Applied Econometrics Assignment-III

We have selected a dataset on the daily Historical prices of Bitcoin. Bitcoin is a decentralized digital currency (cryptocurrency), it is based on a technology developed by Satoshi Nakamoto in 2008, called Block Chain. Block chain is a Public ledger recording transactions but in a distributed manner. The very idea of Blockchain and the existence of bitcoin came in order to mitigate against the flaws in the finance Industry (as a response to the Financial crisis of 2008).

In the recent years of Bitcoin became very popular and its prices rose very quickly. As a part of the assignment, we are Fitting an ARIMA and an ARDL model on the weekly average prices of Bitcoin. We hope this analysis on weekly average prices provide an insight on how volatile the prices of Bitcoin are.

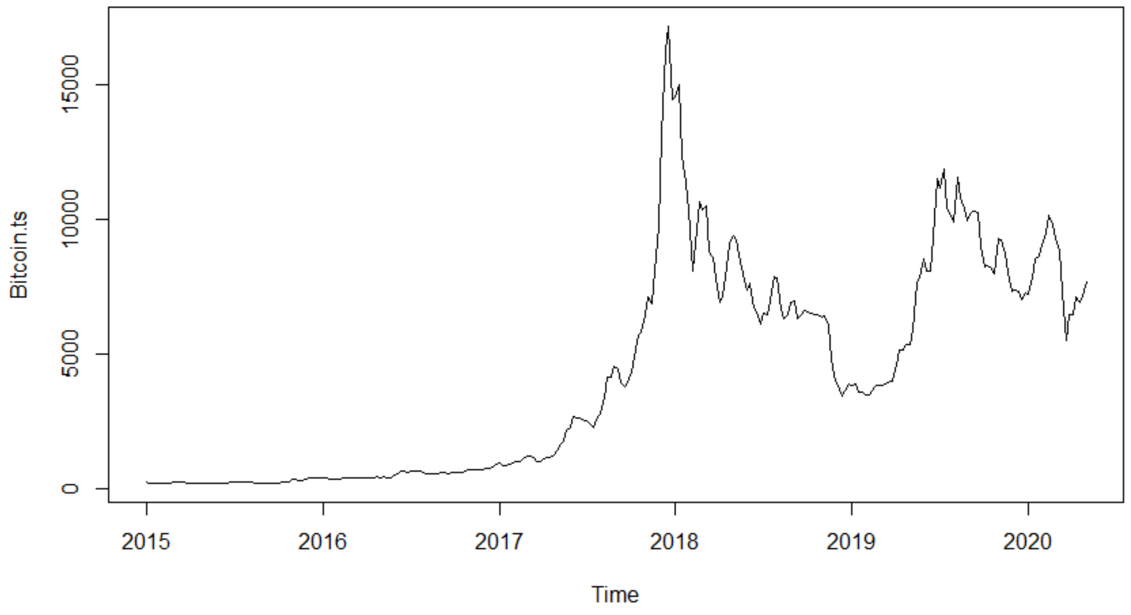
We have used R itself to get the weekly averages of the prices (using lubridate package).

ARIMA Model

For fitting the model, we have taken the weekly averages from 2015 to April 2020.

a. As we know that for many of the time series analysis, the first step would be to check for the non-Stationarity of the data. This is important in our case because ARIMA is defined for Stationary time series data.

We can check for non-stationarity in the data by observing its graph over time and as well as using some formal tests. Initially let's see the graphical method.

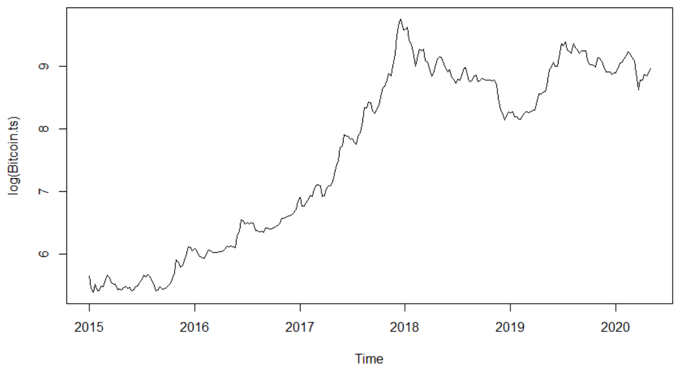


plot 1: Bitcoin average prices vs time

Plot 1 shows the plot between the average Bitcoin prices and time. Bitcoin.ts is the timer series object of our data (the object's frequency is 52 and starts from the first week of 2015). We can see that the graph clearly does not look stationary, as the mean is not zero or constant (over the period) and the standard deviation does not seem to be constant. We can observe that the present prices are highly depending on the previous values in almost all of the parts in the graph.

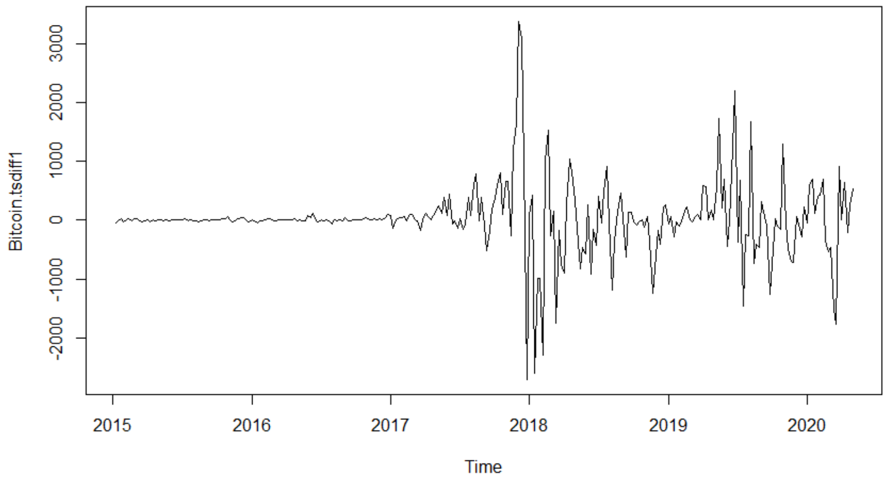
So, in order to remove the non-stationarity in the sample let's try and plot a graph between the differences of the average price with the immediate previous value (let's call it first differences) and time.

In the plot 1 we can see that the price has increased suddenly in 2017 due to the sudden popularity, as the prices range from 0 to very high values, we can use log function to clearly see the price movements before 2017 (plot 2).

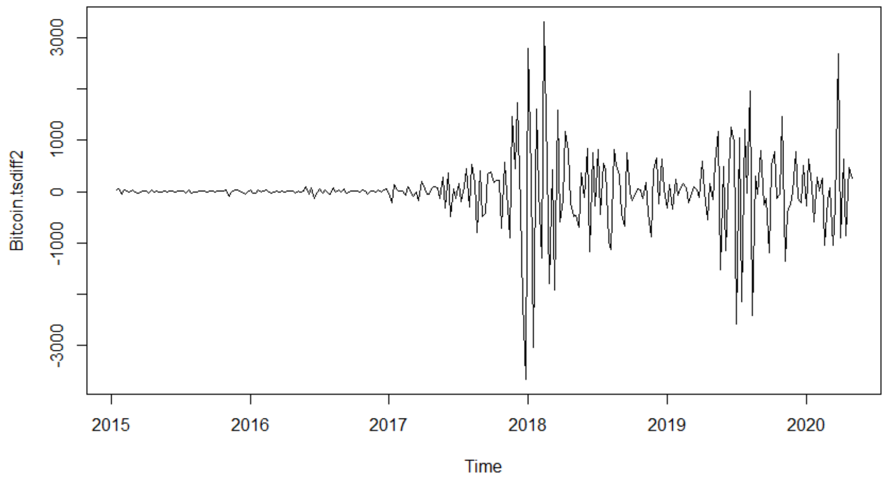


plot 2: logarithm of prices vs time

As discussed, on plotting the first differences of the time series data over time we can see that the graph now looks more stationary on mean than before. The mean is now closer to zero but the standard deviation does not seem to zero still. There are huge spikes in the middle of the graph compared to the other parts in the graph. Let us try to see if there would be any difference on taking one more difference to this data (let’s call this second difference to the original data).



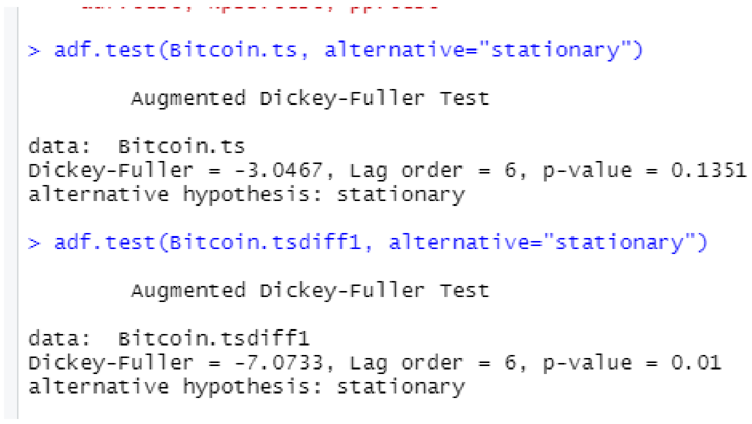
plot 3: first differences vs time



plot 4: second differences vs time

We can see that second differences seem even more stationary compared to the first differences. Let us see if the first differences are stationary using a Formal unit root test. We can perform an augmented dicky fuller test (adf) to check for the non-stationarity in the data.

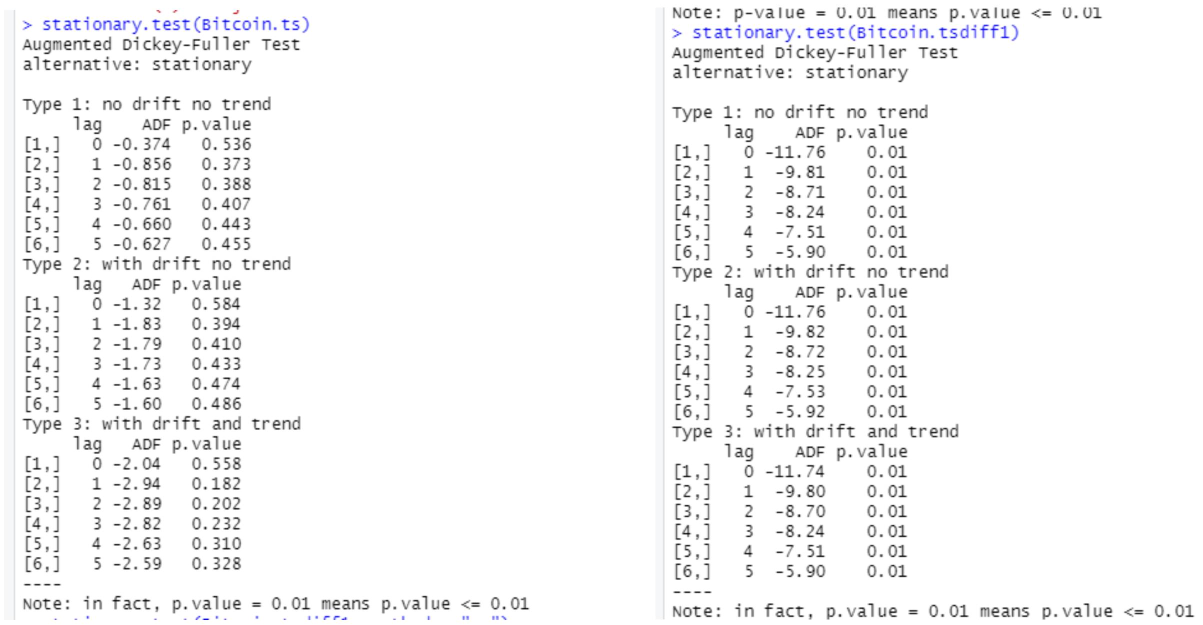
Here, we have not used frequency as 365.25/7 instead we took 52 because many other functions that use ts objects in R require integer frequency. This causes an extra week in leap years, as our sample size is large, we don’t observe any effects in the graphs. But we need to keep this in mind during forecasts.



```
> adf.test(Bitcoin.ts, alternative="stationary")

        Augmented Dickey-Fuller Test

data:  Bitcoin.ts
Dickey-Fuller = -3.0467, Lag order = 6, p-value = 0.1351
alternative hypothesis: stationary

> adf.test(Bitcoin.tsdiff1, alternative="stationary")

        Augmented Dickey-Fuller Test

data:  Bitcoin.tsdiff1
Dickey-Fuller = -7.0733, Lag order = 6, p-value = 0.01
alternative hypothesis: stationary
```

Figure 1: adf test

Figure 1 shows the result on performing an adf test using adf.test function in the tseries package. We can see that the ADF test statistic in the case of original data is -3.05 and for the first differences it is -7.07. In both cases the number of lags taken (to mitigate the effects of autocorrelation) are 6, the p-value for the same are 0.1351 for the original data which means we accept the Null at a 99% confidence level. For the first differences the corresponding p-value is ≤ 0.01 that means we reject the Null hypothesis at 99% confidence interval. As the Null hypothesis states non-stationary and the alternate hypothesis states stationary. As expected, we can see that the original data has some non-stationarity in it, but also the test rejects the Null hypothesis that the data is non-stationary for the first differences.

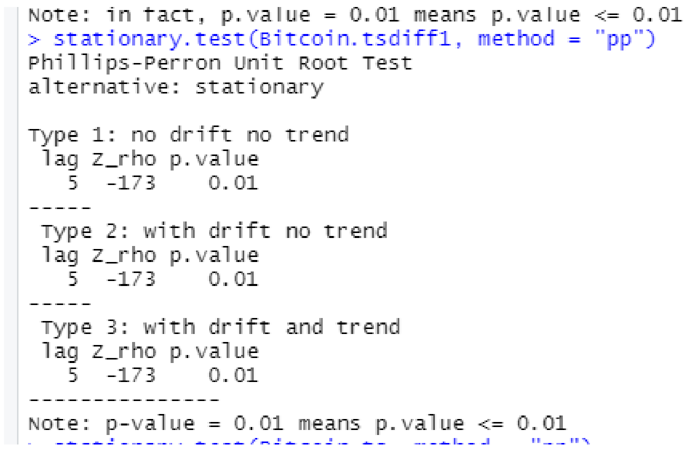


```
> stationary.test(Bitcoin.ts)
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag    ADF p.value
[1,]   0 -0.374   0.536
[2,]   1 -0.856   0.373
[3,]   2 -0.815   0.388
[4,]   3 -0.761   0.407
[5,]   4 -0.660   0.443
[6,]   5 -0.627   0.455
Type 2: with drift no trend
     lag   ADF p.value
[1,]   0 -1.32   0.584
[2,]   1 -1.83   0.394
[3,]   2 -1.79   0.410
[4,]   3 -1.73   0.433
[5,]   4 -1.63   0.474
[6,]   5 -1.60   0.486
Type 3: with drift and trend
     lag   ADF p.value
[1,]   0 -2.04   0.558
[2,]   1 -2.94   0.182
[3,]   2 -2.89   0.202
[4,]   3 -2.82   0.232
[5,]   4 -2.63   0.310
[6,]   5 -2.59   0.328
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
```

```
Note: p-value = 0.01 means p.value <= 0.01
> stationary.test(Bitcoin.tsdiff1)
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag    ADF p.value
[1,]   0 -11.76    0.01
[2,]   1  -9.81    0.01
[3,]   2  -8.71    0.01
[4,]   3  -8.24    0.01
[5,]   4  -7.51    0.01
[6,]   5  -5.90    0.01
Type 2: with drift no trend
     lag    ADF p.value
[1,]   0 -11.76    0.01
[2,]   1  -9.82    0.01
[3,]   2  -8.72    0.01
[4,]   3  -8.25    0.01
[5,]   4  -7.53    0.01
[6,]   5  -5.92    0.01
Type 3: with drift and trend
     lag    ADF p.value
[1,]   0 -11.74    0.01
[2,]   1  -9.80    0.01
[3,]   2  -8.70    0.01
[4,]   3  -8.24    0.01
[5,]   4  -7.51    0.01
[6,]   5  -5.90    0.01
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
```

Figure 2: adf test first differences and original data

Figure 2 shows the same adf test but using a different function stationary.test in aTSA package. Here, we can see the three types using drift and trend in the test. By default, adf.test function includes a trend component and drift which is type 3 in this case. Now, we can see that the p-value (first differences) for all the three types is ≤ 0.01 so we reject the Null (non-stationary). we are more concerned with type1.



```
Note: in fact, p.value = 0.01 means p.value <= 0.01
> stationary.test(Bitcoin.tsdiff1, method = "pp")
Phillips-Perron Unit Root Test
alternative: stationary

Type 1: no drift no trend
 lag Z_rho p.value
   5  -173    0.01
-----
 Type 2: with drift no trend
 lag Z_rho p.value
   5  -173    0.01
-----
 Type 3: with drift and trend
 lag Z_rho p.value
   5  -173    0.01
---------------
Note: p-value = 0.01 means p.value <= 0.01
```

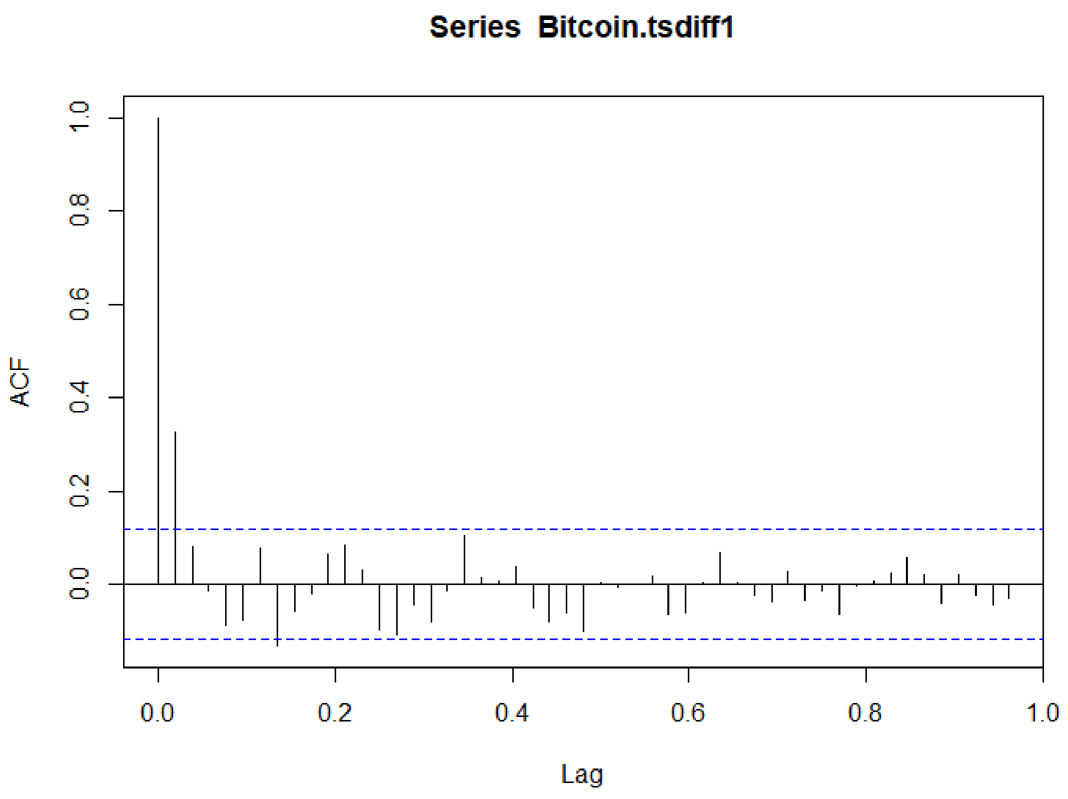
In order to confirm, we have performed a **pp test** as well on the first differences data. Similar to adf test the Null in this case is also that the data is non-stationary. We can see that in all the three cases the p-value is ≤ 0.01. So, we reject Null hypothesis that the data is non-stationary. We can also see that the second differences also give similar results on performing these tests. Similarly, we can also perform a kpss test as well.

Figure 3: pp test first differences

From the graphical approach we have thought that in order to make the data stationary it would not be enough to take a single difference and rather go for the second difference. But from the formal tests we have decided to take only one difference to make the data stationary. However, we will get to the same result on applying auto.arima function on the original data as well.

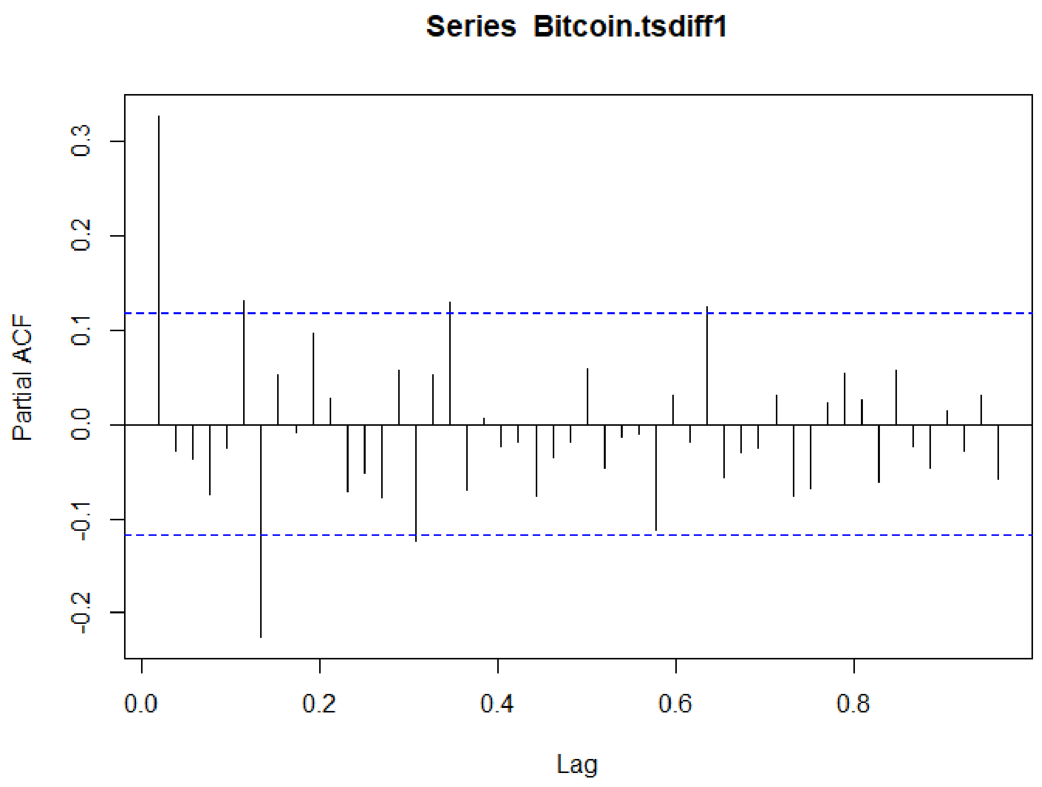
b. Now we need to select the ARIMA model for our data, we can select it by using autocorrelation function and partial autocorrelation function plots. We can directly use the auto.arima function to select the ARIMA model as well.

First let's see how to select our model using the correlogram or autocorrelation function plot and partial correlation function plot.



plot 5: ACF vs Lag

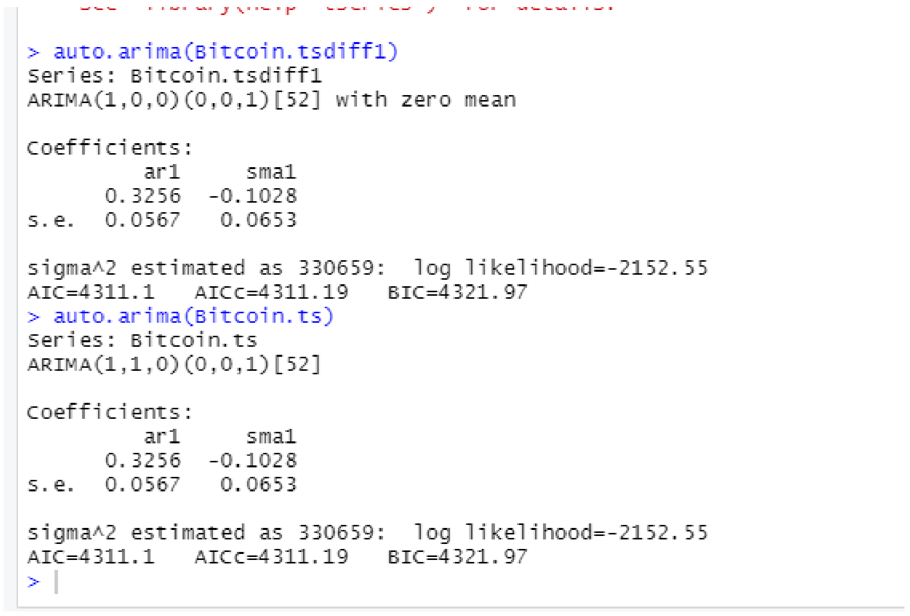
In the above plot we can see that there is only one significant spike in the beginning and another spike in the middle which is almost insignificant, so taking only one spike into consideration ACF plot indicates an MA(1) model. Now, let’s see the PACF plot for the data (single differences).



plot 6: PACF vs Lag

So, in the above plot we can see that it is unclear to perfectly decide how many significant spikes are there. We can see one highly significant spike but after few spikes, there are four more spikes which are on the border. We cannot accurately decide which model this plot could indicate, but as there is one significant spike so it could be an AR(1) model. But we cannot be as certain as in the previous case. So, based on this we can say that the possible models could be ARMA(0,1) model or ARMA(1,0) or ARMA(1,1). Based on the principle of parsimony, it is better we choose either ARIMA(0,1,1) or ARIMA(1,1,0) models on the original dataset (as our original data contains one unit root). Or we can use ARMA(0,1) or ARMA(1,0) for the first differences.

Let's look at the second method which is by using auto.arima function. This function can be used to find the appropriate ARIMA model for our time series data. We will require forecast and urca packages for this command.



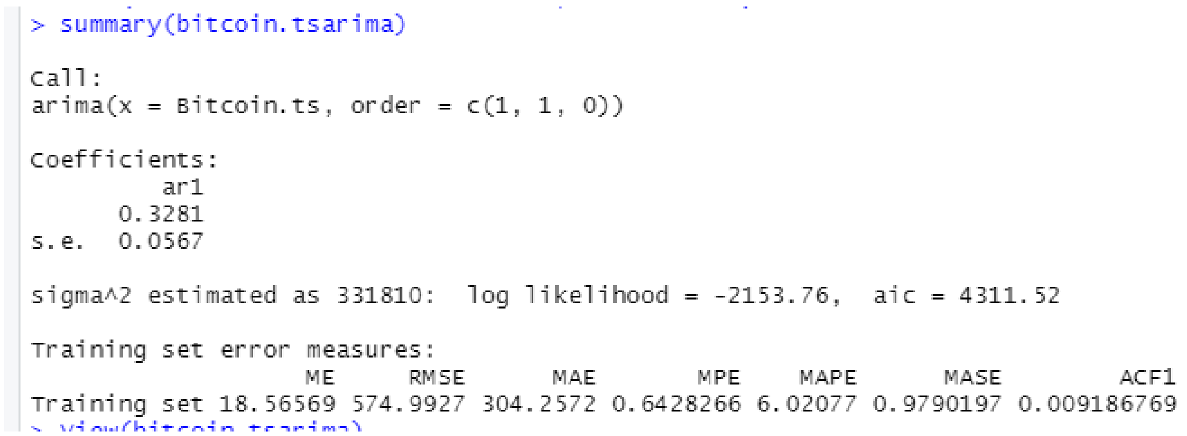
```
> auto.arima(Bitcoin.tsdiff1)
Series: Bitcoin.tsdiff1
ARIMA(1,0,0)(0,0,1)[52] with zero mean

Coefficients:
         ar1     sma1
      0.3256  -0.1028
s.e.  0.0567   0.0653

sigma^2 estimated as 330659:  log likelihood=-2152.55
AIC=4311.1   AICc=4311.19   BIC=4321.97
> auto.arima(Bitcoin.ts)
Series: Bitcoin.ts
ARIMA(1,1,0)(0,0,1)[52]

Coefficients:
         ar1     sma1
      0.3256  -0.1028
s.e.  0.0567   0.0653

sigma^2 estimated as 330659:  log likelihood=-2152.55
AIC=4311.1   AICc=4311.19   BIC=4321.97
>
```

Figure 4: Auto ARIMA

As expected for the first differences, auto.arima function has suggested an ARMA(0,1) or ARMA(1,0) models. But for the original dataset there is only ARIMA(1,1,0) model being suggested, another model is ARIMA(0,0,1) but the data is non-stationary if we don't take at least one difference. So, we selected the model ARIMA(1,1,0) on our original dataset based on the plots and auto.arima function.

c. As we have decided with the ARIMA model that we are going to use for the further analysis, let's estimate the parameters of the model that we have chosen and forecast some future values using the same.

We have used the arima function in R to fit the model ARIMA(1,1,0) on our original data set. We cannot use the arima function on the differences directly though, because in the end we need to predict the values of prices in the future but not the price differences. One can still use this but have to interpret accordingly.

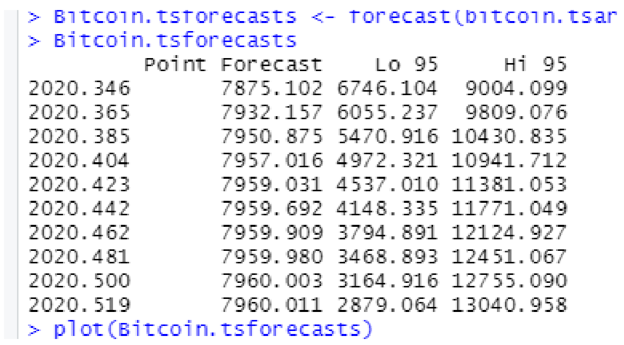


```
> summary(bitcoin.tsarima)

Call:
arima(x = Bitcoin.ts, order = c(1, 1, 0))

Coefficients:
         ar1
      0.3281
s.e.  0.0567

sigma^2 estimated as 331810:  log likelihood = -2153.76,  aic = 4311.52

Training set error measures:
                   ME     RMSE      MAE       MPE    MAPE      MASE        ACF1
Training set 18.56569 574.9927 304.2572 0.6428266 6.02077 0.9790197 0.009186769
```

Figure 5: Arima model

From the above summary figure we can see that the arima function is being called on Bitcoin.ts time series object and the order is (1,1,0) which are our model parameters. The coefficient that is estimated is 0.3281 for the ar1 coefficient with a standard error of 0.0567. So, the estimated model will be

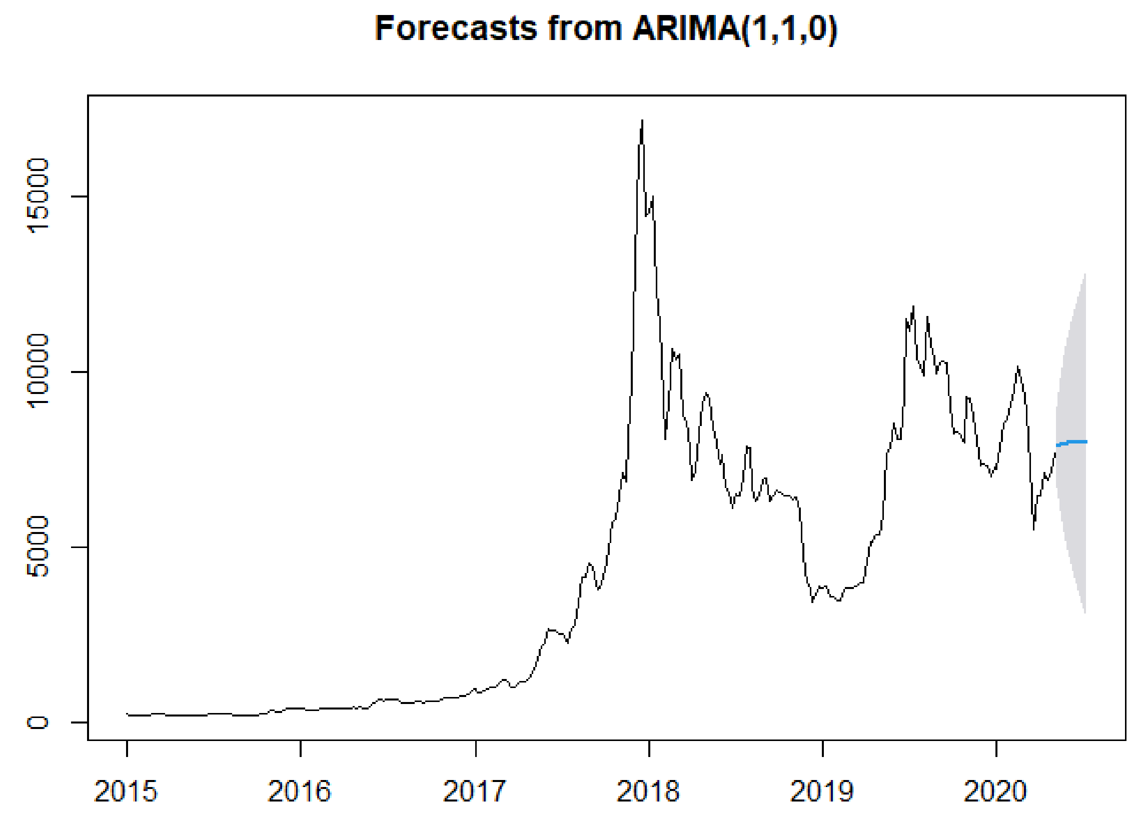
$y_t = \mu + 0.3281y_{t-1} + u_t$ Now the estimated model can be used to forecast the future values of our time series using a forecast function in the forecast package.



```
> Bitcoin.tsforecasts <- forecast(bitcoin.tsar
> Bitcoin.tsforecasts
         Point Forecast    Lo 95     Hi 95
2020.346       7875.102 6746.104  9004.099
2020.365       7932.157 6055.237  9809.076
2020.385       7950.875 5470.916 10430.835
2020.404       7957.016 4972.321 10941.712
2020.423       7959.031 4537.010 11381.053
2020.442       7959.692 4148.335 11771.049
2020.462       7959.909 3794.891 12124.927
2020.481       7959.980 3468.893 12451.067
2020.500       7960.003 3164.916 12755.090
2020.519       7960.011 2879.064 13040.958
> plot(Bitcoin.tsforecasts)
```

Figure 6: ARIMA forecasts

These are the forecasted values by our model for the next 10 weeks (h=10) starting from may 2020 with a confidence level of 95%. Here, we can see the point forecasts as well for the following weeks and the 95% confidence interval forecasts which means there is a 95% chance that the actual week average prices in the future would be in those intervals.

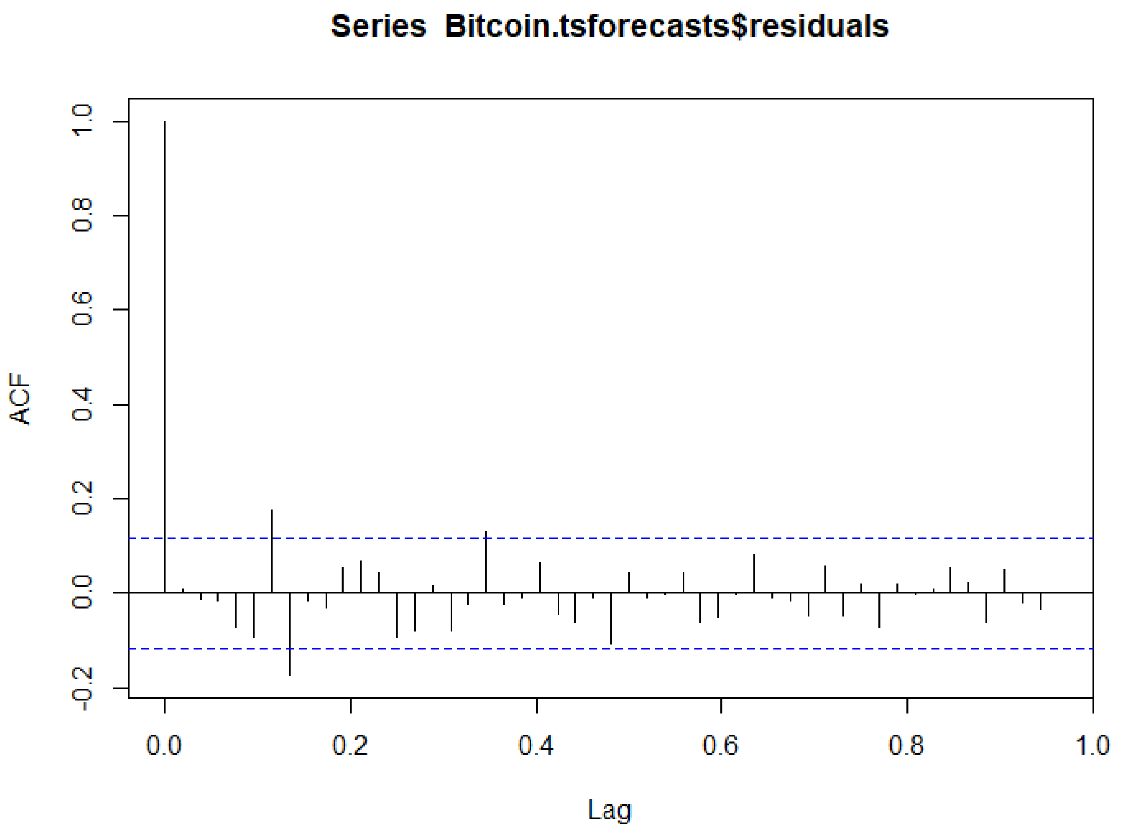


plot 7: ARIMA Forecasts

plot 7 shows the graphical view of the Forecasts (interval and point) made by our model. For the next 3 weeks our model is predicting that the average prices are going to be 7875.102, 7932.157, 7950.875 dollars respectively (these are the point estimates) and the 95% confidence intervals are also given where 6746.104 dollars and 9004.099 dollars are the lower bound and the upper bound for the interval. Similarly, we can conclude for the other weeks as well. Another point that we can notice here is that the confidence interval width is increasing for the later weeks compared to previous ones, as the forecasts standard errors increase for the later periods. So, to attain the same level of confidence the interval width increases.

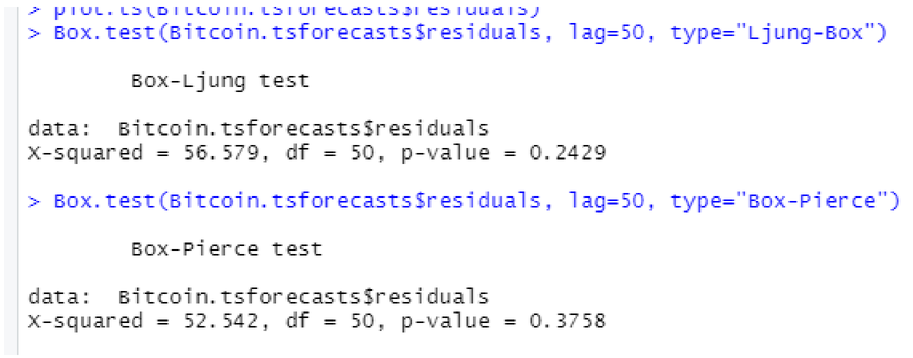
d. In the final step let's see if our predicted model is good? So, we need to check the white noise assumptions for the residuals. The residuals should not be significantly correlated and should follow normal distribution with mean zero and constant variance.

First let's look at the correlogram plot for the residuals of our predicted model.



plot 8: ACF of residuals vs lag

In the above plot we can see that there are few spikes that are significant at 95% confidence level (-0.196 to 0.196). So, it seems that the residuals of the predicted model have some autocorrelation in it. Let's verify if this a high autocorrelation using a box test.



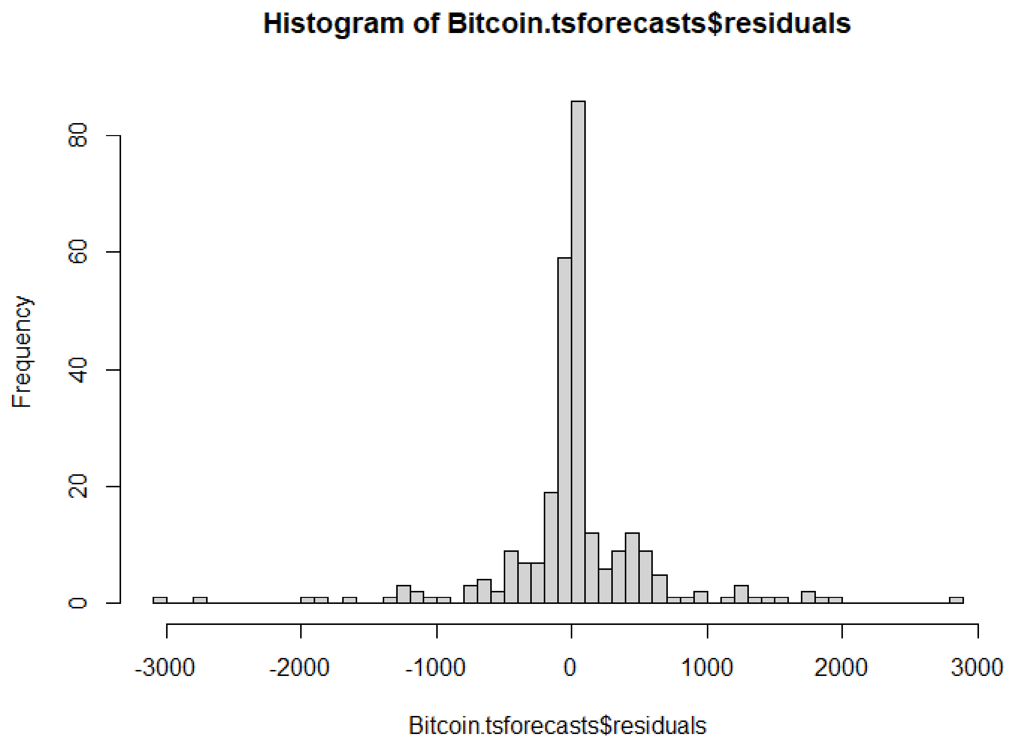
```
> plot.ts(Bitcoin.tsforecasts$residuals)
> Box.test(Bitcoin.tsforecasts$residuals, lag=50, type="Ljung-Box")

        Box-Ljung test

data:  Bitcoin.tsforecasts$residuals
X-squared = 56.579, df = 50, p-value = 0.2429

> Box.test(Bitcoin.tsforecasts$residuals, lag=50, type="Box-Pierce")

        Box-Pierce test

data:  Bitcoin.tsforecasts$residuals
X-squared = 52.542, df = 50, p-value = 0.3758
```

Using both Box-pierce test and Ljung-Box test for a lag value of 50, we can see that the p-values are only somewhat high as expected.

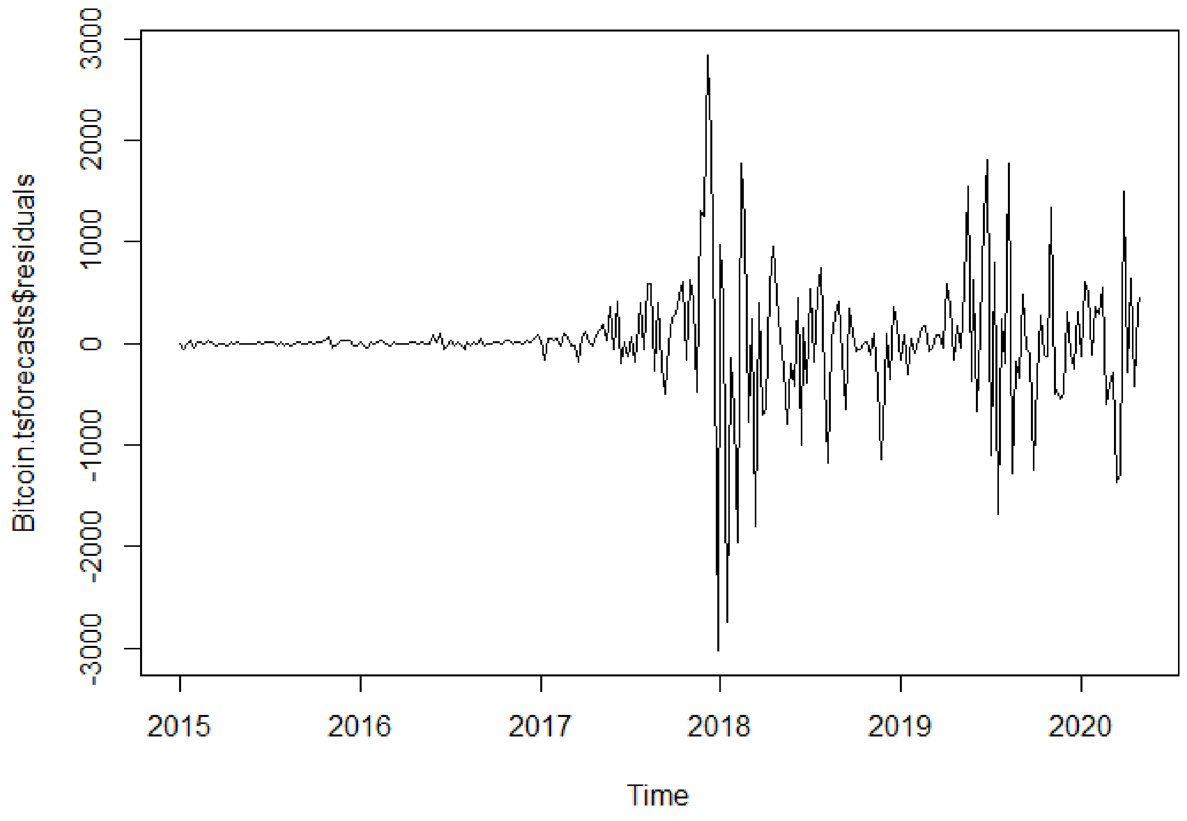
Figure 7: Box-test results using both types

In both of these cases we fail to reject null. Which means that there is no autocorrelation (In these tests the alternative is that there is autocorrelation), but the p-values are not that high and in the plot 8 also we have seen that there could be some autocorrelation. However, the data we are using is the average trading price of Bitcoin which is very similar to financial trading data. For such huge data sets we can use Box-pierce test as a positive definite. Taking the formal tests into consideration we conclude that there is no significant autocorrelation between residuals in the model.

Now, lets look at the distribution of residuals and their time plot to check for the other assumptions.



plot 9: Histogram plot of residuals



plot 10: Residuals vs time

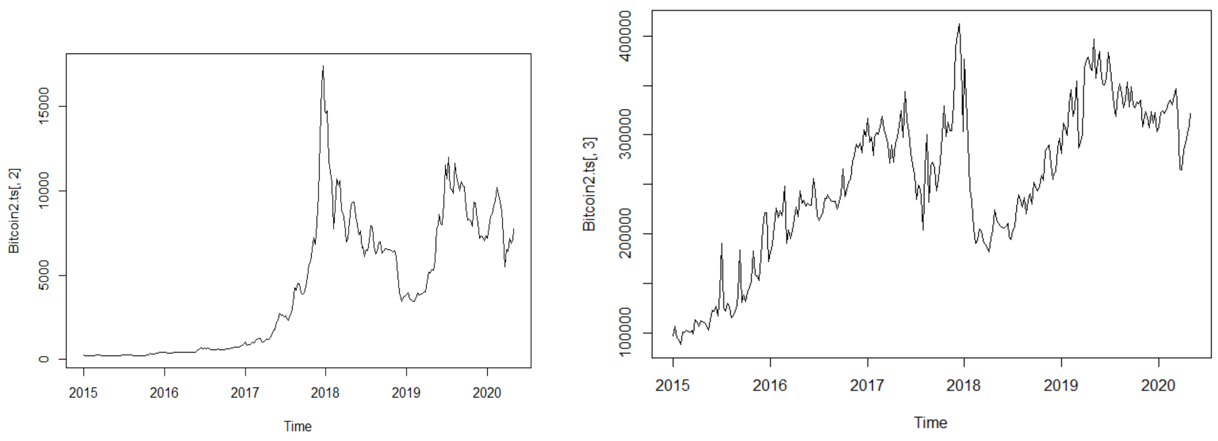
From the plot 9 we can see that the residuals are distributed approximately similar to a normal distribution and in the plot 10 we can see that the mean of the residuals is almost 0 over the period of time and with constant variance. Variance seems to be varying a bit over the period of time, but only during the period of 2017 the variance got affected. If we look at the plot in two different halves, we can observe that both of these parts have almost constant variance individually. Due to some shock (popularity) occurred around 2017 this caused a change in standard deviation compared to previous time period. So, we conclude that the model predicted is reasonably good. But if one requires the autocorrelation of residuals to be strictly very less, then we may need to modify the model.

ARDL/FDL model

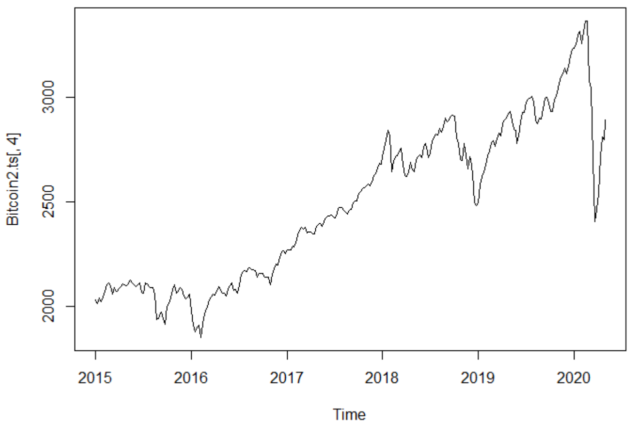
We have seen the ARIMA model and its forecasts, but the ARIMA model only uses single variable (univariate time series). We know that the Bitcoin prices depend on many other factors as well but not just on its previous period prices. Therefore, let’s try to make a distributed lag model using some explanatory variables. We have selected two explanatory variables average number of transactions and the average S&P index. As discussed, these are taken as weekly averages from 2015 to April 2020. From the analysis of ARIMA we have decided to use a lag term of average prices as well, as we have seen that the ARIMA(1,1,0) is also a good model (AR term). So, we have decided to include one lag term of prices initially. Therefore, we will be fitting an AR Distributed Lag model more precisely.

a. Similar to the ARIMA model, the first step will be to check if our data set is stationary or not, and then based on that we can make the necessary changes and select the required model.

So, first let's check for the stationarity. We can check for stationarity in the dataset by checking for all the variables individually. As in ARIMA model we have already checked for the stationarity for price variable- we have taken first difference of prices to make it stationary (one unit root). Similarly, now we will check for both transactions variable and SP variable (average S&P index closing).

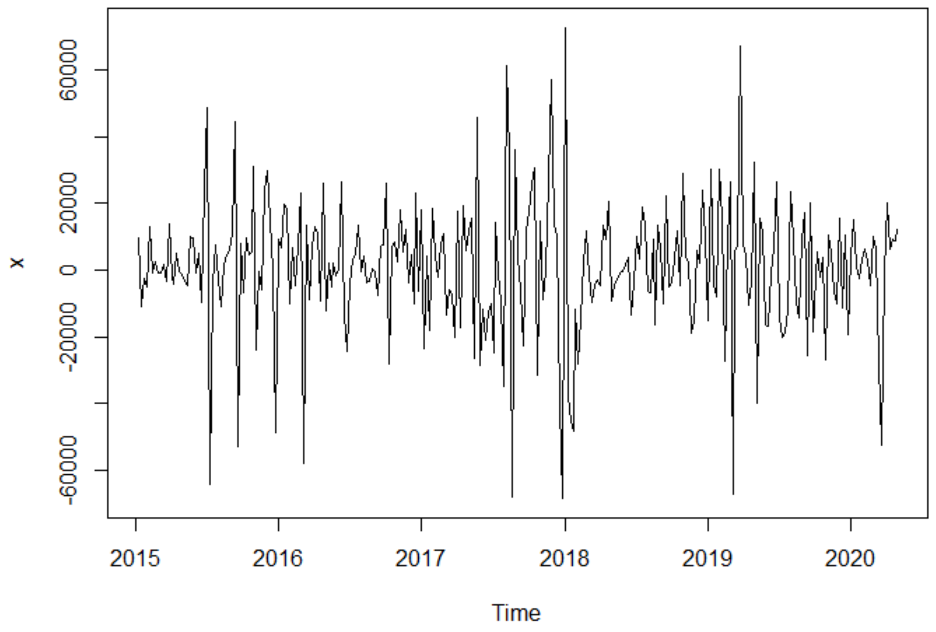


plot 11: price plot and transactions plot



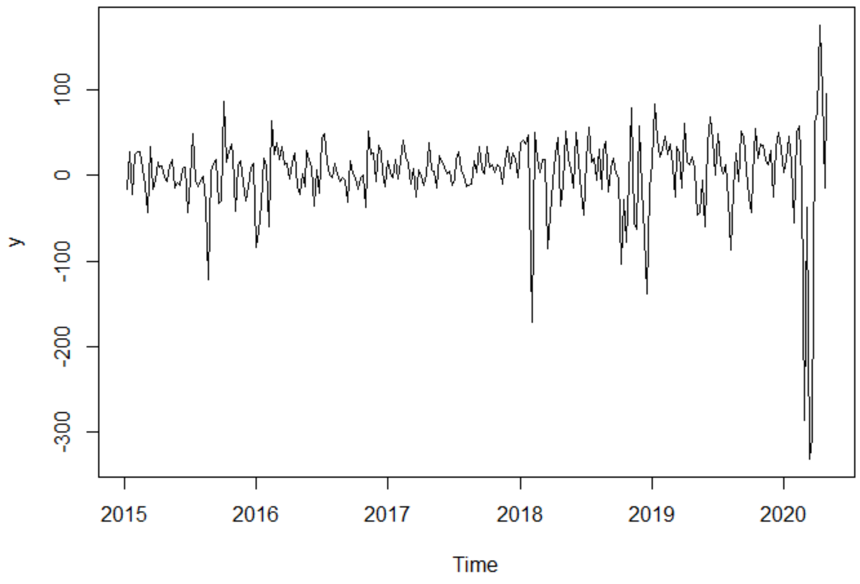
plot 12: S&P time plot

Above plots are the time plots of all the variables that we have chosen for the Distributed lag model. We can see that both transactions and S&P variables does not look stationary, as the mean is not zero or constant as well as the standard deviation is not constant. So, let's difference these variables by once and see if the first differences are stationary.

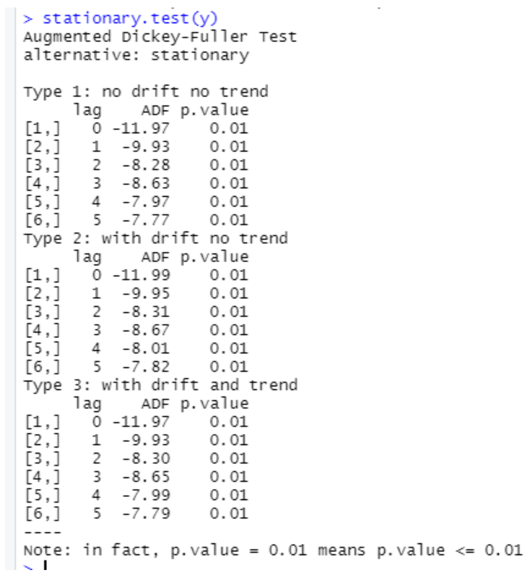


plot 13: first differences of transactions vs time

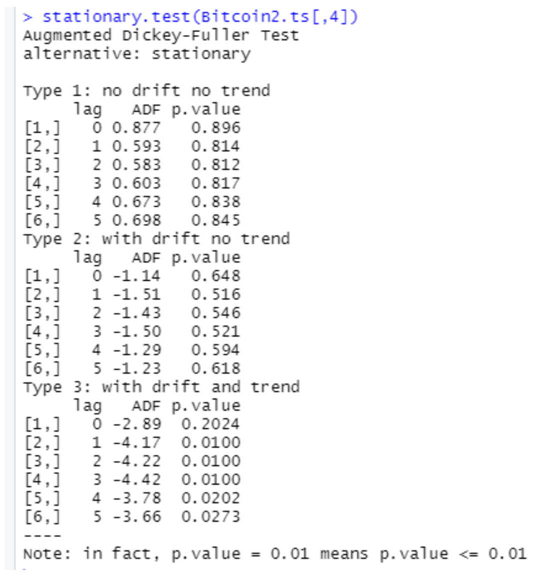
From the plots 13 and 14 we can observe that after taking one difference, the series looks stationary as the mean of the both plots are almost zero (constant) and the standard deviation also seems constant over time. We can run an ADF test similar to the first section to confirm if the converted series is stationary or not. So, let's run an augmented Dicky fuller test on all three of these variables. We have not plotted the differences of price again because it is already given in plot 3 in the first section.



plot 14: first differences of S&P vs time

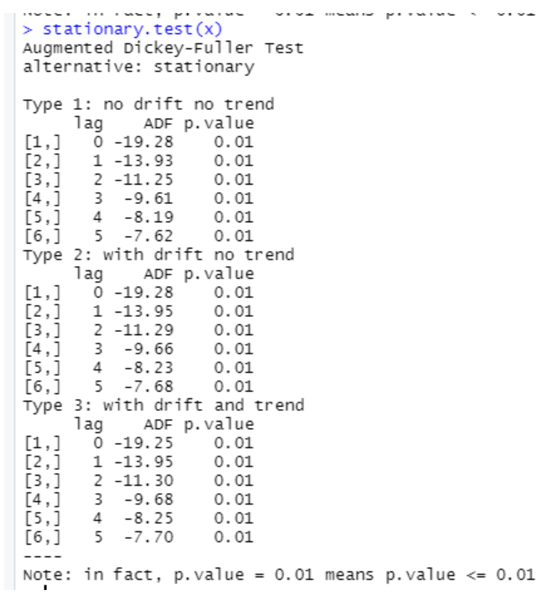


```
> stationary.test(y)
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag    ADF p.value
[1,]   0 -11.97    0.01
[2,]   1  -9.93    0.01
[3,]   2  -8.28    0.01
[4,]   3  -8.63    0.01
[5,]   4  -7.97    0.01
[6,]   5  -7.77    0.01
Type 2: with drift no trend
     lag    ADF p.value
[1,]   0 -11.99    0.01
[2,]   1  -9.95    0.01
[3,]   2  -8.31    0.01
[4,]   3  -8.67    0.01
[5,]   4  -8.01    0.01
[6,]   5  -7.82    0.01
Type 3: with drift and trend
     lag    ADF p.value
[1,]   0 -11.97    0.01
[2,]   1  -9.93    0.01
[3,]   2  -8.30    0.01
[4,]   3  -8.65    0.01
[5,]   4  -7.99    0.01
[6,]   5  -7.79    0.01
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
>
```

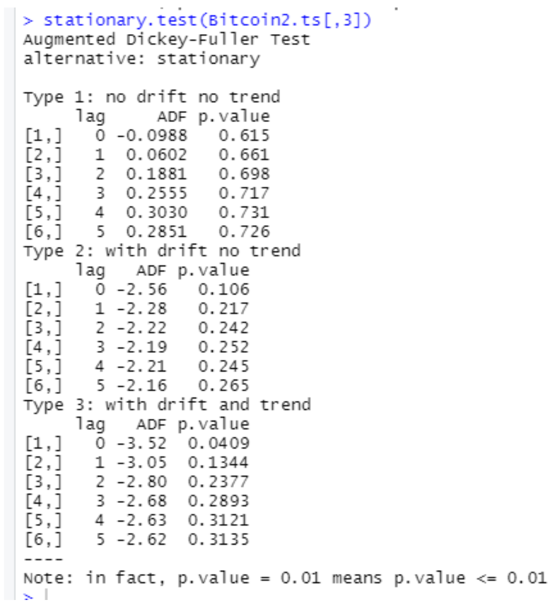


```
> stationary.test(Bitcoin2.ts[,4])
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag   ADF p.value
[1,]   0 0.877   0.896
[2,]   1 0.593   0.814
[3,]   2 0.583   0.812
[4,]   3 0.603   0.817
[5,]   4 0.673   0.838
[6,]   5 0.698   0.845
Type 2: with drift no trend
     lag   ADF p.value
[1,]   0 -1.14   0.648
[2,]   1 -1.51   0.516
[3,]   2 -1.43   0.546
[4,]   3 -1.50   0.521
[5,]   4 -1.29   0.594
[6,]   5 -1.23   0.618
Type 3: with drift and trend
     lag   ADF p.value
[1,]   0 -2.89  0.2024
[2,]   1 -4.17  0.0100
[3,]   2 -4.22  0.0100
[4,]   3 -4.42  0.0100
[5,]   4 -3.78  0.0202
[6,]   5 -3.66  0.0273
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
```

Figure 8: ADF test for S&P and it's first difference



```
> stationary.test(x)
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag    ADF p.value
[1,]   0 -19.28    0.01
[2,]   1 -13.93    0.01
[3,]   2 -11.25    0.01
[4,]   3  -9.61    0.01
[5,]   4  -8.19    0.01
[6,]   5  -7.62    0.01
Type 2: with drift no trend
     lag    ADF p.value
[1,]   0 -19.28    0.01
[2,]   1 -13.95    0.01
[3,]   2 -11.29    0.01
[4,]   3  -9.66    0.01
[5,]   4  -8.23    0.01
[6,]   5  -7.68    0.01
Type 3: with drift and trend
     lag    ADF p.value
[1,]   0 -19.25    0.01
[2,]   1 -13.95    0.01
[3,]   2 -11.30    0.01
[4,]   3  -9.68    0.01
[5,]   4  -8.25    0.01
[6,]   5  -7.70    0.01
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
```



```
> stationary.test(Bitcoin2.ts[,3])
Augmented Dickey-Fuller Test
alternative: stationary

Type 1: no drift no trend
     lag     ADF p.value
[1,]   0 -0.0988   0.615
[2,]   1  0.0602   0.661
[3,]   2  0.1881   0.698
[4,]   3  0.2555   0.717
[5,]   4  0.3030   0.731
[6,]   5  0.2851   0.726
Type 2: with drift no trend
     lag   ADF p.value
[1,]   0 -2.56   0.106
[2,]   1 -2.28   0.217
[3,]   2 -2.22   0.242
[4,]   3 -2.19   0.252
[5,]   4 -2.21   0.245
[6,]   5 -2.16   0.265
Type 3: with drift and trend
     lag   ADF p.value
[1,]   0 -3.52  0.0409
[2,]   1 -3.05  0.1344
[3,]   2 -2.80  0.2377
[4,]   3 -2.68  0.2893
[5,]   4 -2.63  0.3121
[6,]   5 -2.62  0.3135
----
Note: in fact, p.value = 0.01 means p.value <= 0.01
> |
```

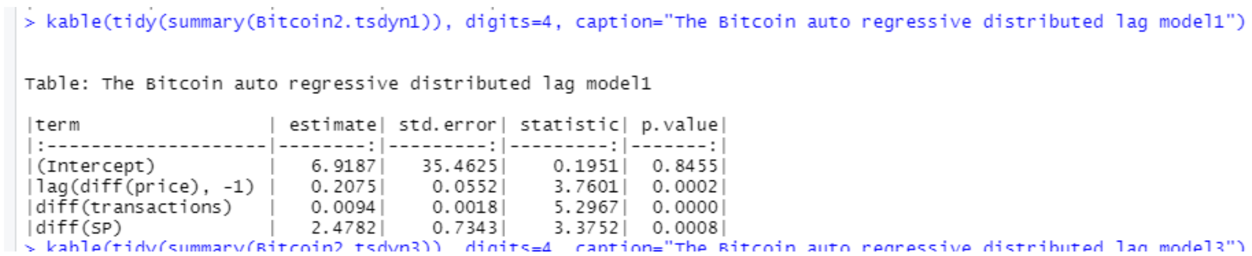
Figure 9: ADF test for transactions and it's first difference

As expected from figures 8,9 we can see that the p-values of the first difference of transactions and S&P are very low for all the three types in the ADF test, which means that the Null hypothesis will be rejected in both cases and the first differences are not non-stationary. In both figures we can see that the actual variables have higher p-values and therefore we cannot reject the Null (data is non stationary). Similar results we have seen for prices in section 1. Therefore, we can conclude that the data we have selected for all three variables (dependent and independent) are non-stationary and we have used differencing to make them stationary. First differences of all the three variables are stationary.

b. Like we have discussed in the beginning of the section, we have included a lag term for the first differences of prices in the model. The remaining explanatory variables are also considered as first differences (non-stationarity problem for original data).

But the main problem is in deciding the number of lags for each explanatory variable. We have estimated few models and compared them with each other to select the best possible choice from those models. So, we have estimated five models. ARDL(1,0,0) ARDL(1,2,2) ARDL(1,3,3) ARDL(1,1,0) ARDL(1,2,0). We have come up with the parameters of the models in a step-by-step process. First we have estimated a normal model ARDL(1,0,0). Here, the three parameters are considered as the lags of estimated variable explanatory variables. From (p,q,r) p indicates the AR part of ARDL and q,r are lags of the two independent variables (first difference of transactions and S&P).

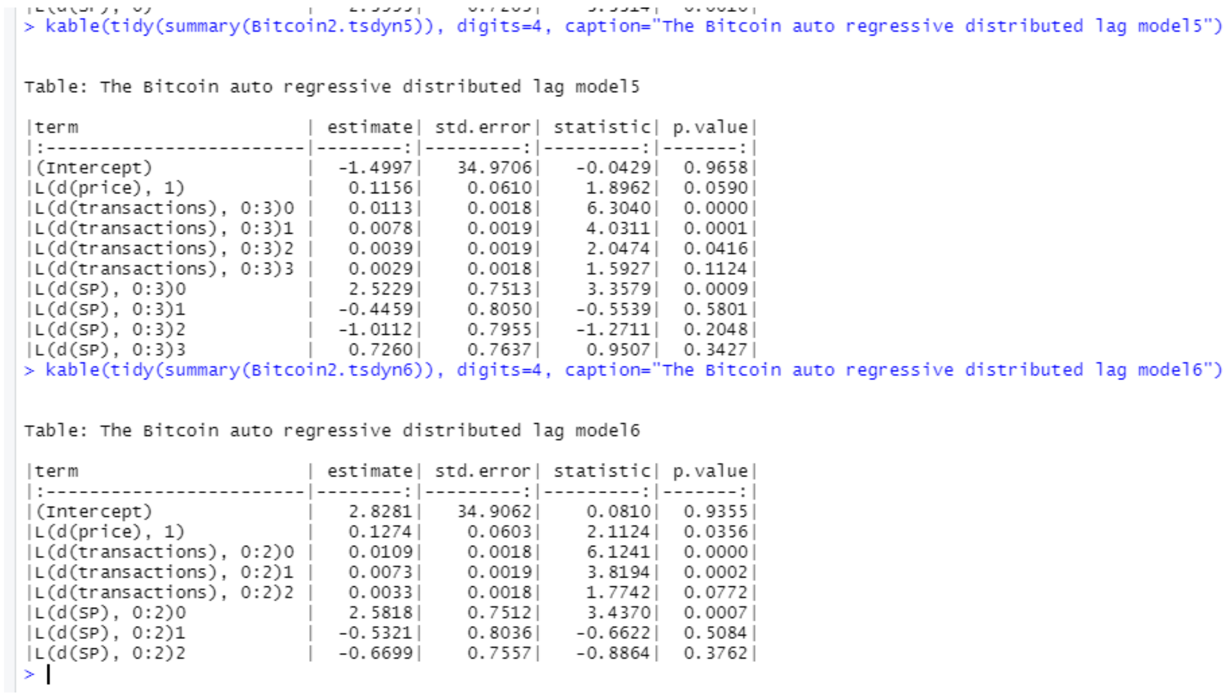
$Bitcoin2.tsdyn1 <- dynlm(d(price) \sim L(d(price),1) + d(transactions) + d(SP), data = Bitcoin2.ts)$ This is the R-command used for ARDL(1,0,0). We have used dynlm function in dynlm package.



```
> kable(tidy(summary(Bitcoin2.tsdyn1)), digits=4, caption="The Bitcoin auto regressive distributed lag model1")


Table: The Bitcoin auto regressive distributed lag model1

|term                  | estimate| std.error| statistic| p.value|
|:---------------------|--------:|---------:|---------:|-------:|
|(Intercept)           |   6.9187|   35.4625|    0.1951|  0.8455|
|lag(diff(price), -1)  |   0.2075|    0.0552|    3.7601|  0.0002|
|diff(transactions)    |   0.0094|    0.0018|    5.2967|  0.0000|
|diff(SP)              |   2.4782|    0.7343|    3.3752|  0.0008|
> kable(tidy(summary(Bitcoin2.tsdyn3)), digits=4, caption="The Bitcoin auto regressive distributed lag model3")
```

Figure 10: ARDL(1,0,0) model

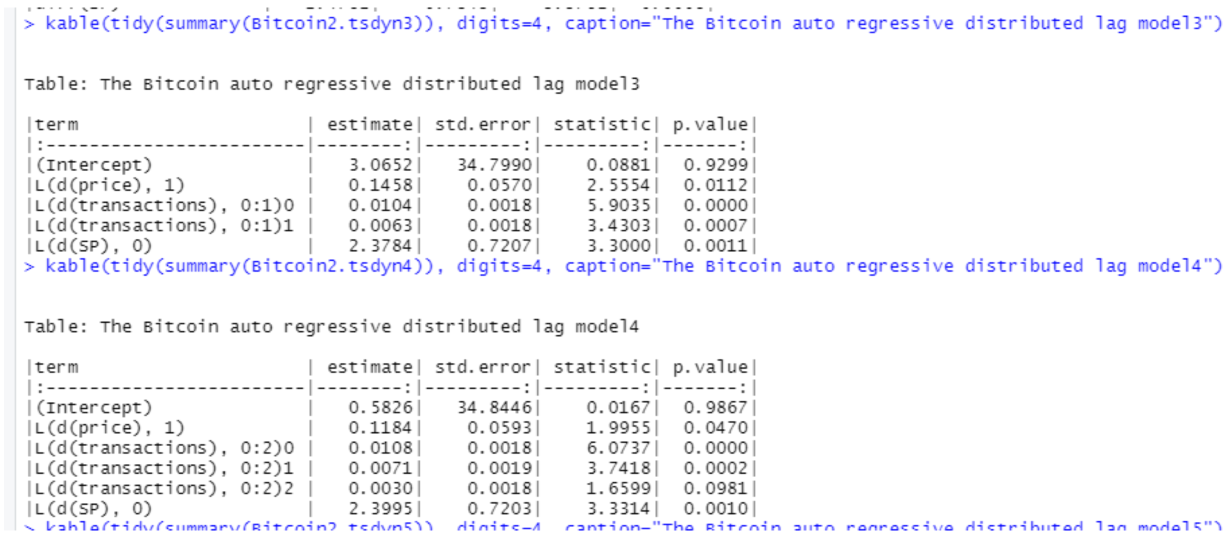
From the above figure we can see that the p-values of all the explanatory variables are low and hence the coefficients are significant. We expect that as the number of transactions increase there could be an increase in the prices, as the number of transactions can be considered as a proxy of demand for the Bitcoin. So, we expect a positive sign for average number of transactions and average price. Similarly, S&P index can be considered as a proxy to the sentiments of people in investing in stocks, as the Bitcoin trading is similar to stock trading, we have considered S&P index to indicate macroeconomic conditions in the economy. As expected, both of the estimates for the variables show a positive sign. So, we can go ahead and add few lags and see how our model will react. (and of course, the lag variable of price shows a positive sign as we know that the higher price difference in previous period can increase the price difference in current period)



```
> kable(tidy(summary(Bitcoin2.tsdyn5)), digits=4, caption="The Bitcoin auto regressive distributed lag model5")


Table: The Bitcoin auto regressive distributed lag model5

|term                       | estimate| std.error| statistic| p.value|
|:--------------------------|--------:|---------:|---------:|-------:|
|(Intercept)                |  -1.4997|   34.9706|   -0.0429|  0.9658|
|L(d(price), 1)             |   0.1156|    0.0610|    1.8962|  0.0590|
|L(d(transactions), 0:3)0   |   0.0113|    0.0018|    6.3040|  0.0000|
|L(d(transactions), 0:3)1   |   0.0078|    0.0019|    4.0311|  0.0001|
|L(d(transactions), 0:3)2   |   0.0039|    0.0019|    2.0474|  0.0416|
|L(d(transactions), 0:3)3   |   0.0029|    0.0018|    1.5927|  0.1124|
|L(d(SP), 0:3)0             |   2.5229|    0.7513|    3.3579|  0.0009|
|L(d(SP), 0:3)1             |  -0.4459|    0.8050|   -0.5539|  0.5801|
|L(d(SP), 0:3)2             |  -1.0112|    0.7955|   -1.2711|  0.2048|
|L(d(SP), 0:3)3             |   0.7260|    0.7637|    0.9507|  0.3427|
> kable(tidy(summary(Bitcoin2.tsdyn6)), digits=4, caption="The Bitcoin auto regressive distributed lag model6")


Table: The Bitcoin auto regressive distributed lag model6

|term                       | estimate| std.error| statistic| p.value|
|:--------------------------|--------:|---------:|---------:|-------:|
|(Intercept)                |   2.8281|   34.9062|    0.0810|  0.9355|
|L(d(price), 1)             |   0.1274|    0.0603|    2.1124|  0.0356|
|L(d(transactions), 0:2)0   |   0.0109|    0.0018|    6.1241|  0.0000|
|L(d(transactions), 0:2)1   |   0.0073|    0.0019|    3.8194|  0.0002|
|L(d(transactions), 0:2)2   |   0.0033|    0.0018|    1.7742|  0.0772|
|L(d(SP), 0:2)0             |   2.5818|    0.7512|    3.4370|  0.0007|
|L(d(SP), 0:2)1             |  -0.5321|    0.8036|   -0.6622|  0.5084|
|L(d(SP), 0:2)2             |  -0.6699|    0.7557|   -0.8864|  0.3762|
> |
```

Figure 11: ARDL(1,2,2) and ARDL(1,3,3) models

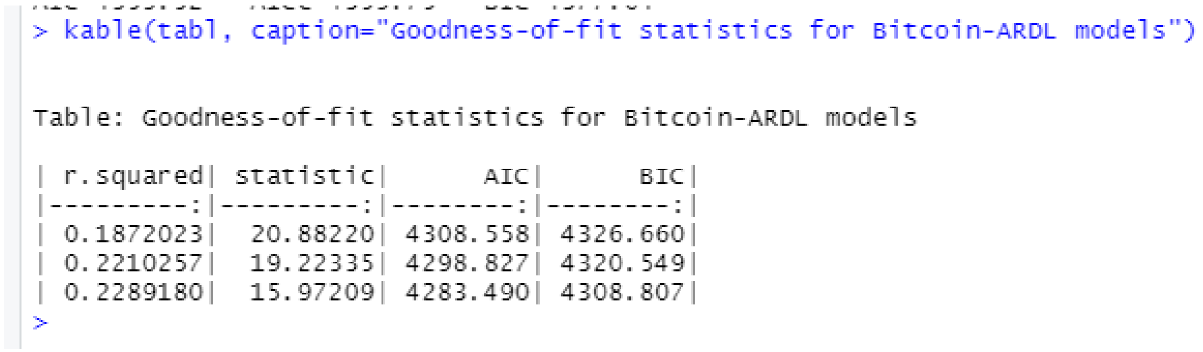
From the above figure we can see that on adding 3 lags for each explanatory variable caused some insignificant lags in the model. All three lags of first difference of S&P are insignificant and also the third lag of first difference of transactions is insignificant. Even some variables are showing negative sign contradictory to what we have discussed. So, we have removed one lag from each of the explanatory variable. However, we still have some insignificant lags in our model. We have decided to remove the lags of first difference of S&P altogether as the p-values for them are very high.



```
> kable(tidy(summary(Bitcoin2.tsdyn3)), digits=4, caption="The Bitcoin auto regressive distributed lag model3")


Table: The Bitcoin auto regressive distributed lag model3

|term                     | estimate| std.error| statistic| p.value|
|:------------------------|--------:|---------:|---------:|-------:|
|(Intercept)              |   3.0652|   34.7990|    0.0881|  0.9299|
|L(d(price), 1)           |   0.1458|    0.0570|    2.5554|  0.0112|
|L(d(transactions), 0:1)0 |   0.0104|    0.0018|    5.9035|  0.0000|
|L(d(transactions), 0:1)1 |   0.0063|    0.0018|    3.4303|  0.0007|
|L(d(SP), 0)              |   2.3784|    0.7207|    3.3000|  0.0011|
> kable(tidy(summary(Bitcoin2.tsdyn4)), digits=4, caption="The Bitcoin auto regressive distributed lag model4")


Table: The Bitcoin auto regressive distributed lag model4

|term                     | estimate| std.error| statistic| p.value|
|:------------------------|--------:|---------:|---------:|-------:|
|(Intercept)              |   0.5826|   34.8446|    0.0167|  0.9867|
|L(d(price), 1)           |   0.1184|    0.0593|    1.9955|  0.0470|
|L(d(transactions), 0:2)0 |   0.0108|    0.0018|    6.0737|  0.0000|
|L(d(transactions), 0:2)1 |   0.0071|    0.0019|    3.7418|  0.0002|
|L(d(transactions), 0:2)2 |   0.0030|    0.0018|    1.6599|  0.0981|
|L(d(SP), 0)              |   2.3995|    0.7203|    3.3314|  0.0010|
```

Figure 12: ARDL(1,1,0) and ARDL(1,2,0) models

On removing the lags for first differences of S&P variable, we get an ARDL model of order 1,2,0. As, you can see in the above figure the ARDL(1,2,0) model has almost all the coefficients significant except the lag2 of first differences of transactions. But the lag2 will be significant at the 90% confidence interval. So, we cannot directly reject the model altogether. We have estimated another model by removing the 2nd lag of first difference of transactions which would be ARDL(1,1,0). We need to select our model among these two estimated models and **also our trivial model ARDL(1,0,0)**.



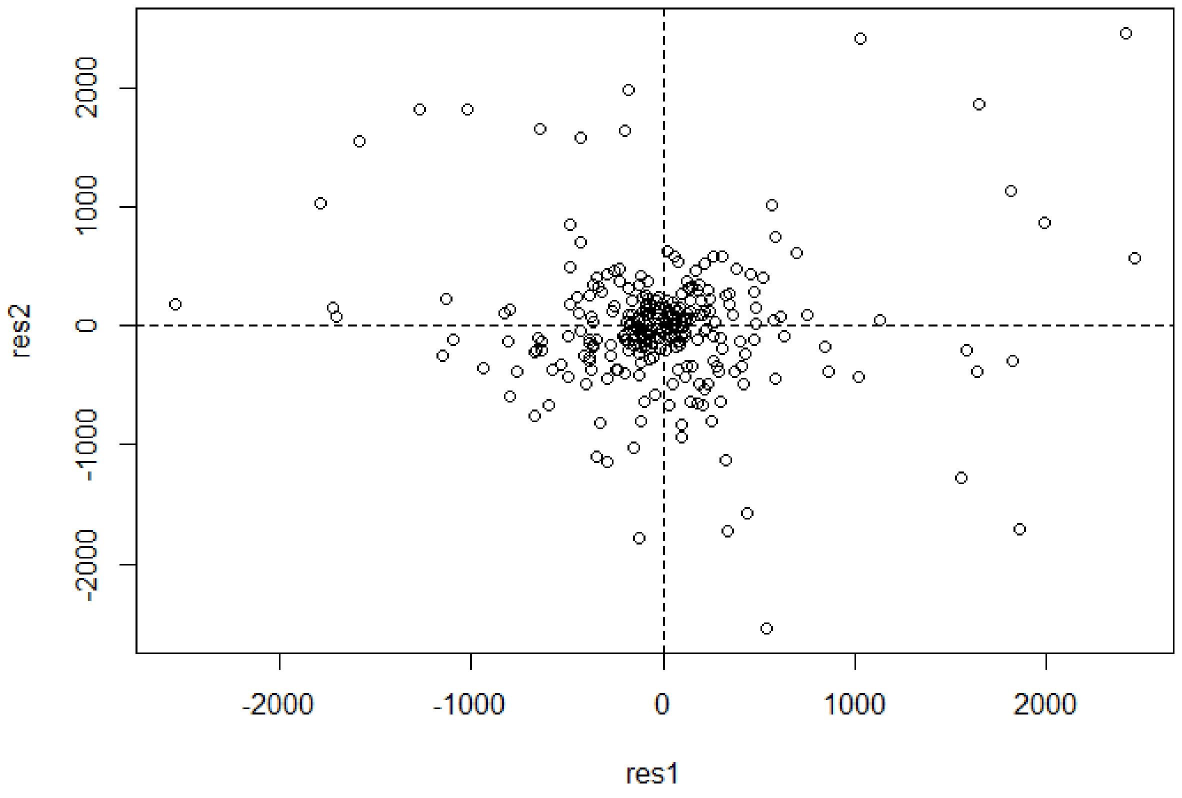
```
> kable(tabl, caption="Goodness-of-fit statistics for Bitcoin-ARDL models")


Table: Goodness-of-fit statistics for Bitcoin-ARDL models

| r.squared| statistic|      AIC|      BIC|
|---------:|---------:|--------:|--------:|
| 0.1872023|  20.88220| 4308.558| 4326.660|
| 0.2210257|  19.22335| 4298.827| 4320.549|
| 0.2289180|  15.97209| 4283.490| 4308.807|
>
```

Figure 13: comparison between models

Figure 13 shows the Goodness of fit for all the three models ARDL(1,0,0), ARDL(1,1,0), ARDL(1,2,0). In the figure we can see that both the information criterion AIC, BIC scores are lowest for ARDL(1,2,0) and also the r-squared value is highest for it. All three models have the expected signs for the estimated coefficients and also the coefficients are significant. So, we have selected ARDL(1,2,0) as our model.

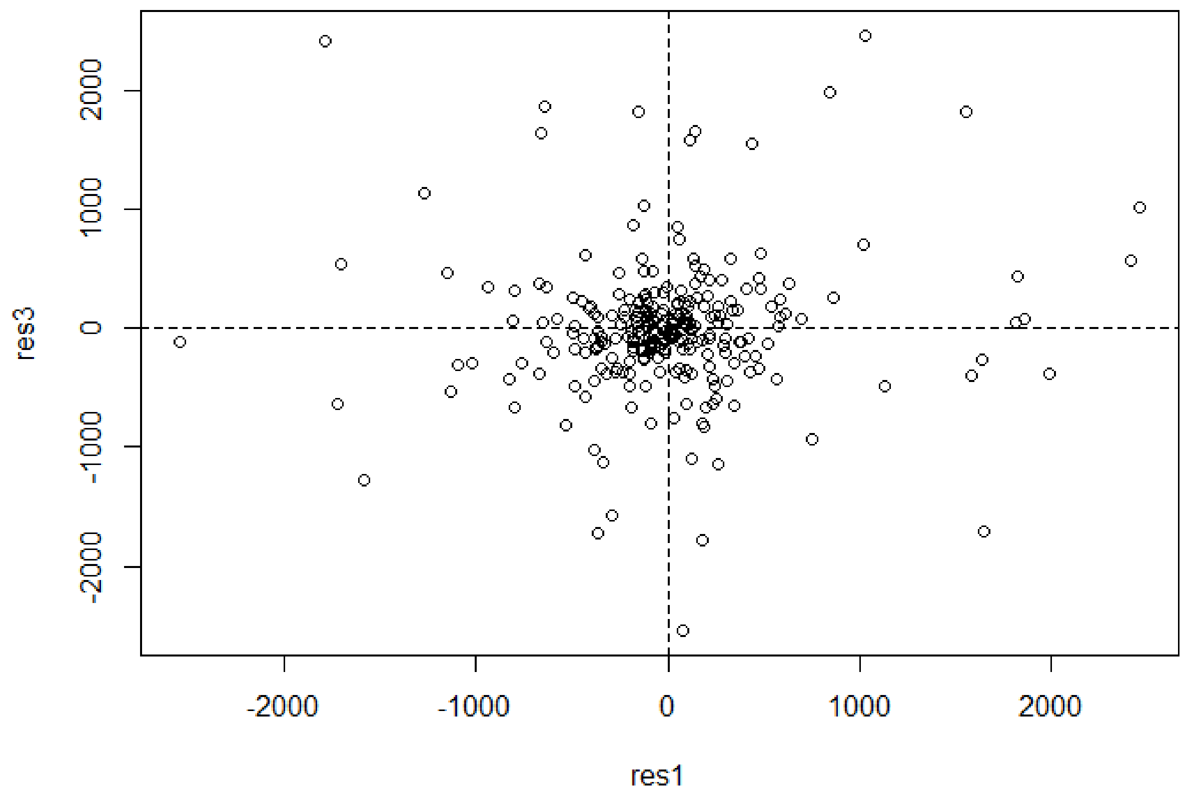
c. To see how good our model is, we can check for any serial correlation among the residuals for our estimated model. Similar to Heteroskedasticity in linear model, Autocorrelation in the residuals do not create biased estimates but the standard errors will ne be no longer reliable, this is the reason we were able to check for the signs of coefficients before running LM test.

We can check for serial correlation/Auto correlation in many ways, let's check for the same using three different approaches. Scatter plot, ACF plot (correlogram), LM test.



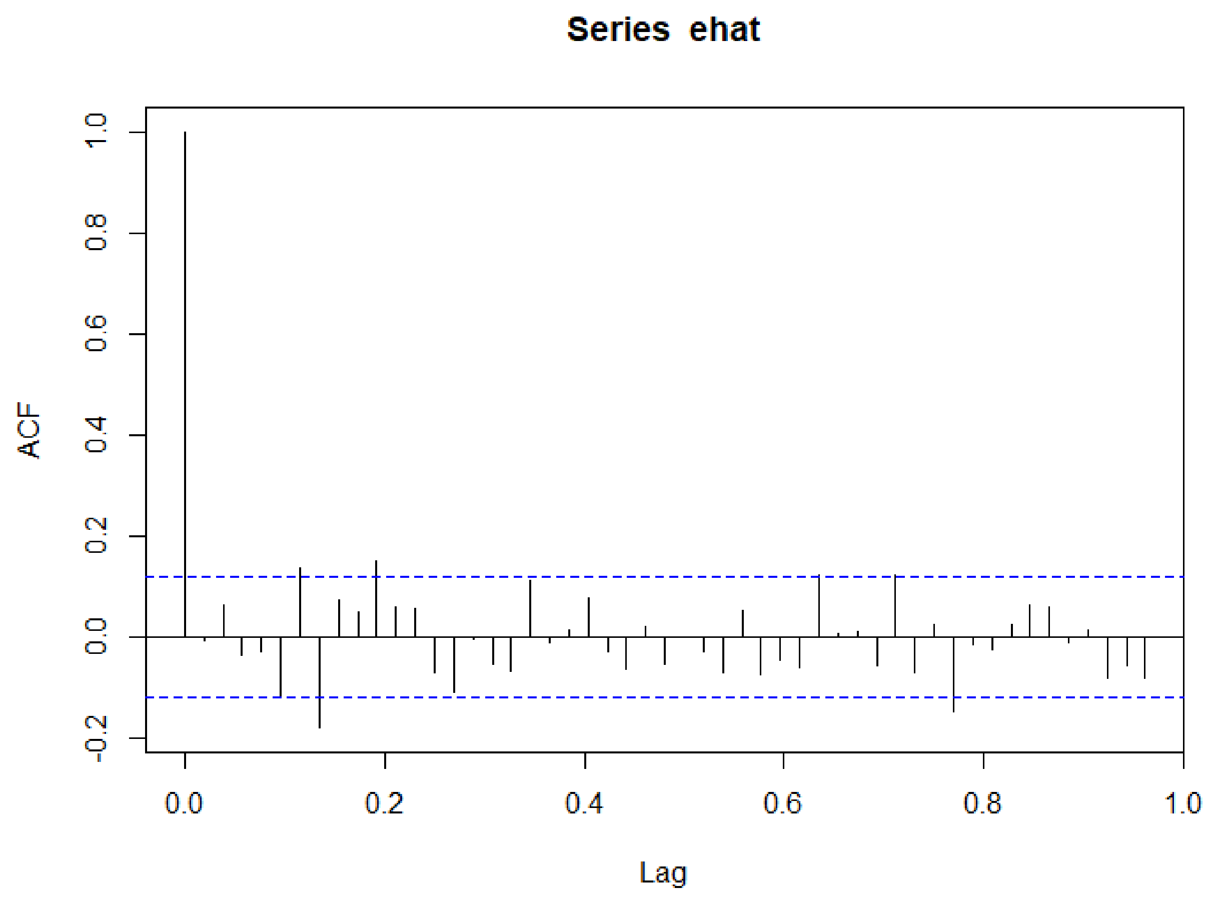
plot 15: scatter plot of lag1 of residuals vs residuals

Plot 15 is the scatter plot between the residuals in x and residuals of lag1 in y for the selected model ARDL(1,2,0). We can see that the values are mostly scattered randomly across the quadrants with few outliers in first and fourth quadrants. Let's look at the scatter plot of lag2 residuals and residuals.



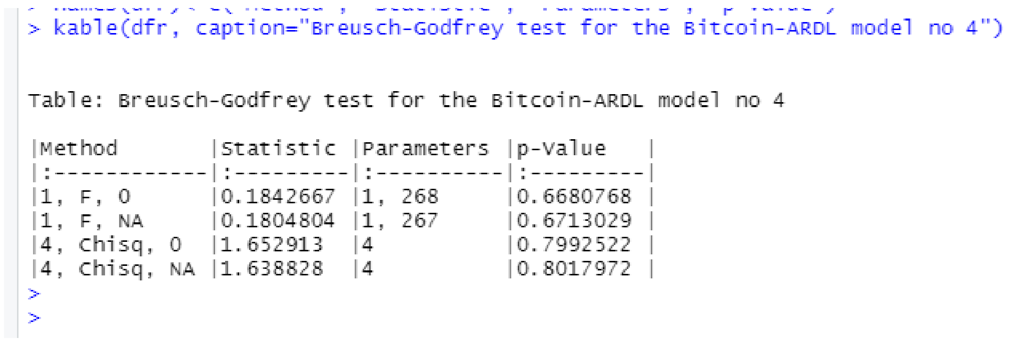
plot 16: Scatter plot between lag2 of residuals and residuals

From the above plot we can see that the values are now even randomly dispersed across the four quadrants than before. We have taken the dotted lines on $x = 0; y = 0$ because we are assuming that the mean value of residuals should be zero, based on that we can see if there is any serial correlation (if there are very high number of values in 1st and 3rd quadrants or in 2nd and 4th quadrants). From both the scatter plots we can see that the serial correlation could be lower. Before jumping into conclusion let's look at the correlogram and the formal test (LM) as well.



plot 17: ACF plot of residuals of ARDL(1,2,0)

Plot 17 is the correlogram or the ACF plot for residuals of the selected model for 50 lags. We can see that there are few significant spikes in the beginning part, but all these spikes are very close the significance boundary and just passed the level. This indicates low serial correlation among residuals for the model.

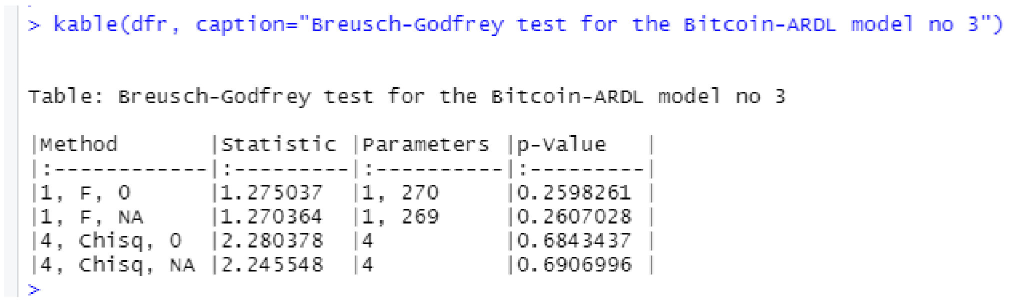


```
> kable(dfr, caption="Breusch-Godfrey test for the Bitcoin-ARDL model no 4")


Table: Breusch-Godfrey test for the Bitcoin-ARDL model no 4

|Method       |Statistic |Parameters |p-Value   |
|:------------|:---------|:----------|:---------|
|1, F, 0      |0.1842667 |1, 268     |0.6680768 |
|1, F, NA     |0.1804804 |1, 267     |0.6713029 |
|4, Chisq, 0  |1.652913  |4          |0.7992522 |
|4, Chisq, NA |1.638828  |4          |0.8017972 |
>
>
```

Figure 14: BG test for ARDL(1,2,0)

Now, lets see the formal test to check for the autocorrelation of residuals in our selected model. Lagrange Multiplier test also known as LM test. For any model we first assume that the residuals of our model follow a structure like $e_t = \rho e_{t-1} + v_t$ which is nothing but an AR(1) model for residuals. So, based on the assumption we will re-write our main model and test for a Null hypothesis $\rho = 0$. All these calculations are done directly by using the Breusch-Godfrey test. From the Figure 14 we can see that the p-values for lags1 using both methods (filling the first value with 0 or NA) are 0.67 and for four lags we even get a higher p-value of 0.8 using both the methods. Hence, we reject the Null in all the cases stating that there is no significant serial correlation among the residuals.

As, we have seen that the ARDL(1,1,0) model also had the AIC, BIC scores very near to our selected model and the R-squared values are also closer. Let us see the BG test for ARDL(1,1,0) as well.



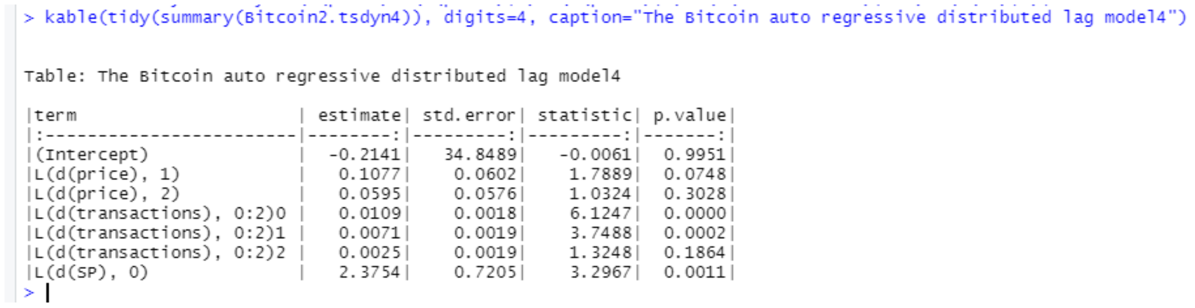
```
> kable(dfr, caption="Breusch-Godfrey test for the Bitcoin-ARDL model no 3")


Table: Breusch-Godfrey test for the Bitcoin-ARDL model no 3

|Method       |Statistic |Parameters |p-Value   |
|:------------|:---------|:----------|:---------|
|1, F, 0      |1.275037  |1, 270     |0.2598261 |
|1, F, NA     |1.270364  |1, 269     |0.2607028 |
|4, Chisq, 0  |2.280378  |4          |0.6843437 |
|4, Chisq, NA |2.245548  |4          |0.6906996 |
>
```

Figure 15: BG test for ARDL(1,1,0)

In the above figure we can see that this model also does not have serial autocorrelation among residuals but the p-values for lag1 are highly different from our selected model. Including the additional lag of first difference of transactions is the only difference among both these variables. Inclusion of lag has reduced the serial correlation between residuals significantly, but we also need to remember that our model ARDL(1,2,0) had one lag coefficient which is insignificant at 95% confidence level but significant at 90% confidence level. We went ahead with this model as the coefficient is significant at 90% confidence level.

d. As we have seen that there is no significant autocorrelation among the residuals of the selected model, we can go ahead use the same for analysis and forecasting the future values (Bitcoin prices).

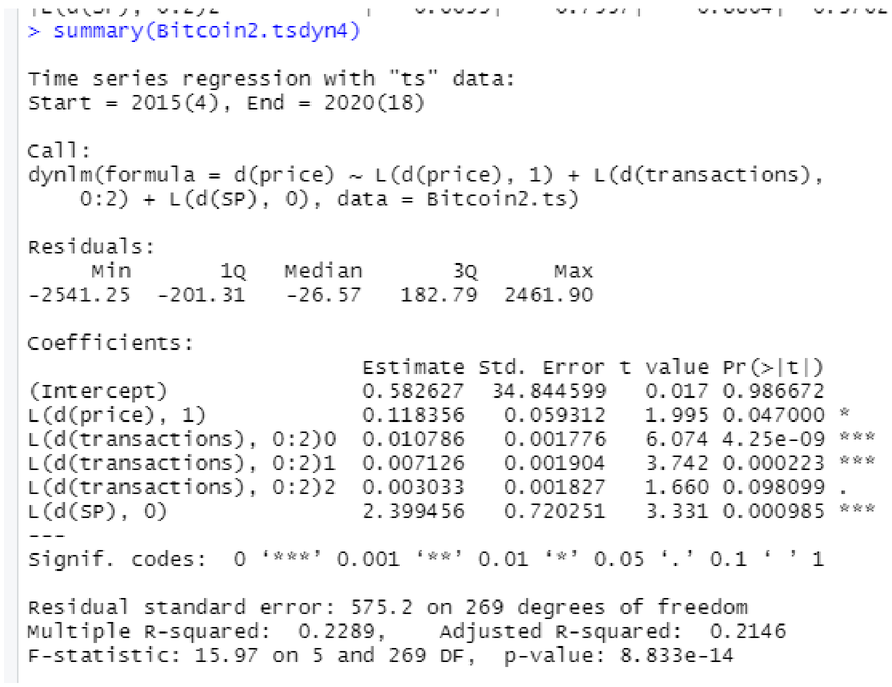
But let us look what could be the effect on our selected model if we include another lag of dependent variable.



```
> kable(tidy(summary(Bitcoin2.tsdyn4)), digits=4, caption="The Bitcoin auto regressive distributed lag model4")


Table: The Bitcoin auto regressive distributed lag model4

|term                      | estimate| std.error| statistic| p.value|
|:-------------------------|--------:|---------:|---------:|-------:|
|(Intercept)               |  -0.2141|   34.8489|   -0.0061|  0.9951|
|L(d(price), 1)            |   0.1077|    0.0602|    1.7889|  0.0748|
|L(d(price), 2)            |   0.0595|    0.0576|    1.0324|  0.3028|
|L(d(transactions), 0:2)0  |   0.0109|    0.0018|    6.1247|  0.0000|
|L(d(transactions), 0:2)1  |   0.0071|    0.0019|    3.7488|  0.0002|
|L(d(transactions), 0:2)2  |   0.0025|    0.0019|    1.3248|  0.1864|
|L(d(SP), 0)               |   2.3754|    0.7205|    3.2967|  0.0011|
> |
```

Figure 16: ARDL(2,2,0)

As we have included another AR lag term therefore our new model becomes ARDL(2,2,0) but from the above figure we can see that the coefficient of 2nd lag term of first difference of price and first difference of transactions have become insignificant. So, we will stick to our previous selected model ARDL(1,2,0) for forecasting. Before forecasting let's look at the results and draw some conclusions for the model.



```
> summary(Bitcoin2.tsdyn4)

Time series regression with "ts" data:
Start = 2015(4), End = 2020(18)

Call:
dynlm(formula = d(price) ~ L(d(price), 1) + L(d(transactions),
    0:2) + L(d(SP), 0), data = Bitcoin2.ts)

Residuals:
     Min       1Q   Median       3Q      Max
-2541.25  -201.31   -26.57   182.79  2461.90

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               0.582627  34.844599   0.017 0.986672
L(d(price), 1)            0.118356   0.059312   1.995 0.047000 *
L(d(transactions), 0:2)0  0.010786   0.001776   6.074 4.25e-09 ***
L(d(transactions), 0:2)1  0.007126   0.001904   3.742 0.000223 ***
L(d(transactions), 0:2)2  0.003033   0.001827   1.660 0.098099 .
L(d(SP), 0)               2.399456   0.720251   3.331 0.000985 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 575.2 on 269 degrees of freedom
Multiple R-squared:  0.2289,    Adjusted R-squared:  0.2146
F-statistic: 15.97 on 5 and 269 DF,  p-value: 8.833e-14
```

Figure 17: Summary of ARDL(1,2,0)

From the estimated model we can see that for every one USD increase in the difference of average prices of last week and last to last week ceteris paribus there will be an increase of 0.118 USD in the difference of average prices for the current week and last week. Based on the increase of the price of last week we can get the increase in price of current price. Suppose, if the previous period increase is solely due to the increase in price of last week, then the price of current week should increase by 1+0.118 USD (such that the difference is maintained) based on our model.

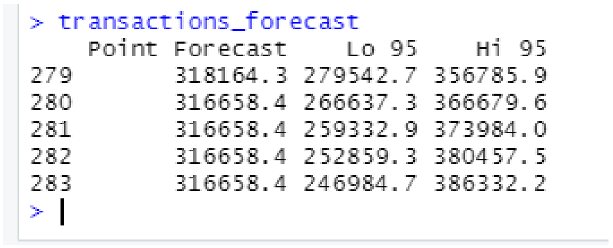
Similarly, we can say that for an increase of 1 average transaction in the difference of average transactions of current week and last week causes an increase of 0.01 USD in the difference of average prices for the current week and last week. For an increase of 1 USD in the difference of average S&P index closing prices of current and last week causes an increase of 2.4 USD in the difference of average prices for the current week and the last week.

We can also get the interim multiplier based on the sustained increase in the x-variable over the period of time. Suppose that the increase in the difference of average transactions compared to previous week are 1 for 2 periods current and last week then the increase in the difference of prices of current week compared to last week will be $\beta_0 + \beta_1$ or in our case $0.011 + 0.007 = 0.018$ USD. Total Multiplier from average number of transactions will be 0.021 USD. So, using the model we can interpret the affects of the chosen explanatory variables but the process will be complex as we worked with first differences for all the variables due to the problem of non-Stationarity.

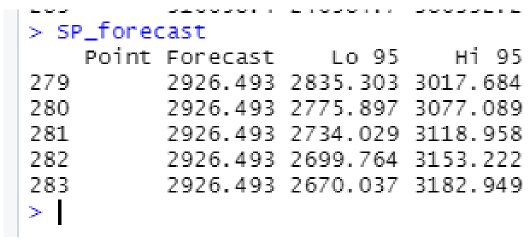
We can see that the number of transactions and S&P index price are good explanatory variables for predicting Bitcoin prices. Number of transactions have a significant effect on more future time periods compared to S&P index prices.

e. For the final part let's predict some Future values of the Bitcoin prices using the ARDL model. We already have the model with us. So, the important part we need will be the future values of the explanatory variables.

We can use the auto.arima function just like how we used in the first section to estimate the future values of the explanatory variables. We can take these values as proxies because the main variable of interest for us is the Average prices of Bitcoin or at least its first differences.



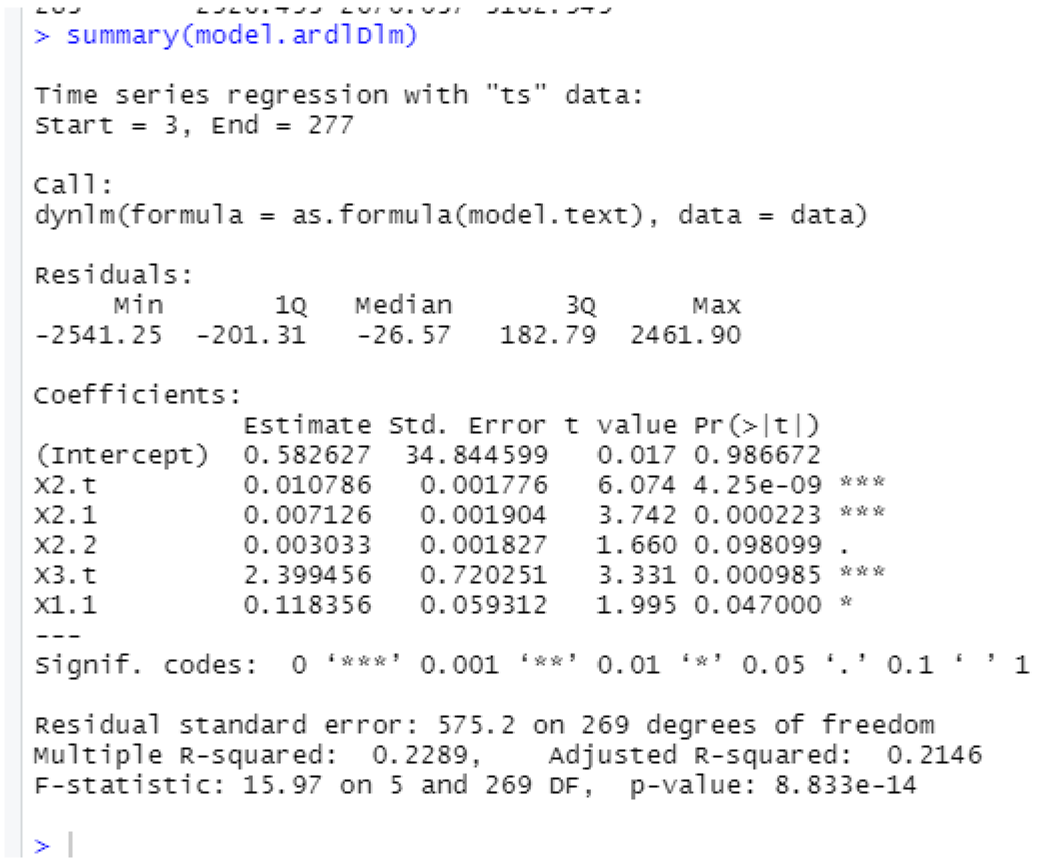
```
> transactions_forecast
    Point Forecast    Lo 95    Hi 95
279       318164.3 279542.7 356785.9
280       316658.4 266637.3 366679.6
281       316658.4 259332.9 373984.0
282       316658.4 252859.3 380457.5
283       316658.4 246984.7 386332.2
> |
```



```
> SP_forecast
    Point Forecast    Lo 95    Hi 95
279       2926.493 2835.303 3017.684
280       2926.493 2775.897 3077.089
281       2926.493 2734.029 3118.958
282       2926.493 2699.764 3153.222
283       2926.493 2670.037 3182.949
> |
```

Figure 18: transactions and S&P forecasts using ARIMA

These are the next five-week forecasts for the transactions and S&P using ARIMA model. Using these we can calculate the first differences of the future forecasts of transactions and S&P index. In order to forecast we can use an R-package called dLagM, this package contains a function similar to forecast which is in forecast package which we used for ARIMA forecasting. But this forecast function can also take the future values of the explanatory variables and based on that it forecasts the future values of the dependent variable using a Monte Carlo simulation. For reference click here. But to use this function we cannot directly use the dynlm object which we have used for the analysis in the previous steps. So, we need to estimate the same ARDL(1,2,0) model using ardlDlm function given in the dLagM package.

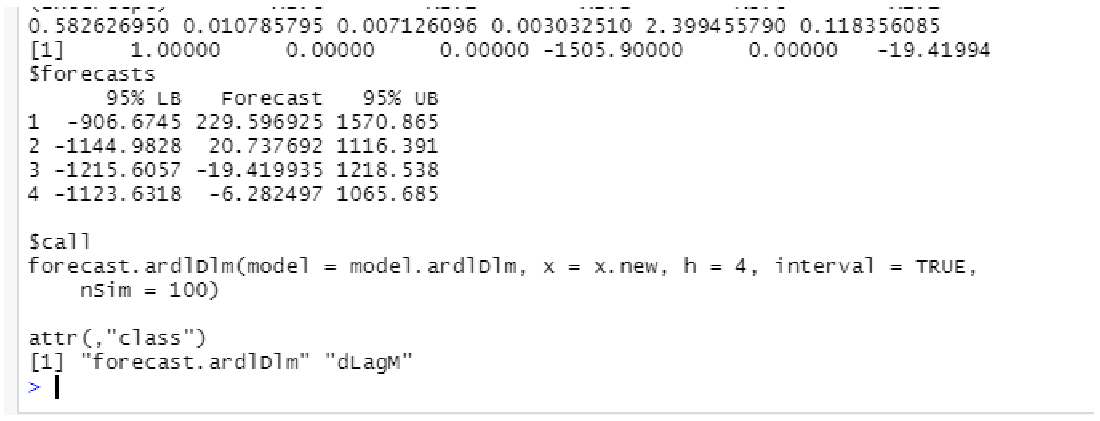


```
> summary(model.ardlDlm)

Time series regression with "ts" data:
Start = 3, End = 277

Call:
dynlm(formula = as.formula(model.text), data = data)

Residuals:
     Min       1Q   Median       3Q      Max 
-2541.25  -201.31   -26.57   182.79  2461.90 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.582627  34.844599   0.017 0.986672    
X2.t         0.010786   0.001776   6.074 4.25e-09 ***
X2.1         0.007126   0.001904   3.742 0.000223 ***
X2.2         0.003033   0.001827   1.660 0.098099 .  
X3.t         2.399456   0.720251   3.331 0.000985 ***
X1.1         0.118356   0.059312   1.995 0.047000 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 575.2 on 269 degrees of freedom
Multiple R-squared:  0.2289,	Adjusted R-squared:  0.2146 
F-statistic: 15.97 on 5 and 269 DF,  p-value: 8.833e-14

> 
```

Figure 19: ARDL(1,2,0) using ardlDlm

Figure 19 shows the summary of the ARDL(1,2,0) model estimated using the ardlDlm function, we can see that all the estimates, standard errors and p-values are exactly the same which we got using dynlm function. Here, X1 is the first differences of prices and X2 is the first differences of transactions and X3 is the first differences of S&P index prices (obviously all averages). This is the reason why we have calculated the first differences of transactions and S&P prices, so that now we can pass these values as a matrix for the dLagM::forecast function. For reference click here.

So, now we are ready to forecast the Average prices of Bitcoin using our ARDL model, but more precisely the first differences of Average prices. However, once if we get the differences then one by one we can get the actual price estimates using the previous actual prices.



```
0.582626950 0.010785795 0.007126096 0.003032510 2.399455790 0.118356085
[1]     1.00000     0.00000     0.00000 -1505.90000     0.00000   -19.41994
$forecasts
      95% LB   Forecast   95% UB
1  -906.6745 229.596925 1570.865
2 -1144.9828  20.737692 1116.391
3 -1215.6057 -19.419935 1218.538
4 -1123.6318  -6.282497 1065.685

$call
forecast.ardlDlm(model = model.ardlDlm, x = x.new, h = 4, interval = TRUE,
    nSim = 100)

attr(,"class")
[1] "forecast.ardlDlm" "dLagM"
> 
```

Figure 20: price difference Forecasts for the next 4 weeks

These are the Forecasts of our model for the next 4 weeks using the Future values predicted by ARIMA model (the process could give more accurate results with the actual future values of explanatory variables) and 100 number of Monte Carlo simulations.

Table 1: Forecasts

	Difference Forecasted	Previous price	Forecasted price
1st week	229.5969	7751.753	7981.35
2nd week	20.7377	7981.35	8002.088
3rd week	-19.4199	8002.088	7982.668
4th week	-6.2825	7982.668	7976.386

Above table summarizes the values forecasted and based on that, one step ahead we have calculated the Forecasted prices for the current period. The Forecast function in the dLagM also calculates the forecasts one step ahead using the explanatory future values provided. So, the forecast function predicted that for the 1st week the difference of prices will be 229.60 USD compared to the previous week price. Similarly for the 2nd week the price difference is predicted to be 20.74 USD, for 3rd week it is -19.42 (price decreased compared to last week). So, using the last Bitcoin average price in our data which is 7751.75 we can calculate the forecast value for the 1st week which is 7981.35 USD. Now using this value as the previous price value, we can calculate the 2nd week Forecasted price which will be 8002.09 USD. For the 3rd week and 4th week we get a value of 7982.67 and 7976.39 USD respectively.

we have also forecasted the 95% confidence interval for the first differences of Average Bitcoin prices at the same time. Based on that similarly we can get the 95% confidence bounds for forecasted prices.

Thank You

Arja Sudheer, 2018B3AA0878H

Repaka Saketh, 2018B3A80996H

Data set:

https://www.kaggle.com/wojtekbonicki/bitcoin-data