Crypto Signals: Identifying the Best-Fit ARIMA Model for Bitcoin Index

Mohammad Wasim Ashraf

```
knitr::opts_chunk$set(
  echo    = TRUE,
  warning = FALSE,
  message = FALSE
)
```

INTRODUCTION

This report focuses on fitting models on stationary series. We will understand on how the presence of trend in a series can impact the stationarity of the series. We will also investigate how to make a non-stationary series into a stationary one. We will use transformation tools to tackle the change in variance in the series. After converting the non-stationary series into stationary series will use ARIMA(p,d,q) model to fit in the time series. The main objective of the report is to find the best fitting model in a time series. For this purpose, we need to find the possible set models. To find the possible set of models we will seek help from the following tools: 1. ACF Plot 2. PACF Plot 3. EACF Plot 4. BIC Table Using the above tools, we will get the possible set of models, that can be fitted in our series. To find the best fitted model out of these possible set of models, we will check the significance of estimate coefficients, AIC values of the fitted models, BIC values of the fitted models and error measures of the fitted models. After completing the above test, hopefully we will get the best fitted model for our series.

Let's dig deeper and complete the analysis to find the best-fitted model.

```
rm(list=ls())
library(TSA)
library(tseries)
library(forecast)
library(lmtest)
```

LOADING THE DATA

```
bitind <- read.csv("assignment2Data2025.csv", header=TRUE)
```

DESCRIPTIVE ANALYSIS

The data provided gives us an insight into monthly historical performance of bitcoin index in USD.

Data has 2 columns namely, Date and Bitcoin. The "Date" column has the month and year, and the "Bitcoin" column contains the bitcoin index for that month and year.
The data contains 162 observations from August 2011 to January 2025.

The minimum bitcoin index in USD is 3988 and the maximum bitcoin index is 128015000. The mean of the bitcoin index is 19781743 and the median of the bitcoin index is 7930612. This gives an insight that there is skewness in the data.

Now, let's forward to plot the time series plot of the data. For this aim, we first need to convert our data into a "ts" (Time Series) object. We take frequency = 12, because this is a monthly data.

```
summary(bitind)
```

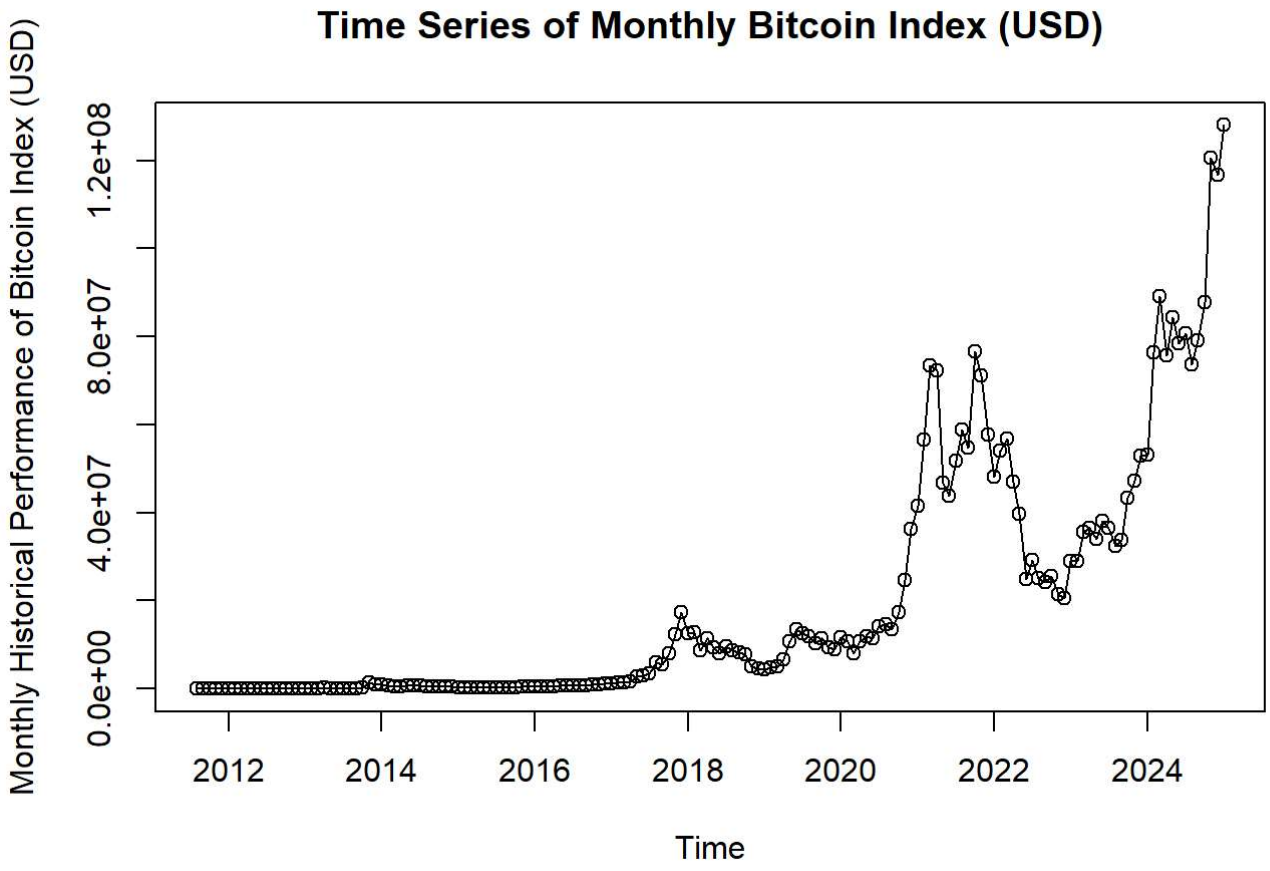
```
##      Date              Bitcoin         
##  Length:162         Min.   :     3988  
##  Class :character   1st Qu.:   470981  
##  Mode  :character   Median :  7930612  
##                     Mean   : 19781743  
##                     3rd Qu.: 31599541  
##                     Max.   :128015000
```

TIME SERIES PLOT OF RAW DATA

1. Trend Observing the time series plot, we can say that there is an upward quadratic trend in the time series plot.
2. Seasonality Seasonality means repeating patterns in the time series plot. There are no repeating patterns. Therefore, there is no seasonality in the time series plot.
3. Change in variance Looking closely at the time series plot, We can see change in variance during 2018 to 2020 and during 2020 to 2022. Therefore, we have change in variance in our time series plot.
4. Change Point/Intervention We can observe a change point around 2021. This was the time of Covid. It may be possible that the Covid might have impacted Bitcoin index.
5. Behaviour In our Time series plot, we have succeeding points initially, and some fluctuations at the end. Succeeding time points indicate Autoregressive behaviour and fluctuations indicate moving average behaviour. Therefore, the time series plot has a mix of both Autoregressive and moving average behaviour.

Therefore, an ARMA (Auto regressive Moving Average) model will best fit this time series. Important point to note here is that ARMA models are fitted into a stationary series. Stationary series means there should be no trend in our series. As discussed above, we have an upward quadratic trend in our time series plot. Therefore, we cannot fit a ARMA model. However, we can confirm the non-stationarity of the series using ACF (Auto Correlation Function) and PACF (Partial Auto Correlation Function) plots as well. If there is a slowly decaying pattern in ACF plot and high first lag in PACF plot that means we have a non-stationary series. We can also confirm non-stationarity through various hypothesis tests namely, ADF test, pp test and kpss test. We can eliminate trend in our series by differencing. As we have change in variance in our series, we need to tackle that too. We can eliminate change in variance by transformation. We have two types of transformations, 1. Log Transformation, and 2. Box-Cox Transformation. Let's have look at the ACF and PACF plot of our series.

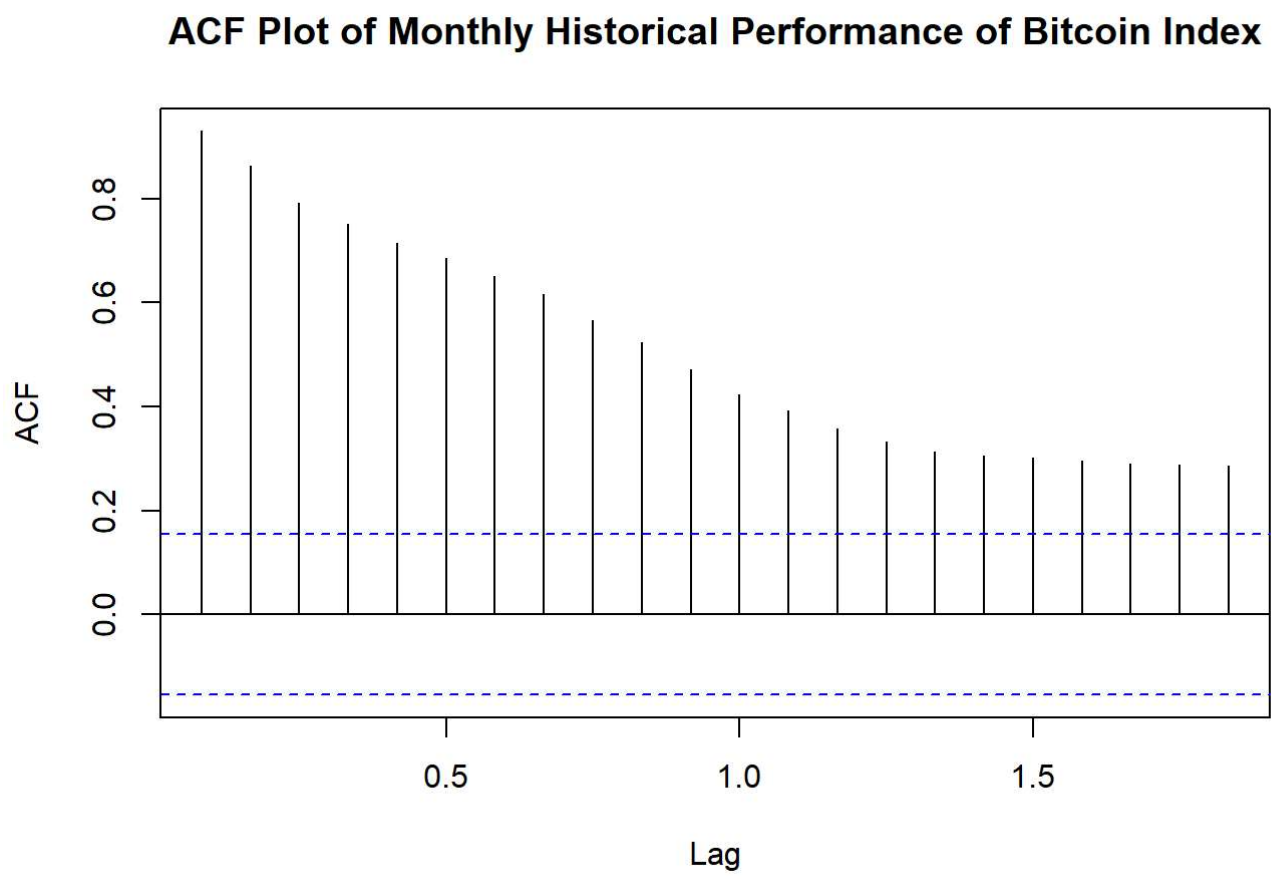
```
bitindTS <- ts(bitind$Bitcoin, start = c(2011, 8), end = c(2025, 1), frequency = 12)
plot(bitindTS, ylab="Monthly Historical Performance of Bitcoin Index (USD)", xlab="Time", type
="o",
     main = "Time Series of Monthly Bitcoin Index (USD)")
```



ACF PLOT OF RAW DATA

We can clearly see the slowly decaying pattern in the ACF plot. Slowly decaying pattern indicates the presence of rend in the series. Therefore, according to ACF plot the raw time series is non-stationary.

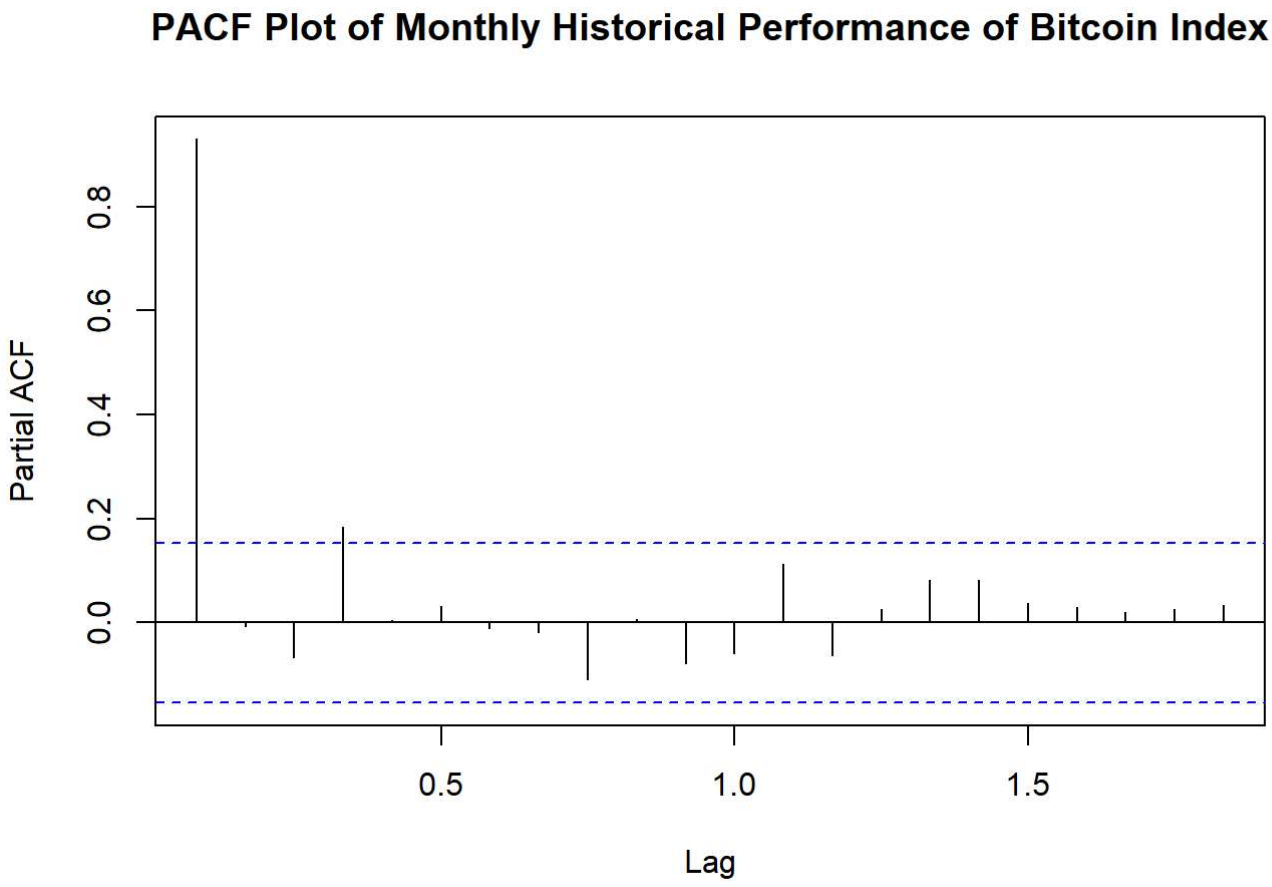
```
acf(bitindTS, main="ACF Plot of Monthly Historical Performance of Bitcoin Index")
```



PACF PLOT OF RAW DATA

we can clearly see high first lag, which is around 0.9. High first lag means the series is non-stationary.

```
pacf(bitindTS, main="PACF Plot of Monthly Historical Performance of Bitcoin Index")
```



HYPOTHESIS TESTS

1. ADF (Augmented Dickey-Fuller Test) unit-root test H0 = The process is difference nonstationary (the process is nonstationary but becomes stationary after first differencing). HA = The process is stationary.

p-value of ADF test is 0.99, which is greater than the significance level (α = 0.05). Therefore, the series is non-stationary.

```
adf.test(bitindTS)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  bitindTS
## Dickey-Fuller = -0.27056, Lag order = 5, p-value = 0.99
## alternative hypothesis: stationary
```

2. Phillips-Perron (pp) unit root test H0 = The process is difference nonstationary (the process is nonstationary but becomes stationary after first differencing). HA = The process is stationary.

p-value of pp (Phillips-Perron) test is 0.9557, which is greater than the significance level (α = 0.05). Therefore, the series is non-stationary.

```
pp.test(bitindTS)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  bitindTS
## Dickey-Fuller Z(alpha) = -2.4275, Truncation lag parameter = 4, p-value
## = 0.9557
## alternative hypothesis: stationary
```

From the ACF and PACF plot we concluded that our raw time series is non-stationary. Upon conducting hypothesis test namely ADF and pp test, we got the time series as non-stationary in both test results.

TRANSFORMATION

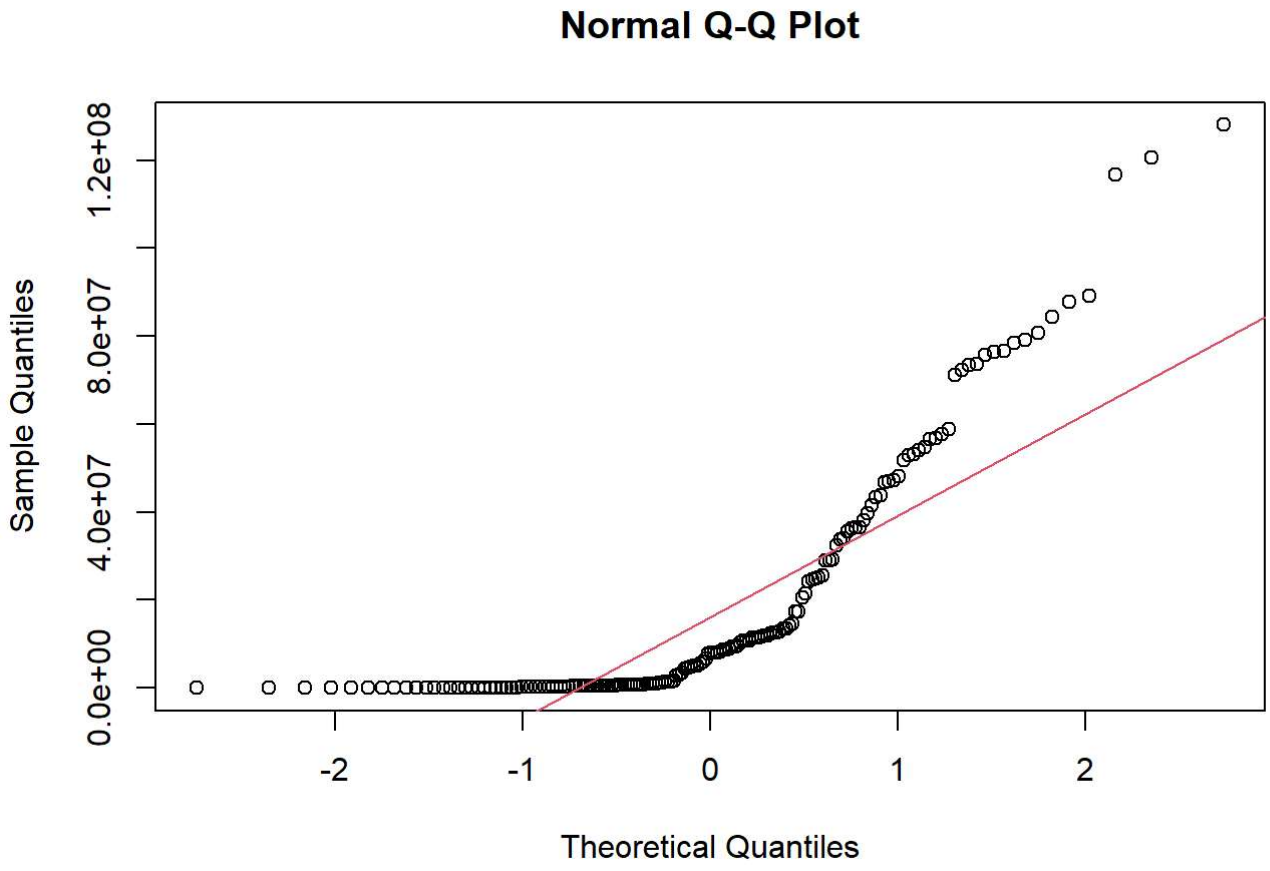
To eliminate change in variance we use transformation. There are two types of transformation transformations that we can use, one is log transformation and the other is Box-Cox Transformation. Let's apply both transformation one by one and choose the one which reduces the variation best and improves the normality of the data.

Before applying transformation, let us have a look at the distribution of our raw data.

1. Q-Q Plot

we see that the data points are deviated from the red reference line at both the tails (Bottom and Top tails). The data points do not stick to the reference line in the middle too. Confirming that the raw data is not normal.

```
qqnorm(bitindTS)
qqline(bitindTS, col = 2)
```



2. Shapiro-Wilk Test

The p-value of the Shapiro-Wilk test is 1.102e-15. The p-value is less than the significance level (α = 0.05), Therefore, the data is not normal.

```
shapiro.test(bitindTS)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bitindTS
## W = 0.73793, p-value = 1.102e-15
```

Now let's apply transformation.

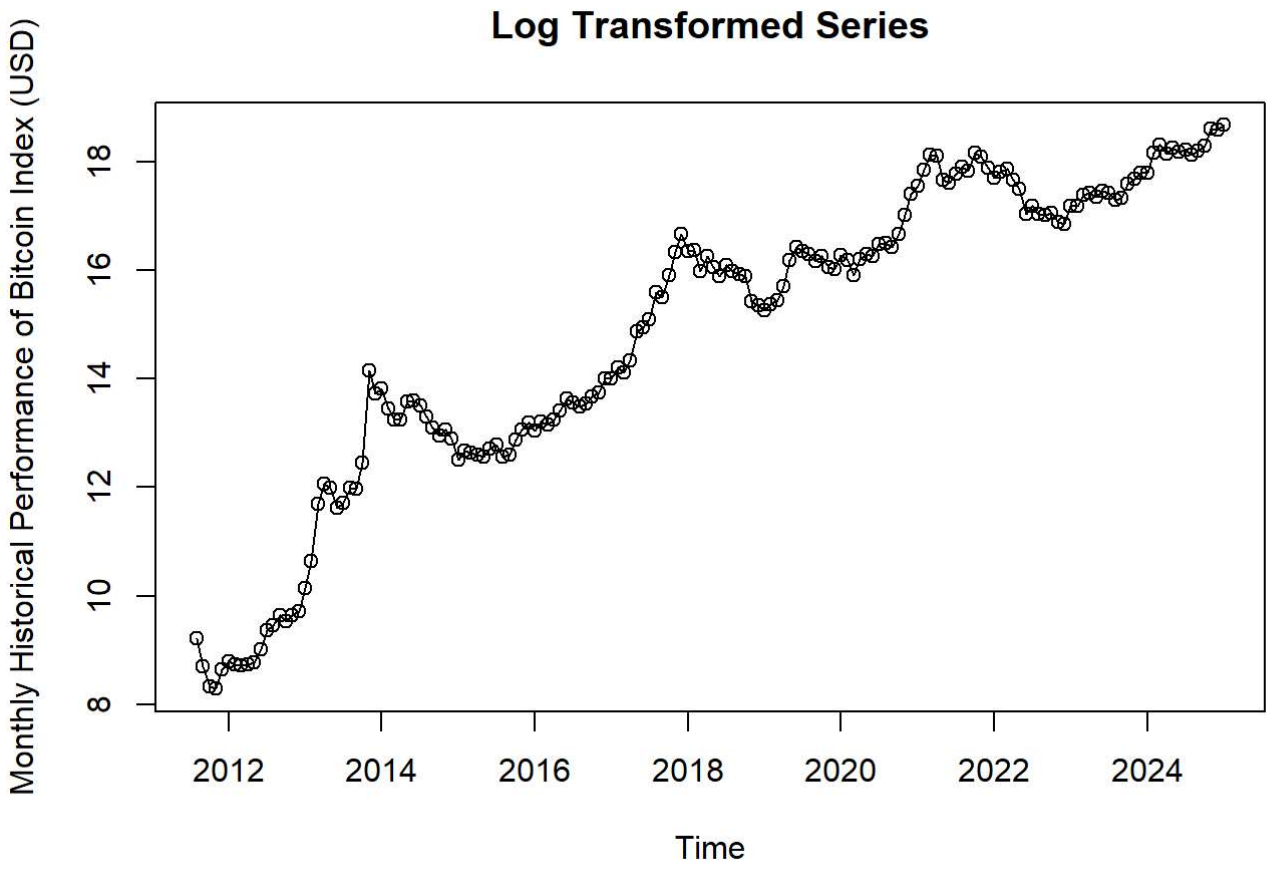
1. Natural Log Transformation

Here we can observe that there has been significant change in variation the series as compared with the raw data series. Now, Let's discuss the 5 bullet points.

1. Trend In the time series plot, We can see an upward linear trend.
2. Seasonality Seasonality means repeating patterns in the time series plot. There are no repeating patterns. Therefore, there is no seasonality in the time series plot.
3. Change in variance After applying natural log transformation, there is a huge reduction in change in variance. In the raw data series, we had a whopping change in variance during 2018 to 2020 and during 2020 to 2022. Log transformation has reduced the variation effectively.
4. Change Point/Intervention In the time series of raw data, we observed a change point around year 2021. Due to transformation and reduced variation, that change point is not noticeable anymore. Therefore, in the log transformed series we do not have a change point/intervention.
5. Behaviour We observe mostly succeeding time points and few fluctuations as well. Therefore, it is a mix of both Autoregressive and moving average behaviour.

```
bitindTSlog <- log(bitindTS)

plot(bitindTSlog, ylab="Monthly Historical Performance of Bitcoin Index (USD)", xlab="Time", mai
n="Log Transformed Series", type="o")
```

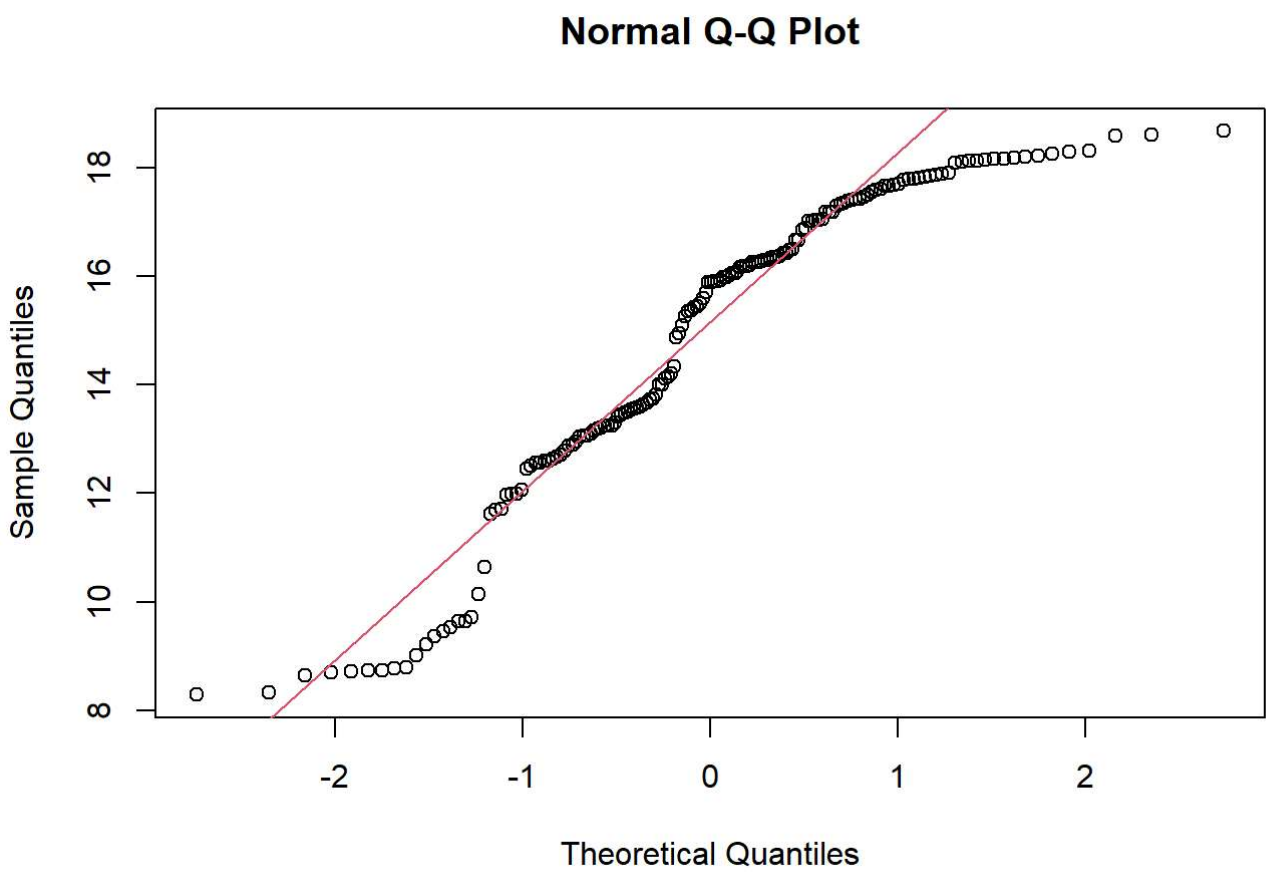


NORMALITY CHECK FOR LOG TRANSFORMED SERIES

1. Q-Q PLOT

The time points are deviated from the reference line from the tails (bottom and top tails). However, in the middle the data points somewhat stick to the reference line, indicating slight normality. Log transformed series is better than the Q-Q plot of the raw series in terms of data points being near to the reference line, indicating that the log transformed series is more normal than the raw series.

```
qqnorm(bitindTSlog)
qqline(bitindTSlog, col = 2)
```



2. SHAPIRO-WILK TEST

The p-value of the Shapiro-Wilk test is 3.363e-08. The p-value is less than the significance level (α = 0.05), Therefore, the data is not normal.

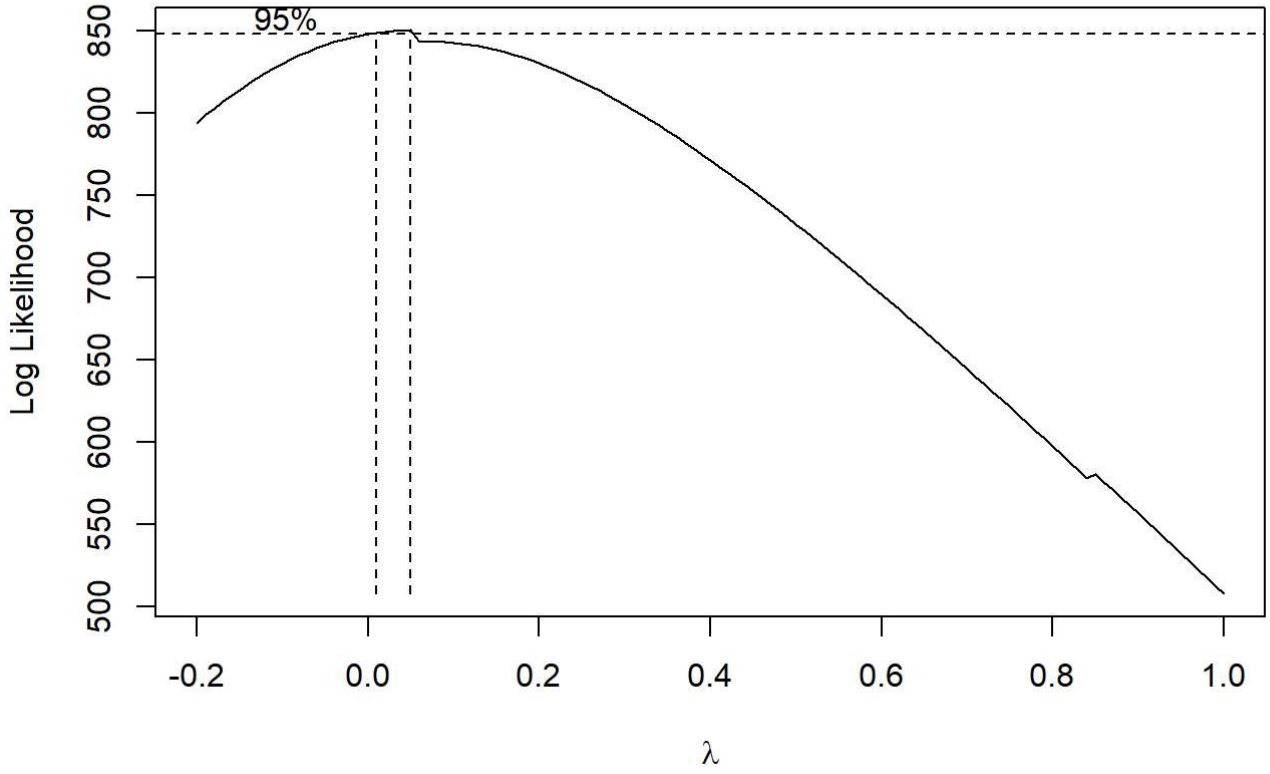
```
shapiro.test(bitindTSlog)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  bitindTSlog
## W = 0.9138, p-value = 3.363e-08
```

2. Box-Cox Transformation

we get confidence interval for Box-Cox transformation as 0.01 to 0.05.

```
BC <- BoxCox.ar(y=bitindTS, lambda = seq(-0.2, 1, 0.01))
```



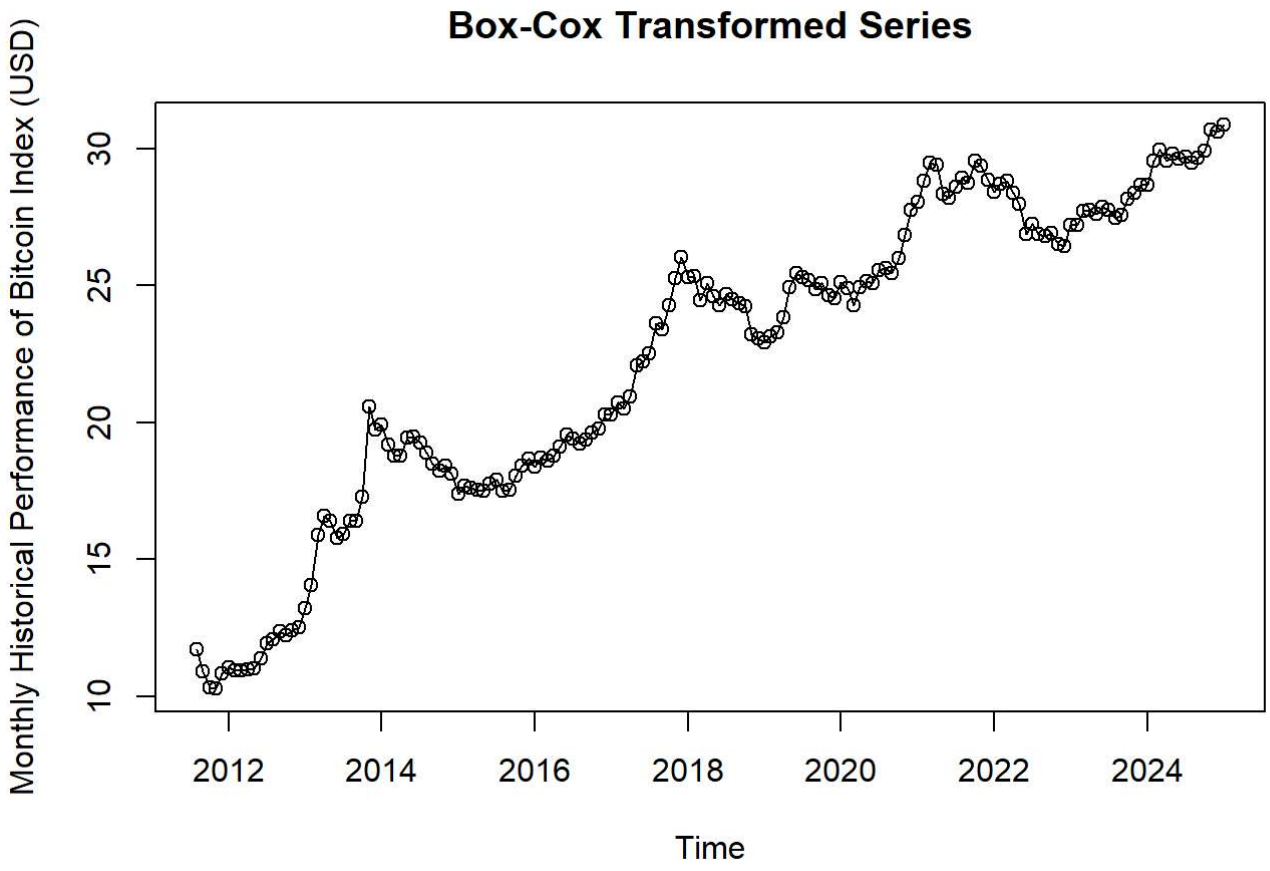

```
BC$ci
```

```
## [1] 0.01 0.05
```

```
lambda <- BC$lambda[which(max(BC$loglike) == BC$loglike)]
lambda
```

```
## [1] 0.05
```

```
bitindTSBC <- ((bitindTS^lambda) - 1) / lambda
plot(bitindTSBC, ylab = "Monthly Historical Performance of Bitcoin Index (USD)", xlab = "Time",
     main = "Box-Cox Transformed Series", type = "o")
```




```
BC$ci
```

```
## [1] 0.01 0.05
```

This gives us the confidence interval as well as the optimal value of lambda. The confidence interval we get is 0.01 to 0.05 and the optimal value of lambda is 0.05. Now, we put the value of optimal lambda in the formula of Box-Cox transformation and plot the Box-Cox transformed series.

```
lambda <- BC$lambda[which(max(BC$loglike) == BC$loglike)]
lambda
```

```
## [1] 0.05
```

After finding the optimal value of lambda, we fit that value in the Box-Cox transformation. The next step will be to plot the Box-Cox transformed series.

The Box-Cox series is very similar to log transformed series. There is no noticeable change. Let's discuss the 5 bullet points for Box-Cox transformed series.

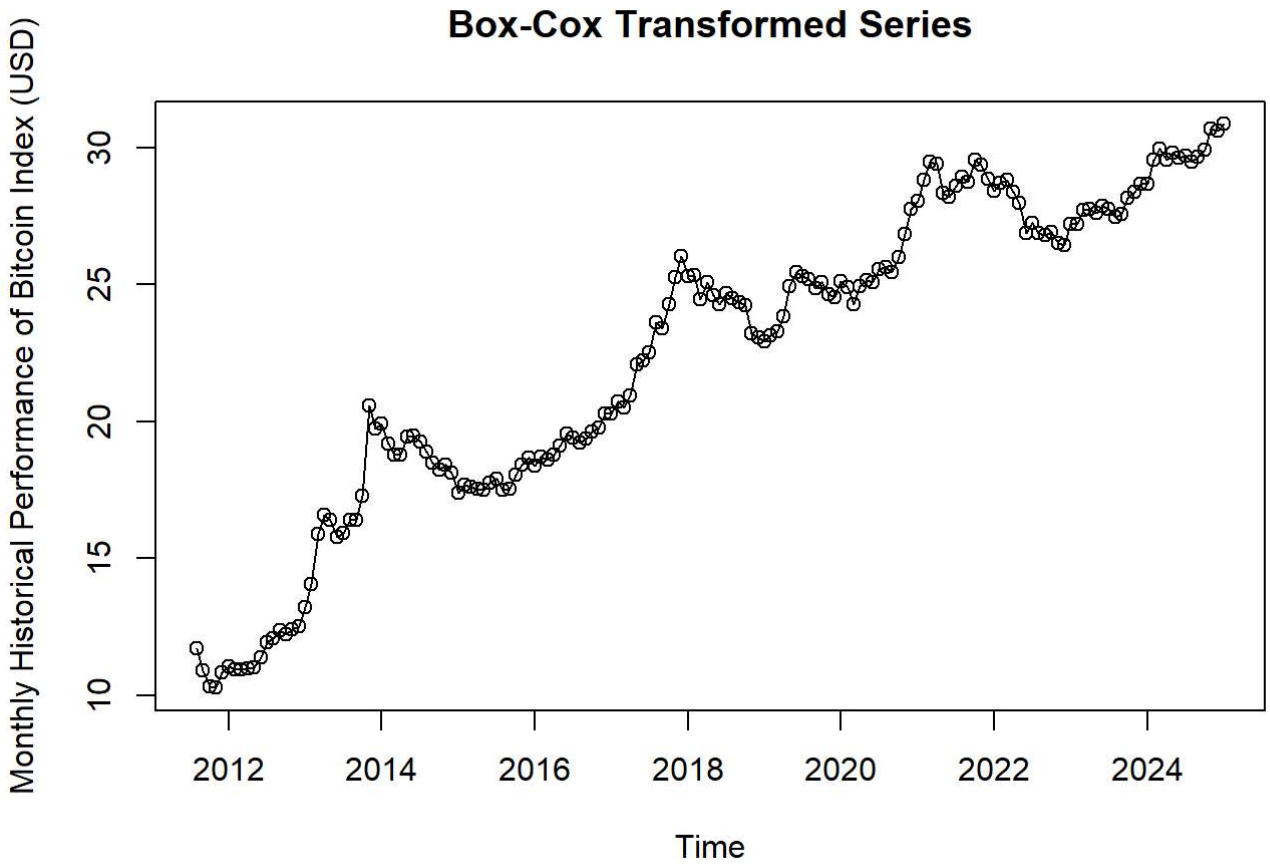
1. Trend In the Box-Cox transformed series, We can see an upward linear trend in the Box-Cox transformed time series.
2. Seasonality Seasonality means repeating patterns in the time series plot. There are no repeating patterns. Therefore, there is no seasonality in the Box-Cox transformed time series plot.

3. Change in variance After applying in the Box-Cox transformation, there is a huge reduction in change in variance. In the raw data series, we had a whopping change in variance during 2018 to 2020 and during 2020 to 2022. Box-Cox transformation has reduced the variation effectively.

4. Change Point/Intervention In the time series of raw data, we observed a change point around year 2021. Due to transformation and reduced variation, that change point is not noticeable anymore. Therefore, in the Box-Cox transformed series, we do not have a change point/intervention.

5. Behaviour We observe mostly succeeding time points and few fluctuations as well. Therefore, it is a mix of both Autoregressive and moving average behaviour.

```
bitindTSBC <- ((bitindTS^lambda) - 1) / lambda
plot(bitindTSBC, ylab = "Monthly Historical Performance of Bitcoin Index (USD)", xlab = "Time",
     main = "Box-Cox Transformed Series", type = "o")
```

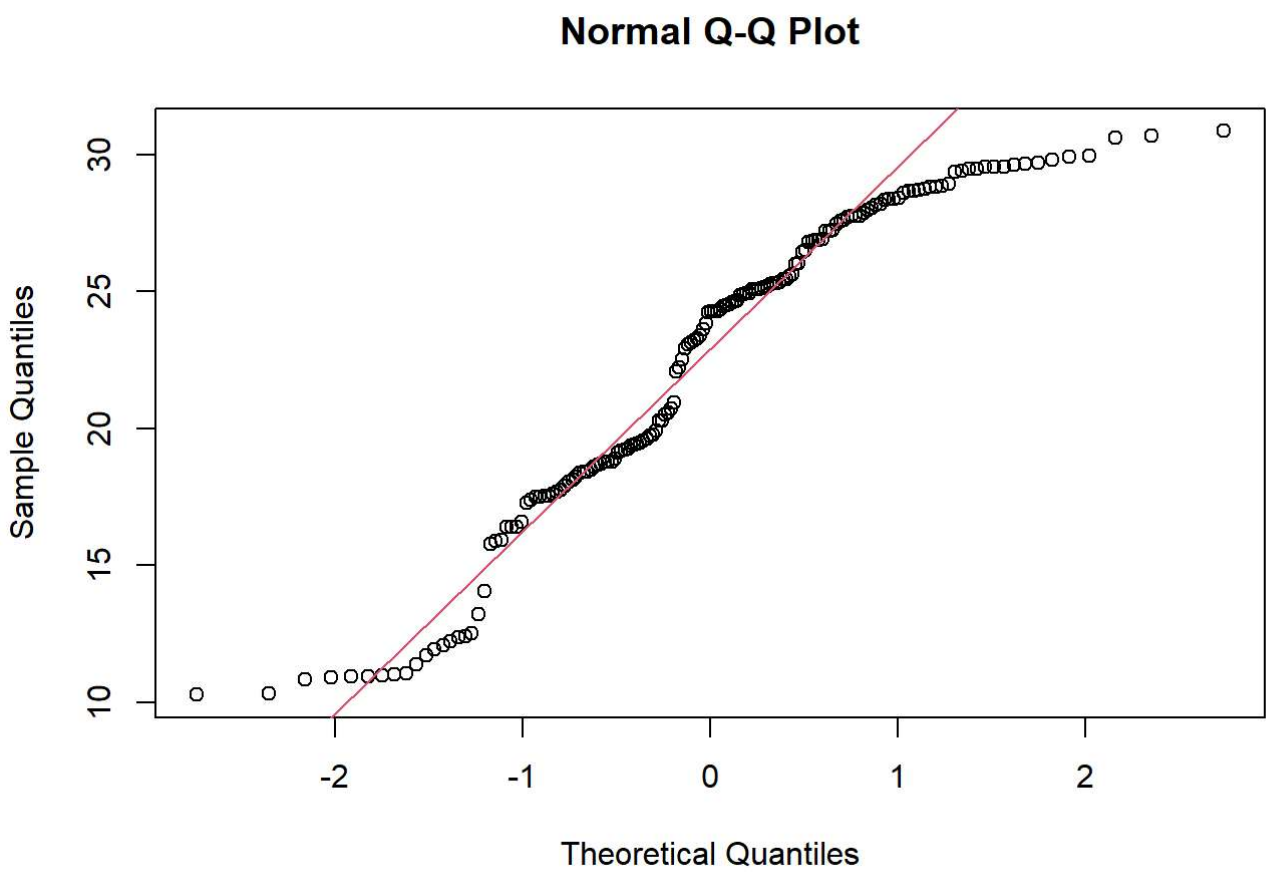


NORMALITY CHECK FOR BOX-COX TRANSFORMED SERIES

1. Q-Q Plot

The time points are deviated from the reference line from the tails (bottom and top tails). However, in the middle the datapoints somewhat stick to the reference line, indicating slight normality. The Q-Q plot of Box-Cox transformed is same as the log transformed series in terms of data points being near to the reference line.

```
qqnorm(bitindTSBC)
qqline(bitindTSBC, col = 2)
```



2. Shapiro-Wilk test

The p-value of the Shapiro-Wilk test is 6.461e-07. The p-value is less than the significance level (α = 0.05), Therefore, the Box-Cox Transformed Series is not normal.

```
shapiro.test(bitindTSBC)
```

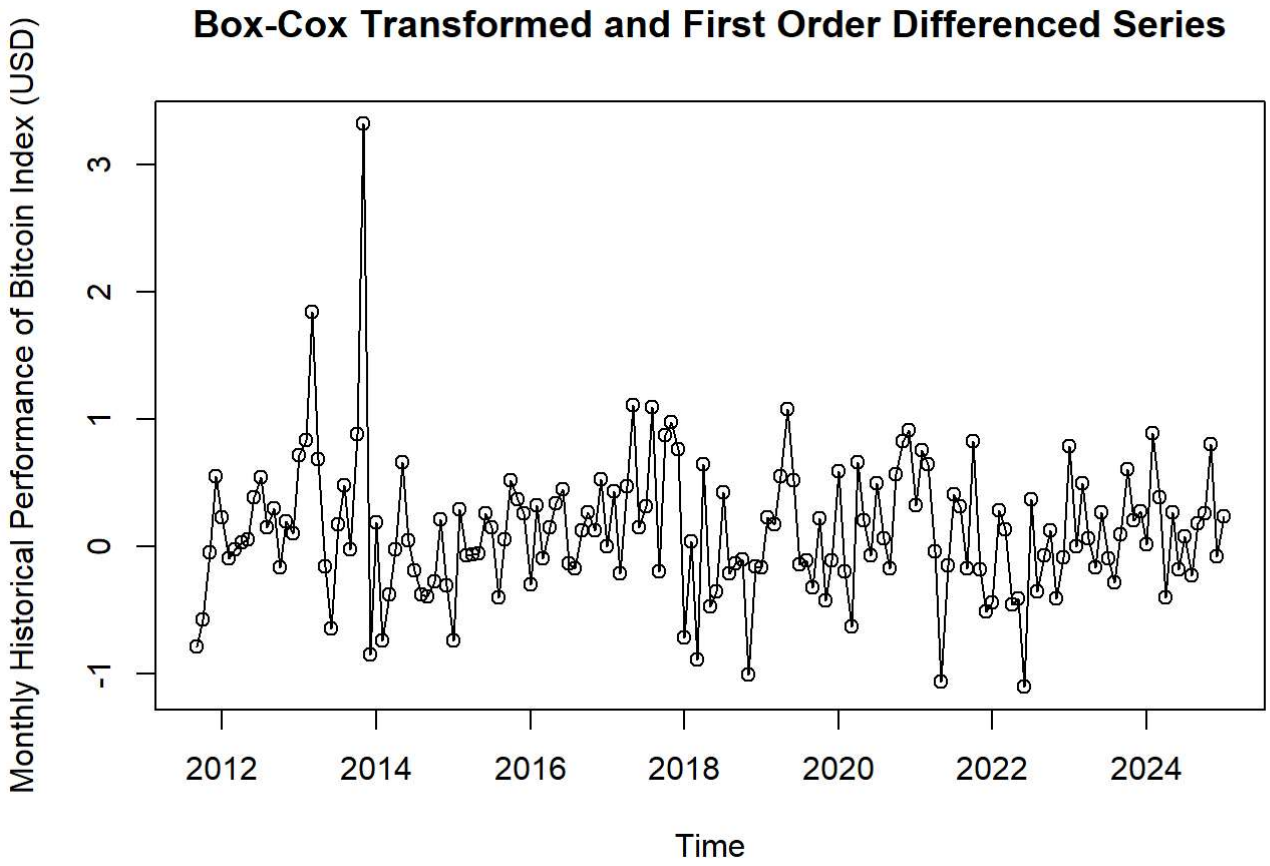
```
##
##  Shapiro-Wilk normality test
##
## data:  bitindTSBC
## W = 0.93252, p-value = 6.461e-07
```

Now, let's answer, which transformation to choose for our analysis. We can choose either of the series, since there is no change in the series and in normality. However, we can choose Box-Cox transformation over log transformation, because in Box-Cox transformation we use optimal value of lambda to transform the series. Therefore, the results for Box-Cox will be more reliable and accurate. **Hence, we will now use Box-Cox Transformed series for the rest of the analysis.**

DIFFERENCING

Differencing is a tool that converts non-stationary series into a stationary series. If Yt is a non-stationary series, the differenced series ∇ Yt = Yt - Yt-1 is stationary. This will be the first order differencing. If the first differenced series is not stationary, then we can go for second order and so on. As we concluded earlier that we will be using Box-Cox transformed series, therefore applying differencing to the Box-Cox transformed series.

```
bitindTSBCdif <- diff(bitindTSBC, differences = 1)
plot(bitindTSBCdif, ylab = "Monthly Historical Performance of Bitcoin Index (USD)", xlab = "Tim
e", main = "Box-Cox Transformed and First Order Differenced Series", type = "o")
```



The above time series plot captures the Box-Cox transformed with First Order differenced series. We can observe that the series is fluctuating at a mean level, indicating stationarity. However, we can confirm non-stationarity by discussing trend and looking at the ACF and PACF plot as well.

Let's discuss the 5 bullet points.

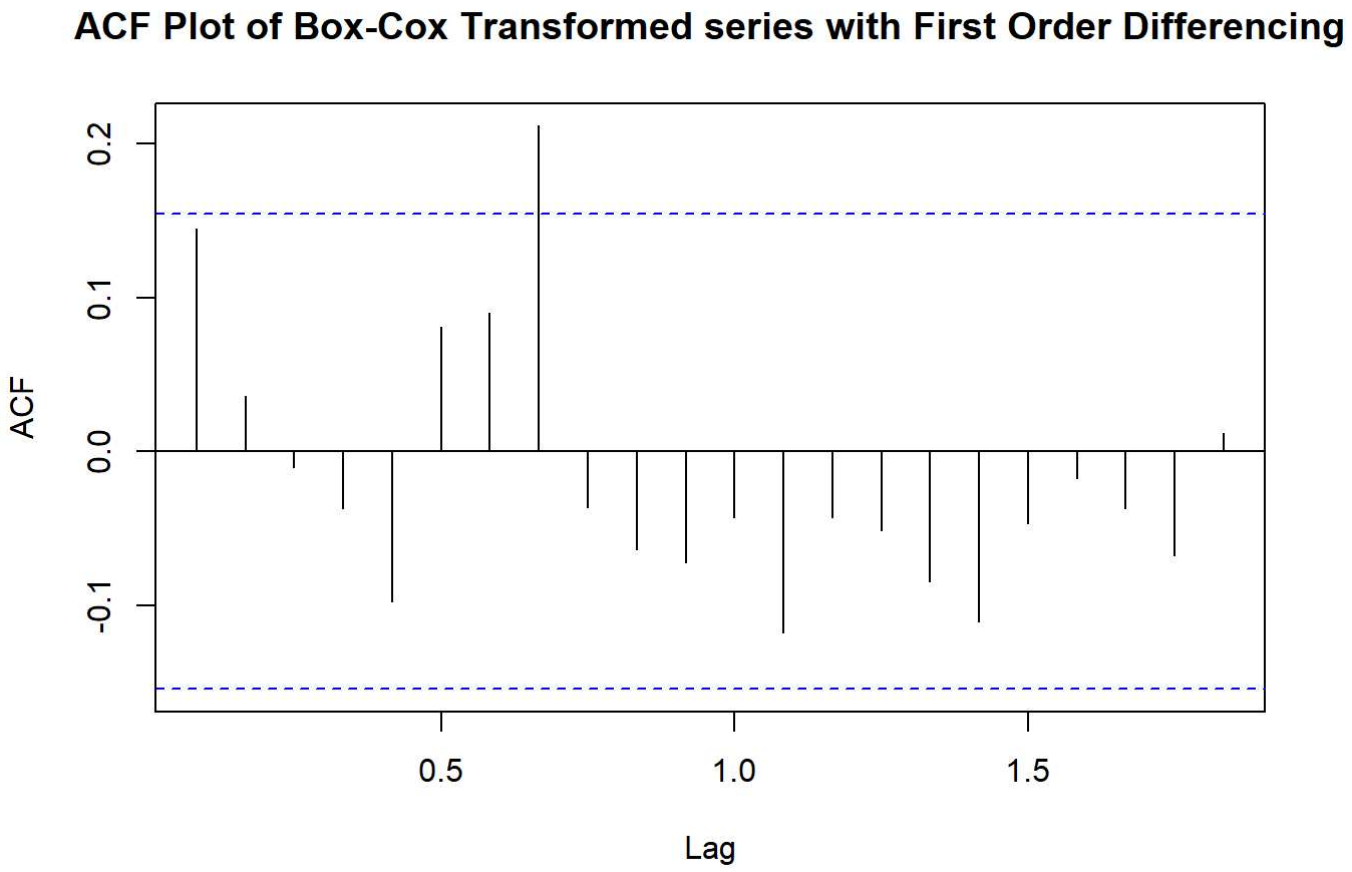
1. Trend In the Box-Cox transformed with First Order differenced series, we can not see any trend in the Box-Cox transformed first order differenced series. The series is fluctuating at a mean level, confirming stationarity of the series.
2. Seasonality Seasonality means repeating patterns in the time series plot. In the Box-Cox transformed with First Order differenced series, there are no repeating patterns. Therefore, there is no seasonality in the Box-Cox transformed time series plot.
3. Change in variance Initially, we can see change in variance. However, in the rest of the series there is slight to no change in variance.
4. Change Point/Intervention There is an intervention point near the year 2014.
5. Behaviour The series is a mix of autoregressive and moving average behaviour as we can see succeeding data points and fluctuations.

As there is no trend in our Box-Cox transformed first order differenced series, we can say that the series is stationary. However, we need more evidence to conclude that. Let's plot ACF and PACF plot and perform hypothesis tests namely, ADF unit root test, pp unit root test and kpss test to get concrete evidence to conclude that the Box-Cox transformed first order differenced series is stationary.

ACF PLOT OF BOX-COX TRANSFORMED SERIES WITH FIRST ORDER TRANSFORMED SERIES

We observe that there is no slowly decaying pattern. This shows stationarity in the series.

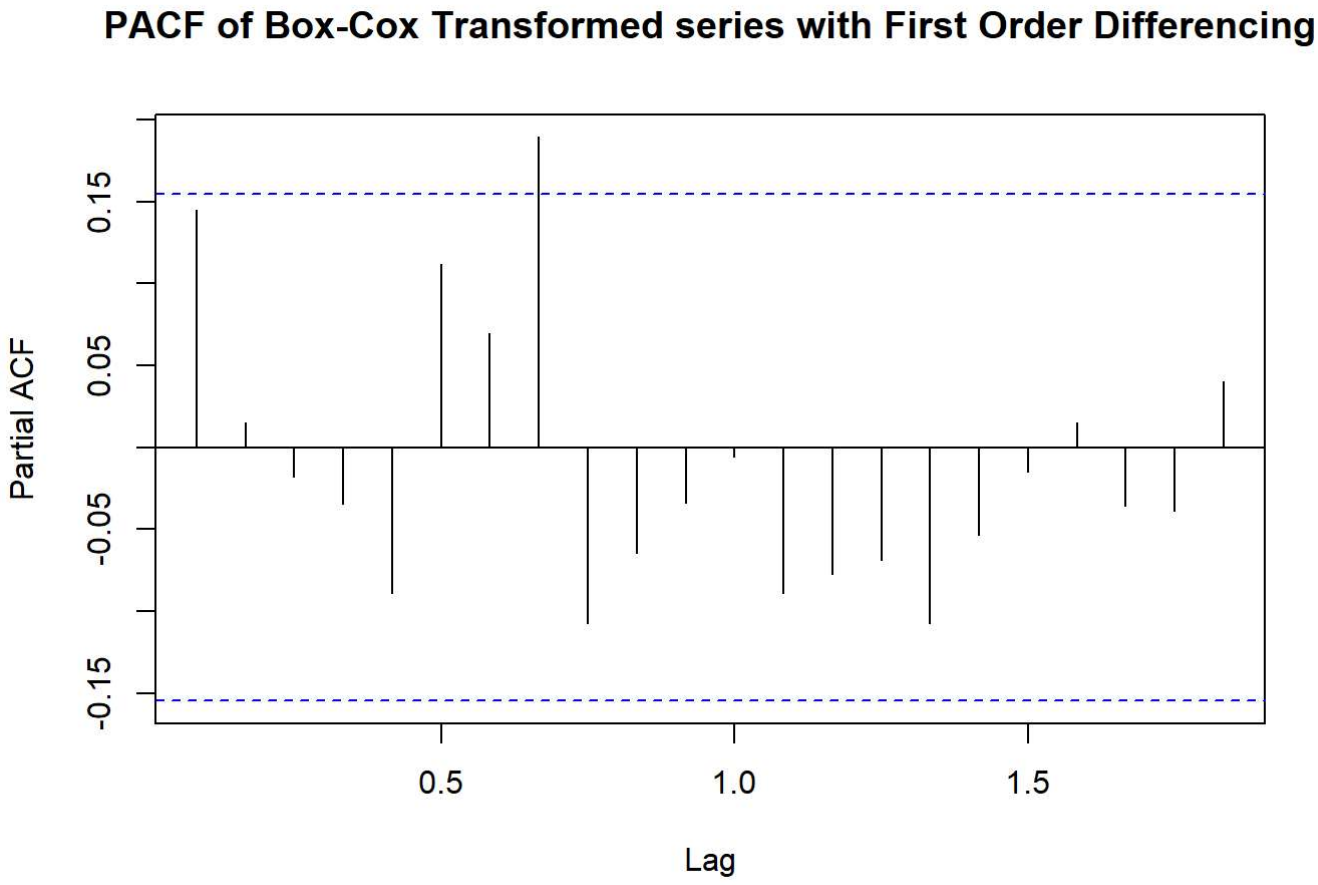
```
acf(bitindTSBCdif, main = "ACF Plot of Box-Cox Transformed series with First Order Differencin
g")
```



PACF PLOT OF BOX-COX TRANSFORMED SERIES WITH FIRST ORDER DIFFERENCED SERIES

In the PACF Plot of Box-Cox Transformed with First Order Differenced Series, we notice that we no more have a high first lag. The first lag is about 0.14, indicating stationarity in the series.

```
pacf(bitindTSBCdif, main = "PACF of Box-Cox Transformed series with First Order Differencing")
```



HYPOTHESIS TESTS

1. ADF (Augmented Dickey-Fuller Test) unit-root test H0 = The process is difference nonstationary (the process is nonstationary but becomes stationary after first differencing). HA = The process is stationary.

p-value of ADF test is 0.01, which is less than the significance level (α = 0.05). Therefore, the series is stationary.

```
adf.test(bitindTSBCdif)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  bitindTSBCdif
## Dickey-Fuller = -4.896, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

2. Phillips-Perron (pp) unit root test

H0 = The process is difference nonstationary (the process is nonstationary but becomes stationary after first differencing). HA = The process is stationary.

p-value of pp (Phillips-Perron) test is 0.01, which is less than the significance level (α = 0.05). Therefore, the series is stationary.

```
pp.test(bitindTSBCdif)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  bitindTSBCdif
## Dickey-Fuller Z(alpha) = -135.32, Truncation lag parameter = 4, p-value
## = 0.01
## alternative hypothesis: stationary
```

WHICH MODEL TO GO FOR ARMA(p,q) OR ARIMA(p,d,q)

When we have a time series that is stationary and have a mix of autoregressive and moving average behaviour, we fit ARMA(p,q) model. However, when we have a non-stationary series and have a mix of autoregressive and moving average behaviour, we must convert that series into stationary series using differencing. When differencing enters the arena, then we can no longer use ARMA(p,q), in this case we must use ARIMA(p,d,q). where, “p” is order of autoregression or number of autoregressive behaviour coefficients, “q” is the order of moving average or number of moving average coefficients and “d” is order of differencing. ARMA(p,q) model is given by: Yt = ∅1 Yt-1 + ∅2 Yt-2 + _ _ _ _ _ _ + ∅p Yt-p + et – Θ1 et-1 – Θ2 et-2 - _ _ _ _ _ _ - Θq et-q Where, Ø is Autoregressive coefficient. Θ is moving average coefficient. p and q are the orders of the model. d is the order of differencing

Since we are using ARIMA(p,d,q), we need to find the values of p, d and q. d is the order of differencing, since, we got a stationary series after first order differencing, therefore d = 1.

FINDING ESTIMATE COEFFICIENTS AND POSSIBLE SET OF MODELS

To find the values of p and q we will use the following tools: 1. ACF 2. PACF 3. EACF 4. BIC Table

Using these tools, we will get multiple values of p and q. Due to multiple values of p and q we will get a set of possible models. We will fit all these models in our series and find the best fitted model.

Let’s Use the tools one by one to get the possible set of models.

1. ACF AND PACF PLOTS

We will now calculate the value of p and q from the ACF and PACF plots. The value of q comes from ACF plot and the value of comes from PACF plot. To get the value of q, we calculate the significant lags in ACF plot. Significant lags are those lags that goes beyond the confidence interval. Confidence interval are represented by dashed lines in ACF and PACF plots.

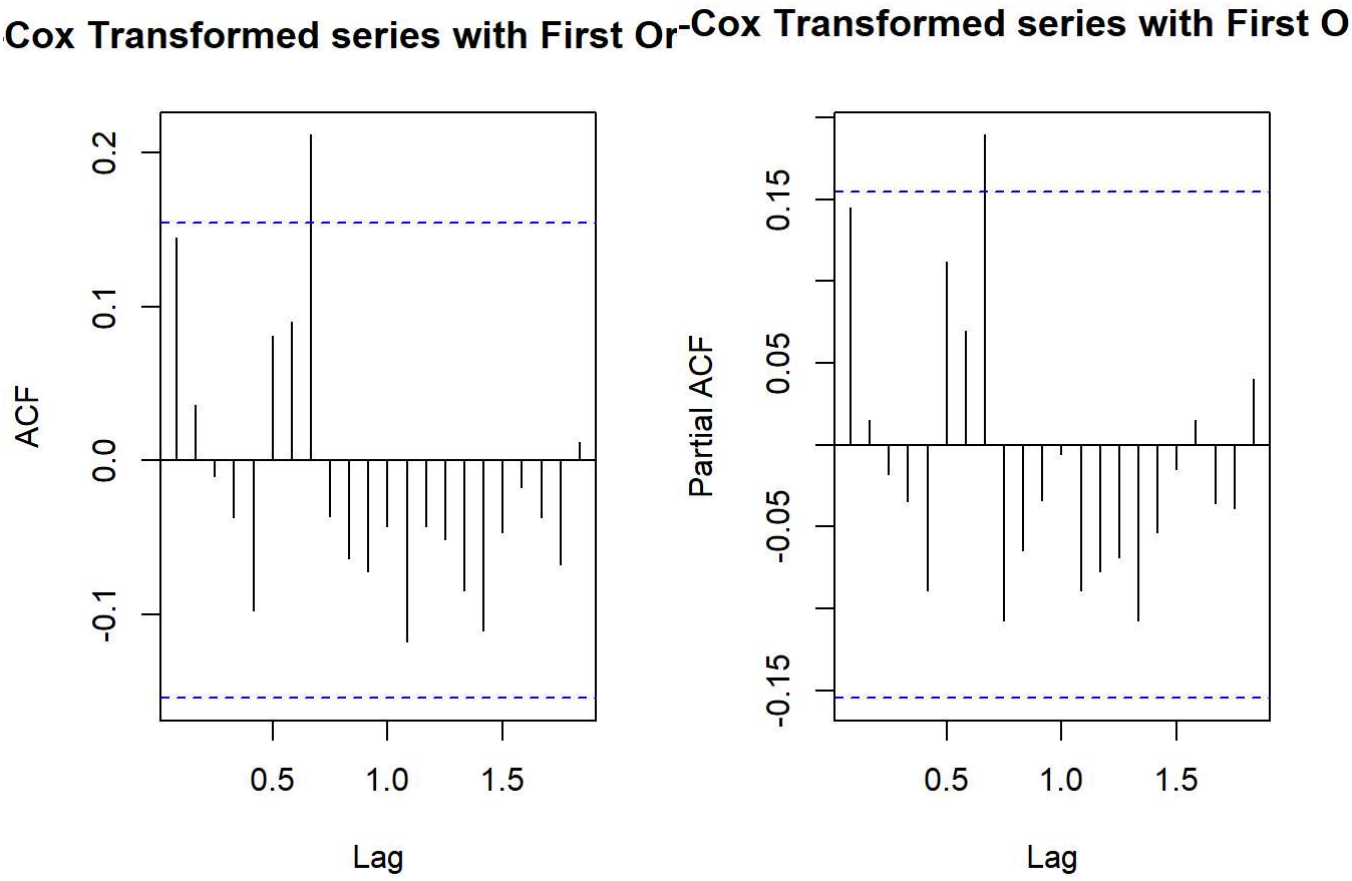
There is only 1 significant lag in ACF plot which is 8th lag. The first lag is near miss. Therefore, we can also include first lag as well. If the type of fist lag would have been in 10th lag, then we would not have included that, because that would have been a later lag. We are not interested in the impact of 10th month. In our case it is first lag, therefore, we must include that. Q will the number of possible significant lags. In our case q can be 1, that is an obvious significant lag (8th lag), or q can be 2, that includes 8th lag and a near miss first lag. Therefore, the values of q can be 1 and 2. (q = 1,2).

To find the value of p, we look at the PACF plot. Same as how we calculated the value of q, we will count the number of significant lags in PACF plot.

We have one obvious significant lag (8th lag) in PACF and one near miss lag (first lag). Therefore, the values of p can be 1 and 2. (p = 1,2).

The possible set of models we get from ACF and PACF are: ARIMA(1,1,1) ARIMA(1,1,2) ARIMA(2,1,1) ARIMA(2,1,2)

```
par(mfrow=c(1,2))
acf(bitindTSBCdif, main = "ACF of Box-Cox Transformed series with First Order Differencing")
pacf(bitindTSBCdif, main = "PACF of Box-Cox Transformed series with First Order Differencing")
```



2. EACF (EXTENDED AUTOCORRELATION FUNCTION) PLOT

In EACF plot, the first column represents the possible values of p and the first row represents the possible values of p. In an EACF plot, we look for the top-left zeros that have consecutive zeros that is not interrupted by x's, then we draw a vertex from that zero and spot the neighbour zeros. Next, we find the corresponding values of p and q for the zeros. We will first locate the top-left zero. As we can see the top-left zero will be the zero that correspond to the value of AR = 0 and MA = 0. This particular zero must be the top-left zero, where we draw the vertex, because it is not interrupted by any x's. now we will locate the neighbour zeros of the top-left zero.

The first column gives the value of p and the first row gives us the value of q. For the top-left zero, the value of MA(q) is 0 and the value of AR(p) is 0. Therefore, the model we get is ARIMA(0,1,0). The neighbour zero located on the right of the top-left zero, gives the value of p and q as 0 and 1 respectively. Therefore, the model we get is ARIMA(0,1,1). The neighbour zero located at the bottom of the top-left zero, gives the value of p and q as 1 and 0 respectively. Therefore, the model we get is ARIMA(1,1,0). The neighbour zero located diagonally to the top-left zero, gives the value of p and q as 1 and 1 respectively. Therefore, the model we get is ARIMA(1,1,1).

Therefore, from EACF we get the following possible set of models: ARIMA(0,1,0) ARIMA(0,1,1) ARIMA(1,1,0) ARIMA(1,1,1)

```
eacf(bitindTSBCdif, ar.max = 5, ma.max = 5)
```

```
## AR/MA
##   0 1 2 3 4 5
## 0 o o o o o o
## 1 o o o o o o
## 2 x o o o o o
## 3 x o o o o o
## 4 x o x o o o
## 5 x x x o x o
```

3. BIC TABLE

In BIC table, the models are arranged as best to worst from top to bottom. The p-lag values give us the value of p and error lag gives us the values of q. The shaded part tells us the value of p and q in these models. We will look at the top three best models to get the possible set of models.

The best model located at the top, has a BIC value of 7.3. The best model has p-lag-1 shaded and is also supported by other models as well, as it shaded in the other models as well and no error lag shaded. Therefore, the value of p will be 1 and the value of q will be 0. Hence, we have ARIMA model as ARIMA(1,1,0).

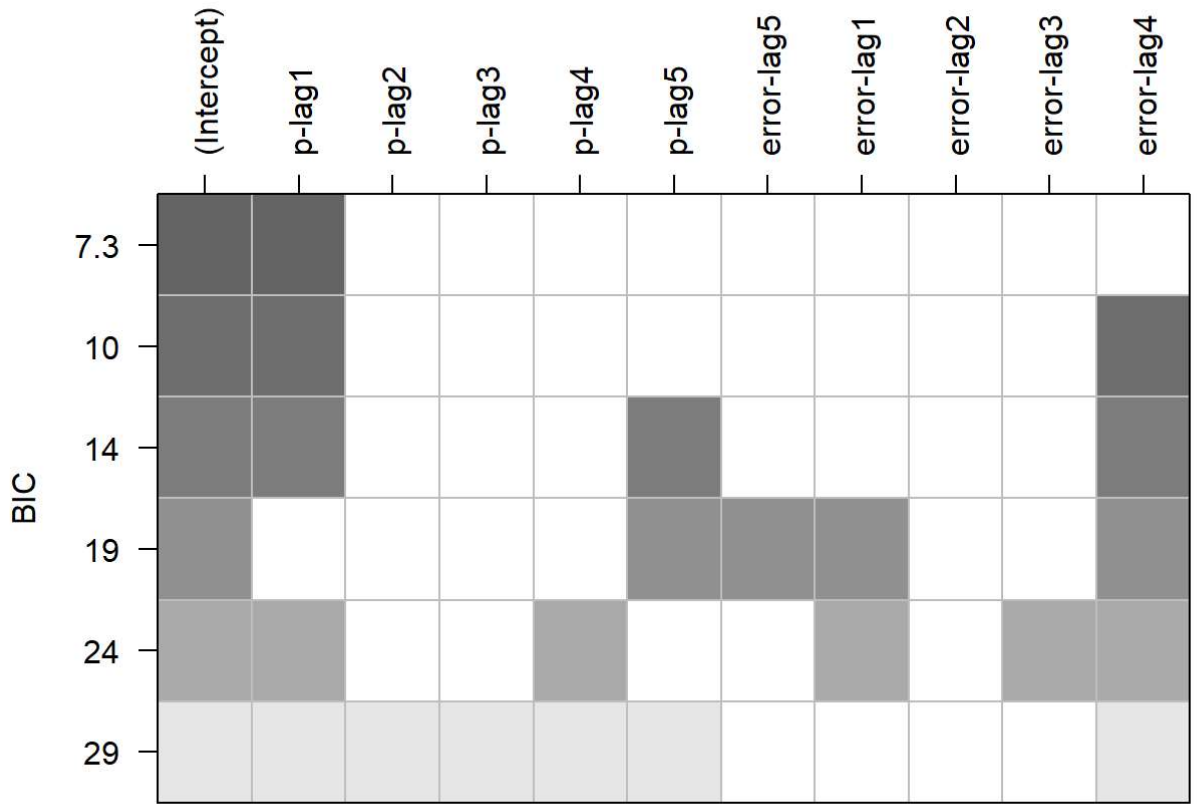
The second-best model has a BIC value of 10. The shaded region in p-lag is p-lag1, which is supported by other models as well. For error lag, the erroe-lag4 is shaded and is supported by other models as well. Therefore, the ARIMA model we get is ARIMA(1,1,4). We will not consider the third best model, because the BIC value of the third best model is 14, which is far high from that of the best model. Therefore, we need not to consider the third best model.

Therefore, from BIC table we get the following possible set of models: ARIMA(1,1,0) ARIMA(1,1,4)

```
res1 = armasubsets(y=bitindTSBCdif, nar = 5, nma = 5, y.name = "p", ar.method = "ols")
```

```
## Reordering variables and trying again:
```

```
plot(res1)
```



Out of the possible set of models we got from ACF, PACF, EACF and BIC table, ARIMA(1,1,1) is consistent in ACF-PACF and EACF. ARIMA(1,1,0) is consistent is EACF and BIC table. Therefore, we have the following set of possible models from ACF, PACF, EACF and BIC table: ARIMA(1,1,1) ARIMA(1,1,2) ARIMA(2,1,1) ARIMA(2,1,2) ARIMA(0,1,1) ARIMA(1,1,0) ARIMA(1,1,4) Hence we will be fitting these 7 models.

MODEL FITTING

To fit the ARIMA models we have 2 methods:

1. Maximum Likelihood Estimation - ML Estimation
2. Least Squares (Conditional SS - CSS) To find the model which best fit our series, we must find the model which have maximum estimate coefficients as significant.

We will use both Maximum Likelihood Estimation - ML Estimation and Least Squares (Conditional SS - CSS) model fitting methods to check the significance of the estimate coefficients.

1. ARIMA(1,1,1) - ML FITTED

For ARIMA(1,1,1), as p = 1 and q = 1, Therefore, we will have 1 AR and 1 MA coefficient respectively. As the p-value of both the estimate coefficients (AR and MA) are greater than the significance level (α = 0.05), the estimate coefficients are not significant. By ML method we got both the estimate coefficients (AR and MA) as insignificant.

```
model.111 = arima(bitindTS, order = c(1,1,1), method = "ML")
coeftest(model.111)
```

```
##
## z test of coefficients:
##
##       Estimate Std. Error z value Pr(>|z|)
## ar1  0.1393832  0.4591484  0.3036   0.7615
## ma1 -0.0062261  0.4607667 -0.0135   0.9892
```

2. ARIMA(1,1,1) - CSS FITTED

As the p-value of both the estimate coefficients (AR and MA) are greater than the significance level (α = 0.05), the estimate coefficients are not significant. By CSS method we got both the estimate coefficients (AR and MA) as insignificant.

```
model.111CSS = arima(bitindTS, order = c(1,1,1), method = "CSS")
coeftest(model.111CSS)
```

```
##
## z test of coefficients:
##
##       Estimate Std. Error z value Pr(>|z|)
## ar1  0.1410742  0.4638558  0.3041   0.7610
## ma1 -0.0094778  0.4659000 -0.0203   0.9838
```

The results of ML and CSS for ARIMA(1,1,1) are consistent.

3. ARIMA(1,1,2) - ML FITTED

For ARIMA(1,1,2), we have p = 1 and q = 2. We have 1 AR and 2 MA estimate coefficients. The p-value of all the three estimate coefficients is less than the significance level (α = 0.05). Therefore, all the three (1 AR and 2 MA) estimate coefficients are significant by ML method.

```
model.112 = arima(bitindTS, order = c(1,1,2), method = "ML")
coeftest(model.112)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error z value  Pr(>|z|)
## ar1 -0.858473   0.103248 -8.3147 < 2.2e-16 ***
## ma1  1.048070   0.130906  8.0063 1.182e-15 ***
## ma2  0.272386   0.098402  2.7681  0.005638 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4. ARIMA(1,1,2) - CSS FITTED

After fitting ARIMA(1,1,2) by CSS method we get all the three estimate coefficients (1 AR and 2 MA) as significant as the p-value is less than the significance level (α = 0.05).

```
model.112CSS = arima(bitindTS, order = c(1,1,2), method = "CSS")
coeftest(model.112CSS)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error z value  Pr(>|z|)
## ar1 -0.866984   0.102901 -8.4254 < 2.2e-16 ***
## ma1  1.059757   0.130629  8.1127 4.951e-16 ***
## ma2  0.280728   0.099493  2.8216  0.004779 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5. ARIMA(2,1,1) - ML FITTED

In ARIMA(2,1,1), we don't have any p-values. For model fitting there is no problem the model is fitted, but we are not getting the values of standardised error and therefore, no p-values. We have to rely on CSS method to get the p-values.

```
model.211 = arima(bitindTS, order = c(2,1,1), method = "ML")
coeftest(model.211)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ar1 0.064386        NaN     NaN      NaN
## ar2 0.015278        NaN     NaN      NaN
## ma1 0.066888        NaN     NaN      NaN
```

6. ARIMA(2,1,1) - CSS FITTED

When we fitted ARIMA(2,,1,1) model using CSS method we get the same result as that of the ML method. We got no p-values.

```
model.211CSS = arima(bitindTS, order = c(2,1,1), method = "CSS")
coeftest(model.211CSS)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ar1 0.064880        NaN     NaN      NaN
## ar2 0.015508        NaN     NaN      NaN
## ma1 0.067292        NaN     NaN      NaN
```

7. ARIMA(2,1,1) - CSS-ML FITTED

ML is an iterative method, where it starts computing from a starting value. In ML-CSS method, we provide ML, the starting values as CSS values. IN ML-CSS method too we cannot get the standardised errors and hence, no p-values.

```
model.211MLCSS = arima(bitindTS, order = c(2,1,1), method = "CSS-ML")
coeftest(model.211MLCSS)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ar1 0.063705        NaN     NaN      NaN
## ar2 0.015436        NaN     NaN      NaN
## ma1 0.066521        NaN     NaN      NaN
```

8. ARIMA(2,1,2) - ML FITTED

In ARIMA(2,1,2) model, we have 2 AR and 2 MA estimate coefficients. The p-value of all four estimate coefficients is less than the significance level (α = 0.05). Therefore, all the four estimate coefficients are significant as per ML method.

```
model.212 = arima(bitindTS, order = c(2,1,2), method = "ML")
coeftest(model.212)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error  z value  Pr(>|z|)
## ar1 -0.529398   0.091141  -5.8086 6.301e-09 ***
## ar2 -0.835877   0.068231 -12.2507 < 2.2e-16 ***
## ma1  0.662452   0.061559  10.7612 < 2.2e-16 ***
## ma2  0.935219   0.053817  17.3777 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9. ARIMA(2,1,2) - CSS FITTED

In CSS method, we got the same result as of ML method. The p-values of all the estimate coefficients are less than the significance level (α = 0.05). therefore, all the estimate coefficients are significant for ARIMA(2,1,2).

```
model.212CSS = arima(bitindTS, order = c(2,1,2), method = "CSS")
coeftest(model.212CSS)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error  z value  Pr(>|z|)
## ar1 -0.545799   0.098730  -5.5282 3.236e-08 ***
## ar2 -0.848059   0.067947 -12.4812 < 2.2e-16 ***
## ma1  0.676515   0.061934  10.9231 < 2.2e-16 ***
## ma2  0.952984   0.053596  17.7810 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

10. ARIMA(0,1,1) - ML FITTED

ARIMA(0,1,1) model have only one MA estimate coefficient, as q = 1 and zero AR estimate coefficients as p = 0. The value of the MA estimate coefficients is greater than significance level (α = 0.05). Therefore, the MA estimate coefficient is not significant in ML method.

```
model.011 = arima(bitindTS, order = c(0,1,1), method = "ML")
coeftest(model.011)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ma1 0.126071   0.075546  1.6688  0.09516 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

11. ARIMA(0,1,1) - CSS FITTED

In CSS method too, the p-value of the MA estimate coefficient is greater than the significance level (α = 0.05). Therefore, the MA estimate coefficient is not significant.

```
model.011CSS = arima(bitindTS, order = c(0,1,1), method = "CSS")
coeftest(model.011CSS)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ma1 0.126806   0.075739  1.6742  0.09408 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

12. ARIMA(1,1,0) - ML FITTED

In ARIMA(1,1,0) model, there is only 1 AR estimate coefficient as p = 1 and zero MA coefficient as q = 0. The p-value of AR estimate coefficient is greater than the significance level (α = 0.05). Therefore, the AR estimate coefficient is not significant in ML method.

```
model.110 = arima(bitindTS, order = c(1,1,0), method = "ML")
coeftest(model.110)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ar1 0.131888   0.078757  1.6746  0.09401 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

13. ARIMA(1,1,0) - CSS FITTED

In CSS method, AR estimate coefficient is not significant as the p-value is greater than the significance level (α = 0.05).

```
model.110CSS = arima(bitindTS, order = c(1,1,0), method = "CSS")
coeftest(model.110CSS)
```

```
##
## z test of coefficients:
##
##     Estimate Std. Error z value Pr(>|z|)
## ar1 0.132726   0.079026  1.6795  0.09305 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

14. ARIMA(1,1,4) - ML FITTED

In ARIMA(1,1,4) model we have 1 AR estimate coefficient and 4 MA estimate coefficient. The p-value of the AR estimate coefficient is less than the significance level (α = 0.05). Therefore, The AR estimate coefficient is significant.

The p-value of the first MA estimate coefficient is less than the significance level (α = 0.05). Hence, The first MA estimate coefficient is significant. However, the rest of the three MA coefficients are insignificant as their p-values are greater than the significance level (α = 0.05).

```
model.114 = arima(bitindTS, order = c(1,1,4), method = "ML")
coeftest(model.114)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error z value  Pr(>|z|)
## ar1 -0.714550   0.184622 -3.8703 0.0001087 ***
## ma1  0.896413   0.184164  4.8675  1.13e-06 ***
## ma2  0.147209   0.113101  1.3016 0.1930639
## ma3 -0.052147   0.118222 -0.4411 0.6591424
## ma4  0.127529   0.105913  1.2041 0.2285527
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

15. ARIMA(1,1,4) - CSS FITTED

In CSS method, we get the same results. The only AR estimate coefficient is significant as the p-value is less than the significance level (α = 0.05). The first MA estimate coefficient is significant as p-value is less than the significance level (α = 0.05). the rest of the MA estimate coefficients are insignificant as p-value is greater than the significance level (α = 0.05).

```
model.114CSS = arima(bitindTS, order = c(1,1,4), method = "CSS")
coeftest(model.114CSS)
```

```
##
## z test of coefficients:
##
##      Estimate Std. Error z value  Pr(>|z|)
## ar1 -0.720819   0.190688 -3.7801 0.0001568 ***
## ma1  0.903800   0.188593  4.7923 1.649e-06 ***
## ma2  0.149802   0.114327  1.3103 0.1900960
## ma3 -0.053474   0.120142 -0.4451 0.6562544
## ma4  0.130248   0.108991  1.1950 0.2320716
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

SORTING OF MODELS BASED ON AIC AND BIC VALUES

After fitting the all the possible set of models, we know must find the best fitting model. For this purpose, we will have a look at the AIC and BIC values of these models. The model having the lowest AIC and BIC value will be the best fitting model. First, let's have a look at the AIC values of the fitted models.

```
#AIC and BIC

sort.score <- function(x, score = c("bic", "aic")){
  if (score == "aic"){
    x[with(x, order(AIC)),]
  } else if (score == "bic") {
    x[with(x, order(BIC)),]
  } else {
    warning('score = "x" only accepts valid arguments ("aic","bic")')
  }
}
```

We observe that ARIMA(2,1,2) have the lowest AIC value as 5477.276. the second least AIC value is 5478.478 and that is of ARIMA(1,1,4). Therefore, the best fitted model according to AIC value is ARIMA(2,1,2). The second best fitted model according to AIC value is ARIMA(1,1,4).

```
sort.score(AIC(model.111,model.112,model.211,model.212,
               model.011,model.110,model.114), score = "aic")
```

```
##           df      AIC
## model.212  5 5477.276
## model.114  6 5478.478
## model.112  4 5479.077
## model.110  2 5479.516
## model.011  2 5479.620
## model.111  3 5481.515
## model.211  4 5483.509
```

Now, let's check the best fitted model according to BIC values.

ARIMA(1,1,0) model have the lowest BIC value as 5485.678. Therefore, we have ARIMA(1,1,0) is the best fitted model according to BIC values.

From AIC and BIC values we got two best fitting models, namely, ARIMA(2,1,2) (from AIC) and ARIMA(1,1,0) (from BIC). To choose the best fitting model we need another tool. We will now fit the models using forecast package, to get the error measures

```
sort.score(BIC(model.111,model.112,model.211,model.212,
               model.011,model.110,model.114), score = "bic")
```

```
##           df      BIC
## model.110  2 5485.678
## model.011  2 5485.783
## model.111  3 5490.760
## model.112  4 5491.403
## model.212  5 5492.683
## model.211  4 5495.834
## model.114  6 5496.967
```

ERROR MEASURES

Now, let's look at the error measures of the fitted models ARIMA(2,1,2) and ARIMA(1,1,0).

We will look at RMSE (Root Mean Square Error), MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error) and MASE (Mean Absolute Standardised Error). • ARIMA(2,1,2) have less Root Mean Square error than ARIMA(1,1,0). • ARIMA(2,1,2) have less Mean Absolute error than ARIMA(1,1,0). • ARIMA(1,1,0) have less Mean Absolute Percentage error than ARIMA(2,1,2). • ARIMA(2,1,2) have less Mean Absolute Standardised error than ARIMA(1,1,0).

According to error measures, ARIMA(2,1,2) is the best fitted model. To confirm this, we should also check the significance of estimate coefficients of ARIMA(2,1,2) model. As discussed earlier, ARIMA(2,1,2) model had two AR estimate coefficients and two MA estimate coefficients. We fitted ARIMA(2,1,2) model using ML and CSS method. We got all the four estimate coefficient significant. ARIMA(1,1,0) had no significant estimate coefficient.

Therefore, the best fitted model, for our time series will be ARIMA(2,1,2).

```
model.111A = Arima(bitindTS, order = c(1,1,1), method = 'ML')
model.112A = Arima(bitindTS, order = c(1,1,2), method = 'ML')
model.211A = Arima(bitindTS, order = c(2,1,1), method = 'ML')
model.212A = Arima(bitindTS, order = c(2,1,2), method = 'ML')
model.011A = Arima(bitindTS, order = c(0,1,1), method = 'ML')
model.110A = Arima(bitindTS, order = c(1,1,0), method = 'ML')
model.114A = Arima(bitindTS, order = c(1,1,4), method = 'ML')

Smodel.111A <- accuracy(model.111A)[1:7]
Smodel.112A <- accuracy(model.112A)[1:7]
Smodel.211A <- accuracy(model.211A)[1:7]
Smodel.212A <- accuracy(model.212A)[1:7]
Smodel.011A <- accuracy(model.011A)[1:7]
Smodel.110A <- accuracy(model.110A)[1:7]
Smodel.114A <- accuracy(model.114A)[1:7]

df.Smodels <- data.frame(
  rbind(Smodel.111A, Smodel.112A, Smodel.211A,
        Smodel.212A, Smodel.011A, Smodel.110A,
        Smodel.114A)
)
colnames(df.Smodels) <- c("ME", "RMSE", "MAE", "MPE", "MAPE", "MASE", "ACF1")
rownames(df.Smodels) <- c("ARIMA(1,1,1)", "ARIMA(1,1,2)", "ARIMA(2,1,1)",
                          "ARIMA(2,1,2)", "ARIMA(0,1,1)", "ARIMA(1,1,0)",
                          "ARIMA(1,1,4)")
df.Smodels
```

```
##                    ME    RMSE     MAE      MPE     MAPE      MASE         ACF1
## ARIMA(1,1,1) 693599.2 5853884 2951392 2.518370 17.44590 0.2270917 -0.013582865
## ARIMA(1,1,2) 631986.9 5770071 2971860 2.438187 18.05708 0.2286666 -0.042599482
## ARIMA(2,1,1) 690963.7 5853769 2951410 2.514455 17.44891 0.2270930 -0.011382345
## ARIMA(2,1,2) 718956.8 5691449 2822234 2.497301 18.72537 0.2171538  0.002874578
## ARIMA(0,1,1) 710175.5 5855822 2945637 2.550276 17.41493 0.2266489 -0.007842588
## ARIMA(1,1,0) 695133.5 5853895 2950675 2.521615 17.44274 0.2270365 -0.012414128
## ARIMA(1,1,4) 654373.5 5684633 2987766 2.448135 19.35778 0.2298904 -0.014770514
```