \hat{v}_j$$

With $\hat{v}_j$ the residual of the regression of p_t on D_t and $\hat{\lambda}$ a consistent estimator of the long-run variance of v_t. Under H_0, we have $\sigma_\varepsilon^2 = 0$. We rejected the null hypothesis when $KPSS >$ Critical Values at some threshold, so it means that σ_ε^2 would be higher than zero. Thus, it would imply the existence of a random walk or a trend component in the process.

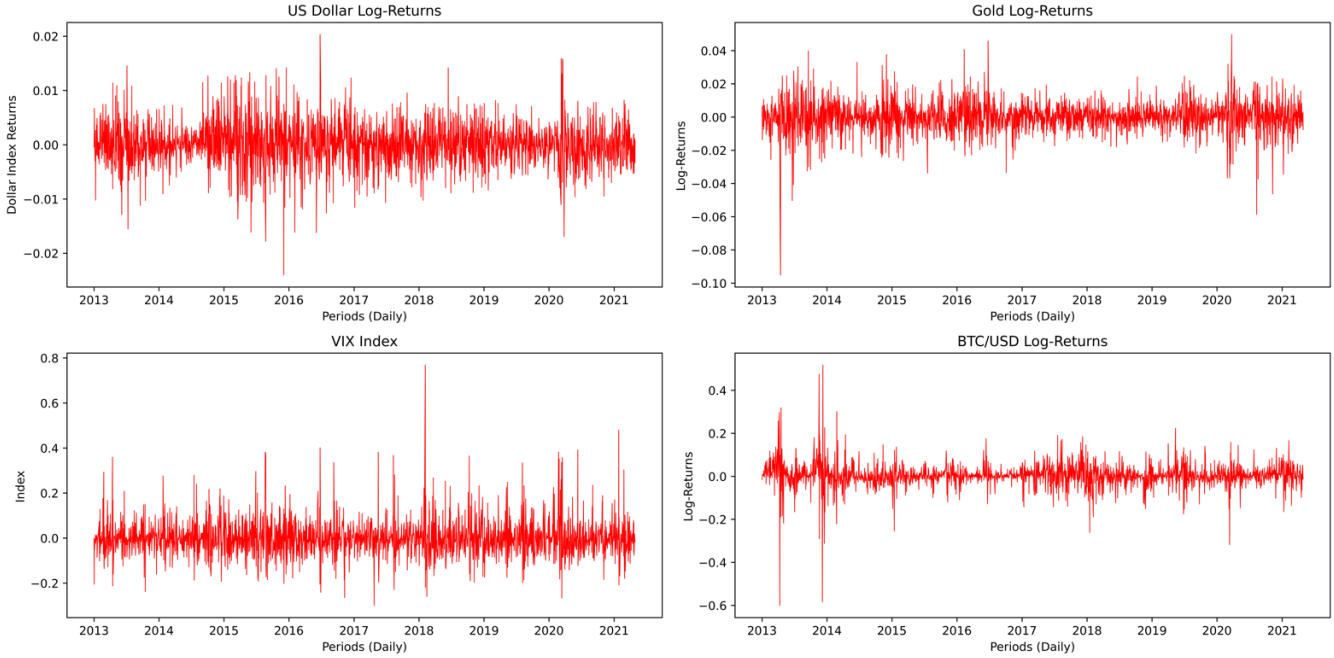
	Dollar Index	Gold	VIX Index	Bitcoin/USD
Test statistic	3.67	3.159	1.792	6.69
p-value	0.01	0.01	0.01	0.01

Critical values are 0.347 at the threshold of 10%, 0.463 at the threshold of 5% and 0.739 at 1%. For all our variables, the test statistic is higher than all critical values. In consequence, H_0 is rejected and these series are not stationary. However, this statement contradicts the conclusion of the Augmented Dickey-Fuller that we computed above where only the VIX was found to be stationary.

If the KPSS test indicates non-stationarity but the ADF test indicates stationarity, this means that the VIX Index process could be difference stationary. To be sure and to manipulate stationary processes on further analysis, we transformed for all variables. To transform these series into stationary ones, we computed differencing equation as $\Delta p_t = p_t - p_{t-1} = r_t$, which are variables log-returns. We run again the Augmented Dickey-Fuller and the KPSS test to check if our new time series are in this case stationary or not.

	Dollar Index	Gold	VIX Index	Bitcoin/USD
ADF t-stat	-19.88	-46.28	-19.22	-9.98
ADF p-value	0	0	0	0
KPSS test-statistic	0.265	0.412	0.0146	0.195
KPSS p-value	0.10	0.072	0.10	0.10

By looking at the following graphs representation of each variables, we see that they all gravitate around the zero axe:



In this case, all our series are stationary for each threshold (1%, 5% and 10%). We kept these series to estimate a Vector Autoregressive Model (VAR). Our model VAR (p) can be represented as follow for n assets with the constant c_i :

$$R_t = \Phi_0 + \Phi_1 R_{t-1} + \cdots + \Phi_p R_{t-p} + \varepsilon_t$$

$$\Leftrightarrow \begin{bmatrix} r_{1,t} \\ \vdots \\ r_{n,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} + \begin{bmatrix} \Phi_{11}^{(1)} & \cdots & \Phi_{1n}^{(1)} \\ \vdots & \ddots & \vdots \\ \Phi_{n1}^{(1)} & \cdots & \Phi_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} r_{1,t-1} \\ \vdots \\ r_{n,t-1} \end{bmatrix} + \cdots + \begin{bmatrix} \Phi_{11}^{(\mathcal{P})} & \cdots & \Phi_{1n}^{(\mathcal{P})} \\ \vdots & \ddots & \vdots \\ \Phi_{n1}^{(\mathcal{P})} & \cdots & \Phi_{nn}^{(\mathcal{P})} \end{bmatrix} \begin{bmatrix} r_{1,t-\mathcal{P}} \\ \vdots \\ r_{n,t-\mathcal{P}} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,t} \\ \vdots \\ \varepsilon_{n,t} \end{bmatrix}$$

First, we had to choose a lag $\mathcal{P}^*$. To determine it, Information Criteria was used with the Akaike Information Criteria (AIC), Bayesian Information Criterion (BIC), Final Prediction Error (FPE) and Hannan-Quinn (HQ) criterion defined for $\mathcal{P} = 0, \ldots, \mathcal{P}_{max}$. We compute then the minimization of each criterion such as $AIC(\mathcal{P}^*) = \min AIC(\mathcal{P})$. The overall idea is to maximize the information captured by the model. The lowest the $\log|\hat{\Sigma}^{(\mathcal{P})}|$, the better is the model.

$$AIC(\mathcal{P}) = \log\left|\hat{\Sigma}^{(\mathcal{P})}\right| + \frac{2\mathcal{P}n^2}{T} \quad ; \quad BIC(\mathcal{P}) = \log\left|\hat{\Sigma}^{(\mathcal{P})}\right| + \frac{\log(T)\mathcal{P}n^2}{T}$$

$$FPE(\mathcal{P}) = \left|\hat{\Sigma}^{(\mathcal{P})}\right| + \left(\frac{T + n\mathcal{P} + 1}{T - n\mathcal{P} - 1}\right)^n \quad ; \quad HQ(\mathcal{P}) = \log\left|\hat{\Sigma}^{(\mathcal{P})}\right| + 2\frac{\log(\log T)}{T}\mathcal{P}n^2$$

With the covariance matrix between P_t and ε_t. T is the number of periods (here $T = 2173$) and we have $n = 4$ different variables. After computing the different criteria, we came out to the conclusion that the necessary lag for our model was $\mathcal{P}^* = 1$ so that our model can be a VAR (1). Our results don't change when the lag order increases:

	AIC	BIC	FPE	HQIC
Lag 1	-31.37	-31.36	2.388e-14	-31.36
Lag 2	-31.37	-31.32	2.376e-14	-31.35
Lag 3	-31.37	-31.27	2.379e-14	-31.33
Lag 4	-31.37	-31.23	2.383e-14	-31.32

Using OLS method, the following estimation of matrices Φ_1 and Φ_0 was obtained including all parameters estimates:

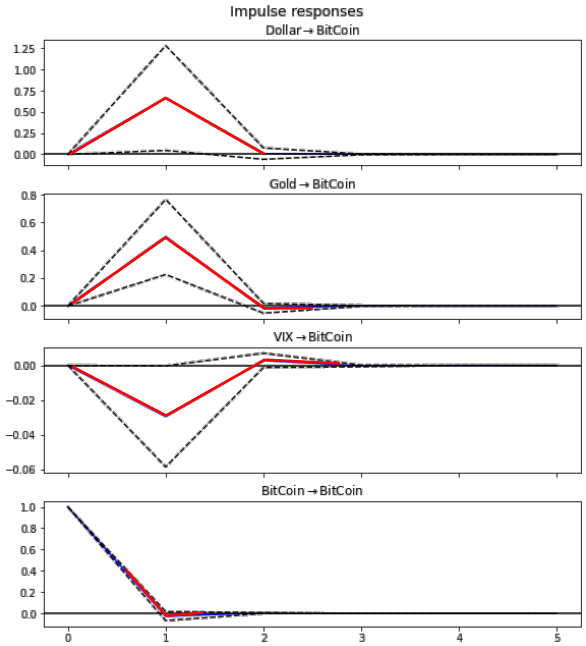
$$R_t = \begin{bmatrix} r_{Dollar,t} & r_{Gold,t} & r_{VIX,t} & r_{Bitcoin,t} \end{bmatrix}'$$

$$\hat{\Phi}_1 = \begin{bmatrix} \hat{\Phi}_{11}^{(1)} & \hat{\Phi}_{12}^{(1)} & \hat{\Phi}_{13}^{(1)} & \hat{\Phi}_{14}^{(1)} \\ \hat{\Phi}_{21}^{(1)} & \hat{\Phi}_{22}^{(1)} & \hat{\Phi}_{23}^{(1)} & \hat{\Phi}_{24}^{(1)} \\ \hat{\Phi}_{31}^{(1)} & \hat{\Phi}_{32}^{(1)} & \hat{\Phi}_{33}^{(1)} & \hat{\Phi}_{34}^{(1)} \\ \hat{\Phi}_{41}^{(1)} & \hat{\Phi}_{42}^{(1)} & \hat{\Phi}_{43}^{(1)} & \hat{\Phi}_{44}^{(1)} \end{bmatrix} = \begin{bmatrix} -0.024 & -0.021 & 0.001127 & -0.000406 \\ 0.0657 & 0.0169 & -0.00189 & 0.00592 \\ -0.335 & -0.0869 & -0.0788 & -0.0118 \\ 0.6647 & 0.496 & -0.029 & -0.0261 \end{bmatrix} \quad ; \quad \hat{\Phi}_0 = \begin{bmatrix} 0.000063 \\ -0.0000023 \\ 0.000041 \\ 0.003872 \end{bmatrix}$$

From our estimations, we can especially see that all variables influence negatively the VIX level since all coefficients are negative. The Dollar index seems to have a strong influence on the level of the Bitcoin with a coefficient estimate of 0.6647. Furthermore, Gold has also an important positive impact on the Bitcoin.

Furthermore, we analyzed the impact of one standard deviation unit shock of the Dollar, the Gold and the VIX on the of the Bitcoin by performing an Impulse Reaction function analysis. A confidence interval of 95% was used. We can clearly see that these shocks are temporary and that it takes 3 unit of times to go back to the initial state. It is interesting to note that a positive shock on Dollar and Gold leads to a positive response on the bitcoin.

For the VIX, we can see in the charts of the evolution of the VIX index and evolution BTC/USD prices, if we focus on mars 2020, the beginning of the pandemic, we can see a high VIX index as the uncertainty was very high on the market. On the other hand, the value of the BTC was relatively low on mars 2020 compared to now. But as the uncertainty decrease on the market (fall of the VIX index), we see a raise on the BTC value. If we link that with the graph where we see a unit impulse of the VIX, and a negative response on the BTC. We can infer that the BTC is a "risky" asset, that risk averse people tend to avoid during a relatively high level of uncertainty on the market.

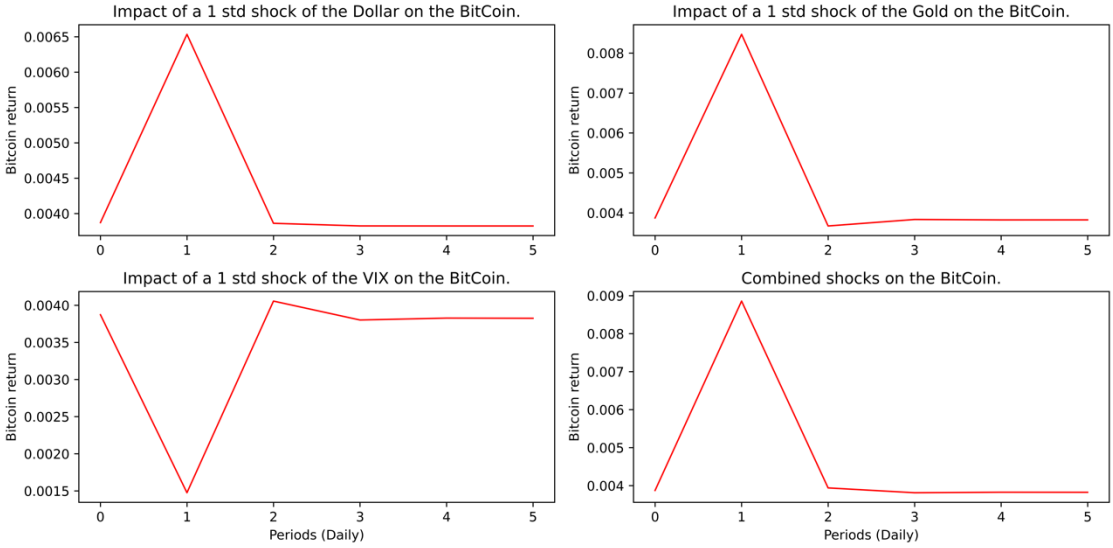


We simulated the following errors ε_0, and we performed an iterative process to look at the Bitcoin return equation:

$$\varepsilon_0 = \begin{bmatrix} \sigma_{Dollar} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad ; \quad \varepsilon_0 = \begin{bmatrix} 0 \\ \sigma_{Gold} \\ 0 \\ 0 \end{bmatrix} \quad ; \quad \varepsilon_0 = \begin{bmatrix} 0 \\ 0 \\ \sigma_{VIX} \\ 0 \end{bmatrix} \quad ; \quad \varepsilon_0 = \begin{bmatrix} \sigma_{Dollar} \\ \sigma_{Gold} \\ \sigma_{VIX} \\ 0 \end{bmatrix}$$

$$R_0 = \Phi_0 + \varepsilon_0 \Longrightarrow R_1 = \Phi_0 + \Phi_1 R_0 \Longrightarrow R_2 = \Phi_0 + \Phi_1 R_1 \Longrightarrow \cdots \Longrightarrow R_5 = \Phi_0 + \Phi_1 R_4$$

As we can see from our results, these shocks go in the same direction as the unit impulse previously. As the different standard deviations are smaller than 1, this leads to a smaller response. The shocks are also temporary, after 3 periods they go back to their initial level (which is the constant term in the bitcoin equation).



Finally, we also considered the case where the Bitcoin would not influence dynamically the other variables. We called this model the unconstrained one. A maximum likelihood minimization has been made to estimate the new parameters of this model. We added constraints in the optimization set such as the coefficients $\widehat{\Phi}_{14}^{(1)} = \widehat{\Phi}_{24}^{(1)} = \widehat{\Phi}_{34}^{(1)} = 0$. We then obtain the following estimates from the optimization problem:

$$\widehat{\Phi}_1^{Const} = \begin{bmatrix} -0.021 & -0.0177 & -0.0106 & 0 \\ 0.042 & 0.0102 & -0.0077 & 0 \\ -0.0015 & -0.00276 & -0.0783 & 0 \\ 0.00152 & 0.00775 & -0.0125 & -0.0203 \end{bmatrix} \quad ; \quad \widehat{\Phi}_0^{Const} = \begin{bmatrix} -0.0000523 \\ 0.00000834 \\ -0.0000223 \\ -0.0039 \end{bmatrix}$$

$$\widehat{\Phi}_1^{Unc} = \begin{bmatrix} -0.024 & 0.0659 & -0.335 & 0.665 \\ -0.0214 & 0.01679 & -0.0867 & 0.49637 \\ 0.00112 & -0.00191 & -0.0788 & -0.0294 \\ -0.00041 & 0.00592 & -0.01186 & -0.0261 \end{bmatrix} \quad ; \quad \widehat{\Phi}_0^{Unc} = \begin{bmatrix} -0.0000628 \\ 0.00000235 \\ -0.00004104 \\ -0.00387 \end{bmatrix}$$

To compare both models, it is necessary to compute the Likelihood Ratio Test with the null hypothesis H_0 that the real model is the constrained one. The Test statistic, with the likelihood of the unconstrained model U and of the constrained one R, is given by (with the number of parameters to estimate K):

$$LR = 2(\log \mathcal{L}_U - \log \mathcal{L}_R) > 0 \xrightarrow{d} \chi^2(J) \quad ; \quad J = K_U - K_R = 20 - 17 = 3$$

In our case, $J = 3$ because we made three restrictions on the constrained model. Finally, we obtain a LR test equal to 0.0567 and a p-value of 0.9965. We reject H_0 if we have $LR \geq \chi^2_{1-\alpha}(3)$.

	Threshold at 1%	*Threshold at 5%*	*Threshold at 10%*
Critical Values $\chi^2_{1-\alpha}(3)$	11.345	7.815	6.251

Our test statistic is for each threshold lower than the critical value meaning that we do not reject the null hypothesis. In consequence, the true model is the constrained one and it shows that the Bitcoin do not have an important impact and dynamical influence on the other variables.

VII. Conclusion:

Through this report, we analyzed the econometric properties of different cryptocurrencies and we studied their common characteristics. Using this information, we were able to build a profitable trading strategy founded on cointegration. We found that, even if theoretically our strategy does not provide at 100% profits, these strategies can be very profitable in some cases. Finally, in a second part we build a Vector Autoregressive Model between the Bitcoin and other variables to study the relationship between them. We saw at the end of this analysis that the Bitcoin does have a negligeable impact on variables such as the Dollar index, the gold price or the VIX level because, as we saw previously, it is more correct to use the restricted VAR model. It would be interesting to go further in our analysis by having a look on the relationships between the Bitcoin and other variables and assets such as stock indices, inflation level, etc.

VIII. Appendix:

The code used for this assignment is given right below:

- *Part 1:*

```
@author: The Quant Managers : Alessandro Loury, Endrit Kastrati, Sinan Guven & Grégory Porchet
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy.stats import skew
from scipy.stats import kurtosis
from scipy.stats import norm
import seaborn as sns
from statsmodels.tsa.ar_model import AutoReg
import math



#=================================================
#Downloading the data
#=================================================



Bitcoin = pd.read_csv("Part1_BTCUSD_d.csv")
Bitcoin = Bitcoin.iloc[:1613, [1, 6]]
Bitcoin = Bitcoin.rename(columns = {"date" : "Date", "close" : "BTC/USD"})
Bitcoin['Date'] = pd.to_datetime(Bitcoin['Date']).dt.date
Bitcoin = Bitcoin.set_index('Date')


Ethereum = pd.read_csv("Part1_ETHUSD_d.csv")
Ethereum = Ethereum.iloc[:1613, [1, 6]]
Ethereum = Ethereum.rename(columns = {"date" : "Date", "close" : "ETH/USD"})
Ethereum['Date'] = pd.to_datetime(Ethereum['Date']).dt.date
Ethereum = Ethereum.set_index('Date')


Monera = pd.read_csv("Part1_XMRUSD_d.csv")
Monera = Monera.iloc[1:, [1, 6]]
Monera = Monera.rename(columns = {"date" : "Date", "close" : "XMR/USD"})
Date_XMR = Monera.iloc[:, 0]
Monera['Date'] = pd.to_datetime(Monera['Date']).dt.date
Monera = Monera.set_index('Date')



#Unique Database for all cyptos



Cryptos = pd.concat([Bitcoin, Ethereum, Monera], axis = 1)
Cryptos = Cryptos.dropna()
Cryptos = Cryptos.iloc[::-1]

Prices = Cryptos

Log_Prices = np.log(Prices)
```

```
#Graphical Representation



plt.figure(dpi = 1000, figsize=(16, 8))
plt.subplot(231)
plt.plot(Cryptos.iloc[:, 0], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Bitcoin/USD')
plt.xticks(rotation = 40)

plt.subplot(232)
plt.plot(Cryptos.iloc[:, 1], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Ethereum/USD Price')
plt.xticks(rotation = 40)

plt.subplot(233)
plt.plot(Cryptos.iloc[:, 2], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Monera/USD Price')
plt.xticks(rotation = 40)

plt.subplot(234)
plt.plot(Log_Prices.iloc[:, 0], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Bitcoin/USD Log-Price')
plt.xticks(rotation = 40)

plt.subplot(235)
plt.plot(Log_Prices.iloc[:, 1], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Ethereum/USD Log-Price')
plt.xticks(rotation = 40)

plt.subplot(236)
plt.plot(Log_Prices.iloc[:, 2], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Monera/USD Log-Price ')
plt.xticks(rotation = 40)
plt.tight_layout()
plt.show()



#Computation of Simple Returns



Returns_Simple = np.divide(Prices.iloc[1:1613, :], Prices.iloc[0:(1613 - 1), :]) - 1

plt.figure(dpi = 1000, figsize=(16, 5))
plt.subplot(131)
plt.plot(Returns_Simple.iloc[:, 0], color = 'red', linewidth = 1)
plt.ylabel('Returns')
plt.xlabel('Periods (Daily)')
plt.title('Bitcoin/USD Returns')
plt.xticks(rotation = 40)

plt.subplot(132)
plt.plot(Returns_Simple.iloc[:, 1], color = 'red', linewidth = 1)
plt.ylabel('Returns')
plt.xlabel('Periods (Daily)')
plt.title('Ethereum/USD Returns')
plt.xticks(rotation = 40)

plt.subplot(133)
plt.plot(Returns_Simple.iloc[:, 2], color = 'red', linewidth = 1)
plt.ylabel('Returns')
plt.xlabel('Periods (Daily)')
plt.title('Monera/USD Returns')
plt.xticks(rotation = 40)
plt.tight_layout()
plt.show()



#Correlation Matrix
```

```
plt.figure(dpi = 1000)
sns.heatmap(Prices.corr(), annot=True)
plt.title('Correlation Matrix Prices', fontdict={'fontsize':12}, pad=12)
plt.show()

plt.figure(dpi = 1000)
sns.heatmap(Log_Prices.corr(), annot=True)
plt.title('Correlation Matrix Log-Prices', fontdict={'fontsize':12}, pad=12)
plt.show()



#Characteristics for Simple Prices



Annualized_Mean = np.mean(Prices, 0)

Sigma = np.cov(np.transpose(Prices))

Volatility = np.power(np.diag(Sigma), 0.5)

Skewness = skew(Prices)

Kurtosis = kurtosis(Prices, fisher = False)

Maximum = Prices.max()

Minimum = Prices.min()



#Characteristics for Log Prices



Annualized_Mean2 = np.mean(Log_Prices, 0)

Sigma2 = np.cov(np.transpose(Log_Prices))

Volatility2 = np.power(np.diag(Sigma2), 0.5)

Skewness2 = skew(Log_Prices)

Kurtosis2 = kurtosis(Log_Prices, fisher = False)

Maximum2 = Log_Prices.max()

Minimum2 = Log_Prices.min()



#Density functions for all time series



plt.figure(dpi = 1000, figsize=(16, 8))
plt.subplot(231)
sns.distplot(Prices.iloc[:, 0], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Prices.iloc[:, 0]) - 3*np.std(Prices.iloc[:, 0]), np.mean(Prices.iloc[:, 0]) +
3*np.std(Prices.iloc[:, 0]), 100)
plt.plot(x, norm.pdf(x, np.mean(Prices.iloc[:, 0]), np.std(Prices.iloc[:, 0])), color = 'black')
plt.legend(['PDF Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Bitcoin')
plt.xlabel('Bitcoin Simple Prices')

plt.subplot(232)
sns.distplot(Prices.iloc[:, 1], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Prices.iloc[:, 1]) - 3*np.std(Prices.iloc[:, 1]), np.mean(Prices.iloc[:, 1]) +
3*np.std(Prices.iloc[:, 1]), 100)
plt.plot(x, norm.pdf(x, np.mean(Prices.iloc[:, 1]), np.std(Prices.iloc[:, 1])), color = 'black')
plt.legend(['PDF Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Ethereum')
plt.xlabel('Ethereum Simple Prices')

plt.subplot(233)
sns.distplot(Prices.iloc[:, 2], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Prices.iloc[:, 2]) - 3*np.std(Prices.iloc[:, 2]), np.mean(Prices.iloc[:, 2]) +
3*np.std(Prices.iloc[:, 2]), 100)
plt.plot(x, norm.pdf(x, np.mean(Prices.iloc[:, 2]), np.std(Prices.iloc[:, 2])), color = 'black')
plt.legend(['PDF Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Monera')
plt.xlabel('Monera Simple Prices')

plt.subplot(234)
sns.distplot(Log_Prices.iloc[:, 0], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Log_Prices.iloc[:, 0]) - 3*np.std(Log_Prices.iloc[:, 0]), np.mean(Log_Prices.iloc[:, 0]) +
3*np.std(Log_Prices.iloc[:, 0]), 100)
plt.plot(x, norm.pdf(x, np.mean(Log_Prices.iloc[:, 0]), np.std(Log_Prices.iloc[:, 0])), color = 'black')
```

```
plt.legend(['PDF Log Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Bitcoin')
plt.xlabel('Bitcoin Log Prices')

plt.subplot(235)
sns.distplot(Log_Prices.iloc[:, 1], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Log_Prices.iloc[:, 1]) - 3*np.std(Log_Prices.iloc[:, 1]), np.mean(Log_Prices.iloc[:, 1]) +
3*np.std(Log_Prices.iloc[:, 1]), 100)
plt.plot(x, norm.pdf(x, np.mean(Log_Prices.iloc[:, 1]), np.std(Log_Prices.iloc[:, 1])), color = 'black')
plt.legend(['PDF Log Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Ethereum')
plt.xlabel('Ethereum Log Prices')

plt.subplot(236)
sns.distplot(Log_Prices.iloc[:, 2], hist=False, kde=True, bins=int(180/5), color = 'red', kde_kws={'linewidth': 1})
x = np.linspace(np.mean(Log_Prices.iloc[:, 2]) - 3*np.std(Log_Prices.iloc[:, 2]), np.mean(Log_Prices.iloc[:, 2]) +
3*np.std(Log_Prices.iloc[:, 2]), 100)
plt.plot(x, norm.pdf(x, np.mean(Log_Prices.iloc[:, 2]), np.std(Log_Prices.iloc[:, 2])), color = 'black')
plt.legend(['PDF Log Prices', 'PDF Normal Distribution'])
plt.title('Density Functions Comparison for Monera')
plt.xlabel('Monera Log Prices')
plt.tight_layout()



#=====================================================
#Part 1.1.1
#=====================================================



#Percentiles = [10, 5, 1, 0.1]
Parameters = []
Std_Errors = []


p = np.zeros((1613, 2))

for i in range(10000):
    Epsilon = np.random.normal(0, 1, 1613)
    p[:, 0] = Epsilon

    for j in range(1613):
        if j == 0:
            p[j, 1] = 0
        else:
            p[j, 1] = p[j-1, 1] + p[j, 0]

    AR_Process = AutoReg(p[:, 1], 1, old_names=False)
    results = AR_Process.fit()
    Parameters.append(results._params[1])
    Std_Errors.append(results.bse[1])

Parameters = np.array(Parameters)
Std_Errors = np.array(Std_Errors)
One = np.ones(10000)
DF_Test = np.divide(np.subtract(Parameters, One), Std_Errors)


def ecdf(a):
    x, counts = np.unique(a, return_counts=True)
    cusum = np.cumsum(counts)
    return x, cusum / cusum[-1]

Critical_Values = ecdf(DF_Test)
s1 = Critical_Values[0][99]
s2 = Critical_Values[0][499]
s3 = Critical_Values[0][999]
print(s1)
print(s2)
print(s3)



#For the second model of the Random Walk



Parameters2 = []
Std_Errors2 = []
p2 = np.zeros((1613, 2))

for i in range(10000):
    Epsilon2 = np.random.normal(0, 1, 1613)
    p2[:, 0] = Epsilon2

    for j in range(1613):
        if j == 0:
            p2[j, 1] = 0
```

```
        else:
            p2[j, 1] = 0.2 * p2[j-1, 1] + p2[j, 0]

    AR_Process2 = AutoReg(p2[:, 1], 1, old_names=False)
    results2 = AR_Process2.fit()
    Parameters2.append(results2._params[1])
    Std_Errors2.append(results2.bse[1])

Parameters2 = np.array(Parameters2)
Std_Errors2 = np.array(Std_Errors2)
One2 = 0.2 * np.ones(10000)
DF_Test2 = np.divide(np.subtract(Parameters2, One2), Std_Errors2)

Critical_Values2 = ecdf(DF_Test2)
S1 = Critical_Values2[0][99]
S2 = Critical_Values2[0][499]
S3 = Critical_Values2[0][999]
print(S1)
print(S2)
print(S3)


#Graphical representation of the t-stats distribution
plt.figure(dpi = 1000)
plt.hist(DF_Test, bins = "auto", rwidth=2, histtype= 'bar', density = True, color = 'white', stacked=True, edgecolor
= 'red')
x = np.linspace(np.mean(DF_Test) - 3*np.std(DF_Test), np.mean(DF_Test) + 3*np.std(DF_Test), 100)
plt.plot(x, norm.pdf(x, np.mean(DF_Test), np.std(DF_Test)), color = 'darkred', linewidth = 1.5)
plt.legend(['Normal PDF'])
plt.title('Histogram first model')
plt.xlabel('Dickey-Fuller t-stats')
plt.ylabel('Frequency')
plt.axvline(s1, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.99 * s1, 0.45,'1%', rotation = 90, color = 'darkred')
plt.axvline(s2, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.99 * s2, 0.45, '5%', rotation = 90, color = 'darkred')
plt.axvline(s3, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.99 * s3, 0.45, '10%', rotation = 90, color = 'darkred')
plt.show()

plt.figure(dpi = 1000)
plt.hist(DF_Test2, bins = "auto", rwidth=2, histtype= 'bar', density = True, color = 'white', stacked=True,
edgecolor = 'red')
x = np.linspace(np.mean(DF_Test2) - 3*np.std(DF_Test2), np.mean(DF_Test2) + 3*np.std(DF_Test2), 100)
plt.plot(x, norm.pdf(x, np.mean(DF_Test2), np.std(DF_Test2)), color = 'darkred', linewidth = 1.5)
plt.legend(['Normal PDF'])
plt.title('Histogram second model')
plt.xlabel('Dickey-Fuller t-stats')
plt.ylabel('Frequency')
plt.axvline(S1, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.97 * S1, 0.35, '1%', rotation = 90, color = 'darkred')
plt.axvline(S2, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.97 * S2, 0.35, '5%', rotation = 90, color = 'darkred')
plt.axvline(S3, linestyle = '--', linewidth = 1, color = 'darkred')
plt.text(0.97 * S3, 0.35, '10%', rotation = 90, color = 'darkred')
plt.show()



#============================================================
#Part 1.1.2
#============================================================



AR_BTC = AutoReg(Log_Prices.iloc[:, 0], 1) #For Bitcoin
Results_BTC = AR_BTC.fit()
Param_BTC = Results_BTC.params
Std_Error_BTC = Results_BTC.bse
t_Stat_BTC = np.divide(np.subtract(Param_BTC[1], 1), Std_Error_BTC[1])

AR_ETH = AutoReg(Log_Prices.iloc[:, 1], 1) #For ETH
Results_ETH = AR_ETH.fit()
Param_ETH = Results_ETH.params
Std_Error_ETH = Results_ETH.bse
t_Stat_ETH = np.divide(np.subtract(Param_ETH[1], 1), Std_Error_ETH[1])

AR_XMR = AutoReg(Log_Prices.iloc[:, 2], 1) #For XMR
Results_XMR = AR_XMR.fit()
Param_XMR = Results_XMR.params
Std_Error_XMR = Results_XMR.bse
t_Stat_XMR = np.divide(np.subtract(Param_XMR[1], 1), Std_Error_XMR[1])


X_BTC = sm.add_constant(Log_Prices.iloc[:(len(Log_Prices) - 1), 0])
X_ETH = sm.add_constant(Log_Prices.iloc[:(len(Log_Prices) - 1), 1])
X_XMR = sm.add_constant(Log_Prices.iloc[:(len(Log_Prices) - 1), 2])

plt.figure(dpi = 1000, figsize=(16, 5))
```

```
plt.subplot(131)
plt.scatter(Log_Prices.iloc[:(len(Log_Prices) - 1), 0], Log_Prices.iloc[1:, 0], s = 1, color = 'red')
plt.plot(X_BTC, Param_BTC[0] + Param_BTC[1] * X_BTC, color = 'blue', linewidth = 1)
plt.axis([6.5, 11.5, 6.5, 11.5])
plt.xlabel('Log-Prices at t-1')
plt.ylabel('Log-Prices at t')
plt.title('Regression for Bitcoin')
plt.legend(["Regression"])

plt.subplot(132)
plt.scatter(Log_Prices.iloc[:(len(Log_Prices) - 1), 1], Log_Prices.iloc[1:, 1], s = 1, color = 'red')
plt.plot(X_ETH, Param_ETH[0] + Param_ETH[1] * X_ETH, color = 'blue', linewidth = 1)
plt.axis([3.5, 8.5, 3.5, 8.5])
plt.xlabel('Log-Prices at t-1')
plt.ylabel('Log-Prices at t')
plt.title('Regression for Ethereum')
plt.legend(["Regression"])

plt.subplot(133)
plt.scatter(Log_Prices.iloc[:(len(Log_Prices) - 1), 2], Log_Prices.iloc[1:, 2], s = 1, color = 'red')
plt.plot(X_XMR, Param_XMR[0] + Param_XMR[1] * X_XMR, color = 'blue', linewidth = 1)
plt.axis([2.5, 6.5, 2.5, 6.5])
plt.xlabel('Log-Prices at t-1')
plt.ylabel('Log-Prices at t')
plt.title('Regression for Monera')
plt.legend(["Regression"])
plt.tight_layout()



#=========================================
#Part 1.2
#=========================================



#Creation of a function that show if a function is stationary or not
def Statio(x):
    diff_x = np.subtract(x[1:], x[0:(len(x) - 1)])
    X0 = sm.add_constant(x[:(len(x) - 1)])
    Regression_Resid = sm.OLS(diff_x, X0)
    Results_Resid = Regression_Resid.fit()
    Param_Resid = Results_Resid.params
    Std_Error_Resid = Results_Resid.bse
    t_Stat_Resid = np.divide(np.subtract(Param_Resid[1], 0), Std_Error_Resid[1])
    plt.plot(diff_x)
    plt.show()
    print(t_Stat_Resid)
    print(Param_Resid)


#Regression BTC-ETH
X1 = sm.add_constant(Log_Prices.iloc[:, 1])
BTC_ETH_Reg = sm.OLS(Log_Prices.iloc[:, 0], X1)
Results11 = BTC_ETH_Reg.fit()
Para_BTC_ETH = Results11.params
Z_1_Resid = Results11.resid
Z_1_Resid = np.array(Z_1_Resid)


#Regression XMR-BTC
X2 = sm.add_constant(Log_Prices.iloc[:, 0])
XMR_BTC_Reg = sm.OLS(Log_Prices.iloc[:, 2], X2)
Results12 = XMR_BTC_Reg.fit()
Para_XMR_BTC = Results12.params
Z_2_Resid = Results12.resid
Z_2_Resid = np.array(Z_2_Resid)


#Regression XMR-ETH
X3 = sm.add_constant(Log_Prices.iloc[:, 1])
XMR_ETH_Reg = sm.OLS(Log_Prices.iloc[:, 2], X3)
Results13 = XMR_ETH_Reg.fit()
Para_XMR_ETH = Results13.params
Z_3_Resid = Results13.resid
Z_3_Resid = np.array(Z_3_Resid)
plt.plot(Z_3_Resid)


#Graphical Representation
Log_Prices_Array = np.array(Log_Prices)
X1 = np.array(X1)
X2 = np.array(X2)
X3 = np.array(X3)


plt.figure(dpi = 1000, figsize=(16, 5))
plt.subplot(131)
plt.scatter(X1[:, 1], Log_Prices_Array[:, 0], s = 1, color = 'red')
```

```
plt.plot(X1, Para_BTC_ETH[0] + Para_BTC_ETH[1] * X1, color = 'blue', linewidth = 1)
plt.axis([1.5, 8.5, 6, 11.5])
plt.xlabel('Ethereum Log-Prices')
plt.ylabel('Bitcoin Log-Prices')
plt.title('Regression of Bitcoin on Ethereum')
plt.legend(["Regression"])

plt.subplot(132)
plt.scatter(X2[:, 1], Log_Prices_Array[:, 2], s = 1, color = 'red')
plt.plot(X2, Para_XMR_BTC[0] + Para_XMR_BTC[1] * X2, color = 'blue', linewidth = 1)
plt.axis([6, 11.5, 1.5, 6.5])
plt.xlabel('Bitcoin Log-Prices')
plt.ylabel('Monera Log-Prices')
plt.title('Regression of Monera on Bitcoin')
plt.legend(["Regression"])

plt.subplot(133)
plt.scatter(X3[:, 1], Log_Prices_Array[:, 2], s = 1, color = 'red')
plt.plot(X3, Para_XMR_ETH[0] + Para_XMR_ETH[1] * X3, color = 'blue', linewidth = 1)
plt.axis([1.5, 8.5, 1.5, 6.5])
plt.xlabel('Ethereum Log-Prices')
plt.ylabel('Monera Log-Prices')
plt.title('Regression of Monera on Ethereum')
plt.legend(["Regression"])
plt.tight_layout()


#Check for stationarity
Statio(Z_1_Resid)
Statio(Z_2_Resid)
Statio(Z_3_Resid)

diff_Z_1 = np.subtract(Z_1_Resid[1:], Z_1_Resid[0:(len(Z_1_Resid) - 1)])
plt.plot(Z_3_Resid)


#Check cointegration with visualization
plt.figure(dpi = 1000)
plt.plot(Log_Prices.iloc[:, 1], color = 'red', linewidth = 1)
plt.plot(Log_Prices.iloc[:, 2], color = 'darkred', linewidth = 1)
plt.plot(Log_Prices.iloc[:, 0], color = 'red', linewidth = 1)
plt.xlabel('Periods (Daily)')
plt.ylabel('Log-Prices')
plt.title('Log-Prices of Monera and Ethereum')
plt.legend(["Ethereum", "Monera"])
plt.xticks(rotation = 40)
plt.show()



#================================================
#Pair trading
#================================================



#Computation of the returns
Log_Returns_XMR =  np.subtract(Log_Prices_Array[1:1613, 2], Log_Prices_Array[0:(1613 - 1), 2])
Log_Returns_ETH =  np.subtract(Log_Prices_Array[1:1613, 1], Log_Prices_Array[0:(1613 - 1), 1])


#Computation of the Spread
Spread = np.subtract(Log_Returns_XMR, Log_Returns_ETH)
Spread_Bar = np.mean(Spread)
Volat_Spread = np.mean(np.power(Spread, 2)) - np.power(Spread_Bar, 2)

#Computation of the Z-score
z_score = np.divide(np.subtract(Spread, Spread_Bar), Volat_Spread)

z_upper = np.quantile(z_score, 0.95)
z_lower = np.quantile(z_score, 0.05)
z_out = np.quantile(z_score, 0.40)

#Graphical Representation of the signals
Open1 = []
Open2 = []
for i in range(len(z_score)):
    if z_score[i] > z_upper:
        Open1.append([z_score[i], Log_Prices.index[i+1]])
    elif z_score[i] < z_lower:
        Open2.append([z_score[i], Log_Prices.index[i+1]])

Open1 = np.array(Open1)
Open2 = np.array(Open2)

plt.figure(dpi = 1000)
plt.plot(Log_Prices.index[1:], z_score, color = 'darkred', linewidth = 0.5)
plt.axhline(z_lower, color = 'red', linestyle = '--', linewidth = 0.8)
plt.axhline(z_upper, color = 'blue', linestyle = '--', linewidth = 0.8)
```

```
plt.plot(Open1[:, 1], Open1[:, 0], color = 'green', linestyle = 'None', marker = 'o', markersize = 2.5)
plt.plot(Open2[:, 1], Open2[:, 0], color = 'red', linestyle = 'None', marker = 'o', markersize = 2.5)
plt.title('Evolution of the z-scores')
plt.xlabel('Periods (Daily)')
plt.ylabel('Value of the z-scores')
plt.legend(['z-scores', 'z-lower', 'z-upper'])
plt.xticks(rotation = 40)

z_score_higher = z_score[z_score > z_upper]
len(z_score_higher)
z_score_lower = z_score[z_score < z_lower]
len(z_score_lower)

#Start of the Wealth
Wealth = np.zeros((1612))

#Array of Portfolio Returns
Portfolio_Returns = np.zeros((1612))

#Matrix of weight
Weights = np.zeros((1612, 2))

#Computation of the algorithm
LongA = []
LongB = []

for t in range(len(z_score)): #Different situations:

    #If nothing happens
    if len(LongA) == 0 and z_lower <  z_score[t] < z_upper and len(LongB) == 0:

        if t == 0:
            Wealth[0] = 100 #Initialize the first wealth at 100
        else:
            Wealth[t] = Wealth[t - 1] #If nothing happens at t, the wealth is the same as yesterday

    #Signal to open: Long B and Short A
    elif z_score[t] > z_upper and len(LongB) == 0 and len(LongA) == 0:

        Weights[t, 0] = 0.5
        Weights[t, 1] = 0.5

        Wealth[t] = Wealth[t - 1]
        LongB.append(1)

    #Nothing happens after opening our position but we adjust our wealth with the price evolution
    elif z_score[t] > z_out and len(LongB) > 0 and len(LongA) == 0:

        Weights[t, 0] = (Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t])) / (Weights[t - 1, 0] * math.exp((-
1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t]))
        Weights[t, 1] = (Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t])) / (Weights[t - 1, 0] * math.exp((-1) *
Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t]))

        Wealth[t] = Wealth[t - 1] * (Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp(Log_Returns_ETH[t]))
        Portfolio_Returns[t] = np.log((Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp(Log_Returns_ETH[t])))

    #If we close the positions of Long B and Short A
    elif z_score[t] <= z_out and len(LongB) > 0 and len(LongA) == 0:

        Weights[t, 0] = (Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t])) / (Weights[t - 1, 0] * math.exp((-
1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t]))
        Weights[t, 1] = (Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t])) / (Weights[t - 1, 0] * math.exp((-1) *
Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp(Log_Returns_ETH[t]))

        Wealth[t] = Wealth[t - 1] * ((Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp(Log_Returns_ETH[t])))
        Portfolio_Returns[t] = np.log((Weights[t - 1, 0] * math.exp((-1) * Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp(Log_Returns_ETH[t])))
        LongB = []

    #Signal to open: Long A and Short B
    elif z_score[t] < z_lower and len(LongA) == 0 and len(LongB) == 0:

        Weights[t, 0] = 0.5
        Weights[t, 1] = 0.5

        Wealth[t] = Wealth[t - 1]
        LongA.append(1)

    #Nothing happens after opening but we adjust our wealth with the price evolution
    elif z_score[t] < (- 1) * z_out and len(LongA) > 0 and len(LongB) == 0:

        Weights[t, 0] = (Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t])) / (Weights[t - 1, 0] *
math.exp(Log_Returns_XMR[t]) +  Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t]))
        Weights[t, 1] = (Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t])) / (Weights[t - 1, 0] *
math.exp(Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t]))
```

```
        Wealth[t] = Wealth[t - 1] * (Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp((-1) * Log_Returns_ETH[t]))
        Portfolio_Returns[t] = np.log(Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t]) +  Weights[t - 1, 1] *
math.exp((-1) * Log_Returns_ETH[t]))

    #If we close the positions of Long A and Short B
    elif z_score[t] >= (- 1) * z_out and len(LongA) > 0 and len(LongB) == 0:

        Weights[t, 0] = (Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t])) / (Weights[t - 1, 0] *
math.exp(Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t]))
        Weights[t, 1] = (Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t])) / (Weights[t - 1, 0] *
math.exp(Log_Returns_XMR[t]) + Weights[t - 1, 1] * math.exp((-1) * Log_Returns_ETH[t]))

        Wealth[t] = Wealth[t - 1] * (Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t]) + Weights[t - 1, 1] *
math.exp((-1) * Log_Returns_ETH[t]))
        Portfolio_Returns[t] = np.log(Weights[t - 1, 0] * math.exp(Log_Returns_XMR[t]) +  Weights[t - 1, 1] *
math.exp((-1) * Log_Returns_ETH[t]))
        LongA = []

plt.figure(dpi = 1000)
plt.plot(Log_Prices.index[1:], Wealth, linewidth = 1, color = 'red')
plt.xticks(rotation = 40)
plt.title('Evolution of the Wealth Performance')
plt.xlabel('Periods (Daily)')
plt.ylabel('Value of our Wealth')
plt.show()

plt.figure(dpi = 1000, figsize=(16, 4))
plt.subplot(121)
sns.scatterplot(Log_Prices.index[1:], Weights[:, 0], s = 5, color = 'red')
plt.xticks(rotation = 40)
plt.title('Evolution of the Weights for Monera')
plt.xlabel('Periods (Daily)')
plt.ylabel('Weights')

plt.subplot(122)
sns.scatterplot(Log_Prices.index[1:], Weights[:, 1], s = 5, color = 'red')
plt.xticks(rotation = 40)
plt.title('Evolution of the Weights for Ethereum')
plt.xlabel('Periods (Daily)')
plt.ylabel('Weights')
plt.show()

plt.figure(dpi = 1000)
plt.plot(Log_Prices.index[1:], Portfolio_Returns, linewidth = 1, color = 'red')
plt.xticks(rotation = 40)
plt.title('Portfolio Log-Returns')
plt.xlabel('Periods (Daily)')
plt.ylabel('Log-Returns')
plt.show()

Annualized_Mean12 = np.mean(Portfolio_Returns, 0) * 252

Volatility12 = np.std(Portfolio_Returns) * np.power(252, 0.5)

Skewness12 = skew(Portfolio_Returns)

Kurtosis12 = kurtosis(Portfolio_Returns, fisher = False)

Maximum12 = Portfolio_Returns.max()

Minimum12 = Portfolio_Returns.min()


Annualized_Mean22 = np.mean(Wealth, 0)

Volatility22 = np.std(Wealth)

Skewness22 = skew(Wealth)

Kurtosis22 = kurtosis(Wealth, fisher = False)

Maximum22 = Wealth.max()

Minimum22 = Wealth.min()
```

- *Part 2:*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.api import VAR
import numpy.matlib
from scipy.stats import multivariate_normal
from scipy.stats.distributions import chi2
from statsmodels.tsa.stattools import kpss
import seaborn as sns

#============================================================
#Download of the data
#============================================================



filename = "Part2_dataset.xlsx"

xls = pd.ExcelFile(filename)

Data = pd.read_excel(xls, 'Sheet1')

Data['Date'] = pd.to_datetime(Data['Unnamed: 0']).dt.date
Data = Data.set_index('Date')



#============================================================
#Database of each price time series
#============================================================



Price = Data.iloc[:, 1:5]

Log_Price = np.log(Price)



#============================================================
#Graphical Representation
#============================================================



plt.figure(dpi = 1000, figsize=(16, 4))
plt.subplot(121)
plt.plot(Price.iloc[:, 0], color = 'red', linewidth = 0.9)
plt.ylabel('Dollar Index')
plt.xlabel('Periods (Daily)')
plt.title('Price of US Dollar in trade weighted terms')

plt.subplot(122)
plt.plot(Log_Price.iloc[:, 0], color = 'red', linewidth = 0.9)
plt.ylabel('Dollar Index')
plt.xlabel('Periods (Daily)')
plt.title('Log-Price of US Dollar in trade weighted terms')
plt.show()

plt.figure(dpi = 1000, figsize=(16, 4))
plt.subplot(121)
plt.plot(Price.iloc[:, 1], color = 'red', linewidth = 0.9)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Gold Price')

plt.subplot(122)
plt.plot(Log_Price.iloc[:, 1], color = 'red', linewidth = 0.9)
plt.ylabel('Log-Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('Gold Log-Price')
plt.show()

plt.figure(dpi = 1000, figsize=(16, 4))
plt.subplot(121)
plt.plot(Price.iloc[:, 2], color = 'red', linewidth = 0.9)
plt.ylabel('Index')
plt.xlabel('Periods (Daily)')
plt.title('VIX Index')

plt.subplot(122)
plt.plot(Log_Price.iloc[:, 2], color = 'red', linewidth = 0.9)
plt.ylabel('Index')
plt.xlabel('Periods (Daily)')
```

```
plt.title('VIX Index (in log)')
plt.show()

plt.figure(dpi = 1000, figsize=(16, 4))
plt.subplot(121)
plt.plot(Price.iloc[:, 3], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('BTC/USD Price')

plt.subplot(122)
plt.plot(Log_Price.iloc[:, 3], color = 'red', linewidth = 1)
plt.ylabel('Price in Dollars')
plt.xlabel('Periods (Daily)')
plt.title('BTC/USD Log-Price')
plt.show()


plt.figure(dpi = 1000)
sns.heatmap(Price.corr(), annot=True)
plt.title('Correlation Matrix Prices', fontdict={'fontsize':12}, pad=12)
plt.show()

plt.figure(dpi = 1000)
sns.heatmap(Log_Price.corr(), annot=True)
plt.title('Correlation Matrix Log-Prices', fontdict={'fontsize':12}, pad=12)
plt.show()



#=====================================================================
#Question 2: Testing Stationarity
#=====================================================================



def adf_test(timeseries): #Augmented Dickey-Fuller Test
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

adf_test(Log_Price.iloc[:,0])
adf_test(Log_Price.iloc[:,1])
adf_test(Log_Price.iloc[:,2])
adf_test(Log_Price.iloc[:,3])


def kpss_test(timeseries): #KPSS Test to complete the precedent test
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print (kpss_output)

kpss_test(Log_Price.iloc[:,0])
kpss_test(Log_Price.iloc[:,1])
kpss_test(Log_Price.iloc[:,2])
kpss_test(Log_Price.iloc[:,3])



#=====================================================
#Creating Stationary processes
#=====================================================



Log_Prices_Stat = np.subtract(Log_Price.iloc[1:, :], Log_Price.iloc[0:(len(Log_Price) - 1), :])

plt.figure(dpi = 1000, figsize=(16, 8))
plt.subplot(221)
plt.plot(Log_Prices_Stat.iloc[:, 0], color = 'red', linewidth = 0.6)
plt.ylabel('Dollar Index Returns')
plt.xlabel('Periods (Daily)')
plt.title('US Dollar Log-Returns')

plt.subplot(222)
plt.plot(Log_Prices_Stat.iloc[:, 1], color = 'red', linewidth = 0.6)
plt.ylabel('Log-Returns')
plt.xlabel('Periods (Daily)')
plt.title('Gold Log-Returns')

plt.subplot(223)
plt.plot(Log_Prices_Stat.iloc[:, 2], color = 'red', linewidth = 0.6)
plt.ylabel('Index')
```

```
plt.xlabel('Periods (Daily)')
plt.title('VIX Index')

plt.subplot(224)
plt.plot(Log_Prices_Stat.iloc[:, 3], color = 'red', linewidth = 0.6)
plt.ylabel('Log-Returns')
plt.xlabel('Periods (Daily)')
plt.title('BTC/USD Log-Returns')
plt.tight_layout()
plt.show()

adf_test(Log_Prices_Stat.iloc[:,0])
adf_test(Log_Prices_Stat.iloc[:,1])
adf_test(Log_Prices_Stat.iloc[:,2])
adf_test(Log_Prices_Stat.iloc[:,3])

kpss_test(Log_Prices_Stat.iloc[:,0])
kpss_test(Log_Prices_Stat.iloc[:,1])
kpss_test(Log_Prices_Stat.iloc[:,2])
kpss_test(Log_Prices_Stat.iloc[:,3])



#=============================================
#Question 3: VAR Model estimation
#=============================================



model = VAR(Log_Prices_Stat)
x = model.select_order(12)
print(x.summary()) #We choose a lag of one

#Other Method
for i in [1,2,3,4,5,6,7,8,9]:
    result = model.fit(i)
    print('Lag Order =', i)
    print('AIC : ', result.aic)
    print('BIC : ', result.bic)
    print('FPE : ', result.fpe)
    print('HQIC: ', result.hqic, '\n')

model_fitted = model.fit(1)
model_fitted.summary()

Covar = model_fitted.bse

Phi1 = model_fitted.coefs;
Phi0 = model_fitted.coefs_exog;
theta = [*Phi1.flatten(),*Phi0.flatten()]


#Impulse Reaction function

#### 3 Impulse
Cov = model_fitted.sigma_u
cov = np.array(Cov)
s = [cov[0,0]**0.5,cov[1,1]**0.5,cov[2,2]**0.5]
#p0 = np.array([cov[0,0]**0.5,cov[1,1]**0.5,cov[2,2]**0.5,0])
shocks = []
for i in range(len(s)):
    p0 = np.zeros((4,1))
    p0[i] = s[i]
    print(p0)
    p = np.add(Phi0,p0)
    phi1 = np.reshape(Phi1,(4,4))
    pp = [p]
    for i in range(1,6) :
        q = Phi0 + np.matmul(phi1,pp[i-1])
        pp.append(q)
        Bp = []
        for i in pp :
            Bp.append(i[3])
    shocks.append(Bp)

p0 = np.array([cov[0,0]**0.5,cov[1,1]**0.5,cov[2,2]**0.5,0])
p0 = np.reshape(p0,(4,1))
p = np.add(Phi0,p0)
phi1 = np.reshape(Phi1,(4,4))
pp = [p]
for i in range(1,6) :
    q = Phi0 + np.matmul(phi1,pp[i-1])
    pp.append(q)
    Bp = []
for i in pp :
    Bp.append(i[3])
shocks.append(Bp)
```

```
#Graphical Representation of the shocks
plt.figure(dpi = 1000, figsize=(12, 6))
plt.subplot(221)
plt.plot(shocks[0], color = 'red', linewidth = 1.1)
plt.ylabel('Bitcoin return')
plt.title('Impact of a 1 std shock of the Dollar on the BitCoin.')

plt.subplot(222)
plt.plot(shocks[1], color = 'red', linewidth = 1.1)
plt.ylabel('Bitcoin return')
plt.title('Impact of a 1 std shock of the Gold on the BitCoin.')

plt.subplot(223)
plt.plot(shocks[2], color = 'red', linewidth = 1.1)
plt.ylabel('Bitcoin return')
plt.xlabel('Periods (Daily)')
plt.title('Impact of a 1 std shock of the VIX on the BitCoin.')

plt.subplot(224)
plt.plot(shocks[3], color = 'red', linewidth = 1.1)
plt.ylabel('Bitcoin return')
plt.xlabel('Periods (Daily)')
plt.title('Combined shocks on the BitCoin.')
print(Phi0[3])
plt.tight_layout()


irf = model_fitted.irf(5)
irf.plot(orth=False)

fig = irf.plot(response=3)
fig.tight_layout()
fig.set_figheight(9)
fig.set_figwidth(8)



#=================================================
#Question 4:
#=================================================



def ML_VAR(theta,X):
    l1 =[theta[0], theta[1], theta[2],theta[3]]
    l2 =[theta[4], theta[5],theta[6], theta[7]]
    l3 =[theta[8], theta[9],theta[10], theta[11]]
    l4 = [theta[12], theta[13], theta[14],theta[15]]
    l5 = [theta[16], theta[17], theta[18],theta[19]]
    Phi1=[l1,l2,l3,l4]
    Phi0=[l5]
    Y=X.iloc[1:np.size(X,0),:]
    Z=X.iloc[0:(np.size(X,0)-1),:]
    Phi0_temp= np.matlib.repmat(Phi0,np.size(Z,0),1)
    temp=np.matmul(Z,Phi1)-Phi0_temp
    res=np.subtract(Y,temp)
    loglik=multivariate_normal.logpdf(res, mean=[0,0,0,0], cov=np.identity(4))
    loglik=-np.sum(loglik)
    return loglik


Constraint = ({'type':'eq', 'fun': lambda x: x[3]},
              {'type':'eq', 'fun': lambda x: x[7]},
              {'type':'eq', 'fun': lambda x: x[11]})

estimation_output2 = minimize(ML_VAR, theta, method='SLSQP', constraints = Constraint, args=(Log_Prices_Stat))
#Constrained model
estimated_para2 = estimation_output2.x
Cons_Likelihood = (- 1) * estimation_output2.fun

Phi1 = model_fitted.coefs;
Phi0 = model_fitted.coefs_exog;
theta = [*Phi1.flatten(),*Phi0.flatten()]

estimation_output = minimize(ML_VAR, theta, method='SLSQP', args=(Log_Prices_Stat)) #Unconstrained model
estimated_para = estimation_output.x
Uncons_Likelihood = (- 1) * estimation_output.fun



#=============================================================
#Likelihood Test
#=============================================================



LR_Test = 2 * (Uncons_Likelihood - Cons_Likelihood)
p_value = chi2.sf(LR_Test, 3)
```