TIME SERIES ANALYSIS FOR ETHEREUM PRICE

2021-22

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

INSTRUCTOR:

Prof. AMIT MITRA

COMPLETED BY:

SHWETANK SINGH - 201422

SANGITA RATHOD-201402

NISTHA SHAH -201358

NIKHIL MUNAKHIYA-201352

GADDAM PRIYANKA-201311

ACKNOWLEDGEMENT

The success and final outcome of this project work required a lot of guidance and assistance from many people and we extremely fortunate to have got this all along the completion of our project work. Whatever we have done is only due to such guidance and assistance and we would like to thank all of them.

We respect and thank **Prof. Amit Mitra**, Department of Mathematics and Statistics, IIT Kanpur for giving us an opportunity to do this project work and providing us all support and guidance which made us complete the project on time. We are extremely grateful to him for providing such a great support and guidance

Table of Contents

ACKNOWLEDGEMENT 2

ABSTRACT 4

1. **INTRODUCTION:** 4

2. **DATA DESCRIPTION:** 4

2.1 Source of data: 5

3. **THEORY:** 5

3.1 Relative Ordering Test 5

3.2 Turning Point Test 5

3.3 Augmented Dickey-Fuller Test 6

3.4 ARMA Model 7

3.5 ACF Plot and PACF Plot 7

4. **DATA AT A GLANCE** 8

4.1 Exploratory Data Analysis 9

4.2 Testing and Elimination of Deterministic components 10

5 RESULTS 13

6. CONCLUSION 17

7. REFERENCE 17

ABSTRACT

The Cryptocurrency Market is highly volatile and there are significant challenges in fitting a good model on a Cryptocurrency market data. This project is an attempt to study the closing price of Ethereum Cryptocurrency data for the last 6 years, Sept. 2015 to Oct. 2021. We use the tools of Time Series analysis to analyse the data. Firstly, we suitably decompose the time series data to obtain its deterministic components and consequently remove the non-stationarity present in the data. Henceforth, we move on to find an appropriate model to model the stationary part, particularly using ARIMA methodologies. Finally, we forecast future values based on past observations to check the accuracy of the model.

1. INTRODUCTION:

The Cryptocurrency market is very unpredictable, any geopolitical change can impact the trend of Cryptocurrencies in the crypto market. This is why it has always drawn the interest and attention of various professionals in the field of science or commerce. In order to get ahead of the market, statisticians have always tried to analyse and forecast the Cryptocurrency price that in essence, reflects all known and unknown information in the public domain.

This project work is an attempt to perform *Time Series Analysis of the Ethereum Cryptocurrency Price*. The motive of this project is to analyse the Ethereum Cryptocurrency closing price movement and possibly fit a suitable model to make forecasts.

Time series is a collection of data points indexed over time. Time series are analysed in order to understand the underlying structure that produce the observation. Time series data is of two types:

1. Univariate Time Series - It is a time series in which observations are sequentially recorded on a single variable over time.
2. Multivariate Time Series - It is a time series in which observations are sequentially recorded on more than one variable over time.

What is Ethereum?

Ethereum is open access to digital money and data-friendly services for everyone – no matter your background or location. It's a community-built technology behind the cryptocurrency ether (ETH) and thousands of applications you can use today.

ETH is a smart contract platform that enables developers to build decentralized applications (dapps) conceptualized by Vitalik Buterin in 2013. ETH is the native currency for the Ethereum platform and also works as the transaction fees to miners on the Ethereum network. Presently ETH has second largest market capitalisation of all cryptocurrencies formed.

2. DATA DESCRIPTION:

We have taken this data on Ethereum cryptocurrency over 6 years (2015 -2021). This data consists of 2221 days cryptocurrency prices (in dollars) from 01 September 2015 to 01 November 2021.

2.1 Source of data:

Data can be obtained through following link:

https://finance.yahoo.com/quote/ETH-USD/history?p=ETH-USD

3. THEORY:

3.1 Relative Ordering Test

This is a non-parametric test procedure used for testing the existence of trend components. Let the time series be denoted by $\{X_1, X_2, \ldots X_n\}$.

Define

$$q_{ij} = \begin{cases} 1, & if\ X_i > X_j\ where\ i < j \\ 0, & Otherwise \end{cases}$$

$$Q = \sum_i \sum_j q_{ij},\ i < j$$

Where Q counts the number of decreasing point in the time series and is also the number of discordances.

Null Hypothesis H_0: There is no trend in the time series

against the Alternate Hypothesis H_1: There is a trend in the time series

Under the null hypothesis, $E(Q) = \sum_i \sum_j E(q_{ij}) = \frac{n(n-1)}{4}$.

If observed the Q < E(Q) then it would be an indication of rising trend and if observed Q> E(Q) then it would be the indication of falling trend. If the observed Q does not differ "significantly" from E(Q) (Under the null hypothesis) then it would indicate no trend. Q is related to the Kendall's τ the rank correlation coefficient through the relationship

$$\tau = 1 - \frac{4Q}{n(n-1)}$$

Under the null hypothesis, $E(\tau) = 0$ and $Var(\tau) = \frac{2(2n+5)}{9n(n-1)}$.

Test Statistics: $Z = \frac{\tau - E(\tau)}{\sqrt{v(\tau)}}$ follow normal N(0,1) asymptotically (under Null hypothesis) We would reject the null hypothesis of no trend at level of significant α if observed $|z| > \tau_{\alpha/2}$ where $\tau_{\alpha/2}$ is the α/2 th upper cutoff points of a standard normal distribution.

3.2 Turning Point Test

Turning point test is a non-parametric test which is used for testing randomness of the time series data set. Let $X_1, X_2, X_3, \ldots X_n$ be the data, then X_i is considered as a turning point if

either $X_{i-1} < X_i$ and $X_i > X_{i+1}$ or $X_{i-1} > X_i$ and $X_i < X_{i+1}$. We count the number of turning points in the data. Define

$$U_i = \begin{cases} 1, if\ X_i\ is\ a\ turning\ point \\ 0, \qquad\qquad otherwise \end{cases}$$

Suppose P is the total number of turning points i.e., $P = \sum_{n=2}^{n-1} U_i$

Null Hypothesis H_0: Series is truly random (Does not contain any deterministic components.) Against the Alternative hypothesis H_1: Series is not truly random.

$$\text{Under the null hypothesis, } E(P) = \frac{2(n-2)}{3} \quad \text{and } Var(P) = \frac{16n-29}{90}$$

$$\text{Test statistics: } Z = \frac{P - E(P)}{\sqrt{\frac{16n-29}{90}}} = \frac{P - \frac{2(n-2)}{3}}{\sqrt{\frac{16n-29}{90}}} \quad \text{follows N(0, 1) asymptotically.}$$

We would reject null hypothesis H_0 at level of significance α if observed $|Z| > \tau_{\alpha/2}$ Where $\tau_{\alpha/2}$ is upper α/2 cutoff point of N(0,1).

3.3 Augmented Dickey-Fuller Test

An Augmented Dickey–Fuller test (ADF) is a test for a unit root in a time series sample. It is an augmented version of the Dickey–Fuller test for a larger and more complicated set of time series models.

The augmented Dickey–Fuller (ADF) statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence.

Let $\{X_t\}$ follows the AR(p) model with mean μ given by

$$X_t - \mu = \varphi_1(X_t - \mu) + \ldots\ldots + \varphi_p(X_{t-p} - \mu + Z_t),\ \{Z_t\} \sim WN(0,\sigma^2)$$

Above equation can be written as

$$\nabla X_t = \varphi_0^* + \varphi_1^* X_{t-1} + \sum_{j=2}^{p} \varphi_j^* \nabla X_{t-j+1} + Z_t$$

$$\text{Where } \varphi_0^* = \mu(i - \sum_{i=1}^{p} \varphi_i),$$

$$\varphi_1^* = \sum_{i=1}^{p} \varphi_i - 1,$$

$$\varphi_j^* = -\sum_{i=1}^{p} \varphi_i,$$

Now if X_t follows AR(p) then ∇X_t follows AR(p − 1)

Hypothesis of Interest:

H_0 : Data is non stationary or $\varphi_1^* = 0$

H_1 : Data is Stationary or $\varphi_1^* < 0$

The estimated standard error of φ_1^* is $SE(\varphi_1^*) = S\ (\sum_{i=2}^{p}(X_{i-1} - \bar{X}))^{\wedge}0.5$

where $S^2 = \sum_{t=2}^{n}(\nabla X_t = \varphi_0^* + \varphi_1^* X_{t-1} + \sum_1^p \varphi_i * \nabla X_{t-j+1})$ ^2/(n − p − 2)

The test statistic for calculating the ADF test is $\tau_\mu^\Lambda = \varphi_1^* / SE(\varphi_1^*)$

Test Criterion:

we reject H_0 if $\tau_\mu^\Lambda <$ DFα. From the test we have found out that $\tau_\mu^\Lambda = -8.6969$ and $DF_{0.05} = -2.863$, so, we reject the null at 5% level, i.e., the residual data we have derived is stationary one.

3.4 ARMA Model

Given a time series X_t, the ARMA model is a tool to understand and predict future values of the series. The AR part involves regressing variables on its own lag and MA part involves modeling the error term up to suitable lag as a linear combination.

Suppose p is the lag for AR model and q is the lag for MA model. Then AR(p) model will be

$$X_t = c + \sum_1^p \varphi_i X_{t-i} + \epsilon_t$$

where $\varphi_1, \varphi_2, ..., \varphi_p$ are parameters with $\varphi_p \neq 0$, c is a constant and ϵ_t is white noice. Similarly, MA(q) model will be

$$X_t = \mu + \epsilon_t + \sum_1^q \theta_i \epsilon_{t-i}$$

where $\theta_1, \theta_2, ..., \theta_q$ are parameters of the model with $\theta_q \neq 0$, μ is the expectation of X_t (can be assumed to be zero) and $\epsilon_t , \epsilon_{t-1}, ...$ are again white noise terms.

Hence ARMA (p, q) model can be written as,

$$X_t = c + \epsilon_t + \sum_1^p \varphi_i \epsilon_{t-i} + \sum_1^q \theta_i \epsilon_{t-i}$$

with usual assumptions on parameters.

3.5 ACF Plot and PACF Plot

Autocorrelation and partial autocorrelation plots are heavily used in time series analysis and forecasting.

These are plots that graphically summarize the strength of a relationship between an observation of a time series and observations at prior time steps. Plots of autocorrelation function (ACF) and partial autocorrelation function (PACF) give us different viewpoints of time series.

A partial autocorrelation plot is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed i.e., PACF only describes the direct relationship between an observation and its lag. This would suggest that there would be no correlation for lag values beyond k. While ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information.

The lags p and q of AR and MA model can be determined from ACF and PACF plots. For plotting ACF, we compute correlation between X_t and X_{t-k} for different lag values k and plot them in the graph. So, it is quite natural to have negative values as well. Now if the correlation values come within the significance band, then we can assume that the correlations are indifferent from zero. So, generally we take that value of k for MA as q, for which the correlation will cross the significant band for the last time i.e., we can assume current time point X_t is directly and indirectly dependent on previous q many time points. Now for plotting PACF, we regress current data point on previous time series data points.

Then we plot the coefficients on the graph where each coefficients indicate the effect of corresponding previous data points. Now similar to ACF plot, if a value goes outside the band for some k, we assume that the observation with lag k has a direct effect on the current observation.

4. DATA AT A GLANCE

Let's see the last five observation of our data

Date	Close
2021-09-26	3062.265381
2021-09-27	2934.138916
2021-09-28	2807.296631
2021-09-29	2853.143311
2021-09-30	3001.678955

Figure 1: Last five observations

Log transformed data:

Date	Close	logged
2021-09-26	3062.265381	8.026910
2021-09-27	2934.138916	7.984169
2021-09-28	2807.296631	7.939977
2021-09-29	2853.143311	7.956177
2021-09-30	3001.678955	8.006927

Figure 2: Last five observations

4.1 Exploratory Data Analysis

The plots of the original data (X_t) and the log transformed data ($\log X_t$) are presented below in

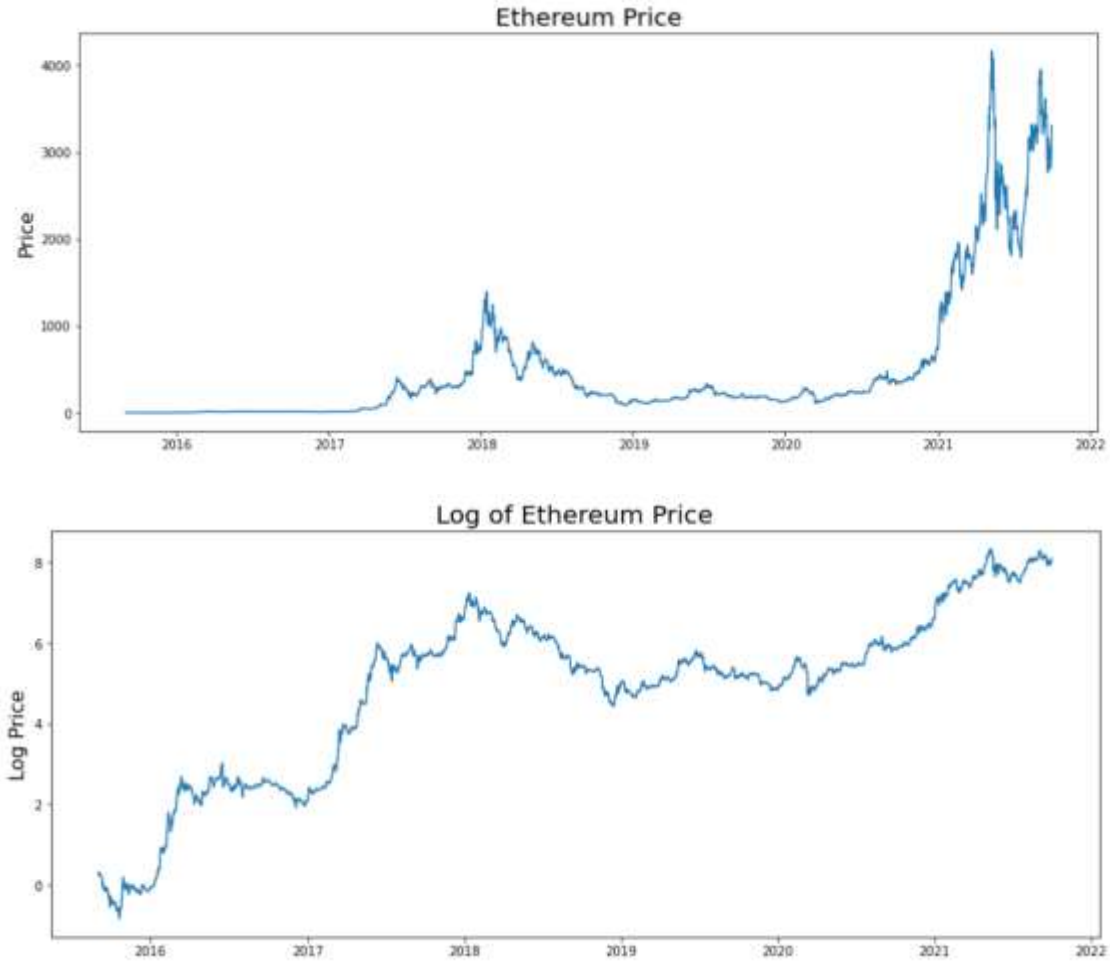
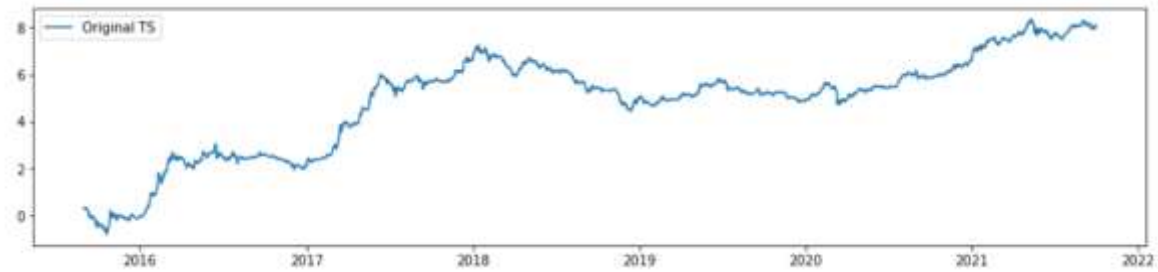


Figure 4.1.1: Plots of original data

By looking at both the plots, it is clear that both the plots have an increasing trend and are not white noise. The log transformed data is considered for further analysis because taking logarithm smooths the trend curve and in turn, helps to forecast easily. Thus our concerned model is given by:

$$\log X_t = Y_t = m_t + s_t + \epsilon_t$$

An attempt is made to forecast the Ethereum price for the last 10 values. Keeping in mind that stock market is functional only during 365 days of the year, the train data is decomposed into its deterministic and stochastic components as shown-



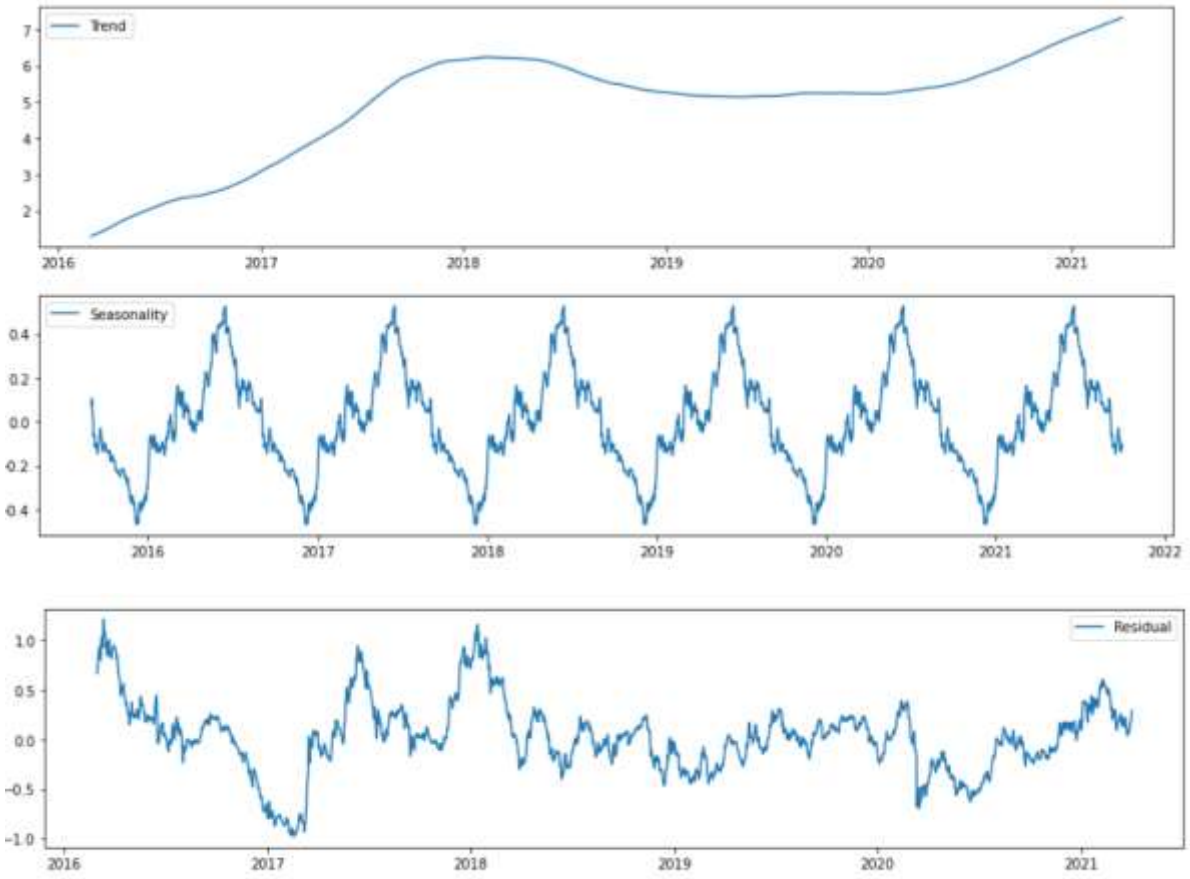


Figure 4.1.2: Plots of log transformed data

Clearly, one can see that there is an increasing trend in the data. So, we applied a non-parametric test for a formal diagnostic of this trend. In particular, we used the Relative ordering test (see Section 3.1 for details).

```
Q =  534637
Z =  39.92566493338747
t_alpha/2 =  1.959963984540054
Trend is present
```

We found that there was a significant trend present. The value of test-statistic came out be 39.9256 which is quite off from the critical z−values which was also visible from the graph.

4.2 Testing and Elimination of Deterministic components

From the above test, it is evident that it has increasing trend and is non-stationary.

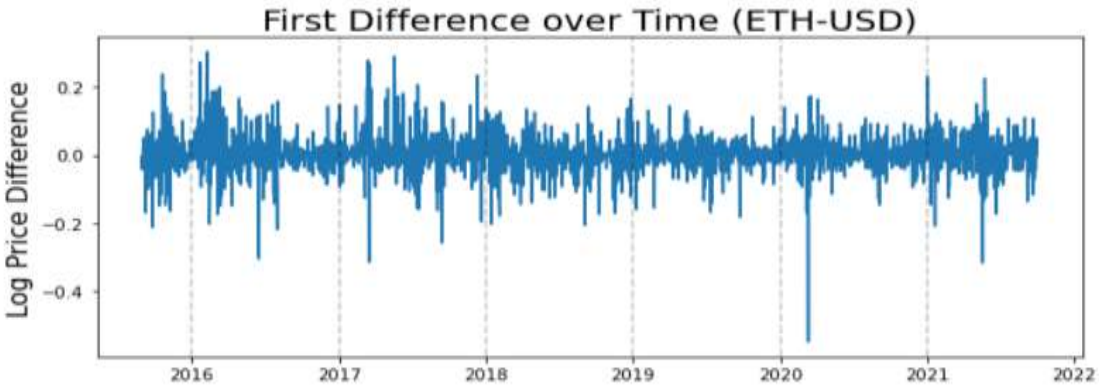


Figure 4.2.1: Detrended series after First order differencing

Then in order to remove trend, we applied a differencing operator of lag 1 on our data. The resultant series Z_t was obtained by the following relation,

$$Z_t = Y_t - Y_{t-1},$$

where Y_t are the values of the original series. The intuition behind choosing the order of differencing to be 1 was that the data is a Ethereum price data and often the prices are assumed to be dependent on the immediate previous value. Figure 4.2.1 shows the series obtained after differencing. One may notice that the trend is significantly removed. Nonetheless, we again applied the Relative ordering test of Section 3.1 to back our conclusion with statistical evidence. The value of the test-statistic was obtained to be equal to -8.697 with a p−value of -2.863. Hence, we concluded that the null hypothesis is true and that the trend is indeed removed from the data.

Looking at the graph, we then hypothesized that our detrended series is purely random and free from any deterministic fluctuations. For testing this hypothesis, we applied the Turning point test of Section 3.2. The evidence from the data was insufficient and we failed to reject the Null hypothesis that the series is purely random. So now we have a series which has no deterministic components in it. A natural way to proceed will be to fit standard time series model on it and see which gives the best approximation. However, before doing that, we need to first verify that the series is stationary. For that, we applied the Augmented Dickey-Fuller Test from Section 3.3.

As described in Section 3.3, the Null hypothesis for the Augmented Dickey-Fuller test is that the series is non-stationary. We used an in-built function from a python library to carry out this test. The test statistic was obtained to be equal to 1.489 against the critical value of 1.96 at 5% level of significance. Hence, we rejected the null hypothesis that the series is non-stationary and proceeded to fitting different models to this series.

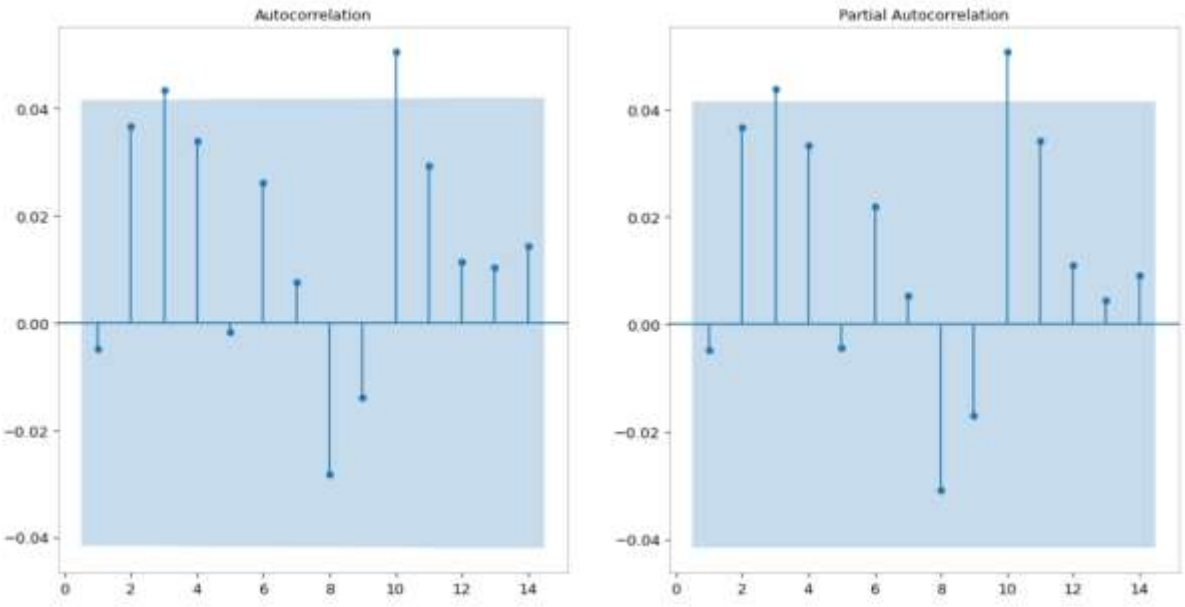


Figure 4.2.2: ACF and PACF plots of the data

We described in section 3.5 a method to identify the model parameters for a time-series data. In both the plots we will look for the first lag for which the ACF / PACF value lies outside the insignificant band. The first such lag from the ACF plot will be considered for the MA parameter while that from the PACF plot will be considered for the AR parameter. From Figure 4.2.2, we can see that in both the plots, the first lag at which the value goes outside the insignificant band. Hence, we will consider all the models with p and q. This gives us a total of 121 different models. From these 121 models, we will then identify the best model by the criterion of minimizing AIC and BIC. Below we have AIC and BIC values for these models.

```
array([[-6178.84528552, -6188.89917047, -6183.93427292, -6182.30009313,
        -6180.36535621, -6179.35783608, -6178.00292116, -6175.87456098,
        -6176.87962181, -6182.08269755, -6180.02146027],
       [-6188.94891142, -6183.78046509, -6181.95126777, -6180.36897549,
        -6178.35900787, -6182.11283176, -6183.83380727, -6179.31634688,
        -6174.90791315, -6184.16601873, -6183.5795504 ],
       [-6182.11959424, -6181.63200581, -6179.95728771, -6182.44306713,
        -6180.51085493, -6178.65395723, -6181.59325382, -6177.79827773,
        -6170.64946589, -6181.54302553, -6182.0936314 ],
       [-6182.54122733, -6180.7435382 , -6182.84959296, -6178.76108913,
        -6182.18608252, -6176.35701332, -6174.55397996, -6178.13531345,
        -6173.05727344, -6178.66750224, -6180.00394311],
       [-6180.5857696 , -6182.75371658, -6179.89090186, -6177.64734598,
        -6174.93742931, -6175.14447458, -6172.44479848, -6176.48561281,
        -6176.1243217 , -6182.33856413, -6181.95450153],
       [-6179.64587816, -6177.7004392 , -6183.12110181, -6178.13942339,
        -6178.86751654, -6175.88438514, -6170.3170915 , -6173.59478014,
        -6171.03266538, -6178.19106062, -6175.04595981],
       [-6177.70219546, -6175.7608257 , -6177.59567652, -6174.99316528,
        -6178.28515591, -6170.45269687, -6176.41076871, -6176.12379249,
        -6174.21281905, -6179.11894951, -6173.46586089],
       [-6177.8286461 , -6182.06360458, -6175.01057191, -6177.9810026 ,
        -6178.92972695, -6175.67813086, -6173.24318879, -6174.09193544,
        -6171.00616944, -6182.56097385, -6190.68775814],
       [-6177.79432616, -6177.88465376, -6177.17553016, -6173.8387316 ,
        -6172.0545382 , -6171.02033501, -6170.46494604, -6170.68859578,
        -6170.51427606, -6177.80122774, -6178.883846  ],
       [-6185.01955175, -6185.38855413, -6177.76075907, -6175.94497584,
        -6174.88903139, -6173.64478679, -6175.53571216, -6171.45448873,
        -6178.39582684, -6175.02865197, -6176.69241144],
       [-6180.74280773, -6183.43248678, -6182.27796853, -6173.8767776 ,
        -6177.72402909, -6174.90770609, -6173.30256026, -6171.22789149,
        -6175.6823492 , -6173.36816073, -6174.82142581]])
```

Figure 4.2.3: AIC values

```
array([[-6156.02603783, -6160.37511085, -6149.70540138, -6142.36640967,
        -6134.72686082, -6128.01452877, -6120.95480193, -6113.12162983,
        -6108.42187874, -6107.92014255, -6100.15409335],
       [-6160.42485181, -6149.55159355, -6142.01758431, -6134.7304801 ,
        -6127.01570056, -6125.06471253, -6121.08087612, -6110.85860381,
        -6100.74535815, -6104.29865181, -6098.00737156],
       [-6147.89072271, -6141.69832235, -6134.31879233, -6131.09975982,
        -6123.4627357 , -6115.90102608, -6113.13551074, -6103.63572273,
        -6090.78209897, -6095.97084669, -6090.81664063],
       [-6142.60754387, -6135.10504281, -6131.50628565, -6121.7129699 ,
        -6119.43315137, -6107.89927024, -6100.39142497, -6098.26794653,
        -6087.4850946 , -6087.39051147, -6083.02214042],
       [-6134.94727422, -6131.41040928, -6122.84278263, -6114.89441483,
        -6106.47968623, -6100.98191958, -6092.57743156, -6090.91343397,
        -6084.84733093, -6085.35676144, -6079.26788692],
       [-6128.30257085, -6120.65231997, -6120.36817066, -6109.68168031,
        -6104.70496155, -6096.01701822, -6084.74491265, -6082.31778937,
        -6074.05086269, -6075.50444601, -6066.65453327],
       [-6120.65407623, -6113.00789455, -6109.13793344, -6100.83061028,
        -6098.41778898, -6084.88051802, -6085.13377794, -6079.1419898 ,
        -6071.52620444, -6070.72752297, -6059.36962163],
       [-6115.07571494, -6113.6058615 , -6100.84801691, -6098.11363568,
        -6093.35754811, -6084.40114009, -6076.2613861 , -6071.40532082,
        -6062.61474291, -6068.46473539, -6070.88670776],
       [-6109.33658309, -6103.72209877, -6097.30816324, -6088.26655276,
        -6080.77754743, -6074.03853232, -6067.77833143, -6062.29716924,
        -6056.4180376 , -6058.00017736, -6053.37798369],
       [-6110.85699675, -6105.52118721, -6092.18858022, -6084.66798507,
        -6077.9072287 , -6070.95817217, -6067.14428563, -6057.35825028,
        -6058.59477646, -6049.52278967, -6045.48173721],
       [-6100.87544081, -6097.86030794, -6091.00097776, -6076.89497491,
        -6075.03741448, -6066.51627956, -6059.2063218 , -6051.42684111,
        -6050.1764869 , -6042.1574865 , -6037.90593966]])
```

Figure 4.2.4: BIC values

From the above images, we conclude that the best model is ARMA(8,11) since it gives us the lowest AIC and BIC. Using ARIMA(8,1,11) on the original data we forecast the 10 future values of the Ethereum Cryptocurrency price .

5 RESULTS

The purpose of this project is to observe the predictability in the Ethereum Cryptocurrency prices, i.e., if the history tells us anything about its future movement. And if positive, then find the most suitable model and approximate values for time in future. We have established that the process is not ideally random. Of the deterministic components, there exists trend and seasonality but the seasonality is not significant on the data as one would expect for the Cryptocurrency prices. Cyclical component, however, is not observable in the chosen length of data.

We conclude that ARIMA(8,1,11) is the most suitable fit for transformed (log) prices. A plot of the fitted values superimposed over original values is shown in Figure 5.3.

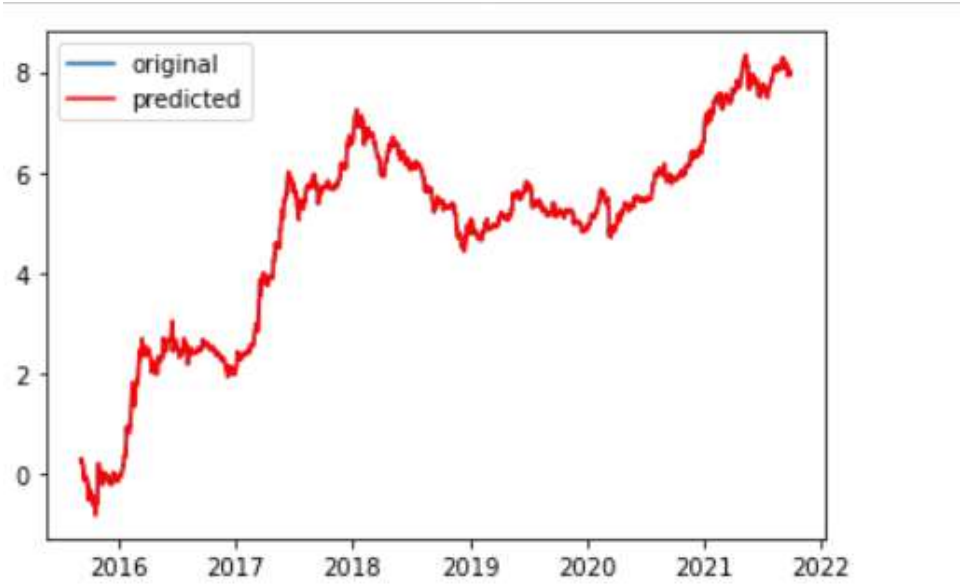


Figure 5.1: Fitted values over original data

Below is the summary of the fitted model:

SARIMAX Results

Dep. Variable:	logged	**No. Observations:**	2219
Model:	ARIMA(8, 1, 11)	**Log Likelihood**	3112.687
Date:	Wed, 17 Nov 2021	**AIC**	-6185.374
Time:	02:16:28	**BIC**	-6071.287
Sample:	0	**HQIC**	-6143.703
	- 2219		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3019	0.172	-1.754	0.080	-0.639	0.036
ar.L2	0.4448	0.147	3.031	0.002	0.157	0.732
ar.L3	0.3820	0.168	2.268	0.023	0.052	0.712
ar.L4	0.2759	0.182	1.517	0.129	-0.081	0.632
ar.L5	0.1886	0.179	1.052	0.293	-0.163	0.540
ar.L6	0.1981	0.163	1.214	0.225	-0.122	0.518
ar.L7	-0.0128	0.139	-0.092	0.927	-0.286	0.260
ar.L8	-0.4560	0.130	-3.506	0.000	-0.711	-0.201

ma.L1	0.2972	0.172	1.724	0.085	-0.041	0.635
ma.L2	-0.4112	0.149	-2.759	0.006	-0.703	-0.119
ma.L3	-0.3228	0.164	-1.963	0.050	-0.645	-0.001
ma.L4	-0.2564	0.181	-1.416	0.157	-0.611	0.098
ma.L5	-0.2211	0.175	-1.262	0.207	-0.565	0.122
ma.L6	-0.2052	0.159	-1.295	0.195	-0.516	0.105
ma.L7	-0.0148	0.135	-0.109	0.913	-0.280	0.251
ma.L8	0.4171	0.126	3.299	0.001	0.169	0.665
ma.L9	-0.0587	0.023	-2.532	0.011	-0.104	-0.013
ma.L10	0.0873	0.023	3.847	0.000	0.043	0.132
ma.L11	0.0932	0.027	3.467	0.001	0.041	0.146
sigma2	0.0035	5.91e-05	59.745	0.000	0.003	0.004

Ljung-Box (L1) (Q):	0.03	**Jarque-Bera (JB):**	3997.82
Prob(Q):	0.87	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.58	**Skew:**	-0.28
Prob(H) (two-sided):	0.00	**Kurtosis:**	9.55

Figure 5.2 : Summary of the fitted model

The AIC of this model came out to be around -6190.6877 and BIC around -6070.8867. Figure 5.2 shows summary of the in-built fit function of a python library. From the summary, we see that all the coefficients of the model are significant at 5% level of significance. If the original series is $\{X_t\}$, the final model equation is written as follows,

$$Z_t = \mu + \varphi_1 Z_{t-1} + \varphi_2 Z_{t-2} + \varphi_3 Z_{t-3} + \varphi_4 Z_{t-4} + \varphi_5 Z_{t-5} + \varphi_6 Z_{t-6} + \varphi_7 Z_{t-7} + \varphi_8 Z_{t-8} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \theta_3 \epsilon_{t-3} + \theta_4 \epsilon_{t-4} + \theta_5 \epsilon_{t-5} \theta_1 + \theta_6 \epsilon_{t-6} + \theta_7 \epsilon_{t-7} \theta_1 + \theta_8 \epsilon_{t-8} + \theta_9 \epsilon_{t-9} + \theta_{10} \epsilon_{t-10} + \theta_{11} \epsilon_{t-11} + \epsilon_t \,, \quad \text{with}$$

$$\varphi_1 = -0.3019,$$

$$\varphi_2 = 0.4448,$$

$$\varphi_3 = 0.3820,$$

$$\varphi_4 = 0.2759,$$

$$\varphi_5 = 0.1886,$$

$$\varphi_6 = 0.1981,$$

$$\varphi_7 = -0.0128,$$

φ_8 = -0.4560,

θ_1 = 0.2972

θ_2 = -0.4112

θ_3 = -0.3228

θ_4 = -0.2564

θ_5 = -0.2211

θ_6 = -0.2052

θ_7 = -0.0148

θ_8 = 0.4171

θ_9 = -0.0587

θ_{10} = 0.0873

θ_{11} = 0.0932

where $Z_t = X_t - X_{t-1}$ and $\epsilon_t \sim N(0, \sigma 2)$; $\sigma^2 = 0.0035$ for all t.

After forecasting the future 10 values, we compared them to the actual 10 test values. The plot for the forecasted and the actual values is shown in Figure 5.3.

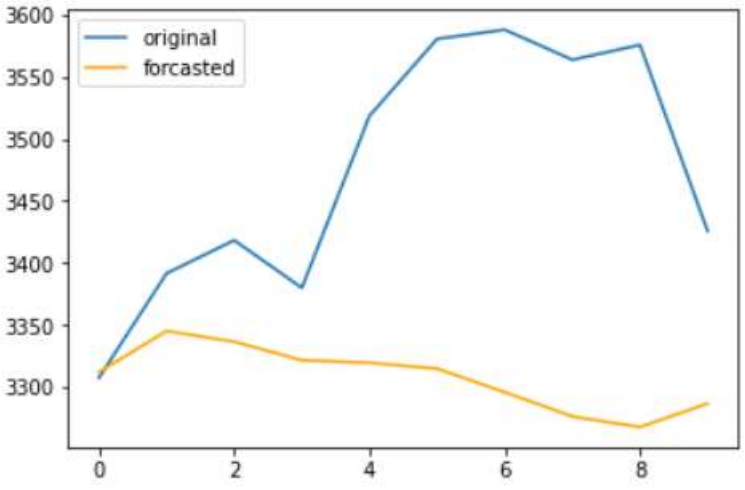


Figure 5.3: Plot of original and forecasted values

```
3312.1669098132747
3345.309455990676
3336.680254911625
3321.7200143952005
3319.713968019474
3314.9542023633553
3296.016329827274
3276.4829813090882
3267.8972047825905
3286.8524687978957
```

Figure 5.4: Predicted values

We got the Mean Absolute Percentage error as 4.755

6. CONCLUSION

To conclude, we found that ARMA(8,11) fits best on the series obtained after first order differencing i.e., ARIMA(8,1,11) fits best for the original data on Ethereum Cryptocurrency prices. The model equation is given by (1).

We employed the standard approach of first eliminating the trend from the data. We used first order differencing for this. After eliminating the trend, we found that the resultant series was purely random and stationary. We plotted the ACF and PACF plots for this resultant series and identified the candidate models for this data. Finally, we fitted all the models and based on the criterion of minimizing AIC, we found the best model. As can be seen from the trace plot in Figure 5.1, the best model was able to fit the data quite well.

7. REFERENCE

1. Time Series Class Notes by Dr. Amit Mitra under MTH 517A.

2. Forecasting: Principles and Practice by Rob J Hyndman and George Athanasopoulos.

3. Introduction to Time Series and Forecasting by Peter J. Brockwell Richard A. Davis.

In [78]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import calendar


import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import pie
from matplotlib import gridspec
import matplotlib.ticker as mtick

import warnings
warnings.filterwarnings('ignore')

import scipy.stats
from scipy.stats import norm, chi2
import os
```

In [79]:

```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /opt/conda/lib/python3.7/site-packages (0.1.66
)
Requirement already satisfied: numpy>=1.15 in /opt/conda/lib/python3.7/site-packages (fro
m yfinance) (1.19.5)
Requirement already satisfied: multitasking>=0.0.7 in /opt/conda/lib/python3.7/site-packa
ges (from yfinance) (0.0.10)
Requirement already satisfied: lxml>=4.5.1 in /opt/conda/lib/python3.7/site-packages (fro
m yfinance) (4.6.3)
Requirement already satisfied: requests>=2.20 in /opt/conda/lib/python3.7/site-packages (
from yfinance) (2.25.1)
Requirement already satisfied: pandas>=0.24 in /opt/conda/lib/python3.7/site-packages (fr
om yfinance) (1.3.4)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (fr
om pandas>=0.24->yfinance) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-pa
ckages (from pandas>=0.24->yfinance) (2.8.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from p
ython-dateutil>=2.7.3->pandas>=0.24->yfinance) (1.16.0)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (fr
om requests>=2.20->yfinance) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packag
es (from requests>=2.20->yfinance) (2021.10.8)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python3.7/site-package
s (from requests>=2.20->yfinance) (4.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.7/site-pac
kages (from requests>=2.20->yfinance) (1.26.6)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting
behaviour with the system package manager. It is recommended to use a virtual environment
instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [80]:

```
import yfinance as yf
```

In [81]:

```
#define the ticker symbol
tickerSymbol = 'ETH-USD'
```

In [82]:

```
#get data on this ticker
tickerData = yf.Ticker(tickerSymbol)
```

In [83]:

```
#get the historical prices for this ticker
tickerDf = tickerData.history(period='1d', start='2015-9-1', end='2021-10-1')
```

In [84]:

```
tickerDf = tickerDf[['Close']]
```

In [85]:

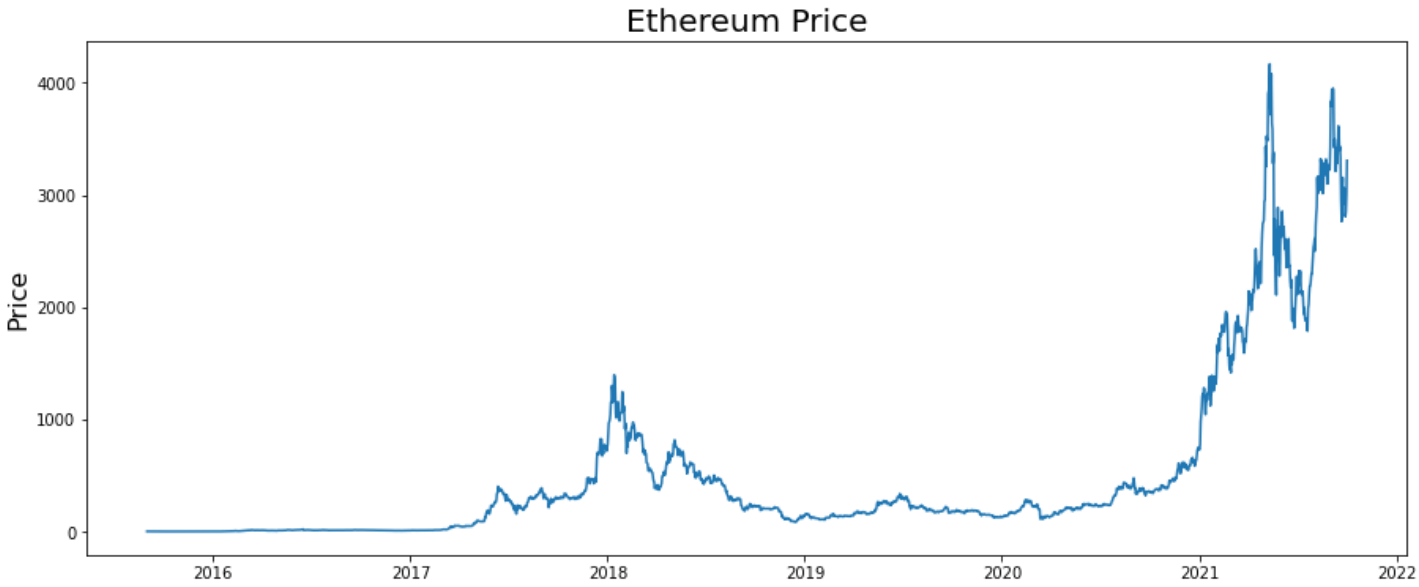
```
tickerDf.tail()
```

Out[85]:

Date	Close
2021-09-27	2934.138916
2021-09-28	2807.296631
2021-09-29	2853.143311
2021-09-30	3001.678955
2021-10-01	3307.516113

In [86]:

```
plt.figure(figsize=(15,6))
plt.plot(tickerDf['Close'], label='Ethereum')
plt.title('Ethereum Price', fontsize=20)
plt.ylabel('Price', fontsize=16)
plt.show()
```



In [87]:

```
tickerDf['logged'] = np.log(tickerDf['Close'])
```

In [88]:

```
tickerDf.tail()
```

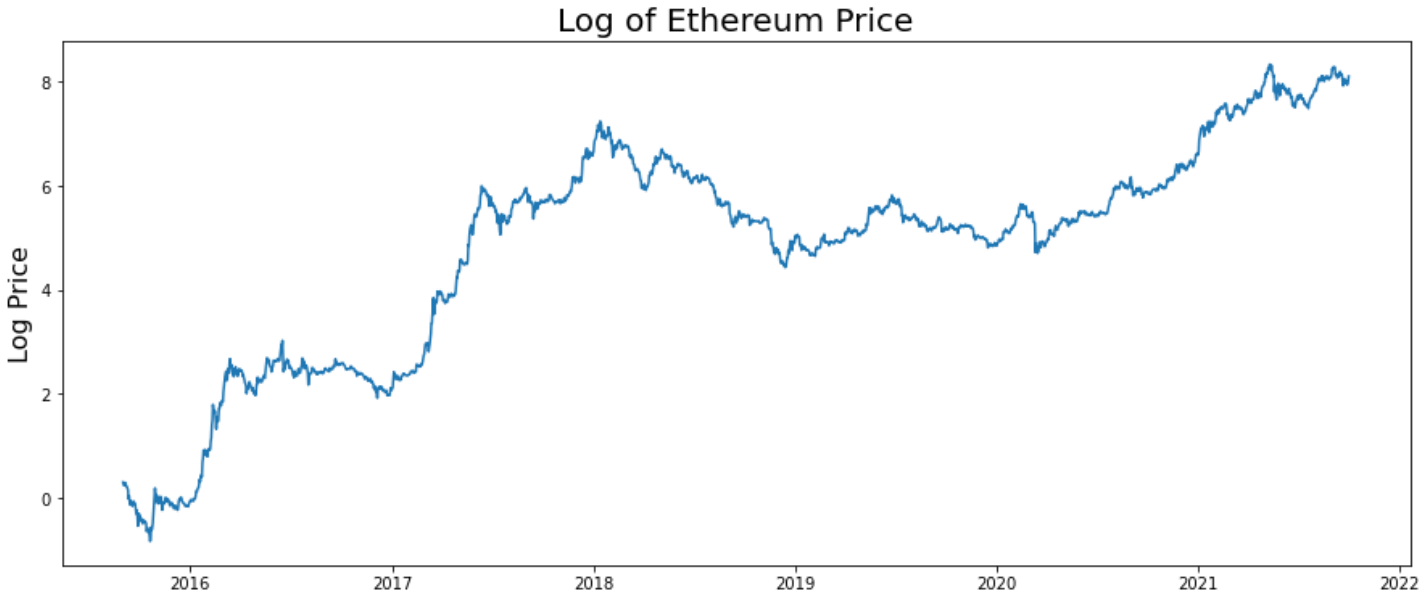
Out[88]:

Date	Close	logged
2021-09-27	2934.138916	7.984169

2021-09-28	2807.20[illegible]	7.[illegible]
2021-09-29	2853.143311	7.956177
2021-09-30	3001.678955	8.006927
2021-10-01	3307.516113	8.103953

In [89]:

```
plt.figure(figsize=(15,6))
plt.plot(tickerDf['logged'], label='Ethereum')
plt.title('Log of Ethereum Price', fontsize=20)
plt.ylabel('Log Price', fontsize=16)
plt.show()
```



In [90]:

```
def turningpoints(x):
    N=0
    for i in range(1, len(x)-1):
        if ((x[i-1] < x[i] and x[i+1] < x[i]) or (x[i-1] > x[i] and x[i+1] > x[i])):
            N += 1
    return N


t = turningpoints(tickerDf['logged'])
expectatoion = ((2*tickerDf.shape[0]) - 4) / 3
variance = ((16*tickerDf.shape[0]) - 29) / 90
z_score = (t - expectatoion) / (variance**0.5)
print("z-score = ",z_score)
p_value = scipy.stats.norm.sf(abs(z_score))*2
print("p-value = ",p_value)# p<0.05 so null hypothesis is rejected at 5% level of signifi
cance. i.e. the the time series is not random.
```

```
z-score =  -15.16095463533037
p-value =  6.413038368338921e-52
```

In [91]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
```

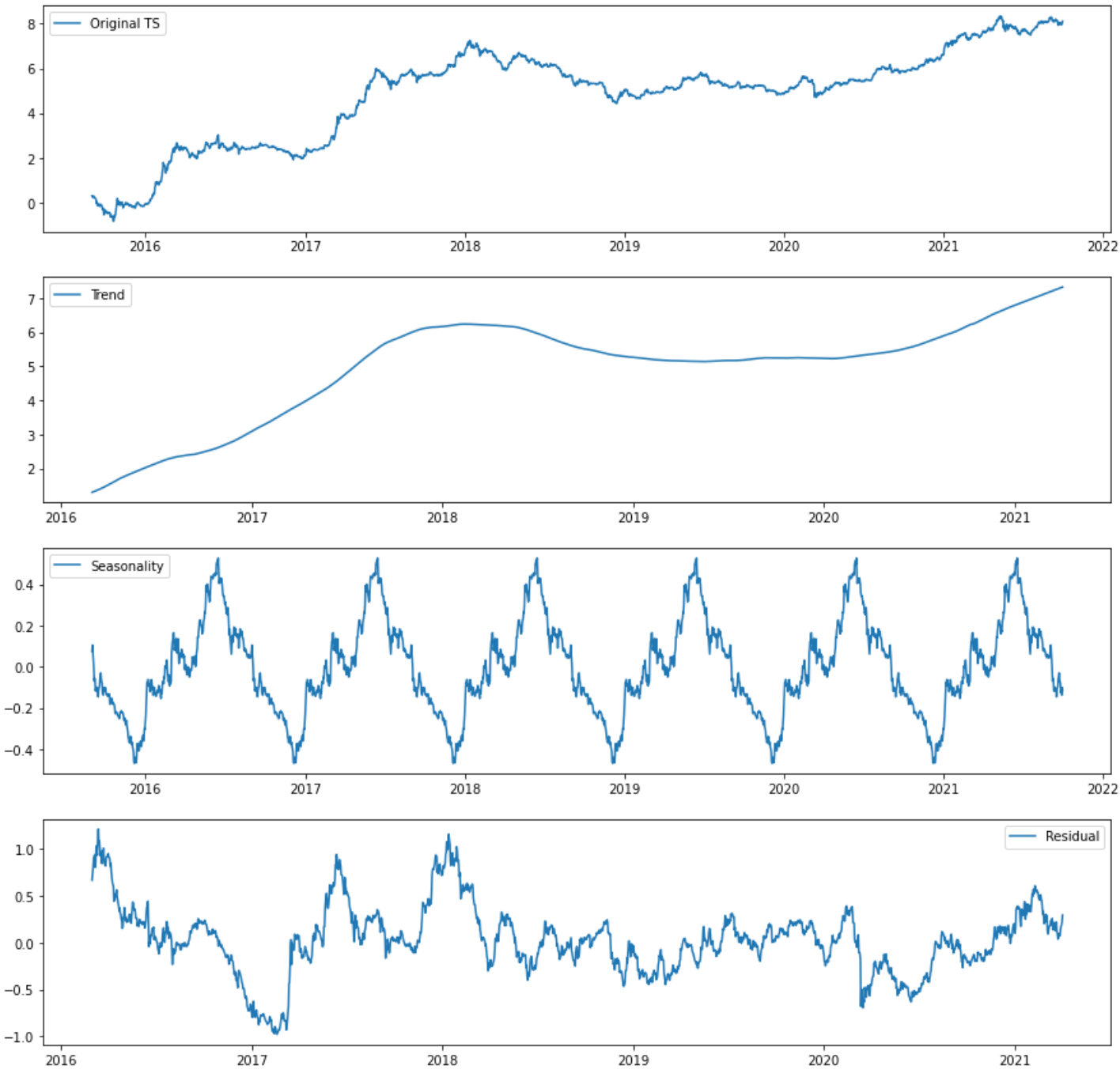
In [92]:

```
decompose_add=seasonal_decompose(tickerDf['logged'], model='additive', period=365)
plt.figure(figsize=(15,15))
plt.subplot(411)
plt.plot(tickerDf['logged'], label='Original TS')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(decompose_add.trend, label='Trend')
```

```
plt.legend(loc='best')
plt.subplot(413)
plt.plot(decompose_add.seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(decompose_add.resid, label='Residual')
plt.legend(loc='best')
```

Out[92]:

```
<matplotlib.legend.Legend at 0x7f9836d1a9d0>
```



In [93]:

```
def relative_ordering(temp,alpha):
    Q = 0
    n = len(temp)
    for i in range(n):
        for j in range(i+1,n):
            if(temp[i] - temp[j] > 0):
                Q+=1
    print('Q = ',Q)
    T=1-4*Q/(n*(n-1))
    VT=2*(2*n+5)/(9*n*(n-1))
    Z=T/(VT**0.5)
    t=norm.ppf(1-alpha/2)
    print('Z = ',Z)
```

```
    print('t_alpha/2 = ',t)
    if(abs(Z) <= t):
        print('So, Trend is not present')
    else:
        print('Trend is present')
```

In [94]:

```
temp=[float(i) for i in tickerDf['logged']]
relative_ordering(temp, 0.05)
```

```
Q =  534532
Z =  39.931689946114595
t_alpha/2 =  1.959963984540054
Trend is present
```

In [95]:

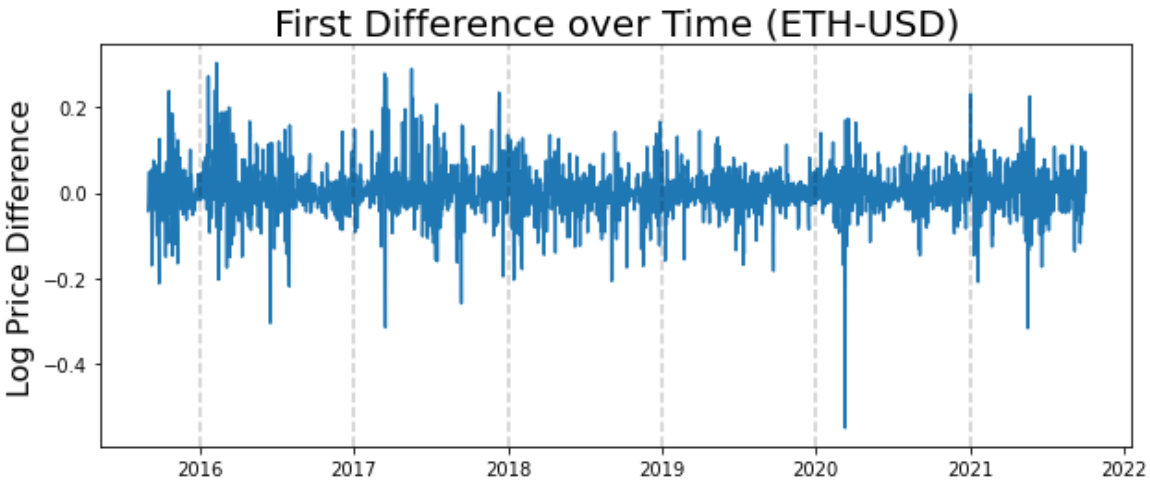
```
#take first difference
first_diffs = tickerDf.logged.values[1:] - tickerDf.logged.values[:-1]
first_diffs = np.concatenate([first_diffs, [0]])
```

In [96]:

```
#set first difference as variable in dataframe
tickerDf['FirstDifference'] = first_diffs
```

In [97]:

```
plt.figure(figsize=(10,4))
plt.plot(tickerDf.FirstDifference)
plt.title('First Difference over Time (%s)'%tickerSymbol, fontsize=20)
plt.ylabel('Log Price Difference', fontsize=16)
for year in range(2016,2022):
    plt.axvline(pd.to_datetime(str(year)+'-01-01'), color='k', linestyle='--', alpha=0.2
)
```



In [98]:

```
#Augmented Dicky-fuller test
from statsmodels.tsa.stattools import adfuller
from numpy import log
result = adfuller(tickerDf['FirstDifference'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

## Data is stationary
```

```
ADF Statistic: -8.666656
p-value: 0.000000
 1%: -3.433
 5%: -2.863
 10%: -2.567
```

In [99]:

```
temp=[float(i) for i in tickerDf['FirstDifference']]
relative_ordering(temp, 0.05)
```

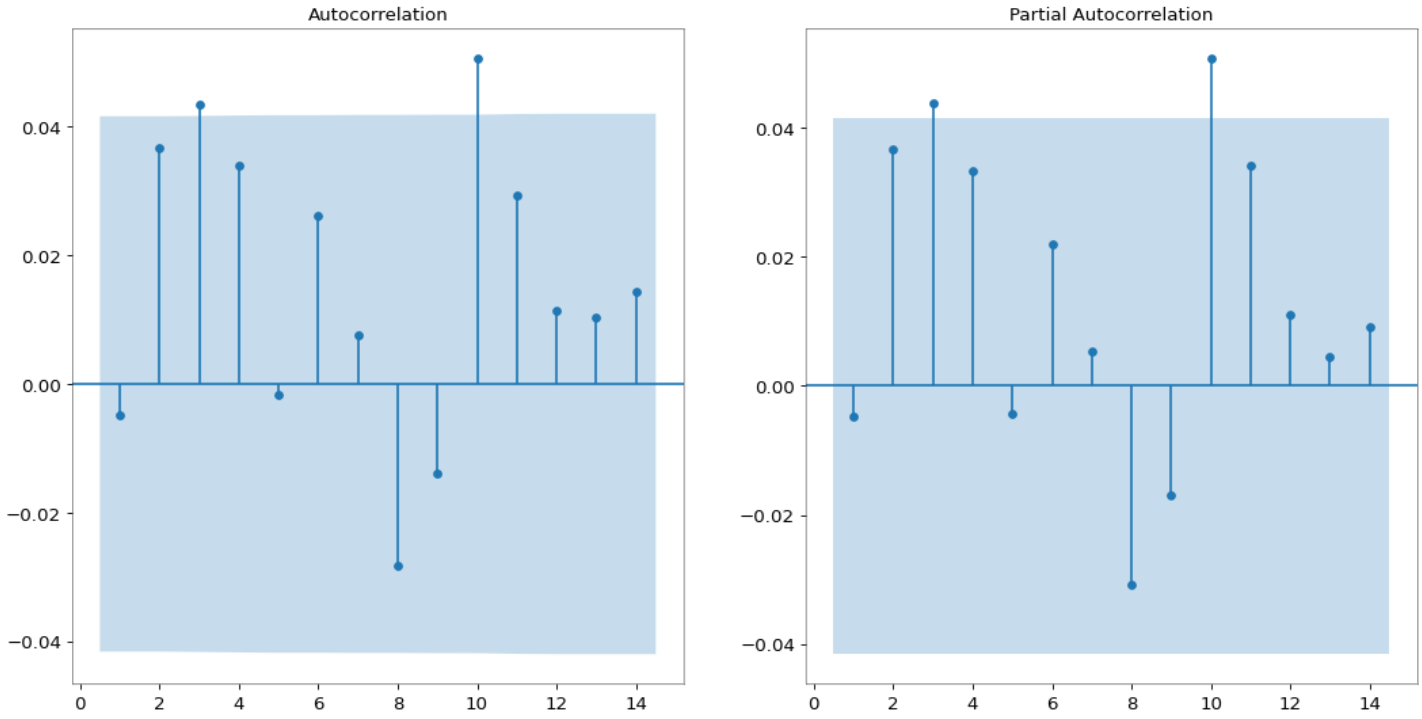
```
Q =  1203633
Z =  1.5379562249374217
t_alpha/2 =  1.959963984540054
So, Trend is not present
```

In [100]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [101]:

```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(16,8), dpi= 80)
acf_plot_diff = plot_acf(tickerDf.FirstDifference, ax=ax1, lags=np.arange(1,15))
pacf_plot_diff = plot_pacf(tickerDf.FirstDifference, ax=ax2, lags = np.arange(1,15), met
hod='ols')
ax1.spines["top"].set_alpha(.3); ax2.spines["top"].set_alpha(.3)
ax1.spines["bottom"].set_alpha(.3); ax2.spines["bottom"].set_alpha(.3)
ax1.spines["right"].set_alpha(.3); ax2.spines["right"].set_alpha(.3)
ax1.spines["left"].set_alpha(.3); ax2.spines["left"].set_alpha(.3)
# font size of tick labels
ax1.tick_params(axis='both', labelsize=12)
ax2.tick_params(axis='both', labelsize=12)
plt.show()
```

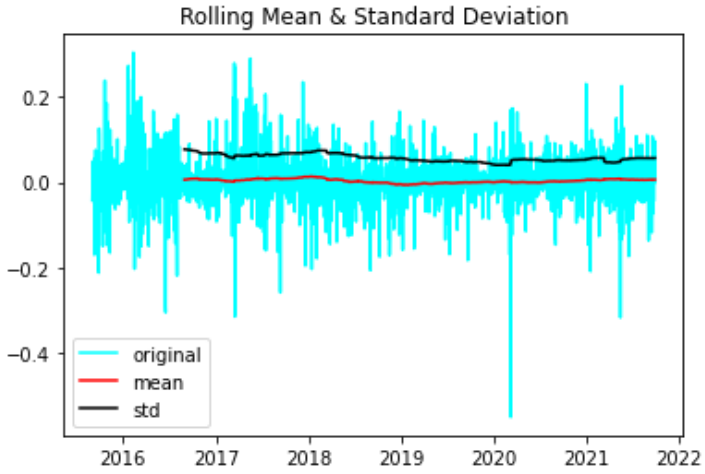


In [102]:

```
def test_stationary(timeseries):
    movingAvg=timeseries.rolling(window=365).mean()
    movingSTD=timeseries.rolling(window=365).std()
    #plotting rolling statistics
    orig=plt.plot(timeseries, color='aqua', label='original')
    mean=plt.plot(movingAvg, color='red', label='mean')
    std=plt.plot(movingSTD, color='black', label='std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
```

In [103]:

```
test_stationary(tickerDf['FirstDifference'])
```



In [104]:

```
from statsmodels.tsa.arima.model import ARIMA
AIC=[]
BIC=[]
for p in range(1,12):
    for q in range(1,12):
        model=ARIMA(tickerDf['FirstDifference'],order=(p,0,q))
        model_fit=model.fit()
        AIC.append(model_fit.aic)
        BIC.append(model_fit.bic)
```

In [122]:

```
np.reshape(AIC,(11,11))
```

Out[122]:

```
array([[-6178.84528552, -6188.89917047, -6183.93427292, -6182.30009313,
        -6180.36535621, -6179.35783608, -6178.00292116, -6175.87456098,
        -6176.87962181, -6182.08269755, -6180.02146027],
       [-6188.94891142, -6183.78046509, -6181.95126777, -6180.36897549,
        -6178.35900787, -6182.11283176, -6183.83380727, -6179.31634688,
        -6174.90791315, -6184.16601873, -6183.5795504 ],
       [-6182.11959424, -6181.63200581, -6179.95728771, -6182.44306713,
        -6180.51085493, -6178.65395723, -6181.59325382, -6177.79827773,
        -6170.64946589, -6181.54302553, -6182.0936314 ],
       [-6182.54122733, -6180.7435382 , -6182.84959296, -6178.76108913,
        -6182.18608252, -6176.35701332, -6174.55397996, -6178.13531345,
        -6173.05727344, -6178.66750224, -6180.00394311],
       [-6180.5857696 , -6182.75371658, -6179.89090186, -6177.64734598,
        -6174.93742931, -6175.14447458, -6172.44479848, -6176.48561281,
        -6176.1243217 , -6182.33856413, -6181.95450153],
       [-6179.64587816, -6177.7004392 , -6183.12110181, -6178.13942339,
        -6178.86751654, -6175.88438514, -6170.3170915 , -6173.59478014,
        -6171.03266538, -6178.19106062, -6175.04595981],
       [-6177.70219546, -6175.7608257 , -6177.59567652, -6174.99316528,
        -6178.28515591, -6170.45269687, -6176.41076871, -6176.12379249,
        -6174.21281905, -6179.11894951, -6173.46586009],
       [-6177.8286461 , -6182.06360458, -6175.01057191, -6177.9810026 ,
        -6178.92972695, -6175.67813086, -6173.24318879, -6174.09193544,
        -6171.00616944, -6182.56097385, -6190.68775814],
       [-6177.79432616, -6177.88465376, -6177.17553016, -6173.8387316 ,
        -6172.0545382 , -6171.02033501, -6170.46494604, -6170.68859578,
        -6170.51427606, -6177.80122774, -6178.883846  ],
       [-6185.01955175, -6185.38855413, -6177.76075907, -6175.94497584,
        -6174.88903139, -6173.64478679, -6175.53571216, -6171.45448873,
        -6178.39582684, -6175.02865197, -6176.69241144],
       [-6180.74280773, -6183.43248678, -6182.27796853, -6173.8767776 ,
        -6177.72402909, -6174.90770609, -6173.30256026, -6171.22789149,
        -6175.6823492 , -6173.36816073, -6174.82142581]])
```

In [106]:

```
print(np.where(x == x.min()))
```

```
(array([7]), array([10]))
```

In [107]:

```
np.reshape(BIC,(11,11))
```

Out[107]:

```
array([[-6156.02603783, -6160.37511085, -6149.70540138, -6142.36640967,
        -6134.72686082, -6128.01452877, -6120.95480193, -6113.12162983,
        -6108.42187874, -6107.92014255, -6100.15409335],
       [-6160.42485181, -6149.55159355, -6142.01758431, -6134.7304801 ,
        -6127.01570056, -6125.06471253, -6121.08087612, -6110.85860381,
        -6100.74535815, -6104.29865181, -6098.00737156],
       [-6147.89072271, -6141.69832235, -6134.31879233, -6131.09975982,
        -6123.4627357 , -6115.90102608, -6113.13551074, -6103.63572273,
        -6090.78209897, -6095.97084669, -6090.81664063],
       [-6142.60754387, -6135.10504281, -6131.50628565, -6121.7129699 ,
        -6119.43315137, -6107.89927024, -6100.39142497, -6098.26794653,
        -6087.4850946 , -6087.39051147, -6083.02214042],
       [-6134.94727422, -6131.41040928, -6122.84278263, -6114.89441483,
        -6106.47968623, -6100.98191958, -6092.57743156, -6090.91343397,
        -6084.84733093, -6085.35676144, -6079.26788692],
       [-6128.30257085, -6120.65231997, -6120.36817066, -6109.68168031,
        -6104.70496155, -6096.01701822, -6084.74491265, -6082.31778937,
        -6074.05086269, -6075.50444601, -6066.65453327],
       [-6120.65407623, -6113.00789455, -6109.13793344, -6100.83061028,
        -6098.41778898, -6084.88051802, -6085.13377794, -6079.1419898 ,
        -6071.52620444, -6070.72752297, -6059.36962163],
       [-6115.07571494, -6113.6058615 , -6100.84801691, -6098.11363568,
        -6093.35754811, -6084.40114009, -6076.2613861 , -6071.40532082,
        -6062.61474291, -6068.46473539, -6070.88670776],
       [-6109.33658309, -6103.72209877, -6097.30816324, -6088.26655276,
        -6080.77754743, -6074.03853232, -6067.77833143, -6062.29716924,
        -6056.4180376 , -6058.00017736, -6053.37798369],
       [-6110.85699675, -6105.52118721, -6092.18858022, -6084.66798507,
        -6077.9072287 , -6070.95817217, -6067.14428563, -6057.35825028,
        -6058.59477646, -6049.52278967, -6045.48173721],
       [-6100.87544081, -6097.86030794, -6091.00097776, -6076.89497491,
        -6075.03741448, -6066.51627956, -6059.2063218 , -6051.42684111,
        -6050.1764869 , -6042.1574865 , -6037.90593966]])
```

In [124]:

```
final=ARIMA(tickerDf,order=(8,1,11))
final=final.fit()
```

In [125]:

```
final.summary()
```

Out[125]:

SARIMAX Results

Dep. Variable:	logged	**No. Observations:**	2219
Model:	ARIMA(8, 1, 11)	**Log Likelihood**	3112.687
Date:	Wed, 17 Nov 2021	**AIC**	-6185.374
Time:	02:16:28	**BIC**	-6071.287
Sample:	0	**HQIC**	-6143.703
	- 2219		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3019	0.172	-1.754	0.080	-0.639	0.036
ar.L2	0.4448	0.147	3.031	0.002	0.157	0.732

ar.L3	0.3820	0.168	2.268	0.023	0.052	0.712
ar.L4	0.2759	0.182	1.517	0.129	-0.081	0.632
ar.L5	0.1886	0.179	1.052	0.293	-0.163	0.540
ar.L6	0.1981	0.163	1.214	0.225	-0.122	0.518
ar.L7	-0.0128	0.139	-0.092	0.927	-0.286	0.260
ar.L8	-0.4560	0.130	-3.506	0.000	-0.711	-0.201
ma.L1	0.2972	0.172	1.724	0.085	-0.041	0.635
ma.L2	-0.4112	0.149	-2.759	0.006	-0.703	-0.119
ma.L3	-0.3228	0.164	-1.963	0.050	-0.645	-0.001
ma.L4	-0.2564	0.181	-1.416	0.157	-0.611	0.098
ma.L5	-0.2211	0.175	-1.262	0.207	-0.565	0.122
ma.L6	-0.2052	0.159	-1.295	0.195	-0.516	0.105
ma.L7	-0.0148	0.135	-0.109	0.913	-0.280	0.251
ma.L8	0.4171	0.126	3.299	0.001	0.169	0.665
ma.L9	-0.0587	0.023	-2.532	0.011	-0.104	-0.013
ma.L10	0.0873	0.023	3.847	0.000	0.043	0.132
ma.L11	0.0932	0.027	3.467	0.001	0.041	0.146
sigma2	0.0035	5.91e-05	59.745	0.000	0.003	0.004

Ljung-Box (L1) (Q):	0.03	**Jarque-Bera (JB):**	3997.82
Prob(Q):	0.87	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.58	**Skew:**	-0.28
Prob(H) (two-sided):	0.00	**Kurtosis:**	9.55

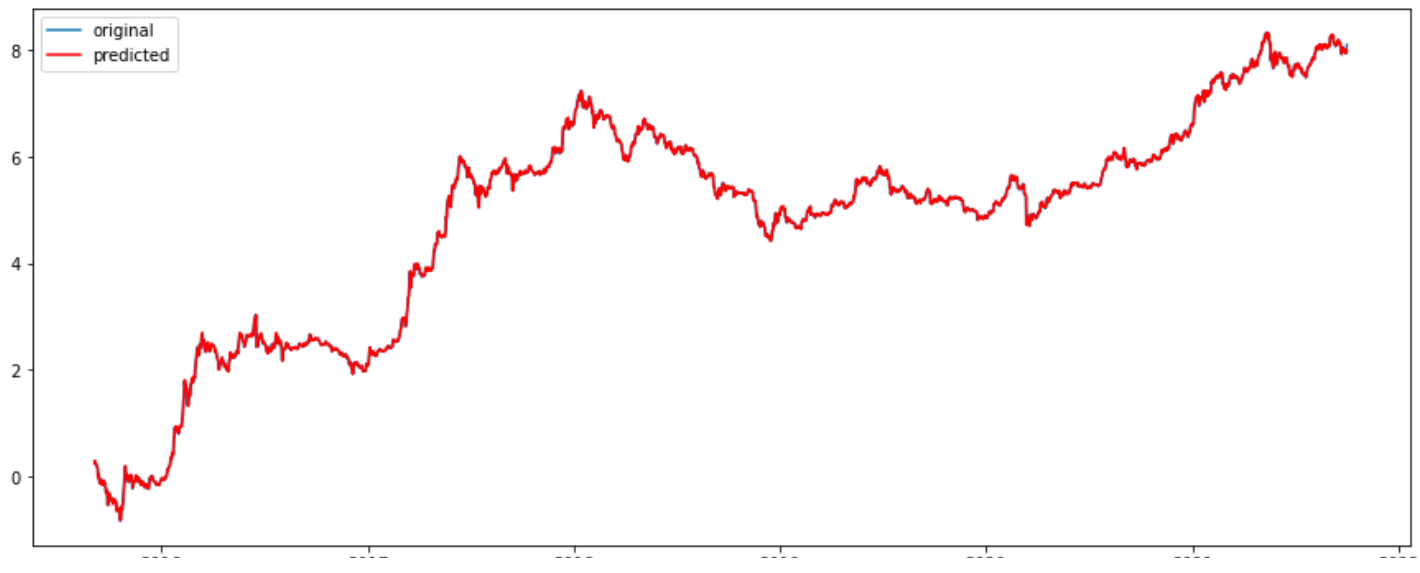
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [126]:

```
pred=final.predict()
```

In [127]:

```
plt.figure(figsize=(15,6))
plt.plot(tickerDf[4:] , label='original')
plt.plot(pred[4:], color='red', label='predicted')
plt.legend(loc='best', prop={'size': 10})
plt.show()
```



In [128]:

```
forecast= final.forecast(steps=10)
forecast
```

Out[128]:

```
2219    8.105358
2220    8.115314
2221    8.112732
2222    8.108238
2223    8.107634
2224    8.106199
2225    8.100470
2226    8.094526
2227    8.091902
2228    8.097686
Name: predicted_mean, dtype: float64
```

In [129]:

```
import math
forecast_data = list(forecast)
forecast_data1 = []
for ele in forecast_data:
    forecast_data1.append(math.exp(ele))
```

In [130]:

```
for ele in forecast_data1:
    print(ele)
```

```
3312.1669098132747
3345.309455990676
3336.680254911625
3321.7200143952005
3319.713968019474
3314.9542023633553
3296.016329827274
3276.4829813090882
3267.8972047825905
3286.8524687978957
```

In [131]:

```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return(np.mean(np.abs((y_true - y_pred) / y_true)) * 100)
```

In [132]:

```
org = tickerData.history(period='1d', start='2021-10-1', end='2021-10-10')
```

In [133]:

```
org_data = list(org['Close'])
```

In [134]:

```
org_data
```

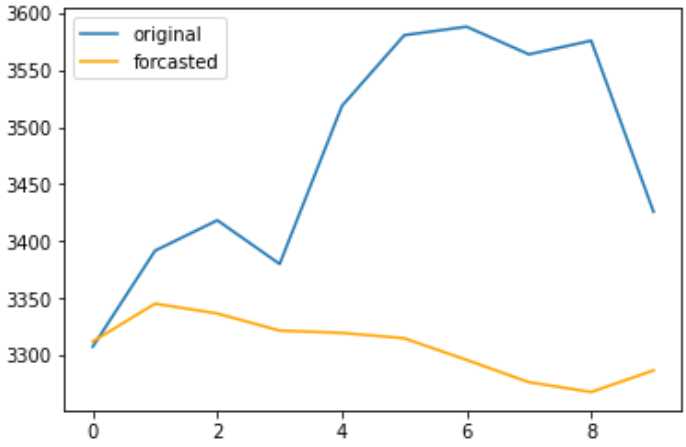
Out[134]:

```
[3307.51611328125,
 3391.6943359375,
 3418.358642578125,
 3380.089111328125,
 3518.5185546875,
 3580.56201171875,
 3587.974853515625,
```

```
 3563.75927734375,
 3575.716796875,
 3425.852783203125]
```

In [135]:

```
plt.plot(org_data , label='original')
plt.plot(forecast_data1, color='orange', label='forcasted')
plt.legend(loc='best', prop={'size': 10})
plt.show()
```



In [136]:

```
mean_absolute_percentage_error(org_data, forecast_data1)
```

Out[136]:

```
4.755694460288888
```